Computing Transforms

Introduction

In this project you will use computing environments for computing Fourier and Laplace transforms. In a later part of the project you will learn how to use discrete Fourier transforms to determine the frequency content of signals.

You will keep track of your observations with your partner and submit the results on the last class day in report form. This project will count as your computer lab/project grade for the course. All work should be typed with double-spacing and 12 pt font. You will be expected to use correct English grammar and punctuation. This is a report and thus you will use proper sentence and paragraph formatting. You will be graded on the evidence of work, mathematical detail and understanding, proper exposition and neatness. Your work should also be supported with properly labeled and embedded plots and properly numbered and punctuated equations. Any references used should be cited as well. These reports will count towards the project component of your grade.

1 Symbolic MATLAB

For those not familiar with MATLAB, there are some additional help files listed at the project site `http://people.uncw.edu/hermanr/mat367/ProjectS16.htm`. You can access MATLAB through Tealware, at https://tealware.uncw.edu. MATLAB is under the Mathematics & Statistics folder, MATLAB2015a. Open the program. The m-files provided on the project web page, or later in this document, can either be saved to the default MATLAB directory, or the text can be copied and pasted into the editor and then saved with the original file name. Data and other files should be copied over to the same directory. You should see your files in the Current Folder panel on the left of the screen. [See the Figure 1.] You can add your own folder, but be sure to open the folder before running the m-file.

The m-files are provided which can be copied into an editor, saved with a **.m** extension into the working directory, and called in MATLAB by entering the file name (without the **.m**) in the Command Window. Many of these can also be run under GNU Octave. More about GNU Octave can be found in the next section.

In the next section we will investigate the use of symbolic computations in MATLAB to compute Fourier and Laplace transforms. In order to do this, we make use of the Symbolic Toolbox commands.

In order to declare variables as symbolic, you can enter the command

```
syms a b c x
```

Now enter the quadratic function,

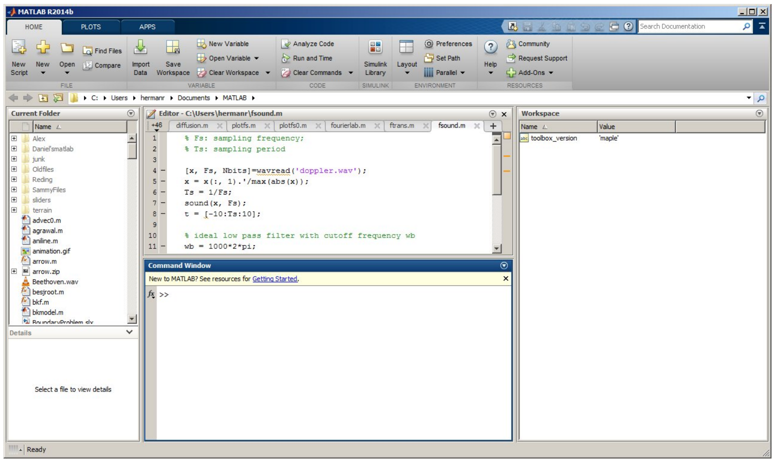

Figure 1: MATLAB workspace showing the Command window in the lower center area and the file structure on the left side.

```
f = a*x^2 + b*x + c;
```

You can now differentiate and integrate f with respect to x:

```
diff(x,x)
int(f,x)
```

You can even solve $ax^2 + bx + c = 0$ by

```
solve(f,x)
```

The output is

```
ans =

 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

You would need to write the answer in standard mathematical form and not in this output format. That means not using *, such as

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Before proceeding, you should clear the MATLAB workspace by entering

```
clear
```

Fourier and Laplace Transforms

In the Command window you can also enter MATLAB commands one line at a time or you can save several lines of code to an m-file to be run separately. For example, enter the line

```
syms t w
```

to say that **t** and **w** are symbolic variables. Then enter the function $f(t) = e^{-|t|}$ as

```
f=exp(-abs(t))
```

Now, compute the Fourier transform:

```
F=fourier(f);
```

Here MATLAB computes

```
F(w) = c*int(f(x)*exp(s*i*w*x),x,-inf,inf)
```

The constant c can be set, but you need not d that. It automatically gives a function of ω.

The inverse Fourier transform can be computed as

```
ifourier(2/(w^2 + 1))
```

If you want to do more, like plotting, you can use the **ezplot** command to plot symbolic functions. A window is provided by inserting [xmin xmax ymin ymax] as an argument.

The plot of $f(t)$ is obtained using

```
ezplot(f,[-10,10,0,1 ])
```

or a plot of the Fourier transform is found using

