

Scilab Download



1. Ubuntu :

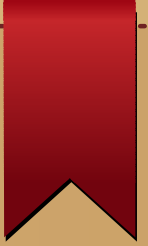
```
sudo apt-get update  
sudo apt-get install scilab
```

2. Windows / Mac

Go to this link : <https://www.scilab.org/download/latest> and click on the appropriate link for your system.



Scilab



- Scilab is a free and open source software for numerical computation.
- It has high-level programming language features including data structures. Some functionalities are:
 1. Math and Simulation
 2. 2D-3D visualization
 3. Optimization
 4. Statistics
 5. Signal processing
 6. Application development



General Commands

- Scilab console, editor (scinotes) – used for executing scilab commands
- .sce and .sci files – script files
- .sci files – contain scilab and/or user defined functions, and executing these loads functions to scilab environment but does not execute them
- .sce files – contain both functions and executable statements
- Trigonometric operations : $\sin()$, $\cos()$ which take arguments in radians
- $\text{Log10}()$, $\text{exp}()$, $\text{factorial}()$, $\text{sqrt}()$, $\text{abs}()$
- %e, %pi are a few constants
- Complex numbers can also be represented using %i .

Eg : $10+5*\%i \rightarrow 10+5i$

General Commands

- `diary("filename.txt")` function creates a log of keyboard input and the resulting text output in a file in the present working directory.
- `diary(0)` saves the log onto the file created and closes it.
- All transactions before `diary(0)` was executed will be saved in the file.
- Note: If same file name is used again , the file will be overwritten.
- `help <command name>` is used to get detailed help regarding a particular command.
- `chdir` is used to change scilab current directory
- `SCI` – variable containing root path to scilab
- `pwd` – prints current working directory
- `SCIHOME` - contains the path to preferences, history files of your Scilab session.

Scilab Functions

Syntax:

```
--> function d = dollars(e,t);
```

```
--> d=e*t;
```

```
--> endfunction;
```

```
--> dollars(2,3);
```

```
disp("Hello")
```

```
disp(string(5)+"is a number")
```

```
disp(A) where A can be a vector or a matrix
```

Every vector is a $1 \times n$ or $n \times 1$ matrix, and a number is a 1×1 matrix in Scilab.

Conditional Statements

The basic conditional statement in Scilab is the following:

```
if...then...elseif....then....else..end
```

```
if expr1 then statements
elseif expri then statements
....
else statements
end
```

Example:

```
i=5
j=6
if (i==j) then
disp("i is equal to j")
elseif(i<j) then
disp("i is less than j")
else
disp("i is greater than j")
```

Iterative Statements

Loops

3:10 (print 3 to 10)

1:2:10 (increment in steps of 2, from 1 to 10)

20:-4:2 (decrement in steps of 4 from 20 to 2)

```
u(1)=2;  
for n=1:5  
    u(n+1)=u(n)+2;  
    disp(u(n));  
end;  
disp("The number is " + string(u(2)));
```

Comparisons (<,>,<=,>=,<>,==)

```
X=[1,2,3];  
Y=[3,2,4];  
X==Y;  
isequal(X,Y);
```

Other useful functions

The **grand()** function :

`grand(1,p,"uin",m,n)` returns a vector of p random integer sequences between m and n.

```
-->t= grand(1,4,"uin",1,6)
```

```
t
```

```
=
```

```
3. 1. 3. 6.
```

`grand(1,p,"unf",a,b)` returns a vector of p random real sequences between m and n.

```
-->tr= grand(1,2,"unf",-1,1)
```

```
tr
```

```
=
```

```
- 0.7460264  0.9377355
```


Other useful functions (Contd..)

Suppose there is a vector $D=[3,3,3]$, and a Matrix $M = [1,2,3;4,5,6]$

The **unique()** function : returns a vector which contains all the distinct elements of D only once.

```
--->unique(D)
ans
=
3.
```

The **length()** function: returns the total number of elements in a matrix or vector.

```
--->length(D)
ans
=
3.
```

The **size()** function: returns the dimensions (rows,columns) of the matrix m.

```
---> size(M)
ans
=
2.
3.
```

Other useful functions (Contd..)

