# Rob's Spring 2011 Math313/513 at Penn, Computer Problem Set 1

These notes and problem sets rely heavily on those prepared by Dr.Michael Robinson, who taught the course at Penn in Spring 2009. We are grateful for his guidance and generosity in an area where we are *much less* experienced (though we are responsible for any goof-ups)! Also helpful is Cleve Moler's MATLAB guidebook (http://www.mathworks.com/moler/chapters.html).

In Matlab, by which we mean MathWorks MATLAB (buy a student license or use for free in the SAS computer labs) or GNU Octave (freely available, like the GNU Emacs editor), almost everything is a matrix or a vector. For example, here's how to specify our favorite 4x3 matrix:

```
>> A=[1 2 3; 4 5 6; 7 8 9; 10 11 12]
```

**Problem 1** How do we specify the transpose of this matrix $A$?

In case we need help, the command

```
>> help
```

is helpful for finding out about other commands or their arguments. Nearly everything in Matlab is self-documenting, so we can figure out much of the language ourselves as we go. (As my Dad used to say when he got in trouble: Help! Help!)

Vectors (row or column) can be entered by hand, or using tricks like

```
>> y=1:.3:25
```

which makes a row vector (a 1xWHAT matrix?) whose entries (from left to right) start at 1, increment by 0.3 and are no larger than 25.

Before going further, it's good to record what we've done. The way we do this is to list the commands we want executed one after another in a plain text file with extention ".m", called a "script". We can (and should) insert comments with a percent sign to clarify what we've written. MATLAB has a convenient editor for writing scripts. Start it by typing

```
>> edit
```

and then cutting and pasting commands into the window. We can later execute the commands in a script by typing its filename (without the ".m") at the Matlab prompt.

**Store your results for Problem 1 in such a script; you'll be printing that out with other results to turn in with your homework from Strang!**

Matrices can be loaded into Matlab in a variety of ways. The easiest is to simply list the entries with semicolons to separate rows; here's the matrix that rotates the plane by an angle you can adjust:

```
>> theta=pi/4;
>> A=[cos(theta) sin(theta); -sin(theta) cos(theta)]
```

Try some other values of $\theta$ (theta) to be sure this works.

If we want to know how big a matrix is, we can use the "size" command:

```
>> size(A)
```

tells us that $A$ is 2x2. Obvious in this case, but this can be very useful for very large matrices.

If we want a matrix with random entries, for instance a 7x4 matrix, use

```
>> B=rand(7,4)
```

To get a new random matrix of the same size, we can use

```
>> C=rand(size(B))
```

There are other useful matrix-making commands, such as "zeros", "ones", "eye", and "meshgrid". Read about them! (Matlab can even read almost any data file format you can imagine, like JPEG images! Look into the documentation....)

We can use matrix operators to do useful things to matrices. For instance, we can multiply two random matrices

```
>> rand(5,3)*rand(3,4)
```

Try it!

In Matlab, we have easy access to Gaussian elimination. It's essentially matrix division, so Matlab represents it by "\". To solve $Ax = b$, we execute

```
>> x=A\b
```

Thinking of the "fraction" $\frac{b}{A}$ as $A^{-1}b$ reminds us to use backslash from the left. (If we want to write $bA^{-1}$ for $b$ a row vector, we use (forward) slash from the right

```
x=b/A
```

which results in $x$ being a row vector....)

Matlab also provides a way to calculate matrix inverses using the "inv" command. As we mentioned in class, matrix inversion is not (numerically) the same as Gaussian elimination. In this problem, we examine this a little (courtesy of the MATLAB documentation). Type the following into Matlab:

```
>> n = 500;
>> Q = orth(randn(n,n));
>> d = logspace(0,-10,n);
>> A = Q*diag(d)*Q';
>> x = randn(n,1);
>> b = A*x;
```

Now, we've got a matrix $A$ (yes, it's invertible) with a random vector $x$, and $b$, their product. The task is to recover an approximation of $x$.... Obviously, $x = A^{-1}b$, so try

```
>> y = inv(A)*b;
```

Now, using Gaussian elimination, we should execute

```
>> y2=A\b;
```

Now these two aren't very different, but compare the error

```
err = norm(y-x)
```

with

```
err2 = norm(y2-x)
```

It's not so different.... But the residuals

```
res = norm(A*y-b)
```

and

```
res2 = norm(A*y2-b)
```

differ significantly!

Matlab provides the ability to define new functions, which is an important way to manage complicated programs. A function has a number of inputs and a number of outputs.... Like scripts, functions are stored in plain text files with the extension ".m". However, the first line of a function file must begin with the word "function" and then an input/output specification. For instance, the following could be stored in "helix.m"

```
function [x,y,z]=helix(t)
% Compute points on a parametric curve (a helix)
% Input: t = matrix or vector of parameter values to evaluate
% Output: x,y,z = coordinates of points computed
x=cos(t);
y=sin(t);
z=t;
```

Notice that the function name in the top line must match the filename! If this file is stored in Matlab's current directory, we can call it:

```
>> [x,y,z]=helix(0:0.1:3*pi);
>> plot3(x,y,z);
```

(Discrete differentiation and integration.) Suppose $v$ is a column vector, which might represent samples of data. If we wanted to approximate a running integral of $v = \int_0^t v(x)dt$, we might compute a cumulative sum of $v$. (There's a Matlab command to do this, "cumsum", but we won't use it for this problem.) This can be computed by multiplying $v$ with a lower-triangular matrix of ones:

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & ... \\ 1 & 1 & 0 & 0 & ... \\ & & ... & & \\ 1 & 1 & 1 & 1 & ... \end{pmatrix} \tag{1}$$

Here's how to make such a matrix in Matlab:

```
>> [cols,rows]=meshgrid(1:length(v),1:length(v));
>> C=1.0*(rows>=cols);
```

which says make $C$ a matrix with entries=1 when the row number is greater than or equal to the column number. It's a square matrix that's as wide as $v$.

**Problem 2** What matrix $D$ would we multiply $v$ by to compute a vector which contains the differences between adjacent elements? Write a Matlab function to construct $D$ given $v$ (or the size of $v$).

**Problem 3** The Fundamental Theorem of calculus states that differentiation and integration are inverse operations. An analogous statement in the discrete context is that $D$ and $C$ are inverses of each other. Is this true for the matrix $D$ we computed? Why or why not? What is the invese of $C$ and how is it different?

Hint: For a few random vectors $v$, compute $(DC - I)v$ ($I$ is the identity matrix) to tell what the matrix $C$ is doing.

**Remember to put your commands into a script and print it out, along with any functions or figures. Please hand these in with your problems from Strang!**