1. Introduction

This is not really an introduction to programming, this is merely a guide to getting started with a mathematical programming tool like Scilab and or Matlab. To the
uninitialed, this can look a lot like programming in a computer language; but actually we stop far short of teaching programming.

But some programming terms and techniques will deepen your understanding, ease the your use and increase your enjoyment of the programs. In particular we show many of the common beginner errors and how to fix them.

Often the most common mistake is forgetting how to use a command. The `help` command is your friend. Typing

```
help plot
```

will show some documentation on the plot command works. Scilab will put you the help browser (which is hyperlinked) in a separate window. Matlab will put the information in the current window but also has also has a help browser one can invoke with the command

```
doc plot
```

### 2. The Assignment Statement

We will use the abbreviations LHS and RHS for *Left Hand Side* and *Right Hand Side* The assignment statement is the basis of most of the commands that you will use and it has the form

\[ \text{LHS} = \text{RHS} \]

This is not an equation, it is not a definition, it is an *action*. It says compute the RHS and place the value in the spot given by the LHS. The equals sign = in the command is almost a sure sign that the command is an assignment statement. (For loops (Section 7) and function definitions (Section 8 also have an = sign.) The assignment statement is the most common programming structure.

The left hand side (LHS) must be a *variable*, which is the name for the location where its value is stored. This can be the first use of the variable, but more often it is a re-using of an old location. (The actual location in computer memory doesn’t matter, only the program needs to know where.) What ever was in this location before the assignment statement is now gone; it cannot be recovered.

Variables must start with a letter, but after that can have digits or letters or even underscores. Case matters, the variable `a` is different from the variable `A`. Can you see the difference between `a1` and `al`, `b0` and `bO`, `cS` and `cS`, `xo` and `XO`?

The right hand side (RHS) must be computable to a value. The RHS can be a variable, but it must be one that is already defined, it can’t be the first use of the variable. More often, the RHS is an expression of numbers, variables and operations. In addition, many mathematical functions are available to use in RHS. Examples include `2 + 2`, `5 * x`, `x + y + z`, `sin(x)` and many more.

The role of a variable that appears on both sides of an assignment statement is very different. Consider the statement:

\[ a = a + 2 \]

This says compute the RHS: get the old value of `a`, add two; then put in in the location of `a`, which becomes the new value of `a`. (Note that `a = a + 2` does not make
sense as an equation. If it made sense, you could substract a from both sides and obtain the extremely false equation 0 = 2.)

3. LHS Errors

While the only requirement on the LHS is that it is a variable there are several ways to arrive at a LHS error.

(1) The typo:
   (a) one is typing VAR5X, but leaves cap key on and gets VAR%X which is not a legal variable name.
   (b) one is typing TheVariable, but puts a space and types The Variable instead. You can’t have a space in a variable name. Nor can you have two variable names on the LHS.
   (c) one is typing variable x but you really want to type y. The program always believes you know what you are doing. So no error is generated, but likely you will get the wrong answer or another error later on.

(2) The definition:
   (a) one types \( f(x) = 2 \times x + 2 \) attempting to define a function \( f(x) \). The program thinks you are looking at a matrix named \( f \) and are storing into the \( x \)-th entry of \( f \). If \( f \) is defined and \( x \) is a small integer, this might generate no error message. But it is not what you want. Functions can be defined but they use a different syntax (section 8).
   The matrix use is so you can have
   \[
   A = \begin{bmatrix}
   1 & 5 \\
   2 & 5 \\
   3 & 6 
   \end{bmatrix}
   \]
   and correct the entry in row 1 column 2 by \( A(1,2) = 4 \) and then
   \[
   A = \begin{bmatrix}
   1 & 4 \\
   2 & 5 \\
   3 & 6 
   \end{bmatrix}
   \]
   (b) one types ‘let x = 4’ attempting to mimic the statement in a math book. The correct way to type this is just ‘x = 4’, (see also the equation item below.)

(3) The equation:
   (a) one types \( x^2 + x - 2 = 0 \), a quadratic equation. Here the LHS is an expression and not a variable.
   (b) one types \( 0 = x^2 + x - 2 \), the same quadratic equation reverse. Here the LHS is the number 0 and not a variable.

