BME 6310, Fall 2021 C. Meyer

Homework #9

Due: Wednesday, December 1, 2021

1. Calculate the following circular convolutions manually in the time domain:

   (a) { 0, 1, 0, 0 } $\circledast$ { 0, 0, 1, 0 }

   (b) { 1, 1, 0, 0 } $\circledast$ { 0, 0, 1, 1 }

   In each case, verify your results in Matlab by using the discrete convolution theorem ; i.e., by taking DFT's, multiplying, and taking inverse DFT's.

2. Using Matlab/Python, define a discrete triangle function as follows:
   Matlab

   ```
   >> f = [0:.01:1 0.99:-.01:0];
   ```

   Python

   ```
   f = np.concatenate((np.arange(0,1,0.01), np.arange(0.99,0,-0.01)))
   ```

   (a) Using the convolution theorem and the Matlab functions fft() and ifft() or Python functions np.fft.fft() and np.fft.ifft(), determine and plot the circular convolution of f with itself. (You may need to take the real part of your result to eliminate a small imaginary component that may result from roundoff errors.) How many data points are in the result? Does it look like what you expect from the self-convolution of a triangle? Explain the result.

   (b) Using the Matlab function conv() or Python function np.convolve(), determine and plot the discrete convolution of f with itself. How many data points are in this result? Does this look more like what you expect from a convolution?

   (c) Determine and plot the discrete convolution by zero-padding and using fft() and ifft(). Compare your result with that of part (b).

   Hand in the commands that you used to generate the plots, along with the plots and the answers to the questions.

3. Assume that we have a discrete time-domain sequence, $x[n]$, composed of samples obtained from sampling an analog signal, $x(t)$. Assume that $x[n]$ contains N = 500 samples and that it was sampled at a rate of f_s = 3000 Hz.

(a) What is the frequency spacing of $X[k]$ = DFT$\{x[n]\}$, measured in Hz?

(b) What is the highest-frequency spectral component that can be present in the analog $x(t)$ signal where no aliasing errors occur in $x[n]$?

(c) If you drew the full $X[k]$ spectrum and several of its spectral replications, what is the spacing between the spectral replications, measured in Hz?

4. In this problem, you will design and apply a finite-impulse-response (FIR) digital filter to remove interference from a measurement.

   First load hw9prob4data.mat and plot the signal over time. The data contains the signal, g, which has a sampling rate of 50 kHz, and also the time points of the signal, t.
   Matlab

   ```
   >> load("hw9prob4data.mat");
   >> plot(t,g)
   ```

   Python

   ```
   from scipy.io import loadmat
   data = loadmat("hw9prob4data.mat")
   g = data['g'][0,:]
   t = data['t'][0,:]
   plt.plot(t,g)
   ```

   There is a large interfering signal with a frequency of around 500 Hz. Your goal is to design a digital filter to remove as much of this interference as possible while disturbing the underlying signal as little as possible. You should explore various filter designs, looking at their frequency responses and their responses in the time domain. You can choose to use a lowpass filter or a highpass filter. (A bandstop filter is another option, but you may have difficulty getting a sensible bandstop design with these parameters.)

   One way to implement an FIR filter is by convolving it with the input using zero-padded FFT's and the convolution theorem. For this problem, use the "filter" function in Matlab to apply the filter. This function implements the digital filter in the time domain, and can handle both nonrecursive (e.g., FIR) and recursive filters. It will illustrate the effect of the time delay inherent in your filter design.

   You may use any FIR filter design technique that you choose. You may find it convenient to use filterDesigner in Matlab. When you want to test a particular design when using fdatool, choose Export from the File menu. Or, you may choose to use the designfilt() function in Matlab. See the attached script designfiltexample.m or designfiltexample.py for examples.

   Be sure to use the information about the frequency of the interfering signal in choosing your filter type and in evaluating the frequency response of your design.

   Show your best filtered output, along with a plot of the filter impulse response and magnitude frequency response. Describe the parameters that you used to generate your filter, along with your rationale for choosing your design.

5. In this problem, you will reconstruct images from MRI data that is acquired in 2D Fourier transform space, which is known as k-space in the MRI literature. (This problem includes teaching material first developed by John Pauly.)

   Centered 2DFT's Generally in medical image reconstruction, the origin for both the k-space data and the reconstructed image is the center of the image or data array. The usual convention for the fft is that the origin is at the beginning of the array, or the corner of a 2D array. The one-dimensional functions, fftc and ifftc, and two-dimensional functions fft2c and ifft2c have been included with this homework. They can be found in their respective matlab files, or inside of the hw9_functions.py file.

