Discrete Fourier Transform (DFT)
&
Fast Fourier Transform (FFT)

Lab 9

Last Time

We found that an approximation to the Continuous Time Fourier Transform may be found by sampling $x(t)$ at every $m\Delta t$ and turning the continuous Fourier integral into a discrete sum.

$$X\left(k\frac{1}{N\Delta t}\right) \cong \Delta t \underbrace{\sum_{m=0}^{N-1} x(m\Delta t) e^{-j2\pi km/N}} = \Delta t \cdot \mathcal{DFT}\big(x(m\Delta t)\big)$$

We gave this a name: Discrete Fourier Transform (DFT).

The DFT Pair

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{+j2\pi kn/N} \quad \overset{\mathcal{DFT}}{\longleftrightarrow} \quad X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$$

These definitions assume that the first nonzero elements of $x[n]$ and $X[k]$ are $x[0]$ and $X[0]$!

If your data (and program) do not follow this convention then there will be a phase shift in the forward DFT. A similar problem occurs for the reverse DFT.

For DFT $X[k]$ is periodic!

As it turns out $X[k]$ is periodic with period N regardless of the nature of $x[n]$!

$$x(t) \overset{\mathcal{F}}{\leftrightarrow} X(f) \qquad\qquad \delta_{T_0} \overset{\mathcal{F}}{\leftrightarrow} \frac{1}{T_0}\delta_{f\,0}$$

Sampling a continuous-time signal is multiplying by a impulse train in the time domain. This of course has the result that the Fourier Transform is convolved with a impulse train resulting in shifted versions (periodic) in the frequency domain.

$$\therefore x(t) \cdot \delta_{T_0} \overset{\mathcal{F}}{\leftrightarrow} \frac{1}{T_0}\delta_{f\,0} * X(f)$$

DFT is a misnomer!
It's actually equivalent to Discrete-Time Fourier Series.

DTFS $$x[n] = \sum_{k=0}^{N-1} c_k[k] e^{j2\pi kn/N}$$

Discrete

$$x(t) = \sum_{k=0}^{N-1} c_k[k] e^{j2\pi kn/N}$$ CTFS

Continuous Discrete

=

Inverse DFT: $$x[n] = \sum_{k=0}^{N-1} \frac{X[k]}{N} e^{j2\pi kn/N}$$

Why do we care?

MATH WORLD

CTFS

$$x(t) = \sum_{k=0}^{\infty} c_x[k] e^{j2\pi k/T}$$

$$c_x[k] = \frac{1}{T}\int_T x(t)e^{-j2\pi k/T}dt$$

Real WORLD (MATLAB)

(Approximate CTFS using DFT)

$$c_x[k] \cong \frac{1}{N} \cdot \mathcal{DFT}\big(x(n\Delta t)\big)$$

You should care!

MATH WORLD

CTFT

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

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

Real WORLD (MATLAB)

(Approximate CTFT using DFT)

$$X\left(\frac{k}{\Delta tN}\right) \cong \Delta t \cdot \mathcal{DFT}\left(x(n\Delta t)\right)$$

$$|k| \ll N$$

Mindblown

MATH WORLD	Real WORLD (MATLAB)
DTFT	(Approximate DTFT using DFT)
	$F \rightarrow \frac{k}{N}$ and $0 < n < N - 1$
$X(F) = \sum_{n=-\infty}^{\infty} x[n] e^{-j2\pi F n}$	$X\left(\frac{k}{N}\right) \cong \mathcal{DFT}(x[n])$
$x[n] = \int_{1} X(F) e^{j2\pi F n} dF$	$x[n] \cong \frac{1}{N} \cdot \mathcal{DFT}^{-1}\left(X\left(\frac{k}{N}\right)\right)$

Fourier Family

	Continuous Frequency, $X(f)$	Discrete Frequency, $X[k]$
Continuous Time, $x(t)$	Continuous Time Fourier Transform (CTFT)	Continuous Time Fourier Series (CTFS)
Discrete Time, $x[n]$	Discrete Time Fourier Transform (DTFT)	Discrete Time Fourier Series (DTFS) -OR- Discrete Fourier Transform (DFT)

DFT is the workhorse for Fourier Analysis in MATLAB!

DFT Implementation

Textbook's code pg. 303 is slow because of the awkward nested for-loops. The code we built in last lab is *much faster* because it has a single for-loo.

Our code

```
Elapsed time is 0.000466 seconds.
Elapsed time is 0.073929 seconds.
```

Textbook's code

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$$

Speed -FFT

"Uncle" Gauss

An DFT algorithm which decrease the number of computation (thereby decreasing the computation time) is called the Fast Fourier Transform (FFT).

BUT!!!! It only is efficient when N is a integer power of two,

N={1,2,4,16,32,64,128,256,512,1024,…}

Otherwise there is no reduction in computation complexity!

fft() and ifft()

In MATLAB the FFT algorithm is already programmed in .

fft(x) operates on a vector x (in our case a discrete-time signal) and gives back the DFT of x. CAREFUL, It may need to be normalized!

