Package 'UMR'

August 14, 2021

Title Unmatched Monotone Regression

Version 1.1.0

Description Unmatched regression refers to the regression setting where covariates and predictors are collected separately/independently and so are not paired together, as in the usual regression setting. Balabdaoui, Doss, and Durot (2021) <arXiv:2007.00830> study the unmatched regression setting where the univariate regression function is known to be monotone. This package implements methods for computing the estimator developed in Balabdaoui, Doss, and Durot (2021). The main method is an active-set-trust-region-based method.

License GPL (>= 3)

Encoding UTF-8

Depends decon, trust, distr

Suggests purrr, Iso

RoxygenNote 7.1.1

NeedsCompilation no

Author Charles Doss [aut, cre] (<https://orcid.org/0000-0003-1364-5222>)

Maintainer Charles Doss <cdoss@stat.umn.edu>

Repository CRAN

Date/Publication 2021-08-14 09:00:09 UTC

R topics documented:

AA *Helper functions for calculating gradient of least-squares Shuffled Isotonic Regression criterion, for Laplace or for Gaussian errors*

Description

Helper functions for calculating gradient of least-squares Shuffled Isotonic Regression criterion, for Laplace or for Gaussian errors

Usage

```
AA(yy, mm, func)

BB(mm, func)

AAfunc_Laplace_generic(dd, LL)

AAfunc_Gauss_generic(dd, sig)

BBfunc_Laplace_generic(dd, LL)

BBfunc_Gauss_generic(dd, sig)

getAAfunc_est_outer(eps, ww = 1/length(eps))

getBBfunc_est_outer(eps, ww = 1/length(eps))

BBfunc_mixGauss_generic(dd, locs, wws, sigs)

BBpfunc_mixGauss_generic(dd, locs, wws, sigs)

BBpfunc_Gauss_generic(xx, sig)

BBpfunc_Laplace_generic(xx, myLL)
```

Arguments

yy	Y (response) observation vector (numeric). Will apply as.vector() so it may be a matrix or array with all dimensions trivial except 1.

mm	Current (unsorted) estimate/iterate at which to compute gradient. (Length equals length of yy). Will apply as.vector() so it may be a matrix or array with all dimensions trivial except 1.
func	This is a function; should be the actual "A" or "B" function from the paper; AA and BB are just wrappers that call outer() with func(). func() should accept vector or matrix arguments.
dd	generic argument to the "A" function; usually of the form m - mmhat, where m is just some value of the regression function
LL	Double Exponential "mean" parameter: corresponding density is $exp(-|dl/LL) / (2LL)$.
sig	is standard deviation of the normal distribution.
eps	is a vector of residuals (or estimated residuals). In current coding, it should have been preprocessed to be *unique*. (If there are repeats this should be encoded in ww).
ww	is vector of weights of same length as eps, and summing to 1. Default is a weight of 1/length(eps) for each value of eps; if eps has been pre-binned then ww is the weights from binning.
locs	Vector (length LL) of mixture locations
wws	Vector (length LL, sum to 1) of mixture weights
sigs	Vector (length LL, positive) of component standard deviations ## here dd should be a matrix (usually from a call to outer())
xx	Point at which to evaluate function
myLL	is Laplace parameter.

Details

See helper functions "A" and "B" in paper.

For getAAfunc_est: returns a function(yy,mm) which is analogous to passing in an estimated 'func' argument to the AA function. (Reason to not do it that way relates to making sure matrix arguments are handled correctly.)

getBBfunc_est returns a function which as of this coding *MUST* take only a numeric vector of length 1; longer vectors will not work. Be careful! Note that ecdf objects are not intended to be stored permanently so storing functions returned by getBBfunc_est_outer or getAAfunc_est_outer may cause issues.

