Project 5: Image Filtering & Signal Processing

You will write up a formal lab report in IEEE double-column format with figures integrated with the text. The exercises should written up in this week's lab report. You should **label** the axes of your plots, have a caption, and Figure number for every plot. Every plot should be referenced by Figure number in your text discussion. Include each plot *inlined* within your report.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports, but you cannot give or receive written material or electronic files. Your submitted work should be original and it should be your own work.

The goal of this lab is to study applications of FIR filters such as the deconvolution of speech signals and images. In the experiments of this lab, you will use `firfilt( )`, or `conv()`, to implement 1-D filters and `conv2()` to implement two-dimensional (2-D) filters. The 2-D filtering operation actually consists of 1-D filters applied to all the rows of the image and then all the columns.

One objective in this lab to learn showing how interpolation is used to create color images for a digital camera. For the interpolation, you will learn how to implement FIR filters in MATLAB, and then use these FIR filters to perform the interpolation of images.

This uses FIR filtering as well as two two MATLAB GUIs: one for sampling and aliasing and one for convolution.

- `con2dis:` GUI for sampling and aliasing. An input sinusoid and its spectrum is tracked through A/D and D/A converters.
- `dconvdemo:` GUI for discrete-time convolution. This is exactly the same as the MATLAB functions `conv()` and `firfilt()` used to implement FIR filters.

Both of these demos are part of the *SP-First Toolbox*, which be downloaded from WebCT for ECE2025 via a link on "Lab Assignments" page.

1 Overview of Filtering

An FIR *filter* converts an input signal $x[n]$ into an output signal $y[n]$ by means of the weighted summation:

$$y[n] = \sum_{k=0}^{M} b_k \, x[n-k], \tag{1}$$

computing the n^{th} value of the output sequence from (1) from certain values of the input sequence. The filter coefficients $\{b_k\}$ are constants that define the filter's behavior. As an example, consider the system for which the output values are given by

$$\begin{aligned} y[n] &= \tfrac{1}{3}x[n] + \tfrac{1}{3}x[n-1] + \tfrac{1}{3}x[n-2] \\ &= \tfrac{1}{3}\left\{x[n] + x[n-1] + x[n-2]\right\} \end{aligned} \tag{2}$$

This equation states that the n^{th} value of the output sequence is the average of the n^{th} value of the input sequence $x[n]$ and the two preceding values, $x[n-1]$ and $x[n-2]$. For this example the b_k's are $b_0 = \frac{1}{3}$, $b_1 = \frac{1}{3}$, and $b_2 = \frac{1}{3}$.

MATLAB has built-in functions, `conv( )` and `filter( )`, for implementing the operation in (1), but we have also supplied another M-file `firfilt( )` for the special case of FIR filtering. The function `filter` implements a wider class of filters than just the FIR case. Technically speaking, the `conv` and `firfilt` functions both implement the operation called *convolution*. The following MATLAB statements implement the three-point averaging system of (2):

```
nn = 0:99;                %<--Time indices
xx = cos( 0.08*pi*nn );   %<--Input signal
bb = [1/3 1/3 1/3];       %<--Filter coefficients
yy = firfilt(bb, xx);     %<--Compute the output
```

In this case, the input signal `xx` is a vector containing a cosine function. In general, the vector `bb` contains the filter coefficients $\{b_k\}$ needed in (1). These are loaded into the `bb` vector in the following way:

`bb = [b0, b1, b2, ...  , bM].`

In MATLAB, all sequences have finite length because they are stored in vectors. If the input signal has, for example, L samples, we would normally only store the L samples in a vector, and would assume that $x[n] = 0$ for n outside the interval of L samples; i.e., we do not have to store any zero samples unless it suits our purposes. If we process a finite-length signal through (1), then the output sequence $y[n]$ will be longer than $x[n]$ by M samples. Whenever `firfilt( )` implements (1), we will find that

`length(yy) = length(xx)+length(bb)-1`

In the experiments of this lab, you will use `firfilt( )` to implement FIR filters and begin to understand how the filter coefficients define a digital filtering algorithm. In addition, this lab will introduce examples to show how a filter reacts to different frequency components in the input.

