

MATH 313 Spring 2009, Homework 5, Computer-based problems

Matlab is very handy for finding properties of linear maps. In particular, it very easily allows you to compute row-reduced echelon forms, ranks, etc. Indeed, to compute the row-reduced form of a matrix, you just use the command “rref”. To find its rank, use “rank”.

Problem 1 Compute the row-reduced echelon form for

$$A = \begin{pmatrix} 1 & 3 & 4 & 8 & 8 & 8 \\ 2 & 2 & 9 & 5 & 0 & 5 \\ 2 & 0 & 5 & 2 & 7 & 7 \\ 6 & 7 & 4 & 7 & 4 & 4 \end{pmatrix}$$

both by hand, and using Matlab. What is its rank? Is this a reliable estimate of its rank? Why or why not?

Although Matlab can often avoid loops through vectorization, it does have loops and conditionals much like other programming languages. In this assignment, we’ll make use of them... They don’t make too much sense outside of a function definition or a script file, so I’ll assume you’re writing these commands in a script or function.

We’ll start with the “if” statement. Here’s an example:

```
if( a == 1 )
    x=5
else
    x=1
end
```

which says, “if a is 1, then set x to 5, otherwise set x to 1”. Notice that to check equality, we use “==”, but to assign a variable a value, we use “=”. This is a common source of frustration for new Matlab programmers! Instead of “==”, we can check for non-equality by “≠”, or use “<” or “>” for less-than or greater-than. You could also leave off the “else” clause entirely

```
if( x == 1 )
    disp('a must have been something other than 1')
end
```

So much for “if” statements. Loops are introduced with “for” statements. Here’s an example of a “for” loop in Matlab

```
for j=[2 3 5 7 11 13]
    j.^2
end
```

It executes the expression j^2 for the following values of j : 2, 3, 5, 7, 11, and 13 (in that order). You can put any row vector you like in there, though:

```
for evens=0:2:26
    evens
end
```

Sometimes you just want to repeat something a bunch of times with some random data... For instance, here’s an approximation to π

```
points_in=0;
for i=1:10000
    x=rand;
```

2

```
y=rand;  
if( sqrt(x.^2+y.^2) < 1 )  
    points_in=points_in+1;  
end  
end  
disp(4*points_in/10000)
```

Problem 2, Part a Modify the above example to find out the percentage of 4x5 random matrices (with entries between 0 and 1) are of full rank.

Problem 2, Part b Try to figure out how this percentage changes as the size and shape of the matrix changes.

Problem 3 Repeat Problem 2 (both parts) by instead having random integer entries between 0 and k for some fixed k . How is this different from the results of Problem 2 (especially when k is small)?

Hint: to get a random number between 0 and 9, use a command like `round(rand*10)`