Applied Econometrics with R

Chapter 6

Time Series

Time Series

Overview

Overview

Time series data: typical in macroeconomics and finance

Notation: y_t, $t = 1, \dots, n$.

Contents:

- Infrastructure and “naive” methods
- ARMA modeling
- Stationarity, unit roots, and cointegration
- Time series regression and structural change
- Extensions (GARCH, structural time series models)

Focus is on time domain methodology.

Overview

Background reading:

- Brockwell and Davis (2002): *Introduction to Time Series and Forecasting*, 2nd edition.
- Brockwell and Davis (1991): *Time Series – Theory and Methods*, 2nd edition.
- Franses (1998): *Time Series Models for Business and Economic Forecasting*
- Hamilton (1994): *Time Series Analysis*
- …

Time Series

Infrastructure and “Naive” Methods

Classes for time series data

Standard time series class in R is "`ts`":

- Aimed at regular series (annual, quarterly, monthly).
- A "`ts`" object is either a numeric vector (univariate series) or a numeric matrix (multivariate series).
- `"tsp"` attribute reflects time series properties:
  a vector of length 3 with start, end and frequency.
- Create via `ts()`: supply data (numeric vector or matrix) plus arguments `start`, `end`, and `frequency`.
- Methods for standard generic functions: `plot()`, `lines()`, `str()`, `summary()`, ...
- Additional time-series-specific methods: `lag()`, `diff()`, ...

Classes for time series data

Example: Quarterly consumption of non-durables in the United Kingdom (from Franses 1998)

Plot:

```
R> data("UKNonDurables")
R> plot(UKNonDurables)
```

Time series properties:

```
R> tsp(UKNonDurables)
```

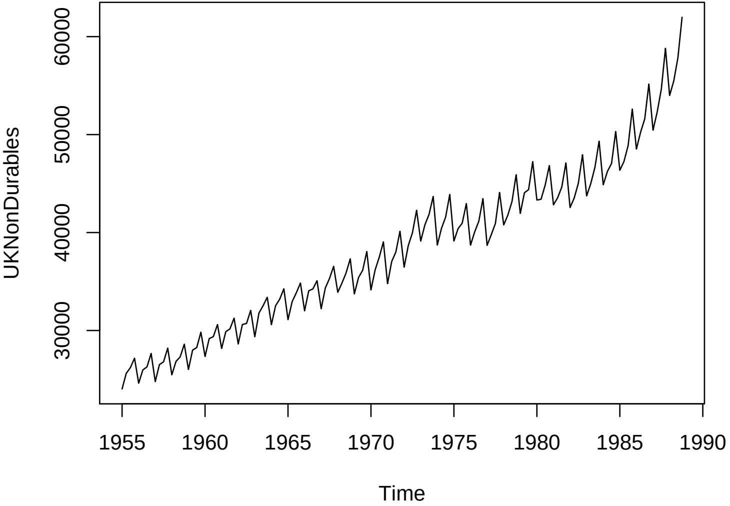
```
[1] 1955 1989    4
```

Subsets via `window()`:

```
R> window(UKNonDurables, end = c(1956, 4))
```

```
      Qtr1  Qtr2  Qtr3  Qtr4
1955 24030 25620 26209 27167
1956 24620 25972 26285 27659
```

Classes for time series data



Classes for time series data

Drawbacks of “`ts`”:

- Only numeric time stamps (more general date/time classes?)
- Missing values cannot be omitted (start/end/frequency no longer sufficient for reconstructing all time stamps!) – a problem with irregular series, e.g., with many financial time series.

R packages for irregular series: several, we use **zoo**

- Generalization of “`ts`”: time stamps of arbitrary type.
- Numeric vectors or matrices, `"index"` attribute contains *vector* of time stamps (not just `"tsp"` attribute!).
- Regular series can be coerced back and forth between “`ts`” and “`zoo`” via `as.zoo()` and `as.ts()`.
- “`zoo`” more convenient for daily data (e.g., “`Date`” time stamps) or intraday data (e.g., “`POSIXct`” or “`chron`” time stamps).
- More details: Zeileis and Grothendieck (*JSS* 2005).

(Linear) filtering

Linear filter: important class are finite moving averages

$$\hat{y}_t = \sum_{j=-r}^{s} a_j y_{t+j}, \quad t = r+1, \dots, n-s.$$

If $r = s$, filter is called symmetric.

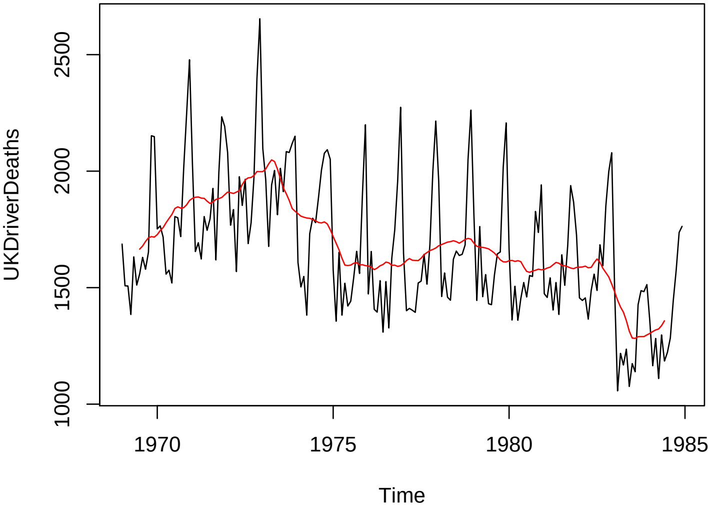
In R: function `filter()`

- Main argument `filter` takes vector containing a_js.
- Can also apply recursive linear filters.

