seasonal: R interface to X-13ARIMA-SEATS

Christoph Sax

May 2, 2017

1 Introduction

seasonal is an easy-to-use and full-featured R-interface to X-13ARIMA-SEATS, the newest seasonal adjustment software developed by the United States Census Bureau. X-13ARIMA-SEATS combines and extends the capabilities of the older X-12ARIMA (developed by the Census Bureau) and TRAMO-SEATS (developed by the Bank of Spain).

seasonal depends on the x13binary package (by Dirk Eddelbuettel and Christoph Sax) to access pre-built binaries of X-13ARIMA-SEATS for all platforms and **does not require any manual installation**.

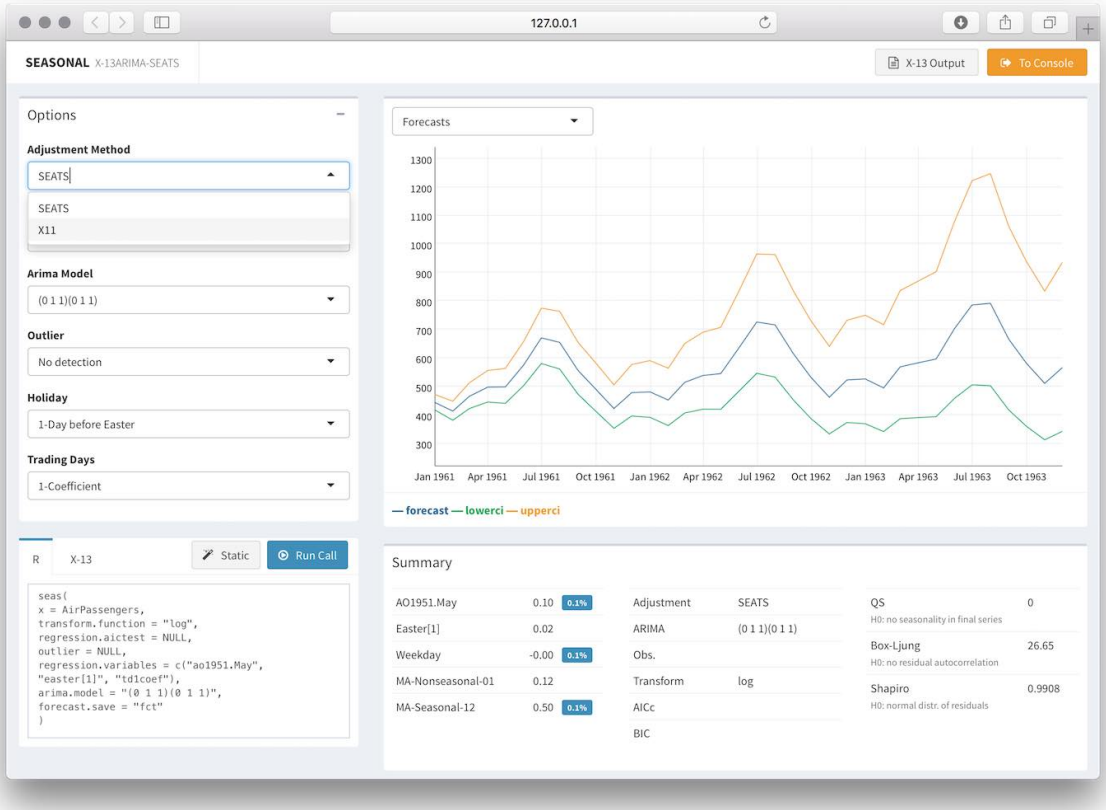

Figure 1: Graphical user interface for X-13

If you are new to seasonal adjustment or X-13ARIMA-SEATS, the automated procedures of *seasonal* allow you to quickly produce good seasonal adjustments of time series. Start with the Installation and Getting started section and skip the rest. Alternatively, `demo(seas)` gives an overview of the package functionality.

If you are familiar with X-13ARIMA-SEATS, you may benefit from the flexible input and output structure of *seasonal*. The package allows you to use (almost) all commands of X-13ARIMA-SEATS, and it can import (almost) all output generated by X-13ARIMA-SEATS. The only exception is the 'composite' spec, which is easy to replicate in basic R. Read the Input and Output sections and have a look at the website of seasonal, where the examples from the official X-13ARIMA-SEATS manual are reproduced in R.

seasonal includes a graphical user interface that facitlitates the use of X-13 both for beginners and advanced users. The final sections of this vignette cover additional topics: User defined holidays, such as Chinese New Year, the use of seasonal for production, and the import of existing X-13 model specs to R.

