Computing General Equilibrium Models

Alexei Goumilevski
and
James Otterson,

An Overview

- We have developed toolboxes for analyzing and running Dynamic Stochastic General Equilibrium (DSGE) and Computable General Equilibrium (CGE) models.

- DSGE models are stochastic in nature, incorporating random shocks and primarily used for macroeconomic analysis. Their complexity arises from this stochasticity, making them more challenging to solve.

- CGE models, on the other hand, are deterministic and focus on analyzing economic policies and their impacts across various sectors. They are generally easier to solve, relying on deterministic equations, with a primary emphasis on inter-sectoral relationships and resource allocation.

- The solutions for these models require specialized techniques that are not readily available in the extensive array of Python's scientific modeling packages.

Why Developing a New Macroeconomic Platform in Python?

- **DYNARE** and **IRIS** are macroeconomic modeling toolboxes extensively utilized by economists at the Fund for **DSGE** modeling. Conversely, the General Algebraic Modeling System (**GAMS**), designed for optimizing linear, nonlinear, and mixed complementarity problems, is employed for **CGE** modeling.

- These platforms utilize commercial software such as **MATLAB**, **Troll**, and **GAMS**. However, **Troll/Fame** is no longer supported, and while **MATLAB** boasts a large user base, it is gradually losing popularity.

- **Python** has a rapidly growing user base. It is a high-level, versatile programming language known for its readability and simplicity, and it is free to use.

- **Julia** is also free; however, **Python** has a larger user base and a more extensive coding environment. Although **Julia** is faster, **Python** is making significant improvements in this area. Users can leverage the open-source **Numba** Just-In-Time compiler, which translates a subset of **Python** and **Numpy** into fast machine code.

Snow-Drop Package

- Although **Python** is a leader in the **ML** domain (e.g., **PyTorch**, **Sklearn**, **TensorFlow**), it is less competitive than **R**, **Stata**, and **Eviews** in the field of econometrics. The **Snow-Drop** package enhances **Python**'s capabilities in this area.

- It offers a flexible, powerful, and user-friendly framework for macroeconomic modeling, featuring tools for filtering, simulation, estimation, forecasting, and model diagnostics specifically for **DSGE** models.

- This framework can be applied to the analysis of **NK**, **RBC**, **Gap**, and **OLG** models.

- The software is written in Python, ensuring platform neutrality, and it can be run on Windows, Linux, Unix, and Mac operating systems.

Snow-Drop Package

- Model files are represented in a human-readable **YAML** format. The framework can parse simple model files from **IRIS**, **DYNARE**, **SIRIUS**, and **TROLL**.

- The framework parses the model file, checks its syntax for errors, and generates the corresponding **Python** function source code. It computes the **Jacobian** symbolically up to the third order.

- It utilizes **binary expression trees** to represent mathematical equations and **abstract syntax trees** to represent the structure of the function's code.

- The framework can execute forecasts with user-defined adjustments on a trajectory of some or all endogenous variables.

Snow-Drop Package

- Non-linear models are solved iteratively using **Newton's** method. The implemented algorithms include **ABLR** stacked matrices and **LBJ** forward-backward substitution methods.

- Linear models are addressed with the **Binder and Pesaran** method, **Anderson and Moore's** method, and two generalized **Schur** methods that replicate calculations utilized in **Dynare** and **Iris** software.

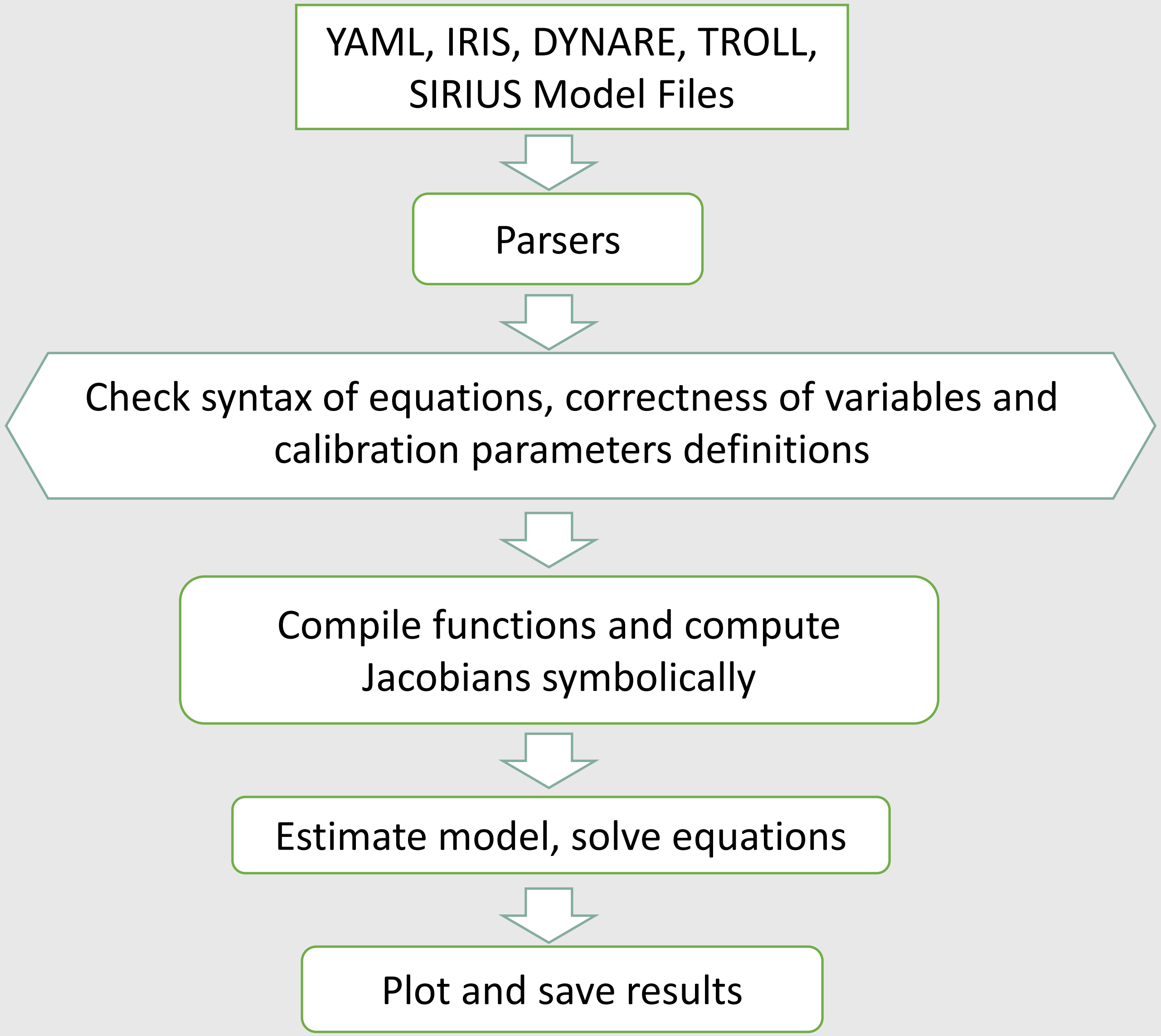
- For algebra involving large matrices, the framework employs the **Pypardiso** package. This serves as an interface to the Intel **MKL PARallel DIrect SOlver** library.

- **PARDISO** is a thread-safe library designed for solving large sparse linear systems of equations on shared-memory multi-core architectures.

Numerical Algorithms for Solving Model Equations

Algorithms	Description
LBJ	Juillard, Laxton, McAdam, Pioro algorithm (mimics DYNARE Toolbox perfect foresight solver)
ABLR	Armstrong, Black, Laxton, Rose algorithm
Villemot	Villemot Sebastien (mimics DYNARE Toolbox perturbations method solver)
Klein	Paul Klein (mimics IRIS Toolbox perturbations method solver)
BinderPesaran	Binder and Pesaran algorithm
AndersonMoore	Anderson and Moore algorithm

Flowchart of Code Execution



MATLAB + IRIS/DYNARE Versus Python Files

	MATLAB + IRIS/DYNARE	PYTHON
Declaration of model specification (equations, variables, parameters, shocks)	model.model or model.mod files	model.yaml file
Calibrated values and model object creation	readModel.m	Model file, excel or text files, python dictionary, model import
Read data and filter	run_filter.m	kalmanfilter.py
Projections and judgements	run_forecast.m	forecast.py and judgements.py
Reports, tables, and figures	report.new function	graphs.util.py and table.py

Model Specification

MATLAB+IRIS

```
% Global
lx_gdp_eq     = lx_gdp_eq{-1} +  g_x/4 + e_lx_gdp_eq;
g_x           = jx1*g_x{-1} + (1-jx1)*ss_g_x + e_g_x;
dot_x_gdp_eq = 4*(lx_gdp_eq-lx_gdp_eq{-1});
lx_gdp_gap    = lx_gdp - lx_gdp_eq;
lx_gdp_gap    = h3*lx_gdp_gap{-1} + e_lx_gdp_gap;

!transition_shocks
e_lx_gdp_gap, e_x_rn, e_dot_x_cpi, e_x_rr_eq
e_lx_gdp_eq
e_g_x

!parameters
ss_dot_x_cpi, ss_x_rr_eq, ss_g_x, jx1, h3

!measurement_variables
obs_lx_cpi
obs_lx_gdp_gap
obs_x_rn
obs_g_x
obs_lx_gdp

!measurement_equations
obs_lx_cpi          = lx_cpi        ;
obs_lx_gdp_gap      = lx_gdp_gap    ;
obs_x_rn            = x_rn          ;
obs_g_x = g_x + mes_g_x;
obs_lx_gdp     = lx_gdp;

!measurement_shocks
mes_g_x
```

Python

```
name:  Simple Real Business Cycle Model
symbols:
   log_variables: [Y,C]
   variables: [K,r,A]
   shocks: [ea]
   parameters: [beta,delta,gamma,rho,a]
equations:
    - 1/C = 1/C(1) * beta * (1 + r)
calibration:
   beta : 0.99
options:
   periods: [1]
   shock_values: [0.1]
```

Set Calibrated Values

MATLAB+IRIS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% MODEL: CALIBRATION AND SOLUTION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Parameter values
%% Aggregate demand block

% IS curve
% lgdp_gap = a1*lgdp_gap{+1} + a2*lgdp_gap{-1} - ...
%     a3*(lrr_gap+0*term40_gap) + a4*lz_gap + a5*lx_gdp_gap + ...
%     a6*lqforcmd_gap + e_lgdp_gap;

% a1 - forward-looking expectations of aggregate demand
p.a1 = 0.15;

% a2 - aggregate demand persistence
p.a2 = 0.7004;

% a3 - policy passthrough (impact of monetary policy on real economy)
p.a3 = 0.15;

% a4 - impact of REER on IS curve
p.a4 = 0.05;

% a5 - external demand impact
p.a5 = 0.15;

