Chapter 6
Spectral analysis

Nature often behaves differently to how our intuition predicts it should. For example, when we observe a periodic motion, immediately we can figure out the period involved, but not the detailed structure of the data within each period. It was Fourier who first pointed out that an arbitrary periodic function $f(t)$, with a period T, can be decomposed into a summation of simple harmonic terms, which are periodic functions of frequencies that are multiples of the fundamental frequency $1/T$ of the function $f(t)$. Each coefficient in the summation is given by an integral of the product of the function and the complex conjugate of that harmonic term.

Assuming that we have a time-dependent function $f(t)$ with a period T, that is, $f(t+T) = f(t)$, the Fourier theorem can be cast as a summation

$$f(t) = \sum_{j=-\infty}^{\infty} g_j e^{-ij\omega t}, \tag{6.1}$$

which is commonly known as the Fourier series. Here $\omega = 2\pi/T$ is the fundamental angular frequency and g_j are the Fourier coefficients, which are given by

$$g_j = \frac{1}{T}\int_0^T f(t) e^{ij\omega t}\, dt. \tag{6.2}$$

The Fourier theorem can be derived from the properties of the exponential functions

$$\phi_j(t) = \frac{1}{\sqrt{T}} e^{-ij\omega t}, \tag{6.3}$$

which form an orthonormal basis set in the region of one period of $f(t)$: namely,

$$\int_{t_0}^{t_0+T} \phi_j^*(t)\phi_k(t)\, dt = \langle j|k\rangle = \delta_{jk}, \tag{6.4}$$

where t_0 is an arbitrary starting point, $\phi_j^*(t)$ is the complex conjugate of $\phi_j(t)$, and δ_{jk} is the Kronecker δ function.

The Fourier coefficients g_k of Eq. (6.2) are then obtained by multiplying Eq. (6.1) by $e^{-ik\omega t}$ and then integrating both sides of the equation over one period of $f(t)$. We can always take $t_0 = 0$ for convenience, because it is arbitrary.

6.1 Fourier analysis and orthogonal functions

We can generalize the Fourier theorem to a nonperiodic function defined in a region of $x \in [a, b]$ if we have a complete basis set of orthonormal functions $\phi_k(x)$ with

$$\int_a^b \phi_j^*(x)\phi_k(x)\,dx = \langle j|k\rangle = \delta_{jk}. \tag{6.5}$$

For any arbitrary function $f(x)$ defined in the region of $x \in [a, b]$, we can always write

$$f(x) = \sum_j g_j\,\phi_j(x) \tag{6.6}$$

if the function is square integrable, defined by

$$\int_a^b |f(x)|^2\,dx < \infty. \tag{6.7}$$

The summation in the series is over all the possible states in the complete set, and the coefficients g_j are given by

$$g_j = \int_a^b \phi_j^*(x)f(x)\,dx = \langle j|f\rangle. \tag{6.8}$$

The continuous Fourier transform is obtained if we restrict the series to the region of $t \in [-T/2, T/2]$ and then extend the period T to infinity. We need to redefine $j\omega \to \omega$ and $\sum_j \to (1/\sqrt{2\pi})\int d\omega$. Then the sum becomes an integral

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\omega)e^{-i\omega t}\,d\omega, \tag{6.9}$$

which is commonly known as the Fourier integral. The Fourier coefficient function is given by

$$g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)e^{i\omega t}\,dt. \tag{6.10}$$

Equations (6.9) and (6.10) define an integral transform and its inverse, which are commonly known as the Fourier transform and the inverse Fourier transform. We can show that the two equations above are consistent: that is, we can obtain the second equation by multiplying the first equation by $e^{i\omega t}$ and then integrating it. We need to use the Dirac δ function during this process. The Dirac δ function is defined by

$$\delta(x - x') = \begin{cases} \infty & \text{if } x = x', \\ 0 & \text{elsewhere,} \end{cases} \tag{6.11}$$

and

$$\int_{-\infty}^{\infty} \delta(x - x')\,dx = \lim_{\epsilon \to +0} \int_{x'-\epsilon}^{x'+\epsilon} \delta(x - x')\,dx = 1. \tag{6.12}$$

So the Dirac δ function $\delta(\omega)$ can also be interpreted as the Fourier transform of a constant function $f(t) = 1/\sqrt{2\pi}$, which we can easily show by carrying out the transform integration.

The Fourier transform can also be applied to other types of variables or to higher dimensions. For example, the Fourier transform of a function $f(\mathbf{r})$ in three dimensions is given by

$$f(\mathbf{r}) = \frac{1}{(2\pi)^{3/2}} \int g(\mathbf{q}) e^{i\mathbf{q}\cdot\mathbf{r}}\, d\mathbf{q}, \tag{6.13}$$

with the Fourier coefficient

$$g(\mathbf{q}) = \frac{1}{(2\pi)^{3/2}} \int f(\mathbf{q}) e^{-i\mathbf{q}\cdot\mathbf{r}}\, d\mathbf{r}. \tag{6.14}$$

Note that both of the above integrals are three-dimensional. The space defined by $\mathbf{q}$ is usually called the *momentum space*.

6.2 Discrete Fourier transform

The Fourier transform of a specific function is usually necessary in the analysis of experimental data, because it is often difficult to establish a clear physical picture just from the raw data taken from an experiment. Numerical procedures for the Fourier transform are inevitable in physics and other scientific fields. Usually we have to deal with a large number of data points, and the speed of the algorithm for the Fourier transform becomes a very important issue. In this section we will discuss a straightforward scheme for the one-dimensional case to illustrate some basic aspects of the DFT (discrete Fourier transform). In the next section, we will outline a fast algorithm for the discrete Fourier transform, commonly known as the FTT (fast Fourier transform).

As we pointed out earlier, the one-dimensional Fourier transform is defined by Eq. (6.9) and Eq. (6.10); as always, we need to convert the continuous variables into discrete ones before we develop a numerical procedure.

Let us consider $f(x)$ as a space-dependent physical quantity obtained from experimental measurements. If the measurements are conducted between $x = 0$ and $x = L$, $f(x)$ is nonzero only for $x \in [0, L]$. To simplify our problem, we can assume that the data are taken at evenly spaced points with each interval $h = L/(N-1)$, where N is the total number of data points. We assume that the data repeat periodically outside the region of $x \in [0, L]$, which is equivalent to imposing the periodic boundary condition on the finite system. The corresponding wavenumber in the momentum space is then discrete too, with an interval $\kappa = 2\pi/L$.

The discrete Fourier transform of such a data set can then be expressed in terms of a summation

$$f_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} g_j e^{i2\pi jk/N}, \tag{6.15}$$

with the Fourier coefficients given by

$$g_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} f_k e^{-i2\pi jk/N}, \tag{6.16}$$

where we have used our convention that $f_k = f(t = k\tau)$ and $g_j = g(\omega = j\nu)$. We can show that the above two summations are consistent, meaning that the inverse Fourier transform of the Fourier coefficients will give the exact values of $f(t)$ at $t = 0, \tau, \ldots, (N-1)\tau$. However, this inverse Fourier transform does not ensure smoothness in the recovered data function.

Note that the exponential functions in the series form a discrete basis set of orthogonal functions: that is,

$$\begin{aligned} \langle \phi_j | \phi_m \rangle &= \sum_{k=0}^{N-1} \frac{1}{\sqrt{N}} e^{-i2\pi kj/N} \frac{1}{\sqrt{N}} e^{i2\pi km/N} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} e^{i2\pi k(m-j)/N} = \delta_{jm}. \end{aligned} \tag{6.17}$$

Before we introduce the fast Fourier transform, let us examine how we can implement the discrete Fourier transform of Eq. (6.16) in a straightforward manner. We can separate the real (Re) and imaginary (Im) parts of the coefficients,

$$\mathrm{Re}\, g_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \left(\cos \frac{2\pi jk}{N} \mathrm{Re}\, f_k + \sin \frac{2\pi jk}{N} \mathrm{Im}\, f_k \right), \tag{6.18}$$

$$\mathrm{Im}\, g_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \left(\cos \frac{2\pi jk}{N} \mathrm{Im}\, f_k - \sin \frac{2\pi jk}{N} \mathrm{Re}\, f_k \right), \tag{6.19}$$

for convenience. Note that it is not necessary to separate them in practice even though most people would do so. Separating the real and imaginary parts is convenient because then we only need to deal with real numbers.

The following program is an implementation of Eqs. (6.18) and (6.19) with the real and imaginary parts of the coefficients separated.

```
// An example of performing the discrete Fourier transform
// with function f(x)=x(1-x) with x in [0,1].  The
// inverse transform is also performed for comparison.

import java.lang.*;
public class Fourier {
  static final int n = 128, m = 8;
  public static void main(String argv[]) {
  double x[] = new double[n];
  double fr[] = new double[n];
  double fi[] = new double[n];
  double gr[] = new double[n];
  double gi[] = new double[n];
  double h = 1.0/(n-1);
  double f0 = 1/Math.sqrt(n);

  // Assign the data and perform the transform
    for (int i=0; i<n; ++i) {
      x[i] = h*i;
```

```
      fr[i] = x[i]*(1-x[i]);
      fi[i] = 0;
    }
    dft(fr, fi, gr, gi);

  // Perform the inverse Fourier transform
    for (int i=0; i<n; ++i) {
      gr[i] =  f0*gr[i];
      gi[i] = -f0*gi[i];
      fr[i] = fi[i] = 0;
    }
    dft(gr, gi, fr, fi);
    for (int i=0; i<n; ++i) {
      fr[i] =  f0*fr[i];
      fi[i] = -f0*fi[i];
    }

  // Output the result in every m data steps
    for (int i=0; i<n; i+=m)
      System.out.println(x[i] + " " + fr[i] + " " fi[i]);
  }

// Method to perform the discrete Foruier transform.  Here
// fr[] and fi[] are the real and imaginary parts of the
// data with the correponding outputs gr[] and gi[].

  public static void dft(double fr[], double fi[],
    double gr[], double gi[]) {
    int n = fr.length;
    double x = 2*Math.PI/n;
    for (int i=0; i<n; ++i) {
      for (int j=0; j<n; ++j) {
        double q = x*j*i;
        gr[i] += fr[j]*Math.cos(q)+fi[j]*Math.sin(q);
        gi[i] += fi[j]*Math.cos(q)-fr[j]*Math.sin(q);
      }
    }
  }
}
```

The above program takes an amount of computing time that is proportional to N^2. We have used $f(x) = x(1 - x)$ as the data function to test the method in the region of $x \in [0, 1]$. This program demonstrates how we can perform the discrete Fourier transform and its inverse, that is, how to obtain g_j from f_k, and to obtain f_k from g_j, in exactly the same manner. Note that we do not have the factor $1/\sqrt{N}$ in the method, which is a common practice in designing a Fourier transform subprogram. It is obvious that the inverse transform is nearly identical to the transform except for a sign.

