DRAFT V2.1

From

Math, Numerics, & Programming

(for Mechanical Engineers)

Masayuki Yano
James Douglass Penn
George Konidaris
Anthony T Patera

August 2013

Contents

Unit I

(Numerical) Calculus. Elementary Programming Concepts.

Chapter 1

Motivation

1.1 A Mobile Robot

Robot self-localization, or the ability of a robot to figure out where it is within its environment, is arguably the most fundamental skill for a mobile robot, such as the one shown in Figure 1.1. We can divide the robot self-localization problem into two parts: global position estimation and local position tracking. Global position estimation is the robot's ability to determine its initial position and orientation (collectively, pose) within a known map of its environment. Local position tracking is then the ability of the robot to track changes in its pose over time. In this assignment, we will consider two basic approaches to global position estimation and local position tracking.

1.2 Global Position Estimation: Infra-red Range-Finding

Many systems exist today for robot global position estimation. Perhaps the most familiar example is the Global Positioning System (GPS), a network of 24 satellites that can give an absolute position estimate accurate to within several meters. For smaller scale position estimation, high-end solutions such as robotic vision and laser range-finding can provide millimeter accuracy distance measurements, which can then be matched with map data to convert local distance measurements to global position. As an alternative to these more expensive systems, ultrasonic and infrared distance sensors can offer similar performance with modest compromises in speed and accuracy. Of these two, infrared distance sensors often have slightly narrower beam width and faster response.

Figure 1.2(a) shows the Sharp GP2Y0A21YK0F, a popular medium range (10-80 cm), infrared (IR) distance sensor. The Sharp sensor uses triangulation to calculate distance by measuring the angle of incidence of a transmitted IR beam reflected from a distant surface onto a receiving position sensitive device (PSD). Because the angle is a nonlinear function of the distance to the surface, the Sharp sensor has the nonlinear calibration curve shown in Figure 1.2(b). Given discrete calibration data, we can linearly interpolate voltage readings taken from the sensor to derive distance measurements.

1.3 Local Position Tracking: Odometry

Dead reckoning, which tracks location by integrating a moving system's speed and heading over time, forms the backbone of many mobile robot navigation systems. The simplest form of dead

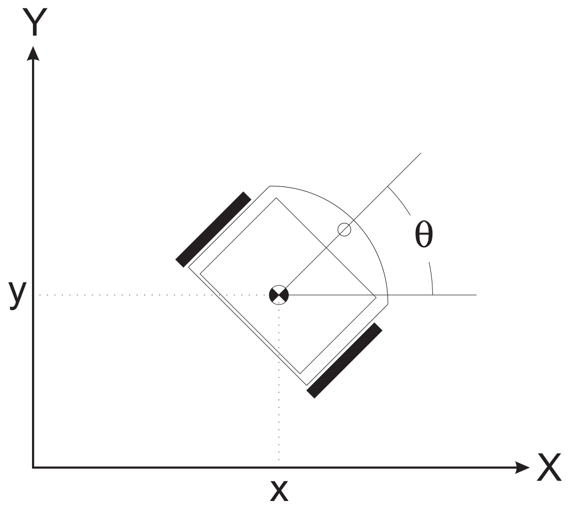
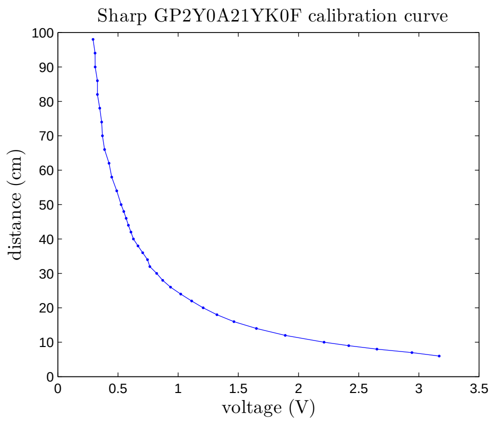


Figure 1.1: A mobile robot with pose (x, y, θ).

(a) The Sharp IR distance sensor



(b) Calibration curve

Figure 1.2: Sharp GP2Y0A21YK0F infrared distance sensor and its calibration curve.

reckoning for land-based vehicles is odometry, which derives speed and heading information from sensed wheel rotations.

Optical encoders like the one shown in Figure 1.3 are often used to derive linear displacements of the left and right wheels (Δs_{left} and Δs_{right}) from incremental wheel rotations. In the optical encoder design shown, the encoder senses changes in the reflectance of a striped pattern on the wheels, generating a pulse or "tick" for each passing stripe. Two sensors A and B placed in quadrature — 90 degrees out of phase with each other (when one is centered on a stripe, the other is centered on an edge) — permit differentiation between forward and reverse rotation. For wheels of diameter d_{wheel} with N ticks per rotation, the distance traveled by each wheel in Δn ticks can be derived as

$$\Delta s_{\text{left}} = \pi d_{\text{wheel}} \frac{\Delta n_{\text{left}}}{N}, \tag{1.1}$$

$$\Delta s_{\text{right}} = \pi d_{\text{wheel}} \frac{\Delta n_{\text{right}}}{N} . \tag{1.2}$$

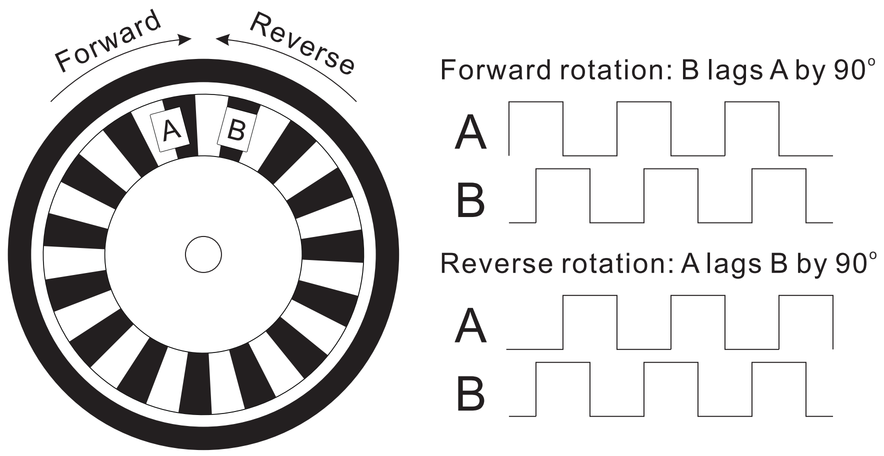

Forward
Reverse
A
B
Forward rotation: B lags A by 90°
A
B
Reverse rotation: A lags B by 90°
A
B

Figure 1.3: A quadrature rotary encoder and its output for clockwise and counterclockwise rotation.

d_{wheel}	2.71	inches
L_{baseline}	5.25	inches
N	60	ticks

Table 1.1: Mobile robot parameters.

By "summing up" the increments, we can compute the total cumulative distances s_{left} and s_{right} traveled by the left and right wheels, respectively.

The variables `time`, `LeftTicks`, and `RightTicks` from `assignment1.mat` contain sample times t^k (in seconds) and cumulative left and right encoder counts n_{left} and n_{right}, respectively, recorded during a single test run of a mobile robot. Note that the quadrature decoding for forward and reverse rotation has already been incorporated in the data, such that cumulative counts increase for forward rotation and decrease for reverse rotation. The values of the odometry constants for the mobile robot are given in Table 1.

For a mobile robot with two-wheel differential drive, in which the two (left and right) driven wheels can be controlled independently, the linear velocities v_{left} and v_{right} at the two wheels must (assuming no slippage of the wheels) be both directed in (or opposite to) the direction θ of the robot's current heading. The motion of the robot can thus be completely described by the velocity v_{center} of a point lying midway between the two wheels and the angular velocity ω about an instantaneous center of curvature (ICC) lying somewhere in line with the two wheels, as shown in Figure 1.4.

We can derive v_{center} and ω from v_{left} and v_{right} as

$$v_{\text{center}} = \frac{v_{\text{left}} + v_{\text{right}}}{2}, \tag{1.3}$$

$$\omega = \frac{v_{\text{right}} - v_{\text{left}}}{L_{\text{baseline}}}\ , \tag{1.4}$$

where

$$v_{\text{left}} = \frac{ds_{\text{left}}}{dt}, \tag{1.5}$$

$$v_{\text{right}} = \frac{ds_{\text{right}}}{dt}\ , \tag{1.6}$$

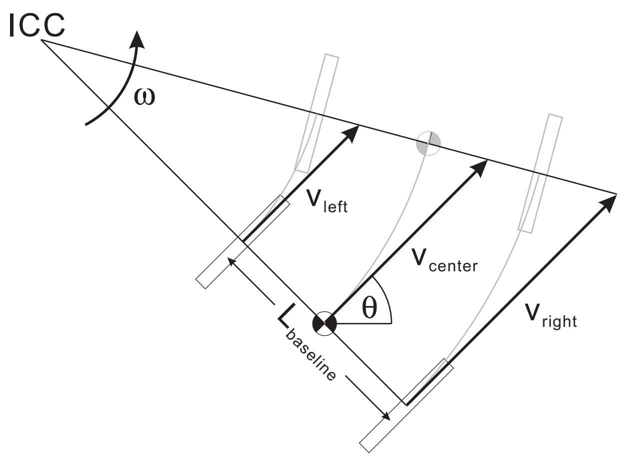


Figure 1.4: Robot trajectory.

and L_{baseline} is the distance between the points of contact of the two wheels. We can then integrate these velocities to track the pose $[x(t), y(t), \theta(t)]$ of the robot over time as

$$x(t) = \int_0^t v_{\text{center}}(t) \cos[\theta(t)]\, dt \ , \tag{1.7}$$

$$y(t) = \int_0^t v_{\text{center}}(t) \sin[\theta(t)]\, dt \ , \tag{1.8}$$

$$\theta(t) = \int_0^t \omega(t)\, dt \ . \tag{1.9}$$

In terms of the sample times t^k, we can write these equations as

$$x^k = x^{k-1} + \int_{t^{k-1}}^{t^k} v_{\text{center}}(t) \cos[\theta(t)]\, dt \ , \tag{1.10}$$

$$y^k = y^{k-1} + \int_{t^{k-1}}^{t^k} v_{\text{center}}(t) \sin[\theta(t)]\, dt \ , \tag{1.11}$$

$$\theta^k = \theta^{k-1} + \int_{t^{k-1}}^{t^k} \omega(t)\, dt \ . \tag{1.12}$$

1.4 The Numerical Tasks

To calculate distance from our transducer we must be able to interpolate; and to calculate our position from dead reckoning we must be able to differentiate and integrate. In this unit we introduce the necessary numerical approaches and also understand the possible sources of error.

Chapter 2

Interpolation

2.1 Interpolation of Univariate Functions

The objective of interpolation is to approximate the behavior of a true underlying function using function values at a limited number of points. Interpolation serves two important, distinct purposes throughout this book. First, it is a mathematical tool that facilitates development and analysis of numerical techniques for, for example, integrating functions and solving differential equations. Second, interpolation can be used to estimate or infer the function behavior based on the function values recorded as a table, for example collected in an experiment (i.e., table lookup).

Let us define the interpolation problem for an univariate function, i.e., a function of single variable. We *discretize* the domain $[x_1, x_N]$ into $N-1$ non-overlapping segments, $\{S_1, \ldots, S_{N-1}\}$, using N points, $\{x_1, \ldots, x_N\}$, as shown in Figure 2.1. Each segment is defined by

$$S_i = [x_i, x_{i+1}], \quad i = 1, \ldots, N-1 \ ,$$

and we denote the length of the segment by h, i.e.

$$h \equiv x_{i+1} - x_i \ .$$

For simplicity, we assume h is constant throughout the domain. Discretization is a concept that is used throughout numerical analysis to approximate a continuous system (infinite-dimensional problem) as a discrete system (finite-dimensional problem) so that the solution can be estimated using a computer. For interpolation, discretization is characterized by the segment size h; smaller h is generally more accurate but more costly.

Suppose, on segment S_i, we are given M interpolation points

$$\bar{x}^m, \quad m = 1, \ldots, M \ ,$$

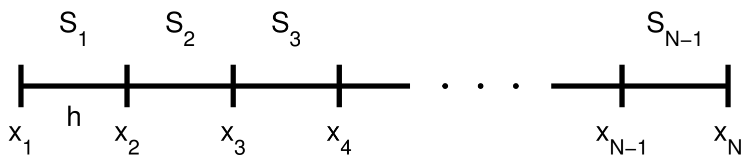
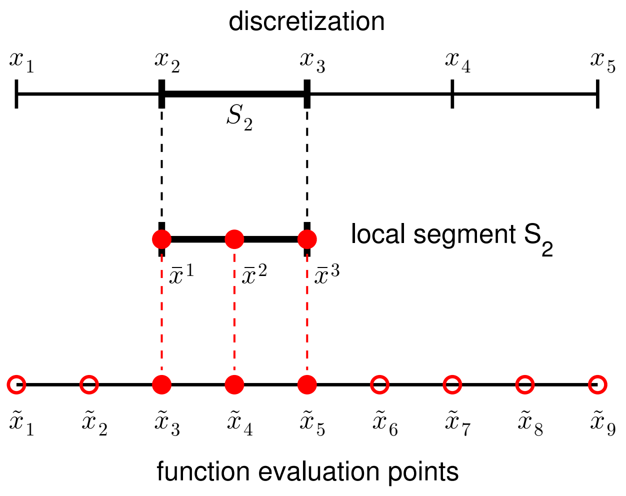
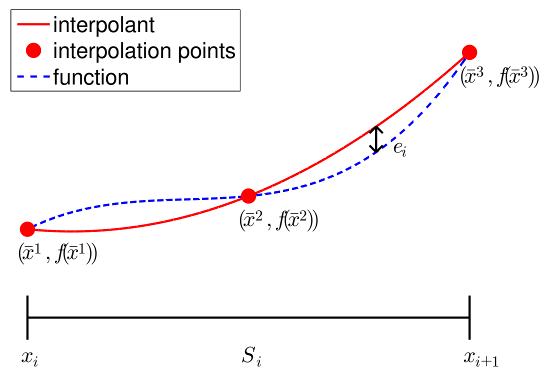


Figure 2.1: Discretization of a 1-D domain into $N-1$ segments.


discretization
x_1
x_2
x_3
x_4
x_5
S_2
local segment S$_2$
$\bar{x}^1$
$\bar{x}^2$
$\bar{x}^3$
$\tilde{x}_1$ $\tilde{x}_2$ $\tilde{x}_3$ $\tilde{x}_4$ $\tilde{x}_5$ $\tilde{x}_6$ $\tilde{x}_7$ $\tilde{x}_8$ $\tilde{x}_9$
function evaluation points

(a) discretization


interpolant
interpolation points
function
$(\bar{x}^1, f(\bar{x}^1))$
$(\bar{x}^2, f(\bar{x}^2))$
$(\bar{x}^3, f(\bar{x}^3))$
e_i
x_i
S_i
x_{i+1}

(b) local interpolation

Figure 2.2: Example of a 1-D domain discretized into four segments, a local segment with $M = 3$ function evaluation points (i.e., interpolation points), and global function evaluation points (left). Construction of an interpolant on a segment (right).

and the associated function values

$$f(\bar{x}^m), \quad m = 1, \dots, M \ .$$

We wish to approximate $f(x)$ for any given x in S_i. Specifically, we wish to construct an interpolant $\mathcal{I}f$ that approximates f in the sense that

$$(\mathcal{I}f)(x) \approx f(x), \quad \forall\, x \in S_i \ ,$$

and satisfies

$$(\mathcal{I}f)(\bar{x}^m) = f(\bar{x}^m), \quad m = 1, \dots, M \ .$$

Note that, by definition, the interpolant matches the function value at the interpolation points, $\{\bar{x}^m\}$.

The relationship between the discretization, a local segment, and interpolation points is illustrated in Figure 2.2(a). The domain $[x_1, x_5]$ is discretized into four segments, delineated by the points x_i, $i = 1, \dots, 5$. For instance, the segment S_2 is defined by x_2 and x_3 and has a characteristic length $h = x_3 - x_2$. Figure 2.2(b) illustrates construction of an interpolant on the segment S_2 using $M = 3$ interpolation points. Note that we only use the knowledge of the function evaluated at the interpolation points to construct the interpolant. In general, the points delineating the segments, x_i, need not be function evaluation points $\tilde{x}_i$, as we will see shortly.

We can also use the interpolation technique in the context of table lookup, where a table consists of function values evaluated at a set of points, i.e., $(\tilde{x}_i, f(\tilde{x}_i))$. Given a point of interest x, we first find the segment in which the point resides, by identifying $S_i = [x_i, x_{i+1}]$ with $x_i \le x \le x_{i+1}$. Then, we identify on the segment S_i the evaluation pairs $(\tilde{x}_j, f(\tilde{x}_j))$, $j = \dots$, $\Rightarrow (\bar{x}^m, f(\bar{x}^m))$, $m = 1, \dots, M$. Finally, we calculate the interpolant at x to obtain an approximation to $f(x)$, $(\mathcal{I}f)(x)$.

(Note that, while we use fixed, non-overlapping segments to construct our interpolant in this chapter, we can be more flexible in the choice of segments in general. For example, to estimate the value of a function at some point x, we can choose a set of M data points in the neighborhood of x. Using the M points, we construct a local interpolant as in Figure 2.2(b) and infer $f(x)$ by evaluating the interpolant at x. Note that the local interpolant constructed in this manner implicitly defines a local segment. The segment "slides" with the target x, i.e., it is adaptively chosen. In the current chapter on interpolation and in Chapter 7 on integration, we will emphasize the fixed segment perspective; however, in discussing differentiation in Chapter 3, we will adopt the sliding segment perspective.)

To assess the quality of the interpolant, we define its error as the maximum difference between the true function and the interpolant in the segment, i.e.

$$e_i \equiv \max_{x \in S_i} |f(x) - (\mathcal{I}f)(x)| \ .$$

Because the construction of an interpolant on a given segment is independent of that on another segment[1], we can analyze the local interpolation error one segment at a time. The locality of interpolation construction and error greatly simplifies the error analysis. In addition, we define the maximum interpolation error, $e_{\max}$, as the maximum error over the entire domain, which is equivalent to the largest of the segment errors, i.e.

$$e_{\max} \equiv \max_{i=1,\dots,N-1} e_i \ .$$

The interpolation error is a measure we use to assess the quality of different interpolation schemes. Specifically, for each interpolation scheme, we bound the error in terms of the function f and the discretization parameter h to understand how the error changes as the discretization is refined.

Let us consider an example of interpolant.

Example 2.1.1 piecewise-constant, left endpoint

The first example we consider uses a piecewise-constant polynomial to approximate the function f. Because a constant polynomial is parameterized by a single value, this scheme requires one interpolation point per interval, meaning $M = 1$. On each segment $S_i = [x_i, x_{i+1}]$, we choose the left endpoint as our interpolation point, i.e.

$$\bar{x}^1 = x_i \ .$$

As shown in Figure 2.3, we can also easily associate the segmentation points, x_i, with the global function evaluation points, $\tilde{x}_i$, i.e.

$$\tilde{x}_i = x_i, \quad i = 1, \dots, N-1 \ .$$

Extending the left-endpoint value to the rest of the segment, we obtain the interpolant of the form

$$(\mathcal{I}f)(x) = f(\bar{x}^1) = f(\tilde{x}_i) = f(x_i), \quad \forall\, x \in S_i \ .$$

Figure 2.4(a) shows the interpolation scheme applied to $f(x) = \exp(x)$ over $[0, 1]$ with $N = 5$. Because $f' > 0$ over each interval, the interpolant $\mathcal{I}f$ always underestimate the value of f. Conversely, if $f' < 0$ over an interval, the interpolant overestimates the values of f in the interval. The interpolant is exact over the interval if f is constant.

If f' exists, the error in the interpolant is bounded by

$$e_i \leq h \cdot \max_{x \in S_i} |f'(x)| \ .$$

[1]for the interpolants considered in this chapter

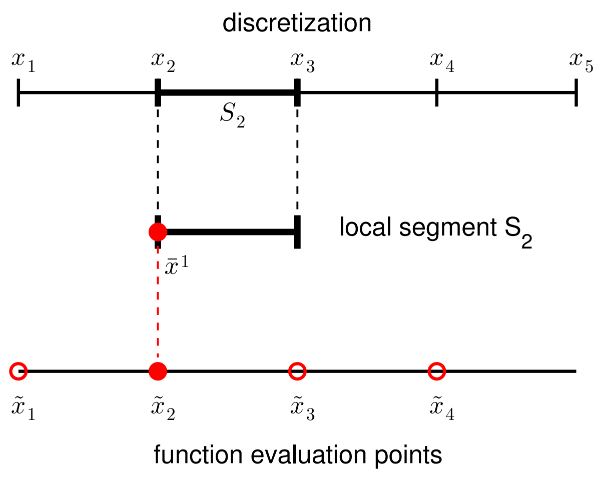
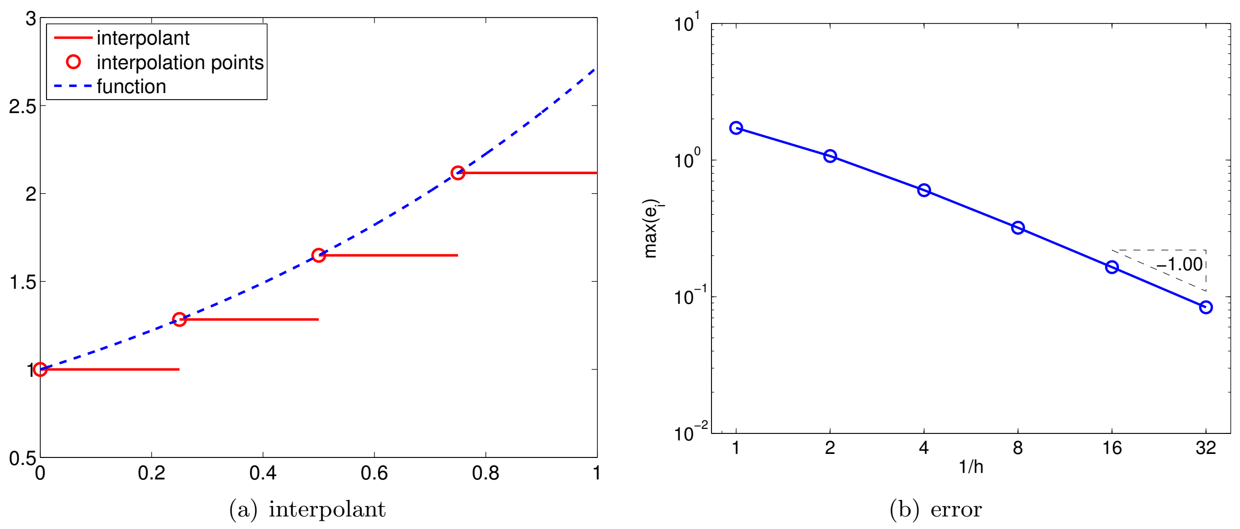


Figure 2.3: The relationship between the discretization, a local segment, and the function evaluation points for a piecewise-constant, left-endpoint interpolant.



(a) interpolant (b) error

Figure 2.4: Piecewise-constant, left-endpoint interpolant.

Since $e_i = \mathcal{O}(h)$ and the error scales as the first power of h, the scheme is said to be *first-order accurate*. The convergence behavior of the interpolant applied to the exponential function is shown in Figure 2.4(b), where the maximum value of the interpolation error, $e_{\max} = \max_i e_i$, is plotted as a function of the number of intervals, $1/h$.

We pause to review two related concepts that characterize asymptotic behavior of a sequence: the big-$\mathcal{O}$ notation ($\mathcal{O}(\,\cdot\,)$) and the asymptotic notation ($\sim$). Say that Q and z are scalar quantities (real numbers) and q is a function of z. Using the big-$\mathcal{O}$ notation, when we say that Q is $\mathcal{O}(q(z))$ as z tends to, say, zero (or infinity), we mean that there exist constants C_1 and z^* such that $|Q| < C_1|q(z)|$, $\forall\, z < z^*$ (or $\forall\, z > z^*$). On the other hand, using the asymptotic notation, when we say that $Q \sim C_2 q(z)$ as z tends to some limit, we mean that there exist a constant C_2 (not necessary equal to C_1) such that $Q/(C_2 q(z))$ tends to unity as z tends to the limit. We shall use these notations in two cases in particular: (i) when z is δ, a discretization parameter (h in our example above) — which tends to zero; (ii) when z is K, an integer related to the number of degrees of freedom that define a problem (N in our example above) — which tends to infinity. Note we need not worry about small effects with the $\mathcal{O}$ (or the asymptotic) notation: for K tends to infinity, for example, $\mathcal{O}(K) = \mathcal{O}(K-1) = \mathcal{O}(K + \sqrt{K})$. Finally, we note that the expression $Q = \mathcal{O}(1)$ means that Q effectively does not depend on some (implicit, or "understood") parameter, z.[2]

If $f(x)$ is linear, then the error bound can be shown using a direct argument. Let $f(x) = mx+b$. The difference between the function and the interpolant over S_i is

$$f(x) - (\mathcal{I}f)(x) = [mx - b] - [m\bar{x}^1 - b] = m \cdot (x - \bar{x}^1)\ .$$

Recalling the local error is the maximum difference in the function and interpolant and noting that $S_i = [x_i, x_{i+1}] = [\bar{x}^1, \bar{x}^1 + h]$, we obtain

$$e_i = \max_{x \in S_i} |f(x) - (\mathcal{I}f)(x)| = \max_{x \in S_i} |m \cdot (x - \bar{x}^1)| = |m| \cdot \max_{x \in [\bar{x}^1, \bar{x}^1 + h]} |x - \bar{x}^1| = |m| \cdot h\ .$$

Finally, recalling that $m = f'(x)$ for the linear function, we have $e_i = |f'(x)| \cdot h$. Now, let us prove the error bound for a general f.

Proof. The proof follows from the definition of the interpolant and the fundamental theorem of calculus, i.e.

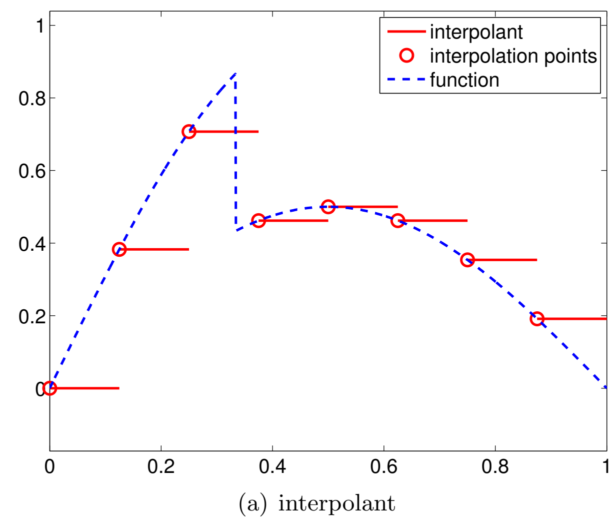
$$\begin{aligned} f(x) - (\mathcal{I}f)(x) &= f(x) - f(\bar{x}^1) && \text{(by definition of } (\mathcal{I}f)) \\ &= \int_{\bar{x}^1}^{x} f'(\xi) d\xi && \text{(fundamental theorem of calculus)} \\ &\le \int_{\bar{x}^1}^{x} |f'(\xi)| d\xi \\ &\le \max_{x \in [\bar{x}^1, x]} |f'(x)| \left| \int_{\bar{x}^1}^{x} d\xi \right| && \text{(Hölder's inequality)} \\ &\le \max_{x \in S_i} |f'(x)| \cdot h, \quad \forall\, x \in S_i = [\bar{x}^1, \bar{x}^1 + h]\ . \end{aligned}$$

Substitution of the expression into the definition of the error yields

$$e_i \equiv \max_{x \in S_i} |f(x) - (\mathcal{I}f)(x)| \le \max_{x \in S_i} |f'(x)| \cdot h\ .$$

□

[2]The engineering notation $Q = \mathcal{O}(10^3)$ is somewhat different and really just means that the number is roughly 10^3.



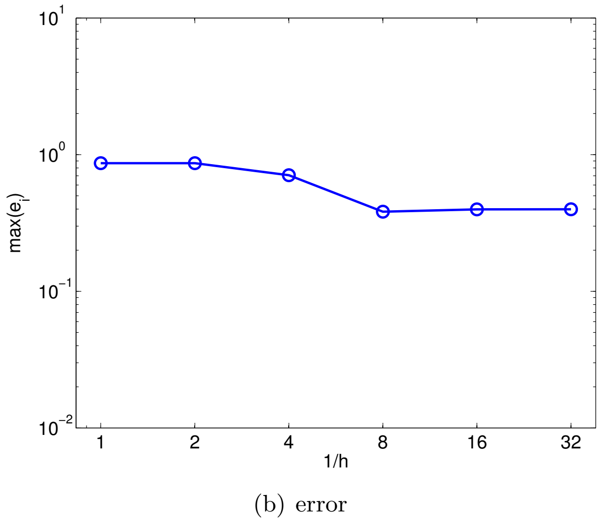


Figure 2.5: Piecewise-constant, left-endpoint interpolant for a non-smooth function.

It is important to note that the proof relies on the smoothness of f. In fact, if f is discontinuous and f' does not exist, then e_i can be $\mathcal{O}(1)$. In other words, the interpolant does not converge to the function (in the sense of maximum error), even if the h is refined. To demonstrate this, let us consider a function

$$f(x) = \begin{cases} \sin(\pi x), & x \le \dfrac{1}{3} \\ \dfrac{1}{2}\sin(\pi x), & x > \dfrac{1}{3} \end{cases},$$

which is discontinuous at $x = 1/3$. The result of applying the piecewise constant, left-endpoint rule to the function is shown in Figure 2.5(a). We note that the solution on the third segment is not approximated well due to the presence of the discontinuity. More importantly, the convergence plot, Figure 2.5(b), confirms that the maximum interpolation error does not converge even if h is refined. This reduction in the convergence rate for non-smooth functions is not unique to this particular interpolation rule; all interpolation rules suffer from this problem. Thus, we must be careful when we interpolate a non-smooth function.

Finally, we comment on the distinction between the "best fit" (in some norm, or metric) and the interpolant. The best fit in the "max" or "sup" norm of a constant function c_i to $f(x)$ over S_i minimizes $|c_i - f(x)|$ over S_i and will typically be different and perforce better (in the chosen norm) than the interpolant. However, the determination of c_i in principle requires knowledge of $f(x)$ at (almost) all points in S_i whereas the interpolant only requires knowledge of $f(x)$ at one point — hence much more useful. We discuss this further in Section 2.1.1.

Let us more formally define some of the key concepts visited in the first example. While we introduce the following concepts in the context of analyzing the interpolation schemes, the concepts apply more generally to analyzing various numerical schemes.

- *Accuracy* relates how well the numerical scheme (finite-dimensional) approximates the continuous system (infinite-dimensional). In the context of interpolation, the accuracy tells how well the interpolant $\mathcal{I}f$ approximates f and is measured by the interpolation error, $e_{\max}$.

- *Convergence* is the property that the error vanishes as the discretization is refined, i.e.

$$e_{\max} \to 0 \quad \text{as} \quad h \to 0 \; .$$

  A convergent scheme can achieve any desired accuracy (error) in infinite prediction arithmetics by choosing h sufficiently small. The piecewise-constant, left-endpoint interpolant is a convergent scheme, because $e_{\max} = \mathcal{O}(h)$, and $e_{\max} \to 0$ as $h \to 0$.

- *Convergence rate* is the power p such that

$$e_{\max} \le Ch^p \quad \text{as} \quad h \to 0 \; ,$$

  where C is a constant independent of h. The scheme is *first-order accurate* for $p = 1$, *second-order accurate* for $p = 2$, and so on. The piecewise-constant, left-endpoint interpolant is *first-order accurate* because $e_{\max} = Ch^1$. Note here p is fixed and the convergence (with number of intervals) is thus algebraic.

  Note that typically as $h \to 0$ we obtain not a bound but in fact asymptotic behavior: $e_{\max} \sim Ch^p$ or equivalently, $e_{\max}/(Ch^p) \to 1$, as $h \to 0$. Taking the logarithm of $e_{\max} \sim Ch^p$, we obtain $\ln(e_{\max}) \sim \ln C + p \ln h$. Thus, a log-log plot is a convenient means of finding p empirically.

- *Resolution* is the characteristic length h_{crit} for any particular problem (described by f) for which we see the asymptotic convergence rate for $h \le h_{\text{crit}}$. Convergence plot in Figure 2.4(b) shows that the piecewise-constant, left-endpoint interpolant achieves the asymptotic convergence rate of 1 with respect to h for $h \le 1/2$; note that the slope from $h = 1$ to $h = 1/2$ is lower than unity. Thus, h_{crit} for the interpolation scheme applied to $f(x) = \exp(x)$ is approximately $1/2$.

- *Computational cost* or *operation count* is the number of floating point operations (FLOPs[3]) to compute I_h. As $h \to 0$, the number of FLOPs approaches ∞. The scaling of the computation cost with the size of the problem is referred to as *computational complexity*. The actual *run-time* of computation is a function of the computational cost and the hardware. The cost of constructing the piecewise-constant, left end point interpolant is proportional to the number of segments. Thus, the cost scales linearly with N, and the scheme is said to have linear complexity.

- *Memory* or *storage* is the number of floating point numbers that must be stored at any point during execution.

We note that the above properties characterize a scheme in *infinite precision* representation and arithmetic. *Precision* is related to machine precision, floating point number truncation, rounding and arithmetic errors, etc, all of which are absent in infinite-precision arithmetics.

We also note that there are two conflicting demands; the accuracy of the scheme increases with decreasing h (assuming the scheme is convergent), but the computational cost increases with decreasing h. Intuitively, this always happen because the dimension of the discrete approximation must be increased to better approximate the continuous system. However, some schemes produce lower error for the same computational cost than other schemes. The performance of a numerical scheme is assessed in terms of the accuracy it delivers for a given computational cost.

We will now visit several other interpolation schemes and characterize the schemes using the above properties.

[3]Not to be confused with the FLOPS (floating point operations per second), which is often used to measure the performance of a computational hardware.

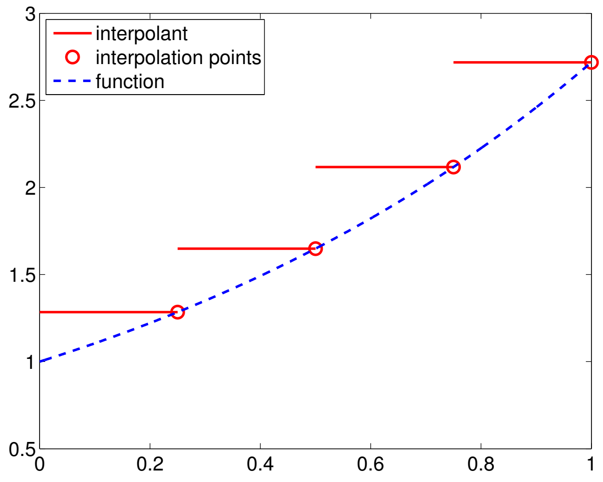


Figure 2.6: Piecewise-constant, right-endpoint interpolant.

Example 2.1.2 piecewise-constant, right end point
This interpolant also uses a piecewise-constant polynomial to approximate the function f, and thus requires one interpolation point per interval, i.e., $M = 1$. This time, the interpolation point is at the right endpoint, instead of the left endpoint, resulting in

$$\bar{x}^1 = x_{i+1}, \quad (\mathcal{I}f)(x) = f(\bar{x}^1) = f(x_{i+1}), \quad \forall\, x \in S_i = [x_i, x_{i+1}] \ .$$

The global function evaluation points, $\tilde{x}_i$, are related to segmentation points, x_i, by

$$\tilde{x}_i = x_{i+1}, \quad i = 1, \ldots, N-1 \ .$$

Figure 2.6 shows the interpolation applied to the exponential function.

If f' exists, the error in the interpolant is bounded by

$$e_i \le h \cdot \max_{x \in S_i} |f'(x)| \ ,$$

and thus the scheme is first-order accurate. The proof is similar to that of the piecewise-constant, right-endpoint interpolant.

·

Example 2.1.3 piecewise-constant, midpoint
This interpolant uses a piecewise-constant polynomial to approximate the function f, but uses the midpoint of the segment, $S_i = [x_i, x_{i+1}]$, as the interpolation point, i.e.

$$\bar{x}^1 = \frac{1}{2}(x_i + x_{i+1}) \ .$$

Denoting the (global) function evaluation point associated with segment S_i as $\tilde{x}_i$, we have

$$\tilde{x}_i = \frac{1}{2}(x_i + x_{i+1}), \quad i = 1, \ldots, N-1 \ ,$$

as illustrated in Figure 2.7. Note that the segmentation points x_i do not correspond to the function evaluation points $\tilde{x}_i$ unlike in the previous two interpolants. This choice of interpolation point results in the interpolant

$$(\mathcal{I}f)(x) = f(\bar{x}^1) = f(\tilde{x}_i) = f\left(\frac{1}{2}(x_i + x_{i+1})\right), \quad x \in S_i \ .$$

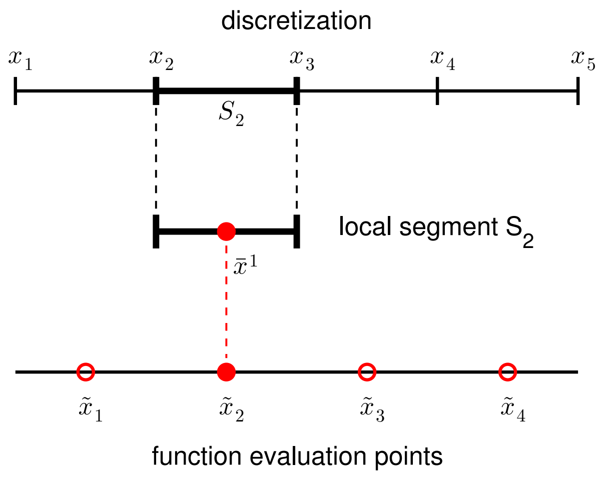


Figure 2.7: The relationship between the discretization, a local segment, and the function evaluation points for a piecewise-constant, midpoint interpolant.

Figure 2.8(a) shows the interpolant for the exponential function. In the context of table lookup, this interpolant naturally arises if a value is approximated from a table of data choosing the nearest data point.

The error of the interpolant is bounded by

$$e_i \leq \frac{h}{2} \cdot \max_{x \in S_i} |f'(x)| \ ,$$

where the factor of half comes from the fact that any function evaluation point is less than $h/2$ distance away from one of the interpolation points. Figure 2.8(a) shows that the midpoint interpolant achieves lower error than the left- or right-endpoint interpolant. However, the error still scales linearly with h, and thus the midpoint interpolant is first-order accurate.

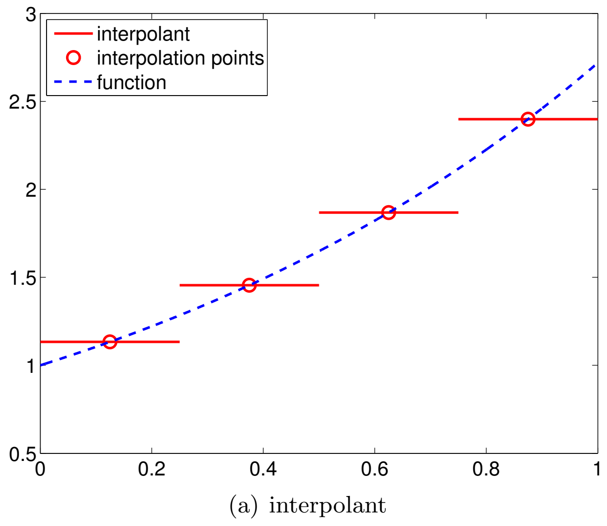
For a linear function $f(x) = mx + b$, the sharp error bound can be obtained from a direct argument. The difference between the function and its midpoint interpolant is

$$f(x) - (\mathcal{I}f)(x) = [mx + b] - \left[m\bar{x}^1 + b\right] = m \cdot (x - \bar{x}^1) \ .$$

The difference vanishes at the midpoint, and increases linearly with the distance from the midpoint. Thus, the difference is maximized at either of the endpoints. Noting that the segment can be expressed as $S_i = [x_i, x_{i+1}] = \left[\bar{x}^1 - \frac{h}{2}, \bar{x}^1 + \frac{h}{2}\right]$, the maximum error is given by

$$\begin{aligned} e_i &\equiv \max_{x \in S_i}(f(x) - (\mathcal{I}f)(x)) = \max_{x \in \left[\bar{x}^1 - \frac{h}{2}, \bar{x}^1 + \frac{h}{2}\right]} |m \cdot (x - \bar{x}^1)| \\ &= |m \cdot (x - \bar{x}^1)|_{x = \bar{x}^1 \pm h/2} = |m| \cdot \frac{h}{2} \ . \end{aligned}$$

Recalling $m = f'(x)$ for the linear function, we have $e_i = |f'(x)|h/2$. A sharp proof for a general f follows essentially that for the piecewise-constant, left-endpoint rule.



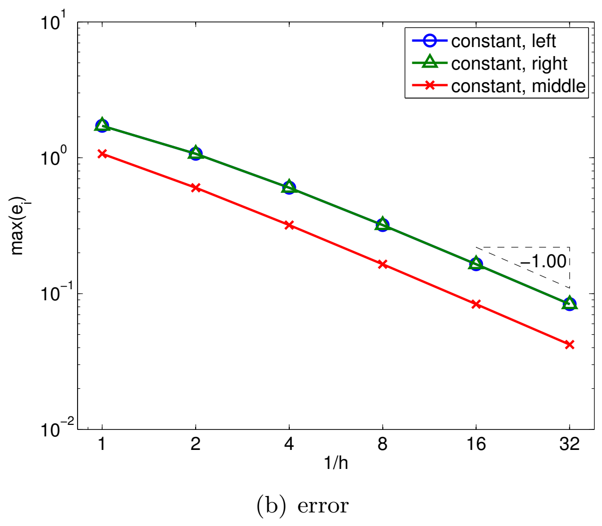


Figure 2.8: Piecewise-constant, mid point interpolant.

Proof. The proof follows from the fundamental theorem of calculus,

$$f(x) - (\mathcal{I}f)(x) = f(x) - f\left(\bar{x}^1\right) = \int_{\bar{x}^1}^{x} f'(\xi)d\xi \leq \int_{\bar{x}^1}^{x} |f'(\xi)|d\xi \leq \max_{x\in[\bar{x}^1,x]} |f'(x)| \left|\int_{\bar{x}^1}^{x} d\xi\right|$$

$$\leq \max_{x\in\left[\bar{x}^1-\frac{h}{2},\bar{x}^1+\frac{h}{2}\right]} |f'(x)| \cdot \frac{h}{2}, \quad \forall\, x \in S_i = \left[\bar{x}^1 - \frac{h}{2}, \bar{x}^1 + \frac{h}{2}\right].$$

Thus, we have

$$e_i = \max_{x\in S_i} |f(x) - (\mathcal{I}f)(x)| \leq \max_{x\in S_i} |f'(x)| \cdot \frac{h}{2} \;.$$

□

·

Example 2.1.4 piecewise-linear

The three examples we have considered so far used piecewise-constant functions to interpolate the function of interest, resulting in the interpolants that are first-order accurate. In order to improve the quality of interpolation, we consider a second-order accurate interpolant in this example. To achieve this, we choose a piecewise-linear function (i.e., first-degree polynomials) to approximate the function behavior. Because a linear function has two coefficients, we must choose two interpolation points per segment to uniquely define the interpolant, i.e., $M = 2$. In particular, for segment $S_i = [x_i, x_{i+1}]$, we choose its endpoints, x_i and x_{i+1}, as the interpolation points, i.e.

$$\bar{x}^1 = x_i \quad \text{and} \quad \bar{x}^2 = x_{i+1} \;.$$

The (global) function evaluation points and the segmentation points are trivially related by

$$\tilde{x}_i = x_i, \quad i = 1, \ldots, N \;,$$

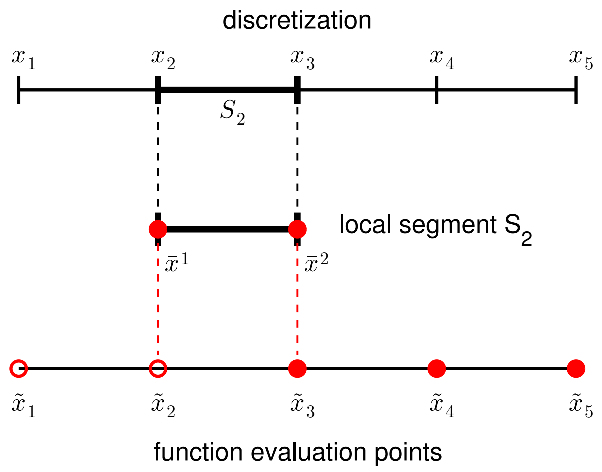


Figure 2.9: The relationship between the discretization, a local segment, and the function evaluation points for a linear interpolant.

as illustrated in Figure 2.9.

The resulting interpolant, defined using the local coordinate, is of the form

$$(\mathcal{I}f)(x) = f(\bar{x}^1) + \left(\frac{f(\bar{x}^2) - f(\bar{x}^1)}{h}\right)(x - \bar{x}^1), \quad \forall\, x \in S_i \ , \tag{2.1}$$

or, in the global coordinate, is expressed as

$$(\mathcal{I}f)(x) = f(x_i) + \left(\frac{f(x_{i+1}) - f(x_i)}{h_i}\right)(x - x_i), \quad \forall\, x \in S_i \ .$$

Figure 2.10(a) shows the linear interpolant applied to $f(x) = \exp(x)$ over $[0, 1]$ with $N = 5$. Note that this interpolant is continuous across the segment endpoints, because each piecewise-linear function matches the true function values at its endpoints. This is in contrast to the piecewise-constant interpolants considered in the previous three examples, which were discontinuous across the segment endpoints in general.

If f'' exists, the error of the linear interpolant is bounded by

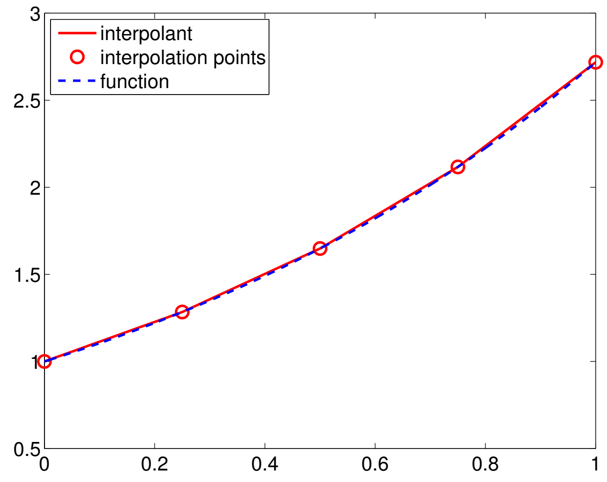
$$e_i \le \frac{h^2}{8} \cdot \max_{x \in S_i} |f''(x)| \ .$$

The error of the linear interpolant converges *quadratically* with the interval length, h. Because the error scales with h^2, the method is said to be *second-order accurate*. Figure 2.10(b) shows that the linear interpolant is significantly more accurate than the piecewise-linear interpolant for the exponential function. This trend is generally true for sufficient smooth functions. More importantly, the higher-order convergence means that the linear interpolant approaches the true function at a faster rate than the piecewise-constant interpolant as the segment length decreases.

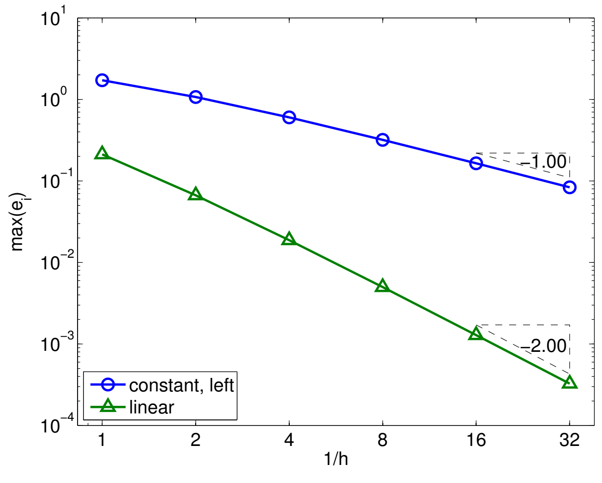
Let us provide a sketch of the proof. First, noting that $f(x) - (\mathcal{I}f)(x)$ vanishes at the endpoints, we express our error as

$$f(x) - (\mathcal{I}f)(x) = \int_{\bar{x}^1}^{x} (f - \mathcal{I}f)'(t)\, dt \ .$$

Next, by the Mean Value Theorem (MVT), we have a point $x^* \in S_i = [\bar{x}^1, \bar{x}^2]$ such that $f'(x^*) - (\mathcal{I}f)'(x^*) = 0$. Note the MVT — for a continuously differentiable function f there exists an



(a) interpolant



(b) error

Figure 2.10: Piecewise-linear interpolant.

$x^* \in [\bar{x}^1, \bar{x}^2]$ such that $f'(x^*) = (f(\bar{x}^2) - f(\bar{x}^1))/h$ — follows from Rolle's Theorem. Rolle's Theorem states that, for a continuously differentiable function g that vanishes at $\bar{x}^1$ and $\bar{x}^2$, there exists a point x^* for which $g'(x^*) = 0$. To derive the MVT we take $g(x) = f(x) - \mathcal{I}f(x)$ for $\mathcal{I}f$ given by Eq. (2.1). Applying the fundamental theorem of calculus again, the error can be expressed as

$$f(x) - (\mathcal{I}f)(x) = \int_{\bar{x}^1}^{x} (f - \mathcal{I}f)'(t)\, dt = \int_{\bar{x}^1}^{x} \int_{x^*}^{t} (f - \mathcal{I}f)''(s)\, ds\, dt = \int_{\bar{x}^1}^{x} \int_{x^*}^{t} f''(s)\, ds\, dt$$

$$\le \max_{x \in S_i} |f''(x)| \int_{\bar{x}^1}^{x} \int_{x^*}^{t} ds\, dt \le \frac{h^2}{2} \cdot \max_{x \in S_i} |f''(x)| \ .$$

This simple sketch shows that the interpolation error is dependent on the second derivative of f and quadratically varies with the segment length h; however, the constant is not sharp. A sharp proof is provided below.

Proof. Our objective is to obtain a bound for $|f(\hat{x}) - \mathcal{I}f(\hat{x})|$ for an arbitrary $\hat{x} \in S_i$. If $\hat{x}$ is the one of the endpoints, the interpolation error vanishes trivially; thus, we assume that $\hat{x}$ is not one of the endpoints. The proof follows from a construction of a particular quadratic interpolant and the application of the Rolle's theorem. First let us form the quadratic interpolant, $q(x)$, of the form

$$q(x) \equiv (\mathcal{I}f)(x) + \lambda w(x) \quad \text{with} \quad w(x) = (x - \bar{x}^1)(x - \bar{x}^2) \ .$$

Since $(\mathcal{I}f)$ matches f at $\bar{x}^1$ and $\bar{x}^2$ and $q(\bar{x}^1) = q(\bar{x}^2) = 0$, $q(x)$ matches f at $\bar{x}^1$ and $\bar{x}^2$. We select λ such that q matches f at $\hat{x}$, i.e.

$$q(\hat{x}) = (\mathcal{I}f)(\hat{x}) + \lambda w(\hat{x}) = f(\hat{x}) \quad \Rightarrow \quad \lambda = \frac{f(\hat{x}) - (\mathcal{I}f)(\hat{x})}{w(\hat{x})} \ .$$

The interpolation error of the quadratic interpolant is given by

$$\phi(x) = f(x) - q(x) \ .$$

Because q is the quadratic interpolant of f defined by the interpolation points $\bar{x}^1$, $\bar{x}^2$, and $\hat{x}$, ϕ has three zeros in S_i. By Rolle's theorem, ϕ' has two zeros in S_i. Again, by Rolle's theorem, ϕ'' has one zero in S_i. Let this zero be denoted by ξ, i.e., $\phi''(\xi) = 0$. Evaluation of $\phi''(\xi)$ yields

$$0 = \phi''(\xi) = f''(\xi) - q''(\xi) = f''(\xi) - (\mathcal{I}f)''(\xi) - \lambda w''(\xi) = f''(\xi) - 2\lambda \quad \Rightarrow \quad \lambda = \frac{1}{2}f''(\xi) \ .$$

Evaluating $\phi(\hat{x})$, we obtain

$$0 = \phi(\hat{x}) = f(\hat{x}) - (\mathcal{I}f)(\hat{x}) - \frac{1}{2}f''(\xi)w(\hat{x}) \tag{2.2}$$

$$f(\hat{x}) - (\mathcal{I}f)(\hat{x}) = \frac{1}{2}f''(\xi)(\hat{x} - \bar{x}^1)(\hat{x} - \bar{x}^2) \ . \tag{2.3}$$

The function is maximized for $\hat{x}^* = (\bar{x}^1 + \bar{x}^2)/2$, which yields

$$f(\hat{x}) - (\mathcal{I}f)(\hat{x}) \le \frac{1}{8}f''(\xi)(\bar{x}^2 - \bar{x}^1)^2 = \frac{1}{8}h^2 f''(\xi), \quad \forall\, \hat{x} \in [\bar{x}^1, \bar{x}^2]$$

Since $\xi \in S_i$, it follows that,

$$e_i = \max_{x \in S_i} |f(x) - (\mathcal{I}f)(x)| \le \frac{1}{8}h^2 f''(\xi) \le \frac{1}{8}h^2 \max_{x \in S_i} |f''(x)| \ .$$

□

Example 2.1.5 piecewise-quadratic
Motivated by the higher accuracy provided by the second-order accurate, piecewise-linear interpolants, we now consider using a piecewise-quadratic polynomial to construct an interpolant. Because a quadratic function is characterized by three parameters, we require three interpolation points per segment ($M = 3$). For segment $S_i = [x_i, x_{i+1}]$, a natural choice are the two endpoints and the midpoint, i.e.

$$\bar{x}^1 = x_i, \quad \bar{x}^2 = \frac{1}{2}(x_i + x_{i+1}), \quad \text{and} \quad \bar{x}^3 = x_{i+1} \ .$$

To construct the interpolant, we first construct Lagrange basis polynomial of the form

$$\phi_1(x) = \frac{(x - \bar{x}^2)(x - \bar{x}^3)}{(\bar{x}^1 - \bar{x}^2)(\bar{x}^1 - \bar{x}^3)}, \quad \phi_2(x) = \frac{(x - \bar{x}^1)(x - \bar{x}^3)}{(\bar{x}^2 - \bar{x}^1)(\bar{x}^2 - \bar{x}^3)}, \quad \text{and} \quad \phi_3(x) = \frac{(x - \bar{x}^1)(x - \bar{x}^2)}{(\bar{x}^3 - \bar{x}^1)(\bar{x}^3 - \bar{x}^2)} \ .$$

By construction, ϕ_1 takes the value of 1 at $\bar{x}^1$ and vanishes at $\bar{x}^2$ and $\bar{x}^3$. More generally, the Lagrange basis has the property

$$\phi_m(\bar{x}^n) = \begin{cases} 1, & n = m \\ 0, & n \neq m \end{cases} \ .$$

Using these basis functions, we can construct the quadratic interpolant as

$$(\mathcal{I}f)(x) = f(\bar{x}^1)\phi_1(x) + f(\bar{x}^2)\phi_2(x) + f(\bar{x}^3)\phi_3(x), \quad \forall\, x \in S_i \ . \tag{2.4}$$

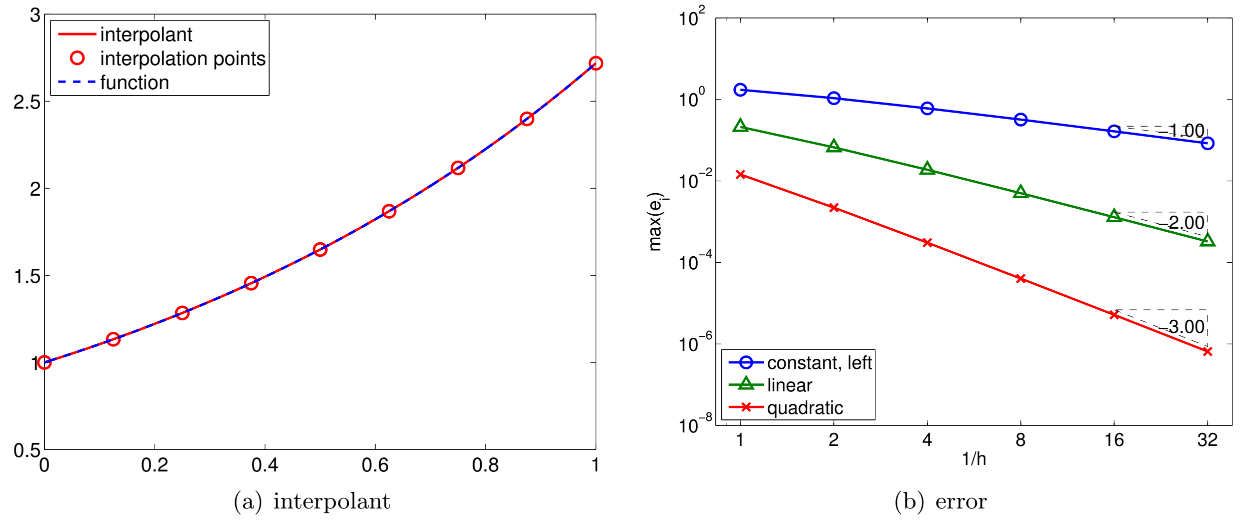


Figure 2.11: Piecewise-quadratic interpolant.

We can easily confirm that the quadratic function goes through the interpolation points, $(\bar{x}^m, f(\bar{x}^m))$, $m = 1, 2, 3$, using the property of the Lagrange basis. Figure 2.11(a) shows the interpolant for the exponential function.

If f''' exists, the error of the quadratic interpolant is bounded by

$$e_i \leq \frac{h^3}{72\sqrt{3}} \max_{x \in S_i} f'''(x) \ .$$

The error converges as the *cubic* power of h, meaning the scheme is *third-order accurate*. Figure 2.11(b) confirms the higher-order convergence of the piecewise-quadratic interpolant.

Proof. The proof is an extension of that for the linear interpolant. First, we form a cubic interpolant of the form

$$q(x) \equiv (\mathcal{I}f)(x) + \lambda w(x) \quad \text{with} \quad w(x) = (x - \bar{x}^1)(x - \bar{x}^2)(x - \bar{x}^3) \ .$$

We select λ such that q matches f at $\hat{x}$. The interpolation error function,

$$\phi(x) = f(x) - q(x) \ ,$$

has four zeros in S_i, specifically $\bar{x}^1$, $\bar{x}^2$, $\bar{x}^3$, and $\hat{x}$. By repeatedly applying the Rolle's theorem three times, we note that $\phi'''(x)$ has one zero in S_i. Let us denote this zero by ξ, i.e., $\phi'''(\xi) = 0$. This implies that

$$\phi'''(\xi) = f'''(\xi) - (c\mathcal{I}f)'''(\xi) - \lambda w'''(\xi) = f'''(\xi) - 6\lambda = 0 \quad \Rightarrow \quad \lambda = \frac{1}{6} f'''(\xi) \ .$$

Rearranging the expression for $\phi(\hat{x})$, we obtain

$$f(\hat{x}) - (\mathcal{I}f)(\hat{x}) = \frac{1}{6} f'''(\xi) w(\hat{x}) \ .$$

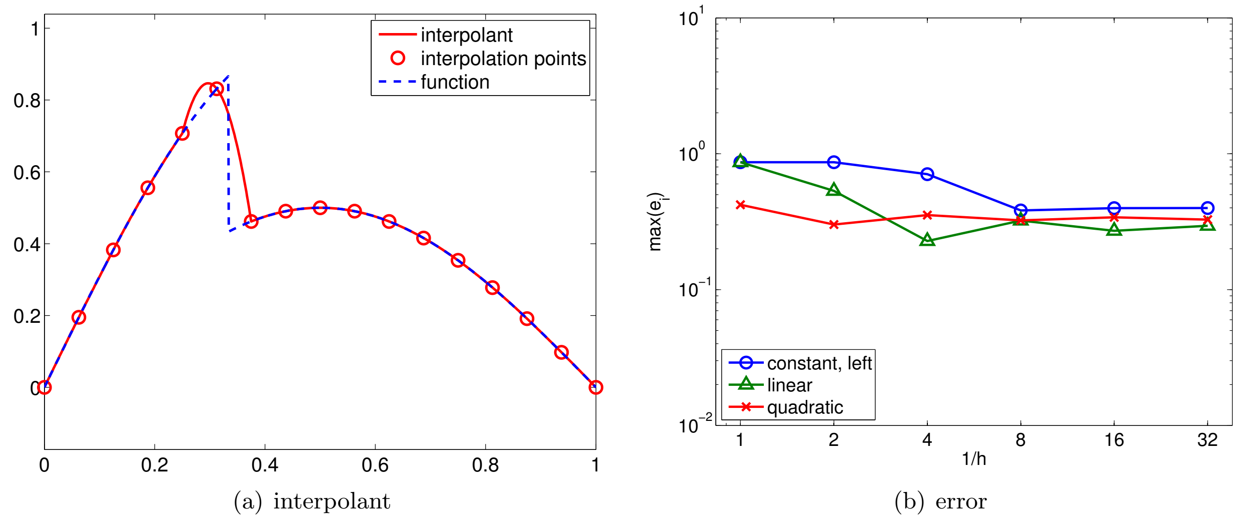


Figure 2.12: Piecewise-quadratic interpolant for a non-smooth function.

The maximum value that w takes over S_i is $h^3/(12\sqrt{3})$. Combined with the fact $f'''(\xi) \leq \max_{x\in S_i} f'''(x)$, we obtain the error bound

$$e_i = \max_{x\in S_i} |f(x) - (\mathcal{I}f)(x)| \leq \frac{h^3}{72\sqrt{3}} \max_{x\in S_i} f'''(x) \ .$$

Note that the extension of this proof to higher-order interpolants is straight forward. In general, a piecewise p^{th}-degree polynomial interpolant exhibits $p+1$ order convergence. $\square$

·

The procedure for constructing the Lagrange polynomials extends to arbitrary degree polynomials. Thus, in principle, we can construct an arbitrarily high-order interpolant by increasing the number of interpolation points. While the higher-order interpolation yielded a lower interpolation error for the smooth function considered, a few cautions are in order.

First, higher-order interpolants are more susceptible to modeling errors. If the underlying data is noisy, the "overfitting" of the noisy data can lead to inaccurate interpolant. This will be discussed in more details in Unit III on regression.

Second, higher-order interpolants are also typically not advantageous for non-smooth functions. To see this, we revisit the simple discontinuous function,

$$f(x) = \begin{cases} \sin(\pi x), & x \leq \dfrac{1}{3} \\ \dfrac{1}{2}\sin(\pi x), & x > \dfrac{1}{3} \end{cases} \ .$$

The result of applying the piecewise-quadratic interpolation rule to the function is shown in Figure 2.12(a). The quadratic interpolant closely matches the underlying function in the smooth region. However, in the third segment, which contains the discontinuity, the interpolant differs considerably from the underlying function. Similar to the piecewise-constant interpolation of the function, we again commit $\mathcal{O}(1)$ error measured in the maximum difference. Figure 2.12(b) confirms that the higher-order interpolants do not perform any better than the piecewise-constant interpolant in the presence of discontinuity. Formally, we can show that the maximum-error convergence of any interpolation scheme can be no better than h^r, where r is the highest-order derivative that is defined everywhere in the domain. In the presence of a discontinuity, $r = 0$, and we observe $\mathcal{O}(h^r) = \mathcal{O}(1)$ convergence (i.e., no convergence).

Third, for a very high-order polynomials, the interpolation points must be chosen carefully to achieve a good result. In particular, the uniform distribution suffers from the behavior known as Runge's phenomenon, where the interpolant exhibits excessive oscillation even if the underlying function is smooth. The spurious oscillation can be minimized by clustering the interpolation points near the segment endpoints, e.g., Chebyshev nodes.

Advanced Material

2.1.1 Best Fit vs. Interpolation: Polynomials of Degree n

We will study in more details how the choice of interpolation points affect the quality of a polynomial interpolant. For convenience, let us denote the space of n^{th}-degree polynomials on segment S by $\mathcal{P}_n(S)$. For consistency, we will denote n^{th}-degree polynomial interpolant of f, which is defined by $n+1$ interpolation points $\{\bar{x}^m\}_{m=1}^{n+1}$, by $\mathcal{I}_n f$. We will compare the quality of the interpolant with the "best" $n+1$ degree polynomial. We will define "best" in the infinity norm, i.e., the best polynomial $v^* \in \mathcal{P}_n(S)$ satisfies

$$\max_{x \in S} |f(x) - v^*(x)| \le \max_{x \in S} |f(x) - v(x)| \ , \quad \forall\, v \in \mathcal{P}_n(x) \ .$$

In some sense, the polynomial v^* fits the function f as closely as possible. Then, the quality of a n^{th}-degree interpolant can be assessed by measuring how close it is to v^*. More precisely, we quantify its quality by comparing the maximum error of the interpolant with that of the best polynomial, i.e.

$$\max_{x \in S} |f(x) - (\mathcal{I}f)(x)| \le (1 + \Lambda(\{\bar{x}^m\}_{m=1}^{n+1})) \max_{x \in S} |f(x) - v^*(x)| \ ,$$

where the constant Λ is called the Lebesgue constant. Clearly, a smaller Lebesgue constant implies smaller error, so higher the quality of the interpolant. At the same time, $\Lambda \ge 0$ because the maximum error in the interpolant cannot be better than that of the "best" function, which by definition minimizes the maximum error. In fact, the Lebesgue constant is given by

$$\Lambda\left(\{\bar{x}^m\}_{m=1}^{n+1}\right) = \max_{x \in S} \sum_{m=1}^{n+1} |\phi_m(x)| \ ,$$

where ϕ_m, $m = 1, \ldots, n+1$, are the Lagrange bases functions defined by the nodes $\{\bar{x}^m\}_{m=1}^{n+1}$.

Proof. We first express the interpolation error in the infinity norm as the sum of two contributions

$$\begin{aligned}\max_{x \in S} |f(x) - (\mathcal{I}f)(x)| &\le \max_{x \in S} |f(x) - v^*(x) + v^*(x) - (\mathcal{I}f)(x)| \\ &\le \max_{x \in S} |f(x) - v^*(x)| + \max_{x \in S} |v^*(x) - (\mathcal{I}f)(x)|.\end{aligned}$$

Noting that the functions in the second term are polynomial, we express them in terms of the Lagrange basis ϕ_m, $m = 1, \ldots, n$,

$$\begin{aligned}
\max_{x\in S} |v^*(x) - (\mathcal{I}f)(x)| &= \max_{x\in S} \left| \sum_{m=1}^{n+1} (v^*(\bar{x}^m) - (\mathcal{I}f)(\bar{x}^m))\phi_m(x) \right| \\
&\le \max_{x\in S} \left| \max_{m=1,\ldots,n+1} |v^*(\bar{x}^m) - (\mathcal{I}f)(\bar{x}^m)| \cdot \sum_{m=1}^{n+1} |\phi_m(x)| \right| \\
&= \max_{m=1,\ldots,n+1} |v^*(\bar{x}^m) - (\mathcal{I}f)(\bar{x}^m)| \cdot \max_{x\in S} \sum_{m=1}^{n+1} |\phi_m(x)| \ .
\end{aligned}$$

Because $\mathcal{I}f$ is an interpolant, we have $f(\bar{x}^m) = (\mathcal{I}f)(\bar{x}^m)$, $m = 1, \ldots, n+1$. Moreover, we recognize that the second term is the expression for Lebesgue constant. Thus, we have

$$\begin{aligned}
\max_{x\in S} |v^*(x) - (\mathcal{I}f)(x)| &\le \max_{m=1,\ldots,n+1} |v^*(\bar{x}^m) - f(\bar{x}^m)| \cdot \Lambda \\
&\le \max_{x\in S} |v^*(x) - f(x)| \, \Lambda \ .
\end{aligned}$$

where the last inequality follows from recognizing $\bar{x}^m \in S$, $m = 1, \ldots, n+1$. Thus, the interpolation error in the maximum norm is bounded by

$$\begin{aligned}
\max_{x\in S} |f(x) - (\mathcal{I}f)(x)| &\le \max_{x\in S} |v^*(x) - f(x)| + \max_{x\in S} |v^*(x) - f(x)| \, \Lambda \\
&\le (1 + \Lambda) \max_{x\in S} |v^*(x) - f(x)| \ ,
\end{aligned}$$

which is the desired result. □

In the previous section, we noted that equally spaced points can produce unstable interpolants for a large n. In fact, the Lebesgue constant for the equally spaced node distribution varies as

$$\Lambda \sim \frac{2^n}{en\log(n)} \ ,$$

i.e., the Lebesgue constant increases exponentially with n. Thus, increasing n does not necessary results in a smaller interpolation error.

A more stable interpolant can be formed using the Chebyshev node distribution. The node distribution is given (on $[-1, 1]$) by

$$\bar{x}^m = \cos\left(\frac{2m-1}{2(n+1)}\pi\right), \quad m = 1, \ldots, n+1 \ .$$

Note that the nodes are clustered toward the endpoints. The Lebesgue constant for Chebyshev node distribution is

$$\Lambda = 1 + \frac{2}{\pi}\log(n+1) \ ,$$

i.e., the constant grows much more slowly. The variation in the Lebesgue constant for the equally-spaced and Chebyshev node distributions are shown in Figure 2.13.

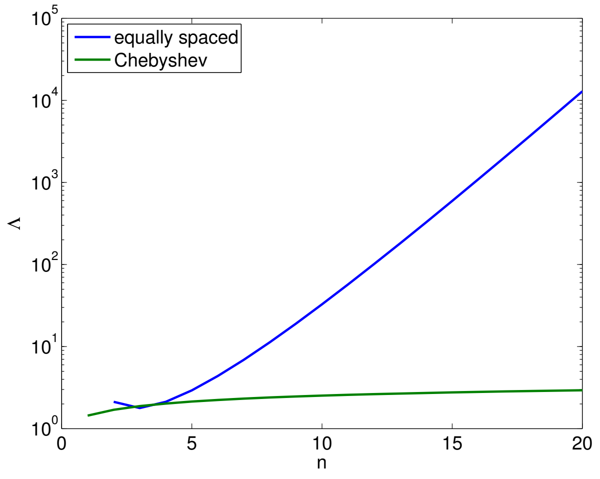


Figure 2.13: The approximate Lebesgue constants for equally-spaced and Chebyshev node distributions.

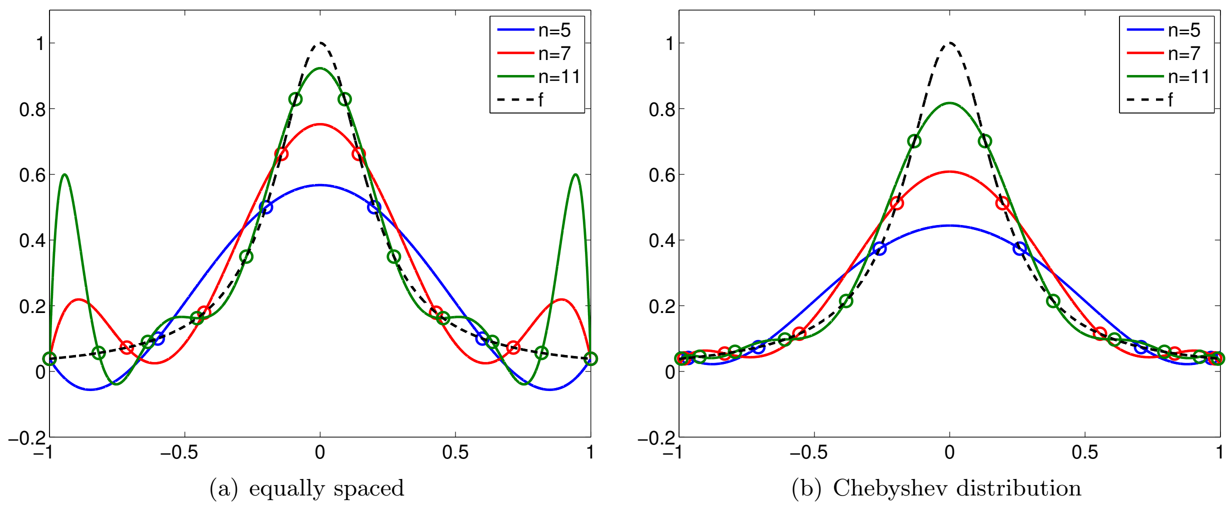


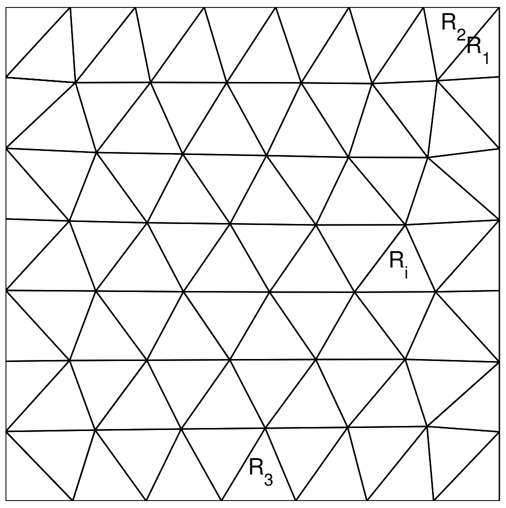
Figure 2.14: High-order interpolants for $f = 1/(x + 25x^2)$ over $[-1, 1]$.

Example 2.1.6 Runge's phenomenon
To demonstrate the instability of interpolants based on equally-spaced nodes, let us consider interpolation of

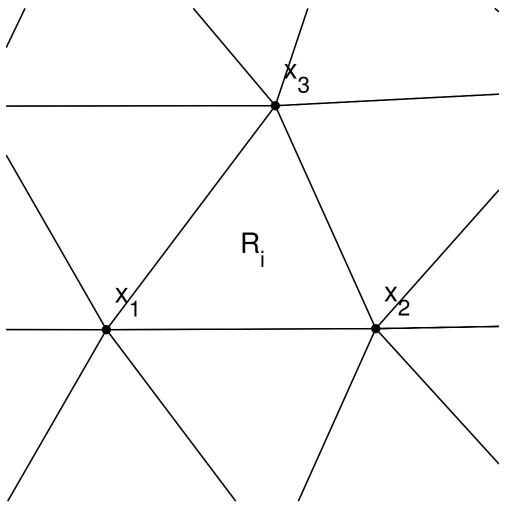
$$f(x) = \frac{1}{1 + 25x^2} \ .$$

The resulting interpolants for $p = 5$, 7, and 11 are shown in Figure 2.14. Note that equally-spaced nodes produce spurious oscillation near the end of the intervals. On the other hand, the clustering of the nodes toward the endpoints allow the Chebyshev node distribution to control the error in the region.

End Advanced Material



(a) mesh



(b) triangle R_i

Figure 2.15: Triangulation of a 2-D domain.

2.2 Interpolation of Bivariate Functions

This section considers interpolation of bivariate functions, i.e., functions of two variables. Following the approach taken in constructing interpolants for univariate functions, we first discretize the domain into smaller regions, namely triangles. The process of decomposing a domain, $D \subset \mathbb{R}^2$, into a set of non-overlapping triangles $\{R_i\}_{i=1}^N$ is called *triangulation*. An example of triangulation is shown in Figure 2.15. By construction, the triangles fill the domain in the sense that

$$D = \bigcup_{i=1}^{N} \overline{R}_i \ ,$$

where $\cup$ denotes the union of the triangles. The triangulation is characterized by the size h, which is the maximum diameter of the circumscribed circles for the triangles.

We will construct the interpolant over triangle R. Let us assume that we are given M interpolation points,

$$\bar{\boldsymbol{x}}^m = (\bar{x}^m, \bar{y}^m) \in R, \quad m = 1, \dots, M \ ,$$

and the function values evaluated at the interpolation points,

$$f(\bar{\boldsymbol{x}}^m), \quad m = 1, \dots, M \ .$$

Our objective is to construct the interpolant $\mathcal{I}f$ that approximates f at any point $\boldsymbol{x} \in R$,

$$\mathcal{I}f(\boldsymbol{x}) \approx f(\boldsymbol{x}), \quad \forall\, \boldsymbol{x} \in R \ ,$$

while matching the function value at the interpolations points,

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^m) = f(\bar{\boldsymbol{x}}^m), \quad m = 1, \dots, M \ .$$

As before, we assess the quality of the interpolant in terms of the error

$$e = \max_{x \in R} |f(\boldsymbol{x}) - (\mathcal{I}f)(\boldsymbol{x})| \ .$$

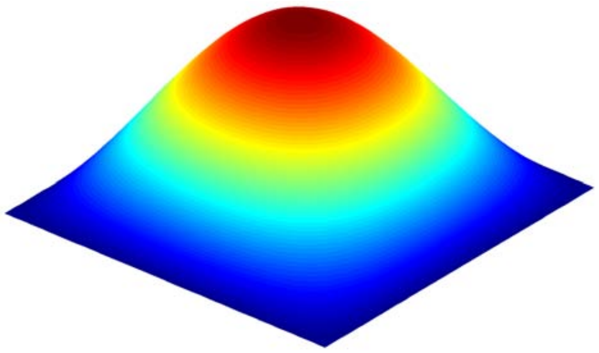


Figure 2.16: Function $f(x,y) = \sin(\pi x)\sin(\pi y)$

For the next two examples, we consider interpolation of bivariate function

$$f(x,y) = \sin(\pi x)\sin(\pi y), \quad (x,y) \in [0,1]^2 \ .$$

The function is shown in Figure 2.16.

Example 2.2.1 Piecewise-constant, centroid
The first interpolant approximates function f by a piecewise-constant function. To construct a constant function on R, we need just one interpolation point, i.e., $M = 1$. Let us choose the centroid of the triangle to be the interpolation point,

$$\bar{\boldsymbol{x}}^{\mathbf{1}} = \frac{1}{3}(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3) \ .$$

The constant interpolant is given by

$$\mathcal{I}f(\boldsymbol{x}) = f(\bar{\boldsymbol{x}}^1), \quad \forall\, \boldsymbol{x} \in R \ .$$

An example of piecewise-constant interpolant is shown in Figure 2.17(a). Note that the interpolant is discontinuous across the triangle interfaces in general.

The error in the interpolant is bounded by

$$e \leq h \max_{\boldsymbol{x} \in R} \|\nabla f(\boldsymbol{x})\|_2 \ ,$$

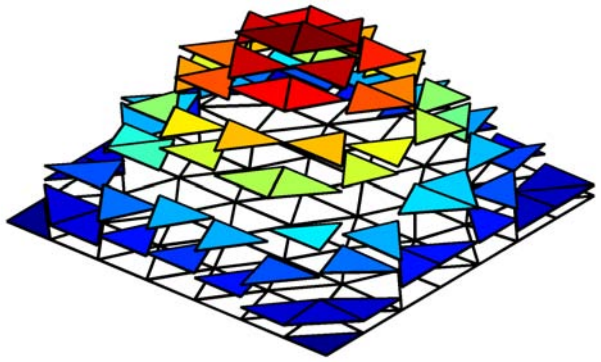
where $\|\nabla f(\boldsymbol{x})\|_2$ is the two-norm of the gradient, i.e.

$$\|\nabla f\|_2 = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \ .$$

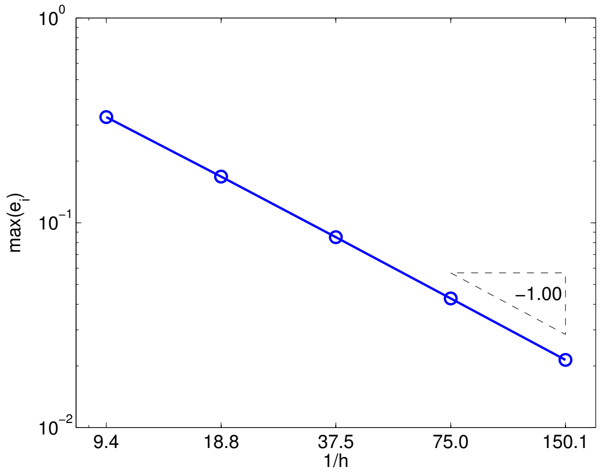
The interpolant is first-order accurate and is exact if f is constant. Figure 2.17(b) confirms the h^1 convergence of the error.

·

Example 2.2.2 Piecewise-linear, vertices
This interpolant approximates function f by a piecewise-linear function. Note that a linear function in two dimension is characterized by three parameters. Thus, to construct a linear function on a



(a) interpolant



(b) error

Figure 2.17: Piecewise-constant interpolation

triangular patch R, we need to choose three interpolation points, i.e., $M = 3$. Let us choose the vertices of the triangle to be the interpolation point,

$$\bar{\boldsymbol{x}}^1 = \boldsymbol{x}_1, \quad \bar{\boldsymbol{x}}^2 = \boldsymbol{x}_2, \quad \text{and} \quad \bar{\boldsymbol{x}}^3 = \boldsymbol{x}_3 \ .$$

The linear interpolant is of the form

$$(\mathcal{I}f)(\boldsymbol{x}) = a + bx + cy \ .$$

To find the three parameters, a, b, and c, we impose the constraint that the interpolant matches the function value at the three vertices. The constraint results in a system of three linear equations

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^1) = a + b\bar{x}^1 + c\bar{y}^1 = f(\bar{\boldsymbol{x}}^1) \ ,$$

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^2) = a + b\bar{x}^2 + c\bar{y}^2 = f(\bar{\boldsymbol{x}}^2) \ ,$$

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^3) = a + b\bar{x}^3 + c\bar{y}^3 = f(\bar{\boldsymbol{x}}^3) \ ,$$

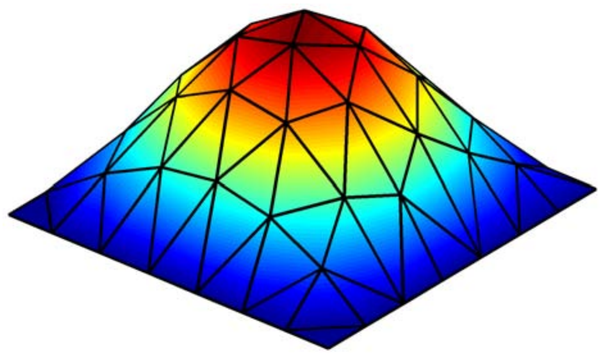
which can be also be written concisely in the matrix form

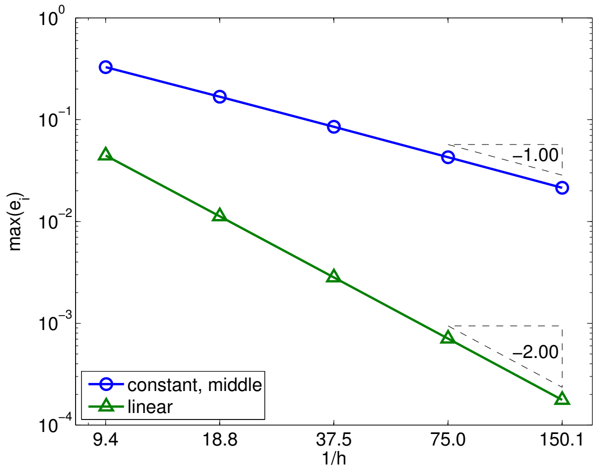
$$\begin{pmatrix} 1 & \bar{x}^1 & \bar{y}^1 \\ 1 & \bar{x}^2 & \bar{y}^2 \\ 1 & \bar{x}^3 & \bar{y}^3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} f(\bar{\boldsymbol{x}}^1) \\ f(\bar{\boldsymbol{x}}^2) \\ f(\bar{\boldsymbol{x}}^3) \end{pmatrix} .$$

The resulting interpolant is shown in Figure 2.18(a). Unlike the piecewise-constant interpolant, the piecewise-linear interpolant is continuous across the triangle interfaces.

The approach for constructing the linear interpolation requires solving a system of three linear equations. An alternative more efficient approach is to consider a different form of the interpolant. Namely, we consider the form

$$(\mathcal{I}f)(\boldsymbol{x}) = f(\bar{\boldsymbol{x}}^1) + b'(x - \bar{x}^1) + c'(y - \bar{y}^1) \ .$$


(a) interpolant



(b) error

Figure 2.18: Piecewise-linear interpolation

Note that the interpolant is still linear, but it already satisfies the interpolation condition at $(\bar{\boldsymbol{x}}^1, f(\bar{\boldsymbol{x}}^1))$ because

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^1) = f(\bar{\boldsymbol{x}}^1) + b'(\bar{x}^1 - \bar{x}^1) + c'(\bar{y}^1 - \bar{y}^1) = f(\bar{\boldsymbol{x}}^1) \; .$$

Thus, our task has been simplified to that of finding the two coefficients b' and c', as oppose to the three coefficients a, b, and c. We choose the two coefficients such that the interpolant satisfies the interpolaion condition at $\bar{\boldsymbol{x}}^2$ and $\bar{\boldsymbol{x}}^3$, i.e.

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^2) = f(\bar{\boldsymbol{x}}^1) + b'(\bar{x}^2 - \bar{x}^1) + c'(\bar{y}^2 - \bar{y}^1) = f(\bar{\boldsymbol{x}}^2) \; ,$$

$$(\mathcal{I}f)(\bar{\boldsymbol{x}}^3) = f(\bar{\boldsymbol{x}}^1) + b'(\bar{x}^3 - \bar{x}^1) + c'(\bar{y}^3 - \bar{y}^1) = f(\bar{\boldsymbol{x}}^3) \; .$$

Or, more compactly, we can write the equations in matrix form as

$$\begin{pmatrix} \bar{x}^2 - \bar{x}^1 & \bar{y}^2 - \bar{y}^1 \\ \bar{x}^3 - \bar{x}^1 & \bar{y}^3 - \bar{y}^1 \end{pmatrix} \begin{pmatrix} b' \\ c' \end{pmatrix} = \begin{pmatrix} f(\bar{\boldsymbol{x}}^2) - f(\bar{\boldsymbol{x}}^1) \\ f(\bar{\boldsymbol{x}}^3) - f(\bar{\boldsymbol{x}}^1) \end{pmatrix} .$$

With some arithmetics, we can find an explicit form of the coefficients,

$$b' = \frac{1}{A}\left[(f(\bar{\boldsymbol{x}}^2) - f(\bar{\boldsymbol{x}}^1))(\bar{y}^3 - \bar{y}^1) - (f(\bar{\boldsymbol{x}}^3) - f(\bar{\boldsymbol{x}}^1))(\bar{y}^2 - \bar{y}^1) \right] \; ,$$

$$c' = \frac{1}{A}\left[(f(\bar{\boldsymbol{x}}^3) - f(\bar{\boldsymbol{x}}^1))(\bar{x}^2 - \bar{x}^1) - (f(\bar{\boldsymbol{x}}^2) - f(\bar{\boldsymbol{x}}^1))(\bar{x}^3 - \bar{x}^1) \right] \; ,$$

with

$$A = (\bar{x}^2 - \bar{x}^1)(\bar{y}^3 - \bar{y}^1) - (\bar{x}^3 - \bar{x}^1)(\bar{y}^2 - \bar{y}^1) \; .$$

Note that A is twice the area of the triangle. It is important to note that this second form of the linear interpolant is identical to the first form; the interpolant is just expressed in a different form.

The error in the interpolant is governed by the Hessian of the function, i.e.

$$e \leq Ch^2 \|\nabla^2 f\|_F \ ,$$

where $\|\nabla^2 f\|_F$ is the Frobenius norm of the Hessian matrix, i.e.

$$\|\nabla^2 f\|_F = \sqrt{\left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2} \ .$$

Thus, the piecewise-linear interpolant is second-order accurate and is exact if f is linear. The convergence result shown in Figure 2.18(b) confirms the h^2 convergence of the error.

Chapter 3

Differentiation

3.1 Differentiation of Univariate Functions

Our objective is to approximate the value of the first derivative, f', for some arbitrary univariate function f. In particular, we assume the values of the function is provided at a set of uniformly spaced points[1] as shown in Figure 3.1. The spacing between any two function evaluation points is denoted by $\tilde{h}$.

Our approach to estimating the derivative is to approximate function f by its interpolant $\mathcal{I}f$ constructed from the sampled points and then differentiate the interpolant. Note that the interpolation rules based on piecewise-constant representation do not provide meaningful results, as they cannot represent nonzero derivatives. Thus, we will only consider linear and higher order interpolation rules.

To construct an interpolant $\mathcal{I}f$ in the neighborhood of $\tilde{x}_i$, we can first choose M interpolation points, $\tilde{x}_j$, $j = s(i), \dots, s(i) + M - 1$ in the neighborhood of $\tilde{x}_i$, where $s(i)$ is the global function evaluation index of the left most interpolation point. Then, we can construct an interpolant $\mathcal{I}f$ from the pairs $(\tilde{x}_j, \mathcal{I}f(\tilde{x}_j))$, $j = s(i), \dots, s(i) + M - 1$; note that $\mathcal{I}f$ depends linearly on the

[1]The uniform spacing is not necessary, but it simplifies the analysis

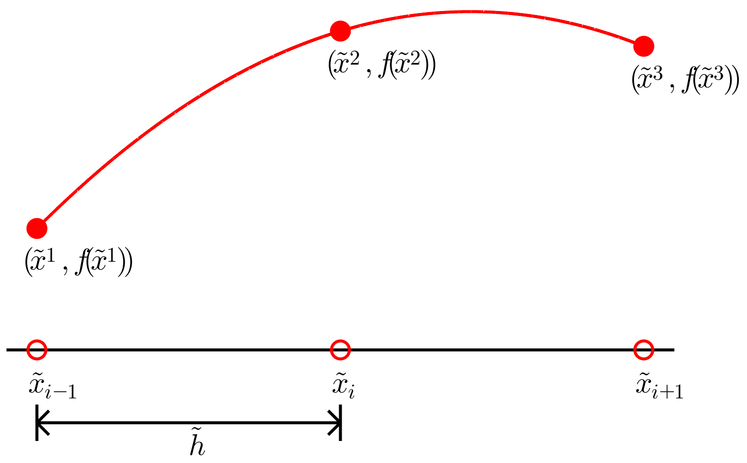

$(\tilde{x}^2, f(\tilde{x}^2))$
$(\tilde{x}^3, f(\tilde{x}^3))$
$(\tilde{x}^1, f(\tilde{x}^1))$
$\tilde{x}_{i-1}$
$\tilde{x}_i$
$\tilde{x}_{i+1}$
$\tilde{h}$

Figure 3.1: Stencil for one-dimensional numerical differentiation.

function values, as we know from the Lagrange basis construction. As a result, the derivative of the interpolant is also a linear function of $f(\tilde{x}_j)$, $j = s(i), \ldots, s(i) + M - 1$. Specifically, our numerical approximation to the derivative, $f_h'(\tilde{x}_i)$, is of the form

$$f_h'(\tilde{x}_i) \approx \sum_{j=s(i)}^{s(i)+M-1} \omega_j(i) f(\tilde{x}_j) \ ,$$

where $\omega_j(i)$, $j = 1, \ldots, M$, are weights that are dependent on the choice of interpolant.

These formulas for approximating the derivative are called *finite difference formulas*. In the context of numerical differentiation, the set of function evaluation points used to approximate the derivative at $\tilde{x}_i$ is called *numerical stencil*. A scheme requiring M points to approximate the derivative has an M-point stencil. The scheme is said to be *one-sided*, if the derivative estimate only involves the function values for either $\tilde{x} \geq \tilde{x}_i$ or $\tilde{x} \leq \tilde{x}_i$. The computational cost of numerical differentiation is related to the size of stencil, M.

Throughout this chapter, we assess the quality of finite difference formulas in terms of the error

$$e \equiv |f'(\tilde{x}_i) - f_h'(\tilde{x}_i)| \ .$$

Specifically, we are interested in analyzing the behavior of the error as our discretization is refined, i.e., as $\tilde{h}$ decreases. Note that, because the interpolant, $\mathcal{I}f$, from which $f_h'(\tilde{x}_i)$ is constructed approaches f as $\tilde{h} \to 0$ for smooth functions, we also expect $f_h'(\tilde{x}_i)$ to approach $f'(\tilde{x}_i)$ as $\tilde{h} \to 0$. Thus, our goal is not just to verify that $f'(\tilde{x}_i)$ approaches $f(\tilde{x}_i)$, but also to quantify how fast it converges to the true value.

Let us provide a few examples of the differentiation rules.

Example 3.1.1 forward difference

The first example is based on the linear interpolation. To estimate the derivative at $\tilde{x}_i$, let us first construct the linear interpolant over segment $[\tilde{x}_i, \tilde{x}_{i+1}]$. Substituting the interpolation points $\bar{x}^1 = \tilde{x}_i$ and $\bar{x}^2 = \tilde{x}_{i+1}$ into the expression for linear interpolant, Eq. (2.1), we obtain

$$(\mathcal{I}f)(x) = f(\tilde{x}_i) + \frac{1}{\tilde{h}}(f(\tilde{x}_{i+1}) - f(\tilde{x}_i))(x - \tilde{x}_i) \ .$$

The derivative of the interpolant evaluated at $x = \tilde{x}_i$ (approaching from $x > \tilde{x}_i$) is

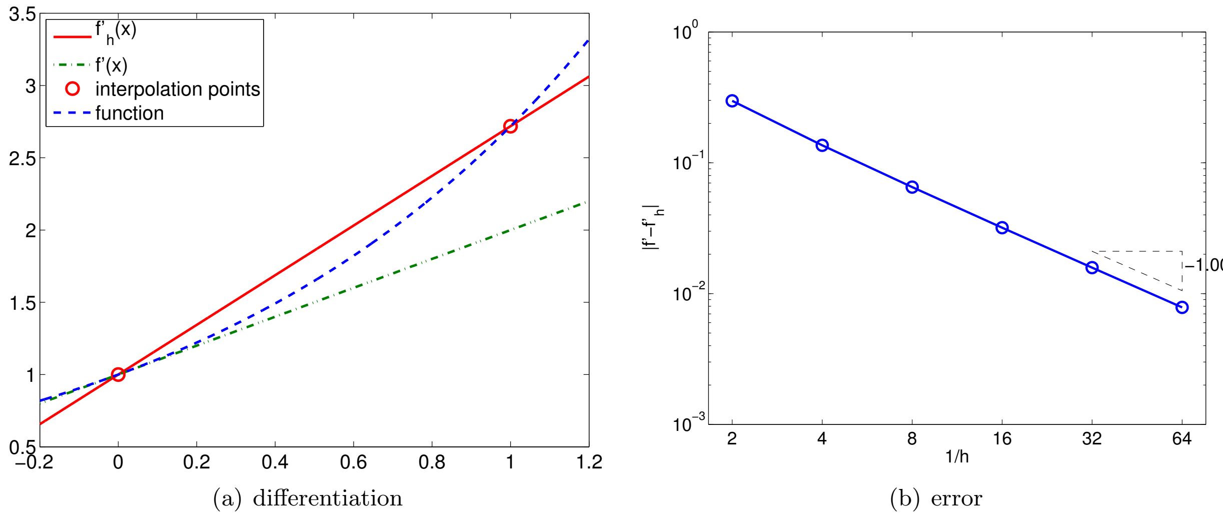
$$f_h'(\tilde{x}_i) = (\mathcal{I}f)'(\tilde{x}_i) = \frac{1}{\tilde{h}}(f(\tilde{x}_{i+1}) - f(\tilde{x}_i)) \ .$$

The forward difference scheme has a one-sided, 2-point stencil. The differentiation rule applied to $f(x) = \exp(x)$ about $x = 0$ is shown in Figure 3.2(a). Note that the linear interpolant matches the function at $\tilde{x}_i$ and $\tilde{x}_i + \tilde{h}$ and approximates derivative at $x = 0$.

The error in the derivative is bounded by

$$e_i = |f'(\tilde{x}_i) - f_h'(\tilde{x}_i)| \leq \frac{\tilde{h}}{2} \max_{x \in [\tilde{x}_i, \tilde{x}_{i+1}]} |f''(x)| \ .$$

The convergence plot in Figure 3.2(b) of the error with respect to h confirms the first-order convergence of the scheme.



(a) differentiation (b) error

Figure 3.2: Forward difference.

Proof. The proof of the error bound follows from Taylor expansion. Recall, assuming $f''(x)$ is bounded in $[\tilde{x}_i, \tilde{x}_{i+1}]$,

$$f(\tilde{x}_{i+1}) = f(\tilde{x}_i + \tilde{h}) = f(\tilde{x}_i) + f'(\tilde{x}_i)\tilde{h} + \frac{1}{2} f''(\xi)\tilde{h}^2 \ ,$$

for some $\xi \in [\tilde{x}_i, \tilde{x}_{i+1}]$. The derivative of the interpolant evaluated at $x = \tilde{x}_i$ can be expressed as

$$(\mathcal{I}f)'(\tilde{x}_i) = \frac{1}{\tilde{h}} \left(f(\tilde{x}_i) + f'(\tilde{x}_i)\tilde{h} + \frac{1}{2} f''(\xi)\tilde{h}^2 - f(\tilde{x}_i) \right) = f'(\tilde{x}_i) + \frac{1}{2} f''(\xi)\tilde{h} \ ,$$

and the error in the derivative is

$$|f'(\tilde{x}_i) - (\mathcal{I}f)'(\tilde{x}_i)| = \left| \frac{1}{2} f''(\xi)\tilde{h} \right| \le \frac{1}{2}\tilde{h} \max_{x \in [\tilde{x}_i, \tilde{x}_{i+1}]} |f''(x)| \ .$$

□

·

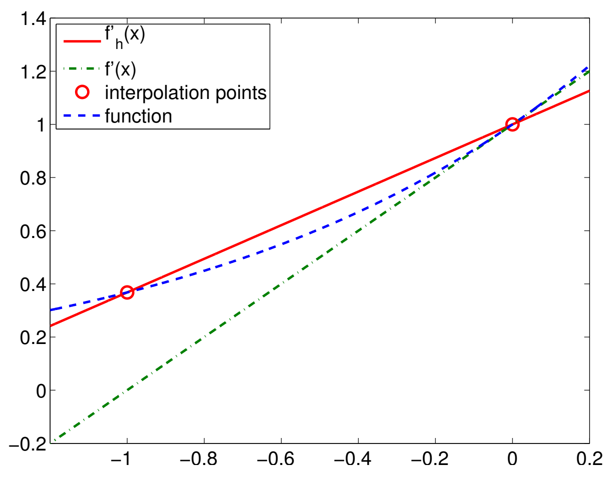
Example 3.1.2 backward difference

The second example is also based on the piecewise linear interpolation; however, instead of constructing the interpolant over segment $[\tilde{x}_i, \tilde{x}_{i+1}]$, we construct the interpolant over segment $[\tilde{x}_{i-1}, \tilde{x}_i]$. Substituting the interpolation points $\bar{x}^1 = \tilde{x}_{i-1}$ and $\bar{x}^2 = \tilde{x}_i$ into the linear interpolant expression, Eq. (2.1), we obtain

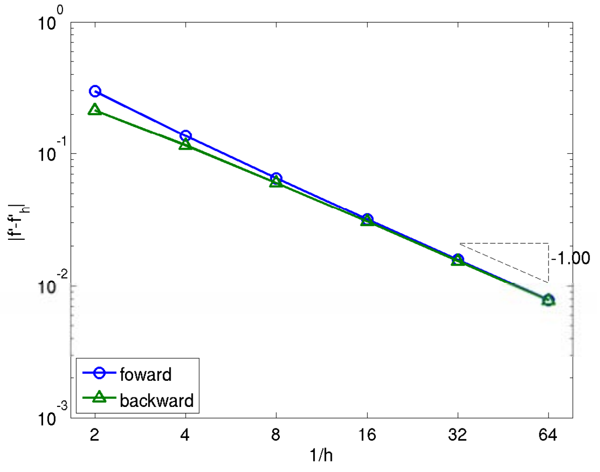
$$(\mathcal{I}f)(x) = f(\tilde{x}_{i-1}) + \frac{1}{\tilde{h}}(f(\tilde{x}_i) - f(\tilde{x}_{i-1}))(x - \tilde{x}_{i-1}) \ .$$

The derivative of the interpolant evaluated at $x = \tilde{x}_i$ (approaching from $x < \tilde{x}_i$) is

$$f'_h(\tilde{x}_i) = (\mathcal{I}f)'(\tilde{x}_i) = \frac{1}{\tilde{h}}(f(\tilde{x}_i) - f(\tilde{x}_{i-1})) \ .$$



(a) differentiation



(b) error

Figure 3.3: Backward difference.

The backward difference scheme has a one-sided, 2-point stencil. The differentiation rule applied to $f(x) = \exp(x)$ about $x = 0$ is shown in Figure 3.3(a). The construction is similar to that of the forward difference, except that the interpolant matches the function at $\tilde{x}_i - \tilde{h}$ and $\tilde{x}_i$.

The error in the derivative is bounded by

$$e_i = |f'(\tilde{x}_i) - f'_h(\tilde{x}_i)| \le \frac{\tilde{h}}{2} \max_{x \in [\tilde{x}_{i-1}, \tilde{x}_i]} |f''(x)| \ .$$

The proof is similar to the proof for the error bound of the forward difference formula. The convergence plot for $f(x) = \exp(x)$ is shown in Figure 3.3(b).

·

Example 3.1.3 centered difference

To develop a more accurate estimate of the derivative at $\tilde{x}_i$, let us construct a quadratic interpolant over segment $[\tilde{x}_{i-1}, \tilde{x}_{i+1}]$ using the function values at $\tilde{x}_{i-1}$, $\tilde{x}_i$, and $\tilde{x}_{i+1}$, and then differentiate the interpolant. To form the interpolant, we first construct the quadratic Lagrange basis functions on $[\tilde{x}_{i-1}, \tilde{x}_{i+1}]$ using the interpolation points $\bar{x}^1 = \tilde{x}_{i-1}$, $\bar{x}^2 = \tilde{x}_i$, and $\bar{x}^3 = \tilde{x}_{i+1}$, i.e.

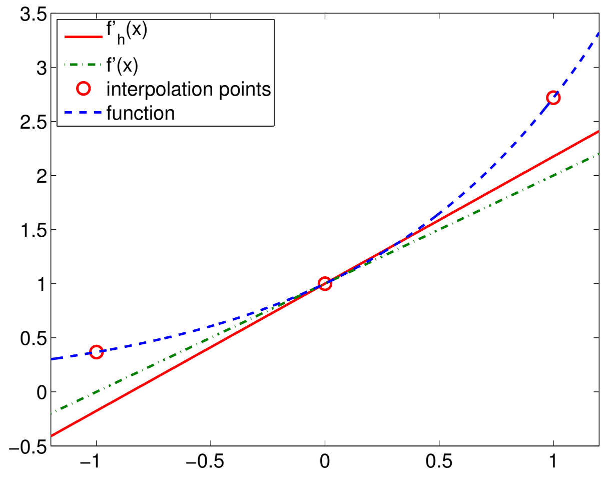
$$\phi_1(x) = \frac{(x - \bar{x}^2)(x - \bar{x}^3)}{(\bar{x}^1 - \bar{x}^2)(\bar{x}^1 - \bar{x}^3)} = \frac{(x - \tilde{x}_i)(x - \tilde{x}_{i+1})}{(\tilde{x}_{i-1} - \tilde{x}_i)(\tilde{x}_{i-1} - \tilde{x}_{i+1})} = \frac{1}{2\tilde{h}^2}(x - \tilde{x}_i)(x - \tilde{x}_{i+1}) \ ,$$

$$\phi_2(x) = \frac{(x - \bar{x}^1)(x - \bar{x}^3)}{(\bar{x}^2 - \bar{x}^1)(\bar{x}^2 - \bar{x}^3)} = \frac{(x - \tilde{x}_{i-1})(x - \tilde{x}_{i+1})}{(\tilde{x}_i - \tilde{x}_{i-1})(\tilde{x}_i - \tilde{x}_{i+1})} = -\frac{1}{\tilde{h}^2}(x - \tilde{x}_i)(x - \tilde{x}_{i+1}) \ ,$$

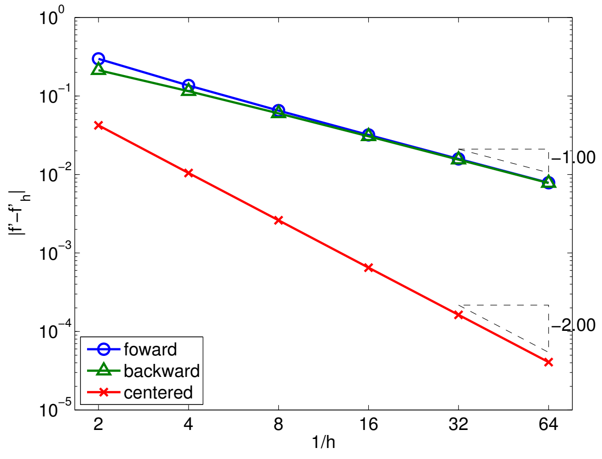
$$\phi_3(x) = \frac{(x - \bar{x}^1)(x - \bar{x}^2)}{(\bar{x}^3 - \bar{x}^1)(\bar{x}^3 - \bar{x}^2)} = \frac{(x - \tilde{x}_{i-1})(x - \tilde{x}_i)}{(\tilde{x}_{i+1} - \tilde{x}_{i-1})(\tilde{x}_{i+1} - \tilde{x}_i)} = \frac{1}{2\tilde{h}^2}(x - \tilde{x}_{i-1})(x - \tilde{x}_i) \ ,$$

where $\tilde{h} = \tilde{x}_{i+1} - \tilde{x}_i = \tilde{x}_i - \tilde{x}_{i-1}$.[2] Substitution of the basis functions into the expression for a

[2]Note that, for the quadratic interpolant, $\tilde{h}$ is half of the h defined in the previous chapter based on the length of the segment.



(a) differentiation



(b) error

Figure 3.4: Centered difference.

quadratic interpolant, Eq. (2.4), yields

$$
\begin{aligned}
(\mathcal{I}f)(x) &= f(\bar{x}^1)\phi_1(x) + f(\bar{x}^2)\phi_2(x) + f(\bar{x}^3)\phi_3(x) \\
&= \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i-1})(x-\tilde{x}_i)(x-\tilde{x}_{i+1}) - \frac{1}{\tilde{h}^2} f(\tilde{x}_i)(x-\tilde{x}_{i-1})(x-\tilde{x}_{i+1}) \\
&\quad + \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i+1})(x-\tilde{x}_{i-1})(x-\tilde{x}_i) \ .
\end{aligned}
$$

Differentiation of the interpolant yields

$$
(\mathcal{I}f)'(x) = \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i-1})(2x-\tilde{x}_i-\tilde{x}_{i+1}) - \frac{1}{\tilde{h}^2} f(\tilde{x}_i)(2x-\tilde{x}_{i-1}-\tilde{x}_{i+1}) + \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i+1})(2x-\tilde{x}_{i-1}-\tilde{x}_i) \ .
$$

Evaluating the interpolant at $x = \tilde{x}_i$, we obtain

$$
\begin{aligned}
f_h'(\tilde{x}_i) = (\mathcal{I}f)'(\tilde{x}_i) &= \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i-1})(\tilde{x}_i-\tilde{x}_{i+1}) + \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i+1})(\tilde{x}_i-\tilde{x}_{i-1}) \\
&= \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i-1})(-\tilde{h}) + \frac{1}{2\tilde{h}^2} f(\tilde{x}_{i+1})(\tilde{h}) \\
&= \frac{1}{2\tilde{h}} (f(\tilde{x}_{i+1}) - f(\tilde{x}_{i-1})) \ .
\end{aligned}
$$

Note that even though the quadratic interpolant is constructed from the three interpolation points, only two of the three points are used in estimating the derivative. The approximation procedure is illustrated in Figure 3.4(a).

The error bound for the centered difference is given by

$$
e_i = |f(\tilde{x}_i) - f_h'(\tilde{x}_i)| \le \frac{\tilde{h}^2}{6} \max_{x\in[\tilde{x}_{i-1},\tilde{x}_{i+1}]} |f'''(x)| \ .
$$

The centered difference formula is *second-order accurate*, as confirmed by the convergence plot in Figure 3.4(b).

Proof. The proof of the error bound follows from Taylor expansion. Recall, assuming $f'''(x)$ is bounded in $[\tilde{x}_{i-1}, \tilde{x}_{i+1}]$,

$$f(\tilde{x}_{i+1}) = f(\tilde{x}_i + \tilde{h}) = f(\tilde{x}_i) + f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 + \frac{1}{6}f'''(\xi^+)\tilde{h}^3 \ ,$$

$$f(\tilde{x}_{i-1}) = f(\tilde{x}_i - \tilde{h}) = f(\tilde{x}_i) - f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 - \frac{1}{6}f'''(\xi^-)\tilde{h}^3 \ ,$$

for some $\xi^+ \in [\tilde{x}_i, \tilde{x}_{i+1}]$ and $\xi^- \in [\tilde{x}_{i-1}, \tilde{x}_i]$. The centered difference formula gives

$$\begin{aligned}
(\mathcal{I}f)'(\tilde{x}_i) &= \frac{1}{2\tilde{h}}(f(x_{i+1}) - f(\tilde{x}_{i-1})) \\
&= \frac{1}{2\tilde{h}}\left[\left(f(\tilde{x}_i) + f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 + \frac{1}{6}f'''(\xi^+)\tilde{h}^3\right)\right. \\
&\qquad \left. - \left(f(\tilde{x}_i) - f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 - \frac{1}{6}f'''(\xi^-)\tilde{h}^3\right)\right] \\
&= f'(\tilde{x}_i) + \frac{1}{12}\tilde{h}^2\left(f'''(\xi^+) - f'''(\xi^-)\right) \ .
\end{aligned}$$

The error in the derivative estimate is

$$\left|f'(\tilde{x}_i) - (\mathcal{I}f)'(\tilde{x}_i)\right| = \left|\frac{1}{12}\tilde{h}^2\left(f'''(\xi^+) - f'''(\xi^-)\right)\right| = \frac{1}{6}\tilde{h}^2 \max_{x\in[\tilde{x}_{i-1},\tilde{x}_{i+1}]} |f'''(x)| \ .$$

□

Using a higher-order interpolation scheme, we can develop higher-order accurate numerical differentiation rules. However, the numerical stencil extends with the approximation order, because the number of interpolation points increases with the interpolation order.

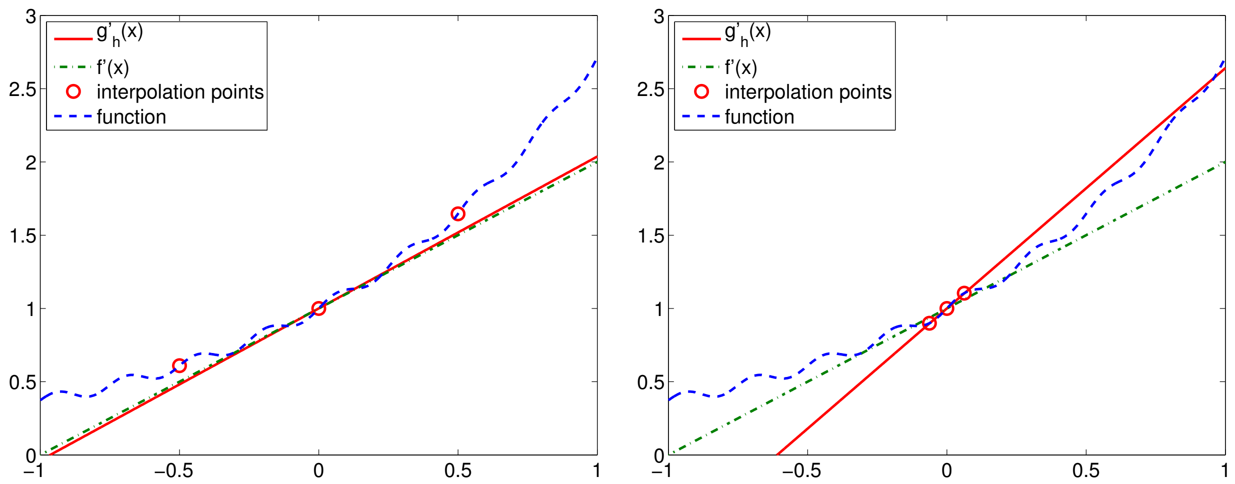
·

We would also like to make some remarks about how noise affects the quality of numerical differentiation. Let us consider approximating a derivative of $f(x) = \exp(x)$ at $x = 0$. However, assume that we do not have access to f itself, but rather a function f with some small noise added to it. In particular, let us consider

$$g(x) = f(x) + \epsilon\sin(kx) \ ,$$

with $\epsilon = 0.04$ and $k = 1/\epsilon$. We can think of $\epsilon\sin(kx)$ as noise added, for example, in a measuring process. Considering $f(0) = 1$, this is a relatively small noise in terms of amplitude.

The result of applying the finite difference formulas to g in an attempt to approximate $f'(0)$ is shown in Figure 3.5. Comparing the approximations obtained for $\tilde{h} = 1/2$ and $1/16$, we see that the approximation in fact gets worse as $\tilde{h}$ is refined. Figure 3.6 confirms that all numerical differentiation formulas considered in this section fail to converge. In fact, they all asymptotically commit $\mathcal{O}(1)$ error as $\tilde{h} \to 0$, even though the error decreases to less than 10^{-2} for a certain choice of $\tilde{h}$.



(a) $\tilde{h} = 1/2$ (b) $\tilde{h} = 1/16$

Figure 3.5: The centered difference formula applied to a noisy function.

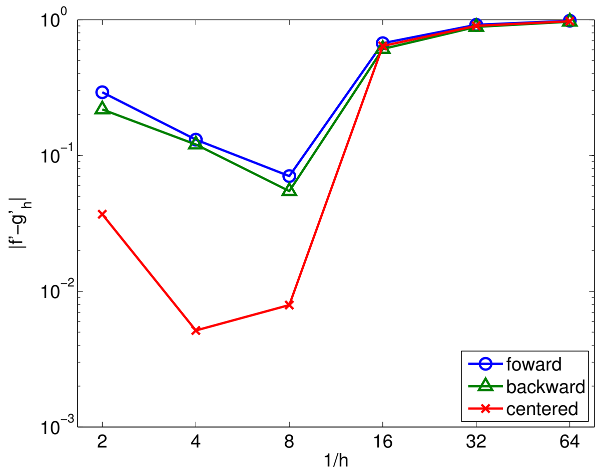


Figure 3.6: Convergence of the numerical differentiation formula applied to a noisy function.

As essentially any data taken in real life is inherently noisy, we must be careful when differentiating the data. For example, let us say that our objective is to estimate the acceleration of an object by differentiating a velocity measurement. Due to the presence of measurement noise, we in general cannot expect the quality of our acceleration estimate to improve as we improve the sampling rate and decreasing the discretization scale, $\tilde{h}$.

One strategy to effectively differentiate a noisy data is to first filter the noisy data. Filtering is a technique to clean a signal by removing frequency content above a certain frequency[3] For example, if a user is only interested in estimating the behavior of a signal below a certain frequency, all content above that threshold can be deemed noise. The cleaned data is essentially smooth with respect to the scale of interest, and thus can be safely differentiated. Another alternative, discussed in Unit III, is to first fit a smooth function to many data points and then differentiate this smooth fit.

3.1.1 Second Derivatives

Following the same interpolation-based template, we can develop a numerical approximation to higher-order derivatives. In general, to estimate the p^{th}-derivative, we must use an interpolation rule based on p^{th}- or higher-degree polynomial reconstruction. As an example, we demonstrate how to estimate the second derivative from a quadratic interpolation.

Example 3.1.4 second-order centered difference
We can use the quadratic interpolant considered in the previous case to estimate the second derivative of the function. Again, choosing $\bar{x}^1 = \tilde{x}_{i-1}$, $\bar{x}^2 = \tilde{x}_i$, $\bar{x}^3 = \tilde{x}_{i+1}$ as the interpolation points, the quadratic reconstruction is given by

$$(\mathcal{I}f)(x) = f(\tilde{x}_{i-1})\phi_1(x) + f(\tilde{x}_i)\phi_2(x) + f(\tilde{x}_{i+1})\phi_3(x) \ ,$$

where the Lagrange basis function are given by

$$\phi_1(x) = \frac{(x-\tilde{x}_i)(x-\tilde{x}_{i+1})}{(\tilde{x}_{i-1}-\tilde{x}_i)(\tilde{x}_{i-1}-\tilde{x}_{i+1})} \ ,$$

$$\phi_2(x) = \frac{(x-\tilde{x}_{i-1})(x-\tilde{x}_{i+1})}{(\tilde{x}_i-\tilde{x}_{i-1})(\tilde{x}_i-\tilde{x}_{i+1})} \ ,$$

$$\phi_3(x) = \frac{(x-\tilde{x}_{i-1})(x-\tilde{x}_i)}{(\tilde{x}_{i+1}-\tilde{x}_{i-1})(\tilde{x}_{i+1}-\tilde{x}_i)} \ .$$

Computing the second derivative of the quadratic interpolant can be proceeded as

$$(\mathcal{I}f)''(x) = f(\tilde{x}_{i-1})\phi_1''(x) + f(\tilde{x}_i)\phi_2''(x) + f(\tilde{x}_{i+1})\phi_3''(x) \ .$$

In particular, note that once the second derivatives of the Lagrange basis are evaluated, we can express the second derivative of the interpolant as a sum of the functions evaluated at three points.

[3]Sometimes signals below a certain frequency is filtered to eliminate the bias.

The derivatives of the Lagrange basis are given by

$$\phi_1''(x) = \frac{2}{(\tilde{x}_{i-1} - \tilde{x}_i)(\tilde{x}_{i-1} - \tilde{x}_{i+1})} = \frac{2}{(-\tilde{h})(-2\tilde{h})} = \frac{1}{\tilde{h}^2} \ ,$$

$$\phi_2''(x) = \frac{2}{(\tilde{x}_i - \tilde{x}_{i-1})(\tilde{x}_i - \tilde{x}_{i+1})} = \frac{2}{(\tilde{h})(-\tilde{h})} = -\frac{2}{\tilde{h}^2} \ ,$$

$$\phi_3''(x) = \frac{2}{(\tilde{x}_{i+1} - \tilde{x}_{i-1})(\tilde{x}_{i+1} - \tilde{x}_i)} = \frac{2}{(2\tilde{h})(\tilde{h})} = \frac{1}{\tilde{h}^2} \ .$$

Substitution of the derivatives to the second derivative of the quadratic interpolant yields

$$\begin{aligned}(\mathcal{I}f)''(\tilde{x}_i) &= f(\tilde{x}_{i-1})\left(\frac{1}{\tilde{h}^2}\right) + f(\tilde{x}_i)\left(\frac{-2}{\tilde{h}^2}\right) + f(\tilde{x}_{i+1})\left(\frac{1}{\tilde{h}^2}\right) \\ &= \frac{1}{\tilde{h}^2}\left(f(\tilde{x}_{i-1}) - 2f(\tilde{x}_i) + f(\tilde{x}_{i+1})\right).\end{aligned}$$

The error in the second-derivative approximation is bounded by

$$e_i \equiv |f''(\tilde{x}_i) - (\mathcal{I}f)''(\tilde{x}_i)| \le \frac{\tilde{h}^2}{12} \max_{x \in [\tilde{x}_{i-1}, \tilde{x}_{i+1}]} |f^{(4)}(x)| \ .$$

Thus, the scheme is *second-order accurate*.

Proof. The proof of the error bound again follows from Taylor expansion. Recall, assuming $f^{(4)}(x)$ is bounded in $[\tilde{x}_{i-1}, \tilde{x}_{i+1}]$,

$$f(\tilde{x}_{i+1}) = f(\tilde{x}_i + \tilde{h}) = f(\tilde{x}_i) + f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 + \frac{1}{6}f'''(\tilde{x}_i)\tilde{h}^3 + \frac{1}{24}f^{(4)}(\xi^+)\tilde{h}^4 \ ,$$

$$f(\tilde{x}_{i-1}) = f(\tilde{x}_i - \tilde{h}) = f(\tilde{x}_i) - f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 - \frac{1}{6}f'''(\tilde{x}_i)\tilde{h}^3 + \frac{1}{24}f^{(4)}(\xi^-)\tilde{h}^4 \ .$$

The second derivative estimation gives

$$\begin{aligned}(\mathcal{I}f)''(\tilde{x}_i) = \frac{1}{\tilde{h}^2}\Bigg[&\left(f(\tilde{x}_i) + f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 + \frac{1}{6}f'''(\tilde{x}_i)\tilde{h}^3 + \frac{1}{24}f^{(4)}(\xi^+)\tilde{h}^4\right) \\ &- 2f(\tilde{x}_i) \\ &+ \left(f(\tilde{x}_i) - f'(\tilde{x}_i)\tilde{h} + \frac{1}{2}f''(\tilde{x}_i)\tilde{h}^2 - \frac{1}{6}f'''(\tilde{x}_i)\tilde{h}^3 + \frac{1}{24}f^{(4)}(\xi^-)\tilde{h}^4\right)\Bigg] \\ = f''(\tilde{x}_i) + \frac{1}{24}\tilde{h}^2&\left(f^{(4)}(\xi^+) + f^{(4)}(\xi^-)\right).\end{aligned}$$

The error in the second derivative is

$$|f''(\tilde{x}_i) - (\mathcal{I}f)''(\tilde{x}_i)| = \left|\frac{1}{24}\tilde{h}^2\left(f^{(4)}(\xi^+) + f^{(4)}(\xi^-)\right)\right| \le \frac{1}{12}\tilde{h}^2 \max_{x \in [\tilde{x}_{i-1}, \tilde{x}_{i+1}]} |f^{(4)}(x)| \ .$$

□

Let us demonstrate that the second-order derivative formula works for constant, linear, and quadratic function. First, we consider $f(x) = c$. Clearly, the second derivative is $f''(x) = 0$. Using the approximation formula, we obtain

$$(\mathcal{I}f)''(\tilde{x}_i) = \frac{1}{\tilde{h}^2}\big(f(\tilde{x}_{i-1}) - 2f(\tilde{x}_i) + f(\tilde{x}_{i+1})\big) = \frac{1}{\tilde{h}^2}(c - 2c + c) = 0 \ .$$

Thus, the approximation provides the exact second derivative for the constant function. This is not surprising, as the error is bounded by the fourth derivative of f, and the fourth derivative of the constant function is zero.

Second, we consider $f(x) = bx+c$. The second derivative is again $f''(x) = 0$. The approximation formula gives

$$(\mathcal{I}f)''(\tilde{x}_i) = \frac{1}{\tilde{h}^2}\big(f(\tilde{x}_{i-1}) - 2f(\tilde{x}_i) + f(\tilde{x}_{i+1})\big) = \frac{1}{\tilde{h}^2}[(b\tilde{x}_{i-1} + c) - 2(b\tilde{x}_i + c) + (b\tilde{x}_{i+1} + c)]$$

$$= \frac{1}{\tilde{h}^2}[(b(\tilde{x}_i - \tilde{h}) + c) - 2(b\tilde{x}_i + c) + (b(\tilde{x}_i + \tilde{h}) + c)] = 0 \ .$$

Thus, the approximation also works correctly for a linear function.

Finally, let us consider $f(x) = ax^2 + bx + c$. The second derivative for this case is $f''(x) = 2a$. The approximation formula gives

$$(\mathcal{I}f)''(\tilde{x}_i) = \frac{1}{\tilde{h}^2}[(a\tilde{x}_{i-1}^2 + b\tilde{x}_{i-1} + c) - 2(a\tilde{x}_i^2 + b\tilde{x}_i + c) + (a\tilde{x}_{i+1}^2 + b\tilde{x}_{i+1} + c)]$$

$$= \frac{1}{\tilde{h}^2}[(a(\tilde{x}_i - \tilde{h})^2 + b(\tilde{x}_i - \tilde{h}) + c) - 2(a\tilde{x}_i^2 + b\tilde{x}_i + c) + (a(\tilde{x}_i + \tilde{h})^2 + b(\tilde{x}_i + \tilde{h}) + c)]$$

$$= \frac{1}{\tilde{h}^2}\left[a(\tilde{x}_i^2 - 2\tilde{h}\tilde{x}_i + \tilde{h}^2) - 2a\tilde{x}_i^2 + a(\tilde{x}_i^2 + 2\tilde{h}\tilde{x}_i + \tilde{h}^2)\right]$$

$$= \frac{1}{\tilde{h}^2}\left[2a\tilde{h}^2\right] = 2a \ .$$

Thus, the formula also yields the exact derivative for the quadratic function.

·

The numerical differentiation rules covered in this section form the basis for the finite difference method — a framework for numerically approximating the solution to differential equations. In the framework, the infinite-dimensional solution on a domain is approximated by a finite number of nodal values on a discretization of the domain. This allows us to approximate the solution to complex differential equations — particularly partial differential equations — that do not have closed form solutions. We will study in detail these numerical methods for differential equations in Unit IV, and we will revisit the differential rules covered in this section at the time.

We briefly note another application of our finite difference formulas: they may be used (say) to approximately evaluate our (say) interpolation *error bounds* to provide an *a posteriori* estimate for the error.

3.2 Differentiation of Bivariate Functions

Let us briefly consider differentiation of bivariate functions. For simplicity, we restrict ourselves to estimation of the first derivative, i.e., the gradient.

Example 3.2.1 first-order gradient
Following the procedure for estimating the first derivative of univariate functions, we first construct a polynomial interpolant and then evaluate its derivative. In particular, we consider a linear interpolant on a triangle. Recall the second form of linear interpolant,

$$(\mathcal{I}f)(\boldsymbol{x}) = f(\bar{\boldsymbol{x}}^{\mathbf{1}}) + b'(x - \bar{x}^1) + c'(y - \bar{y}^1) \ .$$

The partial derivative in the x-direction is

$$\frac{\partial(\mathcal{I}f)}{\partial x} = b' = \frac{1}{A}\Big[(f(\bar{\boldsymbol{x}}^2) - f(\bar{\boldsymbol{x}}^1))(\bar{y}^3 - \bar{y}^1) - (f(\bar{\boldsymbol{x}}^3) - f(\bar{\boldsymbol{x}}^1))(\bar{y}^2 - \bar{y}^1)\Big] \ ,$$

where we recall that A is twice the area of the triangle, i.e., $A = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$. Similarly, the derivative in the y-direction is

$$\frac{\partial(\mathcal{I}f)}{\partial y} = c' = \frac{1}{A}\Big[(f(\bar{\boldsymbol{x}}^3) - f(\bar{\boldsymbol{x}}^1))(\bar{x}^2 - \bar{x}^1) - (f(\bar{\boldsymbol{x}}^2) - f(\bar{\boldsymbol{x}}^1))(\bar{x}^3 - \bar{x}^1)\Big] \ .$$

In general, a directional derivative in the direction $\boldsymbol{s} = (s_x, s_y)$ is

$$\frac{\partial(\mathcal{I}f)}{\partial \boldsymbol{s}} = s_x \frac{\partial(\mathcal{I}f)}{\partial x} + s_y \frac{\partial(\mathcal{I}f)}{\partial y} = s_x b' + s_y c' \ .$$

Because the gradient approximation is constructed from a linear function, the gradient estimate is constant over the triangle. The approximation is first-order accurate.

Chapter 4

Elements of a Program and Matlab Basics

4.1 Computer Architecture and Computer Programming

4.1.1 Virtual Processor

It is convenient when programming a computer to have a mental model of the underlying architecture: the components or "units," functions, and interconnections which ultimately implement the program. It is not important that the mental model directly correspond to the actual hardware of any real computer. But it is important that the mental model lead to the right decisions as regards how to write correct and efficient programs.

We show in Figure 4.1 the architecture for a "virtual computer." There are many ways in which we can extend or refine our model to more accurately represent either particular processes or particular computers of interest: we might break down each unit into smaller units — for example, a memory hierarchy; we might consider additional, special-purpose, units — for example for graphics and visualization; and we might replicate our virtual computer many times to represent (a system of) many interacting processes. We emphasize that if you put a screwdriver to computer case, you would not find a one-to-one mapping from our virtual entities to corresponding hardware elements. But you might come close.

We now describe the different elements. On the far left we indicate the memory. This memory is often hierarchical: faster and more expensive memory in a "cache"; slower and much more extensive "RAM." (Note also there will be archival memory outside the processor accessed through I/O functions, as described shortly.) Next there is the arithmetic unit which performs the various "basic" operations on data — for example, assignment, addition, multiplication, and comparison. (The adjective "arithmetic" for this unit is overly restrictive, but since in this course we are primarily focused on numerical methods the key operations are indeed arithmetic.) We also identify the "instruction stack" which contains the instructions necessary to implement the desired program. And finally there is the I/O (Input/Output) unit which controls communication between our processor and external devices and other processes; the latter may take the form of files on archival media (such as disks), keyboards and other user input devices, sensors and actuators (say, on a robot), and displays.

We note that all the components — memory, arithmetic unit, instruction stack, I/O unit —

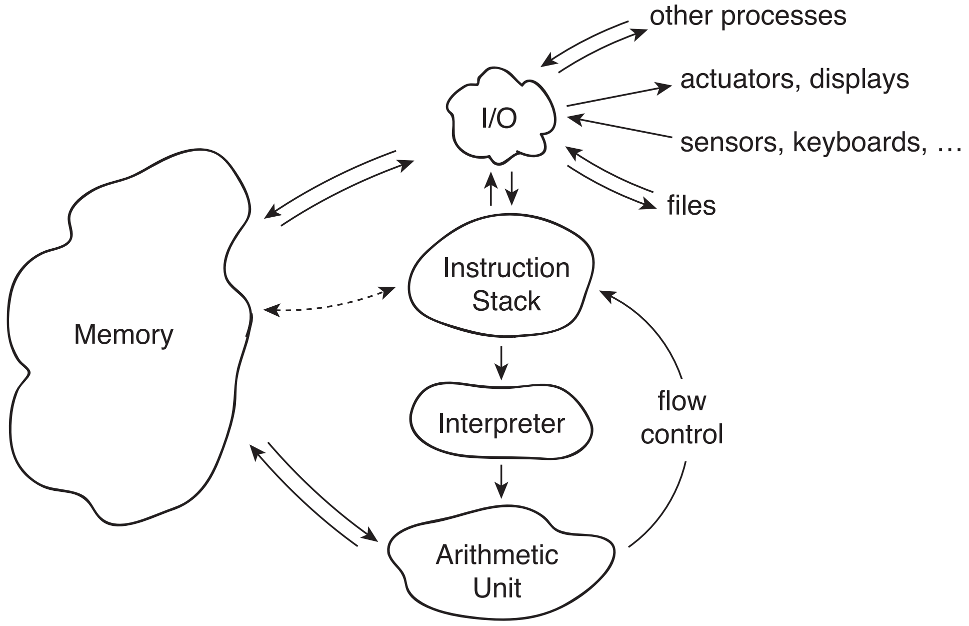


Figure 4.1: Virtual Processor

are interconnected through buses which shuttle the necessary data between the different elements. The arithmetic unit may receive instructions from the instruction stack and read and write data from memory; similarly, the instruction stack may ask the I/O unit to read data from a file to the memory unit or say write data from memory to a display. And the arithmetic unit may effectively communicate directly with the instruction stack to control the flow of the program. These data buses are of course a model for wires (or optical communication paths) in an actual hardware computer.

The "I" in Figure 4.1 stands for interpreter. The Interpreter takes a line or lines of a program written in a high-level programming or "scripting" language — such as Matlab or Python — from the instruction stack, translates these instructions into machine code, and then passes these now machine-actionable directives to the arithmetic unit for execution. (Some languages, such as C, are not interpreted but rather compiled: the program is translated *en masse* into machine code prior to execution. As you might imagine, compiled codes will typically execute faster than interpreted codes.)

There are typically two ways in which an interpreter can feed the arithmetic unit. The first way, more interactive with the user, is "command-line" mode: here a user enters each line, or small batch of lines, from the keyboard to the I/O unit; the I/O unit in turn passes these lines to the interpreter for processing. The second way, much more convenient for anything but the simplest of tasks, is "script" mode: the I/O unit transfers the *entire* program from a file prepared by the user to the instruction stack; the program is then executed proceeding from the first line to the last. (The latter is in fact a simplification, as we shall see when we discuss flow control and functions.) Script mode permits much faster execution, since the user is out of the loop; script mode also permits much faster development/adaptation, since the user can re-use the same script many times — only changing data, or perhaps making incremental corrections/improvements or modifications.

4.1.2 The Matlab Environment

In Matlab the user interface is the command window. The command window provides the prompt `>>` to let the user know that Matlab is ready to accept inputs. The user can either directly enter lines of Matlab code after the `>>` prompt in command-line mode; alternatively, in script mode, the user can enter `>>` *myscript.m* to execute the entire program *myscript.m*. The suffix `.m` indicates that the file contains a Matlab program; files with the `.m` suffix are affectionately known as "m files." (We will later encounter Matlab data files, which take the `.mat` suffix.) Note that most easily we run Matlab programs and subprograms (such as functions, discussed shortly) from the folder which contains these programs; however we can also set "paths" which Matlab will search to find (say) *myscript.m*.

Matlab in fact provides an entire environment of which the command window is just one (albeit the most important) part. In addition to the command window, there is a "current folder" window which displays the contents of the current directory — typically `.m` files and `.mat` files, but perhaps also other "non-Matlab " (say, document) files — and provides tools for navigation within the file system of the computer. The Matlab environment also includes an editor — invoked by the "Edit" pull-down menu — which permits the user to create and modify `.m` files. Matlab also provides several forms of "manual": `doc` invokes an extensive documentation facility window and search capability; and, even more useful, *within* the command window `>>help` *keyword* will bring up a short description of *keyword* (which typically but not always will be a Matlab "built–in" function). Similar environments, more or less graphical, exist for other (interpreted) programming languages such as Python.

We note that in actual practice we execute programs within programs within programs. We boot the system to start the Operating System program; we launch Matlab from within in Operating System to enter the Matlab environment; and then within the Matlab environment we run a script to execute our particular (numerical) program. It is the latter on which we have focused in our description above, though the other layers of the hierarchy are much better known to the "general computer user." It is of course a major and complicated task to orchestrate these different programs at different levels both as regards process control and also memory and file management. We will illustrate how this is done at a much smaller scale when we discuss functions within the particular Matlab context.

4.2 Data Types (and Classes)

All data in the computer is stored as 0's and 1's — binary digits. This is most convenient as binary operations are very efficiently effected in terms of the necessary nonlinear circuit elements. The basic unit of memory is the "word-length" of the machine — the number of binary digits, or "bits," which are used to represent data. Most machines these days are based on 32-bit or 64-bit words (which are 4 Bytes and 8 Bytes, respectively); in some cases particular data types might be represented by two words (or more).

But obviously we will need to interpret these 0's and 1's in different ways: in some cases the 0's and 1's might represent machines codes (instructions) which tell the arithmetic unit what to execute; in other cases, the 0's and 1's might represent data on which the the instructions will operate (the "operands"). Furthermore, there are many different type of data: any piece of data is defined not just by the 0's and 1's that make up the word, but also by the "data type" which tells the computer *how to interpret and operate upon the data*. As regards the latter, we note that the same set of 0's and 1's can mean something very different — and be operated on in very different ways — depending on the data type associated with these 0's and 1's.

There are several important types within Matlab (and homologous types within other programming languages). There are logical variables which are either 0 or 1 which correspond respectively to (and in fact the data may be entered as) `false` or `true` . There is integer data — a signed whole number. There is character data, in which the 0's and 1's encode particular characters such as letters in the alphabet or numerals — the famous ASCII code and recent extensions. And particularly important for us, there are floating point numbers, which in Matlab are 64 bits and are called (for largely historical reasons) simply `double`. Representation of floating point numbers (FPNs) by a 64 bits is probably less obvious than representation of a whole number, and hence we will discuss representation of FPNs, and also floating point operations, in a separate section. Matlab also supports a complex floating point type. These types are "atomic" in that they are part of the core Matlab scripting language.

In some programming languages the user is required upon creation of a variable to specify the data type. (We define variables in the next section — for now, think of a variable as the name of a particular piece of data.) Each variable must be an instance of one of the available (in our case, Matlab) data types. Matlab is much less picky than other programming languages — a blessing and a curse — and typically if no type is specified Matlab will simply assume a (double) floating point number, which is hence the "default."

It is, however, possible to specify that any particular variable is logical, with the `logical` function, an integer, with the (say) `int32` command, a character, with the `char` command (or more simply with quotes), and a floating point number, with the `double` command. We can also determine what type of data a particular variable is with the `islogical`, `isinteger`, `ischar`, and `isfloat` functions.

We already above and will frequently below refer to functions. We do not fully define functions until a later chapter, in which we learn how to create our own functions. For now, think of a function as a program which takes some (zero, one, two, or many) input arguments and yields some output; we provide illustrations in the next section. All of the functions we describe in the preceding paragraph are "built-in" functions that are part of the core Matlab scripting language; we shall exercise these functions in the next section once we are armed with the assignment operation. (In fact, `false` (and `true`) are also built-in Matlab functions: `false` takes no arguments and returns a logical zero.) Note the Matlab "core" is quite large compared to other languages such as Python, in which most functions must be explicitly brought in as modules. There are, however, official Matlab extensions to the core, known as "toolkits." In actual practice, functions can yield many outputs, not just a single output; we discuss this extension in a later chapter.

We note that often we might perform operations on data of different types. In that case we might effect "conversion" before proceeding with the operation, or we might let the programming language automatically perform "coercion" — conversion to a common data type based on rules of precedence. For example, in Matlab , if we attempt to add an integer and a floating point number, Matlab will (effectively) first convert the integer variable to a floating point variable. (Note that even when we invoke the various `round`, `fix`, `floor`, and `ceil` Matlab functions to round a floating point number (the input) to an integer (the output), the output is of *data type* `double`.) In most cases Matlab makes reasonable choices, though on occasion more direct control is warranted.

Finally we note that there is another term used to describe the proper specification and interpretation of given data: "class." Typically data type is part of the language whereas data classes are created by the user. And typically data type would refer to a word or maybe two words whereas class would refer to a compound type of many words (each of which might be specified as a different data type). Class also has a special significance in object-oriented programming: a class defines not just a compound data type (an instance of which is known as an "object"), but also functions or "methods" on members of this class. Matlab supports object-oriented programming both explicitly and implicitly, however we will only briefly touch on object-oriented programming in this course, and primarily for purposes of interpretation of various operators.

This section is rather abstract simply because we do not yet know how to create data and hence we can not demonstrate any of the concepts. As already indicated, we shall illustrate the notion of data types, the matlab functions used to define variables of a particular data type, and the functions used to query variables as to their data type, once we learn how to create variables — in the next section.

4.3 Variables and Assignment

The assignment statement is the most basic operation. We proceed by example (using the command-line input mode):

```
>> pi_approx = 3.14159

pi_approx =

    3.1416

>>
```

where we enter the material after the prompt `>>` and MATLAB responds as indicated in the lines below the prompt line. In general, MATLAB will display the result of an operation in the command window *unless* we put a semi-colon at the end of the instruction.

It is important to understand what this statement actually does. The variable `pi_approx` is a name to which MATLAB will associate a unique address — a location in memory; hence the variable `pi_approx` is in fact an address (a name), *not* a value. The assignment statement puts the value 3.14159 at the address `pi_approx`. Again, `pi_approx` is the address, or reference, or pointer, or box, which *contains* the value. If we change the value

```
>>pi_approx = 3.14

pi_approx =

     3.14

>>
```

we are simply replacing the earlier value 3.14159 at address `pi_approx` with the new value 3.14 at the same address `pi_approx`.

In the above assignment statement the `=` is an operator (and an operator which is part of the core MATLAB functionality). An operator is a function which takes one or two arguments and produces an output; operators, given the few operands, can be conveniently represented as a symbol. In the next few sections we will introduce a number of useful operators. In terms of an operator, `=` looks to the right for the value and then places this value in the address indicated to the left.

In our assignment for `pi_approx` the way in which we have entered the number ensures that `pi_approx` is of type floating point `double`. But we can be more explicit as in

```
>> pi_approx = double(3.14159)

pi_approx =

    3.1416

>> floatcheck = isfloat(pi_approx)

floatcheck =

     1

>>
```

which confirms that the `double` function indeed produces a variable of data type `double`.

We emphasize that both `double` and `isfloat` are functions. The general syntax for a function is simple: the input or inputs are included in parentheses following the function name, and the output (later, many outputs) — the evaluation of the function — is assigned to the variable to the left of the `=`. For example, `isfloat` takes as input a variable — here the input is `pi_approx` — and returns as output a logical variable which indicates whether the input is a floating point number — here the output is a 1 (true), since `pi_approx` is indeed a floating point number (or more precisely, an *instance* of the floating point `double` data type).

It is worth dissecting the above example a bit further. In fact, the statement

```
>> floatcheck = isfloat(pi_approx)

     floatcheck = 1

>>
```

is a very simple example of a very powerful capability — composition. In particular, in this case when `=` looks to the right it finds not a value but a function, `isfloat`; the function `isfloat` is then evaluated and the *output* of `isfloat` (a logical `1`) is then taken as the (right) input to the `=` (assignment) operator; the `=` then looks to the left, `floatcheck`, to know where in memory to write this value (a logical `1`) (or to create a memory location with the "name" `floatcheck`, as needed).

We invite the reader to input a logical with `logical` and check that the result is logical with `islogical`, and an integer with `int32` and check that the result is integer with `isinteger`. We will discuss characters and strings later in a separate section. Most often we will use the `double` data type, which is the MATLAB default and requires no special attention — we just enter the number as `3.14159`. We will also make extensive use of the logical data type.

4.4 The Workspace and Saving/Loading Data

In the previous section, we created variables such as `pi_approx` and `floatcheck`. Where do these variables go? These variables — and all variables we, or the programs we launch from the command window, create — go to what is known as the "workspace." In some sense, the workspace is the part of the memory in Figure 4.1 (interpreted as the MATLAB environment) which has already been allocated. In order to see all the variables in our workspace we can do `>> who`. Note that `>> who` does not evaluate the variables and hence does not provide the values; rather, it provides a list of all the variable names to which we have assigned data. (To determine the value of a particular *variable* we would simply type `>>`*variable*.) Note that `whos` is the same as `who` but with additional information about size and data type (or "class") of each variable.

