The U.S. Federal Reserve quarterly model in R with bimets

Andrea Luciani
Bank of Italy*

Abstract

The US Federal Reserve's econometric model for the US economy (i.e., FRB/US) is publicly available at federalreserve.gov. The website states, *"FRB/US is a large-scale estimated general equilibrium model of the US economy that was developed at the Federal Reserve Board, where it has been in use since 1996 for forecasting, analysis of policy options, and research projects."*

FRB/US is a quarterly model with hundreds of equations and variables. The model definition and time series data are available for download on the Federal Reserve website, as is the source code, which allows users to perform several econometric exercises.

However, the Federal Reserve publicly distributes source codes only for EViews® and python.

bimets is a software framework developed by using R language and designed for time series analysis and econometric modeling. In these pages, we will show how to use **bimets** capabilities to load the FRB/US model and perform in R the same econometric exercises provided by the Federal Reserve. For each exercise, we will compare the numerical results in R and python.

Keywords: R, bimets, system of simultaneous equations, Federal Reserve quarterly model, FRB/US, model simulation, forecasting, endogenous targeting, stochastic simulation, rational expectations.

1. The FRB/US model

The Federal Reserve website states, *"FRB/US is a large-scale estimated general equilibrium model of the US economy that was developed at the Federal Reserve Board, where it has been in use since 1996 for forecasting, analysis of policy options, and research projects. ... Compared with DSGE models, however, FRB/US applies optimization theory more flexibly, which permits its equations to better capture patterns in historical data and facilitates modeling the economy in greater detail. ... A distinctive feature of FRB/US is its ability to switch between*

*Disclaimer: *The views and opinions expressed in these pages are those of the author and do not necessarily reflect the official policy or position of the Bank of Italy. Examples of analysis performed within these pages are only examples. They should not be utilized in real-world analytic products as they are based only on very limited and dated open source information. Assumptions made within the analysis are not reflective of the position of the Bank of Italy.*

alternative assumptions about how economic agents form expectations. Under the VAR-based option, expectations are derived from the average historical dynamics of the economy as manifested in the predictions of estimated VAR models. Under model-consistent (MC), agents are assumed to form accurate expectations of future outcomes as generated by simulations of FRB/US itself."

FRB/US is a quarterly model, and counts 284 equations and 365 variables (Feb. 2024 version). The XML model definition is available for download on the Federal Reserve website, and contains, for each endogenous variable, the following information: the variable name, the variable definition with a short description, the economic sector the variable belongs to, the related equation in both EViews® and python format, coefficients and exogenous variables involved in the equation.

64 endogenous variables are marked as stochastic and, during the stochastic simulation exercise (see section 4.5), will be transformed by applying sequences of shocks as drawn randomly from their historical residuals.

14 endogenous variables belong to the MCE group (i.e., Model-Consistent Expectations) and have an alternative equation that contains forward-looking references (see section 4.2).

Finally, at the end of the XML model definition, users can find additional information on economic sectors and exogenous variables involved in the model definition.

2. The pyfrbus python package

pyfrbus is a python-based simulation platform for the Federal Reserve Board's FRB/US model, which depends on the `SuiteSparse` code, a widely used set of sparse-matrix-related optimized algorithms.

As stated in the reference manual, to load the model in python, create a new `Frbus` object using the constructor, pointing it at the provided model XML:

```
from pyfrbus.frbus import Frbus
frbus = Frbus("model.xml")
```

The equations of the model are loaded from the XML file's `python_equation` tags, and each equation is modified by adding a tracking residual named with the suffix `_trac`.

By default, the backward-looking VAR expectations version of FRB/US is loaded. The constructor takes an optional argument `mce` which allows users to select which version of the forward-looking rational expectations model to load. Valid MCE types are `all`, `mcap`, `wp`, and `mcap+wp`.

The CSV FRB/US dataset `LONGBASE.TXT` can be loaded from the data-only-package with the `load_data` function:

```
from pyfrbus.load_data import load_data
data = load_data("LONGBASE.TXT")
```

The model requires that series exist in the input `pandas` DataFrame for all endogenous and exogenous variables. From here, we can initialize the tracking residuals for the model using the `init_trac` function:

```
start = "2019Q4"
end = "2030Q4"
baseline_with_adds = frbus.init_trac(start, end, data)
```

`init_trac` returns a new DataFrame where the `_trac` variables have shocks filled in such that the model will solve to the specified baseline values.

Model simulations are run by calling the `solve` function (see section 4.1):

```
sim = frbus.solve(start, end, baseline_with_adds)
```

This gives a new DataFrame where the series for endogenous variables take on values consistent with the exogenous series, lags, and model equations over the specified period.

The solver will automatically detect whether the model is forward-looking and will switch to an algorithm suited for solving such models (see section 4.2):

```
# MCE alert!
frbus = Frbus("model.xml", mce="mcap+wp")
...
# Returns a solution with forward-looking expectations
sim = frbus.solve(start, end, baseline_with_adds)
```

The `mcontrol` algorithm is a trajectory-matching control procedure (i.e., endogenous targeting) which adjusts the value of instrument variables such that target variables are forced to specified trajectories, as mediated by the model's dynamics.

`mcontrol` takes three lists of model variables as input: `targ` is the list of endogenous model variables to be forced; `traj` is the list of trajectories to force `targ` variables to; and `inst` is the list of exogenous model variables that will be moved freely to control the `targ` variables (see section 4.4).

The `stochsim` procedure performs a stochastic simulation by applying sequences of shocks to the model, drawn randomly from historical residuals. The procedure begins by randomly drawing sequences of quarters from the residual period. In each quarter of a single replication, all stochastic variables (specified with a `stochastic_type` tag in the model) have a shock applied from a particular quarter in the residual period (see section 4.5).

3. Moving to R

bimets is a software framework developed by using R language and designed for time series analysis and econometric modeling. More details about the package are available at the "Getting started with bimets" vignette. FRB/US model definition has been translated into a **bimets** compliant syntax and is available to R users as the `FRB__MODEL` dataset, which contains the textual definition of the model, using **bimets** `MDL` compliant syntax, i.e., Model Description Language:

```
R> #load bimets
R> library(bimets)

R> #load FRB/US MDL definition
R> data(FRB__MODEL)

R> #print first equations in model definition
R> cat(substring(FRB__MODEL,1,1615))

MODEL

$DOWNLOADED FROM federalreserve.gov AND CONVERTED TO BIMETS MDL IN Feb, 2024

$FRB/US is a large-scale estimated general equilibrium model of the U.S. economy
$that was developed at the Federal Reserve Board, where it has been in use since 1996
$for forecasting, analysis of policy options, and research projects.

$-----------------------------------------------------------------------

$ ENDOGENOUS SECTION

$----------------------------------------------
$Financial Sector
$Monetary policy indicator for both thresholds
$DMPTMAX equals one when either the unemployment threshold or
$the inflation threshold is breached.
IDENTITY> dmptmax
IF> dmptlur>=dmptpi
EQ> dmptmax=
dmptlur
IDENTITY> dmptmax
IF> dmptlur<dmptpi
EQ> dmptmax=
dmptpi

$----------------------------------------------
$Federal funds rate, first diff
IDENTITY> delrff
EQ> delrff=
TSDELTA(rff)

$----------------------------------------------
$Financial Sector
$Monetary policy indicator for unemployment threshold
$DMPTLUR equals zero when the unemployment rate is above its
$threshold (LURTRSH) one when it is below. A logistic function
$smoothes the transition, improving solution convergence properties.
IDENTITY> dmptlur
EQ> dmptlur=
1/(1+EXP(25*(lur-lurtrsh)))

$----------------------------------------------
$Financial Sector
$Monetary policy indicator for inflation threshold
$DMPTPI equals zero when expected inflation is below its threshold
$and one when it is above. A logistic function smoothes the
$transition, improving solution convergence properties.
IDENTITY> dmptpi
```

```
EQ> dmptpi=
1/(1+EXP(-25*(zpic58-pitrsh)))
```

bimets users can save the model definition into a text file, then inspect and modify it in RStudio or in any other text editor, by using the following commands:

```
R> #define file path
R> modelDefinitionFile <- file('~/FRB__MODEL.txt')
R> #save FRB definition in the text file
R> writeLines(FRB__MODEL,modelDefinitionFile)
R> #close connection
R> close(modelDefinitionFile)
```

The **bimets** dataset FRB__MCAP__WP__MODEL contains the MCE version of the FRB/US model, wherein, as said before, 14 equations have been modified accounting for rational expectations (see section 4.2).

