

SIGNALS AND SYSTEMS LABORATORY 11:

The FFT and its Applications

INTRODUCTION

In 1965 there appeared one of the most important papers in the history of signal processing. This paper was entitled “An algorithm for the machine calculation of complex Fourier Series”, by J. W. Cooley and J. W. Tukey. (*Math. Comput.*, vol. 19, no. 2, April 1965). The algorithm mentioned was the Fast Fourier Transform, or FFT. The period in which the paper appeared coincided with the introduction of integrated circuits and the 74xx family of TTL devices. In the next decade these two developments came together and revolutionized the filtering and spectrum analysis of signals. Although the authors were not aware of earlier work, the idea behind the FFT had a long history. This idea involves a factorization or parsing of the DFT sum when the number of terms N is composite, or has integer factors. The more highly composite, the more the computational savings, and almost all FFTs involve sizes which are a power of two. Several predecessors to the Cooley-Tukey FFT have been documented. The earliest appears to be by C. F. Gauss, in 1805. A most interesting account is given in the paper “Gauss and the history of the fast Fourier transform”, by M. T. Heideman, D. H. Johnson, and C. S. Burrus (IEEE ASSP Magazine, October 1984).

NUMERICAL APPROXIMATION OF THE FOURIER TRANSFORM, USING THE FFT: A REVIEW

The FFT computes a DFT, but it can be used to also approximate the computation of Fourier series coefficients, the DTFT, and the Fourier integral transform. The most subtle approximation is for the Fourier transform. Let $X(j\omega)$ be the Fourier transform of a finite energy signal $x(t)$. By definition, this is the integral

$$(1) \quad X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt.$$

The best tool we have for numerically computing the Fourier integral in equation (1) is the Fast Fourier Transform. If we type the command $Y=\text{fft}(y)$ in MATLAB, then we will get back a vector Y having the same size as y , and whose values are given by

$$(2) \quad Y[n] = \sum_{k=0}^{N-1} y[k]W^{-nk}, \quad 0 \leq n \leq N-1.$$

Here $y[k] = x(kt_s)$ is a sample of $x(t)$, and $W = e^{j2\pi/N}$ is the principle N -th root of unity on the unit circle. Now let $\omega_s = 2\pi/t_s$ be the radian sampling frequency, and let $\omega_0 = \omega_s/N = 2\pi/Nt_s = 2\pi/T$. If the signal $x(t)$ is zero outside the interval $0 \leq t \leq T$, then $W = e^{j2\pi/N} = e^{j\omega_0 t_s}$, and

$$(3) \quad X(jn\omega_0) = \int_0^T x(t)e^{-jn\omega_0 t} dt \cong \sum_{k=0}^{N-1} x(kt_s)e^{-jn\omega_0(kt_s)} t_s = t_s \sum_{k=0}^{N-1} y[k]W^{-nk} = t_s Y[n].$$

Thus, the FFT provides a Riemann sum approximation to the integral. We can get approximations of samples of the Fourier Transform spaced uniformly in frequency by

$$(4) \quad f_0 = \frac{\omega_0}{2\pi} = \frac{\omega_s}{2\pi N} = \frac{1}{Nt_s} = \frac{1}{T} \text{ Hz.}$$

Rule 1: The resolution in frequency of the approximation is inversely proportional to the time duration of the set of samples.

The number samples in the sample-data sequence, and its DFT is

$$(5) \quad N = \frac{T}{t_s} = f_s T > 2BT$$

The number $2BT$ is called the *time-bandwidth product* of the signal, and measures its complexity. The approximation $X(jn\omega_0) \cong t_s Y_n$ in equation (3) is only good up to about the $N/2$ -th sample. The others will simply *wrap around* or *fold over*. Thus the computation is useful only up to a frequency of

$$(6) \quad (N/2)f_0 = f_s/2 \text{ Hz.}$$

Rule 2: The highest usable frequency $n\omega_0$ for the approximation $X(jn\omega_0) \cong t_s Y_n$ is the half sampling frequency $f_s/2$ Hz.

WHY IS THE FFT “FAST”? WE REVEAL THE SECRET.

The fast Fourier transform, or FFT, is a computationally efficient means of computing the discrete Fourier transform, or DFT, of size N . It is fast only when N is highly composite, and is usually only implemented when N is a power-of-two, i.e. $N = 2^\nu$.

