Package: foreSIGHT (via r-universe)

May 27, 2026

Version 2.0.0

Depends R (>= 3.5.0)

LinkingTo Rcpp

Imports ggplot2 (>= 3.3.0), GA (>= 3.0.2), Rcpp, rlang, directlabels, cowplot, stats, graphics, grDevices, utils, jsonlite, progress, scales, viridisLite, fields, lattice, mvtnorm, Matrix, SoilHyP, dfoptim, RGN, foreach, BLRPM, doParallel, dplyr, lubridate, tidyr, methods, zoo, airGR

Suggests knitr (>= 1.8), rmarkdown (>= 1.18), testthat (>= 3.0.0),

Title Systems Insights from Generation of Hydroclimatic Timeseries

BugReports https://github.com/ClimateAnalytics/foreSIGHT/issues

Description A tool to create hydroclimate scenarios, stress test systems and visualize system performance in scenario-neutral climate change impact assessments. Scenario-neutral approaches 'stress-test' the performance of a modelled system by applying a wide range of plausible hydroclimate conditions (see Brown & Wilby (2012) <doi:10.1029/2012EO410001> and Prudhomme et al. (2010) <doi:10.1016/j.jhydrol.2010.06.043>). These approaches allow the identification of hydroclimatic variables that affect the vulnerability of a system to hydroclimate variation and change. This tool enables the generation of perturbed time series using a range of approaches including simple scaling of observed time series (e.g. Culley et al. (2016) <doi:10.1002/2015WR018253>) and stochastic simulation of perturbed time series via an inverse approach (see Guo et al. (2018) <doi:10.1016/j.jhydrol.2016.03.025>). It incorporates 'Richardson-type' weather generator model configurations documented in Richardson (1981) <doi:10.1029/WR017i001p00182>, Richardson and Wright (1984), as well as latent variable type model configurations documented in Bennett et al. (2018) <doi:10.1016/j.jhydrol.2016.12.043>, Rasmussen (2013) <doi:10.1002/wrcr.20164>, Bennett et al. (2019) <doi:10.5194/hess-23-4783-2019> to generate hydroclimate variables on a daily basis (e.g. precipitation, temperature,

potential evapotranspiration) and allows a variety of different hydroclimate variable properties, herein called attributes, to be perturbed. Options are included for the easy integration of existing system models both internally in R and externally for seamless 'stress-testing'. A suite of visualization options for the results of a scenario-neutral analysis (e.g. plotting performance spaces and overlaying climate projection information) are also included. Version 1.0 of this package is described in Bennett et al. (2021) <doi:10.1016/j.envsoft.2021.104999>. As further developments in scenario-neutral approaches occur the tool will be updated to incorporate these advances.

License GPL-3

Encoding UTF-8

NeedsCompilation yes

VignetteBuilder knitr

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.2

Config/testthat/edition 3

Collate 'GR4J_funcs.R' 'LogfileEntry.R' 'RcppExports.R' 'default_parameters.R' 'SWG_BLRPM.R' 'SWG_LV.R' 'SWG_WGENotherVars.R' 'SWG_WGENprecip.R' 'SWG_distScaling.R' 'SWG_monAR1.R' 'argumentInputChecker.R' 'attributeCalculator.R' 'attributeManager.R' 'calc.stat.lib.R' 'calibrationArgCheck.R' 'calibrationNation.R' 'checkSimParBounds.R' 'combine_sims.R' 'control.R' 'controlFileFunctions.R' 'control_utilityFuncs.R' 'createExpSpace.R' 'dataInputChecker.R' 'dateManager.R' 'defaultParams_diagAndPlotting.R' 'demoTankModel.R' 'diagnosticPlotLib.R' 'eval_system_metrics.R' 'evaluate_attribute_change.R' 'exposureSpaceSampler.R' 'foreSIGHT.R' 'getSimSummary.R' 'harmonicFit.R' 'modelInputEnvironments.R' 'modelSequencer.R' 'objFuncTown.R' 'optimManagement.R' 'plotExpSpace.R' 'plotOptions.R' 'plotPerformanceOAT.R' 'plotPerformanceSpace.R' 'plotPerformanceSpaceMulti.R' 'postProcessing.R' 'runSystemModel.R' 'simCity.R' 'simClimateFromPar.R' 'simPerformance.R' 'simStochasticMultiSite.R' 'simplyScale_lib.R' 'stochParManager.R' 'summariseMutiSiteSimulatedSeries.R' 'summariseSimulatedSeries.R' 'targetFinder.R' 'zzz.R'

Author Bree Bennett [aut] (ORCID: <https://orcid.org/0000-0002-2131-088X>), David McInerney [aut, cre] (ORCID: <https://orcid.org/0000-0003-4876-8281>), Sam Culley [aut] (ORCID: <https://orcid.org/0000-0003-4798-8522>),

Anjana Devanand [aut] (ORCID: <https://orcid.org/0000-0001-9422-3894>), Seth Westra [aut] (ORCID: <https://orcid.org/0000-0003-4023-6061>), Danlu Guo [ctb] (ORCID: <https://orcid.org/0000-0003-1083-1214>), Holger Maier [ths] (ORCID: <https://orcid.org/0000-0002-0277-6887>)

Maintainer David McInerney <david.mcinerney@adelaide.edu.au>

Config/pak/sysreqs libicu-dev

Repository https://cranhaven.r-universe.dev

Date/Publication 2026-05-27 05:02:00 UTC

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/foreSIGHT

RemoteSha ef0f0205ccab92931821266f3d9bc645e5b0a178

RemoteSubdir foreSIGHT

Contents

barossa_obs — *Multi-site rainfall observations in the Barossa Valley used in examples and vignette*

Description

Dataset of observed rainfall for multiple sites in the Barossa Valley based on SILO point data

Format

A list of observed rainfall data with elements *times* and *P*. *P* is a matrix with rows corresponding to dates, and columns corresponding to 13 sites in the Barossa Valley

Source

SILO point rainfall data obtained from https://www.longpaddock.qld.gov.au. Data obtained for stations *23300, 23302, 23305, 23309, 23312, 23313, 23317, 23318, 23321, 23363, 23373, 23752, 23756* for the period 1 Jan 1972 to 31 December 1999.

boxplot_prob — *Draws a boxplot with the whiskers at specified probability limits*

Description

`boxplot_prob` draws a boxplot with the whiskers at probability limits `whiskersProb`.

Usage

```
boxplot_prob(xin, whiskersProb = c(0.025, 0.975), at = NULL, ...)
```

Arguments

`xin`	a vector, matrix or dataframe; data to be plotted
`whiskersProb`	a vector of length 2; min and max probability limits
`at`	a vector; specifying x coordinates for boxes
`...`	other arguments for `bxp`

Value

The function returns a boxplot. See example in help for `evaluate_system_metrics()`

`calculateAttributes` *Calculates the attributes of the hydroclimate time series*

Description

`calculateAttributes` calculates the specified attributes of the input daily hydroclimate time series.

Usage

```
calculateAttributes(climateData, attSel, startYr = NULL, endYr = NULL)
```

Arguments

`climateData`	list; reference climate data, with vector *times* in POSIXct format, and climate variables **variable_name1**, **variable_name2**. Climate variables are specified as vectors for single-site data, and matrices for multi-site data (with columns for each site). Please refer to data provided with the package that may be loaded using `data(tankDat)` and `data(barossaDat)` for examples of the expected format of single site and multi-site `reference`.
`attSel`	a vector; specifying the names of the attributes to be calculated.
`startYr`	a number (default `NULL`); to specify the starting year to subset `climateData` if required. If `NULL`, `startYr` is starting year in the input `climateData`.
`endYr`	a number (default `NULL`); to specify the ending year to subset `climateData` if required. If `NULL`, `endYr` is last year in the input `climateData`.

Value

The function returns a vector of attributes with names of the attributes (`attSel`). For multi-site data, names are combinations of attribute and site names.

Examples

```
#-----------------------------------------------------------------------
# Example 1: Single-site  input
# load 'tank' example climate data available in the package
data("tankDat")
# specify rainfall and temperature attributes to calculate
attSel <- c(
  "P_day_all_tot_m", "P_day_all_nWet_m", "P_day_all_R10_m",
  "Temp_day_all_rng_m", "Temp_day_all_avg_m"
)
tank_obs_atts <- calculateAttributes(tank_obs, attSel = attSel)
#-----------------------------------------------------------------------
# Example 2: Multi-site  input
# load 'Barossa' example climate data available in the package
```

```
data("barossaDat")
# specify rainfall attributes to calculate
attSel <- c("P_day_all_tot_m", "P_day_all_nWet_m", "P_day_all_P99")
barossa_obs_atts <- calculateAttributes(barossa_obs, attSel = attSel)
```

`calGR4J` *Calibrate GR4J rainfall runoff model parameters*

Description

`calGR4J` calibrates the GR4J model using the airGR package. The NSE is used as the objective function.

Usage

```
calGR4J(dates, P, PET, Qobs, plotResults = F)
```

Arguments

`dates`	is a vector of daily dates
`P`	is a vector of daily precipitation data (in mm)
`PET`	is a vector of daily potential evapotranspiration data (in mm)
`Qobs`	is the observed daily streamflow (in mm)
`plotResults`	is a logical indicating whether airGR summary plots are to be produced

Value

A vector with GR4J parameter values

Examples

```
# load dates, precip, PET and streamflow data for Scott Creek
data('data_A5030502')
clim_ref = list(times = data_A5030502$times,P = data_A5030502$P)
data("egScottCreekSimStoch")
# observed flow
Qobs = data_A5030502$Qobs
# observed PET
PET = data_A5030502$PET
dates = as.Date(clim_ref$times)
# calibrate GR4J parameters
Param = calGR4J(dates = dates,P=clim_ref$P,PET=PET,Qobs=Qobs)
#' @import airGR
```

convert_climYMD_POSIXct

Converts "old" reference climate data format (<V1.2) to "new" format with POSIXct dates.

Description

`convert_climYMD_POSIXct` produces reference climate data list for use in V2.0 and newer.

Usage

```
convert_climYMD_POSIXct(clim)
```

Arguments

`clim`	data.frame or list; contains reference daily climate data from foreSIGHT V1.2 or earlier.

Value

The function returns a list containing `times` in POSIXct format (corresponding to year, month and day from original data `clim`), and climate variables.

create_clim

Create foreSIGHT reference climate object from time information and climate data.

Description

`create_clim` produces reference climate data list.

Usage

```
create_clim(timeStart, timeEnd, timeStep, fmt = NULL, tz = "UTC", ...)
```

Arguments

`timeStart`	a character; the first time for the climate data
`timeEnd`	a character; the last time
`timeStep`	a character; the time step, containing one of "hour", "day", "week", "month" or "year"
`fmt`	a character; format of `timeStart` and `timeEnd`
`tz`	a character; the timezone. Default is 'UTC', which avoids issues with daylight savings.
`...`	vectors or matrices; climate data objects which will be added to the output list.

Value

The function returns a list containing `times` in POSIXct format and climate data.

Examples

```
# create small reference climate list (only 10 days)
clim <- create_clim(
  timeStart = "2007/01/01", # start date
  timeEnd = "2007/01/10", # end date
  timeStep = "day", # time step
  fmt = "%Y/%m/%d", # format of start/end dates
  P = c(0, 0, 0, 0, 0, 4.5, 2.6, 0, 0, 0), # precip data
  Temp = c(25, 24, 30, 32, 33, 27, 21, 21, 22, 30) # temperature data
)
```

---

createExpSpace *Creates exposure space of hydroclimatic targets for generation of scenarios using 'generateScenarios'*

Description

`createExpSpace` returns a list containing the targets (`targetMat`) and the metadata (input arguments) used to create the exposure space.

