Package ‘RGPR’

May 6, 2015

Type Package

Title GPR Processing

Version 1.0

Date 2015-05-06

Author Who wrote it

Maintainer Who to complain to <yourfault@somewhere.net>

Description More about what it does (maybe more than one line)

License What license is it under?

Depends methods

R topics documented:

RGPR-package 5
acfmtx 6
addArg 6
addProfile3D 7
ann 8
ann-methods 9
ann<- 9
ann<–methods 10
apply-methods 10
Arith-methods 10
ar_fb 11
as.matrix-methods 11
byte2volt 12
clip 12
clip-methods 13
coerce-methods 13
convmtx 13
convolution 14
convolution2D 14
coord 15
coord-methods 16
coord<- 16
coord<–methods 16
coords<- 17
coords<–methods 17

crs . . . 18
crs-methods . . . 18
crs<- . . . 18
crs<–methods . . . 19
dcshift . . . 19
dcshift-methods . . . 20
deconvFreq . . . 20
decon_spiking . . . 21
decon_spiking_matrix . . . 22
delineate . . . 23
delineate-methods . . . 24
delineations . . . 24
delineations-methods . . . 25
depth0 . . . 25
depthToTime . . . 25
description . . . 26
description-methods . . . 26
dewow . . . 27
dewow-methods . . . 27
dewow2 . . . 28
dewow2-methods . . . 28
dim-methods . . . 28
doubleVector . . . 29
dx_gkernel . . . 29
dy_gkernel . . . 30
eps . . . 31
exportDelineations . . . 31
exportDelineations-methods . . . 32
exportFID . . . 32
exportFID-methods . . . 33
exportPDF . . . 33
exportPDF-methods . . . 34
extension . . . 34
fid . . . 34
fid-methods . . . 35
fid<- . . . 35
fid<–methods . . . 36
fidpos . . . 36
filename . . . 36
filename-methods . . . 37
firstBreack . . . 37
firstBreack-methods . . . 38
firstBreackPicking . . . 38
FKFilter . . . 39
fkFilter . . . 40
fkFilter-methods . . . 40
FKSpectrum . . . 41
freqFilter . . . 42
freqFilter-methods . . . 42
freqFilter1D . . . 43
fx_deconv . . . 45
gain . . . 46

gain-methods . . . 47
gain_agc . . . 47
gain_exp . . . 48
gain_geospreading . . . 48
gammaCorrection . . . 49
gammaCorrection-methods . . . 49
getAmpl . . . 50
getAmpl-methods . . . 50
getData . . . 51
getData-methods . . . 51
getHD . . . 51
gethd . . . 52
gethd-methods . . . 53
getLine . . . 53
getLine-methods . . . 53
get_args . . . 54
gkernel . . . 54
GPR . . . 55
GPR-class . . . 57
GPRsurvey . . . 59
GPRsurvey-class . . . 61
hammingWindow . . . 62
identifyDelineation . . . 62
identifyDelineation-methods . . . 63
inPoly . . . 63
interpTraces . . . 64
interpTraces-methods . . . 64
intersections . . . 65
intersections-methods . . . 65
is_installed . . . 65
length-methods . . . 66
lengthList . . . 66
lineDist . . . 67
load_install_package . . . 67
localOrientation . . . 68
Math-methods . . . 69
max-methods . . . 69
mean-methods . . . 69
medianFilter . . . 70
medianFilter-methods . . . 70
migration . . . 70
migration-methods . . . 71
min-methods . . . 71
minCommon10 . . . 71
myWhich . . . 72
myWhichMin . . . 73
name . . . 73
name-methods . . . 74
ncol-methods . . . 74
nextpower2 . . . 74
normalize . . . 75
nrow-methods . . . 75

paddMatrix 76
pasteArgs 77
phaseRotation 77
plot.GPR 78
plot.GPRsurvey 81
plot3D 83
plot3D-methods 84
plot3DSlice 84
plotAmpl 86
plotAmpl-methods 87
plotArrows 87
plotDelineations 88
plotDelineations-methods 88
plotDelineations3D 89
plotDelineations3D-methods 89
plotLine 90
plotRaster 90
plotTopo 93
plotWig 94
powSpec 97
print.GPR 99
print.GPRsurvey 99
range-methods 100
readDT1 101
readFID 102
readGPR 103
readGPR-methods 103
readTopo 104
repmat 104
requiredPackage 105
reverse 105
reverse-methods 106
rmDelineations<- 106
rmDelineations<–methods 106
selectBBox 107
setCol 107
setCoordref 108
setCoordref-methods 108
setData<- 109
setData<–methods 109
setGenericVerif 109
show-methods 110
showDelineations 110
showDelineations-methods 111
sincMod 111
spec 112
spec-methods 112
spikingFilter 113
summary-methods 114
surveyIntersections 114
surveyIntersections-methods 114
time0 115

time0-methods 115
time0<- 115
time0<–methods 116
timeToDepth 116
topoShift 117
trim 117
upsample 118
upsample-methods 118
wapply 119
winSincKernel 119
writeGPR 120
writeGPR-methods 121
[-methods 121
[<–methods 121

Index **122**

`RGPR-package` *What the package does (short line) ~~ package title ~~*

Description

More about what it does (maybe more than one line) ~~ A concise (1-5 lines) description of the package ~~

Details

Package:	RGPR
Type:	Package
Version:	1.0
Date:	2015-05-06
License:	What license is it under?
Depends:	methods

~~ An overview of how to use the package, including the most important functions ~~

Author(s)

Who wrote it

Maintainer: Who to complain to <yourfault@somewhere.net> ~~ The author and/or maintainer of the package ~~

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
~~ simple examples of the most important functions ~~
```

acfmtx

Usage

```
acfmtx(Y, ...)
```

Arguments

Y

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (Y, ...)
{
    myACF <- apply(ym, 2, acf, ...)
    myACF2 <- do.call(cbind, lapply(myACF, function(x) x$acf))
    return(myACF2)
  }
```

addArg

Usage

```
addArg(proc, arg)
```

Arguments

proc

arg

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (proc, arg)
{
    proc_add <- paste(names(arg), sapply(arg, pasteArgs, arg),
        sep = "=", collapse = "+")
    if (substr(proc, nchar(proc), nchar(proc)) == ":") {
        proc <- paste(proc, proc_add, sep = "")
    }
    else {
        proc <- paste(proc, "+", proc_add, sep = "")
    }
    return(proc)
  }
```

addProfile3D

Usage

```
addProfile3D(LINES, col = diverge_hcl(101, h = c(246, 10), c = 120, l = c(30, 90)), plotNew = FALSE
```

Arguments

LINES

col

plotNew

normalize

v

zlim

AGC

sig

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (LINES, col = diverge_hcl(101, h = c(246, 10), c = 120,
    l = c(30, 90)), plotNew = FALSE, normalize = TRUE, v = 1,
    zlim = NULL, AGC = FALSE, sig = 10)
{
    if (plotNew) {
        open3d()
    }
```

```
  for (i in seq_along(LINES)) {
      lineName2 <- strsplit(LINES, split = "[.]")
      lineName <- lineName2[[i]][1]
      fileNameHD <- paste(lineName, ".HD", sep = "")
      fileNameDT1 <- paste(lineName, ".DT1", sep = "")
      cat(basename(lineName), "\n")
      GPR <- readDT1(LINES[[i]])
    myGPRdZ <- as.numeric(as.character(GPR$hd[7, 2]))/as.numeric(as.character(GPR$hd[5,
          2]))
      HD <- GPR$dt1hd
      A <- GPR$data
      A[is.na(A)] <- 0
      if (!is.null(zlim)) {
          sel <- seq(1, zlim/myGPRdZ/v, by = myGPRdZ)
          A <- A[sel, ]
      }
      if (normalize) {
          A <- normalizeGPR(A)
      }
      if (AGC) {
          A <- apply(A, 2, gain, sig = sig)
      }
      nr = nrow(A)
      nc = ncol(A)
      X <- matrix(HD$recx, ncol = nc, nrow = nr, byrow = TRUE)
      Y <- matrix(HD$recy, ncol = nc, nrow = nr, byrow = TRUE)
      Z <- matrix(HD$topo, ncol = nc, nrow = nr, byrow = TRUE) -
          matrix(myGPRdZ * v * (0:(nr - 1)), ncol = nc, nrow = nr,
              byrow = FALSE)
      if (all(HD$topo == 0)) {
          warning("No topography \n")
      }
      if (all(HD$recx == 0)) {
          warning("No x-coordinates \n")
      }
      if (all(HD$recy == 0)) {
          warning("No y-coordinates \n")
      }
      A = (A - min(A))/(max(A) - min(A))
      Alim <- range(A)
      Alen <- Alim[2] - Alim[1] + 1
      colA <- col[(A) * 100 + 1]
      rgl.surface(X, Y, Z, color = colA, back = "fill", smooth = TRUE,
          lit = FALSE, lwd = 0)
  }
}
```

ann

Usage

```
ann(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("ann"), generic = structure("ann", package = "RGPR"), package = "RGPR", group = list(), value
stop("invalid call in method dispatch to 'ann' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

ann-methods *~~ Methods for Function* ann *~~*

Description

~~ Methods for function ann ~~

Methods

signature(x = "GPR")

ann<-

Usage

```
ann<-(x, values)
```

Arguments

x

values

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, values)
{
    standardGeneric("ann<-")
  }, generic = structure("ann<-", package = "RGPR"), package = "RGPR", group = list(), valueClass = character(
"values"), default = `\001NULL\001`, skeleton = (function (x,
    values)
stop("invalid call in method dispatch to 'ann<-' (no default method)",
   domain = NA))(x, values), class = structure("nonstandardGenericFunction", package = "methods"))
```

`ann<--methods` ~~ *Methods for Function* `ann<-` ~~

Description

~~ Methods for function `ann<-` ~~

Methods

`signature(x = "GPR")`

`apply-methods` ~~ *Methods for Function* `apply` ~~

Description

~~ Methods for function `apply` ~~

Methods

`signature(X = "ANY")`

`signature(X = "GPR")`

`Arith-methods` ~~ *Methods for Function* `Arith` ~~

Description

~~ Methods for function `Arith` ~~

Methods

`signature(e1 = "ANY", e2 = "GPR")`

`signature(e1 = "GPR", e2 = "ANY")`

`signature(e1 = "GPR", e2 = "GPR")`

ar_fb

Usage

```
ar_fb(y, nf, mu = 0.1, type = 1)
```

Arguments

y

nf

mu

type

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, nf, mu = 0.1, type = 1)
{
    if (type == -1) {
        ny <- length(y)
        H <- convmtx(y, nf)[, nf:1]
        d <- numeric(nf + ny - 1)
        d[(nf + 1):(ny + nf - 1)] <- y[1:(ny - 1)]
        f <- solve(t(H) %*% H + mu * diag(nf)) %*% t(H) %*% d
        y_pred <- numeric(ny)
        y_pred[1:(ny - 1)] <- (H %*% f)[(nf + 1):(nf + ny - 1)]
    }
    else if (type == 1) {
        ny <- length(y)
        H <- convmtx(y, nf)
        d <- numeric(nf + ny - 1)
        d[1:(ny - 1)] <- y[2:ny]
        f <- solve(t(H) %*% H + mu * diag(nf)) %*% t(H) %*% d
        y_pred <- numeric(ny)
        y_pred[2:ny] <- (H %*% f)[1:(ny - 1)]
    }
    return(y_pred)
  }
```

as.matrix-methods ~~ *Methods for Function* as.matrix ~~

Description

~~ Methods for function as.matrix ~~

Methods

```
signature(x = "GPR")
```

byte2volt

Usage

```
byte2volt(V = c(-50, 50), nBytes = 16)
```

Arguments

V

nBytes

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (V = c(-50, 50), nBytes = 16)
{
    abs(diff(V))/(2^nBytes)
  }
```

clip

Usage

```
clip(x, Amax = NULL, Amin = NULL)
```

Arguments

x

Amax

Amin

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, Amax = NULL, Amin = NULL)
standardGeneric("clip"), generic = structure("clip", package = "RGPR"), package = "RGPR", group = list(), val
"Amax", "Amin"), default = `\001NULL\001`, skeleton = (function (x,
    Amax = NULL, Amin = NULL)
stop("invalid call in method dispatch to 'clip' (no default method)",
   domain = NA))(x, Amax, Amin), class = structure("standardGeneric", package = "methods"))
```

clip-methods *~~ Methods for Function* clip *~~*

Description

~~ Methods for function clip ~~

Methods

signature(x = "GPR")

coerce-methods *~~ Methods for Function* coerce *~~*

Description

~~ Methods for function coerce ~~

Methods

signature(from = "GPR", to = "matrix")

convmtx

Usage

```
convmtx(y, nf)
```

Arguments

y

nf

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, nf)
{
    ny <- length(y)
    L <- nf + ny - 1
    yext <- rep(c(y, rep(0, L - ny + 1)), nf)
    yext <- yext[1:(L * nf)]
    return(matrix(yext, nrow = L, ncol = nf))
  }
```

convolution

Usage

```
convolution(a, b)
```

Arguments

a

b

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (a, b)
{
    na <- length(a)
    nb <- length(b)
    L <- na + nb - 1
    a0 <- c(a, rep(0, nb - 1))
    b0 <- c(b, rep(0, na - 1))
    y <- Re(fft(fft(a0) * fft(b0), inverse = TRUE))/L
    return(y[1:(max(na, nb))])
  }
```

convolution2D

Usage

```
convolution2D(h, k, bias = 0)
```

Arguments

h

k

bias

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (h, k, bias = 0)
{
    nh = nrow(h)
    mh = ncol(h)
    nk = nrow(k)
    mk = ncol(k)
    if (nk > nh || mk > mh) {
        stop("Kernel 'k' should be smaller than the matrix 'h'\n")
    }
    h0 <- paddMatrix(h, nk, mk)
    nL <- nrow(h0)
    mL <- ncol(h0)
    k0 <- matrix(0, nrow = nL, ncol = mL)
    h0[1:nh, 1:mh] <- h
    k0[1:nk, 1:mk] <- k
    g <- Re(fft(fft(k0) * fft(h0), inverse = TRUE))
    g2 <- g[nk - 1 + 1:nh, mk - 1 + 1:mh]
    return(g2)
  }
```

coord

Usage

```
coord(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("coord"), generic = structure("coord", package = "RGPR"), package = "RGPR", group = list(), v
stop("invalid call in method dispatch to 'coord' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

coord-methods *~~ Methods for Function* coord *~~*

Description

~~ Methods for function coord ~~

Methods

signature(x = "GPR")

coord<-

Usage

```
coord<-(x, values)
```

Arguments

x

values

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, values)
{
    standardGeneric("coord<-")
  }, generic = structure("coord<-", package = "RGPR"), package = "RGPR", group = list(), valueClass = characte
"values"), default = `\001NULL\001`, skeleton = (function (x,
    values)
stop("invalid call in method dispatch to 'coord<-' (no default method)",
   domain = NA))(x, values), class = structure("nonstandardGenericFunction", package = "methods"))
```

coord<--methods *~~ Methods for Function* coord<- *~~*

Description

~~ Methods for function coord<- ~~

Methods

signature(x = "GPR")

coords<-

Usage

```
coords<-(x, values)
```

Arguments

x

values

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, values)
{
    standardGeneric("coords<-")
  }, generic = structure("coords<-", package = "RGPR"), package = "RGPR", group = list(), valueClass = charact
"values"), default = `\001NULL\001`, skeleton = (function (x,
    values)
stop("invalid call in method dispatch to 'coords<-' (no default method)",
   domain = NA))(x, values), class = structure("nonstandardGenericFunction", package = "methods"))
```

coords<--methods ~~ *Methods for Function* coords<- ~~

Description

~~ Methods for function coords<- ~~

Methods

signature(x = "GPRsurvey")

crs

Usage

```
crs(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("crs"), generic = structure("crs", package = "RGPR"), package = "RGPR", group = list(), value
stop("invalid call in method dispatch to 'crs' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

crs-methods ~~ *Methods for Function* `crs` ~~

Description

~~ Methods for function `crs` ~~

Methods

`signature(x = "GPR")`

crs<-

Usage

```
crs<-(x, value)
```

Arguments

x

value

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, value)
{
    standardGeneric("crs<-")
  }, generic = structure("crs<-", package = "RGPR"), package = "RGPR", group = list(), valueClass = character(
"value"), default = `\001NULL\001`, skeleton = (function (x,
    value)
stop("invalid call in method dispatch to 'crs<-' (no default method)",
   domain = NA))(x, value), class = structure("nonstandardGenericFunction", package = "methods"))
```

crs<--methods ~~ *Methods for Function* crs<- ~~

Description

~~ Methods for function crs<- ~~

Methods

signature(x = "GPR")

dcshift

Usage

```
dcshift(x, u)
```

Arguments

x

u

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, u)
standardGeneric("dcshift"), generic = structure("dcshift", package = "RGPR"), package = "RGPR", group = list(
"u"), default = `\001NULL\001`, skeleton = (function (x, u)
stop("invalid call in method dispatch to 'dcshift' (no default method)",
    domain = NA))(x, u), class = structure("standardGeneric", package = "methods"))
```

dcshift-methods ~~ *Methods for Function* dcshift ~~

Description

~~ Methods for function dcshift ~~

Methods

signature(x = "GPR")

deconvFreq

Usage

```
deconvFreq(y, h, mu = 1e-04)
```

Arguments

y

h

mu

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, h, mu = 1e-04)
{
    ny <- length(y)
    nh <- length(h)
    L <- ny + ny - 1
    H <- fft(c(h, rep(0, ny - 1)))
    Y <- fft(c(y, rep(0, nh - 1)))
    Re(fft(t(Conj(H)) * Y/(t(Conj(H)) * H + mu), inverse = TRUE))[1:ny]/L
  }
```

decon_spiking

Usage

```
decon_spiking(ym, nf = 60, mu = 1e-04, shft = 1, phase_rot = FALSE)
```

Arguments

ym

nf

mu

shft

phase_rot

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ym, nf = 60, mu = 1e-04, shft = 1, phase_rot = FALSE)
{
    ym_max <- max(abs(ym))
    ym <- ym/ym_max
    ny <- length(ym)
    f_min <- spikingFilter(ym, nf, mu = mu)
    v <- numeric(nf)
    v[shft] <- 1
    w_min <- deconvFreq(v, f_min, mu = mu)
    x_dec <- convolution(f_min, ym)
    x_dec <- x_dec[1:ny]
    phi_max <- NULL
    if (phase_rot) {
        pi_seq <- seq(0, pi, by = 0.001)
        kurt <- numeric(length(pi_seq))
        for (i in seq_along(pi_seq)) {
            xrot <- phaseRotation(x_dec, pi_seq[i])
            kurt[i] <- sum((xrot - mean(xrot))^4)/((sum((xrot -
                mean(xrot))^2))^2)
        }
        phi_max <- pi_seq[which.max(kurt)]
        cat("rotation angle =", phi_max, "rad\n")
        dev.off()
        windows()
        plot(pi_seq, kurt, type = "l")
        abline(v = phi_max, col = "red")
        x_dec <- phaseRotation(x_dec, phi_max)
    }
    w_mixed <- deconvFreq(ym, x_dec, mu = mu)[1:nf]
    f_mixed <- deconvFreq(x_dec, ym, mu = mu)[1:nf]
    return(list(x = x_dec, w_min = w_min, f_min = f_min, f = f_mixed,
        w = w_mixed, phi = phi_max))
  }
```

decon_spiking_matrix

Usage

```
decon_spiking_matrix(ym, nf = 60, mu = 1e-04, shft = 1, phase_rot = FALSE, myCols = NULL)
```