Using the GUIs: The first objective is to demonstrate usage of the two GUIs. First of all, you must download the ZIP files for each and install them. Each one installs as a directory containing a number of files. You can put the GUIs on the `matlabpath`, or you can run the GUIs from their home directories.

1.1 Sampling and Aliasing Demo

In this demo, you can change the frequency of an input signal that is a sinusoid, and you can change the sampling frequency. The GUI will show the sampled signal, $x[n]$, its spectrum, and also the reconstructed output signal, $y(t)$ with its spectrum. Figure 1 shows the interface for the `con2dis` GUI.
perform the following stepswith the `con2dis` GUI:

- Set the input to $x(t) = \cos(40\pi t + 0.5\pi)$
- Set the sampling rate to $f_s = 24$ samples/sec.
- Determine the locations of the spectrum lines for the discrete-time signal, $x[n]$, found in the middle panels. Make sure that the `Radian` button is active so that the frequency axis for the discrete-time signal is $\hat{\omega}$.
- Determine the formula for the output signal, $y(t)$ shown in the rightmost panels. What is the output frequency in Hz?

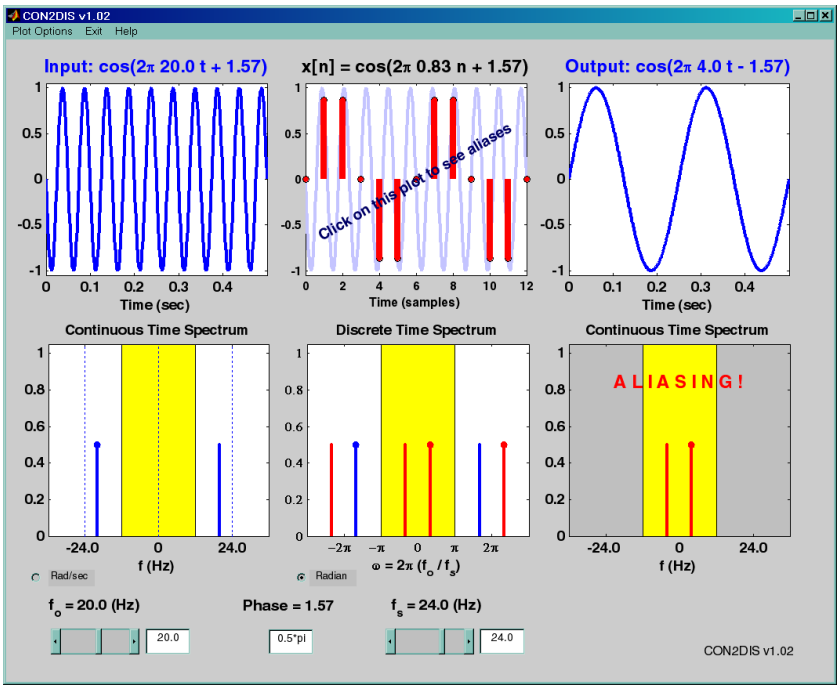

Figure 1: The `con2dis` MATLAB GUI interface.

1.2 Discrete-Time Convolution Demo

In the `dconvdemo` GUI (Fig. 2), you can select an input signal $x[n]$, as well as the impulse response of the filter $h[n]$. Then the demo shows the ***sliding window*** view of FIR filtering. In this view, one of the signals must be *flipped and shifted* along the axis when convolution is computed. Perform the following steps with the `dconvdemo` GUI.

- Click on the `Get x[n]` button and set the input to a finite-length pulse: $x[n] = (u[n] - u[n-10])$. Note the length of this pulse.
- Set the filter to a three-point averager by using the `Get h[n]` button to create the correct impulse response for the three-point averager. Remember that the impulse response is identical to the b_k's for an FIR filter. Also, the GUI allows you to modify the length and values of the pulse.
- Observe that the GUI produces the output signal in the bottom panel.
- When you move the mouse pointer over the index "n" below the signal plot and do a click-hold, you will get a *hand tool* that allows you to move the "n"-pointer. By moving the pointer horizontally you can observe the sliding window action of convolution. You can even move the index beyond the limits of the window and the plot will scroll over to align with "n."

1.3 Filtering via Convolution

You can perform the same convolution as done by the `dconvdemo` GUI by using the MATLAB function `firfilt`, or `conv`. For ECE-2025, the preferred function is `firfilt`.

