

Package ‘stR’

July 23, 2025

Title Seasonal Trend Decomposition Using Regression

Version 0.7

Description Methods for decomposing seasonal data: STR (a Seasonal-Trend time series decomposition procedure based on Regression) and Robust STR. In some ways, STR is similar to Ridge Regression and Robust STR can be related to LASSO. They allow for multiple seasonal components, multiple linear covariates with constant, flexible and seasonal influence. Seasonal patterns (for both seasonal components and seasonal covariates) can be fractional and flexible over time; moreover they can be either strictly periodic or have a more complex topology. The methods provide confidence intervals for the estimated components. The methods can also be used for forecasting.

Depends R (>= 3.5.0)

Imports compiler, foreach, forecast, graphics, grDevices, Matrix, methods, quantreg, SparseM, stats

Suggests demography, doParallel, knitr, markdown, rgl, rmarkdown, seasonal, testthat

URL <https://pkg.robjhyndman.com/stR/>,
<https://github.com/robjhyndman/stR>

BugReports <https://github.com/robjhyndman/stR/issues>

License GPL-3

LazyData true

Encoding UTF-8

Config/testthat/edition 3

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Alexander Dokumentov [aut] (ORCID:
<<https://orcid.org/0000-0003-0478-0983>>),
Rob Hyndman [aut, cre] (ORCID: <<https://orcid.org/0000-0002-2140-5352>>)

Maintainer Rob Hyndman <Rob.Hyndman@monash.edu>

Repository CRAN
Date/Publication 2024-07-28 13:30:01 UTC

Contents

AutoSTR	2
calls	4
components	5
electricity	5
grocery	6
heuristicSTR	7
plot.STR	13
plotBeta	15
RSTRmodel	17
seasadj.STR	21
STR	22
STRmodel	27

Index	31
--------------	-----------

AutoSTR	<i>Automatic STR decomposition for time series data</i>
---------	---

Description

Automatically selects parameters for an STR decomposition of time series data. The time series should be of class `ts` or `msts`.

Usage

```
AutoSTR(  
  data,  
  robust = FALSE,  
  gapCV = NULL,  
  lambdas = NULL,  
  reltol = 0.001,  
  confidence = NULL,  
  nsKnots = NULL,  
  trace = FALSE  
)
```

Arguments

<code>data</code>	A time series of class <code>ts</code> or <code>msts</code> .
<code>robust</code>	When TRUE, Robust STR decomposition is used. Default is FALSE.
<code>gapCV</code>	An optional parameter defining the length of the sequence of skipped values in the cross validation procedure.

lambdas	An optional parameter. A structure which replaces lambda parameters provided with predictors. It is used as either a starting point for the optimisation of parameters or as the exact model parameters.
reltol	An optional parameter which is passed directly to <code>optim()</code> when optimising the parameters of the model.
confidence	A vector of percentiles giving the coverage of confidence intervals. It must be greater than 0 and less than 1. If NULL, no confidence intervals are produced.
nsKnots	An optional vector parameter, defining the number of seasonal knots (per period) for each seasonal component.
trace	When TRUE, tracing is turned on.

Value

A structure containing input and output data. It is an **S3** class STR, which is a list with the following components:

- **output** – contains decomposed data. It is a list of three components:
 - **predictors** – a list of components where each component corresponds to the input predictor. Every such component is a list containing the following:
 - * **data** – fit/forecast for the corresponding predictor (trend, seasonal component, flexible or seasonal predictor).
 - * **beta** – beta coefficients of the fit of the corresponding predictor.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
 - **random** – a list with one component **data**, which contains residuals of the model fit.
 - **forecast** – a list with two components:
 - * **data** – fit/forecast for the model.
 - * **beta** – beta coefficients of the fit.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
- **input** – input parameters and lambdas used for final calculations.
 - **data** – input data.
 - **predictors** – input predictors.
 - **lambdas** – smoothing parameters used for final calculations (same as input lambdas for STR method).
- **cvMSE** – optional cross validated (leave one out) Mean Squared Error.
- **optim.CV.MSE** – best cross validated Mean Squared Error (n-fold) achieved during minimisation procedure.
- **nFold** – the input nFold parameter.
- **gapCV** – the input gapCV parameter.
- **method** – always contains string "AutoSTR" for this function.

Author(s)

Alexander Dokumentov

References

Dokumentov, A., and Hyndman, R.J. (2022) STR: Seasonal-Trend decomposition using Regression, *INFORMS Journal on Data Science*, 1(1), 50-62. <https://robjhyndman.com/publications/str/>

See Also

[STR](#)

Examples

```
# Decomposition of a multiple seasonal time series
decomp <- AutoSTR(calls)
plot(decomp)

# Decomposition of a monthly time series
decomp <- AutoSTR(log(grocery))
plot(decomp)
```

calls	<i>Number of phone calls dataset</i>
-------	--------------------------------------

Description

Number of call arrivals per 5-minute interval handled on weekdays between 7:00 am and 9:05 pm from March 3, 2003 in a large North American commercial bank.

Usage

```
calls
```

Format

A numerical time series of class `msts` and `ts`.

Source

[Data file](#)

References

Forecasting time series with complex seasonal patterns using exponential smoothing A.M. De Livera, R.J. Hyndman & R.D. Snyder J American Statistical Association, 106(496), 1513-1527. <https://robjhyndman.com/publications/complex-seasonality/>

Examples

```
plot(calls, ylab = "Calls handled")
```

components	<i>Extract STR components</i>
------------	-------------------------------

Description

components extracts components as time series from the result of an STR decomposition.

Usage

```
components(object)
```

Arguments

object Result of STR decomposition.

Author(s)

Alexander Dokumentov

See Also

[STRmodel](#), [RSTRmodel](#), [STR](#), [AutoSTR](#)

Examples

```
fit <- AutoSTR(log(grocery))
comp <- components(fit)
plot(comp)
```

electricity	<i>Electricity consumption dataset</i>
-------------	--

Description

The data set provides information about electricity consumption in Victoria, Australia during the 115 days starting on 10th of January, 2000, and comprises the maximum electricity demand in Victoria during 30-minute periods (48 observations per day). For each 30-minute period, the dataset also provides the air temperature in Melbourne.

