Package 'bLSmodelR'

May 26, 2018

Type Package

Title bLSmodelR - An atmospheric dispersion model in R

Version 4.1-0

Date 2018-05-26

Author Christoph Haeni

Maintainer Christoph Haeni <christoph.haeni@bfh.ch>

Description
This is the backward Lagrangian stochastic (bLS) model as described by Flesch et al. (2004).

License GPL-2

Depends R (>= 3.5.0), rlecuyer, snow, snowfall, data.table (>= 1.11.2), maptools

Imports Rcpp (>= 0.12.17), SDMTools, tcltk, parallel, sp, rgeos, geosphere, RgoogleMaps

LinkingTo Rcpp

NeedsCompilation yes

Archs i386, x64

R topics documented:

bLSmodelR-package *bLSmodelR - An atmospheric dispersion model in R*

Description

The package **bLSmodelR** provides functions to set up and run a backward Lagrangian Stochastic (bLS) model.

bLS Model

The model is a first-order Lagrangian Stochastic model that is run in backward mode (i.e. backward in time), assuming horizontally homogeneous and vertically inhomogeneous Gaussian turbulence. It is based on the paper published by *Flesch et al. (2004)*. The basis of the model calculation is a generalized Langevin equation given as

$$du_i = a_i(x,u) * dt + b_{ij}(x,u) * d\xi_j$$

$$dx_i = u_i * dt$$

where $d\xi$ in the noise term is a random increment from a Gaussian distribution $N(0, dt)$. The indices $i, j = 1, 2, 3$ represent the alongwind (x), crosswind (y) and vertical axis (z) of the rotated coordinate system. x_i and u_i are position (x, y, z) and velocity (u, v, w) of the trajectory at time t. An ensemble of upwind trajectories is calculated starting from the sensor position $x = (0, 0, z_{meas} - d)$ and the position and velocity of trajectory touchdowns on ground (i.e. on a horizontal plane at $z = z_0 + d$, where the modelled, average wind speed equals 0.) are recorded to calculate the sensor concentration to source strength relationship as the sum of the inverse touchdown velocities, summed over all touchdowns within the source

$$\frac{C}{E} = \frac{2}{N_{trj}} * \sum_{src} \frac{1}{w_{TD}}$$

Source areas are defined as homogenously emitting sources, having emission rate E. As a consequence, the C/E relationship is expressed in *s/m*. In analogy, the flux to source strength ratios ($\langle u'C' \rangle / E, \langle v'C' \rangle / E, \langle w'C' \rangle / E$) are calculated, mainly for completeness and for possible vertical flux footprint corrections. Trajectories touching ground are perfectly reflected and their velocities are reversed to maintain covariances past reflection. Therefore, no deposition is modelled when trajectory touchdown happens. Further, this model is not capable of modelling chemistry, and trace gases have to be assumed to be reasonably inert within the relevant travelling time. For further details on the model and the concentration to emission rate relationship, see *Flesch et al. (2004)*, *Flesch (1996)* and partly *Flesch et al. (1995)*.

MOST Profiles

The model calculation is based on Monin-Obukhov similarity theory (MOST) profiles of the wind speed and the wind statistics. As a consequence, the wind characteristics in the model domain can be set up completely by providing the three parameters $ustar$ (the friction velocity), L (the Obukhov length) and z_0 (the roughness length). It should be kept in mind that, if MOST can not be applied reasonably, model calculations certainly will be erroneous.

Wind Speed

The vertical profile of the ensemble average of the horizontal wind speed $U = \langle u \rangle$ is defined as

$$U(z) = ustar/k_v * \{ln(z/z_0) + \Psi(z/L) - \Psi(z_0/L)\}$$

where Ψ is a stability correction function defined as

$$if(L >= 0), \Psi(x) = 4.8 * z/L$$

$$if(L < 0), \Psi(x) = -2 * ln((1+\alpha)/2) - ln((1+\alpha^2)/2) - 2 * atan(\alpha) + \pi/2,$$

where

$$\alpha = (1 - 16 * z/L)^{1/4}$$

Wind Statistics

The variances of the different velocity components are assumed to be constant within the model domain (i.e. homogenous), with the exception of the vertical velocity component under unstable atmospheric conditions (i.e. if the Obukhov length is negative, $L < 0$), where the vertical profile function is defined as

$$\sigma w^2 = ustar^2 * bw^2 * (1 - 3 * z/L)^{2/3}$$

and *bw* is derived from the supplied $\sigma w/ustar$ ratio at given height.