If we wish to delete all the variables in the workspace, and start anew, we would do `clear`. (This of course only affects the MATLAB variables you have created, with no effect on the file system — such as the .m files you might have created.) Alternatively, you might wish to just delete certain variables, say *variable_1* and *variable_2*, in which case we would do `>> clear` *variable_1 variable_2*. For smaller programs memory management is not an issue, but for larger programs it is important to clear or more generally allocate and de-allocate memory carefully.

It is often the case that we wish to save the results of our calculations for future use — rather than re-compute. This is done very simply in MATLAB : to save variables *variable_1* and *variable_2* to a data file `save_for_later.mat` we do `save save_for_later` *variable_1 variable_2*. To reload this data we simply do `load save_for_later` which will include the contents of `save_for_later`, in our case *variable_1* and *variable_2*, in the current workspace — that is, in addition to the current contents of your workspace. Note the `.mat` file is a MATLAB specific data file which is the most convenient within MATLAB , but there are also many other (and more universal) formats in which data can be saved.

4.5 Arithmetic Operations

We now turn to arithmetic operations, typically performed in MATLAB on floating point numbers (though the operators also work on integers and indeed logicals through coercion, as discussed above).

As already indicated, an operator is a function, syntactically represented by a symbol, which takes one or two input arguments, or parameters, or operands, and returns a result. The arithmetic operators are `^` (exponentiation), `/` (division), `*` (multiplication), `+` (addition), and `-` (subtraction). These operations do the obvious thing, applied either to data or variables.

We consider addition: for data,

```
>> 2 + 3

ans =

     5
```

or for variables

```
>> x = 2; y = 3;
>> x + y

ans =

     5

>>
```

(Note the semi-colons suppress display of `x` and `y`; if we wish to confirm the value, we need only do (say) `>> x` without a semi-colon to evaluate `x`.) In more detail, the `+` operator takes the two values to the left and right and outputs as the answer (`ans`) the sum. The other arithmetic operators `^`, `/`, `*`, and `-`, perform in a similar obvious fashion.

We note that in the above `x + y` is an *expression* — here a single function, but more generally a composition of many operations — which is *evaluated* to yield the result `ans`. Note when we do an evaluation of an expression *expr*, MATLAB "finishes" the statement for us as `ans =` *expr* — in other words, MATLAB assigns the result of the evaluation to a variable `ans`. We can in fact use this variable `ans` subsequently, but this is highly frowned upon since, as you can imagine, it is quite easy for the generic `ans` to be changed in unanticipated or unnoticed ways.

Rather, if we wish to subsequently use the result of an evaluation, we should explicitly assign the output to a variable, say `z`: we need only do

```
>> z = x + y

     z = 5

>>
```

which is a composition of the addition (`+`) operator with the assignment (`=`) operator: we evaluate the expression (or operation, or function) `x + y` and then assign the result of this evaluation to a variable `z`.

We repeatedly above refer to the addition "function." In fact, for most operators there is an "*output = function_name (inputs)*" syntax which is equivalent to the operator syntax. For instance, we may compute the sum of `x` and `y` as `plus(x,y)`. Obviously the operator form is much easier and more readable, however the `plus` contains the actual code which implements addition and furthermore provides a mechanism by which to change what we mean by addition for different data types (and in the object-oriented context, different classes). We do not recommend that you change the definition of `plus` for `double` data types — but in fact, it is quite easy to do, and could prove an effective form of industrial numerical sabotage.

It is instructive to consider the statement

```
>> z = z + 1

    z = 6

>>
```

which in fact serves quite often in iterative procedures (typically for `z` an integer, even if represented in MATLAB by a double). What does this statement do? First MATLAB evaluates the expression `z + 1` to obtain the *value* `6`; then operator `=` assigns this value `6` to the address (variable) `z`. Although mathematically `z = z + 1` appears nonsensical, it is important to remember that in `z = z + 1` the `=` is the assignment operator and not an equal sign. (Of course, MATLAB contributes to the confusion in very next line, `z = 6`, by using the `=` sign in the convention mathematical sense of equality.)

Up to this point we have composed operators for which we did not need to worry about the order in which we performed the operations. For example, `3 + 4 + 5` can be evaluated either as `(3 + 4) + 5` (i.e., `3 + 4` first, then `+ 5`). (In fact, this is not quite true in finite precision, as we discuss in the next section.) On the other hand, `3*2^4` gives quite different results if we perform either the `*` first or the `^` first. Matlab thus provides rules of precedence for the order in which operators are evaluated: first the items in parentheses, from the inside out; then `^`; then `/` and `*`; then `+` and `-`. The mnemonic is PEDMAS (not very mnemonic, in fact). If the rules of precedence do not not dictate a unique order then Matlab operates from left to right.

The easiest way to avoid misinterpretation of your intentions is to liberally use parentheses, and for readability to always include ample spaces within and between your parentheses. The evaluation of `3*2^4` gives us opportunity to introduce the notion of a bug. If you intended `(3*2)^4` but simply wrote `3*2^4` Matlab would do something different from your intentions. This is not Matlab 's fault, but your fault. There are many kinds of bugs: a statement may in fact not correspond to valid Matlab syntax and the statement will not even be interpreted; a statement may correspond to valid Matlab syntax but may do something *other* than intended by the user — as in our `3*2^4` example; the statement may do what the users intends but in fact to achieve the desired end the user's intentions are not correct — this occurs quite frequently in the context of numerical computation.

4.6 Floating Point Numbers (FPNs): Representation and Operations

4.6.1 FPN Truncation and Representation

Floating point numbers represent a challenge both in how we represent these numbers and in how we perform arithmetic operations on these numbers. To begin, we express a number x in base 2 as

$$x = \sigma_1 \left(\sum_{k=0}^{\infty} b_k 2^{-k} \right) \times 2^{\sigma_2 \mathbb{E}} ,$$

in which the b_k are binary numbers — 0 or 1, $\mathbb{E}$ is an integer, and σ_1 and σ_2 are signs — ± 1. We assume that we have normalized the expansion such that $b_0 = 1$. (In fact, we may express x in say base 10 rather than base 2; this in turn will lead to a different floating point format.)

In some cases, we may only require a finite sum — a sum with a finite number of nonzero terms — to represent x. For example, $x = 2$ may be expressed by the single non-zero term $b_0 = 1$ (and $\mathbb{E} = +1$). However more generally a finite number of non-zero b_k will *not* suffice — even 1/10 leads to a repeating binary fraction. We thus must truncate the series to develop the floating point number (FPN) *approximation* of x:

$$x_{\text{FPN}} = \sigma_1 \left(\sum_{k=0}^{K} b'_k 2^{-k} \right) \times 2^{\sigma_2 \mathbb{E}'} .$$

Here $b'_k = b_k$, $1 \le k \le K$ — we perform truncation of our series — and $\mathbb{E}'$ is the minimum of $\mathbb{E}$ and $\mathbb{E}_{\max}$ — we truncate the range of the exponent.

We now represent or encode x_{FPN} in terms of (a finite number of) 0's and 1's. Towards this end we assign one bit each to the signs σ_1 and σ_2; we assign $p = K$ bits for the binary numbers b'_k, $1 \le k \le K$, to represent the mantissa (or significand); we assign $p_{\mathbb{E}}$ bits to represent the exponent $\mathbb{E}$ (and hence $\mathbb{E}_{\max} = 2^{p_{\mathbb{E}}}$). (Our choice of base 2 makes the encoding of our approximation in 0's and 1's particularly simple.) In the 64-bit IEEE 754 binary double (now called binary64) floating point format, $p = 52$ and $p_{\mathbb{E}} = 10$ (corresponding to $\mathbb{E}_{\max} = 310$ such that in total — including the sign bits — we require $2 + 52 + 10 = 64$ bits. (The storage scheme actually implemented in practice is slightly different: we need not store the leading unity bit and hence we effectively realize $p = 53$; the exponent sign σ_2 is in fact represented as a shift.)

There are two primary sources or types of error in the approximation of x by x_{FPN}: the first is FPN truncation of the mantissa to p bits; the second is the FPN truncation of the exponent to $p_{\mathbb{E}}$ bits. The former, FPN mantissa truncation, is generally rather benign given the rather large value of p. However, in some cases, FPN mantissa truncation errors can be seriously amplified by arithmetic operations. The latter, FPN exponent truncation, takes the form of either overflow — exponents larger than 310, represented in Matlab as plus or minus `Inf` — which is typically an indication of ill-posedness, or underflow — exponents smaller than -310, represented in Matlab as 0 — which is typically less of a concern.

We note that the word "precision" is typically reserved to indicate the number of bits or digits with which a floating point number is approximated on any particular hardware (and IEEE format); typically we focus on the mantissa. For example, 64-bit precision, or "double-precision," corresponds to 52 (or 53) binary digits of precision — roughly 16 decimal digits of precision — in the mantissa. Precision can also be characterized in term of "machine precision" or "machine epsilon" which is essentially the (relative) magnitude of the FPN truncation error in the worst case: we can find machine epsilon from the Matlab built-in function `eps`, as we will illustrate below. We will define machine epsilon more precisely, and later construct a code to find an approximation to machine epsilon, once we have understood floating point arithmetic.

Oftentimes we will analyze a numerical scheme in hypothetical "infinite-precision" arithmetic in order to understand the errors due to numerical approximation and solution in the absence of finite-precision FPN truncation effects. But we must always bear in mind that in finite precision arithmetic additional errors will be incurred due to the amplification of FPN truncation errors by various arithmetic operations. We shortly discuss the latter in particular to identify the kinds of operations which we should, if possible, avoid.

Finally, we remark that there are many ways in which we may choose to display a number say in the command window. How we display the number will not affect how the number is stored in memory or how it is approximated in various operations. The reader can do `>> help format` to understand the different ways to control the length of the mantissa and the form of the exponent in displayed floating point numbers. (Confusingly, format in the context of how we display a number carries a different meaning from format in the context of (IEEE) FPN protocol.)

4.6.2 Arithmetic Operations

We shall focus on addition since in fact this particular (simple) operation is the cause of most difficulties. We shall consider two numbers x_1 and x_2 which we wish to add: the first number has mantissa m_1 and exponent $\mathbb{E}_1$ and the second number has mantissa m_2 and exponent $\mathbb{E}_2$. We presume that $\mathbb{E}_1 > \mathbb{E}_2$ (if not, we simply re-define "first" and "second").

First, we divide the first mantissa by $2^{\mathbb{E}_1-\mathbb{E}_2}$ to obtain $m_2' = m_2 2^{-(\mathbb{E}_1-\mathbb{E}_2)}$: in this form, x_1 now has mantissa m_2' and exponent $\mathbb{E}_1$. (Note this division corresponds to a shift of the mantissa: to obtain m_2' we shift m_1 by $\mathbb{E}_1 - \mathbb{E}_2$ places to the right — and pad with leading zeros.) At this stage we have lost no precision. However, in actual practice we can only retain the first p bits of m_2' (since we only have p bits available for a mantissa): we denote by m_1'' the truncation of m_1 to fit within our p-bit restriction. Finally, we perform our FPN sum $z = x_1 + x_2$: z has mantissa $m_1 + m_2''$ and exponent $\mathbb{E}_1$. (Our procedure here is a simplification of the actual procedure — but we retain most of the key features.)

We can immediately see the difficulty: as we shift m_2 to the right we are losing $\mathbb{E}_1 - \mathbb{E}_2$ bits of precision. If the two exponents $\mathbb{E}_1$ and $\mathbb{E}_2$ are very different, we could lost all the significant digits in x_2. Armed with FPN we can in fact develop a simple definition of machine epsilon: the smallest `epsilon` such that `1 + epsilon = 1`, where of course by `+` we now mean finite precision FPN addition. Later we will take advantage of this definition to write a short program which computes machine epsilon; for our purposes here, we shall simply use the MATLAB built-in function eps.

It is clear that finite-precision and infinite-precision arithmetic are different and will yield different results — the difference is commonly referred to as "round-off" error. Indeed, finite-precision arthmetic does not even honor all the usual (e.g., commutative, associative) rules. We consider the example (recall that in MATLAB operations are performed from left to right in the absence of any precedence rules):

```
>> mach_eps = eps

mach_eps =

   2.2204e-16

>> (mach_eps/2 + 1 + mach_eps/2 - 1)/mach_eps

ans =

     0

>> (mach_eps/2 + mach_eps/2 + 1 - 1)/mach_eps

ans =

     1

>>
```

Clearly, in infinite precision arithmetic both expressions should evaluate to unity. However, in finite precision the order matters: in the first expression by definition `mach_eps/2 + 1` evaluates to 1; in the second expression, `mach_eps/2 + mach_eps/2` adds two numbers of identical exponent — no loss in precision — which are then large enough (just!) to survive addition to `1`. This anomaly is a "bug" but can also be a feature: we can sometimes order our operations to reduce the effect of round-off errors.

But there are situations which are rather difficult to salvage. In the following example we approximate the derivative of $sin(x)$ by a forward first-order difference with increment `dx` which is increasingly small:

```
>> cos(pi/4)

ans =

   0.707106781186548

>> dx = .01;
>> deriv_dx = (sin(pi/4 + dx) - sin(pi/4))/dx

deriv_dx =

   0.703559491689210
```

```
>> dx = 1e-8;
>> deriv_dx = (sin(pi/4 + dx) - sin(pi/4))/dx

deriv_dx =

   0.707106784236800

>> dx = 1e-20;
>> deriv_dx = (sin(pi/4 + dx) - sin(pi/4))/dx

deriv_dx =

     0

>>
```

We observe that what Newton intended — and the error bound we presented in Chapter 3 — is indeed honored: as `dx` tends to zero the finite difference (slope) approaches the derivative ($\cos(\pi/4)$). But not quite: as `dx` falls below machine precision, the numerator can no longer see the difference, and we obtain an $O(1)$ error — precisely in the limit in which we should see a more and more accurate answer. (As the reader can no doubt guess, `pi`, `sin`, and `cos` are all Matlab built-in functions.)

This is in fact very typical behavior. In order to make numerical errors small we must take smaller increments or many degrees-of-freedom, however if we go "too far" then finite-precision effects unfortunately "kick in." This trade-off could in fact be debilitating if machine precision were not sufficiently small, and indeed in the early days of computing with only a relatively few bits to represent FPNs it was a struggle to balance numerical accuracy with finite precision round-off effects. These days, with the luxury of 64-bit precision, round-off errors are somewhat less of a concern. However, there are situations in which round-off effects can become important.

In particular, we note that the problem is our derivative example is not just the numerator but also the `dx` in the denominator. As a general rule, we wish to avoid — where possible — division by small numbers, which tends to amplify the effects of finite-precision truncation. (This relates to *stability*, which is an important theme which we will encounter in many, often related, guises in subsequent chapters.) We will see that even in much more sophisticated examples — solution of large linear systems — "avoid division by small numbers" remains an important guideline and often a feature (by construction) of good algorithms. The problem of course is aggravated when we must perform *many* operations as opposed to just a few operations.

We have focused our attention on addition since, as indicated, this operation is often the proximal cause of the round-off difficulties. Other operations are performed in the "obvious" way. For example, to multiply two numbers, we multiply the mantissas and add the exponents and then re-adjust to conform to the necessary representation. Division and exponentiation follow similar recipes. Obviously underflow and overflow can be undesired byproducts but these are typically easier to avoid and not "fundamental."

4.7 Relational and Logical Operations

4.7.1 Relational Operations

A relational operator performs some kind of comparison between two variables (typically floating point double) and then returns a logical variable to indicate the outcome of the comparison. The relational operators are equal, `==`; not equal, `~=`; less than, `<`; greater than, `>`; less than or equal, `<=`; greater than or equal, `>=`. Note that the equal operator, `==`, is now interpreted in the mathematical sense of `=` (vs. the Matlab assignment operator `=` which of course serves a different function).

As an example, we consider

```
>> x = 4.6; y = 4.7;
>> isless = x < y

isless =

     1

>>
```

Here Matlab first evaluates the expression `x < y` — which, since `x` is less than `y`, is a true statement, and hence returns a logical `1` — and then assigns this value of `1` to the (logical) variable `isless`. (As we know, floating point numbers are truncated in finite precision arithmetic; since the `<` operates on the truncated form, of course the comparison is only good to machine precision.)

We can of course consider composition. Typically, composition of relational operations is by logical operations, as we now discuss. (In fact, we can apply usual arithmetic to logical variables, which are converted to floating point for the purpose. This is often convenient, but must be used with some care.)

4.7.2 Logical Operations

It would be possible to hi-jack operations such as `+` to perform Boolean operations on logical variables, however Matlab prefers to reserve the usual arithmetic functions for these operators applied to logical variables. Hence we introduce new operators.

The three key logical operators are AND, indicated as `&`, OR, indicated as `|`, and NOT, indicated as `~`. (There are also other options, such as the exclusive `or`, or `XOR`.) The AND and OR operators take as (the two) operands logical variables and the result is of course also a logical variable. The NOT takes a single logical operand and the result is also a logical variable. As already indicated, these logical operators are often composed with relational operators.

The AND, OR, and NOT behave in the expected fashion. The statement `L3 = L1 & L2` yields `L3 = 1` (true) only if both `L1` and `L2` are both `== 1` (true), otherwise `L3 = 0`. The statement `L3 = L1 | L2` yields `L3 = 1` if either `L1 == 1` or `L2 == 1`, otherwise `L3 = 0` (note that if both `L1 == 1` and `L2 == 1`, then `L3 = 1`); conversely, `L3 = 0` only if both `L1 == 0` and `L2 == 0`. Finally, the NOT: `L3 = ~L1` yields `L3 = 1` if `L1 == 0` and `L3 = 0` if `L1 == 1`.

As an example, we consider the AND and NOT:

```
>> x = 4.6; y = 4.7;
>> u = 3.1; v = 3.2;
>> z = (x < y) & ~(u < v)
```

```
z =

     0

>>
```

which of course yields the correct result. Note the MATLAB precedence convention is to evaluate relational operators before logical operators and hence our parentheses in the above are redundant — but better too many parentheses than too few. And another example,

```
>> x = 4.6; y = 4.7;
>> u = 3.1; v = 3.2;
>> z = (x < y) | ~(u < v)

z =

     1

>>
```

now with the OR and NOT.

Finally, we mention that MATLAB provides "short-circuit" versions of AND and OR, the operators for which are given by `&&` and `||`, respectively. These short-circuit operators will not evaluate the second operand (i.e., to the right) if the first operand (i.e., to the left) suffices to make the decision. For example, in our OR example above, since `x < y`, `z = (x < y) || ~(u < v)` will only evaluate `(x < y)`. This can lead to efficiencies — if the second operand is much more expensive and or ensure that a (second) operand is only evaluated when it exists and is well-defined.

Logical (and relational) operations play a key role in flow control — in controlling the flow (or "direction") of the program based on the results generated by the program itself. In some sense it is these logical and relations operations and the enabled flow control which distinguishes a program executed by a computer from arithmetic performed on a calculator.

4.8 Flow Control

4.8.1 The `if` Statement

The `if` statement is very simple. The general syntax is given by

```
if logical_expression_1
    BLOCK 1
elseif logical_expression_2
    BLOCK 2
else
    BLOCK 3
end
```

which is interpreted by MATLAB as follows:

`if` *logical_expression_1* evaluates to `true`, execute *BLOCK 1* and go to `end` (and continue on with the next line in the program);

`if` *logical_expression_1* evaluates to `false`, but *logical_expression_2* evaluates to `true`, execute *BLOCK 2* and go to `end` (and continue);

`if` *logical_expression_1* is `false` and *logical_expression_2* is `false`, execute *BLOCK 3* and go to `end` (and continue).

Note several variants are possible: we may include multiple `elseif` statements between the *BLOCK 1* and `else`; and we may omit the `else` and *BLOCK 3* statements.

In the event that in fact there are many cases there is another MATLAB statement which is no doubt preferable to the `if` statement: the `switch` statement. We do not discuss the `switch` statement here, but rather use this opportunity to recall the very helpful help function: `>> help switch` will provide you with a quick (but sufficient) description of the syntax and functionality of the `switch` statement.

4.8.2 The `while` Statement

The syntax of the `while` statement is

```
initialize var_1, var_2,...
while relational_or_logical_expression_while (var_1, var_2,...)
    BLOCK % new values assigned to var_1, var_2, ...
end
```

This statement is a loop which is controlled by the value of the logical variable which is the result of evaluation of `relational_or_logical_expression_while`. What makes the statement powerful is that `relational_or_logical_expression_while` may depend on *var_1, var_2, ...*, the values of which are changed at each pass through the loop. In particular, the `while` is executed as follows: if *relational_or_logical_expression_while (var_1, var_2, ...)* is `true`, execute (the instruction lines in) *BLOCK*, otherwise go to `end` and continue with next line of program of which the `while` is part; repeat. Note that *var_1, var_2, ...* must be initialized prior to entering the `while` loop.

As an example, we consider

```
>> i = 0;
>> sum_of_integers = 0;
>> while i <= 3
         sum_of_integers = sum_of_integers + i;
         i = i + 1;
     end
>> sum

sum_of_integers =

     6

>>
```

Here, *relational_or_logical_expression_while (var_1)* is the simple expression `i <= 3`, where `i` plays the role of *var_1*. In fact, this particular sum is more easily performed with a `for` statement, as we describe shortly. The true value of the `while` is when the *relational_or_logical_expression* is a more complicated function of the variables which are changed by the *BLOCK* and in particular when we do not know *a priori* how many iterations through the loop will be required.

For example, we create the script *machine_eps.m*

```
% script to calculate machine epsilon
mach_eps = 1;
while (1 + mach_eps ~= 1)
    mach_eps = mach_eps/2.;
end
mach_eps
```

in the editor. Note that as we start to construct slightly longer programs we will start to use script mode rather than command-line mode. The input is easier, as are debugging and modification, and of course re-use. In addition, the editor within MATLAB recognizes various keywords and automatically formats statements for improved readability. Note also the comment line: any material on a line which follows a `%` will be ignored by the interpreter and serves only to educate the author and user's as to the intent of the program or particular lines of the program.

We then enter in the command window

```
>> machine_eps

mach_eps =

   1.1102e-16

>>
```

which runs our script. Of course this code is just calculating for us machine precision, which agrees with the MATLAB `eps` to within the factor of two related to our stopping tolerance.

Finally, we note that you may include a `break` in a *BLOCK* statement

`if` (*relational_or_logical_expression_break*) `break`

which will directly go to the `end` statement of the `while` loop quite independent of whether `relational_or_logical_expression_while` (for the current values of *var_1, var_2, . . .*) is `true` or `false`. Purists often try to avoid `break` statements, but pragmatists tolerate the occasional `break` statement. Too many `break` statements may be a symptom of poor program design or more generally passive-aggressive tendencies.

4.8.3 The `for` Statement

The syntax for the `for` statement is

```
for VARCOUNTER = LIM_1 : INCREMENT : LIM_2
    BLOCK % no reassignment of VARCOUNTER
end
```

Typically *LIM_1, INCREMENT, LIM_2* would be integers (even if of data type `double`), however there are also many instances in which these variables could be (mathematically) non-integer. Note also that *INCREMENT* can be positive or negative, and that if *INCREMENT* is not specified then MATLAB chooses the default value *INCREMENT* `= 1`.

The execution of the `for` loop is simple: we execute *BLOCK* for *VARCOUNTER* `=` *LIM_1*; we update *VARCOUNTER* `=` *VARCOUNTER* `+` *INCREMENT*; then, if *VARCOUNTER* `<=` *LIM_2*, repeat. As for a `while` statement, we can interrupt the usual flow of the `for` loop with a `break` statement. Note that if *LIM_1* `+` *INCREMENT* is less than *LIM_2* then *BLOCK* will never be executed.

We repeat our earlier example:

```
>> sum_of_integers = 0;
>> for i = 1:3
      sum_of_integers = sum_of_integers + 1;
   end
>> sum_of_integers

sum_of_integers =

     3

>>
```

There are many situations in numerical methods in which the `for` loop is the ideal construct. However, it is also true that in MATLAB there are a number of functions related to array manipulation that, although implicitly built upon a `for` construction, are typically more efficient than an explicit `for` loop. We introduce some of these functions in the next section.

Finally, we note that our `for` syntax here is not as general as provided for by MATLAB . However, the more general syntax requires the notions of single-index or multi-index arrays, which we have not yet introduced. Following the treatment of arrays the reader should do `>> help for` to understand the generalization: in effect, we can require *VARCOUNTER* to cycle through any particular set of scalars (defined in a single-index array) or *VARCOUNTER* itself may even be a single-index array which cycles through a particular set of single-index arrays (defined in a double-index array).

Chapter 5

Matlab Arrays

5.1 Single-Index Floating Point Arrays

5.1.1 The Concept

It is often the case that we have an ordered set of, say n, "elements" of data which are somehow related. The index could represent directions in three space dimensions ($k = 1, 2, 3$ for x, y, z, respectively) — at which we store, in each array location, the corresponding coordinate of a point (an array of length 3); or the index could represent 15 different times — at which we store, in each location, the time of the measurement, or perhaps the measurement itself (an array of length 15). Note the index plays the role of independent variable and the array value plays the role of dependent variable. In all these cases we will often wish to operate on all n "related elements" in a similar fashion. We would thus like a way to reference all the elements with a common name, and an easy way to reference different elements through this common name, in order to develop succinct, readable, and efficient code for implementing common operations on all the elements. In particular, we would not want to write n lines of code each time we wished, say, to square each of the n elements.

A single-index array — in this section, a floating point single-index array — is the simplest "class" which achieves these objectives. We first introduce a variable name, *array_name*, which shall be associated to all n elements of the array. We then *index* this *array_name* in order to access any particular element of the array: in particular, *array_name*`(i)` is the pointer to element i of the array. We emphasize that, as in the scalar case, *array_name*`(i)` is not the value of element `i` of the array but rather the location in memory at which we shall store the value of element `i`. For example, *array_name*`(2) = 3.14159` would assign the value `3.14159` to the second element of the array. (We discuss more efficient assignment methods below.)

Conceptually, you may think of the array as stored at n contiguous locations in memory. Element 1 of the array is stored in *array_name*`(1)`, element 2 of the array is stored in *array_name*`(2)`, ..., and element n of the array is stored in *array_name*`(n)`. In this (virtual) sense, it suffices to (say) pass to a function simply the variable *array name* — which you may view as the address of the first element — as the addresses of all the other elements of the array can then be readily deduced from *array name*. (In actual practice, there is also some header information associated with the array — for example, in our single-index case, the length n.) Since many common array operations can be performed in Matlab with simple function calls — or user-defined function calls — at a high-level we can often deal exclusively with *array_name* without explicit reference to particular indices. (Of course, under the hood. . .)

5.1.2 Assignment and Access

The most explicit way to create a single-index array is by hand: `X = [1,3,4]` creates a single-index array of length 3, with entries `X(1) = 1, X(2) = 3, and X(3) = 4`. To determine the length of an array MATLAB provides a function `length`. In our example

```
>> X = [1,3,4]

X =

     1     3     4

>> X(1)

   ans = 1

>> length(X)

ans =

     3

>>
```

Note that this single-index array is a *row* single-index array. (We can also consider column single-index arrays, in which the commas above are replaced by semi-colons: `X = [1;3;4]`. We reserve treatment of rows and columns to our discussion of double-index arrays.)

The input process is facilitated by the colon operator. In particular, `Z = [J:D:K]` creates the single-index array `J, J+ D,...,J + m*D]` for `m = fix((K-J)/D)`, where `fix` is a MATLAB function which rounds to the nearest integer towards zero. (Note that `J:D:K` is empty if `D == 0`, if `D > 0 & J > K`, or if `D < 0 & J < K`.) The default value of `D` is 1 and hence `J:K` is equivalent to `J:1:K`, as in the example

```
>> firstfive = [1:5]

firstfive =

     1     2     3     4     5

>>
```

You will in fact recognize this colon construction from the `for` statement; and indeed the `for` statement may take the form `for VARCOUNTER = S` where `S` is any single-index array; we will revisit this construction of the for statement shortly.

We may assign an entire array,

```
>> Y = X

Y =

     1     3     4

>>
```

or we may assign or re-assign a particular element as, say,

```
>> X(3) = 10;
>> X

X =

     1     3    10

>>
```

which we see modifies `X` accordingly.

Of course the point of an array is that we have a systematic numerical approach to indexing, and we may thus easily assign values with a `for` statement. We present two approaches. In the first, we zero out to initialize:

```
>> Z = zeros(1,10);
>> for i = 1:length(Z)
        Z(i) = i^2;
   end
>> Z

Z =

     1     4     9    16    25    36    49    64    81   100

>>
```

Note that `zeros(1,`n`)` is a Matlab function which provides a (row) single-index array of all zeros of length n. (Note the first argument of `zeros` indicates that we wish to form a row single-index array; `zeros(`n`, 1)` would create a column single-index array of all zeros. We will understand `zeros` better once we discuss multi-index arrays.) This approach, of initialization, is preferred whenever possible: it permits Matlab to be more efficient in memory allocation and management.

In the second approach, we concatenate:

```
>> Z = [];
>> for i = 1:10
    Z = [Z,i^2];
   end
>> Z
```

```
Z =

     1     4     9    16    25    36    49    64    81   100

>>
```

Here `Z = []` defines an array but the array is initially empty: we do this because we can not "add to" (append to, or concatenate to) an array which does not exist. Note that the expression `[Z, i^2]` evaluates to an array in which the first `length(Z)` elements are the elements of `Z` and the last element is `i^2`. Thus at each iteration the length of `Z` grows. The above is less efficient than the initialization approach, but very convenient in particular (for example, in a `while` statement) when we do not *a priori* know the number of elements in our (ultimate) array.

As our last point on assignment and access, we note that Matlab supports a very convenient form of indirect addressing. In particular, if we create a single-index array of integers `indvec` then we can extract from (say) `Z` just those elements with indices in `indvec`:

```
>> indvec = [1,3,5,9];
>> U = Z(indvec)

U =

     1     9    25    81

>>
```

Note you may also apply indirect addressing on the left-hand side of the assignment statement, but some care must be exercised as regards the "shape" (row vs. column) of the resulting single-index array.

Note that in all these shortcuts there is always an equivalent underlying program (which is more or less how these shortcuts are implemented). For example, in the above, an array index argument tells Matlab to execute, effectively:

```
>> U_too = zeros(1,length(indvec))
>> for inew = 1:length(indvec)
       U_too(inew) = Z(indvec(inew));
   end
>> U_too

U_too =

     1     9    25    81

>>
```

But of course much better to encapsulate and re-use this feature within the Matlab syntax than to re-write this little loop on each occasion.

5.1.3 (Dotted) Arithmetic Operations

It is one of the major advantages of programming languages that as we develop more convenient data structures (or types, or classes), in particular with many elements, we may also suitably define our operations to deal directly with these new entities — as a whole, rather than explicitly manipulating each element. In this section we define for the case of single-index arrays the corresponding array arithmetic operators.

We first discuss element-by-element operations. In these operations, we consider two arrays of the same length and we apply the same arithmetic operation on each pair of elements which share the same index. We begin with addition/subtraction (and multiplication by a scalar):

```
>> P = [1, 4, 7]; Q = [2, -1, 1];
>> R = P + 3.0*Q

R =

     7     1    10

>>
```

which is simply shorthand for the loop

```
>> for i = 1:length(P)
       R_too(i) = P(i) + 3.0*Q(i);
   end
>> R_too

R_too =

     7     1    10

>>
```

The loop makes clear the interpretation of "element by element," but obviously the one-line statement is much preferred.

Note `3.0` is not an array, it is *scalar*, which scales all elements of the array `Q` in our above example. This simple scaling feature can be very convenient in defining arrays of grid points or data points (for example, in interpolation, differentiation, and integration). To wit, if we have an interval of length L and wish to create N segments of length L/N, we need only do `>> xpts = (L/N)*[0:N]`. Note that `xpts` is of length (in the sense of number of elements in the array) `N+1` since include both endpoints: `xpts(1) = 0` and `xpts(N+1) = L/N`.

Finally, we note one additional shortcut: if `q` is a scalar, then element-by-element addition to our vector `P` will add `q` to each element of `P`. To wit,

```
>> q = 3;
>> P + q

ans =
     4     7    10

>>
```

We might write the above more properly as

```
>> P + q*ones(1,length(P))

ans =

     4     7    10

>>
```

but there is no need given Matlab 's automatic expansion of `q` when encountered in array addition. Note `ones(1,`n`)` is a Matlab function which creates a (row) single-index array of length n with all elements set to unity.

To implement element-by-element multiplication, division, and exponentiation we do

```
>> PdotmultQ = P.*Q

PdotmultQ =

     2    -4     7

>> PdotdivideQ = P./Q

PdotdivideQ =

    0.5000   -4.0000    7.0000

>> PdotexpQ = P.^Q

PdotexpQ =

    1.0000    0.2500    7.0000

>>
```

which is equivalent to

```
>> for i = 1:length(P)
      PdotmultQ_too(i) = P(i)*Q(i);
      PdotdivideQ_too(i) = P(i)/Q(i);
      PdotexpQ_too(i) = P(i)^Q(i);
   end
>> PdotmultQ_too

PdotmultQ_too =
     2    -4     7

>> PdotdivideQ_too

PdotdivideQ_too =

    0.5000   -4.0000    7.0000

>> PdotexpQ_too

PdotexpQ_too =

    1.0000    0.2500    7.0000

>>
```

As for addition, if we replace one of our vectors by a scalar, MATLAB will expand out with a “ones” vector.

Why do we need the “dot” before the `*`, `/`, and `^` operators — so-called “dotted” (or element-by-element) operators: dotted multiplication, dotted division, and dotted exponentiation? It turns out that there are two types of entities which look very similar, respectively arrays and vectors (later multi-index arrays and matrices): both are ordered sets of n floating point numbers. However, the arithmetic operations are defined very differently for these two entities, respectively element-by-element operations for arrays and linear algebraic operations (e.g., inner products) for vectors. We could easily define say `*` to perform element-by-element multiplication for objects which are defined as arrays, and to perform an inner product for objects defined as vectors. Unfortunately, although conceptually quite clean, this would be very cumbersome since often in one line we wish to treat a set of numbers as an array and in the next line we wish to treat the same set of numbers as a vector: there would be much conversion and bookkeeping. Hence MATLAB prefers a kind of “superclass” of array-and-vector (and matrix) entities, and hence perforce some new syntax to indicate whether we wish to treat the array-and-vector as an array (with dotted element-by-element operators) or a vector (with undotted linear algebra operators). Note that we do not require dotted `+` (or dotted `-`) since element-by-element addition and vector addition in fact are equivalent. So there.

We note that there are many arithmetic operations on arrays in addition to element-by-element operations, many of which are available as MATLAB functions. For example, we can easily perform the sum of the first three integers (as we did earlier with a `for` loop) as

```
>> ints = [1:3];
>> sum(ints)

ans =

     6

>> mean(ints)

ans =
     2

>>
```

where sum performs the sum of all the elements in an array (here `ints`) and mean calculates the arithmetic mean of all the element in an array.

Finally, we note here that the many Matlab built-in mathematical functions — we have already encountered sin and cos, but there are many many more — look for "math function" and "special functions" in the `doc` — which also accept single-index (and in fact, double–index) arguments. For example,

```
>> xpts = (pi/4)*0.5*[0:2];
>> sin_values = sin(xpts)

sin_values =

         0    0.3827    0.7071

>>
```

with a single call provides the values of sin for all elements of `xpts`.

5.1.4 Relational and Logical (Array) Operations

For relational and logical operations we do not have the complication of array/vector conflation and hence we need no dots. In effect, when we apply any of our scalar relational/ logical operations to pairs of vectors of the same length, Matlab returns a vector (of the same length as the operands) which is the result of the scalar relational/logical operation element-by-element.

As an example,

```
>> x = [1.2, 3.3, 2.2]; y = [-0.1, 15.6, 2.0];
>> z_1 = (x < y)

z_1 =

     0     1     0

>> z_2 = (x > y)

z_2 =

     1     0     1

>> z_3 = z_1 | ~z_2

z_3 =

     0     1     0
```

```
>>
```

Similar example can be constructed for all of our relational operators and logical operators. Note that `z_1, z_2` and `z_3` are logical arrays: each element is an instance of the logical data type.

For completeness, we indicate the implementation of the above as a `for` loop:

```
>> for i = 1:length(x)
      z_1_too(i) = (x(i) < y(i));
      z_2_too(i) = (x(i) > y(i));
      z_3_too(i) = z_1_too(i) | ~ z_2_too(i) ;
   end
>> z_1_too

z_1_too =

     0     1     0

>> z_2_too

z_2_too =

     1     0     1

>> z_3_too

z_3_too =

     0    1     0

>>
```

which is indeed equivalent, but tedious.

5.1.5 "Data" Operations

There are also a number of operations which albeit numerical are focused as much on the indices as the data — and hence we include these under the heading "data" operations.

A number of MATLAB functions are available to reorder the elements or to identify distinguished elements: `sort`, `min`, and `max` are perhaps the most useful in this regard. We illustrate just one of these functions, `min`:

```
>> T = [4.5, -2.2, 6.3, 4.4];
>> [minimum, minimizer] = min(T)

minimum =

   -2.2000
```

```
minimizer =

     2

>> minimum_too = min(T)

minimum_too =

   -2.2000

>>
```

which yields the obvious result. This is our first example of a function with two outputs: `minimum` is the minimum of the array, and `minimizer` is the index of the minimizing element. Note also that if we just wish to obtain the first output we can abbreviate the call.

Perhaps one of the most useful array data functions is the `find` function. Given a logical vector `L`, `find(L)` will return a vector which contains (in increasing order) the indices of all the elements of `L` which are nonzero (and are hence unity, since `L` is a logical array). (In fact, `find` can also be applied to a `double` array, but one must be careful about round-off effects.) As an example:

```
>> L = logical([0,1,1,0,0,1]);
>> islogical(L)

ans =

     1

>> ind_of_nonzero = find(L)

ind_of_nonzero =

     2     3     6

>>
```

where we have also illustrated the construction of a logical vector. Note that the `find` function effectively implements the `for` loop

```
>> ind_of_nonzero_too = [];
>> for i = 1:length(L)
   if( L(i) ~= 0 )
        ind_of_nonzero_too = [ind_of_nonzero_too,i];
   end
end
>> ind_of_nonzero_too

ind_of_nonzero_too =

     2     3     6
```

```
>>
```

which demonstrates also an application of concatenation.

The function `find` is very useful in the context of comparisons. For example, we may wish to extract just those values of vector greater than some threshold:

```
>> H = [0.2, 1.4, 6.7, -3.4, 4.2];
>> log_H_thresh = (H > 1.2)

log_H_thresh =

     0     1     1     0     1

>> inds_H_thresh = find(log_H_thresh)

inds_H_thresh =

     2     3     5

>> H(inds_H_thresh)

ans =

    1.4000    6.7000    4.2000

>>
```

We can of course replace `H > 1.2` in the above with any more complicated composition of relational and logical operators.

We could of course combine this as

```
>> H ( find ( H > 1.2 ) )

ans =

    1.4000    6.7000    4.2000

>>
```

In fact, MATLAB accepts a further abbreviation as

```
>> H ( H > 1.2 )

ans =

    1.4000    6.7000    4.2000

>>
```

in which a find is automatically applied to a logical index vector. Apparently this somewhat syntactically sloppy approach is in fact the most efficient.

We take this opportunity to revisit the for statement. We initially introduced the for statement of the form for VARCOUNTER = LIM_1:INC:LIM_2 and later noted that the for statement may take a more general form for VARCOUNTER = S where S is any single-index array. We now present an example of this general form of the for statement. Our objective is to find a number of entries in a (row) single-index array that are positive. As before, we can write the for loop based on an index as

```
>> scalars = [1,-3,1,2,-5];
>> num_pos = 0;
>> for i = 1:length(scalars)
      if (scalars(i) > 0)
         num_pos = num_pos + 1;
      end
   end
>> num_pos

num_pos =

     3

>>
```

which gives the expected result. Alternatively, we may use our set of scalars directly as the argument to the for loop. That is

```
>> scalars = [1,-3,1,2,-5];
>> num_pos = 0;
>> for sca = scalars
      if (sca > 0)
         num_pos = num_pos + 1;
      end
   end
>> num_pos

num_pos =

     3

>>
```

which also gives the correct result. In this second form, within the for loop, the loop argument sca is first set to the first element of scalars, 1, sca is then set to the second element of scalars, -3, and so on.

We may interpret the (restrictive) form of the for statement for VARCOUNTER = LIM_1:INC :LIM_2 within this general form: LIM_1:INC:LIM_2 first yields an single-index array [LIM_1, LIM_1+INC, ..., LIM_2] (assuming LIM_2 = LIM_1 + m*INC for some integer m); then the for loop successively assigns an element of the array to VARCOUNTER. (We note that the argument of the `for` loop must be a *row* single-index array, not a *column* single-index array. We revisit this point in our discussion of the `for` statement using double-index arrays.)

5.2 Characters and Character Single-Index Arrays (Strings)

Our focus is on numerical methods and hence we will not have too much demand for character and string processing. However, it is good to know the basics, and there are instances — typically related to more sophisticated applications of software system management, “codes that write codes” (or less dramatically, codes some lines of which can be modified from case to case), and also symbolic manipulation — in which character concepts can play an important role. Note that character manipulation and symbolic manipulation are very different: the former does not attribute any mathematical significance to characters; the latter is built upon the former but now adds mathematical rules of engagement. We consider here only character manipulation.

A character variable (an instance of the character data type), say `c`, must represent a letter of numeral. As always, `c` ultimately must be stored (ultimately) by 0’s and 1’s. We thus need — as part of the data type definition — an encoding of different characters in terms of 0’s and 1’s. The most common such encoding, the original ASCII code, is a mapping from 8-bit words (binary numbers) to the set of letters in the alphabet, numerals, punctuation marks, as well as some special or control characters. (There are now many “extended” ASCII codes which include symbols from languages other than English.)

We can create and assign a character variable as

```
>> c = '3'

c =

3

>> c_ascii = int8(c)

c_ascii =

   51

>> c_too = char(c_ascii)

c_too =

3

>>
```

In the first statment, we enter the single-character data with quotes — which tells MATLAB that 3 is to be interpreted as a character and not a number. We can then obtain the ASCII code for the number 3 — which happens to be 51. We can then recreate the character variable `c` by directly appealing to the ASCII code with the `char` command. Obviously, quotes are easier than memorizing the ASCII code.

A “string” is simply a single-index array of character elements. We can input a string most easily with the quote feature:

```
>> pi_approx_str = '3.1416'

pi_approx_str =

3.1416

>> pi_approx_str(2)

ans =

.

>> pi_approx_str + 1

ans =

   52    47    50    53    50    55

>>
```

We emphasize that `pi_approx_str` is not of type `double` and if we attempt to (say) add 1 to `pi_approx_str` we get (effectively) nonsense: MATLAB adds 1 to each element of the ASCII-translation of the our string according to the rules of single–index array addition.

We can readily concatenate strings, for example:

```
>> leader = 'The value is '

leader =

The value is

>> printstatement = [leader,pi_approx_str,' .']

printstatement =

The value is 3.1416 .

>>
```

However, this is of limited use since typically we would know an approximation to π not as a string but as double.

Fortunately, there are some simple conversion functions available in MATLAB (and other programming languages as well). The MATLAB function `num2str` will take a floating point number and convert it to the corresponding string of characters; conversely, `str2num` will take a string (presumably of ASCII codes for numerals) and convert it to the corresponding floating point (`double`) data type. So for example,

```
>> pi_approx_double = str2num(pi_approx_str)

pi_approx_double =

    3.1416

>> pi_approx_str_too = num2str(pi_approx_double)

pi_approx_str_too =

3.1416

>>
```

This can then be used to create a print statement based on a floating point value (e.g., obtained as part of our numerical calculations):

```
>> printstatement_too = [leader,num2str(pi_approx_double),' .']

printstatement_too =

The value is 3.1416 .

>>
```

In actual practice there are higher level printing functions (such as `fprintf` and `sprintf`) in MATLAB built on the concepts described here. However, the above rather low-level constructs can also serve, for example in developing a title for a figure which must change as (say) the time to which the plot corresponds changes.

5.3 Double-Index Arrays

5.3.1 Concept

Double-index arrays (and more generally, multi-index arrays), are extremely important in the implementation of numerical methods. However, conceptually, they are quite similar to single-index arrays, and inasmuch this section can be rather short: we just consider the "differential innovation." In fact, as we will see shortly, a double-index array really is a single-index array as far as internal representation in memory or "address space": the two indices are just a convenient way to access a single-index array.

Reference by two indices (or three indices,...) can be convenient for a variety of reasons: in a 10×10 structured rectilinear mesh, the two indices might represent the location of a point in a "Cartesian" grid — at which we store, in each array location, say the value of the temperature field (a 10×10 array); in an unstructured three-dimensional mesh (or a Monte Carlo random sample), the first index might represent the label/order of a point within a sample of say length 1000, and the second index might represent the spatial coordinate direction (e.g., $1, 2, 3$ for the x, y, z directions) — at which we store, in the array locations, the coordinate of the point (a 1000×3 array); and most notably, the two indices might represent the rows and columns of an $m \times n$ matrix — at which we store the matrix values (an $m \times n$ array). (We discuss matrices in depth later.) Recall that the indices play the role of the independent variable and the array values the role of the dependent variable.

For a double-index array, just as for a single-index array, we first introduce a variable name, *array_name*, but now this double-index array is associated to $m \times n$ elements: we may think of a double-index arrays as m rows by n columns, as we shall visualize shortly. We then index this *array_name* to access any particular element of the array: in particular, *array_name*`(i,j)` is the pointer to element i, j of the array. As always, the pointer is the location in memory at which we store the value of the array.

In fact, even this double-index array is stored at contiguous locations in memory. (It is in this sense that a double-index array is internally equivalent to a single-index array; we shall take advantage of this equivalence later.) Matlab stores a double-index array as "first address fastest": *array_name*`(1,1),...,`*array_name*`(m,1),`*array_name*`(1,2),...,`*array_name*`(m,2),...,` *array_name*`(m,n)`. As for the single-index case, it suffices to to pass to a function simply the variable *array_name* — the address of the first element — as the addresses of all other elements can then be readily deduced (in practice, thanks to appropriate header information). And of course, as for a single-index array, we will endeavor to similarly define operations on the entire array, as represented by *array_name*, rather than treat each index separately.

A note on nomenclature: we can also think of single-index and double-index arrays as "one-dimensional" and "two-dimensional" arrays. However, we will reserve "dimension" for the linear algebra sense: a vector with n entries is a member of n-dimensional space. Hence the linear algebra "dimension" is analogous to the Matlab `length`. (And just to make sure this paragraph, in attempting to avoid confusion, is sufficiently confusing: to avoid confusion of Matlab `length` with the linear algebra "length" of a vector we shall refer to the latter as the "norm" of the vector. We return to this point in Unit 3.)

5.3.2 Assignment and Access

We start with a simple example and then explain the syntax.

```
>> A = [1,2,3;4,5,6]

A =

     1     2     3
     4     5     6

>> size_of_A = size(A)

size_of_A =

     2     3

>> A(2,2)

ans =

     5

>>
```

We see that a comma separates elements within a row, and a semicolon separates different rows. The size function returns a single-index array of length 2: the first element is the number of rows — the limit for the first index — and the second element is the number of columns — the limit for the second index. Finally, we see that we can access individual elements of A as (say) A(2,2). Of course our interpretation as rows and columns is just an artifice — but a very useful artifice which we invoke on many many occasions — for visualization and interpretation.

Our row single-index array is special case of double-index array:

```
>> X = [1,3,4]

X =

     1     3     4

>> size(X)

ans =

     1     3

>> X(1,3)

ans =

     4

>>
```

And we can now systematically introduce a column single-index array as

```
>> X_col = [1;3;4]

X_col =

     1
     3
     4

>> size(X_col)

ans =

     3     1

>> X_col(3,1)

ans =

     4
```

```
>>
```

Note in this case each row is of length 1 so we require no comma delimiters. Note operations in MATLAB require arrays of similar size, so always make sure that pairs of single-index array operands are both row arrays or both column arrays. This is best ensured by initialization of the array with `zeros` and consistent assignment.

The transpose operation is very convenient: it "flips" the two indices. Hence

```
>> A_transp = A'

A_transp =

     1     4
     2     5
     3     6

>> X'

ans =

     1
     3
     4

>> size(X')

ans =

     3     1

>>
```

Rows become columns and columns become rows. (In fact, the transpose operator is a special case of a more general MATLAB function `reshape` which allows us to "resize" an array.)

As always, the reader should mentally note the more expanded code which effects any particular operation to make sure the operation is well understood: in this case

```
>> for i = 1:size(A,1)
       for j = 1:size(A,2)
           A_transp_too(j,i) = A(i,j);
       end
   end
>> A_transp_too

A_transp_too =

     1     4
     2     5
     3     6

>>
```

Note that `size(A,1)` and `size(A,2)` conveniently return the first element and second element of `size(A)`.

As for single-index arrays, we can directly assign an entire double-index array: `B = A` creates a new array `B` identical to `A` *in terms of size* as well as values; we may also assign or re-assign any particular element of the array as selected by the index — for example, `B(1,1) = 0`. Oftentimes we can create the desired array as an assignment plus modification of an existing array.

We may also create an array with a "double" `for` loop:

```
>> m = 2;n = 3;
>> A = zeros(m,n);
>> for i = 1:size(A,1)
      for j = 1:size(A,2)
         A_too(i,j) = j + (i-1)*size(A,2);
      end
   end
>> A_too

A_too =

     1     2     3
     4     5     6

>>
```

Note initialization of multi-index arrays is particularly important since these arrays tend to be larger and memory management even more of an issue.

However, concatenation also works for multi-index arrays and can be very effective.

```
>> R1 = [1,2,3]; R2 = [4,5,6];
>> C1 = [1;4]; C2 = [2;5]; C3 = [3;6];
>> A_too_too = [R1; R2]

A_too_too =

     1     2     3
     4     5     6

>> A_too_too_too = [C1,C2,C3]

A_too_too_too =

     1     2     3
     4     5     6

>> A_four_times = [A_too, A_too; A_too, A_too]
```

```
A_four_times =

     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6

>> A_four_times_also = [[A_too;A_too],[A_too;A_too]]

A_four_times_also =

     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6

>> A_four_times_expand_by_one = [A_four_times,[C1;C2]; [R1,R2],0]

A_four_times_expand_by_one =

     1     2     3     1     2     3     1
     4     5     6     4     5     6     4
     1     2     3     1     2     3     2
     4     5     6     4     5     6     5
     1     2     3     4     5     6     0

>>
```

The general procedures for concatenation are somewhat difficult to succinctly describe — we must always combine entities that "match" in the direction in which we concatenate — but the cases above include most instances relevant in numerical methods.

We can also do indirect addressing for double-index arrays, as we illustrate on our array `A_four_times`. In particular, let `ind1vec` and `ind2vec` be single-index arrays given by (say)

```
>> ind1vec = [2,3]

ind1vec =

     2     3

>> ind2vec = [2:4]

ind2vec =

     2     3     4

>>
```

Then

```
>> extracted = A_four_times(ind1vec,ind2vec)

extracted =

     5     6     4
     2     3     1

>>
```

which in fact is implemented as

```
>> for i = 1:length(ind1vec)
       for j = 1:length(ind2vec)
            extracted_too(i,j) = A_four_times(ind1vec(i),ind2vec(j));
       end
   end
>> extracted_too

extracted_too =

     5     6     4
     2     3     1

>>
```

This can be very useful for extracting rows and columns, as we now describe.

In particular, to extract say row 1 or column 2 of `A`, we need only do

```
>> R1_too = A(1,1:size(A,2))

R1_too =

     1     2     3

>> C2_too = A(1:size(A,1),2)

C2_too =

     2
     5

>>
```

In fact, MATLAB conveniently provides a function `end` which, when it appears in the place of k^{th} index ($k = 1$ or $k = 2$), evaluates to (say for our array `A`) `size(A,k)`. We then can write more succinctly

```
>> R1_too_too = A(1,1:end)

R1_too_too =

     1     2     3

>> R1_too_too_too = A(1,:)

R1_too_too_too =

     1     2     3

>>
```

where in the last line we see that MATLAB admits even further abbreviation: a colon in the place of an index is interpreted as `1:end` for that index.

Finally, there is simple way to create a single-index array from a multi-index array:

```
>> A_single_index = A(:)

A_single_index =

     1
     4
     2
     5
     3
     6

>>
```

Note that it is probably not a good idea to take advantage of the single-index form above as the shape of this single-index array is rather sensitive to how we specify the index argument. (The colon translation is not unique for a single-index array and in fact is interpreted as a particular choice of `reshape`.) We introduce the above just to illustrate the concept of “all arrays are really single-index arrays” and “first index fastest (or column major)” ordering, and also because the single-index reshape is convenient sometimes for certain global operations (see below).

5.3.3 Operations

As regards arithmetic operations, multi-index arrays “behave” in exactly the same fashion as single-index arrays: `-`, `+`, `.*`, `./`, `.^` all perform the necessary element-by-element operations. Indeed, in these operations, the double-index array is essentially treated as a single-index array. (Note that for example `3.2*A` multiplies each element of `A` by 3.2.)The same is true for relational and logical operations (as well as `find`): the operations are performed element by element and the output is a multi-index array of the same size as the two operands.

For data operations, there are more options. Whenever possible, the easiest is to effect the operation in terms of single-index arrays. The colon operator permits us to find the minimum over (say) the first row as

```
>> min(A(1,:))

ans =

     1

>>
```

or in a similar fashion the minimum over any given column.

If we wish to find the minimum over all elements of the entire array, we can interpret the multi-index array in its “underlying” single-index form: for example,

```
>> A = [1,2,3;4,5,6]

A =

     1     2     3
     4     5     6

>> min(A(:))

ans =

     1

>>
```

In most cases, the above simple constructs suffice.

However, it is also easy to apply (say) the `min` function over the first index to yield a row array which contains the minimum over each column, or to perform the `min` function over the second index to yield a column array which contains the minimum over each row:

```
>> min(A,[],1)

ans =

     1     2     3

>> min(A,[],2)

ans =

     1
     4

>>
```

Note the second null argument in `min` above will be explained shortly, when we discuss functions in greater detail. Essentially, `min` takes three arguments, but the second argument is optional and hence if it is not set then MATLAB will not complain. Nulls are useful for optional inputs or for inputs which can be set to default values.

In general the default with MATLAB — when "single-index" functions are applied to multi-index arrays — is to perform the operation over columns to yield a row:

```
>> min(A)

ans =

     1     2     3

>>
```

Note that `min(A)` is not the same as `min(A(:))` in that `A` is of size `[2,3]` whereas `A(:)` is automatically "reshaped" to be a single-index array.

We take this opportunity to revisit the `for` loop. Let's say that we wish to find the number of two-vectors in an $2 \times m$ array which reside in the first quadrant. We can write the `for` loop based on an index as

```
twovecs = [[1;-1],[1;1],[-3;1],[.2;.5]];
num_in_quad_1 = 0;
for j = 1:size(twovecs,2)
   if( twovecs(1,j) >=0 && twovecs(2,j) >=0 )
       num_in_quad_1 = num_in_quad_1 + 1;
   end
end
num_in_quad_1
```

which will work just fine. However, we can also use for our "counter" not an index but rather the data itself, as in

```
twovecs = [[1;-1],[1;1],[-3;1],[.2;.5]];
num_in_quad_1 = 0;
for vec = twovecs;
   if( vec(1) >= 0 && vec(2) >= 0)
       num_in_quad_1 = num_in_quad_1 + 1;
   end
end
num_in_quad_1
```

which also works just fine. In this second form, within the `for` loop, the loop argument `vec` is first set to the first *column* of `twovecs`, `[1;-1]`, `vec` is then set to the second columns of `twovecs`, `[1;1]`, and so on. It is important to note that, for any double-index array, the loop argument is set to each *column* of the array and the loop is executed the number of *column* times. (In particular, the behavior is independent of the column-grouped assignment of `twovecs` used in this example.) This also implies that, if a *column* single-index array is used for the loop construction as in `for i = [1:10]'`, then `i` would be set to the vector `[1:10]'` and the loop is executed just one time. Note that the behavior is completely different from the case of providing a *row* single-index array, as in `for i = 1:10`.

Finally, this seems a good place to note that there are many thousands of Matlab functions and for each oftentimes quite a few options and optional arguments. If you find that you are doing a particular relatively simple operation many times — or a rather complicated operation perhaps only a few times — it is perhaps worthwhile to search for a syntactically succinct and efficient Matlab built-in function which might do the trick. However, in many other cases it will be more effective to write your own code. Matlab built-in functions are a means and not an end.

5.4 Line Plotting

We include line plotting in this chapter as we now have the necessary pieces: single-index arrays, and characters. We do not present all the various options since these can readily be found by `>> help plot` or the documentation. However, we do provide a template which you can use and adapt accordingly.

```
%A sample plotting script - by Justin Miller

%---------- linear-linear plotting, sine and cosines ---------------

L = 2*pi;                           %Define the ending angle
N = 100;                            %Define number of angle segments

xpts = (L/N)*[0:N];                 %Define a set of angles for plotting (in radians)
                                    %This could also be done using
                                    %xpts = linspace(0,L,N+1);

sin_values = sin(xpts);             %Sine vector of each angle
cos_values = cos(xpts);             %Cosine vector of each angle

figure                              %Create a figure window to draw the plots

plot(xpts,sin_values,'b-')          %Plot the sine values in a blue line

hold on                             %Hold the current figure when plotting
                                    %the next figure

plot(xpts,cos_values,'r--')         %Plot the cosine values in a red dashed line

h_sincos_plot = gcf;                %Get the handle of the current figure
ha_sincos_axis = gca;               %Get the handle of the current axis

axis([0,xpts(end),-1.1,1.1])        %Set the x and y axes [xmin,xmax,ymin,ymax]

set(ha_sincos_axis,'XTick',0:pi/2:2*pi) %Set the location of the x tick marks
set(ha_sincos_axis,'YTick',-1:0.2:1)     %Set the location of the y tick marks

set(ha_sincos_axis,'XTickLabel',{'0','pi/2','pi','3*pi/2','2*pi'})
                                    %Set the names of each x tick mark
```

```
xlabel('Angle (radians)')           %Give a label name to the x axis
ylabel('Trigonomic output')         %Give a label name to the y axis

title(['Plot of sin and cos from x = ',num2str(xpts(1)), ...
       ' to x = ',num2str(xpts(end))])
                                    %Give the figure a title

legend('sin','cos','location','best') %Provide a legend and tell matlab to place
                                      %it in the best location

saveas(h_sincos_plot,'sin_cos.fig') %Take the figure specified by handle
                                    %"h_sincos_plot" and save it
                                    %as "sin_cos.fig" in the working directory

%----------- log-linear plotting, exponential ---------------
clear all

L = 5;
N = 100;

x = (L/N)*[0:N];

y = 2*exp(x);

figure

semilogy(x,y,'b-')             %Create a plot where only the y axis is in log scale
                               %semilogx would plot only the x axis in log scale

xlabel('x')
ylabel('y')

title(['Log-Linear plot of y = 2*exp(x) from x = ',num2str(x(1)), ...
       ' to x = ',num2str(x(end))])

saveas(gcf,'exp.fig')

%----------- log-log plotting, polynomials ---------------
clear all

L = 10^2;
N = 100;

x= (L/N)*[0:N];

y = 2*(x.^3);

figure
```

```
loglog(x,y,'b-')                %Create a plot where both axes are in log scale

xlabel('x')
ylabel('y')

title(['Log-Log plot of y = 2x^3 from x = ',num2str(x(1)), ...
      ' to x = ',num2str(x(end))])

saveas(gcf,'poly.fig')
```

Matlab also has extensive “3-D” plotting capabilities.

Chapter 6

Functions in Matlab

6.1 The Advantage: Encapsulation and Re-Use

As you know, a mathematical function is a “rule” which given some inputs (or arguments), returns an output or outputs. A Matlab function (or a function in any programming language), is very similar: the function, given some inputs/arguments, returns an output or outputs. There are two main reasons that functions are so important and useful as a programming construct: re-use and encapsulation. First, re-use: if we perform a particular set of operations — for example, calculation of $sin(x)$ for given x — we prefer not to re-write this same code over and over again. Rather, we write it once and then use it over and over again, with enormous savings. Second, encapsulation: a user can take advantage of the function, and the “function” it performs — from inputs to outputs — without knowing how this function has been implemented or what is “inside” the code; from another perspective, what happens inside the function does not affect the user’s higher level objectives — the output is the entire “effect” (we discuss this further below).

We have already taken extensive advantage of both re-use and encapsulation: we have used many Matlab built-in functions in all of our examples above; and we have used these functions not only without knowing “what is inside” but in fact without even knowing how a function is defined syntactically. In actual practice, it would not be good to proceed in quite such a trusting fashion.

6.2 Always Test a Function

Functions have serious implications as regards the correctness of results and the control of errors. From the positive side, the fact that the program is re-used many times, and developed once intentionally for re-use, means that typically most bugs will have been identified and fixed. From the negative side, encapsulation means that the user typically will not know what is inside, and hence can not personally vouch for correctness. The latter is, fortunately, quite easy to address in a reasonable if not rigorous fashion: confronted with any new function, it is alway worthwhile to consider several test cases — for which you know the answer — to confirm correct behavior; the more authoritative the source, the more the function has been used, the simpler the task, perhaps the less tests required. But remember that you are not just testing the code, you are also testing your understanding of the inputs and outputs.

Note that is often difficult to find a test case in which we know the answer. In particular in the numerical context there is an artifice by which to side-step this issue. In particular, it is often possible to posit the answer and then easily determine the question (which yields this answer). For example, if we wish to test a code which finds the roots of a fourth-order polynomial, for any particular fourth-order polynomial it is not easy to deduce the correct answer (or we would not need the code in the first place). However, it is easy to posit the four roots — the answer — and multiply out to obtain the polynomial with these roots — the question. We then test the code by going backwards (in fact forwards), and verifying that the question leads to the answer. Note such a test does not confirm that the code works in all cases; in fact, we have only confirmed one case. The numerical approach (or even the logic of the code) could thus be flawed in certain or even many instances. However, we have confirmed that we are using the code correctly, that we understand what in principle should happen, and that in at least one (or several) nontrivial cases that what should happen does indeed happen. You should always test a new function in this fashion.

We emphasize that our discussion here applies both to the many MATLAB "built-in" functions — functions bundled with the MATLAB core — and any third-party of user-defined function. Also in the remainder of this chapter many of the details are relevant both to built-in and user functions — for example, how to call a multi-output, multi-input function; however some details, such as how to create a function, are obviously only important for user-defined functions.

6.3 What Happens in a Function Stays in a Function

When we launch a script (or a function) from the command window we may view this program as the "main" program (to use a somewhat archaic term). The (command-window) workspace of this main program is the set of variables of which the main program is aware (and which have been assigned at some point during the execution). When this main program calls a function, say *function_name*, we can view the process as the creation of a second virtual processor, as shown in Figure 6.1 2. In particular, it is critical to note that the two workspaces — the variables assigned by the main program and by *function_name* — are distinct: the main program is unaware of the variables in workspace_*function_name* and *function_name* is unaware of the variables in workspace_command-window. The *only* connection between these two virtual processes are the inputs and outputs: *function_name* receives the inputs from the main program, and the main program receives the outputs from *function_name*; note the direction of the arrows — inputs are not affected by *function_name*. Furthermore, workspace_*function_name* will *disappear* when execution of *function_name* is completed and there will be no permanent record of workspace_*function_name* (unless of course you have written data to a file).

We make several comments. First, there are in fact ways to share variables between the workspaces (global variables), however it is best to avoid global variables if possible since with proliferation they become a sort of programming duct tape. Second, although our picture is for a main program and a function, the same picture applies when one function calls another function (which may call another function, . . .). In such a case the left virtual processor is associated to the "calling program" more generally (e.g., a calling function) and the right virtual processor is associated to the "called function." Again, the critical point is that the workspaces of these two functions are distinct. Third, our picture of Figure 6.1 2 is a good mental model, but not necessarily representative of actual implementation. It is perhaps more realistic to envision an operation in which the "state" of the calling program is saved, the called function "takes over" the processor, and then the calling program is moved back in when the called function has returned control. This picture suggests that it is rather expensive to call a function, and that is indeed the case in particular in MATLAB ; for this reason, it is good to construct functions which, within their designated

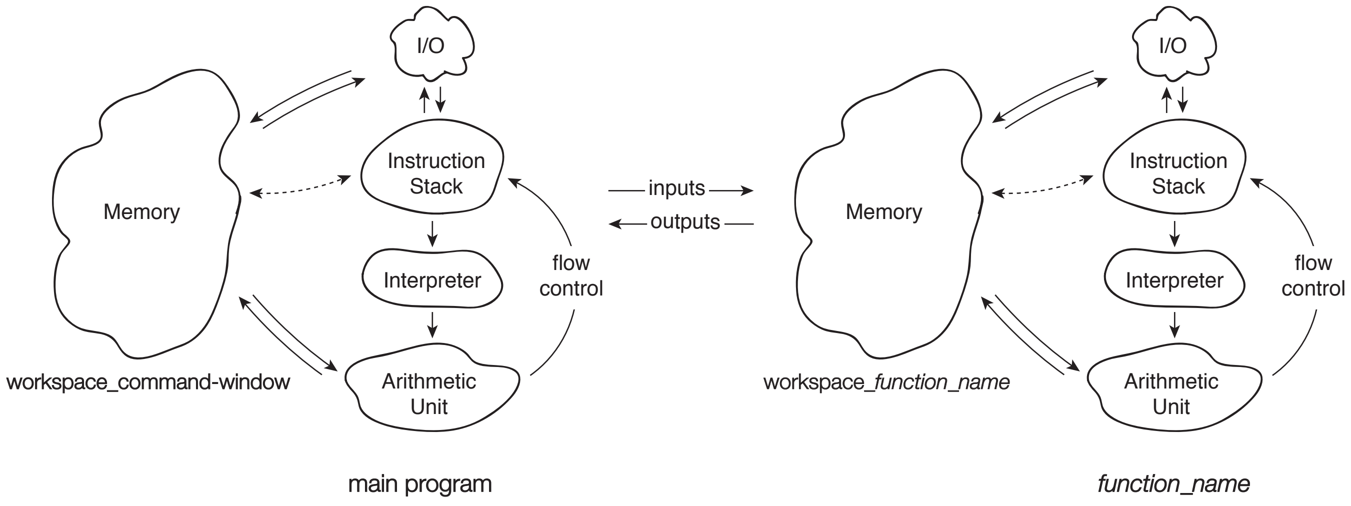


Figure 6.1: Two Virtual Processors

task, compute as much data as possible with each call — for example, operate on arrays rather than scalars — so as to minimize the number of calls. (Note this is not a recommendation to put many different unrelated tasks or many lines of instructions within a single function since obviously this compromises re-use, encapsulation, and efficiency. You do not want to do too much; you just want to operate on as much data as possible.) We discuss this further below.

6.4 Syntax: Inputs (Parameters) and Outputs

Different languages require different syntax for the definition and use (call) of a function. We first consider the former and then the latter. By way of example, we present below the function `x_to_the_2p` which given x evaluates the function (in this case, literally a mathematical function) x^{2p}.

```
function [ value ] = x_to_the_2p( x, p )

value = x.^(2*p);

end
```

The first line declares that this script is a function, that the output will be returned in a variable `value`, that the function name — and also the name of the .m file in which the function is stored — is `x_to_the_2p`, and that the function takes two arguments, `x` and `p`. Next follows the body of the function which produces the output, `value`. The function closes with an `end` statement.

We note that our little function takes a single-index (or even multi-index) array as input. In general, as described above, function calls can be expensive, and hence it is best to generate as much data as possible with each call so as to minimize the number of calls required. For that reason, it is often advantageous to define functions such that the (appropriate) inputs and outputs are arrays rather than scalars. (In our example above, `value` will be of the same size as `x`. This is realized automatically through the assignment statement.) Of course scalars will be a special case of arrays and hence the function may still be called with scalar arguments.

More generally the syntax for a function with `J` inputs and `K` outputs is

```
function [output_1, output_2, . . . , output_K] = function_name(input_1, input_2, ..., input_J)
BODY of FUNCTION
end
```

Note that we may have not only multiple inputs but also multiple outputs. All outputs must be defined within the *BODY of FUNCTION* or MATLAB will complain.

The operation of the function is fairly clear. First our little example (which we call here from the command window, but of course could also be called from a "main" program or another function program):

```
>> clear all
>> y = x_to_the_2p( [1,2], 2)

y =

     1    16

>> value
??? Undefined function or variable 'value'.
 >>
```

Note in the above our function is evaluated and the output assigned to the variable `y`. The variable `value` is internal to the function and the calling program — in this case the function is called from the command-line mode and hence the calling program variables are simply the workspace — has no knowledge of this "dummy" variable.

More generally, we call a function with `J` outputs and `K` inputs as `[`*output_1, output_2, . . . , output_J*`] =` *function_name*`(`*input_1, input_2, . . . , input_J*`);` . (Note that if we omit the semicolon then our outputs will all be displayed in the command window.) Upon being called by the calling program, *function_name* executes *BODY of FUNCTION* for the values of the input arguments passed to *function_name* and then upon reaching the `end` statement *function_name* returns the outputs — and control — to the calling program. Note is possible to force an early return of a function (to the calling program) before the `end` statement is encountered with a `return` statement within the *BODY of FUNCTION*.

It is possible to request only the first K' outputs as `[`*output_1, output_2, . . . , output_K′*`] =` *function_name*`(`*input_1, input_2, . . . , input_J*`);`. Particularly useful is the case in which you only require the first output, as in this case you can directly use the function through composition within a larger expression with the intermediary of assignment. Up to this point, with the exception of `min`, we have considered only single-output functions (or in any event only asked for the first output) in our examples — and for precisely this composition reason. Another example here:

```
>> z = x_to_the_2p( [1,2], 2) + [2,3]

z =

     3    19

>>
```

Note it is important to distinguish between multiple outputs and arrays. An array corresponds to a particular output, not multiple outputs; in our example above, there is a single output, which happens to be a single-index array of length 2.

It is also possible to call a function without all inputs specified, either with `[]` (null) entries or simply with a truncated list — the first J' inputs. However, in this case, it is important that within the function all inputs that will be encountered are defined. In this regard, the Matlab function `isempty` is useful (for nulls) and the Matlab `nargin` is useful (for truncated argument lists) in order to detect any "unset" inputs which must be assigned to default values. (Note different programming languages have different mechanisms for dealing with defaults.) There is also an `nargout` Matlab function useful in tailoring the outputs provided.

6.5 Functions of Functions: Handles

It is often the case that we wish to pass a function to a function: in other words, we wish a called function to be able to operate not just on different data but also on different "input functions." To make matters more concrete (and avoid using the word function too many times with reference to different entities), consider the function `f_o_diff`:

```
function [ value ] = f_o_diff ( func, x, delta_x )

value = (func (x + delta_x) - func (x))./delta_x;

end
```

This little function calculates the first-order finite difference approximation to a function *func* at the point `x` for a given segment-length `delta_x`. Obviously we could include the definition of *func* within `f_o_diff`, but then we would need to have a *different* derivative function for each function we wished to differentiate. In contrast, `f_o_diff` can be re-used for any function *func* — clearly much preferred. (Note we could now perform a much more systematic investigation of round-off error; in our earlier discussion we were not yet armed with functions, or arrays.)

To call `f_o_diff` from a calling program is quite simple with only one wrinkle within the Matlab syntax. In particular, to pass the input function *func* from the calling program to the called function (`f_o_diff`) we do not wish to actually pass the function but rather a kind of pointer — or handle — to where these instructions are stored for use by any (calling) program. (The description here is virtual — a mental model which provides the right intuition. In general, what and how a programming language passes within a function call can be a rather complicated issue.) To create a handle for the function *func* — in other words, to find the pointer to (say) the beginning of the set of instructions which define *func* — we put an "at sign" (@) in front of *func* as in `@`*func*. So for example, to apply `f_o_diff` to the Matlab function `sin` we can either do

```
>> sin_handle = @sin;
>> fprime = f_o_diff( sin_handle, [pi/4, pi/2], .01)

fprime =

   0.7036  -0.0050

>>
```

of more directly

```
>> fprime_too = f_o_diff( @sin, [pi/4, pi/2], .01)

fprime_too = 0.7036 -0.0050

>>
```

Note handles can also be created for other kinds of objects, for example (graphics) figures.

It is often the case that a function *func* we wish to pass to (say) *function_name* is somehow more general — defined with respect to more inputs — than the functions which *function_name* expects. In Matlab there is an easy way to deal with this common occurrence, which we now discuss.

6.6 Anonymous (or In-Line) Functions

A Matlab "anonymous" (or in-line) function is a one-liner with a single output and multiple inputs that can be defined directly in the command window or indeed on the fly in any program (possibly another function). An anonymous function has a very important property: any variables not defined as inputs will be assigned the current values — at the time the anonymous function is created — within the "variable space" (e.g., workspace) of the calling program (e.g., command window).

We provide a concrete example. In particular, we define an anonymous function

```
p = 2;
x_to_the_2p_anon = @(x) x_to_the_2p(x,p);
```

which is identical to `x_to_the_2p` but now a function of single variable, `x`, rather than two variables. The value of `p` is frozen to `2`, though of course more generally we can replace `p = 2` with any expression by which to evaluate `p` in terms of other variables.

To call our anonymous function, we do (following the definition above):

```
>> x_to_the_2p_anon([1,2])

ans =

     1    16

>>
```

The above appears rather pointless, but it serves an important role in passing functions to other functions — in particular in the context of Matlab in which there are many built-in's that require function inputs *of a particular form*.

Let's say that we wish to apply our function `f_o_diff` to our function `x_to_the_2p`. But `f_o_diff` is expecting a function of a single input, `x`, whereas `x_to_the_2p` has two inputs — and two *necessary* inputs, since without `p` we can not evaluate `x_to_the_2p`. This conundrum is easily resolved with inline functions:

```
>> p = 2;
>> x_to_the_2p_anon = @(x) x_to_the_2p(x,p);
>> z = f_o_diff( x_to_the_2p_anon, [1,2], .01 )

z =

    4.0604   32.2408

>>
```

Note that for an in-line function the function "name" is in fact the function handle (hence we need no `@` in front of the `x_to_the_2p_anon` in the above) — the name and handle for a single-line function coalesce.

6.7 String Inputs and the `eval` Function

We note that on occasion we do want the actual function to change — the instructions to be evaluated to change — as we change the inputs. This can be done with the `eval` function. The function `eval` takes as input a string and returns the evaluation of this string given current values for the various variables present in the string; in essence, `eval` is in the interpreter.

For example, we can create the function

```
function [ value ] = f_o_diff_eval ( fstring, x, delta_x )

z = x;
f_x = eval(fstring);
z = x + delta_x;
f_x_plus = eval(fstring);

value = (f_x_plus - f_x)./delta_x;

end
```

which is our finite difference function but now with a string input `fstring` to specify the function to be differentiated. Note that `eval(fstring)` will simply evaluate the expression `fstring` given the current values of the workspace_f_o_diff_eval.

We now call `f_o_diff_eval`:

```
>> fstring = 'z.^4';
>> f_o_diff_eval(fstring,[1,2],.01)

ans =

    4.0604   32.2408

>>
```

which gives us the same result as previously. Note that `f_x = eval(fstring)` in `f_o_diff_eval` for `fstring` as given is equivalent to `f_x = z.^4` but since in the previous line we set `z = x` then `f_x` is assigned `x.^4` as desired. Similarly, two lines down, `f_x_plus` is assigned the appropriate value since we have changed `z` to be `z = x + delta_x`. The user can now specify any desired function (expressed in terms of `z`) without creating a MATLAB function (or anonymous function).

In actual practice there are certainly better ways to accomplish these goals. The purpose of this little section is only to illustrate that on occasions in which we would like to adapt the actual code there are some simple ways to implement this feature.

Chapter 7

Integration

7.1 Integration of Univariate Functions

Our objective is to approximate the value of the integral

$$I = \int_a^b f(x)\,dx \ ,$$

for some arbitrary univariate function $f(x)$. Our approach to this integration problem is to approximate function f by an interpolant $\mathcal{I}f$ and to exactly integrate the interpolant, i.e.

$$I = \sum_{i=1}^{N-1} \int_{S_i} f(x)\,dx \approx \sum_{i=1}^{N-1} \int_{S_i} (\mathcal{I}f)(x)\,dx \equiv I_h \ . \tag{7.1}$$

Recall that in constructing an interpolant, we discretize the domain $[a, b]$ into $N-1$ non-overlapping segments, delineated by segmentation points x_i, $i = 1, \ldots, N$, as illustrated in Figure 7.1[1] Then, we construct a polynomial interpolant on each segment using the function values at the local interpolation points, $\bar{x}^m$, $m = 1, \ldots, M$. These local interpolation points can be mapped to global function evaluation points, $\tilde{x}_i$, $i = 1, \ldots, N_{\text{eval}}$. The quality of the interpolant is dictated by its type and the segment length h, which in turn governs the quality of the integral approximation, I_h. The subscript h on I_h signifies that the integration is performed on a discretization with a segment length h. This integration process takes advantage of the ease of integrating the polynomial interpolant on each segment.

Recalling that the error in the interpolation decreases with h, we can expect the approximation of the integral I_h to approach the true integral I as h goes to 0. Let us formally establish the relationship between the interpolation error bound, $e_{\max} = \max_i e_i$, and the integration error,

[1]For simplicity, we assume h is constant throughout the domain.

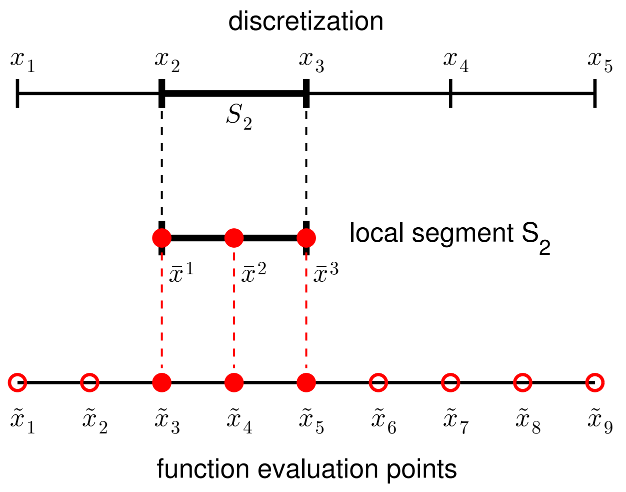


Figure 7.1: Discretization of a 1d domain into $N-1$ (here $N=5$) segments of length h.

$|I - I_h|$.

$$|I - I_h| = \left| \sum_{i=1}^{N-1} \int_{S_i} (f(x) - (\mathcal{I}f)(x))\, dx \right| \tag{7.2}$$

$$\leq \left| \sum_{i=1}^{N-1} \int_{S_i} |f(x) - (\mathcal{I}f)(x)|\, dx \right| \tag{7.3}$$

$$\leq \left| \sum_{i=1}^{N-1} \int_{S_i} e_i\, dx \right| \qquad \text{(local interpolation error bound on } S_i\text{)} \tag{7.4}$$

$$\leq \sum_{i=1}^{N-1} e_i\, h \qquad \text{(definition of } h\text{)} \tag{7.5}$$

$$\leq e_{\max} \sum_{i=1}^{N-1} h \qquad \text{(definition of } e_{\max}\text{)} \tag{7.6}$$

$$= (b-a)e_{\max}\ . \tag{7.7}$$

We make a few observations. First, the global error in the integral is a sum of the local error contributions. Second, since all interpolation schemes considered in Section 2.1 are convergent ($e_{\max} \to 0$ as $h \to 0$), the integration error also vanishes as h goes to zero. Third, while this bound applies to any integration scheme based on interpolation, the bound is not sharp; i.e., some integration schemes would display better convergence with h than what is predicted by the theory.

Recall that the construction of a particular interpolant is only dependent on the location of the interpolation points and the associated function values, $(\tilde{x}_i, f(\tilde{x}_i))$, $i = 1, \ldots, N_{\text{eval}}$, where N_{eval} is the number of the (global) function evaluation points. As a result, the integration rules based on the interpolant is also only dependent on the function values at these N_{eval} points. Specifically, all integration rules considered in this chapter are of the form

$$I_h = \sum_{i=1}^{N_{\text{eval}}} w_i f(\tilde{x}_i) \ ,$$

where w_i are the weights associated with each point and are dependent on the interpolant from which the integration rules are constructed. In the context of numerical integration, the function evaluation points are called *quadrature points* and the associated weights are called *quadrature weights*. The quadrature points and weights together constitute a *quadrature rule*. Not too surprisingly considering the Riemann integration theory, the integral is approximated as a linear combination of the function values at the quadrature points.

Let us provide several examples of integration rules.

Example 7.1.1 rectangle rule, left
The first integration rule considered is a rectangle rule based on the piecewise-constant, left-endpoint interpolation rule considered in Example 2.1.1. Recall the interpolant over each segment is obtained by approximating the value of the function by a constant function that matches the value of the function at the left endpoint, i.e., the interpolation point is $\bar{x}^1 = x_i$ on segment $S_i = [x_i, x_{i+1}]$. The resulting integration formula is

$$I_h = \sum_{i=1}^{N-1} \int_{S_i} (\mathcal{I}f)(x)\,dx = \sum_{i=1}^{N-1} \int_{S_i} f(x_i)\,dx = \sum_{i=1}^{N-1} h f(x_i) \ ,$$

where the piecewise-constant function results in a trivial integration. Recalling that the global function evaluation points, $\tilde{x}_i$, are related to the segmentation points, x_i, by

$$\tilde{x}_i = x_i, \quad i = 1, \dots, N-1 \ ,$$

we can also express the integration rule as

$$I_h = \sum_{i=1}^{N-1} h f(\tilde{x}_i) \ .$$

Figure 7.2(a) illustrates the integration rule applied to $f(x) = \exp(x)$ over $[0, 1]$ with $N = 5$. Recall that for simplicity we assume that all intervals are of the same length, $h \equiv x_{i+1} - x_i$, $i = 1, \dots, N-1$.

Let us now analyze the error associated with this integration rule. From the figure, it is clear that the error in the integrand is a sum of the local errors committed on each segment. The local error on each segment is the triangular gap between the interpolant and the function, which has the length of h and the height proportional to $f'h$. Thus, the local error scales as $f'h^2$. Since there are $(b-a)/h$ segments, we expect the global integration error to scale as

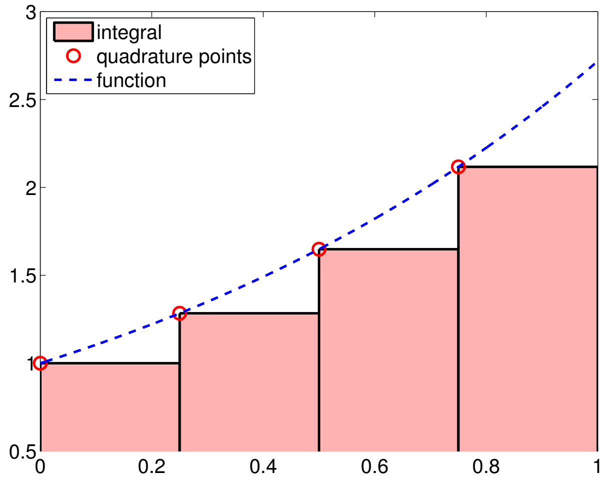
$$|I - I_h| \sim f'h^2(b-a)/h \sim hf' \ .$$

More formally, we can apply the general integration error bound, Eq. (7.7), to obtain

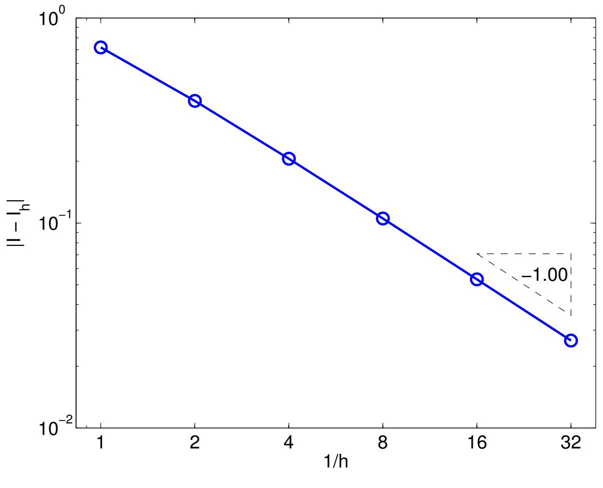
$$|I - I_h| \le (b-a)e_{\max} = (b-a)h \max_{x\in[a,b]} |f'(x)| \ .$$

In fact, this bound can be tightened by a constant factor, yielding

$$|I - I_h| \le (b-a)\frac{h}{2} \max_{x\in[a,b]} |f'(x)| \ .$$



(a) integral



(b) error

Figure 7.2: Rectangle, left-endpoint rule.

Figure 7.2(b) captures the convergence behavior of the scheme applied to the exponential function. As predicted by the theory, this integration rule is *first-order accurate* and the error scales as $\mathcal{O}(h)$. Note also that the approximation I_h underestimates the value of I if $f' > 0$ over the domain.

Before we proceed to a proof of the sharp error bound for a general f, let us analyze the integration error directly for a linear function $f(x) = mx + c$. In this case, the error can be expressed as

$$|I - I_h| = \sum_{i=1}^{N-1} \int_{S_i} f(x) - (\mathcal{I}f)(x)\,dx = \sum_{i=1}^{N-1} \int_{x_i}^{x_{i+1}} (mx - c) - (mx_i - c)\,dx$$

$$= \sum_{i=1}^{N-1} \int_{x_i}^{x_{i+1}} m \cdot (x - x_i)\,dx \; = \sum_{i=1}^{N-1} \frac{1}{2} m (x_{i+1} - x_i)^2$$

$$= \frac{1}{2} mh \sum_{i=1}^{N-1} h = \frac{1}{2} mh(b - a) \; ,$$

Note that the integral of $m \cdot (x - x_i)$ over S_i is precisely equal to the area of the missing triangle, with the base h and the height mh. Because $m = f'(x)$, $\forall\, x \in [a, b]$, we confirm that the general error bound is correct, and in fact sharp, for the linear function. Let us now prove the result for a general f.

Proof. By the fundamental theorem of calculus, we have

$$f(x) - (\mathcal{I}f)(x) = \int_{x_i}^{x} f'(\xi)\,d\xi, \quad x \in S_i = [x_i, x_{i+1}] \; .$$

Integrating the expression over segment S_i and using the Mean Value Theorem,

$$\begin{aligned}
\int_{S_i} f(x) - (\mathcal{I}f)(x)\,dx &= \int_{x_i}^{x_{i+1}} \int_{x_i}^{x} f'(\xi)\,d\xi\,dx \\
&= \int_{x_i}^{x_{i+1}} (x - x_i)\, f'(z)\,dx \qquad \text{(Mean Value Theorem, for some } z \in [x_i, x]) \\
&\le \int_{x_i}^{x_{i+1}} |(x - x_i)\, f'(z)|\,dx \\
&\le \left(\int_{x_i}^{x_{i+1}} |x - x_i|\,dx \right) \max_{z\in[x_i,x_{i+1}]} |f'(z)| \\
&= \frac{1}{2}(x_{i+1} - x_i)^2 \max_{z\in[x_i,x_{i+1}]} |f'(z)| \\
&\le \frac{1}{2} h^2 \max_{x\in[x_i,x_{i+1}]} |f'(x)| \,.
\end{aligned}$$

Summing the local contributions to the integral error,

$$|I - I_h| = \left| \sum_{i=1}^{N-1} \int_{S_i} f(x) - (\mathcal{I}f)(x)\,dx \right| \le \sum_{i=1}^{N-1} \frac{1}{2} h^2 \max_{x\in[x_i,x_{i+1}]} |f'(x)| \le (b-a)\frac{h}{2} \max_{x\in[a,b]} |f'(x)| \,.$$

□

Example 7.1.2 rectangle rule, right
This integration rule is based on the piecewise-constant, right-endpoint interpolation rule considered in Example 2.1.2, in which the interpolation point is chosen as $\bar{x}^1 = x_{i+1}$ on segment $S_i = [x_i, x_{i+1}]$. This results in the integration formula

$$I_h = \sum_{i=1}^{N-1} \int_{S_i} (\mathcal{I}f)(x)\,dx = \sum_{i=1}^{N-1} \int_{S_i} f(x_{i+1})\,dx = \sum_{i=1}^{N-1} h f(x_{i+1}) \,.$$

Recalling that global function evaluation points are related to the segmentation points by $\tilde{x}_i = x_{i+1}$, $i = 1, \ldots, N-1$, we also have

$$I_h = \sum_{i=1}^{N-1} h f(\tilde{x}_i) \,.$$

While the form of the equation is similar to the rectangle rule, left, note that the location of the quadrature points $\tilde{x}_i$, $i = 1, \ldots, N-1$ are different. The integration process is illustrated in Figure 7.1.2

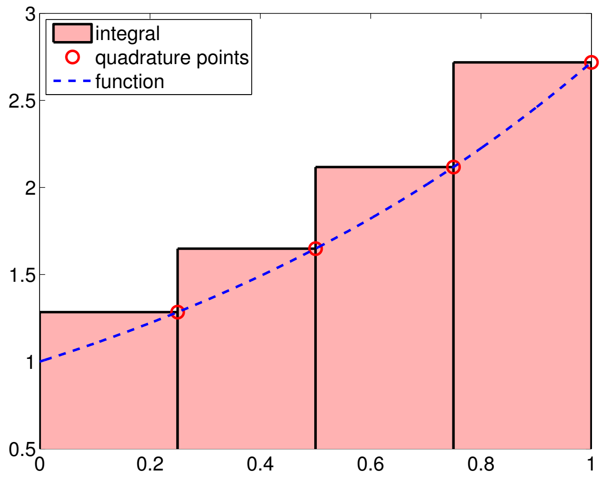


Figure 7.3: Rectangle, right-endpoint rule.

This rule is very similar to the rectangle rule, left. In particular, the integration error is bounded by

$$|I - I_h| \le (b-a)\,\frac{h}{2}\,\max_{x\in[a,b]}\,|f'(x)|\ .$$

The expression shows that the scheme is first-order accurate, and the rule integrates constant function exactly. Even though the error bounds are identical, the left- and right-rectangle rules in general give different approximations. In particular, the right-endpoint rule overestimates I if $f' > 0$ over the domain. The *proof of the error bound* identical to that of the left-rectangle rule.

While the left- and right-rectangle rule are similar for integrating a static function, they exhibit fundamentally different properties when used to integrate an ordinary differential equations. In particular, the left- and right-integration rules result in the Euler forward and backward schemes, respectively. These two schemes exhibit completely different stability properties, which will be discussed in chapters on Ordinary Differential Equations.

Example 7.1.3 rectangle rule, midpoint

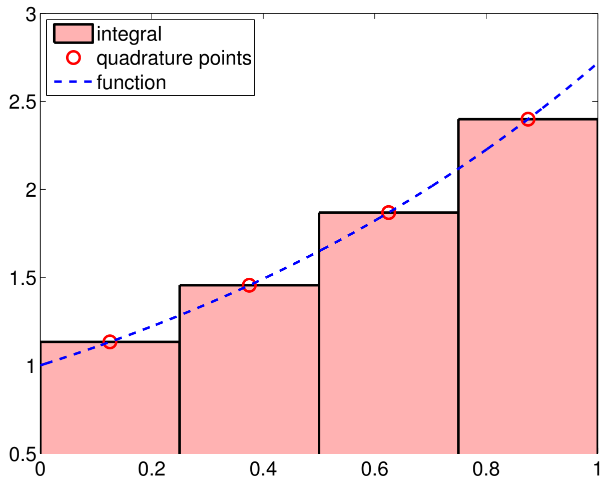
The third integration rule considered is based on the piecewise-constant, midpoint interpolation rule considered in Example 2.1.3. Choosing the midpoint $\bar{x}^1 = (x_i + x_{i+1})$ as the interpolation point for each $S_i = [x_i, x_{i+1}]$, the integration formula is given by

$$I_h = \sum_{i=1}^{N-1}\int_{S_i}(\mathcal{I}f)(x)dx = \sum_{i=1}^{N-1}\int_{S_i} f\left(\frac{x_i + x_{i+1}}{2}\right)dx = \sum_{i=1}^{N-1} hf\left(\frac{x_i + x_{i+1}}{2}\right).$$

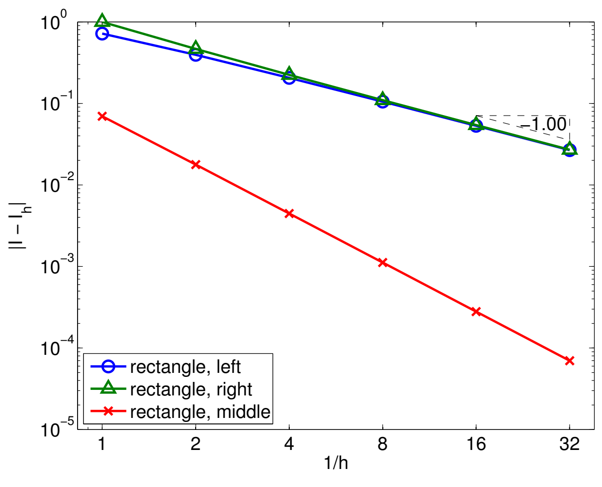
Recalling that the global function evaluation point of the midpoint interpolation is related to the segmentation points by $\tilde{x}_i = (x_i + x_{i+1})/2$, $i = 1, \ldots, N-1$, the quadrature rule can also be expressed as

$$\sum_{i=1}^{N-1} hf(\tilde{x}_i)\ .$$

The integration process is illustrated in Figure 7.4(a).



(a) integral



(b) error

Figure 7.4: Rectangle, midpoint rule.

The error analysis for the midpoint rule is more involved than that of the previous methods. If we apply the general error bound, Eq. (7.7), along with the interpolation error bounds for the midpoint rule, we obtain the error bound of

$$|I - I_h| \le (b-a)\, e_{\max} \le (b-a)\,\frac{h}{2} \max_{x\in[a,b]} |f'(x)| \;.$$

However, this bound is not sharp. The sharp error bound for the rectangle, midpoint integration rule is given by

$$|I - I_h| \le \frac{1}{24}\,(b-a)\,h^2 \max_{x\in[a,b]} |f''(x)| \;.$$

Thus, the rectangle, midpoint rule is *second-order accurate*. The higher accuracy and convergence rate of the midpoint rule are captured in the error convergence plot in Figure 7.4(b).

Before we prove the error bound for a general f, let us show that the rectangle rule in fact integrates a linear function $f(x) = mx + c$ exactly. The integration error for a linear function can be expressed as

$$\begin{aligned} I - I_h &= \sum_{i=1}^{N-1} \int_{S_i} f(x) - (\mathcal{I}f)(x)dx = \sum_{i=1}^{N-1} \int_{x_i}^{x_{i+1}} f(x) - f\left(\frac{x_i + x_{i+1}}{2}\right) dx \\ &= \sum_{i=1}^{N-1} \int_{x_i}^{x_{i+1}} (mx + c) - \left(m \left(\frac{x_i + x_{i+1}}{2}\right) + c \right) dx \\ &= \sum_{i=1}^{N-1} \int_{x_i}^{x_{i+1}} m \left[x - \frac{x_i + x_{i+1}}{2} \right] dx \;. \end{aligned}$$

For convenience, let us denote the midpoint of integral by x_c, i.e., $x_c = (x_i + x_{i+1})/2$. This allows us to express the two endpoints as $x_i = x_c - h/2$ and $x_{i+1} = x_c + h/2$. We split the segment-wise integral at each midpoint, yielding

$$\int_{x_i}^{x_{i+1}} m\left[x - \frac{x_i + x_{i+1}}{2}\right] dx = \int_{x_c-h/2}^{x_c+h/2} m(x - x_c)\, dx$$

$$= \int_{x_c-h/2}^{x_c} m(x - x_c)\, dx + \int_{x_c}^{x_c+h/2} m(x - x_c)\, dx = 0 \ .$$

The first integral corresponds to the (signed) area of a triangle with the base $h/2$ and the height $-mh/2$. The second integral corresponds to the area of a triangle with the base $h/2$ and the height $mh/2$. Thus, the error contribution of these two smaller triangles on S_i cancel each other, and the midpoint rule integrates the linear function exactly.

Proof. For convenience, let us denote the midpoint of segment S_i by x_{m_i}. The midpoint approximation of the integral over S_i is

$$I_h^n = \int_{x_i}^{x_{i+1}} f(x_{m_i})\, dx = \int_{x_i}^{x_{i+1}} f(x_{m_i}) + m(x - x_{m_i})\, dx \ ,$$

for any m. Note that the addition of he linear function with slope m that vanishes at x_{m_i} does not alter the value of the integral. The local error in the approximation is

$$|I^n - I_h^n| = \left| \int_{x_i}^{x_{i+1}} f(x) - f(x_{m_i}) - m(x - x_{m_i})\, dx \right| \ .$$

Recall the Taylor series expansion,

$$f(x) = f(x_{m_i}) + f'(x_{m_i})(x - x_{m_i}) + \frac{1}{2} f''(\xi_i)(x - x_{m_i})^2 \ ,$$

for some $\xi_i \in [x_{m_i}, x]$ (or $\xi_i \in [x, x_{m,n}]$ if $x < x_{m_i}$). Substitution of the Taylor series representation of f and $m = f'(x_{m_i})$ yields

$$|I^n - I_h^n| = \left| \int_{x_i}^{x_{i+1}} \frac{1}{2} f''(\xi_i)(x - x_{m_i})^2\, dx \right| \le \int_{x_i}^{x_{i+1}} \frac{1}{2} |f''(\xi_i)(x - x_{m_i})^2|\, dx$$

$$\le \left(\int_{x_i}^{x_{i+1}} \frac{1}{2}(x - x_{m_i})^2\, dx \right) \max_{\xi \in [x_i, x_{i+1}]} |f''(\xi)| = \left(\frac{1}{6}(x - x_{m_i})^3 \Big|_{x=x_i}^{x_{i+1}} \right) \max_{\xi \in [x_i, x_{i+1}]} |f''(\xi)|$$

$$= \frac{1}{24} h^3 \max_{\xi \in [x_i, x_{i+1}]} |f''(\xi)| \ .$$

Summing the local contributions to the integration error, we obtain

$$|I - I_h| \le \sum_{i=1}^{N-1} \frac{1}{24} h^3 \max_{\xi_i \in [x_i, x_{i+1}]} |f''(\xi_i)| \le \frac{1}{24}(b - a) h^2 \max_{x \in [a,b]} |f''(x)| \ .$$

$\square$

The rectangle, midpoint rule belongs to a family of Newton-Cotes integration formulas, the integration rules with equi-spaced evaluation points. However, this is also an example of Gauss quadrature, which results from picking weights and point locations for optimal accuracy. In particular, k point Gauss quadrature achieves the order $2k$ convergence in one dimension. The midpoint rule is a one-point Gauss quadrature, achieving second-order accuracy.

In the above example, we mentioned that the midpoint rule — which exhibit second-order convergence using just one quadrature point — is an example of Gauss quadrature rules. The Gauss quadrature rules are obtained by choosing both the quadrature points and weights in an "optimal" manner. This is in contrast to Newton-Cotes rules (e.g. trapezoidal rule), which is based on equally spaced points. The "optimal" rule refers to the rule that maximizes the degree of polynomial integrated exactly for a given number of points. In one dimension, the n-point Gauss quadrature integrates $2n-1$ degree polynomial exactly. This may not be too surprising because $2n-1$ degree polynomial has $2n$ degrees of freedom, and n-point Gauss quadrature also gives $2n$ degrees of freedom (n points and n weights).

Example 7.1.4 trapezoidal rule
The last integration rule considered is the trapezoidal rule, which is based on the linear interpolant formed by using the interpolation points $\bar{x}^1 = x_i$ and $\bar{x}^2 = x_{i+1}$ on each segment $S_i = [x_i, x_{i+1}]$. The integration formula is given by

$$\begin{aligned}
I_h &= \sum_{i=1}^{N-1} \int_{S_i} (\mathcal{I}f)(x)\,dx = \sum_{i=1}^{N-1} \int_{S_i} \left[f(x_i) + \left(\frac{f(x_{i+1}) - f(x_i)}{h} \right)(x - x_i) \right] \\
&= \sum_{i=1}^{N-1} \left[f(x_i)h + \frac{1}{2}(f(x_{i+1}) - f(x_i))h \right] \\
&= \sum_{i=1}^{N-1} \frac{1}{2} h (f(x_i) + f(x_{i+1})) \ .
\end{aligned}$$

As the global function evaluation points are related to the segmentation points by $\tilde{x}_i = x_i$, $i = 1, \ldots, N$, the quadrature rule can also be expressed as

$$I_h = \sum_{i=1}^{N-1} \frac{1}{2} h (f(\tilde{x}_i) + f(\tilde{x}_{i+1})) \ ,$$

Rearranging the equation, we can write the integration rule as

$$I_h = \sum_{i=1}^{N-1} \frac{1}{2} h (f(\tilde{x}_i) + f(\tilde{x}_{i+1})) = \frac{1}{2} h f(\tilde{x}_1) + \sum_{i=2}^{N-1} \left[h f(\tilde{x}_i) \right] + \frac{1}{2} h f(\tilde{x}_N) \ .$$

Note that this quadrature rule assigns a different quadrature weight to the quadrature points on the domain boundary from the points in the interior of the domain. The integration rule is illustrated in Figure 7.5(a).

Using the general integration formula, Eq. (7.7), we obtain the error bound

$$|I - I_h| \leq (b-a) e_{\max} = (b-a) \frac{h^2}{8} \max_{x \in [a,b]} |f''(x)| \ .$$

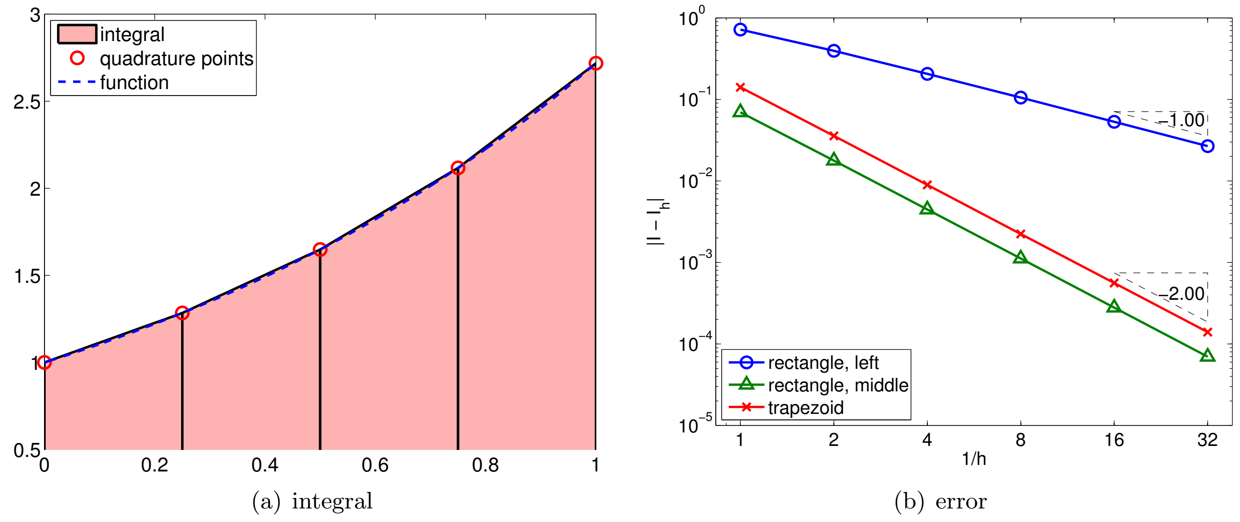


Figure 7.5: Trapezoidal rule.

This bound can be tightened by a constant factor, yielding

$$|I - I_h| \le (b-a)e_{\max} = (b-a)\frac{h^2}{12}\max_{x\in[a,b]}|f''(x)| \ ,$$

which is sharp. The error bound shows that the scheme is second-order accurate.

Proof. To prove the sharp bound of the integration error, recall the following intermediate result from the proof of the linear interpolation error, Eq. (2.3),

$$f(x) - (\mathcal{I}f)(x) = \frac{1}{2}f''(\xi_i)(x-x_i)(x-x_{i+1}) \ ,$$

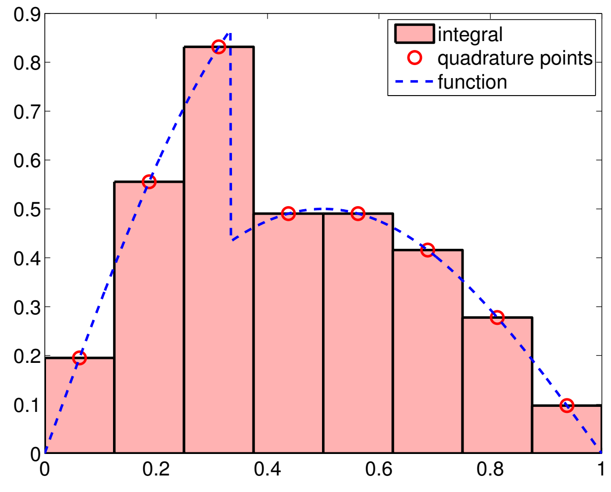
for some $\xi_i \in [x_i, x_{i+1}]$. Integrating the expression over the segment S_i, we obtain the local error representation

$$\begin{aligned} I^n - I_h^n &= \int_{x_i}^{x_{i+1}} f(x) - (\mathcal{I}f)(x)\,dx = \int_{x_i}^{x_{i+1}} \frac{1}{2}f''(\xi_i)(x-x_i)(x-x_{i+1})\,dx \\ &\le \int_{x_i}^{x_{i+1}} \frac{1}{2}|f''(\xi_i)(x-x_i)(x-x_{i+1})|\,dx \le \left(\int_{x_i}^{x_{i+1}} \frac{1}{2}|(x-x_i)(x-x_{i+1})|\,dx\right)\max_{\xi\in[x_i,x_{i+1}]}|f''(\xi)| \\ &= \frac{1}{12}(x_{i+1}-x_i)^3 \max_{\xi\in[x_i,x_{i+1}]}|f''(\xi)| = \frac{1}{12}h^3 \max_{\xi\in[x_i,x_{i+1}]}|f''(\xi)| \ . \end{aligned}$$

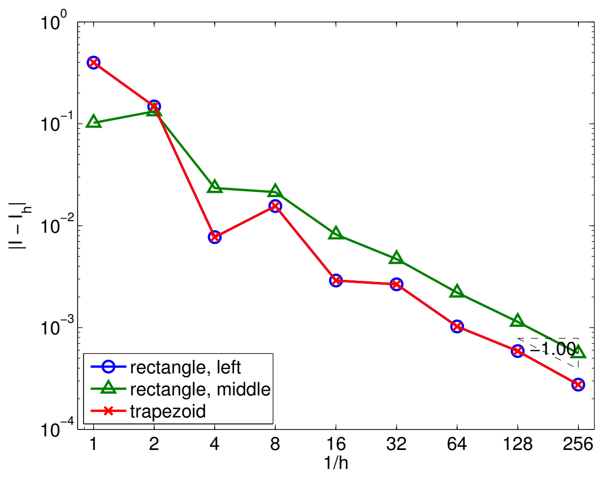
Summing the local errors, we obtain the global error bound

$$|I - I_h| = \left|\sum_{i=1}^{N} \frac{1}{12}h^3 \max_{\xi_i\in[x_i,x_{i+1}]}|f''(\xi_i)|\right| \le \frac{1}{12}(b-a)h^2 \max_{x\in[a,b]}|f''(x)| \ .$$

□



(a) integral



(b) error

Figure 7.6: Integration of a non-smooth function.

Before concluding this section, let us make a few remarks regarding integration of a non-smooth function. For interpolation, we saw that the maximum error can be no better than h^r, where r is the highest order derivative that is defined everywhere in the domain. For integration, the regularity requirement is less stringent. To see this, let us again consider our discontinuous function

$$f(x) = \begin{cases} \sin(\pi x), & x \le \dfrac{1}{3} \\ \dfrac{1}{2}\sin(\pi x), & x > \dfrac{1}{3} \end{cases} .$$

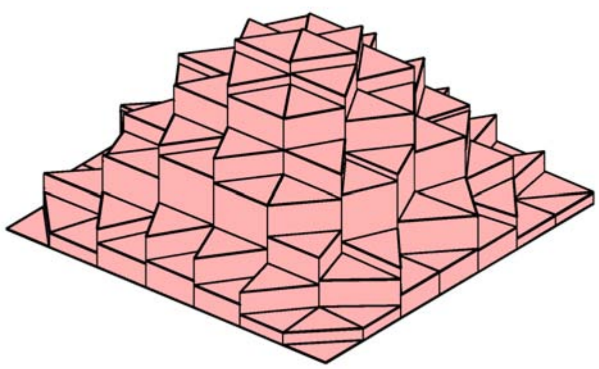
The result of applying the midpoint integration rule with eight intervals is shown in Figure 7.6(a). Intuitively, we can see that the area covered by the approximation approaches that of the true area even in the presence of the discontinuity as $h \to 0$. Figure 7.6(b) confirms that this indeed is the case. All schemes considered converge at the rate of h^1. The convergence rate for the midpoint and trapezoidal rules are reduced to h^1 from h^2. Formally, we can show that the integration schemes converge at the rate of $\min(k, r+1)$, where k is the order of accuracy of the integration scheme for a smooth problem, and r is the highest-order derivative of f that is defined everywhere in the domain. In terms of accuracy, integration smooths and thus helps, whereas differentiation amplifies variations and hence hurts.

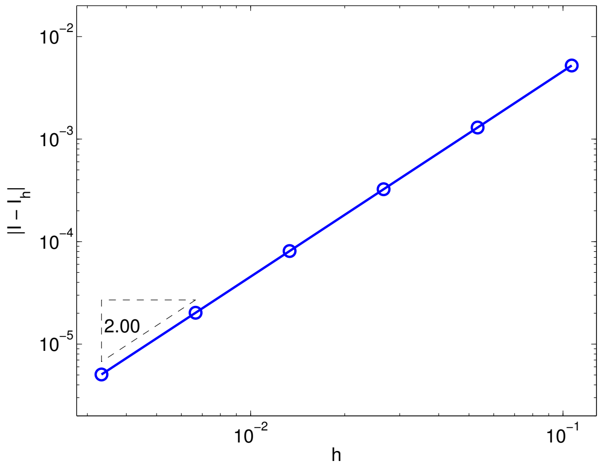
7.2 Integration of Bivariate Functions

Having interpolated bivariate functions, we now consider integration of bivariate functions. We wish to approximate

$$I = \iint_D f(x,y)\,dx\,dy \ .$$

Following the approach used to integrate univariate functions, we replace the function f by its interpolant and integrate the interpolant exactly.


(a) integral



(b) error

Figure 7.7: Midpoint rule.

We triangulate the domain D as shown in Figure 2.15 for constructing interpolants. Then, we approximate the integral as the sum of the contributions from the triangles, $\{R_i\}_{i=1}^N$, i.e.

$$I = \sum_{i=1}^{N} \iint_{R_i} f(x,y)\,dx\,dy \approx \sum_{i=1}^{N} \iint_{R_i} (\mathcal{I}f)(x,y)\,dx\,dy \equiv I_h \ .$$

We now consider two examples of integration rules.

Example 7.2.1 midpoint rule

The first rule is the midpoint rule based on the piecewise-constant, midpoint interpolant. Recall, the interpolant over R_i is defined by the function value at its centroid,

$$(\mathcal{I}f)(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}_i) = f(\boldsymbol{x}_i^c), \quad \forall\, \boldsymbol{x} \in R^n \ ,$$

where the centroid is given by averaging the vertex coordinates,

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i^c = \frac{1}{3}\sum_{i=1}^{3} \boldsymbol{x}_i \ .$$

The integral is approximated by

$$I_h = \sum_{i=1}^{N} \iint_{R_i} (\mathcal{I}f)(x,y)\,dx\,dy = \sum_{i=1}^{N} \iint_{R_i} f(\tilde{x}_i, \tilde{y}_i)\,dx\,dy = \sum_{i=1}^{N} A_i\, f(\tilde{x}_i, \tilde{y}_i) \ ,$$

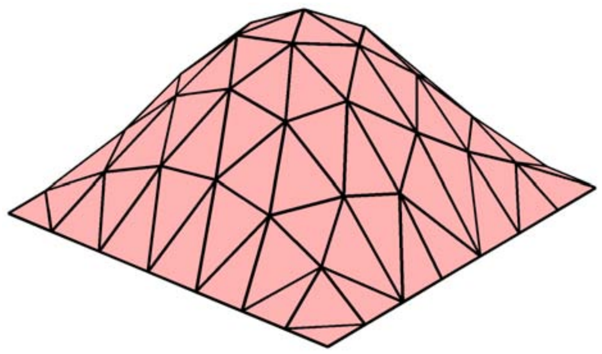
where we have used the fact

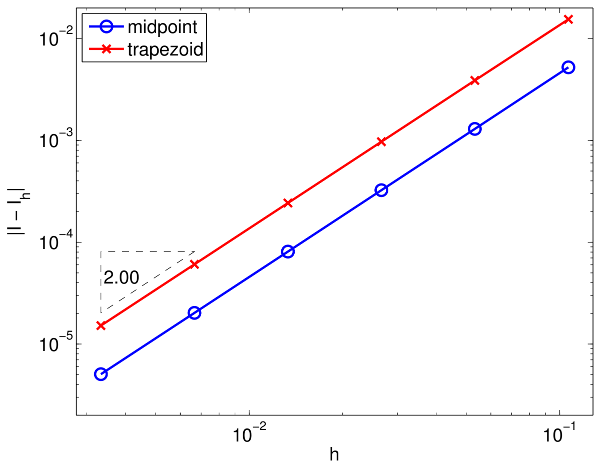
$$\iint_{R_i} dx\,dy = A_i \ ,$$

with A_i denoting the area of triangle R_i. The integration process is shown pictorially in Figure 7.7(a). Note that this is a generalization of the midpoint rule to two dimensions.

The error in the integration is bounded by

$$e \le Ch^2 \|\nabla^2 f\|_F \ .$$


(a) integral



(b) error

Figure 7.8: Trapezoidal rule.

Thus, the integration rule is second-order accurate. An example of error convergence is shown Figure 7.7(b), where the triangles are uniformly divided to produce a better approximation of the integral. The convergence plot confirms the second-order convergence of the scheme.

Similar to the midpoint rule in one dimension, the midpoint rule on a triangle also belongs in the family of Gauss quadratures. The quadrature points and weights are chosen optimally to achieve as high-order convergence as possible.

Example 7.2.2 trapezoidal rule
The trapezoidal-integration rule is based on the piecewise-linear interpolant. Because the integral of a linear function defined on a triangular patch is equal to the average of the function values at its vertices times the area of the triangle, the integral simplifies to

$$I_h = \sum_{i=1}^{N} \left[\frac{1}{3} A_i \sum_{m=1}^{3} f(\bar{\boldsymbol{x}}_i^m) \right] ,$$

where $\{\bar{\boldsymbol{x}}_i^1, \bar{\boldsymbol{x}}_i^2, \bar{\boldsymbol{x}}_i^3\}$ are the vertices of the triangle R_i. The integration process is graphically shown in Figure 7.8(a) .

The error in the integration is bounded by

$$e \le Ch^2 \|\nabla^2 f\|_F \ .$$

The integration rule is second-order accurate, as confirmed by the convergence plot shown in Figure 7.8(b).

The integration rules extend to higher dimensions in principle by using interpolation rules for higher dimensions. However, the number of integration points increases as $(1/h)^d$, where d is the physical dimension. The number of points increases exponentially in d, and this is called the curse of dimensionality. An alternative is to use a integration process based on random numbers, which is discussed in the next unit.

Unit II

Monte Carlo Methods.

Chapter 8

Introduction

8.1 Statistical Estimation and Simulation

8.1.1 Random Models and Phenomena

In science and engineering environments, we often encounter experiments whose outcome cannot be determined with certainty in practice and is better described as random. An example of such a random experiment is a coin flip. The outcome of flipping a (fair) coin is either heads (H) or tails (T), with each outcome having equal probability. Here, we define the probability of a given outcome as the frequency of its occurrence if the experiment is repeated a large number of times.[1] In other words, when we say that there is equal probability of heads and tails, we mean that there would be an equal number of heads and tails if a fair coin is flipped a large (technically infinite) number of times. In addition, we expect the outcome of a random experiment to be unpredictable in some sense; a coin that consistently produces a sequence HTHTHTHT or HHHTTTHHHTTT can be hardly called random. In the case of a coin flip, we may associate this notion of *unpredictability* or *randomness* with the inability to predict with certainty the outcome of the next flip by knowing the outcomes of all preceding flips. In other words, the outcome of any given flip is *independent* of or unrelated to the outcome of other flips.

While the event of heads or tails is random, the distribution of the outcome over a large number of repeated experiments (i.e. the probability density) is determined by non-random parameters. In the case of a coin flip, the sole parameter that dictates the probability density is the probability of heads, which is 1/2 for a fair coin; for a non-fair coin, the probability of heads is (by definition) different from 1/2 but is still some fixed number between 0 and 1.

Now let us briefly consider why the outcome of each coin flip may be considered random. The outcome of each flip is actually governed by a deterministic process. In fact, given a full description of the experiment — the mass and moment of inertia of the coin, initial launch velocity, initial angular momentum, elasticity of the landing surface, density of the air, etc — we can, in principle, predict the outcome of our coin flip by solving a set of deterministic governing equations — Euler's equations for rigid body dynamics, the Navier-Stokes equations for aerodynamics, etc. However, even for something as simple as flipping a coin, the number of variables that describe the state of the system is very large. Moreover, the equations that relate the state to the final outcome (i.e.

[1]We adhere to the *frequentistic* view of probability throughout this unit. We note that the *Baysian* view is an alternative, popular interpretation of probability.

heads or tails) are complicated and the outcome is very sensitive to the conditions that govern the experiments. This renders detailed prediction very difficult, but also suggests that a random model — which considers just the outcome and not the myriad "uncontrolled" ways in which we can observe the outcome — may suffice.

Although we will use a coin flip to illustrate various concepts of probability throughout this unit due to its simplicity and our familiarity with the process, we note that random experiments are ubiquitous in science and engineering. For example, in studying gas dynamics, the motion of the individual molecules is best described using probability distributions. Recalling that 1 mole of gas contains approximately 6×10^{23} particles, we can easily see that deterministic characterization of their motion is impractical. Thus, scientists describe their motion in probabilistic terms; in fact, the macroscale velocity and temperature are parameters that describe the probability distribution of the particle motion, just as the fairness of a given coin may be characterized by the probability of a head. In another instance, an engineer studying the effect of gust on an airplane may use probability distributions to describe the change in the velocity field affected by the gust. Again, even though the air motion is well-described by the Navier-Stokes equations, the highly sensitive nature of turbulence flows renders deterministic prediction of the gust behavior impractical. More importantly, as the engineer is most likely not interested in the detailed mechanics that governs the formation and propagation of the gust and is only interested in its effect on the airplane (e.g., stresses), the gust velocity is best described in terms of a probability distribution.

8.1.2 Statistical Estimation of Parameters/Properties of Probability Distributions

Statistical estimation is a process through which we deduce parameters that characterize the behavior of a random experiment based on a *sample* — a set of typically large but in any event finite number of outcomes of repeated random experiments.[2] In most cases, we postulate a probability distribution — based on some plausible assumptions or based on some descriptive observations such as crude histogram — with several parameters; we then wish to estimate these parameters. Alternatively, we may wish to deduce certain properties — for example, the mean — of the distribution; these properties may not completely characterize the distribution, but may suffice for our predictive purposes. (In other cases, we may need to estimate the full distribution through an empirical cumulative distribution function; We shall not consider this more advanced case in this text.) In Chapter 9, we will introduce a variety of useful distributions, more precisely parametrized discrete probability mass functions and continuous probability densities, as well as various properties and techniques which facilitate the interpretation of these distributions.

Let us illustrate the statistical estimation process in the context of a coin flip. We can flip a coin (say) 100 times, record each observed outcome, and take the mean of the sample — the fraction which are heads — to estimate the probability of heads. We expect from our frequentist interpretation that the sample mean will well approximate the probability of heads. Note that, we can only *estimate* — rather than *evaluate* — the probability of heads because evaluating the probability of heads would require, by definition, an infinite number of experiments. We expect that we can estimate the probability of heads — the sole parameter dictating the distribution of our outcome — with more confidence as the sample size increases. For instance, if we wish to verify the fairness of a given coin, our intuition tells us that we are more likely to deduce its fairness (i.e. the probability of heads equal to 0.5) correctly if we perform 10 flips than 3 flips. The probability of landing HHH using a fair coin in three flips — from which we might incorrectly conclude the coin as unfair — is 1/8, which is not so unlikely, but that of landing HHHHHHHHHH in 10 flips is less than 1 in 1000 trials, which is very unlikely.

[2]We will provide a precise mathematical definition of sample in Chapter 10.

In Chapters 10 and 11, we will introduce a mathematical framework that not only allows us to estimate the parameters that characterize a random experiment but also quantify the confidence we should have in such characterization; the latter, in turn, allows us to make claims — such as the fairness of a coin — with a given level of confidence. We consider two ubiquitous cases: a Bernoulli discrete mass density (relevant to our coin flipping model, for example) in Chapter 10; and the normal density in Chapter 11.

Of course, the ultimate goal of estimation is inference — prediction. In some cases the parameters or quantities estimated, or corresponding "hypothesis tests," in fact suffice for our purposes. In other cases, we can apply the rules of probability to make additional conclusions about how a system will behave and in particular the probability of certain events. In some sense, we go from a finite sample of a population to a probability "law" and then back to inferences about particular events relevant to finite samples from the population.

8.1.3 Monte Carlo Simulation

So far, we have argued that a probability distribution may be effectively used to characterize the outcome of experiments whose deterministic characterization is impractical due to a large number of variables governing its state and/or complicated functional dependencies of the outcome on the state. Another instance in which a probabilistic description is favored over a deterministic description is when their use is computationally advantageous even if the problem is deterministic.

One example of such a problem is determination of the area (or volume) of a region whose boundary is described implicitly. For example, what is the area of a unit-radius circle? Of course, we know the answer is π, but how might we compute the area if we did not know that $A = \pi r^2$? One way to compute the area may be to tessellate (or discretize) the region into small pieces and employ the deterministic integration techniques discussed in Chapter 7. However, application of the deterministic techniques becomes increasingly difficult as the region of interest becomes more complex. For instance, tessellating a volume intersected by multiple spheres is not a trivial task. More generally, deterministic techniques can be increasingly inefficient as the dimension of the integration domain increases.

Monte Carlo methods are better suited for integrating over such a complicated region. Broadly, Monte Carlo methods are a class of computational techniques based on synthetically generating random variables to deduce the implication of the probability distribution. Let us illustrate the idea more precisely for the area determination problem. We first note that if our region of interest is immersed in a unit square, then the area of the region is equal to the probability of a point drawn randomly from the unit square residing in the region. Thus, if we assign a value of 0 (tail) and 1 (head) to the event of drawing a point outside and inside of the region, respectively, approximating the area is equivalent to estimating the probability we land inside (a head). Effectively, we have turned our area determination problem into an statistical estimation problem; the problem is now no different from the coin flip experiment, except the outcome of each "flip" is determined by performing a (simple) check that determines if the point drawn is inside or outside of the region. In other words, we synthetically generate a random variable (by performing the in/out check on uniformly drawn samples) and deduce the implication on the distribution (in this case the area, which is the mean of the distribution). We will study Monte-Carlo-based area integration techniques in details in Chapter 12.

There are several advantages to using Monte Carlo methods compared to deterministic integration approaches. First, Monte Carlo methods are simple to implement: in our case, we do not need to know the domain, we only need to know whether we are in the domain. Second, Monte Carlo methods do not rely on smoothness for convergence — if we think of our integrand as 0 and 1 (depending on outside or inside), our problem here is quite non-smooth. Third, although

Monte Carlo methods do not converge particularly quickly, the convergence rate does not degrade in higher dimensions — for example, if we wished to estimate the volume of a region in a three-dimensional space. Fourth, Monte Carlo methods provide a result, along with a simple built-in error estimator, "gradually" — useful, if not particularly accurate, answers are obtained early on in the process and hence inexpensively and quickly. Note for relatively smooth problems in smooth domains Monte Carlo techniques are not a particularly good idea. Different methods work better in different contexts.

Monte Carlo methods — and the idea of synthetically generating a distribution to deduce its implication — apply to a wide range of engineering problems. One such example is failure analysis. In the example of an airplane flying through a gust, we might be interested in the stress on the spar and wish to verify that the maximum stress anywhere in the spar does not exceed the yield strength of the material — and certainly not the fracture strength so that we may prevent a catastrophic failure. Directly drawing from the distribution of the gust-induced stress would be impractical; the process entails subjecting the wing to various gust and directly measuring the stress at various points. A more practical approach is to instead model the gust as random variables (based on empirical data), propagate its effect through an aeroelastic model of the wing, and synthetically generate the random distribution of the stress. To estimate the properties of the distribution — such as the mean stress or the probability of the maximum stress exceeding the yield stress — we simply need to use a large enough set of realizations of our synthetically generated distribution. We will study the use of Monte Carlo methods for failure analysis in Chapter 14.

Let us conclude this chapter with a practical example of area determination problem in which the use of Monte Carlo methods may be advantageous.

8.2 Motivation: An Example

A museum has enlisted a camera-equipped mobile robot for surveillance purposes. The robot will navigate the museum's premises, pausing to take one or more 360 degree scans in each room. Figure 8.1 shows a typical room filled with various stationary obstructions (in black). We wish to determine the vantage point in each room from which the robot will have the most unobstructed view for its scan by *estimating the visible area (in white) for each candidate vantage point*. We may also wish to provide the robot with an "onboard" visible area estimator for purposes of real-time adaptivity, for example, if the room configuration is temporarily modified. This is a good candidate for Monte Carlo: the domain is complex and non-smooth; we would like quick results based on relatively few evaluations; and we wish to somehow certify the accuracy of our prediction. (In actual practice, the computation would be performed over a three-dimensional museum room — a further reason to consider Monte Carlo.)

We first define, for any vantage point $\boldsymbol{x}_V$ and any surveillance point (to be watched) in the room $\boldsymbol{x}_W$, the line segment $S(\boldsymbol{x}_V, \boldsymbol{x}_W)$ that connects $\boldsymbol{x}_V$ and $\boldsymbol{x}_W$. We can then express the area visible from a vantage point $\boldsymbol{x}_V$ as the integral

$$A(\boldsymbol{x}_V) = \int_{\boldsymbol{x}_W \in R \text{ such that } S(\boldsymbol{x}_V, \boldsymbol{x}_W) \cap O = \varnothing} \mathrm{d}\boldsymbol{x}_W \ , \tag{8.1}$$

where R is the room and O is the collection of obstructions. The visible area is thus defined as the integral over all points in the room such that the line segment $S(\boldsymbol{x}_V, \boldsymbol{x}_W)$ between $\boldsymbol{x}_V$ and $\boldsymbol{x}_W$ does not intersect an obstruction (or, equivalently, such that the intersection of sets S and O is the null set).

There are many ways to do the visibility test $S(\boldsymbol{x}_V, \boldsymbol{x}_W) \cap O \overset{?}{=} \varnothing$, but perhaps the method most amenable to mobile robotics is to use an "occupancy grid," a discretization of the map in which

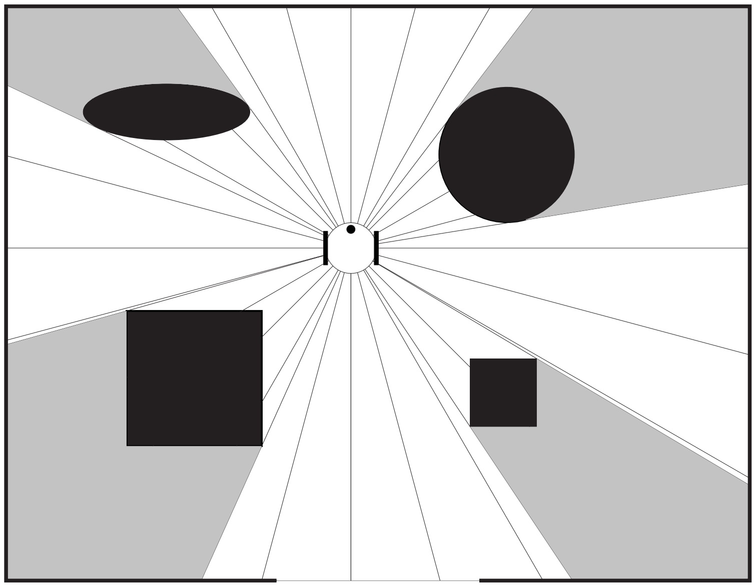


Figure 8.1: A surveillance robot scanning a room. Obstructions (in black) divide the space into visible area (in white) and non-visible area (in gray).

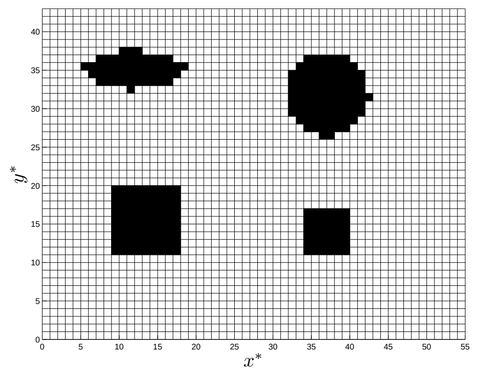


Figure 8.2: Occupancy grid.

each cell's value corresponds to the likelihood that the cell is empty or occupied. We begin by converting our map of the room to an "occupancy grid," a discretization of the map in which each cell's value corresponds to the likelihood that the cell is empty or occupied. In our case, because we know ahead of time the layout of the room, a given cell contains either a zero if the cell is empty, or a one if it is occupied. Figure 8.2 shows a visualization of a fairly low-resolution occupancy grid for our map, where occupied cells are shown in black.

We can use the occupancy grid to determine the visibility of a point $\boldsymbol{x}_W$ in the room from a given vantage point $\boldsymbol{x}_V$. To do this, we draw a line between the two points, determine through which cells the line passes and then check the occupancy condition of each of the intervening cells. If all of the cells are empty, the point is visible. If any of the cells are occupied, the point is not visible. Figure 8.3 shows examples of visible and non-visible cells. Once we have a method for determining if a point is visible or non-visible, we can directly apply our Monte Carlo methods for the estimation of area.

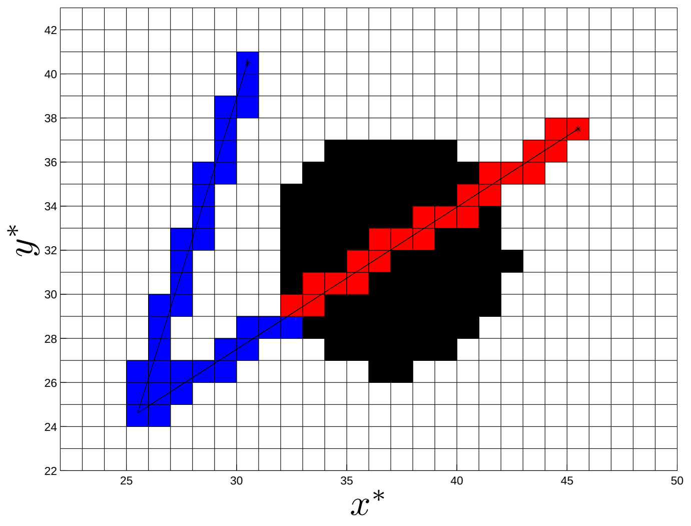


Figure 8.3: Visibility checking of two points from a single vantage point. Visible cells marked in blue, non-visible cells marked in red.

Chapter 9

Introduction to Random Variables

9.1 Discrete Random Variables

9.1.1 Probability Mass Functions

In this chapter, we develop mathematical tools for describing and analyzing *random experiments*, experiments whose outcome cannot be determined with certainty. A coin flip and a roll of a die are classical examples of such experiments. The outcome of a random experiment is described by a *random variable* X that takes on a finite number of values,

$$x_1, \dots, x_J \ ,$$

where J is the number of values that X takes. To fully characterize the behavior of the random variable, we assign a probability to each of these events, i.e.

$$X = x_j, \quad \text{with probability } p_j, \quad j = 1, \dots, J \ .$$

The same information can be expressed in terms of the *probability mass function* (pmf), or discrete density function, f_X, that assigns a probability to each possible outcome

$$f_X(x_j) = p_j, \quad j = 1, \dots, J \ .$$

In order for f_X to be a valid probability density function, $\{p_j\}$ must satisfy

$$0 \le p_j \le 1, \quad j = 1, \dots, J \ ,$$
$$\sum_{j=1}^{J} p_j = 1 \ .$$

The first condition requires that the probability of each event be non-negative and be less than or equal to unity. The second condition states that $\{x_1, \dots, x_J\}$ includes the set of all possible values that X can take, and that the sum of the probabilities of the outcome is unity. The second condition follows from the fact that events x_i, $i = 1, \dots, J$, are *mutually exclusive* and *collectively exhaustive*. *Mutually exclusive* means that X cannot take on two different values of the x_i's in any given experiment. *Collectively exhaustive* means that X must take on one of the J possible values in any given experiment.

Note that for the same random phenomenon, we can choose many different outcomes; i.e. we can characterize the phenomenon in many different ways. For example, $x_j = j$ could simply be a label for the j-th outcome; or x_j could be a numerical value related to some attribute of the phenomenon. For instance, in the case of flipping a coin, we could associate a numerical value of 1 with heads and 0 with tails. Of course, if we wish, we could instead associate a numerical value of 0 with heads and 1 with tails to describe the same experiment. We will see examples of many different random variables in this unit. The key point is that the association of a numerical value to an outcome allows us to introduce meaningful quantitative characterizations of the random phenomenon — which is of course very important in the engineering context.

Let us define a few notions useful for characterizing the behavior of the random variable. The expectation of X, $E[X]$, is defined as

$$E[X] = \sum_{j=1}^{J} x_j p_j \ . \tag{9.1}$$

The expectation of X is also called the mean. We denote the mean by μ or μ_X, with the second notation emphasizing that it is the mean of X. Note that the mean is a weighted average of the values taken by X, where each weight is specified according to the respective probability. This is analogous to the concept of moment in mechanics, where the distances are provided by x_j and the weights are provided by p_j; for this reason, the mean is also called the first moment. The mean corresponds to the centroid in mechanics.

Note that, in frequentist terms, the mean may be expressed as the sum of values taken by X over a large number of realizations divided by the number of realizations, i.e.

$$(\text{Mean}) = \lim_{(\#\ \text{Realizations})\to\infty} \frac{1}{(\#\ \text{Realizations})} \sum_{j=1}^{J} x_j \cdot (\#\ \text{Occurrences of } x_j) \ .$$

Recalling that the probability of a given event is defined as

$$p_j = \lim_{(\#\ \text{Realizations})\to\infty} \frac{(\#\ \text{Occurrences of } x_j)}{(\#\ \text{Realizations})} \ ,$$

we observe that

$$E[X] = \sum_{j=1}^{J} x_j p_j,$$

which is consistent with the definition provided in Eq. (9.1). Let us provide a simple gambling scenario to clarity this frequentist interpretation of the mean. Here, we consider a "game of chance" that has J outcomes with corresponding probabilities p_j, $j = 1, \ldots, J$; we denote by x_j the (net) pay-off for outcome j. Then, in n_{plays} plays of the game, our (net) income would be

$$\sum_{j=1}^{J} x_j \cdot (\#\ \text{Occurrences of } x_j) \ ,$$

which in the limit of large n_{plays} (= # Realizations) yields $n_{\text{plays}} \cdot E[X]$. In other words, the mean $E[X]$ is the expected pay-off per play of the game, which agrees with our intuitive sense of the mean.

The variance, or the second moment about the mean, measures the spread of the values about the mean and is defined by

$$\mathrm{Var}[X] \equiv E[(X-\mu)^2] = \sum_{j=1}^{J} (x_j - \mu)^2 p_j \ .$$

We denote the variance as σ^2. The variance can also be expressed as

$$\begin{aligned}\mathrm{Var}[X] = E[(X-\mu)^2] &= \sum_{j=1}^{J} (x_j - \mu)^2 p_j = \sum_{j=1}^{J} (x_j^2 - 2x_j\mu + \mu^2) p_j \\ &= \underbrace{\sum_{j=1}^{J} x_j^2 p_j}_{E[X^2]} - 2\mu \underbrace{\sum_{j=1}^{J} x_j p_j}_{\mu} + \mu^2 \underbrace{\sum_{j=1}^{J} p_j}_{1} = E[X^2] - \mu^2 \ .\end{aligned}$$

Note that the variance has the unit of X squared. Another useful measure of the expected spread of the random variable is standard deviation, σ, which is defined by

$$\sigma = \sqrt{\mathrm{Var}[X]} \ .$$

We typically expect departures from the mean of many standard deviations to be rare. This is particularly the case for random variables with large range, i.e. J large. (For discrete random variables with small J, this spread interpretation is sometimes not obvious simply because the range of X is small.) In case of the aforementioned "game of chance" gambling scenario, the standard deviation measures the likely (or expected) deviation in the pay-off from the expectation (i.e. the mean). Thus, the standard deviation can be related in various ways to risk; high standard deviation implies a high-risk case with high probability of large payoff (or loss).

The mean and variance (or standard deviation) provide a convenient way of characterizing the behavior of a probability mass function. In some cases the mean and variance (or even the mean alone) can serve as parameters which completely determine a particular mass function. In many other cases, these two properties may not suffice to completely determine the distribution but can still serve as useful measures from which to make further deductions.

Let us consider a few examples of discrete random variables.

Example 9.1.1 rolling a die

As the first example, let us apply the aforementioned framework to rolling of a die. The random variable X describes the outcome of rolling a (fair) six-sided die. It takes on one of six possible values, $1, 2, \ldots, 6$. These events are mutually exclusive, because a die cannot take on two different values at the same time. Also, the events are exhaustive because the die must take on one of the six values after each roll. Thus, the random variable X takes on one of the six possible values,

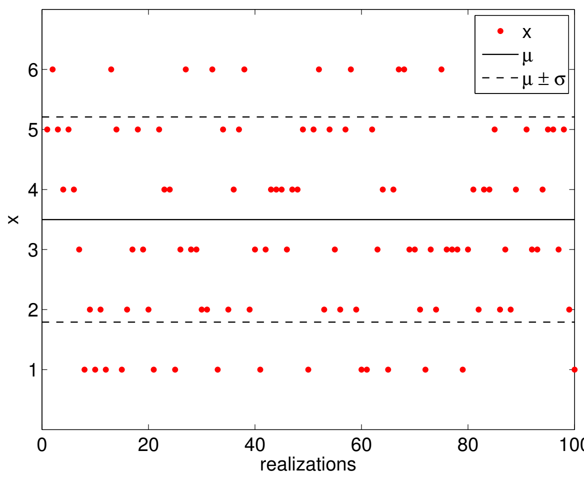
$$x_1 = 1,\ x_2 = 2,\ \ldots,\ x_6 = 6 \ .$$

A fair die has the equal probability of producing one of the six outcomes, i.e.

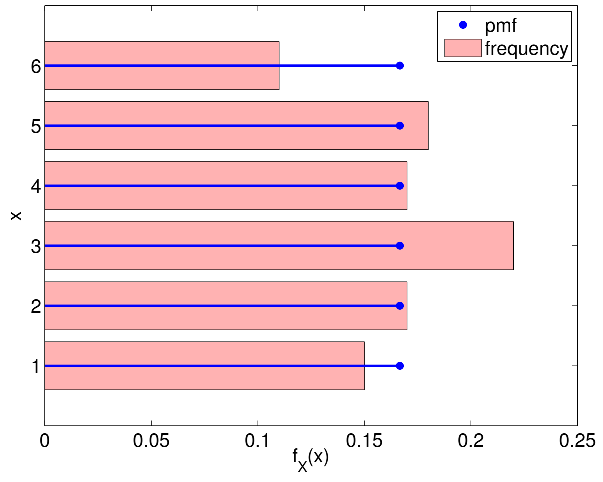
$$X = x_j = j, \quad \text{with probability } \frac{1}{6}, \quad j = 1, \ldots, 6 \ ,$$

or, in terms of the probability mass function,

$$f_X(x) = \frac{1}{6}, \quad x = 1, \ldots, 6 \ .$$



(a) realization



(b) pmf

Figure 9.1: Illustration of the values taken by a fair six-sided die and the probability mass function.

An example of outcome of a hundred die rolls is shown in Figure 9.1(a). The die always takes on one of the six values, and there is no obvious inclination toward one value or the other. This is consistent with the fact that any one of the six values is equally likely. (In practice, we would like to think of Figure 9.1(a) as observed outcomes of actual die rolls, i.e. data, though for convenience here we use synthetic data through random number generation (which we shall discuss subsequently).)

Figure 9.1(b) shows the probability mass function, f_X, of the six equally likely events. The figure also shows the relative frequency of each event — which is defined as the number of occurrences of the event normalized by the total number of samples (which is 100 for this case) — as a histogram. Even for a relatively small sample size of 100, the histogram roughly matches the probability mass function. We can imagine that as the number of samples increases, the relative frequency of each event gets closer and closer to its value of probability mass function.

Conversely, if we have an experiment with an unknown probability distribution, we could infer its probability distribution through a large number of trials. Namely, we can construct a histogram, like the one shown in Figure 9.1(b), and then construct a probability mass function that fits the histogram. This procedure is consistent with the *frequentist interpretation* of probability: the probability of an event is the relative frequency of its occurrence in a large number of samples. The inference of the underlying probability distribution from a limited amount of data (i.e. a small sample) is an important problem often encountered in engineering practice.

