Sheet 7

Create the directory sheet07 and make it your working directory.

Example 1. Go to my homepage and save the files data16.csv and compress.m to your working directory. You are going to use the DFT to compress the signal in the data file. Load the data (see sheet 2, Example 1 if you don't remember how). Assign `t` to the first row (time) and `y` (signal) to the second. We will take r to be the compression rate—it will be a number between 0 and 1 and the compression will be $100r\%$. For example, for $r = .7$ the compression is 70% and that means if your data set had 100 points, then you would be sending 30 points from which the original data would be reconstructed. Enter the following line:

```
r = .9;
```

Now take the DFT of y:

```
fy=fft(y);
```

Next, compress the DFT by throwing out the 100r% smallest coefficients.

```
fyc=compress(fy,r);
```

Invert the DFT to get the compressed signal (represented by `yc`):

```
yc=ifft(fyc);
```

Then plot the signal and the compressed signal on the same graph:

```
plot(t,real(yc),'r',t,y,'k','linewidth',2);
legend('Compressed','Signal')
```

Let's see exactly what the compression does to the DFT:

```
figure
plot(t,abs(fyc),'o',t,abs(fy),'r','linewidth',2)
legend('Compressed DFT','DFT')
```

The `figure` command tells Matlab to allow a new plot (try doing this without it and see what happens).

It is kind of hard to see what is going on near the $y-$axis, so zoom in on it using the `axis` command;

```
axis([0,7,0,25])
```

(Remember the first two numbers give the range for the $x-$axis and the second two are for the $y-$axis).

Finally compute the relative error and the compression:

```
fprintf('Relative Error is %f \n', norm(y-yc)/norm(y))
fprintf('Compression = %f percent\n',100*r)
```

Repeat this for other values of the compression rate r.

Example 2. Repeat this analysis for the signal in data17.csv. When you plot the signal and the compressed signal on the same graph, it's hard to tell if the compressed signal captured the spike. So include an `axis` command to zoom in on it.

Example 3. Save data13.csv to your working directory and load the data as usual. Plot it, and see it is noisy. We are going to use the DFT to filter out the noise. Since the noise is due to high frequency components, we will throw out the high frequencies in the Fourier expansion. This example with ba model for subsequent problems—you will modify it slightly. Take the DFT:

```
fy = fft(y);
```

Set the number of coefficients to retain:

```
n = 6;
```

Filter the DFT:

```
filtfy = [fy(1:n) zeros(1,length(fy)-2*n) fy(length(fy)-(n-1):length(fy))];
```

Invert the filtered DFT:

```
filtery = ifft(filtfy);
```

Plot the signal and the filtered signal on the same graph:

```
plot(t,y,'k',t,filtery,'r','linewidth', 2)
legend('Signal','Filtered Signal');
```

Finally, plot the DFT and the filtered DFT on the same graph to see what has happened to the coefficients—you will need to use an `axis` statement to zoom in on the $x-$axis:

```
figure
plot(t,abs(fy),'o',t,abs(filtfy),'r','linewidth',2)
legend('DFT','Filtered DFT')
```

See what happens as you change the value of n, the number of coefficients retained.

Example 4. Repeat this analysis for the signal in data18.csv.

Example 5. The signal in example 2 has a spike. Filter it out. Again, play around with n and plot the filtered and unfiltered DFT too. Ave your work as ex5.m

Example 6. Save the file noisydata1.csv to your working directory. Filter it as in the preceeding examples. Assuming the same varible names as above, to get a better picture of the noisy signal and the filtered signal on the same graph, replace the plotting lines

```
plot(t,y,'k',t,filtery,'r','linewidth', 2)
legend('Filtered Signal','Signal');
```

by the lines

```
hold on
plot(t,y,'k','linewidth', .5)
plot(t,filtery,'r','linewidth', 2)
legend('Noisy Signal','Filtered Signal')
hold off
```

This will cause the plot of the noisy data to be drawn with a fine line and the filtered signal with a heavy line, giving a better picture.

The noisy signal `y` was generated from the signal with no noise in the file no-noise-data1.csv. Load it and define the second row as the variable `y1`. Then plot the filtered signal on the same graph as the signal without noise so you can see how well the filtering worked with the particular choice of n you made above:

```
figure
plot(t,filtery,'r',t,y1)
legend('filtered','original-no noise')
```

Compute the relative error too:

```
fprintf('relative error is %f\n\n',norm(filtery-y1)/norm(y1))
```

Try other values of n until you get the best result. What did the trick?

Example 7. Compress the signal in the file data5.csv. Try different compression rates. Also, remember this was the data set you compressed using the linear predictive coding scheme on sheet 3.

Example 8. Filter the data noisydata3.csv from my homepage.