Revised 2 November 2015

Project # 1: Digital Signals and Computer Graphics

In this lab you will use MATLAB to study the following topics:

- How to create and use MATLAB function files.
- How to sample and quantize an analog audio signal, and then compare the quantized signal to the original signal.
- How to use vector graphics to create and transform two-dimensional images.

Preliminaries

Textbook: This computer project illustrates the ideas in Sections 1.2–1.6 of the textbook.

Review of MATLAB: You probably already used MATLAB, either in the Math 250-C course, the Computer Science 107 course, or in an engineering course. If you are not familiar with MATLAB or want a quick review, work through the tutorial in Section A.3.1 of the textbook before continuing. Every MATLAB command is documented in a `help` file, which you can access during a MATLAB session. For example typing `help format` gives information about the command `format`. Go to the mathematics department web site for this course for additional MATLAB documents if you want further information.

Function Files: For more complicated MATLAB calculations you should use *function* files. A function file contains one or several MATLAB commands and is stored as a text file with a descriptive name such as `myfunction.m`, for example (the extension `*.m` is required). A *function file* requires the input of variables (which can be numbers or matrices), and gives as output the function evaluated at the input variables. When you type the name of a function file with the specific values for the input variables, then the function defined in your function file is evaluated at these variables.

Writing Function Files: Start MATLAB and click on *File*, then *New*. Move the pointer to the right and click on *Function*. This will open the MATLAB Editor/Debugger Window, and you can type the commands in this window. You can take any m-file, edit it (just as you would edit any text file), and then save it under a different name to obtain a new m-file.

Running Function Files: After you have created a function file and saved it to your directory, you must set the *Path* so that MATLAB can find this file. Click on *File*, then *Set Path* and follow the directions to add your directory to the list of path names.

Function Files for Project 1: Use the text editor in MATLAB to create and save the following MATLAB *function* m-files:

(a) `plot2d.m`: Create a *function* m-file with the commands

```
function plot2d(X)
x = X(1,:)';
y = X(2,:)';
plot(x, y, 'ro', x, y, 'g-');
axis([-10 10 -10 10])
axis('square')
```

(note the semicolons at the end of the lines–these suppress output to the screen). Save this file under the name `plot2d.m`. Be sure that you have set the *Path* as described above so that MATLAB can find this m-file. The input to this program is any matrix X with at least two rows and any number of columns. The output of `plot2d(X)` is the plot of the line figure whose vertices have x-y coordinates given by the first and second entries in each column of X (the vertices of consecutive columns of X are connected by a line). If the first and last columns of X are the same the figure will be a closed polygon. The `axis` command sets the scale to [-10, 10] in both directions. Test the file by clicking on the MATLAB window and typing

```
X = [0 1 1 0; 0 1 -1 0]
plot2d(X)
```

at the MATLAB prompt. You should get a triangle with vertices at $(0, 0)$, $(1, 1)$, and $(1, -1)$ (if MATLAB can't find the file `plot2d.m`, check the *Path* settings as described above).

(b) `randtri.m`**:** Create a function m-file to generate random triangles with one vertex at $(0, 0)$ and scaling factor s by using the commands

```
function Y = randtri(s)
X = zeros(2,4);
X(:, 2:3) = rand(2,2)
Y = s*X
```

Save this m-file under the name `randtri.m`. Test it by typing

```
Y = randtri(5); plot2d(Y)
```

Close the graphics window.

Diary File: Now that you have acquainted yourself with MATLAB and created the MATLAB function files, you can begin to work through this assignment. You will need to record the results of your MATLAB session to generate your lab report. Create a directory (folder) in your computer workspace to save your MATLAB work in. Then use the `Current Directory` field in the desktop toolbar to change the directory to this folder. Now type

```
diary lab1.txt
```

followed by the `Enter` key. Now each computation you make in MATLAB will be saved in your directory in a text file named `lab1.txt`. You can then edit this file using a text editor for plain text (no formatting), such as Emacs, NotePad, or WordPad.

Caution: If you use Microsoft Word to edit your diary file, be sure to save the file as **Plain Text**. Do not use the ∗.doc format, which will destroy the alignment of the columns in the matrices generated by MATLAB.

When you have finished your MATLAB session you can turn off the recording by typing `diary off` at the MATLAB prompt (*don't do this now*).

If you want to stop your MATLAB session before completing a lab assignment, you can reopen the diary file the next time you start MATLAB. If you use the same file name, the results of your new MATLAB session will be written at the end of the old diary file. You may prefer to use different names (such as `lab1a.txt, lab1b.txt`) for each session on an assignment, and then merge the files when you prepare your lab write-up with a text editor. Of course, for the other labs you will change the filename to `lab2.txt`, and so forth.