The DFT has a matrix V of size N by N , with elements

$$(7) \quad V_{n,k} = W^{-nk}, \text{ where } 0 \leq k, n \leq N-1.$$

When N is 2, the DFT matrix is

$$(8) \quad V = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

When N is 8, the DFT matrix is

$$(9) \quad V = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Even though all of its elements are nonzero, this matrix can be factored as shown into very sparse matrices, meaning that the factors contain mostly zeros. The rightmost factor contains only one nonzero element per row, and this element is one. Thus multiplication is trivial. This matrix simply permutes the input data vector. The other two factors on the left contain two nonzero elements on each row. Signal flow graphs for the left and right hand sides of equation (9) are found in Figure One. The arrows are deleted for simplicity, but they all point to the right. The four nodes on the left side of these graphs are the input data vector elements. The four nodes on the right are the output DFT vector elements. For the matrix V the graph is dense, and contains every path connecting an input to an output. The graph for the factored form has three vertical layers corresponding to the three factors. The first layer on the left corresponds to the rightmost factor. When $N = 2^\nu$ the DFT matrix V has a factorization like that in equation (9) except that there will be $\nu+1$ factors. One factor will be a permutation, and involves no multiplication. The remaining factors all have two nonzero elements per row, and at least one of these is one. Thus multiplication by such a matrix requires at most N complex multiplies. A signal flow graph for the FFT for the case $N = 32 = 2^5$ is

shown in Figure Two. It has one vertical layer, which simply rearranges the data, and five layers with two branches to each node. From Figure One it is clear that the FFT is an *in-place* algorithm wherein a given pair of nodes is updated, in place, at each vertical layer. If we count multiplies, we see why the FFT is called *fast*. Multiplication by V without factorization requires N^2 multiplications, which is the number of nonzero elements in the matrix. If we use the factored form, we can do it in $N \nu = N \log_2 N$ multiplies. When $N = 1024 = 2^{10}$ the ratio of N^2 to $N \cdot \log_2 N$ is about 100. Thus the 1024-point FFT is 100 times faster than ordinary matrix multiplication, and this improvement increases with N . By the way, there are several ways to factor the DFT matrix in the power-of-two case. The ones exhibited are called *decimation in time* factorizations.

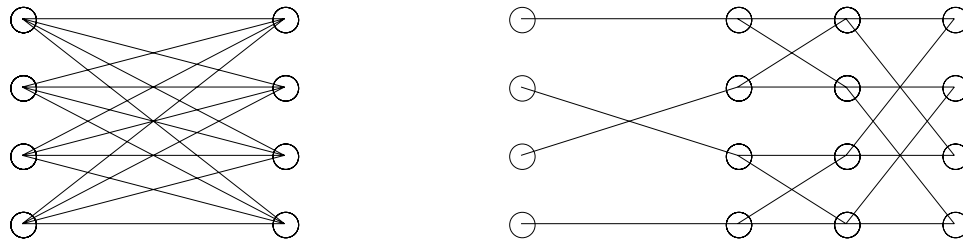


Figure One Signal flow graphs for the unfactored DFT matrix, and the factored form. The factored form is the signal flow graph for a 4 point fast Fourier transform.