The **sum()** and **prod()** functions return the sum and product of the elements of the argument vector.

```
-->U=[1:10];
```

```
-->sum(U)
```

```
ans
```

```
=
```

```
55.
```

```
-->prod(U)
```

```
ans
```

```
=
```

```
3628800.
```

Vectors and Matrices

Square brackets are used to define matrices and vectors

- --> $p = [1 \ 2 \ 3]$ or $[1, 2, 3]$ row vector
- --> $q = [1; 2; 3]$ column vector
- $p+q$, $p-q$ perform addition, subtraction of vectors p , q respectively.

Matrices

```
-->m=[1 2 3;4 5 6]
```

```
m
```

```
=
```

```
1. 2. 3.
```

```
4. 5. 6.
```

Parentheses are used to access or modify elements.

```
-->m(2,3)
```

```
ans
```

```
=
```

```
6.
```

```
-->m(2,3)=23
```

```
m
```

```
=
```

```
1. 2. 3.
```

```
4. 5. 23
```

$\det(A)$, $\text{inv}(A)$, $\text{spec}(A)$ are used to compute determinant, inverse and eigen values of Matrix A.

Matrix Operations

- To view the second row of the matrix m

-->m(2,:)

ans

=

4.

5.

23.

- To view the 3rd column of matrix m

-->m(:,3)

ans

=

3.

23.

- m(:,2:3) returns only 2nd to 3rd column of all rows in m, m(:, \$) - last column of all rows

- Transpose of m

-->m'

ans

=

1. 4.

2. 5.

3. 23.

Matrix Operations

The operations “*” and “/” are used for matrix operations.

For performing element-wise operations, “.*” and “./” are used.

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

```
A
```

```
=
```

```
1. 2. 3.
```

```
4. 5. 6.
```

```
-->B=[1;1;2]
```

```
B
```

```
=
```

```
1.
```

```
1.
```

```
2.
```

```
-->A*B
```

```
ans
```

```
=
```

```
9.
```

```
21.
```

Matrix multiplication

Matrix Operations

```
-->A*A
!-error 10
Inconsistent multiplication.
-->A.*A
ans
=
1. 4. 9.
16. 25. 36.
```

Dimensions are not consistent

Element-wise multiplication

```
-->2*(A+2)
ans
=
6. 8. 10.
12. 14. 16.
```

Element-wise product as 2 is a number

```
-->A/A
ans
=
1. 1.518D-16
3.795D-15 1.
```

Gives the matrix X for which $X*A=A$

Matrix Operations

```
-->A./A
```

Performs element wise division

```
ans
```

```
=
```

```
1. 1. 1.
```

```
1. 1. 1.
```

Solving linear systems :

To solve the linear system $AX=Y$, in which A is a square matrix, use the backslash “\” operator.
 $X=A\backslash Y$.

Be cautious, the operation Y/A will give (only if the dimensions are correct) another result, that is to say the matrix Z for which $ZA=Y$.

```
-->A=[1 2 3;4 5 6];
```

```
-->Y=[1;1];
```

```
-->X=A\Y
```

```
X
```

```
=
```

```
- 0.5
```

```
0.
```

```
0.5
```

```
-->A*X
```

```
ans
```

```
=
```

```
1.
```

```
1.
```


Other useful functions (Contd..)

The **gsort()** function is used to perform sorting in Scilab. It uses the standard quicksort algorithm for the same.

```
gsort(A)  
gsort(A,option)  
gsort(A,option,direction)
```

A : a real,an integer or a character string vector/matrix or a sparse vector.

option : a character string. It gives the type of sort to perform:

 'r' : each column of A is sorted

 'c': each row of A is sorted

 'g': all elements of A are sorted. It is the default value.

 'lr': lexicographic sort of the rows of A

 'lc': lexicographic sort of the columns of A

direction : a character string. It gives the ordering direction: 'i' stand for increasing and 'd' for decreasing order (default).

Question

Alice throws three dice.

1. If she gets three 6's she wins \$20.
 2. If she gets three identical numbers different from 6 she wins \$10.
 3. If she gets two identical numbers she wins \$5.
- Otherwise she wins nothing.