% a6 - mining sector passthrough
p.a6 = 0.023;
```

Python

name: QPM model
symbols:
  variables: [@*include* end_vars.model, …]
  shocks : [@include exo_vars.model, ...]
  parameters : [@include params.model, ...]
equations: @include model_eqs.model
 … extra equations
calibration:
  a1: 0.15
  …
options:
  range : ["2021,1,1","2050,1,1"]
  frequency: 1 # quarterly

Solve Model

MATLAB+IRIS

- **% model steady state**
- **m** = sstate(m, 'growth=‘, true, 'maxfunevals=‘, 50000);
- **mss** = get(**m**, 'sstate');

- **% model solution**
- **m** = **solve**(m);

- **% Save model to mat file**
- save('model.mat', **'m'**);

Python

- **# Create model object**
- **model** = importModel(model_file_path)
- **# Compute steady state**
- ss_values, ss_growth = driver.findSteadyStateSolution(**model**)

- **# Solve model**
- **solve**(**model**)

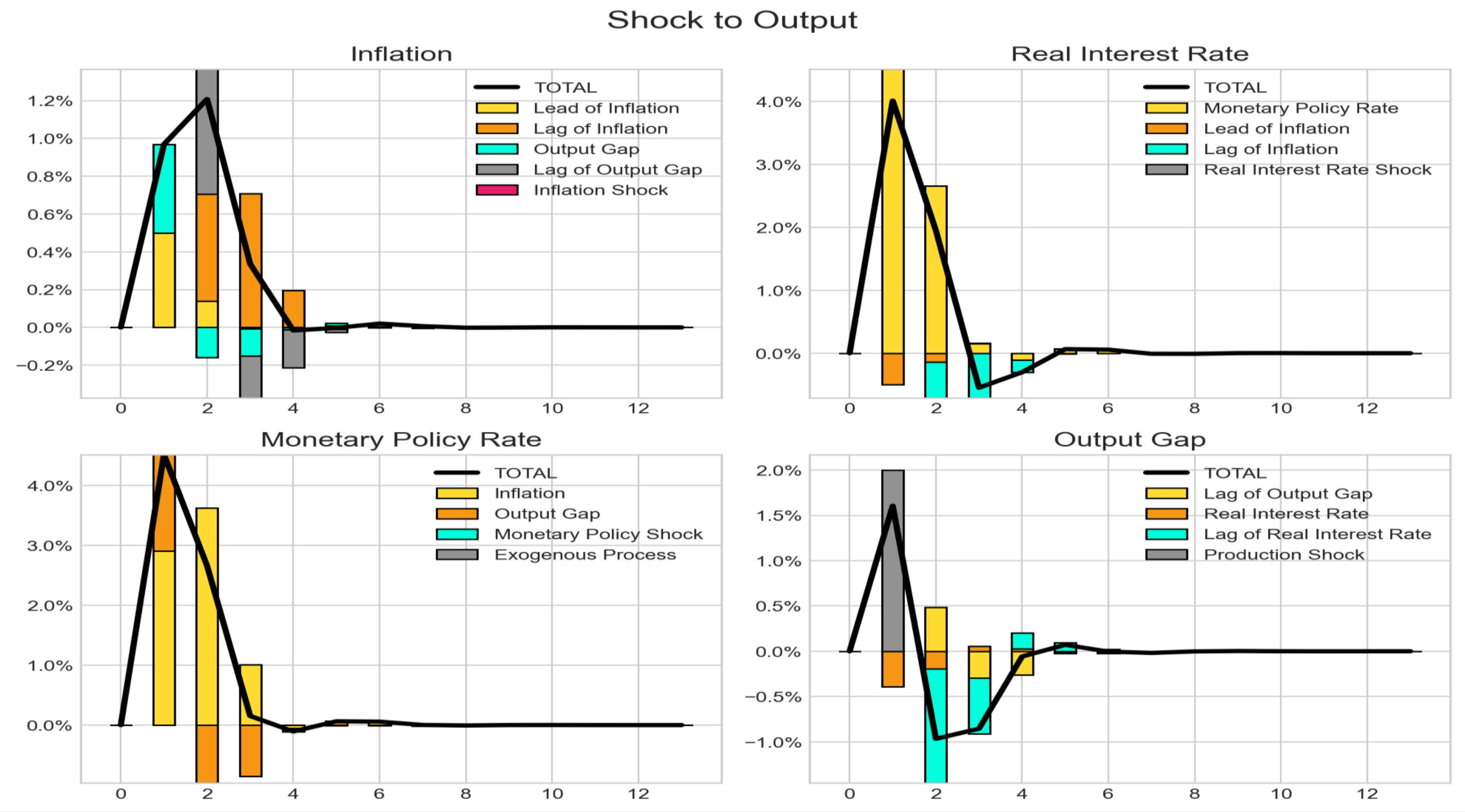
- **# Save model to binary file format**
- saveModel(model_file_path, **model**)

Modeling Examples

Monetary Policy Model File

name: *Monetary policy model example*
symbols:
  variables: [PDOT,RR,RS,Y]
  exogenous: [ers]
  shocks: [ey]
  parameters: [g,p_pdot1,p_pdot2,p_pdot3,p_rs1,p_y1,p_y2,p_y3]
equations:
  - PDOT = p_pdot1*PDOT(+1) + (1-p_pdot1)*PDOT(-1) + p_pdot2*(g^2/(g-Y) - g) + p_pdot3*(g^2/(g-Y(-1)) - g)
  - RR = RS - p_pdot1*PDOT(+1) - (1-p_pdot1)*PDOT(-1)
  - RS = p_rs1*PDOT + Y + ers
  - Y = p_y1*Y(-1) - p_y2*RR - p_y3*RR(-1) + ey
calibration:
  # *parameters*
  g: 0.049
  p_pdot1: 0.414 #[*0.4,0.5,0.6,0.7*] # ← ***You can set time varying parameters. The last value will be used for the rest of this array***.
  ...
  std: **0.02**
  # *exogenous variables*
  # *ers*: [0,0,0,0,0.01,0]
  # *file*: [../../data/exog.csv]
options:
  T : 14
  periods: [**1**]

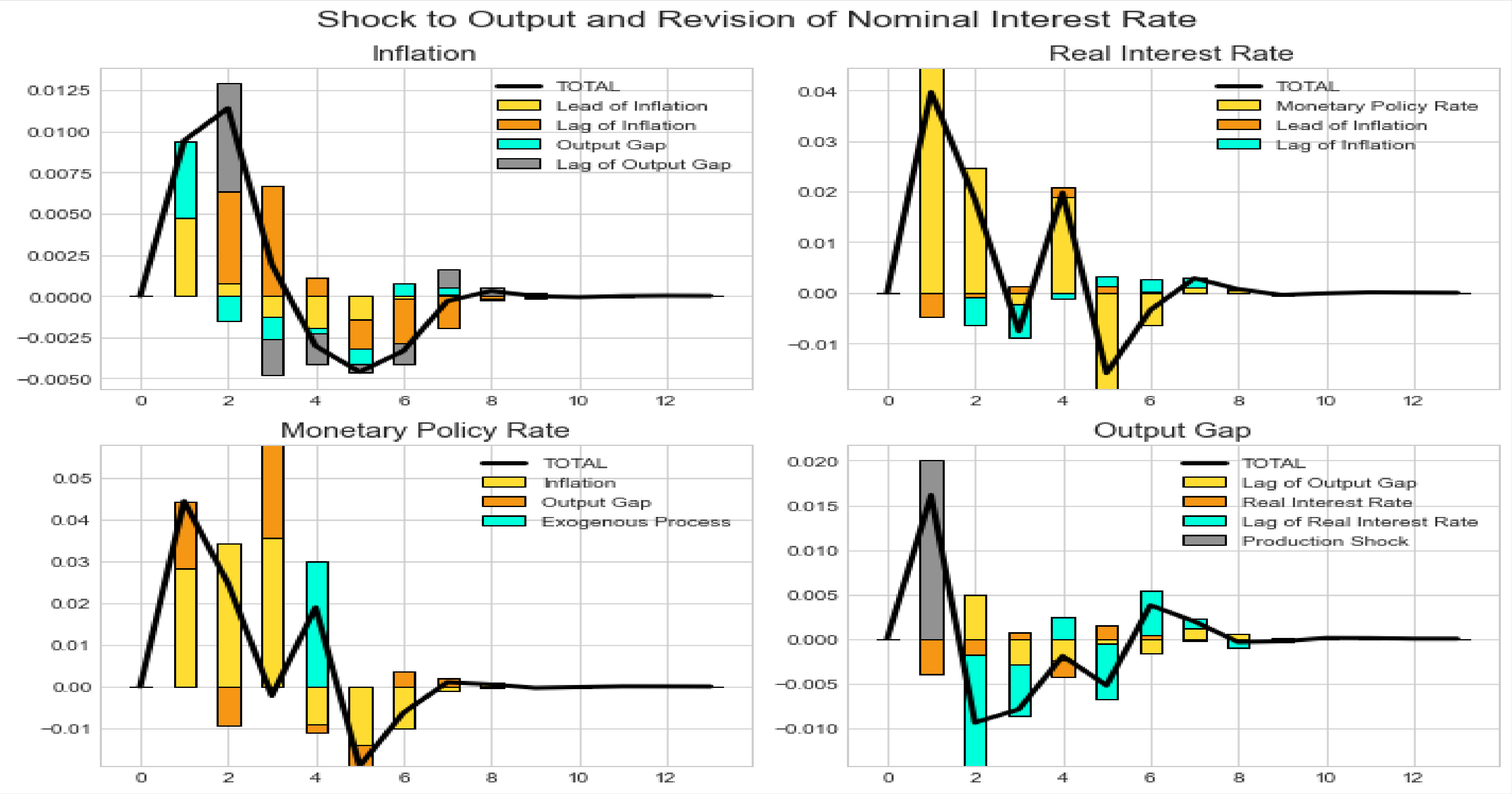
Monetary Policy Example



Python Code

```
model = importModel(model_file_path)

# Set shocks
model.options["periods"] = [1]
model.options["shock_values"] = [0.02]

# Set exogenous variables. Revise Monetary Policy Interest Rate.
exog_data = {"ers": pandas.Series([0,0,0,0.03,0],[1,2,3,4,5])}
model.symbolic.exog_data = exog_data
model.calibration["exogenous"] = getExogenousSeries(model)

# Define list of variables for which decomposition plots are produced
decomp = ['PDOT','RR','RS','Y']

# Run simulations
y, dates = driver.run(model=model, decomp_variables=decomp, Plot=True)
```

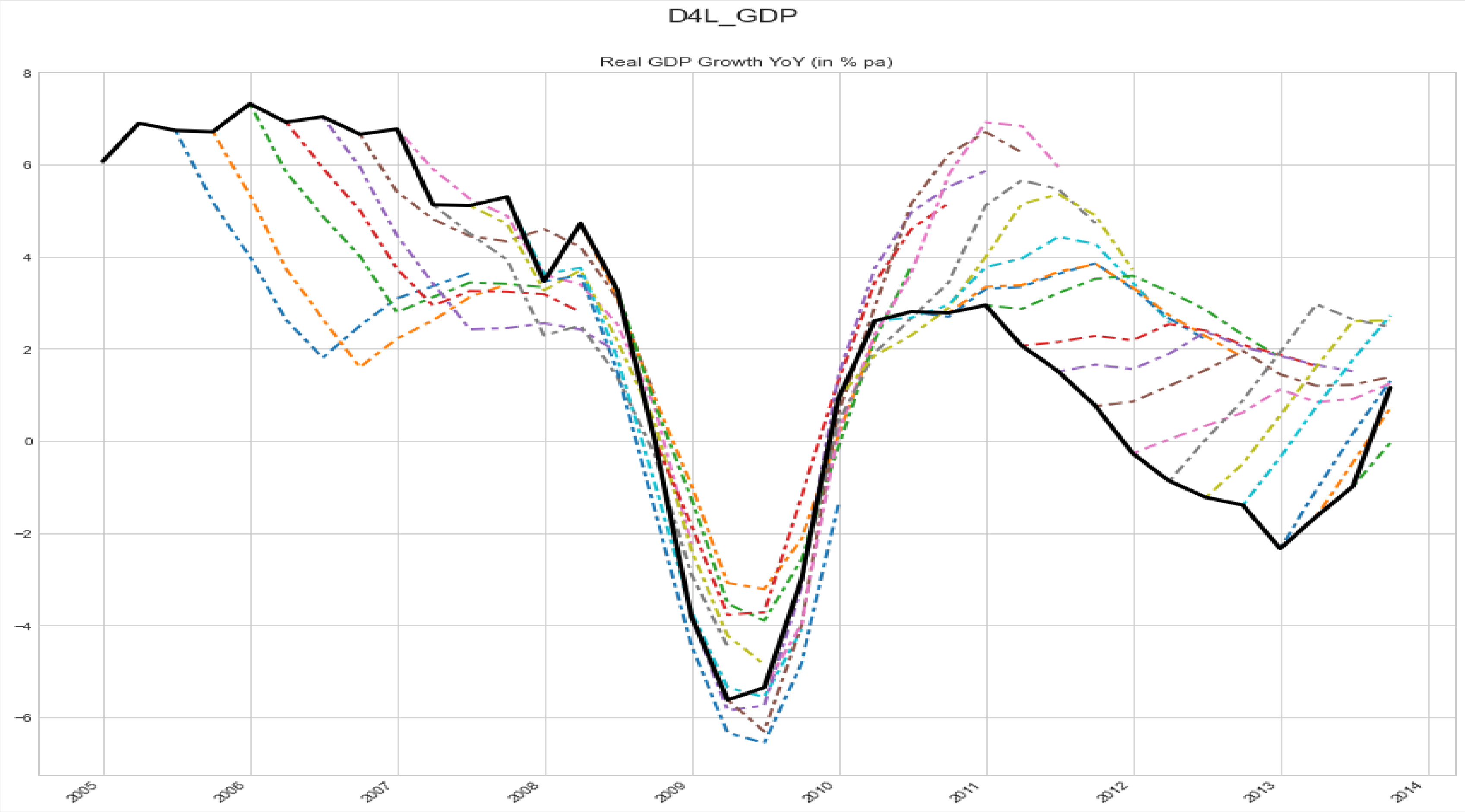
Monetary Policy Example



Historic Simulations

```
model = importModel(model_file_path, Solver=‘Klein')


# Set time span
stime = '2003-1-1’       #starting point of the first simulation
etime = '2016-12-31’   #the end of the known history)
simultion_range = pandas.date_range(start=stime, freq='QS’, end=etime)