Example: (`UKDriverDeaths`, Harvey and Durbin, *JRSS A* 1986)

```
R> data("UKDriverDeaths")
R> plot(UKDriverDeaths)
R> lines(filter(UKDriverDeaths, c(1/2, rep(1, 11), 1/2)/12),
+    col = 2)
```

(Linear) filtering



(Linear) filtering

Further examples:

`rollapply()` computes functions on moving data windows:

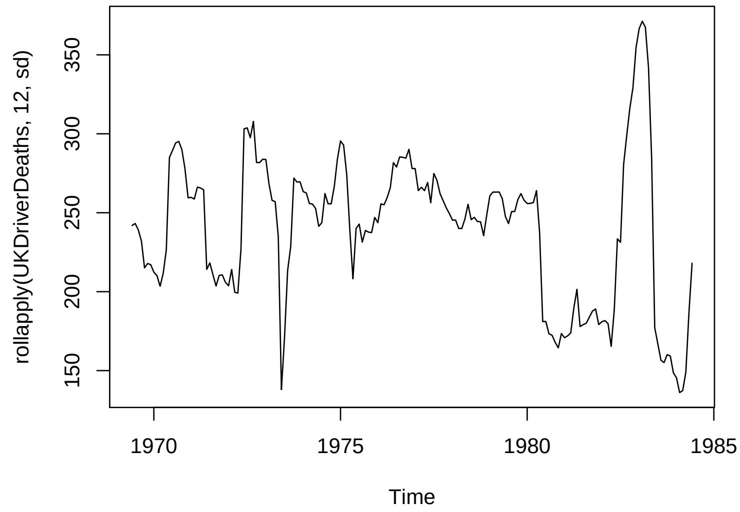
```
R> plot(rollapply(UKDriverDeaths, 12, sd))
```

`filter()` also provides autoregressive (recursive) filtering.

Generate 100 observations from AR(1) process:

```
R> set.seed(1234)
R> x <- filter(rnorm(100), 0.9, method = "recursive")
```

(Linear) filtering



Decomposition

Can use filters for additive or multiplicative decomposition into seasonal, trend, and irregular components.

In R:

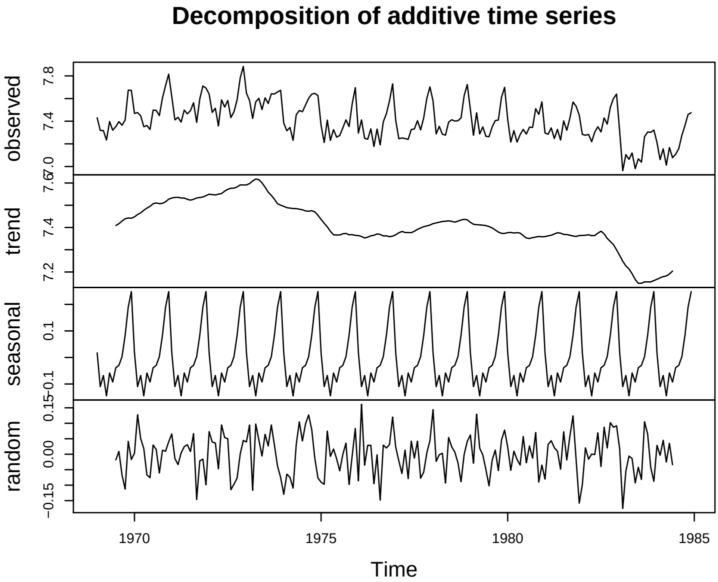
- `decompose()` takes simple symmetric filter for extracting trend, derives seasonal component by averaging trend-adjusted observations from corresponding periods.
- `stl()` iteratively finds seasonal and trend components by loess smoothing in moving data windows.

Examples:

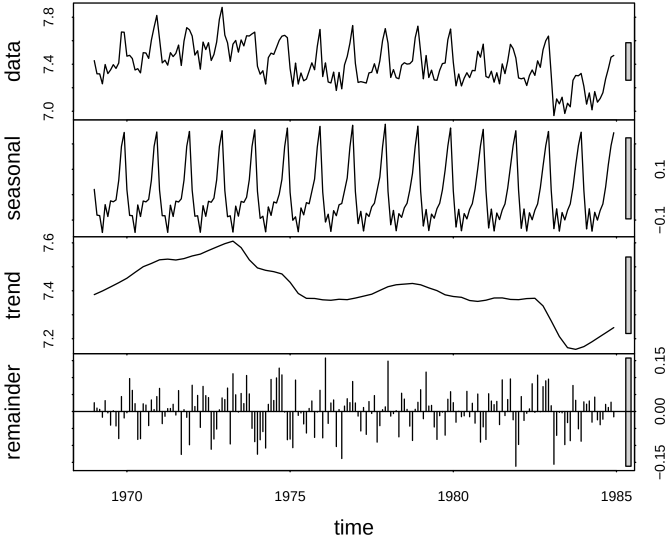
```
R> dd_dec <- decompose(log(UKDriverDeaths))
R> dd_stl <- stl(log(UKDriverDeaths), s.window = 13)
```

```
R> plot(dd_dec$trend, ylab = "trend")
R> lines(dd_stl$time.series[,"trend"], lty = 2, lwd = 2)
```

Decomposition



Decomposition



Decomposition

Exponential smoothing

`HoltWinters()` handles exponential smoothing and generalizations:

- Recursively reweighted lagged observations for predictions.
- Smoothing parameters determined by minimizing squared prediction error on observed data.
- Default: Holt-Winters filter with additive seasonal component.

