SD 372 Pattern Recognition

Lab 0: Matlab Walkthrough
(Ungraded Lab)

1 Purpose

The instructions for this lab are as follows:

1. Create a function to calculate the pdf of a two-dimensional normal distribution.

$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2. Plot:
   - a 3-dimensional picture of the pdf
   - contours showing the portions of the pdf that are the same value
   - a 2-D image that uses colour to show the values of the pdf
3. Plot the region where the pdf is larger than 0.1. On the same graph, plot the mean.

The goal of this lab is to introduce Matlab to those who haven't used it before. In the process, this lab will introduce functions that are essential for the graded labs, as well as provide some insight on the two-dimensional normal (Gaussian) distribution. Some details are intentionally left out so that you get used to using Matlab's `help` command.

The TAs are here to help you get started and to help you out if you get stuck. After you work your way through this lab, you will be well prepared to work on Lab 1. This is not a graded lab so you do not need to hand this in.

2 File Setup

Creating a Directory

Before starting the lab, we want to create a directory to work in. For example, to create the directory **N:/MYFILES/syde372**, enter the following commands into the *Command Window*.

```
>> cd N:/
>> mkdir MYFILES
>> cd MYFILES
>> mkdir syde372
>> cd syde372
```

To verify that you are in the newly created directory, look at the "Current Directory:" under the main menu or enter the following command:

```
>> pwd

ans =
 N:\MYFILES\syde372
```

Creating the Files

Now, we'd like to create two files:

lab0.m – the script that will execute the lab
Gauss2d.m – a function that outputs the pdf of a 2-D normal dist.

First, let's create the script:

1. Select **File** → **New** → **M-file** from the menubar. The M-File Editor will open.
2. Enter the following in the editor:

```
% SYDE Lab 0 - Matlab Introduction
% Name: John Doe  Date: Jan 21, 2004
```

These two lines are comments because they begin with `%`.

3. Select **File** → **Save** from the M-File Editor menubar.

4. Enter the desired filename: **lab0.m**

This script file doesn't do anything yet, but we will add to it once we finish setting up the files we need.

Now let's create the function. Create a new M-File, enter the following into the editor and save it as **Gauss2d.m**:

```
% GAUSS2D 2-D Gaussian pdf
%
% INPUTS:
%  x1 - 1xN vector - range of x1 component
%  x2 - 1xM vector - range of x2 component
%  Mu - 2x1 vector - the mean of the random variable
%  Sigma - 2x2 matrix - the variance matrix of
%                     the random variable
%
% OUTPUT:
%  Y - NxM matrix - the pdf for the range of x1, x2
%
% USAGE (Example):
%  Y = Gauss2d([-3:0.1:3],[0:0.1:6],[0 3]',[1 1; 1 4])

% This function creates the pdf for a 2D Gaussian
%     random variable
function Y = Gauss2d( x1, x2, Mu, Sigma )
```

This file contains many comments and it also declares a function taking four arguments that returns a matrix `Y`. This function has not been defined yet so if we try to run it, the error message shown on Page 4 will be displayed.

It is important to note that Matlab requires the filename and function name to be the same.

Try This

- Type `help Gauss2d` in the *Command Window* and notice what happens. Which comments are *not* displayed?

3 Implementing the Function

Now that we have our files setup, we can work on the function used to calculate the pdf. To gain an understanding of what exactly we want it to do, let's create the script that calls the function first.

lab0.m

Add the following lines to **lab0.m**:

```
clear all  % clear all variables from memory
close all  % close all open figures

% define the mean and variance of the pdf
mu = [0 0]'
sigma = [1 0; 0 1]

dx = 0.5 % step-size
x1 = [-3:dx:3] % range of the random variable x1
x2 = [-3:dx:3] % range of the random variable x2

% this calls the function we create
y = Gauss2d(x1,x2,mu,sigma)
```

Save the file and then enter the following commands in the *Command Window*:

```
>> lab0
>> whos
```

Entering `lab0` at the command line runs the script we created. The `whos` command displays the contents of memory. Notice that all of the variables that are assigned are displayed in the *Command Window*. Use the `help ops` command if you need help understanding what is going on.