Usage

```
createExpSpace(
  attPerturb,
  attPerturbSamp = NULL,
  attPerturbMin,
  attPerturbMax,
  attPerturbType = "regGrid",
  attPerturbBy = NULL,
  attHold = NULL,
  attTied = NULL,
  targetTypes = NULL,
  attTargetsFile = NULL
)
```

Arguments

`attPerturb`	A char vector; the names of the attributes to be perturbed. This vector can contain attributes of different hydroclimatic variables.
`attPerturbSamp`	An integer vector; the number of samples for each attribute `attPerturb`. The length of this vector should be equal to the length of `attPerturb`.

`attPerturbMin` A numeric vector; the minimum bounds for sampling of `attPerturb`. The length of this vector should be equal to the length of `attPerturb`. For variables like precipitation, evapotranspiration, radiation, etc. `attPerturbMin` should specified as a fraction of the original (eg: 0.9 = 90% of the original attribute). For temperature, `attPerturbMin` should be specified in K (eg: 0.9 = 0.9 K).

`attPerturbMax` A numeric vector; the maximum bounds for sampling of `attPerturb`. The length of this vector should be equal to the length of `attPerturb`. For variables like precipitation, evapotranspiration, radiation, etc. `attPerturbMax` should specified as a fraction of the original (eg: 0.9 = 90% of the original attribute). For temperature, `attPerturbMax` should be specified in K (eg: 0.9 = 0.9 K). Note that to create a single sample of the attribute, `attPerturbSamp` could be specified as 1 with `attPerturbMin` and `attPerturbMax` specified as equal.

`attPerturbType` A string to specify the type of sampling, defaults to regular spacing. Valid sampling types are:

- "regGrid" a regular grid sampling all the attributes specified in `attPerturb` simultaneously
- "OAT" one-at-a-time sampling of the attributes specified in `attPerturb`

`attPerturbBy` A numeric vector; increment of values to create samples between `attPerturbMin` and `attPerturbMax`. If `attPerturbBy` is specified, attPerturbSamp should be set as `NULL`.

`attHold` A char vector; the names of the attributes to be held at historical levels. This vector can contain attributes of different hydroclimatic variables.

`attTied` A list; the attributes to be tied to other perturbed or held attributes. First level of list is name of perturbed/held attributes. Second level of list is vector of attributes that are tied (change with) to attribute in first level.

`targetTypes` A list; target type ('frac','diff') used to calculate changes in attributes. First level of list is name of attribute. Second level of list is target type ('frac','diff'). Specifying `targetTypes` overrides default target types ('diff' for temperature attributes, 'frac' for other)

`attTargetsFile` String specifying the full path to a CSV file containing the target exposure space. The column names in the file should correspond to the attributes specified in `attPerturb` and `attHold`. `attTargetsFile` is alternate way to specify exposure space targets that do not form a regular grid. If `attTargetsFile` is specified, the inputs arguments `attPerturbSamp`, `attPerturbMin`, `attPerturbMax`, and `attPerturbType` should be set to `NULL` and will not be used by the function.

Details

See "Detailed Tutorial: Climate 'Stress-Testing' using *fore*SIGHT" vignette for specifying attribute names for `attPerturb` and `attHold`. The definition of the attribute can be viewed using the function `viewAttributeDef`.

Value

The exposure space as a list containing the following fields:

- targetMat a dataframe or matrix; each column is a perturb/hold attribute, each row is a point in the exposure space.
- attRot a char vector containing the one-at-a-time ("OAT") attributes associated with targetMat, attRot is NULL for other types of sampling.
- attPerturb, attHold, attPerturbSamp, attPerturbMin, attPerturbMax, attPerturbType in the function input arguments, if not NULL.

See Also

generateScenarios, viewAttributeDef

Examples

```
# To view the definition of any valid attribute
viewAttributeDef("P_day_all_tot_m")

# To create an exposure space of points on a regular grid
attPerturb <- c("P_day_all_tot_m", "P_day_all_nWet_m", "P_day_all_R10_m")
attPerturbType <- "regGrid"
attPerturbSamp <- c(3, 1, 1)
attPerturbMin <- c(0.9, 1, 1)
attPerturbMax <- c(1.1, 1, 1)
attHold <- c(
  "P_day_Feb_tot_m", "P_day_SON_dyWet_m", "P_day_JJA_avgWSD_m",
  "P_day_MAM_tot_m", "P_day_DJF_avgDSD_m", "Temp_day_all_rng_m", "Temp_day_all_avg_m"
)
expSpace <- createExpSpace(
  attPerturb = attPerturb, attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin, attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType, attHold = attHold, attTargetsFile = NULL
)

# Using attPerturbBy to specify the increment of perturbation (attPerturbSamp set to NULL)

attPerturb <- c("P_day_all_tot_m", "P_day_all_nWet_m", "P_day_all_R10_m")
attPerturbType <- "regGrid"
attPerturbMin <- c(0.9, 1, 1)
attPerturbMax <- c(1.1, 1, 1)
attPerturbBy <- c(0.1, 0, 0)
attHold <- c(
  "P_day_Feb_tot_m", "P_day_SON_dyWet_m", "P_day_JJA_avgWSD_m", "P_day_MAM_tot_m",
  "P_day_DJF_avgDSD_m", "Temp_day_all_rng_m", "Temp_day_all_avg_m"
)
expSpace <- createExpSpace(
  attPerturb = attPerturb, attPerturbSamp = NULL,
 attPerturbMin = attPerturbMin, attPerturbMax = attPerturbMax, attPerturbType = attPerturbType,
  attPerturbBy = attPerturbBy, attHold = attHold, attTargetsFile = NULL
)

# To create an exposure space of observed attributes without perturbation
# Note that attPerturbMin and attPerturbMax values are set to 1 for variables like precipitation,
# and 0 for temperature
```

```
attPerturb <- c("P_day_all_tot_m", "P_day_all_nWet_m", "P_day_all_R10_m", "Temp_day_DJF_avg_m")
attPerturbType <- "regGrid"
attPerturbSamp <- c(1, 1, 1, 1)
attPerturbMin <- c(1, 1, 1, 0)
attPerturbMax <- c(1, 1, 1, 0)
expSpace <- createExpSpace(
  attPerturb = attPerturb, attPerturbSamp = attPerturbSamp,
 attPerturbMin = attPerturbMin, attPerturbMax = attPerturbMax, attPerturbType = attPerturbType,
  attHold = NULL, attTargetsFile = NULL
)

# Example showing tied attributes that tie changes in seasonal attributes
# to changes in annual attributes
attPerturb <- c("P_day_all_P99")
attHold <- c("P_day_all_tot_m","P_day_all_avgDSD_m", "P_day_all_nWet_m")
attPerturbType = "regGrid"
attPerturbSamp = c(5)
attPerturbMin = c(0.9)
attPerturbMax = c(1.3)
# use following function to create seasonally tied attribute list
attTied = setSeasonalTiedAttributes(attSel=c(attPerturb,attHold))
# view attTied
attTied
expSpace <- createExpSpace(attPerturb = attPerturb,
                           attPerturbSamp = attPerturbSamp,
                           attPerturbMin = attPerturbMin,
                           attPerturbMax = attPerturbMax,
                           attPerturbType = attPerturbType,
                           attHold = attHold,
                           attTied = attTied)
```

`data_A5030502` *Catchment data for Scott Creek in South Australia for period 1976-1985.*

Description

Catchment data for Scott Creek in South Australia for period 1976-1985.

Usage

```
data_A5030502
```

Format

A list with 4 elements

times Vector of times in POSIXct format

P Vector of precipitation data (mm)

PET Vector of PET data (mm) (seasonally variable, no changes annually)

Qobs Vector of observed streamflow (mm)

`egClimData` *Climate attributes from projections.*

Description

A example dataset containing the climate attribute values in fraction/additive change

Usage

```
egClimData
```

Format

A data frame with 15 rows and 12 variables:

P_day_all_tot_m change in mean annual total P, fraction

P_day_all_seasRatioMarAug change in seasonal (Mar-Aug) ratio of P, fraction

P_day_all_P99 change in 99th percentile of P, fraction

Temp_day_all_avg change in averageTemp, additive

Name name of the climate model

Avg. Deficit performance metric values

`egMultiSiteSim` *Output from call to generateScenarios() using multi-site model (see example 5 in generateScenarios).*

Description

Output from call to generateScenarios() using multi-site model (see example 5 in generateScenarios).

Usage

```
egMultiSiteSim
```

Format

A list with 4 elements

Rep1 List containing majority of simulation output, including output for different calibration stages

simDates the dates of the simulation

expSpace the exposure space of the simulation

controlFile the setting in the control file

egScalPerformance *Performance metrics of the tank model using simple scaled scenarios.*

Description

Performance metrics of the tank model using simple scaled scenarios.

Usage

```
egScalPerformance
```

Format

A list with 2 elements

volumetric reliability (fraction) Volumetric reliaiblity of tank

reliability (fraction) Reliability of tank

egScalSummary *Summary of a simple scaled scenario.*

Description

Summary generated using the function `getSimSummary`.

Usage

```
egScalSummary
```

Format

A list containing 3 elements

simDates the dates of the simulation

expSpace the exposure space of the simulation

controlFile "scaling"

egSimOATPerformance *Performance metrics of the tank model using OAT scenarios.*

Description

Performance metrics of the tank model using OAT scenarios.

Usage

```
egSimOATPerformance
```

Format

A list with 2 elements

Avg. Deficit average daily deficit of water, litres

Reliability reliability of the tank, fraction

egSimOATSummary *Summary of a OAT scenario.*

Description

Summary generated using the function `getSimSummary` for a scenarios generated using stochastic models for an OAT exposure space

Usage

```
egSimOATSummary
```

Format

A list containing 13 elements

egSimPerformance — *Performance metrics of the tank model using regGrid scenarios.*

Description

Performance metrics of the tank model using regGrid scenarios.

Usage

```
egSimPerformance
```

Format

A list with 2 elements

Avg. Deficit average daily deficit of water, litres

Reliability reliability of the tank, fraction

egSimPerformanceB — *Performance metrics of an alternate tank model using regGrid scenarios.*

Description

Performance metrics of an alternate tank model using regGrid scenarios.

Usage

```
egSimPerformanceB
```

Format

A list with 2 elements

Avg. Deficit average daily deficit of water, litres

Reliability reliability of the tank, fraction

egSimSummary *Summary of a regGrid scenario.*

Description

Summary generated using the function `getSimSummary` for a scenarios generated using stochastic models for a regGrid exposure space

Usage

```
egSimSummary
```

Format

A list containing 13 elements

evaluate_system_metrics *Calculates system metrics for and observed and baseline stochastic climates*

Description

`evaluate_system_metrics` runs observed climate and baseline (unperturbed) stochastic climates through a system model and calculates system metrics for each. This is used to perform evaluation of stochastic climates using the 'virtual observation' approach. See example in Section 5.3 of *'Stress-Testing' using *fore*SIGHT: Stochastic simulation* vignette.

Usage

```
evaluate_system_metrics(
  sim,
  clim,
  systemModel,
  systemArgs,
  metrics,
  varNames = NULL
)
```

Arguments

`sim`	list; a simulation containing the scenarios generated using the function `generateScenarios`.
`clim`	a list; reference climate

systemModel	a function; The function runs the system model using climate data in a list as input. The function is expected to be created by the user for specific system models.
systemArgs	a list; containing the input arguments to systemModel.
metrics	a string vector; the names of the performance metrics the systemModel function returns.
varNames	a string vector; containing the names of the climate variables that are extracted from sim and used in system model. If NULL, then varNames determined from attribute names in sim$expSpace.