- Filtering with a 3-point averager. The filter coefficient vector for the 3-point averager is defined via:

```
bb = 1/3*ones(1,3);
```

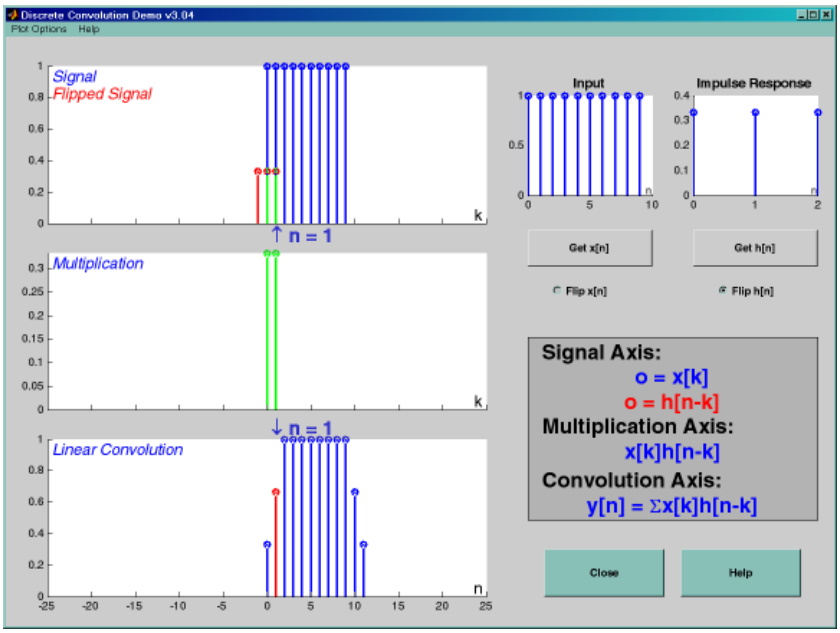

Figure 2: Interface for discrete-time convolution GUI called `dconvdemo`.

Use `firfilt` to process an input signal that is a length-10 pulse:

$$x[n] = \begin{cases} 1 & \text{for } n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \\ 0 & \text{elsewhere} \end{cases}$$

NOTE: in MATLAB indexing can be confusing. Our mathematical signal definitions start at $n = 0$, but MATLAB starts its indexing at "1". Nevertheless, we can ignore the difference and pretend that MATLAB is indexing from zero, as long as we don't try to write `x[0]` in MATLAB. For this experiment, generate the length-10 pulse and put it inside of a longer vector with the statement `xx = [ones(1,10),zeros(1,5)]`. This produces a vector of length 15, which has 5 extra zero samples appended.

- To illustrate the filtering action of the 3-point averager, it is informative to make a plot of the input signal and output signals together. Since $x[n]$ and $y[n]$ are discrete-time signals, a `stem` plot is needed. One way to put the plots together is to use `subplot(2,1,*)` to make a two-panel display:

```
nn = first:last;          %--- use first=1 and last=length(xx)
subplot(2,1,1);
stem(nn-1,xx(nn))
subplot(2,1,2);
stem(nn-1,yy(nn),'filled') %--Make black dots
xlabel('Time Index  (n)')
```

  This code assumes that the output from `firfilt` is called `yy`. Try the plot with `first` equal to the beginning index of the input signal, and `last` chosen to be the last index of the input. In other words, the plotting range for both signals will be equal to the length of the input signal, even though the output signal is longer. Notice that using `nn-1` in the call to `stem( )` causes the x-axis to start at zero in the plot.

- Explain the filtering action of the 3-point averager by comparing the plots in the previous part. This filter might be called a "smoothing" filter. Note how the transitions in $x[n]$ from zero to one, and from one back to zero, have been "smoothed."

2 MATLAB Convolution

2.1 Sampling and Aliasing

Use the `con2dis` GUI to do the following problem:

- Set the input frequency to 12 Hz, and the input phase to $\phi = -\pi/3$.
- Set the sampling frequency to 14 Hz.
- Determine the frequency and phase of the reconstructed output signal.
- Determine the locations in $\hat{\omega}$ of the lines in the spectrum of the discrete-time signal. Give precise numerical values.
- Change the sampling frequency to 10 Hz, and explain the appearance of the output signal.

