```
    ix=(int)x;
    if (x == (float)ix) yy[ix] += y;
    else {
        ilo=LMIN(LMAX((long)(x-0.5*m+1.0),1),n-m+1);
        ihi=ilo+m-1;
        nden=nfac[m];
        fac=x-ilo;
        for (j=ilo+1;j<=ihi;j++) fac *= (x-j);
        yy[ihi] += y*fac/(nden*(x-ihi));
        for (j=ihi-1;j>=ilo;j--) {
            nden=(nden/(j+1-ilo))*(j-ihi);
            yy[j] += y*fac/(nden*(x-j));
        }
    }
}
```

CITED REFERENCES AND FURTHER READING:

Lomb, N.R. 1976, *Astrophysics and Space Science*, vol. 39, pp. 447–462. [1]

Barning, F.J.M. 1963, *Bulletin of the Astronomical Institutes of the Netherlands*, vol. 17, pp. 22–28. [2]

Vaníček, P. 1971, *Astrophysics and Space Science*, vol. 12, pp. 10–33. [3]

Scargle, J.D. 1982, *Astrophysical Journal*, vol. 263, pp. 835–853. [4]

Horne, J.H., and Baliunas, S.L. 1986, *Astrophysical Journal*, vol. 302, pp. 757–763. [5]

Press, W.H. and Rybicki, G.B. 1989, *Astrophysical Journal*, vol. 338, pp. 277–280. [6]

## 13.9 Computing Fourier Integrals Using the FFT

Not uncommonly, one wants to calculate accurate numerical values for integrals of the form

$$I = \int_a^b e^{i\omega t} h(t) dt \, , \tag{13.9.1}$$

or the equivalent real and imaginary parts

$$I_c = \int_a^b \cos(\omega t) h(t) dt \qquad I_s = \int_a^b \sin(\omega t) h(t) dt \, , \tag{13.9.2}$$

and one wants to evaluate this integral for many different values of $\omega$. In cases of interest, $h(t)$ is often a smooth function, but it is not necessarily periodic in $[a, b]$, nor does it necessarily go to zero at $a$ or $b$. While it seems intuitively obvious that the *force majeure* of the FFT ought to be applicable to this problem, doing so turns out to be a surprisingly subtle matter, as we will now see.

Let us first approach the problem naively, to see where the difficulty lies. Divide the interval $[a, b]$ into $M$ subintervals, where $M$ is a large integer, and define

$$\Delta \equiv \frac{b-a}{M}, \quad t_j \equiv a + j\Delta, \quad h_j \equiv h(t_j), \quad j = 0, \ldots, M \tag{13.9.3}$$

Notice that $h_0 = h(a)$ and $h_M = h(b)$, and that there are $M + 1$ values $h_j$. We can approximate the integral $I$ by a sum,

$$I \approx \Delta \sum_{j=0}^{M-1} h_j \exp(i\omega t_j) \tag{13.9.4}$$

which is at any rate first-order accurate. (If we centered the $h_j$'s and the $t_j$'s in the intervals, we could be accurate to second order.) Now for certain values of $\omega$ and $M$, the sum in equation (13.9.4) can be made into a discrete Fourier transform, or DFT, and evaluated by the fast Fourier transform (FFT) algorithm. In particular, we can choose $M$ to be an integer power of 2, and define a set of special $\omega$'s by

$$\omega_m \Delta \equiv \frac{2\pi m}{M} \tag{13.9.5}$$

where $m$ has the values $m = 0, 1, \dots, M/2 - 1$. Then equation (13.9.4) becomes

$$I(\omega_m) \approx \Delta e^{i\omega_m a} \sum_{j=0}^{M-1} h_j e^{2\pi i m j/M} = \Delta e^{i\omega_m a} [\text{DFT}(h_0 \dots h_{M-1})]_m \tag{13.9.6}$$

Equation (13.9.6), while simple and clear, is emphatically *not recommended* for use: It is likely to give wrong answers!