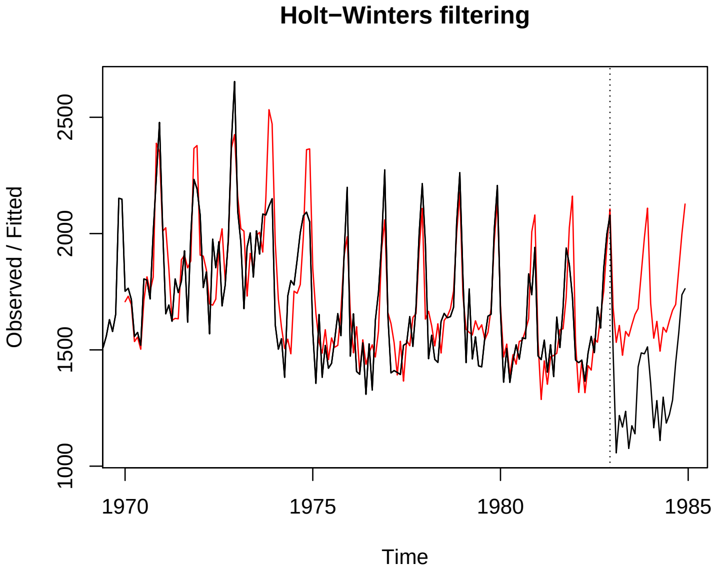
Example: `UKDriverDeaths`

- Historical sample up to 1982(12) (before change in legislation).
- Use Holt-Winters to predict observations for 1983 and 1984.

```
R> dd_past <- window(UKDriverDeaths, end = c(1982, 12))
R> dd_hw <- HoltWinters(dd_past)
R> dd_pred <- predict(dd_hw, n.ahead = 24)
```

```
R> plot(dd_hw, dd_pred, ylim = range(UKDriverDeaths))
R> lines(UKDriverDeaths)
```

Exponential smoothing



Time Series

Classical Model-Based Analysis

Classical model-based analysis

ARIMA(p, d, q) model is

$$\phi(L)(1-L)^d y_t = \theta(L)\varepsilon_t,$$

with

- $\phi(L) = 1 - \phi_1 L - \ldots - \phi_p L^p$, and
- $\theta(L) = 1 + \theta_1 L + \ldots + \theta_q L^q$ (note sign convention!),
- $\varepsilon_t \sim \mathrm{WN}(0, \sigma^2)$.

Generalization for seasonal data: multiplicative seasonal ARIMA

$$\Phi(L^s)\phi(L)(1-L^s)^D(1-L)^d y_t = \theta(L)\Theta(L^s)\varepsilon_t$$

Notation: SARIMA$(p, d, q)(P, D, Q)_s$

Classical model-based analysis

Time series fitting functions in R:

- `ar()` (from **stats**) fits AR models
  - univariate via Yule-Walker, OLS, ML, or Burg, and
  - multivariate (unrestricted VARs) by Yule-Walker, OLS, or Burg.

  Order selection by AIC possible.
- `arima()` (from **stats**) fits univariate ARIMA models, including SARIMA models, ARIMAX, and subset ARIMA models.
  Methods: unconditional ML or CSS.
- `arma()` (from **tseries**) fits ARMA models by CSS.
  Starting values via Hannan-Rissanen.
  Note: Parameterization of intercept different from `arima()`.
- `auto.arima()` (from **forecast**): Order selection via AIC, BIC, or AICC within user-defined set of models, fitting via `arima()`.
- `StructTS()` (from **stats**) fits structural time series models: local level, local trend, and basic structural model.

Classical model-based analysis

Box-Jenkins approach: use ACF and PACF for preliminary analysis.

In R: `acf()` and `pacf()`.

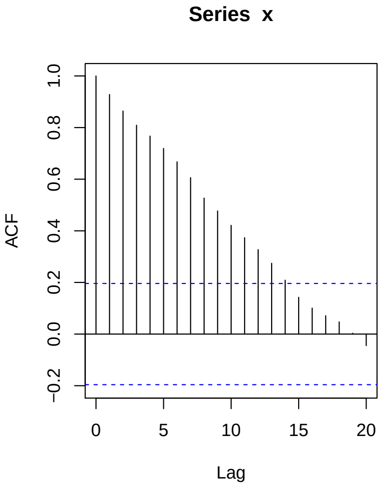
Example: simulated AR(1)

```
R> set.seed(1234)
R> x <- filter(rnorm(100), 0.9, method = "recursive")
R> acf(x)
R> pacf(x)
```

Fit autoregression to `x` via `ar()`:

```
R> ar(x)

Call:
ar(x = x)

Coefficients:
    1
0.928

Order selected 1  sigma^2 estimated as  1.29
```

Classical model-based analysis



Classical model-based analysis

Example: `UKNonDurables`

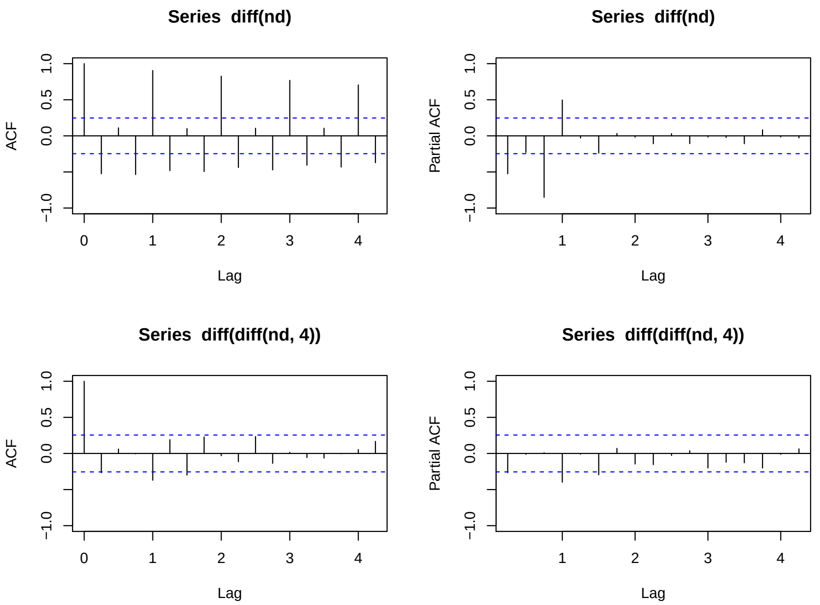
```
R> nd <- window(log(UKNonDurables), end = c(1970, 4))
```

Empirical ACFs and PACFs for

- nonseasonal differences
- seasonal and nonseasonal differences

```
R> acf(diff(nd), ylim = c(-1, 1))
R> pacf(diff(nd), ylim = c(-1, 1))
R> acf(diff(diff(nd, 4)), ylim = c(-1, 1))
R> pacf(diff(diff(nd, 4)), ylim = c(-1, 1))
```

Classical model-based analysis



Classical model-based analysis

Preliminary analysis suggests

- double differencing ($d = 1$, $D = 1$),
- some AR and MA effects – we use $p = 0, 1, 2$ and $q = 0, 1, 2$,
- low-order seasonal AR and MA parts – we use $P = 0, 1$ and $Q = 0, 1$.