(4) A serious design flaw in Matlab:
   (a) matlab uses pi for \( \pi \) but doesn’t protect it. You can type ‘pi = 3’ and change the value of \( \pi \) to 3. The value of \( \pi \) will be wrong until you fix it; ‘clear pi’ will return the correct value.
   (b) scilab uses \( \%pi \) for \( \pi \) and protects it. You can’t change the value of \( \%pi \).
4. RHS Errors

(1) Undefined variable
(a) If you type $x = y$ and the variable $y$ is not defined yet. This generates an easy to understand error message.
(b) If you type $x = y$ and you have forgotten to initialize the variable $y$ you might not get a error message if the variable $y$ was used earlier in your session. The ‘clear’ command removes all variable and will prevent using stale data like this.
(c) If $z$ is undefined, then $z = 7$ would define $z$, but $z = z$ does not. The undefined $z$ error stops the assignment before $z$ would be defined.

(2) RHS expression that makes no sense:
(a) A typo like $w = x + y+$ where the last $z$ got omitted will generate an error message.
(b) There is no implied multiplication, $w = x y$ will produce an error message.
(c) Confusion between matrix multiplication and elementwise multiplication:
(i) If $x = 3$ then matrix multiplication $x \times x$ and elementwise multiplication $x \cdot x$ both are defined and yield 9.
(ii) If $x = [1, 2, 3]$ then matrix multiplication $x \times x$ is undefined, but elementwise multiplication $x \cdot x$ is defined and yields $[1, 4, 9]$.
(iii) If $x = 3$ and $y = [1, 2, 3]$ then matrix multiplication $x \times y$ and elementwise multiplication $x \cdot y$ both are defined and yield $[3, 6, 9]$.
(iv) If $x = [1, 2, 3]$ and $y = [4; 5; 6]$ then matrix multiplication $x \cdot y$ is defined and equal to $(1)(4)+(2)(5)+(3)(6) = 32$, but and elementwise multiplication $x \cdot y$ is not defined.
(d) Confusion between matrix exponentiation and elementwise exponentiation
(i) If $x = 3$ then matrix exponentiation $x^x$ and elementwise exponentiation $x \cdot x$ both are defined and yield 27.
(ii) If $x = [1, 2, 3]$ then matrix exponentiation $x^x$ is undefined, but elementwise exponentiation $x \cdot x$ is defined and yields $[1, 4, 27]$.
(iii) If $x = 3$ and $y = [1, 2, 3]$ then both $x^y$ are undefined $x^y$ while both elementwise exponentiation $x \cdot y$ is defined.

5. Variables and Function Names

Variables and function names must start with a letter, but after that can have digits or letters or even underscores. Case matters, the variable $a$ is different from the variable $A$. Variables and function names live in the same namespace, which is too say you can’t have a variable and a function with the same name at the same time. Consider:

```
sqrt(16)
sqrt=3
sqrt(16)
```
\texttt{sqrt(1)}

The first line calls the square root function, which will return 4, the square root of 16. The second line creates a variable names \texttt{sqrt} and assigns it the value of 3. The function \texttt{sqrt} is no longer available, the third line will produce an error. The error message reflects the fact that the variable \texttt{sqrt} does not have a 16\textsuperscript{th} entry. The last line has a more serious error. It returns 3, because the 1\textsuperscript{st} entry is 3, but it looks like the program has just told you that the square root of 1 is 3. To get the old behavior of \texttt{sqrt} back one can clear the new name by typing: \texttt{clear sqrt}.

You might think that the program should protect the names of important operations like \texttt{sqrt}. But there are way too many functions and predefined variables to protect them all. Scilab protects some variables and function names like \texttt{\%pi}. Matlab has reserved keywords which are protected that can seen by typing: \texttt{iskeyword}.

(Turns out scilab allows the characters \#, !, $ and ? as characters in an variable name and in addition allows the variable to start with \%. How awful.)

\section*{6. The comment character}

It is extremely helpful to add notes to your lists of commands and functions. The comment character is a way of telling the program that what followings is not for the program but notes. The comment character for matlab is the percent sign (\%), while the comment characters for scilab is a pair of slashes (/\!). It takes some thought to write a line that both matlab and scilab will do nothing on, here is one:

\texttt{\%pi; // Comment here}

Since it starts with a \% it is a matlab comment. Scilab does do the \%pi;,, but the semi-colon makes it silent, and the command has no side effects.

