Big Data Econometrics Nowcasting and Early Estimates

Big data handling tool – Source code description

George Kapetanios* Massimiliano Marcellino† Fotis Papailias‡
Katerina Petrova§

Abstract

Functions and usability of the R code are discussed in the paper. Most of the code is originally written by the authors. Whenever a R package is used, it is cited appropriately.

Keywords: Big Data, Nowcasting, Density Forecasting, Density Evaluation, R *Software.*

*kapetaniosgeorge@gmail.com
†massimiliano.marcellino@unibocconi.it
‡fotis.papailias@kcl.ac.uk; fotis.papailias@quantf.com
§katerina.petrova@st-andrews.ac.uk

Contents

1 Methods for feature extraction of Big Data sources to usable time-series for econometric modelling

1.1 High frequency returns – HF.R

Here we use the highfrequency R package, load edit and plot the necessary data as follows. We use various comments in order to explain the code and the inputs used in each function.

Realised volatility based on 5-min cleaned returns.

```
library("highfrequency")

# Load data
data(sample_returns_5min)

# Store the 5 min returns data
r <- sample_returns_5min

# Calculate the realised volatility based on 5 min returns
# Input. r: 5mins return data
RV <- rowSums(r^2)

# Creat a plot
# Input. RV: realised volatility based on 5mins return data as calculated above
plot(RV, type="l", main="EUR/USD 5-min RV", xlab="Time", ylab="RV")
```

1.2 Mobile phone data – Mobile.R

Here we load some .csv files which are also provided. The purpose of this code is to load the mobile phone activity data and aggregate it in terms of calls, SMS and internet activity. We use various comments in order to explain the code and the inputs used in each function.

Below we only demonstrate the part of the code which corresponds to Internet activity. The same procedure is replicated for call and SMS activity.

Mobile phone data aggregation and plots.

```
# Load the necessary data.
# Each file corresponds to a single day.
days <- c("sms-call-internet-mi-2013-11-01.csv",
          "sms-call-internet-mi-2013-11-02.csv",
          "sms-call-internet-mi-2013-11-03.csv",
          "sms-call-internet-mi-2013-11-04.csv",
          "sms-call-internet-mi-2013-11-05.csv",
          "sms-call-internet-mi-2013-11-06.csv",
```

```r
             "sms-call-internet-mi-2013-11-07.csv")

# Create some labels to be used in the plots later
days.l <- c("2013-11-01, Friday", "2013-11-02, Saturday", "2013-11-03, Sunday",
            "2013-11-04, Monday", "2013-11-04, Tuesday", "2013-11-05, Wednesday",
            "2013-11-07, Thursday")
days.l2 <- c("2013-11-01", "2013-11-02", "2013-11-03",
            "2013-11-04", "2013-11-04", "2013-11-05",
            "2013-11-07")

# Create a sparse matrix to store the results
result1 <- array(NA, dim=c(24,5,NROW(days)))

# We start with a loop.
# j runs for each different day. In the above we have 7 days (NROW(days)).
for(j in 1:NROW(days)) {
  # Load the data for day j.
  x <- read.csv(days[j], header=TRUE)

  # Identify unique users
  ud <- unique(x[,1])

  # Extract Total activity during the day across users
  # Create a sparse matrix to store the results
  totact <- matrix(NA, NROW(ud), 5)
  rownames(totact) <- as.character(ud)
  colnames(totact) <- c("smsin","smsout","callin","callout","internet")

  for(i in 1:NROW(ud)) {
    it <- ud[i]
    choose <- which(x[,1]==it)
    # Input. x which has the data and here we are looping across users.
    totact[i,1] <- sum(x[it, 4], na.rm=TRUE)
    totact[i,2] <- sum(x[it, 5], na.rm=TRUE)
    totact[i,3] <- sum(x[it, 6], na.rm=TRUE)
    totact[i,4] <- sum(x[it, 7], na.rm=TRUE)
    totact[i,5] <- sum(x[it, 8], na.rm=TRUE)
  }

  result1[,,j] <- totact
  cat("day ", j, " just done - still left ", NROW(days), "\n")
}

# We are now ready to create some plots
# First, generate the colours.
cols <- rainbow(NROW(days), alpha = 1)

# Calls Internet. Using the 5th column of array across days, "i", we can aggregate
# (sum) the internet activity.
i <- 1
plot(result1[,5,i], type="l", xlab="Hours", ylab="Call", col=cols[i], main="Total
    Internet Activity")

# Add the remaining days looping across "i"
for(i in 2:NROW(days)) {
  lines(result1[,5,i], col=cols[i])
}

legend("topleft", legend=days.l, col=cols, lty=rep(1, NROW(days)), cex=0.6)
```

1.3 Web prices – Prices.R

Here we load a .csv sample file with the appropriate data. We use various comments in order to explain the code and the inputs used in each function.