#Beginning of the "loop"
for k in range( len(simulation_range)):
   t = sim_rng[k]
   # Simulation range is eight quarters (two years)
   f_time = t + dateutil.relativedelta(months=8*3)

   model.options[‘simulation_range'] = [[year, t.month, t.day],
                                        [f_time.year, f_time.month, f_time.day]]
   ....
   driver.run(model)
```

Historic Simulations



Kalman Filter

Create model object
model = importModel(model_file_path, Solver=“*Klein*", Filter="*Durbin_Koopman*",
Smoother="*Durbin_Koopman*", Prior="*Equilibrium*", measurement_file_path=*meas*)

# **Set simulation and filtration time ranges**
simulation_range = [[1997,1,1],[2013,12,1]]
filter_range = [[1998,1,1],[2013,12,1]]

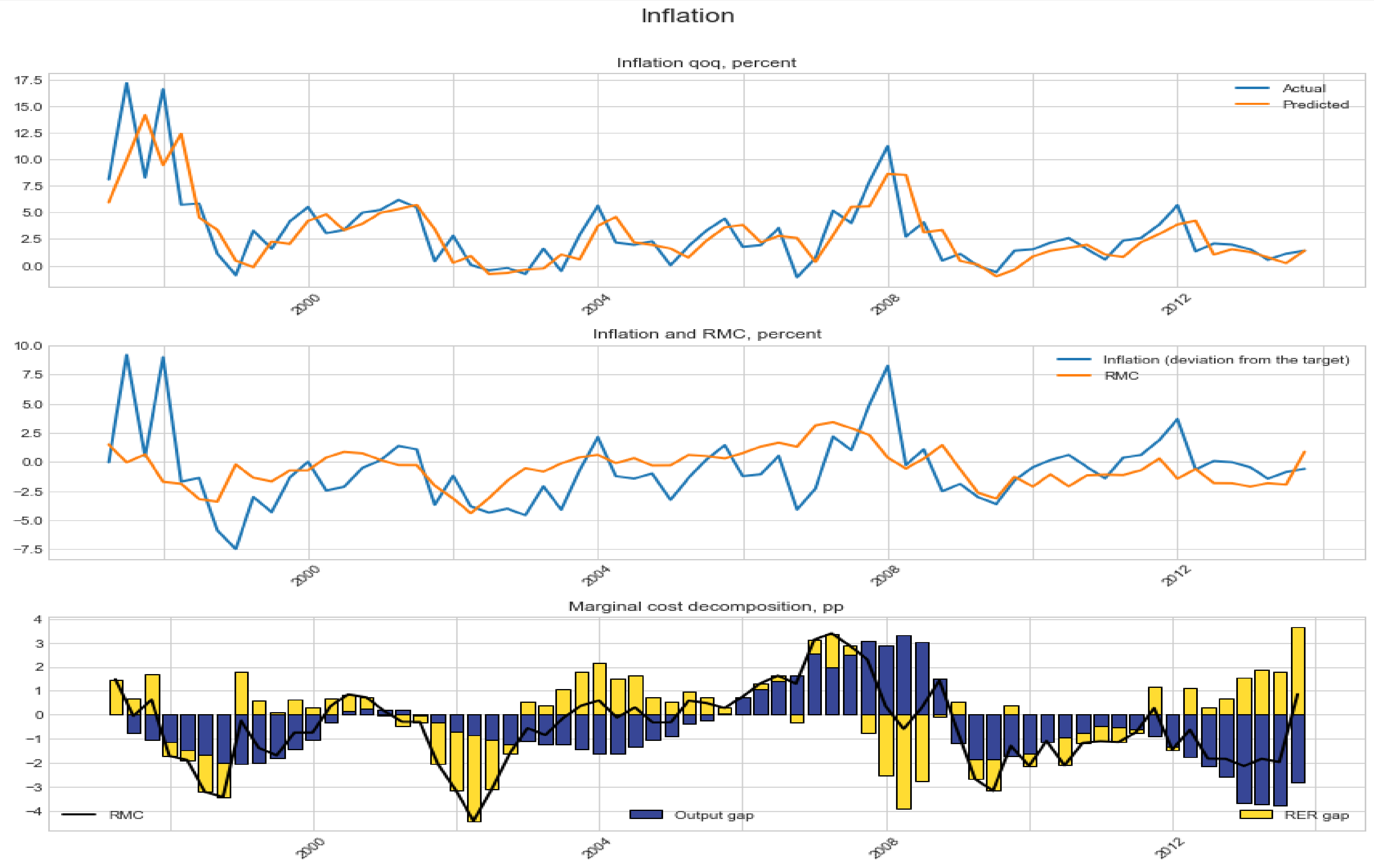
model.options['range'] = simulation_range
model.options['filter_range'] = filter_range

 # Set starting values of endogenous variables
model.setStartingValues(hist=meas)

Run Kalman filter. Get filtered and smoothed results, date range, filtered and smoothed shocks
y, dates, epsilonhat, etahat = kalman_filter(**model**, Output=True, fout=output_file_path)

MPAF Course Example

Continue

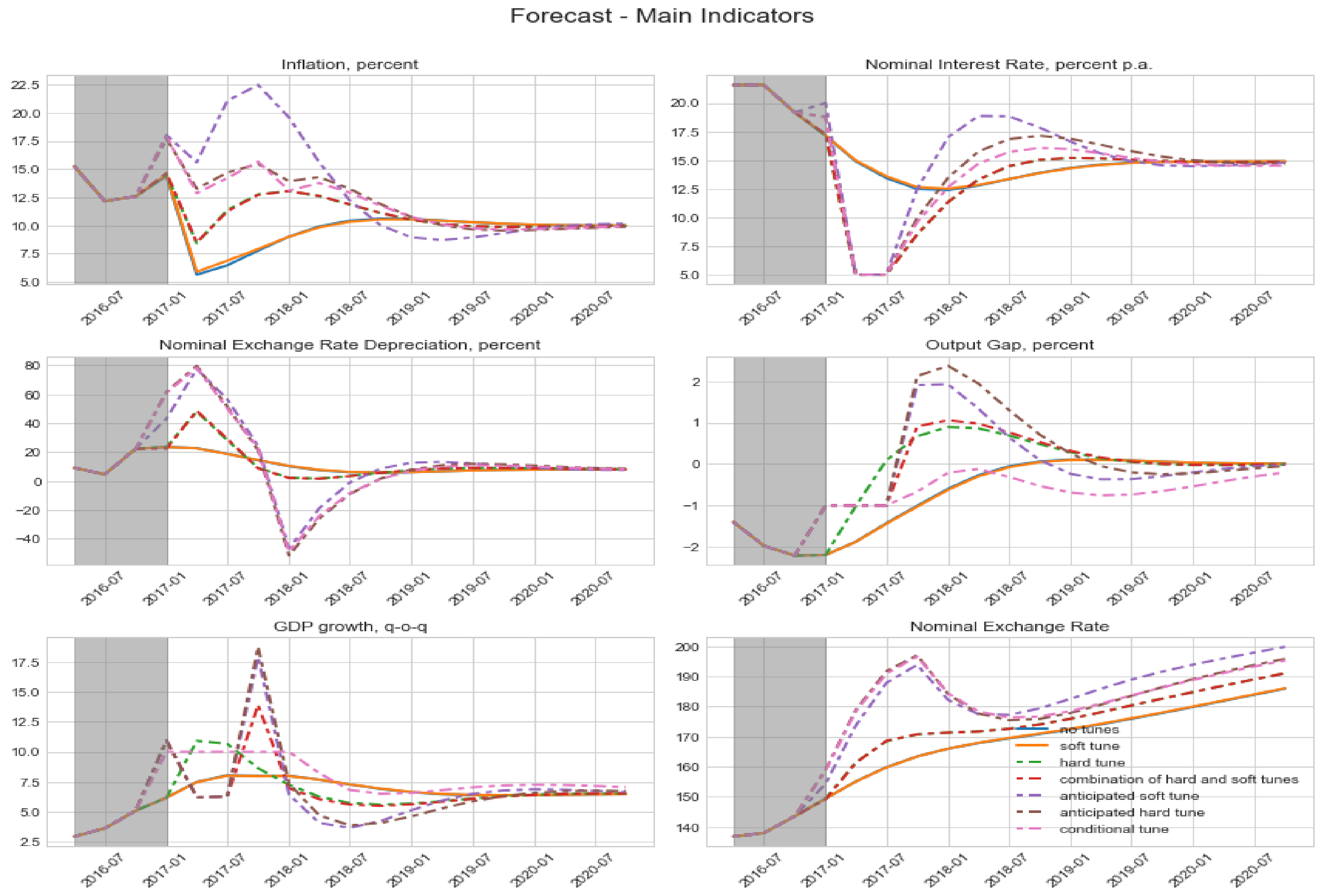


Inflation
Inflation qoq, percent
Actual
Predicted
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2000
2004
2008
2012
Inflation and RMC, percent
Inflation (deviation from the target)
RMC
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
2000
2004
2008
2012
Marginal cost decomposition, pp
4
3
2
1
0
−1
−2
−3
−4
RMC
Output gap
RER gap
2000
2004
2008
2012

User Can Impose Anticipated, Not Anticipated, and Conditional Shocks, and Judgments

```
## Combination of soft and hard tunes:
# Set shock for gap of output variable to 1% at period 3
d = {"SHK_L_GDP_GAP": [(3,1)]}
model.setShocks(d)

# Impose judgments
date_range = pandas.date_range(start, end, freq='QS')
m = {'L_GDP_GAP’: pandas.Series([-1.0, -1.0, -1.0], date_range)}
shocks_names  = ['SHK_L_GDP_GAP']

