LECTURE NOTES CS 321: Lecture 1

Matlab preliminaries

What is Matlab?

Matlab stands for Matrix Laboratory. It is a special programming and interpreter language that makes it possible to perform exploratory numerical calculations with a lot of built-in mathematical support (for example, matrix diagonalization, optimization, solving differential equations).

In the class of "Computational Molecular Biology" we will use MATLAB tools and programs to study molecular biology problems. Using MATLAB will help avoiding writing complex and long numerical codes.

MATLAB is an interpreter. This means that we can get an intermediate answer after every single step. Normally in computer languages like C or Pascal we must have a whole program ready before we can get an answer. In that sense MATLAB is working like a good debugger which makes it possible for us to build the code in small steps and check our progress continuously as the project progresses.

As an interpreter we can do calculator-like calculations in MATLAB in the same ease as on true calculator.

Basic MATLAB operations

For example, if I type "pi" into MATLAB main window the result is

```
>> pi

ans =

   3.1416

>>
```

The ">>" is the prompt, where we type the next instruction. The line without the ">>" is the computer output. Note that MATLAB stored the value of pi in a new variable called "ans(wer)" If we now type ans we get

```
>> ans

ans =

   3.1416
```

Note also that if I continue and type

```
>> 10+1

ans =

   11
```

The value of the “ans” variable will be changed. It makes sense therefore to store the results of the calculations someplace else for example

```
>> volume = (4.*pi*5.^3)/3

volume =

  523.5988
```

Q: Do we need the brackets?

Stores for us the volume of a sphere with a radius of 5. The usual algebraic ordering holds (i.e. exponentiation is coming before a multiplication). Sometimes we do not want to see all of the above “junk”, i.e. the MATLAB echoes of the current result. To eliminate the echoes type “;” at the end of the line, for example,

```
>> volume = (4.*pi*5.^3)/3;
>>
```

There is no echo of the result. However, the value is stored at “volume” and to see it type:

```
>> volume

volume =

  523.5988
```

MATLAB allows some “funny” operations. Like an inverse division

```
>> % an inverse division is defined by A\B and it means B/A
>> 3\1

ans =

   0.3333

>>
```

A “%” means that whatever is coming after the “%” (in the same line) is a comment.

The inverse division is useful when the objects are matrices, (as we shall discover later). You probably discover by now that the accuracy of "pi", as printed, was quite depressing. However, all calculations in MATLAB are done in double precision and if an impressive looking printout of pi is desired, we may use:

```
>> format long
>> pi

ans =

   3.14159265358979
```

By default the format is short.

Vectors and matrices

Vectors and matrices in MATLAB do not require declaration. For example, we can write

```
>> x = [1,2,3];
>> y = [2;3;4];
>> x*y

ans =

    20

>> y*x

ans =

     2     4     6
     3     6     9
     4     8    12
```

The first line defines the vector x to be a row vector with elements 1,2,3 -- $x = (1, 2, 3)$

The second line defines the vector y as a column vector with elements 2,3,4 -- $y = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$

A semicolon always opens a new column. Note that the attempt below is a mistake, as explained by the MATLAB output listed below

```
>> y = [1,2,3;3;4]
??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.

>>
```

The multiplication x*y means the inner product of the two vectors: i.e.

$$x * y = 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 = 20$$

However the tensor product of y*x means something very different:

$$y * x = \begin{pmatrix} 1 \cdot 2 & 2 \cdot 2 & 3 \cdot 2 \\ 1 \cdot 3 & 2 \cdot 3 & 3 \cdot 3 \\ 1 \cdot 4 & 2 \cdot 4 & 3 \cdot 4 \end{pmatrix} = \begin{pmatrix} 2 & 4 & 6 \\ 3 & 6 & 9 \\ 4 & 8 & 12 \end{pmatrix}$$

The above two are common matrix/vector products. MATLAB makes it possible to do a lot more. For example, the element by element operation "." (dot)

```
>> x.*y'

ans =

    2     6    12
```

which corresponds to the following operation

$$x.* y' = (1 \cdot 2, 2 \cdot 3, 3 \cdot 4)$$

Note that the dot operation is working only for matrices with the same dimension (vectors are just matrices with one of the dimensions shrunk to 1). The "prime" in y' is to impose transposing of the vector, without it the "dot" operation will not work!

Q: why?

Q: There is another way of performing a "dot" operation when the dimension of x and y is not the same. Can you suggest when different matrix dimensionality will not be a problem?

It should be suggestive by now what the following operations mean:

sum(x) length(x) sin(x)

In fact there is an extensive help facility in MATLAB

```
>> help sum

 SUM Sum of elements.
    For vectors, SUM(X) is the sum of the elements of X. For
    matrices, SUM(X) is a row vector with the sum over each
    column. For N-D arrays, SUM(X) operates along the first
    non-singleton dimension.

    SUM(X,DIM) sums along the dimension DIM.

    Example: If X = [0 1 2
                     3 4 5]

    then sum(X,1) is [3 5 7] and sum(X,2) is [ 3,12];


    See also PROD, CUMSUM, DIFF.

 Overloaded methods
    help sym/sum.m
--------------------------------------------------------------------------------
```

