```
              data(k1)=data(k1)+tempr
              data(k1+1)=data(k1+1)+tempi
          enddo 15
        enddo 16
        wtemp=wr              Trigonometric recurrence.
        wr=wr*wpr-wi*wpi+wr
        wi=wi*wpr+wtemp*wpi+wi
      enddo 17
      ifp1=ifp2
    goto 2
    endif
    nprev=n*nprev
enddo 18
return
END
```

CITED REFERENCES AND FURTHER READING:

Nussbaumer, H.J. 1982, *Fast Fourier Transform and Convolution Algorithms* (New York: Springer-Verlag).

12.5 Fourier Transforms of Real Data in Two and Three Dimensions

Two-dimensional FFTs are particularly important in the field of image processing. An image is usually represented as a two-dimensional array of pixel intensities, real (and usually positive) numbers. One commonly desires to filter high, or low, frequency spatial components from an image; or to convolve or deconvolve the image with some instrumental point spread function. Use of the FFT is by far the most efficient technique.

In three dimensions, a common use of the FFT is to solve Poisson's equation for a potential (e.g., electromagnetic or gravitational) on a three-dimensional lattice that represents the discretization of three-dimensional space. Here the source terms (mass or charge distribution) and the desired potentials are also real. In two and three dimensions, with large arrays, memory is often at a premium. It is therefore important to perform the FFTs, insofar as possible, on the data "in place." We want a routine with functionality similar to the multidimensional FFT routine fourn (§12.4), but which operates on real, not complex, input data. We give such a routine in this section. The development is analogous to that of §12.3 leading to the one-dimensional routine realft. (You might wish to review that material at this point, particularly equation 12.3.5.)

It is convenient to think of the independent variables $n_1, \ldots, n_L$ in equation (12.4.3) as representing an L-dimensional vector $\vec{n}$ in wave-number space, with values on the lattice of integers. The transform $H(n_1, \ldots, n_L)$ is then denoted $H(\vec{n})$.

It is easy to see that the transform $H(\vec{n})$ is periodic in each of its L dimensions. Specifically, if $\vec{P}_1, \vec{P}_2, \vec{P}_3, \ldots$ denote the vectors $(N_1, 0, 0, \ldots)$, $(0, N_2, 0, \ldots)$, $(0, 0, N_3, \ldots)$, and so forth, then

$$H(\vec{n} \pm \vec{P}_j) = H(\vec{n}) \qquad j = 1, \ldots, L \qquad (12.5.1)$$

Equation (12.5.1) holds for any input data, real or complex. When the data is real, we have the additional symmetry