Zero-Plane Displacement / Displacement Height (d)

In contrast to *Flesch et al. (2004)*, the zero-plane displacement (d) is included in the model profiles, by subtracting d from any measured, geometric height to give the corresponding aerodynamic heights $z = z_{meas} - d$.

Initiaization of velocities

Initial velocities are calculated by using an orthogonal projection of random ... (explain further)

Model Input

All model input is supplied from script/console. See `genInputList` for further information.

Touchdown Catalogs

Existing touchdown catalogs can significantly speed up model runs.

Tolerances

If touchdown catalogs will be read, the tolerances for touchdown selection can be set.

Model Output

csv File

The results of the model run can be saved to a csv file (semicolon separated).

data.frame/data.table

All results will be given as `data.frames` or `data.table`. See `runbLS` for further details.

Touchdown Catalogs

Touchdown catalogs can be saved for future runs.

Parallel Computing

This package provides the option to run some parts of the model calculation in parallel. The number of cores that will be used can be defined via the `ncores` argument in the model input (see `genModel` or `runbLS`). Parallel computing is done using the package `snow` (and parts of `snowfall` and `parallel`). If the model is run in parallel mode, two parts of the model will be processed parallel, a) the core C++ routine, calculating the trajectories and corresponding touchdowns, and b) the calculation of the model results (i.e. *C/E* etc.) from the touchdown catalogs.

To Do:

- Plots (Contour, etc)
- explain output csv
- Docu
- data.table

Author(s)

Christoph Haeni

References

Flesch, T. K., J. D. Wilson, et al. (2004). "Deducing ground-to-air emissions from observed trace gas concentrations: A field trial." Journal of Applied Meteorology 43(3): 487-502.

Flesch, T. K. (1996). "The footprint for flux measurements, from backward Lagrangian stochastic models." Boundary-Layer Meteorology 78(3-4): 399-404.

Flesch, T. K., J. D. Wilson, et al. (1995). "Backward-time Lagrangian stochastic dispersion models and their application to estimate gaseous emissions." Journal of Applied Meteorology 34(6): 1320-1332.

See Also

runbLS, genInputList, genSensors, genSources, genInterval, genModel, coreModel.

Examples

```
## Not run:
## set up some sensors and sources:
Sensors <- genSensors(
  PointSensor = list(x=0,y=0,z=2)
  ,LineSensor = list(x=c(-10,10),y=0,z=2,d=0.5)
  )
Source <- genSources(Source1=list('c',M=c(0,20),R=10))

## get default Interval data with optimized MaxFetch:
Int <- genInterval(MaxFetch=-20)

## generate TD catalog with temporary touchdown catalogs:
Model <- genModel(TDwrite=FALSE)
InList <- genInputList(Sensors,Source,Int,Model)
Run <- runbLS(InList,Cat.Path=getwd())

## look at catalog
TDs <- readCatalog(getCatalogs(Run,rn=1,Sensor='PointSensor')[,Catalog])
plot(TDs,pch=20)

## temporary TD catalogs will be deleted on exiting R or
## by running the funtion cleanTemporary()

## End(Not run)
```

Class-Sensors *S3 methods for class* Sensors

Description

S3 methods for class Sensors:

Usage

```
## S3 method for class 'Sensors'
print(x, Nrows = 30, ...)
## S3 method for class 'Sensors'
head(x, ...)
## S3 method for class 'Sensors'
tail(x, ...)

## S3 method for class 'Sensors'
plot(x, ...)

## S3 method for class 'Sensors'
join(...)
```

Arguments

x	a data.frame of class Sensors.
Nrows	number of maximum rows to print without changing to compact display.
...	further arguments to be passed to other methods. For join.Sensors see the details section below.

Details

For join.Sensors no further arguments other than different data.frames of class Sensor can be supplied.

Author(s)

Christoph Haeni

See Also

bLSmodelR-package, runbLS, genSensors

Class-Sources *S3 methods for class* Sources

Description

S3 methods for class Sources:

Usage

```
## S3 method for class 'Sources'
print(x, Nrows = 30, ...)
## S3 method for class 'Sources'
head(x, ...)
## S3 method for class 'Sources'
tail(x, ...)

## S3 method for class 'Sources'
plot(x, ...)

## S3 method for class 'Sources'
join(...)
```

Arguments

x	a data.frame of class Sources.
Nrows	number of maximum rows to print without changing to compact display.
...	further arguments to be passed to other methods. For join.Sources see the details section below.