2.2 Discrete-Time Convolution

In this section, you will generate filtering results needed in a later section. Use the discrete-time convolution GUI, `dconvdemo`, to do the following:

- Set the input signal to be $x[n] = (0.9)^{n-4}\,(u[n-12] - u[n-4])$. Use the "Exponential" signal type and the "Delay" feature within `Get x[n]`.
- Set the impulse response to be $h[n] = \delta[n-1] - 0.9\delta[n-2]$. Once again, use the "Exponential" signal type within `Get h[n]`.
- Examine the output signal $y[n]$ and explain why it is zero for almost all points. Compute the numerical value of the last point in $y[n]$, i.e., the one that is nonzero.

2.3 Loading Data

In order to exercise the basic filtering function `firfilt`, we will use some "real" data. In MATLAB you can load data from a file called `Lab07f04dat.mat` file by using the `load` command as follows:

```
load Lab07f04dat
```

The data file `Lab07f04dat.mat` contains two filters and three signals, stored as separate MATLAB variables:

`x1:` a stair-step signal such as one might find in one sampled scan line from a TV test pattern (black and white) image.

`xtv:` an actual scan line from a digital (black and white) image.

`x2:` a speech waveform ("oak is strong") sampled at $f_s = 8000$ samples/second.

`h1:` the coefficients for a FIR discrete-time filter.

`h2:` coefficients for a second FIR filter.

After loading the data, use the `whos` function to verify that all five vectors are in your MATLAB workspace.

2.4 Filtering a Signal

You will now use the signal vector `x1` as the input to an FIR filter.

- For the warm-up, you should do the filtering with a six-point averager for the FIR filter. Define the filter coefficient vector `bb` for the six-point averager and then use it in `firfilt` to process `x1`. How long are the input and output signals?

  When unsure about a command, use `help`.

- To illustrate the filtering action of the six-point averager, you must make a plot of the input signal and output signal together (as shown in Section 1.3). The plotting range for both signals should be set equal to the length of the input signal, even though the output signal is longer.

- Since the previous plot is quite crowded, it is useful to show a small part of the signals. Repeat the previous part with the plotting interval chosen to display 25 points from the middle of the signals.

- Explain the filtering action of the six-point averager by comparing the plots from parts (b) and (c).

  Instructor Verification (separate page)

2.5 Filtering Images: 2-D Convolution

One-dimensional FIR filters, such as running averagers and first-difference filters, can be applied to one-dimensional signals such as speech or music. These same filters can be applied to images if we regard each row (or column) of the image as a one-dimensional signal. For example, the 50^{th} row of an image is the N-point sequence `xx[50,n]` for $1 \leq n \leq N$, so we can filter this sequence with a 1-D filter using the `conv` or `firfilt` operator.

One objective of this lab is to show how simple 2-D filtering can be accomplished with 1-D row and column filters. It might be tempting to use a `for` loop to write an M-file that would filter all the rows. For a *first-difference filter*, this would create a new image made up of the filtered rows:

$$y_1[m,n] = x[m,n] - x[m,n-1]$$

However, this image $y_1[m,n]$ would only be filtered in the horizontal direction. Filtering the columns would require another `for` loop, and finally you would have the completely filtered image:

$$y_2[m,n] = y_1[m,n] - y_1[m-1,n]$$

In this case, the image $y_2[m,n]$ has been filtered in both directions by a first-difference filter

These filtering operations involve a lot of `conv` calculations, so the process can be slow. Fortunately, MATLAB has a built-in function `conv2( )` that will do this with a single call. It performs a more general filtering operation than row/column filtering, but since it can do these simple 1-D operations it will be very helpful in this lab.

- Load in the image `echart.mat` with the `load` command (it will create the variable `echart` whose size is 257×256). We can filter all the rows of the image at once with the `conv2( )` function. To filter the image in the horizontal direction using a six-point averager, we form a *row* vector of filter coefficients and use the following MATLAB statements:

```
bbh =  % <--- row vector of filter coefficients
yy1 = conv2(echart, bbh);
```

In other words, the filter coefficients `bbh` for the six-point averager are stored in a *row* vector and will cause `conv2( )` to filter all rows in the *horizontal* direction. Display the input image `echart` and the output image `yy1` on the screen at the same time. Compare the two images and give a qualitative description of what you see.

