Chapter 5
Numerical methods for matrices

Matrix operations are involved in many numerical and analytical scientific problems. Schemes developed for the matrix problems can be applied to the related problems encountered in ordinary and partial differential equations. For example, an eigenvalue problem given in the form of a partial differential equation can be rewritten as a matrix problem. A boundary-value problem after discretization is essentially a linear algebra problem.

5.1 Matrices in physics

Many problems in physics can be formulated in a matrix form. Here we give a few examples to illustrate the importance of matrix operations in physics and related fields.

If we want to study the vibrational spectrum of a molecule with n vibrational degrees of freedom, the first step is to investigate the harmonic oscillations of the system by expanding the potential energy up to the second order of the generalized coordinates around the equilibrium structure. Then we have the potential energy

$$U(q_1, q_2, \ldots, q_n) \simeq \frac{1}{2} \sum_{i,j=1}^{n} A_{ij} q_i q_j, \tag{5.1}$$

where q_i are the generalized coordinates and A_{ij} are the elements of the generalized elastic constant matrix that can be obtained through, for example, a quantum chemistry calculation or an experimental measurement. We have taken the equilibrium potential energy as the zero point. Usually the kinetic energy of the system can be expressed as

$$T(\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_n) \simeq \frac{1}{2} \sum_{i,j=1}^{n} M_{ij} \dot{q}_i \dot{q}_j, \tag{5.2}$$

where $\dot{q}_i = dq_i/dt$ are the generalized velocities, and M_{ij} are the elements of the generalized mass matrix whose values depend upon the specifics of the molecule. Now if we apply the Lagrange equation

$$\frac{\partial \mathcal{L}}{\partial q_i} - \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}_i} = 0, \tag{5.3}$$

where $\mathcal{L} = T - U$ is the Lagrangian of the system, we have

$$\sum_{j=1}^{n}(A_{ij}q_j + M_{ij}\ddot{q}_j) = 0, \tag{5.4}$$

for $i = 1, 2, \ldots, n$. If we assume that the time dependence of the generalized coordinates is oscillatory with

$$q_j = x_j e^{-i\omega t}, \tag{5.5}$$

we have

$$\sum_{j=1}^{n}(A_{ij} - \omega^2 M_{ij})x_j = 0, \tag{5.6}$$

for $i = 1, 2, \ldots, n$. This equation can be rewritten in a matrix form:

$$\begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \vdots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \lambda \begin{pmatrix} M_{11} & \cdots & M_{1n} \\ \vdots & \vdots & \vdots \\ M_{n1} & \cdots & M_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \tag{5.7}$$

or equivalently,

$$\mathbf{Ax} = \lambda \mathbf{Mx}, \tag{5.8}$$

where $\lambda = \omega^2$ is the eigenvalue and $\mathbf{x}$ is the corresponding eigenvector of the above eigenequation. Note that it is a homogeneous linear equation set. In order to have a nontrivial (not all zero components) solution of the equation set, the determinant of the coefficient matrix has to vanish, that is,

$$|\mathbf{A} - \lambda \mathbf{M}| = 0. \tag{5.9}$$

The roots of this secular equation, λ_k with $k = 1, 2, \ldots, n$, give all the possible vibrational angular frequencies $\omega_k = \sqrt{\lambda_k}$ of the molecule.

Another example we illustrate here deals with the problems associated with electrical circuits. We can apply the Kirchhoff rules to obtain a set of equations for the voltages and currents, and then we can solve the equation set to find the unknowns. Let us take the unbalanced Wheatstone bridge shown in Fig. 5.1 as an example. There is a total of three independent loops. We can choose the first as going through the source and two upper bridges and the second and third as the loops on the left and right of the ammeter. Each loop results in one of three independent equations:

$$r_s i_1 + r_1 i_2 + r_2 i_3 = v_0, \tag{5.10}$$

$$-r_x i_1 + (r_1 + r_x + r_a)i_2 - r_a i_3 = 0, \tag{5.11}$$

$$-r_3 i_1 - r_a i_2 + (r_2 + r_3 + r_a)i_3 = 0, \tag{5.12}$$

where r_1, r_2, r_3, r_x, r_a, and r_s are the resistances of the upper left bridge, upper right bridge, lower right bridge, lower left bridge, ammeter, and external source;

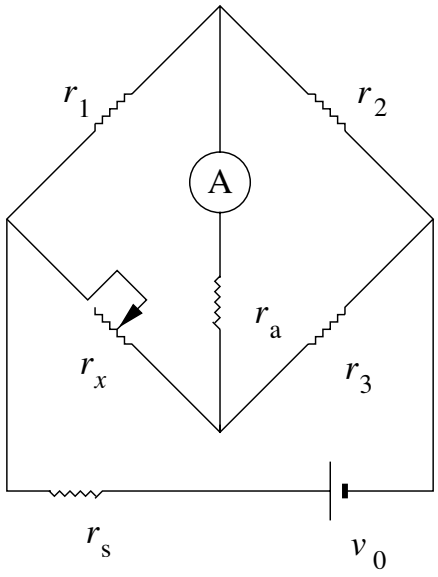

Fig. 5.1 The unbalanced Wheatstone bridge with all the resistors indicated.

i_1, i_2, and i_3 are the currents through the source, upper left bridge, and upper right bridge; and v_0 is the voltage of the external source. The above equation set can be rewritten in a matrix form

$$\mathbf{Ri} = \mathbf{v}, \tag{5.13}$$

where

$$\mathbf{R} = \begin{pmatrix} r_\mathrm{s} & r_1 & r_2 \\ -r_x & r_1 + r_x + r_\mathrm{a} & -r_\mathrm{a} \\ -r_3 & -r_\mathrm{a} & r_2 + r_3 + r_\mathrm{a} \end{pmatrix} \tag{5.14}$$

is the resistance coefficient matrix, and

$$\mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} \quad \text{and} \quad \mathbf{v} = \begin{pmatrix} v_0 \\ 0 \\ 0 \end{pmatrix} \tag{5.15}$$

are the current and voltage arrays that are in the form of column matrices. If we multiply both sides of the equation by the inverse matrix $\mathbf{R}^{-1}$, we have

$$\mathbf{i} = \mathbf{R}^{-1}\mathbf{v}, \tag{5.16}$$

which is solved if we know $\mathbf{R}^{-1}$. We will see in Section 5.3 that, in general, it is not necessary to know the inverse of the coefficient matrix in order to solve a linear equation set. However, as soon as we know the solution of the linear equation set, we can obtain the inverse of the coefficient matrix using the above equation.

A third example lies the calculation of the electronic structure of a many-electron system. Let us examine a very simple, but still meaningful system, H_3^+. Three protons are arranged in an equilateral triangle. The two electrons in the system are shared by all three protons. Assuming that we can describe the system by a simple Hamiltonian $\mathcal{H}$, containing one term for the hopping of an electron

from one site to another and another for the repulsion between the two electrons on a doubly occupied site, we have

$$\mathcal{H} = -t \sum_{i \neq j} a_{i\sigma}^{\dagger} a_{j\sigma} + U \sum n_{i\uparrow} n_{i\downarrow}, \tag{5.17}$$

where $a_{i\sigma}^{\dagger}$ and $a_{j\sigma}$ are creation and annihilation operators of an electron with either spin up ($\sigma = \uparrow$) or spin down ($\sigma = \downarrow$), and $n_{i\sigma} = a_{i\sigma}^{\dagger} a_{i\sigma}$ is the corresponding occupancy at the ith site. This Hamiltonian is called the Hubbard model when it is used to describe highly correlated electronic systems. The parameters t and U in the Hamiltonian can be obtained from either a quantum chemistry calculation or experimental measurement. The Schrödinger equation for the system is

$$\mathcal{H}|\Psi_k\rangle = \mathcal{E}_k|\Psi_k\rangle, \tag{5.18}$$

where $\mathcal{E}_k$ and $|\Psi_k\rangle$ are the kth eigenvalue and eigenstate of the Hamiltonian. Because each site has only one orbital, we have a total of 15 possible states for the two electrons under the single-particle representation,

$$\begin{aligned}
|\phi_1\rangle &= a_{1\uparrow}^{\dagger} a_{1\downarrow}^{\dagger}|0\rangle, \quad |\phi_2\rangle = a_{2\uparrow}^{\dagger} a_{2\downarrow}^{\dagger}|0\rangle, \quad |\phi_3\rangle = a_{3\uparrow}^{\dagger} a_{3\downarrow}^{\dagger}|0\rangle, \\
|\phi_4\rangle &= a_{1\uparrow}^{\dagger} a_{2\downarrow}^{\dagger}|0\rangle, \quad |\phi_5\rangle = a_{1\uparrow}^{\dagger} a_{3\downarrow}^{\dagger}|0\rangle, \quad |\phi_6\rangle = a_{2\uparrow}^{\dagger} a_{3\downarrow}^{\dagger}|0\rangle, \\
|\phi_7\rangle &= a_{1\downarrow}^{\dagger} a_{2\uparrow}^{\dagger}|0\rangle, \quad |\phi_8\rangle = a_{1\downarrow}^{\dagger} a_{3\uparrow}^{\dagger}|0\rangle, \quad |\phi_9\rangle = a_{2\downarrow}^{\dagger} a_{3\uparrow}^{\dagger}|0\rangle, \\
|\phi_{10}\rangle &= a_{1\uparrow}^{\dagger} a_{2\uparrow}^{\dagger}|0\rangle, \quad |\phi_{11}\rangle = a_{1\uparrow}^{\dagger} a_{3\uparrow}^{\dagger}|0\rangle, \quad |\phi_{12}\rangle = a_{2\uparrow}^{\dagger} a_{3\uparrow}^{\dagger}|0\rangle, \\
|\phi_{13}\rangle &= a_{1\downarrow}^{\dagger} a_{2\downarrow}^{\dagger}|0\rangle, \quad |\phi_{14}\rangle = a_{1\downarrow}^{\dagger} a_{3\downarrow}^{\dagger}|0\rangle, \quad |\phi_{15}\rangle = a_{2\downarrow}^{\dagger} a_{3\downarrow}^{\dagger}|0\rangle,
\end{aligned} \tag{5.19}$$

with $|0\rangle$ is the vacuum state. Then we can cast the Schrödinger equation as a matrix equation

$$\mathbf{H}\boldsymbol{\Psi}_k = \mathcal{E}_k \boldsymbol{\Psi}_k, \tag{5.20}$$

where $H_{ij} = \langle \phi_i | \mathcal{H} | \phi_j \rangle$ with $i, j = 1, 2, \ldots, 15$, and $\Psi_{ki} = \langle \phi_i | \Psi_k \rangle$. We leave it as an exercise for the reader to figure out the Hamiltonian elements. The 15 roots of the secular equation

$$|\mathbf{H} - \mathcal{E}\mathbf{I}| = 0 \tag{5.21}$$

are the eigenenergies of the system. Here $\mathbf{I}$ is a unit matrix with $I_{ij} = \delta_{ij}$. Note that the quantum problem here has a mathematical structure similar to that of the classical problem of molecular vibrations. We could have simplified the problem by exploiting the symmetries of the system. The total spin and the z component of the total spin commute with the Hamiltonian; thus they are good quantum numbers. We can reduce the Hamiltonian matrix into block-diagonal form with a maximum block size of 2×2. After we obtain all the eigenvalues and eigenvectors of the system, we can analyze the electronic, optical, and magnetic properties of H_3^+ easily.

5.2 Basic matrix operations

An $n \times m$ matrix $\mathbf{A}$ is defined through its elements A_{ij} with the row index $i = 1, 2, \ldots, n$ and the column index $j = 1, 2, \ldots, m$. It is called a square matrix if $n = m$. We will consider mainly the problems associated with square matrices in this chapter.

A variable array $\mathbf{x}$ with elements $x_1, x_2, \ldots, x_n$ arranged into a column is viewed as an $n \times 1$ matrix, or an n-element column matrix. A typical set of linear algebraic equations is given by

$$\sum_{j=1}^{n} A_{ij} x_j = b_i, \tag{5.22}$$

for $i = 1, 2, \ldots, n$, where x_j are the unknowns to be solved, A_{ij} are the given coefficients, and b_i are the given constants. Equation (5.22) can be written as

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.23}$$

with $\mathbf{A}\mathbf{x}$ defined from the standard matrix multiplication

$$C_{ij} = \sum_{k} A_{ik} B_{kj}, \tag{5.24}$$

for $\mathbf{C} = \mathbf{A}\mathbf{B}$. The summation over k requires the number of columns of the first matrix to be the same as the number of rows of the second matrix. Otherwise, the product does not exist. Basic matrix operations are extremely important. For example, the inverse of a square matrix $\mathbf{A}$ (written as $\mathbf{A}^{-1}$) is defined by

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}, \tag{5.25}$$

where $\mathbf{I}$ is a unit matrix with the elements $I_{ij} = \delta_{ij}$. The determinant of an $n \times n$ matrix $\mathbf{A}$ is defined as

$$|\mathbf{A}| = \sum_{i=1}^{n} (-1)^{i+j} A_{ij} |\mathbf{R}_{ij}| \tag{5.26}$$

for any $j = 1, 2, \ldots, n$, where $|\mathbf{R}_{ij}|$ is the determinant of the residual matrix $\mathbf{R}_{ij}$ of $\mathbf{A}$ with its ith row and jth column removed. The combination $C_{ij} = (-1)^{i+j} |\mathbf{R}_{ij}|$ is called a *cofactor* of $\mathbf{A}$. The determinant of a 1×1 matrix is the element itself. The determinant of a triangular matrix is the product of the diagonal elements. In principle, the inverse of $\mathbf{A}$ can be obtained through

$$A^{-1}{}_{ij} = \frac{C_{ji}}{|\mathbf{A}|}. \tag{5.27}$$

If a matrix has an inverse or nonzero determinant, it is called a nonsingular matrix. Otherwise, it is a singular matrix.

The trace of a matrix $\mathbf{A}$ is the sum of all its diagonal elements, written as

$$\mathrm{Tr}\,\mathbf{A} = \sum_{i=1}^{n} A_{ii}. \tag{5.28}$$

The transpose of a matrix $\mathbf{A}$ (written as $\mathbf{A}^{\mathrm{T}}$) has elements with the row and column indices of $\mathbf{A}$ interchanged, that is,

$$A_{ij}^{\mathrm{T}} = A_{ji}. \tag{5.29}$$

We call $\mathbf{A}$ an orthogonal matrix if $\mathbf{A}^{\mathrm{T}} = \mathbf{A}^{-1}$. The complex conjugate of $\mathbf{A}^{\mathrm{T}}$ is called the Hermitian operation of $\mathbf{A}$ (written as $\mathbf{A}^{\dagger}$) with $A_{ij}^{\dagger} = A_{ji}^{*}$. We call $\mathbf{A}$ a Hermitian matrix if $\mathbf{A}^{\dagger} = \mathbf{A}$ and a unitary matrix if $\mathbf{A}^{\dagger} = \mathbf{A}^{-1}$.

