The cruncher and the package JDCruncher

Anna Smyk (Insee)

The cruncher (1/2)

The cruncher is an additional "executable" module, not an R package. It can be launched via R, SAS. . .

Objective of the cruncher : update a JDemetra+ workspace and export the results (series and diagnostics), without having to open the graphical interface and operate manually.

Suitable for a production process.

Some links :

- to download the cruncher
  https://github.com/jdemetra/jwsacruncher/releases
- the associated help
  https://github.com/jdemetra/jwsacruncher/wiki

The cruncher (2/2)

To automatically update your workspace you would need :

- the cruncher
- a file containing :
  - parameters of the refresh policy used to updated the workspace
  - export parameters ;
- a valid JDemetra+ workspace.

To launch the cruncher in a simpler way (without the parameter file) you can use an R package

- rjwsacruncher on CRAN : update of the workspace and output production
- JDCruncheR : a quality report is added (not on CRAN yet as originally in French, but now in English on InseeFr GitHub)

Sommaire

Installation of the JDCruncheR package

The package is available here : https://github.com/InseeFr/JDCruncheR.

To install it : download the **.zip** or **.tar.gz** file from
https://github.com/InseeFr/JDCruncheR/releases.

Some functions require the packages XLConnect and XML :

```
install.packages(c("XLConnect", "XML"))
```

Using JDCruncheR (1/3)

To load the package :

```
library(JDCruncheR)
```

Using JDCruncheR (2/3)

Three configuration options :
- default_matrix_item (diagnostics to export)
- default_tsmatrix_series (time series to export)
- cruncher_bin_directory (path to the cruncher).

To display the current values of these parameters :

```
getOption("default_matrix_item")
getOption("default_tsmatrix_series")
getOption("cruncher_bin_directory")
```

To modify them, use the function options(). For example :

```
options(default_matrix_item = c("likelihood.aic",
                                "likelihood.aicc",
                                "likelihood.bic",
                                "likelihood.bicc"))
options(default_tsmatrix_series = c("sa", "sa_f"))
```

```
"C:/Software/jwsacruncher-2.2.2/jdemetrycli-2.2.2/bin")
```

Using JDCruncheR (3/3)

Once these three options have been set, use the function
`cruncher_and_param()` :

```r
cruncher_and_param() #calculation using the default parameters

cruncher_and_param(workspace = "D:/my_folder/my_ws.xml",
# If you don't want to change the exported files' name
                   rename_multi_documents = FALSE,
                   policy = "lastoutliers")
```

To use the documentation, compute `help()` or `?function` :

```r
?cruncher_and_param
help(cruncher_and_param)
```

Sommaire

Quality report with JDCruncheR (1/4)

The JDCruncheR package also :

- computes a quality score
- creates a quality report from the diagnostics produced by JDemetra+

The three main functions of the package are :

- extract_QR to extract the quality report from the csv file (demetra_m.csv) that contains all JD+ diagnostics ;
- compute_score to compute a weighted score based on the diagnostics
- export_xlsx to export the quality report.

Quality report with JDCruncheR (2/4) : example

```
# choose the demetra_m.csv file generated by the cruncher
QR <- extract_QR()
QR

?compute_score # to see how the score is calculated (formula)
QR <- compute_score(QR,
                    n_contrib_score = 3)

QR

QR <- sort(QR, decreasing = TRUE, sort_variables = "score")
export_xlsx(QR,
            file_name = "U:/quality_report.xls")
```

When working with several workspaces (or SAPs), quality reports can be piled up with the function rbind() or by creating a mQR_matrix object with the function mQR_matrix()

Quality report with JDCruncher (3/4) : example

Missing values can be ignored and conditions can be set for indicators :

```r
# oos_mse weight reduced to 1 when the other
# indicators are "Bad" ou "Severe"
condition1 <- list(indicator = "oos_mse",
                   conditions = c("residuals_independency",
                                  "residuals_homoskedasticity",
                                  "residuals_normality"),
                   conditions_modalities = c("Bad","Severe"))
BQ <- compute_score(BQ, n_contrib_score = 5,
                    conditional_indicator = list(condition1),
                    na.rm = TRUE)
```

Quality report with JDCruncher (4/4) : example

```r
QR1 <- extract_QR()
QR2 <- extract_QR()
mQR <- mQR_matrix(QR1, QR2)
mQR
# naming each object
names(mQR) <- c("report_1", "report_2")
# Equivalent to:
mQR <- mQR_matrix(report_1 = QR1, report_2 = QR2)
mQR

# score calculation for all reports
mQR <- compute_score(mQR,
                     n_contrib_score = 3)
export_xlsx(mQR,
            export_dir = "U:/")
```

Example of score composition

Diagnostics		Weights (out of 100)
Pre-adjustment	ARIMA Model Residuals	30
	Residual Calendar Effects	20
Decomposition	Residual seasonality	45
	Decomposition Quality (stats M if X11)	5

Customize the score computation

Practical steps if you want to customize the score computation (see package documentation in R)

- select your indicators of interest
- adjust "good", "bad"…threshold in JD+ GUI if necessary
- by default good=0, uncertain=1, bad or severe=3
- change this grading system and/or the weights directly in the package functions
- rebuild your package

Warning : here only diagnostics are taken into account, revisions and numerical effects of potential parameter tuning have to be analysed with a complementary tool