Programming with SCILAB

By

Gilberto E. Urroz, Ph.D., P.E.

Distributed by

©2001 Gilberto E. Urroz
All Rights Reserved

A "zip" file containing all of the programs in this document (and other SCILAB documents at InfoClearinghouse.com) can be downloaded at the following site:

http://www.engineering.usu.edu/cee/faculty/gurro/Software_Calculators/Scilab_Docs/ScilabBookFunctions.zip

The author's SCILAB web page can be accessed at:

http://www.engineering.usu.edu/cee/faculty/gurro/Scilab.html

Please report any errors in this document to: gurro@cc.usu.edu

SCILAB PROGRAMMING, IO, AND STRINGS **2**

SCILAB programming constructs **2**
Comparison and Logical Operators 2
Loops in SCILAB 3
Conditional constructs in SCILAB 3

Functions in SCILAB **5**
Global and local variables 6
Special function commands 6
Debugging 7
An example of a function - Calculation of Frobenius norm of a matrix. 8

Input/Output in SCILAB **9**
Saving and loading variables. 9
Unformatted output to the screen 9
Unformatted output to a file 9
Working with files. 10
Writing to files. 10
Reading from the keyboard 11
Reading from files 12

Manipulating strings in SCILAB **12**
String concatenation 13
String functions 13
Converting numerical values to strings 14
String catenation for a vector of strings 15
Converting strings to numbers 15
Executing SCILAB statements represented by strings 16
Producing labeled output in SCILAB 17
Using the function *disp* 18
The variable *ans* 18

Exercises **19**

 © 2001 Gilberto E. Urroz

SCILAB Programming, IO, and strings

Programming is the basic skill for implementing numerical methods. In this chapter we describe the fundamental programming constructs used in SCILAB and present examples of their applications to some elementary numerical methods. The second part of this chapter is dedicated at exploring input/output functions provided by SCILAB including operations with files. Finally, manipulation of strings in SCILAB is presented.

SCILAB programming constructs

SCILAB provides the user with a number of programming constructs very similar to those available in FORTRAN and other high-level languages. We present some of the constructs below:

Comparison and Logical Operators

SCILAB comparison operators are

Operator	Meaning
==	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<> or ~=	not equal to

SCILAB logical operators are

Operator	Meaning
&	and
|	or
~	not

As an example, try the following commands in SCILAB:

3 <> 2 <enter>
3 == 3 <enter>
(2>1)&(3>1) <enter>
(2>1)&(3>5) <enter>
(2<1)&(3>1) <enter>
(2<1)&(3>5) <enter>
(2>1) | (3>1) <enter>
(2>1) | (3>5) <enter>
(2<1) | (3>1) <enter>
(2<1) | (3>5) <enter>
~(2<1) <enter>
~(2>1) <enter>
~(2>1) | (3>5) <enter>

Download at InfoClearinghouse.com

© 2001 Gilberto E. Urroz

Loops in SCILAB

SCILAB includes *For* and *While* loops. The *For* loop is similar to the DO loop in FORTRAN or the FOR..NEXT loop in Visual Basic. The basic construct for the *For* loop is:

```
for index = starting_value : increment : end_value, …statements…, end
for index = starting_value : end_value, …statements…, end
```

If no increment is included it is supposed to be equal to 1.
For example, enter the following For loops in SCILAB:

```
r = 1; for k = 1:0.5:4, r = r+k, end <enter>
xs = -1.0; dx = 0.25; n = 10; for j = 1:n, x = xs + (j-1)*dx, end <enter>
for m = 1:10, a(m) = m^2, end <enter>
a <enter>
```

The basic construct for the While loop is:

```
while condition, …statements…, end
```

For example, try the following *while* loop:

```
s = 100; while s>50, disp(s^2), s = s - 5, end <enter>
```

For and *while* loops can be terminated with the command *break*, for example, try the following:

```
for j = 1:10, disp(j), if j>5 then break, end, end <enter>
```

Conditional constructs in SCILAB

In the example above we used an *if… then…end* construct. There are two type of conditional constructs in SCILAB, one is the *if-then-else-end* construct (as in the example above) and the second one is *the select-case* conditional construct. Different forms of the *if-then-else* construct are:

if *condition* then *statement*, end
if *condition* then *statement*, else *statement*, end
if *condition* then *statement*, elseif *condition* then *statement*, else *statement*, end

Try the following examples:

```
x = 10; y = 5; if x> 5 then disp(y), end <enter>
x = 3 ; y = 5;  if x>5 then disp(y), else disp(x), end <enter>
x = 3; y = 5; z = 4; if x>5 then disp(x), elseif x>6 then disp(y), else disp(z), end <enter>
```

The general form of the *select-case* construct is:

select *variable*, case *n1*, *statement*, case *n2*, *statement*, …, end

Try the following examples:

© 2001 Gilberto E. Urroz

x = -1; select x, case 1, y = x+5, case -1, y = sqrt(x), end <enter>
r = 7; select r, case 1, disp(r), case 2, disp(r^2), case 7, disp(r^3), end <enter>

All these constructs can be programmed in files following a structure similar to FORTRAN or Visual Basic programs, and then executed from within SCILAB. Such files are referred to as *scripts*. For example, type the following SCILAB script into a file called *program1.txt*:

```
clear    //erase all variables
x = [10. -1. 3. 5. -7. 4. 2.];
suma = 0;
[n,m] = size(x);
for j = 1:m
       suma = suma + x(j);
end
xbar = suma/m;
xbar
```

Save it into the bin sub-directory. Within SCILAB type:

```
exec('program1.txt') <enter>
```

Note that since x is a row vector (actually a matrix with $n = 1$ row and $m = 7$ columns), the *size* function provides you with an array of two values in the statement `[n,m] = size(x)`. Then, m is used in the *for* loop and in the calculation of *xbar*.