Another useful matrix operation is to add one row (or column) multiplied by a factor λ to another row (or column), such as

$$A'_{ij} = A_{ij} + \lambda A_{kj}, \tag{5.30}$$

for $j = 1, 2, \ldots, n$, where i and k are row indices, which can be different or the same. This operation preserves the determinant, that is, $|\mathbf{A}'| = |\mathbf{A}|$, and can be represented by a matrix multiplication

$$\mathbf{A}' = \mathbf{M}\mathbf{A}, \tag{5.31}$$

where $\mathbf{M}$ has unit diagonal elements plus a single nonzero off-diagonal element $M_{ik} = \lambda$. If we interchange any two rows or columns of a matrix, its determinant changes only its sign.

A matrix eigenvalue problem, such as the problem associated with the electronic structure of H_3^+, is defined by the equation

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \tag{5.32}$$

where $\mathbf{x}$ and λ are an eigenvector and its corresponding eigenvalue of the matrix, respectively. Note that this includes the eigenvalue problem associated with molecular vibrations, $\mathbf{A}\mathbf{x} = \lambda\mathbf{M}\mathbf{x}$, if we take $\mathbf{M}^{-1}\mathbf{A} = \mathbf{B}$ as the matrix of the problem. Then we have $\mathbf{B}\mathbf{x} = \lambda\mathbf{x}$.

The matrix eigenvalue problem can also be viewed as a linear equation set problem solved iteratively. For example, if we want to solve the eigenvalues and eigenvectors of the above equation, we can carry out the recursion

$$\mathbf{A}\mathbf{x}^{(k+1)} = \lambda^{(k)}\mathbf{x}^{(k)}, \tag{5.33}$$

where $\lambda^{(k)}$ and $\mathbf{x}^{(k)}$ are the approximate eigenvalue and eigenvector at the kth iteration. This is known as the *iteration method*. In Section 5.5 we will discuss a related scheme in more detail, but here we just want to point out that the problem is equivalent to the solution of the linear equation set at each iteration. We can

also show that the eigenvalues of a matrix under a similarity transformation

$$\mathbf{B} = \mathbf{S}^{-1}\mathbf{A}\mathbf{S} \tag{5.34}$$

are the same as the ones of the original matrix $\mathbf{A}$ because

$$\mathbf{B}\mathbf{y} = \lambda\mathbf{y} \tag{5.35}$$

is equivalent to

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \tag{5.36}$$

if we choose $\mathbf{x} = \mathbf{S}\mathbf{y}$. This, of course, assumes that matrix $\mathbf{S}$ is nonsingular. It also means that

$$|\mathbf{B}| = |\mathbf{A}| = \prod_{i=1}^{n} \lambda_i . \tag{5.37}$$

These basic aspects of matrix operations are quite important and will be used throughout this chapter. Readers who are not familiar with these aspects should consult one of the standard textbooks, for example, Lewis (1991).

5.3 Linear equation systems

A matrix is called an upper-triangular (lower-triangular) matrix if the elements below (above) the diagonal are all zero. The simplest scheme for solving matrix problems is *Gaussian elimination*, which uses very basic matrix operations to transform the coefficient matrix into an upper (lower) triangular one first and then finds the solution of the equation set by means of backward (forward) substitutions. The inverse and the determinant of a matrix can also be obtained in such a manner.

Let us take the solution of the linear equation set

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.38}$$

as an illustrative example. If we assume that $|\mathbf{A}| \neq 0$ and $\mathbf{b} \neq \mathbf{0}$, then the system has a unique solution. In order to show the steps of the Gaussian elimination, we will label the original matrix $\mathbf{A} = \mathbf{A}^{(0)}$ with $\mathbf{A}^{(j)}$ being the resultant matrix after j matrix operations. Similar notation is used for the transformed $\mathbf{b}$ as well.

Gaussian elimination

The basic idea of Gaussian elimination is to transform the original linear equation set to one that has an upper-triangular or lower-triangular coefficient matrix, but has the same solution. Here we want to transform the coefficient matrix into an upper triangular matrix. We can interchange the roles of the rows and the columns if we want to transform the coefficient matrix into a lower triangular one. In each

step of transformation, we eliminate the elements of a column that are not part of the upper triangle. The final linear equation set is then given by

$$\mathbf{A}^{(n-1)}\mathbf{x} = \mathbf{b}^{(n-1)}, \tag{5.39}$$

with $A_{ij}^{(n-1)} = 0$ for $i > j$. The procedure is quite simple. We first multiply the first equation by $-A_{i1}^{(0)}/A_{11}^{(0)}$, that is, all the elements of the first row in the coefficient matrix and $b_1^{(0)}$, and then add it to the ith equation for $i > 1$. The first element of every row except the first row in the coefficient matrix is then eliminated. Now we denote the new matrix $\mathbf{A}^{(1)}$ and multiply the second equation by $-A_{i2}^{(1)}/A_{22}^{(1)}$. Then we add the modified second equation to all the other equations with $i > 2$, the second element of every row except the first row and the second row of the coefficient matrix is eliminated. This procedure can be continued with the third, fourth, ..., and $(n-1)$th equations, and then the coefficient matrix becomes an upper-triangular matrix $\mathbf{A}^{(n-1)}$. A linear equation set with an upper-triangular coefficient matrix can easily be solved with backward substitutions.

Because all the diagonal elements are used in the denominators, the scheme would fail if any of them happened to be zero or a very small quantity. This problem can be circumvented in most cases by interchanging the rows and/or columns to have the elements used for divisions being the ones with largest magnitudes possible. This is the so-called pivoting procedure. This procedure will not change the solutions of the linear equation set but will put them in a different order. Here we will consider only the partial-pivoting scheme, which searches for the pivoting element (the one used for the next division) only from the remaining elements of the given column. A full-pivoting scheme searches for the pivoting element from all the remaining elements. After considering both the computing speed and accuracy, the partial-pivoting scheme seems to be a good compromise.

We first search for the element with the largest magnitude from $|A_{i1}^{(0)}|$ for $i = 1, 2, \ldots, n$. Assuming that the element obtained is $A_{k_1 1}^{(0)}$, we then interchange the first row and the k_1th row and eliminate the first element of each row except the first row. Similarly, we can search for the second pivoting element with the largest magnitude from $|A_{i2}^{(1)}|$ for $i = 2, 3, \ldots, n$. Assuming that the element obtained is $A_{k_2 2}^{(1)}$ with $k_2 > 1$, we can then interchange the second row and the k_2th row and eliminate the second element of each row except the first and second rows. This procedure is continued to complete the elimination and transform the original matrix into an upper-triangular matrix.

Physically, we do not need to interchange the rows of the matrix when searching for pivoting elements. An index can be used to record the order of the pivoting elements taken from all the rows. Furthermore, the ratios $A_{ij}^{(j-1)}/A_{k_j j}^{(j-1)}$ for $i > j$ are also needed to modify $\mathbf{b}^{(j-1)}$ into $\mathbf{b}^{(j)}$ and can be stored in the same two-dimensional array that stores matrix $\mathbf{A}^{(n-1)}$ at the locations where $A_{ij}^{(n-1)} = 0$, that is, the locations below the diagonal. When we determine the pivoting element, we also rescale the element from each row by the largest element magnitude of that row in order to have a fair comparison. This rescaling also reduces some

potential rounding errors. The following method is an implementation of the partial-pivoting Gaussian elimination outlined above.

```
// Method to carry out the partial-pivoting Gaussian
// elimination.  Here index[] stores pivoting order.

  public static void gaussian(double a[][],
    int index[]) {
    int n = index.length;
    double c[] = new double[n];

 // Initialize the index
    for (int i=0; i<n; ++i) index[i] = i;

 // Find the rescaling factors, one from each row
    for (int i=0; i<n; ++i) {
      double c1 = 0;
      for (int j=0; j<n; ++j) {
        double c0 = Math.abs(a[i][j]);
        if (c0 > c1) c1 = c0;
      }
      c[i] = c1;
    }

 // Search for the pivoting element from each column
    int k = 0;
    for (int j=0; j<n-1; ++j) {
      double pi1 = 0;
      for (int i=j; i<n; ++i) {
        double pi0 = Math.abs(a[index[i]][j]);
        pi0 /= c[index[i]];
        if (pi0 > pi1) {
          pi1 = pi0;
          k = i;
        }
      }

   // Interchange rows according to the pivoting order
      int itmp = index[j];
      index[j] = index[k];
      index[k] = itmp;
      for (int i=j+1; i<n; ++i) {
        double pj = a[index[i]][j]/a[index[j]][j];

     // Record pivoting ratios below the diagonal
        a[index[i]][j] = pj;

     // Modify other elements accordingly
        for (int l=j+1; l<n; ++l)
          a[index[i]][l] -= pj*a[index[j]][l];
      }
    }
  }
```

The above method destroys the original matrix. If we want to preserve it, we can just duplicate it in the method and work on the new matrix and return it after completing the procedure as done in the Lagrange interpolation programs for the data array in Chapter 2.

Determinant of a matrix

The determinant of the original matrix can easily be obtained after we have transformed it into an upper-triangular matrix through Gaussian elimination. As we have pointed out, the procedure used in the partial-pivoting Gaussian elimination does not change the value of the determinant only its sign, which can be fixed with the knowledge of the order of pivoting elements. For an upper-triangular matrix, the determinant is given by the product of all its diagonal elements. Therefore, we can obtain the determinant of a matrix as soon as it is transformed into an upper-triangular matrix. Here is an example of obtaining the determinant of a matrix with the partial-pivoting Gaussian elimination.

```
// An example of evaluating the determinant of a matrix
// via the partial-pivoting Gaussian elimination.

import java.lang.*;
public class  Det {
  public static void main(String argv[]) {
    double a[][]= {{ 1, -3,  2, -1, -2},
                   {-2,  2, -1,  2,  3},
                   { 3, -3, -2,  1, -1},
                   { 1, -2,  1, -3,  2},
                   {-3, -1,  2,  1, -3}};
    double d = det(a);
    System.out.println("The determinant is: " + d);
  }

// Method to evaluate the determinant of a matrix.

  public static double det(double a[][]) {
    int n = a.length;
    int index[] = new int[n];

 // Transform the matrix into an upper triangle
    gaussian(a, index);

 // Take the product of the diagonal elements
    double d = 1;
    for (int i=0; i<n; ++i) d = d*a[index[i]][i];

 // Find the sign of the determinant
    int sgn = 1;
    for (int i=0; i<n; ++i) {
      if (i != index[i]) {
        sgn = -sgn;
        int j = index[i];
        index[i] = index[j];
        index[j] = j;
      }
    }
    return sgn*d;
  }

  public static void gaussian(double a[][],
    int index[]) {...}
}
```

We obtain the determinant $|\mathbf{A}| = 262$ after running the above program. Note that we have used the recorded pivoting order to obtain the correct sign of the determinant. The method developed here is quite efficient and powerful and can be used in real problems that require repeated evaluation of the determinant of a large matrix, such as the local energy for a Fermi system in the variational and diffusion quantum Monte Carlo simulations, to be discussed in Chapter 10.

Solution of a linear equation set

Similarly, we can solve a linear equation set after transforming its coefficient matrix into an upper-triangular matrix through Gaussian elimination. The solution is then obtained through backward substitutions with

$$x_i = \frac{1}{A_{k_i i}^{(n-1)}} \left(b_{k_i}^{(n-1)} - \sum_{j=i+1}^{n} A_{k_i j}^{(n-1)} x_j \right), \tag{5.40}$$

for $i = n-1, n-2, \ldots, 1$, starting with $x_n = b_{k_n}^{(n-1)} / A_{k_n n}^{(n-1)}$. We have used k_i as the row index of the pivoting element from the ith column. We can easily show that the above recursion satisfies the linear equation set $\mathbf{A}^{(n-1)}\mathbf{x} = \mathbf{b}^{(n-1)}$. Here we demonstrate it using the example of finding the currents in the Wheatstone bridge, highlighted in Section 5.1 with $r_s = r_1 = r_2 = r_x = r_a = 100\ \Omega$ and $v_0 = 200$ V.

```
// An example of solving a linear equation set via the
// partial-pivoting Gaussian elimination.

import java.lang.*;
public class LinearEq {
  public static void main(String argv[]) {
    int n = 3;
    double x[] = new double[n];
    double b[] = {200,0,0};
    double a[][]= {{ 100,  100,  100},
                   {-100,  300, -100},
                   {-100, -100,  300}};
    int index[] = new int[n];
    x = solve(a, b, index);

    for (int i=0; i<n; i++)
      System.out.println("I_" + (i+1) + " = " + x[i]);
  }

// Method to solve the equation a[][] x[] = b[] with
// the partial-pivoting Gaussian elimination.

  public static double[] solve(double a[][],
    double b[], int index[]) {
    int n = b.length;
    double x[] = new double[n];

 // Transform the matrix into an upper triangle
    gaussian(a, index);

 // Update the array b[i] with the ratios stored
    for(int i=0; i<n-1; ++i) {
```

```
      for(int j =i+1; j<n; ++j) {
        b[index[j]] -= a[index[j]][i]*b[index[i]];
      }
    }

 // Perform backward substitutions
    x[n-1] = b[index[n-1]]/a[index[n-1]][n-1];
    for (int i=n-2; i>=0; --i) {
      x[i] = b[index[i]];
      for (int j=i+1; j<n; ++j) {
        x[i] -= a[index[i]][j]*x[j];
      }
      x[i] /= a[index[i]][i];
    }

 // Put x[i] into the correct order
    order(x,index);

    return x;
  }

  public static void gaussian(double a[][], int index[])
    {...}

// Method to sort an array x[i] from the lowest to the
// highest value of index[i].

  public static void order(double x[], int index[]){
    int m = x.length;
    for (int i = 0; i<m; ++i) {
      for (int j = i+1; j<m; ++j) {
        if (index[i] > index[j]) {
          int itmp = index[i];
          index[i] = index[j];
          index[j] = itmp;
          double xtmp = x[i];
          x[i] = x[j];
          x[j] = xtmp;
        }
      }
    }
  }
}
```

We obtain $i_1 = 1$ A and $i_2 = i_3 = 0.5$ A after running the above program.

