Data Structures: Times, Dates, Ordered Observations . . . and Beyond

Achim Zeileis Kurt Hornik

http://statmath.wu-wien.ac.at/~zeileis/

Overview

- Data structures
  - Principles
  - Object orientation
- Times and dates
  - years, quarters, months
  - days
  - intra-day times
- Time series
  - (Numerical) Observations ordered/indexed by time
  - Operations on time series
- Re-use in more complex objects
  - Empirical fluctuation processes

Data structures

To collaboratively work on a project, communicate and exchange information, participants of the project need to find a common *language*.

Luckily, for the participants of this workshop this question is settled: the lingua franca is **R**.

However, to talk about complex data, quantitative methods and analyses, a common basic language is not enough. We need a precise *terminology* to assure we are talking about the same things. (Just as mathematicians and economists do not necessarily understand each other just because they talk English).

Data structures: Principles

In terms of software, this means that we need common data structures and functions that can be employed by everybody and that always mean the same thing.

Many basic building blocks are already available in R or contributed packages, but new or advanced tools are required for many concepts.

Data structures: Principles

Available building blocks:

- re-use this existing infrastructure,
- possibly by providing common interfaces to different object types denoting similar conceptual entities,
- if this is *open-source* infrastructure: improve it (rather than re-invent it),

Creating new data structures:

- modular design that reflects underlying concepts,
- build on and provide re-usable components,
- flexible user interface, e.g., via object orientation.

Data structures: Object orientation

Important: useful/intuitive object structures, i.e., implementations of objects that capture all necessary information of underlying conceptual entities.

More important: functional interfaces, i.e., accessor functions for relevant information and functions that carry out typical analysis steps.

In the S3 world, these can be generated with minimal overhead even if the interfaced class was generated by someone else.

Time and dates

In finance, business and economics, time-annotated data is ubiquitous. Therefore, a fundamental building block for more complex structures in finance/business/economics are times and dates.

In (macro-)economics, the most important types of times are/were years, quarters and months.

In finance, required times are more often at the daily or intra-day level.

Time and dates

Typical actions and operations:

- set up time from numeric or character input,

- extract underlying numeric scale,
- produce character label,

- use for plotting,
- sequences of times with given spacing,
- time differences,
- move forward/backward on time scale,

- comparison (less/greater).

Time and dates

Typical actions and operations (and R functions/generics):

- set up time from numeric or character input:
  class constructors,
- extract underlying numeric scale: `as.numeric()` method,
- produce character label:
  `format()` and `as.character()` method,
- use for plotting: input for `plot()`, e.g., `Axis()` method,
- sequences of times with given spacing: `seq()` method,
- time differences: group generic functions or `difftime()`,
- move forward/backward on time scale:
  group generic functions,
- comparison (less/greater): group generic functions.

Time: Years

Example: 1997, 1998, 2002, 2004, ...

Class: "`numeric`" or (even better) "`integer`"

```
R> ty <- c(1997, 1998, 2002, 2004)
R> ty
```

```
[1] 1997 1998 2002 2004
```

Time: Years

```
R> as.character(ty)

[1] "1997" "1998" "2002" "2004"

R> ty[2] - ty[1]

[1] 1

R> ty + 1

[1] 1998 1999 2003 2005
```

Time: Quarters

Example: 2000 Q1, 2001 Q3, 2002 Q2, 2002 Q3, ...

Class: "`numeric`" (first attempt)

```
R> tq <- c(2000, 2001.5, 2002.25, 2002.5)
R> tq
```

```
[1] 2000.00 2001.50 2002.25 2002.50
```

Time: Quarters

```
R> tq[2] - tq[1]

[1] 1.5

R> tq + 1/4

[1] 2000.25 2001.75 2002.50 2002.75

R> as.character(tq)

[1] "2000"    "2001.5"  "2002.25" "2002.5"
```

Time: Quarters

Class: "`yearqtr`" (improved)

```
R> tq <- as.yearqtr(tq)
R> as.character(tq)
```

```
[1] "2000 Q1" "2001 Q3" "2002 Q2" "2002 Q3"
```

```
R> as.numeric(tq)
```

```
[1] 2000.00 2001.50 2002.25 2002.50
```

```
R> tq[2] - tq[1]
```

```
[1] 1.5
```

```
R> tq + 1/4
```

```
[1] "2000 Q2" "2001 Q4" "2002 Q3" "2002 Q4"
```

Time: Quarters

Idea: "`numeric`" vector with class attribute that handles matching/rounding correctly and provides coercion to many other classes.

