[Efficient calculation of matrix inverse in R](http://stats.stackexchange.com/questions/14951/efficient-calculation-of-matrix-inverse-in-r)

Have you tried what cardinal suggested and explored some of the alternative methods for computing the inverse? Let's consider a specific example:

library(MASS)

k <- 2000

rho <- .3

S <- matrix(rep(rho, k\*k), nrow=k)

diag(S) <- 1

dat <- mvrnorm(10000, mu=rep(0,k), Sigma=S) ### be patient!

R <- cor(dat)

system.time(RI1 <- solve(R))

system.time(RI2 <- chol2inv(chol(R)))

system.time(RI3 <- qr.solve(R))

all.equal(RI1, RI2)

all.equal(RI1, RI3)

So, this is an example of a 2000×2000 correlation matrix for which we want the inverse. On my laptop (Core-i5 2.50Ghz), solve takes 8-9 seconds, chol2inv(chol()) takes a bit over 4 seconds, and qr.solve() takes 17-18 seconds (multiple runs of the code are suggested to get stable results).

So the inverse via the Choleski decomposition is about twice as fast as solve. There may of course be even faster ways of doing that. I just explored some of the most obvious ones here. And as already mentioned in the comments, if the matrix has a special structure, then this probably can be exploited for more speed.

# [Matrix element division in R](http://stackoverflow.com/questions/11368894/matrix-element-division-in-r)

What you actually have there is a data frame. It's essentially a matrix, you're right, but you access the columns by using the column's names.

Accessing each column of the data frame can be done through a command like this:

Matrix$close

This should give you the desired data frame, if I understood your question correctly.

New\_DataFrame <- data.frame(close = Matrix$close / (Matrix$close.1 \* Matrix$close.2), close.1 = Matrix$close.1 / Matrix$close.2)

These operations are all done in respect to each individual row.

If you want your answer in the form of a matrix instead of a data frame, use this:

New\_Matrix <- data.matrix(New\_DataFrame)

And switching back to a data frame from a matrix is as easy as:

New\_DataFrame <- data.frame(New\_Matrix)

Hope that helps!

f mat is your matrix, then mat[,1]/mat[,2] gives you the element-wise division of each row. If matis actually a data.frame not a matrix, then the above works, as does mat$close/mat$close.1.

|  |
| --- |
| [share](http://stackoverflow.com/a/11368958)|[improve this answer](http://stackoverflow.com/posts/11368958/edit) |

# R for MATLAB users

### Help

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| help.start() | doc help -i % browse with Info | Browse help interactively |
| help() | help help *or* doc doc | Help on using help |
| help(plot) *or* ?plot | help plot | Help for a function |
| help(package='splines') | help splines *or* doc splines | Help for a toolbox/library package |
| demo() | demo | Demonstration examples |
| example(plot) |  | Example using a function |

### Searching available documentation

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| help.search('plot') | lookfor plot | Search help files |
| apropos('plot') |  | Find objects by partial name |
| library() | help | List available packages |
| find(plot) | which plot | Locate functions |
| methods(plot) |  | List available methods for a function |

### Using interactively

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| Rgui | octave -q | Start session |
| source('foo.R') | foo(.m) | Run code from file |
| history() | history | Command history |
| savehistory(file=".Rhistory") | diary on [..] diary off | Save command history |
| q(save='no') | exit *or* quit | End session |

### Operators

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| help(Syntax) | help - | Help on operator syntax |

### Arithmetic operators

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a<-1; b<-2 | a=1; b=2; | Assignment; defining a number |
| a + b | a + b | Addition |
| a - b | a - b | Subtraction |
| a \* b | a \* b | Multiplication |
| a / b | a / b | Division |
| a ^ b | a .^ b | Power, $a^b$ |
| a %% b | rem(a,b) | Remainder |
| a %/% b |  | Integer division |
| factorial(a) | factorial(a) | Factorial, $n!$ |

### Relational operators

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a == b | a == b | Equal |
| a < b | a < b | Less than |
| a > b | a > b | Greater than |
| a <= b | a <= b | Less than or equal |
| a >= b | a >= b | Greater than or equal |
| a != b | a ~= b | Not Equal |

### Logical operators

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a && b | a && b | Short-circuit logical AND |
| a || b | a || b | Short-circuit logical OR |
| a & b | a & b *or* and(a,b) | Element-wise logical AND |
| a | b | a | b *or* or(a,b) | Element-wise logical OR |
| xor(a, b) | xor(a, b) | Logical EXCLUSIVE OR |
| !a | ~a *or* not(a) ~a *or* !a | Logical NOT |
|  | any(a) | True if any element is nonzero |
|  | all(a) | True if all elements are nonzero |

### root and logarithm

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| sqrt(a) | sqrt(a) | Square root |
| log(a) | log(a) | Logarithm, base $e$ (natural) |
| log10(a) | log10(a) | Logarithm, base 10 |
| log2(a) | log2(a) | Logarithm, base 2 (binary) |
| exp(a) | exp(a) | Exponential function |

### Round off

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| round(a) | round(a) | Round |
| ceil(a) | ceil(a) | Round up |
| floor(a) | floor(a) | Round down |
|  | fix(a) | Round towards zero |

### Mathematical constants

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| pi | pi | $\pi=3.141592$ |
| exp(1) | exp(1) | $e=2.718281$ |

### Missing values; IEEE-754 floating point status flags

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
|  | NaN | Not a Number |
|  | Inf | Infinity, $\infty$ |

### Complex numbers

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| 1i | i | Imaginary unit |
| z <- 3+4i | z = 3+4i | A complex number, $3+4i$ |
| abs(3+4i) *or* Mod(3+4i) | abs(z) | Absolute value (modulus) |
| Re(3+4i) | real(z) | Real part |
| Im(3+4i) | imag(z) | Imaginary part |
| Arg(3+4i) | arg(z) | Argument |
| Conj(3+4i) | conj(z) | Complex conjugate |

### Trigonometry

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| atan2(b,a) | atan(a,b) | Arctangent, $\arctan(b/a)$ |

### Generate random numbers

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| runif(10) | rand(1,10) | Uniform distribution |
| runif(10, min=2, max=7) | 2+5\*rand(1,10) | Uniform: Numbers between 2 and 7 |
| matrix(runif(36),6) | rand(6) | Uniform: 6,6 array |
| rnorm(10) | randn(1,10) | Normal distribution |

### Vectors

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a <- c(2,3,4,5) | a=[2 3 4 5]; | Row vector, $1 \times n$-matrix |
| adash <- t(c(2,3,4,5)) | adash=[2 3 4 5]'; | Column vector, $m \times 1$-matrix |

### Sequences

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| seq(10) *or* 1:10 | 1:10 | 1,2,3, ... ,10 |
| seq(0,length=10) | 0:9 | 0.0,1.0,2.0, ... ,9.0 |
| seq(1,10,by=3) | 1:3:10 | 1,4,7,10 |
| seq(10,1) *or* 10:1 | 10:-1:1 | 10,9,8, ... ,1 |
| seq(from=10,to=1,by=-3) | 10:-3:1 | 10,7,4,1 |
| seq(1,10,length=7) | linspace(1,10,7) | Linearly spaced vector of n=7 points |
| rev(a) | reverse(a) | Reverse |
|  | a(:) = 3 | Set all values to same scalar value |

### Concatenation (vectors)

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| c(a,a) | [a a] | Concatenate two vectors |
| c(1:4,a) | [1:4 a] |  |

### Repeating

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| rep(a,times=2) | [a a] | 1 2 3, 1 2 3 |
| rep(a,each=3) |  | 1 1 1, 2 2 2, 3 3 3 |
| rep(a,a) |  | 1, 2 2, 3 3 3 |

### Miss those elements out

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a[-1] | a(2:end) | miss the first element |
| a[-10] | a([1:9]) | miss the tenth element |
| a[-seq(1,50,3)] |  | miss 1,4,7, ... |
|  | a(end) | last element |
|  | a(end-1:end) | last two elements |

### Maximum and minimum

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| pmax(a,b) | max(a,b) | pairwise max |
| max(a,b) | max([a b]) | max of all values in two vectors |
| v <- max(a) ; i <- which.max(a) | [v,i] = max(a) |  |

### Vector multiplication

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a\*a | a.\*a | Multiply two vectors |
|  | dot(u,v) | Vector dot product, $u \cdot v$ |

### Matrices

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| rbind(c(2,3),c(4,5)) array(c(2,3,4,5), dim=c(2,2)) | a = [2 3;4 5] | Define a matrix |

### Concatenation (matrices); rbind and cbind

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| rbind(a,b) | [a ; b] | Bind rows |
| cbind(a,b) | [a , b] | Bind columns |
|  | [a(:), b(:)] | Concatenate matrices into one vector |
| rbind(1:4,1:4) | [1:4 ; 1:4] | Bind rows (from vectors) |
| cbind(1:4,1:4) | [1:4 ; 1:4]' | Bind columns (from vectors) |

### Array creation

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| matrix(0,3,5) *or* array(0,c(3,5)) | zeros(3,5) | 0 filled array |
| matrix(1,3,5) *or* array(1,c(3,5)) | ones(3,5) | 1 filled array |
| matrix(9,3,5) *or* array(9,c(3,5)) | ones(3,5)\*9 | Any number filled array |
| diag(1,3) | eye(3) | Identity matrix |
| diag(c(4,5,6)) | diag([4 5 6]) | Diagonal |
|  | magic(3) | Magic squares; Lo Shu |

### Reshape and flatten matrices

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| matrix(1:6,nrow=3,byrow=T) | reshape(1:6,3,2)'; | Reshaping (rows first) |
| matrix(1:6,nrow=2) array(1:6,c(2,3)) | reshape(1:6,2,3); | Reshaping (columns first) |
| as.vector(t(a)) | a'(:) | Flatten to vector (by rows, like comics) |
| as.vector(a) | a(:) | Flatten to vector (by columns) |
| a[row(a) <= col(a)] | vech(a) | Flatten upper triangle (by columns) |