Arguments

ym

nf

mu

shft

phase_rot

myCols

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ym, nf = 60, mu = 1e-04, shft = 1, phase_rot = FALSE,
    myCols = NULL)
{
    ym_max <- apply(abs(ym), 2, max)
    ym <- t(t(ym)/ym_max)
    ny <- nrow(ym)
    if (is.null(myCols)) {
        myCols <- 1:ncol(ym)
    }
    cat(range(myCols))
    f_min <- apply(ym[, myCols], 2, spikingFilter, nf, mu)
    plot(f_min[, 1], type = "n")
    apply(f_min, 2, lines)
    lines(apply(f_min, 1, median), col = "red")
    v <- numeric(nf)
    v[shft] <- 1
    deconWrap <- function(aa, bb, cc) {
        deconvFreq(bb, aa, cc)
    }
    w_min <- apply(f_min, 2, deconWrap, v, mu)
    w_min2 <- rbind(rep(0, length(myCols)), w_min, rep(0, length(myCols)))
    plot(w_min2[, 1], type = "n", ylim = range(w_min2))
    apply(w_min2, 2, lines)
    abline(h = 0)
    Sys.sleep(2)
    x_dec <- apply(ym[, myCols], 2, convolution, apply(f_min,
      1, median))
  w_mixed <- NULL
  f_mixed <- NULL
  if (phase_rot) {
      pi_seq <- seq(0, pi, by = 0.01)
      kurt <- numeric(length(pi_seq))
      for (i in seq_along(pi_seq)) {
          xrot <- apply(x_dec, 2, phaseRotation, pi_seq[i])
          A <- as.vector(xrot - colMeans(xrot))
          kurt[i] <- sum((xrot - mean(xrot))^4)/((sum((xrot -
              mean(xrot))^2))^2)
      }
      phi_max <- pi_seq[which.max(kurt)]
      cat("rotation angle =", phi_max, "rad\n")
      dev.off()
      windows()
      plot(pi_seq, kurt, type = "l")
      abline(v = phi_max, col = "red")
      Sys.sleep(1)
      w_mixed <- w_min
      f_mixed <- f_min
      for (i in 1:ncol(w_min)) {
          w_mixed[, i] <- phaseRotation(w_min[, i], -phi_max)
          f_mixed[, i] <- phaseRotation(f_mixed[, i], phi_max)
      }
      w_mixed <- rbind(rep(0, length(myCols)), w_mixed, rep(0,
          length(myCols)))
      plot(w_mixed[, 1], type = "n", ylim = range(w_mixed))
      apply(w_mixed, 2, lines)
      abline(h = 0)
      Sys.sleep(2)
      plot(f_mixed[, 1], type = "n")
      apply(f_mixed, 2, lines)
  }
  return(list(w_min = w_min, f_min = f_min, f = f_mixed, w = w_mixed,
      phi = phi_max))
}
```

delineate

Usage

```
delineate(x, name = NULL, type = c("raster", "wiggles"), add_topo = FALSE, upsample = NULL, n = 100
```

Arguments

x

name

type

add_topo

upsample

n

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, name = NULL, type = c("raster", "wiggles"),
    add_topo = FALSE, upsample = NULL, n = 10000, ...)
standardGeneric("delineate"), generic = structure("delineate", package = "RGPR"), package = "RGPR", group = l
"name", "type", "add_topo", "upsample", "n"), default = `\001NULL\001`, skeleton = (function (x,
    name = NULL, type = c("raster", "wiggles"), add_topo = FALSE,
    upsample = NULL, n = 10000, ...)
stop("invalid call in method dispatch to 'delineate' (no default method)",
   domain = NA))(x, name, type, add_topo, upsample, n, ...), class = structure("standardGeneric", package = "
```

delineate-methods ~~ *Methods for Function* delineate ~~

Description

~~ Methods for function delineate ~~

Methods

signature(x = "GPR")

delineations

Usage

```
delineations(x, sel = NULL, ...)
```

Arguments

x

sel

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, sel = NULL, ...)
standardGeneric("delineations"), generic = structure("delineations", package = "RGPR"), package = "RGPR", gro
"sel"), default = `\001NULL\001`, skeleton = (function (x, sel = NULL,
    ...)
stop("invalid call in method dispatch to 'delineations' (no default method)",
   domain = NA))(x, sel, ...), class = structure("standardGeneric", package = "methods"))
```

`delineations-methods` ~~ *Methods for Function* `delineations` ~~

Description

~~ Methods for function `delineations` ~~

Methods

`signature(x = "GPR")`

`depth0`

Usage

```
depth0(time0, v = 0.1, antsep = 1)
```

Arguments

`time0`

`v`

`antsep`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (time0, v = 0.1, antsep = 1)
{
    time0 - antsep/0.299 + antsep/v
  }
```

`depthToTime`

Usage

```
depthToTime(z, time0, v = 0.1, antsep = 1)
```

Arguments

`z`

`time0`

`v`

`antsep`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (z, time0, v = 0.1, antsep = 1)
{
    t0 <- time0 - antsep/0.299
    sqrt((4 * z^2 + antsep^2)/(v^2)) + t0
  }
```

description

Usage

```
description(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("description"), generic = structure("description", package = "RGPR"), package = "RGPR", grou
stop("invalid call in method dispatch to 'description' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

description-methods ~~ *Methods for Function* description ~~

Description

~~ Methods for function description ~~

Methods

signature(x = "GPR")

dewow

Usage

```
dewow(x, w = 100, x0 = 0.1)
```

Arguments

x

w

x0

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, w = 100, x0 = 0.1)
standardGeneric("dewow"), generic = structure("dewow", package = "RGPR"), package = "RGPR", group = list(), v
"w", "x0"), default = `\001NULL\001`, skeleton = (function (x,
    w = 100, x0 = 0.1)
stop("invalid call in method dispatch to 'dewow' (no default method)",
    domain = NA))(x, w, x0), class = structure("standardGeneric", package = "methods"))
```

dewow-methods *~~ Methods for Function* dewow *~~*

Description

~~ Methods for function dewow ~~

Methods

```
signature(x = "GPR")
```

dewow2

Usage

```
dewow2(x, sig = 100)
```

Arguments

x

sig

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, sig = 100)
standardGeneric("dewow2"), generic = structure("dewow2", package = "RGPR"), package = "RGPR", group = list(),
"sig"), default = `\001NULL\001`, skeleton = (function (x, sig = 100)
stop("invalid call in method dispatch to 'dewow2' (no default method)",
    domain = NA))(x, sig), class = structure("standardGeneric", package = "methods"))
```

dewow2-methods ~~ *Methods for Function* dewow2 ~~

Description

~~ Methods for function dewow2 ~~

Methods

```
signature(x = "GPR")
```

dim-methods ~~ *Methods for Function* dim ~~

Description

~~ Methods for function dim ~~

Methods

```
signature(x = "GPR")
```

doubleVector

Usage

```
doubleVector(v, n = 2L)
```

Arguments

v

n

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (v, n = 2L)
{
    if (n > 1) {
        m <- length(v)
        dxpos <- rep(diff(v)/n, n - 1)
        vv <- v[-m] + rep(seq(1, n - 1), each = m - 1) * dxpos
        xvalues <- sort(c(v, vv, v[m] + cumsum(rep(dxpos[length(dxpos)],
            n - 1))))
        xvalues <- xvalues[1:(length(xvalues))]
    }
  }
```

dx_gkernel

Usage

```
dx_gkernel(n, m, sigma = 1)
```

Arguments

n

m

sigma

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, m, sigma = 1)
{
    siz = round((n - 1)/2)
    y = matrix(-siz:siz, n, m)
    siz = (m - 1)/2
    x = matrix(-siz:siz, n, m, byrow = T)
    g = x * exp(-(x^2 + y^2)/(2 * sigma^2))
  }
```

dy_gkernel

Usage

```
dy_gkernel(n, m, sigma = 1)
```

Arguments

n

m

sigma

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, m, sigma = 1)
{
    siz = round((n - 1)/2)
    y = matrix(-siz:siz, n, m)
    siz = (m - 1)/2
    x = matrix(-siz:siz, n, m, byrow = T)
    g = y * exp(-(x^2 + y^2)/(2 * sigma^2))
  }
```

eps

Usage

```
eps(x, ns)
```

Arguments

x

ns

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, ns)
{
    xmean <- c(rep(0, floor(ns/2)), wapply(x, width = ns, by = 1,
        FUN = mean), rep(0, floor(ns/2)))
    xsd <- c(rep(0, floor(ns/2)), wapply(x, width = ns, by = 1,
        FUN = sd), rep(0, floor(ns/2)))
    xtest <- wapply(xsd, width = ns, by = 1, FUN = which.min) +
        (0):(length(xmean) - 2 * floor(ns/2) - 1)
    return(c(rep(0, floor(ns/2)), xmean[xtest], rep(0, floor(ns/2))))
  }
```

exportDelineations

Usage

```
exportDelineations(gpr, path = "")
```

Arguments

gpr

path

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (gpr, path = "")
standardGeneric("exportDelineations"), generic = structure("exportDelineations", package = "RGPR"), package
"path"), default = `\001NULL\001`, skeleton = (function (gpr,
    path = "")
stop("invalid call in method dispatch to 'exportDelineations' (no default method)",
    domain = NA))(gpr, path), class = structure("standardGeneric", package = "methods"))
```

exportDelineations-methods

~~ Methods for Function `exportDelineations` *~~*

Description

~~ Methods for function `exportDelineations` ~~

Methods

`signature(gpr = "GPR")`

exportFID

Usage

```
exportFID(x, filepath = NULL)
```

Arguments

`x`

`filepath`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, filepath = NULL)
standardGeneric("exportFID"), generic = structure("exportFID", package = "RGPR"), package = "RGPR", group = l
"filepath"), default = `\001NULL\001`, skeleton = (function (x,
    filepath = NULL)
stop("invalid call in method dispatch to 'exportFID' (no default method)",
   domain = NA))(x, filepath), class = structure("standardGeneric", package = "methods"))
```

exportFID-methods ~~ *Methods for Function* exportFID ~~

Description

~~ Methods for function exportFID ~~

Methods

signature(x = "GPR")

signature(x = "GPRsurvey")

exportPDF

Usage

```
exportPDF(x, filepath = NULL, add_topo = FALSE, clip = NULL, normalize = NULL, upsample = NULL, ...
```

Arguments

x

filepath

add_topo

clip

normalize

upsample

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, filepath = NULL, add_topo = FALSE, clip = NULL,
    normalize = NULL, upsample = NULL, ...)
standardGeneric("exportPDF"), generic = structure("exportPDF", package = "RGPR"), package = "RGPR", group = l
"filepath", "add_topo", "clip", "normalize", "upsample"), default = `\001NULL\001`, skeleton = (function (x,
    filepath = NULL, add_topo = FALSE, clip = NULL, normalize = NULL,
    upsample = NULL, ...)
stop("invalid call in method dispatch to 'exportPDF' (no default method)",
    domain = NA))(x, filepath, add_topo, clip, normalize, upsample,
    ...), class = structure("standardGeneric", package = "methods"))
```

exportPDF-methods *~~ Methods for Function* exportPDF *~~*

Description

~~ Methods for function exportPDF ~~

Methods

signature(x = "GPR")

extension

Usage

```
extension(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x)
{
    cat("with caution... because split '.' may not be so good\n")
    unlist(lapply(strsplit(basename(x), "[.]"), tail, 1))
  }
```

fid

Usage

```
fid(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("fid"), generic = structure("fid", package = "RGPR"), package = "RGPR", group = list(), value
stop("invalid call in method dispatch to 'fid' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

fid-methods *~~ Methods for Function* fid *~~*

Description

~~ Methods for function `fid` ~~

Methods

```
signature(x = "GPR")
```

fid<-

Usage

```
fid<-(x, values)
```

Arguments

x

values

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, values)
{
    standardGeneric("fid<-")
  }, generic = structure("fid<-", package = "RGPR"), package = "RGPR", group = list(), valueClass = character(
"values"), default = `\001NULL\001`, skeleton = (function (x,
    values)
stop("invalid call in method dispatch to 'fid<-' (no default method)",
   domain = NA))(x, values), class = structure("nonstandardGenericFunction", package = "methods"))
```

fid<--methods *~~ Methods for Function* fid<- ~~

Description

~~ Methods for function fid<- ~~

Methods

signature(x = "GPR")

fidpos

Usage

```
fidpos(xyz, fid)
```

Arguments

xyz

fid

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (xyz, fid)
{
    return(xyz[trim(fid) != "", , drop = FALSE])
  }
```

filename

Usage

```
filename(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("filename"), generic = structure("filename", package = "RGPR"), package = "RGPR", group = lis
stop("invalid call in method dispatch to 'filename' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

filename-methods ~~ *Methods for Function* filename ~~

Description

~~ Methods for function filename ~~

Methods

signature(x = "GPR")

firstBreack

Usage

```
firstBreack(x, nl = 11, ns = NULL, bet = NULL)
```

Arguments

x

nl

ns

bet

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, nl = 11, ns = NULL, bet = NULL)
standardGeneric("firstBreack"), generic = structure("firstBreack", package = "RGPR"), package = "RGPR", group
"nl", "ns", "bet"), default = `\001NULL\001`, skeleton = (function (x,
    nl = 11, ns = NULL, bet = NULL)
stop("invalid call in method dispatch to 'firstBreack' (no default method)",
   domain = NA))(x, nl, ns, bet), class = structure("standardGeneric", package = "methods"))
```

firstBreack-methods ~~ *Methods for Function* firstBreack ~~

Description

~~ Methods for function firstBreack ~~

Methods

signature(x = "GPR")

firstBreackPicking

Usage

```
firstBreackPicking(s, nl = 11, ns = 23, bet = 0.2)
```

Arguments

s

nl

ns

bet

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (s, nl = 11, ns = 23, bet = 0.2)
{
    E1 <- c(wapply(s, width = nl, by = 1, FUN = sum), rep(0,
        2 * floor(nl/2)))
    E2 <- cumsum(s)
    Er <- E1/(E2 + bet)
    Er_fil <- eps(Er, ns = ns)
    first_break <- which.max(abs(diff(Er_fil)))
    return(first_break)
  }
```

FKFilter

Usage

```
FKFilter(A, fk, L = c(5, 5), npad = 1)
```

Arguments

A

fk

L

npad

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, fk, L = c(5, 5), npad = 1)
{
    nr <- nrow(A)
    nc <- ncol(A)
    nk <- npad * (nextpower2(nc))
    nf <- npad * (nextpower2(nr))
    A1 <- matrix(0, nrow = nf, ncol = nk)
    A1[1:nr, 1:nc] <- A
    A1_fft <- fft(A1)
    myFlong <- matrix(0, nrow = nf, ncol = nk)
    myFlong[1:(nf/2), 1:(nk/2)] <- fk[(nf/2):1, (nk/2):1]
    myFlong[(nf/2 + 1):(nf), (nk/2 + 1):nk] <- fk[1:(nf/2), 1:(nk/2)]
    myFlong[1:(nf/2), (nk/2 + 1):nk] <- fk[(nf/2):1, (nk):(nk/2 +
        1)]
    myFlong[(nf/2 + 1):(nf), 1:(nk/2)] <- fk[1:(nf/2), (nk/2 +
        1):nk]
    if (length(L) == 1)
        L <- c(L, L)
    if (all(L != 0)) {
        ham2D = hammingWindow(L[1]) %*% t(hammingWindow(L[2]))
        ham2Dlong = matrix(0, nrow = nf, ncol = nk)
        ham2Dlong[1:L[1], 1:L[2]] <- ham2D
        FF <- Re(fft(fft(myFlong) * fft(ham2Dlong), inv = TRUE))
    }
    else {
        FF <- myFlong
    }
    FF <- FF/sum(FF)
    A_back <- Re(fft(A1_fft * FF, inv = TRUE))[1:nr, 1:nc]
    return(A_back/(max(A_back) - min(A_back)) * (max(A) - min(A)))
  }
```

fkFilter

Usage

```
fkFilter(x, fk = NULL, L = c(5, 5), npad = 1)
```

Arguments

x

fk

L

npad

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, fk = NULL, L = c(5, 5), npad = 1)
standardGeneric("fkFilter"), generic = structure("fkFilter", package = "RGPR"), package = "RGPR", group = lis
"fk", "L", "npad"), default = `\001NULL\001`, skeleton = (function (x,
    fk = NULL, L = c(5, 5), npad = 1)
stop("invalid call in method dispatch to 'fkFilter' (no default method)",
   domain = NA))(x, fk, L, npad), class = structure("standardGeneric", package = "methods"))
```

fkFilter-methods *~~ Methods for Function* fkFilter *~~*

Description

~~ Methods for function fkFilter ~~

Methods

signature(x = "GPR")

FKSpectrum

Usage

```
FKSpectrum(A, dx = 0.25, dz = 0.8, npad = 1, p = 0.01, plot_spec = TRUE, return_spec = FALSE)
```

Arguments

A

dx

dz

npad

p

plot_spec

return_spec

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, dx = 0.25, dz = 0.8, npad = 1, p = 0.01, plot_spec = TRUE,
    return_spec = FALSE)
{
    nr <- nrow(A)
    nc <- ncol(A)
    nk <- npad * (nextpower2(nc))
    nf <- npad * (nextpower2(nr))
    A1 <- matrix(0, nrow = nf, ncol = nk)
    A1[1:nr, 1:nc] <- A
    A1 <- A1 * (-1)^(row(A1) + col(A1))
    A1_fft <- fft(A1)
    A1_fft_pow <- Mod(A1_fft)
    A1_fft_phase <- Arg(A1_fft)
    T = dz * 10^(-9)
    fre <- 1:(nrow(A1_fft_pow)/2)/(2 * (nrow(A1_fft_pow)/2) *
        T)/1e+06
    Ks = 1/dx
    knu <- 1:(ncol(A1_fft_pow)/2)/(2 * (ncol(A1_fft_pow)/2) *
        dx)
    knutot <- c(-rev(knu), knu)
    xat <- c(1, nk/2, nk)
    xLabels <- c(min(knutot), 0, max(knutot))
    yat <- c(1, nf/2, nf)
    yLabels <- c(0, max(fre)/2, max(fre))
    if (plot_spec) {
        plotGPR((A1_fft_pow[1:(nf/2), ])^p, xat = xat, xLabels = xLabels,
            yat = yat, yLabels = yLabels, xlab = "wavenumber (1/m)",
            ylab = "frequency MHz")
    }
    if (return_spec) {
        return(list(pow = A1_fft_pow[1:(nf/2), ], pha = A1_fft_phase[1:(nf/2),
            ]))
    }
  }
```

freqFilter

Usage

```
freqFilter(x, f = 100, type = c("low", "high", "bandpass"), L = 257, plot_spec = FALSE)
```