Details

For join.Sources no further arguments other than different data.frames of class Sources can be supplied.

Author(s)

Christoph Haeni

See Also

bLSmodelR-package, runbLS, genSources

Class-TDcat *S3 methods for class* TDcat

Description

S3 methods for class TDcat:

Usage

```
## S3 method for class 'TDcat'
print(x, ...)

## S3 method for class 'TDcat'
plot(x, asp = 1, panel.first = {
    grid()
    abline(h = 0, col = "darkgrey")
    abline(v = 0, col = "darkgrey")
},...)
## S3 method for class 'TDcat'
points(x, ...)
```

Arguments

x	a data.table of class TDcat.
asp	y/x aspect ratio. see plot.window.
panel.first	plotting expression prior to the main plotting. see plot.default.
...	further arguments to be passed to other methods.

Author(s)

Christoph Haeni

See Also

bLSmodelR-package, runbLS, coreModel

coreModel *Calculate trajectories in backward mode.*

Description

This is the core function to calculate the (air parcel) trajectories in a backward mode (i.e. backward-in-time).

Usage

```
coreModel(u, v, w, zSens, ustar, L, Zo, bw, sigmaUustar, sigmaVustar,
  kv, C0, alpha, MaxFetch, method = "Euler")
```

Arguments

u	numeric vector. Instantaneous velocity of the individual air parcels at time 0 in alongwind direction (given in m/s).
v	instantaneous velocity of air parcel at time 0 in crosswind direction (given in m/s).
w	instantaneous velocity of air parcel at time 0 in vertical direction (given in m/s).
zSens	sensor height in m above ground level.
ustar	friction velocity in m/s.
L	Obukhov length in m.
Zo	roughness length in m.
bw	parameter defining the vertical profile of the variance in the vertical velocity component sigmaW.
sigmaUustar	variance of the alongwind velocity devided by ustar.
sigmaVustar	variance of the crosswind velocity devided by ustar.
kv	von Ka'rma'n constant (usually = 0.4).
C0	Kolmogorov constant.
alpha	Fraction of the velocity decorrelation time scale as given in Flesch et al., 2004 to choose the time increment.
MaxFetch	maximum tracking distance of trajectories (in m).
method	the method of numerical discretization to be used. The default "Euler" corresponds to *Flesch et al. (1995)*. "RK4" corresponds to the classical (4th order) Runge-Kutta method.

Value

A list with following items:

Traj_IDOut	ID of the corresponding trajectory. This ID is identical to the indexing position of the initial velocities u, v and w (see calculation of w'C'/E in example below).
TimeOut	Time until touchdown in seconds.
xOut	x position of touchdown.
yOut	y position of touchdown.
wTDOut	Touchdown velocity.

Author(s)

Christoph Haeni

References

Flesch, T. K., J. D. Wilson, et al. (1995). "Backward-time Lagrangian stochastic dispersion models and their application to estimate gaseous emissions." Journal of Applied Meteorology 34(6): 1320-1332.

Flesch, T. K., J. D. Wilson, et al. (2004). "Deducing ground-to-air emissions from observed trace gas concentrations: A field trial." Journal of Applied Meteorology 43(3): 487-502.

See Also

runbLS, bLSmodelR-package.

Examples

```
## Not run:

## set up model parameters
zSigmaWu <- 2
zSens <- 2
Ustar <- 0.25
L <- -600
Zo <- 0.001
SigmaUu <- 4.5
SigmaVu <- 4
SigmaWu <- 1.7
kv <- 0.4
alpha <- 0.02
MaxFetch <- 500
A <- 0.5

bw <- calcbw(SigmaWu, zSigmaWu/L)
SigmaWm <- calcsigmaW(Ustar, zSens/L, bw)
U <- calcU(Ustar, Zo, L, zSens, kv)
C0 <- calcC0(bw, kv)

## initial velocities
N0 <- 1000
set.seed(1234)
TDcat <- initializeCatalog(N0, Ustar, SigmaUu, SigmaVu, bw,
  zSens, L, Zo, A, alpha, MaxFetch, kv)

uvw <- uvw0(TDcat)

## run trajectory calculation
TDList <- coreModel(uvw[,"u0"], uvw[,"v0"], uvw[,"w0"], zSens, Ustar,
  L, Zo, bw, SigmaUu, SigmaVu, kv, C0, alpha, MaxFetch)

## assign touchdowns to initalized Catalog:
assignCat(TDcat,TDList)

## Calculate C/E and w'C'/E ratios of entire domain with homogeneous
## emission rate:
Ci <- data.table(ID=1:N0,CE=0,u=uvw[,"u0"],v=uvw[,"v0"],w=uvw[,"w0"],key="ID")
Ci[TDcat[,sum(2/wTD),by=Traj_ID],CE:=V1]

stats <- Ci[,cbind(
  round(rbind(ce <- mean(CE),se <- sd(CE)/sqrt(N0),ce - 1.96*se,ce + 1.96*se),1),
  round(rbind(wce <- mean(w*(cs <- CE-ce)),wse <- sd(w*cs)/sqrt(N0)
  ,wce - 1.96*wse,wce + 1.96*wse),2))]

dimnames(stats) <- list(c("E(X)","SE","Lower-CI95%","Upper-CI95%"),c("C/E","w'C'/E"))
print(stats)
```

```
## look at Catalog:
TDcat
plot(TDcat,pch=20)


## End(Not run)
```