As an alternative to using a row (or column) vector is the use of lists. A list is a collection of data objects not necessarily of the same type. In the following example we limit ourselves to using lists of numbers. To define a list we use the list command, for example, try:

```
y = list(0., 1., 2., 3., 4., 6.) <enter>
size(y) <enter>
```

In this case, the size of the list, unlike that of a vector or matrix, is provided as a single number. A modified version of the script in *program1.txt* is shown below. Type this file into *program2.txt* and save it in the *bin* sub-directory under the SCILAB directory:

```
//Same as program1.txt, but using lists
clear    //erase all variables
x = list(10., -1., 3., 5., -7., 4., 2.);
suma = 0;
n = size(x);
for j = 1:n
       suma = suma + x(j);
end
xbar = suma/n;
n
xbar
```

To run the script, from within SCILAB type:

```
exec('program2.txt')<enter>
```

Functions in SCILAB

Functions are procedures that may take input arguments and return zero, one or more values. Functions are defined either *on line*, using the *deff* command, or as a *separate file* that needs to be loaded using the *getf* command. Following some examples of *on-line* functions are presented:

deff('[z]=Euler(r,theta)','z=r*exp(%i*theta)') <enter>
Euler(1.0,-%pi/2) <enter>

deff('[r,theta]=cartpol(x,y)',['r=sqrt(x^2+y^2)'; 'theta=atan(y,x)']) <enter>
[radius,angle] = cartpol(3., 4.) <enter>

These functions could have been defined by using the *Define User Function...* option in SCILAB's *Functions* menu. For example, select this option and enter the following (your reply is shown in italics):

(1) Name of output variable? *x,y* [OK];
(2) Name for function? *polcart*[OK];
(3) Variable/s? *r,theta* [OK];
(4) Code? *['x=r*cos(theta)','y=r*sin(theta)']*[OK].

SCILAB's response is:

→ `deff('[x,y]=polcart(r,theta)',['x=r*cos(theta)';'y=r*sin(theta)]).`

Try the following application:

[h,v] = polcart(10.0,%pi/6) <enter>
polcart(100.0,%pi/3) <enter>

The last command will give you only the result for y since the function call was not assigned to an array as in the first case.

Functions defined in files must start with the command

```
Function [y1,…,yn] = fname(x1,…,xm)
```

Where *fname* is the function name, [y1,...,yn] is an array of output values, and x1,...,xm are the input values. Type in the following function into a file called sphecart.txt using a text editor (e.g., NOTEPAD, or PFE):

```
function [x,y,z] = sphecart(r,theta,rho)
//conversion from spherical to Cartesian coordinates
x = r*cos(rho)*cos(theta)
y = r*cos(rho)*sin(theta)
z = r*sin(rho)
```

In SCILAB load the function using:

getf('sphercart.txt') <enter>
[x1,y1,z1]=sphercart(10.0, %pi/3, %pi/6) <enter>

 © 2001 Gilberto E. Urroz

Notice that SCILAB on-line functions are similar to FORTRAN function declarations, while SCILAB functions defined in files are similar to FORTRAN or Visual Basic function sub-programs or subroutines. The main difference is that FORTRAN and Visual Basic functions can only return one value, while SCILAB functions can return zero, one or more values.

Global and local variables

A global variable is one define in the main SCILAB environment, while a local variable is one defined within a function. If a variable in a function is not defined, or is not among the input parameters, then it takes the value of a variable of the same name that exist in the calling environment. This variable remains local in the sense that modifying it within the function does not alter its value in the calling environment unless the command *resume* is used.
For example, using the function *sphercart*, try the following:

```
clear
getf('sphercart.txt') <enter>
theta = %pi/3 <enter>
rho = %pi/6 <enter>
[x,y,z] = sphercart(10.0,theta)<enter>
```

Since *rho* is defined in the calling environment, even though that value is missing in the calling sequence to the function *sphercart*, it takes the value of *rho* in the calling environment.

Note that it is not possible to call a function if one of the parameters in the calling sequence is not defined. Try the following:

```
clear
getf('sphercart.txt') <enter>
theta = %pi/3 <enter>
[x,y,z]=sphercart(10.0,%pi/3,rho) <enter>
```

Because *rho* is not defined in this case, the function can not be evaluated.

Special function commands

These are SCILAB command used almost exclusively in functions:

`argn`: returns the number of input and output arguments of the function
`error`: suspends a function's operation, prints an error message, and returns to previous environment level if an error is detected
`warning`: prints a warning message
`pause`: temporarily suspends the operation of a function
`break`: forces the end of a loop
`return` or `resume`: use to return to the calling environment and to pass local variables from the function environment to the calling environment.

