Package 'ConConPiWiFun'

October 12, 2022

Type Package

Title Optimisation with Continuous Convex Piecewise (Linear and Quadratic) Functions

Version 0.4.6.1

Date 2013-06-05

Author Robin Girard

Maintainer Robin Girard <robin.girard@mines-paristech.fr>

Description Continuous convex piecewise linear (ccpl) resp. quadratic (ccpq) functions can be implemented with sorted breakpoints and slopes. This includes functions that are ccpl (resp. ccpq) on a convex set (i.e. an interval or a point) and infinite out of the domain. These functions can be very useful for a large class of optimisation problems. Efficient manipulation (such as log(N) insertion) of such data structure is obtained with map standard template library of C++ (that hides balanced trees). This package is a wrapper on such a class based on Rcpp modules.

License GPL (>= 2)

Depends methods, graphics, Rcpp (>= 0.10.3)

LinkingTo Rcpp

RcppModules mod_cplfunction,mod_cpqfunction

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-10-14 16:34:23 UTC

R topics documented:

ConConPiWiFun-package *This package contains an implementation of continuous convex piecewise (linear) functions (quadratic coming soon)*

Description

Continuous convex piecewise linear (ccpl) resp. quadratic (ccpq) functions can be implemented with sorted breakpoints and slopes. This includes functions that are ccpl (resp. ccpq) on a convex set (i.e. an interval or a point) and infinite out of the domain. These functions can be very usefull for a large class of optimisation problems. Efficient manipulation (such as log(N) insertion) of such data structure is obtained with map standard template library of C++ (that hides balanced trees). This package is a wrapper on such a class based on Rcpp modules.

Details

Package:	ConConPiWiFun
Type:	Package
Version:	0.3.0
Date:	2013-02-08
License:	GPL

Author(s)

Robin Girard

Maintainer: <robin.girard@mines-paristech.fr>

References

Related Papers are

Examples

```
library(ConConPiWiFun)
####  See
#? cplfunction for continuous convex piecewise functions
#? cplfunctionvec for (optimized) list of continnuous convex piecewise functions
```

cplfunction *This class implements continuous convex piecewise linear functions*

Description

This includes functions that are ccpl (resp. ccpq) on a convex set (i.e. an interval or a point) and infinite out of the domain. These functions can be very usefull for a large class of optimisation problems. Efficient manipulation (such as log(N) insertion) of such data structure is obtained with map standard template library of C++ (that hides balanced trees). This package is a wrapper on such a class based on Rcpp modules.

Author(s)

Robin Girard

See Also

to See Also as cplfunction,

Examples

```
##
#Construction of a piecewise linear function
##


Slopes=c(-1,2,Inf) # increasing ! convexity is required
Breakpoints=c(-Inf,2,4) # increasing. length is number of slopes +1
FirstNonInfBreakpointVal=3
CCPWLfunc1=new(cplfunction,Slopes,Breakpoints,FirstNonInfBreakpointVal)
plot(CCPWLfunc1) #visualisation method


###Etoile transformation (legendre transform of f)
# Changes f no return value
CCPWLfunc1$Etoile()
plot(CCPWLfunc1)  #if f = CCPWLfunc1 CCPWLfunc1 becomes is f^*(y) =inf_x {xy-f(x)}
CCPWLfunc1$Etoile()
plot(CCPWLfunc1)   ## (f^*)^* is f !


###Squeeze function
# Changes f, no return value
left=-Inf; right=3
CCPWLfunc1$Squeeze(left,right) # CCPWLfunc1 is now infinite (or not definite) out of [left,right]
# i.e. all breakpoints out of [left,right] removed


###Swap function
```

```
# Changes f  no return value !
y=2;
CCPWLfunc1$Swap(y)
plot(CCPWLfunc1); #now f = CCPWLfunc1 is replaced by x -> f(y-x)


### Sum function (uses fast insertion) do not affect operands
CCPWLfunc1=new(cplfunction,c(-1,2,Inf) ,c(-Inf,2,4),0)
CCPWLfunc2=new(cplfunction,c(-1,2,Inf),c(-Inf,1,3),0)
CCPWLfunc1plus2=Suml(CCPWLfunc1,CCPWLfunc2)
CCPWLfunc1plus2


par(mfrow=c(1,3))
plot(CCPWLfunc2,col='red');
plot(CCPWLfunc1,col='blue');
plot(CCPWLfunc1plus2);


rm(list=ls())
gc()
```

`cplfunctionvec` *This class implements "optimized list" of continuous convex piecewise linear functions*

Description

This is a wrapper to stl vector of convex piecewise linear functions. Allows to loop efficiently on such list.

