Package: conduits (via r-universe)

November 12, 2024

Title CONDitional UI for Time Series normalisation

Version 1.0.0

Description Provide a user interface for conditionally normalising a timeseries.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports broom, dplyr, mgcv, purrr, rlang, scales, splines, stats, tibble, tidyr, tsibble, forecast

Depends R (>= 4.1.0)

Suggests fable, knitr, mgcViz, rmarkdown, patchwork

URL https://github.com/PuwasalaG/conduits

BugReports https://github.com/PuwasalaG/conduits/issues

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev libssl-dev

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

RemoteUrl https://github.com/PuwasalaG/conduits

RemoteRef HEAD

RemoteSha 80a48697c05463afa641a318d8ab1f457bc79cc4

Contents

augment.conditional_acf 2
augment.conditional_ccf 3
augment.conditional_moment 5
calc_dt_CI 6
conditional_acf 7

conditional_ccf 9
conditional_mean 11
conditional_var 12
conduits 14
estimate_dt 14
NEON_PRIN_5min_cleaned 16
normalize 16
unnormalize 17

Index **19**

`augment.conditional_acf` *Augment data with information from a conditional auto-correlation fit*

Description

This function produces estimated conditional autocorrelation between x_t and y_t at lag k, i.e. $r_k = E(x_ty_t+klz_t)$.

Usage

```
## S3 method for class 'conditional_acf'
augment(x, ...)
```

Arguments

`x`	Model object of class "conditional_acf" returned from `conditional_acf` with information to append to observations.
`...`	Additional arguments, not currently used.

Value

A `tibble` with information about data points.

Examples

```
old_ts <- NEON_PRIN_5min_cleaned |>
  dplyr::select(
    Timestamp, site, turbidity, level,
    conductance, temperature
  ) |>
  tidyr::pivot_wider(
    names_from = site,
    values_from = turbidity:temperature
  )

fit_mean <- old_ts |>
  conditional_mean(turbidity_downstream ~
```

```
    s(level_upstream, k = 8) +
    s(conductance_upstream, k = 8) +
    s(temperature_upstream, k = 8))

fit_var <- old_ts |>
  conditional_var(
    turbidity_downstream ~
      s(level_upstream, k = 7) +
      s(conductance_upstream, k = 7) +
      s(temperature_upstream, k = 7),
    family = "Gamma",
    fit_mean = fit_mean
  )
fit_c_acf <- old_ts |>
  tidyr::drop_na() |>
  conditional_acf(
    turbidity_upstream ~ splines::ns(level_upstream, df = 5) +
      splines::ns(conductance_upstream, df = 5),
    lag_max = 10, fit_mean = fit_mean, fit_var = fit_var,
    df_correlation = c(5, 5)
  )

data_inf <- fit_c_acf |> augment()
```

`augment.conditional_ccf` *Augment data with information from a conditional cross-correlation fit*

Description

This function produces estimated conditional cross-correlation between x_t and y_t at lag k, i.e. $r_k = E(x_ty_t+klz_t)$.

Usage

```
## S3 method for class 'conditional_ccf'
augment(x, ...)
```

Arguments

x	Model object of class "conditional_ccf" returned from `conditional_ccf` with information to append to observations.
...	Additional arguments, not currently used.

Value

A `tibble` with information about data points.