As we have pointed out, the inverse Fourier transform provides exactly the same values as the original data at the data points. The result of the inverse discrete Fourier transform from the above program and the original data are shown in Fig. 6.1. Some inaccuracy can still appear at these data points, because of the

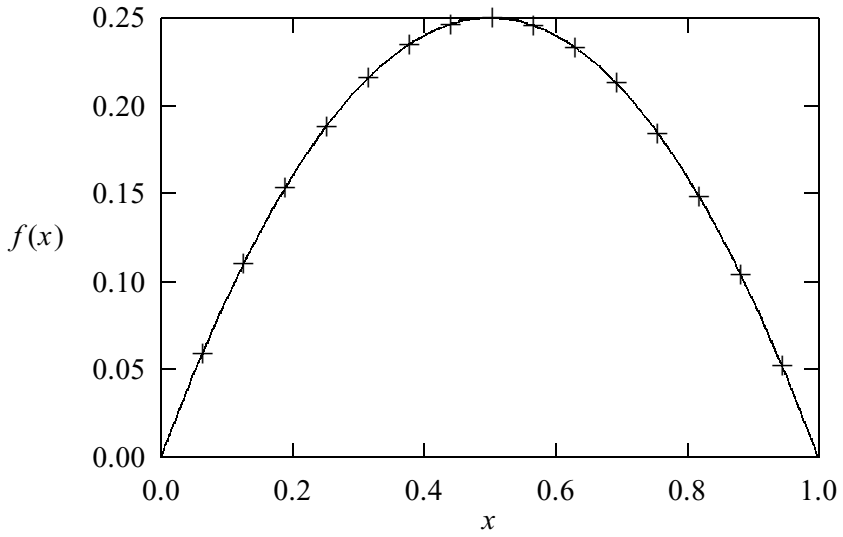

Fig. 6.1 The function $f(x) = x(1 - x)$ (line) and the corresponding values from the inverse transform of its Fourier coefficients (+) calculated with the example program.

rounding error incurred during the computing. A more important issue here is the accuracy of the recovered data function at places other than the data points. When the Fourier transform or its inverse is performed, we can have only a discrete set of data points. So an interpolation is inevitable if we want to know any value that is not at the data points. We can increase the number of points to reduce the errors in the interpolation of the data, but we also have to watch the growth of the rounding errors with the number of points used.

6.3 Fast Fourier transform

As we can clearly see, the straightforward discrete Fourier transform algorithm introduced in the preceding section is very inefficient, because the computing time needed is proportional to N^2. In order to solve this problem, many people have come up with the idea that is now known as the FFT (*fast Fourier transform*). The key element of the fast Fourier transform is to rearrange the terms in the series and to have the summation performed in a hierarchical manner. For example, we can perform a series of pairwise additions to accomplish the summation if the number of data points is of the power of 2: that is, $N = 2^M$, where M is an integer.

The idea behind the fast Fourier transform had been considered a long time ago, even before the first computer was built. As mentioned in Chapter 1, Gauss developed a version of the fast Fourier transform and published his work in neoclassical Latin (Gauss, 1886; Goldstine, 1977, pp. 249–53). However, no one really took notice of Gauss's idea or connected it to modern computation. The fast Fourier transform algorithm was formally discovered and put into practice by Cooley and Tukey (1965). Here we will give a brief outline of their idea.

The simplest fast Fourier transform algorithm is accomplished with the observation that we can separate the odd and even terms in the discrete Fourier transform as

$$\begin{aligned} g_j &= \sum_{k=0}^{N/2-1} f_{2k}\, e^{-i2\pi j(2k)/N} + \sum_{k=0}^{N/2-1} f_{2k+1}\, e^{-i2\pi j(2k+1)/N}, \\ &= x_j + y_j\, e^{-i2\pi j/N}, \end{aligned} \tag{6.20}$$

where

$$x_j = \sum_{k=0}^{N/2-1} f_{2k}\, e^{-i2\pi jk/(N/2)} \tag{6.21}$$

and

$$y_j = \sum_{k=0}^{N/2-1} f_{2k+1}\, e^{-i2\pi jk/(N/2)}. \tag{6.22}$$

Here we have ignored the factor $1/\sqrt{N}$, which can always be added back into the program that calls the fast Fourier transform subprogram. What we have done is to rewrite the Fourier transform with a summation of N terms as two summations, each of $N/2$ terms. This process can be repeated further and further until eventually we have only two terms in each summation if $N = 2^M$, where M is an integer. There is one more symmetry between g_k for $k < N/2$ and g_k for $k \geq N/2$: that is,

$$g_j = x_j + w^j y_j, \tag{6.23}$$

$$g_{j+N/2} = x_j - w^j y_j, \tag{6.24}$$

where $w = e^{-i2\pi/N}$ and $j = 0, 1, \ldots, N/2 - 1$. This is quite important in practice, because now we need to perform the transform only up to $j = N/2 - 1$, and the Fourier coefficients with higher j are obtained with the above equation at the same time.

There are two important ingredients in the fast Fourier transform algorithm. After the summation is decomposed M times, we need to add individual data points in pairs. However, due to the sorting of the odd and even terms in every level of decomposition, the points in each pair at the first level of additions can be very far apart in the original data string. However, Cooley and Tukey (1965) found that if we record the data string index with a binary number, a *bit-reversed order* will put each pair of data points next to each other for the summations at the first level. Let us take a set of 16 data points $f_0, f_1, \ldots, f_{15}$ as an example. If we record them with a binary index, we have 0000, 0001, 0010, 0011, ..., 1111, for all the data points. Bit-reversed order is achieved if we reverse the order of each binary string. For example, the bit-reversed order of 1000 is 0001. So the order of the data after the bit reversal is $f_0, f_8, f_4, f_{12}, \ldots, f_3, f_{11}, f_7, f_{15}$. Then the first level of additions is performed between f_0 and f_8, f_4 and $f_{12}, \ldots, f_3$ and f_{11}, and f_7 and f_{15}. Equations (6.23) and (6.24) can be applied repeatedly to

sum the pairs 2^{l-1} spaces apart in the bit-reversed data stream. Here l indicates the level of additions; for example, the first set of additions corresponds to $l = 1$. At each level of additions, two data points are created from each pair. Note that w^j is associated with the second term in the equation.

With the fast Fourier transform algorithm, the computing time needed for a large set of data points is tremendously reduced. We can see this by examining the calculation steps needed in the transform. Assume that $N = 2^M$, so that after bit reversal, we need to perform M levels of additions and $N/2^l$ additions at the lth level. A careful analysis shows that the total computing time in the fast Fourier transform algorithm is proportional to $N \log_2 N$ instead of to N^2, as is the case for the straightforward discrete Fourier transform. Similar algorithms can be devised for N of the power of 4, 8, and so forth.

Many versions and variations of the fast Fourier transform programs are available in Fortran (Burrus and Parks, 1985) and in other programming languages. One of the earliest Fortran programs of the fast Fourier transform was written by Cooley, Lewis, and Welch (1969). Now many computers come with a fast Fourier transform library, which is usually written in machine language and tailored specifically for the architecture of the system.

Most routines for the fast Fourier transform are written with complex variables. However, it is easier to deal with just real variables. We can always separate the real and imaginary parts in Eqs. (6.20)–(6.24), as discussed earlier. The following method is an example of this. Note also that Java does not have a complex data type at the moment and we need to create a class for complex variables and their operations if we want implement the algorithm without separating the real and imaginary parts of the data. For other languages, it is more concise for programming to have both input and result in a complex form. If the data are real, we can simply remove the lines related to the imaginary part of the input data.

```
// Method to perform the fast Foruier transform.  Here
// fr[] and fi[] are the real and imaginary parts of
// both the input and output data.

  public static void fft(double fr[], double fi[],
    int m) {
    int n = fr.length;
    int nh = n/2;
    int np = (int) Math.pow(2, m);

  // Stop the program if the indices do not match
    if (np != n) {
      System.out.println("Index mismtch detected");
      System.exit(1);
    }

  // Rearrange the data to the bit-reversed order
    int k = 1;
    for (int i=0; i<n-1; ++i) {
      if (i < k-1) {
        double f1 = fr[k-1];
```

```
        double f2 = fi[k-1];
        fr[k-1] = fr[i];
        fi[k-1] = fi[i];
        fr[i] = f1;
        fi[i] = f2;
      }
      int j = nh;
      while (j < k) {
        k -= j;
        j /= 2;
      }
      k += j;
    }
  // Sum up the reordered data at all levels
    k = 1;
    for (int i=0; i<m; ++i) {
      double w = 0;
      int j = k;
      k = 2*j;
      for (int p=0; p<j; ++p) {
        double u =  Math.cos(w);
        double v = -Math.sin(w);
        w += Math.PI/j;
        for (int q=p; q<n; q+=k) {
          int r = q+j;
          double f1 = fr[r]*u-fi[r]*v;
          double f2 = fr[r]*v+fi[r]*u;
          fr[r] = fr[q]-f1;
          fr[q] += f1;
          fi[r] = fi[q]-f2;
          fi[q] += f2;
        }
      }
    }
  }
```

As we can see from the above method, rearranging the data points into bit-reversed order is nontrivial. However, it can still be understood if we examine the part of the program iteration by iteration with a small N. To be convinced that the above subroutine is a correct implementation of the $N = 2^M$ case, the reader can take $N = 16$ as an example and then work out the terms by hand.