Class constructor:

```
yearqtr <- function(x)
  structure(floor(4*x + .001)/4, class = "yearqtr")
```

provided in package **zoo**.

Time: Months

Example: Jan 2000, Oct 2001, Dec 2001, Aug 2002, ...

Class: "`yearmon`" (analogous to "`yearqtr`")

```
R> tm <- yearmon(c(2000, 2001, 2001, 2002) + c(0, 9, 11,
+     7)/12)
R> tm
```

```
[1] "Jan 2000" "Oct 2001" "Dec 2001" "Aug 2002"
```

```
R> as.yearmon(2000)
```

```
[1] "Jan 2000"
```

```
R> as.yearmon("2000 Jan", format = "%Y %b")
```

```
[1] "Jan 2000"
```

Time: Days

Example: 1970-01-01, 2001-07-12, 2005-03-24, ...

Class: "`Date`" (number of days since 1970-01-01)

```
R> td <- as.Date(c("1970-01-01", "2001-07-12", "2005-03-24"))
R> td
```

```
[1] "1970-01-01" "2001-07-12" "2005-03-24"
```

```
R> as.numeric(td)
```

```
[1]     0 11515 12866
```

```
R> as.character(td)
```

```
[1] "1970-01-01" "2001-07-12" "2005-03-24"
```

Time: Days

```
R> format(as.Date(td), "%B %d, %Y")

[1] "January 01, 1970" "July 12, 2001"    "March 24, 2005"

R> td[2] - td[1]

Time difference of 11515 days

R> td + 1

[1] "1970-01-02" "2001-07-13" "2005-03-25"

R> as.Date(2)

[1] "1970-01-03"
```

Time: Intra-day

Example: 1970-01-01 00:00:00, 2001-07-12 12:23:59, ...

Class: "`chron`" (days since 1970-01-01, without time zone or daylight savings time)

```
R> tc <- chron(c(0, 11515 + 12/24 + 23/1440 + 59/86400))
R> tc
```

```
[1] (01/01/70 00:00:00) (07/12/01 12:23:59)
```

```
R> as.character(tc)
```

```
[1] "(01/01/70 00:00:00)" "(07/12/01 12:23:59)"
```

Time: Intra-day

```
R> as.numeric(tc)

[1]     0.00 11515.52

R> paste(format(as.Date(dates(tc))), format(times(tc)%%1))

[1] "1970-01-01 00:00:00" "2001-07-12 12:23:59"

R> tc[2] - tc[1]

Time in days:
[1] 11515.52
```

Time: Intra-day

Example: 1970-01-01 00:00:00 GMT,
2001-07-12 12:23:59 GMT, ...

Class: "`POSIXct`" (seconds since 1970-01-01, with time zone and daylight savings time)

```
R> tp <- structure(c(0, 994940639), class = c("POSIXt",
+     "POSIXct"), tzone = "GMT")
R> tp
```

```
[1] "1970-01-01 00:00:00 GMT" "2001-07-12 12:23:59 GMT"
```

```
R> as.numeric(tp)
```

```
[1]         0 994940639
```

Time: Intra-day

```
R> as.character(tp)

[1] "1970-01-01 00:00:00" "2001-07-12 12:23:59"

R> format(tp, "%B %d, %Y (%H:%M:%S)")

[1] "January 01, 1970 (00:00:00)" "July 12, 2001 (12:23:59)"

R> tp[2] - tp[1]

Time difference of 11515.52 days
```

Time and dates: Summary and outlook

- many time/date classes already available,
- existing infrastructure can be leveraged for customizing (similar to "`yearqtr`"/"`yearmon`"),
- use time/date class that is appropriate for your data (and not more complex),
- coercions between the classes allow for conversions between time/date formats,
- intra-day data is more complicated, especially with time zones and daylight savings times.

