MATH 246, WORKSHEET 2: VECTORS, GRAPHS, AND A PROGRAM

The main features this week are working with vectors, producing graphs, and the beginnings of programming and good programming style.

1. Topics

Reference below are to the suggested reference book, *Mastering* MATLAB.

- Machine numbers: precision and range of values
- Vectors: notation and basic operations [Chapter 6]
- Manipulating strings (vectors of characters) [Chapter 6]
- Producing graphs of data stored in vectors [Chapter 26]
- Refining the appearance of graphs [Chapter 26]
- Programming style: comments in M-files and messages from programs [Chapter 14]
- Interactive input and output in M-file programs [Chapter 14]
- Decision making in programs: the `if` command and arithmetic comparisons [Sections 13.3, 8.1]

2. Exercises

Indicate the MATLAB commands used to get your results as well as the results themselves.

(1) Machine numbers

Read what MATLAB help says about the special variables `eps`, `realmin` and `realmax`, and then check their values. Express them as powers of 2 [recalling $\log_2 x = (\ln x)/(\ln 2)$] and compare to the values discussed in class for machine numbers.

Then divide `realmin` by 2 and comment.

(2) Vectors and graphs

(a) See what vectors are produced by the expressions

- `[ 1 5 8 ]`
- `[ 2, 5, -3, 6 ]`
- `[ 2; 5; -3; 6 ]`
- `1:3`
- `1:2:7`

and store these vectors in variables named `a1` to `a5`.

Date: September 7, 2006.

(b) Describe what the above notations appear to do in general, and test your ideas by finding a compact way of creating the vector

```
21 19 17 15 13 11 9 7 5 3
```

(c) Compute `a1+a4` and `a2-a5` for the vectors above.

(d) The *pointwise product* or *array product* of two vectors is the vector, of the same length as the two original vectors, whose elements are the products of their corresponding elements; for example, the pointwise product of `[2, 6]` with `[3, 8]` is
`[2×3, 6×8] = [6, 48]`. Such products are useful in evaluating function values for graphing.
Find the best way to compute the pointwise product, and test it with `a2` and `a5`.

(e) Note that you can *concatenate* vectors as follows:

```
a1a2 = [a1 a2]
```

Note: MATLAB treats text strings as vectors of characters, which explains what happened in Exercise 3 of Worksheet 1.

(f) What happens when you apply a function [like `sin`] to a vector? Try with one of the vectors above. Then create a vector variable named `x` containing the integers from zero to seven, and store the sine of this vector in the variable `y`.

(g) What does `plot(x,y)` do?

(h) How can you get a better looking graph of $y = \sin x$ on the interval $[0, 2\pi]$?

(i) Investigate the commands `grid`, `title`, `xlabel` and `ylabel` and use them to refine your graph of sine. Then print the graph using the menu command **File▷Print...**

(3) Programming: solving quadatics

The M-file script `quadroots` that you created last week can be improved in many ways, to produce a first good example of MATLAB programming.

After saving your original version under another name (say `quadroots1`) as a backup, make the following improvements and additions. If you finish them all with time to spare, I can suggest more things to try!

(a) Make the script more *flexible*: make it solve different quadratics without needing to edit the file each time. Do this by using interactive input of the quadratic's coefficients, using the MATLAB command `input`].

(b) Make the script more *user friendly*: have it print messages to explain what it does, what the input variables mean, and what the output values signify: use commands like `disp`.

(c) Make the script itself *readable and understandable* to another person (or to you if you reread it some months later): start with *comments* on what the script does and a bit about how it does it; add comments at places within the script that are not self-explanatory; and try to choose reasonable, "self-documenting" names for variables and such.

(d) Add *help documentation* to your script, so that the command `help quadroots` does something useful. (You might already have done this in the previous step without realizing!)

(e) Make the script *robust*: test it for hard cases (a single root, complex roots, $a = 0$) and make sure that it either gives correct answers or a message explaining why it could not.

Print the final version of the script, the dialogue you get when running a few test cases, and at least one sample graph. The command `diary` could be useful for some of this.