6

Basic Numerical Methods

This chapter is devoted to the basic numerical methods. We first discuss various approximations, solution of systems, and eigenvalue problems. Then, we describe how to determine the parameters in stochastic models.

6.1 Approximations

6.1.1 Interpolation

Linear Interpolation. Suppose that the values of a function $f(x)$ are given on the grid points x_m, $m = 0, 1, \cdots, M$, where $x_0 < x_1 < \cdots < x_M$. Sometimes, we may need to find the value of the function at other points. A simple way to do this is to interpolate the function by using the known values of the function. Let f_m denote the value of the function $f(x)$ at a point x_m, $m = 0, 1, \cdots, M$. We want to approximate the value $f(x^*)$ for $x^* \in (x_m, x_{m+1})$. The simplest interpolation is to use a linear function to approximate the function $f(x)$ on the subinterval $[x_m, x_{m+1}]$. Let

$$p_1(x) = a_0 + a_1 x.$$

Using the conditions

$$p_1(x_m) = f_m, \quad p_1(x_{m+1}) = f_{m+1},$$

we find

$$a_0 = \frac{x_{m+1} f_m - x_m f_{m+1}}{x_{m+1} - x_m}, \quad a_1 = \frac{f_{m+1} - f_m}{x_{m+1} - x_m}.$$

Then, we have

$$p_1(x) = \frac{x_{m+1} - x}{x_{m+1} - x_m} f_m + \frac{x - x_m}{x_{m+1} - x_m} f_{m+1}.$$

Y.-l. Zhu et al., *Derivative Securities and Difference Methods*, Springer Finance,
DOI 10.1007/978-1-4614-7306-0_6, © Springer Science+Business Media New York 2013

Thus, we have the approximate value:

$$f(x^*) \approx p_1(x^*) = \frac{x_{m+1} - x^*}{x_{m+1} - x_m} f_m + \frac{x^* - x_m}{x_{m+1} - x_m} f_{m+1}.$$

This is called the linear interpolation. If we do the interpolation for all subintervals, then we obtain a piecewise linear function on the interval $[x_0, x_M]$.

Higher Order Interpolation. If the function data indicates that the function is smooth, then we can use a quadratic or Nth order interpolation to get a better approximation. Assume that we have obtained the values f_{m-1}, f_m, and f_{m+1}. Let

$$p_2(x) = a_0 + a_1 x + a_2 x^2.$$

Using the conditions

$$p_2(x_{m-1}) = f_{m-1}, \quad p_2(x_m) = f_m, \quad p_2(x_{m+1}) = f_{m+1},$$

we find

$$\begin{aligned} p_2(x) = {} & \frac{(x_m - x)(x_{m+1} - x)}{(x_m - x_{m-1})(x_{m+1} - x_{m-1})} f_{m-1} + \frac{(x - x_{m-1})(x_{m+1} - x)}{(x_m - x_{m-1})(x_{m+1} - x_m)} f_m \\ & + \frac{(x - x_{m-1})(x - x_m)}{(x_{m+1} - x_{m-1})(x_{m+1} - x_m)} f_{m+1}. \end{aligned}$$

Then, for any $x^* \in (x_{m-1}, x_{m+1})$, $f(x^*)$ can be approximated by $p_2(x^*)$. This is called the quadratic interpolation.

In general, if f_m, $m = i, i+1, \cdots, i+N$, are known for an integer i, then an Nth Lagrange interpolating polynomial can be obtained. For simplicity, let $i = 0$ and write down the polynomial as follows:

$$p_N(x) = \varphi_0(x) f_0 + \varphi_1(x) f_1 + \cdots + \varphi_N(x) f_N,$$

where

$$\varphi_k(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_N)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_N)}$$

for $k = 0, 1, \cdots, N$. This approximation can be used for any $x \in (x_0, x_N)$. It is clear that the linear and quadratic interpolating polynomials are the Lagrange interpolating polynomials with $N = 1$ and 2, respectively. For an Nth Lagrange interpolating polynomial, the error is given by the following theorem:

Theorem 6.1 *If x_m, $m = 0, 1, \cdots, N$, are distinct numbers and $f(x)$ is $N+1$ times continuous differentiable on $[x_0, x_N]$, then for any $x \in [x_0, x_N]$, there exists a $\xi \in [x_0, x_N]$, such that*

$$f(x) - p_N(x) = \frac{f^{(N+1)}(\xi)}{(N+1)!} (x - x_0)(x - x_1) \cdots (x - x_N).$$

Therefore, the error of linear interpolation is $O(\Delta x^2)$, and the error of quadratic interpolation is $O(\Delta x^3)$, where $\Delta x = \max_m (x_{m+1} - x_m)$.

Cubic Spline Interpolation. As we can see, linear interpolations result in piecewise linear functions on the interval $[x_0, x_M]$: the function is smooth in each subinterval $[x_m, x_{m+1}]$, continuous in $[x_0, x_M]$, but may not be smooth in $[x_0, x_M]$. For quadratic interpolations, the situation is similar. Cubic spline interpolation is the most commonly used piecewise polynomial approximation, which is a cubic polynomial on each subinterval $[x_m, x_{m+1}]$ and has a continuous second derivative on the whole interval. The cubic spline interpolation $S(x)$ satisfies the following conditions:

(A) On the subinterval $[x_m, x_{m+1}]$, $S(x) = S_m(x)$ is a cubic polynomial, $m = 0, 1, \cdots, M-1$;
(B) $S(x_m) = f_m$, $m = 0, 1, \cdots, M$;
(C) $S_m(x_m) = S_{m-1}(x_m)$, $S'_m(x_m) = S'_{m-1}(x_m)$, $S''_m(x_m) = S''_{m-1}(x_m)$, $m = 1, 2, \cdots, M-1$;
(D) $S''(x_0) = S''(x_M) = 0$, or other two conditions.

Let

$$S_m(x) = a_m + b_m(x - x_m) + c_m(x - x_m)^2 + d_m(x - x_m)^3, \quad m = 0, 1, \cdots, M-1.$$

Condition B, $m = 0, 1, \cdots, M-1$, can be written as

$$a_m = S_m(x_m) = f_m, \quad m = 0, 1, \cdots, M-1.$$

Using condition C, we get

$$\begin{cases} a_m = a_{m-1} + b_{m-1}h_{m-1} + c_{m-1}h_{m-1}^2 + d_{m-1}h_{m-1}^3, \\ b_m = b_{m-1} + 2c_{m-1}h_{m-1} + 3d_{m-1}h_{m-1}^2, \\ c_m = c_{m-1} + 3d_{m-1}h_{m-1}, \\ \qquad m = 1, 2, \cdots, M-1, \end{cases} \tag{6.1}$$

where $h_{m-1} = x_m - x_{m-1}$. Define

$$a_M = f_M$$

and

$$c_M = S''(x_M)/2.$$

Then, from the expression $S_{M-1}(x)$ and Condition B with $m = M$, we further have

$$\begin{cases} a_M = a_{M-1} + b_{M-1}h_{M-1} + c_{M-1}h_{M-1}^2 + d_{M-1}h_{M-1}^3, \\ c_M = c_{M-1} + 3d_{M-1}h_{M-1}. \end{cases} \tag{6.2}$$

Rewrite the last relations in the sets of relations (6.1) and (6.2) as

$$d_{m-1} = \frac{c_m - c_{m-1}}{3h_{m-1}}, \quad m = 1, 2, \cdots, M, \tag{6.3}$$

and the first relations in the sets of relations (6.1) and (6.2) as

$$\begin{aligned} b_{m-1} &= \frac{a_m - a_{m-1}}{h_{m-1}} - c_{m-1}h_{m-1} - d_{m-1}h_{m-1}^2 \\ &= \frac{a_m - a_{m-1}}{h_{m-1}} - c_{m-1}h_{m-1} - \frac{c_m - c_{m-1}}{3}h_{m-1}, \\ m &= 1, 2, \cdots, M. \end{aligned} \tag{6.4}$$

Substituting them into the second relation in the set of relations (6.1) yields

$$\begin{aligned} &\frac{a_{m+1} - a_m}{h_m} - c_m h_m - \frac{c_{m+1} - c_m}{3}h_m \\ &= \frac{a_m - a_{m-1}}{h_{m-1}} - c_{m-1}h_{m-1} - \frac{c_m - c_{m-1}}{3}h_{m-1} \\ &\quad +2c_{m-1}h_{m-1} + (c_m - c_{m-1})h_{m-1}, \\ &\quad m = 1, 2, \cdots, M-1, \end{aligned}$$

or

$$\begin{aligned} u_m c_{m-1} + 2c_m + v_m c_{m+1} &= \frac{1}{h_{m-1} + h_m}\left[\frac{3(a_{m+1} - a_m)}{h_m} - \frac{3(a_m - a_{m-1})}{h_{m-1}}\right], \\ m &= 1, 2, \cdots, M-1, \end{aligned}$$

where $u_m = h_{m-1}/(h_{m-1} + h_m)$ and $v_m = h_m/(h_{m-1} + h_m)$. This system is equivalent to Conditions A–C. If Condition D is $S''(x_0) = S''(x_M) = 0$, we have two other equations $c_0 = 0$ and $c_M = 0$. In this case the entire system can be written in the following matrix form:

$$A\mathbf{c} = \mathbf{h}, \tag{6.5}$$

where

$$A = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ u_1 & 2 & v_1 & 0 & \cdots & 0 \\ 0 & u_2 & 2 & v_2 & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & u_{M-1} & 2 & v_{M-1} \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_M \end{bmatrix},$$

$$\mathbf{h} = \begin{bmatrix} 0 \\ \dfrac{1}{h_0 + h_1}\left(\dfrac{3(a_2 - a_1)}{h_1} - \dfrac{3(a_1 - a_0)}{h_0}\right) \\ \vdots \\ \dfrac{1}{h_{M-2} + h_{M-1}}\left(\dfrac{3(a_M - a_{M-1})}{h_{M-1}} - \dfrac{3(a_{M-1} - a_{M-2})}{h_{M-2}}\right) \\ 0 \end{bmatrix}.$$

Solving this linear system, we obtain c_m, $m = 0, 1, \cdots, M$. Then d_m, $m = 0, 1, \cdots, M-1$, can be obtained from the set of relations (6.3) and b_m, $m = 0, 1, \cdots, M-1$, from the set of relations (6.4).

The condition $S''(x_0) = 0$ could be replaced by $S'(x_0) = f'(x_0)$ or $d_0 = 0$, namely, assuming $S_0(x) = a_0 + b_0(x - x_0) + c_0(x - x_0)^2$. At $x = x_M$, the situation is similar. If such a case appears, then the way to determine these coefficients needs to be changed slightly. Here, assuming $S'(x_0) = f'(x_0)$ and $d_{M-1} = 0$, we explain how to modify the way to determine these coefficients. Because $S_0'(x_0) = b_0$, the coefficient b_0 is known in this case, namely, $b_0 = f'(x_0)$. From

$$\begin{cases} a_1 = a_0 + b_0 h_0 + c_0 h_0^2 + d_0 h_0^3, \\ c_1 = c_0 + 3 d_0 h_0, \end{cases}$$

we eliminate d_0 and obtain

$$2c_0 + c_1 = 3\left(\frac{a_1 - a_0}{h_0^2} - \frac{b_0}{h_0}\right) = 3\left(\frac{a_1 - a_0}{h_0^2} - \frac{f'(x_0)}{h_0}\right).$$

This equation should replace the first equation in the system (6.5). From $d_{M-1} = 0$ and the second equation in the set of relations (6.2), we have

$$c_{M-1} - c_M = 0.$$

This equation should replace the last equation in the system (6.5). Solving the modified system (6.5) yields c_m, $m = 0, 1, \cdots, M$, for this case. As soon as all the c_m are obtained, d_m, $m = 0, 1, \cdots, M-1$, can be obtained from the set of relations (6.3) and b_m, $m = 0, 1, \cdots, M-1$, from the set of relations (6.4). For more about cubic spline interpolation, see books on numerical methods.

When we write a code to calculate the approximate value $f(x^*)$ by quadratic interpolation, in order to guarantee to use an interpolation, we need to find a number m such that $x^* \in [x_{m-1}, x_{m+1}]$. This can be realized by using a loop statement. If $x_m = m\Delta x, m = 0, 1, \cdots, M$, then the expression

$$m = \max\left(1, \min\left(\text{int}\left(\frac{x^*}{\Delta x} + 0.5\right), M - 1\right)\right)$$

will also always give such a number.

6.1.2 Approximation of Partial Derivatives

Finite-Difference Approximation. Here, we will discuss how derivatives of a function $u(x, t)$ at a point can be approximated by a linear combination of values of the function at adjacent points. Let $x_m = a + m\Delta x$ and $\tau^n = n\Delta\tau$, where m is an integer and n is an integer or an integer plus a half.

