Lab01: Introduction to Matlab

EECS20: Introduction to Real-Time Digital Systems

©1996 Regents of the University of California.

By K. H. Chiang, William T. Huang, Brian L. Evans.

The last three exercises are copied from *Labs for Signals and Systems* by Virginia Stonick and Kevin Bradley. URL: `http://www-inst.eecs.berkeley.edu/~ee20`

News: `ucb.class.ee20`

Assigned: 29 Jan 97, Checkoff: 5 Feb 97, Writeup Due: 7 Feb 97

1 Matlab

Half of the laboratory assignments will use Matlab. Matlab stands for "Matrix Laboratory" and is a general-purpose programming language. Matlab interprets commands like Lisp does, instead of compiling source code like C and Fortran require. Matlab, unlike Lisp, has an extensive sets of routines, or toolboxes, that perform matrix, signal processing, image processing, controls, optimization, and statistics computations.

By building your own expertise in Matlab, you will be able to attack numerical problems in engineering quickly by using its high-powered routines and by avoiding the compile-link-debug cycle in C and Fortran.[1] Since Matlab is available on a variety of different platforms, your Matlab scripts can run without modification on several different kinds of computers.

Here are a few key points to keep in mind when you're using Matlab:

- Variables in Matlab, as in C or Fortran, must start with an alphabetic character, and must contain only alphabetic characters, numeric characters, and the underscore character, e.g. `dude` is okay, but `dude!` is not. Matlab is also case sensitive, e.g. `dude` and `Dude` are not the same identifier.

- Matlab has only one data type: a complex-valued floating-point matrix. A vector is simply a matrix with either one row [a row vector] or one column [a column vector]. A number, or scalar, is simply a 1×1 matrix.

```
>> rowvector = [1 2 3]
rowvector =
     1     2     3
>> columnvector = [1; 2; 3]
columnvector =
     1
     2
     3
```

[1] However, you take a hit in performance. This is why Matlab is primarily used for prototyping purposes, after which code is migrated to faster environments. Matlab has recently released a compiler to remedy this problem, but it only offers a two to three times speedup over interpretation.

```
>> matrix2x2 = [1 2; 3 4]
matrix2x2 =
     1     2
     3     4
```

- Matlab works with matrix-vector operations; e.g., to add two vectors,

```
>> [1 2 3 4 5]  +  [1 1 1 1 1]
ans =
     2     3     4     5     6
```

- The Matlab language is a cross between a scripting language and C, plus matrix operations:

```
>> fid = fopen('x.dat','wb')
>> fwrite(fid, x, 'double')
>> fclose(fid)
```

  However, Matlab is similar enough to C to cause some extremely annoying errors. One such error is that array and vector indexing begins with 1 and not 0.

```
>> x = [1:2:9]
x =
     1     3     5     7     9
>> x(0)
???  Index exceeds matrix dimensions.
>> x(1)
ans =
     1
```

- Matlab *does not* have pointers or data structures, but it does have automatic memory allocation/deallocation. For example, in solving for x in $Ax = y$, Matlab will automatically create a vector for x:

```
>> A = [1 2 1; 2 1 3; 4 0 5];
>> y = [3; 2; 1];
>> x = A \ y
x =
           0
      1.4000
      0.2000
```

  Unfortunately, its lack of data structures will make handling of multidimensional arrays messy [this problem is addressed in `http://www.mathworks.com/1109.html`.]

- Unlike C but like Lisp, Matlab can return multiple values for a call to a function. The number of return values you ask for makes the function behave differently. The following example involves computing eigenvalues and eigenvectors for the matrix $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$; i.e, factor the matrix into $(\texttt{eigenvectors})(\texttt{eigenValueMatrix})(\texttt{eigenvectors})^*$:

```
>> eigvalues = eig( [ 1 2; 2 1 ] )
eigvalues =
   -1.0000
    3.0000
>> [eigvectors, eigenValueMatrix] = eig( [ 1 2; 2 1 ] )
eigvectors =
    0.7071    0.7071
   -0.7071    0.7071
eigenValueMatrix =
   -1.0000         0
         0    3.0000
```

  Don't worry if you don't know what an eigenvalue or eigenvector is.

- Iteration is very, very slow in Matlab, so do not use `for` loops. Instead, try to write expressions using matrix operators. For example, to add two vectors, we should use the '+' operator:

```
>> c = x + y
```

  The WRONG WAY would be to reimplement the add operation by defining a new function called `add` in a file called `add.m`:

```
function c = add(x, y)
%% Function to add two vectors x and y together

len = max( [ length(x) length(y) ] );
for i = 1:len
  c(i) = x(i) + y(i);
end;
```

  In Matlab, you would access the new function by typing

```
>> add( [1 1], [2 2] )
```

  Matlab will automatically try to load functions that it does not recognize by searching for the file *function*.m on its search path. But in this case you are reinventing the wheel and doing a bad job of it too.

  A short and coherent tutorial on vectorization can be found at
  `http://www.mathworks.com/1109.html`.

Other technical tips are available from Matlab's creators at
`http://www.mathworks.com/technical.html#general`.

In addition, you can have further fun by running the Matlab demo:

```
>> demo
```

Enjoy the pretty pictures. Drive those Pentiums into the ground.

2 Questions

1. Familiarize yourself with Matlab on the PC. UC Berkeley has a site license to run Matlab on all Unix workstations, so all machines in instructional clusters have access to Matlab. This assumes that you have an account on the instructional machines though. [If you don't have an account, you can get one by following the instructions posted on bulletin boards around 105 Cory. All students taking EECS courses are entitled to such an account.]

2. Familiarize yourself with Chapters 1-3, 5, 7, and 8 in *Mastering Matlab*, as you will be applying the concepts and language constructs therein. Note also that this course will *not* be an exercise in who can memorize what Matlab function. Instead, you should try to use existing Matlab commands when possible so as to get in the habit of reusing existing code instead of writing your own. Code reuse is a fundamental concept in software engineering, and will prevent you from reinventing and debugging the wheel repeatedly, when you could be building a Lamborghini instead [well, maybe a Dodge Dart].

3. Make use of comments with the `%` sign [so that you can figure out what you did when you look at your code at a future date]. Use `help` and `lookfor` when necessary.

4. Define a sine wave with frequency 10 Hz and zero phase over the time interval from $t = 0$ to $t = 2$ at increments of 0.01. Call this signal x. Then, plot the signal over the same time interval, making sure that the axes are correctly labeled. Is this a continuous function?

5. Next, define a new signal y that is the signal x multiplied by a decaying exponential term e^{-2t}. Form and plot the sum and product of x and y on the same interval with the same increment.

6. Write a Matlab function that finds and replaces all negative entries in a matrix with zeros.

7. Write a Matlab function `half` that removes every other element from an arbitrary length vetor, creating a shorter vector made of only the odd-numbered elements of the original vector. Use only matrix and vector manipulations; do not use loops.

8. Write a Matlab function `double` that creates a longer vector by adding an additional element, 0, between neighboring elements in the original vector. Then replace each of these new 0 elements with the average of its neighboring elements. Use only matrix and vector manipulations; do not use loops.

9. What happens to a vector x in `half(double(x))` and `double(half(x))`?