

# Package: CLA (via r-universe)

August 29, 2024

**Version** 0.96-3

**Date** 2024-07-29

**Title** Critical Line Algorithm in Pure R

**Depends** R (>= 3.6.0)

**Imports** stats, grDevices, graphics, utils

**Suggests** fGarch, FRAPO, Matrix, sfsmisc

**Description** Implements 'Markowitz' Critical Line Algorithm ('CLA') for classical mean-variance portfolio optimization, see Markowitz (1952) <[doi:10.2307/2975974](https://doi.org/10.2307/2975974)>. Care has been taken for correctness in light of previous buggy implementations.

**License** GPL (>= 3) | file LICENSE

**Encoding** UTF-8

**URL** <https://gitlab.math.ethz.ch/maechler/CLA/>

**NeedsCompilation** no

**Author** Yanhao Shi [aut], Martin Maechler [aut, cre]  
(<<https://orcid.org/0000-0002-8685-9910>>)

**Maintainer** Martin Maechler <[maechler@stat.math.ethz.ch](mailto:maechler@stat.math.ethz.ch)>

**Date/Publication** 2024-07-29 15:50:09 UTC

**Repository** <https://mmaechler.r-universe.dev>

**RemoteUrl** <https://github.com/cran/CLA>

**RemoteRef** HEAD

**RemoteSha** 2d6f7fbc571427e008aac28afc0be70643586b9a

## Contents

CLA . . . . .	2
findMu . . . . .	4
findSig . . . . .	5
MS . . . . .	7
muS.10ex . . . . .	8

muS.sp500 . . . . .	9
muSigmaGarch . . . . .	10
plot.CLA . . . . .	11

<b>Index</b>	<b>13</b>
--------------	-----------

CLA *Critical Line Algorithm for mean-variance optimal portfolio*

## Description

The Critical Line Algorithm was first proposed by Markowitz(1987) to solve the mean-variance optimal portfolio problem.

We solve the problem with “box” constraints, i.e., allow to specify lower and upper bounds (via `lB` and `uB`) for each asset weight.

Here we provide a pure R implementation, quite fine tuned and debugged compared to earlier ones.

## Usage

```
CLA(mu, covar, lB, uB,
    check.cov = TRUE, check.f = TRUE,
    tol.lambda = 1e-07,
    give.MS = TRUE, keep.names = TRUE, trace = 0)
```

## Arguments

<code>mu</code>	numeric vector of length $n$ containing the expected return $E[R_i]$ for $i = 1, 2, \dots, n$ .
<code>covar</code>	the $n \times n$ covariance matrix of the returns, must be positive definite.
<code>lB, uB</code>	vectors of length $n$ with lower and upper bounds for the asset weights.
<code>check.cov</code>	<b>logical</b> indicating if the covar matrix should be checked to be positive definite.
<code>check.f</code>	<b>logical</b> indicating if a warning should be produced when the algorithm cannot produce a new (smaller) lambda even though there are still free weights to be chosen.
<code>tol.lambda</code>	the tolerance when checking for lambda changes or being zero.
<code>give.MS</code>	<b>logical</b> indicating if <code>MS()</code> should be computed (and returned) as well.
<code>keep.names</code>	<b>logical</b> indicating if the <code>weights_set</code> matrix should keep the (asset) names( <code>mu</code> ).
<code>trace</code>	an integer (or <b>logical</b> ) indicating if and how much diagnostic or progress output should be produced.

## Details

The current implementation of the CLA is based (via Norring’s) on Bailey et al.(2013). We have found buglets in that implementation which lead them to introduce their “purge” routines (`purgeNumErr`, `purgeExcess`), which are no longer necessary.

Even though this is a pure R implementation, the algorithm is quite fast also when the number of assets  $n$  is large (1000s), though that depends quite a bit on the exact problem.

**Value**

an object of class "CLA" which is a list with components

weights_set	a $n \times m$ matrix of asset weights, corresponding to the $m$ steps that the CLA has completed or the $m$ "turning points" it has computed.
free_indices	a list of length $m$ , the $k$ -th component with the indices in $1, \dots, n$ of those assets whose weights were not at the boundary after ...
gammas	numeric vector of length $m$ of the values $\gamma_k$ for CLA step $k$ , $k = 1, \dots, m$ .
lambdas	numeric vector of length $m$ of the Lagrange parameters $\lambda_k$ for CLA step $k$ , $k = 1, \dots, m$ .
MS_weights	the $\mu(W)$ and $\sigma(W)$ corresponding to the asset weights weights_set, i.e., simply the same as MS(weights_set = weights_set, mu = mu, covar = covar).