For additional information use the help feature in SCILAB with these functions. The following example illustrate the use of some of these special function commands. Enter the function in a file called func1.txt, and save it in the bin sub-directory of SCILAB:

 © 2001 Gilberto E. Urroz

```
function [z] = func1(x,y)
[out,in]=argn(0)
if x == 0 then
       error('division by zero');
end,
slope = y/x;
pause,
z = sqrt(slope);
s = resume(slope);
```

Then, within SCILAB enter the following:

```
clear <enter>
getf('func1.txt') <enter>
z = func1(0,1) <enter>
z = func1(2,1) <enter>
```

In the second call to *func1*, the `-1->` prompt indicates a *pause* mode. The function operation is temporarily suspended. The user can, at this point, examine values calculated inside the function, plot data, or perform any SCILAB operation. Control is returned to the function by typing the command *return* <enter> (*resume* can also be used here). Operation of the function can be stopped by using *quit* or *abort*. When *return* (or *resume*) is used, the function calculates and reports the value of *z*. Also available in the environment is the local variable *s* which is passed to the global environment by the resume command within the function. Type *s* <enter> to see the value of *s*.

Debugging

The simplest way to debug a SCILAB function is to use a *pause* command in the function. When this command is encountered the function stops and the prompt `-1->` is shown. This indicates a different "level" of calculation that can be used to recall variable values including global variables from the calling environment, experiment with operations, produce a graph if needed, etc. Using a second pause will produce a new level characterized by the prompt `-2->`, and so on. The function resumes execution by typing the command *return* or *resume*, at which point the variables used at the higher level prompts are cleared. Execution of the function can be interrupted with the command *abort*.

An additional feature for debugging that is available in SCILAB is the insertion of breakpoints in the function. These are pre-identified points in the function to which you can access during the function execution to check the values of the variables or perform other operations. Check the commands *setbpt*, *delbpt*, and *disbpt*.

You can also trap errors during the function execution by using the commands *errclear* and *errcatch*. Check these commands using SCILAB help. At a higher level of expertise in SCILAB debugging the user can try the function *debug(i)* where *i* = 0, 1, 2, 3, 4, denotes a debugging level. Check out the *debug* function using help.

 © 2001 Gilberto E. Urroz

An example of a function - Calculation of Frobenius norm of a matrix.

This function is to be stored in file *AbsM.txt* within subdirectory *bin* in the SCILAB directory. (Note: While the name of the file containing a function does not have to be the same as the name of the function, it is recommended that they be the same to facilitate loading and operation of the function).

The Frobenius norm of a matrix **A** = [a_{ij}] with n rows and m columns is defined as the square root of the sum of the squares of each of the elements of the matrix, i.e.,

$$\| A \|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}}\,.$$

The function AbsM(A), listed below, calculates the Frobenius norm of a matrix:

```
function [v]=AbsM(A)
// This function calculates the Frobenius norm of a matrix
// First obtain the matrix size
[n m] = size(A);
// Then initialize suma and add terms a(i,j)^2
suma = 0;
for i = 1:n
	for j = 1:m
		suma = suma + A(i,j)^2;
	end
end;
// take square root and show result
v = sqrt(suma);
// end of the function
```

Within SCILAB try the following commands to load and run the function for a particular case:

```
clear <enter>
getf('AbsM.txt') <enter>
R = [1. 3. 4. 2. <enter>
3. -2. 5. -7. <enter>
1.  3. 4. 5. ] <enter>
AbsM(R) <enter>
```

Functions are defined throughout the book in relation to different mathematical subjects, i.e., vectors, matrices, integrals, differential equations, etc. The following sections of this chapter deal with the subjects of input/output and string manipulation in SCILAB.

 © 2001 Gilberto E. Urroz

Input/Output in SCILAB

Saving and loading variables.

To save variables in a file use the command *save*. Let's try some examples:

```
A = [1. 2. 3.; -3. 4. 5.; 2. 4. 5.; 1. 3. 2.]; b = 1:10; <enter>
A <enter>
b <enter>
save('DataAb.dat', A,b)<enter>
```

Next, using NOTEPAD or PDE, open the file *DataAB.dat* in sub-directory *bin* of SCILAB. You will notice that you cannot see the numbers in the file. That is because they have been saved in a binary format. Let's clear the variables in SCILAB and re-load the values of A and b using the command *load*:

```
clear <enter>
load('DataAb.dat') <enter>
A <enter>
b <enter>
```

Unformatted output to the screen

To print strings and variables without a format you can use the *print* function. The general form of the function is: *print (unit or filename, x1, x2, (y1, ..,))*. The unit value for the screen is either 6 or %io(2). Try the following examples:

```
x = 5; y = sin(%pi*x/10); r = 1:2:25; A = rand(5,3); <enter>
%io(2) <enter>
print(6,x,y) <enter>
print (6,A,r)<enter>
print(%io(2),x,y,r)<enter>
print(%io(2),A) <enter>
```

Notice that the function *print*, as with the function *disp* used earlier, prints the last variable in the list first. Try some more examples:

```
Print(6,x,'x value =') <enter>
```

Notice that, in this case, the string 'x value =' is printed together with the string 'x = ', which is a default from the print command. Therefore, it is not a good idea to include an identifying string when using the *print* function to print to the screen.

