Fast Fourier Transforms on the Rabbit

The Fourier transform is frequently used to analyze and modify real world signals and waveforms. The Fourier transform is usually defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt$$

When the independent variable corresponds to time, the transform variable corresponds to frequency, in which case X(f) is called the spectrum of x(t). The original x(t) may be recovered from X(f) by the *inverse* Fourier transform, defined by:

$$(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}df$$

In the discrete-time, sampled-data domain of the digital computer, the Fourier transform becomes the *discrete-time* Fourier transform. The continuous-time variable t becomes the sample number n, and integrals become summations over a finite block of samples. The discrete-time Fourier transform (DFT) of a sequence of N samples is defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \qquad k = 0, 1, ..., N\text{-}1$$

As in the continuous-time case, we can recover the original sequence by an inverse transform. The inverse DFT (IDFT) is given by:

$$(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi kn/N} \qquad n = 0, 1, ..., N\text{-}1$$

The summations above require on the order of N^2 operations to evaluate, a number which grows impractically large as the number of samples increases. The *fast* discrete-time Fourier transform, or FFT, is a DFT algorithm whose computation time grows only as $N\log_2 N$. For even modest values of N, the speed-up afforded by the FFT algorithm can be dramatic.

It should be stressed that the FFT is merely a fast algorithm for computing the DFT. It has exactly the same mathematical properties as the DFT. In the discussions below, we will usually refer to DFT with the understanding that it is computed using the FFT algorithm.

Applications

Probably the most common use of the DFT is to compute the frequency spectrum of a sampled time waveform, as might be obtained from a microphone or accelerometer output digitized by an analog-to-digital converter. Radar, sonar, and vibration analysis, for example, make extensive use of the DFT to extract and reveal signal characteristics that would otherwise be difficult or impossible to perceive from the time waveforms alone. A practical example using the Rabbit 2000 in vibration analysis is the monitoring of rotating machinery. By attaching one or more accelerometers to a piece of rotating machinery and computing the DFTs of their outputs, one can obtain an operating "signature" of the machinery. Changes in the signature over time can give early warning of mechanical problems or impending failure.

Because of its computational speed, the FFT is often used to perform *fast* discrete convolution. Discrete convolution normally requires on the order of N^2 operations. This number can be reduced to the order of $N \log_2 N$ by first transforming the time-domain waveforms to the frequency domain, multiplying the transforms together, and then inverse transform back to the time domain. For large values of N, the speed-up can be substantial. Fast convolution is especially attractive when a sampled-data sequence must be convolved with a filter function whose FFT can be computed and stored ahead of time.