### Shared data (slicing)

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| b = a | b = a | Copy of a |

### Indexing and accessing elements (Python: slicing)

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a <- rbind(c(11, 12, 13, 14), c(21, 22, 23, 24), c(31, 32, 33, 34)) | a = [ 11 12 13 14 ... 21 22 23 24 ... 31 32 33 34 ] | Input is a 3,4 array |
| a[2,3] | a(2,3) | Element 2,3 (row,col) |
| a[1,] | a(1,:) | First row |
| a[,1] | a(:,1) | First column |
|  | a([1 3],[1 4]); | Array as indices |
| a[-1,] | a(2:end,:) | All, except first row |
|  | a(end-1:end,:) | Last two rows |
|  | a(1:2:end,:) | Strides: Every other row |
| a[-2,-3] |  | All, except row,column (2,3) |
| a[,-2] | a(:,[1 3 4]) | Remove one column |

### Assignment

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a[,1] <- 99 | a(:,1) = 99 |  |
| a[,1] <- c(99,98,97) | a(:,1) = [99 98 97]' |  |
| a[a>90] <- 90 | a(a>90) = 90; | Clipping: Replace all elements over 90 |

### Transpose and inverse

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| t(a) | a' | Transpose |
|  | a.' *or* transpose(a) | Non-conjugate transpose |
| det(a) | det(a) | Determinant |
| solve(a) | inv(a) | Inverse |
| ginv(a) | pinv(a) | Pseudo-inverse |
|  | norm(a) | Norms |
| eigen(a)$values | eig(a) | Eigenvalues |
| svd(a)$d | svd(a) | Singular values |
|  | chol(a) | Cholesky factorization |
| eigen(a)$vectors | [v,l] = eig(a) | Eigenvectors |
| rank(a) | rank(a) | Rank |

### Sum

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| apply(a,2,sum) | sum(a) | Sum of each column |
| apply(a,1,sum) | sum(a') | Sum of each row |
| sum(a) | sum(sum(a)) | Sum of all elements |
| apply(a,2,cumsum) | cumsum(a) | Cumulative sum (columns) |

### Sorting

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
|  | a = [ 4 3 2 ; 2 8 6 ; 1 4 7 ] | Example data |
| t(sort(a)) | sort(a(:)) | Flat and sorted |
| apply(a,2,sort) | sort(a) | Sort each column |
| t(apply(a,1,sort)) | sort(a')' | Sort each row |
|  | sortrows(a,1) | Sort rows (by first row) |
| order(a) |  | Sort, return indices |

### Maximum and minimum

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| apply(a,2,max) | max(a) | max in each column |
| apply(a,1,max) | max(a') | max in each row |
| max(a) | max(max(a)) | max in array |
| i <- apply(a,1,which.max) | [v i] = max(a) | return indices, i |
| pmax(b,c) | max(b,c) | pairwise max |
| apply(a,2,cummax) | cummax(a) |  |

### Matrix manipulation

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a[,4:1] | fliplr(a) | Flip left-right |
| a[3:1,] | flipud(a) | Flip up-down |
|  | rot90(a) | Rotate 90 degrees |
| kronecker(matrix(1,2,3),a) | repmat(a,2,3) kron(ones(2,3),a) | Repeat matrix: [ a a a ; a a a ] |
| a[lower.tri(a)] <- 0 | triu(a) | Triangular, upper |
| a[upper.tri(a)] <- 0 | tril(a) | Triangular, lower |

### Equivalents to "size"

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| dim(a) | size(a) | Matrix dimensions |
| ncol(a) | size(a,2) *or* length(a) | Number of columns |
| prod(dim(a)) | length(a(:)) | Number of elements |
|  | ndims(a) | Number of dimensions |
| object.size(a) |  | Number of bytes used in memory |

### Matrix- and elementwise- multiplication

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a \* b | a .\* b | Elementwise operations |
| a %\*% b | a \* b | Matrix product (dot product) |
| outer(a,b) *or* a %o% b |  | Outer product |
| crossprod(a,b) *or* t(a) %\*% b |  | Cross product |
| kronecker(a,b) | kron(a,b) | Kronecker product |
|  | a / b | Matrix division, $b{\cdot}a^{-1}$ |
| solve(a,b) | a \ b | Left matrix division, $b^{-1}{\cdot}a$ \newline (solve linear equations) |

### Find; conditional indexing

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| which(a != 0) | find(a) | Non-zero elements, indices |
| which(a != 0, arr.ind=T) | [i j] = find(a) | Non-zero elements, array indices |
| ij <- which(a != 0, arr.ind=T); v <- a[ij] | [i j v] = find(a) | Vector of non-zero values |
| which(a>5.5) | find(a>5.5) | Condition, indices |
| ij <- which(a>5.5, arr.ind=T); v <- a[ij] |  | Return values |
|  | a .\* (a>5.5) | Zero out elements above 5.5 |

### Multi-way arrays

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
|  | a = cat(3, [1 2; 1 2],[3 4; 3 4]); | Define a 3-way array |
|  | a(1,:,:) |  |

### File input and output

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| f <- read.table("data.txt") | f = load('data.txt') | Reading from a file (2d) |
| f <- read.table("data.txt") | f = load('data.txt') | Reading from a file (2d) |
| f <- read.table(file="data.csv", sep=";") | x = dlmread('data.csv', ';') | Reading fram a CSV file (2d) |
| write(f,file="data.txt") | save -ascii data.txt f | Writing to a file (2d) |

### Plotting

### Basic x-y plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| plot(a, type="l") | plot(a) | 1d line plot |
| plot(x[,1],x[,2]) | plot(x(:,1),x(:,2),'o') | 2d scatter plot |
|  | plot(x1,y1, x2,y2) | Two graphs in one plot |
| plot(x1,y1) matplot(x2,y2,add=T) | plot(x1,y1) hold on plot(x2,y2) | Overplotting: Add new plots to current |
|  | subplot(211) | subplots |
| plot(x,y,type="b",col="red") | plot(x,y,'ro-') | Plotting symbols and color |

### Axes and titles

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| grid() | grid on | Turn on grid lines |
| plot(c(1:10,10:1), asp=1) | axis equal axis('equal') replot | 1:1 aspect ratio |
| plot(x,y, xlim=c(0,10), ylim=c(0,5)) | axis([ 0 10 0 5 ]) | Set axes manually |
| plot(1:10, main="title", xlab="x-axis", ylab="y-axis") | title('title') xlabel('x-axis') ylabel('y-axis') | Axis labels and titles |

### Log plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| plot(x,y, log="y") | semilogy(a) | logarithmic y-axis |
| plot(x,y, log="x") | semilogx(a) | logarithmic x-axis |
| plot(x,y, log="xy") | loglog(a) | logarithmic x and y axes |

### Filled plots and bar plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| plot(t,s, type="n", xlab="", ylab="") polygon(t,s, col="lightblue") polygon(t,c, col="lightgreen") | fill(t,s,'b', t,c,'g') % fill has a bug? | Filled plot |
| stem(x[,3]) |  | Stem-and-Leaf plot |

### Functions

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| f <- function(x) sin(x/3) - cos(x/5) | f = inline('sin(x/3) - cos(x/5)') | Defining functions |
| plot(f, xlim=c(0,40), type='p') | ezplot(f,[0,40]) fplot('sin(x/3) - cos(x/5)',[0,40]) % no ezplot | Plot a function for given range |

### Polar plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
|  | theta = 0:.001:2\*pi; r = sin(2\*theta); |  |
|  | polar(theta, rho) |  |

### Histogram plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| hist(rnorm(1000)) | hist(randn(1000,1)) |  |
| hist(rnorm(1000), breaks= -4:4) | hist(randn(1000,1), -4:4) |  |
| hist(rnorm(1000), breaks=c(seq(-5,0,0.25), seq(0.5,5,0.5)), freq=F) |  |  |
| plot(apply(a,1,sort),type="l") | plot(sort(a)) |  |

### 3d data

### Contour and image plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| contour(z) | contour(z) | Contour plot |
| filled.contour(x,y,z, nlevels=7, color=gray.colors) | contourf(z); colormap(gray) | Filled contour plot |
| image(z, col=gray.colors(256)) | image(z) colormap(gray) | Plot image data |
|  | quiver() | Direction field vectors |

### Perspective plots of surfaces over the x-y plane

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| f <- function(x,y) x\*exp(-x^2-y^2) n <- seq(-2,2, length=40) z <- outer(n,n,f) | n=-2:.1:2; [x,y] = meshgrid(n,n); z=x.\*exp(-x.^2-y.^2); |  |
| persp(x,y,z, theta=30, phi=30, expand=0.6, ticktype='detailed') | mesh(z) | Mesh plot |
| persp(x,y,z, theta=30, phi=30, expand=0.6, col='lightblue', shade=0.75, ltheta=120, ticktype='detailed') | surf(x,y,z) *or* surfl(x,y,z) % no surfl() | Surface plot |

### Scatter (cloud) plots

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| cloud(z~x\*y) | plot3(x,y,z,'k+') | 3d scatter plot |

### Save plot to a graphics file

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| postscript(file="foo.eps") plot(1:10) dev.off() | plot(1:10) print -depsc2 foo.eps gset output "foo.eps" gset terminal postscript eps plot(1:10) | PostScript |
| pdf(file='foo.pdf') |  | PDF |
| devSVG(file='foo.svg') |  | SVG (vector graphics for www) |
| png(filename = "Rplot%03d.png" | print -dpng foo.png | PNG (raster graphics) |