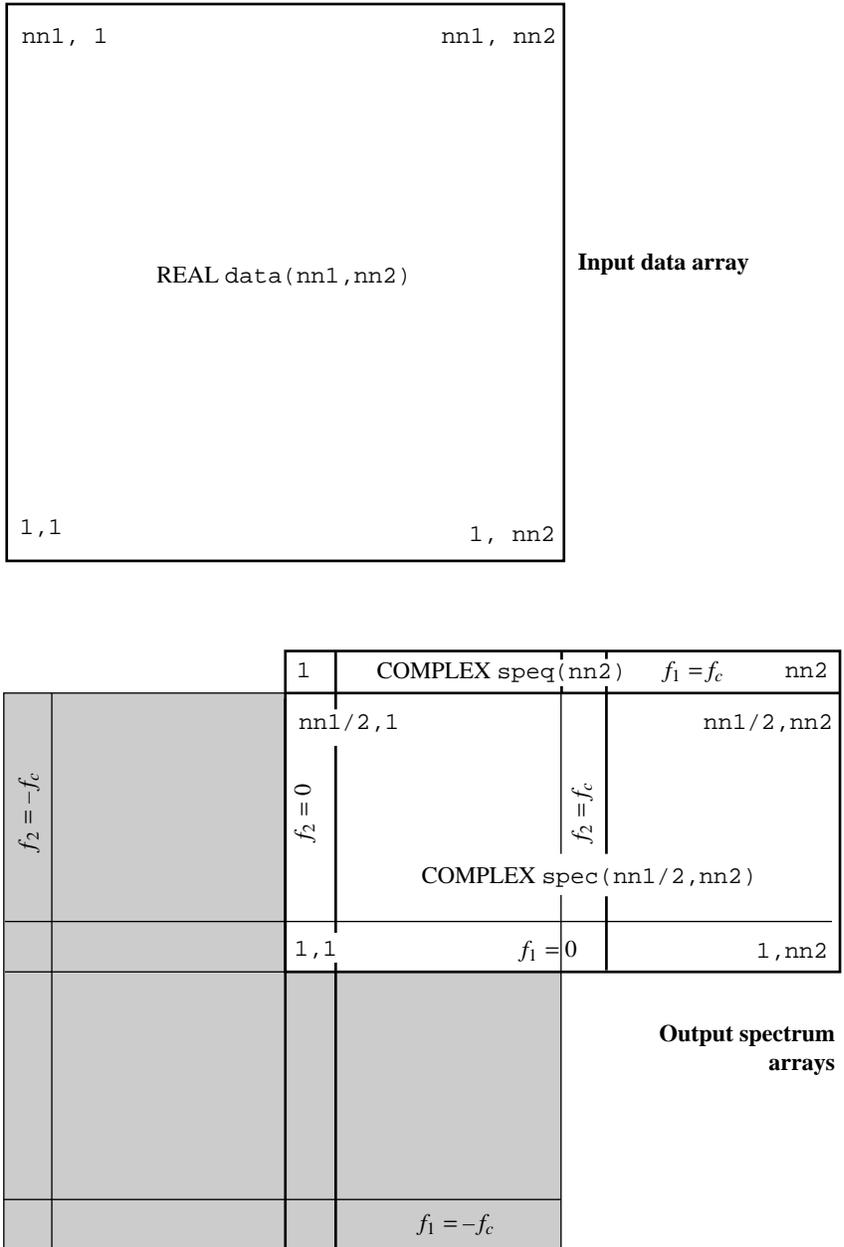$$H(-\vec{n}) = H(\vec{n})^* \tag{12.5.2}$$

Equations (12.5.1) and (12.5.2) imply that the full transform can be trivially obtained from the subset of lattice values $\vec{n}$ that have

$$\begin{aligned} 0 \leq n_1 &\leq \frac{N_1}{2} \\ 0 \leq n_2 &\leq N_2 - 1 \\ &\cdots \\ 0 \leq n_L &\leq N_L - 1 \end{aligned} \tag{12.5.3}$$

In fact, this set of values is overcomplete, because there are additional symmetry relations among the transform values that have $n_1 = 0$ and $n_1 = N_1/2$. However these symmetries are complicated and their use becomes extremely confusing. Therefore, we will compute our FFT on the lattice subset of equation (12.5.3), even though this requires a small amount of extra storage for the answer, i.e., the transform is not *quite* "in place." (Although an in-place transform is in fact possible, we have found it virtually impossible to explain to any user how to unscramble its output, i.e., where to find the real and imaginary components of the transform at some particular frequency!)

Figure 12.5.1 shows the storage scheme that we will use for the input data and the output transform. The figure is specialized to the case of two dimensions, $L = 2$, but the generalization to higher dimensions is obvious. The input data is a two-dimensional real array of dimensions N_1 (called nn1) by N_2 (called nn2). Notice that the FORTRAN subscripts number from 1 to nn1, and not from 0 to $N_1 - 1$. The output spectrum is in two complex arrays, one two-dimensional and the other one-dimensional. The two-dimensional one, spec, has dimensions nn1/2 by nn2. This is exactly half the size of the input data array; but since it is complex, it is the same amount of storage. In fact, spec will share storage with (and overwrite) the input data array. As the figure shows, spec contains those spectral components whose first component of frequency, f_1, ranges from zero to just short of the Nyquist frequency f_c. The full range of positive and negative second-component of frequencies, f_2, is stored, in wrap-around order (see §12.2), with negative frequencies shifted by exactly one period to put them "above" the positive frequencies, as the figure indicates. The figure also indicates how the additional $L - 1$ (here, one-) dimensional array speq stores only that single value of n_1 that corresponds to the Nyquist frequency, but all values of n_2, etc.