Web prices aggregation and plots.

```r
# Load the data
x <- read.csv("arg-sample2.csv", header=TRUE)
x <- as.matrix(x)
x <- x[,c(5, 1, 7, 6)] # Extract the necessary columns
colnames(x) <- c("date","id", "cat", "price")

d <- as.Date(x[,1])       # create the dates sequence
ud <- sort(unique(d))     # extract unique dates for time series aggregation
uid <- unique(x[,2])      # extract unique id's
uc <- unique(x[,3])       # extract unique categories

# Create sparse matrix to store the results
cats <- matrix(NA, NROW(ud)-1, NROW(uc))
colnames(cats) <- uc
rownames(cats) <- as.character(ud[2:NROW(ud)])

# Start the loop across categories
for(i in 1:NROW(uc)) {
  xcat <- which(x[,3]==uc[i])
  xcat <- x[xcat,]

  tmat <- matrix(NA, NROW(ud), NROW(uid))
  colnames(tmat) <- c(uid)

  # Create a doouble loop: (j) across id's and (jj) across dates
  for(j in 1:NROW(uid)) {
    xcat2 <- which(xcat[,2]==uid[j])
    xcat2 <- xcat[xcat2,]

    for(jj in 1:NROW(ud)) {
      cw <- which(xcat2[,1]==ud[jj])

      if(NROW(cw)==0){
        tmat[jj,j] <- NA
      } else {
        tmat[jj,j] <- xcat2[cw,4]
      }
    }
  }
  # ensure that the result is numeric
  tmat <- apply(tmat, 2, as.numeric)

  # Apply the methodology of Cavallo
  R <- tmat[2:NROW(tmat),]/tmat[1:(NROW(tmat)-1),]
  R <- apply(R, 1, prod, na.rm=TRUE)
  R <- R^(1/NCOL(tmat))
  cats[,i] <- R
}

# Cumulate across time
I <- apply(cats, 2, cumprod)
```

```
w <- as.matrix(rep(1/NCOL(I), NCOL(I)))

# use the approprite weights as in Cavallo
S <- I %*% w

# Produce the final CPI estimate plot.
plot(as.Date(rownames(S)), S, type="l", main="Online Prices CPI", xlab="Time", ylab=
    "Index")
```

1.4 Google Trends – Google.R

We use the gtrendsR library to download *Google Trends* in R. Currently, *Google Trends* are offered in monthly frequency.

Google Trends Download.

```
# User Input
kwd <- "gbp"              # Keyword(s)
reg <- "GB"               # Region
tsd <- "all"              # Time Frame: "all" (since 2004),
                          # "today+XXX-y" (last XXX years)
stp <- "web"              # "web", "news", "images", "froogle", "youtube"
cct <- 0                  # category

# Call the function and download data
gt <- gtrends(kwd, reg, tsd, stp, cct)

# Make a plot
plot(gt)
```

1.5 Twitter Data – Twitter.R

We use the twitteR library to download Twitter data in R.

Twitter data fetching.

```
# Inputs:
# user defined keyword, below we use #gbpusd feed
# n: the number of nodes to download
rdmTweets <- searchTwitter('#gbpusd', n=6500)
```

1.6 Reuters data scraping – Reuters.py

We provide below with a stepwise description of the Python code used for scraping Reuters data:

- L. 1-5: import packages: Beautifulsoup (HTML parsing), Scrapy (web crawling), Logging (recording errors), DateTime (parsing dates);
- L. 7: open a log file to record 404 errors (page not found);
- L. 10: create a name for the web scraper within the Scrapy CrawlSpider class;
- L. 13-18: spider settings. Note that we disabled the AUTOTHROTTLE setting and defined parameters manually to limit speed (one page download every 0.2 seconds) and contemporaneous requests (4).
- L. 20-25: define the input URLs for the web scraper. In a typical Scrapy crawler, one would only define a single starting page for the spider to move from, but we have already obtained the full list of pages to scrape in a text file. Therefore, we copy all the URLs in the file to a list and we pass each item to the parse function.
- L. 28-30: writes to a log file 404 errors;
- L. 32: passes the HTML code of the page to the parsing library lxml;
- L. 33-34: finds and isolates the permanent URL of the article;
- L. 35-36: finds and isolates the alternate URL of the article;
- L. 37-38: finds and isolates the article tags;
- L. 39-41: finds and isolates the publication date, headline and text of the article;
- L. 42-50: removes the remaining HTML formatting for clean reading;
- L. 51-52: parses the date and publication time to obtain a short-form date;
- L. 54-56: writes the collected data to a CSV file.

