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 6.8.3 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 two 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 (function cutoff), then reduce the amplitudes you are using.

5. **Fourier Analysis of Images (2D DFT).**

In Parts 1–4 we treated a *time series* as sampled data $y(t_j)$ and examined how sampling, record length, and discretization affect the computed spectrum. An image is the analogous situation in two spatial variables. A grayscale image may be viewed as a sampled function

$$f(x,y) \quad \longrightarrow \quad f[m,n], \qquad m=0,\ldots,M-1,\ n=0,\ldots,N-1,$$

and we analyze its frequency content using the *2D discrete Fourier transform (2D DFT)*

$$F[k,\ell] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f[m,n]\, e^{-2\pi i\left(\frac{km}{M}+\frac{\ell n}{N}\right)}, \qquad k=0,\ldots,M-1,\ \ell=0,\ldots,N-1.$$

As in the 1D case, the magnitude $|F[k,\ell]|$ and the squared magnitude $|F[k,\ell]|^2$ describe how energy is distributed across spatial frequencies (horizontal and vertical oscillations). In MATLAB, the 2D FFT is computed by `fft2`.

a. **A first image and its spectrum (grayscale).**
MATLAB includes several convenient sample images. Begin with a grayscale image and compute its 2D spectrum. In order to view the spectrum clearly, we typically display $\log(1+|F|)$ and we *shift* the zero-frequency component to the center using `fftshift`.

```
I = imread('cameraman.tif');      % built-in grayscale image
I = im2double(I);

figure(1); imshow(I); title('Original image (grayscale)');

F = fft2(I);
S = log(1 + abs(fftshift(F)));    % log-magnitude for visualization

figure(2); imshow(S,[]); title('2D Fourier magnitude (log scale)');
```

Questions/observations.

i. Where is the low-frequency content located in the displayed spectrum, and what does it represent visually?
ii. Where does high-frequency content appear, and what kinds of visual features generate it?
iii. Compare an image with smooth shading to one with sharp edges. Which has more high-frequency content?

Interpretation of the 2D spectrum. In the display produced by `imshow`, the axes are labeled in pixel indices. However, each point in the image actually corresponds to a *discrete spatial frequency*. For an $M\times N$ image, the frequencies may be interpreted as

$$f_x = \frac{k}{M}, \qquad f_y = \frac{\ell}{N}, \qquad k=-\frac{M}{2},\ldots,\frac{M}{2}-1, \quad \ell=-\frac{N}{2},\ldots,\frac{N}{2}-1,$$

so that the units are *cycles per pixel*. This is the spatial analogue of the frequencies encountered earlier in time series.
The use of `fftshift` places the zero-frequency (average value) at the center of the image. Thus, the center of the spectrum represents *low spatial frequencies*, corresponding to slowly varying features such as overall brightness and smooth shading. As one moves away from the center, the frequencies increase, and these regions correspond to more rapid spatial variation.

Guided observations.

i. Identify the brightest region near the center. This corresponds to the average intensity (DC component). How does this relate to subtracting the mean in Part 3?
ii. Regions near the center represent slowly varying structure in the image. Describe what visual features (e.g., large smooth regions) contribute to this part of the spectrum.
iii. Regions farther from the center correspond to higher spatial frequencies. What features of the image (e.g., edges, sharp transitions, fine texture) contribute to these regions?
iv. Compare a smooth image (or a blurred version of the same image) with one containing sharp edges. How does the distribution of energy in the spectrum change?

Remark. You should find that smooth variations concentrate energy near the center of the spectrum, while sharp transitions spread energy toward higher frequencies. This is the spatial analogue of the observations made earlier for rapidly oscillating time signals.

b. **Resolution and sampling: powers of two vs. not a power of two.**
In 1D you changed the number of points N and observed how discretization affects the computed spectrum and the ability to resolve frequencies. Here we do the same by resampling the image to different resolutions. Use several powers of two (e.g., 64×64, 128×128, 256×256) and one size *not* a power of two (e.g., 150×150).

```
I64  = imresize(I,[64  64]);
I128 = imresize(I,[128 128]);
I256 = imresize(I,[256 256]);
I150 = imresize(I,[150 150]);    % not a power of two
```

For each image (say, I_{64}, I_{128}, I_{256}, I_{150}), compute and display the spectrum.

```
J = I128;                              % choose one of the resized images
F = fft2(J);
S = log(1 + abs(fftshift(F)));

figure; imshow(J);  title('Resampled image');
figure; imshow(S,[]); title('Spectrum of resampled image');
```

Questions/observations.

i. How does lowering resolution change the spectrum? What is lost first?

ii. Compare the clarity of edges and fine texture across the resolutions.

iii. Time the FFT for a power-of-two image size versus the non-power-of-two size:

```
tic; fft2(I256); toc
tic; fft2(I150); toc
```

Explain (in words) why powers of two are historically preferred for FFT computations.

c. **Black-and-white vs. color images (channels).**
A color image is (typically) three coupled intensity images: red, green, and blue channels. We can Fourier analyze each channel separately and compare their spectra.

```
C = im2double(imread('peppers.png'));    % built-in color image
R = C(:,:,1); G = C(:,:,2); B = C(:,:,3);

FR = fft2(R); FG = fft2(G); FB = fft2(B);

SR = log(1+abs(fftshift(FR)));
SG = log(1+abs(fftshift(FG)));
SB = log(1+abs(fftshift(FB)));

figure; imshow(C); title('Original color image');

figure; imshow(SR,[]); title('Red-channel spectrum');
figure; imshow(SG,[]); title('Green-channel spectrum');
figure; imshow(SB,[]); title('Blue-channel spectrum');
```