   Displaying Images For displaying images the best option is "imshow" which is part of the image processing toolbox. If you have a complex image "im", you display it with:
   Matlab

   ```
   >> imshow(abs(im),[]).
   ```

   Python

   ```
   plt.imshow(abs(im), cmap='gray').
   ```

   This automatically scales the image from the image minimum to maximum. If you want to explicitly specify these, use:
   Matlab

   ```
   >> imshow(abs(im),[win_min win_max]).
   ```

   Python

   ```
   plt.imshow(abs(im), vmin=win_min, vmax=win_max).
   ```

   This is quite common. Often the brightest part of the image is of little interest (a vessel, or fat, for instance). You will often want to specify a narrower window to accentuate the part of the image that is of interest.

   You can add a title to the plot, which is helpful to remind you what you were thinking when you generated it, Matlab

   ```
   >> title('One cycle of phase correction in x');
   ```

   Python

   ```
   plt.title('One cycle of phase correction in x')
   ```

MR Image Reconstruction

The following attached file is an MRI 2DFT raw data file: fse_t1_ax_data.mat. This data set was acquired on a low-field (0.5T) interventional scanner using a head coil for transmit and receive. It is an axial T1-weighted shot of the head of a normal volunteer at the level of the eyes. CSF is dark due it its very long T1 relaxation time and white matter is brighter than gray matter. Fat is very bright due to its short T1. This was acquired with a fast-spin-echo pulse sequence.

The raw data matrix has 256×256 samples. When you load this into Matlab with

```
>> load fse_t1_ax_data.mat
```

or in Python with

```
data = loadmat("fse_t1_ax_data.mat")
d = data['d']
```

you will have a variable `d` that contains the raw data for this slice.

(a) Because conventional 2DFT MRI raw data is acquired on a grid in 2D Fourier transform space, it is easy to reconstruct an image. Reconstruct an image from this data by performing a 2D inverse FFT on the raw data. Display the magnitude of the reconstructed image next to the image of the raw data. Because of its large dynamic range, display the log of the magnitude of the raw data:
Matlab

```
>> imshow(abs(log(d)),[])
```

Python

```
plt.imshow(np.abs(np.log(d)))
```

(b) Next, reconstruct an image with half of the original data. First, zero out every other column of the raw data matrix:
Matlab

```
>> dhalf = zeros(size(d));
>> dhalf(:,1:2:end) = d(:,1:2:end);
```

Python

```
dhalf = d[:,::2]
```

Then, reconstruct an image from this undersampled raw data and display the magnitude of the image. Explain what you see in this image.

(c) MRI 2DFT raw data is a time signal along one direction. One way to avoid the problems seen in part(b) is to use a temporal filter to suppress MRI signal originating from the edges of the image along this direction. Here, you will design a low-pass filter to suppress signal coming from the right and left portions of the image. The

image should show the center half of the image along the horizontal direction. The nonzero part of the image will thus be a rectangle that is taller than it is wide, with dark bands on the right and left sides of the image. Use the following code to design an FIR filter and apply it to the raw data.

Matlab

```
>> mycutoff = ???; % Empirically set this to a number less than 750.
>> D = designfilt('lowpassfir', 'FilterOrder', 32, 'CutoffFrequency',
mycutoff, 'SampleRate', 1500);
>> dfilt = zeros(size(d));
>> for n = 1:size(d,1)
>>      dfilt(n,:)  = filter(D, d(n,:));
>> end;
```

Python

```
mycutoff = ???  # Empirically set this to a number less than 750.
D = firwin(numtaps=33, cutoff=mycutoff, pass_zero='lowpass', fs=1500)
dfilt = np.zeros(d.shape)
for n in range(d.shape[0]):
     dfilt[n,:]  = np.convolve(D, d[n,:], 'same')
```

Reconstruct the image until you find a cutoff frequency that achieves the desired result. Display the resulting image.

(d) Next, subsample the filtered raw data as you subsampled the original raw data in part (b). Reconstruct and display the image using the resulting filtered and subsampled data. Briefly explain the resulting image in comparison to the part (b) image. Would this image be better for visualizing the center of the brain?

(e) Finally, use the filtered and subsampled data from part (d) to generate a rectangular image that only shows the center half of of the image along the horizontal direction. One way to do this would be to just crop the image in part (b). Instead, here you will remove the columns of the *raw* data that you zeroed out in part (d) (called "dfilthalf" below):

Matlab

```
>> dfiltrect = zeros(256,128);
>> dfiltrect(:, 1:128) = dfilthalf(:, 1:2:end);
```

Python

```
dfiltrect = np.zeros((256,128))
dfiltrect[:, :]  = dfilthalf[:,0::2]
```

Reconstruct and display the resulting rectangular image. This is one way to restrict the image field-of-view along one direction and to reconstruct an image with half of

the data. This method is routinely used clinically with standard MRI hardware and software. (Unfortunately, this does not shorten the scan time; there are other signal processing methods for that purpose.)