Matlab should have also displayed an error message such as:

```
Warning: One or more output arguments not assigned during
call to 'gauss2d'
```

This means that `Gauss2d` isn't returning a value yet and thus we need to work on it.

Gauss2d.m

The purpose of this function is to create a pdf that we can graph. If you look up how graphs are constructed in Matlab, it shows that it is possible to plot a surface (i.e. a pdf) given two vectors and a matrix of values (i.e. the third paragraph shown after typing `help mesh`). So, we want this function to output a matrix that is the appropriate dimension to plot (see Section 4 for more information on plots).

Add this line to the end of **Gauss2d.m**:

```
Y = zeros( length(x1), length(x2) );
```

This initializes `Y` to be a matrix of the correct size. In this function, `Y` is the output because it is on the left side of the equal sign in the function declaration. Once all of the instructions in the function have been executed, the last value calculated for `Y` is returned.

Now, we want fill the matrix with values. The value of the pdf at a point $\underline{v} = [x_1 \;\; x_2]^T$ is:

$$p(\underline{v}) = \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\underline{v}-\underline{\mu})^T\Sigma^{-1}(\underline{v}-\underline{\mu})\right] \tag{1}$$

One way to calculate the matrix is to use two for-loops. In doing so, **Gauss2d.m** will look like:

```
function Y = Gauss2d( x1, x2, Mu, Sigma )

Y = zeros( length(x1), length(x2) );
for i=1:length(x1)
    for j=1:length(x2)
        v = [x1(i);x2(j)];
        Y(i,j) = 1/(sqrt(2*pi)*det(Sigma))*...
            exp(-0.5*(v-Mu)'*inv(Sigma)*(v-Mu));
    end
end
```

Note that the `zeros` command is optional because the loop sets all of the values of `Y`. However, this command tells Matlab how much memory to reserve for the matrix. If this command wasn't included, the size of `Y` would change during the for loop and this function would be less efficient.

Don't forget to save each M-File before using it.

Try This

- Try running **lab0.m** now that the function has been defined. Ensure that you understand how all the variables are generated.
- If you are unclear about what is happening, use the `help` command with the function name (i.e. `help length`), or `help ops` for punctuation.
- What happens if you put semi-colons at the end of variable assignments in the script? (i.e. `mu = [0 0]';`)
- Use the help command to learn about `who`, `what`, `which` and `why`.

4 Plots

The easiest way to demonstrate the plotting techniques is by example. So, let's update **lab0.m** to surpress the variable assignment outputs and illustrate three different methods of plotting. The new version of **lab0.m** is shown on the following page.

```
% SYDE Lab 0 - Matlab Introduction
% Jan 21, 2004
clear all
close all

mu = [0 0]'; % the mean of the pdf
sigma = [1 0; 0 1]; % the variance matrix of the pdf

dx = 0.5; % step-size
x1 = [-3:dx:3]; % range of the random variable x1
x2 = [-3:dx:3]; % range of the random variable x2

% Calculate the pdf
y = Gauss2D(x1,x2,mu,sigma);

% Show a 3-D plot of the pdf
figure
subplot(2,1,1);
surf(x1,x2,y);
xlabel('x_1');
ylabel('x_2');

% Show contours of the pdf
subplot(2,1,2);
contour(x1,x2,y);
xlabel('x_{1}');
ylabel('x_{2}');
axis equal

% Show a colour map of the pdf
figure
imagesc(x1,x2,y)
xlabel('x_{1}');
ylabel('x_{2}');
```

Several new commands are demonstrated. For example, the `subplot` command allows you to put multiple graphs in the same figure. The `xlabel` and `ylabel` show how to label the axes using strings with subscripts. The `surf`, `contour`, and `imagesc` commands are the functions used to create the plots.