This gives 36 parameter combinations in total. Manual solution:

- Set up all parameter combinations via `expand.grid()`.
- Fit each SARIMA model using `arima()` in `for()` loop.
- Store resulting BIC extracted from the model.
  For BIC, use `AIC()` with `k = log(length(nd))`.

Classical model-based analysis

```
R> nd_pars <- expand.grid(ar = 0:2, diff = 1, ma = 0:2,
+    sar = 0:1, sdiff = 1, sma = 0:1)
R> nd_aic <- rep(0, nrow(nd_pars))
R> for(i in seq(along = nd_aic)) nd_aic[i] <- AIC(arima(nd,
+    unlist(nd_pars[i, 1:3]), unlist(nd_pars[i, 4:6])),
+    k = log(length(nd)))
R> nd_pars[which.min(nd_aic),]
```

```
   ar diff ma sar sdiff sma
22  0    1  1   0     1   1
```

Result is $\text{SARIMA}(0,1,1)(0,1,1)_4$ – the *airline model*.

Refit to `nd` via

```
R> nd_arima <- arima(nd, order = c(0,1,1), seasonal = c(0,1,1))
```

Classical model-based analysis

```
R> nd_arima
Call:
arima(x = nd, order = c(0, 1, 1), seasonal = c(0, 1, 1))

Coefficients:
         ma1     sma1
      -0.353  -0.583
s.e.   0.143   0.138

sigma^2 estimated as 9.65e-05:  log likelihood = 188.1,  aic = -370.3
```

Diagnostic plots:

```
R> tsdiag(nd_arima)
```

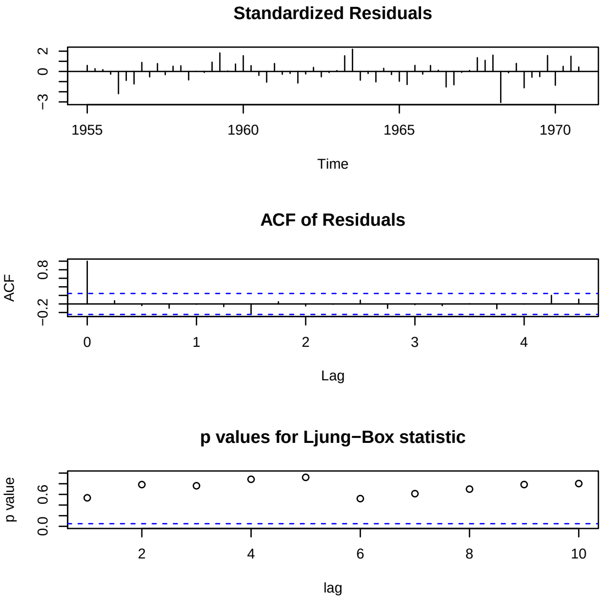
Forecast remaining 18 years:

```
R> nd_pred <- predict(nd_arima, n.ahead = 18 * 4)
```

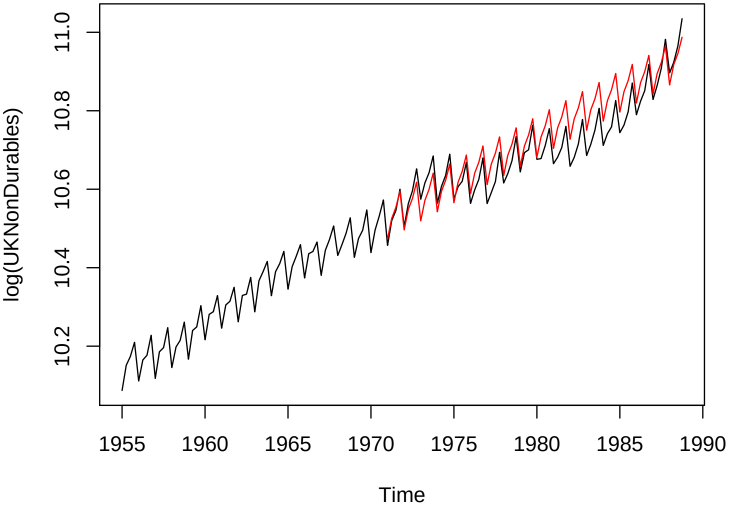
Graphical comparison with observed series:

```
R> plot(log(UKNonDurables))
R> lines(nd_pred$pred, col = 2)
```

Classical model-based analysis



Classical model-based analysis



Classical model-based analysis

Useful convenience functions for exploring ARMA models (all in **stats**):

- `acf2AR()` – computes AR process exactly fitting given autocorrelation function.
- `arima.sim()` – simulation of ARIMA models.
- `ARMAacf()` – theoretical (P)ACF for a given ARMA model.
- `ARMAtoMA()` – MA(∞) representation for a given ARMA model.

Time Series

Stationarity, Unit Roots, and Cointegration

Stationarity, unit roots, and cointegration

Many time series in macroeconomics and finance are nonstationary.

Need tests for

- unit roots,
- stationarity,
- cointegration.

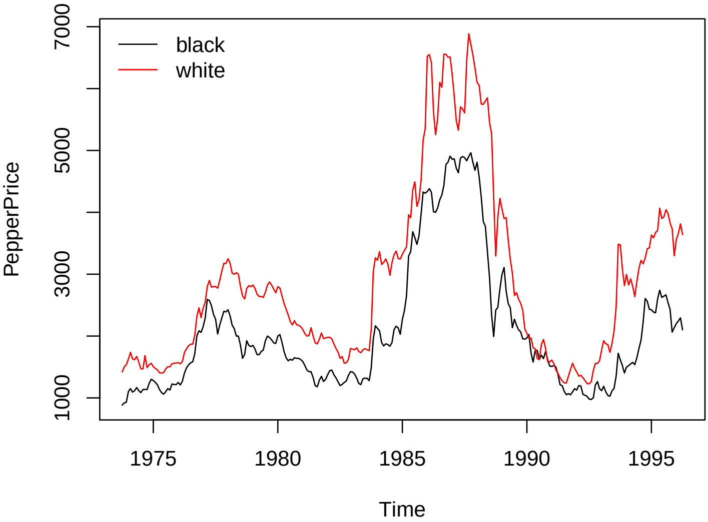
We use same data set for all these topics.

