Package ‘RandomCoefficients’

June 7, 2019

Title Adaptive Estimation in the Linear Random Coefficients Models

Version 0.0.2

Description
We implement adaptive estimation of the joint density linear model where the coefficients - intercept and slopes - are random and independent from regressors which support is a proper subset. The estimator proposed in Gaillac and Gautier (2019) <arXiv:1905.06584> is based on Prolate Spheroidal Wave Functions which are computed efficiently in 'RandomCoefficients'. This package also provides a parallel implementation of the estimator.

Depends R (>= 3.0.0)

License GPL-3

Encoding UTF-8

LazyData true

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.1.0

Imports snowfall, stats, orthopolynom, polynom, fourierin, sfsmisc, tmvtnorm, rdetools, ks, statmod, RCEIM, robustbase, VGAM

NeedsCompilation no

Author Christophe Gaillac [aut, cre],
Eric Gautier [aut]

Maintainer Christophe Gaillac <christophe.gaillac@ensae.fr>

Repository CRAN

Date/Publication 2019-06-07 14:00:03 UTC

R topics documented:

`boot_stat` *Auxiliary function for parallel implementation of rc_estim*

Description

Auxiliary function that compute for fixed value of t the partial Fourier transform of the density of the random coefficients. This function is used in the parallel computation of the estimator.

Usage

```
boot_stat(t_ind, T_seq, und_X, twoN, N2, L1, d, f_X, Y, X, N, limit, g1, M,
  n_f_X)
```

Arguments

t_ind	index of the parameter t
T_seq	discrete grid for the first variable of the Partial Fourier transform
und_X	bound on the suport of the regressors X
twoN	parameter useful to compute the SVD
N2	parameter useful to compute the SVD
L1	parameter useful to compute the SVD
d	dimension of X
f_X	estimated density of X
Y	outcome data
X	regressors data
N	sample size
limit	apriori on the support of the density of the random slope
g1	evaluation grid for the estimator of the density
M	number of points in g1.
n_f_X	estimated supnorm of the inverse of the density of the regressors

Value

a list containing, in order:

- output

- res0

Examples

---

get_psi_mu *Auxiliary function for the computation of the eigenvalues of the SVD of F_c*

Description

This function compute the eigenvalues of the SVD of F_c.

Usage

```
get_psi_mu(c1, N2, twoN, K1, L1)
```

Arguments

c1	parameter indexing the SVC
N2	maximal number of elements of the SVD that are computed.
twoN	number of Legendre polynomials that are loaded
K1	order of the Legendre quadrature
L1	number of Legendre polynomials used in the computation

Value

a list containing, in order:

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
```

```
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Number of Psis
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c1 = 1
K1 = max(twoN+2,30)
K = K1
### get the beta's (for the computation of Psi)
out <- get_psi_mu(c1,N2,twoN,K1, L1)
Psi1 <- out[[1]]
mu1<- out[[2]]
```

`get_u` *Computation of the coefficients of the PSWF on the Legendre polynomials basis of L^2(-1,1)*

Description

The matrix ueven has columns u^0,u^2,...,u^2N, and the matrix uodd has columns u^1,u^3,...,u^2N+1. Each column has length N+1. Below, note that i must have odd length so that ai and ci have even length and can be split into two subvectors containing alternate entries of ai and ci.

Usage

```
get_u(c, N)
```

Arguments

`c`	parameter indexing the functions of the SVD
`N`	maximal index of the functions computed in the SVD

Value

a list containing, in order:

- ueven : the coefficients of the decomposition of the SVD on the Legendre polynomials for the even functions

- uodd : the coefficients of the decomposition of the SVD on the Legendre polynomials for the odd functions

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Number of Psis
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c1 = 1
K1 = max(twoN+2,30)
K = K1
### get the coefficients beta (for the computation of Psi) of the PSWF on the
### basis of Legendre polynomials.
out <- get_u(c1,N2)
```

`insertEO` *Auxiliary function that put together even and odd functions of the SVD in an only one output list.*

Description

Auxiliary function that put together even and odd functions of the SVD in an only one output list.

Usage

```
insertEO(Psi_odd, Psi_even)
```

Arguments

`Psi_odd`	odd singular functions Psi
`Psi_even`	even singular functions Psi

Value

Psi

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Number of Psis
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c1 = 1
K1 = max(twoN+2,30)
K = K1
### get the beta's (for the computation of Psi)
mu1<- MU(N2,c1,K1, L1)
output <- insertEO(mu1[[2]], mu1[[1]])
```

`legendrequad` *Auxiliary function that compute the Legendre quadrature of order K*

Description

Generate nodes and weights for Legendre-Gauss quadrature on [-1,1]. Note that t is a column vector and w is a row vector. Also normalizes and returns the eigenvectors of J so that they are samples of the unit-norm Legendre polynomials

Usage

```
legendrequad(K)
```

Arguments

`K`	order of the Legendre quadraure

Value

a list containing, in order:

- t : points of the Legendre quadrature

- w : weigths for the Legendre quadrature

- Pbarmat : the eigenvectors of J

Examples