Let us now characterize the probability mass function in terms of the mean and variance. The mean of the distribution is

$$\mu = E[X] = \sum_{j=1}^{6} x_j p_j = \sum_{j=1}^{6} j \cdot \frac{1}{6} = \frac{7}{2} \, .$$

The variance of the distribution is

$$\sigma^2 = \mathrm{Var}[X] = E[X^2] - \mu^2 \sum_{j=1}^{6} x_j^2 p_j - \mu^2 = \sum_{j=1}^{6} j^2 \cdot \frac{1}{6} - \left(\frac{7}{2}\right)^2 = \frac{91}{6} - \frac{49}{4} = \frac{35}{12} \approx 2.9167 \, ,$$

and the standard deviation is

$$\sigma = \sqrt{\mathrm{Var}[X]} = \sqrt{\frac{35}{12}} \approx 1.7078 \ .$$

Example 9.1.2 (discrete) uniform distribution
The outcome of rolling a (fair) die is a special case of a more general distribution, called the (discrete) uniform distribution. The uniform distribution is characterized by each event having the equal probability. It is described by two integer parameters, a and b, which assign the lower and upper bounds of the sample space, respectively. The distribution takes on $J = b - a + 1$ values. For the six-sided die, we have $a = 1$, $b = 6$, and $J = b - a + 1 = 6$. In general, we have

$$\begin{aligned} x_j &= a + j - 1, \quad j = 1, \ldots, J \ , \\ f^{\text{disc.uniform}}(x) &= \frac{1}{J} \ . \end{aligned}$$

The mean and variance of a (discrete) uniform distribution are given by

$$\mu = \frac{a+b}{2} \quad \text{and} \quad \sigma^2 = \frac{J^2 - 1}{12} \ .$$

We can easily verify that the expressions are consistent with the die rolling case with $a = 1$, $b = 6$, and $J = 6$, which result in $\mu = 7/2$ and $\sigma^2 = 35/12$.

Proof. The mean follows from

$$\begin{aligned} \mu = E[X] &= E[X - (a-1) + (a-1)] = E[X - (a-1)] + a - 1 \\ &= \sum_{j=1}^{J} (x_j - (a-1)) p_j + a - 1 = \sum_{j=1}^{J} j \frac{1}{J} + a - 1 = \frac{1}{J} \frac{J(J+1)}{2} + a - 1 \\ &= \frac{b - a + 1 + 1}{2} + a - 1 = \frac{b+a}{2} \ . \end{aligned}$$

The variance follows from

$$\begin{aligned} \sigma^2 = \mathrm{Var}[X] &= E[(X - E[X])^2] = E[((X - (a-1)) - E[X - (a-1)])^2] \\ &= E[(X - (a-1))^2] - E[X - (a-1)]^2 \\ &= \sum_{j=1}^{J} (x_j - (a-1))^2 p_j - \left[\sum_{j=1}^{J} (x_j - (a-1)) p_j \right]^2 \\ &= \sum_{j=1}^{J} j^2 \frac{1}{J} - \left[\sum_{j=1}^{J} j p_j \right]^2 = \frac{1}{J} \frac{J(J+1)(2J+1)}{6} - \left[\frac{1}{J} \frac{J(J+1)}{2} \right]^2 \\ &= \frac{J^2 - 1}{12} = \frac{(b - a + 1)^2 - 1}{12} \ . \end{aligned}$$

$\square$

·

Example 9.1.3 Bernoulli distribution (a coin flip)
Consider a classical random experiment of flipping a coin. The outcome of a coin flip is either a head or a tail, and each outcome is equally likely assuming the coin is fair (i.e. unbiased). Without loss of generality, we can associate the value of 1 (success) with head and the value of 0 (failure) with tail. In fact, the coin flip is an example of a Bernoulli experiment, whose outcome takes on either 0 or 1.

Specifically, a Bernoulli random variable, X, takes on two values, 0 and 1, i.e. $J = 2$, and

$$x_1 = 0 \quad \text{and} \quad x_2 = 1.$$

The probability mass function is parametrized by a single parameter, $\theta \in [0, 1]$, and is given by

$$f_{X_\theta}(x) = f^{\text{Bernoulli}}(x; \theta) \equiv \begin{cases} 1-\theta, & x = 0 \\ \theta, & x = 1 \end{cases} .$$

In other words, θ is the probability that the random variable X_θ takes on the value of 1. Flipping of a fair coin is a particular case of a Bernoulli experiment with $\theta = 1/2$. The $\theta = 1/2$ case is also a special case of the discrete uniform distribution with $a = 0$ and $b = 1$, which results in $J = 2$. Note that, in our notation, f_{X_θ} is the probability mass function associated with a particular random variable X_θ, whereas $f^{\text{Bernoulli}}(\cdot; \theta)$ is a family of distributions that describe Bernoulli random variables. For notational simplicity, we will not explicitly state the parameter dependence of X_θ on θ from hereon, unless the explicit clarification is necessary, i.e. we will simply use X for the random variable and f_X for its probability mass function. (Also note that what we call a random variable is of course our choice, and, in the subsequent sections, we often use variable B, instead of X, for a Bernoulli random variable.)

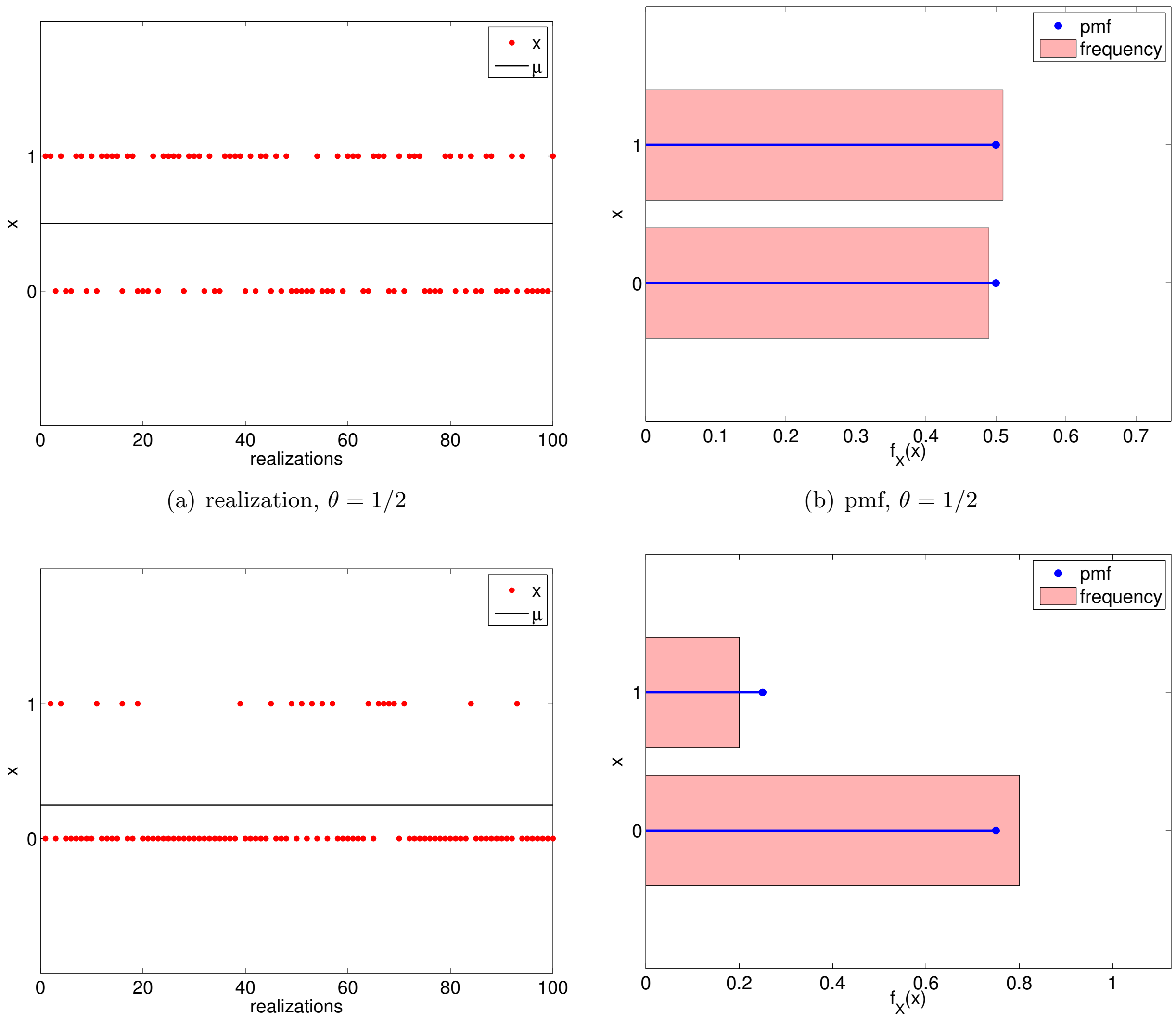
Examples of the values taken by Bernoulli random variables with $\theta = 1/2$ and $\theta = 1/4$ are shown in Figure 9.2. As expected, with $\theta = 1/2$, the random variable takes on the value of 0 and 1 roughly equal number of times. On the other hand, $\theta = 1/4$ results in the random variable taking on 0 more frequently than 1.

The probability mass functions, shown in Figure 9.2, reflect the fact that $\theta = 1/4$ results in X taking on 0 three times more frequently than 1. Even with just 100 samples, the relative frequency histograms captures the difference in the frequency of the events for $\theta = 1/2$ and $\theta = 1/4$. In fact, even if we did not know the underlying pmf — characterized by θ in this case — we can infer from the sampled data that the second case has a lower probability of success (i.e. $x = 1$) than the first case. In the subsequent chapters, we will formalize this notion of inferring the underlying distribution from samples and present a method for performing the task.

The mean and variance of the Bernoulli distribution are given by

$$E[X] = \theta \quad \text{and} \quad \text{Var}[X] = \theta(1-\theta) .$$

Note that lower θ results in a lower mean, because the distribution is more likely to take on the value of 0 than 1. Note also that the variance is small for either $\theta \to 0$ or $\theta \to 1$ as in these cases we are almost sure to get one or the other outcome. But note that (say) $\sigma/E(X)$ scales as $1/\sqrt{(\theta)}$ (recall σ is the standard deviation) and hence the *relative* variation in X becomes *more* pronounced for small θ: this will have important consequences in our ability to predict rare events.



(a) realization, $\theta = 1/2$

(b) pmf, $\theta = 1/2$

(c) realization, $\theta = 1/4$

(d) pmf, $\theta = 1/4$

Figure 9.2: Illustration of the values taken by Bernoulli random variables and the probability mass functions.

Proof. Proof of the mean and variance follows directly from the definitions. The mean is given by

$$\mu = E[X] = \sum_{j=1}^{J} x_j p_j = 0 \cdot (1-\theta) + 1 \cdot \theta = \theta \ .$$

The variance is given by

$$\text{Var}[X] = E[(X-\mu)^2] = \sum_{j=1}^{J} (x_j - \mu)^2 p_j = (0-\theta)^2 \cdot (1-\theta) + (1-\theta)^2 \cdot \theta = \theta(1-\theta) \ .$$

□

Before concluding this subsection, let us briefly discuss the concept of "events." We can define an event of A *or* B as the random variable X taking on one of some set of mutually exclusive outcomes x_j in either the set A *or* the set B. Then, we have

$$P(A \text{ or } B) = P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

That is, probability of event A *or* B taking place is equal to double counting the outcomes x_j in both A and B and then subtracting out the outcomes in both A and B to correct for this double counting. Note that, if A and B are mutually exclusive, we have $A \cap B = \emptyset$ and $P(A \cap B) = 0$. Thus the probability of A *or* B is

$$P(A \text{ or } B) = P(A) + P(B), \qquad (A \text{ and } B \text{ mutually exclusive}).$$

This agrees with our intuition that if A and B are mutually exclusive, we would not double count outcomes and thus would not need to correct for it.

9.1.2 Transformation

Random variables, just like deterministic variables, can be transformed by a function. For example, if X is a random variable and g is a function, then a transformation

$$Y = g(X)$$

produces another random variable Y. Recall that we described the behavior of X that takes on one of J values by

$$X = x_j \quad \text{with probability } p_j, \quad j = 1, \ldots, J \ .$$

The associated probability mass function was $f_X(x_j) = p_j$, $j = 1, \ldots, J$. The transformation $Y = g(X)$ yields the set of outcomes y_j, $j = 1, \ldots, J$, where each y_j results from applying g to x_j, i.e.

$$y_j = g(x_j), \quad j = 1, \ldots, J \ .$$

Thus, Y can be described by

$$Y = y_j = g(x_j) \quad \text{with probability } p_j, \quad j = 1, \ldots, J \ .$$

We can write the probability mass function of Y as

$$f_Y(y_j) = f_Y(g(x_j)) = p_j \quad j = 1, \ldots, J \ .$$

We can express the mean of the transformed variable in a few different ways:

$$E[Y] = \sum_{j=1}^{J} y_j f_Y(y_j) = \sum_{j=1}^{J} y_j p_j = \sum_{j=1}^{J} g(x_j) f_X(x_j) \ .$$

The first expression expresses the mean in terms of Y only, whereas the final expression expresses $E[Y]$ in terms of X and g without making a direct reference to Y.

Let us consider a specific example.

Example 9.1.4 from rolling a die to flipping a coin

Let us say that we want to create a random experiment with equal probability of success and failure (e.g. deciding who goes first in a football game), but all you have is a die instead of a coin. One way to create a Bernoulli random experiment is to roll the die, and assign "success" if an odd number is rolled and assign "failure" if an even number is rolled.

Let us write out the process more formally. We start with a (discrete) uniform random variable X that takes on

$$x_j = j, \quad j = 1, \ldots, 6 \ ,$$

with probability $p_j = 1/6$, $j = 1, \ldots, 6$. Equivalently, the probability density function for X is

$$f_X(x) = \frac{1}{6}, \quad x = 1, 2, \ldots, 6 \ .$$

Consider a function

$$g(x) = \begin{cases} 0, & x \in \{1, 3, 5\} \\ 1, & x \in \{2, 4, 6\} \ . \end{cases}$$

Let us consider a random variable $Y = g(X)$. Mapping the outcomes of X, $x_1, \ldots, x_6$, to $y'_1, \ldots, y'_6$, we have

$$\begin{aligned} y'_1 &= g(x_1) = g(1) = 0 \ , \\ y'_2 &= g(x_2) = g(2) = 1 \ , \\ y'_3 &= g(x_3) = g(3) = 0 \ , \\ y'_4 &= g(x_4) = g(4) = 1 \ , \\ y'_5 &= g(x_5) = g(5) = 0 \ , \\ y'_6 &= g(x_6) = g(6) = 1 \ . \end{aligned}$$

We could thus describe the transformed variable Y as

$$Y = y'_j \quad \text{with probability } p_j = 1/6, \quad j = 1, \ldots, 6 \ .$$

However, because $y'_1 = y'_3 = y'_5$ and $y'_2 = y'_4 = y'_6$, we can simplify the expression. Without loss of generality, let us set

$$y_1 = y'_1 = y'_3 = y'_5 = 0 \quad \text{and} \quad y_2 = y'_2 = y'_4 = y'_6 = 1 \ .$$

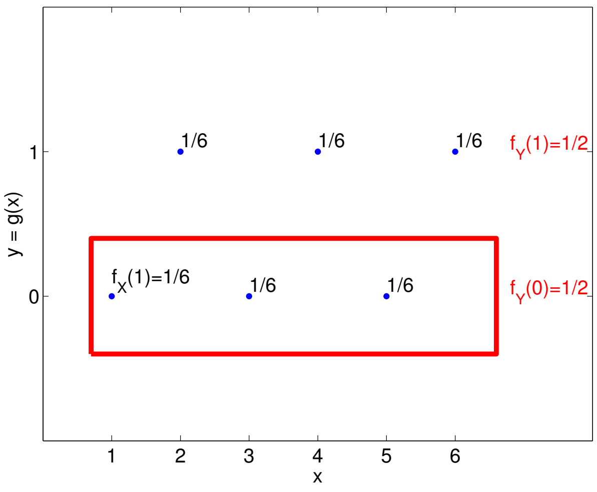


Figure 9.3: Transformation of X associated with a die roll to a Bernoulli random variable Y.

We now combine the frequentist interpretation of probability with the fact that $x_1, \ldots, x_6$ are mutually exclusive. Recall that to a frequentist, $P(Y = y_1 = 0)$ is the probability that Y takes on 0 in a large number of trials. In order for Y to take on 0, we must have $x = 1$, 3, or 5. Because X taking on 1, 3, and 5 are mutually exclusive events (e.g. X cannot take on 1 and 3 at the same time), the number of occurrences of $y = 0$ is equal to the sum of the number of occurrences of $x = 1$, $x = 3$, and $x = 5$. Thus, the relative frequency of Y taking on 0 — or its probability — is equal to the sum of the relative frequencies of X taking on 1, 3, or 5. Mathematically,

$$P(Y = y_1 = 0) = P(X = 1) + P(X = 3) + P(X = 5) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2} \ .$$

Similarly, because X taking on 2, 4, and 6 are mutually exclusive events,

$$P(Y = y_2 = 1) = P(X = 2) + P(X = 4) + P(X = 6) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2} \ .$$

Thus, we have

$$Y = \begin{cases} 0, & \text{with probability } 1/2 \\ 1, & \text{with probability } 1/2 \ , \end{cases}$$

or, in terms of the probability density function,

$$f_Y(y) = \frac{1}{2}, \quad y = 0, 1 \ .$$

Note that we have transformed the uniform random variable X by the function g to create a Bernoulli random variable Y. We emphasize that the mutually exclusive property of $x_1, \ldots, x_6$ is the key that enables the simple summation of probability of the events. In essence, (say), $y = 0$ obtains if $x = 1$ OR if $x = 3$ OR if $x = 5$ (a union of events) and since the events are mutually exclusive the "number of events" that satisfy this condition — ultimately (when normalized) frequency or probability — is the sum of the individual "number" of each event. The transformation procedure is illustrated in Figure 9.3.

Let us now calculate the mean of Y in two different ways. Using the probability density of Y, we can directly compute the mean as

$$E[Y] = \sum_{j=1}^{2} y_j f_Y(y_j) = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2} \ .$$

Or, we can use the distribution of X and the function g to compute the mean

$$E[Y] = \sum_{j=1}^{6} g(x_j) f_X(x_j) = 0 \cdot \frac{1}{6} + 1 \cdot \frac{1}{6} + 0 \cdot \frac{1}{6} + 1 \cdot \frac{1}{6} + 0 \cdot \frac{1}{6} + 1 \cdot \frac{1}{6} = \frac{1}{2} \ .$$

Clearly, both methods yield the same mean.

—————————— · ——————————

9.2 Discrete Bivariate Random Variables (Random Vectors)

9.2.1 Joint Distributions

So far, we have consider scalar random variables, each of whose outcomes is described by a single value. In this section, we extend the concept to random variables whose outcome are vectors. For simplicity, we consider a random vector of the form

$$(X, Y) \ ,$$

where X and Y take on J_X and J_Y values, respectively. Thus, the random vector (X, Y) takes on $J = J_X \cdot J_Y$ values. The probability mass function associated with (X, Y) is denoted by $f_{X,Y}$. Similar to the scalar case, the probability mass function assigns a probability to each of the possible outcomes, i.e.

$$f_{X,Y}(x_i, y_j) = p_{ij}, \quad i = 1, \ldots, J_X, \quad j = 1, \ldots, J_Y \ .$$

Again, the function must satisfy

$$0 \le p_{ij} \le 1, \quad i = 1, \ldots, J_X, \quad j = 1, \ldots, J_Y \ ,$$
$$\sum_{j=1}^{J_Y} \sum_{i=1}^{J_X} p_{ij} = 1 \ .$$

Before we introduce key concepts that did not exist for a scalar random variable, let us give a simple example of joint probability distribution.

Example 9.2.1 rolling two dice
As the first example, let us consider rolling two dice. The first die takes on $x_i = i$, $i = 1, \ldots, 6$, and the second die takes on $y_j = j$, $j = 1, \ldots, 6$. The random vector associated with rolling the two dice is

$$(X, Y) \ ,$$

where X takes on $J_X = 6$ values and Y takes on $J_Y = 6$ values. Thus, the random vector (X, Y) takes on $J = J_X \cdot J_Y = 36$ values. Because (for a fair die) each of the 36 outcomes is equally likely, the probability mass function $f_{X,Y}$ is

$$f_{X,Y}(x_i, y_j) = \frac{1}{36}, \quad i = 1, \ldots, 6, \quad j = 1, \ldots, 6 \ .$$

The probability mass function is shown graphically in Figure 9.4.

—————————— · ——————————

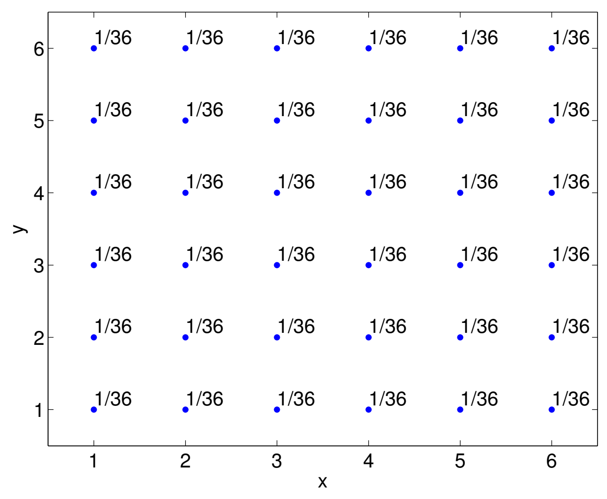


Figure 9.4: The probability mass function for rolling two dice.

.

9.2.2 Characterization of Joint Distributions

Now let us introduce a few additional concepts useful for describing joint distributions. Throughout this section, we consider a random vector (X, Y) with the associated probability distribution $f_{X,Y}$. First is the *marginal density*, which is defined as

$$f_X(x_i) = \sum_{j=1}^{J_Y} f_{X,Y}(x_i, y_j), \quad i = 1, \ldots, J_X \ .$$

In words, marginal density of X is the probability distribution of X disregarding Y. That is, we ignore the outcome of Y, and ask ourselves the question: How frequently does X take on the value x_i? Clearly, this is equal to summing the joint probability $f_{X,Y}(x_i, j_j)$ for all values of y_j. Similarly, the marginal density for Y is

$$f_Y(y_j) = \sum_{i=1}^{J_X} f_{X,Y}(x_i, y_j), \quad j = 1, \ldots, J_Y \ .$$

Again, in this case, we ignore the outcome of X and ask: How frequently does Y take on the value y_j? Note that the marginal densities are valid probability distributions because

$$f_X(x_i) = \sum_{j=1}^{J_Y} f_{X,Y}(x_i, y_j) \le \sum_{k=1}^{J_X} \sum_{j=1}^{J_Y} f_{X,Y}(x_k, y_j) = 1, \quad i = 1, \ldots, J_X \ ,$$

and

$$\sum_{i=1}^{J_X} f_X(x_i) = \sum_{i=1}^{J_X} \sum_{j=1}^{J_Y} f_{X,Y}(x_i, y_j) = 1 \ .$$

The second concept is the *conditional probability*, which is the probability that X takes on the value x_i given Y has taken on the value y_j. The conditional probability is denoted by

$$f_{X|Y}(x_i|y_j), \quad i = 1, \ldots, J_X, \quad \text{for a given } y_j \ .$$

The conditional probability can be expressed as

$$f_{X|Y}(x_i|y_j) = \frac{f_{X,Y}(x_i, y_j)}{f_Y(y_j)} \ .$$

In words, the probability that X takes on x_i given that Y has taken on y_j is equal to the probability that both events take on (x_i, y_j) normalized by the probability that Y takes on y_j disregarding x_i. We can consider a different interpretation of the relationship by rearranging the equation as

$$f_{X,Y}(x_i, y_j) = f_{X|Y}(x_i|y_j) f_Y(y_j) \tag{9.2}$$

and then summing on j to yield

$$f_X(x_i) = \sum_{j=1}^{J_Y} f(x_i, y_j) = \sum_{j=1}^{J_Y} f_{X|Y}(x_i|y_j) f_Y(y_j) \ .$$

In other words, the marginal probability of X taking on x_i is equal to the sum of the probabilities of X taking on x_i given Y has taken on y_j multiplied by the probability of Y taking on y_j disregarding x_i.

From (9.2), we can derive *Bayes' law* (or Bayes' theorem), a useful rule that relates conditional probabilities of two events. First, we exchange the roles of x and y in (9.2), obtaining

$$f_{Y,X}(y_j, x_i) = f_{Y|X}(y_j|x_i) f_X(x_i).$$

But, since $f_{Y,X}(y_j, x_i) = f_{X,Y}(x_i, y_j)$,

$$f_{Y|X}(y_j|x_i) f_X(x_i) = f_{X|Y}(x_i|y_j) f_Y(y_j),$$

and rearranging the equation yields

$$f_{Y|X}(y_j|x_i) = \frac{f_{X|Y}(x_i|y_j) f_Y(y_j)}{f_X(x_i)}. \tag{9.3}$$

Equation (9.3) is called Bayes' law. The rule has many useful applications in which we might know one conditional density and we wish to infer the other conditional density. (We also note the theorem is fundamental to Bayesian statistics and, for example, is exploited in estimation and inverse problems — problems of inferring the underlying parameters of a system from measurements.)

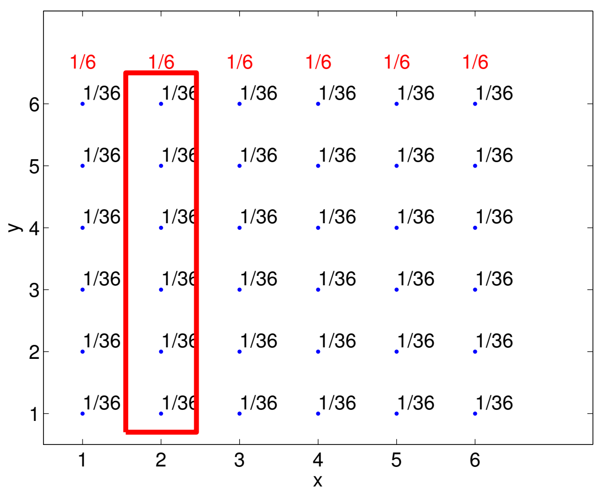
Example 9.2.2 marginal and conditional density of rolling two dice
Let us revisit the example of rolling two dice, and illustrate how the marginal density and conditional density are computed. We recall that the probability mass function for the problem is

$$f_{X,Y}(x, y) = \frac{1}{36}, \quad x = 1, \ldots, 6, \ y = 1, \ldots, 6 \ .$$

The calculation of the marginal density of X is illustrated in Figure 9.5(a). For each x_i, $i = 1, \ldots, 6$, we have

$$f_X(x_i) = \sum_{j=1}^{6} f_{X,Y}(x_i, y_j) = \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} = \frac{1}{6}, \quad i = 1, \ldots, 6 \ .$$

We can also deduce this from intuition and arrive at the same conclusion. Recall that marginal density of X is the probability density of X ignoring the outcome of Y. For this two-dice rolling


(a) marginal density, f_X

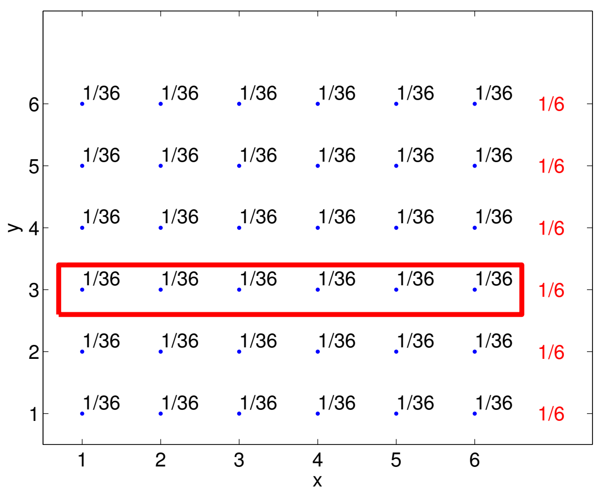

(b) marginal density, f_Y

Figure 9.5: Illustration of calculating marginal density $f_X(x=2)$ and $f_Y(y=3)$.

example, it simply corresponds to the probability distribution of rolling a single die, which is clearly equal to

$$f_X(x) = \frac{1}{6}, \quad x = 1, \ldots, 6 \ .$$

Similar calculation of the marginal density of Y, f_Y, is illustrated in Figure 9.5(b). In this case, ignoring the first die (X), the second die produces $y_j = j$, $j = 1, \ldots, 6$, with the equal probability of $1/6$.

Let us now illustrate the calculation of conditional probability. As the first example, let us compute the conditional probability of X given Y. In particular, say we are given $y = 3$. As shown in Figure 9.6(a), the joint probability of all outcomes except those corresponding to $y = 3$ are irrelevant (shaded region). Within the region with $y = 3$, we have six possible outcomes, each with the equal probability. Thus, we have

$$f_{X|Y}(x|y=3) = \frac{1}{6}, \quad x = 1, \ldots, 6 \ .$$

Note that, we can compute this by simply considering the select set of joint probabilities $f_{X,Y}(x, y = 3)$ and re-normalizing the probabilities by their sum. In other words,

$$f_{X|Y}(x|y=3) = \frac{f_{X,Y}(x, y=3)}{\sum_{i=1}^{6} f_{X,Y}(x_i, y=3)} = \frac{f_{X,Y}(x, y=3)}{f_Y(y=3)} \ ,$$

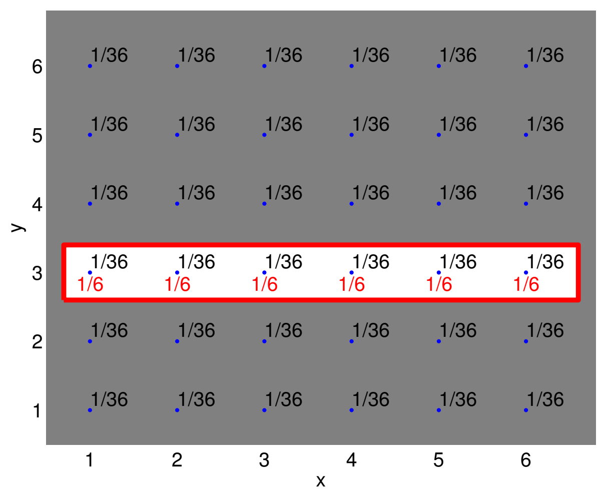
which is precisely equal to the formula we have introduced earlier.

Similarly, Figure 9.6(b) illustrates the calculation of the conditional probability $f_{Y|X}(y, x = 2)$. In this case, we only consider joint probability distribution of $f_{X,Y}(x = 2, y)$ and re-normalize the density by $f_X(x = 2)$.

·

A very important concept is *independence*. Two events are said to be independent if the occurrence of one event does not influence the outcome of the other event. More precisely, two random variables X and Y are said to be independent if their probability density function satisfies

$$f_{X,Y}(x_i, y_j) = f_X(x_i) \cdot f_Y(y_j), \quad i = 1, \ldots, J_X, \quad j = 1, \ldots, J_Y \ .$$


(a) conditional density, $f_{X|Y}(x|y=3)$

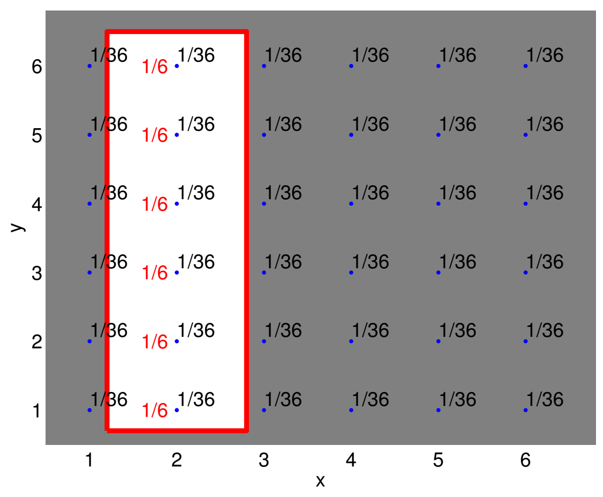

(b) conditional density, $f_{Y|X}(y|x=2)$

Figure 9.6: Illustration of calculating conditional density $f_{X|Y}(x|y=3)$ and $f_{Y|X}(y|x=2)$.

The fact that the probability density is simply a product of marginal densities means that we can draw X and Y separately according to their respective marginal probability and then form the random vector (X,Y).

Using conditional probability, we can connect our intuitive understanding of independence with the precise definition. Namely,

$$f_{X|Y}(x_i|y_j) = \frac{f_{X,Y}(x_i,y_j)}{f_Y(y_j)} = \frac{f_X(x_i)f_Y(y_j)}{f_Y(y_j)} = f_X(x_i) \ .$$

That is, the conditional probability of X given Y is no different from the probability that X takes on x disregarding y. In other words, knowing the outcome of Y adds no additional information about the outcome of X. This agrees with our intuitive sense of independence.

We have discussed the notion of "*or*" and related it to the union of two sets. Let us now briefly discuss the notion of "*and*" in the context of joint probability. First, note that $f_{X,Y}(x,y)$ is the probability that $X=x$ *and* $Y=y$, i.e. $f_{X,Y}(x,y) = P(X=x \textit{ and } Y=y)$. More generally, consider two events A and B, and in particular A *and* B, which is the intersection of A and B, $A\cap B$. If the two events are independent, then

$$P(A \textit{ and } B) = P(A)P(B)$$

and hence $f_{X,Y}(x,y) = f_X(x)f_Y(y)$ which we can think of as probability of $P(A\cap B)$. Pictorially, we can associate event A with X taking on a specified value as marked in Figure 9.5(a) and event B with Y taking on a specified value as marked in Figure 9.5(b). The intersection of A and B is the intersection of the two marked regions, and the joint probability $f_{X,Y}$ is the probability associated with this intersection.

To solidify the idea of independence, let us consider two canonical examples involving coin flips.

Example 9.2.3 independent events: two random variables associated with two independent coin flips

Let us consider flipping two fair coins. We associate the outcome of flipping the first and second coins with random variables X and Y, respectively. Furthermore, we associate the values of 1 and

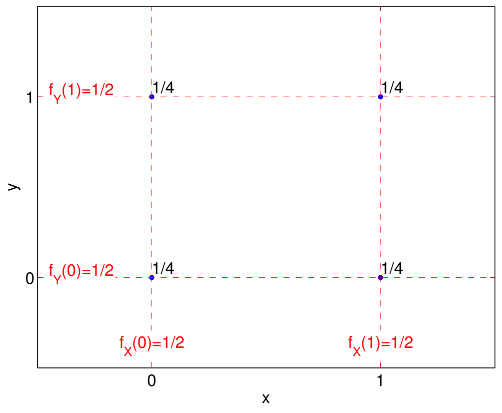


Figure 9.7: The probability mass function for flipping two independent coins.

0 to head and tail, respectively. We can associate the two flips with a random vector (X, Y), whose possible outcomes are

$$(0,0), \quad (0,1), \quad (1,0), \quad \text{and} \quad (1,1) \ .$$

Intuitively, the two variables X and Y will be independent if the outcome of the second flip, described by Y, is not influenced by the outcome of the first flip, described by X, and vice versa.

We postulate that it is equally likely to obtain any of the four outcomes, such that the joint probability mass function is given by

$$f_{X,Y}(x,y) = \frac{1}{4}, \quad (x,y) \in \{(0,0),(0,1),(1,0),(1,1)\} \ .$$

We now show that this assumption implies independence, as we would intuitively expect. In particular, the marginal probability density of X is

$$f_X(x) = \frac{1}{2}, \quad x \in \{0,1\} \ ,$$

since (say) $P(X = 0) = P((X,Y) = (0,0)) + P((X,Y) = (0,1)) = 1/2$. Similarly, the marginal probability density of Y is

$$f_Y(y) = \frac{1}{2}, \quad y \in \{0,1\} \ .$$

We now note that

$$f_{X,Y}(x,y) = f_X(x) \cdot f_Y(y) = \frac{1}{4}, \quad (x,y) \in \{(0,0),(0,1),(1,0),(1,1)\} \ ,$$

which is the definition of independence.

The probability mass function of (X, Y) and the marginal density of X and Y are shown in Figure 9.7. The figure clearly shows that the joint density of (X, Y) is the product of the marginal density of X and Y. Let us show that this agrees with our intuition, in particular by considering the probability of $(X, Y) = (0, 0)$. First, the relative frequency that X takes on 0 is 1/2. Second, of the events in which $X = 0$, 1/2 of these take on $Y = 0$. Note that this probability is independent of the value that X takes. Thus, the relative frequency of X taking on 0 and Y taking on 0 is 1/2 of 1/2, which is equal to 1/4.

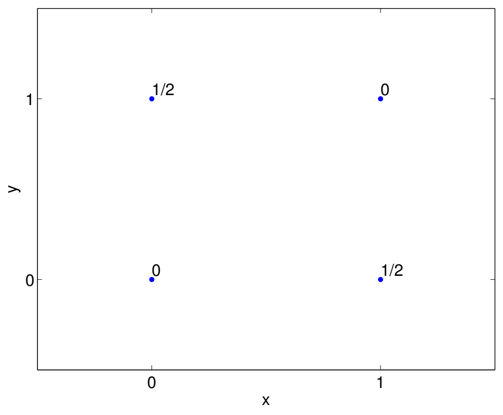


Figure 9.8: The probability mass function for flipping two independent coins.

We can also consider conditional probability of an event that X takes on 1 given that Y takes on 0. The conditional probability is

$$f_{X|Y}(x=1|y=0) = \frac{f_{X,Y}(x=1,y=0)}{f_Y(y=0)} = \frac{1/4}{1/2} = \frac{1}{2} \ .$$

This probability is equal to the marginal probability of $f_X(x=1)$. This agrees with our intuition; given that two events are independent, we gain no additional information about the outcome of X from knowing the outcome of Y.

·

Example 9.2.4 non-independent events: two random variables associated with a single coin flip

Let us now consider flipping a single coin. We associate a Bernoulli random variables X and Y with

$$X = \begin{cases} 1, & \text{head} \\ 0, & \text{tail} \end{cases} \quad \text{and} \quad Y = \begin{cases} 1, & \text{tail} \\ 0, & \text{head} \end{cases} \ .$$

Note that a head results in $(X,Y) = (1,0)$, whereas a tail results in $(X,Y) = (0,1)$. Intuitively, the random variables are not independent, because the outcome of X completely determines Y, i.e. $X + Y = 1$.

Let us show that these two variables are not independent. We are equally like to get a head, $(1,0)$, or a tail, $(0,1)$. We cannot produce $(0,0)$, because the coin cannot be head and tail at the same time. Similarly, $(1,1)$ has probably of zero. Thus, the joint probability density function is

$$f_{X,Y}(x,y) = \begin{cases} \frac{1}{2}, & (x,y) = (0,1) \\ \frac{1}{2}, & (x,y) = (1,0) \\ 0, & (x,y) = (0,0) \text{ or } (x,y) = (1,1) \ . \end{cases}$$

The probability mass function is illustrated in Figure 9.8.

The marginal density of each of the event is the same as before, i.e. X is equally likely to take on 0 or 1, and Y is equally like to take on 0 or 1. Thus, we have

$$f_X(x) = \frac{1}{2}, \quad x \in \{0,1\}$$
$$f_Y(y) = \frac{1}{2}, \quad y \in \{0,1\} \ .$$

For $(x, y) = (0, 0)$, we have

$$f_{X,Y}(x, y) = 0 \neq \frac{1}{4} = f_X(x) \cdot f_Y(y) \ .$$

So, X and Y are not independent.

We can also consider conditional probabilities. The conditional probability of $x = 1$ given that $y = 0$ is

$$f_{X|Y}(x = 1|y = 0) = \frac{f_{X,Y}(x = 1, y = 0)}{f_Y(y = 0)} = \frac{1/2}{1/2} = 1 \ .$$

In words, given that we know Y takes on 0, we know that X takes on 1. On the other hand, the conditional probability of $x = 1$ given that $y = 1$ is

$$f_{X|Y}(x = 0|y = 0) = \frac{f_{X,Y}(x = 0, y = 0)}{f_Y(y = 0)} = \frac{0}{1/2} = 0 \ .$$

In words, given that Y takes on 1, there is no way that X takes on 1. Unlike the previous example that associated (X, Y) with two independent coin flips, we know with certainty the outcome of X given the outcome of Y, and vice versa.

·

We have seen that independence is one way of describing the relationship between two events. Independence is a binary idea; either two events are independent or not independent. Another concept that describes how closely two events are related is correlation, which is a normalized covariance. The covariance of two random variables X and Y is denoted by $\mathrm{Cov}(X, Y)$ and defined as

$$\mathrm{Cov}(X, Y) \equiv E[(X - \mu_X)(Y - \mu_Y)] \ .$$

The correlation of X and Y is denoted by ρ_{XY} and is defined as

$$\rho_{XY} = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y} \ ,$$

where we recall that σ_X and σ_Y are the standard deviation of X and Y, respectively. The correlation indicates how strongly two random events are related and takes on a value between -1 and 1. In particular, two perfectly correlated events take on 1 (or -1), and two independent events take on 0.

Two independent events have zero correlation because

$$\begin{aligned}
\mathrm{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] &= \sum_{j=1}^{J_Y} \sum_{i=1}^{J_X} (x_i - \mu_X)(y_j - \mu_Y) f_{X,Y}(x_i, y_j) \\
&= \sum_{j=1}^{J_Y} \sum_{i=1}^{J_X} (x_i - \mu_X)(y_j - \mu_Y) f_X(x_i) f_Y(y_j) \\
&= \left[\sum_{j=1}^{J_Y} (y_j - \mu_Y) f_Y(y_j) \right] \cdot \left[\sum_{i=1}^{J_X} (x_i - \mu_X) f_X(x_i) \right] \\
&= E[Y - \mu_Y] \cdot E[X - \mu_X] = 0 \cdot 0 = 0 \ .
\end{aligned}$$

The third inequality follows from the definition of independence, $f_{X,Y}(x_i, y_j) = f_X(x_i) f_Y(y_j)$. Thus, if random variables are independent, then they are uncorrelated. However, the converse is not true in general.

9.3 Binomial Distribution

In the previous section, we saw random vectors consisting of two random variables, X and Y. Let us generalize the concept and introduce a random vector consisting of n components

$$(X_1, X_2, \ldots, X_n) \ ,$$

where each X_i is a random variable. In particular, we are interested in the case where each X_i is a Bernoulli random variable with the probability of success of θ. Moreover, we assume that X_i, $i = 1, \ldots, n$, are independent. Because the random variables are independent and each variable has the same distribution, they are said to be *independent and identically distributed* or i.i.d. for short. In other words, if a set of random variables $X_1, \ldots, X_n$ is i.i.d., then

$$f_{X_1,X_2,\ldots,X_n}(x_1, x_2, \ldots, x_n) = f_X(x_1) \cdot f_X(x_2) \cdots f_X(x_n) \ ,$$

where f_X is the common probability density for $X_1, \ldots, X_n$. This concept plays an important role in statistical inference and allows us to, for example, make a probabilistic statement about behaviors of random experiments based on observations.

Now let us transform the i.i.d. random vector $(X_1, \ldots, X_n)$ of Bernoulli random variables to a random variable Z by summing its components, i.e.

$$Z_n = \sum_{i=1}^{n} X_i \ .$$

(More precisely we should write $Z_{n,\theta}$ since Z depends on both n and θ.) Note Z_n is a function of $(X_1, X_2, \ldots, X_n)$, and in fact a simple function — the sum. Because X_i, $i = 1, \ldots, n$, are random variables, their sum is also a random variable. In fact, this sum of Bernoulli random variable is called a *binomial random variable*. It is denoted by $Z_n \sim \mathcal{B}(n, \theta)$ (shorthand for $f_{Z_{n,\theta}}(z) = f^{\text{binomial}}(z; n, \theta)$), where n is the number of Bernoulli events and θ is the probability of success of each event. Let us consider a few examples of binomial distributions.

Example 9.3.1 total number of heads in flipping two fair coins
Let us first revisit the case of flipping two fair coins. The random vector considered in this case is

$$(X_1, X_2) \ ,$$

where X_1 and X_2 are independent Bernoulli random variables associated with the first and second flip, respectively. As in the previous coin flip cases, we associate 1 with heads and 0 with tails. There are four possible outcome of these flips,

$$(0, 0), \quad (0, 1), \quad (1, 0), \quad \text{and} \quad (1, 1) \ .$$

From the two flips, we can construct the binomial distribution $Z_2 \sim \mathcal{B}(2, \theta = 1/2)$, corresponding to the total number of heads that results from flipping two fair coins. The binomial random variable is defined as

$$Z_2 = X_1 + X_2 \ .$$

Counting the number of heads associated with all possible outcomes of the coin flip, the binomial random variable takes on the following value:

First flip	0	0	1	1
Second flip	0	1	0	1
Z_2	0	1	1	2

Because the coin flips are independent and each coin flip has the probability density of $f_{X_i}(x) = 1/2$, $x = 0, 1$, their joint distribution is

$$f_{X_1,X_2}(x_1, x_2) = f_{X_1}(x_1) \cdot f_{X_2}(x_2) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}, \quad (x_1, x_2) \in \{(0,0), (0,1), (1,0), (1,1)\} .$$

In words, each of the four possible events are equally likely. Of these four equally likely events, $Z_2 \sim \mathcal{B}(2, 1/2)$ takes on the value of 0 in one event, 1 in two events, and 2 in one event. Thus, the behavior of the binomial random variable Z_2 can be concisely stated as

$$Z_2 = \begin{cases} 0, & \text{with probability } 1/4 \\ 1, & \text{with probability } 1/2 \ (= 2/4) \\ 2, & \text{with probability } 1/4 \ . \end{cases}$$

Note this example is very similar to Example 9.1.4: Z_2, the sum of X_1 and X_2, is our $g(X)$; we assign probabilities by invoking the mutually exclusive property, OR (union), and summation. Note that the mode, the value that Z_2 is most likely to take, is 1 for this case. The probability mass function of Z_2 is given by

$$f_{Z_2}(x) = \begin{cases} 1/4, & x = 0 \\ 1/2, & x = 1 \\ 1/4, & x = 2 \ . \end{cases}$$

Example 9.3.2 total number of heads in flipping three fair coins
Let us know extend the previous example to the case of flipping a fair coin three times. In this case, the random vector considered has three components,

$$(X_1, X_2, X_3) \ ,$$

with each X_1 being a Bernoulli random variable with the probability of success of 1/2. From the three flips, we can construct the binomial distribution $Z_3 \sim \mathcal{B}(3, 1/2)$ with

$$Z_3 = X_1 + X_2 + X_3 \ .$$

The all possible outcomes of the random vector and the associated outcomes of the binomial distribution are:

First flip	0	1	0	0	1	0	1	1
Second flip	0	0	1	0	1	1	0	1
Third flip	0	0	0	1	0	1	1	1
Z_3	0	1	1	1	2	2	2	3

Because the Bernoulli random variables are independent, their joint distribution is

$$f_{X_1,X_2,X_3}(x_1, x_2, x_3) = f_{X_1}(x_1) \cdot f_{X_2}(x_2) \cdot f_{X_3}(x_3) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8} \ .$$

In other words, each of the eight events is equally likely. Of the eight equally likely events, Z_3 takes on the value of 0 in one event, 1 in three events, 2 in three events, and 3 in one event. The behavior of the binomial variable Z_3 is summarized by

$$Z_3 = \begin{cases} 0, & \text{with probability } 1/8 \\ 1, & \text{with probability } 3/8 \\ 2, & \text{with probability } 3/8 \\ 3, & \text{with probability } 1/8\ . \end{cases}$$

The probability mass function (for $\theta = 1/2$) is thus given by

$$f_{Z_3}(x) = \begin{cases} 1/8, & x = 0 \\ 3/8, & x = 1 \\ 3/8, & x = 2 \\ 1/8, & x = 3\ . \end{cases}$$

·

Example 9.3.3 total number of heads in flipping four fair coins
We can repeat the procedure for four flips of fair coins ($n = 4$ and $\theta = 1/2$). In this case, we consider the sum of the entries of a random vector consisting of four Bernoulli random variables, (X_1, X_2, X_3, X_4). The behavior of $Z_4 = \mathcal{B}(4, 1/2)$ is summarized by

$$Z_4 = \begin{cases} 0, & \text{with probability } 1/16 \\ 1, & \text{with probability } 1/4 \\ 2, & \text{with probability } 3/8 \\ 3, & \text{with probability } 1/4 \\ 4, & \text{with probability } 1/16\ . \end{cases}$$

Note that Z_4 is much more likely to take on the value of 2 than 0, because there are many equally-likely events that leads to $Z_4 = 2$, whereas there is only one event that leads to $Z_4 = 0$. In general, as the number of flips increase, the deviation of $Z_n \sim \mathcal{B}(n, \theta)$ from $n\theta$ becomes increasingly unlikely.

·

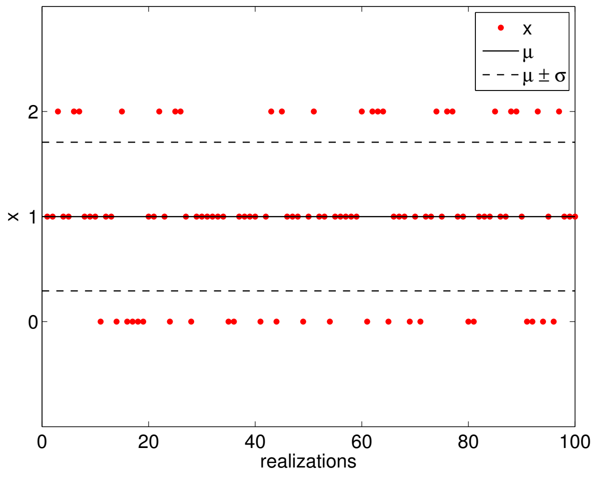
Figure 9.9 illustrates the values taken by binomial random variables for $n = 2$ and $n = 4$, both with $\theta = 1/2$. The histogram confirms that the distribution is more likely to take on the values near the mean because there are more sequences of the coin flips that realizes these values. We also note that the values become more concentrated near the mean, $n\theta$, relative to the range of the values it can take, $[0, n]$, as n increases. This is reflected in the decrease in the standard deviation relative to the width of the range of the values Z_n can take.

In general, a binomial random variable $Z_n \sim \mathcal{B}(n, \theta)$ behaves as

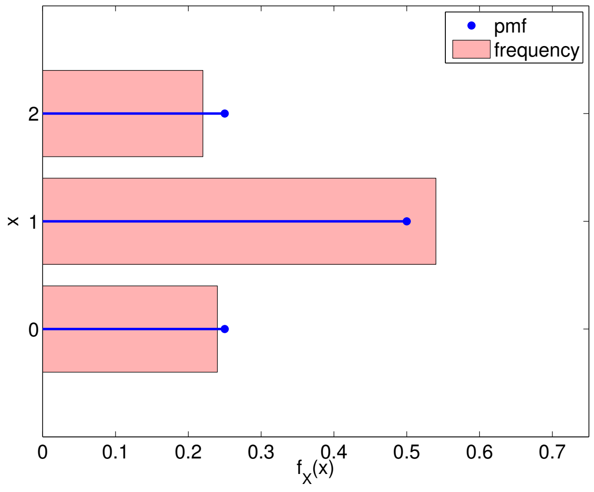
$$Z_n = k, \quad \text{with probability } \begin{pmatrix} n \\ k \end{pmatrix} \theta^k (1-\theta)^{n-k}\ ,$$

where $k = 1, \ldots, n$. Recall that $\begin{pmatrix} n \\ k \end{pmatrix}$ is the binomial coefficient, read "n choose k: the number of ways of picking k unordered outcomes from n possibilities. The value can be evaluated as

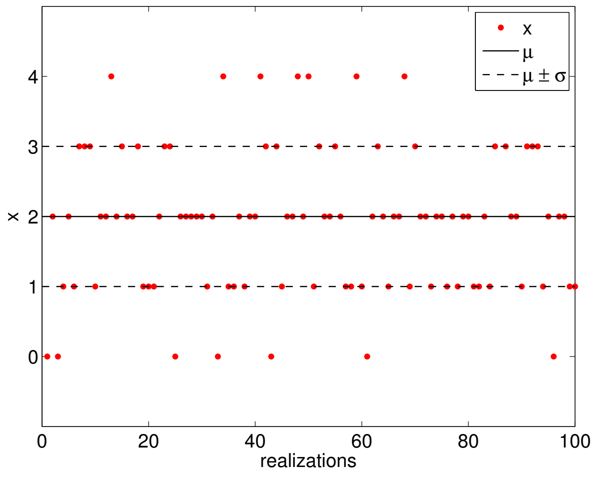
$$\begin{pmatrix} n \\ k \end{pmatrix} \equiv \frac{n!}{(n-k)!k!}\ , \tag{9.4}$$



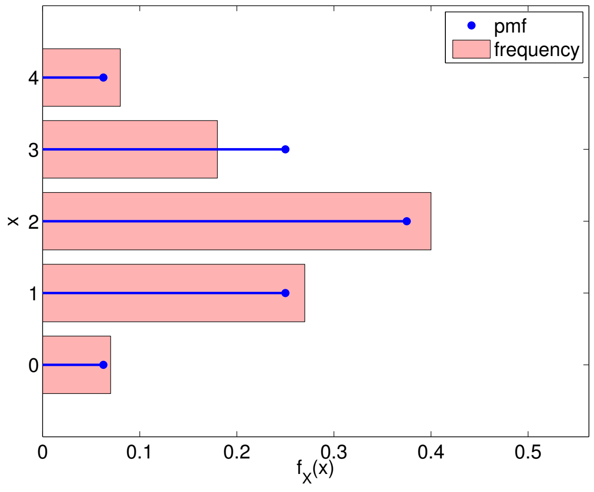
(a) realization, $n = 2$, $\theta = 1/2$



(b) pmf, $n = 2$, $\theta = 1/2$



(c) realization, $n = 4$, $\theta = 1/2$



(d) pmf, $n = 4$, $\theta = 1/2$

Figure 9.9: Illustration of the values taken by binomial random variables.

where ! denotes the factorial.

We can readily derive the formula for $\mathcal{B}(n,\theta)$. We think of n tosses as a binary number with n bits, and we ask how many ways k ones can appear. We can place the first one in n different places, the second one in $n-1$ different places, ..., which yields $n!/(n-k)!$ possibilities. But since we are just counting the number of ones, the order of appearance does not matter, and we must divide $n!/(n-k)!$ by the number of different orders in which we can construct the same pattern of k ones — the first one can appear in k places, the second one in $k-1$ places, ..., which yields $k!$. Thus there are "n choose k" ways to obtain k ones in a binary number of length n, or equivalently "n choose k" different binary numbers with k ones. Next, by independence, each pattern of k ones (and hence $n-k$ zeros) has probability $\theta^k(1-\theta)^{n-k}$. Finally, by the mutually exclusive property, the probability that $Z_n = k$ is simply the number of patterns with k ones multiplied by the probability that each such pattern occurs (note the probability is the same for each such pattern).

The mean and variance of a binomial distribution is given by

$$E[Z_n] = n\theta \quad \text{and} \quad \text{Var}[Z_n] = n\theta(1-\theta) \ .$$

Proof. The proof for the mean follows from the linearity of expectation, i.e.

$$E[Z_n] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \theta = n\theta \ .$$

Note we can readily prove that the expectation of the sum is the sum of the expectations. We consider the n-dimensional sum over the joint mass function of the X_i weighted — per the definition of expectation — by the sum of the X_i, $i = 1, \ldots, n$. Then, for any given X_i, we factorize the joint mass function: $n-1$ of the sums then return unity, while the last sum gives the expectation of X_i. The proof for variance relies on the pairwise independence of the random variables

$$\begin{aligned}\text{Var}[Z_n] = E[(Z_n - E[Z_n])^2] &= E\left[\left\{\left(\sum_{i=1}^{n} X_i\right) - n\theta\right\}^2\right] = E\left[\left\{\sum_{i=1}^{n}(X_i-\theta)\right\}^2\right] \\ &= E\left[\sum_{i=1}^{n}(X_i-\theta)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(X_i-\theta)(X_j-\theta)\right] \\ &= \sum_{i=1}^{n} E[(X_i-\theta)^2] + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\cancel{E[(X_i-\theta)(X_j-\theta)]} \\ &= \sum_{i=1}^{n}\text{Var}[X_i] = \sum_{i=1}^{n}\theta(1-\theta) = n\theta(1-\theta) \ .\end{aligned}$$

The cross terms cancel because coin flips are independent. □

Note that the variance scales with n, or, equivalently, the standard deviation scales with $\sqrt{n}$. This turns out to be the key to Monte Carlo methods — numerical methods based on random variables — which is the focus of the later chapters of this unit.

Let us get some insight to how the general formula works by applying it to the binomial distribution associated with flipping coins three times.

Example 9.3.4 applying the general formula to coin flips
Let us revisit $Z_3 = \mathcal{B}(3, 1/2)$ associated with the number of heads in flipping three coins. The probability that $Z_3 = 0$ is, by substituting $n = 3$, $k = 0$, and $\theta = 1/2$,

$$f_{Z_3}(0) = \begin{pmatrix} n \\ k \end{pmatrix} \theta^k (1-\theta)^{n-k} = \begin{pmatrix} 3 \\ 0 \end{pmatrix} \left(\frac{1}{2}\right)^0 \left(1 - \frac{1}{2}\right)^{3-0} = \frac{3!}{0!(3-0)!} \left(\frac{1}{2}\right)^3 = \frac{1}{8} ,$$

which is consistent with the probability we obtained previously. Let us consider another case: the probability of $Z_3 = 2$ is

$$f_{Z_3}(2) = \begin{pmatrix} n \\ k \end{pmatrix} \theta^k (1-\theta)^{n-k} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \left(\frac{1}{2}\right)^2 \left(1 - \frac{1}{2}\right)^{3-2} = \frac{3!}{2!(3-2)!} \left(\frac{1}{2}\right)^3 = \frac{3}{8} .$$

Note that the $\theta^k(1-\theta)^{n-k}$ is the probability that the random vector of Bernoulli variables

$$(X_1, X_2, \ldots, X_n) ,$$

realizes $X_1 = X_2 = \ldots = X_k = 1$ and $X_{k+1} = \ldots = X_n = 0$, and hence $Z_n = k$. Then, we multiply the probability with the number of different ways that we can realize the sum $Z_n = k$, which is equal to the number of different way of rearranging the random vector. Here, we are using the fact that the random variables are identically distributed. In the special case of (fair) coin flips, $\theta^k(1-\theta)^{n-k} = (1/2)^k(1-1/2)^{n-k} = (1/2)^n$, because each random vector is equally likely.

9.4 Continuous Random Variables

9.4.1 Probability Density Function; Cumulative Distribution Function

Let X be a random variable that takes on any real value in (say) an interval,

$$X \in [a, b] .$$

The *probability density function* (pdf) is a function over $[a, b]$, $f_X(x)$, such that

$$f_X(x) \geq 0, \quad \forall x \in [a, b] ,$$
$$\int_a^b f_X(x)\, dx = 1 .$$

Note that the condition that the probability mass function sums to unity is replaced by an integral condition for the continuous variable. The probability that X take on a value over an infinitesimal interval of length dx is

$$P(x \leq X \leq x + dx) = f_X(x)\, dx ,$$

or, over a finite subinterval $[a', b'] \subset [a, b]$,

$$P(a' \leq X \leq b') = \int_{a'}^{b'} f_X(x)\, dx .$$

In other words, the probability that X takes on the value between a' and b' is the integral of the probability density function f_X over $[a', b']$.

A particular instance of this is a *cumulative distribution function* (cdf), $F_X(x)$, which describes the probability that X will take on a value less than x, i.e.

$$F_X(x) = \int_a^x f_X(x)\, dx \ .$$

(We can also replace a with $-\infty$ if we define $f_X(x) = 0$ for $-\infty < x < a$.) Note that any cdf satisfies the conditions

$$F_X(a) = \int_a^a f_X(x)\, dx = 0 \quad \text{and} \quad F_X(b) = \int_a^b f_X(x)\, dx = 1 \ .$$

Furthermore, it easily follows from the definition that

$$P(a' \le X \le b') = F_X(b') - F_X(a').$$

That is, we can compute the probability of X taking on a value in $[a', b']$ by taking the difference of the cdf evaluated at the two end points.

Let us introduce a few notions useful for characterizing a pdf, and thus the behavior of the random variable. The *mean*, μ, or the expected value, $E[X]$, of the random variable X is

$$\mu = E[X] = \int_a^b f(x)\, x\, dx \ .$$

The *variance*, $\mathrm{Var}(X)$, is a measure of the spread of the values that X takes about its mean and is defined by

$$\mathrm{Var}(X) = E[(X-\mu)^2] = \int_a^b (x-\mu)^2 f(x)\, dx \ .$$

The variance can also be expressed as

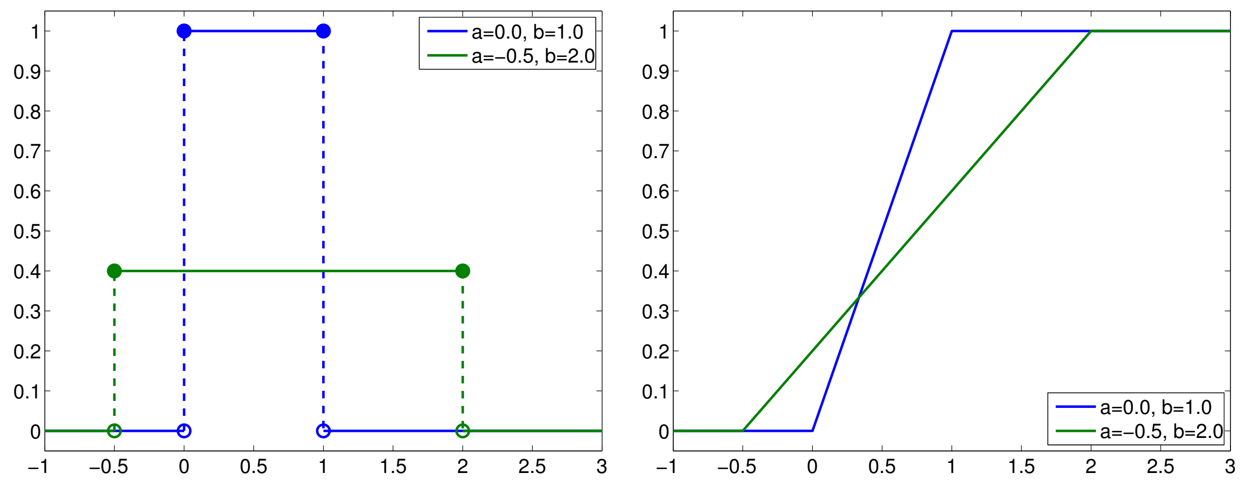
$$\begin{aligned}\mathrm{Var}(X) = E[(X-\mu)^2] &= \int_a^b (x-\mu)^2 f(x)\, dx \\ &= \int_a^b x^2 f(x)\, dx - 2\mu \underbrace{\int_a^b x f(x)\, dx}_{\mu} + \mu^2 \int_a^b f(x)\, dx \\ &= E[X^2] - \mu^2 \ .\end{aligned}$$

The α-th *quantile* of a random variable X is denoted by $\tilde{z}_\alpha$ and satisfies

$$F_X(\tilde{z}_\alpha) = \alpha.$$

In other words, the quantile $\tilde{z}_\alpha$ partitions the interval $[a, b]$ such that the probability of X taking on a value in $[a, \tilde{z}_\alpha]$ is α (and conversely $P(\tilde{z}_\alpha \le X \le b) = 1 - \alpha$). The $\alpha = 1/2$ quantile is the *median*.

Let us consider a few examples of continuous random variables.



(a) probability density function (b) cumulative density function

Figure 9.10: Uniform distributions

Example 9.4.1 Uniform distribution
Let X be a uniform random variable. Then, X is characterized by a constant pdf,

$$f_X(x) = f^{\text{uniform}}(x; a, b) \equiv \frac{1}{b-a} \ .$$

Note that the pdf satisfies the constraint

$$\int_a^b f_X(x)\, dx = \int_a^b f^{\text{uniform}}(x; a, b)\, dx = \int_a^b \frac{1}{b-a}\, dx = 1 \ .$$

Furthermore, the probability that the random variable takes on a value in the subinterval $[a', b'] \in [a, b]$ is

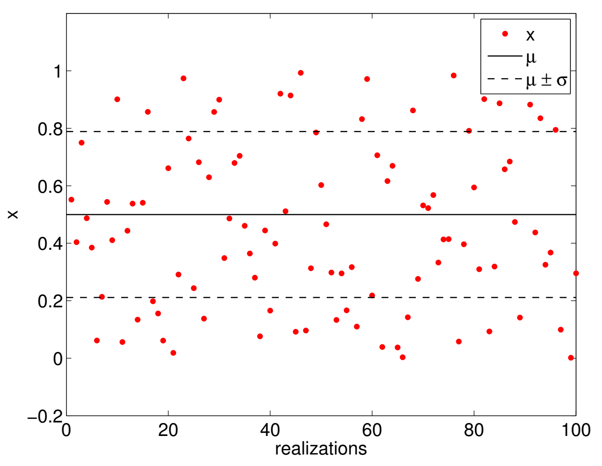
$$P(a' \le X \le b') = \int_{a'}^{b'} f_X(x)\, dx = \int_{a'}^{b'} f^{\text{uniform}}(x; a, b)\, dx = \int_{a'}^{b'} \frac{1}{b-a}\, dx = \frac{b'-a'}{b-a} \ .$$

In other words, the probability that $X \in [a', b']$ is proportional to the relative length of the interval as the density is equally distributed. The distribution is compactly denoted as $\mathcal{U}(a, b)$ and we write $X \sim \mathcal{U}(a, b)$. A straightforward integration of the pdf shows that the cumulative distribution function of $X \sim \mathcal{U}(a, b)$ is

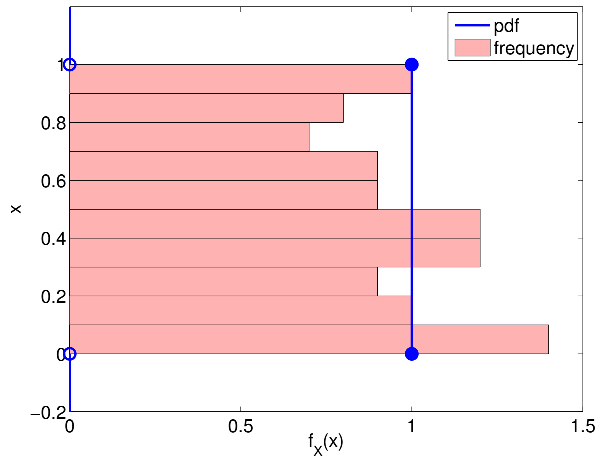
$$F_X(x) = F^{\text{uniform}}(x; a, b) \equiv \frac{x-a}{b-a}.$$

The pdf and cdf for a few uniform distributions are shown in Figure 9.10.

An example of the values taken by a uniform random variable $\mathcal{U}(0, 1)$ is shown in Figure 9.11(a). By construction, the range of values that the variable takes is limited to between $a = 0$ and $b = 1$. As expected, there is no obvious concentration of the values within the range $[a, b]$. Figure 9.11(b) shows a histrogram that summarizes the frequency of the event that X resides in bins $[x_i, x_i + \delta x]$, $i = 1, \ldots, n_{\text{bin}}$. The relative frequency of occurrence is normalized by δx to be consistent with the definition of the probability density function. In particular, the integral of the region filled by the histogram is unity. While there is some spread in the frequencies of occurrences due to



(a) realization



(b) probability density

Figure 9.11: Illustration of the values taken by an uniform random variable ($a = 0$, $b = 1$).

the relatively small sample size, the histogram resembles the probability density function. This is consistent with the frequentist interpretation of probability.

The mean of the uniform distribution is given by

$$E[X] = \int_a^b x f_X(x)\,dx = \int_a^b x \frac{1}{b-a}\,dx = \frac{1}{2}(a+b)\ .$$

This agrees with our intuition, because if X is to take on a value between a and b with equal probability, then the mean would be the midpoint of the interval. The variance of the uniform distribution is

$$\begin{aligned}\mathrm{Var}(X) = E[X^2] - (E[X])^2 &= \int_a^b x^2 f_X(x)\,dx - \left(\frac{1}{2}(a+b)\right)^2 \\ &= \int_a^b \frac{x^2}{b-a}\,dx - \left(\frac{1}{2}(a+b)\right)^2 = \frac{1}{12}(b-a)^2\ .\end{aligned}$$

·

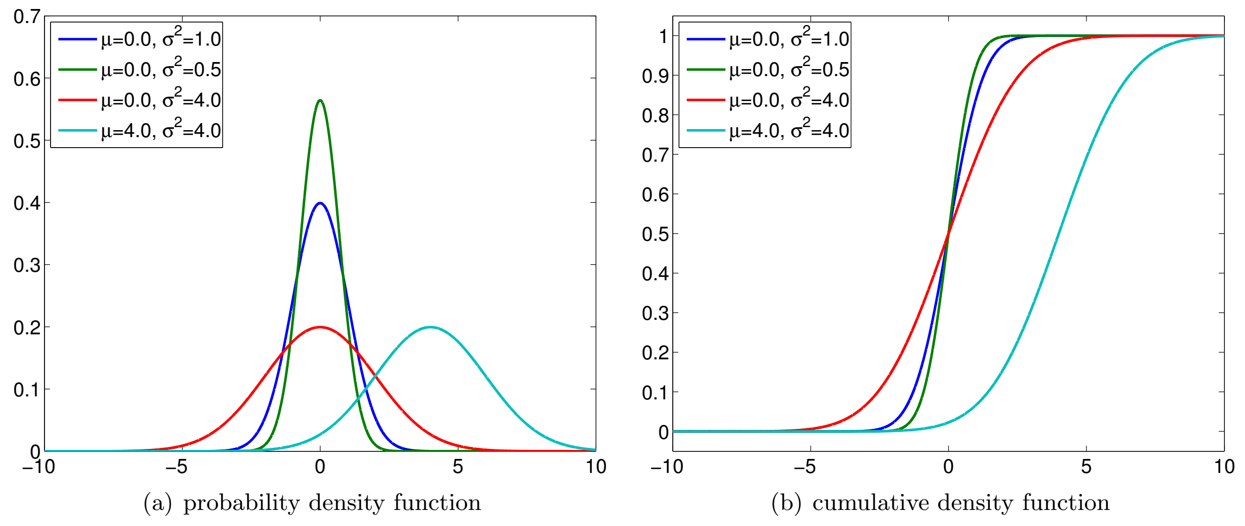
Example 9.4.2 Normal distribution

Let X be a normal random variable. Then the probability density function of X is of the form

$$f_X(x) = f^{\text{normal}}(x;\mu,\sigma^2) \equiv \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\ .$$

The pdf is parametrized by two variables, the mean μ and the variance σ^2. (More precisely we would thus write X_{μ,σ^2}.) Note that the density is non-zero over the entire real axis and thus in principle X can take on any value. The normal distribution is concisely denoted by $X \sim \mathcal{N}(\mu,\sigma^2)$. The cumulative distribution function of a normal distribution takes the form

$$F_X(x) = F^{\text{normal}}(x;\mu,\sigma^2) \equiv \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]\ ,$$



(a) probability density function (b) cumulative density function

Figure 9.12: Normal distributions

where erf is the *error function*, given by

$$\operatorname{erf}(x) = \frac{2}{\pi}\int_0^x e^{-z^2} dz \ .$$

We note that it is customary to denote the cdf for the standard normal distribution (i.e. $\mu = 0$, $\sigma^2 = 1$) by Φ, i.e.

$$\Phi(x) = F^{\text{normal}}(x; \mu = 0, \sigma^2 = 1).$$

We will see many use of this cdf in the subsequent sections. The pdf and cdf for a few normal distributions are shown in Figure 9.12.

An example of the values taken by a normal random variable is shown in Figure 9.13. As already noted, X can *in principle* take on any real value; however, in practice, as the Gaussian function decays quickly away from the mean, the probability of X taking on a value many standard deviations away from the mean is small. Figure 9.13 clearly illustrates that the values taken by X is clustered near the mean. In particular, we can deduce from the cdf that X takes on values within σ, 2σ, and 3σ of the mean with probability 68.2%, 95.4%, and 99.7%, respectively. In other words, the probability of X taking of the value outside of $\mu \pm 3\sigma$ is given by

$$1 - \int_{\mu-3\sigma}^{\mu+3\sigma} f^{\text{normal}}(x; \mu, \sigma^2)\, dx \equiv 1 - (F^{\text{normal}}(\mu + 3\sigma; \mu, \sigma^2) - F^{\text{normal}}(\mu - 3\sigma; \mu, \sigma^2)) \approx 0.003 \ .$$

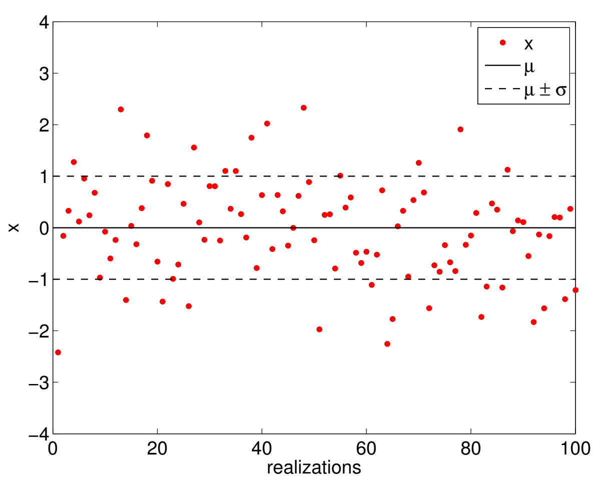
We can easily compute a few quantiles based on this information. For example,

$$\tilde{z}_{0.841} \approx \mu + \sigma, \quad \tilde{z}_{0.977} \approx \mu + 2\sigma, \quad \text{and} \quad \tilde{z}_{0.9987} \approx \mu + 3\sigma.$$

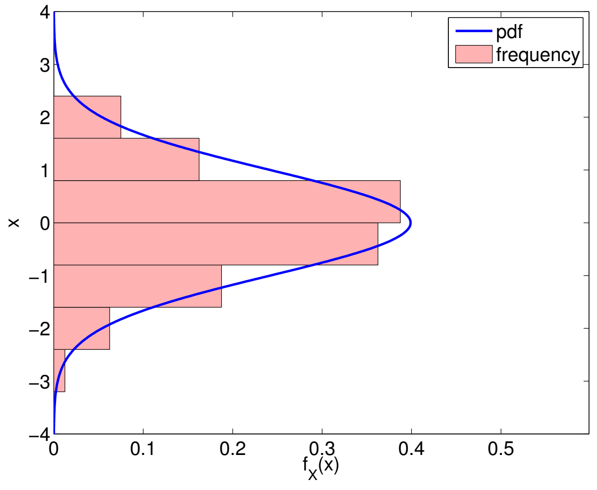
It is worth mentioning that $\tilde{z}_{0.975} \approx \mu + 1.96\sigma$, as we will frequently use this constant in the subsequent sections.

·

Although we only consider either discrete or continuous random variables in this notes, random variables can be mixed discrete and continuous in general. Mixed discrete-continuous random variables are characterized by the appearance of discontinuities in their cumulative distribution function.



(a) realization



(b) probability density

Figure 9.13: Illustration of the values taken by a normal random variable ($\mu = 0$, $\sigma = 1$).

9.4.2 Transformations of Continuous Random Variables

Just like discrete random variables, continuous random variables can be transformed by a function. The transformation of a random variable X by a function g produces another random variable, Y, and we denote this by

$$Y = g(X) .$$

We shall consider here only monotonic functions g.

Recall that we have described the random variable X by distribution

$$P(x \le X \le x + dx) = f_X(x)\, dx .$$

The transformed variable follows

$$P(y \le Y \le y + dy) = f_Y(y)\, dy .$$

Substitution of $y = g(x)$ and $dy = g'(x)dx$ and noting $g(x) + g'(x)dx = g(x + dx)$ results in

$$\begin{aligned} f_Y(y)\, dy &= P(g(x) \le g(X) \le g(x) + g'(x)\, dx) = P(g(x) \le g(X) \le g(x + dx)) \\ &= P(x \le X \le x + dx) = f_X(x)\, dx . \end{aligned}$$

In other words, $f_Y(y)dy = f_X(x)\, dx$. This is the continuous analog to $f_Y(y_j) = p_j = f_X(x_j)$ in the discrete case.

We can manipulate the expression to obtain an explicit expression for f_Y in terms of f_X and g. First we note (from monotonicity) that

$$y = g(x) \quad \Rightarrow \quad x = g^{-1}(y) \quad \text{and} \quad dx = \frac{dg^{-1}}{dy}\, dy .$$

Substitution of the expressions in $f_Y(y)dy = f_X(x)\, dx$ yields

$$f_Y(y)\, dy = f_X(x)\, dx = f_X(g^{-1}(y)) \cdot \frac{dg^{-1}}{dy}\, dy$$

or,

$$f_Y(y) = f_X(g^{-1}(y)) \cdot \frac{dg^{-1}}{dy} \ .$$

Conversely, we can also obtain an explicit expression for f_X in terms of f_Y and g. From $y = g(x)$ and $dy = g'(x)\,dx$, we obtain

$$f_X(x)\,dx = f_Y(y)\,dy = f_Y(g(x)) \cdot g'(x)\,dx \quad \Rightarrow \quad f_X(x) = f_Y(g(x)) \cdot g'(x) \ .$$

We shall consider several applications below.

Assuming X takes on a value between a and b, and Y takes on a value between c and d, the mean of Y is

$$E[Y] = \int_c^d y f_Y(y)\,dy = \int_a^b g(x) f_X(x)\,dx \ ,$$

where the second equality follows from $f_Y(y)\,dy = f_X(x)\,dx$ and $y = g(x)$.

Example 9.4.3 Standard uniform distribution to a general uniform distribution
As the first example, let us consider the standard uniform distribution $U \sim \mathcal{U}(0,1)$. We wish to generate a general uniform distribution $X \sim \mathcal{U}(a,b)$ defined on the interval $[a,b]$. Because a uniform distribution is uniquely determined by the two end points, we simply need to map the end point 0 to a and the point 1 to b. This is accomplished by the transformation

$$g(u) = a + (b-a)u \ .$$

Thus, $X \sim \mathcal{U}(a,b)$ is obtained by mapping $U \sim \mathcal{U}(0,1)$ as

$$X = a + (b-a)U \ .$$

Proof. Proof follows directly from the transformation of the probability density function. The probability density function of U is

$$f_U(u) = \begin{cases} 1, & u \in [0,1] \\ 0, & \text{otherwise} \end{cases} \ .$$

The inverse of the transformation $x = g(u) = a + (b-a)u$ is

$$g^{-1}(x) = \frac{x-a}{b-a} \ .$$

From the transformation of the probability density function, f_X is

$$f_X(x) = f_U(g^{-1}(x)) \cdot \frac{dg^{-1}}{dx} = f_U\left(\frac{x-a}{b-a}\right) \cdot \frac{1}{b-a} \ .$$

We note that f_U evaluates to 1 if

$$0 \le \frac{x-a}{b-a} \le 1 \quad \Rightarrow \quad a \le x \le b \ ,$$

and f_U evaluates to 0 otherwise. Thus, f_X simplifies to

$$f_X(x) = \begin{cases} \frac{1}{b-a}, & x \in [a,b] \\ 0, & \text{otherwise} \end{cases} \ ,$$

which is precisely the probability density function of $\mathcal{U}(a,b)$. □

Example 9.4.4 Standard uniform distribution to a discrete distribution
The uniform distribution can also be mapped to a discrete random variable. Consider a discrete random variable Y takes on three values ($J = 3$), with

$$y_1 = 0, \quad y_2 = 2, \quad \text{and} \quad y_3 = 3$$

with probability

$$f_Y(y) = \begin{cases} 1/2, & y_1 = 0 \\ 1/4, & y_2 = 2 \\ 1/4, & y_3 = 3 \end{cases} .$$

To generate Y, we can consider a discontinuous function g. To get the desired discrete probability distribution, we subdivide the interval $[0, 1]$ into three subintervals of appropriate lengths. In particular, to generate Y, we consider

$$g(x) = \begin{cases} 0, & x \in [0, 1/2) \\ 2, & x \in [1/2, 3/4) \\ 3, & x \in [3/4, 1] \end{cases} .$$

If we consider $Y = g(U)$, we have

$$Y = \begin{cases} 0, & U \in [0, 1/2) \\ 2, & U \in [1/2, 3/4) \\ 3, & U \in [3/4, 1] \end{cases} .$$

Because the probability that the standard uniform random variable takes on a value within a subinterval $[a', b']$ is equal to

$$P(a' \leq U \leq b') = \frac{b' - a'}{1 - 0} = b' - a' ,$$

the probability that Y takes on 0 is $1/2 - 0 = 1/2$, on 2 is $3/4 - 1/2 = 1/4$, and on 3 is $1 - 3/4 = 1/4$. This gives the desired probability distribution of Y.

Example 9.4.5 Standard normal distribution to a general normal distribution
Suppose we have the standard normal distribution $Z \sim \mathcal{N}(0, 1)$ and wish to map it to a general normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$ with the mean μ and the variance σ^2. The transformation is given by

$$X = \mu + \sigma Z .$$

Conversely, we can map any normal distribution to the standard normal distribution by

$$Z = \frac{X - \mu}{\sigma} .$$

Proof. The probability density function of the standard normal distribution $Z \sim \mathcal{N}(0,1)$ is

$$f_Z(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) \ .$$

Using the transformation of the probability density and the inverse mapping, $z(x) = (x-\mu)/\sigma$, we obtain

$$f_X(x) = f_Z(z(x))\frac{dz}{dx} = f_Z\left(\frac{x-\mu}{\sigma}\right) \cdot \frac{1}{\sigma} \ .$$

Substitution of the probability density function f_Z yields

$$f_X(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \cdot \frac{1}{\sigma} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \ ,$$

which is exactly the probability density function of $\mathcal{N}(\mu, \sigma^2)$. □

Example 9.4.6 General transformation by inverse cdf, F^{-1}
In general, if $U \sim \mathcal{U}(0,1)$ and F_Z is the cumulative distribution function from which we wish to draw a random variable Z, then

$$Z = F_Z^{-1}(U)$$

has the desired cumulative distribution function, F_Z.

Proof. The proof is straightforward from the definition of the cumulative distribution function, i.e.

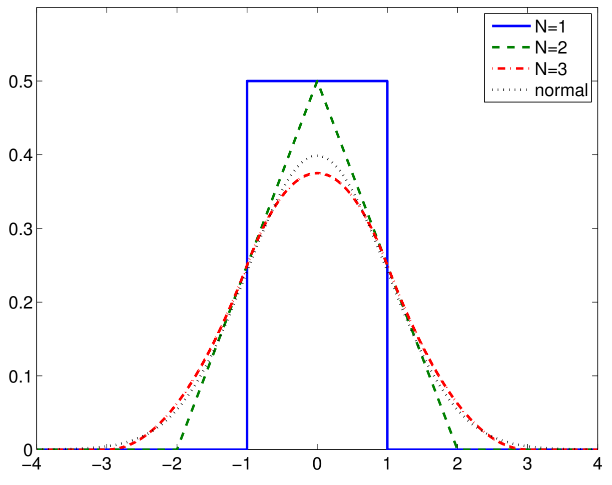
$$P(Z \leq z) = P(F_Z^{-1}(U) \leq z) = P(U \leq F_Z(z)) = F_Z(z).$$

Here we require that F_Z is monotonically increasing in order to be invertible. □

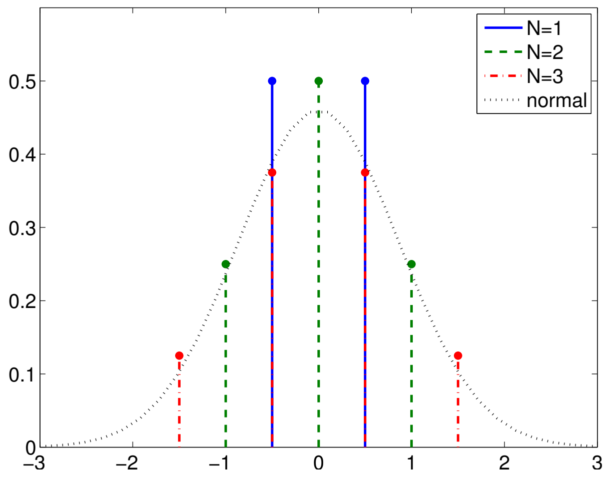
9.4.3 The Central Limit Theorem

The ubiquitousness of the normal distribution stems from the central limit theorem. (The normal density is also very convenient, with intuitive location (μ) and scale (σ^2) parameters.) The central limits theorem states that the sum of a sufficiently larger number of i.i.d. random variables tends to a normal distribution. In other words, if an experiment is repeated a larger number of times, the outcome on average approaches a normal distribution. Specifically, given i.i.d. random variables X_i, $i = 1, \ldots, N$, each with the mean $E[X_i] = \mu$ and variance $\mathrm{Var}[X_i] = \sigma^2$, their sum converges to

$$\sum_{i=1}^{N} X_i \to \mathcal{N}(\mu N, \sigma^2 N), \quad \text{as} \quad N \to \infty \ .$$



(a) Sum of uniform random variables



(b) Sum of (shifted) Bernoulli random variables

Figure 9.14: Illustration of the central limit theorem for continuous and discrete random variables.

(There are a number of mathematical hypotheses which must be satisfied.)

To illustrate the theorem, let us consider the sum of uniform random variables $X_i \sim \mathcal{U}(-1/2, 1/2)$. The mean and variance of the random variable are $E[X_i] = 0$ and $\text{Var}[X_i] = 1/3$, respectively. By central limit theorem, we expect their sum, Z_N, to approach

$$Z_N \equiv \sum_{i=1}^{N} X_i \to \mathcal{N}(\mu N, \sigma^2 N) = \mathcal{N}(0, N/3) \quad \text{as} \quad N \to \infty\ .$$

The pdf of the sum Z_i, $i = 1, 2, 3$, and the normal distribution $\mathcal{N}(0, N/3)|_{N=3} = \mathcal{N}(0, 1)$ are shown in Figure 9.14(a). Even though the original uniform distribution ($N = 1$) is far from normal and $N = 3$ is not a large number, the pdf for $N = 3$ can be closely approximated by the normal distribution, confirming the central limit theorem in this particular case.

The theorem also applies to discrete random variable. For example, let us consider the sum of (shifted) Bernoulli random variables,

$$X_i = \begin{cases} -1/2, & \text{with probability } 1/2 \\ 1/2, & \text{with probability } 1/2 \end{cases} \quad .$$

Note that the value that X takes is shifted by $-1/2$ compared to the standard Bernoulli random variable, such that the variable has zero mean. The variance of the distribution is $\text{Var}[X_i] = 1/4$. As this is a discrete distribution, their sum also takes on discrete values; however, Figure 9.14(b) shows that the probability mass function can be closely approximated by the pdf for the normal distribution.

9.4.4 Generation of Pseudo-Random Numbers

To generate a realization of a random variable X — also known as a *random variate* — computationally, we can use a pseudo-random number generator. Pseudo-random number generators are algorithms that generate a sequence of numbers that appear to be random. However, the actual sequence generated is completely determined by a seed — the variable that specifies the initial state of the generator. In other words, given a seed, the sequence of the numbers generated is completely deterministic and reproducible. Thus, to generate a different sequence each time, a pseudo-random number generator is seeded with a quantity that is not fixed; a common choice is to use the current machine time. However, the deterministic nature of the pseudo-random number can be useful, for example, for debugging a code.

A typical computer language comes with a library that produces the standard continuous uniform distribution and the standard normal distribution. To generate other distributions, we can apply the transformations we considered earlier. For example, suppose that we have a pseudo-random number generator that generates the realization of $U \sim \mathcal{U}(0,1)$,

$$u_1, u_2, \dots \; .$$

Then, we can generate a realization of a general uniform distribution $X \sim \mathcal{U}(a,b)$,

$$x_1, x_2, \dots \; ,$$

by using the transformation

$$x_i = a + (b-a)u_i, \quad i = 1, 2, \dots \; .$$

Similarly, we can generate given a realization of the standard normal distribution $Z \sim \mathcal{N}(0,1)$, $z_1, z_2, \dots$, we can generate a realization of a general normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, $x_1, x_2, \dots$, by

$$x_i = \mu + \sigma z_i, \quad i = 1, 2, \dots \; .$$

These two transformations are perhaps the most common.

Finally, if we wish to generate a discrete random number Y with the probability mass function

$$f_Y(y) = \begin{cases} 1/2, & y_1 = 0 \\ 1/4, & y_2 = 2 \\ 1/4, & y_3 = 3 \end{cases} \; ,$$

we can map a realization of the standard continuous uniform distribution $U \sim \mathcal{U}(0,1)$, $u_1, u_2, \dots$, according to

$$y_i = \begin{cases} 0, & u_i \in [0, 1/2) \\ 2, & u_i \in [1/2, 3/4) \\ 3, & u_i \in [3/4, 1] \end{cases} \qquad i = 1, 2, \dots \; .$$

(Many programming languages directly support the uniform pmf.)

More generally, using the procedure described in Example 9.4.6, we can sample a random variable Z with cumulative distribution function F_Z by mapping realizations of the standard uniform distribution, u_1, u_2, $\dots$ according to

$$z_i = F_Z^{-1}(u_i), \quad i = 1, 2, \dots \; .$$

We note that there are other sampling techniques which are even more general (if not always efficient), such as "acceptance-rejection" approaches.

9.5 Continuous Random Vectors

Following the template used to extend discrete random variables to discrete random vectors, we now introduce the concept of continuous random vectors. Let $X = (X_1, X_2)$ be a random variable with

$$\begin{aligned} a_1 &\le X_1 \le b_1 \\ a_2 &\le X_2 \le b_2 \ . \end{aligned}$$

The probability density function (pdf) is now a function over the rectangle

$$R \equiv [a_1, b_1] \times [a_2, b_2]$$

and is denoted by

$$f_{X_1,X_2}(x_1, x_2) \quad \text{(or, more concisely, } f_X(x_1, x_2)) \ .$$

The pdf must satisfy the following conditions:

$$\begin{aligned} f_X(x_1, x_2) \ge 0, \quad \forall\, (x_1, x_2) \in R \\ \int_{a_1}^{b_1} \int_{a_2}^{b_2} f_X(x_1, x_2) = 1 \ . \end{aligned}$$

The value of the pdf can be interpreted as a probability per unit area, in the sense that

$$P(x_1 \le X_1 \le x_1 + dx_1, x_2 \le X_2 \le x_2 + dx_2) = f_X(x_1, x_2)\, dx_1\, dx_2 \ ,$$

and

$$P(X \in D) = \iint_D f_X(x_1, x_2)\, dx_1\, dx_2 \ ,$$

where $\iint_D$ refers to the integral over $D \subset R$ (a subset of R).

Let us now revisit key concepts used to characterize discrete joint distributions in the continuous setting. First, the *marginal density function* of X_1 is given by

$$f_{X_1}(x_1) = \int_{a_2}^{b_2} f_{X_1,X_2}(x_1, x_2)\, dx_2 \ .$$

Recall that the marginal density of X_1 describes the probability distribution of X_1 disregarding the state of X_2. Similarly, the marginal density function of X_2 is

$$f_{X_2}(x_2) = \int_{a_1}^{b_1} f_{X_1,X_2}(x_1, x_2)\, dx_1 \ .$$

As in the discrete case, the marginal densities are also valid probability distributions.

The conditional probability density function of X_1 given X_2 is

$$f_{X_1|X_2}(x_1|x_2) = \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_2}(x_2)} \ .$$

Similar to the discrete case, the marginal and conditional probabilities are related by

$$f_{X_1,X_2}(x_1, x_2) = f_{X_1|X_2}(x_1|x_2) \cdot f_{X_2}(x_2) \ ,$$

or

$$f_{X_1}(x_1) = \int_{a_2}^{b_2} f_{X_1,X_2}(x_1,x_2)\,dx_2 = \int_{a_2}^{b_2} f_{X_1|X_2}(x_1|x_2) \cdot f_{X_2}(x_2)\,dx_2 \ .$$

In words, the marginal probability density function of X_1 is equal to the integration of the conditional probability density of f_{X_1,X_2} weighted by the probability density of X_2.

Two continuous random variables are said to be independent if their joint probability density function satisfies

$$f_{X_1,X_2}(x_1,x_2) = f_{X_1}(x_1) \cdot f_{X_2}(x_2) \ .$$

In terms of conditional probability, the independence means that

$$f_{X_1|X_2}(x_1,x_2) = \frac{f_{X_1,X_2}(x_1,x_2)}{f_{X_2}(x_2)} = \frac{f_{X_1}(x_1) \cdot f_{X_2}(x_2)}{f_{X_2}(x_2)} = f_{X_1}(x_1) \ .$$

In words, knowing the outcome of X_2 does not add any new knowledge about the probability distribution of X_1.

The covariance of X_1 and X_2 in the continuous case is defined as

$$\mathrm{Cov}(X_1,X_2) = E[(X_1-\mu_1)(X_2-\mu_2)] \ ,$$

and the correlation is given by

$$\rho_{X_1X_2} = \frac{\mathrm{Cov}(X_1,X_2)}{\sigma_{X_1}\sigma_{X_2}} \ .$$

Recall that the correlation takes on a value between -1 and 1 and indicates how strongly the outcome of two random events are related. In particular, if the random variables are independent, then their correlation evaluates to zero. This is easily seen from

$$\begin{aligned}\mathrm{Cov}(X_1,X_2) = E[(X_1-\mu_1)(X_2-\mu_2)] &= \int_{a_2}^{b_2}\int_{a_1}^{b_1} (x_1-\mu_1)(x_2-\mu_2) f_{X_1,X_2}(x_1,x_2)\,dx_1\,dx_2 \\ &= \int_{a_2}^{b_2}\int_{a_1}^{b_1} (x_1-\mu_1)(x_2-\mu_2) f_{X_1}(x_1) f_{X_2}(x_2)\,dx_1\,dx_2 \\ &= \left[\int_{a_2}^{b_2} (x_2-\mu_2) f_{X_2}(x_2)\,dx_2\right] \cdot \left[\int_{a_1}^{b_1} (x_1-\mu_1) f_{X_1}(x_1)\,dx_1\right] \\ &= 0 \cdot 0 = 0 \ .\end{aligned}$$

Note the last step follows from the definition of the mean.

Example 9.5.1 Bivariate uniform distribution

A bivariate uniform distribution is defined by two sets of parameters $[a_1,b_1]$ and $[a_2,b_2]$ that specify the range that X_1 and X_2 take on, respectively. The probability density function of (X_1,X_2) is

$$f_{X_1,X_2}(x_1,x_2) = \frac{1}{(b_1-a_1)(b_2-a_2)} \ .$$

Note here X_1 and X_2 are independent, so

$$f_{X_1,X_2}(x_1,x_2) = f_{X_1}(x_1) \cdot f_{X_2}(x_2) \ ,$$

where

$$f_{X_1}(x_1) = \frac{1}{b_1 - a_1} \quad \text{and} \quad f_{X_2}(x_2) = \frac{1}{b_2 - a_2} \ .$$

As for the univariate case, we have

$$P(X \in D) = \frac{A_D}{A_R} \ ,$$

where A_D is the area of some arbitrary region D and A_R is the area of the rectangle. In words, the probability that a uniform random vector — a random "dart" lands in D — is simply the ratio of A_D to the total area of the dartboard (A_R).[1] This relationship — together with our binomial distribution — will be the key ingredients for our Monte Carlo methods for area calculation.

Note also that if A_D is itself a rectangle aligned with the coordinate directions, $A_D \equiv c_1 \leq x_1 \leq d_1, c_2 \leq x_2 \leq d_2$, then $P(X \in D)$ simplifies to the product of the length of D in x_1, $(d_1 - c_1)$, divided by $b_1 - a_1$, and the length of D in x_2, $(d_2 - c_2)$, divided by $b_2 - a_2$. Independence is manifested as a normalized product of lengths, or equivalently as the AND or intersection (*not* OR or union) of the two "event" rectangles $c_1 \leq x_1 \leq d_1, a_2 \leq x_2 \leq b_2$ and $a_1 \leq x_1 \leq b_1, c_2 \leq x_2 \leq d_2$.

To generate a realization of $X = (X_1, X_2)$, we express the vector as a function of two independent (scalar) uniform distributions. Namely, let us consider $U_1 \sim \mathcal{U}(0,1)$ and $U_2 \sim \mathcal{U}(0,1)$. Then, we can express the random vector as

$$\begin{aligned} X_1 &= a_1 + (b_1 - a_1)U_1 \\ X_2 &= a_2 + (b_2 - a_2)U_2 \\ X &= (X_1, X_2) \ . \end{aligned}$$

We stress that U_1 and U_2 must be independent in order for X_1 and X_2 to be independent.

·

Advanced Material

Example 9.5.2 Bivariate normal distribution
Let (X_1, X_2) be a bivariate normal random vector. The probability density function of (X_1, X_2) is of the form

$$\begin{aligned} f_{X_1,X_2}(x_1, x_2) &= f^{\text{bi-normal}}(x_1, x_2; \mu_1, \mu_2, \sigma_1, \sigma_2, \rho) \\ &\equiv \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2(1-\rho^2)} \left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \right] \right\} , \end{aligned}$$

where (μ_1, μ_2) are the means, (σ_1^2, σ_2^2) are the variances, and ρ is the correlation. The pairs $\{\mu_1, \sigma_1^2\}$ and $\{\mu_2, \sigma_2^2\}$ describe the marginal distributions of X_1 and X_2, respectively. The correlation coefficient must satisfy

$$-1 < \rho < 1$$

and, if $\rho = 0$, then X_1 and X_2 are uncorrelated. For a joint normal distribution, uncorrelated implies independence (this is not true for a general distribution).

[1] A bullseye (highest score) in darts is not difficult because it lies at the center, but rather because it occupies the least area.

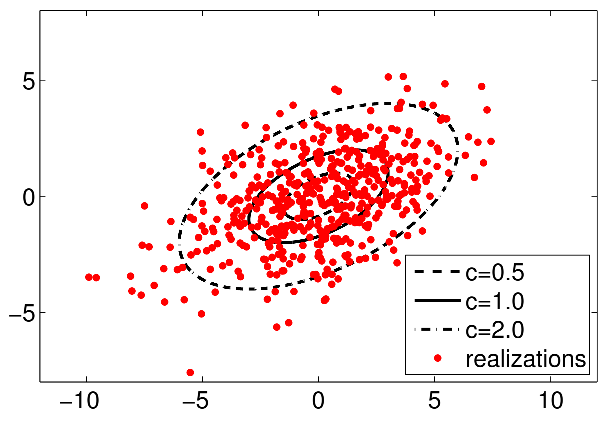


Figure 9.15: A bivariate normal distribution with $\mu_1 = \mu_2 = 0$, $\sigma_1 = 3$, $\sigma_2 = 2$, and $\rho = 1/2$.

The probability density function for the bivariate normal distribution with $\mu_1 = \mu_2 = 0$, $\sigma_1 = 3$, $\sigma_2 = 2$, and $\rho = 1/2$ is shown in Figure 9.15. The lines shown are the lines of equal density. In particular, the solid line corresponds to the 1σ line, and the dashed lines are for $\sigma/2$ and 2σ as indicated. 500 realizations of the distribution are also shown in red dots. For a bivariate distribution, the chances are 11.8%, 39.4%, and 86.5% that (X_1, X_2) takes on the value within $\sigma/2$, 1σ, and 2σ, respectively. The realizations shown confirm this trend, as only a small fraction of the red dots fall outside of the 2σ contour. This particular bivariate normal distribution has a weak positive correlation, i.e. given that X_2 is greater than its mean μ_{X_2}, there is a higher probability that X_1 is also greater than its mean, μ_{X_1}.

To understand the behavior of bivariate normal distributions in more detail, let us consider the marginal distributions of X_1 and X_2. The marginal distribution of X_1 of a bivariate normal distribution characterized by $\{\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho\}$ is a univariate normal distribution with the mean μ_1 and the variance σ_1^2, i.e.

$$f_{X_1}(x_1) \equiv \int_{x_2=-\infty}^{\infty} f_{X_1,X_2}(x_1,x_2)dx_2 = f^{\text{normal}}(x_1;\mu_1,\sigma_1) \ .$$

In words, if we look at the samples of the binormal random variable (X_1, X_2) and focus on the behavior of X_1 only (i.e. disregard X_2), then we will observe that X_1 is normally distributed. Similarly, the marginal density of X_2 is

$$f_{X_2}(x_2) \equiv \int_{x_1=-\infty}^{\infty} f_{X_1,X_2}(x_1,x_2)dx_1 = f^{\text{normal}}(x_2;\mu_2,\sigma_2) \ .$$

This rather surprising result is one of the properties of the binormal distribution, which in fact extends to higher-dimensional multivariate normal distributions.

Proof. For convenience, we will first rewrite the probability density function as

$$f_{X_1,X_2}(x_1,x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2}q(x_1,x_2)\right)$$

where the quadratic term is

$$q(x_1,x_2) = \frac{1}{1-\rho^2}\left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2}\right] \ .$$

We can manipulate the quadratic term to yield

$$\begin{aligned}
q(x_1,x_2) &= \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{1}{1-\rho^2}\left[\frac{\rho^2(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2}\right] \\
&= \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{1}{1-\rho^2}\left[\frac{\rho(x_1-\mu_1)}{\sigma_1} - \frac{x_2-\mu_2}{\sigma_2}\right]^2 \\
&= \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{1}{\sigma_2^2(1-\rho^2)}\left[x_2 - \left(\mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1)\right)\right]^2 .
\end{aligned}$$

Substitution of the expression into the probability density function yields

$$\begin{aligned}
f_{X_1,X_2}(x_1,x_2) &= \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2}q(x_1,x_2)\right) \\
&= \frac{1}{\sqrt{2\pi}\sigma_1}\exp\left(-\frac{1}{2}\frac{(x_1-\mu_1)^2}{\sigma_1^2}\right) \\
&\quad \times \frac{1}{\sqrt{2\pi}\sigma_2\sqrt{1-\rho^2}}\exp\left(-\frac{1}{2}\frac{(x_2-(\mu_2+\rho(\sigma_2/\sigma_1)(x_1-\mu_1)))^2}{\sigma_2^2(1-\rho^2)}\right) \\
&= f^{\text{normal}}(x_1;\mu_1,\sigma_1^2)\cdot f^{\text{normal}}\left(x_2;\mu_2+\rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1),\sigma_2^2(1-\rho^2)\right) .
\end{aligned}$$

Note that we have expressed the joint probability as the product of two univariate Gaussian functions. We caution that this does not imply independence, because the mean of the second distribution is dependent on the value of x_1. Applying the definition of marginal density of X_1 and integrating out the x_2 term, we obtain

$$\begin{aligned}
f_{X_1}(x_1) &= \int_{x_2=-\infty}^{\infty} f_{X_1,X_2}(x_1,x_2)dx_2 \\
&= \int_{x_2=-\infty}^{\infty} f^{\text{normal}}(x_1;\mu_1,\sigma_1^2)\cdot f^{\text{normal}}\left(x_2;\mu_2+\rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1),\sigma_2^2(1-\rho^2)\right)dx_2 \\
&= f^{\text{normal}}(x_1;\mu_1,\sigma_1^2)\cdot\int_{x_2=-\infty}^{\infty} f^{\text{normal}}\left(x_2;\mu_2+\rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1),\sigma_2^2(1-\rho^2)\right)dx_2 \\
&= f^{\text{normal}}(x_1;\mu_1,\sigma_1^2) .
\end{aligned}$$

The integral of the second function evaluates to unity because it is a probability density function. Thus, the marginal density of X_1 is simply the univariate normal distribution with parameters μ_1 and σ_1. The proof for the marginal density of X_2 is identical due to the symmetry of the joint probability density function. □

Figure 9.16 shows the marginal densities f_{X_1} and f_{X_2} along with the $\sigma = 1$- and $\sigma = 2$-contours of the joint probability density. The dots superimposed on the joint density are 500 realizations of (X_1, X_2). The histogram on the top summarizes the relative frequency of X_1 taking on a value within the bins for the 500 realizations. Similarly, the histogram on the right summarizes relative frequency of the values that X_2 takes. The histograms closely matches the theoretical marginal distributions for $\mathcal{N}(\mu_1,\sigma_1^2)$ and $\mathcal{N}(\mu_2,\sigma_2^2)$. In particular, we note that the marginal densities are independent of the correlation coefficient ρ.

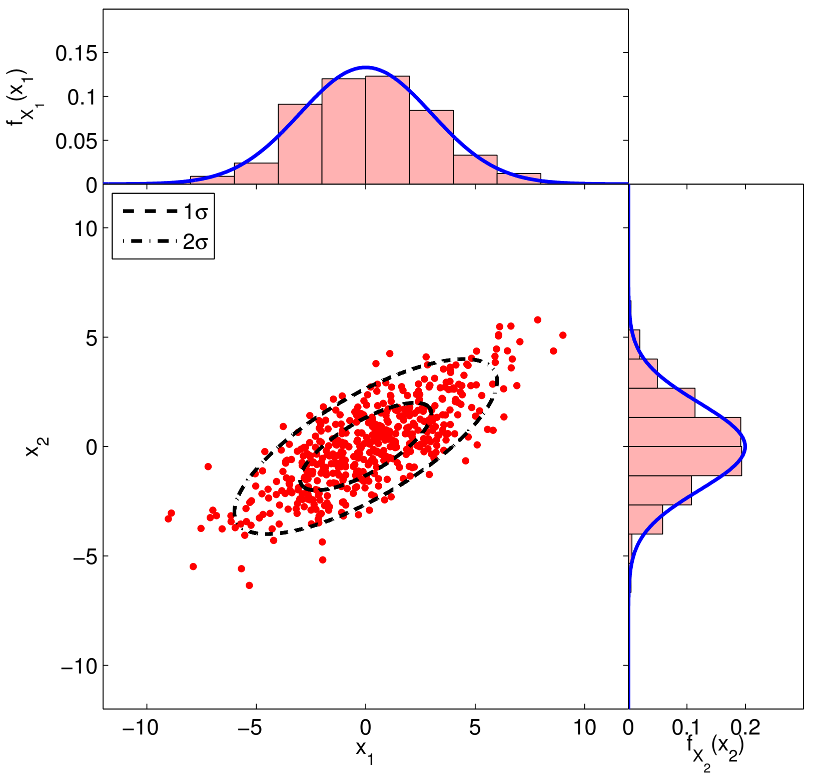


Figure 9.16: Illustration of marginal densities for a bivariate normal distribution ($\mu_1 = \mu_2 = 0$, $\sigma_1 = 3$, $\sigma_2 = 2$, $\rho = 3/4$).

Having studied the marginal densities of the bivariate normal distribution, let us now consider conditional probabilities. Combining the definition of conditional density and the expression for the joint and marginal densities, we obtain

$$\begin{aligned} f_{X_1|X_2}(x_1|x_2) = \frac{f_{X_1,X_2}(x_1,x_2)}{f_{X_2}(x_2)} &= f^{\text{normal}}\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2), (1-\rho^2)\sigma_1^2\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma_1\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2}\frac{(x_1-(\mu_1+\rho(\sigma_1/\sigma_2)x_2))^2}{\sigma_1^2(1-\rho^2)}\right) \ . \end{aligned}$$

Similarly, the conditional density of X_2 given X_1 is

$$f_{X_2|X_1}(x_2,x_1) = \frac{f_{X_1,X_2}(x_1,x_2)}{f_{X_1}(x_1)} = f^{\text{normal}}\left(x_2; \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1), (1-\rho^2)\sigma_2^2\right) \ .$$

Note that unlike the marginal probabilities, the conditional probabilities are function of the correlation coefficient ρ. In particular, the standard deviation of the conditional distribution (i.e. its spread about its mean) decreases with $|\rho|$ and vanishes as $\rho \to \pm 1$. In words, if the correlation is high, then we can deduce with a high probability the state of X_1 given the value that X_2 takes. We also note that the positive correlation ($\rho > 0$) results in the mean of the conditional probability $X_1|X_2$ shifted in the direction of X_2. That is, if X_2 takes on a value higher than its mean, then it is more likely than not that X_1 takes on a value higher than its mean.

Proof. Starting with the definition of conditional probability and substituting the joint and marginal probability density functions,

$$
\begin{aligned}
f_{X_1|X_2}(x_1|x_2) &= \frac{f_{X_1,X_2}(x_1,x_2)}{f_{X_2}(x_2)} \\
&= \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2(1-\rho^2)} \left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \right] \right\} \\
&\qquad \times \frac{\sqrt{2\pi}\sigma_2}{1} \exp\left(\frac{1}{2} \frac{(x_2-\mu_2)^2}{\sigma_2^2} \right) \\
&= \frac{1}{\sqrt{2\pi}\sigma_1\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2} s(x_1,x_2) \right\}
\end{aligned}
$$

where

$$
s(x_1,x_2) = \frac{1}{1-\rho^2} \left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} - (1-\rho^2)\frac{(x_2-\mu_2)^2}{\sigma_2^2} \right] .
$$

Rearrangement of the quadratic term $s(x_1, x_2)$ yields

$$
\begin{aligned}
s(x_1,x_2) &= \frac{1}{1-\rho^2} \left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{\rho^2(x_2-\mu_2)^2}{\sigma_2^2} \right] \\
&= \frac{1}{1-\rho^2} \left[\frac{x_1-\mu_1}{\sigma_1} - \frac{\rho(x_2-\mu_2)}{\sigma_2} \right]^2 \\
&= \frac{1}{\sigma_1^2(1-\rho^2)} \left[x_1 - \left(\mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2) \right) \right]^2 .
\end{aligned}
$$

Substitution of the quadratic term into the conditional probability density function yields

$$
\begin{aligned}
f_{X_1|X_2}(x_1|x_2) &= \frac{1}{\sqrt{2\pi}\sigma_1\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2}\frac{1}{\sigma_1^2(1-\rho^2)} \left[x_1 - \left(\mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2) \right) \right]^2 \right\} \\
&= f^{\text{normal}}\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2), (1-\rho^2)\sigma_1^2 \right) ,
\end{aligned}
$$

where the last equality follows from recognizing the univariate normal probability distribution function. □

Figure 9.17 shows the conditional densities $f_{X_1|X_2}(x_1|x_2 = -2)$ and $f_{X_2|X_1}(x_2|x_1 = 3)$ for a bivariate normal distribution ($\mu_1 = \mu_2 = 0$, $\sigma_1 = 3$, $\sigma_2 = 2$, $\rho = 3/4$). The histograms are constructed by counting the relative frequency of occurrence for those realizations that falls near the conditional value of $x_2 = -2$ and $x_1 = 3$, respectively. Clearly, the mean of the conditional probability densities are shifted relative to the respective marginal densities. As $\rho = 3/4 > 0$ and $x_2 - \mu_2 = -2 < 0$, the mean for $X_1|X_2$ is shifted in the negative direction. Conversely, $\rho > 0$ and $x_1 - \mu_1 = 3 > 0$ shifts the mean for $X_2|X_1$ in the positive direction. We also note that the conditional probability densities are tighter than the respective marginal densities; due to the relative strong correlation of $\rho = 3/4$, we have a better knowledge of the one state when we know the value of the other state.

Finally, to solidify the idea of correlation, let us consider the 1σ-contour for bivariate normal distributions with several different values of ρ, shown in Figure 9.18. A stronger (positive) correlation implies that there is a high chance that a positive value of x_2 implies a positive value of

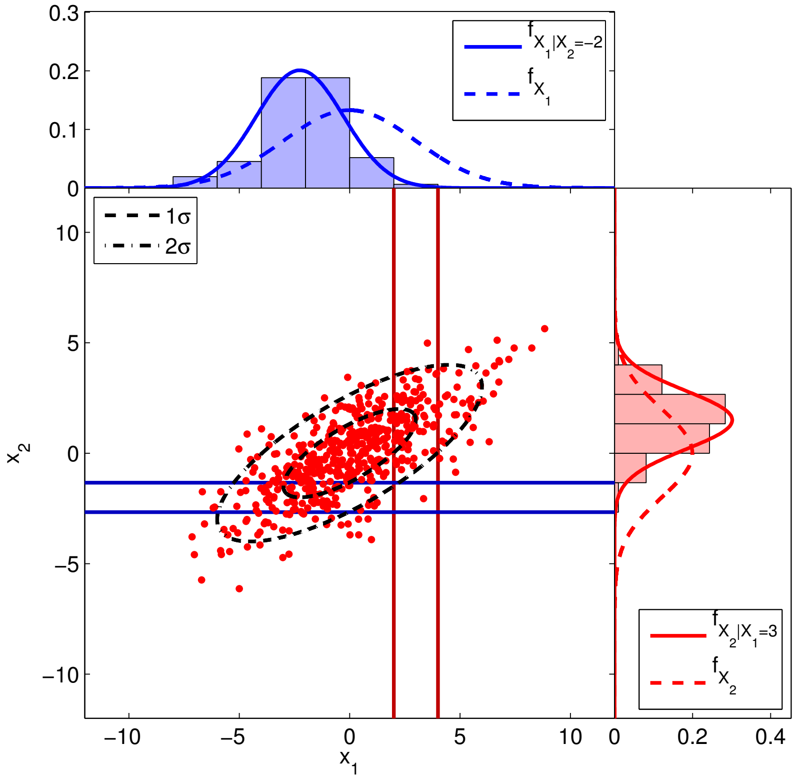


Figure 9.17: Illustration of conditional densities $f_{X_1|X_2}(x_1|x_2 = -2)$ and $f_{X_2|X_1}(x_2|x_1 = 3)$ for a bivariate normal distribution ($\mu_1 = \mu_2 = 0$, $\sigma_1 = 3$, $\sigma_2 = 2$, $\rho = 3/4$).

x_1. Conversely, a strong negative correlation implies that there is a high chance a positive value of x_2 implies a negative value of x_1. Zero correlation — which implies independence for normal distributions — means that we gain no additional information about the value that X_1 takes on by knowing the value of X_2; thus, the contour of equal probability density is not tilted.

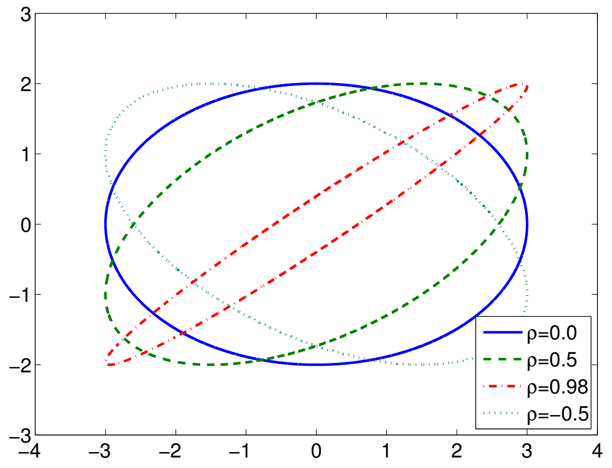


Figure 9.18: Bivariate normal distributions with $\mu_1 = \mu_2 = 0$, $\sigma_1 = 3$, $\sigma_2 = 2$, and several values of ρ.

End Advanced Material

Chapter 10

Statistical Estimation: Bernoulli (Coins)

10.1 Introduction

Recall that statistical estimation is a process through which we deduce parameters of the density that characterize the behavior of a random experiment based on a *sample* — a typically large but in any event finite number of *observable* outcomes of the random experiment. Specifically, a sample is a set of n independent and identically distributed (i.i.d.) random variables; we recall that a set of random variables

$$X_1, X_2, \ldots, X_n$$

is i.i.d. if

$$f_{X_1,\ldots,X_n}(x_1,\ldots,x_n) = f_X(x_1)\cdots f_X(x_n),$$

where f_X is the common probability density for $X_1,\ldots,X_n$. We also define a *statistic* as a function of a sample which returns a random number that represents some attribute of the sample; a statistic can also refer to the actual variable so calculated. Often a statistic serves to estimate a parameter. In this chapter, we focus on statistical estimation of parameters associated with arguably the simplest distribution: Bernoulli random variables.

10.2 The Sample Mean: An Estimator / Estimate

Let us illustrate the idea of sample mean in terms of a coin flip experiment, in which a coin is flipped n times. Unlike the previous cases, the coin may be unfair, i.e. the probability of heads, θ, may not be equal to $1/2$. We assume that we do not know the value of θ, and we wish to estimate θ from data collected through n coin flips. In other words, this is a parameter estimation problem, where the unknown parameter is θ. Although this chapter serves as a prerequisite for subsequence chapters on Monte Carlo methods — in which we apply probabilistic concepts to calculates areas and more generally integrals — in fact the current chapter focuses on how we might deduce physical parameters from noisy measurements. In short, statistics can be applied either to physical quantities treated as random variables or deterministic quantities which are re-interpreted as random (or pseudo-random).

As in the previous chapter, we associate the outcome of n flips with a random vector consisting of n i.i.d. Bernoulli random variables,

$$(B_1, B_2, \ldots, B_n) ,$$

where each B_i takes on the value of 1 with probably of θ and 0 with probability of $1 - \theta$. The random variables are i.i.d. because the outcome of one flip is independent of another flip and we are using the same coin.

We define the sample mean of n coin flips as

$$\overline{B}_n \equiv \frac{1}{n} \sum_{i=1}^{n} B_i ,$$

which is equal to the fraction of flips which are heads. Because $\overline{B}_n$ is a transformation (i.e. sum) of random variables, it is also a random variable. Intuitively, given a large number of flips, we "expect" the fraction of flips which are heads — the frequency of heads — to approach the probability of a head, θ, for n sufficiently large. For this reason, the sample mean is our *estimator* in the context of parameter estimation. Because the estimator estimates the parameter θ, we will denote it by $\widehat{\Theta}_n$, and it is given by

$$\widehat{\Theta}_n = \overline{B}_n = \frac{1}{n} \sum_{i=1}^{n} B_i .$$

Note that the sample mean is an example of a statistic — a function of a sample returning a random variable — which, in this case, is intended to estimate the parameter θ.

We wish to estimate the parameter from a particular realization of coin flips (i.e. a realization of our random sample). For any particular realization, we calculate our estimate as

$$\hat{\theta}_n = \hat{b}_n \equiv \frac{1}{n} \sum_{i=1}^{n} b_i ,$$

where b_i is the particular outcome of the i-th flip. It is important to note that the b_i, $i = 1, \ldots, n$, are numbers, each taking the value of either 0 or 1. Thus, $\hat{\theta}_n$ is a number and not a (random) distribution. Let us summarize the distinctions:

	r.v.?	Description
θ	no	Parameter to be estimated that governs the behavior of underlying distribution
$\widehat{\Theta}_n$	yes	Estimator for the parameter θ
$\hat{\theta}_n$	no	Estimate for the parameter θ obtained from a particular realization of our sample

In general, how the random variable $\widehat{\Theta}_n$ is distributed — in particular about θ — determines if $\widehat{\Theta}_n$ is a good estimator for the parameter θ. An example of convergence of $\hat{\theta}_n$ to θ with n is shown in Figure 10.1. As n increases, $\hat{\theta}$ converges to θ for essentially all realization of B_i's. This follows from the fact that $\widehat{\Theta}_n$ is an unbiased estimator of θ — an estimator whose expected value is equal to the true parameter. We shall prove this shortly.

To gain a better insight into the behavior of $\widehat{\Theta}_n$, we can construct the empirical distribution of $\widehat{\Theta}_n$ by performing a large number of experiments for a given n. Let us denote the number of

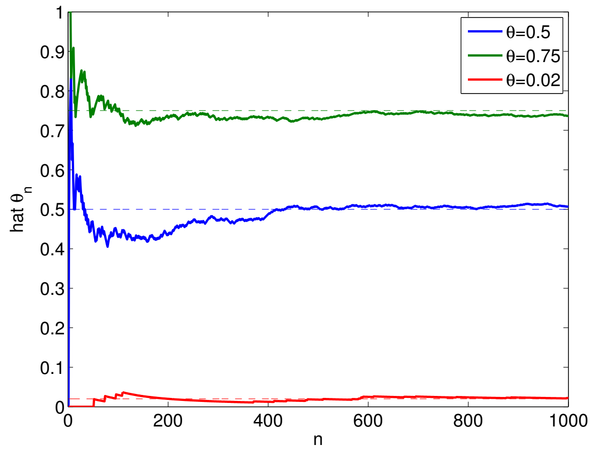


Figure 10.1: Convergence of estimate with n from a particular realization of coin flips.

experiments by n_{exp}. In the first experiment, we work with a realization $(b_1, b_2, \ldots, b_n)^{\text{exp 1}}$ and obtain the estimate by computing the mean, i.e.

$$\text{exp 1} : (b_1, b_2, \ldots, b_n)^{\text{exp 1}} \quad \Rightarrow \quad \bar{b}_n^{\text{exp 1}} = \frac{1}{n}\sum_{i=1}^{n} (b_i)^{\text{exp 1}} \ .$$

Similarly, for the second experiment, we work with a new realization to obtain

$$\text{exp 2} : (b_1, b_2, \ldots, b_n)^{\text{exp 2}} \quad \Rightarrow \quad \bar{b}_n^{\text{exp 2}} = \frac{1}{n}\sum_{i=1}^{n} (b_i)^{\text{exp 2}} \ .$$

Repeating the procedure n_{exp} times, we finally obtain

$$\text{exp } n_{\text{exp}} : (b_1, b_2, \ldots, b_n)^{\text{exp } n_{\text{exp}}} \quad \Rightarrow \quad \bar{b}_n^{\text{exp } n_{\text{exp}}} = \frac{1}{n}\sum_{i=1}^{n} (b_i)^{\text{exp } n_{\text{exp}}} \ .$$

We note that $\bar{b}_n$ can take any value k/n, $k = 0, \ldots, n$. We can compute the frequency of $\bar{b}_n$ taking on a certain value, i.e. the number of experiments that produces $\bar{b}_n = k/n$.

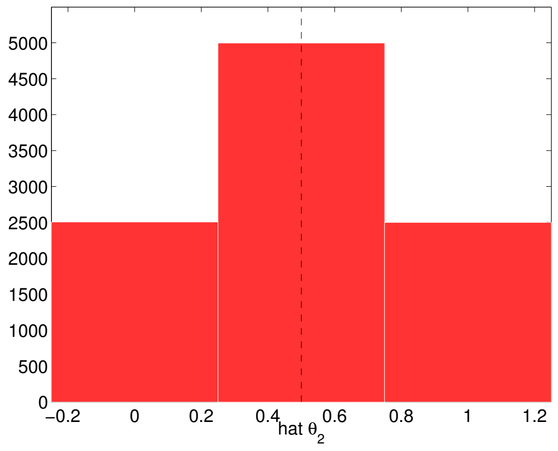
The numerical result of performing 10,000 experiments for $n = 2$, 10, 100, and 1000 flips are shown in Figure 10.2. The empirical distribution of $\widehat{\Theta}_n$ shows that $\widehat{\Theta}_n$ more frequently takes on the values close to the underlying parameter θ as the number of flips, n, increases. Thus, the numerical experiment confirms that $\widehat{\Theta}_n$ is indeed a good estimator of θ if n is sufficiently large.

Having seen that our estimate converges to the true parameter θ in practice, we will now analyze the convergence behavior to the true parameter by relating the sample mean to a binomial distribution. Recall, that the binomial distribution represents the number of heads obtained in flipping a coin n times, i.e. if $Z_n \sim \mathcal{B}(n, \theta)$, then

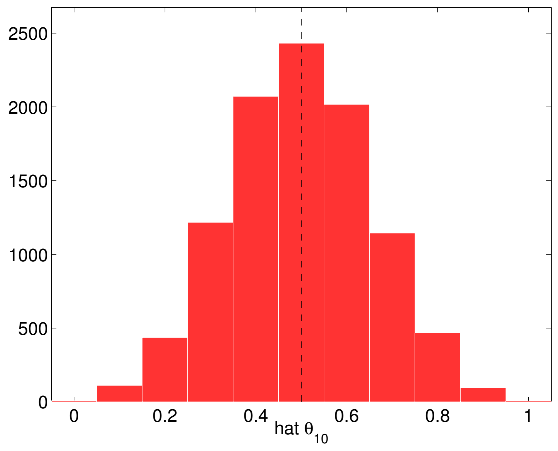
$$Z_n = \sum_{i=1}^{n} B_i \ ,$$

where B_i, $i = 1, \ldots, n$, are the i.i.d. Bernoulli random variable representing the outcome of coin flips (each having the probability of head of θ). The binomial distribution and the sample mean are related by

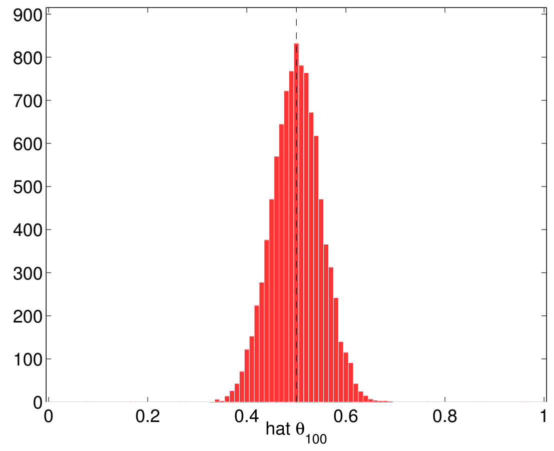
$$\widehat{\Theta}_n = \frac{1}{n} Z_n \ .$$



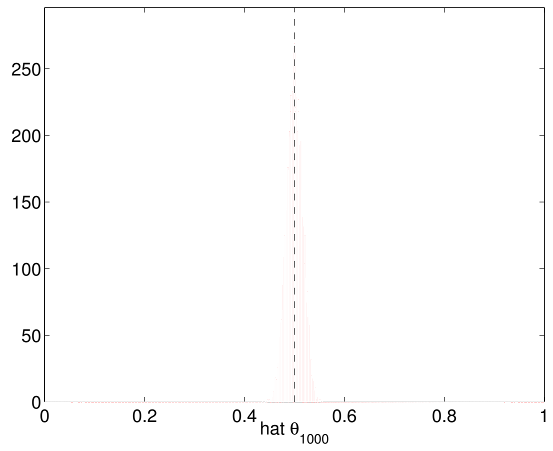
(a) $n = 2$



(b) $n = 10$



(c) $n = 100$



(d) $n = 1000$

Figure 10.2: Empirical distribution of $\widehat{\Theta}_n$ for $n = 2$, 10, 100, and 1000 and $\theta = 1/2$ obtained from 10,000 experiments.

The mean (a deterministic parameter) of the sample mean (a random variable) is

$$E[\widehat{\Theta}_n] = E\left[\frac{1}{n}Z_n\right] = \frac{1}{n}E[Z_n] = \frac{1}{n}(n\theta) = \theta \ .$$

In other words, $\widehat{\Theta}_n$ is an unbiased estimator of θ. The variance of the sample mean is

$$\begin{aligned}\operatorname{Var}[\widehat{\Theta}_n] &= E[(\widehat{\Theta}_n - E[\widehat{\Theta}_n])^2] = E\left[\left(\frac{1}{n}Z_n - \frac{1}{n}E[Z_n]\right)^2\right] = \frac{1}{n^2}E\left[(Z_n - E[Z_n])^2\right] \\ &= \frac{1}{n^2}\operatorname{Var}[Z_n] = \frac{1}{n^2}n\theta(1-\theta) = \frac{\theta(1-\theta)}{n} \ .\end{aligned}$$

The standard deviation of $\widehat{\Theta}_n$ is

$$\sigma_{\widehat{\Theta}_n} = \sqrt{\operatorname{Var}[\widehat{\Theta}_n]} = \sqrt{\frac{\theta(1-\theta)}{n}} \ .$$

Thus, the standard deviation of $\widehat{\Theta}_n$ decreases with n, and in particular tends to zero as $1/\sqrt{n}$. This implies that $\widehat{\Theta}_n \to \theta$ as $n \to \infty$ because it is very unlikely that $\widehat{\Theta}_n$ will take on a value many standard deviations away from the mean. In other words, the estimator converges to the true parameter with the number of flips.

10.3 Confidence Intervals

10.3.1 Definition

Let us now introduce the concept of confidence interval. The confidence interval is a probabilistic *a posteriori* error bound. *A posteriori* error bounds, as oppose to *a priori* error bounds, incorporate the information gathered in the experiment in order to assess the error in the prediction.

To understand the behavior of the estimator $\widehat{\Theta}_n$, which is a random variable defined by

$$B_1, \dots, B_n \quad \Rightarrow \quad \widehat{\Theta}_n = \overline{B}_n = \frac{1}{n}\sum_{i=1}^{n} B_i \ ,$$

we typically perform (in practice) a single experiment to generate a realization $(b_1, \dots, b_n)$. Then, we estimate the parameter by a number $\hat{\theta}_n$ given by

$$b_1, \dots, b_n \quad \Rightarrow \quad \hat{\theta}_n = \bar{b}_n = \frac{1}{n}\sum_{i=1}^{n} b_i \ .$$

A natural question: How good is the estimate $\hat{\theta}_n$? How can we quantify the small deviations of $\widehat{\Theta}_n$ from θ as n increases?

To answer these questions, we may construct a confidence interval, [CI], defined by

$$[\mathrm{CI}]_n \equiv \left[\widehat{\Theta}_n - z_\gamma\sqrt{\frac{\widehat{\Theta}_n(1-\widehat{\Theta}_n)}{n}}, \ \widehat{\Theta}_n + z_\gamma\sqrt{\frac{\widehat{\Theta}_n(1-\widehat{\Theta}_n)}{n}}\right]$$

such that

$$P(\theta \in [\mathrm{CI}]_n) = \gamma(z_\gamma) \ .$$

We recall that θ is the true parameter; thus, γ is the confidence level that the true parameter falls within the confidence interval. Note that $[\mathrm{CI}]_n$ is a random variable because $\widehat{\Theta}_n$ is a random variable.

For a large enough n, a (oft-used) confidence level of $\gamma = 0.95$ results in $z_\gamma \approx 1.96$. In other words, if we use $z_\gamma = 1.96$ to construct our confidence interval, there is a 95% probability that the true parameter lies within the confidence interval. In general, as γ increases, z_γ increases: if we want to ensure that the parameter lies within a confidence interval at a higher level of confidence, then the width of the confidence interval must be increased for a given n. The appearance of $1/\sqrt{n}$ in the confidence interval is due to the appearance of the $1/\sqrt{n}$ in the standard deviation of the estimator, $\sigma_{\widehat{\Theta}_n}$: as n increases, there is less variation in the estimator.

Strictly speaking, the above result is only valid as $n \to \infty$ (and $\theta \notin \{0, 1\}$), which ensures that $\widehat{\Theta}_n$ approaches the normal distribution by the central limit theorem. Then, under the normality assumption, we can calculate the value of the confidence-level-dependent multiplication factor z_γ according to

$$z_\gamma = \tilde{z}_{(1+\gamma)/2},$$

where $\tilde{z}_\alpha$ is the α quantile of the standard normal distribution, i.e. $\Phi(\tilde{z}_\alpha) = \alpha$ where Φ is the cumulative distribution function of the standard normal distribution. For instance, as stated above, $\gamma = 0.95$ results in $z_{0.95} = \tilde{z}_{0.975} \approx 1.96$. A practical rule for determining the validity of the normality assumption is to ensure that

$$n\theta > 5 \quad \text{and} \quad n(1-\theta) > 5.$$

In practice, the parameter θ appearing in the rule is replaced by its estimate, $\hat{\theta}_n$; i.e. we check

$$n\hat{\theta}_n > 5 \quad \text{and} \quad n(1-\hat{\theta}_n) > 5. \tag{10.1}$$

In particular, note that for $\hat{\theta}_n = 0$ or 1, we cannot construct our confidence interval. This is not surprising, as, for $\hat{\theta}_n = 0$ or 1, our confidence interval would be of zero length, whereas clearly there is some uncertainty in our prediction. We note that there are binomial confidence intervals that do not require the normality assumption, but they are slightly more complicated and less intuitive. Note also that in addition to the "centered" confidence intervals described here we may also develop one-sided confidence intervals.

10.3.2 Frequentist Interpretation

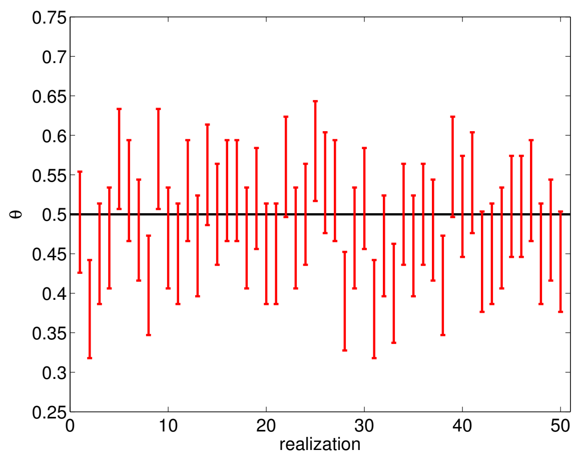
To get a better insight into the behavior of the confidence interval, let us provide an frequentist interpretation of the interval. Let us perform n_{exp} experiments and construct n_{exp} realizations of confidence intervals, i.e.

$$[\mathrm{ci}]_n^j = \left[\hat{\theta}_n^j - z_\gamma\sqrt{\frac{\hat{\theta}_n^j(1-\hat{\theta}_n^j)}{n}},\ \hat{\theta}_n^j + z_\gamma\sqrt{\frac{\hat{\theta}_n^j(1-\hat{\theta}_n^j)}{n}}\right], \quad j = 1, \ldots, n_{\text{exp}}\ ,$$

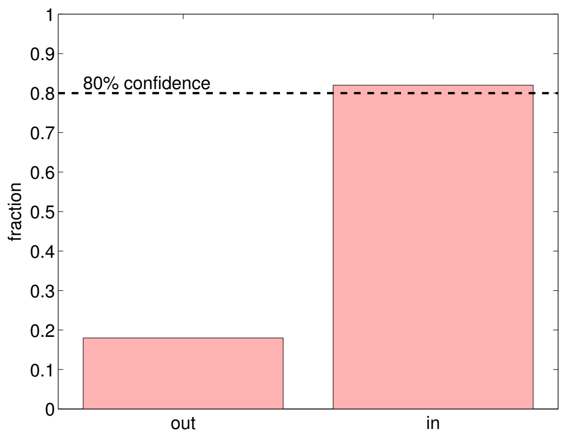
where the realization of sample means is given by

$$(b_1, \ldots, b_n)^j \quad \Rightarrow \quad \hat{\theta}^j = \frac{1}{n}\sum_{i=1}^{n} b_i^j\ .$$

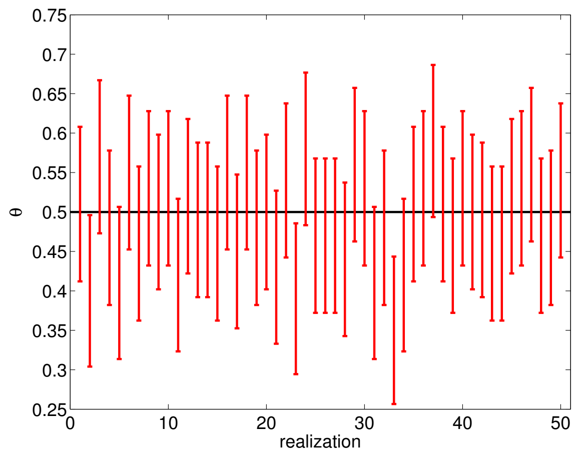
Then, as $n_{\text{exp}} \to \infty$, the fraction of experiments for which the true parameter θ lies inside $[\mathrm{ci}]_n^j$ tends to γ.



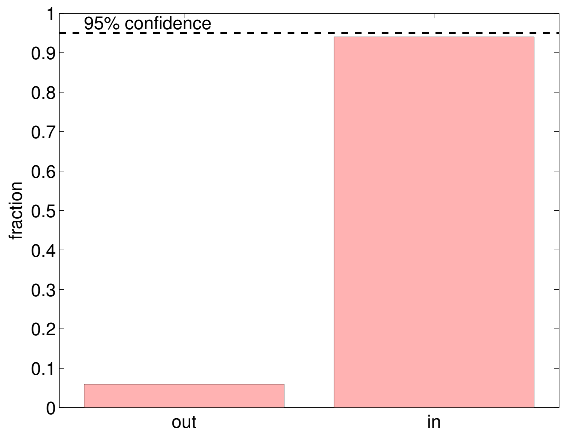
(a) 80% confidence



(b) 80% confidence in/out



(c) 95% confidence



(d) 95% confidence in/out

Figure 10.3: An example of confidence intervals for estimating the mean of a Bernoulli random variable ($\theta = 0.5$) using 100 samples.

An example of confidence intervals for estimating the mean of Bernoulli random variable ($\theta = 0.5$) using samples of size $n = 100$ is shown in Figure 10.3. In particular, we consider sets of 50 different realizations of our sample (i.e. 50 experiments, each with a sample of size 100) and construct 80% ($z_\gamma = 1.28$) and 95% ($z_\gamma = 1.96$) confidence intervals for each of the realizations. The histograms shown in Figure 10.3(b) and 10.3(d) summarize the relative frequency of the true parameter falling in and out of the confidence intervals. We observe that 80% and 95% confidence intervals include the true parameter θ in 82% (9/51) and 94% (47/50) of the realizations, respectively; the numbers are in good agreement with the frequentist interpretation of the confidence intervals. Note that, for the same number of samples n, the 95% confidence interval has a larger width, as it must ensure that the true parameter lies within the interval with a higher probability.

10.3.3 Convergence

Let us now characterize the convergence of our prediction to the true parameter. First, we define the half length of the confidence interval as

$$\text{Half Length}_{\theta;n} \equiv z_\gamma \sqrt{\frac{\hat{\theta}_n(1-\hat{\theta}_n)}{n}} \ .$$

Then, we can define a relative error a relative error estimate in our prediction as

$$\text{RelErr}_{\theta;n} = \frac{\text{Half Length}_{\theta;n}}{\hat{\theta}} = z_\gamma \sqrt{\frac{1-\hat{\theta}_n}{\hat{\theta}_n n}} \ .$$

The appearance of $1/\sqrt{n}$ convergence of the relative error is due to the $1/\sqrt{n}$ dependence in the standard deviation $\sigma_{\widehat{\Theta}_n}$. Thus, the relative error converges in the sense that

$$\text{RelErr}_{\theta;n} \to 0 \quad \text{as} \quad n \to \infty \ .$$

However, the convergence rate is slow

$$\text{RelErr}_{\theta;n} \sim n^{-1/2} \ ,$$

i.e. the convergence rate if of order $1/2$ as $n \to \infty$. Moreover, note that rare events (i.e. low θ) are difficult to estimate accurately, as

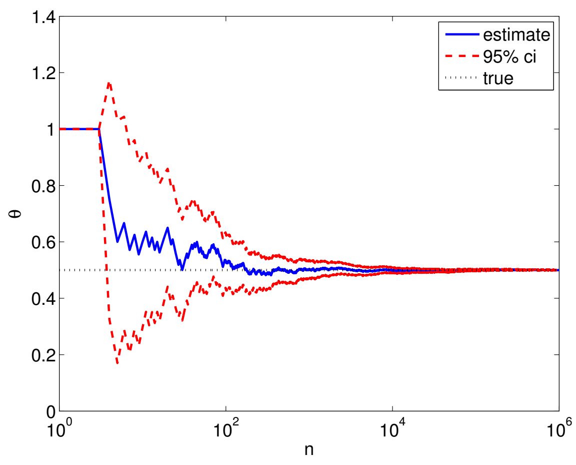
$$\text{RelErr}_{\theta;n} \sim \hat{\theta}_n^{-1/2} \ .$$

This means that, if the number of experiments is fixed, the relative error in predicting an event that occurs with 0.1% probability ($\theta = 0.001$) is 10 times larger than that for an event that occurs with 10% probability ($\theta = 0.1$). Combined with the convergence rate of $n^{-1/2}$, it takes 100 times as many experiments to achieve the similar level of relative error if the event is 100 times less likely. Thus, predicting the probability of a rare event is costly.