\section*{7. The For Loop}

The for loop is what is called a control structure. (There are other control structures like the the while loop and the if then else statement which are not covered in this notes.) The for loop is basically a way to get the computer do very similar statements many times. The syntax is

\texttt{for variable=expression, instruction, instruction, ... instruction, end}

The variable is call the loop variable and often \texttt{i, j} or \texttt{k} fill this role. The for loop can be spread over many lines, it keeps going until the word end appears. Here are some examples that are all equivalent and end up with \texttt{a = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}}

\texttt{clear a;}
\texttt{for i = 1:3, a(1,i) = i; end}

The indentation of the middle command is a common way of emphasising the instructions inside the body of the for loop

\texttt{clear a;}
\texttt{for i = 1:3}
\hspace{1cm} \texttt{a(1,i) = i;}
\texttt{end}
This command repeats 3 times the first with \( i = 1 \) which makes \( a(1,1) = 1 \). The second with \( i = 2 \) and makes \( a(1,2) = 2 \). The third has \( i = 3 \) and makes \( a(1,3) = 3 \). The clear statement undefines the variable \( a \). If \( a = \begin{bmatrix} 5 & 5 & 5 & 5 \end{bmatrix} \) and we didn’t clear \( a \) then the result would be \( a = \begin{bmatrix} 1 & 2 & 3 & 5 \end{bmatrix} \).

7.1. Some common for loop errors. Some common for loop errors:

1. Not including the word \textit{for}. The for loop needs the key word \textit{for} to be recognized.

2. Not including the word \textit{end}. The for loop doesn’t stop until the key word \textit{end} appears. Beginners often compound this error by starting another for loop before ending this one. The next end matches the inner \textit{for} and not the outer \textit{for}. Finally when the first \textit{for} gets matched with an \textit{end} the loop has inner loops. The calculation takes longer and does the wrong commands.

3. The typo, the command
   \[
   \text{clear } a; \\
   \text{for } i = 1;3, \ a(1,i) = i; \ \text{end}
   \]
   only does the loop once for \( i = 1 \) because a semi-colon typo for a colon.

4. Too much output. Most times you want the instructions inside the loop to be silent, so they should end with a semi-colon.

7.2. More examples. The \textit{for} loop allows us to repeat certain commands. If you want to repeat some action in a predetermined way, you can use the \textit{for} loop. All of the loop structures in Matlab are started with a keyword such as \textit{for}, or \textit{while} and they all end with the word \textit{end}. The \textit{for} loop is written around some set of statements, and you must tell Matlab where to start and where to end. Basically, you give a vector in the \textit{for} statement, and Matlab will loop through for each value in the vector. For example, a simple loop will go around four times each time changing a loop variable, \( j \):

   \begin{verbatim}
   INPUT:
   for j=1:4,
     j
   end
   \end{verbatim}

   \begin{verbatim}
   OUTPUT:
   j = 1
   j = 2
   j = 3
   j = 4
   \end{verbatim}

   In the previous example, the loop starts with \( j = 1 \), increments \( j \) by 1 each time, and goes up to \( j = 4 \). We could instruct Matlab to increment the loop variable \( j \)
by a different value every time. For example, the following instruction will render a
different output:

INPUT:
for j=1:2:4,
j
end

What is the output in this case?

OUTPUT:
j =
   1
j =
   3

8. Function Definitions

The function definition is one of the differences between scilab and matlab. The
function definition is similar, but matlab puts additional requirements, namely that
the function must be located in a dot-m with the same name. Furthermore, this file
must be in the current matlab directory.

For example, to define a square function in matlab create a file square.m which
contains
function y = square(x)
y = x .* x;
and make sure it is located in the current directory. Then typing square([1 2 3]) will
produce [1 4 9]. (Functions often get vector input like [1 2 3], which is why the square
function is using elementwise multiplication .* and not matrix multiplication.

The first line of a function tells the inputs, just x in this case and the outputs, just
y in this case. Functions can have multiple inputs and outputs. Somewhere in the
function body, there must be an assignment to the output variable.

To make this function work in scilab one can use an inline version. Just type

function y = square(x)
y = x .* x;
endfunction

at the prompt. Since scilab doesn’t require the separate file, the endfunction (all one
word) is needed. Matlab allows for an optional word end to end a function. (Scilab
already has a function called square that does something else.)

Like matlab, you can put a scilab function (or many functions) into a separate file.
By default, these are put into a dot sci file. Suppose we put the function into a file
named square.sci. To load the function we would need to type getf(’square.sci’).

Script files (basically a collection of commands) for matlab also have the exten-
sion dot m like function files. Scilab executes script files with the exec(’file.name’)
command so any extension can be used but dot sce is suggested.

Common function definition errors include:
1. No endfunction or misspelling endfunction with a space (scilab). Like the missing end in a for loop, this can cause beginners much confusion.
2. File name and function name don’t match (matlab).
3. File is not located in correct directory.
4. Using matrix operations when elementwise operations are needed (and vice versa).