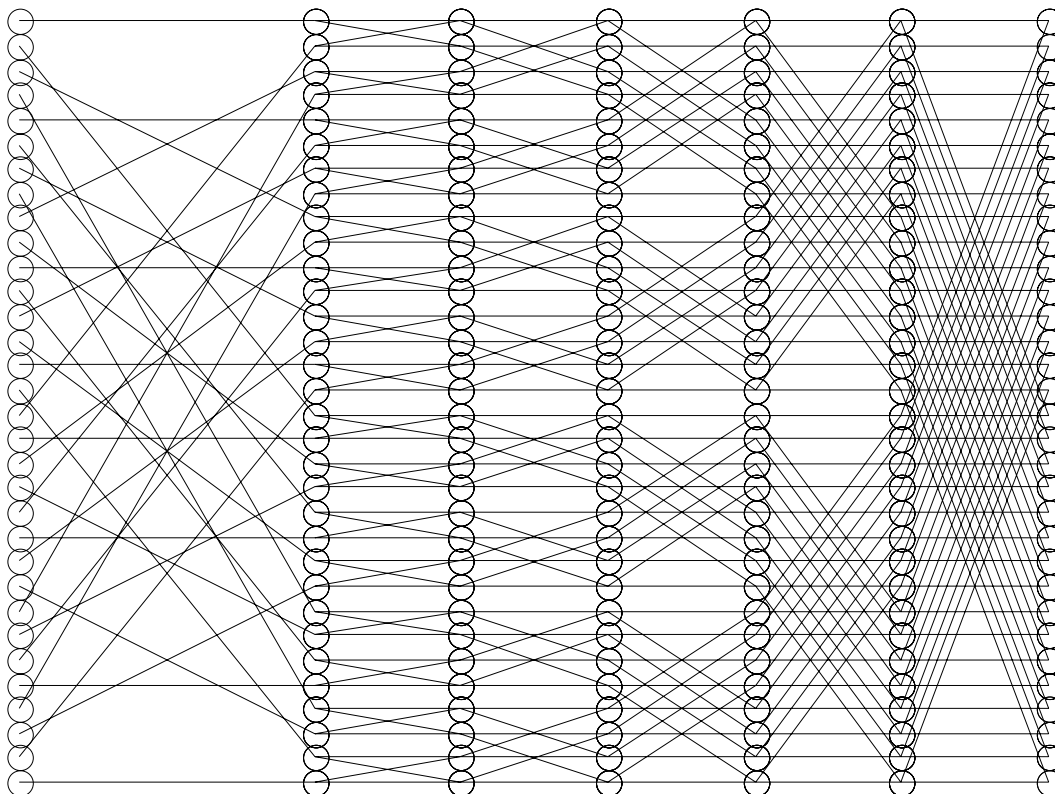


Figure Two A signal flow graph for a 32 point decimation in time fast Fourier Transform.

THE RELATION BETWEEN FREQUENCY AND INDEX IN THE DFT

The approximation in equation (3) suggests that the n -th component of the DFT corresponds to the frequency $n\omega_0$. This is true only up to the half sampling frequency, or $\omega_s/2 = N\omega_0/2 = n\omega_0$, at DFT index $n = N/2$. For the components with indices beyond this, in the range $N/2 < n \leq N-1$, there is a correspondence with *negative* frequencies, so that

$$(10) \quad \text{for } N/2 < n \leq N-1, \quad t_s Y[n] \cong X(n\omega_0 - \omega_s) = X(n\omega_0 - N\omega_0).$$

There is an associated symmetry property. If $x(t)$ is real, then $X(-j\omega) = \bar{X}(j\omega)$. For the DFT, if $y[n]$ is real valued, then

$$(11) \quad \text{For } N/2 < n \leq N-1, \quad Y[n] = \bar{Y}[N-n].$$

This should be considered when plotting the output of the MATLAB ‘fft’ function. An example is given in Figure Three. Suppose that we construct a 512-point vector, and its DFT magnitude as follows:

```
»[b,a]=butter(6,.1);
»N=512;
»x=filter(b,a,randn(1,N));
»Xm=abs(fft(x));
```