The problem lies in the oscillatory nature of the integral (13.9.1). If $h(t)$ is at all smooth, and if $\omega$ is large enough to imply several cycles in the interval $[a, b]$ — in fact, $\omega_m$ in equation (13.9.5) gives exactly $m$ cycles — then the value of $I$ is typically very small, so small that it is easily swamped by first-order, or even (with centered values) second-order, truncation error. Furthermore, the characteristic "small parameter" that occurs in the error term is not $\Delta/(b-a) = 1/M$, as it would be if the integrand were not oscillatory, but $\omega\Delta$, which can be as large as $\pi$ for $\omega$'s within the Nyquist interval of the DFT (cf. equation 13.9.5). The result is that equation (13.9.6) becomes systematically inaccurate as $\omega$ increases.

It is a sobering exercise to implement equation (13.9.6) for an integral that can be done analytically, and to see just how bad it is. We recommend that you try it.

Let us therefore turn to a more sophisticated treatment. Given the sampled points $h_j$, we can approximate the function $h(t)$ everywhere in the interval $[a, b]$ by interpolation on nearby $h_j$'s. The simplest case is linear interpolation, using the two nearest $h_j$'s, one to the left and one to the right. A higher-order interpolation, e.g., would be cubic interpolation, using two points to the left and two to the right — except in the first and last subintervals, where we must interpolate with three $h_j$'s on one side, one on the other.

The formulas for such interpolation schemes are (piecewise) polynomial in the independent variable $t$, but with coefficients that are of course linear in the function values $h_j$. Although one does not usually think of it in this way, interpolation can be viewed as approximating a function by a sum of kernel functions (which depend only on the interpolation scheme) times sample values (which depend only on the function). Let us write

$$h(t) \approx \sum_{j=0}^{M} h_j \, \psi\left(\frac{t-t_j}{\Delta}\right) + \sum_{j=\text{endpoints}} h_j \, \varphi_j\left(\frac{t-t_j}{\Delta}\right) \tag{13.9.7}$$

Here $\psi(s)$ is the kernel function of an interior point: It is zero for $s$ sufficiently negative or sufficiently positive, and becomes nonzero only when $s$ is in the range where the $h_j$ multiplying it is actually used in the interpolation. We always have $\psi(0) = 1$ and $\psi(m) = 0$, $m = \pm1, \pm2, \dots$, since interpolation right on a sample point should give the sampled function value. For linear interpolation $\psi(s)$ is piecewise linear, rises from 0 to 1 for $s$ in $(-1, 0)$, and falls back to 0 for $s$ in $(0, 1)$. For higher-order interpolation, $\psi(s)$ is made up piecewise of segments of Lagrange interpolation polynomials. It has discontinuous derivatives at integer values of $s$, where the pieces join, because the set of points used in the interpolation changes discretely.

As already remarked, the subintervals closest to $a$ and $b$ require different (noncentered) interpolation formulas. This is reflected in equation (13.9.7) by the second sum, with the special endpoint kernels $\varphi_j(s)$. Actually, for reasons that will become clearer below, we have included *all* the points in the *first* sum (with kernel $\psi$), so the $\varphi_j$'s are actually differences between true endpoint kernels and the interior kernel $\psi$. It is a tedious, but straightforward, exercise to write down all the $\varphi_j(s)$'s for any particular order of interpolation, each one consisting of differences of Lagrange interpolating polynomials spliced together piecewise.

Now apply the integral operator $\int_a^b dt \exp(i\omega t)$ to both sides of equation (13.9.7), interchange the sums and integral, and make the changes of variable $s = (t - t_j)/\Delta$ in the first sum, $s = (t - a)/\Delta$ in the second sum. The result is

$$I \approx \Delta e^{i\omega a}\left[W(\theta)\sum_{j=0}^{M} h_j e^{ij\theta} + \sum_{j=\text{endpoints}} h_j \alpha_j(\theta)\right] \tag{13.9.8}$$