- Now filter the "eye-chart" image `echart` in the *vertical* direction with a six-point averager to produce the image `yy2`. This is done by using `yy2 = conv2(echart,bbh')` so that the transpose operator creates a column vector of filter coefficients. Display the image `yy2` on the screen and describe in words how the output image compares to the input.

3 FIR Filter Opportunities

In the following sections you will study how a filter can produce some of all of the following special effects:

- *Blurring:* FIR filters that do local averaging tend to blur an image.
- *Echo:* FIR filters can produce echoes and reverberations when the FIR filtering formula contains delay terms. In an image, such phenomena would be called "ghosts."
- *Deconvolution:* one FIR filter can (approximately) undo the effects of another—we will investigate a cascade of two FIR filters that distort and then restore an image. This process is called *deconvolution.*

3.1 Cascading Two Systems

More complicated systems are often made up from simple building blocks. In the system of Fig. 3, two FIR filters are connected "in cascade." For this section, assume that the the filters in Fig. 3 are described by the two equations:

$$w[n] = \sum_{\ell=0}^{M} q^{\ell} x[n-\ell] \qquad \text{(FIR Filter-1)} \tag{3}$$

$$y[n] = w[n] + r\, w[n-1] \qquad \text{(FIR Filter-2)} \tag{4}$$

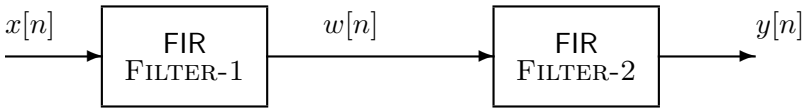

Figure 3: Cascading two FIR filters: the second filter attempts to "deconvolve" the distortion introduced by the first.

3.1.1 Overall Impulse Response

- Implement the system in Fig. 3 using Matlab to get the impulse response of the overall cascaded system for the case where $q = 0.88$, $r = -0.88$ and $M = 19$. Plot the impulse response of the overall cascaded system.
- Work out the impulse response $h[n]$ of the cascaded system by hand to verify that your Matlab result in part (a) is correct. (Hint: consult old Homework problems.)

- In a *deconvolution* application, the second system (FIR FILTER-2) tries to undo the convolutional effect of the first. Perfect deconvolution would require that the cascade combination of the two systems be equivalent to the identity system: $y[n] = x[n]$. If the impulse responses of the two systems are $h_1[n]$ and $h_2[n]$, state the condition on $h_1[n] * h_2[n]$ to achieve perfect deconvolution.[1]

3.2 Deconvolution Experiment for 1-D Filters

Use the function `firfilt( )` to implement the following FIR filter

$$w[n] = \sum_{\ell=0}^{19} (0.88)^{\ell} x[n-\ell] \tag{5}$$

on the input signal $x[n]$ defined via the MATLAB statement: `xx = 256*(rem(0:100,50)<10);`

- In MATLAB you must define the vector of filter coefficients `bb` needed in `firfilt`. Make a plot of the impulse response of this filter. Recall the correspondence between the impulse response $h[n]$ and the filter coefficients, $\{b_k\}$.

- Plot both the input and output waveforms $x[n]$ and $w[n]$ on the same figure, using `subplot`. Make the discrete-time signal plots with MATLAB's `stem` function, but restrict the horizontal axis to the range $0 \leq n \leq 75$. Explain why the output appears the way it does by figuring out the effect of the filter coefficients in (5). *Hint:* use the "sliding window" interpretation of convolution.

- Note that $w[n]$ and $x[n]$ are not the same length. Determine the length of the filtered signal $w[n]$, and explain how its length is related to the length of $x[n]$ and the length of the FIR filter.

3.2.1 Restoration Filter

The following FIR filter

$$y[n] = w[n] + rw[n-1] \qquad \text{(FIR FILTER-2)} \tag{6}$$

can be use to undo the effects of the FIR filter in the previous section (see the block diagram in Fig. 3). It attempts to perform restoration, but only does so approximately. Use the following steps to show how well it works when $r = -0.88$.

- Process the signal $w[n]$ from Section 3.2 with FILTER-2 to obtain the output signal $y[n]$.

- Make stem plots of $w[n]$ and $y[n]$ using a time-index axis n that is the same for both signals. Put the stem plots in the same window for comparison—using a two-panel subplot.