Examples

```
 ## the "!!" de-quote (see ?partial) so e.g., can save mygradSIR for future runs.

####### gradient settings/setup for Gaussian


## set.seed(501)
```

```
library(distr)
mysig <- 1 ##  std dev
errdist <- Norm(0, sd=mysig)
mm0 <- function(xx){xx}
nn <- 300
xx <- sort(runif(n=nn, 0, 7))
yy <- mm0(xx) + errdist@r(nn)
## plot(xx,yy)

myScale <- mysig

AAfunc_Gauss <- purrr::partial(AAfunc_Gauss_generic, sig=!!mysig)
AA_Gauss <- purrr::partial(AA, func=!!AAfunc_Gauss)
BBfunc_Gauss <- purrr::partial(BBfunc_Gauss_generic, sig=!!mysig)
BB_Gauss <- purrr::partial(BB, func=!!BBfunc_Gauss)
mygradSIR <-
    grad_SIR_Gauss <- ## just for ease of reference
        purrr::partial(grad_SIR_generic,
                       rescale=TRUE, ## factor of nn/2
                       AAfunc=!!AA_Gauss, BBfunc=!!BB_Gauss)

####### gradient settings/setup for Laplace




set.seed(501)
library(distr)
myLL <- .7 ## (1/"rate") parameter, aka "mean" parameter (except Laplace mean is 0)
errdist <- DExp(1/myLL)


nn <- 200
mm0 <- function(xx){
   (xx<=0)*0 + (0<=xx & xx<=2)*1 +
       (2<xx & xx<=3)*3 +
       (3<xx)*6
}
xx <- sort(runif(n=nn, 0, 7))
yy <- mm0(xx) + errdist@r(nn)

myScale <- myLL;

## CS settings
#'mysig <- sqrt(2) * myLL;
#'
AAfunc_Laplace <- purrr::partial(AAfunc_Laplace_generic, LL=!!myLL)
AA_Laplace <- purrr::partial(AA, func=!!AAfunc_Laplace)
BBfunc_Laplace <- purrr::partial(BBfunc_Laplace_generic, LL=!!myLL)
BB_Laplace <- purrr::partial(BB, func=!!BBfunc_Laplace)
mygradSIR <-
    grad_SIR_Laplace <- purrr::partial(grad_SIR_generic,
                                       rescale=TRUE, ## factor of nn/2
```

```
                                    AAfunc=!!AA_Laplace, BBfunc=!!BB_Laplace)
```

`gradDesc_fixed_df` *Gradient Descent with a fixed number of constant pieces (degrees of freedom)*

Description

Gradient Descent with a fixed number of constant pieces (degrees of freedom)

Usage

```
gradDesc_fixed_df(
  yy,
  grad,
  init = stats::median(yy),
  counts = length(yy),
  stepsize,
  MM,
  tol = 1e-07,
  printevery = Inf,
  filename
)
```

Arguments

`yy`	Y (response) observation vector (numeric)
`grad`	a function(yy, mm) where mm may be shorter length than yy and is the previous iterate value (i.e., the estimate vector).
`init`	Initial value of estimate ('mm'). I.e., numeric vector of length <= length(mm). The output will be of length length(init).
`counts`	Vector of length length(init); each entry indicates how many values of yy the corresponding value of init (and output) corresponds to. Alternatively, can think of counts as a vector of weights for each estimator value.
`stepsize`	Gradient descent stepsize. Set carefully!
`MM`	Number of iterations in which "support reduction" (combining of approximately equal values into a region of constancy) is done (see details and paper). Depending on tol, may not use all MM iterations.
`tol`	Tolerance: end algorithm once sum(abs(mm-mmprev)) < tol or you hit MM iterations.
`printevery`	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
`filename`	path1/path2/filename to save output to.

Details

Prefer using UMRgradDesc_fixed_df now; this function deprecated.

xxxx Implements a gradient descent. See paper for details. Right now stepsize is fixed. Right now: init gets sorted in gradDesc_PC so does not need to be sorted on input. Roughly, the difference between this algorithm and gradDesc() (which is just vanilla gradient descent on this problem) is that: if mm is the current value of the output estimate, then gradDesc_PC 'collapses' or combines values of mm that are (roughly, up to tolerance 'eps') equal. Because the solution is generally piecewise constant with a relatively small number of constant regions this enormously speeds up the later stages of the algorithm. Note that once points are combined/collapsed they contribute identically to the objective function, so they will never be "uncombined".

