Université de Nice Sophia-Antipolis
Master 1 MathMods Année 2008/2009

Some notes on *SCILAB*

STEP 1
Get starting

To run *Scilab* on Unix/Linux O.S. type in the window the word:
scilab

You get a *Scilab* window on your computer screen. This *Scilab* window has a menu (File, Control, Graphic Window, Help).

1.1 Manipulating variables, constants

As programming language *Scilab* owns variables, constants, vectors.

1.1.1 Scalars

Few examples are given
$----> x = 2. + sqrt(5)$
$----> x = 3.e-2$
Usual arithmetic operations are valable with usual priority $+, -, *, /, **$.

1.1.2 Constants

Scilab possesses predefined variables or protected variables known as *constants*. They cannot be changed. There are given in table 1.

Constant	meaning
%pi	$\pi = 3.14...$
%e	$e = 2.73...$
%i	complexe number i s.t. $i^2 = -1$
%eps	machine epsilon
%inf	$+\infty$
%nan	"Not A Number"

Table 1: *Scilab* constants.

1.1.3 String variables

Examples of chains of caracters known as *string* variables are given.
$----> s =' subject', v =' verb', c ='' complement''$

Some operations on string variables are shown in table 2.

operations	meaning
$+$	concatenation
strcat	concatenation
strindex	caracter research
strsubst	caracter substitution
length	number of caracters in a chain

Table 2: Some operations on string variables.

1.1.4 Logical variables

Logical variable or boolean variable corresponds to a logical expression. A logical variable can only take two values: $\%t$ for *true* and $\%f$ for *false*. Basic operations on logical variables are given in table 3.

operations	meaning
˜	negation
|	or
&	and

Table 3: Operations on logical variables.

Comparison operations between boolean variables are displayed in table 4.

operations	meaning
$==$	equals to
$<>$ or ˜$=$	is not equal to
$<$	lower than
$>$	greater than
$\leq$	lower or equal to
$\geq$	greater or equal to

Table 4: Logical variables comparison operations.

1.2 Manipulating matrices

A matrix is a set of variable types defined in previous sec. 1.1. However, everything in *Scilab* is a matrix.

Here are are some examples
$- - -- > A = [1\ \ 2\ \ 3; 4\ \ 5\ \ 6]; B = [7,\ \ 8,\ \ 9]; C = [\ \ ];$
$- - -- > D = [cos(1)\ \ \%e\ ; sin(\%pi/2)\ \ \%i]$
A matrix can be defined explicitly by enumerating its elements:

- rows are separated by "**;**"
- columns are separated by "**,**"

Operations on matrices format are shown in table 5.

operations	meaning
A(i,j)	element of A in entry i,j
A(i,:)	row i of A
A(:,j)	column j of A
$A(i_1 : i_2, j_1 : j_2)$	matrix extracted from A from rows i_1 to row i_2
	and from colomn j_1 to column j_2
size(A, 1)	rows number of matrix A
size(A, 2)	columns number of matrix A
size(A)	rows number of matrix A, columns number of matrix A
length(A)	total number of elements of matrix A

Table 5: Matrix format operations.

Usual Operations on matrices $+, -, *$ are valable. In addition *Scilab* allows another operations known as *element by element operations*. They are displayed in table 6.

operations	meaning
A.*B	$(a_{ij} * b_{ij})$
A./B	(a_{ij}/b_{ij})
A.ˆB	$(a_{ij}^{b_{ij}})$
f(A)	$f(a_{ij})$, f being a defined or predefined function

Table 6: Element by element matrix operations.

As example type following instructions:
$- - -- > A = [1\ \ 2\ \ 3; 4\ \ 5\ \ 6];\ B = exp(A);$
There are also predefined matrices given in table 7 and left to the reader to complete meanings her(him)self (n,m are integers, A a matrix, u a scalar).

Sparse matrices
Sparse matrices can be easily performed on *Scilab*. The keyword is *Sparse*.
As example one can execute following instructions
$- - -- > A = [2\ \ -1\ \ -1\ \ 0\ \ 5\ ; -1\ \ 2\ \ -1\ \ 0\ \ 0\ ; 0\ \ -1\ \ 2\ \ -1\ \ 0\ ; 0\ \ 0\ \ -1\ \ 2\ \ -1\ ; 0\ \ 0\ \ 3\ \ -1\ \ 2];$
$- - -- > SparseA = sparse(A)$
$- - -- > size(SparseA)$
$- - -- > u = [1\ \ 1\ \ 1\ \ 1\ \ 1]'$
$- - -- > SparseA * u$

operations	meaning
zeros(n,m)	
ones(n,m)	
eyes(n,m)	
diag(A)	
diag(u)	
diag(u,i)	
tril(u)	
triu(u)	

Table 7: Some predefined matrix operations.

In pratical applications, one is faced to large sparse matrix which cannot be stored entirely in *Scilab* memory. However, knowning the location of nonvanishing elements of the sparse matrix under consideration, one can reperesent this matrix. The procedure is the following:

- One constructs a vector u containing nonvanishing elements of the matrix;
- A vector ii of integer containing the rows entries of nonvanishing elements of the matrix is considered;
- One constructs a vector jj of integer containing the columns entries of nonvanishing elements of the matrix is considered, according to the vector ii.