Here $\theta \equiv \omega\Delta$, and the functions $W(\theta)$ and $\alpha_j(\theta)$ are defined by

$$W(\theta) \equiv \int_{-\infty}^{\infty} ds\, e^{i\theta s}\psi(s) \tag{13.9.9}$$

$$\alpha_j(\theta) \equiv \int_{-\infty}^{\infty} ds\, e^{i\theta s}\varphi_j(s - j) \tag{13.9.10}$$

The key point is that equations (13.9.9) and (13.9.10) can be evaluated, analytically, once and for all, for any given interpolation scheme. Then equation (13.9.8) is an algorithm for applying "endpoint corrections" to a sum which (as we will see) can be done using the FFT, giving a result with high-order accuracy.

We will consider only interpolations that are left-right symmetric. Then symmetry implies

$$\varphi_{M-j}(s) = \varphi_j(-s) \qquad \alpha_{M-j}(\theta) = e^{i\theta M}\alpha_j^*(\theta) = e^{i\omega(b-a)}\alpha_j^*(\theta) \tag{13.9.11}$$

where * denotes complex conjugation. Also, $\psi(s) = \psi(-s)$ implies that $W(\theta)$ is real.

Turn now to the first sum in equation (13.9.8), which we want to do by FFT methods. To do so, choose some $N$ that is an integer power of 2 with $N \geq M + 1$. (Note that $M$ need not be a power of two, so $M = N - 1$ is allowed.) If $N > M + 1$, define $h_j \equiv 0$, $M + 1 < j \leq N - 1$, i.e., "zero pad" the array of $h_j$'s so that $j$ takes on the range $0 \leq j \leq N - 1$. Then the sum can be done as a DFT for the special values $\omega = \omega_n$ given by

$$\omega_n\Delta \equiv \frac{2\pi n}{N} \equiv \theta \qquad n = 0, 1, \ldots, \frac{N}{2} - 1 \tag{13.9.12}$$

For fixed $M$, the larger $N$ is chosen, the finer the sampling in frequency space. The value $M$, on the other hand, determines the *highest* frequency sampled, since $\Delta$ decreases with increasing $M$ (equation 13.9.3), and the largest value of $\omega\Delta$ is always just under $\pi$ (equation 13.9.12). In general it is advantageous to oversample by *at least* a factor of 4, i.e., $N > 4M$ (see below). We can now rewrite equation (13.9.8) in its final form as

$$\begin{aligned} I(\omega_n) = \Delta e^{i\omega_n a}\Big\{ & W(\theta)[\text{DFT}(h_0 \ldots h_{N-1})]_n \\ & + \alpha_0(\theta)h_0 + \alpha_1(\theta)h_1 + \alpha_2(\theta)h_2 + \alpha_3(\theta)h_3 + \ldots \\ & + e^{i\omega(b-a)}\left[\alpha_0^*(\theta)h_M + \alpha_1^*(\theta)h_{M-1} + \alpha_2^*(\theta)h_{M-2} + \alpha_3^*(\theta)h_{M-3} + \ldots\right]\Big\} \end{aligned} \tag{13.9.13}$$

For cubic (or lower) polynomial interpolation, at most the terms explicitly shown above are nonzero; the ellipses (...) can therefore be ignored, and we need explicit forms only for the functions $W, \alpha_0, \alpha_1, \alpha_2, \alpha_3$, calculated with equations (13.9.9) and (13.9.10). We have worked these out for you, in the trapezoidal (second-order) and cubic (fourth-order) cases. Here are the results, along with the first few terms of their power series expansions for small $\theta$:

**Trapezoidal order:**

$$W(\theta) = \frac{2(1 - \cos\theta)}{\theta^2} \approx 1 - \frac{1}{12}\theta^2 + \frac{1}{360}\theta^4 - \frac{1}{20160}\theta^6$$

$$\begin{aligned} \alpha_0(\theta) &= -\frac{(1 - \cos\theta)}{\theta^2} + i\frac{(\theta - \sin\theta)}{\theta^2} \\ &\approx -\frac{1}{2} + \frac{1}{24}\theta^2 - \frac{1}{720}\theta^4 + \frac{1}{40320}\theta^6 + i\theta\left(\frac{1}{6} - \frac{1}{120}\theta^2 + \frac{1}{5040}\theta^4 - \frac{1}{362880}\theta^6\right) \end{aligned}$$

$$\alpha_1 = \alpha_2 = \alpha_3 = 0$$

**Cubic order:**

$$W(\theta) = \left(\frac{6+\theta^2}{3\theta^4}\right)(3 - 4\cos\theta + \cos 2\theta) \approx 1 - \frac{11}{720}\theta^4 + \frac{23}{15120}\theta^6$$

$$\begin{aligned}\alpha_0(\theta) &= \frac{(-42+5\theta^2)+(6+\theta^2)(8\cos\theta - \cos 2\theta)}{6\theta^4} + i\frac{(-12\theta+6\theta^3)+(6+\theta^2)\sin 2\theta}{6\theta^4} \\ &\approx -\frac{2}{3} + \frac{1}{45}\theta^2 + \frac{103}{15120}\theta^4 - \frac{169}{226800}\theta^6 + i\theta\left(\frac{2}{45} + \frac{2}{105}\theta^2 - \frac{8}{2835}\theta^4 + \frac{86}{467775}\theta^6\right)\end{aligned}$$

$$\begin{aligned}\alpha_1(\theta) &= \frac{14(3-\theta^2) - 7(6+\theta^2)\cos\theta}{6\theta^4} + i\frac{30\theta - 5(6+\theta^2)\sin\theta}{6\theta^4} \\ &\approx \frac{7}{24} - \frac{7}{180}\theta^2 + \frac{5}{3456}\theta^4 - \frac{7}{259200}\theta^6 + i\theta\left(\frac{7}{72} - \frac{1}{168}\theta^2 + \frac{11}{72576}\theta^4 - \frac{13}{5987520}\theta^6\right)\end{aligned}$$

$$\begin{aligned}\alpha_2(\theta) &= \frac{-4(3-\theta^2) + 2(6+\theta^2)\cos\theta}{3\theta^4} + i\frac{-12\theta + 2(6+\theta^2)\sin\theta}{3\theta^4} \\ &\approx -\frac{1}{6} + \frac{1}{45}\theta^2 - \frac{5}{6048}\theta^4 + \frac{1}{64800}\theta^6 + i\theta\left(-\frac{7}{90} + \frac{1}{210}\theta^2 - \frac{11}{90720}\theta^4 + \frac{13}{7484400}\theta^6\right)\end{aligned}$$

$$\begin{aligned}\alpha_3(\theta) &= \frac{2(3-\theta^2) - (6+\theta^2)\cos\theta}{6\theta^4} + i\frac{6\theta - (6+\theta^2)\sin\theta}{6\theta^4} \\ &\approx \frac{1}{24} - \frac{1}{180}\theta^2 + \frac{5}{24192}\theta^4 - \frac{1}{259200}\theta^6 + i\theta\left(\frac{7}{360} - \frac{1}{840}\theta^2 + \frac{11}{362880}\theta^4 - \frac{13}{29937600}\theta^6\right)\end{aligned}$$