Usage

```
electricity
```

Format

An numerical matrix of class `msts` and `ts`.

Details

- Consumption column contains maximum electricity consumption during 30 minute periods
- Temperature column contains temperature in Melbourne during the corresponding 30 minute interval
- Time column contains number of 30 minute interval in the dataset
- DailySeasonality column contains positions of 30 minute interval inside days
- WeeklySeasonality column contains positions of 30 minute interval inside weeks
- WorkingDaySeasonality column contains positions of 30 minute intervals inside working day/holiday transition diagram

Examples

```
plot(electricity[, 1:2],
     xlab = "Weeks",
     main = "Electricity demand and temperature in Melbourne, Australia"
)
```

grocery	<i>Grocery and supermarkets turnover</i>
---------	--

Description

Turnover of supermarkets and grocery stores in New South Wales, Australia.

Usage

grocery

Format

An object of class ts.

References

Australian Bureau of Statistics, CAT 8501.0. (TABLE 11. Retail Turnover, State by Industry Subgroup, Original)

Examples

```
plot(grocery, ylab = "NSW Grocery, $ 10^6")
```

heuristicSTR

Automatic STR decomposition with heuristic search of the parameters

Description

Automatically selects parameters (lambda coefficients) for an STR decomposition of time series data. Heuristic approach can give a better estimate compare to a standard optimisation methods used in [STR](#).

If a parallel backend is registered for use before STR call, heuristicSTR will use it for n-fold cross validation computations.

Usage

```
heuristicSTR(
  data,
  predictors,
  confidence = NULL,
  lambdas = NULL,
  pattern = extractPattern(predictors),
  nFold = 5,
  reltol = 0.005,
  gapCV = 1,
  solver = c("Matrix", "cholesky"),
  trace = FALSE,
  ratioGap = 1e+12,
  relCV = 0.01
)
```

Arguments

data	Time series or a vector of length L .
predictors	<p>List of predictors.</p> <p>According to the paradigm of this implementation, the trend, the seasonal components, the flexible predictors and the seasonal predictors are all presented in the same form (as predictors) and must be described in this list.</p> <p>Every predictor is a list of the following structures:</p>

- **data** – vector of length **L** (length of input data, see above). For trend or for a seasonal component it is a vector of ones. For a flexible or a seasonal predictor it is a vector of the predictor's data.
- **times** – vector of length **L** of times of observations.
- **seasons** – vector of length **L**. It is a vector of ones for a trend or a flexible predictor. It is vector assigning seasons to every observation (for a seasonal component or a seasonal predictor). Seasons can be fractional for observations in between seasons.

- **timeKnots** – vector of times (time knots) where knots are positioned (for a seasonal component or a seasonal predictor a few knots have the same time; every knot is represented by time and season). Usually this vector coincides with **times** vector described above, or **timeKnots** is a subset of **times** vector.
- **seasonalStructure** – describes seasonal topology (which can have complex structure) and seasonal knots. The seasonal topology is described by a list of segments and seasonal knots, which are positioned inside the segments, on borders of the segments or, when they are on borders, they can connect two or more segments.

seasonalStructure is a list of two elements:

- **segments** – a list of vectors representing segments. Each vector must contain two ordered real values which represent left and right borders of a segment. Segments should not intersect (inside same predictor).
- **sKnots** – a list of real values (vectors of length one) or vectors of lengths two or greater (seasonal knots) defining seasons of the knots (every knot is represented by time and season). All real values must belong (be inside or on border of) segments listed in **segments**. If a few values represent a single seasonal knot then all these values must be on borders of some segments (or a single segment). In this case they represent a seasonal knot which connects a few segments (or both sides of one segment).

- **lambdas** – a vector with three values representing lambda (smoothing) parameters (time-time, season-season, time-season flexibility parameters) for this predictor.

confidence	A vector of percentiles giving the coverage of confidence intervals. It must be greater than 0 and less than 1. If NULL, no confidence intervals are produced.
lambdas	An optional parameter. A structure which replaces lambda parameters provided with predictors. It is used as either a starting point for the optimisation of parameters or as the exact model parameters.
pattern	An optional parameter which has the same structure as lambdas although with a different meaning. All zero values correspond to lambda (smoothing) parameters which will not be estimated.
nFold	An optional parameter setting the number of folds for cross validation.
reltol	An optional parameter which is passed directly to <code>optim()</code> when optimising the parameters of the model.
gapCV	An optional parameter defining the length of the sequence of skipped values in the cross validation procedure.
solver	A vector with two string values. The only supported combinations are: <code>c("Matrix", "cholesky")</code> (default), and <code>c("Matrix", "qr")</code> . The parameter is used to specify a particular library and method to solve the minimisation problem during STR decomposition.
trace	When TRUE, tracing is turned on.
ratioGap	Ratio to define hyperparameter bounds for one-dimensional search.

relCV Minimum improvement required after all predictors tried. It is used to exit heuristic search of lambda parameters.

Value

A structure containing input and output data. It is an **S3** class STR, which is a list with the following components:

- **output** – contains decomposed data. It is a list of three components:
 - **predictors** – a list of components where each component corresponds to the input predictor. Every such component is a list containing the following:
 - * **data** – fit/forecast for the corresponding predictor (trend, seasonal component, flexible or seasonal predictor).
 - * **beta** – beta coefficients of the fit of the corresponding predictor.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
 - **random** – a list with one component **data**, which contains residuals of the model fit.
 - **forecast** – a list with two components:
 - * **data** – fit/forecast for the model.
 - * **beta** – beta coefficients of the fit.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
- **input** – input parameters and lambdas used for final calculations.
 - **data** – input data.
 - **predictors** – input predictors.
 - **lambdas** – smoothing parameters used for final calculations (same as input lambdas for STR method).
- **cvMSE** – optional cross validated (leave one out) Mean Squared Error.
- **optim.CV.MSE** or **optim.CV.MAE** – best cross validated Mean Squared Error or Mean Absolute Error (n-fold) achieved during minimisation procedure.
- **nFold** – the input nFold parameter.
- **gapCV** – the input gapCV parameter.
- **method** – contains strings "STR" or "RSTR" depending on used method.