### Data analysis

### Set membership operators

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| a <- c(1,2,2,5,2) b <- c(2,3,4) | a = [ 1 2 2 5 2 ]; b = [ 2 3 4 ]; | Create sets |
| unique(a) | unique(a) | Set unique |
| union(a,b) | union(a,b) | Set union |
| intersect(a,b) | intersect(a,b) | Set intersection |
| setdiff(a,b) | setdiff(a,b) | Set difference |
| setdiff(union(a,b),intersect(a,b)) | setxor(a,b) | Set exclusion |
| is.element(2,a) *or* 2 %in% a | ismember(2,a) | True for set member |

### Statistics

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| apply(a,2,mean) | mean(a) | Average |
| apply(a,2,median) | median(a) | Median |
| apply(a,2,sd) | std(a) | Standard deviation |
| apply(a,2,var) | var(a) | Variance |
| cor(x,y) | corr(x,y) | Correlation coefficient |
| cov(x,y) | cov(x,y) | Covariance |

### Interpolation and regression

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| z <- lm(y~x) plot(x,y) abline(z) | z = polyval(polyfit(x,y,1),x) plot(x,y,'o', x,z ,'-') | Straight line fit |
| solve(a,b) | a = x\y | Linear least squares $y = ax + b$ |
|  | polyfit(x,y,3) | Polynomial fit |

### Non-linear methods

### Polynomials, root finding

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| polyroot(c(1,-1,-1)) | roots([1 -1 -1]) | Find zeros of polynomial |
|  | f = inline('1/x - (x-1)') fzero(f,1) | Find a zero near $x = 1$ |
|  | solve('1/x = x-1') | Solve symbolic equations |
|  | polyval([1 2 1 2],1:10) | Evaluate polynomial |

### Differential equations

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
|  | diff(a) | Discrete difference function and approximate derivative |
|  |  | Solve differential equations |

### Fourier analysis

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| fft(a) | fft(a) | Fast fourier transform |
| fft(a, inverse=TRUE) | ifft(a) | Inverse fourier transform |

### Symbolic algebra; calculus

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
|  | factor() | Factorization |

### Programming

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| .R | .m | Script file extension |
| # | % % *or* # | Comment symbol (rest of line) |
| library(RSvgDevice) | % must be in MATLABPATH % must be in LOADPATH | Import library functions |
| string <- "a <- 234" eval(parse(text=string)) | string='a=234'; eval(string) | Eval |

### Loops

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| for(i in 1:5) print(i) | for i=1:5; disp(i); end | for-statement |
| for(i in 1:5) { print(i) print(i\*2) } | for i=1:5 disp(i) disp(i\*2) end | Multiline for statements |

### Conditionals

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| if (1>0) a <- 100 | if 1>0 a=100; end | if-statement |
|  | if 1>0 a=100; else a=0; end | if-else-statement |
| ifelse(a>0,a,0) |  | Ternary operator (if?true:false) |

### Debugging

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| .Last.value | ans | Most recent evaluated expression |
| objects() | whos *or* who | List variables loaded into memory |
| rm(x) | clear x *or* clear [all] | Clear variable $x$ from memory |
| print(a) | disp(a) | Print |

### Working directory and OS

|  |  |  |
| --- | --- | --- |
| **R/S-Plus** | **MATLAB/Octave** | **Description** |
| list.files() *or* dir() | dir *or* ls | List files in directory |
| list.files(pattern="\.r$") | what | List script files in directory |
| getwd() | pwd | Displays the current working directory |
| setwd('foo') | cd foo | Change working directory |
| system("notepad") | !notepad system("notepad") | Invoke a System Command |

Time-stamp: "2007-11-09T16:46:36 vidar"  
[©2006 Vidar Bronken Gundersen, /mathesaurus.sf.net](http://mathesaurus.sourceforge.net/)  
Permission is granted to copy, distribute and/or modify this document as long as the above attribution is retained.

Optimization:

Method "SANN" is by default a variant of simulated annealing given in Belisle (1992). Simulated-annealing belongs to the class of stochastic global optimization methods. It uses only function values but is relatively slow. It will also work for non-differentiable functions. This implementation uses the Metropolis function for the acceptance probability. By default the next candidate point is generated from a Gaussian Markov kernel with scale proportional to the actual temperature. If a function to generate a new candidate point is given, method "SANN" can also be used to solve combinatorial optimization problems. Temperatures are decreased according to the logarithmic cooling schedule as given in Belisle (1992, p. 890); specifically, the temperature is set to temp / log(((t-1) %/% tmax)\*tmax + exp(1)), where t is the current iteration step and temp and tmax are specifiable via control, see below. Note that the "SANN" method depends critically on the settings of the control parameters. It is not a general-purpose method but can be very useful in getting to a good value on a very rough surface.

The default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.

Method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.

Method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak–Ribiere or Beale–Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

Method "L-BFGS-B" is that of Byrd et. al. (1995) which allows box constraints, that is each variable can be given a lower and/or upper bound. The initial value must satisfy the constraints. This uses a limited-memory modification of the BFGS quasi-Newton method. If non-trivial bounds are supplied, this method will be selected, with a warning.

Nocedal and Wright (1999) is a comprehensive reference for the previous three methods.

# [Moving beyond R's optim function](http://stackoverflow.com/questions/3757321/moving-beyond-rs-optim-function)

Tried with the nlm() function already? Don't know if it's much faster, but it does improve speed. Also check the options. optim uses a slow algorithm as the default. You can gain a > 5-fold speedup by using the Quasi-Newton algorithm (method="BFGS") instead of the default. If you're not concerned too much about the last digits, you can also set the tolerance levels higher of nlm() to gain extra speed.

f <- function(x) sum((x-1:length(x))^2)

a <- 1:5

system.time(replicate(500,

optim(a,f)

))

user system elapsed

0.78 0.00 0.79

system.time(replicate(500,

optim(a,f,method="BFGS")

))

user system elapsed

0.11 0.00 0.11

system.time(replicate(500,

nlm(f,a)

))

user system elapsed

0.10 0.00 0.09

system.time(replicate(500,

nlm(f,a,steptol=1e-4,gradtol=1e-4)

))

user system elapsed

0.03 0.00 0.03

|  |  |
| --- | --- |
| constrOptim {stats} | R Documentation |

## Linearly Constrained Optimization

### Description

Minimise a function subject to linear inequality constraints using an adaptive barrier algorithm.

### Usage

constrOptim(theta, f, grad, ui, ci, mu = 1e-04, control = list(),

method = if(is.null(grad)) "Nelder-Mead" else "BFGS",

outer.iterations = 100, outer.eps = 1e-05, ...,

hessian = FALSE)

### Arguments

|  |  |
| --- | --- |
| theta | numeric (vector) starting value (of length *p*): must be in the feasible region. |
| f | function to minimise (see below). |
| grad | gradient of f (a [function](http://stat.ethz.ch/R-manual/R-patched/library/base/html/function.html) as well), or NULL (see below). |
| ui | constraint matrix (*k x p*), see below. |
| ci | constraint vector of length *k* (see below). |
| mu | (Small) tuning parameter. |
| control, method, hessian | passed to [optim](http://stat.ethz.ch/R-manual/R-patched/library/stats/html/optim.html). |
| outer.iterations | iterations of the barrier algorithm. |
| outer.eps | non-negative number; the relative convergence tolerance of the barrier algorithm. |
| ... | Other named arguments to be passed to f and grad: needs to be passed through [optim](http://stat.ethz.ch/R-manual/R-patched/library/stats/html/optim.html) so should not match its argument names. |

### Details

The feasible region is defined by ui %\*% theta - ci >= 0. The starting value must be in the interior of the feasible region, but the minimum may be on the boundary.

A logarithmic barrier is added to enforce the constraints and then [optim](http://stat.ethz.ch/R-manual/R-patched/library/stats/html/optim.html) is called. The barrier function is chosen so that the objective function should decrease at each outer iteration. Minima in the interior of the feasible region are typically found quite quickly, but a substantial number of outer iterations may be needed for a minimum on the boundary.

The tuning parameter mu multiplies the barrier term. Its precise value is often relatively unimportant. As mu increases the augmented objective function becomes closer to the original objective function but also less smooth near the boundary of the feasible region.

Any optim method that permits infinite values for the objective function may be used (currently all but "L-BFGS-B").

The objective function f takes as first argument the vector of parameters over which minimisation is to take place. It should return a scalar result. Optional arguments ... will be passed to optim and then (if not used byoptim) to f. As with optim, the default is to minimise, but maximisation can be performed by setting control$fnscale to a negative value.

The gradient function grad must be supplied except with method = "Nelder-Mead". It should take arguments matching those of f and return a vector containing the gradient.

### Value

As for [optim](http://stat.ethz.ch/R-manual/R-patched/library/stats/html/optim.html), but with two extra components: barrier.value giving the value of the barrier function at the optimum and outer.iterations gives the number of outer iterations (calls to optim). The countscomponent contains the sum of all [optim](http://stat.ethz.ch/R-manual/R-patched/library/stats/html/optim.html)()$counts.