```
K=30
res2 <-  legendrequad(K)
```

`MU` *Auxiliary function that computes the singular values of the SVD of the operator F_c in Gaillac and Gautier 2018.*

Description

Auxiliary function that computes the singular values of the SVD of the operator F_c in Gaillac and Gautier 2018.

Usage

```
MU(N, c, K, L1)
```

Arguments

`N`	number of singular values to compute
`c`	parameter indexing the singular values
`K`	ordre of the Legendre quadrature
`L1`	number of Legendre polynomials used in the computation of the coefficients of the singular functions;

Value

a list containing, in order:

- mu_even

- mu_odd

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Bandwidth 1
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c1 = 1
K1 = max(twoN+2,30)
K = K1
mu1<- MU(N2,c1,K1, L1)
```

MU_fourier — *Auxiliary function that computes the singular values of the SVD of the operator F_c in Gaillac and Gautier 2018 using the Fast Fourier transform for the integration.*

Description

Auxiliary function that computes the singular values of the SVD of the operator F_c in Gaillac and Gautier 2018 using the Fast Fourier transform for the integration.

Usage

```
MU_fourier(psi, xseq, splin)
```

Arguments

psi	Prolate spheroidal wave functions
xseq	grid on which to evaluate them, output of the Legendre quadrature
splin	use interpolation by splines or not (boolean).

Value

mu

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Bandwidth 1
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c1 = 1
K1 = max(twoN+2,30)
K = K1
c = 1
b=1
bound=1
out <- get_psi_mu(c,N2,twoN,K, L1)
Psi <- out[[1]]
mu<- out[[2]]
xseq = seq(-bound,bound, length.out=10)
resol=2^7
psix <- PSI_mu_fourier(xseq,c,b,Psi,resol)
psi <- psix[[1]]
xseq <- psix[[2]]
xseq2 <- xseq/b
splin =FALSE
mu_ev<- MU_fourier(psi,xseq,splin)
mu <- mu_ev[[1]]
```

`myDiag` *Auxiliary function to form matrices equal to x everywhere except from the upper/lower k diagonal, which values are vec*

Description

Auxiliary function to form matrices equal to x everywhere except from the upper/lower k diagonal, which values are vec

Usage

```
myDiag(x, vec, k)
```

Arguments

x	initial matriw
vec	vector with specified new values for the upper/lower k diagonal elements of x
k	index of the diagonal

Value

x

Examples

```
K=3
u <- sqrt(1/(4-1/seq(1,(K-1))^2))
n = length(u)+1
trans = myDiag(matrix(0,n, n),u,1) + myDiag(matrix(0,n, n),u,-1)
```

PSI_mu *Ausiliary function that evaluates the SVD of F_c on a pre-specified grid divided by the singular values to the square.*

Description

Ausiliary function that evaluates the SVD of F_c on a pre-specified grid divided by the singular values to the square.

Usage

```
PSI_mu(x, N, c, K, L1)
```

Arguments

x	the pre-specified grid
N	number of singular values to compute
c	parameter indexing the singular values
K	ordre of the Legendre quadrature
L1	number of Legendre polynomials used in the computation of the coefficients of the singular functions;

Value

a list containing, in order:

- ipeven

- ipodd

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Bandwidth 1
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c= 1
K1 = max(twoN+2,30)
K = K1
res2 <-  legendrequad(K)
t <- res2[[1]]
psi_even <- abs(PSI_mu(t,N2,c,K, L1)[[1]])
```

PSI_mu_fourier — *Auxiliary funciton for the evaluation of the SVD of F_c on a pre-specified grid divided by the singular values to the square.*

Description

Computation is done using the Fast Fourier transform for the integration.

Usage

```
PSI_mu_fourier(x, c, b, Psi, res)
```

Arguments

x	a pre-specified grid
c	the parameter indexing the singular functions
b	the parameter indexing the smoothness class (see Gaillac and Gautier 2018)
Psi	the Prolate Spheroidal wave functions
res	the resolution level for the FFT.

Value

a list containing, in order:

- ipeven

- ipodd

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
#### Bandwidth 1
L =15
L1 = L+1
N2 = max(L,3)
twoN = 2*N2
#### Bandwidth 1
c1 = 1
K1 = max(twoN+2,30)
K = K1
c = 1
b=1
bound=1
out <- get_psi_mu(c,N2,twoN,K, L1)
Psi <- out[[1]]
mu<- out[[2]]
xseq = seq(-bound,bound, length.out=10)
resol=2^7
psix <- PSI_mu_fourier(xseq,c,b,Psi,resol)
```

`rc_estim` *Adaptive estimation of the joint density of random coefficients model in a linear random coefficient model*

Description

This function implements the adaptive estimation of the joint density of random coefficients model as in Gaillac and Gautier (2019). It takes as inputs data (Y,X) then estimates the density and return its evaluation on a grid b_grid times a_grid. By setting nbCores greater than 1 computations are done in parallel.

Usage