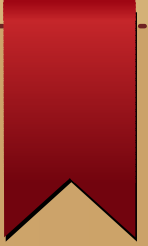
Calculate Alice's winning using the 3 functions:

1. `grand`
2. `unique(D)`
3. `length(unique(D))`, where `D` is a vector.

Answer

```
D = grand(1,3,"uin",1,6);  
If D == [6,6,6] then  
W=20;  
Elseif length(unique(D)) == 1 then  
W=10;  
Elseif length(unique(D)) == 2 then  
W=5;  
Else  
W=0;  
End;  
disp("alice wins "+string(w)+" dollars");
```

Question



Write a small scilab function to find the inverse of the argument matrix.

(Do not use the standard function to perform the operation)



Answer

```
function y=f(X);  
    [m,n]=size(X);  
    if(m<>n) then  
        disp("Inverse does not exist");  
    else  
        I=eye(m,m)  
        y=X\I  
    end  
endfunction
```

The above function gives the same result as the standard function **inv(A)** which returns the inverse of the square matrix **A** passed as argument if it exists. A warning message is displayed if the matrix is wrongly scaled or singular.

Plotting functions

`Plot(1,2,".r");` ----- plot a point

`Plot([1,3],[2,5]);` ----- plot a line

Plotting points options : ".", "*", "+", "o", "x"

To plot a curve $f(x)=(x^2+2x)e^{-x}$

Function $y = f(x)$;

$y=(x^2+2*x)*\exp(-x)$;

Endfunction;

`x=linspace(-2,5,50)` ----- `linspace(a,b,n)` assigns to x, n values between a and b.

`plot(x,f)`;

Clf command is used to clear a plot. Use it before creating another plot.

To plot 2 curves

Function $y = g(x)$;

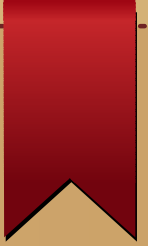
$y=\sin(x/2)$;

Endfunction;


`x=linspace(-2,5,50)`;

`Clf`;

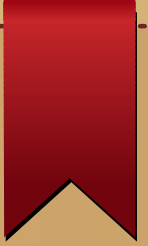
`plot(x,f,"r",x,g,"g")`;



```
To plot a sequence of points  
for n=1:50  
u(n)=(-0.8)^n;  
end  
Clf;  
plot(u,"*r");
```



3D Plotting



To plot the surface $z = 2x^2 + y^2$ (elliptical paraboloid)

```
function z=f(x,y)
```

```
z=2*x^2+y^2;
```

```
endfunction
```

```
x=linspace(-1,1,100);
```

```
y=linspace(-2,2,200);
```

```
z=feval(x,y,f)';
```

```
clf
```

```
surf(x,y,z)
```

`feval(x,y,f)` returns the $m \times n$ matrix whose i,j element is $f(x_i, y_j)$, and this is transposed by using the single quote symbol (`'`). Only a column vector can be passed for the plotting of a graph.

For 2D graphs we use `plot` and for 3D graphs we use `surf`.



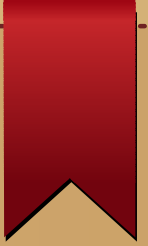
Parametric plot

To plot the helix defined by $(x = \cos t, y = \sin t, z = t)$

```
t=linspace(0,4*pi,100);  
param3d(cos(t),sin(t),t)
```

Curves in space can be plotted using param3d function. It has 3 arguments x,y and z, and they correspond to the respective points on the curve.

Optimization



- Mathematical optimization is the selection of a best element from some set of available alternatives.
- In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.
- More generally, optimization includes finding "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains.



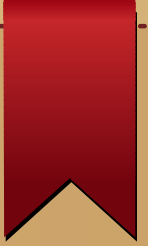
Mathematical Representation

- Optimization can be mathematically represented as :

Given: a function $f : A \rightarrow \mathbb{R}$ from some set A to the real numbers

Sought: an element x in A such that $f(x) \leq f(y)$ for all y in A ("minimization") or such that $f(x) \geq f(y)$ for all y in A ("maximization").