The program `dftcor`, below, implements the endpoint corrections for the cubic case. Given input values of $\omega, \Delta, a, b$, and an array with the eight values $h_0, \ldots, h_3, h_{M-3}, \ldots, h_M$, it returns the real and imaginary parts of the endpoint corrections in equation (13.9.13), and the factor $W(\theta)$. The code is turgid, but only because the formulas above are complicated. The formulas have cancellations to high powers of $\theta$. It is therefore necessary to compute the right-hand sides in double precision, even when the corrections are desired only to single precision. It is also necessary to use the series expansion for small values of $\theta$. The optimal cross-over value of $\theta$ depends on your machine's wordlength, but you can always find it experimentally as the largest value where the two methods give identical results to machine precision.

```
#include <math.h>

void dftcor(float w, float delta, float a, float b, float endpts[],
    float *corre, float *corim, float *corfac)
```
For an integral approximated by a discrete Fourier transform, this routine computes the correction factor that multiplies the DFT and the endpoint correction to be added. Input is the angular frequency `w`, stepsize `delta`, lower and upper limits of the integral `a` and `b`, while the array `endpts` contains the first 4 and last 4 function values. The correction factor $W(\theta)$ is returned as `corfac`, while the real and imaginary parts of the endpoint correction are returned as `corre` and `corim`.
```
{
    void nrerror(char error_text[]);
    float a0i,a0r,a1i,a1r,a2i,a2r,a3i,a3r,arg,c,cl,cr,s,sl,sr,t;
    float t2,t4,t6;
    double cth,ctth,spth2,sth,sth4i,stth,th,th2,th4,tmth2,tth4i;

    th=w*delta;
    if (a >= b || th < 0.0e0 || th > 3.01416e0) nrerror("bad arguments to dftcor");
    if (fabs(th) < 5.0e-2) {        Use series.
        t=th;
        t2=t*t;
        t4=t2*t2;
        t6=t4*t2;
```

```
		*corfac=1.0-(11.0/720.0)*t4+(23.0/15120.0)*t6;
		a0r=(-2.0/3.0)+t2/45.0+(103.0/15120.0)*t4-(169.0/226800.0)*t6;
		a1r=(7.0/24.0)-(7.0/180.0)*t2+(5.0/3456.0)*t4-(7.0/259200.0)*t6;
		a2r=(-1.0/6.0)+t2/45.0-(5.0/6048.0)*t4+t6/64800.0;
		a3r=(1.0/24.0)-t2/180.0+(5.0/24192.0)*t4-t6/259200.0;
		a0i=t*(2.0/45.0+(2.0/105.0)*t2-(8.0/2835.0)*t4+(86.0/467775.0)*t6);
		a1i=t*(7.0/72.0-t2/168.0+(11.0/72576.0)*t4-(13.0/5987520.0)*t6);
		a2i=t*(-7.0/90.0+t2/210.0-(11.0/90720.0)*t4+(13.0/7484400.0)*t6);
		a3i=t*(7.0/360.0-t2/840.0+(11.0/362880.0)*t4-(13.0/29937600.0)*t6);
	} else {				Use trigonometric formulas in double precision.
		cth=cos(th);
		sth=sin(th);
		ctth=cth*cth-sth*sth;
		stth=2.0e0*sth*cth;
		th2=th*th;
		th4=th2*th2;
		tmth2=3.0e0-th2;
		spth2=6.0e0+th2;
		sth4i=1.0/(6.0e0*th4);
		tth4i=2.0e0*sth4i;
		*corfac=tth4i*spth2*(3.0e0-4.0e0*cth+ctth);
		a0r=sth4i*(-42.0e0+5.0e0*th2+spth2*(8.0e0*cth-ctth));
		a0i=sth4i*(th*(-12.0e0+6.0e0*th2)+spth2*stth);
		a1r=sth4i*(14.0e0*tmth2-7.0e0*spth2*cth);
		a1i=sth4i*(30.0e0*th-5.0e0*spth2*sth);
		a2r=tth4i*(-4.0e0*tmth2+2.0e0*spth2*cth);
		a2i=tth4i*(-12.0e0*th+2.0e0*spth2*sth);
		a3r=sth4i*(2.0e0*tmth2-spth2*cth);
		a3i=sth4i*(6.0e0*th-spth2*sth);
	}
	cl=a0r*endpts[1]+a1r*endpts[2]+a2r*endpts[3]+a3r*endpts[4];
	sl=a0i*endpts[1]+a1i*endpts[2]+a2i*endpts[3]+a3i*endpts[4];
	cr=a0r*endpts[8]+a1r*endpts[7]+a2r*endpts[6]+a3r*endpts[5];
	sr = -a0i*endpts[8]-a1i*endpts[7]-a2i*endpts[6]-a3i*endpts[5];
	arg=w*(b-a);
	c=cos(arg);
	s=sin(arg);
	*corre=cl+c*cr-s*sr;
	*corim=sl+s*cr+c*sr;
}
```