Value

a list containing systemPerf_base and systemPerf_obsClim, with performance metrics for baseline stochastic climate and observed climate, respectively.

---

func_avg *Calculates average of time series*

Description

Calculates average of time series

Usage

```
func_avg(data)
```

Arguments

data	is a vector, representing a time series

---

func_avgDSD *Calculates average dry spell duration (below threshold)*

Description

Calculates average dry spell duration (below threshold)

Usage

```
func_avgDSD(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_avgDwellTime *Calculates the average dwell time, i.e. average time for below median value spells*

Description

Calculates the average dwell time, i.e. average time for below median value spells

Usage

```
func_avgDwellTime(data)
```

Arguments

data	is a vector, representing a time series

func_avgWSD *Calculates average wet spell duration (below threshold)*

Description

Calculates average wet spell duration (below threshold)

Usage

```
func_avgWSD(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_cor *Calculates the lag-1 autocorrelation*

Description

Calculates the lag-1 autocorrelation

Usage

```
func_cor(data)
```

Arguments

data	is a vector, representing a time series

func_cv *Calculates the coefficient of variation (mead/sd)*

Description

Calculates the coefficient of variation (mead/sd)

Usage

```
func_cv(data)
```

Arguments

data	is a vector, representing a time series

func_dyWet *Calculates average rainfall on wet days (above threshold)*

Description

Calculates average rainfall on wet days (above threshold)

Usage

```
func_dyWet(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_F0 *Calculates the number of frost days*

Description

Calculates the number of frost days

Usage

```
func_F0(data)
```

Arguments

data	is a vector, representing a time series

func_maxDSD *Calculates maximum dry spell duration (below threshold)*

Description

Calculates maximum dry spell duration (below threshold)

Usage

```
func_maxDSD(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_maxWSD *Calculates maximum wet spell duration (above threshold)*

Description

Calculates maximum wet spell duration (above threshold)

Usage

```
func_maxWSD(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_normP *Calculates normalised quantile (quantile divided by mean)*

Description

Calculates normalised quantile (quantile divided by mean)

Usage

```
func_normP(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$quant denoting the probability of the quantile

func_nWet *Calculates number of wet days (above threshold)*

Description

Calculates number of wet days (above threshold)

Usage

```
func_nWet(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_P *Calculates a quantile value*

Description

Calculates a quantile value

Usage

```
func_P(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$quant denoting the probability of the quantile

func_R *Calculates the number of days above a threshold (often used for temperature)*

Description

Calculates the number of days above a threshold (often used for temperature)

Usage

```
func_R(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$threshold denoting the threshold

func_rng *Calculates the inter-quantile range*

Description

Calculates the inter-quantile range

Usage

```
func_rng(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$lim denoting the probability limit width

func_seasRatio *Calculates seasonality ratio*

Description

Calculates seasonality ratio

Usage

```
func_seasRatio(data, attArgs)
```

Arguments

data	is a vector, representing a time series
attArgs	is a list, with attArgs$indexSeas corresponding to season of interest

func_tot *Calculates total of time series*

Description

Calculates total of time series

Usage

```
func_tot(data)
```

Arguments

data	is a vector, representing a time series

func_WDcor *Calculates the lag-1 autocorrelation for wet days*

Description

Calculates the lag-1 autocorrelation for wet days

Usage

```
func_WDcor(data)
```

Arguments

data	is a vector, representing a time series

`func_wettest6monPeakDay` *Calculates the day of year corresponding to the wettest 6 months*

Description

Calculates the day of year corresponding to the wettest 6 months

Usage

```
func_wettest6monPeakDay(data, attArgs = NULL)
```

Arguments

`data`	is a vector, representing a time series
`attArgs`	is a list, with attArgs$doy denoting the day of year for each value in the time series

`func_wettest6monSeasRatio` *Calculates the ratio of wet season to dry season rainfall, based on wettest6monPeakDay*

Description

Calculates the ratio of wet season to dry season rainfall, based on wettest6monPeakDay

Usage

```
func_wettest6monSeasRatio(data, attArgs = NULL)
```

Arguments

`data`	is a vector, representing a time series
`attArgs`	is a list, with attArgs$doy denoting the day of year for each value in the time series

`generateScenarios` *Produces time series of hydroclimatic variables for an exposure space.*

Description

`generateScenarios` produces time series of hydroclimatic variables using requested climate attributes that correspond to a target exposure space using a reference daily time series as an input.

Usage

```
generateScenarios(
  reference,
  expSpace,
  simLengthNyrs = NULL,
  numReplicates = 1,
  seedID = NULL,
  cores = 1,
  controlFile = NULL,
  targetTol = 0.05
)
```

Arguments

`reference`	list; reference climate data, with vector *times* in POSIXct format, and climate variables **variable_name1**, **variable_name2**. Climate variables are specified as vectors for single-site data, and matrices for multi-site data (with columns for each site). Please refer to data provided with the package that may be loaded using `data(tankDat)` and `data(barossaDat)` for examples of the expected format of single site and multi-site `reference`.
`expSpace`	a list; created using the function `createExpSpace`
`simLengthNyrs`	a number; a scalar that specifies the length in years of each generated scenario. This argument is used only with stochastic generation. If `NULL` (the default), the generated simulation will be as long as `reference`.
`numReplicates`	a number; a scalar that specific the number of stochastic replicates to be generated. The default is 1.
`seedID`	a number; a scalar that specifies the seed to be used for the first replicate. Subsequent replicates will use seeds incremented by one. If `seedID` is `NULL` (which is the default), the function will use a random seed for stochastic time series generation. The seed used will be specified in the output. This argument is intended for use in cases that aim to reproduce an existing simulation.
`cores`	a number; Number of core sued for parallel processing.
`controlFile`	a string; to specify the model/optimisation options used for simulating time series data. The valid values are: • `NULL`: the simulation uses the foreSIGHT default stochastic model settings.

- "scaling": the simulation uses scaling (simple/seasonal) instead of a stochastic model. If all attributes in *expSpace* are annual totals/averages, then simple scaling is used. If seasonality ratio attributes are also included in *expSpace*, then seasonal scaling is used.
- path to a JSON file: the JSON file contains advanced options specify the stochastic model and optimisation inputs. These options can be used to change stochastic model types, overwrite default model parameter bounds, change default optimisation arguments, and set penalty attributes to be used in optimisation. Please refer to the function writeControlFile in order to create an controlFile JSON file.

targetTol — a number; Acceptable tolerance between target and simulated attributes. Error larger than targetTol for a single replicate/target produces a warning.

Value

The function returns a list containing the time series data generated. The list can contain multiple replicates (named as Rep1, Rep2 etc.) equal to the numReplicates function argument. Each replicate can contain multiple targets (named as Target1, Target2 etc.) based on the specified exposure space (expSpace). The expSpace and controlFile are also returned as part of this output list.

See Also

createExpSpace, writeControlFile, viewModels

Examples

```
# Example 1: Simple scaling
#------------------------------------------------------------------------
attPerturb <- c("P_day_all_tot", "Temp_day_all_avg")
attPerturbType <- "regGrid"
attPerturbSamp <- c(2, 2)
attPerturbMin <- c(0.8, -1)
attPerturbMax <- c(1.2, 1)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType
)
data(tankDat)
simScaling <- generateScenarios(
  reference = tank_obs,
  expSpace = expSpace,
  controlFile = "scaling"
)

# Example 2: Seasonal scaling
#------------------------------------------------------------------------
attPerturb <- c("P_day_all_tot", "P_day_all_seasRatio")
attPerturbType <- "regGrid"
```

```
attPerturbSamp <- c(2, 2)
attPerturbMin <- c(0.8, 0.9)
attPerturbMax <- c(1.1, 1.2)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType
)
data(tankDat)
seasScaling <- generateScenarios(
  reference = tank_obs,
  expSpace = expSpace,
  controlFile = "scaling"
)

# Example 3: Stochastic simulation using foreSIGHT default settings
#-----------------------------------------------------------------------
## Not run:
# create an exposure space
attPerturb <- c("P_day_all_tot_m", "P_day_all_nWet_m", "P_day_all_R10_m")
attHold <- c(
  "P_day_Feb_tot_m", "P_day_SON_dyWet_m", "P_day_JJA_avgWSD_m", "P_day_MAM_tot_m",
  "P_day_DJF_avgDSD_m", "Temp_day_all_rng_m", "Temp_day_all_avg_m"
)
attPerturbType <- "regGrid"
attPerturbSamp <- c(2, 1, 1)
attPerturbMin <- c(0.8, 1, 1)
attPerturbMax <- c(1.1, 1, 1)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType,
  attHold = attHold,
  attTargetsFile = NULL
)
# load example data available in foreSIGHT
data(tankDat)
# perform stochastic simulation
simStochastic <- generateScenarios(
  reference = tank_obs,
  expSpace = expSpace,
  simLengthNyrs = 30
)

## End(Not run)
# Example 4: Simple Scaling with multi-site data
#-----------------------------------------------------------------------
attPerturb <- c("P_day_all_tot_m", "P_day_all_seasRatio")
attPerturbType <- "regGrid"
```

```
attPerturbSamp <- c(3, 3)
attPerturbMin <- c(0.8, 1.2)
attPerturbMax <- c(0.8, 1.2)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType
)
# load multi-site rainfall data
data(barossaDat)
# perform simple scaling
simScaling <- generateScenarios(
  reference = barossa_obs,
  expSpace = expSpace,
  controlFile = "scaling"
)

# Example 5: Multi-site stochastic simulation
#-----------------------------------------------------------------------
## Not run:
attPerturb <- c("P_day_all_tot_m")
attHold <- c(
  "P_day_all_wettest6monSeasRatio", "P_day_all_wettest6monPeakDay",
  "P_day_all_P99", "P_day_all_avgWSD_m", "P_day_all_nWetT0.999_m"
)
attPerturbType <- "regGrid"
# consider unperturbed climates in this example
attPerturbSamp <- attPerturbMin <- attPerturbMax <- c(1)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType,
  attHold = attHold
)
# load multi-site rainfall data
data(barossaDat)
# specify the penalty settings in a list
controlFileList <- list()
controlFileList[["penaltyAttributes"]] <- c(
  "P_day_all_tot_m",
  "P_day_all_wettest6monSeasRatio", "P_day_all_wettest6monPeakDay"
)
controlFileList[["penaltyWeights"]] <- c(0.5, 0.5, 0.5)
# specify the alternate model selections
controlFileList[["modelType"]] <- list()
controlFileList[["modelType"]][["P"]] <- "latent"
# specify model parameter selection
controlFileList[["modelParameterVariation"]] <- list()
controlFileList[["modelParameterVariation"]][["P"]] <- "har"
```

```
# specify settings for multi-site model
controlFileList[["spatialOptions"]] <- list()
# specify spatial correlation perturbation factor
controlFileList[["spatialOptions"]][["spatCorFac"]] <- 0.9
# write control file sttings to file
controlFileJSON <- jsonlite::toJSON(controlFileList, pretty = TRUE, auto_unbox = TRUE)
write(controlFileJSON, file = paste0(tempdir(), "controlFile.json"))
# run multi-site stochastic simulation - this will take a long time (e.g. hours)
sim <- generateScenarios(
  reference = barossa_obs, expSpace = expSpace,
  controlFile = paste0(tempdir(), "controlFile.json"), seed = 1
)

## End(Not run)

# Example 6: Subdaily stochastic simulation
#-------------------------------------------------------------------------
## Not run:
# specify attributes for range of time scales from hourly to daily
attPerturb = c('P_day_all_tot')
attHold = c('P_hour_all_sd','P_hour_all_cor','P_hour_all_nWet',
            'P_3hour_all_sd','P_3hour_all_cor','P_3hour_all_nWet',
            'P_12hour_all_sd','P_12hour_all_cor','P_12hour_all_nWet',
            'P_day_all_sd','P_day_all_cor','P_day_all_nWet')
# consider unperturbed climate
attPerturbType = "regGrid"
attPerturbSamp = c(1)
attPerturbMin = c(1.)
attPerturbMax = c(1.)

# create the exposure space
expSpace = createExpSpace(attPerturb = attPerturb,
                          attPerturbSamp = attPerturbSamp,
                          attPerturbMin = attPerturbMin,
                          attPerturbMax = attPerturbMax,
                          attPerturbType = attPerturbType,
                          attHold = attHold)

# load synthetic subdaily data
data('subdailySyntheticDat')

controlFileList = list()
controlFileList$modelType = list()
controlFileList$modelType$P = "BLRPM"  # Bartlett-Lewis rectangular pulse model
controlFileList$modelParameterVariation = list()
controlFileList$modelParameterVariation$P = "ann" # annual parameters (don't vary with season)
controlFileList = jsonlite::toJSON(controlFileList, pretty = TRUE, auto_unbox = TRUE)
controlFile = paste0(tempdir(), "\\eg_controlFile.json")
write(controlFileList, file = controlFile)

sim = generateScenarios(reference = subdaily_synthetic_obs,
                        expSpace = expSpace,
                        controlFile = controlFile,
```

```
                          seedID = 1)
# plot biases in target and simulated attributes
plotScenarios(sim)

## End(Not run)
#------------------------------------------------------------------------
```