- Since the objective of the restoration filter is to produce a $y[n]$ that is almost identical to $x[n]$, make a plot of the error between $x[n]$ and $y[n]$ over the range $0 \leq n < 50$. Define the error signal to b e $e[n] = y[n] - x[n]$.

- Use the overall impulse response of the cascade system consisting of the FIR filter defined in (5) followed by FILTER-2 to figure out a mathematical formula for the error $e[n]$.

[1] Note: the cascade of FIR FILTER-1 and FILTER-2 does not perform *perfect* deconvolution.

3.2.2 Worst-Case Error

- Evaluate the *worst-case error* by doing the following: find the maximum absolute value of the difference between $y[n]$ and $x[n]$ in the range $0 \leq n < 50$.
- What does the error plot and worst case error tell you about the quality of the restoration of $x[n]$? How small do you think the worst case error has to be so that it cannot be seen on a plot?

3.3 Distorting and Restoring Images

If we pick q to be a little less than 1.0, then the first system (FIR FILTER-1) will cause distortion when applied to the rows and columns of an image. The objective in this section is to show that we can use the second system (FIR FILTER-2) to undo this distortion (more or less). Since FIR FILTER-2 will try to undo the convolutional effect of the first, it acts as a *deconvolution* operator.

- Read in the image `pccat.png` with the `imread` command.
- Pick $q = 0.88$ and $M = 19$ in FILTER-1 and filter the `pccat` image in both directions: apply FILTER-1 along the horizontal direction and then filter the resulting image along the vertical direction also with FILTER-1. Call the result `cat88`. Describe the degradation done to the `pccat` image by FILTER-1.
- Deconvolve `cat88` with FIR FILTER-2, choosing $r = -0.88$. Describe the visual appearance of the output image, and point out any notable features. Justify your observations by invoking your mathematical understanding of the cascade filtering process. Explain why the regions around the edges of the picture look different from the rest.
- Explain why you don't see "ghosts" in the output image, but use some previous mathematical calculations to determine that there should be ghosts (or echoes) of a certain size, and that their locations are known. Evaluate the *worst-case error* in order to say how big the ghosts are relative to "black-white" transitions in the gray scale which runs from 0 to 255.

Include all images and plots for the previous two parts to support your discussions in the lab report.

3.4 Restoring a Filtered Speech Waveform

In this section, you will try to restore the "oak is strong" speech signal from a degraded version of the signal. The degradation was done by filtering the original speech signal `x2` through an FIR filter defined by FILTER-1. However the filter parameters, q and M are unknown. The degraded speech signal is called `dd` in the `Lab07f04dat` file.

Since a filter of the type described by FILTER-1 did the degradation, it should be possible to recover something very close to the original speech signal by using a filter of the type defined by FILTER-2, and the result should sound very much like the original. This process is called *deconvolution.*

- Use an FIR filter of the type described by FILTER-2 in your attempt to restore the speech to its original form. Try different values of r, and for each one listen to the result. Pick the value of r for which the recovered speech sounds most like the original signal, which is in the vector `x2`.
  Note: complete this listening test before starting the next part.

- It is quite likely that your choice of r in the previous part could have been any one of several values because your hearing could not detect tiny differences among different signals. Therefore, it is common practice to formulate a mathematical way to evaluate how well the deconvolution is done. If we follow the notation of Fig. 3 and denote the original signal as $x[n]$ and the deconvolution output as $y[n]$, then we can define an *error signal* $e[n]$ as

$$e[n] = y[n] - x[n]$$

Intuitively, the best deconvolution result would give zero error, but zero is unlikely. Instead, we can evaluate the "size" of the error by doing a sum of squares:

$$E = \frac{\sum_{n=0}^{N_x-1} |e[n]|^2}{\sum_{n=0}^{N_x-1} |x[n]|^2}$$

where N_x is the number of samples in the signal $x[n]$. The denominator is used to normalize the error with respect to the "size" of the original signal.

For your deconvolution processing, make a plot of E versus r over a set of values near the value of r chosen in the previous part. You should be able to show that one value of r is a local minimum for E versus r. Finally listen to the deconvolved signal for this case and verify that it sounds as well as the output for the value of r that you found in part (3.4).