**Note**

The exact results of the algorithm, e.g., the assets with non-zero weights, may slightly depend on the (computer) platform, e.g., for the S&P 500 example, differences between 64-bit or 32-bit, version of BLAS or Lapack libraries etc, do have an influence, see the R script 'tests/SP500-ex.R' in the package sources.

**Author(s)**

Alexander Norring did the very first version (unpublished master thesis). Current implementation: Yanhao Shi and Martin Maechler

**References**

- Markowitz, H. (1952) Portfolio selection, *The Journal of Finance* **7**, 77–91; doi:10.2307/2975974.
- Markowitz, H. M. (1987, 1st ed.) and Markowitz, H. M. and Todd, P. G. (2000) *Mean-Variance Analysis in Portfolio Choice and Capital Markets*; chapters 7 and 13.
- Niedermayer, A. and Niedermayer, D. (2010) Applying Markowitz's Critical Line Algorithm, in J. B. Guerard (ed.), *Handbook of Portfolio Construction*, Springer; chapter 12, 383–400; doi:10.1007/9780387774398\_12.
- Bailey, D. H. and López de Prado, M. (2013) An open-source implementation of the critical-line algorithm for portfolio optimization, *Algorithms* **6**(1), 169–196; doi:10.3390/a6010169,
- Yanhao Shi (2017) Implementation and applications of critical line algorithm for portfolio optimization; unpublished Master's thesis, ETH Zurich.

**See Also**

MS; for plotting CLA results: plot.CLA.

## Examples

```

data(muS.sp500)
## Full data taking too much time for example
set.seed(47)
iS <- sample.int(length(muS.sp500$mu), 24)

CLsp.24 <- CLA(muS.sp500$mu[iS], muS.sp500$covar[iS, iS], lB=0, uB=1/10)
CLsp.24 # using the print() method for class "CLA"

plot(CLsp.24)

if(require(Matrix)) { ## visualize how weights change "along turning points"
  show(image(Matrix(CLsp.24$weights_set, sparse=TRUE),
    main = "CLA(muS.sp500 <random_sample(size=24)>) $ weights_set",
    xlab = "turning point", ylab = "asset number"))
}

## A 3x3 example (from real data) where CLA()'s original version failed
## and 'check.f = TRUE' produces a warning :
mc3 <- list(
  mu = c(0.0408, 0.102, -0.023),
  cv = matrix(c(0.00648, 0.00792, 0.00473,
    0.00792, 0.0334, 0.0121,
    0.00473, 0.0121, 0.0793), 3, 3,
    dimnames = list(NULL,
      paste0(c("TLT", "VTI", "GLD"), ".Adjusted"))))

rc3 <- with(mc3, CLA(mu=mu, covar=cv, lB=0, uB=1, trace=TRUE))

```

---

findMu

*Find  $\mu(W)$  and  $W$ , given  $\sigma(W)$  and CLA result*

---

## Description

Find  $\mu(W)$  and  $W$ , given  $\sigma(W)$  and CLA result.

## Usage

```
findMu(Sig0, result, covar, tol.unir = 1e-06, equal.tol = 1e-06)
```

## Arguments

Sig0	numeric vector of $\sigma(W)$ values.
result	a <b>list</b> with components MS_weight and weights_set as resulting from CLA().
covar	the same $n \times n$ covariance matrix (of asset returns) as the argument of CLA().
tol.unir	numeric tolerance passed to <b>uniroot</b> .
equal.tol	numeric tolerance to be used in <b>all.equal</b> (..., tolerance = equal.tol) in the check to see if the $\mu$ of two neighbouring turning points are equal.

**Value**

a [list](#) with components

Mu                    numeric vector of same length, say  $M$ , as  $\text{Sig}\theta$ .  
 weight                numeric  $n \times M$  matrix of weights.

**References**

Master thesis, p.33

**See Also**

[findSig](#), [CLA](#), [MS](#).