Reuters data scraping.

```
1 from bs4 import BeautifulSoup
2 import logging
3 import scrapy
4 from scrapy.spiders import CrawlSpider
5 from datetime import datetime
6
7 logging.basicConfig(filename='log_50.log', level=logging.ERROR)
```

```
class SpiderReuters (CrawlSpider):
    name = 'crawler_final_v21_tags_2'
    handle_httpstatus_list = [404]

    custom_settings = {
        'AUTOTHROTTLE_ENABLED': False,
        'CONCURRENT_REQUESTS_PER_DOMAIN': '4',
        'DOWNLOAD_DELAY': '0.2',
        'USER_AGENT': "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0"
    }

    def start_requests(self):
        f = open(r"/home/PATH-to-file/list_of_URLs.txt", 'r')
        start_urls = [url.strip() for url in f.readlines()]
        f.close()
        for urls in start_urls:
            yield scrapy.Request(url=urls, callback=self.parse)

    def parse(self, response):
        if response.status == 404:
            with open('log_404.txt', 'a') as l:
                l.write(str(response) + '\n')
        else:
            soup = BeautifulSoup(response.body, 'lxml')
            extractor_a = soup.find('link', rel='canonical')
            a_return = extractor_a['href']
            extractor_alt = soup.find('link', rel='alternate')
            alt_return = extractor_alt['href']
            extractor_tags = soup.find('meta', property="og:article:tag")
            tags_return = extractor_tags['content']
            extractor_1 = soup.find_all('span', class_='timestamp')
            extractor_2 = soup.find_all('h1', class_='article-headline')
            extractor = soup.find_all('span', id='article-text')
            composer = [extractor_1, a_return, alt_return, extractor_2, extractor, tags_return]
            composer_2 = []
            for element in composer:
                element_2 = ' '.join(str(element).strip('[]').splitlines())
                composer_2.append(element_2)
            composer_text = "|".join([str(item).replace("|", "") for item in composer_2])
            clean_page = BeautifulSoup(composer_text, 'lxml')
            all_text = ' '.join(clean_page.findAll(text=True))
            all_text_uni = all_text.decode('unicode_escape').encode('utf-8', 'strict')
            date = all_text_uni[4:17].replace(',', '').strip(' ')
            date_object = datetime.strptime(date, '%b %d %Y')
            page = str(response).replace("/", "").replace(".", "").replace(":", "").replace("<", "").replace(">", "")
            with open(r'/home/PATH/output_%s.csv' % page, 'a') as a:
                a.write(date_object.strftime('%b %d %Y') + '|')
                a.write(' '.join(all_text_uni.splitlines()).replace("  ", " "))
```

We also provide below with a stepwise description of the code used for the index construction:

- L. 1-5: imports packages: time (algorithm timer), os (interfacing with operating system), re (regular expressions), csv (comma separatted values file reading and writing), pandas (time series statistical package);
- L. 7: starts timer;
- L. 9: initializes article counter;
- L. 11: dictionary of search terms (regular expressions);
- L. 13-15: creates a CSV file for articles with column headers 'date', 'url', 'headline' and 'uncertainty';
- L. 18-19: iterates through directories and obtains a list of files in each folder;
- L. 21-23: opens and counts each article;
- L. 24-27: excludes articles containing the word "SPORT" among the tags;
- L. 28-34: if the article body contains one of the terms in line 11, the date, url, tags and a "1" binary indicator are written to a csv file;
- L. 35-40: otherwise, date, url, tags and a "0" binary indicator are written to the same csv file;
- L. 41-44: handles csvError exceptions (i.e. ignores files in folder that are not csv format);
- 48-55: the tagged article list just obtained is loaded in a Pandas dataframe, dates are converted to a machine-readable format and daily frequencies are obtained and written to a csv file; the actual index is computed at line 62.

Index construction.

```
1 import os, re, time
2 import csv
3 import pandas as pd
4
5 start_time = time.time()
6
7 article_count = 0
8
9 combined_semantic = "\\risk\\b|\\brisks\\b|\\brisks\\b"
```

```
with open('art_list_RISK.csv', 'w', encoding='utf-8') as el:
    spamwriter = csv.DictWriter(el, delimiter=',', fieldnames=['date', 'url', 'alt_url', 'tags', 'filename', 'uncertainty'], lineterminator='\n')
    spamwriter.writeheader()

# search iteration
for root, dirs, filename in os.walk(r'/home/mattia/Desktop/All_articles'):
    for sub_file in filename:
        try:
            with open(os.path.join(root, sub_file), 'r', encoding="utf8", errors='ignore') as f:
                spamreader = csv.DictReader(f, delimiter='|', fieldnames=['date', 'longdate', 'url', 'alt_url', 'header', 'article', 'tags'])
                article_count += 1
                for row in spamreader:
                    if 'SPORT' in [x.strip() for x in row['tags'].split(',')]:
                        print('sport', row['url'])
                        break
                match = re.search(combined_semantic, row['article'], re.IGNORECASE)
                if match:
                    print('progress', '%.3f' % ((article_count / 2803677) * 100), '%')
                    line_with_dummy = dict(url=row['url'], alt_url=row['alt_url'], date=row['date'], tags=row['tags'], filename=sub_file, uncertainty='1')
                    with open('art_list_RISK.csv', 'a', encoding='UTF-8') as out:
                        spamwriter_2 = csv.DictWriter(out, delimiter=',', fieldnames=['date', 'url', 'alt_url', 'tags', 'filename', 'uncertainty'], lineterminator='\n')
                        spamwriter_2.writerow(line_with_dummy)
                else:
                    row.update({'uncertainty': '0'})
                    line_with_zero = dict(url=row['url'], alt_url=row['alt_url'], date=row['date'], tags=row['tags'], filename=sub_file, uncertainty='0')
                    with open('art_list_RISK.csv', 'a', encoding='UTF-8') as out:
                        spamwriter_3 = csv.DictWriter(out, delimiter=',', fieldnames=['date', 'url', 'alt_url', 'tags', 'filename', 'uncertainty'], lineterminator='\n')
                        spamwriter_3.writerow(line_with_zero)
        except csv.Error:
            with open('csverror.txt', 'a', encoding='utf-8') as csverr:
                csverr.write(root + sub_file)
                pass

print('end list article list')

df = pd.read_csv('art_list_RISK.csv')
print('dataset in memory')
df["date"] = pd.to_datetime(df["date"])
frequency_table = pd.crosstab(index=df["date"], columns=df['uncertainty'], margins=True)
df2 = pd.DataFrame(frequency_table)
df2['ratio'] = df2[1]/df2['All']
df2.columns = ['no_uncertainty', 'uncertainty', 'all', 'ratio']
df2.to_csv('Marcellino_daily_RISK_index.csv', encoding='utf-8')

elapsed_time = time.time() - start_time

print(elapsed_time/60, 'minutes')
```

