**Introduction to MATLAB Programming**

Originally created by  
*Kristian Sandberg  
Department of Applied Mathematics  
University of Colorado*

Updated for compatibility with Release 13 by  
*Grady Wright  
Department of Mathematics  
University of Utah*

The goal of this tutorial is to introduce the fundamental ideas of programming in MATLAB. It requires no programming experience, but some familiarity with MATLAB is recomended. All background needed can be found on the tutorial [Introduction to MATLAB](http://www.math.utah.edu/~wright/courses/5610/matlab/matlabintro.html).

The programming structures presented below apply to MATLAB. However, these structures look very similar in other computer languages, such as C, Java, Pascal, etc., so by understanding how loops, logical operations, etc., work in MATLAB, you will be well-prepared for beginning programming in other languages as well.

Many of the examples given may seem to be irrelevant to this course. However, by understanding the seemingly stupid and sometimes mathematically irrelevant examples and exercises in this tutorial, you will have all the background to write programs to solve all the homework problems.

A number of examples will be given below. They will be given as MATLAB code but the output which you will get when you run these programs will not be given. When going over this tutorial, you are recommended to implement the examples yourself and then run them in the MATLAB command window and carefully study the outcome and compare it to the code.

**How to edit and run a program in MATLAB**

When writing programs, you need to do this in a separate window, called the *editor*. To open the editor, go to the "File" menu and choose either the "New...M-file" (if you want to create a new program) or "Open" (to open an old document) option. In the editor you can now type in your code, similarily to how you would type using a word processor. There are menus for editing the text, just like your favorite word processor. While typing your code in the editor, **no commands will be performed**! In order to run a program do the following:

1. Save your code as *<filename>.m*, where *<filename>* is anything you wish to name the file. **It is important to add ".m" at the end of your filename.** Otherwise MATLAB may not understand your program.
2. Go to the command window. If necessary, change directories to the directory containing your file. For example, if you saved your file in the directory "C:\mymatlab", you need to change the current directory of MATLAB to this directory. This can be accomplished using the cd command common to UNIX and DOS. Or, alternatively, you can select the button labeled "..." on the "Current Directory" window, and change directories graphically. You can view your current directory by typing the pwd command.
3. In order to run the program, type the name of the file containing your program at the prompt. **When typing the filename in the command window do not include ".m"**. By pressing *enter*, MATLAB will run your program and perform all the commands given in your file.

In case your code has errors, MATLAB will complain when you try to run the program in the command window. When this happens, try to interpret the error message and make necessary changes to you code in the editor. The error that is reported by MATLAB is hyperlinked to the line in the file that caused the problem. Using the mouse you can thus jump right to the line in your program that has caused the error. **After you have made the changes, make sure you save your file before trying to run the program again in the command window.**

**General programming advice**

When programming, it is extremely important, even for the most experienced programmer, to never try to write all the code at once. You have to program in small steps and make sure each of these small steps work as expected before proceeding to program the next step. When writing longer code, always use pen and paper to outline the code before you actually type it into the editor.

**Basic programming structures**

**Input/Output**

To make MATLAB give an output, such as return the value of a variable, you type the name of the variable without any semi-colon (;) following the variable. In many cases, there is no need to see the value of a variable every single time MATLAB uses it. If MATLAB re-computes the value of a variable 1000 times, we probably don't want to see the result every single time. To surpress the output of a value, just add a semi-colon after the variable. Then MATLAB will perform the command, but will not show it on the screen.

Let's say that we want the user to enter some value that we want the program to work with. This can be done using the *input* command with the syntax

*variable*=input('*text*');

This command will print out *text* on the screen and then wait for the user to enter a number. The variable will now be assigned the number that the user entered. (Using this command for reading letters instead of numbers is slightly more complicated and will not be covered in this tutorial.)

Now it is time for our first example. The following program asks the user for an amount in dollars, and returns the value of this amount in a foreign currency.

*Example 1.*

| clear  exchange\_rate = 0.5;  amount = input('Give amount in dollars: ');  amount\_in\_foreign\_currency = exchange\_rate\*amount |
| --- |

A good practice is to begin your code with clear, which erases all variables. If you do not do this, you can get errors when you run your program that are very hard to discover.

**Relational and logical operators**

This heading may sound scary but it is really just a fancy name for some of the fundamental operators used for programming. Below follows a list with some useful commands.

| *Logical operators* |
| --- |
| **Operation:** | **MATLAB command:** |
| Logical and | & |
| Logical or | | |
| Negate | ~ |

| *Relational operators* |
| --- |
| **Operation:** | **MATLAB command:** |
| Strictly less than | < |
| Less than or equal to | <= |
| Strictly greater than | > |
| Greater than or equal to | >= |
| Equal to | == |
| Not equal to | ~= |

It is important to know the difference between = and ==. The former, =, is used when assigning a number to a variable, e.g., x=3;. The latter, ==, is used to check if two expressions are equal. This is illustrated in the examples below, but first we need to know what an if-statement is. When programming we often want the computer to check whether a statement is true or false and perform different operations depending on the result of this test. This can be done using a so-called if-statement. The syntax is given below.

| if *logical expression*  *commands*  else  *commands*  end |
| --- |

Note that for each if, you need to "close" the if-statement with an end. Make sure the if:s and end:s always match! (This is a common source for programming errors.)