Author(s)

Robin Girard

See Also

to See Also as `cplfunction`, `cpqfunctionvec`

Examples

```
####
# construction of a vector of
# continuous convex piecewise linear functions

CCPWLfuncList=new(cplfunctionvec)
```

```
CCPWLfuncList$push_back(new(cplfunction,c(-1,1) ,c(-Inf,0),0))
CCPWLfuncList$push_back(new(cplfunction,c(-1,1) ,c(-Inf,0),0))

CCPWLfuncList=new(cplfunctionvec)
n=1000; Y=rnorm(n); S1=array(-1,n);S2=array(1,n); B0=array(-Inf,n); B1=rnorm(n);
for (i in 1:n){
  CCPWLfuncList$push_back(new(cplfunction,c(S1[i],S2[i]) ,c(B0[i],B1[i]),0))
}
CCPWLfuncList$size() ## gives the size
## The same but faster
CCPWLfuncList=new(cplfunctionvec)
CCPWLfuncList$SerialPush_2Breaks_Functions(S1,S2,B0,B1);

#### method OptimMargInt solves
#          min_x sum_i=1^n C_i(x_i)
#                   Pmoins_i<= x_i <=Pplus_i  i=1,...,n
# Cmoins_i<= sum_j=1^i x_j <=Cplus_i  i=1,...,n

Pmoins=array(-1,n);Pplus=array(1,n);Cmoins=array(0,n);Cplus=array(5,n);
res=CCPWLfuncList$OptimMargInt(Pmoins,Pplus,Cmoins,Cplus)

par(mfrow=c(1,2))
plot(Y,type='l',ylim=range(res$xEtoile))
lines(y=Pmoins,x=1:n,col='blue'); lines(y=Pplus,x=1:n,col='blue');
lines(y=res$xEtoile,x=1:n,col='red')
text(x=800,y=3,paste("Optimum=",signif(sum(abs(res$xEtoile-Y)),digits=6)))
plot(Y,type='l',ylim=c(min(Y),max(diffinv(res$xEtoile)[1:n+1])))
lines(y=Cmoins,x=1:n,col='blue'); lines(y=Cplus,x=1:n,col='blue');
lines(y=diffinv(res$xEtoile)[1:n+1],x=1:n,col='red')

rm(list=ls())
gc()
```

`cpqfunction` *This class implements continuous convex piecewise quadratic functions*

Description

This includes functions that are ccpq on a convex set (i.e. an interval or a point) and infinite out of the domain. These functions can be very usefull for a large class of optimisation problems. Efficient manipulation (such as log(N) insertion) of such data structure is obtained with map standard template library of C++ (that hides balanced trees). This package is a wrapper on such a class based on Rcpp modules.

Author(s)

Robin Girard

See Also

to See Also as cplfunction,

Examples

```
##
#Construction of a piecewise quadratic function
##
Slopes1=c(-1,2)
Slopes0=c(-2,0)# increasing ! convexity is required
Breakpoints=c(-Inf,2,4) # increasing. length is number of slopes +1
FirstNonInfBreakpointVal=3
CCPWLfunc1=new(cpqfunction,Slopes0,Slopes1,Breakpoints,FirstNonInfBreakpointVal)
CCPWLfunc1$get_BreakPoints_() ## return Breaks AND Slopes
plot(CCPWLfunc1)

###Etoile transformation (legendre transform of f)
# Changes f no return value
CCPWLfunc1$Etoile()
CCPWLfunc1$get_BreakPoints_()
CCPWLfunc1$Etoile()
CCPWLfunc1$get_BreakPoints_()  ## (f^*)^* is f !


###Squeeze function
# Changes f, no return value
left=-1; right=4
CCPWLfunc1$Squeeze(left,right) # CCPWLfunc1 is now infinite (or not definite) out of [left,right]
# i.e. all breakpoints out of [left,right] removed
CCPWLfunc1$get_BreakPoints_()

###Swap function
# Changes f  no return value !
y=2;
CCPWLfunc1$Swap(y)
CCPWLfunc1$get_BreakPoints_() #now f = CCPWLfunc1 is replaced by x -> f(y-x)


### Sum function (uses fast insertion) do not affect operands
CCPWLfunc1=new(cpqfunction,Slopes0,Slopes1,Breakpoints,FirstNonInfBreakpointVal)
CCPWLfunc2=new(cpqfunction,Slopes0,Slopes1+1,Breakpoints,FirstNonInfBreakpointVal)
CCPWLfunc1plus2=Sumq(CCPWLfunc1,CCPWLfunc2)
CCPWLfunc1plus2$get_BreakPoints_()


rm(list=ls())
gc()
```

cpqfunctionvec *This class implements "optimized list" of continuous convex piecewise quadratic functions*

Description

This is a wrapper to stl vector of convex piecewise quadratic functions. Allows to loop efficiently on such list.