Example: from Franses 1998

Bivariate time series of average monthly European spot prices for black and white pepper (in US dollars per ton).

```
R> data("PepperPrice")
R> plot(PepperPrice, plot.type = "single", col = 1:2)
R> legend("topleft", c("black", "white"), bty = "n",
+  col = 1:2, lty = rep(1,2))
```

Stationarity, unit roots, and cointegration



Unit-root tests

Available tests:

- Augmented Dickey-Fuller (ADF) test: t test of $H_0 : \varrho = 0$ in

$$\Delta y_t = \alpha + \delta t + \varrho y_{t-1} + \sum_{j=1}^{k} \phi_j \Delta y_{t-j} + \varepsilon_t.$$

  In R: `adf.test()` from **tseries**.

- Phillips-Perron (PP) test:
  Same idea as ADF, but nonparametric (HAC) correction for autocorrelation.
  In R: `pp.test()` from **tseries**.

- Elliott-Rothenberg-Stock (ERS):
  Same idea as ADF, but GLS detrending.
  In R: `ur.ers()` from **urca**.

Unit-root tests

ADF in levels:

```
R> library("tseries")
R> adf.test(log(PepperPrice[, "white"]))

        Augmented Dickey-Fuller Test

data:  log(PepperPrice[, "white"])
Dickey-Fuller = -1.7, Lag order = 6, p-value = 0.7
alternative hypothesis: stationary
```

ADF in first differences:

```
R> adf.test(diff(log(PepperPrice[, "white"])))

        Augmented Dickey-Fuller Test

data:  diff(log(PepperPrice[, "white"]))
Dickey-Fuller = -5.3, Lag order = 6, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(diff(log(PepperPrice[, "white"]))) :
  p-value smaller than printed p-value
```

Unit-root tests

PP in levels (by default with time trend):

```
R> pp.test(log(PepperPrice[, "white"]), type = "Z(t_alpha)")

	Phillips-Perron Unit Root Test

data:  log(PepperPrice[, "white"])
Dickey-Fuller Z(t_alpha) = -1.6, Truncation lag
parameter = 5, p-value = 0.7
alternative hypothesis: stationary
```

Stationarity tests

Kwiatkowski, Phillips, Schmidt and Shin (*J. Econometrics* 1992):

Test H_0 : $\quad r_t \equiv 0$ in

$$y_t = d_t + r_t + \varepsilon_t,$$

where

- d_t deterministic trend,
- r_t random walk,
- ε_t stationary ($I(0)$) error process.

Two variants:

- $d_t = \alpha$, level stationarity (under H_0).
- $d_t = \alpha + \beta t$, trend stationarity (under H_0).

Stationarity tests

KPSS without time trend:

```
R> kpss.test(log(PepperPrice[, "white"]))

        KPSS Test for Level Stationarity

data:  log(PepperPrice[, "white"])
KPSS Level = 0.91, Truncation lag parameter = 3, p-value
= 0.01

Warning message:
p-value smaller than printed p-value in:
kpss.test(log(PepperPrice[, "white"]))
```

Cointegration

Pepper series exhibit common nonstationary features.

Cointegration tests in R:

- Engle-Granger two-step method
  Available in `po.test()` from **tseries** (named after Phillips and Ouliaris, *Econometrica* 1990).
- Johansen test
  Full-information maximum likelihood approach in *p*th-order cointegrated VAR. Error correction form (ECM) is (without deterministic components)

$$\Delta y_t = \Pi y_{t-1} + \sum_{j=1}^{p-1} \Gamma_j \Delta y_{t-j} + \varepsilon_t.$$

  Trace and lambda-max tests available in `ca.jo()` from **urca**.

Cointegration

Engle-Granger two-step with black pepper regressed on white pepper:

```
R> po.test(log(PepperPrice))

        Phillips-Ouliaris Cointegration Test

data:  log(PepperPrice)
Phillips-Ouliaris demeaned = -24, Truncation lag
parameter = 2, p-value = 0.02
```

Suggests both series are cointegrated.

Remarks:

- Test with reverse regression is `po.test(log(PepperPrice[,2:1]))`
- Problem: treatment asymmetric, but concept cointegration demands symmetric treatment!

Cointegration

Johansen test with constant term

```
R> library("urca")
R> pepper_jo <- ca.jo(log(PepperPrice), ecdet = "const",
+    type = "trace")
R> summary(pepper_jo)

######################
# Johansen-Procedure #
######################

Test type: trace statistic , without linear trend and
constant in cointegration

Eigenvalues (lambda):
[1] 4.932e-02 1.351e-02 2.082e-17
```

Cointegration

```
Values of teststatistic and critical values of test:

          test 10pct  5pct  1pct
r <= 1 |  3.66  7.52  9.24 12.97
r = 0  | 17.26 17.85 19.96 24.60

Eigenvectors, normalised to first column:
(These are the cointegration relations)

         black.l2 white.l2 constant
black.l2   1.0000    1.000    1.000
white.l2  -0.8892   -5.099    2.281
constant  -0.5570   33.027  -20.032

Weights W:
(This is the loading matrix)

        black.l2 white.l2   constant
black.d -0.07472 0.002453 -4.958e-18
white.d  0.02016 0.003537  8.850e-18
```

Time Series Regression and Structural Change

More on fitting dynamic regression models

Example: SARIMA$(1,0,0)(1,0,0)_{12}$ for `UKDriverDeaths`

$$y_t = \beta_1 + \beta_2 \, y_{t-1} + \beta_3 \, y_{t-12} + \varepsilon_t, \quad t = 13, \dots, 192.$$

Two approaches:

Approach 1: set up regressors “by hand” and call `lm()`

```
R> dd <- log(UKDriverDeaths)
R> dd_dat <- ts.intersect(dd, dd1 = lag(dd, k = -1),
+    dd12 = lag(dd, k = -12))
R> lm(dd ~ dd1 + dd12, data = dd_dat)

Call:
lm(formula = dd ~ dd1 + dd12, data = dd_dat)

Coefficients:
(Intercept)          dd1         dd12
      0.421        0.431        0.511
```

More on fitting dynamic regression models

Approach 2: use convenience interface `dynlm()` from **dynlm**

```
R> library("dynlm")
R> dynlm(dd ~ L(dd) + L(dd, 12))

Time series regression with "ts" data:
Start = 1970(1), End = 1984(12)

Call:
dynlm(formula = dd ~ L(dd) + L(dd, 12))

Coefficients:
(Intercept)        L(dd)    L(dd, 12)
      0.421        0.431        0.511
```

Structural change tests

Features of `UKDriverDeaths`:

- Decrease in mean number of casualties after policy change.
- Parameters of time series model unlikely to be stable throughout sample period.

Package **strucchange** implements large collection of tests for structural change (parameter instability).

Two types of tests:

- Fluctuation tests.
- Tests based on F statistics.

Structural change tests

Fluctuation tests:

- Assess structural stability by capturing fluctuation in CUSUMs or MOSUMs of
  - residuals (OLS or recursive),
  - model scores (empirical estimating functions), or
  - parameter estimates (recursive or rolling).
- Idea: under null hypothesis of parameter stability, resulting “fluctuation processes” exhibit limited fluctuation, under alternative of structural change, fluctuation is generally increased.
- Evidence for structural change if empirical fluctuation process crosses boundary that corresponding limiting process crosses only with probability α.

Structural change tests

Fluctuation tests in strucchange:

- empirical fluctuation processes via `efp()`.
- Result is object of class "`efp`".
- `plot()` method for performing test graphically.
- `sctest()` method (for structural change test) for traditional significance test.

Example: OLS-CUSUM for `UKDriverDeaths`

OLS-CUSUM process: Scaled CUSUM of OLS residuals $\hat{\varepsilon}_t = y_t - x_t^\top \hat{\beta}$

$$efp(s) = \frac{1}{\hat{\sigma}\sqrt{n}} \sum_{t=1}^{\lfloor ns \rfloor} \hat{\varepsilon}_t, \quad 0 \leq s \leq 1.$$

Structural change tests

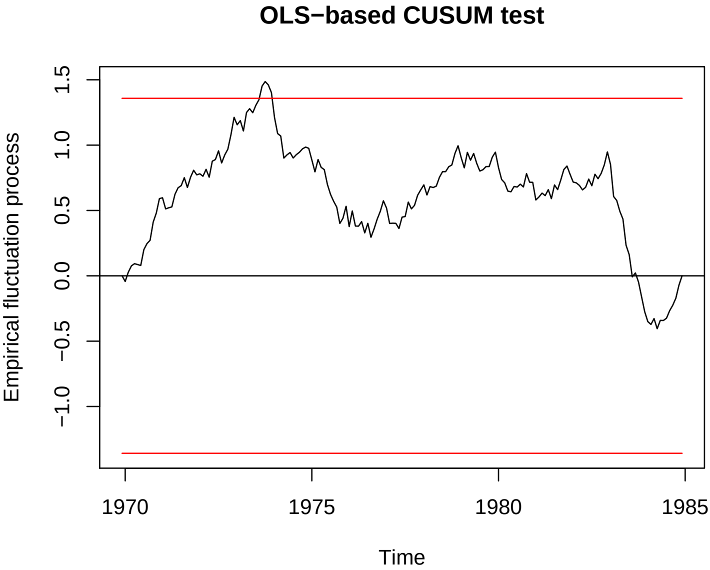
In R:

```
R> library("strucchange")
R> dd_ocus <- efp(dd ~ dd1 + dd12, data = dd_dat,
+    type = "OLS-CUSUM")
```

Test using maximum absolute deviation of `efp` (default functional)

```
R> sctest(dd_ocus)

        OLS-based CUSUM test

data:  dd_ocus
S0 = 1.5, p-value = 0.02

R> plot(dd_ocus)
```

Structural change tests



Structural change tests

Tests based on F statistics:

- Designed to have good power for single-shift alternatives (of unknown timing).
- Basic idea is to compute an F statistic (or Chow statistic) for each conceivable breakpoint in given interval (trimming parameter).
- Reject the null hypothesis of structural stability if
  - any of these statistics ($\sup F$ test)
  - some other functional (Andrews-Ploberger, *Econometrica* 1994: mean-F , exp-F)

  exceeds critical value.

Structural change tests

In R: function `Fstats()`, with interface similar to `efp()`

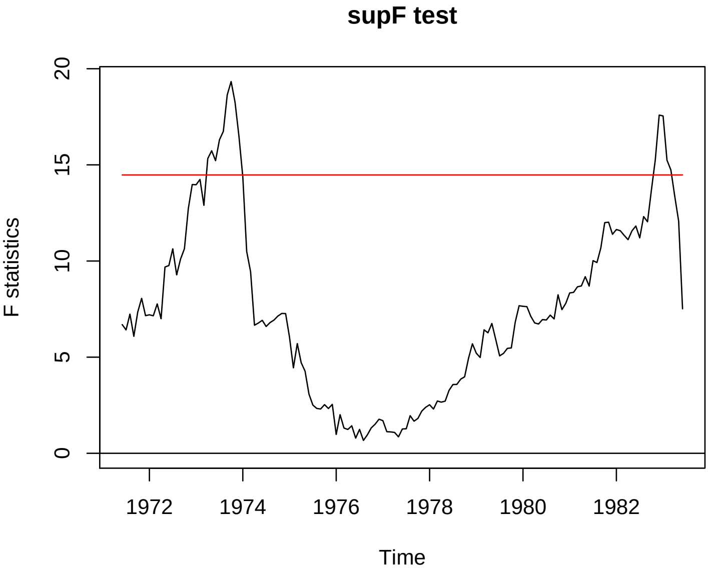
sup$F$ test with 10% trimming via

```
R> dd_fs <- Fstats(dd ~ dd1 + dd12, data = dd_dat, from = 0.1)
R> sctest(dd_fs)

        supF test

data:  dd_fs
sup.F = 19, p-value = 0.007
```

Visualization:

```
R> plot(dd_fs, main = "supF test")
```

Structural change tests



Structural change tests

Further Example: German M1 money demand

- Lütkepohl, Teräsvirta and Wolters (*JAE* 1999) use error correction model (ECM) for German M1.
- `GermanM1` contains data from 1961(1) to 1995(4) on per capita M1, price index, per capita GNP (all in logs) and an interest rate.