Author(s)

Alexander Dokumentov

References

Dokumentov, A., and Hyndman, R.J. (2022) STR: Seasonal-Trend decomposition using Regression, *INFORMS Journal on Data Science*, 1(1), 50-62. <https://robjhyndman.com/publications/str/>

See Also

[STR STRmodel AutoSTR](#)

Examples

```
TrendSeasonalStructure <- list(
  segments = list(c(0, 1)),
  sKnots = list(c(1, 0))
)
WDSeasonalStructure <- list(
  segments = list(c(0, 48), c(100, 148)),
  sKnots = c(as.list(c(1:47, 101:147)), list(c(0, 48, 100, 148)))
)

TrendSeasons <- rep(1, nrow(electricity))
WDSeasons <- as.vector(electricity[, "WorkingDaySeasonality"])

Data <- as.vector(electricity[, "Consumption"])
Times <- as.vector(electricity[, "Time"])
TempM <- as.vector(electricity[, "Temperature"])
TempM2 <- TempM^2

TrendTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 116)
SeasonTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 24)

TrendData <- rep(1, length(Times))
SeasonData <- rep(1, length(Times))

Trend <- list(
  name = "Trend",
  data = TrendData,
  times = Times,
  seasons = TrendSeasons,
  timeKnots = TrendTimeKnots,
  seasonalStructure = TrendSeasonalStructure,
  lambdas = c(1500, 0, 0)
)
WDSeason <- list(
  name = "Dayly seas",
  data = SeasonData,
  times = Times,
  seasons = WDSeasons,
  timeKnots = SeasonTimeKnots,
  seasonalStructure = WDSeasonalStructure,
  lambdas = c(0.003, 0, 240)
)
StaticTempM <- list(
  name = "Temp M1",
  data = TempM,
  times = Times,
  seasons = NULL,
  timeKnots = NULL,
```

```

    seasonalStructure = NULL,
    lambdas = c(0, 0, 0)
  )
  StaticTempM2 <- list(
    name = "Temp Me1^2",
    data = TempM2,
    times = Times,
    seasons = NULL,
    timeKnots = NULL,
    seasonalStructure = NULL,
    lambdas = c(0, 0, 0)
  )
  Predictors <- list(Trend, WDSeason, StaticTempM, StaticTempM2)

  elec.fit <- heuristicSTR(
    data = Data,
    predictors = Predictors,
    gapCV = 48 * 7
  )

  plot(elec.fit,
    xTime = as.Date("2000-01-11") + ((Times - 1) / 48 - 10),
    forecastPanels = NULL
  )

#####

TrendSeasonalStructure <- list(
  segments = list(c(0, 1)),
  sKnots = list(c(1, 0))
)
DailySeasonalStructure <- list(
  segments = list(c(0, 48)),
  sKnots = c(as.list(1:47), list(c(48, 0)))
)
WeeklySeasonalStructure <- list(
  segments = list(c(0, 336)),
  sKnots = c(as.list(seq(4, 332, 4)), list(c(336, 0)))
)
WDSeasonalStructure <- list(
  segments = list(c(0, 48), c(100, 148)),
  sKnots = c(as.list(c(1:47, 101:147)), list(c(0, 48, 100, 148)))
)

TrendSeasons <- rep(1, nrow(electricity))
DailySeasons <- as.vector(electricity[, "DailySeasonality"])
WeeklySeasons <- as.vector(electricity[, "WeeklySeasonality"])
WDSeasons <- as.vector(electricity[, "WorkingDaySeasonality"])

Data <- as.vector(electricity[, "Consumption"])
Times <- as.vector(electricity[, "Time"])
TempM <- as.vector(electricity[, "Temperature"])
TempM2 <- TempM^2

```

```

TrendTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 116)
SeasonTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 24)
SeasonTimeKnots2 <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 12)

TrendData <- rep(1, length(Times))
SeasonData <- rep(1, length(Times))

Trend <- list(
  name = "Trend",
  data = TrendData,
  times = Times,
  seasons = TrendSeasons,
  timeKnots = TrendTimeKnots,
  seasonalStructure = TrendSeasonalStructure,
  lambdas = c(1500, 0, 0)
)
WSeason <- list(
  name = "Weekly seas",
  data = SeasonData,
  times = Times,
  seasons = WeeklySeasons,
  timeKnots = SeasonTimeKnots2,
  seasonalStructure = WeeklySeasonalStructure,
  lambdas = c(0.8, 0.6, 100)
)
WDSeason <- list(
  name = "Dayly seas",
  data = SeasonData,
  times = Times,
  seasons = WDSeasons,
  timeKnots = SeasonTimeKnots,
  seasonalStructure = WDSeasonalStructure,
  lambdas = c(0.003, 0, 240)
)
TrendTempM <- list(
  name = "Trend temp Mel",
  data = TempM,
  times = Times,
  seasons = TrendSeasons,
  timeKnots = TrendTimeKnots,
  seasonalStructure = TrendSeasonalStructure,
  lambdas = c(1e7, 0, 0)
)
TrendTempM2 <- list(
  name = "Trend temp Mel^2",
  data = TempM2,
  times = Times,
  seasons = TrendSeasons,
  timeKnots = TrendTimeKnots,
  seasonalStructure = TrendSeasonalStructure,
  lambdas = c(0.01, 0, 0)
) # Starting parameter is too far from the optimal value

```

```

Predictors <- list(Trend, WSeason, WDSeason, TrendTempM, TrendTempM2)

elec.fit <- heuristicSTR(
  data = Data,
  predictors = Predictors,
  gapCV = 48 * 7
)

plot(elec.fit,
  xTime = as.Date("2000-01-11") + ((Times - 1) / 48 - 10),
  forecastPanels = NULL
)

plotBeta(elec.fit, predictorN = 4)
plotBeta(elec.fit, predictorN = 5)

```

plot.STR

Plots the results of decomposition

Description

plot.STR plots results of STR decomposition.