Unformatted output to a file

You can use the print function to print to a filename, for example, try:

```
print('data1.txt',A,r)<enter>
print ('data2.txt',x,y)<enter>
```

Download at InfoClearinghouse.com

© 2001 Gilberto E. Urroz

Next, using NOTEPAD open the files data1.txt and data2.txt. Notice that the output includes all the identifiers and brackets (!) provided by SCILAB.

Working with files.

The following command allows you to open a file:

[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])

file-name: string, file name of the file to be opened

status: string, The status of the file to be opened
"new" : file must not exist new file (default)
"old" : file must already exists.
"unknown" : unknown status
"scratch" : file is to be deleted at end of session

access: string, The type of access to the file
"sequential" : sequential access (default)
"direct" : direct access.

format: string,
"formatted" : for a formatted file (default)
"unformatted" : binary record.

recl: integer, is the size of records in bytes when access="direct"

unit: integer, logical unit descriptor of the opened file

err: integer, error message number (see error), if open fails. If err is omitted an error message is issued.

You can also use the command: *file(action,unit)*

where *action* is one of the following strings:

"close": closes the file.
"rewind": puts the pointer at beginning of file.
"backspace": puts the pointer at beginning of last record.
"last": puts the pointer after last record.

Once a file is open it can be used for input (read function) or output (write function). Some examples of file opening, input and output are shown below.

Writing to files.

The following programs use the values of x, y, A, and r defined above. In these examples we open and write to files, and close them. Notice that this command is oriented towards printing matrices -- one at a time -- therefore, as shown in Example 2, it is better if you put together your data into a matrix before printing it. Notice also that the format part, which is enclosed between quotes, is basically a FORTRAN format.

 © 2001 Gilberto E. Urroz

- Example 1.

```
u = file('open','data3.txt','new')<enter>
write(u,A,'(3f10.6)') <enter>
file('close',u)<enter>
```

- Example 2.

```
x1 = 0:0.5:10 <enter>
x2 = x1^2 <enter>
B = [x1',x2'] <enter>
m = file('open','data4.txt','new')<enter>
write(m,B,'(2(f10.6,2x))') <enter>
file('close',m)<enter>
```

- Example 3. Including labels. Note: labels are written separated from the variables

```
A = rand(2,3); B = rand(2,3); C = A + B <enter>
u = file('open','data5.txt','new) <enter>
write(u,'this is matrix A','(a)') <enter>
write(u,A,'(3(f10.6,2x)') <enter>
write(u,'this is matrix B','(a)') <enter>
write(u,B,'(3(f10.6,2x)') <enter>
write(u,'this is matrix C = A + B','(a)') <enter>
write(u,C,'(3(f10.6,2x)') <enter>
file('close',u) <enter>
```

Reading from the keyboard

Reading from the keyboard can be accomplished by using the *read* function with unit %io(1) or 5. The general form of the read function is:

[x]=read(file-description,n,m,[format]),

i.e., a variable must be assigned a value (could be a matrix of size n,m) during the operation of *read*. The file description can be a unit or number assigned to a file or to the keyboard. The format is not necessary. Also, to read a single value use m = 1, n= 1, as shown below.
For example, type the following function into a file called *inout.txt*:

```
function inout()
//this script illustrates using read and write
write(%io(2),'Enter a real variable x:','(a)');
x = read (%io(1),1,1);
write(%io(2),'Enter a real variable y:','(a)');
y = read (%io(1),1,1);
z = x+y;
write(%io(2),'the sum of x and y is:','(a)')
write(%io(2),z,'(10x,e13.7)')
//end of function
```

 © 2001 Gilberto E. Urroz

Within SCILAB, type the following commands, and responses to prompts:

```
Getf('inout.txt') <enter>
inout( ) <enter>
1.2 <enter>
2.4 <enter>
```

Notice that the function *inout* has no arguments. Still, in the function definition as well as in the function call it has to have a pair of parentheses.

Reading from files

Use the same read command as used while reading from the keyboard, but using an open file unit to read. For example, suppose that you have a file called *signal.txt*, containing the following values:

```
1.0    2.0    4.0
2.0    3.0    9.0
3.0    4.0    16.0
4.0    5.0    25.0
5.0    6.0    36.0
6.0    7.0    49.0
```

If you know the number of rows (n=6, in this case). To read the matrix of values, use:

```
u=file('open','signal.txt','old') <enter>
A=read(u,6,3); <enter>
A<enter>
```

If the number of rows is unknown using `n=-1` will ensure that the entire file is read. It is assumed that the file contains only the matrix of interest. For example,

```
file('rewind',u) <enter>
B = read(u,-1,3); <enter>
B <enter>
file('close',u) <enter>
```