**Examples**

```
data(muS.sp500)
## Full data taking too much time for example
if(getRversion() >= "3.6") .Rk <- RNGversion("3.5.0") # for back compatibility & warning
set.seed(2016)
iS <- sample.int(length(muS.sp500$mu), 17)
if(getRversion() >= "3.6") do.call(RNGkind, as.list(.Rk)) # revert
cov17 <- muS.sp500$covar[iS, iS]
CLsp.17 <- CLA(muS.sp500$mu[iS], covar=cov17, lb=0, uB = 1/2)
CLsp.17 # 16 turning points
summary(tpS <- CLsp.17$MS_weights[, "Sig"])
str(s0 <- seq(0.0186, 0.0477, by = 0.0001))
mu.. <- findMu(s0, result=CLsp.17, covar=cov17)
str(mu..)
stopifnot(dim(mu..$weight) == c(17, length(s0)))
plot(s0, mu..$Mu, xlab=quote(sigma), ylab = quote(mu),
     type = "o", cex = 1/4)
points(CLsp.17$MS_weights, col = "tomato", cex = 1.5)
```

---

 findSig

---

*Find  $\sigma(W)$  and  $W$ , given  $\mu(W)$  and CLA result*


---

**Description**

Find  $\sigma(W)$  and  $W$ , given  $\mu(W)$  and [CLA](#) result.

**Usage**

```
findSig(Mu0, result, covar, equal.tol)
```

**Arguments**

<code>Mu0</code>	numeric vector of $\mu(W)$ values.
<code>result</code>	a <code>list</code> with components <code>MS_weight</code> and <code>weights_set</code> as resulting from <code>CLA()</code> .
<code>covar</code>	the same $n \times n$ covariance matrix (of asset returns) as the argument of <code>CLA()</code> .
<code>equal.tol</code>	numeric tolerance to be used in <code>all.equal(..., tolerance = equal.tol)</code> in the check to see if the $\mu$ of two neighbouring turning points are equal.

**Value**

a `list` with components

<code>Sig</code>	numeric vector of same length, say $M$ , as <code>Mu0</code> .
<code>weight</code>	numeric $n \times M$ matrix of weights.

**References**

Master thesis, p.33

**See Also**

`findMu`, `CLA`, `MS`.

**Examples**

```
data(muS.sp500)
## Full data taking too much time for example: Subset of n=21:
if(getRversion() >= "3.6") .Rk <- RNGversion("3.5.0") # for back compatibility & warning
set.seed(2018)
iS <- sample.int(length(muS.sp500$mu), 21)
if(getRversion() >= "3.6") do.call(RNGkind, as.list(.Rk)) # revert
cov21 <- muS.sp500$covar[iS, iS]
CLsp.21 <- CLA(muS.sp500$mu[iS], covar=cov21, lB=0, uB = 1/2)
CLsp.21 # 14 turning points
summary(tpM <- CLsp.21$MS_weights[, "Mu"])
str(m0 <- c(min(tpM), seq(0.00205, 0.00525, by = 0.00005), max(tpM)))
sig. <- findSig(m0, result=CLsp.21, covar=cov21)
str(sig.)
stopifnot(dim(sig.$weight) == c(21, length(m0)))
plot(sig.$Sig, m0, xlab=quote(sigma), ylab = quote(mu),
      type = "o", cex = 1/4)
points(CLsp.21$MS_weights, col = "tomato", cex = 1.5)
title("Efficient Frontier from CLA()")
mtext("findSig() to interpolate between turning points", side=3)
```

---

MS *Means (Mu) and Standard Deviations (Sigma) of the “Turning Points” from CLA*

---

### Description

Compute the vectors of means ( $\mu_i$ ) and standard deviations ( $\sigma_i$ ), for all the turning points of a [CLA](#) result.

### Usage

```
MS(weights_set, mu, covar)
```

### Arguments

`weights_set` numeric matrix ( $n \times m$ ) of optimal asset weights  $W = (w_1, w_2, \dots, w_m)$ , as resulting from [CLA\(\)](#).

`mu` expected (log) returns (identical to *argument* of [CLA\(\)](#)).

`covar` covariance matrix of (log) returns (identical to *argument* of [CLA\(\)](#)).

### Details

These are trivially computable from the [CLA\(\)](#)'s result. To correctly *interpolate* this, “hyperbolic” interpolation is needed, provided by the [findSig](#) and [findMu](#) functions.

