Math 307: Problems for section 3.5

1. **Compute the discrete Fourier transform of** $\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$ **using (i) the matrix product (i.e., using the** F **matrix from the notes) and (ii) the fast Fourier transform (using a butterfly diagram).**

   Let's do the matrix product first, either by hand or with MATLAB/Octave.

```
> F=[1 1 1 1;1 -i -1 i; 1 -1 1 -1;1 i -1 -i];
> F*[1 -1 1 -1]'

ans =

   0
   0
   4
   0
```

   Therefore we find $c_0 = 0$, $c_1 = 0$, $c_2 = 4/4 = 1$ and $c_3 = 0$.

   Doing the FFT by hand using the butterfly diagram, we first write the vector in bit reversed order and then combine the entries according to the diagram we derived in lectures. The bit-reversed form is $\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$. Now the butterfly diagram is

   Let's check this calculation.

```
> fft([1 -1 1 -1])/4

ans =

   0 + 0i   0 + 0i   1 + 0i   0 - 0i
```

2. **Compute the discrete Fourier transform of** $\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}$ **using a butterfly diagram.**

Since all the entries are the same, we don't have to bother with the bit-reversing. Now the butterfly diagram is:

Here $\overline{\omega}_8 = e^{-2\pi i/8} = (1-i)/\sqrt{2}$. We have $c_0 = 8/8 = 1$ and all the remaining $c_i = 0$.
We can check this with

```
> fft([1 1 1 1 1 1 1 1])/8

ans =

   1 + 0i   0 + 0i   0 + 0i   0 + 0i   0 + 0i   0 - 0i   0 - 0i   0 - 0i
```

Being a bit more clever about it, we could also notice that this vector is one of the rows of the matrix for the discrete Fourier transform. Since this matrix is unitary, the inner product with the other rows must be zero. (A bit of care is required, since it's the complex inner product that is zero. But since the vector is completely real, it doesn't make any difference.)

3. **The file `dutch_harbor_tides.m` contains the tidal height in millimetres at Dutch Harbor, Alaska. The data, at hourly intervals from 1985 to 2005, is contained in the file `dutch_harbor_tides.m`. (If you download this file, and then type `dutch_harbor_tides` at the MATLAB/Octave prompt `y` will be defined to have these values.) Using days as the unit of time, hand in a frequency amplitude plot for the range 0.5days^{-1} to 3.0days^{-1} and a list of the MATLAB/Octave commands you used to find it. Does Dutch Harbor have a predominantly twice-daily or daily tide?**

   Here is the code to find the frequency, amplitude plot (with `y` as given in `dutch_harbor_tides.m`).

```
dutch_harbor_tides; % Defines y

Fs = 24; % Sampling frequency is 24 samples/day
N  = length(y); % Length of signal

omega=[0:N-1]*Fs/N; % the frequencies
ampl=abs(fft(y))/N; % the amplitudes

% Plot single-sided amplitude spectrum.
Nmax=N/4;
plot(omega(1:Nmax),ampl(1:Nmax))
axis([0.5 3 1 200])
```

   The plot we obtain for amplitude against frequency is the following:

We see that the dominant tidal frequencies are around 1 day^{-1}. In other words Dutch Harbor has a predominantly daily tide. We could perhaps also have seen this from plotting the tidal height against time, but the decomposition into Fourier series allows us to say quantitatively the amplitude of the different frequency tides.

4. **One way of representing information from an audio signal is using a spectrogram. A spectrogram is a representation of the how the frequency spectrum of a signal varies over time. Usually it is a graph with time along one axis (horizontal) and frequency along the other (vertical). The magnitude of each frequency component at a given point of time is then represented by the colour or shading of each point on the graph.**

   Spectrograms can be created by using a short-time Fourier transform. A short-time Fourier transform determines the frequency components of a small section of the signal. Taking the Fourier transform of the entire signal will give the frequency components of the entire signal over all time. We, however, want to see how the frequency components of the signal change with time. To do this we break the signal up into smaller sections (these sections can overlap), and take the Fourier transform of each section. This essentially gives us the frequency components of the signal at a certain time (say the average time for that particular section of the signal). Keeping track of the frequency components for each section we can build a matrix that gives the Fourier coefficient for each frequency at a specific time. Then plotting the magnitude of these Fourier coefficients versus time and frequency we get a spectrogram for the signal.

   When we break the signal into smaller sections the size of these sections (referred to as the window size) determines how well you can resolve certain features. For a small window size details in time are more easily resolved but the frequency resolution is poor, whereas for a large window the opposite is true. Below are two spectrograms for the signal

$$y(t) = \begin{cases} \sin(2\pi(20t)) + \sin(2\pi(50t)) & 0 \le t < 5 \\ \sin(2\pi(20t)) + \sin(2\pi(100t)) & 5 \le t < 10 \\ \sin(2\pi(20t)) + \sin(2\pi(150t)) & 10 \le t < 15 \\ \sin(2\pi(20t)) + \sin(2\pi(200t)) & 15 \le t < 20, \end{cases} \tag{1}$$

   where t is measured in seconds. The signal is sampled with a sampling frequency of $F_s = 1000$. The $0.1\,s$ window allows us to clearly determine when the frequency of the signal changes, but not to clearly identify the frequencies in the signal. The $1\,s$ window clearly identifies the signal frequencies, but the time when the frequencies change is less clear. Notice that on the figure for the $1\,s$ window there is no data near $t = 0$ (and also near $t = 20$). This is because the first window looks at the signal from time $t = 0$ to $t = 1$ and so the Fourier coefficients calculated correspond to time $t = 0.5$. So there is no data that actually corresponds to $t = 0$. The same is true at $t = 20$. This also occurs in the figure for the $0.1\,s$ window (where the first set of coefficients corresponds to $t = 0.05$), however, $t = 0.05$ is so close to $t = 0$ that it is hard to notice.

