

# Package ‘StableEstim’

July 2, 2014

**Type** Package

**Title** Estimate the 4 parameters of stable law using different methods

**Version** 2.0

**Date** 2013-11-20

**Depends** R(>= 2.10.0),methods,Matrix,stats,utils,stabledist,testthat

**Imports** numDeriv,xtable,fBasics,MASS

**Author** Tarak Kharrat, Georgi N. Boshnakov

**Maintainer** Tarak Kharrat <tarak.kharrat@manchester.ac.uk>

**Description** Compute the maximum-likelihood(ML), generalised method of moment with finite and continuum number of points, the iterative Koutrouvelis regression and the Kogon-McCulloch methods to estimate the 4 parameters of stable law. The asymptotic properties of the estimator (covariance matrix, confidence intervals) are also provided.

**License** GPL (>= 2)

**Collate** cte.R ExternalPackageInterface.R ToolsFct.R Interpolation.R  
RegularInverse.R stableCF.R CFbasedMoment.R WeightingMatrix.R  
eCFfirstZero.R tSchemes.R MultiDimIntegral.R InitialGuess.R  
KoutParamsEstim.R MLParamsEstim.R GMMParamsEstim.R  
CgmmParamsEstim.R OutputFileManip.R CheckPoint.R BestT\_Class.R  
Estim\_Class.R Estim.R Simulation.R BestT.R

**LazyLoad** yes

**LazyData** yes

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2014-02-24 11:19:02

**R topics documented:**

StableEstim-package	2
Best_t-class	4
CF Jacobian	5
CgmmParametersEstim	6
ComplexCF	9
ComputeBest_t	10
ComputeBest_tau	10
ComputeDuration	11
ComputeFirstRootRealeCF	12
ConcatFiles	13
Estim	14
Estim-class	15
Estim_Simulation	16
expect_almost_equal	18
get.abMat	19
get.StatFcts	19
getTime_	20
GMMParametersEstim	21
IGParametersEstim	25
IntegrateRandomVectorsProduct	26
KoutParametersEstim	27
McCullochParametersEstim	29
MLParametersEstim	30
PrintDuration	31
PrintEstimatedRemainingTime	32
RegularisedSol	32
sampleComplexCFMoment	34
sampleRealCFMoment	35
StatFcts	36
StatObjectFromFiles	37
TexSummary	38
<b>Index</b>	<b>40</b>

---

StableEstim-package    *stable law estimation functions*

---

**Description**

A collection of methods to estimate the 4 parameters of the stable law. The package also provides functions to compute the characteristic function and tools to run Monte Carlo simulations.

## Details

The main functions of the package are briefly described below:

**main function:** `Estim` is the most useful function of the package which provides the estimated parameters and the asymptotic properties of the estimator.

**estimation function:** the methods provided so far are the maximum-likelihood (`MLParametersEstim`), the generalised method of moment with a finite (`GMMParametersEstim`) or continuum moment conditions (`CgmmParametersEstim`), the iterative Koutrouvelis regression method (`KoutParametersEstim`) and the fast Kogon-McCulloch method used for first guess estimation (`IGParametersEstim`).

**characteristic function:** the characteristic function (`ComplexCF`) and its Jacobian (`jacobianComplexCF`) can be computed and will return a vector (respectively a matrix) of complex numbers.

**Monte Carlo simulation** a tool to run a Monte Carlo simulation (`Estim_Simulation`) is provided and can save output files and/or produce statistical summary.

## Note

This version (2) of the package implements a 2D interpolation routine (and hence doesn't call package 'akima' anymore). Therefore, **StableEstim** is now under GPL license. The package is related to upcoming work by the authors where the different methods are compared using MC simulations.

## Author(s)

Tarak Kharrat, Georgi N. Boshnakov

## References

- Carrasco M and Florens J (2000). "Generalization of GMM to a continuum of moment conditions." *Econometric Theory*, **16**(06), pp. 797–834.
- Carrasco M and Florens J (2002). "Efficient GMM estimation using the empirical characteristic function." *IDEI Working Paper*, **140**.
- Carrasco M and Florens J (2003). "On the asymptotic efficiency of GMM." *IDEI Working Paper*, **173**.
- Carrasco M, Chernov M, Florens J and Ghysels E (2007). "Efficient estimation of general dynamic models with a continuum of moment conditions." *Journal of Econometrics*, **140**(2), pp. 529–573.
- Carrasco M, Florens J and Renault E (2007). "Linear inverse problems in structural econometrics estimation based on spectral decomposition and regularization." *Handbook of econometrics*, **6**, pp. 5633–5751.
- Carrasco M and Kotchoni R (2010). "Efficient estimation using the characteristic function." Mimeo. University of Montreal.
- Nolan J (2001). "Maximum likelihood estimation and diagnostics for stable distributions." *L'evy processes: theory and applications*, pp. 379–400.
- Nolan JP (2013). *Stable Distributions - Models for Heavy Tailed Data*. Birkhauser, Boston. In progress, Chapter 1 online at [academic2.american.edu/~sim\\$jp/nolan](http://academic2.american.edu/~sim$jp/nolan).
- Hansen LP (1982). "Large sample properties of generalized method of moments estimators." *Econometrica: Journal of the Econometric Society*, pp. 1029–1054.

Hansen LP, Heaton J and Yaron A (1996). "Finite-sample properties of some alternative GMM estimators." *Journal of Business & Economic Statistics*, **14**(3), pp. 262–280.

Feuerverger A and McDunnough P (1981). "On efficient inference in symmetric stable laws and processes." *Statistics and Related Topics*, **99**, pp. 109–112.

Feuerverger A and McDunnough P (1981). "On some Fourier methods for inference." *Journal of the American Statistical Association*, **76**(374), pp. 379–387.

Schmidt P (1982). "An improved version of the Quandt-Ramsey MGF estimator for mixtures of normal distributions and switching regressions." *Econometrica: Journal of the Econometric Society*, pp. 501–516.

Besbeas P and Morgan B (2008). "Improved estimation of the stable laws." *Statistics and Computing*, **18**(2), pp. 219–231.

### See Also

`.mleStableFit`, `.qStableFit`, `stabledist`

---

Best_t-class	Class "Best_t"
--------------	----------------

---

### Description

Class used to store the result of function [ComputeBest\\_t](#)

### Objects from the Class

Objects can be created by calls of the form `new("Best_t", theta, nbt, tvec, detVal, convcode, ...)` where user can specify some/all of the inputs or call function [ComputeBest\\_t](#).

### Slots

`theta`: Object of class "vector"; value of the 4 parameters.

`nbt`: Object of class "vector"; number of points used in the minimisation.

`tvec`: Object of class "list"; value of the best t-vectors.

`detVal`: Object of class "vector"; value of the optimal determinant found after minimisation.

`convcode` Convergence code.

### Methods

`+` signature(`e1 = "Best_t"`, `e2 = "Best_t"`): sum 2 objects from class `Best_t`

**initialize** signature(`.Object = "Best_t"`): initialise an object from class `Best_t` as described above.

**show** signature(`object = "Best_t"`): print a summary of the object.

### See Also

[ComputeBest\\_t](#)

---

 CF Jacobian

*Jacobian of the characteristic function of the stable law.*


---

### Description

Numeric jacobian of the characteristic function (CF) as a function of the parameter  $\theta$  evaluated at a specific point(vector)  $t$  and a given value  $\theta$ .

### Usage

```
jacobianComplexCF(t, theta, pm = 0)
```

### Arguments

<code>t</code>	Vector of (real) numbers where the jacobian of the CF is evaluated ;numeric
<code>theta</code>	Given value of the parameters where the jacobian of the CF is evaluated; vector of length 4.
<code>pm</code>	Parametrisation, an integer (0 or 1); default: <code>pm=0</code> ( the Nolan 'S0' parametrisation).

### Details

The numerical derivation is obtained by a call to the function `jacobian` from the package **numDeriv**. We have set up its arguments by default and the user is not allowed to modify them.

### Value

Returns a matrix  $\text{length}(t) \times 4$  of complex numbers.