`getSimSummary` *Produces a summary object containing the metadata of a full simulation*

Description

`getSimSummary` uses a full simulation generated using the function `generateScenarios` as input and outputs a summary object containing the metadata of the full simulation. The output summary object may be used as an input to the plotting functions in this package. The output summary object will be much smaller in size than the full simulation for ease of storage and use with the plotting functions.

Usage

```
getSimSummary(sim)
```

Arguments

`sim`	list; a simulation containing the scenarios generated using the function `generateScenarios`.

See Also

`generateScenarios`, `plotPerformanceSpace`, `plotPerformanceOAT`

`GR4J_wrapper` *System model wrapper GR4J*

Description

`GR4J_wrapper` runs the GR4J model, using the airGR package, for a given set of climate inputs and parameter values and produces a set of runoff metrics

Usage

```
GR4J_wrapper(data, systemArgs, metrics)
```

Arguments

data	list; contains daily precipitation and PET to be used in GR4J, in a list with entries *times*, *P* and optionally *PET*.
systemArgs	list; contains Param which is a vector of GR4J parameters (obtained from calGR4J), dates which is a vector of dates, and PET which is a optional vector of potential transpiration (required if PET not included in data)
metrics	a vector of metric names (including 'meanQ' for mean daily flow, 'P99' and 'P25' for 99th and 25th percentile daily flows, and 'min3yr' for minimum 3-year total flow)

Value

A vector of metric values

Examples

```
# load dates, precip, PET and streamflow data for Scott Creek
data('data_A5030502')

clim_ref = list(times = data_A5030502$times,P = data_A5030502$P)

# observed flow
Qobs = data_A5030502$Qobs
# observed PET
PET = data_A5030502$PET

dates = as.Date(clim_ref$times)

# calibrate GR4J parameters
Param = calGR4J(dates = dates,P=clim_ref$P,PET=PET,Qobs=Qobs)

# setup systemArgs and metrics
systemArgs = list(dates=dates,Param=Param,PET=PET)
metrics = c('meanQ','P99','P25','min3yr')

metricsObs = GR4J_wrapper(data=clim_ref,systemArgs = systemArgs,metrics=metrics)
metricsObs
```

---

modCalibrator *modCalibrator*

Description

Calibrates weather generator models specified using modelTag.

Usage

```
modCalibrator(obs = NULL, modelTag = NULL, window = NULL)
```

Arguments

obs	A list of observed climate data
modelTag	A character vector of which stochastic models to use to create each climate variable. Supported tags are shown in under details below.
window	moving average window to calibrate daily gamma parameters for the modelTag "P-har-wgen".

Details

modelTag provides the main function with requested models. modelTag is vector of any of the following supported models:

- "P-ann-wgen" a four parameter annual rainfall model
- "P-seas-wgen" a 16 parameter seasonal rainfall model
- "P-har-wgen" a harmonic rainfall model
- "Temp-har-wgen0" a harmonic temperature model not conditional on rainfall
- "Temp-harWD-wgen0" a harmonic temperature model dependent on wet or dry day
- "Temp-har-wgen0" a harmonic temperature model
- "Temp-harWD-wgen0" a harmonic temperature model dependent on wet or dry day
- "PET-har-wgen" a harmonic potential evapotranspiration model
- "PET-harWD-wgen" a harmonic potential evapotranspiration model dependent on wet or dry day

Examples

```
data(tankDat) # Load tank data (tank_obs)
modelTag <- c("P-ann-wgen", "Temp-har-wgen0") # Select a rainfall and a temperature generator
out <- modCalibrator(
  obs = tank_obs, # Calibrate models
  modelTag = modelTag
)
```

---

modSimulator *modSimulator*

Description

Simulates using weather generator models specified using modelTag.

Usage

```
modSimulator(
  datStart,
  datFinish,
  modelTag = NULL,
  parS = NULL,
  seed = NULL,
  file = NULL,
  IOmode = "suppress"
)
```

Arguments

datStart	A date string in an accepted date format e.g. *"01-10-1990"*.
datFinish	A date string in an accepted date format e.g. *"01-10-1990"*. Must occur after datStart.
modelTag	A character vector of which stochastic models to use to create each climate variable. Supported tags are shown in details below.
parS	A list (names must match supplied modelTags) containing numeric vectors of model parameters.
seed	Numeric. Seed value supplied to weather generator.
file	Character. Specifies filename for simulation output.
IOmode	A string that specifies the input-output mode for the time series = "verbose", "dev" or "suppress".

Details

modelTag provides the main function with requested models. modelTag is vector of any of the following supported models:

- "P-ann-wgen" a four parameter annual rainfall model
- "P-seas-wgen" a 16 parameter seasonal rainfall model
- "P-har-wgen" a harmonic rainfall model
- "Temp-har-wgen0" a harmonic temperature model not conditional on rainfall
- "Temp-harWD-wgen0" a harmonic temperature model dependent on wet or dry day
- "Temp-har-wgen0" a harmonic temperature model
- "Temp-harWD-wgen0" a harmonic temperature model dependent on wet or dry day
- "PET-har-wgen" a harmonic potential evapotranspiration model
- "PET-harWD-wgen" a harmonic potential evapotranspiration model dependent on wet or dry day

Examples

```
## Not run:
data(tankDat)
obs <- tank_obs # Get observed data
modelTag <- c("P-har-wgen", "Temp-har-wgenO") # Select models
pars <- modCalibrator(obs = obs, modelTag = modelTag) # Calibrate models
sim <- modSimulator(
  datStart = "1970-01-01", # Simulate!
  datFinish = "1999-12-31",
  modelTag = modelTag,
  parS = pars,
  seed = 123,
  file = paste0("tester.csv"),
  IOmode = "verbose"
)
plot(sim$P[1:365]) # Plot first year of rainfall

## End(Not run)
```

`mvFunc_avgDryDay` *Calculates the average value of a non-rainfall time series on dry-days*

Description

Calculates the average value of a non-rainfall time series on dry-days

Usage

```
mvFunc_avgDryDay(data.1, data.2)
```

Arguments

`data.1`	represents a non-rainfall time series
`data.2`	represents rainfall time series

`mvFunc_avgWetDay` *Calculates the average value of a non-rainfall time series on wet-days*

Description

Calculates the average value of a non-rainfall time series on wet-days

Usage

```
mvFunc_avgWetDay(data.1, data.2)
```

Arguments

data.1	represents a non-rainfall time series
data.2	represents rainfall time series

---

mvFunc_cor *Calculates the correlation between two time series*

Description

Calculates the correlation between two time series

Usage

```
mvFunc_cor(data.1, data.2)
```

Arguments

data.1	a vector for the first climate variable
data.2	a vector for the second climate variable

---

mvFunc_cvDryDay *Calculates the coefficient of variation (sdev/mean) value of a non-rainfall time series on dry-days*

Description

Calculates the coefficient of variation (sdev/mean) value of a non-rainfall time series on dry-days

Usage

```
mvFunc_cvDryDay(data.1, data.2)
```

Arguments

data.1	represents a non-rainfall time series
data.2	represents rainfall time series

`mvFunc_cvWetDay` *Calculates the coefficient of variation (sdev/mean) value of a non-rainfall time series on wet-days*

Description

Calculates the coefficient of variation (sdev/mean) value of a non-rainfall time series on wet-days

Usage

```
mvFunc_cvWetDay(data.1, data.2)
```

Arguments

`data.1`	represents a non-rainfall time series
`data.2`	represents rainfall time series

`plotExpSpace` *Plots the location of points in a two-dimensional exposure space*

Description

The function uses an exposure space created using the function `createExpSpace` as input and creates a plot of the two dimensional (2D) exposure space. `plotExpSpace` plots only 2D spaces consisting of samples of 2 attributes.

Usage

```
plotExpSpace(
  expSpace,
  y = expSpace[["attPerturb"]][1],
  x = expSpace[["attPerturb"]][2]
)
```

Arguments

`expSpace`	list; an exposure space created using the function `createExpSpace`
`y`	a string; tag of a perturbed attribute to plot on the y-axis. Defaults to `expSpace[["attPerturb"]][1]`.
`x`	a string; tag of a perturbed attribute to plot on the x-axis. Defaults to `expSpace[["attPerturb"]][2]`.

Details

The number of dimensions of an exposure space is equal to the number of perturbed attributes in that space. If the exposure space has more than 2 dimensions (perturbed attributes), this function can be used to plot 2D slices of the space. Note that the default arguments of this function is defined to plot a slice showing the first two dimensions of the space, arguments `x` and `y` may be specified to plot alternate dimensions.

See Also

createExpSpace

Examples

```
# create an exposure space that has more than 2 dimensions
attPerturb <- c("P_day_all_tot_m", "P_day_all_nWet_m", "P_day_Feb_tot_m")
attHold <- c(
  "P_day_SON_dyWet_m", "P_day_JJA_avgWSD_m", "P_day_MAM_tot_m", "P_day_DJF_avgDSD_m",
  "Temp_day_all_rng_m", "Temp_day_all_avg_m"
)
attPerturbType <- "regGrid"
attPerturbSamp <- c(5, 5, 5)
attPerturbMin <- c(0.8, 0.9, 0.85)
attPerturbMax <- c(1, 1.1, 1.05)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType,
  attHold = attHold,
  attTargetsFile = NULL
)
# plot the first two dimensions
plotExpSpace(expSpace)
# plot another slice
plotExpSpace(expSpace, y = "P_day_all_tot_m", x = "P_day_Feb_tot_m")
```

---

plotOptions *Plots the differences in performance metrics from two system options*

Description

plotOptions uses the system model performances calculated using the function runSystemModel for two alternate system model options, and the summary of the simulation generated using the functions generateScenarios & getSimSummary as input. The function plots the differences in the performance metrics between the two options, and the changes in performance thresholds in the space. The user may specify the attributes to be used as the axes of the plot. The function contains arguments to control the finer details of the plot.