Author(s)

Robin Girard

See Also

to See Also as cpqfunction, cplfunctionvec

Examples

```
CCPWLfuncList=new(cpqfunctionvec)
CCPWLfuncList$push_back(new(cpqfunction,c(0),c(1),c(-2, 2),0))
CCPWLfuncList$push_back(new(cpqfunction,c(0),c(1),c(-2, 2),0))

CCPWLfuncList=new(cpqfunctionvec)
n=1000; Y=rnorm(n); S0=array(0,n)+Y;S1=array(1,n)+Y; B0=array(-Inf,n); B1=array(Inf,n);
for (i in 1:n){
  CCPWLfuncList$push_back(new(cpqfunction,S0[i],S1[i] ,c(B0[i],B1[i]),0))
}
CCPWLfuncList$size() ## gives the size
## The same but faster
CCPWLfuncList=new(cpqfunctionvec)
CCPWLfuncList$SerialPush_0Breaks_Functions(S0,S1);

#### method OptimMargInt solves
#          min_x sum_i=1^n C_i(x_i)
#                   Pmoins_i<= x_i <=Pplus_i  i=1,...,n
# Cmoins_i<= sum_j=1^i x_j <=Cplus_i  i=1,...,n

Pmoins=array(-1,n);Pplus=array(1,n);Cmoins=array(0,n);Cplus=array(5,n);
res=CCPWLfuncList$OptimMargInt(Pmoins,Pplus,Cmoins,Cplus)

par(mfrow=c(1,2))
plot(Y,type='l')
lines(y=Pmoins,x=1:n,col='blue'); lines(y=Pplus,x=1:n,col='blue');
lines(y=res$xEtoile,x=1:n,col='red')
text(x=800,y=3,paste("Optimum=",signif(sum(abs(res$xEtoile-Y)),digits=6)))
plot(Y,type='l',ylim=c(min(Y),max(diffinv(res$xEtoile)[1:n+1])))
lines(y=Cmoins,x=1:n,col='blue'); lines(y=Cplus,x=1:n,col='blue');
lines(y=diffinv(res$xEtoile)[1:n+1],x=1:n,col='red')
```

```
rm(list=ls())
gc()
```

`OptimPriceStorage` *Optimisation of storage operation with market prices taking into acount storage efficiency and network taxes.*

Description

Optimisation of storage operation with market prices taking into acount storage efficiency and network taxes.

Usage

```
OptimPriceStorage(Prices,Pplus,Pmoins,Cplus,Cmoins=0,
                              efficiencyS=0,efficiencyP=efficiencyS,networkTax=0)
```

Arguments

`Prices`	A vector of prices
`Pplus`	A value for the upper power constraint or a vector of values with the same size as Prices
`Pmoins`	A value for the lower power constraint or a vector of values with the same size as Prices
`Cplus`	A value for the upper capacity constraint or a vector of values with the same size as Prices
`Cmoins`	A value for the lower capacity constraint or a vector of values with the same size as Prices
`efficiencyS`	storage efficiency when storing electricity
`efficiencyP`	storage efficiency when producing electricity
`networkTax`	networkTax

Details

function OptimPriceStorage solves # min_x sum_i=1^n Y_i*efficiencyP x_i*(x_i<0)+(Y_i*efficiencyS +networkTax)*x_i*(x_i>0) # Pmoins_i<= x_i <=Pplus_i i=1,...,n # Cmoins_i<= sum_j=1^i x_j <=Cplus_i i=1,...,n

when efficiency=1 and networkTax=0 this gives # min_x sum_i=1^n Y_i x_i # Pmoins_i<= x_i <=Pplus_i i=1,...,n # Cmoins_i<= sum_j=1^i x_j <=Cplus_i i=1,...,n

Value

A list with

`Operation`	the optimal operation for each time step
`Revenue`	the revenue for each time step

Note

TODO

Author(s)

Robin Girard

References

TODO

See Also

to See Also `cplfunction` (method OptimMargInt that is more general)

Examples

```
n=8760
Prices=runif(n,1,100) ##uniform random prices in [1;100] in Euro/MWh
Pmax=1; Pmin=-1; Cmax=5; ## 1MW maximum  during 5 hours.
res=OptimPriceStorage(Prices,Pmax,Pmin,Cmax) # solving the optimization problem
sum(res$Revenue)## Revenue
res=OptimPriceStorage(Prices,Pmax,Pmin,Cmax,efficiencyS=0.8) # solving the optimization problem
sum(res$Revenue)## Revenue
```

Index