Usage

```

## S3 method for class 'STR'
plot(
  x,
  xTime = NULL,
  dataPanels = 1,
  predictorPanels = as.list(seq_along(x$output$predictors)),
  randomPanels = length(x$output$predictors) + 1,
  forecastPanels = length(x$output$predictors) + 2,
  dataColor = "black",
  predictorColors = rep("red", length(x$output$predictors)),
  randomColor = "red",
  forecastColor = "blue",
  vLines = NULL,
  xlab = "Time",
  main = ifelse(x$method %in% c("STR", "STRmodel"), "STR decomposition",
    "Robust STR decomposition"),
  showLegend = TRUE,
  ...
)

```

Arguments

<code>x</code>	Result of STR decomposition.
<code>xTime</code>	Times for data to plot.
<code>dataPanels</code>	Vector of panel numbers in which to plot the original data. Set to NULL to not show data.
<code>predictorPanels</code>	A list of vectors of numbers where every such vector describes which panels should be used for plotting the corresponding predictor.
<code>randomPanels</code>	Vector of panel numbers in which to plot the residuals. Set to NULL to not show residuals.
<code>forecastPanels</code>	Vector of panel numbers in which to plot the fit/forecast. Set to NULL to not show forecasts.
<code>dataColor</code>	Color to plot data.
<code>predictorColors</code>	Vector of colors to plot components corresponding to the predictors.
<code>randomColor</code>	Color to plot the residuals.
<code>forecastColor</code>	Color to plot the fit/forecast.
<code>vLines</code>	Vector of times where vertical lines will be plotted.
<code>xlab</code>	Label for horizontal axis.
<code>main</code>	Main heading for plot.
<code>showLegend</code>	When TRUE (default) legend is shown at top of plot.
<code>...</code>	Other parameters to be passed directly to plot and lines functions in the implementation.

Author(s)

Alexander Dokumentov

See Also

[STRmodel](#), [RSTRmodel](#), [STR](#), [AutoSTR](#)

Examples

```
fit <- AutoSTR(log(grocery))
plot(fit, forecastPanels = 0, randomColor = "DarkGreen", vLines = 2000:2010, lwd = 2)
```

plotBeta*Plots the varying beta coefficients of decomposition*

Description

plotBeta plots the varying beta coefficients of STR decomposition. It plots coefficients only for independent seasons (one less season than defined).

Usage

```
plotBeta(  
  x,  
  xTime = NULL,  
  predictorN = 1,  
  dim = c(1, 2, 3),  
  type = "o",  
  pch = 20,  
  palette = function(n) rainbow(n, start = 0, end = 0.7)  
)
```

Arguments

x	Result of STR decomposition.
xTime	Times for data to plot.
predictorN	Predictor number in the decomposition to plot the corresponding beta coefficients.
dim	Dimensions to use to plot the beta coefficients. When 1, the standard charts are used. When 2, <code>graphics::filled.contour</code> function is used. When 3, <code>rgl::persp3d</code> is used. The default value is 1.
type	Type of the graph for one dimensional plots.
pch	Symbol code to plot points in 1-dimensional charts. Default value is 20.
palette	Color palette for 2 - and 3 - dimensional plots.

Author(s)

Alexander Dokumentov

See Also

[plot.STR](#)

Examples

```

fit <- AutoSTR(log(grocery))
for (i in 1:2) plotBeta(fit, predictorN = i, dim = 2)

#####

TrendSeasonalStructure <- list(
  segments = list(c(0, 1)),
  sKnots = list(c(1, 0))
)
DailySeasonalStructure <- list(
  segments = list(c(0, 48)),
  sKnots = c(as.list(1:47), list(c(48, 0)))
)
WeeklySeasonalStructure <- list(
  segments = list(c(0, 336)),
  sKnots = c(as.list(seq(4, 332, 4)), list(c(336, 0)))
)
WDSeasonalStructure <- list(
  segments = list(c(0, 48), c(100, 148)),
  sKnots = c(as.list(c(1:47, 101:147)), list(c(0, 48, 100, 148)))
)

TrendSeasons <- rep(1, nrow(electricity))
DailySeasons <- as.vector(electricity[, "DailySeasonality"])
WeeklySeasons <- as.vector(electricity[, "WeeklySeasonality"])
WDSeasons <- as.vector(electricity[, "WorkingDaySeasonality"])

Data <- as.vector(electricity[, "Consumption"])
Times <- as.vector(electricity[, "Time"])
TempM <- as.vector(electricity[, "Temperature"])
TempM2 <- TempM^2

TrendTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 116)
SeasonTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 24)
SeasonTimeKnots2 <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 12)

TrendData <- rep(1, length(Times))
SeasonData <- rep(1, length(Times))

Trend <- list(
  name = "Trend",
  data = TrendData,
  times = Times,
  seasons = TrendSeasons,
  timeKnots = TrendTimeKnots,
  seasonalStructure = TrendSeasonalStructure,
  lambdas = c(1500, 0, 0)
)
WSeason <- list(
  name = "Weekly seas",
  data = SeasonData,