Using the Taylor expansion, we have[1]

$$u(x_m, \tau^{n+1}) = u(x_m, \tau^n) + \Delta\tau \frac{\partial u}{\partial \tau}(x_m, \tau^n) + \frac{\Delta\tau^2}{2} \frac{\partial^2 u}{\partial \tau^2}(x_m, \eta),$$

[1] In this book $\Delta\tau^2$ stands for $(\Delta\tau)^2$. For $\Delta\tau^3$, Δx^2, Δx^3 etc., the situation is similar.

where $\tau^n < \eta < \tau^{n+1}$. Then,

$$\frac{\partial u}{\partial \tau}(x_m, \tau^n) = \frac{u(x_m, \tau^{n+1}) - u(x_m, \tau^n)}{\Delta\tau} - \frac{\Delta\tau}{2}\frac{\partial^2 u}{\partial \tau^2}(x_m, \eta).$$

If $\Delta\tau$ is small, we have

$$\frac{\partial u}{\partial \tau}(x_m, \tau^n) \approx \frac{u(x_m, \tau^{n+1}) - u(x_m, \tau^n)}{\Delta\tau}.$$

This approximation is called the forward finite-difference approximation or the **forward difference** for $\dfrac{\partial u}{\partial \tau}$. Similarly, we can obtain the **backward difference**

$$\frac{\partial u}{\partial \tau}(x_m, \tau^n) \approx \frac{u(x_m, \tau^n) - u(x_m, \tau^{n-1})}{\Delta\tau}.$$

Both forward and backward finite-difference approximations have errors of first order in $\Delta\tau$ (first-order accurate). To obtain a second-order accurate finite-difference approximation, we use the following Taylor expansions:

$$\begin{aligned} u(x_m, \tau^{n+1}) &= u(x_m, \tau^{n+1/2}) + \frac{\Delta\tau}{2}\frac{\partial u}{\partial \tau}(x_m, \tau^{n+1/2}) \\ &\quad + \frac{\Delta\tau^2}{8}\frac{\partial^2 u}{\partial \tau^2}(x_m, \tau^{n+1/2}) + \frac{\Delta\tau^3}{48}\frac{\partial^3 u}{\partial \tau^3}(x_m, \eta_1), \\ u(x_m, \tau^n) &= u(x_m, \tau^{n+1/2}) - \frac{\Delta\tau}{2}\frac{\partial u}{\partial \tau}(x_m, \tau^{n+1/2}) \\ &\quad + \frac{\Delta\tau^2}{8}\frac{\partial^2 u}{\partial \tau^2}(x_m, \tau^{n+1/2}) - \frac{\Delta\tau^3}{48}\frac{\partial^3 u}{\partial \tau^3}(x_m, \eta_2), \end{aligned}$$

where $\tau^{n+1/2} < \eta_1 < \tau^{n+1}$ and $\tau^n < \eta_2 < \tau^{n+1/2}$. Subtracting the second equation from the first one, we get

$$\frac{\partial u}{\partial \tau}(x_m, \tau^{n+1/2}) = \frac{u(x_m, \tau^{n+1}) - u(x_m, \tau^n)}{\Delta\tau} - \frac{\Delta\tau^2}{24}\frac{\partial^3 u}{\partial \tau^3}(x_m, \eta_3),$$

where $\tau^n < \eta_3 < \tau^{n+1}$. Then, we have

$$\frac{\partial u}{\partial \tau}(x_m, \tau^{n+1/2}) \approx \frac{u(x_m, \tau^{n+1}) - u(x_m, \tau^n)}{\Delta\tau}.$$

This approximation is called the central finite-difference approximation or a **central difference** for $\dfrac{\partial u}{\partial \tau}$.

Similarly, for $\dfrac{\partial u}{\partial x}(x_m, \tau^n)$ we can have the following approximations

$$\frac{\partial u}{\partial x}(x_m,\tau^n) \approx \frac{u(x_{m+1},\tau^n)-u(x_{m-1},\tau^n)}{2\Delta x},$$
$$\frac{\partial u}{\partial x}(x_m,\tau^n) \approx \frac{u(x_m,\tau^n)-u(x_{m-1},\tau^n)}{\Delta x}$$

and

$$\frac{\partial u}{\partial x}(x_m,\tau^n) \approx \frac{u(x_{m+1},\tau^n)-u(x_m,\tau^n)}{\Delta x}.$$

The first one is second order and called the **second-order central difference** for first derivatives. The second and third approximations are first order and called the **first-order one-sided difference**. Sometimes, we also need the following **second-order one-sided differences**:

$$\frac{\partial u}{\partial x}(x_m,\tau^n) \approx \frac{3u(x_m,\tau^n)-4u(x_{m-1},\tau^n)+u(x_{m-2},\tau^n)}{2\Delta x}$$

and

$$\frac{\partial u}{\partial x}(x_m,\tau^n) \approx \frac{-3u(x_m,\tau^n)+4u(x_{m+1},\tau^n)-u(x_{m+2},\tau^n)}{2\Delta x}.$$

For the approximation of the second-order partial derivative with respect to x, we use the following Taylor expansions:

$$\begin{aligned} u(x_{m+1},\tau^n) &= u(x_m,\tau^n)+\Delta x\frac{\partial u}{\partial x}(x_m,\tau^n)+\frac{\Delta x^2}{2}\frac{\partial^2 u}{\partial x^2}(x_m,\tau^n)\\ &\quad+\frac{\Delta x^3}{6}\frac{\partial^3 u}{\partial x^3}(x_m,\tau^n)+\frac{\Delta x^4}{24}\frac{\partial^4 u}{\partial x^4}(\xi_1,\tau^n),\\ u(x_{m-1},\tau^n) &= u(x_m,\tau^n)-\Delta x\frac{\partial u}{\partial x}(x_m,\tau^n)+\frac{\Delta x^2}{2}\frac{\partial^2 u}{\partial x^2}(x_m,\tau^n)\\ &\quad-\frac{\Delta x^3}{6}\frac{\partial^3 u}{\partial x^3}(x_m,\tau^n)+\frac{\Delta x^4}{24}\frac{\partial^4 u}{\partial x^4}(\xi_2,\tau^n), \end{aligned}$$

where $x_m<\xi_1<x_{m+1}$ and $x_{m-1}<\xi_2<x_m$. Adding these two equations, we obtain

$$\frac{\partial^2 u}{\partial x^2}(x_m,\tau^n)=\frac{u(x_{m+1},\tau^n)-2u(x_m,\tau^n)+u(x_{m-1},\tau^n)}{\Delta x^2}-\frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_3,\tau^n),$$

where $x_{m-1}<\xi_3<x_{m+1}$. Thus, we have the **second-order central difference** for second derivatives:

$$\frac{\partial^2 u}{\partial x^2}(x_m,\tau^n) \approx \frac{u(x_{m+1},\tau^n)-2u(x_m,\tau^n)+u(x_{m-1},\tau^n)}{\Delta x^2}.$$

Sometimes, we also need to have an approximation to mixed second-order partial derivatives. For $\dfrac{\partial^2 u}{\partial x \partial y}$, we have

$$\frac{\partial^2 u}{\partial x \partial y}(x_m, y_l, \tau^n) \approx \frac{1}{2\Delta x}\left[\frac{u(x_{m+1}, y_{l+1}, \tau^n) - u(x_{m+1}, y_{l-1}, \tau^n)}{2\Delta y} - \frac{u(x_{m-1}, y_{l+1}, \tau^n) - u(x_{m-1}, y_{l-1}, \tau^n)}{2\Delta y}\right],$$

where $y_l = b + l\Delta y$, Δy being a small number. It is clear that this is a second-order scheme, and this formula is called the **second-order central difference** for mixed second-order partial derivatives.

Pseudo-Spectral Approximation. By using more points, we can also construct higher order finite-difference approximations for the partial derivatives. An alternative way to obtain higher order approximations for partial derivatives is to use a pseudo-spectral method. To illustrate the method, we consider the approximation to the partial derivatives with respect to x for a fixed τ. Assume that we want to find the solution u in the interval $0 \le x \le 1$. Suppose we use non-equidistant nodes. For example, we can use the following grid points

$$x_m = \frac{1}{2}\left(1 - \cos\frac{m\pi}{M}\right), \quad m = 0, 1, \cdots, M. \tag{6.6}$$

These points are in $[0, 1]$ and equal to $(1 - x_m^*)/2$, x_m^* being the extrema of the Mth order Chebyshev polynomial $T_M(x)$. Here, the Mth order Chebyshev polynomial is defined by $T_M(x) = \cos(M \cos^{-1} x)$. Assume that the solution for a fixed τ is a polynomial in x with degree M. If we require the polynomial to have a value $u(x_m)$ at $x = x_m$, then we can determine the coefficients of the polynomial, each of which is a linear combination of $u(x_i)$, $i = 0, 1, \cdots, M$. Thus, the derivatives of the polynomial at the point x_m is also a linear combination of $u(x_i)$, $i = 0, 1, \cdots, M$, with coefficients depending on x_m and x_i. Therefore,

$$\frac{\partial u}{\partial x}(x_m) = \sum_{i=0}^{M} D_{x,m,i} u(x_i). \tag{6.7}$$

This is an M**th order approximation to the derivative** with respect to x used in the pseudo-spectral method. If the grid points are given by the expression (6.6), then $D_{x,m,i}$ has the following expression:

$$D_{x,m,i} = \begin{cases} \dfrac{c_m(-1)^{m+i}}{c_i(x_m - x_i)}, & m \neq i, \\ -\dfrac{2M^2+1}{3}, & m = i = 0, \\ \dfrac{1-2x_i}{4x_i(1-x_i)}, & m = i = 1, 2, \cdots, M-1, \\ \dfrac{2M^2+1}{3}, & m = i = M, \end{cases} \tag{6.8}$$

where $c_0 = c_M = 2$ and $c_i = 1$, $i = 1, 2, \cdots, M-1$ (see [36]). Similarly, we have

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2}(x_m) &= \sum_{j=0}^{M} D_{x,m,j} \frac{\partial u}{\partial x}(x_j) \\ &= \sum_{j=0}^{M} D_{x,m,j} \left[\sum_{i=0}^{M} D_{x,j,i} u(x_i) \right] \\ &= \sum_{i=0}^{M} \left(\sum_{j=0}^{M} D_{x,m,j} D_{x,j,i} \right) u(x_i) \\ &= \sum_{i=0}^{M} D_{xx,m,i} u(x_i), \end{aligned} \tag{6.9}$$

where

$$D_{xx,m,i} = \sum_{j=0}^{M} D_{x,m,j} D_{x,j,i}. \tag{6.10}$$

When the solution is very smooth, only a small M may be needed in order to get a satisfying result. In such a case, its performance could be better than the finite-difference approximations.

6.1.3 Approximate Integration

Trapezoidal Rule. The approximation of the integral

$$\int_a^b f(x)dx$$

is needed in the numerical solution of integro-differential equations and sometimes in the numerical solution of partial differential equations. The simplest method for the approximation is called the **trapezoidal rule.** Let $h = (b-a)/M$, and $x_m = a + mh$, $m = 0, 1, \cdots, M$. In the subinterval $[x_m, x_{m+1}]$, we use the linear function

$$p_1(x) = \frac{x_{m+1} - x}{h} f(x_m) + \frac{x - x_m}{h} f(x_{m+1})$$

to approximate $f(x)$. Thus,

$$\begin{aligned}\int_{x_m}^{x_{m+1}} f(x)dx &\approx \frac{1}{h} \int_{x_m}^{x_{m+1}} [(x_{m+1} - x)f(x_m) + (x - x_m)f(x_{m+1})]dx \\ &= \frac{h}{2} \left[f(x_m) + f(x_{m+1})\right].\end{aligned}$$

Using this for all subintervals, we obtain

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{m=0}^{M-1} \int_{x_m}^{x_{m+1}} f(x)dx \\ &\approx \sum_{m=0}^{M-1} \frac{h}{2} \left[f(x_m) + f(x_{m+1})\right] \\ &= \frac{h}{2} \left[f(a) + 2 \sum_{m=1}^{M-1} f(x_m) + f(b)\right].\end{aligned}$$

The error of the trapezoidal rule is

$$-\frac{(b-a)h^2}{12} f''(\xi),$$

where $\xi \in (a, b)$.

Simpson's Rule. Simpson's rule is a better approximation for the integral by using the quadratic interpolation polynomial. In the subinterval $[x_{m-1}, x_{m+1}]$, we use

$$\begin{aligned}p_2(x) = &\frac{(x_m - x)(x_{m+1} - x)}{(x_m - x_{m-1})(x_{m+1} - x_{m-1})} f(x_{m-1}) \\ &+ \frac{(x - x_{m-1})(x_{m+1} - x)}{(x_m - x_{m-1})(x_{m+1} - x_m)} f(x_m) \\ &+ \frac{(x - x_{m-1})(x - x_m)}{(x_{m+1} - x_{m-1})(x_{m+1} - x_m)} f(x_{m+1})\end{aligned}$$

to approximate $f(x)$. Thus

$$\begin{aligned}\int_{x_{m-1}}^{x_{m+1}} f(x)dx &\approx \frac{1}{2h^2} \int_{x_{m-1}}^{x_{m+1}} [(x_m - x)(x_{m+1} - x)f(x_{m-1}) \\ &\qquad +2(x - x_{m-1})(x_{m+1} - x)f(x_m) \\ &\qquad +(x - x_{m-1})(x - x_m)f(x_{m+1})]dx \\ &= \frac{h}{3}[f(x_{m-1}) + 4f(x_m) + f(x_{m+1})].\end{aligned}$$

Suppose that M is an even number and using this for all subintervals, we obtain

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 2\sum_{m=1}^{M/2-1} f(x_{2m}) + 4\sum_{m=1}^{M/2} f(x_{2m-1}) + f(b)\right].$$

The error of the Simpson's rule is

$$-\frac{(b-a)h^4}{180} f^{(4)}(\xi),$$

where $\xi \in (a, b)$.

6.1.4 Least Squares Approximation

In Sect. 6.1.1 we discussed various interpolations. In those cases, all the given points (x_m, f_m) are on the interpolation function. Here, we will discuss how to find an approximate function satisfying the following two conditions:

(A) The number of parameters in the function is less than the number of given points.
(B) Let the function have the "best fit" to those given points (x_m, f_m) in some sense.

Let x_m, $m = 0, 1, \cdots, M$, be distinct, and let $M + 1$ points (x_m, f_m) be given. We want to find a product of a given function $g(x)$ and a polynomial of degree $N < M$

$$g(x)\sum_{n=0}^{N} a_n x^n$$

such that the value of the total least squares error

$$\sum_{m=0}^{M} b_m \left[f_m - g(x_m)\sum_{n=0}^{N} a_n x_m^n\right]^2$$

has a minimum, where b_m, $m = 0, 1, \cdots, M$, are given positive numbers called the weights. In order to minimize the least squares error, the necessary conditions are

$$\frac{\partial}{\partial a_i}\left\{\sum_{m=0}^{M} b_m \left[f_m - g(x_m)\sum_{n=0}^{N} a_n x_m^n\right]^2\right\}$$
$$= -2\sum_{m=0}^{M} b_m \left[f_m - g(x_m)\sum_{n=0}^{N} a_n x_m^n\right] g(x_m) x_m^i = 0,$$
$$i = 0, 1, \cdots, N.$$

This system can be written as

$$\sum_{n=0}^{N}\left[\sum_{m=0}^{M} b_m g^2(x_m) x_m^{n+i}\right] a_n = \sum_{m=0}^{M} b_m f_m g(x_m) x_m^i, \qquad (6.11)$$
$$i = 0, 1, \cdots, N.$$

It is a linear system for $a_0, a_1, \cdots, a_N$, which is usually known as the system of normal equations. When x_0, x_1, $\cdots$, x_M are distinct, we can find $a_0, a_1, \cdots, a_N$ without any difficulty. This method is usually referred to as the least squares method with weights.