Likewise ifft(y) operates on a vector y (in our case a discrete-frequency representation of a signal) and gives back the inverse DFT of y.

DFT Code to Approximate CTFT

```
deltat=0.01;                    %time resolution
T=1.28;                         %period
N=T/deltat;                     %number of sample points
m=0:N;                          %time index
a=-1j*2*pi/N;                   %constant used below
xt=us(m*deltat).*exp(-m.*deltat); %time sampled signal
%preallotcate frequency domain vector
Xf=zeros(1,N);
for k=0:N-1
    temp=xt.*exp(a*k.*m);
    Xf(k+1)=deltat*sum(temp);
end
stem(abs(Xf))
```

$$X\left(k\frac{1}{N\Delta t}\right) \cong \Delta t \cdot \mathcal{DFT}\big(x(m\Delta t)\big)$$

FFT Code to Approximate CTFT

```
deltat=0.01;                        %time resolution
T=1.28;                             %period
N=T/deltat;                         %number of sample points
m=0:N;                              %time index
xt=us(m*deltat).*exp(-m.*deltat);%time sampled signal
Xf=deltat*fft(xt);                  %FFT to compute CTFT approx.
stem(abs(y))
```

$$X\left(k\frac{1}{N\Delta t}\right) \cong \Delta t \cdot \mathcal{FFT}\big(x(m\Delta t)\big)$$

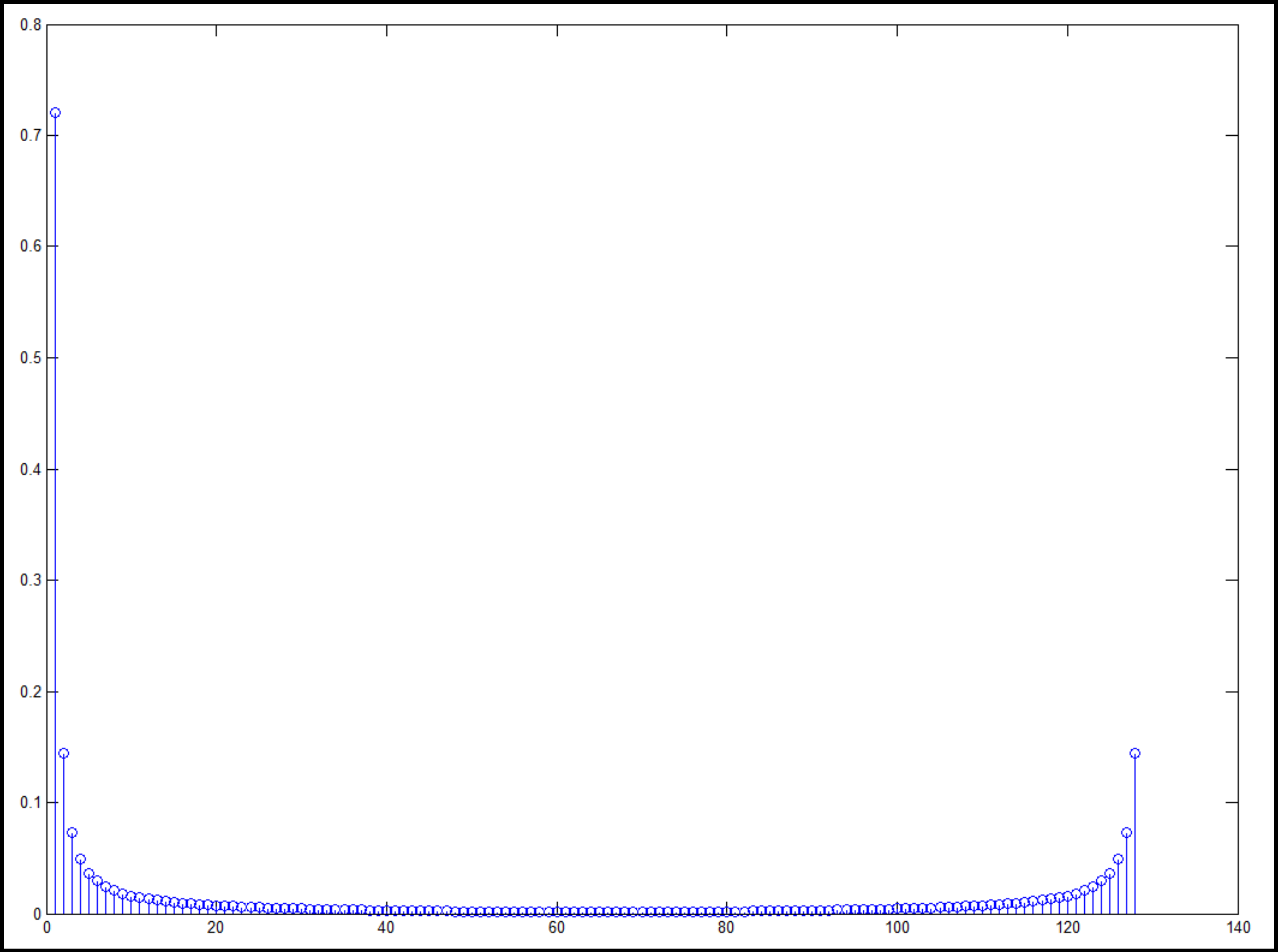
fftshift()

Remember I said $X[k]$ is periodic with period N? We can use this to our advantage to plot the frequency "spectrum" by shifting the period to center around the DC component.