With this much introduction, the implementing procedure, called rlft3, is something of an anticlimax. The routine is written for the case of $L = 3$ dimensions, but (we will explain below) it can be used without modification for $L = 2$ also; and it is quite trivial to generalize it to larger L. Look at the innermost ("do $_{13}$") loop in the procedure, and you will see equation (12.3.5) implemented on the *first* transform index. The case of i1=1 is coded separately, to account for the fact that speq is to be filled instead of spec (which is here called data since it shares storage with

Figure 12.5.1. Input and output data arrangement for `rlft3` in the case of two-dimensional data. The input data array is a real, two-dimensional array. The output data array `spec` is a complex, two-dimensional array whose $(1, 1)$ element contains the $f_1 = f_2 = 0$ spectral component; a complete set of f_2 values are stored in wrap-around order, while only positive f_1 values are stored (others being obtainable by symmetry). The output array `speq` contains components with f_1 equal to the Nyquist frequency.

the input array). The three enclosing do loops (indices `i2`, `i1`, and `i3`, from inside to outside) could in fact be done in any order — their actions all commute. We chose the order shown because of the following considerations: (i) `i1` should not be the inner loop, because if it is, then the recurrence relations on `wr` and `wi` become burdensome. (ii) On virtual-memory machines, `i3` should be the outer loop, because (with FORTRAN order of array storage) this results in the array `data`, which might be very large, being accessed in block sequential order.

Note that the work done in `rlft3` is quite (logarithmically) small, compared to the associated complex FFT, `fourn`. For this reason, we allow ourselves the clarity of using FORTRAN complex arithmetic even when (as in the multiplications by `c1` and `c2`) there are a few unnecessary operations. The routine `rlft3` is based on an earlier routine by G.B. Rybicki.

```
      SUBROUTINE rlft3(data,speq,nn1,nn2,nn3,isign)
      INTEGER isign,nn1,nn2,nn3
      COMPLEX data(nn1/2,nn2,nn3),speq(nn2,nn3)
C     USES fourn
          Given a two- or three-dimensional real array data whose dimensions are nn1, nn2, nn3
          (where nn3 is 1 for the case of a two-dimensional array), this routine returns (for isign=1)
          the complex fast Fourier transform as two complex arrays: On output, data contains the
          zero and positive frequency values of the first frequency component, while speq contains
          the Nyquist critical frequency values of the first frequency component. Second (and third)
          frequency components are stored for zero, positive, and negative frequencies, in standard
          wrap-around order. For isign=-1, the inverse transform (times nn1*nn2*nn3/2 as a
          constant multiplicative factor) is performed, with output data (viewed as a real array)
          deriving from input data (viewed as complex) and speq. For inverse transforms on data
          not generated first by a forward transform, make sure the complex input data array satisfies
          property (12.5.2). The dimensions nn1, nn2, nn3 must always be integer powers of 2.
      INTEGER i1,i2,i3,j1,j2,j3,nn(3)
      DOUBLE PRECISION theta,wi,wpi,wpr,wr,wtemp
      COMPLEX c1,c2,h1,h2,w              Note that data is dimensioned as complex, its output
      c1=cmplx(0.5,0.0)                      format.
      c2=cmplx(0.0,-0.5*isign)
      theta=6.28318530717959d0/dble(isign*nn1)
      wpr=-2.0d0*sin(0.5d0*theta)**2
      wpi=sin(theta)
      nn(1)=nn1/2
      nn(2)=nn2
      nn(3)=nn3
      if(isign.eq.1)then                 Case of forward transform.
          call fourn(data,nn,3,isign)    Here is where most all of the compute time is spent.
          do 12 i3=1,nn3                 Extend data periodically into speq.
              do 11 i2=1,nn2
                  speq(i2,i3)=data(1,i2,i3)
              enddo 11
          enddo 12
      endif
      do 15 i3=1,nn3
          j3=1
          if (i3.ne.1) j3=nn3-i3+2       Zero frequency is its own reflection, otherwise locate cor-
          wr=1.0d0                           responding negative frequency in wrap-around order.
          wi=0.0d0                       Initialize trigonometric recurrence.
          do 14 i1=1,nn1/4+1
              j1=nn1/2-i1+2
              do 13 i2=1,nn2
                  j2=1
                  if (i2.ne.1) j2=nn2-i2+2
                  if(i1.eq.1)then        Equation (12.3.5).
                      h1=c1*(data(1,i2,i3)+conjg(speq(j2,j3)))
                      h2=c2*(data(1,i2,i3)-conjg(speq(j2,j3)))
```