# Endogenize shock and exogenize output gap endogenous variable
model.swap(m, shocks_names)

# Run simulations
y, dates = driver.run(model)
```

And Impose Judgmental Adjustments



Model Parameters Estimation

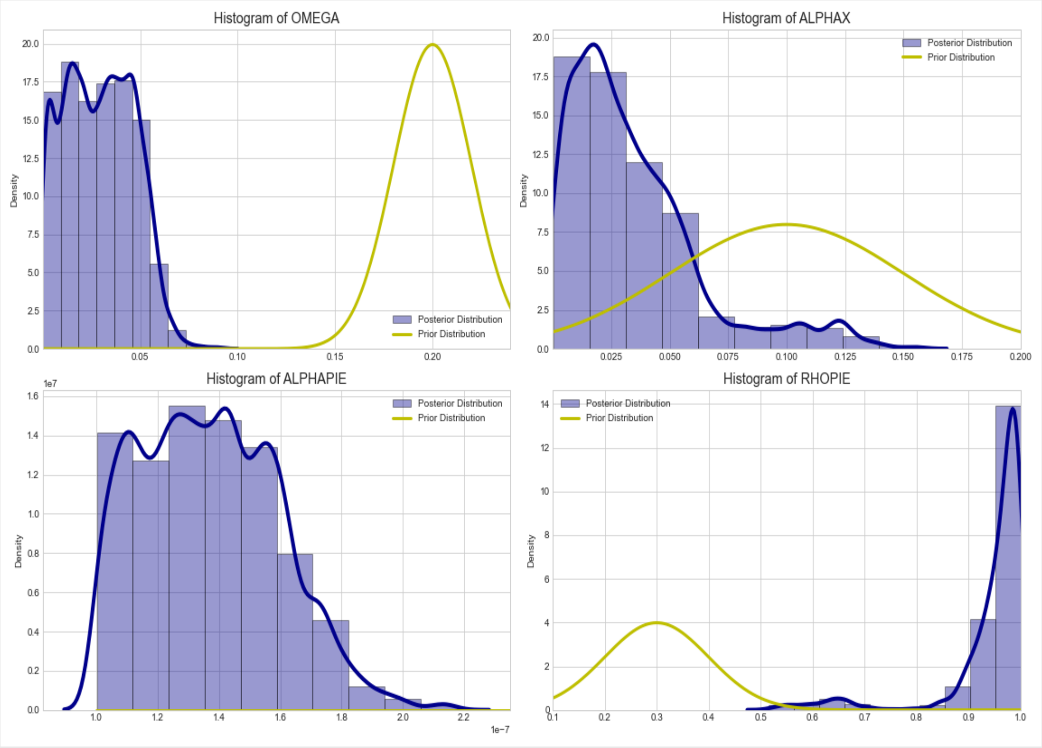
estimated_parameters:
  # Please choose one of the following distributions:
  # normal_pdf, lognormal_pdf, beta_pdf, gamma_pdf, t_pdf, weibull_pdf, inv_gamma_pdf,
  # inv_weibull_pdf, wishart_pdf, inv_wishart_pdf.
  #
  # PARAM NAME, INITVAL, LB, UB, PRIOR_SHAPE, PRIOR_P1, PRIOR_P2, PRIOR_P3, PRIOR_P4, PRIOR_P5
  # The first parameter is the parameter name, the second is the initial value, the third and
  # the fourth are the lower and the upper bounds, the fifth is the prior shape,
  # the sixth to tenth are prior parameters (mean, standard deviation, shape, etc...).

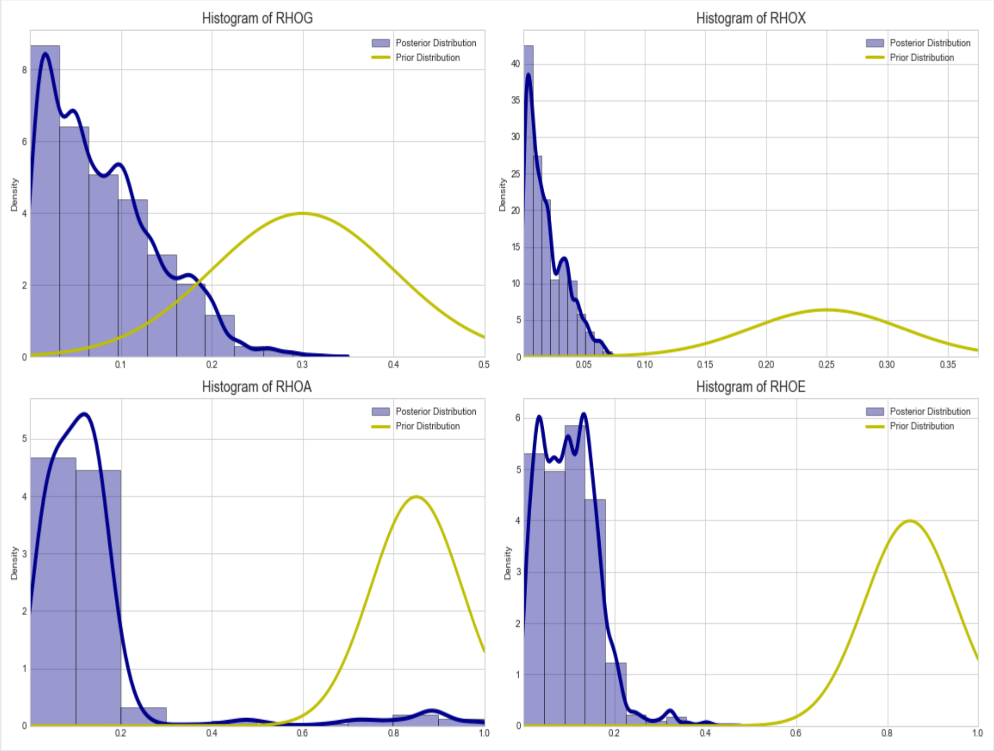
- *beta*, 0.25, 0, 10, normal_pdf, 0.25, 0.01
- *lmbda*, 0.25, 0, 1., normal_pdf, 0.25, 0.01
- *phi*,   0.75, 0, 1., normal_pdf, 0.75, 0.01
- *theta*, 0.1, 0, 0.5, normal_pdf, 0.1, 0.01

....

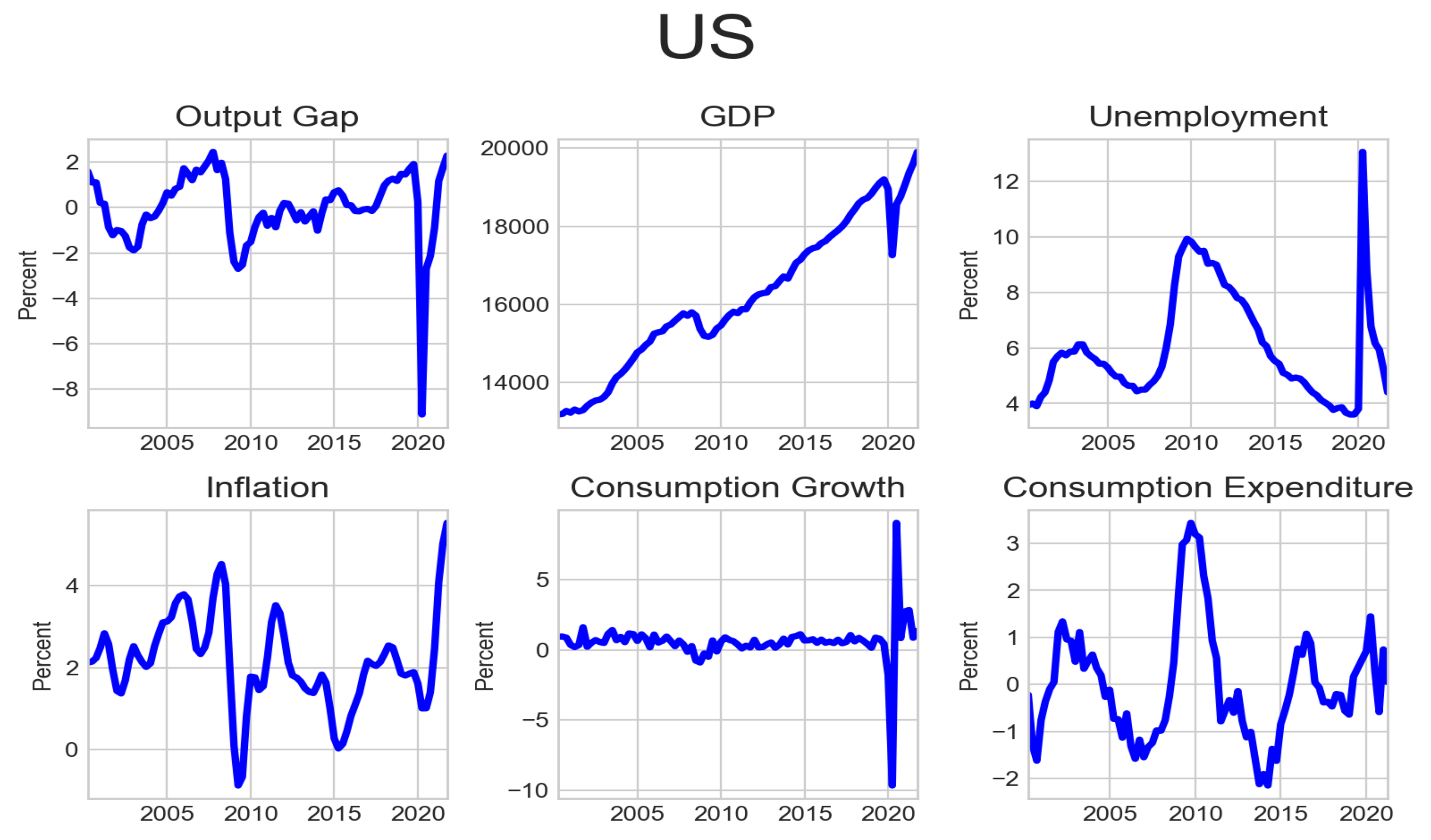
Peter’s Ireland Model For Technology Shocks

- The framework employs a **Bayesian** approach to maximize the likelihood function, evaluating the goodness of fit of the model to the data.
- It can sample model parameters using the **Markov Chain Monte Carlo** affine invariant ensemble sampler algorithm developed by **Jonathan Goodman**, in conjunction with the adaptive Metropolis-Hastings algorithms of **Paul Miles**. The former algorithm is particularly effective for sampling poorly scaled parameter distributions, while the latter offers an adaptive sampling mechanism. This algorithm utilizes adaptive Metropolis methods that incorporate delayed rejection to enhance the mixing of sample states.





Stylized Facts of the Covid-19 Epidemic



Source: World Economic Outlook and Haver Analytics databases

Eichenbaum-Rebelo-Trabandt Model

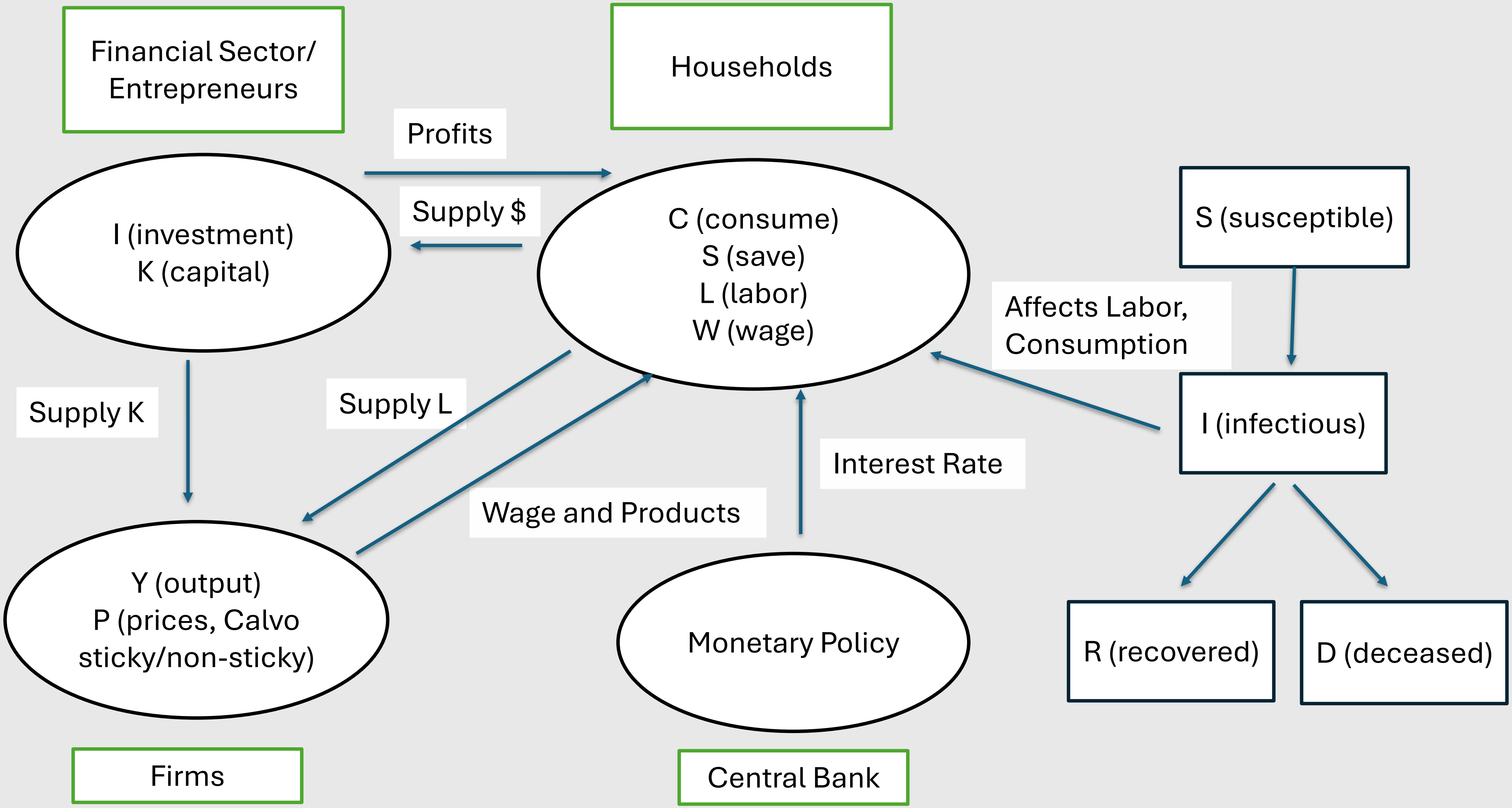
- The **ERT** model incorporates epidemiological concepts into the **DSGE** framework, which includes a Neoclassical model, a flexible-price model, and a New Keynesian model with sticky prices.