---

deposition *Dry Deposition Post-Processing*

Description

Run the 'bLSmodelR' dry deposition post-processing function.

Usage

```
deposition(x, vDep, rn = NULL, Sensor = NULL, Source = NULL, vDepSpatial = NULL, ncores = 1)
```

Arguments

x	A data.frame (or data.table) of class bLSresult output by runbLS containing the results of a model run.
vDep	either a numeric vector providing the model surface deposition velocity (at height d + z0) for the rows specified by argument index or a character string specifying the column that contains the vDep values.
rn	integer. Subsetting by column 'rn'. Only matching rows will be post-processed.
Sensor	character. Subsetting by column 'Sensor'. Only matching rows will be post-processed.
Source	character. Subsetting by column 'Source'. Only matching rows will be post-processed.
vDepSpatial	a list with two entries (this argument is still experimental!). First list entry is a named list or named vector that defines area names with corresponding deposition velocities. Second list entry is an object of class Sources (see genSources) that defines the geometry of those areas (see example below). Note: Areas should not overlap each other!
ncores	integer. Number of cores used for parallel processing.

Value

A data.frame (or data.table) of class bLSresult and deposition containing the results of the model run and the deposition post-processing. The following attribute is additionally attached to the existing bLSresult attributes:

vDep vector of surface deposition velocities

Author(s)

Christoph Haeni

See Also

bLSmodelR-package, genInputList, coreModel.

Examples

```
## Not run:
# Example Run with default values:
Sensor <- genSensors(
  PointSensor1=list(x=0,y=0,z=2)
  ,PointSensor2=list(x=10,y=0,z=2)
  )
Sources <- genSources("Circle 1" = list("c", M = c(0,50), R = 10))
Ints <- genInterval(MaxFetch=70)
InputList <- genInputList(Sensor,Ints,Sources)

## run model
Run <- runbLS(InputList,Cat.Path=getwd())

## dry deposition post-processing:
RunDep <- deposition(Run,vDep=0.035,Sensor='PointSensor1')

## spatially inhomogeneous deposition velocity (still experimental!)
DepAreas <- genSources(
"Circle 2" = list("c", M = c(0,30), R = 10)
,"Circle 3" = list("c", M = c(0,10), R = 10)
)
DepVel <- list("Circle 2" = 0.005,"Circle 3" = 0.01)
plot(Sensor,join(Sources,DepAreas),SourceTextArgs=list(labels=paste0('vDep = ',c(0,0.005,0.01))))
RunDep2 <- deposition(Run,vDep=0.035,Sensor='PointSensor1',vDepSpatial=list(DepVel,DepAreas))



## End(Not run)
```

---

genInputList *Model Input*

Description

Create a `list` gathering the required model input.

Usage

```
genInputList(..., Tol.Zero = FALSE)
```

Arguments

`...`	optional arguments containing pre-defined model input:
	`Class-Interval` a `data.frame` as generated by `genInterval` containing the Interval input.
	`Class-Sensors` a `data.frame` as generated by `genSensors` containing the Sensors input.

Class-Sources a data.frame as generated by genSources containing the Sources input.

Class-Model a list as generated by genModel containing the Model parameter input.

Class-Tolerances a data.frame as generated by genTolerances containing the Tolerances input.

Tol.Zero logical. If TRUE, all touchdown catalog selection tolerances will be set to zero.

Value

a list containing all necessary model input.

Author(s)

Christoph Haeni

See Also

genInterval, genSensors, genSources, genModel, genTolerances, runbLS, bLSmodelR-package.