### See Also

[ComplexCF](#)

### Examples

```
# define the parameters
nt <- 10
t <- seq(0.1, 3, length.out=nt)
theta <- c(1.5, 0.5, 1, 0)
pm <- 0

# Compute the jacobian of the characteristic function
jack_CF <- jacobianComplexCF(t=t, theta=theta, pm=pm)
```

---

CgmmParametersEstim    *The generalised method of moment with a continuum of moment conditions*

---

### Description

Cgmm method of Carrasco and Florens to estimate the 4 parameters of stable law based on a continuum of complex moment conditions (the modulus). Those moments are computed by matching the characteristic function with its sample counterpart. The resulting (ill-posed) estimation problem is solved by a regularisation technique.

### Usage

```
CgmmParametersEstim(x, type = c("2S", "IT", "Cue"), alphaReg = 0.01,
  subdivisions=50,
  IntegrationMethod = c("Uniform", "Simpson"),
  randomIntegrationLaw = c("unif", "norm"),
  s_min=0,s_max = 1,
  theta0 = NULL,
  IterationControl = list(),
  pm = 0, PrintTime = FALSE,...)
```

### Arguments

x	Data used to perform the estimation: a vector of length n.
type	Cgmm algorithm: "2S" is the two steps GMM proposed by Hansen(1982) and the "Cue" and "IT" are respectively the continuous updated and the iterative GMM proposed by Hansen, Eaton et Yaron (1996) and adapted to the continuum case.
alphaReg	Value of the regularisation parameter; numeric, default=0.01
subdivisions	Number of subdivision used to compute the different integrals involved in the computation of the objective function (to minimise); numeric
IntegrationMethod	Numerical integration method to be used to approximate the (vectorial) integrals. User can choose between the "Uniform" discretization or the "Simpson"'s rule (the 3-point Newton-Cotes quadrature rule).
randomIntegrationLaw	Probability measure associated to the Hilbert space spanned by the moment conditions. See Carrasco and Florens (2003) for more details.
s_min,s_max	Lower and Upper bounds of the interval where the moment conditions are considered; numeric
theta0	Initial guess for the 4 parameters values: vector of length 4
IterationControl	Only used with type="IT" or type="Cue" to control the iterations. See details

pm	Parametrisation, an integer (0 or 1); default: pm=0 (the Nolan 'S0' parametrisation).
PrintTime	Logical flag; if set to TRUE, the estimation duration is printed out to the screen in a readable format (h/min/sec).
...	Other arguments to be passed to the optimisation function and/or to the integration function.

## Details

### The moment conditions

The moment conditions are given by:

$$g_t(X, \theta) = g(t, X; \theta) = e^{itX} - \phi_\theta(t)$$

If one has a sample  $x_1, \dots, x_n$  of i.i.d realisations of the same random variable  $X$ , then:

$$\hat{g}_n(t, \theta) = \frac{1}{n} \sum_{i=1}^n g(t, x_i; \theta) = \phi_n(t) - \phi_\theta(t)$$

where  $\phi_n(t)$  is the eCF associated to the sample  $x_1, \dots, x_n$  and defined by  $\phi_n(t) = \frac{1}{n} \sum_{j=1}^n e^{itX_j}$

### Objective function

Following Carrasco and al (Proposition 3.4, 2007), the objective function to minimise is given by:

$$obj(\theta) = \bar{v}'(\theta) [\alpha_{Reg} \mathcal{I}_n + C^2]^{-1} \underline{v}(\theta)$$

where:

$$\underline{v} = [v_1, \dots, v_n]'; \quad v_i(\theta) = \int_I \bar{g}_i(t; \hat{\theta}_n^1) \hat{g}(t; \theta) \pi(t) dt$$

$I_n$  is the identity matrix of size  $n$

$$C \text{ is a } n \times n \text{ matrix with (i,j) elements given by } c_{ij} = \frac{1}{n-4} \int_I \bar{g}_i(t; \hat{\theta}_n^1) g_j(t; \hat{\theta}_n^1) \pi(t) dt$$

To compute  $C$  and  $v_i()$  we will use the function [IntegrateRandomVectorsProduct](#).

### The IterationControl

If type="IT" or type="Cue" is selected, user can control each iteration by setting up the list IterationControl which contains the following elements:

**NbIter:** maximum number of iteration. The loop stops when NBIter is reached; default=10

**PrintIterlogical:** if set to TRUE the value of the current parameter estimation is printed to the screen at each iteration; default=TRUE

**RelativeErrMax:** the loop stops if the relative error between two consecutive estimation is smaller than RelativeErrMax; default=1e-3

## Value

Returns a list with the following elements:

Estim	output of the optimisation function
duration	estimation duration in numerical format
method	character describing the method used

**Note**

nlmminb as used to minimise the Cgmm objective function.

**References**

Carrasco M and Florens J (2000). “Generalization of GMM to a continuum of moment conditions.” *Econometric Theory*, **16**(06), pp. 797–834.

Carrasco M and Florens J (2002). “Efficient GMM estimation using the empirical characteristic function.” *IDEI Working Paper*, **140**.

Carrasco M and Florens J (2003). “On the asymptotic efficiency of GMM.” *IDEI Working Paper*, **173**.

Carrasco M, Chernov M, Florens J and Ghysels E (2007). “Efficient estimation of general dynamic models with a continuum of moment conditions.” *Journal of Econometrics*, **140**(2), pp. 529–573.

Carrasco M and Katchoni R (2010). “Efficient estimation using the characteristic function.” Mimeo. University of Montreal.

**See Also**

[Estim](#), [IntegrateRandomVectorsProduct](#)

**Examples**

```
## general inputs
theta <- c(1.45,0.55,1,0)
pm <- 0
set.seed(2345);x <- rstable(50,theta[1],theta[2],theta[3],theta[4],pm)

## GMM specific params
alphaReg=0.01
subdivisions=20
randomIntegrationLaw="unif"
IntegrationMethod="Uniform"

## Estimation
twoS <- CgmmParametersEstim(x=x,type="2S",alphaReg=alphaReg,
                             subdivisions=subdivisions,
                             IntegrationMethod=IntegrationMethod,
                             randomIntegrationLaw=randomIntegrationLaw,
                             s_min=0,s_max=1,theta0=NULL,
                             pm=pm,PrintTime=TRUE)
```



---

`ComplexCF`*The characteristic function of the stable law*

---

**Description**

Theoretical characteristic function (CF) of stable law under parametrisation 'S0' or 'S1'. See Nolan (2013) for more details.

**Usage**

```
ComplexCF(t, theta, pm = 0)
```

**Arguments**

<code>t</code>	Vector of (real) numbers where the CF is evaluated ; numeric
<code>theta</code>	Vector of parameters of the stable law; vector of length 4.
<code>pm</code>	Parametrisation, an integer (0 or 1); default: <code>pm=0</code> ( the Nolan 'S0' parametrisation).

**Details**

For more details about the different parametrisation of the CF, see *Nolan(2013)*.

**Value**

Vector of complex numbers with dimension `length(t)`.

**References**

Nolan JP (2013). *Stable Distributions - Models for Heavy Tailed Data*. Birkhauser, Boston. In progress, Chapter 1 online at [academic2.american.edu/~sim\\$jp/nolan](http://academic2.american.edu/~sim$jp/nolan).

**See Also**

[jacobianComplexCF](#)

**Examples**

```
# define the parameters
nt <- 10
t <- seq(0.1,3,length.out=nt)
theta <- c(1.5,0.5,1,0)
pm <- 0

# Compute the characteristic function
CF <- ComplexCF(t=t,theta=theta,pm=pm)
```

---

ComputeBest_t	<i>Monte Carlo Simulation to investigate the optimal number of points to use in the moment conditions.</i>
---------------	--

---

### Description

Runs Monte Carlo Simulation for different values of  $\alpha$  and  $\beta$  and computes a specified number of t-points that minimises the determinant of the asymptotic covariance matrix.