Arguments

x

f

type

L

plot_spec

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, f = 100, type = c("low", "high", "bandpass"),
    L = 257, plot_spec = FALSE)
standardGeneric("freqFilter"), generic = structure("freqFilter", package = "RGPR"), package = "RGPR", group =
"f", "type", "L", "plot_spec"), default = `\001NULL\001`, skeleton = (function (x,
    f = 100, type = c("low", "high", "bandpass"), L = 257, plot_spec = FALSE)
stop("invalid call in method dispatch to 'freqFilter' (no default method)",
   domain = NA))(x, f, type, L, plot_spec), class = structure("standardGeneric", package = "methods"))
```

freqFilter-methods ~~ *Methods for Function* freqFilter ~~

Description

~~ Methods for function freqFilter ~~

Methods

signature(x = "GPR")

freqFilter1D

Usage

```
freqFilter1D(A, f = c(100), type = c("low", "high", "bandpass"), L = 257, T = 0.8, plot_spec = FALS
```

Arguments

A

f

type

L

T

plot_spec

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, f = c(100), type = c("low", "high", "bandpass"),
    L = 257, T = 0.8, plot_spec = FALSE)
{
    type = match.arg(type)
    A <- as.matrix(A)
    M = nrow(A)
    Ts = T * 10^(-9)
    Fs = 1/Ts
    f = sort(f) * 10^6
    if (type == "low" || type == "high") {
        if (length(f) > 1) {
            BW = (f[2] - f[1])/Fs
            fc = f[1] + (f[2] - f[1])/2
            L = 4/BW
            L = round(L)
            if (L%%2 == 0)
                L = L + 1
        }
        else if (length(f) == 1) {
            fc = f[1]
        }
        h <- winSincKernel(L, fc/Fs, type)
    }
    else if (type == "bandpass") {
        if (length(f) == 2) {
            h1 <- winSincKernel(L, f[1]/Fs, "low")
            h2 <- winSincKernel(L, f[2]/Fs, "high")
        }
        else if (length(f) == 4) {
            BW = (f[2] - f[1])/Fs
            fc = f[1] + (f[2] - f[1])/2
            L = 4/BW
            L = round(L)
            if (L%%2 == 0)
                L = L + 1
            h1 <- winSincKernel(L, fc/Fs, "low")
            BW = (f[4] - f[3])/Fs
            fc = f[3] + (f[4] - f[3])/2
            L = 4/BW
            L = round(L)
            if (L%%2 == 0)
                L = L + 1
            h2 <- winSincKernel(L, fc/Fs, "high")
        }
        L = max(length(h1), length(h2))
        cat("lenght max", L, "\n")
        if (length(h2) < L) {
            h2 = c(rep(0, (L - length(h2))/2), h2, rep(0, (L -
                length(h2))/2))
        }
        if (length(h1) < L) {
            h1 = c(rep(0, (L - length(h1))/2), h1, rep(0, (L -
                length(h1))/2))
        }
        h = -h1 - h2
        h[(L + 1)/2] = h[(L + 1)/2] + 1
    }
    Nfft = 2^(ceiling(log2(L + M - 1)))
    h_long = c(h, rep(0, Nfft - L))
    A = rbind(as.matrix(A), matrix(0, nrow = Nfft - M, ncol = ncol(A)))
    fft_A = mvfft(A)
    fft_h = fft(h_long)
    Y = fft_A * fft_h
    if (type == "bandpass") {
    }
    pow_A = Mod(fft_A)
    pow_h = Mod(fft_h)
    pow_y = Mod(Y)
    if (!is.null(dim(A))) {
        pow_A = apply(pow_A, 1, mean, na.rm = T)
        pow_y = apply(pow_y, 1, mean, na.rm = T)
    }
    pow_A = pow_A[1:(Nfft/2 + 1)]
    pow_y = pow_y[1:(Nfft/2 + 1)]
    pow_h = pow_h[1:(Nfft/2 + 1)]
    fre = Fs * (0:(Nfft/2))/Nfft/1e+06
    if (plot_spec == TRUE) {
        m = seq(0, 900, by = 50)
        par(mar = c(0, 4, 0.3, 2) + 0.1, oma = c(3, 2, 1, 2))
        plot(fre, pow_A, type = "l", xaxt = "n", ylim = c(0,
            max(pow_A, pow_y)), ylab = "power", lwd = 2)
        lines(fre, pow_y, type = "l", col = "blue", lwd = 2)
        Axis(side = 1, tcl = +0.3, labels = m, at = m)
        par(new = TRUE)
        plot(fre, pow_h, type = "l", col = "red", yaxt = "n",
            ylab = "")
        legend("topright", c("input signal", "filter", "filtered signal"),
            col = c("black", "red", "blue"), lwd = c(2, 1, 2),
            bg = "white")
        abline(v = f/1e+06, col = "grey", lty = 2)
    }
    a = (L - 1)/2
    y = mvfft(Y, inverse = TRUE)
    y = y[a:(a + M - 1), ]/nrow(y)
    return(Re(y))
  }
```

fx_deconv

Usage

```
fx_deconv(Y, nf, mu = 0.1, flow = NULL, fhigh = NULL, dz, type = 1)
```

Arguments

Y

nf

mu

flow

fhigh

dz

type

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (Y, nf, mu = 0.1, flow = NULL, fhigh = NULL, dz, type = 1)
{
    npts <- nrow(Y)
    npos <- ncol(Y)
    nfft <- nextpower2(npts)
    Y0 <- matrix(0, nrow = nfft, ncol = npos)
    FX_pred <- Y0
    FX_pred_b <- Y0
    Y0[1:npts, 1:npos] <- Y
    FX <- mvfft(Y0)
    if (is.null(flow)) {
        ilow <- 1
    }
    else {
        ilow <- floor(flow * dz * nfft) + 1
        ilow <- ifelse(ilow < 1, 1, ilow)
    }
```

```
        if (is.null(fhigh)) {
            ihigh <- floor(nfft/2) + 1
        }
        else {
            ihigh <- floor(fhigh * dz * nfft) + 1
            ihigh <- ifelse(ihigh > floor(nfft/2) + 1, floor(nfft/2) +
                1, ihigh)
        }
        for (k in ilow:ihigh) {
            FX_pred[k, ] <- ar_fb(FX[k, ], nf = nf, mu = mu, type = 1)
            FX_pred_b[k, ] <- ar_fb(FX[k, ], nf = nf, mu = mu, type = -1)
        }
        for (k in (nfft/2 + 2):nfft) {
            FX_pred[k, ] <- Conj(FX_pred[nfft - k + 2, ])
            FX_pred_b[k, ] <- Conj(FX_pred_b[nfft - k + 2, ])
        }
        Y_pred_f <- Re(mvfft(FX_pred, inverse = TRUE))/nfft
        Y_pred_b <- Re(mvfft(FX_pred_b, inverse = TRUE))/nfft
        Y_pred <- Y_pred_f[1:npts, ] + Y_pred_b[1:npts, ]
        Y_pred[, (nf + 1):(npos - nf)] <- Y_pred[, (nf + 1):(npos -
            nf)]/2
        return(Y_pred)
      }
```

gain

Usage

```
gain(x, type = c("geospreading", "exp", "agc"), ...)
```

Arguments

x

type

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, type = c("geospreading", "exp", "agc"),
    ...)
standardGeneric("gain"), generic = structure("gain", package = "RGPR"), package = "RGPR", group = list(), val
"type"), default = `\001NULL\001`, skeleton = (function (x, type = c("geospreading",
    "exp", "agc"), ...)
stop("invalid call in method dispatch to 'gain' (no default method)",
   domain = NA))(x, type, ...), class = structure("standardGeneric", package = "methods"))
```

gain-methods ~~ *Methods for Function* gain ~~

Description

~~ Methods for function gain ~~

Methods

signature(x = "GPR")

gain_agc

Usage

```
gain_agc(A, d_t, sig = 10, p = 2, r = 0.5)
```

Arguments

A

d_t

sig

p

r

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, d_t, sig = 10, p = 2, r = 0.5)
{
    sig <- sig/d_t
    Anew <- apply(A, 2, .gain_agc, sig, p, r)
    s1 = ((max(A)) - (min(A)))
    s2 = ((max(Anew)) - (min(Anew)))
    return(Anew * s1/s2)
  }
```

gain_exp

Usage

```
gain_exp(A, alpha, d_t, t_0 = NULL, t_end = NULL)
```

Arguments

A

alpha

d_t

t_0

t_end

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, alpha, d_t, t_0 = NULL, t_end = NULL)
{
    g <- .gain_exp(A[, 1], alpha, d_t, t_0, t_end)
    Anew <- (A) * g
    s1 = ((max(A)) - (min(A)))
    s2 = ((max(Anew)) - (min(Anew)))
    s12 <- s1/s2
    A3 <- (Anew * s12)
    return(Anew)
  }
```

gain_geospreading

Usage

```
gain_geospreading(A, alpha, d_t, t_0 = NULL, t_end = NULL, t_cst = NULL)
```

Arguments

A

alpha

d_t

t_0

t_end

t_cst

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, alpha, d_t, t_0 = NULL, t_end = NULL, t_cst = NULL)
{
    g <- .gain_geospreading(A[, 1], alpha, d_t, t_0, t_end, t_cst)
    Anew <- (A) * g
    s1 = ((max(A)) - (min(A)))
    s2 = ((max(Anew)) - (min(Anew)))
    return(Anew * s1/s2)
  }
```

gammaCorrection

Usage

```
gammaCorrection(x, a = 1, b = 1)
```

Arguments

x

a

b

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, a = 1, b = 1)
standardGeneric("gammaCorrection"), generic = structure("gammaCorrection", package = "RGPR"), package = "RGP
"a", "b"), default = `\001NULL\001`, skeleton = (function (x,
    a = 1, b = 1)
stop("invalid call in method dispatch to 'gammaCorrection' (no default method)",
    domain = NA))(x, a, b), class = structure("standardGeneric", package = "methods"))
```

gammaCorrection-methods

~~ Methods for Function gammaCorrection *~~*

Description

~~ Methods for function gammaCorrection ~~

Methods

signature(x = "GPR")

getAmpl

Usage

```
getAmpl(x, FUN = mean, ...)
```

Arguments

`x`

`FUN`

`...`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, FUN = mean, ...)
standardGeneric("getAmpl"), generic = structure("getAmpl", package = "RGPR"), package = "RGPR", group = list(
"FUN"), default = `\001NULL\001`, skeleton = (function (x, FUN = mean,
    ...)
stop("invalid call in method dispatch to 'getAmpl' (no default method)",
   domain = NA))(x, FUN, ...), class = structure("standardGeneric", package = "methods"))
```

getAmpl-methods *~~ Methods for Function* getAmpl *~~*

Description

~~ Methods for function `getAmpl` ~~

Methods

`signature(x = "GPR")`

getData

Usage

```
getData(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("getData"), generic = structure("getData", package = "RGPR"), package = "RGPR", group = list(
stop("invalid call in method dispatch to 'getData' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

getData-methods ~~ *Methods for Function* getData ~~

Description

~~ Methods for function getData ~~

Methods

signature(x = "GPR")

getHD

Usage

```
getHD(A, string, number = TRUE, position = FALSE)
```

Arguments

A

string

number

position

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, string, number = TRUE, position = FALSE)
{
    if (position) {
        which((trim(A[, 1]) == string) == TRUE)[1]
    }
    else {
        if (number) {
            as.numeric(A[trim(A[, 1]) == string, 2])
        }
        else {
            A[trim(A[, 1]) == string, 2]
        }
    }
  }
```

gethd

Usage

```
gethd(x, hd = NULL)
```

Arguments

x

hd

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, hd = NULL)
standardGeneric("gethd"), generic = structure("gethd", package = "RGPR"), package = "RGPR", group = list(), v
"hd"), default = `\001NULL\001`, skeleton = (function (x, hd = NULL)
stop("invalid call in method dispatch to 'gethd' (no default method)",
    domain = NA))(x, hd), class = structure("standardGeneric", package = "methods"))
```

gethd-methods ~~ *Methods for Function* gethd ~~

Description

~~ Methods for function gethd ~~

Methods

signature(x = "GPR")

getLine

Usage

```
getLine(x, no)
```

Arguments

x

no

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, no)
standardGeneric("getLine"), generic = structure("getLine", package = "RGPR"), package = "RGPR", group = list(
"no"), default = `\001NULL\001`, skeleton = (function (x, no)
stop("invalid call in method dispatch to 'getLine' (no default method)",
    domain = NA))(x, no), class = structure("standardGeneric", package = "methods"))
```

getLine-methods ~~ *Methods for Function* getLine ~~

Description

~~ Methods for function getLine ~~

Methods

signature(x = "GPRsurvey")

get_args

Usage

```
get_args(return_character = TRUE)
```

Arguments

`return_character`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (return_character = TRUE)
{
    arg <- as.list(match.call(def = sys.function(-1), call = sys.call(-1),
        expand.dots = TRUE))
    narg <- length(arg)
    if (return_character) {
        if (narg >= 3) {
            eval_arg <- sapply(arg[3:narg], eval)
            paste(arg[[1]], ":", paste(names(arg[3:narg]), sapply(eval_arg,
                pasteArgs, arg[3:narg]), sep = "=", collapse = "+"),
                sep = "")
        }
        else {
            paste(arg[[1]], ":", sep = "")
        }
    }
    else {
        return(arg)
    }
  }
```

gkernel

Usage

```
gkernel(n, m, sigma = 1)
```

Arguments

`n`

`m`

`sigma`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, m, sigma = 1)
{
    siz = (n - 1)/2
    y = matrix(-siz:siz, n, m)
    siz = (m - 1)/2
    x = matrix(-siz:siz, n, m, byrow = T)
    g = exp(-(x^2 + y^2)/(2 * sigma^2))
    sumg = sum(g)
    if (sumg != 0) {
        g/sumg
    }
    else {
        g
    }
  }
```

GPR

Usage

```
GPR(x, name = "", description = "", filename = "")
```

Arguments

x

name

description

filename

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, name = "", description = "", filename = "")
{
    rec_coord <- cbind(x$dt1$recx, x$dt1$recy, x$dt1$recz)
    trans_coord <- cbind(x$dt1$transx, x$dt1$transy, x$dt1$transz)
    if (sum(is.na(rec_coord)) > 0) {
        warning(paste(sum(is.na(rec_coord)), "NA's in the receiver coordinates\n"))
    }
    if (sum(is.na(trans_coord)) > 0) {
        warning(paste(sum(is.na(trans_coord)), "NA's in the transmitter coordinates\n"))
    }
```

```
if (sum(is.na(x$dt1$topo)) > 0) {
    warning(paste(sum(is.na(x$dt1$topo)), "NA's in the topo coordinates\n"))
}
if (sum(abs(rec_coord), na.rm = TRUE) == 0) {
    rec_coord <- matrix(nrow = 0, ncol = 0)
}
if (sum(abs(trans_coord), na.rm = TRUE) == 0) {
    trans_coord <- matrix(nrow = 0, ncol = 0)
}
if (sum(abs(x$dt1$topo), na.rm = TRUE) == 0) {
    coord <- matrix(nrow = 0, ncol = 0)
}
else {
    coord <- matrix(0, nrow = ncol(x$data), ncol = 3)
    coord[, 3] <- x$dt1$topo
}
dz <- getHD(x$hd, "TOTAL TIME WINDOW")/getHD(x$hd, "NUMBER OF PTS/TRC")
if (sum(abs(x$dt1$time0)) == 0) {
    time0 <- rep(getHD(x$hd, "TIMEZERO AT POINT") * dz, ncol(x$data))
}
else {
    time0 <- x$dt1$time0
}
if (!grepl("^([0-9]{4})(-)([0-9]{2})(-)([0-9]{2})", x$hd[3,
    2])) {
    d <- "1970-01-01"
}
else {
    d <- x$hd[3, 2]
}
myT <- as.double(as.POSIXct(x$dt1$time, origin = as.Date(d)))
GPR_device <- x$hd[2, 2]
if (!grepl("^(Data.)", GPR_device)) {
    GPR_device <- ""
}
hd_list <- list(startpos = getHD(x$hd, "STARTING POSITION"),
    endpos = getHD(x$hd, "FINAL POSITION"), nstacks = getHD(x$hd,
        "NUMBER OF STACKS"), nstacks = getHD(x$hd, "NUMBER OF STACKS"),
    gprdevice = GPR_device)
if (nrow(x$hd) > 17) {
    key <- trim(x$hd[, 1])
    test <- key != "" & seq_along(key) > 17
    key <- key[test]
    key2 <- gsub("[[:punct:]]", replacement = "", key)
    key2 <- gsub(" ", replacement = "_", key2)
    nameL <- trim(x$hd[test, 2])
    names(nameL) <- as.character(key2)
    hd_list_supp <- as.list(nameL)
    hd_list <- c(hd_list, hd_list_supp)
}
new("GPR", data = byte2volt() * x$data, traces = x$dt1$traces,
    com = x$dt1$com, coord = coord, pos = x$dt1$pos, depth = seq(0,
        by = dz, length.out = nrow(x$data)), rec = rec_coord,
    trans = trans_coord, time0 = time0, time = myT, proc = character(0),
    vel = list(0.1), name = name, description = description,
    filename = filename, ntr = ncol(x$data), w = getHD(x$hd,
        "TOTAL TIME WINDOW"), dz = dz, dx = getHD(x$hd, "STEP SIZE USED"),
        depthunit = "ns", posunit = getHD(x$hd, "POSITION UNITS",
            number = FALSE), freq = getHD(x$hd, "NOMINAL FREQUENCY"),
        antsep = getHD(x$hd, "ANTENNA SEPARATION"), surveymode = getHD(x$hd,
            "SURVEY MODE", number = FALSE), date = d, crs = "",
        hd = hd_list)
}
```

GPR-class *Class* "GPR"

Objects from the Class

Objects can be created by calls of the form new("GPR", ...).

Slots

data: Object of class "matrix" ~~

traces: Object of class "numeric" ~~

depth: Object of class "numeric" ~~

pos: Object of class "numeric" ~~

time0: Object of class "numeric" ~~

time: Object of class "numeric" ~~

com: Object of class "character" ~~

ann: Object of class "character" ~~

coord: Object of class "matrix" ~~

rec: Object of class "matrix" ~~

trans: Object of class "matrix" ~~

coordref: Object of class "numeric" ~~

ntr: Object of class "numeric" ~~

w: Object of class "numeric" ~~

freq: Object of class "numeric" ~~

dz: Object of class "numeric" ~~

dx: Object of class "numeric" ~~

antsep: Object of class "numeric" ~~

name: Object of class "character" ~~

description: Object of class "character" ~~

filename: Object of class "character" ~~

depthunit: Object of class "character" ~~

posunit: Object of class "character" ~~

surveymode: Object of class "character" ~~

date: Object of class "character" ~~

crs: Object of class "character" ~~

proc: Object of class "character" ~~

vel: Object of class "list" ~~

delineations: Object of class "list" ~~

hd: Object of class "list" ~~

Methods

[`signature(x = "GPR", i = "ANY", j = "ANY", drop = "ANY")`: ...