Examples

```
## Not run:
## generate source areas:
Sources <- genSources("Circle 1" = list("c", M = c(0,0), R = 10),
  "Circle 2" = list("c", M = c(0,30), R = 20, N = 50))

## generate sensors:
Sensors <- genSensors(LineSensor1=list(x=-30,y=c(-10,50),z=1.25,n=40)
  ,LineSensor2=list(x=30,y=c(-10,50),z=c(1.05,1.25),d=0.5))

## look at site map:
siteMap(Sensors,Sources,PolyArgs=list(col="darkgreen")
  ,PtArgs=list(pch=3,cex=0.8,lwd=2),SourceTextArgs=list(cex=1,col="navyblue")
  ,SensorTextArgs=list(cex=0.7,pos=3),xlim=c(-60,60))
addScaleBar()
addWindrose(WD = 270)

# generate interval data:
Int <- genInterval(WD=270,N0=1E4,SensorNames="LineSensor2",SourceNames="Circle 1,Circle 2")

## generate List of model input:
Model <- genModel(TDwrite=FALSE)
InputList <- genInputList(Int, Sources, Sensors, Model)

## run bLSmodelR (takes 2 to 5 mins):
Run <- runbLS(InputList,Cat.Path=getwd())

# switch names to "informative" and extract specific results (Source column is sorted accordingly)
switchNames(Run,simple=FALSE)
extractResult(Run,list(Source=c("Circle 2","Circle 1")),keep=c("Sensor","Source","C/E","C/E SE"))

## clean up TD catalogs
cleanTemporary()

## End(Not run)
```

genInterval *Interval Data*

Description

Generate a data.frame providing data on an interval basis (i.e. processed Sonic-Anemometer data etc.).

Usage

```
genInterval(Data = NULL, Ustar = 0.25, L = -2000, Zo = 0.01, sUu = 2.5, sVu = 2,
  sWu = 1.25, z_sWu = 2, WD = 0, d = 0, N0=5E4, MaxFetch=500, SensorNames = "",
  SourceNames = "")
```

Arguments

Data	a data.frame. Columns that match argument names below will be taken as input to the correspondent argument. Columns that don't match will be appended as additional data.
Ustar	friction velocity in m/s.
L	Obukhov-Length in meters.
Zo	roughness length in meters.
sUu	scaled standard deviation of the wind speed component **u** (i.e. alongwind): sigmaU/ustar.
sVu	scaled standard deviation of the wind speed component **v** (i.e. crosswind): sigmaV/ustar.
sWu	scaled standard deviation of the vertical wind speed component **w**: sigmaW/ustar.
z_sWu	measurement height of the wind statistics sigmaW/ustar (sWu) in meters. Only relevant for unstable conditions (L<0), where sigmaW/ustar is assumed to vary with height. Usually, this is the Sonic-Anemometer measurement height above ground.
WD	average wind direction in degrees deviating from North. I.e. wind blowing from 0/360 = north, 90 = east, 180 = south, 270 = west.
d	displacement height in meters.
N0	integer. Number of trajectories in model runs.
MaxFetch	numeric. Maximum upwind fetch (alongwind tracking distance). Negative numbers will try to optimize this value (see details below).
SensorNames	Names of the Sensors to calculate TDs, separated by ','.
SourceNames	Names of the Sources to include, separated by ','.

Details

Providing arguments directly will overwrite corresponding input from argument Data (see examples below).

Providing a negative MaxFetch argument will calculate the maximum distance between the sensor and the source and add the absolute value of the provided MaxFetch argument to this distance (see example below).

Value

a data.frame of class Sonic.

Author(s)

Christoph Haeni

See Also

genInputList, genSensors, genSources, genModel, genTolerances, runbLS, bLSmodelR-package.

Examples

```
## Not run:
## default:
print(x <- genInterval())
class(x)

## providing Data argument:
dat <- data.frame(Ustar=0.2,L=c(-30,-300),add1='test',ustar=0.1)
genInterval(dat)

## partially overwriting Data argument:
genInterval(dat,Ustar=0.3)

## optimizing MaxFetch
Sensor <- genSensors(PS=list(x=0,y=0,z=2))
Source <- genSources(
  Src = list('c',M=c(0,50),R=20)
  ,Src = list(x = c(25,35,35,25), y = c(20,20,50,50))
  )
Ints <- genInterval(WD=c(0,22.5,45,67.5),MaxFetch=-20)
InList <- genInputList(Ints,Source,Sensor)

IntExt <- prepareIntervals(InList,getwd())
IntExt[,MaxFetch]

## checking graphically:
plot(InList)
IntExt[,{
  x0 <- sin(WD/180*pi)
  y0 <- cos(WD/180*pi)
  x <- x0*(MaxFetch - 20)
  y <- y0*(MaxFetch - 20)
  for(i in 1:4){
    lines(c(0,x[i]),c(0,y[i]),lty=i)
    lines(x[i]+c(-100,100)*y0[i],y[i]-c(-100,100)*x0[i],lwd=2,col='lightgrey',lty=i)
  }}]


## End(Not run)
```

genModel *Model Parameters*

Description

Generate a `data.frame` defining (global) model specific data.