### Examples

## from optim

fr <- function(x) { ## Rosenbrock Banana function

x1 <- x[1]

x2 <- x[2]

100 \* (x2 - x1 \* x1)^2 + (1 - x1)^2

}

grr <- function(x) { ## Gradient of 'fr'

x1 <- x[1]

x2 <- x[2]

c(-400 \* x1 \* (x2 - x1 \* x1) - 2 \* (1 - x1),

200 \* (x2 - x1 \* x1))

}

optim(c(-1.2,1), fr, grr)

#Box-constraint, optimum on the boundary

constrOptim(c(-1.2,0.9), fr, grr, ui = rbind(c(-1,0), c(0,-1)), ci = c(-1,-1))

# x <= 0.9, y - x > 0.1

constrOptim(c(.5,0), fr, grr, ui = rbind(c(-1,0), c(1,-1)), ci = c(-0.9,0.1))

## Solves linear and quadratic programming problems

## but needs a feasible starting value

#

# from example(solve.QP) in 'quadprog'

# no derivative

fQP <- function(b) {-sum(c(0,5,0)\*b)+0.5\*sum(b\*b)}

Amat <- matrix(c(-4,-3,0,2,1,0,0,-2,1), 3, 3)

bvec <- c(-8, 2, 0)

constrOptim(c(2,-1,-1), fQP, NULL, ui = t(Amat), ci = bvec)

# derivative

gQP <- function(b) {-c(0, 5, 0) + b}

constrOptim(c(2,-1,-1), fQP, gQP, ui = t(Amat), ci = bvec)

## Now with maximisation instead of minimisation

hQP <- function(b) {sum(c(0,5,0)\*b)-0.5\*sum(b\*b)}

constrOptim(c(2,-1,-1), hQP, NULL, ui = t(Amat), ci = bvec,

control = list(fnscale = -1))

# [How to set limits using constrOptim in R?](http://stats.stackexchange.com/questions/27030/how-to-set-limits-using-constroptim-in-r)

Your constraints are of two types, either θi≥ai, or θi≤bi. The first ones are already in the right form (and the matrix ui is just the identity matrix), while the others can be written as −θi≥−bi: ui is then −In and ci is −b.

# Constraints

bounds <- matrix(c(

0,5,

0,Inf,

0,Inf,

0,1

), nc=2, byrow=TRUE)

colnames(bounds) <- c("lower", "upper")

# Convert the constraints to the ui and ci matrices

n <- nrow(bounds)

ui <- rbind( diag(n), -diag(n) )

ci <- c( bounds[,1], - bounds[,2] )

# Remove the infinite values

i <- as.vector(is.finite(bounds))

ui <- ui[i,]

ci <- ci[i]

# Constrained minimization

f <- function(u) sum((u+1)^2)

constrOptim(c(1,1,.01,.1), f, grad=NULL, ui=ui, ci=ci)

We can check how the constraint matrices ci and ui are interpreted:

# Print the constraints

k <- length(ci)

n <- dim(ui)[2]

for(i in seq\_len(k)) {

j <- which( ui[i,] != 0 )

cat(paste( ui[i,j], " \* ", "x[", (1:n)[j], "]", sep="", collapse=" + " ))

cat(" >= " )

cat( ci[i], "\n" )

}

# 1 \* x[1] >= 0

# 1 \* x[2] >= 0

# 1 \* x[3] >= 0

# 1 \* x[4] >= 0

# -1 \* x[1] >= -5

# -1 \* x[4] >= -1

Some of the algorithms in optim allow you to specify the lower and upper bounds directly: that is probably easier to use.

# [Apply function to xts object](http://stackoverflow.com/questions/12015741/apply-function-to-xts-object)

When you call apply with MARGIN=1, it's like passing each row to FUN. Your function is already vectorized, so you don't need to use apply. However, your function does not return anything. Try this:

library(quantmod)

getSymbols("SPY", src='yahoo', from='2010-01-01', to='2012-01-01')

dat <- cbind(Ad(SPY), SMA=SMA(Ad(SPY)))

signal<-function(x,y,z)

{

z$signals<-ifelse(x>y,1,0)

z

}

tail(signal(dat[, 1], dat[, 2], dat))

# SPY.Adjusted SMA signals

#2011-12-22 124.08 121.693 1

#2011-12-23 125.19 121.805 1

#2011-12-27 125.29 122.108 1

#2011-12-28 123.64 122.361 1

#2011-12-29 124.92 122.871 1

#2011-12-30 124.31 123.276 1

Actually, I try to avoid ifelse in situations like these because it is slower than doing this

signal<-function(x,y,z)

{

z$signals <- 0

z$signals[x > y] <- 1

z

}

# DEoptim.control {DEoptim}

**Control various aspects of the DEoptim implementation**

**Package:**

 DEoptim

**Version:**

 2.2-2

### Description

Allow the user to set some characteristics of the Differential Evolution optimization algorithm implemented in[DEoptim](http://inside-r.org/packages/cran/DEoptim).

### Usage

DEoptim.control(VTR = -Inf, strategy = 2, bs = FALSE, NP = NA,

itermax = 200, CR = 0.5, F = 0.8, trace = TRUE, initialpop = NULL,

storepopfrom = itermax + 1, storepopfreq = 1, p = 0.2, c = 0, reltol,

steptol, parallelType = 0, packages = c(), parVar = c(),

foreachArgs = list())

### Arguments

VTR

the value to be reached. The optimization process will stop if either the maximum number of iterationsitermax is reached or the best parameter vector bestmem has found a value fn(bestmem) <= VTR. Default to -**Inf**.

strategy

defines the Differential Evolution strategy used in the optimization procedure:  
1: DE / rand / 1 / bin (classical strategy)  
2: DE / local-to-best / 1 / bin (default)  
3: DE / best / 1 / bin with jitter  
4: DE / rand / 1 / bin with per-vector-dither  
5: DE / rand / 1 / bin with per-generation-dither  
6: DE / current-to-p-best / 1  
any value not above: variation to DE / rand / 1 / bin: either-or-algorithm. Default strategy is currently 2. See \*Details\*.

bs

if **FALSE** then every mutant will be tested against a member in the previous generation, and the best value will proceed into the next generation (this is standard trial vs. target selection). If **TRUE** then the old generation and NP mutants will be sorted by their associated objective function values, and the best NPvectors will proceed into the next generation (best of parent and child selection). Default is **FALSE**.

NP

number of population members. Defaults to **NA**; if the user does not change the value of NP from **NA** or specifies a value less than 4 it is reset when [DEoptim](http://inside-r.org/packages/cran/DEoptim) is called as 10\*[**length**](http://inside-r.org/r-doc/base/length)(lower). For many problems it is best to set NP to be at least 10 times the length of the parameter vector.

itermax

the maximum iteration (population generation) allowed. Default is 200.

CR

crossover probability from interval [0,1]. Default to 0.5.

F

differential weighting factor from interval [0,2]. Default to 0.8.

trace

Positive integer or logical value indicating whether printing of progress occurs at each iteration. The default value is **TRUE**. If a positive integer is specified, printing occurs every [**trace**](http://inside-r.org/r-doc/base/trace) iterations.

initialpop

an initial population used as a starting population in the optimization procedure. May be useful to speed up the convergence. Default to **NULL**. If given, each member of the initial population should be given as a row of a numeric matrix, so that initialpop is a matrix with NP rows and a number of columns equal to the length of the parameter vector to be optimized.

storepopfrom

from which generation should the following intermediate populations be stored in memory. Default toitermax + 1, i.e., no intermediate population is stored.

storepopfreq

the frequency with which populations are stored. Default to 1, i.e., every intermediate population is stored.

p

when strategy = 6, the top (100 \* p)% best solutions are used in the mutation. p must be defined in (0,1].

c

[**c**](http://inside-r.org/r-doc/base/c) controls the speed of the crossover adaptation. Higher values of [**c**](http://inside-r.org/r-doc/base/c) give more weight to the current successful mutations. [**c**](http://inside-r.org/r-doc/base/c) must be defined in (0,1].

reltol

relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor ofreltol \* ([**abs**](http://inside-r.org/r-doc/base/abs)(val) +    reltol) after steptol steps. Defaults to[**sqrt**](http://inside-r.org/r-doc/base/sqrt)([**.Machine**](http://inside-r.org/r-doc/base/.Machine)$double.eps), typically about 1e-8.

steptol

see reltol. Defaults to itermax.

parallelType

Defines the type of parallelization to employ, if any.  : The default, this uses [DEoptim](http://inside-r.org/packages/cran/DEoptim) one only one core.1: This uses all available cores, via the parallel package, to run [DEoptim](http://inside-r.org/packages/cran/DEoptim). If parallelType=1, then thepackages argument and the parVar argument need to specify the packages required by the objective function and the variables required in the environment, respectively. 2: This uses the foreach package for parallelism; see the sandbox directory in the source code for examples. If parallelType=1, then theforeachArgs argument can pass the options to be called with foreach.

packages

Used if parallelType=1; a list of package names (as strings) that need to be loaded for use by the objective function.

parVar

Used if parallelType=1; a list of variable names (as strings) that need to exist in the environment for use by the objective function or are used as arguments by the objective function.

foreachArgs

A list of named arguments for the foreach function from the package foreach. The arguments i,.combine and .export are not possible to set here; they are set internally.

### Details

This defines the Differential Evolution strategy used in the optimization procedure, described below in the terms used by Price et al. (2006); see also Mullen et al. (2009) for details.

* strategy = 1: DE / rand / 1 / bin.   
  This strategy is the classical approach for DE, and is described in [DEoptim](http://inside-r.org/packages/cran/DEoptim).
* strategy = 2: DE / local-to-best / 1 / bin.   
  In place of the classical DE mutation the expression is used, where old\_i,g and best\_g are the i-th member and best member, respectively, of the previous population. This strategy is currently used by default.
* strategy = 3: DE / best / 1 / bin with jitter.  
  In place of the classical DE mutation the expression is used, where jitter is defined as 0.0001 \* rand + F.
* strategy = 4: DE / rand / 1 / bin with per vector dither.  
  In place of the classical DE mutation the expression is used, where dither is calculated as F + \code{rand} \* (1 - F).
* strategy = 5: DE / rand / 1 / bin with per generation dither.  
  The strategy described for 4 is used, but dither is only determined once per-generation.
* strategy = 6: DE / current-to-p-best / 1.  
  The top (100\*p) percent best solutions are used in the mutation, where p is defined in (0,1].
* any value not above: variation to DE / rand / 1 / bin: either-or algorithm.  
  In the case that rand < 0.5, the classical strategy strategy = 1 is used. Otherwise, the expression is used.