Original FRB/US time series are provided in a CSV text file containing quarterly data from 1962 up to 2173. These data are available to R users in the LONGBASE dataset, which includes the list of all endogenous and exogenous time series.

```
R> #load FRB/US model data
R> data(LONGBASE)

R> #print GDP in 2022-2024
R> TABIT(LONGBASE$xgdp,TSRANGE = c(2022,1,2024,1))

      Date, Prd., LONGBASE$xgdp

   2022 Q1, 1   ,  21738.87
   2022 Q2, 2   ,  21708.16
   2022 Q3, 3   ,  21851.13
   2022 Q4, 4   ,  21989.98
   2023 Q1, 1   ,  22112.33
   2023 Q2, 2   ,  22225.35
   2023 Q3, 3   ,  22490.69
   2023 Q4, 4   ,  22561.72
   2024 Q1, 1   ,  22655.04
```

FRB/US MDL definition can be translated into a **bimets** model by using the LOAD_MODEL function, as usual:

```
R> #create the bimets model
R> model <- LOAD_MODEL(modelText = FRB__MODEL)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "FRB__MODEL":
    0 behaviorals
  284 identities
    0 coefficients
...LOAD MODEL OK
```

```
R> #print a sample of endogenous variables
R> model$vendog[1:10]

 [1] "dmptmax" "delrff"  "dmptlur" "dmptpi"  "dmptr"   "dpadj"   "dpgap"
 [8] "ebfi"    "ebfin"   "ec"

R> #print a sample of exogenous variables
R> model$vexog[1:10]

 [1] "lurtrsh" "pitrsh"  "d83"     "pkir"    "ddockm"  "uemot"   "emptrt"
 [8] "ddockx"  "ufcbr"   "rfnict"

R> #print GDP equation
R> model$identities$xgdp$eqFull

[1] "TSDELTALOG(xgdp)=(0.9985)*TSDELTALOG(xfs)+(0.6264)*TSDELTALOG(ki)+(-0.6249)*TSDELTALOG(TSLAG(ki));"
```

4. The econometric excercises

4.1. Dynamic simulation in a monetary policy shock

The first econometric exercise proposed by the Federal Reserve is a dynamic simulation of the FRB/US model under a monetary policy shock. The simulation is operated from 2040-Q1 to 2045-Q4, after the `rffintay` time series, defined as *"Value of eff. federal funds rate given by the inertial Taylor rule"*, is shocked by 100 base points in 2040-Q1.

python code follows:

```
import pandas

from pyfrbus.frbus import Frbus
from pyfrbus.sim_lib import sim_plot
from pyfrbus.load_data import load_data

# Load data
data = load_data("LONGBASE.TXT")

# Load model
frbus = Frbus("model.xml")

# Specify dates
start = pandas.Period("2040Q1")
end = start + 23

# Standard configuration, use surplus ratio targeting
data.loc[start:end, "dfpdbt"] = 0
data.loc[start:end, "dfpsrp"] = 1
```

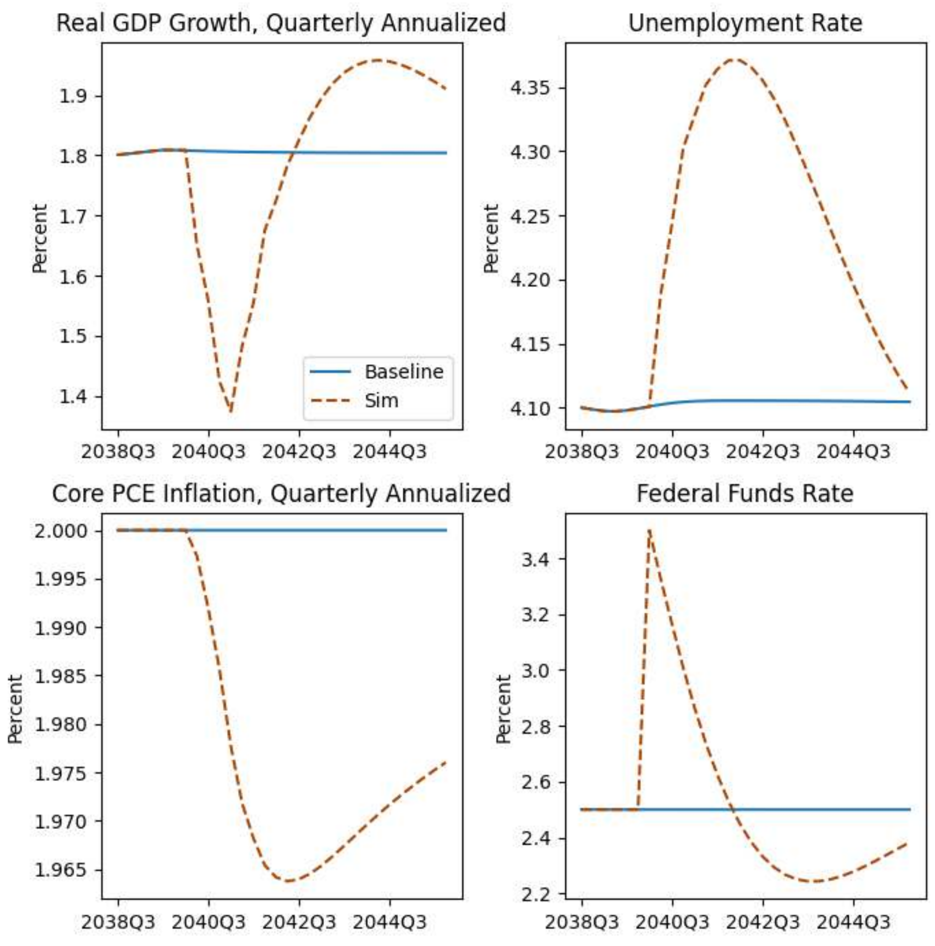
```
# Solve to baseline with adds
with_adds = frbus.init_trac(start, end, data)

# 100 bp monetary policy shock
with_adds.loc[start, "rffintay_aerr"] += 1

# Solve
sim = frbus.solve(start, end, with_adds)

# View results
sim_plot(with_adds, sim, start, end)
```

R version of the same exercise follows:

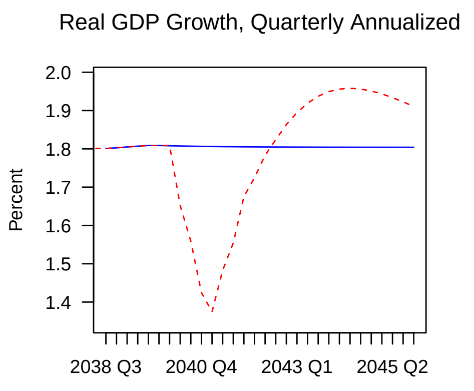
```
R> library(bimets)

R> # Load data
R> data(LONGBASE)

R> # Load model
R> data(FRB__MODEL)
R> model <- LOAD_MODEL(modelText = FRB__MODEL)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "FRB__MODEL":
    0 behaviorals
  284 identities
    0 coefficients
...LOAD MODEL OK

R> # Load data into model
R> model <- LOAD_MODEL_DATA(model, LONGBASE, quietly=TRUE)

R> # Specify dates
R> start <- c(2040,1)
R> end <- normalizeYP(start+c(0,23),4)

R> # Standard configuration, use surplus ratio targeting
R> model$modelData$dfpdbt[[start,end]] <- 0
R> model$modelData$dfpsrp[[start,end]] <- 1

R> # Solve to baseline with adds
R> model <- SIMULATE(model,
                     simType='RESCHECK',
                     TSRANGE=c(start,end),
                     ZeroErrorAC = TRUE,
                     quietly=TRUE)

R> # 100 bp monetary policy shock
R> trac <- model$ConstantAdjustmentRESCHECK
R> trac$rffintay[[start]] <- trac$rffintay[[start]]+1
```

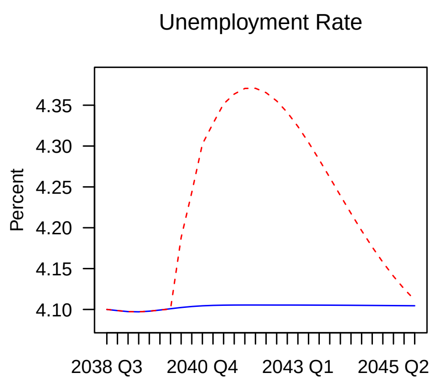
```
R> # Solve
R> model <- SIMULATE(model,
                     simAlgo = 'NEWTON',
                     TSRANGE = c(start,end),
                     ConstantAdjustment = trac,
                     BackFill = 12,
                     quietly=TRUE)

R> # View results
R> sim_plot(model,c(start,end),1)
```