Manipulating strings in SCILAB

A string is basically text that can be manipulated through SCILAB commands. Strings in SCILAB are written between single or double quotes. The following are examples of strings:

```
'myFile'  'The result is: '  'a b c' 'abc' 'a' 'b' 'c'
"Text to be included"   "Please enter the graphic window number"  "1" "3" "5"
```

 © 2001 Gilberto E. Urroz

String concatenation

The joining of two or more strings is called *concatenation*. The *plus* symbol (+), when placed between two strings concatenates the strings into a single one. In the next example variables s1, s2, and s3 are defined and concatenated:

```
-->s1 = 'The result from '
 s1  =

 The result from

-->s2 = 'multiplication '
 s2  =

 multiplication

-->s3 = 'is given below.'
 s3  =

 is given below.

-->sOut = s1 + s2 + s3
 sOut  =

 The result from multiplication is given below.
```

String functions

The function *length* determines the number of characters in a given string, for example:

```
-->length(sOut)
 ans  =

    46.
```

The function *part* allows the extraction of characters from a given string. For example, to extract the first character of a string use:

```
-->part('abcd',1)
 ans  =

 a
```

The next command extracts the first and second character of a string:

```
-->part('abcd',[1,2])
 ans  =

 ab
```

In the next example, characters 1 and 3 of the string are extracted:

```
-->part('abcd',[1,3])
```

 © 2001 Gilberto E. Urroz

```
 ans  =

 ac
```

To extract a series of character, the characters' positions in the string are indicated as a sequence of values in the vector representing the second argument to function *part*:

```
-->part(sOut,[4:1:15])
 ans  =

  result from
```

The function *strindex* (*str*ing *index*), with a typical call of the form `strindex`*(string1,string2)* determines the position of the first occurrence of sub-string *string2* within *string1*. For example,

```
-->strindex(sOut,'mult')
 ans  =

    17.
```

Once the position of a sub-string has been determined you can use the function *part* to extract that sub-string or other sub-string starting at that position. For example, this function call extracts characters 17 to 24 of string *sOut*:

```
-->part(sOut,[17:24])
 ans  =

 multipli
```

The function *strsubst* (*str*ing *subst*itution), with a typical call of the form

`strsubst`*(string1,string2,string3)*

replaces sub-string *string2* with sub-string *string3* within string *string1*. For example, the next call to function *strsubst* replaces the sub-string `'multiplication'` with `'division'` within string *sOut*:

```
-->strsubst(sOut,'multiplication','division')
 ans  =

 The result from division is given below.
```

Converting numerical values to strings

The function *string* is used to convert a numerical result into a string. This operation is useful when showing output from numerical calculations. For example, the next SCILAB input line performs a numerical calculation, whose immediate output is suppressed by the semi-colon, and then produces an output string showing the result. The output string produced consists of the sub-string `"The sum is"` concatenated to the numerical result that has been converted to a string with `string(s)`.

```
-->s = 5+2; "The sum is " + string(s)
```

 © 2001 Gilberto E. Urroz

```
 ans  =

 The sum is 7
```

The following command produces an array or vector of strings. The strings in the vector represent the numbers from 1 to 5.

```
-->sNum = string(1:5)
 sNum  =

!1  2  3  4  5  !
```

An attempt to add the first two elements of vector *sNum* produces instead their concatenation, verifying that the elements are indeed strings, and not numbers:

```
-->sNum(1)+sNum(2)
 ans  =

 12
```

String catenation for a vector of strings

To generate a string consisting in inserting a particular sub-string between the characters of a vector or array of strings use the function *strcat* (*str*ing *cat*enation). The next example produces a string resulting from inserting the character `'-'` between the elements of *sNum*:

```
-->strcat(sNum,' - ')
 ans  =
 1 - 2 - 3 - 4 - 5
```

Converting strings to numbers

To convert a string representing numbers into their numerical equivalent you can use function *evstr* (*ev*aluate *str*ing). The next command, for example, converts the string elements of vector *sNum*, defined earlier, into their numerical equivalents:

```
-->nNum = evstr(sNum)
 nNum  =

!   1.    2.    3.    4.    5. !
```

The plus sign (+) applied to the two first elements of *nNum* would add, rather than concatenate, those elements:

```
-->nNum(1) + nNum(2)
 ans  =

    3.
```

 © 2001 Gilberto E. Urroz

The function *evstr* can be used to evaluate numerically any string representing number operations. Some examples are shown below:

```
-->evstr('2+2')
 ans  =

    4.

-->evstr('sin(%pi/6) + 1/3')
 ans  =

      .8333333
```

The following example uses function *evstr* to evaluate the numerical values defined in the elements of a vector. This particular example uses the values of a couple of variables, *s* and *m*, which must be defined before attempting the evaluation of the strings.

```
-->s = 2, m = 3
 s  =

    2.
 m  =

    3.

-->evstr(['2' 'sqrt(s)' 'm + s'])
 ans  =

!   2.    1.4142136    5. !
```

Executing SCILAB statements represented by strings

To evaluate assignment statements or SCILAB commands defined by strings we use function *execstr* (*exec*ute *str*ing). For example,

```
-->execstr('a=1')
```

Although the statement *a=1* is executed through the use of *execstr*, no output is produced. To check that the statement was indeed executed, request that SCILAB show the value of a:

```
-->a
 a  =

    1.
```