- f is called the objective function.
- A is some subset of the Euclidean space \mathbb{R}^n , often specified by a set of constraints, equalities or inequalities that the members of A have to satisfy.
- The domain A of f is called the search space or the choice set, while the elements of A are called candidate solutions or feasible solutions.
- A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an optimal solution.

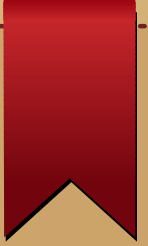
- 
- In mathematics, conventional optimization problems are usually stated in terms of minimization.
 - A local minimum x^* is defined as a point for which there exists some $\delta > 0$ so that for all x such that

$$\|x - x^*\| \leq \delta$$

the expression being

$$f(x^*) \leq f(x)$$

- On some region around x^* all of the function values are greater than or equal to the value at that point.
 - While a local minimum is at least as good as any nearby points, a global minimum is at least as good as every feasible point.
 - In a convex problem, if there is a local minimum that is interior (not on the edge of the set of feasible points), it is also the global minimum, but a nonconvex problem may have more than one local minimum not all of which need be global minima.
- 



Constrained optimization is the process of optimizing an objective function with respect to some variables in the presence of constraints on those variables.

A general constrained minimization problem may be written as follows:

Min $f(x)$ subject to

$g(x) = c_i$, for $i=1$ to n -----> Equality constraints

$h(x) \leq d_i$, for $i=1$ to n -----> Inequality constraints

Multi-objective optimization is when multiple objective functions are optimized simultaneously.

If there is no optimal solution which optimizes all the objective functions, then its called non-trivial multi-objective optimization. This occurs when all the objective functions are conflicting each other.

$$\min (f_1(x), f_2(x), \dots, f_k(x)) \text{ s.t. } x \in X$$


Fgoalattain

Fgoalattain basically optimizes multiple objective functions using Scilab, for the attainment of a goal. The input definition of fgoalattain is :

$x = \text{fgoalattain}(f, x_0, \text{goal}, \text{weight}, A, b, A_{\text{eq}}, b_{\text{eq}}, \text{lb}, \text{ub}, \text{nonlcon})$

F --> set of objective functions

X0 --> The initial value of x

Goal --> **Vector of values that the objectives attempt to attain.** The vector is the same length as the number of objective functions in f.

Weight --> It is a vector to control the relative underattainment or overattainment of the objectives in fgoalattain. It is generally $\text{abs}(\text{goal})$. When weight is positive, fgoalattain attempts to make the objectives less than the goal values. To make the objective functions greater than the goal values, we set weight to be negative rather than positive.

$Ax \leq b$ --> The linear inequality constraint.

$A_{\text{eq}}x = b_{\text{eq}}$ --> The linear equality constraint

$\text{lb} \leq x \leq \text{ub}$ --> The lower and upper bounds for x

Nonlcon --> This mentions the non-linear constraints $c(x) \leq 0$ and $c_{\text{eq}}(x) = 0$.

Fgoalattain (contd..)

The output is in the following format:

`[x,fval,attainfactor,exitflag,output,lambda] = fgoalattain(...)` returns a structure `lambda` whose fields contain the Lagrange multipliers at the solution `x`.

`X` --> The optimized value

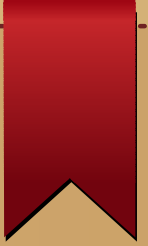
`Fval` --> the set of multi-objective functions

`Attainfactor` --> The amount of over- or underachievement of the goals. If `attainfactor` is negative, the goals have been overachieved; if `attainfactor` is positive, the goals have been underachieved.

`Exitflag` --> Integer identifying the reason the algorithm terminated. Some integers are 1 (function converged to `x`), 0 (no. of iterations exceeded) and -2 (no feasible point found).

`Output` --> It contains information about the optimization, like no. of iterations, no. of function evaluations, etc.

`Lambda` --> Structure containing the Lagrange multipliers at the solution `x`

- 
- For more information about scilab, its toolboxes and mailing lists visit :
 - www.scilab.org
 - www.scilab.in
 - <http://www.scilab.in/scilab-toolbox-help-files/fgoalattain.php>
- 