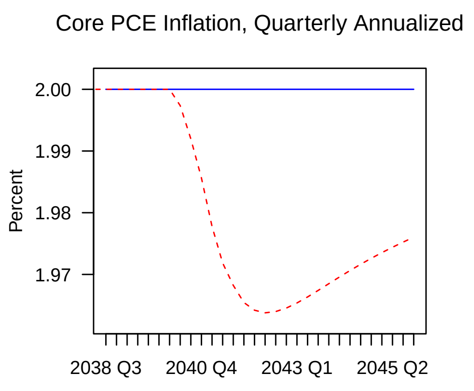
python code produces the following charts:

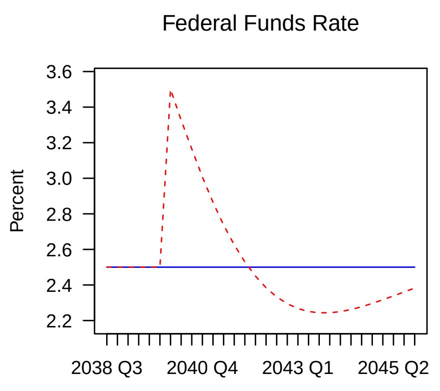


On the other hand, **bimets** code produces very similar results:









4.2. Rational expectations

The second econometric exercise proposed by the Federal Reserve is the simulation of a rational expectations variation of the FRB/US model (i.e, MCE, Model-Consistent Expectations) under a monetary policy shock. Again, the `rffintay` time series, is shocked by 100 base points.

14 equations of the original model (i.e. `zdivgr`, `zgap05`, `zgap10`, `zgap30`, `zpi10`, `zpi10f`, `zpib5`, `zpic30`, `zpic58`, `zpicxfe`, `zpieci`, `zrff10`, `zrff30`, `zrff5`), have been modified and present a forward-looking definition.

For example, in the MCE version of the model, the `zdivgr` equation presents a forward-looking reference to its future values:
`zdivgr=(0.0097572642574346l7)*TSLEAD(hgynid)+(0.9902427357425654)*TSLEAD(zdivgr)`

Original simulation time range has been lowered from 60 years to 2 years in order to reduce calculation time in examples (see *"Computational details"* on section 5).

python code follows:

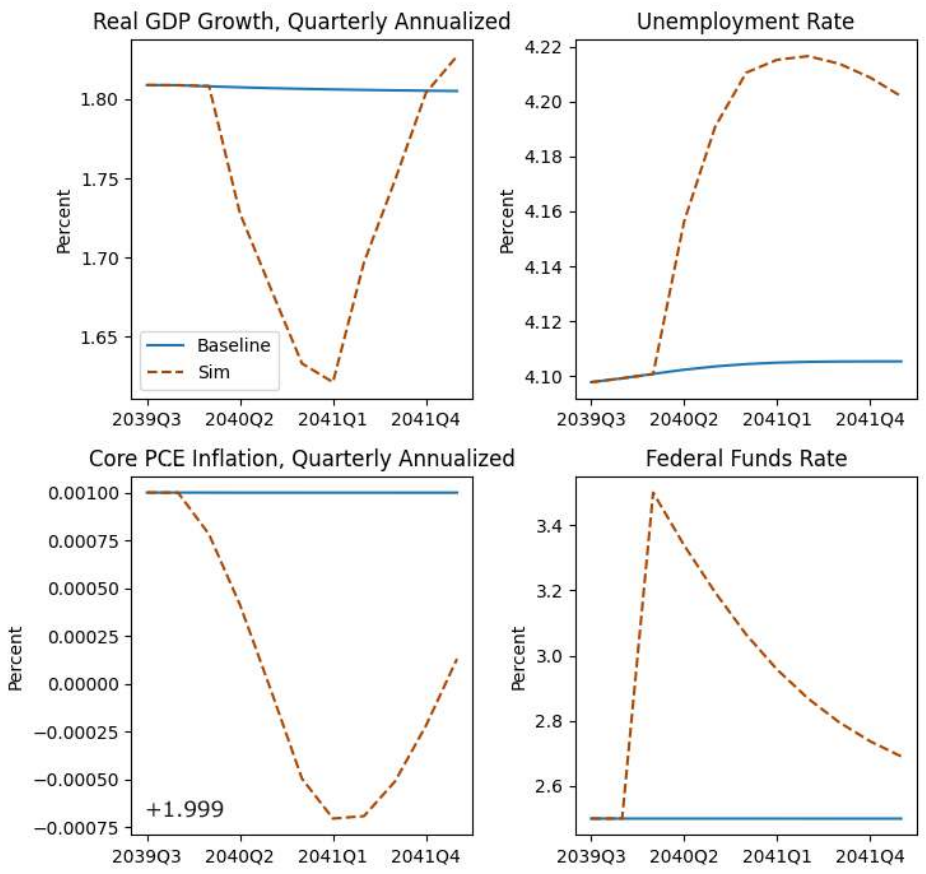
```
import pandas

from pyfrbus.frbus import Frbus
from pyfrbus.sim_lib import sim_plot
from pyfrbus.load_data import load_data

# Load data
data = load_data("./LONGBASE.TXT")

# Load model
frbus = Frbus("./model.xml", mce="mcap+wp")

# Specify dates
start = pandas.Period("2040q1")
end = start + 8

# Standard MCE configuration, use surplus ratio targeting, rstar endogenous in long run
data.loc[start:end, "dfpdbt"] = 0
data.loc[start:end, "dfpsrp"] = 1
data.loc[start:end, "drstar"] = 0
data.loc[(start+4):end, "drstar"] = 1

# Solve to baseline with adds
with_adds = frbus.init_trac(start, end, data)

# 100 bp monetary policy shock and solve
with_adds.loc[start, "rffintay_aerr"] += 1

# Solve
sim = frbus.solve(start, end, with_adds)

# View results
sim_plot(with_adds, sim, start, end)
```

R version of the same exercise follows:

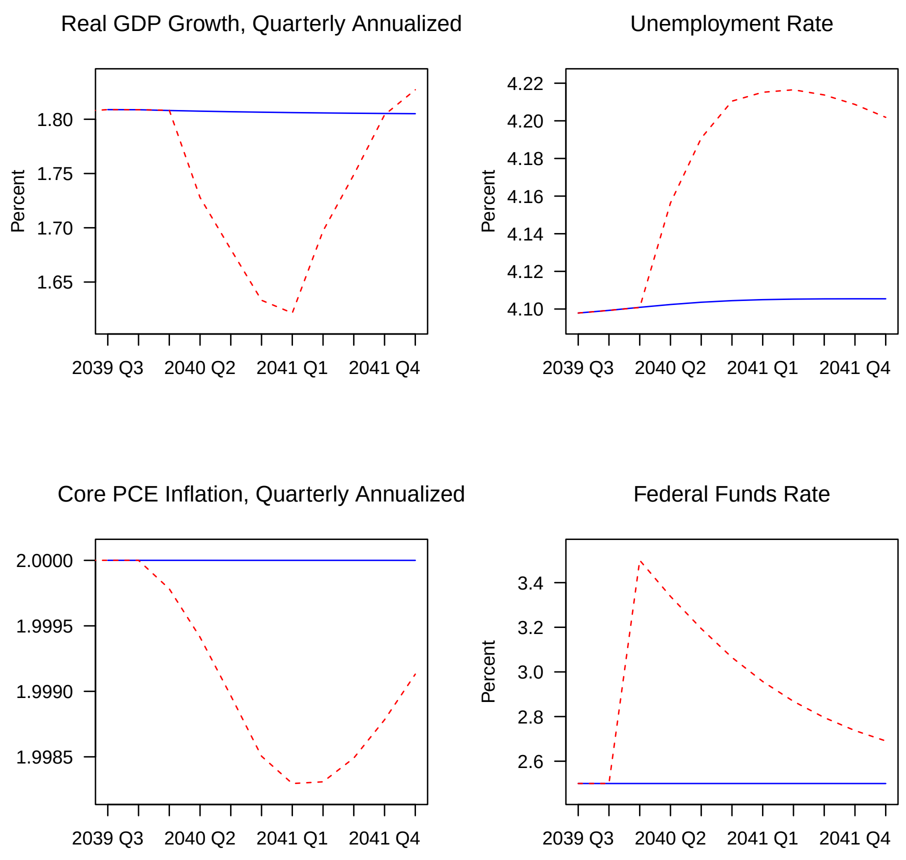
```
R> library(bimets)

R> # Load data
R> data(LONGBASE)

R> # Load model
R> data(FRB__MCAP__WP__MODEL)
R> model <- LOAD_MODEL(modelText = FRB__MCAP__WP__MODEL)
```

```
Analyzing behaviorals...
Analyzing identities...
This is a forward looking model...
Loaded model "FRB__MCAP__WP__MODEL":
    0 behaviorals
  284 identities
    0 coefficients
...LOAD MODEL OK

R> # Load data into model
R> model <- LOAD_MODEL_DATA(model, LONGBASE, quietly=TRUE)

R> # Specify dates
R> start <- c(2040,1)
R> end <- normalizeYP(start+c(0,8),4)

R> # Standard MCE configuration, use surplus ratio targeting, rstar endogenous in long run
R> model$modelData$dfpdbt[[start,end]] <- 0
R> model$modelData$dfpsrp[[start,end]] <- 1
R> model$modelData$drstar[[start,end]] <- 0
R> model$modelData$drstar[[normalizeYP(start+c(0,4),4),end]] <- 1

R> # Solve to baseline with adds
R> model <- SIMULATE(model,
                     simType = 'RESCHECK',
                     TSRANGE = c(start,end),
                     ZeroErrorAC = TRUE,
                     quietly=TRUE)

R> # 100 bp monetary policy shock
R> shock <- model$ConstantAdjustmentRESCHECK
R> shock$rffintay[[start]] <- shock$rffintay[[start]]+1

R> # Solve
R> model <- SIMULATE(model,
                     simAlgo = 'NEWTON',
                     TSRANGE = c(start,end),
                     ConstantAdjustment = shock,
                     BackFill = 12,
                     quietly=TRUE)

R> # View results
R> sim_plot(model,c(start,end),2)
```