A very important issue here is how much time can use of the fast Fourier transform save. On a scalar machine, it is quite significant. However, on a vector machine, the saving is somewhat restricted. A vector processor can perform the inner loop in the discrete Fourier transform in parallel within each clock cycle, and this makes the computing time for the straightforward discrete Fourier transform proportional to N^α, with α somewhere between 1 and 2. In real problems, α may be a little bit higher than 2 for a scalar machine. However, the advantage of vectorization in the fast Fourier transform is not as significant as in the straightforward discrete Fourier transform. So in general, we may need to examine the problem under study, and the fast Fourier transform is certainly an important tool, because the available computing resources are always limited.

6.4 Power spectrum of a driven pendulum

In Chapter 4 we discussed the fact that a driven pendulum with damping can exhibit either periodic or chaotic behavior. One way to analyze the dynamics of a nonlinear system is to study its power spectrum.

The power spectrum of a dynamical variable is defined as the square of the modulus of the Fourier coefficient function,

$$S(\omega) = |g(\omega)|^2, \tag{6.25}$$

where $g(\omega)$ is given by

$$g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} y(t) e^{i\omega t}\, dt, \tag{6.26}$$

with $y(t)$ being the time-dependent dynamic variable. We discussed in the preceding section how to achieve the fast Fourier transform numerically. With the availability of such a scheme, the evaluation of the power spectrum of a time-dependent variable becomes straightforward.

The driven pendulum with damping is described by

$$\frac{dy_1}{dt} = y_2, \tag{6.27}$$

$$\frac{dy_2}{dt} = -q\, y_2 - \sin y_1 + b \cos \omega_0 t, \tag{6.28}$$

where $y_1(t) = \theta(t)$ is the angle between the pendulum and the vertical, $y_2(t) = d\theta(t)/dt$ is its corresponding angular velocity, q is a measure of the damping, and b and ω_0 are the amplitude and angular frequency of the external driving force. We have already shown in Chapter 4 how to solve this problem numerically with the fourth-order Runge–Kutta algorithm. The power spectra of the time-dependent angle and the time-dependent angular velocity can be obtained easily by performing a discrete Fourier transform with the fast Fourier transform algorithm. We show the numerical results in Fig. 6.2(a) and (b), which correspond to the power spectra of the time-dependent angle $\theta(t)$ in periodic and chaotic regions, respectively.

We have taken 65 536 data points with 192 points in one period of the driving force. The initial condition, $y_1(0) = 0$ and $y_2(0) = 2$, is used in generating the data. In Fig. 6.2, we have shown the first 2048 points of the power spectra. As we can see from the figure, the power spectrum for periodic motion has separated sharp peaks with δ-function-like distributions. The broadening of the major peak at the angular frequency of the driving force, ω_0, is due to rounding errors. We can reduce the broadening by increasing the number of points used. The power spectrum for the chaotic case is quite irregular, but still has noticeable peaks at the same positions as the periodic case. This is because the contribution at the angular frequency of the driving force is still high, even though the system is chaotic. The peaks at multiples of ω_0 are due to relatively large contributions at the higher frequencies with $\omega = n\omega_0$, where n is an integer, in the Fourier coefficient. If we have more data points, we can also study the fractal structure of the power

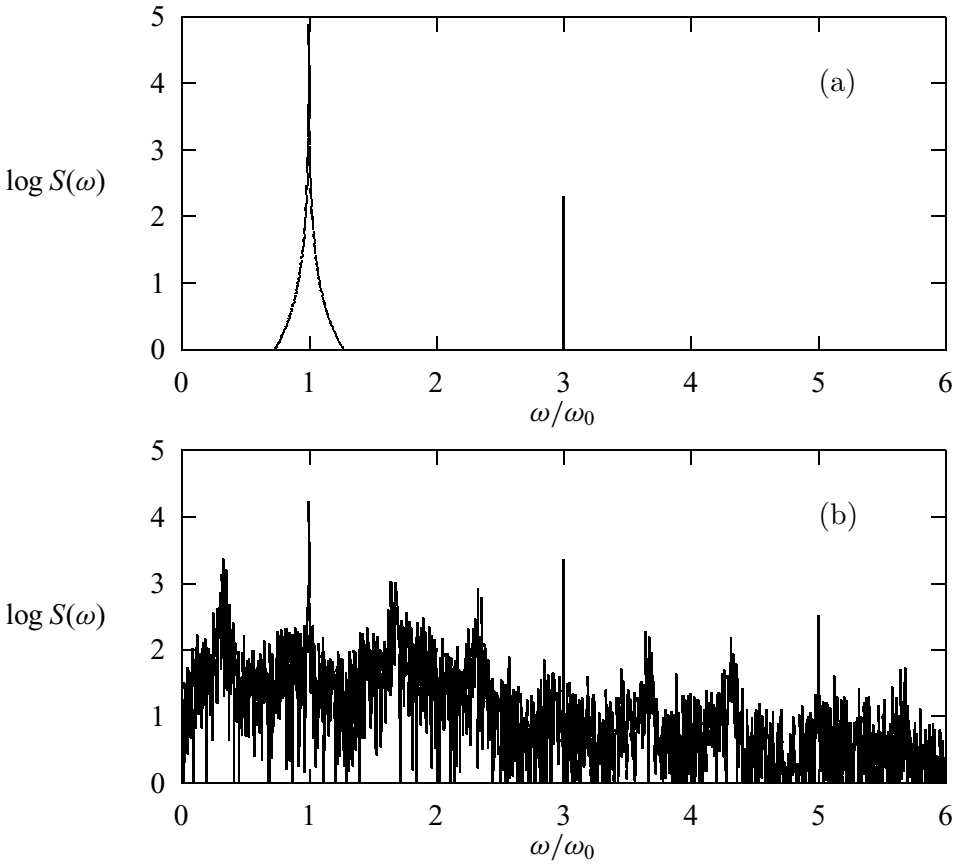

Fig. 6.2 Power spectra of the time-dependent angle of a driven pendulum with damping, with $\omega_0 = 2/3$, $q = 0.5$, for (a) periodic behavior at $b = 0.9$ and (b) chaotic behavior at $b = 1.15$.

spectrum. This could be achieved by examining the data at different scales of the frequency; we would observe similar patterns emerging from different scales with a unique fractal dimensionality.

6.5 Fourier transform in higher dimensions

The Fourier transform can be obtained in a very straightforward manner in higher dimensions if we realize that we can transform each coordinate as if it were a one-dimensional problem, with all other coordinate indices held constant.

Let us take the two-dimensional case as an example. Assume that the data are from a rectangular domain with N_1 mesh points in one direction and N_2 in the other. So the total number of points is $N = N_1 N_2$. The discrete Fourier transform is then given by

$$\begin{aligned} g_{jk} &= \frac{1}{\sqrt{N}} \sum_{l=0}^{N_1-1} \sum_{m=0}^{N_2-1} f_{lm} e^{-i2\pi(jl/N_1 + km/N_2)} \\ &= \frac{1}{\sqrt{N_1}} \sum_{l=0}^{N_1-1} e^{-i2\pi jl/N_1} \frac{1}{\sqrt{N_2}} \sum_{m=0}^{N_2-1} f_{lm} e^{-i2\pi km/N_2}. \end{aligned} \tag{6.29}$$

Thus, we can obtain the transform first for all the terms under index m with a fixed l and then for all the terms under index l with a fixed k. The procedure can be seen easily from the method given below.

```
// Method to carry out the fast Fourier transform for a
// 2-dimenisonal array.  Here fr[][] and fi[][] are real
// and imaginary parts in both the input and output.

  public static void fft2d(double fr[][],
    double fi[][], int m1, int m2) {
    int n1 = fr.length;
    int n2 = fr[0].length;
    double hr[] = new double[n2];
    double hi[] = new double[n2];
    double pr[] = new double[n1];
    double pi[] = new double[n1];

 // Perform fft on the 2nd index
    for (int i=0; i<n1; ++i) {
      for (int j=0; j<n2; ++j) {
        hr[j] = fr[i][j];
        hi[j] = fi[i][j];
      }
      fft(hr, hi, m2);
      for (int j=0; j<n2; ++j) {
        fr[i][j] = hr[j];
        fi[i][j] = hi[j];
      }
    }

 // Perform fft on the 1st index
    for (int j=0; j<n2; ++j) {
      for (int i=0; i<n1; ++i) {
        pr[i] = fr[i][j];
        pi[i] = fi[i][j];
      }
      fft(pr, pi, m1);
      for (int i=0; i<n1; ++i) {
        fr[i][j] = pr[i];
        fi[i][j] = pi[i];
      }
    }
  }

  public static void fft(double fr[], double fi[],
    int m) {...}
```

We have used the one-dimensional fast Fourier transform twice, once for each of the two indices. This procedure works well if the boundary of the data is rectangular.

6.6 Wavelet analysis

Wavelet analysis was first introduced by Haar (1910) but not recognized as a powerful mathematical tool until the 1980s. It was Morlet *et al.* (1982a; 1982b)

who first used the wavelet approach in seismic data analysis. The wavelet transform contains spectral information at different scales and different locations of the data stream, for example, the intensity of a signal around a specific frequency and a specific time. This is in contrast to Fourier analysis, in which a specific transform coefficient contains information about a specific scale or frequency from the entire data space without referring to its relevance to the location in the original data stream. The wavelet method is extremely powerful in the analysis of short time signals, transient data, or complex patterns. The development and applications of wavelet analysis since the 1980s have shown that many more applications will emerge in the future. Here we give just a brief introduction to the subject and point out its potential applications in computational physics. More details on the method and some applications can be found in several monographs (Daubechies, 1992; Chui, 1992; Meyer, 1993; Young, 1993; Holschneider, 1999; Percival and Walden, 2000) and collections (Combes, Grossmann, and Tchanmitchian, 1990; Chui, Montefusco, and Puccio, 1994; Foufoula-Georgiou and Kumar, 1994; Silverman and Vassilicos, 1999; van den Berg, 1999).