`objective_fn_numint` *Compute Unlinked Monotone Regression objective function numerically*

Description

Compute Unlinked Monotone Regression objective function numerically

Usage

```
objective_fn_numint(
  mm,
  ww_m = NULL,
  yy,
  ww_y = NULL,
  Phi,
  subdivisions = 1000L
)
```

Arguments

`mm`	Current (unsorted) estimate/iterate at which to compute gradient. (Length is <= than the number of X observations in the problem).
`ww_m`	Weights (nonnegative, sum to 1) corresponding to mm. Same length as mm.
`yy`	Y (response) observation vector (numeric vector). Alternatively, yy may be an ecdf, i.e. ecdf(yy) or getEcdf(yy, weights).
`ww_y`	Weights (nonnegative, sum to 1) corresponding to yy. Same length as yy. Default is just 1/length(yy) for each value. If yy is non-numeric i.e. yy is an ecdf() then ww_y is ignored.
`Phi`	This is the error (cumulative) distribution function, a function object (Balabdaoui, Doss, Durot (2020+). Function accepting vector or matrix arguments.
`subdivisions`	Passed argument to integrate().

Details

See paper for derivations.

UMR *UMR: For computing an estimator in Unlinked Monotone Regression.*

Description

A package for computing an estimator in the problem of univariate Unlinked Monotone Regression. See Balabdaoui, Doss, and Durot (2021).

UMR functions

The main function is UMRactiveSet_trust, which uses the trust region for second order optimization of the nonconvex objective function as a subroutine. Other functions for optimizing are also provided, for comparisons; these include gradDesc_PC (for Gradient Descent for Piecwise Constant functions), gradDesc. The former is faster than the latter (but slower than the second order method). The latter is the more naive vanilla gradient descent method (can be used for instance to double check results from gradDesc_PC).

UMRactiveSet *An active set approach to minimizing objective in Unlinked Monotone Regression*

Description

An active set approach to minimizing objective in Unlinked Monotone Regression

Usage

```
UMRactiveSet(
  yy,
  grad,
  CC_SIR,
  init,
  counts = rep(1, length(init)),
  stepsize,
  MM,
  tol_end = 1e-04,
  tol_collapse,
  printevery,
  filename
)
```

Arguments

yy	Y (response) observation vector (numeric)
grad	a function(yy, mm) where mm is the previous iterate value (i.e., the estimate vector).
CC_SIR	A curvature function object (denoted "C" in the paper). See CC_SIR_generic() and examples.
init	Initial value of estimate ('mm'). Vector, length may be different than length(yy). See 'counts' input.
counts	Together 'init' and 'counts' serve as the initialization; the implied initial vector is rep.int(init, counts).
stepsize	Gradient descent stepsize.
MM	A number of iterations. May not use them all. MM is not exactly the total number of iterations used in the sense that within each of MM iterations, we will possibly run another algorithm which may take up to MM iterations (but usually takes many fewer).
tol_end	Used as tolerance at various points . Generally algorithm (and some subalgorithms) end once sum(abs(mm-mmprev)) < tol, or you hit MM iterations.
tol_collapse	Collapsing roughly equal mm values into each other.
printevery	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
filename	filename (path) to save output to.
	param ww_y Weights (nonnegative, sum to 1) corresponding to yy. Same length as yy.

Details

Uses first order (gradient) for optimization, and uses certain second derivative computations to leave saddle points. See Balabdaoui, Doss, and Durot (20xx). Note that yy and mm (i.e., number covariates) may have different length.

`UMRactiveSet_trust` *An active set approach to minimizing objective in Unlinked Monotone Regression*

Description

An active set approach to minimizing objective in Unlinked Monotone Regression

Usage

```
UMRactiveSet_trust(
  yy,
  ww_y = NULL,
  grad,
  hess,
  UMR_curv,
  CDF,
  init,
  counts = rep(1, length(init)),
  stepsize,
  MM,
  tol_end = 1e-04,
  tol_collapse,
  printevery,
  filename
)
```