python code produces the following charts:



On the other hand, **bimets** code produces very similar results:



4.3. Auto-correlation on tracking residuals

The 3rd econometric exercise proposed by the Federal Reserve is a dynamic simulation of the FRB/US model (backward-looking version) wherein historical tracking residuals have been forward extended by using an auto-correlation (i.e., *persistence*) factor of `0.5`.

python code follows:

```
import pandas

from pyfrbus.frbus import Frbus
from pyfrbus.sim_lib import sim_plot
from pyfrbus.load_data import load_data
from pyfrbus.time_series_data import TimeSeriesData

# Load data
data = load_data("./LONGBASE.TXT")
```

```
# Load model
frbus = Frbus("./model.xml")

# Specify dates
start = pandas.Period("2040Q1")
end = start + 24

# Standard configuration, use surplus ratio targeting
data.loc[start:end, "dfpdbt"] = 0
data.loc[start:end, "dfpsrp"] = 1

# Use non-inertial Taylor rule
data.loc[start:end, "dmptay"] = 1
data.loc[start:end, "dmpintay"] = 0

# Enable thresholds
data.loc[start:end, "dmptrsh"] = 1

# Arbitrary threshold values
data.loc[start:end, "lurtrsh"] = 6.0
data.loc[start:end, "pitrsh"] = 3.0

# Solve to baseline with adds
with_adds = frbus.init_trac(start, end, data)

# Zero tracking residuals for funds rate and thresholds
with_adds.loc[start:end, "rfftay_trac"] = 0
with_adds.loc[start:end, "rffrule_trac"] = 0
with_adds.loc[start:end, "rff_trac"] = 0
with_adds.loc[start:end, "dmptpi_trac"] = 0
with_adds.loc[start:end, "dmptlur_trac"] = 0
with_adds.loc[start:end, "dmptmax_trac"] = 0
with_adds.loc[start:end, "dmptr_trac"] = 0

# Shocks vaguely derived from historical residuals
with_adds.loc[start:start + 3, "eco_aerr"] = [-0.002, -0.0016, -0.0070, -0.0045]
with_adds.loc[start:start + 3, "ecd_aerr"] = [-0.0319, -0.0154, -0.0412, -0.0838]
with_adds.loc[start:start + 3, "eh_aerr"] = [-0.0512, -0.0501, -0.0124, -0.0723]
with_adds.loc[start:start + 3, "rbbbp_aerr"] = [0.3999, 2.7032, 0.3391, -0.7759]
with_adds.loc[start:start + 8, "lhp_aerr"] = [-0.0029,-0.0048,-0.0119,-0.0085, ...
-0.0074,-0.0061,-0.0077,-0.0033,-0.0042,]

# Set up time-series object
d = TimeSeriesData(with_adds)
# Roll off residuals with 0.5 persistence
rho = 0.5
# Set range
```

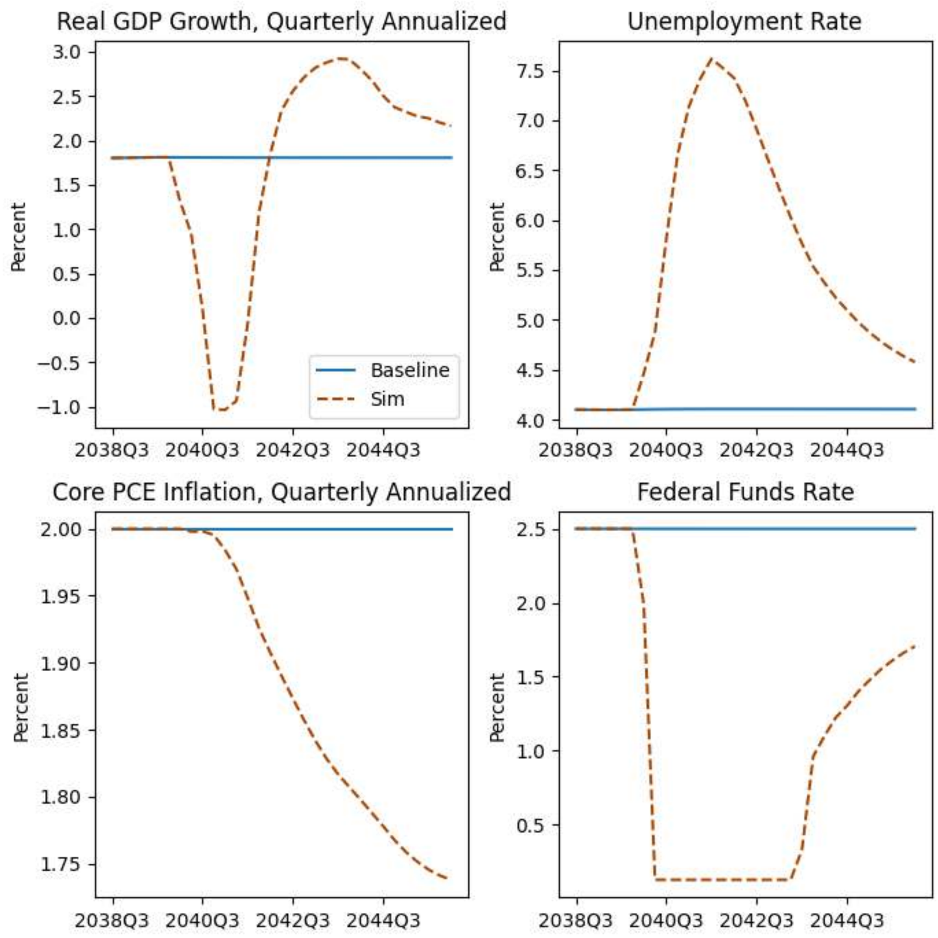
```
d.range = pandas.period_range(start + 4, end)
d.eco_aerr = rho * d.eco_aerr(-1)
d.ecd_aerr = rho * d.ecd_aerr(-1)
d.eh_aerr = rho * d.eh_aerr(-1)
d.rbbbp_aerr = rho * d.rbbbp_aerr(-1)
d.range = pandas.period_range(start + 9, end)
d.lhp_aerr = rho * d.lhp_aerr(-1)

# Adds so that thresholds do not trigger before shocks are felt
with_adds.loc[start, "dmptr_aerr"] = -1
with_adds.loc[start :  start + 2, "dmptlur_aerr"] = -1

# Solve
sim = frbus.solve(start, end, with_adds)

# View results, unemployment threshold binds
sim_plot(with_adds, sim, start, end)
```

R version of the same exercise follows:

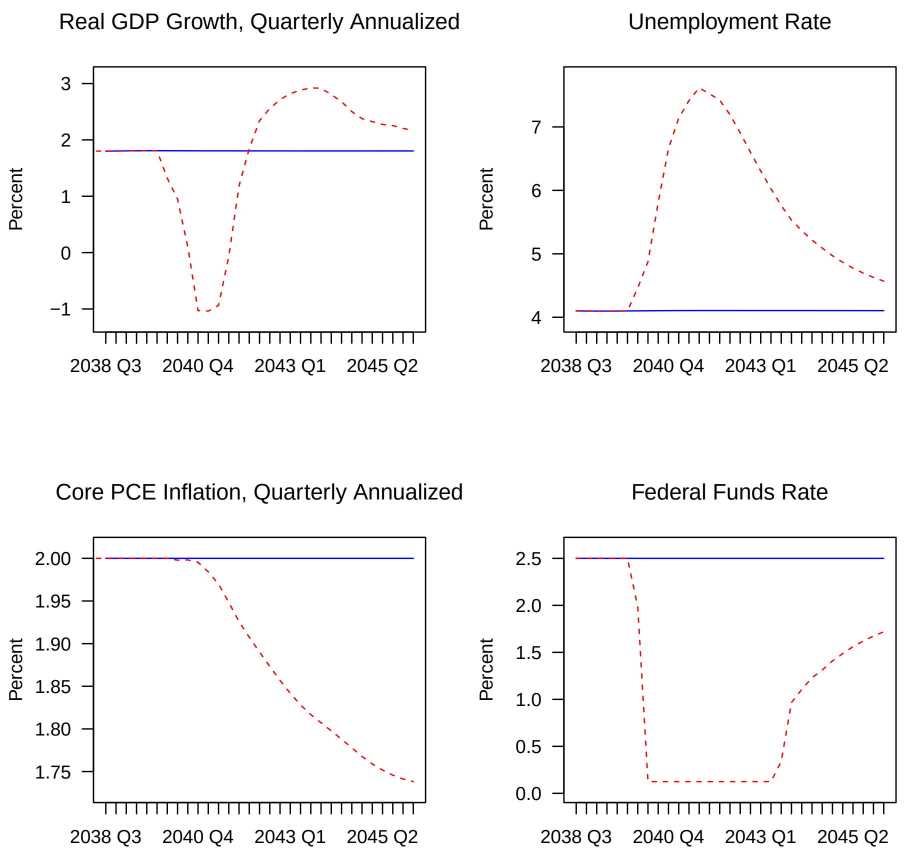
```
R> library(bimets)

R> # Load data
R> data(LONGBASE)

R> # Load model
R> data(FRB__MODEL)
R> model <- LOAD_MODEL(modelText = FRB__MODEL)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "FRB__MODEL":
    0 behaviorals
  284 identities
    0 coefficients
...LOAD MODEL OK

R> # Load data into model
R> model <- LOAD_MODEL_DATA(model, LONGBASE, quietly=TRUE)

R> # Specify dates
R> start <- c(2040,1)
R> end <- normalizeYP(start+c(0,24),4)

R> # Standard configuration, use surplus ratio targeting
R> model$modelData$dfpdbt[[start,end]] <- 0
R> model$modelData$dfpsrp[[start,end]] <- 1

R> # Use non-inertial Taylor rule
R> model$modelData$dmptay[[start,end]] <- 1
R> model$modelData$dmpintay[[start,end]] <- 0
```

```
R> # Enable thresholds
R> model$modelData$dmptrsh[[start,end]] <- 1

R> # Arbitrary threshold values
R> model$modelData$lurtrsh[[start,end]] <- 6
R> model$modelData$pitrsh[[start,end]] <- 3

R> # Solve to baseline with adds
R> model <- SIMULATE(model,
                     simType = 'RESCHECK',
                     TSRANGE = c(start,end),
                     ZeroErrorAC = TRUE,
                     quietly=TRUE)

R> # Get tracking residuals
R> trac <- model$ConstantAdjustmentRESCHECK

R> # Zero tracking residuals for funds rate and thresholds
R> trac$rfftay[[start,end]] <- 0
R> trac$rffrule[[start,end]] <- 0
R> trac$rff[[start,end]] <- 0
R> trac$dmptpi[[start,end]] <- 0
R> trac$dmptlur[[start,end]] <- 0
R> trac$dmptmax[[start,end]] <- 0
R> trac$dmptr[[start,end]] <- 0

R> # Shocks vaguely derived from historical residuals
R> aerr <- list()
R> aerr$eco <- TSERIES(c(-0.002, -0.0016, -0.0070, -0.0045),START=start,FREQ=4)
R> aerr$ecd <- TSERIES(c(-0.0319, -0.0154, -0.0412, -0.0838),START=start,FREQ=4)
R> aerr$eh <- TSERIES(c(-0.0512, -0.0501, -0.0124, -0.0723),START=start,FREQ=4)
R> aerr$rbbbp <- TSERIES(c(0.3999, 2.7032, 0.3391, -0.7759),START=start,FREQ=4)
R> aerr$lhp <- TSERIES(c(-0.0029,-0.0048,-0.0119,-0.0085,-0.0074,-0.0061,-0.0077,-0.0033,-0.0042),
              START=start,FREQ=4)

R> # Roll off residuals with 0.5 persistence
R> rho <- 0.5
R> aerr$eco <- TSEXTEND(aerr$eco,UPTO=end,EXTMODE='MYRATE',FACTOR=rho)
R> aerr$ecd <- TSEXTEND(aerr$ecd,UPTO=end,EXTMODE='MYRATE',FACTOR=rho)
R> aerr$eh <- TSEXTEND(aerr$eh,UPTO=end,EXTMODE='MYRATE',FACTOR=rho)
R> aerr$rbbbp <- TSEXTEND(aerr$rbbbp,UPTO=end,EXTMODE='MYRATE',FACTOR=rho)
R> aerr$lhp <- TSEXTEND(aerr$lhp,UPTO=end,EXTMODE='MYRATE',FACTOR=rho)

R> # Adds so that thresholds do not trigger before shocks are felt
R> aerr$dmptr <- TSERIES(c(-1),START=start,FREQ=4)
R> aerr$dmptlur <- TSERIES(c(-1,-1,-1),START=start,FREQ=4)

R> # Create Constant Adjustments for SIMULATE op.
R> for (idx in names(aerr)) trac[[idx]] <- trac[[idx]]+aerr[[idx]]

R> # Solve
R> model <- SIMULATE(model,
                     simAlgo = 'NEWTON',
                     TSRANGE = c(start,end),
                     ConstantAdjustment = trac,
                     BackFill = 12,
                     quietly=TRUE)

R> # View results, unemployment threshold binds
R> sim_plot(model,c(start,end),3)
```

python code produces the following charts:



On the other hand, **bimets** code produces very similar results:



4.4. Endogenous targeting

The 4th econometric exercise proposed by the Federal Reserve is an endogenous targeting exercise. The `mcontrol` procedure in the **pyfrbus** package calculates the constant adjustments values to be applied to 5 instruments variables (i.e., `eco`, `lhp`, `picxfe`, `rff`, `rg10p`, all in the `inst` list) such that the simulated values for the endogenous variables in the target list `targ` (i.e., `xgdp`, `lur`, `picxfe`, `rff`, `rg10`) are equal to the values of the arbitrary trajectories in the `traj` list.

In a similar way in R, the **bimets** `RENORM` procedure determines the values for the `INSTRUMENT` exogenous variables (i.e., the constant adjustments in the `inst` list in this exercise) that allow the objective `TARGET` endogenous values to be achieved, with respect to the constraints given by the model equations.