Examples

```
old_ts <- NEON_PRIN_5min_cleaned |>
  dplyr::select(
    Timestamp, site, turbidity, level,
    conductance, temperature
  ) |>
  tidyr::pivot_wider(
    names_from = site,
    values_from = turbidity:temperature
  )

fit_mean_y <- old_ts |>
  conditional_mean(turbidity_downstream ~
    s(level_upstream, k = 8) +
    s(conductance_upstream, k = 8) +
    s(temperature_upstream, k = 8))

fit_var_y <- old_ts |>
  conditional_var(
    turbidity_downstream ~
      s(level_upstream, k = 7) +
      s(conductance_upstream, k = 7) +
      s(temperature_upstream, k = 7),
    family = "Gamma",
    fit_mean = fit_mean_y
  )

fit_mean_x <- old_ts |>
  conditional_mean(turbidity_upstream ~
    s(level_upstream, k = 8) +
    s(conductance_upstream, k = 8) +
    s(temperature_upstream, k = 8))

fit_var_x <- old_ts |>
  conditional_var(
    turbidity_upstream ~
      s(level_upstream, k = 7) +
      s(conductance_upstream, k = 7) +
      s(temperature_upstream, k = 7),
    family = "Gamma",
    fit_mean = fit_mean_x
  )

fit_c_ccf <- old_ts |>
  tidyr::drop_na() |>
  conditional_ccf(
    I(turbidity_upstream * turbidity_downstream) ~ splines::ns(
      level_upstream,
      df = 5
    ) +
      splines::ns(conductance_upstream, df = 5),
    lag_max = 10,
```

```
    fit_mean_x = fit_mean_x, fit_var_x = fit_var_x,
    fit_mean_y = fit_mean_y, fit_var_y = fit_var_y,
    df_correlation = c(5, 5)
  )

data_inf <- fit_c_ccf |> augment()
```

`augment.conditional_moment` *Augment data with information from a conditional mean fit or conditional variance fit*

Description

This function produces partial residuals for each predictor, and the estimated conditional means, standard error and confidence limits.

Usage

```
## S3 method for class 'conditional_moment'
augment(x, level = 0.95, ...)
```

Arguments

`x`	Model object of class "conditional_moment" returned from `conditional_mean` or `conditional_var` with information to append to observations.
`level`	Confidence level. Default is set to 0.95.
`...`	Additional arguments, not currently used

Value

A `tibble` with information about data points.

See Also

`gam`

Examples

```
data <- NEON_PRIN_5min_cleaned |>
  dplyr::filter(site == "upstream") |>
  dplyr::select(Timestamp, turbidity, level, conductance, temperature)

fit_mean <- data |>
  conditional_mean(turbidity ~ s(level, k = 8) +
    s(conductance, k = 8) + s(temperature, k = 8))

data_inf <- fit_mean |> augment()
```

calc_dt_CI *Computing bootstrapped confidence intervals for dt*

Description

This function computes the bootstrapped confidence intervals for dt. It resample the residuals from the various models used in the conditional cross-correlation calculation to generate new data. As the residuals are serially correlated, a sieve bootstrap approach to capture the autocorrelation structure in the data.

Usage

```
calc_dt_CI(x, m, new_data = NULL)
```

Arguments

x	Model object of class "conditional_ccf" returned from conditional_ccf
m	number of replications for boostrap confidence intervals
new_data	the dataset with the some predictors that are set to the median value (if required). Default is set to NULL.

Value

A tibble with estimated time lag "dt"

Author(s)

Priyanga Dilini Talagala & Puwasala Gamakumara

Examples

```
## Not run:
old_ts <- NEON_PRIN_5min_cleaned |>
  dplyr::select(
    Timestamp, site, turbidity, level, temperature
  ) |>
  tidyr::pivot_wider(
    names_from = site,
    values_from = turbidity:temperature
  )
fit_mean_y <- old_ts |>
  conditional_mean(turbidity_downstream ~
    s(level_upstream, k = 5) +
    s(temperature_upstream, k = 5)
  )
fit_var_y <- old_ts |>
  conditional_var(
    turbidity_downstream ~
```

```
      s(level_upstream, k = 4) +
      s(temperature_upstream, k = 4),
    family = "Gamma",
    fit_mean = fit_mean_y
  )
fit_mean_x <- old_ts |>
  conditional_mean(turbidity_upstream ~
    s(level_upstream, k = 5) +
    s(temperature_upstream, k = 5)
  )
fit_var_x <- old_ts |>
  conditional_var(
    turbidity_upstream ~
      s(level_upstream, k = 4) +
      s(temperature_upstream, k = 4),
    family = "Gamma",
    fit_mean = fit_mean_x
  )
fit_c_ccf <- old_ts |>
  tidyr::drop_na() |>
  conditional_ccf(
    I(turbidity_upstream * turbidity_downstream) ~
      splines::ns(level_upstream, df = 3) +
      splines::ns(temperature_upstream, df = 3),
    lag_max = 10,
    fit_mean_x = fit_mean_x, fit_var_x = fit_var_x,
    fit_mean_y = fit_mean_y, fit_var_y = fit_var_y,
    df_correlation = c(3, 3)
  )
df_dt <- fit_c_ccf |> calc_dt_CI(100)

# Calculate  dt vs an  upstream covariate while holding the
# remaining upstream covariates at their medians
new_data <- fit_c_ccf$data
new_data <- new_data |>
  dplyr::mutate(temperature_upstream = median(temperature_upstream))
df_dt2 <- fit_c_ccf |> calc_dt_CI(100, new_data)

## End(Not run)
```