```
rc_estim(X, Y, b_grid, a_grid = b_grid, nbCores = 1, M_T = 60,
  N_u = 10, epsilon = (log(N)/log(log(N)))^(-2), n_0 = 0,
  trunc = 0, center = 0)
```

Arguments

X	Vector of size N, N being the number of observation and the number of regressors limited to 1 in this version of the package.
Y	Outcome vector of size $N $.
b_grid	vector grid on which the estimator of the density of the random slope will we evaluated.
a_grid	Vector grid on which the estimator of the density of the random intercept will we evaluated.
nbCores	number of cores for the parallel implementation. Default is 1, no parallel computation.
M_T	number of discretisation points for the estimated partial Fourier transform. Default is 60.
N_u	aximal number of singular functions used in the estimation. Default is the maximum of 10 and N_max.
epsilon	parameter for the interpolation. Default is (log(N)/log(log(N)))^(-4) as is (T5.1) in Gaillac and Gautier (2019).
n_0	Parameter for the sample splitting. If n_0 = 0 then no sample splitting is done and we use the same sample of size N to perform the estimation of the truncated density. If n_0 >0 , then this is the size of the sample used to perform the estimation of the truncated density. Default is $n_0 =0$.\
trunc	Dummy for the truncation of the density of the regressors to an hypercube [x_0,x_0]^p. If trunc=1, then truncation is performed and [x_0,x_0]^p is defined using the argmin of the ratio of the estimated constant c_X over the percentage of observation in [x_0,x_0]^p. Default is 0, no truncation.
center	Dummy to trigger the use of X -x_bar instead of X as regressor. If center=1, then use X - x_bar where x_bar is the vector of the medians coordinates by coordinates for X. Default is center=0, where regressors are left unchanged.

Value

a list containing, in order:

- outcome : the matrix of size length(b_grid)x length(a_grid) which contains the evaluation of the estimator of the density on the grid b_grid times a_grid

- b_grid : vector grid on which the estimator of the density is evaluated for the random slope

- a_grid : vector grid on which the estimator of the density is evaluated for the random intercept

- x_bar : vector used to center the regressors, if center=1

- n_f_X : estimated supnorm of the inverse of the density of the regressors

- und_X : parameter x_0 used for the truncation (truncate =1) of the density of the regressors.

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)

# beta (output) Grid
M=100
limit =7.5
b_grid  <- seq(-limit ,limit ,length.out=M)
a = limit

up =1.5
down = -up
und_beta <- a
x2 <- b_grid
x.grid <- as.matrix(expand.grid(b_grid ,b_grid ))
# DATA generating process
d = 1
Mean_mu1 = c(-2,- 3)
Mean_mu2= c(3,  0)
Sigma= diag(2, 2)
Sigma[1,2] = 1
Sigma[2,1] = 1
limit2 = 6



N <- 1000
xi1 <- rtmvnorm(N, mean = Mean_mu1, sigma=Sigma, lower=c( -limit2,-limit2), upper=c(limit2,limit2))
xi2 <- rtmvnorm(N, mean = Mean_mu2, sigma=Sigma, lower=c( -limit2,-limit2), upper=c(limit2,limit2))
theta = runif(N, -1 , 1)
beta <- 1*(theta >=0) * xi1  + 1*(theta <0) * xi2
X <- rtmvnorm(N, mean = c(0), sigma=2.5, lower=c( down), upper=c(up))
X_t <- cbind(matrix(1, N,1),X)
Y <-rowSums(beta*X_t)
out <- rc_estim( X,Y,b_grid,b_grid,nbCores = 1, M_T = 60)
```

`repmat` *Auxiliary function that extends the matrix X*

Description

Auxiliary function that extends the matrix X

Usage

```
repmat(X, m, n)
```

Arguments

`X`	A vector of inputs
`m`	the first dimension of the desired output matrix
`n`	the second dimension of the desired output matrix

Value

A matrix os size (dim(X)[1]*m,dim(X)[2]*n)

Examples

```
library(orthopolynom)
library(polynom)
library(tmvtnorm)
library(ks)
library(sfsmisc)
library(snowfall)
library(fourierin)
library(rdetools)
library(statmod)
library(RCEIM)
library(robustbase)
library(VGAM)
library(RandomCoefficients)
K=3
u <- sqrt(1/(4-1/seq(1,(K-1))^2))
n = length(u)+1
trans = myDiag(matrix(0,n, n),u,1) + myDiag(matrix(0,n, n),u,-1)
eigen_trans <- eigen(trans)
V<- eigen_trans$vectors
Lambda <- eigen_trans$values
t = sort(Lambda)
i= sort(seq(1, length(Lambda)), decreasing=TRUE)
V = V[,i,drop=FALSE]
Vtop = V[1,,drop=FALSE]
w = 2*Vtop^2
Pbarmat = V/repmat(Vtop*sqrt(2),K,1)
```

Index