Usage

```
genModel(kv = 0.4, A = 0.5, alpha = 0.02, wTDcutoff = 1E-4, TDwrite = TRUE,
  overwriteTD = TRUE, TDread = TRUE, TDonly = FALSE, ncores = 1)
```

Arguments

`kv`	von-Karman constant.
`A`	scaling constant. (see *Flesch et al. (2004)*)
`alpha`	time step fraction of the velocity decorrelation time scale. note: alpha > 10^-3
`wTDcutoff`	cutoff velocity used as lower bound of calculated touchdown velocities.
`TDwrite`	logical. Should touchdown catalogs be saved?
`overwriteTD`	logical. Should existing touchdowns catalogs be overwritten?
`TDread`	logical. Check for existing touchdown catalogs?
`TDonly`	logical. Skip calculation of C/E etc., only simulate touchdown catalogs? Default is `FALSE`.
`ncores`	integer. Number of cores used for parallel processing.

Value

a `list` with the above defined model specifications. These model parameters are used 'globally', i.e. for all intervals, in the specified model run.

Author(s)

Christoph Haeni

References

Flesch, T. K., J. D. Wilson, et al. (2004). "Deducing ground-to-air emissions from observed trace gas concentrations: A field trial." Journal of Applied Meteorology 43(3): 487-502.

See Also

`genInputList`, `genInterval`, `genSensors`, `genSources`, `genTolerances`, `runbLS`, `bLSmodelR-package`.

Examples

```
## Not run:
## defaults:
genModel()

## End(Not run)
```

genSensors *Sensor Geometry*

Description

Generate a data.frame providing sensor geometry data.

Usage

```
genSensors(...)
```

Arguments

...	a data.frame and/or individual arguments defining point and/or line sensors. Names of arguments will be taken as sensor names. (see examples below)

Details

see examples below on the usage of genSensors...

Value

a data.frame of class Sensors

Author(s)

Christoph Haeni

See Also

Class-Sensors, genInputList, genInterval, genSources, genModel, genTolerances, runbLS, bLSmodelR-package.

Examples

```
## Not run:
## generate a point and two line sensors (way1):
Sensors1 <- genSensors(PointSensor=list(x=0,y=0,z=1.5)
  ,LineSensor1=list(x=c(-10,10),y=0,z=1.25,n=40)
  ,LineSensor2=list(x=0,y=c(-20,10),z=c(1.05,1.25),d=0.5))
Sensors1

## generate a point and two line sensors (way2):
SensorDF2 <- data.frame(c("PointSensor2","LS1b_1","LS1b_2","LS2b_1","LS2b_2"),c(0,-10,10,0,0)
  ,c(0,0,0,-20,10),c(1.5,1.25,1.25,1.05,1.25),c("","LineSensor1b","LineSensor1b","LineSensor2b",
  "LineSensor2b"),c(1,40,0,NA,NA),c(0,0,0,0.5,0))
Sensors2 <- genSensors(SensorDF2)
Sensors2

## join Sensors data frames:
Sensors4 <- join(Sensors1,Sensors2)
Sensors4

siteMap(Sensors4,PtArgs=list(pch=4,cex=0.8,lwd=2,col="darkred")
```

```
  ,SensorTextArgs=list(cex=0.7,pos=3),LSArgs = list(col="navyblue"))
addScaleBar()


## End(Not run)
```

---

genSources *Source Geometry*

Description

Generate a data.frame providing source geometry data.

Usage

```
genSources(...)
```

Arguments

...	a list, a data.frame and/or individual arguments defining source areas. Names of list entries and/or arguments will be taken as source names. (see examples below)

Details

see examples below on the usage of genSources...

Value

a data.frame of class Sources

Author(s)

Christoph Haeni

See Also

Class-Sources, genInputList, genInterval, genSensors, genModel, genTolerances, runbLS, bLSmodelR-package.

