Foundations of Neuroimaging

Homework #2: Fourier Transform and Image Reconstruction

The goal of this homework is to learn basic concepts of 1-D and 2-D Fourier analyses in MATLAB, including forward and inverse Fourier transforms (FT) of signals in the spatial and spatial frequency domains, amplitude and phase spectra, image reconstruction, artifacts, aliasing, ghosts. It's fine to use Octave/Mathematica/R/etc instead. Turn in (email me) answers, graphs, and code (compact code appreciated) as a single pdf.
Overall hint: remember that matrix entries are identified by (row,column), which is up/down, then left/right, while pixels are conventionally identified by (x,y), which is left/right, then down/up.

1: 1-D FT. Plot the following 1-D functions and their fourier *amplitude* spectra:

$$f_1(x) = 0.3*\cos(3*2\pi*x), \quad f_2(x) = 1.0*\cos(8*2\pi*x), \quad \text{and} \quad f_3(x) = f_1(x) + f_2(x)$$

Qualitatively describe the spectrum of the first two functions and then the spectrum of their sum.
Hint (MATLAB):

```
  step=1/256;                % [stuff after a percent sign is a comment]
  x=0:step:1-step;           % get vector of 256 values of x from 0 to 1
  f1=cos(24*2*pi*x);         % f1 is a real vector with, e.g., 24 cycles from 0-1
  FT_f1=fft(f1);             % fft() returns the discrete fourier transform of f1
  FT_f1s=fftshift(FT_f1);    % fftshift displays zero freq of spect in middle of graph
  plot(abs(FT_f1s));         % FT_f1 is complex vector; abs() gets element amplitudes
```

2: 1-D Inverse FT. (this is a continuation of Problem 1). $F_1(f)$ is the frequency spectrum of $f_1(x)$. Reconstruct a function $f'_1(x)$ from $F_1(f)$ using the inverse Fourier transform, `ifft()`. Plot the real and imaginary parts, and the amplitude and phase of $f'_1(x)$. using `real()`, `imag()`, `abs()`, and `angle()`. Explain why the amplitude plot looks different than the real plot. The reconstructed phase and/or imaginary plots may be jagged; if so, explain why.

3: Display 2-D Image. Download the 256x256 sagittal T1 brain image tiff from `http://www.cogsci.ucsd.edu/~sereno/276/t1sag.tiff` (or use any other, uncompressed, 256x256 grayscale brain slice tiff!) and convert it into a matrix with (MATLAB):

```
        Im=double(imread('t1sag.tiff'));
```

Plot/display the image with:

```
        colormap gray; imagesc(Im,[minIm maxIm]); axis square;
```

Play with different numbers for *minIm* and *maxIm* (don't forget the brackets, and use a space, not a comma to separate them). The function `imagesc` autoscales if you omit the `[minIm maxIm]` vector.

4: 2-D FT. Compute the 2-D Fourier transform of image *Im* using:

```
        FT=fftshift(fft2(Im));
```

The *fftshift()* function puts the zero spatial frequency in the middle for a matrix (2-D) as well as vector (1-D) (you can apply it to the image to see what it does). Then make four plots of *FT* (which is a 2-D matrix of complex numbers): first the real and the imaginary components, and then the corresponding amplitude and phase components. Use the functions: `real(), imag(), abs(), angle()` to extract the components and `imagesc(Component,[minComponent maxComponent]);` to plot them. You will have to experiment with the minimum and maximum values to make sense of the pictures. You can use the functions `min()` and `max()` (apply them twice, that is, recursively, to get one number out of a 2-D matrix!). Describe the resulting distributions in spatial frequency space (k-space).

Hint (image and its amplitude spectrum plotted on one page, MATLAB):

```
  figure;        % multiple plots on one figure
  Im = double(imread('t1sag.tiff'));
  FT = fftshift(fft2(Im));
  FT_Amp = abs(FT);
  minAmp = min(min(FT_Amp)); maxAmp = max(max(FT_Amp));
  colormap gray;
  subplot(1,2,1); imagesc(Im);                          axis square;title('Image');
  subplot(1,2,2); imagesc(FT_Amp,[minAmp 0.01*maxAmp]);axis square;title('K Ampl');
```

5: k-Space Center. Manipulate the amplitude of the center point of k-space: *$F(k_{x0},k_{y0})$*, which in the present case can be referenced with `FT(129,129)` (N.B.: those coordinates assume `fftshift()` has *first* been applied). First *triple* the center point, reconstruct the images by using `ifft2(ifftshift())`, plot the original and reconstructed *amplitude* image using the same maximums and minimums, and describe the result. Then do the same reconstruction after *zeroing* the center k-space point. Finally, multiply the original k-space center by -3. Briefly describe how does the center of k-space affects contrast.

