Package ‘sTSD’

January 27, 2025

Type Package

Title Simulate Time Series Diagnostics

Version 0.2.0

Maintainer Steven Miller <steve@svmiller.com>

Description These are tools that allow users to do time series diagnostics, primarily tests of unit root, by way of simulation. While there is nothing necessarily wrong with the received wisdom of critical values generated decades ago, simulation provides its own perks. Not only is simulation broadly informative as to what these various test statistics do and what are their plausible values, simulation provides more flexibility for assessing unit root by way of different thresholds or different hypothesized distributions.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0)

RoxygenNote 7.3.2

NeedsCompilation no

Author Steven Miller [aut, cre] (<https://orcid.org/0000-0003-4072-6263>)

Repository CRAN

Date/Publication 2025-01-27 08:50:02 UTC

Contents

`adf_lag_select` *Identify Optimal Lag Order Selection for (Augmented) Dickey-Fuller Tests*

Description

`adf_lag_select()` runs a series of (Augmented) Dickey-Fuller tests and returns information that may (or may not) be useful in identify a potential lag order for unit root tests.

Usage

```
adf_lag_select(x, min_lag = 0, max_lag = NULL)
```

Arguments

`x`	a vector
`min_lag`	the minimum lag order to use. Defaults to 0.
`max_lag`	the maximum lag order to use. Defaults to Schwert's (1989) upper lag.

Details

This function removes missing values from the vector before calculating test statistics.

The lower bound lag order suggested by Schwert (1989) and the default suggested by Said and Dickey (1984) do not meaningfully separate from each other until the length of the series reaches 127. Under those conditions, if the `note` column returned by this function for a finite series does not identify the Said and Dickey (1984) default, but identifies the Schwert (1989) lower bound, interpret the latter as the former.

Value

`adf_lag_select()` returns a list of length 3. The first element in the list is a data frame of a series of (Augmented) Dickey-Fuller tests for no-drift, no-trend. The second is a data frame of a series of (Augmented) Dickey-Fuller tests for drift, no trend. The third is a data frame of a series of (Augmented) Dickey-Fuller tests for a drift and trend. Each data frame has the following columns communicating the following information.

1. The lag order
2. The (A)DF statistic for the lag order.
3. The Akaike information criterion for the model.

4. Schwartz' (Bayesian) criteron for the model.
5. The absolute value of the last lagged first difference in the model.
6. The "modified" Akaike information criterion for the model.
7. The "modified" Schwarz' (Bayesian) criterion for the model.
8. A note indicating if the lag was suggested by Schwert (1989) or Said and Dickey (1984)

Author(s)

Steven V. Miller

Examples

```
x <- head(tbills$tb3m, 500)
adf_lag_select(x)
```

`exCopdab` *Dynamic Foreign Policy Behavior (COPDAB)*

Description

A data frame on monthly dyadic foreign policy behavior from 1948 to 1978 for select dyads, using COPDAB data. The data offer the opportunity for a basic replication of Lebo and Moore (2003).

Usage

```
exCopdab
```

Format

A data frame with 372 observations on the following 12 variables.

`ym` a year-month indicator, in the format of YYMM

`eg2is` an estimate of the dyadic foreign policy behavior of Egypt to Israel

`is2eg` an estimate of the dyadic foreign policy behavior of Israel to Egypt

`us2ussr` an estimate of the dyadic foreign policy behavior of the U.S. to the Soviet Union

`ussr2us` an estimate of the dyadic foreign policy behavior of the Soviet Union to the U.S.

`us2fra` an estimate of the dyadic foreign policy behavior of the U.S. to France

`fra2us` an estimate of the dyadic foreign policy behavior of France to the U.S.

`us2is` an estimate of the dyadic foreign policy behavior of the U.S. to Israel

`is2us` an estimate of the dyadic foreign policy behavior of Israel to the U.S.

`suez` a dummy variable indicating if the observation corresponds with the Suez Crisis

`sixday` a dummy variable indicating if the observation corresponds with the Six-Day War

`yomk` a dummy variable indicating if the observation corresponds with the Yom Kippur War

Details

Lebo and Moore (2003, 22-24) will offer more context about how these variables are coded. Important details for replication from scratch are assuredly lost to history, but the authors are clear about what they're doing and the procedure they used to weight fundamentally ordinal data to create some kind of continuous estimate. Context clues offer more information as well.