### Usage

```
ComputeBest_t(AlphaBetaMatrix = abMat, nb_ts = seq(10, 100, 10),
              alphaReg = 0.001, FastOptim=TRUE,...)
```

### Arguments

AlphaBetaMatrix	Values of the parameter $\alpha$ and $\beta$ from which we simulate the data. By default, The values of $\gamma$ and $\delta$ are set to 1 and 0 respectively; matrix $2 \times n$
nb_ts	Vector of number of t-points to use for the minimisation; default seq(10, 100, 10).
alphaReg	Value of the regularisation parameter; numeric, default=0.001
FastOptim	Logical flag; if set to TRUE, optim with "Nelder-Mead" method is used (fast but not accurate). Otherwise, nlm is used (more accurate but slower).
...	Other arguments to pass to the optimisation function.

### Value

Returns a list containing slots from class [Best\\_t-class](#) corresponding to one value of the parameters  $\alpha$  and  $\beta$ .

### See Also

[ComputeBest\\_tau](#), [Best\\_t-class](#)

---

ComputeBest_tau	<i>Runs Monte Carlo Simulation to investigate the optimal <math>\tau</math>.</i>
-----------------	--

---

### Description

Runs Monte Carlo Simulation to investigate the optimal number of points to use when one of the reduced spacing scheme is considered.

**Usage**

```
ComputeBest_tau(AlphaBetaMatrix = abMat, nb_ts = seq(10, 100, 10),
                tScheme = c("uniformOpt", "ArithOpt"),
                Constrained = TRUE, alphaReg = 0.001, ...)
```

**Arguments**

AlphaBetaMatrix	Values of the parameter $\alpha$ and $\beta$ from which we simulate the data. By default, The values of $\gamma$ and $\delta$ are set to 1 and 0 respectively; matrix $2 \times n$
nb_ts	Vector of number of t-points to use for the minimisation; default seq(10, 100, 10).
tScheme	Scheme used to select the points where the moment conditions are evaluated. User can choose between "uniformOpt" (uniform optimal placement) and "ArithOpt" (arithmetic optimal placement). See function <a href="#">GMMPParametersEstim</a>
Constrained	Logical flag: if set to True lower and upper bands will be computed as discussed function <a href="#">GMMPParametersEstim</a> .
alphaReg	Value of the regularisation parameter; numeric, default=0.001.
...	Other arguments to pass to the optimisation function.

**Value**

Returns a list containing slots from class [Best\\_t-class](#) corresponding to one value of the parameters  $\alpha$  and  $\beta$ .

**See Also**

[ComputeBest\\_t,Best\\_t-class](#)

ComputeDuration      *Duration*

**Description**

Compute the duration between 2 time points.

**Usage**

```
ComputeDuration(t_init, t_final, OneNumber = FALSE)
```

**Arguments**

t_init	Starting time; numeric
t_final	Final time; numeric
OneNumber	Logical flag; if set to True, the duration in seconds will be returned. Otherwise, a vector of length 3 will be computed representing the time in h/min/sec.

**Value**

Returns a numeric of length 1 or 3 depending on the value of OneNumber flag.

**See Also**

[PrintDuration](#), [PrintEstimatedRemainingTime](#).

**Examples**

```
ti=getTime_()
for (i in 1:100) x=i*22.1
tf=getTime_()
duration=ComputeDuration(ti,tf)
```

---

ComputeFirstRootRealeCF

*first root of the empirical characteristic function.*

---

**Description**

Computes the first root of the real part of the empirical characteristic function.

**Usage**

```
ComputeFirstRootRealeCF(x, ..., tol = 0.001, maxIter = 100,
  lowerBand = 1e-04, upperBand = 30)
```

**Arguments**

x	Data used to perform the estimation: vector of length n.
...	Other arguments to pass to the optimisation function .
tol	Tolerance to accept the solution; default=1e-3
maxIter	maximum number of iteration in the Welsh algorithm; default=100
lowerBand	Lower band of the domain where the graphical seach is performed; default=1e-4
upperBand	Lower band of the domain where the graphical seach is performed; default=30

**Details**

The Welsh algorithm is first applied. If it fails to provide a satisfactory value ( $< tol$ ), a graphical/numerical approach is used. We first plot the real part of the eCF vs t in order to determine the first zero directly and use it as the initial guess of a numerical minimisation routine.

**Value**

numeric: first zero of the real part of the eCF.

**References**

Welsh A (1986). "Implementing empirical characteristic function procedures." *Statistics & probability letters*, 4(2), pp. 65–67.

**See Also**

[ComplexCF](#)

**Examples**

```
set.seed(345); x <- rstable(500,1.5,0.5)
ComputeFirstRootRealeCF(x)
```

---

ConcatFiles	<i>Concatenates output files.</i>
-------------	-----------------------------------

---

**Description**

Creates a unique file by concatenating several output files associated to one set of parameters.

**Usage**

```
ConcatFiles(files, sep_ = ",", outfile, headers_ = TRUE,
            DeleteIfExists=TRUE)
```

**Arguments**

files	character Vector containing the files name to be concatenated. See details.
sep_	Field separator character to be used in function <code>read.csv()</code> and <code>write.table()</code> . Values on each line of the file are separated by this character; It can also be a vector character (same length as files) if different separators are used for each file; default: ","
outfile	Name of the output file; character
headers_	Vector of boolean of length 1 or same length as files to indicate for each file if the header argument is to be considered or not. To be passed to function <code>read.csv()</code> .
DeleteIfExists	if outfile exists, it will be deleted and recreated (over-written).

**Details**

The files to be concatenated should be related to the same set of parameters alpha and beta. The function stops if one of the file contains 2 (or more) different set of parameters (the function compares the values of columns 1 and 2 row by row) or if the set of parameters within one file is different from the one from other files.

**Value**

Returns an output file `outfile` saved in the working directory.

**See Also**

[Estim\\_Simulation](#)

---

Estim	<i>Main estimation function</i>
-------	---------------------------------

---

**Description**

Main estimation function which computes all the information about the estimator (and its asymptotic properties). It allows the user to choose the preferred method and several related options.

**Usage**

```
Estim(EstimMethod = c("ML", "GMM", "Cgmm", "Kout"), data, theta0 = NULL,
      ComputeCov = FALSE, HandleError = TRUE, ...)
```

**Arguments**

EstimMethod	Estimation method to be used. User can choose between ML, GMM, Cgmm or Koutrouvelis regression method.
data	Data used to perform the estimation: vector of length <code>n</code> .
theta0	Initial guess for the 4 parameters values: If NULL the Kogon-McCulloch method is called, see <a href="#">IGParametersEstim</a> ; vector of length 4.
ComputeCov	Logical flag: If set to TRUE, the asymptotic covariance matrix (4x4) is computed (except for the Koutrouvelis method).
HandleError	Logical flag: If set to TRUE and if an error occurs during the estimation procedure, the computation will carry on and NA will be returned. Useful for Monte Carlo simulations, see <a href="#">Estim_Simulation</a>
...	Other arguments to be passed to the estimation function or the asymptotic confidence level. See details.

**Details**

This function should be used in priority for estimation purpose as it provides more information about the estimator. However, user needs to pass the appropriate parameters to the selected method in `...`. See the documentation of the selected method.

**Asymptotic Confidence Interval:**

The *normal* asymptotic confidence interval (C.I) are computed. The user can set the *level* of confidence by inputting the `level` argument (in the `...`); default `level=0.95`. The theoretical justification for asymptotic normal C.I could be find in the references of each method. Note the C.I are not computed for the Koutrouvelis regression method.

**Value**

Returns an object of class `Estim`. See [Estim](#) for more details.

**See Also**

[CgmmParametersEstim](#), [GMMParametersEstim](#), [MLParametersEstim](#), [KoutParametersEstim](#)

**Examples**

```
## general inputs
theta <- c(1.45,0.55,1,0)
pm <- 0
set.seed(2345)
x <- rstable(200,theta[1],theta[2],theta[3],theta[4],pm)

objKout <- Estim(EstimMethod="Kout",data=x,pm=pm,
                ComputeCov=FALSE,HandleError=FALSE,
                spacing="Kout")
```

---

Estim-class

Class "Estim"

---

**Description**

Class storing all the information about the estimation method; output of function `Estim`

**Objects from the Class**

Objects can be created by calls of the form `new("Estim", par, ...)`. User can provide some/all of the inputs stated below to create an object of this class or call function [Estim](#) with appropriate arguments.

