CS515 Spring 08
Prof. Ron

Assignment #1

Prepared by Narfi Stefansson

Due January 29, 2008

Matlab assignment

The Matlab assignments are not intended to be complete tutorials, and they will definitely not explain all the Matlab commands you will need in this course. But we will explain a few of the Matlab commands here, and point out many others that may be useful.
After this first Matlab assignment, you should be familiar with the following commands:

```
help            helpdesk
:  (colon)      linspace
size            plot
print           diary
zeros           ones
eye             xlabel
ylabel          title
sin             exp
+,-,*           ' (transpose)
\ (backslash)   end (as in x(2:end-1))
```

as well the concepts of:
matrices
column vectors
row vectors
scripts
submatrices: A(ind1, ind2) = ...
calling a function
creating a function

If you are familiar with Matlab as well as with the above Matlab commands, skip the preface and go directly to the last page, where the actual assignment is to be found. Otherwise, keep on reading.

Preface

To start matlab, simply type Matlab at the Unix prompt. Matlab should then start, and you get the Matlab prompt, >>.

The most important commands in Matlab are:

- help

- `helpwin`
- `helpdesk`

The first command gives you help inside `Matlab`. The second command gives you a separate window with the help texts, and the third command starts Netscape and gives you access to the extensive HTML help that comes with `Matlab`. If you type in `help helpdesk` at the `Matlab` prompt, you get:

```
HELPDESK Comprehensive hypertext documentation and troubleshooting.
    HELPDESK loads the main MATLAB Help Desk page into the Web browser.
```

Try the following:

```
a=1
size(a)
```

and compare the output with:

```
a=1;
size(a);
```

The commands do the same, but the `; (semicolon)` suppresses printing. You just defined a to be a 1-by-1 matrix. To type in larger matrices, type them in row by row, and use spaces or commas between elements, and use `;` to separate rows. E.g.

```
A = [1,2,3;4,5,6];
```

To look at the contents of the variable `A`, just type it in at the `Matlab` prompt:

```
A
```

gives:

```
A =


     1     2     3
     4     5     6
```

You can access the elements of `A` through regular indexing: `A(2,3)` would give you 6 here.

You can only use an index that exceeds the dimensions of `A` if you are assigning a value to `A`. `Matlab` will not complain about:

```
A(2, 5) = 1000
```

But if you try to get the value of an element that doesn't exist:

```
A(100, 100)
```

`Matlab` will complain.

- `:  (the colon operator)`

The `:` can be used in two ways: To create sequences and for indexing. `1:10` gives the sequence

```
 1     2     3     4     5     6     7     8     9    10
```

The other way of using the colon is to gain access to all rows or columns of a matrix.

```
A(1,:)
```

means: the 1st row of `A` and all columns. Similarly `A(:, 2)` means: all rows of `A`, and the 2nd column.

- `end`

The keyword `end` is useful for indexing (and other things). Look at the following:

```
x = 1:10;
x(3:end) = 7
```

```
 1      2      7      7      7      7      7      7      7     7
```

so here

```
3:end
```

refers to elements 3 to the end of `x`. You can also use e.g.

```
end - 1
```

to refer to the second last element of `x`.

- `diary`

The `diary` command saves the text of your `Matlab` session. `diary on` causes `Matlab` to save all input and most of the output to the file named `diary` until you give the command `diary off`. `diary 000126` will save the input/output in the file `000126`.

- `linspace`
- `plot`

The `linspace` command stands for linearly spaced and is especially useful when generating graphs. Try the following:

```
x = linspace(0,4,50); plot(x, exp(-x))
```

to see the graph of $x \mapsto e^{-x}$. In general, if `x` and `y` are vectors of the same length, `plot(x,y)` plots the vector `y` versus the vector `x`, and `plot(y)` plots the vector `y` against the vector `1:length(y)`.

