Introduction to Scilab

Aditya Sengupta

Indian Institute of Technology Bombay
apsengupta@iitb.ac.in

April 15th, 2010- Smt. Indira Gandhi College of Engineering

Outline

1. Introduction
2. Scilab Objects: Matrices and Polynomials.
3. Basic Programming
4. Basic Input And Output
5. Basic Graphics

About Scilab

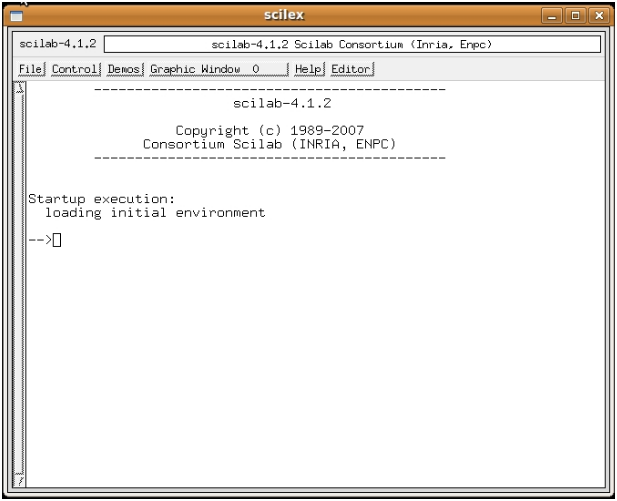
- Around since 1990
- Numerical Computational package
- Free and Open Source
- Maintained by INRIA

About Scilab

- Inspired by Cleve Molers MATLAB
- Interpreted
- Very High Level

  $Scilab : C = C : Assembly$
- Available for Linux, Mac and Windows

Scilab Window looks like



```
        -------------------------------------------
                         scilab-4.1.2

                   Copyright (c) 1989-2007
               Consortium Scilab (INRIA, ENPC)
        -------------------------------------------


Startup execution:
  loading initial environment

-->
```

Try This Stuff

```
→ 36 + 4^2 - 20/2
→ a=1, b=2, c=3
→ a + b + c
→ institute = ‘‘IITB’’;
→ typeof(institute)
→ clear(‘‘institute’’)
→ exists(‘‘institute’’)
```

Try This Stuff

→ `1/0`

→ `ieee(2)`

→ `1/0`

→ `%e`

→ `sin(%pi/2), cos(%pi/2)`

→ `(10+5*%i)*(2*%i)`

→ `2*cos(%pi/5)`

About Scilab

- Everything is a matrix!
- Even a real scalar is a 1×1 matrix
- You can define numbers, character strings, booleans, polynomials and lists

Try This Stuff

- → `a=[1 2 3], b=[2 3 4]`
- → `a’`
- → `a*b`
- → `a.*b`
- → `a’*b`
- → `a*b’`
- → `size(a)`
- → `length(a)`
- → `diag(a)`

Try This Stuff

→ `A=[1 2; 0 4], B=[1 2; 3 4]`

→ `A+B, A-B`

→ `A*B, B*A, A.*B, B.*A`

→ `det(A)`

→ `inv(A)`

→ `size(A)`

→ `length(A)`

→ `diag(A)`

Try This Stuff

```
→ A=1:4   //This is a comment
→ B=2:2:8   // range
→ B([3 4])   // submatrix extraction
→ A(6)=6   // add an element
    // oops!  Forgot the fifth element!
→ A($-1)=5   // ‘‘ $’’ is last element
→ B=2*A   // reassignment
→ B=[B 2*B; 3*B]   // new rows
→ B($+1,:)=4*B(1,:)
→ B([2 3], 2:$-1)   //submatrix extraction
```

Try This Stuff

```
→ pwd
→ help pwd
→ cd(‘‘path-to-directory’’)
→ diary(‘‘ my-record-of-what-follows.sci’’)
→ help("diary")
→ inv([1 2; 0 4])
→ C=rand(3,3)
→ C>.5   // boolean matrix
→ find(C>.5)
→ C(find(C>.5))
→ disp(C)
```

Try This Stuff

→ P=poly([2 3 1], ‘x’, ‘coeff’)

→ Q=poly([-1 4], ‘x’)

→ P*Q, P+Q, P-Q

→ roots(P), roots(Q), roots(P*Q)

→ factors(P), factors(Q), factors(P*Q)

→ 1/P

→ Q/P

→ derivat(P), derivat(Q), derivat(Q/P)

→ horner(P, 0), horner(P, [0 1 2]) //to evaluate at a value or a set of values

Conventions

- Commands may be put in scripts.
- Extension is `.sce`
- If it only contains function definitions, the extension is `.sci`
- These are conventions!
- Execute:`exec('path-to-script/script-name.sce')`

Functions

```
function [y1, y2, ...]=foo(x1, x2, ...)
     statement
     statement
     statement
endfunction
           OR
deff(‘‘[y]=foo([x])’’, ‘‘statements’’)
```

Functions

- If function definitions are in a script file, use `getf('path/script.sci')`
- To see the source of a Scilab coded function use `fun2string(function-name)`

Branching

```
if condition then
       statement
       statement
       statement
else
       statement
       statement
       statement
end
```

Iterations

```
for name = expression
     statement
     statement
     statement
end
// Use break to stop execution within statement block
```

Iterations

```
while condition
   statement
   statement
   statement
// Use break to stop execution within statement block
```

Try This Stuff

```
function y = myfactorial(x)
   if x==0 then y=1
   else y = x*myfactorial(x-1)
   end
endfunction
```

Try This Stuff

- → `// try a few examples:`
- → `myfactorial(5), myfactorial(0)`
- → `// now try Scilabs own function:`
- → `factorial(5), factorial(0)`

Input

→ `name=input(''Enter your name:  '')`

→ // oops (try entering your name in “ ”)

→ or try this:

→ `name=input(''Enter your name:  '', ''string'')`

→ `disp(name);`

→ more comfortable with C? try this:

→ `mprintf(''Your name is %s'', name)`

[Optional] Look these up in help:

→ `mopen`

→ `mprintf`

→ `mfprintf`

→ `mscanf`

→ `mfscanf`

→ `mclose`

plot2d

```
→ x=linspace(-%pi, %pi, 40)
→ plot2d(x, sin(x))
→ //Try getting the axes in the centre
→ //Don't like the continuous version?
→ plot2d3(x,sin(x))
```

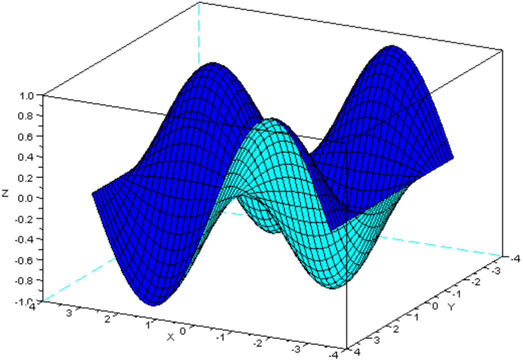
plot3d

→ y=x

→ plot3d(x, y, sin(x)'*cos(y))

— Notice the transpose

z=sin(x)'*cos(y)



Thank You!

- www.scilab.org
- www.scilab.in
- http://scilab.in/cgi-bin/mailman/listinfo/scilab-india
- “Modeling and Simulation in Scilab/Scicos” by Stephen L.Campbell, Jean-Philippe Chancelier and Ramine Nikoukah, (Springer)