Since the use of dftcor can be confusing, we also give an illustrative program dftint which uses dftcor to compute equation (13.9.1) for general $a$, $b$, $\omega$, and $h(t)$. Several points within this program bear mentioning: The parameters M and NDFT correspond to $M$ and $N$ in the above discussion. On successive calls, we recompute the Fourier transform only if $a$ or $b$ or $h(t)$ has changed.

Since dftint is designed to work for any value of $\omega$ satisfying $\omega\Delta < \pi$, not just the special values returned by the DFT (equation 13.9.12), we do polynomial interpolation of degree MPOL on the DFT spectrum. You should be warned that a large factor of oversampling ($N \gg M$) is required for this interpolation to be accurate. After interpolation, we add the endpoint corrections from dftcor, which can be evaluated for any $\omega$.

While dftcor is good at what it does, dftint is illustrative only. It is not a general purpose program, because it does not adapt its parameters M, NDFT, MPOL, or its interpolation scheme, to any particular function $h(t)$. You will have to experiment with your own application.

```
#include <math.h>
#include "nrutil.h"
#define M 64
#define NDFT 1024
#define MPOL 6
#define TWOPI (2.0*3.14159265)
```

The values of M, NDFT, and MPOL are merely illustrative and should be optimized for your particular application. M is the number of subintervals, NDFT is the length of the FFT (a power of 2), and MPOL is the degree of polynomial interpolation used to obtain the desired frequency from the FFT.

```
void dftint(float (*func)(float), float a, float b, float w, float *cosint,
    float *sinint)
```
Example program illustrating how to use the routine dftcor. The user supplies an external function func that returns the quantity $h(t)$. The routine then returns $\int_a^b \cos(\omega t)h(t)\,dt$ as cosint and $\int_a^b \sin(\omega t)h(t)\,dt$ as sinint.
```
{
    void dftcor(float w, float delta, float a, float b, float endpts[],
        float *corre, float *corim, float *corfac);
    void polint(float xa[], float ya[], int n, float x, float *y, float *dy);
    void realft(float data[], unsigned long n, int isign);
    static int init=0;
    int j,nn;
    static float aold = -1.e30,bold = -1.e30,delta,(*funcold)(float);
    static float data[NDFT+1],endpts[9];
    float c,cdft,cerr,corfac,corim,corre,en,s;
    float sdft,serr,*cpol,*spol,*xpol;

    cpol=vector(1,MPOL);
    spol=vector(1,MPOL);
    xpol=vector(1,MPOL);
    if (init != 1 || a != aold || b != bold || func != funcold) {
        Do we need to initialize, or is only ω changed?
        init=1;
        aold=a;
        bold=b;
        funcold=func;
        delta=(b-a)/M;
        Load the function values into the data array.
        for (j=1;j<=M+1;j++)
            data[j]=(*func)(a+(j-1)*delta);
        for (j=M+2;j<=NDFT;j++)          Zero pad the rest of the data array.
            data[j]=0.0;
        for (j=1;j<=4;j++) {                       Load the endpoints.
            endpts[j]=data[j];
            endpts[j+4]=data[M-3+j];
        }
        realft(data,NDFT,1);
        realft returns the unused value corresponding to ω_{N/2} in data[2]. We actually want
        this element to contain the imaginary part corresponding to ω_0, which is zero.
        data[2]=0.0;
    }
    Now interpolate on the DFT result for the desired frequency. If the frequency is an ω_n,
    i.e., the quantity en is an integer, then cdft=data[2*en-1], sdft=data[2*en], and you
    could omit the interpolation.
    en=w*delta*NDFT/TWOPI+1.0;
    nn=IMIN(IMAX((int)(en-0.5*MPOL+1.0),1),NDFT/2-MPOL+1);  Leftmost point for the
    for (j=1;j<=MPOL;j++,nn++) {                                 interpolation.
        cpol[j]=data[2*nn-1];
        spol[j]=data[2*nn];
        xpol[j]=nn;
    }
    polint(xpol,cpol,MPOL,en,&cdft,&cerr);
    polint(xpol,spol,MPOL,en,&sdft,&serr);
    dftcor(w,delta,a,b,endpts,&corre,&corim,&corfac);       Now get the endpoint cor-
    cdft *= corfac;                                              rection and the mul-
    sdft *= corfac;                                              tiplicative factor W(θ).
    cdft += corre;
    sdft += corim;
```

```
    c=delta*cos(w*a);                         Finally multiply by Δ and exp(iωa).
    s=delta*sin(w*a);
    *cosint=c*cdft-s*sdft;
    *sinint=s*cdft+c*sdft;
    free_vector(cpol,1,MPOL);
    free_vector(spol,1,MPOL);
    free_vector(xpol,1,MPOL);
}
```