6.2 Solution of Systems and Eigenvalue Problems

6.2.1 LU Decomposition of Linear Systems

One efficient method for solving linear systems is the LU decomposition, which decomposes the matrix $\mathbf{A}$ into a product of a unit lower triangular matrix[2] $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$, i.e., $\mathbf{A} = \mathbf{LU}$. In this subsection, we will give the details of the method for linear tridiagonal systems. For more complicated systems, the procedure is similar.

Consider a general linear tridiagonal system

$$\mathbf{Ax} = \mathbf{q}, \qquad (6.12)$$

where

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & 0 \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ & 0 & & a_{m-1} & b_{m-1} & c_{m-1} \\ & & & & a_m & b_m \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix}.$$

Let us discuss how to find the solution of this system. The first equation of the system is

$$b_1 x_1 + c_1 x_2 = q_1.$$

Let

$$u_1 = b_1, \quad y_1 = q_1,$$

the equation above can be written as

$$u_1 x_1 + c_1 x_2 = y_1. \qquad (6.13)$$

[2] A matrix $\mathbf{A}$ is called a unit lower triangular matrix if $a_{ii} = 1$ and $a_{ij} = 0$ for $i < j$. If $a_{ij} = 0$ for $i > j$, then the matrix $\mathbf{A}$ is called an upper triangular matrix.

Now suppose we have a relation in the form

$$u_{i-1}x_{i-1} + c_{i-1}x_i = y_{i-1}.$$

We put this relation and the ith equation of the system together and obtain

$$\begin{cases} u_{i-1}x_{i-1} + c_{i-1}x_i = y_{i-1}, \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} = q_i. \end{cases}$$

Subtracting the first relation multiplied by a_i/u_{i-1} from the second equation, we can eliminate x_{i-1} and have another relation in the same form:

$$\left(b_i - c_{i-1}\frac{a_i}{u_{i-1}}\right)x_i + c_i x_{i+1} = q_i - y_{i-1}\frac{a_i}{u_{i-1}}$$

or

$$u_i x_i + c_i x_{i+1} = y_i,$$

where

$$u_i = b_i - \frac{c_{i-1}a_i}{u_{i-1}},$$
$$y_i = q_i - \frac{y_{i-1}a_i}{u_{i-1}}.$$

Because we have Eq. (6.13) that is in this form, this procedure can be done for $i = 2, 3, \cdots, m$ successively and generates

$$u_i x_i + c_i x_{i+1} = y_i, \quad i = 2, 3, \cdots, m-1 \tag{6.14}$$

and

$$u_m x_m = y_m. \tag{6.15}$$

Because in the last equation of the system x_{m+1} does not appear, which means $c_m = 0$, the last relation is in the form Eq. (6.15) instead of the set of equations (6.14). This procedure can be called elimination or forward substitution.

When we obtain Eq. (6.15), we can have

$$x_m = \frac{y_m}{u_m}.$$

Furthermore, from Eq. (6.13) and the set of equations (6.14) we can get

$$x_i = \frac{y_i - c_i x_{i+1}}{u_i}, \quad i = m-1, \cdots, 1$$

successively. This procedure is called back substitution. Through these two procedures, we can obtain the solution of this system.

The elimination procedure can be written in matrix form

$$\mathbf{L}_{m-1}\cdots\mathbf{L}_2\mathbf{L}_1\mathbf{A}\mathbf{x} = \mathbf{U}\mathbf{x} = \mathbf{L}_{m-1}\cdots\mathbf{L}_1\mathbf{q} = \mathbf{y}, \tag{6.16}$$

where

$$\mathbf{L}_i = \begin{bmatrix} 1 & 0 & & \cdots & & 0 \\ 0 & 1 & \ddots & & & \\ & \ddots & \ddots & \ddots & & \\ \vdots & & 0 & 1 & \ddots & \vdots \\ & & & -l_i & 1 & \ddots \\ & & & & 0 & \ddots & \ddots \\ & & & & & \ddots & \ddots & 0 \\ 0 & & & \cdots & & & 0 & 1 \end{bmatrix},$$

$$\mathbf{U} = \begin{bmatrix} u_1 & c_1 & 0 & \cdots & 0 \\ 0 & u_2 & c_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & u_{m-1} & c_{m-1} \\ 0 & \cdots & \cdots & 0 & u_m \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix},$$

l_i equalling a_{i+1}/u_i and being in the ith column and the $(i+1)$th row of $\mathbf{L}_i$, $i = 1, 2, \cdots, m-1$.

Let

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_1 & 1 & 0 & & \vdots \\ 0 & l_2 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & l_{m-1} & 1 \end{bmatrix}.$$

It is easy to see

$$\mathbf{L}_{m-1}\mathbf{L}_{m-2}\cdots\mathbf{L}_1\mathbf{L} = \mathbf{I}.$$

Thus, we have

$$\mathbf{L}_{m-1}\mathbf{L}_{m-2}\cdots\mathbf{L}_1 = \mathbf{L}^{-1}$$

and Eq. (6.16) can be written as

$$\mathbf{L}^{-1}\mathbf{A}\mathbf{x} = \mathbf{U}\mathbf{x} = \mathbf{L}^{-1}\mathbf{q} = \mathbf{y}.$$

Consequently,

$$\mathbf{A}\mathbf{x} = \mathbf{L}\mathbf{U}\mathbf{x} = \mathbf{q}.$$

This means that $\mathbf{A}$ can be decomposed into a unit lower triangular matrix $\mathbf{L}$ multiplied by an upper triangular matrix $\mathbf{U}$. The procedure of solving the system is as follows. We first multiply the equation by $\mathbf{L}^{-1}$ so that the equation becomes $\mathbf{U}\mathbf{x} = \mathbf{L}^{-1}\mathbf{q} = \mathbf{y}$ and then solve $\mathbf{U}\mathbf{x} = \mathbf{y}$ to get $\mathbf{x} = \mathbf{U}^{-1}\mathbf{y}$. Because these two procedures are easy to perform, the method is quite popular.

6.2.2 Iteration Methods for Linear Systems

An alternative to LU decomposition is iteration. Iteration methods are especially effective for large systems with sparse coefficient matrices. Consider the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{q}.$$

$\mathbf{A}$ may be decomposed as

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$$

where

$$\mathbf{L} = \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ a_{2,1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,m-1} & 0 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,m} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & a_{m-1,m} \\ 0 & \cdots & \cdots & 0 \end{bmatrix},$$

and $\mathbf{D} = \text{diag}\{a_{1,1}, a_{2,2}, \cdots, a_{m,m}\}$. Then, the linear system can be rewritten as the following system,

$$\mathbf{x} = \mathbf{D}^{-1}\left[\mathbf{q} - (\mathbf{L} + \mathbf{U})\mathbf{x}\right],$$

where we assume that $\mathbf{D}$ is invertible.

Jacobi Iteration. A simple way to find the solution is to use the following iteration:

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{q} - (\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)}\right], \quad k = 0, 1, \cdots,$$

or in component form

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{a_{1,1}}[q_1 - (a_{1,2}x_2^{(k)} + \cdots + a_{1,m}x_m^{(k)})], \\ x_2^{(k+1)} &= \frac{1}{a_{2,2}}[q_2 - (a_{2,1}x_1^{(k)} + a_{2,3}x_3^{(k)} + \cdots + a_{2,m}x_m^{(k)})], \\ &\vdots \\ x_m^{(k+1)} &= \frac{1}{a_{m,m}}[q_m - (a_{m,1}x_1^{(k)} + \cdots + a_{m,m-1}x_{m-1}^{(k)})]. \end{aligned}$$

It is clear that in order to implement this iteration, an initial guess $\mathbf{x}^{(0)}$ should be given. This method is called the Jacobi iteration.

Gauss–Seidel Iteration. In the Jacobi iteration, at the iteration step for $x_i^{(k+1)}$, all the solutions $x_1^{(k+1)}, \cdots, x_{i-1}^{(k+1)}$ have been obtained. Therefore, new variables can be used in the iteration, namely, we can have the following iteration:

$$\begin{aligned}
x_1^{(k+1)} &= \frac{1}{a_{1,1}}[q_1 - (a_{1,2}x_2^{(k)} + \cdots + a_{1,m}x_m^{(k)})],\\
x_2^{(k+1)} &= \frac{1}{a_{2,2}}[q_2 - (a_{2,1}x_1^{(k+1)} + a_{2,3}x_3^{(k)} + \cdots + a_{2,m}x_m^{(k)})],\\
x_3^{(k+1)} &= \frac{1}{a_{3,3}}[q_3 - (a_{3,1}x_1^{(k+1)} + a_{3,2}x_2^{(k+1)} + a_{3,4}x_4^{(k)} + \cdots + a_{3,m}x_m^{(k)})],\\
&\vdots\\
x_m^{(k+1)} &= \frac{1}{a_{m,m}}[q_m - (a_{m,1}x_1^{(k+1)} + \cdots + a_{m,m-1}x_{m-1}^{(k+1)})]
\end{aligned}$$

or in matrix form

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{q} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right].$$

This method is called the Gauss–Seidel iteration.

SOR (Successive Over Relaxation). The Gauss–Seidel iteration can be modified in the following way: Take a combination of the previous value of $\mathbf{x}$ and the current update (from the Gauss–Seidel method) as the next approximation:

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\mathbf{D}^{-1}\left[\mathbf{q} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right],$$

or in component form

$$\begin{aligned}
x_1^{(k+1)} &= (1-\omega)x_1^{(k)} + \frac{\omega}{a_{1,1}}[q_1 - (a_{1,2}x_2^{(k)} + \cdots + a_{1,m}x_m^{(k)})],\\
x_2^{(k+1)} &= (1-\omega)x_2^{(k)} + \frac{\omega}{a_{2,2}}[q_2 - (a_{2,1}x_1^{(k+1)} + a_{2,3}x_3^{(k)} + \cdots + a_{2,m}x_m^{(k)})],\\
x_3^{(k+1)} &= (1-\omega)x_3^{(k)} + \frac{\omega}{a_{3,3}}[q_3 - (a_{3,1}x_1^{(k+1)} + a_{3,2}x_2^{(k+1)} + a_{3,4}x_4^{(k)} + \cdots +\\
&\qquad\qquad a_{3,m}x_m^{(k)})],\\
&\vdots\\
x_m^{(k+1)} &= (1-\omega)x_m^{(k)} + \frac{\omega}{a_{m,m}}[q_m - (a_{m,1}x_1^{(k+1)} + \cdots + a_{m,m-1}x_{m-1}^{(k+1)})].
\end{aligned}$$

Here, ω is a real number. This method usually is called the method of successive over relaxation (SOR). When $\omega = 1$, it is the Gauss–Seidel iteration. The parameter ω should be chosen so that the method will converge and work better than the Gauss–Seidel iteration. The following result has been proved:

Theorem 6.2 *If* $\mathbf{A}$ *is a symmetric positive definite matrix and* $0 < \omega < 2$*, then the method of successive over relaxation will converge for any initial vector* $\mathbf{x}$*.*

Practical computation shows that this method also works for some nonsymmetric linear systems if ω is chosen properly. For many cases, this method gives faster convergence than the Gauss–Seidel iteration if $\omega \in (1, 2)$. We would like to point out that in the books by Golub and Loan [35] and Saad [71], there are some other iteration methods that can also be used for solving linear systems in Chaps. 8–10. Interested readers are referred to these books.

6.2.3 Iteration Methods for Nonlinear Systems

In the numerical solution of partial differential equations, the resulting algebraic systems are sometimes nonlinear. In this section, we discuss three iteration methods for the nonlinear systems.

Newton's Method. Consider the following nonlinear system,

$$\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0, \\ f_2(x_1, x_2, \cdots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \cdots, x_n) &= 0. \end{aligned}$$

Let

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{bmatrix}.$$

Then, the nonlinear system has the form

$$\mathbf{f}(\mathbf{x}) = 0.$$

Suppose $\mathbf{x}^{(0)} = [x_1^0, x_2^0, \cdots, x_n^0]^T$ is a good initial guess to the true solution $\mathbf{x}^* = [x_1^*, x_2^*, \cdots, x_n^*]^T$, i.e.,

$$\delta\mathbf{x} = \mathbf{x}^* - \mathbf{x}^{(0)} = [\delta x_1, \delta x_2, \cdots, \delta x_n]^T$$

is small in norm. Then, for $i = 1, 2, \cdots, n$

$$\begin{aligned} 0 = f_i(x_1^*, x_2^*, \cdots, x_n^*) &= f_i(x_1^0 + \delta x_1, x_2^0 + \delta x_2, \cdots, x_n^0 + \delta x_n) \\ &\approx f_i(x_1^0, x_2^0, \cdots, x_n^0) + \sum_{k=1}^{n} \frac{\partial f_i(x_1^0, x_2^0, \cdots, x_n^0)}{\partial x_k} \delta x_k. \end{aligned}$$

In matrix form, we have

$$\begin{bmatrix} \dfrac{\partial f_1(\mathbf{x}^{(0)})}{\partial x_1} & \dfrac{\partial f_1(\mathbf{x}^{(0)})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\mathbf{x}^{(0)})}{\partial x_n} \\ \dfrac{\partial f_2(\mathbf{x}^{(0)})}{\partial x_1} & \dfrac{\partial f_2(\mathbf{x}^{(0)})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\mathbf{x}^{(0)})}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial f_n(\mathbf{x}^{(0)})}{\partial x_1} & \dfrac{\partial f_n(\mathbf{x}^{(0)})}{\partial x_2} & \cdots & \dfrac{\partial f_n(\mathbf{x}^{(0)})}{\partial x_n} \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \\ \delta x_n \end{bmatrix} + \begin{bmatrix} f_1(\mathbf{x}^{(0)}) \\ f_2(\mathbf{x}^{(0)}) \\ \vdots \\ f_n(\mathbf{x}^{(0)}) \end{bmatrix} \approx 0,$$

or

$$\mathbf{J_f}(\mathbf{x}^{(0)}) \cdot \delta\mathbf{x} + \mathbf{f}(\mathbf{x}^{(0)}) \approx 0,$$

where $\mathbf{J_f}(\mathbf{x}^{(0)})$ denotes the above Jacobian matrix. Solving for $\delta\mathbf{x}$ we get

$$\delta\mathbf{x} \approx -[\mathbf{J_f}(\mathbf{x}^{(0)})]^{-1}\mathbf{f}(\mathbf{x}^{(0)})$$

or

$$\mathbf{x}^* \approx \mathbf{x}^{(0)} - [\mathbf{J_f}(\mathbf{x}^{(0)})]^{-1}\mathbf{f}(\mathbf{x}^{(0)}).$$

This means that the vector

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - [\mathbf{J_f}(\mathbf{x}^{(0)})]^{-1}\mathbf{f}(\mathbf{x}^{(0)})$$

will be a better approximation to the solution $\mathbf{x}^*$. In general, suppose $\mathbf{x}^{(k)}$ has been obtained, then

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\mathbf{J_f}(\mathbf{x}^{(k)})]^{-1}\mathbf{f}(\mathbf{x}^{(k)}). \tag{6.17}$$

When $n = 1$, it is an equation, not a system:

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})/f'(x^{(k)}). \tag{6.18}$$

This iteration method is called Newton's method. Because finding an inverse of a matrix is time consuming, in the real computation, Newton's method has the form

$$\begin{cases} \mathbf{J_f}(\mathbf{x}^{(k)})\mathbf{y} = -\mathbf{f}(\mathbf{x}^{(k)}), \\ \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{y}. \end{cases}$$

Newton's method converges locally with second order. More precisely, it can be proved that the following result holds.