```



```

    times = Times,
    seasons = WeeklySeasons,
    timeKnots = SeasonTimeKnots2,
    seasonalStructure = WeeklySeasonalStructure,
    lambdas = c(0.8, 0.6, 100)
  )
  WSeason <- list(
    name = "Dayly seas",
    data = SeasonData,
    times = Times,
    seasons = WSeasons,
    timeKnots = SeasonTimeKnots,
    seasonalStructure = WSeasonalStructure,
    lambdas = c(0.003, 0, 240)
  )
  TrendTempM <- list(
    name = "Trend temp Mel",
    data = TempM,
    times = Times,
    seasons = TrendSeasons,
    timeKnots = TrendTimeKnots,
    seasonalStructure = TrendSeasonalStructure,
    lambdas = c(1e7, 0, 0)
  )
  TrendTempM2 <- list(
    name = "Trend temp Mel^2",
    data = TempM2,
    times = Times,
    seasons = TrendSeasons,
    timeKnots = TrendTimeKnots,
    seasonalStructure = TrendSeasonalStructure,
    lambdas = c(0.01, 0, 0)
  ) # Starting parameter is too far from the optimal value
  Predictors <- list(Trend, WSeason, WSeason, TrendTempM, TrendTempM2)

  elec.fit <- STR(
    data = Data,
    predictors = Predictors,
    gapCV = 48 * 7
  )

  plot(elec.fit,
    xTime = as.Date("2000-01-11") + ((Times - 1) / 48 - 10),
    forecastPanels = NULL
  )

  plotBeta(elec.fit, predictorN = 4)
  plotBeta(elec.fit, predictorN = 5) # Beta coefficients are too "wiggly"

```

Description

Robust Seasonal-Trend decomposition of time series data using Regression (robust version of [STRmodel](#)).

Usage

```
RSTRmodel(
  data,
  predictors = NULL,
  strDesign = NULL,
  lambdas = NULL,
  confidence = NULL,
  nMCIter = 100,
  control = list(nnzlmax = 1e+06, nsubmax = 3e+05, tmpmax = 50000),
  reportDimensionsOnly = FALSE,
  trace = FALSE
)
```

Arguments

<code>data</code>	Time series or a vector of length L .
<code>predictors</code>	List of predictors. According to the paradigm of this implementation, the trend, the seasonal components, the flexible predictors and the seasonal predictors are all presented in the same form (as predictors) and must be described in this list. Every predictor is a list of the following structures:

- **data** – vector of length **L** (length of input data, see above). For trend or for a seasonal component it is a vector of ones. For a flexible or a seasonal predictor it is a vector of the predictor's data.
- **times** – vector of length **L** of times of observations.
- **seasons** – vector of length **L**. It is a vector of ones for a trend or a flexible predictor. It is vector assigning seasons to every observation (for a seasonal component or a seasonal predictor). Seasons can be fractional for observations in between seasons.
- **timeKnots** – vector of times (time knots) where knots are positioned (for a seasonal component or a seasonal predictor a few knots have the same time; every knot is represented by time and season). Usually this vector coincides with **times** vector described above, or **timeKnots** is a subset of **times** vector.
- **seasonalStructure** – describes seasonal topology (which can have complex structure) and seasonal knots. The seasonal topology is described by a list of segments and seasonal knots, which are positioned inside the segments, on borders of the segments or, when they are on on borders, they can connect two or more segments.
seasonalStructure is a list of two elements:

	<ul style="list-style-type: none"> – segments – a list of vectors representing segments. Each vector must contain two ordered real values which represent left and right borders of a segment. Segments should not intersect (inside same predictor). – sKnots – a list of real values (vectors of length one) or vectors of lengths two or greater (seasonal knots) defining seasons of the knots (every knot is represented by time and season). All real values must belong (be inside or on border of) segments listed in segments. If a few values represent a single seasonal knot then all these values must be on borders of some segments (or a single segment). In this case they represent a seasonal knot which connects a few segments (or both sides of one segment). • lambdas – a vector with three values representing lambda (smoothing) parameters (time-time, season-season, time-season flexibility parameters) for this predictor.
strDesign	An optional parameter used to create the design matrix. It is used internally in the library to improve performance when the design matrix does not require full recalculation.
lambdas	An optional parameter. A structure which replaces lambda parameters provided with predictors. It is used as either a starting point for the optimisation of parameters or as the exact model parameters.
confidence	A vector of percentiles giving the coverage of confidence intervals. It must be greater than 0 and less than 1. If NULL, no confidence intervals are produced.
nMCIter	Number of Monte Carlo iterations used to estimate confidence intervals for Robust STR decomposition.
control	Passed directly to <code>rq.fit.sfn()</code> during Robust STR decomposition.
reportDimensionsOnly	A boolean parameter. When TRUE the method constructs the design matrix and reports its dimensions without proceeding further. It is mostly used for debugging.
trace	When TRUE, tracing is turned on.

Value

A structure containing input and output data. It is an **S3** class STR, which is a list with the following components:

- **output** – contains decomposed data. It is a list of three components:
 - **predictors** – a list of components where each component corresponds to the input predictor. Every such component is a list containing the following:
 - * **data** – fit/forecast for the corresponding predictor (trend, seasonal component, flexible or seasonal predictor).
 - * **beta** – beta coefficients of the fit of the corresponding predictor.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
 - **random** – a list with one component **data**, which contains residuals of the model fit.
 - **forecast** – a list with two components:

- * **data** – fit/forecast for the model.
- * **beta** – beta coefficients of the fit.
- * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
- * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
- **input** – input parameters and lambdas used for final calculations.
 - **data** – input data.
 - **predictors** - input predictors.
 - **lambdas** – smoothing parameters used for final calculations (same as input lambdas for STR method).
- **method** – always contains string "RSTRmodel" for this function.