**Slots**

`par`: Object of class "numeric"; Value of the 4 estimated parameters.  
`par0`: Object of class "numeric"; Initial guess for the 4 parameters.  
`vcov`: Object of class "matrix" with 4 rows and 4 columns representing the covariance matrix  
`confint`: Object of class "matrix" with 4 rows and 2 columns representing the confidence interval computed at a specific level (attribute of the object).  
`data`: Object of class "numeric" used to compute the estimation.  
`sampleSize`: Object of class "numeric" ; length of the data  
`others`: Object of class "list" ; more information about the estimation method  
`duration`: Object of class "numeric" ; duration in seconds  
`failure`: Object of class "numeric" representing the status of the procedure: 0 failure or 1 success  
`method`: Object of class "character" description of the parameter used in the estimation.

**Methods**

- initialize** signature(.Object = "Estim"): creates an object of this class using the inputs described above
- show** signature(object = "Estim"): summarised print of the object.

**See Also**

[Estim](#)

---

Estim\_Simulation      *Monte Carlo Simulation*

---

**Description**

Runs Monte Carlo simulation for a selected estimation method. The function can save a file and produce a statistical summary.

**Usage**

```
Estim_Simulation(AlphaBetaMatrix = abMat, SampleSizes = c(200, 1600),
                 MCparam = 100, Estimfct = c("ML", "GMM", "Cgmm", "Kout"),
                 HandleError = TRUE, FctsToApply = StatFcts,
                 saveOutput = TRUE, StatSummary = FALSE,
                 CheckMat=TRUE, tolFailCheck=tolFailure,
                 SeedOptions=NULL,...)
```

**Arguments**

AlphaBetaMatrix	Values of the parameter $\alpha$ and $\beta$ from which we simulate the data. By default, The values of $\gamma$ and $\delta$ are set to 1 and 0 respectively; matrix $2 \times n$
SampleSizes	Sample sizes to be used to simulate the data. By default, we use 200 (small sample size) and 1600 (large sample size); vector of integer.
MCparam	Number of Monte Carlo simulation for each couple of parameter, default=100; integer
Estimfct	The estimation function to be used. User can choose between ML, GMM, Cgmm or Kout.
HandleError	Logical flag: if set to TRUE, the simulation doesn't stop when an error in the estimation function is encountered. A vector of (size 4) NA is saved and the simulation carries on. See details.
FctsToApply	Functions used to produce the statistical summary. See details ; vector of character.
saveOutput	Logical flag: if set to TRUE, a csv file (for each couple of parameter $\alpha$ and $\beta$ ) with the estimation information is saved in the current directory. See details
StatSummary	Logical flag: if set to TRUE, a statistical summary (using FctsToApply) is returned. See details.



CheckMat	Logical flag: if set to TRUE, an estimation is declared failed if the squared error of the estimation is larger than tolFailCheck; default TRUE
tolFailCheck	Tolerance on the squared error of the estimation to be declared failed; default 1.5
SeedOptions	List to control the seed generation. See details.
...	Other arguments to be passed to the estimation function.

## Details

### Error Handling

It is advisable to set it to TRUE when user is planning to launch long simulations as it will prevent the procedure to stop if an error occurs for one sample data. The estimation function will produce a vector of NA as estimated parameters related to this (error generating) sample data and move on to the next Monte Carlo step.

### Statistical summary

The function is able to produce a statistical summary of the Monte Carlo simulation for each parameter (slices of the list). Each slice is a matrix where the rows represents the true values of the parameters and the columns the statistical information.

In all cases, the following quantities are computed:

**sample size:** the sample size used to produce the simulated data

**alphaT, betaT:** the true values of the parameters

**failure:** the number of times the procedure failed to produce relevant estimation.

**time:** The average running time in seconds of the estimation procedure

Besides, the (vector of character) `FctsToApply` controls the other quantities to be computed by providing the name of the function object to be applied to the vector of estimated parameter. The signature of the function should be of the form `fctName=function(p,...){...}` where `p` is the vector (`length(p)=MCparam`) of parameter estimate and `...` is the extra arguments to be passed the function.

By default, the functions of the `StatFcts` will be applied but user can pass his own functions by providing their name ( in the `FctsToApply` vector and their definition in the global environment.

Note that if `CheckMat` is set to TRUE, the estimation is considered failed if the squared error (of the first 2 parameters `alpha` and `beta`) is larger than `tolFailCheck`. **Output file**

Setting `saveOutput` to TRUE will have the side effect of saving a csv file in the working directory. This file will have `MCparam*length(SampleSizes)` lines and its columns will be :

**alphaT, betaT:** the true value of the parameters.

**data size:** the sample size used to generate the simulated data.

**seed:** the seed value used to generate the simulated data.

**alphaE, betaE, gammaE, deltaE:** the estimate of the 4 parameters.

**failure:** binary: 0 for success, 1 for failure.

**time:** estimation running time in seconds.

The file name is informative to let the user identify the value of the true parameters, the MC parameters as well as the options selected for the estimation method.

The csv file is updated after each MC estimation which is useful when the simulation stops before it finishes. Besides, using the check-pointing mechanism explained below, the simulation can re-start from where it stopped.

*Check-pointing* Checkpointing is the act of saving enough program state and results so far calculated that a computation can be stopped and restarted. The way we did it here is to save a text file with some useful information about the state of the estimation. This text file is updated after each MC iteration and read at the beginning of function `Estim_Simulation` to allow the simulation to re-start from where it stopped. This file is deleted at the end of the simulation procedure.

*SeedOptions* If user does not want to control the seed generation, he could ignore this argument (default value NULL). This argument can be more useful when one wants to cut the simulation (even for one parameter value) into pieces. In that case, he can control which part of the seed vector he wants to use.

`MCtot`: total values of MC simulations in the entire process.

`seedStart`: starting index in the seed vector. The vector extracted will be of size `MCparam`.

## Value

If `StatSummary` is set to `TRUE`, a `list` with 4 slices (corresponding to the 4 parameters) is returned. Each slice is a matrix. If `SaveOutput` is set to `TRUE`, only a csv file is saved and nothing is returned (if `StatSummary` is `FALSE`). If both are `FALSE`, the function stops.

## See Also

[Estim](#), [CgmmParametersEstim](#), [GMMPParametersEstim](#), [MLParametersEstim](#)

---

`expect_almost_equal`    *test approximate equality*

---

## Description

tests the approximate equality of 2 objects. Useful for running tests.

## Usage

```
expect_almost_equal(x, y, tolExpect = 0.001)
```

## Arguments

<code>x</code>	First object.
<code>y</code>	Second object
<code>tolExpect</code>	Tolerance accepted; default=1e-3

**Details**

This function works with the `expect_that` function from package `testthat` to test equality between 2 objects with a given tolerance. It used particularly for testing functions output. See the CF examples in the Examples folder.

**See Also**

`expect_that`, `testthat`

**Examples**

```
x=1.1 ; y=1.5
expect_almost_equal(x,y,1) ## pass
#expect_almost_equal(x,y,0.3) -> fails
```

---

`get.abMat`

*Default set of parameters to pass to the [Estim\\_Simulation](#).*

---

**Description**

Default set of parameters to pass to the [Estim\\_Simulation](#) inspired by the one used by Koutrevelis (1980) in his simulation procedure.

**Usage**

```
abMat()
```

**Value**

Returns a 2-columns matrix containing a wide range of parameters  $\alpha$  and  $\beta$  covering the entire parameters space.

---

`get.StatFcts`

*Default functions used to produce the statistical summary.*

---

**Description**

Default functions used to produce the statistical summary in the Monte Carlo simulations.

**Usage**

```
get.StatFcts()
```

**Value**

The functions computed are :

**Mean** .mean <- function(p,...) mean(p)

**Min** .min <- function(p,...) min(p)

**Max** .max <- function(p,...) max(p)

**Sn** .Sn <- function(p,n,...) sqrt(n)\*sd(p)

**MSE** .MSE <- function(p,paramT,...) (1/length(p))\*sum((p-paramT)\*\*2)

**Std error** .st.err <- function(p,...) sd(p)/sqrt(length(p))

When user wants to change the statistical summary, he needs to provide function with similar signatures and pass the a character vector containing the function names to [Estim\\_Simulation](#).