Theorem 6.3 *Let $\mathbf{x}^*$ be a solution of $\mathbf{f}(\mathbf{x}) = 0$. Assume that $\mathbf{J_f}(\mathbf{x}^*)$ is not singular, and that $f_i(\mathbf{x})$ has continuous second-order partial derivatives near $\mathbf{x}^*$. Then, if $\mathbf{x}^{(0)}$ is close enough to $\mathbf{x}^*$, Newton's method converges and*

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\|_\infty \le C\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|_\infty^2.$$

Generalized Secant Method. One of weaknesses of Newton's method for solving nonlinear systems is that the Jacobian matrix must be computed at each iteration. The Jacobian matrix associated with a system $\mathbf{f}(\mathbf{x}) = 0$ requires n^2 partial derivatives to be evaluated. In many situations, the exact evaluation of the partial derivatives is inconvenient. This difficulty can be overcome by using finite-difference approximations to the partial derivatives. For example,

$$\begin{aligned}&\frac{\partial f_i(x_1, x_2, \cdots, x_n)}{\partial x_k}\\ &\approx \frac{1}{\Delta x_k}[f_i(x_1, \cdots, x_k + \Delta x_k, \cdots, x_n) - f_i(x_1, \cdots, x_k, \cdots, x_n)],\\ &\qquad\qquad k = 1, 2, \cdots, n,\end{aligned}$$

where Δx_k is small in absolute value. This approximation, however, still requires at least n^2 function evaluations to be performed in order to approximate the Jacobian and does not decrease the amount of calculations. Actually, if we have $\mathbf{f}(\mathbf{x})$ at $n+1$ points, then we usually can have an approximate Jacobian at some point. Suppose that we have $\mathbf{x}^{(l)}$ and $\mathbf{f}(\mathbf{x}^{(l)})$, $l = k-n, k-n+1, \cdots, k$. Because

$$\begin{aligned}&\left[\mathbf{f}(\mathbf{x}^{(k-n)}) - \mathbf{f}(\mathbf{x}^{(k)}), \mathbf{f}(\mathbf{x}^{(k-n+1)}) - \mathbf{f}(\mathbf{x}^{(k)}), \cdots, \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{f}(\mathbf{x}^{(k)})\right]\\ &\approx \mathbf{J}_{\mathbf{f}}(\mathbf{x}^{(k)}) \left[\mathbf{x}^{(k-n)} - \mathbf{x}^{(k)}, \mathbf{x}^{(k-n+1)} - \mathbf{x}^{(k)}, \cdots, \mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\right],\end{aligned}$$

we have

$$\begin{aligned}&\mathbf{J}_{\mathbf{f}}(\mathbf{x}^{(k)})\\ &\approx \left[\mathbf{f}(\mathbf{x}^{(k-n)}) - \mathbf{f}(\mathbf{x}^{(k)}), \mathbf{f}(\mathbf{x}^{(k-n+1)}) - \mathbf{f}(\mathbf{x}^{(k)}), \cdots, \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{f}(\mathbf{x}^{(k)})\right]\\ &\quad \times \left[\mathbf{x}^{(k-n)} - \mathbf{x}^{(k)}, \mathbf{x}^{(k-n+1)} - \mathbf{x}^{(k)}, \cdots, \mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\right]^{-1}.\end{aligned}$$

Therefore, Newton's method can be modified to

$$\begin{aligned}\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} &- \left[\mathbf{x}^{(k-n)} - \mathbf{x}^{(k)}, \mathbf{x}^{(k-n+1)} - \mathbf{x}^{(k)}, \cdots, \mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\right]\\ &\times \left[\mathbf{f}(\mathbf{x}^{(k-n)}) - \mathbf{f}(\mathbf{x}^{(k)}), \mathbf{f}(\mathbf{x}^{(k-n+1)}) - \mathbf{f}(\mathbf{x}^{(k)}),\right.\\ &\left.\cdots, \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{f}(\mathbf{x}^{(k)})\right]^{-1} \mathbf{f}(\mathbf{x}^{(k)}).\end{aligned} \tag{6.19}$$

Consequently, if we have $n+1$ guesses $\mathbf{x}^{(l)}$, $l = 0, 1, \cdots, n$, and the values of the function $\mathbf{f}(\mathbf{x})$ at these points, then we can do the iteration (6.19) for $k = n, n+1, \cdots$ and at each iteration we spend very little time to calculate a Jacobian. Of course, it needs to be guaranteed that the matrix

$$\left[\mathbf{f}(\mathbf{x}^{(k-n)}) - \mathbf{f}(\mathbf{x}^{(k)}), \cdots, \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{f}(\mathbf{x}^{(k)})\right]$$

is invertible. If during the iteration this matrix is not invertible, we need to find the guess $\mathbf{x}^{(k+1)}$ that is close to $\mathbf{x}^{(k)}$ in another way, for example, by changing a component of $\mathbf{x}^{(k)}$ a little bit. In practice, it happens very seldom.

If $n = 1$, then the vectors $\mathbf{x}$ and $\mathbf{f}$ become scalars x and f and the iteration (6.19) becomes

$$x^{(k+1)} = x^{(k)} - \frac{(x^{(k-1)} - x^{(k)})f(x^{(k)})}{f(x^{(k-1)}) - f(x^{(k)})}. \tag{6.20}$$

Thus, if we have two initial guesses $x^{(0)}$ and $x^{(1)}$, we can do this iteration starting from $k = 1$. This method is called the secant method, and the iteration (6.19) is referred to as the generalized secant method. Under some conditions, for the iteration (6.19) we can prove that the following relation holds:

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\|_\infty \le C\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|_\infty^2 + C \sup_{1\le l\le n} \left\|\mathbf{x}^{(k-l)} - \mathbf{x}^{(k)}\right\|_\infty \left\|\mathbf{f}(\mathbf{x}^{(k)})\right\|_\infty$$

for $k = n, n+1, \cdots$, where C is a constant (see [97]).

Bisection Method and Modified Secant Method. Consider the case $n = 1$ and suppose $x_1^{(k-1)}$ and $x_2^{(k-1)}$ be a pair of guess for the $(k-1)$th iteration with the property $f\left(x_1^{(k-1)}\right) \cdot f\left(x_2^{(k-1)}\right) < 0$. Set $\bar{x}^{(k)} = \frac{1}{2}\left(x_1^{(k-1)} + x_2^{(k-1)}\right)$. If $f\left(\bar{x}^{(k)}\right) \cdot f\left(x_1^{(k-1)}\right) > 0$, then let $x_1^{(k)} = \bar{x}^{(k)}$ and $x_2^{(k)} = x_2^{(k-1)}$; otherwise, let $x_1^{(k)} = x_1^{(k-1)}$ and $x_2^{(k)} = \bar{x}^{(k)}$. Because $\bar{x}^{(k)}$ always replaces the component $x_i^{(k-1)}$ with the condition $f\left(\bar{x}^{(k)}\right) \cdot f\left(x_i^{(k-1)}\right) > 0$, $i = 1$ or 2, $f\left(x_1^{(k)}\right) \cdot f\left(x_2^{(k)}\right) < 0$ still holds. It is clear that

$$\left|x_2^{(k)} - x_1^{(k)}\right| = \frac{1}{2}\left|x_2^{(k-1)} - x_1^{(k-1)}\right|.$$

Thus the method is always convergent. For the secant method, if $f\left(x_1^{(k-1)}\right) \cdot f\left(x_2^{(k-1)}\right) < 0$ holds, then we can make a modification on choosing a pair of guess, so that $f\left(x_1^{(k)}\right) \cdot f\left(x_2^{(k)}\right) < 0$. In this way the convergence of the modified secant method is also guaranteed.

Broyden's Method. There are some other ways to avoid calculating the Jacobian for each iteration except for the first iteration. Another weakness of Newton's method is that an $n \times n$ linear system has to be solved at each iteration, which usually requires $O(n^3)$ arithmetic calculations. Here, we introduce Broyden's method, which avoids calculating the Jacobian at each iteration and reduces the number of arithmetic calculations to $O(n^2)$ at each iteration if we get the inverse of the matrix for the first iteration.

Suppose that an initial approximation $\mathbf{x}^{(0)}$ is given, and $\mathbf{x}^{(1)}$ is computed by Newton's method

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - [\mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)})]^{-1}\mathbf{f}(\mathbf{x}^{(0)}).$$

In order to get $\mathbf{x}^{(2)}$, we replace the matrix $\mathbf{J}_\mathbf{f}(\mathbf{x}^{(1)})$ in Newton's method by a matrix $\mathbf{A}_1$ satisfying

$$\mathbf{A}_1(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}) = \mathbf{f}(\mathbf{x}^{(1)}) - \mathbf{f}(\mathbf{x}^{(0)})$$

and

$$\mathbf{A}_1\mathbf{z} = \mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)})\mathbf{z} \quad \text{whenever} \quad (\mathbf{x}^{(1)} - \mathbf{x}^{(0)})^T\mathbf{z} = 0.$$

From these conditions, it can be proved that

$$\mathbf{A}_1 = \mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)}) + \frac{\mathbf{f}(\mathbf{x}^{(1)}) - \mathbf{f}(\mathbf{x}^{(0)}) - \mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)})(\mathbf{x}^{(1)} - \mathbf{x}^{(0)})}{\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|_2^2}(\mathbf{x}^{(1)} - \mathbf{x}^{(0)})^T.$$

Using this matrix in place of $\mathbf{J}_\mathbf{f}(\mathbf{x}^{(1)})$, we have

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \mathbf{A}_1^{-1}\mathbf{f}(\mathbf{x}^{(1)}).$$

In general, suppose we have $\mathbf{x}^{(i-1)}$, $\mathbf{x}^{(i)}$ and $\mathbf{A}_{i-1}$, then we can have $\mathbf{x}^{(i+1)}$ by

$$\mathbf{A}_i = \mathbf{A}_{i-1} + \frac{\mathbf{y}^{(i)} - \mathbf{A}_{i-1}\mathbf{s}^{(i)}}{\left\|\mathbf{s}^{(i)}\right\|_2^2}(\mathbf{s}^{(i)})^T,$$

and

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \mathbf{A}_i^{-1}\mathbf{f}(\mathbf{x}^{(i)}),$$

where $\mathbf{y}^{(i)} = \mathbf{f}(\mathbf{x}^{(i)}) - \mathbf{f}(\mathbf{x}^{(i-1)})$ and $\mathbf{s}^{(i)} = \mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}$. However, at each iteration step, the linear system

$$\mathbf{A}_i\mathbf{s}^{(i+1)} = -\mathbf{f}(\mathbf{x}^{(i)})$$

still needs to be solved. To further improve the method, we need the following theorem.