`conditional_acf` *Computing conditional autocorrelations at given lags*

Description

This function computes autocorrelation between x_t and y_t+k at $k = 1,2,...$ conditional on a set of time series z_t

Usage

```
conditional_acf(data, formula, lag_max, fit_mean, fit_var, df_correlation)
```

Arguments

data	a tibble containing all the time series including $ystar*ystar_t-k$ which are uniquely identified by the corresponding Timestamp.
formula	A GAM formula. See formula.gam.
lag_max	Maximum lag at which to calculate the conditional acf
fit_mean	Model object of class "conditional_moment" returned from conditional_mean
fit_var	Model object of class "conditional_moment" returned from conditional_var
df_correlation	a vector specifying the degrees of freedom to be considered for each numerical predictor when fitting additive models for conditional auto-correlations. Each component of the vector should corresponds to each predictor specified in "z_numeric".

Details

Suppose x_t and y_t are conditionally normalised with respect to z_t using conditional_mean and conditional_var. Then we can estimate the conditional cross-correlation between x_t and y_t at lag k, i.e. $r_k = E(x_ty_t+k|z_t)$ via generalised additive models (GAM). conditional_ccf uses natural splines implemented in splines package to estimate the conditional cross-correlations between two time series given a set of time series predictors. Users first need to normalise x_t and y_t at lag k using conditional_mean and conditional_var

Value

The function returns a list of objects of class "glm" as described in glm.

See Also

glm

Examples

```
old_ts <- NEON_PRIN_5min_cleaned |>
  dplyr::select(
    Timestamp, site, turbidity, level,
    conductance, temperature
  ) |>
  tidyr::pivot_wider(
    names_from = site,
    values_from = turbidity:temperature
  )

fit_mean <- old_ts |>
  conditional_mean(turbidity_downstream ~
    s(level_upstream, k = 8) +
    s(conductance_upstream, k = 8) +
```

```
    s(temperature_upstream, k = 8))

fit_var <- old_ts |>
  conditional_var(
    turbidity_downstream ~
      s(level_upstream, k = 7) +
      s(conductance_upstream, k = 7) +
      s(temperature_upstream, k = 7),
    family = "Gamma",
    fit_mean = fit_mean
  )
fit_c_acf <- old_ts |>
  tidyr::drop_na() |>
  conditional_acf(
    turbidity_upstream ~ splines::ns(level_upstream, df = 5) +
      splines::ns(conductance_upstream, df = 5),
    lag_max = 10, fit_mean = fit_mean, fit_var = fit_var,
    df_correlation = c(5, 5)
  )
```

`conditional_ccf` *Computing conditional cross-correlations at given lags*

Description

This function computes cross correlation between x_t and y_t+k at $k = 1,2,...$ conditional on a set of time series z_t

Usage

```
conditional_ccf(
  data,
  formula,
  lag_max = 10,
  fit_mean_x,
  fit_var_x,
  fit_mean_y,
  fit_var_y,
  df_correlation
)
```

Arguments

`data`	a tibble containing all the time series including ystar*xstar which are uniquely identified by the corresponding Timestamp.
`formula`	A GAM formula. The response variable should be in the format of I(x*y) ~ . See `formula.gam`.
`lag_max`	Maximum lag at which to calculate the conditional ccf

fit_mean_x	Model object of class "conditional_moment" returned from conditional_mean for series x
fit_var_x	Model object of class "conditional_moment" returned from conditional_var for series x
fit_mean_y	Model object of class "conditional_moment" returned from conditional_mean for series y
fit_var_y	Model object of class "conditional_moment" returned from conditional_var for series y
df_correlation	a vector specifying the degrees of freedom to be considered for each numerical predictor when fitting additive models for conditional cross-correlations. Each component of the vector should corresponds to the degrees of freedom each predictor.