Inverse of a matrix

The inverse of a matrix **A** can be obtained in exactly the same fashion if we realize that

$$A^{-1}{}_{ij} = x_{ij}, \tag{5.41}$$

for $i, j = 1, 2, \ldots, n$, where x_{ij} is the solution of the equation $\mathbf{A}\mathbf{x}_j = \mathbf{b}_j$, with $b_{ij} = \delta_{ij}$. If we expand **b** into a unit matrix in the program for solving a linear equation set, the solution corresponding to each column of the unit matrix forms

the corresponding column of $\mathbf{A}^{-1}$. With a little modification of the above routine for solving the linear equation set, we obtain the program for matrix inversion.

```
// An example of performing matrix inversion through the
// partial-pivoting Gaussian elimination.
import java.lang.*;
public class Inverse {
  public static void main(String argv[]) {
    double a[][]= {{ 100,  100,  100},
                   {-100,  300, -100},
                   {-100, -100,  300}};
    int n = a.length;
    double d[][] = invert(a);
    for (int i=0; i<n; ++i)
      for (int j=0; j<n; ++j)
        System.out.println(d[i][j]);
  }

  public static double[][] invert(double a[][]) {
    int n = a.length;
    double x[][] = new double[n][n];
    double b[][] = new double[n][n];
    int index[] = new int[n];
    for (int i=0; i<n; ++i) b[i][i] = 1;

 // Transform the matrix into an upper triangle
    gaussian(a, index);

 // Update the matrix b[i][j] with the ratios stored
    for (int i=0; i<n-1; ++i)
      for (int j=i+1; j<n; ++j)
        for (int k=0; k<n; ++k)
          b[index[j]][k]
            -= a[index[j]][i]*b[index[i]][k];

 // Perform backward substitutions
    for (int i=0; i<n; ++i) {
      x[n-1][i] = b[index[n-1]][i]/a[index[n-1]][n-1];
      for (int j=n-2; j>=0; --j) {
        x[j][i] = b[index[j]][i];
        for (int k=j+1; k<n; ++k) {
          x[j][i] -= a[index[j]][k]*x[k][i];
        }
        x[j][i] /= a[index[j]][j];
      }
    }
  return x;
  }

  public static void gaussian(double a[][], int index[])
    {...}
}
```

After running the above program, we obtain the inverse of the input matrix used in the program. Even though we have only used a very simple matrix to illustrate the method, the scheme itself is efficient enough for most real problems.

LU decomposition

A scheme related to the Gaussian elimination is called *LU decomposition*, which splits a nonsingular matrix into the product of a lower-triangular and an upper-triangular matrix. A simple example of the LU decomposition of a tridiagonal matrix was introduced in Section 2.4 in association with the cubic spline. Here we examine the general case.

As pointed out earlier in this section, the Gaussian elimination corresponds to a set of $n-1$ matrix operations that can be represented by a matrix multiplication

$$\mathbf{A}^{(n-1)} = \mathbf{M}\mathbf{A}, \tag{5.42}$$

where

$$\mathbf{M} = \mathbf{M}^{(n-1)}\mathbf{M}^{(n-2)}\cdots\mathbf{M}^{(1)}, \tag{5.43}$$

with each $\mathbf{M}^{(r)}$, for $r = 1, 2, \ldots, n-1$, completing one step of the elimination. It is easy to show that $\mathbf{M}$ is a lower-triangular matrix with all the diagonal elements being -1 and $M_{k_i j} = -A^{(j-1)}_{k_i j}/A^{(j-1)}_{k_j j}$ for $i > j$. So Eq. (5.42) is equivalent to

$$\mathbf{A} = \mathbf{L}\mathbf{U}, \tag{5.44}$$

where $\mathbf{U} = \mathbf{A}^{(n-1)}$ is an upper-triangular matrix and $\mathbf{L} = \mathbf{M}^{-1}$ is a lower-triangular matrix. Note that the inverse of a lower-triangular matrix is still a lower-triangular matrix. Furthermore, because $M_{ii} = -1$, we must have $L_{ii} = 1$ and $L_{ij} = -M_{ij} = A^{(j-1)}_{k_i j}/A^{(j-1)}_{k_j j}$ for $i > j$, which are the ratios stored in the matrix in the method introduced earlier in this section to perform the partial-pivoting Gaussian elimination. The method performs the LU decomposition with nonzero elements of $\mathbf{U}$ and nonzero, nondiagonal elements of $\mathbf{L}$ stored in the returned matrix. The choice of $L_{ii} = 1$ in the LU decomposition is called the Doolittle factorization. An alternative is to choose $U_{ii} = 1$; this is called the Crout factorization. The Crout factorization is equivalent to rescaling U_{ij} by U_{ii} for $i < j$ and multipling L_{ij} by U_{jj} for $i > j$ from the Doolittle factorization.

In general the elements in $\mathbf{L}$ and $\mathbf{U}$ can be obtained from comparison of the elements of the product of $\mathbf{L}$ and $\mathbf{U}$ with the elements of $\mathbf{A}$ in Eq. (5.44). The result can be cast into a recursion with

$$L_{ij} = \frac{1}{U_{jj}}\left(A_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj}\right), \tag{5.45}$$

$$U_{ij} = \frac{1}{L_{ii}}\left(A_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{kj}\right), \tag{5.46}$$

where $L_{i1} = A_{i1}/U_{11}$ and $\mathrm{U}_{1j} = A_{1j}/L_{11}$ are the starting values. If we implement the LU decomposition following the above recursion, we still need to consider the issue of pivoting, which shows up in the denominator of the expression for U_{ij} or L_{ij}. We can manage it in exactly the same way as in the Gaussian elimination. In practice, we can always store both $\mathbf{L}$ and $\mathbf{U}$ in one matrix, with

the elements below the diagonal for **L**, those above the diagonal for **U**, and the diagonal elements for either U_{ii} or L_{ii} depending on which of them is not equal to 1.

The determinant of **A** is given by the product of the diagonal elements of **L** and **U** because

$$|\mathbf{A}| = |\mathbf{L}||\mathbf{U}| = \prod_{i=1}^{n} L_{ii} U_{ii}. \tag{5.47}$$

As soon as we obtain **L** and **U** for a given matrix **A**, we can solve the linear equation set $\mathbf{Ax} = \mathbf{b}$ with one set of forward substitutions and another set of backward substitutions because the linear equation set can now be rewritten as

$$\mathbf{Ly} = \mathbf{b}, \tag{5.48}$$

$$\mathbf{Ux} = \mathbf{y}. \tag{5.49}$$

If we compare the elements on both sides of these equations, we have

$$y_i = \frac{1}{L_{ii}} \left(b_i - \sum_{k=1}^{i-1} L_{ik} y_k \right), \tag{5.50}$$

with $y_1 = b_1/L_{11}$ and $i = 2, 3, \ldots, n$. Then we can obtain the solution of the original linear equation set from

$$x_i = \frac{1}{U_{ii}} \left(y_i - \sum_{k=i+1}^{n} U_{ik} x_k \right), \tag{5.51}$$

for $i = n-1, n-2, \ldots, 1$, starting with $x_n = y_n/U_{nn}$.

We can also obtain the inverse of a matrix by a method similar to that used in the matrix inversion through the Gaussian elimination. We can choose a set of vectors with elements $b_{ij} = \delta_{ij}$ with $i, j = 1, 2, \ldots, n$. Here i refers to the element and j to a specific vector. The inverse of **A** is then given by ${A^{-1}}_{ij} = x_{ij}$ solving from $\mathbf{Ax}_j = \mathbf{b}_j$.

5.4 Zeros and extremes of multivariable functions

The numerical solution of a linear equation set discussed in the preceding section is important in computational physics and scientific computing because many problems there appear in the form of a linear equation set. A few examples of them are given in the exercises for this chapter.

However, there is another class of problems that are nonlinear in nature but can be solved iteratively with the linear schemes just developed. Examples include the solution of a set of nonlinear multivariable equations and a search for the maxima or minima of a multivariable function. In this section, we will show how to extend the matrix methods discussed so far to study nonlinear problems. We will also demonstrate the application of these numerical schemes to an actual physics problem, the stable geometric configuration of a multicharge system,

which is relevant in the study of the geometry of molecules and the formation of solid salts. From the numerical point of view, this is also a problem of searching for the global or a local minimum on a potential energy surface.

The multivariable Newton method

Nonlinear equations can also be solved with matrix techniques. In Chapter 3, we introduced the Newton method in obtaining the zeros of a single-variable function. Now we would like to extend our study to the solution of a set of multivariable equations. Assume that the equation set is given by

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \tag{5.52}$$

with $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. If the equation has at least one solution at $\mathbf{x}_\mathrm{r}$, we can perform the Taylor expansion around a neighboring point $\mathbf{x}$ with

$$\mathbf{f}(\mathbf{x}_\mathrm{r}) = \mathbf{f}(\mathbf{x}) + \Delta\mathbf{x} \cdot \boldsymbol{\nabla}\mathbf{f}(\mathbf{x}) + O(\Delta\mathbf{x}^2) \simeq \mathbf{0}, \tag{5.53}$$

where $\Delta\mathbf{x} = \mathbf{x}_\mathrm{r} - \mathbf{x}$. This is a linear equation set that can be cast into a matrix form

$$\mathbf{A}\Delta\mathbf{x} = \mathbf{b}, \tag{5.54}$$

where

$$A_{ij} = \frac{\partial f_i(\mathbf{x})}{\partial x_j} \tag{5.55}$$

and $b_i = -f_i$. Next we can find the root of the nonlinear equation set iteratively by carrying out the solutions of Eq. (5.54) at every point of iteration. The Newton method is then given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k, \tag{5.56}$$

where $\Delta\mathbf{x}_k$ is the solution of Eq. (5.54) with $\mathbf{x} = \mathbf{x}_k$. The Newton method for a single-variable equation discussed in Chapter 3 is the special case with $n = 1$.

As with the secant method discussed in Chapter 3, if the expression for the first-order derivative is not easy to obtain, the two-point formula can be used for the partial derivative

$$A_{ij} = \frac{f_i(\mathbf{x} + h_j\hat{\mathbf{x}}_j) - f_i(\mathbf{x})}{h_j}, \tag{5.57}$$

where h_j is a finite interval and $\hat{\mathbf{x}}_j$ is the unit vector along the direction of x_j. A rule of thumb in practice is to choose

$$h_j \simeq \delta_0 x_j, \tag{5.58}$$

where δ_0 is roughly the square root of the tolerance of the floating-point data used in the calculation; it is related to the total number of bits allocated for that data

type either by the programming language or by the processor of the computer. For example, the tolerance is about $2^{-31} \approx 5 \times 10^{-10}$ for the 32-bit floating-point data.

As a matter of fact, both the Newton and the secant method converge locally if the function is smooth enough, and the main computing cost is in solving the linear equation set defined in Eq. (5.54). Here we demonstrate the scheme using the following example in which we find the root of the set of two equations, $f_1(x, y) = e^{x^2} \ln y - x^2 = 0$ and $f_2(x, y) = e^{y^2} \ln x - y^2 = 0$, around $x = y = 1.5$.

```
// An example of searching for the root via the secant method
// for f_i[x_k] for i=1,2,...,n.

import java.lang.*;
public class Rootm {
  public static void main(String argv[]) {
    int n = 2;
    int ni = 10;
    double del = 1e-6;
    double x[] = {1.5, 1.5};
    secantm(ni, x, del);

 // Output the root obtained
    System.out.println("The root is at x = " + x[0]
      + "; y = " + x[1]);
  }

// Method to carry out the multivariable secant search.

  public static void secantm(int ni, double x[],
    double del) {
    int n = x.length;
    double h = 2e-5;
    int index[] = new int[n];
    double a[][] = new double[n][n];

    int k = 0;
    double dx = 0.1;
    while ((Math.abs(dx)>del) && (k<ni)) {
      double b[] = f(x);
      for (int i=0; i<n; ++i) {
        for (int j=0; j<n; ++j) {
          double hx = x[j]*h;
          x[j] += hx;
          double c[] = f(x);
          a[i][j] = (c[i]-b[i])/hx;
        }
      }
      for (int i=0; i<n; ++i) b[i] = -b[i];
      double d[] = solve(a, b, index);
      dx = 0;
      for (int i=0; i<n; ++i) {
        dx += d[i]*d[i];
        x[i] += d[i];
      }
```

```
      dx = Math.sqrt(dx/n);
      k++;
    }
    if (k==n) System.out.println("Convergence not" +
      " found after " + ni + " iterations");
  }

  public static double[] solve(double a[][], double b[],
    int index[]) {...}

  public static void gaussian(double a[][], int index[])
    {...}

// Method to provide function f_i[x_k].

  public static double[] f(double x[]) {
    double fx[] = new double[2];
    fx[0] = Math.exp(x[0]*x[0])*Math.log(x[1])
            -x[0]*x[0];
    fx[1] = Math.exp(x[1])*Math.log(x[0])-x[1]*x[1];
    return fx;
  }
}
```

Running the above program we obtain the root, which is at $x \simeq 1.559\,980$ and $y \simeq 1.238\,122$. The accuracy can be improved if we reduce the tolerance further.

Extremes of a multivariable function

Knowing the solution of a nonlinear equation set, we can develop numerical schemes to obtain the minima or maxima of a multivariable function. The extremes of a multivariable function $g(\mathbf{x})$ are the solutions of a nonlinear equation set

$$\mathbf{f}(\mathbf{x}) = \nabla g(\mathbf{x}) = \mathbf{0}, \tag{5.59}$$

where $\mathbf{f}(\mathbf{x})$ is an n-dimensional vector with each component given by a partial derivative of $g(\mathbf{x})$.

In Chapter 3, we introduced the steepest-descent method in the search of an extreme of a multivariable function. We can also use the multivariable Newton or secant method introduced above for such a purpose, except that special care is needed to ensure that $g(\mathbf{x})$ decreases (increases) during the updates for a minimum (maximum) of $g(\mathbf{x})$. For example, if we want to obtain a minimum of $g(\mathbf{x})$, we can update the position vector $\mathbf{x}$ following Eq. (5.56). We can also modify the matrix defined in Eq. (5.55) to

$$A_{ij} = \frac{\partial f_i}{\partial x_j} + \mu\delta_{ij}, \tag{5.60}$$

with μ taken as a small positive quantity to ensure that the modified matrix $\mathbf{A}$ is positive definite, that is, $\mathbf{w}^{\mathrm{T}}\mathbf{A}\mathbf{w} \geq 0$ for any nonzero $\mathbf{w}$. This will force the updates always to move in the direction of decreasing $g(\mathbf{x})$.

A special choice of μ, known as the BFGS (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) updating scheme, is to have

$$\mathbf{A}_k = \mathbf{A}_{k-1} + \frac{\mathbf{y}\mathbf{y}^{\mathrm{T}}}{\mathbf{y}^{\mathrm{T}}\mathbf{w}} - \frac{\mathbf{A}_{k-1}\mathbf{w}\mathbf{w}^{\mathrm{T}}\mathbf{A}_{k-1}}{\mathbf{w}^{\mathrm{T}}\mathbf{A}_{k-1}\mathbf{w}}, \tag{5.61}$$

where

$$\mathbf{w} = \mathbf{x}_k - \mathbf{x}_{k-1} \tag{5.62}$$

and

$$\mathbf{y} = \mathbf{f}_k - \mathbf{f}_{k-1}. \tag{5.63}$$

The BFGS scheme ensures that the updating matrix is positive definite; thus the search is always moving toward a minimum of $g(\mathbf{x})$. This scheme has been very successful in many practical problems. The reason behind its success is still unclear. If we want to find the maximum of $g(\mathbf{x})$, we can carry out exactly the same steps to find the minimum of $-g(\mathbf{x})$ with $\mathbf{f}(\mathbf{x}) = -\nabla g(\mathbf{x})$. If the gradient is not available analytically, we can use the finite difference instead, which is in the spirit of the secant method. For more details on the optimization of a function, see Dennis and Schnabel (1983). Much work has been done to develop a better numerical method for optimization of a multivariable function in the last few decades. However, the problem of global optimization of a multivariable function with many local minima is still open and may never be solved. For some discussions on the problem, see Wenzel and Hamacher (1999), Wales and Scheraga (1999), and Baletto *et al.* (2004).