References

Lebo, Matthew J. and Will H. Moore. 2003. "Dynamic Foreign Policy Behavior." *Journal of Conflict Resolution* 47(1): 13-32.

`lag_suggests` *Suggested Lags for Your Time Series*

Description

A data frame on various suggestions for lags for your time series, given the length of your time series. You are not compelled to use these. These are just suggestions.

Usage

```
lag_suggests
```

Format

A data frame with 1000 observations on the following 4 variables.

`n` an integer corresponding with an assumed length of your time series

`schwert_ub` the upper bound lag order suggested by Schwert (1989) for a time series of that length

`schwert_lb` the lower bound lag order suggested by Schwert (1989) for a time series of that length

`qiuetal2013` the suggested lag order from Qiu et al. (2013)

`sd84` the suggested lag order from Said and Dickey (1984)

Details

The lower bound lag order suggested by Schwert (1989) and the default suggested by Said and Dickey (1984) do not meaningfully separate from each other until the length of the series reaches 127. You should think long and hard about doing any of this if your time series is so finite that it has fewer than 25 observations.

The Qiu et al. (2013) suggestion is the default lag if you're using the **aTSA** package. It is almost equivalent to the Schwert (1989) lower bound, except the length of the series is raised to 2/9 and not 2/8. The two do not meaningfully separate until the length of the series reaches 5,720 observations (which is when the difference between two reaches two lags of separation).

References

Qiu, D., Q. Shao, and L. Yang. 2013. "Efficient Inference for Autoregressive Coefficients in the Presence of Trends." *Jounal of Multivariate Analysis* 114: 40–53.

Said, Said E. and David A. Dickey. 1984. "Testing for Unit Roots in Autoregressive-Moving Average Models of Unknown Order." *Biometrika* 71(3): 599-607.

Schwert, G. William. 1989. "Tests for Unit Roots: A Monte Carlo Investigation". *Journal of Business & Economic Statistics* 7(2): 147–59.

`money_demand` *Quarterly Money Demand in the United States*

Description

A data frame of quarterly indicators useful for modeling the demand for money in the United States. Data go from the first quarter of 1960 to the third quarter of 2024.

Usage

```
money_demand
```

Format

A data frame with 259 observations on the following 6 variables.

`date` a date

`m1` so-called 'narrow' money (M1) in supply, in billions, not seasonally adjusted

`m2` monetary supply (M2), in billions, not seasonally adjusted

`gnpdef` an implicit price deflator for gross national product (index, 2017 = 100)

`ffer` the federal funds effective rate

`rgnp` real gross national product (in 2017 dollars)

`pcepi` the chain-type price index (index, 2017 == 100)

Details

Data come by way of **fredr** call. Be mindful of changes in the definition of the money supply, especially as they manifest in May 2020. Subject domain expertise goes a long way here. The "M2" indicator is the "M1" indicator with small-time deposits that are "close substitutes" for M1.

sadf_test *Simulate a (Augmented) Dickey-Fuller Test to Assess Unit Root in a Time Series*

Description

`sadf_test()` provides a simulation approach to assessing unit root in a time series by way of the (Augmented) Dickey-Fuller test. It takes a vector and performs three (Augmented) Dickey-Fuller tests (no drift, no trend; drift, no trend; drift and trend) and calculates tau statistics as one normally would. Rather than interpolate or approximate a *p*-value, it simulates some user-specified number of (Augmented) Dickey-Fuller tests of either a known, non-stationary time series or a known, white-noise time series matching the length of the time series the user provides. This allows the user to make assessments of non-stationarity or stationarity by way of simulation rather than approximation from received critical values by way of books or tables some years out of date.

Usage

```
sadf_test(x, n_lags = NULL, n_sims = 1000, sim_hyp = "nonstationary")
```