Examples

```
## Not run:
## generate 2 circular source areas:
Sources1 <- genSources(list(
  "Circle1" = list("c", M = c(0,0), R = 10),
  "Circle2" = list("c", M = c(0,30), R = 20, N = 50))
  )
head(Sources1)

## generate a rectangular and a polygon source area:
Sources2 <- genSources(
  RectSource = list("r", x1 = -10, x2 = 10, y1 = 50, y2 = 70),
  Polygon = list(x = c(-40, -30, -20, -30), y = c(30, 20, 30, 40))
```

```
  )
Sources2

# join Sources:
Sources <- join(Sources1,Sources2)
head(Sources)
## look at site map:
siteMap(Sources,PolyArgs=list(col="darkgreen"),SourceTextArgs=list(cex=1,col="navyblue"))
addScaleBar()


## generate a source consisting of 3 circular polygon areas:
SouList <- lapply(list(c(-20,5),c(-20,-5),c(-11.3,0)),function(x)list("c",M=x,R=1,N=50))
names(SouList) <- rep("ArtSource",3)
Sources3 <- genSources(SouList)
head(Sources3)

siteMap(Sources3,PolyArgs=list(col="darkgreen")
,SourceTextArgs=list(cex=1,col="navyblue"),xlim=c(-5,15),ylim=c(-10,10))
addScaleBar()


## generate a rectangular source area from a data frame:
SouDF <- data.frame("Rect",c(-10,-10,10,10),c(50,70,70,50),1)
Sources4 <- genSources(SouDF)
Sources4

plot(join(Sources,Sources3,Sources4))


## End(Not run)
```

---

genTolerances *Calculation Tolerances*

Description

Generate a `data frame` defining calculation tolerances.

Usage

```
genTolerances(Tol.Z = 2.5, Tol.L = 10, Tol.Zo = 5, Tol.sUu = 10, Tol.sVu = 10,
  Tol.sWu = 5, Tol.Zero = FALSE)
```

Arguments

`Tol.Z`	Tolerance (in %) of sensor height when checking existing touchdown catalogs.
`Tol.L`	Tolerance (in %) of Obukhov-Length when checking existing touchdown catalogs.
`Tol.Zo`	Tolerance (in %) of roughness height when checking existing touchdown catalogs.

Tol.sUu	Tolerance (in %) of sigmaU/ustar when checking existing touchdown catalogs.
Tol.sVu	Tolerance (in %) of sigmaV/ustar when checking existing touchdown catalogs.
Tol.sWu	Tolerance (in %) of sigmaW/ustar (at sensor height) when checking existing touchdown catalogs.
Tol.Zero	logical. If TRUE, all touchdown catalog selection tolerances will be set to zero.

Author(s)

Christoph Haeni

See Also

genInputList, genInterval, genSensors, genSources, genModel, runbLS, bLSmodelR-package.

Examples

```
## Not run:
## default:
genTolerances()

## Zero tolerances:
genTolerances(Tol.Zero=TRUE)

## End(Not run)
```

ModelFunctions *Model functions*

Description

Functions to calculate some desciptive parameters outside of the model.

Usage

```
calcU(ustar, Zo, L, z, kv = 0.4)
calcepsilon(ustar, L, bw, z, kv = 0.4)
calcsigmaW(ustar, zL, bw)
calcC0(bw, kv = 0.4, A = 0.5)
calcTL(sigmaW, C0, epsilon)
calcbw(sigmaWustar, zL)
```

Arguments

ustar	friction velocity in m/s.
Zo	roughness length in meters.
L	Obukhov-Length in meters.
z	(aerodynamic) height in meters.
zL	z/L.
bw	scaled standard deviation of the vertical wind speed component **w** at z = 0.
kv	von-Karman constant.

A	scaling constant. (see *Flesch et al. (2004)*)
sigmaW	standard deviation of the vertical wind speed component **w**.
C0	Kolmogorov's (universal) constant
epsilon	dissipation rate of turbulent kinetic energy in m^2/s^3.
sigmaWustar	scaled standard deviation of the vertical wind speed component **w**: sigmaW/ustar.

Author(s)

Christoph Haeni

References

Flesch, T. K., J. D. Wilson, et al. (2004). "Deducing ground-to-air emissions from observed trace gas concentrations: A field trial." Journal of Applied Meteorology 43(3): 487-502.

See Also

example of coreModel, bLSmodelR-package.

---

runbLS *Run bLSmodelR*

Description

Run 'bLSmodelR' using the specified model input.