Sometimes one is interested only in the discrete frequencies $\omega_m$ of equation (13.9.5), the ones that have integral numbers of periods in the interval $[a, b]$. For smooth $h(t)$, the value of $I$ tends to be much smaller in magnitude at these $\omega$'s than at values in between, since the integral half-periods tend to cancel precisely. (That is why one must oversample for interpolation to be accurate: $I(\omega)$ is oscillatory with small magnitude near the $\omega_m$'s.) If you want these $\omega_m$'s without messy (and possibly inaccurate) interpolation, you have to set $N$ to a multiple of $M$ (compare equations 13.9.5 and 13.9.12). In the method implemented above, however, $N$ must be at least $M + 1$, so the smallest such multiple is $2M$, resulting in a factor $\sim$2 unnecessary computing. Alternatively, one can derive a formula like equation (13.9.13), but with the last sample function $h_M = h(b)$ omitted from the DFT, but included entirely in the endpoint correction for $h_M$. Then one can set $M = N$ (an integer power of 2) and get the special frequencies of equation (13.9.5) with no additional overhead. The modified formula is

$$\begin{aligned}I(\omega_m) = \Delta e^{i\omega_m a}\Big\{&W(\theta)[\mathrm{DFT}(h_0 \dots h_{M-1})]_m \\ &+ \alpha_0(\theta)h_0 + \alpha_1(\theta)h_1 + \alpha_2(\theta)h_2 + \alpha_3(\theta)h_3 \\ &+ e^{i\omega(b-a)}\Big[A(\theta)h_M + \alpha_1^*(\theta)h_{M-1} + \alpha_2^*(\theta)h_{M-2} + \alpha_3^*(\theta)h_{M-3}\Big]\Big\}\end{aligned} \tag{13.9.14}$$

where $\theta \equiv \omega_m\Delta$ and $A(\theta)$ is given by

$$A(\theta) = -\alpha_0(\theta) \tag{13.9.15}$$

for the trapezoidal case, or

$$\begin{aligned}A(\theta) &= \frac{(-6 + 11\theta^2) + (6 + \theta^2)\cos 2\theta}{6\theta^4} - i\,\mathrm{Im}[\alpha_0(\theta)] \\ &\approx \frac{1}{3} + \frac{1}{45}\theta^2 - \frac{8}{945}\theta^4 + \frac{11}{14175}\theta^6 - i\,\mathrm{Im}[\alpha_0(\theta)]\end{aligned} \tag{13.9.16}$$

for the cubic case.

Factors like $W(\theta)$ arise naturally whenever one calculates Fourier coefficients of smooth functions, and they are sometimes called attenuation factors [1]. However, the endpoint corrections are equally important in obtaining accurate values of integrals. Narasimhan and Karthikeyan [2] have given a formula that is algebraically equivalent to our trapezoidal formula. However, their formula requires the evaluation of *two* FFTs, which is unnecessary. The basic idea used here goes back at least to Filon [3] in 1928 (before the FFT!). He used Simpson's rule (quadratic interpolation). Since this interpolation is not left-right symmetric, two Fourier transforms are required. An alternative algorithm for equation (13.9.14) has been given by Lyness in [4]; for related references, see [5]. To our knowledge, the cubic-order formulas derived here have not previously appeared in the literature.