Author(s)

Alexander Dokumentov

References

Dokumentov, A., and Hyndman, R.J. (2022) STR: Seasonal-Trend decomposition using Regression, *INFORMS Journal on Data Science*, 1(1), 50-62. <https://robjhyndman.com/publications/str/>

See Also

[STRmodel STR](#)

Examples

```
n <- 70
trendSeasonalStructure <- list(segments = list(c(0, 1)), sKnots = list(c(1, 0)))
ns <- 5
seasonalStructure <- list(
  segments = list(c(0, ns)),
  sKnots = c(as.list(1:(ns - 1)), list(c(ns, 0)))
)
seasons <- (0:(n - 1)) %% ns + 1
trendSeasons <- rep(1, length(seasons))
times <- seq_along(seasons)
data <- seasons + times / 4
set.seed(1234567890)
data <- data + rnorm(length(data), 0, 0.2)
data[20] <- data[20] + 3
data[50] <- data[50] - 5
plot(times, data, type = "l")
timeKnots <- times
trendData <- rep(1, n)
seasonData <- rep(1, n)
trend <- list(
  data = trendData, times = times, seasons = trendSeasons,
  timeKnots = timeKnots, seasonalStructure = trendSeasonalStructure, lambdas = c(1, 0, 0)
)
```

```

season <- list(
  data = seasonData, times = times, seasons = seasons,
  timeKnots = timeKnots, seasonalStructure = seasonalStructure, lambdas = c(1, 0, 1)
)
predictors <- list(trend, season)
rstr <- RSTRmodel(data, predictors, confidence = 0.8)
plot(rstr)

```

seasadj.STR

*Seasonal adjustment based on STR***Description**

seasadj.STR extracts seasonally adjusted data by removing the seasonal components from the result of STR decomposition.

Usage

```

## S3 method for class 'STR'
seasadj(object, include = c("Trend", "Random"), ...)

```

Arguments

object	Result of STR decomposition.
include	Vector of component names to include in the result. The default is c("Trend", "Random").
...	Other arguments not currently used.

Author(s)

Alexander Dokumentov

See Also

[STRmodel](#), [RSTRmodel](#), [STR](#), [AutoSTR](#)

Examples

```

fit <- AutoSTR(log(grocery))
plot(seasadj(fit))

```

STR

*Automatic STR decomposition***Description**

Automatically selects parameters for an STR decomposition of time series data.

If a parallel backend is registered for use before STR call, STR will use it for n-fold cross validation computations.

Usage

```
STR(
  data,
  predictors,
  confidence = NULL,
  robust = FALSE,
  lambdas = NULL,
  pattern = extractPattern(predictors),
  nFold = 5,
  reltol = 0.005,
  gapCV = 1,
  solver = c("Matrix", "cholesky"),
  nMCIter = 100,
  control = list(nnzlmax = 1e+06, nsubmax = 3e+05, tmpmax = 50000),
  trace = FALSE,
  iterControl = list(maxiter = 20, tol = 1e-06)
)
```

Arguments

data	Time series or a vector of length L .
predictors	<p>List of predictors.</p> <p>According to the paradigm of this implementation, the trend, the seasonal components, the flexible predictors and the seasonal predictors are all presented in the same form (as predictors) and must be described in this list.</p> <p>Every predictor is a list of the following structures:</p>

- **data** – vector of length **L** (length of input data, see above). For trend or for a seasonal component it is a vector of ones. For a flexible or a seasonal predictor it is a vector of the predictor's data.
- **times** – vector of length **L** of times of observations.
- **seasons** – vector of length **L**. It is a vector of ones for a trend or a flexible predictor. It is vector assigning seasons to every observation (for a seasonal component or a seasonal predictor). Seasons can be fractional for observations in between seasons.

- **timeKnots** – vector of times (time knots) where knots are positioned (for a seasonal component or a seasonal predictor a few knots have the same time; every knot is represented by time and season). Usually this vector coincides with **times** vector described above, or **timeKnots** is a subset of **times** vector.
- **seasonalStructure** – describes seasonal topology (which can have complex structure) and seasonal knots. The seasonal topology is described by a list of segments and seasonal knots, which are positioned inside the segments, on borders of the segments or, when they are on borders, they can connect two or more segments.

seasonalStructure is a list of two elements:

- **segments** – a list of vectors representing segments. Each vector must contain two ordered real values which represent left and right borders of a segment. Segments should not intersect (inside same predictor).
- **sKnots** – a list of real values (vectors of length one) or vectors of lengths two or greater (seasonal knots) defining seasons of the knots (every knot is represented by time and season). All real values must belong (be inside or on border of) segments listed in **segments**. If a few values represent a single seasonal knot then all these values must be on borders of some segments (or a single segment). In this case they represent a seasonal knot which connects a few segments (or both sides of one segment).

- **lambdas** – a vector with three values representing lambda (smoothing) parameters (time-time, season-season, time-season flexibility parameters) for this predictor.

confidence	A vector of percentiles giving the coverage of confidence intervals. It must be greater than 0 and less than 1. If NULL, no confidence intervals are produced.
robust	When TRUE, Robust STR decomposition is used. Default is FALSE.
lambdas	An optional parameter. A structure which replaces lambda parameters provided with predictors. It is used as either a starting point for the optimisation of parameters or as the exact model parameters.
pattern	An optional parameter which has the same structure as lambdas although with a different meaning. All zero values correspond to lambda (smoothing) parameters which will not be estimated.
nFold	An optional parameter setting the number of folds for cross validation.
reitol	An optional parameter which is passed directly to <code>optim()</code> when optimising the parameters of the model.
gapCV	An optional parameter defining the length of the sequence of skipped values in the cross validation procedure.
solver	A vector with two string values. The only supported combinations are: <code>c("Matrix", "cholesky")</code> (default), and <code>c("Matrix", "qr")</code> . The parameter is used to specify a particular library and method to solve the minimisation problem during STR decomposition.
nMCIter	Number of Monte Carlo iterations used to estimate confidence intervals for Robust STR decomposition.

control	Passed directly to <code>rq.fit.sfn()</code> during Robust STR decomposition.
trace	When TRUE, tracing is turned on.
iterControl	Control parameters for some experimental features. This should not be used by an ordinary user.

