Package 'hydroState'

August 9, 2025

Title Hidden Markov Modelling of Hydrological State Change

Version 0.2.0.0

Maintainer Tim Peterson <tim.peterson@monash.edu>

Depends R (>= 3.5.0)

Description Identifies regime changes in streamflow runoff not explained by variations in precipitation. The package builds a flexible set of Hidden Markov Models of annual, seasonal or monthly streamflow runoff with precipitation as a predictor. Suites of models can be built for a single site, ranging from one to three states and each with differing combinations of error models and auto-correlation terms. The most parsimonious model is easily identified by AIC, and useful for understanding catchment drought non-recovery: Peterson TJ, Saft M, Peel MC & John A (2021) <doi:10.1126/science.abd5085>.

Imports methods, DEoptim, sn, truncnorm, diagram, padr, zoo, graphics, checkmate, mathjaxr

BugReports https://github.com/peterson-tim-j/HydroState/issues

URL https://github.com/peterson-tim-j/HydroState, https://peterson-tim-j.github.io/HydroState/

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

ByteCompile true

Suggests knitr,
rmarkdown,
testthat (>= 3.0.0),
V8

Config/testthat/edition 3

VignetteBuilder knitr

RdMacros mathjaxr

BuildManual TRUE

Collate 'abstracts.R'
'parameters.R'
'Qhat.boxcox.R'
'Qhat.burbidge.R'

'Qhat.log.R'
'Qhat.none.R'
'QhatModel.homo.normal.linear.R'
'QhatModel.homo.normal.linear.AR1.R'
'QhatModel.homo.gamma.linear.R'
'QhatModel.homo.gamma.linear.AR1.R'
'QhatModel.homo.normal.linear.AR2.R'
'QhatModel.homo.gamma.linear.AR2.R'
'QhatModel.homo.normal.linear.AR3.R'
'QhatModel.homo.gamma.linear.AR3.R'
'QhatModel.homo.skewedNormal.linear.R'
'QhatModel.homo.skewedNormal.linear.AR1.R'
'QhatModel.homo.skewedNormal.linear.AR2.R'
'QhatModel.homo.skewedNormal.linear.AR3.R'
'QhatModel.subAnnual.homo.gamma.linear.R'
'QhatModel.subAnnual.homo.gamma.linear.AR1.R'
'QhatModel.subAnnual.homo.gamma.linear.AR2.R'
'QhatModel.subAnnual.homo.gamma.linear.AR3.R'
'RhatModel.homo.normal.linear.R'
'hydroState-package.R'
'markov.annualHomogeneous.R'
'hydroState.R'
'hydroState.allModels.R'
'hydroState.subAnnual.allModels.R'
'markov.annualHomogeneous.flickering.R'
'wrapper.R'

R topics documented:

`hydroState-package` *hydroState: Hidden Markov modeling for hydrological state change*

Description

`hydroState` is an R-package that identifies regime changes in streamflow time-series that is not explained by variations in precipitation.

Details

For details of the mathematics, and its use in understanding catchment drought non-recovery, see:

Peterson TJ, Saft M, Peel MC & John A (2021), Watersheds may not recover from drought, Science, DOI: 10.1126/science.abd5085

The package allows a flexible set of Hidden Markov Models (HMMs) of annual, seasonal or monthly time-step to be built and which includes precipitation as a predictor of streamflow. Suites of models can be build for a single catchment, ranging from from one to three states and each with differing combinations of error models and auto-correlation terms, allowing the most parsimonious model to easily be identified (by AIC). The entire package is written in R S4 object oriented code with user facing functions and vignettes.

Functions

- `build()` — Builds hydroState model
- `build.all()` — Builds all hydroState models
- `fit.hydroSata()` — Fit hydroState model(s)
- `setInitialYear()` — Sets state names given initial year
- `plot()` — Plot states or pseudo residuals over time
- `get.residuals` — Get pseudo residuals
- `get.states` — Get states
- `check()` — Check reliability of state predictions
- `get.AIC()` — Get AIC
- `get.seasons` — Get pseudo residuals

Vignettes

- `vignette("hydroState", package = "hydroState")`
- `vignette("Adjust the default state model", package = "hydroState")`
- `vignette("Seasonal and monthly models", package = "hydroState")`

