MATLAB Course
November-December 2006

Chapter 2: M-files

General information
Windows

MATLAB works with several windows. In the ***command window*** statements can be typed and executed by pressing the 'Enter' button. MATLAB makes a history of all statements used in the ***history window***. Previous statements can be toggled through with the '↑' button or can be selected from the ***history window***.

Functions can be edited in the ***m-file editor*** (If not open: File > New > M-file). MATLAB saves a function under the name (*.m) corresponding to the name used within the function. Functions can be executed by running the function name in the ***command window***. Note, that the ***current directory*** must indicate the location of the function. This can be changed by selecting the location with the scroll down button or using the command:

cd '*location of function*' (for example: cd 'c:\windows\desktop\' or cd 'a:\matlab')

When a statement ends with ';' the result is not shown in the ***command window***. The '%' sign is used to indicate comment. Both functions and loops are indicated with blue.

Load SPSS data file into MATLAB

- SPSS: save data file as fixed ASCII file: NAME.dat (use points, no comma's)
- MATLAB: see that NAME.dat is in the current directory
- Top bar: file, import data, NAME.dat, next, finish
- result: NAME contains the data (ask size(NAME) for checking the data matrix)
- *remark*: the description above is for the English versions

Adding a new current directory

- command window: line above shows the current directory
- ▼ (at the right above) shows the list of possible current directories (you can choose)
- **...** (at the right above) makes it possible to add a file or folder to the list

Assignment

To define a matrix in MATLAB one must specify all the elements per row between square brackets. The different rows need to be separated by a ';'. Of course, all rows must be of equal length. The MATLAB code to define a matrix **X** and a vector **y**, is:

```
>> % Define matrix X
>>  X = [1 2 3;5 4 1;3 6 2]

X =
   1   2   3
   5   4   1
   3   6   2

>>  % Define matrix y
>>  y = [1; 3; 2]

y =
   1
   3
   2
```

Using functions

The above statements can also be put into a function:

```
function define
X = [1 2 3;5 4 1;3 6 2]
y = [1; 3; 2]
```

The function can be executed, by typing the name of the function in the ***command window***:

```
>>  define

X =
   1   2   3
   5   4   1
   3   6   2

y =
   1
   3
   2
```

X, y are not defined outside the function. If you want to keep the values of **X** and **y**:

```
function [X,y]=define
X=[1 2 3;5 4 1;3 6 2]
y=[1;2;3]
```

```
>>  [X,y]=define
 X =
     1     2     3
     5     4     1
     3     6     2

 y =
     1
     3
     2
```

Now **X** and **y** are known in the MATLB command window. Other names can also be used e.g.

```
 >>  [Z,p]=define

 Z =
     1     2     3
     5     4     1
     3     6     2
 p =
     1
     3
     2
```

If the data is already defined, it can be used in the function by using the expression

```
function multiply (X,y)
A = X * y
```

The function above will use both matrix **X** and vector **y** and calculate the product **A**. Note: the function's name for the example above is '*multiply.m*'.

If we have the above function, then we may type:

```
>>  X = [1 2 3;5 4 1;3 6 2];
>>  y = [1; 3; 2];
```

The ';'' at the end is used to avoid the printing of the new value.

```
>>  multiply

A =
   13
   19
   25
```

If we want the function to produce data so that it may be used outside the function, we use the expression:

```
function [A]=multiply(X,y)
A = X * y
```

The result **A** obtained from the above function can be used for further purposes:

```
>>  X = [1 2 3;5 4 1;3 6 2];
>>  y = [1; 3; 2];

>> [A]=multiply(X,y)

>>  A
        13
        19
        25
```

Remark: the brackets [A] are not necessary, if only one matrix is between the brackets.

For more than one matrix, like the function [X,y]=define, they are necessary.

Using loops

We will consider in short two kinds of loops. The first one is the ***for*** – loop, which can be used when the number of iterations is known. The second is the ***while*** **–** loop, which can used when the number of iterations is not known and some criterion needs to be satisfied.

To demonstrate the ***for*** – loop we may define a column vector and add all element:

```
function loop(y)
%determine order of y
[n,p] = size(y);
%run loop
sum = 0;
for i = 1 : n
   sum = sum + y(i);
end
sum
```

```
>>  y=[1;3;2]

y =
    1
    3
    2

>>  loop(y)

sum =
       6
```

Note the ‘end’ expression, which marks the end of the loop. Within ‘for’ and ‘end’ all statements are repeated *n* times.

To demonstrate the ***while*** – loop, we will add again the elements of a vector. But, this time, the loop ends when the criterion ‘sum’ is satisfied:

```
function loop1(y)
sum = 0;
i = 1;
while sum < 3
    sum = sum + y(i);
    i = i + 1;
end
sum
```

```
>> y

y =

     1
     3
     2

>> loop1(y)

sum =

     4
```

Note, that the loop ends after two iterations, not starting a third, because the criterion ‘sum < 3’ is satisfied, as ‘sum = 4’.

Example 1

It holds:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Make an m-file for finding an approximation of e^x. Check this result for several values of x.

```
function f=example1(x)
f=0;
crit=1e-10
dif=1;
i=-1;
while abs(dif) > crit
   i=i+1;
   dif=(x^i)/factorial(i);
   f=f+dif;
end
```

Another way to jump out off the loop in case of a very small difference:

```
function f=example1a(x)
f=0;
for i=0:200
   dif=(x^i)/(factorial(i));
   f=f+dif;
   if dif<.0000000001
   break
   end
end
i
```

```
>> f=example1a(1)

i =
   14

f =
   2.7183
```

logicals

symbols:

>	larger
> =	larger or equal
<	smaller
< =	smaller or equal
= =	equal
~ =	not equal

Result is 0 (false) or 1 (true)

Example 2

```
>>   a = 1;
>>   b = 2;
>>   a > b

ans =
     0

>>   a < b

ans =
     1

>>   a == b

ans =
     0

>>   a ~= b

ans =
     1
```

The “if” statement

The structure is:

```
if “condition is true”
   execute statement(s)
else
   execute other statement(s)
end
```

Example 3

```
function example3
a=[1 -2 3 -4 -3 2]
sum1=0;
sum2=0;
for i=1:6
   if a(i) > 0
      sum1=sum1 + a(i);
   else
      sum2=sum2 + a(i);
   end
end
[sum1 sum2]
```

```
>>  example3

a =
     1    -2     3    -4    -3     2


ans =
     6    -9
```

Exercises chapter 2

1: Generate matrix **A** as a (5 x 3) random matrix and matrix **B** as a (3 x 4) matrix. Compute the product of theses matrices by using several loops. Make an m-file of this function and check the result with the standard operator of MATLAB.

2: It holds:

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \text{ for } -1 < x \leq 1.$$

Make an m-file for finding an approximation of $\log(1+x)$. Check this result for several values of x.

3: Generate n standard normally distributed scores and compute the mean and the standard deviations of these scores conditionally that they are larger than 0. Repeat this for different values of n. For instance, n = 100, 1000, 10000,….

4: A polynomial is given as $-x^2 + 2x + 1$. Equate this polynomial equal to 0 and find a solution for x. Hint: start with the interval [1,4] and find the function values of the endpoints of this interval. Compute the midpoint of this interval and compute the corresponding function value. Find a way for finding a new smaller interval in which the solution is situated and start the whole procedure all over. Repeat this till the end points are “close” to each other.

5. Load the Sesamstreet data and compute the minimum, the maximum and the mean of the ages. Compare the results with the SPSS output.