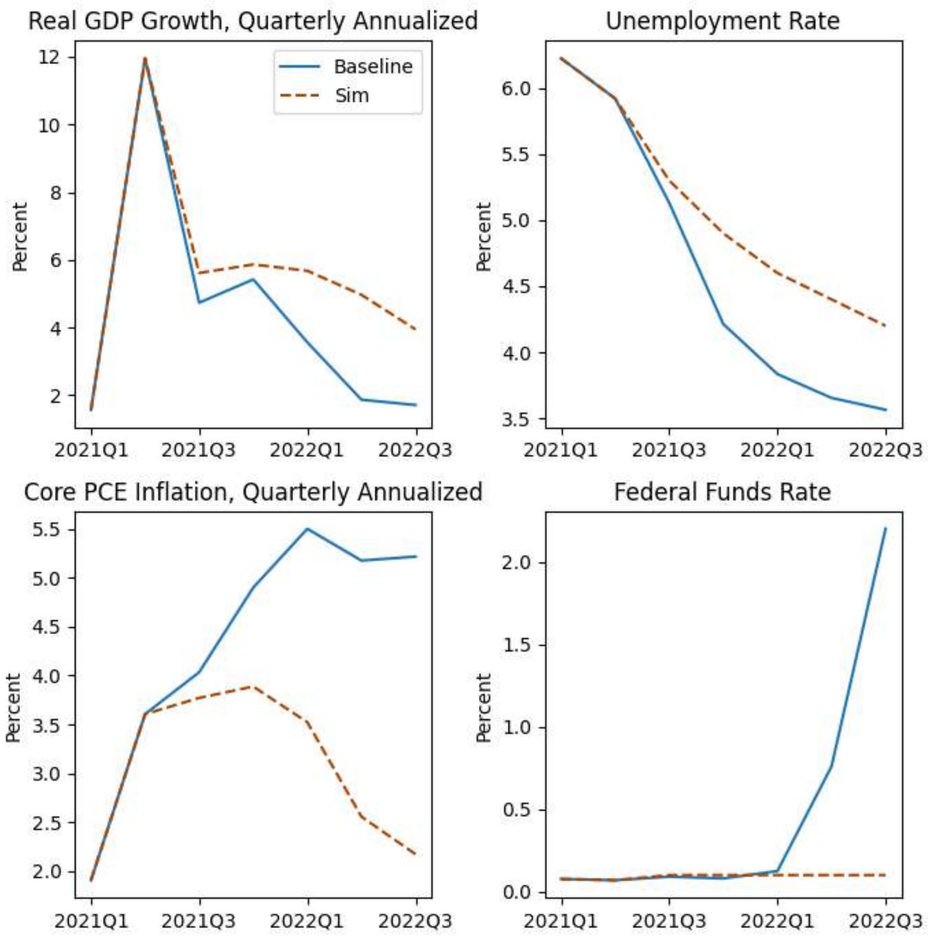
python code follows:

```
import pandas
from numpy import array, cumprod
from pyfrbus.frbus import Frbus
from pyfrbus.sim_lib import sim_plot
from pyfrbus.load_data import load_data

# Load data
data = load_data("./LONGBASE.TXT")

# Load model
frbus = Frbus("./model.xml")

# Specify dates
start = pandas.Period("2021Q3")
end = "2022Q3"

# Standard configuration, use surplus ratio targeting
data.loc[start:end, "dfpdbt"] = 0
data.loc[start:end, "dfpsrp"] = 1

# Solve to baseline with adds
with_adds = frbus.init_trac(start, end, data)

# Scenario based on 2021Q3 Survey of Professional Forecasters
with_adds.loc[start:end, "lurnat"] = 3.78

# Set up trajectories for mcontrol
with_adds.loc[start:end, "lur_t"] = [5.3, 4.9, 4.6, 4.4, 4.2]
with_adds.loc[start:end, "picxfe_t"] = [3.7, 2.2, 2.1, 2.1, 2.2]
with_adds.loc[start:end, "rff_t"] = [0.1, 0.1, 0.1, 0.1, 0.1]
with_adds.loc[start:end, "rg10_t"] = [1.4, 1.6, 1.6, 1.7, 1.9]

# Get GDP level as accumulated growth from initial period
gdp_growth = cumprod((array([6.8, 5.2, 4.5, 3.4, 2.7]) / 100 + 1) ** 0.25)
with_adds.loc[start:end, "xgdp_t"] = with_adds.loc[start - 1, "xgdp"] * gdp_growth

targ = ["xgdp", "lur", "picxfe", "rff", "rg10"]
traj = ["xgdp_t", "lur_t", "picxfe_t", "rff_t", "rg10_t"]
inst = ["eco_aerr", "lhp_aerr", "picxfe_aerr", "rff_aerr", "rg10p_aerr"]

# Run mcontrol
sim = frbus.mcontrol(start, end, with_adds, targ, traj, inst)

# View results
sim_plot(with_adds, sim, start, end)
```

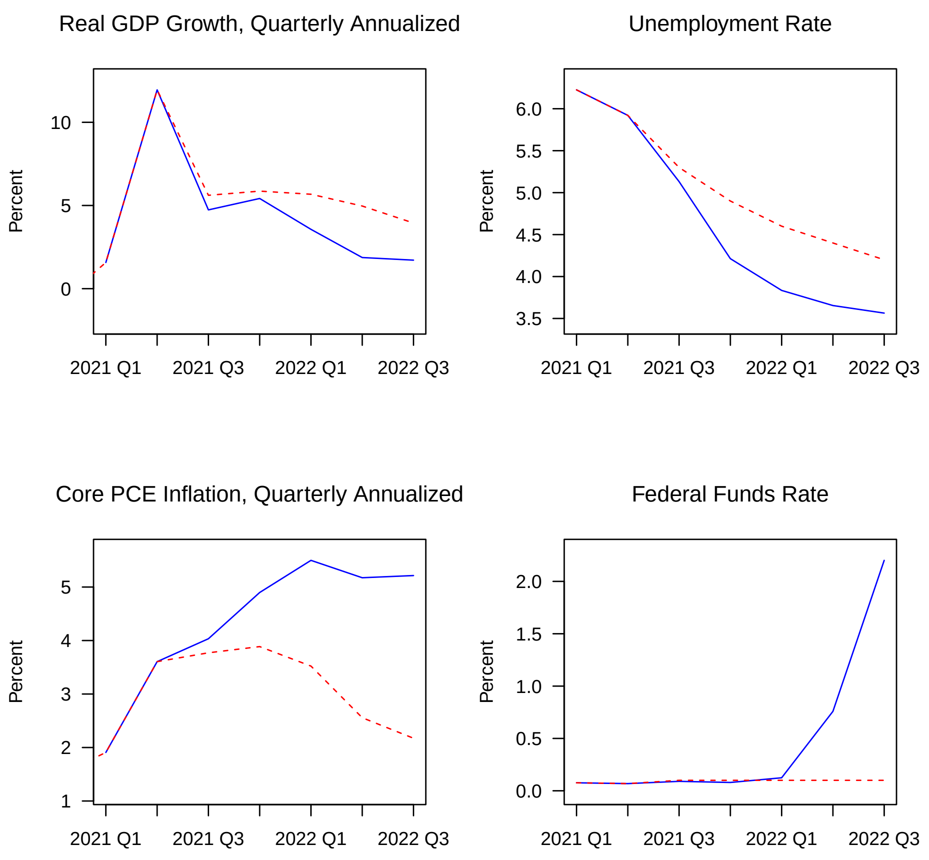
R version of the same exercise follows:

```
R> library(bimets)

R> # Load data
R> data(LONGBASE)

R> # Load model
R> data(FRB__MODEL)
R> model <- LOAD_MODEL(modelText = FRB__MODEL)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "FRB__MODEL":
    0 behaviorals
  284 identities
    0 coefficients
...LOAD MODEL OK

R> # Load data into model
R> model <- LOAD_MODEL_DATA(model, LONGBASE, quietly=TRUE)

R> # Specify dates
R> start <- c(2021,3)
R> end <- c(2022,3)

R> # Standard configuration, use surplus ratio targeting
R> model$modelData$dfpdbt[[start,end]] <- 0
R> model$modelData$dfpsrp[[start,end]] <- 1

R> # Solve to baseline with adds
R> model <- SIMULATE(model,
                     simType = 'RESCHECK',
                     TSRANGE = c(start,end),
                     ZeroErrorAC = TRUE
                     ,quietly=TRUE)

R> # Scenario based on 2021Q3 Survey of Professional Forecasters
R> model$modelData$lurnat[[start,end]] <- 3.78

R> # Set up trajectories for mcontrol
R> targ <- list()
R> targ$lur <- TSERIES(c(5.3, 4.9, 4.6, 4.4, 4.2),START=start,FREQ=4)
R> targ$picxfe <- TSERIES(c(3.7, 2.2, 2.1, 2.1, 2.2),START=start,FREQ=4)
R> targ$rff <- TSERIES(c(0.1, 0.1, 0.1, 0.1, 0.1),START=start,FREQ=4)
R> targ$rg10 <- TSERIES(c(1.4, 1.6, 1.6, 1.7, 1.9),START=start,FREQ=4)

R> # Get GDP level as accumulated growth from initial period
R> gdp_growth <- model$modelData$xgdp[[2021,2]]*CUMPROD((c(6.8,5.2,4.5,3.4,2.7) / 100 + 1) ** 0.25)
R> targ$xgdp <- TSERIES(gdp_growth,START=start,FREQ=4)

R> # define INSTRUMENT
R> inst <- c("eco", "lhp", "picxfe", "rff", "rg10p")

R> # Run RENORM
R> model <- RENORM(model,
                   simAlgo = 'NEWTON',
                   TSRANGE=c(start,end),
                   ConstantAdjustment = model$ConstantAdjustmentRESCHECK,
                   TARGET=targ,
                   INSTRUMENT=inst,
                   BackFill = 8,
                   quietly=TRUE)

R> # View results
R> sim_plot(model,c(start,end),4)
```

python code produces the following charts:



On the other hand, **bimets** code produces very similar results:



4.5. Stochastic simulation

The 5th econometric exercise proposed by the Federal Reserve is a stochastic simulation of the FRB/US model (backward-looking version). The `stochsim` procedure in the **pyfrbus** package performs a stochastic simulation by applying sequences of shocks to the model, as drawn randomly from historical residuals.