- Infection spreads through interactions between susceptible and infected individuals, as well as through economic activities such as work and shopping.

- The model comprises sixty-four equations that represent macroeconomic variables from both sticky-price and flexible-price economies.

- The macroeconomic variables of these two economies are interconnected through the Taylor rule equation governing the policy interest rate.

- The model is highly non-linear and is solved using a homotopy method, where parameters are adjusted incrementally. This approach is essential because the numerical method may diverge when applied with final parameters.


Financial Sector/ Entrepreneurs
Households
Profits
Supply $
I (investment)
K (capital)
C (consume)
S (save)
L (labor)
W (wage)
S (susceptible)
Affects Labor, Consumption
Supply K
Supply L
I (infectious)
Interest Rate
Wage and Products
Y (output)
P (prices, Calvo sticky/non-sticky)
Monetary Policy
R (recovered)
D (deceased)
Firms
Central Bank

Multi-Strain Epidemic Model for Covid-19

The framework integrates New Keynesian and epidemiological models for enhanced analysis

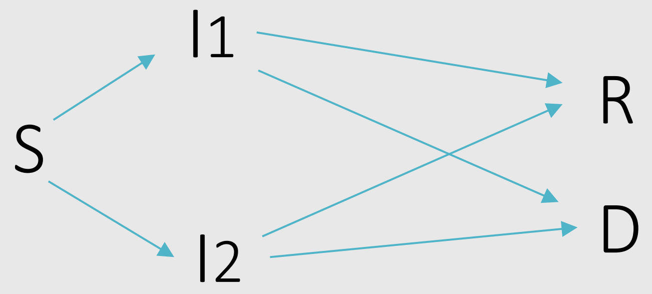
$$dS/dt = -(\beta_1 I_1 + \beta_2 I_2)S - vS$$

$$dI_1/dt = \beta_1 I_1 S - (\mu + \gamma_1)I_1$$

$$dI_2/dt = \beta_2 I_2 S - (\mu + \gamma_2)I_2$$

$$dR/dt = \mu(I_1 + I_2) + vS$$

$$dD/dt = \gamma_1 I_1 + \gamma_2 I_2$$



S – stock of susceptible, I_1, I_2 – flow of infected, R – total recovered, D – total deaths, v – vaccination rate.

ERT Model

- Households aim to maximize their utility function while adhering to budget constraints,

$$U = \sum_{t=0}^{\infty} \beta^t \left\{ S_t \left[\log(c_t^S) - \frac{\theta}{2}(n_t^S)^2 \right] + I_t \left[\log(c_t^I) - \frac{\theta}{2}(n_t^I)^2 \right] + R_t \left[\log(c_t^R) - \frac{\theta}{2}(n_t^R)^2 \right] \right\}$$

- Firms utilize Cobb-Douglas technology to produce intermediate goods and Dixit-Stiglitz aggregator for final goods.

- Firms maximize profits subject to Calvo style price-setting frictions,

$$\max_{P_t} \sum_{j=0}^{\infty} (\xi\beta)^j \left(P_t Y_{i,t+j} - P_{t+j}\, mc_{t+j} Y_{i,t+j} \right)$$