Arguments

yy	Y (response) observation vector (numeric)
ww_y	Weights (nonnegative, sum to 1) corresponding to yy. Samelength as yy. Or NULL in which yy are taken as being evenly weighted.
grad	Is function(mm, ww_m). (Will be defined based on yy [and maybe ww_y] before being passed in.) Returns vector of length(mm). Gradient of objective function.
hess	Is function(mm, ww_m). (Will be defined based on yy [and maybe ww_y] before being passed in.) Returns matrix of dimensions length(mm) by length(mm). Hessian of objective function.
UMR_curv	A curvature function object (giving mathfrak(C) in the paper; and related to "C" in the paper). See UMR_curv_generic() and examples. This is generally a "curried" version of UMR_curv_generic with densfunc and BBp passed in.
CDF	This is the error (cumulative) distribution function, a function object. Function accepting vector or matrix arguments.
init	Initial value of estimate ('mm'). Vector, length may be different than length(yy). See 'counts' input.
counts	Together 'init' and 'counts' serve as the initialization; the implied initial vector is rep.int(init, counts).
stepsize	Stepsize for moving out of saddle points.
MM	A number of iterations. May not use them all. MM is not exactly the total number of iterations used in the sense that within each of MM iterations, we will possibly run another algorithm which may take up to MM iterations (but usually takes many fewer).
tol_end	Used as tolerance at various points . Generally algorithm (and some subalgorithms) end once sum(abs(mm-mmprev)) < tol, or you hit MM iterations.

tol_collapse	Collapsing roughly equal mm values into each other.
printevery	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
filename	filename (path) to save output to.

Details

Uses first order (gradient) for optimization, and uses certain second derivative computations to leave saddle points. See Balabdaoui, Doss, and Durot (2021). Note that yy and mm (i.e., number covariates) may have different length.

UMRactiveSet_trust2 *An active set approach to minimizing objective in Unlinked Monotone Regression*

Description

An active set approach to minimizing objective in Unlinked Monotone Regression

Usage

```
UMRactiveSet_trust2(
  yy,
  ww_y = NULL,
  grad,
  hess,
  UMR_curv,
  CDF,
  init,
  counts = rep(1, length(init)),
  stepsize,
  MM,
  tol_end = 1e-04,
  tol_collapse,
  printevery,
  filename
)
```

Arguments

yy	Y (response) observation vector (numeric)
ww_y	Weights (nonnegative, sum to 1) corresponding to yy. Samelength as yy. Or NULL in which yy are taken as being evenly weighted.
grad	Is function(mm, ww_m). (Will be defined based on yy [and maybe ww_y] before being passed in.) Returns vector of length(mm). Gradient of objective function.

hess	Is function(mm, ww_m). (Will be defined based on yy [and maybe ww_y] before being passed in.) Returns matrix of dimensions length(mm) by length(mm). Hessian of objective function.
UMR_curv	A curvature function object (giving mathfrak(C) in the paper; and related to "C" in the paper). See UMR_curv_generic() and examples. This is generally a "curried" version of UMR_curv_generic with densfunc and BBp passed in.
CDF	This is the error (cumulative) distribution function, a function object. Function accepting vector or matrix arguments.
init	Initial value of estimate ('mm'). Vector, length may be different than length(yy). See 'counts' input.
counts	Together 'init' and 'counts' serve as the initialization; the implied initial vector is rep.int(init, counts).
stepsize	Stepsize for moving out of saddle points.
MM	A number of iterations. May not use them all. MM is not exactly the total number of iterations used in the sense that within each of MM iterations, we will possibly run another algorithm which may take up to MM iterations (but usually takes many fewer).
tol_end	Used as tolerance at various points . Generally algorithm (and some subalgorithms) end once sum(abs(mm-mmprev)) < tol, or you hit MM iterations.
tol_collapse	Collapsing roughly equal mm values into each other.
printevery	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
filename	filename (path) to save output to.

Details

Uses first order (gradient) for optimization, and uses certain second derivative computations to leave saddle points. See Balabdaoui, Doss, and Durot (2021). Note that yy and mm (i.e., number covariates) may have different length.

dens and bbp are deprecated

param dens This is the error density, a function object. Function accepting vector or matrix arguments.

param BBp This is derivative of "B" function ("B prime"), where B is defined in the paper (Balabdaoui, Doss, Durot (2020+)). Function accepting vector or matrix arguments.