Geometric structures of multicharge clusters

We now turn to a physics problem, the stable geometric structure of a multicharge cluster, which is extremely important in the analysis of small clusters of atoms, ions, and molecules. We will take clusters of $(\mathrm{Na}^+)_l(\mathrm{Cl}^-)_m$ with small positive integers l and m as illustrative examples. We will study geometric structures of the clusters with $2 \leq n = l + m \leq 6$. If we take the empirical form of the interaction potential between two ions in the NaCl crystal,

$$V(r_{ij}) = \eta_{ij}\frac{e^2}{4\pi\epsilon_0 r_{ij}} + \delta_{ij} V_0 e^{-r_{ij}/r_0}, \tag{5.64}$$

where $\eta_{ij} = -1$ and $\delta_{ij} = 1$ if the particles are of opposite charges; otherwise, $\eta_{ij} = 1$ and $\delta_{ij} = 0$. Here V_0 and r_0 are empirical parameters, which can be determined from either experiment or first-principles calculations. For solid NaCl, $V_0 = 1.09 \times 10^3$ eV and $r_0 = 0.321$ Å (Kittel, 1995). We will use these two quantities in what follows.

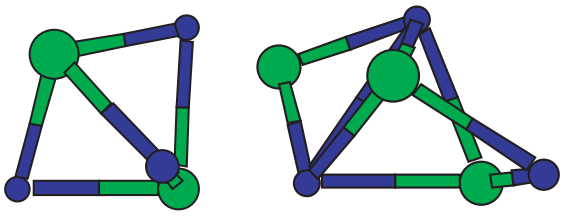

Fig. 5.2 Top view of stable structures of $Na^+(NaCl)_2$ (left) and $(NaCl)_3$ (right).

The function to be optimized is the total interaction potential energy of the system,

$$U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n) = \sum_{i>j}^{n} V(r_{ij}). \tag{5.65}$$

A local optimal structure of the cluster is obtained when U reaches a local minimum. This minimum cannot be a global minimum for large clusters, because no relaxation is allowed and a large cluster can have many local minima. There are $3n$ coordinates as independent variables in $U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n)$. However, because the position of the center of mass and rotation around the center of mass do not change the total potential energy U, we remove the center-of-mass motion and the rotational motion around the center of mass during the optimization. We can achieve this by imposing several restrictions on the cluster. First, we fix one of the particles at the origin of the coordinates. This removes three degrees of freedom of the cluster. Second, we restrict another particle on an axis, for example, the x axis. This removes two more degrees of freedom of the cluster. Finally, we restrict the third particle in a plane, for example, the xy plane. This removes the last (sixth) degree of freedom in order to fix the center of mass and make the system irrotational. The potential energy U now only has $3n - 6$ independent variables (coordinates).

The BFGS scheme has been applied to search for the local minima of U for NaCl, $Na^+(NaCl)$, $(NaCl)_2$, $Na^+(NaCl)_2$, and $(NaCl)_3$. The stable structures for the first three systems are very simple: a dimer, a straight line, and a square. The stable structures obtained for $Na^+(NaCl)_2$ and $(NaCl)_3$ are shown in Fig. 5.2. In order to avoid reaching zero separation of any two ions, a term $b(c/r_{ij})^{12}$ has been added in the interaction, with $b = 1$ eV and $c = 0.1$ Å, which adds on an amount of energy on the order of 10^{-18} eV to the total potential energy. It is worth pointing out that the structure of $(NaCl)_3$ is similar to the structure of $(H_2O)_6$ discovered by Liu *et al.* (1996). This might be an indication that water molecules are actually partially charged because of the intermolecular polarization.

5.5 Eigenvalue problems

Matrix eigenvalue problems are very important in physics. They are defined by

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \tag{5.66}$$

where λ is the eigenvalue corresponding to the eigenvector $\mathbf{x}$ of the matrix $\mathbf{A}$, determined from the secular equation

$$|\mathbf{A} - \lambda \mathbf{I}| = 0, \tag{5.67}$$

where $\mathbf{I}$ is a unit matrix. An $n \times n$ matrix has a total of n eigenvalues. The eigenvalues do not have to be different. If two or more eigenstates have the same eigenvalue, they are degenerate. The degeneracy of an eigenvalue is the total number of the corresponding eigenstates.

The eigenvalue problem is quite general in physics, because the matrix in Eq. (5.66) can come from many different problems, for example, the Lagrange equation for the vibrational modes of a large molecule, or the Schrödinger equation for H_3^+ discussed in Section 5.1. In this section, we will discuss the most common and useful methods for obtaining eigenvalues and eigenvectors in a matrix eigenvalue problem.

Eigenvalues of a Hermitian matrix

In many problems in physics and related fields, the matrix in question is Hermitian with

$$\mathbf{A}^\dagger = \mathbf{A}. \tag{5.68}$$

The simplicity of the Hermitian eigenvalue problem is due to three important properties associated with a Hermitian matrix:

(1) the eigenvalues of a Hermitian matrix are all real;
(2) the eigenvectors of a Hermitian matrix can be made orthonormal;
(3) a Hermitian matrix can be transformed into a diagonal matrix with the same set of eigenvalues under a similarity transformation of a unitary matrix that contains all its eigenvectors.

Furthermore, the eigenvalue problem of an $n \times n$ complex Hermitian matrix is equivalent to that of a $2n \times 2n$ real symmetric matrix. We can show this easily. For a complex matrix, we can always separate its real part from its imaginary part with

$$\mathbf{A} = \mathbf{B} + i\mathbf{C}, \tag{5.69}$$

where $\mathbf{B}$ and $\mathbf{C}$ are the real and imaginary parts of $\mathbf{A}$, respectively. If $\mathbf{A}$ is Hermitian, then $\mathbf{B}$ is a real symmetric matrix and $\mathbf{C}$ is a real skew symmetric matrix; they satisfy

$$B_{ij} = B_{ji}, \tag{5.70}$$

$$C_{ij} = -C_{ji}. \tag{5.71}$$

If we decompose the eigenvector $\mathbf{z}$ in a similar fashion, we have $\mathbf{z} = \mathbf{x} + i\mathbf{y}$. The original eigenvalue problem becomes

$$(\mathbf{B} + i\mathbf{C})(\mathbf{x} + i\mathbf{y}) = \lambda(\mathbf{x} + i\mathbf{y}), \tag{5.72}$$

which is equivalent to

$$\begin{pmatrix} \mathbf{B} & -\mathbf{C} \\ \mathbf{C} & \mathbf{B} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}. \tag{5.73}$$

This is a real symmetric eigenvalue problem with the same set of eigenvalues that have an overall double degeneracy. Therefore we need to solve only the real symmetric eigenvalue problem if the matrix is Hermitian.

The matrix used in similarity transformation of a real symmetric matrix to a diagonal matrix is orthogonal instead of unitary. This simplifies the procedure considerably. The problem can be simplified further if we separate the procedure into two steps:

(1) use an orthogonal matrix to perform a similarity transformation of the real symmetric matrix defined in Eq. (5.73) into a real symmetric tridiagonal matrix;
(2) solve the eigenvalue problem of the resulting tridiagonal matrix.

Note that the similarity transformation preserves the eigenvalues of the original matrix, and the eigenvectors are the columns or rows of the orthogonal matrix used in the transformation. The most commonly used method for tridiagonalizing a real symmetric matrix is the *Householder method*. Sometimes the method is also called the *Givens method*. Here we give a brief description of the scheme. The tridiagonalization is achieved with a total of $n - 2$ consecutive transformations, each operating on a row and a column of the matrix. The transformations can be cast into a recursion

$$\mathbf{A}^{(k)} = \mathbf{O}_k^{\mathrm{T}} \mathbf{A}^{(k-1)} \mathbf{O}_k, \tag{5.74}$$

for $k = 1, 2, \ldots, n-2$, where $\mathbf{O}_k$ is an orthogonal matrix that works on the row elements with $i = k+2, \ldots, n$ of the kth column and the column elements with $j = k+2, \ldots, n$ of the kth row. The recursion begins with $\mathbf{A}^{(0)} = \mathbf{A}$. To be specific, we can write the orthogonal matrix as

$$\mathbf{O}_k = \mathbf{I} - \frac{1}{\eta_k} \mathbf{w}_k \mathbf{w}_k^{\mathrm{T}}, \tag{5.75}$$

where the lth component of the vector $\mathbf{w}_k$ is given by

$$w_{kl} = \begin{cases} 0 & \text{for } l \le k, \\ A_{kk+1}^{(k-1)} + \alpha_k & \text{for } l = k+1, \\ A_{kl}^{(k-1)} & \text{for } l \ge k+2, \end{cases} \tag{5.76}$$

with

$$\alpha_k = \pm \sqrt{\sum_{l=k+1}^{n} \left[A_{kl}^{(k-1)} \right]^2} \tag{5.77}$$

and

$$\eta_k = \alpha_k\left[\alpha_k + A_{kk+1}^{(k-1)}\right]. \tag{5.78}$$

Even though the sign of α_k is arbitrary in the above equations, it is always taken to be the same as that of $A_{kk+1}^{(k-1)}$ in practice in order to avoid any possible cancelation, which can make the algorithm ill-conditioned (with a zero denominator in $\mathbf{O}_k$). We can show that $\mathbf{O}_k$ defined above is the desired matrix, which is orthogonal and converts the row elements with $i = k+2, \ldots, n$ for the kth column and the column elements with $j = k+2, \ldots, n$ for the kth row of $\mathbf{A}^{(k-1)}$ to zero just by comparing the elements on the both sides of Eq. (5.74). We are not going to provide a program here for the Householder scheme for tridiagonalizing a real symmetric matrix because it is in all standard mathematical libraries and reference books, for example, Press *et al.* (2002).

After we obtain the tridiagonalized matrix, the eigenvalues can be found using one of the root search routines available. Note that the secular equation $|\mathbf{A} - \lambda\mathbf{I}| = 0$ is equivalent to a polynomial equation $p_n(\lambda) = 0$. Because of the simplicity of the symmetric tridiagonal matrix, the polynomial $p_n(\lambda)$ can be generated recursively with

$$p_i(\lambda) = (a_i - \lambda)p_{i-1}(\lambda) - b_{i-1}^2 p_{i-2}(\lambda), \tag{5.79}$$

where $a_i = A_{ii}$ and $b_i = A_{ii+1} = A_{i+1i}$. The polynomial $p_i(\lambda)$ is given from the submatrix of A_{jk} with $j, k = 1, 2, \ldots, i$ with the starting values $p_0(\lambda) = 1$ and $p_1(\lambda) = a_1 - \lambda$. Note that this recursion is similar to that of the orthogonal polynomials introduced in Section 2.2 and can be generated easily using the following method.

```
// Method to generate the polynomial for the secular
// equation of a symmetric tridiagonal matrix.

  public static double polynomial(double a[],
    double b[], double x, int i) {
    if (i==0) return 1;
     else if (i==1) return a[0]-x;
      else return (a[i-1]-x)*polynomial(a, b, x, i-1)
        -b[i-2]*b[i-2]*polynomial(a, b, x, i-2);
  }
```

In principle, we can use any of the root searching routines to find the eigenvalues from the secular equation as soon as the polynomial is generated. However, two properties associated with the zeros of $p_n(\lambda)$ are useful in developing a fast and accurate routine to obtain the eigenvalues of a symmetric tridiagonal matrix.

(1) All the roots of $p_n(\lambda) = 0$ lie in the interval $[-\|\mathbf{A}\|, \|\mathbf{A}\|]$, where

$$\|\mathbf{A}\| = \max\left\{\sum_{j=1}^{n} |A_{ij}|\right\}, \tag{5.80}$$

for $i = 1, 2, \ldots, n$, is the column modulus of the matrix. We can also use the row modulus of the matrix in the above statement. The row modulus of a matrix is given by

$$\|\tilde{\mathbf{A}}\| = \max\left\{\sum_{i=1}^{n} |A_{ij}|\right\}, \tag{5.81}$$

for $j = 1, 2, \ldots, n$.

(2) The number of roots for $p_n(\lambda) = 0$ with $\lambda \geq \lambda_0$ is given by the number of agreements of the signs of $p_j(\lambda_0)$ and $p_{j-1}(\lambda_0)$ for $j = 1, 2, \ldots, n$. If any of the polynomials, for example, $p_j(\lambda_0)$, is zero, the sign of the previous polynomial $p_{j-1}(\lambda_0)$ is assigned to that polynomial.

With the help of these properties, we can develop a quite simple but fast algorithm in connection with the bisection method discussed in Chapter 3 to obtain the eigenvalues of a real symmetric matrix. We outline this algorithm below.

We first evaluate the column modulus of the matrix with Eq. (5.80). This sets the boundaries for the eigenvalues. Note that each summation in Eq. (5.80) has only two or three terms, $|A_{ii-1}|$, $|A_{ii}|$, and $|A_{ii+1}|$. We can then start the search for the first eigenvalue in the region of $[-\|\mathbf{A}\|, \|\mathbf{A}\|]$. Because there is a total of n eigenvalues, we need to decide which of those are sought. For example, if we are only interested in the ground state, in other words, the lowest eigenvalue, we can design an algorithm to target it directly.

A specific scheme can be devised and altered, based on the following general procedure. We can bisect the maximum region of $[-\|\mathbf{A}\|, \|\mathbf{A}\|]$ and evaluate the signs of $p_i(0)$ for $i = 1, 2, \ldots, n$. Note that $p_0(\lambda) = 1$. Based on what we have discussed, we will know the number of eigenvalues lying within either $[-\|\mathbf{A}\|, 0]$ or $[0, \|\mathbf{A}\|]$. This includes any possible degeneracy. We can divide the two subintervals into four equal regions and check the signs of the polynomials at each bisected point. This procedure can be continued to narrow down the region where each eigenvalue resides. After l steps of bisection, each eigenvalue sought is narrowed down to a region with a size of $\|\mathbf{A}\|/2^{l-1}$. Note that we can work either on a specific eigenvalue, for example, the one associated with the ground state, or on a group of eigenvalues simultaneously. The errorbars are bounded by $\|\mathbf{A}\|/2^{l-1}$ after l steps of bisection. A more realistic estimate of the errorbar is obtained from the improvement of a specific eigenvalue at each step of bisection, as in the estimates, made in Chapter 3.