Figure 12.5.2. (a) A two-dimensional image with intensities either purely black or purely white. (b) The same image, after it has been low-pass filtered using `rlft3`. Regions with fine-scale features become gray.

```
              data(1,i2,i3)=h1+h2
              speq(j2,j3)=conjg(h1-h2)
          else
              h1=c1*(data(i1,i2,i3)+conjg(data(j1,j2,j3)))
              h2=c2*(data(i1,i2,i3)-conjg(data(j1,j2,j3)))
              data(i1,i2,i3)=h1+w*h2
              data(j1,j2,j3)=conjg(h1-w*h2)
          endif
      enddo 13
      wtemp=wr                      Do the recurrence.
      wr=wr*wpr-wi*wpi+wr
      wi=wi*wpr+wtemp*wpi+wi
      w=cmplx(sngl(wr),sngl(wi))
    enddo 14
enddo 15
if(isign.eq.-1)then               Case of reverse transform.
    call fourn(data,nn,3,isign)
endif
return
END
```

We now give some fragments from notional calling programs, to clarify the use of `rlft3` for two- and three-dimensional data. Note that the routine does not actually distinguish between two and three dimensions; two is treated like three, but with the third dimension having length 1. Since the third dimension is the outer loop, almost no inefficiency is introduced.

The first program fragment FFTs a two-dimensional data array, allows for some processing on it, e.g., filtering, and then takes the inverse transform. Figure 12.5.2 shows an example of the use of this kind of code: A sharp image becomes blurry when its high-frequency spatial components are suppressed by the factor (here) $\max\left(1 - 6f^2/f_c^2, 0\right)$. The second program example illustrates a three-dimensional transform, where the three dimensions have different lengths. The third program example is an example of convolution, as it might occur in a program to compute the potential generated by a three-dimensional distribution of sources.

```
      PROGRAM exmpl1
         This fragment shows how one might filter a 256 by 256 digital image.
      INTEGER N1,N2,N3
      PARAMETER (N1=256,N2=256,N3=1)  Note that the third component must be set to 1.
C     USES rlft3
      REAL data(N1,N2)
      COMPLEX spec(N1/2,N2),speq(N2)
      EQUIVALENCE (data,spec)
C     ...                          Here the image would be loaded into data.
      call rlft3(data,speq,N1,N2,N3,1)
C     ...                          Here the arrays spec and speq would be multiplied by a suit-
      call rlft3(data,speq,N1,N2,N3,-1)      able filter function (of frequency).
C     ...                          Here the filtered image would be unloaded from data.
      END


      PROGRAM exmpl2
         This fragment shows how one might FFT a real three-dimensional array of size 32 by 64
         by 16.
      INTEGER N1,N2,N3
      PARAMETER (N1=32,N2=64,N3=16)
C     USES rlft3
      REAL data(N1,N2,N3)
      COMPLEX spec(N1/2,N2,N3),speq(N2,N3)
      EQUIVALENCE (data,spec)
C     ...                          Here load data.
      call rlft3(data,speq,N1,N2,N3,1)
C     ...                          Here unload spec and speq.
      END


      PROGRAM exmpl3
         This fragment shows how one might convolve two real, three-dimensional arrays of size 32
         by 32 by 32, replacing the first array by the result.
      INTEGER N
      PARAMETER (N=32)
C     USES rlft3
      INTEGER j
      REAL fac,data1(N,N,N),data2(N,N,N)
      COMPLEX spec1(N/2,N,N),speq1(N,N),spec2(N/2,N,N),speq2(N,N),
     *    zpec1(N*N*N/2),zpeq1(N*N),zpec2(N*N*N/2),zpeq2(N*N)
      EQUIVALENCE (data1,spec1,zpec1), (data2,spec2,zpec2),
     *    (speq1,zpeq1), (speq2,zpeq2)
C     ...
      call rlft3(data1,speq1,N,N,N,1)          FFT both input arrays.
      call rlft3(data2,speq2,N,N,N,1)
      fac=2./(N*N*N)                           Factor needed to get normalized inverse.
      do 11 j=1,N*N*N/2                         The sole purpose of the zpecs and zpeqs is to make
         zpec1(j)=fac*zpec1(j)*zpec2(j)         this a single do-loop instead of three-nested ones.
      enddo 11
      do 12 j=1,N*N
         zpeq1(j)=fac*zpeq1(j)*zpeq2(j)
      enddo 12
      call rlft3(data1,speq1,N,N,N,-1)   Inverse FFT the product of the two FFTs.
C     ...
      END
```

To extend `rlft3` to four dimensions, you simply add an additional (outer) nested do loop in `i4`, analogous to the present `i3`. (Modifying the routine to do an *arbitrary* number of dimensions, as in `fourn`, is a good programming exercise for the reader.)