Usage

The package contains a default `hydroState` model object that explains streamflow as a function of precipitation using a linear model. Once the model object is built (`build()`), the model is fitted (`fit.hydroState()`) to determine the most likely rainfall-runoff state at each time-step. To assess the adequacy of the fit, the residuals are plotted (`plot()`), and an adequate fit requires the residuals to be normally distributed, uniform, with minimal correlation and minimal trends. The resulting runoff states from the fitted model can then be evaluated over time (`plot()`) and even exported (`get.states()`) with the state values, confidence intervals, and conditional probabilities at each time-step. Input data requires a dataframe with catchment average runoff and precipitation at annual, seasonal, or monthly timesteps, and gaps with missing data are permitted. An example of this workflow with the default model is demonstrated within the `vignette("hydroState", package = "hydroState")` vignette.

To better explain the rainfall-runoff relationship, the default model can be adjusted by selecting various items within the `build()` function. These include:

- `data.transform`: transform streamflow observations in order to reduce skew: 'boxcox', 'log', 'burbidge', or 'none'

- `parameters`: account for auto-correlation through including the degree of auto-correlation: 'AR1', 'AR2', or 'AR3'
- `state.shift.parameters`: assign either the intercept, slope, or auto-correlation parameter as state dependent parameter
- `error.distribution`: adjust the error distribution with 'normal', 'gamma', or 'truc.normal'
- `seasonal.parameters`: account for intra-annual varation within the rainfall-runoff relationship
- `transition.graph`: set the number of possible states in the model (1, 2, or 3).

An example of how to adjust the default model is demonstrated within the `vignette("Adjust the default state model", package = "hydroState")` and `vignette("Seasonal and monthly models", package = "hydroState")`.

There is an additional option to construct all possible types of models using the `build.all()`, and compare them using the same `fit.hydroState()` function. The most likely model can be selected based on the AIC where the best model will have the lowest AIC. An example of this is demonstrated at the end of the `vignette("Adjust the default state model", package = "hydroState")` and `vignette("Seasonal and monthly models", package = "hydroState")`. To get stated, it is recommended to evaluate the default model at first with one state and again with two states.

Acknowledgments

The package development was funded by the Victorian Government The Department of Energy, Environment, and Climate Action (https://www.water.vic.gov.au/).

Author(s)

Maintainer: Tim Peterson <tim.peterson@monash.edu> (ORCID) [copyright holder]

Authors:

- Thomas Westfall <thomas.westfall1@monash.edu> (ORCID)

See Also

Useful links:

- https://github.com/peterson-tim-j/HydroState
- https://peterson-tim-j.github.io/HydroState/
- Report bugs at https://github.com/peterson-tim-j/HydroState/issues

---

`build` *Builds hydroState model*

Description

`build` builds a hydrostate model object with either a default model or the model can be specified with options from below. Every model depends on a linear base model where streamflow, Q, is a function of precipitation, P: $\hat{Q} = Pa_1 + a_0$. The default model is a variant of this base linear model with state shifts expected in the intercept, a_0, and standard deviation, std, of the rainfall-runoff relationship. There are additional options to adjust this model with auto-correlation terms and seasonal parameters that can be both independent of state or change with state. The number of states and assumed error distribution can also be selected. After the model is built, the hydroState model is ready to be fitted with `fit.hydroState()`

Usage

```
build(
  input.data = data.frame(year = c(), flow = c(), precip = c()),
  data.transform = "boxcox",
  parameters = c("a0", "a1", "std"),
  seasonal.parameters = NULL,
  state.shift.parameters = c("a0", "std"),
  error.distribution = "truc.normal",
  flickering = FALSE,
  transition.graph = matrix(TRUE, 2, 2)
)
```

