13.1 Convolution and Deconvolution Using the FFT

We have defined the *convolution* of two functions for the continuous case in equation (12.0.8), and have given the *convolution theorem* as equation (12.0.9). The theorem says that the Fourier transform of the convolution of two functions is equal to the product of their individual Fourier transforms. Now, we want to deal with the discrete case. We will mention first the context in which convolution is a useful procedure, and then discuss how to compute it efficiently using the FFT.

The convolution of two functions $r(t)$ and $s(t)$, denoted $r * s$, is mathematically equal to their convolution in the opposite order, $s * r$. Nevertheless, in most applications the two functions have quite different meanings and characters. One of the functions, say s, is typically a signal or data stream, which goes on indefinitely in time (or in whatever the appropriate independent variable may be). The other function r is a "response function," typically a peaked function that falls to zero in both directions from its maximum. The effect of convolution is to smear the signal $s(t)$ in time according to the recipe provided by the response function $r(t)$, as shown in Figure 13.1.1. In particular, a spike or delta-function of unit area in s which occurs at some time t_0 is supposed to be smeared into the shape of the response function itself, but translated from time 0 to time t_0 as $r(t - t_0)$.

In the discrete case, the signal $s(t)$ is represented by its sampled values at equal time intervals s_j. The response function is also a discrete set of numbers r_k, with the following interpretation: r_0 tells what multiple of the input signal in one channel (one particular value of j) is copied into the identical output channel (same value of j); r_1 tells what multiple of input signal in channel j is additionally copied into output channel $j + 1$; r_{-1} tells the multiple that is copied into channel $j - 1$; and so on for both positive and negative values of k in r_k. Figure 13.1.2 illustrates the situation.

Example: a response function with $r_0 = 1$ and all other r_k's equal to zero is just the identity filter: convolution of a signal with this response function gives identically the signal. Another example is the response function with $r_{14} = 1.5$ and all other r_k's equal to zero. This produces convolved output that is the input signal multiplied by 1.5 and delayed by 14 sample intervals.

Evidently, we have just described in words the following definition of discrete convolution with a response function of finite duration M:

$$(r * s)_j \equiv \sum_{k=-M/2+1}^{M/2} s_{j-k}\ r_k \tag{13.1.1}$$

If a discrete response function is nonzero only in some range $-M/2 < k \leq M/2$, where M is a sufficiently large even integer, then the response function is called a *finite impulse response (FIR)*, and its *duration* is M. (Notice that we are defining M as the number of nonzero *values* of r_k; these values span a time interval of $M - 1$ sampling times.) In most practical circumstances the case of finite M is the case of interest, either because the response really has a finite duration, or because we choose to truncate it at some point and approximate it by a finite-duration response function.

The *discrete convolution theorem* is this: If a signal s_j is *periodic* with period N, so that it is completely determined by the N values $s_0, \ldots, s_{N-1}$, then its

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

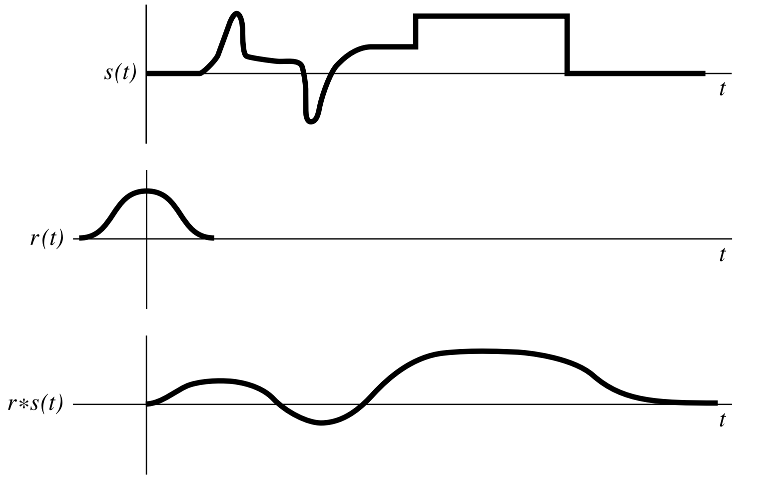

Figure 13.1.1. Example of the convolution of two functions. A signal $s(t)$ is convolved with a response function $r(t)$. Since the response function is broader than some features in the original signal, these are "washed out" in the convolution. In the absence of any additional noise, the process can be reversed by deconvolution.

