This is an introduction to those aspects of Scilab which will be needed in class. There is a great deal more to it and you can explore further if you like. It could be very useful in others of your classes as well. A good place to start would be the documentation listed on the web page http://www.scilab.org/

In addition, see the help menu in the scilab window itself. Scilab is often used interactively but it is also possible to program it to do more complicated tasks. In addition, it is designed to be extended by the user to include more sophisticated data types and operations on them.

Scilab is very like MATLAB but there are significant differences. In both the basic object is the matrix and the syntax for operations on matrices is the same in both languages.

In what follows I will give enough to solve the problems in the text. This is similar to the appendix in the text (Numerical Methods using Matlab, by Mathews and Fink) but the explanations here are correct for Scilab.

The symbol “-- >” is the prompt given by Scilab when it is listening for a command. The symbol “//” is used to make the remainder of the command line a comment.

**ARITHMETIC OPERATIONS**

These are mainly

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation

Some important constants are %pi, %e, %i, %eps. (This differs from their definitions in MATLAB.)

Ex. -- > (2 + 3 * %pi)/2
ans =
5.712389

**BUILT-IN FUNCTIONS**

There are over a thousand functions built into Scilab. A few of the more useful are

- abs()  
- cos()  
- exp()  
- log()  
- log10()  
- sinh()  
- sin()  
- tanh()  
- sqrt()  
- floor()  
- acos()  

Ex. -- > 3 * cos(sqrt(4.7))
Ans =
-1.65865689

Keep in mind that you must give the arguments of the trigonometric functions in radians, just as we did in the Calculus. Thus, to get the sine of 30 degrees we can write “sin(30*%pi/180)”.

The default format shows about 8 significant decimal digits. This can be increased to about 15 digits with the “format” command:

Ex. -- > %pi
%pi =
3.1415927
-- > format(18)
-- > %pi
%pi =
3.141592653589793
ASSIGNMENT STATEMENTS

The values of expressions can be assigned to variables using the equals sign.

Ex. \[-\rightarrow > a = 3 - \text{floor}(\exp(2.9))\]

\[
\text{ans} = 
-15
\]

Often one does not wish to see the output from a command. The output can be suppressed with a semicolon at the end of the command.

Ex. \[-\rightarrow > a = 3 - \text{floor}(\exp(2.9)); \quad //\text{output is suppressed}\]

\[-\rightarrow > a \ast a \quad //\text{output is given, next}\]

\[
\text{ans} = 
225
\]

DEFINING FUNCTIONS

In Scilab a function can be defined in a file with a name ending in “.sci” and written with an editor. In Microsoft Windows you can use the editor “Notepad”. Do not use a word processor. Then, after starting Scilab, you can do a “getf” to load the function or you can ”Exec” its definition. Exec is found on the File menu. After that it can be used like the built-in functions.

Ex. We will create a function \(area=\pi \ast r^2\) and then use it in Scilab to find the area of a circle of radius 1.5.

First use your favorite editor to create a file named “geom.sci” and containing the lines

```plaintext
function [y]=area(r)
y=\%pi\ast r\ast r
endfunction
```

Save it in a “working directory” of your choosing. I recommend “C:\scilabwd” You must make this directory if it does not already exist. Now start Scilab and enter the commands:

Ex. \[-\rightarrow > \text{cdir('C:\scilabwd')} \quad //\text{change the working directory to C:\scilabwd}\]

\[-\rightarrow > \text{getf geom.sci} \quad //\text{this loads the function(s) contained in the file “geom.sci”}\]

\[-\rightarrow > \text{area(1.5)} \quad //\text{this “calls” our new function named “area”}\]

\[
\text{ans} = 
7.0685835
\]

Note that the filename and function name do not have to be the same. In fact, the file may contain many functions. The function has to be called by the function’s name, not by the filename. The function can also be loaded by choosing “Exec...” under the file menu at the top of the Scilab window. This allows searching for the file.

MATRICES

Scilab has extensive matrix manipulation capabilities. We will only need a small part of them. Matrices are entered row first. It can be done on one line:

Ex. \[-\rightarrow > A=[1 \ 2 \ 3; \ 4 \ 5 \ 6; \ 7 \ 8 \ 9]\]

\[
A = 
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

Note that semicolons separate the rows and spaces separate the entries in the same row. (Commas can also be used.) Matrix entry can be done on several lines. In this case the “end of line” serves as the row separator.