Several conditions can cause the optimization process to stop:

* if the best parameter vector (bestmem) produces a value less than or equal to VTR (i.e. fn(bestmem)<= VTR), or
* if the maximum number of iterations is reached (itermax), or
* if a number (steptol) of consecutive iterations are unable to reduce the best function value by a certain amount (reltol \*       ([**abs**](http://inside-r.org/r-doc/base/abs)(val) + reltol)). 100\*reltol is approximately the percent change of the objective value required to consider the parameter set an improvement over the current best member.

Zhang and Sanderson (2009) define several extensions to the DE algorithm, including strategy 6, DE/current-to-p-best/1. They also define a self-adaptive mechanism for the other control parameters. This self-adaptation will speed convergence on many problems, and is defined by the control parameter [**c**](http://inside-r.org/r-doc/base/c). If [**c**](http://inside-r.org/r-doc/base/c) is non-zero, crossover and mutation will be adapted by the algorithm. Values in the range of [**c**](http://inside-r.org/r-doc/base/c)=.05 to [**c**](http://inside-r.org/r-doc/base/c)=.5appear to work best for most problems, though the adaptive algorithm is robust to a wide range of [**c**](http://inside-r.org/r-doc/base/c).

### Values

The default value of [**control**](http://inside-r.org/r-doc/boot/control) is the return value of DEoptim.control(), which is a list (and a member of the S3 class DEoptim.control) with the above elements.

### References

Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with DEoptim. An Application to Non-Convex Portfolio Optimization. URL The R Journal, 3(1), 27-34. URL <http://journal.r-project.org/2011-1/>.

Ardia, D., Ospina Arango, J.D., Giraldo Gomez, N.D. (2011) Jump-Diffusion Calibration using Differential Evolution. Wilmott Magazine, 55 (September), 76-79. URL [http://www.wilmott.com](http://www.wilmott.com/). Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline, J. (2011). DEoptim: An R Package for Global Optimization by Differential Evolution.Journal of Statistical Software, 40(6), 1-26. URL <http://www.jstatsoft.org/v40/i06/>. Price, K.V., Storn, R.M., Lampinen J.A. (2006) Differential Evolution - A Practical Approach to Global Optimization. Berlin Heidelberg: Springer-Verlag. ISBN 3540209506. Zhang, J. and Sanderson, A. (2009) Adaptive Differential EvolutionSpringer-Verlag. ISBN 978-3-642-01526-7

### Note

Further details and examples of the R package DEoptim can be found in Mullen et al. (2011) and Ardia et al. (2011a, 2011b) or look at the package's vignette by typing [**vignette**](http://inside-r.org/r-doc/utils/vignette)("DEoptim"). Also, an illustration of the package usage for a high-dimensional non-linear portfolio optimization problem is available by typing[**vignette**](http://inside-r.org/r-doc/utils/vignette)("DEoptimPortfolioOptimization").

Please cite the package in publications. Use [**citation**](http://inside-r.org/r-doc/utils/citation)("DEoptim").

### See Also

[DEoptim](http://inside-r.org/packages/cran/DEoptim) and DEoptim-[**methods**](http://inside-r.org/r-doc/utils/methods).

### Examples

*## set the population size to 20*

DEoptim.control(NP = 20)

*## set the population size, the number of iterations and don't*

*## display the iterations during optimization*

DEoptim.control(NP = 20, itermax = 100, [**trace**](http://inside-r.org/r-doc/base/trace) = **FALSE**)

### Author(s)

David Ardia, Katharine Mullen [mullenkate@gmail.com](mailto://mullenkate@gmail.com), Brian Peterson and Joshua Ulrich.

*Documentation reproduced from package DEoptim, version 2.2-2. License: GPL (>= 2)*

# DEoptim {DEoptim}

**Differential Evolution Optimization**

**Package:**

 DEoptim

**Version:**

 2.2-2

### Description

Performs evolutionary global optimization via the Differential Evolution algorithm.

### Usage

DEoptim(fn, lower, upper, control = DEoptim.control(), ..., fnMap=NULL)

### Arguments

fn

the function to be optimized (minimized). The function should have as its first argument the vector of real-valued parameters to optimize, and return a scalar real result. **NA** and **NaN** values are not allowed.

lower, upper

two vectors specifying scalar real lower and upper bounds on each parameter to be optimized, so that the i-th element of lower and upper applies to the i-th parameter. The implementation searches between lower and upper for the global optimum (minimum) of fn.

control

a list of control parameters; see DEoptim.control.

fnMap

an optional function that will be run after each population is created, but before the population is passed to the objective function. This allows the user to impose integer/cardinality constriants.

...

further arguments to be passed to fn.

### Details

[DEoptim](http://inside-r.org/packages/cran/DEoptim) performs optimization (minimization) of fn.

The [**control**](http://inside-r.org/r-doc/boot/control) argument is a list; see the help file for DEoptim.control for details.

The R implementation of Differential Evolution (DE), DEoptim, was first published on the Comprehensive RArchive Network (CRAN) in 2005 by David Ardia. Early versions were written in pure R. Since version 2.0-0 (published to CRAN in 2009) the package has relied on an interface to a C implementation of DE, which is significantly faster on most problems as compared to the implementation in pure R. The C interface is in many respects similar to the MS Visual C++ v5.0 implementation of the Differential Evolution algorithm distributed with the book Differential Evolution -- A Practical Approach to Global Optimization by Price, K.V., Storn, R.M., Lampinen J.A, Springer-Verlag, 2006, and found on-line at <http://www.icsi.berkeley.edu/~storn/>. Since version 2.0-3 the C implementation dynamically allocates the memory required to store the population, removing limitations on the number of members in the population and length of the parameter vectors that may be optimized. Since version 2.2-0, the package allows for parallel operation, so that the evaluations of the objective function may be performed using all available cores. This is accomplished using either the built-in parallel package or the foreach package. If parallel operation is desired, the user should set parallelType and make sure that the arguments and packages needed by the objective function are available; see DEoptim.control, the example below and examples in the sandbox directory for details. Since becoming publicly available, the package DEoptim has been used by several authors to solve optimization problems arising in diverse domains; see Mullen et al. (2011) for a review. To perform a maximization (instead of minimization) of a given function, simply define a new function which is the opposite of the function to maximize and apply [DEoptim](http://inside-r.org/packages/cran/DEoptim) to it. To integrate additional constraints (other than box constraints) on the parameters x of fn(x), for instance x[1] + x[2]^2 < 2, integrate the constraint within the function to optimize, for instance:

fn <- function(x){ if (x[1] + x[2]^2 >= 2){ r <- Inf else{ ... } return(r) }

This simplistic strategy usually does not work all that well for gradient-based or Newton-type methods. It is likely to be alright when the solution is in the interior of the feasible region, but when the solution is on the boundary, optimization algorithm would have a difficult time converging. Furthermore, when the solution is on the boundary, this strategy would make the algorithm converge to an inferior solution in the interior. However, for methods such as DE which are not gradient based, this strategy might not be that bad.

Note that [DEoptim](http://inside-r.org/packages/cran/DEoptim) stops if any **NA** or **NaN** value is obtained. You have to redefine your function to handle these values (for instance, set **NA** to **Inf** in your objective function).

It is important to emphasize that the result of [DEoptim](http://inside-r.org/packages/cran/DEoptim) is a random variable, i.e., different results may be obtained when the algorithm is run repeatedly with the same settings. Hence, the user should set the random seed if they want to reproduce the results, e.g., by setting **[set.seed](http://inside-r.org/r-doc/base/set.seed)**(1234) before the call of[DEoptim](http://inside-r.org/packages/cran/DEoptim).

[DEoptim](http://inside-r.org/packages/cran/DEoptim) relies on repeated evaluation of the objective function in order to move the population toward a global minimum. Users interested in making [DEoptim](http://inside-r.org/packages/cran/DEoptim) run as fast as possible should consider using the package in parallel mode (so that all CPU's available are used), and also ensure that evaluation of the objective function is as efficient as possible (e.g. by using vectorization in pure R code, or writing parts of the objective function in a lower-level language like C or Fortran). Further details and examples of the Rpackage DEoptim can be found in Mullen et al. (2011) and Ardia et al. (2011a, 2011b) or look at the package's vignette by typing [**vignette**](http://inside-r.org/r-doc/utils/vignette)("DEoptim"). Also, an illustration of the package usage for a high-dimensional non-linear portfolio optimization problem is available by typing[**vignette**](http://inside-r.org/r-doc/utils/vignette)("DEoptimPortfolioOptimization"). Please cite the package in publications. Use[**citation**](http://inside-r.org/r-doc/utils/citation)("DEoptim").

### Values

The output of the function [DEoptim](http://inside-r.org/packages/cran/DEoptim) is a member of the S3 class [DEoptim](http://inside-r.org/packages/cran/DEoptim). More precisely, this is a list (of length 2) containing the following elements:

[**optim**](http://inside-r.org/r-doc/stats/optim), a list containing the following elements:

* bestmem: the best set of parameters found.
* bestval: the value of fn corresponding to bestmem.
* nfeval: number of function evaluations.
* iter: number of procedure iterations.

member, a list containing the following elements:

* lower: the lower boundary.
* upper: the upper boundary.
* bestvalit: the best value of fn at each iteration.
* bestmemit: the best member at each iteration.
* pop: the population generated at the last iteration.
* storepop: a list containing the intermediate populations.