Arguments

`x`	a vector
`n_lags`	defaults to NULL, but must be 0 or a positive integer. This argument determines the number of lagged first differences to include in the estimation procedure. Recall that the test statistic (tau) is still the t-statistic for the *level* value of the vector at t-1, whether the constant (drift) and time trend is included or not. If this value is 0, the procedure is the classic Dickey-Fuller test. If this value is greater than 0, this is the "augmented" Dickey-Fuller test, so-called because it is "augmented" by the number of lagged first differences to assess higher-order AR processes. If no argument is specified, the default lag is Schwert's suggested lower bound. The `lag_suggests` data provides more information about these suggested lags.
`n_sims`	the number of simulations for calculating an interval or distribution of test statistics for assessing stationarity or non-stationarity. Defaults to 1,000.
`sim_hyp`	can be either "stationary" or "nonstationary". If "stationary", the function runs (A)DF tests on simulated stationary (pure white noise) data. This allows the user to assess compatibility/plausibility of the test statistic against a distribution of test statistics that are known to be pure white noise (in expectation). If "non-stationary" (default), the function generates three different data sets of a pure random walk, a random walk with a drift, and a random walk with a drift and trend. It then runs (A)DF tests on all those. This allows the user to assess the compatibility/plausibility of their test statistics with data that are known to be nonstationary in some form.

Details

The Dickey-Fuller and its "augmented" corollary are curious statistical procedures, even if the underlying concept is straightforward. I have seen various implementations of these procedures use slightly different terminology to describe its procedure, though this particular implementation will impose nomenclature in which the classic Dickey-Fuller procedure that assumes just the AR(1) process is one in which `n_lags` is 0. The addition of lags (of first differences) is what ultimately makes the Dickey-Fuller procedure to be "augmented."

The function employs the default suggested by Schwert (1989) for the number of lagged first differences to include in this procedure. Schwert (1989) recommends taking the length of the series and dividing it by 100 before raising that number to the power of 1/4. Thereafter, multiply it by 12 and round down the number to the nearest integer. There are other suggested defaults you can consider. `adf.test` in **aTSA** takes the length of the series, divides it by 100 and raises it to the power of 2/9. It multiplies that by 4 and floors the result. `adf.test` in **tseries** subtracts 1 from the length of the series before raising it to the power of 1/3 (flooring that result as well). The Examples section will show you how you can do this.

This function specifies three different types of tests: 1) no drift, no trend, 2) drift, no trend, and 3) drift and trend. In the language of the `lm()` function, the first is `lm(y ~ ly - 1)` where `y` is the value of `y` and `ly` is its first-order lag. The second test is `lm(y ~ ly)`, intuitively suggesting the *y*-intercept in this equation is the "drift". The third would be `lm(y ~ ly + t)` with `t` being a simple integer that increases by 1 for each observation (i.e. a time-trend).

None of this is meant to discourage the use of Fuller (1976) or its various reproductions for the sake of diagnosing stationarity or non-stationary, and I will confess their expertise on these matters outpaces mine. Consider the justification for this function to be largely philosophical and/or experimental. Why not simulate it? It's not like time or computing power are huge issues anymore.

This is always awkwardly stated, but it's a good reminder that the classic Dickey-Fuller statistics are mostly intended to come back negative. That's not always the case, to be clear, but it is the intended case. You assess the statistic by "how negative" it is. Stationary time series will produce test statistics more negative ("smaller") than those produced by non-stationary time series. In a way, this makes the hypotheses implicitly one-tailed (to use that language).

This function removes missing values from the vector before calculating test statistics.

Value

`sadf_test()` returns a list of length 3. The first element in the list is a matrix of tau statistics calculated by the test. The second element is a data frame of the simulated tau statistics of either a known white-noise time series or three different non-stationary time series (pure random walk, random walk with drift, random walk with drift and trend). The third element contains some attributes about the procedure for post-processing.

Author(s)

Steven V. Miller

References

Schwert, G. William. 1989. "Tests for Unit Roots: A Monte Carlo Investigation." *Journal of Business & Economic Statistics* 7(2): 147–159.

Examples

```
y <- na.omit(USDSEK[1:500,])$close # there is one missing value here. n = 499.


sadf_test(y, n_sims = 25) # Doing 25, just to make it quick
```

`sim_df_mod` *Simulate an Individual (Augmented) Dickey-Fuller Model*

Description

`sim_df_mod()` is designed as a helper function, to be used internally in this package in `sadf_test()`. But, you can use it here to simulate a time series and perform a(n Augmented) Dickey-Fuller test.

Usage

```
sim_df_mod(x, ts_type, df_lags = NULL, classic_df = FALSE, wn = FALSE)
```

Arguments

`x`	a numeric vector corresponding to a series to replicate/simulate
`ts_type`	a type of time-series to simulate (either 'ndnt', 'dnt', or 'dt')
`df_lags`	a numeric vector for the number of lags to calculate for the test.
`classic_df`	logical, defaults to FALSE. If FALSE, the function calculates an "Augmented" Dickey-Fuller test on a simulated series with the number of lagged first differences requested in the `df_lags` argument. If `TRUE`, the classic Dickey-Fuller test is executed without the lagged first differences.
`wn`	logical, defaults to FALSE. If FALSE, generates a random walk of some description for a DF/ADF test. If TRUE, series to be simulated for a DF/ADF test is white noise.