Windowed Fourier transform

It is desirable in many applications that the local structure in a set of data can be amplified and analyzed. This is because we may not be able to obtain the data throughout the entire space or on a specific scale of particular interest. Sometimes we also want to filter out the noise around the boundaries of the data. A *windowed Fourier transform* can be formulated to select the information from the data at a specific location. We can define

$$g(\omega, \tau) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) w(t - \tau) e^{i\omega t}\, dt \tag{6.30}$$

as the windowed Fourier transform of the function $f(t)$ under the window function $w(t - \tau)$. The window function $w(t - \tau)$ is used here to extract information about $f(t)$ in the neighborhood of $t = \tau$. The window function $w(t)$ is commonly chosen to be a real, even function, and satisfies

$$\int_{-\infty}^{\infty} w^2(t)\, dt = 1. \tag{6.31}$$

Typical window functions include the triangular window function

$$w(t) = \begin{cases} \dfrac{1}{\sqrt{\mathcal{N}}} \left(1 - \dfrac{|t|}{\sigma}\right) & \text{if } |t| < \sigma, \\ 0 & \text{elsewhere,} \end{cases} \tag{6.32}$$

and the Gaussian window function

$$w(t) = \frac{1}{\sqrt{\mathcal{N}}} e^{-t^2/2\sigma^2}, \tag{6.33}$$

where $\mathcal{N}$ is the normalization constant and σ is a measure of the window width. As soon as σ is selected, $\mathcal{N}$ can be evaluated with Eq. (6.31). From the definition, we can also recover the data from their Fourier coefficients through the inverse transform

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\omega, \tau) w(\tau - t) e^{-i\omega t} \, d\omega \, d\tau \tag{6.34}$$

as in the conventional Fourier transform. The inner product, that is, the integral over the square of the data amplitude, is equal to the integral over the square of its transform coefficient amplitude: that is,

$$\int_{-\infty}^{\infty} |f(t)|^2 dt = \int_{-\infty}^{\infty} |g(\omega, \tau)|^2 \, d\omega \, d\tau. \tag{6.35}$$

The advantage of the windowed Fourier transform is that it tunes the data with the window function so that we can obtain the local structure of the data or suppress unwanted effects in the data string. However, the transform treats the whole data space uniformly and would not be able to distinguish detailed structures of the data at different scales.

Continuous wavelet transform

The windowed Fourier transform can provide information at a given location in time, but it fails to provide the data with a specific scale at the selected location. We can obtain information about a set of data locally and also at different scales through wavelet analysis.

The continuous wavelet transform of a function $f(t)$ is defined through the integral

$$g(\lambda, \tau) = \int_{-\infty}^{\infty} f(t) u^*_{\lambda\tau}(t) \, dt, \tag{6.36}$$

where $u^*_{\lambda\tau}(t)$ is the complex conjugate of the *wavelet*

$$u_{\lambda\tau}(t) = \frac{1}{\sqrt{|\lambda|}} u\left(\frac{t - \tau}{\lambda}\right), \tag{6.37}$$

with $\lambda \neq 0$ being the *dilate* and τ being the *translate* of the wavelet transform. The parameters λ and τ are usually chosen to be real, and they select, respectively, the scale and the location of the data stream during the transformation. The function $u(t)$ is the generator of all the wavelets $u_{\lambda\tau}(t)$ and is called the *mother wavelet* or just the *wavelet*. Note that $u_{10}(t) = u(t)$ as expected for the case without any dilation or translation in picking up the data.

There are some constraints on the selection of a meaningful wavelet $u(t)$. For example, in order to have the inverse transform defined, we must have

$$\mathcal{Z} = \int_{-\infty}^{\infty} \frac{1}{|\omega|} |z(\omega)|^2 \, d\omega < \infty, \tag{6.38}$$

where $z(\omega)$ is the Fourier transform of $u(t)$. The above constraint is called the *admissibility condition* of the wavelet (Daubechies, 1992), which is equivalent to

$$z(0) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u(t)\, dt = 0, \tag{6.39}$$

if $u(t)$ is square integrable and decays as $t \to \pm\infty$. This is why $u(t)$ is called a wavelet, meaning a small wave, in comparison with a typical plane wave, which satisfies the above condition but is not square integrable. The wavelet $u(t)$ is usually normalized with

$$\langle u|u\rangle = \int_{-\infty}^{\infty} |u(t)|^2 dt = 1 \tag{6.40}$$

for convenience.

Let us examine a simple wavelet

$$u(t) = \Theta(t) - 2\Theta\left(t - \frac{1}{2}\right) + \Theta(t-1), \tag{6.41}$$

which is called the Haar wavelet, with the step function

$$\Theta(t) = \begin{cases} 1 & \text{if } t > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6.42}$$

The wavelet transform coefficients $g(\lambda, \tau)$ are obtained through Eq. (6.36) with the wavelet given by

$$u_{\lambda\tau}(t) = \frac{1}{\sqrt{|\lambda|}} \left[\Theta(t-\tau) - 2\Theta\left(t - \tau - \frac{\lambda}{2}\right) + \Theta(t-\tau-\lambda)\right]. \tag{6.43}$$

The difficulty here is to have an analytic form of $g(\lambda, \tau)$, even for a simple form of $f(t)$. In general the integration must be carried out numerically. Let us consider a simple function $f(t) = t(1-t)$ for $t \in [0, 1]$, zero otherwise. The following program shows how to obtain the continuous wavelet transform of the function numerically.

```
// An example of performing the continuous wavelet
// transform with function f(t)=t(1-t) for 0<t<1.

import java.lang.*;
public class Wavelet{
  static final int nt = 21, nv = 11;
  public static void main(String argv[]) {
  double t[] = new double[nt];
  double f[] = new double[nt];
  double fi[] = new double[nt];
  double v[] = new double[nv];
  double g[][] = new double[nv][nv];
  double dt = 1.0/(nt-1), dv = 1.0/(nv-1);
  double rescale = 100;

  // Assign the data, dilate, and translate
    for (int i=0; i<nt; ++i) {
      t[i] = dt*i;
```

```
      f[i] = t[i]*(1-t[i]);
    }
    for (int i=0; i<nv; ++i) v[i] = dv*(i+1);

  // Perform the transform
    for (int i=0; i<nv; ++i){
      double sa = Math.sqrt(Math.abs(v[i]));
      for (int j=0; j<nv; ++j){
        for (int k=0; k<nt; ++k){
          fi[k]  = f[k]*(theta(t[k]-v[j])
                 - 2*theta(t[k]-v[j]-v[i]/2)
                 + theta(t[k]-v[j]-v[i]))/sa;
        }
        g[i][j] = simpson(fi,dt);
      }
    }

  // Output the coefficients obtained
    for (int i=0; i<nv; ++i){
      for (int j=0; j<nv; ++j){
        double g2 = rescale*g[i][j]*g[i][j];
        System.out.println(v[i] + " " + v[j] + " " + g2);
      }
    }
  }

// Method to create a step funciton.

  public static double theta(double x) {
    if (x>0) return 1.0;
    else return 0.0;
  }

  public static double simpson(double y[], double h) {...}
}
```

In Fig. 6.3 we show the surface and contour plots of $|g(\lambda, \tau)|^2$, generated with the above program. The contour plot is called the *scalogram* of $f(t)$, and can be interpreted geometrically in analyzing the information contained in the data (Grossmann, Kronland-Martinet, and Morlet, 1989).

From the definition of the continuous wavelet transform and the constraints on the selection of the wavelet $u(t)$, we can show that the data function can be recovered from the inverse wavelet transform

$$f(t) = \frac{1}{\mathcal{Z}} \int_{-\infty}^{\infty} \frac{1}{\lambda^2} g(\lambda, \tau) u_{\lambda\tau}(t)\, d\lambda\, d\tau. \tag{6.44}$$

We can also show that the wavelet transform satisfies an identity similar to that in the Fourier transform or the windowed Fourier transform with

$$\int_{-\infty}^{\infty} |f(t)|^2 dt = \frac{1}{\mathcal{Z}} \int_{-\infty}^{\infty} \frac{1}{\lambda^2} |g(\lambda, \tau)|^2 d\lambda\, d\tau. \tag{6.45}$$

Analytically, the continuous wavelet transform is easier to deal with than the discrete wavelet transform. However, most data obtained are discrete in nature.

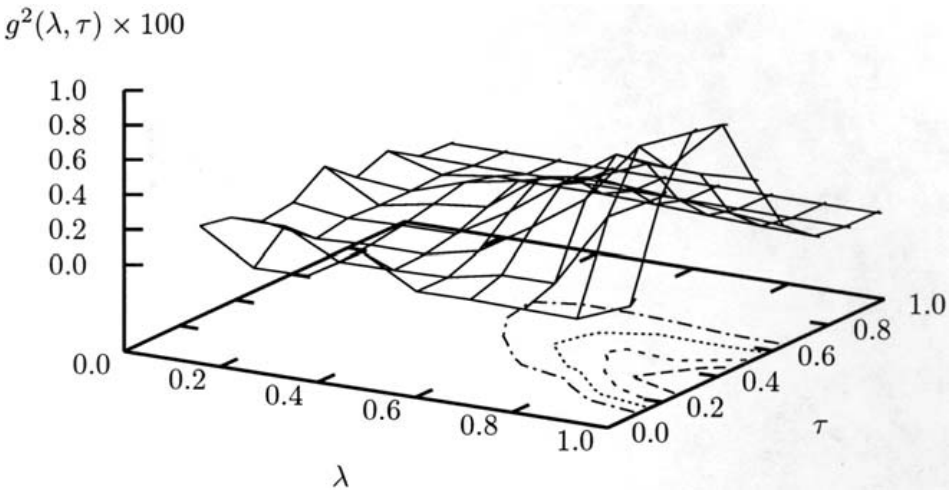

Fig. 6.3 Surface and contour plots of the scalogram of the function $f(t) = t(1 - t)$ for $t \in [0, 1]$ and zero otherwise.

More importantly, it is much easier to implement the data analysis numerically if the transform is defined with discrete variables.

6.7 Discrete wavelet transform

From the continuous wavelet transform, we can obtain detailed information on the data at different locations and scales. The drawback is that the information received is redundant because we effectively decompose a one-dimensional data stream into a two-dimensional data stream. To remove this redundancy, we can take the wavelet at certain selected scales (dyadic scales at $\lambda_j = 2^j$ for integers j) and certain locations (k points apart on the scale of λ_j). The transform can then be achieved level by level with the multiresolution analysis of Mallat (1989) under a pyramid scheme.