Calculating Fourier transforms when the range of integration is $(-\infty, \infty)$ can be tricky. If the function falls off reasonably quickly at infinity, you can split the integral at a large enough value of $t$. For example, the integration to $+\infty$ can be written

$$\begin{aligned}\int_a^\infty e^{i\omega t}h(t)\,dt &= \int_a^b e^{i\omega t}h(t)\,dt + \int_b^\infty e^{i\omega t}h(t)\,dt \\ &= \int_a^b e^{i\omega t}h(t)\,dt - \frac{h(b)e^{i\omega b}}{i\omega} + \frac{h'(b)e^{i\omega b}}{(i\omega)^2} - \cdots\end{aligned} \tag{13.9.17}$$

The splitting point $b$ must be chosen large enough that the remaining integral over $(b, \infty)$ is small. Successive terms in its asymptotic expansion are found by integrating by parts. The integral over $(a, b)$ can be done using `dftint`. You keep as many terms in the asymptotic expansion as you can easily compute. See [6] for some examples of this idea. More powerful methods, which work well for long-tailed functions but which do not use the FFT, are described in [7-9].

CITED REFERENCES AND FURTHER READING:

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), p. 88. [1]

Narasimhan, M.S. and Karthikeyan, M. 1984, *IEEE Transactions on Antennas & Propagation*, vol. 32, pp. 404–408. [2]

Filon, L.N.G. 1928, *Proceedings of the Royal Society of Edinburgh*, vol. 49, pp. 38–47. [3]

Giunta, G. and Murli, A. 1987, *ACM Transactions on Mathematical Software*, vol. 13, pp. 97–107. [4]

Lyness, J.N. 1987, in *Numerical Integration*, P. Keast and G. Fairweather, eds. (Dordrecht: Reidel). [5]

Pantis, G. 1975, *Journal of Computational Physics*, vol. 17, pp. 229–233. [6]

Blakemore, M., Evans, G.A., and Hyslop, J. 1976, *Journal of Computational Physics*, vol. 22, pp. 352–376. [7]

Lyness, J.N., and Kaper, T.J. 1987, *SIAM Journal on Scientific and Statistical Computing*, vol. 8, pp. 1005–1011. [8]

Thakkar, A.J., and Smith, V.H. 1975, *Computer Physics Communications*, vol. 10, pp. 73–79. [9]

# 13.10 Wavelet Transforms

Like the fast Fourier transform (FFT), the discrete wavelet transform (DWT) is a fast, linear operation that operates on a data vector whose length is an integer power of two, transforming it into a numerically different vector of the same length. Also like the FFT, the wavelet transform is invertible and in fact orthogonal — the inverse transform, when viewed as a big matrix, is simply the transpose of the transform. Both FFT and DWT, therefore, can be viewed as a rotation in function space, from the input space (or time) domain, where the basis functions are the unit vectors $\mathbf{e}_i$, or Dirac delta functions in the continuum limit, to a different domain. For the FFT, this new domain has basis functions that are the familiar sines and cosines. In the wavelet domain, the basis functions are somewhat more complicated and have the fanciful names "mother functions" and "wavelets."

Of course there are an infinity of possible bases for function space, almost all of them uninteresting! What makes the wavelet basis interesting is that, *unlike* sines and cosines, individual wavelet functions are quite localized in space; simultaneously, *like* sines and cosines, individual wavelet functions are quite localized in frequency or (more precisely) characteristic scale. As we will see below, the particular kind of dual localization achieved by wavelets renders large classes of functions and operators sparse, or sparse to some high accuracy, when transformed into the wavelet domain. Analogously with the Fourier domain, where a class of computations, like convolutions, become computationally fast, there is a large class of computations