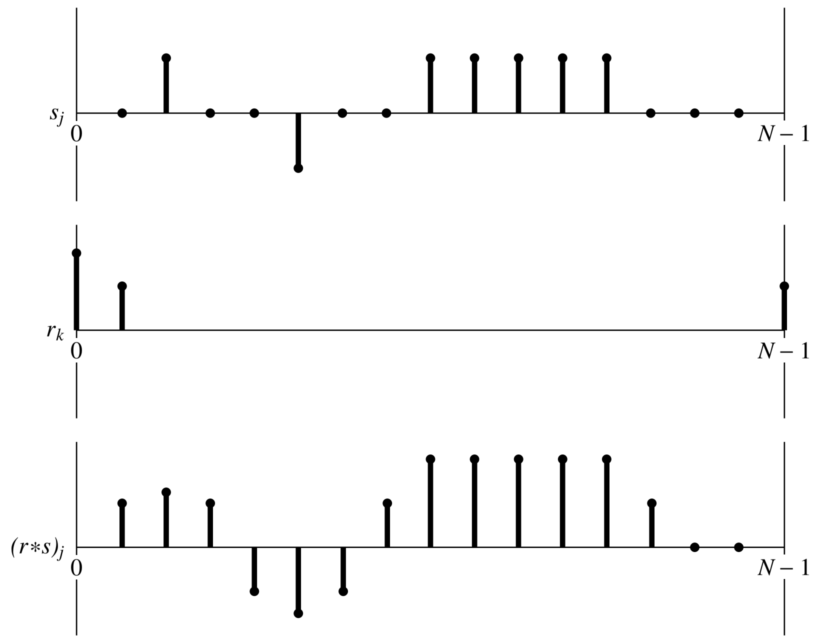

Figure 13.1.2. Convolution of discretely sampled functions. Note how the response function for negative times is wrapped around and stored at the extreme right end of the array r_k.

discrete convolution with a response function *of finite duration* N is a member of the discrete Fourier transform pair,

$$\sum_{k=-N/2+1}^{N/2} s_{j-k}\, r_k \quad \Longleftrightarrow \quad S_n R_n \tag{13.1.2}$$

Here $S_n,\ (n = 0, \ldots, N-1)$ is the discrete Fourier transform of the values $s_j,\ (j = 0, \ldots, N-1)$, while $R_n,\ (n = 0, \ldots, N-1)$ is the discrete Fourier transform of the values $r_k,\ (k = 0, \ldots, N-1)$. These values of r_k are the same ones as for the range $k = -N/2+1, \ldots, N/2$, but in wrap-around order, exactly as was described at the end of §12.2.

Treatment of End Effects by Zero Padding

The discrete convolution theorem presumes a set of two circumstances that are not universal. First, it assumes that the input signal is periodic, whereas real data often either go forever without repetition or else consist of one nonperiodic stretch of finite length. Second, the convolution theorem takes the duration of the response to be the same as the period of the data; they are both N. We need to work around these two constraints.

The second is very straightforward. Almost always, one is interested in a response function whose duration M is much shorter than the length of the data set N. In this case, you simply extend the response function to length N by padding it with zeros, i.e., define $r_k = 0$ for $M/2 \le k \le N/2$ and also for $-N/2+1 \le k \le -M/2+1$. Dealing with the first constraint is more challenging. Since the convolution theorem rashly assumes that the data are periodic, it will falsely "pollute" the first output channel $(r * s)_0$ with some wrapped-around data from the far end of the data stream s_{N-1}, s_{N-2}, etc. (See Figure 13.1.3.) So, we need to set up a buffer zone of zero-padded values at the end of the s_j vector, in order to make this pollution zero. How many zero values do we need in this buffer? Exactly as many as the most negative index for which the response function is nonzero. For example, if r_{-3} is nonzero, while $r_{-4}, r_{-5}, \ldots$ are all zero, then we need three zero pads at the end of the data: $s_{N-3} = s_{N-2} = s_{N-1} = 0$. These zeros will protect the first output channel $(r * s)_0$ from wrap-around pollution. It should be obvious that the second output channel $(r * s)_1$ and subsequent ones will also be protected by these same zeros. Let K denote the number of padding zeros, so that the last actual input data point is s_{N-K-1}.