- Authorities utilize the Taylor rule to establish bond interest rate,

$$R_t^b = r_{ss} + \theta_\pi \, log\left(\frac{\pi_t}{\pi_{ss}}\right) + \theta_x \, log\left(\frac{y_t}{y_t^f}\right)$$

Calibration

Three channels of Infection transmission: i) consumption activity, ii) work activity, iii) interaction of susceptible and infected.

$$T_t = \pi_1 (S_t c_t^S)(I_t c_t^I) + \pi_2 (S_t n_t^S)(I_t n_t^I) + \pi_3 (S_t I_t)$$

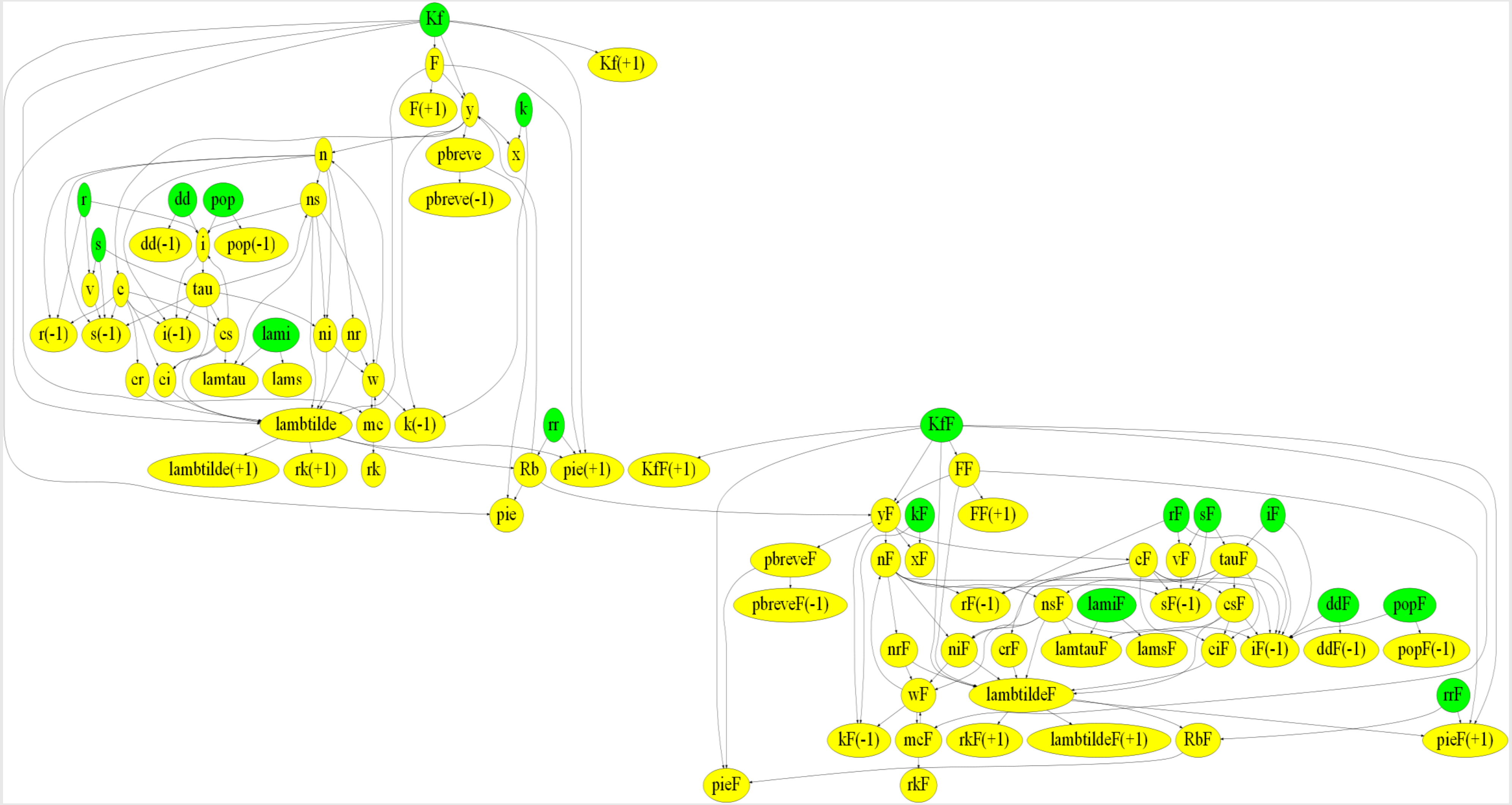
Parameters π_1, π_2, π_3 are calculated by applying a constrained minimization method to

$$\frac{\pi_1 c_{ss}^2}{\pi_1 c_{ss}^2 + \pi_2 n_{ss}^2 + \pi_3} = \frac{\pi_2 n_{ss}^2}{\pi_1 c_{ss}^2 + \pi_2 n_{ss}^2 + \pi_3} = 1/6 \qquad \text{s.t.} \qquad \pi_3 \geq \mu + \gamma$$

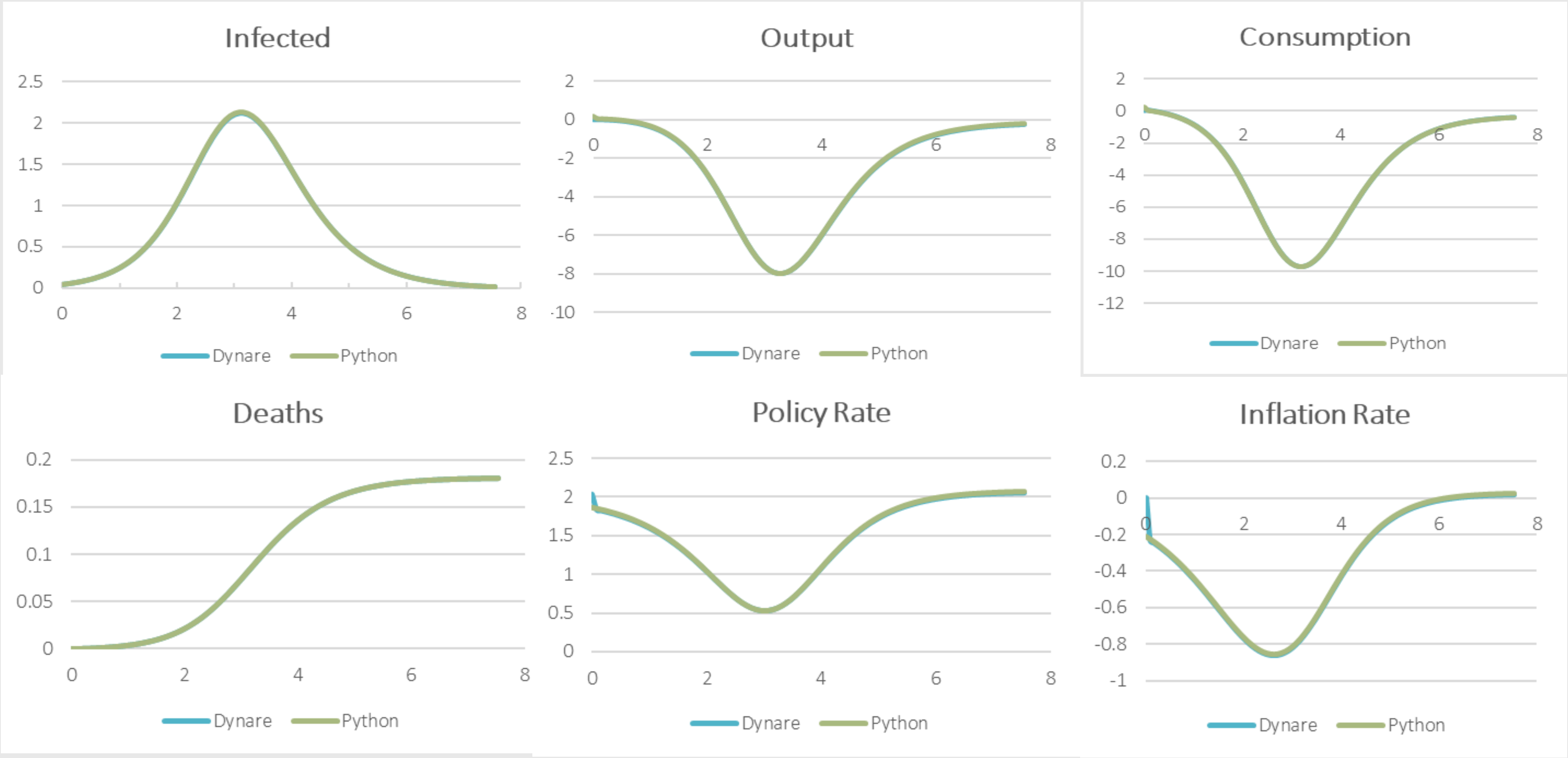
$$R_\infty + D_\infty = 1 - I_\infty = 0.4$$

The calibrated parameters are: $\pi_1 = 1.5\ 10^{-7}$, $\pi_2 = 9.5\ 10^{-5}$, $\pi_3 = 0.5$.

Graphical Representation of ERT Model

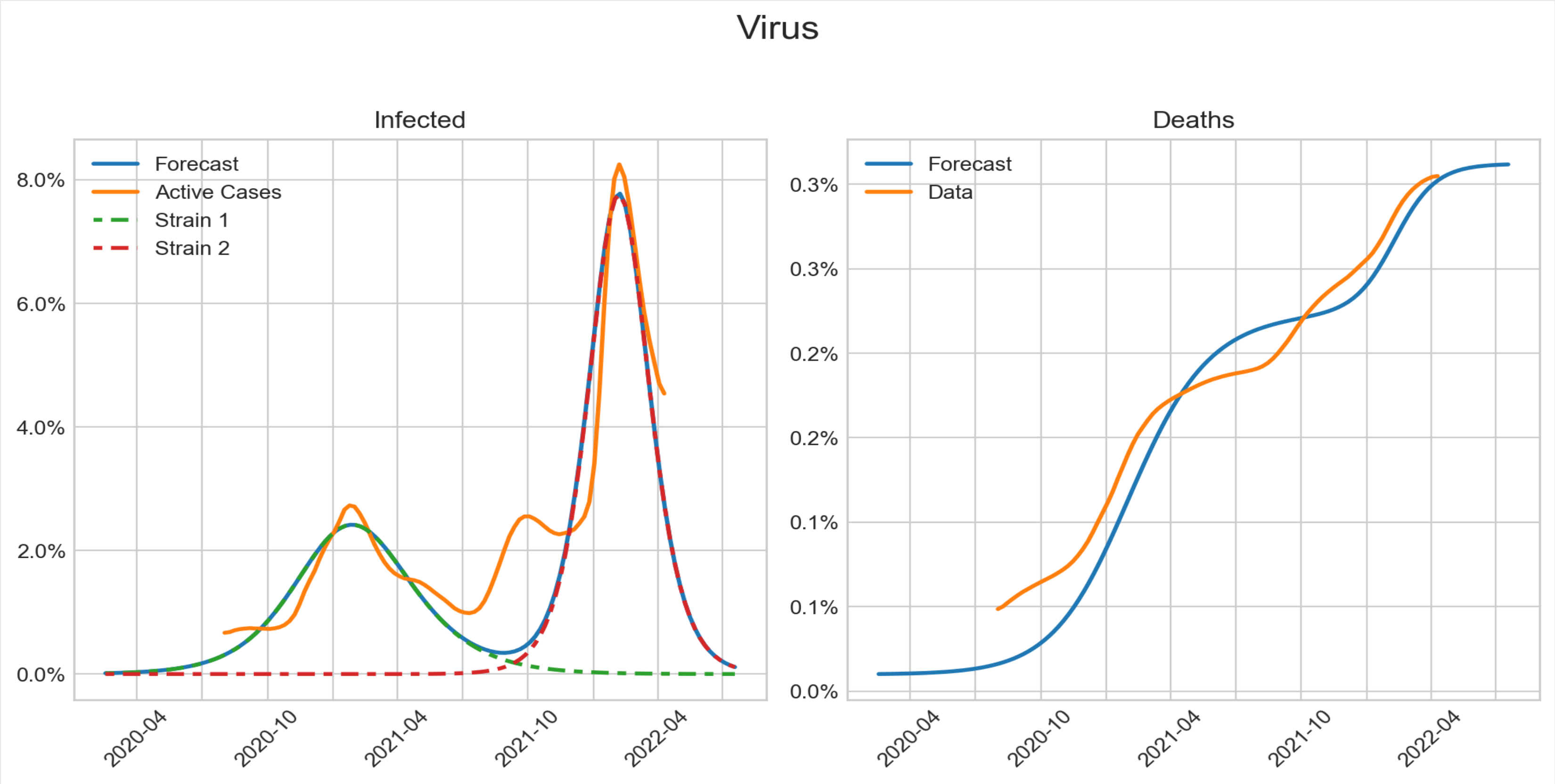


Dynare Versus Python (One-Strain Virus)



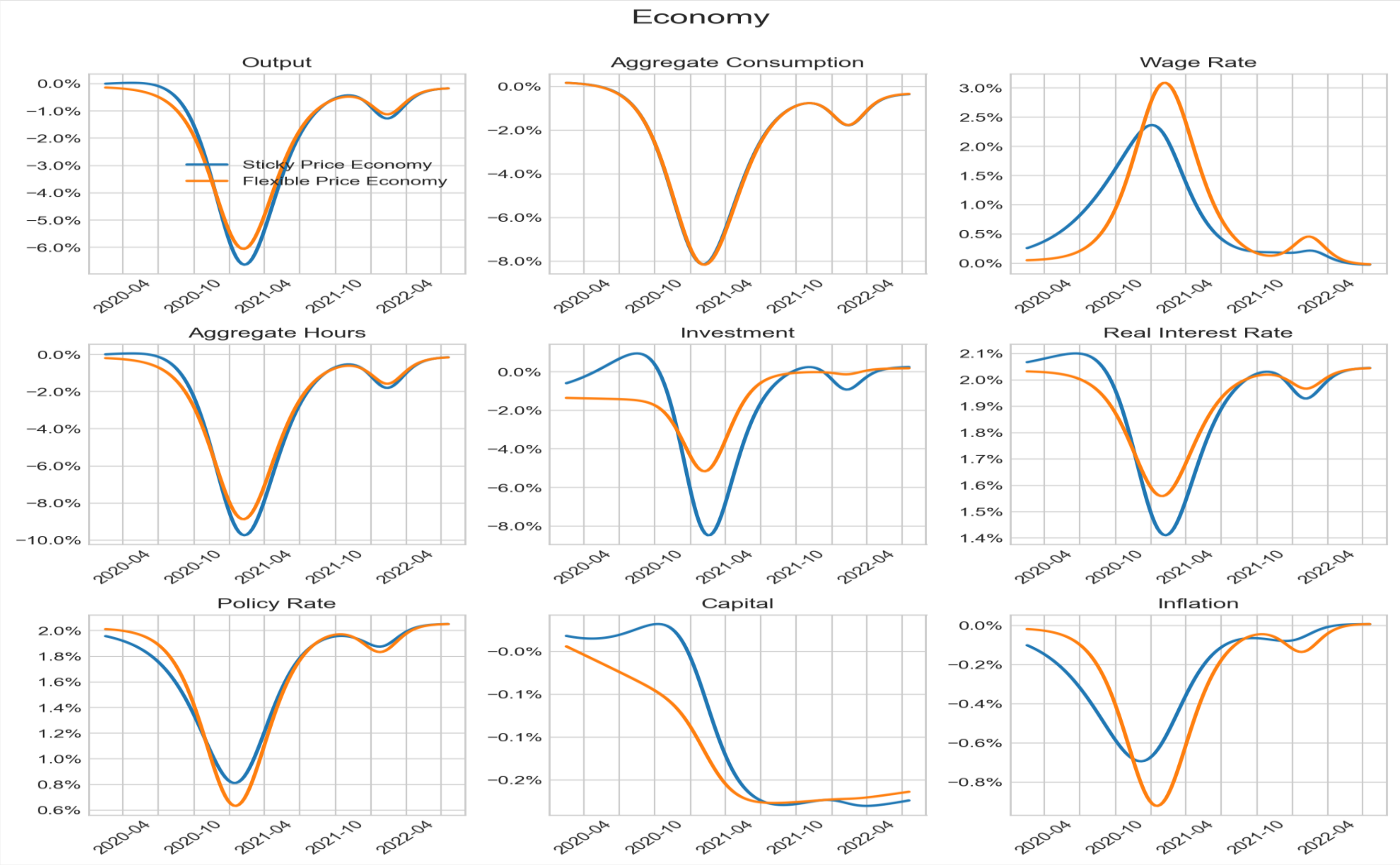
Comparison of forecasts from the DYNARE model and the Framework ERT model. In accordance with the ERT paper, we utilized an unconstrained optimization algorithm to calibrate the parameters, π_1, π_2, π_3 .

Two-Strain Virus

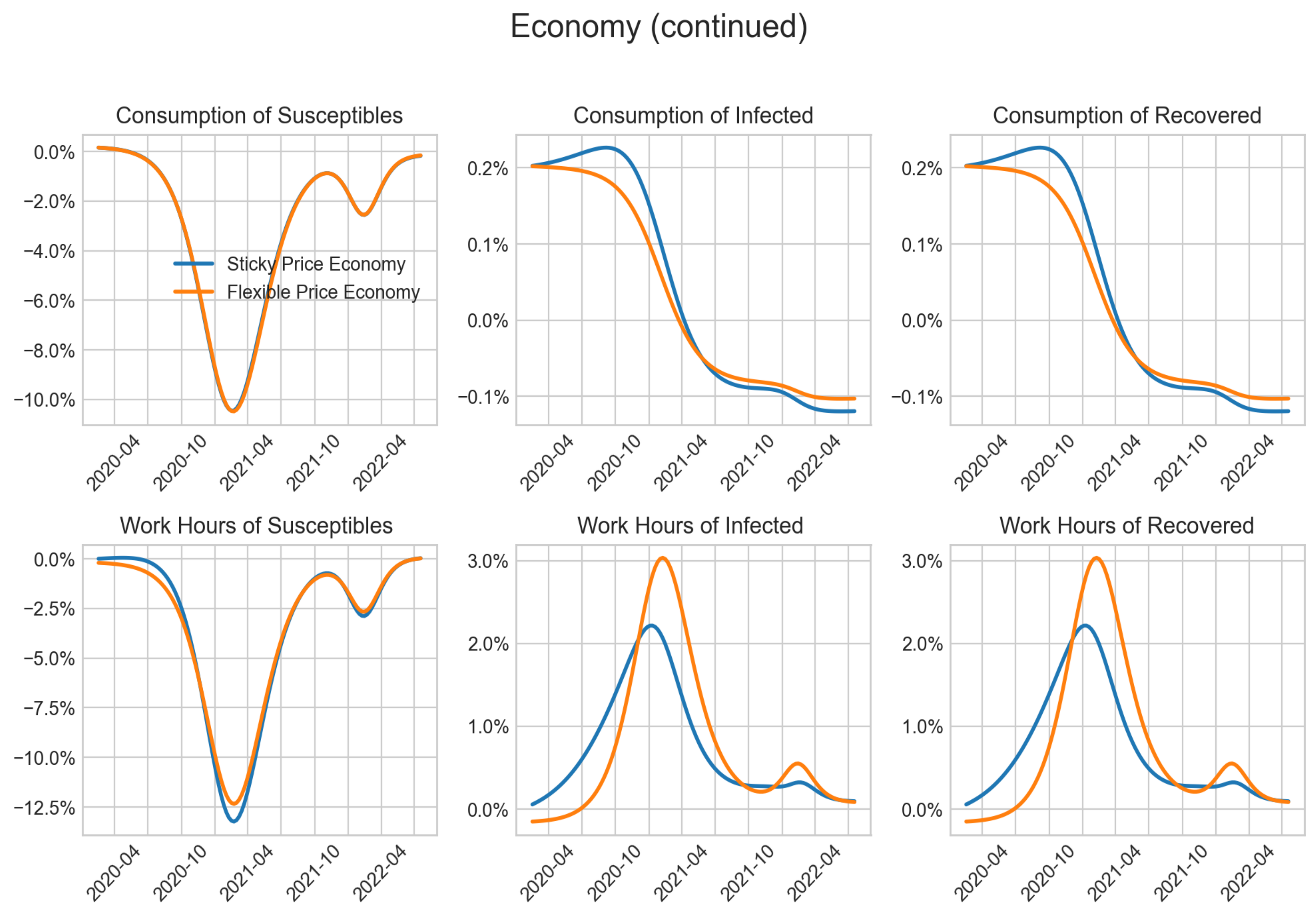


At time zero, the infection rate is 0.05%. The solid blue lines show model predictions, and the dashed orange lines show US data for Delta and Omicron viruses. Source: COVID-19 Cases in the United States, https://www.worldometers.info/coronavirus/country/us/#graph-cases-daily

Two-Strain Virus

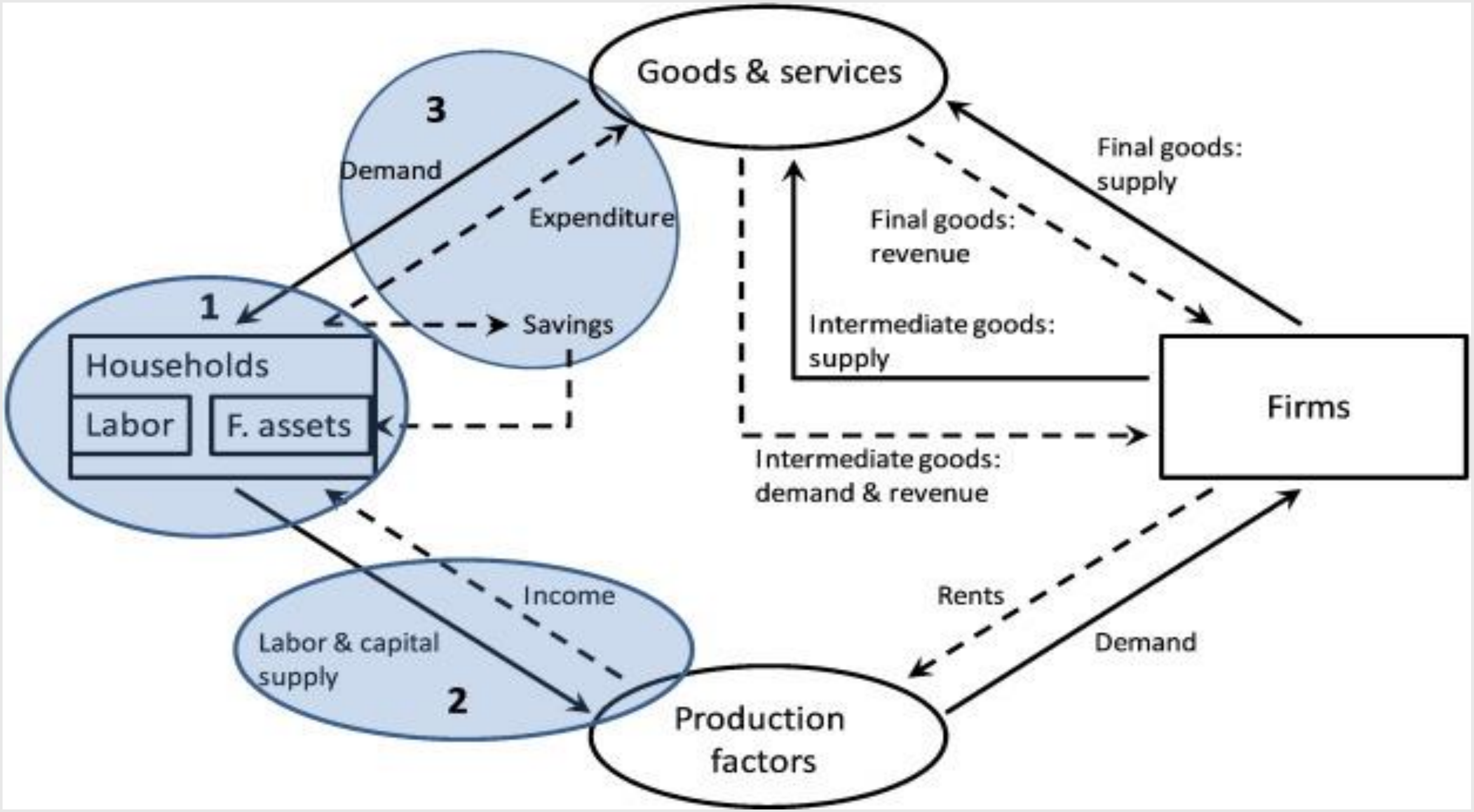


We assumed that the government containment measures were more lenient during the second strain of virus compared to the first one, i.e. the total infected population in equations is: $I_t = I_{1,t} + \delta\, I_{2,t}$, where $\delta = 0.05$ is the attenuation factor.


Economy (continued)
Consumption of Susceptibles
Consumption of Infected
Consumption of Recovered
Work Hours of Susceptibles
Work Hours of Infected
Work Hours of Recovered
Sticky Price Economy