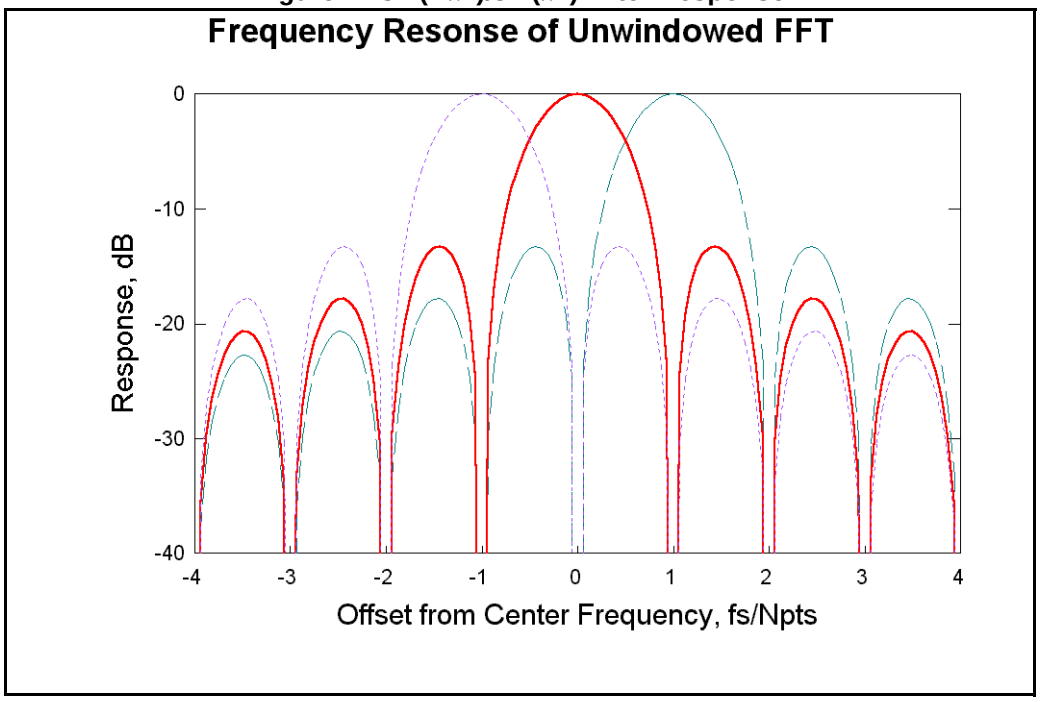
Another application of the FFT is in band-limited interpolation, in which a sequence's sample rate is increased by interpolating samples between original samples. For example, imagine that we have a sequence to convert to an analog waveform by a digital-to-analog converter. The waveform might be a stimulus for an experiment or a segment of speech. Such waveforms are often computed or stored at the minimum possible sample rate to save processor cycles and memory. Operating the digital to analog converter at the lowest possible rate complicates the design of the analog reconstruction filter used to smooth the DAC's stepped output. The requirements of the reconstruction filter can be relaxed considerably by increasing the sample rate into the converter. The DFT provides a convenient way to increase the sample rate without altering the waveform's character.

To see how the DFT can be used to increase the sample rate, suppose that the sequence to be converted has a sample rate of 2,500 samples per second and that we wish to increase the sample rate by a factor of 4 to 10,000 samples per second. We start with a convenient block of, say, 256 samples, and then compute its DFT. Since the samples represent a real signal, we use a version of the FFT that transforms 256 purely real samples into a 128-point complex spectrum that extends from zero frequency to 1.25 kHz. (The real-to-complex version of the FFT is about twice as fast as the complex-to-complex FFT.) Next we append 384 zero-valued points to the 128-point positive-frequency spectrum. This enlarged 512-point spectrum is precisely the spectrum that would have been computed had the original sequence been sampled at 10,000 samples/s. Performing a 512-point inverse DFT on the enlarged spectrum (using a complex-to-real version of the IFFT) produces a block of 1024 real samples that perfectly reproduce the waveform contained in the original 256-sample block.

Windowing

The DFT can be often viewed as a bank of digital filters, all having identical shape and having center frequencies whose absolute values range from zero to $F_s/2$, where F_s is the sampling frequency. The frequency response of these filters is described by $\sin(N\pi f)/\sin(\pi f)$. Three adjacent responses are shown in Figure 1.