```
ezplot(abs(F),[-20,20,0,2])
```

Next we use these commands to build an m-file and embellish the plots. Type, or copy, the text in the code below in the Editor. Save the file as **ftrans.m** to the working directory. Then all you need to type is **ftrans** in the Command window. [Note: Whenever copying and pasting code with quotes, you might need to retype the quotes.]

- **ftrans.m**

  In the following code the function **fourier** computes the Fourier transform symbolically. The rest of the code is used to plot $f(t) = e^{-|t|}$ and its transform, $F = \dfrac{2}{\omega^2 + 1}$. The result is shown in Figure 2.

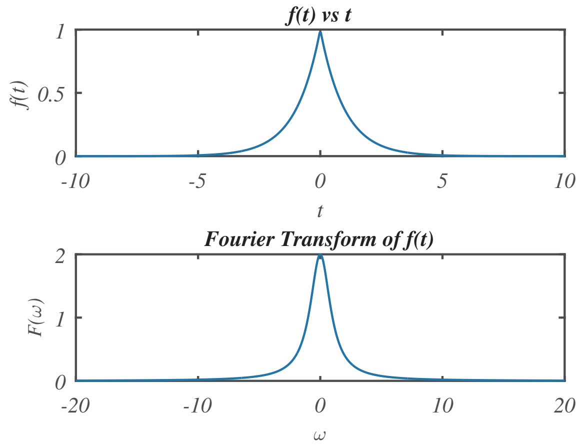

Figure 2: Plot of $f(t) = e^{-|t|}$ and its transform.

```
clear
syms t w

f=exp(-abs(t));
F=fourier(f);

figure(1)
subplot(2,1,1)
ezplot(f,[-10,10,0,1 ])
xlabel('t')
ylabel('f(t)')
title(['f(t) vs t'] )

subplot(2,1,2)
ezplot(abs(F),[-20,20,0,2])
xlabel('\omega')
ylabel('F(\omega)')
title(['Fourier Transform of f(t)'] )
```

One can also perform Laplace transforms,

```
syms x y
f = 1/sqrt(x);
laplace(f, x, y)
```

and inverse Laplace transforms

```
syms x y
F = 1/y^2;
ilaplace(F, y, x)
```

Convolution

Recall, the convolution is defined as

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\xi) g(x-\xi)\, d\xi. \tag{1}$$

We can compute the convolution directly by defining the functions $f(x)$ and $g(x)$ and carrying out the integration in MATLAB.

Consider the text example. We compute the convolution of the box function

$$f(x) = \begin{cases} 1, & |x| \le 1, \\ 0, & |x| > 1, \end{cases}$$

and the half-triangular function

$$g(x) = \begin{cases} x, & 0 \le x \le 1, \\ 0, & \text{otherwise}, \end{cases}$$

as shown in Figure 3.

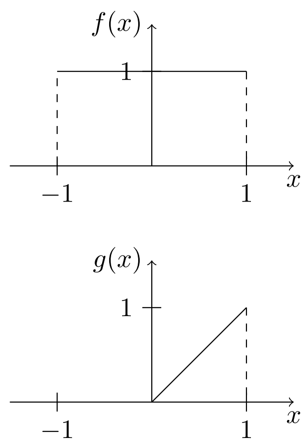

Figure 3: A plot of the box function $f(x)$ and the triangle function $g(x)$.

We begin by defining these functions and then computing the convolution. We have to make use of the Heaviside step function,

$$H(x) = \begin{cases} 0, & x < 0, \\ 1, & x > 0. \end{cases} \tag{2}$$

The Heaviside function can be used to create piecewise-defined functions. Thus, the box function is given by

$$f(x) = H(x+1) - H(x-1)$$

and the half-triangular function is

$$g(x) = x(H(x) - H(x-1)).$$

Now, we turn to MATLAB. Enter the lines

```
clear
syms x xi
```

Next, enter the functions

```
f=(heaviside(x+1)-heaviside(x-1));
g=x*(heaviside(x)-heaviside(x-1));
```

In order to compute the convolution, as given in Equation (1), we need to write f and g as functions of the integration variable. We do this with the next lines.

```
g2=subs(g,x,x-xi);
f2=subs(f,x,xi);
```

Here we replaced the symbolic variable, x, with either ξ or $x - \xi$.

We are now ready to carry out the integration.