In the spirit of carrying out the Fourier analysis in terms of the Fourier series, we expand the time sequence as

$$f(t) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} c_{jk} u_{jk}(t), \tag{6.46}$$

where the basis set

$$u_{jk}(t) = 2^{j/2} u(2^j t - k) \tag{6.47}$$

is generated from dilating and translating the wavelet $u(t)$. The transform is obtained from

$$c_{jk} = \langle u_{jk} | f \rangle = \int_{-\infty}^{\infty} f(t) u_{jk}(t)\, dt, \tag{6.48}$$

following the general argument that $u_{jk}(t)$ form a complete, orthonormal basis set. The expansion in Eq. (6.46) is also known as *synthesis* because we can reconstruct

the time sequence $f(t)$ if all the coefficients c_{jk} are given. Consequently, the transformation of Eq. (6.48) is called *analysis*. To simplify our problem, we have taken the basis set to be orthonormal even though completeness is the only necessary condition.

We have also made the assumption that the basis set is real and orthonormal for simplicity, even though completeness is the only necessary condition needed. An example of an orthonormal wavelet is the Haar wavelet discussed in the preceding section. We can follow the approach that we have used in the preceding section for the continuous wavelet transform to obtain all the integrals in the transform coefficients. However, the hierarchical structure of the discrete wavelet transform allows us to use a much more efficient method to obtain the transform without worrying about all the integrals involved.

Multiresolution analysis

We can first define a set of linear vector spaces W_j, with each span over the function space covered by u_{jk} for $-\infty < k < \infty$, allowing us to decompose $f(t)$ into components residing in individual W_j with

$$f(t) = \sum_{j=-\infty}^{\infty} d_j(t), \tag{6.49}$$

where $d_j(t)$ is called the *detail* of $f(t)$ in W_j and is given by

$$d_j(t) = \sum_{k=-\infty}^{\infty} c_{jk} u_{jk}(t). \tag{6.50}$$

Then we can introduce another set of linear vector spaces V_j, with each the addition of its nested subspace $V_{j-1} \subset V_j$ and W_{j-1}. We can visualize this by examining three-dimensional Euclidean space. If V_j were taken as the entire Euclidean space and V_{j-1} the xy plane, W_{j-1} would be the z axis. Note specifically that $u(t-k)$ form the complete basis set for the space W_0. A symbolic sketch of the spaces W_j and V_j is given in Fig. 6.4. Note that there is no overlap between any two different W_j spaces, whereas V_j are nested within V_k for $j < k$.

Then the projection of $f(t)$ into the space V_j can be written as

$$a_j(t) = \sum_{k=-\infty}^{j-1} \sum_{l=-\infty}^{\infty} c_{kl} u_{kl}(t), \tag{6.51}$$

with the limits $a_\infty(t) = f(t)$ and $a_{-\infty}(t) = 0$. The function $a_j(t)$ is called the *approximation* of $f(t)$ in V_j. The spaces W_j and V_j, and the approximations $a_j(t)$ and details $d_j(t)$ are simply related by

$$V_{j+1} = V_j \oplus W_j, \tag{6.52}$$

$$a_{j+1}(t) = a_j(t) + d_j(t). \tag{6.53}$$

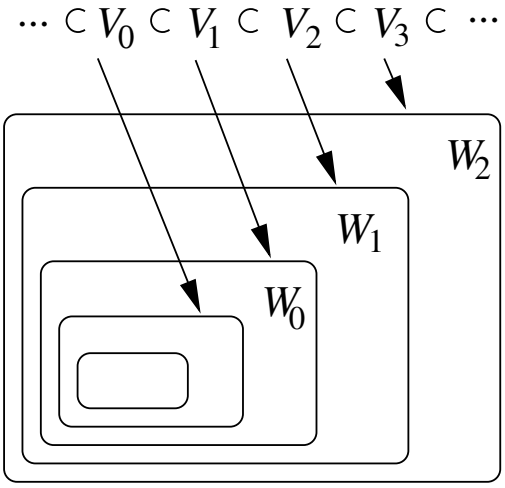

Fig. 6.4 A symbolic representation of the spaces W_j, which are orthogonal with each other, and the spaces V_j, which are nested within V_k for $j < k$.

The above relations define the multiresolution of the space and function. The nested spaces V_j possess certain unique properties. For example, $V_{-\infty}$ contains only one element, the zero vector, which is shared among all the spaces V_j. This means that the projection of any function $f(t)$ into $V_{-\infty}$ is zero. Furthermore, if $f(t)$ is a function in space V_j, $f(2t)$ is automatically a function in space V_{j+1}, and vice versa. This is the consequence of $u_{j+1k}(t) = \sqrt{2}u_{jk}(2t)$. The space V_∞ becomes equivalent to the space $L^2(R)$, which holds all the square-integrable functions $f(t)$.

Let us consider a simple example of decomposing a function $f(t)$ given in a certain space V_j a few times. The space hierarchy is given as

$$\begin{aligned} V_j &= V_{j-1} \oplus W_{j-1} \\ &= V_{j-2} \oplus W_{j-2} \oplus W_{j-1} \\ &= V_{j-3} \oplus W_{j-3} \oplus W_{j-2} \oplus W_{j-1}, \end{aligned} \tag{6.54}$$

which allows us to expand the function as

$$\begin{aligned} f(t) &= A_1(t) + D_1(t) \\ &= A_2(t) + D_2(t) + D_1(t) \\ &= A_3(t) + D_3(t) + D_2(t) + D_1(t), \end{aligned} \tag{6.55}$$

where $A_1(t)$ is the first-level approximation or the projection of $f(t)$ into V_{j-1}, $A_2(t)$ is the second-level approximation or the projection into V_{j-2}, $D_1(t)$ is the first-level detail in W_{j-1}, $D_2(t)$ is the second-level detail in W_{j-2}, and so forth. This is just a special case of $f(t) = a_j(t)$ with $A_k(t) = a_{j-k}(t)$ and $D_k(t) = d_{j-k}(t)$. The essence of the multiresolution analysis is therefore to decompose a data sequence into difference levels of approximations and details.

Scaling function

The multiresolution analysis of $f(t)$ can be realized in a concise form if there exist a set of functions $v_{jk}(t)$ that form a complete, orthogonal basis set in space V_j with

$$\langle v_{jk}(t)|v_{jl}(t)\rangle = \int_{-\infty}^{\infty} v_{jk}(t)v_{jl}(t)\,dt = \delta_{kl}. \tag{6.56}$$

Here $v_{jk}(t)$ are called scaling functions that satisfy

$$v_{jk}(t) = 2^{j/2}v(2^j t - k), \tag{6.57}$$

where the scaling function $v(t)$ is sometimes also referred to as the *father scaling function* or *father wavelet*. Note that $v_{jk}(t)$ with different j cannot be made orthogonal to each other because the spaces V_j are nested rather than orthogonal, unlike the spaces W_j. More importantly, V_0 and W_0 are both the subspaces of V_1 because $V_0 \oplus W_0 = V_1$. We can therefore expand a basis function in V_0, $v(t) = v_{00}(t)$, and a basis function in W_0, $u(t) = u_{00}(t)$, in terms of the basis set in V_1 as

$$v(t) = \sum_{k=-\infty}^{\infty} h(k)v_{1k}(t) = \sum_{k=-\infty}^{\infty} h(k)\sqrt{2}v(2t - k), \tag{6.58}$$

$$u(t) = \sum_{k=-\infty}^{\infty} g(k)v_{1k}(t) = \sum_{k=-\infty}^{\infty} g(k)\sqrt{2}v(2t - k), \tag{6.59}$$

where $h(k)$ and $g(k)$ are referred to as filters, which we will discuss in more detail below. The two-scale relations given in Eqs. (6.58) and (6.59) are instrumental to the development of an efficient algorithm in achieving the discrete wavelet transform.

There are certain properties that we can derive from the orthogonal conditions of the scaling functions and wavelets. For example, from the integration over time on the two-scale relation for the scaling function, we obtain

$$\sum_{k=-\infty}^{\infty} h(k) = \sqrt{2}. \tag{6.60}$$

Furthermore, because $\langle v(t)|v(t - l)\rangle = \delta_{0l}$, we also have

$$\sum_{k=-\infty}^{\infty} h(k)h(k + 2l) = \delta_{0l}, \tag{6.61}$$

after applying the two-scale equation for the scaling function. Furthermore, we can show, from the Fourier transform of the scaling functions, that

$$\int_{-\infty}^{\infty} v(t)\,dt = 1. \tag{6.62}$$

Using the admissibility condition

$$\int_{-\infty}^{\infty} u(t)\,dt = 0 \tag{6.63}$$

and $\langle v(t)|u(t-l)\rangle = \delta_{0l}$, a consequence of W_0 being orthogonal to V_0, we can also obtain

$$g(k) = (-1)^k h(r-k), \tag{6.64}$$

where r is an arbitrary, odd integer. However, if the number of nonzero $h(k)$ is finite and equal to n, we must have $r = n-1$.

In order to have all $h(k)$ for $k = 0, 1, \ldots, n-1$ determined, we need a total of n independent equations. Equations (6.60) and (6.61) provide a total of $n/2+1$ equations. So we are still free to impose more conditions on $h(k)$. If we want to recover polynomial data up to the $(n/2-1)$th order, the moments under the filters must satisfy (Strang, 1989)

$$\sum_{k=0}^{n-1} (-1)^k k^l h(k) = 0, \tag{6.65}$$

for $l = 0, 1, \ldots, n/2-1$, which supplies another $n/2$ equations. Note that the case of $l = 0$ in either Eq. (6.61) or Eq. (6.65) can be derived from Eq. (6.60) and other given relations. So Eqs. (6.60), (6.61), and (6.65) together provide n independent equations for n coefficients $h(k)$, and therefore they can be uniquely determined for a given n. For small n we can solve $h(k)$ easily with this procedure.