---

getTime_	<i>read time.</i>
----------	-------------------

---

**Description**

Reads the time when the function is called.

**Usage**

```
getTime_()
```

**Value**

Returns a numeric.

**See Also**

[PrintDuration](#), [PrintEstimatedRemainingTime](#), [ComputeDuration](#)

**Examples**

```
ti=getTime_()
```

---

GMMPParametersEstim	<i>generalised method of moment with finite number of moment conditions</i>
---------------------	---

---

### Description

Generalised method of moment with finite number of moment conditions. It uses a regularisation technique to make the method more robust (when the number of moment condition is large) and allows different schemes to select where the moment conditions are computed.

### Usage

```
GMMPParametersEstim(x, algo = c("2SGMM", "ITGMM", "CueGMM"),
                    alphaReg = 0.01,
                    regularization = c("Tikhonov", "LF", "cut-off"),
                    WeightingMatrix = c("OptAsym", "DataVar", "Id"),
                    t_scheme = c("equally", "NonOptAr", "uniformOpt",
                                "ArithOpt", "VarOpt", "free"),
                    theta0 = NULL,
                    IterationControl = list(),
                    pm = 0, PrintTime = FALSE, ...)
```

### Arguments

x	Data used to perform the estimation: vector of length n.
algo	GMM algorithm: "2SGMM" is the two step GMM proposed by Hansen(1982) and the "CueGMM" and "ITGMM" are respectively the continuous updated and the iterative GMM proposed by Hansen, Eaton et Yaron (1996) and adapted to the continuum case.
alphaReg	Value of the regularisation parameter; numeric,default=0.01
regularization	Regularization scheme to be used. User can choose between "Tikhonov" scheme , "LF" (Landweber-Fridmann) and "cut-off"(spectral cut-off). See <a href="#">RegularisedSol</a> .
WeightingMatrix	Type of Weighting matrix used to compute the objective function. User can choose between "OptAsym" (the optimal asymptotic), "DataVar" (the data driven) and "Id" (the identity matrix). See details
t_scheme	Scheme used to select the points where the moment conditions are evaluated. User can choose between "equally" (equally placed), "NonOptAr" (non optimal arithmetic placement), "uniformOpt" (uniform optimal placement), "ArithOpt" (arithmetic optimal placement) , "Var Opt"(optimal variance placement) and "free" (user needs to pass his own set of point in the . . .). See details.
theta0	Initial guess for the 4 parameters values: If NULL, the Kogon-McCulloch method is called, see <a href="#">IGParametersEstim</a> ; vector of length 4.
IterationControl	Only used if type="IT" or type="Cue" to control the iterations. See details

pm	Parametrisation, an integer (0 or 1); default: pm=0 (the Nolan 'S0' parametrisation).
PrintTime	Logical flag; if set to TRUE, the estimation duration is printed out to the screen in a readable format (h/min/sec).
...	Other arguments to pass to the regularisation function, the optimisation function or the selection scheme (including the function that finds the first zero of the eCF). See details.

## Details

### The moment conditions

The moment conditions are given by:

$$g_t(X, \theta) = g(t, X; \theta) = e^{itX} - \phi_\theta(t)$$

If one has a sample  $x_1, \dots, x_n$  of i.i.d realisations of the same random variable  $X$ , then:

$$\hat{g}_n(t, \theta) = \frac{1}{n} \sum_{i=1}^n g(t, x_i; \theta) = \phi_n(t) - \phi_\theta(t)$$

where  $\phi_n(t)$  is the eCF associated to the sample  $x_1, \dots, x_n$  and defined by  $\phi_n(t) = \frac{1}{n} \sum_{j=1}^n e^{itX_j}$

### Objective function

$$obj\theta = \langle K^{-1/2} \hat{g}_n(\cdot; \theta), K^{-1/2} \hat{g}_n(\cdot; \theta) \rangle$$

where  $K^{-1}f$  denotes the solution  $\varphi$  (when it exists) of the equation  $K\varphi = f$  and  $K^{-1/2} = (K^{-1})^{1/2}$ . The optimal choice of the Weighting operator  $K$  (a matrix in the GMM case) and its estimation are discussed in Hansen (1982).

### Weighting operator (Matrix)

OptAsym: the optimal asymptotic choice as described by Hansen. The expression of the components of this matrix could be found for example in Feuerverger and McDunnough (1981b)

DataVar: the covariance matrix of the data provided

Id: the identity matrix

### the t-scheme

One of the most important feature of this method is that it allows the user to choose how to place the points where the moment conditions are evaluated. The general rule is that user can provide his own set of points (option "free") or choose one of the other scheme. In the latter case he *needs to specify the number of points* nb\_t in the ... argument and eventually the lower and upper limit (by setting Constrained to FALSE and providing min\_t and max\_t) in the non-optimised case. If one of the optimised case is selected, setting Constrained to FALSE will not constrain the choice of  $\tau$ , see below. We mean by optimised set of point, the set that minimises the (determinant) of the asymptotic covariance matrix as suggested by Schmidt (1982) and Besbeas and Morgan (2008).

6 options have been implemented:

"equally": equally placed point in  $[\min\_t, \max\_t]$ . When provided, user's  $\min\_t$  and  $\max\_t$  will be used (when `whenCoinstrained=FALSE`). Otherwise,  $\text{eps}$  and  $A_n$  will be used instead (where  $A_n$  is the first zero of the eCF)

"NonOptAr": non optimal arithmetic placement:  $t_j = \frac{j(j+1)}{nbt(nbt+1)}(\max - \text{eps}); j = 1, \dots, nbt$  where  $\max$  is the upper band of the set of points selected as discussed before.

"uniformOpt": uniform optimal placement:  $t_j = j\tau, j = 1, \dots, nbt$

"ArithOpt": arithmetic optimal placement:  $t_j = j(j+1)\tau, j = 1, \dots, nbt$

"Var Opt": optimal variance placement as explained above.

"free": user needs to pass his own set of point in the . . .

For the "ArithOpt" and "uniformOpt" schemes, the function to minimise is seen as a function of the real parameter  $\tau$  instead of doing a vectorial optimisation as in the "Var Opt" case. In the latter case, one can choose between a fast (but less accurate) optimisation routine or a slow (but more accurate) one by setting the `FastOptim` flag to the desired value.

### The IterationControl

If `type="IT"` or `type="Cue"` user can control each iteration by setting up the list `IterationControl` which contains the following elements:

`NbIter`: maximum number of iteration. The loop stops when `NbIter` is reached; default=10

`PrintIterlogical`: if set to `TRUE`, the value of the current parameter estimation is printed to the screen at each iteration; default=`TRUE`

`RelativeErrMax`: the loop stops if the relative error between two consecutive estimation is smaller than `RelativeErrMax`; default=`1e-3`

### Value

Returns a list with the following elements:

<code>Estim</code>	output of the optimisation function
<code>duration</code>	estimation duration in a numerical format
<code>method</code>	character describing the method used
<code>tEstim</code>	final set of points selected for the estimation. Only relevant when one of the optimisation scheme is selected.

### Note

`n1minb` was used for the minimisation of the GMM objective function and to compute  $\tau$  in the "uniformOpt" and "ArithOpt" schemes. In the "Var Opt" scheme, `optim` was preferred. All those routines have been selected after running different tests using the summary table produced by package **optimx** for comparing the performance of different optimisation methods.





---

IGParametersEstim      *Kogon regression method + McCulloch quantile method.*

---

### Description

Kogon regression method is used together with the McCulloch quantile method to provide initial estimates.

### Usage

```
IGParametersEstim(x,pm=0,...)
```

### Arguments

x	Data used to perform the estimation: vector of length n.
pm	Parametrisation, an integer (0 or 1); default: pm=0( the Nolan 'S0' parametrisation).
...	Other arguments. Currently not used

### Details

The parameters  $\gamma$  and  $\delta$  are estimated using the McCulloch(1986) quantile method from **fBasics**. The data is rescaled using those estimates and used to perform the Kogon regression method to estimate  $\alpha$  and  $\beta$ .