The `stochsim` procedure begins by drawing `nrepl` sequences of quarters over the periods `residstart` to `residend`, where the length of that sequence goes from `simstart` to `simend`. That is, for a particular replication, each quarter in the simulation period is randomly assigned a quarter from residual period. In that quarter of the simulation, all the 64 stochastic variables (specified with a `stochastic_type` tag in the XML model file) have a shock applied from a particular quarter in the residual period.

In a similar way in R, the **bimets** `STOCHSIMULATE` procedure allows users to shock, with arbitrary values, all the `StochReplica` replicas of variables that appear in the `StochStructure` list; the shocks, in this exercise, are the randomly sampled historical residuals in the time range from `residstart` to `residend`, and are added up to the 64 constant adjustment of the related endogenous variable listed as stochastic, thus replicating the python code. The initial mapping per period, between historical residuals and stochastic realizations, is stored in the `sampleHistoricalResidual` variable.

python code follows:

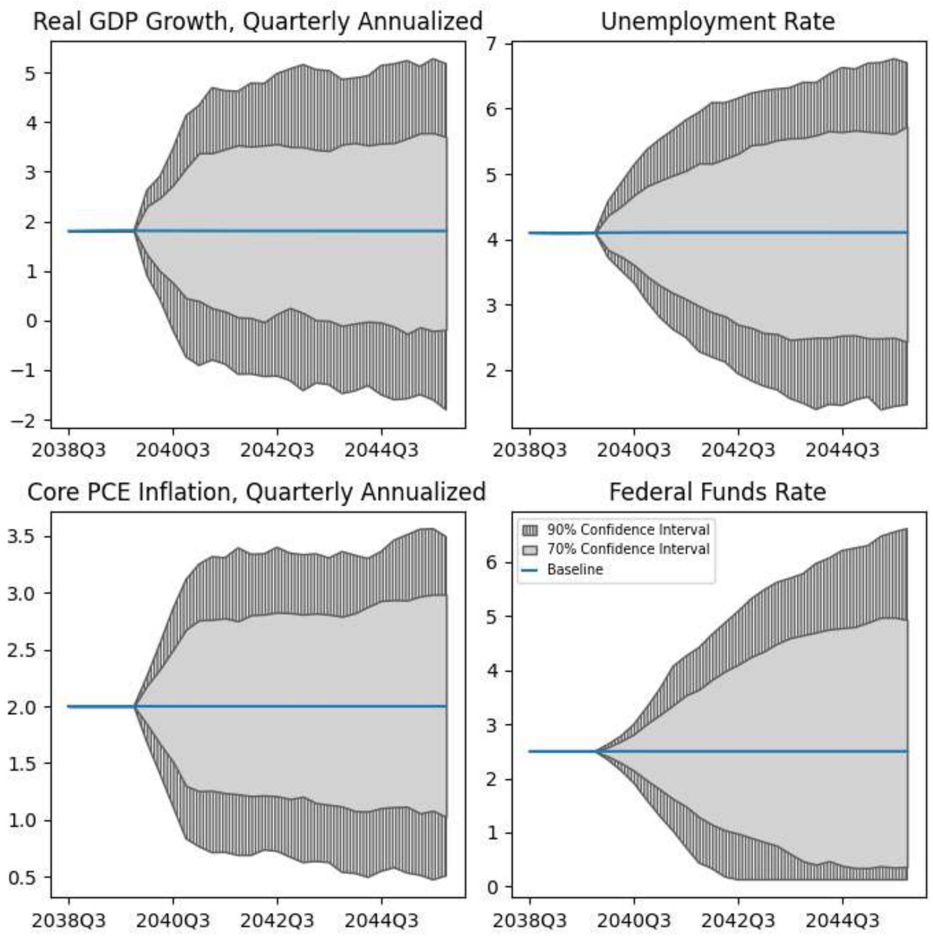
```
from pyfrbus.frbus import Frbus
from pyfrbus.sim_lib import stochsim_plot
from pyfrbus.load_data import load_data

# Load data
data = load_data("./LONGBASE.TXT")

# Load model
frbus = Frbus("./model.xml")

# Specify dates and other params
residstart = "1975q1"
residend = "2018q4"
simstart = "2040q1"
simend = "2045q4"
# Number of replications
nrepl = 1000
# Run up to 5 extra replications, in case of failures
nextra = 5

# Policy settings
data.loc[simstart:simend, "dfpdbt"] = 0
data.loc[simstart:simend, "dfpsrp"] = 1

# Compute add factors
# Both for baseline tracking and over history, to be used as shocks
with_adds = frbus.init_trac(residstart, simend, data)

# Call FRBUS stochsim procedure
solutions = frbus.stochsim(nrepl, with_adds, simstart, simend, residstart, residend, nextra=nextra)

# View results
stochsim_plot(with_adds, solutions, simstart, simend)
```

R version of the same exercise follows:

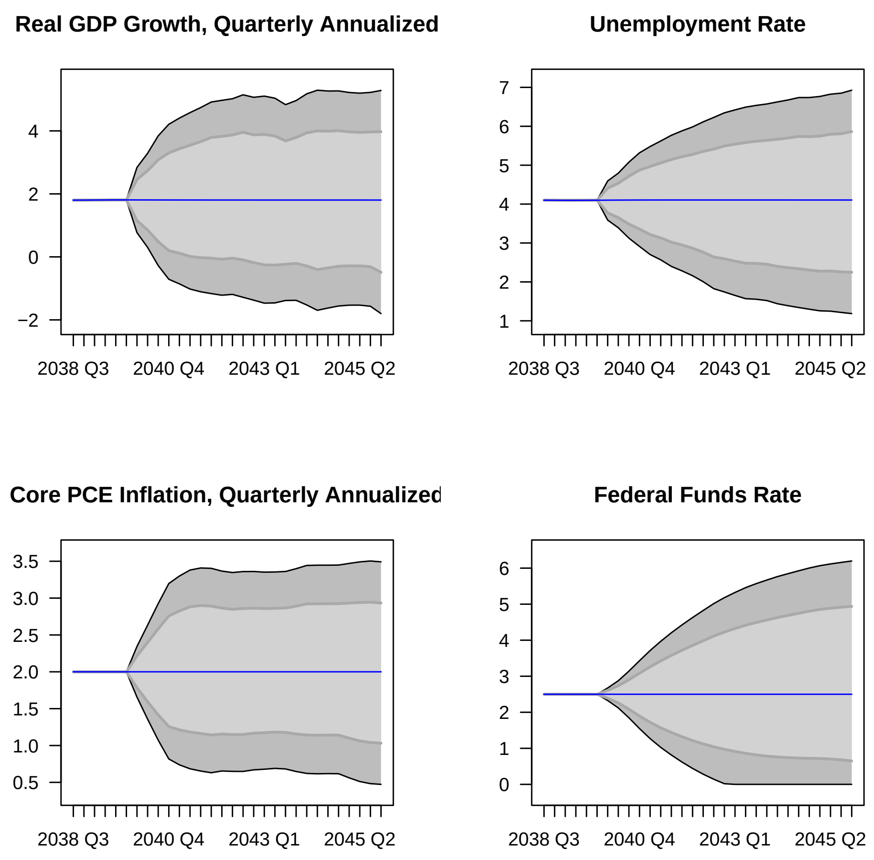
```
R> library(bimets)

R> # Load data
R> data(LONGBASE)

R> # Load model
R> data(FRB__MODEL)
R> model <- LOAD_MODEL(modelText = FRB__MODEL)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "FRB__MODEL":
    0 behaviorals
  284 identities
    0 coefficients
...LOAD MODEL OK

R> # Load data into model
R> model <- LOAD_MODEL_DATA(model, LONGBASE, quietly=TRUE)

R> # Specify dates and other params
R> residstart <- c(1975,1)
R> residend <- c(2018,4)
R> simstart <- c(2040,1)
R> simend <- c(2045,4)

R> # Number of replications
R> nrepl <- 1000

R> # Policy settings
R> model$modelData$dfpdbt[[simstart,simend]] <- 0
R> model$modelData$dfpsrp[[simstart,simend]] <- 1

R> # Compute add factors
R> # Both for baseline tracking and over history, to be used as shocks
R> model <- SIMULATE(model,
                     simType = 'RESCHECK',
                     TSRANGE = c(residstart,simend),
                     ZeroErrorAC = TRUE,
                     quietly=TRUE)

R> # Get tracking residuals
R> trac <- model$ConstantAdjustmentRESCHECK

R> # Set seed
R> set.seed(9)

R> # 64 Stochastic vars as listed in XML FRB/US model file
R> stochasticVars <- c("ebfi","ecd","ech","eco","egfe","egfen","egfet","egfl","egse",
                     "egsen","egset","egsl","eh","emo","emp","ex","fpxrr","fxgap",
                     "ugfsrp","gtn","gtr","gtrd","hmfpt","hqlfpr","hqlww","ki","leg",
                     "leo","lfpr","lhp","lurnat","lww","mfpt","pbfir","pcer",
                     "pcfr","pegfr","pegsr","phouse","phr","picxfe","pieci","pmo",
                     "poilr","pxr","rbbbp","rcar","rcgain","reqp","rfynic",
                     "rfynil","rg10p","rg30p","rg5p","rgfint","rme","tcin","tpn",
                     "trci","trp","trpt","uynicpnr","ynidn","ynirn")
```

```
R> # Pseudo random array that maps residuals range to simulation range for each replica
R> residualsLength <- NUMPERIOD(residstart,residend,4)+1
R> stochSimLength <- NUMPERIOD(simstart,simend,4)+1
R> sampleHistoricalResidual <- sample(1:residualsLength,stochSimLength*nrepl,replace=T)

R> # Create BIMETS stochastic structure
R> modelStochStructure <- list()
R> for (tmpStochVar in stochasticVars)
   {
       #see BIMETS reference manual for details on STOCHSIMULATE and StochStructure
       modelStochStructure[[tmpStochVar]] <- list()
       modelStochStructure[[tmpStochVar]]$TSRANGE <- TRUE
       modelStochStructure[[tmpStochVar]]$TYPE <- 'MATRIX'
       shockMatrix <- matrix(trac[[tmpStochVar]][sampleHistoricalResidual],
                             nrow=stochSimLength,ncol=nrepl)
       shockMatrix <- shockMatrix - ave(shockMatrix)
       modelStochStructure[[tmpStochVar]]$PARS <- shockMatrix
   }

R> # Call BIMETS stoch sim procedure
R> model <- STOCHSIMULATE(model,
                          simAlgo = 'NEWTON',
                          TSRANGE = c(simstart,simend),
                          StochStructure = modelStochStructure,
                          StochReplica = nrepl,
                          ConstantAdjustment = trac,
                          quietly=TRUE)

R> # View results
R> stochsim_plot(model,c(simstart,simend))
```