2 Filtering techniques for high frequency data

2.1 Outliers detection – Outliers.R

In this task we were mostly concerned with outliers detection, seasonalities and data cleaning. Below, we present the functions we use in the Outliers.R script.

Using scores from the outliers package.

```
# Inputs
# x is a vector of data, unstructured or aggregated depending on the theme.
# type: "z" calculates normal scores (differences between each value and the mean
    divided
# by sd)
# prob: the corresponding p-values are returned level choice depends on the user
od <- scores(x, type="z", prob=0.99)

# identify the position of outliers
which(od==TRUE)
```

Now, we use the built-in stl to identify and extract the trend and seasonal component. This will lead to the cleaned series.

Seasonal decomposition of time series by Loess.

```
# Input: aggx is a numeric vector of aggregated time series in weekly frequency
# First, we transform the numeric vector in a time series (ts) object correctly
# specifying the frequency.
tsaggx <- ts(aggx, frequency=7)

# Then, we use the ts object, tsaggx, as the input in the LOESS function.
# s.window: can be a string "periodic" or "per" which reads the frequency from the
# ts transformation, otherwise it can be a user choice.
ss <- stl(tsaggx, s.window="per")

# Plot the output
plot(ss, main="Daily Aggregation, Weekly Pattern")

# Extract the seasonal component (xs) and the trend component (xt)
xs <- ss$time.series[,1]
xt <- ss$time.series[,2]

# Calculate the cleaned series (xc)
xc <- tsaggx-xs-xt

# And create a plot
plot(xc, type="l", xlab="", ylab="", main="Detrended and Deseasonalised")
```

3 Modelling techniques for Big Data

In this section, we present the codes for Ridge, Lasso, Elastic Net penalised regressions, regression trees and random forests. For the penalised regression models, we use several functions from the glmnet package.

3.1 Penalised Regression Models – Ridge.R, ElasticNet.R

For the estimation of Ridge and Elastic Net regression, we use the glmnet package. The main function is glmnet(x, y, family=".", alpha=.) and it implements penalised regression of an $N \times p$ matrix of explanatory variables x on a N-dimensional vector y.

The package also performs k-fold cross validation, with the function cv.glmnet. Finally, to generate predictions the following command can be used: predict(fit.info, newdata=x.test), where fit.info is the output from the glmnet (e.g. coefficients/ confidence intervals) etc.

Ridge penalised regression.

```
# Example Code Ridge
rm(list=ls())
install.packages("glmnet") # download and install package

library (glmnet)

#generate some artificial data
set.seed(1)
n <- 200  # Number of observations
p <- 300  # Number of predictors included in model

beta<- c(1/(1:p)^2)
# approximately sparse model, slope coefficients small but not zero
x <- matrix(rnorm(n*p), nrow=n, ncol=p)
y <- x%*%beta + rnorm(n)

#generate the dependant variable y
fit.ridge <- glmnet(x, y, family="gaussian", alpha=0)

#select ridge penalty
nforecast=5
xnew <- matrix(rnorm(nforecast*p), nrow= nforecast, ncol=p)

predict(fit.ridge, newdata=xnew) # predictions based on estimated coefficients
```

Lasso regression.

```
# Example Code Lasso
rm(list=ls())
install.packages("glmnet") # download and install package
```

```

library (glmnet)

#generate some artificial data
set.seed(1)
n <- 200  # Number of observations
p <- 300  # Number of predictors included in model

beta<- matrix(c(rep(1,p/2),rep(0,p/2)))
# sparse model, some slope coefficients are zero
x <- matrix(rnorm(n*p), nrow=n, ncol=p)

y <- x%*%beta + rnorm(n)
# generate the dependant variable y

fit.lasso <- glmnet(x, y, family="gaussian", alpha=1) #select Lasso penalty

nforecast=5
xnew <- matrix(rnorm(nforecast*p), nrow= nforecast, ncol=p)

predict(fit.lasso, newdata=xnew) # predictions based on estimated coefficients
```