Theorem 6.4 *If $\mathbf{A} \in \mathbb{R}^{n\times n}$ is nonsingular, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, and $\mathbf{y}^T\mathbf{A}^{-1}\mathbf{x} \neq -1$, then $\mathbf{A} + \mathbf{x}\mathbf{y}^T$ is also nonsingular, moreover,*

$$(\mathbf{A} + \mathbf{x}\mathbf{y}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{x}\mathbf{y}^T\mathbf{A}^{-1}}{1 + \mathbf{y}^T\mathbf{A}^{-1}\mathbf{x}}.$$

This theorem suggests a simple way to find the inverse of $\mathbf{A}_i$. By setting

$$\begin{aligned}
&\mathbf{A} = \mathbf{A}_{i-1},\\
&\mathbf{x} = \frac{\mathbf{y}^{(i)} - \mathbf{A}_{i-1}\mathbf{s}^{(i)}}{\left\|\mathbf{s}^{(i)}\right\|_2^2},\\
&\mathbf{y} = \mathbf{s}^{(i)},\\
&\mathbf{A} + \mathbf{x}\mathbf{y}^T = \mathbf{A}_i
\end{aligned}$$

in the above theorem, we have

$$\begin{aligned}\mathbf{A}_i^{-1} &= \mathbf{A}_{i-1}^{-1} - \frac{\mathbf{A}_{i-1}^{-1}\left(\dfrac{\mathbf{y}^{(i)}-\mathbf{A}_{i-1}\mathbf{s}^{(i)}}{\left\|\mathbf{s}^{(i)}\right\|_2^2}(\mathbf{s}^{(i)})^T\right)\mathbf{A}_{i-1}^{-1}}{1+(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}\left(\dfrac{\mathbf{y}^{(i)}-\mathbf{A}_{i-1}\mathbf{s}^{(i)}}{\left\|\mathbf{s}^{(i)}\right\|_2^2}\right)} \\ &= \mathbf{A}_{i-1}^{-1} - \frac{(\mathbf{A}_{i-1}^{-1}\mathbf{y}^{(i)}-\mathbf{s}^{(i)})(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}}{\left\|\mathbf{s}^{(i)}\right\|_2^2+(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}\mathbf{y}^{(i)}-\left\|\mathbf{s}^{(i)}\right\|_2^2} \\ &= \mathbf{A}_{i-1}^{-1} + \frac{(\mathbf{s}^{(i)}-\mathbf{A}_{i-1}^{-1}\mathbf{y}^{(i)})(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}}{(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}\mathbf{y}^{(i)}}.\end{aligned}$$

This computation requires only $O(n^2)$ arithmetic calculations because it involves only matrix-vector multiplications. Therefore, we have the following Broyden's method:

- Given initial guess $\mathbf{x}^{(0)}$, compute $\mathbf{A}_0^{-1} = [\mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)})]^{-1}$ and $\mathbf{x}^{(1)}$.
- For $i = 1, 2, \cdots$, do the following:

$$\begin{cases}\mathbf{y}^{(i)} = \mathbf{f}(\mathbf{x}^{(i)}) - \mathbf{f}(\mathbf{x}^{(i-1)}),\ \ \mathbf{s}^{(i)} = \mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}, \\ \mathbf{A}_i^{-1} = \mathbf{A}_{i-1}^{-1} + \dfrac{(\mathbf{s}^{(i)}-\mathbf{A}_{i-1}^{-1}\mathbf{y}^{(i)})(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}}{(\mathbf{s}^{(i)})^T\mathbf{A}_{i-1}^{-1}\mathbf{y}^{(i)}}, \\ \mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \mathbf{A}_i^{-1}\mathbf{f}(\mathbf{x}^{(i)}).\end{cases} \tag{6.21}$$

Broyden's method reduces a large amount of work from Newton's method. However, the quadratic convergence of Newton's method is lost. For Broyden's method, we have

$$\lim_{i\to\infty}\frac{\left\|\mathbf{x}^{(i+1)}-\mathbf{x}^*\right\|}{\left\|\mathbf{x}^{(i)}-\mathbf{x}^*\right\|} = 0.$$

This type of convergence is called **superlinear**. For more about Broyden's method, see books on numerical methods.

6.2.4 Obtaining Eigenvalues and Eigenvectors

In this subsection, we will discuss how to get eigenvalues and eigenvectors of a square matrix, especially, a symmetric matrix. Before that, we introduce some basic tools we will need.

Consider an $m \times m$ matrix in the form

$$\mathbf{H}_m = \mathbf{I}_m - \alpha \mathbf{v}\mathbf{v}^T,$$

where $\mathbf{I}_m$ is an $m \times m$ identity matrix, $\mathbf{v}$ is an m-dimensional vector, and α is a number. Obviously, $\mathbf{H}_m$ is a symmetric matrix. We also want $\mathbf{H}_m$ to be orthogonal, namely,

$$\begin{aligned}\mathbf{H}_m^T \mathbf{H}_m &= \left(\mathbf{I}_m - \alpha \mathbf{v}\mathbf{v}^T\right)\left(\mathbf{I}_m - \alpha \mathbf{v}\mathbf{v}^T\right) \\ &= \mathbf{I}_m - 2\alpha \mathbf{v}\mathbf{v}^T + \alpha^2 \mathbf{v}\mathbf{v}^T\mathbf{v}\mathbf{v}^T \\ &= \mathbf{I}_m - \left(2\alpha - \alpha^2 \mathbf{v}^T\mathbf{v}\right)\mathbf{v}\mathbf{v}^T = \mathbf{I}_m.\end{aligned}$$

Therefore, we require

$$\alpha = \frac{2}{\mathbf{v}^T\mathbf{v}}$$

and

$$\mathbf{H}_m = \mathbf{I}_m - \frac{2}{\mathbf{v}^T\mathbf{v}}\mathbf{v}\mathbf{v}^T. \tag{6.22}$$

The matrix defined by the expression (6.22) is called a Householder matrix. We are especially interested in the Householder matrix satisfying

$$\mathbf{H}_m\mathbf{x} = \beta \mathbf{e}_1, \tag{6.23}$$

where $\mathbf{x} = [x_1, x_2, \cdots, x_m]^T$ is an m-dimensional vector, β is a number whose value may depend on the components of $\mathbf{x}$, and $\mathbf{e}_1 = [1, 0, \cdots, 0]^T$. Because

$$\mathbf{H}_m\mathbf{x} = \mathbf{x} - \frac{2}{\mathbf{v}^T\mathbf{v}}\mathbf{v}\mathbf{v}^T\mathbf{x} = \mathbf{x} - \frac{2\mathbf{v}^T\mathbf{x}}{\mathbf{v}^T\mathbf{v}}\mathbf{v} = \beta \mathbf{e}_1,$$

we have

$$\mathbf{u} \equiv \frac{2\mathbf{v}^T\mathbf{x}}{\mathbf{v}^T\mathbf{v}}\mathbf{v} = \mathbf{x} - \beta \mathbf{e}_1 \tag{6.24}$$

and

$$\mathbf{u}^T\mathbf{x} = \mathbf{x}^T\mathbf{x} - \beta x_1, \quad \mathbf{u}^T\mathbf{u} = \mathbf{x}^T\mathbf{x} - 2\beta x_1 + \beta^2.$$

Therefore, we further obtain

$$\begin{aligned}\mathbf{H}_m\mathbf{x} &= \left(\mathbf{I}_m - \frac{2}{\mathbf{v}^T\mathbf{v}}\mathbf{v}\mathbf{v}^T\right)\mathbf{x} = \left(\mathbf{I}_m - \frac{2}{\mathbf{u}^T\mathbf{u}}\mathbf{u}\mathbf{u}^T\right)\mathbf{x} \\ &= \mathbf{x} - \frac{2\mathbf{u}^T\mathbf{x}}{\mathbf{u}^T\mathbf{u}}\mathbf{u} = \left(1 - \frac{2\mathbf{u}^T\mathbf{x}}{\mathbf{u}^T\mathbf{u}}\right)\mathbf{x} + \frac{2\mathbf{u}^T\mathbf{x}}{\mathbf{u}^T\mathbf{u}}\beta \mathbf{e}_1.\end{aligned}$$

Because we want the relation (6.23) to hold, we require

$$1 - \frac{2\mathbf{u}^T\mathbf{x}}{\mathbf{u}^T\mathbf{u}} = 1 - \frac{2\left(\mathbf{x}^T\mathbf{x} - \beta x_1\right)}{\mathbf{x}^T\mathbf{x} - 2\beta x_1 + \beta^2} = 0$$

or

$$\beta = \pm\sqrt{\mathbf{x}^T\mathbf{x}}. \tag{6.25}$$

Usually, we take the + sign so that the first component of the vector $\mathbf{H}_m\mathbf{x}$ is nonnegative. In this case

$$\mathbf{H}_m = \mathbf{I}_m - \frac{2}{\mathbf{u}^T\mathbf{u}}\mathbf{u}\mathbf{u}^T = \mathbf{I}_m - \frac{1}{\beta(\beta - x_1)}\mathbf{u}\mathbf{u}^T, \tag{6.26}$$

where $\mathbf{u}$ and β are given by the expressions (6.24) and (6.25).

An $n \times n$ matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}$$

is called an upper triangular matrix if $a_{ij} = 0$ for $i > j$ and an upper Hessenberg matrix if $a_{ij} = 0$ for $i > j + 1$. Because a Householder matrix defined by the expression (6.26) has the property (6.23), it can be used to reduce a matrix $\mathbf{A}$ to an upper triangular matrix or an upper Hessenberg matrix, which will be described below. Based on this fact, we can have the so-called **QR** algorithm for finding the eigenvalues of a matrix.

The first step of the **QR** algorithm for finding the eigenvalues of a matrix $\mathbf{A}$ is to reduce the matrix to an upper Hessenberg matrix. Let $\mathbf{P}_k$ be an $n \times n$ matrix in the form:

$$\mathbf{P}_k = \begin{bmatrix} \mathbf{I}_k & 0 \\ 0 & \mathbf{H}_{n-k} \end{bmatrix},$$

where k is equal to $0, 1, \cdots$, or $n - 2$, $\mathbf{H}_{n-k}$ is an $(n - k) \times (n - k)$ matrix defined by the expression (6.26). Clearly, $\mathbf{P}_k$ is a Householder matrix. Suppose that after using $k - 1$ Householder transformations, $\mathbf{A}$ is changed to

$$\mathbf{A}_{k-1} = (\mathbf{P}_1 \cdots \mathbf{P}_{k-1})^T \mathbf{A} (\mathbf{P}_1 \cdots \mathbf{P}_{k-1}) = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} & \mathbf{C}_{13} \\ \mathbf{C}_{21} & \mathbf{C}_{22} & \mathbf{C}_{23} \\ 0 & \mathbf{C}_{32} & \mathbf{C}_{33} \end{bmatrix},$$

where

$$\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix}$$

is a $k \times k$ upper Hessenberg matrix and $\mathbf{C}_{32}$ is a column vector. Now let us define $\mathbf{A}_k = \mathbf{P}_k^T \mathbf{A}_{k-1} \mathbf{P}_k$, and from the forms of $\mathbf{A}_{k-1}$ and $\mathbf{P}_k$ we have

$$\mathbf{A}_k = \mathbf{P}_k^T \mathbf{A}_{k-1} \mathbf{P}_k = \mathbf{P}_k \mathbf{A}_{k-1} \mathbf{P}_k = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} & \mathbf{C}_{13}\mathbf{H}_{n-k} \\ \mathbf{C}_{21} & \mathbf{C}_{22} & \mathbf{C}_{23}\mathbf{H}_{n-k} \\ 0 & \mathbf{H}_{n-k}\mathbf{C}_{32} & \mathbf{H}_{n-k}\mathbf{C}_{33}\mathbf{H}_{n-k} \end{bmatrix}.$$

If we take $\mathbf{C}_{32}$ as $\mathbf{x}$ and determine $\mathbf{H}_{n-k}$, then we arrive at

$$\mathbf{H}_{n-k}\mathbf{C}_{32} = \begin{bmatrix} \beta, 0, \cdots, 0 \end{bmatrix}^T.$$

Therefore, the $(k+1)\times(k+1)$ submatrix at the upper-left corner of $\mathbf{A}_k$ is an upper Hessenberg matrix, and the procedure can continue for $k+1$. For $k=1$, this procedure can be done. Consequently, we can do this procedure from $k=1$ to $n-2$, and finally obtain an upper Hessenberg matrix:

$$\mathbf{A}_{n-2} = (\mathbf{P}_1 \cdots \mathbf{P}_{n-2})^T \mathbf{A} (\mathbf{P}_1 \cdots \mathbf{P}_{n-2}). \tag{6.27}$$

Now let us discuss the second step. If the procedure above starts from $\mathbf{P}_0$ and a matrix is multiplied only by $\mathbf{P}_k^T$ from the left-hand side, then we will obtain an upper triangular matrix with nonnegative main diagonal entries. Therefore, for any matrix $\mathbf{B}$, we can find an orthogonal matrix $\mathbf{Q}^T$ such that $\mathbf{Q}^T\mathbf{B} = \mathbf{R}$ or $\mathbf{B} = \mathbf{QR}$, where $\mathbf{R}$ is an upper triangular matrix with nonnegative main diagonal entries. This procedure is called QR factorization. Using the QR factorization and letting $\mathbf{B}_1 = \mathbf{A}_{n-2}$, we have the following iteration:

$$\begin{aligned} \mathbf{B}_k &= \mathbf{Q}_k \mathbf{R}_k, \\ \mathbf{B}_{k+1} &= \mathbf{R}_k \mathbf{Q}_k = \mathbf{Q}_k^T \mathbf{B}_k \mathbf{Q}_k \end{aligned} \tag{6.28}$$

for $k = 1, 2, \cdots$. That is, first get $\mathbf{Q}_k$ and $\mathbf{R}_k$ from $\mathbf{B}_k$ and then multiplying $\mathbf{R}_k$ by $\mathbf{Q}_k$ from the right-hand side yields $\mathbf{B}_{k+1}$. For this iteration, we have the following relation

$$\begin{aligned} \mathbf{B}_{k+1} &= \mathbf{Q}_k^T \mathbf{B}_k \mathbf{Q}_k = \mathbf{Q}_k^T \cdots \mathbf{Q}_1^T \mathbf{A}_{n-2} \mathbf{Q}_1 \cdots \mathbf{Q}_k \\ &= (\mathbf{Q}_1 \cdots \mathbf{Q}_k)^T \mathbf{A}_{n-2} (\mathbf{Q}_1 \cdots \mathbf{Q}_k) \\ &= (\mathbf{P}_1 \cdots \mathbf{P}_{n-2} \mathbf{Q}_1 \cdots \mathbf{Q}_k)^T \mathbf{A} (\mathbf{P}_1 \cdots \mathbf{P}_{n-2} \mathbf{Q}_1 \cdots \mathbf{Q}_k), \end{aligned}$$

or

$$\mathbf{B}_{k+1} = \mathbf{S}_k^T \mathbf{A} \mathbf{S}_k,$$

where

$$\mathbf{S}_k = \mathbf{P}_1 \cdots \mathbf{P}_{n-2} \mathbf{Q}_1 \cdots \mathbf{Q}_k.$$

Let $\mathbf{B}$ and $\mathbf{S}$ be the limits of $\mathbf{B}_{k+1}$ and $\mathbf{S}_k$ as $k \to \infty$ respectively, then we have

$$\mathbf{B} = \mathbf{S}^T \mathbf{A} \mathbf{S}.$$

The goal of the iteration is to find an upper triangular matrix that is similar to $\mathbf{A}$, so that we can have the eigenvalues of $\mathbf{A}$ from the main diagonal entries of the upper triangular matrix. From the relation (6.28), we can see as follows. First, we get an upper triangular matrix by multiplying an orthogonal matrix from the left-hand side, but in order to let the new matrix be similar to the old one, multiplying the same orthogonal matrix from the right-hand side is needed, which may destroy the goal of finding an upper triangular matrix. However, under certain conditions it will be proved that the limit $\mathbf{B}$ is an upper triangular matrix. Therefore, we may reach our goal at the end of the iteration.

