finaquant

Innovation in Financial Analytics

www.finaquant.com

Quick Start Reference for MATLAB and R

Columns: **MATLAB** | **R**

Useful help commands

MATLAB:
```
% general help
>>help
% general help
>>help
% elementary math functions
>>help elfun
% how while works
>>help while
% search for keywords
>>lookfor optimization
```

R:
```
# open general help document (online)
>help.start()
# get help for a specific function or word
>help('min')   # or
>?min
# search for keywords
>help.search('optimization')   # or
>??optimization
```

Current (working) directory

MATLAB:
```
>>pwd
% You can easily change the working directory by using the directory toolbox in GUI
```

R:
```
>> getwd()
# see menu "File/Change diré  " for changing the working directory
```

Comment symbol

MATLAB:
```
%
```

R:
```
#
```

Assignment

MATLAB:
```
Assignment % Assignment with automatic output to console if there is no semicolon at the end of the statement
>> a = 5
 a =
          5
% Assignment without output to console
>> a = 5;
```

R:
```
# Assignment
> A=5    # or
> A <- 5
> A
[1] 5
# R doesn't display anything in the console automatically even if there is no semicolon at the end of the statement
```

Create vectors

MATLAB:
```
>> v = [1 3 6 9]
v =
     1     3     6     9
% Series: StartValue:Interval:EndValue
>> v = 1:5
v =
     1     2     3     4     5
>> v = 1:2:9
v =
     1     3     5     7     9
>> v = 10:-1:5
v =
    10     9     8     7     6     5
% vector with multiple (repeating) constant values
>> v = ones(1,4) * 2
v =
     2     2     2     2
% horizontal and vertical vectors
>> w = [1 3]'
w =
     1
     3
% For matlab, a horizontal vector is a 1xN, a vertical vector is a Nx1 matrix. There is not a separate entity as "vector" in addition to "matrix".
```

R:
```
>v = c(1, 3, 6, 9)
> v
[1] 1 3 6 9
# Series: seq(StartValue,EndValue,Interval)
>v = 1:5
> v
[1] 1 2 3 4 5
> v = seq(1,9,2)
> v
[1] 1 3 5 7 9
> v = seq(10,5,-1)
> v
[1] 10  9  8  7  6  5
# vector with multiple (repeating) constant values
> v = rep(3,4)
> v
[1] 3 3 3 3
# horizontal and vertical vectors
# R doesn't make a distinction between vertical and horizontal vectors. A "vector" is not a "matrix" for R.
```

Create matrices

MATLAB:
```
% 2x3 matrix with given element values
% ";" as row delimiter
% "," or blank as column delimiter
>> D = [1 2 3; 4 5 6]
D =
     1     2     3
     4     5     6
% Create 2x4 matrix with random elements (values uniformly distributed between 0 and 1)
>>A = rand(2,4)
```

R:
```
# 2x3 matrix with given element values
# fill values row-wise
>D = matrix(c(1,2,3,4,5,6),nrow=2,byrow=TRUE)
>D
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
# fill values column-wise
>D = matrix(c(1,4,2,5,3,6), nrow=2)
# Create 2x4 matrix with random elements (values uniformly distributed between 0 and
A =
    0.8147    0.1270    0.6324    0.2785
    0.9058    0.9134    0.0975    0.5469
% 1x3 matrix with all 3s
>>B = ones(1,3) * 3;
% 3x2 matrix with all 0s
>>C = zeros(3,2);
```

```
1)
>A = matrix(runif(2*4),2,4)
          [,1]      [,2]      [,3]      [,4]
[1,] 0.2180768 0.1598089 0.7320473 0.7478383
[2,] 0.9897023 0.5809282 0.7781437 0.7134574
# 1x3 matrix with all 3s
>B = matrix(3,1,3)
# 3x2 matrix with all 0s
>C = matrix(0,3,2)
```

Show all predefined variables

```
>>who
```

```
>ls()
```

Size of matrix

```
% number of rows and columns
>> [row,col] = size(A);
% number of rows
>> row= size(A,1);
% number of columns
>> col= size(A,2);
% maximum dimension (max(#row, #col))
>>length(A);
>>length([1 2 3]);
% total number of elements in a matrix
>>numel(A)
```

```
# number of rows
>row = nrow(A)
# number of columns
>col = ncol(A)
# maximum dimension (max(#row, #col))
>max(dim(A))
# total number of elements in a vector
>length(v)
# total number of elements in a matrix
>length(A)
# dimensions of a vector
> dim(c(1,2,3))
NULL
```