Value

A structure containing input and output data. It is an **S3** class STR, which is a list with the following components:

- **output** – contains decomposed data. It is a list of three components:
 - **predictors** – a list of components where each component corresponds to the input predictor. Every such component is a list containing the following:
 - * **data** – fit/forecast for the corresponding predictor (trend, seasonal component, flexible or seasonal predictor).
 - * **beta** – beta coefficients of the fit of the corresponding predictor.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
 - **random** – a list with one component **data**, which contains residuals of the model fit.
 - **forecast** – a list with two components:
 - * **data** – fit/forecast for the model.
 - * **beta** – beta coefficients of the fit.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
- **input** – input parameters and lambdas used for final calculations.
 - **data** – input data.
 - **predictors** – input predictors.
 - **lambdas** – smoothing parameters used for final calculations (same as input lambdas for STR method).
- **cvMSE** – optional cross validated (leave one out) Mean Squared Error.
- **optim.CV.MSE** or **optim.CV.MAE** – best cross validated Mean Squared Error or Mean Absolute Error (n-fold) achieved during minimisation procedure.
- **nFold** – the input nFold parameter.
- **gapCV** – the input gapCV parameter.
- **method** – contains strings "STR" or "RSTR" depending on used method.

Author(s)

Alexander Dokumentov

References

Dokumentov, A., and Hyndman, R.J. (2022) STR: Seasonal-Trend decomposition using Regression, *INFORMS Journal on Data Science*, 1(1), 50-62. <https://robjhyndman.com/publications/str/>

See Also

[STRmodel](#) [RSTRmodel](#) [AutoSTR](#)

Examples

```
TrendSeasonalStructure <- list(
  segments = list(c(0, 1)),
  sKnots = list(c(1, 0))
)
WDSeasonalStructure <- list(
  segments = list(c(0, 48), c(100, 148)),
  sKnots = c(as.list(c(1:47, 101:147)), list(c(0, 48, 100, 148)))
)

TrendSeasons <- rep(1, nrow(electricity))
WDSeasons <- as.vector(electricity[, "WorkingDaySeasonality"])

Data <- as.vector(electricity[, "Consumption"])
Times <- as.vector(electricity[, "Time"])
TempM <- as.vector(electricity[, "Temperature"])
TempM2 <- TempM^2

TrendTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 116)
SeasonTimeKnots <- seq(from = head(Times, 1), to = tail(Times, 1), length.out = 24)

TrendData <- rep(1, length(Times))
SeasonData <- rep(1, length(Times))

Trend <- list(
  name = "Trend",
  data = TrendData,
  times = Times,
  seasons = TrendSeasons,
  timeKnots = TrendTimeKnots,
  seasonalStructure = TrendSeasonalStructure,
  lambdas = c(1500, 0, 0)
)
WDSeason <- list(
  name = "Dayly seas",
  data = SeasonData,
  times = Times,
  seasons = WDSeasons,
  timeKnots = SeasonTimeKnots,
  seasonalStructure = WDSeasonalStructure,
  lambdas = c(0.003, 0, 240)
)
StaticTempM <- list(
  name = "Temp M1",
  data = TempM,
  times = Times,
  seasons = NULL,
  timeKnots = NULL,
```

```

    seasonalStructure = NULL,
    lambdas = c(0, 0, 0)
  )
  StaticTempM2 <- list(
    name = "Temp Me1^2",
    data = TempM2,
    times = Times,
    seasons = NULL,
    timeKnots = NULL,
    seasonalStructure = NULL,
    lambdas = c(0, 0, 0)
  )
  Predictors <- list(Trend, WDSeason, StaticTempM, StaticTempM2)

  elec.fit <- STR(
    data = Data,
    predictors = Predictors,
    gapCV = 48 * 7
  )

  plot(elec.fit,
    xTime = as.Date("2000-01-11") + ((Times - 1) / 48 - 10),
    forecastPanels = NULL
  )

#####

n <- 70
trendSeasonalStructure <- list(segments = list(c(0, 1)), sKnots = list(c(1, 0)))
ns <- 5
seasonalStructure <- list(
  segments = list(c(0, ns)),
  sKnots = c(as.list(1:(ns - 1)), list(c(ns, 0)))
)
seasons <- (0:(n - 1)) % ns + 1
trendSeasons <- rep(1, length(seasons))
times <- seq_along(seasons)
data <- seasons + times / 4
set.seed(1234567890)
data <- data + rnorm(length(data), 0, 0.2)
data[20] <- data[20] + 3
data[50] <- data[50] - 5
plot(times, data, type = "l")
timeKnots <- times
trendData <- rep(1, n)
seasonData <- rep(1, n)
trend <- list(
  data = trendData, times = times, seasons = trendSeasons,
  timeKnots = timeKnots, seasonalStructure = trendSeasonalStructure, lambdas = c(1, 0, 0)
)
season <- list(
  data = seasonData, times = times, seasons = seasons,
  timeKnots = timeKnots, seasonalStructure = seasonalStructure, lambdas = c(1, 0, 1)
)

```

```

)
predictors <- list(trend, season)
rstr <- STR(data, predictors,
  reltol = 0.0000001, gapCV = 10,
  confidence = 0.95, nMCIter = 400, robust = TRUE
)
plot(rstr)

```

STRmodel

STR decomposition

Description

Seasonal-Trend decomposition of time series data using Regression.

Usage

```

STRmodel(
  data,
  predictors = NULL,
  strDesign = NULL,
  lambdas = NULL,
  confidence = NULL,
  solver = c("Matrix", "cholesky"),
  reportDimensionsOnly = FALSE,
  trace = FALSE
)

```

Arguments

<code>data</code>	Time series or a vector of length L .
<code>predictors</code>	<p>List of predictors.</p> <p>According to the paradigm of this implementation, the trend, the seasonal components, the flexible predictors and the seasonal predictors are all presented in the same form (as predictors) and must be described in this list.</p> <p>Every predictor is a list of the following structures:</p>

- **data** – vector of length **L** (length of input data, see above). For trend or for a seasonal component it is a vector of ones. For a flexible or a seasonal predictor it is a vector of the predictor's data.
- **times** – vector of length **L** of times of observations.
- **seasons** – vector of length **L**. It is a vector of ones for a trend or a flexible predictor. It is vector assigning seasons to every observation (for a seasonal component or a seasonal predictor). Seasons can be fractional for observations in between seasons.