Ex. \[-\rightarrow > A=[1 \ 2 \ 3 \\
4 \ 5 \ 6 \\
7 \ 8 \ 9]\]

\[
A = 
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]
There are several built-in functions for generating matrices:

Ex. -- >I=eye(3,3);  // creates a 3x3 identity matrix
     <-- >Z=zeros(3,5);  // creates a 3x5 matrix of all zeros
     <-- >W=ones(5,4);  // creates a 5x4 matrix of all ones
     <-- >V=[0;0.5;3]  // creates a row vector (i.e. 1xn matrix) as displayed
V =
    ! 0 0.5 1. 1.5 2. 2.5 3. !

This last construction is extremely important to understand. The “0;0.5;3” specifies a list of numbers starting at 0, stepping along by 0.5 and each step and stopping at 3. If the middle number is left out then 1 is understood. Thus 0:3 denotes the row vector [0 1 2 3].

Most functions, when fed a vector, simply operate on each component and produce a vector of the same size (length). Thus

Ex. -- >cos(V)
ans =
    ! 1. 0.8775826 0.5403023 0.0707372 - 0.4161468 - 0.8011436 - 0.9899925 !

Note that we can think of V as a vector of “x-values” and then the new vector will be a vector of the corresponding “y-values” of the points on the graph of the cosine function. This is often used in making graphs of functions.

Submatrices can be extracted and also assigned new values.

Ex. -- >A(2,3)  // the element in row 2, column 3
ans =
    6.
     <-- >A(1:2,2:3)  // elements of rows 1 and 2 and columns 2 and 3
ans =
    ! 2.  3. !
    ! 5.  6. !
     <-- >A([1 3],[1 3])  // we give vectors of indices for the rows and columns to pull out
ans =
    ! 1.  3. !
    ! 7.  9. !
     <-- >A(2,3)=12  // assigns a new value to the row 2, col 3 position
A =
    ! 1.  2.  3. !
    ! 4.  5. 12. !
    ! 7.  8.  9. !

See the on-line help or the printed documentation for many additional commands for matrix manipulation.

MATRIX OPERATIONS

The arithmetic operations mentioned at the beginning of this tutorial all have meaning for matrices and the notation is exactly the same, except that division is more complicated. An additional operation for the case of matrices is the conjugate transpose, denoted ‘. Thus, we have, continuing with our definitions above,

Ex. -- >A’
ans =
    ! 1.  4.  7. !
    ! 2.  5.  8. !
    ! 3. 12.  9. !
     <-- >A*A’
ans =
    ! 14.  50.  50. !
    ! 50. 185. 176. !
    ! 50. 176. 194. !
     <-- >A^2  // the matrix A multiplied by itself
ans =
! 30. 36. 54. !
! 108. 129. 180. !
! 102. 126. 198. !

It would not make sense to ask for the square of our vector V but we could ask to square each of its components. This is accomplished by

Ex. --- >V^2
ans =
! 0. 0.25 1. 2.25 4. 6.25 9. !

Note the “dot” before the “ operator. This makes it apply to each component individually. We can do the same with a matrix. Note that the result is very different from A^2.

Ex. --- >A^2     //each entry in A is squared
ans =
! 1. 4. 9. !
! 16. 25. 144. !
! 49. 64. 81. !

We can do the same with division:

Ex. --- >A./2
ans =
! 0.5 1. 1.5 !
! 2. 2.5 3.6 !
! 3.5 4. 4.5 !

Also, to double each entry in A:

Ex. --- >A.*2
ans =
! 2. 4. 6. !
! 8. 10. 12. !
! 14. 16. 18. !

GRAPHICS

Scilab has extensive capabilities for plotting graphs in both 2 and 3 dimensions. One command for generating 2 dimensional plots is “plot2d”:

Ex. --- >x=[0:0.1:2*%pi];       //make a COLUMN vector of “x” values; stepsize is 0.1
     --- >y=sin(x);         //make a COLUMN vector of “y” values
     --- >plot2d(x,y)       //plot the y-values versus the x-values

It is a good rule (although not always necessary) to make all vectors column (as opposed to row) vectors. Thus, we supplied plot2d with two column vectors, the first containing the x-values and the second containing the y-values.

Ex. --- >xgrid()      //add a grid so we can better read off values from the graph
     --- >title(’The sine function’) //put this string as a title at the top of the graph window

Try these commands in your copy of Scilab to see the resulting plot. Note that the graph pops up in its own window and you can use the ”print” command in the ”file” dropdown list to print out the graph.