Now let us use a simple example to illustrate the points made above. Consider here the case of the Haar scaling function and wavelet for $n = 2$. The Haar scaling function is a box between 0 and 1, namely, $v(t) = \Theta(t) - \Theta(t-1)$, where $\Theta(t)$ is the step function. Then we have

$$v(t) = v(2t) + v(2t-1), \tag{6.66}$$

which gives $h(0) = h(1) = 1/\sqrt{2}$ and $h(k) = 0$ for $k \neq 0, 1$. We also have $g(0) = 1/\sqrt{2}$, $g(1) = -1/\sqrt{2}$, and $g(k) = 0$ if $k \neq 0, 1$, which comes from $g(k) = (-1)^k h(n-1-k) = (-1)^k h(1-k)$, with $n = 2$. Considering now $n = 4$, we have

$$\begin{aligned} h(0) &= \frac{1+\sqrt{3}}{4\sqrt{2}};\ h(1) = \frac{3+\sqrt{3}}{4\sqrt{2}}; \\ h(2) &= \frac{3-\sqrt{3}}{4\sqrt{2}};\ h(3) = \frac{1-\sqrt{3}}{4\sqrt{2}}, \end{aligned} \tag{6.67}$$

which is commonly known as the D4 wavelet, named after Daubechies (1988). For even larger n, however, we may have to solve the n coupled equations for $h(k)$ numerically.

A better scheme can be devised (Daubechies, 1988) through the zeros z_k of a polynomial

$$p(z) = \sum_{k=0}^{n/2-1} \frac{(n/2-1+k)!}{k!(n/2-1)!} \frac{(z-1)^{2k}}{(-z)^k} \tag{6.68}$$

inside the unit circle. The coefficients $h(k)$ are given by $h(k) = b_k/\sqrt{\mathcal{N}}$, where b_k is from the expansion

$$(z+1)^{n/2}\prod_{k=1}^{n/2-1}(z-z_k) = \sum_{k=0}^{n-1} b_k z^{n-k-1} \tag{6.69}$$

and the normalization constant $\mathcal{N} = \sum_{k=0}^{n-1} b_k^2$.

In general, we can express the basis functions for the spaces V_{j-1} and W_{j-1} in terms of those for the space V_j as

$$v_{j-1\,k}(t) = \sum_{l=-\infty}^{\infty} h(l) v_{j\,2k+l}(t), \tag{6.70}$$

$$u_{j-1\,k}(t) = \sum_{l=-\infty}^{\infty} g(l) v_{j\,2k+l}(t), \tag{6.71}$$

following Eqs. (6.47), (6.57), (6.58), and (6.59). This provides a means for us to decompose a given set of data level by level, starting from any chosen resolution.

Filter bank and pyramid algorithm

Now let us turn to the analysis of the data in the spaces V_j and W_j. If we start by approximating the data function $f(t)$ by $A_0(t) = a_j(t)$ with a reasonably large j, we can perform the entire wavelet analysis level by level. First we decompose $A_0(t)$ once to have

$$A_0(t) = \sum_{k=-\infty}^{\infty} a^{(0)}(k) v_{jk}(t) = A_1(t) + D_1(t), \tag{6.72}$$

where

$$A_1(t) = \sum_{k=-\infty}^{\infty} a^{(1)}(k) v_{j-1k}(t), \tag{6.73}$$

$$D_1(t) = \sum_{k=-\infty}^{\infty} d^{(1)}(k) u_{j-1k}(t). \tag{6.74}$$

This is the consequence of $V_j = V_{j-1} \oplus W_{j-1}$, and $v_{jk}(t)$ for all integers k form the basis set for V_j and $u_{jk}(t)$ for all integers k form the basis set for W_j. The coefficients in the above expression are obtained from

$$a^{(1)}(k) = \langle v_{j-1\,k} | A_1 \rangle = \langle v_{j-1\,k} | A_0 \rangle = \sum_{l=-\infty}^{\infty} h(l-2k) a^{(0)}(l), \tag{6.75}$$

$$d^{(1)}(k) = \langle u_{j-1\,k} | D_1 \rangle = \langle u_{j-1\,k} | A_0 \rangle = \sum_{l=-\infty}^{\infty} g(l-2k) a^{(0)}(l). \tag{6.76}$$

We can then decompose $A_1(t)$ into $A_2(t)$ and $D_2(t)$ to obtain the next level analysis. The transform is used to find $a^{(n)}(k)$ and $d^{(n)}(k)$, or $A_n(t)$ and $D_n(t)$, provided the filters $h(k)$ and $g(k)$ are given. Now if we know $a^{(0)}(k)$ and the filters,

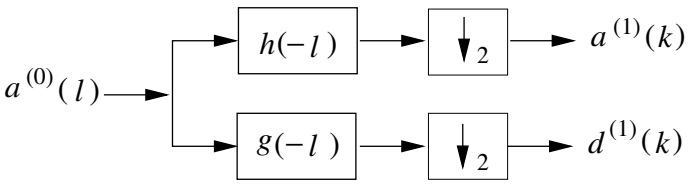

Fig. 6.5 The filtering and downsampling processes of the first-level decomposition in the discrete wavelet transform.

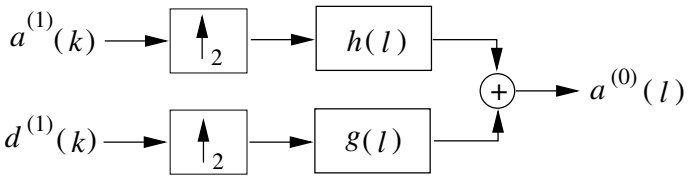

Fig. 6.6 The filtering and upsampling processes of the one-level reconstruction in the discrete wavelet transform.

we can obtain the transform, that is, calculate $a^{(n)}(k)$ and $d^{(n)}(k)$ for a given integer n, easily. As soon as we have the scaling function and its corresponding filters specified, we also have the wavelet from the second two-scale relation and the coefficients $a^{(0)}(k) = \langle v(t-k)|f(t)\rangle$. A digital filter is defined by the convolution

$$a * b(t) = \sum_{k=-\infty}^{\infty} a(k)b(t-k) = \sum_{k=-\infty}^{\infty} a(k-t)b(t), \tag{6.77}$$

where $a(t)$ is called the filter and the time sequence $b(t)$ is the data function that is filtered by $a(t)$. From Eqs. (6.75) and (6.76), we see that the filters associated with the wavelet analysis are time-reversed and skip every other data point during the filtering, which is called *downsampling*. We therefore can represent each level of analysis by two operations, the filtering and downsampling, as shown in Fig. 6.5. The downsampling is equivalent to converting function $x(k)$ into $x(2k)$. The analysis can be continued with the same pair of filters to the next level with a decomposition of $A_1(t)$ into $A_2(t)$ and $D_2(t)$, and so forth, forming a multistage filter bank pyramid. We can design a pyramid algorithm to carry out n levels of decompositions to obtain $\{a^{(n)}, d^{(n)}, \ldots, d^{(1)}\}$, for $n \geq 1$.

The inverse transform (synthesis) can be obtained with the same pair of filters without reversing the time. From the expansion of Eq. (6.72), we have

$$\begin{aligned} a^{(0)}(k) &= \langle v_{jk}(t)|A_0(t)\rangle \\ &= \sum_{l=-\infty}^{\infty} a^{(1)}(l)h(k-2l) + \sum_{l=-\infty}^{\infty} d^{(1)}(l)g(k-2l), \end{aligned} \tag{6.78}$$

which can be interpreted as a combination of an *upsampling* and a filtering operation. The upsampling is equivalent to converting function $x(k)$ into $x(k/2)$ for even k and into zero for odd k. Graphically, we can express this one level of synthesis as that shown in Fig. 6.6. Of course, the process can be continued to the second level of synthesis, the third level of synthesis, and so forth, until we have the time sequence completely reconstructed.

Like the discrete Fourier transform, the discrete wavelet transform can also be used in higher-dimensional spaces. Interested readers can find discussions of the two-dimensional wavelet transform in Newland (1993). There are many other scaling functions and wavelets available; interested readers should consult the references cited at the beginning of the preceding section. For a particular problem, one may work better than others.

6.8 Special functions

The solutions of a special set of differential equations can be expressed in terms of polynomials. In some cases, each polynomial has an infinite number of terms. For example, the Schrödinger equation for a particle in a central potential $V(r)$ is

$$-\frac{\hbar^2}{2m}\nabla^2\Psi(\mathbf{r}) + V(r)\Psi(\mathbf{r}) = \varepsilon\Psi(\mathbf{r}), \tag{6.79}$$

with

$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}r^2\frac{\partial}{\partial r} + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\sin\theta\frac{\partial}{\partial\theta} + \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\phi^2} \tag{6.80}$$

in spherical coordinates. We can assume that the solution $\Psi(\mathbf{r})$ is of the form

$$\Psi(r,\theta,\phi) = R(r)Y(\theta,\phi), \tag{6.81}$$

and then the equation becomes

$$\begin{aligned}\frac{1}{R}\frac{d}{dr}r^2\frac{dR}{dr} + \frac{2mr^2}{\hbar^2}(\varepsilon - V) &= -\frac{1}{Y}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\sin\theta\frac{\partial Y}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2 Y}{\partial\phi^2}\right)\\ &= \lambda,\end{aligned} \tag{6.82}$$

where λ is an introduced parameter to be determined by solving the eigenvalue problem of $Y(\theta,\phi)$. We can further assume that $Y(\theta,\phi) = \Theta(\theta)\Phi(\phi)$ and then the equation for $\Theta(\theta)$ becomes

$$\frac{1}{\sin\theta}\frac{d}{d\theta}\sin\theta\frac{d\Theta}{d\theta} + \left(\lambda - \frac{m^2}{\sin^2\theta}\right)\Theta = 0, \tag{6.83}$$

with the corresponding equation for $\Phi(\phi)$ given as

$$\Phi''(\phi) = -m^2\Phi(\phi), \tag{6.84}$$

where m is another introduced parameter and has to be an integer, because $\Phi(\phi)$ needs to be a periodic function of ϕ, that is, $\Phi(\phi + 2\pi) = \Phi(\phi)$. Note that $\Phi(\phi) = Ae^{im\phi} + Be^{-im\phi}$ is the solution of the differential equation for $\Phi(\phi)$.

