X-13 Stuff You Should Know

Brian C. Monsell and Osbert Pang

Brian.C.Monsell@census.gov

SAPW 2018
April 26, 2018

Any views expressed are those of the author(s) and not necessarily those of the U.S. Census Bureau.

United States Census Bureau
U.S. Department of Commerce
Economics and Statistics Administration
U.S. CENSUS BUREAU
census.gov

Outline

- Seasonal adjustment production in R
  - Continued from SAPW 2016
- *qcheck* argument in *spectrum* spec
- A feature of the X-13ARIMA-SEATS HTML output
  - How to read seasonal factors directly from HTML output

The problem (from last time)

- The Center for Economic Studies was planning on publishing seasonally adjusted estimates for quarterly Business Formation Series (BFS)
  - 12 types of series
  - In each type, there is an estimate for each state, the District of Columbia, and the total for the US
- Need seasonally adjusted series, graphs
- Need a quick turnaround time

All analysis done with R

- Data stored in list objects
- For “annual review”, automatic model and filter selection done
  - A list of X-13 objects is produced using the `seasonal` package
- Diagnostic summary is generated
  - Modeling and adjustment options can be refined based on the diagnostics
- A “static” version of the seasonal objects are generated to produce final seasonal adjustments.
- Graphs of the seasonal adjustments are generated
- Final adjustments saved to Excel files

R seasonal package

- Allows users to run X-13ARIMA-SEATS with R
- Eliminates many of the external files generated by X-13ARIMA-SEATS
- Data and diagnostic information can be stored in efficient data structures within R

Example

```
# load seasonal package
library("seasonal”)
Sys.setenv(X13_PATH = "h:/x13ashtml")
checkX13()

# run Airline Series,
# do X-11 seasonal adjustment
m <- seas(AirPassengers, x11="")
# examine output file for run
out(m)
```

R functions for seasonal adjustment production

- Osbert and I have developed many R functions for this project that:
  - access and summarize the diagnostic information using the `udg()` function.
  - save seasonal arguments for an individual series,
  - save a seas object to an external spec file,
  - determines for which series `seas()` has failed.
- Functions available upon request.

Two problems

- Saving final results to Excel files
- Saving final spec files for another group's use

Saving final adjustments to Excel files

- CES wanted separate Excel files for state, US series for each group for distribution
  - Need separate worksheets for the seasonally adjusted and original series
- We needed a package that would write Excel spreadsheets that could create individual workbooks.
- Used the `xlsx` package
  - Writes R data structures to individual workbooks
  - Requires `rJava`, `xlsxjars`

Saving final adjustments to Excel files in R

- Function that generates the Excel files for each of the groups
- Create separate data frames for the original series and the seasonally adjusted series
  - Have separate data frames for the state series, US national numbers
  - First column: Quarter, Year of each observations
  - State data frames: column for each state series
- Generate Excel files, using `xlsx` package
  - Use `write.xlsx` function to write each worksheet

```
write.xlsx(a1st, stateFile, sheetName = 'BA_SA', row.names = F)
write.xlsx(a2st, stateFile, sheetName = 'BA_NSA', append = T,
        row.names = F)
```

```
# generic output function that splits off US data and stores it in a separate Excel file

outputSplit <- function(X.data, Y.models, stem, thisDir = 'March2018', thisExt='q4y17') {
  # data.list to X.data
  # list of (final) seas objects to Y.models
  # group name as string to stem

  require(xlsx)
  require(seasonal)
  require(zoo)

# create matrix of seasonally adjusted series (a1), original data (a2)
  a1 <- do.call(cbind, lapply(Y.models, final))
  a2 <- do.call(cbind, X.data)

# create data frames from matrices, adding a column for the time of each obs.
  a1 <- data.frame(Time = yearqtr(time(a1)), a1)
  a2 <- data.frame(Time = yearqtr(time(a2)), a2)
```

```
#  Generate data frames containing only the states
  a1st <- a1[c(-53)]
  a2st <- a2[c(-53)]

#  Generate data frames containing only the US total
  ivec <- c("Time","us")
  a1us <- a1[ivec]
  a2us <- a2[ivec]

#  Generate file names
  stateFile <- sprintf('N:/timeseriescsrm/%s/out.%s/%s_ST.xlsx', thisDir, thisExt, stem)
  USFile <- sprintf('N:/timeseriescsrm/%s/out.%s/%s_US.xlsx', thisDir, thisExt, stem)

#  Generate excel files for SA and original data, each in their own worksheet
  write.xlsx(a1st, stateFile, sheetName = sprintf('%s_SA', stem), row.names = F)
  write.xlsx(a2st, stateFile, thisDir, thisExt, stem), sheetName = sprintf('%s_NSA', stem),
                  append = T, row.names = F)
  write.xlsx(a1us, USFile, sheetName = sprintf('%s_SA', stem), row.names = F)
  write.xlsx(a2us, USFile, sheetName = sprintf('%s_NSA', stem), append = T, row.names = F)
}
```