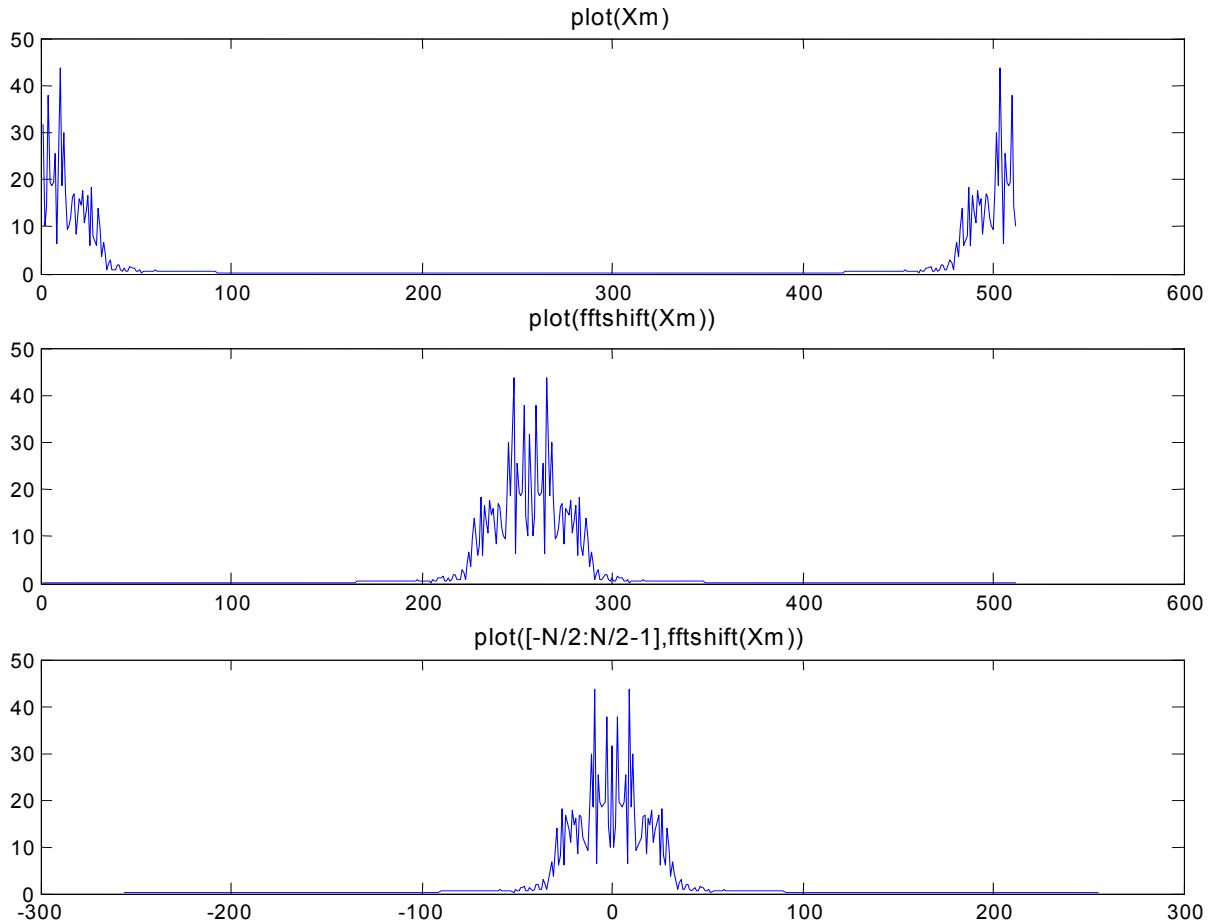


Figure Three Plotting spectra produced by the FFT.

The signal is lowpass with bandwidth one tenth of the half sampling frequency. If we plot the magnitude spectrum X_m directly, we get the result in the top panel of Figure Three. This is misleading since the right hand half looks like a high frequency signal. MATLAB provides a function called 'fftshift' which rearranges the DFT vector so that it plots in a more natural way, with negative and positive frequency parts. It takes the second half of the vector (with DFT indices in the range $N/2 \leq n \leq N-1$) and inserts it in front of the first half of the vector (with indices in the range $0 \leq n \leq (N/2)-1$). (Type `fftshift(1:16)` to see the permutation.) If one plots the result, it looks like the middle panel of Figure Three. This is better, except that the horizontal labeling is now incorrect. One can correct this by constructing a new index set as shown in the title for the third panel in Figure Three. To plot against frequency in Hertz, use `(fs/N)*[-N/2:N/2-1]` for the first argument in the plot, where `fs` is the sampling frequency.

Assignment

1. The FFT algorithm

Look at the signal flow graph for the 32-point FFT in Figure Two. Then decide the following questions. True or False:

- For any input node i and output node j , there is exactly one path which connects them.
- The path in part a. can be coded by a sequence of up/down decisions. For output node j , the sequence of decisions is the same for all input nodes.
- The sequence of decisions has a binary code which is the reversal of the binary code for the output node address. For example if $j = 5$, its address is 00101. The reversal is 10100. Interpret this as (down, up, down, up, up). Start at any input node i and use this sequence when you encounter a choice of branches. Keep in mind the fact that indexing begins with all zeros. Thus node $j = 5$ is the sixth one down.

Now on the computer, examine signal flow graphs for power-of-two FFT's using the 'drawfft.m' function on the web page under 'Functions for Lab 11'. Type

»drawfft(1).