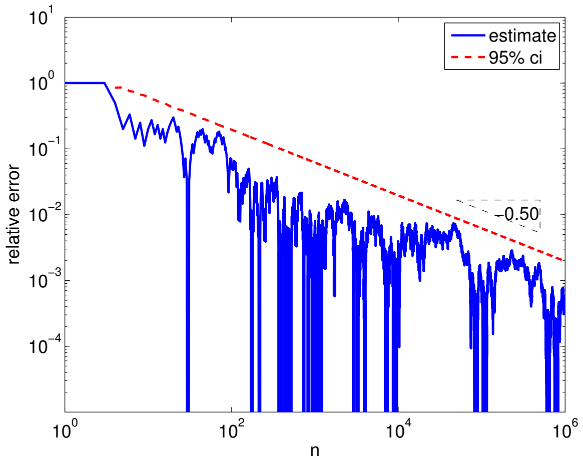
10.4 Cumulative Sample Means

In this subsection, we present a practical means of computing sample means. Let us denote the total number of coin flips by n_{max}, which defines the size of our sample. We assume $n_{\text{exp}} = 1$, as is almost always the case in practice. We create our sample of size n_{max}, and then for $n = 1, \ldots, n_{\text{max}}$ we compute a sequence of cumulative sample means. That is, we start with a realization of n_{max} coin tosses,

$$b_1, b_2, \ldots, b_n, \ldots, b_{n_{\text{max}}} \ ,$$



(a) value



(b) relative error

Figure 10.4: Cumulative sample mean, confidence intervals, and their convergence for a Bernoulli random variable ($\theta = 0.5$).

and then compute the cumulative values,

$$\hat\theta_1 = \bar b_1 = \frac{1}{1}\cdot b_1 \quad \text{and} \quad [\text{ci}]_1 = \left[\hat\theta_1 - z_\gamma\sqrt{\frac{\hat\theta_1(1-\hat\theta_1)}{1}},\ \hat\theta_1 + z_\gamma\sqrt{\frac{\hat\theta_1(1-\hat\theta_1)}{1}}\right]$$

$$\hat\theta_2 = \bar b_2 = \frac{1}{2}(b_1+b_2) \quad \text{and} \quad [\text{ci}]_2 = \left[\hat\theta_2 - z_\gamma\sqrt{\frac{\hat\theta_2(1-\hat\theta_2)}{2}},\ \hat\theta_2 + z_\gamma\sqrt{\frac{\hat\theta_2(1-\hat\theta_2)}{2}}\right]$$

$$\vdots$$

$$\hat\theta_n = \bar b_n = \frac{1}{n}\sum_{i=1}^{n} b_i \quad \text{and} \quad [\text{ci}]_n = \left[\hat\theta_n - z_\gamma\sqrt{\frac{\hat\theta_n(1-\hat\theta_n)}{n}},\ \hat\theta_n + z_\gamma\sqrt{\frac{\hat\theta_n(1-\hat\theta_n)}{n}}\right]$$

$$\vdots$$

$$\hat\theta_{n_{\max}} = \bar b_{n_{\max}} = \frac{1}{n}\sum_{i=1}^{n_{\max}} b_i \quad \text{and} \quad [\text{ci}]_{n_{\max}} = \left[\hat\theta_{n_{\max}} - z_\gamma\sqrt{\frac{\hat\theta_{n_{\max}}(1-\hat\theta_{n_{\max}})}{n_{\max}}},\ \hat\theta_{n_{\max}} + z_\gamma\sqrt{\frac{\hat\theta_{n_{\max}}(1-\hat\theta_{n_{\max}})}{n_{\max}}}\right] .$$

Note that the random variables $\overline{B}_1, \ldots, \overline{B}_{n_{\max}}$ realized by $\bar b_1, \ldots, \bar b_{n_{\max}}$ are not independent because the sample means are computed from the same set of realizations; also, the random variable, $[\text{CI}]_n$ realized by $[\text{ci}]_n$ are not joint with confidence γ. However in practice this is a computationally efficient way to estimate the parameter with typically only small loss in rigor. In particular, by plotting $\hat\theta_n$ and $[\text{ci}]_n$ for $n = 1, \ldots, n_{\max}$, one can deduce the convergence behavior of the simulation. In effect, we only perform one experiment, but we interpret it as $n_{\max}$ experiments.

Figure 10.4 shows an example of computing the sample means and confidence intervals in a cumulative manner. The figure confirms that the estimate converges to the true parameter value of $\theta = 0.5$. The confidence interval is a good indicator of the quality of the solution. The error (and the confidence interval) converges at the rate of $n^{-1/2}$, which agrees with the theory.

Chapter 11

Statistical Estimation: the Normal Density

We first review the "statistical process." We typically begin with some population we wish to characterize; we then draw a sample from this population; we then inspect the data — for example as a histogram — and postulate an underlying probability density (here taking advantage of the "frequency as probability" perspective); we then estimate the parameters of the density from the sample; and finally we are prepared to make inferences about the population. It is critical to note that in general we can "draw" from a population without knowing the underlying density; this in turn permits us to calibrate the postulated density.

We already observed one instance of this process with our coin flipping experiment. In this case, the population is all possible "behaviours" or flips of our coin; our sample is a finite number, n, of coin flips; our underlying probability density is Bernoulli. We then estimate the Bernoulli parameter — the probability of heads, θ — through our sample mean and associated (normal-approximation) confidence intervals. We are then prepared to make inferences: is the coin suitable to decide the opening moments of a football game? Note that in our experiments we effectively sample from a Bernoulli probability mass function with parameter θ but without knowing the value of θ.

Bernoulli estimation is very important, and occurs in everything from coin flips to area and integral estimation (by Monte Carlo techniques as introduced in Chapter 12) to political and product preference polls. However, there are many other important probability mass functions and densities that arise often in the prediction or modeling of various natural and engineering phenomena. Perhaps premier among the densities is the normal, or Gaussian, density.

We have introduced the univariate normal density in Section 9.4. In this chapter, to avoid confusion with typical variables in our next unit, regression, we shall denote our normal random variable as $W = W_{\mu,\sigma} \sim \mathcal{N}(\mu, \sigma^2)$ corresponding to probability density function $f_W(w) = f^{\text{normal}}(w; \mu, \sigma^2)$. We recall that the normal density is completely determined by the two parameters μ and σ which are in fact the mean and the standard deviation, respectively, of the normal density.

The normal density is ubiquitous for several reasons. First, more pragmatically, it has some rather intuitive characteristics: it is symmetric about the mean, it takes its maximum (the *mode*) at the mean (which is also the *median*, by symmetry), and it is characterized by just two parameters — a center (mean) and a spread (standard deviation). Second, and more profoundly, the normal density often arises "due" to the central limit theorem, described in Section 9.4.3. In short (in fact, way too short), one form of the central limit theorem states that the average of many random perturbations — perhaps described by different underlying probability densities — approaches the normal density. Since the behavior of many natural and engineered systems can be viewed as the consequence of many random influences, the normal density is often encountered in practice.

As an intuitive example from biostatistics, we consider the height of US females (see L Winner notes on Applied Statistics, University of Florida, http://www.stat.ufl.edu/~winner/statnotescomp/appstat.pdf Chapter 2, p 26). In this case our population is US females of ages 25–34. Our sample might be the US Census data of 1992. The histogram appears quite normal-like, and we can thus postulate a normal density. We would next apply the estimation procedures described below to determine the mean and standard deviation (the two parameters associated with our "chosen" density). Finally, we can make inferences — go beyond the sample to the population as whole — for example related to US females in 2012.

The choice of population is important both in the sampling/estimation stage and of course also in the inference stage. And the generation of appropriate samples can also be a very thorny issue. There is an immense literature on these topics which goes well beyond our scope and also, to a certain extent — given our focus on engineered rather than social and biological systems — beyond our immediate needs. As but one example, we would be remiss to apply the results from a population of US females to different demographics such as "females around the globe" or "US female jockeys" or indeed "all genders."

We should emphasize that the normal density is in almost all cases an approximation. For example, very rarely can a quantity take on all values however small or large, and in particular quantities must often be positive. Nevertheless, the normal density can remain a good approximation; for example if $\mu - 3\sigma$ is positive, then negative values are effectively "never seen." We should also emphasize that there are many cases in which the normal density is not appropriate — not even a good approximation. As always, the data must enter into the decision as to how to model the phenomenon — what probability density with what parameters will be most effective?

As an engineering example closer to home, we now turn to the Infra-Red Range Finder distance-voltage data of Chapter 1 of Unit I. It can be motivated that in fact distance D and voltage V are inversely related, and hence it is plausible to assume that $DV = C$, where C is a constant associated with our particular device. Of course, in actual practice, there will be measurement error, and we might thus plausibly assume that

$$(DV)^{\text{meas}} = C + W$$

where W is a normal random variable with density $\mathcal{N}(0, \sigma^2)$. Note we assume that the noise is centered about zero but of unknown variance. From the transformation property of Chapter 4, Example 9.4.5, we can further express our measurements as

$$(DV)^{\text{meas}} \sim \mathcal{N}(C, \sigma^2)$$

since if we add a constant to a zero-mean normal random variable we simply shift the mean. Note we now have a classical statistical estimation problem: determine the mean C and standard deviation σ of a normal density. (Note we had largely ignored noise in Unit I, though in fact in interpolation and differentiation noise is often present and even dominant; in such cases we prefer to "fit," as described in more detail in Unit III.)

In terms of the statistical process, our population is all possible outputs of our IR Range Finder device, our sample will be a finite number of distance-voltage measurements, $(DV)_i^{\text{meas}}$, $1 \le i \le n$, our estimation procedure is presented below, and finally our inference will be future predictions of distance from voltage readings — through our simple relation $D = C/V$. Of course, it will also be important to somehow justify or at least inspect our assumption that the noise is Gaussian.

We now present the standard and very simple estimation procedure for the normal density. We present the method in terms of particular realization: the connection to probability (and random variables) is through the frequentist interpretation. We presume that W is a normal random variable with mean μ and standard deviation σ.

We first draw a sample of size n, w_j, $1 \leq j \leq n$, from $f_W(w) = f^{\text{normal}}(w;\mu,\sigma^2)$. We then calculate the sample mean as

$$\overline{w}_n = \frac{1}{n}\sum_{j=1}^{n} w_j \ ,$$

and the sample standard deviation as

$$s_n = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(w_j - \overline{w}_n)^2} \ .$$

(Of course, the w_j, $1 \leq j \leq n$, are realizations of random variables W_j, $1 \leq j \leq n$, $\overline{w}_n$ is a realization of a *random variable* $\overline{W}_n$, and s_n is a realization of a random variable S_n.) Not surprisingly, $\overline{w}_n$, which is simply the average of the data, is an estimate for the mean, μ, and s_n, which is simply the standard deviation of the data, is an estimate for the standard deviation, σ. (The $n-1$ rather than n in the denominator of s_n is related to a particular choice of estimator and estimator properties; in any event, for n large, the difference is quite small.)

Finally, we calculate the confidence interval for the mean

$$[\text{ci}]_{\mu;n} = \left[\overline{w}_n - t_{\gamma,n-1}\frac{s_n}{\sqrt{n}}, \overline{w}_n + t_{\gamma,n-1}\frac{s_n}{\sqrt{n}}\right] \ ,$$

where γ is the confidence level and $t_{\gamma,n-1}$ is related to the Student-t distribution.[1] For the particular case of $\gamma = 0.95$ you can find values for $t_{\gamma=0.95,n}$ for various n (sample sizes) in a table in Unit III. Note that for large n, $t_{\gamma,n-1}$ approaches z_γ discussed earlier in the context of (normal–approximation) binomial confidence intervals.

We recall the meaning of this confidence interval. If we perform n_{exp} realizations (with $n_{\text{exp}} \to \infty$) — in which each realization corresponds to a (different) sample $w_1, \ldots, w_n$, and hence different sample mean $\overline{w}_n$, different sample standard deviation s_n, and different confidence interval $[\text{ci}]_{\mu;n}$ — then in a fraction γ of these realizations the true mean μ will reside within the confidence interval. (Or course this statement is only completely rigorous if the underlying density is precisely the normal density.)

We can also translate our confidence interval into an "error bound" (with confidence level γ). In particular, unfolding our confidence interval yields

$$|\mu - \overline{w}_n| \leq t_{\gamma,n-1}\frac{s_n}{\sqrt{n}} \equiv \text{ Half Length}_{\mu;n} \ .$$

We observe the "same" square root of n, sample size, that we observed in our Bernoulli estimation procedure, and in fact for the same reasons. Intuitively, say in our female height example, as we increase our sample size there are many more ways to obtain a sample mean close to μ (with much cancellation about the mean) than to obtain a sample mean say σ above μ (e.g., with all heights well above the mean). As you might expect, as γ increases, $t_{\gamma,n-1}$ also increases: if we insist upon greater certainty in our claims, then we will lose some accuracy as reflected in the Half Length of the confidence interval.

[1] The multiplier $t_{\gamma,n-1}$ satisfies $F^{\text{student}-\text{t}}(t_{\gamma,n-1}; n-1) = (\gamma+1)/2$ where $F^{\text{student}-\text{t}}(\,\cdot\,; n-1)$ is the cdf of the Student's-t distribution with $n-1$ degrees of freedom; i.e. $t_{\gamma,n-1}$ is the $(\gamma+1)/2$ quantile of the Student's-t distribution.

Chapter 12

Monte Carlo: Areas and Volumes

12.1 Calculating an Area

We have seen in Chapter 10 and 11 that parameters that describe probability distributions can be estimated using a finite number of realizations of the random variable and that furthermore we can construct an error bound (in the form of confidence interval) for such an estimate. In this chapter, we introduce Monte Carlo methods to estimate the area (or volume) of implicitly-defined regions. Specifically, we recast the area determination problem as a problem of estimating the mean of a certain Bernoulli distribution and then apply the tools developed in Chapter 10 to estimate the area and also assess errors in our estimate.

12.1.1 Objective

We are given a two-dimensional domain D in a rectangle $R = [a_1, b_1] \times [a_2, b_2]$. We would like to find, or estimate, the area of D, A_D. Note that the area of the bounding rectangle, $A_R = (b_1 - a_1)(b_2 - a_2)$, is known.

12.1.2 A Continuous Uniform Random Variable

Let $X \equiv (X_1, X_2)$ be a uniform random variable over R. We know that, by the definition of uniform distribution,

$$f_{X_1,X_2}(x_1, x_2) = \begin{cases} 1/A_R, & (x_1, x_2) \in R \\ 0, & (x_1, x_2) \notin R \end{cases} ,$$

and we know how to sample from f_{X_1,X_2} by using independence and univariate uniform densities. Finally, we can express the probability that X takes on a value in D as

$$P(X \in D) = \iint_D f_{X_1,X_2}(x_1, x_2)\, dx_1\, dx_2 = \frac{1}{A_R} \iint_D dx_1\, dx_2 = \frac{A_D}{A_R} .$$

Intuitively, this means that the probability of a random "dart" landing in D is equal to the fraction of the area that D occupies with respect to R.

12.1.3 A Bernoulli Random Variable

Let us introduce a Bernoulli random variable,

$$B = \begin{cases} 1 & X \in D \text{ with probability } \theta \\ 0 & X \notin D \text{ with probability } 1-\theta \end{cases} .$$

But,

$$P(X \in D) = A_D/A_R ,$$

So, by our usual transformation rules,

$$\theta \equiv \frac{A_D}{A_R} .$$

In other words, if we can estimate θ we can estimate $A_D = A_R\theta$. We know how to estimate θ — same as coin flips. But, how do we sample B if we do not know θ?

12.1.4 Estimation: Monte Carlo

We draw a sample of random vectors,

$$(x_1,x_2)_1,\ (x_1,x_2)_2,\dots,(x_1,x_2)_n,\dots,(x_1,x_2)_{n_{\max}}$$

and then map the sampled pairs to realization of Bernoulli random variables

$$(x_1,x_2)_i \to b_i, \quad i=1,\dots,n_{\max} .$$

Given the realization of Bernoulli random variables,

$$b_1,\dots,b_n,\dots,b_{n_{\max}} ,$$

we can apply the technique discussed in Section 10.4 to compute the sample means and confidence intervals: for $n = 1,\dots,n_{\max}$,

$$\hat\theta_n = \bar b_n = \frac{1}{n}\sum_{i=1}^{n} b_i \quad\text{and}\quad [\mathrm{ci}]_n = \left[\hat\theta_n - z_\gamma\sqrt{\frac{\hat\theta_n(1-\hat\theta_n)}{n}},\ \hat\theta_n + z_\gamma\sqrt{\frac{\hat\theta_n(1-\hat\theta_n)}{n}}\right] .$$

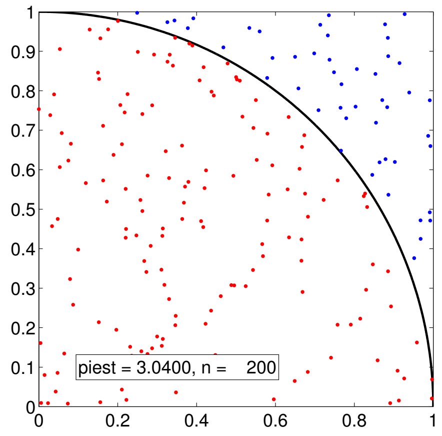
Thus, given the mapping from the sampled pairs to Bernoulli variables, we can estimate the parameter.

The only remaining question is how to construct the mapping $(x_1,x_2)_n \to b_n$, $n=1,\dots,n_{\max}$. The appropriate mapping is, given $(x_1,x_2)_n \in R$,

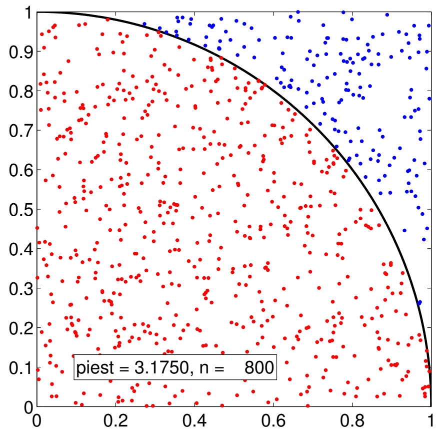
$$b_n = \begin{cases} 1, & (x_1,x_2)_n \in D \\ 0, & (x_1,x_2)_n \notin D \end{cases} .$$

To understand why this mapping works, we can look at the random variables: given $(X_1,X_2)_n$,

$$B_n = \begin{cases} 1, & (X_1,X_2)_n \in D \\ 0, & (X_1,X_2)_n \notin D \end{cases} .$$



(a) $n = 200$



(b) $n = 800$

Figure 12.1: Estimation of π by Monte Carlo method.

But,

$$P((X_1, X_2)_n \in D) = \theta \ ,$$

so

$$P(B_n = 1) = \theta \ ,$$

for $\theta = A_D/A_R$.

This procedure can be intuitively described as

1. Throw n "random darts" at R
2. Estimate $\theta = A_D/A_R$ by fraction of darts that land in D

Finally, for A_D, we can develop an estimate

$$(\widehat{A}_D)_n = A_R \hat{\theta}_n$$

and confidence interval

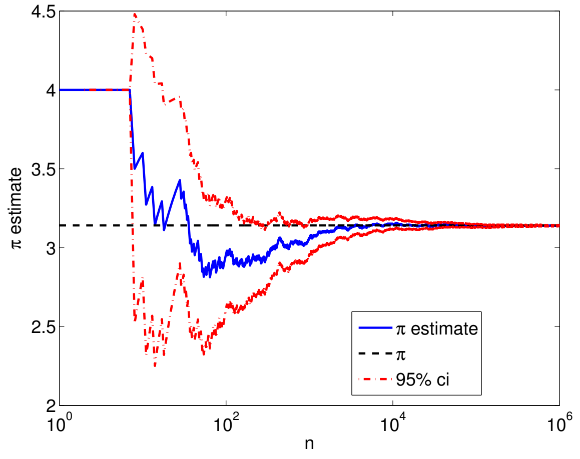
$$[\text{ci}_{A_D}]_n = A_R [\text{ci}]_n \ .$$

Example 12.1.1 Estimating π by Monte Carlo method

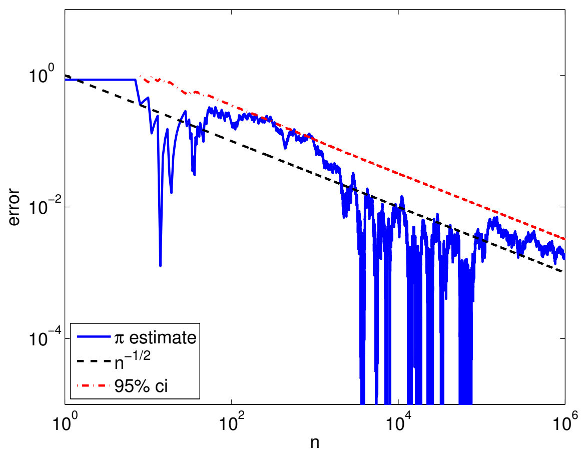
Let us consider an example of estimating the area using Monte Carlo method. In particular, we estimate the area of a circle with unit radius centered at the origin, which has the area of $\pi r^2 = \pi$. Noting the symmetry of the problem, let us estimate the area of the quarter circle in the first quadrant and then multiply the area by four to obtain the estimate of π. In particular, we sample from the square

$$R = [0, 1] \times [0, 1]$$

having the area of $A_R = 1$ and aim to determine the area of the quarter circle D with the area of A_D. Clearly, the analytical answer to this problem is $A_D = \pi/4$. Thus, by estimating A_D using the Monte Carlo method and then multiplying A_D by four, we can estimate the value π.



(a) value



(b) error

Figure 12.2: Convergence of the π estimate with the number of samples.

The sampling procedure is illustrated in Figure 12.1. To determine whether a given sample $(x_1, x_2)_n$ is in the quarter circle, we can compute its distance from the center and determine if the distance is greater than unity, i.e. the Bernoulli variable is assigned according to

$$b_n = \begin{cases} 1, & \sqrt{x_1^2 + x_2^2} \le 1 \\ 0, & \text{otherwise} \end{cases} .$$

The samples that evaluates to $b_n = 1$ and 0 are plotted in red and blue, respectively. Because the samples are drawn uniformly from the square, the fraction of red dots is equal to the fraction of the area occupied by the quarter circle. We show in Figure 12.2 the convergence of the Monte Carlo estimation: we observe the anticipated square-root behavior. Note in the remainder of this section we shall use the more conventional N rather than n for sample size.

·

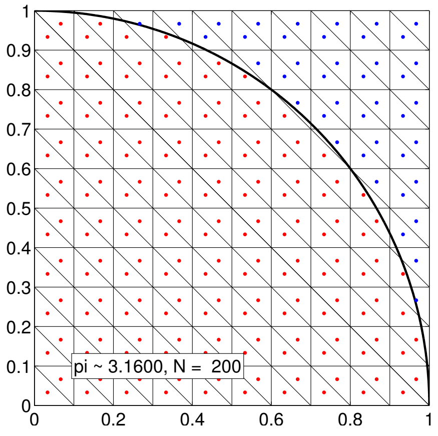
12.1.5 Estimation: Riemann Sum

As a comparison, let us also use the midpoint rule to find the area of a two-dimensional region D. We first note that the area of D is equal to the integral of a characteristic function

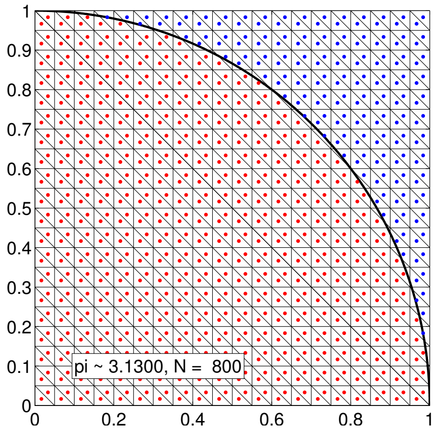
$$\chi(x_1, x_2) = \begin{cases} 1, & (x_1, x_2) \in D \\ 0, & \text{otherwise} \end{cases} ,$$

over the domain of integration R that encloses D. For simplicity, we consider rectangular domain $R = [a_1, b_1] \times [a_2, b_2]$. We discretize the domain into $N/2$ little rectangles, each with the width of $(b_1 - a_1)/\sqrt{N/2}$ and the height of $(b_2 - a_2)/\sqrt{N/2}$. We further divide each rectangle into two right triangles to obtain a triangulation of R. The area of each little triangle K is $A_K = (b_1 - a_1)(b_2 - a_2)/N$. Application of the midpoint rule to integrate the characteristic function yields

$$A_D \approx (\hat{A}_D^{\text{Rie}})_N = \sum_K A_K \chi(x_{c,K}) = \sum_K \frac{(b_1 - a_1)(b_2 - a_2)}{N} \chi(x_{c,K}) ,$$



(a) $N = 200$



(b) $N = 800$

Figure 12.3: Estimation of π by deterministic Riemann sum.

where $x_{c,K}$ is the centroid (i.e. the midpoint) of the triangle. Noting that $A_R = (b_1 - a_1)(b_2 - a_2)$ and rearranging the equation, we obtain

$$(\widehat{A}_D^{\text{Rie}})_N = A_R \frac{1}{N} \sum_K \chi(x_{c,K}) \ .$$

Because the characteristic function takes on either 0 or 1, the summation is simply equal to the number of times that $x_{c,K}$ is in the region D. Thus, we obtain our final expression

$$\frac{(\widehat{A}_D^{\text{Rie}})_N}{A_R} = \frac{\text{number of points in } D}{N} \ .$$

Note that the expression is very similar to that of the Monte Carlo integration. The main difference is that the sampling points are structured for the Riemann sum (i.e. the centroid of the triangles).

We can also estimate the error incurred in this process. First, we note that we cannot directly apply the error convergence result for the midpoint rule developed previously because the derivation relied on the smoothness of the integrand. The characteristic function is discontinuous along the boundary of D and thus is not smooth. To estimate the error, for simplicity, let us assume the domain size is the same in each dimension, i.e. $a = a_1 = a_2$ and $b = b_1 = b_2$. Then, the area of each square is

$$h^2 = (b - a)^2/N \ .$$

Then,

$$(\widehat{A}_D^{\text{Rie}})_N = (\text{number of points in } D) \cdot h^2 \ .$$

There are no errors created by little squares fully inside or fully outside D. All the error is due to the squares that intersect the perimeter. Thus, the error bound can be expressed as

$$\begin{aligned} \text{error} &\approx (\text{number of squares that intersect } D) \cdot h^2 \approx (\text{Perimeter}_D/h) \cdot h^2 \\ &= \mathcal{O}(h) = \mathcal{O}(\sqrt{A_R/N}) \ . \end{aligned}$$

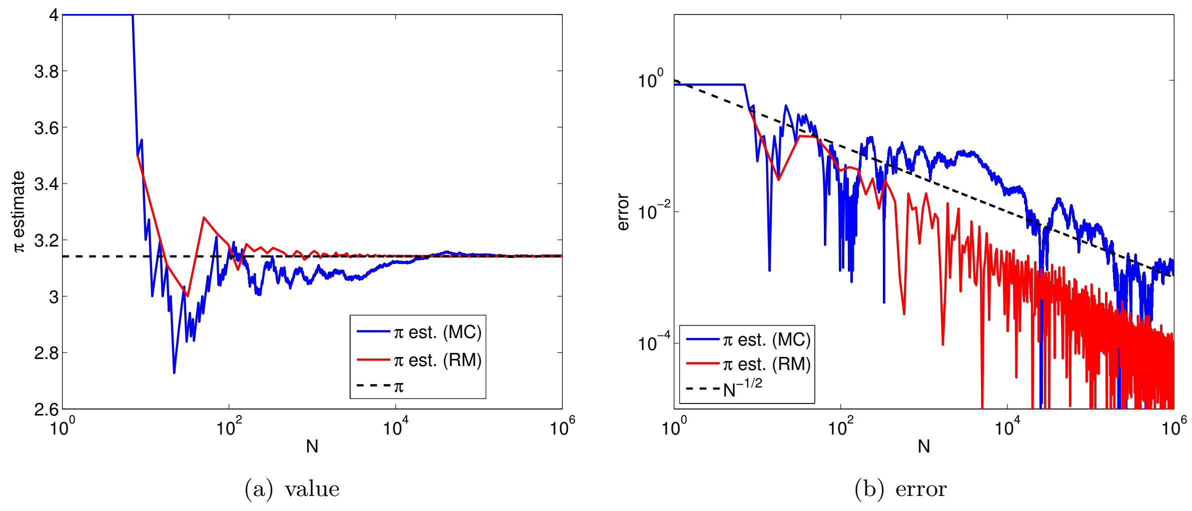


Figure 12.4: Convergence of the π estimate with the number of samples using Riemann sum.

Note that this is an example of *a priori* error estimate. In particular, unlike the error estimate based on the confidence interval of the sample mean for Monte Carlo, this estimate is not constant-free. That is, while we know the asymptotic rate at which the method converges, it does not tell us the actual magnitude of the error, which is problem-dependent. A constant-free estimate typically requires an *a posteriori* error estimate, which incorporates the information gathered about the particular problem of interest. We show in Figure 12.3 the Riemann sum grid, and in Figure 12.4 the convergence of the Riemann sum approach compared to the convergence of the Monte Carlo approach for our π example.

Example 12.1.2 Integration of a rectangular area

In the case of finding the area of a quarter circle, the Riemann sum performed noticeably better than the Monte Carlo method. However, this is not true in general. To demonstrate this, let us consider integrating the area of a rectangle. In particular, we consider the region

$$D = [0.2, 0.7] \times [0.2, 0.8] \ .$$

The area of the rectangle is $A_D = (0.7 - 0.2) \cdot (0.8 - 0.2) = 0.3$.

The Monte Carlo integration procedure applied to the rectangular area is illustrated in Figure 12.5(a). The convergence result in Figure 12.5(b) confirms that both Monte Carlo and Riemann sum converge at the rate of $N^{-1/2}$. Moreover, both methods produce the error of similar level for all ranges of the sample size N considered.

·

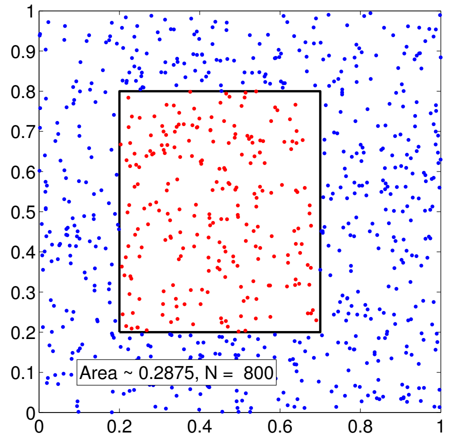
Example 12.1.3 Integration of a complex area

Let us consider estimating the area of a more complex region, shown in Figure 12.6(a). The region D is implicitly defined in the polar coordinate as

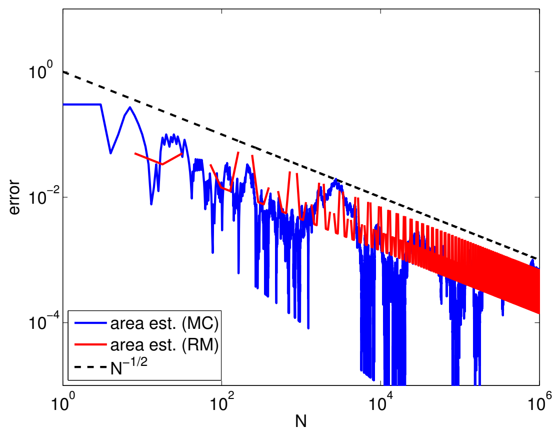
$$D = \left\{ (r, \theta) : r \le \frac{2}{3} + \frac{1}{3}\cos(4\beta\theta),\ 0 \le \theta \le \frac{\pi}{2} \right\},$$

where $r = \sqrt{x^2 + y^2}$ and $\tan(\theta) = y/x$. The particular case shown in the figure is for $\beta = 10$. For any natural number β, the area of the region D is equal to

$$A_D = \int_{\theta=0}^{\pi/2} \int_{r=0}^{2/3+1/3\cos(4\beta\theta)} r\, dr\, d\theta = \frac{\pi}{8}, \quad \beta = 1, 2, \dots \ .$$

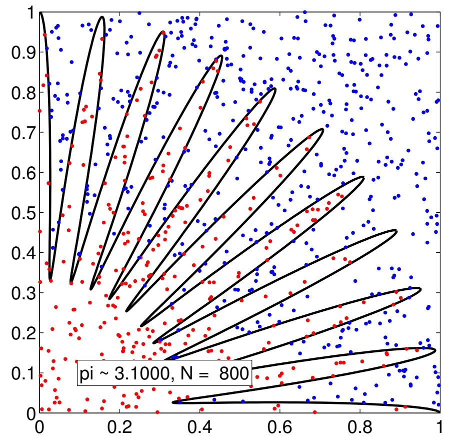


(a) Monte Carlo example

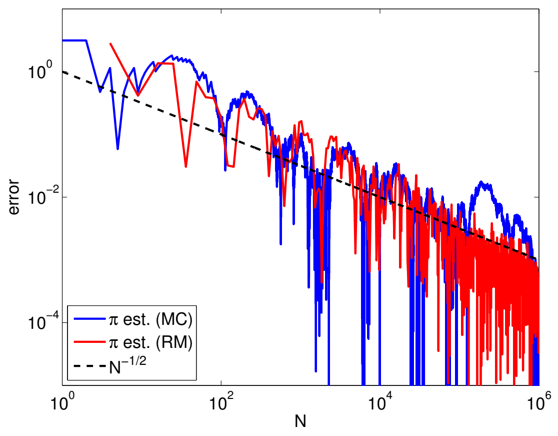


(b) error

Figure 12.5: The geometry and error convergence for the integration over a rectangular area.



(a) Monte Carlo example



(b) error

Figure 12.6: The geometry and error convergence for the integration over a more complex area.

Thus, we can again estimate π by multiplying the estimated area of D by 8.

The result of estimating π by approximating the area of D is shown in Figure 12.6(b). The error convergence plot confirms that both Monte Carlo and Riemann sum converge at the rate of $N^{-1/2}$. In fact, their performances are comparable for this slightly more complex domain.

12.2 Calculation of Volumes in Higher Dimensions

12.2.1 Three Dimensions

Both the Monte Carlo method and Riemann sum used to estimate the area of region D in two dimensions trivially extends to higher dimensions. Let us consider their applications in three dimensions.

Monte Carlo

Now, we sample (X_1, X_2, X_3) uniformly from a parallelepiped $R = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$, where the X_i's are mutually independent. Then, we assign a Bernoulli random variable according to whether (X_1, X_2, X_3) is inside or outside D as before, i.e.

$$B = \begin{cases} 1, & (X_1, X_2, X_3) \in D \\ 0, & \text{otherwise} \end{cases} .$$

Recall that the convergence of the sample mean to the true value — in particular the convergence of the confidence interval — is related to the Bernoulli random variable and not the X_i's. Thus, even in three dimensions, we still expect the error to converge as $N^{-1/2}$, where N is the size of the sample.

Riemann Sum

For simplicity, let us assume the parallelepiped is a cube, i.e. $a = a_1 = a_2 = a_3$ and $b = b_1 = b_2 = b_3$. We consider a grid of N points centered in little cubes of size

$$h^3 = \frac{b-a}{N} ,$$

such that $Nh^3 = V_R$. Extending the two-dimensional case, we estimate the volume of the region D according to

$$\frac{\widehat{V}_D^{\text{Rie}}}{V_R} = \frac{\text{number of points in } D}{N} .$$

However, unlike the Monte Carlo method, the error calculation is dependent on the dimension. The error is given by

$$\begin{aligned} \text{error} &\approx (\text{number of cubes that intersect } D) \cdot h^3 \\ &\approx (\text{surface area of } D/h^2) \cdot h^3 \\ &\approx h \approx N^{-1/3} . \end{aligned}$$

Note that the convergence rate has decreased from $N^{-1/2}$ to $N^{-1/3}$ in going from two to three dimensions.

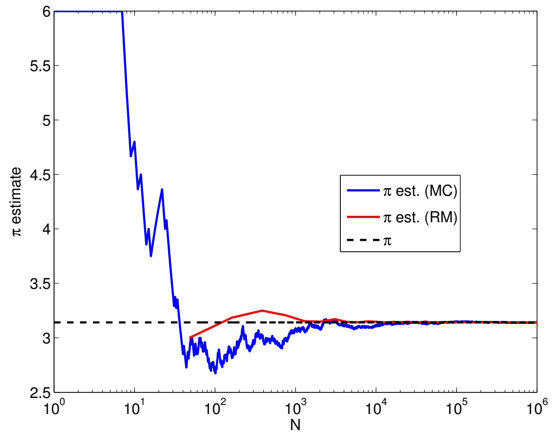
Example 12.2.1 Integration of a sphere
Let us consider a problem of finding the volume of a unit 1/8th sphere lying in the first octant. We sample from a unit cube

$$R = [0,1] \times [0,1] \times [0,1]$$

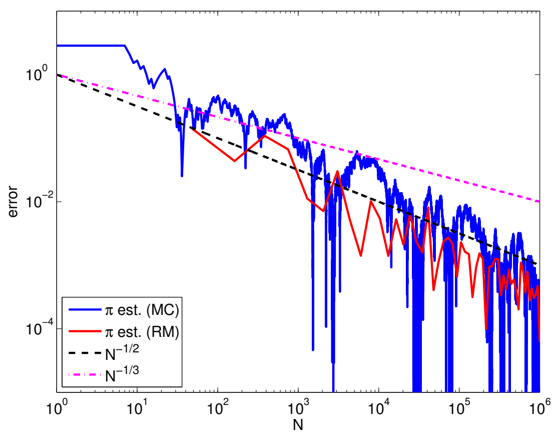
having a volume of $V_R = 1.0$. As in the case of circle in two dimensions, we can perform simple in/out check by measuring the distance of the point from the origin; i.e the Bernoulli variable is assigned according to

$$b_n = \begin{cases} 1, & \sqrt{x_1^2 + x_2^2 + x_3^2} \le 1 \\ 0, & \text{otherwise} \end{cases} .$$

The result of estimating the value of π based on the estimate of the volume of the 1/8th sphere is shown in Figure 12.7. (The raw estimated volume of the 1/8th sphere is scaled by 6.) As expected



(a) value



(b) error

Figure 12.7: Convergence of the π estimate using the volume of a sphere.

both the Monte Carlo method and the Riemann sum converge to the correct value. In particular, the Monte Carlo method converges at the expected rate of $N^{-1/2}$. The Riemann sum, on the other hand, converges at a faster rate than the expected rate of $N^{-1/3}$. This superconvergence is due to the symmetry in the tetrahedralization used in integration and the volume of interest. This does not contradict our *a priori* analysis, because the analysis tells us the asymptotic convergence rate for the worst case. The following example shows that the asymptotic convergence rate of the Riemann sum for a general geometry is indeed $N^{-1/2}$.

·

Example 12.2.2 Integration of a parallelpiped
Let us consider a simpler example of finding the volume of a parallelpiped described by

$$D = [0.1, 0.9] \times [0.2, 0.7] \times [0.1, 0.8] \ .$$

The volume of the parallelpiped is $V_D = 0.28$.

Figure 12.8 shows the result of the integration. The figure shows that the convergence rate of the Riemann sum is $N^{-1/3}$, which is consistent with the *a priori* analysis. On the other hand, the Monte Carlo method performs just as well as it did in two dimension, converging at the rate of $N^{-1/2}$. In particular, the Monte Carlo method performs noticeably better than the Riemann sum for large values of N.

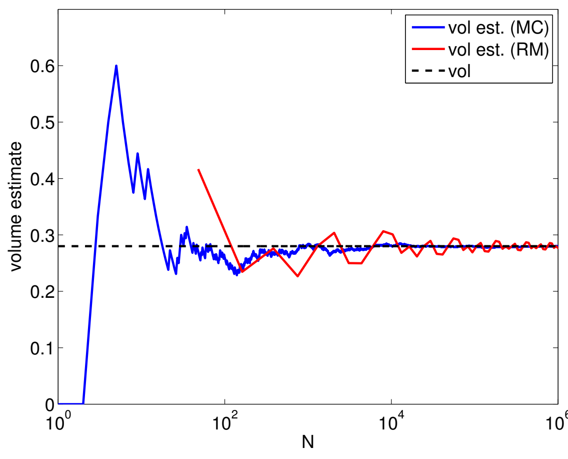
·

Example 12.2.3 Integration of a complex volume
Let us consider a more general geometry, with the domain defined in the spherical coordinate as

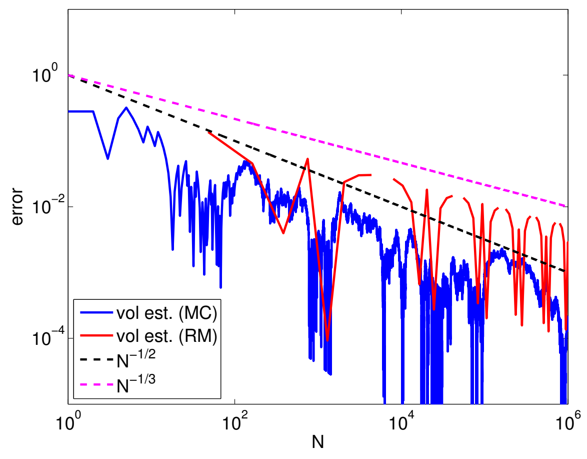
$$D = \left\{ (r, \theta, \phi) : r \leq \sin(\theta) \left(\frac{2}{3} + \frac{1}{3} \cos(40\phi) \right), \ 0 \leq \theta \leq \frac{\pi}{2}, \ 0 \leq \phi \leq \frac{\pi}{2} \right\} \ .$$

The volume of the region is given by

$$V_D = \int_{\phi=0}^{\pi/2} \int_{\theta=0}^{\pi/2} \int_{r=0}^{\sin(\theta)\left(\frac{2}{3}+\frac{1}{3}\cos(40\phi)\right)} r^2 \sin(\theta) \, dr \, d\theta \, d\phi = \frac{88}{2835}\pi \ .$$

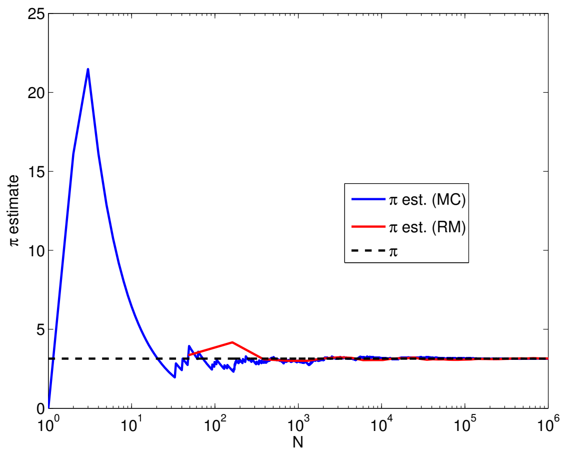


(a) value

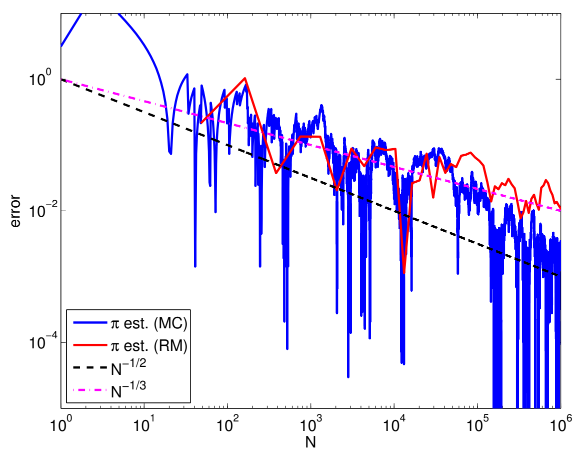


(b) error

Figure 12.8: Area of a parallelepiped.



(a) value



(b) error

Figure 12.9: Convergence of the π estimate using a complex three-dimensional integration.

Thus, we can estimate the value of π by first estimating the volume using Monte Carlo or Riemann sum, and then multiplying the result by 2835/88.

Figure 12.9 shows the result of performing the integration. The figure shows that the convergence rate of the Riemann sum is $N^{-1/3}$, which is consistent with the *a priori* analysis. On the other hand, the Monte Carlo method performs just as well as it did for the simple sphere case.

12.2.2 General d-Dimensions

Let us generalize our analysis to the case of integrating a general d-dimensional region. In this case, the Monte Carlo method considers a random d-vector, $(X_1, \ldots, X_d)$, and associate with the vector a Bernoulli random variable. The convergence of the Monte Carlo integration is dependent on the Bernoulli random variables and not directly affected by the random vector. In particular, the Monte Carlo method is oblivious to the length of the vector, d, i.e. the dimensionality of the space. Because the standard deviation of the binomial distribution scales as $N^{-1/2}$, we still expect the Monte Carlo method to converge at the rate of $N^{-1/2}$ regardless of d. Thus, Monte Carlo methods do not suffer from so-called curse of dimensionality, in which a method becomes intractable with increase of the dimension of the problem.

On the other hand, the performance of the Riemann sum is a function of the dimension of the space. In a d-dimensional space, each little cube has the volume of N^{-1}, and there are $N^{\frac{d-1}{d}}$ cube that intersect the boundary of D. Thus, the error scales as

$$\text{error} \approx N^{\frac{d-1}{d}} N^{-1} = N^{-1/d} \ .$$

The convergence rate worsens with the dimension, and this is an example of the curse of dimensionality. While the integration of a physical volume is typically limited to three dimensions, there are many instances in science and engineering where a higher-dimensional integration is required.

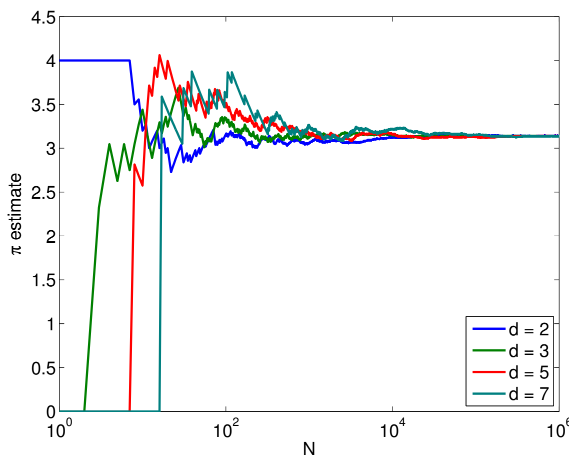
Example 12.2.4 integration over a hypersphere

To demonstrate that the convergence of Monte Carlo method is independent of the dimension, let us consider integration of a hypersphere in d-dimensional space. The volume of d-sphere is given by

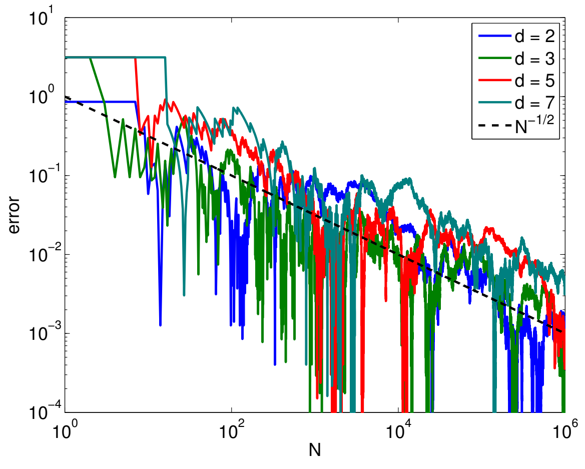
$$V_D = \frac{\pi^{d/2}}{\Gamma(n/2+1)} r^d \ ,$$

where Γ is the gamma function. We can again use the integration of a d-sphere to estimate the value of π.

The result of estimating the d-dimensional volume is shown in Figure 12.10 for $d = 2, 3, 5, 7$. The error convergence plot shows that the method converges at the rate of $N^{-1/2}$ for all d. The result confirms that Monte Carlo integration is a powerful method for integrating functions in higher-dimensional spaces.



(a) value



(b) error

Figure 12.10: Convergence of the π estimate using the Monte Carlo method on d-dimensional hyperspheres.

Chapter 13

Monte Carlo: General Integration Procedures

Chapter 14

Monte Carlo: Failure Probabilities

14.1 Calculating a Failure Probability

14.1.1 Objective

Let's say there is a set of "environmental" or "load" variables $(x_1, x_2, \dots)$ that affect the performance of an engineering system. For simplicity, let us restrict ourselves to the parameter size of two, so that we only have (x_1, x_2). We also assume that there are two "performance" metrics, $g_1(x_1, x_2)$ and $g_2(x_1, x_2)$. Without loss of generality, let's assume smaller g_1 and g_2 means better performance (we can always consider negative of the performance variable if larger values imply better performance). In fact, we assume that we wish to confirm that the performance metrics are below certain thresholds, i.e.

$$g_1(x_1, x_2) \le \tau_1 \quad \text{and} \quad g_2(x_1, x_2) \le \tau_2 \ . \tag{14.1}$$

Equivalently, we wish to avoid failure, which is defined as

$$g_1(x_1, x_2) > \tau_1 \quad \text{or} \quad g_2(x_1, x_2) > \tau_2 \ .$$

Note that in this chapter failure is interpreted liberally as the condition (14.1) even if this condition is not equivalent in any given situation as actual failure.

Suppose that (x_1, x_2) reside in some rectangle R. We now choose to interpret (x_1, x_2) as realizations of a random vector $X = (X_1, X_2)$ with prescribed probability density function $f_X(x_1, x_2) = f_{X_1,X_2}(x_1, x_2)$. We then wish to quantify the *failure probability* θ_F, defined by

$$\theta_F = P(g_1(X_1, X_2) > \tau_1 \text{ or } g_2(X_1, X_2) > \tau_2) \ .$$

We note that g_1 and g_2 are deterministic functions; however, because the argument to the functions are random variables, the output $g_1(X_1, X_2)$ and $g_2(X_1, X_2)$ are random variables. Thus, the failure is described probabilistically. If the bounds on the environmental variables (x_1, x_2) are known *a priori* one could design a system to handle the worst possible cases; however, the system design to handle very rare events may be over designed. Thus, a probabilistic approach may be appropriate in many engineering scenarios.

In any probabilistic simulation, we must make sure that the probability density of the random variable, f_X, is meaningful and that the interpretation of the probabilistic statement is relevant.

For example, in constructing the distribution, a good estimate may be obtained from statistical data (i.e. by sampling a population). The failure probability θ_F can be interpreted as either (i) probability of failure for the next "random" set of environmental or operating conditions, or (ii) frequency of failure over a population (based on the frequentist perspective).

14.1.2 An Integral

We now show that the computation of failure probability is similar to computation of an area. Let us define R to be the region from which $X = (X_1, X_2)$ is sampled (not necessarily uniformly). In other words, R encompasses all possible values that the environmental variable X can take. Let us also define D to be the region whose element $(x_1, x_2) \in D$ would lead to failure, i.e.

$$D \equiv \{(x_1, x_2) : g_1(x_1, x_2) > \tau_1 \quad \text{or} \quad g_2(x_1, x_2) > \tau_2\} \ .$$

Then, the failure probability can be expressed as an integral

$$\theta_F = \iint_D f_X(x_1, x_2) dx_1 dx_2 \ .$$

This requires a integration over the region D, which can be complicated depending on the failure criteria.

However, we can simplify the integral using the technique previously used to compute the area. Namely, we introduce a failure indicator or characteristic function,

$$\mathbf{1}_F(x_1, x_2) = \begin{cases} 1, & g_1(x_1, x_2) > \tau_1 \quad \text{or} \quad g_2(x_1, x_2) > \tau_2 \\ 0, & \text{otherwise} \end{cases} \ .$$

Using the failure indicator, we can write the integral over D as an integral over the simpler domain R, i.e.

$$\theta_F = \iint_R \mathbf{1}(x_1, x_2) f_X(x_1, x_2) \, dx_1 \, dx_2 \ .$$

Note that Monte Carlo methods can be used to evaluate any integral in any number of dimensions. The two main approaches are "hit or miss" and "sample mean," with the latter more efficient. Our case here is a natural example of the sample mean approach, though it also has the flavor of "hit or miss." In practice, variance reduction techniques are often applied to improve the convergence.

14.1.3 A Monte Carlo Approach

We can easily develop a Monte Carlo approach if we can reduce our problem to a Bernoulli random variable with parameter θ_F such that

$$B = \begin{cases} 1, & \text{with probability } \theta_F \\ 0, & \text{with probability } 1 - \theta_F \end{cases} \ .$$

Then, the computation of the failure probability θ_F becomes the estimation of parameter θ_F through sampling (as in the coin flip example). This is easy: we draw a random vector (X_1, X_2) from f_X — for example, uniform or normal — and then define

$$B = \begin{cases} 1, & g_1(X) > \tau_1 \quad \text{or} \quad g_2(X) > \tau_2 \\ 0, & \text{otherwise} \end{cases} \ . \tag{14.2}$$

Determination of B is easy assuming we can evaluate $g_1(x_1,x_2)$ and $g_2(x_1,x_2)$. But, by definition

$$\begin{aligned}\theta_F &= P(g_1(X) > \tau_1 \quad \text{or} \quad g_2(X) > \tau_2)\\ &= \iint_R \mathbf{1}_F(x_1,x_2) f_X(x_1,x_2)\,dx_1\,dx_2\ .\end{aligned}$$

Hence we have identified a Bernoulli random variable with the requisite parameter θ_F.

The Monte Carlo procedure is simple. First, we draw $n_{\max}$ random variables,

$$(X_1,X_2)_1,\ (X_1,X_2)_2,\ldots,(X_1,X_2)_n,\ldots,(X_1,X_2)_{n_{\max}}\ ,$$

and map them to Bernoulli random variables

$$(X_1,X_2)_n \to B_n \quad n = 1,\ldots,n_{\max}\ ,$$

according to (14.2). Using this mapping, we assign sample means, $\widehat{\Theta}_n$, and confidence intervals, $[\mathrm{CI}_F]_n$, according to

$$(\widehat{\Theta}_F)_n = \frac{1}{n}\sum_{i=1}^{n} B_i\ , \tag{14.3}$$

$$[\mathrm{CI}_F]_n = \left[(\widehat{\Theta}_F)_n - z_\gamma\sqrt{\frac{(\widehat{\Theta}_F)_n(1-(\widehat{\Theta}_F)_n)}{n}},\ (\widehat{\Theta}_F)_n + z_\gamma\sqrt{\frac{(\widehat{\Theta}_F)_n(1-(\widehat{\Theta}_F)_n)}{n}}\right]\ . \tag{14.4}$$

Note that in cases of failure, typically we would like θ_F to be very small. We recall from Section 10.3.3 that it is precisely this case for which the relative error RelErr is quite large (and furthermore for which the normal density confidence interval is only valid for quite large n). Hence, in practice, we must consider very large sample sizes in order to obtain relatively accurate results with reasonable confidence. More sophisticated approaches partially address these issues, but even these advanced approaches often rely on basic Monte Carlo ingredients.

Finally, we note that although the above description is for the cumulative approach we can also directly apply equations 14.3 and 14.4 for any fixed n. In this case we obtain $\Pr(\theta_F \in [\mathrm{CI}_f]_n) = \gamma$.

Unit III

Linear Algebra 1: Matrices and Least Squares. Regression.

Chapter 15

Motivation

In odometry-based mobile robot navigation, the accuracy of the robot's dead reckoning pose tracking depends on minimizing slippage between the robot's wheels and the ground. Even a momentary slip can lead to an error in heading that will cause the error in the robot's location estimate to grow linearly over its journey. It is thus important to determine the friction coefficient between the robot's wheels and the ground, which directly affects the robot's resistance to slippage. Just as importantly, this friction coefficient will significantly affect the performance of the robot: the ability to push loads.

When the mobile robot of Figure 15.1 is commanded to move forward, a number of forces come into play. Internally, the drive motors exert a torque (not shown in the figure) on the wheels, which is resisted by the friction force F_f between the wheels and the ground. If the magnitude of F_f dictated by the sum of the drag force F_drag (a combination of all forces resisting the robot's motion) and the product of the robot's mass and acceleration is less than the maximum static friction force $F^\mathrm{max}_\mathrm{f,\,static}$ between the wheels and the ground, the wheels will roll without slipping and the robot will move forward with velocity $v = \omega r_\mathrm{wheel}$. If, however, the magnitude of F_f reaches $F^\mathrm{max}_\mathrm{f,\,static}$, the wheels will begin to slip and F_f will drop to a lower level $F_\mathrm{f,\,kinetic}$, the kinetic friction force. The wheels will continue to slip ($v < \omega r_\mathrm{wheel}$) until zero relative motion between the wheels and the ground is restored (when $v = \omega r_\mathrm{wheel}$).

The critical value defining the boundary between rolling and slipping, therefore, is the maximum

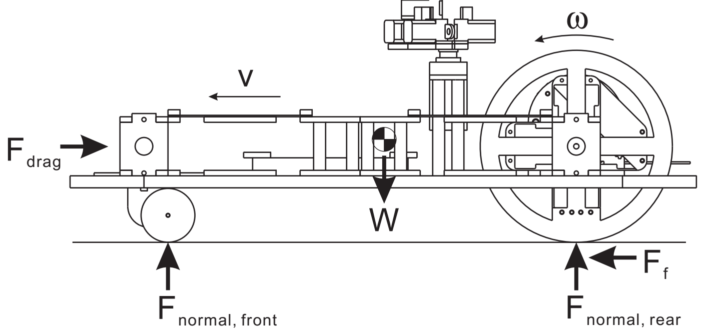


Figure 15.1: A mobile robot in motion.

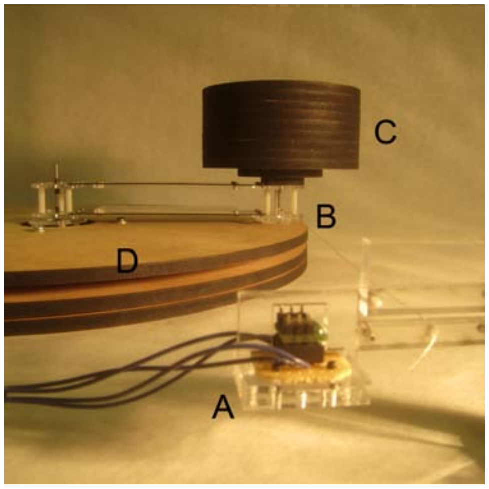


Figure 15.2: Experimental setup for friction measurement: Force transducer (A) is connected to contact area (B) by a thin wire. Normal force is exerted on the contact area by load stack (C). Tangential force is applied using turntable (D) via the friction between the turntable surface and the contact area. Apparatus and photograph courtesy of James Penn.

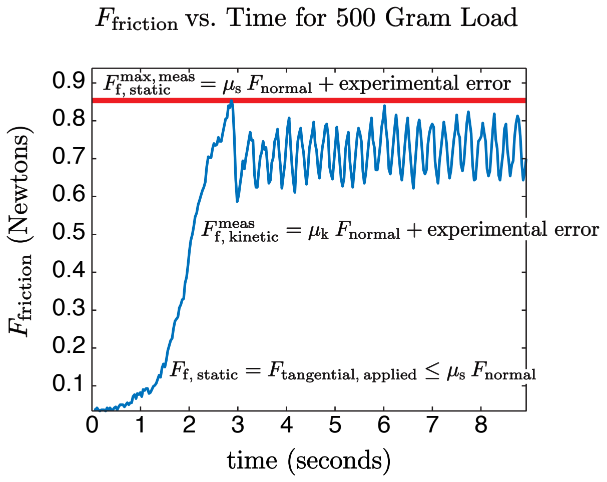


Figure 15.3: Sample data for one friction measurement, yielding one data point for $F^{\text{max, meas}}_{\text{f, static}}$. Data courtesy of James Penn.

static friction force. We expect that

$$F^{\text{max}}_{\text{f, static}} = \mu_s\, F_{\text{normal, rear}} \ , \tag{15.1}$$

where μ_s is the static coefficient of friction and $F_{\text{normal, rear}}$ is the normal force from the ground on the rear, driving, wheels. In order to minimize the risk of slippage (and to be able to push large loads), robot wheels should be designed for a high value of μ_s between the wheels and the ground. This value, although difficult to predict accurately by modeling, can be determined by experiment.

We first conduct experiments for the friction force $F^{\text{max}}_{\text{f, static}}$ (in Newtons) as a function of normal load $F_{\text{normal, applied}}$ (in Newtons) and (nominal) surface area of contact A_{surface} (in cm^2) with the friction turntable apparatus depicted in Figure 15.2. Weights permit us to vary the normal load and "washer" inserts permit us to vary the nominal surface area of contact. A typical experiment (at a particular prescribed value of $F_{\text{normal, applied}}$ and A_{surface}) yields the time trace of Figure 15.3 from which the $F^{\text{max, means}}_{\text{f, static}}$ (our *measurement* of $F^{\text{max}}_{\text{f, static}}$) is deduced as the maximum of the response.

We next postulate a dependence (or "model")

$$F^{\text{max}}_{\text{f, static}}(F_{\text{normal, applied}}, A_{\text{surface}}) = \beta_0 + \beta_1\, F_{\text{normal, applied}} + \beta_2\, A_{\text{surface}} \ , \tag{15.2}$$

where we expect — but do not *a priori* assume — from Messieurs Amontons and Coulomb that $\beta_0 = 0$ and $\beta_2 = 0$ (and of course $\beta_1 \equiv \mu_s$). In order to confirm that $\beta_0 = 0$ and $\beta_2 = 0$ — or at least confirm that $\beta_0 = 0$ and $\beta_2 = 0$ is not untrue — and to find a good estimate for $\beta_1 \equiv \mu_s$, we must appeal to our measurements.

The mathematical techniques by which to determine μ_s (and β_0, β_2) "with some confidence" from noisy experimental data is known as regression, which is the subject of Chapter 19. Regression, in turn, is best described in the language of linear algebra (Chapter 16), and is built upon the linear algebra concept of least squares (Chapter 17).

Chapter 16

Matrices and Vectors: Definitions and Operations

16.1 Basic Vector and Matrix Operations

16.1.1 Definitions

Let us first introduce the primitive objects in linear algebra: vectors and matrices. A m-vector $v \in \mathbb{R}^{m\times 1}$ consists of m real numbers [1]

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}.$$

It is also called a column vector, which is the default vector in linear algebra. Thus, by convention, $v \in \mathbb{R}^m$ implies that v is a column vector in $\mathbb{R}^{m\times 1}$. Note that we use subscript $(\cdot)_i$ to address the i-th component of a vector. The other kind of vector is a row vector $v \in \mathbb{R}^{1\times n}$ consisting of n entries

$$v = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix}.$$

Let us consider a few examples of column and row vectors.

Example 16.1.1 vectors
Examples of (column) vectors in $\mathbb{R}^3$ are

$$v = \begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix}, \quad u = \begin{pmatrix} \sqrt{3} \\ -7 \\ \pi \end{pmatrix}, \quad \text{and} \quad w = \begin{pmatrix} 9.1 \\ 7/3 \\ \sqrt{\pi} \end{pmatrix}.$$

[1] The concept of vectors readily extends to complex numbers, but we only consider real vectors in our presentation of this chapter.

To address a specific component of the vectors, we write, for example, $v_1 = 1$, $u_1 = \sqrt{3}$, and $w_3 = \sqrt{\pi}$. Examples of row vectors in $\mathbb{R}^{1\times 4}$ are

$$v = \begin{pmatrix} 2 & -5 & \sqrt{2} & e \end{pmatrix} \quad \text{and} \quad u = \begin{pmatrix} -\sqrt{\pi} & 1 & 1 & 0 \end{pmatrix}.$$

Some of the components of these row vectors are $v_2 = -5$ and $u_4 = 0$.

·

A matrix $A \in \mathbb{R}^{m\times n}$ consists of m rows and n columns for the total of $m \cdot n$ entries,

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{pmatrix}.$$

Extending the convention for addressing an entry of a vector, we use subscript $(\cdot)_{ij}$ to address the entry on the i-th row and j-th column. Note that the order in which the row and column are referred follows that for describing the size of the matrix. Thus, $A \in \mathbb{R}^{m\times n}$ consists of entries

$$A_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, n .$$

Sometimes it is convenient to think of a (column) vector as a special case of a matrix with only one column, i.e., $n = 1$. Similarly, a (row) vector can be thought of as a special case of a matrix with $m = 1$. Conversely, an $m \times n$ matrix can be viewed as m row n-vectors or n column m-vectors, as we discuss further below.

Example 16.1.2 matrices
Examples of matrices are

$$A = \begin{pmatrix} 1 & \sqrt{3} \\ -4 & 9 \\ \pi & -3 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 & 1 \\ -2 & 8 & 1 \\ 0 & 3 & 0 \end{pmatrix}.$$

The matrix A is a 3×2 matrix ($A \in \mathbb{R}^{3\times 2}$) and matrix B is a 3×3 matrix ($B \in \mathbb{R}^{3\times 3}$). We can also address specific entries as, for example, $A_{12} = \sqrt{3}$, $A_{31} = -4$, and $B_{32} = 3$.

·

While vectors and matrices may appear like arrays of numbers, linear algebra defines special set of rules to manipulate these objects. One such operation is the transpose operation considered next.

Transpose Operation

The first linear algebra operator we consider is the transpose operator, denoted by superscript $(\cdot)^{\mathrm{T}}$. The transpose operator swaps the rows and columns of the matrix. That is, if $B = A^{\mathrm{T}}$ with $A \in \mathbb{R}^{m\times n}$, then

$$B_{ij} = A_{ji}, \quad 1 \le i \le n, \quad 1 \le j \le m .$$

Because the rows and columns of the matrix are swapped, the dimensions of the matrix are also swapped, i.e., if $A \in \mathbb{R}^{m\times n}$ then $B \in \mathbb{R}^{n\times m}$.

If we swap the rows and columns twice, then we return to the original matrix. Thus, the transpose of a transposed matrix is the original matrix, i.e.

$$(A^{\mathrm{T}})^{\mathrm{T}} = A .$$

Example 16.1.3 transpose
Let us consider a few examples of transpose operation. A matrix A and its transpose $B = A^{\mathrm{T}}$ are related by

$$A = \begin{pmatrix} 1 & \sqrt{3} \\ -4 & 9 \\ \pi & -3 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 & -4 & \pi \\ \sqrt{3} & 9 & -3 \end{pmatrix} .$$

The rows and columns are swapped in the sense that $A_{31} = B_{13} = \pi$ and $A_{12} = B_{21} = \sqrt{3}$. Also, because $A \in \mathbb{R}^{3\times 2}$, $B \in \mathbb{R}^{2\times 3}$. Interpreting a vector as a special case of a matrix with one column, we can also apply the transpose operator to a column vector to create a row vector, i.e., given

$$v = \begin{pmatrix} \sqrt{3} \\ -7 \\ \pi \end{pmatrix} ,$$

the transpose operation yields

$$u = v^{\mathrm{T}} = \begin{pmatrix} \sqrt{3} & -7 & \pi \end{pmatrix} .$$

Note that the transpose of a column vector is a row vector, and the transpose of a row vector is a column vector.

·

16.1.2 Vector Operations

The first vector operation we consider is multiplication of a vector $v \in \mathbb{R}^m$ by a scalar $\alpha \in \mathbb{R}$. The operation yields

$$u = \alpha v ,$$

where each entry of $u \in \mathbb{R}^m$ is given by

$$u_i = \alpha v_i, \quad i = 1, \ldots, m .$$

In other words, multiplication of a vector by a scalar results in each component of the vector being scaled by the scalar.

The second operation we consider is addition of two vectors $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$. The addition yields

$$u = v + w ,$$

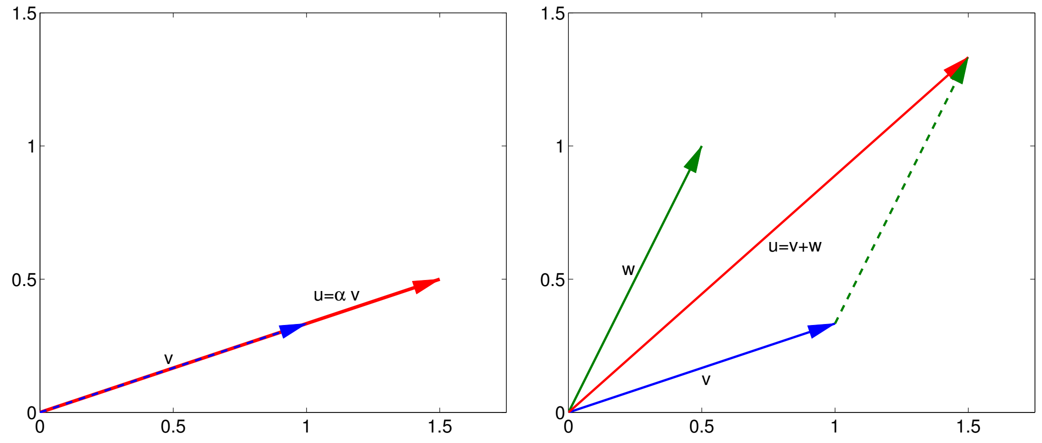
where each entry of $u \in \mathbb{R}^m$ is given by

$$u_i = v_i + w_i, \quad i = 1, \ldots, m .$$

In order for addition of two vectors to make sense, the vectors must have the same number of components. Each entry of the resulting vector is simply the sum of the corresponding entries of the two vectors.

We can summarize the action of the scaling and addition operations in a single operation. Let $v \in \mathbb{R}^m$, $w \in \mathbb{R}^m$ and $\alpha \in \mathbb{R}$. Then, the operation

$$u = v + \alpha w$$



(a) scalar scaling (b) vector addition

Figure 16.1: Illustration of vector scaling and vector addition.

yields a vector $u \in \mathbb{R}^m$ whose entries are given by

$$u_i = v_i + \alpha w_i, \quad i = 1, \ldots, m \ .$$

The result is nothing more than a combination of the scalar multiplication and vector addition rules.

Example 16.1.4 vector scaling and addition in $\mathbb{R}^2$
Let us illustrate scaling of a vector by a scalar and addition of two vectors in $\mathbb{R}^2$ using

$$v = \begin{pmatrix} 1 \\ 1/3 \end{pmatrix} \quad , w = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix}, \quad \text{and} \quad \alpha = \frac{3}{2} \ .$$

First, let us consider scaling of the vector v by the scalar α. The operation yields

$$u = \alpha v = \frac{3}{2} \begin{pmatrix} 1 \\ 1/3 \end{pmatrix} = \begin{pmatrix} 3/2 \\ 1/2 \end{pmatrix} .$$

This operation is illustrated in Figure 16.1(a). The vector v is simply stretched by the factor of $3/2$ while preserving the direction.

Now, let us consider addition of the vectors v and w. The vector addition yields

$$u = v + w = \begin{pmatrix} 1 \\ 1/3 \end{pmatrix} + \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3/2 \\ 4/3 \end{pmatrix} .$$

Figure 16.1(b) illustrates the vector addition process. We translate w so that it starts from the tip of v to form a parallelogram. The resultant vector is precisely the sum of the two vectors. Note that the geometric intuition for scaling and addition provided for $\mathbb{R}^2$ readily extends to higher dimensions.

·

Example 16.1.5 vector scaling and addition in $\mathbb{R}^3$

Let $v = \begin{pmatrix} 1 & 3 & 6 \end{pmatrix}^{\mathrm{T}}$, $w = \begin{pmatrix} 2 & -1 & 0 \end{pmatrix}^{\mathrm{T}}$, and $\alpha = 3$. Then,

$$u = v + \alpha w = \begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix} + 3 \cdot \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix} + \begin{pmatrix} 6 \\ -3 \\ 0 \end{pmatrix} = \begin{pmatrix} 7 \\ 0 \\ 6 \end{pmatrix} .$$

·

Inner Product

Another important operation is the inner product. This operation takes two vectors of the same dimension, $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$, and yields a scalar $\beta \in \mathbb{R}$:

$$\beta = v^{\mathrm{T}} w \quad \text{where} \quad \beta = \sum_{i=1}^{m} v_i w_i \ .$$

The appearance of the transpose operator will become obvious once we introduce the matrix-matrix multiplication rule. The inner product in a Euclidean vector space is also commonly called the dot product and is denoted by $\beta = v \cdot w$. More generally, the inner product of two elements of a vector space is denoted by $(\cdot, \cdot)$, i.e., $\beta = (v, w)$.

Example 16.1.6 inner product

Let us consider two vectors in $\mathbb{R}^3$, $v = \begin{pmatrix} 1 & 3 & 6 \end{pmatrix}^{\mathrm{T}}$ and $w = \begin{pmatrix} 2 & -1 & 0 \end{pmatrix}^{\mathrm{T}}$. The inner product of these two vectors is

$$\beta = v^{\mathrm{T}} w = \sum_{i=1}^{3} v_i w_i = 1 \cdot 2 + 3 \cdot (-1) + 6 \cdot 0 = -1 \ .$$

·

Norm (2-Norm)

Using the inner product, we can naturally define the 2-norm of a vector. Given $v \in \mathbb{R}^m$, the 2-norm of v, denoted by $\|v\|_2$, is defined by

$$\|v\|_2 = \sqrt{v^{\mathrm{T}} v} = \sqrt{\sum_{i=1}^{m} v_i^2} \ .$$

Note that the norm of any vector is non-negative, because it is a sum m non-negative numbers (squared values). The ℓ_2 norm is the usual Euclidean length; in particular, for $m = 2$, the expression simplifies to the familiar Pythagorean theorem, $\|v\|_2 = \sqrt{v_1^2 + v_2^2}$. While there are other norms, we almost exclusively use the 2-norm in this unit. Thus, for notational convenience, we will drop the subscript 2 and write the 2-norm of v as $\|v\|$ with the implicit understanding $\| \cdot \| \equiv \| \cdot \|_2$.

By definition, any norm must satisfy the triangle inequality,

$$\|v + w\| \leq \|v\| + \|w\| \ ,$$

for any $v, w \in \mathbb{R}^m$. The theorem states that the sum of the lengths of two adjoining segments is longer than the distance between their non-joined end points, as is intuitively clear from Figure 16.1(b). For norms defined by inner products, as our 2-norm above, the triangle inequality is automatically satisfied.

Proof. For norms induced by an inner product, the proof of the triangle inequality follows directly from the definition of the norm and the Cauchy-Schwarz inequality. First, we expand the expression as

$$\|v+w\|^2 = (v+w)^{\mathrm{T}}(v+w) = v^{\mathrm{T}}v + 2v^{\mathrm{T}}w + w^{\mathrm{T}}w \ .$$

The middle terms can be bounded by the Cauchy-Schwarz inequality, which states that

$$v^{\mathrm{T}}w \le |v^{\mathrm{T}}w| \le \|v\|\|w\| \ .$$

Thus, we can bound the norm as

$$\|v+w\|^2 \le \|v\|^2 + 2\|v\|\|w\| + \|w\|^2 = (\|v\| + \|w\|)^2 \ ,$$

and taking the square root of both sides yields the desired result. □

Example 16.1.7 norm of a vector
Let $v = \begin{pmatrix} 1 & 3 & 6 \end{pmatrix}^{\mathrm{T}}$ and $w = \begin{pmatrix} 2 & -1 & 0 \end{pmatrix}^{\mathrm{T}}$. The ℓ_2 norms of these vectors are

$$\|v\| = \sqrt{\sum_{i=1}^{3} v_i^2} = \sqrt{1^2 + 3^2 + 6^2} = \sqrt{46}$$

$$\text{and} \quad \|w\| = \sqrt{\sum_{i=1}^{3} w_i^2} = \sqrt{2^2 + (-1)^2 + 0^2} = \sqrt{5} \ .$$

·

Example 16.1.8 triangle inequality
Let us illustrate the triangle inequality using two vectors

$$v = \begin{pmatrix} 1 \\ 1/3 \end{pmatrix} \quad \text{and} \quad w = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} .$$

The length (or the norm) of the vectors are

$$\|v\| = \sqrt{\frac{10}{9}} \approx 1.054 \quad \text{and} \quad \|w\| = \sqrt{\frac{5}{4}} \approx 1.118 \ .$$

On the other hand, the sum of the two vectors is

$$v + w = \begin{pmatrix} 1 \\ 1/3 \end{pmatrix} + \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3/2 \\ 4/3 \end{pmatrix} ,$$

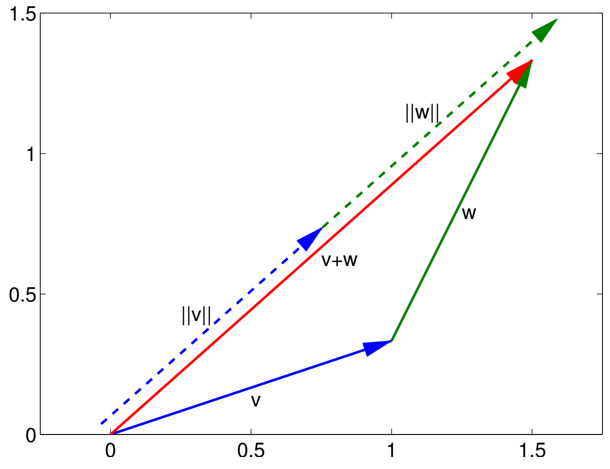


Figure 16.2: Illustration of the triangle inequality.

and its length is

$$\|v+w\| = \frac{\sqrt{145}}{6} \approx 2.007 \ .$$

The norm of the sum is shorter than the sum of the norms, which is

$$\|v\| + \|w\| \approx 2.172 \ .$$

This inequality is illustrated in Figure 16.2. Clearly, the length of $v+w$ is strictly less than the sum of the lengths of v and w (unless v and w align with each other, in which case we obtain equality).

In two dimensions, the inner product can be interpreted as

$$v^{\mathrm{T}} w = \|v\| \|w\| \cos(\theta) \ , \tag{16.1}$$

where θ is the angle between v and w. In other words, the inner product is a measure of how well v and w align with each other. Note that we can show the Cauchy-Schwarz inequality from the above equality. Namely, $|\cos(\theta)| \leq 1$ implies that

$$|v^{\mathrm{T}} w| = \|v\| \|w\| |\cos(\theta)| \leq \|v\| \|w\| \ .$$

In particular, we see that the inequality holds with equality if and only if $\theta = 0$ or π, which corresponds to the cases where v and w align. It is easy to demonstrate Eq. (16.1) in two dimensions.

Proof. Noting $v, w \in \mathbb{R}^2$, we express them in polar coordinates

$$v = \|v\| \begin{pmatrix} \cos(\theta_v) \\ \sin(\theta_v) \end{pmatrix} \quad \text{and} \quad w = \|w\| \begin{pmatrix} \cos(\theta_w) \\ \sin(\theta_w) \end{pmatrix} .$$

The inner product of the two vectors yield

$$\begin{aligned}
\beta = v^{\mathrm{T}} w = \sum_{i=1}^{2} v_i w_i &= \|v\| \cos(\theta_v) \|w\| \cos(\theta_w) + \|v\| \sin(\theta_v) \|w\| \sin(\theta_w) \\
&= \|v\| \|w\| \big(\cos(\theta_v)\cos(\theta_w) + \sin(\theta_v)\sin(\theta_w)\big) \\
&= \|v\| \|w\| \left(\frac{1}{2}(e^{i\theta_v} + e^{-i\theta_v}) \frac{1}{2}(e^{i\theta_w} + e^{-i\theta_w}) + \frac{1}{2i}(e^{i\theta_v} - e^{-i\theta_v}) \frac{1}{2i}(e^{i\theta_w} - e^{-i\theta_w}) \right) \\
&= \|v\| \|w\| \left(\frac{1}{4}\left(e^{i(\theta_v+\theta_w)} + e^{-i(\theta_v+\theta_w)} + e^{i(\theta_v-\theta_w)} + e^{-i(\theta_v-\theta_w)} \right) \right. \\
&\qquad \left. - \frac{1}{4}\left(e^{i(\theta_v+\theta_w)} + e^{-i(\theta_v+\theta_w)} - e^{i(\theta_v-\theta_w)} - e^{-i(\theta_v-\theta_w)} \right) \right) \\
&= \|v\| \|w\| \left(\frac{1}{2}\left(e^{i(\theta_v-\theta_w)} + e^{-i(\theta_v-\theta_w)} \right) \right) \\
&= \|v\| \|w\| \cos(\theta_v - \theta_w) = \|v\| \|w\| \cos(\theta) \ ,
\end{aligned}$$

where the last equality follows from the definition $\theta \equiv \theta_v - \theta_w$. □

Begin Advanced Material

For completeness, let us introduce a more general class of norms.

Example 16.1.9 *p*-norms
The 2-norm, which we will almost exclusively use, belong to a more general class of norms, called the p-norms. The p-norm of a vector $v \in \mathbb{R}^m$ is

$$\|v\|_p = \left(\sum_{i=1}^{m} |v_i|^p \right)^{1/p} ,$$

where $p \geq 1$. Any p-norm satisfies the positivity requirement, the scalar scaling requirement, and the triangle inequality. We see that 2-norm is a case of p-norm with $p = 2$.

Another case of p-norm that we frequently encounter is the 1-norm, which is simply the sum of the absolute value of the entries, i.e.

$$\|v\|_1 = \sum_{i=1}^{m} |v_i| \ .$$

The other one is the infinity norm given by

$$\|v\|_\infty = \lim_{p \to \infty} \|v\|_p = \max_{i=1,\ldots,m} |v_i| \ .$$

In other words, the infinity norm of a vector is its largest entry in absolute value.

·

End Advanced Material

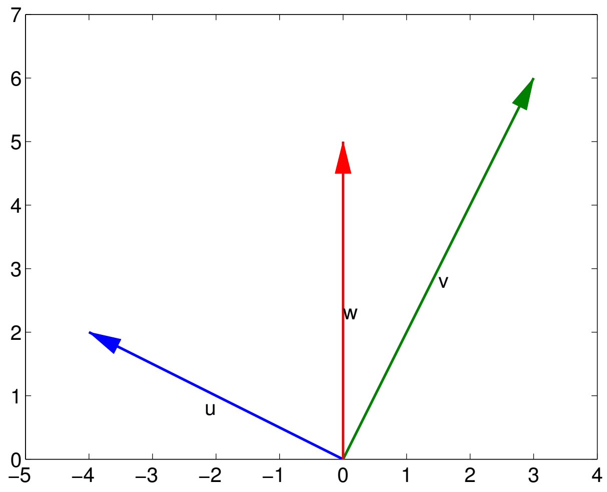


Figure 16.3: Set of vectors considered to illustrate orthogonality.

Orthogonality

Two vectors $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$ are said to be orthogonal to each other if

$$v^{\mathrm{T}} w = 0 \ .$$

In two dimensions, it is easy to see that

$$v^{\mathrm{T}} w = \|v\| \|w\| \cos(\theta) = 0 \quad \Rightarrow \quad \cos(\theta) = 0 \quad \Rightarrow \quad \theta = \pi/2 \ .$$

That is, the angle between v and w is $\pi/2$, which is the definition of orthogonality in the usual geometric sense.

Example 16.1.10 orthogonality
Let us consider three vectors in $\mathbb{R}^2$,

$$u = \begin{pmatrix} -4 \\ 2 \end{pmatrix}, \quad v = \begin{pmatrix} 3 \\ 6 \end{pmatrix}, \quad \text{and} \quad w = \begin{pmatrix} 0 \\ 5 \end{pmatrix},$$

and compute three inner products formed by these vectors:

$$\begin{aligned} u^{\mathrm{T}} v &= -4 \cdot 3 + 2 \cdot 6 = 0 \\ u^{\mathrm{T}} w &= -4 \cdot 0 + 2 \cdot 5 = 10 \\ v^{\mathrm{T}} w &= 3 \cdot 0 + 6 \cdot 5 = 30 \ . \end{aligned}$$

Because $u^{\mathrm{T}} v = 0$, the vectors u and v are orthogonal to each other. On the other hand, $u^{\mathrm{T}} w \neq 0$ and the vectors u and w are not orthogonal to each other. Similarly, v and w are not orthogonal to each other. These vectors are plotted in Figure 16.3; the figure confirms that u and v are orthogonal in the usual geometric sense.

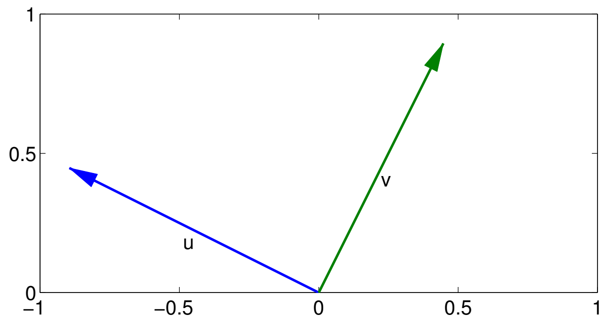


Figure 16.4: An orthonormal set of vectors.

Orthonormality

Two vectors $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$ are said to be orthonormal to each other if they are orthogonal to each other and each has unit length, i.e.

$$v^{\mathrm{T}} w = 0 \quad \text{and} \quad \|v\| = \|w\| = 1 \ .$$

Example 16.1.11 orthonormality
Two vectors

$$u = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 \\ 1 \end{pmatrix} \quad \text{and} \quad v = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

are orthonormal to each other. It is straightforward to verify that they are orthogonal to each other

$$u^{\mathrm{T}} v = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 \\ 1 \end{pmatrix}^{\mathrm{T}} \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} -2 \\ 1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 0$$

and that each of them have unit length

$$\|u\| = \sqrt{\frac{1}{5}((-2)^2 + 1^2)} = 1$$
$$\|v\| = \sqrt{\frac{1}{5}((1)^2 + 2^2)} = 1 \ .$$

Figure 16.4 shows that the vectors are orthogonal and have unit length in the usual geometric sense.

16.1.3 Linear Combinations

Let us consider a set of n m-vectors

$$v^1 \in \mathbb{R}^m,\ v^2 \in \mathbb{R}^m, \dots, v^n \in \mathbb{R}^m \ .$$

A linear combination of the vectors is given by

$$w = \sum_{j=1}^{n} \alpha^j v^j \ ,$$

where $\alpha^1, \alpha^2, \dots, \alpha^n$ is a set of real numbers, and each v^j is an m-vector.

Example 16.1.12 linear combination of vectors
Let us consider three vectors in $\mathbb{R}^2$, $v^1 = \begin{pmatrix} -4 & 2 \end{pmatrix}^{\mathrm{T}}$, $v^2 = \begin{pmatrix} 3 & 6 \end{pmatrix}^{\mathrm{T}}$, and $v^3 = \begin{pmatrix} 0 & 5 \end{pmatrix}^{\mathrm{T}}$. A linear combination of the vectors, with $\alpha^1 = 1$, $\alpha^2 = -2$, and $\alpha^3 = 3$, is

$$\begin{aligned} w = \sum_{j=1}^{3} \alpha^j v^j &= 1 \cdot \begin{pmatrix} -4 \\ 2 \end{pmatrix} + (-2) \cdot \begin{pmatrix} 3 \\ 6 \end{pmatrix} + 3 \cdot \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\ &= \begin{pmatrix} -4 \\ 2 \end{pmatrix} + \begin{pmatrix} -6 \\ -12 \end{pmatrix} + \begin{pmatrix} 0 \\ 15 \end{pmatrix} = \begin{pmatrix} -10 \\ 5 \end{pmatrix} . \end{aligned}$$

Another example of linear combination, with $\alpha^1 = 1$, $\alpha^2 = 0$, and $\alpha^3 = 0$, is

$$w = \sum_{j=1}^{3} \alpha^j v^j = 1 \cdot \begin{pmatrix} -4 \\ 2 \end{pmatrix} + 0 \cdot \begin{pmatrix} 3 \\ 6 \end{pmatrix} + 0 \cdot \begin{pmatrix} 0 \\ 5 \end{pmatrix} = \begin{pmatrix} -4 \\ 2 \end{pmatrix} .$$

Note that a linear combination of a set of vectors is simply a weighted sum of the vectors.

·

Linear Independence

A set of n m-vectors are linearly independent if

$$\sum_{j=1}^{n} \alpha^j v^j = 0 \quad \text{only if} \quad \alpha^1 = \alpha^2 = \cdots = \alpha^n = 0 .$$

Otherwise, the vectors are linearly dependent.

Example 16.1.13 linear independence
Let us consider four vectors,

$$w^1 = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \quad w^2 = \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}, \quad w^3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \text{and} \quad w^4 = \begin{pmatrix} 2 \\ 0 \\ 5 \end{pmatrix} .$$

The set of vectors $\{w^1, w^2, w^4\}$ is linearly dependent because

$$1 \cdot w^1 + \frac{5}{3} \cdot w^2 - 1 \cdot w^4 = 1 \cdot \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} + \frac{5}{3} \cdot \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix} - 1 \cdot \begin{pmatrix} 2 \\ 0 \\ 5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} ;$$

the linear combination with the weights $\{1, 5/3, -1\}$ produces the zero vector. Note that the choice of the weights that achieves this is not unique; we just need to find one set of weights to show that the vectors are not linearly independent (i.e., are linearly dependent).

On the other hand, the set of vectors $\{w^1, w^2, w^3\}$ is linearly independent. Considering a linear combination,

$$\alpha^1 w^1 + \alpha^2 w^2 + \alpha^3 w^3 = \alpha^1 \cdot \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} + \alpha^2 \cdot \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix} + \alpha^3 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} ,$$

we see that we must choose $\alpha^1 = 0$ to set the first component to 0, $\alpha^2 = 0$ to set the third component to 0, and $\alpha^3 = 0$ to set the second component to 0. Thus, only way to satisfy the equation is to choose the trivial set of weights, $\{0, 0, 0\}$. Thus, the set of vectors $\{w^1, w^2, w^3\}$ is linearly independent.

·

Begin Advanced Material

Vector Spaces and Bases

Given a set of n m-vectors, we can construct a vector space, V, given by

$$V = \text{span}(\{v^1, v^2, \dots, v^n\}) \ ,$$

where

$$\begin{aligned}\text{span}(\{v^1, v^2, \dots, v^n\}) &= \left\{ v \in \mathbb{R}^m : v = \sum_{k=1}^{n} \alpha^k v^k, \ \alpha^k \in \mathbb{R}^n \right\} \\ &= \text{space of vectors which are linear combinations of } v^1, v^2, \dots, v^n \ .\end{aligned}$$

In general we do not require the vectors $\{v^1, \dots, v^n\}$ to be linearly independent. When they are linearly independent, they are said to be a basis of the space. In other words, a basis of the vector space V is a set of linearly independent vectors that spans the space. As we will see shortly in our example, there are many bases for any space. However, the number of vectors in any bases for a given space is unique, and that number is called the dimension of the space. Let us demonstrate the idea in a simple example.

Example 16.1.14 Bases for a vector space in $\mathbb{R}^3$
Let us consider a vector space V spanned by vectors

$$v^1 = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \quad v^2 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \quad \text{and} \quad v^3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} .$$

By definition, any vector $x \in V$ is of the form

$$x = \alpha^1 v^1 + \alpha^2 v^2 + \alpha^3 v^3 = \alpha^1 \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} + \alpha^2 \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + \alpha^3 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha^1 + 2\alpha^2 \\ 2\alpha^1 + \alpha^2 + \alpha^3 \\ 0 \end{pmatrix} .$$

Clearly, we can express any vector of the form $x = (x_1, x_2, 0)^{\mathrm{T}}$ by choosing the coefficients α^1, α^2, and α^3 judiciously. Thus, our vector space consists of vectors of the form $(x_1, x_2, 0)^{\mathrm{T}}$, i.e., all vectors in $\mathbb{R}^3$ with zero in the third entry.

We also note that the selection of coefficients that achieves $(x_1, x_2, 0)^{\mathrm{T}}$ is not unique, as it requires solution to a system of two linear equations with three unknowns. The non-uniqueness of the coefficients is a direct consequence of $\{v^1, v^2, v^3\}$ not being linearly independent. We can easily verify the linear dependence by considering a non-trivial linear combination such as

$$2v^1 - v^2 - 3v^3 = 2 \cdot \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} - 1 \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} - 3 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} .$$

Because the vectors are not linearly independent, they do not form a basis of the space.

To choose a basis for the space, we first note that vectors in the space V are of the form $(x_1, x_2, 0)^{\mathrm{T}}$. We observe that, for example,

$$w^1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad w^2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

would span the space because any vector in V — a vector of the form $(x_1, x_2, 0)^{\mathrm{T}}$ — can be expressed as a linear combination,

$$\begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \alpha^1 w^1 + \alpha^2 w^2 = \alpha^1 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \alpha^2 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha^1 \\ \alpha^2 \\ 0 \end{pmatrix},$$

by choosing $\alpha^1 = x^1$ and $\alpha^2 = x^2$. Moreover, w^1 and w^2 are linearly independent. Thus, $\{w^1, w^2\}$ is a basis for the space V. Unlike the set $\{v^1, v^2, v^3\}$ which is not a basis, the coefficients for $\{w^1, w^2\}$ that yields $x \in V$ is unique. Because the basis consists of two vectors, the dimension of V is two. This is succinctly written as

$$\dim(V) = 2 \ .$$

Because a basis for a given space is not unique, we can pick a different set of vectors. For example,

$$z^1 = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \quad \text{and} \quad z^2 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix},$$

is also a basis for V. Since z^1 is not a constant multiple of z^2, it is clear that they are linearly independent. We need to verify that they span the space V. We can verify this by a direct argument,

$$\begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \alpha^1 z^1 + \alpha^2 z^2 = \alpha^1 \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} + \alpha^2 \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha^1 + 2\alpha^2 \\ 2\alpha^1 + \alpha^2 \\ 0 \end{pmatrix}.$$

We see that, for any $(x_1, x_2, 0)^{\mathrm{T}}$, we can find the linear combination of z^1 and z^2 by choosing the coefficients $\alpha^1 = (-x_1 + 2x_2)/3$ and $\alpha^2 = (2x_1 - x_2)/3$. Again, the coefficients that represents x using $\{z^1, z^2\}$ are unique.

For the space V, and for any given basis, we can find a unique set of two coefficients to represent any vector $x \in V$. In other words, any vector in V is uniquely described by two coefficients, or parameters. Thus, a basis provides a parameterization of the vector space V. The dimension of the space is two, because the basis has two vectors, i.e., the vectors in the space are uniquely described by two parameters.

·

While there are many bases for any space, there are certain bases that are more convenient to work with than others. Orthonormal bases — bases consisting of orthonormal sets of vectors — are such a class of bases. We recall that two set of vectors are orthogonal to each other if their inner product vanishes. In order for a set of vectors $\{v^1, \ldots, v^n\}$ to be orthogonal, the vectors must satisfy

$$(v^i)^{\mathrm{T}} v^j = 0, \quad i \neq j \ .$$

In other words, the vectors are mutually orthogonal. An orthonormal set of vectors is an orthogonal set of vectors with each vector having norm unity. That is, the set of vectors $\{v^1, \ldots, v^n\}$ is mutually orthonormal if

$$\begin{aligned} (v^i)^{\mathrm{T}} v^j &= 0, \quad i \neq j \\ \|v^i\| = (v^i)^{\mathrm{T}} v^i &= 1, \quad i = 1, \ldots, n \ . \end{aligned}$$

We note that an orthonormal set of vectors is linearly independent by construction, as we now prove.

Proof. Let $\{v^1, \ldots, v^n\}$ be an orthogonal set of (non-zero) vectors. By definition, the set of vectors is linearly independent if the only linear combination that yields the zero vector corresponds to all coefficients equal to zero, i.e.

$$\alpha^1 v^1 + \cdots + \alpha^n v^n = 0 \quad \Rightarrow \quad \alpha^1 = \cdots = \alpha^n = 0 \ .$$

To verify this indeed is the case for any orthogonal set of vectors, we perform the inner product of the linear combination with $v^1, \ldots, v^n$ to obtain

$$\begin{aligned} (v^i)^{\mathrm{T}}(\alpha^1 v^1 + \cdots + \alpha^n v^n) &= \alpha^1 (v^i)^{\mathrm{T}} v^1 + \cdots + \alpha^i (v^i)^{\mathrm{T}} v^i + \cdots + \alpha^n (v^i)^{\mathrm{T}} v^n \\ &= \alpha^i \|v^i\|^2, \quad i = 1, \ldots, n \ . \end{aligned}$$

Note that $(v^i)^{\mathrm{T}} v^j = 0$, $i \neq j$, due to orthogonality. Thus, setting the linear combination equal to zero requires

$$\alpha^i \|v^i\|^2 = 0, \quad i = 1, \ldots, n \ .$$

In other words, $\alpha^i = 0$ or $\|v^i\|^2 = 0$ for each i. If we restrict ourselves to a set of non-zero vectors, then we must have $\alpha^i = 0$. Thus, a vanishing linear combination requires $\alpha^1 = \cdots = \alpha^n = 0$, which is the definition of linear independence. $\square$

Because an orthogonal set of vectors is linearly independent by construction, an orthonormal basis for a space V is an orthonormal set of vectors that spans V. One advantage of using an orthonormal basis is that finding the coefficients for any vector in V is straightforward. Suppose, we have a basis $\{v^1, \ldots, v^n\}$ and wish to find the coefficients $\alpha^1, \ldots, \alpha^n$ that results in $x \in V$. That is, we are looking for the coefficients such that

$$x = \alpha^1 v^1 + \cdots + \alpha^i v^i + \cdots + \alpha^n v^n \ .$$

To find the i-th coefficient α^i, we simply consider the inner product with v^i, i.e.

$$\begin{aligned} (v^i)^{\mathrm{T}} x &= (v^i)^{\mathrm{T}}(\alpha^1 v^1 + \cdots + \alpha^i v^i + \cdots + \alpha^n v^n) \\ &= \alpha^1 (v^i)^{\mathrm{T}} v^1 + \cdots + \alpha^i (v^i)^{\mathrm{T}} v^i + \cdots + \alpha^n (v^i)^{\mathrm{T}} v^n \\ &= \alpha^i (v^i)^{\mathrm{T}} v^i = \alpha^i \|v^i\|^2 = \alpha^i, \quad i = 1, \ldots, n \ , \end{aligned}$$

where the last equality follows from $\|v^i\|^2 = 1$. That is, $\alpha^i = (v^i)^{\mathrm{T}} x$, $i = 1, \ldots, n$. In particular, for an orthonormal basis, we simply need to perform n inner products to find the n coefficients. This is in contrast to an arbitrary basis, which requires a solution to an $n \times n$ linear system (which is significantly more costly, as we will see later).

Example 16.1.15 Orthonormal Basis
Let us consider the space vector space V spanned by

$$v^1 = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \quad v^2 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \quad \text{and} \quad v^3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} .$$

Recalling every vector in V is of the form $(x_1, x_2, 0)^{\mathrm{T}}$, a set of vectors

$$w^1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad w^2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

forms an orthonormal basis of the space. It is trivial to verify they are orthonormal, as they are orthogonal, i.e., $(w^1)^{\mathrm{T}} w^2 = 0$, and each vector is of unit length $\|w^1\| = \|w^2\| = 1$. We also see that we can express any vector of the form $(x_1, x_2, 0)^{\mathrm{T}}$ by choosing the coefficients $\alpha^1 = x_1$ and $\alpha^2 = x_2$. Thus, $\{w^1, w^2\}$ spans the space. Because the set of vectors spans the space and is orthonormal (and hence linearly independent), it is an orthonormal basis of the space V.

Another orthonormal set of basis is formed by

$$w^1 = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \quad \text{and} \quad w^2 = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} .$$

We can easily verify that they are orthogonal and each has a unit length. The coefficients for an arbitrary vector $x = (x_1, x_2, 0)^{\mathrm{T}} \in V$ represented in the basis $\{w^1, w^2\}$ are

$$\alpha^1 = (w^1)^{\mathrm{T}} x = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{5}} (x_1 + 2x_2)$$

$$\alpha^2 = (w^2)^{\mathrm{T}} x = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{5}} (2x_1 - x_2) .$$

·

End Advanced Material

16.2 Matrix Operations

16.2.1 Interpretation of Matrices

Recall that a matrix $A \in \mathbb{R}^{m \times n}$ consists of m rows and n columns for the total of $m \cdot n$ entries,

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{pmatrix} .$$

This matrix can be interpreted in a column-centric manner as a set of n column m-vectors. Alternatively, the matrix can be interpreted in a row-centric manner as a set of m row n-vectors. Each of these interpretations is useful for understanding matrix operations, which is covered next.

16.2.2 Matrix Operations

The first matrix operation we consider is multiplication of a matrix $A \in \mathbb{R}^{m_1 \times n_1}$ by a scalar $\alpha \in \mathbb{R}$. The operation yields

$$B = \alpha A \ ,$$

where each entry of $B \in \mathbb{R}^{m_1 \times n_1}$ is given by

$$B_{ij} = \alpha A_{ij}, \quad i = 1, \ldots, m_1,\ j = 1, \ldots, n_1 \ .$$

Similar to the multiplication of a vector by a scalar, the multiplication of a matrix by a scalar scales each entry of the matrix.

The second operation we consider is addition of two matrices $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{m_2 \times n_2}$. The addition yields

$$C = A + B \ ,$$

where each entry of $C \in \mathbb{R}^{m_1 \times n_1}$ is given by

$$C_{ij} = A_{ij} + B_{ij}, \quad i = 1, \ldots, m_1,\ j = 1, \ldots, n_1 \ .$$

In order for addition of two matrices to make sense, the matrices must have the same dimensions, m_1 and n_1.

We can combine the scalar scaling and addition operation. Let $A \in \mathbb{R}^{m_1 \times n_1}$, $B \in \mathbb{R}^{m_1 \times n_1}$, and $\alpha \in \mathbb{R}$. Then, the operation

$$C = A + \alpha B$$

yields a matrix $C \in \mathbb{R}^{m_1 \times n_1}$ whose entries are given by

$$C_{ij} = A_{ij} + \alpha B_{ij}, \quad i = 1, \ldots, m_1,\ j = 1, \ldots, n_1 \ .$$

Note that the scalar-matrix multiplication and matrix-matrix addition operations treat the matrices as arrays of numbers, operating entry by entry. This is unlike the matrix-matrix prodcut, which is introduced next after an example of matrix scaling and addition.

Example 16.2.1 matrix scaling and addition
Consider the following matrices and scalar,

$$A = \begin{pmatrix} 1 & \sqrt{3} \\ -4 & 9 \\ \pi & -3 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 \\ 2 & -3 \\ \pi & -4 \end{pmatrix}, \quad \text{and} \quad \alpha = 2 \ .$$

Then,

$$C = A + \alpha B = \begin{pmatrix} 1 & \sqrt{3} \\ -4 & 9 \\ \pi & -3 \end{pmatrix} + 2 \cdot \begin{pmatrix} 0 & 2 \\ 2 & -3 \\ \pi & -4 \end{pmatrix} = \begin{pmatrix} 1 & \sqrt{3}+4 \\ 0 & 3 \\ 3\pi & -11 \end{pmatrix} .$$

Matrix-Matrix Product

Let us consider two matrices $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{m_2 \times n_2}$ with $n_1 = m_2$. The matrix-matrix product of the matrices results in

$$C = AB$$

with

$$C_{ij} = \sum_{k=1}^{n_1} A_{ik} B_{kj}, \quad i = 1, \ldots, m_1,\ j = 1, \ldots, n_2\ .$$

Because the summation applies to the second index of A and the first index of B, the number of columns of A must match the number of rows of B: $n_1 = m_2$ *must* be true. Let us consider a few examples.

Example 16.2.2 matrix-matrix product
Let us consider matrices $A \in \mathbb{R}^{3\times 2}$ and $B \in \mathbb{R}^{2\times 3}$ with

$$A = \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 2 & 3 & -5 \\ 1 & 0 & -1 \end{pmatrix}.$$

The matrix-matrix product yields

$$C = AB = \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} \begin{pmatrix} 2 & 3 & -5 \\ 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 5 & 3 & -8 \\ 1 & -12 & 11 \\ -3 & 0 & 3 \end{pmatrix},$$

where each entry is calculated as

$$\begin{aligned}
C_{11} &= \sum_{k=1}^{2} A_{1k} B_{k1} = A_{11}B_{11} + A_{12}B_{21} = 1\cdot 2 + 3 \cdot 1 = 5 \\
C_{12} &= \sum_{k=1}^{2} A_{1k} B_{k2} = A_{11}B_{12} + A_{12}B_{22} = 1\cdot 3 + 3 \cdot 0 = 3 \\
C_{13} &= \sum_{k=1}^{2} A_{1k} B_{k3} = A_{11}B_{13} + A_{12}B_{23} = 1\cdot -5 + 3 \cdot (-1) = -8 \\
C_{21} &= \sum_{k=1}^{2} A_{2k} B_{k1} = A_{21}B_{11} + A_{22}B_{21} = -4\cdot 2 + 9 \cdot 1 = 1 \\
&\vdots \\
C_{33} &= \sum_{k=1}^{2} A_{3k} B_{k3} = A_{31}B_{13} + A_{32}B_{23} = 0\cdot -5 + (-3) \cdot (-1) = 3\ .
\end{aligned}$$

Note that because $A \in \mathbb{R}^{3\times 2}$ and $B \in \mathbb{R}^{2\times 3}$, $C \in \mathbb{R}^{3\times 3}$.

This is very different from

$$D = BA = \begin{pmatrix} 2 & 3 & -5 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} = \begin{pmatrix} -10 & 48 \\ 1 & 6 \end{pmatrix},$$

where each entry is calculated as

$$D_{11} = \sum_{k=1}^{3} A_{1k}B_{k1} = B_{11}A_{11} + B_{12}A_{21} + B_{13}A_{31} = 2 \cdot 1 + 3 \cdot (-4) + (-5) \cdot 0 = -10$$

$$\vdots$$

$$D_{22} = \sum_{k=1}^{3} A_{2k}B_{k2} = B_{21}A_{12} + B_{22}A_{22} + B_{23}A_{32} = 1 \cdot 3 + 0 \cdot 9 + (-1) \cdot (-3) = 6 \ .$$

Note that because $B \in \mathbb{R}^{2\times 3}$ and $A \in \mathbb{R}^{3\times 2}$, $D \in \mathbb{R}^{2\times 2}$. Clearly, $C = AB \neq BA = D$; C and D in fact have different dimensions. Thus, matrix-matrix product is not commutative in general, even if both AB and BA make sense.

·

Example 16.2.3 inner product as matrix-matrix product
The inner product of two vectors can be considered as a special case of matrix-matrix product. Let

$$v = \begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix} \quad \text{and} \quad w = \begin{pmatrix} -2 \\ 0 \\ 4 \end{pmatrix} .$$

We have $v, w \in \mathbb{R}^3 (= \mathbb{R}^{3\times 1})$. Taking the transpose, we have $v^{\mathrm{T}} \in \mathbb{R}^{1\times 3}$. Noting that the second dimension of v^{T} and the first dimension of w match, we can perform matrix-matrix product,

$$\beta = v^{\mathrm{T}} w = \begin{pmatrix} 1 & 3 & 6 \end{pmatrix} \begin{pmatrix} -2 \\ 0 \\ 4 \end{pmatrix} = 1 \cdot (-2) + 3 \cdot 0 + 6 \cdot 4 = 22 \ .$$

·

Example 16.2.4 outer product
The outer product of two vectors is yet another special case of matrix-matrix product. The outer product B of two vectors $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$ is defined as

$$B = vw^{\mathrm{T}} \ .$$

Because $v \in \mathbb{R}^{m\times 1}$ and $w^{\mathrm{T}} \in \mathbb{R}^{1\times m}$, the matrix-matrix product vw^{T} is well-defined and yields as $m \times m$ matrix.

As in the previous example, let

$$v = \begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix} \quad \text{and} \quad w = \begin{pmatrix} -2 \\ 0 \\ 4 \end{pmatrix} .$$

The outer product of two vectors is given by

$$wv^{\mathrm{T}} = \begin{pmatrix} -2 \\ 0 \\ 4 \end{pmatrix} \begin{pmatrix} 1 & 3 & 6 \end{pmatrix} = \begin{pmatrix} -2 & -6 & -12 \\ 0 & 0 & 0 \\ 4 & 12 & 24 \end{pmatrix} .$$

Clearly, $\beta = v^{\mathrm{T}} w \neq wv^{\mathrm{T}} = B$, as they even have different dimensions.

·

In the above example, we saw that $AB \neq BA$ in general. In fact, AB might not even be allowed even if BA is allowed (consider $A \in \mathbb{R}^{2\times 1}$ and $B \in \mathbb{R}^{3\times 2}$). However, although the matrix-matrix product is not commutative in general, the matrix-matrix product *is* associative, i.e.

$$ABC = A(BC) = (AB)C \ .$$

Moreover, the matrix-matrix product is also distributive, i.e.

$$(A + B)C = AC + BC \ .$$

Proof. The associative and distributive properties of matrix-matrix product is readily proven from its definition. For associativity, we consider ij-entry of the $m_1 \times n_3$ matrix $A(BC)$, i.e.

$$\begin{aligned}(A(BC))_{ij} &= \sum_{k=1}^{n_1} A_{ik}(BC)_{kj} = \sum_{k=1}^{n_1} A_{ik} \left(\sum_{l=1}^{n_2} B_{kl} C_{lj} \right) = \sum_{k=1}^{n_1} \sum_{l=1}^{n_2} A_{ik} B_{kl} C_{lj} = \sum_{l=1}^{n_2} \sum_{k=1}^{n_1} A_{ik} B_{kl} C_{lj} \\ &= \sum_{l=1}^{n_2} \left(\sum_{k=1}^{n_1} A_{ik} B_{kl} \right) C_{lj} = \sum_{l=1}^{n_2} (AB)_{il} C_{lj} = ((AB)C)_{ij}, \quad \forall\, i, j \ .\end{aligned}$$

Since the equality $(A(BC))_{ij} = ((AB)C)_{ij}$ holds for all entries, we have $A(BC) = (AB)C$.

The distributive property can also be proven directly. The ij-entry of $(A+B)C$ can be expressed as

$$\begin{aligned}((A + B)C)_{ij} &= \sum_{k=1}^{n_1} (A + B)_{ik} C_{kj} = \sum_{k=1}^{n_1} (A_{ik} + B_{ik}) C_{kj} = \sum_{k=1}^{n_1} (A_{ik} C_{kj} + B_{ik} C_{kj}) \\ &= \sum_{k=1}^{n_1} A_{ik} C_{kj} + \sum_{k=1}^{n_1} B_{ik} C_{kj} = (AC)_{ij} + (BC)_{ij}, \quad \forall\, i, j \ .\end{aligned}$$

Again, since the equality holds for all entries, we have $(A + B)C = AC + BC$. □

Another useful rule concerning matrix-matrix product and transpose operation is

$$(AB)^{\mathrm{T}} = B^{\mathrm{T}} A^{\mathrm{T}} \ .$$

This rule is used very often.

Proof. The proof follows by checking the components of each side. The left-hand side yields

$$((AB)^{\mathrm{T}})_{ij} = (AB)_{ji} = \sum_{k=1}^{n_1} A_{jk} B_{ki} \ .$$

The right-hand side yields

$$(B^{\mathrm{T}} A^{\mathrm{T}})_{ij} = \sum_{k=1}^{n_1} (B^{\mathrm{T}})_{ik} (A^{\mathrm{T}})_{kj} = \sum_{k=1}^{n_1} B_{ki} A_{jk} = \sum_{k=1}^{n_1} A_{jk} B_{ki} \ .$$

Thus, we have

$$((AB)^{\mathrm{T}})_{ij} = (B^{\mathrm{T}}A^{\mathrm{T}})_{ij}, \quad i = 1, \dots, n_2,\ j = 1, \dots, m_1 \ .$$

□

16.2.3 Interpretations of the Matrix-Vector Product

Let us consider a special case of the matrix-matrix product: the matrix-vector product. The special case arises when the second matrix has only one column. Then, with $A \in \mathbb{R}^{m\times n}$ and $w = B \in \mathbb{R}^{n\times 1} = \mathbb{R}^n$, we have

$$C = AB \ ,$$

where

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj} = \sum_{k=1}^{n} A_{ik}w_k, \quad i = 1, \dots, m_1,\ j = 1 \ .$$

Since $C \in \mathbb{R}^{m\times 1} = \mathbb{R}^m$, we can introduce $v \in \mathbb{R}^m$ and concisely write the matrix-vector product as

$$v = Aw \ ,$$

where

$$v_i = \sum_{k=1}^{n} A_{ik}w_k, \quad i = 1, \dots, m \ .$$

Expanding the summation, we can think of the matrix-vector product as

$$\begin{aligned} v_1 &= A_{11}w_1 + A_{12}w_2 + \cdots + A_{1n}w_n \\ v_2 &= A_{21}w_1 + A_{22}w_2 + \cdots + A_{2n}w_n \\ &\vdots \\ v_m &= A_{m1}w_1 + A_{m2}w_2 + \cdots + A_{mn}w_n \ . \end{aligned}$$

Now, we consider two different interpretations of the matrix-vector product.

Row Interpretation

The first interpretation is the "row" interpretation, where we consider the matrix-vector multiplication as a series of inner products. In particular, we consider v_i as the inner product of i-th row of A and w. In other words, the vector v is computed entry by entry in the sense that

$$v_i = \begin{pmatrix} A_{i1} & A_{i2} & \cdots & A_{in} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}, \quad i = 1, \dots, m \ .$$

Example 16.2.5 row interpretation of matrix-vector product
An example of the row interpretation of matrix-vector product is

$$v = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}^{\mathrm{T}} \\ \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}^{\mathrm{T}} \\ \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}^{\mathrm{T}} \\ \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 0 \\ 1 \end{pmatrix} .$$

·

Column Interpretation

The second interpretation is the "column" interpretation, where we consider the matrix-vector multiplication as a sum of n vectors corresponding to the n columns of the matrix, i.e.

$$v = \begin{pmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{m1} \end{pmatrix} w_1 + \begin{pmatrix} A_{12} \\ A_{22} \\ \vdots \\ A_{m2} \end{pmatrix} w_2 + \cdots + \begin{pmatrix} A_{1n} \\ A_{2n} \\ \vdots \\ A_{mn} \end{pmatrix} w_n \ .$$

In this case, we consider v as a linear combination of columns of A with coefficients w. Hence $v = Aw$ is simply another way to write a linear combination of vectors: the columns of A are the vectors, and w contains the coefficients.

Example 16.2.6 column interpretation of matrix-vector product
An example of the column interpretation of matrix-vector product is

$$v = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} = 3 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 1 \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 0 \\ 1 \end{pmatrix} .$$

Clearly, the outcome of the matrix-vector product is identical to that computed using the row interpretation.

·

Left Vector-Matrix Product

We now consider another special case of the matrix-matrix product: the left vector-matrix product. This special case arises when the first matrix only has one row. Then, we have $A \in \mathbb{R}^{1\times m}$ and $B \in \mathbb{R}^{m\times n}$. Let us denote the matrix A, which is a row vector, by w^{T}. Clearly, $w \in \mathbb{R}^m$, because $w^{\mathrm{T}} \in \mathbb{R}^{1\times m}$. The left vector-matrix product yields

$$v = w^{\mathrm{T}} B \ ,$$

where

$$v_j = \sum_{k=1}^{m} w_k B_{kj}, \quad j = 1, \ldots, n \ .$$

The resultant vector v is a row vector in $\mathbb{R}^{1\times n}$. The left vector-matrix product can also be interpreted in two different manners. The first interpretation considers the product as a series of dot products, where each entry v_j is computed as a dot product of w with the j-th column of B, i.e.

$$v_j = \begin{pmatrix} w_1 & w_2 & \cdots & w_m \end{pmatrix} \begin{pmatrix} B_{1j} \\ B_{2j} \\ \vdots \\ B_{mj} \end{pmatrix}, \quad j = 1, \dots, n \ .$$

The second interpretation considers the left vector-matrix product as a linear combination of rows of B, i.e.

$$v = w_1 \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1n} \end{pmatrix} + w_2 \begin{pmatrix} B_{21} & B_{22} & \cdots & B_{2n} \end{pmatrix}$$
$$+ \cdots + w_m \begin{pmatrix} B_{m1} & B_{m2} & \cdots & B_{mn} \end{pmatrix}.$$

16.2.4 Interpretations of the Matrix-Matrix Product

Similar to the matrix-vector product, the matrix-matrix product can be interpreted in a few different ways. Throughout the discussion, we assume $A \in \mathbb{R}^{m_1\times n_1}$ and $B \in \mathbb{R}^{n_1\times n_2}$ and hence $C = AB \in \mathbb{R}^{m_1\times n_2}$.

Matrix-Matrix Product as a Series of Matrix-Vector Products

One interpretation of the matrix-matrix product is to consider it as computing C one column at a time, where the j-th column of C results from the matrix-vector product of the matrix A with the j-th column of B, i.e.

$$C_{\cdot j} = AB_{\cdot j}, \quad j = 1, \dots, n_2 \ ,$$

where $C_{\cdot j}$ refers to the j-th column of C. In other words,

$$\begin{pmatrix} C_{1j} \\ C_{2j} \\ \vdots \\ C_{m_1 j} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n_1} \\ A_{21} & A_{22} & \cdots & A_{2n_1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m_1 1} & A_{m_1 2} & \cdots & A_{m_1 n_1} \end{pmatrix} \begin{pmatrix} B_{1j} \\ B_{2j} \\ \vdots \\ B_{n_1 j} \end{pmatrix}, \quad j = 1, \dots, n_2 \ .$$

Example 16.2.7 matrix-matrix product as a series of matrix-vector products
Let us consider matrices $A \in \mathbb{R}^{3\times 2}$ and $B \in \mathbb{R}^{2\times 3}$ with

$$A = \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 2 & 3 & -5 \\ 1 & 0 & -1 \end{pmatrix}.$$

The first column of $C = AB \in \mathbb{R}^{3\times 3}$ is given by

$$C_{\cdot 1} = AB_{\cdot 1} = \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -3 \end{pmatrix}.$$

Similarly, the second and third columns are given by

$$C_{\cdot 2} = AB_{\cdot 2} = \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ -12 \\ 0 \end{pmatrix}$$

and

$$C_{\cdot 3} = AB_{\cdot 3} = \begin{pmatrix} 1 & 3 \\ -4 & 9 \\ 0 & -3 \end{pmatrix} \begin{pmatrix} -5 \\ -1 \end{pmatrix} = \begin{pmatrix} -8 \\ 11 \\ 3 \end{pmatrix} .$$

Putting the columns of C together

$$C = \begin{pmatrix} C_{\cdot 1} & C_{\cdot 2} & C_{\cdot 3} \end{pmatrix} = \begin{pmatrix} 5 & 3 & -8 \\ 1 & -12 & 11 \\ -3 & 0 & 3 \end{pmatrix} .$$

·

Matrix-Matrix Product as a Series of Left Vector-Matrix Products

In the previous interpretation, we performed the matrix-matrix product by constructing the resultant matrix one column at a time. We can also use a series of left vector-matrix products to construct the resultant matrix one row at a time. Namely, in $C = AB$, the i-th row of C results from the left vector-matrix product of i-th row of A with the matrix B, i.e.

$$C_{i\cdot} = A_{i\cdot}B, \quad i = 1, \dots, m_1 ,$$

where $C_{i\cdot}$ refers to the i-th row of C. In other words,

$$\begin{pmatrix} C_{i1} & \cdots & C_{in_1} \end{pmatrix} = \begin{pmatrix} A_{i1} & \cdots & A_{in_1} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1n_2} \\ \vdots & \ddots & \vdots \\ B_{m_2 1} & \cdots & B_{m_2 n_2} \end{pmatrix}, \quad i = 1, \dots, m_1 .$$

16.2.5 Operation Count of Matrix-Matrix Product

Matrix-matrix product is ubiquitous in scientific computing, and significant effort has been put into efficient performance of the operation on modern computers. Let us now count the number of additions and multiplications required to compute this product. Consider multiplication of $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{n_1 \times n_2}$. To compute $C = AB$, we perform

$$C_{ij} = \sum_{k=1}^{n_1} A_{ik}B_{kj}, \quad i = 1, \dots, m_1,\ j = 1, \dots, n_2 .$$

Computing each C_{ij} requires n_1 multiplications and n_1 additions, yielding the total of $2n_1$ operations. We must perform this for $m_1 n_2$ entries in C. Thus, the total operation count for computing C is $2m_1 n_1 n_2$. Considering the matrix-vector product and the inner product as special cases of matrix-matrix product, we can summarize how the operation count scales.

Operation	Sizes	Operation count
Matrix-matrix	$m_1 = n_1 = m_2 = n_2 = n$	$2n^3$
Matrix-vector	$m_1 = n_1 = m_2 = n$, $n_2 = 1$	$2n^2$
Inner product	$n_1 = m_1 = n$, $m_1 = n_2 = 1$	$2n$

The operation count is measured in FLoating Point Operations, or FLOPs. (Note FLOPS is different from FLOPs: FLOPS refers to FLoating Point Operations per Second, which is a "speed" associated with a particular computer/hardware and a particular implementation of an algorithm.)

16.2.6 The Inverse of a Matrix (Briefly)

We have now studied the matrix vector product, in which, given a vector $x \in \mathbb{R}^n$, we calculate a new vector $b = Ax$, where $A \in \mathbb{R}^{n\times n}$ and hence $b \in \mathbb{R}^n$. We may think of this as a "forward" problem, in which given x we calculate $b = Ax$. We can now also ask about the corresponding "inverse" problem: given b, can we find x such that $Ax = b$? Note in this section, and for reasons which shall become clear shortly, we shall exclusively consider square matrices, and hence we set $m = n$.

To begin, let us revert to the scalar case. If b is a scalar and a is a non-zero scalar, we know that the (very simple linear) equation $ax = b$ has the solution $x = b/a$. We may write this more suggestively as $x = a^{-1}b$ since of course $a^{-1} = 1/a$. It is important to note that the equation $ax = b$ has a solution only if a is non-zero; if a is zero, then of course there is no x such that $ax = b$. (This is not quite true: in fact, if $b = 0$ and $a = 0$ then $ax = b$ has an infinity of solutions — any value of x. We discuss this "singular but solvable" case in more detail in Unit V.)

We can now proceed to the matrix case "by analogy." The matrix equation $Ax = b$ can of course be viewed as a system of linear equations in n unknowns. The first equation states that the inner product of the first row of A with x must equal b_1; in general, the i^{th} equation states that the inner product of the i^{th} row of A with x must equal b_i. Then if A is non-zero we could plausibly expect that $x = A^{-1}b$. This statement is clearly deficient in two related ways: what we do mean when we say a matrix is non-zero? and what do we in fact mean by A^{-1}.

As regards the first question, $Ax = b$ will have a solution when A is non-singular: non-singular is the proper extension of the scalar concept of "non-zero" in this linear systems context. Conversely, if A is singular then (except for special b) $Ax = b$ will have no solution: singular is the proper extension of the scalar concept of "zero" in this linear systems context. How can we determine if a matrix A is singular? Unfortunately, it is not nearly as simple as verifying, say, that the matrix consists of at least one non-zero entry, or contains all non-zero entries.

There are variety of ways to determine whether a matrix is non-singular, many of which may only make good sense in later chapters (in particular, in Unit V): a non-singular $n \times n$ matrix A has n independent columns (or, equivalently, n independent rows); a non-singular $n \times n$ matrix A has all non-zero eigenvalues; a non-singular matrix A has a non-zero determinant (perhaps this condition is closest to the scalar case, but it is also perhaps the least useful); a non-singular matrix A has all non-zero pivots in a (partially pivoted) "LU" decomposition process (described in Unit V). For now, we shall simply assume that A is non-singular. (We should also emphasize that in the numerical context we must be concerned not only with matrices which might be singular but also with matrices which are "almost" singular in some appropriate sense.) As regards the second question, we must first introduce the *identity* matrix, I.

Let us now define an identity matrix. The identity matrix is a $m \times m$ square matrix with ones on the diagonal and zeros elsewhere, i.e.

$$I_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad .$$

Identity matrices in $\mathbb{R}^1$, $\mathbb{R}^2$, and $\mathbb{R}^3$ are

$$I = \begin{pmatrix} 1 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \text{and} \quad I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The identity matrix is conventionally denoted by I. If $v \in \mathbb{R}^m$, the i-th entry of Iv is

$$\begin{aligned}(Iv)_i &= \sum_{k=1}^{m} I_{ik} v_k \\ &= \cancelto{0}{I_{i1}} v_1 + \cdots + \cancelto{0}{I_{i,i-1}} v_{i-1} + I_{ii} v_i + \cancelto{0}{I_{i,i+1}} v_{i+1} + \cdots + \cancelto{0}{I_{im}} v_m \\ &= v_i, \quad i = 1, \ldots, m\ .\end{aligned}$$

So, we have $Iv = v$. Following the same argument, we also have $v^{\mathrm{T}} I = v^{\mathrm{T}}$. In essence, I is the m-dimensional version of "one."

We may then define A^{-1} as that (unique) matrix such that $A^{-1}A = I$. (Of course in the scalar case, this defines a^{-1} as the unique scalar such that $a^{-1}a = 1$ and hence $a^{-1} = 1/a$.) In fact, $A^{-1}A = I$ and also $AA^{-1} = I$ and thus this is a case in which matrix multiplication does indeed commute. We can now "derive" the result $x = A^{-1}b$: we begin with $Ax = b$ and multiply both sides by A^{-1} to obtain $A^{-1}Ax = A^{-1}b$ or, since the matrix product is associative, $x = A^{-1}b$. Of course this definition of A^{-1} does not yet tell us how to find A^{-1}: we shall shortly consider this question from a pragmatic Matlab perspective and then in Unit V from a more fundamental numerical linear algebra perspective. We should note here, however, that the matrix inverse is very rarely computed or used in practice, for reasons we will understand in Unit V. Nevertheless, the inverse can be quite useful for very small systems (n small) and of course more generally as an central concept in the consideration of linear systems of equations.

Example 16.2.8 The inverse of a 2×2 matrix
We consider here the case of a 2×2 matrix A which we write as

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\ .$$

If the columns are to be independent we must have $a/b \neq c/d$ or $(ad)/(bc) \neq 1$ or $ad - bc \neq 0$ which in fact is the condition that the determinant of A is nonzero. The inverse of A is then given by

$$A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}\ .$$

Note that this inverse is only defined if $ad - bc \neq 0$, and we thus see the necessity that A is nonsingular. It is a simple matter to show by explicit matrix multiplication that $A^{-1}A = AA^{-1} = I$, as desired.

·

16.3 Special Matrices

Let us now introduce a few special matrices that we shall encounter frequently in numerical methods.

16.3.1 Diagonal Matrices

A square matrix A is said to be diagonal if the off-diagonal entries are zero, i.e.

$$A_{ij} = 0, \quad i \neq j \ .$$

Example 16.3.1 diagonal matrices
Examples of diagonal matrix are

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 7 \end{pmatrix}, \quad \text{and} \quad C = \begin{pmatrix} 4 \end{pmatrix} .$$

The identity matrix is a special case of a diagonal matrix with all the entries in the diagonal equal to 1. Any 1×1 matrix is trivially diagonal as it does not have any off-diagonal entries.

·

16.3.2 Symmetric Matrices

A square matrix A is said to be symmetric if the off-diagonal entries are symmetric about the diagonal, i.e.

$$A_{ij} = A_{ji}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, m \ .$$

The equivalent statement is that A is not changed by the transpose operation, i.e.

$$A^{\mathrm{T}} = A \ .$$

We note that the identity matrix is a special case of symmetric matrix. Let us look at a few more examples.

Example 16.3.2 Symmetric matrices
Examples of symmetric matrices are

$$A = \begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & \pi & 3 \\ \pi & 1 & -1 \\ 3 & -1 & 7 \end{pmatrix}, \quad \text{and} \quad C = \begin{pmatrix} 4 \end{pmatrix} .$$

Note that any scalar, or a 1×1 matrix, is trivially symmetric and unchanged under transpose.

·

16.3.3 Symmetric Positive Definite Matrices

A $m \times m$ square matrix A is said to be symmetric positive definite (SPD) if it is symmetric and furthermore satisfies

$$v^{\mathrm{T}} A v > 0, \quad \forall\, v \in \mathbb{R}^m \ (v \neq 0) \ .$$

Before we discuss its properties, let us give an example of a SPD matrix.

Example 16.3.3 Symmetric positive definite matrices
An example of a symmetric positive definite matrix is

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} .$$

We can confirm that A is symmetric by inspection. To check if A is positive definite, let us consider the quadratic form

$$\begin{aligned} q(v) \equiv v^{\mathrm{T}} A v = \sum_{i=1}^{2} v_i \left(\sum_{j=1}^{2} A_{ij} v_j \right) &= \sum_{i=1}^{2} \sum_{j=1}^{2} A_{ij} v_i v_j \\ &= A_{11} v_1^2 + A_{12} v_1 v_2 + A_{21} v_2 v_1 + A_{22} v_2^2 \\ &= A_{11} v_1^2 + 2A_{12} v_1 v_2 + A_{22} v_2^2 \ , \end{aligned}$$

where the last equality follows from the symmetry condition $A_{12} = A_{21}$. Substituting the entries of A,

$$q(v) = v^{\mathrm{T}} A v = 2v_1^2 - 2v_1 v_2 + 2v_2^2 = 2 \left[\left(v_1 - \frac{1}{2} v_2 \right)^2 - \frac{1}{4} v_2^2 + v_2^2 \right] = 2 \left[\left(v_1 - \frac{1}{2} v_2 \right)^2 + \frac{3}{4} v_2^2 \right] .$$

Because $q(v)$ is a sum of two positive terms (each squared), it is non-negative. It is equal to zero only if

$$v_1 - \frac{1}{2} v_2 = 0 \quad \text{and} \quad \frac{3}{4} v_2^2 = 0 \ .$$

The second condition requires $v_2 = 0$, and the first condition with $v_2 = 0$ requires $v_1 = 0$. Thus, we have

$$q(v) = v^{\mathrm{T}} A v > 0, \quad \forall \, v \in \mathbb{R}^2 \ ,$$

and $v^{\mathrm{T}} A v = 0$ if $v = 0$. Thus A is symmetric positive definite.

Symmetric positive definite matrices are encountered in many areas of engineering and science. They arise naturally in the numerical solution of, for example, the heat equation, the wave equation, and the linear elasticity equations. One important property of symmetric positive definite matrices is that they are always invertible: A^{-1} always exists. Thus, if A is an SPD matrix, then, for any b, there is always a unique x such that

$$Ax = b \ .$$

In a later unit, we will discuss techniques for solution of linear systems, such as the one above. For now, we just note that there are particularly efficient techniques for solving the system when the matrix is symmetric positive definite.

16.3.4 Triangular Matrices

Triangular matrices are square matrices whose entries are all zeros either below or above the diagonal. A $m \times m$ square matrix is said to be upper triangular if all entries below the diagonal are zero, i.e.

$$A_{ij} = 0, \quad i > j \ .$$

A square matrix is said to be lower triangular if all entries above the diagonal are zero, i.e.

$$A_{ij} = 0, \quad j > i \ .$$

We will see later that a linear system, $Ax = b$, in which A is a triangular matrix is particularly easy to solve. Furthermore, the linear system is guaranteed to have a unique solution as long as all diagonal entries are nonzero.

Example 16.3.4 triangular matrices
Examples of upper triangular matrices are

$$A = \begin{pmatrix} 1 & -2 \\ 0 & 3 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 4 & 1 \\ 0 & 0 & -3 \end{pmatrix} .$$

Examples of lower triangular matrices are

$$C = \begin{pmatrix} 1 & 0 \\ -7 & 6 \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} 2 & 0 & 0 \\ 7 & -5 & 0 \\ 3 & 1 & 4 \end{pmatrix} .$$

·

Begin Advanced Material

16.3.5 Orthogonal Matrices

A $m \times m$ square matrix Q is said to be orthogonal if its columns form an orthonormal set. That is, if we denote the j-th column of Q by q_j, we have

$$Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_m \end{pmatrix},$$

where

$$q_i^{\mathrm{T}} q_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} .$$

Orthogonal matrices have a special property

$$Q^{\mathrm{T}} Q = I \ .$$

This relationship follows directly from the fact that columns of Q form an orthonormal set. Recall that the ij entry of $Q^{\mathrm{T}}Q$ is the inner product of the i-th row of Q^{T} (which is the i-th column of Q) and the j-th column of Q. Thus,

$$(Q^{\mathrm{T}}Q)_{ij} = q_i^{\mathrm{T}} q_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} ,$$

which is the definition of the identity matrix. Orthogonal matrices also satisfy

$$QQ^{\mathrm{T}} = I \ ,$$

which in fact is a minor miracle.

Example 16.3.5 Orthogonal matrices
Examples of orthogonal matrices are

$$Q = \begin{pmatrix} 2/\sqrt{5} & -1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{pmatrix} \quad \text{and} \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

We can easily verify that the columns the matrix Q are orthogonal to each other and each are of unit length. Thus, Q is an orthogonal matrix. We can also directly confirm that $Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$. Similarly, the identity matrix is trivially orthogonal.

·

Let us discuss a few important properties of orthogonal matrices. First, the action by an orthogonal matrix preserves the 2-norm of a vector, i.e.

$$\|Qx\|_2 = \|x\|_2, \quad \forall\, x \in \mathbb{R}^m \ .$$

This follows directly from the definition of 2-norm and the fact that $Q^{\mathrm{T}}Q = I$, i.e.

$$\|Qx\|_2^2 = (Qx)^{\mathrm{T}}(Qx) = x^{\mathrm{T}}Q^{\mathrm{T}}Qx = x^{\mathrm{T}}Ix = x^{\mathrm{T}}x = \|x\|_2^2 \ .$$

Second, orthogonal matrices are always invertible. In fact, solving a linear system defined by an orthogonal matrix is trivial because

$$Qx = b \quad \Rightarrow \quad Q^{\mathrm{T}}Qx = Q^{\mathrm{T}}b \quad \Rightarrow \quad x = Q^{\mathrm{T}}b \ .$$

In considering linear spaces, we observed that a basis provides a unique description of vectors in V in terms of the coefficients. As columns of Q form an orthonormal set of m m-vectors, it can be thought of as an basis of $\mathbb{R}^m$. In solving $Qx = b$, we are finding the representation of b in coefficients of $\{q_1, \ldots, q_m\}$. Thus, the operation by Q^{T} (or Q) represent a simple coordinate transformation. Let us solidify this idea by showing that a rotation matrix in $\mathbb{R}^2$ is an orthogonal matrix.

Example 16.3.6 Rotation matrix
Rotation of a vector is equivalent to representing the vector in a rotated coordinate system. A rotation matrix that rotates a vector in $\mathbb{R}^2$ by angle θ is

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} .$$

Let us verify that the rotation matrix is orthogonal for any θ. The two columns are orthogonal because

$$r_1^{\mathrm{T}} r_2 = \begin{pmatrix} \cos(\theta) & \sin(\theta) \end{pmatrix} \begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \end{pmatrix} = -\cos(\theta)\sin(\theta) + \sin(\theta)\cos(\theta) = 0, \quad \forall\, \theta \ .$$

Each column is of unit length because

$$\begin{aligned} \|r_1\|_2^2 &= (\cos(\theta))^2 + (\sin(\theta))^2 = 1 \\ \|r_2\|_2^2 &= (-\sin(\theta))^2 + (\cos(\theta))^2 = 1, \quad \forall\, \theta \ . \end{aligned}$$

Thus, the columns of the rotation matrix is orthonormal, and the matrix is orthogonal. This result verifies that the action of the orthogonal matrix represents a coordinate transformation in $\mathbb{R}^2$. The interpretation of an orthogonal matrix as a coordinate transformation readily extends to higher-dimensional spaces.

·

16.3.6 Orthonormal Matrices

Let us define orthonormal matrices to be $m \times n$ matrices whose columns form an orthonormal set, i.e.

$$Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_n \end{pmatrix} ,$$

with

$$q_i^{\mathrm{T}} q_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \ . \end{cases}$$

Note that, unlike an orthogonal matrix, we do not require the matrix to be square. Just like orthogonal matrices, we have

$$Q^{\mathrm{T}} Q = I \ ,$$

where I is an $n \times n$ matrix. The proof is identical to that for the orthogonal matrix. However, QQ^{T} does not yield an identity matrix,

$$QQ^{\mathrm{T}} \neq I \ ,$$

unless of course $m = n$.

Example 16.3.7 orthonormal matrices
An example of an orthonormal matrix is

$$Q = \begin{pmatrix} 1/\sqrt{6} & -2/\sqrt{5} \\ 2/\sqrt{6} & 1/\sqrt{5} \\ 1/\sqrt{6} & 0 \end{pmatrix} .$$

We can verify that $Q^{\mathrm{T}} Q = I$ because

$$Q^{\mathrm{T}} Q = \begin{pmatrix} 1/\sqrt{6} & 2/\sqrt{6} & 1/\sqrt{6} \\ -2/\sqrt{5} & 1/\sqrt{5} & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{6} & -2/\sqrt{5} \\ 2/\sqrt{6} & 1/\sqrt{5} \\ 1/\sqrt{6} & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

However, $QQ^{\mathrm{T}} \neq I$ because

$$QQ^{\mathrm{T}} = \begin{pmatrix} 1/\sqrt{6} & -2/\sqrt{5} \\ 2/\sqrt{6} & 1/\sqrt{5} \\ 1/\sqrt{6} & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{6} & 2/\sqrt{6} & 1/\sqrt{6} \\ -2/\sqrt{5} & 1/\sqrt{5} & 0 \end{pmatrix} = \begin{pmatrix} 29/30 & -1/15 & 1/6 \\ -1/15 & 13/15 & 1/3 \\ 1/6 & 1/3 & 1/6 \end{pmatrix} .$$

·

End Advanced Material

Begin Advanced Material

16.4 Further Concepts in Linear Algebra

16.4.1 Column Space and Null Space

Let us introduce more concepts in linear algebra. First is the column space. The column space of matrix A is a space of vectors that can be expressed as Ax. From the column interpretation of matrix-vector product, we recall that Ax is a linear combination of the columns of A with the weights provided by x. We will denote the column space of $A \in \mathbb{R}^{m\times n}$ by $\text{col}(A)$, and the space is defined as

$$\text{col}(A) = \{v \in \mathbb{R}^m : v = Ax \text{ for some } x \in \mathbb{R}^n\} \ .$$

The column space of A is also called the image of A, $\text{img}(A)$, or the range of A, $\text{range}(A)$.

The second concept is the null space. The null space of $A \in \mathbb{R}^{m\times n}$ is denoted by $\text{null}(A)$ and is defined as

$$\text{null}(A) = \{x \in \mathbb{R}^n : Ax = 0\} \ ,$$

i.e., the null space of A is a space of vectors that results in $Ax = 0$. Recalling the column interpretation of matrix-vector product and the definition of linear independence, we note that the columns of A must be linearly dependent in order for A to have a non-trivial null space. The null space defined above is more formally known as the right null space and also called the kernel of A, $\ker(A)$.

Example 16.4.1 column space and null space
Let us consider a 3×2 matrix

$$A = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} .$$

The column space of A is the set of vectors representable as Ax, which are

$$Ax = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \cdot x_1 + \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} \cdot x_2 = \begin{pmatrix} 2x_2 \\ x_1 \\ 0 \end{pmatrix} .$$

So, the column space of A is a set of vectors with arbitrary values in the first two entries and zero in the third entry. That is, $\text{col}(A)$ is the 1-2 plane in $\mathbb{R}^3$.

Because the columns of A are linearly independent, the only way to realize $Ax = 0$ is if x is the zero vector. Thus, the null space of A consists of the zero vector only.

Let us now consider a 2×3 matrix

$$B = \begin{pmatrix} 1 & 2 & 0 \\ 2 & -1 & 3 \end{pmatrix} .$$

The column space of B consists of vectors of the form

$$Bx = \begin{pmatrix} 1 & 2 & 0 \\ 2 & -1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 + 2x_2 \\ 2x_1 - x_2 + 3x_3 \end{pmatrix} .$$

By judiciously choosing x_1, x_2, and x_3, we can express any vectors in $\mathbb{R}^2$. Thus, the column space of B is entire $\mathbb{R}^2$, i.e., $\mathrm{col}(B) = \mathbb{R}^2$.

Because the columns of B are not linearly independent, we expect B to have a nontrivial null space. Invoking the row interpretation of matrix-vector product, a vector x in the null space must satisfy

$$x_1 + 2x_2 = 0 \quad \text{and} \quad 2x_1 - x_2 + 3x_3 = 0 \ .$$

The first equation requires $x_1 = -2x_2$. The combination of the first requirement and the second equation yields $x_3 = \frac{5}{3}x_2$. Thus, the null space of B is

$$\mathrm{null}(B) = \left\{ \alpha \cdot \begin{pmatrix} -2 \\ 1 \\ 5/3 \end{pmatrix} : \alpha \in \mathbb{R} \right\} .$$

Thus, the null space is a one-dimensional space (i.e., a line) in $\mathbb{R}^3$.

·

16.4.2 Projectors

Another important concept — in particular for least squares covered in Chapter 17 — is the concept of projector. A projector is a square matrix P that is idempotent, i.e.

$$P^2 = PP = P \ .$$

Let v be an arbitrary vector in $\mathbb{R}^m$. The projector P projects v, which is not necessary in $\mathrm{col}(P)$, onto $\mathrm{col}(P)$, i.e.

$$w = Pv \in \mathrm{col}(P), \quad \forall\, v \in \mathbb{R}^m \ .$$

In addition, the projector P does not modify a vector that is already in $\mathrm{col}(P)$. This is easily verified because

$$Pw = PPv = Pv = w, \quad \forall\, w \in \mathrm{col}(P) \ .$$

Intuitively, a projector projects a vector $v \in \mathbb{R}^m$ onto a smaller space $\mathrm{col}(P)$. If the vector is already in $\mathrm{col}(P)$, then it would be left unchanged.

The complementary projector of P is a projector $I - P$. It is easy to verify that $I - P$ is a projector itself because

$$(I-P)^2 = (I-P)(I-P) = I - 2P + PP = I - P \ .$$

It can be shown that the complementary projector $I - P$ projects onto the null space of P, $\mathrm{null}(P)$.

When the space along which the projector projects is orthogonal to the space onto which the projector projects, the projector is said to be an orthogonal projector. Algebraically, orthogonal projectors are symmetric.

When an orthonormal basis for a space is available, it is particularly simple to construct an orthogonal projector onto the space. Say $\{q_1, \dots, q_n\}$ is an orthonormal basis for a n-dimensional subspace of $\mathbb{R}^m$, $n < m$. Given any $v \in \mathbb{R}^m$, we recall that

$$u_i = q_i^{\mathrm{T}} v$$

is the component of v in the direction of q_i represented in the basis $\{q_i\}$. We then introduce the vector

$$w_i = q_i(q_i^{\mathrm{T}} v) \ ;$$

the sum of such vectors would produce the projection of $v \in \mathbb{R}^m$ onto V spanned by $\{q_i\}$. More compactly, if we form an $m \times n$ matrix

$$Q = \begin{pmatrix} q_1 & \cdots & q_n \end{pmatrix},$$

then the projection of v onto the column space of Q is

$$w = Q(Q^{\mathrm{T}} v) = (QQ^{\mathrm{T}})v \ .$$

We recognize that the orthogonal projector onto the span of $\{q_i\}$ or $\mathrm{col}(Q)$ is

$$P = QQ^{\mathrm{T}} \ .$$

Of course P is symmetric, $(QQ^{\mathrm{T}})^{\mathrm{T}} = (Q^{\mathrm{T}})^{\mathrm{T}} Q^{\mathrm{T}} = QQ^{\mathrm{T}}$, and idempotent, $(QQ^{\mathrm{T}})(QQ^{\mathrm{T}}) = Q(Q^{\mathrm{T}}Q)Q^{\mathrm{T}} = QQ^{\mathrm{T}}$.

End Advanced Material

Chapter 17

Least Squares

17.1 Data Fitting in Absence of Noise and Bias

We motivate our discussion by reconsidering the friction coefficient example of Chapter 15. We recall that, according to Amontons, the static friction, $F_{\text{f, static}}$, and the applied normal force, $F_{\text{normal, applied}}$, are related by

$$F_{\text{f, static}} \le \mu_{\text{s}}\, F_{\text{normal, applied}}\ ;$$

here μ_{s} is the coefficient of friction, which is only dependent on the two materials in contact. In particular, the *maximum* static friction is a linear function of the applied normal force, i.e.

$$F^{\max}_{\text{f, static}} = \mu_{\text{s}}\, F_{\text{normal, applied}}\ .$$

We wish to deduce μ_{s} by measuring the maximum static friction attainable for several different values of the applied normal force.

