Programming with MATLAB

Edward Neuman

Department of Mathematics

Southern Illinois University at Carbondale

This tutorial is intended for those who want to learn basics of MATLAB programming language. Even with a limited knowledge of this language a beginning programmer can write his/her own computer code for solving problems that are complex enough to be solved by other means. Numerous examples included in this text should help a reader to learn quickly basic programming tools of this language. Topics discussed include the m-files, inline functions, control flow, relational and logical operators, strings, cell arrays, rounding numbers to integers and MATLAB graphics.

2.1 The m-files

Files that contain a computer code are called the *m-files*. There are two kinds of m-files: the *script files* and the *function files*. Script files do not take the input arguments or return the output arguments. The function files may take input arguments or return output arguments.

To make the m-file click on **File** next select **New** and click on **M-File** from the pull-down menu. You will be presented with the **MATLAB Editor/Debugger** screen. Here you will type your code, can make changes, etc. Once you are done with typing, click on **File,** in the **MATLAB Editor/Debugger** screen and select **Save As...** . Chose a name for your file, e.g., **firstgraph.m** and click on **Save.** Make sure that your file is saved in the directory that is in MATLAB's search path. If you have at least two files with duplicated names, then the one that occurs first in MATLAB's search path will be executed. To open the m-file from within the **Command Window** type **edit firstgraph** and then press **Enter** or **Return** key.

2.2 Control flow

To control the flow of commands, the makers of MATLAB supplied four devices a programmer can use while writing his/her computer code

the **for** loops
the **while** loops
the **if-else-end** constructions
the **switch-case** constructions

2.3.1 Repeating with for loops

Syntax of the **for** loop is shown below

```
for k = array
    commands
end
```

The commands between the **for** and **end** statements are executed for all values stored in the **array.**

Suppose that one-need values of the sine function at eleven evenly spaced points **πn/10**, for **n = 0, 1, … 10**. To generate the numbers in question one can use the **for** loop

```
for n=0:10
      x(n+1) = sin(pi*n/10);
end
x
```

```
x =

  Columns 1 through 6

         0    0.3090    0.5878    0.8090    0.9511    1.0000

  Columns 7 through 11

    0.9511    0.8090    0.5878    0.3090    0.0000
```

The **for** loops can be nested

```
H = zeros(5);
for k=1:5
    for l=1:5
        H(k,l) = 1/(k+l-1);
    end
end
H
```

```
H =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

Matrix **H** created here is called the *Hilbert matrix*. First command assigns a space in computer's memory for the matrix to be generated. This is added here to reduce the overhead that is required by loops in MATLAB.

The **for** loop should be used only when other methods cannot be applied. Consider the following problem. Generate a 10-by-10 matrix **A = [a_{kl}]**, where **a_k = sin(k)cos(l)**. Using nested loops one can compute entries of the matrix **A** using the following code

```
A = zeros(10);
for k=1:10
    for l=1:10
        A(k,l) = sin(k)*cos(l);
    end
end
```

A loop free version might look like this

```
k = 1:10;
```

```
A = sin(k)'*cos(k);
```

First command generates a row array **k** consisting of integers 1, 2, ... , 10. The command **sin(k)'** creates a column vector while **cos(k)** is the row vector. Components of both vectors are the values of the two trig functions evaluated at **k.** Code presented above illustrates a powerful feature of MATLAB called *vectorization*. This technique should be used whenever it is possible.

2.3.2 Repeating with while loops

Syntax of the **while** loop is

```
while expression
      statements
end
```

This loop is used when the programmer does not know the number of repetitions a priori.

Here is an almost trivial problem that requires a use of this loop. Suppose that the number **71** is divided by 2. The resulting quotient is divided by 2 again. This process is continued till the current quotient is less than or equal to 0.01. What is the largest quotient that is greater than 0.01? To answer this question we write a few lines of code

```
q = pi;
while q > 0.01
      q = q/2;
end
q
```

```
q =

0.0061
```

2.3.3 The if-else-end constructions

Syntax of the simplest form of the construction under discussion is

```
if expression commands end
```

This construction is used if there is one alternative only. Two alternatives require the construction

```
if expression
    commands (evaluated if expression is true)
else
    commands (evaluated if expression is false)
end
```

If there are several alternatives one should use the following construction

```
if expression1
     commands (evaluated if expression 1 is true)
elseif expression 2
      commands (evaluated if expression 2 is true)
elseif …
```

...

...
else
commands (executed if all previous expressions evaluate to false)
end

Chebyshev polynomials $T_n(x)$, **n = 0, 1, ...** of the first kind are of great importance in numerical analysis. They are defined recursively as follows

$$Tn(x) = 2xTn - i(x) - Tn - z(x),\ n = 2, 3, \ldots ,\ To(x) = 1,\ Ti(x) = x.$$

Implementation of this definition is easy

```
function T = ChebT(n)
% Coefficients T of the nth Chebyshev polynomial of the first kind.
% They are stored in the descending order of powers.
t0 = 1;
t1 =   [1 0];
if n == 0
   T = t0;
elseif n == 1;
   T = t1;
else
   for k=2:n
      T =   [2*t1 0]   -   [0 0 t0];
      t0 = t1; t1 = T;
   end
end
```

Coefficients of the cubic Chebyshev polynomial of the first kind are

```
coeff = ChebT(3)

coeff =
      4     0    -3     0
```

Thus $T_3(x) = 4x^3 - 3x$.

2.3.4 The switch-case construction

Syntax of the **switch-case** construction is

switch expression (scalar or string)
case valuel (executes if expression evaluates to valuel)
commands
case value2 (executes if expression evaluates to value2)
commands
otherwise
statements
end

Switch compares the input expression to each case value. Once the match is found it executes the associated commands.

In the following example a random integer number x from the set {1, 2, ... , 10} is generated. If x = 1 or x = 2, then the message Probability = 20% is displayed to the screen. If x = 3 or 4 or 5, then the message Probability = 30% is displayed, otherwise the message Probability = 50% is generated. The script file **fswitch** utilizes a switch as a tool for handling all cases mentioned above

```
% Script file fswitch.
x = ceil(10*rand);
% Generate a random integer in {1,  2,   ...   , 10}
switch x
case {1,2}
      disp('Probability = 20%');
case {3,4,5}
      disp('Probability = 30%');
otherwise
      disp('Probability = 50%');
end
```

Note use of the curly braces after the word **case.** This creates the so-called *cell array* rather than the one-dimensional array, which requires use of the square brackets. Here are new MATLAB functions that are used in file **fswitch**

rand - uniformly distributed random numbers in the interval (0, 1)
ceil - round towards plus infinity infinity (see Section 2.5 for more details)
disp - display string/array to the screen

```
for k = 1:10
      fswitch
end
```

```
Probability =50%
Probability =30%
Probability =50%
Probability =50%
Probability =50%
Probability =30%
Probability =20%
Probability =50%
Probability =30%
Probability =50%
```

References

[1] D. Hanselman and B. Littlefield, Mastering MATLAB 5. A Comprehensive Tutorial and Reference, Prentice Hall, Upper Saddle River, NJ, 1998.

[2] P. Marchand, Graphics and GUIs with MATLAB, Second edition, CRC Press, Boca Raton, 1999.

[3] K. Sigmon, MATLAB Primer, Fifth edition, CRC Press, Boca Raton, 1998.

[4] Using MATLAB, Version 5, The MathWorks, Inc., 1996.

[5] Using MATLAB Graphics, Version 5, The MathWorks, Inc., 1996.