In order to find the eigenvectors of $\mathbf{A}$, we first need to find the eigenvectors of $\mathbf{B}$. As soon as we find the eigenvectors of $\mathbf{B}$, the eigenvectors of $\mathbf{A}$ can be obtained through multiplying the eigenvectors of $\mathbf{B}$ from the left-hand side by $\mathbf{S}$. If $\mathbf{A}$ is symmetric, then $\mathbf{B}$ is diagonal and every column of $\mathbf{S}$ is an eigenvector of $\mathbf{A}$.

For the convergence of the iteration we have

Theorem 6.5 *Assume that the eigenvalues of* $\mathbf{B}_1$ *have distinct absolute values, and* $\mathbf{X}^{-1}$ *has an* $\mathbf{LU}$ *decomposition, where* $\mathbf{X}$ *is the matrix of eigenvectors. Then,* $\mathbf{B}_k$ *converges to an upper triangular matrix.*

Proof. Suppose $\mathbf{X}$ has the decomposition

$$\mathbf{X} = \mathbf{Q}_x \mathbf{R}_x,$$

where $\mathbf{Q}_x$ is orthogonal and $\mathbf{R}_x$ is upper triangular with positive main diagonal entries. Then, we have

$$\begin{aligned}\mathbf{B}_1^k &= \mathbf{X}\mathbf{\Lambda}^k\mathbf{X}^{-1} = \mathbf{X}(\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k})\mathbf{\Lambda}^k\mathbf{U} \\ &= \mathbf{Q}_x\mathbf{R}_x(\mathbf{I}+\mathbf{E}_k)\mathbf{\Lambda}^k\mathbf{U} \\ &= \mathbf{Q}_x(\mathbf{I}+\mathbf{R}_x\mathbf{E}_k\mathbf{R}_x^{-1})\mathbf{R}_x\mathbf{\Lambda}^k\mathbf{U},\end{aligned}$$

where $\mathbf{\Lambda}$ is the Jordan canonical matrix of $\mathbf{B}_1$ and $\mathbf{E}_k = \mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k} - \mathbf{I} \to 0$ as $k \to \infty$ because we assume $|\lambda_1| > |\lambda_2| > \cdots |\lambda_n|$, $|\lambda_i|$ being an eigenvalue of $\mathbf{B}_1$. Let

$$\mathbf{I} + \mathbf{R}_x\mathbf{E}_k\mathbf{R}_x^{-1} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)},$$

where $\mathbf{Q}^{(k)}$ is orthogonal and $\mathbf{R}^{(k)}$ is upper triangular with positive main diagonal entries. Obviously,

$$\mathbf{Q}^{(k)} \to \mathbf{I},\ \mathbf{R}^{(k)} \to \mathbf{I}.$$

Let $\mathbf{D}$ and $\mathbf{D}_u$ be diagonal matrices defined by

$$\begin{aligned}\mathbf{D} &= \mathrm{diag}(\lambda_1/|\lambda_1|, \cdots, \lambda_n/|\lambda_n|), \\ \mathbf{D}_u &= \mathrm{diag}(u_{11}/|u_{11}|, \cdots, u_{nn}/|u_{nn}|),\end{aligned}$$

where u_{ii}, $i = 1, \cdots, n$, are the main diagonal entries of $\mathbf{U}$. Then, we have

$$\begin{aligned}\mathbf{B}_1^k &= \mathbf{Q}_x\mathbf{Q}^{(k)}\mathbf{R}^{(k)}\mathbf{R}_x\mathbf{\Lambda}^k\mathbf{U} \\ &= (\mathbf{Q}_x\mathbf{Q}^{(k)}\mathbf{D}_u\mathbf{D}^k)(\mathbf{D}^{-k}\mathbf{D}_u^{-1}\mathbf{R}^{(k)}\mathbf{R}_x\mathbf{\Lambda}^k\mathbf{U}).\end{aligned}$$

Because a product of two upper triangular matrices is an upper triangular matrix, and because a main diagonal entry of the new matrix is the product of the corresponding main diagonal entries in each original matrix, this is a QR decomposition of $\mathbf{B}_1^k$ and the upper triangular matrix $\mathbf{D}^{-k}\mathbf{D}_u^{-1}\mathbf{R}^{(k)}\mathbf{R}_x\mathbf{\Lambda}^k\mathbf{U}$ has positive main diagonal entries. On the other hand, it can be shown that

$$\mathbf{B}_1^k = \hat{\mathbf{Q}}_k\hat{\mathbf{R}}_k,$$

where

$$\hat{\mathbf{Q}}_k = \mathbf{Q}_1 \cdots \mathbf{Q}_k, \ \hat{\mathbf{R}}_k = \mathbf{R}_k \cdots \mathbf{R}_1.$$

In fact

$$\mathbf{B}_k = \hat{\mathbf{Q}}_{k-1}^T \mathbf{B}_1 \hat{\mathbf{Q}}_{k-1}$$

or

$$\mathbf{B}_1 \hat{\mathbf{Q}}_{k-1} = \hat{\mathbf{Q}}_{k-1} \mathbf{B}_k = \hat{\mathbf{Q}}_{k-1} \mathbf{Q}_k \mathbf{R}_k = \hat{\mathbf{Q}}_k \mathbf{R}_k.$$

Multiplying $\hat{\mathbf{R}}_{k-1}$ from the right-hand side on both sides of the relation $\hat{\mathbf{Q}}_k \mathbf{R}_k = \mathbf{B}_1 \hat{\mathbf{Q}}_{k-1}$, we get

$$\hat{\mathbf{Q}}_k \hat{\mathbf{R}}_k = \mathbf{B}_1 \hat{\mathbf{Q}}_{k-1} \hat{\mathbf{R}}_{k-1}$$

and furthermore we obtain

$$\hat{\mathbf{Q}}_k \hat{\mathbf{R}}_k = \mathbf{B}_1 \hat{\mathbf{Q}}_{k-1} \hat{\mathbf{R}}_{k-1} = \mathbf{B}_1^2 \hat{\mathbf{Q}}_{k-2} \hat{\mathbf{R}}_{k-2} = \cdots = \mathbf{B}_1^k.$$

Therefore, we have another QR decomposition of $\mathbf{B}_1^k$. Because the QR decomposition is unique, we have

$$\hat{\mathbf{Q}}_k = \mathbf{Q}_x \mathbf{Q}^{(k)} \mathbf{D}_u \mathbf{D}^k, \ \hat{\mathbf{R}}_k = \mathbf{D}^{-k} \mathbf{D}_u^{-1} \mathbf{R}^{(k)} \mathbf{R}_x \mathbf{\Lambda}^k \mathbf{U}.$$

Therefore,

$$\begin{aligned} \mathbf{B}_{k+1} &= (\mathbf{D}^T)^k \mathbf{D}_u^T (\mathbf{Q}^{(k)})^T \mathbf{Q}_x^T \mathbf{B}_1 \mathbf{Q}_x \mathbf{Q}^{(k)} \mathbf{D}_u \mathbf{D}^k \\ &= (\mathbf{D}^T)^k \mathbf{D}_u^T (\mathbf{Q}^{(k)})^T \mathbf{Q}_x^T \mathbf{Q}_x \mathbf{R}_x \mathbf{\Lambda} \mathbf{R}_x^{-1} \mathbf{Q}_x^{-1} \mathbf{Q}_x \mathbf{Q}^{(k)} \mathbf{D}_u \mathbf{D}^k \\ &= (\mathbf{D}^T)^k \mathbf{D}_u^T (\mathbf{Q}^{(k)})^T \mathbf{R}_x \mathbf{\Lambda} \mathbf{R}_x^{-1} \mathbf{Q}^{(k)} \mathbf{D}_u \mathbf{D}^k. \end{aligned}$$

Because $\mathbf{Q}^{(k)} \to \mathbf{I}$ and an inverse of an upper triangular matrix is still an upper triangular matrix, $\mathbf{B}_{k+1}$ converges to an upper triangular matrix. □

From the proof, we can see that it is not necessary for $\mathbf{B}_1$ to be an upper Hessenberg matrix. Having a Hessenberg matrix at the first step is for the practical reason of reducing computational cost. If $\mathbf{B}_k$ is in upper Hessenberg form, then $\mathbf{B}_{k+1}$ is also in upper Hessenberg form. Thus, in the entire iteration process, we deal with upper Hessenberg matrices. For an upper Hessenberg matrix, the amount of computational work at each step of the QR factorization is $O(n^2)$, which is much smaller than $O(n^3)$ for a full matrix. In order to make computation faster, we can also speed up the convergence of the **QR** algorithm by combining the shifting technique. In addition, there are some other methods for finding eigenvalues of a matrix, for example, the Jacobi algorithm. For more about the **QR** algorithm, the details of the shifting technique, and other methods, see books on matrix computation, for example, the book [35] by Golub and Loan.

6.3 Determination of Parameters in Models

In order to price an option on a specified underlying asset, we must have a model for the asset. We can have various models, and we have to determine the parameters in the model before pricing. In this section, we will discuss how to determine the parameters in models from the market data.

6.3.1 Constant Variances and Covariances

Assume that the stochastic process of an asset price S can be described by

$$dS = adt + bdX,$$

where a and b are constants and dX is a Wiener process. Because we assume that the parameters in the stochastic process do not depend on time, we can determine a and b according to the historical data. Clearly,

$$\mathrm{E}\left[dS\right] = adt$$

and

$$\mathrm{Var}\left[dS\right] = \mathrm{E}\left[(dS - adt)^2\right] = \mathrm{E}\left[(bdX)^2\right] = b^2 dt,$$

that is,

$$a = \frac{1}{dt}\mathrm{E}\left[dS\right]$$

and

$$b^2 = \frac{1}{dt}\mathrm{Var}\left[dS\right].$$

Suppose that from the market, we have the values of the asset price S at time $t^i = T_1 + (i-1)dt$, $i = 1, 2, \cdots, I+1$. From any statistics textbook, we know that the mean and variance of dS can be approximated by

$$\mathrm{E}\left[dS\right] \approx \frac{1}{I}\sum_{i=1}^{I} dS_i = \frac{1}{I}\sum_{i=1}^{I}(S_{i+1} - S_i)$$

and

$$\mathrm{Var}\left[dS\right] \approx \frac{1}{I-1}\sum_{i=1}^{I}\left[S_{i+1} - S_i - \frac{1}{I}\sum_{i=1}^{I}(S_{i+1} - S_i)\right]^2.$$

Thus, we have the estimates for a and b^2 as follows:

$$a \approx \frac{1}{Idt}\sum_{i=1}^{I}(S_{i+1} - S_i) \tag{6.29}$$

and

$$b^2 \approx \frac{1}{(I-1)dt}\sum_{i=1}^{I}\left[S_{i+1}-S_i-\frac{1}{I}\sum_{i=1}^{I}(S_{i+1}-S_i)\right]^2$$
$$= \frac{1}{(I-1)dt}\left[\sum_{i=1}^{I}(S_{i+1}-S_i)^2-\frac{1}{I}\left(\sum_{i=1}^{I}(S_{i+1}-S_i)\right)^2\right]. \quad (6.30)$$

Now suppose

$$dS = \mu S dt + \sigma S dX$$

and let us discuss how to find μ and σ from the market data. Because $dS = \mu S dt + \sigma S dX$ can be written as

$$d\ln S = (\mu - \sigma^2/2)dt + \sigma dX,$$

then we can estimate μ and σ^2 by

$$\sigma^2 \approx \frac{1}{(I-1)dt}\left[\sum_{i=1}^{I}(\ln S_{i+1}-\ln S_i)^2-\frac{1}{I}\left(\sum_{i=1}^{I}(\ln S_{i+1}-\ln S_i)\right)^2\right]$$
$$\approx \frac{1}{(I-1)dt}\left[\sum_{i=1}^{I}\left(\frac{S_{i+1}-S_i}{S_i}\right)^2-\frac{1}{I}\left(\sum_{i=1}^{I}\frac{S_{i+1}-S_i}{S_i}\right)^2\right] \quad (6.31)$$

and

$$\mu - \sigma^2/2 \approx \frac{1}{Idt}\sum_{i=1}^{I}(\ln S_{i+1}-\ln S_i) \approx \frac{1}{Idt}\sum_{i=1}^{I}\frac{S_{i+1}-S_i}{S_i}$$

or

$$\mu \approx \frac{1}{Idt}\sum_{i=1}^{I}\frac{S_{i+1}-S_i}{S_i}+\sigma^2/2. \quad (6.32)$$

Here, we have used the approximate relation

$$\ln S_{i+1}-\ln S_i \approx \frac{S_{i+1}-S_i}{S_i}.$$

Suppose that there are two stochastic processes:

$$dS_1 = a_1 dt + b_1 dX_1$$

and

$$dS_2 = a_2 dt + b_2 dX_2,$$

where a_1, b_1, a_2, and b_2 are constants, dX_1, dX_2 are two Wiener processes correlated with $\mathrm{E}[dX_1 dX_2] = \rho dt$. Assume that we have the values of the asset prices S_1 and S_2 at time $t^i = T_1 + (i-1)dt$, which are denoted by $S_{1,i}$ and $S_{2,i}$, $i = 1, 2, \cdots, I+1$. We can have estimates for a_1, b_1, a_2, and b_2 by

the formulae (6.29) and (6.30). Now let us discuss how to estimate ρ from $S_{1,i}$ and $S_{2,i}$, $i = 1, 2, \cdots, I+1$. Because

$$\mathrm{E}\left[dX_1 dX_2\right] = \mathrm{E}\left[\frac{dS_1 - a_1 dt}{b_1} \times \frac{dS_2 - a_2 dt}{b_2}\right] = \frac{1}{b_1 b_2}\left\{\mathrm{E}\left[dS_1 dS_2\right] - a_1 a_2 dt^2\right\},$$

we have

$$\rho = \frac{1}{b_1 b_2 dt}\left\{\mathrm{E}\left[dS_1 dS_2\right] - a_1 a_2 dt^2\right\}.$$

From statistics, we know

$$\mathrm{E}\left[dS_1 dS_2\right] \approx \frac{1}{I-1}\sum_{i=1}^{I}\left(S_{1,i+1} - S_{1,i}\right)\left(S_{2,i+1} - S_{2,i}\right),$$

so we have

$$\rho \approx \frac{1}{b_1 b_2 dt}\left[\frac{1}{I-1}\sum_{i=1}^{I}\left(S_{1,i+1} - S_{1,i}\right)\left(S_{2,i+1} - S_{2,i}\right) - a_1 a_2 dt^2\right]. \tag{6.33}$$

On the market, the data are given hourly, daily, and so forth, and only on workdays. Suppose we use the data given daily and the adopted time unit is year. When doing the computation, we should think that dt between two successive workdays is always equal to $1/I_w$, where I_w is the number of workdays per year.