Time and dates: Summary and outlook

```
R> as.Date(tq)
[1] "2000-01-01" "2001-07-01" "2002-04-01" "2002-07-01"

R> as.POSIXct(tq)
[1] "2000-01-01 01:00:00 CET"  "2001-07-01 02:00:00 CEST"
[3] "2002-04-01 02:00:00 CEST" "2002-07-01 02:00:00 CEST"

R> as.yearqtr(td)
[1] "1970 Q1" "2001 Q3" "2005 Q1"

R> as.yearqtr(tp)
[1] "1970 Q1" "2001 Q3"

R> as.Date(tp)
[1] "1970-01-01" "2001-07-12"
```

Time and dates: Summary and outlook

```
R> as.chron(tp)

[1] (01/01/70 00:00:00) (07/12/01 12:23:59)
attr(,"tzone")
[1] GMT

R> as.POSIXct(td)

[1] "1970-01-01 01:00:00 CET"  "2001-07-12 02:00:00 CEST"
[3] "2005-03-24 01:00:00 CET"

R> as.POSIXct(tc)

[1] "1970-01-01 01:00:00 CET"  "2001-07-12 14:23:58 CEST"
```

Time and dates: Summary and outlook

If the local time zone is set to `"GMT"`, such problems do not occur. However, working with time zones requires a lot of care and is typically dependent on the operating system. See Grothendieck & Petzoldt (2004) for more information.

The class "`timeDate`" in package **fCalendar** tries to address this problem by leveraging available POSIX structures but using *financial centers* rather than time zones to account for differences between different locations.

However, the interface of "`timeDate`" is somewhat different than that of the classes above. Hence, a leaner implementation is planned (but not much code has been written yet).

Time series: Structure

Time/date objects are usually not interesting as standalone objects but are used to annotate other data.

The most important application of this are *time series* where there is for each time point a vector of (typically numeric) observations.

The observations are most easily arranged in a vector (of length n) or an $n \times k$ matrix whose elements are ordered and indexed by a time vector of length n.

Time series: Structure

A time series can either be *irregular* (unequally spaced), strictly *regular* (equally spaced) or have an underlying regularity, i.e., be created from a regular series by omitting some observations.

For strictly regular series, the whole vector of time vector can be reconstructed from start, end and time difference between two observations. The reciprocal value of the time difference is also called *frequency*.

Time series: Operations

Typical operations for time series:

- visualization,
- extraction of observations or associated times,

- lags and differences,
- subsets in a certain time window
- union and intersection of several time series,
- aggregation along a coarser time grid,
- rolling computations such as means or standard deviations.

Time series: Operations

Typical operations for time series (and suitable R generics):

- visualization: `plot()`,
- extraction of observations or associated times: `time()` (and `coredata()`),
- lags and differences: `lag()` and `diff()`,
- subsets in a certain time window: `window()`,
- union and intersection of several time series: `merge()`,
- aggregation along a coarser time grid: `aggregate()`,
- rolling computations such as means or standard deviations: (`rollapply()`).

Time series: Implementations

There are many implementations for time-series data in R.

Virtually all of them are focused on numeric data and fix some particular class for the time index:

- "`ts`": regular "`numeric`" time index (e.g., annual, quarterly, monthly),
- "`its`": irregular time index of class "`POSIXct`",
- "`irts`": irregular time index of class "`POSIXct`",
- "`timeSeries`": irregular time index of class "`timeDate`",
- "`zoo`": regular or irregular time index of arbitrary class.

Time series: Implementations

How can an arbitrary time index work?

To provide the desired functionality, few actions are required:

- ordering,
- matching,
- combining,
- subsetting,
- querying length n.

Time series: Implementations

How can an arbitrary time index work?

To provide the desired functionality, few actions are required:

- ordering: `ORDER()` (by default calling the non-generic `order()`),
- matching: `MATCH()` (by default calling the non-generic `match()`),
- combining: `c()`,
- subsetting: `[,`,
- querying length n: `length()`.