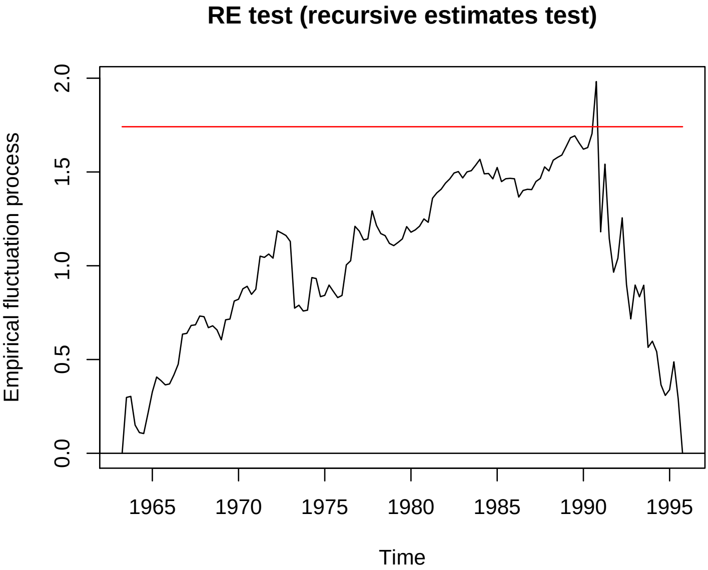
Load and set up model

```
R> data("GermanM1")
R> LTW <- dm ~ dy2 + dR + dR1 + dp + m1 + y1 + R1 + season
```

Recursive estimates (RE) test (Ploberger, Krämer and Kontrus, *J. Econometrics* 1989)

```
R> m1_re <- efp(LTW, data = GermanM1, type = "RE")
R> plot(m1_re)
```

Structural change tests



Dating structural changes

Setup is linear regression model

$$y_t = x_t^\top \beta^{(j)} + \varepsilon_t, \qquad t = n_{j-1}+1, \dots, n_j, \quad j = 1, \dots, m+1,$$

where

- $j = 1, \dots, m$ segment index,
- $\beta^{(j)}$ segment-specific set of regression coefficients,
- $\{n_1, \dots, n_m\}$ set of unknown breakpoints (convention: $n_0 = 0$ and $n_{m+1} = n$).

In R: function `breakpoints()`

- Uses dynamic programming algorithm based on Bellman principle.
- Finds those m breakpoints that minimize RSS of model with $m+1$ segments.
- Bandwidth parameter h determines minimal segment size of $h \cdot n$ observations.

Dating structural changes

Example: `UKDriverDeaths`

Breakpoints for SARIMA model with minimal segment size of 10%

```
R> dd_bp <- breakpoints(dd ~ dd1 + dd12, data = dd_dat, h = 0.1)
```

```
R> coef(dd_bp, breaks = 2)
```

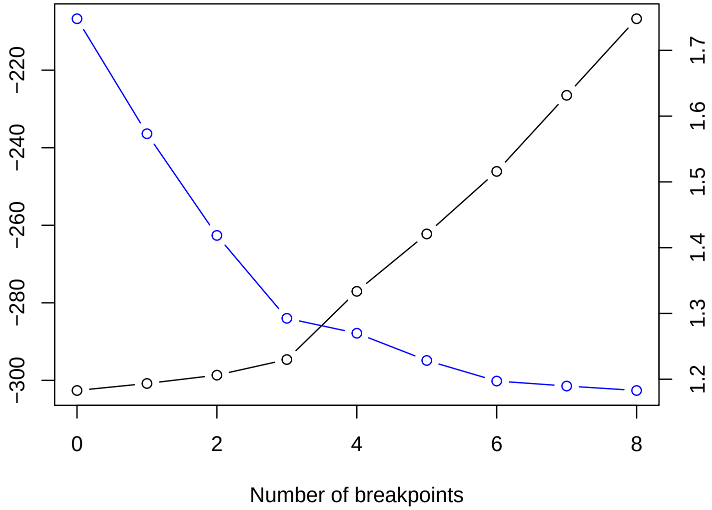
```
                    (Intercept)    dd1   dd12
1970(1) - 1973(10)        1.458 0.1173 0.6945
1973(11) - 1983(1)        1.534 0.2182 0.5723
1983(2) - 1984(12)        1.687 0.5486 0.2142
```

Visualization

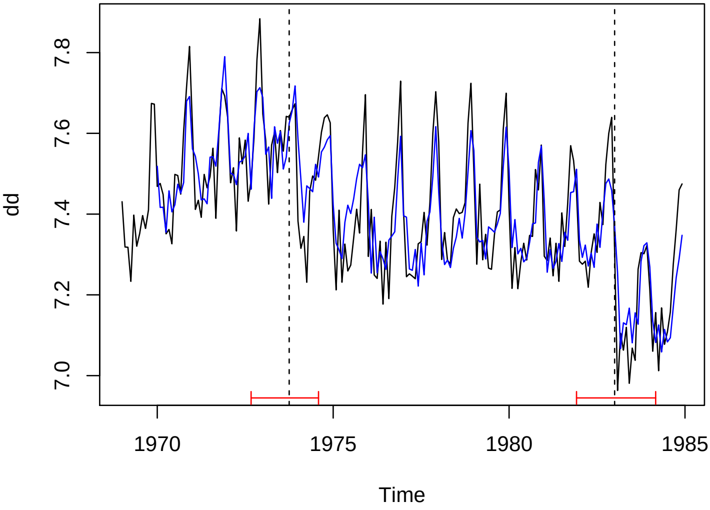
```
R> plot(dd_bp, legend = FALSE, main = "")
```

```
R> plot(dd)
R> lines(fitted(dd_bp, breaks = 2), col = 4)
R> lines(confint(dd_bp, breaks = 2))
```