Our approach to this problem is to first choose the form of a model based on physical principles and then deduce the parameters based on a set of measurements. In particular, let us consider a simple affine model

$$y = Y_{\text{model}}(x;\beta) = \beta_0 + \beta_1 x\ .$$

The variable y is the predicted quantity, or the output, which is the maximum static friction $F^{\max}_{\text{f, static}}$. The variable x is the independent variable, or the input, which is the maximum normal force $F_{\text{normal, applied}}$. The function Y_{model} is our predictive model which is parameterized by a parameter $\beta = (\beta_0, \beta_1)$. Note that Amontons' law is a particular case of our general affine model with $\beta_0 = 0$ and $\beta_1 = \mu_{\text{s}}$. If we take m *noise-free* measurements and Amontons' law is exact, then we expect

$$F^{\max}_{\text{f, static}\ i} = \mu_{\text{s}}\, F_{\text{normal, applied}\ i}, \quad i = 1,\ldots,m\ .$$

The equation should be satisfied exactly for each one of the m measurements. Accordingly, there is also a unique solution to our model-parameter identification problem

$$y_i = \beta_0 + \beta_1 x_i, \quad i = 1,\ldots,m\ ,$$

with the solution given by $\beta_0^{\text{true}} = 0$ and $\beta_1^{\text{true}} = \mu_{\text{s}}$.

Because the dependency of the output y on the model parameters $\{\beta_0, \beta_1\}$ is linear, we can write the system of equations as a $m \times 2$ matrix equation

$$\underbrace{\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}}_{\beta} = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}}_{Y} ,$$

or, more compactly,

$$X\beta = Y \ .$$

Using the row interpretation of matrix-vector multiplication, we immediately recover the original set of equations,

$$X\beta = \begin{pmatrix} \beta_0 + \beta_1 x_1 \\ \beta_0 + \beta_1 x_2 \\ \vdots \\ \beta_0 + \beta_1 x_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = Y \ .$$

Or, using the column interpretation, we see that our parameter fitting problem corresponds to choosing the two weights for the two m-vectors to match the right-hand side,

$$X\beta = \beta_0 \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \beta_1 \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = Y \ .$$

We emphasize that the linear system $X\beta = Y$ is overdetermined, i.e., more equations than unknowns ($m > n$). (We study this in more detail in the next section.) However, we can still find a solution to the system because the following two conditions are satisfied:

Unbiased: Our model includes the true functional dependence $y = \mu_{\text{s}} x$, and thus the model is capable of representing this true underlying functional dependence. This would not be the case if, for example, we consider a constant model $y(x) = \beta_0$ because our model would be incapable of representing the linear dependence of the friction force on the normal force. Clearly the assumption of no bias is a very strong assumption given the complexity of the physical world.

Noise free: We have perfect measurements: each measurement y_i corresponding to the independent variable x_i provides the "exact" value of the friction force. Obviously this is again rather naïve and will need to be relaxed.

Under these assumptions, there exists a parameter β^{true} that completely describe the measurements, i.e.

$$y_i = Y_{\text{model}}(x; \beta^{\text{true}}), \quad i = 1, \ldots, m \ .$$

(The β^{true} will be unique if the columns of X are independent.) Consequently, our predictive model is perfect, and we can exactly predict the experimental output for any choice of x, i.e.

$$Y(x) = Y_{\text{model}}(x; \beta^{\text{true}}), \quad \forall\, x \ ,$$

where $Y(x)$ is the experimental measurement corresponding to the condition described by x. However, in practice, the bias-free and noise-free assumptions are rarely satisfied, and our model is never a perfect predictor of the reality.

In Chapter 19, we will develop a probabilistic tool for quantifying the effect of noise and bias; the current chapter focuses on developing a least-squares technique for solving overdetermined linear system (in the deterministic context) which is essential to solving these data fitting problems. In particular we will consider a strategy for solving overdetermined linear systems of the form

$$Bz = g \ ,$$

where $B \in \mathbb{R}^{m\times n}$, $z \in \mathbb{R}^n$, and $g \in \mathbb{R}^m$ with $m > n$.

Before we discuss the least-squares strategy, let us consider another example of overdetermined systems in the context of polynomial fitting. Let us consider a particle experiencing constant acceleration, e.g. due to gravity. We know that the position y of the particle at time t is described by a quadratic function

$$y(t) = \frac{1}{2}at^2 + v_0 t + y_0 \ ,$$

where a is the acceleration, v_0 is the initial velocity, and y_0 is the initial position. Suppose that we do not know the three parameters a, v_0, and y_0 that govern the motion of the particle and we are interested in determining the parameters. We could do this by first measuring the position of the particle at several different times and recording the pairs $\{t_i, y(t_i)\}$. Then, we could fit our measurements to the quadratic model to deduce the parameters.

The problem of finding the parameters that govern the motion of the particle is a special case of a more general problem: polynomial fitting. Let us consider a quadratic polynomial, i.e.

$$y(x) = \beta_0^{\text{true}} + \beta_1^{\text{true}} x + \beta_2^{\text{true}} x^2 \ ,$$

where $\beta^{\text{true}} = \{\beta_0^{\text{true}}, \beta_1^{\text{true}}, \beta_2^{\text{true}}\}$ is the set of *true* parameters characterizing the modeled phenomenon. Suppose that we do not know β^{true} but we do know that our output depends on the input x in a quadratic manner. Thus, we consider a model of the form

$$Y_{\text{model}}(x; \beta) = \beta_0 + \beta_1 x + \beta_2 x^2 \ ,$$

and we determine the coefficients by measuring the output y for several different values of x. We are free to choose the number of measurements m and the measurement points x_i, $i = 1, \dots, m$. In particular, upon choosing the measurement points and taking a measurement at each point, we obtain a system of linear equations,

$$y_i = Y_{\text{model}}(x_i; \beta) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2, \quad i = 1, \dots, m \ ,$$

where y_i is the measurement corresponding to the input x_i.

Note that the equation is linear in our unknowns $\{\beta_0, \beta_1, \beta_2\}$ (the appearance of x_i^2 only affects the manner in which data enters the equation). Because the dependency on the parameters is linear, we can write the system as matrix equation,

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}}_{Y} = \underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix}}_{\beta},$$

or, more compactly,

$$Y = X\beta \ .$$

Note that this particular matrix X has a rather special structure — each row forms a geometric series and the ij-th entry is given by $B_{ij} = x_i^{j-1}$. Matrices with this structure are called Vandermonde matrices.

As in the friction coefficient example considered earlier, the row interpretation of matrix-vector product recovers the original set of equation

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 \\ \beta_0 + \beta_1 x_2 + \beta_2 x_2^2 \\ \vdots \\ \beta_0 + \beta_1 x_m + \beta_2 x_m^2 \end{pmatrix} = X\beta \ .$$

With the column interpretation, we immediately recognize that this is a problem of finding the three coefficients, or parameters, of the linear combination that yields the desired m-vector Y, i.e.

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \beta_0 \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \beta_1 \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} + \beta_2 \begin{pmatrix} x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \end{pmatrix} = X\beta \ .$$

We know that if have three or more non-degenerate measurements (i.e., $m \geq 3$), then we can find the unique solution to the linear system. Moreover, the solution is the coefficients of the underlying polynomial, $(\beta_0^{\text{true}}, \beta_1^{\text{true}}, \beta_2^{\text{true}})$.

Example 17.1.1 A quadratic polynomial
Let us consider a more specific case, where the underlying polynomial is of the form

$$y(x) = -\frac{1}{2} + \frac{2}{3}x - \frac{1}{8}cx^2 \ .$$

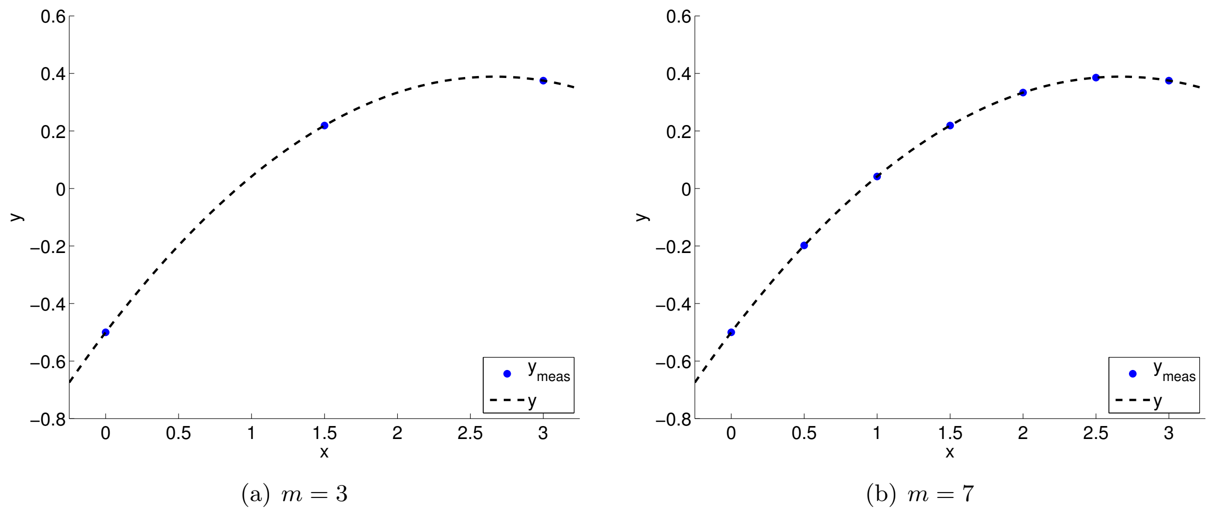
We recognize that $y(x) = Y_{\text{model}}(x; \beta^{\text{true}})$ for $Y_{\text{model}}(x;\beta) = \beta_0 + \beta_1 x + \beta_2 x^2$ and the true parameters

$$\beta_0^{\text{true}} = -\frac{1}{2}, \quad \beta_1^{\text{true}} = \frac{2}{3}, \quad \text{and} \quad \beta_2^{\text{true}} = -\frac{1}{8}c \ .$$

The parameter c controls the degree of quadratic dependency; in particular, $c = 0$ results in an affine function.

First, we consider the case with $c = 1$, which results in a strong quadratic dependency, i.e., $\beta_2^{\text{true}} = -1/8$. The result of measuring y at three non-degenerate points ($m = 3$) is shown in Figure 17.1(a). Solving the 3×3 linear system with the coefficients as the unknown, we obtain

$$\beta_0 = -\frac{1}{2}, \quad \beta_1 = \frac{2}{3}, \quad \text{and} \quad \beta_2 = -\frac{1}{8} \ .$$



(a) $m = 3$ (b) $m = 7$

Figure 17.1: Deducing the coefficients of a polynomial with a strong quadratic dependence.

Not surprisingly, we can find the true coefficients of the quadratic equation using three data points.

Suppose we take more measurements. An example of taking seven measurements ($m = 7$) is shown in Figure 17.1(b). We now have seven data points and three unknowns, so we must solve the 7×3 linear system, i.e., find the set $\beta = \{\beta_0, \beta_1, \beta_2\}$ that satisfies all seven equations. The solution to the linear system, of course, is given by

$$\beta_0 = -\frac{1}{2}, \quad \beta_1 = \frac{2}{3}, \quad \text{and} \quad \beta_2 = -\frac{1}{8} .$$

The result is correct ($\beta = \beta^{\text{true}}$) and, in particular, no different from the result for the $m = 3$ case.

We can modify the underlying polynomial slightly and repeat the same exercise. For example, let us consider the case with $c = 1/10$, which results in a much weaker quadratic dependency of y on x, i.e., $\beta_2^{\text{true}} = -1/80$. As shown in Figure 17.1.1, we can take either $m = 3$ or $m = 7$ measurements. Similar to the $c = 1$ case, we identify the true coefficients,

$$\beta_0 = -\frac{1}{2}, \quad \beta_1 = \frac{2}{3}, \quad \text{and} \quad \beta_2 = -\frac{1}{80} ,$$

using the either $m = 3$ or $m = 7$ (in fact using any three or more non-degenerate measurements).

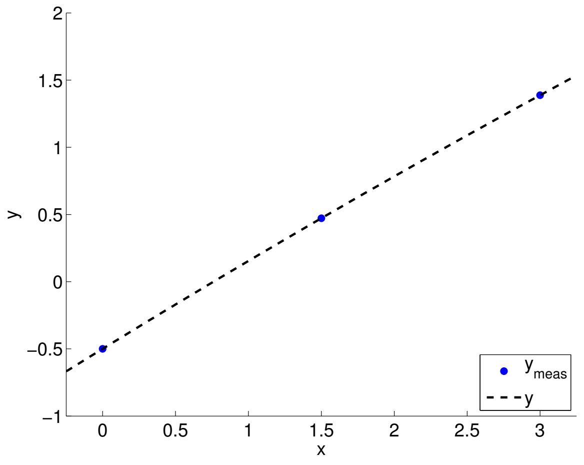
·

In the friction coefficient determination and the (particle motion) polynomial identification problems, we have seen that we can find a solution to the $m \times n$ overdetermined system ($m > n$) if

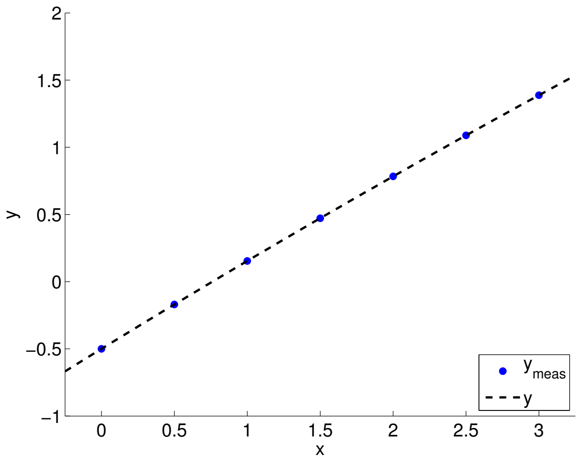
(a) our model includes the underlying input-output functional dependence — no bias;

(b) and the measurements are perfect — no noise.

As already stated, in practice, these two assumptions are rarely satisfied; i.e., models are often (in fact, always) incomplete and measurements are often inaccurate. (For example, in our particle motion model, we have neglected friction.) We can still construct a $m \times n$ linear system $Bz = g$ using our model and measurements, but the solution to the system in general does not exist. Knowing that we cannot find the "solution" to the overdetermined linear system, our objective is



(a) $m = 3$



(b) $m = 7$

Figure 17.2: Deducing the coefficients of a polynomial with a weak quadratic dependence.

to find a solution that is "close" to satisfying the solution. In the following section, we will define the notion of "closeness" suitable for our analysis and introduce a general procedure for finding the "closest" solution to a general overdetermined system of the form

$$Bz = g \ ,$$

where $B \in \mathbb{R}^{m\times n}$ with $m > n$. We will subsequently address the meaning and interpretation of this (non-) solution.

17.2 Overdetermined Systems

Let us consider an overdetermined linear system — such as the one arising from the regression example earlier — of the form

$$Bz = g \ ,$$

or, more explicitly,

$$\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix} .$$

Our objective is to find z that makes the three-component vector equation true, i.e., find the solution to the linear system. In Chapter 16, we considered the "forward problem" of matrix-vector multiplication in which, given z, we calculate $g = Bz$. We also briefly discussed the "inverse" problem in which given g we would like to find z. But for $m \neq n$, B^{-1} does not exist; as discussed in the previous section, there may be no z that satisfies $Bz = g$. Thus, we will need to look for a z that satisfies the equation "closely" in the sense we must specify and interpret. This is the focus of this section.[1]

[1]Note later (in Unit V) we shall look at the ostensibly simpler case in which B is square and a solution z exists and is even unique. But, for many reasons, overdetermined systems are a nicer place to start.

Row Interpretation

Let us consider a row interpretation of the overdetermined system. Satisfying the linear system requires

$$B_{i1}z_1 + B_{i2}z_2 = g_i, \quad i = 1, 2, 3 \ .$$

Note that each of these equations define a line in $\mathbb{R}^2$. Thus, satisfying the three equations is equivalent to finding a point that is shared by all three lines, which in general is not possible, as we will demonstrate in an example.

Example 17.2.1 row interpretation of overdetermined system
Let us consider an overdetermined system

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 2 & -3 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 5/2 \\ 2 \\ -2 \end{pmatrix} .$$

Using the row interpretation of the linear system, we see there are three linear equations to be satisfied. The set of points $x = (x_1, x_2)$ that satisfies the first equation,

$$1 \cdot x_1 + 2 \cdot x_2 = \frac{5}{2} \ ,$$

form a line

$$L_1 = \{(x_1, x_2) : 1 \cdot x_2 + 2 \cdot x_2 = 5/2\}$$

in the two dimensional space. Similarly, the sets of points that satisfy the second and third equations form lines described by

$$\begin{aligned} L_2 &= \{(x_1, x_2) : 2 \cdot x_1 + 1 \cdot x_2 = 2\} \\ L_3 &= \{(x_1, x_2) : 2 \cdot x_1 - 3 \cdot x_2 = -2\} \ . \end{aligned}$$

These set of points in L_1, L_2, and L_3, or the lines, are shown in Figure 17.3(a).

The solution to the linear system must satisfy each of the three equations, i.e., belong to all three lines. This means that there must be an intersection of all three lines and, if it exists, the solution is the intersection. This linear system has the solution

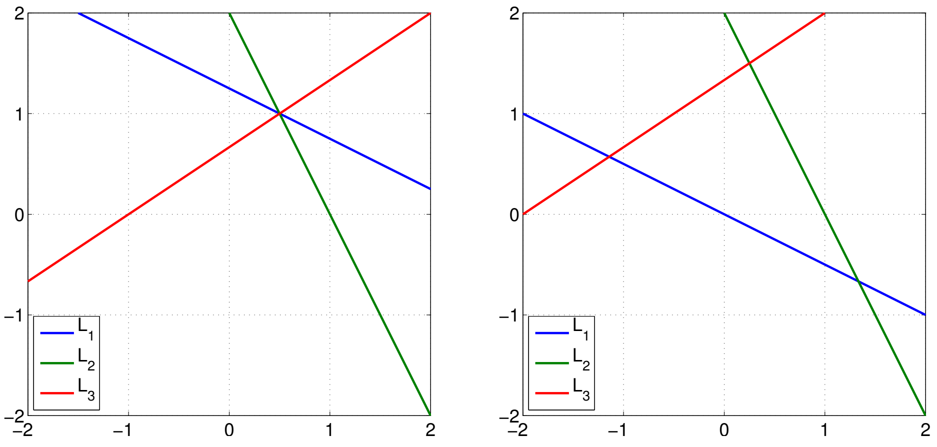
$$z = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} .$$

However, three lines intersecting in $\mathbb{R}^2$ is a rare occurrence; in fact the right-hand side of the system was chosen carefully so that the system has a solution in this example. If we perturb either the matrix or the right-hand side of the system, it is likely that the three lines will no longer intersect.

A more typical overdetermined system is the following system,

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 2 & -3 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ -4 \end{pmatrix} .$$

Again, interpreting the matrix equation as a system of three linear equations, we can illustrate the set of points that satisfy each equation as a line in $\mathbb{R}^2$ as shown in Figure 17.3(b). There is no solution to this overdetermined system, because there is no point (z_1, z_2) that belongs to all three lines, i.e., the three lines do not intersect at a point.



(a) system with a solution (b) system without a solution

Figure 17.3: Illustration of the row interpretation of the overdetermined systems. Each line is a set of points that satisfies $B_i x = g_i$, $i = 1, 2, 3$.

Column Interpretation

Let us now consider a column interpretation of the overdetermined system. Satisfying the linear system requires

$$z_1 \cdot \begin{pmatrix} B_{11} \\ B_{21} \\ B_{31} \end{pmatrix} + z_2 \cdot \begin{pmatrix} B_{12} \\ B_{22} \\ B_{32} \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix} .$$

In other words, we consider a linear combination of two vectors in $\mathbb{R}^3$ and try to match the right-hand side $g \in \mathbb{R}^3$. The vectors span at most a plane in $\mathbb{R}^3$, so there is no weight (z_1, z_2) that makes the equation hold unless the vector g happens to lie in the plane. To clarify the idea, let us consider a specific example.

Example 17.2.2 column interpretation of overdetermined system
For simplicity, let us consider the following special case:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3/2 \\ 2 \end{pmatrix} .$$

The column interpretation results in

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} z_1 + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} z_2 = \begin{pmatrix} 1 \\ 3/2 \\ 2 \end{pmatrix} .$$

By changing z_1 and z_2, we can move in the plane

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} z_1 + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} z_2 = \begin{pmatrix} z_1 \\ z_2 \\ 0 \end{pmatrix} .$$

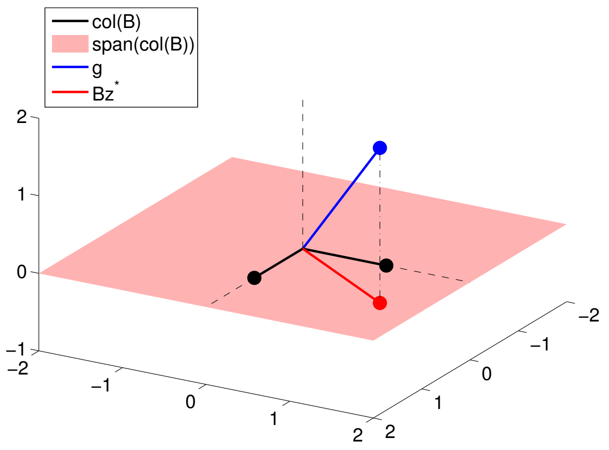


Figure 17.4: Illustration of the column interpretation of the overdetermined system.

Clearly, if $g_3 \neq 0$, it is not possible to find z_1 and z_2 that satisfy the linear equation, $Bz = g$. In other words, g must lie in the plane spanned by the columns of B, which is the $1-2$ plane in this case.

Figure 17.4 illustrates the column interpretation of the overdetermined system. The vector $g \in \mathbb{R}^3$ does not lie in the space spanned by the columns of B, thus there is no solution to the system. However, if g_3 is "small", then we can find a z^* such that Bz^* is "close" to g, i.e., a good approximation to g. Such an approximation is shown in the figure, and the next section discusses how to find such an approximation.

17.3 Least Squares

17.3.1 Measures of Closeness

In the previous section, we observed that it is in general not possible to find a solution to an overdetermined system. Our aim is thus to find z such that Bz is "close" to g, i.e., z such that

$$Bz \approx g \ ,$$

for $B \in \mathbb{R}^{m\times n}$, $m > n$. For convenience, let us introduce the *residual* function, which is defined as

$$r(z) \equiv g - Bz \ .$$

Note that

$$r_i = g_i - (Bz)_i = g_i - \sum_{j=1}^{n} B_{ij} z_j, \quad i = 1, \dots, m \ .$$

Thus, r_i is the "extent" to which i-th equation $(Bz)_i = g_i$ is not satisfied. In particular, if $r_i(z) = 0$, $i = 1, \dots, m$, then $Bz = g$ and z is the solution to the linear system. We note that the residual is a measure of closeness described by m values. It is more convenient to have a single scalar value for assessing the extent to which the equation is satisfied. A simple way to achieve this is to take a norm of the residual vector. Different norms result in different measures of closeness, which in turn produce different best-fit solutions.

Begin Advanced Material

Let us consider first two examples, neither of which we will pursue in this chapter.

Example 17.3.1 ℓ_1 minimization
The first method is based on measuring the residual in the 1-norm. The scalar representing the extent of mismatch is

$$J_1(z) \equiv \|r(z)\|_1 = \sum_{i=1}^{m} |r_i(z)| = \sum_{i=1}^{m} |(g - Bz)_i| \ .$$

The best z, denoted by z^*, is the z that minimizes the extent of mismatch measured in $J_1(z)$, i.e.

$$z^* = \arg \min_{z \in \mathbb{R}^m} J_1(z) \ .$$

The $\arg\min_{z\in\mathbb{R}^n} J_1(z)$ returns the argument z that minimizes the function $J_1(z)$. In other words, z^* satisfies

$$J_1(z^*) \le J_1(z), \quad \forall\, z \in \mathbb{R}^m \ .$$

This minimization problem can be formulated as a linear programming problem. The minimizer is not necessarily unique and the solution procedure is not as simple as that resulting from the 2-norm. Thus, we will not pursue this option here.

·

Example 17.3.2 ℓ_∞ minimization
The second method is based on measuring the residual in the ∞-norm. The scalar representing the extent of mismatch is

$$J_\infty(z) \equiv \|r(z)\|_\infty = \max_{i=1,\dots,m} |r_i(z)| = \max_{i=1,\dots,m} |(g - Bz)_i| \ .$$

The best z that minimizes $J_\infty(z)$ is

$$z^* = \arg \min_{z \in \mathbb{R}^n} J_\infty(z) \ .$$

This so-called min-max problem can also be cast as a linear programming problem. Again, this procedure is rather complicated, and the solution is not necessarily unique.

·

End Advanced Material

17.3.2 Least-Squares Formulation (ℓ_2 minimization)

Minimizing the residual measured in (say) the 1-norm or ∞-norm results in a linear programming problem that is not so easy to solve. Here we will show that measuring the residual in the 2-norm results in a particularly simple minimization problem. Moreover, the solution to the minimization problem is unique assuming that the matrix B is full rank — has n independent columns. We shall assume that B does indeed have independent columns.

The scalar function representing the extent of mismatch for ℓ_2 minimization is

$$J_2(z) \equiv \|r(z)\|_2^2 = r^{\mathrm{T}}(z)r(z) = (g - Bz)^{\mathrm{T}}(g - Bz) \ .$$

Note that we consider the square of the 2-norm for convenience, rather than the 2-norm itself. Our objective is to find z^* such that

$$z^* = \arg\min_{z \in \mathbb{R}^n} J_2(z) \ ,$$

which is equivalent to find z^* with

$$\|g - Bz^*\|_2^2 = J_2(z^*) < J_2(z) = \|g - Bz\|_2^2, \quad \forall\, z \neq z^* \ .$$

(Note "arg min" refers to the argument which minimizes: so "min" is the minim*um* and "arg min" is the minimiz*er*.) Note that we can write our objective function $J_2(z)$ as

$$J_2(z) = \|r(z)\|_2^2 = r^{\mathrm{T}}(z)r(z) = \sum_{i=1}^{m} (r_i(z))^2 \ .$$

In other words, our objective is to minimize the sum of the square of the residuals, i.e., *least squares*. Thus, we say that z^* is the least-squares solution to the overdetermined system $Bz = g$: z^* is that z which makes $J_2(z)$ — the sum of the squares of the residuals — as small as possible.

Note that if $Bz = g$ does have a solution, the least-squares solution is the solution to the overdetermined system. If z is the solution, then $r = Bz - g = 0$ and in particular $J_2(z) = 0$, which is the minimum value that J_2 can take. Thus, the solution z is the minimizer of J_2: $z = z^*$. Let us now derive a procedure for solving the least-squares problem for a more general case where $Bz = g$ does not have a solution.

For convenience, we drop the subscript 2 of the objective function J_2, and simply denote it by J. Again, our objective is to find z^* such that

$$J(z^*) < J(z), \quad \forall\, z \neq z^* \ .$$

Expanding out the expression for $J(z)$, we have

$$\begin{aligned} J(z) &= (g - Bz)^{\mathrm{T}}(g - Bz) = (g^{\mathrm{T}} - (Bz)^{\mathrm{T}})(g - Bz) \\ &= g^{\mathrm{T}}(g - Bz) - (Bz)^{\mathrm{T}}(g - Bz) \\ &= g^{\mathrm{T}}g - g^{\mathrm{T}}Bz - (Bz)^{\mathrm{T}}g + (Bz)^{\mathrm{T}}(Bz) \\ &= g^{\mathrm{T}}g - g^{\mathrm{T}}Bz - z^{\mathrm{T}}B^{\mathrm{T}}g + z^{\mathrm{T}}B^{\mathrm{T}}Bz \ , \end{aligned}$$

where we have used the transpose rule which tells us that $(Bz)^{\mathrm{T}} = z^{\mathrm{T}}B^{\mathrm{T}}$. We note that $g^{\mathrm{T}}Bz$ is a scalar, so it does not change under the transpose operation. Thus, $g^{\mathrm{T}}Bz$ can be expressed as

$$g^{\mathrm{T}}Bz = (g^{\mathrm{T}}Bz)^{\mathrm{T}} = z^{\mathrm{T}}B^{\mathrm{T}}g \ ,$$

again by the transpose rule. The function J thus simplifies to

$$J(z) = g^{\mathrm{T}}g - 2z^{\mathrm{T}}B^{\mathrm{T}}g + z^{\mathrm{T}}B^{\mathrm{T}}Bz \ .$$

For convenience, let us define $N \equiv B^{\mathrm{T}}B \in \mathbb{R}^{n\times n}$, so that

$$J(z) = g^{\mathrm{T}}g - 2z^{\mathrm{T}}B^{\mathrm{T}}g + z^{\mathrm{T}}Nz \ .$$

It is simple to confirm that each term in the above expression is indeed a scalar.

The solution to the minimization problem is given by

$$Nz^* = d \ ,$$

where $d = B^{\mathrm{T}}g$. The equation is called the "normal" equation, which can be written out as

$$\begin{pmatrix} N_{11} & N_{12} & \cdots & N_{1n} \\ N_{21} & N_{22} & \cdots & N_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ N_{n1} & N_{n2} & \cdots & N_{nn} \end{pmatrix} \begin{pmatrix} z_1^* \\ z_2^* \\ \vdots \\ z_n^* \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} .$$

The existence and uniqueness of z^* is guaranteed assuming that the columns of B are independent.

We provide below the proof that z^* is the unique minimizer of $J(z)$ in the case in which B has independent columns.

Proof. We first show that the normal matrix N is symmetric positive definite, i.e.

$$x^{\mathrm{T}}Nx > 0, \quad \forall\, x \in \mathbb{R}^n \ (x \neq 0) \ ,$$

assuming the columns of B are linearly independent. The normal matrix $N = B^{\mathrm{T}}B$ is symmetric because

$$N^{\mathrm{T}} = (B^{\mathrm{T}}B)^{\mathrm{T}} = B^{\mathrm{T}}(B^{\mathrm{T}})^{\mathrm{T}} = B^{\mathrm{T}}B = N \ .$$

To show N is positive definite, we first observe that

$$x^{\mathrm{T}}Nx = x^{\mathrm{T}}B^{\mathrm{T}}Bx = (Bx)^{\mathrm{T}}(Bx) = \|Bx\|^2 \ .$$

That is, $x^{\mathrm{T}}Nx$ is the 2-norm of Bx. Recall that the norm of a vector is zero if and only if the vector is the zero vector. In our case,

$$x^{\mathrm{T}}Nx = 0 \quad \text{if and only if} \quad Bx = 0 \ .$$

Because the columns of B are linearly independent, $Bx = 0$ if and only if $x = 0$. Thus, we have

$$x^{\mathrm{T}}Nx = \|Bx\|^2 > 0, \quad x \neq 0 \ .$$

Thus, N is symmetric positive definite.

Now recall that the function to be minimized is

$$J(z) = g^{\mathrm{T}}g - 2z^{\mathrm{T}}B^{\mathrm{T}}g + z^{\mathrm{T}}Nz \ .$$

If z^* minimizes the function, then for any $\delta z \neq 0$, we must have

$$J(z^*) < J(z^* + \delta z) \ ;$$

Let us expand $J(z^* + \delta z)$:

$$\begin{aligned} J(z^* + \delta z) &= g^{\mathrm{T}}g - 2(z^* + \delta z)^{\mathrm{T}}B^{\mathrm{T}}g + (z^* + \delta z)^{\mathrm{T}}N(z^* + \delta z) \ , \\ &= \underbrace{g^{\mathrm{T}}g - 2z^*B^{\mathrm{T}}g + (z^*)^{\mathrm{T}}Nz^*}_{J(z^*)} - 2\delta z^{\mathrm{T}}B^{\mathrm{T}}g + \delta z^{\mathrm{T}}Nz^* + \underbrace{(z^*)^{\mathrm{T}}N\delta z}_{\delta z^{\mathrm{T}}N^{\mathrm{T}}z^* = \delta z^{\mathrm{T}}Nz^*} + \delta z^{\mathrm{T}}N\delta z \ , \\ &= J(z^*) + 2\delta z^{\mathrm{T}}(Nz^* - B^{\mathrm{T}}g) + \delta z^{\mathrm{T}}N\delta z \ . \end{aligned}$$

Note that $N^{\mathrm{T}} = N$ because $N^{\mathrm{T}} = (B^{\mathrm{T}}B)^{\mathrm{T}} = B^{\mathrm{T}}B = N$. If z^* satisfies the normal equation, $Nz^* = B^{\mathrm{T}}g$, then

$$Nz^* - B^{\mathrm{T}}g = 0\ ,$$

and thus

$$J(z^* + \delta z) = J(z^*) + \delta z^{\mathrm{T}} N \delta z\ .$$

The second term is always positive because N is positive definite. Thus, we have

$$J(z^* + \delta z) > J(z^*), \quad \forall\, \delta z \neq 0\ ,$$

or, setting $\delta z = z - z^*$,

$$J(z^*) < J(z), \quad \forall\, z \neq z^*\ .$$

Thus, z^* satisfying the normal equation $Nz^* = B^{\mathrm{T}}g$ is the minimizer of J, i.e., the least-squares solution to the overdetermined system $Bz = g$. □

Example 17.3.3 2×1 **least-squares and its geometric interpretation**
Consider a simple case of a overdetermined system,

$$B = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \begin{pmatrix} z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Because the system is 2×1, there is a single scalar parameter, z, to be chosen. To obtain the normal equation, we first construct the matrix N and the vector d (both of which are simply scalar for this problem):

$$N = B^{\mathrm{T}}B = \begin{pmatrix} 2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = 5$$

$$d = B^{\mathrm{T}}g = \begin{pmatrix} 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 4\ .$$

Solving the normal equation, we obtain the least-squares solution

$$Nz^* = d \quad \Rightarrow \quad 5z^* = 4 \quad \Rightarrow \quad z^* = 4/5\ .$$

This choice of z yields

$$Bz^* = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \cdot \frac{4}{5} = \begin{pmatrix} 8/5 \\ 4/5 \end{pmatrix},$$

which of course is different from g.

The process is illustrated in Figure 17.5. The span of the column of B is the line parameterized by $\begin{pmatrix} 2 & 1 \end{pmatrix}^{\mathrm{T}} z$, $z \in \mathbb{R}$. Recall that the solution Bz^* is the point on the line that is closest to g in the least-squares sense, i.e.

$$\|Bz^* - g\|_2 < \|Bz - g\|, \quad \forall\, z \neq z^*\ .$$

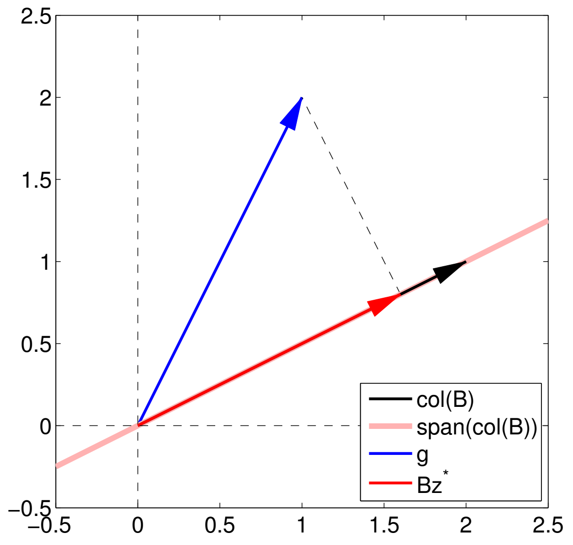


Figure 17.5: Illustration of least-squares in $\mathbb{R}^2$.

Recalling that the ℓ_2 distance is the usual Euclidean distance, we expect the closest point to be the orthogonal projection of g onto the line span(col(B)). The figure confirms that this indeed is the case. We can verify this algebraically,

$$B^{\mathrm{T}}(Bz^* - g) = \begin{pmatrix} 2 & 1 \end{pmatrix} \left(\begin{pmatrix} 2 \\ 1 \end{pmatrix} \cdot \frac{4}{5} - \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right) = \begin{pmatrix} 2 & 1 \end{pmatrix} \begin{pmatrix} 3/5 \\ -6/5 \end{pmatrix} = 0 \ .$$

Thus, the residual vector $Bz^* - g$ and the column space of B are orthogonal to each other. While the geometric illustration of orthogonality may be difficult for a higher-dimensional least squares, the orthogonality condition can be checked systematically using the algebraic method.

·

17.3.3 Computational Considerations

Let us analyze the computational cost of solving the least-squares system. The first step is the formulation of the normal matrix,

$$N = B^{\mathrm{T}} B \ ,$$

which requires a matrix-matrix multiplication of $B^{\mathrm{T}} \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times n}$. Because N is symmetric, we only need to compute the upper triangular part of N, which corresponds to performing $n(n+1)/2$ m-vector inner products. Thus, the computational cost is $mn(n+1)$. Forming the right-hand side,

$$d = B^{\mathrm{T}} g \ ,$$

requires a matrix-vector multiplication of $B^{\mathrm{T}} \in \mathbb{R}^{n \times m}$ and $g \in \mathbb{R}^m$. This requires n m-vector inner products, so the computational cost is $2mn$. This cost is negligible compared to the $mn(n+1)$ operations required to form the normal matrix. Finally, we must solve the n-dimensional linear system

$$Nz = d \ .$$

As we will see in the linear algebra unit, solving the $n \times n$ symmetric positive definite linear system requires approximately $\frac{1}{3}n^3$ operations using the Cholesky factorization (as we discuss further in Unit V). Thus, the total operation count is

$$C^{\text{normal}} \approx mn(n+1) + \frac{1}{3}n^3 \ .$$

For a system arising from regression, $m \gg n$, so we can further simplify the expression to

$$C^{\text{normal}} \approx mn(n+1) \approx mn^2 \ ,$$

which is quite modest for n not too large.

While the method based on the normal equation works well for small systems, this process turns out to be numerically "unstable" for larger problems. We will visit the notion of stability later; for now, we can think of stability as an ability of an algorithm to control the perturbation in the solution under a small perturbation in data (or input). In general, we would like our algorithm to be stable. We discuss below the method of choice.

Begin Advanced Material

QR Factorization and the Gram-Schmidt Procedure

A more stable procedure for solving the overdetermined system is that based on QR factorization. QR factorization is a procedure to factorize, or decompose, a matrix $B \in \mathbb{R}^{m\times n}$ into an orthonormal matrix $Q \in \mathbb{R}^{m\times n}$ and an upper triangular matrix $R \in \mathbb{R}^{n\times n}$ such that $B = QR$. Once we have such a factorization, we can greatly simplify the normal equation $B^{\mathrm{T}}Bz^* = B^{\mathrm{T}}g$. Substitution of the factorization into the normal equation yields

$$B^{\mathrm{T}}Bz^* = B^{\mathrm{T}}g \quad \Rightarrow \quad R^{\mathrm{T}}\underbrace{Q^{\mathrm{T}}Q}_{I}Rz^* = R^{\mathrm{T}}Q^{\mathrm{T}}g \quad \Rightarrow \quad R^{\mathrm{T}}Rz^* = R^{\mathrm{T}}Q^{\mathrm{T}}g \ .$$

Here, we use the fact that $Q^{\mathrm{T}}Q = I$ if Q is an orthonormal matrix. The upper triangular matrix is invertible as long as its diagonal entries are all nonzero (which is the case for B with linearly independent columns), so we can further simplify the expression to yield

$$Rz^* = Q^{\mathrm{T}}g \ .$$

Thus, once the factorization is available, we need to form the right-hand side $Q^{\mathrm{T}}g$, which requires $2mn$ operations, and solve the $n \times n$ upper triangular linear system, which requires n^2 operations. Both of these operations are inexpensive. The majority of the cost is in factorizing the matrix B into matrices Q and R.

There are two classes of methods for producing a QR factorization: the Gram-Schmidt procedure and the Householder transform. Here, we will briefly discuss the Gram-Schmidt procedure. The idea behind the Gram-Schmidt procedure is to successively turn the columns of B into orthonormal vectors to form the orthonormal matrix Q. For convenience, we denote the j-th column of B by b_j, i.e.

$$B = \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix},$$

where b_j is an m-vector. Similarly, we express our orthonormal matrix as

$$Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_n \end{pmatrix}.$$

Recall $q_i^{\mathrm{T}} q_j = \delta_{i\,j}$ (Kronecker-delta), $1 \le i, j \le n$.

The Gram-Schmidt procedure starts with a set which consists of a single vector, b_1. We construct an orthonormal set consisting of single vector q_1 that spans the same space as $\{b_1\}$. Trivially, we can take

$$q_1 = \frac{1}{\|b_1\|} b_1 \ .$$

Or, we can express b_1 as

$$b_1 = q_1 \|b_1\| \ ,$$

which is the product of a unit vector and an amplitude.

Now we consider a set which consists of the first two columns of B, $\{b_1, b_2\}$. Our objective is to construct an orthonormal set $\{q_1, q_2\}$ that spans the same space as $\{b_1, b_2\}$. In particular, we will keep the q_1 we have constructed in the first step unchanged, and choose q_2 such that (i) it is orthogonal to q_1, and (ii) $\{q_1, q_2\}$ spans the same space as $\{b_1, b_2\}$. To do this, we can start with b_2 and first remove the component in the direction of q_1, i.e.

$$\tilde{q}_2 = b_2 - (q_1^{\mathrm{T}} b_2) q_1 \ .$$

Here, we recall the fact that the inner product $q_1^{\mathrm{T}} b_2$ is the component of b_2 in the direction of q_1. We can easily confirm that $\tilde{q}_2$ is orthogonal to q_1, i.e.

$$q_1^{\mathrm{T}} \tilde{q}_2 = q_1^{\mathrm{T}} (b_2 - (q_1^{\mathrm{T}} b_2) q_1) = q_1^{\mathrm{T}} b_2 - (q_1^{\mathrm{T}} b_2) q_1^{\mathrm{T}} q_1 = q_1^{\mathrm{T}} b_2 - (q_1^{\mathrm{T}} b_2) \cdot 1 = 0 \ .$$

Finally, we normalize $\tilde{q}_2$ to yield the unit length vector

$$q_2 = \tilde{q}_2 / \|\tilde{q}_2\| \ .$$

With some rearrangement, we see that b_2 can be expressed as

$$b_2 = (q_1^{\mathrm{T}} b_2) q_1 + \tilde{q}_2 = (q_1^{\mathrm{T}} b_2) q_1 + \|\tilde{q}_2\| q_2 \ .$$

Using a matrix-vector product, we can express this as

$$b_2 = \begin{pmatrix} q_1 & q_2 \end{pmatrix} \begin{pmatrix} q_1^{\mathrm{T}} b_2 \\ \|\tilde{q}_2\| \end{pmatrix} .$$

Combining with the expression for b_1, we have

$$\begin{pmatrix} b_1 & b_2 \end{pmatrix} = \begin{pmatrix} q_1 & q_2 \end{pmatrix} \begin{pmatrix} \|b_1\| & q_1^{\mathrm{T}} b_2 \\ & \|\tilde{q}_2\| \end{pmatrix} .$$

In two steps, we have factorized the first two columns of B into an $m \times 2$ orthogonal matrix (q_1, q_2) and a 2×2 upper triangular matrix. The Gram-Schmidt procedure consists of repeating the procedure n times; let us show one more step for clarity.

In the third step, we consider a set which consists of the first three columns of B, $\{b_1, b_2, b_3\}$. Our objective it to construct an orthonormal set $\{q_1, q_2, q_3\}$. Following the same recipe as the second step, we keep q_1 and q_2 unchanged, and choose q_3 such that (i) it is orthogonal to q_1 and q_2, and (ii) $\{q_1, q_2, q_3\}$ spans the same space as $\{b_1, b_2, b_3\}$. This time, we start from b_3, and remove the components of b_3 in the direction of q_1 and q_2, i.e.

$$\tilde{q}_3 = b_3 - (q_1^{\mathrm{T}} b_3) q_1 - (q_2^{\mathrm{T}} b_3) q_2 \ .$$

Again, we recall that $q_1^{\mathrm{T}} b_3$ and $q_2^{\mathrm{T}} b_3$ are the components of b_3 in the direction of q_1 and q_2, respectively. We can again confirm that $\tilde{q}_3$ is orthogonal to q_1

$$q_1^{\mathrm{T}} \tilde{q}_3 = q_1^{\mathrm{T}}(b_3 - (q_1^{\mathrm{T}} b_3) q_1 - (q_2^{\mathrm{T}} b_3) q_2) = q_1^{\mathrm{T}} b_3 - (q_1^{\mathrm{T}} b_3) \cancelto{1}{q_1^{\mathrm{T}} q_1} - (q_2^{\mathrm{T}} b_3) \cancelto{0}{q_1^{\mathrm{T}} q_2} = 0$$

and to q_2

$$q_2^{\mathrm{T}} \tilde{q}_3 = q_2^{\mathrm{T}}(b_3 - (q_1^{\mathrm{T}} b_3) q_1 - (q_2^{\mathrm{T}} b_3) q_2) = q_2^{\mathrm{T}} b_3 - (q_1^{\mathrm{T}} b_3) \cancelto{0}{q_2^{\mathrm{T}} q_1} - (q_2^{\mathrm{T}} b_3) \cancelto{1}{q_2^{\mathrm{T}} q_2} = 0 \ .$$

We can express b_3 as

$$b_3 = (q_1^{\mathrm{T}} b_3) q_1 + (q_2^{\mathrm{T}} b_3) q_2 + \|\tilde{q}_3\| q_3 \ .$$

Or, putting the first three columns together

$$\begin{pmatrix} b_1 & b_2 & b_3 \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix} \begin{pmatrix} \|b_1\| & q_1^{\mathrm{T}} b_2 & q_1^{\mathrm{T}} b_3 \\ & \|\tilde{q}_2\| & q_2^{\mathrm{T}} b_3 \\ & & \|\tilde{q}_3\| \end{pmatrix} .$$

We can see that repeating the procedure n times would result in the complete orthogonalization of the columns of B.

Let us count the number of operations of the Gram-Schmidt procedure. At j-th step, there are $j-1$ components to be removed, each requiring of $4m$ operations. Thus, the total operation count is

$$C^{\text{Gram-Schmidt}} \approx \sum_{j=1}^{n} (j-1) 4m \approx 2mn^2 \ .$$

Thus, for solution of the least-squares problem, the method based on Gram-Schmidt is approximately twice as expensive as the method based on normal equation for $m \gg n$. However, the superior numerical stability often warrants the additional cost.

We note that there is a modified version of Gram-Schmidt, called the modified Gram-Schmidt procedure, which is more stable than the algorithm presented above. The modified Gram-Schmidt procedure requires the same computational cost. There is also another fundamentally different QR factorization algorithm, called the Householder transformation, which is even more stable than the modified Gram-Schmidt procedure. The Householder algorithm requires approximately the same cost as the Gram-Schmidt procedure.

End Advanced Material

Begin Advanced Material

17.3.4 Interpretation of Least Squares: Projection

So far, we have discussed a procedure for solving an overdetermined system,

$$Bz = g \ ,$$

in the least-squares sense. Using the column interpretation of matrix-vector product, we are looking for the linear combination of the columns of B that minimizes the 2-norm of the residual — the mismatch between a representation Bz and the data g. The least-squares solution to the problem is

$$B^{\mathrm{T}}Bz^* = B^{\mathrm{T}}g \quad \Rightarrow \quad z^* = (B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}g \ .$$

That is, the closest approximation of the data g using the columns of B is

$$g^{\mathrm{LS}} = Bz^* = B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}g = P^{\mathrm{LS}}g \ .$$

Our best representation of g, g^{LS}, is the projection of g by the projector P^{LS}. We can verify that the operator $P^{\mathrm{LS}} = B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}$ is indeed a projector:

$$\begin{aligned}(P^{\mathrm{LS}})^2 = (B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}})^2 = B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}} &= B\underbrace{((B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}B)}_{I}(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}} \\ &= B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}} = P^{\mathrm{LS}} \ .\end{aligned}$$

In fact, P^{LS} is an orthogonal projector because P^{LS} is symmetric. This agrees with our intuition; the closest representation of g using the columns of B results from projecting g onto $\mathrm{col}(B)$ along a space orthogonal to $\mathrm{col}(B)$. This is clearly demonstrated for $\mathbb{R}^2$ in Figure 17.5 considered earlier.

Using the orthogonal projector onto $\mathrm{col}(B)$, P^{LS}, we can think of another interpretation of least-squares solution. We first project the data g orthogonally to the column space to form

$$g^{\mathrm{LS}} = P^{\mathrm{LS}}g \ .$$

Then, we find the coefficients for the linear combination of the columns of B that results in $P^{\mathrm{LS}}g$, i.e.

$$Bz^* = P^{\mathrm{LS}}g \ .$$

This problem has a solution because $P^{\mathrm{LS}}g \in \mathrm{col}(B)$.

This interpretation is useful especially when the QR factorization of B is available. If $B = QR$, then $\mathrm{col}(B) = \mathrm{col}(Q)$. So, the orthogonal projector onto $\mathrm{col}(B)$ is the same as the orthogonal projector onto $\mathrm{col}(Q)$ and is given by

$$P^{\mathrm{LS}} = QQ^{\mathrm{T}} \ .$$

We can verify that P^{LS} is indeed an orthogonal projector by checking that it is (i) idempotent ($P^{\mathrm{LS}}P^{\mathrm{LS}} = P^{\mathrm{LS}}$), and (ii) symmetric ($(P^{\mathrm{LS}})^{\mathrm{T}} = P^{\mathrm{LS}}$), i.e.

$$\begin{aligned}P^{\mathrm{LS}}P^{\mathrm{LS}} &= (QQ^{\mathrm{T}})(QQ^{\mathrm{T}}) = Q\underbrace{Q^{\mathrm{T}}Q}_{I}Q^{\mathrm{T}} = QQ^{\mathrm{T}} = P^{\mathrm{LS}} \ , \\ (P^{\mathrm{LS}})^{\mathrm{T}} &= (QQ^{\mathrm{T}})^{\mathrm{T}} = (Q^{\mathrm{T}})^{\mathrm{T}}Q^{\mathrm{T}} = QQ^{\mathrm{T}} = P^{\mathrm{LS}} \ .\end{aligned}$$

Using the QR factorization of B, we can rewrite the least-squares solution as

$$Bz^* = P^{\mathrm{LS}}g \quad \Rightarrow \quad QRz^* = QQ^{\mathrm{T}}g \ .$$

Applying Q^{T} on both sides and using the fact that $Q^{\mathrm{T}}Q = I$, we obtain

$$Rz^* = Q^{\mathrm{T}}g \ .$$

Geometrically, we are orthogonally projecting the data g onto $\mathrm{col}(Q)$ but representing the projected solution in the basis $\{q_i\}_{i=1}^n$ of the n-dimensional space (instead of in the standard basis of $\mathbb{R}^m$). Then, we find the coefficients z^* that yield the projected data.

End Advanced Material

17.3.5 Error Bounds for Least Squares

Perhaps the most obvious way to measure the goodness of our solution is in terms of the residual $\|g - Bz^*\|$ which indicates the extent to which the equations $Bz^* = g$ are satisfied — how well Bz^* predicts g. Since we choose z^* to minimize $\|g - Bz^*\|$ we can hope that $\|g - Bz^*\|$ is small. But it is important to recognize that in most cases g only reflects data from a particular experiment whereas we would like to then use our prediction for z^* in other, different, experiments or even contexts. For example, the friction coefficient we measure in the laboratory will subsequently be used "in the field" as part of a larger system prediction for, say, robot performance. In this sense, not only might the residual not be a good measure of the "error in z," a smaller residual might not even imply a "better prediction" for z. In this section, we look at how noise and incomplete models (bias) can be related directly to our prediction for z.

Note that, for notational simplicity, we use subscript 0 to represent superscript "true" in this section.

Error Bounds with Respect to Perturbation in Data, g (constant model)

Let us consider a parameter fitting for a simple constant model. First, let us assume that there is a solution z_0 to the overdetermined system

$$\underbrace{\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}}_{B} z_0 = \underbrace{\begin{pmatrix} g_{0,1} \\ g_{0,2} \\ \vdots \\ g_{0,m} \end{pmatrix}}_{g_0} .$$

Because z_0 is the solution to the system, g_0 must be a constant multiple of B. That is, the entries of g_0 must all be the same. Now, suppose that the data is perturbed such that $g \neq g_0$. With the perturbed data, the overdetermined system is unlikely to have a solution, so we consider the least-squares solution z^* to the problem

$$Bz = g .$$

We would like to know how much perturbation in the data $g - g_0$ changes the solution $z^* - z_0$.

To quantify the effect of the perturbation, we first note that both the original solution and the solution to the perturbed system satisfy the normal equation, i.e.

$$B^{\mathrm{T}} B z_0 = B^{\mathrm{T}} g_0 \quad \text{and} \quad B^{\mathrm{T}} B z^* = B^{\mathrm{T}} g .$$

Taking the difference of the two expressions, we obtain

$$B^{\mathrm{T}} B (z^* - z_0) = B^{\mathrm{T}} (g - g_0) .$$

For B with the constant model, we have $B^{\mathrm{T}} B = m$, simplifying the expression to

$$\begin{aligned} z^* - z_0 &= \frac{1}{m} B^{\mathrm{T}} (g - g_0) \\ &= \frac{1}{m} \sum_{i=1}^{m} (g - g_0)_i . \end{aligned}$$

Thus if the "noise" is close to zero-mean, z^* is close to Z_0. More generally, we can show that

$$|z^* - z_0| \leq \frac{1}{\sqrt{m}} \|g - g_0\| \ .$$

We see that the deviation in the solution is bounded by the perturbation data. Thus, our least-squares solution z^* is a good approximation as long as the perturbation $\|g - g_0\|$ is small.

To prove this result, we apply the Cauchy-Schwarz inequality, i.e.

$$|z^* - z_0| = \frac{1}{m}|B^{\mathrm{T}}(g - g_0)| \leq \frac{1}{m}\|B\| \, \|g - g_0\| = \frac{1}{m}\sqrt{m}\|g - g_0\| = \frac{1}{\sqrt{m}}\|g - g_0\| \ .$$

Recall that the Cauchy-Schwarz inequality gives a rather pessimistic bound when the two vectors are not very well aligned.

Let us now study more formally how the alignment of the two vectors B and $g - g_0$ affects the error in the solution. To quantify the effect let us recall that the least-squares solution satisfies

$$Bz^* = P^{\mathrm{LS}} g \ ,$$

where P^{LS} is the orthogonal projector onto the column space of B, $\mathrm{col}(B)$. If $g - g_0$ is exactly zero mean, i.e.

$$\frac{1}{m}\sum_{i=1}^{m}(g_{0,i} - g_i) = 0 \ ,$$

then $g - g_0$ is orthogonal to $\mathrm{col}(B)$. Because any perturbation orthogonal to $\mathrm{col}(B)$ lies in the direction along which the projection is performed, it does not affect $P^{\mathrm{LS}}g$ (and hence Bz^*), and in particular z^*. That is, the least-squares solution, z^*, to

$$Bz = g = g_0 + (g - g_0)$$

is z_0 if $g - g_0$ has zero mean. We can also show that the zero-mean perturbation has no influence in the solution algebraically using the normal equation, i.e.

$$B^{\mathrm{T}}Bz^* = B^{\mathrm{T}}(g_0 + (g - g_0)) = B^{\mathrm{T}}g_0 + \cancelto{0}{B^{\mathrm{T}}(g - g_0)} = B^{\mathrm{T}}g_0 \ .$$

The perturbed data g does not enter the calculation of z^* if $g - g_0$ has zero mean. Thus, any error in the solution $z - z_0$ must be due to the non-zero-mean perturbation in the data. Consequently, the bound based on the Cauchy-Schwarz inequality is rather pessimistic when the perturbation is close to zero mean.

Error Bounds with Respect to Perturbation in Data, g (general)

Let us now generalize the perturbation analysis to a general overdetermined system,

$$Bz_0 = g_0 \ ,$$

where $B \in \mathbb{R}^{m\times n}$ with $m > n$. We assume that g_0 is chosen such that the solution to the linear system exists. Now let us say measurement error has corrupted g_0 to $g = g_0 + \epsilon$. In particular, we assume that the linear system

$$Bz = g$$

does not have a solution. Thus, we instead find the least-squares solution z^* of the system.

To establish the error bounds, we will first introduce the concept of maximum and minimum singular values, which help us characterize the behavior of B. The maximum and minimum singular values of B are defined by

$$\nu_{\max}(B) = \max_{v \in \mathbb{R}^n} \frac{\|Bv\|}{\|v\|} \quad \text{and} \quad \nu_{\min}(B) = \min_{v \in \mathbb{R}^n} \frac{\|Bv\|}{\|v\|} \ .$$

Note that, because the norm scales linearly under scalar multiplication, equivalent definitions of the singular values are

$$\nu_{\max}(B) = \max_{\substack{v \in \mathbb{R}^n \\ \|v\|=1}} \|Bv\| \quad \text{and} \quad \nu_{\min}(B) = \min_{\substack{v \in \mathbb{R}^n \\ \|v\|=1}} \|Bv\| \ .$$

In other words, the maximum singular value is the maximum stretching that B can induce to a unit vector. Similarly, the minimum singular value is the maximum contraction B can induce. In particular, recall that if the columns of B are not linearly independent, then we can find a non-trivial v for which $Bv = 0$. Thus, if the columns of B are linearly dependent, $\nu_{\min}(B) = 0$.

We also note that the singular values are related to the eigenvalues of $B^{\mathrm{T}}B$. Recall that 2-norm is related to the inner product by

$$\|Bv\|^2 = (Bv)^{\mathrm{T}}(Bv) = v^{\mathrm{T}}B^{\mathrm{T}}Bv \ ,$$

thus, from the Rayleigh quotient, the square root of the maximum and minimum eigenvalues of $B^{\mathrm{T}}B$ are the maximum and minimum singular values of B.

Let us quantify the sensitivity of the solution error to the right-hand side in two different manners. First is the absolute conditioning, which relates $\|z^* - z_0\|$ to $\|g - g_0\|$. The bound is given by

$$\|z^* - z_0\| \leq \frac{1}{\nu_{\min}(B)} \|g - g_0\| \ .$$

Second is the relative conditioning, which relates the relative perturbation in the solution $\|z^* - z_0\|/\|z_0\|$ and the relative perturbation in the right-hand side $\|g - g_0\|/\|g_0\|$. This bound is give by

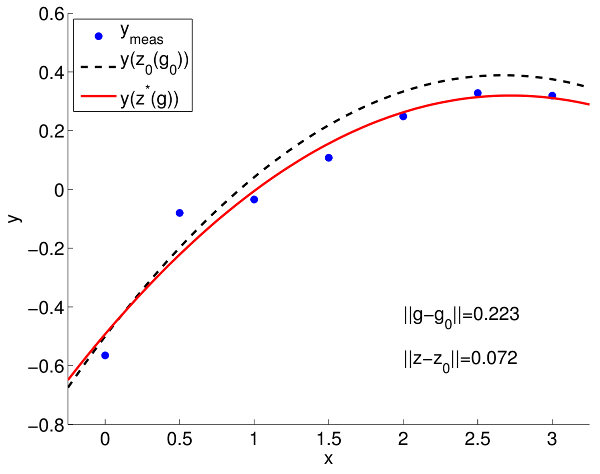
$$\frac{\|z^* - z_0\|}{\|z_0\|} = \frac{\nu_{\max}(B)}{\nu_{\min}(B)} \frac{\|g - g_0\|}{\|g_0\|} \ .$$

We derive these results shortly.

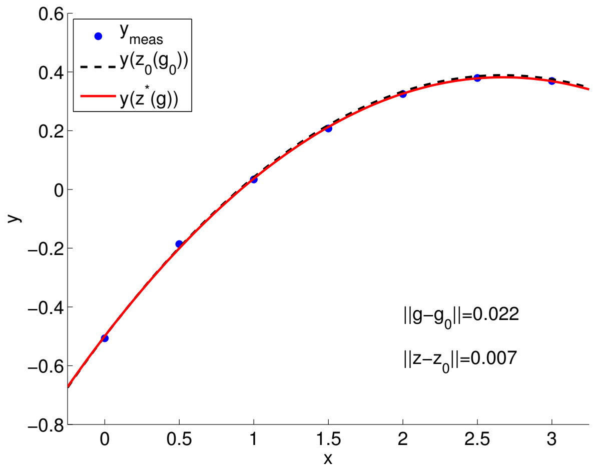
If the perturbation $\|g - g_0\|$ is small, we expect the error $\|z^* - z_0\|$ to be small as long as B is well-conditioned in the sense that $\nu_{\max}(B)/\nu_{\min}(B)$ is not too large. Note that if B has linearly dependent columns, then $\nu_{\min} = 0$ and $\nu_{\max}/\nu_{\min}$ is infinite; thus, $\nu_{\max}/\nu_{\min}$ is a measure of the independence of the columns of B and hence the extent to which we can independently determine the different elements of z. More generally, $\nu_{\max}/\nu_{\min}$ is a measure of the sensitivity or stability of our least-squares solutions to perturbations (e.g. in g). As we have already seen in this chapter, and will see again in Chapter 19 within the regression context, we can to a certain extent "control" B through the choice of variables, functional dependencies, and measurement points (which gives rise to the important field of "design of experiment(s)"); we can thus strive to control $\nu_{\max}/\nu_{\min}$ through good "independent" choices and thus ensure good prediction of z.

Example 17.3.4 Measurement Noise in Polynomial Fitting

Let us demonstrate the effect of perturbation in g — or the measurement error — in the context



(a) large perturbation



(b) small perturbation

Figure 17.6: The effect of data perturbation on the solution.

of polynomial fitting we considered earlier. As before, we assume that the output depends on the input quadratically according to

$$y(x) = -\frac{1}{2} + \frac{2}{3}x - \frac{1}{8}cx^2 \ ,$$

with $c = 1$. We construct clean data $g_0 \in \mathbb{R}^m$, $m = 7$, by evaluating y at

$$x_i = (i-1)/2, \quad i = 1, \ldots, m \ ,$$

and setting

$$g_{0,i} = y(x_i), \quad i = 1, \ldots, m \ .$$

Because g_0 precisely follows the quadratic function, $z_0 = (-1/2, 2/3, -1/8)$ satisfies the overdetermined system $Bz_0 = g_0$. Recall that B is the $m \times n$ Vandermonde matrix with the evaluation points $\{x_i\}$.

We then construct perturbed data g by adding random noise to g_0, i.e.

$$g_i = g_{0,i} + \epsilon_i, \quad i = 1, \ldots, m \ .$$

Then, we solve for the least-squares solution z^* of $Bz^* = g$.

The result of solving the polynomial fitting problem for two different perturbation levels is shown in Figure 17.6. For the large perturbation case, the perturbation in data and the error in the solution — both measured in 2-norm — are

$$\|g - g_0\| = 0.223 \quad \text{and} \quad \|z - z_0\| = 0.072 \ .$$

In contrast, the small perturbation case produces

$$\|g - g_0\| = 0.022 \quad \text{and} \quad \|z - z_0\| = 0.007 \ .$$

The results confirm that a smaller perturbation in data results in a smaller error in the solution.

We can also verify the error bounds. The minimum singular value of the Vandermonde matrix is

$$\nu_{\min}(B) = 0.627 \ .$$

Application of the (absolute) error bound to the large perturbation case yields

$$0.072 = \|z - z_0\| \le \frac{1}{\nu_{\min}(B)} \|g - g_0\| = 0.356 \ .$$

The error bound is clearly satisfied. The error bound for the small perturbation case is similarly satisfied.

·

We now prove the error bounds.

Proof. To establish the absolute error bound, we first note that the solution to the clean problem, z_0, and the solution to the perturbed problem, z^*, satisfy the normal equation, i.e.

$$B^{\mathrm{T}} B z_0 = B^{\mathrm{T}} g_0 \quad \text{and} \quad B^{\mathrm{T}} B z^* = B^{\mathrm{T}} g \ .$$

Taking the difference of the two equations

$$B^{\mathrm{T}} B (z^* - z_0) = B^{\mathrm{T}} (g - g_0) \ .$$

Now, we multiply both sides by $(z^* - z_0)^{\mathrm{T}}$ to obtain

$$\begin{aligned}
\text{(LHS)} &= (z^* - z_0)^{\mathrm{T}} B^{\mathrm{T}} B (z^* - z_0) = (B(z^* - z_0))^{\mathrm{T}} (B(z^* - z_0)) = \|B(z^* - z_0)\|^2 \\
\text{(RHS)} &= (z^* - z_0)^{\mathrm{T}} B^{\mathrm{T}} (g - g_0) = (B(z^* - z_0))^{\mathrm{T}} (g - g_0) \le \|B(z^* - z_0)\| \|g - g_0\| \ ,
\end{aligned}$$

where we have invoked the Cauchy-Schwarz inequality on the right-hand side. Thus, we have

$$\|B(z^* - z_0)\|^2 \le \|B(z^* - z_0)\| \|g - g_0\| \quad \Rightarrow \quad \|B(z^* - z_0)\| \le \|g - g_0\| \ .$$

We can bound the left-hand side from below using the definition of the minimum singular value

$$\nu_{\min}(B) \|z^* - z_0\| \le \|B(z^* - z_0)\| \ .$$

Thus, we have

$$\nu_{\min} \|z^* - z_0\| \le \|B(z^* - z_0)\| \le \|g - g_0\| \quad \Rightarrow \quad \|z^* - z_0\| \le \frac{1}{\nu_{\min}(B)} \|g - g_0\| \ ,$$

which is the desired absolute error bound.

To obtain the relative error bound, we first divide the absolute error bound by $\|z_0\|$ to obtain

$$\frac{\|z^* - z_0\|}{\|z_0\|} \le \frac{1}{\nu_{\min}(B)} \frac{\|g - g_0\|}{\|z_0\|} = \frac{1}{\nu_{\min}(B)} \frac{\|g - g_0\|}{\|g_0\|} \frac{\|g_0\|}{\|z_0\|} \ .$$

To bound the quotient $\|g_0\|/\|z_0\|$, we take the norm of both sides of $Bz_0 = g_0$ and invoke the definition of the maximum singular value, i.e.

$$\|g_0\| = \|Bz_0\| \le \nu_{\max} \|z_0\| \quad \Rightarrow \quad \frac{\|g_0\|}{\|z_0\|} \le \nu_{\max} \ .$$

Substituting the expression to the previous bound

$$\frac{\|z^* - z_0\|}{\|z_0\|} \le \frac{1}{\nu_{\min}(B)} \frac{\|g - g_0\|}{\|g_0\|} \frac{\|g_0\|}{\|z_0\|} \le \frac{\nu_{\max}(B)}{\nu_{\min}(B)} \frac{\|g - g_0\|}{\|g_0\|} \ ,$$

which is the desired relative error bound.

□

Proof (using singular value decomposition). We start with the singular value decomposition of matrix B,

$$B = U\Sigma V^{\mathrm{T}} \ ,$$

where U is an $m \times m$ unitary matrix, V is an $n \times n$ unitary matrix, and Σ is an $m \times n$ diagonal matrix. In particular, Σ consists of singular values of B and is of the form

$$\Sigma = \begin{pmatrix} \nu_1 & & & \\ & \nu_2 & & \\ & & \ddots & \\ & & & \nu_n \\ & & & \\ & & & \end{pmatrix} = \begin{pmatrix} \widehat{\Sigma} \\ 0 \end{pmatrix} .$$

The singular value decomposition exists for any matrix. The solution to the original problem is given by

$$Bz = g \quad \Rightarrow \quad U\Sigma V^{\mathrm{T}} z = g \quad \Rightarrow \quad \Sigma V^{\mathrm{T}} z = U^{\mathrm{T}} g \ .$$

The solution to the least-squares problem is

$$\begin{aligned} z^* &= \arg\min_z \|Bz - g\| = \arg\min_z \|U\Sigma V^{\mathrm{T}} z - g\| = \arg\min_z \|\Sigma V^{\mathrm{T}} z - U^{\mathrm{T}} g\| \\ &= V\left(\arg\min_{\tilde{z}} \|\Sigma\tilde{z} - \tilde{g}\|\right) \ , \end{aligned}$$

where the third equality follows from the fact that the action by an unitary matrix does not alter the 2-norm, and we have made the substitutions $\tilde{z} = V^{\mathrm{T}} z$ and $\tilde{g} = U^{\mathrm{T}} g$. We note that because Σ is diagonal, the 2-norm to be minimized is particularly simple,

$$\Sigma\tilde{z} - \tilde{g} = \Sigma = \begin{pmatrix} \nu_1 & & \\ & \ddots & \\ & & \nu_n \\ & & \\ & & \end{pmatrix} \begin{pmatrix} \tilde{z}_1 \\ \vdots \\ \tilde{z}_n \end{pmatrix} - \begin{pmatrix} \tilde{g}_1 \\ \vdots \\ \tilde{g}_n \\ \tilde{g}_{n+1} \\ \vdots \\ \tilde{g}_m \end{pmatrix} .$$

Note that choosing $\tilde{z}_1, \ldots, \tilde{z}_n$ only affects the first n component of the residual vector. Thus, we should pick $\tilde{z}_1, \ldots, \tilde{z}_n$ such that

$$\begin{pmatrix} \nu_1 & & \\ & \ddots & \\ & & \nu_n \end{pmatrix} \begin{pmatrix} \tilde{z}_1 \\ \vdots \\ \tilde{z}_n \end{pmatrix} = \begin{pmatrix} \tilde{g}_1 \\ \vdots \\ \tilde{g}_n \end{pmatrix} \quad \Rightarrow \quad \tilde{z}_i = \frac{\tilde{g}_i}{\nu_i}, \quad i = 1, \ldots, n \ .$$

By introducing a $n \times m$ restriction matrix that extracts the first n entries of $\tilde{g}$, we can concisely write the above as

$$\widehat{\Sigma}\tilde{z} = R\tilde{g} \quad \Rightarrow \quad \tilde{z} = \widehat{\Sigma}^{-1} R\tilde{g} \ ,$$

and the solution to the least-squares problem as

$$z^* = V\tilde{z}^* = V\widehat{\Sigma}^{-1}R\tilde{g} = V\widehat{\Sigma}^{-1}RU^{\mathrm{T}}g \ .$$

The absolute condition number bound is obtained by

$$\begin{aligned}\|z^* - z_0\| = \|V\widehat{\Sigma}^{-1}RU^{\mathrm{T}}(g - g_0)\| &= \frac{\|V\widehat{\Sigma}^{-1}RU^{\mathrm{T}}(g - g_0)\|}{\|g - g_0\|}\|g - g_0\| \\ &\le \left(\sup_{\delta g} \frac{\|V\widehat{\Sigma}^{-1}RU^{\mathrm{T}}\delta g\|}{\|\delta g\|}\right)\|g - g_0\| \ .\end{aligned}$$

The term in the parenthesis is bounded by noting that orthogonal transformations preserve the 2-norm and that the restriction operator does not increase the 2-norm, i.e.

$$\sup_{\delta g}\left(\frac{\|V\widehat{\Sigma}^{-1}RU^{\mathrm{T}}\delta g\|}{\|\delta g\|}\right) = \sup_{\delta\tilde{g}}\left(\frac{\|V\widehat{\Sigma}^{-1}R\delta\tilde{g}\|}{\|U\delta\tilde{g}\|}\right) = \sup_{\delta\tilde{g}}\left(\frac{\|\widehat{\Sigma}^{-1}R\delta\tilde{g}\|}{\|\delta\tilde{g}\|}\right) \le \frac{1}{\nu_{\min}(B)} \ .$$

Thus, we have the desired absolute error bound

$$\|z^* - z_0\| \le \frac{1}{\nu_{\min}(B)}\|g - g_0\| \ .$$

Now let us consider the relative error bound. We first note that

$$\frac{\|z^* - z_0\|}{\|z_0\|} = \frac{1}{\nu_{\min}(B)}\|g - g_0\|\frac{1}{\|z_0\|} = \frac{1}{\nu_{\min}(B)}\frac{\|g - g_0\|}{\|g_0\|}\frac{\|g_0\|}{\|z_0\|} \ .$$

The term $\|g_0\|/\|z_0\|$ can be bounded by expressing z_0 in terms of g using the explicit expression for the least-squares solution, i.e.

$$\frac{\|g_0\|}{\|z_0\|} = \frac{\|Bz_0\|}{\|z_0\|} = \frac{\|U\Sigma V^{\mathrm{T}}z_0\|}{\|z_0\|} \le \sup_z \frac{\|U\Sigma V^{\mathrm{T}}z\|}{\|z\|} = \sup_{\tilde{z}} \frac{\|U\Sigma\tilde{z}\|}{\|V\tilde{z}\|} = \sup_{\tilde{z}} \frac{\|\Sigma\tilde{z}\|}{\|\tilde{z}\|} = \nu_{\max}(B) \ .$$

Thus, we have the relative error bound

$$\frac{\|z^* - z_0\|}{\|z_0\|} \le \frac{\nu_{\max}(B)}{\nu_{\min}(B)}\frac{\|g - g_0\|}{\|g_0\|} \ .$$

This concludes the proof. □

Error Bounds with Respect to Reduction in Space, B

Let us now consider a scenario that illustrates the effect of *bias*. Again, we start with an overdetermined linear system,

$$B_0 z_0 = g \ ,$$

where $B_0 \in \mathbb{R}^{m\times n}$ with $m > n$. We assume that z_0 satisfies all m equations. We recall that, in the context of polynomial fitting, B_0 is of the form,

$$B_0 = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{pmatrix} ,$$

where m is the number of data points and n is the degree of polynomial. Now, suppose that we decide to use a p-th degree polynomial rather than the n-th degree polynomial, where $p < n$. In other words, we can partition B_0 into

$$B_0 = \left(\begin{array}{c|c} B_\mathrm{I} & B_\mathrm{II} \end{array} \right) = \left(\begin{array}{cccc|ccc} 1 & x_1^1 & \cdots & x_1^p & x_1^{p+1} & \cdots & x_1^n \\ 1 & x_2^1 & \cdots & x_2^p & x_2^{p+1} & \cdots & x_m^n \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 1 & x_m^1 & \cdots & x_m^p & x_m^{p+1} & \cdots & x_m^n \end{array} \right),$$

where $B_\mathrm{I} \in \mathbb{R}^{m\times(p+1)}$ and $B_\mathrm{II} \in \mathbb{R}^{m\times(n-p)}$. Then we can solve the least-squares problem resulting from the first partition, i.e.

$$B_\mathrm{I} z^* = g \ .$$

For convenience, let us also partition the solution to the original system z_0 into two parts corresponding to B_I and B_II, i.e.

$$z_0 = \begin{pmatrix} z_\mathrm{I} \\ z_\mathrm{II} \end{pmatrix},$$

where $z_\mathrm{I} \in \mathbb{R}^{p+1}$ and $z_\mathrm{II} \in \mathbb{R}^{n-p}$. The question is, how close are the coefficients $z^* = (z_1, \ldots, z_{p-1})$ of the reduced system compared to the coefficients of the first partition of the original system, z_I?

We can in fact bound the error in the solution $\|z^* - z_\mathrm{I}\|$ in terms of the "missing space" B_II. In particular, the absolute error bound is given by

$$\|z^* - z_\mathrm{I}\| \le \frac{1}{\nu_\mathrm{min}(B_\mathrm{I})} \|B_\mathrm{II} z_\mathrm{II}\|$$

and the relative error bound is given by

$$\frac{\|z^* - z_\mathrm{I}\|}{\|z_\mathrm{I}\|} \le \frac{\nu_\mathrm{max}(B_\mathrm{I})}{\nu_\mathrm{min}(B_\mathrm{I})} \frac{\|B_\mathrm{II} z_\mathrm{II}\|}{\|g - B_\mathrm{II} z_\mathrm{II}\|} \ ,$$

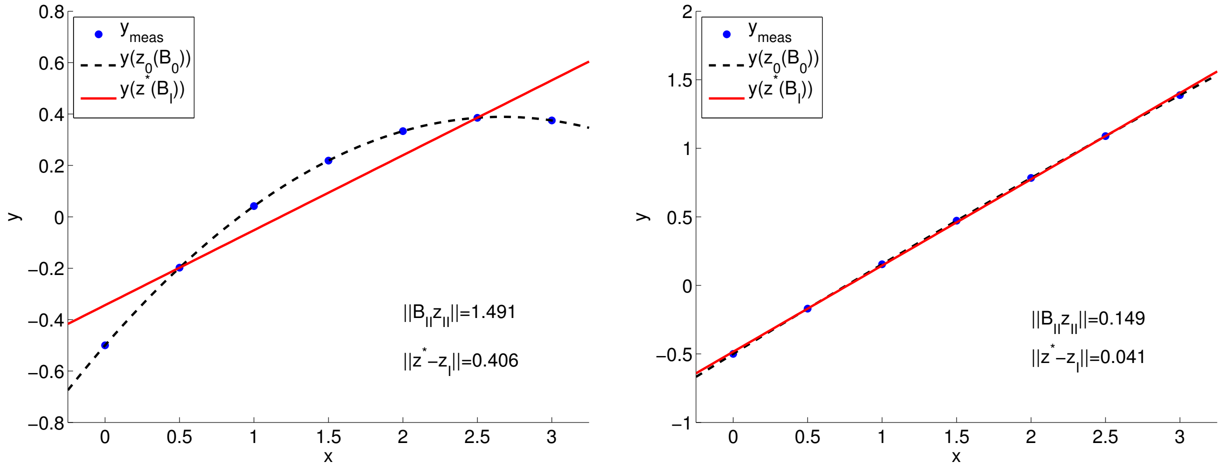
where $\nu_\mathrm{min}(B_\mathrm{I})$ and $\nu_\mathrm{max}(B_\mathrm{I})$ are the minimum and maximum singular values of B_I.

Example 17.3.5 Bias Effect in Polynomial Fitting
Let us demonstrate the effect of reduced solution space — or the bias effect — in the context of polynomial fitting. As before, the output depends on the input quadratically according to

$$y(x) = -\frac{1}{2} + \frac{2}{3}x - \frac{1}{8}cx^2 \ .$$

Recall that c controls the strength of quadratic dependence. The data g is generated by evaluating y at $x_i = (i-1)/2$ and setting $g_i = y(x_i)$ for $i = 1, \ldots, m$, with $m = 7$. We partition our Vandermonde matrix for the quadratic model B_0 into that for the affine model B_I and the quadratic only part B_II, i.e.

$$B_0 = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{pmatrix} = \left(\begin{array}{cc|c} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{array} \right) = \left(\begin{array}{c|c} B_\mathrm{I} & B_\mathrm{II} \end{array} \right).$$



(a) $c = 1$ (b) $c = 1/10$

Figure 17.7: The effect of reduction in space on the solution.

As before, because the underlying data is quadratic, we can exactly match the function using the full space B_0, i.e., $B_0 z_0 = g$.

Now, we restrict ourselves to affine functions, and find the least-squares solution z^* to $B_{\rm I} z^* = g$. We would like to quantify the difference in the first two coefficients of the full model z_I and the coefficients of the reduced model z^*.

Figure 17.7 shows the result of fitting an affine function to the quadratic function for $c = 1$ and $c = 1/10$. For the $c = 1$ case, with the strong quadratic dependence, the effect of the missing quadratic function is

$$\|B_{\rm II} z_{\rm II}\| = 1.491 \ .$$

This results in a relative large solution error of

$$\|z^* - z_{\rm I}\| = 0.406 \ .$$

We also note that, with the minimum singular value of $\nu_{\min}(B_{\rm I}) = 1.323$, the (absolute) error bound is satisfied as

$$0.406 = \|z^* - z_{\rm I}\| \le \frac{1}{\nu_{\min}(B_{\rm I})} \|B_{\rm II} z_{II}\| = 1.1267 \ .$$

In fact, the bound in this particular case is reasonable sharp.

Recall that the least-squares solution z^* minimizes the ℓ_2 residual

$$0.286 = \|B_{\rm I} z^* - g\| \le \|B_{\rm I} z - g\|, \quad \forall\, z \in \mathbb{R}^2 \ ,$$

and the residual is in particular smaller than that for the truncated solution

$$\|B_{\rm I} z_{\rm I} - g\| = 1.491 \ .$$

However, the error for the least-squares solution — in terms of predicting the first two coefficients of the underlying polynomial — is larger than that of the truncated solution (which of course is zero). This case demonstrates that minimizing the residual does not necessarily minimize the error.

For the $c = 1/10$ with a weaker quadratic dependence, the effect of missing the quadratic function is

$$\|B_{\mathrm{II}} z_{\mathrm{II}}\| = 0.149$$

and the error in the solution is accordingly smaller as

$$\|z^* - z_{\mathrm{I}}\| = 0.041 \ .$$

This agrees with our intuition. If the underlying data exhibits a weak quadratic dependence, then we can represent the data well using an affine function, i.e., $\|B_{\mathrm{II}} z_{\mathrm{II}}\|$ is small. Then, the (absolute) error bound suggests that the small residual results in a small error.

·

We now prove the error bound.

Proof. We rearrange the original system as

$$B_0 z_0 = B_{\mathrm{I}} z_{\mathrm{I}} + B_{\mathrm{II}} z_{\mathrm{II}} = g \quad \Rightarrow \quad B_{\mathrm{I}} z_{\mathrm{I}} = g - B_{\mathrm{II}} z_{\mathrm{II}} \ .$$

By our assumption, there is a solution z_{I} that satisfies the $m \times (p+1)$ overdetermined system

$$B_{\mathrm{I}} z_{\mathrm{I}} = g - B_{\mathrm{II}} z_{\mathrm{II}} \ .$$

The reduced system,

$$B_{\mathrm{I}} z^* = g \ ,$$

does not have a solution in general, so is solved in the least-squares sense. These two cases are identical to the unperturbed and perturbed right-hand side cases considered the previous subsection. In particular, the perturbation in the right-hand side is

$$\|g - (g - B_{\mathrm{II}} z_{\mathrm{II}})\| = \|B_{\mathrm{II}} z_{\mathrm{II}}\| \ ,$$

and the perturbation in the solution is $\|z^* - z_{\mathrm{I}}\|$. Substitution of the perturbations into the absolute and relative error bounds established in the previous subsection yields the desired results. □

End Advanced Material

Chapter 18

Matlab Linear Algebra (Briefly)

18.1 Matrix Multiplication (and Addition)

We can think of a hypothetical computer (or scripting) language in which we must declare a "tableau" of m by n numbers to be either a double-index array or a matrix; we also introduce a *hypothetical* "multiplication" operator `#`. (Note that `#` is not an actual MATLAB multiplication character/operator — it is introduced here solely for temporary pedagogical purposes.) In the case in which we (say) declare `A` and `B` to be *arrays* then the product `C = A # B` would be automatically interpreted as element-by-element multiplication: both `A` and `B` must be of the same size $m \times n$ for the operation to make sense, and the result `C` would of course also be of size $m \times n$. In the case in which we declare `A` and `B` to be *matrices* then the product `A # B` would be automatically interpreted as matrix-matrix multiplication: if `A` is m_1 by n_1 and `B` is m_2 by n_2 then n_1 must equal m_2 for the operation to make sense and the product `C = A # B` would be of dimensions $m_1 \times n_2$. This is a very simple example of object-oriented programming in which an operation, say multiplication, is defined — in potentially different ways — for different classes of objects (in our case here, arrays and matrices) — but we could also envision an extension to functions and other entities as well. This model for programming languages and abstraction can be very powerful for a variety of reasons.

However, in some cases such abstraction can arguably be more of a burden than a blessing. For example, in MATLAB we often wish to re-interpret arrays as matrices or matrices as arrays on many different occasions even with a single code or application. To avoid conversion issues between these two classes, MATLAB prefers to treat arrays and matrices as (effectively) a single class and then to distinguish the two options for multiplication through special operators. In particular, as we already know, element-by-element multiplication of two arrays is effected by the `.*` operator — `C = A.*B` forms `C` as the element-by-element product of `A` and `B`; matrix-matrix multiplication (in the sense of linear algebra) is then effected simply by `*` — `C = A*B` forms `C` as the matrix product of `A` and `B`. In fact, the emphasis in MATLAB at least historically is on linear algebra, and thus matrix multiplication is in some sense the default; element-by-element operations are the "special case" and require the "dotted operators."

In principle, we should also need to distinguish element-by-element addition and subtraction as `.+` and `.-` from matrix-matrix addition and subtraction as `+` and `-`. However, element-by-element addition and subtraction and matrix-matrix addition and subtraction are identical — both in terms of the requirements on the operands and on the result of the operation — and hence it suffices to introduce only a single addition and subtraction operator, `+` and `-`, respectively. (In particular, note that there *are no* operators `.+` and `.-` in Matlab.) In a similar fashion, we need only a single transpose operator, `'`, which is directly applicable to both arrays and matrices.[2]

It thus follows that the matrix-matrix addition, subtraction, multiplication, and transpose are effected in Matlab in essentially the same way as we would write such operations in the linear algebra context: in the addition or subtraction of two vectors x and y, the $x+y$ and $x-y$ of linear algebra becomes `x + y` and `x - y` in Matlab; in the multiplication of two matrices A and B, the AB of linear algebra becomes `A*B` in Matlab; and in the transpose of a matrix (or vector) M, the M^{T} of linear algebra becomes `M'` in Matlab.

Of course, you could also always implement these matrix operations in Matlab "explicitly" with `for` loops and appropriate indexing: for example, $z = x + y$ could be implemented as

```
z = 0.*x; % initialize z to be same size as x
for i = 1:length(x)
    z(i) = x(i) + y(i);
end
```

however this leads to code which is both much less efficient and also much longer and indeed much less readable (and hence de-buggable). (Note also that the above does not yet contain any check on dimensions or error flags.) We have already discussed the power of function abstraction. In the case of these very ubiquitous functions — standard array and matrix manipulations — Matlab provides the further convenience of special characters and hence very simple syntax. (Note that as these special characters are, as always, just an easy way to invoke the underlying Matlab operator or function: for example, the element-by-element multiplication operation `A.*B` can also be written (but less conveniently) as `times(A,B)`, and the matrix-matrix multiplication `A*B` can also be written as `mtimes(A,B)`.)

We close with a simple example to again illustrate the differences between array and matrix operations. We introduce two column vectors $x = (1\ \ 1)^{\mathrm{T}}$ and $y = (2\ \ 2)^{\mathrm{T}}$ which in Matlab we express as `x = [1; 1]` and `y = [2; 2]`. (Note the distinction: parentheses for vectors and matrices in the linear algebra context, brackets for vectors and matrices in Matlab; parentheses in Matlab are used for indexing and function calls, not to define a vector or matrix.) We may then perform the linear algebra operation of inner product, $\alpha = x^{\mathrm{T}}y$, in two fashions: with element-by-element multiplication (and hence `times`) as `alpha = sum(x.*y)`; with matrix multiplication (and hence `mtimes`) as `alpha_too = x'*y`.

18.2 The Matlab Inverse Function: `inv`

This section is short. Given a non-singular square matrix A, `A` in Matlab, we can find A^{-1} in Matlab as `inv(A)` (which of course may also be assigned to a new matrix, as in `Ainv = inv(A)`). To within round-off error we can anticipate that `inv(A)*A` and `A*inv(A)` should both evaluate to the identity matrix. (In finite-precision arithmetic, of course we will not obtain exactly an identity

[2] In fact, the array transpose and the matrix transpose are different: the array transpose is given by `.'` and switches rows and columns; the matrix transpose is given by `'` and effects the conjugate, or Hermitian transpose, in which $A^{\mathrm{H}}_{ij} = \overline{A}_{ij}$ and $\overline{}$ refers to the complex conjugate. The Hermitian transpose (superscript H) is the correct generalization from real matrices to complex matrices in order to ensure that all our linear algebra concepts (e.g., norm) extend correctly to the complex case. We will encounter complex variables in Unit IV related to eigenvalues. Note that for real matrices we can use either `'` (array) or `.'` (matrix) to effect the (Hermitian) matrix transpose since the complex conjugate of a real number is simply the real number.

matrix; however, for "well-conditioned" matrices we should obtain a matrix which differs from the identity by roughly machine precision.)

As we have already discussed, and as will be demonstrated in Unit V, the `inv` operation is quite expensive, and in most cases there are better ways to achieve any desired end than through a call to `inv`. Nevertheless for small systems, and in cases in which we do explicitly require the inverse for some reason, the `inv` function is very convenient.

18.3 Solution of Linear Systems: Matlab Backslash

We now consider a system of n linear equations in n unknowns: $Ax = b$. We presume that the matrix A is non-singular such that there is indeed a solution, and in fact a unique solution, to this system of equations. We know that we may write this solution if we wish as $x = A^{-1}b$. There are two ways in which we find x in MATLAB. Actually, more than two ways: we restrict attention to the most obvious (and worst) and then the best.

As our first option we can simply write `x = inv(A)*b`. However, except for small systems, this will be unnecessarily expensive. This "inverse" approach is in particular very wasteful in the case in which the matrix A is quite sparse — with many zeros — a situation that arises very (very) often in the context of mechanical engineering and physical modeling more generally. We discuss the root cause of this inefficiency in Unit V.

As our second option we can invoke the MATLAB "backslash" operator \ (corresponding to the function `mldivide`) as follows: `x = A \ b`. This backslash operator is essentially a collection of related (direct) solution options from which MATLAB will choose the most appropriate based on the form of A; these options are all related to the "LU" decomposition of the matrix A (followed by forward and back substitution), as we will discuss in greater detail in Unit V. Note that these LU approaches do *not* form the inverse of A but rather directly attack the problem of solution of the linear system. The MATLAB backslash operator is very efficient not only due to the algorithm chosen but also due to the careful and highly optimized implementation.

18.4 Solution of (Linear) Least-Squares Problems

In Chapter 17 we considered the solution of least squares problems: given $B \in \mathbb{R}^{m\times n}$ and $g \in \mathbb{R}^m$ find $z^* \in \mathbb{R}^n$ which minimizes $\|Bz - g\|^2$ over all $z \in \mathbb{R}^n$. We showed that z^* satisfies the normal equations, $Nz^* = B^{\mathrm{T}}g$, where $N \equiv B^{\mathrm{T}}B$. There are (at least) three ways we can implement this least-squares solution in MATLAB.

The first, and worst, is to write `zstar = inv(B'*B)*(B'*g)`. The second, and slightly better, is to take advantage of our backslash operator to write `zstar_too = (B'*B)\(B'*g)`. However, both of the approaches are less than numerically stable (and more generally we should avoid taking powers of matrices since this just exacerbates any intrinsic conditioning or "sensitivity" issues). The third option, and by far the best, is to write `zstar_best = B\g`. Here the backslash operator "recognizes" that B is not a square matrix and automatically pursues a least-squares solution based on the stable and efficient QR decomposition discussed in Chapter 17.

Finally, we shall see in Chapter 19 on statistical regression that some elements of the matrix $(B^{\mathrm{T}}B)^{-1}$ will be required to construct confidence intervals. Although it is possible to efficiently calculate certain select elements of this inverse matrix without construction of the full inverse matrix, in fact our systems shall be relatively small and hence `inv(B'*B)` is quite inexpensive. (Nevertheless, the solution of the least-squares problem is still best implemented as `zstar_best = B \ g`, even if we subsequently form the inverse `inv(B'*B)` for purposes of confidence intervals.)

Chapter 19

Regression: Statistical Inference

19.1 Simplest Case

Let us first consider a "simple" case of regression, where we restrict ourselves to one independent variable and linear basis functions.

19.1.1 Friction Coefficient Determination Problem Revisited

Recall the friction coefficient determination problem we considered in Section 17.1. We have seen that in presence of m perfect measurements, we can find a μ_s that satisfies m equations

$$F^{\text{max, meas}}_{\text{f, static}\,i} = \mu_\mathrm{s}\, F_{\text{normal, applied}\,i}, \quad i = 1, \dots, m\ .$$

In other words, we can use any one of the m-measurements and solve for μ_s according to

$$\mu_{\mathrm{s},i} = \frac{F^{\text{max, meas}}_{\text{f, static}\,i}}{F_{\text{normal, applied}\,i}}\ ,$$

and all $\mu_{\mathrm{s},i}$, $i = 1, \dots, m$, will be identical and agree with the true value μ_s.

Unfortunately, real measurements are corrupted by noise. In particular, it is unlikely that we can find a single coefficient that satisfies all m measurement pairs. In other words, μ_s computed using the m different pairs are likely not to be identical. A more suitable model for static friction that incorporates the notion of measurement noise is

$$F^{\text{max, meas}}_{\text{f, static}} = \mu_\mathrm{s}\, F_{\text{normal, applied}} + \epsilon\ .$$

The noise associated with each measurement is obviously unknown (otherwise we could correct the measurements), so the equation in the current form is not very useful. However, if we make some weak assumptions on the behavior of the noise, we can in fact:

(a) infer the value of μ_s with associated confidence,

(b) estimate the noise level,

(c) confirm that our model is correct (more precisely, not incorrect),

(d) and detect significant unmodeled effects.

This is the idea behind *regression* — a framework for deducing the relationship between a set of inputs (e.g. $F_{\text{normal,applied}}$) and the outputs (e.g. $F_{\text{f, static}}^{\text{max, meas}}$) in the presence of noise. The regression framework consists of two steps: (i) construction of an appropriate response model, and (ii) identification of the model parameters based on data. We will now develop procedures for carrying out these tasks.

19.1.2 Response Model

Let us describe the relationship between the input x and output Y by

$$Y(x) = Y_{\text{model}}(x;\beta) + \epsilon(x) \ , \qquad (19.1)$$

where

(a) x is the independent variable, which is deterministic.

(b) Y is the measured quantity (i.e., data), which in general is noisy. Because the noise is assumed to be random, Y is a random variable.

(c) Y_{model} is the predictive model with no noise. In linear regression, Y_{model} is a linear function of the model parameter β by definition. In addition, we assume here that the model is an affine function of x, i.e.

$$Y_{\text{model}}(x;\beta) = \beta_0 + \beta_1 x \ ,$$

where β_0 and β_1 are the components of the model parameter β. We will relax this affine-in-x assumption in the next section and consider more general functional dependencies as well as additional independent variables.

(d) ϵ is the noise, which is a random variable.

Our objective is to infer the model parameter β that best describes the behavior of the measured quantity and to build a model $Y_{\text{model}}(\cdot;\beta)$ that can be used to predict the output for a new x. (Note that in some cases, the estimation of the parameter itself may be of interest, e.g. deducing the friction coefficient. In other cases, the primary interest may be to predict the output using the model, e.g. predicting the frictional force for a given normal force. In the second case, the parameter estimation itself is simply a means to the end.)

As considered in Section 17.1, we assume that our model is *unbiased*. That is, in the absence of noise ($\epsilon = 0$), our underlying input-output relationship can be perfectly described by

$$y(x) = Y_{\text{model}}(x;\beta^{\text{true}})$$

for some "true" parameter β^{true}. In other words, our model includes the true functional dependency (but may include more generality than is actually needed). We observed in Section 17.1 that if the model is unbiased *and* measurements are noise-free, then we can deduce the true parameter, β^{true}, using a number of data points equal to or greater than the degrees of freedom of the model ($m \geq n$).

In this chapter, while we still assume that the model is unbiased[1], we relax the noise-free assumption. Our measurement (i.e., data) is now of the form

$$Y(x) = Y_{\text{model}}(x;\beta^{\text{true}}) + \epsilon(x) \ ,$$

where ϵ is the noise. In order to estimate the true parameter, β^{true}, with confidence, we make three important assumptions about the behavior of the noise. These assumptions allow us to make quantitative (statistical) claims about the quality of our regression.

[1]In Section 19.2.4, we will consider effects of bias (or undermodelling) in one of the examples.

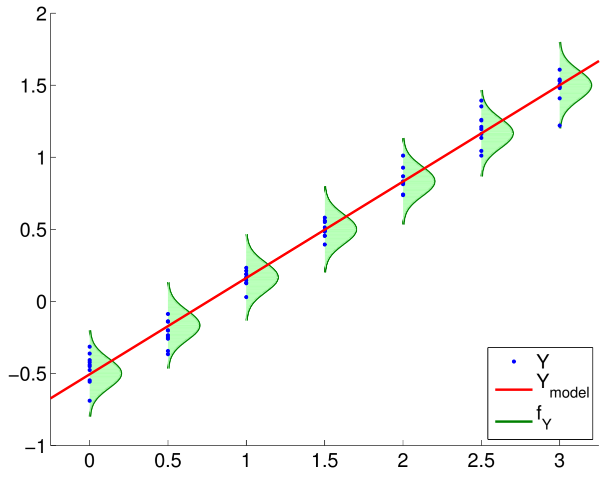


Figure 19.1: Illustration of the regression process.

(i) **Normality (N1)**: We assume the noise is a normally distributed with zero-mean, i.e., $\epsilon(x) \sim \mathcal{N}(0, \sigma^2(x))$. Thus, the noise $\epsilon(x)$ is described by a single parameter $\sigma^2(x)$.

(ii) **Homoscedasticity (N2)**: We assume that ϵ is not a function of x in the sense that the distribution of ϵ, in particular σ^2, does not depend on x.

(iii) **Independence (N3)**: We assume that $\epsilon(x_1)$ and $\epsilon(x_2)$ are independent and hence uncorrelated.

We will refer to these three assumptions as (N1), (N2), and (N3) throughout the rest of the chapter. These assumptions imply that $\epsilon(x) = \epsilon = \mathcal{N}(0, \sigma^2)$, where σ^2 is the single parameter for *all* instances of x.

Note that because

$$Y(x) = Y_{\text{model}}(x; \beta) + \epsilon = \beta_0 + \beta_1 x + \epsilon$$

and $\epsilon \sim \mathcal{N}(0, \sigma^2)$, the deterministic model $Y_{\text{model}}(x; \beta)$ simply shifts the mean of the normal distribution. Thus, the measurement is a random variable with the distribution

$$Y(x) \sim \mathcal{N}(Y_{\text{model}}(x; \beta), \sigma^2) = \mathcal{N}(\beta_0 + \beta_1 x, \sigma^2) \ .$$

In other words, when we perform a measurement at some point x_i, we are in theory drawing a random variable from the distribution $\mathcal{N}(\beta_0 + \beta_1 x_i, \sigma^2)$. We may think of $Y(x)$ as a random variable (with mean) parameterized by x, or we may think of $Y(x)$ as a random function (often denoted a random process).

A typical regression process is illustrated in Figure 19.1. The model Y_{model} is a linear function of the form $\beta_0 + \beta_1 x$. The probability density functions of Y, f_Y, shows that the error is normally distributed (N1) and that the variance does not change with x (N2). The realizations of Y sampled for $x = 0.0, 0.5, 1.0, \ldots, 3.0$ confirms that it is unlikely for realizations to fall outside of the 3σ bounds plotted. (Recall that 99.7% of the samples falls within the 3σ bounds for a normal distribution.)

Figure 19.1 suggests that the likely outcome of Y depends on our independent variable x in a linear manner. This does not mean that Y is a function of x only. In particular, the outcome of an experiment is in general a function of many independent variables,

$$x = \begin{pmatrix} x_{(1)} & x_{(2)} & \cdots & x_{(k)} \end{pmatrix} .$$

But, in constructing our model, we assume that the outcome only strongly depends on the behavior of $x = x_{(1)}$, and the net effect of the other variables $\begin{pmatrix} x_{(2)} & \cdots & x_{(k)} \end{pmatrix}$ can be modeled as random through ϵ. In other words, the underlying process that governs the input-output relationship may be completely deterministic if we are given k variables that provides the full description of the system, i.e.

$$y(x_{(1)}, x_{(2)}, \ldots, x_{(k)}) = f(x_{(1)}, x_{(2)}, \ldots, x_{(k)}) \ .$$

However, it is unlikely that we have the full knowledge of functional dependencies as well as the state of the system.

Knowing that the deterministic prediction of the output is intractable, we resort to understanding the functional dependency of the most significant variable, say $x_{(1)}$. If we know that the dependency of y on $x_{(1)}$ is most dominantly affine (say based on a physical law), then we can split our (intractable) functional dependency into

$$y(x_{(1)}, x_{(2)}, \ldots, x_{(k)}) = \beta_0 + \beta_1 x_{(1)} + g(x_{(1)}, x_{(2)}, \ldots, x_{(k)}) \ .$$

Here $g(x_{(1)}, x_{(2)}, \ldots, x_{(k)})$ includes both the unmodeled system behavior and the unmodeled process that leads to measurement errors. At this point, we assume the effect of $(x_{(2)}, \ldots, x_{(k)})$ on y and the weak effect of $x_{(1)}$ on y through g can be lumped into a zero-mean random variable ϵ, i.e.

$$Y(x_{(1)}; \beta) = \beta_0 + \beta_1 x_{(1)} + \epsilon \ .$$

At some level this equation is almost guaranteed to be *wrong*.

First, there will be some bias: here bias refers to a deviation of the mean of $Y(x)$ from $\beta_0 + \beta_1 x_{(1)}$ — which of course can not be represented by ϵ which is assumed zero mean. Second, our model for the noise (e.g., (N1), (N2), (N3)) — indeed, any model for noise — is certainly not perfect. However, if the bias is small, and the deviations of the noise from our assumptions (N1), (N2), and (N3) are small, our procedures typically provide good answers. Hence we must always question whether the response model Y_{model} is correct, in the sense that it includes the correct model. Furthermore, the assumptions (N1), (N2), and (N3) do not apply to all physical processes and should be treated with skepticism.

We also note that the appropriate number of independent variables that are explicitly modeled, without being lumped into the random variable, depends on the system. (In the next section, we will treat the case in which we must consider the functional dependencies on more than one independent variable.) Let us solidify the idea using a very simple example of multiple coin flips in which in fact we need not consider *any* independent variables.

Example 19.1.1 Functional dependencies in coin flips
Let us say the system is 100 fair coin flips and Y is the total number of heads. The outcome of each coin flip, which affects the output Y, is a function of many variables: the mass of the coin, the moment of inertia of the coin, initial launch velocity, initial angular momentum, elasticity of the surface, density of the air, etc. If we had a complete description of the environment, then the outcome of each coin flip is deterministic, governed by Euler's equations (for rigid body dynamics), the Navier-Stokes equations (for air dynamics), etc. We see this deterministic approach renders our simulation intractable — both in terms of the number of states and the functional dependencies — even for something as simple as coin flips.

Thus, we take an alternative approach and lump some of the functional dependencies into a random variable. From Chapter 9, we know that Y will have a binomial distribution $\mathcal{B}(n = 100, \theta = 1/2)$. The mean and the variance of Y are

$$E[Y] = n\theta = 50 \quad \text{and} \quad E[(Y - \mu_Y)^2] = n\theta(1-\theta) = 25 \ .$$

In fact, by the central limit theorem, we know that Y can be approximated by

$$Y \sim \mathcal{N}(50, 25) \ .$$

The fact that Y can be modeled as $\mathcal{N}(50, 25)$ without any explicit dependence on any of the many independent variables we cited earlier does not mean that Y does not depend on the variables. It only means that the cumulative effect of the all independent variables on Y can be modeled as a zero-mean normal random variable. This can perhaps be motivated more generally by the central limit theorem, which heuristically justifies the treatment of many small random effects as normal noise.

19.1.3 Parameter Estimation

We now perform m experiments, each of which is characterized by the independent variable x_i. Each experiment described by x_i results in a measurement Y_i, and we collect m variable-measurement pairs,

$$(x_i, Y_i), \quad i = 1, \dots, m \ .$$

In general, the value of the independent variables x_i can be repeated. We assume that our measurements satisfy

$$Y_i = Y_{\text{model}}(x_i; \beta) + \epsilon_i = \beta_0 + \beta_1 x_i + \epsilon_i \ .$$

From the experiments, we wish to estimate the true parameter $\beta^{\text{true}} = (\beta_0^{\text{true}}, \beta_1^{\text{true}})$ without the precise knowledge of ϵ (which is described by σ). In fact we will estimate β^{true} and σ by $\hat{\beta}$ and $\hat{\sigma}$, respectively.

It turns out, from our assumptions (N1), (N2), and (N3), that the *maximum likelihood estimator* (MLE) for β — the most likely value for the parameter given the measurements (x_i, Y_i), $i = 1, \dots, m$ — is precisely our least squares fit, i.e., $\hat{\beta} = \beta^*$. In other words, if we form

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \quad \text{and} \quad Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{pmatrix} ,$$

then the MLE, $\hat{\beta}$, satisfies

$$\|X\hat{\beta} - Y\|_2 < \|X\beta - Y\|_2, \quad \forall \, \beta \neq \hat{\beta} \ .$$

Equivalently, $\hat{\beta}$ satisfies the normal equation

$$(X^{\mathrm{T}} X)\hat{\beta} = X^{\mathrm{T}} Y \ .$$

We provide the proof.

Proof. We show that the least squares solution is the maximum likelihood estimator (MLE) for β. Recall that we consider each measurement as $Y_i = \mathcal{N}(\beta_0 + \beta_1 x_i, \sigma^2) = \mathcal{N}(X_{i\cdot}\beta, \sigma^2)$. Noting the noise is independent, the m measurement collectively defines a joint distribution,

$$Y = \mathcal{N}(X\beta, \Sigma) \ ,$$

where Σ is the diagonal covariance matrix $\Sigma = \operatorname{diag}(\sigma^2, \ldots, \sigma^2)$. To find the MLE, we first form the conditional probability density of Y assuming β is given, i.e.

$$f_{Y|\mathcal{B}}(y|\beta) = \frac{1}{(2\pi)^{m/1}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(y - X\beta)^{\mathrm{T}}\Sigma^{-1}(y - X\beta)\right),$$

which can be viewed as a likelihood function if we now fix y and let β vary — $\beta|y$ rather than $y|\beta$. The MLE — the β that maximizes the likelihood of measurements $\{y_i\}_{i=1}^m$ — is then

$$\hat{\beta} = \arg\max_{\beta\in\mathbb{R}^2} f_{Y|\mathcal{B}}(y|\beta) = \arg\max_{\beta\in\mathbb{R}^2} \frac{1}{(2\pi)^{m/1}|\Sigma|^{1/2}} \exp\left(-\underbrace{\frac{1}{2}(y - X\beta)^{\mathrm{T}}\Sigma^{-1}(y - X\beta)}_{J}\right).$$

The maximum is obtained when J is minimized. Thus,

$$\hat{\beta} = \arg\min_{\beta\in\mathbb{R}^2} J(\beta) = \arg\min_{\beta\in\mathbb{R}^2} \frac{1}{2}(y - X\beta)^{\mathrm{T}}\Sigma^{-1}(y - X\beta) \ .$$

Recalling the form of Σ, we can simplify the expression to

$$\begin{aligned}\hat{\beta} &= \arg\min_{\beta\in\mathbb{R}^2} \frac{1}{2\sigma^2}(y - X\beta)^{\mathrm{T}}(y - X\beta) = \arg\min_{\beta\in\mathbb{R}^2}(y - X\beta)^{\mathrm{T}}(y - X\beta) \\ &= \arg\min_{\beta\in\mathbb{R}^2} \|y - X\beta\|^2 \ .\end{aligned}$$

This is precisely the least squares problem. Thus, the solution to the least squares problem $X\beta = y$ is the MLE. □

Having estimated the unknown parameter β^{true} by $\hat{\beta}$, let us now estimate the noise ϵ characterized by the unknown σ (which we may think of as σ^{true}). Our estimator for σ, $\hat{\sigma}$, is

$$\hat{\sigma} = \left(\frac{1}{m-2}\|Y - X\hat{\beta}\|^2\right)^{1/2} \ .$$

Note that $\|Y - X\hat{\beta}\|$ is just the root mean square of the residual as motivated by the least squares approach earlier. The normalization factor, $1/(m-2)$, comes from the fact that there are m measurement points and two parameters to be fit. If $m = 2$, then all the data goes to fitting the parameters $\{\beta_0, \beta_1\}$ — two points determine a line — and none is left over to estimate the error; thus, in this case, we cannot estimate the error. Note that

$$(X\hat{\beta})_i = Y_{\text{model}}(x_i;\beta)|_{\beta=\hat{\beta}} \equiv \widehat{Y}_i$$

is our response model evaluated at the parameter $\beta = \hat{\beta}$; we may thus write

$$\hat{\sigma} = \left(\frac{1}{m-2}\,\|Y - \widehat{Y}\|^2\right)^{1/2} \ .$$

In some sense, $\hat{\beta}$ minimizes the misfit and what is left is attributed to noise $\hat{\sigma}$ (per our model). *Note that we use the data at all points, $x_1, \ldots, x_m$, to obtain an estimate of our single parameter*, σ; this is due to our homoscedasticity assumption (N2), which assumes that ϵ (and hence σ) is independent of x.

We also note that the least squares estimate preserves the mean of the measurements in the sense that

$$\overline{Y} \equiv \frac{1}{m}\sum_{i=1}^{m} Y_i = \frac{1}{m}\sum_{i=1}^{m} \widehat{Y}_i \equiv \overline{\widehat{Y}} \ .$$

Proof. The preservation of the mean is a direct consequence of the estimator $\hat{\beta}$ satisfying the normal equation. Recall, $\hat{\beta}$ satisfies

$$X^{\mathrm{T}} X \hat{\beta} = X^{\mathrm{T}} Y \ .$$

Because $\widehat{Y} = X\hat{\beta}$, we can write this as

$$X^{\mathrm{T}} \widehat{Y} = X^{\mathrm{T}} Y \ .$$

Recalling the "row" interpretation of matrix-vector product and noting that the column of X is all ones, the first component of the left-hand side is

$$(X^{\mathrm{T}}\widehat{Y})_1 = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} \widehat{Y}_1 \\ \vdots \\ \widehat{Y}_m \end{pmatrix} = \sum_{i=1}^{m} \widehat{Y}_i \ .$$

Similarly, the first component of the right-hand side is

$$(X^{\mathrm{T}}Y)_1 = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} Y_1 \\ \vdots \\ Y_m \end{pmatrix} = \sum_{i=1}^{m} Y_i \ .$$

Thus, we have

$$(X^{\mathrm{T}}\widehat{Y})_1 = (X^{\mathrm{T}}Y)_1 \quad \Rightarrow \quad \sum_{i=1}^{m} \widehat{Y}_i = \sum_{i=1}^{m} Y_i \ ,$$

which proves that the model preserves the mean. □

19.1.4 Confidence Intervals

We consider two sets of confidence intervals. The first set of confidence intervals, which we refer to as individual confidence intervals, are the intervals associated with each individual parameter. The second set of confidence intervals, which we refer to as joint confidence intervals, are the intervals associated with the joint behavior of the parameters.

Individual Confidence Intervals

Let us introduce an estimate for the covariance of $\hat{\beta}$,

$$\widehat{\Sigma} \equiv \hat{\sigma}^2 (X^{\mathrm{T}} X)^{-1} \ .$$

For our case with two parameters, the covariance matrix is 2×2. From our estimate of the covariance, we can construct the confidence interval for β_0 as

$$I_0 \equiv \left[\hat{\beta}_0 - t_{\gamma,m-2}\sqrt{\widehat{\Sigma}_{11}}, \hat{\beta}_0 + t_{\gamma,m-2}\sqrt{\widehat{\Sigma}_{11}} \right] \ ,$$

and the confidence interval for β_1 as

$$I_1 \equiv \left[\hat{\beta}_1 - t_{\gamma,m-2}\sqrt{\widehat{\Sigma}_{22}}, \hat{\beta}_1 + t_{\gamma,m-2}\sqrt{\widehat{\Sigma}_{22}} \right] \ .$$

The coefficient $t_{\gamma,m-2}$ depends on the confidence level, γ, and the degrees of freedom, $m-2$. Note that the Half Length of the confidence intervals for β_0 and β_1 are equal to $t_{\gamma,m-2}\sqrt{\widehat{\Sigma}_{11}}$ and $t_{\gamma,m-2}\sqrt{\widehat{\Sigma}_{22}}$, respectively.

The confidence interval I_0 is an interval such that the probability of the parameter β_0^{true} taking on a value within the interval is equal to the confidence level γ, i.e.

$$P(\beta_0^{\text{true}} \in I_0) = \gamma \ .$$

Separately, the confidence interval I_1 satisfies

$$P(\beta_1^{\text{true}} \in I_1) = \gamma \ .$$

The parameter $t_{\gamma,q}$ is the value that satisfies

$$\int_{-t_{\gamma,q}}^{t_{\gamma,q}} f_{T,q}(s)\, ds = \gamma \ ,$$

where $f_{T,q}$ is the probability density function for the Student's t-distribution with q degrees of freedom. We recall the frequentistic interpretation of confidence intervals from our earlier estmation discussion of Unit II.

Note that we can relate $t_{\gamma,q}$ to the cumulative distribution function of the t-distribution, $F_{T,q}$, as follows. First, we note that $f_{T,q}$ is symmetric about zero. Thus, we have

$$\int_{0}^{t_{\gamma,q}} f_{T,q}(s)\, ds = \frac{\gamma}{2}$$

and

$$F_{T,q}(x) \equiv \int_{-\infty}^{x} f_{T,q}(s)\, ds = \frac{1}{2} + \int_{0}^{x} f_{T,q}(s)\, ds \ .$$

Evaluating the cumulative distribution function at $t_{\gamma,q}$ and substituting the desired integral relationship,

$$F_{T,q}(t_{\gamma,q}) = \frac{1}{2} + \int_{0}^{t_{\gamma,q}} f_{T,q}(t_{\gamma,q})\, ds = \frac{1}{2} + \frac{\gamma}{2} \ .$$

In particular, given an inverse cumulative distribution function for the Student's t-distribution, we can readily compute $t_{\gamma,q}$ as

$$t_{\gamma,q} = F_{T,q}^{-1}\left(\frac{1}{2} + \frac{\gamma}{2}\right) .$$

For convenience, we have tabulated the coefficients for 95% confidence level for select values of degrees of freedom in Table 19.1(a).

(a) t-distribution

q	$t_{\gamma,q}\vert_{\gamma=0.95}$
5	2.571
10	2.228
15	2.131
20	2.086
25	2.060
30	2.042
40	2.021
50	2.009
60	2.000
∞	1.960

(b) F-distribution

	$s_{\gamma,k,q}\vert_{\gamma=0.95}$							
q	$k=1$	2	3	4	5	10	15	20
5	2.571	3.402	4.028	4.557	5.025	6.881	8.324	9.548
10	2.228	2.865	3.335	3.730	4.078	5.457	6.533	7.449
15	2.131	2.714	3.140	3.496	3.809	5.044	6.004	6.823
20	2.086	2.643	3.049	3.386	3.682	4.845	5.749	6.518
25	2.060	2.602	2.996	3.322	3.608	4.729	5.598	6.336
30	2.042	2.575	2.961	3.280	3.559	4.653	5.497	6.216
40	2.021	2.542	2.918	3.229	3.500	4.558	5.373	6.064
50	2.009	2.523	2.893	3.198	3.464	4.501	5.298	5.973
60	2.000	2.510	2.876	3.178	3.441	4.464	5.248	5.913
∞	1.960	2.448	2.796	3.080	3.327	4.279	5.000	5.605

Table 19.1: The coefficient for computing the 95% confidence interval from Student's t-distribution and F-distribution.

Joint Confidence Intervals

Sometimes we are more interested in constructing joint confidence intervals — confidence intervals within which the true values of *all* the parameters lie in a fraction γ of all realizations. These confidence intervals are constructed in essentially the same manner as the individual confidence intervals and take on a similar form. Joint confidence intervals for β_0 and β_1 are of the form

$$I_0^{\text{joint}} \equiv \left[\hat{\beta}_0 - s_{\gamma,2,m-2}\sqrt{\widehat{\Sigma}_{11}}\,, \hat{\beta}_0 + s_{\gamma,2,m-2}\sqrt{\widehat{\Sigma}_{11}}\right]$$

and

$$I_1^{\text{joint}} \equiv \left[\hat{\beta}_1 - s_{\gamma,2,m-2}\sqrt{\widehat{\Sigma}_{22}}\,, \hat{\beta}_1 + s_{\gamma,2,m-2}\sqrt{\widehat{\Sigma}_{22}}\right] \ .$$

Note that the parameter $t_{\gamma,m-2}$ has been replaced by a parameter $s_{\gamma,2,m-2}$. More generally, the parameter takes the form $s_{\gamma,n,m-n}$, where γ is the confidence level, n is the number of parameters in the model (here $n = 2$), and m is the number of measurements. With the joint confidence interval, we have

$$P\left(\beta_0^{\text{true}} \in I_0^{\text{joint}} \ \textit{and}\ \beta_1^{\text{true}} \in I_1^{\text{joint}}\right) \geq \gamma \ .$$

Note the inequality — $\geq \gamma$ — is because our intervals are a "bounding box" for the actual sharp confidence ellipse.

The parameter $s_{\gamma,k,q}$ is related to γ-quantile for the F-distribution, $g_{\gamma,k,q}$, by

$$s_{\gamma,k,q} = \sqrt{k g_{\gamma,k,q}} \ .$$

Note $g_{\gamma,k,q}$ satisfies

$$\int_0^{g_{\gamma,k,q}} f_{F,k,q}(s)\, ds = \gamma \ ,$$

where $f_{F,k,q}$ is the probability density function of the F-distribution; we may also express $g_{\gamma,k,q}$ in terms of the cumulative distribution function of the F-distribution as

$$F_{F,k,q}(g_{\gamma,k,q}) = \int_0^{g_{\gamma,k,q}} f_{F,k,q}(s)\, ds = \gamma \ .$$

In particular, we can explicitly write $s_{\gamma,k,q}$ using the inverse cumulative distribution for the F-distribution, i.e.

$$s_{\gamma,k,q} = \sqrt{k g_{\gamma,k,q}} = \sqrt{k F^{-1}_{F,k,q}(\gamma)} \ .$$

For convenience, we have tabulated the values of $s_{\gamma,k,q}$ for several different combinations of k and q in Table 19.1(b).

We note that

$$s_{\gamma,k,q} = t_{\gamma,q}, \quad k = 1 \ ,$$

as expected, because the joint distribution is same as the individual distribution for the case with one parameter. Furthermore,

$$s_{\gamma,k,q} > t_{\gamma,q}, \quad k > 1 \ ,$$

indicating that the joint confidence intervals are larger than the individual confidence intervals. In other words, the individual confidence intervals are too small to yield jointly the desired γ.

We can understand these confidence intervals with some simple examples.

Example 19.1.2 least-squares estimate for a constant model
Let us consider a simple response model of the form

$$Y_{\text{model}}(x; \beta) = \beta_0 \ ,$$

where β_0 is the single parameter to be determined. The overdetermined system is given by

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \beta_0 = X\beta_0 \ ,$$

and we recognize $X = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}^{\mathrm{T}}$. Note that we have

$$X^{\mathrm{T}} X = m \ .$$

For this simple system, we can develop an explicit expression for the least squares estimate for β_0^{true}, $\hat{\beta}_0$ by solving the normal equation, i.e.

$$X^{\mathrm{T}} X \hat{\beta}_0 = X^{\mathrm{T}} Y \quad \Rightarrow \quad m\hat{\beta}_0 = \sum_{i=1}^{m} Y_i \quad \Rightarrow \quad \hat{\beta}_0 = \frac{1}{m}\sum_{i=1}^{m} Y_i \ .$$

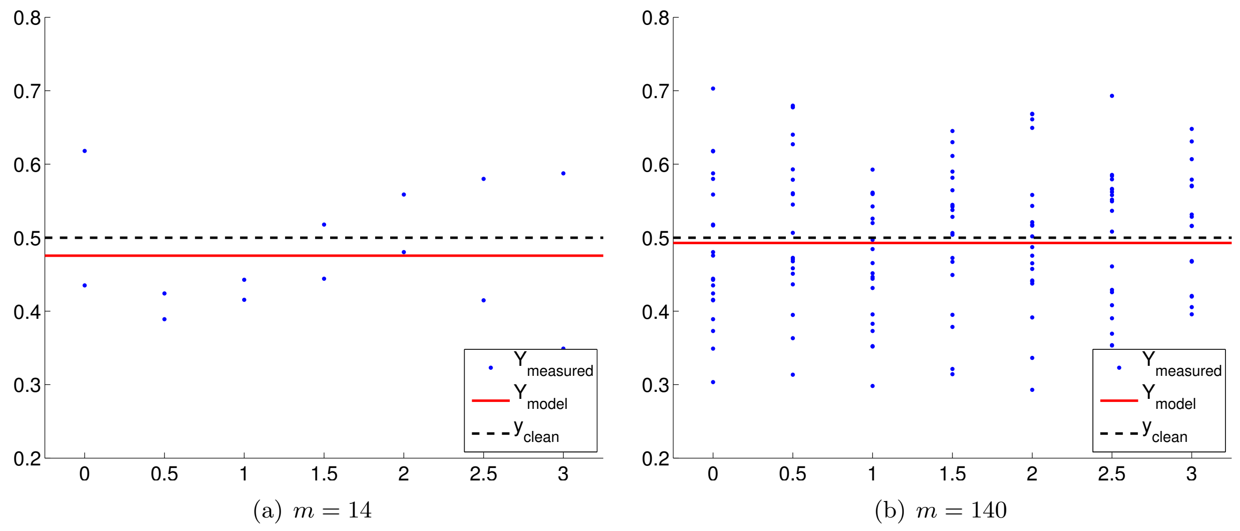
Our parameter estimator $\hat{\beta}_0$ is (not surprisingly) identical to the sample mean of Chapter 11 since our model here $Y = \mathcal{N}(\beta_0^{\text{true}}, \sigma^2)$ is identical to the model of Chapter 11.

The covariance matrix (which is a scalar for this case),

$$\widehat{\Sigma} = \hat{\sigma}^2 (X^{\mathrm{T}} X)^{-1} = \hat{\sigma}^2/m \ .$$

Thus, the confidence interval, I_0, has the Half Length

$$\text{Half Length}(I_0) = t_{\gamma,m-1}\sqrt{\widehat{\Sigma}} = t_{\gamma,m-1}\hat{\sigma}/\sqrt{m} \ .$$



(a) $m = 14$ (b) $m = 140$

Figure 19.2: Least square fitting of a constant function using a constant model.

Our confidence in the estimator $\hat{\beta}_0$ converges as $1/\sqrt{m} = m^{-1/2}$. Again, the convergence rate is identical to that in Chapter 11.

As an example, consider a random function of the form

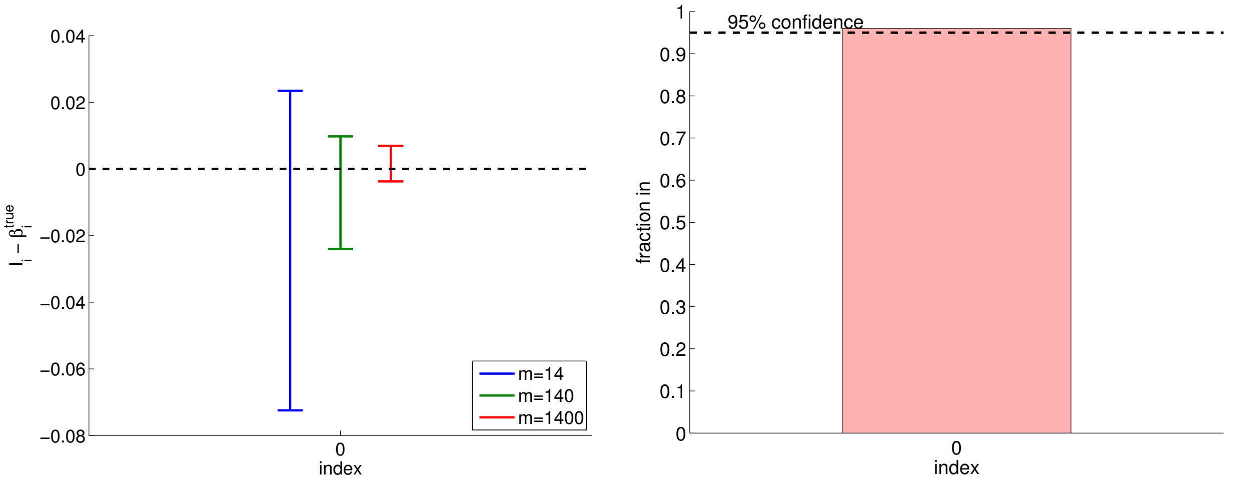
$$Y \sim \frac{1}{2} + \mathcal{N}(0, \sigma^2) \; ,$$

with the variance $\sigma^2 = 0.01$, and a constant (polynomial) response model, i.e.

$$Y_{\text{model}}(x; \beta) = \beta_0 \; .$$

Note that the true parameter is given by $\beta_0^{\text{true}} = 1/2$. Our objective is to compute the least-squares estimate of β_0^{true}, $\hat{\beta}_0$, and the associated confidence interval estimate, I_0. We take measurements at seven points, $x = 0, 0.5, 1.0, \ldots, 3.0$; at each point we take n_{sample} measurements for the total of $m = 7 \cdot n_{\text{sample}}$ measurements. Several measurements (or replication) at the same x can be advantageous, as we will see shortly; however it is also possible in particular thanks to our homoscedastic assumption to take only a single measurement at each value of x.

The results of the least squares fitting for $m = 14$ and $m = 140$ are shown in Figure 19.2. Here y_{clean} corresponds to the noise-free data, $y_{\text{clean}} = 1/2$. The convergence of the 95% confidence interval with number of samples is depicted in Figure 19.3(a). We emphasize that for the purpose of these figures and later similar figures we plot the confidence intervals shifted by β_0^{true}. We would not know β_0^{true} in practice, however these figures are intended to demonstrate the performance of the confidence intervals in a case in which the true values are indeed known. Each of the realizations of the confidence intervals includes the true parameter value. In fact, for the $m = 140$ case, Figure 19.3(b) shows that 96 out of 100 realizations of the confidence interval include the true parameter value, which is consistent with the 95% confidence level for the interval. (Of course in practice we would compute only a single confidence interval.)



(a) 95% shifted confidence intervals (b) 95% ci in/out (100 realizations, $m = 140$)

Figure 19.3: (a) The variation in the 95% confidence interval with the sampling size m for the constant model fitting. (b) The frequency of the confidence interval I_0 including the true parameter β_0^{true}.

Example 19.1.3 constant regression model and its relation to deterministic analysis
Earlier, we studied how a data perturbation $g - g_0$ affects the least squares solution $z^* - z_0$. In the analysis we assumed that there is a unique solution z_0 to the clean problem, $Bz_0 = g_0$, and then compared the solution to the least squares solution z^* to the perturbed problem, $Bz^* = g$. As in the previous analysis, we use subscript 0 to represent superscript "true" to declutter the notation.

Now let us consider a statistical context, where the perturbation in the right-hand side is induced by the zero-mean normal distribution with variance σ^2. In this case,

$$\frac{1}{m}\sum_{i=1}^{m}(g_{0,i} - g_i)$$

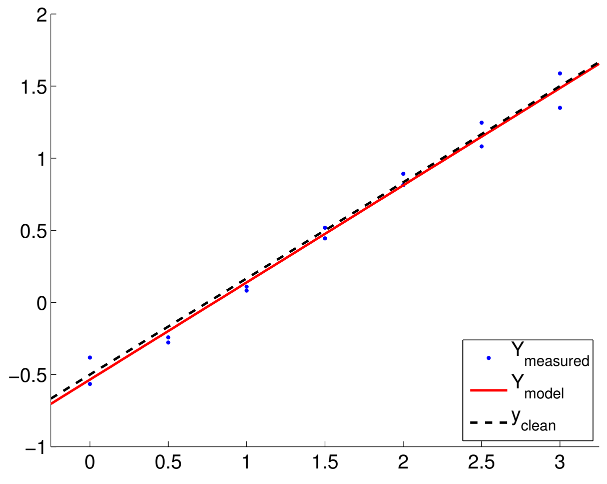
is the sample mean of the normal distribution, which we expect to incur fluctuations on the order of $\sigma/\sqrt{m}$. In other words, the deviation in the solution is

$$z_0 - z^* = (B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}(g_0 - g) = m^{-1}\sum_{i=1}^{m}(g_{0,i} - g_i) = \mathcal{O}\left(\frac{\sigma}{\sqrt{m}}\right).$$

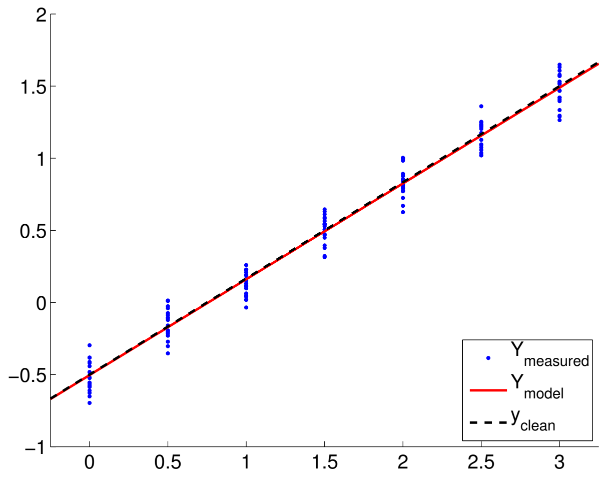
Note that this convergence is faster than that obtained directly from the earlier perturbation bounds,

$$|z_0 - z^*| \le \frac{1}{\sqrt{m}}\|g_0 - g\| = \frac{1}{\sqrt{m}}\sqrt{m}\sigma = \sigma\ ,$$

which suggests that the error would not converge. The difference suggests that the perturbation resulting from the normal noise is different from any arbitrary perturbation. In particular, recall that the deterministic bound based on the Cauchy-Schwarz inequality is pessimistic when the perturbation is not well aligned with col(B), which is a constant. In the statistical context, the noise $g_0 - g$ is relatively orthogonal to the column space col(B), resulting in a faster convergence than for an arbitrary perturbation.



(a) $m = 14$



(b) $m = 140$

Figure 19.4: Least square fitting of a linear function using a linear model.

Example 19.1.4 least-squares estimate for a linear model
As the second example, consider a random function of the form

$$Y(x) \sim -\frac{1}{2} + \frac{2}{3}x + \mathcal{N}(0, \sigma^2) \ ,$$

with the variance $\sigma^2 = 0.01$. The objective is to model the function using a linear model

$$Y_{\text{model}}(x; \beta) = \beta_0 + \beta_1 x \ ,$$

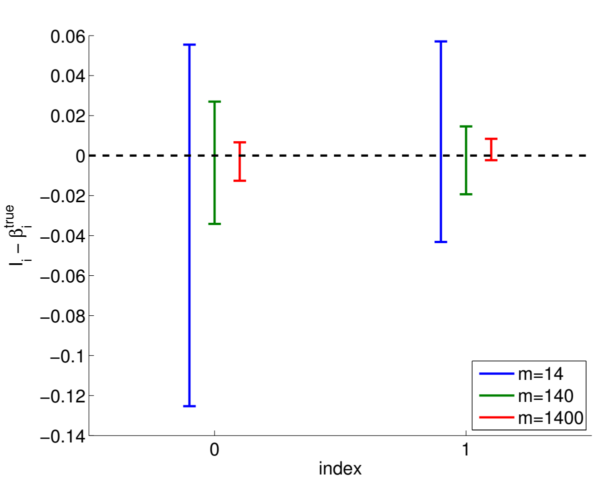
where the parameters (β_0, β_1) are found through least squares fitting. Note that the true parameters are given by $\beta_0^{\text{true}} = -1/2$ and $\beta_1^{\text{true}} = 2/3$. As in the constant model case, we take measurements at seven points, $x = 0, 0.5, 1.0, \ldots, 3.0$; at each point we take n_{sample} measurements for the total of $m = 7 \cdot n_{\text{sample}}$ measurements. Here, it is important that we take measurements at at least two different x locations; otherwise, the matrix B will be singular. This makes sense because if we choose only a single x location we are effectively trying to fit a line through a single point, which is an ill-posed problem.

The results of the least squares fitting for $m = 14$ and $m = 140$ are shown in Figure 19.4. We see that the fit gets tighter as the number of samples, m, increases.

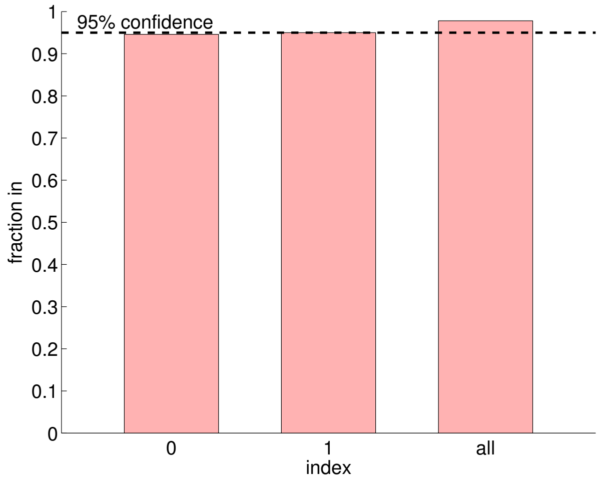
We can also quantify the quality of the parameter estimation in terms of the confidence intervals. The convergence of the individual 95% confidence interval with number of samples is depicted in Figure 19.5(a). Recall that the individual confidence intervals, I_i, $i = 0, 1$, are constructed to satisfy

$$P(\beta_0^{\text{true}} \in I_0) = \gamma \quad \text{and} \quad P(\beta_1^{\text{true}} \in I_1) = \gamma$$

with the confidence level γ (95% for this case) using the Student's t-distribution. Clearly each of the individual confidence intervals gets tighter as we take more measurements and our confidence in our parameter estimate improves. Note that the realization of confidence intervals include the true parameter value for each of the sample sizes considered.



(a) 95% shifted confidence intervals



(b) 95% ci in/out (1000 realizations, $m = 140$)

Figure 19.5: a) The variation in the 95% confidence interval with the sampling size m for the linear model fitting. b) The frequency of the individual confidence intervals I_0 and I_1 including the true parameters β_0^{true} and β_1^{true} (0 and 1, respectively), and $I_0^{\text{joint}} \times I_1^{\text{joint}}$ jointly including $(\beta_0^{\text{true}}, \beta_1^{\text{true}})$ (all).

We can verify the validity of the individual confidence intervals by measuring the frequency that each of the true parameters lies in the corresponding interval for a large number of realizations. The result for 1000 realizations is shown in Figure 19.5(b). The column indexed "0" corresponds to the frequency of $\beta_0^{\text{true}} \in I_0$, and the column indexed "1" corresponds to the frequency of $\beta_1^{\text{true}} \in I_1$. As designed, each of the individual confidence intervals includes the true parameter $\gamma = 95\%$ of the times.

We can also check the validity of the joint confidence interval by measuring the frequency that the parameters (β_1, β_2) jointly takes on values within $I_0^{\text{joint}} \times I_1^{\text{joint}}$. Recall that the our joint intervals are designed to satisfy

$$P\left(\beta_0^{\text{true}} \in I_0^{\text{joint}} \ \textit{and} \ \beta_1^{\text{true}} \in I_1^{\text{joint}}\right) \geq \gamma$$

and it uses the F-distribution. The column indexed "all" in Figure 19.5(b). corresponds to the frequency that $(\beta_0^{\text{true}}, \beta_1^{\text{true}}) \in I_0^{\text{joint}} \times I_1^{\text{joint}}$. Note that the joint success rate is a slightly higher ($\approx 97\%$) than γ since the confidence intervals we provide are a simple but conservative bound for the actual elliptical confidence region. On the other hand, if we mistakenly use the individual confidence intervals instead of the joint confidence interval, the individual confidence intervals are too small and jointly include the true parameters only $\approx 92\%$ of the time. Thus, we emphasize that it is important to construct confidence intervals that are appropriate for the question of interest.

·

19.1.5 Hypothesis Testing

We can also, in place of our CI's (or in fact, based on our CI's), consider a hypotheses on the parameters — and then test these hypotheses. For example, in this last example, we might wish to test the hypothesis (known as the null hypothesis) that $\beta_0^{\text{true}} = 0$. We consider Example 19.1.4 for the case in which $m = 1400$. Clearly, our CI does not include $\beta_0^{\text{true}} = 0$. Thus most likely $\beta_0^{\text{true}} \neq 0$, and we reject the hypothesis. In general, we reject the hypothesis when the CI does not include zero.

We can easily analyze the Type I error, which is defined as the probability that we reject the hypothesis when the hypothesis is in fact true. We assume the hypothesis is true. Then, the probability that the CI does not include zero — and hence that we reject the hypothesis — is 0.05, since we know that 95% of the time our CI will include zero — the true value under our hypothesis. (This can be rephrased in terms of a test statistic and a critical region for rejection.) We denote by 0.05 the "size" of the test — the probability that we incorrectly reject the hypothesis due to an unlucky (rare) "fluctuation."

We can also introduce the notion of a Type II error, which is defined as the probability that we accept the hypothesis when the hypothesis is in fact false. And the "power" of the test is the probability that we reject the hypothesis when the hypothesis in fact false: the power is $1 -$ the Type II error. Typically it is more difficult to calculate Type II errors (and power) than Type I errors.

19.1.6 Inspection of Assumptions

In estimating the parameters for the response model and constructing the corresponding confidence intervals, we relied on the noise assumptions (N1), (N2), and (N3). In this section, we consider examples that illustrate how the assumptions may be broken. Then, we propose methods for verifying the plausibility of the assumptions. Note we give here some rather simple tests without any underlying statistical structure; in fact, it is possible to be more rigorous about when to accept or reject our noise and bias hypotheses by introducing appropriate statistics such that "small" and "large" can be quantified. (It is also possible to directly pursue our parameter estimation under more general noise assumptions.)

Checking for Plausibility of the Noise Assumptions

Let us consider a system governed by a random affine function, but assume that the noise $\epsilon(x)$ is perfectly correlated in x. That is,

$$Y(x_i) = \beta_0^{\text{true}} + \beta_1^{\text{true}} x_i + \epsilon(x_i) \ ,$$

where

$$\epsilon(x_1) = \epsilon(x_2) = \cdots = \epsilon(x_m) \sim \mathcal{N}(0, \sigma^2) \ .$$

Even though the assumptions (N1) and (N2) are satisfied, the assumption on independence, (N3), is violated in this case. Because the systematic error shifts the output by a constant, the coefficient of the least-squares solution corresponding to the constant function β_0 would be shifted by the error. Here, the (perfectly correlated) noise ϵ is incorrectly interpreted as signal.

Let us now present a test to verify the plausibility of the assumptions, which would detect the presence of the above scenario (amongst others). The verification can be accomplished by sampling the system in a controlled manner. Say we gather N samples evaluated at x_L,

$$L_1, L_2, \dots, L_N \quad \text{where} \quad L_i = Y(x_L), \quad i = 1, \dots, N \ .$$

Similarly, we gather another set of N samples evaluated at $x_R \neq x_L$,

$$R_1, R_2, \ldots, R_N \quad \text{where} \quad R_i = Y(x_R), \quad i = 1, \ldots, N \ .$$

Using the samples, we first compute the estimate for the mean and variance for L,

$$\hat{\mu}_L = \frac{1}{N}\sum_{i=1}^{N} L_i \quad \text{and} \quad \hat{\sigma}_L^2 = \frac{1}{N-1}\sum_{i=1}^{N}(L_i - \hat{\mu}_L)^2 \ ,$$

and those for R,

$$\hat{\mu}_R = \frac{1}{N}\sum_{i=1}^{N} R_i \quad \text{and} \quad \hat{\sigma}_R^2 = \frac{1}{N-1}\sum_{i=1}^{N}(R_i - \hat{\mu}_R)^2 \ .$$

To check for the normality assumption (N1), we can plot the histogram for L and R (using an appropriate number of bins) and for $\mathcal{N}(\hat{\mu}_L, \hat{\sigma}_L^2)$ and $\mathcal{N}(\hat{\mu}_R, \hat{\sigma}_R^2)$. If the error is normally distributed, these histograms should be similar, and resemble the normal distribution. In fact, there are much more rigorous and quantitative statistical tests to assess whether data derives from a particular (here normal) population.

To check for the homoscedasticity assumption (N2), we can compare the variance estimate for samples L and R, i.e., is $\hat{\sigma}_L^2 \approx \hat{\sigma}_R^2$? If $\hat{\sigma}_L^2 \not\approx \hat{\sigma}_R^2$, then assumption (N2) is not likely plausible because the noise at x_L and x_R have different distributions.

Finally, to check for the uncorrelatedness assumption (N3), we can check the correlation coefficient $\rho_{L,R}$ between L and R. The correlation coefficient is estimated as

$$\hat{\rho}_{L,R} = \frac{1}{\hat{\sigma}_L \hat{\sigma}_R}\frac{1}{N-1}\sum_{i=1}^{N}(L_i - \hat{\mu}_L)(R_i - \hat{\mu}_R) \ .$$

If the correlation coefficient is not close to 0, then the assumption (N3) is not likely plausible. In the example considered with the correlated noise, our system would fail this last test.

Checking for Presence of Bias

Let us again consider a system governed by an affine function. This time, we assume that the system is noise free, i.e.

$$Y(x) = \beta_0^{\text{true}} + \beta_1^{\text{true}} x \ .$$

We will model the system using a constant function,

$$Y_{\text{model}} = \beta_0 \ .$$

Because our constant model would match the mean of the underlying distribution, we would interpret $Y - \text{mean}(Y)$ as the error. In this case, the signal is interpreted as a noise.

We can check for the presence of bias by checking if

$$|\hat{\mu}_L - \widehat{Y}_{\text{model}}(x_L)| \sim \mathcal{O}(\hat{\sigma}) \ .$$

If the relationship does not hold, then it indicates a lack of fit, i.e., the presence of bias. Note that replication — as well as data exploration more generally — is crucial in understanding the assumptions. Again, there are much more rigorous and quantitative statistical (say, hypothesis) tests to assess whether bias is present.

19.2 General Case

We consider a more general case of regression, in which we do not restrict ourselves to a linear response model. However, we still assume that the noise assumptions (N1), (N2), and (N3) hold.