6: Spikes in k-Space. An individual data point in k-space is sometimes mistakenly assigned a very large value (e.g., as a result of an electrical transient at the exact moment that the data point was being collected). Modify the following three k-space points by setting them to a large value (e.g., 10^7), one at a time:

(a) FT(129+5, 129)   (k_y=5, k_x=0)
(b) FT(129,   129+33)   (k_y=0, k_x=33)
(c) FT(129+5, 129+33)   (k_y=5, k_x=33)

Reconstruct the images in each case using `ifft2(ifftshift())` and plot and describe the effects of each manipulation.

7: Zero Portions of k-Space. By setting portions of k-space to zero, certain ranges of spatial frequency will be removed when the image is reconstructed. Set the following regions of k-space to zero, reconstruct the images as above, then plot k-space (amplitude) and the reconstructed image in each case, and comment on the result:

(a) set FT(k_y, k_x) = 0, where both k_x and k_y are between 129-32 and 129+32 (high pass)
(b) set FT(k_y, k_x) = 0, *except* where k_x and k_y are between 129-32 and 129+32 (low pass)
(c) set FT(k_y, k_x) = 0, when x is between 193 and 256 (zero right edge of k-space)

Hint: One approach is to initialize a blank mask using the functions `ones()` *or* `zeros()`*, set ranges of the mask to 0 or 1 using* `low:high` *syntax, then apply mask to k-space with element-wise ('dot') operators (e.g., for multiplication:* `.*` *):*

```
  center = (129-32):(129+32);
  mask = zeros(256,256);
  mask(center,center) = 1;
  FT = FT .* mask;
```

8: Subsample k-Space. If an *image* (or a time signal) is not sampled frequently enough, aliasing (wraparound) will occur in the *frequency* domain (that is, after a Fourier transform). This is also true when going from the *frequency* domain back to *space* (or time); that is, if *k-space* is not sampled

frequently enough, aliasing will result in the *image* (or time) domain. Simulate this by zeroing every other line in k-space (e.g., *even* numbered lines). Comment on the effect of this undersampling after reconstructing the images with `ifft2(fftshift())`.
Hint:

```
  ev = 2:2:10
  od = 1:2:10
```

9: Shift Alternate Lines of k-Space. When k-space data is collected during an EPI scan, the even and odd lines may not be properly aligned because of imperfections of the gradients. Simulate this by shifting even k-space lines to the left and the odd k-space lines to the right (do this for two different cases using the shifts given in (a) and (b)):

(a) set FT(k_x, k_y) = FT(k_x-1, k_y), when k_y is odd and FT(k_x+1, k_y), when k_y is even
(b) set FT(k_x, k_y) = FT(k_x-4, k_y), when k_y is odd and FT(k_x+4, k_y), when k_y is even

Plot both k-space and reconstructed images for the above manipulations. How do the wraparound ghosts subtly differ from the ones generated in the previous problem?
Hint: watch limits so you don't go off the edge. Also, when shifting in a particular direction, say right, just leave the furthest left values alone.

10: Simulate B0 defect (challenging!). When k-space data is collected during an EPI scan in the presence of local B0 defects, the phase angle of the spins at that point in the image become distorted. Because of the small size of the phase-encode 'blips', this effect occurs mainly in the phase-encode direction. Model the effect of a 8x8 pixel B0 offset in the middle of the image by: (1) adding some phase (=multiplying by a complex exponential) to the data points there when calculating the Image->Signal using the (very!) slow explicit 2D Fourier transform (below) and then (2) reconstructing the image from the distorted data (use fast Fourier transform for this). Describe what occurs.
Hint1: Explicitly written-out FT (MATLAB) so phase of indiv. terms in Fourier sum can be modified:

```
  %%% slow 2D FT (square image) -- operating even on 32x32 pix image takes a while!
  n = length(Im);
  for ky=1:n; for kx=1:n
    ksum = 0;
    for y=1:n; for x=1:n
      term = Im(y,x) * exp(-i*2*pi*( (ky-1)*(y-1) + (kx-1)*(x-1) )/n);
      % here modify phase of term by multiplying by complex exp() before add to sum
      ksum = ksum + term;
    end; end
    FT(ky,kx) = ksum;
  end; end
  FT=fftshift(FT);
```

Hint2: Maybe downsample 256x256 image (e.g., slower computer)! The Fourier transform code above is easy to understand since it looks exactly like the equation, but it runs slowly in the Matlab interpreter since it does not take advantage of Matlab matrix operations. Quick and dirty way to downsample the image to 64x64:

```
  Im2 = Im(4:4:256,4:4:256);
```

Hint3: The phase errors <u>accumulate</u> over the course of the EPI readout, so take this critical factor into consideration when adding phase to terms (what happens if you add the <u>same</u> phase angle to each spatial frequency?).
Hint4: Example of how to add phase to term: `term = term * exp(-i*2*pi*blah)`.