Usage

```
runbLS(ModelInput, Cat.Path = NULL, TDonly = NULL, ncores = NULL,
  writeCsv = FALSE, asDT = TRUE, simpleNames = asDT)
```

Arguments

ModelInput	a list of data.frames defining the model input as generated by genInputList.
Cat.Path	optional. Character string specifying the path to the touchdown catalogs folder.
TDonly	logical. Skip calculation of C/E etc., only simulate touchdown catalogs? Default is FALSE.
ncores	integer. Number of cores used for parallel processing.
writeCsv	either a logical value or a character string specifying the path to a text file where the results will be stored as semicolon seperated values. If TRUE, the output file is selected via a tcltk GUI. If FALSE (default), no output file will be written.
asDT	logical. If TRUE, the Output will be a data.table. The default FALSE returns a data.frame.
simpleNames	logical. Use simple column names for Output, if TRUE. Default FALSE provides more descriptive column names.

Value

A `data.frame` (or `data.table`) of class `bLSresult` containing the results of the model run. The following `attributes` are additionally attached:

CalcSteps a `data.table` containing the individual model calculation intervals. Keys: `rn,Sensor`

CatPath path to catalog folder

Catalogs a `data.table` containing the catalog names. Keys: `rn,Sensor,Cat.Name`

ModelInput the ModelInput as supplied.

Author(s)

Christoph Haeni

References

Flesch, T. K., J. D. Wilson, et al. (1995). "Backward-time Lagrangian stochastic dispersion models and their application to estimate gaseous emissions." Journal of Applied Meteorology 34(6): 1320-1332.

See Also

`bLSmodelR-package`, `genInputList`, `coreModel`.

Examples

```
## Not run:
# Example Run with default values:
Sensor <- genSensors(PointSensor=list(x=0,y=0,z=2))
Ints <- genInterval(N0=1E4)
InputList <- genInputList(Sensor,Ints)

## run model
Run <- runbLS(InputList,Cat.Path=getwd())

## get catalog
TDcat <- readCatalog(getCatalogs(Run))

# look at TD catalog:
TDcat
plot(TDcat)


## End(Not run)
```

`Undocumented Functions` *undocumented functions*

Description

List of undocumented bLSmodelR functions:

Details

- `%w/o%`: x without y, analogous to `%in%`
- `addScaleBar`: add a scale bar to the current plot
- `addWindrose`: add a wind rose (arrow showing wind direction) to the current plot
- `assignCat`: assign `coreModel` output to an empty, initialized TD catalog
- `CalcSteps`: Get attribute CalcSteps from bLS output
- `Catalogs`: Get attribute Catalogs from bLS output
- `CatPath`: Get attribute CatPath from bLS output
- `check.compact`: check TD catalog for compact attributes
- `cleanTemporary`: remove temporary TD catalogs
- `compactCatalog`: make TD catalog compact for saving (not reversible due to rounding!)
- `ConcPalette`: Colour palette
- `extractResult`: extract results from model output
- `FluxPalette`: Colour palette
- `getArea`: get area of a polygon
- `getCatalogs`: get TD catalog pathnames from model output
- `initializeCatalog`: initialize TD catalog
- `join`: see `join.Sensors` and `join.Sources`
- `join.bLSresult`: merge bLS results consistently
- `join.deposition`: merge deposition results consistently
- `ModelInput`: Get attribute ModelInput from bLS output
- `prepareIntervals`: link existing and upcoming TD catalogs to intervals and possibly optimize MaxFetch
- `plot.InputList`: S3method calling `siteMap`
- `plotFootprint`: plot C-/u'C'-/w'C'-Footprint
- `readCatalog`: read TD catalog
- `rebuildCatListFile`: rebuild file with TD catalog list
- `rotate`: rotate coordinates around a fix point
- `rotateCatalog`: rotate Catalog (into WD)
- `round125`: round digits to nearest predefined numbers
- `siteMap`: plot site map
- `sortData`: sort data, partially adopted from `setorderv`
- `tagInside`: tag TDs inside source area
- `updatePath`: update path to TD catalogs
- `uvw0`: get initial (t=0) velocities from TD catalog
- `writeCatalog`: write TD catalog (default: saved as compact TD catalog)
- `switchNames`: change column names (simple/descriptive) of model output
- `setDT`: change model output to a `data.table`, see `setDT`
- `setDF`: change model output to a `data.frame`, see `setDF`

Note

some of these functions will be documented or dropped in future package versions.

Author(s)

Christoph Haeni

See Also

`bLSmodelR-package`, `runbLS`

Index