Access elements of a vector or matrix

```
% element of a vector
>> a = v(2);
% access element of a matrix with row and column index
>>x = M(2,3);
```

```
# element of a vector
> v[2]
# access element of a matrix with row and column indexD
> x = D[2,3]
```

Transpose vectors or matrices

```
% make a horizontal vector (i.e. 1xN matrix) vertical, or vice versa
>> v = [1 2 3]'
v =
     1
     2
     3
% transpose matrix
>> X = [1 2 3; 4 5 6]'
X =
     1     4
     2     5
     3     6
```

```
# Transpose operation converts a vector into a horizontal (1xN) matrix
> t(c(1,2,3))
     [,1] [,2] [,3]
[1,]    1    2    3
# transpose matrix
>D = matrix(c(1,4,2,5,3,6), nrow=2)
> D
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> t(D)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Add column(s) or row(s) to a matrix

```
>> v = [5 10]'
v =
     5
    10
% horizontal (column-wise) concatenation
>> X = [1 2; 3 4];
>> X = [X, v]
X =
     1     2     5
     3     4    10
% vertical (row-wise) concatenation
>>v = [5 10];
>> X = [1 2; 3 4];
>> X = [X; v]
X =
     1     2
     3     4
     5    10
```

```
> v = matrix(c(5,10), nrow=2)
> v
     [,1]
[1,]    5
[2,]   10
# horizontal (column-wise) concatenation
>X = matrix(c(1,2,3,4), nrow=2,byrow=TRUE)
>X = cbind(X,v)
> X
     [,1] [,2] [,3]
[1,]    1    2    5
[2,]    3    4   10
# vertical (row-wise) concatenation
> v = matrix(c(5,10), nrow=1)
>X = matrix(c(1,2,3,4), nrow=2,byrow=TRUE)
>X=rbind(X,v)
> X
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5   10
```

Matrix partitioning (submatrices)

```
>>X = [1 2 3 4; 5 6 7 8]
X =
     1     2     3     4
     5     6     7     8
% single row of a matrix
>> X(1,:)
ans =
     1     2     3     4
% submatrix with specified rows and columns
>>S = X(1:2, 2:4)
S =
     2     3     4
     6     7     8
```

```
>X=matrix(c(1,2,3,4,5,6,7,8),nrow=2,byrow=TRUE)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
# single row of a matrix
> X[1,]  # returns a vector
[1] 1 2 3 4
>X[1,,drop=FALSE]
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
# single column of a matrix
>X[,2,drop=FALSE]
```

```
% submatrix with specified rows and columns
>>S = X([1 2], [2  4])
S =
     2     4
     6     8
```

```
     [,1]
[1,]    2
[2,]    6
# submatrix with specified rows and columns
> S = X[c(1, 2), c(2, 4)]
> S
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

Insert a matrix into another matrix

```
>>X = [1 2 3 4; 5 6 7 8]
X =
     1     2     3     4
     5     6     7     8
>> M = X;
% insert by replacing elements (element assignment)
>>Y = [10 11; 12 13];
>> M(:, [2 3]) = Y
M =
     1    10    11     4
     5    12    13     8
% insert by (horizontal) extension
>>X = [X(:,1), Y, X(:, 2:4)]
X =
     1    10    11     2     3     4
     5    12    13     6     7     8
```

```
>X=matrix(c(1,2,3,4,5,6,7,8),nrow=2,byrow=TRUE)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
>M=X
# insert by replacing elements (element assignment)
>Y=matrix(c(10,11,12,13), nrow=2,byrow=TRUE)
>M[,c(2,3)] = Y
> M
     [,1] [,2] [,3] [,4]
[1,]    1   10   11    4
[2,]    5   12   13    8
# insert by (horizontal) extension
>X = cbind(X[,1], Y, X[,2:4])
> X
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1   10   11    2    3    4
[2,]    5   12   13    6    7    8
```