### Value

a vector of length 4 containing the estimate of the 4 parameters.

### References

Kogon SM and Williams DB (1998). "Characteristic function based estimation of stable distribution parameters." *A practical guide to heavy tailed data*, pp. 311–335.

McCulloch JH (1986). "Simple consistent estimators of stable distribution parameters." *Communications in Statistics-Simulation and Computation*, **15**(4), pp. 1109–1136.

### See Also

[Estim](#), [McCullochParametersEstim](#)

### Examples

```
x <- rstable(200,1.2,0.5,1,0,pm=0)
IGParametersEstim(x,pm=0)
```

---

IntegrateRandomVectorsProduct

*Integrate Ranndom vectors product*


---

### Description

Computes the integral of two random vector.

### Usage

```
IntegrateRandomVectorsProduct(f_fct, X, g_fct, Y, s_min, s_max,
                              subdivisions = 50,
                              method = c("Uniform", "Simpson"),
                              randomIntegrationLaw = c("norm", "unif"),
                              ...)
```

### Arguments

f_fct	Function object with signature <code>f_fct=function(s,X)</code> and returns a matrix $ns \times nx$ where $nx=length(X)$ and $ns=length(s)$ ; $s$ is the points where the integrand is evaluated.
X	Random vector where the function <code>f_fct</code> is evaluated. See details
g_fct	Function object with signature <code>g_fct=function(s,Y)</code> and returns a matrix $ns \times ny$ where $ny=length(Y)$ and $ns=length(s)$ ; $s$ is the points where the integrand is evaluated.
Y	Random vector where the function <code>g_fct</code> is evaluated. See details
s_min,s_max	Limits of integration. Should be finite
subdivisions	Maximum number of subintervals.
method	Numerical integration rule. User ca choose between "uniform" (fast) or "Simpson" (more accurate quadratic rule).
randomIntegrationLaw	Random law $\pi(s)$ to be applied to the Random product vector, see details. Choices are "unif" (uniform) and "norm" (normal distribution).
...	Other arguments to pass to random integration law. Mainly, the mean ( $\mu$ ) and standard deviation ( $\sigma$ ) of the normal law.

### Details

The function computes the  $nx \times ny$  matrix  $C = \int_{s_{min}}^{s_{max}} f_s(X)g_s(Y)\pi(s)ds$  such as the one used in the objective function of the Cgmm method.

There is no function in R to compute vectorial integration and computing  $C$  element by element using `integrate` may be very slow when  $length(X)$  (or  $length(y)$ ) is large.

The function allows complex vectors as its integrands.

**Value**

Returns a  $n_x \times n_y$  matrix  $C$  with elements:

$$c_{ij} = \int_{s_{min}}^{s_{max}} f_s(X_i) g_s(Y_j) \pi(s) ds$$

**Examples**

```
## Define the integrand
f_fct <- function(s,x){sapply(X=x,
                             FUN=sampleComplexCFMoment,
                             t=s,theta=theta)
}
f_bar_fct <- function(s,x){Conj(f_fct(s,x))}

## Function specific arguments
theta <- c(1.5,0.5,1,0)
set.seed(345);X=rstable(3,1.5,0.5,1,0)
s_min=0;s_max=2
numberIntegrationPoints=10
randomIntegrationLaw="norm"

Estim_Simpson<- IntegrateRandomVectorsProduct(f_fct,X,f_bar_fct,X,s_min,s_max,
                                             numberIntegrationPoints,
                                             "Simpson",randomIntegrationLaw)
```

---

KoutParametersEstim    *Iterative Koutrouvelis regression method*

---

**Description**

Iterative Koutrouvelis regression method with different spacing schemes (points where the eCF is computed).

**Usage**

```
KoutParametersEstim(x, theta0 = NULL,
                   spacing = c("Kout", "UniformSpac", "ArithSpac", "free"),
                   pm = 0, tol = 0.05, NbIter = 10, PrintTime = FALSE, ...)
```

**Arguments**

**x**                      Data used to perform the estimation: vector of length  $n$ .

**theta0**                Initial guess for the 4 parameters values: vector of length 4

spacing	Scheme used to select the points where the moment conditions are evaluated. Kout is the scheme suggested by Koutrouvelis, UniformSpac and ArithSpac are the uniform and arithmetic spacing schemes over the informative interval $[\epsilon, A_n]$ . If user choose free, he needs to provide a set of points t_points and u_points in . . . .
pm	Parametrisation, an integer (0 or 1); default: pm=0 (the Nolan 'S0' parametrisation).
tol	The loop stops if the relative error between two consecutive estimation is smaller than tol; default=0.05
NbIter	Maximum number of iteration. The loop stops when NbIter is reached; default=10
PrintTime	Logical flag; if set to TRUE, the estimation duration is printed out to the screen in a readable format (h/min/sec).
. . .	Other arguments to pass to the function. See details

### Details

#### spacing

4 options for the spacing scheme are implemented as described above. In particular:

UniformSpac, ArithSpac: user can specify the number of points to choose in both regression by inputting nb\_t and nb\_u. Otherwise the Koutrouvelis table will be used to compute them.

free: User is expected to provide t\_points and u\_points otherwise the Kout scheme will be used.

### Value

Returns a list with the following elements:

Estim	list containing the vector of 4 parameters estimate (par), the 2 regressions objects (reg1 and reg2) and the matrix of iterations estimate (vals).
duration	estimation duration in a numerical format
method	character describing the method used

### References

Koutrouvelis IA (1980). "Regression-type estimation of the parameters of stable laws." *Journal of the American Statistical Association*, **75**(372), pp. 918–928.

Koutrouvelis IA (1981). "An iterative procedure for the estimation of the parameters of stable laws: An iterative procedure for the estimation." *Communications in Statistics-Simulation and Computation*, **10**(1), pp. 17–28.

### See Also

[Estim](#)

## Examples

```
pm=0
theta <- c(1.45,0.5,1.1,0.4)
set.seed(1235);x <- rstable(200,theta[1],theta[2],theta[3],theta[4],pm=pm)
theta0=theta-0.1
spacing="Kout"

KoutParametersEstim(x=x,theta0=theta0,
                    spacing=spacing,pm=pm)
```

---

McCullochParametersEstim

*quantile-based method.*

---

## Description

McCulloch quantile-based method.

## Usage

```
McCullochParametersEstim(x)
```

## Arguments

x                      Data used to perform the estimation: vector of length n.

## Details

The code is a modified version of function `.qStableFit` from package **fBasics**.

## Value

numeric of length(4) representing the value of the 4 parameters.

## References

McCulloch JH (1986). "Simple consistent estimators of stable distribution parameters." *Communications in Statistics-Simulation and Computation*, **15**(4), pp. 1109–1136.

## See Also

[Estim,IGParametersEstim](#)

## Examples

```
set.seed(333);
x=rstable(500,1.3,0.4,1,0)
McCullochParametersEstim(x)
```

---

MLParametersEstim      *Maximum likelihood (ML) method.*

---

### Description

Uses the numerical ML approach described by Nolan to estimate the 4 parameters of stable law. The method may be slow for large sample size due to the use of numerical optimisation routine.

### Usage

```
MLParametersEstim(x, theta0 = NULL,
                  pm = 0, PrintTime = FALSE, ...)
```

### Arguments

x	Data used to perform the estimation: vector of length n.
theta0	Initial guess for the 4 parameters values: If NULL, the Kogon-McCulloch method is called, see <a href="#">IGParametersEstim</a> ; a vector of length 4.
pm	Parametrisation, an integer (0 or 1); default: pm=0 (the Nolan 'S0' parametrisation).
PrintTime	Logical flag; if set to TRUE, the estimation duration is printed out to the screen in a readable format (h/min/sec).
...	Other argument to be passed to the optimisation function.

### Details

The function performs the minimisation of the numerical (-)log-density of stable law computed by function `dstable` from the **stabledist** package.

After testing several optimisation routines, we have found out that the "L-BFGS-B" algorithm performs better with the ML method (faster, more accurate).

### Value

Returns a list with the following elements:

Estim	output of the optimisation function
duration	estimation duration in a numerical format
method	character describing the method used

### References

Nolan J (2001). "Maximum likelihood estimation and diagnostics for stable distributions." *L'evy processes: theory and applications*, pp. 379–400.

