Lecture 18: Crash course on Matlab

Gantumur Tsogtgerel

Math 319: Introduction to PDE
McGill University, Montréal

Monday February 14, 2011

The interface

The main window of Matlab is called **Command Window**, and the line that starts with » is called **Command Line**. This is where you type commands in, and get simple text responses from Matlab. Click *Enter* to execute a command, and use *Up* and *Down* arrows to go over the commands you recently entered.

There are many other windows (manually activable from Desktop menu).

- **Workspace** shows the variables that are present in the system memory.
- **Command History** displays the list of commands you ever entered.
- **Current Folder** shows the content of the current folder.
- **Figures** window is activated when a picture needs to be displayed.
- **Editor** window is used to edit an M-file (which is a way to store a sequence of Matlab commands together so that one does not need to type them in if at a later time one needs that particular sequence of commands).

Variables

A **variable** is a group of boxes in computer memory that has

- an identifiable name,
- a type, that determines what sort of information the boxes contain,
- a value, which is the content of the boxes.

In Matlab, assigning a value to a variable name immediately creates a variable with that name. For example, the following command creates a variable named `i` and assigns it the value 3. (The type is “number”.)

```
>> i=3;
```

To check the value of the variable `i`, give the following command in Matlab command line:

```
>> i
```

One can assign a new value to an existing variable:

```
>> i=4;
```

Numbers

Reading the value of an existing variable, and assigning it to another variable:

```
>> j = i;
```

One can manipulate the value before assigning:

```
>> j = i+3;
```

All arithmetic operations are allowed.

```
>> x = (26-3.4*3)/(23+4.5);  y = 2^8;
```

There are some built-in constants.

```
>> x = pi/2;
```

Confusing at first sight, but perfectly legal (and vital):

```
>> i = i+2;
```

Matrices

Matlab's real specialty is in working with matrices. Matrix is the fundamental data type in Matlab. One way to create a matrix is:

```
>> A = [1 1; 3 4]; B = [2 3; 6 4];
```

Standard operations are supported (for matrices with compatible sizes):

```
>> C = A+B; D = A*B; E = A-B; F = inv(A);
```

Handy notation to solve a matrix-vector equation (aka linear system):

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

Accessing individual rows, columns, and all columns as one:

```
>> r1 = A(1,:); c2 = A(:,2); g = A(:);
```

Elementwise operations:

```
>> C = A.*B; E = A./B;
```

Special matrices

Arithmetic progressions:

```
>> a = 1:n; b = 1:2:10;
```

Matrix whose entries are all ones, or all zeroes:

```
>> A = ones(2,4); B = zeros(3,1);
```

The identity matrix:

```
>> E = eye(3);
```

Diagonal matrices:

```
>> D = diag(ones(3,1)); D1 = diag(ones(3,1),1);
```

Built-in functions

The built-in functions are all designed to work with matrices.

```
>> y = sin(0.5); z = sin([0:pi/6:pi]);
```

An example of plotting a graph:

```
>> x = [0:100]*pi/100; plot(x,sin(x));
```

Taking diagonal, size and length of a matrix:

```
>> d = diag(A); s = size(A); l = length(A);
```

Sum of all entries:

```
>> s = sum(A);
```

User-defined functions

Suppose that one stores the following in a file named `plotsin.m`.

```
x = [0:100]*pi/100;
plot(x,sin(x));
```

If one enters `plotsin` in the command line now, Matlab executes the sequence of commands in the file. We can also arrange so that the whole process depends on parameters that should be specified by the user:

```
function plotsin(x1,x2)
x = [0:100]*(x2-x1)/100+x1;
plot(x,sin(x));
```

This **function** *does* something depending on the parameter values. A function can **return a value**, like the ordinary functions `sin`, `cos`, etc.

```
function z = spc(x)
z = sin(x)+cos(x);
```

Conditional statements and loops

User-defined functions are good for keeping organized and saving lots of re-typing, but the real power of user-defined functions, and programming in general lies in the use of **conditional statements**:

```
function z = green(x,y)
if x<y
   z = x*(y-1);
else
   z = (x-1)*y;
end;
```

and in particular, **loops**:

```
function z = sumar(n1,n2)
z = 0;
for i = n1:n2
   z = z + i;
end;
```

Free alternatives to Matlab

Fairly close to being Matlab clones:

- Octave
- Scilab
- FreeMat
- There are many more choices, with varying degree of Matlab compatibility

Not exactly Matlab clones:

- SciPy - Collection of Python packages for scientific computing
- `C/C++`, `FORTRAN`,...