9. Command introduction

Expressions use familiar arithmetic operators and precedence rules. There are also functions, such as cosine or sine. You surround the input to the function by parenthesis.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>⋆</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>'</td>
<td>Power</td>
</tr>
<tr>
<td>(</td>
<td>Complex conjugate transpose</td>
</tr>
<tr>
<td>)</td>
<td>Specify evaluation order</td>
</tr>
<tr>
<td>abs(x)</td>
<td>Absolute value (</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Square root ( \sqrt{x} )</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Exponential function ( e^x )</td>
</tr>
<tr>
<td>sin(x)</td>
<td>( \sin(x) ) where ( x ) is assumed to be in radians</td>
</tr>
<tr>
<td>cos(x)</td>
<td>( \cos(x) ) where ( x ) is assumed to be in radians</td>
</tr>
</tbody>
</table>

Constants are slightly different in the two programs. You can make \( \pi \) different in matlab by assignment like \( \pi = 17 \). How awful.

<table>
<thead>
<tr>
<th>Constant(matlab)</th>
<th>Constant(scilab)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi )</td>
<td>( %pi )</td>
<td>( \pi = 3.14159265 )</td>
</tr>
<tr>
<td>( i )</td>
<td>( %i )</td>
<td>Imaginary unit, ( i = \sqrt{-1} )</td>
</tr>
<tr>
<td>( j )</td>
<td></td>
<td>Same as ( i ) in matlab</td>
</tr>
<tr>
<td>( \text{Inf} )</td>
<td>( %inf )</td>
<td>Infinity (( \infty ))</td>
</tr>
<tr>
<td>( \text{NaN} )</td>
<td>( %nan )</td>
<td>Not-a-number</td>
</tr>
<tr>
<td>\true</td>
<td>%t</td>
<td>true</td>
</tr>
<tr>
<td>\false</td>
<td>%f</td>
<td>false</td>
</tr>
<tr>
<td>( %e )</td>
<td></td>
<td>Euler’s constant ( e = 2.7182818 )</td>
</tr>
</tbody>
</table>

10. List of Commands

Here is the list of matlab commands from Fall 2008 lessons. There is a separate list for the symbolic toolbox commands.

10.1. Colon Operator. The colon, \( : \), is one of MATLAB’s most important operators. It occurs in several different forms. The expression

\[ \text{colonOutput} = 1:10 \]
is a row vector containing the integers from 1 to 10. So, when there are only 2
numbers with a colon, a list of numbers is generated that are incremented by one,
where the first number corresponds to the number to the left of the colon and the
last number corresponds to the number to the right of the colon.

Check to see if

```
colonOutput = 1 2 3 4 5 6 7 8 9 10
```

Now try `colonOutput1 = 1:2:10`

Check to see if

```
colonOutput1 = 1 3 5 7 9
```

Notice that when there are 3 numbers separated by a colon, the middle number
indicates what the increment value is.

So the colon operator is a shorthand notation for a list of numbers.

10.2. Plotting a matrix as an image.

```latex
Syntax: image(A)
Output: draws a figure of a rectangle divided into sub-rectangles. The
color of the rectangle at location \((i,j)\) is determined by the value of
\(A(i,j)\) and the current colormap.
```

10.3. Random number generation.

```latex
Syntax: x = rand()
Assigns \(x\) a random number strictly between 0.0 and 1.0, which is dif-
ferent for different calls. To get a random integer in the set \(\{1, 2, \ldots n\}\)
use
\[
i = \text{ceil}(n \times \text{rand}).
\]
To get a random integer in the set \(\{-1, 1\}\) use
```
HeadsTails = 2 * floor(2 * rand()) - 1
```
Syntax: `rand('seed',s)`
starts the random sequence in a set point given by the number \(s\).

10.4. The For loop.