Arguments

`input.data`	dataframe of annual, seasonal, or monthly runoff and precipitation observations. Gaps with missing data in either streamflow or precipitation are permitted. Monthly data is required when using `seasonal.parameters`.
`data.transform`	character sting with the method of transformation. The default is 'boxcox'. Other options: 'log', 'burbidge', 'none'
`parameters`	character vector of parameters to construct model. Required and default: `a0`, `a1`, `std`. Auto-correlation terms optional: `AR1`, `AR2`, or `AR3`.
`seasonal.parameters`	character vector of one or all parameters (`a0`, `a1`, `std`) defined as a sinusoidal function to represent seasonal variation. Requires monthly or seasonal data. Default is empty, no seasonal parameters.
`state.shift.parameters`	character vector of one or all parameters (`a0`, `a1`, `std`, `AR1`, `AR2`, `AR3`) able to shift as dependent on state. Default is `a0` and `std`.
`error.distribution`	character string of the distribution in the HMM error. Default is 'truc.normal'. Others include: 'normal' or 'gamma'
`flickering`	logical `TRUE`/`FALSE`. `TRUE` = allows more sensitive markov flickering between states over time. When `FALSE` (default), state needs to persist for at least three time steps before state shift can occur.
`transition.graph`	matrix given the number of states. Default is a 2-state matrix (2 by 2): `matrix(TRUE,2,2)`.

Details

`build`

There are a selection of items to consider when defining the rainfall-runoff relationship and investigating state shifts in this relationship. hydroState provides various options for modelling the rainfall-runoff relationship.

- Data gaps with `input.data`: When there is missing `input.data` in either the dependent variable, streamflow, or independent variable, precipitation, the emissions probability of the missing time-step is set equal to one. This essentially ignores the missing periods. The time step after the missing period has a state probability dependent on the length of the gap. The larger the gap, the closer the state probability gets to approaching a finite probability near zero (as the transition probabilities are recursively multiplied). When the model has auto-correlation terms and there are gaps in the dependent variable, the auto-correlation function

restarts at the beginning of each continuous period after the gap. This ignores auto-correlation at the first time steps after the gap. For instance, an 'AR1' model would ignore the contribution of the prior time step for the first (1) observation after the gap.

- Transform Observations with `data.transform`: Transforms streamflow observations to remove heteroscedasticity. Often there is skew within hydrologic data. When defining relationships between rainfall-runoff, this skew results in an unequal variance in the residuals, heteroscedasticity. Transforming streamflow observations is often required. There are several options to transform observations. Since the degree of transformation is not typically known, `boxcox` is the default. Other options include: `log`, `burbidge`, and of course, `none` when no transformation is performed.
- Model Structure with `parameters` and `seasonal.parameters`: The structure of the model depends on the `parameters`. hydroState simulates runoff, Q, as being in one of a finite states, i, at every time-step, t, depending on the distribution of states at prior time steps. This results in a runoff distribution for each state that can vary over time (${}_t\hat{Q}_i$). The model defines the relationship that is susceptible to state shifts with precipitation, P_t, as a predictor. This takes the form as a simple linear model ${}_t\hat{Q}_i = f(P_t)$:

$$ {}_t\hat{Q}_i = P_t a_1 + a_0 $$

  where a_0 and a_1 are constant parameters. These parameters and the model error, std, establish the rainfall-runoff relationship and are required parameters for every built model object. These are the default parameters: `c('a0', 'a1', 'std')`.
- Auto-correlation `parameters`: The relationship may contain serial correlation and would be better defined with an auto-regressive term:

$$ {}_t\hat{Q}_i = P_t a_1 + a_0 + AR1 {}_{t-1}\hat{Q} $$

  where `AR1` is the lag-1 auto-correlation term. Either, lag-1: `AR1`, lag-2: `AR2`, and lag-3: `AR3` auto-correlation coefficients are an option as additional parameters to better define the rainfall-runoff relationship.
- Sub-annual analysis with `seasonal.parameters`: Additional options include explaining the seasonal rainfall-runoff relationship with a sinusoidal function that better defines either of the constant parameters or error (a_0, a_1, std) throughout the year, i.e:

$$ a_0 = a_{0.disp} + a_{0.amp} * sin(2\pi(\frac{M_t}{12} + a_{0.phase})) $$

  where M_t is an integer month at t. Monthly streamflow and precipitation are required as `input.data` for the sub-annual analysis.
- State Dependent Parameters with `state.shift.parameters`: These are state dependent parameters where they are subject to shift in order to better explain the state of streamflow over time. Any or all of the previously chosen parameters can be selected (`a_0`, `a_1`, `std`, `AR1`, `AR2`, `AR3`). The default model evaluates shifts in the rainfall-runoff relationship with `a_0` `std` as state dependent parameters.
- Distribution of the Residuals with `error.distribution`: The distribution of the residuals (error) within a state of the model can be chosen to reduce skew and assist with making models statistically adequate (see `plot(pse.residuals = TRUE)`). Either normal: `normal`, truncated normal: `truc.normal`, or gamma: `gamma` distributions are acceptable. These error distribution ensures streamflow is greater than zero `Q > 0`, and specifically for `truc.normal` greater than or equal to zero `Q >= 0`. The default is `truc.normal`. Sub-annual models are restricted to only a `gamma` distribution.