Usage

```
plotOptions(
  performanceOpt1,
  performanceOpt2,
  sim,
  metric = NULL,
  attX = NULL,
```

```
  attY = NULL,
  topReps = NULL,
  opt1Label = "Option 1",
  opt2Label = "Option 2",
  titleText = paste0(opt2Label, " - ", opt1Label),
  type = "filled.contour",
  nContour = perfSpace_nContour,
  perfThresh = NULL,
  perfThreshLabel = "Threshold",
  attSlices = NULL,
  climData = NULL,
  colMap = NULL,
  colLim = NULL
)
```

Arguments

performanceOpt1	a named list; contains the system model performance calculated using runSystemModel for system model option 1. If the list contains more than one performance metric, the argument metric can be used to specify the metric to be used.
performanceOpt2	a named list; contains the system model performance calculated using runSystemModel for system model option 2. If the list contains more than one performance metric, the argument metric can be used to specify the metric to be used.
sim	a list; summary of the simulation containing the scenarios generated using the function generateScenarios that is used to run the system model using runSystemModel. The summary of the simulation may be obtained by using the function getSimSummary on the full simulation. The summary object is much smaller in size for ease of storage and use with the performance plotting functions like plotPerformanceSpace.
metric	a string; the name of the performance metric to be plotted. The argument can be used to select the metric from performanceOpt1 and performanceOpt2 lists for plotting. If NULL (the default), the first metric in the lists will be used.
attX	a string; the tag of the perturbed attribute to plot on the xaxis. The attribute must be one of the perturbed attributes of sim. Type sim$expSpace$attPerturb to view all perturbed attributes of sim. If NULL (default), the first perturbed attribute of sim will be used.
attY	a string; the tag of the perturbed attribute to plot on the yaxis. The attribute must be another perturbed attribute of sim. If NULL, the second perturbed attribute of sim will be used.
topReps	an integer (default is NULL); the number of "top" replicates in terms of simulation fitness to be used. If topReps is specified, topReps number of replicates will be identified for each target and the average performance across these replicates will be plotted. If NULL, the average performance across all the replicates will be plotted.
opt1Label	a string; the text to label performanceOpt1.
opt2Label	a string; the text to label performanceOpt2.

titleText — a string; text for the title of the plot. The default is `paste0(opt2Label, " - ", opt1Label)`.

type — a string; indicates type of plot as "heat.plot" (default) or "filled.contour"

nContour — a number; specifies number of contours in the performance metric (if contourBreaks not specified)

perfThresh — a number; the minimum or maximum threshold value of the performance metric. A line will be drawn to mark this threshold value in the performance space.

perfThreshLabel — a string; the text to label `perfThresh`.

attSlices — a list; used to subset perturbed attributes in `sim` for the plot. This argument would typically be used in cases where there are more than two perturbed attributes. The elements of the list correspond to the perturbed attributes to be subsetted and must be named using the attribute tag. Each element may contain a single value or a two-element vector specifying the minimum-maximum values. If the element is a single value, the exposure space is sliced on this single value of the attribute. If minimum-maximum values are specified, the exposure space will be sliced to subset this range. If `attSlices` includes `attX` or `attY`, these attributes will be sliced and the resulting plot will be a "zoomed-in" space.

climData — data.frame; the values of attX and attY from other sources like climate models. This data will be plotted as points in the performance space. The data frame may contain columns with values of the performance metric to be plotted and the "Name" of the dataset. If the performance metric is available in the data.frame, the points will be coloured based on the performance `colMap` scale. If the `Name` of the data is available in the data.frame, the points will be identified using the `Name`. Please refer data provided with the package that may be loaded using `data("egClimData")` for an example of the expected format of `climData`.

colMap — a vector of colours; to specify the colourmap to be used. If `NULL`, the default foreSIGHT colourmap is used.

colLim — a vector of 2 values; the minimum and maximum limits of the colour scale.

Value

The plot of the differences in the performance metrics (option 2 - option 1) in a ggplot object.

See Also

`runSystemModel`, `plotPerformanceSpace`, `generateScenarios`, `getSimSummary`

Examples

```
# load example datasets
data("egSimSummary")
data("egSimPerformance") # performance of option1
data("egSimPerformanceB") # performance of option2
data("egClimData")
plotOptions(egSimPerformance[1], egSimPerformanceB[1], egSimSummary,
  attX = "P_day_all_seasRatioMarAug", attY = "P_day_all_tot_m", topReps = 7,
```

```
  perfThreshLabel = "Threshold (28L)",
  perfThresh = 28, opt1Label = "System A", opt2Label = "System B",
  climData = egClimData
)
```

`plotPerformanceAttributesOAT` *Plots changes in attributes for a specified perturbed attribute*

Description

`plotPerformanceAttributesOAT` plots OAT changes in attributes based on simulated climate object, for a given perturbed attribute.

Usage

```
plotPerformanceAttributesOAT(
  clim,
  sim,
  attPerturb,
  attEval,
  vSel = NULL,
  cSel = NULL,
  ylim = NULL,
  baseSettings = list(),
  cex.main = 0.8,
  cex.xaxis = 0.5,
  cex.yaxis = 0.5
)
```

Arguments

`clim`	list; reference climate data.
`sim`	list; perturbed climate from `generateScenarios()`
`attPerturb`	string; name of perturbed attribute
`attEval`	string; name of attribute that will be evaluated
`vSel`	string; variable name for selection site/aggregation
`cSel`	integer or string 'mean'; how to summarize multisite data - select site number or 'mean' for average

ylim	numeric vector of length 2; min and max y limits for plotting
baseSettings	list containing 'bias_base_thresh' for threshold for bias in baseline performance (and 'slope_thresh' for threshold for slope of relationship between perturbed and plotted attribute
cex.main	number; size for title
cex.xaxis	number; size for x-axis
cex.yaxis	number; size for y-axis

Value

The function returns a single plot showing changes in attribute attEval for changes in perturbed attribute attPerturb. Dashed blue lines indicate the target values for the perturbed attributes. Green lines represent the target values for held and tied attributes. Black line shows the median of the simulated attribute values, while the grey band indicates the 90 Red lines highlight attributes that change significantly more than the intended perturbation (i.e. the slope of attribute vs. perturbed attribute > slope_thresh). Red crosses mark attributes with large biases—greater than bias_base_thresh relative to their observed (unperturbed) historical values.

Examples

```
## Not run:
# load dates, precip, PET and streamflow data for Scott Creek
data('data_A5030502')
clim_ref = list(times = data_A5030502$times,P = data_A5030502$P)
data("egScottCreekSimStoch")
attSel = colnames(sim.stoch$expSpace$targetMat)
# other attributes
attSel = c(attSel,'P_year_all_avgDwellTime','P_day_all_avgWSD',
           'P_day_all_P99.9','P_day_JJA_P99.9','P_day_SON_P99.9',
           'P_day_DJF_P99.9','P_day_MAM_P99.9')
# plot changes in attributes for perturbations in seasonality ratio
att = "P_day_all_seasRatioMarMay"
par(mfrow=c(3,3),mar=c(4,7,2,1))
# plot changes in a single attribute with respect to perturbed attrbiutes
plotPerformanceAttributesOAT(clim=clim_ref,
                             sim=sim.stoch,
                             attPerturb=att,
                             attEval=attSel,
                             cex.main = 1.5,cex.xaxis = 1,cex.yaxis = 1)

## End(Not run)
```

plotPerformanceOAT *Plots performance for one-at-a-time (OAT) perturbations in attributes*

Description

plotPerformanceOAT uses the system model performance calculated using the function runSystemModel and the summary of the simulation generated using the function generateScenarios & getSimSummary as input. The function creates line plots, each panel shows the variations in performance with perturbations in a single attribute. The function is intended for use with simulations with attributes perturbed on a one-at-a-time (OAT) grid.

Usage

```
plotPerformanceOAT(
  performance,
  sim,
  metric = NULL,
  topReps = NULL,
  climData = NULL,
  col = NULL,
  ylim = NULL,
  noPlot = T,
  plim = c(0.05, 0.95),
  attSel = NULL,
  returnPlotData = F
)
```

Arguments

performance	a named list; contains the system model performance calculated using runSystemModel. If the list contains more than one performance metric, the first metric will be plotted.
sim	a list; a summary of a simulation containing the scenarios generated using the function generateScenarios that is used to run the system model using runSystemModel. The summary may be obtained using the function getSimSummary
metric	a string; the name of the performance metric to be plotted. The argument can be used to select a metric from performance for plotting.
topReps	an integer (default = NULL); the number of "top" replicates to be used. The "top" replicates will be identified for each target based on the simulation fitness. The average performance across topReps replicates will be plotted.
climData	data.frame; the values of attributes from other sources like climate models. This data will be plotted as "hairs" on the bottom of the plot.
col	a colour; the colour of the lines. If NULL, the a default colour is used.
ylim	a vector of 2 values; the minimum and maximum limits of the y-axis (performance) scale.

noPlot	a logical; whether or not to show plot (or just return ggplot object). noPlot=TRUE does not show plot.
plim	a vector of 2 values; probability limits for performance metric plots
attSel	a string vector; selected perturbed attribute to plot
returnPlotData	logical; for internal use only

Details

The plots show the mean value of performance across replicates. The ranges between the minimum and maximum values of performance across replicates are shaded. The function is intended for use with simulations containing attributes perturbed on an "OAT" grid. If the perturbations are on a "regGrid", this function will subset OAT perturbations, if available, to create the plots. The function creates separate plots for perturbations in attributes of temperature and other variables. The function may be called with performance argument specifying the metric to be plotted to plot other metrics.

Value

The plot of the performance space and the ggplot object.

See Also

runSystemModel, generateScenarios, plotPerformanceSpace, getSimSummary

Examples

```
# load example datasets
data("egSimSummary")
data("egSimPerformance")
plotPerformanceOAT(egSimPerformance[2], egSimSummary)
plotPerformanceOAT(egSimPerformance[1], egSimSummary)
# using the metric argument
plotPerformanceOAT(egSimPerformance, egSimSummary, metric = "reliability (fraction)")
```

---

plotPerformanceSpace *Plots a performance space using the system performance and scenarios as input*

Description

plotPerformanceSpace uses the system model performance calculated using the function runSystemModel and the summary of the simulation generated using the functions generateScenarios & getSimSummary as input to plot the performance space of the system. The user may specify the attributes to be used as the axes of the performance space.