Saving final spec files

- We received a request for spec files equivalent to the adjustments done in R
- Used the sink function to generate the files
  - However, entries for the `file` and `title` arguments of the `series` spec reflect the temporary directory used by seasonal
  - Can read the generated spec into a character array.
  - Use the `grep` function to find the lines with the `file` and `title` arguments, replace them with more useful values.

```
#  saveMetaFile is a function that generates an X-13 meta file and returns a list
#  of the names of the series stored in the metafile
DUR8Q.names <-
   saveMetaFile("DUR8Q", "N:/timeseriesCSRM/March2018/test/DUR8Q", "_", "lauto")

setwd("N:/timeseriesCSRM/March2018/test/DUR8Q")

for (i in DUR8Q.names) {
  x <- DUR8Q.q4y17.data.list[[i]]
  sink(sprintf("DUR8Q_%s.spc", i))
  print(spc(eval(static(DUR8Q.lauto[[i]], x11.filter = T, test = F))))
  sink()
}
for (fn in list.files(getwd(), 'DUR8Q.+spc')) {
  aa <- readLines(fn)
  stem <- substr(fn, 1, nchar(fn) - 4)
  aa[grep('title', aa)] <- sprintf("  title = \"Final X-13 run for %s\"", stem)
  aa[grep('file', aa)] <- sprintf("  file = \"%s.dat\"", stem)
  writeLines(aa, fn)
}
```

`qcheck` argument

- There are situations where
  - the monthly version of a series is not seasonal,
  - but the quarterly series is.
- To test for this condition, the `qcheck` argument was added to the `spectrum` spec
  - Generates a quarterly version of the monthly series
  - Generates the QS statistic for the quarterly original series, seasonally adjusted series

QS calculation

- Find the lag s and 2s (12&24 or 4&8) autocorrelations, r_s and r_{2s} of the differenced series
- If r_s is negative, then QS is 0. Else

$$QS = n(n+2)\left\{\frac{{r_s}^2}{n-s} + \frac{{R_{2s}}^2}{n-2s}\right\}$$

where R_{2s} is r_{2s} if r_{2s} >0, 0 o.w.

and n = series length – differencing order

- QS ~ X^2 with 2 degrees of freedom

Generating monthly, quarterly QS with R

- If you have data in a list object, use `lapply` to generate seas objects for all the series in the list
  - Need to add the argument `spectrum.qcheck = "yes"` to the call to seas.
- Use the `udg` function to gather QS statistics for monthly and quarterly series
  - For monthly QS : `udg(x, "qsori")`
  - For quarterly QS: `udg(x, "qsori.qseas")`
  - Need to use lappy to apply udg function to every element of the list
- Can use `xlsx` package to save to Excel files

Generating results of qcheck

```
#  X-13 runs on FTD data list, using qcheck
lauto <- lapply(ftd.data.list, function(x) try(seas(x, x11 = "", spectrum.qcheck = "yes",
   slidingspans=NULL, history=NULL)))

#  create matrix of QS diagnostics for monthly and quarterly original series
ftd.qsori <-
  cbind(matrix(unlist(lapply(lauto, function(x) try(udg(x,"qsori")))), ncol=2,
            byrow = TRUE),
        matrix(unlist(lapply(lauto, function(x) try(udg(x,"qsori.qseas")))), ncol=2,
            byrow = TRUE))
ftd.qssori <-
  cbind(matrix(unlist(lapply(lauto, function(x) try(udg(x,"qssori")))), ncol=2,
             byrow = TRUE),
        matrix(unlist(lapply(lauto, function(x) try(udg(x,"qssori.qseas")))), ncol=2,
             byrow = TRUE))
```