19.2.1 Response Model

Consider a general relationship between the measurement Y, response model Y_{model}, and the noise ϵ of the form

$$Y(x) = Y_{\text{model}}(x;\beta) + \epsilon \ ,$$

where the independent variable x is vector valued with p components, i.e.

$$x = \begin{pmatrix} x_{(1)}, x_{(2)}, \cdots, x_{(p)} \end{pmatrix}^{\mathrm{T}} \in D \subset \mathbb{R}^p \ .$$

The response model is of the form

$$Y_{\text{model}}(x;\beta) = \beta_0 + \sum_{j=1}^{n-1} \beta_j h_j(x) \ ,$$

where h_j, $j = 0, \ldots, n-1$, are the basis functions and β_j, $j = 0, \ldots, n-1$, are the regression coefficients. Note that we have chosen $h_0(x) = 1$. Similar to the affine case, we assume that Y_{model} is sufficiently rich (with respect to the underlying random function Y), such that there exists a parameter β^{true} with which $Y_{\text{model}}(\cdot;\beta^{\text{true}})$ perfectly describes the behavior of the noise-free underlying function, i.e., unbiased. (Equivalently, there exists a β^{true} such that $Y(x) \sim \mathcal{N}(Y_{\text{model}}(x;\beta^{\text{true}}), \sigma^2)$.

It is important to note that this is still a *linear* regression. It is linear in the sense that the regression coefficients β_j, $j = 0, \ldots, n-1$, appear linearly in Y_{model}. The basis functions h_j, $j = 0, \ldots, n-1$, do not need to be linear in x; for example, $h_1(x_{(1)}, x_{(2)}, x_{(3)}) = x_{(1)} \exp(x_{(2)} x_{(3)})$ is perfectly acceptable for a basis function. The simple case considered in the previous section corresponds to $p = 1$, $n = 2$, with $h_0(x) = 1$ and $h_1(x) = x$.

There are two main approaches to choose the basis functions.

(i) Functions derived from anticipated behavior based on physical models. For example, to deduce the friction coefficient, we can relate the static friction and the normal force following the Amontons' and Coulomb's laws of friction,

$$F_{\text{f, static}} = \mu_{\text{s}} \, F_{\text{normal, applied}} \ ,$$

where $F_{\text{f, static}}$ is the friction force, μ_{s} is the friction coefficient, and $F_{\text{normal, applied}}$ is the normal force. Noting that $F_{\text{f, static}}$ is a linear function of $F_{\text{normal, applied}}$, we can choose a linear basis function $h_1(x) = x$.

(ii) Functions derived from general mathematical approximations, which provide good accuracy in some neighborhood D. Low-order polynomials are typically used to construct the model, for example

$$Y_{\text{model}}(x_{(1)}, x_{(2)};\beta) = \beta_0 + \beta_1 x_{(1)} + \beta_2 x_{(2)} + \beta_3 x_{(1)} x_{(2)} + \beta_4 x_{(1)}^2 + \beta_5 x_{(2)}^2 \ .$$

Although we can choose n large and let least-squares find the good β — the good model within our general expansion — this is typically not a good idea: to avoid *overfitting*, we must ensure the number of experiments is much greater than the order of the model, i.e., $m \gg n$. We return to overfitting later in the chapter.

19.2.2 Estimation

We take m measurements to collect m independent variable-measurement pairs

$$(x_i, Y_i), \quad i = 1, \dots, m \ ,$$

where $x_i = (x_{(1)}, x_{(2)}, \dots, x_{(p)})_i$. We claim

$$\begin{aligned} Y_i &= Y_{\text{model}}(x_i; \beta) + \epsilon_i \\ &= \beta_0 + \sum_{j=1}^{n-1} \beta_j h_j(x_i) + \epsilon_i, \quad i = 1, \dots, m \ , \end{aligned}$$

which yields

$$\underbrace{\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{pmatrix}}_{Y} = \underbrace{\begin{pmatrix} 1 & h_1(x_1) & h_2(x_1) & \dots & h_{n-1}(x_1) \\ 1 & h_1(x_2) & h_2(x_2) & \dots & h_{n-1}(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & h_1(x_m) & h_2(x_m) & \dots & h_{n-1}(x_m) \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \epsilon(x_1) \\ \epsilon(x_2) \\ \vdots \\ \epsilon(x_m) \end{pmatrix}}_{\epsilon} .$$

The least-squares estimator $\hat{\beta}$ is given by

$$(X^{\mathrm{T}} X)\hat{\beta} = X^{\mathrm{T}} Y \ ,$$

and the goodness of fit is measured by $\hat{\sigma}$,

$$\hat{\sigma} = \left(\frac{1}{m-n} \| Y - \widehat{Y} \|^2 \right)^{1/2} \ ,$$

where

$$\widehat{Y} = \begin{pmatrix} \widehat{Y}_{\text{model}}(x_1) \\ \widehat{Y}_{\text{model}}(x_2) \\ \vdots \\ \widehat{Y}_{\text{model}}(x_m) \end{pmatrix} = \begin{pmatrix} \hat{\beta}_0 + \sum_{j=1}^{n-1} \hat{\beta}_j h_j(x_1) \\ \hat{\beta}_0 + \sum_{j=1}^{n-1} \hat{\beta}_j h_j(x_2) \\ \vdots \\ \hat{\beta}_0 + \sum_{j=1}^{n-1} \hat{\beta}_j h_j(x_m) \end{pmatrix} = X\hat{\beta} \ .$$

As before, the mean of the mean of the model is equal to the mean of the measurements, i.e.

$$\overline{\widehat{Y}} = \overline{Y} \ ,$$

where

$$\overline{\widehat{Y}} = \frac{1}{m} \sum_{i=1}^{m} \widehat{Y}_i \quad \text{and} \quad \overline{Y} = \frac{1}{m} \sum_{i=1}^{m} Y_i \ .$$

The preservation of the mean is ensured by the presence of the constant term $\beta_0 \cdot 1$ in our model.

19.2.3 Confidence Intervals

The construction of the confidence intervals follows the procedure developed in the previous section. Let us define the covariance matrix

$$\widehat{\Sigma} = \hat{\sigma}^2 (X^{\mathrm{T}} X)^{-1} \ .$$

Then, the individual confidence intervals are given by

$$I_j = \left[\hat{\beta}_j - t_{\gamma,m-n}\sqrt{\widehat{\Sigma}_{j+1,j+1}},\ \hat{\beta}_j + t_{\gamma,m-n}\sqrt{\widehat{\Sigma}_{j+1,j+1}} \right], \quad j = 0,\dots,n-1 \ ,$$

where $t_{\gamma,m-n}$ comes from the Student's t-distribution as before, i.e.

$$t_{\gamma,m-n} = F^{-1}_{T,m-n}\left(\frac{1}{2} + \frac{\gamma}{2}\right) ,$$

where $F^{-1}_{T,q}$ is the inverse cumulative distribution function of the t-distribution. The shifting of the covariance matrix indices is due to the index for the parameters starting from 0 and the index for the matrix starting from 1. Each of the individual confidence intervals satisfies

$$P(\beta_j^{\text{true}} \in I_j) = \gamma, \quad j = 0,\dots,n-1 \ ,$$

where γ is the confidence level.

We can also develop joint confidence intervals,

$$I_j^{\text{joint}} = \left[\hat{\beta}_j - s_{\gamma,n,m-n}\sqrt{\widehat{\Sigma}_{j+1,j+1}},\ \hat{\beta}_j + s_{\gamma,n,m-n}\sqrt{\widehat{\Sigma}_{j+1,j+1}} \right], \quad j = 0,\dots,n-1 \ ,$$

where the parameter $s_{\gamma,n,m-n}$ is calculated from the inverse cumulative distribution function for the F-distribution according to

$$s_{\gamma,n,m-n} = \sqrt{n F^{-1}_{F,n,m-n}(\gamma)} \ .$$

The joint confidence intervals satisfy

$$P\left(\beta_0^{\text{true}} \in I_0^{\text{joint}}, \beta_1^{\text{true}} \in I_1^{\text{joint}}, \dots, \beta_{n-2}^{\text{true}} \in I_{n-2}^{\text{joint}},\ \textit{and}\ \beta_{n-1}^{\text{true}} \in I_{n-1}^{\text{joint}}\right) \geq \gamma \ .$$

Example 19.2.1 least-squares estimate for a quadratic function
Consider a random function of the form

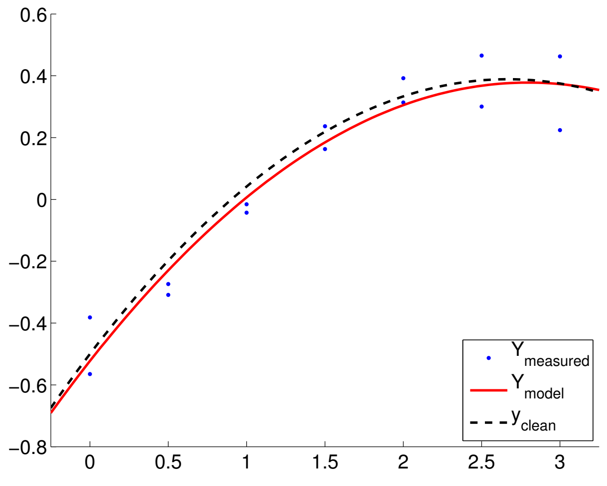
$$Y(x) \sim -\frac{1}{2} + \frac{2}{3}x - \frac{1}{8}x^2 + \mathcal{N}(0,\sigma^2) \ ,$$

with the variance $\sigma^2 = 0.01$. We would like to model the behavior of the function. Suppose we know (though a physical law or experience) that the output of the underlying process depends quadratically on input x. Thus, we choose the basis functions

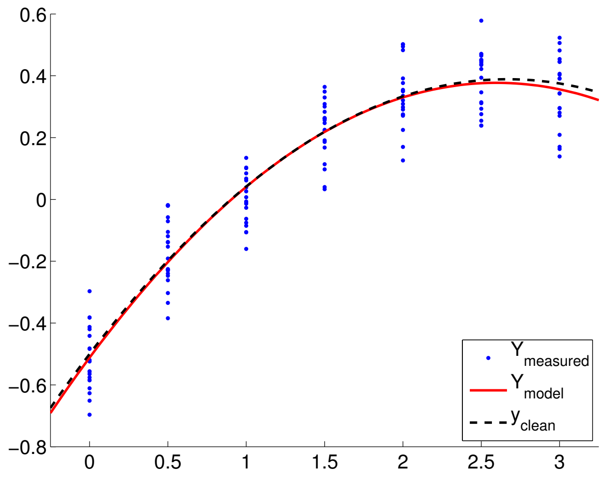
$$h_1(x) = 1, \quad h_2(x) = x, \quad \text{and} \quad h_3(x) = x^2 \ .$$

The resulting model is of the form

$$Y_{\text{model}}(x;\beta) = \beta_0 + \beta_1 x + \beta_2 x^2 \ ,$$



(a) $m = 14$



(b) $m = 140$

Figure 19.6: Least squares fitting of a quadratic function using a quadratic model.

where $(\beta_0, \beta_1, \beta_2)$ are the parameters to be determined through least squares fitting. Note that the true parameters are given by $\beta_0^{\text{true}} = -1/2$, $\beta_1^{\text{true}} = 2/3$, and $\beta_2^{\text{true}} = -1/8$.

The result of the calculation is shown in Figure 19.6. Our model qualitatively matches well with the underlying "true" model. Figure 19.7(a) shows that the 95% individual confidence interval for each of the parameters converges as the number of samples increase.

Figure 19.7(b) verifies that the individual confidence intervals include the true parameter approximately 95% of the times (shown in the columns indexed 0, 1, and 2). Our joint confidence interval also jointly include the true parameter about 98% of the times, which is greater than the prescribed confidence level of 95%. (Note that individual confidence intervals jointly include the true parameters only about 91% of the times.) These results confirm that both the individual and joint confidence intervals are reliable indicators of the quality of the respective estimates.

19.2.4 Overfitting (and Underfitting)

We have discussed the importance of choosing a model with a sufficiently large n — such that the true underlying distribution is representable and there would be no bias — but also hinted that n much larger than necessary can result in an *overfitting* of the data. Overfitting significantly degrades the quality of our parameter estimate and predictive model, especially when the data is noisy or the number of data points is small. Let us illustrate the effect of overfitting using a few examples.

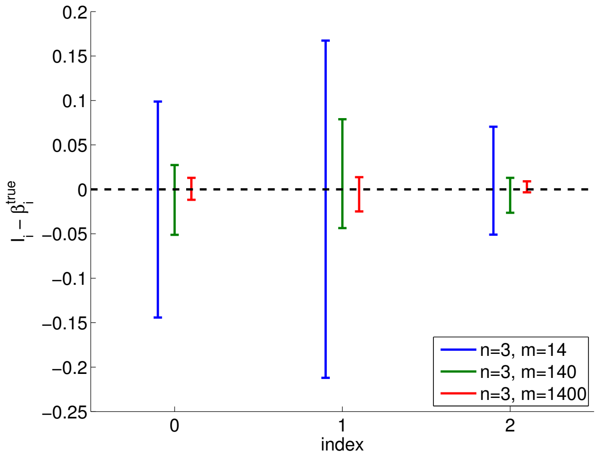
Example 19.2.2 overfitting of a linear function

Let us consider a noisy linear function

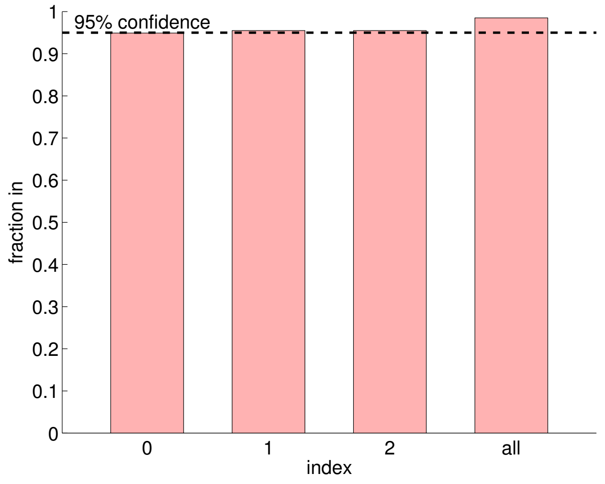
$$Y(x) \sim \frac{1}{2} + 2x + \mathcal{N}(0, \sigma^2) \ .$$

However, unlike in the previous examples, we assume that we do not know the form of the input-output dependency. In this and the next two examples, we will consider a general $n-1$ degree polynomial fit of the form

$$Y_{\text{model},n}(x; \beta) = \beta_0 + \beta_1 x^1 + \cdots + \beta_{n-1} x^{n-1} \ .$$



(a) 95% shifted confidence intervals



(b) 95% ci in/out (1000 realizations, $m = 140$)

Figure 19.7: (a) The variation in the 95% confidence interval with the sampling size m for the linear model fitting. (b) The frequency of the individual confidence intervals I_0, I_1, and I_2 including the true parameters β_0^{true}, β_1^{true}, and β_2^{true} (0, 1, and 2, respectively), and $I_0^{\text{joint}} \times I_1^{\text{joint}} \times I_2^{\text{joint}}$ jointly including $(\beta_0^{\text{true}}, \beta_1^{\text{true}}, \beta_2^{\text{true}})$ (all).

Note that the true parameters for the noisy function are

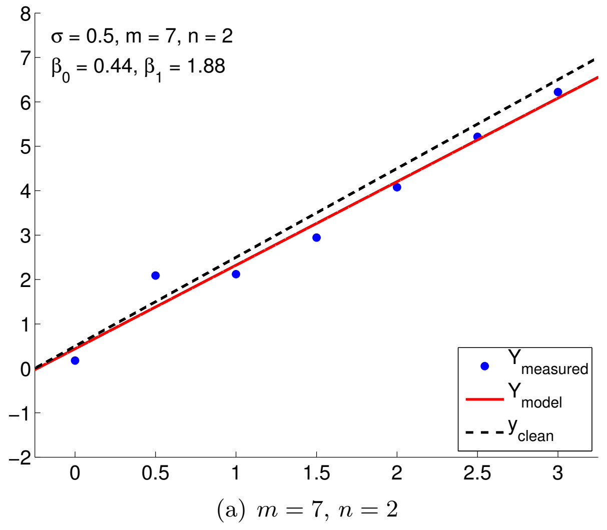
$$\beta_0^{\text{true}} = \frac{1}{2}, \quad \beta_1^{\text{true}} = 2, \quad \text{and} \quad \beta_2^{\text{true}} = \cdots = \beta_n^{\text{true}} = 0 \ ,$$

for any $n \geq 2$.

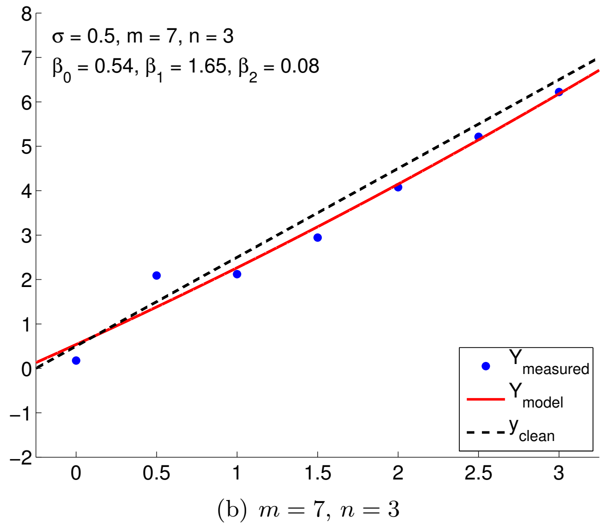
The results of fitting the noisy linear function using $m = 7$ measurements for the $n = 2$, $n = 3$, and $n = 5$ response models are shown in Figure 19.8(a), (b), and (c), respectively. The $n = 2$ is the nominal case, which matches the true underlying functional dependency, and the $n = 3$ and $n = 5$ cases correspond to overfitting cases. For each fit, we also state the least-squares estimate of the parameters. Qualitatively, we see that the prediction error, $y_{\text{clean}}(x) - Y_{\text{model}}(x)$, is larger for the quartic model ($n = 5$) than the affine model ($n = 2$). In particular, because the quartic model is fitting five parameters using just seven data points, the model is close to interpolating the noise, resulting in an oscillatory behavior that follows the noise. This oscillation becomes more pronounced as the noise level, σ, increases.

In terms of estimating the parameters β_0^{true} and β_1^{true}, the affine model again performs better than the overfit cases. In particular, the error in $\hat{\beta}_1$ is over an order of magnitude larger for the $n = 5$ model than for the $n = 2$ model. Fortunately, this inaccuracy in the parameter estimate is reflected in large confidence intervals, as shown in Figure 19.9. The confidence intervals are valid because our models with $n \geq 2$ are capable of representing the underlying functional dependency with $n^{\text{true}} = 2$, and the unbiasedness assumption used to construct the confidence intervals still holds. Thus, while the estimate may be poor, we are informed that we should not have much confidence in our estimate of the parameters. The large confidence intervals result from the fact that overfitting effectively leaves no degrees of freedom (or information) to estimate the noise because relatively too many degrees of freedom are used to determine the parameters. Indeed, when $m = n$, the confidence intervals are infinite.

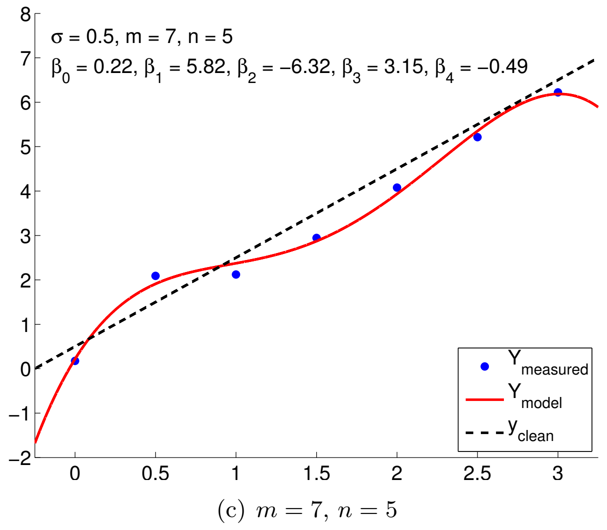
Because the model is unbiased, more data ultimately resolves the poor fit, as shown in Figure 19.8(d). However, recalling that the confidence intervals converge only as $m^{-1/2}$, a large



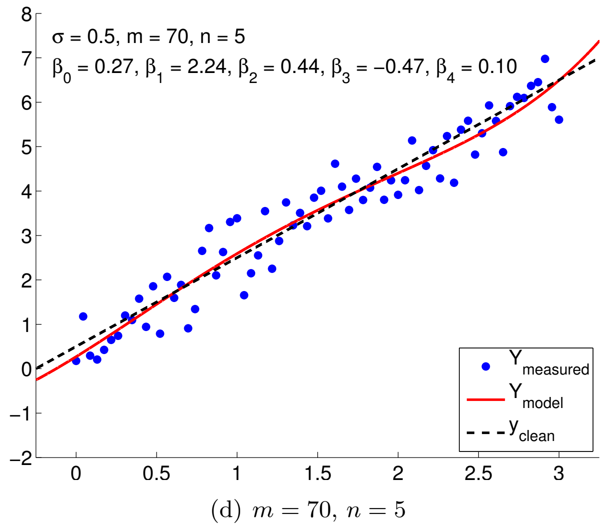
(a) $m = 7,\ n = 2$



(b) $m = 7,\ n = 3$



(c) $m = 7,\ n = 5$



(d) $m = 70,\ n = 5$

Figure 19.8: Least squares fitting of a linear function using polynomial models of various orders.

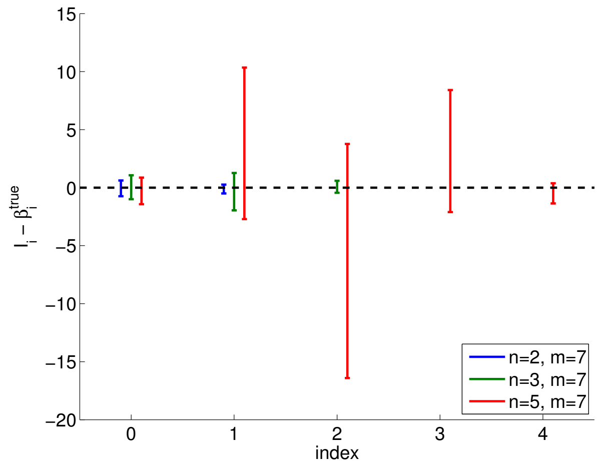


Figure 19.9: The 95% shifted confidence intervals for fitting a linear function using polynomial models of various orders.

number of samples are required to tighten the confidence intervals — and improve our parameter estimates — for the overfitting cases. Thus, deducing an appropriate response model based on, for example, physical principles can significantly improve the quality of the parameter estimates and the performance of the predictive model.

·

Begin Advanced Material

Example 19.2.3 overfitting of a quadratic function

In this example, we study the effect of overfitting in more detail. We consider data governed by a random quadratic function of the form

$$Y(x) \sim -\frac{1}{2} + \frac{2}{3}x - \frac{1}{8}cx^2 + \mathcal{N}(0, \sigma^2) ,$$

with $c = 1$. We again consider for our model the polynomial form $Y_{\text{model},n}(x; \beta)$.

Figure 19.10(a) shows a typical result of fitting the data using $m = 14$ sampling points and $n = 4$. Our cubic model includes the underlying quadratic distribution. Thus there is no bias and our noise assumptions are satisfied. However, compared to the quadratic model ($n = 3$), the cubic model is affected by the noise in the measurement and produces spurious variations. This spurious variation tend to disappear with the number of sampling points, and Figure 19.10(b) with $m = 140$ sampling points exhibits a more stable fit.

Figure 19.10(c) shows a realization of confidence intervals for the cubic model ($n = 4$) using $m = 14$ and $m = 140$ sampling points. A realization of confidence intervals for the quadratic model ($n = 3$) is also shown for comparison. Using the same set of data, the confidence intervals for the cubic model are larger than those of the quadratic model. However, the confidence intervals of the cubic model include the true parameter value for most cases. Figure 19.10(d) confirms that the 95% of the realization of the confidence intervals include the true parameter. Thus, the confidence intervals are reliable indicators of the quality of the parameter estimates, and in general the intervals get tighter with m, as expected. Modest overfitting, $n = 4$ *vs.* $n = 3$, with m sufficiently large, poses little threat.

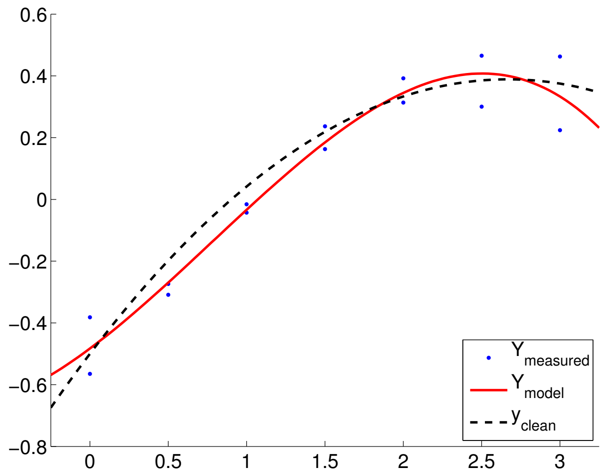
Let us check how overfitting affects the quality of the fit using two different measures. The first is a measure of how well we can predict, or reproduce, the clean underlying function; the second is a measure for how well we approximate the underlying parameters.

First, we quantify the quality of prediction using the maximum difference in the model and the clean underlying data,

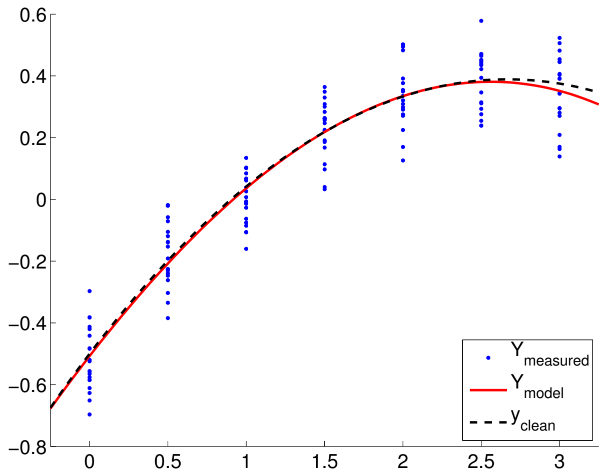
$$e_{\max} \equiv \max_{x \in [-1/4, 3+1/4]} |Y_{\text{model},n}(x; \hat{\beta}) - Y_{\text{clean}}(x)| .$$

Figure 19.11(a) shows the variation in the maximum prediction error with n for a few different values of m. We see that we get the closest fit (in the sense of the maximum error), when $n = 3$ — when there are no "extra" terms in our model. When only $m = 7$ data points are used, the quality of the regression degrades significantly as we overfit the data ($n > 3$). As the dimension of the model n approaches the number of measurements, m, we are effectively interpolating the noise. The interpolation induces a large error in the parameter estimates, and we can not estimate the noise since we are fitting the noise. We observe in general that the quality of the estimate improves as the number of samples is increased.

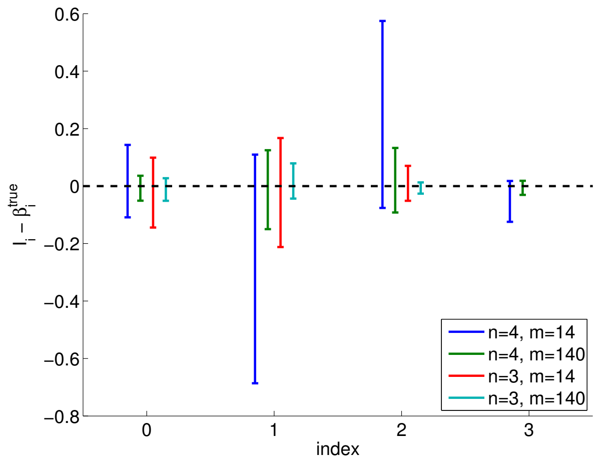
Second, we quantify the quality of the parameter estimates by measuring the error in the quadratic coefficient, i.e., $|\beta_2 - \hat{\beta}_2|$. Figure 19.11(b) shows that, not surprisingly, the error in the



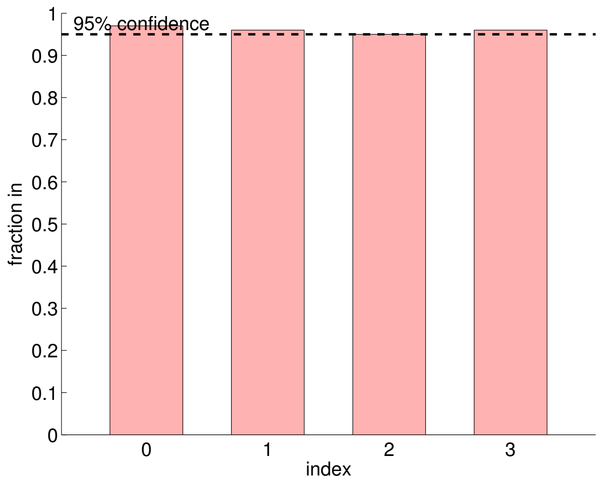
(a) $m = 14$



(b) $m = 140$

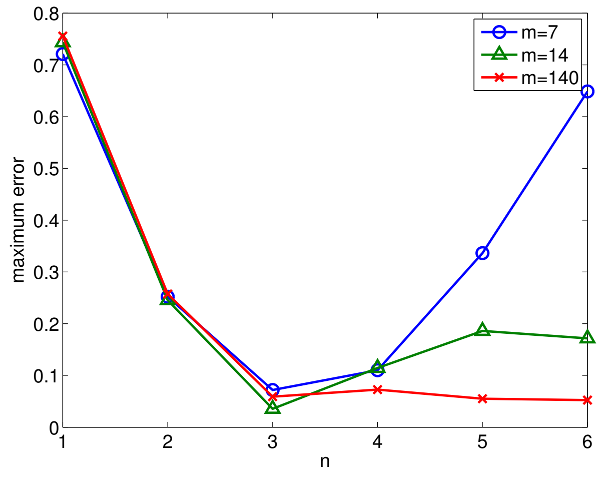


(c) 95% shifted confidence intervals

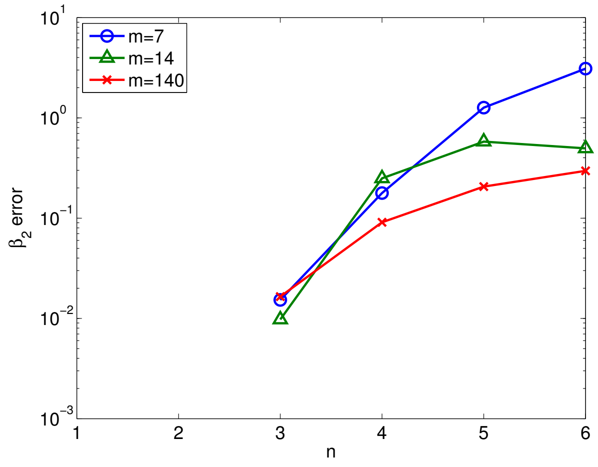


(d) 95% ci in/out (100 realizations, $m = 140$)

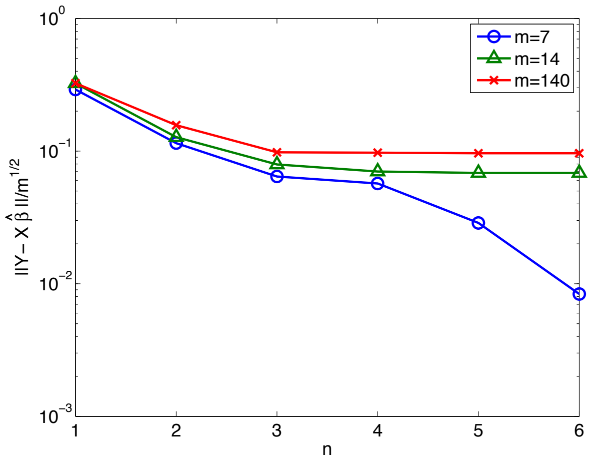
Figure 19.10: Least squares fitting of a quadratic function ($c = 1$) using a cubic model.



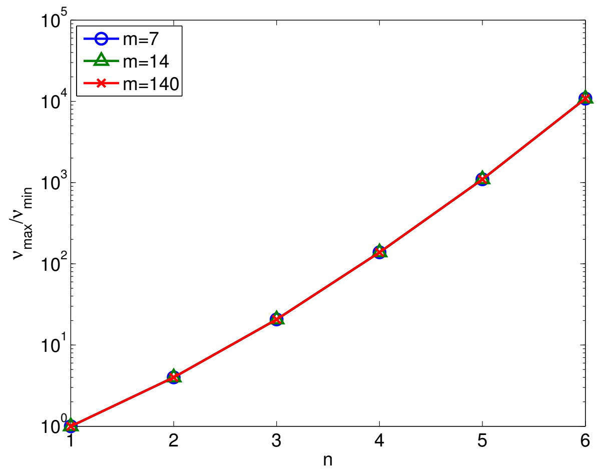
(a) maximum prediction error



(b) error in parameter β_2



(c) (normalized) residual



(d) condition number

Figure 19.11: Variation in the quality of regression with overfitting.

parameter increases under overfitting. In particular, for the small sample size of $m = 7$, the error in the estimate for β_3 increases from $\mathcal{O}(10^{-2})$ for $n = 3$ to $\mathcal{O}(1)$ for $n \geq 5$. Since β_3 is an $\mathcal{O}(1)$ quantity, this renders the parameter estimates for $n \geq 5$ essentially meaningless.

It is important to recognize that the degradation in the quality of estimate — either in terms of predictability or parameter error — is not due to the poor fit *at the data points*. In particular, the (normalized) residual,

$$\frac{1}{m^{1/2}}\|Y - X\hat{\beta}\| \ ,$$

which measures the fit at the data points, decreases as n increases, as shown in Figure 19.11(c). The decrease in the residual is not surprising. We have new coefficients which were previously implicitly zero and hence the least squares must provide a residual which is non-increasing as we increase n and let these coefficients realize their optimal values (with respect to residual minimization). However, as we see in Figure 19.11(a) and 19.11(b), better fit at data points does not imply better representation of the underlying function or parameters.

The worse prediction of the parameter is due to the increase in the conditioning of the problem ($\nu_{\max}/\nu_{\min}$), as shown in Figure 19.11(d). Recall that the error in the parameter is a function of both residual (goodness of fit at data points) *and* conditioning of the problem, i.e.

$$\frac{\|\hat{\beta} - \beta\|}{\|\beta\|} \leq \frac{\nu_{\max}}{\nu_{\min}} \frac{\|X\hat{\beta} - Y\|}{\|Y\|} \ .$$

As we increase n for a fixed m, we do reduce the residual. However, clearly the error is larger both in terms of output prediction and parameter estimate. Once again we see that the residual — and similar commonly used goodness of fit statistics such as R^2 — is not the "final answer" in terms of the success of any particular regression exercise.

Fortunately, similar to the previous example, this poor estimate of the parameters is reflected in large confidence intervals, as shown in Figure 19.12. Thus, while the estimates may be poor, we are informed that we should not have much confidence in our estimate of the parameters and that we need more data points to improve the fit.

Finally, we note that the conditioning of the problem reflects where we choose to make our measurements, our choice of response model, *and* how we choose to represent this response model. For example, as regards the latter, a Legendre (polynomial) expansion of order n would certainly decrease $\nu_{\max}/\nu_{\min}$, albeit at some complication in how we extract various parameters of interest.

·

Example 19.2.4 underfitting of a quadratic function

We consider data governed by a noisy quadratic function ($n^{\text{true}} \equiv 3$) of the form

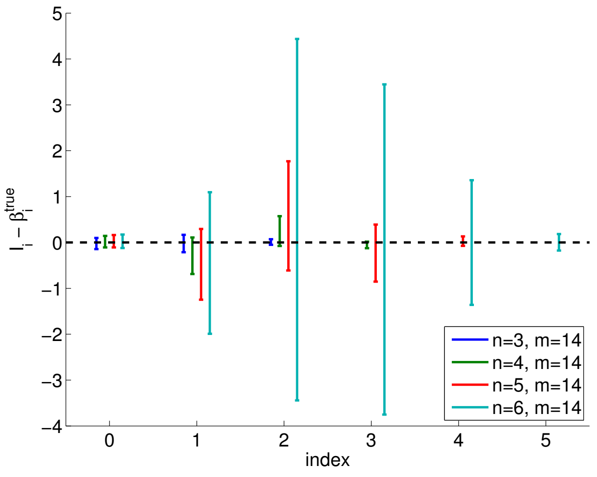
$$Y(x) \sim -\frac{1}{2} + \frac{2}{3}x - \frac{1}{8}cx^2 + \mathcal{N}(0,\sigma^2) \ .$$

We again assume that the input-output dependency is unknown. The focus of this example is underfitting; i.e., the case in which the degree of freedom of the model n is less than that of data n^{true}. In particular, we will consider an affine model ($n = 2$),

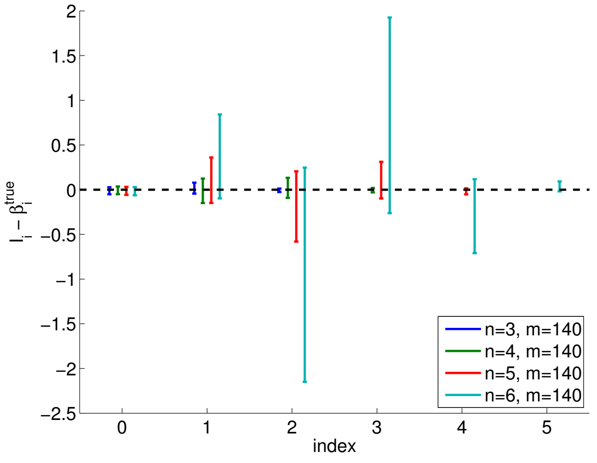
$$Y_{\text{model},2}(x;\beta) = \beta_0 + \beta_1 x \ ,$$

which is clearly *biased* (unless $c = 0$).

For the first case, we consider the true underlying distribution with $c = 1$, which results in a strong quadratic dependency of Y on x. The result of fitting the function is shown in Figure 19.13.



(a) $m = 14$



(b) $m = 140$

Figure 19.12: The variation in the confidence intervals for fitting a quadratic function using quadratic ($n = 3$), cubic ($n = 4$), quartic ($n = 5$), and quintic ($n = 6$) polynomials. Note the difference in the scales for the $m = 14$ and $m = 140$ cases.

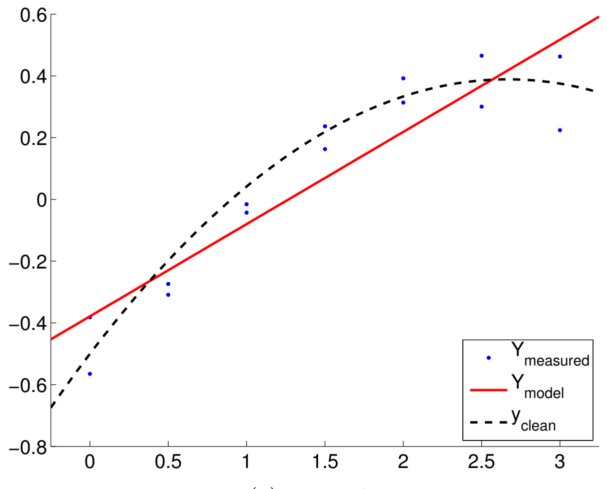
Note that the affine model is incapable of representing the quadratic dependency even in the absence of noise. Thus, comparing Figure 19.13(a) and 19.13(b), the fit does not improve with the number of sampling points.

Figure 19.13(c) shows typical individual confidence intervals for the affine model ($n = 2$) using $m = 14$ and $m = 140$ sampling points. Typical confidence intervals for the quadratic model ($n = 3$) are also provided for comparison. Let us first focus on analyzing the fit of the affine model ($n = 2$) using $m = 14$ sampling points. We observe that this realization of confidence intervals I_0 and I_1 does not include the true parameters β_0^{true} and β_1^{true}, respectively. In fact, Figure 19.13(d) shows that only 37 of the 100 realizations of the confidence interval I_0 include β_0^{true} and that none of the realizations of I_1 include β_1^{true}. Thus the frequency that the true value lies in the confidence interval is significantly lower than 95%. This is due to the presence of the bias error, which violates our assumptions about the behavior of the noise — the assumptions on which our confidence interval estimate rely. In fact, as we increase the number of sampling point from $m = 14$ to $m = 140$ we see that the confidence intervals for both β_0 and β_1 tighten; however, they converge toward wrong values. Thus, in the presence of bias, the confidence intervals are unreliable, and their convergence implies little about the quality of the estimates.

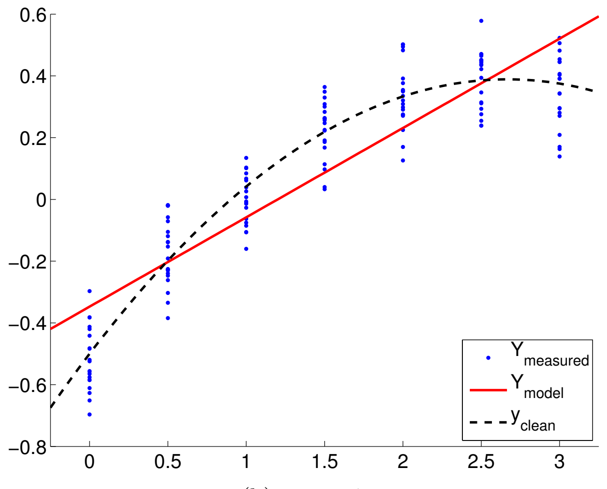
Let us now consider the second case with $c = 1/10$. This case results in a much weaker quadratic dependency of Y on x. Typical fits obtained using the affine model are shown in Figure 19.14(a) and 19.14(b) for $m = 14$ and $m = 140$ sampling points, respectively. Note that the fit is better than the $c = 1$ case because the $c = 1/10$ data can be better represented using the affine model.

Typical confidence intervals, shown in Figure 19.14(c), confirm that the confidence intervals are more reliable than in the $c = 1$ case. Of the 100 realizations for the $m = 14$ case, 87% and 67% of the confidence intervals include the true values β_0^{true} and β_1^{true}, respectively. The frequencies are lower than the 95%, i.e., the confidence intervals are not as reliable as their pretension, due to the presence of bias. However, they are more reliable than the case with a stronger quadratic dependence, i.e. a stronger bias. Recall that a smaller bias leading to a smaller error is consistent with the deterministic error bounds we developed in the presence of bias.

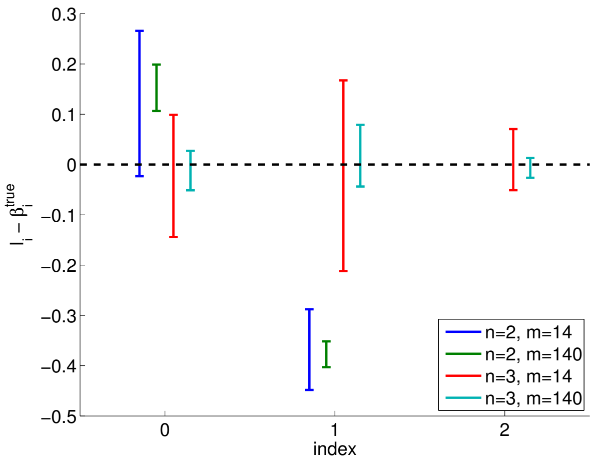
Similar to the $c = 1$ case, the confidence interval tightens with the number of samples m, but



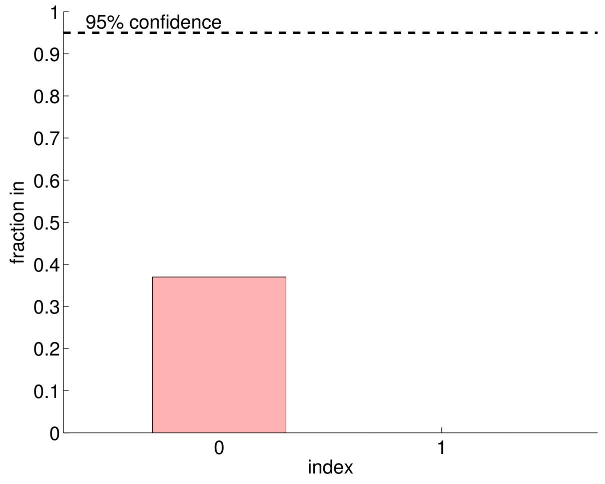
(a) $m = 14$



(b) $m = 140$

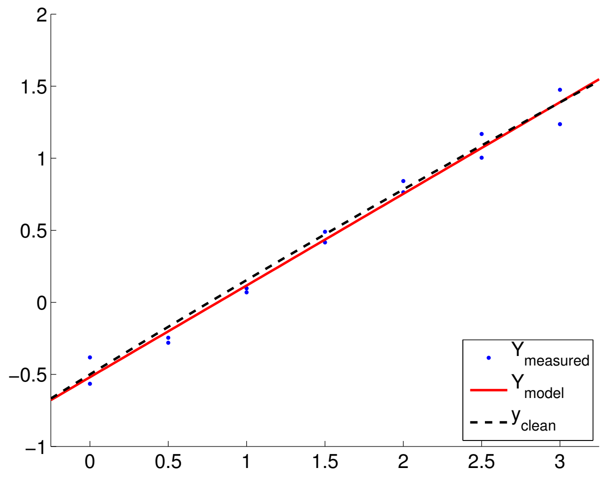


(c) 95% shifted confidence intervals

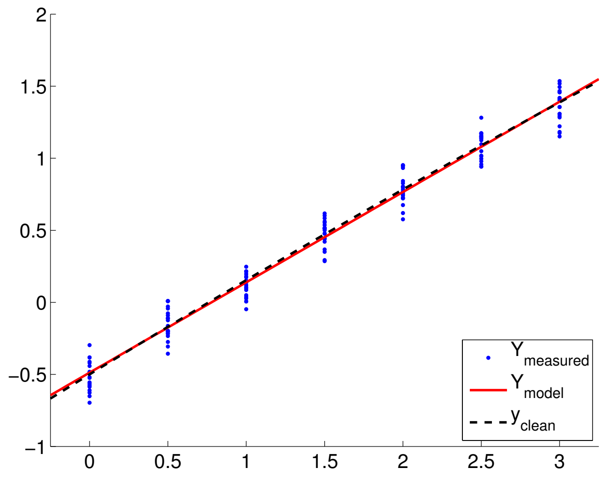


(d) 95% ci in/out (100 realizations, $m = 14$)

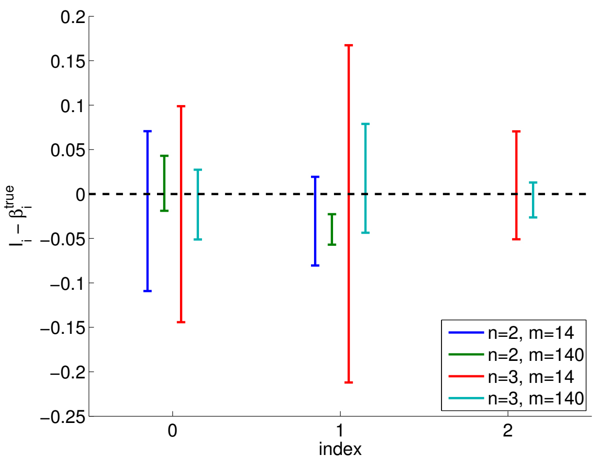
Figure 19.13: Least squares fitting of a quadratic function ($c = 1$) using an affine model.



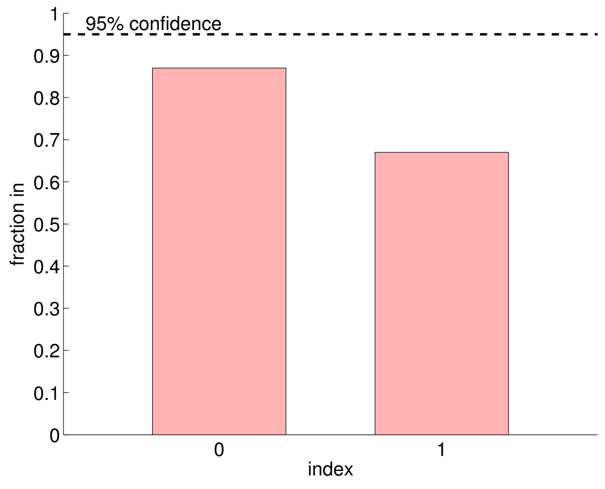
(a) $m = 14$



(b) $m = 140$



(c) 95% shifted confidence intervals



(d) 95% ci in/out (100 realizations, $m = 14$)

Figure 19.14: Least squares fitting of a quadratic function ($c = 1/10$) using an affine model.

they converge to a wrong value. Accordingly, the reliability of the confidence intervals decreases with m.

·

End Advanced Material

Unit IV

(Numerical) Differential Equations

Chapter 20

Motivation

Although mobile robots operating in flat, indoor environments can often perform quite well without any suspension, in uneven terrain, a well-designed suspension can be critical.

An actual robot suspension and its simplified model are shown in Figure 20.1. The rear and front springs with spring constants k_1 and k_2 serve to decouple the rest of the robot chassis from the wheels, allowing the chassis and any attached instrumentation to "float" relatively unperturbed while the wheels remain free to follow the terrain and maintain traction. The rear and front dampers with damping coefficients c_1 and c_2 (shown here inside the springs) dissipate energy to prevent excessive chassis displacements (e.g., from excitation of a resonant mode) and oscillations. Note that in our "half-robot" model, k_1 accounts for the *combined* stiffness of both rear wheels, and k_2 accounts for the combined stiffness of both front wheels. Similarly, c_1 and c_2 account for the combined damping coefficients of both rear wheels and both front wheels, respectively.

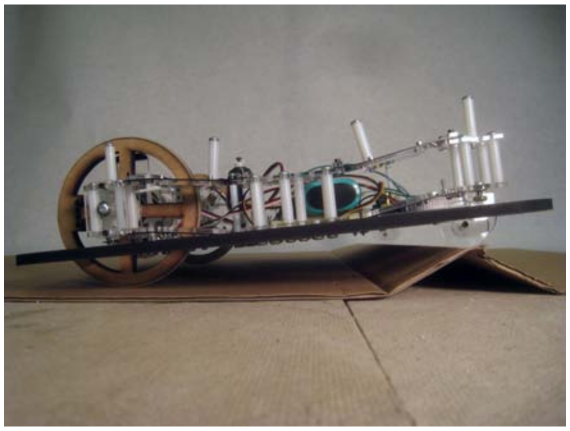
We are particularly concerned with the possibility of either the front or rear wheels losing contact with the ground, the consequences of which — loss of control and a potentially harsh landing — we wish to avoid.

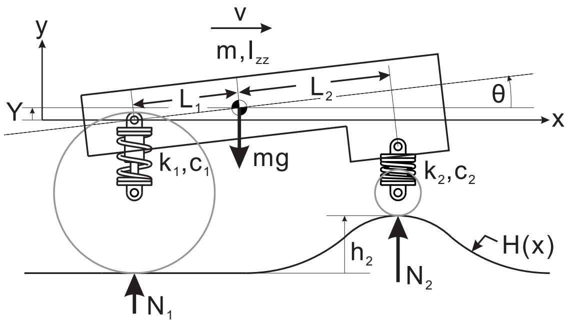
To aid in our understanding of robot suspensions and, in particular, to understand the conditions resulting in loss of contact, we wish to develop a simulation based on the simple model of Figure 20.1(b). Specifically, we wish to simulate the transient (time) response of the robot with suspension traveling at some constant velocity v over a surface with profile $H(x)$, the height of the ground as a function of x, and to check if loss of contact occurs. To do so, we must integrate the differential equations of motion for the system.

First, we determine the motion at the rear (subscript 1) and front (subscript 2) wheels in order to calculate the normal forces N_1 and N_2. Because we assume constant velocity v, we can determine the position in x of the center of mass at any time t (we assume $X(t = 0) = 0$) as

$$X = vt \; . \tag{20.1}$$

Given the current state Y, $\dot{Y}$, θ (the inclination of the chassis), and $\dot{\theta}$, we can then calculate the


(a) Actual robot suspension.



(b) Robot suspension model.

Figure 20.1: Mobile robot suspension

positions and velocities in both x and y at the rear and front wheels (assuming θ is small) as

$$\begin{aligned} X_1 &= X - L_1, \quad (\dot{X}_1 = v) , \\ X_2 &= X + L_2, \quad (\dot{X}_2 = v) , \\ Y_1 &= Y - L_1\theta , \\ \dot{Y}_1 &= \dot{Y} - L_1\dot{\theta} , \\ Y_2 &= Y + L_2\theta , \\ \dot{Y}_2 &= \dot{Y} + L_2\dot{\theta} , \end{aligned} \tag{20.2}$$

where L_1 and L_2 are the distances to the system's center of mass from the rear and front wheels. (Recall $\dot{}$ refers to time derivative.) Note that we define $Y = 0$ as the height of the robot's center of mass with both wheels in contact with flat ground and both springs at their unstretched and uncompressed lengths, i.e., when $N_1 = N_2 = 0$. Next, we determine the heights of the ground at the rear and front contact points as

$$\begin{aligned} h_1 &= H(X_1) , \\ h_2 &= H(X_2) . \end{aligned} \tag{20.3}$$

Similarly, the rates of change of the ground height at the rear and front are given by

$$\begin{aligned} \frac{dh_1}{dt} &= \dot{h}_1 = v\frac{d}{dx}H(X_1) , \\ \frac{dh_2}{dt} &= \dot{h}_2 = v\frac{d}{dx}H(X_2) . \end{aligned} \tag{20.4}$$

Note that we must multiply the spatial derivatives $\frac{dH}{dx}$ by $v = \frac{dX}{dt}$ to find the temporal derivatives.

While the wheels are in contact with the ground we can determine the normal forces at the rear and front from the constitutive equations for the springs and dampers as

$$\begin{aligned} N_1 &= k_1(h_1 - Y_1) + c_1(\dot{h}_1 - \dot{Y}_1) , \\ N_2 &= k_2(h_2 - Y_2) + c_2(\dot{h}_2 - \dot{Y}_2) . \end{aligned} \tag{20.5}$$

If either N_1 or N_2 is calculated from Equations (20.5) to be less than or equal to zero, we can determine that the respective wheel has lost contact with the ground and stop the simulation, concluding loss of contact, i.e., failure.

Finally, we can determine the rates of change of the state from the linearized ($\cos\theta \approx 1$, $\sin\theta \approx \theta$) equations of motion for the robot, given by Newton-Euler as

$$\begin{aligned} \ddot{X} &= 0, \quad \dot{X} = v, \quad X(0) = 0 \ , \\ \ddot{Y} &= -g + \frac{N_1 + N_2}{m}, \quad \dot{Y}(0) = \dot{Y}_0, \quad Y(0) = Y_0 \ , \\ \ddot{\theta} &= \frac{N_2 L_2 - N_1 L_1}{I_{\mathrm{zz}}}, \quad \dot{\theta}(0) = \dot{\theta}_0, \quad \theta(0) = \theta_0 \ , \end{aligned} \tag{20.6}$$

where m is the mass of the robot, and I_{zz} is the moment of inertia of the robot about an axis parallel to the Z axis passing through the robot's center of mass.

In this unit we shall discuss the numerical procedures by which to integrate systems of ordinary differential equations such as (20.6). This integration can then permit us to determine loss of contact and hence failure.

Chapter 21

Initial Value Problems

21.1 Scalar First-Order Linear ODEs

21.1.1 Model Problem

Let us consider a canonical initial value problem (IVP),

$$\frac{du}{dt} = \lambda u + f(t), \quad 0 < t < t_f \ ,$$

$$u(0) = u_0 \ .$$

The objective is to find u over all time $t \in \,]0, t_f]$ that satisfies the ordinary differential equation (ODE) and the initial condition. This problem belongs to the class of *initial value problems* (IVP) since we supplement the equation with condition(s) only at the initial time. The ODE is *first order* because the highest derivative that appears in the equation is the first-order derivative; because it is first order, only one initial condition is required to define a unique solution. The ODE is *linear* because the expression is linear with respect to u and its derivative du/dt; note that f does not have to be a linear function of t for the ODE to be linear. Finally, the equation is *scalar* since we have only a single unknown, $u(t) \in \mathbb{R}$. The coefficient $\lambda \in \mathbb{R}$ controls the behavior of the ODE; $\lambda < 0$ results in a stable (i.e. decaying) behavior, whereas $\lambda > 0$ results in an unstable (i.e. growing) behavior.

We can motivate this model problem (with $\lambda < 0$) physically with a simple heat transfer situation. We consider a body at initial temperature $u_0 > 0$ which is then "dunked" or "immersed" into a fluid flow — forced or natural convection — of ambient temperature (away from the body) zero. (More physically, we may view u_0 as the temperature elevation above some non-zero ambient temperature.) We model the heat transfer from the body to the fluid by a heat transfer coefficient, h. We also permit heat generation within the body, $\dot{q}(t)$, due (say) to Joule heating or radiation. If we now assume that the Biot number — the product of h and the body "diameter" in the numerator, thermal conductivity of the body in the denominator — is small, the temperature of the body will be roughly uniform in space. In this case, the temperature of the body as a function of time, $u(t)$, will be governed by our ordinary differential equation (ODE) initial value problem (IVP), with $\lambda = -h\,\text{Area}/\rho c\,\text{Vol}$ and $f(t) = \dot{q}(t)/\rho c\,\text{Vol}$, where ρ and c are the body density and specific heat, respectively, and Area and Vol are the body surface area and volume, respectively.

In fact, it is possible to express the solution to our model problem in closed form (as a convolution). Our interest in the model problem is thus not because we require a numerical solution procedure for this particular simple problem. Rather, as we shall see, our model problem will provide a foundation on which to construct and understand numerical procedures for much more difficult problems — which do *not* admit closed-form solution.

21.1.2 Analytical Solution

Before we pursue numerical methods for solving the IVP, let us study the analytical solution for a few cases which provide insight into the solution and also suggest test cases for our numerical approaches.

Homogeneous Equation

The first case considered is the homogeneous case, i.e., $f(t) = 0$. Without loss of generality, let us set $u_0 = 1$. Thus, we consider

$$\frac{du}{dt} = \lambda u, \quad 0 < t < t_f \ ,$$

$$u(0) = 1 \ .$$

We find the analytical solution by following the standard procedure for obtaining the homogeneous solution, i.e., substitute $u = \alpha e^{\beta t}$ to obtain

$$(\text{LHS}) = \frac{du}{dt} = \frac{d}{dt}(\alpha e^{\beta t}) = \alpha \beta e^{t} \ ,$$

$$(\text{RHS}) = \lambda \alpha e^{\beta t} \ .$$

Equating the LHS and RHS, we obtain $\beta = \lambda$. The solution is of the form $u(t) = \alpha e^{\lambda t}$. The coefficient α is specified by the initial condition, i.e.

$$u(t = 0) = \alpha = 1 \ ;$$

thus, the coefficient is $\alpha = 1$. The solution to the homogeneous ODE is

$$u(t) = e^{\lambda t} \ .$$

Note that solution starts from 1 (per the initial condition) and decays to zero for $\lambda < 0$. The decay rate is controlled by the time constant $1/|\lambda|$ — the larger the λ, the faster the decay. The solution for a few different values of λ are shown in Figure 21.1.

We note that for $\lambda > 0$ the solution grows exponentially in time: the system is unstable. (In actual fact, in most physical situations, at some point additional terms — for example, nonlinear effects not included in our simple model — would become important and ensure saturation in some steady state.) In the remainder of this chapter *unless specifically indicated otherwise* we shall assume that $\lambda < 0$.

Constant Forcing

Next, we consider a constant forcing case with $u_0 = 0$ and $f(t) = 1$, i.e.

$$\frac{du}{dt} = \lambda u + 1 \ ,$$

$$u_0 = 0 \ .$$

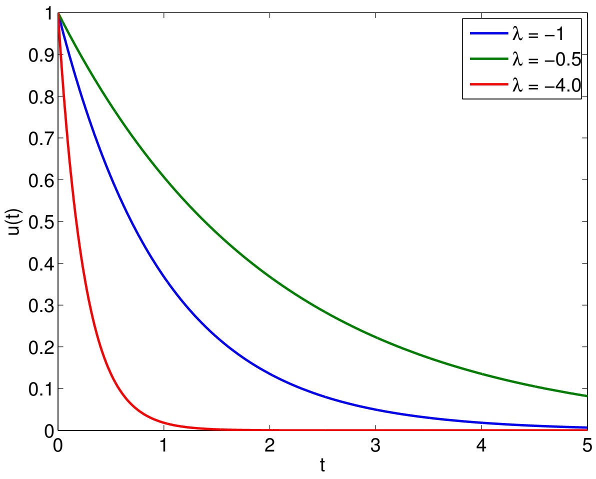


Figure 21.1: Solutions to the homogeneous equation.

We have already found the homogeneous solution to the ODE. We now find the particular solution. Because the forcing term is constant, we consider a particular solution of the form $u_p(t) = \gamma$. Substitution of u_p yields

$$0 = \lambda\gamma + 1 \quad \Rightarrow \quad \gamma = -\frac{1}{\lambda} \ .$$

Thus, our solution is of the form

$$u(t) = -\frac{1}{\lambda} + \alpha e^{\lambda t} \ .$$

Enforcing the initial condition,

$$u(t=0) = -\frac{1}{\lambda} + \alpha = 0 \quad \Rightarrow \quad \alpha = \frac{1}{\lambda} \ .$$

Thus, our solution is given by

$$u(t) = \frac{1}{\lambda}\left(e^{\lambda t} - 1\right) \ .$$

The solutions for a few different values of λ are shown in Figure 21.2. For $\lambda < 0$, after the transient which decays on the time scale $1/|\lambda|$, the solution settles to the steady state value of $-1/\lambda$.

Sinusoidal Forcing

Let us consider a final case with $u_0 = 0$ and a sinusoidal forcing, $f(t) = \cos(\omega t)$, i.e.

$$\frac{du}{dt} = \lambda u + \cos(\omega t) \ ,$$

$$u_0 = 0 \ .$$

Because the forcing term is sinusoidal with the frequency ω, the particular solution is of the form $u_p(t) = \gamma \sin(\omega t) + \delta \cos(\omega t)$. Substitution of the particular solution to the ODE yields

$$(\text{LHS}) = \frac{du_p}{dt} = \omega(\gamma \cos(\omega t) - \delta \sin(\omega t)) \ ,$$

$$(\text{RHS}) = \lambda(\gamma \sin(\omega t) + \delta \cos(\omega t)) + \cos(\omega t) \ .$$

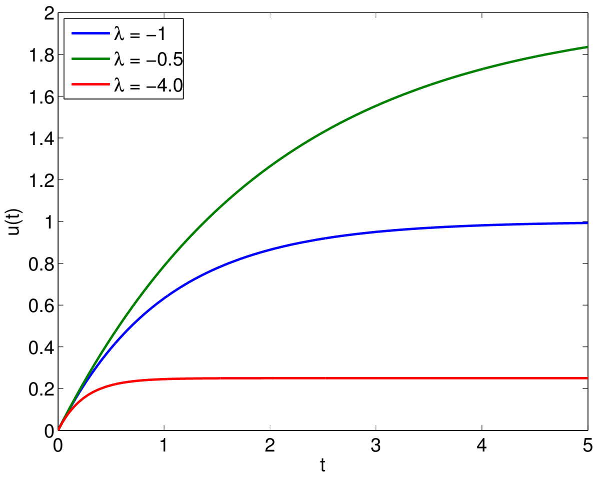


Figure 21.2: Solutions to the ODE with unit constant forcing.

Equating the LHS and RHS and collecting like coefficients we obtain

$$\omega\gamma = \lambda\delta + 1 \ ,$$

$$-\omega\delta = \lambda\gamma \ .$$

The solution to this linear system is given by $\gamma = \omega/(\omega^2 + \lambda^2)$ and $\delta = -\lambda/(\omega^2 + \lambda^2)$. Thus, the solution is of the form

$$u(t) = \frac{\omega}{\omega^2 + \lambda^2}\sin(\omega t) - \frac{\lambda}{\omega^2 + \lambda^2}\cos(\omega t) + \alpha e^{\lambda t} \ .$$

Imposing the boundary condition, we obtain

$$u(t = 0) = -\frac{\lambda}{\omega^2 + \lambda^2} + \alpha = 0 \quad \Rightarrow \quad \alpha = \frac{\lambda}{\omega^2 + \lambda^2} \ .$$

Thus, the solution to the IVP with the sinusoidal forcing is

$$u(t) = \frac{\omega}{\omega^2 + \lambda^2}\sin(\omega t) - \frac{\lambda}{\omega^2 + \lambda^2}\left(\cos(\omega t) - e^{\lambda t}\right) \ .$$

We note that for low frequency there is no phase shift; however, for high frequency there is a $\pi/2$ phase shift.

The solutions for $\lambda = -1$, $\omega = 1$ and $\lambda = -20$, $\omega = 1$ are shown in Figure 21.3. The steady state behavior is controlled by the sinusoidal forcing function and has the time scale of $1/\omega$. On the other hand, the initial transient is controlled by λ and has the time scale of $1/|\lambda|$. In particular, note that for $|\lambda| \gg \omega$, the solution exhibits very different time scales in the transient and in the steady (periodic) state. This is an example of a *stiff equation* (we shall see another example at the conclusion of this unit). Solving a stiff equation introduces additional computational challenges for numerical schemes, as we will see shortly.

21.1.3 A First Numerical Method: Euler Backward (Implicit)

In this section, we consider the Euler Backward integrator for solving initial value problems. We first introduce the time stepping scheme and then discuss a number of properties that characterize the scheme.

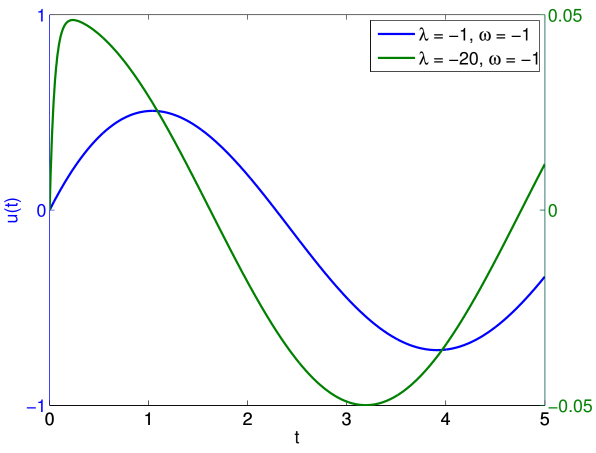


Figure 21.3: Solutions to the ODE with sinusoidal forcing.

Discretization

In order to solve an IVP numerically, we first discretize the time domain $]0, t_f]$ into J segments. The discrete time points are given by

$$t^j = j\Delta t, \quad j = 0, 1, \ldots, J = t_f/\Delta t \ ,$$

where Δt is the time step. For simplicity, we assume in this chapter that the time step is constant throughout the time integration.

The Euler Backward method is obtained by applying the first-order Backward Difference Formula (see Unit I) to the time derivative. Namely, we approximate the time derivative by

$$\frac{du}{dt} \approx \frac{\tilde{u}^j - \tilde{u}^{j-1}}{\Delta t} \ ,$$

where $\tilde{u}^j = \tilde{u}(t^j)$ is the approximation to $u(t^j)$ and $\Delta t = t^j - t^{j-1}$ is the time step. Substituting the approximation into the differential equation, we obtain a difference equation

$$\frac{\tilde{u}^j - \tilde{u}^{j-1}}{\Delta t} = \lambda \tilde{u}^j + f(t^j), \quad j = 1, \ldots, J \ ,$$

$$\tilde{u}^0 = u_0 \ ,$$

for $\tilde{u}^j$, $j = 0, \ldots, J$. Note the scheme is called "implicit" because time level j appears on the right-hand side. We can think of Euler Backward as a kind of rectangle, right integration rule — but now the integrand is not known *a priori*.

We anticipate that the solution $\tilde{u}^j$, $j = 1, \ldots, J$, approaches the true solution $u(t^j)$, $j = 1, \ldots, J$, as the time step gets smaller and the finite difference approximation approaches the continuous system. In order for this convergence to the true solution to take place, the discretization must possess two important properties: consistency and stability. Note our analysis here is more subtle than the analysis in Unit I. In Unit I we looked at the error in the finite difference approximation; here, we are interested in the error *induced* by the finite difference approximation on the approximate solution of the ODE IVP.

Consistency

Consistency is a property of a discretization that ensures that the discrete equation approximates the same process as the underlying ODE as the time step goes to zero. This is an important property, because if the scheme is not consistent with the ODE, then the scheme is modeling a different process and the solution would not converge to the true solution.

Let us define the notion of consistency more formally. We first define the *truncation error* by substituting the true solution $u(t)$ into the Euler Backward discretization, i.e.

$$\tau^j_{\text{trunc}} \equiv \frac{u(t^j) - u(t^{j-1})}{\Delta t} - \lambda u(t^j) - f(t^j), \quad j = 1, \ldots, J \ .$$

Note that the truncation error, τ^j_{trunc}, measures the extent to which the exact solution to the ODE does not satisfy the difference equation. In general, the exact solution does not satisfy the difference equation, so $\tau^j_{\text{trunc}} \neq 0$. In fact, as we will see shortly, if $\tau^j_{\text{trunc}} = 0$, $j = 1, \ldots, J$, then $\tilde{u}^j = u(t^j)$, i.e., $\tilde{u}^j$ is the exact solution to the ODE at the time points.

We are particularly interested in the largest of the truncation errors, which is in a sense the largest discrepancy between the differential equation and the difference equation. We denote this using the infinity norm,

$$\|\tau_{\text{trunc}}\|_\infty = \max_{j=1,\ldots,J} |\tau^j_{\text{trunc}}| \ .$$

A scheme is *consistent* with the ODE if

$$\|\tau_{\text{trunc}}\|_\infty \to 0 \quad \text{as} \quad \Delta t \to 0 \ .$$

The difference equation for a consistent scheme approaches the differential equation as $\Delta t \to 0$. However, this does not necessary imply that the solution to the difference equation, $\tilde{u}(t^j)$, approaches the solution to the differential equation, $u(t^j)$.

The Euler Backward scheme is consistent. In particular

$$\|\tau_{\text{trunc}}\|_\infty \leq \frac{\Delta t}{2} \max_{t \in [0, t_f]} \left| \frac{d^2 u}{dt^2}(t) \right| \to 0 \quad \text{as} \quad \Delta t \to 0 \ .$$

We demonstrate this result below.

Begin Advanced Material

Let us now analyze the consistency of the Euler Backward integration scheme. We first apply Taylor expansion to $u(t^{j-1})$ about t^j, i.e.

$$u(t^{j-1}) = u(t^j) - \Delta t \frac{du}{dt}(t^j) - \underbrace{\int_{t^{j-1}}^{t^j} \left(\int_{t^{j-1}}^{\tau} \frac{d^2 u}{dt^2}(\gamma) d\gamma \right) d\tau}_{s^j(u)} \ .$$

This result is simple to derive. By the fundamental theorem of calculus,

$$\int_{t^{j-1}}^{\tau} \frac{du^2}{dt^2}(\gamma) d\gamma = \frac{du}{dt}(\tau) - \frac{du}{dt}(t^{j-1}) \ .$$

Integrating both sides over $]t^{j-1}, t^j[$,

$$\int_{t^{j-1}}^{t^j} \left(\int_{t^{j-1}}^{\tau} \frac{du^2}{dt^2}(\gamma) d\gamma \right) d\tau = \int_{t^{j-1}}^{t^j} \left(\frac{du}{dt}(\tau) \right) d\tau - \int_{t^{j-1}}^{t^j} \left(\frac{du}{dt}(t^{j-1}) \right) d\tau$$

$$= u(t^j) - u(t^{j-1}) - (t^j - t^{j-1}) \frac{du}{dt}(t^{j-1})$$

$$= u(t^j) - u(t^{j-1}) - \Delta t \frac{du}{dt}(t^{j-1}) \ .$$

Substitution of the expression to the right-hand side of the Taylor series expansion yields

$$u(t^j) - \Delta t \frac{du}{dt}(t^j) - s^j(u) = u(t^j) - \Delta t \frac{du}{dt}(t^j) - u(t^j) + u(t^{j-1}) + \Delta t \frac{du}{dt}(t^{j-1}) = u(t^{j-1}) \ ,$$

which proves the desired result.

Substituting the Taylor expansion into the expression for the truncation error,

$$\tau^j_{\text{trunc}} = \frac{u(t^j) - u(t^{j-1})}{\Delta t} - \lambda u(t^j) - f(t^j)$$

$$= \frac{1}{\Delta t} \left(u(t^j) - \left(u(t^j) - \Delta t \frac{du}{dt}(t^j) - s^j(u) \right) \right) - \lambda u(t^j) - f(t^j)$$

$$= \underbrace{\frac{du}{dt}(t^j) - \lambda u(t^j) - f(t^j)}_{=0\,:\,\text{by ODE}} + \frac{s^j(u)}{\Delta t}$$

$$= \frac{s^j(u)}{\Delta t} \ .$$

We now bound the remainder term $s^j(u)$ as a function of Δt. Note that

$$s^j(u) = \int_{t^{j-1}}^{t^j} \left(\int_{t^{j-1}}^{\tau} \frac{d^2u}{dt^2}(\gamma) d\gamma \right) d\tau \le \int_{t^{j-1}}^{t^j} \left(\int_{t^{j-1}}^{\tau} \left| \frac{d^2u}{dt^2}(\gamma) \right| d\gamma \right) d\tau$$

$$\le \max_{t \in [t^{j-1}, t^j]} \left| \frac{d^2u}{dt^2}(t) \right| \int_{t^{j-1}}^{t^j} \int_{t^{j-1}}^{\tau} d\gamma d\tau$$

$$= \max_{t \in [t^{j-1}, t^j]} \left| \frac{d^2u}{dt^2}(t) \right| \frac{\Delta t^2}{2}, \quad j = 1, \ldots, J \ .$$

So, the maximum truncation error is

$$\|\tau_{\text{trunc}}\|_\infty = \max_{j=1,\ldots,J} |\tau^j_{\text{trunc}}| \le \max_{j=1,\ldots,J} \left(\frac{1}{\Delta t} \max_{t \in [t^{j-1}, t^j]} \left| \frac{d^2u}{dt^2}(t) \right| \frac{\Delta t^2}{2} \right) \le \frac{\Delta t}{2} \max_{t \in [0, t_f]} \left| \frac{d^2u}{dt^2}(t) \right| \ .$$

We see that

$$\|\tau_{\text{trunc}}\|_\infty \le \frac{\Delta t}{2} \max_{t \in [0, t_f]} \left| \frac{d^2u}{dt^2}(t) \right| \to 0 \quad \text{as} \quad \Delta t \to 0 \ .$$

Thus, the Euler Backward scheme is consistent.

End Advanced Material

Stability

Stability is a property of a discretization that ensures that the error in the numerical approximation does not grow with time. This is an important property, because it ensures that a small truncation error introduced at each time step does not cause a catastrophic divergence in the solution over time.

To study stability, let us consider a homogeneous IVP,

$$\frac{du}{dt} = \lambda u \ ,$$

$$u(0) = 1 \ .$$

Recall that the true solution is of the form $u(t) = e^{\lambda t}$ and decays for $\lambda < 0$. Applying the Euler Backward scheme, we obtain

$$\frac{\tilde{u}^j - \tilde{u}^{j-1}}{\Delta t} = \lambda \tilde{u}^j, \quad j = 1, \dots, J \ ,$$

$$u^0 = 1 \ .$$

A scheme is said to be absolutely stable if

$$|\tilde{u}^j| \le |\tilde{u}^{j-1}|, \quad j = 1, \dots, J \ .$$

Alternatively, we can define the amplification factor, γ, as

$$\gamma \equiv \frac{|\tilde{u}^j|}{|\tilde{u}^{j-1}|} \ .$$

Absolute stability requires that $\gamma \le 1$ for all $j = 1, \dots, J$.

Let us now show that the Euler Backward scheme is stable for all Δt (for $\lambda < 0$). Rearranging the difference equation,

$$\tilde{u}^j - \tilde{u}^{j-1} = \lambda \Delta t \, \tilde{u}^j$$

$$\tilde{u}^j (1 - \lambda \Delta t) = \tilde{u}^{j-1}$$

$$|\tilde{u}^j| \, |1 - \lambda \Delta t| = |\tilde{u}^{j-1}| \ .$$

So, we have

$$\gamma = \frac{|\tilde{u}^j|}{|\tilde{u}^{j-1}|} = \frac{1}{|1 - \lambda \Delta t|} \ .$$

Recalling that $\lambda < 0$ (and $\Delta t > 0$), we have

$$\gamma = \frac{1}{1 - \lambda \Delta t} < 1 \ .$$

Thus, the Euler Backward scheme is stable for all Δt for the model problem considered. The scheme is said to be *unconditionally stable* because it is stable for all Δt. Some schemes are only *conditionally stable*, which means the scheme is stable for $\Delta t \le \Delta t_{\text{cr}}$, where Δt_{cr} is some critical time step.

Convergence: Dahlquist Equivalence Theorem

Now we define the notion of convergence. A scheme is convergent if the numerical approximation approaches the analytical solution as the time step is reduced. Formally, this means that

$$\tilde{u}^j \equiv \tilde{u}(t^j) \to u(t^j) \quad \text{for fixed } t^j \text{ as } \Delta t \to 0 \ .$$

Note that fixed time t^j means that the time index must go to infinity (i.e., an infinite number of time steps are required) as $\Delta t \to 0$ because $t^j = j\Delta t$. Thus, convergence requires that not too much error is accumulated at each time step. Furthermore, the error generated at a given step should not grow over time.

The relationship between consistency, stability, and convergence is summarized in the Dahlquist equivalence theorem. The theorem states that consistency and stability are the necessary and sufficient condition for a convergent scheme, i.e.

$$\text{consistency} + \text{stability} \Leftrightarrow \text{convergence} \ .$$

Thus, we only need to show that a scheme is consistent and (absolutely) stable to show that the scheme is convergent. In particular, the Euler Backward scheme is convergent because it is consistent and (absolutely) stable.

Begin Advanced Material

Example 21.1.1 Consistency, stability, and convergence for Euler Backward
In this example, we will study in detail the relationship among consistency, stability, and convergence for the Euler Backward scheme. Let us denote the error in the solution by e^j,

$$e^j \equiv u(t^j) - \tilde{u}(t^j) \ .$$

We first relate the evolution of the error to the truncation error. To begin, we recall that

$$u(t^j) - u(t^{j-1}) - \lambda \Delta t u(t^j) - \Delta t f(t^j) = \Delta t \tau^j_{\text{trunc}} \ ,$$

$$\tilde{u}(t^j) - \tilde{u}(t^{j-1}) - \lambda \Delta t \tilde{u}(t^j) - \Delta t f(t^j) = 0 \ ;$$

subtracting these two equations and using the definition of the error we obtain

$$e^j - e^{j-1} - \lambda \Delta t e^j = \Delta t \tau^j_{\text{trunc}} \ ,$$

or, rearranging the equation,

$$(1 - \lambda \Delta t) e^j - e^{j-1} = \Delta t \tau^j_{\text{trunc}} \ .$$

We see that the error itself satisfies the Euler Backward difference equation with the truncation error as the source term. Clearly, if the truncation error τ^j_{trunc} is zero for all time steps (and initial error is zero), then the error remains zero. In other words, if the truncation error is zero then the scheme produces the exact solution at each time step.

However, in general, the truncation error is nonzero, and we would like to analyze its influence on the error. Let us multiply the equation by $(1 - \lambda \Delta t)^{j-1}$ to get

$$(1 - \lambda \Delta t)^j e^j - (1 - \lambda \Delta t)^{j-1} e^{j-1} = (1 - \lambda \Delta t)^{j-1} \Delta t \tau^j_{\text{trunc}} \ ,$$

Now, let us compute the sum for $j = 1, \ldots, n$, for some $n \leq J$,

$$\sum_{j=1}^{n} \Big[(1-\lambda\Delta t)^j e^j - (1-\lambda\Delta t)^{j-1} e^{j-1} \Big] = \sum_{j=1}^{n} \Big[(1-\lambda\Delta t)^{j-1} \Delta t \tau_{\text{trunc}}^j \Big] .$$

This is a telescopic series and all the middle terms on the left-hand side cancel. More explicitly,

$$(1-\lambda\Delta t)^n e^n - (1-\lambda\Delta t)^{n-1} e^{n-1} = (1-\lambda\Delta t)^{n-1} \Delta t \tau_{\text{trunc}}^n$$

$$(1-\lambda\Delta t)^{n-1} e^{n-1} - (1-\lambda\Delta t)^{n-2} e^{n-2} = (1-\lambda\Delta t)^{n-2} \Delta t \tau_{\text{trunc}}^{n-1}$$

$$\vdots$$

$$(1-\lambda\Delta t)^2 e^2 - (1-\lambda\Delta t)^1 e^1 = (1-\lambda\Delta t)^1 \Delta t \tau_{\text{trunc}}^2$$

$$(1-\lambda\Delta t)^1 e^1 - (1-\lambda\Delta t)^0 e^0 = (1-\lambda\Delta t)^0 \Delta t \tau_{\text{trunc}}^1$$

simplifies to

$$(1-\lambda\Delta t)^n e^n - e^0 = \sum_{j=1}^{n} (1-\lambda\Delta t)^{j-1} \Delta t \tau_{\text{trunc}}^j .$$

Recall that we set $\tilde{u}^0 = \tilde{u}(t^0) = u(t^0)$, so the initial error is zero ($e^0 = 0$). Thus, we are left with

$$(1-\lambda\Delta t)^n e^n = \sum_{j=1}^{n} (1-\lambda\Delta t)^{j-1} \Delta t \tau_{\text{trunc}}^j$$

or, equivalently,

$$e^n = \sum_{j=1}^{n} (1-\lambda\Delta t)^{j-n-1} \Delta t \tau_{\text{trunc}}^j .$$

Recalling that $\|\tau_{\text{trunc}}\|_\infty = \max_{j=1,\ldots,J} |\tau_{\text{trunc}}^j|$, we can bound the error by

$$|e^n| \leq \Delta t \|\tau_{\text{trunc}}\|_\infty \sum_{j=1}^{n} (1-\lambda\Delta t)^{j-n-1} .$$

Recalling the amplification factor for the Euler Backward scheme, $\gamma = 1/(1-\lambda\Delta t)$, the summation can be rewritten as

$$\begin{aligned} \sum_{j=1}^{n} (1-\lambda\Delta t)^{j-n-1} &= \frac{1}{(1-\lambda\Delta t)^n} + \frac{1}{(1-\lambda\Delta t)^{n-1}} + \cdots + \frac{1}{(1-\lambda\Delta t)} \\ &= \gamma^n + \gamma^{n-1} + \cdots + \gamma . \end{aligned}$$

Because the scheme is stable, the amplification factor satisfies $\gamma \leq 1$. Thus, the sum is bounded by

$$\sum_{j=1}^{n} (1-\lambda\Delta t)^{j-n-1} = \gamma^n + \gamma^{n-1} + \cdots + \gamma \leq n\gamma \leq n .$$

Thus, we have

$$|e^n| \le (n\Delta t)\|\tau_{\text{trunc}}\|_\infty = t^n \|\tau_{\text{trunc}}\|_\infty \ .$$

Furthermore, because the scheme is consistent, $\|\tau_{\text{trunc}}\|_\infty \to 0$ as $\Delta t \to 0$. Thus,

$$\|e^n\| \le t^n \|\tau_{\text{trunc}}\|_\infty \to 0 \quad \text{as} \quad \Delta t \to 0$$

for fixed $t^n = n\Delta t$. Note that the proof of convergence relies on stability ($\gamma \le 1$) and consistency ($\|\tau_{\text{trunc}}\|_\infty \to 0$ as $\Delta t \to 0$).

·

End Advanced Material

Order of Accuracy

The Dahlquist equivalence theorem shows that if a scheme is consistent and stable, then it is convergent. However, the theorem does not state how quickly the scheme converges to the true solution as the time step is reduced. Formally, a scheme is said to be p^{th}-order accurate if

$$|e^j| < C\Delta t^p \quad \text{for a fixed } t^j = j\Delta t \text{ as } \Delta t \to 0 \ .$$

The Euler Backward scheme is first-order accurate ($p = 1$), because

$$\|e^j\| \le t^j \|\tau_{\text{trunc}}\|_\infty \le t^j \frac{\Delta t}{2} \max_{t\in[0,t_f]} \left| \frac{d^2u}{dt^2}(t) \right| \le C\Delta t^1$$

with

$$C = \frac{t_f}{2} \max_{t\in[0,t_f]} \left| \frac{d^2u}{dt^2}(t) \right| .$$

(We use here $t^j \le t_f$.)

In general, for a stable scheme, if the truncation error is p^{th}-order accurate, then the scheme is p^{th}-order accurate, i.e.

$$\|\tau_{\text{trunc}}\|_\infty \le C\Delta t^p \quad \Rightarrow \quad |e^j| \le C\Delta t^p \quad \text{for a fixed } t^j = j\Delta t \ .$$

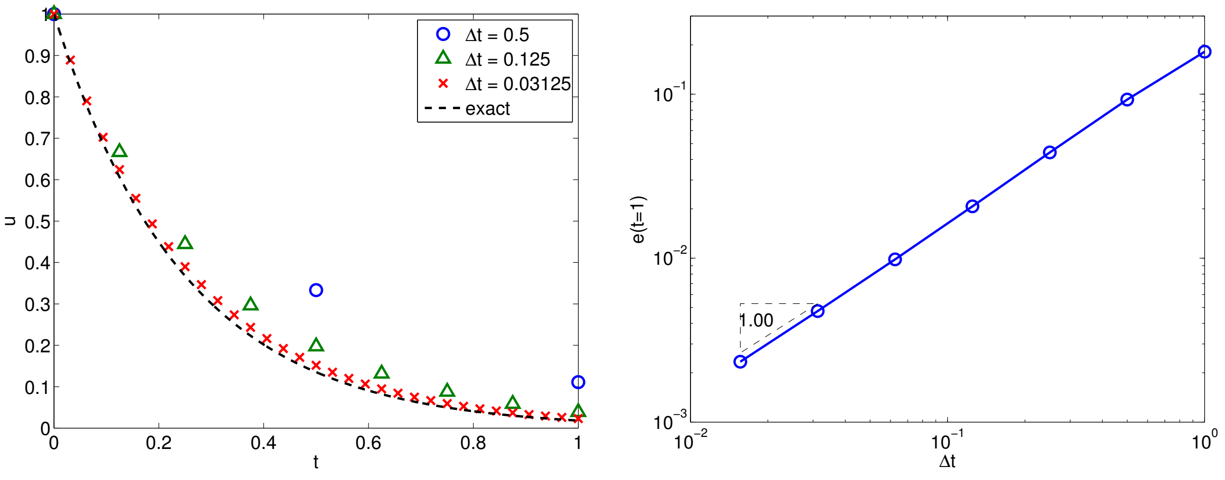
In other words, once we prove the stability of a scheme, then we just need to analyze its truncation error to understand its convergence rate. This requires little more work than checking for consistency. It is significantly simpler than deriving the expression for the evolution of the error and analyzing the error behavior directly.

Figure 21.4 shows the error convergence behavior of the Euler Backward scheme applied to the homogeneous ODE with $\lambda = -4$. The error is measured at $t = 1$. Consistent with the theory, the scheme converges at the rate of $p = 1$.

21.1.4 An Explicit Scheme: Euler Forward

Let us now introduce a new scheme, the Euler Forward scheme. The Euler Forward scheme is obtained by applying the first-order forward difference formula to the time derivative, i.e.

$$\frac{du}{dt} \approx \frac{\tilde{u}^{j+1} - \tilde{u}^j}{\Delta t} \ .$$



(a) solution (b) error

Figure 21.4: The error convergence behavior for the Euler Backward scheme applied to the homogeneous ODE ($\lambda = -4$). Note $e(t = 1) = |u(t^j) - \tilde{u}^j|$ at $t^j = j\Delta t = 1$.

Substitution of the expression to the linear ODE yields a difference equation,

$$\frac{\tilde{u}^{j+1} - \tilde{u}^j}{\Delta t} = \lambda \tilde{u}^j + f(t^j), \quad j = 0, \ldots, J-1 \ ,$$

$$\tilde{u}^0 = u_0 \ .$$

To maintain the same time index as the Euler Backward scheme (i.e., the difference equation involves the unknowns $\tilde{u}^j$ and $\tilde{u}^{j-1}$), let us shift the indices to obtain

$$\frac{\tilde{u}^j - \tilde{u}^{j-1}}{\Delta t} = \lambda \tilde{u}^{j-1} + f(t^{j-1}), \quad j = 1, \ldots, J \ ,$$

$$\tilde{u}^0 = u_0 \ .$$

The key difference from the Euler Backward scheme is that the terms on the right-hand side are evaluated at t^{j-1} instead of at t^j. Schemes for which the right-hand side does *not* involve time level j are known as "explicit" schemes. While the change may appear minor, this significantly modifies the stability. (It also changes the computational complexity, as we will discuss later.) We may view Euler Forward as a kind of "rectangle, left" integration rule.

Let us now analyze the consistency and stability of the scheme. The proof of consistency is similar to that for the Euler Backward scheme. The truncation error for the scheme is

$$\tau^j_{\text{trunc}} = \frac{u(t^j) - u(t^{j-1})}{\Delta t} - \lambda u(t^{j-1}) - f(t^{j-1}) \ .$$

To analyze the convergence of the truncation error, we apply Taylor expansion to $u(t^j)$ about t^{j-1} to obtain,

$$u(t^j) = u(t^{j-1}) + \Delta t \frac{du}{dt}(t^{j-1}) + \underbrace{\int_{t^{j-1}}^{t^j} \left(\int_{t^{j-1}}^{\tau} \frac{du^2}{dt^2}(\gamma) d\gamma \right) d\tau}_{s^j(u)} \ .$$

Thus, the truncation error simplifies to

$$\tau^j_{\text{trunc}} = \frac{1}{\Delta t}\left(u(t^{j-1}) + \Delta t \frac{du}{dt}(t^{j-1}) + s^j(u) - u(t^{j-1})\right) - \lambda u(t^{j-1}) - f(t^{j-1})$$

$$= \underbrace{\frac{du}{dt}(t^{j-1}) - \lambda u(t^{j-1}) - f(t^{j-1})}_{=0\,:\,\text{by ODE}} + \frac{s^j(u)}{\Delta t}$$

$$= \frac{s^j(u)}{\Delta t}\ .$$

In proving the consistency of the Euler Backward scheme, we have shown that $s^j(u)$ is bounded by

$$s^j(u) \le \max_{t\in[t^{j-1},t^j]} \left|\frac{d^2u}{dt^2}(t)\right| \frac{\Delta t^2}{2}, \quad j = 1, \dots, J\ .$$

Thus, the maximum truncation error is bounded by

$$\|\tau_{\text{trunc}}\|_\infty \le \max_{t\in[0,t_f]} \left|\frac{d^2u}{dt^2}(t)\right| \frac{\Delta t}{2}\ .$$

Again, the truncation error converges linearly with Δt and the scheme is consistent because $\|\tau_{\text{trunc}}\|_\infty \to 0$ as $\Delta t \to 0$. Because the scheme is consistent, we only need to show that it is stable to ensure convergence.

To analyze the stability of the scheme, let us compute the amplification factor. Rearranging the difference equation for the homogeneous case,

$$\tilde{u}^j - \tilde{u}^{j-1} = \lambda \Delta t \tilde{u}^{j-1}$$

or

$$|\tilde{u}^j| = |1 + \lambda\Delta t||\tilde{u}^{j-1}|$$

which gives

$$\gamma = |1 + \lambda \Delta t|\ .$$

Thus, absolute stability (i.e., $\gamma \le 1$) requires

$$-1 \le 1 + \lambda\Delta t \le 1$$

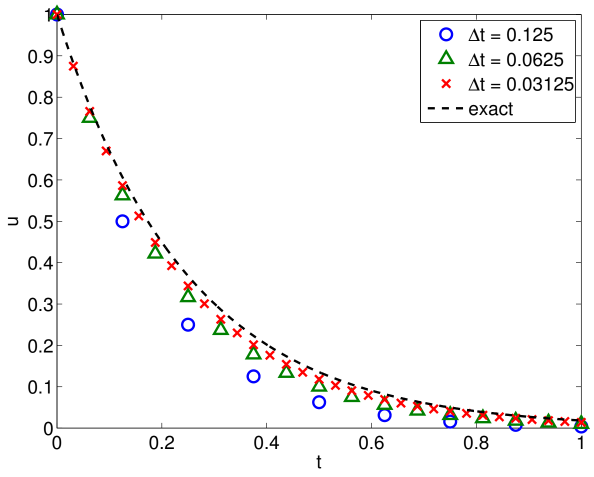
$$-2 \le \lambda\Delta t \le 0\ .$$

Noting $\lambda\Delta t \le 0$ is a trivial condition for $\lambda < 0$, the condition for stability is

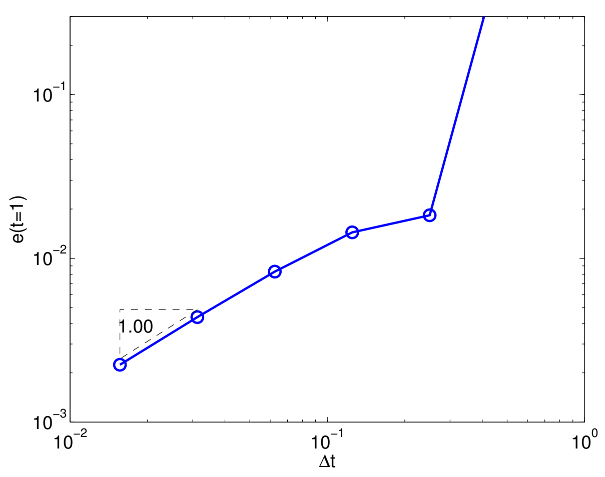
$$\Delta t \le -\frac{2}{\lambda} \equiv \Delta t_{\text{cr}}\ .$$

Note that the Euler Forward scheme is stable only for $\Delta t \le 2/|\lambda|$. Thus, the scheme is conditionally stable. Recalling the stability is a necessary condition for convergence, we conclude that the scheme converges for $\Delta t \le \Delta t_{\text{cr}}$, but diverges (i.e., blows up) with j if $\Delta t > \Delta t_{\text{cr}}$.

Figure 21.5 shows the error convergence behavior of the Euler Forward scheme applied to the homogeneous ODE with $\lambda = -4$. The error is measured at $t = 1$. The critical time step for stability is $\Delta t_{\text{cr}} = -2/\lambda = 1/2$. The error convergence plot shows that the error grows exponentially for



(a) solution



(b) error

Figure 21.5: The error convergence behavior for the Euler Forward scheme applied to $du/dt = -4u$. Note $e(t = 1) = |u(t^j) - \tilde{u}^j|$ at $t^j = j\Delta t = 1$.

$\Delta t > 1/2$. As Δt tends to zero, the numerical approximation converges to the exact solution, and the convergence rate (order) is $p = 1$ — consistent with the theory.

We should emphasize that the instability of the Euler Forward scheme for $\Delta t > \Delta t_{\mathrm{cr}}$ is *not* due to round-off errors and floating point representation (which involves "truncation," but not truncation of the variety discussed in this chapter). In particular, all of our arguments for instability hold in *infinite-precision arithmetic* as well as finite-precision arithmetic. The instability derives from the difference equation; the instability amplifies truncation error, which is a property of the difference equation and differential equation. Of course, an unstable difference equation will *also* amplify round-off errors, but that is an additional consideration and not the main reason for the explosion in Figure 21.5.

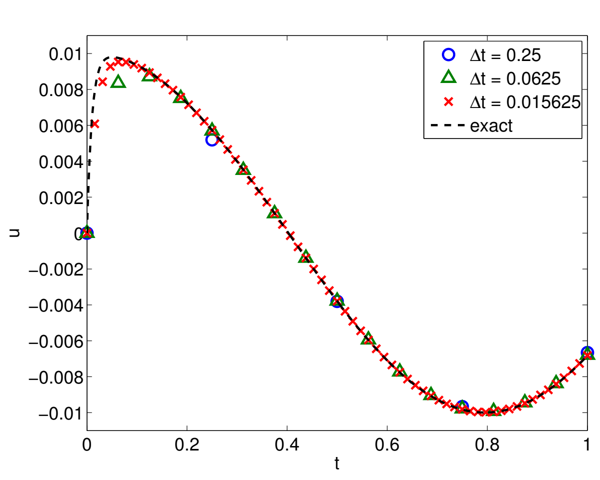
21.1.5 Stiff Equations: Implicit *vs*. Explicit

Stiff equations are the class of equations that exhibit a wide range of time scales. For example, recall the linear ODE with a sinusoidal forcing,

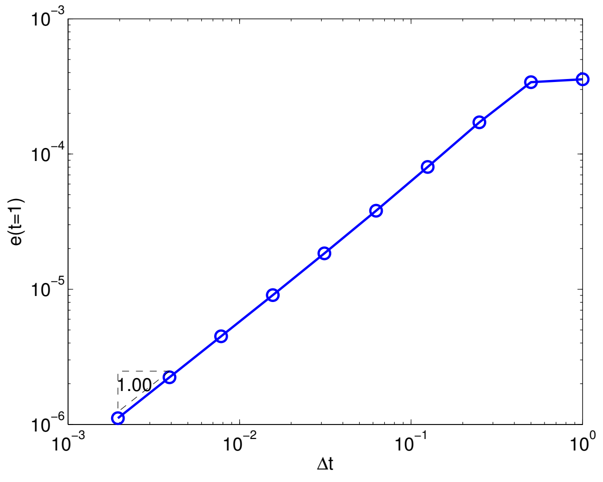
$$\frac{du}{dt} = \lambda t + \cos(\omega t) \ ,$$

with $|\lambda| \gg \omega$. The transient response of the solution is dictated by the time constant $1/|\lambda|$. However, this initial transient decays exponentially with time. The long time response is governed by the time constant $1/\omega \gg 1/|\lambda|$.

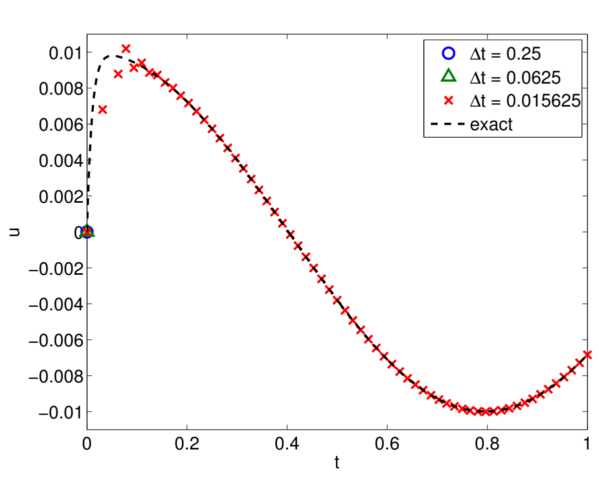
Let us consider the case with $\lambda = -100$ and $\omega = 4$; the time scales differ by a factor of 25. The result of applying the Euler Backward and Euler Forward schemes with several different time steps is shown in Figure 21.6. Recall that the Euler Backward scheme is stable for any time step for $\lambda < 0$. The numerical result confirms that the solution is bounded for all time steps considered. While a large time step (in particular $\Delta t > 1/|\lambda|$) results in an approximation which does not capture the initial transient, the long term behavior of the solution is still well represented. Thus, if the initial transient is not of interest, we can use a Δt optimized to resolve only the long term behavior associated with the characteristic time scale of $1/\omega$ — in other words, $\Delta t \sim O(1/10)$,



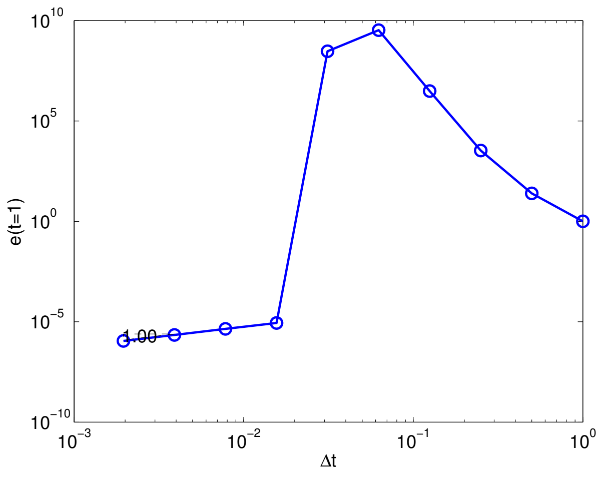
(a) Euler Backward (solution)



(b) Euler Backward (convergence)



(c) Euler Forward (solution)



(d) Euler Forward (convergence)

Figure 21.6: Application of the Euler Backward and Euler Forward schemes to a stiff equation. Note $e(t = 1) = |u(t^j) - \tilde{u}^j|$ at $t^j = j\Delta t = 1$.

rather than $\Delta t \sim O(1/|\lambda|)$. If $|\lambda| \gg \omega$, then we significantly reduce the number of time steps (and thus the computational cost).

Unlike its implicit counterpart, the Euler Forward method is only conditionally stable. In particular, the critical time step for this problem is $\Delta t_{\text{cr}} = 2/|\lambda| = 0.02$. Thus, even if we are not interested in the initial transient, we cannot use a large time step because the scheme would be unstable. Only one of the three numerical solution ($\Delta t = 1/64 < \Delta t_{\text{cr}}$) is shown in Figure 21.6(c) because the other two time steps ($\Delta t = 1/16$, $\Delta t = 1/4$) result in an unstable discretization and a useless approximation. The exponential growth of the error for $\Delta t > \Delta t_{\text{cr}}$ is clearly reflected in Figure 21.6(d).

Stiff equations are ubiquitous in the science and engineering context; in fact, it is not uncommon to see scales that differ by over ten orders of magnitude. For example, the time scale associated with the dynamics of a passenger jet is several orders of magnitude larger than the time scale associated with turbulent eddies. If the dynamics of the smallest time scale is not of interest, then an unconditionally stable scheme that allows us to take arbitrarily large time steps may be computationally advantageous. In particular, we can select the time step that is necessary to achieve sufficient accuracy without any time step restriction arising from the stability consideration. Put another way, integration of a stiff system using a conditionally stable method may place a stringent requirement on the time step, rendering the integration prohibitively expensive. As none of the explicit schemes are unconditionally stable, implicit schemes are often preferred for stiff equations.

We might conclude from the above that explicit schemes serve very little purpose. In fact, this is not the case, because the story is a bit more complicated. In particular, we note that for Euler Backward, at every time step, we must effect a division operation, $1/(1-(\lambda\Delta t))$, whereas for Euler Forward we must effect a multiplication, $1+(\lambda\Delta t)$. When we consider real problems of interest — systems, often large systems, of many and often nonlinear ODEs — these scalar algebraic operations of division for implicit schemes and multiplication for explicit schemes will translate into matrix inversion (more precisely, solution of matrix equations) and matrix multiplication, respectively. In general, and as we shall see in Unit V, matrix inversion is much more costly than matrix multiplication.

Hence the total cost equation is more nuanced. An implicit scheme will typically enjoy a larger time step and hence fewer time steps — but require more work for each time step (matrix solution). In contrast, an explicit scheme may require a much smaller time step and hence many more time steps — but will entail much less work for each time step. For stiff equations in which the Δt for accuracy is much, much larger than the Δt_{cr} required for stability (of explicit schemes), typically implicit wins. On the other hand, for non-stiff equations, in which the Δt for accuracy might be on the same order as Δt_{cr} required for stability (of explicit schemes), explicit can often win: in such cases we would in any event (for reasons of accuracy) choose a $\Delta t \approx \Delta t_{\text{cr}}$; hence, since an explicit scheme will be stable for this Δt, we might as well choose an explicit scheme to minimize the work per time step.

Begin Advanced Material

21.1.6 Unstable Equations

End Advanced Material

21.1.7 Absolute Stability and Stability Diagrams

We have learned that different integration schemes exhibit different stability characteristics. In particular, implicit methods tend to be more stable than explicit methods. To characterize the stability of different numerical integrators, let us introduce absolute *stability diagrams*. These diagrams allow us to quickly analyze whether an integration scheme will be stable for a given system.

Euler Backward

Let us construct the stability diagram for the Euler Backward scheme. We start with the homogeneous equation

$$\frac{dz}{dt} = \lambda z \ .$$

So far, we have only considered a real λ; now we allow λ to be a general complex number. (Later λ will represent an eigenvalue of a system, which in general will be a complex number.) The Euler

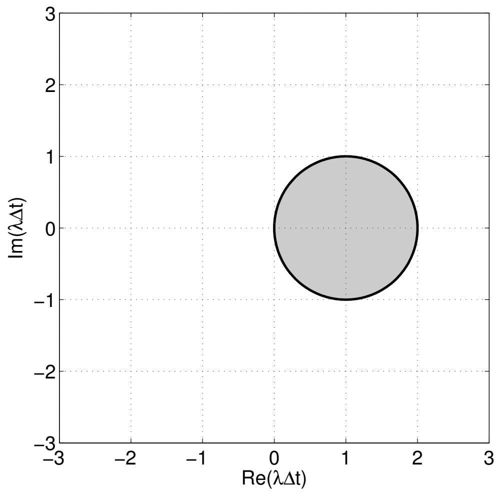


Figure 21.7: The absolute stability diagram for the Euler Backward scheme.

Backward discretization of the equation is

$$\frac{\tilde{z}^j - \tilde{z}^{j-1}}{\Delta t} = \lambda \tilde{z}^j \quad \Rightarrow \quad \tilde{z}^j = (1 - (\lambda \Delta t))^{-1} \tilde{z}^{j-1} \ .$$

Recall that we defined the absolute stability as the region in which the amplification factor $\gamma \equiv |\tilde{z}^j|/|\tilde{z}^{j-1}|$ is less than or equal to unity. This requires

$$\gamma = \frac{|\tilde{z}^j|}{|\tilde{z}^{j-1}|} = \left| \frac{1}{1 - (\lambda \Delta t)} \right| \leq 1 \ .$$

We wish to find the values of $(\lambda \Delta t)$ for which the numerical solution exhibits a stable behavior (i.e., $\gamma \leq 1$). A simple approach to achieve this is to solve for the stability *boundary* by setting the amplification factor to $1 = |e^{i\theta}|$, i.e.

$$e^{i\theta} = \frac{1}{1 - (\lambda \Delta t)} \ .$$

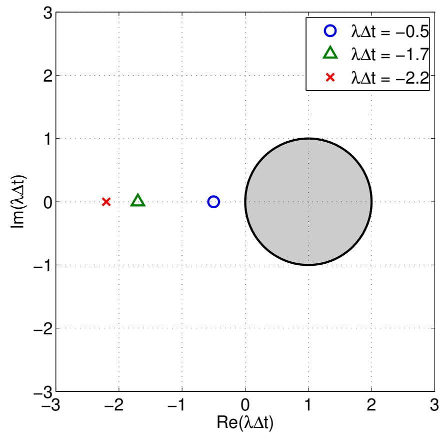
Solving for $(\lambda \Delta t)$, we obtain

$$(\lambda \Delta t) = 1 - e^{-i\theta} \ .$$

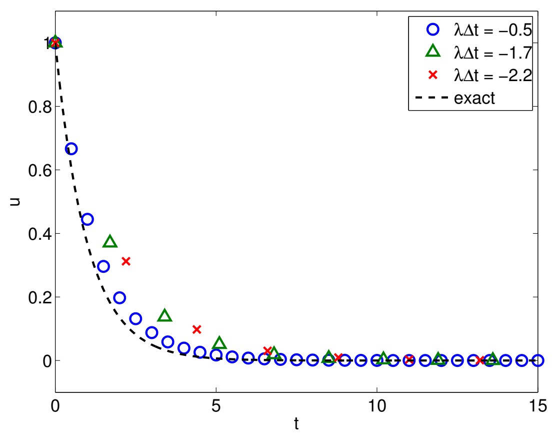
Thus, the stability boundary for the Euler Backward scheme is a circle of unit radius (the "one" multiplying $e^{i\theta}$) centered at 1 (the one directly after the = sign).

To deduce on which side of the boundary the scheme is stable, we can check the amplification factor evaluated at a point not on the circle. For example, if we pick $\lambda \Delta t = -1$, we observe that $\gamma = 1/2 \leq 1$. Thus, the scheme is stable *outside* of the unit circle. Figure 21.7 shows the stability diagram for the Euler Backward scheme. The scheme is *unstable* in the shaded region; it is *stable* in the unshaded region; it is *neutrally* stable, $|\tilde{z}^j| = |\tilde{z}^{j-1}|$, *on* the unit circle. The unshaded region ($\gamma < 1$) and the boundary of the shaded and unshaded regions ($\gamma = 1$) represent the absolute stability region; the entire picture is denoted the absolute stability diagram.

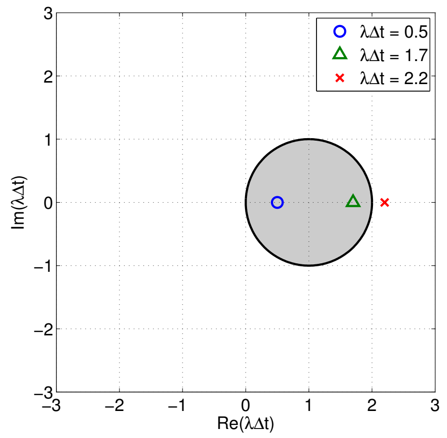
To gain understanding of the stability diagram, let us consider the behavior of the Euler Backward scheme for a few select values of $\lambda \Delta t$. First, we consider a stable homogeneous equation, with $\lambda = -1 < 0$. We consider three different values of $\lambda \Delta t$, -0.5, -1.7, and -2.2. Figure 21.8(a) shows



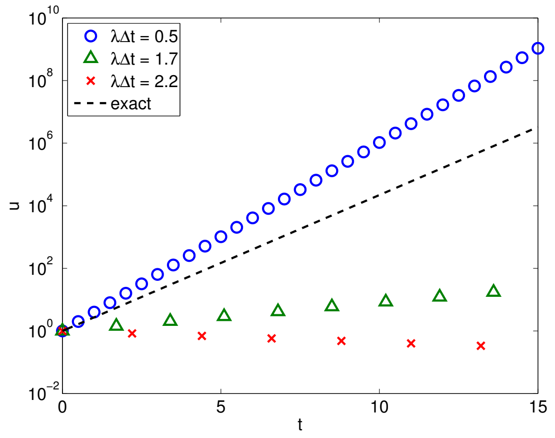
(a) $\lambda\Delta t$ for $\lambda = -1$



(b) solution ($\lambda = -1$)



(c) $\lambda\Delta t$ for $\lambda = 1$



(d) solution ($\lambda = 1$)

Figure 21.8: The behavior of the Euler Backward scheme for selected values of $(\lambda\Delta t)$.

the three points on the stability diagram that correspond to these choices of $\lambda\Delta t$. All three points lie in the unshaded region, which is a stable region. Figure 21.8(b) shows that all three numerical solutions decay with time as expected. While the smaller Δt results in a smaller error, all schemes are stable and converge to the same steady state solution.

Begin Advanced Material

Next, we consider an unstable homogeneous equation, with $\lambda = 1 > 0$. We again consider three different values of $\lambda\Delta t$, 0.5, 1.7, and 2.2. Figure 21.8(c) shows that two of these points lie in the unstable region, while $\lambda\Delta t = 2.2$ lies in the stable region. Figure 21.8(d) confirms that the solutions for $\lambda\Delta t = 0.5$ and 1.7 grow with time, while $\lambda\Delta t = 2.2$ results in a decaying solution. The true solution, of course, grows exponentially with time. Thus, if the time step is too large (specifically $\lambda\Delta t > 2$), then the Euler Backward scheme can produce a decaying solution even if the true solution grows with time — which is undesirable; nevertheless, as $\Delta t \to 0$, we obtain the correct behavior. In general, the interior of the absolute stability region should not include $\lambda\Delta t = 0$. (In fact $\lambda\Delta t = 0$ should be on the stability boundary.)

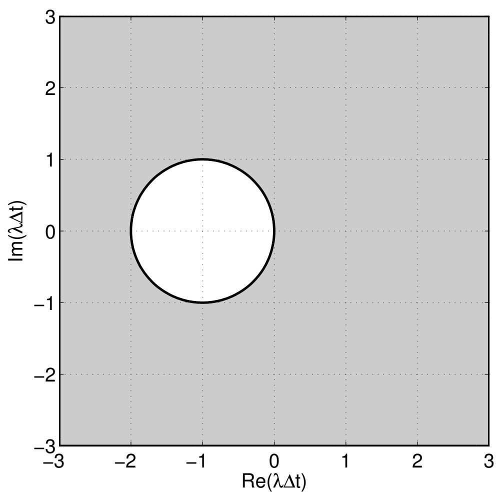


Figure 21.9: The absolute stability diagram for the Euler Forward scheme. The white area corresponds to stability (the absolute stability region) and the gray area to instability.

End Advanced Material

Euler Forward

Let us now analyze the absolute stability characteristics of the Euler Forward scheme. Similar to the Euler Backward scheme, we start with the homogeneous equation. The Euler Forward discretization of the equation yields

$$\frac{\tilde{z}^j - \tilde{z}^{j-1}}{\Delta t} = \lambda \tilde{z}^{j-1} \quad \Rightarrow \quad \tilde{z}^j = (1 + (\lambda \Delta t))\tilde{z}^{j-1} \ .$$

The stability boundary, on which the amplification factor is unity, is given by

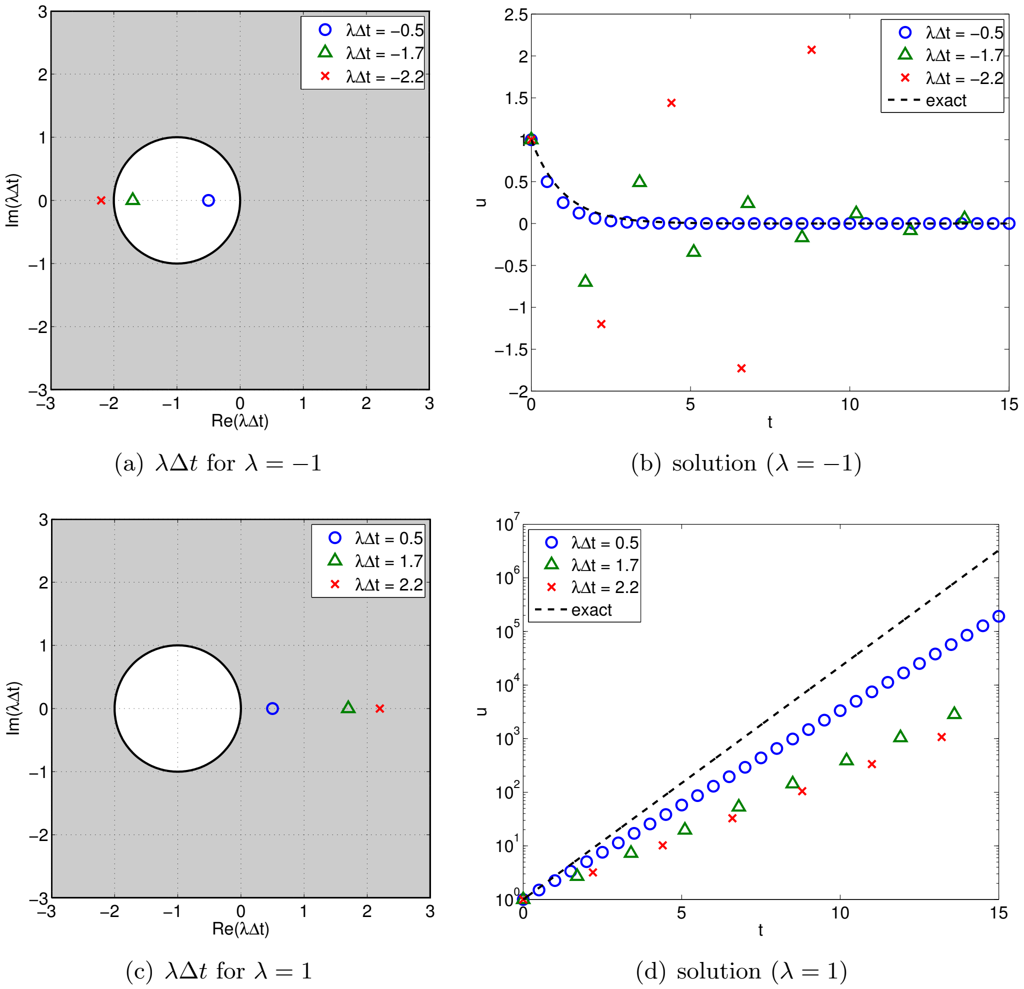
$$\gamma = |1 + (\lambda \Delta t)| = 1 \quad \Rightarrow \quad (\lambda \Delta t) = e^{-i\theta} - 1 \ .$$

The stability boundary is a circle of unit radius centered at -1. Substitution of, for example, $\lambda \Delta t = -1/2$, yields $\gamma(\lambda \Delta t = -1/2) = 1/2$, so the amplification is less than unity inside the circle. The stability diagram for the Euler Forward scheme is shown in Figure 21.9.

As in the Euler Backward case, let us pick a few select values of $\lambda \Delta t$ and study the behavior of the Euler Forward scheme. The stability diagram and solution behavior for a stable ODE ($\lambda = -1 < 0$) are shown in Figure 21.10(a) and 21.10(b), respectively. The cases with $\lambda \Delta t = -0.5$ and -1.7 lie in the stable region of the stability diagram, while $\lambda \Delta t = -2.2$ lies in the unstable region. Due to instability, the numerical solution for $\lambda \Delta t = -2.2$ diverges exponentially with time, even though the true solution decays with time. The solution for $\lambda \Delta t = -1.7$ shows some oscillation, but the magnitude of the oscillation decays with time, agreeing with the stability diagram. (For an unstable ODE ($\lambda = 1 > 0$), Figure 21.10(c) shows that all time steps considered lie in the unstable region of the stability diagram. Figure 21.10(d) confirms that all these choices of Δt produce a growing solution.)

21.1.8 Multistep Schemes

We have so far considered two schemes: the Euler Backward scheme and the Euler Forward scheme. These two schemes compute the state $\tilde{u}^j$ from the previous state $\tilde{u}^{j-1}$ and the source function



(a) $\lambda\Delta t$ for $\lambda = -1$ (b) solution ($\lambda = -1$)

(c) $\lambda\Delta t$ for $\lambda = 1$ (d) solution ($\lambda = 1$)

Figure 21.10: The behavior of the Euler Forward scheme for selected values of $\lambda\Delta t$.

evaluated at t^j or t^{j-1}. The two schemes are special cases of *multistep schemes*, where the solution at the current time $\tilde{u}^j$ is approximated from the previous solutions. In general, for an ODE of the form

$$\frac{du}{dt} = g(u,t) \ ,$$

a K-step multistep scheme takes the form

$$\sum_{k=0}^{K} \alpha_k \tilde{u}^{j-k} = \Delta t \sum_{k=0}^{K} \beta_k g^{j-k}, \quad j = 1, \ldots, J \ ,$$

$$\tilde{u}^j = u_0 \ ,$$

where $g^{j-k} = g(\tilde{u}^{j-k}, t^{j-k})$. Note that the linear ODE we have been considering results from the choice $g(u,t) = \lambda u + f(t)$. A K-step multistep scheme requires solutions (and derivatives) at K previous time steps. Without loss of generality, we choose $\alpha_0 = 1$. A scheme is uniquely defined by choosing $2K+1$ coefficients, α_k, $k = 1, \ldots, K$, and β_k, $k = 0, \ldots, K$.

Multistep schemes can be categorized into implicit and explicit schemes. If we choose $\beta_0 = 0$, then $\tilde{u}^j$ does not appear on the right-hand side, resulting in an explicit scheme. As discussed before, explicit schemes are only conditionally stable, but are computationally less expensive per step. If we choose $\beta_0 \neq 0$, then $\tilde{u}^j$ appears on the right-hand side, resulting in an implicit scheme. Implicit schemes tend to be more stable, but are more computationally expensive per step, especially for a system of nonlinear ODEs.

Let us recast the Euler Backward and Euler Forward schemes in the multistep method framework.

Example 21.1.2 Euler Backward as a multistep scheme
The Euler Backward scheme is a 1-step method with the choices

$$\alpha_1 = -1, \quad \beta_0 = 1, \quad \text{and} \quad \beta_1 = 0 \ .$$

This results in

$$\tilde{u}^j - \tilde{u}^{j-1} = \Delta t g^j, \quad j = 1, \ldots, J \ .$$

·

Example 21.1.3 Euler Forward as a multistep scheme
The Euler Forward scheme is a 1-step method with the choices

$$\alpha_1 = -1, \quad \beta_0 = 0, \quad \text{and} \quad \beta_1 = 1 \ .$$

This results in

$$\tilde{u}^j - \tilde{u}^{j-1} = \Delta t g^{j-1}, \quad j = 1, \ldots, J \ .$$

·

Now we consider three families of multistep schemes: Adams-Bashforth, Adams-Moulton, and Backward Differentiation Formulas.

Adams-Bashforth Schemes

Adams-Bashforth schemes are explicit multistep time integration schemes ($\beta_0 = 0$). Furthermore, we restrict ourselves to

$$\alpha_1 = -1 \quad \text{and} \quad \alpha_k = 0, \quad k = 2, \ldots, K \ .$$

The resulting family of the schemes takes the form

$$\tilde{u}^j = \tilde{u}^{j-1} + \sum_{k=1}^{K} \beta_k g^{j-k} \ .$$

Now we must choose β_k, $k = 1, \ldots K$, to define a scheme. To choose the appropriate values of β_k, we first note that the true solution $u(t^j)$ and $u(t^{j-1})$ are related by

$$u(t^j) = u(t^{j-1}) + \int_{t^{j-1}}^{t^j} \frac{du}{dt}(\tau) d\tau = u(t^{j-1}) + \int_{t^{j-1}}^{t^j} g(u(\tau), \tau) d\tau \ . \qquad (21.1)$$

Then, we approximate the integrand $g(u(\tau), \tau)$, $\tau \in (t^{j-1}, t^j)$, using the values g^{j-k}, $k = 1, \ldots, K$. Specifically, we construct a $(K-1)^{\text{th}}$-degree polynomial $p(\tau)$ using the K data points, i.e.

$$p(\tau) = \sum_{k=1}^{K} \phi_k(\tau) g^{j-k} \ ,$$

where $\phi_k(\tau)$, $k = 1, \ldots, K$, are the Lagrange interpolation polynomials defined by the points t^{j-k}, $k = 1, \ldots, K$. Recalling the polynomial interpolation theory from Unit I, we note that the $(K-1)^{\text{th}}$-degree polynomial interpolant is K^{th}-order accurate for $g(u(\tau), \tau)$ sufficiently smooth, i.e.

$$p(\tau) = g(u(\tau), \tau) + \mathcal{O}(\Delta t^K) \ .$$

(Note in fact here we consider "extrapolation" of our interpolant.) Thus, we expect the order of approximation to improve as we incorporate more points given sufficient smoothness. Substitution of the polynomial approximation of the derivative to Eq. (21.1) yields

$$u(t^j) \approx u(t^{j-1}) + \int_{t^{j-1}}^{t^j} \sum_{k=1}^{K} \phi_k(\tau) g^{j-k} d\tau = u(t^{j-1}) + \sum_{k=1}^{K} \int_{t^{j-1}}^{t^j} \phi_k(\tau) d\tau \, g^{j-k} \ .$$

To simplify the integral, let us consider the change of variable $\tau = t^j - (t^j - t^{j-1})\hat{\tau} = t^j - \Delta t \hat{\tau}$. The change of variable yields

$$u(t^j) \approx u(t^{j-1}) + \Delta t \sum_{k=1}^{K} \int_0^1 \hat{\phi}_k(\hat{\tau}) d\hat{\tau} \, g^{j-k} \ ,$$

where the $\hat{\phi}_k$ are the Lagrange polynomials associated with the interpolation points $\hat{\tau} = 1, 2, \ldots, K$. We recognize that the approximation fits the Adams-Bashforth form if we choose

$$\beta_k = \int_0^1 \hat{\phi}_k(\hat{\tau}) d\hat{\tau} \ .$$

Let us develop a few examples of Adams-Bashforth schemes.

Example 21.1.4 1-step Adams-Bashforth (Euler Forward)
The 1-step Adams-Bashforth scheme requires evaluation of β_1. The Lagrange polynomial for this case is a constant polynomial, $\hat{\phi}_1(\hat{\tau}) = 1$. Thus, we obtain

$$\beta_1 = \int_0^1 \hat{\phi}_1(\hat{\tau}) d\hat{\tau} = \int_0^1 1 d\hat{\tau} = 1 \ .$$

Thus, the scheme is

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t g^{j-1} \ ,$$

which is the Euler Forward scheme, first-order accurate.

·

Example 21.1.5 2-step Adams-Bashforth
The 2-step Adams-Bashforth scheme requires specification of β_1 and β_2. The Lagrange interpolation polynomials for this case are linear polynomials

$$\hat{\phi}_1(\hat{\tau}) = -\hat{\tau} + 2 \quad \text{and} \quad \hat{\phi}_2(\hat{\tau}) = \hat{\tau} - 1 \ .$$

It is easy to verify that these are the Lagrange polynomials because $\hat{\phi}_1(1) = \hat{\phi}_2(2) = 1$ and $\hat{\phi}_1(2) = \hat{\phi}_2(1) = 0$. Integrating the polynomials

$$\beta_1 = \int_0^1 \phi_1(\hat{\tau}) d\hat{\tau} = \int_0^1 (-\hat{\tau} + 2) d\hat{\tau} = \frac{3}{2} \ ,$$

$$\beta_2 = \int_0^1 \phi_2(\hat{\tau}) d\hat{\tau} = \int_0^1 (\hat{\tau} - 1) d\hat{\tau} = -\frac{1}{2} \ .$$

The resulting scheme is

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \left(\frac{3}{2} g^{j-1} - \frac{1}{2} g^{j-2} \right) .$$

This scheme is second-order accurate.

·

Adams-Moulton Schemes

Adams-Moulton schemes are implicit multistep time integration schemes ($\beta_0 \neq 0$). Similar to Adams-Bashforth schemes, we restrict ourselves to

$$\alpha_1 = -1 \quad \text{and} \quad \alpha_k = 0, \quad k = 2, \ldots, K \ .$$

The Adams-Moulton family of the schemes takes the form

$$\tilde{u}^j = \tilde{u}^{j-1} + \sum_{k=0}^{K} \beta_k g^{j-k} \ .$$

We must choose β_k, $k = 1, \ldots, K$ to define a scheme. The choice of β_k follows exactly the same procedure as that for Adams-Bashforth. Namely, we consider the expansion of the form Eq. (21.1) and approximate $g(u(\tau), \tau)$ by a polynomial. This time, we have $K+1$ points, thus we construct a K^{th}-degree polynomial

$$p(\tau) = \sum_{k=0}^{K} \phi_k(\tau) g^{j-k} ,$$

where $\phi_k(\tau)$, $k = 0, \dots, K$, are the Lagrange interpolation polynomials defined by the points t^{j-k}, $k = 0, \dots, K$. Note that these polynomials are different from those for the Adams-Bashforth schemes due to the inclusion of t^j as one of the interpolation points. (Hence here we consider true interpolation, not extrapolation.) Moreover, the interpolation is now $(K+1)^{\text{th}}$-order accurate.

Using the same change of variable as for Adams-Bashforth schemes, $\tau = t^j - \Delta t \hat{\tau}$, we arrive at a similar expression,

$$u(t^j) \approx u(t^{j-1}) + \Delta t \sum_{k=0}^{K} \int_0^1 \hat{\phi}_k(\hat{\tau}) d\hat{\tau} g^{j-k} ,$$

for the Adams-Moulton schemes; here the $\hat{\phi}_k$ are the K^{th}-degree Lagrange polynomials defined by the points $\hat{\tau} = 0, 1, \dots, K$. Thus, the β_k are given by

$$\beta_k = \int_0^1 \hat{\phi}_k(\hat{\tau}) d\hat{\tau} .$$

Let us develop a few examples of Adams-Moulton schemes.

Example 21.1.6 0-step Adams-Moulton (Euler Backward)
The 0-step Adams-Moulton scheme requires just one coefficient, β_0. The "Lagrange" polynomial is 0^{th} degree, i.e. a constant function $\hat{\phi}_0(\hat{\tau}) = 1$, and the integration of the constant function over the unit interval yields

$$\beta_0 = \int_0^1 \hat{\phi}_0(\hat{\tau}) d\hat{\tau} = \int_0^1 1 d\hat{\tau} = 1.$$

Thus, the 0-step Adams-Moulton scheme is given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t g^j,$$

which in fact is the Euler Backward scheme. Recall that the Euler Backward scheme is first-order accurate.

·

Example 21.1.7 1-step Adams-Moulton (Crank-Nicolson)
The 1-step Adams-Moulton scheme requires determination of two coefficients, β_0 and β_1. The Lagrange polynomials for this case are linear polynomials

$$\hat{\phi}_0(\hat{\tau}) = -\tau + 1 \quad \text{and} \quad \hat{\phi}_1(\hat{\tau}) = \tau .$$

Integrating the polynomials,

$$\beta_0 = \int_0^1 \hat{\phi}_0(\hat{\tau}) d\hat{\tau} = \int_0^1 (-\tau + 1) d\hat{\tau} = \frac{1}{2} ,$$

$$\beta_1 = \int_0^1 \hat{\phi}_1(\hat{\tau}) d\hat{\tau} = \int_0^1 \hat{\tau} d\hat{\tau} = \frac{1}{2} .$$

The choice of β_k yields the Crank-Nicolson scheme

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \left(\frac{1}{2} g^j + \frac{1}{2} g^{j-1} \right) .$$

The Crank-Nicolson scheme is second-order accurate. We can view Crank-Nicolson as a kind of "trapezoidal" rule.

·

Example 21.1.8 2-step Adams-Moulton
The 2-step Adams-Moulton scheme requires three coefficients, β_0, β_1, and β_2. The Lagrange polynomials for this case are the quadratic polynomials

$$\hat{\phi}_0(\hat{\tau}) = \frac{1}{2}(\hat{\tau}-1)(\hat{\tau}-2) = \frac{1}{2}(\hat{\tau}^2 - 3\hat{\tau} + 2) \ ,$$

$$\hat{\phi}_1(\hat{\tau}) = -\hat{\tau}(\hat{\tau}-2) = -\hat{\tau}^2 + 2\hat{\tau} \ ,$$

$$\hat{\phi}_2(\hat{\tau}) = \frac{1}{2}\hat{\tau}(\hat{\tau}-1) = \frac{1}{2}\left(\hat{\tau}^2 - \hat{\tau}\right) .$$

Integrating the polynomials,

$$\beta_0 = \int_0^1 \hat{\phi}_0(\hat{\tau}) d\hat{\tau} = \int_0^1 \frac{1}{2}(\hat{\tau}^2 - 3\hat{\tau} + 2)\hat{\tau} = \frac{5}{12}$$

$$\beta_1 = \int_0^1 \hat{\phi}_1(\hat{\tau}) d\hat{\tau} = \int_0^1 (-\hat{\tau}^2 + 2\hat{\tau}) d\hat{\tau} = \frac{2}{3} \ ,$$

$$\beta_2 = \int_0^1 \hat{\phi}_2(\hat{\tau}) d\hat{\tau} = \int_0^1 \frac{1}{2}\left(\hat{\tau}^2 - \hat{\tau}\right) d\hat{\tau} = -\frac{1}{12} \ .$$

Thus, the 2-step Adams-Moulton scheme is given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \left(\frac{5}{12} g^j + \frac{2}{3} g^{j-1} - \frac{1}{12} g^{j-2} \right) .$$

This AM2 scheme is third-order accurate.

·

Convergence of Multistep Schemes: Consistency and Stability

Let us now introduce techniques for analyzing the convergence of a multistep scheme. Due to the Dahlquist equivalence theorem, we only need to show that the scheme is consistent and stable.

To show that the scheme is consistent, we need to compute the truncation error. Recalling that the local truncation error is obtained by substituting the exact solution to the difference equation (normalized such that $\tilde{u}^j$ has the coefficient of 1) and dividing by Δt, we have for any multistep schemes

$$\tau^j_{\text{trunc}} = \frac{1}{\Delta t} \left[u(t^j) + \sum_{k=1}^{K} \alpha_k \, u(t^{j-k}) \right] - \sum_{k=0}^{K} \beta_k \, g(t^{j-k}, u(t^{j-k})) \ .$$

For simplicity we specialize our analysis to the Adams-Bashforth family, such that

$$\tau^j_{\text{trunc}} = \frac{1}{\Delta t}\Big(u(t^j) - u(t^{j-1})\Big) - \sum_{k=1}^{K} \beta_k\, g(t^{j-k}, u(t^{j-k})) \ .$$

We recall that the coefficients β_k were selected to match the extrapolation from polynomial fitting. Backtracking the derivation, we simplify the sum as follows

$$\begin{aligned}\sum_{k=1}^{K} \beta_k\, g(t^{j-k}, u(t^{j-k})) &= \sum_{k=1}^{K} \int_0^1 \hat{\phi}_k(\hat{\tau}) d\hat{\tau}\; g(t^{j-k}, u(t^{j-k})) \\ &= \sum_{k=1}^{K} \frac{1}{\Delta t} \int_{t^{j-1}}^{t^j} \phi_k(\tau) d\tau\; g(t^{j-k}, u(t^{j-k})) \\ &= \frac{1}{\Delta t} \int_{t^{j-1}}^{t^j} \left[\sum_{k=1}^{K} \phi_k(\tau)\, g(t^{j-k}, u(t^{j-k})) \right] d\tau \\ &= \frac{1}{\Delta t} \int_{t^{j-1}}^{t^j} p(\tau) d\tau \ .\end{aligned}$$

We recall that $p(\tau)$ is a $(K-1)^{\text{th}}$-degree polynomial approximating $g(\tau, u(\tau))$. In particular, it is a K^{th}-order accurate interpolation with the error $\mathcal{O}(\Delta t^K)$. Thus,

$$\begin{aligned}\tau^j_{\text{trunc}} &= \frac{1}{\Delta t}\Big(u(t^j) - u(t^{j-1})\Big) - \sum_{k=1}^{K} \beta_k\, g(t^{j-k}, u(t^{j-k})) \\ &= \frac{1}{\Delta t}\Big(u(t^j) - u(t^{j-1})\Big) - \frac{1}{\Delta t}\int_{t^{j-1}}^{t^j} g(\tau, u(\tau)) d\tau + \frac{1}{\Delta t}\int_{j^{j-1}}^{t^j} \mathcal{O}(\Delta t^K) d\tau \\ &= \frac{1}{\Delta t}\left[u(t^j) - u(t^{j-1}) - \int_{t^{j-1}}^{t^j} g(\tau, u(\tau)) d\tau \right] + \mathcal{O}(\Delta t^K) \\ &= \mathcal{O}(\Delta t^K) \ .\end{aligned}$$

Note that the term in the bracket vanishes from $g = du/dt$ and the fundamental theorem of calculus. The truncation error of the scheme is $\mathcal{O}(\Delta t^K)$. In particular, since $K > 0$, $\tau_{\text{trunc}} \to 0$ as $\Delta t \to 0$ and the Adams-Bashforth schemes are consistent. Thus, if the schemes are stable, they would converge at Δt^K.

The analysis of stability relies on a solution technique for difference equations. We first restrict ourselves to linear equation of the form $g(t, u) = \lambda u$. By rearranging the form of difference equation for the multistep methods, we obtain

$$\sum_{k=0}^{K} (\alpha_k - (\lambda \Delta t)\, \beta_k)\, \tilde{u}^{j-k} = 0, \quad j = 1, \ldots, J \ .$$

The solution to the difference equation is governed by the initial condition and the K roots of the polynomial

$$q(x) = \sum_{k=0}^{K} (\alpha_k - (\lambda\Delta t)\, \beta_k) x^{K-k} \ .$$

In particular, for any initial condition, the solution will exhibit a stable behavior if all roots r_k, $k = 1, \dots, K$, have magnitude less than or equal to unity. Thus, the absolute stability condition for multistep schemes is

$$(\lambda\Delta t) \text{ such that } |r_K| \le 1, \quad k = 1, \dots, K \ ,$$

where r_k, $k = 1, \dots, K$ are the roots of q.

Example 21.1.9 Stability of the 2-step Adams-Bashforth scheme
Recall that the 2-step Adams-Bashforth results from the choice

$$\alpha_0 = 1, \quad \alpha_1 = -1, \quad \alpha_2 = 0, \quad \beta_0 = 0, \quad \beta_1 = \frac{3}{2}, \quad \text{and} \quad \beta_2 = -\frac{1}{2} \ .$$

The stability of the scheme is governed by the roots of the polynomial

$$q(x) = \sum_{k=0}^{2} (\alpha_k - (\lambda\Delta t)\, \beta_k) x^{2-k} = x^2 + \left(-1 - \frac{3}{2}(\lambda\Delta t)\right) x + \frac{1}{2}(\lambda\Delta t) = 0 \ .$$

The roots of the polynomial are given by

$$r_{1,2} = \frac{1}{2}\left[1 + \frac{3}{2}(\lambda\Delta t) \pm \sqrt{\left(1 + \frac{3}{2}(\lambda\Delta t)\right)^2 - 2(\lambda\Delta t)}\,\right] .$$

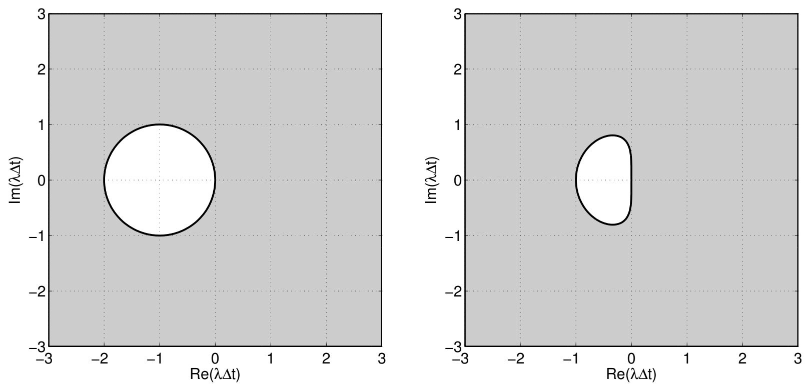
We now look for $(\lambda\Delta t)$ such that $|r_1| \le 1$ and $|r_2| \le 1$.

It is a simple matter to determine if a particular $\lambda\Delta t$ is inside, on the boundary of, or outside the absolute stability region. For example, for $\lambda\Delta t = -1$ we obtain $r_1 = -1$, $r_2 = 1/2$ and hence — since $|r_1| = 1$ — $\lambda\Delta t = -1$ is in fact on the boundary of the absolute stability diagram. Similarly, it is simple to confirm that $\lambda\Delta t = -1/2$ yields both r_1 and r_2 of modulus strictly less than 1, and hence $\lambda\Delta t = -1/2$ is inside the absolute stability region. We can thus in principle check each point $\lambda\Delta t$ (or enlist more sophisticated solution procedures) in order to construct the full absolute stability diagram.

We shall primarily be concerned with the *use* of the stability diagram rather than the construction of the stability diagram — which for most schemes of interest are already derived and well documented. We present in Figure 21.11(b) the absolute stability diagram for the 2-step Adams-Bashforth scheme. For comparison we show in Figure 21.11(a) the absolute stability diagram for Euler Forward, which is the 1-step Adams-Bashforth scheme. Note that the stability region of the Adams-Bashforth schemes are quite small; in fact the stability region decreases further for higher order Adams-Bashforth schemes. Thus, the method is only well suited for non-stiff equations.

Example 21.1.10 Stability of the Crank-Nicolson scheme
Let us analyze the absolute stability of the Crank-Nicolson scheme. Recall that the stability of a multistep scheme is governed by the roots of the polynomial

$$q(x) = \sum_{k=0}^{K} (\alpha_k - \lambda\Delta t\, \beta_k)\, x^{K-k} \ .$$



(a) Euler Forward (AB1) (b) 2-step Adams-Bashforth (AB2)

Figure 21.11: The stability diagrams for Adams-Bashforth methods.

For the Crank-Nicolson scheme, we have $\alpha_0 = 1$, $\alpha_1 = -1$, $\beta_0 = 1/2$, and $\beta_1 = 1/2$. Thus, the polynomial is

$$q(x) = \left(1 - \frac{1}{2}(\lambda\Delta t)\right)x + \left(-1 - \frac{1}{2}(\lambda\Delta t)\right).$$

The root of the polynomial is

$$r = \frac{2 + (\lambda\Delta t)}{2 - (\lambda\Delta t)}.$$

To solve for the stability boundary, let us set $|r| = 1 = |e^{i\theta}|$ and solve for $(\lambda\Delta t)$, i.e.

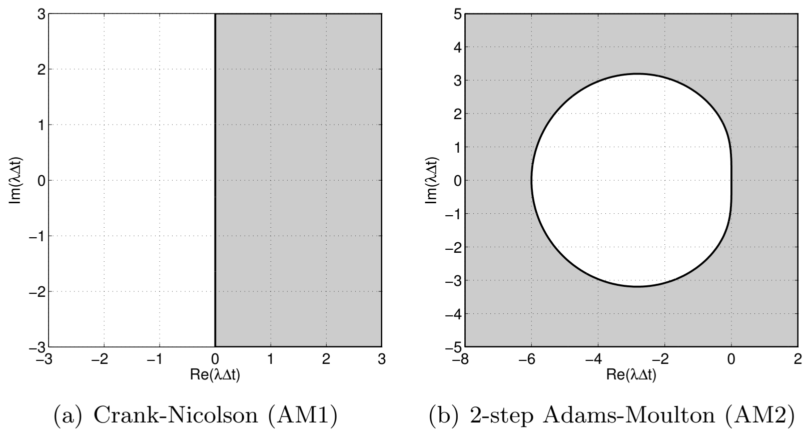
$$\frac{2 + (\lambda\Delta t)}{2 - (\lambda\Delta t)} = e^{i\theta} \quad \Rightarrow \quad (\lambda\Delta t) = \frac{2(e^{i\theta} - 1)}{e^{i\theta} + 1} = \frac{i2\sin(\theta)}{1 + \cos(\theta)}.$$

Thus, as θ varies from 0 to $\pi/2$, $\lambda\Delta t$ varies from 0 to $i\infty$ along the imaginary axis. Similarly, as θ varies from 0 to $-\pi/2$, $\lambda\Delta t$ varies from 0 to $-i\infty$ along the imaginary axis. Thus, the stability boundary is the imaginary axis. The absolute stability region is the entire left-hand (complex) plane.