[<- `signature(x = "GPR", i = "ANY", j = "ANY", value = "ANY")`: ...

ann `signature(x = "GPR")`: ...

ann<- `signature(x = "GPR")`: ...

apply `signature(X = "GPR")`: ...

Arith `signature(e1 = "ANY", e2 = "GPR")`: ...

Arith `signature(e1 = "GPR", e2 = "ANY")`: ...

Arith `signature(e1 = "GPR", e2 = "GPR")`: ...

as.matrix `signature(x = "GPR")`: ...

clip `signature(x = "GPR")`: ...

coerce `signature(from = "GPR", to = "matrix")`: ...

coord `signature(x = "GPR")`: ...

coord<- `signature(x = "GPR")`: ...

crs `signature(x = "GPR")`: ...

crs<- `signature(x = "GPR")`: ...

dcshift `signature(x = "GPR")`: ...

delineate `signature(x = "GPR")`: ...

delineations `signature(x = "GPR")`: ...

description `signature(x = "GPR")`: ...

dewow `signature(x = "GPR")`: ...

dewow2 `signature(x = "GPR")`: ...

dim `signature(x = "GPR")`: ...

exportDelineations `signature(gpr = "GPR")`: ...

exportFID `signature(x = "GPR")`: ...

exportPDF `signature(x = "GPR")`: ...

fid `signature(x = "GPR")`: ...

fid<- `signature(x = "GPR")`: ...

filename `signature(x = "GPR")`: ...

firstBreack `signature(x = "GPR")`: ...

fkFilter `signature(x = "GPR")`: ...

freqFilter `signature(x = "GPR")`: ...

gain `signature(x = "GPR")`: ...

gammaCorrection `signature(x = "GPR")`: ...

getAmpl `signature(x = "GPR")`: ...

getData `signature(x = "GPR")`: ...

gethd `signature(x = "GPR")`: ...

identifyDelineation `signature(x = "GPR")`: ...

interpTraces `signature(x = "GPR")`: ...

length `signature(x = "GPR")`: ...

Math `signature(x = "GPR")`: ...

max `signature(x = "GPR")`: ...

mean `signature(x = "GPR")`: ...

medianFilter `signature(x = "GPR")`: ...

migration `signature(x = "GPR")`: ...

min `signature(x = "GPR")`: ...

name `signature(x = "GPR")`: ...

ncol `signature(x = "GPR")`: ...

nrow `signature(x = "GPR")`: ...

plot3D `signature(x = "GPR")`: ...

plotAmpl `signature(x = "GPR")`: ...

plotDelineations `signature(x = "GPR")`: ...

plotDelineations3D `signature(x = "GPR")`: ...

range `signature(x = "GPR")`: ...

reverse `signature(x = "GPR")`: ...

rmDelineations<- `signature(x = "GPR")`: ...

setData<- `signature(x = "GPR")`: ...

show `signature(object = "GPR")`: ...

showDelineations `signature(x = "GPR")`: ...

spec `signature(x = "GPR")`: ...

summary `signature(object = "GPR")`: ...

time0 `signature(x = "GPR")`: ...

time0<- `signature(x = "GPR")`: ...

upsample `signature(x = "GPR")`: ...

writeGPR `signature(x = "GPR")`: ...

Examples

```
showClass("GPR")
```

GPRsurvey

Usage

```
GPRsurvey(LINES)
```

Arguments

`LINES`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (LINES)
{
    n <- length(LINES)
    line_names <- character(n)
    line_descriptions <- character(n)
    line_surveymodes <- character(n)
    line_dates <- character(n)
    line_freq <- numeric(n)
    line_antsep <- numeric(n)
    line_lengths <- numeric(n)
    posunit <- character(1)
    crs <- character(1)
    coords <- list()
    fids <- list()
    for (i in seq_along(LINES)) {
        gpr <- readGPR(LINES[[i]])
        line_names[i] <- name(gpr)
        line_descriptions[i] <- description(gpr)
        line_surveymodes[i] <- gpr@surveymode
        line_dates[i] <- gpr@date
        line_freq[i] <- gpr@freq
        line_antsep[i] <- gpr@antsep
        posunit <- gpr@posunit
        crs <- gpr@crs
        if (length(gpr@coord) > 0) {
            if (is.null(colnames(gpr@coord))) {
                coords[[line_names[i]]] <- gpr@coord
            }
            else if (all(toupper(colnames(gpr@coord)) %in% c("E",
                "N", "Z"))) {
                coords[[line_names[i]]] <- gpr@coord[, c("E",
                  "N", "Z")]
            }
            else if (all(toupper(colnames(gpr@coord)) %in% c("X",
                "Y", "Z"))) {
                coords[[line_names[i]]] <- gpr@coord[, c("X",
                  "Y", "Z")]
            }
            else {
                coords[[line_names[i]]] <- gpr@coord
            }
            line_lengths[i] <- lineDist(gpr@coord[, 1:2], last = TRUE)
        }
        else {
            line_lengths[i] <- gpr@dx * gpr@ntr
        }
        fids[[line_names[i]]] <- gpr@com
    }
    x <- new("GPRsurvey", filepaths = LINES, names = line_names,
        descriptions = line_descriptions, surveymodes = line_surveymodes,
        dates = line_dates, freqs = line_freq, lengths = line_lengths,
        antseps = line_antsep, posunit = posunit, crs = crs,
        coords = coords, fids = fids, intersections = list())
    x <- setCoordref(x)
    return(x)
  }
```

GPRsurvey-class *Class* "GPRsurvey"

Objects from the Class

Objects can be created by calls of the form new("GPRsurvey", ...).

Slots

filepaths**:** Object of class "character" ~~

names**:** Object of class "character" ~~

descriptions**:** Object of class "character" ~~

freqs**:** Object of class "numeric" ~~

lengths**:** Object of class "numeric" ~~

surveymodes**:** Object of class "character" ~~

dates**:** Object of class "character" ~~

antseps**:** Object of class "numeric" ~~

posunit**:** Object of class "character" ~~

crs**:** Object of class "character" ~~

coordref**:** Object of class "numeric" ~~

coords**:** Object of class "list" ~~

intersections**:** Object of class "list" ~~

fids**:** Object of class "list" ~~

Methods

[signature(x = "GPRsurvey", i = "ANY", j = "ANY", drop = "ANY"): ...

coords<- signature(x = "GPRsurvey"): ...

exportFID signature(x = "GPRsurvey"): ...

getLine signature(x = "GPRsurvey"): ...

interpTraces signature(x = "GPRsurvey"): ...

intersections signature(x = "GPRsurvey"): ...

length signature(x = "GPRsurvey"): ...

plot3D signature(x = "GPRsurvey"): ...

plotDelineations3D signature(x = "GPRsurvey"): ...

setCoordref signature(x = "GPRsurvey"): ...

show signature(object = "GPRsurvey"): ...

surveyIntersections signature(x = "GPRsurvey"): ...

writeGPR signature(x = "GPRsurvey"): ...

Examples

```
showClass("GPRsurvey")
```

hammingWindow

Usage

```
hammingWindow(L)
```

Arguments

L

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (L)
{
    N = L - 1
    n <- 0:N
    return(0.54 - 0.46 * cos(2 * pi * n/N))
  }
```

identifyDelineation

Usage

```
identifyDelineation(x, sel = NULL, ...)
```

Arguments

x

sel

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, sel = NULL, ...)
standardGeneric("identifyDelineation"), generic = structure("identifyDelineation", package = "RGPR"), packag
"sel"), default = `\001NULL\001`, skeleton = (function (x, sel = NULL,
    ...)
stop("invalid call in method dispatch to 'identifyDelineation' (no default method)",
   domain = NA))(x, sel, ...), class = structure("standardGeneric", package = "methods"))
```

identifyDelineation-methods

~~ Methods for Function identifyDelineation *~~*

Description

~~ Methods for function identifyDelineation ~~

Methods

signature(x = "GPR")

inPoly

Usage

```
inPoly(x, y, vertx, verty)
```

Arguments

x

y

vertx

verty

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, vertx, verty)
{
    inPo <- rep(0L, length(x))
    nvert <- length(vertx)
    for (i in 1:nvert) {
        j <- ifelse(i == 1, nvert, i - 1)
        myTest <- ((verty[i] > y) != (verty[j] > y)) & (x < (vertx[j] -
            vertx[i]) * (y - verty[i])/(verty[j] - verty[i]) +
            vertx[i])
        inPo[myTest] <- !inPo[myTest]
    }
    return(inPo)
  }
```

interpTraces

Usage

```
interpTraces(x, topo)
```

Arguments

x

topo

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, topo)
standardGeneric("interpTraces"), generic = structure("interpTraces", package = "RGPR"), package = "RGPR", gro
"topo"), default = `\001NULL\001`, skeleton = (function (x, topo)
stop("invalid call in method dispatch to 'interpTraces' (no default method)",
    domain = NA))(x, topo), class = structure("standardGeneric", package = "methods"))
```

interpTraces-methods ~~ *Methods for Function* interpTraces ~~

Description

~~ Methods for function interpTraces ~~

Methods

signature(x = "GPR")

signature(x = "GPRsurvey")

intersections

Usage

```
intersections(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("intersections"), generic = structure("intersections", package = "RGPR"), package = "RGPR", g
stop("invalid call in method dispatch to 'intersections' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

intersections-methods ~~ *Methods for Function* intersections ~~

Description

~~ Methods for function intersections ~~

Methods

```
signature(x = "GPRsurvey")
```

is_installed

Usage

```
is_installed(mypkg)
```

Arguments

mypkg

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (mypkg)
is.element(mypkg, installed.packages()[, 1])
```

---

length-methods ~~ *Methods for Function* length ~~

Description

~~ Methods for function length ~~

Methods

signature(x = "GPR")

signature(x = "GPRsurvey")

---

lengthList

Usage

```
lengthList(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x)
{
    if (typeof(x) == "list") {
        return(length(x))
    }
    else {
        return(1)
    }
  }
```

lineDist

Usage

```
lineDist(loc, last = FALSE)
```

Arguments

loc

last

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (loc, last = FALSE)
{
    loc <- as.matrix(loc)
    all_dist <- cumsum(c(0, sqrt(apply(diff(loc)^2, 1, sum))))
    if (last) {
        return(all_dist[length(all_dist)])
    }
    else {
        return(as.numeric(all_dist))
    }
  }
```

load_install_package

Usage

```
load_install_package(package_names)
```

Arguments

package_names

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (package_names)
{
    for (package_name in package_names) {
        if (!is_installed(package_name)) {
            install.packages(package_name, repos = "http://lib.stat.cmu.edu/R/CRAN")
        }
        library(package_name, character.only = TRUE, quietly = TRUE,
            verbose = FALSE)
    }
  }
```

localOrientation

Usage

```
localOrientation(P, blksze = c(5, 10), thresh = 0.1, winEdge = c(7, 7), winBlur = c(3, 3), winTenso
```

Arguments

P

blksze

thresh

winEdge

winBlur

winTensor

sdTensor

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (P, blksze = c(5, 10), thresh = 0.1, winEdge = c(7,
    7), winBlur = c(3, 3), winTensor = c(5, 10), sdTensor = 2,
    ...)
{
    n = nrow(P)
    m = ncol(P)
    Pn = (P - mean(P))/sd(as.vector(P))
    P <- Pn
    blurWinX = winBlur[1]
    blurWinY = winBlur[2]
    k = matrix(1, nrow = blurWinX, ncol = blurWinY, byrow = T)/(blurWinX *
        blurWinY)
    P_f = convolution2D(P, k, 0)
    nnx = winEdge[1]
    nny = winEdge[2]
    vx = convolution2D(P_f, dx_gkernel(nnx, nny, 1), 0)
    vy = convolution2D(P_f, dy_gkernel(nnx, nny, 1), 0)
    Gxx = vx^2
    Gyy = vy^2
    Gxy = vx * vy
    Jxx = convolution2D(Gxx, gkernel(winTensor[1], winTensor[2],
        sdTensor), 0)
    Jyy = convolution2D(Gyy, gkernel(winTensor[1], winTensor[2],
        sdTensor), 0)
    Jxy = convolution2D(Gxy, gkernel(winTensor[1], winTensor[2],
        sdTensor), 0)
    o_alpha = Jxx + Jyy
    o_beta = sqrt((Jxx - Jyy)^2 + 4 * (Jxy)^2)/o_alpha
    o_theta = 1/2 * atan2(2 * Jxy, (Jxx - Jyy)) + pi/2
    o_lambda1 = (Jxx + Jyy + sqrt((Jxx - Jyy)^2 + 4 * (Jxy)^2))/2
    o_lambda2 = (Jxx + Jyy - sqrt((Jxx - Jyy)^2 + 4 * (Jxy)^2))/2
    return(list(energy = o_alpha, anisotropy = o_beta, orientation = o_theta,
        lambda1 = o_lambda1, lambda2 = o_lambda2))
}
```

Math-methods ~~ *Methods for Function* Math ~~

Description

~~ Methods for function Math ~~

Methods

signature(x = "GPR")

max-methods ~~ *Methods for Function* max ~~

Description

~~ Methods for function max ~~

Methods

signature(x = "GPR")

mean-methods ~~ *Methods for Function* mean ~~

Description

~~ Methods for function mean ~~

Methods

signature(x = "ANY")

signature(x = "GPR")

medianFilter

Usage

```
medianFilter(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("medianFilter"), generic = structure("medianFilter", package = "RGPR"), package = "RGPR", gro
stop("invalid call in method dispatch to 'medianFilter' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

medianFilter-methods ~~ *Methods for Function* medianFilter ~~

Description

~~ Methods for function medianFilter ~~

Methods

signature(x = "GPR")

migration

Usage

```
migration(x, type = c("static", "kirchhoff"), ...)
```

Arguments

x

type

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, type = c("static", "kirchhoff"), ...)
standardGeneric("migration"), generic = structure("migration", package = "RGPR"), package = "RGPR", group = l
"type"), default = `\001NULL\001`, skeleton = (function (x, type = c("static",
    "kirchhoff"), ...)
stop("invalid call in method dispatch to 'migration' (no default method)",
   domain = NA))(x, type, ...), class = structure("standardGeneric", package = "methods"))
```

migration-methods ~~ *Methods for Function* `migration` ~~

Description

~~ Methods for function `migration` ~~

Methods

`signature(x = "GPR")`

min-methods ~~ *Methods for Function* `min` ~~

Description

~~ Methods for function `min` ~~

Methods

`signature(x = "GPR")`

minCommon10

Usage

```
minCommon10(xmin, xmax)
```

Arguments

`xmin`

`xmax`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (xmin, xmax)
{
    xmin <- as.numeric(xmin)
    xmax <- as.numeric(xmax)
    D <- xmax - xmin
    n <- nchar(D)
    if (as.numeric(substr(xmin, nchar(xmin) - n + 1, nchar(xmin))) +
        D < 10^(n)) {
        return(as.numeric(substr(xmin, 1, n + 1)) * 10^(nchar(xmin) -
            n - 1))
    }
    else {
        return(xmin)
    }
  }
```

---

myWhich

Usage

```
myWhich(x, y)
```

Arguments

x

y

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y)
{
    which(x == y)
  }
```

myWhichMin

Usage

```
myWhichMin(x, y)
```

Arguments

x

y

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y)
{
    which.min(abs(x - y))
  }
```

name

Usage

```
name(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("name"), generic = structure("name", package = "RGPR"), package = "RGPR", group = list(), val
stop("invalid call in method dispatch to 'name' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

name-methods *~~ Methods for Function* name *~~*

Description

~~ Methods for function name ~~

Methods

signature(x = "GPR")

ncol-methods *~~ Methods for Function* ncol *~~*

Description

~~ Methods for function ncol ~~

Methods

signature(x = "ANY")

signature(x = "GPR")

nextpower2

Usage

```
nextpower2(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x)
{
    return(2^(ceiling(log2(x))))
  }
```

normalize

Usage

```
normalize(A, type = c("stat", "min-max", "95", "eq", "sum"))
```

Arguments

A

type

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, type = c("stat", "min-max", "95", "eq", "sum"))
{
    A = as.matrix(A)
    type = match.arg(type)
    if (type == "stat") {
        Anorm <- scale(A, center = .colMeans(A, nrow(A), ncol(A)),
            scale = apply(A, 2, sd, na.rm = TRUE))
    }
    else if (type == "sum") {
        Anorm <- scale(A, center = FALSE, scale = colSums(abs(A)))
    }
    else if (type == "eq") {
        amp <- apply((A)^2, 2, sum)
        Anorm <- A * sqrt(amp)/sum(sqrt(amp))
    }
    else if (type == "95") {
        A_q95 = (apply((A), 2, quantile, 0.99, na.rm = TRUE))
        A_q05 = (apply((A), 2, quantile, 0.01, na.rm = TRUE))
        Anorm = (A)/(A_q95 - A_q05)
    }
    else {
        Anorm <- scale(A, center = FALSE, scale = (apply((A),
            2, max, na.rm = TRUE)) - (apply((A), 2, min, na.rm = TRUE)))
    }
    return(Anorm)
  }
```

nrow-methods ~~ *Methods for Function* nrow ~~

Description

~~ Methods for function nrow ~~

Methods

signature(x = "ANY")

signature(x = "GPR")

paddMatrix

Usage

```
paddMatrix(I, p1, p2 = NULL)
```

Arguments

I

p1

p2

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (I, p1, p2 = NULL)
{
    if (is.null(p2)) {
        p2 <- p1
    }
    nI <- nrow(I)
    mI <- ncol(I)
    Ipad <- matrix(0, nrow = nI + 2 * p1, ncol = mI + 2 * p2)
    Ipad[(p1 + 1):(p1 + nI), (p2 + 1):(p2 + mI)] <- I
    Ipad[1:p1, (p2 + 1):(p2 + mI)] <- repmat(I[1, ], p1, 1)
    Ipad[(p1 + nI + 1):(nI + 2 * p1), (p2 + 1):(p2 + mI)] <- repmat(I[nI,
        ], p1, 1)
    Ipad[(p1 + 1):(p1 + nI), 1:p2] <- repmat(I[, 1], 1, p2)
    Ipad[(p1 + 1):(p1 + nI), (p2 + mI + 1):(mI + 2 * p2)] <- repmat(I[,
        mI], 1, p2)
    Ipad[1:p1, 1:p2] <- I[1, 1]
    Ipad[1:p1, (p2 + mI + 1):(mI + 2 * p2)] <- I[1, mI]
    Ipad[(p1 + nI + 1):(nI + 2 * p1), 1:p2] <- I[nI, 1]
    Ipad[(p1 + nI + 1):(nI + 2 * p1), (p2 + mI + 1):(mI + 2 *
        p2)] <- I[nI, mI]
    return(Ipad)
  }
```

pasteArgs

Usage

```
pasteArgs(eval_arg, arg)
```

Arguments

eval_arg

arg

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (eval_arg, arg)
{
    if (is.numeric(eval_arg) || is.character(eval_arg)) {
        return(paste(eval_arg, collapse = ",", sep = ""))
    }
    else if (is.list(eval_arg)) {
        return(paste(names(eval_arg), "<-", (eval_arg), collapse = ",",
            sep = ""))
    }
    else if (is.matrix(eval_arg)) {
        return(paste(arg))
    }
    else if (any(is.null(eval_arg))) {
        return("")
    }
  }
```

phaseRotation

Usage