Time series: Implementations

If suitable methods are available for the chosen time index (including all time/date classes above), all tasks (merging, aggregating, etc.) can be performed without any knowledge about the particular time index class.

For some special operations, further methods are useful/necessary, e.g., `axis()`/`Axis()`, `as.numeric()`, `as.character()`.

If the time index is of a particular class, coercions to and from other time series classes can be easily provided.

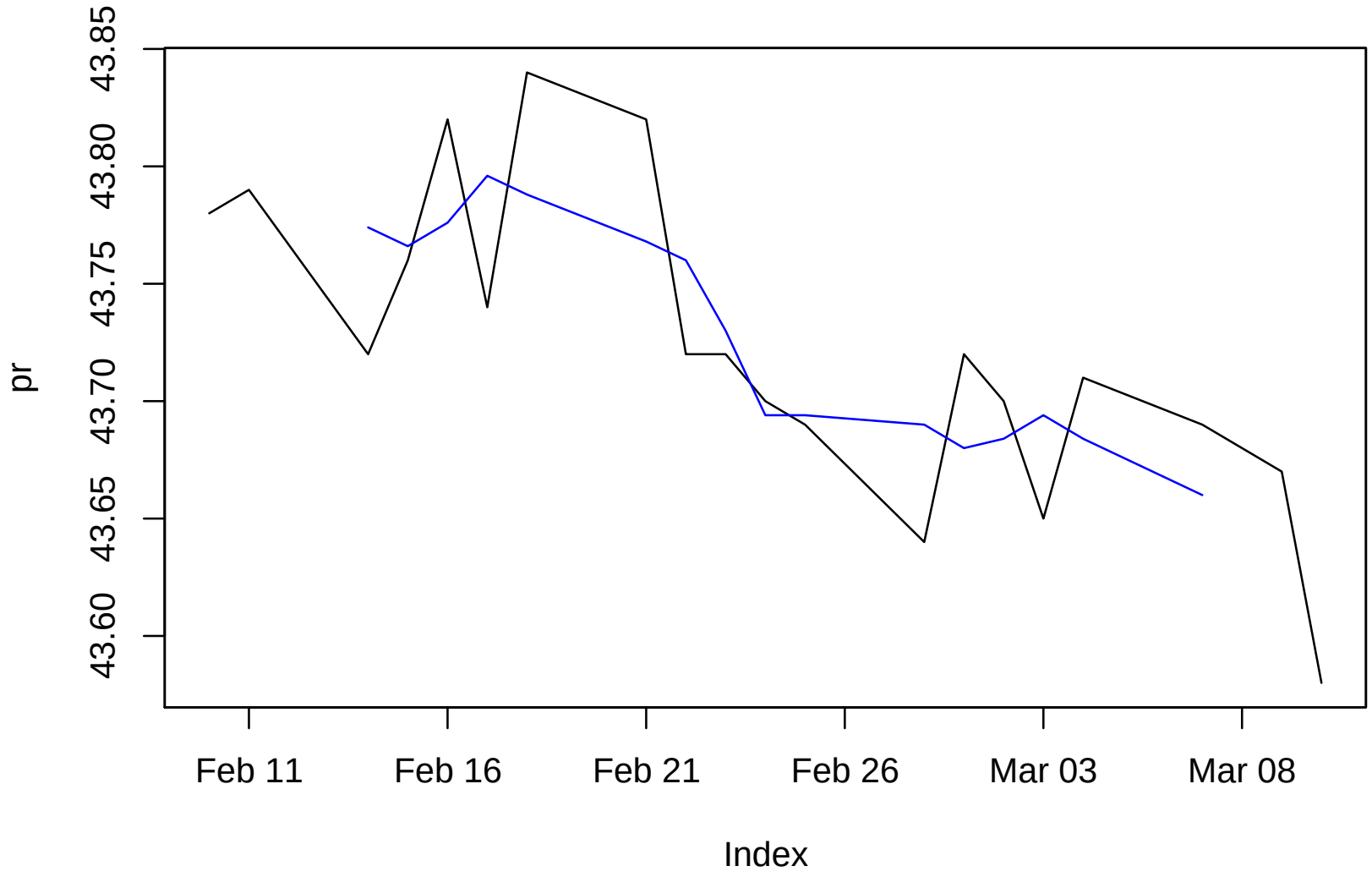
Time series: Implementations

Simple example of using **zoo** for data from an external text file. Each row looks like

```
10 Feb 2005|43.78
```

```
R> pr <- read.zoo(system.file(file.path("doc", "demo1.txt"),
+     package = "zoo"), sep = "|", format = "%d %b %Y")
R> plot(pr)
R> lines(rollapply(pr, 5, mean), col = 4)
```