The stability diagrams for the 1- and 2-step Adams-Moulton methods are shown in Figure 21.11. The Crank-Nicolson scheme shows the ideal stability diagram; it is stable for all stable ODEs ($\lambda \leq 0$) and unstable for all unstable ODEs ($\lambda > 0$) regardless of the time step selection. (Furthermore, for neutrally stable ODEs, $\lambda = 0$, Crank-Nicolson is neutrally stable — γ, the amplification factor, is unity.) The selection of time step is dictated by the accuracy requirement rather than stability concerns.[1] Despite being an implicit scheme, AM2 is not stable for all $\lambda\Delta t$ in the left-hand plane; for example, along the real axis, the time step is limited to $-\lambda\Delta t \leq 6$. While the stability region is larger than, for example, the Euler Forward scheme, the stability region of AM2 is rather disappointing considering the additional computational cost associated with each step of an implicit scheme.

·

[1]However, the Crank-Nicolson method does exhibit undesirable oscillations for $\lambda\Delta t \to -$ (real) ∞, and the lack of any dissipation on the imaginary axis can also sometimes cause difficulties. Nobody's perfect.



(a) Crank-Nicolson (AM1) (b) 2-step Adams-Moulton (AM2)

Figure 21.12: The stability diagrams for 2-step Adams-Moulton methods.

Backward Differentiation Formulas

The Backward Differentiation Formulas are implicit multistep schemes that are well suited for stiff problems. Unlike the Adams-Bashforth and Adams-Moulton schemes, we restrict ourselves to

$$\beta_k = 0, \quad k = 1, \dots, K \ .$$

Thus, the Backward Differential Formulas are of the form

$$\tilde{u}^j + \sum_{k=1}^{K} \alpha_k \tilde{u}^{j-k} = \Delta t \, \beta_0 g^j \ .$$

Our task is to find the coefficients α_k, $k = 1, \dots, K$, and β_0. We first construct a K^{th}-degree interpolating polynomial using $\tilde{u}^{j-k}$, $k = 0, \dots, K$, to approximate $u(t)$, i.e.

$$u(t) \approx \sum_{k=0}^{K} \phi_k(t) \tilde{u}^{j-k} \ ,$$

where $\phi_k(t)$, $k = 0, \dots, K$, are the Lagrange interpolation polynomials defined at the points t^{j-k}, $k = 0, \dots, K$; i.e., the same polynomials used to develop the Adams-Moulton schemes. Differentiating the function and evaluating it at $t = t^j$, we obtain

$$\left.\frac{du}{dt}\right|_{t^j} \approx \sum_{k=0}^{K} \left.\frac{d\phi_k}{dt}\right|_{t^j} \tilde{u}^{j-k} \ .$$

Again, we apply the change of variable of the form $t = t^j - \Delta t \hat{\tau}$, so that

$$\left.\frac{du}{dt}\right|_{t^j} \approx \sum_{k=0}^{K} \left.\frac{d\hat{\phi}_k}{d\hat{\tau}}\right|_{0} \left.\frac{d\hat{\tau}}{dt}\right|_{t^j} \tilde{u}^{j-k} = -\frac{1}{\Delta t} \sum_{k=0}^{K} \left.\frac{d\hat{\phi}_k}{d\hat{\tau}}\right|_{0} \tilde{u}^{j-k} \ .$$

Recalling $g^j = g(u(t^j), t^j) = du/dt|_{t^j}$, we set

$$\tilde{u}^j + \sum_{k=1}^{K} \alpha_k \tilde{u}^{j-k} \approx \Delta t \beta_0 \left(-\frac{1}{\Delta t} \sum_{k=0}^{K} \left.\frac{d\hat{\phi}_k}{d\hat{\tau}}\right|_{0} \tilde{u}^{j-k} \right) = -\beta_0 \sum_{k=0}^{K} \left.\frac{d\hat{\phi}_k}{d\hat{\tau}}\right|_{0} \tilde{u}^{j-k} \ .$$

Matching the coefficients for $\tilde{u}^{j-k}$, $k = 0, \dots, K$, we obtain

$$1 = -\beta_0 \left. \frac{d\hat{\phi}_k}{d\hat{\tau}} \right|_0$$

$$\alpha_k = -\beta_0 \left. \frac{d\hat{\phi}_k}{d\hat{\tau}} \right|_0 , \quad k = 1, \dots, K .$$

Let us develop a few Backward Differentiation Formulas.

Example 21.1.11 1-step Backward Differentiation Formula (Euler Backward)
The 1-step Backward Differentiation Formula requires specification of β_0 and α_1. As in the 1-step Adams-Moulton scheme, the Lagrange polynomials for this case are

$$\hat{\phi}_0(\hat{\tau}) = -\tau + 1 \quad \text{and} \quad \hat{\phi}_1(\hat{\tau}) = \tau .$$

Differentiating and evaluating at $\hat{\tau} = 0$

$$\beta_0 = -\left(\left. \frac{d\hat{\phi}_0}{d\hat{\tau}} \right|_0 \right)^{-1} = -(-1)^{-1} = 1 ,$$

$$\alpha_1 = -\beta_0 \left. \frac{d\hat{\phi}_1}{d\hat{\tau}} \right|_0 = -1 .$$

The resulting scheme is

$$\tilde{u}^j - \tilde{u}^{j-1} = \Delta t g^j ,$$

which is the Euler Backward scheme. Again.

Example 21.1.12 2-step Backward Differentiation Formula
The 2-step Backward Differentiation Formula requires specification of β_0, α_1, and α_2. The Lagrange polynomials for this case are

$$\hat{\phi}_0(\hat{\tau}) = \frac{1}{2}(\hat{\tau}^2 - 3\hat{\tau} + 2) ,$$

$$\hat{\phi}_1(\hat{\tau}) = -\hat{\tau}^2 + 2\hat{\tau} ,$$

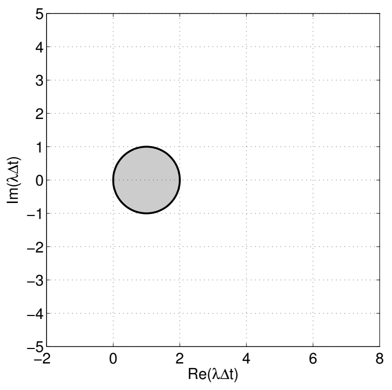
$$\hat{\phi}_2(\hat{\tau}) = \frac{1}{2}\left(\hat{\tau}^2 - \hat{\tau}\right) .$$

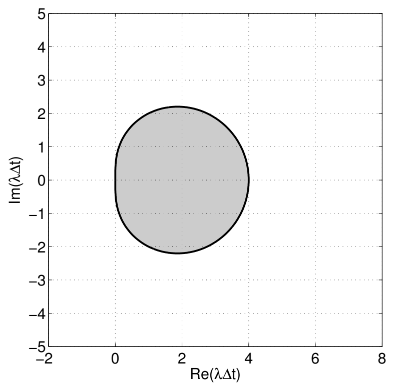
Differentiation yields

$$\beta_0 = -\left(\left. \frac{d\hat{\phi}_0}{d\hat{\tau}} \right|_0 \right)^{-1} = \frac{2}{3} ,$$

$$\alpha_1 = -\beta_0 \left. \frac{d\hat{\phi}_1}{d\hat{\tau}} \right|_0 = -\frac{2}{3} \cdot 2 = -\frac{4}{3} ,$$

$$\alpha_2 = -\beta_0 \left. \frac{d\hat{\phi}_2}{d\hat{\tau}} \right|_0 = -\frac{2}{3} \cdot -\frac{1}{2} = \frac{1}{3} .$$


(a) BDF1 (Euler Backward)


(b) BDF2

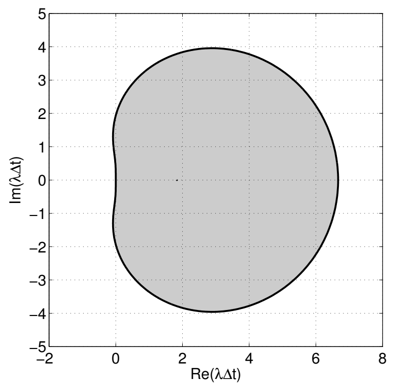

(c) BDF3

Figure 21.13: The absolute stability diagrams for Backward Differentiation Formulas.

The resulting scheme is

$$\tilde{u}^j - \frac{4}{3}\tilde{u}^{j-1} + \frac{1}{3}\tilde{u}^{j-2} = \frac{2}{3}\Delta t g^j \ .$$

The 2-step Backward Differentiation Formula (BDF2) is unconditionally stable and is second-order accurate.

·

Example 21.1.13 3-step Backward Differentiation Formula
Following the same procedure, we can develop the 3-step Backward Differentiation Formula (BDF3). The scheme is given by

$$\tilde{u}^j - \frac{18}{11}\tilde{u}^{j-1} + \frac{9}{11}\tilde{u}^{j-2} - \frac{2}{11}\tilde{u}^{j-3} = \frac{6}{11}\Delta t g^j \ .$$

The scheme is unconditionally stable and is third-order accurate.

·

The stability diagrams for the 1-, 2-, and 3-step Backward Differentiation Formulas are shown in Figure 21.13. The BDF1 and BDF2 schemes are A-stable (i.e., the stable region includes the entire left-hand plane). Unfortunately, BDF3 is not A-stable; in fact the region of instability in the left-hand plane increases for the higher-order BDFs. However, for stiff engineering systems whose eigenvalues are clustered along the real axis, the BDF methods are attractive choices.

21.1.9 Multistage Schemes: Runge-Kutta

Another family of important and powerful integration schemes are multistage schemes, the most famous of which are the Runge-Kutta schemes. While a detailed analysis of the Runge-Kutta schemes is quite involved, we briefly introduce the methods due to their prevalence in the scientific and engineering context.

Unlike multistep schemes, multistage schemes only require the solution at the previous time step $\tilde{u}^{j-1}$ to approximate the new state $\tilde{u}^j$ at time t^j. To develop an update formula, we first observe that

$$u(t^j) = \tilde{u}(t^{j-1}) + \int_{t^{j-1}}^{t^j} \frac{du}{dt}(\tau)d\tau = \tilde{u}(t^{j-1}) + \int_{t^{j-1}}^{t^j} g(u(\tau),\tau)d\tau \ .$$

Clearly, we cannot use the formula directly to approximate $u(t^j)$ because we do not know $g(u(\tau), \tau)$, $\tau \in]t^{j-1}, t^j[$. To derive the Adams schemes, we replaced the unknown function g with its polynomial approximation based on g evaluated at K previous time steps. In the case of Runge-Kutta, we directly apply numerical quadrature to the integral to obtain

$$u(t^j) \approx u(t^{j-1}) + \Delta t \sum_{k=1}^{K} b_k \, g\left(u(t^{j-1} + c_k \Delta t), t^{j-1} + c_k \Delta t\right) \ ,$$

where the b_k are the quadrature weights and the $t^j + c_k \Delta t$ are the quadrature points. We need to make further approximations to define a scheme, because we do not know the values of u at the K stages, $u(t^j + c_k \Delta t)$, $k = 1, \ldots, K$. Our approach is to replace the K stage values $u(t^{j-1} + c_k \Delta t)$ by approximations v_k and then to form the K stage derivatives as

$$G_k = g\left(v_k, t^{j-1} + c_k \Delta t\right) \ .$$

It remains to specify the approximation scheme.

For an explicit Runge-Kutta scheme, we construct the k^{th}-stage approximation as a linear combination of the previous stage derivatives and $\tilde{u}^{j-1}$, i.e.

$$v_k = \tilde{u}^{j-1} + \Delta t \left(A_{k1} G_1 + A_{k2} G_2 + \cdots + A_{k,k-1} G_{k-1}\right).$$

Because this k^{th}-stage estimate only depends on the previous stage derivatives, we can compute the stage values in sequence,

$$\begin{aligned}
v_1 &= \tilde{u}^{j-1} && (\Rightarrow G_1) \ , \\
v_2 &= \tilde{u}^{j-1} + \Delta t A_{21} G_1 && (\Rightarrow G_2) \ , \\
v_3 &= \tilde{u}^{j-1} + \Delta t A_{31} G_1 + \Delta t A_{32} G_2 && (\Rightarrow G_3) \ , \\
&\vdots \\
v_K &= \tilde{u}^{j-1} + \Delta t \textstyle\sum_{k=1}^{K-1} A_{Kk} G_k && (\Rightarrow G_K) \ .
\end{aligned}$$

Once the stage values are available, we estimate the integral by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \sum_{k=1}^{K} b_k \, G_k \ ,$$

and proceed to the next time step.

Note that a Runge-Kutta scheme is uniquely defined by the choice of the vector b for quadrature weight, the vector c for quadrature points, and the matrix A for the stage reconstruction. The coefficients are often tabulated in a Butcher table, which is a collection of the coefficients of the form

$$\begin{array}{c|c} c & A \\ \hline & b^{\mathrm{T}} \end{array} \ .$$

For explicit Runge-Kutta methods, we require $A_{ij} = 0$, $i \leq j$. Let us now introduce two popular explicit Runge-Kutta schemes.

Example 21.1.14 Two-stage Runge-Kutta
A popular two-stage Runge-Kutta method (RK2) has the Butcher table

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} .$$

This results in the following update formula

$$\begin{aligned} v_1 &= \tilde{u}^{j-1}, & G_1 &= g(v_1, t^{j-1}) , \\ v_2 &= \tilde{u}^{j-1} + \tfrac{1}{2}\Delta t G_1, & G_2 &= g\left(v_2, t^{j-1} + \frac{1}{2}\Delta t\right) , \\ \tilde{u}^j &= \tilde{u}^j + \Delta t G_2 . \end{aligned}$$

The two-stage Runge-Kutta scheme is conditionally stable and is second-order accurate. We might view this scheme as a kind of midpoint rule.

Example 21.1.15 Four-stage Runge-Kutta
A popular four-stage Runge-Kutta method (RK4) — and perhaps the most popular of all Runge-Kutta methods — has the Butcher table of the form

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} .$$

This results in the following update formula

$$\begin{aligned} v_1 &= \tilde{u}^{j-1}, & G_1 &= g(v_1, t^{j-1}) , \\ v_2 &= \tilde{u}^{j-1} + \tfrac{1}{2}\Delta t G_1, & G_2 &= g\left(v_2, t^{j-1} + \frac{1}{2}\Delta t\right), \\ v_3 &= \tilde{u}^{j-1} + \tfrac{1}{2}\Delta t G_2, & G_3 &= g\left(v_3, t^{j-1} + \frac{1}{2}\Delta t\right), \\ v_4 &= \tilde{u}^{j-1} + \Delta t G_3, & G_4 &= g\left(v_4, t^{j-1} + \Delta t\right), \\ \tilde{u}^j &= \tilde{u}^{j-1} + \Delta t\left(\frac{1}{6}G_1 + \frac{1}{3}G_2 + \frac{1}{3}G_3 + \frac{1}{6}G_4\right) . \end{aligned}$$

The four-stage Runge-Kutta scheme is conditionally stable and is fourth-order accurate.

The accuracy analysis of the Runge-Kutta schemes is quite involved and is omitted here. There are various choices of coefficients that achieve p^{th}-order accuracy using p stages for $p \leq 4$. It is also worth noting that even though we can achieve fourth-order accuracy using a four-stage Runge-Kutta method, six stages are necessary to achieve fifth-order accuracy.

Explicit Runge-Kutta methods required that a stage value is a linear combination of the previous stage derivatives. In other words, the A matrix is lower triangular with zeros on the diagonal. This made the calculation of the state values straightforward, as we could compute the stage values in sequence. If we remove this restriction, we arrive at family of *implicit Runge-Kutta methods* (IRK). The stage value updates for implicit Runge-Kutta schemes are fully coupled, i.e.

$$v_k = \tilde{u}^{j-1} + \Delta t \sum_{i=1}^{K} A_{ki} G_i, \quad k = 1, \ldots, K \ .$$

In other words, the matrix A is full in general. Like other implicit methods, implicit Runge-Kutta schemes tend to be more stable than their explicit counterparts (although also more expensive per time step). Moreover, for all K, there is a unique IRK method that achieves $2K$ order of accuracy. Let us introduce one such scheme.

Example 21.1.16 Two-stage Gauss-Legendre Implicit Runge-Kutta
The two-stage Gauss-Legendre Runge-Kutta method[2] (GL-IRK2) is described by the Butcher table

$$\begin{array}{c|cc} \frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\ \frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \ .$$

To compute the update we must first solve a system of equations to obtain the stage values v_1 and v_2

$$v_1 = \tilde{u}^{j-1} + A_{11} \Delta t G_1 + A_{12} \Delta G_2 \ ,$$

$$v_2 = \tilde{u}^{j-1} + A_{21} \Delta t G_1 + A_{12} \Delta G_2 \ ,$$

or

$$v_1 = \tilde{u}^{j-1} + A_{11} \Delta t g(v_1, t^{j-1} + c_1 \Delta t) + A_{12} \Delta t g(v_2, t^{j-1} + c_2 \Delta t) \ ,$$

$$v_2 = \tilde{u}^{j-1} + A_{21} \Delta t g(v_1, t^{j-1} + c_1 \Delta t) + A_{22} \Delta t g(v_2, t^{j-1} + c_2 \Delta t) \ ,$$

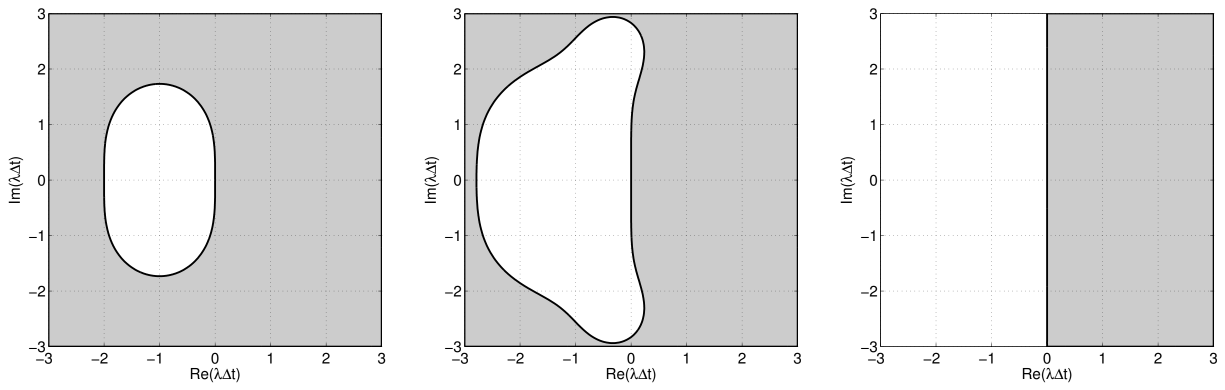
where the coefficients A and c are provided by the Butcher table. Once the stage values are computed, we compute $\tilde{u}^j$ according to

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \left(b_1 \, g(v_1, t^{j-1} + c_1 \Delta t) + b_2 \, g(v_2, t^{j-1} + c_2 \Delta t) \right) ,$$

where the coefficients b are given by the Butcher table.

The two-stage Gauss-Legendre Runge-Kutta scheme is A-stable and is fourth-order accurate. While the method is computationally expensive and difficult to implement, the A-stability and fourth-order accuracy are attractive features for certain applications.

[2]The naming is due to the use of the Gauss quadrature points, which are the roots of Legendre polynomials on the unit interval.



(a) RK2 (b) RK4 (c) Gauss-Legendre IRK2

Figure 21.14: The absolute stability diagrams for the Runge-Kutta family of schemes.

There is a family of implicit Runge-Kutta methods called *diagonally implicit Runge-Kutta* (DIRK). These methods have an A matrix that is lower triangular with the same coefficients in each diagonal element. This family of methods inherits the stability advantage of IRK schemes while being computationally more efficient than other IRK schemes for nonlinear systems, as we can incrementally update the stages.

The stability diagrams for the three Runge-Kutta schemes presented are shown in Figure 21.14. The two explicit Runge-Kutta methods, RK2 and RK4, are not A-stable. The time step along the real axis is limited to $-\lambda\Delta t \leq 2$ for RK2 and $-\lambda\Delta t \lesssim 2.8$ for RK4. However, the stability region for the explicit Runge-Kutta schemes are considerably larger than the Adams-Bashforth family of explicit schemes. While the explicit Runge-Kutta methods are not suited for very stiff systems, they can be used for moderately stiff systems. The implicit method, GL-IRK2, is A-stable; it also correctly exhibits growing behavior for unstable systems.

Figure 21.15 shows the error behavior of the Runge-Kutta schemes applied to $du/dt = -4u$. The higher accuracy of the Runge-Kutta schemes compared to the Euler Forward scheme is evident from the solution. The error convergence plot confirms the theoretical convergence rates for these methods.

21.2 Scalar Second-Order Linear ODEs

21.2.1 Model Problem

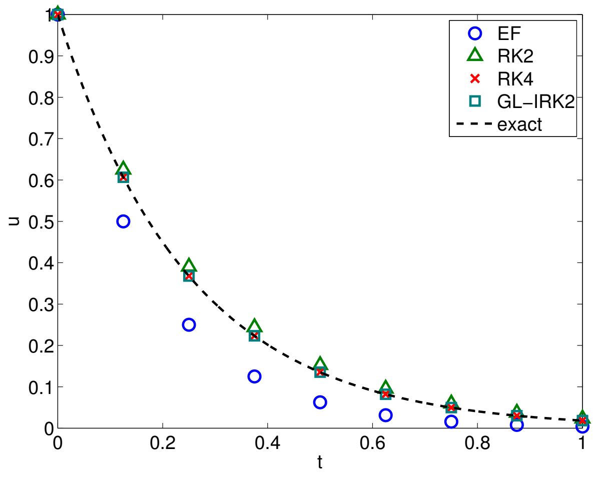
Let us consider a canonical second-order ODE,

$$m\frac{d^2u}{dt^2} + c\frac{du}{dt} + ku = f(t), \quad 0 < t < t_f \ ,$$

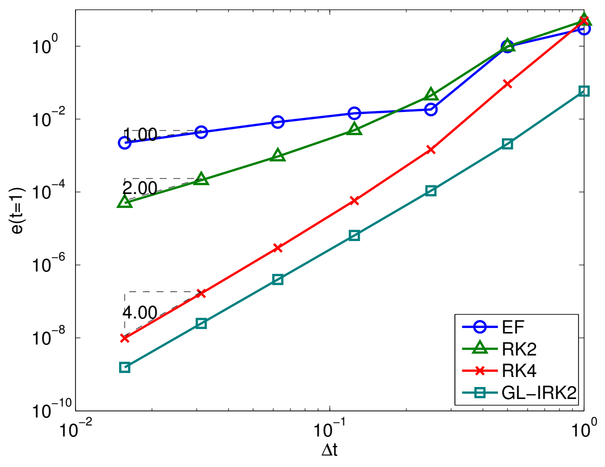
$$u(0) = u_0 \ ,$$

$$\frac{du}{dt}(0) = v_0 \ .$$

The ODE is second order, because the highest derivative that appears in the equation is the second derivative. Because the equation is second order, we now require *two* initial conditions: one for



(a) solution ($\Delta t = 1/8$)



(b) error

Figure 21.15: The error convergence behavior for the Runge-Kutta family of schemes applied to $du/dt = -4u$. Here $e(t = 1) = |u(t^j) - \tilde{u}^j|$ for $t^j = j\Delta t = 1$.

displacement, and one for velocity. It is a linear ODE because the equation is linear with respect to u and its derivatives.

A typical spring-mass-damper system is governed by this second-order ODE, where u is the displacement, m is the mass, c is the damping constant, k is the spring constant, and f is the external forcing. This system is of course a damped oscillator, as we now illustrate through the classical solutions.

21.2.2 Analytical Solution

Homogeneous Equation: Undamped

Let us consider the undamped homogeneous case, with $c = 0$ and $f = 0$,

$$m\frac{d^2u}{dt^2} + ku = 0, \quad 0 < t < t_f \ ,$$

$$u(0) = u_0 \ ,$$

$$\frac{du}{dt}(0) = v_0 \ .$$

To solve the ODE, we assume solutions of the form $e^{\lambda t}$, which yields

$$(m\lambda^2 + k)\, e^{\lambda t} = 0 \ .$$

This implies that $m\lambda^2 + k = 0$, or that λ must be a root of the characteristic polynomial

$$p(\lambda) = m\lambda^2 + k = 0 \quad \Rightarrow \quad \lambda_{1,2} = \pm i\sqrt{\frac{k}{m}} \ .$$

Let us define the *natural frequency*, $\omega_n \equiv \sqrt{k/m}$. The roots of the characteristic polynomials are then $\lambda_{1,2} = \pm i\omega_n$. The solution to the ODE is thus of the form

$$u(t) = \alpha e^{i\omega_n t} + \beta e^{-i\omega_n t} \ .$$

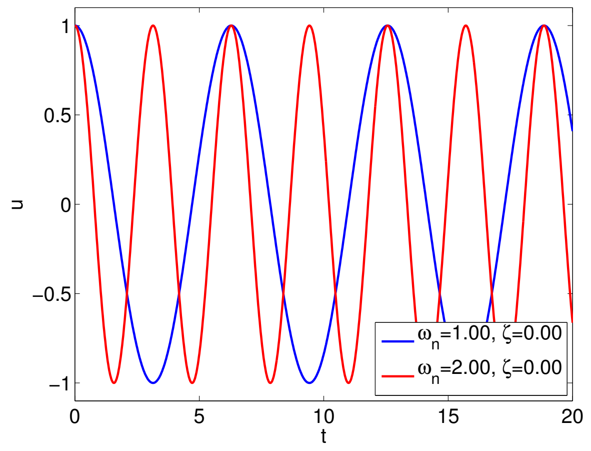


Figure 21.16: Response of undamped spring-mass systems.

Rearranging the equation,

$$u(t) = \alpha e^{i\omega_n t} + \beta e^{-i\omega_n t} = \frac{\alpha+\beta}{2}(e^{i\omega_n t} + e^{-i\omega_n t}) + \frac{\alpha-\beta}{2}(e^{i\omega_n t} - e^{-i\omega_n t})$$

$$= (\alpha+\beta)\ \cos(\omega_n t) + i(\alpha-\beta)\ \sin(\omega_n t)\ .$$

Without loss of generality, let us redefine the coefficients by $c_1 = \alpha + \beta$ and $c_2 = i(\alpha - \beta)$. The general form of the solution is thus

$$u(t) = c_1\ \cos(\omega_n t) + c_2\ \sin(\omega_n t)\ .$$

The coefficients c_1 and c_2 are specified by the initial condition. In particular,

$$u(t=0) = c_1 = u_0 \quad \Rightarrow \quad c_1 = u_0\ ,$$
$$\frac{du}{dt}(t=0) = c_2\omega_n = v_0 \quad \Rightarrow \quad c_2 = \frac{v_0}{\omega_n}\ .$$

Thus, the solution to the undamped homogeneous equation is

$$u(t) = u_0 \cos(\omega_n t) + \frac{v_0}{\omega_n} \sin(\omega_n t)\ ,$$

which represents a (non-decaying) sinusoid.

Example 21.2.1 Undamped spring-mass system

Let us consider two spring-mass systems with the natural frequencies $\omega_n = 1.0$ and 2.0. The responses of the systems to initial displacement of $u(t=0) = 1.0$ are shown in Figure 21.16. As the systems are undamped, the amplitudes of the oscillations do not decay with time.

Homogeneous Equation: Underdamped

Let us now consider the homogeneous case ($f = 0$) but with finite (but weak) damping

$$m\frac{d^2u}{dt^2} + c\frac{du}{dt} + ku = 0, \quad 0 < t < t_f \ ,$$

$$u(0) = u_0 \ ,$$

$$\frac{du}{dt}(0) = v_0 \ .$$

To solve the ODE, we again assume behavior of the form $u = e^{\lambda t}$. Now the roots of the characteristic polynomial are given by

$$p(\lambda) = m\lambda^2 + c\lambda + k = 0 \quad \Rightarrow \quad \lambda_{1,2} = -\frac{c}{2m} \pm \sqrt{\left(\frac{c}{2m}\right)^2 - \frac{k}{m}} \ .$$

Let us rewrite the roots as

$$\lambda_{1,2} = -\frac{c}{2m} \pm \sqrt{\left(\frac{c}{2m}\right)^2 - \frac{k}{m}} = -\sqrt{\frac{k}{m}}\frac{c}{2\sqrt{mk}} \pm \sqrt{\frac{k}{m}}\sqrt{\frac{c^2}{4mk} - 1} \ .$$

For convenience, let us define the *damping ratio* as

$$\zeta = \frac{c}{2\sqrt{mk}} = \frac{c}{2m\omega_n} \ .$$

Together with the definition of natural frequency, $\omega_n = \sqrt{k/m}$, we can simplify the roots to

$$\lambda_{1,2} = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1} \ .$$

The underdamped case is characterized by the condition

$$\zeta^2 - 1 < 0 \ ,$$

i.e., $\zeta < 1$.

In this case, the roots can be conveniently expressed as

$$\lambda_{1,2} = -\zeta\omega_n \pm i\omega_n\sqrt{1 - \zeta^2} = -\zeta\omega_n \pm i\omega_d \ ,$$

where $\omega_d \equiv \omega_n\sqrt{1-\zeta^2}$ is the damped frequency. The solution to the underdamped homogeneous system is

$$u(t) = \alpha e^{-\zeta\omega_n t + i\omega_d t} + \beta e^{-\zeta\omega_n t - i\omega_d t} \ .$$

Using a similar technique as that used for the undamped case, we can simplify the expression to

$$u(t) = e^{-\zeta\omega_n t}\left(c_1 \cos(\omega_d t) + c_2 \sin(\omega_d t)\right) .$$

Substitution of the initial condition yields

$$u(t) = e^{-\zeta\omega_n t}\left(u_0 \cos(\omega_d t) + \frac{v_0 + \zeta\omega_n u_0}{\omega_d} \sin(\omega_d t)\right) .$$

Thus, the solution is sinusoidal with exponentially decaying amplitude. The decay rate is set by the damping ratio, ζ. If $\zeta \ll 1$, then the oscillation decays slowly — over many periods.

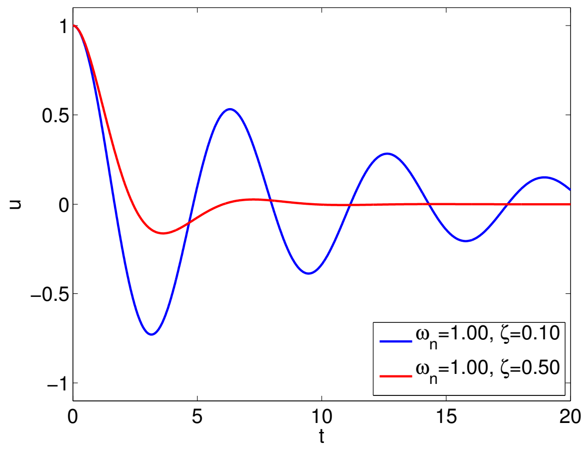


Figure 21.17: Response of underdamped spring-mass-damper systems.

Example 21.2.2 Underdamped spring-mass-damper system
Let us consider two underdamped spring-mass-damper systems with

$$\begin{aligned} &\text{System 1:} \quad \omega_n = 1.0 \quad \text{and} \quad \zeta = 0.1 \\ &\text{System 2:} \quad \omega_n = 1.0 \quad \text{and} \quad \zeta = 0.5\,. \end{aligned}$$

The responses of the systems to initial displacement of $u(t = 0) = 1.0$ are shown in Figure 21.17. Unlike the undamped systems considered in Example 21.2.1, the amplitude of the oscillations decays with time; the oscillation of System 2 with a higher damping coefficient decays quicker than that of System 1.

·

Homogeneous Equation: Overdamped

In the underdamped case, we assumed $\zeta < 1$. If $\zeta > 1$, then we have an *overdamped* system. In this case, we write the roots as

$$\lambda_{1,2} = -\omega_n\left(\zeta \pm \sqrt{\zeta^2 - 1}\right) \,,$$

both of which are real. The solution is then given by

$$u(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t} \,.$$

The substitution of the initial conditions yields

$$c_1 = \frac{\lambda_2 u_0 - v_0}{\lambda_2 - \lambda_1} \quad \text{and} \quad c_2 = \frac{-\lambda_1 u_0 + v_0}{\lambda_2 - \lambda_1} \,.$$

The solution is a linear combination of two exponentials that decay with time constants of $1/|\lambda_1|$ and $1/|\lambda_2|$, respectively. Because $|\lambda_1| > |\lambda_2|$, $|\lambda_2|$ dictates the long time decay behavior of the system. For $\zeta \to \infty$, λ_2 behaves as $-\omega_n/(2\zeta) = -k/c$.

Example 21.2.3 Overdamped spring-mass-damper system
Let us consider two overdamped spring-mass-damper systems with

$$\begin{aligned} &\text{System 1:} \quad \omega_n = 1.0 \quad \text{and} \quad \zeta = 1.0 \\ &\text{System 2:} \quad \omega_n = 1.0 \quad \text{and} \quad \zeta = 5.0\,. \end{aligned}$$

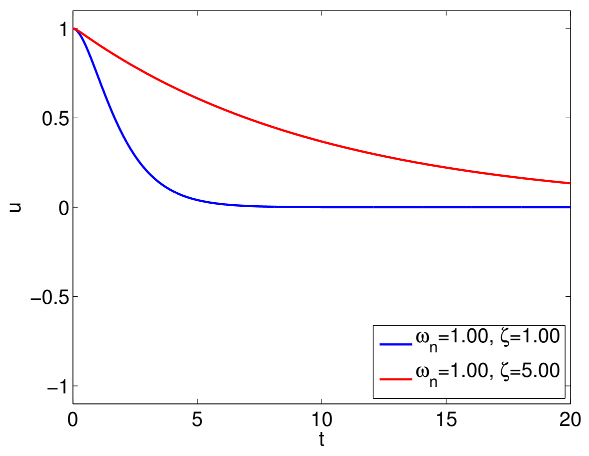


Figure 21.18: Response of overdamped spring-mass-damper systems.

The responses of the systems to initial displacement of $u(t = 0) = 1.0$ are shown in Figure 21.17. As the systems are overdamped, they exhibit non-oscillatory behaviors. Note that the oscillation of System 2 with a higher damping coefficient decays more slowly than that of System 1. This is in contrast to the underdamped cases considered in Example 21.2.2, in which the oscillation of the system with a higher damping coefficient decays more quickly.

·

Sinusoidal Forcing

Let us consider a sinusoidal forcing of the second-order system. In particular, we consider a system of the form

$$m\frac{d^2u}{dt^2} + c\frac{du}{dt} + ku = A\cos(\omega t) \; .$$

In terms of the natural frequency and the damping ratio previously defined, we can rewrite the system as

$$\frac{d^2u}{dt^2} + 2\zeta\omega_n\frac{du}{dt} + \omega_n^2 u = \frac{A}{m}\;\cos(\omega t) \; .$$

A particular solution is of the form

$$u_p(t) = \alpha\cos(\omega t) + \beta\sin(\omega t) \; .$$

Substituting the assumed form of particular solution into the governing equation, we obtain

$$\begin{aligned} 0 &= \frac{d^2u_p}{dt^2} + 2\zeta\omega_n\frac{du_p}{dt} + \omega_n^2 u_p - \frac{A}{m}\;\cos(\omega t) \\ &= -\alpha\omega^2\cos(\omega t) - \beta\omega^2\sin(\omega t) + 2\zeta\omega_n(-\alpha\omega\sin(\omega t) + \beta\omega\cos(\omega t)) \\ &\quad + \omega_n^2(\alpha\cos(\omega t) + \beta\sin(\omega t)) - A\cos(\omega t) \; . \end{aligned}$$

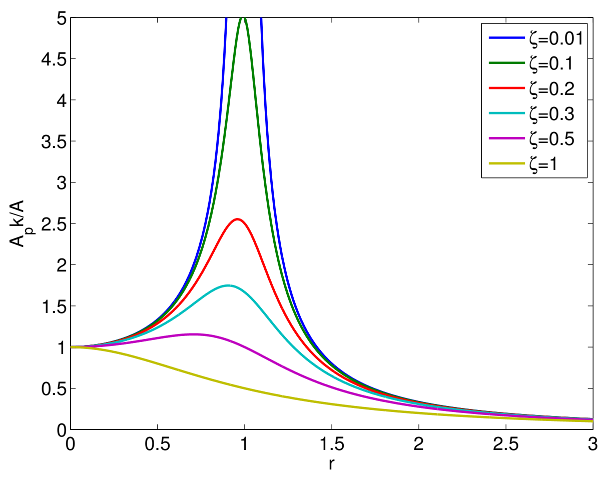


Figure 21.19: The variation in the amplification factor for the sinusoidally forced system.

We next match terms in sin and cos to obtain

$$\alpha(\omega_n^2 - \omega^2) + \beta(2\zeta\omega\omega_n) = \frac{A}{m} \ ,$$

$$\beta(\omega_n^2 - \omega^2) - \alpha(2\zeta\omega\omega_n) = 0 \ ,$$

and solve for the coefficients,

$$\alpha = \frac{(\omega_n^2 - \omega^2)}{(\omega_n^2 - \omega^2)^2 + (2\zeta\omega\omega_n)^2}\frac{A}{m} = \frac{1 - r^2}{(1-r^2)^2 + (2\zeta r)^2}\frac{A}{m\omega_n^2} = \frac{1 - r^2}{(1-r^2)^2 + (2\zeta r)^2}\frac{A}{k} \ ,$$

$$\beta = \frac{(2\zeta\omega\omega_n)}{(\omega_n^2 - \omega^2)^2 + (2\zeta\omega\omega_n)^2}\frac{A}{m} = \frac{2\zeta r}{(1-r^2)^2 + (2\zeta r)^2}\frac{A}{m\omega_n^2} = \frac{2\zeta r}{(1-r^2)^2 + (2\zeta r)^2}\frac{A}{k} \ ,$$

where $r \equiv \omega/\omega_n$ is the ratio of the forced to natural frequency.

Using a trigonometric identity, we may compute the amplitude of the particular solution as

$$A_p = \sqrt{\alpha^2 + \beta^2} = \frac{\sqrt{(1-r^2)^2 + (2\zeta r)^2}}{(1-r^2)^2 + (2\zeta r)^2}\frac{A}{k} = \frac{1}{\sqrt{(1-r^2)^2 + (2\zeta r)^2}}\frac{A}{k} \ .$$

Note that the magnitude of the amplification varies with the frequency ratio, r, and the damping ratio, ζ. This variation in the amplification factor is plotted in Figure 21.19. For a given ζ, the amplification factor is maximized at $r = 1$ (i.e., $\omega_n = \omega$), and the peak amplification factor is $1/(2\zeta)$. This increase in the magnitude of oscillation near the natural frequency of the system is known as *resonance*. The natural frequency is clearly crucial in understanding the forced response of the system, in particular for lightly damped systems.[3]

21.3 System of Two First-Order Linear ODEs

It is possible to directly numerically tackle the second-order system of Section 21.2 for example using Newmark integration schemes. However, we shall focus on a state-space approach which is much more general and in fact is the basis for numerical solution of systems of ODEs of virtually any kind.

[3] Note that for $\zeta = 0$ (which in fact is not realizable physically in any event), the amplitude is only infinite as $t \to \infty$; in particular, in resonant conditions, the amplitude will grow linearly in time.

21.3.1 State Space Representation of Scalar Second-Order ODEs

In this section, we develop a state space representation of the canonical second-order ODE. Recall that the ODE of interest is of the form

$$\frac{d^2u}{dt^2} + 2\zeta\omega_n \frac{du}{dt} + \omega_n^2 u = \frac{1}{m} f(t), \quad 0 < t < t_f \ ,$$

$$u(0) = u_0 \ ,$$

$$\frac{du}{dt}(0) = v_0 \ .$$

Because this is a second-order equation, we need two variables to fully describe the state of the system. Let us choose these state variables to be

$$w_1(t) = u(t) \quad \text{and} \quad w_2(t) = \frac{du}{dt}(t) \ ,$$

corresponding to the displacement and velocity, respectively. We have the trivial relationship between w_1 and w_2

$$\frac{dw_1}{dt} = \frac{du}{dt} = w_2 \ .$$

Furthermore, the governing second-order ODE can be rewritten in terms of w_1 and w_2 as

$$\frac{dw_2}{dt} = \frac{d}{dt}\frac{du}{dt} = \frac{d^2u}{dt^2} - 2\zeta\omega_n \frac{du}{dt} = -\omega_n^2 u + \frac{1}{m} f = -2\zeta\omega_n w_2 - \omega_n^2 w_1 + \frac{1}{m} f \ .$$

Together, we can rewrite the original second-order ODE as a system of two first-order ODEs,-

$$\frac{d}{dt}\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_2 \\ -\omega_n^2 w_1 - 2\zeta\omega_n w_2 + \frac{1}{m} f \end{pmatrix}.$$

This equation can be written in the matrix form

$$\frac{d}{dt}\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\omega_n^2 & -2\zeta\omega_n \end{pmatrix}}_{A} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} f \end{pmatrix} \tag{21.2}$$

with the initial condition

$$w_1(0) = u_0 \quad \text{and} \quad w_2(0) = v_0 \ .$$

If we define $w = (w_1 \ \ w_2)^{\mathrm{T}}$ and $F = (0 \ \ \frac{1}{m} f)^{\mathrm{T}}$, then

$$\frac{dw}{dt} = Aw + F, \quad w(t=0) = w_0 = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \ , \tag{21.3}$$

succinctly summarizes the "state-space" representation of our ODE.

Solution by Modal Expansion

To solve this equation, we first find the eigenvalues of A. We recall that the eigenvalues are the roots of the characteristic equation $p(\lambda; A) = \det(\lambda I - A)$, where det refers to the determinant. (In actual practice for large systems the eigenvalues are not computed from the characteristic equation. In our 2×2 case we obtain

$$p(\lambda; A) = \det(\lambda I - A) = \det \begin{pmatrix} \lambda & -1 \\ \omega_n^2 & \lambda + 2\zeta\omega_n \end{pmatrix} = \lambda^2 + 2\zeta\omega_n\lambda + \omega_n^2 \ .$$

The eigenvalues, the roots of characteristic equation, are thus

$$\lambda_{1,2} = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1} \ .$$

We shall henceforth assume that the system is underdamped (i.e., $\zeta < 1$), in which case it is more convenient to express the eigenvalues as

$$\lambda_{1,2} = -\zeta\omega_n \pm i\omega_n\sqrt{1 - \zeta^2} \ .$$

Note since the eigenvalue has non-zero imaginary part the solution will be oscillatory and since the real part is negative (left-hand of the complex plane) the solution is stable. We now consider the eigenvectors.

Towards that end, we first generalize our earlier discussion of vectors of real-valued components to the case of vectors of complex-valued components. To wit, if we are given two vectors $v \in \mathbb{C}^{m\times 1}$, $w \in \mathbb{C}^{m\times 1}$ — v and w are each column vectors with m complex entries — the inner product is now given by

$$\beta = v^{\mathrm{H}} w = \sum_{j=1}^{m} v_j^* \, w_j \ , \tag{21.4}$$

where β is in general complex, H stands for Hermitian (complex transpose) and replaces T for transpose, and * denotes complex conjugate — so $v_j = \mathrm{Real}(v_j) + i\,\mathrm{Imag}(v_j)$ and $v_j^* = \mathrm{Real}(v_j) - i\,\mathrm{Imag}(v_j)$, for $i = \sqrt{-1}$.

The various concepts built on the inner product change in a similar fashion. For example, two complex-valued vectors v and w are orthogonal if $v^{\mathrm{H}} w = 0$. Most importantly, the norm of complex-valued vector is now given by

$$\|v\| = \sqrt{v^{\mathrm{H}} v} = \left(\sum_{j=1}^{m} v_j^* \, v_j \right)^{1/2} = \left(\sum_{j=1}^{m} |v_j|^2 \right)^{1/2} \ , \tag{21.5}$$

where $|\cdot|$ denotes the complex modulus; $|v_j|^2 = v_j^* \, v_j = (\mathrm{Real}(v_j))^2 + (\mathrm{Imag}(v_j))^2$. Note the definition (21.5) of the norm ensures that $\|v\|$ is a non-negative real number, as we would expect of a length.

To obtain the eigenvectors, we must find a solution to the equation

$$(\lambda I - A)\chi = 0 \tag{21.6}$$

for $\lambda = \lambda_1$ ($\Rightarrow$ eigenvector $\chi^1 \in \mathbb{C}^2$) and $\lambda = \lambda_2$ ($\Rightarrow$ eigenvector $\chi^2 \in \mathbb{C}^2$). The equations (21.6) will have a solution since λ has been chosen to make $(\lambda I - A)$ singular: the columns of $\lambda I - A$ are *not* linearly independent, and hence there exists a (in fact, many) nontrivial linear combination, $\chi \neq 0$, of the columns of $\lambda I - A$ which yields the zero vector.

Proceeding with the first eigenvector, we write $(\lambda_1 I - A)\chi^1 = 0$ as

$$\begin{pmatrix} -\zeta\omega_n + i\omega_n\sqrt{1-\zeta^2} & -1 \\ \omega_n^2 & \zeta\omega_n + i\omega_n\sqrt{1-\zeta^2} \end{pmatrix} \begin{pmatrix} \chi_1^1 \\ \chi_2^1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

to obtain (say, setting $\chi_1^1 = c$),

$$\chi^1 = c \begin{pmatrix} 1 \\ \dfrac{-\omega_n^2}{\zeta\omega_n + i\omega_n\sqrt{1-\zeta^2}} \end{pmatrix} .$$

We now choose c to achieve $\|\chi^1\| = 1$, yielding

$$\chi^1 = \frac{1}{\sqrt{1+\omega_n^2}} \begin{pmatrix} 1 \\ -\zeta\omega_n + i\omega_n\sqrt{1-\zeta^2} \end{pmatrix} .$$

In a similar fashion we obtain from $(\lambda_2 I - A)\chi^2 = 0$ the second eigenvector

$$\chi^2 = \frac{1}{\sqrt{1+\omega_n^2}} \begin{pmatrix} 1 \\ -\zeta\omega_n - i\omega_n\sqrt{1-\zeta^2} \end{pmatrix} ,$$

which satisfies $\|\chi^2\| = 1$.

We now introduce two additional vectors, ψ^1 and ψ^2. The vector ψ^1 is chosen to satisfy $(\psi^1)^{\mathrm{H}}\chi^2 = 0$ and $(\psi^1)^{\mathrm{H}}\chi^1 = 1$, while the vector ψ^2 is chosen to satisfy $(\psi^2)^{\mathrm{H}}\chi^1 = 0$ and $(\psi^2)^{\mathrm{H}}\chi^2 = 1$. We find, after a little algebra,

$$\psi^1 = \frac{\sqrt{1+\omega_n^2}}{2i\omega_n\sqrt{1-\zeta^2}} \begin{pmatrix} -\zeta\omega_n + i\omega_n\sqrt{1-\zeta^2} \\ -1 \end{pmatrix} , \quad \psi^2 = \frac{\sqrt{1+\omega_n^2}}{-2i\omega_n\sqrt{1-\zeta^2}} \begin{pmatrix} -\zeta\omega_n - i\omega_n\sqrt{1-\zeta^2} \\ -1 \end{pmatrix} .$$

These choices may appear mysterious, but in a moment we will see the utility of this "bi-orthogonal" system of vectors. (The steps here in fact correspond to the "diagonalization" of A.)

We now write w as a linear combination of the two eigenvectors, or "modes,"

$$\begin{aligned} w(t) &= z_1(t)\,\chi^1 + z_2(t)\,\chi^2 \\ &= S\,z(t) \end{aligned} \tag{21.7}$$

where

$$S = (\chi^1 \;\; \chi^2)$$

is the 2×2 matrix whose jth-column is given by the jth-eigenvector, χ^j. We next insert (21.7) into (21.3) to obtain

$$\chi^1 \frac{dz_1}{dt} + \chi^2 \frac{dz_2}{dt} = A(\chi^1 z_1 + \chi^2 z_2) + F , \tag{21.8}$$

$$(\chi^1 z_1 + \chi^2 z_2)(t = 0) = w_0 . \tag{21.9}$$

We now take advantage of the ψ vectors.

First we multiply (21.8) by $(\psi^1)^{\mathrm{H}}$ and take advantage of $(\psi^1)^{\mathrm{H}}\,\chi^2 = 0$, $(\psi^1)^{\mathrm{H}}\,\chi^1 = 1$, and $A\chi^j = \lambda_j \chi^j$ to obtain

$$\frac{dz_1}{dt} = \lambda_1\, z_1 + (\psi^1)^{\mathrm{H}}\, F \ ; \tag{21.10}$$

if we similarly multiply (21.9) we obtain

$$z_1(t=0) = (\psi^1)^{\mathrm{H}}\, w_0 \ . \tag{21.11}$$

The same procedure but now with $(\psi^2)^{\mathrm{H}}$ rather than $(\psi^1)^{\mathrm{H}}$ gives

$$\frac{dz_2}{dt} \ = \ \lambda_2\, z_2 + (\psi^2)^{\mathrm{H}}\, F \ ; \tag{21.12}$$

$$z_2(t=0) \ = \ (\psi^2)^{\mathrm{H}}\, w_0 \ . \tag{21.13}$$

We thus observe that our modal expansion reduces our coupled 2×2 ODE system into two decoupled ODEs.

The fact that λ_1 and λ_2 are complex means that z_1 and z_2 are also complex, which might appear inconsistent with our original *real* equation (21.3) and *real* solution $w(t)$. However, we note that $\lambda_2 = \lambda_1^*$ and $\psi^2 = (\psi^1)^*$ and thus $z_2 = z_1^*$. It thus follows from (21.7) that, since $\chi^2 = (\chi^1)^*$ as well,

$$w = z_1\chi^1 + z_1^*(\chi^1)^* \ ,$$

and thus

$$w = 2\,\mathrm{Real}(z_1\chi^1) \ .$$

Upon superposition, our solution is indeed real, as desired.

It is possible to use this modal decomposition to construct numerical procedures. However, our interest here in the modal decomposition is as a way to understand how to choose an ODE scheme for a system of two (later n) ODEs, and, for the chosen scheme, how to choose Δt for stability.

21.3.2 Numerical Approximation of a System of Two ODEs

Crank-Nicolson

The application of the Crank-Nicolson scheme to our system (21.3) is identical to the application of the Crank-Nicolson scheme to a scalar ODE. In particular, we directly take the scheme of example 21.1.8 and replace $\tilde{u}^j \in \mathbb{R}$ with $\tilde{w}^j \in \mathbb{R}^2$ and g with $A\tilde{w}^j + F^j$ to obtain

$$\tilde{w}^j = \tilde{w}^{j-1} + \frac{\Delta t}{2}\,(A\tilde{w}^j + A\tilde{w}^{j-1}) + \frac{\Delta t}{2}\,(F^j + F^{j-1}) \ . \tag{21.14}$$

(Note if our force f is constant in time then $F^j = F$.) In general if follows from consistency arguments that we will obtain the same order of convergence as for the scalar problem — *if* (21.14) is stable. The difficult issue for systems is *stability*: Will a particular scheme have good stability properties for a particular equation (e.g., our particular A of (21.2))? And for what Δt will the scheme be stable? (The latter is particularly important for explicit schemes.)

To address these questions we again apply modal analysis but now to our discrete equations (21.14). In particular, we write

$$\tilde{w}^j = \tilde{z}_1^j\chi^1 + \tilde{z}_2^j\chi^2 \ , \tag{21.15}$$

where χ^1 and χ^2 are the eigenvectors of A as derived in the previous section. We now insert (21.15) into (21.14) and multiply by $(\psi^1)^{\mathrm{H}}$ and $(\psi^2)^{\mathrm{H}}$ — just as in the previous section — to obtain

$$\tilde{z}_1^j = \tilde{z}_1^{j-1} + \frac{\lambda_1 \Delta t}{2}\left(\tilde{z}_1^j + \tilde{z}_1^{j-1}\right) + (\psi^1)^{\mathrm{H}} \frac{\Delta t}{2}\left(F^j + F^{j-1}\right) , \tag{21.16}$$

$$\tilde{z}_2^j = \tilde{z}_2^{j-1} + \frac{\lambda_2 \Delta t}{2}\left(\tilde{z}_2^j + \tilde{z}_2^{j-1}\right) + (\psi^2)^{\mathrm{H}} \frac{\Delta t}{2}\left(F^j + F^{j-1}\right) , \tag{21.17}$$

with corresponding initial conditions (which are not relevant to our current discussion).

We now recall that for the model problem

$$\frac{du}{dt} = \lambda u + f , \tag{21.18}$$

analogous to (21.10), we arrive at the Crank-Nicolson scheme

$$\tilde{u}^j = \tilde{u}^{j-1} + \frac{\lambda \Delta t}{2}(\tilde{u}^j + \tilde{u}^{j-1}) + \frac{\Delta t}{2}(f^j + f^{j-1}) , \tag{21.19}$$

analogous to (21.16). Working backwards, for (21.19) and hence (21.16) to be a stable approximation to (21.18) and hence (21.10), we must require $\lambda\Delta t$, and hence $\lambda_1\Delta t$, to reside in the Crank-Nicolson absolute stability region depicted in Figure 21.12(a). Put more bluntly, we know that the difference equation (21.16) will blow up — and hence also (21.14) by virtue of (21.15) — if $\lambda_1\Delta t$ is not in the unshaded region of Figure 21.12(a). By similar arguments, $\lambda_2\Delta t$ must also lie in the unshaded region of Figure 21.12(a). In this case, we know that both λ_1 and λ_2 — for our *particular* equation, that is, for our *particular* matrix A (which determines the eigenvalues λ_1, λ_2) — are in the left-hand plane, and hence in the Crank-Nicolson absolute stability region; thus Crank-Nicolson is unconditionally stable — stable for all Δt — for our particular equation and will converge as $O(\Delta t^2)$ as $\Delta t \to 0$.

We emphasize that the numerical procedure is given by (21.14) , and *not* by (21.16), (21.17). The modal decomposition is just for the purposes of understanding and analysis — to determine if a scheme is stable and if so for what values of Δt. (For a 2×2 matrix A the full modal decomposition is simple. But for larger systems, as we will consider in the next section, the full modal decomposition is very expensive. Hence we prefer to directly discretize the original equation, as in (21.14). This direct approach is also more general, for example for treatment of nonlinear problems.) It follows that Δt in (21.16) and (21.17) are the *same* — both originate in the equation (21.14). We discuss this further below in the context of stiff equations.

General Recipe

We now consider a general system of $n = 2$ ODEs given by

$$\begin{aligned} \frac{dw}{dt} &= Aw + F , \\ w(0) &= w_0 , \end{aligned} \tag{21.20}$$

where $w \in \mathbb{R}^2$, $A \in \mathbb{R}^{2\times 2}$ (a 2×2 matrix), $F \in \mathbb{R}^2$, and $w_0 \in \mathbb{R}^2$. We next discretize (21.20) by any of the schemes developed earlier for the scalar equation

$$\frac{du}{dt} = g(u, t)$$

simply by substituting w for u and $Aw + F$ for $g(u,t)$. We shall denote the scheme by $\mathbb{S}$ and the associated absolute stability region by $\mathcal{R}_{\mathbb{S}}$. Recall that $\mathcal{R}_{\mathbb{S}}$ is the subset of the complex plane which contains all $\lambda\Delta t$ for which the scheme $\mathbb{S}$ applied to $g(u,t) = \lambda u$ is absolutely stable.

For example, if we apply the Euler Forward scheme $\mathbb{S}$ we obtain

$$\tilde{w}^j = \tilde{w}^{j-1} + \Delta t(A\tilde{w}^{j-1} + F^{j-1}) \ , \tag{21.21}$$

whereas Euler Backward as $\mathbb{S}$ yields

$$\tilde{w}^j = \tilde{w}^{j-1} + \Delta t(A\tilde{w}^{j} + F^{j}) \ , \tag{21.22}$$

and Crank-Nicolson as $\mathbb{S}$ gives

$$\tilde{w}^j = \tilde{w}^{j-1} + \frac{\Delta t}{2}(A\tilde{w}^{j} + A\tilde{w}^{j-1}) + \frac{\Delta t}{2}(F^{j} + F^{j-1}) \ . \tag{21.23}$$

A multistep scheme such as AB2 as $\mathbb{S}$ gives

$$\tilde{w}^j = \tilde{w}^{j-1} + \Delta t\left(\frac{3}{2}\,A\tilde{w}^{j-1} - \frac{1}{2}\,A\tilde{w}^{j-2}\right) + \Delta t\left(\frac{3}{2}\,F^{j-1} - \frac{1}{2}\,F^{j-2}\right) \ . \tag{21.24}$$

The stability diagrams for these four schemes, $\mathcal{R}_{\mathbb{S}}$, are given by Figure 21.9, Figure 21.7, Figure 21.12(a), and Figure 21.11(b), respectively.

We next assume that we can calculate the two eigenvalues of A, λ_1, and λ_2. A particular Δt will lead to a *stable* scheme if and only if the two points $\lambda_1\Delta t$ and $\lambda_2\Delta t$ *both* lie inside $\mathcal{R}_{\mathbb{S}}$. If either or both of the two points $\lambda_1\Delta t$ or $\lambda_2\Delta t$ lie outside $\mathcal{R}_{\mathbb{S}}$, then we must decrease Δt until both $\lambda_1\Delta t$ and $\lambda_2\Delta t$ lie inside $\mathcal{R}_{\mathbb{S}}$. The critical time step, Δt_{cr}, is defined to be the *largest* Δt for which the two *rays* $[0, \lambda_1\Delta t]$, $[0, \lambda_2\Delta t]$, *both* lie within $\mathcal{R}_{\mathbb{S}}$; Δt_{cr} will depend on the shape and size of $\mathcal{R}_{\mathbb{S}}$ and the "orientation" of the two rays $[0, \lambda_1\Delta t]$, $[0, \lambda_2\Delta t]$.

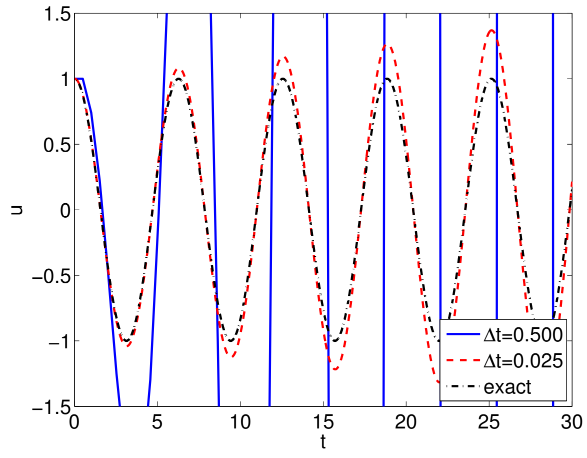
We can derive Δt_{cr} in a slightly different fashion. We first define $\widehat{\Delta t}_1$ to be the largest Δt such that the ray $[0, \lambda_1\Delta t]$ is in $\mathcal{R}_{\mathbb{S}}$; we next define $\widehat{\Delta t}_2$ to be the largest Δt such that the ray $[0, \lambda_2\Delta t]$ is in $\mathcal{R}_{\mathbb{S}}$. We can then deduce that $\Delta t_{\text{cr}} = \min(\widehat{\Delta t}_1, \widehat{\Delta t}_2)$. *In particular, we note that if $\Delta t > \Delta t_{\text{cr}}$ then one of the two modes — and hence the entire solution — will explode.* We can also see here again the difficulty with stiff equations in which λ_1 and λ_2 are very different: $\widehat{\Delta t}_1$ may be (say) much larger than $\widehat{\Delta t}_2$, but $\widehat{\Delta t}_2$ will dictate Δt and thus force us to take many time steps — many more than required to resolve the slower mode (smaller $|\lambda_1|$ associated with slower decay or slower oscillation) which is often the behavior of interest.

In the above we assumed, as is almost always the case, that the λ are in the left-hand plane. For any λ which are in the right-hand plane, our condition is flipped: we now must make sure that the $\lambda\Delta t$ are *not* in the absolute stability region in order to obtain the desired growing (unstable) solutions.

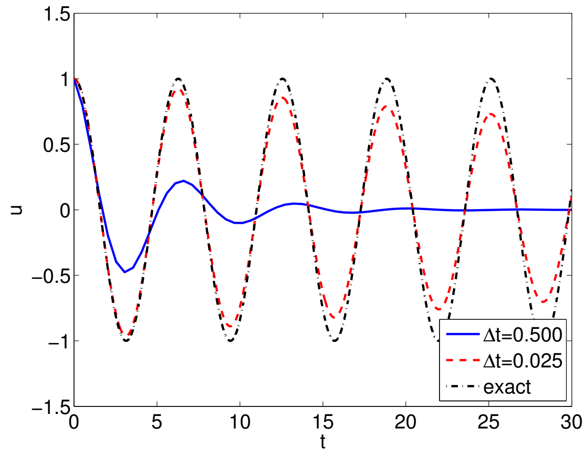
Let us close this section with two examples.

Example 21.3.1 Undamped spring-mass system
In this example, we revisit the undamped spring-mass system considered in the previous section. The two eigenvalues of A are $\lambda_1 = i\omega_n$ and $\lambda_2 = i\omega_n$; without loss of generality, we set $\omega_n = 1.0$. We will consider application of several different numerical integration schemes to the problem; for each integrator, we assess its applicability based on theory (by appealing to the absolute stability diagram) and verify our assessment through numerical experiments.

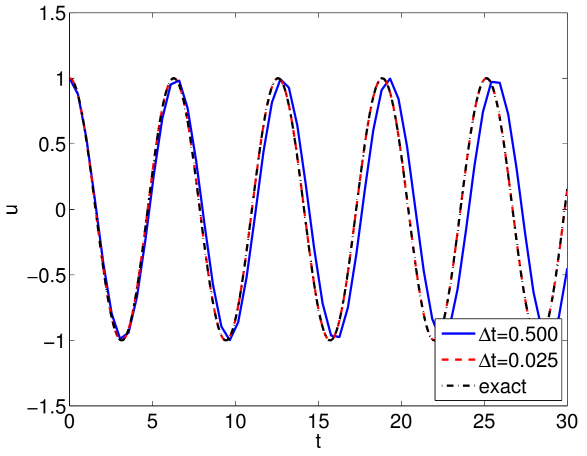
(i) Euler Forward is a poor choice since both $\lambda_1\Delta t$ and $\lambda_2\Delta t$ are outside $\mathcal{R}_{\mathbb{S}=\text{EF}}$ for all Δt. The result of numerical experiment, shown in Figure 21.20(a), confirms that the amplitude of the oscillation grows for both $\Delta t = 0.5$ and $\Delta t = 0.025$; the smaller time step results in a smaller (artificial) amplification.



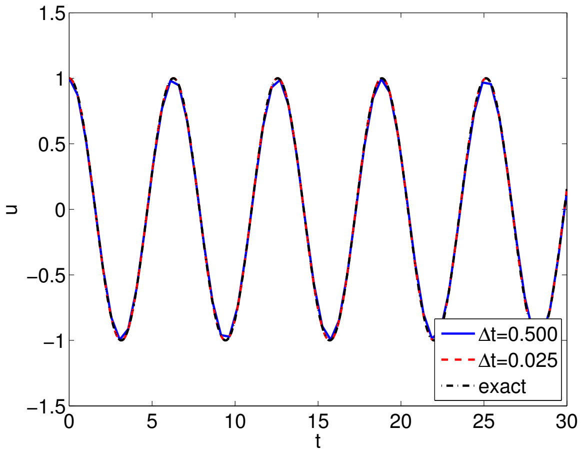
(a) Euler Forward



(b) Euler Backward



(c) Crank-Nicolson



(d) Four-stage Runge-Kutta

Figure 21.20: Comparison of numerical integration schemes for an undamped spring-mass system with $\omega_n = 1.0$.

(*ii*) Euler Backward is also a poor choice since $\lambda_1 \Delta t$ and $\lambda_2 \Delta t$ are in the *interior* of $\mathcal{R}_{\mathbb{S}=\mathrm{EB}}$ for all Δt and hence the discrete solution will decay even though the exact solution is a *non-decaying* oscillation. Figure 21.20(b) confirms the assessment.

(*iii*) Crank-Nicolson is a very good choice since $\lambda_1 \Delta t \in \mathcal{R}_{\mathbb{S}=\mathrm{CN}}$, $\lambda_2 \Delta t \in \mathcal{R}_{\mathbb{S}=\mathrm{CN}}$ for all Δt, and furthermore $\lambda_1 \Delta t$, $\lambda_2 \Delta t$ lie on the *boundary* of $\mathcal{R}_{\mathbb{S}=\mathrm{CN}}$ and hence the discrete solution, just as the exact solution, will not decay. Figure 21.20(c) confirms that Crank-Nicolson preserves the amplitude of the response regardless of the choice of Δt; however, the $\Delta t = 0.5$ case results in a noticeable phase error.

(*iv*) Four-stage Runge-Kutta (RK4) is a reasonably good choice since $\lambda_1 \Delta t$ and $\lambda_2 \Delta t$ lie close to the boundary of $\mathcal{R}_{\mathbb{S}=\mathrm{RK4}}$ for $|\lambda_i \Delta t| \lesssim 1$. Figure 21.20(d) shows that, for the problem considered, RK4 excels at not only preserving the amplitude of the oscillation but also at attaining the correct phase.

Note in the above analysis the absolute stability diagram serves not only to determine stability but also the nature of the discrete solution as regards growth, *or* decay, *or* even neutral stability — no growth or decay. (The latter does not imply that the discrete solution is exact, since in addition to amplitude errors there are also phase errors. Our Crank-Nicolson result, shown in Figure 21.20(c), in particular demonstrate the presence of phase errors in the absence of amplitude errors.)

Example 21.3.2 Overdamped spring-mass-damper system: a stiff system of ODEs
In our second example, we consider a (very) overdamped spring-mass-damper system with $\omega_n = 1.0$ and $\zeta = 100$. The eigenvalues associated with the system are

$$\lambda_1 = -\zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1} = -0.01$$
$$\lambda_2 = -\zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1} = -99.99\,.$$

As before, we perturb the system by a unit initial displacement. The slow mode with $\lambda_1 = -0.01$ dictates the response of the system. However, for conditionally stable schemes, the stability is governed by the fast mode with $\lambda_2 = -99.99$. We again consider four different time integrators: two explicit and two implicit.

(*i*) Euler Forward is stable for $\Delta t \lesssim 0.02$ (i.e. $\Delta t_{\mathrm{cr}} = 2/|\lambda_2|$). Figure 21.21(a) shows that the scheme accurately tracks the (rather benign) exact solution for $\Delta t = 0.02$, but becomes unstable and diverges exponentially for $\Delta t = 0.0201$. Thus, the maximum time step is limited not by the ability to approximate the system response (dictated by λ_1) but rather by stability (dictated by λ_2). In other words, even though the system response is benign, we cannot use large time steps to save on computational cost.

(*ii*) Similar to the Euler Forward case, the four-stage Runge-Kutta (RK4) scheme exhibits an exponentially diverging behavior for $\Delta t > \Delta t_{\mathrm{cr}} \approx 0.028$, as shown in Figure 21.21(b). The maximum time step is again limited by stability.

(*iii*) Euler Backward is unconditionally stable, and thus the choice of the time step is dictated by the ability to approximate the system response, which is dictated by λ_1. Figure 21.21(c) shows that Euler Backward in fact produces a good approximation even for a time step as large as $\Delta t = 5.0$ since the system response is rather slow.

(*iv*) Crank-Nicolson is also unconditionally stable. For the same set of time steps, Crank-Nicolson produces a more accurate approximation than Euler Backward, as shown in Figure 21.21(d), due to its higher-order accuracy.

In the above comparison, the unconditionally stable schemes required many fewer time steps (and hence much less computational effort) than conditionally stable schemes. For instance, Crank-Nicolson with $\Delta t = 5.0$ requires approximately 200 times fewer time steps than the RK4 scheme (with a stable choice of the time step). More importantly, as the shortest time scale (i.e. the largest eigenvalue) dictates stability, conditionally stable schemes do not allow the user to use large time steps *even if the fast modes are of no interest to the user*. As mentioned previously, stiff systems are ubiquitous in engineering, and engineers are often not interested in the smallest time scale present in the system. (Recall the example of the time scale associated with the dynamics of a passenger jet and that associated with turbulent eddies; engineers are often only interested in characterizing the dynamics of the aircraft, not the eddies.) In these situations, unconditionally stable schemes allow users to choose an appropriate time step independent of stability limitations.

·

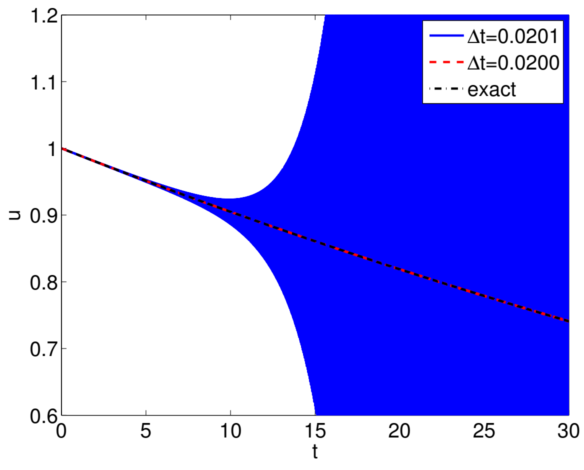
In closing, it is clear even from these simple examples that a general purpose *explicit* scheme would ideally include some part of both the negative real axis *and* the imaginary axis. Schemes that exhibit this behavior include AB3 and RK4. Of these two schemes, RK4 is often preferred due to a large stability region; also RK4, a multi-*stage* method, does not suffer from the start-up issues that sometimes complicate multi-step techniques.

21.4 IVPs: System of n Linear ODEs

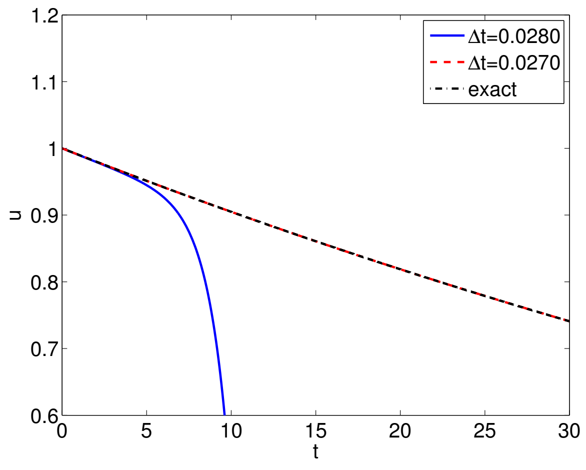
We consider here for simplicity a particular family of problems: $n/2$ coupled oscillators. This family of systems can be described by the set of equations.

$$\begin{aligned}
\frac{d^2u^{(1)}}{dt^2} &= h^{(1)}\left(\frac{du^{(j)}}{dt}, u^{(j)},\ 1 \le j \le n/2\right) + f^{(1)}(t)\ ,\\
\frac{d^2u^{(2)}}{dt^2} &= h^{(2)}\left(\frac{du^{(j)}}{dt}, u^{(j)},\ 1 \le j \le n/2\right) + f^{(2)}(t)\ ,\\
&\ \ \vdots\\
\frac{d^2u^{(n/2)}}{dt^2} &= h^{(n/2)}\left(\frac{du^{(j)}}{dt}, u^{(j)},\ 1 \le j \le n/2\right) + f^{(n/2)}(t)\ ,
\end{aligned}$$

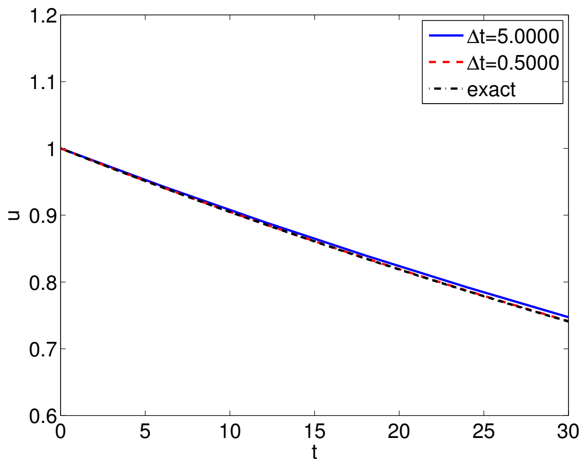
where $h^{(k)}$ is assumed to be a *linear* function of all its arguments.



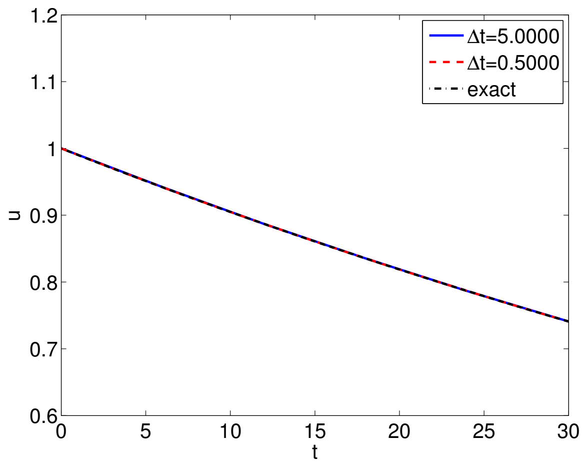
(a) Euler Forward



(b) Four-stage Runge-Kutta



(c) Euler Backward



(d) Crank-Nicolson

Figure 21.21: Comparison of numerical integration schemes for an overdamped spring-mass-damper system with $\omega_n = 1.0$ and $\zeta = 50$. Note that the time step used for the explicit schemes are different from those for the implicit schemes.

We first convert this system of equations to state space form. We identify

$$\begin{aligned} w_1 &= u^{(1)}, & w_2 &= \frac{du^{(1)}}{dt}\,, \\ w_3 &= u^{(2)}, & w_4 &= \frac{du^{(2)}}{dt}\,, \\ &\vdots \\ w_{n-1} &= u^{(n/2)}, & w_n &= \frac{du^{(n/2)}}{dt}\,. \end{aligned}$$

We can then write our system — using the fact that h is linear in its arguments — as

$$\begin{aligned} \frac{dw}{dt} &= Aw + F \\ w(0) &= w_0 \end{aligned} \tag{21.25}$$

where h determines A, F is given by $\begin{pmatrix} 0 & f^{(1)}(t) & 0 & f^{(2)}(t) & \dots & 0 & f^{(n/2)}(t) \end{pmatrix}^{\mathrm{T}}$, and

$$w_0 = \begin{pmatrix} u^{(1)}(0) & \frac{du^{(1)}}{dt}(0) & u^{(2)}(0) & \frac{du^{(2)}}{dt}(0) & \dots & u^{(n/2)}(0) & \frac{du^{(n/2)}}{dt}(0) \end{pmatrix}^{\mathrm{T}}.$$

We have now reduced our problem to an abstract form identical to (21.20) and hence we may apply any scheme $\mathbb{S}$ to (21.25) in the same fashion as to (21.20).

For example, Euler Forward, Euler Backward, Crank-Nicolson, and AB2 applied to (21.25) will take the same form (21.21), (21.22), (21.23), (21.24), respectively, except that now $w \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $F \in \mathbb{R}^n$, $w_0 \in \mathbb{R}^n$ are given in (21.25), where $n/2$, the number of oscillators (or masses) in our system, is no longer restricted to $n/2 = 1$ (i.e., $n = 2$). We can similarly apply AB3 or BD2 or RK4.

Our stability criterion is also readily extended. We first note that A will now have in general n eigenvalues, $\lambda_1, \lambda_2, \dots, \lambda_n$. (In certain cases multiple eigenvalues can create difficulties; we do not consider these typically rather rare cases here.) Our stability condition is then simply stated: a time step Δt will lead to stable behavior if and only if $\lambda_i \Delta t$ is in $\mathcal{R}_{\mathbb{S}}$ for all i, $1 \le i \le n$. If this condition is not satisfied then there will be one (or more) modes which will explode, taking with it (or them) the entire solution. (For certain very special initial conditions — in which the w_0 is chosen such that all of the dangerous modes are initially *exactly* zero — this blow-up could be avoided in *infinite* precision; but in finite precision we would still be doomed.) For explicit schemes, Δt_{cr} is the *largest* time step such that *all* the rays $[0, \lambda_i \Delta t]$, $1 \le i \le n$, lie within $\mathcal{R}_{\mathbb{S}}$.

There are certainly computational difficulties that arise for large n that are not an issue for $n = 2$ (or small n). First, for implicit schemes, the necessary division — *solution* rather than evaluation of matrix-vector equations — will become considerably more expensive. Second, for explicit schemes, determination of Δt_{cr}, or a bound $\Delta t_{\mathrm{cr}}^{\mathrm{conservative}}$ such that $\Delta t_{\mathrm{cr}}^{\mathrm{conservative}} \approx \Delta t_{\mathrm{cr}}$ and $\Delta t_{\mathrm{cr}}^{\mathrm{conservative}} \le \Delta t_{\mathrm{cr}}$, can be difficult. As already mentioned, the full modal decomposition can be expensive. Fortunately, in order to determine Δt_{cr}, we often only need as estimate for say the most negative real eigenvalue, or the largest (in magnitude) imaginary eigenvalue; these extreme eigenvalues can often be estimated relatively efficiently.

Finally, we note that in practice often adaptive schemes are used in which stability and accuracy are monitored and Δt modified appropriately. These methods can also address nonlinear problems — in which h no longer depends linearly on its arguments.

Chapter 22

Boundary Value Problems

Chapter 23

Partial Differential Equations

Unit V

(Numerical) Linear Algebra 2: Solution of Linear Systems

Chapter 24

Motivation

We thank Dr Phuong Huynh of MIT for generously developing and sharing the robot arm model, finite element discretization, and computational results reported in this chapter.

24.1 A Robot Arm

In the earlier units we have frequently taken inspiration from applications related to robots — navigation, kinematics, and dynamics. In these earlier analyses we considered systems consisting of relatively few "lumped" components and hence the computational tasks were rather modest. However, it is also often important to address not just lumped components but also the detailed deformations and stresses within say a robot arm: excessive deformation can compromise performance in precision applications; and excessive stresses can lead to failure in large manufacturing tasks.

The standard approach for the analysis of deformations and stresses is the finite element (FE) method. In the FE approach, the spatial domain is first broken into many (many) small regions denoted elements: this decomposition is often denoted a triangulation (or more generally a grid or mesh), though elements may be triangles, quadrilaterals, tetrahedra, or hexahedra; the vertices of these elements define nodes (and we may introduce additional nodes at say edge or face midpoints). The displacement field within each such element is then expressed in terms of a low order polynomial representation which interpolates the displacements at the corresponding nodes. Finally, the partial differential equations of linear elasticity are invoked, in variational form, to yield equilibrium equations at (roughly speaking) each node in terms of the displacements at the neighboring nodes. Very crudely, the coefficients in these equations represent effective spring constants which reflect the relative nodal geometric configuration and the material properties. We may express this system of n equations — one equation for each node — in n unknowns — one displacement (or "degree of freedom") for each node — as $Ku = f$, in which K is an $n \times n$ matrix, u is an $n \times 1$ vector of the unknown displacements, and f is an $n \times 1$ vector of imposed forces or "loads."[1]

We show in Figure 24.1 the finite element solution for a robot arm subject only to the "self-load" induced by gravity. The blue arm is the unloaded (undeformed) arm, whereas the multi-color arm is the loaded (deformed) arm; note in the latter we greatly amplify the actual displacements for purposes of visualization. The underlying triangulation — in particular surface triangles associated

[1] In fact, for this vector-valued displacement field, there are three equations and three degrees of freedom for each (geometric) node. For simplicity we speak of (generalized) nodes equated to degrees of freedom.

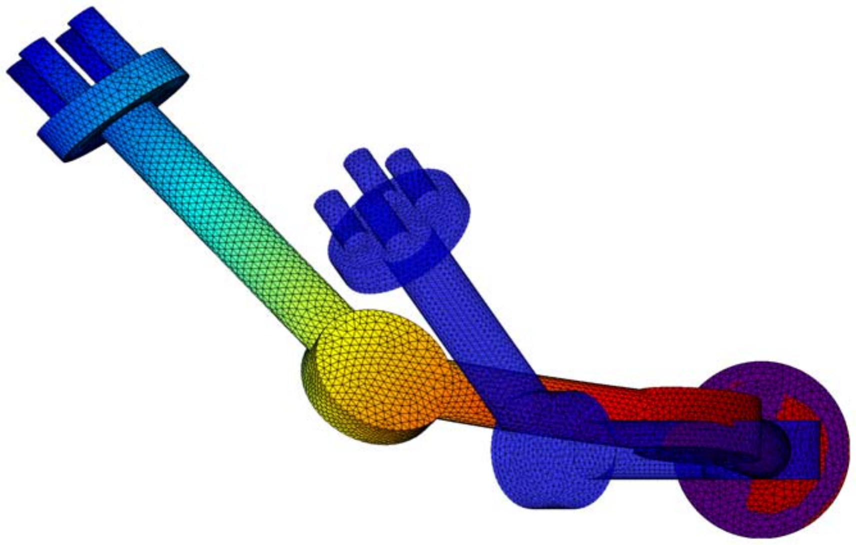


Figure 24.1: Deflection of robot arm.

with volumetric tetrahedral elements — is also shown. In this FE discretization there are $n = 60{,}030$ degrees of freedom (for technical reasons we do not count the nodes at the robot shoulder). The issue is thus how to effectively solve the linear system of equations $Ku = f$ given the very large number of degrees of freedom. In fact, many finite element discretizations result not in 10^5 unknowns but rather 10^6 or even 10^7 unknowns. The computational task is thus formidable, in particular since typically one analysis will not suffice — rather, many analyses will be required for purposes of design and optimization.

24.2 Gaussian Elimination and Sparsity

If we were to consider the most obvious tactic — find K^{-1} and then compute $K^{-1}f$ — the result would be disastrous: days of computing (if the calculation even completed). And indeed even if we were to apply a method of choice — Gaussian elimination (or LU decomposition) — without any regard to the actual structure of the matrix K, the result would still be disastrous. Modern computational solution strategies must and do take advantage of a key attribute of K — sparseness.[2] In short, there is no reason to perform operations which involve zero operands and will yield zero for a result. In MechE systems sparseness is not an exceptional attribute but rather, and very fortunately, the rule: the force in a (Hookean) spring is just determined by the deformations of nearest neighbors, not by distant neighbors; similar arguments apply in heat transfer, fluid mechanics, and acoustics. (Note this is not to say that the equilibrium displacement at one node does not depend on the applied forces at the other nodes; quite to the contrary, a force applied at one node will yield nonzero displacements at all the nodes of the system. We explore this nuance more deeply when we explain why formation of the inverse of a (sparse) matrix is a very poor idea.)

We present in Figure 24.2 the structure of the matrix K: the dots indicate nonzero entries in the matrix. We observe that most of the matrix elements are in fact zero. Indeed, of the 3,603,600,900 entries of K, 3,601,164,194 entries are zero; put differently, there are only 2,436,706

[2]Note in this unit we shall consider only *direct* solution methods; equally important, but more advanced in terms of formulation, analysis, and robust implementation (at least if we consider the more efficient varieties), are *iterative* solution methods. In actual practice, most state-of-the-art solvers include some combination of direct and iterative aspects. And in all cases, sparsity plays a key role.

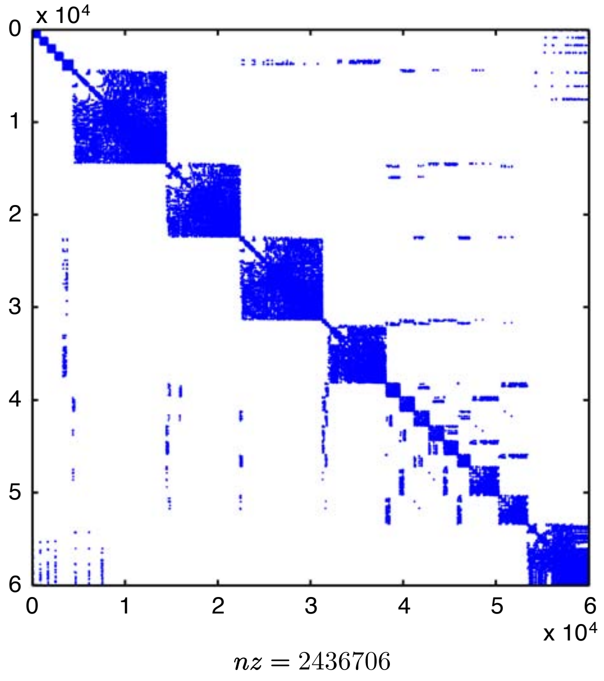


Figure 24.2: Structure of stiffness matrix K.

nonzero entries of K — only 0.06% of the entries of K are nonzero. If we exploit these zeros, *both* in our numerical approach and in the implementation/programming, we can now solve our system in reasonable time: about 230 seconds on a Mac laptop (performing a particular version of sparse Gaussian elimination based on Cholesky factorization). However, we can do still better.

In particular, although some operations "preserve" all sparsity, other operations — in particular, Gaussian elimination — result in "fill-in": zero entries become nonzero entries which thus must be included in subsequent calculations. The extent to which fill-in occurs depends on the way in which we order the equations and unknowns (which in turn determines the structure of the matrix). There is no unique way that we must choose to order the unknowns and equations: a particular node say near the elbow of the robot arm could be node (column) "1" — or node (column) "2,345"; similarly, the equilibrium equation for this node could be row "2" — or row "58,901".[3] We can thus strive to find a best ordering which minimizes fill-in and thus maximally exploits sparsity. In fact, this optimization problem is very difficult, but there are efficient heuristic procedures which yield very good results. The application of such a heuristic to our matrix K yields the new (that is, reordered) matrix K' shown in Figure 24.3. If we now reapply our sparse Cholesky approach the computational time is now very modest — only 7 seconds. Hence proper choice of algorithm and an appropriate implementation of the algorithm can reduce the computational effort from days to several seconds.

24.3 Outline

In this unit we first consider the well-posedness of linear systems: n equations in n unknowns. We understand the conditions under which a solution exists and is unique, and we motivate — from a physical perspective — the situations in which a solution might not exist or might exist but not be

[3]For our particular problem it is best to permute the unknowns and equations in the same fashion to preserve symmetry of K.

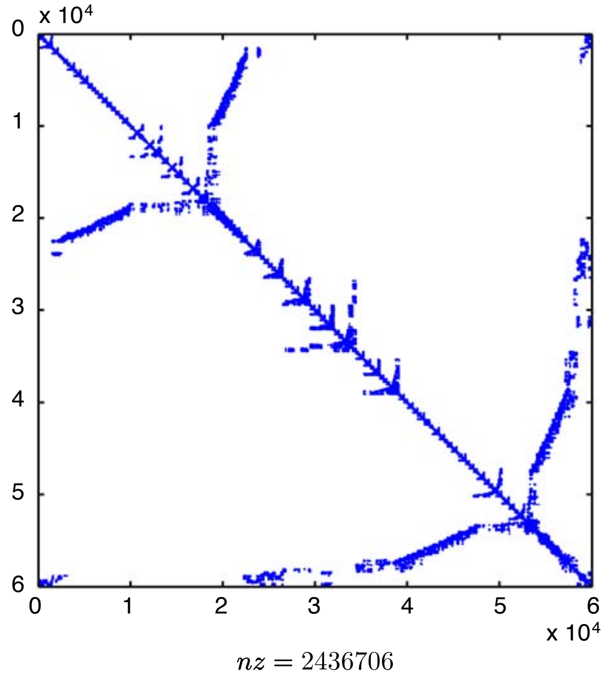


Figure 24.3: Structure of reordered stiffness matrix K'.

unique.

We next consider the basic Gaussian eliminate algorithm. We then proceed to Gaussian elimination for sparse systems — motivated by the example and numerical results presented above for the robot arm. Finally, we consider the Matlab implementation of these approaches. (Note that all results in this chapter are based on Matlab implementations.)

We notably omit several important topics: we do not consider iterative solution procedures; we do not consider, except for a few remarks, the issue of numerical stability and conditioning.

Chapter 25

Linear Systems

25.1 Model Problem: $n = 2$ Spring-Mass System in Equilibrium

25.1.1 Description

We will introduce here a simple spring-mass system, shown in Figure 25.1, which we will use throughout this chapter to illustrate various concepts associated with linear systems and associated solution techniques. Mass 1 has mass m_1 and is connected to a stationary wall by a spring with stiffness k_1. Mass 2 has mass of m_2 and is connected to the mass m_1 by a spring with stiffness k_2.

We denote the displacements of mass 1 and mass 2 by u_1 and u_2, respectively: positive values correspond to displacement away from the wall; we choose our reference such that in the absence of applied forces — the springs unstretched — $u_1 = u_2 = 0$. We next introduce (steady) forces f_1 and f_2 on mass 1 and mass 2, respectively; positive values correspond to force away from the wall. We are interested in predicting the equilibrium displacements of the two masses, u_1 and u_2, for prescribed forces f_1 and f_2.

We note that while all real systems are inherently dissipative and therefore are characterized not just by springs and masses but also dampers, the dampers (or damping coefficients) do not affect the system at equilibrium — since d/dt vanishes in the steady state — and hence for equilibrium considerations we may neglect losses. Of course, it is damping which ensures that the system ultimately achieves a stationary (time-independent) equilibrium.[1]

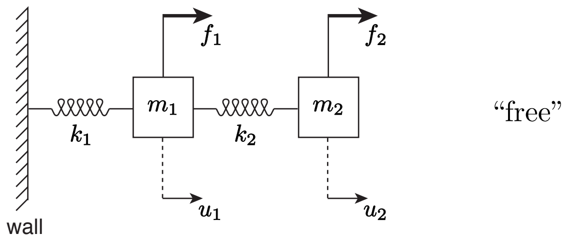


Figure 25.1: A system of two masses and two springs anchored to a wall and subject to applied forces.

[1]In some rather special cases — which we will study later in this chapter — the equilibrium displacement is indeed affected by the initial conditions and damping. This special case in fact helps us better understand the mathematical aspects of systems of linear equations.

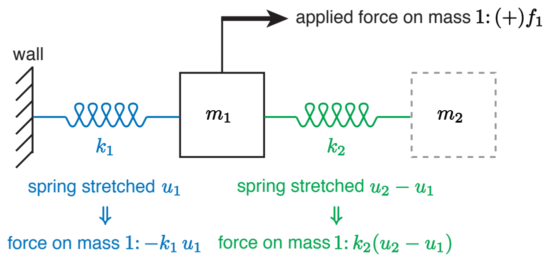

applied force on mass 1: $(+)f_1$
wall
m_1
m_2
k_1
k_2
spring stretched u_1
$\Downarrow$
force on mass 1: $-k_1 u_1$
spring stretched $u_2 - u_1$
$\Downarrow$
force on mass 1: $k_2(u_2 - u_1)$

Figure 25.2: Forces on mass 1.

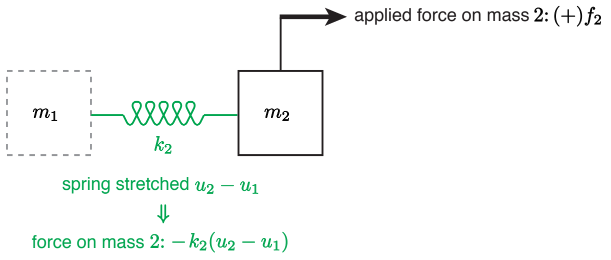

applied force on mass 2: $(+)f_2$
m_1
m_2
k_2
spring stretched $u_2 - u_1$
$\Downarrow$
force on mass 2: $-k_2(u_2 - u_1)$

Figure 25.3: Forces on mass 2.

We now derive the equations which must be satisfied by the displacements u_1 and u_2 at equilibrium. We first consider the forces on mass 1, as shown in Figure 25.2. Note we apply here Hooke's law — a constitutive relation — to relate the force in the spring to the compression or extension of the spring. In equilibrium the sum of the forces on mass 1 — the applied forces and the forces due to the spring — must sum to zero, which yields

$$f_1 - k_1 u_1 + k_2(u_2 - u_1) = 0 \ .$$

(More generally, for a system *not* in equilibrium, the right-hand side would be $m_1\ddot{u}_1$ rather than zero.) A similar identification of the forces on mass 2, shown in Figure 25.3, yields for force balance

$$f_2 - k_2(u_2 - u_1) = 0 \ .$$

This completes the physical statement of the problem.

Mathematically, our equations correspond to a system of $n = 2$ linear equations, more precisely, 2 equations in 2 unknowns:

$$(k_1 + k_2)\, u_1 - k_2\, u_2 \;=\; f_1 \ , \qquad (25.1)$$

$$-k_2\, u_1 + k_2\, u_2 \;=\; f_2 \ . \qquad (25.2)$$

Here u_1 and u_2 are *unknown*, and are placed on the left-hand side of the equations, and f_1 and f_2 *are known*, and placed on the right-hand side of the equations. In this chapter we ask several questions about this linear system — and more generally about linear systems of n equations in n unknowns. First, existence: when do the equations have a solution? Second, uniqueness: if a solution exists, is it unique? Although these issues appear quite theoretical in most cases the mathematical subtleties are in fact informed by physical (modeling) considerations. In later chapters in this unit we will ask a more obviously practical issue: how do we solve systems of linear equations efficiently?

But to achieve these many goals we must put these equations in matrix form in order to best take advantage of both the theoretical and practical machinery of linear algebra. As we have already addressed the translation of sets of equations into corresponding matrix form in Unit III (related to overdetermined systems), our treatment here shall be brief.

We write our two equations in two unknowns as $Ku = f$, where K is a 2×2 matrix, $u = (u_1 \;\; u_2)^{\mathrm{T}}$ is a 2×1 vector, and $f = (f_1 \;\; f_2)^{\mathrm{T}}$ is a 2×1 vector. The elements of K are the coefficients of the equations (25.1) and (25.2):

$$\begin{array}{cccccc} & \textit{unknown} & & \textit{known} & & \\ \begin{pmatrix} k_1+k_2 & -k_2 \\ -k_2 & k_2 \end{pmatrix} & \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} & = & \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} & \cdots & \begin{matrix} \leftarrow \text{Equation (25.1)} \\ \leftarrow \text{Equation (25.2)} \end{matrix} \\ K & u & & f & & \\ 2\times 2 & 2\times 1 & & 2\times 1 & & \end{array} \; . \tag{25.3}$$

We briefly note the connection between equations (25.3) and (25.1). We first note that $Ku = F$ implies equality of the two vectors Ku and F and hence equality of each component of Ku and F. The first component of the vector Ku, from the row interpretation of matrix multiplication,[2] is given by $(k_1 + k_2)u_1 - k_2u_2$; the first component of the vector F is of course f_1. We thus conclude that $(Ku)_1 = f_1$ correctly produces equation (25.1). A similar argument reveals that the $(Ku)_2 = f_2$ correctly produces equation (25.2).

25.1.2 SPD Property

We recall that a real $n \times n$ matrix A is Symmetric Positive Definite (SPD) if A is symmetric

$$A^{\mathrm{T}} = A \; , \tag{25.4}$$

and A is positive definite

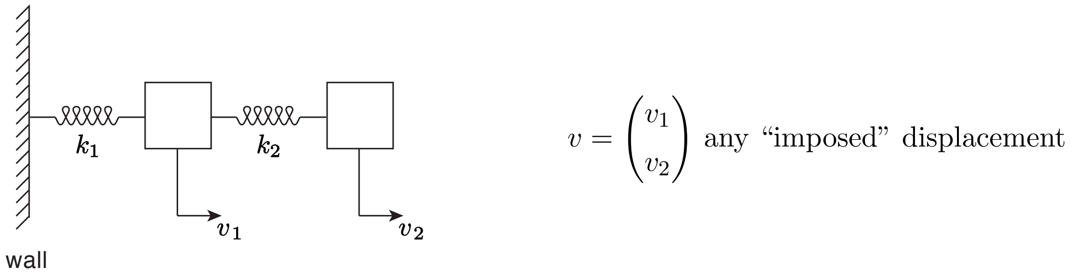
$$v^{\mathrm{T}}Av > 0 \text{ for any } v \neq 0 \; . \tag{25.5}$$

Note in equation (25.5) that Av is an $n \times 1$ vector and hence $v^{\mathrm{T}}(Av)$ is a scalar — a real number. Note also that the positive definite property (25.5) implies that if $v^{\mathrm{T}}Av = 0$ then v must be the zero vector. There is also a connection to eigenvalues: symmetry implies real eigenvalues, and positive definite implies strictly positive eigenvalues (and in particular, no zero eigenvalues).

There are many implications of the SPD property, all very pleasant. In the context of the current unit, an SPD matrix ensures positive eigenvalues which in turn will ensure existence and uniqueness of a solution — which is the topic of the next section. Furthermore, an SPD matrix ensures stability of the Gaussian elimination process — the latter is the topic in the following chapters. We also note that, although in this unit we shall focus on direct solvers, SPD matrices also offer advantages in the context of iterative solvers: the very simple and efficient conjugate gradient method can be applied (only to) SPD matrices. The SPD property is also the basis of minimization principles which serve in a variety of contexts. Finally, we note that the SPD property is often, though not always, tied to a natural physical notion of energy.

We shall illustrate the SPD property for our simple 2×2 matrix K associated with our spring system. In particular, we now again consider our system of two springs and two masses but now we introduce an arbitrary imposed displacement vector $v = (v_1 \;\; v_2)^{\mathrm{T}}$, as shown in Figure 25.4. In this case our matrix A is given by K where

[2] In many, but not all, cases it is more intuitive to develop matrix equations from the row interpretation of matrix multiplication; however, as we shall see, the column interpretation of matrix multiplication can be very important from the theoretical perspective.



k_1
k_2
v_1
v_2
wall
$v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ any "imposed" displacement

Figure 25.4: Spring-mass system: imposed displacements v.

$$K = \begin{pmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{pmatrix} .$$

We shall assume that $k_1 > 0$ and $k_2 > 0$ — our spring constants are strictly positive. We shall return to this point shortly.

We can then form the scalar $v^{\mathrm{T}}Kv$ as

$$\begin{aligned} v^{\mathrm{T}}Kv &= v^{\mathrm{T}} \underbrace{\begin{pmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}} \\ &= (v_1 \quad v_2) \underbrace{\begin{pmatrix} (k_1 + k_2)v_1 & -k_2 v_2 \\ -k_2 v_1 & k_2 v_2 \end{pmatrix}}_{Kv} \\ &= v_1^2(k_1 + k_2) - v_1 v_2 k_2 - v_2 v_1 k_2 + v_2^2 k_2 \\ &= v_1^2 k_1 + \left(v_1^2 - 2v_1 v_2 + v_2^2\right) k_2 \\ &= k_1 v_1^2 + k_2 (v_1 - v_2)^2 . \end{aligned}$$

We now inspect this result.

In particular, we may conclude that, under our assumption of positive spring constants, $v^{\mathrm{T}}Kv \geq 0$. Furthermore, $v^{\mathrm{T}}Kv$ can only be zero if $v_1 = 0$ and $v_1 = v_2$, which in turn implies that $v^{\mathrm{T}}Kv$ can only be zero if both v_1 and v_2 are zero — $v = 0$. We thus see that K is SPD: $v^{\mathrm{T}}Kv > 0$ unless $v = 0$ (in which case of course $v^{\mathrm{T}}Kv = 0$). Note that if either $k_1 = 0$ or $k_2 = 0$ then the matrix is not SPD: for example, if $k_1 = 0$ then $v^{\mathrm{T}}Kv = 0$ for any $v = (c \;\; c)^{\mathrm{T}}$, c a real constant; similarly, if $k_2 = 0$, then $v^{\mathrm{T}}Kv = 0$ for any $v = (0 \;\; c)^{\mathrm{T}}$, c a real constant.

We can in this case readily identify the connection between the SPD property and energy. In particular, for our spring system, the potential energy in the spring system is simply $\frac{1}{2}v^{\mathrm{T}}Kv$:

$$\text{PE (potential/elastic energy)} =$$

$$\underbrace{\frac{1}{2}k_1 v_1^2}_{\substack{\text{energy in} \\ \text{spring 1}}} + \underbrace{\frac{1}{2}k_2(v_2 - v_1)^2}_{\substack{\text{energy in} \\ \text{spring 2}}} = \frac{1}{2}v^{\mathrm{T}}Av > 0 \qquad (\text{unless } v = 0) ,$$

where of course the final conclusion is only valid for strictly positive spring constants.

Finally, we note that many MechE systems yield models which in turn may be described by SPD systems: structures (trusses, ...); linear elasticity; heat conduction; flow in porous media (Darcy's Law); Stokes (or creeping) flow of an incompressible fluid. (This latter is somewhat complicated by the incompressibility constraint.) All these systems are very important in practice and indeed ubiquitous in engineering analysis and design. However, it is also essential to note that many other very important MechE phenomena and systems — for example, forced convection heat transfer, non-creeping fluid flow, and acoustics — do *not* yield models which may be described by SPD matrices.

25.2 Existence and Uniqueness: $n = 2$

25.2.1 Problem Statement

We shall now consider the existence and uniqueness of solutions to a general system of ($n =$) 2 equations in ($n =$) 2 unknowns. We first introduce a matrix A and vector f as

$$\left.\begin{aligned} 2 \times 2 \text{ matrix} \quad A \quad &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \\ 2 \times 1 \text{ vector} \quad f \quad &= \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \end{aligned}\right. \quad ;$$

our equation for the 2×1 unknown vector u can then be written as

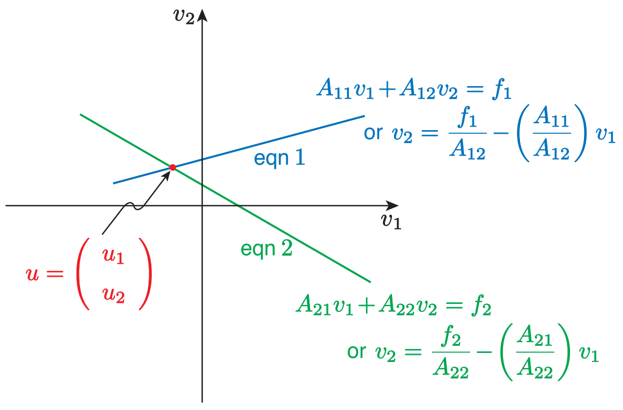
$$Au = f\,, \quad \text{or} \quad \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}\,, \quad \text{or} \quad \left.\begin{aligned} A_{11}u_1 + A_{12}u_2 &= f_1 \\ A_{21}u_1 + A_{22}u_2 &= f_2 \end{aligned}\right\} .$$

Note these three expressions are equivalent statements proceeding from the more abstract to the more concrete. We now consider existence and uniqueness; we shall subsequently interpret our general conclusions in terms of our simple $n = 2$ spring-mass system.

25.2.2 Row View

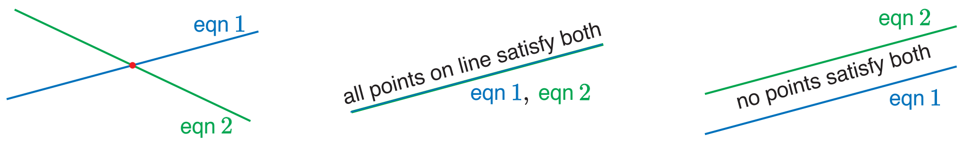
We first consider the *row* view, similar to the row view of matrix multiplication. In this perspective we consider our solution vector $u = (u_1 \;\; u_2)^{\mathrm{T}}$ as a point (u_1, u_2) in the two dimensional Cartesian plane; a *general* point in the plane is denoted by (v_1, v_2) corresponding to a vector $(v_1 \;\; v_2)^{\mathrm{T}}$. In particular, u is the particular point in the plane which lies both on the straight line described by the first equation, $(Av)_1 = f_1$, denoted 'eqn1' and shown in Figure 25.5 in blue, *and* on the straight line described by the second equation, $(Av)_2 = f_2$, denoted 'eqn2' and shown in Figure 25.5 in green.

We directly observe three possibilities, familiar from any first course in algebra; these three cases are shown in Figure 25.6. In case (i), the two lines are of different slope and there is clearly one and only one intersection: the solution thus exists and is furthermore unique. In case (ii) the two lines are of the same slope and furthermore coincident: a solution exists, but it is not unique — in fact, there are an infinity of solutions. This case corresponds to the situation in which the two equations in fact contain *identical* information. In case (iii) the two lines are of the same slope but *not* coincident: no solution exists (and hence we need not consider uniqueness). This case corresponds to the situation in which the two equations contain *inconsistent* information.



v_2
v_1
$A_{11}v_1 + A_{12}v_2 = f_1$
or $v_2 = \frac{f_1}{A_{12}} - \left(\frac{A_{11}}{A_{12}}\right) v_1$
eqn 1
eqn 2
$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$
$A_{21}v_1 + A_{22}v_2 = f_2$
or $v_2 = \frac{f_2}{A_{22}} - \left(\frac{A_{21}}{A_{22}}\right) v_1$

Figure 25.5: Row perspective: u is the intersection of two straight lines.



(i)	(ii)	(iii)
exists ✓	exists ✓	exists ✗
unique ✓	unique ✗	~~unique~~
	redundant information,	*inconsistent* information
	infinity of solutions	*no* solution

Figure 25.6: Three possibilities for existence and uniqueness; row interpretation.

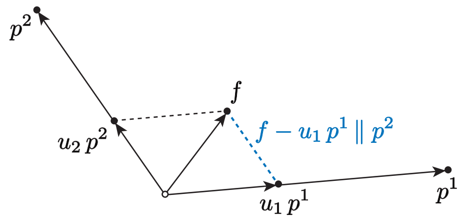


Figure 25.7: Parallelogram construction for the case in case in which a unique solution exists.

We see that the condition for (both) existence and uniqueness is that the slopes of 'eqn1' and 'eqn2' must be different, or $A_{11}/A_{12} \neq A_{21}/A_{22}$, or $A_{11}A_{22} - A_{12}A_{21} \neq 0$. We recognize the latter as the more familiar condition $det(A) \neq 0$. In summary, if $det(A) \neq 0$ then our matrix A is non-singular and the system $Au = f$ has a unique solution; if $det(A) \neq 0$ then our matrix A is singular and either our system has an infinity of solutions or no solution, depending on f. (In actual practice the determinant condition is not particularly practical computationally, and serves primarily as a convenient "by hand" check for very small systems.) We recall that a non-singular matrix A has an inverse A^{-1} and hence in case (i) we can write $u = A^{-1}f$; we presented this equation earlier under the assumption that A is non-singular — now we have provided the condition under which this assumption is true.

25.2.3 The Column View

We next consider the column view, analogous to the column view of matrix multiplication. In particular, we recall from the column view of matrix-vector multiplication that we can express Au as

$$Au = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \underbrace{\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}}_{p^1} u_1 + \underbrace{\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}}_{p^2} u_2 \quad ,$$

where p^1 and p^2 are the first and second column of A, respectively. Our system of equations can thus be expressed as

$$Au = f \quad \Leftrightarrow \quad p^1 u_1 + p^2 u_2 = f \ .$$

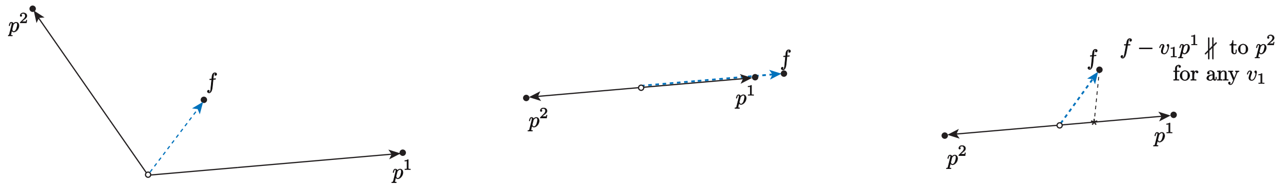
Thus the question of existence and uniqueness can be stated alternatively: is there a (unique?) combination, u, of columns p^1 and p^2 which yields f?

We start by answering this question pictorially in terms of the familiar parallelogram construction of the sum of two vectors. To recall the parallelogram construction, we first consider in detail the case shown in Figure 25.7. We see that in the instance depicted in Figure 25.7 there is clearly a unique solution: we choose u_1 such that $f - u_1 p^1$ is parallel to p^2 (there is clearly only one such value of u_1); we then choose u_2 such that $u_2 p^2 = f - u_1 p^1$.

We can then identify, in terms of the parallelogram construction, three possibilities; these three cases are shown in Figure 25.8. Here case (i) is the case already discussed in Figure 25.7: a unique solution exists. In both cases (ii) and (iii) we note that

$$p^2 = \gamma p^1 \quad \text{or} \quad p^2 - \gamma p^1 = 0 \ (p^1 \text{ and } p^2 \text{ are linearly dependent})$$

for some γ, where γ is a scalar; in other words, p^1 and p^2 are colinear — *point in the same direction to within a sign* (though p^1 and p^2 may of course be of different magnitude). We now discuss these two cases in more detail.



(i)

exists ✓

unique ✓

(ii)

exists ✓

unique ✗

(only p^1, *or*
more p^1 and some p^2, *or* ...)

(iii)

exists ✗

~~unique~~

Figure 25.8: Three possibilities for existence and uniqueness; the column perspective.

In case (ii), p^1 and p^2 are colinear but f *also* is colinear with p^1 (and p^2) — say $f = \beta p^1$ for some scalar β. We can thus write

$$\begin{aligned} f &= p^1 \cdot \beta + p^2 \cdot 0 \\ &= \begin{pmatrix} p^1 & p^2 \end{pmatrix} \begin{pmatrix} \beta \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \underbrace{\begin{pmatrix} \beta \\ 0 \end{pmatrix}}_{u^*} \\ &= Au^* \; , \end{aligned}$$

and hence u^* is *a* solution. However, we also know that $-\gamma p^1 + p^2 = 0$, and hence

$$\begin{aligned} 0 &= p^1 \cdot (-\gamma) + p^2 \cdot (1) \\ &= \begin{pmatrix} p^1 & p^2 \end{pmatrix} \begin{pmatrix} -\gamma \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} -\gamma \\ 1 \end{pmatrix} \\ &= A \begin{pmatrix} -\gamma \\ 1 \end{pmatrix} \; . \end{aligned}$$

Thus, for any α,

$$u = \underbrace{u^* + \alpha \begin{pmatrix} -\gamma \\ 1 \end{pmatrix}}_{\text{infinity of solutions}}$$

satisfies $Au = f$, since

$$
\begin{aligned}
A\left(u^* + \alpha \begin{pmatrix} -\gamma \\ 1 \end{pmatrix}\right) &= Au^* + A\left(\alpha \begin{pmatrix} -\gamma \\ 1 \end{pmatrix}\right) \\
&= Au^* + \alpha A \begin{pmatrix} -\gamma \\ 1 \end{pmatrix} \\
&= f + \alpha \cdot 0 \\
&= f \ .
\end{aligned}
$$

This demonstrates that in case (*ii*) there are an infinity of solutions parametrized by the arbitrary constant α.

Finally, we consider case (*iii*). In this case it is clear from our parallelogram construction that for no choice of v_1 will $f - v_1 p^1$ be parallel to p^2, and hence for no choice of v_2 can we form $f - v_1 p^1$ as $v_2 p^2$. Put differently, a linear combination of two colinear vectors p^1 and p^2 can not combine to form a vector perpendicular to both p^1 and p^2. Thus no solution exists.

Note that the vector $(-\gamma \quad 1)^{\mathrm{T}}$ is an eigenvector of A corresponding to a zero eigenvalue.[3] By definition the matrix A has no effect on an eigenvector associated with a zero eigenvalue, and it is for this reason that if we have one solution to $Au = f$ then we may add to this solution *any* multiple — here α — of the zero-eigenvalue eigenvector to obtain yet another solution. More generally a matrix A is non-singular if and only if it has no zero eigenvalues; in that case — case (*i*) — the inverse exists and we may write $u = A^{-1} f$. On the other hand, if A has any zero eigenvalues then A is singular and the inverse does not exist; in that case $Au = f$ may have either many solutions or no solutions, depending on f. From our discussion of SPD systems we also note that A SPD is a sufficient (but not necessary) condition for the existence of the inverse of A.

25.2.4 A Tale of Two Springs

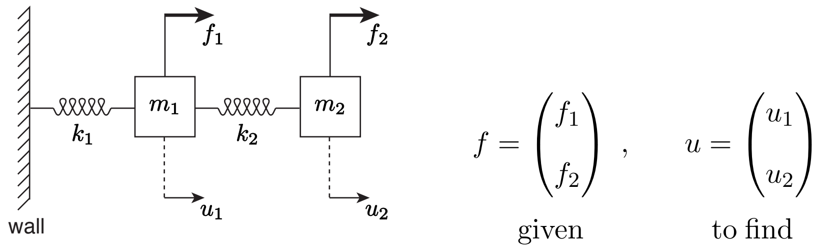
We now interpret our results for existence and uniqueness for a mechanical system — our two springs and masses — to understand the connection between the model and the mathematics. We again consider our two masses and two springs, shown in Figure 25.9, governed by the system of equations

$$
Au = f \quad \text{for} \quad A = K \equiv \begin{pmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{pmatrix} \ .
$$

We analyze three different scenarios for the spring constants and forces, denoted (I), (II), and (III), which we will see correspond to cases (*i*), (*ii*), and (*iii*), respectively, as regards existence and uniqueness. We present first (I), then (III), and then (II), as this order is more physically intuitive.

(I) In scenario (I) we choose $k_1 = k_2 = 1$ (more physically we would take $k_1 = k_2 = \overline{k}$ for some value of $\overline{k}$ expressed in appropriate units — but our conclusions will be the same) and $f_1 = f_2 = 1$ (more physically we would take $f_1 = f_2 = \overline{f}$ for some value of $\overline{f}$ expressed in appropriate units — but our conclusions will be the same). In this case our matrix A and

[3]All scalar multiples of this eigenvector define what is known as the right nullspace of A.

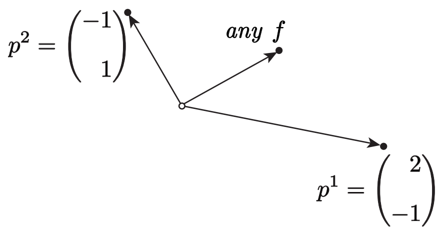


f_1
f_2
m_1
m_2
k_1
k_2
u_1
u_2
wall
$f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$,
$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$
given
to find

Figure 25.9: System of equilibrium equations for two springs and two masses.

associated column vectors p^1 and p^2 take the form shown below. It is clear that p^1 and p^2 are not colinear and hence a unique solution exists for any f. We are in case (i).

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$$

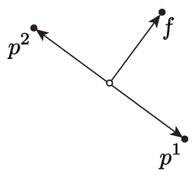


$p^2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
any f
$p^1 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$

case (i): exists ✓, unique ✓

(III) In scenario (III) we chose $k_1 = 0$, $k_2 = 1$ and $f_1 = f_2 = 1$. In this case our vector f and matrix A and associated column vectors p^1 and p^2 take the form shown below. It is clear that a linear combination of p^1 and p^2 can not possibly represent f — and hence no solution exists. We are in case (iii).

$$f = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$



$$A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

case (iii): exists ✗, ~~unique~~

We can readily identify the cause of the difficulty. For our particular choice of spring constants in scenario (III) the first mass is no longer connected to the wall (since $k_1 = 0$); thus our spring system now appears as in Figure 25.10. We see that there is a net force on our *system* (of two masses) — the net force is $f_1 + f_2 = 2 \neq 0$ — and hence it is clearly inconsistent to assume equilibrium.[4] In even greater detail, we see that the equilibrium equations for each mass are inconsistent (note $f_{\text{spr}} = k_2(u_2 - u_1)$) and hence we must replace the zeros on the right-hand sides with mass $\times$ acceleration terms. At fault here is not the mathematics but rather the model provided for the physical system.

[4]In contrast, in scenario (I), the wall provides the necessary reaction force in order to ensure equilibrium.

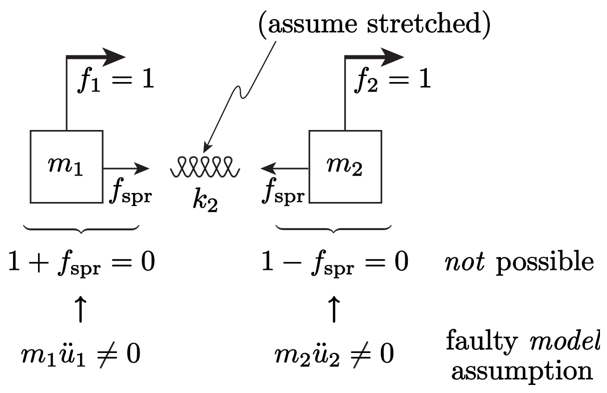
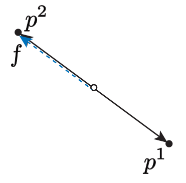


Figure 25.10: Scenario III

(II) In this scenario we choose $k_1 = 0$, $k_2 = 1$ and $f_1 = 1, f_2 = -1$. In this case our vector f and matrix A and associated column vectors p^1 and p^2 take the form shown below. It is clear that a linear combination of p^1 and p^2 now *can* represent f — and in fact there are many possible combinations. We are in case (ii).

$$f = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$



case (ii): exists ✓, unique ✗

We can explicitly construct the family of solutions from the general procedure described earlier:

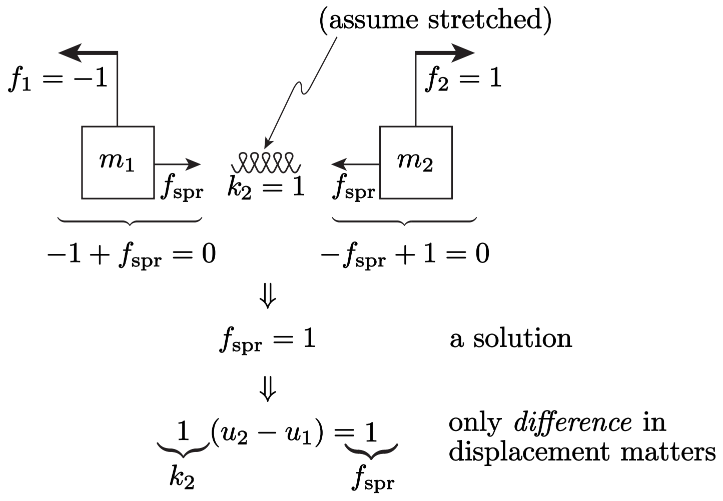
$$p^2 \;=\; = \underbrace{-1}_{\gamma} p^1 \;,$$

$$f \;=\; \underbrace{-1}_{\beta} p^1 \Rightarrow u^* = \begin{pmatrix} = -1 \\ 0 = \end{pmatrix}$$

$$\Downarrow$$

$$u \;=\; u^* + \alpha \begin{pmatrix} -\gamma \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} -1 \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$



(assume stretched)
$f_1 = -1$
$f_2 = 1$
m_1
m_2
f_{spr}
$k_2 = 1$
f_{spr}
$\underbrace{-1 + f_{\text{spr}} = 0}$
$\underbrace{-f_{\text{spr}} + 1 = 0}$
$\Downarrow$
$f_{\text{spr}} = 1$
a solution
$\Downarrow$
$\underbrace{1}_{k_2}(u_2 - u_1) = \underbrace{1}_{f_{\text{spr}}}$
only *difference* in displacement matters

Figure 25.11: Scenario II. (Note on the left mass the f_1 arrow indicates the direction of the force $f_1 = -1$, not the direction of positive force.)

for *any* α. Let us check the result explicitly:

$$A\left(\begin{pmatrix}-1\\0\end{pmatrix}+\begin{pmatrix}\alpha\\\alpha\end{pmatrix}\right)=\begin{pmatrix}1 & -1\\-1 & 1\end{pmatrix}\begin{pmatrix}1+\alpha\\\alpha\end{pmatrix}$$

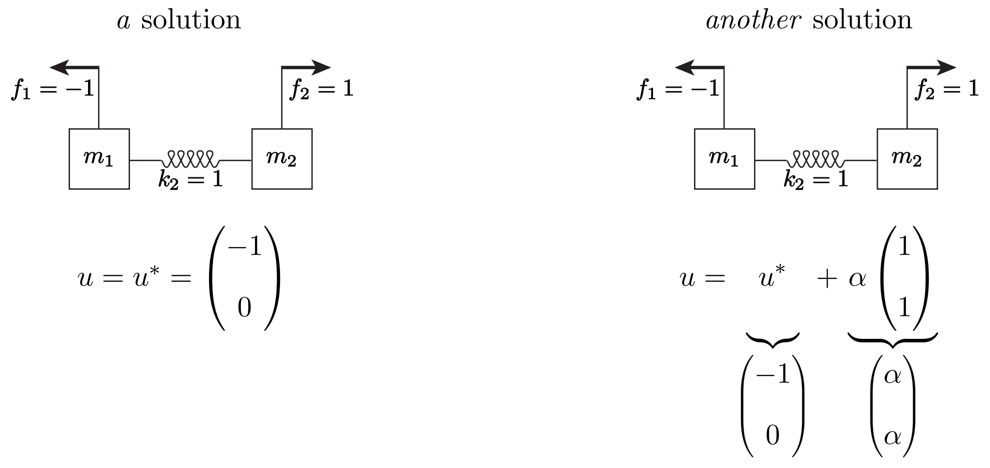
$$=\begin{pmatrix}(-1+\alpha)-\alpha\\(1-\alpha)+\alpha\end{pmatrix}$$

$$=\begin{pmatrix}-1\\1\end{pmatrix}$$

$$= f\ ,$$

as desired. Note that the zero-eigenvalue eigenvector here is given by $(-\gamma \quad 1)^{\mathrm{T}} = (1 \quad 1)^{\mathrm{T}}$ and corresponds to an equal (or translation) shift in both displacements, which we will now interpret physically.

In particular, we can readily identify the cause of the non-uniqueness. For our choice of spring constants in scenario (II) the first mass is no longer connected to the wall (since $k_1 = 0$), just as in scenario (III). Thus our spring system now appears as in Figure 25.11. But unlike in scenario (III), in scenario (II) the net force on the system is zero — f_1 and f_2 point in opposite directions — and hence an equilibrium is possible. Furthermore, we see that each mass is in equilibrium for a spring force $f_{\text{spr}} = 1$. Why then is there not a *unique* solution? Because to obtain $f_{\text{spr}} = 1$ we may choose any displacements u such that $u_2 - u_1 = 1$ (for $k_2 = 1$): the system is not anchored to wall — it just floats — and thus equilibrium is maintained if we shift (or translate) both masses by the same displacement (our eigenvector) such that the "stretch" remains invariant. This is illustrated in Figure 25.12, in which α is the shift in displacement. Note α is *not* determined by the *equilibrium* model; α *could* be determined from a *dynamical* model and in particular would depend on the initial conditions *and* the



a solution
$f_1 = -1$
$f_2 = 1$
m_1
m_2
$k_2 = 1$
$u = u^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$
another solution
$f_1 = -1$
$f_2 = 1$
m_1
m_2
$k_2 = 1$
$u = \underbrace{u^*}_{\begin{pmatrix} -1 \\ 0 \end{pmatrix}} + \underbrace{\alpha \begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{\begin{pmatrix} \alpha \\ \alpha \end{pmatrix}}$

Figure 25.12: Scenario (II): non-uniqueness.

damping in the system.

We close this section by noting that for scenario (I) $k_1 > 0$ and $k_2 > 0$ and hence A ($\equiv K$) is SPD: thus A^{-1} exists and $Au = f$ has a unique solution for any forces f. In contrast, in scenarios (II) and (III), $k_1 = 0$, and hence A is no longer SPD, and we are no longer guaranteed that A^{-1} exists — and indeed it does not exist. We also note that in scenario (III) the zero-eigenvalue eigenvector $(1 \;\; 1)^{\mathrm{T}}$ is precisely the v which yields zero energy, and indeed a shift (or translation) of our unanchored spring system does not affect the energy in the spring.

25.3 A "Larger" Spring-Mass System: n Degrees of Freedom

We now consider the equilibrium of the system of n springs and masses shown in Figure 25.13. (This string of springs and masses in fact is a model (or discretization) of a continuum truss; each spring-mass is a small segment of the truss.) Note for $n = 2$ we recover the small system studied in the preceding sections. This larger system will serve as a more "serious" model problem both as regards existence and uniqueness but even more importantly as regard computational procedures. We then consider force balance on mass 1,

$$\sum \text{forces on mass } 1 = 0$$

$$\Rightarrow f_1 - k_1 u_1 + k_2(u_2 - u_1) = 0 \ ,$$

and then on mass 2,

$$\sum \text{forces on mass } 2 = 0$$

$$\Rightarrow f_2 - k_2(u_2 - u_1) + k_3(u_3 - u_2) = 0 \ ,$$

and then on a typical interior mass i (hence $2 \le i \le n-1$)

$$\sum \text{forces on mass } i = 0 \ (i \ne 1, \ i \ne n)$$

$$\Rightarrow f_i - k_i(u_i - u_{i-1}) + k_{i+1}(u_{i+1} - u_i) = 0 \ ,$$

and finally on mass n,

$$\sum \text{forces on mass } n = 0$$

$$\Rightarrow f_n - k_n(u_n - u_{n-1}) = 0 \ .$$

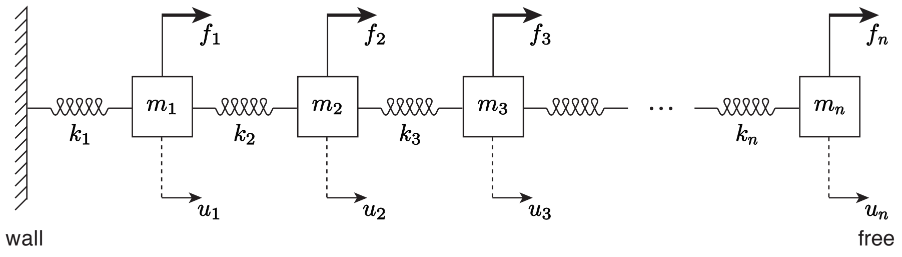


Figure 25.13: System of n springs and masses.

We can write these equations as

$$\begin{array}{cccccc}
(k_1+k_2)u_1 & -\,k_2u_2 & 0\dots & & & = f_1 \\
-\,k_2u_1 & +\,(k_2+k_3)u_2 & -\,k_3u_3 & 0\dots & & = f_2 \\
0 & -\,k_3u_2 & +\,(k_3+k_4)u_3 & -\,k_4u_4 & & = f_3 \\
 & & \ddots & & & \vdots \\
 & & \dots 0 & -\,k_nu_{n-1} & +\,k_nu_n & = f_n
\end{array}$$

or

$$\underbrace{\begin{pmatrix}
k_1+k_2 & -k_2 & & & & \\
-k_2 & k_2+k_3 & -k_3 & & \mathbf{0} & \\
 & -k_3 & k_3+k_4 & -k_4 & & \\
 & & \ddots & \ddots & \ddots & \\
 & \mathbf{0} & & & & -k_n \\
 & & & & -k_n & k_n
\end{pmatrix}}_{\substack{K \\ n\times n}}
\underbrace{\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix}}_{\substack{u \\ n\times 1}}
\underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix}}_{\substack{f \\ n\times 1}}$$

which is simply $Au = f$ $(A \equiv K)$ but now for n equations in n unknowns.

In fact, the matrix K has a number of special properties. In particular, K is *sparse* — K is mostly zero entries since only "nearest neighbor" connections affect the spring displacement and hence the force in the spring[5]; *tri-diagonal* — the nonzero entries are all on the main diagonal and diagonal just below and just above the main diagonal; *symmetric* — $K^{\mathrm{T}} = K$; and positive definite (as proven earlier for the case $n = 2$) — $\frac{1}{2}(v^{\mathrm{T}}Kv)$ is the potential/elastic energy of the system. Some of these properties are important to establish existence and uniqueness, as discussed in the next section; some of the properties are important in the efficient computational solution of $Ku = f$, as discussed in the next chapters of this unit.

[5]This sparsity property, ubiquitous in MechE systems, will be the topic of its own chapter subsequently.

25.4 Existence and Uniqueness: General Case (Square Systems)

We now consider a general system of n equations in n unknowns,

$$\underbrace{A}_{\text{given}}\ \underbrace{u}_{\text{to find}} = \underbrace{f}_{\text{given}}$$

where A is $n \times n$, u is $n \times 1$, and f is $n \times 1$.

If A has n independent columns then A is non-singular, A^{-1} exists, and $Au = f$ has a unique solution u. There are in fact many ways to confirm that A is non-singular: A has n independent columns; A has n independent rows; A has nonzero determinant; A has no zero eigenvalues; A is SPD. (We will later encounter another condition related to Gaussian elimination.) Note all these conditions are necessary and sufficient except the last: A is SPD is only a sufficient condition for non-singular A. Conversely, if any of the necessary conditions is *not* true then A is singular and $Au = f$ either will have many solutions or no solution, depending of f.[6] In short, all of our conclusions for $n = 2$ directly extend to the case of general n.

[6]Note in the computational context we must also understand and accommodate "*nearly*" singular systems. We do not discuss this more advanced topic further here.

Chapter 26

Gaussian Elimination and Back Substitution

26.1 A 2×2 System ($n = 2$)

Let us revisit the two-mass mass-spring system ($n = 2$) considered in the previous chapter; the system is reproduced in Figure 26.1 for convenience. For simplicity, we set both spring constants to unity, i.e. $k_1 = k_2 = 1$. Then, the equilibrium displacement of mass m_1 and m_2, u_1 and u_2, is described by a linear system

$$\underset{(K)}{A}\, u = f \quad \rightarrow \quad \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}, \tag{26.1}$$

where f_1 and f_2 are the forces applied to m_1 and m_2. We will use this 2×2 system to describe a systematic two-step procedure for solving a linear system: a linear solution strategy based on *Gaussian elimination* and *back substitution*. While the description of the solution strategy may appear overly detailed, we focus on presenting a systematic approach such that the approach generalizes to $n \times n$ systems and can be carried out by a computer.

By row-wise interpretation of the linear system, we arrive at a system of linear equations

$$\begin{aligned} \underset{\text{pivot}}{2}\, u_1 \ - u_2 &= f_1 \\ -1 u_1 + u_2 &= f_2 . \end{aligned}$$

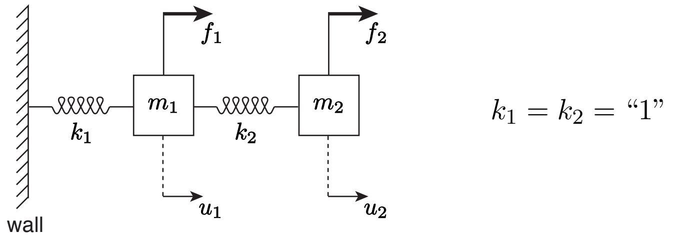


Figure 26.1: $n = 2$ spring-mass system.

We recognize that we can eliminate u_1 from the second equation by adding $1/2$ of the first equation to the second equation. The scaling factor required to eliminate the first coefficient from the second equation is simply deduced by diving the first coefficient of the second equation (-1) by the "pivot" — the leading coefficient of the first equation (2) — and negating the sign; this systematic procedure yields $(-(-1)/2) = 1/2$ in this case. Addition of $1/2$ of the first equation to the second equation yields a new second equation

$$\begin{array}{rcl} u_1 - \frac{1}{2}u_2 & = & \frac{1}{2}f_1 \\ -u_1 + u_2 & = & f_2 \\ \hline 0u_1 + \frac{1}{2}u_2 & = & \frac{1}{2}f_1 + f_2 \end{array} .$$

Note that the solution to the linear system is unaffected by this addition procedure as we are simply adding "0" — expressed in a rather complex form — to the second equation. (More precisely, we are adding the same value to both sides of the equation.)

Collecting the new second equation with the original first equation, we can rewrite our system of linear equations as

$$\begin{aligned} 2u_1 - u_2 &= f_1 \\ 0u_1 + \frac{1}{2}u_2 &= f_2 + \frac{1}{2}f_1 \end{aligned}$$

or, in the matrix form,

$$\underbrace{\begin{pmatrix} 2 & -1 \\ 0 & \frac{1}{2} \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}}_{u} = \underbrace{\begin{pmatrix} f_1 \\ f_2 + \frac{1}{2}f_1 \end{pmatrix}}_{\hat{f}} .$$

Here, we have identified the new matrix, which is *upper triangular*, by U and the modified right-hand side by $\hat{f}$. In general, an upper triangular matrix has all zeros below the main diagonal, as shown in Figure 26.2; the zero entries of the matrix are shaded in blue and (possibly) nonzero entries are shaded in red. For the 2×2 case, upper triangular simply means that the $(2, 1)$ entry is zero. Using the newly introduced matrix U and vector $\hat{f}$, we can concisely write our 2×2 linear system as

$$U u = \hat{f} . \tag{26.2}$$

The key difference between the original system Eq. (26.1) and the new system Eq. (26.2) is that the new system is upper triangular; this leads to great simplification in the solution procedure as we now demonstrate.

First, note that we can find u_2 from the second equation in a straightforward manner, as the equation only contains one unknown. A simple manipulation yields

$$\begin{array}{c} \text{eqn 2} \\ \text{of } U \end{array} \qquad \frac{1}{2}u_2 = f_2 + \frac{1}{2}f_1 \;\Rightarrow\; u_2 = f_1 + 2f_2$$

Having obtained the value of u_2, we can now treat the variable as a "known" (rather than "unknown") from hereon. In particular, the first equation now contains only one "unknown", u_1; again,

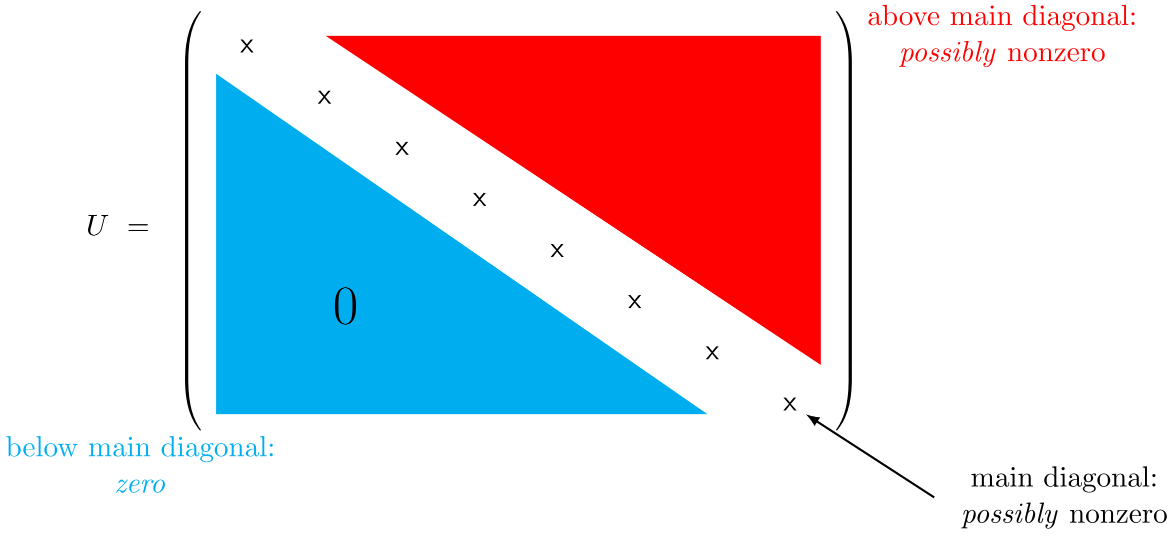


Figure 26.2: Illustration of an upper triangular matrix.

it is trivial to solve for the single unknown of a single equation, i.e.

$$\begin{array}{lll}
\begin{array}{c}\text{eqn 1}\\ \text{of } U\end{array} & 2u_1 - u_2 & = f_1 \\[2ex]
\Rightarrow & 2u_1 & = f_1 + \underset{\text{(already know)}}{u_2} \\[2ex]
\Rightarrow & 2u_1 & = f_1 + f_1 + 2f_2 = 2(f_1 + f_2) \\[2ex]
\Rightarrow & u_1 & = (f_1 + f_2)\ .
\end{array}$$

Note that, even though the 2×2 linear system Eq. (26.2) is still a fully coupled system, the solution procedure for the upper triangular system is greatly simplified because we can sequentially solve (two) single-variable-single-unknown equations.

In above, we have solved a simple 2×2 system using a *systematic* two-step approach. In the first step, we reduced the original linear system into an upper triangular system; this step is called *Gaussian elimination (GE)*. In the second step, we solved the upper triangular system sequentially starting from the equation described by the last row; this step is called *back substitution (BS)*. Schematically, our linear system solution strategy is summarized by

$$\begin{cases} \text{GE}: & Au = f \ \Rightarrow \ Uu = \hat{f} & \text{STEP 1} \\[1ex] \text{BS}: & Uu = \hat{f} \ \Rightarrow \ u & \text{STEP 2.} \end{cases}$$

This systematic approach to solving a linear system in fact generalize to general $n \times n$ systems, as we will see shortly. Before presenting the general procedure, let us provide another concrete example using a 3×3 system.

26.2 A 3×3 System ($n = 3$)

Figure 26.3 shows a three-mass spring-mass system ($n = 3$). Again assuming unity-stiffness springs

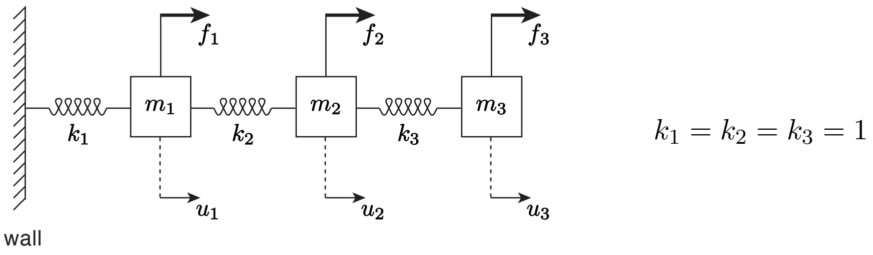


Figure 26.3: A $n = 3$ spring-mass system.

for simplicity, we obtain a linear system describing the equilibrium displacement:

$$\underset{(K)}{A}\, u = f \quad \rightarrow \quad \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} .$$

As described in the previous chapter, the linear system admits a unique solution for a given f.

Let us now carry out Gaussian elimination to transform the system into an upper triangular system. As before, in the first step, we identify the first entry of the first row (2 in this case) as the "pivot"; we will refer to this equation containing the pivot for the current elimination step as the "pivot equation." We then add $(-(-1/2))$ of the "pivot equation" to the second equation, i.e.

$$\begin{array}{ccccccc} \underset{\text{pivot}}{2} & -1 & 0 & & f_1 & & \frac{1}{2} \text{ eqn } 1 \\ -1 & 2 & -1 & & f_2 & & + 1 \text{ eqn } 2 \\ 0 & -1 & 1 & & f_3 & & \end{array} ,$$

where the system *before* the reduction is shown on the left, and the operation to be applied is shown on the right. The operation eliminates the first coefficient (i.e. the first-column entry, or simply "column 1") of eqn 2, and reduces eqn 2 to

$$0u_1 + \frac{3}{2}u_2 - u_3 = f_2 + \frac{1}{2}f_1 .$$

Since column 1 of eqn 3 is already zero, we need not add the pivot equation to eqn 3. (Systematically, we may interpret this as adding $(-(0/2))$ of the pivot equation to eqn 3.) At this point, we have completed the elimination of the column 1 of eqn 2 through eqn 3 ($= n$) by adding to each appropriately scaled pivot equations. We will refer to this partially reduced system, as "U-to-be"; in particular, we will denote the system that has been reduced up to (and including) the k-th pivot by $\tilde{U}(k)$. Because we have so far processed the first pivot, we have $\tilde{U}(k = 1)$, i.e.

$$\tilde{U}(k = 1) = \begin{pmatrix} 2 & -1 & 0 \\ 0 & \frac{3}{2} & -1 \\ 0 & -1 & 1 \end{pmatrix} .$$

In the second elimination step, we identify the modified second equation (eqn $2'$) as the "pivot equation" and proceed with the elimination of column 2 of eqn $3'$ through eqn n'. (In this case, we modify only eqn $3'$ since there are only three equations.) Here the prime refers to the equations in $\tilde{U}(k=1)$. Using column 2 of the pivot equation as the pivot, we add $(-(-1/(3/2)))$ of the pivot equation to eqn $3'$, i.e.

$$\begin{array}{ccccc} 2 & -1 & 0 & f_1 & \\ 0 & \underset{\text{pivot}}{\frac{3}{2}} & -1 & f_2 + \frac{1}{2}f_1 & \frac{2}{3}\text{ eqn } 2' \\ 0 & -1 & 1 & f_3 & 1 \text{ eqn } 3' \end{array} ,$$

where, again, the system before the reduction is shown on the left, and the operation to be applied is shown on the right. The operation yields a new system,

$$\begin{array}{cccc} 2 & -1 & 0 & f_1 \\ 0 & \frac{3}{2} & -1 & f_2 + \frac{1}{2}f_1 \\ 0 & 0 & \frac{1}{3} & f_3 + \frac{2}{3}f_2 + \frac{1}{3}f_1 \end{array} ,$$

or, equivalently in the matrix form

$$\begin{pmatrix} 2 & -1 & 0 \\ 0 & \frac{3}{2} & -1 \\ 0 & 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 + \frac{1}{2}f_1 \\ f_3 + \frac{2}{3}f_2 + \frac{1}{3}f_1 \end{pmatrix}$$

$$U \qquad u \qquad = \qquad \hat{f},$$

which is an upper triangular system. Note that this second step of Gaussian elimination — which adds an appropriately scaled eqn $2'$ to eliminate column 3 of all the equations below it — can be reinterpreted as performing the first step of Gaussian elimination to the $(n-1) \times (n-1)$ lower sub-block of the matrix (which is 2×2 in this case). This interpretation enables extension of the Gaussian elimination procedure to general $n \times n$ matrices, as we will see shortly.