```
h=int(f2*g2,xi,[-10 10]);
```

Typing **h** in the Command window, we obtain the convolution in the form

```
piecewise([10 <= x, 0], [x <= 10, (heaviside(x + 1)*(x + 1)^2)/2
    - (heaviside(x - 1)*(x - 1)^2)/2 - (x*heaviside(x)*(x + 2))/2
    + (x*heaviside(x - 2)*(x - 2))/2])
```

This is a bit unweildy. So, a plot would be nice. Below we provide an m-file which carries out the above convolution and plots the results. The plots are shown in Figure 4. Note that the plot of the convolution agrees with the text example.

There are commented out lines, denoted with the %. One can remove the % to try other convolutions, such as convolving a function with itself or using the full-triangular function that is provided.

- **myconv.m**

```
% Compute the convolution of f and g

clear
close all
syms x xi

% Enter f and g
f=(heaviside(x+1)-heaviside(x-1)); % box function
g=x*(heaviside(x)-heaviside(x-1)); % half triangle

% full triangle
```

```
%g=(x+1)*(heaviside(x+1)-heaviside(x))+(1-x)*(heaviside(x)-heaviside(x-1));

%g=f;
%f=g;

% Compute convolution integral
g2=subs(g,x,x-xi);          % g(x-xi)
f2=subs(f,x,xi);            % f(xi)
h=int(f2*g2,xi,[-10 10]);  % Set large integration range

% Set x values for plotting in t variable
dt = 0.01;
t = [ -5:dt:5 ];

% Change symbolic functions to vectors for plotting
f1=subs(f,x,t);
f1=double(f1);
g1=subs(g,x,t);
g1=double(g1);

%Plot functions
figure(1)
subplot(3,1,1)
plot(t,f1)
xlabel('x')
ylabel('f(x)')
title('f(x) vs x')

subplot(3,1,2)
plot(t,g1)
xlabel('x')
ylabel('g(x)')
title('g(x) vs x')

h1=subs(h,x,t);
h1=double(h1);
subplot(3,1,3)
plot(t,h1)
axis([-5 5 0 1])
xlabel('x')
ylabel('(f * g)(x)')
title('(f * g)(x) vs x')
```

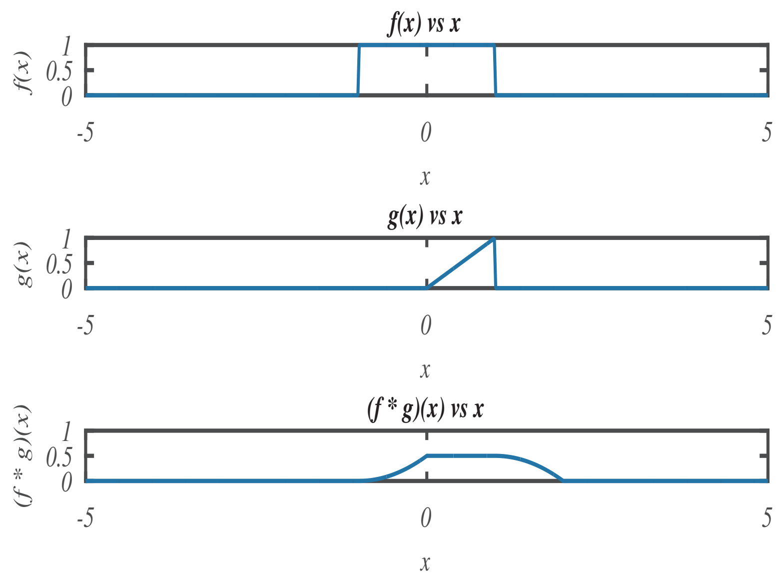

Figure 4: Plots of the box and half-triangular functions and the convolution.

2 Problems

Here are some problems to try in MATLAB. For more complicated output, try using the **simplify** command.

1. For each function, find the Fourier transform and plot the function and the magnitude of its transform. Make sure to plot on large enough interval to capture the function.

a. $f(t) = t^2 e^{-|t|}$.

b. $f(t) = e^{-t^2+2t}$.

c. $f(t) = H(t+1) - H(t-1)$.

d. $f(t) = (t+1)(H(t+1) - H(t)) + (1-t)(H(t) - H(t-1))$.

e. $f(t) = (H(t+\pi) - H(t-\pi)) \cos 5\pi t$.

2. For each function, find the inverse Fourier transform.

a. $\hat{f}(\omega) = \dfrac{2}{\omega^2 + 9}$.

b. $\hat{f}(\omega) = \dfrac{1}{4 + i\omega^2}$.

c. $\hat{f}(\omega) = \dfrac{4}{(\omega^2 + 4)^2}$.

3. Consider the function $f(t)$ and its Fourier transform, $\hat{f}(\omega)$. Establish an uncertainty principle between the spreads of the function and its transform: $\Delta t \Delta \omega = \text{const}$. Define the width of a function as the Full Width at Half Maximum (FWHM).

a. $f(t) = e^{-|t|}$

b. $f(t) = e^{-at^2}$, $a > 0$.

4. Find the Laplace transform of the following functions:

a. $f(t) = t^2 - 5t + 1$.

b. $f(t) = e^{3t-4}$.

c. $f(t) = e^{3t} \sin 2t$.

d. $f(t) = t^2 H(t-1)$.

e. $f(t) = \int_0^t (t-u)^2 \sin u \, du$.

5. Find the inverse Laplace transform of the following functions.

a. $F(s) = \frac{9}{s^4} + \frac{7}{s^3}$.

b. $F(s) = \frac{s+1}{s^2+1}$.

c. $F(s) = \frac{3}{s^2+2s+2}$.

d. $F(s) = \frac{1}{(s-1)^2}$.

e. $F(s) = \frac{e^{-3s}}{s^2-1}$.

f. $F(s) = \frac{1}{s^2+4s-5}$.

6. Consider the following functions:

- the box function, $f(x) = H(x+1) - H(x-1)$
- the full triangular function, $h(x) = (1+x)(H(x+1) - H(x)) + (1-x)(H(x) - H(x-1))$.

Find and plot the convolution of the following pairs of functions,

a. Box function with itself, $(f * f)(x)$.

b. Box function with full triangle function, $(f * h)(x)$.

b. The full triangle function with itself, $(h * h)(x)$.

3 Other Software

MATLAB is not the only software useful for carrying out these computations. We could use Maple, but sometimes open source alternatives are better. In this section we will discuss the use of GNU Octave and Python.

GNU Octave

GNU Octave is a MATLAB clone which is free and open. It can be obtained from `https://www.gnu.org/software/octave/`. Many of the MATLAB m-files can be run in GNU Octave with little change. However, the symbolic package has to be installed.

Open GNU Octave and type