6.3.2 Variable Parameters

From Figs. 1.1–1.7, we can see that the assumption of the volatility being constant might not be a good assumption. For example, Figs. 1.1 and 1.2 show that the prices of IBM and GE stocks have less volatilities if the price is lower. Therefore, we assume that volatilities are functions of stock prices S. That is, the stochastic process of S is described by

$$dS = a\left(S\right)dt + b\left(S\right)dX,$$

where $a\left(S\right)$ and $b\left(S\right)$ are functions of S to be determined. Because we do not assume the dependence of the parameters on time t, we can still determine $a\left(S\right)$ and $b\left(S\right)$ from the historical data.

Again, suppose that we have $I+1$ prices of an asset from the market: S_i, $i = 1, 2, \cdots, I+1$. Let $S_{\max}$ and $S_{\min}$ be the maximum and minimum values among them. Set $S_{(m)} = S_{\min} - \varepsilon + m\left(S_{\max} - S_{\min} + \varepsilon\right)/\left(M+1\right)$, $m = 0, 1, \cdots, M+1$, where ε is a small positive number. Clearly, $S_{(0)} = S_{\min} - \varepsilon$ and $S_{(M+1)} = S_{\max}$. The entire interval $\left(S_{(0)}, S_{(M+1)}\right]$ is divided into $M+1$ subintervals $\left(S_{(m-1)}, S_{(m)}\right]$, $m = 1, 2, \cdots, M+1$. Every S_i belongs to one of these subintervals. Consider S_i, $i = 1, 2, \cdots, I$. If $S_i \in \left(S_{(m-1)}, S_{(m)}\right]$, then

we say that S_i belongs to the set $\mathcal{S}_{(m)}$. Let I_m be the number of elements in the set $\mathcal{S}_{(m)}$. It is clear that $\sum_{m=1}^{M+1} I_m = I$. For each set $\mathcal{S}_{(m)}$, we can have a mean $a_{(m)}$ and a variance $b_{(m)}^2$ by the formulae (6.29) and (6.30).

The variance $b_{(m)}^2$ is an approximate variance of the random variable S at $S = (S_{(m-1)} + S_{(m)})/2$, $m = 1, 2, \cdots, M+1$. We define $S_{(m-1/2)} = \left(S_{(m-1)} + S_{(m)}\right)/2$, so $b(S_{(m-1/2)}) \approx b_{(m)}$. Because S is defined on $[0, \infty)$, $b(S)$ is a function on $[0, \infty)$. However, it is not convenient to approximate the function $b(S)$ defined on an infinite interval. Hence we introduce a transformation

$$\xi = \frac{S}{S + P_m},$$

where P_m is a positive number. This transformation maps $[0, \infty)$ to $[0, 1)$. Therefore, we assume that $b\,(S)$ is in the form $\bar{b}(\xi)$ and find $\bar{b}(\xi)$ on the interval $[0, 1)$. It is clear that $b_{(m)}$ should be an approximation to $\bar{b}\left(\xi_{(m-1/2)}\right)$, where $\xi_{(m-1/2)} = \dfrac{S_{(m-1/2)}}{S_{(m-1/2)} + P_m}$. Now the problem is reduced to finding a function $\bar{b}(\xi)$ such that the points $\left(\xi_{(m-1/2)}, b_{(m)}\right)$, $m = 1, 2, \cdots, M+1$, are as close to $\bar{b}(\xi)$ as possible. Assume

$$\bar{b}\,(\xi) = g\,(\xi) \sum_{n=0}^{N} a_n \xi^n,$$

where $N < M$ and $g\,(\xi)$ is a given function, for example, $g\,(\xi) = 1$ or $\dfrac{P_m \xi}{1 - \xi}$. Under this assumption, using the points $\left(\xi_{\left(m-\frac{1}{2}\right)}, b_{(m)}\right)$, $m = 1, 2, \cdots, M+1$ and taking the weights $b_m = I_m / I$, we can find $a_0, a_1, \cdots, a_N$ by the least squares method with weights in Sect. 6.1.4. As soon as we find $\bar{b}\,(\xi)$, we have $b\,(S)$ by

$$b\,(S) = g\left(\frac{S}{S + P_m}\right) \sum_{n=0}^{N} a_n \left(\frac{S}{S + P_m}\right)^n.$$

If $b(S) < 0$ in some small regions, then a local modification is needed in order to guarantee $b(S) \geq 0$ for all $S \in [0, \infty)$. For $a\,(S)$, the method is similar.

Now let us discuss the case involving several stochastic processes. For simplicity, suppose we have two stochastic processes governed by

$$dS_1 = a_1(S_1)dt + b_1(S_1)dX_1$$

and

$$dS_2 = a_2(S_2)dt + b_2(S_2)dX_2$$

with $\mathrm{E}[dX_1 dX_2] = \rho dt$, ρ being a constant. Using the method given above, we can find $a_1(S_1)$, $b_1(S_1)$, $a_2(S_2)$, and $b_2(S_2)$. Because we assume that ρ is a constant, it can be determined by

$$\begin{aligned}\rho &= \frac{1}{dt}\mathrm{E}[dX_1 dX_2] \\ &= \frac{1}{dt}\mathrm{E}\left[\frac{dS_1 - a_1(S_1)dt}{b_1(S_1)} \times \frac{dS_2 - a_2(S_2)dt}{b_2(S_2)}\right] \\ &\approx \frac{1}{(I-1)dt}\sum_{i=1}^{I}\left[\frac{S_{1,i+1} - S_{1,i} - a_1(S_{1,i})dt}{b_1(S_{1,i})} \times \frac{S_{2,i+1} - S_{2,i} - a_2(S_{2,i})dt}{b_2(S_{2,i})}\right].\end{aligned}$$

Problems

Table 6.1. Problems and Sections

Problems	Sections	Problems	Sections	Problems	Sections
1–6	6.1	7–14	6.2	15	6.3

1. Suppose $x_m = m\Delta x$.
   (a) Find the order of the error of the following approximate function

$$u(x) \approx \frac{x_{m+1} - x}{\Delta x}u(x_m) + \frac{x - x_m}{\Delta x}u(x_{m+1})$$

   by the Taylor expansion. Here $x \in [x_m, x_{m+1}]$.
   (b) Find the order of the error of the following approximate function

$$\begin{aligned}u(x) \approx &\frac{(x - x_m)(x - x_{m+1})}{2\Delta x^2}u(x_{m-1}) \\ &-\frac{(x - x_{m-1})(x - x_{m+1})}{\Delta x^2}u(x_m) \\ &+\frac{(x - x_{m-1})(x - x_m)}{2\Delta x^2}u(x_{m+1})\end{aligned}$$

   by the Taylor expansion. Here $x \in [x_{m-1}, x_{m+1}]$ and $x_{m-1} < x_m < x_{m+1}$.
2. *Show that from

$$\begin{cases} a_m = a_{m-1} + b_{m-1}h_{m-1} + c_{m-1}h_{m-1}^2 + d_{m-1}h_{m-1}^3, \\ b_m = b_{m-1} + 2c_{m-1}h_{m-1} + 3d_{m-1}h_{m-1}^2, \\ c_m = c_{m-1} + 3d_{m-1}h_{m-1}, \qquad m = 1, 2, \cdots, M-1, \end{cases}$$

and

$$\begin{cases} a_M = a_{M-1} + b_{M-1}h_{M-1} + c_{M-1}h_{M-1}^2 + d_{M-1}h_{M-1}^3, \\ c_M = c_{M-1} + 3d_{M-1}h_{M-1}, \end{cases}$$

the following relation can be derived:

$$\begin{aligned} &\frac{h_{m-1}}{h_{m-1}+h_m}c_{m-1} + 2c_m + \frac{h_m}{h_{m-1}+h_m}c_{m+1} \\ &= \frac{1}{h_{m-1}+h_m}\left[\frac{3(a_{m+1}-a_m)}{h_m} - \frac{3(a_m-a_{m-1})}{h_{m-1}}\right], \\ &\qquad m = 1, 2, \cdots, M-1. \end{aligned}$$

3. Consider the cubic spline problem. Suppose that the derivative is given at $x = x_M$, instead of assuming $c_M = 0$. Derive the equation which should replace the equation $c_M = 0$ in the system for $c_0, c_1, \cdots, c_M$.
4. Suppose $x_m = m\Delta x$, $y_l = l\Delta y$, and $\tau^n = n\Delta\tau$. Find the expression of the error of each of the following approximations:

(a) $u(x_m, \tau^{n+1/2}) \approx \dfrac{u(x_m, \tau^{n+1}) + u(x_m, \tau^n)}{2}$;

(b) $\dfrac{\partial u}{\partial \tau}(x_m, \tau^n) \approx \dfrac{u(x_m, \tau^{n+1}) - u(x_m, \tau^n)}{\Delta\tau}$;

(c) $\dfrac{\partial u}{\partial \tau}(x_m, \tau^{n+1/2}) \approx \dfrac{u(x_m, \tau^{n+1}) - u(x_m, \tau^n)}{\Delta\tau}$;

(d) $\dfrac{\partial u}{\partial x}(x_m, \tau^n) \approx \dfrac{u(x_{m+1}, \tau^n) - u(x_m, \tau^n)}{\Delta x}$;

(e) $\dfrac{\partial u}{\partial x}(x_m, \tau^n) \approx \dfrac{u(x_{m+1}, \tau^n) - u(x_{m-1}, \tau^n)}{2\Delta x}$;

(f) $\dfrac{\partial u}{\partial x}(x_m, \tau^n) \approx \dfrac{3u(x_m, \tau^n) - 4u(x_{m-1}, \tau^n) + u(x_{m-2}, \tau^n)}{2\Delta x}$;

(g) $\dfrac{\partial^2 u}{\partial x^2}(x_m, \tau^n) \approx \dfrac{u(x_{m+1}, \tau^n) - 2u(x_m, \tau^n) + u(x_{m-1}, \tau^n)}{\Delta x^2}$;

(h)

$$\begin{aligned} \frac{\partial^2 u}{\partial x \partial y}(x_m, y_l, \tau^n) \approx \frac{1}{2\Delta x}\Bigg[&\frac{u(x_{m+1}, y_{l+1}, \tau^n) - u(x_{m+1}, y_{l-1}, \tau^n)}{2\Delta y} \\ &- \frac{u(x_{m-1}, y_{l+1}, \tau^n) - u(x_{m-1}, y_{l-1}, \tau^n)}{2\Delta y}\Bigg]. \end{aligned}$$

5. The Chebyshev polynomial of first kind with degree N is defined by

$$T_N(y) = \cos\left(N \cos^{-1} y\right),$$

where N is an integer and $y \in [-1, 1]$. Let

$$y_j = \cos\frac{j\pi}{N}, \quad j = 0, 1, \cdots, N.$$

Show

(a) $T_{k+1}(y) - 2yT_k(y) + T_{k-1}(y) = 0$, $k \geq 1$.

(b) $T_N(y)$ is a polynomial of degree N for any nonnegative integer.

(c) $$\frac{dT_N(y_j)}{dy} = \begin{cases} N^2, & j = 0, \\ 0, & j = 1, 2, \cdots, N-1, \\ (-1)^{N+1} N^2, & j = N; \end{cases}$$

(d) $$\frac{d^2T_N(y_j)}{dy^2} = \begin{cases} \dfrac{N^2\left(N^2-1\right)}{3}, & j = 0, \\ \dfrac{(-1)^{j+1} N^2}{\left(1-y_j^2\right)}, & j = 1, 2, \cdots, N-1, \\ \dfrac{(-1)^N N^2\left(N^2-1\right)}{3}, & j = N; \end{cases}$$

(e) $$\frac{d^3T_N(y_j)}{dy^3} = \frac{(-1)^{j+1} 3N^2 y_j}{\left(1-y_j^2\right)^2}, \quad j = 1, 2, \cdots, N-1.$$

6. Let

$$h_j(y) = \frac{(-1)^{j+1}\left(1-y^2\right) T_N'(y)}{c_j N^2 (y - y_j)}, \quad j = 0, 1, \cdots, N,$$

where $T_N(y)$ is the Chebyshev polynomial of first kind with degree N, $y_j = \cos\frac{j\pi}{N}$, $j = 0, 1, \cdots, N$, and

$$c_j = \begin{cases} 2, & j = 0, \\ 1, & j = 1, 2, \cdots, N-1, \\ 2, & j = N. \end{cases}$$

(a) Show

$$h_j(y_i) = \frac{(-1)^{j+1}\left(1-y_i^2\right) T_N'(y_i)}{c_j N^2 (y_i - y_j)} = \delta_{ij}, \quad i, j = 0, 1, \cdots, N,$$

where δ_{ij} is the Kronecker delta.