The difficulty with the help command is that you need to know the name of the command in advance. If I am looking for a command that perform optimization, I do not know what the name of the command should be … In that case a useful command is “lookfor” which is essentially a keyword search throughout all the help documentation. It is taking longer but it extracts it all. The output may be long but it start with the name of the file / command description so the next “help” command can be used more effectively.

```
>> lookfor optimization
FOPTIONS Default parameters used by the optimization routines.
CHAINGUI Chain matrix multiply optimization GUI.
CHAINMULT Optimization of chain matrix multiply problem.
TRIMFCN Used as a gateway to the optimization routine SIMCNSTR
OPTPORT Portfolio optimization demonstration.
CONVERTM converts constraint matrix into optimization format.
COSTFUN Cost function for NCD Blockset optimization.
GRADFUN Gradient of the Cost function for NCD Blockset optimization.
NCD1INIT Sets up necessary data files for optimization of ncddemo1.
NCD2INIT Sets up necessary data files for optimization of ncddemo2.
NCD3INIT Sets up necessary data files for optimization of ncddemo2.
NCD4INIT Setup for optimization of ncddemo4.
NLINOPT Runs the optimization algorithm.
PARAMDLG Manages a dialog box for NCD Blockset Optimization Parameters.
```

ATTGOAL Solves the multi-objective goal attainment optimization problem.
DEMOS Demo List information for Optimization Toolbox
DFILDEMO Nonlinear filter design problem using discrete optimization.
FGOALATTAIN Solves the multi-objective goal attainment optimization problem.
FSEMINF Solves semi-infinite constrained optimization problems.
GRADERR Used to check gradient discrepancy in optimization routines.
optdeblur.m: % This demo shows how the Optimization Toolbox can be used to
optdemo.m: % ------- OPTIMIZATION TOOLBOX Demonstrations---
OPTDEMS Set up Optimization demos for The MATLAB Expo.
OPTSIMINIT Sets up necessary data files for optimization of optsim.
semicon.m: %SEMIFUN Translation routine for semi-infinite optimization.
SEMIFUN Translation routine for semi-infinite optimization.
SEMINF Solves semi-infinite constrained optimization problems.
TUTDEMO Tutorial for Optimization Toolbox.
DISPLAY1 displays progress information during optimization.
>>

Another important feature is the ability to run loops. For example:

```
>> for x=1:9
y =y + x;
end
>> y

y =

   45

>>
```

Note that x is increment by one more complex options are possible. For example:

for x=0:0.1:10

Which use steps of size 0.1 to interpolate bewteen 0 and 10.

Or

for x=10:-1:1

Here 10 is the beginning. The increment is –1 and the final target is one.
Matlab is considerably more flexible compared to other languages by allowing a mixture Of integer and floating point numbers.

How can I create a vector x = [1,4,7,10]? One way, x = [1:3:10]

Another way to loop in MATLAB is the while command. It is of the form:

while condition
    expression
end

A condition can be (for example) a>3)

MATLAB also has a branching command “if / elseif / end”

Section material

1. Find out what the function linspace can do for you.
2. Replace the loop
   For i=1:10
       x(i) = i;
   End
   By appropriate linspace command that produces the same x vector
3. What the symbol “~=” means?
4. Create two vectors of length 101 x=t^2 and y=sin(t) where t is equally spaced between 0 and 100. Find a way of generating a two-dimensional plot of x against y.

Homework

1. Let $x_i = a+(b-a)i/N$, $i = 0,1,2,\ldots,N$ be $N+1$ evenly-spaced points on the interval $[a,b]$. The distance between consecutive points is $h = (b-a)/N$. The derivative of a function f on this interval can therefore be approximated by $f_p = [f(x_{i+1})-f(x_i)]/h$. One way to measure the error in this approximation is to compute $e_N = \sum_{i=0}^{N-1}[\, f_p(x_i) - f'(x_i)]^2$, where f' is the true derivative. Compute e_N for $f(x) = sin(x)$ on the interval given by $a = 0$ and $b = 2\pi$. You may define only two variables. One of them must be N (e.g., $N = 100$). For what values of N is e_N smaller than 10^{-3}?
2. Compute an approximation to π as follows. Construct two N-by-N matrices, x and y, satisfying x(i,j) = j – 1 and y(i,j)=i-1, for i,j = 1, 2,…,N. These matrices represent the coordinates of a square grid whose sides have length N-1 and whose center is $x_c=(N-1)/2, y_c=(N-1)/2$. Construct a matrix c with the property that c(i,j) = 1 iff the point (x(i,j),y(i,j)) is inside (or on) the circle of radius N/2 centered at (x_c,y_c); otherwise c(i,j)=0. Use c to approximate π. It’s easier to do this exercise with N odd. Show your results for N=11 and N=101 (the approximation won’t be very good for these N). The only matlab functions you may use are: linspace,

reshape, mod, and sum. You may not use any loops. Remember your dot notation!

Let’s break this problem down into parts, for the case N=11: Construct the matrix x as follows. First let x be a vector from 0 to N^2 (i.e., 121). Then reshape x so that it is an 11-by-11 matrix. Then use modulo and transpose to make x have the property x(i,j)=j-1. Now build the matrix y similarly. Read up on relational operations in the text to build the matrix c. Finally, read ‘help sum’ (or read the text) to learn how to use sum to count the number of 1’s in matrix c.