Members of the class [DEoptim](http://inside-r.org/packages/cran/DEoptim) have a [**plot**](http://inside-r.org/r-doc/graphics/plot) method that accepts the argument plot.type.  
plot.type = "bestmemit" results in a plot of the parameter values that represent the lowest value of the objective function each generation. plot.type = "bestvalit" plots the best value of the objective function each generation. Finally, plot.type = "storepop" results in a plot of stored populations (which are only available if these have been saved by setting the [**control**](http://inside-r.org/r-doc/boot/control) argument of [DEoptim](http://inside-r.org/packages/cran/DEoptim) appropriately). Storing intermediate populations allows us to examine the progress of the optimization in detail. A summary method also exists and returns the best parameter vector, the best value of the objective function, the number of generations optimization ran, and the number of times the objective function was evaluated.

### References

Differential Evolution homepage: URL <http://www.icsi.berkeley.edu/~storn/code.html>.

Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with DEoptim. An Application to Non-Convex Portfolio Optimization. The R Journal, 3(1), 27-34. URL <http://journal.r-project.org/2011-1/>.

Ardia, D., Ospina Arango, J.D., Giraldo Gomez, N.D. (2011) Jump-Diffusion Calibration using Differential Evolution. Wilmott Magazine, 55 (September), 76-79. URL [http://www.wilmott.com](http://www.wilmott.com/). Mitchell, M. (1998) An Introduction to Genetic Algorithms. The MIT Press. ISBN 0262631857.

Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline, J. (2011). DEoptim: An R Package for Global Optimization by Differential Evolution. Journal of Statistical Software, 40(6), 1-26. URL<http://www.jstatsoft.org/v40/i06/>.

Price, K.V., Storn, R.M., Lampinen J.A. (2006) Differential Evolution - A Practical Approach to Global Optimization. Berlin Heidelberg: Springer-Verlag. ISBN 3540209506.

Storn, R. and Price, K. (1997) Differential Evolution -- A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization, 11:4, 341--359.

### Note

Differential Evolution (DE) is a search heuristic introduced by Storn and Price (1997). Its remarkable performance as a global optimization algorithm on continuous numerical minimization problems has been extensively explored; see Price et al. (2006). DE belongs to the class of genetic algorithms which use biology-inspired operations of crossover, mutation, and selection on a population in order to minimize an objective function over the course of successive generations (see Mitchell, 1998). As with other evolutionary algorithms, DE solves optimization problems by evolving a population of candidate solutions using alteration and selection operators. DE uses floating-point instead of bit-string encoding of population members, and arithmetic operations instead of logical operations in mutation. DE is particularly well-suited to find the global optimum of a real-valued function of real-valued parameters, and does not require that the function be either continuous or differentiable.

Let NP denote the number of parameter vectors (members) x in R^d in the population. In order to create the initial generation, NP guesses for the optimal value of the parameter vector are made, either using random values between lower and upper bounds (defined by the user) or using values given by the user. Each generation involves creation of a new population from the current population members {x\_i | i=1,...,NP}, where i indexes the vectors that make up the population. This is accomplished using differential mutation of the population members. An initial mutant parameter vector v\_i is created by choosing three members of the population, x\_{r\_0}, x\_{r\_1} and x\_{r\_2}, at random. Then v\_i is generated as

v\_i := x\_{r\_0} + F \* (x\_{r\_1} - x\_{r\_2})

where F is the differential weighting factor, effective values for which are typically between 0 and 1. After the first mutation operation, mutation is continued until d mutations have been made, with a crossover probability CR in [0,1]. The crossover probability CR controls the fraction of the parameter values that are copied from the mutant. If an element of the trial parameter vector is found to violate the bounds after mutation and crossover, it is reset in such a way that the bounds are respected (with the specific protocol depending on the implementation). Then, the objective function values associated with the children are determined. If a trial vector has equal or lower objective function value than the previous vector it replaces the previous vector in the population; otherwise the previous vector remains. Variations of this scheme have also been proposed; see Price et al. (2006) and DEoptim.control.

Intuitively, the effect of the scheme is that the shape of the distribution of the population in the search space is converging with respect to size and direction towards areas with high fitness. The closer the population gets to the global optimum, the more the distribution will shrink and therefore reinforce the generation of smaller difference vectors.

As a general advice regarding the choice of NP, F and CR, Storn et al. (2006) state the following: Set the number of parents NP to 10 times the number of parameters, select differential weighting factor F = 0.8 and crossover constant CR = 0.9. Make sure that you initialize your parameter vectors by exploiting their full numerical range, i.e., if a parameter is allowed to exhibit values in the range [-100, 100] it is a good idea to pick the initial values from this range instead of unnecessarily restricting diversity. If you experience misconvergence in the optimization process you usually have to increase the value for NP, but often you only have to adjust F to be a little lower or higher than 0.8. If you increase NP and simultaneously lower F a little, convergence is more likely to occur but generally takes longer, i.e., DE is getting more robust (there is always a convergence speed/robustness trade-off).

DE is much more sensitive to the choice of F than it is to the choice of CR. CR is more like a fine tuning element. High values of CR like CR = 1 give faster convergence if convergence occurs. Sometimes, however, you have to go down as much as CR = 0 to make DE robust enough for a particular problem. For more details on the DE strategy, we refer the reader to Storn and Price (1997) and Price et al. (2006).

### See Also

DEoptim.control for control arguments, DEoptim-[**methods**](http://inside-r.org/r-doc/utils/methods) for methods on [DEoptim](http://inside-r.org/packages/cran/DEoptim) objects, including some examples in plotting the results; **[optim](http://inside-r.org/r-doc/stats/optim)** or **[constrOptim](http://inside-r.org/r-doc/stats/constrOptim)** for alternative optimization algorithms.

### Examples

*## Rosenbrock Banana function*

*## The function has a global minimum f(x) = 0 at the point (1,1).*

*## Note that the vector of parameters to be optimized must be the first*

*## argument of the objective function passed to DEoptim.*

Rosenbrock <- [**function**](http://inside-r.org/r-doc/base/function)(x){

x1 <- x[1]

x2 <- x[2]

100 \* (x2 - x1 \* x1)^2 + (1 - x1)^2

}

*## DEoptim searches for minima of the objective function between*

*## lower and upper bounds on each parameter to be optimized. Therefore*

*## in the call to DEoptim we specify vectors that comprise the*

*## lower and upper bounds; these vectors are the same length as the*

*## parameter vector.*

lower <- [**c**](http://inside-r.org/r-doc/base/c)(-10,-10)

upper <- -lower

*## run DEoptim and set a seed first for replicability*

[**set.seed**](http://inside-r.org/r-doc/base/set.seed)(1234)

DEoptim(Rosenbrock, lower, upper)

*## increase the population size*

DEoptim(Rosenbrock, lower, upper, DEoptim.control(NP = 100))

*## change other settings and store the output*

outDEoptim <- DEoptim(Rosenbrock, lower, upper, DEoptim.control(NP = 80,

itermax = 400, F = 1.2, CR = 0.7))

*## plot the output*

[**plot**](http://inside-r.org/r-doc/graphics/plot)(outDEoptim)

*## 'Wild' function, global minimum at about -15.81515*

Wild <- [**function**](http://inside-r.org/r-doc/base/function)(x)

10 \* [**sin**](http://inside-r.org/r-doc/base/sin)(0.3 \* x) \* [**sin**](http://inside-r.org/r-doc/base/sin)(1.3 \* x^2) +

0.00001 \* x^4 + 0.2 \* x + 80

[**plot**](http://inside-r.org/r-doc/graphics/plot)(Wild, -50, 50, n = 1000, main = "'Wild function'")

outDEoptim <- DEoptim(Wild, lower = -50, upper = 50,

[**control**](http://inside-r.org/r-doc/boot/control) = DEoptim.control([**trace**](http://inside-r.org/r-doc/base/trace) = **FALSE**))

[**plot**](http://inside-r.org/r-doc/graphics/plot)(outDEoptim)

DEoptim(Wild, lower = -50, upper = 50,

[**control**](http://inside-r.org/r-doc/boot/control) = DEoptim.control(NP = 50))

*## The below examples shows how the call to DEoptim can be*

*## parallelized; see the sandbox directory in the source*

*## code for additional examples.*

*## Note that if your objective function requires packages to be*

*## loaded or has arguments supplied via \code{...}, these should be*

*## specified using the \code{packages} and \code{parVar} arguments*

*## in control.*

*## Not run:*

Genrose <- [**function**](http://inside-r.org/r-doc/base/function)(x) {

*## One generalization of the Rosenbrock banana valley function (n parameters)*

n <- [**length**](http://inside-r.org/r-doc/base/length)(x)

*## make it take some time ...*

[**Sys.sleep**](http://inside-r.org/r-doc/base/Sys.sleep)(.001)

1.0 + [**sum**](http://inside-r.org/r-doc/base/sum) (100 \* (x[-n]^2 - x[-1])^2 + (x[-1] - 1)^2)

}

*# get some run-time on simple problems*

maxIt <- 250

n <- 5

oneCore <- [**system.time**](http://inside-r.org/r-doc/base/system.time)( DEoptim(fn=Genrose, lower=[**rep**](http://inside-r.org/r-doc/base/rep)(-25, n), upper=[**rep**](http://inside-r.org/r-doc/base/rep)(25, n),

[**control**](http://inside-r.org/r-doc/boot/control)=[**list**](http://inside-r.org/r-doc/base/list)(NP=10\*n, itermax=maxIt)))

withParallel <- [**system.time**](http://inside-r.org/r-doc/base/system.time)( DEoptim(fn=Genrose, lower=[**rep**](http://inside-r.org/r-doc/base/rep)(-25, n), upper=[**rep**](http://inside-r.org/r-doc/base/rep)(25, n),

[**control**](http://inside-r.org/r-doc/boot/control)=[**list**](http://inside-r.org/r-doc/base/list)(NP=10\*n, itermax=maxIt, parallelType=1)))

*## Compare timings*

(oneCore)

(withParallel)

*## End(Not run)*

### Author(s)

David Ardia, Katharine Mullen [mullenkate@gmail.com](mailto://mullenkate@gmail.com), Brian Peterson and Joshua Ulrich.

|  |  |
| --- | --- |
| nlm {stats} | R Documentation |

## Non-Linear Minimization

### Description

This function carries out a minimization of the function f using a Newton-type algorithm. See the references for details.