What now about pollution of the very *last* output channel? Since the data now end with s_{N-K-1}, the last output channel of interest is $(r * s)_{N-K-1}$. This channel can be polluted by wrap-around from input channel s_0 unless the number K is also large enough to take care of the most positive index k for which the response function r_k is nonzero. For example, if r_0 through r_6 are nonzero, while $r_7, r_8 \ldots$ are all zero, then we need at least $K = 6$ padding zeros at the end of the data: $s_{N-6} = \ldots = s_{N-1} = 0$.

To summarize — we need to pad the data with a number of zeros *on one end* equal to the maximum positive duration *or* maximum negative duration of the response function, *whichever is larger*. (For a symmetric response function of duration M, you will need only $M/2$ zero pads.) Combining this operation with the

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

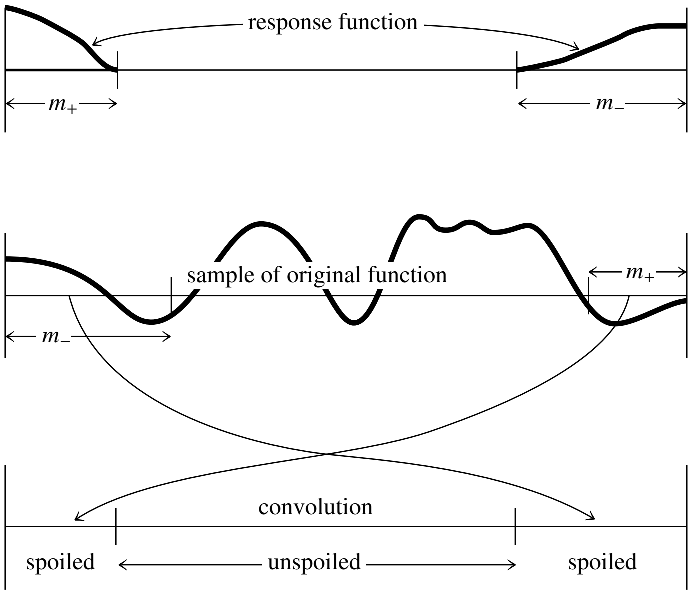

Figure 13.1.3. The wrap-around problem in convolving finite segments of a function. Not only must the response function wrap be viewed as cyclic, but so must the sampled original function. Therefore a portion at each end of the original function is erroneously wrapped around by convolution with the response function.

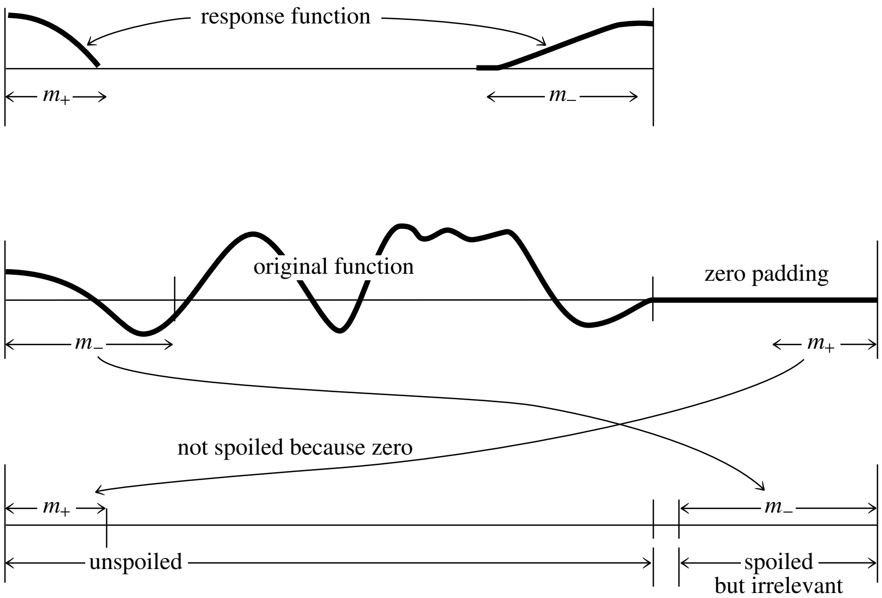

Figure 13.1.4. Zero padding as solution to the wrap-around problem. The original function is extended by zeros, serving a dual purpose: When the zeros wrap around, they do not disturb the true convolution; and while the original function wraps around onto the zero region, that region can be discarded.

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

padding of the response r_k described above, we effectively insulate the data from artifacts of undesired periodicity. Figure 13.1.4 illustrates matters.

