Scilab 4.1.2
Part I: Introduction

Gianluca Antonelli **Stefano Chiaverini**

Università degli Studi di Cassino

{antonelli,chiaverini}@unicas.it

http://webuser.unicas.it/antonelli

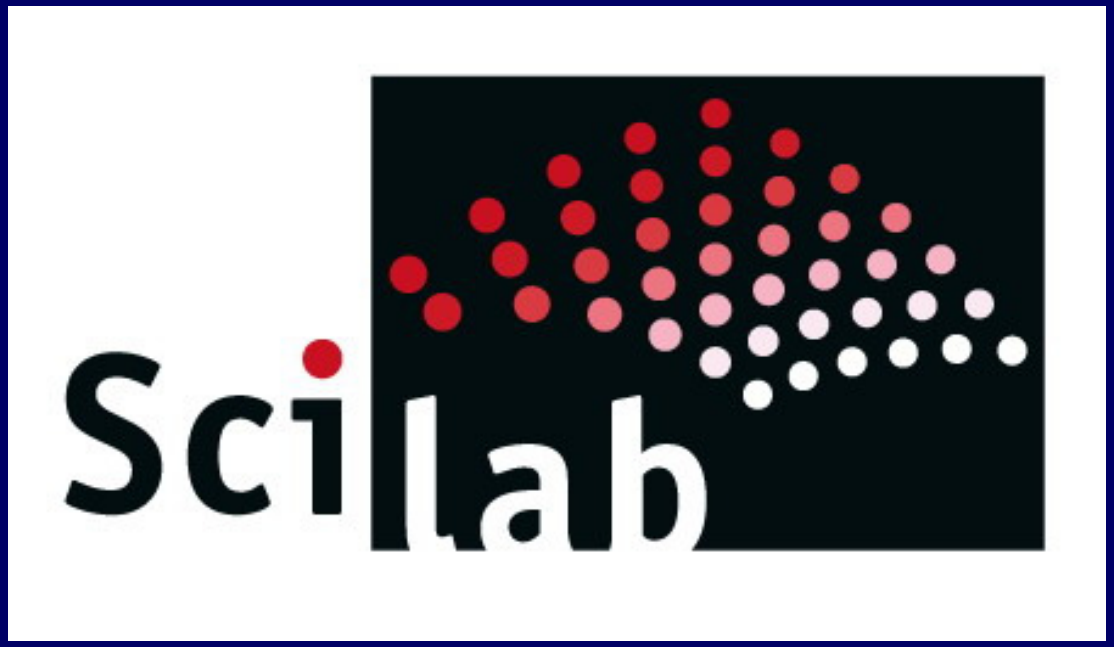
License

Scilab 4.1.2

- Part I: Introduction
- Part II: 2D Graphics
- Part III: Systems and Control
- Part IV: Scicos

Introduction

The open source platform for numerical computation



Available at `http://www.scilab.org`

The others

- Matlab
- Octave
- Freemat
- . . .

Running Scilab

- From Linux or Windows just selecting the application from the Graphical Menu
- From Linux also from the shell typing `./scilab` from the correct path

Scilab command line

- Scilab command line is denoted with the symbol

```
-->
```

- It accepts variables declaration, expressions, script and function calls
- Scilab scripts and functions are ASCII files
- To re-use a previous command type the up arrow ↑

Help

- `help` opens a new window where browse for commands and search keywords
- `help name_fun` opens the help window directly at the page corresponding to the function `name_fun`
- An on-line help is available

  `http://www.scilab.org/product/man/`
- A comparison between Matlab/Scilab function in

  `http://www.scilab.org/product/dic-mat-sci/M2SCI_do`

Variables

- A variable can be defined simply with the syntax

```
-->name = instruction;
```

- The `;` symbol prevents to print on screen the results
- Standard rules common to languages as C or Pascal apply for Scilab variables
- Names are case sensitive: `a`$\neq$`A`
- To see the variable value just type its name on the prompt or use the tool `browsevar()`

Workspace

- Variables and functions stay in the workspace
- commands of general use are

```
who
who_user
clear
load
save
diary
browsevar()
```