Elastic Net regression.

```
# Example Code Elastic Net
rm(list=ls())
install.packages("glmnet") # download and install package

library (glmnet)

#generate some artificial data
set.seed(1)
n <- 200  # Number of observations
p <- 300  # Number of predictors included in model

beta1<- c(1/(1:p/2)^2)
beta<- matrix(c(beta1,rep(0,p/2)))
#combination of sparse and approximately sparse coefficient vector
x <- matrix(rnorm(n*p), nrow=n, ncol=p)

# generate the dependant variable y
y <- x%*%betaâĂŽ + rnorm(n)

fit.elnet <- glmnet(x, y, family="gaussian", alpha=0.4) # 40% weight to Lasso penalty

nforecast=5
xnew <- matrix(rnorm(nforecast*p), nrow= nforecast, ncol=p)

predict(fit.elnet, newdata=xnew) # generate predictions based on estimated coefficients
```

3.2 Spike and Slab regression – SpikeSlab.R

For the Spike and Slab regression model, we use the BoomSpikeSlab package. The main function is lm.spike(y x, iter) where x is an $N \times p$ matrix of explanatory variables, y is an N-dimensional vector and iter is the number of Metropolis draws from the posterior distribution of the parameters.

Slab and Spike regression.

```
# Example Code Slab and Spike
rm(list=ls())
install.packages("BoomSpikeSlab") # download and install package

library (BoomSpikeSlab)

#generate some artificial data
set.seed(1)
n = 200 #sample size
p = 300 # number of variables
nonzerob = 3 # nubmer of variables with non-zero coefficients
niter <- 1000 # nubmer of MCMC draws
sigma <- .8

x <- cbind(1, matrix(rnorm(n * (p-1)), nrow=n))
beta <- c(rep(2,ngood),rep(0, p-nonzerob ))
y <- rnorm(n, x %*% beta, sigma)
x <- x[,-1]

# estimate spike and Slab regression
model <- lm.spike(y ~ x, niter=niter)

# plots of coefficients
plot.ts(model$beta)
hist(model$sigma) ## should be near 8
plot(model)
summary(model)

# plot residuals
plot(model, "residuals")

Xnew = cbind( matrix(rnorm(n * (p-1)), nrow=n))

# if out-of-sample forecasts are required
yhat.slab.new = predict.lm.spike(model, newdata=Xnew) #out-of-sample prediction
```

3.3 Regression Trees and Forests – Tree.R, Forest.R

For the regression trees and forests, we make use of four R packages ISLS, randomForest, rpart and rpart.plot. To implement a standard regression tree with default settings: fit.trees<- rpart(y x) where x is an $N \times p$ matrix of explanatory variables and y is N-

dimensional vector.

To prune a tree, the following instructions can be used:

Tree pruning.

```
bestcp          <- trees$cptable[which.min(trees$cptable[,"xerror"]),"CP"]
fit.prunedtree          <- prune(fit.trees,cp=bestcp)
prp(fit.prunedtree)
```

The main function to optimally estimate a forest is RFfit<- tuneRF(x, y). Finally, the packages can be used to generate predictions. For regression trees, this can be achieved with the command:

```
yhat.pt<- predict(fit.prunedtree, newdata=as.data.frame(..))
```

while, for forests, this can be achieved with the command:

```
yhat.rf2<- predict(RFfit, newdata=(..))
```

Below, we illustrate with an example.

Standard Regression Tree

```
rm(list=ls())

# Regression Tree

library (ISLR)
library(randomForest)
library(rpart)
library(rpart.plot)

#generate some artificial data
set.seed(1)
n <- 200  # Number of observations
p <- 300  # Number of predictors included in model

beta<- c(10/(1:p)^2)
x <- matrix(rnorm(n*p), nrow=n, ncol=p)
y <- x%*%beta + rnorm(n)*4

# estimate Standard Regression tree on the atrificial data
fit.trees<- rpart(y~x)

prp(fit.trees)

# estimate pruned Regression tree on the atrificial data
bestcp          <- trees$cptable[which.min(trees$cptable[,"xerror"]),"CP"]
fit.prunedtree <- prune(fit.trees,cp=bestcp)

prp(fit.prunedtree)
```

Random Forest

```
rm(list=ls())

# Random Forest

library (ISLR)
library(randomForest)
library(rpart)
library(rpart.plot)

#generate some artificial data
set.seed(1)
n <- 200  # Number of observations
p <- 300  # Number of predictors included in model

beta<- c(10/(1:p)^2)
x <- matrix(rnorm(n*p), nrow=n, ncol=p)
y <- x%*%beta + rnorm(n)*4

#Estimate a Random forest on the atrificial data
RFfit<- tuneRF(x, y, mtryStart=floor(sqrt(ncol(x))),stepFactor=1.5, improve=0.05,
    nodesize=5, ntree=2000, doBest=TRUE)

#Find the best fir for the Random forest on the atrificial data
min <- RFfit$mtry
fit.rf2 <-randomForest(x, y, nodesize=5, mtry=min, ntree=2000)
```

3.4 Bayesian VAR models – BVAR.R

We make use the of the MSBVAR package for Bayesian vectorautoregressive models. The main function is szbvar, which takes the following inputs:

- Y is $T \times M$ matrix of time series;
- p is lag length;
- z is $T \times N$ matrix of exogenous vars, can be NULL;
- lambda0 is the overall tightness between 0 and 1;
- lambda1 is the standard deviation or tightness of the prior around the AR(1) parameters;
- lambda3 is the lag decay (> 0, with 1=harmonic);
- lambda4 is the standard deviation or tightness around the intercept > 0;

- lambda5 is the standard deviation or tightness around the exogneous variable coefficients;
- mu5 is the sum of coefficients prior weight (larger values imply difference stationarity);
- mu6 is dummy initial observations or drift prior (larger values allow for common trends);
- nu is the prior degrees of freedom, $m + 1$;
- qm is the frequency of the data for lag decay equivalence;
- prior can be of three values: 0 = Normal-Wishart prior, 1 = Normal-flat prior, 2 = flat-flat prior;
- posterior.fit is a logical, FALSE implies no estimation of log-posterior fit measures.