- Markov flickering with `flickering`: When flickering is `FALSE`, the markov avoids state shifts for very short duration, and hence for a state shift to occur it should last for an extended period. The default is FALSE. If TRUE, flickering between more states is more sensitive. For further explanation on this method, see: Lambert et al., 2003. The current form of the Markov model is homogeneous where the transition probabilities are time-invariant.
- Number of States with `transition.graph`: The number of possible states in the rainfall-runoff relationship and transition between the states is selected with the transition.graph. The default is a 2-state model in a 2 by 2 unstructured matrix with a TRUE transition to and from each state (i.e. `matrix(TRUE,2,2)`). hydroState accepts 1-state up to 3-states (i.e. for 3-state unstructured transition graph: `matrix(TRUE,3,3)`). The unstructured transition graph allows either state to remain in the current state or transition between any state. For the 3-state transition graph, one may want to assume the transitions can only occur in a particular order, as in (Very low -> Low -> Normal->) rather than (Very low <-> Low <-> Normal <-> Very low). Thus, a structured graph is also acceptable (3-state structured: `matrix(c(TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE,TRUE),3,3)`). More details in Supplementary Materials of (Peterson TJ, Saft M, Peel MC & John A (2021), Watersheds may not recover from drought, Science, DOI: doi:10.1126/science.abd5085).

Value

A built hydroState model object ready to be fitted with `fit.hydroState()`

Examples

```
# Load data
data(streamflow_annual_221201)

## Build default annual hydroState model
model = build(input.data = streamflow_annual_221201)

# OR

## Build annual hydroState model with specified objects
# Build hydroState model with: 2-state, normal error distribution,
# 1-lag of auto-correlation, and state dependent parameters ('a1', 'std')
model = build(input.data = streamflow_annual_221201,
                  data.transform = 'boxcox',
                  parameters = c('a0','a1','std','AR1'),
                  state.shift.parameters = c('a1','std'),
                  error.distribution = 'normal',
                  flickering = FALSE,
                  transition.graph = matrix(TRUE,2,2))
```

---

build.all *Builds all hydroState models*

Description

`build.all` builds all possible combinations of hydroState models. The same fields are available as in `build()` in order to specify the type of models to be built. After all models are built, they are fitted using the same `fit.hydroState()` function.

Usage

```
build.all(
  input.data = data.frame(year = c(), flow = c(), precip = c()),
  data.transform = NULL,
  parameters = NULL,
  seasonal.parameters = NULL,
  state.shift.parameters = NULL,
  error.distribution = NULL,
  flickering = FALSE,
  transition.graph = NULL,
  summary.table = NULL,
  siteID = NULL
)
```

Arguments

`input.data`	dataframe of annual, seasonal, or monthly runoff and precipitation observations. Gaps with missing data in either streamflow or precipitation are permitted, and the handling of them is further discussed in `build`. Monthly data is required when using `seasonal.parameters` that assumes selected model parameters are better defined with a sinusoidal function.
`data.transform`	character string of method of transformation. If empty, the default builds all possible combinations of models with `boxcox` data transformation.
`parameters`	character vector of parameters to determine model form. If empty, the default builds all possible combinations of model forms.
`seasonal.parameters`	character vector of parameters with sinusoidal function to represent seasonal variation. Requires monthly or seasonal data. If empty and monthly or seasonal data is given, the default builds all possible combinations of models with a seasonal parameter for each and all parameters.
`state.shift.parameters`	character vector of one or all parameters to identify state dependent parameters. Only one set of parameters permitted. If empty, the default builds all possible model combinations with `c('a0','std')` as state shift parameters.
`error.distribution`	character string of the distribution in the HMM error. If empty, the default builds models with all possible combinations of error distribution: `c('truc.normal', 'normal','gamma')`
`flickering`	logical `TRUE/FALSE`. `TRUE` = allows more sensitive markov flickering between states over time. When `FALSE` (default), state needs to persist for at least three time steps before state shift can occur.
`transition.graph`	matrix given the number of states. If empty, the default builds models with all possible combinations of states: 1-state matrix (1 by 1): `matrix(TRUE,1,1)`, 2-state matrix (2 by 2): `matrix(TRUE,2,2)`, 3-state matrix (3 by 3): `matrix(TRUE,3,3)`.
`summary.table`	data frame with a table summarizing all built models and corresponding reference model. From function `summary()`. If empty, summary table will be built automatically.
`siteID`	character string of site identifier.