Lab Write-up: Now that your diary file is open, type the comment line

```
% Math 357 MATLAB Project #1
```

at the MATLAB prompt. Type

```
format compact
```

so that your diary file will not have unnecessary spaces. Put labels to mark the beginning of your work on each part of each question, so that your edited lab write-up has the format

```
% Question 1 (a) ...
⋮
% Question 1 (b) ...
```

and so on. Be sure to answer all the questions in the lab assignment. Insert comments in your diary file as you work through each part of the assignment. Don't wait until after you have done all the MATLAB calculations.

Random Seed: When you start your MATLAB session, initialize the random number generator by typing

```
rand('seed', abcd)
```

where *abcd* are the last four digits of your Student ID number. This will ensure that you generate your own particular random vectors and matrices.

BE SURE TO INCLUDE THIS LINE IN YOUR LAB WRITE-UP

The lab report that you hand in must be your own work. The following problems use randomly generated matrices and vectors, so the matrices and vectors in your lab report will not be the same as those of other students doing the lab. Sharing of lab report files is not allowed in this course.

Question 1. Sampling and Quantizing an Audio Signal

(a) Sampling: Consider the function $f(t) = \sin(2\pi(440)t)$ on the interval $0 < t < 1$, where t is time measured in seconds. Sample this function at $8192 = 2^{13}$ points (this is the default sampling rate in MATLAB) to obtain a vector $\mathbf{f} \in \mathbf{R}^{8192}$ with components $\mathbf{f}[k] = f(k\Delta T)$, where $\Delta T = 1/8192$. You can do this in MATLAB by the command

```
f = sin(2*pi*440*(1/8192)*(0:8191));
```

Don't forget to put a semicolon at the end of the command; without it, MATLAB *will print out all 8192 values on the screen and copy them into your diary file.* In this command `(1/8192)*(0:8191)` generates a row vector $\mathbf{t}$ in $\mathbf{R}^{8192}$ with components $0, \Delta T, 2\Delta T, \ldots, 8191\Delta T$.

The sample vector **f** is stored in double precision floating point (about 15 significant figures), so we can consider the components of **f** as real numbers that vary continuously. Type `sound(f)` to play the sound through the computer (you should use headphones if you are working in a computer lab). By default, MATLAB plays all sound files at 8192 samples per second, and assumes that the sampled audio signal is in the range -1 to 1. Our signal satisfies these conditions, and you should hear the *standard tuning pitch* A (440 Hertz) for one second.

The graph of $f(t)$ goes up and down 440 times in the range $0 < t < 1$ corresponding to the frequency of the sound; consequently, plotting the entire signal on one page would give a almost solid black band. To show the sine-wave shape plot just the first 100 samples by the command

```
plot(f(1:100)), hold on
```

(b) 2-bit Quantization: Construct a 2-bit version of the audio signal $f(t)$ with $2^2 = 4$ quantization levels by dividing the range $-1 \le f(t) \le 1$ into four equal intervals. Define the *two-bit quantized vector* **q2f** by

```
q2f = min(floor(2*(f+1)), 3) ;
```

(*Don't forget to type the semicolon ; at the end of the line.*) Look at the first 16 entries in **q2f** by `q2f(1:16)`. Then prove the following by algebra (*hand calculation*):

when $-1 \le f(t) < -0.5$, the corresponding entry in **q2f** is 0 ;
when $-0.5 \le f(t) < 0$, the corresponding entry in **q2f** is 1 ;
when $0 \le f(t) < 0.5$, the corresponding entry in **q2f** is 2 ;
when $0.5 \le f(t) \le 1$, the corresponding entry in **q2f** is 3 .

Thus all the entries in the vector **q2f** are unsigned 2-bit integers $0, 1, 2, 3$ (in binary notation: $00, 01, 10, 11$). A double-precision floating point number requires 64 bits, so the 2-bit quantized signal requires only $2/64$ (about 3%) as many data bits to store or transmit as the original signal.

To construct a 2-bit approximation $\widetilde{\mathbf{f}2}$ to the signal **f** from the quantized signal, we change the values $0, 1, 2, 3$ in **q2f** to $-0.75, -0.25, 0.25, 0.75$ (the midpoints of the four quantization intervals). Define

```
f2tilde = -0.75 + 0.5*q2f ;
```

(*Don't forget to type the semicolon ; at the end of the line.*) Plot the first hundred values of $\widetilde{\mathbf{f}2}$ on the same graph as **f** by the command `plot(f2tilde(1:100))`. Click on Insert and title in the Figure toolbar and insert the title Fig. 1: Sine Wave and 2-bit Quantization (all the figures that you generate in these assignments will need titles). Save the figure and print a copy for your lab report. Play the quantized signal with the command `sound(f2tilde)`. It will sound very harsh compared to `f`.