Because the Householder scheme for a real symmetric matrix is carried out in two steps, transformation of the original matrix into a tridiagonal matrix and solution of the eigenvalue problem of the tridiagonal matrix, we can design different algorithms to achieve each of these two steps separately, depending on the specific problems involved. Interested readers can find several of these algorithms in Wilkinson (1963; 1965). It is worth emphasizing again that a Hermitian matrix eigenvalue problem can be converted into a real symmetric matrix eigenvalue problem with the size of the matrix expanded to twice that of the original matrix

along each direction. This seems to be a reasonable approach in most cases, as long as the size of the matrix is not limited by the available resources.

Eigenvalues of general matrices

Even though most problems in physics are likely to be concerned with Hermitian matrices, there are still problems that require us to deal with general matrices. Here we discuss briefly how to obtain the eigenvalues of a general nondefective matrix. A matrix is nondefective if it can be diagonalized under a matrix transformation and its eigenvectors can form a complete vector space. Here we consider nondefective matrices only, Because there is always a well-defined eigenvalue equation $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ for a nondefective matrix $\mathbf{A}$, and we seldom encounter defective matrices in physics.

In most cases, we can define a matrix function $f(\mathbf{A})$ with $\mathbf{A}$ playing the same role as x in $f(x)$. For example,

$$f(\mathbf{A}) = e^{-\alpha\mathbf{A}} = \sum_{k=0}^{\infty} \frac{(-\alpha\mathbf{A})^k}{k!} \tag{5.82}$$

is similar to

$$f(x) = e^{-\alpha x} = \sum_{k=0}^{\infty} \frac{(-\alpha x)^k}{k!}, \tag{5.83}$$

except that the power of a matrix is treated as matrix multiplications with $\mathbf{A}^0 = \mathbf{I}$. When the matrix function operates on an eigenvector of the matrix, the matrix can be replaced by the corresponding eigenvalue, such as

$$f(\mathbf{A})\mathbf{x}_i = f(\lambda_i)\mathbf{x}_i, \tag{5.84}$$

where $\mathbf{x}_i$ and λ_i satisfy $\mathbf{A}\mathbf{x}_i = \lambda_i\mathbf{x}_i$. If the function involves the inverse of the matrix and the eigenvalue happens to be zero, we can always add a term $\eta\mathbf{I}$ to the original matrix to remove the singularity. The modified matrix has the eigenvalue $\lambda_i - \eta$ for the corresponding eigenvector $\mathbf{x}_i$. The eigenvalue of the original matrix is recovered by taking $\eta \to 0$ after the problem is solved. Based on this property of nondefective matrices, we can construct a recursion

$$\mathbf{x}^{(k)} = \frac{1}{\sqrt{\mathcal{N}_k}}(\mathbf{A} - \mu\mathbf{I})^{-1}\mathbf{x}^{(k-1)} \tag{5.85}$$

to extract the eigenvalue that is closest to the parameter μ. Here $k = 1, 2, \ldots$, and $\mathcal{N}_k$ is a normalization constant to ensure that

$$\left\langle \mathbf{x}^{(k)} \middle| \mathbf{x}^{(k)} \right\rangle = \left(\mathbf{x}^{(k)}\right)^{\dagger}\mathbf{x}^{(k)} = 1. \tag{5.86}$$

To analyze this recursive procedure, let us express the trial state $\mathbf{x}^{(0)}$ in terms of a linear combination of all the eigenstates with

$$\mathbf{x}^{(0)} = \sum_{i=1}^{n} a_i^{(0)}\mathbf{x}_i, \tag{5.87}$$

which is always possible, because the eigenstates form a complete vector space. If we substitute the above trial state into the recursion, we have

$$\mathbf{x}^{(k)} = \frac{1}{\sqrt{\prod_{j=1}^{k} \mathcal{N}_j}} \sum_{i=1}^{n} \frac{a_i^{(0)} \mathbf{x}_i}{(\lambda_i - \mu)^k} = \sum_{i=1}^{n} a_i^{(k)} \mathbf{x}_i. \tag{5.88}$$

If we normalize the state at each step of iteration, only the state $\mathbf{x}_j$, with the eigenvalue λ_j that is the closest to μ, survive after a large number of iterations. All other coefficients vanish approximately as

$$a_i^{(k)} \sim \left(\frac{\lambda_j - \mu}{\lambda_i - \mu} \right)^k \frac{a_i^{(0)}}{a_j^{(0)}} \tag{5.89}$$

for $i \neq j$, after k iterations. However, $a_j^{(k)}$ will grow with k to approach 1. The corresponding eigenvalue is obtained with

$$\lambda_j = \mu + \lim_{k \to \infty} \frac{1}{\sqrt{\mathcal{N}_k}} \frac{x_l^{(k-1)}}{x_l^{(k)}}, \tag{5.90}$$

where l is the index for a specific nonzero component (usually the one with the maximum magnitude) of $\mathbf{x}^{(k)}$ or $\mathbf{x}^{(k-1)}$.

Here $\mathbf{x}^{(0)}$ is a trial state, which should have a nonzero overlap with the eigenvector $\mathbf{x}_j$. We have to be very careful to make sure the overlap is nonzero. Otherwise, we end up with the eigenvalue that belongs to the eigenvector with a nonzero overlap with $\mathbf{x}^{(0)}$ but which is still closer to μ than the rest. One way to avoid a zero overlap of $\mathbf{x}^{(0)}$ with $\mathbf{x}_j$ is to generate each component of $\mathbf{x}^{(0)}$ with a random-number generator and check each result with at least two different trial states.

The method outlined here is usually called the *inverse iteration method*. It is quite straightforward if we have an efficient algorithm to perform the matrix inversion. We can also rewrite the recursion at each step as a linear equation set

$$\sqrt{\mathcal{N}_k}(\mathbf{A} - \mu \mathbf{I})\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)}, \tag{5.91}$$

which can be solved with Gaussian elimination, for example, from one iteration to another. Note that this iterative scheme also solves the eigenvectors of the matrix at the same time. This is a very useful feature in many applications for which the eigenvectors and eigenvalues are both needed.

Eigenvectors of matrices

We sometimes also need the eigenvectors of an eigenequation. For a nondefective matrix, we can always obtain a complete set of eigenvectors, including the degenerate eigenvalue cases.

First, if the matrix is symmetric, it is much easier to obtain its eigenvectors. In principle, we can transform a real symmetric matrix $\mathbf{A}$ into a tridiagonal matrix $\mathbf{T}$ with a similarity transformation

$$\mathbf{T} = \mathbf{O}^{\mathrm{T}} \mathbf{A} \mathbf{O}, \tag{5.92}$$

where $\mathbf{O}$ is an orthogonal matrix. The eigenvalue equation then becomes

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} = \mathbf{O}\mathbf{T}\mathbf{O}^{\mathrm{T}}\mathbf{x}, \tag{5.93}$$

which is equivalent to solving $\mathbf{y}$ from

$$\mathbf{T}\mathbf{y} = \lambda\mathbf{y} \tag{5.94}$$

and then $\mathbf{x}$ is obtained from

$$\mathbf{x} = \mathbf{O}\mathbf{y}. \tag{5.95}$$

Thus, we can develop a practical scheme that solves the eigenvectors of $\mathbf{T}$ first and then those of $\mathbf{A}$ by multiplying $\mathbf{y}$ with $\mathbf{O}$, which is a byproduct of any tridiagonalization scheme, such as the Householder scheme.

As we discussed earlier, the recursion

$$\sqrt{\mathcal{N}_k}(\mathbf{T} - \mu\mathbf{I})\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} \tag{5.96}$$

will lead to both the eigenvector $\mathbf{y}_j$ and the eigenvalue λ_j that is closest to μ. Equation (5.96) can be solved with an elimination scheme. Note that the elimination scheme is now extremely simple because there are no more than three nonzero elements in each row. In every step of the above recursion, we need to normalize the resulting vector. We can use the LU decomposition in solving a linear equation set with a tridiagonal coefficient matrix, as given in detail in Section 2.4.

If the eigenvalue corresponds to more than one eigenvector, we can obtain the other vectors simply by changing the value of μ or the initial guess of $\mathbf{y}^{(0)}$. When all the vectors $\mathbf{x}_k$ corresponding to the same eigenvalue are found, we can transform them into an orthogonal set $\mathbf{z}_k$ with

$$\mathbf{z}_k = \mathbf{x}_k - \sum_{j=1}^{k-1}\langle\mathbf{z}_j|\mathbf{x}_k\rangle\mathbf{z}_j = \mathbf{x}_k - \sum_{j=1}^{k-1}\left(\mathbf{z}_j^{\mathrm{T}}\mathbf{x}_k\right)\mathbf{z}_j, \tag{5.97}$$

which is known as the Gram–Schmidt orthogonalization procedure.

If the tridiagonal matrix is obtained by means of the Householder scheme, we can also obtain the eigenvectors of the original matrix at the same time using

$$\begin{aligned}\mathbf{x} &= \mathbf{O}_1\mathbf{O}_2\cdots\mathbf{O}_{n-2}\mathbf{y}\\ &= \left(\mathbf{I} - \frac{1}{\eta_1}\mathbf{w}_1\mathbf{w}_1^{\mathrm{T}}\right)\cdots\left(\mathbf{I} - \frac{1}{\eta_{n-2}}\mathbf{w}_{n-2}\mathbf{w}_{n-2}^{\mathrm{T}}\right)\mathbf{y},\end{aligned} \tag{5.98}$$

which can be carried out in a straightforward fashion if we realize that

$$\left(\mathbf{I} - \frac{1}{\eta_k}\mathbf{w}_k\mathbf{w}_k^{\mathrm{T}}\right)\mathbf{y} = \mathbf{y} - \beta_k\mathbf{w}_k, \tag{5.99}$$

with $\beta_k = \mathbf{w}_k^{\mathrm{T}}\mathbf{y}/\eta_k$. Note that the first k elements in $\mathbf{w}_k$ are all zero.

For a general nondefective matrix, we can also follow a similar scheme to obtain the eigenvectors of the matrix. First we transform the matrix into an upper Hessenberg matrix. A matrix with all the elements below (above) the first

off-diagonal elements equal to zero is called an upper (lower) Hessenberg matrix. This transformation can always be achieved for a nondefective matrix $\mathbf{A}$ with

$$\mathbf{H} = \mathbf{U}^{\dagger}\mathbf{A}\mathbf{U} \tag{5.100}$$

under the unitary matrix $\mathbf{U}$. Here $\mathbf{H}$ is in the form of a Hessenberg matrix. There are several methods that can be used to reduce a matrix to a Hessenberg matrix; they are given in Wilkinson (1965). A typical scheme is similar to the Householder scheme for reducing a symmetric matrix to a tridiagonal matrix. We can then solve the eigenvalue problem of the Hessenberg matrix. The eigenvalue problem of an upper (lower) Hessenberg matrix is considerably simpler than that of a general matrix under a typical algorithm, such as the QR algorithm. The so-called QR algorithm splits a nonsingular matrix $\mathbf{A}$ into a product of a unitary matrix $\mathbf{Q}$ and an upper-triangular matrix $\mathbf{R}$ as

$$\mathbf{A} = \mathbf{Q}\mathbf{R}. \tag{5.101}$$

We can construct a series of similarity transformations by alternating the order of $\mathbf{Q}$ and $\mathbf{R}$ to reduce the original matrix to an upper-triangular matrix that has the eigenvalues of the original matrix on the diagonal. Assume that $\mathbf{A}^{(0)} = \mathbf{A}$ and

$$\mathbf{A}^{(k)} = \mathbf{Q}_k\mathbf{R}_k; \tag{5.102}$$

then we have

$$\mathbf{A}^{(k+1)} = \mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^{\dagger}\mathbf{A}^{(k)}\mathbf{Q}_k = \mathbf{Q}_k^{\dagger}\cdots\mathbf{Q}_0^{\dagger}\mathbf{A}\mathbf{Q}_0\cdots\mathbf{Q}_k, \tag{5.103}$$

with $k = 0, 1, \ldots$. This algorithm works best if $\mathbf{A}$ is a Hessenberg matrix. Taking account of stability and computing speed, a Householder type of scheme to transform a nondefective matrix into a Hessenberg matrix seems to be an excellent choice (Wilkinson, 1965).

When the matrix has been transformed into the Hessenberg form, we can use the inverse iteration method to obtain the eigenvectors of the Hessenberg matrix $\mathbf{H}$ with

$$\sqrt{\mathcal{N}_k}(\mathbf{H} - \mu\mathbf{I})\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} \tag{5.104}$$

and then the eigenvectors of the original matrix $\mathbf{A}$ from

$$\mathbf{x} = \mathbf{U}\mathbf{y}. \tag{5.105}$$

Of course, the normalization of the vector at each iteration is assumed with the normalization constant $\mathcal{N}_k$, with a definition similar to that of $\mathcal{N}_k$ in Eq. (5.86), to ensure the convergence.

Many numerical packages are available for dealing with linear algebra and matrix problems, including JMSL, a collection of mathematical, statistical, and charting classes in Java by Visual Numerics Inc. We should understand the basics and then learn how to apply the routines from numerical packages in research. A good survey of the existing packages can be found in Heath (2002).