The package can be used to generate out-of-sample forecasts, using the function forecast, for example:

```
forecasts <- forecast(fit.bvar, nsteps))
```

with nsteps the numbers of horizons for the out-of-sample forecasts.

Bayesian VAR

```
rm(list=ls())

# Bayesian VAR
install.packages("MSBVAR") # download and install package

library (MSBVAR)

#generate some artificial data
set.seed(1)
n = 200 #sample size
p = 10 # number of variables
X0 = rep(0,p)
beta = rep(0.5,p)
B=diag(beta)
y=matrix(0,nrow=p,ncol=n)
for (i in 1:n) {
    e = rnorm(p)
    y[,i]=B%*%X0+e
    X0=y[,i]
}

# Reference prior model -- Normal-IW prior pdf
```

```
Bvar.Model <- szbvar(y, p=6, z=NULL, lambda0=0.6, lambda1=0.1, lambda3=2, lambda4=0.5, lambda5=0, mu5=0, mu6=0, nu=ncol(KEDS)+1, qm=4, prior=0, posterior.fit=F)

# Forecast -- this gives back the sample PLUS the forecasts.
forecasts <- forecast(Bvar.Model, nsteps=10,burnin=3000, gibbs=5000, exog=NULL)

# Conditional forecasts
conditional.forcs.ref <- hc.forecast(Bvar.Model, yhat, nsteps,
burnin=3000, gibbs=5000, exog=NULL)
```

4 Modelling strategies for nowcasting/early estimates purposes

The codes presented in this section address the following issues:

(i) data manipulation,

(ii) nowcasting, and

(iii) post-processing of results.

The first part refers to all codes we used to manipulate the downloaded data. This part is "data-specific" and it applies to data downloaded from the same sources as in these tasks. If a researchers uses Bloomberg, Reuters, Macrobond or other data collection software, the original data is different and cannot be used as input in these functions. Therefore, we do not discuss them here. In the following section, we take as granted that the user has already downloaded and cleaned the data.

4.1 Weekly Google Trends – Weekly-Google.R

As mentioned in a previous section, *Google Trends* are downloaded in monthly frequency. We have developed a function which loads Google Trends over smaller time frames at weekly frequency, and then scales the data in order to obtain the weekly *Google Trends* for the overall period. The function uses the main function from gtrendsR package.

Weekly Google Trends.

```
# Set dates for all data
dfrom <- "2004-01-01"   # starting date
dto <- "2017-09-01"     # ending date
```

```

#  (the news doesn't really have a lot of data, so stick to the web)
stp <- "web"              # web; news;
cct <- 0                  # category

# General Indexes - web
reg <- ""                 # Region, blank for all regions or use "GB" for the UK, etc.
kwd <- "uncertainty"

# Download the weekly trends and store them in c1 variable
# Input: kwd (keyword), reg (region)
#        cct (category), stp (domain)
#        dfrom (start), dto (end)
c1 <- weekly.GOOGLE(kwd, reg, cct, stp, dfrom, dto)
```

4.2 Transformation

We transform the final nowcast/forecast estimates to levels according to the nature of the series. In order to avoid repetition, the code is described here.

From change to levels.

```
# YTRANSF: if 3, we translate from growth to levels
#          if 2, we translate from 1-st diff to levels
#   zlast: is the last observed value for the dependent variable
#   zboot: are the bootstrap estimate for densities
#   zave: (or different name) is the final estimate in levels.
if(YTRANSF==3){
  zlast <- YSAV[NROW(YSAV)]
  zboot <- zlast*(1+zboot)
  zave <- zlast*(1+zave)
}
if(YTRANSF==2){
  zlast <- YSAV[NROW(YSAV)]
  zboot <- zlast +zboot
  zave <- zlast + zave
}
```

4.3 Averaging – averaging.R

We calculate the average value of the last observations and also use Bootstrap to calculate the corresponding density estimates.

Averaging and Bootstrap for density.

```
# Input:
# z is the dependent variable, vector of data
# zp: window length, e.g. zp=4, then we have the 4-period MA.
zave <- mean(z[(NROW(z)-zp+1):NROW(z),])

```

```
# Bootstrap for density estimation
# b: bootstrap window length
# B: number of bootstraps
b <- round(NROW(z)^(1/3))
boots <- matrix(NA, NROW(z), B)
for(j in 1:B){
  boots[,j] <- MBB(z, b) # MBB: Moving Block Bootstrap
}

# Calculate the mean value for the same zp
zboot <- colMeans(boots[(NROW(boots)-zp+1):NROW(boots),])
```

4.4 Naive Estimate – naive.R

We calculate the naive forecast and also use bootstrap to calculate the corresponding density estimates.