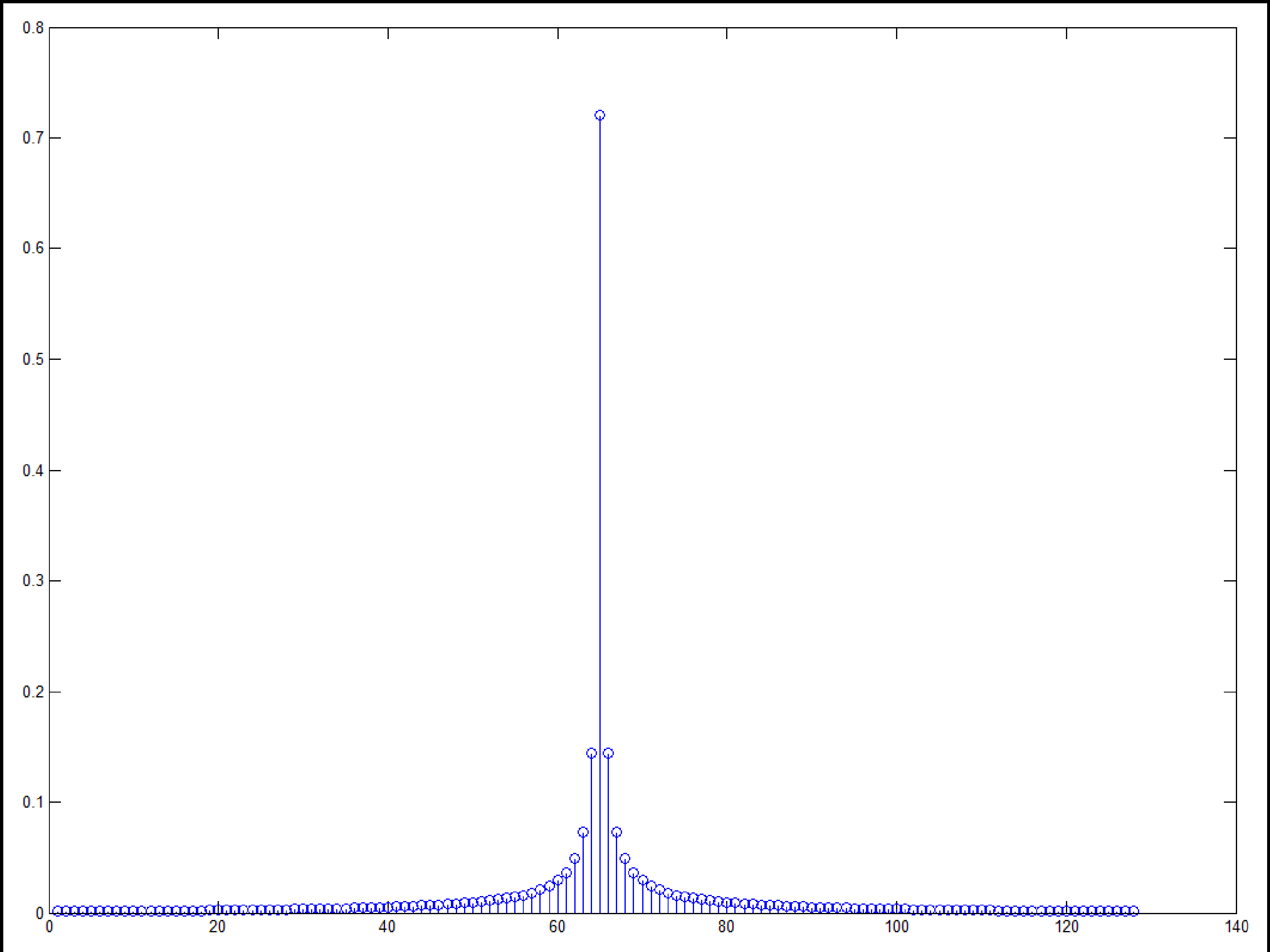
In MATLAB the function that does this is called fftshift().

Note there is also a function ifftshift() which does the reverse.

Before fftshift()



After fftshift()



One last thought

When you use fft() you need to normalize the output.

In the case of the this lab where you are finding the DFT of a sampled rectangle function with amplitude one you should divide the result of fft() by N.

```
y=fft(x)/N;
```

Parseval ->
$$\frac{1}{N}\sum_{n}^{Period}|x[n]|^2 = \frac{1}{N^2}\sum_{k}^{Period}|X[k]|^2$$