```
pkg install -forge symbolic
```

This need only been done one time.

Before running any m-files containing symbolic computation, you have to invoke the command

```
pkg load symbolic
```

Python

One can use Python to carry out similar computations as one can do in MATLAB. You can either install a Python interface like Anaconda at `https://www.continuum.io/downloads` or use online Python sites such as found at `https://www.getdatajoy.com` or `http://live.sympy.org/`.

- **ftrans.py**

```
from sympy import *
x, k = symbols('x k')
F=fourier_transform(exp(-x**2), x, k)
print F
plot(F, (k, -1, 1), ylim=(0, 2))
```

- **ltrans.py**

```
from sympy import *
t, s = symbols('t s')
Y=laplace_transform(t**2, t, s)
print Y
```

4 Simulink

Simulink is a graphical environment for designing simulations of systems. When you have access to Simulink and MATLAB, you can start MATLAB and on the icon bar there is an icon that you can click to launch Simulink. Alternatively, you can type to bring up the Simulink Library Browser as shown in Figure 5. Next, click the yellow plus to bring up a new model. We build models by dragging and connecting the needed components, or blocks, from groups such as the Continuous, Math Operations, Sinks, or Sources.

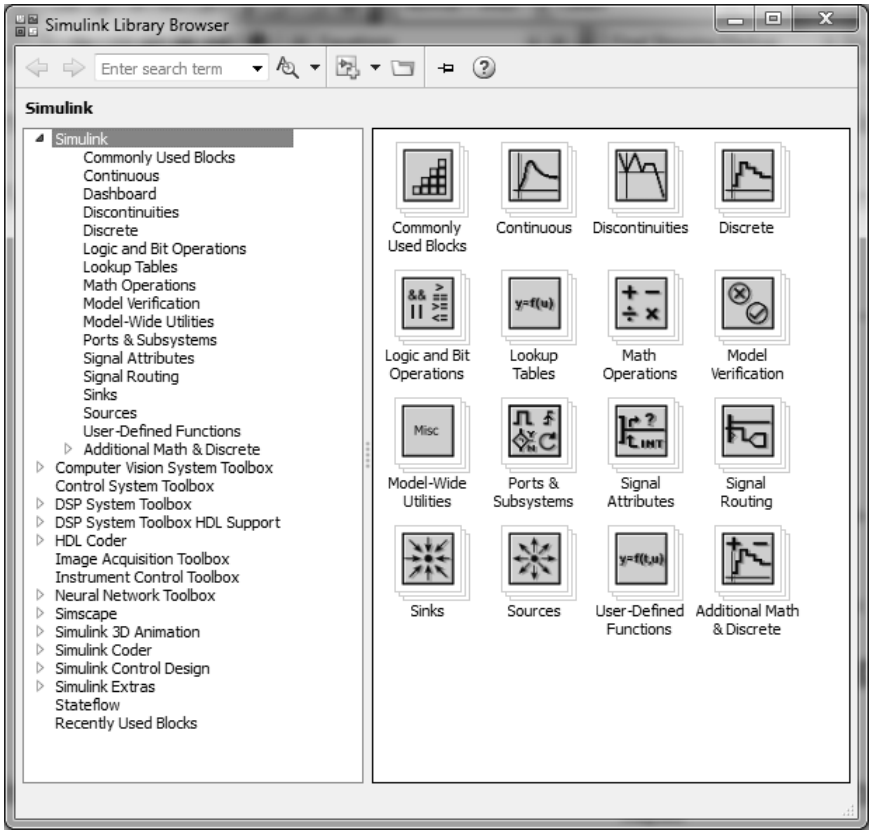

Figure 5: The Simulink Library Browser. This is where various blocks can be found for constructing models. [As seen in MATLAB 2015a.]

5 Discrete Fourier Transform

The goal of this part of the project is to use MATLAB to investigate the use of the Discrete Fourier Transform (DFT) for the spectral analysis of time series. Several files will be needed in this project. They are provided on the project webpage.

MATLAB for the Discrete Fourier Transform

In this section we provide implementations of the discrete trigonometric transform in MATLAB. The first implementation is a straightforward one which can be done in most programming languages. The second implementation makes use of matrix computations that can be performed in MATLAB or similar programs like GNU Octave or Pylab. Sums can be done with matrix multiplication, as described in the next section. This eliminates the loops in the first program below and speeds up the computation for large data sets.

Direct Implementation for a data set

The following code was used to produce results for a data set. It shows a direct implementation using loops to compute the trigonometric DFT as developed in this chapter. The data is entered in vector **y**. The Fourier coefficients are entered using the matrix capabilities of MATLAB as described in the next section. The signal is then reconstructed using the finite series representation. Plots are provided to show this implementations as demonstrated in earlier examples.