Basic matrix operations

```
>>X = [2 4 9; 16 25 36];
% square root of all elements
>>C = sqrt(X)
C =
    1.4142    2.0000    3.0000
    4.0000    5.0000    6.0000
% square of all elements
>>C = C .^ 2;
% matrix matrix multiplication
>> v = [1 2 3]'
v =
     1
     2
     3
>>Y = X * v
Y =
    37
   174
% matrix scalar multiplication
>>0.1 * X
ans =
    0.2000    0.4000    0.9000
    1.6000    2.5000    3.6000
% matrix scalar division
>>X / 10
ans =
    0.2000    0.4000    0.9000
    1.6000    2.5000    3.6000
% elementwise multiplication
>> X = [1 2 3 ; 4  5 6] ;
>>X .* X
% elementwise division
>> X ./ X
ans =
     1     1     1
     1     1     1
% inverse matrix
>> X = [1 3; 2 4] ;
>> Z = inv(X)
Z =
   -2.0000    1.5000
    1.0000   -0.5000
>> X * Z
ans =
     1     0
     0     1
```

```
>X = matrix(c(2,4,9,16,25,36), nrow=2,byrow=TRUE)
# square root of all elements
>C = sqrt(X)
> C
         [,1] [,2] [,3]
[1,] 1.414214    2    3
[2,] 4.000000    5    6
# square of all elements
>C = C ^2
# matrix matrix multiplication %*%
>v = matrix(c(1,2,3), nrow=3)
> v
     [,1]
[1,]    1
[2,]    2
[3,]    3
>Y = X %*% v
> Y
     [,1]
[1,]   37
[2,]  174
# matrix scalar multiplication
> 0.1 * X
     [,1] [,2] [,3]
[1,]  0.2  0.4  0.9
[2,]  1.6  2.5  3.6
# matrix scalar division
> X / 10
     [,1] [,2] [,3]
[1,]  0.2  0.4  0.9
[2,]  1.6  2.5  3.6
# elementwise multiplication
> X * X
# elementwise division
> X / X
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
# inverse matrix
>X = matrix(c(1,3,2,4), nrow=2,byrow=TRUE)
>Z = solve(X)
> Z
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> X %*% Z
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Solve a typical matrix equation: A x b = c --> b = ?

```
>>A = [1 2; 3 4];
>>c = [1 1]'
>>b = inv(A) * c
b =
   -1.0000
```

```
>A = matrix(c(1,2,3,4), nrow=2,byrow=TRUE)
>c = matrix(c(1,1), nrow=2)
>b = solve(A) %*% c
> b
     [,1]
```

```
    1.0000
% Test the result
>>A * b
ans =
    1.0000
    1.0000
```

```
[1,]   -1
[2,]    1
# Test the result
> A %*% b
     [,1]
[1,]    1
[2,]    1
```

Sorting vectors

```
>>v = [2 5 1 4];
% sort vector in ascending order
% w: sorted vector
% ind: element indices such that w = v(ind)
[w, ind] = sort(v,'ascend')
w =
     1     2     4     5
ind =
     3     1     4     2
>> v(ind)
ans =
     1     2     4     5
```

```
>v = c(2,5,1,4)
# sort vector in ascending order
# w: sorted vector
# ind: element indices such that w = v(ind)
>res=sort(v,index.return=TRUE)
>w = res$x
>ind = res$ix
> w
[1] 1 2 4 5
> ind
[1] 3 1 4 2
> v[ind]
[1] 1 2 4 5
```

Sorting matrices

```
>> X=[3 5 2 7;1 10 12 5; 3 2 8 10;1 6 4 2]
X =

     3     5     2     7
     1    10    12     5
     3     2     8    10
     1     6     4     2
% sort rows after first column
>> sortrows(X, 1)
ans =
     1    10    12     5
     1     6     4     2
     3     5     2     7
     3     2     8    10
% sort rows after first column in descending order
>> sortrows(X, -1)
ans =
     3     5     2     7
     3     2     8    10
     1    10    12     5
     1     6     4     2
% sort rows after first and second columns
>> sortrows(X, [1 2])
ans =
     1     6     4     2
     1    10    12     5
     3     2     8    10
     3     5     2     7
% sort columns after the first row; sort rows of the transposed matrix
>> sortrows(X',1)'
ans =
     2     3     5     7
    12     1    10     5
     8     3     2    10
     4     1     6     2
```