You can use *execstr* to evaluate a series of commands by placing the commands in an array or vector:

```
-->execstr(['a=1','b=2','a+b'])
```

Once again, no output is shown, so the result from the last element in the vector is lost, but variable b (from the second element in the vector) was indeed stored:

 © 2001 Gilberto E. Urroz

```
-->b
 b  =

    2.
```

A second example of multiple statements executed through *execstr* follows:

```
-->execstr(['s=2' 'm=3' 'r=sqrt(s)' 'q=m+s'])
```

Check the results of the statements by using:

```
-->[s m r q]
 ans  =

!   2.    3.    1.4142136    5. !
```

The following example shows the execution of a small program whose lines are presented as string elements of a vector:

```
-->execstr(['a=2' 'x=[]' 'for j = 1:4' 'x = [x a^j]' 'end'])
```

The result from the last command can be seen by entering:

```
-->x
 x  =

!   2.    4.    8.  16.  !
```

Producing labeled output in SCILAB

The following example shows a way to produce labeled output in SCILAB. The data for the output is contained in vector *d* of dimensions 1xm:

```
-->d = [0.5:0.25:1.5];

-->[n m] = size(d);        // m is the list of the

-->for j = 1:m, 'distance no. ' + string(j) + ' is ' + string(d(j)) + '.', end
 ans  =

 distance no. 1 is  .5.
 ans  =

 distance no. 2 is  .75.
 ans  =

 distance no. 3 is 1.
 ans  =

 distance no. 4 is 1.25.
 ans  =

 distance no. 5 is 1.5.
```

 © 2001 Gilberto E. Urroz

Using the function *disp*

The previous result uses the variable *ans* to show each line of output. This is the standard way that SCILAB uses to show the current output. The result shown above can be simplified even further by using the function *disp* (*disp*lay), as follows:

```
-->for j=1:m, disp('distance no. '+ string(j) + ' is ' + string(d(j)) + '.'),
end

 distance no. 1 is  .5.

 distance no. 2 is  .75.

 distance no. 3 is 1.

 distance no. 4 is 1.25.

 distance no. 5 is 1.5.
```

The function *disp* can be used to display any result, not only strings. The following example shows the function *disp* used with string as well as numerical output:

```
-->a = 2; A = [2,3;-1,4]; B = a*A;

-->disp('Matrix B is:'), disp(B)

 Matrix B is:

!   4.    6. !
! - 2.    8. !
```

The variable *ans*

The variable *ans* (*ans*wer) contains SCILAB's current output. You can refer to the last SCILAB output by using the variable name *ans*. For example, the following commands uses the contents of *ans* to operate on the most recent SCILAB output:

```
-->3+2
 ans  =
    5.

-->exp(ans)
 ans  =
    148.41316
```

To verify that the result obtained is correct use:

```
-->exp(5)
 ans  =
    148.41316
```

 © 2001 Gilberto E. Urroz

Exercises

[1]. Write a SCILAB function to calculate the factorial of an integer number:

$$n! = n\cdot(n-1)\cdot(n-2)\ldots3\cdot2\cdot1$$

[2]. Write a SCILAB function to calculate the standard deviation of the data contained in a vector $x = [x_1\ x_2\ \ldots\ x_3]$.

$$s = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}\left(x_k - \bar{x}\right)^2},$$

where $\bar{x}$ is the mean value of the data,

$$s = \frac{1}{n}\sum_{k=1}^{n} x_k.$$

[3]. Write a SCILAB function to calculate the function defined by

$$h(\xi) = \begin{Bmatrix} \ln(\xi+1),\ 0<\xi\le 1 \\ \ln(2)+\exp(-\frac{\xi}{2}), \quad 1<\xi\le 4 \\ 0, \quad elsewhere \end{Bmatrix}$$

[4]. Plot the function $h(\xi)$ in the interval $-1<\xi<10$.

[5]. Save the data used in exercise [4] into a text file, then, retrieve the data into vectors x and y and calculate the mean and standard deviation of x and y using the function developed in exercise [2].

[6]. Write a SCILAB function that finds the median of a data sample. The median is defined as that value located exactly in the middle of the data sample once it has been sorted in increasing order. The algorithm to find such value is given by:

$x_m = x_{(n+1)/2}$, if n is even

$x_m = (x_{n/2}+x_{(n+2)/2})$, if n is odd

where n is the sample size. To order the data sample you can use the SCILAB function *sortup* (use `-->help sort` to find more about this function).

[7]. The coefficients of the binomial expansion

$$(a+b)^n = C(n,0)a^n + C(n,1)a^{n-1}b + C(n,2)a^{n-2}b^2 + \ldots + C(n,n-1)ab^{n-1} + C(n,n)b^n,$$

are given by

 © 2001 Gilberto E. Urroz

$$C(n,k) = \frac{n!}{k!(n-k)!}.$$

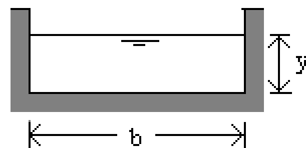
Write a SCILAB function that produces a table of binomial coefficients for $n = 1, 2, \ldots, 5$. Use the function developed in exercise [1] to calculate factorials of integer numbers.