Usage

```
plotPerformanceSpace(
  performance,
  sim,
  metric = NULL,
  attX = NULL,
  attY = NULL,
  topReps = NULL,
  perfThresh = NULL,
  perfThreshLabel = "Threshold",
  attSlices = NULL,
  climData = NULL,
  colMap = NULL,
  colLim = NULL,
  contourBreaks = NULL,
  nContour = perfSpace_nContour,
  axesPercentLabel = "fraction",
  type = "heat.plot",
  noPlot = F
)
```

Arguments

performance	a named list; contains the system model performance calculated using runSystemModel. If the list contains more than one performance metric, the argument metric can be used to specify the metric to be used.
sim	a list; summary of the simulation containing the scenarios generated using the function generateScenarios that is used to run the system model using runSystemModel. The summary of the simulation may be obtained by using the function getSimSummary on the full simulation. The summary object is much smaller in size for ease of storage and use with the performance plotting functions like plotPerformanceSpace.
metric	a string; the name of the performance metric to be plotted. The argument can be used to select a metric from performance for plotting. If NULL (the default), the first metric in the list will be used.
attX	a string; the tag of the perturbed attribute to plot on the xaxis. The attribute must be one of the perturbed attributes of sim. Type sim$expSpace$attPerturb to view all perturbed attributes of sim. If NULL (default), the first perturbed attribute of sim will be used.
attY	a string; the tag of the perturbed attribute to plot on the yaxis. The attribute must be another perturbed attribute of sim. If NULL, the second perturbed attribute of sim will be used.
topReps	an integer (default is NULL); the number of "top" replicates in terms of simulation fitness to be used. If topReps is specified, topReps number of replicates will be identified for each target and the average performance across these replicates will be plotted. If NULL, the average performance across all the replicates will be plotted.

perfThresh — a number; the minimum or maximum threshold value of the performance metric. A line will be drawn to mark this threshold value in the performance space.

perfThreshLabel — a string; the text to label perfThresh.

attSlices — a list; used to subset perturbed attributes in sim for the plot. This argument would typically be used in cases where there are more than two perturbed attributes. The elements of the list correspond to the perturbed attributes to be subsetted and must be named using the attribute tag. Each element may contain a single value or a two-element vector specifying the minimum-maximum values. If the element is a single value, the exposure space is sliced on this single value of the attribute. If minimum-maximum values are specified, the exposure space will be sliced to subset this range. If attSlices includes attX or attY, these attributes will be sliced and the resulting plot will be a "zoomed-in" space.

climData — data.frame; the values of attX and attY from other sources like climate models. This data will be plotted as points in the performance space. The data frame may contain columns with values of the performance metric to be plotted and the "Name" of the dataset. If the performance metric is available in the data.frame, the points will be coloured based on the performance colMap scale. If the Name of the data is available in the data.frame, the points will be identified using the Name. Please refer data provided with the package that may be loaded using data("egClimData") for an example of the expected format of climData.

colMap — a vector of colours; to specify the colourmap to be used. If NULL, the default foreSIGHT colourmap is used.

colLim — a vector of 2 values; the minimum and maximum limits of the colour scale.

contourBreaks — a vector; specifies breaks in the performance metric

nContour — a number; specifies number of contours in the performance metric (if contourBreaks not specified)

axesPercentLabel — a string; indicates display format for x and y axes. To display as a fraction, use "fraction", to display as a percentage change use "percentage.change", and to display as a percentage increase or decrease use "percentage.total".

type — a string; indicates type of plot as "heat.plot" (default) or "filled.contour"

noPlot — logical; indicates whether plots will be printed (TRUE) or not printed (FALSE) and only saved as an object.

Details

If the space contains more than two perturbed attributes, the performance values are averaged across the perturbations in the attributes other than attX and attY. The user may specify argument attSlices to slice the performance space at specific values of the other perturbed attributes. If attSlices are used to specify minimum-maximum values to subset other perturbed attributes, the performance values are averaged across the subsetted perturbations in these attributes. If the input performance list contains multiple performance metrics, the function plots the first metric. The function may be called with performance argument specifying the metric to be plotted plotPerformanceSpace(performance[2], sim) to plot other metrics.

Value

The plot of the performance space and the ggplot object.

See Also

runSystemModel, generateScenarios, getSimSummary, plotPerformanceOAT

Examples

```
## Not run:
# load example datasets
data("egSimSummary") # summary of stochastic simulation
data("egSimPerformance") # system performance calculated using the stochastic simulation
data("egClimData") # alternate climate data and system performance
colnames(egClimData)[6] = "average daily deficit (L)" # change metric name to match egSimPerformance

plotPerformanceSpace(performance = egSimPerformance[2], sim = egSimSummary)

# change plot style to "filled.contour" and specify contours - show contours from
# 0.76 to 0.9 in increments of 0.02
plotPerformanceSpace(
  performance = egSimPerformance[2],
  sim = egSimSummary, contourBreaks = seq(0.76, 0.84, 0.02),type='filled.contour'
)


# adding climate data, using top 10 replicates
plotPerformanceSpace(
  performance = egSimPerformance[1], sim = egSimSummary,
  topReps = 10, climData = egClimData
)

# adding a threshold
plotPerformanceSpace(
 performance = egSimPerformance, sim = egSimSummary, metric = "average daily deficit (L)",
 climData = egClimData, perfThresh = 27.5, perfThreshLabel = "Max Avg. Deficit",type = "heat.plot"
)

# user specified colMap
plotPerformanceSpace(
  performance = egSimPerformance[1], sim = egSimSummary,
  climData = egClimData, perfThresh = 27.5,
  perfThreshLabel = "Max Avg. Deficit",
  colMap = viridisLite::inferno(100)
)

# modify theme to change axes positioning to stacked vertically and left aligned
plotPerformanceSpace(
  performance = egSimPerformance[1], sim = egSimSummary,
  climData = egClimData, perfThresh = 27.5,
  perfThreshLabel = "Max Avg. Deficit",
  colMap = viridisLite::inferno(100)
```

```
) +
  ggplot2::theme(
    legend.box = "vertical",
    legend.position = "bottom",
    legend.box.just = "left",
    legend.margin = ggplot2::margin(t = 0.01, r = 0.1, b = 0.01, l = 0.1, "cm"),
    legend.justification = c(0.01, 0.01)
  )

# display fractional changes axes as percentage change
plotPerformanceSpace(
  performance = egSimPerformance, sim = egSimSummary,
  metric = "average daily deficit (L)",
  climData = egClimData, perfThresh = 27.5,
  perfThreshLabel = "Max Avg. Deficit",
  axesPercentLabel = "percentage.change"
)

# change displayed contours on performance space - show contours
# from 18 to 34 in increments of 2 L
plotPerformanceSpace(
  performance = egSimPerformance, sim = egSimSummary,
  metric = "average daily deficit (L)",
  climData = egClimData, perfThresh = 27.5,
  perfThreshLabel = "Max Avg. Deficit",
  contourBreaks = seq(18, 34, 2)
)

# change plot type to filled.contour style
plotPerformanceSpace(
  type = "filled.contour", performance = egSimPerformance,
  sim = egSimSummary, metric = "average daily deficit (L)",
  climData = egClimData, perfThresh = 27.5,
  perfThreshLabel = "Max Avg. Deficit",
  contourBreaks = seq(18, 34, 2)
)

# example overlay points manually from a dataset in a similar style to egClimData
ptStyle <- 21:25 # select set of pt styles (e.g. circle, square, triangle)
plotPerformanceSpace(performance = egSimPerformance[1],
                     sim = egSimSummary) +
  ggplot2::geom_point(
    data = egClimData,
    mapping = ggplot2::aes(
      x = .data[["P_day_all_tot_m"]],
      y = .data[["P_day_all_seasRatioMarAug"]],
      shape = .data[["Name"]]
    ),
    show.legend = TRUE, size = 5, colour = "black", fill = "lightgray"
  ) +
  ggplot2::scale_shape_manual(
    name = NULL, values = ptStyle,
    guide = ggplot2::guide_legend(order = 2, nrow = 1)
```

```
  ) +
  # one row of legend for specified ptStyle types
  ggplot2::theme(
    legend.box = "vertical", # vertical arrangement of items in legends
    legend.position = "bottom", # position legends base of figure
    legend.justification = c(0, 0)
  ) # justification according to the plot area

# example of performance generated using simple scaled simulation
data("egScalPerformance")
data("egScalSummary")
data("egClimData")
plotPerformanceSpace(
  performance = egScalPerformance[2], sim = egScalSummary,
  perfThresh = 0.8, perfThreshLabel = "Reliability threshold"
)

## End(Not run)
```

---

plotPerformanceSpaceMulti *Plots contours of the number of performance thresholds exceeded in the perturbation space*

Description

plotPerformanceSpaceMulti uses multiple system model performances calculated using the function runSystemModel and the summary of the simulation generated using the functions generateScenarios & getSimSummary as input to plot filled contours showing the number of performance thresholds exceeded in the perturbation space. The user may specify the attributes to be used as the axes of the perturbation space.

Usage

```
plotPerformanceSpaceMulti(
  performance,
  sim,
  perfThreshMin,
  perfThreshMax,
  attX = NULL,
  attY = NULL,
  attSlices = NULL,
  topReps = NULL,
  climData = NULL,
  col = NULL,
  axesPercentLabel = FALSE
)
```

Arguments

performance	a list; each element of the list should be a performance metric. May be calculated using the function runSystemModel
sim	a list; summary of the simulation containing the scenarios generated using the function generateScenarios that is used to run the system model using runSystemModel. The summary of the simulation may be obtained by using the function getSimSummary on the full simulation. The summary object is much smaller in size for ease of storage and use with the performance plotting functions like plotPerformanceSpace.
perfThreshMin	a vector; the minimum threshold value of each performance metric. The length of the vector should be equal to length(performance). If the metric does not have a minimum threshold, specify the corresponding element in perfThreshMin as NA.
perfThreshMax	a vector; the maximum threshold value of each performance metric. The length of the vector should be equal to length(performance). If the metric does not have a maximum threshold, specify the corresponding element in perfThreshMax as NA.
attX	a string; the tag of the perturbed attribute to plot on the xaxis. The attribute must be one of the perturbed attributes of sim. Type sim$expSpace$attPerturb to view all perturbed attributes of sim. If NULL (default), the first perturbed attribute of sim will be used.
attY	a string; the tag of the perturbed attribute to plot on the yaxis. The attribute must be another perturbed attribute of sim. If NULL, the second perturbed attribute of sim will be used.
attSlices	a list; used to subset perturbed attributes in sim for the plot. This argument would typically be used in cases where there are more than two perturbed attributes. The elements of the list correspond to the perturbed attributes to be subsetted and must be named using the attribute tag. Each element may contain a single value or a two-element vector specifying the minimum-maximum values. If the element is a single value, the exposure space is sliced on this single value of the attribute. If minimum-maximum values are specified, the exposure space will be sliced to subset this range. If attSlices includes attX or attY, these attributes will be sliced and the resulting plot will be a "zoomed-in" space.
topReps	an integer (default is NULL); the number of "top" replicates in terms of simulation fitness to be used. If topReps is specified, topReps number of replicates will be identified for each target and the average performance across these replicates will be plotted. If NULL, the average performance across all the replicates will be plotted.
climData	data.frame; the values of attX and attY from other sources like climate models. This data will be plotted as points in the perturbation space. If the Name of the data is available in the data.frame, the points will be identified using the Name. Please refer data provided with the package that may be loaded using data("egClimData") for an example of the expected format of climData.
col	a vector of colours; The length of the vector should at least be sufficient to assign unique colours to all the different values in the generated plot. If NULL, the default foreSIGHT colours is used.

axesPercentLabel
: a logical flag; if TRUE x and y axes to be displayed in terms of percentage change instead of fraction

Details

If the space contains more than two perturbed attributes, the performance values are averaged across the perturbations in the attributes other than attX and attY. The user may specify argument attSlices to slice the performance space at specific values of the other perturbed attributes. If attSlices are used to specify minimum-maximum values to subset other perturbed attributes, the performance values are averaged across the subsetted perturbations in these attributes. This function cannot be used with sim perturbed on an "OAT" grid since contours of the number of performance thresholds exceeded cannot be calculated for an irregular perturbation space.

Value

The plot showing the number of thresholds exceeded and the ggplot object.