```
phaseRotation(x, phi)
```

Arguments

x

phi

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, phi)
{
    nf <- length(x)
    X = fft(x)
    phi2 <- numeric(nf)
    phi2[2:(nf/2)] <- phi
    phi2[(nf/2 + 1):(nf)] <- -phi
    Phase = exp(-complex(imaginary = -1) * phi2)
    xcor = fft(X * Phase, inverse = TRUE)/nf
    return(Re(xcor))
  }
```

plot.GPR

Usage

```
plot.GPR(x, y, ...)
```

Arguments

x

y

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, ...)
{
    type <- "raster"
    add_topo <- FALSE
    clip = NULL
    normalize = NULL
    upsample = NULL
    dots <- list()
    if (length(list(...))) {
        dots <- list(...)
        if (!is.null(dots$type)) {
            type <- dots$type
            dots$type <- NULL
        }
        if (!is.null(dots$clip)) {
            clip <- dots$clip
            dots$clip <- NULL
        }
        if (!is.null(dots$normalize)) {
            normalize <- dots$normalize
            dots$normalize <- NULL
        }
        if (!is.null(dots$upsample)) {
            upsample <- dots$upsample
            dots$upsample <- NULL
        }
        add_ann <- TRUE
        if (!is.null(dots$add_ann) && !isTRUE(dots$add_ann)) {
            add_ann <- FALSE
        }
        add_fid <- TRUE
        if (!is.null(dots$add_fid) && !isTRUE(dots$add_fid)) {
            add_fid <- FALSE
        }
        add_topo <- FALSE
        if (!is.null(dots$add_topo) && isTRUE(dots$add_topo)) {
            add_topo <- TRUE
        }
        dots$add_fid <- NULL
        dots$add_topo <- NULL
        dots$addArrows <- NULL
        if (!is.null(dots$lwd)) {
            lwd <- dots$lwd
        }
        dots$add <- NULL
        if (!is.null(dots$shp_files)) {
            add_shp_files <- TRUE
            shp_files <- dots$shp_files
        }
        dots$shp_files <- NULL
    }
    if (length(x@vel) > 0) {
        vel <- x@vel[[1]]
    }
    else {
        vel <- 0
    }
    if (any(dim(x) == 1)) {
        par(mar = c(5, 4, 3, 2) + 0.1, oma = c(0, 0, 3, 0), mgp = c(2,
            0.5, 0))
        z <- seq(0, by = x@dz, length.out = length(x@data))
        plot(z, x@data, type = "n", xlab = x@depthunit, ylab = "mV",
            xaxt = "n")
        x_axis <- pretty(seq(x@time0, by = x@dz, length.out = length(x@data)))
        axis(side = 1, at = x_axis + x@time0, labels = x_axis,
            tck = +0.02)
        depth0 <- depth0(x@time0, vel, antsep = x@antsep)
        depth <- (seq(0, by = 2.5, max(z) * vel))
        depth2 <- seq(0.1, by = 0.1, 0.9)
        depthat <- depthToTime(depth, x@time0, vel, antsep = x@antsep)
        depthat2 <- depthToTime(depth2, x@time0, vel, antsep = x@antsep)
        axis(side = 3, at = depthat, labels = depth, tck = +0.02)
        axis(side = 3, at = depthat2, labels = FALSE, tck = +0.01)
        axis(side = 3, at = depthToTime(1, x@time0, vel, antsep = x@antsep),
            labels = FALSE, tck = +0.02)
        abline(h = 0, lty = 3, col = "grey")
        abline(v = x@time0, col = "red")
        abline(v = depth0, col = "grey", lty = 3)
        lines(z, x@data)
        title(paste(x@name, ": trace n0", x@traces, " @", x@pos,
            x@posunit, sep = ""), outer = TRUE)
        mtext(paste("depth (m),   v=", vel, "m/ns", sep = ""),
            side = 3, line = 2)
    }
    else {
        if (!is.null(upsample)) {
            x <- upsample(x, n = upsample)
        }
        if (!is.null(normalize)) {
            x@data <- normalize(x@data, type = normalize)
        }
        if (!is.null(clip) && is.numeric(clip)) {
            if (length(clip) > 1) {
                x@data <- .clip(x@data, clip[2], clip[1])
            }
            else if (length(clip) == 1) {
                x@data <- .clip(x@data, clip[1])
            }
        }
        if (add_fid == FALSE) {
            x@com <- character(length(x@com))
        }
        type = match.arg(type, c("raster", "wiggles"))
        if (type == "raster") {
            if (add_topo) {
                x <- migration(x)
            }
            if (grepl("[m]$", x@depthunit)) {
                ylab <- paste("depth (", x@depthunit, ")", sep = "")
            }
            else if (grepl("[s]$", x@depthunit)) {
                ylab <- paste("two-way travel time (", x@depthunit,
                  ")", sep = "")
            }
            if (length(x@coord) > 0 && sum(abs(x@coord[, 1:2]) >
                0)) {
                xvalues <- lineDist(x@coord)
            }
            else {
                xvalues <- x@pos
            }
            cat(xvalues)
            cat("\n")
            cat(-rev(x@depth))
            cat("\n")
            do.call(plotRaster, c(list(A = x@data, col = diverge_hcl(101,
                h = c(246, 10), c = 120, l = c(30, 90)), x = xvalues,
                y = -rev(x@depth), main = x@name, xlab = x@posunit,
                ylab = ylab, note = x@filename, time0 = x@time0,
                antsep = x@antsep, v = vel, fid = x@com, ann = x@ann,
                depthunit = x@depthunit), dots))
        }
        else if (type == "wiggles") {
            if (add_topo && length(x@coord) > 0) {
                topo <- x@coord[, 3]
            }
            else {
                topo = NULL
            }
            if (grepl("[m]$", x@depthunit)) {
                ylab <- paste("depth (", x@depthunit, ")", sep = "")
            }
            else if (grepl("[s]$", x@depthunit)) {
                if (add_topo) {
                  ylab <- paste("depth (m)", sep = "")
                }
                else {
                  ylab <- paste("two-way travel time (", x@depthunit,
                    ")", sep = "")
                }
            }
            if (length(x@coord) > 0) {
                xvalues <- lineDist(x@coord)
            }
            else {
                xvalues <- x@pos
            }
            do.call(plotWig, c(list(A = x@data, x = xvalues,
                y = -rev(x@depth), main = x@name, xlab = x@posunit,
                ylab = ylab, topo = topo, note = x@filename,
                col = "black", time0 = x@time0, antsep = x@antsep,
                v = vel, fid = x@com, ann = x@ann, depthunit = x@depthunit),
                dots))
        }
    }
  }
```

plot.GPRsurvey

Usage

```
plot.GPRsurvey(x, y, ...)
```

Arguments

x

y

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, ...)
{
    if (length(x@coords) > 0) {
        plotAdd <- FALSE
        addArrows <- TRUE
        add_shp_files <- FALSE
        addIntersections <- TRUE
        addFid <- TRUE
        dots <- list()
        lwd = 1
        if (length(list(...))) {
            dots <- list(...)
            if (!is.null(dots$add) && isTRUE(dots$add)) {
                plotAdd <- TRUE
            }
            if (!is.null(dots$addArrows) && isTRUE(!dots$addArrows)) {
                addArrows <- FALSE
            }
            dots$addArrows <- NULL
            if (!is.null(dots$lwd)) {
                lwd <- dots$lwd
            }
            if (!is.null(dots$addIntersections)) {
                addIntersections <- dots$addIntersections
            }
            dots$addIntersections <- NULL
            if (!is.null(dots$addFid)) {
                addFid <- dots$addFid
            }
            dots$addFid <- NULL
            dots$add <- NULL
            if (!is.null(dots$shp_files)) {
                add_shp_files <- TRUE
                shp_files <- dots$shp_files
            }
            dots$shp_files <- NULL
        }
        dots <- c(dots, list(type = "n"))
        if (!plotAdd) {
            do.call("plot", c(list((do.call(rbind, x@coords))[,
                1:2]), dots))
        }
        if (add_shp_files) {
            if (length(shp_files) > 0) {
                BASEName <- unlist(strsplit(basename(shp_files),
                  "[.]"))[seq(from = 1, length.out = length(shp_files),
                  by = 2)]
                DIRName <- dirname(shp_files)
                for (i in seq_along(shp_files)) {
                  shp <- readOGR(DIRName[i], BASEName[i])
```

```
                    cat(DIRName[i], BASEName[i], "\n", sep = "")
                    plot(shp, add = TRUE, pch = 13, col = "darkblue")
                  }
              }
          }
          niet <- lapply(x@coords, plotLine, lwd = lwd)
          if (addArrows) {
              niet <- lapply(x@coords, plotArrows, lwd = lwd)
          }
          if (addFid) {
              for (i in 1:length(x)) {
                  fidxyz <- fidpos(x@coords[[i]], x@fids[[i]])
                  if (length(fidxyz) > 0) {
                    points(fidxyz[, 1:2], pch = 21, col = "black",
                      bg = "red", cex = 0.7)
                  }
              }
          }
          if (length(x@intersections) > 0 && addIntersections) {
              for (i in 1:length(x@intersections)) {
                  if (!is.null(x@intersections[[i]])) {
                    points(x@intersections[[i]][, 1:2], pch = 1,
                      cex = 0.8)
                  }
              }
          }
      }
      else {
          warning("no coordinates")
      }
  }
```

---

plot3D

Usage

```
plot3D(x, add_topo = FALSE, clip = NULL, normalize = NULL, upsample = NULL, add = TRUE, xlim = NULL
```

Arguments

x

add_topo

clip

normalize

upsample

add

xlim

ylim

zlim

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, add_topo = FALSE, clip = NULL, normalize = NULL,
    upsample = NULL, add = TRUE, xlim = NULL, ylim = NULL, zlim = NULL,
    ...)
standardGeneric("plot3D"), generic = structure("plot3D", package = "RGPR"), package = "RGPR", group = list(),
"add_topo", "clip", "normalize", "upsample", "add", "xlim", "ylim",
"zlim"), default = `\001NULL\001`, skeleton = (function (x, add_topo = FALSE,
    clip = NULL, normalize = NULL, upsample = NULL, add = TRUE,
    xlim = NULL, ylim = NULL, zlim = NULL, ...)
stop("invalid call in method dispatch to 'plot3D' (no default method)",
    domain = NA))(x, add_topo, clip, normalize, upsample, add,
    xlim, ylim, zlim, ...), class = structure("standardGeneric", package = "methods"))
```

plot3D-methods *~~ Methods for Function* plot3D *~~*

Description

~~ Methods for function plot3D ~~

Methods

signature(x = "GPR")

signature(x = "GPRsurvey")

plot3DSlice

Usage

```
plot3DSlice(XYZ, slice = c("x", "y", "z"), section = 1, col = diverge_hcl(101, h = c(246, 10), c =
```

Arguments

XYZ

slice

section

col

sampling

rmStripes

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (XYZ, slice = c("x", "y", "z"), section = 1, col = diverge_hcl(101,
    h = c(246, 10), c = 120, l = c(30, 90)), sampling = c(0.25,
    0.25, 0.04), rmStripes = TRUE)
{
    slice = match.arg(slice)
    if (length(slice) > 1) {
        slice = slice[1]
    }
    dimXYZ = dim(XYZ)
    vz = seq(0, dimXYZ[3] - 1, by = 1) * sampling[3]
    vx = seq(0, dimXYZ[1] - 1, by = 1) * sampling[1]
    vy = seq(0, dimXYZ[2] - 1, by = 1) * sampling[2]
    if (rgl.cur() == 0) {
        rgl.open()
        rgl.bg(color = c("white"))
    }
    i = section
    j = i
    k = i
    if (slice == "x") {
        if (rmStripes == TRUE) {
            Xside = normalizeGPR(removeStripes(t(XYZ[, j, ])))
        }
        else {
            Xside = normalizeGPR((t(XYZ[, j, ])))
        }
        Xside_x = matrix(vx, nrow = dimXYZ[3], ncol = dimXYZ[1],
            byrow = TRUE)
        Xside_y = matrix(vy[j], nrow = dimXYZ[3], ncol = dimXYZ[1],
            byrow = TRUE)
        Xside_z = matrix(max(vz) - vz, nrow = dimXYZ[3], ncol = dimXYZ[1],
            byrow = FALSE)
        CCX = (Xside - min(Xside))/(max(Xside) - min(Xside))
        ClimX <- range(CCX)
        ClenX <- ClimX[2] - ClimX[1] + 1
        colCX <- col[(CCX) * 100 + 1]
        surface3d(Xside_x, Xside_z, Xside_y, col = setCol(Xside),
            lit = FALSE, front = "fill", back = "fill")
    }
    else if (slice == "z") {
        if (rmStripes == TRUE) {
            Zside = (removeStripes(t(XYZ[, , k])))
        }
        else {
            Zside = ((t(XYZ[, , k])))
        }
        Zside_x = matrix(vx, nrow = dimXYZ[2], ncol = dimXYZ[1],
            byrow = TRUE)
        Zside_y = matrix(vy, nrow = dimXYZ[2], ncol = dimXYZ[1],
            byrow = FALSE)
        Zside_z = matrix(max(vz) - vz[k], nrow = dimXYZ[2], ncol = dimXYZ[1],
            byrow = FALSE)
        CCZ = (Zside - min(Zside))/(max(Zside) - min(Zside))
        ClimZ <- range(CCZ)
        ClenZ <- ClimZ[2] - ClimZ[1] + 1
        colCZ <- col[(CCZ) * 100 + 1]
        surface3d(Zside_x, Zside_z, Zside_y, col = setCol(Zside),
            lit = FALSE, front = "fill", back = "fill")
    }
    else if (slice == "y") {
        if (rmStripes == TRUE) {
            Yside = normalizeGPR(removeStripes(t(XYZ[i, , ])))
        }
        else {
            Yside = normalizeGPR((t(XYZ[i, , ])))
        }
        Yside_x = matrix(vx[i], nrow = dimXYZ[3], ncol = dimXYZ[2],
            byrow = TRUE)
        Yside_y = matrix(vy, nrow = dimXYZ[3], ncol = dimXYZ[2],
            byrow = TRUE)
        Yside_z = matrix(max(vz) - vz, nrow = dimXYZ[3], ncol = dimXYZ[2],
            byrow = FALSE)
        CCY = (Yside - min(Yside))/(max(Yside) - min(Yside))
        ClimY <- range(CCY)
        ClenY <- ClimY[2] - ClimY[1] + 1
        colCY <- col[(CCY) * 100 + 1]
        surface3d(Yside_x, Yside_z, Yside_y, col = setCol(Yside),
            lit = FALSE, front = "fill", back = "fill")
    }
  }
```

plotAmpl

Usage

```
plotAmpl(x, FUN = mean, add = FALSE, ylim = NULL, xlim = NULL, col = 1, all = FALSE, ...)
```

Arguments

x

FUN

add

ylim

xlim

col

all

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, FUN = mean, add = FALSE, ylim = NULL,
    xlim = NULL, col = 1, all = FALSE, ...)
standardGeneric("plotAmpl"), generic = structure("plotAmpl", package = "RGPR"), package = "RGPR", group = lis
"FUN", "add", "ylim", "xlim", "col", "all"), default = `\001NULL\001`, skeleton = (function (x,
    FUN = mean, add = FALSE, ylim = NULL, xlim = NULL, col = 1,
    all = FALSE, ...)
stop("invalid call in method dispatch to 'plotAmpl' (no default method)",
   domain = NA))(x, FUN, add, ylim, xlim, col, all, ...), class = structure("standardGeneric", package = "met
```

plotAmpl-methods ~~ *Methods for Function* plotAmpl ~~

Description

~~ Methods for function plotAmpl ~~

Methods

signature(x = "GPR")

plotArrows

Usage

```
plotArrows(xyz, ...)
```

Arguments

xyz

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (xyz, ...)
{
    arrows(xyz[nrow(xyz) - 1, 1], xyz[nrow(xyz) - 1, 2], xyz[nrow(xyz),
        1], xyz[nrow(xyz), 2], length = 0.1, col = "red", ...)
  }
```

plotDelineations

Usage

```
plotDelineations(x, sel = NULL, col = NULL, ...)
```

Arguments

x

sel

col

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, sel = NULL, col = NULL, ...)
standardGeneric("plotDelineations"), generic = structure("plotDelineations", package = "RGPR"), package = "R
"sel", "col"), default = `\001NULL\001`, skeleton = (function (x,
    sel = NULL, col = NULL, ...)
stop("invalid call in method dispatch to 'plotDelineations' (no default method)",
   domain = NA))(x, sel, col, ...), class = structure("standardGeneric", package = "methods"))
```

plotDelineations-methods

~~ Methods for Function plotDelineations ~~

Description

~~ Methods for function plotDelineations ~~

Methods

signature(x = "GPR")

plotDelineations3D

Usage

```
plotDelineations3D(x, sel = NULL, col = NULL, add = TRUE, ...)
```

Arguments

x

sel

col

add

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, sel = NULL, col = NULL, add = TRUE, ...)
standardGeneric("plotDelineations3D"), generic = structure("plotDelineations3D", package = "RGPR"), package
"sel", "col", "add"), default = `\001NULL\001`, skeleton = (function (x,
    sel = NULL, col = NULL, add = TRUE, ...)
stop("invalid call in method dispatch to 'plotDelineations3D' (no default method)",
   domain = NA))(x, sel, col, add, ...), class = structure("standardGeneric", package = "methods"))
```

plotDelineations3D-methods

~~ Methods for Function `plotDelineations3D` *~~*

Description

~~ Methods for function `plotDelineations3D` ~~

Methods

`signature(x = "GPR")`

`signature(x = "GPRsurvey")`

plotLine

Usage

```
plotLine(xyz, col = 1, ...)
```

Arguments

xyz

col

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (xyz, col = 1, ...)
{
    lines(xyz[, 1:2], ...)
  }
```

plotRaster

Usage