Naive forecasting.

```
# Input:
# z is the dependent variable, vector of data
# here we extract the last observed value
zf <- z[NROW(z)]

# Bootstrap for density estimation
# b: bootstrap window length
# B: number of bootstraps
b <- round(NROW(z)^(1/3))
boots <- matrix(NA, NROW(z), B)
for(j in 1:B){
  boots[,j] <- MBB(z, b)
}

# Extract the corresponding naives
zboot <- boots[NROW(boots),]
```

4.5 ARIMA – ar.R

We calculate various ARIMA model-based forecasts using the forecast package.

ARIMA forecasting.

```
# If an AR order is set to zero, we use AIC
if(arp==0){ arp <- NROW(ar.ols(z, aic=TRUE)$ar) }

# Estimate ARIMA using
# z: the vector of the observed dependent variable
# order: ARIMA order
# method: conditional sum of squares
fit <- Arima(z,order=c(arp,0,0), method=c("CSS"))
fout <- NULL
```

```

# Calculate percentiles
try(fout <- forecast(fit,h=1, bootstrap=TRUE, npaths=B, level=seq(51,99,1)), silent=
    TRUE)

# Put all percentiles together in the correct order
zout <- c(fout$mean, rev(as.numeric(fout$lower))[1], rev(fout$lower), fout$mean,
    fout$upper, as.numeric(fout$upper)[49])
```

4.6 Dynamic Factor Analysis – dfa.R

We calculate forecasts/nowcasts based on Dynamic Factor Analysis. Codes are adapted versions of original Giannone and Reichlin's Matlab programs.

Dynamic Factor Analysis forecasting.

```
# Extract factors using:
# XX: matrix of observed regressors
# q: dynamic rank
# r: static rank (r>=q)
# p: ar order of the state vector
fac.out <- FactorExtraction(XX,q=fq,r=fr,p=fp)
Fac <- fac.out$Fac
rownames(Fac) <- rownames(XX)

# Then use the linreg with the factors
z <- YY; f <- Fac; vlag <- 0; source(".../linreg.R")
```

4.7 Factor Linear Regression – Flinreg.R

We calculate forecasts/nowcasts based on linear regression using factors which come from standardised matrices.

Factor Linear regression.

```
# jj: step-ahead
# z: target, vector
# f: factor which comes from standardised data
# ysd: the standard deviation of target
# ymu: the mean of target
jj <- 1
zreg <- as.matrix(z[(jj+1):NROW(z),])
freg <- as.matrix(f[1:(NROW(f)-jj),])
out <- lm(zreg~freg)
b <- out$coefficients;

# Now make sure to use ysd, ymu as we used factors from standardised input
outf <- ((f[NROW(f),]%*%b[2:NROW(b)]) + b[1])*ysd+ymu
```

4.8 Partial Least Squares – pls.R

We calculate forecasts/nowcasts based on partial least squares methodology. We are using the plsr package.

Factor Linear regression.

```
# Lag the matrix of Regressors if necessary
# vlag: lag order
# XX : matrix, panel of regressors
# YY : vector, observed target
vlag <- 1
if(vlag>0){
  XX <- cbind(lagf(YY, vlag)[,2:(vlag+1)], XX)
  XX <- as.matrix(XX[(vlag+1):NROW(XX),])
  YY <- as.matrix(YY[(vlag+1):NROW(YY),])
}

# Standardise
xxin <- xstd(XX)
ymu <- mean(YY)
ysd <- sd(YY)
yyin <- (YY-ymu)/ysd

# Extract factors
pp <- plsr(yyin~xxin, ncomp=qncomp, scale=FALSE)
f <- as.matrix(pp$scores)
z <- as.matrix(yyin)

# Use Factor linear regression
source("Flinreg.R")
```

4.9 Sparse Principal Components – spc.R

We calculate forecasts/nowcasts based on sparse principal components methodology. We are using the nsprcomp package.

Sparse Principal Components.

```
# jj: step-ahead
# z: target, vector
# f: factor which comes from standardised data
# ysd: the standard deviation of target
# ymu: the mean of target
vlag <- 1
if(vlag>0){
  # XX <- lagmv(XX, vlag)   # no because of small T dimension
  XX <- cbind(lagf(YY, vlag)[,2:(vlag+1)], XX)

  XX <- as.matrix(XX[(vlag+1):NROW(XX),])
  YY <- as.matrix(YY[(vlag+1):NROW(YY),])
}

```

```
# Standardise
xxin <- xstd(XX)
ymu <- mean(YY)
ysd <- sd(YY)
yyin <- (YY-ymu)/ysd

# Extract factors
pc.out <- nsprcomp(x=xxin, retx=TRUE, ncomp=qncomp, nneg = FALSE, center=FALSE,
    scale.=FALSE)
f <- as.matrix(pc.out$x)
z <- as.matrix(yyin)

# Use Factor linear regression
source(".../Flinreg.R")
```

4.10 Sparse Regression – sparse.R

We calculate forecasts/nowcasts based on sparse regression methodology. We are using the glmnet package.