UMRgradDesc *Basic gradient descent implementation*

Description

Basic gradient descent implementation

Usage

```
gradDesc(yy, grad, init, stepsize, MM, printevery, filename)
```

Arguments

yy	Y (response) observation vector (numeric)
grad	a function(yy, mm) where mm is same length of yy and is the previous iterate value (i.e., the estimate vector).
init	Initial value of estimate ('mm'). I.e., numeric vector of same length as yy.
stepsize	Gradient descent stepsize. Set carefully! (I often use nn^2 / 2 where nn = length(yy), or nn if gradient is 'rescaled'.)
MM	Number of iterations
printevery	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
filename	filename (path) to save output to.

Details

Implements a very basic gradient descent. Right now stepsize is fixed.

Examples

```
#### Set up the gradient function
 mysig <- 1 ##  std dev
errdist <- distr::Norm(0, sd=mysig)
modeldistname <- truedistname <- "Gauss" ## used for savefile name
mm0 <- function(xx){xx}
nn <- 300
xx <- sort(runif(n=nn, 0, 7))
yy <- mm0(xx) + errdist@r(nn)
## plot(xx,yy)

myScale <- mysig

AAfunc_Gauss <- purrr::partial(AAfunc_Gauss_generic, sig=!!mysig)
AA_Gauss <- purrr::partial(AA, func=!!AAfunc_Gauss)
BBfunc_Gauss <- purrr::partial(BBfunc_Gauss_generic, sig=!!mysig)
BB_Gauss <- purrr::partial(BB, func=!!BBfunc_Gauss)
mygradSIR <-
    grad_SIR_Gauss <- ## just for ease of reference
        purrr::partial(grad_SIR_generic,
                       rescale=TRUE, ## factor of nn/2
                       AAfunc=!!AA_Gauss, BBfunc=!!BB_Gauss)

 ## Now run the gradient descent
savefilenameUnique <- paste("graddesc_", modeldistname, "_", truedistname,
                            "_n", nn,
```

```
                            "_", format(Sys.time(), "%Y-%m-%d-%T"), ".rsav", sep="")
print(paste("The unique save file name for this run is", savefilenameUnique))
stepsize <- nn^(1/2) ## Has to be tuned
MM <-  100 ## Total number iterations is MM * JJ
JJ <- 2
eps <- (max(yy)-min(yy)) / (1000 * nn^(1/5) * myScale)
## print *and* SAVE every 'printevery' iterations.
## here no save occurs, printevery > MM
printevery <- 1000
init <- yy

mmhat <- UMRgradDesc(yy=yy, grad=mygradSIR, ## from settings file
                     init=init,
                     stepsize=stepsize, MM=MM,
                     printevery=printevery,
                     filename=paste0("../saves/", savefilenameUnique))
#### some classical/matched [oracle] estimators
isoreg_std <- Iso::ufit(y=yy, x=xx, lmode=Inf)
mmhat_std = isoreg_std$y ## Isotonic regression
linreg_std <- lm(yy~xx)
```

`UMRgradDesc_fixed_df` *Gradient Descent with a fixed number of constant pieces (degrees of freedom)*

Description

Gradient Descent with a fixed number of constant pieces (degrees of freedom)

Usage

```
UMRgradDesc_fixed_df(
  grad,
  init,
  stepsize,
  MM,
  tol = 1e-07,
  printevery = Inf,
  filename
)
```

Arguments

`grad`	a function(mm) where mm is the previous iterate value (i.e., the estimate vector).
`init`	Initial value of estimate ('mm'). The output will be of length length(init).
`stepsize`	Gradient descent stepsize. Set carefully!

MM	Number of iterations in which "support reduction" (combining of approximately equal values into a region of constancy) is done (see details and paper). Depending on tol, may not use all MM iterations.
tol	Tolerance: end algorithm once sum(abs(mm-mmprev)) < tol or you hit MM iterations.
printevery	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
filename	path1/path2/filename to save output to.