Details

`classic_df = TRUE` suppresses the need to specify `df_lags = 0`, but `df_lags` cannot be 0 if `classic_df = FALSE`.

This might change in future iterations, but it's worth clarifying the values assigned to the parameters of a drift and trend. The drift is randomly generated from a Rademacher distribution for both the times series with drift and drift-and-trend. The series with a deterministic trend divides the value from the Rademacher distribution by 10. My rationale is largely based on what I've seen other pedagogical guides do, the extent to which they talk about simulating values for these types of random walks.

Value

`sim_df_mod()` returns the output of a linear model (with class `lm`) that performs a(n Augmented) Dickey-Fuller test on a simulated time series. This is mostly for internal use, but it might pique the user's interest to see such a test in action independent of simulated summaries generated by `sadf_test()`.

Author(s)

Steven V. Miller

Examples

```
set.seed(8675309) # don't want new numbers in documentation every time...

sim_df_mod(rnorm(25), ts_type = 'ndnt', classic_df = TRUE)

sim_df_mod(rnorm(25), ts_type = 'ndnt', df_lags = 2, classic_df = FALSE)
```

`sim_ts` *Simulate a Time Series*

Description

`sim_ts()` is mostly a helper function, to be used internally in this package, but you can use it here to simulate a time series.

Usage

```
sim_ts(n, b0 = 0, bt = 0, rho = 1, white_noise = FALSE, rsd = 1)
```

Arguments

`n`	a numeric vector for the length of the series
`b0`	a numeric vector for a potential drift in the series. Defaults to 0
`bt`	a numeric vector for a potential trend in the series. Defaults to 0.
`rho`	a numeric vector for the simple autoregressive parameter. Defaults to 1.
`white_noise`	= logical, defaults to FALSE. If FALSE, generates a random walk. If TRUE, series is white noise.
`rsd`	the standard deviation for a normal distribution to be simulated. Defaults to 1.

Value

`sim_ts()` returns a numeric vector of a simulated time series that would follow the user's input.

Author(s)

Steven V. Miller

Examples

```
set.seed(8675309) # don't want new numbers in documentation every time...

sim_ts(25)

sim_ts(25, b0 = 1)

sim_ts(25, b0 = 1, bt = .05)
```

---

skpss_test *Simulate a KPSS Test to Assess Unit Root in a Time Series*

Description

`skpss_test()` provides a simulation approach to assessing unit root in a time series by way of KPSS test, first proposed by Kwiatkowski et al. (1992). It takes a vector and extracts the residuals from two models to assess stationarity around a level or trend, producing a KPSS test statistic (eta). Rather than interpolate or approximate a *p*-value, it simulates some user-specified number of KPSS of either a known, stationary time series (default) or a known, non-stationary time series matching the length of the time series the user provides. This allows the user to make assessments of non-stationarity or stationarity by way of simulation rather than approximation from received critical values by way of various books/tables.

Usage

```
skpss_test(x, lag_short = TRUE, n_sims = 1000, sim_hyp = "stationary")
```

Arguments

`x`	a vector
`lag_short`	logical, defaults to `TRUE`. If `TRUE`, the "short-term" lag is used for the KPSS test. If `FALSE`, the "long-term" lag is used. These lags are those suggested by Schwert (1989).
`n_sims`	the number of simulations for calculating an interval or distribution of test statistics for assessing stationarity or non-stationarity. Defaults to 1,000.
`sim_hyp`	can be either "stationary" or "nonstationary". If "stationary" (default), the function runs KPSS tests on simulated stationary (pure white noise) data. This allows the user to assess compatibility/plausibility of the test statistic against a distribution of test statistics that are known to be pure white noise (in expectation). If "nonstationary", the simulations are conducted on two different random walks. The "trend" test includes a level drawn from a Rademacher distribution with a time trend of that level, divided by 10.

Details

Recall that this procedure defaults from almost every other unit root test by making stationarity a null hypothesis. Non-stationarity is the alternative hypothesis.

