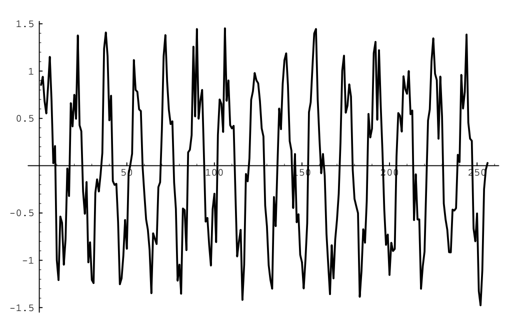

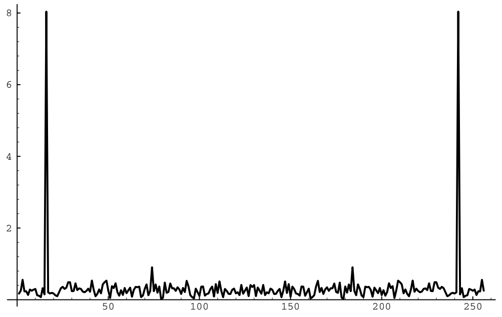

INPUT OUTPUT

13.11 Discrete Fourier Transform

Input description: A sequence of n real or complex values h_i, $0 \leq i \leq n-1$, sampled at uniform intervals from a function h.

Problem description: The discrete Fourier transform $H_m = \sum_{k=0}^{n-1} h_k e^{2\pi ikm/n}$ for $0 \leq m \leq n-1$.

Discussion: Although computer scientists tend to be fairly ignorant about Fourier transforms, electrical engineers and signal processors eat them for breakfast. Functionally, Fourier transforms provide a way to convert samples of a standard time-series into the *frequency domain*. This provides a dual representation of the function in which certain operations become easier than in the time domain. Applications of Fourier transforms include:

- *Filtering* – Taking the Fourier transform of a function is equivalent to representing it as the sum of sine functions. By eliminating undesirable high- and/or low-frequency components (i.e. , dropping some of the sine functions) and taking an inverse Fourier transform to get us back into the time domain, we can filter an image to remove noise and other artifacts. For example, the sharp spike in the figure above represents the period of the single sine function that closely models the input data. The rest is noise.

- *Image compression* – A smoothed, filtered image contains less information than the original, while retaining a similar appearance. By eliminating the coefficients of sine functions that contribute relatively little to the image, we can reduce the size of the image at little cost in image fidelity.

- *Convolution and deconvolution* – Fourier transforms can efficiently compute convolutions of two sequences. A *convolution* is the pairwise product of elements from two different sequences, such as in multiplying two n-variable polynomials f and g or comparing two character strings. Implementing such products directly takes $O(n^2)$, while the fast Fourier transform led to a $O(n \lg n)$ algorithm.

Another example comes from image processing. Because a scanner measures the darkness of an image patch instead of a single point, the scanned input is always blurred. A reconstruction of the original signal can be obtained by deconvoluting the input signal with a Gaussian point-spread function.

- *Computing the correlation of functions* – The *correlation function* of two functions $f(t)$ and $g(t)$ is defined by

$$z(t) = \int_{-\infty}^{\infty} f(\tau) g(t+\tau) d\tau$$

and can be easily computed using Fourier transforms. When two functions are similar in shape but one is shifted relative to the other (such as $f(t) = \sin(t)$ and $g(t) = \cos(t)$), the value of $z(t_0)$ will be large at this shift offset t_0. As an application, suppose that we want to detect whether there are any funny periodicities in our random-number generator. We can generate a large series of random numbers, turn them into a time series (the ith number at time i), and take the Fourier transform of this series. Any funny spikes will correspond to potential periodicities.

The discrete Fourier transform takes as input n complex numbers h_k, $0 \leq k \leq n-1$, corresponding to equally spaced points in a time series, and outputs n complex numbers H_k, $0 \leq k \leq n-1$, each describing a sine function of given frequency. The discrete Fourier transform is defined by

$$H_m = \sum_{k=0}^{n-1} h_k e^{-2\pi i k m/n}$$

and the inverse Fourier transform is defined by

$$h_m = \frac{1}{n} \sum_{k=0}^{n-1} H_k e^{2\pi i k m/n}$$

which enables us move easily between h and H.

Since the output of the discrete Fourier transform consists of n numbers, each of which is computed using a formula on n numbers, they can be computed in $O(n^2)$ time. The fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform in $O(n \log n)$. This is arguably the most important algorithm known, for it opened the door to modern signal processing. Several different algorithms call themselves FFTs, all of which are based on a divide-and-conquer approach. Essentially, the problem of computing the discrete Fourier transform on

n points is reduced to computing two transforms on $n/2$ points each, and is then applied recursively.

The FFT usually assumes that n is a power of two. If this is not the case, you are usually better off padding your data with zeros to create $n = 2^k$ elements rather than hunting for a more general code.

Many signal-processing systems have strong real-time constraints, so FFTs are often implemented in hardware, or at least in assembly language tuned to the particular machine. Be aware of this possibility if the codes prove too slow.

Implementations: FFTW is a C subroutine library for computing the discrete Fourier transform in one or more dimensions, with arbitrary input size, and supporting both real and complex data. It is the clear choice among freely available FFT codes. Extensive benchmarking proves it to be the "Fastest Fourier Transform in the West." Interfaces to Fortran and C++ are provided. FFTW received the 1999 J. H. Wilkinson Prize for Numerical Software. It is available at *http://www.fftw.org/*.

FFTPACK is a package of Fortran subprograms for the fast Fourier transform of periodic and other symmetric sequences, written by P. Swartzrauber. It includes complex, real, sine, cosine, and quarter-wave transforms. FFTPACK resides on Netlib (see Section 19.1.5 (page 659)) at *http://www.netlib.org/fftpack*. The GNU Scientific Library for C/C++ provides a reimplementation of FFTPACK. See *http://www.gnu.org/software/gsl/*.

Algorithm 545 [Fra79] of the *Collected Algorithms of the ACM* is a Fortran implementation of the fast Fourier transform optimizing virtual memory performance. See Section 19.1.5 (page 659) for further information.

Notes: Bracewell [Bra99] and Brigham [Bri88] are excellent introductions to Fourier transforms and the FFT. See also the exposition in [PFTV07]. Credit for inventing the fast Fourier transform is usually given to Cooley and Tukey [CT65], but see [Bri88] for a complete history.

A cache-oblivious algorithm for the fast Fourier transform is given in [FLPR99]. This paper first introduced the notion of cache-oblivious algorithms. The FFTW is based on this algorithm. See [FJ05] for more on the design of the FFTW.

An interesting divide-and-conquer algorithm for polynomial multiplication [KO63] does the job in $O(n^{1.59})$ time and is discussed in [AHU74, Man89]. An FFT-based algorithm that multiplies two n-bit numbers in $O(n \lg n \lg \lg n)$ time is due to Schönhage and Strassen [SS71] and is presented in [AHU74].

It is an open question of whether complex variables are really fundamental to fast algorithms for convolution. Fortunately, fast convolution can be used as a black box in most applications. Many variants of string matching are based on fast convolution [Ind98].

In recent years, wavelets have been proposed to replace Fourier transforms in filtering. See [Wal99] for an introduction to wavelets.

Related Problems: Data compression (see page 637), high-precision arithmetic (see page 423).