Details

UMRgradDesc_fixed_df does a gradient descent with a fixed (upper bound) on the number of constant segments of the function.

Output of UMRgradDesc_fixed_df is unsorted. Note weights for 'mm' are not passed in; rather they will be contained/used in grad().

UMRgradDesc_PC *Gradient Descent implemented for Piecewise Constant functions*

Description

Gradient Descent implemented for Piecewise Constant functions

Usage

```
UMRgradDesc_PC(
  yy,
  grad,
  init,
  stepsize,
  MM,
  eps,
  JJ = 50,
  printevery,
  filename
)
```

Arguments

yy	Y (response) observation vector (numeric)
grad	a function(yy, mm) where mm may be shorter length than yy and is the previous iterate value (i.e., the estimate vector).
init	Initial value of estimate ('mm'). I.e., numeric vector usually of same length as yy.
stepsize	Gradient descent stepsize. Set carefully!

MM	Number of iterations in which "support reduction" (combining of approximately equal values into a region of constancy) is done (see details and paper).
eps	Roughly, points that are eps apart are considered to be equal and are thus collapsed into a single region of piecewise constancy of the output. (This is not precisely true because one can have a long sorted-increasing vector of points that are each eps from their two neighboring points but such that the first and last points are not eps apart. See algorithm description in paper for details.)
JJ	Total number of gradient steps is MM*JJ. JJ gradient steps are taken for each of the MM steps.
printevery	integer value (generally « MM). Every 'printevery' iterations, a count will be printed and the output saved.
filename	path1/path2/filename to save output to.

Details

Implements a gradient descent. See paper for details. Right now stepsize is fixed. Right now: init gets sorted in gradDesc_PC so does not need to be sorted on input. Roughly, the difference between this algorithm and gradDesc() (which is just vanilla gradient descent on this problem) is that: if mm is the current value of the output estimate, then gradDesc_PC 'collapses' or combines values of mm that are (roughly, up to tolerance 'eps') equal. Because the solution is generally piecewise constant with a relatively small number of constant regions this enormously speeds up the later stages of the algorithm. Note that once points are combined/collapsed they contribute identically to the objective function, so they will never be "uncombined".

Examples

```
#'
#### Set up the gradient function
 mysig <- 1 ##  std dev
errdist <- distr::Norm(0, sd=mysig)
modeldistname <- truedistname <- "Gauss" ## used for savefile name
mm0 <- function(xx){xx}
nn <- 300
xx <- sort(runif(n=nn, 0, 7))
yy <- mm0(xx) + errdist@r(nn)
## plot(xx,yy)

myScale <- mysig

AAfunc_Gauss <- purrr::partial(AAfunc_Gauss_generic, sig=!!mysig)
AA_Gauss <- purrr::partial(AA, func=!!AAfunc_Gauss)
BBfunc_Gauss <- purrr::partial(BBfunc_Gauss_generic, sig=!!mysig)
BB_Gauss <- purrr::partial(BB, func=!!BBfunc_Gauss)
mygradSIR <-
    grad_SIR_Gauss <- ## just for ease of reference
        purrr::partial(grad_SIR_generic,
                       rescale=TRUE, ## factor of nn/2
                       AAfunc=!!AA_Gauss, BBfunc=!!BB_Gauss)
```

```
 ## Now run the gradient descent
savefilenameUnique <- paste("graddesc_", modeldistname, "_", truedistname,
                            "_n", nn,
                            "_", format(Sys.time(), "%Y-%m-%d-%T"), ".rsav", sep="")
print(paste("The unique save file name for this run is", savefilenameUnique))
stepsize <- nn^(1/2) ## Has to be tuned
MM <-  200 ## Total number iterations is MM * JJ
JJ <- 2
eps <- (max(yy)-min(yy)) / (1000 * nn^(1/5) * myScale)
## print *and* SAVE every 'printevery' iterations;
## here no save occurs, printevery > MM
printevery <- 1000
init <- yy


 mmhat <- UMRgradDesc_PC(yy=yy, grad=mygradSIR, ## from settings file
                      init=init,
                      stepsize=stepsize, MM=MM,
                      JJ=JJ, eps=eps,
                      printevery=printevery,
                      filename=paste0("../saves/", savefilenameUnique))
####  some classical/matched [oracle] estimators
isoreg_std <- Iso::ufit(y=yy, x=xx, lmode=Inf)
mmhat_std = isoreg_std$y ## Isotonic regression
linreg_std <- lm(yy~xx)
```