As of writing, the default lags are those suggested by Schwert (1989) to apply to the Bartlett kernel generating the KPSS test statistic (eta).

aTSA has a particularly interesting approach to this test that draws on on insights seemingly proposed by Hobijn et al. (2004). Future updates may include those, but this function, as is, performs calculations of eta identical to what **tseries** or **urca** would produce. Right now, I don't have the time or energy to get into the weeds of what Hobijn et al. (2004) are doing or what **aTSA** is implementing, but it seems pretty cool.

This function removes missing values from the vector before calculating test statistics.

Value

`skpss_test()` returns a list of length 3. The first element in the list is a matrix of eta statistics calculated by the test. The first of those is the level statistic and the second of those is the trend statistic. The second element is a data frame of the simulated eta statistics, where the type of simulation (level, trend) is communicated in the `cat` column. The third element contains some attributes about the procedure for post-processing.

Author(s)

Steven V. Miller

References

Hobijn, Bart, Philip Hans Franses, and Marius Ooms. 2004. "Generalizations of the KPSS-test for Stationarity". *Statistica Neerlandica* 58(4): 483–502.

Kwiatkowski, Denis, Peter C.B. Phillips, Peter Schmidt, and Yongcheol Shin. 1992. "Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root: How Sure Are We that Economic Time Series Have a Unit Root?" *Journal of Econometrics* 54(1-3): 159–78.

Examples

```
x <- USDSEK$close[1:500] # note: one missing obs here; becomes series of 499

skpss_test(x, n_sims = 25) # make it quick...
```

`spp_test` *Simulate a Phillips-Perron Test to Assess Unit Root in a Time Series*

Description

`spp_test()` provides a simulation approach to assessing unit root in a time series by way of the Phillips-Perron test. It takes a vector and performs three Phillips-Perron tests (no drift, no trend; drift, no trend; drift and trend) and calculates both rho and tau statistics as one normally would. Rather than interpolate or approximate a *p*-value, it simulates some user-specified number of Phillips-Perron tests of either a known, non-stationary time series or a known, white-noise time series matching the length of the time series the user provides. This allows the user to make assessments of non-stationarity or stationarity by way of simulation rather than approximation from received critical values by way of various books/tables.

Usage

```
spp_test(x, lag_short = TRUE, n_sims = 1000, sim_hyp = "nonstationary")
```

Arguments

`x`	a vector
`lag_short`	logical, defaults to `TRUE`. If `TRUE`, the "short-term" lag is used for the Phillips-Perron test. If `FALSE`, the "long-term" lag is used.
`n_sims`	the number of simulations for calculating an interval or distribution of test statistics for assessing stationarity or non-stationarity. Defaults to 1,000.
`sim_hyp`	can be either "stationary" or "nonstationary". If "stationary", the function runs Phillips-Perron tests on simulated stationary (pure white noise) data. This allows the user to assess compatibility/plausibility of the test statistic against a distribution of test statistics that are known to be pure white noise (in expectation). If "nonstationary" (default), the function generates three different data sets of a pure random walk, a random walk with a drift, and a random walk with a drift and trend. It then runs Phillips-Perron tests on all those. This allows the user to assess the compatibility/plausibility of their test statistics with data that are known to be nonstationary in some form.

Details

Some knowledge of Augmented Dickey-Fuller and the Phillips-Perron procedure is assumed here. Generally, the Phillips-Perron test purports to build on the Augmented Dickey-Fuller (ADF) procedure through two primary means. The first is relaxing the need to be as deliberate about lag structures as ADF is, typically asking for some kind of long- or short-term lag for the bandwidth/kernel. The second, related to this, is that its nonparametric procedure makes it robust to various forms of heteroskedasticity in the error term.

The short-term and long-term lags follow the convention introduced in the Phillips-Perron test. The short-term lag uses the default number of Newey-West lags, defined as the floor of 4*(n/100)^.25 where `n` is the length of the time series. The long-term lag substitutes 4 for 12 in this equation.

We have Schwert (1989) to thank for these defaults. Future iterations of this function may permit manual lag selection, but this is fine for now.

This function specifies three different types of tests: 1) no drift, no trend, 2) drift, no trend, and 3) drift and trend. In the language of the `lm()` function, the first is `lm(y ~ ly - 1)` where `y` is the value of `y` and `ly` is its first-order lag. The second test is `lm(y ~ ly)`, intuitively suggesting the *y*-intercept in this equation is the "drift". The third would be `lm(y ~ ly + t)` with `t` being a simple integer that increases by 1 for each observation (i.e. a time-trend).