Sparse regression.

```
# jj: step-ahead
# z: target, vector
# f: factor which comes from standardised data
# ysd: the standard deviation of target
# ymu: the mean of target
vlag <- 1
if(vlag>0){
  XX <- cbind(lagf(YY, vlag)[,2:(vlag+1)], XX)
  XX <- as.matrix(XX[(vlag+1):NROW(XX),])
  YY <- as.matrix(YY[(vlag+1):NROW(YY),])
}
z <- YY
f <- XX

# jj: step ahead, then correctly lead/lag the variables
jj <- 1
zreg <- as.matrix(z[(jj+1):NROW(z),])
freg <- as.matrix(f[1:(NROW(f)-jj),])

# Calculate beta which comes from sparse
# freg: regressors,  matrix
# zreg: target, vector
# type.measure: "mse" for the calculation of lambda
# alpha: 1 for Lasso, 0.5 for LAR
fit.lasso.cv <- cv.glmnet(freg, zreg, type.measure="mse", alpha=salpha, family="
    gaussian", standardize=TRUE)
s <- fit.lasso.cv$lambda.min
b <- as.numeric(coef(fit.lasso.cv, s))
outf <- ((f[NROW(f),]%*%b[2:NROW(b)]) + b[1])

# Calculate percentiles
sigmah <- sd(zreg-freg%*%b[2:NROW(b)]-b[1])
spseq <- qnorm(seq(0.51, 0.99, 0.01))*sigmah
```

```
zout <- c(outf, outf-rev(spseq)[1], outf-rev(spseq), outf, outf+spseq, outf+spseq[
    NROW(spseq)])
```

4.11 Spike and Slab – spike.R

We calculate forecasts/nowcasts based on sparse regression methodology. We are using the BoomSpikeSlab package.

Spike and Slab Regression.

```
# jj: step-ahead
# z: target, vector
# f: factor which comes from standardised data
# ysd: the standard deviation of target
# ymu: the mean of target
lag <- 1
if(vlag>0){
  # XX <- lagmv(XX, vlag)   # no because of small T dimension
  XX <- cbind(lagf(YY, vlag)[,2:(vlag+1)], XX)
  XX <- as.matrix(XX[(vlag+1):NROW(XX),])
  YY <- as.matrix(YY[(vlag+1):NROW(YY),])
}
# Standardise
xxin <- xstd(XX)
ymu <- mean(YY)
ysd <- sd(YY)
yyin <- (YY-ymu)/ysd

z <- yyin
f <-xxin

jj <- 1
zreg <- as.matrix(z[(jj+1):NROW(z),])
freg <- as.matrix(f[1:(NROW(f)-jj),])

# Spike and Slab
# niter: The number of MCMC iterations to run
# ping: output printing parameter
out <- lm.spike(zreg ~ freg, niter=niters, ping=B)

# keep the last B rounds
b <- out$beta
b <- b[(NROW(b)-B+1):NROW(b),]
bf <- colMeans(b)
outf <- ((f[NROW(f),]%*%bf[2:NROW(bf)]) + bf[1])*ysd+ymu

#Calculate percentiles
sigmah <- sd((zreg-freg%*%bf[2:NROW(bf)]-bf[1])*ysd+ymu)
spseq <- qnorm(seq(0.51, 0.99, 0.01))*sigmah
zout <- c(outf, outf-rev(spseq)[1], outf-rev(spseq), outf, outf+spseq, outf+spseq[
    NROW(spseq)])
```

4.12 Evaluation Statistics

We calculate various evaluation statistics. Check the main files for more details.

Mean Absolute Error.

```
# err: matrix with errors of various methods
mae <- as.matrix(colMeans(abs(err)))

# relative MAE
benchmark <- "AR(1)"
maeR <- mae/mae[benchmark,1]
```

Root Mean Squared Forecast Error.

```
# err: matrix with errors of various methods
rmsfe <- as.matrix(sqrt(colMeans(err^2)))

# relative RMSFE
benchmark <- "AR(1)"
rmsfeR <- rmsfe/rmsfe[benchmark,1]
```

Two Sided Diebold-Mariano.

```
# Using the package ‘‘forecast"
# err: matrix with errors of various methods
# i:  method i stored in column i of err
# h: horizon
# power: power
benchmark <- "AR(1)"
dmp <- dm.test(err[,i], err[,benchmark], alternative=c("two.sided"), h=1, power=2)

# Extract the p-value
dmp <- as.numeric(dmp$p.value)
```

Sign Success Ratio.

```
# err: matrix with errors of various methods
# forc: signs of forecast direction compared to the last period for a given method
# sgnt: signs of direction of the target
ssr <- sum(sgnt==sign(forc))

# relative SSR
benchmark <- "AR(1)"
ssrR <- ssr/ssr[benchmark,1]
```

Berkowitz LR Test.

```
# xcloud: all percentile forecasts
xcloud <- allf
z <- pnorm(xcloud[,1], mean=apply(xcloud[,2:NCOL(xcloud)], 1, mean), sd=apply(xcloud
    [,2:NCOL(xcloud)], 1, sd))
Z <- qnorm(z)

```

```
# Extract Berkowitz P-value
berk <- BerkowitzTest(Z, lags=1, significance = 0.05)$LRp
```