```
plotRaster(A, x = NULL, y = NULL, plot_raster = TRUE, barscale = TRUE, add = FALSE, mai = c(1, 0.8,
```

Arguments

A

x

y

plot_raster

barscale

add

mai

col

note

main

time0

antsep

v

ann

add_ann

fid

depthunit

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, x = NULL, y = NULL, plot_raster = TRUE, barscale = TRUE,
    add = FALSE, mai = c(1, 0.8, 0.8, 1.8), col = heat.colors(101),
    note = NULL, main = "", time0 = 0, antsep = 1, v = 0.1, ann = NULL,
    add_ann = TRUE, fid = NULL, depthunit = "ns", ...)
{
    GPR = as.matrix(A)
    GPR[is.na(GPR)] = 0
    time0 <- mean(time0)
    zlim = range(GPR)
    if (length(list(...))) {
        Lst <- list(...)
        if (!is.null(Lst$zlim)) {
            zlim <- Lst$zlim
        }
    }
    if (grepl("[m]$", depthunit)) {
        mai <- c(1, 0.8, 0.8, 0.5)
    }
    reverse <- nrow(GPR):1
    GPR <- t(GPR[reverse, ])
    if (is.null(x)) {
        x <- (1:nrow(GPR))
    }
    if (is.null(y)) {
        y <- -(ncol(GPR):1)
    }
    if (add == TRUE) {
        par(new = TRUE)
    }
    else {
        par(mai = mai, oma = c(0, 0, 3, 0))
    }
    y <- y + time0
    image(x, y, GPR, col = col, zlim = zlim, xaxs = "i", yaxs = "i",
        yaxt = "n", ...)
    title(main, outer = TRUE, line = 1)
    usr <- par()$usr
    pin <- par()$pin
    dxin <- diff(usr[1:2])/(pin[1])
    dylim <- diff(usr[3:4])
    dusr <- dylim/length(y)
```

```
pretty_y <- pretty(y)
if (!is.null(fid) && length(fid) > 0 && any(fid != "")) {
    cin <- par()$cin[2]
    posfid <- x
    testfid <- (fid != "")
    ylim = range(y)
    yr <- diff(usr[3:4])/(pin[2])
    if (sum(testfid) > 0) {
        par(xpd = TRUE)
        cst <- yr * cin
        points(posfid[testfid], cst/2 * 0.75 + rep(ylim[2],
            sum(testfid)), pch = 25, col = "red", bg = "yellow",
            cex = 1)
        text(posfid[testfid], cst + rep(ylim[2], sum(testfid)),
            fid[testfid], cex = 0.6)
        par(xpd = FALSE)
    }
}
if (add_ann && !is.null(ann) && length(ann) > 0) {
    posann <- x
    testann <- (ann != "")
    ann <- gsub("#", "\n", ann)
    if (sum(testann) > 0) {
        abline(v = posann[testann], col = "red", lwd = 1)
        mtext(ann[testann], side = 3, line = 1.7, at = posann[testann],
            col = "red", cex = 0.9)
    }
}
axis(side = 2, at = pretty_y + dusr/2, labels = -pretty_y)
abline(h = 0, col = "red", lwd = 0.5)
if (grepl("[s]$", depthunit)) {
    depth <- (seq(0, by = 2.5, max(abs(y)) * v))
    depth2 <- seq(0.1, by = 0.1, 0.9)
    depthat <- depthToTime(depth, 0, v, antsep)
    depthat2 <- depthToTime(depth2, 0, v, antsep)
    axis(side = 4, at = -depthat, labels = depth, tck = -0.02)
    axis(side = 4, at = -depthat2, labels = FALSE, tck = -0.01)
    axis(side = 4, at = -1 * depthToTime(1, 0, v, antsep),
        labels = FALSE, tck = -0.02)
    mtext(paste("depth (m),   v=", v, "m/ns", sep = ""),
        side = 4, line = 2)
}
else {
    axis(side = 4, at = pretty_y + dusr/2, labels = -pretty_y)
}
if (!is.null(note)) {
    mtext(note, side = 1, line = 4, cex = 0.6)
}
box()
op <- par(no.readonly = TRUE)
if (barscale && grepl("[s]$", depthunit)) {
    fin <- par()$fin
    mai2 <- c(1, 0.8 + pin[1] + 1, 0.8, 0.6)
    par(mai = mai2)
    fin2 <- par()$fin
    wstrip <- fin2[1] - mai2[2] - mai2[4]
    xpos <- diff(usr[1:2]) * (mai2[2] - mai[2])/pin[1]
```

```
        zstrip <- matrix(seq(zlim[1], zlim[2], length.out = length(col)),
            nrow = 1)
        xstrip <- c(xpos, xpos + wstrip * dxin) * c(0.9, 1.1)
        ystrip <- seq(min(y), max(y), length.out = length(col))
        pretty_z <- pretty(as.vector(zstrip))
        dzlim <- zlim[2] - zlim[1]
        pretty_at <- usr[3] - dylim * (zlim[1] - pretty_z)/dzlim
        axis(side = 4, las = 2, at = pretty_at, labels = pretty_z)
        image(xstrip, ystrip, zstrip, zlim = zlim, add = TRUE,
            col = col, axes = FALSE, xlab = "", ylab = "", xaxs = "i",
            yaxs = "i")
        box()
    }
    par(op)
  }
```

plotTopo

Usage

```
plotTopo(NEZ_file, add = TRUE)
```

Arguments

NEZ_file

add

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (NEZ_file, add = TRUE)
{
    topo <- read.table(NEZ_file, header = TRUE, sep = ",", stringsAsFactors = FALSE)
    PCODE <- unique(topo$PCODE)
    TS <- agrep("TS", PCODE)
    REF <- agrep("REF", PCODE)
    WATER <- agrep("WATER", PCODE)
    CROSS <- which("CROSS" == PCODE)
    REVERSE <- agrep("REVERSE", PCODE)
    LINES <- agrep("LINE", PCODE)
    LINES <- LINES[!(agrep("LINE", PCODE) %in% REVERSE)]
    POINTS <- which(!(1:length(PCODE) %in% c(LINES, TS, REVERSE,
        WATER, CROSS, REF)))
    NOT_REVERSE <- !(1:length(PCODE) %in% agrep("REVERSE", PCODE))
    not_rev <- !(1:nrow(topo) %in% agrep("REVERSE", topo$PCODE))
    if (add == FALSE) {
        plot(topo[not_rev, c("E", "N")], type = "n", asp = 1)
    }
    for (i in 1:length(REVERSE)) {
        points(-topo[topo[, "PCODE"] == PCODE[REVERSE[i]], c("E",
            "N")], pch = 20, col = 1)
    }
    points(topo[topo[, "PCODE"] %in% PCODE[WATER], c("E", "N")],
        pch = 10, col = 1)
    for (i in 1:length(POINTS)) {
        points(topo[topo[, "PCODE"] == PCODE[POINTS[i]], c("E",
            "N")], pch = 3, col = 1, cex = 0.7)
    }
    points(topo[topo[, "PCODE"] %in% PCODE[REF], c("E", "N")],
        pch = 25, col = 3, bg = "green")
  }
```

plotWig

Usage

```
plotWig(A, x = NULL, y = NULL, xlim = NULL, ylim = NULL, topo = NULL, main = "", note = NULL, fid = N
```

Arguments

A

x

y

xlim

ylim

topo

main

note

fid

ann

add_ann

pdfName

ws

side

dx

dz

ratio

col

time0

antsep

v

depthunit

lwd

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, x = NULL, y = NULL, xlim = NULL, ylim = NULL, topo = NULL,
    main = "", note = NULL, fid = NULL, ann = NULL, add_ann = TRUE,
    pdfName = NULL, ws = 1, side = 1, dx = 0.25, dz = 0.4, ratio = 1,
    col = black, time0 = 0, antsep = 1, v = 0.1, depthunit = "ns",
    lwd = 0.5, ...)
{
    dx <- mean(diff(x))
    A[is.na(A)] = 0
    A = A/max(abs(A)) * dx
    nr = nrow(A)
    nc = ncol(A)
    A <- A[nr:1, ]
    time0 <- mean(time0)
    if (is.null(y)) {
        y <- -(ncol(GPR):1)
    }
    if (is.null(topo)) {
        topo <- rep(0L, nc)
    }
    else {
        if (grepl("[s]$", depthunit)) {
            y <- y * v/2
            depthunit <- "m"
        }
        topo <- topo - max(topo)
    }
    if (grepl("[s]$", depthunit)) {
    }
    else if (grepl("[m]$", depthunit)) {
        depth0 <- depthToTime(z = 0, time0, v = v, antsep = antsep) *
            v/2
        y <- y + depth0
    }
    if (is.null(xlim)) {
        xlim <- range(x) + c(-1, 1) * dx
        test <- rep(TRUE, length(x))
    }
    else {
        test <- (x >= xlim[1] & x <= xlim[2])
        xlim <- xlim + c(-1, 1) * dx
    }
    if (is.null(ylim)) {
        ylim <- range(y) + range(topo)
    }
    omi = c(0, 0, 0.6, 0)
    mgp = c(2.5, 0.75, 0)
    fac <- 0.2
    if (grepl("[m]$", depthunit)) {
        mai = c(1, 0.8, 0.6, 0.4) + 0.02
        heightPDF <- fac * diff(ylim) + sum(omi[c(1, 3)] + mai[c(1,
            3)])
        widthPDF <- fac * diff(xlim) * ratio + sum(omi[c(2, 4)] +
            mai[c(2, 4)])
    }
    else {
        mai = c(1, 0.8, 0.6, 0.8) + 0.02
        heightPDF <- fac * (ylim[2] - ylim[1]) * v/2 + sum(omi[c(1,
            3)] + mai[c(1, 3)])
        widthPDF <- fac * (xlim[2] - xlim[1]) * ratio + sum(omi[c(2,
            4)] + mai[c(2, 4)])
    }
    if (!is.null(pdfName)) {
        CairoPDF(file = paste(pdfName, ".pdf", sep = ""), width = widthPDF,
            height = heightPDF, bg = "white", pointsize = 10,
            title = pdfName)
    }
    par(mai = mai, omi = omi, mgp = mgp)
    plot(0, 0, type = "n", xaxs = "i", yaxs = "i", axes = FALSE,
        xlim = xlim, ylim = ylim, ...)
    title(main, outer = TRUE, line = 1)
    if (!is.null(fid) && length(fid) > 0 && any(fid != "")) {
        pin <- par("pin")
        usr <- par("usr")
        cin <- par()$cin[2]
        posfid <- x[test]
        fid <- fid[test]
        testfid <- (fid != "")
        yr <- diff(usr[3:4])/(pin[2])
        if (sum(testfid) > 0) {
            par(xpd = TRUE)
            cst <- yr * cin
            points(posfid[testfid], cst/2 * 0.75 + rep(ylim[2],
                sum(testfid)), pch = 25, col = "red", bg = "yellow",
                cex = 1)
            text(posfid[testfid], cst + rep(ylim[2], sum(testfid)),
                fid[testfid], cex = 0.6)
            par(xpd = FALSE)
        }
    }
    if (side == 1) {
        for (i in rev(seq_along(x))) {
            y2 <- y + topo[i]
            wig = cbind(ws * A[, i] + x[i], y2)
            wig1 = rbind(c(x[i], y2[1]), wig, c(x[i], y2[nr]))
            polygon(wig1, col = col, border = NA)
            rect(min(wig1[, 1]), ylim[1], x[i], ylim[2], col = "white",
                border = NA)
        }
    }
    else {
        for (i in (seq_along(x))) {
            y2 <- y + topo[i]
            wig = cbind(ws * A[, i] + x[i], y2)
            wig1 = rbind(c(x[i], y2[1]), wig, c(x[i], y2[nr]))
            polygon(wig1, col = col, border = NA)
            rect(max(wig1[, 1]), ylim[1], x[i], ylim[2], col = "white",
                border = NA)
        }
    }
    for (i in (seq_along(x))) {
        y2 <- y + topo[i]
        lines(x[i] + ws * A[, i], y2, lwd = lwd)
    }
    if (add_ann && !is.null(ann) && length(ann) > 0) {
        posann <- x[test]
        ann <- ann[test]
        testann <- (ann != "")
        ann <- gsub("#", "\n", ann)
        if (sum(testann) > 0) {
            abline(v = posann[testann], col = "red", lwd = 0.5)
            mtext(ann[testann], side = 3, line = 1.7, at = posann[testann],
                col = "red", cex = 0.9)
        }
    }
    axis(side = 1, tck = -0.02)
    if (grepl("[s]$", depthunit)) {
        abline(h = -time0, col = "red", lwd = 0.5)
        depth <- (seq(0, by = 2.5, max(abs(y)) * v))
        depth2 <- seq(0.1, by = 0.1, 0.9)
        depthat <- depthToTime(depth, time0, v, antsep)
        depthat2 <- depthToTime(depth2, time0, v, antsep)
        axis(side = 4, at = -depthat, labels = depth, tck = -0.02)
        axis(side = 4, at = -depthat2, labels = FALSE, tck = -0.01)
        axis(side = 4, at = -1 * depthToTime(1, time0, v, antsep),
            labels = FALSE, tck = -0.02)
        axis(side = 2, at = pretty(y) - time0, labels = -pretty(y),
            tck = -0.02)
        mtext(paste("depth (m),   v=", v, "m/ns", sep = ""),
            side = 4, line = 2)
    }
    else {
        abline(h = 0, col = "red", lwd = 0.5)
        axis(side = 2, at = pretty(y), labels = -pretty(y), tck = -0.02)
        axis(side = 4, at = pretty(y), labels = -pretty(y), tck = -0.02)
    }
    box()
    if (!is.null(note)) {
        mtext(note, side = 1, line = 4, cex = 0.6)
    }
    if (!is.null(pdfName)) {
        dev.off()
    }
  }
```

powSpec

Usage

```
powSpec(A, T = 0.8, fac = 1e+06, plot_spec = TRUE, return_spec = FALSE, title_spec = NULL)
```

Arguments

A

T

fac

plot_spec

return_spec

title_spec

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, T = 0.8, fac = 1e+06, plot_spec = TRUE, return_spec = FALSE,
    title_spec = NULL)
{
    A = as.matrix(A)
    nr = nrow(A)
    nc = ncol(A)
    N = 2^(ceiling(log2(nr)))
    A = rbind(A, matrix(0, nrow = N - nr, ncol = nc))
    fft_A = mvfft(A)
    pow = as.matrix(Mod(fft_A))
    pow = as.matrix(Mod(fft_A))
    pha = as.matrix(Arg(fft_A))
    nfreq <- N/2 + 1
    pha = pha[1:nfreq, , drop = FALSE]
    pow = pow[1:nfreq, , drop = FALSE]
    pow_mean = apply(pow, 1, mean, na.rm = TRUE)
    unwrap_pha <- apply(pha, 2, unwrap)
    pha_mean = apply(unwrap_pha, 1, mean, na.rm = TRUE)
    Ts = T * (10^(-9))
    Fs = 1/Ts
    Fc = 1/(2 * Ts)
    fre = Fs * seq(0, N/2)/N/fac
    if (plot_spec) {
        m = seq(0, 10000, by = 50)
        par(mfrow = c(2, 1))
        par(mar = c(0, 4, 4, 2) + 0.1, oma = c(1, 1, 1, 1))
        plot(fre, pow_mean, type = "n", xaxt = "n", ylim = c(0,
            max(pow)), ylab = "amplitude", xlab = "")
        if (!is.null(dim(A))) {
            nothing <- apply(pow, 2, lines, x = fre, col = rgb(0.2,
                0.2, 0.2, 7/max(ncol(A), 7)))
        }
        lines(fre, pow_mean, col = "red")
        Axis(side = 1, tcl = +0.3, labels = FALSE, at = m)
        if (!is.null(title_spec)) {
            title(title_spec)
        }
        par(mar = c(4, 4, 0.3, 2))
        plot(fre, pha_mean, type = "n", xaxt = "n", ylim = range(unwrap_pha),
            xlab = "frequency MHz", ylab = "phase")
        if (!is.null(dim(A))) {
            nothing <- apply(unwrap_pha, 2, lines, x = fre, col = rgb(0.2,
                0.2, 0.2, 7/max(ncol(A), 7)))
        }
        lines(fre, pha_mean, col = "red")
        Axis(side = 1, tcl = +0.3, labels = m, at = m)
    }
    if (return_spec) {
        return(list(freq = fre, pow = pow, pha = pha))
    }
  }
```

print.GPR

Usage

```
print.GPR(x, ...)
```

Arguments

x

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, ...)
{
    jj <- .GPR.print(x, ...)
    cat(jj)
    return(invisible(jj))
  }
```

print.GPRsurvey

Usage

```
print.GPRsurvey(x, ...)
```

Arguments

x

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, ...)
{
    cat("*** Class GPRsurvey ***\n")
    n <- length(x)
    dirNames <- dirname(x@filepaths)
    if (length(unique(dirNames)) == 1) {
        cat("Unique directory:", dirNames[1], "\n")
    }
    else {
        cat("One directory among others:", dirNames[1], "\n")
    }
    testCoords <- rep(0, n)
    names(testCoords) <- x@names
    if (length(x@coords) > 0) {
        testLength <- sapply(x@coords, length)
        testCoords[names(testLength)] <- testLength
    }
    testCoords <- as.numeric(testCoords > 0) + 1
    testIntersecs <- rep(0, n)
    names(testIntersecs) <- x@names
    if (length(x@intersections) > 0) {
        testLength <- sapply(x@intersections, length)
        testIntersecs[names(testLength)] <- testLength
    }
    testIntersecs <- as.numeric(testIntersecs > 0) + 1
    is_test <- c("NO", "YES")
    cat("- - - - - - - - - - - - - - -\n")
    overview <- data.frame(name = .filename(x@filepaths), length = round(x@lengths,
        2), units = rep(x@posunit, n), date = x@dates, fequency = x@freqs,
        coordinates = is_test[testCoords], intersections = is_test[testIntersecs])
    print(overview)
    if (length(x@coords) > 0) {
        cat("- - - - - - - - - - - - - - -\n")
        if (x@crs != "") {
            cat("Coordinate system:", x@crs, "\n")
        }
        else {
            cat("Coordinate system: unknown\n")
        }
        cat
    }
    cat("****************\n")
    return(invisible(overview))
  }
```

range-methods *~~ Methods for Function* range *~~*

Description

~~ Methods for function `range` ~~

Methods

`signature(x = "GPR")`

---

readDT1

Usage