### Value

a [list](#) with components

`Sig` numeric vector of length  $m$  of standard deviations,  $\sigma(W)$ .

`Mu` numeric vector of length  $m$  of means  $\mu(W)$ .

### Author(s)

Yanhao Shi

### See Also

[CLA](#).

### Examples

```
## The function is quite simply
MS
## and really an auxiliary function for CLA().

## TODO: add small (~12 assets) example
```

---

muS.10ex

*10 Assets Example Data from Markowitz & Todd*


---

### Description

The simple example Data of Markowitz and Todd (2000); used for illustrating the CLA; reused in Bailey and López de Prado (2013).

### Usage

```
data("muS.10ex")
```

### Format

A list with two components,

**mu** Named num [1:10] 1.175 1.19 0.396 1.12 0.346 ...

names : chr [1:10] "X1" "X2" "X3" "X4" ...

**covar** num [1:10, 1:10] 0.4076 0.0318 0.0518 0.0566 0.033 ...

### Source

From 'http://www.quantresearch.info/CLA\_Data.csv.txt' (URL no longer working, Aug.2020!) by López de Prado.

### References

Markowitz, H. M. (1987, 1st ed.) and Markowitz, H. M. and Todd, P. G. (2000) *Mean-Variance Analysis in Portfolio Choice and Capital Markets*, page 335.

Bailey, D. H. and López de Prado, M. (2013) An open-source implementation of the critical-line algorithm for portfolio optimization, *Algorithms* **6**(1), 169–196; doi:10.3390/a6010169, p. 16f.

### Examples

```
data(muS.10ex)
```

```
str(muS.10ex)
```

```
CLA.10ex <- with(muS.10ex, CLA(mu, covar, lB=0, uB=1))
```

```
if(require("Matrix"))
```

```
  drop0(zapsmall(CLA.10ex$weights_set))
```

```
## The results, summarized, as in Bayley and López de Prado (Table 2, p.18) :
```

```
with(CLA.10ex, round(cbind(MS_weights[,2:1], lambda=lambdas, t(weights_set)), 3))
```

```
CLA.10ex.1c <- with(muS.10ex, CLA(mu, covar, lB=1/100, uB=1))
```

```
round(CLA.10ex.1c$weights_set, 3)
```



---

muS.sp500

*Return Expectation and Covariance for "FRAPO"s SP500 data*


---

### Description

If  $R_{j,t}$  are the basically the scale standardized log returns for  $j = 1, 2, \dots, 476$  of 476 stocks from S&P 500, as from [SP500](#), then  $\mu_j = E[R_{j,*}]$  somehow averaged over time; actually as predicted by `muSigma()` at the end of the time period, and  $\Sigma_{j,k} = Cov(R_j, R_k)$  are estimated covariances.

These are the main “inputs” needed for the CLA algorithm, see [CLA](#).

### Usage

```
data("muS.sp500")
```

### Format

A list with two components,

```
mu Named num [1:476] 0.00233 0.0035 0.01209 0.00322 0.00249 ...
     names : chr [1:476] "A" "AA" "AAPL" "ABC" ...
covar num [1:476, 1:476] 0.001498 0.000531 0.000536 ...
```

### Source

It is as simple as this:

```
data(SP500, package="FRAPO")
system.time(muS.sp500 <- muSigmaGarch(SP500)) # 26 sec. (lynne, 2017)
```

### See Also

[muSigmaGarch\(\)](#) which was used to construct it.

### Examples

```
data(muS.sp500)
str(muS.sp500)
```

---

muSigmaGarch

*Compute (mu, Sigma) for a Set of Assets via GARCH fit*


---

### Description

Compute (mu, Sigma) for a set of assets via a GARCH fit to each individual asset, using package **fGarch**'s `garchFit()`.

### Usage

```
muSigmaGarch(x, formula = ~garch(1, 1), cond.dist = "std", trace = FALSE,
             ...)
```

### Arguments

<code>x</code>	numeric matrix or data frame ( $T \times d$ ) of log returns of $d$ assets, observed on a common set of $T$ time points.
<code>formula</code>	optional formula for <code>garchFit</code> .
<code>cond.dist</code>	the conditional distribution to be used for the garch process.
<code>trace</code>	logical indicating if some progress of <code>garchFit()</code> should printed to the console.
<code>...</code>	optional arguments to <code>cor</code> , i.e., use or method.