Details

Suppose x_t and y_t are conditionally normalised with respect to z_t using conditional_mean and conditional_var. Then we can estimate the conditional cross-correlation between x_t and y_t at lag k, i.e. $r_k = E(x_ty_t+k|z_t)$ via generalised additive models (GAM). conditional_ccf uses natural splines implemented in splines package to estimate the conditional cross-correlations between two time series given a set of time series predictors. Users first need to normalise x_t and y_t at lag k using conditional_mean and conditional_var

Value

The function returns a list of objects of class "glm" as described in glm. the length og the list is equal to lag_max

See Also

glm

Examples

```
old_ts <- NEON_PRIN_5min_cleaned |>
  dplyr::select(
    Timestamp, site, turbidity, level,
    conductance, temperature
  ) |>
  tidyr::pivot_wider(
    names_from = site,
    values_from = turbidity:temperature
  )

fit_mean_y <- old_ts |>
  conditional_mean(turbidity_downstream ~
    s(level_upstream, k = 8) +
    s(conductance_upstream, k = 8) +
    s(temperature_upstream, k = 8))

fit_var_y <- old_ts |>
```

```
  conditional_var(
    turbidity_downstream ~
      s(level_upstream, k = 7) +
      s(conductance_upstream, k = 7) +
      s(temperature_upstream, k = 7),
    family = "Gamma",
    fit_mean = fit_mean_y
  )

fit_mean_x <- old_ts |>
  conditional_mean(turbidity_upstream ~
    s(level_upstream, k = 8) +
    s(conductance_upstream, k = 8) +
    s(temperature_upstream, k = 8))

fit_var_x <- old_ts |>
  conditional_var(
    turbidity_upstream ~
      s(level_upstream, k = 7) +
      s(conductance_upstream, k = 7) +
      s(temperature_upstream, k = 7),
    family = "Gamma",
    fit_mean = fit_mean_x
  )

fit_c_ccf <- old_ts |>
  tidyr::drop_na() |>
  conditional_ccf(
    I(turbidity_upstream * turbidity_downstream) ~ splines::ns(
      level_upstream,
      df = 5
    ) +
      splines::ns(temperature_upstream, df = 5),
    lag_max = 10,
    fit_mean_x = fit_mean_x, fit_var_x = fit_var_x,
    fit_mean_y = fit_mean_y, fit_var_y = fit_var_y,
    df_correlation = c(5, 5)
  )
```

`conditional_mean` *Estimating conditional mean of a time series*

Description

This function estimates the means of a time series conditional on a set of other times series via additive models.

Usage

```
conditional_mean(data, formula)
```

Arguments

data	a tibble containing all the time series which are uniquely identified by the corresponding Timestamp.
formula	A GAM formula. See formula.gam. The details of model specification are given under 'Details'.

Details

Suppose x_t is a time series where its mean is a function of z_t. i.e. $E(x_t|z_t) = m_x(z_t)$. Then $m_x(z_t)$ can be estimated via generalised additive models (GAM). This function uses GAMs implemented in mgcv package to estimate the conditional means of a time series given a set of time series predictors.

Value

The function returns an object of class "gam" as described in gamObject.

See Also

gam

Examples

```
data <- NEON_PRIN_5min_cleaned |>
  dplyr::filter(site == "upstream") |>
  dplyr::select(Timestamp, turbidity, level, conductance, temperature)

fit_mean <- data |>
  conditional_mean(turbidity ~ s(level, k = 8) +
    s(conductance, k = 8) + s(temperature, k = 8))
```

---

conditional_var *Estimating conditional variance of a time series*

Description

This function estimates the variance of a time series conditional on a set of other times series via additive models.

Usage

```
conditional_var(data, formula, family = c("Gamma", "lognormal"), fit_mean)
```

Arguments

data	A tibble containing all the time series which are uniquely identified by the corresponding Timestamp.
formula	An object of class "formula": a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
family	the family to be used in conditional variance model. Currently this can take either "Gamma" or "lognormal".
fit_mean	A GAM object return from conditional_mean

Details

Suppose x_t is a time series where its variance is a function of z_t. i.e. $Var(x_t|z_t) = v_x(z_t)$. Then $v_x(z_t)$can be estimated via generalised additive models (GAM). This function uses GAMs implemented in mgcv package to estimate the conditional variance of a time series given a set of time series predictors.