For more details about these and other functions, consult Matlab's help.

Run this script and notice how colour is used to indicate the values in the variable y. To find out more about how the colour changes, lookup `help colormap`.

Try This

- Change the value of `dx` to 0.1 and re-run **lab0.m**. The graphs should look a lot smoother now.
- Try using the zoom tools (the magnifying glasses) and rotate tool (the dotted circle with an arrow on it) in the figures.
- Look up the following functions needed to label graphs: `title`, `xlabel`, `ylabel`, `text`, `legend` and `axis`.

5 Plotting Regions

As mentioned in the purpose, we would like to plot the region where the pdf is greater than 0.1, and the mean of the pdf. To do plot a region using the same axes (and plotting techniques) as before, we need a matrix the same size as the one used in the previous section. The easiest way to do this is add this line to the end of **lab0.m**:

```
z = (y>0.1);
```

This compares each element of the matrix `y` with the scalar 0.1. If an element is greater than 0.1, the corresponding element in `z` is assigned to equal 1. Otherwise, that element in `z` is assigned to equal 0. Therefore, if we plot `z` like we did with `y`, then we have successfully plotted the region.

Adding the following to the end of **lab0.m** will plot the region and a yellow point that represents the mean:

```
figure
imagesc(x1,x2,z)
hold on % allow us to plot more on the same figure
plot(mu(1,1),mu(2,1),'y.'); % plot the mean
xlabel('x_{1}');
ylabel('x_{2}');
```

To make this graph look nice, `dx` should be set to 0.1 or less.

Try This

- Display the matrices `y` and `z` in the *Command Window* to make sure you understand how the data is stored in them.
- What happens when you compare two matrices of the same size? i.e. `w = (y>z)`
- Label the mean with a green 'X' instead of a yellow dot (hint: `help plot`).
- Change the colors used to plot the regions.
- Try changing `mu` and `sigma` and see how the pdf changes. You may have to change `x1` and `x2`.
- Redefine `x1` and `x2` to be functions of `mu` and `sigma`.

6 Concluding Remarks

That's the end of this Matlab tutorial! If you understand everything that happened here, you are in excellent shape for the graded labs in this course. If you need help understanding some particular concepts, do not hesitate to contact one of the TAs.

7 Matlab Resources

On the course website[1], there are two Matlab tutorials:

- http://antares.uwaterloo.ca/obadawy/tutorials/matlab/
- http://ocho.uwaterloo.ca/∼pfieguth/Teaching/372/Matlab/index.html

Also, the best place to get references for Matlab is the Mathworks website:

- http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

To learn the aspects of Matlab that are most relevant to this course, consider browsing through the following help topics (examples shown in brackets):

`general`	General purpose commands (`help`, `who`, `pwd`)
`ops`	Operators and special characters (∗, .∗, ∼=, [;], ...)
`lang`	Programming language constructs (`if`, `for`, `function`)
`elmat`	Matrices and matrix manipulation (`eye`, `repmat`, `size`, `find`)
`elfun`	Elementary math functions (`cos`, `exp`, `floor`)
`matfun`	Matrix functions (`det`, `inv`, `eig`)
`datafun`	Data analysis and Fourier transforms (`max`, `sum`, `cov`)
`graph2d`	Two dimensional graphs (`plot`, `hold`, `title`)
`specgraph`	Specialized graphs (`contour`, `imagesc`, `surf`)
`graphics`	Handle graphics (`figure`, `axis`, `close`)
`strfun`	Character strings (`blanks`, `num2str`)
`iofun`	File input/output (`load`, `input`, `pause`)
`timefun`	Time and dates (`tic`, `toc`, `cputime`)
`datatypes`	Data types and structures (`double`, `cell`)
	– cell arrays, {}, are powerful for storing data

[1] http://ocho.uwaterloo.ca/∼pfieguth/Teaching/372/sd372.html