```
%
% DFT in a direct implementation
%
% Enter Data in y
y=[7.6 7.4 8.2 9.2 10.2 11.5 12.4 13.4 13.7 11.8 10.1 ...
    9.0 8.9 9.5 10.6 11.4 12.9 12.7 13.9 14.2 13.5 11.4 10.9 8.1];
% Get length of data vector or number of samples
N=length(y);
% Compute Fourier Coefficients
for p=1:N/2+1
    A(p)=0;
    B(p)=0;
    for n=1:N
        A(p)=A(p)+2/N*y(n)*cos(2*pi*(p-1)*n/N)';
        B(p)=B(p)+2/N*y(n)*sin(2*pi*(p-1)*n/N)';
    end
end
A(N/2+1)=A(N/2+1)/2;
% Reconstruct Signal - pmax is number of frequencies used
% in increasing order
pmax=13; for n=1:N
    ynew(n)=A(1)/2;
    for p=2:pmax
        ynew(n)=ynew(n)+A(p)*cos(2*pi*(p-1)*n/N)+B(p) ...
                *sin(2*pi*(p-1)*n/N);
    end
end
% Plot Data
plot(y,'o')
% Plot reconstruction over data
hold on
plot(ynew,'r')
hold off
title('Reconstruction of Monthly
Mean Surface Temperature')
xlabel('Month')
ylabel('Temperature')
```

Direct Implementation for a function.

The next routine shows how we can determine the spectral content of a signal, given in this case by a function and not a measured time series. The output is the original data and reconstructed Fourier series in Figure 1, the trigonometric DFT coefficients in Figure 2, and the the power spectrum in Figure 3. This code will be referred to as **ftex.m**.

```
% ftex.m
% IMPLEMENTATION OF DFT USING TRIGONOMETRIC FORM
% N = Number of samples
% T = Record length in time
% y = Sampled signal
%
clear
N=128;
T=5;
dt=T/N;
t=(1:N)*dt;
f0=30;
y=sin(2*pi*f0*t);

% Compute arguments of trigonometric functions
for n=1:N
    for p=0:N/2
        Phi(p+1,n)=2*pi*p*n/N;
    end
end

% Compute Fourier Coefficients
for p=1:N/2+1
    A(p)=2/N*y*cos(Phi(p,:))';
    B(p)=2/N*y*sin(Phi(p,:))';
end
A(1)=2/N*sum(y);
A(N/2+1)=A(N/2+1)/2;
B(N/2+1)=0;

% Reconstruct Signal - pmax is number of frequencies
% used in increasing order
pmax=N/2;
for n=1:N
    ynew(n)=A(1)/2;
    for p=2:pmax
        ynew(n)=ynew(n)+A(p)*cos(Phi(p,n))+B(p)*sin(Phi(p,n));
    end
end

% Plot Data
figure(1)
plot(t,y,'o')
```

```
% Plot reconstruction over data
hold on
plot(t,ynew,'r')
xlabel('Time')
ylabel('Signal Height')
title('Reconstructed Signal')
hold off

% Compute Frequencies
n2=N/2;
f=(0:n2)/(n2*2*dt);

% Plot Fourier Coefficients
figure(2)
subplot(2,1,1)
stem(f,A)
xlabel('Frequency')
ylabel('Amplitude')
title('A')
subplot(2,1,2)
stem(f,B)
xlabel('Frequency')
ylabel('Amplitude')
title('B')

% Plot Fourier Spectrum
figure(3)
Power=sqrt(A.^2+B.^2);
stem(f,Power(1:n2+1))
xlabel('Frequency')
ylabel('Power')
title('Periodogram')

% Show Figure 1
figure(1)
```

Compact Implementation - without loops.

The next implementation uses matrix products to eliminate the for loops in the previous code. The way this works is described in the next section.