Having constructed an upper triangular system, we can find the solution using the back substitution procedure. First, solving for the last variable using the last equation (i.e. solving for u_3 using eqn 3),

$$\underset{\text{of } U}{\text{eqn } n(=3)} \qquad \frac{1}{3}u_3 = f_3 + \frac{2}{3}f_2 + \frac{1}{3}f_1 \quad \Rightarrow \quad u_3 = 3f_3 + 2f_2 + f_1 .$$

Now treating u_3 as a "known", we solve for u_2 using the second to last equation (i.e. eqn 2),

$$\underset{\text{of } U}{\text{eqn } 2} \qquad \frac{3}{2}u_2 - \underset{\substack{\text{known;} \\ \text{(move to r.h.s.)}}}{u_3} = f_2 + \frac{1}{2}f_1$$

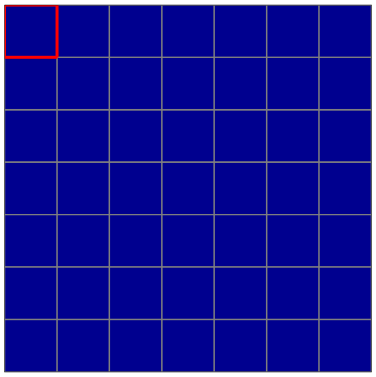
$$\frac{3}{2}u_2 = f_2 + \frac{1}{2}f_1 + u_3 \quad \Rightarrow \quad u_2 = 2f_2 + f_1 + 2f_3 .$$

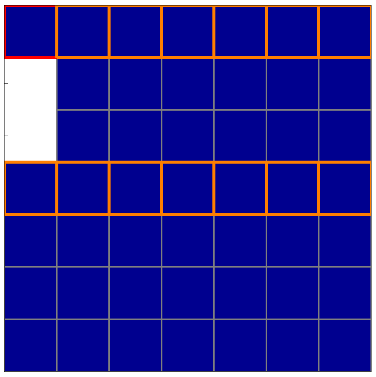
Finally, treating u_3 and u_2 as "knowns", we solve for u_1 using eqn 1,

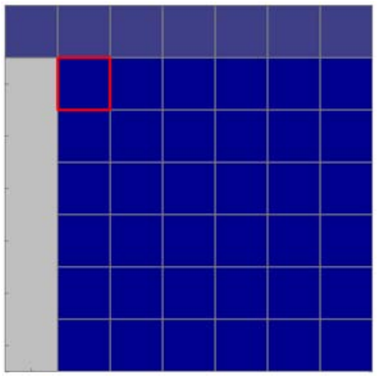
$$\underset{\text{of } U}{\text{eqn } 1} \qquad 2u_1 - \underset{\substack{\text{known;} \\ \text{(move to r.h.s.)}}}{u_2} + \underset{\substack{\text{known;} \\ \text{(move to r.h.s.)}}}{0 \cdot u_3} = f_1$$

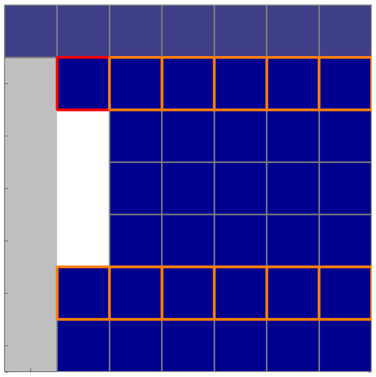
$$2u_1 = f_1 + u_2 \ (+\, 0 \cdot u_3) \quad \Rightarrow \quad u_1 = f_1 + f_2 + f_3 .$$

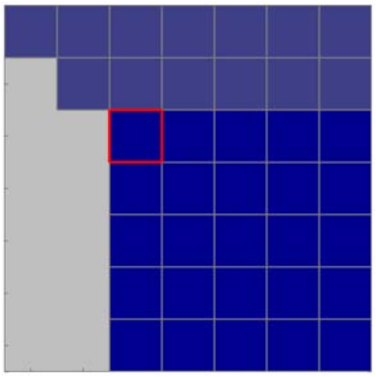
Again, we have taken advantage of the upper triangular structure of the linear system to sequentially solve for unknowns starting from the last equation.


(a) original system $A = \tilde{U}(k=0)$


(b) processing pivot 1


(c) beginning of step 2, $\tilde{U}(k=1)$


(d) processing pivot 2


(e) beginning of step 3, $\tilde{U}(k=2)$

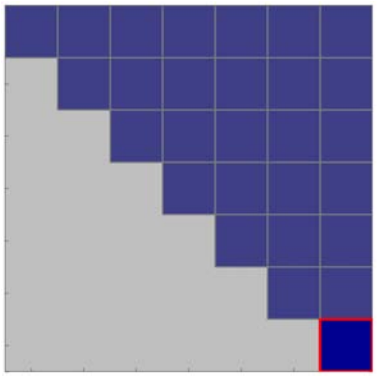

(f) final matrix $U = \tilde{U}(k=n)$

Figure 26.4: Illustration of Gaussian elimination applied to a 6×6 system. See the main text for a description of the colors.

26.3 General $n \times n$ Systems

Now let us consider a general $n \times n$ linear system. We will again use a systematic, two-step approach: Gaussian elimination and back substitution:

$$\text{STEP 1:} \quad \underset{n\times n}{A} \quad \underset{n\times 1}{u} \quad = \quad \underset{n\times 1}{f} \quad \rightarrow \quad \underset{n\times n}{U} \quad \underset{n\times 1}{u} \quad = \quad \underset{n\times 1}{\hat{f}} \qquad \text{(GE)}$$

.

$$\text{STEP 2:} \qquad Uu = \hat{f} \qquad \Rightarrow \quad u \qquad\qquad \text{(BS)}$$

This time, we will pay particular attention to the operation count required for each step. In addition, we will use the graphical representation shown in Figure 26.4 to facilitate the discussion. In the figure, the blue represents (in general) a nonzero entry, the white represents a zero entry, the red square represents the pivot, the orange squares identify the working rows, the shaded regions represents the rows or columns of pivots already processed, and the unshaded regions represents the rows and columns of pivots not yet processed.

As before, the first step of Gaussian elimination identifies the first equation (eqn 1) as the pivot equation and eliminates the first coefficient (column 1) of the eqn 2 through eqn n. To each such row, we add the appropriately scaled (determined by the ratio of the first coefficient of the row and the pivot) pivot row. We must scale (i.e. multiply) and add n coefficients, so the elimination of the first coefficient requires $2n$ operations *per row*. Since there are $n - 1$ rows to work on, the total operation count for the elimination of column 1 of eqn 2 through eqn n is $2n(n-1) \approx 2n^2$. Figure 26.4(b) illustrates the elimination process working on the fourth row. Figure 26.4(c) shows the partially processed matrix with zeros in the first column: U-to-be after the first step, i.e. $\tilde{U}(k=1)$.

In the second step, we identify the second equation as the pivot equation. The elimination of column 2 of eqn 3 through eqn n requires addition of an $(n-1)$-vector — an appropriately scaled version of the pivot row of $\tilde{U}(k=1)$ — from the given row. Since there are $n-2$ rows to work on, the total operation count for the elimination of column 2 of eqn 3 through eqn n is $2(n-1)(n-2) \approx 2(n-1)^2$. Note that the work required for the elimination of the second coefficient in this second step is lower than the work required for the elimination of the first coefficient in the first step because 1) we do not alter the first row (i.e. there is one less row from which to eliminate the coefficient) and 2) the first coefficient of all working rows have already been set to zero. In other word, we are working on the lower $(n-1)\times(n-1)$ sub-block of the original matrix, eliminating the first coefficient of the sub-block. This sub-block interpretation of the elimination process is clear from Figures 26.4(c) and 26.4(d); because the first pivot has already been processed, we only need to work on the unshaded area of the matrix.

In general, on the k^{th} step of Gaussian elimination, we use the k^{th}row to remove the k^{th} coefficient of eqn $k+1$ through eqn n, working on the $(n-k+1)\times(n-k+1)$ sub-block. Thus, the operation count for the step is $2(n-k+1)$. Summing the work required for the first to the n^{th} step, the total operation count for Gaussian elimination is

$$2n^2 + 2(n-1)^2 + \cdots + 2\cdot 3^2 + 2\cdot 2^2 \approx \sum_{k=1}^{n} 2k^2 \approx \frac{2}{3}n^3 \text{ FLOPs} .$$

Note that the cost of Gaussian elimination grows quite rapidly with the size of the problem: as the third power of n. The upper-triangular final matrix, $U = \tilde{U}(k=n)$, is shown in Figure 26.4(f).

During the Gaussian elimination process, we must also construct the modified right-hand side $\hat{f}$. In eliminating the first coefficient, we modify the right-hand side of eqn 2 through eqn n ($n-1$ equations), each requiring two operations for multiplication and addition, resulting in $2(n-1) \approx 2n$ total operations. In general, the k^{th} step of Gaussian elimination requires modification of the $(n-k)$-sub-vector on the right-hand side. Thus, the total operation count for the construction of the right-hand side is

$$2n + 2(n-1) + \cdots + 2\cdot 3 + 2\cdot 2 \approx \sum_{k=1}^{n} 2k \approx n^2 \text{ FLOPs} .$$

As the cost for constructing the modified right-hand side scales as n^2, it becomes insignificant compared to $2n^3/3$ operations required for the matrix manipulation for a large n. Thus, we conclude that the total cost of Gaussian elimination, including the construction of the modified right-hand side, is $2n^3/3$.

Now let us consider the operation count of back substitution. Recall that the $n \times n$ upper triangular system takes the form

$$\begin{pmatrix} U_{11} & U_{12} & \cdots & \cdots & U_{1n} \\ & U_{22} & & & U_{2n} \\ & & \ddots & & \vdots \\ & 0 & & U_{n-1\,n-1} & U_{n-1\,n} \\ & & & & U_{n\,n} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_{n-1} \\ \hat{f}_n \end{pmatrix} .$$

We proceed to solve for the unknowns $u_1, \ldots, u_n$ starting from the last unknown u_n using the n^{th} equation and sequentially solving for $u_{n-1}, \ldots, u_1$ in that order. Schematically, the solution process takes the form

eqn n: $$U_{n\,n} u_n - \hat{f}_n \;\Rightarrow\; u_n = \frac{\hat{f}_n}{U_{n\,n}}$$

eqn $n-1$: $$U_{n-1\,n-1} u_{n-1} + U_{n-1\,n} u_n = \hat{f}_{n-1}$$
$$\Downarrow$$
$$U_{n-1\,n-1} u_{n-1} = \hat{f}_{n-1} - U_{n-1\,n-1} u_{n-1} \;\Rightarrow\; u_{n-1}$$

$\vdots$

eqn 1: $$U_{11} u_1 + U_{12} u_2 + \cdots + U_{1\,n} u_n = \hat{f}_1$$
$$\Downarrow$$
$$U_{11} u_1 = \hat{f}_1 - U_{12} u_2 - \cdots - U_{1\,n} u_n \;\Rightarrow\; u_1 .$$

Solving for u_n requires one operation. Solving for u_{n-1} requires one multiplication-subtraction pair (two operations) and one division. In general, solving for u_k requires $(n-k)$ multiplication-subtraction pairs ($2(n-k)$ operations) and one division. Summing all the operations, the total operation count for back substitution is

$$1 + (1+2) + (1 + 2 \cdot 2) + \cdots + (1 + 2(n-1)) \approx \sum_{k=1}^{N} 2k \approx n^2 \,\text{FLOPs} .$$

Note that the cost for the back substitution step scales as the second power of the problem size n; thus, the cost of back substitution becomes negligible compared to that of Gaussian elimination for a large n.

26.4 Gaussian Elimination and LU Factorization

In this chapter, we introduced a systematic procedure for solving a linear system using Gaussian elimination and back substitution. We interpreted Gaussian elimination as a process of triangulating the system matrix of interest; the process relied, in the k^{th} step, on adding appropriately scaled versions of the k^{th} equation to all the equations below it in order to eliminate the leading coefficient. In particular, we also modified the right-hand side of the equation in the triangulation procedure such that we are adding the same quantity to both sides of the equation and hence not affecting the solution. The end product of our triangulation process is an upper triangular matrix U and a modified right-hand side $\hat{f}$. If we are given a new right-hand side, we would have to repeat the same procedure again (in $\mathcal{O}(n^3)$ cost) to deduce the appropriate new modified right-hand side.

It turns out that a slight modification of our Gaussian elimination procedure in fact would permit solution to the problem with a different right-hand side in $\mathcal{O}(n^2)$ operations. To achieve this, instead of modifying the right-hand side in the upper triangulation process, we *record the operations used in the upper triangulation process with which we generated the right-hand side.* It turns out that this recording operation in fact can be done using a lower triangular matrix L, such that the modified right-hand side $\hat{f}$ is the solution to

$$L\hat{f} = f, \tag{26.3}$$

where f is the original right-hand side. Similar to back substitution for an upper triangular system, *forward substitution* enables solution to the lower triangular system in $\mathcal{O}(n^2)$ operations. This lower triangular matrix L that records all operations used in transforming matrix A into U in fact is a matrix that satisfies

$$A = LU \ .$$

In other words, the matrices L and U arise from a *factorization* of the matrix A into lower and upper triangular matrices.

This procedure is called *LU factorization*. (The fact that L and U must permit such a factorization is straightforward to see from the fact that $Uu = \hat{f}$ and $L\hat{f} = f$; multiplication of both sides of $Uu = \hat{f}$ by L yields $LUu = L\hat{f} = f$, and because the relationship must hold for any solution-right-hand-side pair $\{u, f\}$ to $Au = f$, it must be that $LU = A$.) The factorization process is in fact identical to our Gaussian elimination and requires $2n^3/3$ operations. Note we did compute all the pieces of the matrix L in our elimination procedure; we simply did not form the matrix for simplicity.

In general the LU decomposition will exist if the matrix A is non-singular. There is, however, one twist: we may need to permute the rows of A — a process known as (partial) pivoting — in order to avoid a zero pivot which would prematurely terminate the process. (In fact, permutations of rows can also be advantageous to avoid small pivots which can lead to amplification of round-off errors.) If even — say in infinite precision — with row permutations we arrive at an exactly zero pivot then this is in fact demonstrates that A is singular.[1]

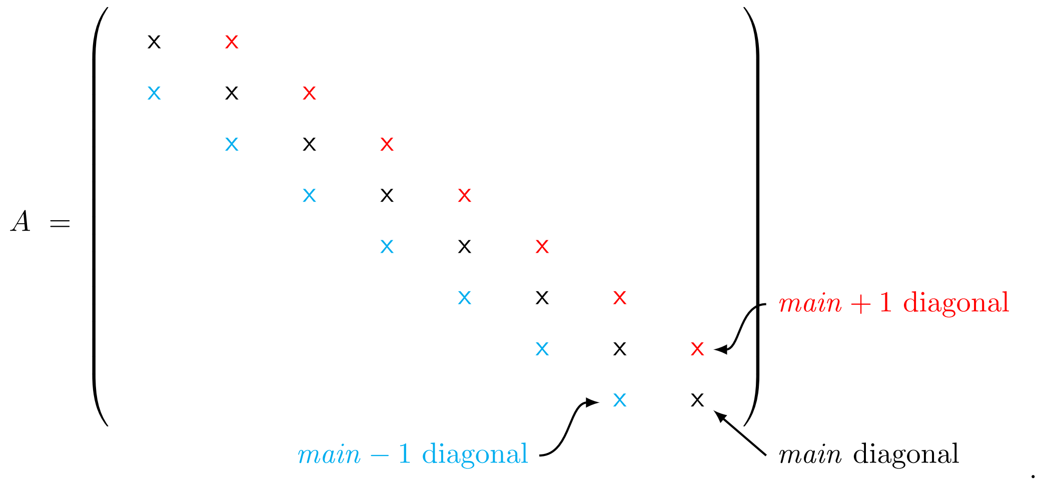
There are some matrices for which no pivoting is required. One such important example in mechanical engineering is SPD matrices. For an SPD matrix (which is certainly non-singular — all eigenvalues are positive) we will never arrive at a zero pivot nor we will need to permute rows to ensure stability. Note, however, that we may still wish to permute rows to improve the efficiency of the LU decomposition for sparse systems — which is the topic of the next section.

26.5 Tridiagonal Systems

While the cost of Gaussian elimination scales as n^3 for a general $n \times n$ linear system, there are instances in which the scaling is much weaker and hence the computational cost for a large problem

[1]The latter is a more practical test for singularity of A than say the determinant of A or the eigenvalues of A, however typically singularity of "mechanical engineering" A matrices are not due to devious cancellations but rather due to upfront modeling errors — which are best noted and corrected prior to LU decomposition.

is relatively low. A *tridiagonal system* is one such example. A tridigonal system is characterized by having nonzero entries only along the main diagonal and the immediate upper and lower diagonal, i.e.



The immediate upper diagonal is called the *super-diagonal* and the immediate lower diagonal is called the *sub-diagonal*. A significant reduction in the computational cost is achieved by taking advantage of the *sparsity* of the tridiagonal matrix. That is, we omit addition and multiplication of a large number of zeros present in the matrix.

Let us apply Gaussian elimination to the $n \times n$ tridiagonal matrix. In the first step, we compute the scaling factor (one FLOP), scale the second entry of the coefficient of the first row by the scaling factor (one FLOP), and add that to the second coefficient of the second row (one FLOP). (Note that we do not need to scale and add the first coefficient of the first equation to that of the second equation because we know it will vanish by construction.) We do not have to add the first equation to any other equations because the first coefficient of all other equations are zero. Moreover, note that the addition of the (scaled) first row to the second row does not introduce any new nonzero entry in the second row. Thus, the updated matrix has zeros above the super-diagonal and *retains the tridiagonal structure of the original matrix* (with the (2,1) entry eliminated, of course); in particular, the updated $(n-1) \times (n-1)$ sub-block is again tridiagonal. We also modify the right-hand side by multiplying the first entry by the scaling factor (one FLOP) and adding it to the second entry (one FLOP). Combined with the three FLOPs required for the matrix manipulation, the total cost for the first step is five FLOPs.

Similarly, in the second step, we use the second equation to eliminate the leading nonzero coefficient of the third equation. Because the structure of the problem is identical to the first one, this also requires five FLOPs. The updated matrix retain the tridiagonal structure in this elimination step and, in particular, the updated $(n-2) \times (n-2)$ sub-block is tridiagonal. Repeating the operation for n steps, the total cost for producing an upper triangular system (and the associated modified right-hand side) is $5n$ FLOPs. Note that the cost of Gaussian elimination for a tridiagonal system scales *linearly* with the problem size n: a dramatic improvement compared to $\mathcal{O}(n^3)$ operations required for a general case.

At this point, we have produced an upper triangular system of the form

$$\begin{pmatrix} \times & \times & & & & & & \\ & \times & \times & & & 0 & & \\ & & \times & \times & & & & \\ & & & \times & \times & & & \\ & & & & \times & \times & & \\ & 0 & & & & \times & \times & \\ & & & & & & \times & \times \\ & & & & & & & \times \end{pmatrix} \begin{pmatrix} \underset{\text{3 FLOPs}}{u_1} \\ u_2 \\ \vdots \\ \underset{\text{3 FLOPs}}{u_{n-2}} \\ \underset{\text{3 FLOPs}}{u_{n-1}} \\ \underset{\text{1 FLOP}}{u_n} \end{pmatrix} = \begin{pmatrix} \hat{f}_1 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{n-2} \\ \hat{f}_{n-1} \\ \hat{f}_n \end{pmatrix} .$$

The system is said to be *bidiagonal* — or more precisely upper bidiagonal — as nonzero entries appear only on the main diagonal and the super-diagonal. (A matrix that has nonzero entries only on the main and sub-diagonal are also said to be bidiagonal; in this case, it would be lower bidiagonal.)

In the back substitution stage, we can again take advantage of the sparsity — in particular the bidiagonal structure — of our upper triangular system. As before, evaluation of u_n requires a simple division (one FLOP). The evaluation of u_{n-1} requires one scaled subtraction of u_n from the right-hand side (two FLOPs) and one division (one FLOP) for three total FLOPs. The structure is the same for the remaining $n-2$ unknowns; the evaluating each entry takes three FLOPs. Thus, the total cost of back substitution for a bidiagonal matrix is $3n$ FLOPs. Combined with the cost of the Gaussian elimination for the tridiagonal matrix, the overall cost for solving a tridiagonal system is $8n$ FLOPs. Thus, the operation count of the entire linear solution procedure (Gaussian elimination and back substitution) scales linearly with the problem size for tridiagonal matrices.

We have achieved a significant reduction in computational cost for a tridiagonal system compared to a general case by taking advantage of the sparsity structure. In particular, the computational cost has been reduced from $2n^3/3$ to $8n$. For example, if we wish to solve for the equilibrium displacement of a $n = 1000$ spring-mass system (which yields a tridiagonal system), we have reduced the number of operations from an order of a billion to a thousand. In fact, with the tridiagonal-matrix algorithm that takes advantage of the sparsity pattern, we can easily solve a spring-mass system with millions of unknowns on a desktop machine; this would certainly not be the case if the general Gaussian elimination algorithm is employed, which would require $\mathcal{O}(10^{18})$ operations.

While many problems in engineering require solution of a linear system with millions (or even billions) of unknowns, these systems are typically sparse. (While these systems are rarely tridiagonal, most of the entries of these matrices are zero nevertheless.) In the next chapter, we consider solution to more general sparse linear systems; just as we observed in this tridiagonal case, the key to reducing the computational cost for large sparse matrices — and hence making the computation tractable — is to study the nonzero pattern of the sparse matrix and design an algorithm that does not execute unnecessary operations.

Chapter 27

Gaussian Elimination: Sparse Matrices

In the previous chapter, we observed that the number of floating point operations required to solve a $n \times n$ tridiagonal system scales as $\mathcal{O}(n)$ whereas that for a general (dense) $n \times n$ system scales as $\mathcal{O}(n^3)$. We achieved this significant reduction in operation count by taking advantage of the sparsity of the matrix. In this chapter, we will consider solution of more general sparse linear systems.

27.1 Banded Matrices

A class of sparse matrices that often arise in engineering practice — especially in continuum mechanics — is the banded matrix. An example of banded matrix is shown in Figure 27.1. As the figure shows, the nonzero entries of a banded matrix is confined to within m_b entries of the main diagonal. More precisely,

$$A_{ij} = 0, \quad \text{for} \quad j > i + m_\mathrm{b} \quad \text{or} \quad j < i - m_\mathrm{b},$$

and A may take on any value within the band (including zero). The variable m_b is referred to as the *bandwidth*. Note that the number of nonzero entries in a $n \times n$ banded matrix with a bandwidth m_b is less than $n(2m_\mathrm{b} + 1)$.

Let us consider a few different types of banded matrices.

$$\underset{n\times n}{A} \; \underset{n\times 1}{u} = \underset{n\times 1}{f}, \quad A =$$

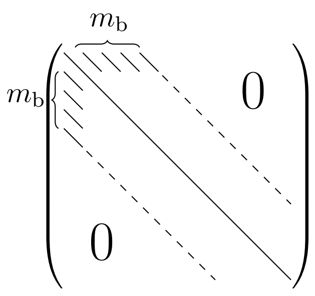


Figure 27.1: A banded matrix with bandwidth m_b.

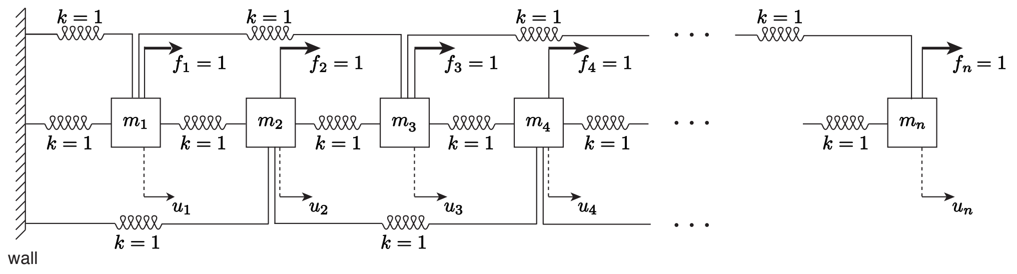
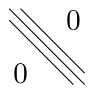


Figure 27.2: A spring-mass system whose equilibrium state calculation gives rise to a pentadiagonal matrix.

Example 27.1.1 *Tri*diagonal matrix: $m_\text{b} = 1$

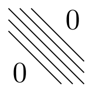
As we have discussed in the previous two chapters, tridiagonal matrices have nonzero entries only along the main diagonal, sub-diagonal, and super-diagonal. Pictorially, a tridiagonal matrix takes the following form:

main diagonal, main ± 1 diagonals  .

Clearly the bandwidth of a tridiagonal matrix is $m_\text{b} = 1$. A $n \times n$ tridiagonal matrix arise from, for example, computing the equilibrium displacement of n masses connected by springs, as we have seen in previous chapters.

Example 27.1.2 *Penta*diagonal matrix: $m_\text{b} = 2$

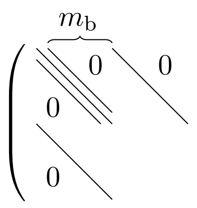
As the name suggests, a pentadiagonal matrix is characterized by having nonzero entries along the main diagonal and the two diagonals above and below it, for the total of five diagonals. Pictorially, a pentadigonal matrix takes the following form:

main diagonal, main ± 1, ± 2 diagonals  .

The bandwidth of a pentadiagonal matrix is $m_\text{b} = 2$. A $n \times n$ pentadiagonal matrix arise from, for example, computing the equilibrium displacement of n masses each of which is connected to not only the immediate neighbor but also to the neighbor of the neighbors. An example of such a system is shown in Figure 27.2.

Example 27.1.3 "Outrigger" matrix

Another important type of banded matrix is a matrix whose zero entries are confined to within the m_b band of the main diagonal but for which a large number of entries between the main diagonal and the most outer band is zero. We will refer to such a matrix as "outrigger." An example of such a matrix is

"outrigger"  .

In this example, there are five nonzero diagonal bands, but the two outer bands are located far from the middle three bands. The bandwidth of the matrix, m_b, is specified by the location of the outer diagonals. (Note that this is *not* a pentadiagonal matrix since the nonzero entries are not confined to within $m_\text{b} = 2$.) "Outrigger" matrices often arise from finite difference (or finite element) discretization of partial differential equations in two or higher dimensions.

27.2 Matrix-Vector Multiplications

To introduce the concept of sparse operations, let us first consider multiplication of a $n \times n$ sparse matrix with a (dense) n-vector. Recall that matrix-vector multiplication may be interpreted row-wise or column-wise. In row-wise interpretation, we consider the task of computing $w = Av$ as performing n inner products, one for each entry of w, i.e.

$$w_i = \begin{pmatrix} A_{i1} & A_{i2} & \dots & A_{in} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}, \quad i = 1, \dots, n.$$

If the matrix A is dense, the n inner products of n-vectors requires $n \cdot (2n) = 2n^2$ FLOPs. However, if the matrix A is sparse, then each row of A contains few nonzero entries; thus, we may skip a large number of trivial multiplications in our inner products. In particular, the operation count for the inner product of a sparse n-vector with a dense n-vector is equal to twice the number of nonzero entries in the sparse vector. Thus, the operation count for the entire matrix-vector multiplication is equal to twice the number of nonzero entries in the matrix, i.e. $2 \cdot \text{nnz}(A)$, where $\text{nnz}(A)$ is the number of nonzero entries in A. This agrees with our intuition because the matrix-vector product requires simply visiting each nonzero entry of A, identifying the appropriate multiplier in v based on the column index, and adding the product to the appropriate entry of w based on the row index.

Now let us consider a column interpretation of matrix-vector multiplication. In this case, we interpret $w = Av$ as

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = v_1 \begin{pmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{n1} \end{pmatrix} + v_2 \begin{pmatrix} A_{12} \\ A_{22} \\ \vdots \\ A_{n2} \end{pmatrix} + \cdots + v_n \begin{pmatrix} A_{1n} \\ A_{2n} \\ \vdots \\ A_{nn} \end{pmatrix}.$$

If A is sparse then, each column of A contains few nonzero entries. Thus, for each column we simply need to scale these few nonzero entries by the appropriate entry of v and augment the corresponding entries of w; the operation count is twice the number of nonzero entries in the column. Repeating the operation for all columns of A, the operation count for the entire matrix-vector multiplication is again $2 \cdot \text{nnz}(A)$.

Because the number of nonzero entries in a sparse matrix is (by definition) $\mathcal{O}(n)$, the operation count for sparse matrix-vector product is $2 \cdot \text{nnz}(A) \sim \mathcal{O}(n)$. For example, for a banded matrix with a bandwidth m_b, the operation count is at most $2n(2m_\text{b} + 1)$. Thus, we achieve a significant reduction in the operation count compared to dense matrix-vector multiplication, which requires $2n^2$ operations.

27.3 Gaussian Elimination and Back Substitution

27.3.1 Gaussian Elimination

We now consider the operation count associated with solving a sparse linear system $Au = f$ using Gaussian elimination and back substitution introduced in the previous chapter. Recall that the Gaussian elimination is a process of turning a linear system into an upper triangular system, i.e.

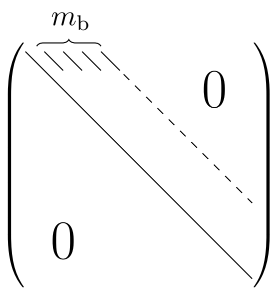
$$\text{STEP 1: } Au = f \rightarrow \underset{\substack{(n\times n)\\ \text{upper}\\ \text{triangular}}}{U} \quad u = \hat{f} \quad .$$

For a $n \times n$ dense matrix, Gaussian elimination requires approximately $\frac{2}{3}n^3$ FLOPs.

Densely-Populated Banded Systems

Now, let us consider a $n \times n$ banded matrix with a bandwidth m_b. To analyze the worst case, we assume that all entries within the band are nonzero. In the first step of Gaussian elimination, we identify the first row as the "pivot row" and eliminate the first entry (column 1) of the first m_b rows by adding appropriately scaled pivot row; column 1 of rows $m_\mathrm{b} + 2, \ldots, n$ are already zero. Elimination of column 1 of a given row requires addition of scaled $m_\mathrm{b} + 1$ entries of the pivot row, which requires $2(m_\mathrm{b} + 1)$ operations. Applying the operation to m_b rows, the operation count for the first step is approximately $2(m_\mathrm{b} + 1)^2$. Note that because the nonzero entries of the pivot row is confined to the first $m_\mathrm{b} + 1$ entries, addition of the scaled pivot row to the first $m_\mathrm{b} + 1$ rows does not increase the bandwidth of the system (since these rows already have nonzero entries in these columns). In particular, the sparsity pattern of the upper part of A is unaltered in the process.

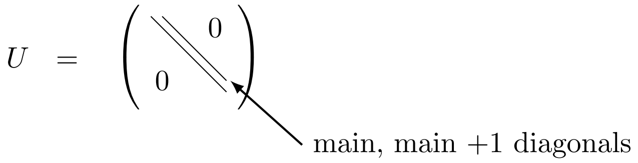
The second step of Gaussian elimination may be interpreted as applying the first step of Gaussian elimination to $(n - 1) \times (n - 1)$ submatrix, which itself is a banded matrix with a bandwidth m_b (as the first step does not alter the bandwidth of the matrix). Thus, the second elimination step also requires approximately $2(m_\mathrm{b} + 1)^2$ FLOPs. Repeating the operation for all n pivots of the matrix, the total operation count for Gaussian elimination is approximately $2n(m_\mathrm{b} + 1)^2 \sim \mathcal{O}(n)$. The final upper triangular matrix U takes the following form:



.

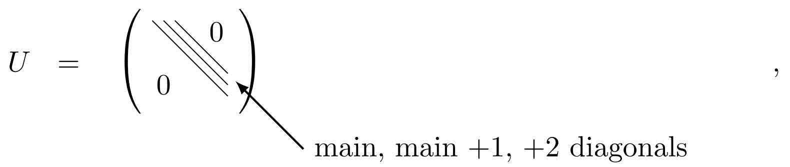
The upper triangular matrix has approximately $n(m_\mathrm{b} + 1) \sim \mathcal{O}(n)$ nonzero entries. Both the operation count and the number of nonzero in the final upper triangular matrix are $\mathcal{O}(n)$, compared to $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ entries for a dense system. (We assume here m_b is fixed independent of n.)

In particular, as discussed in the previous chapter, Gaussian elimination of a tridiagonal matrix yields an upper bidiagonal matrix



,

in approximately $5n$ operations (including the formation of the modified right-hand side $\hat{f}$). Similarly, Gaussian elimination of a pentadiagonal system results in an upper triangular matrix of the form



and requires approximately $14n$ operations.

"Outrigger" Systems: Fill-Ins

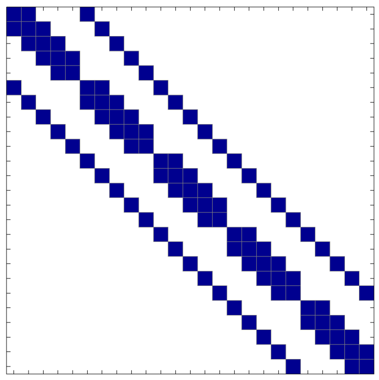
Now let us consider application of Gaussian elimination to an "outrigger" system. First, because a $n \times n$ "outrigger" system with a bandwidth m_b is a special case of a "densely-populated banded" system with a bandwidth m_b considered above, we know that the operation count for Gaussian elimination is at most $n(m_\mathrm{b}+1)^2$ and the number of nonzero in the upper triangular matrix is at most $n(m_\mathrm{b}+1)$. In addition, due to a large number of zero entries between the outer bands of the matrix, we *hope* that the operation count and the number of nonzero are less than those for the "densely-populated banded" case. Unfortunately, inspection of the Gaussian elimination procedure reveals that this reduction in the cost and storage is not achieved in general.

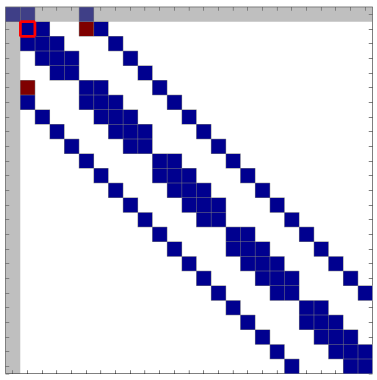
The inability to reduce the operation count is due to the introduction of "fill-ins": the entries of the sparse matrix that are originally zero but becomes nonzero in the Gaussian elimination process. The introduction of fill-ins is best described graphically. Figure 27.3 shows a sequence of matrices generated through Gaussian elimination of a 25×25 outrigger system. In the subsequent figures, we color code entries of partially processed U: the shaded area represents rows *or* columns of pivots already processed; the unshaded area represents the rows *and* columns of pivots not yet processed; the blue represent initial nonzeros in A which remain nonzeros in U-to-be; and the red are initial zeros of A which become nonzero in U-to-be, i.e. fill-ins.

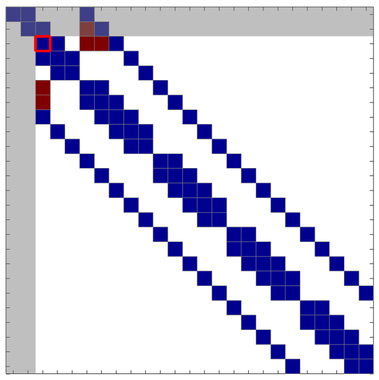
As Figure 27.3(a) shows, the bandwidth of the original matrix is $m_\mathrm{b} = 5$. (The values of the entries are hidden in the figure as they are not important in this discussion of fill-ins.) In the first elimination step, we first eliminate column 1 of row 2 by adding an appropriately scaled row 1 to the row. While we succeed in eliminating column 1 of row 2, note that we introduce a nonzero element in column 6 of row 2 as column 6 of row 1 "falls" to row 2 in the elimination process. This nonzero element is called a "fill-in." Similarly, in eliminating column 1 of row 6, we introduce a "fill-in" in column 2 as column 2 of row 1 "falls" to row 6 in the elimination process. Thus, we have introduced two fill-ins in this first elimination step as shown in Figure 27.3(b): one in the upper part and another in the lower part.

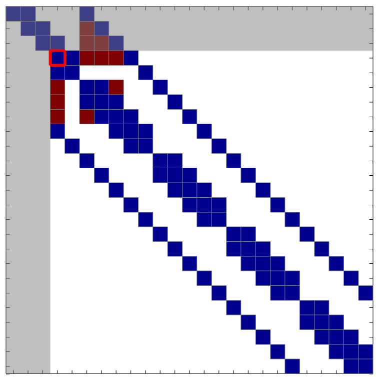
Now, consider the second step of elimination starting from Figure 27.3(b). We first eliminate column 2 of row 3 by adding appropriately scaled row 2 to row 3. This time, we introduce fill in not only from column 7 of row 2 "falling" to row 3, but also from column 6 of row 2 "falling" to row 3. Note that the latter is in fact a fill-in introduced in the first step. In general, once a fill-in is introduced in the upper part, *the fill-in propagates from one step to the next, introducing further fill-ins* as it "falls" through. Next, we need to eliminate column 2 of row 6; this entry was zero in the original matrix but was filled in the first elimination step. Thus, fill-in introduced in the lower part *increases the number of rows whose leading entry must be eliminated in the upper-triangulation process*. The matrix after the second step is shown in Figure 27.3(c). Note that the number of fill-in continue to increase, and we begin to lose the zero entries between the outer bands of the outrigger system.

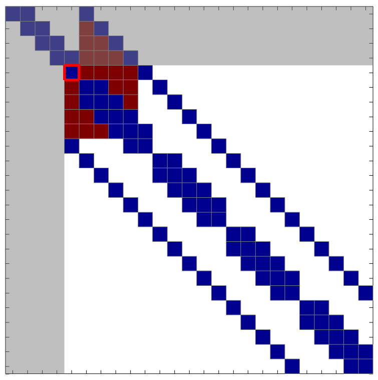
As shown in Figure 27.3(e), by the beginning of the fifth elimination step, the "outrigger"


(a) original system A


(b) beginning of step 2, $\tilde{U}(k=1)$


(c) beginning of step 3, $\tilde{U}(k=2)$


(d) beginning of step 4, $\tilde{U}(k=3)$


(e) beginning of step 5, $\tilde{U}(k=4)$

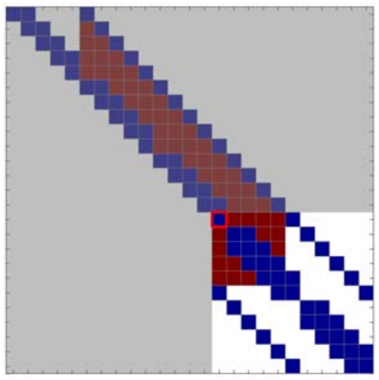
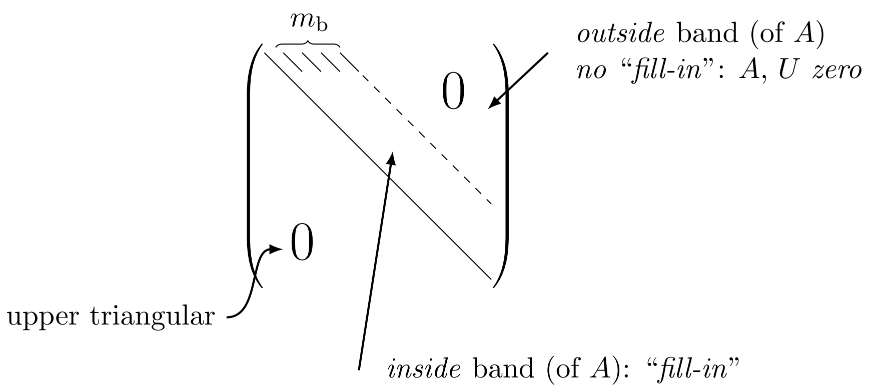

(f) beginning of step 15, $\tilde{U}(k=14)$

Figure 27.3: Illustration of Gaussian elimination applied to a 25×25 "outrigger" system. The blue entries are the entries present in the original system, and the red entries are "fill-in" introduced in the factorization process. The pivot for each step is marked by a red square.

system has largely lost its sparse structure in the leading $(m_\mathrm{b} + 1) \times (m_\mathrm{b} + 1)$ subblock of the working submatrix. Thus, for the subsequent $n - m_\mathrm{b}$ steps of Gaussian elimination, each step takes $2m_\mathrm{b}^2$ FLOPs, which is approximately the same number of operations as the densely-populated banded case. Thus, the total number of operations required for Gaussian elimination of an outrigger system is approximately $2n(m_\mathrm{b} + 1)^2$, the same as the densely-populated banded case. The final matrix takes the form:


m_b
outside band (of A)
no "fill-in": A, U *zero*
0
0
upper triangular
inside band (of A): "*fill-in*"

Note that the number of nonzero entries is approximately $n(m_\mathrm{b} + 1)$, which is much larger than the number of nonzero entries in the original "outrigger" system.

The "outrigger" system, such as the one considered above, naturally arise when a partial differential equation (PDE) is discretized in two or higher dimensions using a finite difference or finite element formulation. An example of such a PDE is the heat equation, describing, for example, the equilibrium temperature of a thermal system shown in Figure 27.4. With a natural ordering of the degrees of freedom of the discretized system, the bandwidth m_b is equal to the number of grid points in one coordinate direction, and the number of degrees of freedom of the linear system is $n = m_\mathrm{b}^2$ (i.e. product of the number of grid points in two coordinate directions). In other words, the bandwidth is the square root of the matrix size, i.e. $m_\mathrm{b} = n^{1/2}$. Due to the outrigger structure of the resulting system, factorizing the system requires approximately $n(m_\mathrm{b} + 1)^2 \approx n^2$ FLOPs. This is in contrast to one-dimensional case, which yields a tridiagonal system, which can be solved in $\mathcal{O}(n)$ operations. In fact, in three dimensions, the bandwidth is equal to the product of the number of grid points in two coordinate directions, i.e. $m_\mathrm{b} = (n^{1/3})^2 = n^{2/3}$. The number of operations required for factorization is $n(m_\mathrm{b} + 1)^2 \approx n^{7/3}$. Thus, the cost of solving a PDE is significantly higher in three dimensions than in one dimension *even if both discretized systems had the same number of unknowns.*[1]

27.3.2 Back Substitution

Having analyzed the operation count for Gaussian elimination, let us inspect the operation count for back substitution. First, recall that back substitution is a process of finding the solution of an upper triangular system, i.e.

$$\text{STEP 2: } Uu = \hat{f} \to u \quad .$$

Furthermore, recall that the operation count for back substitution is equal to twice the number of nonzero entries in U. Because the matrix U is unaltered, we can simply count the number of

[1]Not unknowns per dimension, but the total number of unknowns.

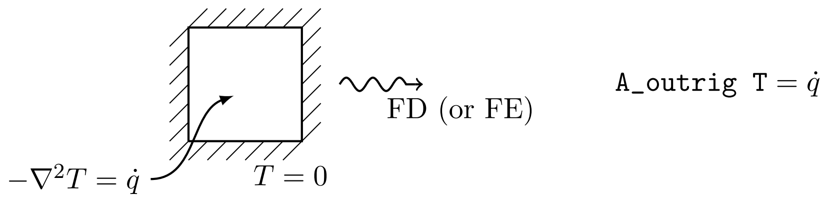


Figure 27.4: Heat equation in two dimensions. Discretization of the equation by finite difference (FD) or finite element (FE) method yields an "outrigger" system.

nonzero entries in the U that we obtain after Gaussian elimination; there is nothing equivalent to "fill-in" — modifications to the matrix that increases the number of entries in the matrix and hence the operation count — in back substitution.

Densely-Populated Banded Systems

For a densely-populated banded system with a bandwidth m_b, the number of unknowns in the factorized matrix U is approximately equal to $n(m_\mathrm{b}+1)$. Thus, back substitution requires approximately $2n(m_\mathrm{b}+1)$ FLOPs. In particular, back substitution for a tridiagonal system (which yields an upper bidiagonal U) requires approximately $3n$ FLOPs. A pentadiagonal system requires approximately $5n$ FLOPs.

"Outrigger"

As discussed above, a $n \times n$ outrigger matrix of bandwidth m_b produces an upper triangular matrix U whose entries between the main diagonal and the outer band are nonzero due to fill-ins. Thus, the number of nonzeros in U is approximately $n(m_\mathrm{b}+1)$, and the operation count for back substitution is approximately $2n(m_\mathrm{b}+1)$. (Note in particular that even if an outrigger system only have five bands (as in the one shown in Figure 27.3), the number of operations for back substitution is $2n(m_\mathrm{b}+1)$ and *not* $5n$.)

Begin Advanced Material

27.4 Fill-in and Reordering

The previous section focused on the computational cost of solving a linear system governed by banded sparse matrices. This section introduces a few additional sparse matrices and also discussed additional concepts on Gaussian elimination for sparse systems.

27.4.1 A Cyclic System

First, let us show that a small change in a physical system — and hence the corresponding linear system A — can make a large difference in the sparsity pattern of the factored matrix U. Here, we consider a modified version of n-mass mass-spring system, where the first mass is connected to the last mass, as shown in Figure 27.5. We will refer to this system as a "cyclic" system, as the springs form a circle. Recall that a spring-mass system without the extra connection yields a tridiagonal system. With the extra connection between the first and the last mass, now the $(1, n)$ entry and $(n, 1)$ entry of the matrix are nonzero as shown in Figure 27.6(a) (for $n = 25$); clearly, the matrix

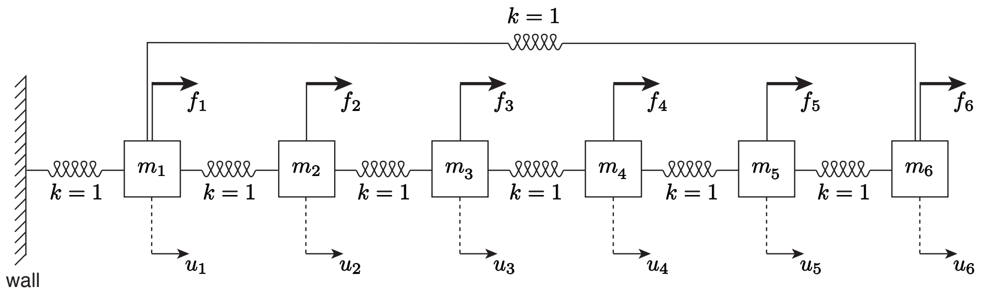
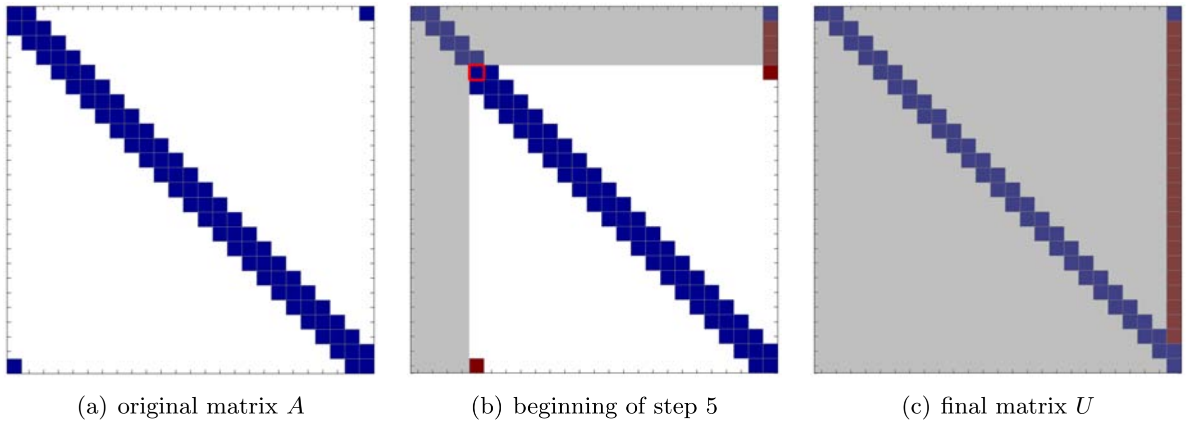


Figure 27.5: "Cyclic" spring-mass system with $n = 6$ masses.



(a) original matrix A (b) beginning of step 5 (c) final matrix U

Figure 27.6: Illustration of Gaussian elimination applied to a 25×25 "arrow" system. The red entries are "fill-in" introduced in the factorization process. The pivot for each step is marked by a red square.

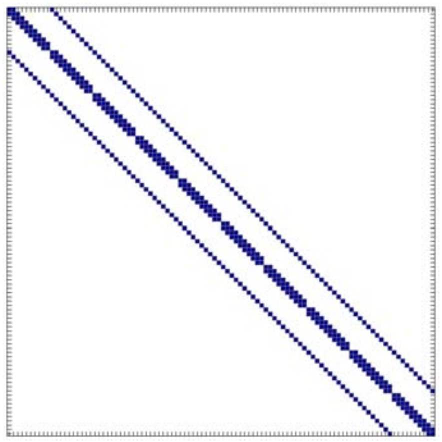
is no longer tridiagonal. In fact, if apply our standard classification for banded matrices, the cyclic matrix would be characterized by its bandwidth of $m_\mathrm{b} = n - 1$.

Applying Gaussian elimination to the "cyclic" system, we immediately recognize that the $(1, n)$ entry of the original matrix "falls" along the last column, creating $n - 2$ fill-ins (see Figures 27.6(b) and 27.6(c)). In addition, the original $(n, 1)$ entry also creates a nonzero entry on the bottom row, which moves across the matrix with the pivot as the matrix is factorized. As a result, the operation count for the factorization of the "cyclic" system is in fact similar to that of a pentadiagonal system: approximately $14n$ FLOPs. Applying back substitution to the factored matrix — which contains approximately $3n$ nonzeros — require $5n$ FLOPs. Thus, solution of the cyclic system — which has just two more nonzero entries than the tridiagonal system — requires more than twice the operations ($19n$ vs. $8n$). However, it is also important to note that this $\mathcal{O}(n)$ operation count is a significant improvement compared to the $\mathcal{O}(n(m_\mathrm{b} + 1)^2) = \mathcal{O}(n^3)$ operation estimate based on classifying the system as a standard "outrigger" with a bandwidth $m_\mathrm{b} = n - 1$.

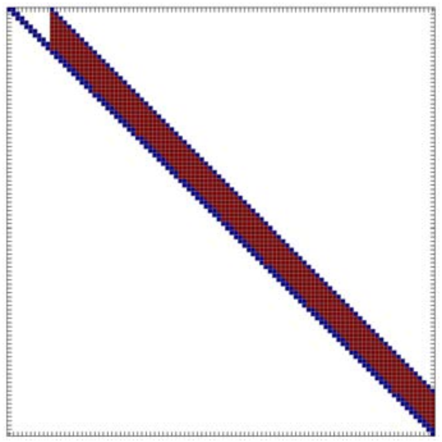
We note that the fill-in structure of U takes the form of a skyline defined by the envelope of the columns of the original matrix A. This is a general principal.

27.4.2 Reordering

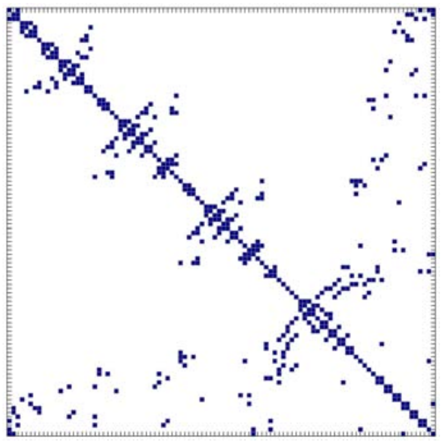
In constructing a linear system corresponding to our spring-mass system, we associated the j^th entry of the solution vector — and hence the j^th column of the matrix — with the displacement of the j^th mass (counting from the wall) and associated the i^th equation with the force equilibrium condition



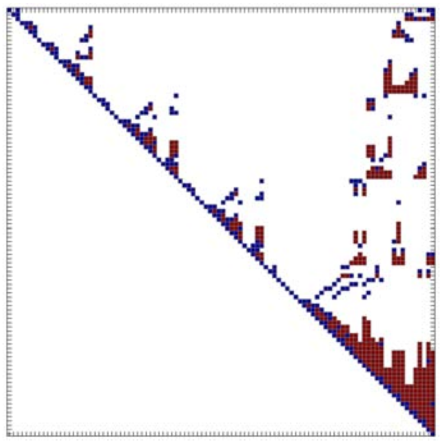
(a) A (natural, $\text{nnz}(A) = 460$)



(b) U (natural, $\text{nnz}(U) = 1009$)



(c) A' (AMD, $\text{nnz}(A') = 460$)



(d) U' (AMD, $\text{nnz}(U') = 657$)

Figure 27.7: Comparison of the sparsity pattern and Gaussian elimination fill-ins for a $n = 100$ "outrigger" system resulting from natural ordering and an equivalent system using the approximate minimum degree (AMD) ordering.

of the i^{th} mass. While this is arguably the most "natural" ordering for the spring-mass system, we could have associated a given column and row of the matrix with a different displacement and force equilibrium condition, respectively. Note that this "reordering" of the unknowns and equations of the linear system is equivalent to "swapping" the rows of columns of the matrix, which is formally known as permutation. Importantly, we can describe the same physical system using many different orderings of the unknowns and equations; even if the matrices appear different, these matrices describing the same physical system may be considered equivalent, as they all produce the same solution for a given right-hand side (given both the solution and right-hand side are reordered in a consistent manner).

Reordering can make a significant difference in the number of fill-ins and the operation count. Figure 27.7 shows a comparison of number of fill-ins for an $n = 100$ linear system arising from two different orderings of a finite different discretization of two-dimensional heat equation on a 10×10 computational grid. An "outrigger" matrix of bandwidth $m_{\text{b}} = 10$ arising from "natural" ordering is shown in Figure 27.7(a). The matrix has 460 nonzero entries. As discussed in the previous section, Gaussian elimination of the matrix yields an upper triangular matrix U with approximately

$n(m_\mathrm{b}+1) = 1100$ nonzero entries (more precisely 1009 for this particular case), which is shown in Figure 27.7(b). An equivalent system obtained using the approximate minimum degree (AMD) ordering is shown in Figure 27.7(c). This newly reordered matrix A' also has 460 nonzero entries because permuting (or swapping) rows and columns clearly does not change the number of nonzeros. On the other hand, application of Gaussian elimination to this reordered matrix yields an upper triangular matrix U shown in Figure 27.7(d), which has only 657 nonzero entries. Note that the number of fill-in has been reduced by roughly a factor of two: from $1009 - 280 = 729$ for the "natural" ordering to $657 - 280 = 377$ for the AMD ordering. (The original matrix A has 280 nonzero entries in the upper triangular part.)

In general, using an appropriate ordering can significantly reduced the number of fill-ins and hence the computational cost. In particular, for a sparse matrix arising from n-unknown finite difference (or finite element) discretization of two-dimensional PDEs, we have noted that "natural" ordering produces an "outrigger" system with $m_\mathrm{b} = \sqrt{n}$; Gaussian elimination of the system yields an upper triangular matrix with $n(m_\mathrm{b}+1) \approx n^{3/2}$ nonzero entries. On the other hand, the number of fill-ins for the same system with an optimal (i.e. minimum fill-in) ordering yields an upper triangular matrix with $\mathcal{O}(n\log(n))$ unknowns. Thus, ordering can have significant impact in both the operation count and storage for large sparse linear systems.

End Advanced Material

27.5 The Evil Inverse

In solving a linear system $Au = f$, we advocated a two-step strategy that consists of Gaussian elimination and back substitution, i.e.

$$\text{Gaussian elimination:} \quad Au = f \quad \Rightarrow \quad Uu = \hat{f}$$
$$\text{Back substitution:} \quad Uu = \hat{f} \quad \Rightarrow \quad u\,.$$

Alternatively, we *could* find u by explicitly forming the inverse of A, A^{-1}. Recall that if A is non-singular (as indicated by, for example, independent columns), there exists a *unique* matrix A^{-1} such that

$$AA^{-1} = I \quad \text{and} \quad A^{-1}A = I.$$

The inverse matrix A^{-1} is relevant to solution systems because, in principle, we could

1. Construct A^{-1};
2. Evaluate $u = A^{-1}f$ (i.e. matrix-vector product).

Note that the second step follows from the fact that

$$\begin{aligned} Au &= f \\ A^{-1}Au &= A^{-1}f \\ Iu &= A^{-1}f\,. \end{aligned}$$

While the procedure is mathematically valid, we warn that *a linear system should never be solved by explicitly forming the inverse.*

To motivate why explicitly construction of inverse matrices should be avoided, let us study the sparsity pattern of the inverse matrix for a n-mass spring-mass system, an example of which for $n = 5$ is shown in Figure 27.8. We use the column interpretation of the matrix and associate the column j of A^{-1} with a vector p^j, i.e.

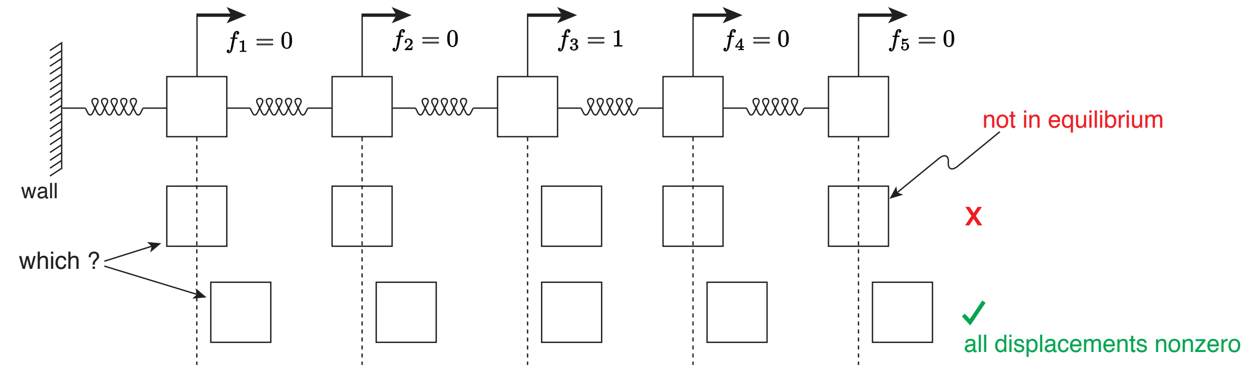


Figure 27.8: Response of a $n = 5$ spring-mass system to unit loading on mass 3.

$$A^{-1} = \begin{pmatrix} | & | & | & & | \\ p^1 & p^2 & p^3 & \cdots & p^n \\ | & | & | & & | \end{pmatrix}$$

1st column of A^{-1} 2nd column of A^{-1} nth column of A^{-1}

Since $Au = f$ and $u = A^{-1}f$, we have (using one-handed matrix-vector product),

$$\begin{aligned} u = A^{-1}f &= \begin{pmatrix} | & | & | & & | \\ p^1 & p^2 & p^3 & \cdots & p^n \\ | & | & | & & | \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \\ &= p^1 f_1 + p^2 f_2 + \cdots + p^n f_n \, . \end{aligned}$$

From this expression, it is clear that the vector p^j is equal to the displacements of masses due to the unit force acting on mass j. In particular the ith entry of p^j is the displacement of the ith mass due to the unit force on the jth mass.

Now, to deduce nonzero pattern of a vector p^j, let us focus on the case shown in Figure 27.8; we will deduce the nonzero entries of p^3 for the $n = 5$ system. Let us consider a sequence of events that takes place when f_3 is applied (we focus on qualitative result rather than quantitative result, i.e. whether masses move, not by how much):

1. Mass 3 moves to the right due to the unit load f_3.

2. Force exerted by the spring connecting mass 3 and 4 increases as the distance between mass 3 and 4 decreases.

3. Mass 4 is no longer in equilibrium as there is a larger force from the left than from the right (i.e. from the spring connecting mass 3 and 4, which is now compressed, than from the spring connecting mass 4 and 5, which is neutral).

4. Due to the unbalanced force mass 4 moves to the right.

5. The movement of mass 4 to the left triggers a sequence of event that moves mass 5, just as the movement of mass 3 displaced mass 4. Namely, the force on the spring connecting mass 4 and 5 increases, mass 5 is no longer in equilibrium, and mass 5 moves to the right.

Thus, it is clear that the unit load on mass 3 not only moves mass 3 but also mass 4 and 5 in Figure 27.8. Using the same qualitative argument, we can convince ourselves that mass 1 and 2 must also move when mass 3 is displaced by the unit load. Thus, in general, the unit load f_3 on mass 3 results in displacing all masses of the system. Recalling that the i^{th} entry of p^3 is the displacement of the i^{th} mass due to the unit load f_3, we conclude that all entries of p^3 are nonzero. (In absence of damping, the system excited by the unit load would oscillate and never come to rest; in a real system, intrinsic damping present in the springs brings the system to a new equilibrium state.)

Generalization of the above argument to a n-mass system is straightforward. Furthermore, using the same argument, we conclude that forcing of any of one of the masses results in displacing all masses. Consequently, for $p^1, \ldots, p^n$, we have

$$\begin{aligned}
u[\text{for } f = (1 \;\; 0 \;\; \cdots \;\; 0)^{\mathrm{T}}] &= p^1 \;\leftarrow\; \text{nonzero in all entries!}\\
u[\text{for } f = (0 \;\; 1 \;\; \cdots \;\; 0)^{\mathrm{T}}] &= p^2 \;\leftarrow\; \text{nonzero in all entries!}\\
&\vdots\\
u[\text{for } f = (0 \;\; 0 \;\; \cdots \;\; 0)^{\mathrm{T}}] &= p^n \;\leftarrow\; \text{nonzero in all entries!}
\end{aligned}$$

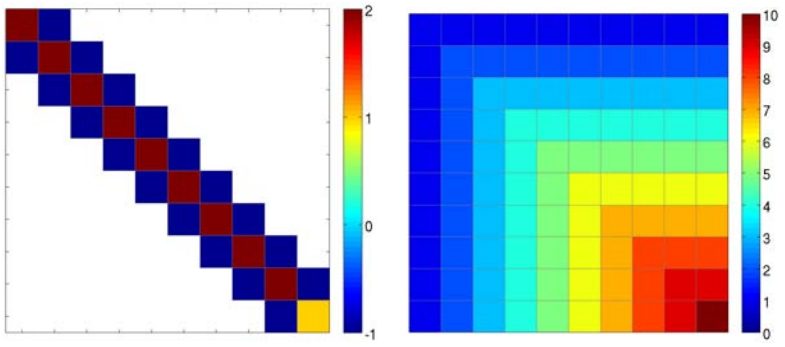
Recalling that p^j is the j^{th} column of A^{-1}, we conclude that

$$A^{-1} = \begin{pmatrix} p^1 & p^2 & \cdots & p^n \end{pmatrix}$$

is *full even though (here) A is tridiagonal.* In general A^{-1} does not preserve sparsity of A and is in fact often full. This is unlike the upper triangular matrix resulting from Gaussian elimination, which preserves a large number of zeros (modulo the fill-ins).

Figure 27.9 shows the system matrix and its inverse for the $n = 10$ spring-mass system. The colors represent the value of each entries; for instance, the A matrix has the typical $[-1 \;\; 2 \;\; -1]$ pattern, except for the first and last equations. Note that the inverse matrix is not sparse and is in fact full. In addition, the values of each column of A^{-1} agrees with our physical intuition about the displacements to a unit load. For example, when a unit load is applied to mass 3, the distance between the wall and mass 1 increases by 1 unit, the distance between mass 1 and 2 increases by 1 unit, and the distance between mass 3 and 2 increases by 1 unit; the distances between the remaining masses are unaltered because there is no external force acting on the remaining system at equilibrium (because our system is not clamped on the right end). Accumulating displacements starting with mass 1, we conclude that mass 1 moves by 1, mass 2 moves by 2 (the sum of the increased distances between mass 1 and 2 and mass 2 and 3), mass 3 moves by 3, and all the remaining masses move by 3. This is exactly the information contained in the third column of A^{-1}, which reads $[1 \;\; 2 \;\; 3 \;\; 3 \;\; \ldots \;\; 3]^{\mathrm{T}}$.

In concluding the section, let us analyze the operation count for solving a linear system by explicitly forming the inverse and performing matrix-vector multiplication. We assume that our $n \times n$ matrix A has a bandwidth of m_{b}. First, we construct the inverse matrix one column at a time



(a) A (b) A^{-1}

Figure 27.9: Matrix A for the $n = 10$ spring-mass system and its inverse A^{-1}. The colors represent the value of each entry as specified by the color bar.

by solving for the equilibrium displacements associated with unit load on each mass. To this end, we first compute the LU factorization of A and then repeat forward/backward substitution. Recalling the operation count for a single forward/backward substitution is $\mathcal{O}(nm_\mathrm{b}^2)$, the construction of A^{-1} requires

$$\begin{aligned} Ap^1 &= \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \Rightarrow \quad p^1 \quad \mathcal{O}(nm_\mathrm{b}^2) \text{ FLOPs} \\ &\vdots \\ Ap^n &= \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \quad \Rightarrow \quad p^n \quad \mathcal{O}(nm_\mathrm{b}^2) \text{ FLOPs} \end{aligned}$$

for the total work of $n \cdot \mathcal{O}(nm_\mathrm{b}^2) \sim \mathcal{O}(n^2 m_\mathrm{b}^2)$ FLOPs. Once we formed the inverse matrix, we can solve for the displacement by performing (dense) matrix-vector multiplication, i.e.

$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \underbrace{\begin{pmatrix} \times & \times & \cdots & \times \\ \times & \times & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ \times & \times & \cdots & \times \end{pmatrix}}_{A^{-1}\ \text{(full)}} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \qquad \underbrace{\mathcal{O}(n \cdot n) = \mathcal{O}(n^2) \text{ FLOPs}}_{\text{one-handed or two-handed}} \quad .$$

Thus, both the construction of the inverse matrix and the matrix-vector multiplication require $\mathcal{O}(n^2)$ operations. In contrast recall that Gaussian elimination and back substitution solves a sparse linear system in $\mathcal{O}(n)$ operations. Thus, a large sparse linear system should never be solved by explicitly forming its inverse.

Chapter 28

Sparse Matrices in Matlab

Throughout this chapter we shall assume that A is an $n \times n$ sparse matrix. By "sparse" here we mean that most of the entries of A are zero. We shall define the number of nonzero entries of A by nnz(A). Thus, by our assumption on sparsity, nnz(A) is small compared to n^2; in fact, in all of our examples, and indeed in many MechE examples, nnz(A) is typically cn, for a constant c which is $\mathcal{O}(1)$ — say $c = 3$, or 4, or 10. (We will often consider families of matrices A in which case we could state more precisely that c is independent of n.)

28.1 The Matrix Vector Product

To illustrate some of the fundamental aspects of computations with sparse matrices we shall consider the calculation of the matrix vector product, $w = Av$, for A a given $n \times n$ sparse matrix as defined above and v a given $n \times 1$ vector. (Note that we considering here the simpler forward problem, in which v is known and w unknown; in a later section we consider the more difficult "inverse" problem, in which w is known and v is unknown.)

28.1.1 A Mental Model

We first consider a mental model which provides intuition as to how the sparse matrix vector product is calculated. We then turn to actual Matlab implementation which is different in detail from the mental model but very similar in concept. There are two aspects to sparse matrices: how these matrices are stored (efficiently); and how these matrices are manipulated (efficiently). We first consider storage.

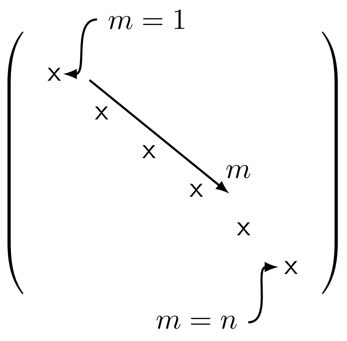
Storage

By definition our matrix A is mostly zeros and hence it would make no sense to store all the entries. Much better is to just store the nnz(A) nonzero entries with the convention that all other entries are indeed zero. This can be done by storing the indices of the nonzero entries as well as the values, as indicated in (28.1).

$$\left.\begin{array}{l} I(m), J(m),\ 1 \le m \le \texttt{nnz}(A)\colon \\ \qquad \text{indices } i = I(m),\ j = J(m) \\ \qquad \text{for which } A_{ij} \ne 0 \\ \\ V_A(m),\ 1 \le m \le \texttt{nnz}(A)\colon \\ \qquad \text{value of } A_{I(m),J(m)} \end{array}\right\} \begin{array}{c} \mathcal{O}(\texttt{nnz}(A)) \textit{ storage} \\ \ll n^2 \textit{ if } \text{sparse} \end{array} . \tag{28.1}$$

Here m is an index associated to each nonzero entry, $I(m), J(m)$ are the indices of the m^{th} nonzero entry, and $V_A(m)$ is the value of A associated with this index, $V_A(m) \equiv A_{I(m),J(m)}$. Note that V_A is a vector and not a matrix. It is clear that if A is in fact dense then the scheme (28.1) actually requires more storage than the conventional non-sparse format since we store values of the indices as well as the values of the matrix; but for sparse matrices the scheme (28.1) can result in significant economies.

As a simple example we consider $A = I$, the $n \times n$ identity matrix, for which $\texttt{nnz}(A) = n$ — a very sparse matrix. Here our sparse storage scheme can be depicted as in (28.2).



$$\left.\begin{array}{llll} m = 1\colon & I(1) = 1,\ J(1) = 1 & V_A(1) = 1 \\ m = 2\colon & I(2) = 2,\ J(2) = 2 & V_A(2) = 1 \\ & \quad\vdots & \\ m = n\colon & I(n) = 1,\ J(n) = n & V_A(n) = 1 \end{array}\right\} \begin{array}{c} \mathcal{O}(n)\ (\ll n^2) \\ \text{storage} \end{array} . \tag{28.2}$$

We note that the mapping between the index m and the nonzero entries of A is of course non-unique; in the above we identify m with the row (or equivalently, column) of the entry on the main diagonal, but we could equally well number backwards or "randomly."

Operations

We now consider the *sparse* matrix-vector product in terms of the storage scheme introduced above. In particular, we claim that

$$\underbrace{w}_{\text{to find}} = A \underbrace{v}_{\text{given}}$$

can be implemented as

```
$w = \texttt{zeros}(n, 1)$
for $m = 1\colon \texttt{nnz}(A)$
    $w(I(m)) = w(I(m)) + \underset{A_{I(m),J(m)}}{V_A(m)} \times v(J(m))$
end
```

We now discuss why this algorithm yields the desired result.

We first note that for any i such that $I(m) \neq i$ for any m — in other words, a row i of A which is entirely zero — $w(i)$ should equal zero by the usual row interpretation of the matrix vector product: $w(i)$ is the inner product between the i^{th} row of A — all zeros — and the vector v, which vanishes for any v.

$$i \rightarrow \underbrace{\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ & & & & \\ & & & & \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \times \\ \times \\ \vdots \\ \times \end{pmatrix}}_{v} = \underbrace{\begin{pmatrix} 0 \\ \\ \\ \end{pmatrix}}_{w}$$

On the other hand, for any i such that $I(m) = i$ for some m,

$$w(i) = \sum_{j=1}^{n} A_{ij} v_j = \sum_{\substack{j=1 \\ A_{ij} \neq 0}}^{n} A_{ij} v_j .$$

In both cases the sparse procedure yields the correct result and furthermore does not perform all the unnecessary operations associated with elements A_{ij} which are zero and which will clearly not contribute to the matrix vector product: zero rows of A are never visited; and in rows of A with nonzero entries only the nonzero columns are visited. We conclude that the operation count is $\mathcal{O}(\texttt{nnz}(A))$ which is much less than n^2 if our matrix is indeed sparse.

We note that we have not necessarily used all possible sparsity since in addition to zeros in A there may also be zeros in v; we may then not only disregard any row of A which are is zero but we may also disregard any column k of A for which $v_k = 0$. In practice in most MechE examples the sparsity in A is much more important to computational efficiency and arises much more often in actual practice than the sparsity in v, and hence we shall not consider the latter further.

28.1.2 Matlab Implementation

It is important to recognize that sparse is an attribute associated not just with the matrix A in a linear algebra or mathematical sense but also an attribute in Matlab (or other programming languages) which indicates a particular data type or class (or, in Matlab , attribute). In situations in which confusion might occur we shall typically refer to the former simply as sparse and the latter as "declared" sparse. In general we will realize the computational savings associated with a mathematically sparse matrix A only if the corresponding Matlab entity, `A`, is also declared sparse — it is the latter that correctly invokes the sparse storage and algorithms described in the previous section. (We recall here that the Matlab implementation of sparse storage and sparse methods is conceptually similarly to our mental model described above but not identical in terms of details.)

Storage

We proceed by introducing a brief example which illustrates most of the necessary Matlab functionality and syntax. In particular, the script

```
n = 5;
```

```
K = spalloc(n,n,3*n);

K(1,1) = 2;
K(1,2) = -1;
for i = 2:n-1
    K(i,i) = 2;
    K(i,i-1) = -1;
    K(i,i+1) = -1;
end
K(n,n) = 1;
K(n,n-1) = -1;

is_K_sparse = issparse(K)

K

num_nonzeros_K = nnz(K)

spy(K)

K_full = full(K)

K_sparse_too = sparse(K_full)
```

yields the output

```
is_K_sparse =

     1

K =

   (1,1)        2
   (2,1)       -1
   (1,2)       -1
   (2,2)        2
   (3,2)       -1
   (2,3)       -1
   (3,3)        2
   (4,3)       -1
   (3,4)       -1
   (4,4)        2
   (5,4)       -1
   (4,5)       -1
   (5,5)        1

num_nonzeros_K =
```

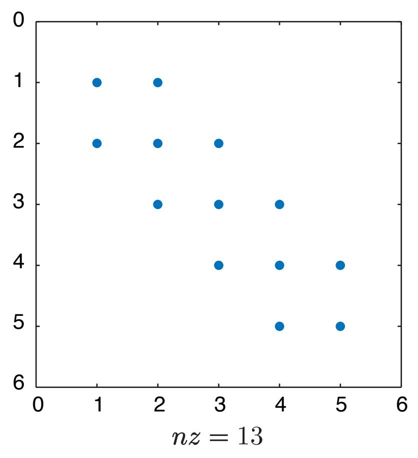


Figure 28.1: Output of `spy(K)`.

```
    13

K_full =

     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     1

K_sparse_too =

   (1,1)        2
   (2,1)       -1
   (1,2)       -1
   (2,2)        2
   (3,2)       -1
   (2,3)       -1
   (3,3)        2
   (4,3)       -1
   (3,4)       -1
   (4,4)        2
   (5,4)       -1
   (4,5)       -1
   (5,5)        1
```

as well as Figure 28.1.

We now explain the different parts in more detail:

First, `M = spalloc(n1,n2,k)` (i) creates a *declared* sparse array `M` of size `n1`$\times$ `n2` with allocation for `k` nonzero matrix entries, and then (ii) initializes the array `M` to all zeros. (If later in the script this allocation may be exceeded there is no failure or error, however efficiency will suffer as memory will not be as contiguously assigned.) In our case here, we anticipate that `K` will be tri-diagonal and hence there will be less than `3*n` nonzero entries.

Then we assign the elements of K — we create a simple tri-diagonal matrix associated with $n = 5$ springs in series. Note that although K is declared sparse we do not need to assign values according to any complicated sparse storage scheme: we assign and more generally refer to the elements of K with the usual indices, and the sparse storage bookkeeping is handled by Matlab under the hood.

We can confirm that a matrix M is indeed (declared) sparse with `issparse` — `issparse(M)` returns a logical 1 if `M` is sparse and a logical 0 if `M` is not sparse. (In the latter case, we will not save on memory or operations.) As we discuss below, some Matlab operations may accept sparse operands but under certain conditions return non-sparse results; it is often important to confirm with `issparse` that a matrix which is intended to be sparse is indeed (declared) sparse.

Now we display K. We observe directly the sparse storage format described in the previous section: Matlab displays effectively $(I(m), J(m), V_A(m))$ triplets (Matlab does not display m, which as we indicated is in any event an arbitrary label).

The Matlab built-in function `nnz(M)` returns the number of nonzero entries in a matrix `M`. The Matlab built-in function `spy(M)` displays the `n1` $\times$ `n2` matrix `M` as a rectangular grid with (only) the nonzero entries displayed as blue filled circles — Figure 28.1 displays `spy(K)`. In short, `nnz` and `spy` permit us to quantify the sparsity and structure, respectively, of a matrix `M`.

The Matlab built-in functions `full` and `sparse` create a full matrix from a (declared) sparse matrix and a (declared) sparse matrix from a full matrix respectively. Note however, that it is better to initialize a sparse matrix with `spalloc` rather than simply create a full matrix and then invoke `sparse`; the latter will require at least temporarily (but sometimes fatally) much more memory than the former.

There are many other sparse Matlab built-in functions for performing various operations.

Operations

This section is very brief: once a matrix `A` is declared sparse, then the Matlab statement `w = A*v` will invoke the efficient sparse matrix vector product described above. In short, the (matrix) multiplication operator `*` recognizes that `A` is a (declared) sparse matrix object and then automatically invokes the correct/efficient "method" of interpretation and evaluation. Note in the most common application and in our case most relevant application the matrix `A` will be declared sparse, the vector `v` will be full (i.e., not declared sparse), and the output vector `w` will be full (i.e., not declared sparse).[1] We emphasize that if the matrix `A` is mathematically sparse but *not* declared sparse then the Matlab `*` operand will invoke the standard full matrix-vector multiply and we not realize the potential computational savings.

[1]Note if the vector `v` is also declared sparse then the result `w` will be declared sparse as well.

28.2 Sparse Gaussian Elimination

This section is also very brief. As we already described in Unit III, in order to solve the matrix system $Au = f$ in Matlab we need only write `u = A \ f` — the famous backslash operator. We can now reveal, armed with the material from the current unit, that the backslash operator in fact performs Gaussian elimination (except for overdetermined systems, in which case the least-squares problem is solved by a QR algorithm). The backslash will automatically perform partial pivoting — permutations of rows to choose the maximum-magnitude available pivot — to ensure for a non-singular matrix K that a zero pivot is never encountered and that furthermore amplification of numerical round-off errors (in finite precision) is minimized.

The sparse case is similarly streamlined. If A is a mathematically sparse matrix and we wish to solve $Au = f$ by sparse Gaussian elimination as described in the previous chapter, we need only make sure that `A` is declared sparse and then write `u = A \ f` . (As for the matrix vector product, `f` need not be declared sparse and the result `u` will not be sparse.) In this case the backslash does more than simply eliminate unnecessary calculations with zero operands: the backslash will permute columns (a reordering) in order to minimize fill-in during the elimination procedure. (As for the non-sparse case, row permutations will also be pursued, for purposes of numerical stability.)

The case of A SPD is noteworthy. As already indicated, in this case the Gaussian elimination process is numerically stable without any row permutations. For an SPD matrix, the backslash operator will thus permute the rows in a similar fashion to the columns; the columns, as before, are permuted to minimize fill-in, as described in the previous chapter. A particular variant of Gaussian elimination, the Cholesky factorization, is pursued in the SPD case.

Unit VI

Nonlinear Equations

Chapter 29

Newton Iteration

29.1 Introduction

The demonstration robot arm of Figure 29.1 is represented schematically in Figure 29.2. Note that although the robot of Figure 29.1 has three degrees-of-freedom ("shoulder," "elbow," and "waist"), we will be dealing with only two degrees-of-freedom — "shoulder" and "elbow" — in this assignment.

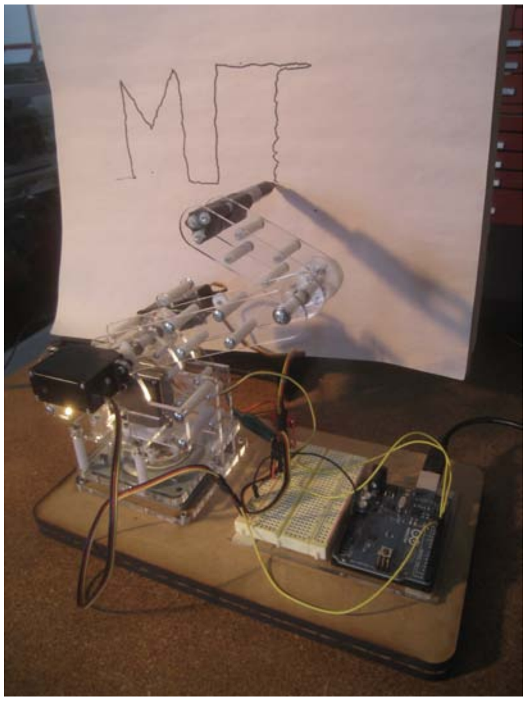


Figure 29.1: Demonstration robot arm.
(Robot and photograph courtesy of James Penn.)

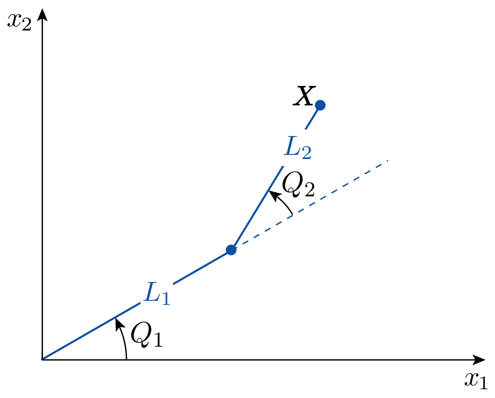


Figure 29.2: Schematic of robot arm.

The forward kinematics of the robot arm determine the coordinates of the end effector $\boldsymbol{X} =$

$[X_1, X_2]^{\mathrm{T}}$ for given joint angles $\boldsymbol{Q} = [Q_1, Q_2]^{\mathrm{T}}$ as

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}(\boldsymbol{Q}) = \begin{bmatrix} L_1\cos(Q_1) + L_2\cos(Q_1+Q_2) \\ L_1\sin(Q_1) + L_2\sin(Q_1+Q_2) \end{bmatrix}, \tag{29.1}$$

where L_1 and L_2 are the lengths of the first and second arm links, respectively. For our robot, $L_1 = 4$ inches and $L_2 = 3.025$ inches.

The inverse kinematics of the robot arm — the joint angles $\boldsymbol{Q}$ needed to realize a particular end effector position $\boldsymbol{X}$ — are not so straightforward and, for many more complex robot arms, a closed-form solution does not exist. In this assignment, we will solve the inverse kinematic problem for a two degree-of-freedom, planar robot arm by solving numerically for Q_1 and Q_2 from the set of nonlinear Equations (29.1) .

Given a trajectory of data vectors $\boldsymbol{X}_{(i)}$, $1 \le i \le p$ — a sequence of p desired end effector positions — the corresponding joint angles satisfy

$$\boldsymbol{F}\big(\boldsymbol{Q}_{(i)}, \boldsymbol{X}_{(i)}\big) = 0, \quad 1 \le i \le p\,, \tag{29.2}$$

where

$$\boldsymbol{F}(\boldsymbol{q}, \boldsymbol{X}) = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} L_1\cos(q_1) + L_2\cos(q_1+q_2) - X_1 \\ L_1\sin(q_1) + L_2\sin(q_1+q_2) - X_2 \end{bmatrix}.$$

For the robot "home" position, $\boldsymbol{X}^{\text{home}} \approx [-0.7154, 6.9635]^{\mathrm{T}}$, the joint angles are known: $\boldsymbol{Q}^{\text{home}} = [1.6, 0.17]^{\mathrm{T}}$ (in radians). We shall assume that $\boldsymbol{X}_{(1)} = \boldsymbol{X}^{\text{home}}$ and hence $\boldsymbol{Q}_{(1)} = \boldsymbol{Q}^{\text{home}}$ in all cases; it will remain to find $\boldsymbol{Q}_{(2)}, \ldots, \boldsymbol{Q}_{(p)}$.

Based on the design of our robot, we impose the following physical constraints on Q_1 and Q_2:

$$\sin(Q_1) \ge 0\,; \qquad \sin(Q_2) \ge 0\,. \tag{29.3}$$

Note that a mathematically valid solution of Equation (29.2) might not satisfy the constraints of Equation (29.3) and, therefore, will need to be checked for physical consistency.

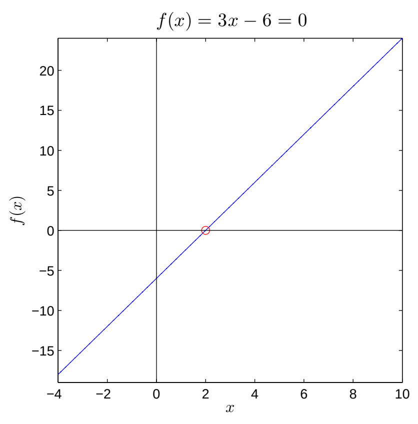
Previously we considered solving equations of the form $\boldsymbol{Ax} = \boldsymbol{b}$ for solution $\boldsymbol{X}$, given appropriately sized matrix and vector $\boldsymbol{A}$ and $\boldsymbol{b}$. In the univariate case with scalar (1×1) A and b, we could visualize the solution of these *linear* systems of equations as finding the zero crossing (root) of the line $f(x) = Ax - b$, as shown in Figure 29.3(a).

Now we will consider the solution of *nonlinear* systems of equations $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{0}$ for root $\boldsymbol{Z}$, where terms such as powers of $\boldsymbol{z}$, transcendental functions of $\boldsymbol{z}$, discontinuous functions of $\boldsymbol{z}$, or any other such nonlinearities preclude the linear model. In the univariate case, we can visualize the solution of the nonlinear system as finding the roots of a nonlinear function, as shown in Figure 29.3(b) for a cubic, $f(z)$. Our robot example of Figures 29.1 and 29.2 represent a bivariate example of a nonlinear system (in which $\boldsymbol{F}$ plays the role of $\boldsymbol{f}$, and $\boldsymbol{Q}$ plays the role of $\boldsymbol{Z}$ — the root we wish to find).

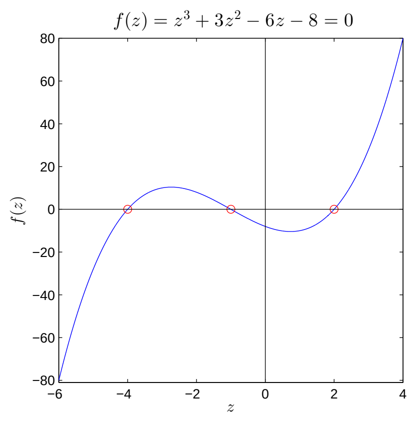
In general, a linear system of equations may have no solution, one solution (a unique solution), or an infinite family of solutions. In contrast, a nonlinear problem may have no solution, one solution, two solutions, three solutions (as in Figure 29.3(b)), *any* number of solutions, or an infinite family of solutions. We will also need to decide which solutions (of a nonlinear problem) are of interest or relevant given the context and also stability considerations.

The nonlinear problem is typically solved as a sequence of linear problems — hence builds directly on linear algebra. The sequence of linear problems can be generated in a variety of fashions; our focus here is Newton's method. Many other approaches are also possible — for example, least squares/optimization — for solution of nonlinear problems.

The fundamental approach of Newton's method is simple:



(a) linear function



(b) nonlinear function

Figure 29.3: Solutions of univariate linear and nonlinear equations.

- Start with an initial guess or approximation for the root of a nonlinear system.
- Linearize the system around that initial approximation and solve the resulting *linear* system to find a better approximation for the root.
- Continue linearizing and solving until satisfied with the accuracy of the approximation.

This approach is identical for both univariate (one equation in one variable) and multivariate (n equations in n variables) systems. We will first consider the univariate case and then extend our analysis to the multivariate case.

29.2 Univariate Newton

29.2.1 The Method

Given a univariate function $f(z)$, we wish to find a *real* zero/root Z of f such that $f(Z) = 0$. Note that z is any real value (for which the function is defined), whereas Z is a particular value of z at which $f(z = Z)$ is *zero*; in other words, Z is a *root* of $f(z)$.

We first start with an initial approximation (guess) $\hat{z}^0$ for the zero Z. We next approximate the function $f(z)$ with its first-order Taylor series expansion around $\hat{z}^0$, which is the line tangent to $f(z)$ at $z = \hat{z}^0$

$$f^0_{\text{linear}}(z) \equiv f'(\hat{z}^0)(z - \hat{z}^0) + f(\hat{z}^0) \ . \tag{29.4}$$

We find the zero $\hat{z}^1$ of the linearized system $f^0_{\text{linear}}(z)$ by

$$f^0_{\text{linear}}(\hat{z}^1) \equiv f'(\hat{z}^0)(\hat{z}^1 - \hat{z}^0) + f(\hat{z}^0) = 0 \ , \tag{29.5}$$

which yields

$$\hat{z}^1 = \hat{z}^0 - \frac{f(\hat{z}^0)}{f'(\hat{z}^0)} \ . \tag{29.6}$$

We then repeat the procedure with $\hat{z}^1$ to find $\hat{z}^2$ and so on, finding successively better approximations to the zero of the original system $f(z)$ from our linear approximations $f^k_{\text{linear}}(z)$ until we reach $\hat{z}^N$ such that $|f(\hat{z}^N)|$ is within some desired tolerance of zero. (To be more rigorous, we must relate $|f(\hat{z}^N)|$ to $|Z - \hat{z}^N|$.)

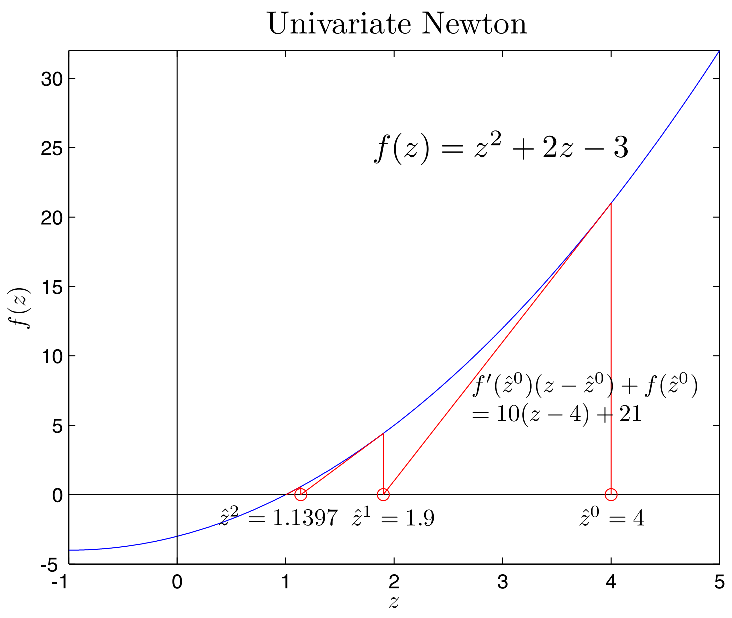


Figure 29.4: Graphical illustration of the Newton root finding method.

29.2.2 An Example

The first few steps of the following example have been illustrated in Figure 29.4.

We wish to use the Newton method to find a solution to the equation

$$z^2 + 2z = 3 \ . \tag{29.7}$$

We begin by converting the problem to a root-finding problem

$$f(Z) = Z^2 + 2Z - 3 = 0 \ . \tag{29.8}$$

We next note that the derivative of f is given by

$$f'(z) = 2z + 2 \ . \tag{29.9}$$

We start with initial guess $\hat{z}^0 = 4$. We then linearize around $\hat{z}^0$ to get

$$f^0_{\text{linear}}(z) \equiv f'(\hat{z}^0)(z - \hat{z}^0) + f(\hat{z}^0) = 10(z - 4) + 21 \ . \tag{29.10}$$

We solve the linearized system

$$f^0_{\text{linear}}(\hat{z}^1) \equiv 10(z - 4) + 21 = 0 \tag{29.11}$$

to find the next approximation for the root of $f(z)$,

$$\hat{z}^1 = 1.9 \ . \tag{29.12}$$

We repeat the procedure to find

$$f^1_{\text{linear}}(z) \equiv f'(\hat{z}^1)(z - \hat{z}^1) + f(\hat{z}^1) = 5.8(z - 1.9) + 4.41 \tag{29.13}$$

$$f^1_{\text{linear}}(\hat{z}^2) = 0 \tag{29.14}$$

$$\hat{z}^2 = 1.1397\ ; \tag{29.15}$$

$$f^2_{\text{linear}}(z) \equiv f'(\hat{z}^2)(z - \hat{z}^2) + f(\hat{z}^2) = 4.2793(z - 1.1397) + 0.5781 \tag{29.16}$$

$$f^2_{\text{linear}}(\hat{z}^3) = 0 \tag{29.17}$$

$$\hat{z}^3 = 1.0046\ . \tag{29.18}$$

Note the rapid convergence to the actual root $Z = 1$. Within three iterations the error has been reduced to 0.15% of its original value.

29.2.3 The Algorithm

The algorithm for the univariate Newton's method is a simple **while** loop. If we wish to store the intermediate approximations for the zero, the algorithm is shown in Algorithm 1. If we don't need to save the intermediate approximations, the algorithm is shown in Algorithm 2.

If, for some reason, we cannot compute the derivative $f'(\hat{z}^k)$ directly, we can substitute the finite difference approximation of the derivative (see Chapter 3) for some arbitrary (small) given Δz, which, for the backward difference is given by

$$f'(\hat{z}^k) \approx \frac{f(\hat{z}^k) - f(\hat{z}^k - \Delta z)}{\Delta z}\ . \tag{29.19}$$

In practice, for Δz sufficiently small (but not too small — round-off), Newton with approximate derivative will behave similarly to Newton with exact derivative.

Algorithm 1 Newton algorithm with storage of intermediate approximations

```
$k \leftarrow 0$
while $\left|f(\hat{z}^k)\right| > \text{tol}$ do
    $\hat{z}^{k+1} \leftarrow \hat{z}^k - \frac{f(\hat{z}^k)}{f'(\hat{z}^k)}$
    $k \leftarrow k + 1$
end while
$Z \leftarrow \hat{z}^k$
```

Algorithm 2 Newton algorithm without storage

```
$\hat{z} \leftarrow \hat{z}^0$
while $\left|f(\hat{z})\right| > \text{tol}$ do
    $\delta\hat{z} \leftarrow \frac{-f(\hat{z})}{f'(\hat{z})}$
    $\hat{z} \leftarrow \hat{z} + \delta\hat{z}$
end while
$Z \leftarrow \hat{z}$
```

There also exists a method, based on Newton iteration, that directly incorporates a finite difference approximation of the derivative by using the function values at the two previous iterations (thus requiring two initial guesses) to construct

$$f'(\hat{z}^k) \approx \frac{f(\hat{z}^k) - f(\hat{z}^{k-1})}{\hat{z}^k - \hat{z}^{k-1}} \ . \tag{29.20}$$

This is called the *secant* method because the linear approximation of the function is no longer a line tangent to the function, but a secant line. This method works well with one variable (with a modest reduction in convergence rate); the generalization of the secant method to the multivariate case (and quasi-Newton methods) is more advanced.

Another root-finding method that works well (although slowly) in the univariate case is the bisection (or "binary chop") method. The bisection method finds the root within a given interval by dividing the interval in half on each iteration and keeping only the half whose function evaluations at its endpoints are of opposite sign — and which, therefore, must contain the root. This method is very simple and robust — it works even for non-smooth functions — but, because it fails to exploit any information regarding the derivative of the function, it is slow. It also cannot be generalized to the multivariate case.

29.2.4 Convergence Rate

When Newton works, it works extremely fast. More precisely, if we denote the error in Newton's approximation for the root at the k^{th} iteration as

$$\epsilon^k = \hat{z}^k - Z, \tag{29.21}$$

then if

(*i*) $f(z)$ is smooth (e.g., the second derivative exists),

(*ii*) $f'(Z) \neq 0$ (i.e., the derivative at the root is nonzero), and

(*iii*) $|\epsilon^0|$ (the error of our initial guess) is sufficiently small,

we can show that we achieve *quadratic* (i.e., $\epsilon^{k+1} \sim (\epsilon^k)^2$) convergence:

$$\epsilon^{k+1} \sim (\epsilon^k)^2 \left(\frac{1}{2} \frac{f''(Z)}{f'(Z)} \right) \ . \tag{29.22}$$

Each iteration *doubles* the number of correct digits; this is extremely fast convergence. For our previous example, the sequence of approximations obtained is shown in Table 29.1. Note that the doubling of correct digits applies only once we have at least one correct digit. For the secant method, the convergence rate is slightly slower:

$$\epsilon^{k+1} \sim (\epsilon^k)^\gamma \ , \tag{29.23}$$

where $\gamma \approx 1.618$.

Proof. A sketch of the proof for Newton's convergence rate is shown below:

iteration	approximation	number of correct digits
0	4	0
1	1.9	1
2	1.1...	1
3	1.004...	3
4	1.000005...	6
5	1.000000000006...	12

Table 29.1: Convergence of Newton for the example problem.

$$\hat{z}^{k+1} = \hat{z}^k - \frac{f(\hat{z}^k)}{f'(\hat{z}^k)} \tag{29.24}$$

$$\cancel{Z} + \epsilon^{k+1} = \cancel{Z} + \epsilon^k - \frac{f(Z+\epsilon^k)}{f'(Z+\epsilon^k)} \tag{29.25}$$

$$\epsilon^{k+1} = \epsilon^k - \frac{\cancel{f(Z)} + \epsilon^k f'(Z) + \frac{1}{2}(\epsilon^k)^2 f''(Z) + \cdots}{f'(Z) + \epsilon^k f''(Z) + \cdots} \tag{29.26}$$

$$\epsilon^{k+1} = \epsilon^k - \epsilon^k \frac{f'(Z) + \frac{1}{2}\epsilon^k f''(Z) + \cdots}{f'(Z)(1 + \epsilon^k \frac{f''(Z)}{f'(Z)} + \cdots)} \tag{29.27}$$

$$\epsilon^{k+1} = \epsilon^k - \epsilon^k \frac{\cancel{f'(Z)}(1 + \frac{1}{2}\epsilon^k \frac{f''(Z)}{f'(Z)} + \cdots}{\cancel{f'(Z)}(1 + \epsilon^k \frac{f''(Z)}{f'(Z)} + \cdots)} \; ; \tag{29.28}$$

since $\frac{1}{1+\rho} \sim 1 - \rho + \cdots$ for small ρ,

$$\epsilon^{k+1} = \epsilon^k - \epsilon^k \left(1 + \frac{1}{2}\epsilon^k \frac{f''(Z}{f'(Z)}\right)\left(1 - \epsilon^k \frac{f''(Z)}{f'(Z)}\right) + \cdots \tag{29.29}$$

$$\epsilon^{k+1} = \cancel{\epsilon^k} - \cancel{\epsilon^k} + \frac{1}{2}(\epsilon^k)^2 \frac{f''(Z)}{f'(Z)} + \cdots \tag{29.30}$$

$$\epsilon^{k+1} = \frac{1}{2}\frac{f''(Z)}{f'(Z)}(\epsilon^k)^2 + \cdots \; . \tag{29.31}$$

We thus confirm the quadratic convergence. □

Note that, if $f'(Z) = 0$, we must stop at equation (29.26) to obtain the *linear* (i.e., $\epsilon^{k+1} \sim \epsilon^k$) convergence rate

$$\epsilon^{k+1} = \epsilon^k - \frac{\frac{1}{2}(\epsilon^k)^2 f''(Z) + \cdots}{\epsilon^k f''(Z)} \tag{29.32}$$

$$\epsilon^{k+1} = \frac{1}{2}\epsilon^k + \cdots \; . \tag{29.33}$$

In this case, we gain only a *constant* number of correct digits after each iteration. The bisection method also displays linear convergence.

29.2.5 Newton Pathologies

Although Newton often does work well and very fast, we must always be careful not to excite pathological (i.e., atypically bad) behavior through our choice of initial guess or through the nonlinear function itself.

For example, we can easily — and, thus, this might be less pathology than generally bad behavior — arrive at an "incorrect" solution with Newton if our initial guess is poor. For instance, in our earlier example of Figure 29.4, say that we are interested in finding a positive root, $Z > 0$, of $f(z)$. If we had chosen an initial guess of $\hat{z}^0 = -4$ instead of $\hat{z}^0 = +4$, we would have (deservingly) found the root $Z = -3$ instead of $Z = 1$. Although in the univariate case we can often avoid this behavior by basing our initial guess on a prior inspection of the function, this approach becomes more difficult in higher dimensions.

Even more diabolical behavior is possible. If the nonlinear function has a local maximum or minimum, it is often possible to excite oscillatory behavior in Newton through our choice of initial guess. For example, if the linear approximations at two points on the function both return the other point as their solution, Newton will oscillate between the two indefinitely and never converge to any roots. Similarly, if the first derivative is not well behaved in the vicinity of the root, then Newton may diverge to infinity.

We will later address these issues (when possible) by continuation and homotopy methods.

29.3 Multivariate Newton

29.3.1 A Model Problem

Now we will apply the Newton method to solve multivariate nonlinear systems of equations. For example, we can consider the simple *bivariate* system of nonlinear equations

$$\begin{aligned} f_1(z_1, z_2) &= z_1^2 + 2z_2^2 - 22 = 0 \,, \\ f_2(z_1, z_2) &= 2z_1^2 + z_2^2 - 17 = 0 \,. \end{aligned} \tag{29.34}$$

Note that f_1 and f_2 are the two paraboloids each with principal axes aligned with the coordinate directions; we plot f_1 and f_2 in shown in Figure 29.5. We wish to find a zero $\boldsymbol{Z} = (z_1 \;\; z_2)^{\mathrm{T}}$ of $\boldsymbol{f}(\boldsymbol{z}) = (f_1(z_1, z_2) \;\; f_2(z_1, z_2))^{\mathrm{T}}$ such that $\boldsymbol{f}(\boldsymbol{Z}) = \boldsymbol{0}$. We can visualize $\boldsymbol{Z}$ as the intersections of the ellipses $f_1 = 0$ (the intersection of paraboloid f_1 with the zero plane) and $f_2 = 0$ (the intersection of paraboloid f_2 with the zero plane). The four solutions $(Z_1, Z_2) = (\pm 2, \pm 3)$ are shown as the red circles in the contour plot of Figure 29.6.

29.3.2 The Method

The Newton method for the multivariate case follows the same approach as for the univariate case:

- Start with an initial approximation $\hat{\boldsymbol{z}}^0$ of a root to the nonlinear system $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{0}$.
- Linearize the system at $\hat{\boldsymbol{z}}^0$ and solve the resulting linear system to derive a better approximation $\hat{\boldsymbol{z}}^1$.
- Continue linearizing and solving until the norm of $\boldsymbol{f}(\hat{\boldsymbol{z}}^N)$ is within some desired tolerance of zero.

However, the multivariate case can be much more challenging computationally.

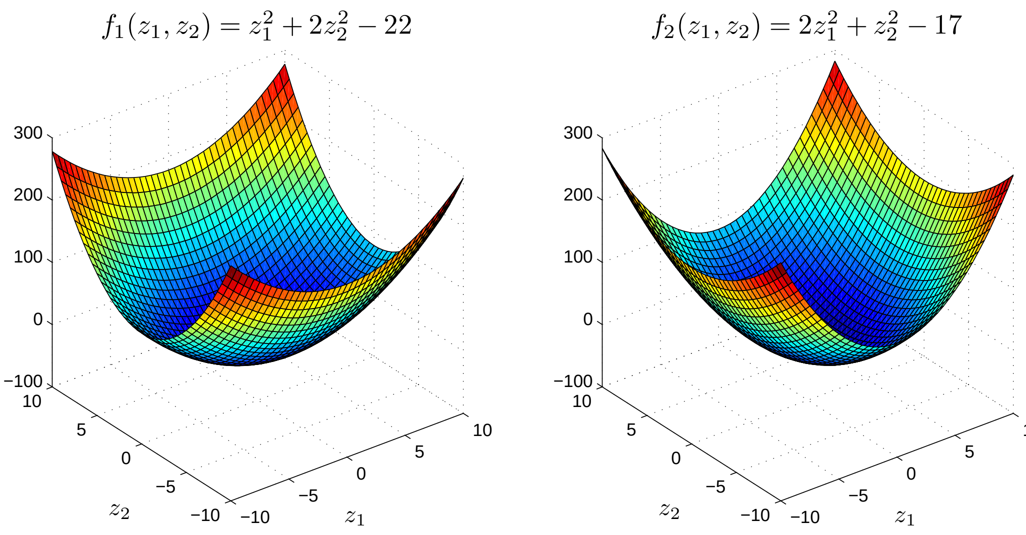


Figure 29.5: Elliptic Paraboloids f_1 and f_2.

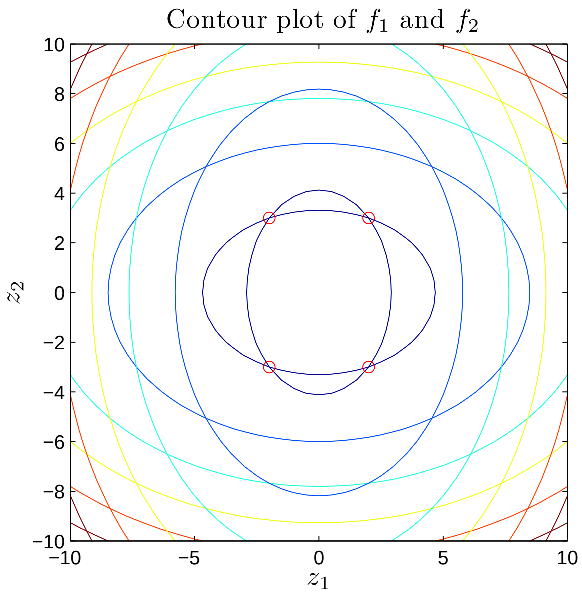


Figure 29.6: Contour plots of f_1 and f_2 with their intersections of the zero contours (the solutions to $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{0}$) shown as red circles.

We present the method for the general case in which $\boldsymbol{z}$ is an n-vector, $(z_1 \;\; z_2 \;\; \cdots \;\; z_n)^{\mathrm{T}}$ and $\boldsymbol{f}(\boldsymbol{z})$ is also an n-vector, $(f_1(\boldsymbol{z}) \;\; f_2(\boldsymbol{z}) \;\; \cdot \;\; f_n(\boldsymbol{z}))^{\mathrm{T}}$. This represents n *nonlinear* equations in n unknowns. (Of course, even more generally, we could consider more equations than unknowns or less equations than unknowns.) To linearize the multivariate system, we again use a first-order Taylor series expansion, which, for a single multivariate function linearized at the point $\hat{\boldsymbol{z}}^k$, is given by

$$f^k_{\text{linear}}(\boldsymbol{z}) \equiv \left.\frac{\partial f}{\partial z_1}\right|_{\hat{\boldsymbol{z}}^k}(z_1 - \hat{z}^k_1) + \left.\frac{\partial f}{\partial z_2}\right|_{\hat{\boldsymbol{z}}^k}(z_2 - \hat{z}^k_2) + \cdots + \left.\frac{\partial f}{\partial z_n}\right|_{\hat{\boldsymbol{z}}^k}(z_n - \hat{z}^k_n) + f(\hat{\boldsymbol{z}}^k) \tag{29.35}$$

(cf. equation (29.4) in the univariate case). Because we have n equations in n variables, we must linearize each of the equations $(f_1(\boldsymbol{z}) \;\; f_2(\boldsymbol{z}) \;\; \ldots \;\; f_n(\boldsymbol{z}))^{\mathrm{T}}$ and then, per the Newton recipe, set $(f_1(\hat{\boldsymbol{z}}^k) = 0 \;\; \ldots \;\; f_n(\hat{\boldsymbol{z}}^k) = 0)^{\mathrm{T}}$. Our full linearized system looks like

$$f^k_{1,\text{linear}}(\hat{\boldsymbol{z}}^{k+1}) \equiv \left.\frac{\partial f_1}{\partial z_1}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_1 - \hat{z}^k_1) + \left.\frac{\partial f_1}{\partial z_2}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_2 - \hat{z}^k_2) + \cdots + \left.\frac{\partial f_1}{\partial z_n}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_n - \hat{z}^k_n) + f_1(\hat{\boldsymbol{z}}^k) = 0\ , \tag{29.36}$$

$$f^k_{2,\text{linear}}(\hat{\boldsymbol{z}}^{k+1}) \equiv \left.\frac{\partial f_2}{\partial z_1}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_1 - \hat{z}^k_1) + \left.\frac{\partial f_2}{\partial z_2}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_2 - \hat{z}^k_2) + \cdots + \left.\frac{\partial f_2}{\partial z_n}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_n - \hat{z}^k_n) + f_2(\hat{\boldsymbol{z}}^k) = 0\ , \tag{29.37}$$

up through

$$f^k_{n,\text{linear}}(\hat{\boldsymbol{z}}^{k+1}) \equiv \left.\frac{\partial f_n}{\partial z_1}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_1 - \hat{z}^k_1) + \left.\frac{\partial f_n}{\partial z_2}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_2 - \hat{z}^k_2) + \cdots + \left.\frac{\partial f_n}{\partial z_n}\right|_{\hat{\boldsymbol{z}}^k}(\hat{z}^{k+1}_n - \hat{z}^k_n) + f_n(\hat{\boldsymbol{z}}^k) = 0 \tag{29.38}$$

(cf. equation (29.5) in the univariate case).

We can write the *linear* system of equations (29.36)–(29.38) in matrix form as

$$\boldsymbol{f}^k_{\text{linear}}(\hat{\boldsymbol{z}}^{k+1}) \equiv \left.\begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \frac{\partial f_1}{\partial z_2} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \frac{\partial f_2}{\partial z_1} & \frac{\partial f_2}{\partial z_2} & \cdots & \frac{\partial f_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \frac{\partial f_n}{\partial z_2} & \cdots & \frac{\partial f_n}{\partial z_n} \end{bmatrix}\right|_{\hat{\boldsymbol{z}}^k} \begin{bmatrix} (\hat{z}^{k+1}_1 - \hat{z}^k_1) \\ (\hat{z}^{k+1}_2 - \hat{z}^k_2) \\ \vdots \\ (\hat{z}^{k+1}_n - \hat{z}^k_n) \end{bmatrix} + \begin{bmatrix} f_1(\hat{z}^k) \\ f_2(\hat{z}^k) \\ \vdots \\ f_n(\hat{z}^k) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{29.39}$$

or, equivalently,

$$\boldsymbol{J}(\hat{\boldsymbol{z}}^k)\delta\hat{\boldsymbol{z}}^k = -\boldsymbol{f}(\hat{\boldsymbol{z}}^k)\ . \tag{29.40}$$

Here the $n \times n$ *Jacobian* matrix $\boldsymbol{J}$ is defined as the matrix of all first-order partial derivatives of the function vector $(f_1 \;\; \ldots \;\; f_n)^{\mathrm{T}}$ with respect to the state vector $(z_1 \;\; \ldots \;\; z_n)^{\mathrm{T}}$:

$$\boldsymbol{J}(\boldsymbol{z}) = \left.\begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \frac{\partial f_1}{\partial z_2} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \frac{\partial f_2}{\partial z_1} & \frac{\partial f_2}{\partial z_2} & \cdots & \frac{\partial f_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \frac{\partial f_n}{\partial z_2} & \cdots & \frac{\partial f_n}{\partial z_n} \end{bmatrix}\right|_{\boldsymbol{z}}\ , \tag{29.41}$$

such that the i, j^{th} component of the Jacobian corresponds to the partial derivative of the i^{th} function with respect to the j^{th} variable,

$$J_{ij}(\boldsymbol{z}) = \frac{\partial f_i}{\partial z_j}(\boldsymbol{z}) \ . \tag{29.42}$$

Thus $\boldsymbol{J}(\hat{\boldsymbol{z}}^k)$ denotes the Jacobian matrix for the system evaluated at the point $\hat{\boldsymbol{z}}^k$. Note also that $\delta\hat{\boldsymbol{z}}^k$ is the displacement vector pointing from the current approximation $\hat{\boldsymbol{z}}^k$ to the next approximation $\hat{\boldsymbol{z}}^{k+1}$

$$\delta\hat{\boldsymbol{z}}^k = \begin{bmatrix} (\hat{z}_1^{k+1} - \hat{z}_1^k) \\ (\hat{z}_2^{k+1} - \hat{z}_2^k) \\ \vdots \\ (\hat{z}_n^{k+1} - \hat{z}_n^k) \end{bmatrix} . \tag{29.43}$$

Hence $\delta\hat{\boldsymbol{z}}^k$ is the Newton update to our current iterate.

29.3.3 An Example

We can now apply Newton to solve the $n = 2$ model problem described in Section 29.3.1:

$$\begin{aligned} f_1(z_1, z_2) &= z_1^2 + 2z_2^2 - 22 = 0 \ , \\ f_2(z_1, z_2) &= 2z_1^2 + z_2^2 - 17 = 0 \ . \end{aligned} \tag{29.44}$$

We first compute the elements of the Jacobian matrix as

$$J_{11} = \frac{\partial f_1}{\partial z_1} = 2z_1, \quad J_{12} = \frac{\partial f_1}{\partial z_2} = 4z_2, \quad J_{21} = \frac{\partial f_2}{\partial z_1} = 4z_1, \quad J_{22} = \frac{\partial f_2}{\partial z_2} = 2z_2 \ , \tag{29.45}$$

and write the full matrix as

$$\boldsymbol{J}(\boldsymbol{z}) = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} = \begin{bmatrix} 2z_1 & 4z_2 \\ 4z_1 & 2z_2 \end{bmatrix} . \tag{29.46}$$

We can now perform the iteration.

We start with initial guess $\hat{\boldsymbol{z}}^0 = (10 \ \ 10)^{\mathrm{T}}$. We next linearize around $\hat{\boldsymbol{z}}^0$ to get

$$\boldsymbol{f}^0_{\text{linear}}(\boldsymbol{z}) \equiv \boldsymbol{J}(\hat{\boldsymbol{z}}^0)(\boldsymbol{z} - \hat{\boldsymbol{z}}^0) + \boldsymbol{f}(\hat{\boldsymbol{z}}^0) = \begin{bmatrix} 20 & 40 \\ 40 & 20 \end{bmatrix} \delta\hat{\boldsymbol{z}}^0 + \begin{bmatrix} 278 \\ 283 \end{bmatrix} . \tag{29.47}$$

Note that we can visualize this system as two planes tangent to f_1 and f_2 at $\hat{\boldsymbol{z}}^0$. We now solve the linearized system

$$\boldsymbol{f}^0_{\text{linear}}(\hat{\boldsymbol{z}}^1) \equiv \begin{bmatrix} 20 & 40 \\ 40 & 20 \end{bmatrix} \delta\hat{\boldsymbol{z}}^0 + \begin{bmatrix} 278 \\ 283 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{29.48}$$

or

$$\begin{bmatrix} 20 & 40 \\ 40 & 20 \end{bmatrix} \delta\boldsymbol{z}^0 = \begin{bmatrix} -278 \\ -283 \end{bmatrix} \tag{29.49}$$

to find

$$\delta\hat{\boldsymbol{z}}^0 = \begin{bmatrix} -4.8 \\ \\ -4.55 \end{bmatrix} ; \tag{29.50}$$

thus the next approximation for the root of $\boldsymbol{f}(z)$ is given by

$$\hat{\boldsymbol{z}}^1 = \hat{\boldsymbol{z}}^0 + \delta\hat{\boldsymbol{z}}^0 = \begin{bmatrix} 5.2 \\ \\ 5.45 \end{bmatrix} . \tag{29.51}$$

We repeat the procedure to find

$$\boldsymbol{f}^1_{\text{linear}}(\boldsymbol{z}) \equiv \boldsymbol{J}(\hat{\boldsymbol{z}}^1)(\boldsymbol{z} - \hat{\boldsymbol{z}}^1) + \boldsymbol{f}(\hat{\boldsymbol{z}}^1) = \begin{bmatrix} 10.4 & 21.8 \\ \\ 20.8 & 10.9 \end{bmatrix} \delta\hat{\boldsymbol{z}}^1 + \begin{bmatrix} 64.445 \\ \\ 66.7825 \end{bmatrix} , \tag{29.52}$$

$$\boldsymbol{f}^1_{\text{linear}}(\hat{\boldsymbol{z}}^2) = \boldsymbol{0} \tag{29.53}$$

$$\hat{\boldsymbol{z}}^2 = \begin{bmatrix} 2.9846 \\ \\ 3.5507 \end{bmatrix} ; \tag{29.54}$$

$$\boldsymbol{f}^2_{\text{linear}}(\boldsymbol{z}) = \equiv \boldsymbol{J}(\hat{\boldsymbol{z}}^2)(\boldsymbol{z} - \hat{\boldsymbol{z}}^2) + \boldsymbol{f}(\hat{\boldsymbol{z}}^2) = \begin{bmatrix} 5.9692 & 14.2028 \\ \\ 11.9385 & 7.1014 \end{bmatrix} \delta\hat{\boldsymbol{z}}^2 + \begin{bmatrix} 12.1227 \\ \\ 13.4232 \end{bmatrix} , \tag{29.55}$$

$$\boldsymbol{f}^2_{\text{linear}}(\hat{\boldsymbol{z}}^3) = \boldsymbol{0} \tag{29.56}$$

$$\hat{\boldsymbol{z}}^3 = \begin{bmatrix} 2.1624 \\ \\ 3.0427 \end{bmatrix} . \tag{29.57}$$

We see that the solution rapidly approaches the (nearest) exact solution $\boldsymbol{Z} = (2 \;\; 3)^{\text{T}}$.

29.3.4 The Algorithm

The multivariate Newton algorithm is identical to the univariate algorithm, except that now for each pass through the **while** loop we must now solve a linearized *system* of equations involving the Jacobian matrix.

Algorithm 3 Multivariate Newton algorithm without storage

```
$\hat{\boldsymbol{z}} \leftarrow \hat{\boldsymbol{z}}^0$
while $\|\boldsymbol{f}(\hat{\boldsymbol{z}})\| >$ tol do
    {Solve the linearized system for $\delta\hat{\boldsymbol{z}}$.}
    $\boldsymbol{J}(\hat{\boldsymbol{z}})\delta\hat{\boldsymbol{z}} = -\boldsymbol{f}(\hat{\boldsymbol{z}})$
    $\hat{\boldsymbol{z}} \leftarrow \hat{\boldsymbol{z}} + \delta\hat{\boldsymbol{z}}$
end while
$\boldsymbol{Z} \leftarrow \hat{\boldsymbol{z}}$
```

We can see that the computational cost for the multivariate Newton iteration is essentially the total number of iterations multiplied by the cost to solve an $n \times n$ linear system — which, if dense, could be as much as $\mathcal{O}(n^3)$ and, if sparse, as little as $\mathcal{O}(n)$ operations. Additionally, for "pure" Newton, the Jacobian needs to be recomputed ($\mathcal{O}(n^2)$ operations) at each iteration. (In "impure" Newton the Jacobian sometimes is held fixed for several iterations or updated selectively.)

Note that, as before in the univariate case, we can substitute a finite difference approximation for the Jacobian if we can not (or choose not to) use the analytical expressions. Here we first introduce a scalar Δz (small); note that Δz is not related to $\delta \boldsymbol{z}$ — i.e., this is not a secant approach. Then we approximate, for $1 \leq i \leq n$, $1 \leq j \leq n$,

$$J_{ij}(\boldsymbol{z}) \equiv \frac{\partial f_i}{\partial z_j}(\boldsymbol{z}) \approx \frac{f_i(\boldsymbol{z} + \Delta z \boldsymbol{e}^j) - f_i(\boldsymbol{z})}{\Delta z} \equiv \widetilde{J}_{ij}(\boldsymbol{z}), \tag{29.58}$$

where $\boldsymbol{e}^j$ is the unit vector in the j-direction such that $\boldsymbol{z} + \Delta z \boldsymbol{e}^j$ differs from $\boldsymbol{z}$ in only the j^{th} component — a partial difference approximation to the partial derivative $\frac{\partial f_i}{\partial z_j}$. Note that to compute the full finite difference approximation $n \times n$ matrix $\widetilde{\boldsymbol{J}}(\boldsymbol{z})$ we need $\mathcal{O}(n^2)$ function evaluations.

29.3.5 Comments on Multivariate Newton

For multivariate Newton, both the convergence rate and the pathologies are similar to the univariate case, although the pathologies are somewhat more likely in the multivariate case given the greater number of degrees of freedom.

The main difference for the multivariate case, then, is the relative cost of each Newton iteration (worst case $\mathcal{O}(n^3)$ operations) owing to the size of the linear system which must be solved. For this reason, Newton's rapid convergence becomes more crucial with growing dimensionality. Thanks to the rapid convergence, Newton often outperforms "simpler" approaches which are less computationally expensive per iteration but require many more iterations to converge to a solution.

29.4 Continuation and Homotopy

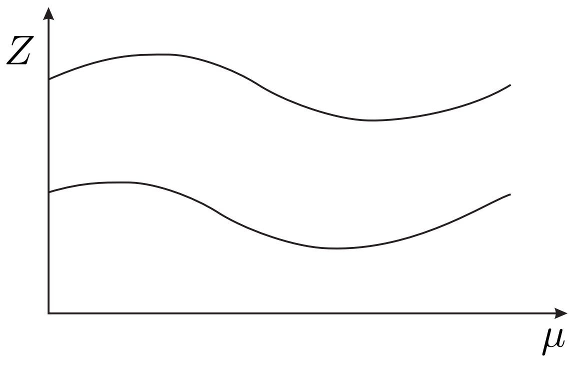
Often we are interested not in solving just a single nonlinear problem, but rather a family of nonlinear problems $\boldsymbol{f}(\boldsymbol{Z};\mu) = \boldsymbol{0}$ with real parameter μ which can take on a sequence of values $\mu_{(1)}, \ldots, \mu_{(p)}$. Typically we supplement $\boldsymbol{f}(\boldsymbol{Z};\mu) = \boldsymbol{0}$ with some simple constraints or continuity conditions which select a particular solution from several possible solutions.

We then wish to ensure that, (i) we are able to start out at all, often by transforming the initial problem for $\mu = \mu_{(1)}$ (and perhaps also subsequent problems for $\mu = \mu_{(i)}$) into a series of simpler problems (homotopy) and, (ii) once started, and as the parameter μ is varied, we continue to converge to "correct" solutions as defined by our constraints and continuity conditions (continuation).

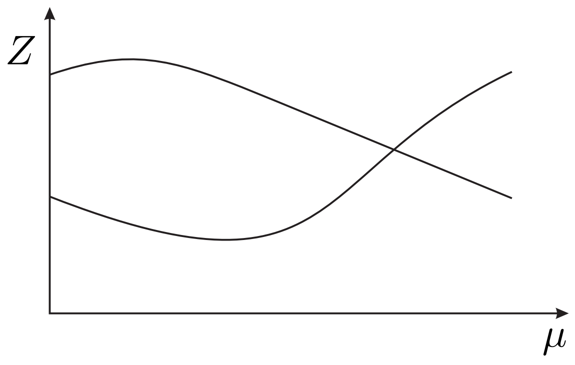
29.4.1 Parametrized Nonlinear Problems: A Single Parameter

Given $\boldsymbol{f}(\boldsymbol{Z};\mu) = 0$ with real single parameter μ, we are typically interested in how our solution $\boldsymbol{Z}$ changes as we change μ; in particular, we now interpret (*à la* the implicit function theorem) $\boldsymbol{Z}$ as a function $\boldsymbol{Z}(\mu)$. We can visualize this dependency by plotting Z (here $n = 1$) with respect to μ, giving us a *bifurcation diagram* of the problem, as depicted in Figure 29.7 for several common modes of behavior.

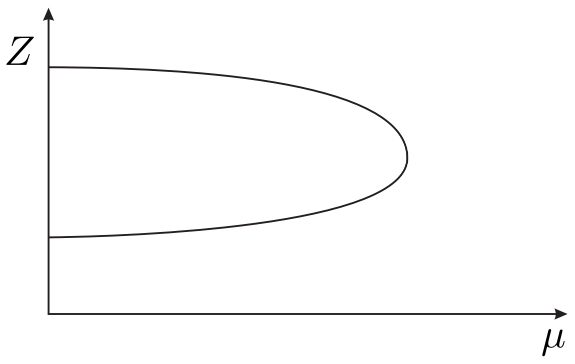
Figure 29.7(a) depicts two isolated solution branches with two, distinct real solutions over the whole range of μ. Figure 29.7(b) depicts two solution branches that converge at a *singular point*, where a change in μ can drive the solution onto either of the two branches. Figure 29.7(c) depicts two solution branches that converge at a *limit point*, beyond which there is no solution.



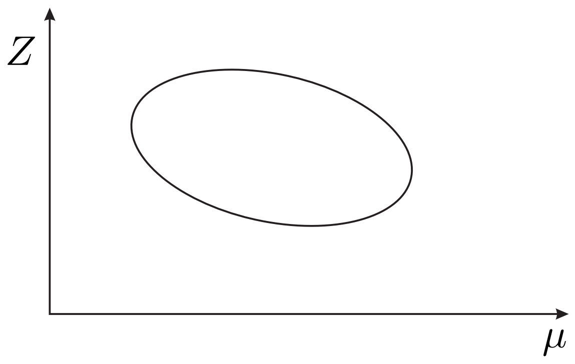
(a) isolated branches



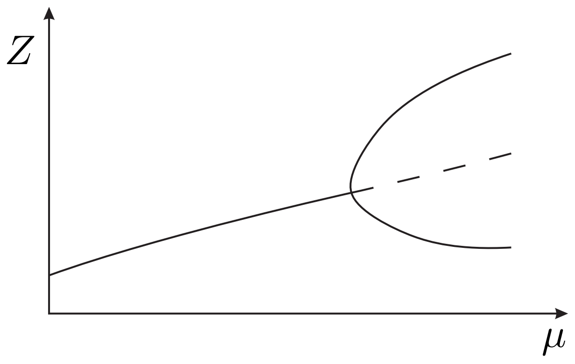
(b) singular point



(c) limit point



(d) isola



(e) pitchfork

Figure 29.7: Bifurcation Diagram: Several Common Modes of Behavior.

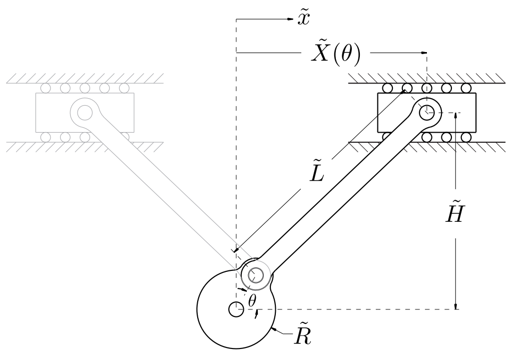


Figure 29.8: Simple mechanical linkage.

Figure 29.7(d) depicts an *isola*, an isolated interval of two solution branches with two limit points for endpoints. Figure 29.7(e) depicts a single solution branch that "bifurcates" into several solutions at a *pitchfork*; pitchfork bifurcations often correspond to nonlinear dynamics that can display either *stable* (represented by solid lines in the figure) or *unstable* (represented by dotted lines) behavior, where stability refers to the ability of the state variable $\boldsymbol{Z}$ to return to the solution when perturbed.

Note that, for all of the mentioned cases, when we reach a singular or limit point — characterized by the convergence of solution branches — the Jacobian becomes singular (non-invertible) and hence Newton breaks down unless supplemented by additional conditions.

29.4.2 A Simple Example

We can develop an intuitive understanding of these different modes of behavior and corresponding bifurcation diagrams by considering a simple example.

We wish to analyze the simple mechanical linkage shown in Figure 29.8 by finding $\widetilde{X}$ corresponding to an arbitrary θ for given (constant) $\widetilde{H}$, $\widetilde{R}$, and $\widetilde{L}$. In this example, then, θ corresponds to the earlier discussed generic parameter μ.

We can find an analytical solution for $\widetilde{X}(\theta; \widetilde{R}, \widetilde{H}, \widetilde{L})$ by solving the geometric constraint

$$(\widetilde{X} - \widetilde{R}\cos\theta)^2 + (\widetilde{H} - \widetilde{R}\sin\theta)^2 = \widetilde{L}^2 \ , \tag{29.59}$$

which defines the distance between the two joints as $\widetilde{L}$. This is clearly a nonlinear equation, owing to the quadratic term in $\tilde{x}$. We can eliminate one parameter from the equation by non-dimensionalizing with respect to $\widetilde{L}$, giving us

$$(X - R\cos\theta)^2 + (H - R\sin\theta)^2 = 1 \ , \tag{29.60}$$

where $X = \frac{\widetilde{X}}{\widetilde{L}}$, $R = \frac{\widetilde{R}}{\widetilde{L}}$, and $H = \frac{\widetilde{H}}{\widetilde{L}}$. Expanding and simplifying, we get

$$aX^2 + bX + c \equiv f(X; \theta; R, H) = 0 \ , \tag{29.61}$$

where $a = 1$, $b = -2R\cos\theta$, and $c = R^2 + H^2 - 2HR\sin\theta - 1$. A direct application of the quadratic formula then gives us the two roots

$$\begin{aligned} X_+ &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \ , \\ X_- &= \frac{-b - \sqrt{b^2 - 4ac}}{2a} \ , \end{aligned} \tag{29.62}$$

which may be real or complex.

We observe three categories for the number of solutions to our quadratic equation depending on the value of the *discriminant* $\Delta(\theta; R, H) \equiv b^2 - 4ac$. First, if $\Delta < 0$, $\sqrt{\Delta}$ is imaginary and there is no (real) solution. An example is the case in which $\widetilde{H} > \widetilde{L} + \widetilde{R}$. Second, if $\Delta = 0$, there is exactly one solution, $X = \frac{-b}{2a}$. An example is the case in which $\widetilde{H} = \widetilde{L} + \widetilde{R}$ and $\theta = \frac{\pi}{2}$. Third, if $\Delta > 0$, there are two distinct solutions, X_+ and X_-; an example is shown in Figure 29.8. Note that with our simple crank example we can obtain all the cases of Figure 29.7 except Figure 29.7(e).

We note that the case of two distinct solutions is new — our linear systems of equations (for the univariate case, $f(x) = Ax - b$) had either no solution ($A = 0$, $b \neq 0$; line parallel to x axis), exactly one solution ($A \neq 0$; line intersecting x axis), or an infinite number of solutions ($A = 0$, $b = 0$; line on x axis). Nonlinear equations, on the other hand, have no such restrictions. They can have no solution, one solution, two solutions (as in our quadratic case above), three solutions (e.g., a cubic equation) — *any* finite number of solutions, depending on the nature of the particular function $f(z)$ — or an infinite number of solutions (e.g., a sinusoidal equation). For example, if $f(z)$ is an n^{th}-order polynomial, there could be anywhere from zero to n (real) solutions, depending on the values of the $n + 1$ parameters of the polynomial.

It is important to note, for the cases in which there are two, distinct solution branches corresponding to X_+ and X_-, that, as we change the θ of the crank, it would be physically impossible to jump from one branch to the other — unless we stopped the mechanism, physically disassembled it, and then reassembled it as a mirror image of itself. Thus for the physically relevant solution we must require a continuity condition, or equivalently a constraint that requires $|X(\theta_{(i)}) - X(\theta_{(i-1)})|$ not too large or perhaps $X(\theta_{(i)})X(\theta_{(i-1)}) > 0$; here $\theta_{(i)}$ and $\theta_{(i-1)}$ are successive parameters in our family of solutions.

In the Introduction, Section 29.1, we provide an example of a linkage with two degrees of freedom. In this robot arm example the parameter $\boldsymbol{\mu}$ is given by $\boldsymbol{X}$, the desired position of the end effector.

29.4.3 Path Following: Continuation

As already indicated, as we vary our parameter μ (corresponding to θ in the crank example), we must reflect any constraint (such as, in the crank example, no "re-assembly") in our numerical approach to ensure that the solution to which we converge is indeed the "correct" one. One approach is through an appropriate choice of initial guess. Inherent in this imperative is an opportunity — we can exploit information about our previously converged solution not only to keep us on the appropriate solution branch, but also to assist continued (rapid) convergence of the Newton iteration.

We denote our previously converged solution to $f(Z; \mu) = 0$ as $Z(\mu_{(i-1)})$ (we consider here the univariate case). We wish to choose an initial guess $\widehat{Z}(\mu_{(i)})$ to converge to (a nearby root) $Z(\mu_{(i)}) = Z(\mu_{(i-1)} + \delta\mu)$ for some step $\delta\mu$ in μ. The simplest approach is to use the previously converged solution itself as our initial guess for the next step,

$$\widehat{Z}(\mu_{(i)}) = Z(\mu_{(i-1)}) \ . \qquad (29.63)$$

This is often sufficient for small changes $\delta\mu$ in μ and it is certainly the simplest approach.

We can improve our initial guess if we use our knowledge of the rate of change of $Z(\mu)$ with respect to μ to help us extrapolate, to wit

$$\widehat{Z}(\mu_{(i)}) = Z(\mu_{(i-1)}) + \frac{\mathrm{d}Z}{\mathrm{d}\mu}\delta\mu \ . \qquad (29.64)$$

We can readily calculate $\frac{\mathrm{d}Z}{\mathrm{d}\mu}$ as

$$\frac{\mathrm{d}Z}{\mathrm{d}\mu} = \frac{-\frac{\partial f}{\partial \mu}}{\frac{\partial f}{\partial z}} \ , \tag{29.65}$$

since

$$f(Z(\mu);\mu) = 0 \Rightarrow \frac{\mathrm{d}f}{\mathrm{d}\mu}(Z(\mu);\mu) \equiv \frac{\partial f}{\partial z}\frac{\mathrm{d}Z}{\mathrm{d}\mu} + \frac{\partial f}{\partial \mu}\cancelto{1}{\frac{\mathrm{d}\mu}{\mathrm{d}\mu}} = 0 \Rightarrow \frac{\mathrm{d}Z}{\mathrm{d}\mu} = \frac{-\frac{\partial f}{\partial \mu}}{\frac{\partial f}{\partial z}} \ , \tag{29.66}$$

by the chain rule.

29.4.4 Cold Start: Homotopy

In many cases, given a previous solution $Z(\mu_{(i-1)})$, we can use either of equations (29.63) or (29.64) to arrive at an educated guess $\widehat{Z}(\mu_{(i)})$ for the updated parameter $\mu_{(i)}$. If we have no previous solution, however, (e.g., $i = 1$) or our continuation techniques fail, we need some other means of generating an initial guess $\widehat{Z}(\mu_{(i)})$ that will be sufficiently good to converge to a correct solution.

A common approach to the "cold start" problem is to transform the original nonlinear problem $f(Z(\mu_{(i)});\mu_{(i)}) = 0$ into a form $\tilde{f}(Z(\mu_{(i)},t);\mu_{(i)},t) = 0$, i.e., we replace $f(Z;\mu_{(i)}) = 0$ with $\tilde{f}(\widetilde{Z};\mu_{(i)},t) = 0$. Here t is an additional, *artificial*, continuation parameter such that, *when* $t = 0$, the solution of the nonlinear problem

$$\tilde{f}(\widetilde{Z}(\mu_{(i)},t=0);\mu_{(i)},t=0) = 0 \tag{29.67}$$

is relatively simple (e.g., linear) or, perhaps, coincides with a preceding, known solution, and, *when* $t = 1$,

$$\tilde{f}(z;\mu_{(i)},t=1) = f(z;\mu_{(i)}) \tag{29.68}$$

such that $\tilde{f}(\widetilde{Z}(\mu_{(i)},t=1);\mu_{(i)},t=1) = 0$ implies $f(\widetilde{Z}(\mu_{(i)});\mu_{(i)}) = 0$ and hence $Z(\mu_{(i)})$ (the desired solution) $= \widetilde{Z}(\mu_{(i)},t=1)$.

We thus transform the "cold start" problem to a continuation problem in the artificial parameter t as t is varied from 0 to 1 with a start of its own (when $t = 0$) made significantly less "cold" by its — by construction — relative simplicity, and an end (when $t = 1$) that brings us smoothly to the solution of the original "cold start" problem.

As an example, we could replace the crank function f of (29.61) with a function $\tilde{f}(X;\theta,t;R,H) = at\,X^2 + bX + c$ such that for $t = 0$ the problem is linear and the solution readily obtained.

29.4.5 A General Path Approach: Many Parameters

We consider now an n-vector of functions

$$\boldsymbol{f}(\boldsymbol{z};\boldsymbol{\mu}) = (f_1(\boldsymbol{z};\boldsymbol{\mu}) \quad f_2(\boldsymbol{z};\boldsymbol{\mu}) \quad \ldots \quad f_n(\boldsymbol{z};\boldsymbol{\mu}))^{\mathrm{T}} \tag{29.69}$$

that depends on an n-vector of unknowns $\boldsymbol{z}$

$$\boldsymbol{z} = (z_1 \quad z_2 \quad \ldots \quad z_n)^{\mathrm{T}} \tag{29.70}$$

and a parameter ℓ-vector $\boldsymbol{\mu}$ (independent of $\boldsymbol{z}$)

$$\boldsymbol{\mu} = (\mu_1 \quad \mu_2 \quad \ldots \quad \mu_\ell)^{\mathrm{T}} \ . \tag{29.71}$$

We also introduce an inequality constraint function

$$C(\boldsymbol{Z}) = \begin{cases} 1 & \text{if constraint satisfied} \\ 0 & \text{if constraint not satisfied} \end{cases} . \tag{29.72}$$

Note this is not a constraint on $\boldsymbol{z}$ but rather a constraint on the (desired) root $\boldsymbol{Z}$. Then, given $\boldsymbol{\mu}$, we look for $\boldsymbol{Z} = (Z_1 \quad Z_2 \quad \dots \quad Z_n)^{\mathrm{T}}$ such that

$$\begin{cases} \boldsymbol{f}(\boldsymbol{Z};\boldsymbol{\mu}) = \boldsymbol{0} \\ C(\boldsymbol{Z}) = 1 \end{cases} . \tag{29.73}$$

In words, $\boldsymbol{Z}$ is a solution of n nonlinear equations in n unknowns subject which satisfies the constraint C.

Now we consider a path or "trajectory" — a sequence of p parameter vectors $\boldsymbol{\mu}_{(1)}, \dots, \boldsymbol{\mu}_{(p)}$. We wish to determine $\boldsymbol{Z}_{(i)}$, $1 \le i \le p$, such that

$$\begin{cases} \boldsymbol{f}(\boldsymbol{Z}_{(i)};\boldsymbol{\mu}_{(i)}) = \boldsymbol{0} \\ C(\boldsymbol{Z}_{(i)}) = 1 \end{cases} . \tag{29.74}$$

We assume that $\boldsymbol{Z}_{(1)}$ is known and that $\boldsymbol{Z}_{(2)}, \dots, \boldsymbol{Z}_{(p)}$ remain to be determined. We can expect, then, that as long as consecutive parameter vectors $\boldsymbol{\mu}_{(i-1)}$ and $\boldsymbol{\mu}_{(i)}$ are sufficiently close, we should be able to use our continuation techniques equations (29.63) or (29.64) to converge to a correct (i.e., satisfying $C(\boldsymbol{Z}_{(i)}) = 1$) solution $\boldsymbol{Z}_{(i)}$. If, however, $\boldsymbol{\mu}_{(i-1)}$ and $\boldsymbol{\mu}_{(i)}$ are not sufficiently close, our continuation techniques will not be sufficient (i.e. we will fail to converge to a solution at all or we will fail to converge to a solution satisfying C) and we will need to apply a — we hope — more fail-safe homotopy.

We can thus combine our continuation and homotopy frameworks into a single algorithm. One such approach is summarized in Algorithm 4. The key points of this algorithm are that (i) we are using the simple continuation approach given by equation (29.63) (i.e., using the previous solution as the initial guess for the current problem), and (ii) we are using a bisection-type homotopy that, each time Newton fails to converge to a correct solution, inserts a new point in the trajectory halfway between the previous correct solution and the failed point. The latter, in the language of homotopy, can be expressed as

$$\tilde{\boldsymbol{f}}(\boldsymbol{z};\boldsymbol{\mu}_{(i)},t) = \boldsymbol{f}(\boldsymbol{z};(1-t)\boldsymbol{\mu}_{(i-1)} + t\boldsymbol{\mu}_{(i)}) \tag{29.75}$$

with $t = 0.5$ for the inserted point.

Although there are numerous other approaches to non-convergence in addition to Algorithm 4, such as relaxed Newton — in which Newton provides the direction for the update, but then we take just some small fraction of the proposed step — Algorithm 4 is included mainly for its generality, simplicity, and apparent robustness.

Algorithm 4 General Path Following Algorithm

```
for $i = 2 \colon p$ do
  $\widehat{\boldsymbol{Z}}_{(i)} \leftarrow \boldsymbol{Z}_{(i-1)}$
  repeat
    $\boldsymbol{Z}_{(i)} \leftarrow$ {Solve $\boldsymbol{f}(\boldsymbol{z}; \boldsymbol{\mu}_{(i)}) = \boldsymbol{0}$ via Newton given initial guess $\widehat{\boldsymbol{Z}}_{(i)}$}
    if Newton does not converge OR $C(\boldsymbol{Z}_{(i)}) \neq 1$ then
      for $j = p \colon -1 \colon i$ do
        $\boldsymbol{\mu}_{(j+1)} \leftarrow \boldsymbol{\mu}_{(j)}$ {Shift path parameters by one index to accommodate insertion}
      end for
      $\boldsymbol{\mu}_{(i)} \leftarrow \frac{1}{2}(\boldsymbol{\mu}_{(i-1)} + \boldsymbol{\mu}_{(i)})$ {Insert point in path halfway between $\boldsymbol{\mu}_{(i-1)}$ and $\boldsymbol{\mu}_{(i)}$}
      $p \leftarrow p + 1$ {Increment $p$ to account for insertion}
    end if
  until Newton converges AND $C(\boldsymbol{Z}_{(i)}) = 1$
end for
```