[8]. Write a SCILAB program to define a function given by

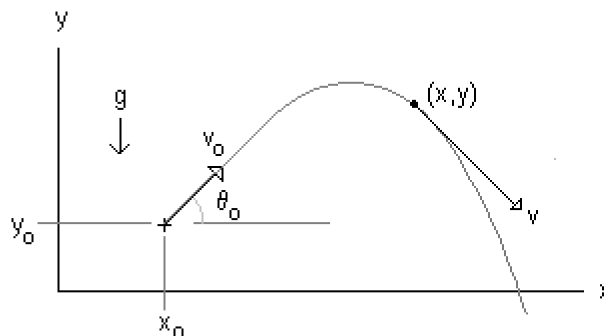
$$f(x) = \begin{cases} x^2 - \sin(x), 0 < x \le 1 \\ 1/(1+x^2), 1 < x \le 2 \\ \sqrt{x^2+1}, 2 < x \le 3 \end{cases}$$

Plot the function for $0 < x < 3$.

[9]. Write a SCILAB function that request from the user the values of the bottom width (b) and water depth (y) for a rectangular cross-section open channel (see figure below) and prints the area ($A = bh$), wetted perimeter ($P = b+2h$), and hydraulic radius ($R = A/P$) properly labeled. Try the function for values of b = 3.5 and y = 1.2.



[10]. Write a SCILAB function that request from the user the values of the initial position (x_0, y_0) of a projectile, the initial velocity given as a magnitude v_0, and an angle θ_0, and the acceleration of gravity g (see figure below). The function also requests from the user an initial time t_0, a time increment Δt, and an ending time t_f. The function produces a table of values of the velocity components $v_x = v_0 \cos(\theta_0)$, $v_y = v_0 \cos(\theta_0)$, the magnitude of the velocity, $v = (v_x^2+v_y^2)^{1/2}$, the position of the projectile, $x = x_o + v_0 \cos(\theta_0)t$, $y = y_o + v_0 \sin(\theta_0)t - gt^2/2$, and the distance of the projectile from the launching point, $r_0 = ((x-x_0)^2+(y-y_0)^2)^{1/2}$. The function also produces plots of x - *vs.* - t, y - *vs.* - t, r_0 - *vs.* - t, and y - *vs.* -x in different graphic windows. [Note: to generate a new graphics window use the SCILAB command `-->xset('window',j)` where j is the window number.]



 © 2001 Gilberto E. Urroz

[11]. Suppose you want to plot the function $r(\theta) = 3.5(1 - \cos(2\theta))$. Write a SCILAB function that generates values of θ from 0 to 2π, calculates the values of r, and the Cartesian coordinates $x = r \cos(\theta)$, $y = r \sin(\theta)$, and prints a table showing those values, i.e., θ, r, x, and y. The function also produces a plot of *y-vs.-x*.

REFERENCES (for all SCILAB documents at INFOCLEARINGHOUSE.com)

Abramowitz, M. and I.A. Stegun (editors), 1965,"*Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*," Dover Publications, Inc., New York.

Arora, J.S., 1985, "*Introduction to Optimum Design*," Class notes, The University of Iowa, Iowa City, Iowa.

Asian Institute of Technology, 1969, "*Hydraulic Laboratory Manual*," AIT - Bangkok, Thailand.

Berge, P., Y. Pomeau, and C. Vidal, 1984,"*Order within chaos - Towards a deterministic approach to turbulence*," John Wiley & Sons, New York.

Bras, R.L. and I. Rodriguez-Iturbe, 1985,"*Random Functions and Hydrology*," Addison-Wesley Publishing Company, Reading, Massachussetts.

Brogan, W.L., 1974,"*Modern Control Theory*," QPI series, Quantum Publisher Incorporated, New York.

Browne, M., 1999, "*Schaum's Outline of Theory and Problems of Physics for Engineering and Science*," Schaum's outlines, McGraw-Hill, New York.

Farlow, Stanley J., 1982, "*Partial Differential Equations for Scientists and Engineers*," Dover Publications Inc., New York.

Friedman, B., 1956 (reissued 1990), "*Principles and Techniques of Applied Mathematics*," Dover Publications Inc., New York.

Gomez, C. (editor), 1999, *"Engineering and Scientific Computing with Scilab,"* Birkhäuser, Boston.

Gullberg, J., 1997, "*Mathematics - From the Birth of Numbers*," W. W. Norton & Company, New York.

Harman, T.L., J. Dabney, and N. Richert, 2000, "*Advanced Engineering Mathematics with MATLAB® - Second edition*," Brooks/Cole - Thompson Learning, Australia.

Harris, J.W., and H. Stocker, 1998, "*Handbook of Mathematics and Computational Science*," Springer, New York.

Hsu, H.P., 1984, "*Applied Fourier Analysis*," Harcourt Brace Jovanovich College Outline Series, Harcourt Brace Jovanovich, Publishers, San Diego.

Journel, A.G., 1989, "*Fundamentals of Geostatistics in Five Lessons*," Short Course Presented at the 28th International Geological Congress, Washington, D.C., American Geophysical Union, Washington, D.C.