5.6 The Faddeev–Leverrier method

A very interesting method developed for matrix inversion and eigenvalue problem is the *Faddeev–Leverrier method*. The scheme was first discovered by Leverrier in the middle of the nineteenth century and later modified by Faddeev (Faddeev and Faddeeva, 1963). Here we give a brief discussion of the method. The characteristic polynomial of the matrix is given by

$$p_n(\lambda) = |\mathbf{A} - \lambda\mathbf{I}| = \sum_{k=0}^{n} c_k \lambda^k, \tag{5.106}$$

where $c_n = (-1)^n$. Then we can introduce a set of supplementary matrices $\mathbf{S}_k$ with

$$p_n(\lambda)(\lambda\mathbf{I} - \mathbf{A})^{-1} = \sum_{k=0}^{n-1} \lambda^{n-k-1}\mathbf{S}_k. \tag{5.107}$$

If we multiply the above equation by $(\lambda\mathbf{I} - \mathbf{A})$, we have

$$\sum_{k=0}^{n} c_k \lambda^k \mathbf{I} = \mathbf{S}_0\lambda^n + \sum_{k=1}^{n-1}(\mathbf{S}_k - \mathbf{A}\mathbf{S}_{k-1})\lambda^{n-k} - \mathbf{A}\mathbf{S}_{n-1}. \tag{5.108}$$

Comparing the coefficients from the same order of λ^l for $l = 0, 1, \ldots, n$ on the both sides of the equation, we obtain the recursion for c_{n-k} and $\mathbf{S}_k$,

$$c_{n-k} = -\frac{1}{k}\,\mathrm{Tr}\,\mathbf{A}\mathbf{S}_{k-1}, \tag{5.109}$$

$$\mathbf{S}_k = \mathbf{A}\mathbf{S}_{k-1} + c_{n-k}\mathbf{I}, \tag{5.110}$$

for $k = 1, 2, \ldots, n$. The recursion is started with $\mathbf{S}_0 = \mathbf{I}$ and is ended at c_0. From Eq. (5.107) with $\lambda = 0$, we can easily show that

$$\mathbf{A}\mathbf{S}_{n-1} + c_0\mathbf{I} = \mathbf{0}, \tag{5.111}$$

which can be used to obtain the inverse

$$\mathbf{A}^{-1} = -\frac{1}{c_0}\mathbf{S}_{n-1}. \tag{5.112}$$

The following program is an example of generating $\mathbf{S}_k$ and c_k for a given matrix $\mathbf{A}$ with an evaluation of $\mathbf{A}^{-1}$ from $\mathbf{S}_{n-1}$ and c_0 as an illustration.

```
// An example of using the Faddeev-Leverrier method to
// obtain the inversion of a matrix.

import java.lang.*;
public class Faddeev {
  public static void main(String argv[]) {
    double a[][]= {{ 1,  3,  2},
                   {-2,  3, -1},
                   {-3, -1,  2}};
    int n = a.length;
    double c[] = new double[n];
    double d[][] = new double[n][n];
    double s[][][] = (double[][][]) fl(a, c);
```

```
    for (int i=0; i<n; ++i)
      for (int j=0; j<n; ++j)
        d[i][j] = -s[n-1][i][j]/c[0];
    for (int i=0; i<n; ++i)
      for (int j=0; j<n; ++j)
        System.out.println(d[i][j]);
  }

// Method to complete the Faddeev-Leverrier recursion.

  public static double[][][] fl(double a[][],
    double c[]) {
    int n = c.length;
    double s[][][] = new double[n][n][n];
    for (int i=0; i<n; ++i) s[0][i][i] = 1;
    for (int k=1; k<n; ++k) {
      s[k] = mm(a,s[k-1]);
      c[n-k] = -tr(s[k])/k;
      for (int i=0; i<n; ++i)
      s[k][i][i] += c[n-k];
    }
      c[0] = -tr(mm(a,s[n-1]));
    return s;
  }

// Method to calculate the trace of a matrix.

  public static double tr(double a[][]) {
    int n = a.length;
    double sum = 0;
    for (int i=0; i<n; ++i) sum += a[i][i];
    return sum;
  }

// Method to evaluate the product of two matrices.

  public static double[][] mm(double a[][],
    double b[][]) {
    int n = a.length;
    double c[][] = new double[n][n];
    for (int i=0; i<n; ++i)
      for (int j=0; j<n; ++j)
        for (int k=0; k<n; ++k)
          c[i][j] += a[i][k]*b[k][j];
    return c;
  }
}
```

After running the above program, we obtain the inverse of the matrix. Note that each method in the example program above is kept general enough to deal with any real matrix.

Because the Faddeev–Leverrier recursion also generates all the coefficients c_k for the characteristic polynomial $p_n(\lambda)$, we can use a root search method to obtain all the eigenvalues from $p_n(\lambda) = 0$. This is the same as the situation for the Householder method discussed in the preceding section. We will further explore the zeros of a general real polynomial, including complex ones, in the next section.

After we have found all the eigenvalues λ_k, we can also obtain the corresponding eigenvectors with the availability of the supplementary matrices $\mathbf{S}_k$. If we define a new matrix

$$\mathbf{X}(\lambda_k) = \sum_{l=1}^{n} \mathbf{S}_{l-1} \lambda_k^{n-l}, \tag{5.113}$$

then the columns of $\mathbf{X}(\lambda_k)$ are the eigenvectors $\mathbf{x}_k$. This can be shown from the limit of the matrix $\mathbf{X}(\lambda)$ with $\lambda = \lambda_k + \delta$. Here $\delta \to 0$ is introduced to make $\mathbf{X}$ a nonsingular matrix. From Eq. (5.107), we can easily show that

$$\mathbf{X}(\lambda) = |\lambda \mathbf{I} - \mathbf{A}|(\lambda \mathbf{I} - \mathbf{A})^{-1}, \tag{5.114}$$

which leads each column of the matrix to the eigenvector $\mathbf{x}_k$ as $\delta \to 0$. The advantage of this scheme over the iterative scheme discussed in Section 5.5 is that we do not need to perform iterations as soon as we have the supplementary matrices $\mathbf{S}_k$. Note that we have not specified anything about the matrix $\mathbf{A}$ except that it is nonsingular. Therefore, the Faddeev–Leverrier method can be used for general matrices as well as symmetric matrices. However, the scheme can be time consuming if the dimension of the matrix is large.

5.7 Complex zeros of a polynomial

In scientific computing, we deal with problems mainly involving real symmetric or Hermitian matrices, which always have real eigenvalues. We rarely encounter a general complex matrix problem. If we do, we can still split the matrix into its real and imaginary parts, and then the problem becomes a general problem with two coupled real matrix equations. So occasionally we need to work with a general real matrix that may have complex eigenvalues and eigenvectors. If we can find the complex eigenvalues, we can also find the corresponding eigenvectors either by the inverse iteration method or the Faddeev–Leverrier method. Then how can we find all the eigenvalues of a general real matrix?

In principle, the eigenvalues of an $n \times n$ real matrix $\mathbf{A}$ are given by the zeros of the polynomial

$$p_n(\lambda) = |\mathbf{A} - \lambda \mathbf{I}| = \sum_{k=0}^{n} c_k \lambda^k, \tag{5.115}$$

The coefficients c_k are real, because $\mathbf{A}$ is real, with $c_n = (-1)^n$; the polynomial has a total of n zeros. We have discussed how to obtain all the coefficients c_k with the Faddeev–Leverrier method in the preceding section.

Here we explore some major properties of such a polynomial, especially its zeros, which can be complex even though all c_k are real. Note that if a complex value $z_0 = x_0 + iy_0$ for $y_0 \neq 0$ is a zero of the polynomial, its complex conjugate $z_0^* = x_0 - iy_0$ is also a zero. For a polynomial of degree n, that is, $c_n \neq 0$, there is a total of n zeros. This means that a polynomial with an odd n must have at least one real zero.

Locations of the zeros

In principle, we can rewrite the polynomial as a product

$$p_n(z) = \sum_{k=0}^{n} c_k z^k = c_n \prod_{k=1}^{n} (z - z_k), \tag{5.116}$$

where z_k is the kth zero of the polynomial, or the kth root of the equation $p_n(z) = 0$. If we take $z = 0$ in the above equation, we reach

$$\prod_{k=1}^{n} z_k = (-1)^n \frac{c_0}{c_n}. \tag{5.117}$$

We can force $|c_n| = |c_0| = 1$ by first dividing the polynomial by c_0 and then using z to denote $|c_n/c_0|^{1/n} z$. Then we have, from the above equation, some of the zeros residing inside the unit circle on the complex z plane, centered at the origin of $z = x + iy$. The rest of zeros are outside the unit circle. Thus we can use, for example, the bisection method to find the corresponding real and imaginary parts (or the amplitudes and phases) of the zeros inside the unit circle easily.

For the zeros outside the unit circle, we can further change the variable z to $1/z$ and then multiply the polynomial by z^n. The new polynomial has the coefficients c_k changed to c_{n-k}, for $k = 0, 1, \ldots, n$, and has zeros that are the inverses of the zeros of the original polynomial. Thus we can find the zeros of the original polynomial outside the unit circle by searching for the zeros of the new polynomial inside the unit circle.

The evaluation of a polynomial is necessary for any search scheme that looks for zeros within the unit circle. We can evaluate the polynomial efficiently by realizing that

$$\begin{aligned} p_n(z) &= u_n(x, y) + i v_n(x, y) \\ &= c_0 + z\{c_1 + z[c_2 + \cdots + (c_{n-1} + z c_n) \cdots]\}, \end{aligned} \tag{5.118}$$

where u_n and v_n are the real and imaginary parts of p_n, respectively. Then we can construct the recursion

$$u_k = c_{n-k} + x u_{k-1} - y v_{k-1}, \tag{5.119}$$

$$v_k = x v_{k-1} + y u_{k-1}, \tag{5.120}$$

starting with $u_0 = c_n$ and $v_0 = 0$. The following method implements such a recursive evaluation of p_n.

```
// Method to evaluate the polynomial with given c_k and
// z=x+iy.

  public static double[] polynomial2(double c[],
    double x, double y) {
    double p[] = new double[2];
    int n = c.length-1;
    double u = c[n];
    double v = 0;
    for (int i=0; i<n; ++i) {
```

```
      double t = x*v+y*u;
      u = c[n-i-1]+x*u-y*v;
      v = t;
    }
    p[0] = u;
    p[1] = v;
    return p;
  }
```

In order to find a zero of $p_n(z)$, we effectively have to solve two equations, $u_n(x, y) = 0$ and $v_n(x, y) = 0$, simultaneously. This can be done, for example, with the multivariable Newton method discussed in Section 5.4.

A related issue is to evaluate the coefficients of a polynomial with its imaginary axis shift by an amount x_0, that is,

$$q_n(z) = p_n(z - x_0) = \sum_{k=0}^{n} d_k z^k, \tag{5.121}$$

where the new coefficients d_n are given by

$$d_k = c_k - x_0 d_{k+1}, \tag{5.122}$$

for $k = n, n-1, \ldots, 0$. Note that x_0 is real and we need $d_{n+1} = 0$ to start the recursion.

For a polynomial obtained from the secular equation of a real matrix, the zeros are in general bounded by the column or row modulus of the matrix on the z plane. For example, if we use the row modulus, we have

$$|z_k| \leq \|\mathbf{A}\|, \tag{5.123}$$

which provides a circle that circumscribes all the eigenvalues of the matrix. After we obtain the coefficients c_k, for example, with the Faddeev–Leverrier method, we can conduct an exhaustive search of all the eigenvalues within the circle of $|z| < \|\mathbf{A}\|$. This method of searching for all the eigenvalues is primitive and can be slow if n is large. If we want a more efficient scheme for searching for the zeros of the polynomial or the eigenvalues of the corresponding matrix, we must develop some new ideas.

Factoring a polynomial

A better strategy is to divide the original polynomial by a linear function or a quadratic function, and then search for the zeros of the polynomial by turning the coefficients of the remainders to zero. This is more efficient because we can obtain the zero (or a pair of zeros) and reduce the polynomial to its quotient for further search at the same time.

For example, if we divide the polynomial $p_n(z)$ by a divisor that is a first-order polynomial $f_1(z) = \alpha + \beta z$, we have the quotient

$$p_{n-1}(z) = p_n(x)/f_1(x) - r_0 = \sum_{k=0}^{n-1} d_k z^k, \tag{5.124}$$

where d_k is given by

$$d_{k-1} = (c_k + \alpha d_k)/\beta, \tag{5.125}$$

for $k = n, n-1, \ldots, 1$. Note that we take $d_n = 0$ to start the recursion, and the remainder is given by $r_0 = c_0 - \alpha d_0$. We can easily implement this factoring scheme to locate real zeros. Note that we can use the sign of the remainder to narrow down the region. Consider two points x_1 and $x_2 > x_1$ on the real axis and use $f_1(z) = z - x_1$ to obtain the remainder $r_0(x_1)$ and $f_1(z) = z - x_2$ to obtain the remainder $r_0(x_2)$. If we have $r_0(x_1)r_0(x_2) < 0$, we know that there is at least one real zero $z_0 = x_0$ in the region $[x_1, x_2]$. A combination of the bisection method and the above factoring scheme can locate a real zero of $p_n(z)$ quickly. This process can be continued with a search for a zero in the quotient, the quotient of the quotient, ... , until we have exhausted all the real zeros of the polynomial $p_n(z)$.

To divide the polynomial $p_n(z)$ by $f_2(z) = \alpha + \beta z + \gamma z^2$, we have the quotient

$$p_{n-2}(z) = p_n(x)/f_2(x) - (r_0 + r_1 z) = \sum_{k=0}^{n-2} e_k z^k, \tag{5.126}$$

where e_k is given by

$$e_{k-2} = (c_k - \alpha e_k - \beta e_{k-1})/\gamma, \tag{5.127}$$

for $k = n, n-1, \ldots, 1$. Note that we take $e_n = e_{n-1} = 0$ to start the recursion, and the remainder coefficients are given by $r_1 = c_1 - \alpha e_1 - \beta e_0$ and $r_0 = c_0 - \alpha e_0$.

The above factoring scheme can be applied in the search for a pair of complex zeros in the polynomial $p_n(z)$ by simultaneously solving $r_0(\alpha, \beta) = 0$ and $r_1(\alpha, \beta) = 0$ if we set $\gamma = 1$ in $f_2(z)$. One can use the discrete Newton method introduced in Section 5.4 to accomplish the root search. After we locate a root with parameters $\alpha = \alpha_0$ and $\beta = \beta_0$, we can relate them back to the pair of zeros of $p_n(z)$ as $\left(-\beta_0 \pm i\sqrt{4\alpha_0 - \beta_0^2}\right)/2$.

We can also work out the analytical expressions for the partial derivatives $\partial r_i/\partial\alpha$ and $\partial r_i/\partial\beta$ for $i = 0, 1$, and then apply the Newton method to solve the two equations efficiently. This scheme is called the *Bairstow method*. For a detailed discussion of the Bairstow method, see Wilkinson (1965).

The Routh–Hurwitz test

An interesting scheme for testing the real parts of the zeros of a polynomial $p_n(z)$ is called the *Routh–Hurwitz test*. The scheme uses the coefficients c_k to construct a sequence whose signs determine how many zeros are on the right-hand side of the imaginary axis. Here is how to construct the sequence.

We can build an $(n+1) \times m$ matrix with

$$B_{ij} = \frac{1}{B_{i+11}} \begin{vmatrix} B_{i+11} & B_{i+1j+1} \\ B_{i+21} & B_{i+2j+1} \end{vmatrix}, \tag{5.128}$$

for $i = n-1, n-2, \ldots, 1$ and $j = 1, 2, \ldots, m$, where

$$B_{n+1j} = c_{n-2(j-1)}, \tag{5.129}$$

$$B_{nj} = c_{n-2j+1}, \tag{5.130}$$

with $m = 1 + n/2$ if n is even and $m = (n+1)/2$ if n is odd. Note that if c_{-1} or any B_{ij} outside the range of the matrix shows up, it is treated as 0. Then we count how many sign agreements that we have between B_{i1} and B_{i+11} for $i = 1, 2, \ldots, n$, which is the number of zeros on the right-hand side of the imaginary axis.

There are a couple of problems in the above recursion. First, if $B_{i+11} = 0$, the recursion cannot go on. We can cure this by multiplying the polynomial by $z - x_0$ with $x_0 < 0$. This does not change the zeros in the original polynomial but merely adds another real zero to the polynomial on the left-hand side of the imaginary axis.