UMRgrad_generic *Gradient of least-squares Shuffled Isotonic Regression criterion*

Description

Gradient of least-squares Shuffled Isotonic Regression criterion

Usage

```
UMRgrad_generic(
  yy,
  ww_y = rep(1/length(yy), length(yy)),
  mm,
  ww_m = rep(1/length(mm), length(mm)),
  AAfunc,
  BBfunc
)

grad_SIR_generic(
  yy,
  mm,
  counts = rep(1, length(mm)),
```

```
  AAfunc,
  BBfunc,
  rescale = FALSE
)
```

Arguments

yy	Y (response) observation vector (numeric)
ww_y	Weight vector for yy.
mm	Current (unsorted) estimate/iterate at which to compute gradient. (Length equals length of yy).
ww_m	Weight vector for mm.
AAfunc	This is the function "A" defined in the gradient calculations in the paper (Balabdaoui, Doss, Durot (2020+).
BBfunc	This is the function "B" defined in the gradient calculations in the paper (Balabdaoui, Doss, Durot (2020+). @details Returns gradient as a column matrix. See calculations in the paper. @examples #### See help for gradDesc_PC, gradDesc, or grad_helpers
counts	If the function that mm represents is piecewise constant, then mm may be passed in as only the unique entries. In that case counts contains the number of times each element of mm is repeated. Thus length(counts)==length(mm). (Default for counts is thus a vector of all 1's.)
rescale	Boolean: if False then the final return value is the

UMRhess *Compute Hessian of Unlinked Monotone Regression objective function from Balabdaoui, Doss, and Durot*

Description

Compute Hessian of Unlinked Monotone Regression objective function from Balabdaoui, Doss, and Durot

Usage

```
UMRhess_generic(mm, ww_m, yy, ww_y = rep(1/length(yy), length(yy)), dens, BBp)
```

Arguments

mm	Current (unsorted) estimate/iterate at which to compute gradient. (Length is <= than the number of X observations in the problem).
ww_m	Weights (nonnegative, sum to 1) corresponding to mm. Same length as mm.
yy	Y (response) observation vector (numeric)

ww_y	Weights (nonnegative, sum to 1) corresponding to yy. Same length as yy. Default is just 1/length(yy) for each value.
dens	This is the error density, a function object (Balabdaoui, Doss, Durot (2020+). Function accepting vector or matrix arguments.
BBp	This is derivative of "B" function ("B prime"), where B is defined in the paper. Function accepting vector or matrix arguments.

Details

See paper for derivations.

`UMR_curv_generic` *@title Second derivative computations of least-squares Unlinked Isotonic Regression criterion ("SIR" comes from "shuffled isotonic regression" although this terminology is now outdated).*

Description

@title Second derivative computations of least-squares Unlinked Isotonic Regression criterion ("SIR" comes from "shuffled isotonic regression" although this terminology is now outdated).

Usage

```
UMR_curv_generic(
  yy,
  mm,
  ww_y = rep(1/length(yy), length(yy)),
  ww_m = rep(1/length(mm), length(mm)),
  densfunc,
  BBpfunc
)

UMR_curv_generic2(
  yy,
  mm,
  ww_y = rep(1/length(yy), length(yy)),
  ww_m = rep(1/length(mm), length(mm)),
  densfunc,
  DDfunc
)

UMR_CC_generic(
  yy,
  mm,
  ww_y = rep(1/length(yy), length(yy)),
  ww_m = rep(1/length(mm), length(mm)),
```

```
  densfunc,
  DDfunc
)
```