You can decorate your graphs even further by using:

- `xlabel`
- `ylabel`
- `title`

and use single quote characters to enclose the string:

```
xlabel('The x-axis')
ylabel('The y-axis')
title('The title')
```

It's easy to print to a file in `Matlab`:

```
print -dps myfig.ps
```

tells `Matlab` to print the current figure to the file `myfig.ps` (and overwrite the current contents of the file). If you want to append to the file, use

```
print -append -dps myfig.ps
```

The following commands are indispensable when creating matrices:

- zeros
- ones
- eye

zeros(5,3) creates a 5-by-3 matrix of zeros, ones(3,2) creates a 3-by-2 matrix filled with 1's, and eye(3,2) creates the 3-by-2 upper left corner of an identity matrix. If the matrix is to be square, you can use just one argument; e.g., ones(3) is the same as ones(3,3).

Matrices are truly the building blocks of Matlab, so it comes at no surprise that the arithmetic operators:

- +
- -
- *

work on matrices. Try this:

```
A = [1, 2; 3, 4];
B = [0, 1; 1, 1];
A + B
A - B
A * B
```

and notice that A * B gives you the matrix multiplication of A and B.

It is very easy to solve linear equations in Matlab. The backslash does the trick.

- \ (backslash)

If A is an n-by-n invertible matrix and b is a row vector of length n, the solution to

$$Ax = b$$

is given by A\b.

The following operation in Matlab:

- ´ (transpose)

leads us to the next topic.

The most common mistake in Matlab is beyond any doubt that of confusing row and column vectors. We normally think of a vector as just a vector, but Matlab thinks of it as a matrix. So the question is:
is the vector an n-by-1 matrix or an 1-by-n matrix?

For example, let A = [1, 2; 3, 4]; and b = [1, 0];.

Then Matlab will respond to the command A*b by:

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

because b is of the size 1-by-2 and not 2-by-1 as it should be. Remember that size(b) gives you the size. You can fix this by using any of the following:

- Typing `b` in again, now as a column vector: `b = [0; 1];`
- Transposing `b` by using the t́ranspose operator: `b = b’;`.
- Using the `colon`: `b(:)` returns a column vector, so `b = b(:);` also does the trick.

The last topic for this introduction to `Matlab` is how you can create your own functions and scripts. Suppose you want to create a function called **myfirst** that accepts two matrices as input arguments and returns their sum. Then you would have to put these commands:

```
function sum = myfirst(v, w)
% sum the two inputs
sum = v + w;
```

into a file called `myfirst.m` in your working directory. Once you have saved it, you can invoke the function:

```
myfirst(1, 0)
myfirst([1, 2], [3, 4])
```

If you look back at the function that you saved in **myfirst.m**, the first line:

```
function sum = myfirst(v, w)
```

tells `Matlab` that your function should return the variable `sum`, that its name is **myfirst**, and that it accepts two input arguments. Finally, `Matlab` will print out that percented first line any time you say `help myfirst`.

You should also try to put the following into a file called, say, **firstscript** in your working directory:

```
A = eye(3);
b = ones([3, 1]);
x = A\b;
```

and then type `firstscript` at the `Matlab` prompt. This simply runs the commands in the file.

Remember that more information on `Matlab` can be/will/was obtained from (i) the first class demo, (ii) the `Matlab` primer (at DoIT), (iii) the extensive online `help`.

It’s time to practice the above notions...

Assignment

1. Generate a plot of the function `sin` on the interval $[-1, 1]$ with an appropriate title. Turn in the figure together with the commands you used to generate the figure.

2. Find the solution to the following 20-by-20 system of equations. Turn in your code and the output.

$$\begin{pmatrix} 1 & & \dots & 1 & 0 \\ & 1 & & & \\ \vdots & & \ddots & & \vdots \\ & & & 1 & 0 \\ 1 & & \dots & & 1 \end{pmatrix} x = \begin{pmatrix} 17 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

   [Note: All entries in the matrix not explicitly mentioned are meant to be zero].

3. Create a `Matlab` function called **ssolve** that:

   (a) Accepts three parameters, a, b, and n, in that order.

   (b) Returns the solution to the following n by n system:

$$\begin{pmatrix} 1 & a & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ b & b & \dots & b & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ n-2 \\ 0 \\ n \end{pmatrix}$$

   as a column vector.

   [Note: All entries in the matrix not explicitly mentioned are meant to be zero].

4. Turn in a printout of the function **ssolve** and plots of the vectors

   (a) **ssolve**(10, 0.1, 30)

   (b) **ssolve**(2, -0.05, 50)

   with the appropriate titles. Also turn in the code that generated these figures.