```
%
% DFT in a compact implementation
%
% Enter Data in y
y=[7.6 7.4 8.2 9.2 10.2 11.5 12.4 13.4 13.7 11.8 ...
```

```
    10.1 9.0 8.9 9.5 10.6 11.4 12.9 12.7 13.9 14.2 ...
    13.5 11.4 10.9 8.1];
N=length(y);

% Compute the matrices of trigonometric functions
p=1:N/2+1;
n=1:N;
C=cos(2*pi*n'*(p-1)/N);
S=sin(2*pi*n'*(p-1)/N);

% Compute Fourier Coefficients
A=2/N*y*C;
B=2/N*y*S;
A(N/2+1)=A(N/2+1)/2;

% Reconstruct Signal - pmax is number of frequencies used
% in increasing order
pmax=13;
ynew=A(1)/2+C(:,2:pmax)*A(2:pmax)'+S(:,2:pmax)*B(2:pmax)';

% Plot Data
plot(y,'o')

% Plot reconstruction over data
hold on
plot(ynew,'r')
hold off
title('Reconstruction of Monthly
Mean Surface Temperature')
xlabel('Month')
ylabel('Temperature')
```

MATLAB Implementation of FFT

In this section we provide implementations of the Fast Fourier Transform in MATLAB. The MATLAB code provided in Section 5 can be simplified by using the built-in **fft** function. The following code employs MATLAB's built-in **fft** function to replace the direct implementation code.

Implementation of the fft function.

```
% fanal.m
%
% Analysis Using FFT
%
clear
n=128;
```

```
T=5;
dt=T/n;
f0=6.2;
f1=10;
y=sin(2*pi*f0*(1:n)*dt)+2*sin(2*pi*f1*(1:n)*dt);
% y=exp(-0*(1:n)*dt).*sin(2*pi*f0*(1:n)*dt);
Y=fft(y,n);
n2=n/2;
Power=Y.*conj(Y)/n^2;
f=(0:n2)/(n2*2*dt);
stem(f,2*Power(1:n2+1))
xlabel('Frequency')
ylabel('Power')
title('Periodogram')
figure(2)
subplot(2,1,1)
stem(f,real(Y(1:n2+1)))
xlabel('Frequency')
ylabel('Amplitude')
title('A')
subplot(2,1,2)
stem(f,imag(Y(1:n2+1)))
xlabel('Frequency')
ylabel('Amplitude')
title('B')
figure(1)
```

This code can be simplified even further by placing the FFT computation and plot functions into a function that accepts the signal as input. Below is the function **fanalf**, which can make the main program more compact.

Putting fft and plots into a stand alone function.

```
function z=fanalf(y,T)
%
% FFT Analysis Function
%
% Enter Data in y and record length T
% Example:
%    n=128;
%    T=5;
%    dt=T/n;
%    f0=6.2;
%    f1=10;
%    fanal(sin(2*pi*f0*(1:n)*dt)+2*sin(2*pi*f1*(1:n)*dt));
% or
```

```
%    fanal(sin(2*pi*6.2*(1:128)/128*5),5);
n=length(y);
dt=T/n;
Y=fft(y,n);
n2=floor(n/2);
Power=Y.*conj(Y)/n^2;
f=(0:n2)/(n2*2*dt);
z=Power;
stem(f,2*Power(1:n2+1))
xlabel('Frequency')
ylabel('Power')
title('Periodogram')
```

Examples of the use of this function for doing a spectral analysis of functions, data sets, and sound files are provided below. This code snippet can be used with **fanalf** to analyze a given function.

Code for analysis of functions.

```
% fanal2.m

clear
n=128;
T=5;
dt=T/n;
f0=6.2;
f1=10;
t=(1:n)*dt;
y=sin(2*pi*f0*t)+2*sin(2*pi*f1*t);
fanalf(y,T);
```

The code snippet that can be used with **fanalf** to analyze a data set inserted in vector **y** is given by the next set of code.

Code for analysis of typed data values.