Julien, P.Y., 1998,*" Erosion and Sedimentation*,*"* Cambridge University Press, Cambridge CB2 2RU, U.K.

Keener, J.P., 1988, "*Principles of Applied Mathematics - Transformation and Approximation*," Addison-Wesley Publishing Company, Redwood City, California.

Kitanidis, P.K., 1997,*" Introduction to Geostatistics - Applications in Hydogeology*,*"* Cambridge University Press, Cambridge CB2 2RU, U.K.

Koch, G.S., Jr., and R. F. Link, 1971, "*Statistical Analysis of Geological Data - Volumes I and II*," Dover Publications, Inc., New York.

Korn, G.A. and T.M. Korn, 1968, "*Mathematical Handbook for Scientists and Engineers*," Dover Publications, Inc., New York.

Kottegoda, N. T., and R. Rosso, 1997, "*Probability, Statistics, and Reliability for Civil and Environmental Engineers*," The Mc-Graw Hill Companies, Inc., New York.

Kreysig, E., 1983, "*Advanced Engineering Mathematics - Fifth Edition*," John Wiley & Sons, New York.

Lindfield, G. and J. Penny, 2000, "*Numerical Methods Using Matlab®,*" Prentice Hall, Upper Saddle River, New Jersey.

Magrab, E.B., S. Azarm, B. Balachandran, J. Duncan, K. Herold, and G. Walsh, 2000, "*An Engineer's Guide to MATLAB®*", Prentice Hall, Upper Saddle River, N.J., U.S.A.

McCuen, R.H., 1989,"*Hydrologic Analysis and Design - second edition*," Prentice Hall, Upper Saddle River, New Jersey.

Middleton, G.V., 2000, "*Data Analysis in the Earth Sciences Using Matlab®,*" Prentice Hall, Upper Saddle River, New Jersey.

Montgomery, D.C., G.C. Runger, and N.F. Hubele, 1998, "*Engineering Statistics*," John Wiley & Sons, Inc.

Newland, D.E., 1993, "*An Introduction to Random Vibrations, Spectral & Wavelet Analysis - Third Edition*," Longman Scientific and Technical, New York.

Nicols, G., 1995, "*Introduction to Nonlinear Science*," Cambridge University Press, Cambridge CB2 2RU, U.K.

Parker, T.S. and L.O. Chua, , "*Practical Numerical Algorithms for Chaotic Systems*," 1989, Springer-Verlag, New York.

Peitgen, H-O. and D. Saupe (editors), 1988, "*The Science of Fractal Images*," Springer-Verlag, New York.

Peitgen, H-O., H. Jürgens, and D. Saupe, 1992, "*Chaos and Fractals - New Frontiers of Science*," Springer-Verlag, New York.

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, 1989, "*Numerical Recipes - The Art of Scientific Computing (FORTRAN version)*," Cambridge University Press, Cambridge CB2 2RU, U.K.

Raghunath, H.M., 1985, "*Hydrology - Principles, Analysis and Design*," Wiley Eastern Limited, New Delhi, India.

Recktenwald, G., 2000, "*Numerical Methods with Matlab - Implementation and Application*," Prentice Hall, Upper Saddle River, N.J., U.S.A.

Rothenberg, R.I., 1991, "*Probability and Statistics*," Harcourt Brace Jovanovich College Outline Series, Harcourt Brace Jovanovich, Publishers, San Diego, CA.

Sagan, H., 1961,"*Boundary and Eigenvalue Problems in Mathematical Physics*," Dover Publications, Inc., New York.

Spanos, A., 1999,"*Probability Theory and Statistical Inference - Econometric Modeling with Observational Data*," Cambridge University Press, Cambridge CB2 2RU, U.K.

Spiegel, M. R., 1971 (second printing, 1999), "*Schaum's Outline of Theory and Problems of Advanced Mathematics for Engineers and Scientists*," Schaum's Outline Series, McGraw-Hill, New York.

Tanis, E.A., 1987, "*Statistics II - Estimation and Tests of Hypotheses*," Harcourt Brace Jovanovich College Outline Series, Harcourt Brace Jovanovich, Publishers, Fort Worth, TX.

Tinker, M. and R. Lambourne, 2000, "*Further Mathematics for the Physical Sciences*," John Wiley & Sons, LTD., Chichester, U.K.

Tolstov, G.P., 1962, "*Fourier Series*," (Translated from the Russian by R. A. Silverman), Dover Publications, New York.

Tveito, A. and R. Winther, 1998, "*Introduction to Partial Differential Equations - A Computational Approach*," Texts in Applied Mathematics 29, Springer, New York.

Urroz, G., 2000, "*Science and Engineering Mathematics with the HP 49 G - Volumes I & II*", www.greatunpublished.com, Charleston, S.C.

Urroz, G., 2001, "*Applied Engineering Mathematics with Maple*", www.greatunpublished.com, Charleston, S.C.

Winnick, J., , "*Chemical Engineering Thermodynamics - An Introduction to Thermodynamics for Undergraduate Engineering Students*," John Wiley & Sons, Inc., New York.

 © 2001 Gilberto E. Urroz