```latex
Syntax for \(i = n:m\), \(<\text{commands}>\), end
Repeats the sequence of commands \(m - n + 1\) times, the value of \(i\) each
time. So for \(i = 1:3\), \(i\), end, will list the value of \(i\) three times.
Example: for \(i = 3:5\), \(x = i^2\), end
will output, \(x = 9, x = 16\) and \(x = 25\).
Example: for \(i = 3:5\), \(x = i^2;\) end
will output nothing because of the semi-colon, but at the end of the
loop, \(i\) will be 5 and \(x\) will be 25.
```

10.5. Best linear fit.
Syntax: \( c = \text{polyfit}(xdata, ydata, 1) \) matlab
\([m \ b] = \text{reglin}(xdata, ydata)\) scilab

Output: Fits the best linear fit for the data with \( x \)-values \( xdata \) and \( y \)-values \( ydata \). (The number 1 says to fit polynomials of degree 1, lines).
The slope of the line is given by \( m = c(1) \) and the \( y \)-intercept is given by \( b = c(2) \). The Linear fit is \( y = mx + b \)
Example (matlab): \( c = \text{polyfit}([1 \ 2 \ 3],[2 \ 4 \ 8],1) \) returns \( 3.0000 \ -1.3333 \)
Example (scilab): \([m \ b] = \text{reglin}([1 \ 2 \ 3],[2 \ 4 \ 8],1)\) returns \( b = -1.3333333 \ m = 3.0000 \)
(The same matlab polyfit function can fit polynomials of higher degree. Changing the 1 to 3 will fill the best cubic fit. Scilab has a general datafit command.)
Best means it has the minimum sum of the squares of the errors.

10.6. exp and log.

Syntax: \( \text{exp}(X) \)
Output: evalulates the exponential of the elements of \( X \).

Syntax: \( \text{log}(X) \)
Description: Calculates the natural logarithm, \( \ln \), of the elements \( X \).

Syntax:
\( \text{log10}(X) \)
Description: Calculates the base 10 logarithm of the elements \( X \).

10.7. Best Exponential fit.

Syntax: \( c = \text{polyfit}(xdata, \log(ydata), 1) \) matlab
\([k \ c2] = \text{reglin}(xdata, \log(ydata))\) scilab

Output: Fits the best exponential fit for the data with \( x \)-values \( xdata \) and \( y \)-values \( ydata \). (The number 1 says to fit polynomials of degree 1, lines).
The exponential rate constant is given by \( k = c(1) \) and the initial value is given by \( B = \exp(c(2)) \). The exponential fit is given by \( y = Be^{kt} \)
Example: \( c = \text{polyfit}([1 \ 2 \ 3],\log([2 \ 4 \ 8]),1) \) returns \( 0.6931 \ -0.0000 \) and \( \exp(-0.0000) \) is \( 1.0000 \)

10.8. Sorting.
SCILAB/MATLAB PROGRAMMING TIPS

Syntax:

```
sort(dataset)
```
Output: An array of the input data set, ordered within columns. Description: Orders the entries in an array column by column. It does not preserve rows.
The sort command has many optional actions which are different in matlab and scilab; but the default action is the same. Scilab has a `mtlb_sort` which has similar optional actions to the matlab version.

Syntax:

```
sortrows(dataset,colNum)
```
Output: An array of the input data set, ordered by a specified column. Description: Orders the entries in an array preserving rows. Entries ordered by the specified column. In scilab, similar behavior is obtained via

```
sort([dataset(:,colNum), dataset],'r')
```

10.9. **Min, Max, Mean, Median, Standard Deviation.**

Syntax:

```
min(dataset)
```
Output: The minimum value in the dataset. Description: Determines and displays the minimum value in a column of data.

```
max(dataset)
```
Output: The maximum value in the dataset. Description: Determines and displays the maximum value in a column of data.

```
mean(dataset)
```
Output: The mean (or average) value in the dataset. Description: Determines and displays the average value in a column of data.

Syntax: `median(dataset)`
Output: The middle value in the dataset. Description: Orders the dataset. If the size of the dataset is an odd number, it displays the middle number. If the size is an even number, it displays the average of the two middle-most numbers.

Syntax: `var(dataset)`

```
matlab
variance(dataset)
scilab
```
Output: The variance of the dataset. Description: A measure of statistical dispersion. A measure of the distance from the mean.