Figure 1. $\sin(N\pi f)/\sin(\pi f)$ Filter Response



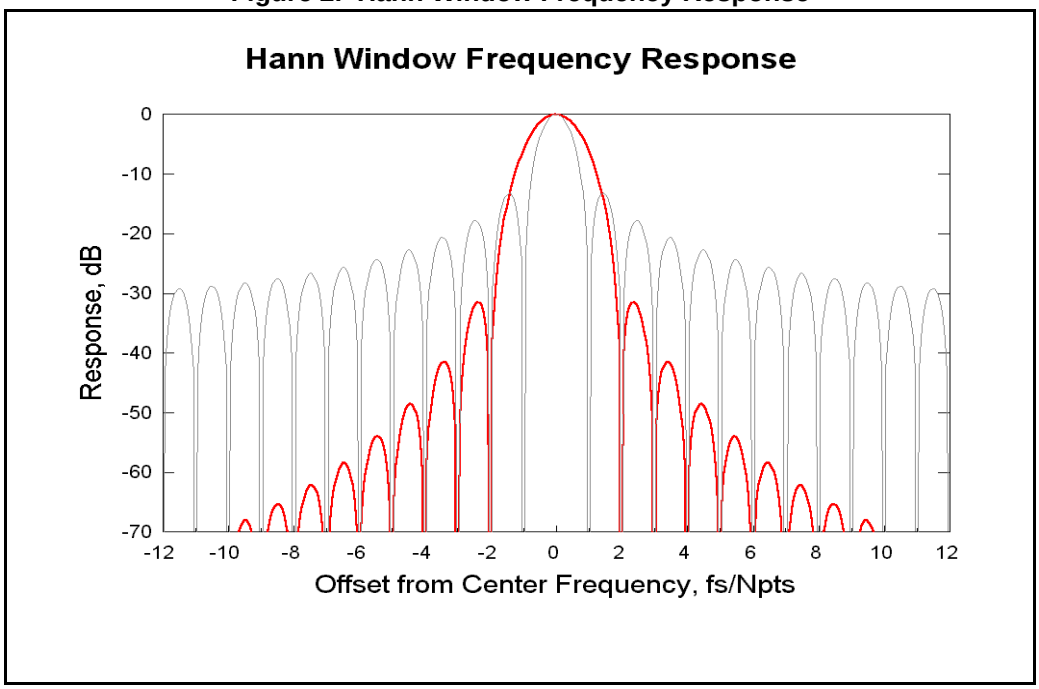
The bank of filters can be thought of as dividing the spectrum into a series of bins. Unfortunately the filters are often not so selective as one would like. First of all, the main lobes of adjacent filters overlap. As a result, a frequency component that falls in between the center frequencies of two filters contributes to the outputs of both filters. Second, because the filters have sizeable side lobes that extend far from the main lobe, a frequency component that falls in between the centers of two filters also contributes to the outputs of distant filters as well. Consequently, a given DFT output bin may contain misleading contributions from frequency components that lie far away from its center frequency.

While the overlap of the main lobes is unavoidable, the poor side lobe attenuation of the DFT can be improved by a process called “windowing.” Windowing modifies the frequency response of the FFT bins by giving a lower weight to samples that lie closer to the ends of the block of input samples. A frequently used window function, and the one implemented by the Rabbit FFT library, is the Hann window, sometimes called the "raised cosine" window. It is defined as

$$w(n) = 0.5\left\{1.0 + \cos\left[\frac{2\pi}{N}\left(n - \frac{N}{2}\right)\right]\right\} \qquad n = 0,1, \ldots, N\text{-}1$$

Each input sample is multiplied by the corresponding window sample before being transformed. Figure 2 compares the Hann window frequency response to the basic response of the DFT. The improved side-lobe response is clearly evident. With the improved side-lobe response comes what is usually a minor drawback: the width of the main lobe produced by the Hann window is twice the width of the basic response.

Figure 2. Hann Window Frequency Response



(A note on implementation: The Hann window has a particularly simple, three-point DFT. In the Rabbit FFT library, the Hann window function convolves the three-point DFT with the FFT output spectrum. This approach has the advantage of not requiring the computation or storage of raised cosine curves for all possible FFT sizes.)

Real Signals

Practical applications of the DFT usually deal with real signals, as might be produced by an analog-to-digital converter. It is possible to use an N-point complex DFT to analyze a $2N$-point real input sequence. The computed N-point complex spectrum will be the positive-frequency portion of the sequence's complex spectrum. No information is lost by computing only the positive-frequency portion of the spectrum because the spectrum of a real sequence is *conjugate symmetric*. This means that the negative-frequency portion of the spectrum can be computed from the positive-frequency spectrum via the relations:

$$\text{Re } X[-f] = \text{Re } X[f]$$
$$\text{Im } X[-f] = -\text{Im } X[f]$$

In practice the negative-frequency portion of the spectrum is rarely needed.