See Also

runSystemModel, generateScenarios, getSimSummary, plotPerformanceSpace

Examples

```
# load example datasets
data("egSimPerformance")
data("egSimSummary")
data("egClimData")

plotPerformanceSpaceMulti(
  performance = egSimPerformance, sim = egSimSummary,
  perfThreshMin = c(NA, 0.80), perfThreshMax = c(30, NA)
)

# replot with axes as percentage changes
plotPerformanceSpaceMulti(
  performance = egSimPerformance, sim = egSimSummary,
  perfThreshMin = c(NA, 0.80), perfThreshMax = c(30, NA), axesPercentLabel = TRUE
)

# add alternate climate data and specify different colours for the plot
plotPerformanceSpaceMulti(
  performance = egSimPerformance, sim = egSimSummary,
  perfThreshMin = c(NA, 0.80), perfThreshMax = c(30, NA),
  climData = egClimData, col = viridisLite::magma(3)
)

# example using simple scaled simulations
data("egScalPerformance")
data("egScalSummary")
data("egClimData")
plotPerformanceSpaceMulti(
```

```
  performance = egScalPerformance, sim = egScalSummary,
  perfThreshMin = c(NA, 0.80), perfThreshMax = c(30, NA),
  climData = egClimData
)

# replot with axes as percentage changes (Note: modifies fractional change attributes only)
plotPerformanceSpaceMulti(
  performance = egScalPerformance, sim = egScalSummary,
  perfThreshMin = c(NA, 0.80), perfThreshMax = c(30, NA),
  climData = egClimData, axesPercentLabel = TRUE
)
```

`plotScenarios` *Creates summary plots of the biases in the scenarios*

Description

`plotScenarios` uses a simulation performed using the function `generateScenarios` as input and creates heatmaps that show the biases in the simulated attributes with respect to the specified target values of the attributes. The plots show the magnitude (absolute value) of the mean biases, and the standard deviation of biases across replicates. The heatmaps can be used to evaluate how well the simulated attributes match the specified targets. The biases are in units of percentage for attributes of variables like precipitation, and in units of degrees K for attributes of temperature. The function creates two heatmaps that show:

- magnitude of the mean biases across all the replicates
- standard deviation of biases across all the replicates

Usage

```
plotScenarios(sim, colMapRange = "default", plotAbs = T, showSD = T)
```

Arguments

`sim`	a list; contains a stochastic simulation or the summary of a stochastic simulation created using the function `generateScenarios`
`colMapRange`	a string; may be set to the character `"default"` or `"full"` or to a numeric vector of length 2. The argument specifies the range of data spanned in the colormap of the heatmap. If set to `"default"`, the colourmap limits of attributes that are in units of percentage is set to 0% to 10%, and the colourmap limits of the attributes of temperature is set to 0 degrees K to 1 degrees K. If set to `"full"`, the colourmap limits are set to the minimum and maximum values in the data. If a numeric vector is specified, the colourmap limits are set to the first (minimum) and second (maximum) values in the vector.
`plotAbs`	logical value, defaults to TRUE; determines whether the absolute value of the data is plotted (TRUE), or the raw value (which can be positive/negative) is plotted (FALSE).
`showSD`	logical value, defaults to TRUE; determines whether to plot heat maps showing standard deviation in biases (TRUE), or only mean biases (FALSE).

Details

The argument sim may be a full stochastic simulation generated using the function generateScenarios or the summary of the stochastic simulation generated using getSimSummary

Value

The function returns two R plots showing the biases in the targets of the scenarios generated using the function generateScenarios.

See Also

createExpSpace, generateScenarions, getSimSummary

Examples

```
## Not run:
# load simulated climates from Scott Creek example
data('egScottCreekSimStoch')
plotScenarios(sim.stoch)

## End(Not run)
```

---

runSystemModel *Runs a system model and outputs the system performance*

Description

runSystemModel uses time series of hydroclimatic variables generated using the function generateScenarios as input to a systemModel and collates the system performance for all the targets and replicates in the scenarios.

Usage

```
runSystemModel(sim, systemModel, systemArgs, metrics, varNames = NULL)
```

Arguments

sim	list; a simulation containing the scenarios generated using the function generateScenarios.
systemModel	a function; The function runs the system model using climate data in a list as input. The function is expected to be created by the user for specific system models. tankWrapper is an example system model function available in this package. runSystemModel calls the function systemModel with two arguments: • data: list; the climate data as a list in reference format, with POSIXct formatted *times* and climate variables **variable_name1** **variable_name2** in vector or matrix.

- systemArgs: list; containing the other arguments required by the system model.systemModel unpack the arguments from the list and uses them as required.
- metrics: string vector; containing the names of the performance metrics that the system model returns. It is recommended that the names also contain the units of the metric. See viewTankMetrics() for examples.

systemArgs	a list; containing the input arguments to systemModel.
metrics	a string vector; the names of the performance metrics the systemModel function returns.
varNames	a string vector; containing the names of the climate variables that are extracted from sim and used in system model. If NULL, then varNames determined from attribute names in sim$expSpace.

Details

The runSystemModel function code is structured to be simple and may be used as an example to create scripts that use scenarios generated using generateScenarios to run system models in other programming languages. Type runSystemModel to view the function code. The function tankWrapper in this package may be used as an example to create user defined functions for the systemModel argument. Refer to tankWrapper to understand how the systemModel is expected to use systemArgs and return the calculated performance metrics. The systemModel function is expected to return a named list of performance metrics. The elements of the vector should correspond to metrics.

Value

The function returns a list containing the performance metrics calculated by the systemModel. Each element of the list corresponds to a performance metric and is named using the metrics argument. Each element contains performance values calculated at all the target points in the exposure space in a matrix with nrow corresponding to the targets and ncol corresponding to the replicates.

See Also

tankWrapper, generateScenarios

Examples

```
## Not run:
# Example using tankWrapper as the systemModel
# ====================================================
# create an exposure space
attPerturb <- c("P_day_all_tot", "Temp_day_all_avg")
attPerturbType <- "regGrid"
attPerturbSamp <- c(10, 10)
attPerturbMin <- c(0.8, -1)
attPerturbMax <- c(1.2, 1)
expSpace <- createExpSpace(
  attPerturb = attPerturb,
  attPerturbSamp = attPerturbSamp,
```

```
  attPerturbMin = attPerturbMin,
  attPerturbMax = attPerturbMax,
  attPerturbType = attPerturbType
)
data(tankDat)
simScaling <- generateScenarios(
  reference = tank_obs,
  expSpace = expSpace,
  controlFile = "scaling"
)
# use the simulation to run a system model
systemArgs <- list(
  roofArea = 205, nPeople = 1, tankVol = 2400,
  firstFlush = 2.0, write.file = FALSE
)
tankMetrics <- viewTankMetrics()
systemPerf <- runSystemModel(
  sim = simScaling,
  systemModel = tankWrapper,
  systemArgs = systemArgs,
  metrics = tankMetrics[1:2]
)

## End(Not run)
```

`setSeasonalTiedAttributes` *Creates tied attributes which tie seasonal changes in attributes to annual changes*

Description

`setSeasonalTiedAttributes` returns a list containing tied attributes, matching seasonal attributes to non-stratified attributes. This list can be used to specify tied attributes in `createExpSpace()`

Usage

```
setSeasonalTiedAttributes(attSel)
```

Arguments

`attSel`	A char vector; the names of the attributes (perturbed/held) for which seasonal attributes will be tied to.

Value

A list describing tied attributes with first level corresponding to original perturbed/held attributes, and second level a vector with tied seasonal attributes.

Examples

```
attSel <- c("P_day_all_tot_m", "P_day_all_P99")
attTied <- setSeasonalTiedAttributes(attSel)
attTied
```

`shuffle_sim` *Post-processing to apply changes in temporal structure of annual precipitation.*

Description

`shuffle_sim` changes the temporal structure of annual climate to match a target change in a perturbed attribute

Usage

```
shuffle_sim(
  sim,
  clim,
  attPerturb = "P_day_all_tot_dwellTime",
  targetVals,
  targetType = "frac",
  seed = 1,
  annAR1coeffList = seq(-0.2, 0.9, 0.01),
  cSel = "mean"
)
```

Arguments

`sim`	a list; simulated climate object from `generateScenarios`
`clim`	a list; reference climate
`attPerturb`	a string; name of attribute to be perturbed
`targetVals`	a vector of numbers; targets for perturbed attributes
`targetType`	a vector of strings; type of change in target attributes (either 'frac' or 'diff')
`seed`	an integer; random number seed
`annAR1coeffList`	a vector of numbers; the range of values for annual AR(1) parameters used in grid search
`cSel`	an integer or string; for multi-site data, can either cvalculate perturbed attributes on a single site (integer) or the mean climate over all sites (`cSel='mean'`).

Value

A list with simulated perturbed climates. Same format as output from `generateScenarios`.

Examples

```
## Not run:
###############
# load dates, precip, PET and streamflow data for Scott Creek
data('data_A5030502')
# create reference data - won't include PET here since not modelled
clim_ref = list(times = data_A5030502$times,
                P = data_A5030502$P)
###############
# create exposure space for baseline climate - no perturbations

attPerturbType = "regGrid"
attPerturb = c('P_day_all_tot_m')
attPerturbSamp = c(1)
attPerturbMin = c(1)
attPerturbMax = c(1)
# will hold a number of attributes to historical values
# note: normP = P99/avg rainfall.
attHold = c('P_day_all_P99','P_day_all_avgDSD','P_day_all_nWet_m',
            'P_day_DJF_tot_m','P_day_DJF_normP99','P_day_DJF_avgDSD','P_day_DJF_nWet_m',
            'P_day_MAM_tot_m','P_day_MAM_normP99','P_day_MAM_avgDSD','P_day_MAM_nWet_m',
            'P_day_JJA_tot_m','P_day_JJA_normP99','P_day_JJA_avgDSD','P_day_JJA_nWet_m',
            'P_day_SON_tot_m','P_day_SON_normP99','P_day_SON_avgDSD','P_day_SON_nWet_m')
expSpace = createExpSpace(attPerturb = attPerturb,
                          attPerturbSamp = attPerturbSamp,
                          attPerturbMin = attPerturbMin,
                          attPerturbMax = attPerturbMax,
                          attPerturbType = attPerturbType,
                          attHold = attHold)

###############
# setup model settings

modelSelection = list()

modelSelection$modelType = list()
modelSelection$modelType$P = "latent" # latent variable model

modelSelection$modelParameterVariation = list()
modelSelection$modelParameterVariation$P = "seas" # parameters vary with season

modelSelection[["optimisationArguments"]] = list()
modelSelection[["optimisationArguments"]][["OFtol"]] = 0.1 # stop optimization when OF < OFtol

# set penalty weights. More weights to total, and attributes for 'all' data)
modelSelection[["penaltyAttributes"]]=c('P_day_all_tot_m','P_day_all_P99',
                                        'P_day_all_avgDSD','P_day_all_nWet_m')
modelSelection[["penaltyWeights"]] = c(3,2,2,2)

# write to JSON file
modelSelectionJSON = jsonlite::toJSON(modelSelection, pretty = TRUE, auto_unbox = TRUE)
controlFile = paste0(tempdir(), "\\eg_controlFile.json")
```

```
write(modelSelectionJSON, file = controlFile)

###############
# generate baseline climate scenarios

time.1 = Sys.time()
sim.base = generateScenarios(reference = clim_ref,
                             expSpace = expSpace,
                             controlFile = controlFile,
                             seedID = 1,
                             numReplicates = 1)
time.2 = Sys.time()
print(time.2-time.1)

###############
# shuffle baseline climate to introduce temporal structure at annual scale

sim.shuffle = shuffle_sim(sim=sim.base,
                          clim=clim_ref,
                          attPerturb = 'P_day_all_tot_dwellTime',
                          targetVals = c(1,1.5,2,2.5),
                          targetType = 'frac')

###############
# evaluate how changes in 'P_day_all_tot_dwellTime' affect other attributes

attSel = colnames(sim.base$expSpace$targetMat)
# other attributes
attSel = c(attSel,'P_day_all_tot_dwellTime','P_day_all_avgWSD',
           'P_day_all_P99.9','P_day_JJA_P99.9','P_day_SON_P99.9',
           'P_day_DJF_P99.9','P_day_MAM_P99.9')

att = 'P_day_all_tot_dwellTime'
par(mfrow=c(5,3),mar=c(4,7,2,1))
# plot changes in a single attribute with respect to perturbed attributes
plotPerformanceAttributesOAT(clim=clim_ref,
                             sim=sim.shuffle,
                             attPerturb=att,
                             attEval=attSel,
                             cex.main = 1.5,cex.xaxis = 1,cex.yaxis = 1)

## End(Not run)
```

`sim.stoch` *Example perturbed stochastic climates for Scott Creek.*

Description

Based on example in vignette for stochastic simulation.