```
readDT1(filePath)
```

Arguments

`filePath`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (filePath)
{
    dirName <- dirname(filePath)
    splitBaseName <- unlist(strsplit(basename(filePath), "[.]"))
    baseName <- paste(splitBaseName[1:(length(splitBaseName) -
        1)], sep = "")
    fileNameHD <- paste(dirName, "/", baseName, ".HD", sep = "")
    fileNameDT1 <- paste(dirName, "/", baseName, ".DT1", sep = "")
    headHD <- scan(fileNameHD, what = character(), strip.white = TRUE,
        quiet = TRUE, fill = TRUE, blank.lines.skip = TRUE, flush = TRUE,
        sep = "\n")
    nHD <- length(headHD)
    headerHD <- data.frame(nrow = nHD, ncol = 2)
    for (i in seq_along(headHD)) {
        hdline <- strsplit(headHD[i], "=")[[1]]
        if (length(hdline) < 2) {
            headerHD[i, 1] <- ""
            headerHD[i, 2] <- trim(hdline[1])
        }
        else {
            headerHD[i, 1:2] <- as.character(sapply(hdline[1:2],
                trim))
        }
    }
    nbTraces = as.integer(as.character(headerHD[4, 2]))
    nbPt = as.integer(as.character(headerHD[5, 2]))
    dt1 <- file(fileNameDT1, "rb")
    indexDT1Header = c("traces", "position", "samples", "topo",
        "NA1", "bytes", "tracenb", "stack", "window", "NA2",
        "NA3", "NA4", "NA5", "NA6", "recx", "recy", "recz", "transx",
        "transy", "transz", "time0", "zeroflag", "NA7", "time",
        "x8", "com")
    headerDT1 = list()
    myData = matrix(NA, nrow = nbPt, ncol = nbTraces)
    for (i in 1:nbTraces) {
        for (j in 1:25) {
            headerDT1[[indexDT1Header[j]]][i] = readBin(dt1,
                what = numeric(), n = 1L, size = 4)
        }
        headerDT1[[indexDT1Header[26]]][i] = readChar(dt1, 28)
        myData[, i] = readBin(dt1, what = integer(), n = nbPt,
            size = 2)
    }
    close(dt1)
    return(list(hd = headerHD, dt1hd = headerDT1, data = myData))
  }
```

readFID

Usage

```
readFID(FID, sep = ",")
```

Arguments

FID

sep

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (FID, sep = ",")
{
    myFid <- list()
    for (i in seq_along(FID)) {
        A <- read.table(FID[[i]], sep = ",", stringsAsFactors = FALSE,
            header = TRUE)
        colnames(A) <- toupper(colnames(A))
        if (!all(c("E", "N", "Z", "TRACE") %in% colnames(A))) {
            stop("The headers should be \"E\",\"N\",\"Z\",\"TRACE\"!\n")
        }
        myFid[[i]] <- A[, c("E", "N", "Z", "TRACE")]
    }
    return(myFid)
  }
```

readGPR

Usage

```
readGPR(filename, description = "", coordfile = NULL, crs = "", intfile = NULL)
```

Arguments

filename

description

coordfile

crs

intfile

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (filename, description = "", coordfile = NULL,
    crs = "", intfile = NULL)
standardGeneric("readGPR"), generic = structure("readGPR", package = "RGPR"), package = "RGPR", group = list(
"description", "coordfile", "crs", "intfile"), default = `\001NULL\001`, skeleton = (function (filename,
    description = "", coordfile = NULL, crs = "", intfile = NULL)
stop("invalid call in method dispatch to 'readGPR' (no default method)",
   domain = NA))(filename, description, coordfile, crs, intfile), class = structure("standardGeneric", packa
```

readGPR-methods ~~ *Methods for Function* readGPR ~~

Description

~~ Methods for function readGPR ~~

Methods

```
signature(filename = "character")
```

readTopo

Usage

```
readTopo(TOPO, sep = ",")
```

Arguments

TOPO

sep

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (TOPO, sep = ",")
{
    myTopo <- list()
    for (i in seq_along(TOPO)) {
        A <- read.table(TOPO[[i]], sep = ",", stringsAsFactors = FALSE,
            header = TRUE)
        colnames(A) <- toupper(colnames(A))
        if (ncol(A) < 3) {
            stop("The headers should be \"E\",\"N\",\"Z\"!\n")
        }
        myTopo[[i]] <- A[, 1:3]
    }
    return(myTopo)
  }
```

repmat

Usage

```
repmat(a, n, m)
```

Arguments

a

n

m

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (a, n, m)
{
    kronecker(matrix(1, n, m), a)
  }
```

requiredPackage

Usage

```
data("requiredPackage")
```

Format

The format is: chr [1:10] "MASS" "signal" "colorspace" "Cairo" "rgeos" "sp" "rgl" ...

Examples

```
data(requiredPackage)
## maybe str(requiredPackage) ; plot(requiredPackage) ...
```

reverse

Usage

```
reverse(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("reverse"), generic = structure("reverse", package = "RGPR"), package = "RGPR", group = list(
stop("invalid call in method dispatch to 'reverse' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

reverse-methods *~~ Methods for Function* reverse *~~*

Description

~~ Methods for function reverse ~~

Methods

signature(x = "GPR")

rmDelineations<-

Usage

```
rmDelineations<-(x, values = NULL)
```

Arguments

x

values

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, values = NULL)
standardGeneric("rmDelineations<-"), generic = structure("rmDelineations<-", package = "RGPR"), package = "R
"values"), default = `\001NULL\001`, skeleton = (function (x,
    values = NULL)
stop("invalid call in method dispatch to 'rmDelineations<-' (no default method)",
    domain = NA))(x, values), class = structure("standardGeneric", package = "methods"))
```

rmDelineations<--methods *~~ Methods for Function* rmDelineations<- *~~*

Description

~~ Methods for function rmDelineations<- ~~

Methods

signature(x = "GPR")

selectBBox

Usage

```
selectBBox(border = "red", lwd = 2, ...)
```

Arguments

border

lwd

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (border = "red", lwd = 2, ...)
{
    bbox <- locator(type = "p", n = 2)
    LIM <- sapply(bbox, range)
    rect(LIM[1, "x"], LIM[1, "y"], LIM[2, "x"], LIM[2, "y"],
        border = border)
    return(list(xlim = LIM[, "x"], ylim = LIM[, "y"]))
  }
```

setCol

Usage

```
setCol(A, col = diverge_hcl(101, h = c(246, 10), c = 120, l = c(30, 90)))
```

Arguments

A

col

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, col = diverge_hcl(101, h = c(246, 10), c = 120,
    l = c(30, 90)))
{
    CCY = (A - min(A))/(max(A) - min(A))
    ClimY <- range(CCY)
    ClenY <- ClimY[2] - ClimY[1] + 1
    col[(CCY) * 100 + 1]
  }
```

setCoordref

Usage

```
setCoordref(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("setCoordref"), generic = structure("setCoordref", package = "RGPR"), package = "RGPR", grou
stop("invalid call in method dispatch to 'setCoordref' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

setCoordref-methods ~~ *Methods for Function* setCoordref ~~

Description

~~ Methods for function setCoordref ~~

Methods

signature(x = "GPRsurvey")

setData<-

Usage

```
setData<-(x, value)
```

Arguments

x

value

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, value)
standardGeneric("setData<-"), generic = structure("setData<-", package = "RGPR"), package = "RGPR", group = l
"value"), default = `\001NULL\001`, skeleton = (function (x,
    value)
stop("invalid call in method dispatch to 'setData<-' (no default method)",
    domain = NA))(x, value), class = structure("standardGeneric", package = "methods"))
```

setData<--methods ~~ *Methods for Function* setData<- ~~

Description

~~ Methods for function setData<- ~~

Methods

signature(x = "GPR")

setGenericVerif

Usage

```
setGenericVerif(x, y)
```

Arguments

x

y

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y)
{
    setGeneric(x, y)
  }
```

---

show-methods *~~ Methods for Function* show *~~*

Description

~~ Methods for function show ~~

Methods

`signature(object = "GPR")`

`signature(object = "GPRsurvey")`

---

showDelineations

Usage

```
showDelineations(x, sel = NULL, ...)
```

Arguments

x

sel

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, sel = NULL, ...)
standardGeneric("showDelineations"), generic = structure("showDelineations", package = "RGPR"), package = "R
"sel"), default = `\001NULL\001`, skeleton = (function (x, sel = NULL,
    ...)
stop("invalid call in method dispatch to 'showDelineations' (no default method)",
   domain = NA))(x, sel, ...), class = structure("standardGeneric", package = "methods"))
```

showDelineations-methods

~~ Methods for Function showDelineations ~~

Description

~~ Methods for function showDelineations ~~

Methods

signature(x = "GPR")

sincMod

Usage

```
sincMod(x, ff)
```

Arguments

x

ff

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, ff)
{
    r = length(x)
    n0 = which(x == 0)
    v = rep(0, r)
    ww <- c(1:(n0 - 1), (n0 + 1):r)
    v[ww] = sin(ff * x[ww])/(x[ww])
    v[n0] = ff
    return(v)
  }
```

spec

Usage

```
spec(x, type = c("f-x", "f-k"), return_spec = FALSE, plot_spec = TRUE, ...)
```

Arguments

x

type

return_spec

plot_spec

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, type = c("f-x", "f-k"), return_spec = FALSE,
    plot_spec = TRUE, ...)
standardGeneric("spec"), generic = structure("spec", package = "RGPR"), package = "RGPR", group = list(), val
"type", "return_spec", "plot_spec"), default = `\001NULL\001`, skeleton = (function (x,
    type = c("f-x", "f-k"), return_spec = FALSE, plot_spec = TRUE,
    ...)
stop("invalid call in method dispatch to 'spec' (no default method)",
   domain = NA))(x, type, return_spec, plot_spec, ...), class = structure("standardGeneric", package = "metho
```

spec-methods ~~ *Methods for Function* spec ~~

Description

~~ Methods for function spec ~~

Methods

signature(x = "GPR")

spikingFilter

Usage

```
spikingFilter(y, nf = 32, mu = 0.1, shft = 1)
```

Arguments

y

nf

mu

shft

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, nf = 32, mu = 0.1, shft = 1)
{
    y_acf <- as.numeric(acf(y, lag = nf - 1, plot = FALSE)[[1]])
    y_acf[1] <- y_acf[1] + mu
    YtY <- toeplitz(y_acf)
    if (is.null(shft)) {
        ny <- length(y)
        L <- nf + ny - 1
        Y <- convmtx(y, nf)
        H <- solve(YtY) %*% t(Y)
        v <- numeric(L)
        P <- Y %*% H
        i <- which.max(diag(P))
        v[i] <- 1
        h <- H %*% v
        return(list(h = h, delay = i))
    }
    else {
        v <- numeric(nf)
        v[shft] <- 1
        h <- solve(YtY) %*% v
        return(h)
    }
  }
```

summary-methods *~~ Methods for Function* summary *~~*

Description

~~ Methods for function summary ~~

Methods

signature(object = "GPR")

surveyIntersections

Usage

```
surveyIntersections(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("surveyIntersections"), generic = structure("surveyIntersections", package = "RGPR"), packag
stop("invalid call in method dispatch to 'surveyIntersections' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

surveyIntersections-methods *~~ Methods for Function* surveyIntersections *~~*

Description

~~ Methods for function surveyIntersections ~~

Methods

signature(x = "GPRsurvey")

time0

Usage

```
time0(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x)
standardGeneric("time0"), generic = structure("time0", package = "RGPR"), package = "RGPR", group = list(), v
stop("invalid call in method dispatch to 'time0' (no default method)",
    domain = NA))(x), class = structure("standardGeneric", package = "methods"))
```

time0-methods ~~ *Methods for Function* time0 ~~

Description

~~ Methods for function time0 ~~

Methods

signature(x = "GPR")

time0<-

Usage

```
time0<-(x, value)
```

Arguments

x

value

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, value)
{
    standardGeneric("time0<-")
  }, generic = structure("time0<-", package = "RGPR"), package = "RGPR", group = list(), valueClass = characte
"value"), default = `\001NULL\001`, skeleton = (function (x,
    value)
stop("invalid call in method dispatch to 'time0<-' (no default method)",
   domain = NA))(x, value), class = structure("nonstandardGenericFunction", package = "methods"))
```

`time0<--methods` ~~ *Methods for Function* `time0<-` ~~

Description

~~ Methods for function `time0<-` ~~

Methods

`signature(x = "GPR")`

`timeToDepth`

Usage

```
timeToDepth(tt, time0, v = 0.1, antsep = 1)
```

Arguments

`tt`

`time0`

`v`

`antsep`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (tt, time0, v = 0.1, antsep = 1)
{
    t0 <- time0 - antsep/0.299
    sqrt(v^2 * (tt - t0) - antsep^2)/2
  }
```

topoShift

Usage

```
topoShift(A, topo, dz)
```

Arguments

A

topo

dz

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (A, topo, dz)
{
    zShift <- (max(topo) - topo)
    old_t <- seq(0, length.out = nrow(A), by = dz)
    A_topoShift <- matrix(0, nrow = nrow(A) + floor(max(zShift)/dz),
        ncol = ncol(A))
    n <- 1:(nrow(A) - 2)
    for (i in 1:ncol(A)) {
        new_t <- old_t + zShift[i]
        xit <- seq(ceiling(new_t[1]/dz), ceiling(new_t[nrow(A) -
            2]/dz))
        A_topoShift[xit + 1, i] = signal::interp1(new_t, A[,
            i], xi = xit * dz, method = "cubic", extrap = TRUE)
    }
    return(A_topoShift)
  }
```

trim

Usage

```
trim(x)
```

Arguments

x

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x)
gsub("^\s+|\s+$", "", x)
```

upsample

Usage

```
upsample(x, n)
```

Arguments

x

n

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, n)
standardGeneric("upsample"), generic = structure("upsample", package = "RGPR"), package = "RGPR", group = lis
"n"), default = `\001NULL\001`, skeleton = (function (x, n)
stop("invalid call in method dispatch to 'upsample' (no default method)",
    domain = NA))(x, n), class = structure("standardGeneric", package = "methods"))
```

upsample-methods ~~ *Methods for Function* upsample ~~

Description

~~ Methods for function upsample ~~

Methods

signature(x = "GPR")

wapply

Usage

```
wapply(x, width, by = NULL, FUN = NULL, ...)
```

Arguments

x

width

by

FUN

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, width, by = NULL, FUN = NULL, ...)
{
    FUN <- match.fun(FUN)
    if (is.null(by))
        by <- width
    lenX <- length(x)
    SEQ1 <- seq(1, lenX - width + 1, by = by)
    SEQ2 <- lapply(SEQ1, function(x) x:(x + width - 1))
    OUT <- lapply(SEQ2, function(a) FUN(x[a], ...))
    OUT <- base:::simplify2array(OUT, higher = TRUE)
    return(OUT)
  }
```

winSincKernel

Usage

```
winSincKernel(L, f, type = c("low", "high"))
```

Arguments

L

f

type

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (L, f, type = c(”low”, ”high”))
{
    type = match.arg(type)
    x = (-(L - 1)/2):((L - 1)/2)
    h = hammingWindow(L) * sincMod(x, 2 * pi * f)
    h = h/sum(h)
    if (type == ”high”) {
        h = -h
        h[(L + 1)/2] = h[(L + 1)/2] + 1
    }
    return(h)
  }
```

writeGPR

Usage

```
writeGPR(x, path, format = c(”DT1”, ”rds”))
```

Arguments

x

path

format

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
structure(function (x, path, format = c(”DT1”, ”rds”))
standardGeneric(”writeGPR”), generic = structure(”writeGPR”, package = ”RGPR”), package = ”RGPR”, group = lis
”path”, ”format”), default = `\001NULL\001`, skeleton = (function (x,
    path, format = c(”DT1”, ”rds”))
stop(”invalid call in method dispatch to 'writeGPR' (no default method)”,
   domain = NA))(x, path, format), class = structure(”standardGeneric”, package = ”methods”))
```

`writeGPR-methods` *~~ Methods for Function* `writeGPR` *~~*

Description

~~ Methods for function `writeGPR` ~~

Methods

`signature(x = "GPR")`

`signature(x = "GPRsurvey")`

`[-methods` *~~ Methods for Function* `[` *~~*

Description

~~ Methods for function `[` ~~

Methods

`signature(x = "GPR", i = "ANY", j = "ANY", drop = "ANY")`

`signature(x = "GPRsurvey", i = "ANY", j = "ANY", drop = "ANY")`

`[<--methods` *~~ Methods for Function* `[<-` *~~*

Description

~~ Methods for function `[<-` ~~

Methods

`signature(x = "GPR", i = "ANY", j = "ANY", value = "ANY")`

Index

∗Topic **\textasciitilde\textasciitilde other possible keyword(s) \textasciitilde\textasciitilde**
[-methods, 121
[<--methods, 121
ann-methods, 9
ann<--methods, 10
apply-methods, 10
Arith-methods, 10
as.matrix-methods, 11
clip-methods, 13
coerce-methods, 13
coord-methods, 16
coord<--methods, 16
coords<--methods, 17
crs-methods, 18
crs<--methods, 19
dcshift-methods, 20
delineate-methods, 24
delineations-methods, 25
description-methods, 26
dewow-methods, 27
dewow2-methods, 28
dim-methods, 28
exportDelineations-methods, 32
exportFID-methods, 33
exportPDF-methods, 34
fid-methods, 35
fid<--methods, 36
filename-methods, 37
firstBreack-methods, 38
fkFilter-methods, 40
freqFilter-methods, 42
gain-methods, 47
gammaCorrection-methods, 49
getAmpl-methods, 50
getData-methods, 51
gethd-methods, 53
getLine-methods, 53
identifyDelineation-methods, 63
interpTraces-methods, 64
intersections-methods, 65
length-methods, 66
Math-methods, 69
max-methods, 69
mean-methods, 69
medianFilter-methods, 70
migration-methods, 71
min-methods, 71
name-methods, 74
ncol-methods, 74
nrow-methods, 75
plot3D-methods, 84
plotAmpl-methods, 87
plotDelineations-methods, 88
plotDelineations3D-methods, 89
range-methods, 100
readGPR-methods, 103
reverse-methods, 106
rmDelineations<--methods, 106
setCoordref-methods, 108
setData<--methods, 109
show-methods, 110
showDelineations-methods, 111
spec-methods, 112
summary-methods, 114
surveyIntersections-methods, 114
time0-methods, 115
time0<--methods, 116
upsample-methods, 118
writeGPR-methods, 121