The content of this paragraph may have seemed abstract but by carefully studying the following three examples and doing Exercise 1 it will hopefully become clearer.

*Example 2.*

| clear  N = input('Give numerator: ');  D = input('Give denominator: ');   if D==0  'Sorry, cannot divide by zero'  else  ratio = N/D  end |
| --- |

In the next example, we make MATLAB write something depending on which test our month "passes". To make MATLAB write text as output, use single quote ' around the text.

*Example 3.*

| clear  month = input('Give month number (1-12): ' );   if month==1 | month==3 | month ==5 | month==7 | month==10 | month==12  'Your month has 31 days'  else  if month==2  'Your month has 28 days'  else  'Your month has 30 days'  end  end |
| --- |

In the next example we use the command rem (remainder). The syntax is

rem(x,y)

and returns the remainder after the division of the two integers x and y.

*Example 4.*

| clear  number = input('Give an integer: ' ); remainder2 = rem(number,2);  remainder3 = rem(number,3);   if remainder2==0 & remainder3==0  'Your number is divisible by both 2 and 3'  else  if remainder2==0  'Your number is divisble by 2 but not by 3'  else  if remainder3==0  'Your number is divisible by 3 but not by 2'  else  'Your number is not divisible by either 2 or 3'  end  end  end |
| --- |

| **Exercise 1.** Write a "currency exchange program" similar to the one in Example 1 which can handle two different exchange rates, exchange\_rate1 = 0.5 and exchange\_rate2 = 0.25. Design the program to first ask for the amount in dollars and then ask the user which rate (represented by the numbers 1 and 2 respectively) he/she wants. Let the program return the amount in the requested foreign currency. |
| --- |

**Repetitive operations (loops)**

The power of computers is that they can do operations repeatedly (without getting bored!). For example, as we will see next semester, an ODE-solver may compute the value of a function several thousands of times when computing the solution with a small time step. An operation that is performed repeatedly is called a repetitive operation or, more common, a loop. There are different kinds of loops but the most common one is the for-loop. The syntax for a for-loop is:

| for *loop variable* = *startvalue* : *endvalue*  *commands*  end |
| --- |

This loop will initate *loop variable* as *start value*, increment *loop variable* by 1 each step until *end value* is reached. Below follows a few examples of how to use for-loops.

*Example 5.*

| clear  for i=1:20  x(i)=i/7;  end   x |
| --- |

In the following example we see a so-called nested for-loop. This is nothing else then a "loop within a loop". Note how we must "close" each for with an end. Make sure you understand how this example works!

*Example 6.*

| clear  for i=1:5  for j=1:5  A(i,j)=10\*i+j;  end  end   A |
| --- |

In the following example, make sure you understand the purpose of the variable mysum. This is a common way of performing summations when programming.

*Example 7.*

| clear  mysum = 0;  for k=0:10  mysum = mysum+1/gamma(k+1);  end   e\_approximation = mysum  e\_exact = exp(1) |
| --- |

In the next example, notice how we have to "shift" the indexing of the vector. **MATLAB must have non-zero, positive integers as vector- or matrix-indices!** One of the most common mistakes when programming in MATLAB is that your program begins indexing at zero instead of one. Also note how by typing a percent sign (%) before text in the code, MATLAB does not interpret this text as code. It just serves as a comment for any person using the code. **Commenting your code is essential when writing programs**.

*Example 8.*

| clear   for k=0:70  x(k+1)=0.1\*k; % Indices of vectors must be NON-ZERO!  sum = 0;  for m=0:10  sum = sum+(x(k+1)^m)/gamma(m+1); % Approx exp(x) using its Taylor series  end   e(k+1) = sum; % e(k+1) contains the Taylor series approx. for x=x(k+1);  end   semilogy(x,e)  title('Approximation of e^x for x between 0 and 7')  xlabel('x')  ylabel('e^x') |
| --- |

| **Exercise 2.** Write a program that approximates PI by computing the sum  http://www.math.utah.edu/~wright/misc/matlab/images/ws5formula.txtgr1.gif.  The more terms you keep in the summation, the more accurate your answer will be. (In fact, the series converges to PI as m goes to infinity.) See how many terms you need to approximate PI with 5 decimals. (Note: This is by no means the most efficient way to approximate PI, but the formula is quite beautiful...) |
| --- |

| **Exercise 3.** Use the sum given in Exercise 2 to approximate PI using 10, 100, 1000, 10000 and 100000 terms. For each of these number, compute the error of the approximation. Plot the error as a function of the number of terms used in the sum. |
| --- |