### Usage

nlm(f, p, hessian = FALSE, typsize=rep(1, length(p)), fscale=1,

print.level = 0, ndigit=12, gradtol = 1e-6,

stepmax = max(1000 \* sqrt(sum((p/typsize)^2)), 1000),

steptol = 1e-6, iterlim = 100, check.analyticals = TRUE, ...)

### Arguments

|  |  |
| --- | --- |
| f | the function to be minimized. If the function value has an attribute called gradient or both gradient and hessian attributes, these will be used in the calculation of updated parameter values. Otherwise, numerical derivatives are used. [deriv](http://ugrad.stat.ubc.ca/R/library/stats/html/deriv.html) returns a function with suitable gradient attribute. This should be a function of a vector of the length of p followed by any other arguments specified by the ... argument. |
| p | starting parameter values for the minimization. |
| hessian | if TRUE, the hessian of f at the minimum is returned. |
| typsize | an estimate of the size of each parameter at the minimum. |
| fscale | an estimate of the size of f at the minimum. |
| print.level | this argument determines the level of printing which is done during the minimization process. The default value of 0 means that no printing occurs, a value of 1 means that initial and final details are printed and a value of 2 means that full tracing information is printed. |
| ndigit | the number of significant digits in the function f. |
| gradtol | a positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in f in each direction p[i] divided by the relative change in p[i]. |
| stepmax | a positive scalar which gives the maximum allowable scaled step length. stepmax is used to prevent steps which would cause the optimization function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. stepmax would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step. |
| steptol | A positive scalar providing the minimum allowable relative step length. |
| iterlim | a positive integer specifying the maximum number of iterations to be performed before the program is terminated. |
| check.analyticals | a logical scalar specifying whether the analytic gradients and Hessians, if they are supplied, should be checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians. |
| ... | additional arguments to f. |

### Details

If a gradient or hessian is supplied but evaluates to the wrong mode or length, it will be ignored if check.analyticals = TRUE (the default) with a warning. The hessian is not even checked unless the gradient is present and passes the sanity checks.

From the three methods available in the original source, we always use method “1” which is line search.

### Value

A list containing the following components:

|  |  |
| --- | --- |
| minimum | the value of the estimated minimum of f. |
| estimate | the point at which the minimum value of f is obtained. |
| gradient | the gradient at the estimated minimum of f. |
| hessian | the hessian at the estimated minimum of f (if requested). |
| code | an integer indicating why the optimization process terminated.  1:  relative gradient is close to zero, current iterate is probably solution.  2:  successive iterates within tolerance, current iterate is probably solution.  3:  last global step failed to locate a point lower than estimate. Either estimate is an approximate local minimum of the function or steptol is too small.  4:  iteration limit exceeded.  5:  maximum step size stepmax exceeded five consecutive times. Either the function is unbounded below, becomes asymptotic to a finite value from above in some direction or stepmax is too small. |
| iterations | the number of iterations performed. |

### References

Dennis, J. E. and Schnabel, R. B. (1983) Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Englewood Cliffs, NJ.

Schnabel, R. B., Koontz, J. E. and Weiss, B. E. (1985) A modular system of algorithms for unconstrained minimization. ACM Trans. Math. Software, **11**, 419–440.

### See Also

[optim](http://ugrad.stat.ubc.ca/R/library/stats/html/optim.html) and [nlminb](http://ugrad.stat.ubc.ca/R/library/stats/html/nlminb.html).

[constrOptim](http://ugrad.stat.ubc.ca/R/library/stats/html/constrOptim.html) for constrained optimization, [optimize](http://ugrad.stat.ubc.ca/R/library/stats/html/optimize.html) for one-dimensional minimization and [uniroot](http://ugrad.stat.ubc.ca/R/library/stats/html/uniroot.html) for root finding. [deriv](http://ugrad.stat.ubc.ca/R/library/stats/html/deriv.html) to calculate analytical derivatives.

For nonlinear regression, [nls](http://ugrad.stat.ubc.ca/R/library/stats/html/nls.html) may be better.

### Examples

f <- function(x) sum((x-1:length(x))^2)

nlm(f, c(10,10))

nlm(f, c(10,10), print.level = 2)

str(nlm(f, c(5), hessian = TRUE))

f <- function(x, a) sum((x-a)^2)

nlm(f, c(10,10), a=c(3,5))

f <- function(x, a)

{

res <- sum((x-a)^2)

attr(res, "gradient") <- 2\*(x-a)

res

}

nlm(f, c(10,10), a=c(3,5))

## more examples, including the use of derivatives.

## Not run: demo(nlm)

# A comparison of some heuristic optimization methods

Posted on [2012/07/23](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/) by [Pat](http://www.portfolioprobe.com/author/pat/)

A simple [portfolio optimization](http://www.portfolioprobe.com/2012/01/05/the-top-7-portfolio-optimization-problems/) problem is used to look at several R functions that use randomness in various ways to do optimization.

## Orientation

Some optimization problems are really hard. In these cases sometimes the best approach is to use randomness to get an approximate answer.

Once you decide to go down this route, you need to decide on two things:

* how to formulate the problem you want to solve
* what algorithm to use

These decisions should seldom be taken independently. The best algorithm may well depend on the formulation, and how you formulate the problem may well depend on the algorithm you use.

The heuristic algorithms we will look at mostly fall into three broad categories:

* simulated annealing
* traditional genetic algorithm
* evolutionary algorithms

Genetic algorithms and evolutionary algorithms are really the same thing, but have different ideas about specifics.

The [Portfolio Probe optimization algorithm](http://www.portfolioprobe.com/features/computing-engine/) is a blend of these traditions plus additional techniques.

## The test case

The problem is to maximize a mean-variance utility where the universe is 10 assets and we have the constraints that the portfolio is long-only (weights must be non-negative), the weights must sum to 1, and there can be at most 5 assets in the portfolio.

In terms of portfolio optimization this is a tiny and overly trivial problem.  Portfolio Probe solves this problem consistently to 6 decimal places in about the same time as the algorithms tested here.

Actually there are two problems.  The variance matrix is the same in both but there are two expected return vectors. In one the optimal answer contains only 3 assets so the integer constraint of at most 5 is non-binding.  In the other case the integer constraint is binding.

### Formulation

What we really want is a vector of length 10 with non-negative numbers that sum to 1 and at most 5 positive numbers. The tricky part is how to specify which five of the ten are to be allowed to be positive.

The solution used here is to optimize a vector that is twice as long as the weight vector — 20 in this case.  The second half of the vector holds the weights (which are not normalized to sum to 1).  The first half of the vector holds numbers that order the assets by their desirability to be in the portfolio.  So the assets with the five largest numbers in this first half are allowed to have positive weights.

The first half of the solution vector tells us which assets are to be included in the portfolio.  Then the weight vector is prepared: it is extracted from the solution vector, the weights for assets outside the portfolio are set to zero, and the weights are normalized to sum to 1.

The original intention was that all the numbers in the solution vector should be between 0 and 1.  However, not all of the optimizers support such constraints.  The constraint of being less than 1 is purely arbitrary anyway.  We’ll see an interesting result related to this.

## The optimizers

Here are the R packages or functions that appear. If you are looking for optimization routines in R, then have a look at the [optimization task view](http://cran.r-project.org/web/views/Optimization.html).

### Rmalschains package

The [Rmalschains package](http://cran.r-project.org/web/packages/Rmalschains/index.html" \t "_blank) has the malschains function.  The name stands for “memetic algorithm with local search chains”.  I haven’t looked but I suspect it has substantial similarities with genopt.

### GenSA package

The [GenSA package](http://cran.r-project.org/web/packages/GenSA/index.html" \t "_blank) implements a generalized simulating annealing.

### genopt function

The genopt function is the horse that I have in the race. It is not in a package, but you can source the [genopt.R](http://www.burns-stat.com/pages/Freecode/genopt.R" \t "_blank) file to use it. You can get a sense of how to use it from [S Poetry](http://www.burns-stat.com/pages/Spoetry/Spoetry.pdf). The line of thinking that went into it can be found in [“An Introduction to Genetic Algorithms”](http://www.burns-stat.com/pages/Tutor/genetic.html).

### DEoptim package

The [DEoptim package](http://cran.r-project.org/web/packages/DEoptim/index.html" \t "_blank) implements a differential evolutionary algorithm.

### soma package

The [soma package](http://cran.r-project.org/web/packages/soma/index.html) gives us a self-organizing migrating algorithm.

### rgenoud package

The [rgenoud package](http://cran.r-project.org/web/packages/rgenoud/index.html" \t "_blank) implements an algorithm that combines a genetic algorithm and derivative-based optimization.

### GA package

The [GA package](http://cran.r-project.org/web/packages/GA/index.html) is a reasonably complete implementation in the realm of genetic algorithms.

### NMOF package

The [NMOF package](http://cran.r-project.org/web/packages/NMOF/index.html) contains a set of functions that are introductory examples of various algorithms. This package is support for the book [Numerical Methods and Optimization in Finance](http://www.portfolioprobe.com/2011/10/27/introduction-to-numerical-methods-and-optimization-in-finance/).

The optimizers that this package contributes to the race are:

* DEopt — another implementation of the differential evolutionary algorithm
* LSopt — local stochastic search, which is very much like simulated annealing
* TAopt — threshold accepting algorithm, another relative of simulated annealing

### SANN method of optim function

optim(method="SANN", ...) does a simulated annealing optimization.