Arguments

yy	Y (response) observation vector (numeric)
mm	Current (unsorted) estimate/iterate at which to compute gradient. (Length is <= than the number of X observations in the problem).
ww_y	Weights (nonnegative, sum to 1) corresponding to yy. Same length as yy. Default is just 1/length(yy) for each value.
ww_m	Weights (nonnegative, sum to 1) corresponding to mm. Same length as mm.
densfunc	This is the error density, a function object (Balabdaoui, Doss, Durot (2021+).
BBpfunc	This is the function B', i.e. derivative of "B" function in the paper. @details The "CC" or "curv" functions are used to be passed in to UMRactiveSet_trust() (generally after 'currying'/substituting in for the parameter arguments). UMR_CC_generic returns a 1xlength(mm) matrix giving the C function defined in the paper. UMR_curv_generic is returning also a 1xlength(mm) matrix giving the (d^2/dtheta^2)(objective function), where "theta" is as defined in the paper. [This is mathfrakC in the paper.] These are similar quantities, the "curv" quantity is just C rescaled by the weight. See calculations in paper. The more substantive difference is that UMR_CC_generic requires a closed form for the "D" function whereas UMR_curv_generic simply uses the hessian computation (i.e., requires B', the derivative of the "B" function). (The closed form of the "D" function can be found from the closed form of the hessian, but it is not necessary.) UMR_curv_generic2 is analogous to UMR_curv_generic but the latter relies on UMR_CC_generic. UMR_CC_generic1 is analogous to UMR_CC_generic (aka CC_SIR_generic) but the former is calculated in fashion identical to UMR_curv_generic (i.e., relying on UMRhess). DDfunc_Gauss_generic is the "D" function that can be passed in (after substituting for sig) for DDfunc in various other functions to compute the "C" function (e.g., UMR_CC_generic). Note: "CC" and "DD", etc., refer to the "C" or "D" functions. Double lettering is a convention often used in the code to refer to the single letter.
DDfunc	This is the function "D" defined in the second derivative calculations in the paper (Balabdaoui, Doss, Durot (2021+).

---

umr_deconv *Carpentier and Schluter 2016 deconvolution method for unmatched monotone regression*

Description

Carpentier and Schluter 2016 deconvolution method for unmatched monotone regression

Usage

```
umr_deconv(xx, yy, sig, error = "normal", bw = "dboot1", adjust = 1, n = 512)

quant_deconv(
  yy,
  sig,
  error = "normal",
  bw = "dboot1",
  adjust = 1,
  n = 512,
  monotonize = base::cummax
)
```

Arguments

xx	X (covariate or predictor) observation vector
yy	Y (response) observation vector (numeric)
sig	standard deviation of epsilon (passed to DeconCdf)
error	Must be "normal" or "laplacian" or "snormal"; see help("DeconCdf")
bw	Bandwidth choice or method for kernel estimator; see help("DeconCdf")
adjust	See help("DeconCdf")
n	See help("DeconCdf")
monotonize	is a function taking a numeric vector argument which returns an increasing numeric vector of the same length. This is used to monotonize the output of the CDF from deconvolution, which is not guaranteed to be a "bona-fide" CDF in the sense that it may not be monotone.

Details

quant_deconv implements Carpentier and Schluter 2016 deconvolution method for unmatched monotone regression, using deconv package. Note that because the DeconCdf() function computes the CDF but there is no direct code for computing the quantile function, we use approxfun to create the quantile function; this may be slow. quant_deconv() returns a vector of length length(yy). Then umr_deconv is a wrapper for quant_deconv. NOTE: It returns the output of approxfun, which is may change over time. The output value is of type function. We linearly interpolate between the points i/n.

Examples

```
library(distr)
mysig <- 1 ##  std dev
```

```
errdist <- distr::Norm(0, sd=mysig)
mm0 <- function(xx){xx}
 nn <- 300
 xx <- sort(runif(n=nn, 0, 7))
 yy <- mm0(xx) + errdist@r(nn)
 ## plot(xx,yy)
 modeldistname <- truedistname <- "Gauss" ## used for savefile name
myScale <- mysig

xx <- sort(runif(n=nn, 0, 7))
mmtrue <- mm0(xx)
yy <- mmtrue + errdist@r(nn)
plot(xx,yy)
qq <- quant_deconv(yy, sig=1, error="normal")
lines(xx, ## already sorted
     qq)
```

Index