python code produces the following charts:



On the other hand, **bimets** code produces similar results, despite the random numbers generator being different between R and python:



5. Computational details and capabilities: pyfrbus vs bimets

Both R (R-4.2.0 optimized with Intel® MKL libraries) and python (Intel® python 3.7.9) code have been tested on Red Hat Enterprise Linux 8.8 with Intel® Xeon® Gold 6248 CPU @ 2.50GHz 40 cores and 1.2TB RAM; all econometric exercises on the backward-looking version of the FRB/US model are fast in both **bimets** and **pyfrbus**, having just a few seconds of computation time for the simulation in both scenarios. Endogenous targeting (see 4.4) is faster in **pyfrbus**, while stochastic simulation (see 4.5) is faster in **bimets**, but generally speaking the **pyfrbus** package relies on compiled code and, in scenarios that require heavy computational resources, e.g., rational expectation models, is faster than **bimets**, that is pure R code.

Indeed, **pyfrbus** is way faster than **bimets** in forward-looking large models with an extensive simulation time range. **pyfrbus** can simulate 60 years in FRB/US rational expectations exercise (see 4.2), with just a few minutes of execution time; in **bimets**, exclusively for that exercise, a FRB/US simulation with a time range greater than 10 years is not feasible. The python package relies on the `SuiteSparse` libraries, a suite of compiled C code that provides optimized algorithms for sparse matrices. Moreover, in **pyfrbus**, the model Jacobian is computed symbolically using **SymPy** and **SymEngine**, thus the computation time is very fast also in Netwon algorithms; on the other hand, there is no option for a numerically-approximated Jacobian. As a result, the platform supports a limited number of functions; thus, in **pyfrbus** only the following functions are supported in model equations: `log`, `exp`, `abs`, `min`, and `max`.

On the other hand, the **pyfrbus** package has been developed with a focus on the FRB/US model and presents the following limitations with respect to **bimets**, which is a general framework for econometric modeling:

– **pyfrbus** only supports quarterly models, while **bimets** supports models of any frequency listed in the time series constructor `TIMESERIES` (e.g., annual, semiannual, quarterly, monthly, daily, weekly, etc.);

– in **pyfrbus**, no estimation nor structural stability procedures are available, while in **bimets** users can insert coefficients in model equations, then perform estimation and structural stability analysis accounting for linear restrictions on coefficients, polynomial distributed lags, and residuals auto-correlation;

– **pyfrbus** does not support conditional evaluation in equations, while in **bimets** users can use the `IF>` statements in model equations;

– as said, in **pyfrbus**, due to the symbolic Jacobian implementation, only `log`, `exp`, `abs`, `min`, and `max` functions are allowed in model equations, while **bimets** allows for more complex time series transformations to be inserted directly into the model definition, e.g., `TSDELTA`, `TSDELTALOG`, `MOVAVG`, etc.; on the other hand, at the moment **bimets** does not support `MIN` and `MAX` functions in model equations, but users can use `IF>` statements in `MDL` model definitions in order to create an equivalent syntax, as in the FRB/US `MDL` version in these exercises; `MIN`, `MAX`, and other `MDL` functions will be available in the upcoming **bimets** versions.

– in **bimets**, detailed messages in error and in verbose operations can help users in identifying and solving grammatical and numerical issues;

– in time series operations, **pyfrbus** relies on **numpy** and **pandas** capabilities, while **bimets** allows for more complex time series transformations, e.g., (dis)aggregations, extension, merge, projection, etc.;

– optimal control for econometric models is a missing capability in **pyfrbus**;

– stochastic simulation in **pyfrbus** only allows for shocks randomly selected from past residuals tracking, while in **bimets** users have more flexibility, i.e., uniform and normal stochastic shocks, and arbitrary matrices of shocks, in arbitrary periods; moreover in **pyfrbus** the list of stochastic variables is hard-coded into the model definition, while in **bimets** users can change stochastic variables interactively in code by changing the `StochStructure` variable in the procedure's call;

– in **bimets**, constant adjustments are function's arguments, therefore users can disable or change, in arbitrary periods, the add-factor impact in model equilibrium, interactively in code with no need to modify the model data set, as required in **pyfrbus**;

- in **bimets**, users can exogenize an endogenous variable in arbitrary periods, while in **pyfrbus** exogenization is forced on the whole simulation time range;
- in **pyfrbus**, multiplier analysis is not available;
- **pyfrbus** is distributed only for Linux operating system, with instructions on how to run Linux code on a Microsoft® Windows machine. Moreover, the package has a complex list of dependencies (e.g., `UMFPACK`, `SuiteSparse`, `scipy`, `sympy`, `symengine`, `scikit`, `numpy`, `pandas`, etc.) and its installation could be difficult; **bimets**, which depends only on base `R` and `xts`, does not require any compilation, and is easy to install in Linux, Microsoft® Windows, and Apple® OS-X.

More details about **bimets** are available at the "Getting started with bimets" vignette.

References

[1] Andrea Luciani, Bank of Italy - *bimets: Time Series and Econometric Modeling.* R package version 4.0.1, https://CRAN.R-project.org/package=bimets/,
GIT: https://github.com/andrea-luciani/bimets

[2] Board of governors of The Federal Reserve System - *pyfrbus: a Python-based platform to run simulations with the FRB/US model..* python package version 1.0.0, https://www.federalreserve.gov/econres/us-models-about.htm

Affiliation:
Andrea Luciani
Bank of Italy
Directorate General for Economics, Statistics and Research
Via Nazionale, 91
00184, Rome - Italy
E-mail: `andrea.luciani@bancaditalia.it`