The Rabbit FFT library contains both complex-in/complex-out and real-in/complex-out FFT and their inverse counterparts.

Power Spectrum

When the DFT is used to analyze a signal (either real or complex), one is often concerned not so much with the amplitude and phase of the signal's spectrum as with either its power or its magnitude. The power spectrum is given by:

$$Power[k] = (\mathrm{Re}\ X[k])^2 + (\mathrm{Im}\ X[k])^2$$

When the magnitude is required, it can be computed by extracting the square root of each term in the power spectrum. The Rabbit FFT library provides a fast, full-precision function to compute the power spectrum from a complex spectrum.

The Rabbit FFT Library

The Rabbit FFT library comprises seven functions:

```
fftcplx()
fftcplxinv()
fftreal()
fftrealinv()
hanncplx()
hannreal()
powerspectrum()
```

These functions operate on a sequence or a spectrum stored in a single array. Data in the array are considered to be in the form of signed, 16-bit fractions with values that range from –1 to $1 - 2^{15}$. The effect of interpreting the data as fractions shows up when two 16-bit numbers are multiplied together to produce a 32-bit result. The 16-bit fractional arithmetic retains the 16 more significant bits of the product, whereas 16-bit integer arithmetic retains the 16 less significant bits.

The best numerical accuracy is obtained when the data to be transformed (or inverse transformed) are multiplied by a scaling factor so as to be as large as possible and still be representable by 16 bits. This means that the data should be scaled so that the resulting values lie between -32768 and +32767 (which correspond to fraction values -1 and +32767/32768). For example, for best results a sequence whose values range from -2048 to +2047 should be scaled so that 2047 becomes 32767. In this case, the scaling factor is exactly 16. For a sequence whose values range from zero to +200, the optimum scaling factor is 32767/200 or 163. (One might be tempted to use a suboptimum scaling factor of 128 and accomplish the scaling by left-shifting seven positions. But because the Rabbit's multiply instruction is so fast, left-shifting seven places takes twice as long as multiplying.)

Depending on the particular function, the array data may be either complex or real. Complex values are stored such that real parts are contained in the even-numbered array positions and the imaginary parts are contained in the odd-numbered array positions.

The *k*th complex point in the array is given by

$$\mathrm{Re}\ X[k] = x[2k]$$
$$\mathrm{Im}\ X[k] = x[2k + 1]$$

Block floating point arithmetic is used to prevent arithmetic overflow. A binary exponent is associated with the entire array of data, *viz*,

$$x_k = x[k] \times 2^{\mathbf{blockexp}}$$

During processing any addition or subtraction that would result in an arithmetic overflow causes the entire array to be scaled down by a factor of two, thereby preventing the overflow. Each time the array is scaled, its block exponent is increased by one to compensate for this scaling. The actual value of the exponent is arbitrary, and the calling program may set it to any convenient value.

The Complex FFT Functions

These functions are in FFT.LIB.

fftcplx

```
void fftcplx(int *x, int N, int *blockexp);
```

DESCRIPTION

Computes the complex DFT of the N-point complex sequence contained in the array x and returns the complex result in x. N must be a power of 2 and lie between 4 and 1024. An invalid N causes a RANGE exception. The N-point complex sequence in array x is replaced with its N-point complex spectrum. The value of blockexp is increased by 1 each time array x has to be scaled, to avoid arithmetic overflow.

PARAMETERS

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array x.
blockexp	Pointer to integer block exponent.

fftcplxinv

```
void fftcplxinv(int *x, int N, int *blockexp);
```

DESCRIPTION

Computes the inverse complex DFT of the N-point complex spectrum contained in the array x and returns the complex result in x. N must be a power of 2 and lie between 4 and 1024. An invalid N causes a RANGE exception. The value of blockexp is increased by 1 each time array x has to be scaled, to avoid arithmetic overflow. The value of blockexp is also *decreased* by $\log_2 N$ to include the 1/N factor in the definition of the inverse DFT

PARAMETERS

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array x.
blockexp	Pointer to integer block exponent.