In order for the solution of $\Theta(\theta)$ to be finite everywhere with $\theta \in [0,\pi]$, we must have $\lambda = l(l+1)$, where l is a positive integer. Such solutions $\Theta(\theta) = P_l^m(\cos\theta)$ are called associated Legendre polynomials and can be written as

$$P_l^m(x) = (1-x^2)^{m/2}\frac{d^m}{dx^m}P_l(x), \tag{6.85}$$

where $P_l(x)$ are the Legendre polynomials that satisfy the recursion

$$(l+1)P_{l+1}(x) = (2l+1)x\,P_l(x) - l\,P_{l-1}(x), \tag{6.86}$$

starting with $P_0(x) = 1$ and $P_1(x) = x$. We can then obtain all $P_l(x)$ from Eq. (6.86) with $l = 2, 3, 4, \ldots$. We can easily show that $P_l(x)$ is indeed the solution of the equation

$$\frac{d}{dx}(1-x^2)\frac{d}{dx}P_l(x) + l(l+1)P_l(x) = 0 \tag{6.87}$$

and is a polynomial of order l in the region of $x \in [-1, 1]$, as discussed in Section 5.8 in the calculation of the electronic structures of atoms.

Legendre polynomials are useful in almost every subfield of physics and engineering where partial differential equations involving spherical coordinates need to be solved. For example, the electrostatic potential of a charge distribution $\rho(\mathbf{r})$ can be written as

$$\Phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int \frac{\rho(\mathbf{r}')\,d\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|}, \tag{6.88}$$

where $1/|\mathbf{r}-\mathbf{r}'|$ can be expanded with the application of Legendre polynomials as

$$\frac{1}{|\mathbf{r}-\mathbf{r}'|} = \sum_{l=0}^{\infty}\frac{r_<^l}{r_>^{l+1}}P_l(\cos\theta), \tag{6.89}$$

where $r_<$ ($r_>$) is the smaller (greater) value of r and r' and θ is the angle between $\mathbf{r}$ and $\mathbf{r}'$. So if we can generate $P_l(\cos\theta)$, we can evaluate the electrostatic potential for any given charge distribution $\rho(\mathbf{r})$ term by term.

The Legendre polynomials can be produced from the recursion given in Eq. (6.86). Below is a simple implementation of the recursion to create the Legendre polynomials for a given x and a maximum l.

```
// Method to create the Legendre polynomials p_l(x) for
// a given x and maximum l.

  public static double[] p (double x, int lmax) {
    double pl[] = new double[lmax+1];
    pl[0] = 1;
    if (lmax==0) return pl;
    else {
      pl[1] = x;
      if (lmax==1) return pl;
      else {
        for (int l=1; l<lmax; ++l)
          pl[l+1] = ((2*l+1)*x*pl[l]
                    -(l+1)*pl[l-1])/(l+1);
        return pl;
      }
    }
  }
```

Legendre polynomials $P_l(x)$ form a complete set of orthogonal basis functions in the region of $x \in [-1, 1]$, which can be used to achieve the least-squares

approximations, as discussed in Chapter 2, or to accomplish a generalized Fourier transform. If we choose

$$U_l(x) = \sqrt{\frac{2l+1}{2}} P_l(x), \tag{6.90}$$

we can easily show that $U_l(x)$ satisfy

$$\int_{-1}^{1} U_l(x) U_{l'}(x)\, dx = \delta_{ll'}. \tag{6.91}$$

A function $f(x)$ in the region of $x \in [-1, 1]$ can be written as

$$f(x) = \sum_{l=0}^{\infty} a_l U_l(x) \tag{6.92}$$

with

$$a_l = \int_{-1}^{1} f(x) U_l(x)\, dx. \tag{6.93}$$

This is a generalized Fourier transform of the function $f(t)$. In fact, almost every aspect associated with the Fourier transform can be generalized to other orthogonal functions that form a complete basis set. There is a whole class of orthogonal polynomials that are similar to Legendre polynomials and can be applied to similar problems in the same fashion. Detailed discussions on these orthogonal polynomials can be found in Hochstrasser (1965).

In the case of cylindrical coordinates, the equation governing the Laplace operator in the radial direction is the so-called Bessel equation

$$\frac{d^2 J(x)}{dx^2} + \frac{1}{x}\frac{dJ(x)}{dx} + \left(1 - \frac{\nu^2}{x^2}\right) J(x) = 0, \tag{6.94}$$

where ν is a parameter and the solution of the equation is called the Bessel function of order ν. Here ν can be an integer or a fraction. The recursion for the Bessel functions is

$$J_{\nu\pm1}(x) = \frac{2\nu}{x} J_\nu(x) - J_{\nu\mp1}(x), \tag{6.95}$$

which can be used in a similar fashion to Legendre polynomials to generate $J_{\nu\pm1}(x)$ from $J_\nu(x)$ and $J_{\nu\mp1}(x)$.

Bessel functions can be further divided into two types, depending on their asymptotic behavior. We call the ones with finite values as $x \to 0$ functions of the first kind and denote them by $J_\nu(x)$; the ones that diverge as $x \to 0$ are called functions of the second kind and are denoted by $Y_\nu(x)$. We will consider only the case where ν is an integer. Both $J_\nu(x)$ and $Y_\nu(x)$ can be applied to a generalized Fourier transform because they form orthogonal basis sets under certain given conditions: for example,

$$\int_0^a J_\nu(\kappa_{\nu k}\rho) J_\nu(\kappa_{\nu l}\rho)\rho\, d\rho = \frac{a^2}{2} J_{\nu+1}^2(x_{\nu k})\delta_{kl}, \tag{6.96}$$

where $\kappa_{\nu k}$ and $x_{\nu k} = \kappa_{\nu k} a$ are from the kth zero (root) of $J_\nu(x) = 0$. Then for a function $f(\rho)$ defined in the region $\rho \in [0, a]$, we have

$$f(\rho) = \sum_{k=1}^{\infty} A_{\nu k} J_\nu(\kappa_{\nu k} \rho), \tag{6.97}$$

with

$$A_{\nu k} = \frac{2}{a^2 J_{\nu+1}^2(x_{\nu k})} \int_0^a f(\rho) J_\nu(\kappa_{\nu k} \rho) \rho \, d\rho. \tag{6.98}$$

In practice, there are two more problems in generating Bessel functions numerically. Bessel functions have an infinite number of terms in the series representation, so it is difficult to initiate the recursion numerically, and the Bessel function of the second kind, $Y_\nu(x)$, increases exponentially with ν when $\nu > x$.

These problems can be resolved if we carry out the recursion forward for $Y_\nu(x)$ and backward for $J_\nu(x)$. We can use several properties of the functions to initiate the recursion. For $J_\nu(x)$, we can set the first two points as $J_N(x) = 0$ and $J_{N-1}(x) = 1$ in the backward recursion. The functions generated can be rescaled afterward with

$$R_N = J_0(x) + 2J_2(x) + \cdots + 2J_N(x), \tag{6.99}$$

because $R_\infty = 1$ for the actual Bessel functions and

$$\lim_{\nu \to \infty} J_\nu(x) = 0. \tag{6.100}$$

For $Y_\nu(x)$, we can use the values obtained for $J_0(x), J_1(x), \ldots, J_N(x)$ to initiate the first two points,

$$Y_0(x) = \frac{2}{\pi} \left(\ln \frac{x}{2} + \gamma \right) J_0(x) - \frac{4}{\pi} \sum_{k=1}^{\infty} (-1)^k \frac{J_{2k}(x)}{k} \tag{6.101}$$

and

$$Y_1(x) = \frac{1}{J_0(x)} \left[J_1(x) Y_0(x) - \frac{2}{\pi x} \right], \tag{6.102}$$

in the forward recursion. Here γ is the Euler constant,

$$\gamma = \lim_{N \to \infty} \left(\sum_{k=1}^{N} \frac{1}{k} - \ln N \right) = 0.577\,215\,664\,9 \ldots. \tag{6.103}$$

The summation in Eq. (6.101) is truncated at $2k = N$, which should not introduce too much error into the functions, because $J_\nu(x)$ exponentially decreases with ν for $\nu > x$. The following method is an implementation of the above schemes for generating Bessel functions of the first and second kinds.

```
// Method to create Bessel functions for a given x, index n,
// and index buffer nb.

  public static double[][] b(double x, int n, int nb) {
    int nmax = n+nb;
    double g = 0.5772156649;
```

```
    double y[][] = new double[2][n+1];
    double z[] = new double[nmax+1];
// Generate the Bessel function of 1st kind J_n(x)
    z[nmax-1] = 1;
    double s = 0;
    for (int i=nmax-1; i>0; --i) {
      z[i-1] = 2*i*z[i]/x-z[i+1];
      if (i%2 == 0) s += 2*z[i];
    }
    s += z[0];
    for (int i=0; i<=n; ++i) y[0][i] = z[i]/s;
// Generate the Bessel function of 2nd kind Y_n(x)
    double t = 0;
    int sign = -1;
    for (int i=1; i<nmax/2; ++i) {
      t += sign*z[2*i]/i;
      sign *= -1;
    }
    t *= -4/(Math.PI*s);
    y[1][0] = 2*(Math.log(x/2)+g)*y[0][0]/Math.PI+t;
    y[1][1] = (y[0][1]*y[1][0]-2/(Math.PI*x))/y[0][0];
    for (int i=1; i<n; ++i)
      y[1][i+1] = 2*i*y[1][i]/x-y[1][i-1];
    return y;
  }
```

If we only need to generate $J_\nu(x)$, the lines associated with $Y_\nu(x)$ can be deleted from the above method. Note that the variable x needs to be smaller than the maximum ν in order for the value of $J_\nu(x)$ to be accurate.

6.9 Gaussian quadratures

We can use special functions to construct numerical integration quadratures that automatically minimize the possible errors due to the deviation of approximate functions from the data, the same concept used in Chapter 3 to obtain the approximation of a function by orthogonal polynomials. In fact, all special functions form basis sets for certain vector spaces.

An integral defined in the region $[a, b]$ can be written as

$$I = \int_a^b w(x) f(x)\, dx, \tag{6.104}$$

where $w(x)$ is the weight of the integral, which has exactly the same meaning as the weight of the orthogonal functions defined in Chapter 3. Examples of $w(x)$ will be given in this section.

Now we divide the region $[a, b]$ into n points ($n - 1$ slices) and approximate the integral as

$$I \simeq \sum_{k=1}^{n} w_k f(x_k), \tag{6.105}$$

with x_k and w_k determined according to two criteria: the simplicity of the expression and the accuracy of the approximation.