Flexible Price Economy
0.0%
−2.0%
−4.0%
−6.0%
−8.0%
−10.0%
0.2%
0.1%
−0.1%
−2.5%
−5.0%
−7.5%
−12.5%
3.0%
2.0%
1.0%
2020-04
2020-10
2021-04
2021-10
2022-04

Work in Progress: Computable General Equilibrium (CGE) Models

Armington Model Structure



Armington Model

Composite commodity

$$Q_{j,r} = Q_{0,j,r} \left({P_{0,j,r}} \middle/ {P_{j,r}} \right)^{\theta}$$

Price index

$$P_{j,s} = \left[\sum_{r} \left(\tau_{j,r,s}\, c_{j,r} \right)^{1-\sigma_j} \right]^{1/1-\sigma_j}$$

Market clearance condition

$$Y_{j,r} = \sum_{s} \tau_{j,r,s}\, Q_{j,r} \left(\frac{P_{j,s}}{\tau_{j,r,s}\, c_{j,r}} \right)^{\sigma_j}$$

Input supply

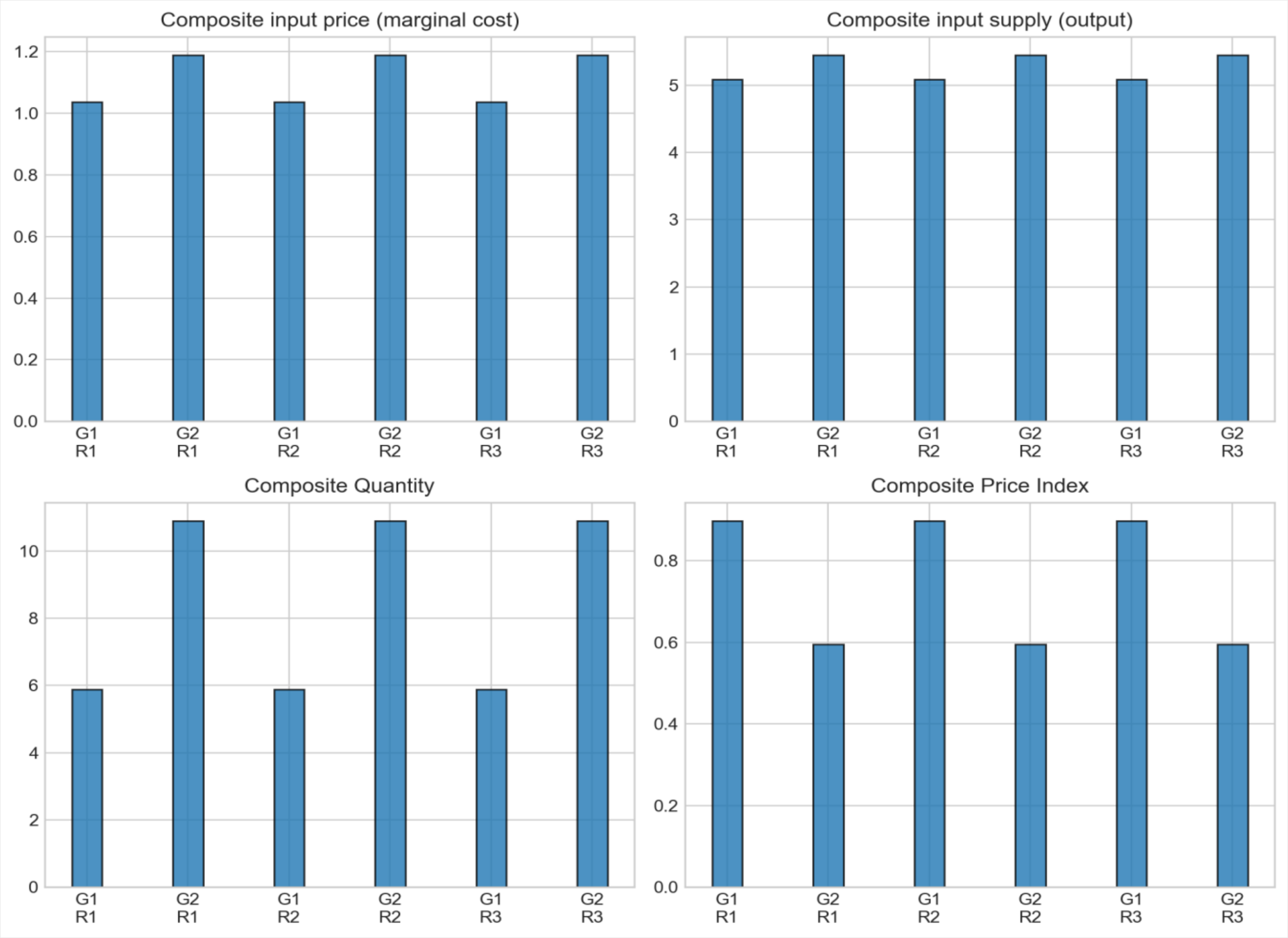
$$Y_{j,r} = Y_{0,j,r} \left(\frac{c_j}{c_{0,j,t}} \right)^{\mu}$$

Armington Model

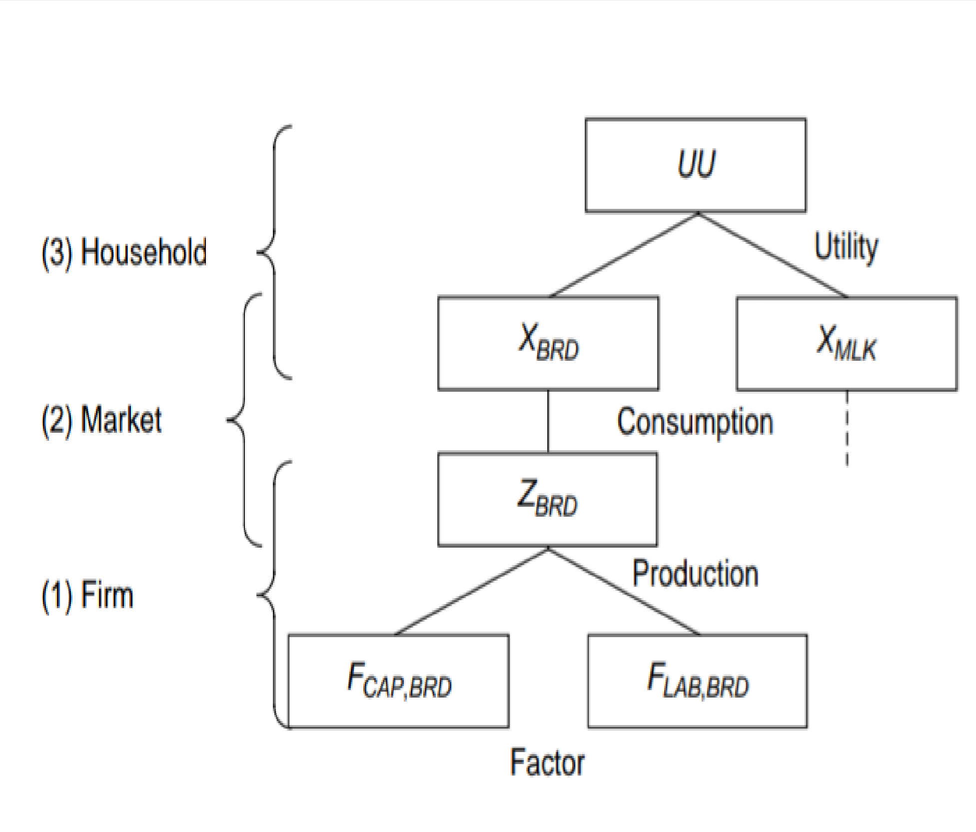
```
sets:
  regions r:   [R1, R2, R3]
  goods j:    [G1,G2]
  regions alias s: r
symbols:
  variables:  [Q(j)(r),P(j)(r),c(j)(r),Y(j)(r)]
  parameters: [sig(j),eta,mu,Q0(j)(r),P0(j)(r),Y0(j)(r),c0(j)(r),tau(j)(r)(s),vx0(j)(r)(s),zeta(j)(r)(s)]
equations:
# Eq.1 Aggregate demand
 - DEM(j)(r):    Q(j)(r) - Q0(j)(r) * (P0(j)(r) / P(j)(r))**eta
# Eq.2 Armington unit cost function
 - ARM(j)(s):   P(j)(s) -  sum(r, zeta(j)(r)(s)**sig(j) * (tau(j)(r)(s)*c(j)(r))**(1-sig(j)))**(1/(1-sig(j)))
# Eq.3 Market clearance
 - MKT(j)(r):    Y(j)(r) - sum(s, tau(j)(r)(s)*Q(j)(s)* (zeta(j)(r)(s)*P(j)(s)/(tau(j)(r)(s)*c(j)(r)))**sig(j))
# Eq.4 Input supply (output)
 - SUP(j)(r):    Y(j)(r) - Y0(j)(r) * (c(j)(r)/c0(j)(r)) ** mu
calibration:
   # Parameters
   sig_G1        : 3
   sig_G2        : 2
   P(j)(r)       : 1
constraints:
  # Positive Variables
  - Q(j)(r) >= 5.1
  - P(j)(r) >= 0
  # Positive LHS of equations
  - DEM(j)(r) >= 0
  - ARM(j)(s) >= 0
Model: [DEM.Q, ARM.P, MKT.c, SUP.Y]
Solver: 'CONSTRAINED_OPTIMIZATION'   # 'MCP', 'ROOT'
```

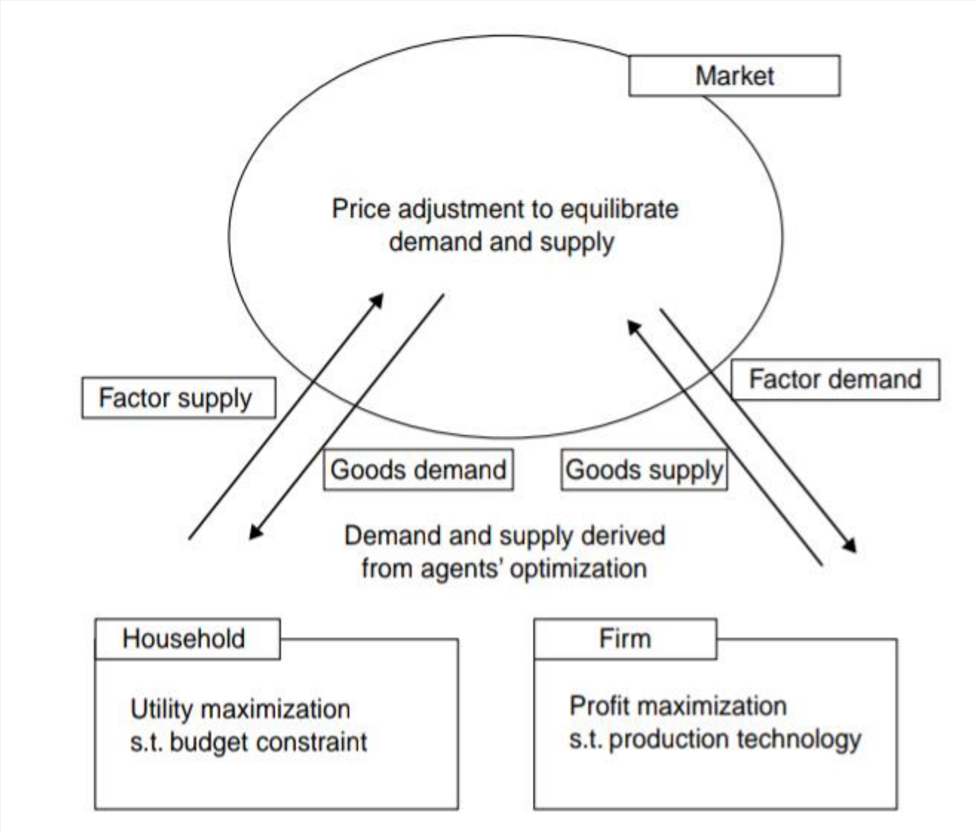
From Ch.23 of "Computing General Equilibrium Theories of Monopolistic Competition and Heterogeneous Firms" by Edward J. Balistreri, Thomas F. Rutherford

Framework Can be Used for CGE Modeling



Simple CGE Model Structure



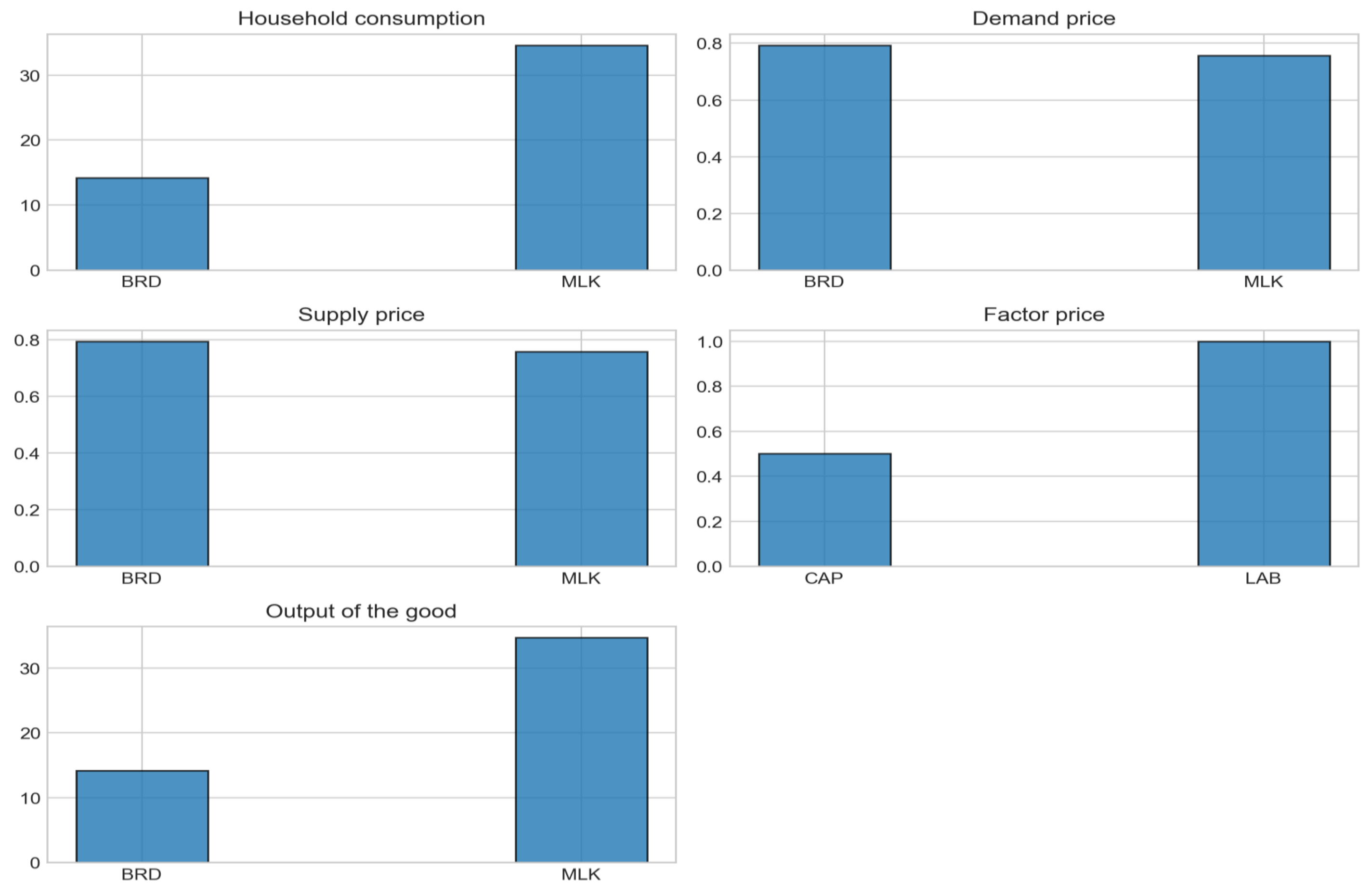


Simple CGE Model