```
> X = matrix(c(3,5,2,7,1,10,12,5,3,2,8,
         10,1,6,4,2),nrow=4,byrow=TRUE)
> X
     [,1] [,2] [,3] [,4]
[1,]    3    5    2    7
[2,]    1   10   12    5
[3,]    3    2    8   10
[4,]    1    6    4    2
# sort rows after first column
> X[order(X[,1]),]
     [,1] [,2] [,3] [,4]
[1,]    1   10   12    5
[2,]    1    6    4    2
[3,]    3    5    2    7
[4,]    3    2    8   10
# sort rows after first column in descending order
> X[order(-X[,1]),]
     [,1] [,2] [,3] [,4]
[1,]    3    5    2    7
[2,]    3    2    8   10
[3,]    1   10   12    5
[4,]    1    6    4    2
# sort rows after first and second columns
> X[order(X[,1],X[,2]),]
     [,1] [,2] [,3] [,4]
[1,]    1    6    4    2
[2,]    1   10   12    5
[3,]    3    2    8   10
[4,]    3    5    2    7
# sort columns after the first row
> X[,order(X[1,])]
     [,1] [,2] [,3] [,4]
[1,]    2    3    5    7
[2,]   12    1   10    5
[3,]    8    3    2   10
[4,]    4    1    6    2
```

Aggregation functions like sum, mean, max, min, stdev, variance

```
>> X = [1 4 2; 3 6 2]
X =
     1     4     2
     3     6     2
% sum of each column (columnwise sum)
>>sum(X)
ans =
     4    10     4
% Generally, func(X, dim):
% dim = 1 ' columnwise operation
% dim = 2 ' rowwise operation
% default dim (dimension) is 1 (columnwise) if no explicit dimension is given

% sum of each row
>>sum(X, 2)
ans =
     7
    11
% sum of all elements of a matrix
>>sum(sum(X));
% mean value of each column
>> mean(X)
ans =
     2     5     2
% look for other functions like min(), max(), std(), var(), cov(), median() in matlab help
```

```
>X = matrix(c(1,4,2,3,6,2),
nrow=2,byrow=TRUE)
> X
     [,1] [,2] [,3]
[1,]    1    4    2
[2,]    3    6    2
# sum of each column (columnwise sum)
>colSums(X)
[1]  4 10  4
# sum of each row
>rowSums(X)
[1]  7 11
# Note that the functions colSums() and rowSums() returns vectors; not matrices.

# sum of all elements of a matrix

>sum(A)

# mean value of each column
>colMeans(X)
[1] 2 5 2
# mean value of each row
>rowMeans(X)
[1] 2.333333 3.666667
# see other functions like apply(X,1,sd), apply(X,1,var), cov() and median() in R
```

Text output to console

```
% display simple text only
>> disp('this is a sentence');
this is a sentence
% text concatenate
>> str = ['motor ', 'car', 's']
str =
motor cars
>> v = [1 2 3 4];
>> disp(['v = ', num2str(v)]);
v = 1  2  3  4
% text output with an integer parameter, "\n" for explicit line feed
>> fprintf('a is equal to %d\n', 5)
a is equal to 5
% text output with a floating number
>>fprintf('b is equal to %f\n', sqrt(2))
b is equal to 1.414214
% number formatting: Show two decimals after fractional point
>> fprintf('b is equal to %.2f\n', sqrt(2))
b is equal to 1.41
>> fprintf('Results: a = %d, b = %.1f,
   c = %.1f\n', 4, 5.18, 6.12);
Results: a = 4, b = 5.2, c = 6.1
```

```
# display simple text only
>print('this is a sentence');
[1] "this is a sentence"
# text concatenate
> str = paste('motor ','car','s',sep='');
> str
[1] "motor cars"
>v = c(1,2,3,4)
> print(paste('v =',
  paste(as.character(v),collapse=' ')))
[1] "v = 1 2 3 4"
# text output with an integer parameter
> str = sprintf('a is equal to %d', 5)
> str
[1] "a is equal to 5"
# text output with a floating number
> str = sprintf('b is equal to %f', sqrt(2))
> str
[1] "b is equal to 1.414214"
# number formatting: Show two decimals after fractional point
> sprintf('b is equal to %.2f', sqrt(2))
[1] "b is equal to 1.41"
> sprintf('Results: a = %d, b = %.1f,
  c = %.2f', 4, 5.18, 6.12);
[1] "Results: a = 4, b = 5.2, c = 6.12"
```

Programming language constructs