(b) Define

$$d_{ij} = \frac{dh_j(y_i)}{dy}, \quad i, j = 0, 1, \cdots, N.$$

Show that

$$d_{ij} = \begin{cases} \dfrac{(-1)^{i+j} c_i}{c_j (y_i - y_j)}, & i \neq j, \\ \dfrac{2N^2+1}{6}, & i = j = 0, \\ -\dfrac{y_j}{2\left(1-y_j^2\right)}, & i = j = 1, 2, \cdots, N-1, \\ -\dfrac{2N^2+1}{6}, & i = j = N. \end{cases}$$

(c) Let $f_1(y_j)$ denote the values of the function $f_1(y)$ at $y = y_j$, $j = 0, 1, \cdots, N$. Show that

$$p_{N1}(y) = \sum_{j=0}^{N} h_j(y) f_1(y_j)$$

is an interpolation polynomial with degree N for $f_1(y)$ on $[-1, 1]$ and

$$\frac{dp_{N1}(y_i)}{dy} = \sum_{j=0}^{N} d_{ij} f_1(y_j).$$

(d) Define $x = (1-y)/2$ or $y = 1-2x$. Let $f(x_j)$ denote the values of the function $f(x)$ at $x = x_j$, $j = 0, 1, \cdots, N$. Show that

$$p_N(x) = \sum_{j=0}^{N} h_j(1-2x) f(x_j)$$

is an interpolation polynomial with degree N for $f(x)$ on $[0, 1]$ and

$$\frac{dp_N(x_i)}{dx} = \sum_{j=0}^{N} D_{ij} f(x_j),$$

where

$$D_{ij} = \begin{cases} \dfrac{(-1)^{i+j} c_i}{c_j (x_i - x_j)}, & i \neq j, \\ -\dfrac{2N^2+1}{3}, & i = j = 0, \\ \dfrac{1-2x_j}{4x_j(1-x_j)}, & i = j = 1, 2, \cdots, N-1, \\ \dfrac{2N^2+1}{3}, & i = j = N. \end{cases}$$

7. Derive the formulae of the LU decomposition method for the following almost tridiagonal system

$$\mathbf{A}\mathbf{x} = \mathbf{q},$$

where

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & & d_1 \\ a_2 & b_2 & c_2 & & 0 & d_2 \\ & \ddots & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots & \vdots \\ & 0 & & a_{m-1} & b_{m-1} & d_{m-1} \\ & & & & a_m & d_m \end{bmatrix},$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix}.$$

8. Suppose that we already have a solver for solving tridiagonal system:

$$\mathbf{A}\mathbf{x} = \mathbf{q},$$

where

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ & & & a_m & b_m \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{m-1} \\ q_m \end{bmatrix}.$$

In order to solve the following almost tridiagonal system

$$\mathbf{A}\mathbf{x} = \mathbf{q},$$

where

$$\mathbf{A} = \begin{bmatrix} a_1 & b_1 & c_1 & & \\ a_2 & b_2 & c_2 & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ & & a_m & b_m & c_m \end{bmatrix} \quad \text{or} \quad \mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ & & a_m & b_m & c_m \end{bmatrix},$$

we can convert it to a tridiagonal system and solve the new system by the existing solver. Design such a method.

9. *Describe the Jacobi iteration, the Gauss–Seidel iteration, and the method of successive over relaxation for an $n \times n$ system of linear equations.

10. *Suppose $f(x) = 0$ is a nonlinear equation. Derive the iteration formulae of Newton's method and the secant method for solving the nonlinear equation.
11. (a) For each of the following methods, describe the details of the method and its advantage and disadvantage:
    i. The secant method;
    ii. The bisection method;
    iii. The modified secant method.
    (b) Based on the methods in part a), design an efficient and robust method of finding a root of the equation $f(x) = 0$.
12. Suppose

$$\mathbf{A}_1 = \mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)}) + \frac{\mathbf{f}(\mathbf{x}^{(1)}) - \mathbf{f}(\mathbf{x}^{(0)}) - \mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)})(\mathbf{x}^{(1)} - \mathbf{x}^{(0)})}{\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|_2^2}(\mathbf{x}^{(1)} - \mathbf{x}^{(0)})^T.$$

    Show that the following relations hold:

$$\mathbf{A}_1(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}) = \mathbf{f}(\mathbf{x}^{(1)}) - \mathbf{f}(\mathbf{x}^{(0)})$$

    and

$$\mathbf{A}_1\mathbf{z} = \mathbf{J}_\mathbf{f}(\mathbf{x}^{(0)})\mathbf{z} \quad \text{whenever} \quad (\mathbf{x}^{(1)} - \mathbf{x}^{(0)})^T\mathbf{z} = 0.$$

13. Prove that if $\mathbf{A} \in \mathbb{R}^{n\times n}$ is nonsingular, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, and $\mathbf{y}^T\mathbf{A}^{-1}\mathbf{x} \neq -1$, then $\mathbf{A} + \mathbf{x}\mathbf{y}^T$ is also nonsingular, moreover,

$$(\mathbf{A} + \mathbf{x}\mathbf{y}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{x}\mathbf{y}^T\mathbf{A}^{-1}}{1 + \mathbf{y}^T\mathbf{A}^{-1}\mathbf{x}}.$$

14. (a) *Show

$$\mathbf{H}_m\mathbf{x} = \beta\mathbf{e}_1,$$

    where

$$\begin{aligned} \mathbf{x} &= [x_1, x_2, \cdots, x_m]^T, \\ \beta &= \sqrt{\mathbf{x}^T\mathbf{x}}, \\ \mathbf{H}_m &= \mathbf{I}_m - \frac{1}{\beta(\beta - x_1)}\mathbf{u}\mathbf{u}^T, \\ &\quad \mathbf{u} \text{ being } \mathbf{x} - \beta\mathbf{e}_1. \end{aligned}$$

    (b) *Using the result in part a), design a method to obtain an orthogonal matrix $\mathbf{Q}$ from $\mathbf{A}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{R}$, where $\mathbf{R}$ is an upper triangular matrix with nonnegative diagonals.
15. *Assume that the volatility of a stock is a function of the stock price. Describe a method determining the function from the market data.

Projects

General Requirements

(A) Submit a code or codes in C or C++ that will work on a computer the instructor can get access to. At the beginning of the code, write down the name of the student and indicate on which computer it works and the procedure to make it work.

(B) Each code should use an input file to specify all the problem parameters and the computational parameters and an output file to store all the results. In an output file, the name of the problem, all the problem parameters, and the computational parameters should be given, so that one can know what the results are and how they were obtained. The input file should be submitted with the code.

(C) Submit results in form of tables. When a result is given, always provide the problem parameters and the computational parameters.

1. **Cumulative Distribution Functions and Black–Scholes Formulae**.

Write five functions:

(a) **double** N(double z)

for computing approximate values of the cumulative distribution function for the standardized normal variable by using the expression given in a footnote of Sect. 2.6.3, where z is the independent variable.

- Give the values of $N(z)$ for $z = -2, -1, 0, 1, 2$.

(b) **double** BS(double S, double E, double tau, double r, double D0, double sigma, char option),

which gives prices of the European options by using Black–Scholes formulae (see Sect. 2.6.5). When the value of the character 'option' is equal to 'c' or 'C', the value of the European call needs to be evaluated. Otherwise, the value of the European put needs to be evaluated.

- For European call and put options, give the results for the cases: $S = 100$, $E = 95, 100, 105$, $T = 1$, $r = 0.1$, $D_0 = 0.05$, $\sigma = 0.2$.
- For European call and put options, give the results for the cases: $S = 100$, $E = 95, 100, 105$, $T = 1$, $r = 0.05$, $D_0 = 0.1$, $\sigma = 0.2$.

(c) **double** BS_bar(double xi, double E, double tau, double r, double D0, double sigma, char option)

This function gives the value of $\bar{c}(\xi, \tau) = c(S, t)/(S + E)$ or $\bar{p}(\xi, \tau) = p(S, t)/(S + E)$.

- For $\xi = 0.5128, 0.5000, 0.4878$, $E = 95, 100, 105$, $\tau = 1$, $r = 0.1$, $D_0 = 0.05$, $\sigma = 0.2$, calculate the results of $\bar{c}(\xi, \tau)$ and $\bar{p}(\xi, \tau)$ by this function.

(d) **double** N_2(double x1, double x2, double rho)

for computing approximate values of the cumulative distribution function for the bivariate standard normal distribution by using the expression given in a footnote of Sect. 4.5.3, where x1, x2 and rho are parameters.

- Give the values of $N_2(x_1, x_2, \rho)$ for the following sets of (x_1, x_2, ρ): $(0.6, 0.5, 0.6)$, $(0.4, 0.5, 0.8)$, $(0.3, 0.4, -0.6)$, $(0.5, 0.7, -0.8)$.

(e) **double** BS_2(double S1, double S2, double E, double tau, double r, double D01, double D02, double sigma1, double sigma2, double rho, char option)

which gives prices of the European call option on the maximum of two assets and the European put option on the minimum of two assets by using the closed-form solutions (4.76) and (4.77) given in Sect. 4.5.3. When the value of the character 'option' is equal to 'c' or 'C', the value of the European call needs to be evaluated. Otherwise, the value of the European put needs to be evaluated.

- Find the prices of the European call option on the maximum of two assets for the following parameter sets of $(S_1, S_2, E, \tau, r, D_{01}, D_{02}, \sigma_1, \sigma_2, \rho, \text{option})$:

$$
\begin{aligned}
&(100, 100, 100, 1.0, 0.02, 0.01, 0.01, 0.20, 0.20, 0.8, \text{c}),\\
&(100, 105, 100, 1.0, 0.02, 0.01, 0.01, 0.20, 0.15, 0.8, \text{c}),\\
&(100, 105, 100, 1.0, 0.02, 0.01, 0.01, 0.15, 0.20, 0.8, \text{c}),\\
&(100,\ \ 95, 100, 1.0, 0.02, 0.01, 0.01, 0.20, 0.15, 0.8, \text{c}),\\
&(100,\ \ 95, 100, 1.0, 0.02, 0.01, 0.01, 0.15, 0.20, 0.8, \text{c}).
\end{aligned}
$$

- Find the prices of the European put option on the minimum of two assets for the following parameter sets of $(S_1, S_2, E, \tau, r, D_{01}, D_{02}, \sigma_1, \sigma_2, \rho, \text{option})$:

$$
\begin{aligned}
&(100, 100, 100, 1.0, 0.02, 0.01, 0.01, 0.20, 0.20, 0.8, \text{p}),\\
&(100, 105, 100, 1.0, 0.02, 0.01, 0.01, 0.20, 0.15, 0.8, \text{p}),\\
&(100, 105, 100, 1.0, 0.02, 0.01, 0.01, 0.15, 0.20, 0.8, \text{p}),\\
&(100,\ \ 95, 100, 1.0, 0.02, 0.01, 0.01, 0.20, 0.15, 0.8, \text{p}),\\
&(100,\ \ 95, 100, 1.0, 0.02, 0.01, 0.01, 0.15, 0.20, 0.8, \text{p}).
\end{aligned}
$$

2. **Quadratic Interpolation and LU Decomposition of a Tridiagonal System**.
For the quadratic interpolation method (see Sect. 6.1.1), write a function

(a) **double** Interpolation(double x, int M, double *y)

Suppose that x, M, and $y_m = y(x_m), m = 0, 1, \ldots, M$, are given, where $x_m = m/M$. This function gives an approximate value of y(x) by quadratic interpolation. The concrete method is as follows. If $x < 1/2M$, then interpolate or extrapolate $y(x)$ by (x_0, y_0), (x_1, y_1), (x_2, y_2), if $x_m - 1/2M \le x < x_m + 1/2M$, $m = 1, 2, \cdots, M-1$, then interpolate $y(x)$ by (x_{m-1}, y_{m-1}), (x_m, y_m), (x_{m+1}, y_{m+1}), and if $x_M - 1/2M \le x$, then interpolate or extrapolate $y(x)$ by (x_{M-2}, y_{M-2}), (x_{M-1}, y_{M-1}), (x_M, y_M).

- Let $M = 5$ and the six components from y_0 to y_5 are 0.0, 0.008, 0.064, 0.216, 0.512, 1.0. Calculate the values of $y(x)$ for $x = -0.1, 0.45, 1.01$ by this function.

For LU decomposition (see Sect. 6.2.1), write two functions:

(b) **int** LUT(int m, double *a, double *b, double *c, double *q, double *x).

Suppose that we have a tridiagonal system (6.12). The number of unknowns is given in the integer 'm'. The nonhomogeneous term q_i is given in q[i-1] (the ith component of the array 'q'). The coefficients a_i, b_i, and c_i are given in the ith component of the arrays 'a', 'b', and 'c', respectively. Write a function to solve the system by using the method described in Sect. 6.2.1. If all the u_i are not equal to zero, then the code should return an integer number 0 and gives the value of the ith unknown in the ith component of the array **x**. If one of u_i is equal to zero, then the solution(s) of the system cannot be found by the method (or the system has no solution), and the code should return an integer number 1. The values in the arrays 'a', 'b', 'c', and 'q' are required unchanged.

- Let $m = 4$, $a_2 = a_3 = a_4 = -0.48$, $b_1 = b_2 = b_3 = b_4 = 1$, $c_1 = c_2 = c_3 = -0.49$, $q_1 = 0.02$, $q_2 = 0.05$, $q_3 = 0.08$, and $q_4 = 2.56$. Find the solution of the system (6.12).

(c) **int** LUAT(int m, double *a, double *b, double *c, double *q, double *x)

This is a solver for an almost tridiagonal system by LU decomposition. The almost tridiagonal system is in the following form:

$$\mathbf{A}\mathbf{x} = \mathbf{q},$$

where

$$\mathbf{A} = \begin{bmatrix} a_1 & b_1 & c_1 & & \\ a_2 & b_2 & c_2 & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ & & a_m & b_m & c_m \end{bmatrix}.$$

This function calculates $\mathbf{x}$ if m, $\mathbf{a}, \mathbf{b}, \mathbf{c}$, and $\mathbf{q}$ are given. Require m, $\mathbf{a}, \mathbf{b}, \mathbf{c}$, and $\mathbf{q}$ unchanged.

- Let $m = 5$, $\mathbf{a} = \{1.75, -0.48, -0.48, -0.48, 0.25\}$, $\mathbf{b} = \{-0.5, 1, 1, 1, -0.5\}$, $\mathbf{c} = \{0.25, -0.49, -0.49, -0.49, 1.75\}$, $\mathbf{q} = \{1.5, 0.05, 0.08, 0.11, 7.5\}$, calculate the result of $\mathbf{x}$ by this function.