(c) 2-bit Quantization Error: The difference in the two graphs in Figure 1 is the distortion in the quantized signal, and corresponds to the vector $\mathbf{f} - \widetilde{\mathbf{f}2}$. We can measure the distortion (as a percentage) by the ratio of the energy in the distortion vector to the energy in the original signal:

$$100\|\mathbf{f} - \widetilde{\mathbf{f}2}\|^2/\|\mathbf{f}\|^2 \ .$$

Calculate this ratio in MATLAB by the command

```
distortion2bit = 100*norm(f - f2tilde)^2/norm(f)^2
```

The bad sound generated by $\widetilde{\mathbf{f2}}$ corresponds to the relatively large distortion energy ratio (around 5%).

(d) 4-bit Quantization: Now construct a 4-bit version of the audio signal $f(t)$ with $2^4 = 16$ quantization levels by dividing the range $-1 \leq f(t) \leq 1$ into sixteen equal intervals. Define the *four-bit quantized vector* **q4f** by

```
q4f = min(floor(8*(f+1)), 15) ;
```

(*Don't forget to type the semicolon ; at the end of the line.*) Look at the first 16 entries in **q4f** by the command `q4f(1:16)`. Observe that they are unsigned 4-bit integers $0, 1, \ldots, 15$ (in binary notation: $0000, 0001, \ldots, 1111$). A double-precision floating point number requires 64 bits, so the 4-bit quantized signal requires only 4/64 (about 6%) as many data bits to store or transmit as the original signal.

To construct a 4-bit approximation $\widetilde{\mathbf{f}4}$ to the signal **f** from the quantized signal, we change the values $0, 1, \ldots, 14, 15$ in **q4f** to $-0.9375, -0.8125, \ldots, 0.8125, 0.9375$ (the midpoints of the sixteen quantization intervals). Define

```
f4tilde = -0.9375 + 0.125*q4f ;
```

(*Don't forget to type the semicolon ; at the end of the line.*) Generate a new copy of the graph of the first 100 samples of by

```
figure, plot(f(1:100)), hold on
```

Then plot the first hundred values of $\widetilde{\mathbf{f}4}$ on the same graph as **f** by the command `plot(f4tilde(1:100))`. Click on Insert and title in the Figure toolbar and insert the title Fig. 2: Sine Wave and 4-bit Quantization. Save the figure and print a copy for your lab report. Play the quantized signal with the command `sound(f4tilde)`. It will sound harsh compared to `f`, but not as harsh as `f2tilde`. The sound quality might be acceptable for telephone communication but not for music.

(e) 4-bit Quantization Error: Repeat the calculations in **(c)** with $\widetilde{\mathbf{f}2}$ replaced by $\widetilde{\mathbf{f}4}$:

```
distortion4bit = 100*norm(f - f4tilde)^2/norm(f)^2
```

The improved sound generated by $\widetilde{\mathbf{f4}}$ corresponds to the much smaller distortion energy ratio. Calculate the ratio of the two distortion measurements:

```
distortion4bit/distortion2bit
```

It should be about 1/20.

Question 2. Vector Graphics

This question explores the use of matrices and vectors in two-dimensional computer vector graphics (see Section 1.6 of the textbook). Read the Wikipedia article on *Vector Graphics* if you want more information on this topic. You will need to create some MATLAB *function* m-files. Use the text editor in MATLAB to create and save the following files

(a) Rotations: Create a function m-file to generate 2×2 rotation matrices by using the commands

```
function R = rot(t)
R = [cos(t), -sin(t); sin(t), cos(t)]
```

Save this m-file under the name `rot.m`. Test it by typing

```
R = rot(pi/3); Y = randtri(5);
figure, plot2d(Y); hold on
```

at the MATLAB prompt to generate a random triangle. Then type

```
Y = R*Y; plot2d(Y)
```

The triangle encoded by `Y` should appear in the figure rotated counterclockwise by 60° ($\pi/3$ radians). Repeat this command four more times using the up-arrow key $\uparrow$ to generate six triangles that make a symmetric figure. Be sure that you typed the `hold on` command so that all the triangles appear in the same graphic window. Click on Insert and title in the Figure toolbar and insert the title Fig. 3: Figure with 6-fold Symmetry. Save and print the figure.