Details

build.all

All possible combinations of hydroState models are built for each auto-correlation lag and residual distribution from 1 to 3 states for a specified data transformation. This allows for investigation of state changes in the state.shift.parameters: the intercept c('a0', 'std') or slope c('a1', 'std'). To reduce the number of models in the search, specify which field(s) to remain constant. For example, to investigate the best model with the number of auto-correlation terms and number of states with a boxcox data transform and gamma distribution of the residuals, set data.transform to boxcox and error.distribution to gamma. If no fields are specified, all possible model combinations are built. If investigating state shifts in the intercept a0 and slope a1, it is recommended to build and fit the model combinations separately.

Value

A list of built hydroState models with every combination of objects ready to be fitted

Examples

```
# Load data
data(streamflow_annual_221201)

# Build all annual models with state shift in intercept 'a0'
all.annual.models = build.all(input.data = streamflow_annual_221201,
                                    state.shift.parameters = c('a0','std'),
                                     siteID = '221201')

# OR

# Build all annual models with state shift in slope 'a1'
all.annual.models = build.all(input.data = streamflow_annual_221201,
                                    state.shift.parameters = c('a1','std'),
                                     siteID = '221201')
```

---

check *Check reliability of state predictions*

Description

check After fitting the model, the reliability of the estimated states can be assessed by generating synthetic state sequences and then assessing how well the model identified them.

Usage

```
check(model, n.samples = 1e+05)
```

Arguments

model	fitted hydroState model.
n.samples	integer of samples to re-sample. Default is 100000.

Details

check

This validates the model's states at each time-step through re-sampling the input data and re-running the Viterbi algorithm. The input data is duplicated 100 times, and a synthetic series is generated from the model with sample states. This provides a time series of the transformed streamflow observations that can be compared with observations of the 'known' state. The Viterbi states of the re-sampled transformed observations are inferred, and the probability of the inferred state equaling the 'known' state is calculated.

Value

A data frame is returned with a matrix depending on the number of states. For a 2 state model, a 2x2 matrix is returned. The diagonal cell estimates the probability of correctly identifying that state. The off diagonals estimate the probability of incorrectly identifying a state that in state 2.

Examples

```
## Check reliability of state predictions (>5s to run)
## Not run:
check(model = model.annual.fitted.221201)

## End(Not run)
```

---

fit.hydroState *Fit hydroState model*

Description

`fit.hydroState` fits a single hydroState model (`build`) or multiple models (`build.all`) using global optimization by differential evolution DEoptim library. If fitting all models be sure to install and load the parallelly library. The fitting of all models may take hours or days, but the calibration can occur in parallel if the parallelly library is installed and loaded.

Usage

```
fit.hydroState(
  model,
  pop.size.perParameter = 10,
  max.generations = 500,
  doParallel = FALSE
)
```

Arguments

`model`	built hydroState model object, hydroState.allModels object, or hydroState.subAnnual.allModels object
`pop.size.perParameter`	integer that should be greater than or equal to the number of parameters in the model. The default is '10' and is sufficient for all models.

`max.generations`
: integer that will stop the optimizer when set number of generations are reached. The default is '500'.