Use of FFT for Convolution

The data, complete with zero padding, are now a set of real numbers s_j, $j = 0, \ldots, N-1$, and the response function is zero padded out to duration N and arranged in wrap-around order. (Generally this means that a large contiguous section of the r_k's, in the middle of that array, is zero, with nonzero values clustered at the two extreme ends of the array.) You now compute the discrete convolution as follows: Use the FFT algorithm to compute the discrete Fourier transform of s and of r. Multiply the two transforms together component by component, remembering that the transforms consist of complex numbers. Then use the FFT algorithm to take the inverse discrete Fourier transform of the products. The answer is the convolution $r * s$.

What about *deconvolution*? Deconvolution is the process of *undoing* the smearing in a data set that has occurred under the influence of a known response function, for example, because of the known effect of a less-than-perfect measuring apparatus. The defining equation of deconvolution is the same as that for convolution, namely (13.1.1), except now the left-hand side is taken to be known, and (13.1.1) is to be considered as a set of N linear equations for the unknown quantities s_j. Solving these simultaneous linear equations in the time domain of (13.1.1) is unrealistic in most cases, but the FFT renders the problem almost trivial. Instead of multiplying the transform of the signal and response to get the transform of the convolution, we just divide the transform of the (known) convolution by the transform of the response to get the transform of the deconvolved signal.

This procedure can go wrong *mathematically* if the transform of the response function is exactly zero for some value R_n, so that we can't divide by it. This indicates that the original convolution has truly lost all information at that one frequency, so that a reconstruction of that frequency component is not possible. You should be aware, however, that apart from mathematical problems, the process of deconvolution has other practical shortcomings. The process is generally quite sensitive to noise in the input data, and to the accuracy to which the response function r_k is known. Perfectly reasonable attempts at deconvolution can sometimes produce nonsense for these reasons. In such cases you may want to make use of the additional process of *optimal filtering*, which is discussed in §13.3.

Here is our routine for convolution and deconvolution, using the FFT as implemented in `four1` of §12.2. Since the data and response functions are real, not complex, both of their transforms can be taken simultaneously using `twofft`. Note, however, that two calls to `realft` should be substituted if `data` and `respns` have very different magnitudes, to minimize roundoff. The data are assumed to be stored in a `float` array `data[1..n]`, with `n` an integer power of two. The response function is assumed to be stored in wrap-around order in a sub-array `respns[1..m]` of the array `respns[1..n]`. The value of `m` can be any *odd* integer less than or equal to `n`, since the first thing the program does is to recopy the response function into the appropriate wrap-around order in `respns[1..n]`. The answer is provided in `ans`.

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```
#include "nrutil.h"

void convlv(float data[], unsigned long n, float respns[], unsigned long m,
    int isign, float ans[])
```
Convolves or deconvolves a real data set data[1..n] (including any user-supplied zero padding) with a response function respns[1..n]. The response function must be stored in wrap-around order in the first m elements of respns, where m is an odd integer $\leq$ n. Wrap-around order means that the first half of the array respns contains the impulse response function at positive times, while the second half of the array contains the impulse response function at negative times, counting down from the highest element respns[m]. On input isign is $+1$ for convolution, -1 for deconvolution. The answer is returned in the first n components of ans. However, ans must be supplied in the calling program with dimensions [1..2*n], for consistency with twofft. n MUST be an integer power of two.
```
{
    void realft(float data[], unsigned long n, int isign);
    void twofft(float data1[], float data2[], float fft1[], float fft2[],
        unsigned long n);
    unsigned long i,no2;
    float dum,mag2,*fft;