Saving results of qcheck

```
#  create dimension names for matrixes
dimnames(ftd.qsori)[[1]]<-names(ftd.data.list)
dimnames(ftd.qsori)[[2]]<-c("qsori","qsori.pv","qsori.qseas","qsori.qseas.pv")

dimnames(ftd.qssori)[[1]]<-names(ftd.data.list)
dimnames(ftd.qssori)[[2]]<-c("qssori","qssori.pv","qssori.qseas","qssori.qseas.pv")

# save QS matrixes to Excel files
require(xlsx)
write.xlsx(ftd.qsori,"ftd.qs.xlsx",sheetName = "qsori", row.names = TRUE)
write.xlsx(ftd.qssori,"ftd.qs.xlsx",sheetName = "qssori", row.names = TRUE, append=TRUE)
```

X-13 tables in HTML output

- Each table is encapsulated within a set of `<div>` tags with that has a unique name
- This creates an entry in the Document Object Model (DOM) for the HTML file
  - Allows access to the contents of the named `<div>` tag
  - The R `xml2` package can be useful for this
- Problem:
  - Read seasonal factors directly from the HTML output

```
<div id="otl">
<table class="x11"
  summary ="A 8  RegARIMA combined outlier component" >
<caption><strong>A 8  RegARIMA combined outlier component</strong></caption>
<tr>
<td class="head"> </td>
<th scope="col"><abbr title="January">    Jan           </abbr></th>
<th scope="col"><abbr title="February">    Feb           </abbr></th>
<th scope="col"><abbr title="March">    Mar           </abbr></th>
<th scope="col"><abbr title="April">    Apr           </abbr></th>
<th scope="col">    May           </th>
<th scope="col"><abbr title="June">    Jun           </abbr></th>
<th scope="col"><abbr title="July">    Jul           </abbr></th>
<th scope="col"><abbr title="August">    Aug           </abbr></th>
<th scope="col"><abbr title="September">    Sep           </abbr></th>
<th scope="col"><abbr title="October">    Oct           </abbr></th>
<th scope="col"><abbr title="November">    Nov           </abbr></th>
<th scope="col"><abbr title="December">    Dec           </abbr></th>
<th scope="col"><abbr title="Average                    "> AVGE</abbr></th>
</tr>
```

One Solution

- Use the R package `xml2` to create an R time series object with the seasonal factors
- Use `xml.read` to read the HTML file into an R object
- Use `xml.children` to locate the node of the div tag that contains the seasonal factors
- Use `xml.text` to extract the text from that node

```
> library("xml2", lib.loc="~/R/win-library/3.3")
> ta <- read_xml("c:/x13ashtml/testairline.html")

> xml_children(ta)
{xml_nodeset (2)}
[1] <head>\n  <title>testairline.html</title>\n  <meta name="keywords” ...
[2] <body>\n  <div id="content">\n    <p>Reading input spec file from ...

> xml_children(xml_children(ta)[2])
{xml_nodeset (2)}
[1] <div id="content">\n  <p>Reading input spec file from ...
[2] <div id="rightnavigation">\n  <a name="index"/>\n  ...
```

```
> xml_children(xml_children(xml_children(ta)[2])[1])
{xml_nodeset (25)}
 ...
 [7] <h3>D 10  Final seasonal factors</h3>
 [8] <p>From  1952.Jan to 1960.Dec<br/>\n  Observations       108</p>
 [9] <p>Seasonal filter : 3 x 3 moving average</p>
[10] <div id="d10">\n  <table class="x11" summary="D 10  Final seasonal factors"> ...
[11] <p class="center">\nTable Total-            10804.     ...
[12] <p> </p>
[13] <h3>D 10.A  Final seasonal component forecasts</h3>
...

> xml_children(xml_children(xml_children(ta)[2])[1])[10]
{xml_nodeset (1)}
[1] <div id="d10">\n  <table class="x11" summary="D 10  Final seasonal factors">\n  ...
```