Write a MATLAB/Octave function `spectrogram(y,Fs,TW,M)` **that takes as input a signal** `y`**, the sampling frequency** `Fs` **(samples/second), the window size** `TW` **(in seconds) and the number** `M` **of windows, and produces the spectrogram associated to this data. Start with the following skeleton and fill in the required code. In several places when computing the value of an index it is convenient to use the** `floor(x)` **function which returns the closest integer below** `x`**. The following code fragments are contained in the file** `spectrogramhints.txt` **which should be completed and renamed** `spectrogram.m`**.**

```
function spectrogram(y,Fs,TW,M)

% y is the sampled signal
% Fs is the sampling frequency
% TW is the window size in seconds
% M is the number of windows

% Calculate
% N: the length of y
% T: the time length of the signal
% NW: the number of samples per window

N =
T =
NW =

% Calculate
% NWmax: the number of frequencies to be plotted (1/4 of NW)
% Omega: the vector of frequencies

NWmax =
Omega =

% Calculate
% Times: a vector containing the times for the centre of each window
% C: a matrix containing the amplitudes for each window in its columns
% First initialize them to be zero

Times=zeros(1,M);
C=zeros(NWmax,M);

% Then use a for loop to fill in the values
```

```
for i=1:M
jmin = %put the lower index for the ith window here
jmax = %put the upper index for the ith window here
Times(i) = %put the time corresponding to the centre of the ith window here
tildec = %compute the fft of the part of the signal between jmin and jmax
C(:,i) = %set the ith column to be the first NWmax absolute values of tildec
end

% the following commands make the plot

pcolor(Times, Omega, C);
shading interp;
colormap(1-gray);

end
```

You can test your code with the file `testsig.mat` which contains the a signal of the function (1) sampled at $F_s = 1000$ (the command `load testsig.mat` will load the signal vector `y`, and the sampling frequency `Fs`). You should be able to produce spectrograms which look similar to the figures above.

Spectrograms are often used with marine sonar data. The command `load shipx10.mat` will load the signal vector `y`, and the sampling frequency `Fs` for actual sonar audio of a passing ship. (The same data is in the file `shipx10.wav`. You can listen to the audio if you like.) The audio has be speeded up by a factor of 10. Using your code, generate a spectrogram for this signal. Try and find an intermediate window size where both time and frequency are reasonably resolved. (If things looks streaky in one direction the signal is probably not well resolved in that direction.) Hand in a printout of the spectrogram as well as the commands you used to generate it.

Different ships have a different frequency "signatures", and so different ships can be distinguished from their sonar spectrograms. Also, the movement of the ship generates the curved shape of the different frequency components in time. The specific shape of the curve can be used to determine the velocity or location of the ship.

The command `load Bluex10.mat` will load the signal vector `y`, and the sampling frequency `Fs` for actual sonar audio of a Blue Whale. (The same data is in the file `Bluex10.au`. You can listen to this one too.) Again, the audio has be speeded up by a factor of 10 to make it better audible to humans. Different types of whales and even different whales of the same species can be distinguished by their spectrograms. For instance, Atlantic Blue Whales usually make a series of almost identical moans, whereas Pacific Blue Whales produce a higher frequency "trill" followed by three lower frequency moans. Use your code to generate a spectrogram for the sonar signal. Was this signal produced from Atlantic or Pacific Blue Whales? Also hand in a printout of the spectrogram and the commands you used to generate it.

Here is the completed code for the file `spectrogram.m`.

```
function spectrogram(y,Fs,TW,M)

% y is the sampled signal
% Fs is the sampling frequency
% TW is the window size in seconds
```

```
% M is the number of windows

% Calculate
% N: the length of y
% T: the time length of the signal
% NW: the number of samples per window

N = length(y);
T = N/Fs;
NW = floor(Fs*TW);

% Calculate
% NWmax: the number of frequencies to be plotted (1/4 of NW)
% Omega: the vector of frequencies

NWmax = floor(NW/4);
Omega = [0:NWmax-1]*Fs/NW;

% Calculate
% Times: a vector containing the times for the centre of each window
% C: a matrix containing the amplitudes for each window in its columns

Times=zeros(1,M);
C=zeros(NWmax,M);

for i=1:M
jmin = 1+floor((i-1)*(N-NW-1)/(M-1));
jmax = jmin+NW;
Times(i) = (jmin+NW/2)/Fs;
tildec = fft(y(jmin:jmax));
C(:,i) = abs(tildec(1:NWmax));
end

pcolor(Times, Omega, C);
shading interp;
colormap(1-gray);

end
```

After the command `load testsig.mat` we can compute the total length of the signal in seconds with `length(y)/Fs`. The result is 20 seconds. So we could try 200 windows of 0.1 seconds each, or perhaps 40 windows of 0.5 seconds each. The code

```
spectrogram(y,Fs,0.1,200)
spectrogram(y,Fs,0.5,40)
```

result in the following two plots

After the command `load shipx10.mat` we can determine, as above, that the total length of the signal is 91 seconds. So it is reasonable to try

```
spectrogram(y,Fs,1,200)
```

resulting in

After the command `load Bluex10.mat` we can determine, as above, that the total length of the signal is 19 seconds. So it is reasonable to try

```
spectrogram(y,Fs,1,30)
```

resulting in

The trill is clearly visible at the beginning of the signal, so this is a Pacific Blue whale.