Questions/observations.

i. Are the three channel spectra similar or noticeably different? Describe what you see.

ii. Which channel seems to carry the strongest edge information for this image?

iii. What happens if you reconstruct the image using only one channel (e.g., keep red and set the others to zero)?

d. **Fourier-based compression by truncation in frequency space.**
In 1D, retaining only the dominant frequencies corresponds to removing small Fourier coefficients. In 2D, we can do the same by applying a mask in frequency space. A simple (and very instructive) approach is to keep only a centered square of low frequencies and set the rest to zero, then invert the transform. This is *not* JPEG, but it captures the key mathematical idea behind transform-based compression.

```
J = I256;                              % choose a grayscale image
F = fftshift(fft2(J));                 % shift so low-frequencies are
   centered
[M,N] = size(F);
```

```
mask = zeros(M,N);
r = 20;                                  % half-width of kept low-frequency
    block
mask(M/2-r:M/2+r, N/2-r:N/2+r) = 1;  % keep low frequencies only

Fcomp = F .* mask;
Jrec = real(ifft2(ifftshift(Fcomp)));

figure;
subplot(1,2,1); imshow(J);     title('Original');
subplot(1,2,2); imshow(Jrec,[]); title('Reconstruction (low-pass)');
```

Questions/observations.

i. Vary r (e.g., $r = 5, 10, 20, 40, 80$). How does image quality change?

ii. Compute the fraction of Fourier coefficients kept:

$$\text{fraction kept} = \frac{\#\{(k,\ell) : \text{mask}(k,\ell) = 1\}}{MN}.$$

How small can this fraction be before the image becomes unrecognizable?

iii. Describe the artifacts you see when r is small. Do you observe blurring? ringing?

iv. Compare a photograph-like image to a line drawing (high-contrast edges). Which compresses better under this simple scheme, and why?

e. **Optional extension: coefficient thresholding ("keep the largest").**
Instead of keeping a centered block (a low-pass filter), keep the largest-magnitude Fourier coefficients wherever they occur. This mimics a common compression idea: keep the coefficients that carry most of the energy.

```
J = I256;
F = fft2(J);
A = abs(F);

p = 0.05;                                % keep 5% of coefficients
th = quantile(A(:), 1-p);                % threshold so that p fraction are above
    it
Fkeep = F .* (A >= th);

Jrec = real(ifft2(Fkeep));

figure;
subplot(1,2,1); imshow(J); title('Original');
subplot(1,2,2); imshow(Jrec,[]); title('Reconstruction (largest
    coefficients)');
```

Questions/observations.

i. Compare this reconstruction to the low-pass reconstruction for the same percentage of coefficients kept.

ii. Which method better preserves edges? Which produces fewer artifacts?

Here is a way to obtain 25% low-pass filtering. Let the area kept be 25% of the total area, MN. For a square, this means a width $w = \sqrt{.25}M = .5M$. This is accomplished for our 2D spectrum as follows:

```
 F = fftshift(fft2(I));
[M,N] = size(F);

mask = zeros(M,N);

% keep 50% width -> 25% of coefficients
rM = round(M/4);
rN = round(N/4);

mask(M/2-rM:M/2+rM, N/2-rN:N/2+rN) = 1;

F_lp = F .* mask;
I_lp = real(ifft2(ifftshift(F_lp)));
```

```
% visualize spectrum
S_lp = log(1 + abs(F_lp));
figure, imshow(S_lp,[]), title('Low-pass (25% coefficients)')
```

6. **Filtering via Convolution (Optional).**
In the previous section, we modified the Fourier transform directly by removing or retaining selected frequency components. There is another way to achieve similar effects directly in the spatial domain using *convolution*. In fact, convolution in the spatial domain corresponds to multiplication in the frequency domain:

$$f * g \longleftrightarrow \hat{f}\,\hat{g}.$$

Thus, smoothing, sharpening, and edge detection can all be viewed as frequency filtering.

a. **Blurring (low-pass filtering).**
A simple way to smooth an image is to average nearby pixels. This can be done using a Gaussian filter.

```
I = im2double(imread('cameraman.tif'));

h = fspecial('gaussian',[15 15],2);   % Gaussian kernel
Iblur = imfilter(I,h,'replicate');

figure; imshow(I);     title('Original');
figure; imshow(Iblur); title('Blurred image');
```

Questions.

i. Compare this result with the low-pass reconstruction in Part 5. In what ways are they similar?

ii. What features of the image are most affected by the blurring?

b. **Edge detection (high-frequency emphasis).**
Edges correspond to rapid changes in intensity. These can be detected using simple difference operators such as the Sobel filter.

```
hx = [-1 0 1; -2 0 2; -1 0 1];   % Sobel operator
Ix = imfilter(I,hx,'replicate');

figure; imshow(Ix,[]); title('Edge detection');
```

Questions.

i. What parts of the image are highlighted by this operation?

ii. Explain why edges correspond to high-frequency components.

c. **Sharpening.**
Sharpening can be achieved by enhancing the high-frequency components. One simple method is to subtract a blurred version of the image and add it back to the original.

```
Isharp = I + 0.5*(I - Iblur);

figure; imshow(Isharp); title('Sharpened image');
```

Questions.

i. Compare the sharpened image with the original. What features are enhanced?

ii. Explain how this operation relates to separating low- and high-frequency components.

Remark. These examples illustrate that modifying an image through convolution produces effects similar to those obtained by manipulating its Fourier transform. In particular, blurring corresponds to removing high-frequency components, while sharpening enhances them. Thus, convolution provides a spatial-domain interpretation of the frequency-domain ideas explored earlier.