13.2 Correlation and Autocorrelation Using the FFT

Correlation is the close mathematical cousin of convolution. It is in some ways simpler, however, because the two functions that go into a correlation are not as conceptually distinct as were the data and response functions that entered into convolution. Rather, in correlation, the functions are represented by different, but generally similar, data sets. We investigate their "correlation," by comparing them both directly superposed, and with one of them shifted left or right.

We have already defined in equation (12.0.10) the correlation between two continuous functions $g(t)$ and $h(t)$, which is denoted $\text{Corr}(g, h)$, and is a function of *lag* t. We will occasionally show this time dependence explicitly, with the rather awkward notation $\text{Corr}(g, h)(t)$. The correlation will be large at some value of t if the first function (g) is a close copy of the second (h) but lags it in time by t, i.e., if the first function is shifted to the right of the second. Likewise, the correlation will be large for some negative value of t if the first function *leads* the second, i.e., is shifted to the left of the second. The relation that holds when the two functions are interchanged is

$$\text{Corr}(g, h)(t) = \text{Corr}(h, g)(-t) \tag{13.2.1}$$

The discrete correlation of two sampled functions g_k and h_k, each periodic with period N, is defined by

$$\text{Corr}(g, h)_j \equiv \sum_{k=0}^{N-1} g_{j+k} h_k \tag{13.2.2}$$

The *discrete correlation theorem* says that this discrete correlation of two real functions g and h is one member of the discrete Fourier transform pair

$$\text{Corr}(g, h)_j \Longleftrightarrow G_k H_k{}^* \tag{13.2.3}$$

where G_k and H_k are the discrete Fourier transforms of g_j and h_j, and the asterisk denotes complex conjugation. This theorem makes the same presumptions about the functions as those encountered for the discrete convolution theorem.

We can compute correlations using the FFT as follows: FFT the two data sets, multiply one resulting transform by the complex conjugate of the other, and inverse transform the product. The result (call it r_k) will formally be a complex vector of length N. However, it will turn out to have all its imaginary parts zero since the original data sets were both real. The components of r_k are the values of the correlation at different lags, with positive and negative lags stored in the by now familiar wrap-around order: The correlation at zero lag is in r_0, the first component; the correlation at lag 1 is in r_1, the second component; the correlation at lag -1 is in r_{N-1}, the last component; etc.

Just as in the case of convolution we have to consider end effects, since our data will not, in general, be periodic as intended by the correlation theorem. Here again, we can use zero padding. If you are interested in the correlation for lags as large as $\pm K$, then you must append a buffer zone of K zeros at the end of both input data sets. If you want all possible lags from N data points (not a usual thing), then you will need to pad the data with an equal number of zeros; this is the extreme case. So here is the program:

```
      SUBROUTINE correl(data1,data2,n,ans)
      INTEGER n,NMAX
      REAL data1(n),data2(n)
      COMPLEX ans(n)
      PARAMETER (NMAX=4096)           Maximum anticipated FFT size.
C     USES realft,twofft
        Computes the correlation of two real data sets data1(1:n) and data2(1:n) (includ-
        ing any user-supplied zero padding). n MUST be an integer power of two. The answer
        is returned as the first n points in ans stored in wrap-around order, i.e., correlations at
        increasingly negative lags are in ans(n) on down to ans(n/2+1), while correlations at
        increasingly positive lags are in ans(1) (zero lag) on up to ans(n/2). Note that ans
        must be supplied in the calling program with length at least 2*n, since it is also used as
        working space. Sign convention of this routine: if data1 lags data2, i.e., is shifted to the
        right of it, then ans will show a peak at positive lags.
      INTEGER i,no2
      COMPLEX fft(NMAX)
      call twofft(data1,data2,fft,ans,n)    Transform both data vectors at once.
      no2=n/2                                Normalization for inverse FFT.
      do 11 i=1,no2+1
        ans(i)=fft(i)*conjg(ans(i))/float(no2)     Multiply to find FFT of their corre-
      enddo 11                                          lation.
      ans(1)=cmplx(real(ans(1)),real(ans(no2+1)))   Pack first and last into one element.
      call realft(ans,n,-1)                  Inverse transform gives correlation.
      return
      END
```

As in convlv, it would be better to substitute two calls to realft for the one call to twofft, if data1 and data2 have very different magnitudes, to minimize roundoff error.

The *discrete autocorrelation* of a sampled function g_j is just the discrete correlation of the function with itself. Obviously this is always symmetric with respect to positive and negative lags. Feel free to use the above routine correl to obtain autocorrelations, simply calling it with the same data vector in both arguments. If the inefficiency bothers you, routine realft can, of course, be used to transform the data vector instead.

CITED REFERENCES AND FURTHER READING:

Brigham, E.O. 1974, *The Fast Fourier Transform* (Englewood Cliffs, NJ: Prentice-Hall), §13–2.

13.3 Optimal (Wiener) Filtering with the FFT

There are a number of other tasks in numerical processing that are routinely handled with Fourier techniques. One of these is filtering for the removal of noise from a "corrupted" signal. The particular situation we consider is this: There is some underlying, uncorrupted signal $u(t)$ that we want to measure. The measurement process is imperfect, however, and what comes out of our measurement device is a corrupted signal $c(t)$. The signal $c(t)$ may be less than perfect in either or both of two respects. First, the apparatus may not have a perfect "delta-function" response,