```
y=[7.6 7.4 8.2 9.2 10.2 11.5 12.4 13.4 13.7 11.8 ...
    10.1 9.0 8.9 9.5 10.6 11.4 12.9 12.7 13.9 14.2 ...
    13.5 11.4 10.9 8.1];
n=length(y);
y=y-mean(y);
T=24;
dt=T/n;
fanalf(y,T);
```

One can also enter data stored in an ASCII file. The following code shows how this is done.

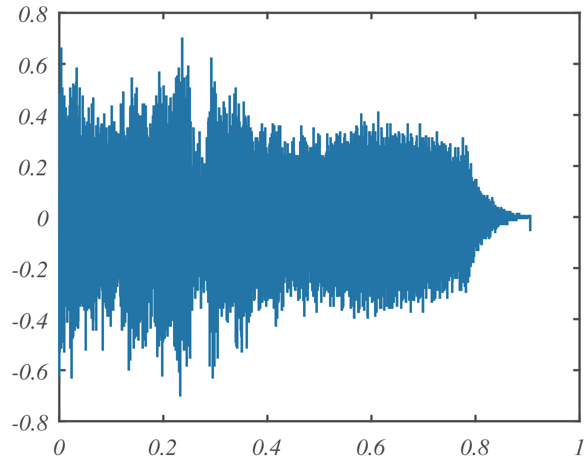

Figure 6: Typical signal from a sound file.

Code for analysis of data read from ASCII file.

```
[year,y]=textread('sunspot.txt','%d %f');
n=length(y);
t=year-year(1);
y=y-mean(y);
T=t(n);
dt=T/n;
fanalf(y',T);
```

The next code snippet can be used with **fanalf** to analyze a given sound stored in a wav file. As an example, Figure 6 shows the signal from the ringing of an old-style phone. The frequency content is displayed in Figure 7. We should note that the **waveread** function will be superceded by **audioread** in MATLAB.

Code for analysis of sound files.

```
[y,NS,NBITS]=wavread('sound1.wav');
n=length(y);
T=n/NS
dt=T/n;
figure(1)
plot(dt*(1:n),y)
figure(2)
fanalf(y,T);
```

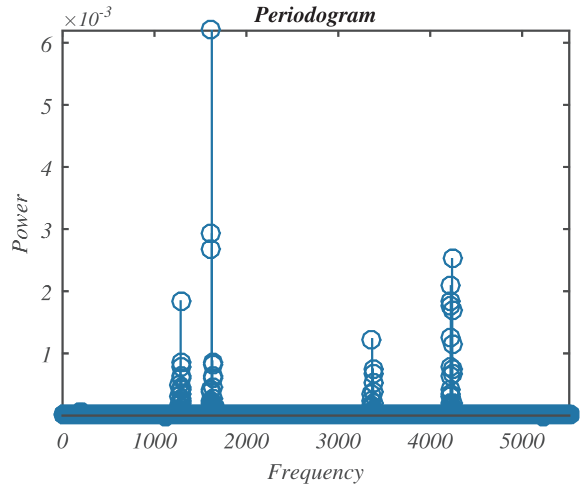

Figure 7: Spectral content of the sound in Figure 6.

Problems

1. The goal of this problem is to use MATLAB to investigate the use of the Discrete Fourier Transform (DFT) for the spectral analysis of time series. Running the m-files is done by typing the filename without the extension (.m) after the prompt (¿¿) in the Command Window, which is the middle panel of the MATLAB program. Note, most of the code provided can be run in GNU Octave.

1. Analysis of simple functions.
   a. File **ftex.m** - See the MATLAB section for the code.
      This file is a MATLAB program implementing the discrete Fourier transform using trigonometric functions like that derived in the text. The input is a function, sometimes with different frequencies. The output is a plot of the data points and the function fit, the Fourier coefficients and the periodogram giving the power spectrum.
      i. Save the file **ftex.m** in your working directory under MATLAB.
      ii. View the file by entering edit **ftex**. Note how the first function is defined by the variable **y**.
      iii. Run the file by typing **ftex** in MATLAB's Command window.
      iv. Change the parameters in **ftex.m**, remembering the original ones. In particular, change the number of points, **N**, (keeping them even), the frequency, **f0**, and the record length, **L**. Note the effects. If you get an error, enter clear and try again. Always save the m-file after making any changes before running ftex.
      v. Reset the parameters to the original values. What happens for frequencies of **f0** $= 5$, 5.5 and 5.6? Is this what you expected?
      vi. Repeat the last set of frequencies for double the record length, **T**. Is there a change?
      vii. Reset the parameters. Put in frequencies of **f0** $= 20, 30, 40$. What frequencies are present in the periodogram? Is this what you expected?
   b. Look at sums of several trigonometric functions.

i. Reset the parameters.

ii. Change the function in **y** to `y=sin(2*pi*f0*t)+sin(2*pi*f1*t);` and add a line defining **f1**. Start with **f1** $= 3$; and look at several other values for the two frequencies. Try different amplitudes; for example, `3*sin(2*pi*f0*t)` has an amplitude of 3. Record your observations.

iii. Change one of the sines to a cosine. What is the effect? What happens when the sine and cosine terms have the same frequency?

c. Investigate non-sinusoidal functions.

i. Investigate the following functions:

1. `y=t;`
2. `y=t.^2;`
3. `y=sin(2*pi*f0*(t-T/5))./(t-T/5);` What is this function?
4. `y(1,1:M)=ones(1,M); y(1,M+1:N)=zeros(1,N-M);` Start with $M = N/2$; What is this function? How are the last two problems related? Do they relate to anything from earlier class lectures? What effect results from changing **M**?
5. Try multiplying the function in 4 by a simple sinusoid; for example, add the line `y=sin(2*pi*f0*t).*y`, for $M = N/2$. How does this affect what you had gotten for the sinusoid without multiplication?

2. Use the FFT function.

In MATLAB there is a built in set of functions, **fft** and **ifft** for the computation of the Discrete Exponential Transform and its inverse using the Fast Fourier Transform (FFT). The files needed to do this are **fanal.m** or **fanal2.m** and **fanalf.m**. [See Section 5 for the code.] Put these codes into the MATLAB editor and see what they look like. Note that **fanal.m** was split into the two files **fanal2.m** and **fanalf.m**. This will allow us to be confident when we later create new data files and then using fanalf.m. Test this set of functions for simple sine functions to see that you get results similar to Part 1. First run **fanal** and then **fanal2**. Is there any difference between these last two approaches?

3. Analysis of data sets.

One often does not have a function to analyze. Some measurements are made over time of a certain quantity. This is called a time series. It could be a set of data describing things like the stock market fluctuations, sunspots activity, ocean wave heights, etc. Large sets of data can be read into y and small sets can be input as vectors. We will look at how this can be done. After the data is entered, one can analyze the time series to look for any periodic behavior, or its frequency content. In the two cases below, make sure you look at the original time series using **plot(y,+)**.

a. Ocean Waves

In this example the data consists of monthly mean sea surface temperatures (^{o}C) at one point over a 2 year period. The temperatures are placed in **y** as a row vector. Note how the data is continued to a second line through the use of ellipsis. Also, one typically subtracts the average from the data. What affect should this have on the spectrum? Copy the following code into a new m-file called **fdata.m** and run **fdata**. Determine the dominant periods in the monthly mean sea surface temperature.