```
std(dataset)
```
Output: The standard deviation of the dataset. Description: The square root of the variance.
10.10. **Loading a text file.**

Syntax:
```
s = load('filename.txt')     % matlab
s = loadmatfile('filename.txt') % scilab
```

Description: Creates an array of the data in filename.txt and stores it in the variable `s`.

10.11. **Norm.**

Syntax:
```
norm(X)
```

Output: The standard norm of the given list.

Description: Calculates and displays the output from the standard norm of the list. This is given by the following equation.

10.12. **Matrix entry.**

Syntax:
```
A=[A11,A12,A13;A21,A22,A23;A31,A32,A33]
```

Output: 3x3 matrix. Changing the number of elements gives you a matrix of any size.

Description: Creates matrix $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$.

Syntax: `size(B)`

Output: two numbers (n,m) where n is the number of rows and m is the number of columns in matrix B.

Syntax: `sum(X)`

Output (matlab): If X is a vector, the output is the sum of the elements of vector X. If X is a matrix, then the output is a vector that has the same number of columns as matrix X where each element represents the sum of each column of matrix X. So `sum(sum(X))` will sum all the entries. The command to add columns of X is `sum(X,1)` and to add rows is `sum(X,2)`.

Output (scilab) It is the sum of all the entries of the matrix X. The command to add columns of X is `sum(X,1)` and to add rows is `sum(X,2)`.

10.13. **Clear.**

Syntax: `clear`

Description: Erase values of all previously declared variable.

Syntax: `who`

Lists the variables that are currently defined. The command `whos` also shows their type and size.

**Syntax:** `plot(X,Y)`  
**Description:** The `plot(X,Y)` command plots vector Y versus vector X. This produces a graph of Y versus X.  
Can be combined and used like `plot(x1,y1,'ColorMarkerStyle1',x2,y2,'ColorMarkerStyle2')` So you can specify the appearance of each curve within the command.  
While basic plotting is the same in MATLAB and Scilab the options can be different.

**Syntax:** `figure(n)` MATLAB  
`scf(n)` Scilab  
**Description:** makes n the current figure, forces it to become visible and raises it above all other figures on the screen. If figure n does not exist, and n is an integer, a new figure n is created.  
The command `clf` will clear the figure window.

**Graph Title Syntax:** `title('text')` MATLAB  
`xtitle('text')` Scilab  
When giving titles for graphs, the convention is to list the dependent variable (y) first. For example, if x = time and y = speed, a valid graph title would be ”Speed Versus Time”, or alternatively, ”Y versus X”  
Scilab’s `xtitle` can also include the xlabel and ylabel.  
`xtitle('title text','x label text','y label text').`

**Axis Label Syntax:** `xlabel('text')` and `ylabel('text')` MATLAB  
use `xtitle` in Scilab see above.

**Axis Scaling Syntax:** `axis([xmin xmax ymin ymax])`

**Text in Graph at location (x,y) Syntax:** `text(x,y,'text')`

**Legend Syntax:**  
`legend('text1','text2', ...)` or `legend('text1','text2', ..., 'location', LOC)`  
uses the specified ‘text’ as labels in the legend. If the word ‘location’ (in quotes) is included along with a value for LOC, the location of the legend is indicated by the value of LOC. Common values for LOC are ‘NorthEast’, ‘NorthWest’, ‘North’, etc, ‘Best’. These correspond to inside plot top right (default), inside plot top left, inside plot top, etc, best location inside plot to least conflict with data in plot. A full list of these options can be found at: help legend Scilab uses a different mode to direct placement of the legend, check help legend.
**Syntax:** print -dpnf 'figure.pdf' matlab  
**Description:** Stores a color pdf version of the current figure in the file figure.pdf.  
**Syntax:** print -depsc 'figure.eps' matlab xs2eps(n,'figure.eps') scilab  
**Description:** Stores a color postscript version of the current figure (scilab the graph window number is n) in the file figure.eps. The option -depsc can be changed to other formats like jpeg. See help print.

### 10.15. Vis Lab.

**Syntax:**
```
subplot(m,n,p)
```
**Description:** Breaks the figure window into an $m$ by $n$ matrix of small subplots and selects the $p$th subplot for the current plot. The subplot command is followed by a graphics command (such as plot or image) that is the plot or image that is to be displayed. The plots are numbered along the top row of the figure window, then the second row, and so on.

**Syntax:**
```
for k = 1:n,
    image(D(:,:,k)); colormap(map);
    M(k) = getframe;
end
```
**Description:** Repeat statements a specific number of times, where $n$ is an integer representing the total number of times. The loop is begins with the for part of the command and must terminate with end. This is a matlab example.

**Syntax:**
```
movie(M,N,FPS) matlab
```
**Description:** Plays the movie $M$ a total of $N$ times. If $N$ is negative, each play is once forward and once backward. Replace FPS with an integer to give the speed of the movie in frames per second. The default speed is 12 fps.

**Syntax:**
```
movie2avi(M,’filename’,’fps’,f) matlab
```
**Description:** Creates an AVI movie from the MATLAB movie $M$ and saves it in the file called $filename$.avi. Keep $fps$ as is and replace $f$ with an integer to specify the number of frames per seconds for the AVI movie. The default is 15 fps.