The second problem is when a row of the matrix turns out to be zero. This terminates the recursion prematurely. This problem arises from the elements from the previous row (one line lower on the matrix); they happen to be the coefficients of an exact divisor with a zero remainder. We can cure this by replacing the zeros with the coefficients from the first-order derivative of the exact divisor. The highest order of the exact divisor is determined by the index of the row that is zero. For example, if $B_{kj} = 0$ for all j, the divisor is $d(x) = B_{k+11}x^k + B_{k+12}x^{k-2} + B_{k+13}x^{k-4} + \cdots$.

The Routh–Hurwitz test provides an efficient and powerful method of searching for zeros and of factoring the polynomial if we combine it with the shift operation discussed earlier and the bisection search along the imaginary axis.

5.8 Electronic structures of atoms

As we have pointed out, there are many applications of matrix operations in physics. We have given a few examples in Section 5.1 and a concrete case study on multicharge clusters in Section 5.4. In this section, we demonstrate how to apply the matrix eigenvalue schemes in the calculation of the electronic structures of many-electron atomic systems within the framework of the Hartree–Fock approximation.

The Schrödinger equation for a multielectron atom is given by

$$\mathcal{H}\Psi_k(\mathbf{R}) = \mathcal{E}_k\Psi_k(\mathbf{R}) \tag{5.131}$$

for $k = 0, 1, \ldots,$ where $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ is a $3N$-dimensional position vector for all N electrons in the system, $\mathcal{H}$ is the Hamiltonian of the N electrons in the system, given by

$$\mathcal{H} = -\frac{\hbar^2}{2m_\mathrm{e}} \sum_{i=1}^{N} \nabla_i^2 - \frac{Ze^2}{4\pi\epsilon_0} \sum_{i=1}^{N} \frac{1}{r_i} + \frac{e^2}{4\pi\epsilon_0} \sum_{i>j}^{N} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}, \tag{5.132}$$

and $\mathcal{E}_k$ is the kth eigenvalue of the the Hamiltonian. Here Z is the atomic number, ϵ_0 is the electric permittivity of vacuum, m_e is the mass of a free electron, and e is the fundamental charge. In general, it is not possible to obtain an exact solution of the multielectron Schrödinger equation; approximations must be made in order to solve the underlying eigenvalue problem. To simplify the expressions, we use atomic units, that is, $m_\mathrm{e} = e = 4\pi\epsilon_0 = \mu_0/4\pi = \hbar = 1$, where μ_0 is the magnetic permeability of vacuum. Under these choices of units, lengths are given in the Bohr radius, $a_0 = 4\pi\epsilon_0\hbar^2/m_\mathrm{e}e^2 = 0.529\,177\,249(24) \times 10^{-10}$ m, and energies are given in the hartree, $e^2/4\pi\epsilon_0 a_0 = 27.211\,396\,1(81)$ eV.

The Hartree–Fock approximation assumes that the ground state of a system of fermions (electrons in this case) can be viewed as occupying the lowest set of certain single-particle states after considering the Pauli exclusion principle. Then the ground state is approximated by the *Hartree–Fock ansatz*, which can be cast into a determinant

$$\Psi_\mathrm{HF}(\mathbf{R}) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \phi_1(\mathbf{r}_1) & \phi_1(\mathbf{r}_2) & \cdots & \phi_1(\mathbf{r}_N) \\ \phi_2(\mathbf{r}_1) & \phi_2(\mathbf{r}_2) & \cdots & \phi_2(\mathbf{r}_N) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_N(\mathbf{r}_1) & \phi_N(\mathbf{r}_2) & \cdots & \phi_N(\mathbf{r}_N) \end{vmatrix}; \tag{5.133}$$

this is also known as the *Slater determinant*. The index used above includes both the spatial and spin indices. In most cases, it is more convenient to write the single particle states as $\phi_{i\sigma}(\mathbf{r})$ with the spin index σ ($\uparrow$ or $\downarrow$) separated from the spatial index i.

Because $\mathcal{E}_0$ is the ground-state energy of the system, we must have

$$\mathcal{E}_\mathrm{HF} = \frac{\langle \Psi_\mathrm{HF} | \mathcal{H} | \Psi_\mathrm{HF} \rangle}{\langle \Psi_\mathrm{HF} | \Psi_\mathrm{HF} \rangle} \geq \mathcal{E}_0. \tag{5.134}$$

To optimize (minimize) $\mathcal{E}_\mathrm{HF}$, we perform the functional variation with respect to $\phi_{i\sigma}^\dagger(\mathbf{r})$; then we have

$$\left[-\frac{1}{2}\nabla^2 - \frac{Z}{r} + V_\mathrm{H}(\mathbf{r}) \right] \phi_{i\sigma}(\mathbf{r}) - \int V_{\mathrm{x}\sigma}(\mathbf{r}', \mathbf{r}) \phi_{i\sigma}(\mathbf{r}')\, d\mathbf{r}' = \varepsilon_i \phi_{i\sigma}(\mathbf{r}), \tag{5.135}$$

which is known as the Hartree–Fock equation (see Exercise 5.15). Here $V_\mathrm{H}(\mathbf{r})$ is the Hartree potential given by

$$V_\mathrm{H}(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}\, d\mathbf{r}', \tag{5.136}$$

where $\rho(\mathbf{r}) = \rho_\uparrow(\mathbf{r}) + \rho_\downarrow(\mathbf{r})$ is the total density of the electrons at $\mathbf{r}$. The exchange interaction $V_{\mathrm{x}\sigma}(\mathbf{r}, \mathbf{r}')$ is given by

$$V_{\mathrm{x}\sigma}(\mathbf{r}', \mathbf{r}) = \frac{\rho_\sigma(\mathbf{r}, \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}, \tag{5.137}$$

where $\rho_\sigma(\mathbf{r}, \mathbf{r}')$ is the density matrix of the electrons, given by

$$\rho_\sigma(\mathbf{r}, \mathbf{r}') = \sum_{i=1}^{N_\sigma} \phi_{i\sigma}^\dagger(\mathbf{r})\phi_{i\sigma}(\mathbf{r}'). \tag{5.138}$$

Here N_σ is the total number of states occupied by the electrons with spin σ. The density of the electrons is the diagonal of the density matrix, that is, $\rho_\sigma(\mathbf{r}) = \rho_\sigma(\mathbf{r}, \mathbf{r})$. The eigenvalue ε_i above is a multiplier introduced during the optimization (minimization) of $\mathcal{E}_{\mathrm{HF}}$.

The Hartree potential can also be obtained from the solution of the Poisson equation

$$\nabla^2 V_{\mathrm{H}}(\mathbf{r}) = -4\pi\rho(\mathbf{r}). \tag{5.139}$$

The single-particle wavefunctions in the atomic systems can be assumed to have the form

$$\phi_{i\sigma}(\mathbf{r}) = \frac{1}{r} u_{nl\sigma}(r) Y_{lm}(\theta, \phi), \tag{5.140}$$

where $Y_{lm}(\theta, \phi)$ are the spherical harmonics with θ and ϕ being the polar and azimuthal angles, respectively. Quantum numbers n, l, and m correspond to the energy, angular momentum, and z component of the angular momentum.

We can make the further assumption that the electron density is spherically symmetric and divide the space $[0, r_{\mathrm{c}}]$ into discrete points with an evenly spaced interval h. Here r_{c} is the cut-off in the radial direction, typically a few Bohr radii. The spherical approximation of the density of electrons is valid only if the formation of the magnetic moment is not considered. If we use a numerical expression for the second-order derivative, for example, the three-point formula, the Hartree–Fock equation for a given l is converted into a matrix equation

$$\mathbf{H}\mathbf{u} = \varepsilon\mathbf{u}, \tag{5.141}$$

with the diagonal elements of $\mathbf{H}$ given by

$$H_{ii} = \frac{1}{h^2} + \frac{l(l+1)}{2r_i^2} - \frac{Ze^2}{r_i} + V_{\mathrm{H}}(r_i), \tag{5.142}$$

and the corresponding off-diagonal elements given by

$$H_{ij} = -\frac{\delta_{ij\pm1}}{2h^2} + hV_{\mathrm{x}\sigma}^{(l)}(r_i, r_j), \tag{5.143}$$

where $V_{\mathrm{x}\sigma}^{(l)}(r_i, r_j)$ is the lth component of $V_{\mathrm{x}\sigma}(\mathbf{r}_i, \mathbf{r}_j)$ expanded in spherical harmonics. We have used $\mathbf{H}$ for the matrix form of $\mathcal{H}$ and $\mathbf{u}$ for the discrete form

of the wavefunction $u_{nl\sigma}(r)$. The angular momentum index is suppressed in $\mathbf{H}$ and $\mathbf{u}$ for convenience. We can easily apply the numerical schemes introduced in Section 5.5 to solve this matrix eigenvalue problem. The energy levels for different n are obtained for a fixed l. We can, of course, evaluate the density matrix and the Hartree–Fock ground-state energy with the method described here.

5.9 The Lanczos algorithm and the many-body problem

One of the most powerful methods in large-scale matrix computing is the *Lanczos method*, which is an iterative scheme suitable for large, especially sparse (most elements are zero) matrices. The advantage of the Lanczos method is extremely noticeable when we only need a few eigenvalues and eigenvectors, or when the system is extremely large and sparse. Here we just sketch a very basic Lanczos algorithm. More elaborate discussions on various Lanczos methods can be found in several specialized books, for example, Wilkinson (1965), Cullum and Willoughby (1985), and Hackbusch (1994).

Assuming that the matrix $\mathbf{H}$ is an $n \times n$ real symmetric matrix, we can tridiagonalize an $m \times m$ subset of $\mathbf{H}$ with

$$\mathbf{O}^{\mathrm{T}}\mathbf{H}\mathbf{O} = \tilde{\mathbf{H}}, \tag{5.144}$$

where $\mathbf{O}$ is an $n \times m$ matrix with its kth column given by

$$\mathbf{v}_k = \frac{\mathbf{u}_k}{\sqrt{\mathcal{N}_k}}, \tag{5.145}$$

for $k = 1, 2, \ldots, m$, where $\mathcal{N}_k = \mathbf{u}_k^{\mathrm{T}}\mathbf{u}_k$ is the normalization constant and the vectors $\mathbf{u}_k$ are generated recursively from an arbitrary vector $\mathbf{u}_1$ as

$$\mathbf{u}_{k+1} = \mathbf{H}\mathbf{v}_k - \alpha_k\mathbf{v}_k - \beta_k\mathbf{v}_{k-1}, \tag{5.146}$$

with $\beta_k = \tilde{H}_{k-1k} = \mathbf{v}_{k-1}^{\mathrm{T}}\mathbf{H}\mathbf{v}_k$ and $\alpha_k = \tilde{H}_{kk} = \mathbf{v}_k^{\mathrm{T}}\mathbf{H}\mathbf{v}_k$. The recursion is started at $\beta_1 = 0$ with $\mathbf{u}_1$ being a selected vector. Note that $\mathbf{u}_1$ can be a normalized vector with each element generated from a uniform random-number generator, for example. In principle, the vectors $\mathbf{v}_k$, for $k = 1, 2, \ldots, m$, form an orthonormal set, but in practice we still need to carry out the Gram–Schmidt orthogonalization procedure, at every step of the recursion, to remove the effects of rounding errors. We can show that the eigenvalues of the tridiagonal submatrix $\tilde{\mathbf{H}}$ are the approximations of the ones of $\mathbf{H}$ with the largest magnitudes. We can use the standard methods discussed earlier to diagonalize $\tilde{\mathbf{H}}$ to obtain its eigenvectors and eigenvalues from $\tilde{\mathbf{H}}\tilde{\mathbf{x}}_k = \lambda_k\tilde{\mathbf{x}}_k$. The eigenvectors $\tilde{\mathbf{x}}_k$ can be used to obtain the approximate eigenvectors of $\mathbf{H}$ with $\mathbf{x}_k \simeq \mathbf{O}\tilde{\mathbf{x}}_k$.

The approximation is improved if we construct a new initial state $\mathbf{u}_1$ from the eigenvectors $\tilde{\mathbf{x}}_k$ with $k = 1, 2, \ldots, m$, for example,

$$\mathbf{u}_1 = \sum_{k=1}^{m} c_k\tilde{\mathbf{x}}_k, \tag{5.147}$$

and then the recursion is repeated again and again. We can show that this iterative scheme will eventually lead to the m eigenvectors of $\mathbf{H}$, corresponding to the eigenvalues with the largest magnitudes. In practice, the selection of the coefficients c_k is rather important in order to have a fast and accurate algorithm. Later in this section we will introduce one of the choices made by Dagotto and Moreo (1985) in the study of the ground state of a quantum many-body system.

We can work out the eigenvalue problem for a specified region of the spectrum of $\mathbf{H}$ with the introduction of the matrix

$$\mathbf{G} = (\mathbf{H} - \mu\mathbf{I})^{-1}. \tag{5.148}$$

We can solve $\mathbf{G}$ with the Lanczos algorithm to obtain the eigenvectors with eigenvalues near μ. Note that

$$\mathbf{G}\mathbf{x}_k = \frac{1}{\lambda_k - \mu}\mathbf{x}_k \tag{5.149}$$

if $\mathbf{H}\mathbf{x}_k = \lambda_k\mathbf{x}_k$. This is useful if one wants to know about the spectrum of a particular region.

At the beginning of this chapter, we used a many-body Hamiltonian (the Hubbard model) in Eq. (5.17) to describe the electronic behavior of H_3^+. It is generally believed that the Hubbard model and its variants can describe the majority of highly correlated quantum systems, for example, transition metals, rare earth compounds, conducting polymers, and oxide superconducting materials. There are good reviews of the Hubbard model in Rasetti (1991). Usually, we want to know the properties of the ground state and the low-lying excited states. For example, if we want to know the ground state and the first excited state of a cluster of N sites with $N_0 < N$ electrons, we can solve the problem using the Lanczos method by taking $m \simeq 10$ and iterating the result a few times. Note that the number of many-body states increases exponentially with both N_0 and N. The iteration converges to the ground state and the low-lying excited state only if they have the largest eigenvalue magnitudes. The ground state and low-lying excited state energies carry the largest magnitudes if the chemical potential of the system is set to be zero. We also have to come up with a construction for the next guess of $\mathbf{u}_1$. We can, for example, use

$$\mathbf{u}_1^{(l+1)} = \sum_{k=1}^{5} \mathbf{v}_k^{(l)}. \tag{5.150}$$

A special choice of the iteration scheme for the ground-state properties is given by Dagotto and Moreo (1985). The $(l+1)$th iteration of $\mathbf{v}_1$ is taken as

$$\mathbf{v}_1^{(l+1)} = \frac{1}{\sqrt{1+a^2}}\left(\mathbf{v}_1^{(l)} + a\mathbf{v}_2^{(l)}\right), \tag{5.151}$$

where a is determined from the minimization of the expectation value of $\mathbf{H}$ under $\mathbf{v}_1^{(l+1)}$, which gives

$$a = b - \sqrt{1+b^2}, \tag{5.152}$$

where b is expressed in terms of the expectation values of the lth iteration as

$$b = \frac{d_3 - 3d_1d_2 + 2d_1^3}{2(d_2 - d_1^2)^{3/2}} \tag{5.153}$$

with $d_1 = \mathbf{v}_1^{\mathrm{T}}\mathbf{H}\mathbf{v}_1$, $d_2 = \mathbf{v}_1^{\mathrm{T}}\mathbf{H}^2\mathbf{v}_1$, and $d_3 = \mathbf{v}_1^{\mathrm{T}}\mathbf{H}^3\mathbf{v}_1$. The second vector

$$\mathbf{v}_2 = \frac{1}{\sqrt{d_2 - d_1^2}}(\mathbf{H}\mathbf{v}_1 - d_1\mathbf{v}_1) \tag{5.154}$$

is also normalized under such a choice of $\mathbf{v}_1$. The advantage of this algorithm is that we only need to store three vectors $\mathbf{v}_1$, $\mathbf{H}\mathbf{v}_1$, and $\mathbf{H}^2\mathbf{v}_1$ during each iteration. The eigenvalue with the largest magnitude is also given iteratively from

$$\lambda_1 = \mathbf{v}_1^{\mathrm{T}}\mathbf{H}\mathbf{v}_1 + \frac{a}{\sqrt{d_2 - d_1^2}}. \tag{5.155}$$

This algorithm is very efficient for calculating the ground-state properties of many-body quantum systems. For more discussions on the method and its applications, see Dagotto (1994).