Dating structural changes



Dating structural changes



Time Series

Extensions

Extensions

Further packages for time series analysis

- **dse** – Multivariate time series modeling with state-space and vector ARMA (VARMA) models.
- **FinTS** – R companion to Tsay (2005).
- **forecast** – Univariate time series forecasting, including exponential smoothing, state space, and ARIMA models.
- **fracdiff** – ML estimation of ARFIMA models and semiparametric estimation of the fractional differencing parameter.
- **longmemo** – Convenience functions for long-memory models.
- **mFilter** – Time series filters, including Baxter-King, Butterworth, and Hodrick-Prescott.
- **Rmetrics** – Some 20 packages for financial engineering and computational finance, including GARCH modeling in **fGarch**.
- **tsDyn** – Nonlinear time series models: STAR, ESTAR, LSTAR.
- **vars** – (Structural) vector autoregressive (VAR) models

Structural time series models

Basic structural model has measurement equation

$$y_t = \mu_t + \gamma_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2) \text{ i.i.d.}$$

Seasonal component γ_t (with frequency s) is

$$\gamma_{t+1} = -\sum_{j=1}^{s-1} \gamma_{t+1-j} + \omega_t, \quad \omega_t \sim \mathcal{N}(0, \sigma_\omega^2) \text{ i.i.d.}$$

Local level and trend components are

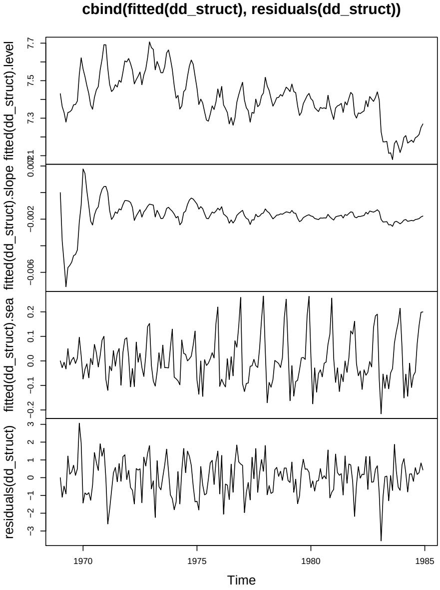
$$\begin{aligned} \mu_{t+1} &= \mu_t + \eta_t + \xi_t, \quad \xi_t \sim \mathcal{N}(0, \sigma_\xi^2) \text{ i.i.d.}, \\ \eta_{t+1} &= \eta_t + \zeta_t, \quad \zeta_t \sim \mathcal{N}(0, \sigma_\zeta^2) \text{ i.i.d.} \end{aligned}$$

All error terms mutually independent.

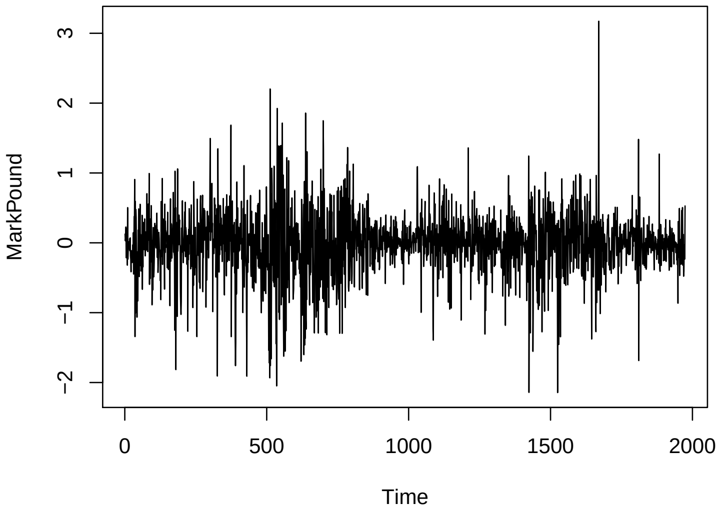
In R:

```
R> dd_struct <- StructTS(log(UKDriverDeaths))
R> plot(cbind(fitted(dd_struct), residuals(dd_struct)))
```

Structural time series models



GARCH models



GARCH models

tseries function `garch()` fits GARCH(p, q) with Gaussian innovations. Default is GARCH(1, 1):

$$
\begin{aligned}
y_t &= \sigma_t \nu_t, \quad \nu_t \sim \mathcal{N}(0, 1) \text{ i.i.d.}, \\
\sigma_t^2 &= \omega + \alpha y_{t-1}^2 + \beta \sigma_{t-1}^2, \quad \omega > 0, \alpha > 0, \beta \geq 0.
\end{aligned}
$$

Example: DEM/GBP FX returns for 1984-01-03 through 1991-12-31

```
R> mp <- garch(MarkPound, grad = "numerical", trace = FALSE)
R> summary(mp)
```

```
Call:
garch(x = MarkPound, grad = "numerical", trace = FALSE)

Model:
GARCH(1,1)

Residuals:
     Min       1Q   Median       3Q      Max
-6.79739 -0.53703 -0.00264  0.55233  5.24867
```

GARCH models

```
Coefficient(s):
    Estimate  Std. Error  t value Pr(>|t|)
a0    0.0109      0.0013     8.38   <2e-16
a1    0.1546      0.0139    11.14   <2e-16
b1    0.8044      0.0160    50.13   <2e-16

Diagnostic Tests:
        Jarque Bera Test

data:  Residuals
X-squared = 1100, df = 2, p-value <2e-16


        Box-Ljung test

data:  Squared.Residuals
X-squared = 2.5, df = 1, p-value = 0.1
```

Remarks:

- *Warning:* OPG standard errors assuming Gaussian innovations.
- More flexible GARCH modeling via `garchFit()` in **fGarch**.