`doParallel`
: TRUE/FALSE to perform fitting in parallel on all computer cores. Default is FALSE

Details

`fit.hydroState`

After a hydroState model object is built, the model is ready to fit to the observed streamflow through minimizing the negative log-likelihood function too calibrate model parameters. The only required input is the given built hydroState model object or hydroState.allModels object (all models from `build.all = TRUE`). When fitting all models, the models are fitted from least to most complex (least to max amount of parameters). Each model has a minimum of 5 and maximum of 20 calibration attempts to outperform the prior reference model else the model is rejected. For instance 'model.1State.normal.log.AR0' contains 3-parameters and is the reference model for 'model.1State.normal.log.AR1' which contains 4-parameters. The objective function of 'model.1State.normal.log.AR1' must calibrate the model with a lower negative log-likelihood than 'model.1State.normal.log.AR0'. These reference models are pre-defined, but this function allows the user to edit the reference models in the data.frame if needed using `summary`. Details on the likelihood function is as follows:

The likelihood function is estimated as:

$L_T = \delta P(x_1) + \Gamma \delta P(x_2) ... \Gamma \delta P(x_T) 1'$

where:

- δ is the initial state distribution, the initial probability of being in each state: $\delta = \begin{pmatrix} \delta_1 \\ 1 - \delta_1 \end{pmatrix}$
- $P(x)$ is the m x m diagonal emissions matrix of the probability density for each state using a lower tail truncated Gaussian distribution or a two-parameter Gamma distribution
  - $f_{Gau}(x = {}_{obs}\hat{q}_t; \mu = {}_t\hat{q}_i, \sigma = \sigma_i, a = 0) = \frac{\phi(\frac{x-\mu}{\sigma})}{\sigma(1-\Phi(\frac{a-\mu}{\sigma}))}$
  - $f_{Gam}(x = {}_{obs}\hat{q}_t; k = \frac{{}_t\hat{q}_i{}^2}{\sigma_i^2}, \theta = \frac{\sigma_i^2}{{}_t\hat{q}_i}) = \frac{x^{k-1}e^{\frac{x}{\theta}}}{\theta^k \Gamma(k)}$
  - where ϕ is the probability density function for the standard normal distribution, Φ is the cumulative distribution function for the standard normal distribution, k is the shape parameter, θ is the scale parameter, and $\Gamma(k)$ is the gamma function. For more details, refer to pg. 8-17 in Supplementary Materials of (Peterson TJ, Saft M, Peel MC & John A (2021), Watersheds may not recover from drought, Science, DOI: doi:10.1126/science.abd5085).
- Γ is the transition matrix
- T is the number of time-steps.

Value

A fitted hydroState model

Examples

```
# Load data
data(streamflow_annual_221201)

## Build default annual hydroState model
model = build(input.data = streamflow_annual_221201)
```

```
## Fit built model (runtime ~ 14 sec)
## Not run:
model = fit.hydroState(model)

## End(Not run)

## Fit all built models (runtime > several hours)
# Load data
data(streamflow_annual_221201)

## Build all annual models
all.annual.models = build.all(input.data = streamflow_annual_221201, siteID = '221201')

## Not run:
# Fit all (runtime > several hours)
all.annual.models = fit.hydroState(all.annual.models)


## End(Not run)
```

`get.AIC` *Get AIC*

Description

`get.AIC` retrieves Akaike information criteria from a fitted hydroState model object or all models.

Usage

```
get.AIC(model)
```

Arguments

`model`	fitted `hydroState` model object.

Details

`get.AIC`

The AIC is the negative log-likelihood of the model plus a penalty for model parameters. This function can be performed on a single model or a selection of models to find the lowest AIC of the set.

Value

AIC value of a single model or a list variable of AIC values for al models

Examples

```
# Load fitted model
data(model.annual.fitted.221201)

## AIC of a single model
get.AIC(model.annual.fitted.221201)

## Lowest AIC of a model set
get.AIC(all.models.annual.fitted.407211)
```

`get.residuals` *Get pseudo residuals*

Description

The pseudo residuals were derived from the conditional probabilities of the observations. At each time-step, the pseudo residual is the probability of an observation occurring given the prior observations and latter observations.

Usage

```
get.residuals(model)
```

Arguments

`model`	fitted hydroState model object.

Details

`get.residuals`

`get.residuals` retrieves residuals from the fitted model and exports them as a data frame.

Value

Data frame of residuals for each time-step

Examples

```
# Load fitted model
data(model.annual.fitted.221201)

## Get residuals in a dataframe
get.residuals(model = model.annual.fitted.221201)
```

get.seasons *Get seasons*

Description

Aggregates monthly data to 4 seasons in a year.