Next an example is considered.
$----> ii = [1\ \ 2\ \ 3\ \ 4\ \ 1\ \ :\ \ 4];$
$----> jj = [2\ \ 1\ \ 4\ \ 3\ \ 1\ \ :\ \ 4];$
$----> u = [-1\ \ -1\ \ -1\ \ -1\ \ 2\ \ 2\ \ 2\ \ 2];$
$----> pos = [ii; jj]';$
$----> spx = sparse(pos, u)$
$----> full(spx)$

Vectors

A vector is a particular matrix having n rows and 1 column.
For example
$----> v = [1\ \ 2\ \ 7\ \ 4\ \ 1\ \ 20]$
$----> u = 5\ :2\ :12$
$----> w = 0\ :10$
Following useful commands on vectors are shown in table 8.

Some examples are now given.
$----> linspace(0.2\ ,2\ ,5)$
$----> logspace(0.1\ ,4\ ,10)$
$----> logspace(1\ ,\%pi\ ,10)$

operations	meaning
linspace(a,b,n)	vector of size n whose components are equidistant
logspace(a,b,n)	vector of size n whose logarithm of components are equidistant

Table 8: Equidistant vector operations.

STEP 2
Graphics plotting

2.1 Curves

Curves are plotted in *Scilab* by using the command *plot*.

Here are examples.

$----> x = linspace(1\ ,15)\ \ ,y = cos(x)\ \ ;plot(x,y)$

$----> x = linspace(1\ ,10)'$

$----> plot(x,\ cos(x),\ 'b*-',\ x,\ sin(x),\ 'ro-',\ x,\ cos(x).*sin(x),\ 'g+-')$

2.2 Surfaces, level sets

Following commands are left to readers to be fimiliar with:

plot3d, contour, contour2d

2.3 Save Graphics

Table 9 gives commands regarding how to save and load a graphic.

commands	meaning
xsave	save a graphic in a window
xload	load a file containing a graphic in a window

Table 9: Save graphics and load files containing graphics commands.

2.4 Graphics managing

Useful commands enabling one to manage graphics context are given in table 10 and left to the reader to check their utilities.

commands	meaning
clf	
xbasc	
xset	
xget	

Table 10: Graphics managing commands.

STEP 3
Programming in *Scilab*

3.1 Conditional instructions

3.1.1 The 'If' instruction

The syntax is as follows

If condition **then**

instructions

end

or

If condition **then**
instruction1
else
instruction2
end

3.1.2 The 'Select' instruction

The syntax is as follows

select expression

case expression1 **then**
instructions1

case expressionn **then**
instructionsn

end

or

select expression
case expression1 **then**
instructions1
case expressionn **then**
instructionsn
else
instructions
end

3.2 Iterative instructions

3.2.1 The 'for' loop

The syntax of the *for* loop is as follows

for var=begin : step : end

instructions

end

3.2.2 The 'while' loop

The syntax of the *while* loop is as follows
while condition **do**

instructions

end

3.3 Functions, scripts

A function can be defined either in the calling program (in-line function), or in other file distinct from the one of the calling program.

3.3.1 Function in-line

The keyword is *deff*.
Here is an example
$----> deff('[plus, minus] = pm(a, b)', ['plus = a + b', 'minus = a - b'])$
$----> pm(2, 4)$

3.3.2 Function defined in a file

The syntax of a function written in a file is as follows

function [output arguments] = functionname(input arguments)

instructions

endfunction

The above in-line function example is rewritten as a defined function in a file

function [plus, minus] = pmf(a, b)

plus = a+b
minus = a-b

endfunction

3.3.3 Scripts

To save typing the same *Scilab* instructions, on can write one of all these instructions in a file, known as a *script*.

STEP 4
Applications

One would like to solve with Jacobi and Gauss-Seidel iterative methods the linear system $Ax = b$ where $b \in \mathbb{R}^n$ and A is the following n-order square matrix:

$$A = \begin{pmatrix} 2 & -1 & & & & & & & \\ -1 & 2 & -1 & & & & & & \\ & -1 & 2 & -1 & & 0 & & & \\ & & -1 & 2 & -1 & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & -1 & 2 & -1 & & \\ & & 0 & & & -1 & 2 & -1 & \\ & & & & & & -1 & 2 & -1 \\ & & & & & & & -1 & 2 \end{pmatrix}.$$

a. The approximate solution sequence $(x^k)_{k\geq 0}$ given by the Jacobi method is recalled

$$\begin{cases} x_i^{k+1} = \dfrac{b_i - \displaystyle\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j^k}{a_{ii}}, \quad \text{for } \ k \geq 0\,, \text{for } \ i = 1\,, \ldots\,, n\,, \\ x^0 \ \text{ given}\,. \end{cases} \tag{4.1}$$

Write a program returning the iterative solution given by the Jacobi method.

b. The approximate solution sequence $(x^k)_{k\geq 0}$ generated by the Gauss-Seidel method is as follows

$$\begin{cases} x_i^{k+1} = \dfrac{b_i - \displaystyle\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k}{a_{ii}}, \quad \text{for } \ k \geq 0\,, \text{for } \ i = 1\,, \ldots\,, n\,, \\ x^0 \ \text{ given}\,. \end{cases} \tag{4.2}$$

Write a program returning the iterative solution produced by the Gauss-Seidel method (left to the reader).