Variable types

- A specific set of operators is available for the corresponding type
- Common types are

```
constant, polynomial, function,
handle, string, boolean,
list, rational, state-space,
sparse, boolean sparse
```

- The type of a variable can be obtained

```
typeof(variable_name)
```

Write protected variables

- Certain variables are predefined and write-protected

`%i`	$i = \sqrt{-1}$	immaginary unit
`%pi`	$\pi = 3.1415927\ldots$	pi grek
`%e`	$e = 2.718281\ldots$	number of Nepero
`%eps`	$\varepsilon = 2.2 \cdot 10^{-16}$	precision (machine dependent)
`%inf`		infinity
`%nan`		NotANumber
`%s`	s	polynomial variable
`%z`	z	polynomial variable
`%t`	true	boolean variable
`%f`	false	boolean variable

Matrices

- To insert a matrix the syntaxes are

```
-->A = [1, 2, 3; 4, 5, 6; 7, 8, 9];


-->A = [1, 2, 3
-->4 5 6
-->7,8,9]
 A   =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

Scalars and vectors

- Scalars and vectors are matrices!

```
-->a=[1 3 5];

-->size(a)
 ans  =

    1.    3.

-->b=3;size(b)
 ans  =

    1.    1.
```

Incremental vectors

```
-->x=1:4
 x   =
    1.    2.    3.    4.

-->x=1:2:10
 x   =
    1.    3.    5.    7.    9.
```

Also available `linspace(x0,xf,npti)` and `logspace(...)`

Accessing matrix elements

- `A(i,j)` access the element in row `i` and column `j`
- `A(2,4:6)` select the columns from 4 to 6 of the second row
- `B=A(1:3,4:6)` assign to `B` the selected submatrix
- `A(:,5)` select the column 5
- `A([1 3],5)` select a `1x2` vector of two elements: `A(1,5)` and `A(3,5)`

Matrix operations

- A wide number of operations are available, in addition to the basic operations such as sum, product, transpose, matrix esponential, inverse, rank, kernel, etc.
- A library of operations are available under the Linear Algebra section of the help
- Some operations are also defined to be performed on the single elements of the matrix

Polynomial operations

- Defining a variable as polynomial allows to access several specific operations

```
-->my_pol = 3*%s^2 + 2*%s
 my_pol  =
             2
    2s + 3s

-->roots(my_pol)
 ans  =
    0
  - 0.6666667
```

Polynomial operations

- Define a polynomial from the coefficient: `poly`
- Define a polynomial from its expression (previous example)
- Define a polynomial by operations on elementary polynomials
- Extract the coefficients of a polynomial: `coeff`
- Evaluate the polynomial in one single point: `horner`
- Symbolic substitution of the polynomial variable with another

Rational operations

- Rational functions are the division between two polynomials
- Several commands defined to their use similar to the polynomial type

Script

- It is a collection of commands saved in a single ASCII file whose extension is conventionally `.sce`
- It is executed typing

```
-->exec('file_name.sce');
```

Functions

- A function is a piece of code returning an output given several inputs whose syntax is

```
function [y1,...,ym] = fun(x1,...,xn)
   commands
endfunction
```

- Several functions can be saved in a single ASCII file whose extension is conventionally `.sci`

Functions

- User-defined functions are not available until not explicitely called by `getf('name_file.sci')`
- Loading, saving or clearing the variables causes Scilab to do the same to the functions!
- Functions see all the workspace
- Inputs are passed by reference if the function does not change their value otherwise by copy

Programming

- Common programming constructions
  - `for, end`
  - `while, end`
  - `if, then, else, end`
  - `select, case, else, end`
- Logical operators: `&` `|` `~`
  - `&` and
  - `|` or
  - `~` not

Load/Save data

- Data in the workspace can be saved in a binary file:

```
save('name_file')
```

- Data previously saved can be loaded in the workspace:

```
load('name_file')
```

- C and Fortran-like commands are available to save formatted data in ASCII files
- Data saved with older version of Matlab can also be loaded