2 Installation

seasonal relies on the x13binary package to access prebuilt binaries of X-13ARIMA-SEATS. To install both packages, type to the R console:

```
install.packages("seasonal")
```

See the documentation of `?seasonal` if you want to set the path to X-13 manually.

3 Getting started

seas is the core function of the *seasonal* package. By default, seas calls the automatic procedures of X-13ARIMA-SEATS to perform a seasonal adjustment that works well in most circumstances:

```
m <- seas(AirPassengers)
```

The first argument of `seas` has to be a time series of class `"ts"`. The function returns an object of class `"seas"` that contains all necessary information on the adjustment.

There are several functions and methods for `"seas"` objects: The `final` function returns the adjusted series, the `plot` method shows a plot with the unadjusted and the adjusted series. The `summary` method allows you to display an overview of the model:

```
final(m)
plot(m)
summary(m)
```

By default, `seas` calls the SEATS adjustment procedure. If you prefer the X11 adjustment procedure, use the following option (see the Input section for details on how to use arbitrary options with X-13):

```
seas(AirPassengers, x11 = "")
```

A default call to `seas` also invokes the following automatic procedures of X -13ARIMA-SEATS:

- Transformation selection (log / no log)
- Detection of trading day and Easter effects
- Outlier detection
- ARIMA model search

Alternatively, all inputs may be entered manually, as in the following example:

```
seas(x = AirPassengers,
     regression.variables = c("td1coef", "easter[1]", "ao1951.May"),
     arima.model = "(0 1 1)(0 1 1)",
     regression.aictest = NULL,
     outlier = NULL,
     transform.function = "log")
```

The `static` command returns the manual call of a model. The call above can be easily generated from the automatic model:

```
static(m)
static(m, coef = TRUE)  # also fixes the coefficients
```

If you have seasonalview installed, the `view` command offers an easy way to analyze and modify a seasonal adjustment procedure (see the section below for details):

```
view(m)
```

4 Input

In *seasonal*, it is possible to use almost the complete syntax of X-13ARIMA- SEATS. This is done via the `...` argument in the `seas` function. The X -13ARIMA-SEATS syntax uses *specs* and *arguments*, with each spec optionally containing some arguments. These spec-argument combinations can be added to `seas` by separating the spec and the argument by a dot (`.`). For example, in order to set the 'variables' argument of the 'regression' spec equal to `td` and `ao1999.jan`, the input to `seas` looks like this:

```
m <- seas(AirPassengers, regression.variables = c("td", "ao1955.jan"))
```

Note that R vectors may be used as an input. If a spec is added without any arguments, the spec should be set equal to an empty string (or, alternatively, to an empty list, as in early versions). Several defaults of `seas` are empty strings, such as the default `seats = ""`. See the help page (`?seas`) for more details on the defaults. Note the difference between `""` (meaning the spec is enabled but has no arguments) and `NULL` (meaning the spec is disabled).

It is possible to manipulate almost all inputs to X-13ARIMA-SEATS in this way. The best way to learn about the relationship between the syntax of X-13ARIMA- SEATS and *seasonal* is to study the comprehensive list of examples. For instance, example 1 in section 7.1 from the manual,

```
series { title  =  "Quarterly Grape Harvest" start = 1950.1
       period =  4
       data   = (8997 9401 ... 11346) }
arima { model = (0 1 1) }
estimate { }
```

translates to R in the following way:

```
seas(AirPassengers,
     x11 = "",
     arima.model = "(0 1 1)"
)
```

`seas` takes care of the 'series' spec, and no input beside the time series has to be provided. As `seas` uses the SEATS procedure by default, the use of X11 has to be specified manually. When the 'x11' spec is added as an input (like above), the mutually exclusive and default 'seats' spec is automatically disabled. With `arima.model`, an additional spec-argument is added to the input of X-13ARIMA-SEATS. As the spec cannot be used in the same call as the 'automdl' spec, the latter is automatically disabled.