It is possible to plot different functions on the same x-axis. (This is very useful for determining where their graphs cross—a very important problem in numerical analysis.)

Ex. --- >/multiple plot
     --- >xbasc()          //this clears out previous graph stuff from the graph window
     --- >plot2d(x,[sin(x) sin(2*x) sin(3*x)])    //the second argument to plot2d is actually a matrix
     --- with 3 columns of “y-values”

Ex. --- >xgrid()       //puts on the grid
     --- >title(’sin(x) and sin(2x) and sin(3x)’) //shows a nice title

Often a better way to specify a curve or surface is “parametrically”. No problem.
Ex. -- >xbasc();  //clear out old graph
   -- >plot2d(2*cos(x), 3*sin(x));  //plot the new one
   -- >xgrid();
   -- >xtitle('A beautiful ellipse')
Try it. You'll like it.

An example of a surface, expressed parametrically and plotted:
Ex. -- >>x=[0:100]*%pi/100;  //a vector of x-values from 0 to \pi, stepsize = \pi/100
   -- >>y=[0:100]*2*%pi/100;  //same but y-values from 0 to 2\pi
   -- >>z=10*sin(x)^*cos(y);  //the resulting z-values; a 101x101 matrix
   -- >>plot3d(x,y,z);  //go for it
Try it. It's awesome.

LOGIC, LOOPS and CONDITIONALS

Some relational (Boolean) operators are the following. Their values are true (%t) or false (%f). Try some.

==  Equal to
~== Not Equal to (<>) is also used
<  Less than
>  Greater than
<=  Less than or equal
>=  Greater than or equal

Thus
Ex. -- > 1==2
ans =
   F
and
Ex. -- >3==3
ans =
   T

The “for loop” is perhaps the most common loop. Note how it takes a vector of indices over which to run:
Ex. -- >>x=1; for n=1:4, x=x*n, end
x =
   1.
   2.
   6.
   24.

A “while loop” is also possible and operates as expected, that is, until its condition (here x<14) evaluates to false.
Ex. -- >>x=1; while x<14, x=2*x, end
x =
   2.
   4.
   8.
   16.
Either type of loop can be exited by the use of the “break” command. Notice also the use of the “if ... then” clause.

Ex. \--- >a=0; for i=1:5:100, a=a+1; if i > 10 then break, end; end

\--- >a

```
a =

3
```

The command “disp” is useful to display strings or values of variables as the loop is executed. In the last example we could have inserted it as follows to show each value of the loop index:

Ex. \--- >a=0; for i=1:5:100, a=a+1; disp(i), if i > 10 then break, end; end

```
```
SOME PROGRAMMING EXAMPLES

function []=evenodd(n)
//An example showing the "if" statement. Always put "then" on same line as "if"
//Input:
//  An integer n
//Output:
//  A displayed message of parity
//First, is it really an integer?
  if n-floor(n)<0 then
    disp('Gvan. That's not an integer!')
    return
  end
//It really is so test it
  if n/2-floor(n/2)==0 then
    disp('It is even.')
  else
    disp('It is odd.')
  end
endfunction

POLYNOMIALS

Scilab has extensive facilities for handling polynomials. (This is very different from MATLAB.) There are various ways to proceed. One is to first define a symbol for the polynomial variable. One then can manipulate it in the usual ways. Thus

Ex. -- > x=%s;
   -- > p = x^2 + 5*x - 1
   -- > p = 2
       - 1 + 5s + s
To evaluate the polynomial, use horner:

Ex. -- >horner(p,3)
ans =
     23.

MISCELLANEOUS

For a positive integer n we can get n! by e.g.

Ex. -- >prod(1:5)
ans =
       120.
There is a very useful operation on matrices which does not correspond to those you met in linear algebra. First an example.

Ex.--- >v=1:5
    v =
    ! 1. 2. 3. 4. 5. !
    -- >v*v
    !-error 10 inconsistent multiplication
To simply multiply two vectors does not make sense. What is often needed is an “elementwise multiplication”. Thus

Ex -- >v.*v
ans =
    ! 1. 4. 9. 16. 25. !
An application of this is when you have a vector of x-values and wish to plot the function “x*sin(x)” say. You can use elementwise multiplication to get the needed y-values. Thus

Ex -- >x=[0:0.01:%pi]';
    -- >y=x.*sin(x);
    -- >plot(x,y)

I hope you find this tutorial useful. I welcome criticisms, suggestions, etc.

Prof. Le Baron O. Ferguson
Surge 230
X25003
ferguson@math.ucr.edu