```
sets:
  SAM_entry u: ['BRD', 'MLK', 'CAP', 'LAB', 'HOH']
  goods  i: ['BRD', 'MLK']
  factor h: ['CAP', 'LAB']
  goods alias j: I
symbols:
  variables:  [X(i),F(h)(j),Z(j),px(i),pz(i),pf(h)]
  parameters: [alpha(i),beta(h)(j),b(j),SAM(u)(v),FF(h)]
equations:
   - eqX(i)    : X(i)    - alpha(i)*sum(h, pf(h)*FF(h))/px(i)
   - eqpz(j)   : Z(j)    - b(j)*prod(h, F(h)(j)**beta(h)(j))
   - eqF(h)(j) : F(h)(j) - beta(h)(j)*pz(j)*Z(j)/pf(h)
   - eqpx(i)   : X(i)    - Z(i)
   - eqpf(h)   : FF(h)   - sum(j, F(h)(j))
   - eqZ(i)    : px(i)   - pz(i)
calibration:  # Social Accounting Matrix
#               BRD   MLK   CAP   LAB   HOH
 SAM(u)(v): [[0,    0,      0,       0,       15],      # BRD
...
                  [0,     0,    25,     25,     0]]        # HOH
objective_function:
   - prod(i, X(i)**alpha(i))
constraints:
   - X(i)    > 0.0
   - pf_LAB  = 1.0 # fixed price numeraire
```

From Ch.5. of "Handbook of Computable General Equilibrium Modeling" by Hosoe N., Gasawa K., and Hashimoto

Results



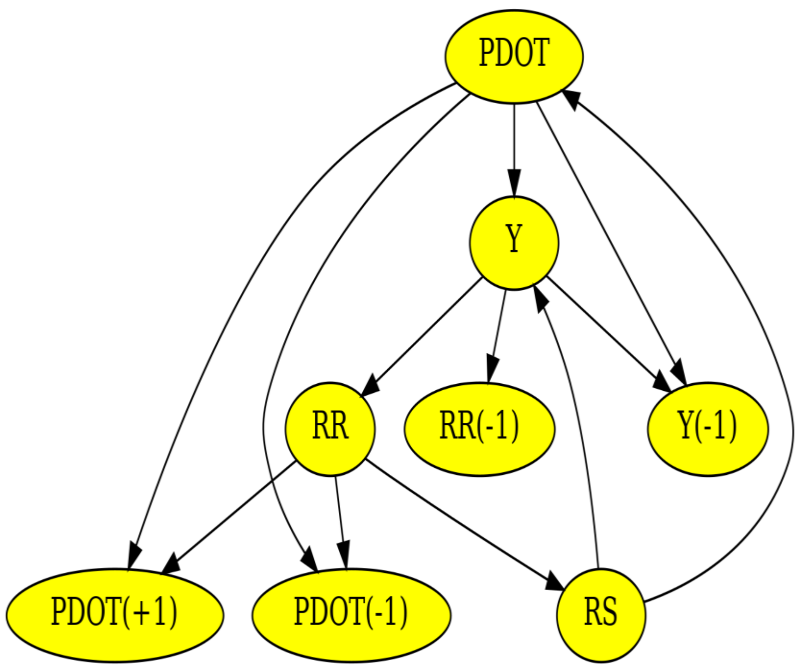
Household consumption
BRD
MLK
Demand price
BRD
MLK
Supply price
BRD
MLK
Factor price
CAP
LAB
Output of the good
BRD
MLK

Conclusions

- We have developed a powerful, user-friendly framework for macroeconomic modeling in **Python**.
- This framework can simulate a wide range of models, including **DSGE Models** such as **New Keynesian Models** with frictions, **Overlapping Generations Models**, **Real Business Cycle Models**, and **Computable General Equilibrium Models**.
- We applied the **Eichenbaum-Rebelo-Trabandt** (2020) model, which integrates **Neoclassical** and **New Keynesian** approaches with epidemiological concepts.
- We demonstrate that this model can be effectively implemented in the new Python framework, providing consistent forecasts of the economic impact of COVID-19.
- Lastly, we illustrated that the platform can be utilized for modeling **Computable General Equilibrium** Models.


Q
&
A

Appendices



Model File

Generated by Python Framework

December 1, 2024

1 Model Information

name: *Monetary policy model example*
file: */home/alexei/work/Framework/snowdrop/models/TOY/JLMP98.yaml*

1.1 Endogenous Variables Values

PDOT = 0.0, RR = 0.0, RS = 0.0, Y = 0.0

1.2 Parameters

g = 0.05, p_pdot1 = 0.41, p_pdot2 = 0.20, p_pdot3 = 0.28, p_rs1 = 3.00, p_y1 = 0.30, p_y2 = 0.10, p_y3 = 0.32

1.3 Shocks

ey

1.4 Equations

1 : PDOT = p_pdot1*PDOT(+1) + (1-p_pdot1)*PDOT(-1) + p_pdot2*(g^2/(g-Y) - g) + p_pdot3*(g^2/(g-Y(-1)) - g)

2 : RR = RS - p_pdot1*PDOT(+1) - (1-p_pdot1)*PDOT(-1)

3 : RS = p_rs1*PDOT + Y + ers

4 : Y = p_y1*Y(-1) - p_y2*RR - p_y3*RR(-1) + ey

1.5 Legend

PDOT – Inflation
PDOT(+1) – Lead of Inflation
PDOT(-1) – Lag of Inflation

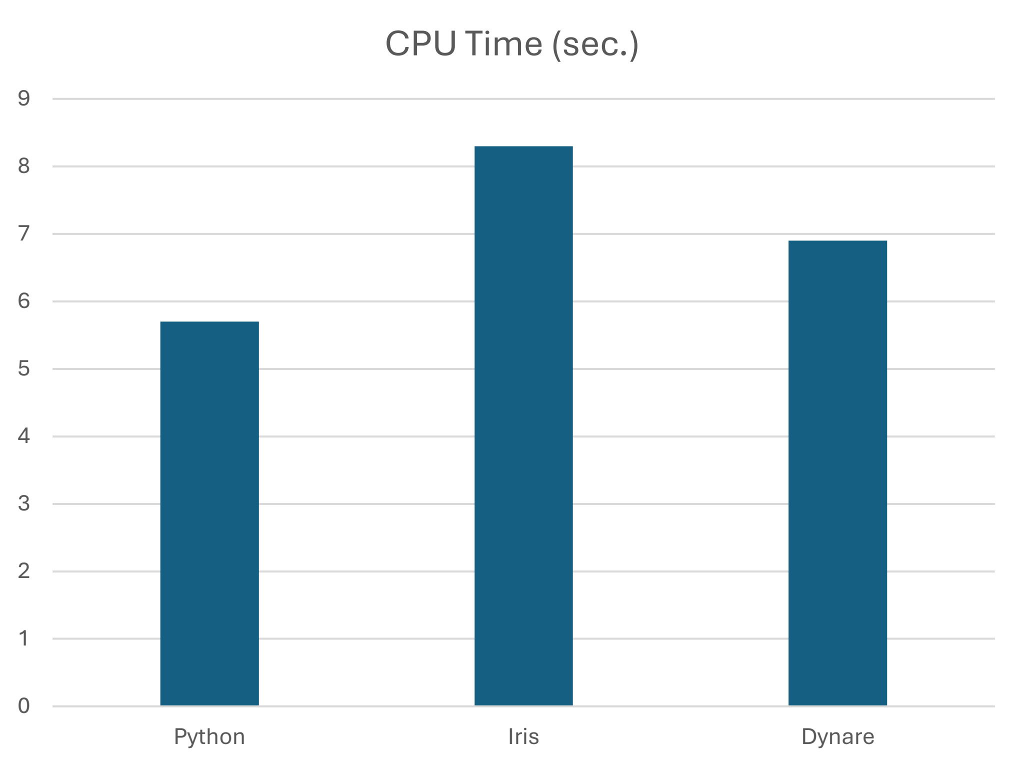
Details of Package Structure

- The “*supplements/models*” subfolder contains examples of several macroeconomic models presented.
- The *DIGNAR* and *DIGNAD* subfolders contain Python code that replicates the MATLAB implementation of these two macroeconomic models.
- The “*Epidemic*” subfolder contains code to forecast the effects of COVID-19 on the economy.
- The user guide manual is available in the “*supplements/docs*” folder.
- API documentation can be accessed by opening the “*supplements/api_docs/_build/html/index.html*” file in a web browser.

ERT Model Parameters

Notation	Economic Interpretation	Parameter Value
β	Weekly discount factor	$0.98^{1/52} = 0.9996$
θ	Working hours multiplier in the household utility function	0.19
θ_π	Taylor rule coefficient of inflation	1.5
θ_x	Taylor rule coefficient of output gap	0.5/52
ξ	Calvo price stickiness (weekly)	0.98
$\hat{\gamma}$	Price dispersion parameter	1.35
δ	Capital depreciation rate (weekly)	0.06/52
α	Marginal product of labor	2/3
γ	Weekly probability of dying	0.25%
μ	Weekly probability of recovering	${}^{7}/_{14} - \gamma = 49.8\%$
π_{ss}	Steady-state inflation	1
r_{ss}	Steady-state nominal interest rate	${}^{1}/_{\beta} = 1.0004$
y_{ss}	Weekly average income	$58{,}000/52 = 1{,}115\,\frac{5}{13}$
n_{ss}	Steady-state number of work hours per week	28
A	Cobb-Douglass production function multiplier	$\frac{\beta(1-\alpha)}{1-\beta(1-\delta)}\left(\frac{y_{ss}}{n_{ss}}\right)^2 = 2.148$

The CPU Time Required to Run the Ghana Small Open Economy Model is Comparable, However, the Memory Footprint is Notably Smaller



Framework Settings

Kalman Filter and Smoother Algorithms

Filter Algorithms	Description
Diffuse	Diffuse Kalman filter (multivariate and univariate) with missing observations
Durbin_Koopman	Non-diffusive variant of Durbin-Koopman Kalman filter
Non_Diffuse_Filter	Non diffuse Kalman filter (multivariate and univariate) with missing observations
Unscented	Unscented Kalman filter
Particle	Particle filter

Smoother Algorithms	Description
BrysonFrazier	Bryson and Frazier algorithm
Diffuse	Diffuse Kalman Smoother (multivariate and univariate) with missing observations
Durbin_Koopman	Non-diffuse variant of Kalman smoother (multivariate)

Initial Conditions for Filtered Variables and Their Error Variance-Covariance Matrix

Variables	Description
StartingValues	Model starting values of variables are used
SteadyState	Steady-state values are used as starting values
History	Starting values are read from a history file

Error Variance-Covariance Matrix	Description
Diffuse	Diffuse prior for covariance matrices (Pinf and Pstar)
StartingValues	Starting values for covariance matrices (Pinf with diagonal values of 1.E6 on diagonal and Pstar=T*Q*T')
Equilibrium	Equilibrium error covariance matrices obtained by discrete Lyapunov solver by using stable part of transition and shock matrices
Asymptotic	Asymptotic values for error covariance matrices; it is obtained by solving time discrete Riccati equation

Markov Chain Monte Carlo Sampling Algorithms

Sampling Algorithms	Description
Emcee	Affine Invariant Markov Chain Monte Carlo Ensemble sampler
Pymcmcstat	Adaptive Metropolis based sampling techniques include
Pymcmcstat_mh	Metropolis-Hastings (MH): Primary sampling method
Pymcmcstat_am	Adaptive-Metropolis (AM): Adapts covariance matrix at specified intervals.
Pymcmcstat_dr	Delayed-Rejection (DR): Delays rejection by sampling from a narrower distribution. Capable of n-stage delayed rejection.
Pymcmcstat_dram	Delayed Rejection Adaptive Metropolis (DRAM): DR + AM
Pymc3	Markov Chain Monte Carlo (MCMC) and variational inference (VI) algorithms
Particle_pmmh	Particle Marginal Metropolis Hastings sampling
Particle_smc	Particle Sequential Quasi
Particle_gibbs	Particle Generic Gibbs sampling