The Real FFT Functions

The `fftreal()` and `fftrealinv()` functions manipulate real sequences and positive-frequency spectra. The spectrum of a 2*N*-point real sequence has 2*N* complex points. However, it also has conjugate symmetry, so that its *N* + 1 positive-frequency components contain all the spectral information. The imaginary part of the zero-frequency, or dc term, and the imaginary part of the maximum-frequency, or *fmax* term, are always zero. The *fmax* term is the *N* + 1 term of the 2*N*-point spectrum, and so lies one position beyond the *N*-point positive-frequency complex spectrum. However, since the *fmax* term is purely real, we can store its value in the otherwise zero imaginary part of the dc term.

These functions are in `FFT.LIB`.

---

fftreal

```
void fftreal(int *x, int N, int *blockexp);
```

DESCRIPTION

Computes the `N`-point, positive-frequency complex spectrum of the 2`N`-point real sequence in array `x`. The 2`N`-point real sequence in array `x` is replaced with its `N`-point positive-frequency complex spectrum. To avoid arithmetic overflow, the value of `blockexp` is increased by 1 each time array `x` has to be scaled,.

The imaginary part of the *X*[0] term (stored in *x*[1]) is set to the real part of the *fmax* term.

The 2`N`-point real sequence is stored in natural order. The zeroth element of the sequence is stored in `x[0]`, the first element in `x[1]`, and the *k*th element in *x*[*k*].

`N` must be a power of 2 and lie between 4 and 1024. An invalid `N` causes a RANGE exception.

PARAMETERS

x	Pointer to 2`N`-point sequence of real fractions.
N	Number of complex elements in output spectrum
blockexp	Pointer to integer block exponent.

fftrealinv

```
void fftrealinv(int *x, int N, int *blockexp);
```

DESCRIPTION

Computes the 2N-point real sequence corresponding to the N-point, positive-frequency complex spectrum in array x. The N-point, positive-frequency spectrum contained in array x is replaced with its corresponding 2N-point real sequence. The value of blockexp is increased by 1 each time array x has to be scaled, to avoid arithmetic overflow. The value of blockexp is also *decreased* by $\log_2 N$ to include the 1/N factor in the definition of the inverse DFT.

The function expects to find the real part of the *fmax* term in the imaginary part of the zero-frequency X[0] term (stored x[1]).

The 2N-point real sequence is stored in natural order. The zeroth element of the sequence is stored in x[0], the first element in x[1], and the *k*th element in x[k].

N must be a power of 2 and lie between 4 and 1024. An invalid N causes a RANGE exception.

PARAMETERS

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array x.
blockexp	Pointer to integer block exponent.

The Auxiliary Functions

These functions are in FFT.LIB.

hanncplx

```
void hanncplx(int *x, int N, int *blockexp);
```

DESCRIPTION

Convolves an N-point complex spectrum with the three-point Hann kernel. The filtered spectrum replaces the original spectrum.

The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero–crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

N must be a power of 2 and lie between 4 and 1024. An invalid N causes a RANGE exception.

PARAMETERS

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array x.
blockexp	Pointer to integer block exponent.

hannreal

```
void hannreal(int *x, int N, int *blockexp);
```

DESCRIPTION

Convolves an N-point positive-frequency complex spectrum with the three-point Hann kernel. The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero–crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

The imaginary part of the dc term (stored in `x[1]`) is considered to be the real part of the *fmax* term. The dc and *fmax* spectral components take part in the convolution along with the other spectral components. The real part of *fmax* component affects the real part of the $X[N\text{-}1]$ component (and vice versa), and should not arbitrarily be set to zero unless these components are unimportant.