    fft=vector(1,n<<1);
    for (i=1;i<=(m-1)/2;i++)                Put respns in array of length n.
        respns[n+1-i]=respns[m+1-i];
    for (i=(m+3)/2;i<=n-(m-1)/2;i++)        Pad with zeros.
        respns[i]=0.0;
    twofft(data,respns,fft,ans,n);          FFT both at once.
    no2=n>>1;
    for (i=2;i<=n+2;i+=2) {
        if (isign == 1) {
            ans[i-1]=(fft[i-1]*(dum=ans[i-1])-fft[i]*ans[i])/no2;     Multiply FFTs
            ans[i]=(fft[i]*dum+fft[i-1]*ans[i])/no2;                  to convolve.
        } else if (isign == -1) {
            if ((mag2=SQR(ans[i-1])+SQR(ans[i])) == 0.0)
                nrerror("Deconvolving at response zero in convlv");
            ans[i-1]=(fft[i-1]*(dum=ans[i-1])+fft[i]*ans[i])/mag2/no2;Divide FFTs
            ans[i]=(fft[i]*dum-fft[i-1]*ans[i])/mag2/no2;             to deconvolve.
        } else nrerror("No meaning for isign in convlv");
    }
    ans[2]=ans[n+1];                        Pack last element with first for realft.
    realft(ans,n,-1);                       Inverse transform back to time domain.
    free_vector(fft,1,n<<1);
}
```

Convolving or Deconvolving Very Large Data Sets

If your data set is so long that you do not want to fit it into memory all at once, then you must break it up into sections and convolve each section separately. Now, however, the treatment of end effects is a bit different. You have to worry not only about spurious wrap-around effects, but also about the fact that the ends of each section of data *should* have been influenced by data at the nearby ends of the immediately preceding and following sections of data, but were not so influenced since only one section of data is in the machine at a time.

There are two, related, standard solutions to this problem. Both are fairly obvious, so with a few words of description here, you ought to be able to implement them for yourself. The first solution is called the *overlap-save method*. In this technique you pad only the very beginning of the data with enough zeros to avoid wrap-around pollution. After this initial padding, you forget about zero padding

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

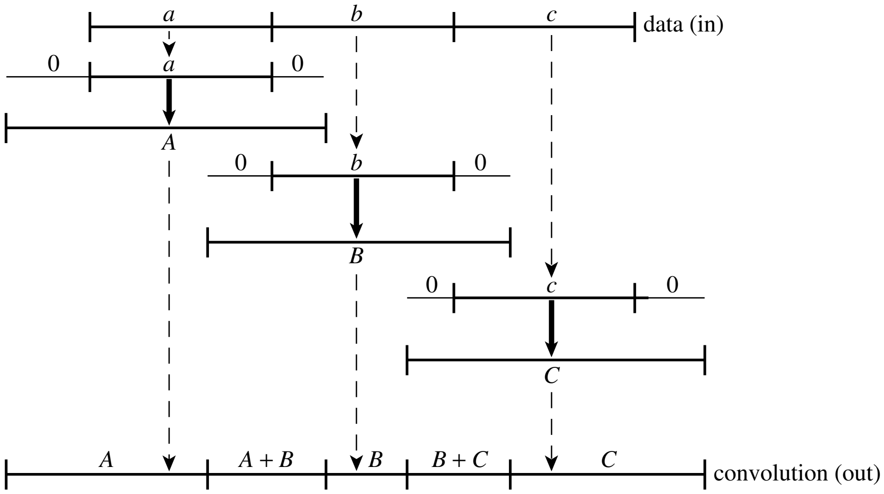

Figure 13.1.5. The overlap-add method for convolving a response with a very long signal. The signal data is broken up into smaller pieces. Each is zero padded at both ends and convolved (denoted by bold arrows in the figure). Finally the pieces are added back together, including the overlapping regions formed by the zero pads.

altogether. Bring in a section of data and convolve or deconvolve it. Then throw out the points at each end that are polluted by wrap-around end effects. Output only the remaining good points in the middle. Now bring in the next section of data, but not all new data. The first points in each new section overlap the last points from the preceding section of data. The sections must be overlapped sufficiently so that the polluted output points at the end of one section are recomputed as the first of the unpolluted output points from the subsequent section. With a bit of thought you can easily determine how many points to overlap and save.

The second solution, called the *overlap-add method*, is illustrated in Figure 13.1.5. Here you *don't* overlap the input data. Each section of data is disjoint from the others and is used exactly once. However, you carefully zero-pad it at both ends so that there is no wrap-around ambiguity in the output convolution or deconvolution. Now you overlap *and add* these sections of output. Thus, an output point near the end of one section will have the response due to the input points at the beginning of the next section of data properly added in to it, and likewise for an output point near the beginning of a section, *mutatis mutandis*.

Even when computer memory is available, there is some slight gain in computing speed in segmenting a long data set, since the FFTs' $N \log_2 N$ is slightly slower than linear in N. However, the log term is so slowly varying that you will often be much happier to avoid the bookkeeping complexities of the overlap-add or overlap-save methods: If it is practical to do so, just cram the whole data set into memory and FFT away. Then you will have more time for the finer things in life, some of which are described in succeeding sections of this chapter.

CITED REFERENCES AND FURTHER READING:

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

Sample page from NUMERICAL RECIPES IN C: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43108-5)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).