## The results

Each optimizer was run 100 times on each of the two problems.  The computational time and the utility achieved was recorded for each run.  One or more control parameters were adjusted so that the typical run took about a second on my machine (which is about 3 years old and running Windows 7).

The figures show the difference in utilities between the runs and the optimal solution as found by Portfolio Probe.  The optimizers are sorted by the median deficiency.

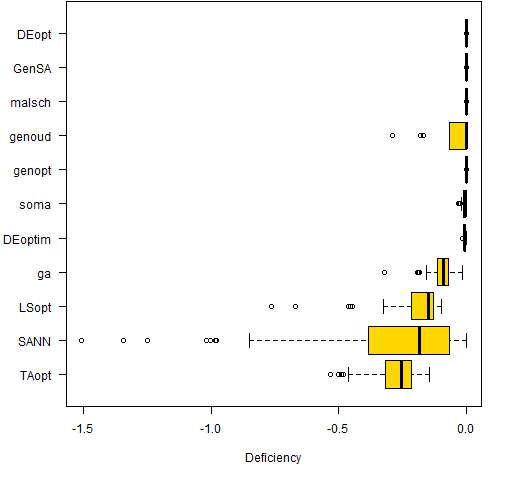
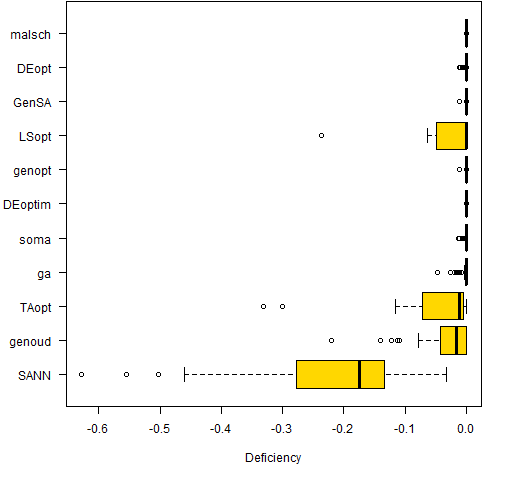
Figure 1: Difference in utility from optimal for all optimizers on the non-binding problem.[](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/lacking10a/)

Figure 2: Difference in utility from optimal for all optimizers on the binding problem.[](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/lacking10s/)We can characterize the results as: evolutionary better than genetic better than simulated annealing.  With one big exception.  GenSA — which hails from simulated annealing land — does very well.

I’m guessing that genoud would have done better if the differentiation were applied only to the weights and not the first part of the solution vector.

The other thing of note is that DEoptim is a more robustly developed version of differential evolution than is DEopt.  However, DEopt outperforms DEoptim.  DEoptdoes not have box constraints, so its solution vectors grow in size as the algorithm progresses.  This seems to make the problem easier. A weakness of DEopt turned out to be a strength.

Figures 3 and 4 show the results for the six best optimizers — same picture, different scale.

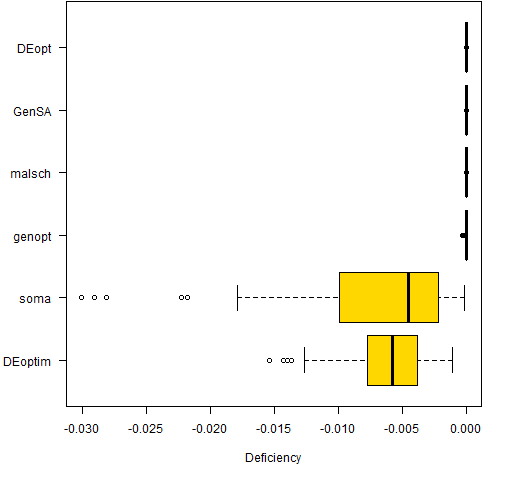
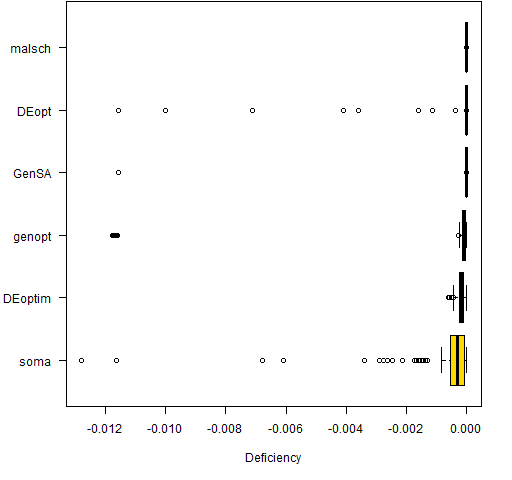
Figure 3: Difference in utility from optimal for the best optimizers on the non-binding problem.[](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/lacking10ax/)

Figure 4: Difference in utility from optimal for the best optimizers on the binding problem.[](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/lacking10sx/)

## Update 2012/07/26

This update shows an advantage of heuristic algorithms that I was hoping I wouldn’t teach.

*Randomization, for better or worse, often compensates for bugs.*

– Jon Bentley More Programming Pearls (page 32)

Even though the code was not doing anything close to its intended behavior, the algorithms still managed to move towards the optimum.

Luca  Scrucca spotted that I used order when I meant rank.  I have re-run the race with the new version.  There are two changes in the new race:

* the right code makes it easier for the optimizers
* the new code is slower, so the optimizers get fewer evalations

I adjusted control arguments so that about a second on my machine would be used for each run.  Since rank is significantly slower than order in this case (see [“R Inferno-ism: order is not rank”](http://www.portfolioprobe.com/2012/07/26/r-inferno-ism-order-is-not-rank/)), only about one-quarter to one-third as many evaluations were allowed.

(By the way, I’m rather suspicious of the timings — they seem to jump around a bit too much.  It is a Windows machine.)

The new pictures are in Figures 5 and 6.

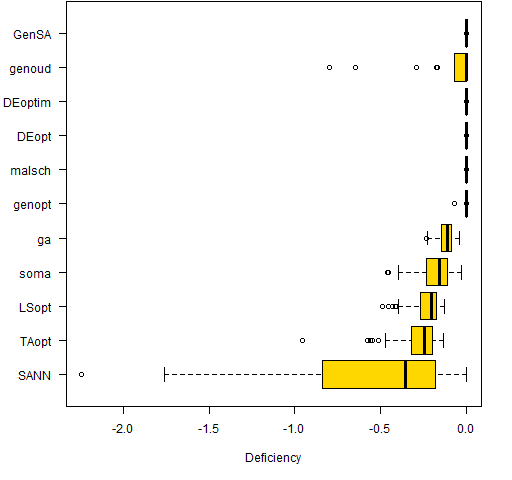
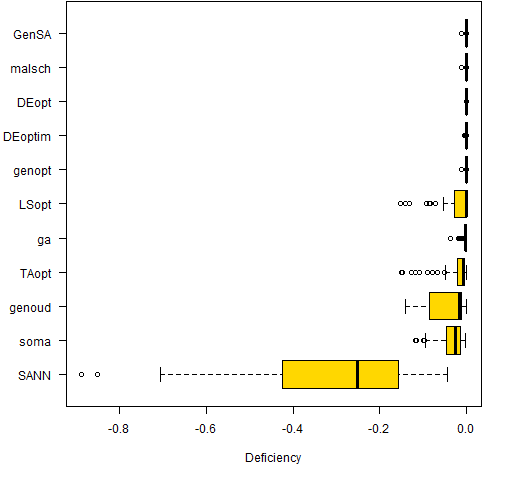
Figure 5: Difference in utility from optimal for all optimizers on the non-binding problem with the revised code. [](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/lacking10arev-2/)

Figure 6: Difference in utility from optimal for all optimizers on the binding problem with the revised code. [](http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/lacking10srev/)

The results that look good in Figures 5 and 6 generally look good under a microscope as well — there are a lot of results that are essentially perfect.

The revised functions are in [heuristicfuns\_rev.R](http://www.portfolioprobe.com/R/blog/heuristicfuns_rev.R" \t "_blank) while the results from the second race are [testresults10rev.R](http://www.portfolioprobe.com/R/blog/testresults10rev.R) and the original results are [testresults10.R](http://www.portfolioprobe.com/R/blog/testresults10.R).

## Caveat

You should never (ever) think that you understand the merits of optimizers from one example.

I have no doubt that very different results could be obtained by modifying the control parameters of the optimizers.  In particular the results are highly dependent on the time allowed.  Some optimizers will be good at getting approximately right but not good at homing in on the exact solution — these will look good when little time is allowed.  Other algorithms will be slow to start but precise once they are in the neighborhood — these will look good when a lot of time is allowed.

For genetic and evolutionary algorithms there is a big interaction between time allowed and population size.  A small population will get a rough approximation fast.  A large population will optimize much slower but (generally) achieve a better answer in the end.

Exact circumstances are quite important regarding which optimizer is best.

## Summary

If your problem is anything like this problem, then the Rmalschains and GenSApackages are worth test driving.

## See also

* (**update 2012 August 20**) [Another comparison of heuristic optimizers](http://www.portfolioprobe.com/2012/08/20/another-comparison-of-heuristic-optimizers/)

## Appendix R

The functions that ran the optimizers plus the code and data for the problems are in [heuristic\_objs.R](http://www.portfolioprobe.com/R/blog/heuristic_objs.R" \t "_blank).  (The 10a problem is non-binding and 10s is binding.)

The objective achieved by Portfolio for the non-binding problem is -0.38146061845033 and for the binding problem it is -1.389656885372.

In case you want to test routines on these problems outside R: the variance is in[variance10.csv](http://www.portfolioprobe.com/R/blog/variance10.csv) and the two expected return vectors are in [expectedreturns10.csv](http://www.portfolioprobe.com/R/blog/expectedreturns10.csv).