CITED REFERENCES AND FURTHER READING:

Brigham, E.O. 1974, *The Fast Fourier Transform* (Englewood Cliffs, NJ: Prentice-Hall).

Swartztrauber, P. N. 1986, *Mathematics of Computation*, vol. 47, pp. 323–346.

12.6 External Storage or Memory-Local FFTs

Sometime in your life, you might have to compute the Fourier transform of a *really large* data set, larger than the size of your computer's physical memory. In such a case, the data will be stored on some external medium, such as magnetic or optical tape or disk. Needed is an algorithm that makes some manageable number of sequential passes through the external data, processing it on the fly and outputting intermediate results to other external media, which can be read on subsequent passes.

In fact, an algorithm of just this description was developed by Singleton [1] very soon after the discovery of the FFT. The algorithm requires four sequential storage devices, each capable of holding half of the input data. The first half of the input data is initially on one device, the second half on another.

Singleton's algorithm is based on the observation that it is possible to bit-reverse 2^M values by the following sequence of operations: On the first pass, values are read alternately from the two input devices, and written to a single output device (until it holds half the data), and then to the other output device. On the second pass, the output devices become input devices, and vice versa. Now, we copy *two* values from the first device, then *two* values from the second, writing them (as before) first to fill one output device, then to fill a second. Subsequent passes read 4, 8, etc., input values at a time. After completion of pass $M - 1$, the data are in bit-reverse order.

Singleton's next observation is that it is possible to alternate the passes of essentially this bit-reversal technique with passes that implement one stage of the Danielson-Lanczos combination formula (12.2.3). The scheme, roughly, is this: One starts as before with half the input data on one device, half on another. In the first pass, one complex value is read from each input device. Two combinations are formed, and one is written to each of two output devices. After this "computing" pass, the devices are rewound, and a "permutation" pass is performed, where groups of values are read from the first input device and alternately written to the first and second output devices; when the first input device is exhausted, the second is similarly processed. This sequence of computing and permutation passes is repeated $M - K - 1$ times, where 2^K is the size of internal buffer available to the program. The second phase of the computation consists of a final K computation passes. What distinguishes the second phase from the first is that, now, the permutations are local enough to do in place during the computation. There are thus no separate permutation passes in the second phase. In all, there are $2M - K - 2$ passes through the data.

Here is an implementation of Singleton's algorithm, based on [1]:

```
      SUBROUTINE fourfs(iunit,nn,ndim,isign)
      INTEGER ndim,nn(ndim),isign,iunit(4),KBF
      PARAMETER (KBF=128)
C     USES fourew
          One- or multi-dimensional Fourier transform of a large data set stored on external media.
          On input, ndim is the number of dimensions, and nn(1:ndim) contains the lengths of
          each dimension (number of complex values), which must be powers of two. iunit(1:4)
          contains the unit numbers of 4 sequential files, each large enough to hold half of the data.
          The four units must be opened for FORTRAN unformatted access. The input data must be
          in FORTRAN normal order, with its first half stored on unit iunit(1), its second half on
          iunit(2), in unformatted form, with KBF real numbers per record. isign should be set
          to 1 for the Fourier transform, to −1 for its inverse. On output, values in the array iunit
          may have been permuted; the first half of the result is stored on iunit(3), the second
          half on iunit(4). N.B.: For ndim > 1, the output is stored by rows, i.e., *not* in FORTRAN
          normal order; in other words, the output is the transpose of that which would have been
          produced by routine fourn.
      INTEGER j,j12,jk,k,kk,n,mm,kc,kd,ks,kr,nr,ns,nv,jx,
     *    mate(4),na,nb,nc,nd
      REAL tempr,tempi,afa(KBF),afb(KBF),afc(KBF)
      DOUBLE PRECISION wr,wi,wpr,wpi,wtemp,theta
      SAVE mate
      DATA mate /2,1,4,3/
```