### Value

a list with components

<code>mu</code>	numeric vector of length $n$ of mean returns ( $= E[R_i]$ ).
<code>covar</code>	covariance matrix ( $n \times n$ ) of the returns.

### See Also

[muS.sp500](#) which has been produced via `muSigmaGarch`. [CLA](#) which needs (mu, covar) as crucial input.

### Examples

```
if(requireNamespace("FRAPO")) {
  data(NASDAQ, package = "FRAPO")
  ## 12 randomly picked stocks from NASDAQ data
  iS <- if(FALSE) { ## created (w/ warning, in new R) by
    RNGversion("3.5.0"); set.seed(17); iS <- sample(ncol(NASDAQ), 12)
  } else c(341L, 2126L, 1028L, 1704L, 895L, 1181L, 454L, 410L, 1707L, 425L, 950L, 5L)
  X. <- NASDAQ[, iS]
  muSig <- muSigmaGarch(X.)
  stopifnot(identical(names(muSig$mu), names(NASDAQ)[iS]),
            identical(dim(muSig$covar), c(12L,12L)),
            all.equal(unname(muSig$mu),
```

```

    c( 7.97, -4.05, -14,    21.5, -5.36, -15.3,
      -15.9, 11.8,  -1.64, -14,   3.13, 121) / 10000,
    tol = 0.0015))
}

```

plot.CLA

*Plotting CLA() results including Efficient Frontier***Description**

A partly experimental `plot()` method for `CLA()` objects.

It draws the efficient frontier in the  $\mu_w, \sigma_w$  aka (mean, std.dev.) plane.

Currently, this is quite rudimentary.

Future improvements would allow - to add the/some single asset points, - to correctly ('hyperbolically') interpolate between turning points - add text about the number of (unique) critical points - add option `add = FALSE` which when `TRUE` would use `lines` instead `plot`.

**Usage**

```

## S3 method for class 'CLA'
plot(x, type = "o", main = "Efficient Frontier",
     xlab = expression(sigma(w)),
     ylab = expression(mu(w)),
     col = adjustcolor("blue", alpha.f = 0.5),
     pch = 16, ...)

```

**Arguments**

<code>x</code>	a named <code>list</code> as resulting from <code>CLA()</code> .
<code>type</code>	the <code>lines/points</code> types used for the efficient frontier. This will become more sophisticated, i.e., <i>may change non-compatibly!!</i>
<code>main</code>	main <code>title</code> .
<code>xlab, ylab</code>	x- and y- axis labels, passed to <code>plot.default</code> .
<code>col, pch</code>	color and point type, passed to <code>plot.default</code> , but with differing defaults in this method.
<code>...</code>	potentially further arguments passed to <code>plot</code> , i.e., <code>plot.default</code> .

**Author(s)**

Martin Maechler.

**See Also**

`CLA`, `plot.default`.

**Examples**

```
## TODO %% Add A. Norring's small 12-asset example see --> ../TODO  
## ---- one example is in help(CLA)
```

# Index

- \* **arith**
  - CLA, [2](#)
  - findMu, [4](#)
  - findSig, [5](#)
  - MS, [7](#)
- \* **datasets**
  - muS.10ex, [8](#)
  - muS.sp500, [9](#)
- \* **dplot**
  - findSig, [5](#)
- \* **hplot**
  - plot.CLA, [11](#)
- \* **multivariate**
  - muSigmaGarch, [10](#)
- \* **optimize**
  - CLA, [2](#)
  - findMu, [4](#)

all.equal, [4, 6](#)

CLA, [2, 4–7, 9–11](#)

class, [3](#)

cor, [10](#)

findMu, [4, 6, 7](#)

findSig, [5, 5, 7](#)

garchFit, [10](#)

lines, [11](#)

list, [3–7, 11](#)

logical, [2](#)

MS, [2, 3, 5, 6, 7](#)

muS.10ex, [8](#)

muS.sp500, [9, 10](#)

muSigmaGarch, [9, 10](#)

plot, [11](#)

plot.CLA, [3, 11](#)

plot.default, [11](#)

points, [11](#)

SP500, [9](#)

title, [11](#)

uniroot, [4](#)