The expression in Eq. (6.105) is quite general and includes the simplest quadratures introduced in Chapter 3 with $w(x) = 1$. For example, if we take

$$x_k = a + \frac{k-1}{n-1}(b-a) \tag{6.106}$$

for $k = 1, 2, \ldots, n$ and

$$w_k = \frac{1}{n(1 + \delta_{k1} + \delta_{kn})}, \tag{6.107}$$

we recover the trapezoid rule. The Simpson rule is recovered if we take a more complicated w_k with

$$w_k = \begin{cases} 1/3n & \text{for } k = 1 \text{ or } n, \\ 1/n & \text{for } k = \text{even numbers}, \\ 2/3n & \text{for } k = \text{odd numbers}, \end{cases} \tag{6.108}$$

but still the same set of x_k with $w(x) = 1$.

The so-called Gaussian quadrature is constructed from a set of orthogonal polynomials $\phi_l(x)$ with

$$\int_a^b \phi_l(x) w(x) \phi_k(x)\, dx = \langle \phi_l | \phi_k \rangle = \mathcal{N}_l \delta_{lk}, \tag{6.109}$$

where the definition of each quantity is exactly the same as in Section 3.2. We can show that to choose x_k to be the kth root of $\phi_n(x) = 0$ and to choose

$$w_k = \frac{-a_n \mathcal{N}_n}{\phi_n'(x_k)\phi_{n+1}(x_k)}, \tag{6.110}$$

with $n = 1, 2, \ldots, N$, we ensure that the error in the quadrature is given by

$$\Delta I = \frac{\mathcal{N}_n}{A_n^2 (2n)!} f^{(2n)}(x_0), \tag{6.111}$$

where x_0 is a value of $x \in [a, b]$, A_n is the coefficient of the x^n term in $\phi_n(x)$, and $a_n = A_{n+1}/A_n$. We can use any kind of orthogonal polynomials for this purpose. For example, with the Legendre polynomials, we have $a = -1, b = 1, w(x) = 1$, $a_n = (2n+1)/(n+1)$, and $\mathcal{N}_n = 2/(2n+1)$.

Another set of very useful orthogonal polynomials is the Chebyshev polynomials, defined in the region $[-1, 1]$. For example, the recursion relation for the Chebyshev polynomials of the first kind is

$$T_{k+1}(x) = 2x\, T_k(x) - T_{k-1}(x), \tag{6.112}$$

starting with $T_0(x) = 1$ and $T_1(x) = x$. We can easily show that

$$\int_{-1}^{1} T_k(x) w(x) T_l(x)\, dx = \delta_{kl}, \tag{6.113}$$

with $w(x) = 1/\sqrt{1-x^2}$. If we write as a series an integral with the same weight,

$$\int_{-1}^{1} \frac{f(x)\,dx}{\sqrt{1-x^2}} \simeq \sum_{k=1}^{n} w_k f(x_k), \tag{6.114}$$

we have

$$x_k = \cos\frac{(2k-1)\pi}{2n} \tag{6.115}$$

and $w_k = \pi/n$, which are extremely easy to use. Note that we can translate a given integration region $[a, b]$ to another, for example, $[0, 1]$ or $[-1, 1]$, through a linear coordinate transformation.

Exercises

6.1 Run the discrete Fourier transform program and the fast Fourier transform program on a computer and establish the time dependence on the number of points for both of them. Would it be different if the processor were a vector processor, that is, if a segment of the inner loop were performed in one clock cycle?

6.2 Develop a method that implements the fast Fourier transform for N data points with $N = 4^M$, where M is an integer. Is this algorithm faster that the one with $N = 2^M$ given in Section 6.3 when applied to the same data stream?

6.3 Analyze the power spectrum of the Duffing model defined in Exercise 4.6. What is the most significant difference between the Duffing model and the driven pendulum with damping?

6.4 Develop a method that carries out the continuous wavelet transform with the Haar wavelet. Apply the method to the angular data in the driven pendulum with damping. Illustrate the wavelet coefficients in a surface plot. What can we learn from the wavelet coefficients at different parameter regions?

6.5 Show that the real-value Morlet wavelet

$$u(t) = e^{-t^2}\cos\left(\pi\sqrt{\frac{2}{\ln 2}}t\right)$$

satisfies the admissibility condition roughly. Apply this wavelet in a continuous wavelet transform to the time-dependent function $f(t) = 1$ for $t \in [0, 1]$, zero otherwise. Plot the scalogram of $f(t)$ and discuss the features shown.

6.6 Show that the wavelet

$$u(t) = e^{-t^2}(1 - 2t^2)$$

satisfies the admissibility condition. Apply this wavelet in a continuous wavelet transform to a sequence $f_i = f(t_i)$ from a uniform random-number generator with the assumption that t_i are uniformly

spaced in the region [0, 1]. Does the scalogram show any significant structures?

6.7 Solve the equation set outlined in Section 6.7 for the scaling filters for $n = 4, 8, 16$. Is the result for the $n = 4$ the same as the D4 wavelet?

6.8 Find the zeros of the polynomial given in Eq. (6.68) inside the unit circle, and then use Eq. (6.69) to obtain the scaling filters for $n = 4, 8, 16$. Is the result for the $n = 4$ case the same as the D4 wavelet?

6.9 Develop a method that performs the pyramid algorithm in the discrete wavelet transform under the D4 wavelet. Apply the method to the displacement data in the Duffing model. What can we learn from both the wavelet and scaling coefficients at different parameter regions?

6.10 Based on the inverse pyramid algorithm, write a method that recovers the original data from the wavelet and scaling coefficients. Test the method with a simple data stream generated from a random-number generator.

6.11 Write a method that generates the Chebyshev polynomials of the first kind discussed in Section 6.9.

6.12 The Laguerre polynomials form an orthogonal basis set in the region $[0, \infty]$ and satisfy the following recursion

$$(n+1)L_{n+1}(x) = (2n+1-x)L_n(x) - nL_{n-1}(x),$$

starting with $L_0(x) = 1$ and $L_1(x) = 1 - x$. (a) Write a subroutine that generates the Laguerre polynomials for a given x and maximum n. (b) Show that the weight of the polynomials $w(x) = e^{-x}$, and that the normalization factor $\mathcal{N}_n = \int_0^\infty w(x)L_n^2(x)dx = 1$. (c) If we want to construct the Gaussian quadrature for the integral

$$I = \int_0^\infty e^{-x} f(x)\, dx = \sum_{n=1}^{N} w_n f(x_n) + \Delta I,$$

where x_n is the nth root of $L_N(x) = 0$, show that w_n is given by

$$w_n = \frac{(N!)^2 x_n}{(N+1)^2 L_{N+1}^2(x_n)}.$$

This quadrature is extremely useful in statistical physics, where the integrals are typically weighted with an exponential function.

6.13 The Hermite polynomials are another set of important orthogonal polynomials in the region $[-\infty, \infty]$ and satisfy the following recursion

$$H_{n+1}(x) = 2x H_n(x) - 2n H_{n-1}(x),$$

starting with $H_0(x) = 1$ and $H_1(x) = 2x$. (a) Write a subroutine that generates the Hermite polynomials for a given x and maximum n. (b) Show that the weight of the polynomials $w(x) = e^{-x^2}$ and that the normalization factor $\mathcal{N}_n = \int_{-\infty}^\infty w(x)H_n^2(x)dx = \sqrt{\pi}2^n n!$. (c) Now if we want to construct

the Gaussian quadrature for the integral

$$I = \int_{\infty}^{\infty} e^{-x^2} f(x)\, dx = \sum_{n=1}^{N} w_n f(x_n) + \Delta I,$$

where x_n is the nth root of $H_N(x) = 0$, show that w_n is given by

$$w_n = \frac{2^{N-1} N! \sqrt{\pi}}{N^2 H_{N-1}^2(x_n)}.$$

Many integrals involving the Gaussian distribution can take advantage of this quadrature.

6.14 The integral expressions for the Bessel functions of the first and second kinds of order zero are given by

$$J_0(x) = \frac{2}{\pi} \int_0^{\infty} \sin(x \cosh t)\, dt,$$
$$Y_0(x) = -\frac{2}{\pi} \int_0^{\infty} \cos(x \cosh t)\, dt,$$

with $x > 0$. Calculate these two expressions numerically and compare them with the values generated with the subroutine in Section 6.8.

6.15 Demonstrate numerically that for a small angle θ but large l,

$$P_l(\cos\theta) \simeq J_0(l\theta).$$

6.16 In quantum scattering, when the incident particle has very high kinetic energy, the cross section

$$\sigma = 2\pi \int_0^{\pi} \sin\theta |f(\theta)|^2\, d\theta$$

has a very simple form with

$$f(\theta) \simeq -\frac{2m}{\hbar^2} \int_0^{\infty} r V(x) \sin(qr)\, dr,$$

where $q = |\mathbf{k}_f - \mathbf{k}_i| = 2k \sin(\theta/2)$ and $\mathbf{k}_i$ and $\mathbf{k}_f$ are initial and final momenta of the particle with the same magnitude k. (a) Show that the above $f(\theta)$ is the dominant term as $k \to \infty$. (b) If the particle is an electron and the scattering potential is an ionic potential

$$V(r) = \frac{1}{4\pi\epsilon_0} \frac{Ze^2}{r} e^{-r/r_0},$$

write a program to evaluate the cross section of the scattering with the Gaussian quadrature from the Laguerre polynomials for the integrals. Here Z, e, and r_0 are the number of protons, the proton charge, and the screening length. Use $Z = 2$ and $r_0 = a_0/4$, where a_0 is the Bohr radius, as a testing example.

6.17 Another approximation in quantum scattering, called the eikonal approximation, is valid for high-energy and small-angle scattering. The scattering

cross section can be written as

$$\sigma = 8\pi \int_0^\infty b \sin^2[\alpha(b)]\, db,$$

with

$$\alpha(b) = -\frac{m}{2\hbar^2 k} \int_{-\infty}^{\infty} V(b, x)\, dx,$$

where b is the impact parameter and x is the coordinate of the particle along the impact direction with the scattering center at $x = 0$. (a) Show that the eikonal approximation is valid for high-energy and small-angle scattering. (b) If the scattering potential is

$$V(r) = -V_0 e^{-(r/r_0)^2},$$

calculate the cross section with the Gaussian quadratures of Hermite and Laguerre polynomials for the integrals. Here V_0 and r_0 are given parameters.