- **timeKnots** – vector of times (time knots) where knots are positioned (for a seasonal component or a seasonal predictor a few knots have the same time; every knot is represented by time and season). Usually this vector coincides with **times** vector described above, or **timeKnots** is a subset of **times** vector.
- **seasonalStructure** – describes seasonal topology (which can have complex structure) and seasonal knots. The seasonal topology is described by a list of segments and seasonal knots, which are positioned inside the segments, on borders of the segments or, when they are on borders, they can connect two or more segments.

seasonalStructure is a list of two elements:

- **segments** – a list of vectors representing segments. Each vector must contain two ordered real values which represent left and right borders of a segment. Segments should not intersect (inside same predictor).
- **sKnots** – a list of real values (vectors of length one) or vectors of lengths two or greater (seasonal knots) defining seasons of the knots (every knot is represented by time and season). All real values must belong (be inside or on border of) segments listed in **segments**. If a few values represent a single seasonal knot then all these values must be on borders of some segments (or a single segment). In this case they represent a seasonal knot which connects a few segments (or both sides of one segment).
- **lambdas** – a vector with three values representing lambda (smoothing) parameters (time-time, season-season, time-season flexibility parameters) for this predictor.

strDesign	An optional parameter used to create the design matrix. It is used internally in the library to improve performance when the design matrix does not require full recalculation.
lambdas	An optional parameter. A structure which replaces lambda parameters provided with predictors. It is used as either a starting point for the optimisation of parameters or as the exact model parameters.
confidence	A vector of percentiles giving the coverage of confidence intervals. It must be greater than 0 and less than 1. If NULL, no confidence intervals are produced.
solver	A vector with two string values. The only supported combinations are: c("Matrix", "cholesky") (default), and c("Matrix", "qr"). The parameter is used to specify a particular library and method to solve the minimisation problem during STR decomposition.
reportDimensionsOnly	A boolean parameter. When TRUE the method constructs the design matrix and reports its dimensions without proceeding further. It is mostly used for debugging.
trace	When TRUE, tracing is turned on.

Value

A structure containing input and output data. It is an **S3** class STR, which is a list with the following components:

- **output** – contains decomposed data. It is a list of three components:
 - **predictors** – a list of components where each component corresponds to the input predictor. Every such component is a list containing the following:
 - * **data** – fit/forecast for the corresponding predictor (trend, seasonal component, flexible or seasonal predictor).
 - * **beta** – beta coefficients of the fit of the corresponding predictor.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
 - **random** – a list with one component **data**, which contains residuals of the model fit.
 - **forecast** – a list with two components:
 - * **data** – fit/forecast for the model.
 - * **beta** – beta coefficients of the fit.
 - * **lower** – optional (if requested) matrix of lower bounds of confidence intervals.
 - * **upper** – optional (if requested) matrix of upper bounds of confidence intervals.
- **input** – input parameters and lambdas used for final calculations.
 - **data** – input data.
 - **predictors** – input predictors.
 - **lambdas** – smoothing parameters used for final calculations (same as input lambdas for STR method).
- **cvMSE** – optional cross validated (leave one out) Mean Squared Error.
- **method** – always contains string "STRmodel" for this function.

Author(s)

Alexander Dokumentov

References

Dokumentov, A., and Hyndman, R.J. (2022) STR: Seasonal-Trend decomposition using Regression, *INFORMS Journal on Data Science*, 1(1), 50-62. <https://robjhyndman.com/publications/str/>

See Also

[AutoSTR](#)

Examples

```
n <- 50
trendSeasonalStructure <- list(segments = list(c(0, 1)), sKnots = list(c(1, 0)))
ns <- 5
seasonalStructure <- list(
```

```

    segments = list(c(0, ns)),
    sKnots = c(as.list(1:(ns - 1)), list(c(ns, 0)))
  )
  seasons <- (0:(n - 1)) %% ns + 1
  trendSeasons <- rep(1, length(seasons))
  times <- seq_along(seasons)
  data <- seasons + times / 4
  plot(times, data, type = "l")
  timeKnots <- times
  trendData <- rep(1, n)
  seasonData <- rep(1, n)
  trend <- list(
    data = trendData, times = times, seasons = trendSeasons,
    timeKnots = timeKnots, seasonalStructure = trendSeasonalStructure, lambdas = c(1, 0, 0)
  )
  season <- list(
    data = seasonData, times = times, seasons = seasons,
    timeKnots = timeKnots, seasonalStructure = seasonalStructure, lambdas = c(10, 0, 0)
  )
  predictors <- list(trend, season)

  str1 <- STRmodel(data, predictors)
  plot(str1)

  data[c(3, 4, 7, 20, 24, 29, 35, 37, 45)] <- NA
  plot(times, data, type = "l")
  str2 <- STRmodel(data, predictors)
  plot(str2)

```

Index

* datasets

calls, [4](#)

electricity, [5](#)

grocery, [6](#)

AutoSTR, [2](#), [5](#), [10](#), [14](#), [21](#), [25](#), [29](#)

calls, [4](#)

components, [5](#)

electricity, [5](#)

grocery, [6](#)

heuristicSTR, [7](#)

optim, [3](#), [8](#), [23](#)

plot.STR, [13](#), [15](#)

plotBeta, [15](#)

rq.fit.sfn, [19](#), [24](#)

RSTRmodel, [5](#), [14](#), [17](#), [21](#), [25](#)

seasadj.STR, [21](#)

STR, [4](#), [5](#), [7](#), [10](#), [14](#), [20](#), [21](#), [22](#)

STRmodel, [5](#), [10](#), [14](#), [18](#), [20](#), [21](#), [25](#), [27](#)