Time series: Implementations



Time series: Implementations

```
R> rt <- 100 * diff(log(pr))
R> z <- merge(pr, rt)
R> window(z, end = as.Date("2005-02-15"))

               pr          rt
2005-02-10 43.78          NA
2005-02-11 43.79  0.02283887
2005-02-14 43.72 -0.15998175
2005-02-15 43.76  0.09144948
```

See Zeileis & Grothendieck (2005) and Shah, Zeileis, Grothendieck (2007) for more background information and hands-on examples.

More complex objects

Just as time/date classes are re-used in "zoo" series, these can be re-used in more complex objects.

Example: empirical fluctuation processes (original motivation for starting the **zoo** project)

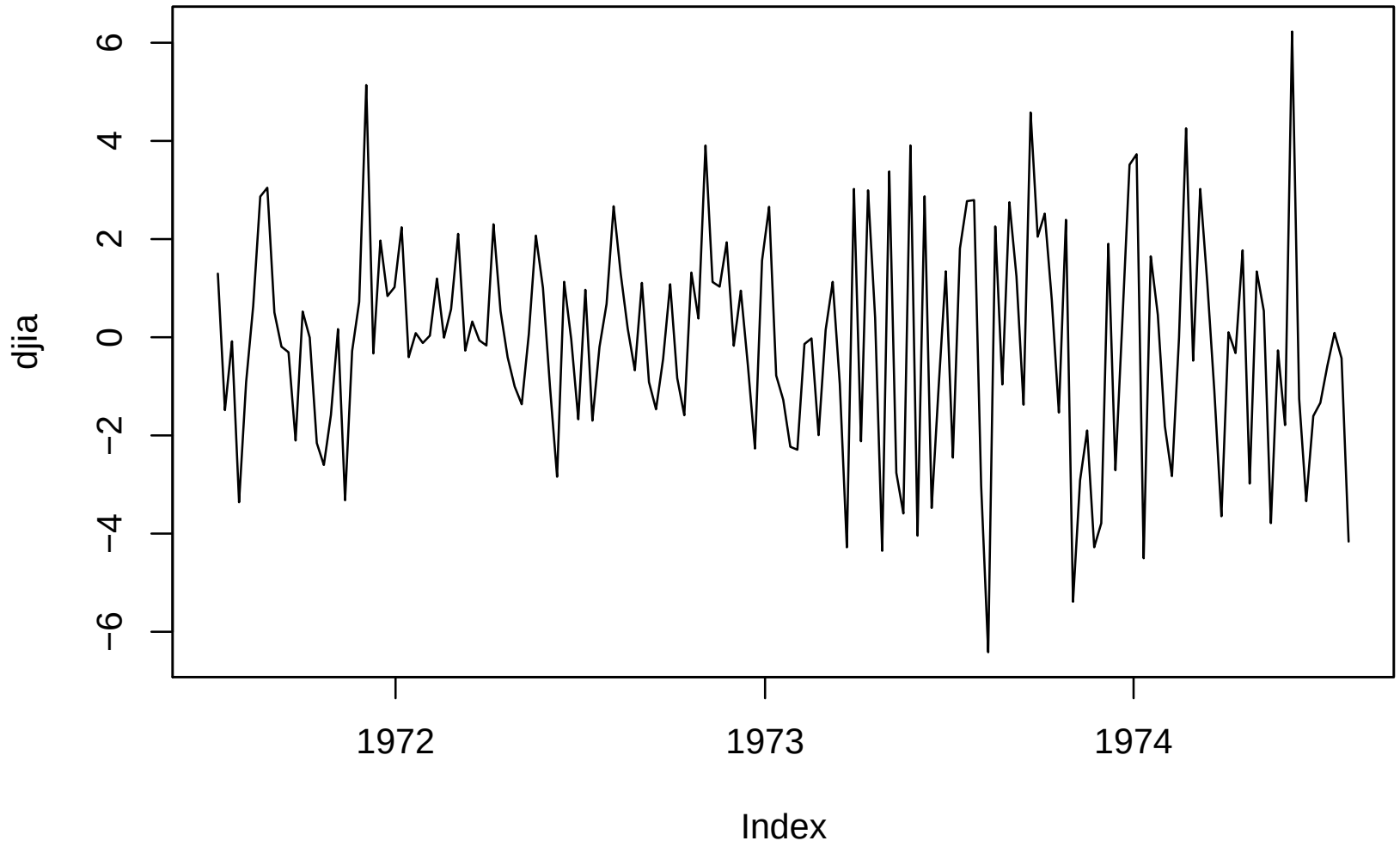
Idea: Empirical fluctuation processes capture systematic parameter instabilities in time-series models by computing a cumulative sum process of model deviations.