> xml_text(xml_children(xml_children(xml_children(ta)[2])[1])[10])

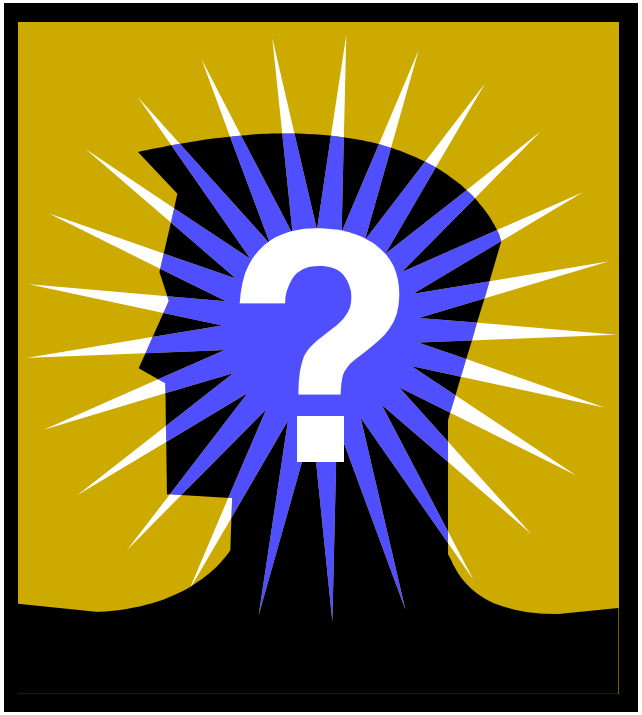
[1] "D 10 Final seasonal factors   Jan Feb Mar
Apr May Jun Jul Aug Sep
Oct Nov Dec AVGE1952 89.52 88.59 102.96 97.61
98.13 112.63 123.82 119.30 104.82 92.62 79.84 90.26 100.011953 89.53 88.53
102.66 97.55 98.11 112.47 124.18 119.49 105.31 92.31 79.89 90.21 100.021954
89.83 88.20 101.75 97.47 98.00 112.40 124.72 120.29 105.79 91.90 80.08 89.93
100.031955 90.46 87.55 100.37 97.24 97.87 112.51 125.29 121.53 106.26 91.84
80.11 89.58 100.051956 91.04 86.67 98.77 96.92 97.64 112.81 126.06 123.21
106.18 92.26 80.16 88.88 100.051957 91.19 85.91 97.65 96.51 97.52 112.87
127.03 124.64 106.24 92.77 80.04 88.37 100.061958 90.87 85.26 96.74 96.38
97.46 112.86 128.04 125.85 106.08 93.16 79.99 87.86 100.051959
90.41 84.77 96.17 96.43... <truncated>

One Solution

- Extract the time series from the text string
  - Use `strsplit` to create an array of text split on " "
  - Filter the empty string elements out of the array
  - Find the range of the seasonal factors
  - Pick just the factors into another array, avoiding the entries for the year and average of the seasonals
  - Convert that array into real numbers
  - Convert this object to a time series object

```
ta <- read_xml("c:/x13ashtml/testairline.html")
xx <- xml_children(xml_children(xml_children(ta)))
sf.table <- xml_text(xml_children(xx[10]))

sfList <- unlist(strsplit(sf.table,split=" "))
thisFilter <- sfList %in% ""
sfListFiltered <- sfList[!thisFilter]

iVec <- which(sfListFiltered %in% grep('AVGE',sfListFiltered, value=TRUE))
tempVec <- sfListFiltered[seq(iVec[1], iVec[2]-2)]

startSf <- c(as.integer(substr(tempVec[1], 5, 8)),1)

thisSf <- ts(as.double(as.vector(matrix(tempVec[1:length(tempVec)],
   byrow=TRUE)[,2:13])), start=startSf, frequency = 12)
```

Questions?



Brian C. Monsell
Email : brian.c.monsell@census.gov

Osbert C. Pang
Email : osbert.c.pang@census.gov