### See Also

[Estim](#)

**Examples**

```

theta <- c(1.5,0.4,1,0)
pm <- 0
## 50 points does not give accurate estimation
## but it makes estimation fast for installation purposes
## use at least 200 points to get decent results.
set.seed(1333);x <- rstable(50,theta[1],theta[2],theta[3],theta[4],pm)

## This example takes > 30 sec hence commented
##ML <- MLParametersEstim(x=x,pm=pm,PrintTime=TRUE)
## see the Examples folder for more examples.

```

---

PrintDuration	<i>print duration</i>
---------------	-----------------------

---

**Description**

print duration in human readable format.

**Usage**

```
PrintDuration(t, CallingFct = "")
```

**Arguments**

t	Duration; numeric of length 1 or 3.
CallingFct	Name of the calling function.

**Details**

The duration will be printed in the format: hours/minutes/seconds.

**Value**

Prints a character to the screen.

**Examples**

```

ti=getTime_()
for (i in 1:100) x=i*22.1
tf=getTime_()
duration=ComputeDuration(ti,tf)
PrintDuration(duration, "test")

```

---

```
PrintEstimatedRemainingTime
    Estimated Remaining Time
```

---

**Description**

Prints the estimated remaining time in a loop. Useful in Monte Carlo Simulation.

**Usage**

```
PrintEstimatedRemainingTime(ActualIter, ActualIterStartTime, TotalIterNbr)
```

**Arguments**

```
ActualIter      Actual Iteration; integer
ActualIterStartTime
                Actual Iteration Starting time; numeric
TotalIterNbr    Total number of iterations; integer
```

**Details**

Called at the end of each Monte Carlo step, this function will compute the duration of the actual step, an estimate of the remaining MC loops duration and prints the result to the screen in a human readable format using function [PrintDuration](#).

**See Also**

[PrintDuration](#), [ComputeDuration](#).

---

```
RegularisedSol      Regularised Inverse
```

---

**Description**

Regularised solution of the (ill-posed) problem  $K\phi = r$  where  $K$  is a  $n \times n$  matrix,  $r$  is a given vector of length  $n$ . User can choose one of the 3 schemes described in Carrasco and Florens(2007).

**Usage**

```
RegularisedSol(Kn, alphaReg, r,
               regularization = c("Tikhonov", "LF", "cut-off"),
               ...)
```



**Arguments**

Kn	Numeric $n \times n$ matrix.
alphaReg	Regularisation parameter; numeric in ]0,1].
r	Nummeric vector of length n.
regularization	Regularization scheme to be used. User can choose between "Tikhonov" scheme , "LF" (Landweber-Fridmann) and "cut-off"(spectral cut-off). See details.
...	The value of $c$ used in the "LF" scheme. See Carrasco and Florens(2007).

**Details**

Following Carrasco and Florens(2007), the regularised solution of the problem  $K\phi = r$  is given by :

$$\varphi_{\alpha_{reg}} = \sum_{j=1}^n q(\alpha_{reg}, \mu_j) \frac{\langle r, \psi_j \rangle}{\mu_j} \phi_j$$

where  $q$  is a (positive) real function with some regularity conditions and  $\mu, \phi, \psi$  the singular decomposition of the matrix  $K$ .

The regularization parameter defines the form of the function  $q$ . For example, the "Tikhonov" scheme defines  $q(\alpha_{reg}, \mu) = \frac{\mu^2}{\alpha_{reg} + \mu^2}$ .

When the matrix  $K$  is symmetric, the singular decomposition is replaced by a spectral decomposition.

**Value**

Returns the regularised solution, a vector of length n.

**References**

Carrasco M, Florens J and Renault E (2007). "Linear inverse problems in structural econometrics estimation based on spectral decomposition and regularization." *Handbook of econometrics*, **6**, pp. 5633–5751.

**See Also**

[solve](#)

**Examples**

```
## Adapted from R examples for Solve
## We compare the result of the regularized sol to the expected solution

hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+")}

K_h8 <- hilbert(8);
r8 <- 1:8

alphaReg_robust<- 1e-4
Sa8_robust <- RegularisedSol(K_h8,alphaReg_robust,r8,"LF")
```

```
alphaReg_accurate<- 1e-10
Sa8_accurate <- RegularisedSol(K_h8,alphaReg_accurate,r8,"LF")

## when pre multiplied by K_h8 ,the expected solution is 1:8
## User can check the influence of the choice of alphaReg
```

---

sampleComplexCFMoment *Complex moment condition based on the characteristic function.*

---

### Description

Computes the moment condition based on the characteristic function as a complex vector.

### Usage

```
sampleComplexCFMoment(x, t, theta, pm = 0)
```

### Arguments

x	vector of data where the ecf is computed.
t	Vector of (real) numbers where the CF is evaluated; numeric
theta	Vector of parameters of the stable law; vector of length 4.
pm	Parametrisation, an integer (0 or 1); default: pm=0( the Nolan 'S0' parametrisation).

### Details

#### The moment conditions

The moment conditions are given by:

$$g_t(X, \theta) = g(t, X; \theta) = e^{itX} - \phi_\theta(t)$$

If one has a sample  $x_1, \dots, x_n$  of i.i.d realisations of the same random variable  $X$ , then:

$$\hat{g}_n(t, \theta) = \frac{1}{n} \sum_{i=1}^n g(t, x_i; \theta) = \phi_n(t) - \phi_\theta(t)$$

where  $\phi_n(t)$  is the eCF associated to the sample  $x_1, \dots, x_n$  and defined by  $\phi_n(t) = \frac{1}{n} \sum_{j=1}^n e^{itX_j}$

The function compute the vector of difference between the eCF and the CF at a set of given point t.

### Value

Returns a complex vector of length(t).

**See Also**[ComplexCF](#), [sampleRealCFMoment](#)**Examples**

```
## define the parameters
nt <- 10
t <- seq(0.1,3,length.out=nt)
theta <- c(1.5,0.5,1,0)
pm <- 0

set.seed(222);x=rstable(200,theta[1],theta[2],theta[3],theta[4],pm)

# Compute the characteristic function
CFMC <- sampleComplexCFMoment(x=x,t=t,theta=theta,pm=pm)
```

---

sampleRealCFMoment      *Real moment condition based on the characteristic function.*

---

**Description**

Computes the moment condition based on the characteristic function as a real vector.

**Usage**

```
sampleRealCFMoment(x, t, theta, pm = 0)
```

**Arguments**

x	vector of data where the ecf is computed.
t	Vector of (real) numbers where the CF is evaluated ; numeric
theta	Vector of parameters of the stable law; vector of length 4.
pm	Parametrisation, an integer (0 or 1); default: pm=0 (the Nolan 'S0' parametrisation).

**Details****The moment conditions**

The moment conditions are given by:

$$g_t(X, \theta) = g(t, X; \theta) = e^{itX} - \phi_\theta(t)$$

If one has a sample  $x_1, \dots, x_n$  of i.i.d realisations of the same random variable  $X$ , then:

$$\hat{g}_n(t, \theta) = \frac{1}{n} \sum_{i=1}^n g(t, x_i; \theta) = \phi_n(t) - \phi_\theta(t)$$

where  $\phi_n(t)$  is the eCF associated to the sample  $x_1, \dots, x_n$  and defined by  $\phi_n(t) = \frac{1}{n} \sum_{j=1}^n e^{itX_j}$   
 The function compute the vector of difference between the eCF and the CF at a set of given point  $t$ . If  $\text{length}(t)=n$ , the resulting vector will be of  $\text{length}=2n$  where the first  $n$  components will be the real part and the remaining the imaginary part

### Value

Returns a vector of length  $2 * \text{length}(t)$ .

### See Also

[ComplexCF](#), [sampleComplexCFMoment](#)

### Examples

```
## define the parameters
nt <- 10
t <- seq(0.1, 3, length.out=nt)
theta <- c(1.5, 0.5, 1, 0)
pm <- 0

set.seed(222); x=rstable(200, theta[1], theta[2], theta[3], theta[4], pm)

# Compute the characteristic function
CFMR <- sampleRealCFMoment(x=x, t=t, theta=theta, pm=pm)
```

---

StatFcts

*Default functions used to produce the statistical summary.*

---

### Description

Default functions used to produce the statistical summary in the Monte Carlo simulations.