More complex objects

Application: changes in mean and/or variance of Dow Jones Industrial Average returns

```
R> djia <- 100 * diff(log(DJIA))
R> plot(djia)
R> djia_lm <- lm(djia ~ 1)
```

More complex objects



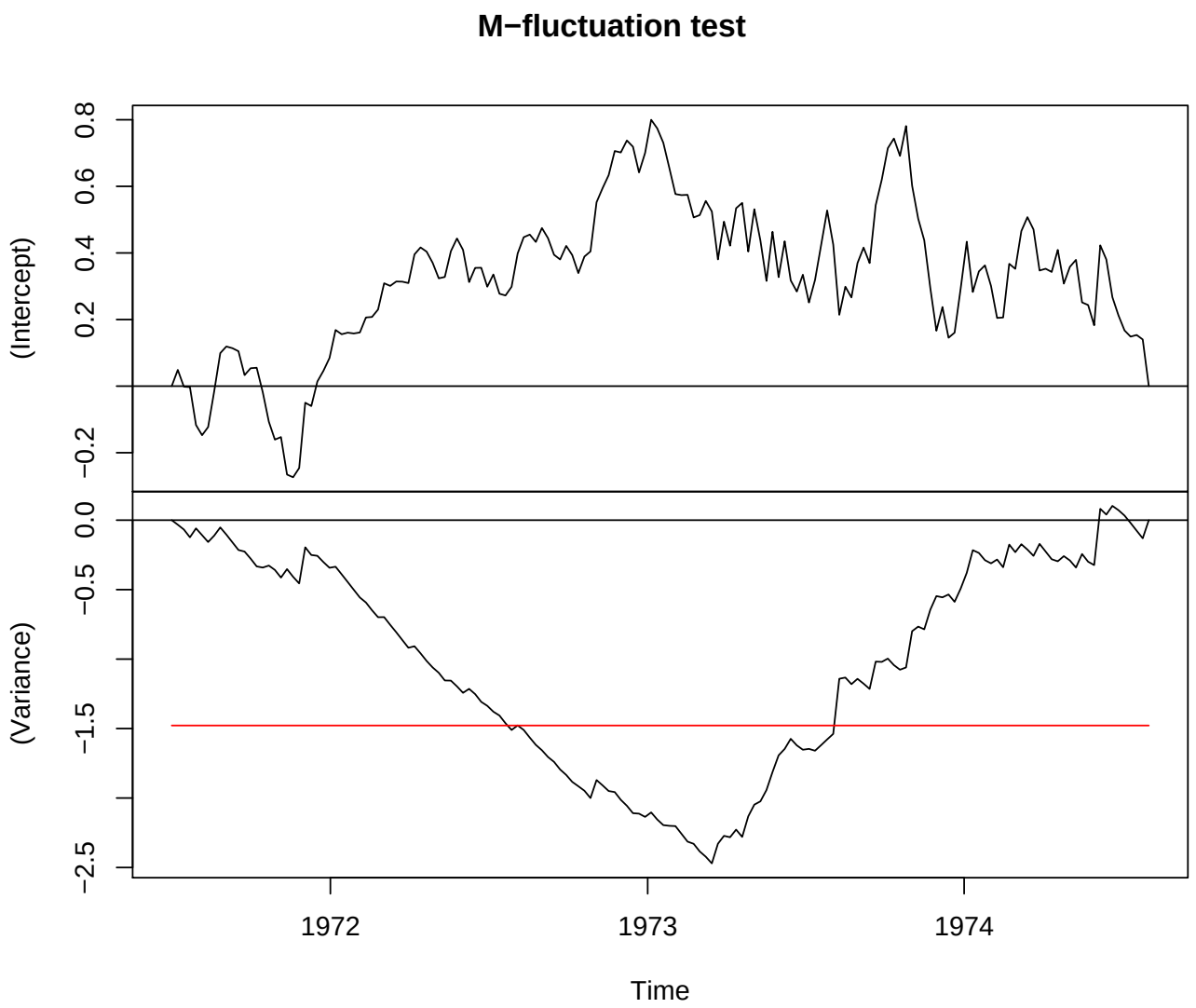
More complex objects

Capture deviations from overall mean (residuals) and variance (centered squared residuals):

```
R> mvscore <- function(obj) cbind("(Intercept)" = residuals(obj),
+     "(Variance)" = residuals(obj)^2 - mean(residuals(obj)^2))
R> djia_efp <- gefp(djia_lm, fit = NULL, scores = mvscore)
R> plot(djia_efp, aggregate = FALSE)
```

Here, there is a significant increase in the variance while the mean remains constant.

More complex objects



More complex objects

The empirical fluctuation process object contains several meta-informations such as the fitted model or the actual process (among many others):

```
R> class(djia_efp)

[1] "gefp"


R> names(djia_efp)

 [1] "process"      "nreg"         "nobs"         "call"
 [5] "fit"          "scores"       "fitted.model" "par"
 [9] "lim.process"  "type.name"    "order.name"   "J12"
```

More complex objects

The object is completely object-oriented concerning the model class tested (see Zeileis, 2005). Here, a simple "`lm`" object is re-used. Other components are re-used and stored as well:

```
R> class(djia_efp$process)

[1] "zoo"


R> class(djia_efp$fitted.model)

[1] "lm"


R> class(djia_efp$scores)

[1] "function"
```

Summary and outlook

Available data structures:

- many time/date classes available,
- working with time zones and daylight savings times requires care,
- flexible time-series class(es) available,
- many suitable generics for typical tasks.

Summary and outlook

Creating new data structures:

- if possible, re-use existing structures for creating new ones,
- functional interfaces can keep approaches modular,
- identify typical tasks and suitable R representations/functions for them,
- R objects (including functions) should reflect what we think the underlying entities do conceptually.

Summary and outlook

Required data structures:

- meta information objects for financial information,
- annotated time series, e.g., stock prices/returns or exchange rates,
- portfolios,
- ...

References

Grothendieck G, Petzoldt T (2004). "R Help Desk: Date and Time Classes in R." *R News*, **4**(1), 29–32.

Ripley BD, Hornik K (2001). "Date-Time Classes." *R News*, **1**(2), 8–11.

Shah A, Zeileis A, Grothendieck G (2007). "**zoo** Quick Reference." Package vignette. Version 1.3-2.

Zeileis A (2005). "Implementing a Class of Structural Change Tests: An Econometric Computing Approach." *Computational Statistics & Data Analysis*, **50**(11), 2987–3008. doi:10.1016/j.csda.2005.07.001

Zeileis A, Grothendieck G (2005). "**zoo**: S3 Infrastructure for Regular and Irregular Time Series." *Journal of Statistical Software*, **14**(6), 1–27. URL http://www.jstatsoft.org/v14/i06/. Updated version contained as vignette in **zoo** package.