There are two types of statistics in the Phillips-Perron test: rho and tau. Of the two, tau is the more intuitive statistic and compares favorably to its corollary statistic in the Augmented Dickey-Fuller test. It's why you'll typically see tau reported as the statistic of interest in other implementations. rho has its utility for more advanced diagnostics, though. Both are calculated in this function, though tau is the default statistic.

None of this is meant to discourage the use of Fuller (1976) or its various reproductions for the sake of diagnosing stationarity or non-stationary, and I will confess their expertise on these matters outpaces mine. Consider the justification for this function to be largely philosophical and/or experimental. Why not simulate it? It's not like time or computing power are huge issues anymore.

This is always awkwardly stated, but it's a good reminder that the classic Dickey-Fuller statistics are mostly intended to come back negative. That's not always the case, to be clear, but it is the intended case. You assess the statistic by "how negative" it is. Stationary time series will produce test statistics more negative ("smaller") than those produced by non-stationary time series. In a way, this makes the hypotheses implicitly one-tailed (to use that language).

This function removes missing values from the vector before calculating test statistics.

Value

`spp_test()` returns a list of length 3. The first element in the list is a matrix of rho statistics and tau statistics calculated by the Phillips-Perron test. The second element is a data frame of the simulated rho and tau statistics of either a known white-noise time series or three different non-stationary time series (pure random walk, random walk with drift, random walk with drift and trend). The third element is some attributes about the procedure for post-processing.

Author(s)

Steven V. Miller

Examples

```
a <- rnorm(25) # white noise
b <- cumsum(a) # random walk

spp_test(a, n_sims = 25)
spp_test(b, n_sims = 25)
```

`tbills` *Daily maturity rates for U.S. Treasury Bills*

Description

A data frame on daily (when applicable/available) U.S. Treasury Bill rates. These are the yield received for investing in a government-issued treasury security that has a maturity of a given period of time (three months, six months, or a year).

Usage

```
tbills
```

Format

A data frame with 17,741 observations on the following 4 variables.

`date`	a date
`tb3m`	the three-month treasury bill rate
`tb6m`	the six-month treasury bill rate
`tb1y`	the one-year treasury bill rate

Details

Data come by way of **fredr** call. The one-year (52-week) treasury bill rate was discontinued in 2001 and re-introduced in 2008. Be mindful of that gap in the series.

`ur_summary` *Summarize Unit Root Test Simulations*

Description

`ur_summary()` provides a summary of the unit root tests included in this package.

Usage

```
ur_summary(obj, pp_stat = "tau", ...)
```

Arguments

`obj`	the object to be summarized, of class 'spp_test'
`pp_stat`	a statistic to be summarized: either "tau" or "rho". Applicable only to Phillips-Perron tests generated by functions in this package.
`...`	additional argument, currently ignored

Details

This function makes ample use of the "attributes" element in the list produced by the unit root simulations.

Value

`ur_summary()` produces console output that offers a summary assessment about the presence of a unit root based on your simulations.

Author(s)

Steven V. Miller

Examples

```
A <- spp_test(money_demand$ffer, n_sims = 100)
ur_summary(A)
```

`USDICE` *Quarterly disposable income and personal consumption expenditures in the United States*

Description

A data frame on personal consumption expenditures and disposable personal income in the United States.

Usage

```
USDICE
```

Format

A data frame with 299 observations on the following 4 variables.

`date`	a date
`pce`	personal consumption expenditures, seasonally adjusted, in billions
`dpi`	disposable personal income, seasonally adjusted, in billions
`pira`	personal income receipts on assets (personal dividend income), in billions
`cpiu`	consumer price index for all urban consumers (all items in U.S. city average)

Details

Data come by way of **fredr** call. Data are quarterly. Personal consumption expenditure. disposable personal income, and personal dividend income are not inflation-adjusted. The data on the consumer price index allow for such inflation adjustment to "real" dollars based on researcher discretion.

`USDSEK` *The USD/SEK Exchange Rate*

Description

A data frame on the USD/SEK exchange rate (i.e. how many Swedish crowns does one dollar get you).

Usage

```
USDSEK
```

Format

A data frame with 3905 observations on the following 2 variables.

`date`	a date
`close`	the exchange rate at the close of trading

Details

Data come by way of **quantmod**.

Index