```
% while loop
>> v = [8 5 3 7 2 8 1 2];
% start from left, find the first number smaller than or equal to 2
>> i = 1;
>> while v(i) > 2
i = i+1;
end
>> fprintf('Results: i = %d,v(i) = %d\n',i,v(i))
Results: i = 5, v(i) = 2
% alternative method
>>a = find(v <= 2);
>>i = a(1)
% look for find() in matlab help: It is a useful and versatile search function

% for loop
% generate identity matrix (all diagonal elements 1, others 0)
>> I = zeros(3,3);
>> for i=1:3
I(i, i) = 1;
end
>> I
I =
     1      0      0
     0      1      0
     0      0      1
% alternative method
>> I = eye(3);

% if statements
>> x = 5;
% check if x is in range (2,10)
>> if (x > 2) && (x < 10)
disp('x is in range')
else
disp('x is not in range')
end
x is in range

% see keywords like switch, break and continue for other programming constructs
```

```
# while loop
>> v = c(8,5,3,7,2,8,1,2);
# start from left, find the first number smaller than or equal to 2
> i=1
> while (v[i]>2){
i=i+1
}
> str=sprintf('Results: i = %d, v[i] = %d',i,v[i])
> print(str)
[1] "Results: i = 5, v[i] = 2"
# alternative method
> a = which(v <= 2)
> i = a[1]
# look for which() in R help; it is a useful search function

# for loop
# generate identity matrix (all diagonal elements 1, others 0)
> I = matrix(0, 3, 3)
> for (i in 1:3) {
I[i,i]=1
}
> I
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
# alternative method
> I = diag(3)

# if statements
> x = 5;
# check if x is in range (2,10)
> if ((x > 2) && (x < 10)) {
print('x is in range')
} else {
print('x is not in range')}
[1] "x is in range"

# see keywords like switch, ifelse, repeat and next for other programming constructs
```

Writing scripts and functions

```
% script "test1.m" (text file) in current (working) directory
% start script
disp('This is a test script');
x=5
% end script
% running the script from console
>>test1
This is a test script
x =
     5

% function with multiple arguments and a single return value: text file "difference.m" in
current directory:
function c = difference(a,b)
c = a - b;
end
% calling the function from console
>> d = difference(10,2)
d =
```

```
# script "test1.r" (text file) in current (working) directory (menu "File>New script")
# change the working directory from menu "File>Change diré " if necessary
# start script
print('This is a test script');
x = 5
print(sprintf('x = %d', x))
# end script
# running the script from console
> source('test1.r')
[1] "This is a test script"
[1] "x = 5"

# function with multiple arguments and a single return value: any text file, for example
"test1.r" in current directory
difference = function(a,b) {
return(a - b)
}
# calling the function from console
> source('test1.r')
```

```
     8

% function with multiple return values; text file sumdif.m in current directory:
function [s,d] = sumdif(a,b)
s = a + b;
d = a - b;
end
% calling the function from console
>> [s,d] = sumdif(5,3)
s =
     8
d =
     2
```

```
> d = difference(10,2)
> d
[1] 8

# function with multiple return values; any text file, for example "test1.r" in current directory
sumdif = function(a,b) {
s = a + b
d = a - b
return(list(s, d))
}
# calling the function from console
> source('test1.r')
> results = sumdif(5,3)
> s = results[[1]]
> s
[1] 8
> d = results[[2]]
> d
[1] 2
```

Plotting graphs

```
% plot x-square function
>>x = 1:100;
>> y = x .^ 2;
>> plot(x, y);
>> title('y = x-square')
>> xlabel('x')
>> ylabel('y')
>> figure % opens new diagram
% plot two curves on the same graph
% first curve is red, second is blue
>>x1 = 1:100;
>>y1 = x1 .^ 2;
>>x2 = x1;
>>y2 = (x2 - 5) .^ 2;
>> plot(x1, y1,'r', x2, y2,'b')

% histogram
% random number with normal distribution
>> x = randn(1,10000);
>> hist(x)
% random number with uniform distribution
>> x = rand(1,10000);
>> hist(x)