PARAMETERS

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array x.
blockexp	Pointer to integer block exponent.

RETURN VALUE

None. The filtered spectrum replaces the original spectrum.

powerspectrum

```
void powerspectrum(int *x, int N, *int blockexp);
```

DESCRIPTION

Computes the power spectrum from a complex spectrum according to

$$\text{Power[k]} = (\text{Re X[k]})^2 + (\text{Im X[k]})^2$$

The N-point power spectrum replaces the N-point complex spectrum. The power of each complex spectral component is computed as a 32-bit fraction. Its more significant 16-bits replace the imaginary part of the component; its less significant 16-bits replace the real part.

If the complex input spectrum is a positive-frequency spectrum computed by `fftreal()`, the imaginary part of the *X*[0] term (stored `x[1]`) will contain the real part of the *fmax* term and will affect the calculation of the dc power. If the dc power or the *fmax* power is important, the *fmax* term should be retrieved from `x[1]` and `x[1]` set to zero before calling `powerspectrum()`.

The power of the *k*th term can be retrieved by:

$$P[k] = *(long*) \ \& \ x[2k] \ * \ 2^{\text{blockexp}}$$

The value of `blockexp` is first doubled to reflect the squaring operation applied to all elements in array `x`. Then it is further increased by 1 to reflect an inherent division-by-two that occurs during the squaring operation.

PARAMETERS

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array `x`.
blockexp	Pointer to integer block exponent.

Timing

The execution time of all functions other than `powerspectrum()` depends somewhat on whether and by how much the data must be scaled to avoid arithmetic overflow. The following tables present the measured best case (no scaling) and worst case (maximum scaling) execution times for a 30 MHz Rabbit 2000 running under Dynamic C.

	Number of Complex Output Points					
	32		64		128	
	Execution Time (ms)		Execution Time (ms)		Execution Time (ms)	
	min	max	min	max	min	max
cplxfft	1.4	1.9	3.2	4.2	7.2	9.6
cplxfftinv	1.4	1.9	3.2	4.2	7.1	9.4
realfft	1.7	2.4	3.8	5.2	8.5	11.5
realfftinv	1.7	2.4	3.9	5.3	8.6	11.6
ffthanncplx	0.2	0.5	0.4	0.8	0.8	1.4
ffthannreal	0.2	0.5	0.4	0.8	0.8	1.4
powerspectrum	0.1		0.3		0.5	

	Number of Complex Output Points					
	256		512		1024	
	Execution Time (ms)		Execution Time (ms)		Execution Time (ms)	
	min	max	min	max	min	max
cplxfft	16.2	21.3	35.8	46.6	78.5	102.5
cplxfftinv	16.2	21.0	35.9	46.7	78.7	102.7
realfft	18.8	25.1	41.1	54.5	N/A[1]	
realfftinv	19.0	25.2	41.2	54.6		
ffthanncplx	1.5	2.7	2.9	5.3	5.8	10.6
ffthannreal	1.5	2.7	2.9	5.3	5.8	10.6
powerspectrum	1.0		2.0		4.0	

1 `realfft` and `realfftinv` are currently limited to 512 complex output points.

References

- A.V. Oppenheim and R.W. Shaeffer, *Digital Signal Processing*, Prentice-Hall, Inc., 1975.
  A good primer on digital signal processing with a good discussion of various forms of the FFT.

- A.V. Oppenheim and R.W. Shaeffer, *Discrete-Time Signal Processing*, Prentice-Hall, Inc., 1989.

- E.O. Brigham, *The Fast Fourier Transform*, Prentice-Hall, 1974.
  Gives a concise derivation of the real-input/complex-output FFT.

- W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, 1988. (A second edition was published in 1997.)
  Contains C-language examples of the FFT and of the bit-reversal function employed by Rabbit FFT.

- D.F. Elliot, Ed., *Handbook of Digital Signal Processing*, Academic Press, 1987.
  The appendix to Chapter 3 describes and compares various window functions.