5.10 Random matrices

The distributions of the energy levels of many physical systems have some universal features determined by the fundamental symmetry of the Hamiltonian. This type of feature is usually qualitative. For example, when disorders are introduced into metallic systems, the resistivities increase and the systems become insulators if the disorders are strong enough. For each metal, the degree of disorder needed to become an insulator is different, but the general behavior of metallic systems to become insulators under strong disorders is the same. If we want to represent a disordered material with a matrix Hamiltonian, the elements of the matrix have to be randomly selected. The general feature of a physical system is obtained with an ensemble of random matrices satisfying the physical constraints of the system.

Even though the elements of the matrices are random, at least three types of fundamental symmetries can exist in the ensembles of the random matrices for a given physical system. For example, if the system has time-reversal symmetry plus rotational invariance for the odd-half-integer spin case, the ensemble is real symmetric, or *orthogonal*.

If rotational invariance is not present in the odd-half-integer spin case, the ensemble is quaternion real, or *symplectic*. The general case without time-reversal symmetry is described by general Hermitian matrices, or a *unitary* ensemble.

The structure of the matrix is the other relevant factor that determines the detailed properties of a specific system. Traditionally, the focus of the general properties was on the orthogonal ensemble with real symmetric matrices and a Gaussian distribution for the matrix elements. Efforts made in the last 30 years are described in Mehta (1991). In this section, we will not be able to cover many aspects of random matrices and will only provide a very brief introduction. Interested readers should consult Mehta (1991), Brody *et al.* (1981), and Bohigas (1991).

We can easily generate a random matrix if the symmetry and the structure of the matrix are specified. The Gaussian orthogonal ensemble of $n \times n$ matrices is specified with a distribution $\mathcal{W}_n(\mathbf{H})$ that is invariant under an orthogonal transformation with

$$\mathcal{W}_n(\mathbf{H}')\, d\mathbf{H}' = \mathcal{W}_n(\mathbf{H})\, d\mathbf{H}, \tag{5.156}$$

where

$$\mathbf{H}' = \mathbf{O}^{\mathrm{T}}\mathbf{H}\mathbf{O}, \tag{5.157}$$

with $\mathbf{O}$ being an orthogonal matrix. The above condition also implies that $\mathcal{W}_n(\mathbf{H})$ is invariant under the orthogonal transformation because $d\mathbf{H}$ is invariant. This restricts the distribution to be

$$\mathcal{W}_n(\mathbf{H}) = e^{-\mathrm{Tr}\,\mathbf{H}^2/4\sigma^2}, \tag{5.158}$$

where the trace is given by

$$\mathrm{Tr}\,\mathbf{H}^2 = \sum_{i=1}^{n} H_{ii}^2 + \sum_{i\neq j}^{n} H_{ij}^2. \tag{5.159}$$

The average of the elements and the variance satisfy $\langle H_{ij}\rangle = 0$ and $\langle H_{ij}^2\rangle = (1 + \delta_{ij})\sigma^2$ for the Gaussian orthogonal ensemble. The variance for the off-diagonal elements is only half that of for the diagonal elements because $H_{ij} = H_{ji}$ in symmetric matrices.

We can generate a random-matrix ensemble numerically and diagonalize each matrix to obtain the distribution of the eigenvalues and eigenvectors. For example, the Gaussian orthogonal random matrix can be obtained with the method given below. The Gaussian random-number generator used is from the Java language.

```
// Method to generate a random matrix for the Gaussian
// orthogonal ensemble with sigma being the standard
// deviation of the off-diagonal elements.

import java.util.Random;
  public static double[][] rm(int n, double sigma) {
    double a[][] = new double[n][n];
    double sigmad = Math.sqrt(2)*sigma;
    Random r = new Random();

    for (int i=0; i<n; ++i)
      a[i][i] = sigmad*r.nextGaussian();

    for (int i=0; i<n; ++i) {
      for (int j=0; j<i; ++j) {
        a[i][j] = sigma*r.nextGaussian();
        a[j][i] = a[i][j];
      }
    }
    return a;
  }
```

This matrix can then be diagonalized by any method discussed earlier in this chapter. In order to obtain the statistical information, the matrix needs to be

generated and diagonalized many times before a smooth and correct distribution can be reached. Based on the distributions of the eigenvalues and the eigenvectors of the random-matrix ensemble, we can obtain the correlation functions of the eigenvalues and other important statistical information. Statistical methods and correlation functions are discussed in more detail in Chapters 8 and 10. We can also use the Gaussian random-number generator introduced in Chapter 2 instead the one from Java in the above method.

The distribution density of the eigenvalues in the Gaussian orthogonal ensemble can be obtained analytically, and it is given by a semicircle function

$$\rho(\lambda) = \begin{cases} \dfrac{1}{2\pi}\sqrt{4-\lambda^2} & \text{if } |\lambda| < 2, \\ 0 & \text{elsewhere,} \end{cases} \tag{5.160}$$

which was first derived by Wigner in the 1950s. Here $\rho(\lambda)$ is the normalized density at the eigenvalue λ measured in $\sigma\sqrt{n}$. The numerical simulations carried out so far seem to suggest that the semicircle distribution is true for any other ensemble as long as the elements satisfy $\langle H_{ij}\rangle = 0$ and $\langle H_{ij}^2\rangle = \sigma^2$ for $i < j$. We will leave this as an exercise for the reader.

There has been a lot of activity in the field of random-matrix theory and its applications. Here we mention just a couple of examples. Hackenbroich and Weidenmüller (1995) have shown that a general distribution of the form

$$\mathcal{W}_n(\mathbf{H}) = \frac{1}{\mathcal{Z}} e^{-n\,\mathrm{Tr}\,\mathbf{V}(\mathbf{H})}, \tag{5.161}$$

where $\mathcal{Z}$ is a normalization constant, would lead to the same correlation functions among the eigenvalues as those of a Gaussian orthogonal ensemble, in the limit of $n \to \infty$. This is true for any ensemble, orthogonal, symplectic, or unitary. Here $\mathbf{V}(\mathbf{H})$ is a function of $\mathbf{H}$ with the restriction that its eigenvalues are confined within a finite interval with a smooth distribution and that $V(\lambda)$ grows at least linearly as $\lambda \to \infty$. The Gaussian ensemble is a special case with $\mathbf{V}(\mathbf{H}) \propto \mathbf{H}^2$.

Akulin, Bréchignac, and Sarfati (1995) have used the random-matrix method in the study of the electronic, structural, and thermal properties of metallic clusters. They have introduced a random interaction V, where $\langle V\rangle = 0$ and $\langle V^2\rangle$ is a function characterized by the electron–electron, electron–ion, and/or electron–phonon interactions, and the shape fluctuations of the clusters. Using this theory, they have predicted deformation transformation in the cluster when the temperature is lowered. This effect is predicted only for open shell clusters and is quite similar to the Jahn–Teller effect, which creates a finite distortion in the lattice in ionic solids due to the electron–phonon interaction.

Exercises

5.1 Find the currents in the unbalanced Wheatstone bridge (Fig. 5.1). Assume that $v_0 = 1.5$ V, $r_1 = r_2 = 100\ \Omega$, $r_3 = 150\ \Omega$, $r_x = 120\ \Omega$, $r_\mathrm{a} = 1000\ \Omega$, and $r_\mathrm{s} = 10\ \Omega$.

5.2 A typical problem in physics is that physical quantities can be calculated for a series of finite systems, but ultimately, we would like to know the results for the infinite system. Hulthén (1938) studied the one-dimensional spin-$\frac{1}{2}$ Heisenberg model with the Hamiltonian

$$\mathcal{H} = \sum_{i=1}^{n-1} \mathbf{s}_i \cdot \mathbf{s}_{i+1},$$

where $\mathbf{s}_i$ is the spin at the ith site and n is the total number of sites in the system. From the eigenequation

$$\mathcal{H}\Psi_k = \mathcal{E}_k \Psi_k,$$

Hulthén obtained the ground-state energy per site, $\varepsilon_n = \mathcal{E}_0/n$, for a series of finite systems with $\varepsilon_2 = -2.0000$, $\varepsilon_4 = -1.5000$, $\varepsilon_6 = -1.4343$, $\varepsilon_8 = -1.4128$, and $\varepsilon_{10} = -1.4031$. Now assume that the ground-state energy per site is given by

$$\varepsilon_n = \varepsilon_\infty + \frac{c_1}{n} + \frac{c_2}{n^2} + \cdots + \frac{c_l}{n^l} + \cdots .$$

Truncate the above series at $l = 4$ and find ε_∞ by solving the linear equation set numerically.

5.3 Consider the least-squares approximation of a discrete function $f(x_i)$ for $i = 0, 1, \ldots, n$ with the polynomial

$$p_m(x) = \sum_{k=0}^{m} c_k x^k.$$

Write a subprogram that evaluates all the c_k from

$$\frac{\partial \chi^2}{\partial c_l} = 0,$$

for $l = 0, 1, \ldots, m$, where

$$\chi^2 = \sum_{i=0}^{n} [p_m(x_i) - f(x_i)]^2.$$

5.4 Develop a subprogram to achieve the LU decomposition of an $n \times n$ banded matrix with l subdiagonals and m superdiagonals with either the Crout or the Doolittle factorization. Simplify the subprogram for the cases of $l = m$ and a symmetric matrix.

5.5 Apply the secant method to obtain the stable geometric structure of clusters of ions $(Na^+)_n(Cl^-)_m$, where n and m are small positive integers. Use the empirical interaction potential given in Eq. (5.64) for the ions.

5.6 Write a subprogram to implement the BFGS optimization scheme. Test it by searching for the stable geometric structure of $(NaCl)_5$. Use the empirical interaction potential given in Eq. (5.64) for the ions.

5.7 Write a subprogram that utilizes the Householder scheme to tridiagonalize a real symmetric matrix.

5.8 Write a subprogram that uses the properties of determinant polynomials of a tridiagonal matrix and a root-search method to solve its first few eigenvalues.

5.9 Discretize the one-dimensional Schrödinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\phi(x)}{dx^2} + V(x)\phi(x) = \varepsilon\phi(x)$$

by applying the three-point formula to the second-order derivative above. Assuming that the potential $V(x)$ is

$$V(x) = \frac{\hbar^2}{2m}\alpha^2\lambda(\lambda-1)\left[\frac{1}{2} - \frac{1}{\cosh^2(\alpha x)}\right],$$

solve the corresponding eigenvalue problem by the inverse iteration method to obtain the eigenvalues and eigenvectors for the four lowest states. Compare the numerical result obtained here with the analytical result given in Eq. (4.98).

5.10 Find all the 15×15 elements of the Hamiltonian for H_3^+ and solve the corresponding eigenvalues and eigenvectors numerically. Compare the numerical results with the analytic results by reducing the matrix to block-diagonal form with the largest block being a 2×2 matrix.

5.11 It is of special interest in far infrared spectroscopy to know the vibrational spectrum of a molecule. Find all the the vibrational modes of Na_2Cl_2. Use the empirical interaction potential for the ions that is given in Eq. (5.64).

5.12 Implement the Faddeev–Leverrier method in a program to obtain the inverse, eigenvalues, and eigenvectors of a general real matrix.

5.13 Divide the polynomial $p_n(z) = \sum_{k=0}^{n} c_k z^k$ by $f_2(z) = z^2 + \beta z + \alpha$ twice and obtain the recursions for the coefficients in the quotients and remainders in each step. Find analytical expressions for $\partial r_i(\alpha, \beta)/\partial\alpha$ and $\partial r_i(\alpha, \beta)/\partial\beta$, where r_0 and r_1 are the coefficients of the remainder $r = r_1 z + r_0$ after the first division. Derive the Bairstow method that utilizes the Newton method to factor the polynomial $p_n(z)$ and to obtain a pair of its zeros. Implement the scheme in a program and test it with $p_6(z) = (z^2 + z + 1)(z^2 + 2z + 2)(z^2 + 3z + 3)$.

5.14 Combine the Routh–Hurwitz test and bisection method in a program to locate all the zeros of the polynomial $p_n(z) = \sum_{k=0} c_k z^k$. Test it with $p_6(z) = (z^2 + z + 1)(z^2 + 2z - 4)(z^2 + 3z + 3)$.

5.15 Derive the Hartree–Fock equation for the atomic systems given in Section 5.8. Show that the matrix form of the Hartree–Fock equation does represent the original equation if the electron density is spherically symmetric, and find the expression for $V_{x\sigma}^{(l)}(r_i, r_j)$ in the matrix equation.

5.16 Write a program to generate and diagonalize ensembles of real symmetric matrices with the following distributions:

$$\mathcal{W}_n(H_{ij}) = \begin{cases} \frac{1}{2} & \text{if } |H_{ij}| < 1, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathcal{W}_n(H_{ij}) = \frac{1}{2}[\delta(H_{ij} - 1) + \delta(H_{ij} + 1)],$$

$$\mathcal{W}_n(H_{ij}) = \frac{1}{\sqrt{2\pi}} e^{-H_{ij}^2/2},$$

for $i \leq j$. Compare the density of eigenvalues with that of the Wigner semicircle.

5.17 Find the optimized configurations of $N < 20$ charges confined on the surface of a unit sphere. Discuss whether the configurations found are the global minima of the electrostatic potential energies of the systems.

5.18 Find the optimized configurations of $N < 20$ particles, such as argon atoms, interacting with each other through the Lennard–Jones potential

$$V_{ij} = 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right],$$

where both ε and σ are model parameters. Find the N dependence of the number of local minima that are close to the global minimum of the total interaction energy of the system.