% z = (x + 0.5*y - 5) ^ 2
>> x = 0:0.1:5;
>> y = 0:0.2:10;
>> [X,Y] = meshgrid(x,y);
>> Z = (X + 0.5*Y -5) .^2;
% surface plot
>> surf(X,Y,Z)
% contour lines (level curves)
>> contour(X,Y,Z)
```

```
# plot x-square function
> x=1:100;
> y = x * x
> plot(x,y,type='l',xlab='x',ylab='y',
  main='y = x-square')
> dev.new() # opens new diagram
# plot two curves on the same graph
# first curve is red, second is blue
> x1 = 1:100
> y1 = x ^2
> y2 = (x-5)^2
> plot(x,y1,type="l",col="red")
> lines(x,y2,col="blue")

# histogram
# random number with normal distribution
> x = rnorm(10000, sd=4, mean=5);
> hist(x)
# random number with uniform distribution
> x = runif(10000);
> hist(x)

# z = (x + 0.5*y - 5) ^ 2
> x = seq(0, 5, 0.1)
> y = seq(0, 10, 0.2)
> f = function(x,y) return((x + 0.5*y -5)^2)
> z = outer(x,y,f)
# surface plot
> persp(x,y,z)
# contour lines (level curves)
> contour(x,y,z)
```

Constrained optimization

Solving optimization (max/min) problems subject to given constraints (boundary conditions)

```
% general syntax:
% x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
- fun: single-output objective function whose return is to be minimized
- x0: start value for vector x
- inequality constraint: Ax Òb
- equality constraint: Aeq x = beq
- lb: lower bound for x
- hb: higher bound for x

% Problem:
Minimize z = sin(x + 0.5y) subject to constraints:
x + y Ó5 and x - y Ò2

% STEP 1: Create objective function objfun; text file objfun.m in current directory:

% Note: x --> v(1), y --> v(2)
function z = objfun(v)
z = sin(v(1) + 0.5*v(2));
end

% STEP 2: Formulate boundary conditions

% inequlity condition:
% -(x +y) Ò-5
% x - y Ò2
>>A = [-1 -1; 1 -1];
>>b = [-5 2];
% there is no equality condition
>> Aeq = [];  % empty matrix
>> beq = [];
% there are no lower or upper boundaries for x or y (v(1) or v(2))
```

```
# general syntax: # x = constrOptim(x0, fun, grad, A, b, é )
- fun: single-output objective function whose return is to be minimized
- x0: start value for vector x
- inequality constraint: Ax Ób
- grad: gradient function for fun. It can be "null" if it is not known.

# Problem:
Minimize z = sin(x + 0.5y) subject to constraints:
x + y Ó5 and x - y Ò2

# STEP 1: Create objective function objfun; in any text file, for example test1.r in current directory:

# Note: x --> v(1), y --> v(2)
objfun = function(v) {
z = sin(v[1] + 0.5*v[2])
return(z)
}
# declaring the function in console
> source('test1.r')

# STEP 2: Formulate boundary conditions
# x + y Ó5
# x - y Ò2 --> -x + y Ó-2
> A = matrix(c(1,1,-1,1), 2,2,byrow=TRUE)
> b=matrix(c(5,-2),2,1)
>grad = NULL
# set a starting point for vector v
# Note: initial value should satisfy the boundary conditions
> v0 = c(3,4)
```

```
>>lb = [];
>>ub = [];
% set a starting point for vector v
>>v0 = [0 0];

% STEP 3: Run optimization function
>> v = fmincon(@objfun,v0,A,b,Aeq,beq,lb,ub);
>> v
v =
    2.9302    3.5663

% Test the result
% Value of objective function at this point v
>> objfun(v)
z =
   -1.0000
% Inequality condition 1: check if x + y Ó5
>> sum(v)
ans =
    6.4965
% Inequality condition 2: check if x - y Ò2
>> v(1)-v(2)
ans =
   -0.6361
```

```

# STEP 3: Run optimization function
> result = constrOptim(v0, objfun, grad, A, b)
> result
$par
[1] 2.740897 3.942860
$value
[1] -1

# Test the result
# Value of objective function at this point v
> v = result[[1]]
> objfun(v)
[1] -1
# Inequality condition 1: check if x + y Ó5
> sum(v)
[1] 6.683757
# Inequality condition 2: check if x - y Ò2
> v[1]-v[2]
[1] -1.201963
```