Usage

```
get.seasons(
  input.data = data.frame(year = c(), month = c(), flow = c(), precip = c())
)
```

Arguments

input.data	dataframe of monthly runoff and precipitation observations. Gaps with missing data in either streamflow or precipitation are permitted, and the handling of them is further discussed in build. Monthly data is required when using seasonal.parameters that assumes selected model parameters are better defined with a sinusoidal function.

Details

get.seasons

This function takes sums monthly runoff and precipitation observations into 4 seasons of a year.

Value

A dataframe of seasonal observations with an additional column counting the number of months in each season.

Examples

```
# Load data
data(streamflow_monthly_221201)

# aggregate monthly data to seasonal
streamflow_seasonal_221201 = get.seasons(streamflow_monthly_221201)
```

get.states *Get states*

Description

get.states uses the Viterbi algorithm to globally decode the model and estimate the most probable sequence of states.

Usage

```
get.states(model)
```

Arguments

model	fitted hydroState model object.

Details

get.states

These dataframe of results include:

- time-step: year and possibly either season or month for subannual analysis
- Viterbi State Number: state number (i.e. 1, 2, or 3) to differentiate states
- Obs. flow: streamflow observations
- Viterbi Flow: flow values of the Viterbi state including the 5\
- Normal State Flow: flow values of the normal state including the 5\
- Conditional Prob: conditional probabilities for each state show the probability of remaining in the given state. When the conditional probability is closer to 1, there is a higher probability that hydroState model remains in that state for the next time-step.
- Emission Density: emission density for each state is the result of multiplying the conditional probabilities by the transition probabilities at each timestep.

Value

data frame of results to evaluate the rainfall-runoff states over time

Examples

```
# Load fitted model
data(model.annual.fitted.221201)

## Set initial year to set state names
model.annual.fitted.221201 =
                setInitialYear(model = model.annual.fitted.221201,
                               initial.year = 1990)

## Get states
model.annual.fitted.221201.states =
                get.states(model = model.annual.fitted.221201)
```

plot.hydroState *Plot states or pseudo residuals over time*

Description

plot produces several figures to visualize pseudo residuals or results of the markov states over time. setInitialYear is required before plot. It is recommend to evaluate the pseudo residuals before the markov states. The pseudo residuals are the probability of an observation occurring at each time-step given the prior observations and latter observations, and these are derived from the conditional probabilities of the observations. The markov states are from the Viterbi algorithm globally decoding the model to estimate the most probable sequence of states.

Usage

```
## S3 method for class 'hydroState'
plot(
  x,
  ...,
  pse.residuals = FALSE,
  ind.variable = TRUE,
  dep.variable = TRUE,
  dep.variable.transformed = TRUE,
  cond.state.prob = TRUE,
  siteID = NULL,
  do.pdf = FALSE
)
```

Arguments

x	is the fitted hydroState model object.
...	additional arguments passed for plotting, none available at this time.
pse.residuals	option to plot pseudo residuals. Default is FALSE.
ind.variable	option to plot independent variable over time. Default is TRUE.
dep.variable	option to plot dependent variable and states over time. Default is TRUE.
dep.variable.transformed	option to plot transformed dependent variable and states over time. Default is TRUE.
cond.state.prob	option to plot the conditional state probabilities over time for each state. Default is TRUE.
siteID	character string of catchment identifier (i.e. gauge ID). Default is NULL. Only recommended when do.pdf = TRUE.
do.pdf	option to export figures as a pdf. Default is FALSE.

Details

plot

plot produces five figures of psuedo residuals OR up to four figures of the results from the fitted hydroState model. When the pse.residuals is FALSE, the default plot produces all four result

figures. Figures are more easily viewed as a pdf exported to the current working directory (`do.pdf = TRUE`).