```
y=[7.6 7.4 8.2 9.2 10.2 11.5 12.4 13.4 13.7 11.8 ...
    10.1 9.0 8.9 9.5 10.6 11.4 12.9 12.7 13.9 14.2 ...
    13.5 11.4 10.9 8.1];
```

```
n=length(y);
y=y-mean(y);
T=24;
dt=T/n;
fanalf(y,T);
```

b. Sunspots

Sunspot data was exported to a text file. Download the file sunspot.txt. You can open the data in Notepad and see the two column format. The times are given as years (like 1850). This example shows how a time series can be read and analyzed. From your spectrum, determine the major period of sunspot activity. Note that we first subtracted the average so as not to have a spike at zero frequency. Copy and paste into the editor and save as **fanaltxt.m**. Note: Copy and Paste of single quotes often does not work correctly. Retype the single quotes after copying.

```
[year,y]=textread('sunspot.txt','%d %f');
n=length(y);
t=year-year(1);
y=y-mean(y);
T=t(n);
dt=T/n;
fanalf(y',T);
```

4. Analysis of sounds. Sounds can be input into MATLAB. You can create your own sounds in MATLAB or sound editing programs like Audacity or Goldwave to create audio files. These files can be input into MATLAB for analysis.

   Save the following sample code as **fanalwav.m** and save the first sound file. This code shows how one can read in a WAV file. There will be two plots, the first showing the wave profile and the second giving the spectrum. Try some of the other wav files and report your findings. Note: Copy and Paste of single quotes often does not work correctly. Retype the single quotes after copying.

```
[y,NS,NBITS]=wavread('sound1.wav');
n=length(y);
T=n/NS
dt=T/n;
figure(1)
plot(dt*(1:n),y)
figure(2)
fanalf(y,T);
```

   Several wav files are be provided for you to analyze. If you are able to hear the sounds, you can run them from MATLAB by typing `sound(y,NS)`. In fact, you can even create your own sounds based upon simple functions and save them as wav files. For example, try the following code: Note: Copy and Paste of single quotes often does not work correctly. Retype the single quotes after copying.

```
smp=11025;
t=(1:2000)/smp;
y=0.75*sin(2*pi*440*t);
sound(y,smp,8);
wavwrite(y,smp,8, 'myfile.wav');
```

Try some other functions, using several frequencies. If you get a clipping error, then reduce the amplitudes you are using.