Value

The function returns an object of class "gam" as described in gamObject.

See Also

gam and ns.

Examples

```
data <- NEON_PRIN_5min_cleaned |>
  dplyr::filter(site == "upstream") |>
  dplyr::select(Timestamp, turbidity, level, conductance, temperature)

fit_mean <- data |>
  conditional_mean(turbidity ~ s(level, k = 8) +
    s(conductance, k = 8) + s(temperature, k = 8))
## Not run:
fit_var <- data |>
  conditional_var(
    turbidity ~ s(level, k = 7) + s(conductance, k = 7) + s(temperature, k = 7),
    family = "Gamma",
    fit_mean = fit_mean
  )

## End(Not run)
```

conduits *conduits: CONDitional User Interface for Time Series normalisation*

Description

Methods and tools for conditional normalisation of time series using additive models. This includes functions to estimate conditional means, conditional variances, conditional autocorrelation functions and conditional cross-correlation functions. Examples show these functions being used to estimate river flow time between two sensor locations in a river system.

Author(s)

Puwasala Gamakumara, Priyanga Dilini Talagala, Rob J Hyndman

estimate_dt *Estimating time delay between two sensors in a river system*

Description

This function estimates the time that takes water to flow from an upstream location to a downstream location conditional on the observed water-quality variables from the upstream sensor. That time lag is defined as the lag that gives maximum cross-correlation conditional on upstream water-quality variables.

Usage

```
estimate_dt(x)
```

Arguments

x	Model object of class "conditional_ccf" returned from conditional_ccf

Value

A tibble with estimated time lag "dt" and corresponding maximum cross-correlation

Author(s)

Puwasala Gamakumara & Priyanga Dilini Talagala

Examples

```
old_ts <- NEON_PRIN_5min_cleaned |>
  dplyr::select(
    Timestamp, site, turbidity, level, temperature
  ) |>
  tidyr::pivot_wider(
    names_from = site,
    values_from = turbidity:temperature
  )

fit_mean_y <- old_ts |>
  conditional_mean(turbidity_downstream ~
    s(level_upstream, k = 5) +
    s(temperature_upstream, k = 5))

fit_var_y <- old_ts |>
  conditional_var(
    turbidity_downstream ~
      s(level_upstream, k = 4) +
      s(temperature_upstream, k = 4),
    family = "Gamma",
    fit_mean = fit_mean_y
  )

fit_mean_x <- old_ts |>
  conditional_mean(turbidity_upstream ~
    s(level_upstream, k = 5) +
    s(temperature_upstream, k = 5))

fit_var_x <- old_ts |>
  conditional_var(
    turbidity_upstream ~
      s(level_upstream, k = 4) +
      s(temperature_upstream, k = 4),
    family = "Gamma",
    fit_mean = fit_mean_x
  )

fit_c_ccf <- old_ts |>
  tidyr::drop_na() |>
  conditional_ccf(
    I(turbidity_upstream * turbidity_downstream) ~
      splines::ns(level_upstream, df = 3) +
      splines::ns(temperature_upstream, df = 3),
    lag_max = 10,
    fit_mean_x = fit_mean_x, fit_var_x = fit_var_x,
    fit_mean_y = fit_mean_y, fit_var_y = fit_var_y,
    df_correlation = c(3, 3)
  )

new_data <- fit_c_ccf |> estimate_dt()
```

NEON_PRIN_5min_cleaned

Anomaly removed data for water quality variables aggregated at 5-minute intervals from Pringle Creek, Texas.

Description

NEON_PRIN_5min_cleaned consists anomaly removed data for water quality variables from upstream and downstream sensors in Pringle Creek in Texas for the period spanning from 2019-07-01 to 2019-12-31 aggregated at 5-minute intervals.