Repeat this for input parameters of 2, 3, 4, and 5. Don't print any of these graphs, for it would take too much time. In each case count the number of vertical *butterfly* layers. (The order in which the branches are drawn is the order in which multiplies would be done in the FFT algorithm.)

Assignment

2. The DFT matrix

To construct the FFT matrix in MATLAB of size N by N , type

```
»V=fft(eye(N)).
```

This operation returns a matrix whose columns are the DFT of the columns of the identity matrix, which is of course the matrix V . All elements of V are powers of the N -th root of unity $W = e^{j2\pi/N}$. To see the matrix of exponents, type

```
»theta=round(N/(2*pi)*angle(V))
```

- Hand record the matrix of powers for $N = 2, 4, 8$. Does the pattern make sense?
- Usually, the eigenvalues of a matrix are complicated, but for the DFT they don't seem to be. Look at the eigenvalues of V by typing

```
»eig(V).
```

Record the eigenvalues for sizes $N = 2, 4, 8$. Make a conjecture about what the eigenvalues are for the general case, i.e. for any N . Test this conjecture for large N , and for N not a power of two.

- In MATLAB, the *prime* operator applied to a matrix delivers the conjugate transpose. The product of V and its transpose conjugate is meaningful. To see this, display

```
»V=fft(eye(4)),V',V*V'
```

Do this for various sizes of N , and figure out what the general case is. Then come up with a formula for the inverse of V .

- Investigate V^2 , and V^4 for various sizes of N . For $N = 4$, type

```
»V=fft(eye(4)),V^2,V^4
```

What do you conclude about the general case (arbitrary N)?

- Type

```
»V=fft(eye(4));R=toeplitz([0 0 0 1]',[0 1 0 0]),V*R*V',V'*R*V
```

Record the results. The matrix R is called the *rotation matrix*. Look at its powers:

```
»R,R^2,R^3,R^4,R^5
```

and explain the pattern. Why is V^*R^*V' a diagonal matrix?

Assignment

3. Zero padding and zero filling

Construct a vector s of size 1 by 32 containing samples of a sinusoid at $.4*fs/2$. For each of the following cases, make a plot of x and $\text{abs}(\text{fft}(x))$ using the style of the third panel of Figure Three. Then explain what you see using the properties of the DFT. Can you think of applications?

- a. `»x=s; % no zeros added`
- b. `»N=length(s);x=[s,zeros(1,7*N)]; % zero-pad in time`
- c. `»x=[s,s,s]; % repetition in time`
- d. `»x=s;x=d_up(s,4,0); % zero-fill in time`
- e. `»Y=fft(s);N=length(Y);M=N/2;`
`»X=[Y(1:M-1),.5*Y(M),zeros(1,3*N),.5*Y(M),Y(M+1:N)]; % zero-pad in frequency`
`»x=real(ifft(X));`
- f. `»Y=fft(s);X=d_up(Y,4,0);x=real(ifft(X)); % zero-fill in frequency`

4. Bandpass filters

One can implement bandpass filters using the FFT, as follows. For a signal vector of length N ,

- i. Take the FFT.
- ii. Replace all values in frequency *bins* falling in the stop band with zeros.
- iii. Compute the inverse FFT.

One must be careful to treat those indices corresponding to negative frequencies properly. Do the following example.

```
»load speech1
```

This data file contains a vector x of length 8192 samples, and the number fs , which is a sampling frequency. Listen to it via

```
»sound(x,fs).
```

Now let $X=\text{fft}(x)$. Construct the complex vector Y by setting it to X and then zeroing out all values which correspond to the frequencies 800 Hz to 1000 Hz. (You will need the value of fs to do this.) The stop band will consist of two groups of consecutive values, one group for positive frequencies, and one for negative frequencies. The MATLAB indices for Y are 1 to 8192. Report the index ranges of the values that you zero out. After you have constructed Y , then type the following:

```
»Y=X;
»plot((fs/8192)*[-4096:4095],fftshift(abs(Y)))
»y=ifft(Y);
»check=sum(abs(imag(y)))
»y=real(y);
```

The value of `check` should be nearly zero. If it isn't, you haven't zeroed out the right groups of values. Listen to the signal y .