There are some mutually exclusive specs in X-13ARIMA-SEATS. If more than one mutually exclusive spec is included in `seas`, specs are overwritten according the following priority rules:

- Model selection
    1. `arima`
    2. `pickmdl`
    3. `automdl` (default)
- Adjustment procedure

1. `x11`
2. `seats` (default)

As an alternative to the `...` argument, spec-arguments can also be supplied as a named list. This is useful for programming:

```
seas(list = list(x = AirPassengers, x11 = ""))
```

Additionally, `"seas"` objects can be altered using the `update` method. It uses the same syntax as the `seas` function. The following example turns off trading day adjustment and switches the adjustment method to X-11:

```
update(m, regression.variables = "td", x11 = "")
```

A common use of `update` involves the recomputing with an `x` argument containing new data:

```
update(m, x = sqrt(AirPassengers))
```

Lastly, a `predict` method can be used to extract the final series from an updated call, using the familar `newdata` argument:

```
predict(m, newdata = sqrt(AirPassengers))
```

5 Output

seasonal has a flexible mechanism to read data from X-13ARIMA-SEATS. With the `series` function, it is possible to import almost all output that can be generated by X-13ARIMA-SEATS. For example, the following command returns the forecasts of the ARIMA model as a `"ts"` time series:

```
m <- seas(AirPassengers)
series(m, "forecast.forecasts")
```

Because the `forecast.save = "forecasts"` argument has not been specified in the model call, `series` re-evaluates the call with the 'forecast' spec enabled. It is also possible to return more than one output table at the same time:

```
series(m, c("forecast.forecasts", "s12"))
```

You can use either the unique short names of X-13 (such as `d1`), or the long names (such as `forecasts`). Because the long table names are not unique, they need to be combined with the spec name (`forecast`). See `?series` for a complete list of options.

Note that re-evaluation doubles the overall computation time. If you want to speed it up, you have to be explicit about the output in the model call:

```
m <- seas(AirPassengers, forecast.save = "forecasts")
series(m, "forecast.forecasts")
```

Some specs, like 'slidingspans' and 'history', are time consuming. Re-evaluation allows you to separate these specs from the basic model call:

```
m <- seas(AirPassengers)
series(m, "history.saestimates")
series(m, "slidingspans.sfspans")
```

The `udg` function provides access to a large number of diagnostical statistics:

```
udg(m, "x13mdl")
```

If you are using the HTML version of X-13, the `out` function shows the content of the main output in the browser:

```
out(m)
```

6 Graphs

There are several graphical tools to analyze a `seas` model. The main plot function draws the seasonally adjusted and unadjusted series, as well as the outliers. Optionally, it also draws the trend of the seasonal decomposition:

```
m <- seas(AirPassengers, regression.aictest = c("td", "easter"))
plot(m)
```

The `monthplot` function allows for a monthwise plot (or quarterwise, with the same function name) of the seasonal and the SI component:

```
monthplot(m)
monthplot(m, choice = "irregular")
```

Also, many standard R function can be used to analyze a `"seas"` model:

```
pacf(resid(m))
spectrum(diff(resid(m)))
plot(density(resid(m)))
qqnorm(resid(m))
```

The `identify` method can be used to select or deselect outliers by point and click. Click several times to loop through different outlier types.

```
identify(m)
```

7 Graphical User Interface

The `view` function is a graphical tool for choosing a seasonal adjustment model, using the new seasonalview package, with the same structure as the demo website of seasonal. To install seasonalview, type:

```
install.packages("seasonalview")
```

The goal of `view` is to summarize all relevant options, plots and statistics that should be usually considered. `view` uses a `"seas"` object as its main argument:

```
view(m)
```

Frequently used options can be modified using the drop-down selectors in the upper left panel. Each change will result in a re-estimation of the seasonal adjustment model. The R-call, the output and the summary are updated accordingly.

Alternatively, the R-Call can be modified manually in the lower left panel. Press 'Run Call' to re-estimate the model and to adjust the options selectors, the output, and the summary. With the 'To Console' button, `view` is closed and an updated model returned to the R console. The 'Static' button substitutes automatic procedures by their corresponding static spec-argument options, in the same way as the `static` function.

The views in the upper right panel can be selected from the drop-down menu.

The lower right panel shows the summary, including the same information as the `summary` method. The 'X-13 output' button opens the complete output of X-13 in a separate tab or window.

8 Chinese New Year, Indian Diwali and other customized holidays

seasonal includes `genhol`, a function that makes it easy to model user-defined holiday regression effects. `genhol` is an R replacement for the equally named software by the Census Office; no additional installation is required. The function uses an object of class `"Date"` as its first argument, which specifies the occurrence of the holiday.

In order to adjust Indian industrial production for Diwali effects, use, e.g.,:

```
# variables included in seasonal
# iip: Indian industrial production
# cny, diwali, easter: dates of Chinese New Year, Indian Diwali and Easter

seas(iip,
     x11 = "",
     xreg = genhol(diwali, start = 0, end = 0, center = "calendar"),
     regression.usertype = "holiday")
```

For more examples, including Chinese New Year and complex pre- and post-holiday adjustments, see `?genhol`.