Usage

```
NEON_PRIN_5min_cleaned
```

Format

A data frame with water-quality variables, level and temperature data:

`Timestamp` Timestamp

`site` site position

`conductance` specific conductance

`dissolvedOxygen` dissolved oxygen

`pH` pH

`chlorophyll` chlorophyll

`turbidity` turbidity

`fDOM` fDOM

`level` elevation of surface water

`temperature` temperature in surface water

normalize

Normalize a series using conditional moments

Description

This function produces a normalized series using conditional moments.

Usage

```
normalize(data, y, fit_mean, fit_var)
```

Arguments

data	a tsibble containing all the time series which are uniquely identified by the corresponding Timestamp.
y	The variable name
fit_mean	Model object of class "conditional_moment" returned from conditional_mean with information to append to observations.
fit_var	Model object of class "conditional_moment" returned from conditional_var with information to append to observations.

Value

A vector of conditional normliased series

Examples

```
data <- NEON_PRIN_5min_cleaned |>
  dplyr::filter(site == "upstream") |>
  dplyr::select(Timestamp, turbidity, level, conductance, temperature) |>
  tsibble::as_tsibble(index = Timestamp)

fit_mean <- data |>
  conditional_mean(turbidity ~ s(level, k = 8) +
    s(conductance, k = 8) + s(temperature, k = 8))

fit_var <- data |>
  conditional_var(
    turbidity ~ s(level, k = 7) + s(conductance, k = 7) + s(temperature, k = 7),
    family = "Gamma",
    fit_mean = fit_mean
  )

new_ts <- data |>
  dplyr::mutate(ystar = conduits::normalize(data, turbidity, fit_mean, fit_var))
```

---

unnormalize *Unnormalize a series using conditional moments*

Description

This function produces an unnormalized series using conditional moments.

Usage

```
unnormalize(data, ystar, fit_mean, fit_var)
```

Arguments

data	a tsibble containing all the time series which are uniquely identified by the corresponding Timestamp.
ystar	The normalized variable name
fit_mean	Model object of class "conditional_moment" returned from conditional_mean with information to append to observations.
fit_var	Model object of class "conditional_moment" returned from conditional_var with information to append to observations.

Value

A tsibble with the conditional normliased series

Examples

```
data <- NEON_PRIN_5min_cleaned |>
  dplyr::filter(site == "upstream") |>
  dplyr::select(Timestamp, turbidity, level, conductance, temperature) |>
  tsibble::as_tsibble(index = Timestamp)

fit_mean <- data |>
  conditional_mean(turbidity ~ s(level, k = 8) +
    s(conductance, k = 8) + s(temperature, k = 8))

fit_var <- data |>
  conditional_var(
    turbidity ~ s(level, k = 7) + s(conductance, k = 7) + s(temperature, k = 7),
    family = "Gamma",
    fit_mean = fit_mean
  )

new_ts <- data |>
  dplyr::mutate(ystar = normalize(data, turbidity, fit_mean, fit_var))

# For demonstrative purposes, declare three data points
# as missing values.
new_ts[3:5, 6] <- NA

## Not run:
library(fable)
library(dplyr)
impute_ts <- new_ts |>
  model(ARIMA(ystar)) |>
  interpolate(new_ts) |>
  rename(y_star_impt = ystar) |>
  full_join(new_ts, by = "Timestamp")
impute_ts <- impute_ts
  mutate(y = unnormalize(impute_ts, y_star_impt, fit_mean, fit_var))

## End(Not run)
```

Index

∗ **datasets**
NEON_PRIN_5min_cleaned, 16

augment.conditional_acf, 2
augment.conditional_ccf, 3
augment.conditional_moment, 5

calc_dt_CI, 6
conditional_acf, *2*, 7
conditional_ccf, *3*, *6*, 9, *14*
conditional_mean, *5*, *8*, *10*, 11, *13*, *17, 18*
conditional_var, *5*, *8*, *10*, 12, *17, 18*
conduits, 14

estimate_dt, 14

formula.gam, *8, 9*, *12*

gam, *5*, *12, 13*
gamObject, *12, 13*
glm, *8*, *10*

NEON_PRIN_5min_cleaned, 16
normalize, 16
ns, *13*

tibble, *2*, *3*, *5, 6*, *14*
tsibble, *18*

unnormalize, 17