- psuedo residual figures
  - A) Time-series of normal-pseudo residuals to ensure the residuals each year are within the confidence intervals.
  - B) Auto-correlation function (ACF) of normal-pseudo residuals to ensure there is minimal serial correlation in residuals. Lag spikes should be below confidence interval at each lag (except 0).
  - C) Histogram of uniform-pseudo residuals should show uniform distribution (equal frequency for each residual value)
  - D) Histogram of normal-pseudo residuals should show normal distribution centered on zero and with no skew
  - E) Quantile-Quantile (Q-Q) plot where normal-pseudo residuals vs. theoretical quantities should align on the diagonal line. The last plot contains the Akaike information criterion (AIC) and Shapiro-Wilk p-value. The AIC is an estimator to determine the most parsimonious, best performing model given the number of parameters. When comparing models, the lowest AIC is the best performing model. Shapiro-Wilks test for normality in the residuals and a p-value greater than 0.05 (chosen alpha level) indicates the residuals are normally distributed; the null hypothesis that the residuals are normally distributed is not rejected.
- markov state figures
  - A) independent variable: precipitation
  - B) dependent variable and states: streamflow observations, most likely state, and relative normal state estimate
  - C) transformed dependent variable and states: transformed streamflow observations and most likely state
  - D) conditional state probabilities for each state: probability of hydroState model remaining in given state

These figures are often large, and below are a few common errors when the plotting window is too small. Exporting the plots as a pdf is recommend for the pseudo residual figure (`do.pdf = TRUE`).

- "Error in plot.new() : figure margins too large": reset plot window with "dev.off()", enlarge plot area and re-run `plot.residuals`.
- "Error in par(op) : invalid value specified for graphical parameter "pin" if the R plot window is not reset with "dev.off", an additional `plot.residuals` attempt will result in this error.

Value

plots to evaluate rainfall-runoff states over time along with observations and the conditional probabilities of each state.

Examples

```
# Load fitted model
data(model.annual.fitted.221201)

## Set initial year to set state names
model.annual.fitted.221201 =
                  setInitialYear(model = model.annual.fitted.221201,
                  initial.year = 1990)
```

```
## Plot only residuals
plot(model.annual.fitted.221201, pse.residuals = TRUE)

## Plot all markov state figures
plot(model.annual.fitted.221201)

## Plot only dependent variable transformed with markov states
plot(model.annual.fitted.221201,
             ind.variable = FALSE,
             dep.variable = FALSE,
             dep.variable.transformed = TRUE,
             cond.state.prob = FALSE)
```

`setInitialYear` *Sets state names given initial year*

Description

sets the state names for each time-step relative to the initial year given

Usage

```
setInitialYear(model, initial.year)
```

Arguments

`model`	fitted hydroState model object.
`initial.year`	integer with year (YYYY). Default is first year in input.data.

Details

`setInitialYear`

hydroState assigns names to the computed states. This requires choosing an initial year where the state value from that year will be named 'Normal'. Other state values will be given names relative to the state value in the initial year. The choice of the initial year does not affect results. It is a means to more easily interpret the difference in state values relative to each other. It is best to choose a year based on the question being asked. For example, in testing the impact of drought, a year before the beginning of the drought, 1990, was selected as an initial year when conditions were considered 'Normal' (Peterson TJ, Saft M, Peel MC & John A (2021), Watersheds may not recover from drought, Science, DOI: doi:10.1126/science.abd5085)

Value

A fitted hydroState model object with state names for each time-step ready for `plot`

Examples

```
# Load fitted model
data(model.annual.fitted.221201)

## Set initial year to set state names
model.annual.fitted.221201 =
                setInitialYear(model = model.annual.fitted.221201,
                               initial.year = 1990)
```

`summary.hydroState.allModels`
Summarize all models

Description

`summary` outputs a summary table of all built hydroState models and allows users to edit the reference models for calibration.

Usage

```
## S3 method for class 'hydroState.allModels'
summary(object, ...)
```

Arguments

`object`	hydroState.allModels object with a list of models from `build.all`
`...`	No additional input required

Details

`summary`

For every model object in `build.all`, there is a reference model for calibration. The reference model is a slightly simpler model with one less parameter. During calibration with `fit`, the model performance must exceed the the performance of the reference model else the model is rejected. This function is used to output the `summary.table` and adjust the reference models. Afterwards, all models can be re-build with including this `summary.table` in the `build.all` function.

Value

A data.frame with a summary table of all models and reference models

Examples

```
# Show summary table of all fitted model details and reference models

## Not run:
all.models.ref.table = summary(all.models)

## End(Not run)
```

Index