∗Topic **\textasciitildekwd1**
acfmtx, 6
addArg, 6
addProfile3D, 7
ann, 8
ann<-, 9
ar_fb, 11
byte2volt, 12
clip, 12
convmtx, 13
convolution, 14
convolution2D, 14
coord, 15
coord<-, 16
coords<-, 17

crs, 18
crs<-, 18
dcshift, 19
decon_spiking, 21
decon_spiking_matrix, 22
deconvFreq, 20
delineate, 23
delineations, 24
depth0, 25
depthToTime, 25
description, 26
dewow, 27
dewow2, 28
doubleVector, 29
dx_gkernel, 29
dy_gkernel, 30
eps, 31
exportDelineations, 31
exportFID, 32
exportPDF, 33
extension, 34
fid, 34
fid<-, 35
fidpos, 36
filename, 36
firstBreack, 37
firstBreackPicking, 38
FKFilter, 39
fkFilter, 40
FKSpectrum, 41
freqFilter, 42
freqFilter1D, 43
fx_deconv, 45
gain, 46
gain_agc, 47
gain_exp, 48
gain_geospreading, 48
gammaCorrection, 49
get_args, 54
getAmpl, 50
getData, 51
getHD, 51
gethd, 52
getLine, 53
gkernel, 54
GPR, 55
GPRsurvey, 59
hammingWindow, 62
identifyDelineation, 62
inPoly, 63
interpTraces, 64
intersections, 65
is_installed, 65
lengthList, 66
lineDist, 67
load_install_package, 67
localOrientation, 68
medianFilter, 70
migration, 70
minCommon10, 71
myWhich, 72
myWhichMin, 73
name, 73
nextpower2, 74
normalize, 75
paddMatrix, 76
pasteArgs, 77
phaseRotation, 77
plot.GPR, 78
plot.GPRsurvey, 81
plot3D, 83
plot3DSlice, 84
plotAmpl, 86
plotArrows, 87
plotDelineations, 88
plotDelineations3D, 89
plotLine, 90
plotRaster, 90
plotTopo, 93
plotWig, 94
powSpec, 97
print.GPR, 99
print.GPRsurvey, 99
readDT1, 101
readFID, 102
readGPR, 103
readTopo, 104
repmat, 104
reverse, 105
rmDelineations<-, 106
selectBBox, 107
setCol, 107
setCoordref, 108
setData<-, 109
setGenericVerif, 109
showDelineations, 110
sincMod, 111
spec, 112
spikingFilter, 113
surveyIntersections, 114
time0, 115
time0<-, 115
timeToDepth, 116
topoShift, 117

trim, 117
upsample, 118
wapply, 119
winSincKernel, 119
writeGPR, 120
∗Topic **\textasciitildekwd2**
acfmtx, 6
addArg, 6
addProfile3D, 7
ann, 8
ann<-, 9
ar_fb, 11
byte2volt, 12
clip, 12
convmtx, 13
convolution, 14
convolution2D, 14
coord, 15
coord<-, 16
coords<-, 17
crs, 18
crs<-, 18
dcshift, 19
decon_spiking, 21
decon_spiking_matrix, 22
deconvFreq, 20
delineate, 23
delineations, 24
depth0, 25
depthToTime, 25
description, 26
dewow, 27
dewow2, 28
doubleVector, 29
dx_gkernel, 29
dy_gkernel, 30
eps, 31
exportDelineations, 31
exportFID, 32
exportPDF, 33
extension, 34
fid, 34
fid<-, 35
fidpos, 36
filename, 36
firstBreack, 37
firstBreackPicking, 38
FKFilter, 39
fkFilter, 40
FKSpectrum, 41
freqFilter, 42
freqFilter1D, 43
fx_deconv, 45
gain, 46
gain_agc, 47
gain_exp, 48
gain_geospreading, 48
gammaCorrection, 49
get_args, 54
getAmpl, 50
getData, 51
getHD, 51
gethd, 52
getLine, 53
gkernel, 54
GPR, 55
GPRsurvey, 59
hammingWindow, 62
identifyDelineation, 62
inPoly, 63
interpTraces, 64
intersections, 65
is_installed, 65
lengthList, 66
lineDist, 67
load_install_package, 67
localOrientation, 68
medianFilter, 70
migration, 70
minCommon10, 71
myWhich, 72
myWhichMin, 73
name, 73
nextpower2, 74
normalize, 75
paddMatrix, 76
pasteArgs, 77
phaseRotation, 77
plot.GPR, 78
plot.GPRsurvey, 81
plot3D, 83
plot3DSlice, 84
plotAmpl, 86
plotArrows, 87
plotDelineations, 88
plotDelineations3D, 89
plotLine, 90
plotRaster, 90
plotTopo, 93
plotWig, 94
powSpec, 97
print.GPR, 99
print.GPRsurvey, 99
readDT1, 101

readFID, 102
readGPR, 103
readTopo, 104
repmat, 104
reverse, 105
rmDelineations<-, 106
selectBBox, 107
setCol, 107
setCoordref, 108
setData<-, 109
setGenericVerif, 109
showDelineations, 110
sincMod, 111
spec, 112
spikingFilter, 113
surveyIntersections, 114
time0, 115
time0<-, 115
timeToDepth, 116
topoShift, 117
trim, 117
upsample, 118
wapply, 119
winSincKernel, 119
writeGPR, 120
∗Topic **classes**
GPR-class, 57
GPRsurvey-class, 61
∗Topic **datasets**
requiredPackage, 105
∗Topic **methods**
[-methods, 121
[<--methods, 121
ann-methods, 9
ann<--methods, 10
apply-methods, 10
Arith-methods, 10
as.matrix-methods, 11
clip-methods, 13
coerce-methods, 13
coord-methods, 16
coord<--methods, 16
coords<--methods, 17
crs-methods, 18
crs<--methods, 19
dcshift-methods, 20
delineate-methods, 24
delineations-methods, 25
description-methods, 26
dewow-methods, 27
dewow2-methods, 28
dim-methods, 28
exportDelineations-methods, 32
exportFID-methods, 33
exportPDF-methods, 34
fid-methods, 35
fid<--methods, 36
filename-methods, 37
firstBreack-methods, 38
fkFilter-methods, 40
freqFilter-methods, 42
gain-methods, 47
gammaCorrection-methods, 49
getAmpl-methods, 50
getData-methods, 51
gethd-methods, 53
getLine-methods, 53
identifyDelineation-methods, 63
interpTraces-methods, 64
intersections-methods, 65
length-methods, 66
Math-methods, 69
max-methods, 69
mean-methods, 69
medianFilter-methods, 70
migration-methods, 71
min-methods, 71
name-methods, 74
ncol-methods, 74
nrow-methods, 75
plot3D-methods, 84
plotAmpl-methods, 87
plotDelineations-methods, 88
plotDelineations3D-methods, 89
range-methods, 100
readGPR-methods, 103
reverse-methods, 106
rmDelineations<--methods, 106
setCoordref-methods, 108
setData<--methods, 109
show-methods, 110
showDelineations-methods, 111
spec-methods, 112
summary-methods, 114
surveyIntersections-methods, 114
time0-methods, 115
time0<--methods, 116
upsample-methods, 118
writeGPR-methods, 121
∗Topic **package**
RGPR-package, 5
<pkg>, *5*
[,GPR,ANY,ANY,ANY-method *(*GPR-class*)*, 57

[,GPR,ANY,ANY,ANY-method ([-methods), 121
[,GPRsurvey,ANY,ANY,ANY-method (GPRsurvey-class), 61
[,GPRsurvey,ANY,ANY,ANY-method ([-methods), 121
[-methods, 121
[<-,GPR,ANY,ANY,ANY-method (GPR-class), 57
[<-,GPR,ANY,ANY,ANY-method ([<--methods), 121
[<--methods, 121

acfmtx, 6
addArg, 6
addProfile3D, 7
ann, 8
ann,GPR-method (GPR-class), 57
ann,GPR-method (ann-methods), 9
ann-methods, 9
ann<-, 9
ann<-,GPR-method (GPR-class), 57
ann<-,GPR-method (ann<--methods), 10
ann<--methods, 10
apply,ANY-method (apply-methods), 10
apply,GPR-method (GPR-class), 57
apply,GPR-method (apply-methods), 10
apply-methods, 10
ar_fb, 11
Arith,ANY,GPR-method (Arith-methods), 10
Arith,ANY,GPR-method (GPR-class), 57
Arith,GPR,ANY-method (Arith-methods), 10
Arith,GPR,ANY-method (GPR-class), 57
Arith,GPR,GPR-method (Arith-methods), 10
Arith,GPR,GPR-method (GPR-class), 57
Arith-methods, 10
as.matrix,GPR-method (GPR-class), 57
as.matrix,GPR-method (as.matrix-methods), 11
as.matrix-methods, 11

byte2volt, 12

clip, 12
clip,GPR-method (GPR-class), 57
clip,GPR-method (clip-methods), 13
clip-methods, 13
coerce,GPR,matrix-method (GPR-class), 57
coerce,GPR,matrix-method (coerce-methods), 13
coerce-methods, 13
convmtx, 13
convolution, 14
convolution2D, 14
coord, 15
coord,GPR-method (GPR-class), 57
coord,GPR-method (coord-methods), 16
coord-methods, 16
coord<-, 16
coord<-,GPR-method (GPR-class), 57
coord<-,GPR-method (coord<--methods), 16
coord<--methods, 16
coords<-, 17
coords<-,GPRsurvey-method (GPRsurvey-class), 61
coords<-,GPRsurvey-method (coords<--methods), 17
coords<--methods, 17
crs, 18
crs,GPR-method (GPR-class), 57
crs,GPR-method (crs-methods), 18
crs-methods, 18
crs<-, 18
crs<-,GPR-method (GPR-class), 57
crs<-,GPR-method (crs<--methods), 19
crs<--methods, 19

dcshift, 19
dcshift,GPR-method (GPR-class), 57
dcshift,GPR-method (dcshift-methods), 20
dcshift-methods, 20
decon_spiking, 21
decon_spiking_matrix, 22
deconvFreq, 20
delineate, 23
delineate,GPR-method (GPR-class), 57
delineate,GPR-method (delineate-methods), 24
delineate-methods, 24
delineations, 24
delineations,GPR-method (GPR-class), 57
delineations,GPR-method (delineations-methods), 25
delineations-methods, 25
depth0, 25
depthToTime, 25
description, 26
description,GPR-method (GPR-class), 57
description,GPR-method (description-methods), 26
description-methods, 26
dewow, 27
dewow,GPR-method (GPR-class), 57
dewow,GPR-method (dewow-methods), 27
dewow-methods, 27
dewow2, 28

dewow2,GPR-method *(*GPR-class*)*, 57
dewow2,GPR-method *(*dewow2-methods*)*, 28
dewow2-methods, 28
dim,GPR-method *(*GPR-class*)*, 57
dim,GPR-method *(*dim-methods*)*, 28
dim-methods, 28
doubleVector, 29
dx_gkernel, 29
dy_gkernel, 30

eps, 31
exportDelineations, 31
exportDelineations,GPR-method *(*GPR-class*)*, 57
exportDelineations,GPR-method *(*exportDelineations-methods*)*, 32
exportDelineations-methods, 32
exportFID, 32
exportFID,GPR-method *(*GPR-class*)*, 57
exportFID,GPR-method *(*exportFID-methods*)*, 33
exportFID,GPRsurvey-method *(*GPRsurvey-class*)*, 61
exportFID,GPRsurvey-method *(*exportFID-methods*)*, 33
exportFID-methods, 33
exportPDF, 33
exportPDF,GPR-method *(*GPR-class*)*, 57
exportPDF,GPR-method *(*exportPDF-methods*)*, 34
exportPDF-methods, 34
extension, 34

fid, 34
fid,GPR-method *(*GPR-class*)*, 57
fid,GPR-method *(*fid-methods*)*, 35
fid-methods, 35
fid<-, 35
fid<-,GPR-method *(*GPR-class*)*, 57
fid<-,GPR-method *(*fid<--methods*)*, 36
fid<--methods, 36
fidpos, 36
filename, 36
filename,GPR-method *(*GPR-class*)*, 57
filename,GPR-method *(*filename-methods*)*, 37
filename-methods, 37
firstBreack, 37
firstBreack,GPR-method *(*GPR-class*)*, 57
firstBreack,GPR-method *(*firstBreack-methods*)*, 38
firstBreack-methods, 38
firstBreackPicking, 38
FKFilter, 39
fkFilter, 40
fkFilter,GPR-method *(*GPR-class*)*, 57
fkFilter,GPR-method *(*fkFilter-methods*)*, 40
fkFilter-methods, 40
FKSpectrum, 41
freqFilter, 42
freqFilter,GPR-method *(*GPR-class*)*, 57
freqFilter,GPR-method *(*freqFilter-methods*)*, 42
freqFilter-methods, 42
freqFilter1D, 43
fx_deconv, 45

gain, 46
gain,GPR-method *(*GPR-class*)*, 57
gain,GPR-method *(*gain-methods*)*, 47
gain-methods, 47
gain_agc, 47
gain_exp, 48
gain_geospreading, 48
gammaCorrection, 49
gammaCorrection,GPR-method *(*GPR-class*)*, 57
gammaCorrection,GPR-method *(*gammaCorrection-methods*)*, 49
gammaCorrection-methods, 49
get_args, 54
getAmpl, 50
getAmpl,GPR-method *(*GPR-class*)*, 57
getAmpl,GPR-method *(*getAmpl-methods*)*, 50
getAmpl-methods, 50
getData, 51
getData,GPR-method *(*GPR-class*)*, 57
getData,GPR-method *(*getData-methods*)*, 51
getData-methods, 51
getHD, 51
gethd, 52
gethd,GPR-method *(*GPR-class*)*, 57
gethd,GPR-method *(*gethd-methods*)*, 53
gethd-methods, 53
getLine, 53
getLine,GPRsurvey-method *(*GPRsurvey-class*)*, 61
getLine,GPRsurvey-method *(*getLine-methods*)*, 53
getLine-methods, 53
gkernel, 54
GPR, 55
GPR-class, 57
GPRsurvey, 59

GPRsurvey-class, 61

hammingWindow, 62

identifyDelineation, 62
identifyDelineation,GPR-method (GPR-class), 57
identifyDelineation,GPR-method (identifyDelineation-methods), 63
identifyDelineation-methods, 63
inPoly, 63
interpTraces, 64
interpTraces,GPR-method (GPR-class), 57
interpTraces,GPR-method (interpTraces-methods), 64
interpTraces,GPRsurvey-method (GPRsurvey-class), 61
interpTraces,GPRsurvey-method (interpTraces-methods), 64
interpTraces-methods, 64
intersections, 65
intersections,GPRsurvey-method (GPRsurvey-class), 61
intersections,GPRsurvey-method (intersections-methods), 65
intersections-methods, 65
is_installed, 65

length,GPR-method (GPR-class), 57
length,GPR-method (length-methods), 66
length,GPRsurvey-method (GPRsurvey-class), 61
length,GPRsurvey-method (length-methods), 66
length-methods, 66
lengthList, 66
lineDist, 67
load_install_package, 67
localOrientation, 68

Math,GPR-method (GPR-class), 57
Math,GPR-method (Math-methods), 69
Math-methods, 69
max,GPR-method (GPR-class), 57
max,GPR-method (max-methods), 69
max-methods, 69
mean,ANY-method (mean-methods), 69
mean,GPR-method (GPR-class), 57
mean,GPR-method (mean-methods), 69
mean-methods, 69
medianFilter, 70
medianFilter,GPR-method (GPR-class), 57
medianFilter,GPR-method (medianFilter-methods), 70
medianFilter-methods, 70
migration, 70
migration,GPR-method (GPR-class), 57
migration,GPR-method (migration-methods), 71
migration-methods, 71
min,GPR-method (GPR-class), 57
min,GPR-method (min-methods), 71
min-methods, 71
minCommon10, 71
myWhich, 72
myWhichMin, 73

name, 73
name,GPR-method (GPR-class), 57
name,GPR-method (name-methods), 74
name-methods, 74
ncol,ANY-method (ncol-methods), 74
ncol,GPR-method (GPR-class), 57
ncol,GPR-method (ncol-methods), 74
ncol-methods, 74
nextpower2, 74
normalize, 75
nrow,ANY-method (nrow-methods), 75
nrow,GPR-method (GPR-class), 57
nrow,GPR-method (nrow-methods), 75
nrow-methods, 75

paddMatrix, 76
pasteArgs, 77
phaseRotation, 77
plot.GPR, 78
plot.GPRsurvey, 81
plot3D, 83
plot3D,GPR-method (GPR-class), 57
plot3D,GPR-method (plot3D-methods), 84
plot3D,GPRsurvey-method (GPRsurvey-class), 61
plot3D,GPRsurvey-method (plot3D-methods), 84
plot3D-methods, 84
plot3DSlice, 84
plotAmpl, 86
plotAmpl,GPR-method (GPR-class), 57
plotAmpl,GPR-method (plotAmpl-methods), 87
plotAmpl-methods, 87
plotArrows, 87
plotDelineations, 88
plotDelineations,GPR-method (GPR-class), 57

plotDelineations,GPR-method
(plotDelineations-methods), 88
plotDelineations-methods, 88
plotDelineations3D, 89
plotDelineations3D,GPR-method
(GPR-class), 57
plotDelineations3D,GPR-method
(plotDelineations3D-methods), 89
plotDelineations3D,GPRsurvey-method
(GPRsurvey-class), 61
plotDelineations3D,GPRsurvey-method
(plotDelineations3D-methods), 89
plotDelineations3D-methods, 89
plotLine, 90
plotRaster, 90
plotTopo, 93
plotWig, 94
powSpec, 97
print.GPR, 99
print.GPRsurvey, 99

range,GPR-method *(GPR-class)*, 57
range,GPR-method *(range-methods)*, 100
range-methods, 100
readDT1, 101
readFID, 102
readGPR, 103
readGPR,character-method
(readGPR-methods), 103
readGPR-methods, 103
readTopo, 104
repmat, 104
requiredPackage, 105
reverse, 105
reverse,GPR-method *(GPR-class)*, 57
reverse,GPR-method *(reverse-methods)*, 106
reverse-methods, 106
RGPR *(RGPR-package)*, 5
RGPR-package, 5
rmDelineations<-, 106
rmDelineations<-,GPR-method
(GPR-class), 57
rmDelineations<-,GPR-method
(rmDelineations<--methods), 106
rmDelineations<--methods, 106

selectBBox, 107
setCol, 107
setCoordref, 108
setCoordref,GPRsurvey-method
(GPRsurvey-class), 61
setCoordref,GPRsurvey-method
(setCoordref-methods), 108
setCoordref-methods, 108
setData<-, 109
setData<-,GPR-method *(GPR-class)*, 57
setData<-,GPR-method
(setData<--methods), 109
setData<--methods, 109
setGenericVerif, 109
show,GPR-method *(GPR-class)*, 57
show,GPR-method *(show-methods)*, 110
show,GPRsurvey-method
(GPRsurvey-class), 61
show,GPRsurvey-method *(show-methods)*, 110
show-methods, 110
showDelineations, 110
showDelineations,GPR-method
(GPR-class), 57
showDelineations,GPR-method
(showDelineations-methods), 111
showDelineations-methods, 111
sincMod, 111
spec, 112
spec,GPR-method *(GPR-class)*, 57
spec,GPR-method *(spec-methods)*, 112
spec-methods, 112
spikingFilter, 113
summary,GPR-method *(GPR-class)*, 57
summary,GPR-method *(summary-methods)*, 114
summary-methods, 114
surveyIntersections, 114
surveyIntersections,GPRsurvey-method
(GPRsurvey-class), 61
surveyIntersections,GPRsurvey-method
(surveyIntersections-methods), 114
surveyIntersections-methods, 114

time0, 115
time0,GPR-method *(GPR-class)*, 57
time0,GPR-method *(time0-methods)*, 115
time0-methods, 115
time0<-, 115
time0<-,GPR-method *(GPR-class)*, 57
time0<-,GPR-method *(time0<--methods)*, 116
time0<--methods, 116
timeToDepth, 116
topoShift, 117

trim, 117

upsample, 118
upsample,GPR-method *(*GPR-class*)*, 57
upsample,GPR-method *(*upsample-methods*)*, 118
upsample-methods, 118

wapply, 119
winSincKernel, 119
writeGPR, 120
writeGPR,GPR-method *(*GPR-class*)*, 57
writeGPR,GPR-method *(*writeGPR-methods*)*, 121
writeGPR,GPRsurvey-method *(*GPRsurvey-class*)*, 61
writeGPR,GPRsurvey-method *(*writeGPR-methods*)*, 121
writeGPR-methods, 121