(b) Homogeneous coordinates: All linear transformations of $\mathbf{R}^2$ leave the origin $(0, 0)$ fixed. To treat transformations given by *translation* (which move the origin) we have to use *homogeneous coordinates* and 3×3 matrices of a special type (see GOODMAN, Section 1.6). Create the following function m-file to transform a two-rowed vector or matrix `A` into a three-rowed vector or matrix with the third row consisting of 1s:

```
function B = homcoord(A)
n = size(A, 2);
B = [A; ones(1,n)];
```

Save this m-file under the name `homcoord.m`. Test it by typing

```
u = rand(2,1), v = homcoord(u)
```

at the MATLAB prompt. You should get a random vector $\mathbf{u} \in \mathbf{R}^2$ and a vector $\mathbf{v} \in \mathbf{R}^3$. The first two components of $\mathbf{v}$ are the same as those of $\mathbf{u}$ and third component is 1.

(c) Translations: Create the following function m-file to implement translation by a vector $\mathbf{u} \in \mathbf{R}^2$ as a 3×3 matrix:

```
function T = translate(u)
A = [eye(2) u];
T =[A; 0 0 1];
```

Save this m-file under the name `translate.m`. Test it by typing

```
u = 2*rand(2,1), T = translate(u)
```

at the MATLAB prompt. You should get a random vector $\mathbf{u} \in \mathbf{R}^2$ and a 3×3 matrix T, with the third column of T being the vector of homogeneous coordinates of $\mathbf{u}$.

Generate a random triangle and plot it using the commands

```
Y = randtri(3); Z = homcoord(Y);
figure, plot2d(Z); hold on;
```

Now translate the triangle by $\mathbf{u}$:

```
Z = T*Z; plot2d(Z)
```

The graphic window should show your random triangle and the translation of your triangle in the direction of your random vector $\mathbf{u}$ (notice that the function `plot2d(Z)` only uses rows one and two of the matrix Z, which are the homogeneous coordinates of the vertices of the triangle). Use the up-arrow key to repeat the command `Z = T*Z; plot2d(Z)` three times. This should give five triangles in the same graph, each one a translate of the original triangle. Put the title Fig. 4: Translations of a Triangle on the figure. Save and print the figure.

(d) Affine Transformations: The action of a 2×2 matrix A on $\mathbf{R}^2$ is given in homogeneous coordinates by a 3×3 matrix C (see Section 1.6 of the textbook). Create the following function m-file to convert from A to C:

```
function C = affine(A)
B = [A zeros(2,1)];
C = [B; 0 0 1];
```

Save this m-file under the name `affine.m`. Now type

```
S = randtri(3); H = homcoord(S);
figure, plot2d(H), hold on
```

at the MATLAB prompt. Click on `Insert` in the tool bar of the Figure window, and then use the `Arrow` and `TextBox` tools to label the triangle S (you may have to experiment to put the arrow and label in a good position). Now apply translations and rotations to S:

```
u = 7*rand(2,1); T = translate(u); R = rot(2*pi/3); C = affine(R);
plot2d(T*H); plot2d(C*H); plot2d(T*C*H)
```

Create arrows and text labels for the other three triangles in the figure. Identify the different triangles with the correct labels: Translated Triangle, Rotated Triangle, and Translated and Rotated Triangle. You may have to experiment to put the arrows and labels in good positions, but it is a useful skill to acquire so that you can create informative MATLAB presentations. Insert the title Fig. 5: Translated and Rotated Triangles. Save and print the figure.

(e) Matrices and Affine Transformations: Generate a random vector `u = rand(2,1)` and form the translation matrix `T = translate(u)`. Then generate a random 2×2 matrix `A = rand(2,2)` and form the affine version `B = affine(A)`.

(i) Use MATLAB to calculate `A*u` and the matrix product `B*T`. How are the entries in `B*T` related to the matrix A and the vector $\mathbf{u}$? Prove that your answer is valid for *every* A and $\mathbf{u}$ by replacing the numbers in your matrices by symbols and writing out the symbolic matrix algebra calculations on your lab printout, as in Section 1.6 of the textbook. (*Do not do numerical calculations by hand.*)

(ii) Use MATLAB to calculate the matrix product `B*T*B^(-1)`. Observe how the entries in this matrix are related to the vector $A\mathbf{u}$. Then prove that this relation is valid for any A and $\mathbf{u}$ by writing out a suitable symbolic calculation as in part (i).

Final Editing of Lab Write-up: After you have worked through all the parts of the lab assignment, you will need to edit your diary file. Remove all errors and other material that is not directly related to Question 1 and Question 2. **Remove from your diary file any parts of the MATLAB tutorial that you executed**. Your write-up should only contain your keyboard input and the MATLAB output (including Figures 1-5), together with the answers to the questions that you have written.

Preview the document before printing and remove unnecessary page breaks and blank space. Put your name and four-digit ID number on each page. (If you have difficulty doing this using your text editor, you can write this information by hand after printing the report.)

Do not include the codes for the m-files in your lab writeup, but be sure to save these m-files in your personal file space for future use.