9 Production use

While *seasonal* offers a quick way to adjust a time series in R, it is equally suited for the recurring processing of potentially large numbers of time series. There are two kind of seasonal adjustments in production use:

1. a periodic application of an adjustment model to a time series
2. an automated adjustment to a large number of time series

This section shows how both tasks can be accomplished with *seasonal* and basic R.

9.1 Storing models and batch processing

Seasonal adjustment models can be stored (using `save`) and re-evaluated at a later date when new data becomes available.

```
ap.short <- window(AirPassengers, end = c(1959, 12))
m <- seas(ap.short)
```

The `static` function comes particularly handy to save the specifics of a model (e.g., outliers, ARIMA model) for future use. Often, it is desirable to store the following 'static' version of the model where the default automatic procedures in the model call are substituted by the choices they made. For example, in the model above, the automated procedures decided to perform a logarithmic pre- transformation of the data, to use an (0 1 1)(0 1 1) ARIMA model, to use trading day correction and to add an additive outlier in May, 1951. The static function 'hard-codes' these findings into the model call, so that these decisions would not be re-evaluated at a later date:

```
m.static <- static(m, evaluate = TRUE)
```

Either the static or the automated model can be re-evaluated when new data becomes available:

```
update(m, x = AirPassengers)
update(m.static, x = AirPassengers)
```

9.2 Automated adjustment of large numbers of series

X-13 can also be applied to a large number of series, using automated adjustment methods. This can be accomplished with a loop or an apply function. It is useful to wrap the call to `seas` in a `try` statement; that way, an error will not break the execution. You need to develop an error handling strategy for these cases: You can either drop them, use them without adjustment or switch to a different automated routine.

```
# collect data
dta <- list(fdeaths = fdeaths, mdeaths = mdeaths)

# loop over dta
l1 <- lapply(dta, function(e) try(seas(e, x11 = "")))

# list failing models
is.err <- sapply(l1, class) == "try-error"
l1[is.err]

# return final series of successful evaluations
do.call(cbind, lapply(l1[!is.err], final))
```

If you have several cores and want to speed things up, the process is well suited for parallelization:

```
# a list with 100 time series
l2 <- rep(list(AirPassengers), 100)

library(parallel)  # this is part of a standard R installation
```

If you are on Windows or want to use cluster parallelization, use `parLapply`:

```
# set up cluster
cl <- makeCluster(detectCores())

# load 'seasonal' for each node
clusterEvalQ(cl, library(seasonal))

# export data to each node
clusterExport(cl, varlist = "l2")

# run in parallel (2.2s on a 4-core / 8-thread Macbook, vs 9.6s with standard lapply)
parLapply(cl, l2, function(e) try(seas(e, x11 = "")))

# finally, stop the cluster
stopCluster(cl)
```

On Linux or OS-X, 'forking' parallelization allows you to do the same in a single line:

```
mclapply(l2, function(e) try(seas(e, x11 = "")), mc.cores = detectCores())
```

10 Import X-13 models and series

Two experimental utility functions allow you to import `.spc` files and X-13 data files from any X-13 set-up. Simply locate the path of your X-13 `.spc` file, and the `import.spc` function will construct the corresponding call to `seas` as well as the calls for importing the data.

```
# importing the orginal X-13 example file
import.spc(system.file("tests", "Testairline.spc", package="seasonal"))
```

If data is stored outside the `.spc` file (as it usually will be), the calls will make use of the `import.ts` function, which imports arbitrary X-13 data files as R time series. See `?import.ts` for examples.

11 License and Credits

seasonal is free and open source, licensed under GPL-3. It requires the X -13ARIMA-SEATS software by the U.S. Census Bureau, which is open source and freely available under the terms of its own license.

seasonal has been originally developed for the use at the Swiss State Secretariat of Economic Affairs. It has been greatly improved over time thanks to suggestions and support from Matthias Bannert, Freya Beamish, Vidur Dhanda, Alain Galli, Ronald Indergand, Preetha Kalambaden, Stefan Leist, James Livsey, Brian Monsell, Pinaki Mukherjee, Bruno Parnisari, and many others. I am especially grateful to Dirk Eddelbuettel for the fantastic work on the x13binary package.

Please report bugs and suggestions on Github or send me an e-mail. Thank you!