### Details

The functions computed are :

**Mean** `.mean <- function(p,...) mean(p)`

**Min** `.min <- function(p,...) min(p)`

**Max** `.max <- function(p,...) max(p)`

**Sn** `.Sn <- function(p,n,...) sqrt(n)*sd(p)`

**MSE** `.MSE <- function(p,paramT,...) (1/length(p))*sum((p-paramT)**2)`

**Std error** `.st.err <- function(p,...) sd(p)/sqrt(length(p))`

When user wants to change the statistical summary, he needs to provide function with similar signatures and pass the a character vector containing the function names to [Estim\\_Simulation](#).

---

StatObjectFromFiles    *parse an output file to create a summary object (list).*

---

### Description

Parses the file saved by [Estim\\_Simulation](#) and re-creates a summary list identical to the one produced by [Estim\\_Simulation](#) when StatSummary is set to TRUE.

### Usage

```
ComputeStatObjectFromFiles(files, sep_ = ",",
                           FctsToApply = StatFcts,
                           headers_=TRUE,readSizeFrom=1,
                           CheckMat=TRUE,
                           tolFailCheck=tolFailure,
                           MCparam=1000,...)
```

### Arguments

files	character vector containing the files name to be parsed. See details.
sep_	Field separator character to be used in function <code>read.csv()</code> and <code>write.table()</code> . Values on each line of the file are separated by this character; It can also be a vector character (same length as files) if different separators are used for each file; default: ","
FctsToApply	Functions used to produce the statistical summary. See <a href="#">Estim_Simulation</a> ; vector of character
headers_	Vector of boolean of length 1 or same length as files to indicate for each file if the header argument is to be considered or not. To be passed to function <code>read.csv()</code> .
readSizeFrom	index of the file from which the sample sizes are determined; default 1 (from first file in files)
CheckMat	Logical flag: if set to TRUE, an estimation is declared failed if the squared error of the estimation is larger than <code>tolFailCheck</code> ; default TRUE
tolFailCheck	Tolerance on the squared error of the estimation to be declared failed; default 1.5
MCparam	Number of Monte Carlo simulation for each couple of parameter, default=1000; integer
...	Other arguments to be passed to the estimation function. See <a href="#">Estim_Simulation</a>

### Details

The same sample sizes are assumed for all the files and we also assume a different set of parameters (alpha,beta) within each file (one and one only).

This function is particularly useful when simulation are run in parallel on different computers/CPU's and the output files collected afterwards. This function is also used to create the Latex summary table: see [TexSummary](#).

Some examples are provided in the example folder.

### Value

Returns a list of length 4 containing a summary matrix object associated to each parameter.

### See Also

[Estim\\_Simulation](#)

---

TexSummary

*LaTeX summary.*

---

### Description

Creates a Tex table from a summary object or a vector of files.

### Usage

```
TexSummary(obj, files = NULL, sep_ = ",", FctsToApply = StatFcts,
           caption = "Statistical Summary", label = "Simtab",
           digits = 3, par_index = 1, MCparam=1000,...)
```

### Arguments

obj	list of length 4 containing a summary matrix object associated to each parameter identical to the one produced by the function <a href="#">ComputeStatObjectFromFiles</a> .
files	character vector containing the files name to be parsed. It will be passed to function <a href="#">ComputeStatObjectFromFiles</a> .
sep_	Field separator character to be passed to the function <a href="#">ComputeStatObjectFromFiles</a> .
FctsToApply	Functions used to produce the statistical summary to be passed to the function <a href="#">ComputeStatObjectFromFiles</a> .
caption	character vector with length equal to length par_index containing the table's caption or title.
label	Character vector with length equal to length par_index containing the LaTeX label.
digits	numeric vector of length equal to one (in which case it will be replicated as necessary) or to the number of columns of the resulting table or length of FctsToApply or matrix of the same size as the resulting table indicating the number of digits to display in the corresponding columns. See <code>xtable</code> .
par_index	numeric or character vector of length 1,2,3 or 4 of the desired indexes to be selected in obj. See details

MCparam	Number of Monte Carlo simulation for each couple of parameter, default=1000; integer
...	Other arguments to be passed to the function <a href="#">ComputeStatObjectFromFiles</a> .

**Details**

Accepted values for `par_index` are `c(1,2,3,4)` or `c("alpha","beta","gamma","delta")` or mixed.

Some examples are provided in the example folder.

**Value**

Returns a list of objects (same length as `par_index`) from class `Latex`. See `toLatex`

**See Also**

[Estim\\_Simulation](#), [ComputeStatObjectFromFiles](#), `xtable`

# Index

- \*Topic **Estim-functions**
  - CgmmParametersEstim, 6
  - Estim, 14
  - GMMPParametersEstim, 21
  - IGParametersEstim, 25
  - KoutParametersEstim, 27
  - McCullochParametersEstim, 29
  - MLParametersEstim, 30
- \*Topic **Simulation**
  - ComputeBest\_t, 10
  - ComputeBest\_tau, 10
  - ConcatFiles, 13
  - Estim\_Simulation, 16
  - StatObjectFromFiles, 37
  - TexSummary, 38
- \*Topic **classes**
  - Best\_t-class, 4
  - Estim-class, 15
- \*Topic **data-functions**
  - get.abMat, 19
  - get.StatFcts, 19
- \*Topic **general-functions**
  - ComputeDuration, 11
  - expect\_almost\_equal, 18
  - getTime\_, 20
  - IntegrateRandomVectorsProduct, 26
  - PrintDuration, 31
  - PrintEstimatedRemainingTime, 32
  - RegularisedSol, 32
  - StatFcts, 36
- \*Topic **package**
  - StableEstim-package, 2
- \*Topic **stable-functions**
  - CF Jacobian, 5
  - ComplexCF, 9
  - ComputeFirstRootRealeCF, 12
  - sampleComplexCFMoment, 34
  - sampleRealCFMoment, 35
- + , Best\_t, Best\_t-method (Best\_t-class), 4
- Best\_t-class, 4
- CF Jacobian, 5
- CgmmParametersEstim, 3, 6, 15, 18
- ComplexCF, 3, 5, 9, 13, 35, 36
- ComputeBest\_t, 4, 10, 11
- ComputeBest\_tau, 10, 10
- ComputeDuration, 11, 20, 32
- ComputeFirstRootRealeCF, 12
- ComputeStatObjectFromFiles, 38, 39
- ComputeStatObjectFromFiles (StatObjectFromFiles), 37
- ConcatFiles, 13
- Estim, 3, 8, 14, 15, 16, 18, 24, 25, 28–30
- Estim-class, 15
- Estim\_Simulation, 3, 14, 16, 19, 20, 36–39
- expect\_almost\_equal, 18
- get.abMat, 19
- get.StatFcts, 19
- getTime\_, 20
- GMMPParametersEstim, 3, 11, 15, 18, 21
- IGParametersEstim, 3, 14, 21, 25, 29, 30
- initialize, Best\_t-method (Best\_t-class), 4
- initialize, Estim-method (Estim-class), 15
- IntegrateRandomVectorsProduct, 7, 8, 26
- jacobianComplexCF, 3, 9
- jacobianComplexCF (CF Jacobian), 5
- KoutParametersEstim, 3, 15, 27
- McCullochParametersEstim, 25, 29
- MLParametersEstim, 3, 15, 18, 30
- PrintDuration, 12, 20, 31, 32
- PrintEstimatedRemainingTime, 12, 20, 32



RegularisedSol, [21](#), [32](#)

sampleComplexCFMoment, [34](#), [36](#)

sampleRealCFMoment, [35](#), [35](#)

show, Best\_t-method (Best\_t-class), [4](#)

show, Estim-method (Estim-class), [15](#)

solve, [33](#)

StableEstim-package, [2](#)

StatFcts, [36](#)

StatObjectFromFiles, [37](#)

TexSummary, [38](#), [38](#)