Format

A large list with perturbed climate simulations and metadata

Source

Generated by generateScenarios() - see example in vignette

---

subdaily_synthetic_obs *Synthetic sub-daily rainfall data*

Description

Dataset of synthetic observations generated using BLRPM model.

Format

A list of climate data with *times* and *P*.

---

tank_obs *Observations for demo tank model examples and vignette*

Description

Dataset of observations for tank model examples

Format

A dataframe of observed climate data in the form *Year Month Day P Temp*.

tankPerformance — *A function to calculate difference performance from simulated tank behaviour*

Description

A function to calculate difference performance from simulated tank behaviour

Usage

```
tankPerformance(data=NULL,
                      roofArea=50,
                      nPeople=1,
                      tankVol=3000,
        firstFlush=1,
                      write.file=TRUE,
                      fnam="tankperformance.csv")
```

Arguments

data	A dataframe of observed climate data in the form *Year Month Day P Temp*.
roofArea	roof area in m2
nPeople	number of people using water
tankVol	tank volume in L
firstFlush	first flush depth over roof in mm
write.file	logical. write output tank timeseries to file T/F?
fnam	string indicating name of file

tankWrapper — *Wrapper function for a rain water tank system model*

Description

tankWrapper is a wrapper function for a rainwater tank system model in foreSIGHT. This function is used in examples in function help files and vignettes. This function may also be used as an example to create wrapper functions for other system models with scenarios generated using foreSIGHT in R or other programming languages.

Usage

```
tankWrapper(data, systemArgs, metrics)
```

Arguments

data	list; contains observed daily precipitation and temperature to be used to run the rain water tank system model in a list with entries *times*, *P*, *Temp*. Please refer data provided with the package that may be loaded using `data(tankDat)` for an example of the expected format of `data`.
systemArgs	a list; contains the input arguments to the rain water tank system model. The valid fields in the list are: • `roofArea`: numeric; the roof area in sq.m • `nPeople`: integer; number of people using water • `tankVol`: numeric; volume of the tank in L • `firstFlush`: numeric; first flush depth over roof in mm • `write.file`: logical; indicates whether output is to be written to file • `fnam`: string; name of the output file
metrics	string vector; the metrics of performance of the system model to be reported. The valid strings may be viewed using the function `viewTankMetrics()`

Value

The function returns a list containing the calculated values of the performance metrics specified in `metrics` after running the system model.

See Also

`runSystemModel`, `viewTankMetrics`

Examples

```
# view available performance metrics
viewTankMetrics()
# load example climate data to run the system model
data(tankDat)
systemArgs <- list(
  roofArea = 205, nPeople = 1, tankVol = 2400,
  firstFlush = 2.0, write.file = FALSE
)
tankWrapper(tank_obs, systemArgs,
  metrics = c("average daily deficit (L)", "reliability (fraction)")
)
```

---

viewAttributeDef *Prints the definition of an attribute*

Description

`viewAttributeDef` prints the short definition of a valid attribute

Usage

```
viewAttributeDef(attribute)
```

Arguments

attribute	A string; the name of the attribute.

See Also

createExpSpace

Examples

```
# To view the definition of any valid attribute
viewAttributeDef("P_day_all_tot_m")
```

---

viewAttributeFuncs *Prints the list of built-in attribute functions*

Description

viewAttributeFuncs prints the list of built-in attribute functions

Usage

```
viewAttributeFuncs()
```

See Also

viewAttributeDef, createExpSpace

Examples

```
# To view the list of built-in functions used to calculate attributes
viewAttributeFuncs()
```

viewDefaultOptimArgs *Prints the default optimisation arguments*

Description

viewDefautOptimArgs() prints the default values of optimisation arguments (optimisationArguments) used by generateScenarios

Usage

```
viewDefaultOptimArgs(optimizer = "RGN")
```

Arguments

optimizer	A string for the numerical optimizer. Default optimizer is 'RGN'.

Details

This a helper function that prints the default values of the optimisation arguments. The user may specify alternate values of these arguments in fields named according to the corresponding argument name nested under optimisationArguments in a JSON file to use as the controlFile input to the generateScenarios function.

See Also

writeControlFile

Examples

```
# To view the default optimisation arguments
viewDefaultOptimArgs()
```

viewModelParameters *Prints the names and bounds of the parameters of the stochastic models*

Description

viewModelParameters prints the names of the parameters of the stochastic model and its default minimum and maximum bounds. The stochastic model is specified using the function arguments.

Usage

```
viewModelParameters(variable, modelType, modelParameterVariation)
```

Arguments

variable	A string; the name of the variable. Type viewModels() to view valid variable names
modelType	A string; the model type. Use viewModels to view the valid values.
modelParameterVariation	A string; the parameter variation. Use viewModels to view the valid values.

Details

The available stochastic models can be viewed using the function viewModels(). This function prints the default ranges of the parameters of the stochastic model specified the stochastic model of interest.

See Also

viewModels, writeControlFile

Examples

```
viewModelParameters("P", "wgen", "annual")
viewModelParameters("P", "wgen", "harmonic")
```

---

viewModels *Prints the available stochastic model options*

Description

viewModels prints the stochastic model options available for the different hydroclimatic variables in foreSIGHT. These options may be used to create an controlFile for input to function generateScenarios.

Usage

```
viewModels(variable = NULL)
```

Arguments

variable	String; the variable name. Type viewModels() without arguments to view the valid variable names.

See Also

writeControlFile, generateScenarios

Examples

```
# To view the valid variable names use the function without arguments
viewModels()

# Examples to view the model options available for different variables
viewModels("P")
viewModels("Temp")
viewModels("Radn")
viewModels("PET")
```

`viewTankMetrics` *Prints the names of the performance metrics of the rain water tank system model*

Description

`viewTankMetrics` prints the names of the performance metrics available in the example rain water tank system model. T

Usage

```
viewTankMetrics()
```

Details

This is a helper function that does not take any input arguments. The user may specify one or more of the metric names as the `metric` argument of `tankWrapper` to select the performance metrics from the tank system model. to select the performance metrics.

See Also

`tankWrapper`

Examples

```
viewTankMetrics()
```

writeControlFile *Writes a sample* controlFile.json *file*

Description

writeControlFile() writes a sample controlFile.json file. The controlFile.json file is used to specify alternate model and optimisation options and used as an input to the function generateScenarios. The user may use the sample file created by this function as a guide to create an "controlFile.json" file for their application.

Usage

```
writeControlFile(
  jsonfile = "sample_controlFile.json",
  basic = TRUE,
  nml = NULL
)
```

Arguments

jsonfile	string; to specify the name of the json file to be written. The default name of the sample file is "sample_controlFile.json". The file will be written to the working directory of the user.
basic	logical (TRUE/FALSE); used to specify whether a "basic" or "advanced" sample file is to be written. The default is TRUE. A "basic" controlFile does not contain modelParameterBounds, and is sufficient for most applications.
nml	list; the namelist to be written to the json file, as an R list. This argument may be used to create a JSON file using an controlFile from an existing simulation. If this argument is set to NULL, the function writes the default model/optimisation options defined in the package to the json file.

Details

The function may be used without any input arguments to write a "basic" sample controlFile.

Value

A json file. The file may be used as an example to create an "controlFile.json" file for input to generateScenarios. An "controlFile.json" file may contain any subset of the fields listed below. The user may delete the unused fields from the file. The exception cases where it is mandatory to specify two fields together in controlFile are detailed as part of the list below.

- modelType: a list by variable. Each element of the list is a string specifying the type of stochastic model. if modelType is specified for a variable in controlFile, modelParameterVariation should also be specified. This is because these two fields together define the stochastic model. Use viewModels() to view the valid options for modelType by variable.

- modelParameterVariation: a list by variable. Each element of the list is a string specifying the type of the parameter variation (annual, seasonal, harmonic etc.) of the stochastic model. if modelParameterVariation is specified for a variable in controlFile, modelType should also be specified. This is because these two fields together define the stochastic model. Use viewModels() to view valid options for modelParameterVariation by variable.
- modelParameterBounds: a nested list by variable. Each element is a list containing the bounds of the parameters of the chosen stochastic model. This field exists to provide an option to overwrite the default bounds of the parameters of the stochastic model. Careful consideration is recommended prior to setting modelParameterBounds in the controlFile to overwrite the defaults provided in the package.
- optimisationArguments: a list. Contains the optimisation options used by function ga from the ga package. Brief definitions are given below.
  - optimizer: the numerical optimization routine. Options include 'RGN' for Robust Gauss Newton (using RGN::rgn), 'NM' for Nelder-Mead (using dfoptim::nmkb), 'SCE' for Shuffled Complex Evolution (using SoilHyP::SCEoptim). 'GA' for Genetic Algorithm (using GA::ga), Defaults to 'RGN'.
  - seed: random seed used (for first multistart) in numerical optimization (often for determining random initial parameter values). Default is 1.
  - obj.func: the type of objective function used (important only when penalty weights are not equal.
  - suggestions: suggestions for starting values of parameters for optimisation. Options include 'WSS' (weighted sum of squares) and SS_absPenalty (sum of squares plus absolute penalty)
  - nMultiStart: the number of multistarts used in optimization. Default is 5.
  - RGN.control: RGN optional arguments specified by control list in RGN::rgn.
  - NM.control: NM optional arguments specified by control list in dfoptim::nmkb.
  - SCE.control: SCE optional arguments specified by control list in SoilHyP::SCEoptim.
  - GA.args: GA optional arguments specified in GA::ga.
- penaltyAttributes: a character vector of climate attributes to place specific focus on during targeting via the use of a penalty function during the optimisation process. The penaltyAttributes should belong to attPerturb or attHold that are specified in the exposure space used as input to generateScenarios. If penaltyAttributes are specified in the controlFile, penaltyWeights should also be specified.
- penaltyWeights: a numeric vector; the length of the vector should be equal to the length of penaltyAttributes. penaltyWeights are the multipliers of the corresponding penaltyAttributes used during the optimisation.

See Also

generateScenarios, viewModels, viewDefaultOptimArgs

Examples

```
## Not run:
# To write a sample controlFile
writeControlFile()
```

```
# To write an advanced sample controlFile
writeControlFile(jsonfile = "sample_controlFile_advanced.json", basic = FALSE)

## End(Not run)
```

Index