Minimum working example for programming package

Carolin Pflueger
University of Chicago, Harris School of Public Policy, NBER, and CEPR

December 2023

If you have any questions or comments regarding this package, we encourage you to reach us by leaving a comment in the "Issues" section of the corresponding **GitHub repository**. Alternatively, you can also contact us directly via the following email addresses: cpflueger@uchicago.edu/l.yepezsa95@gmail.com.

1 Replication File Structure

This document describes the replication file for the minimum working example base on the paper Why Does the Fed Move Markets so Much? A Model of Monetary Policy and Time-Varying Risk Aversion (JFE, 2022).[1] Specifically, in this code, we want to replicate some results from the original calibration of the model presented in Tables 3 and 4, and Figure 3 of the paper.[2] The results of this replication are in Figures 12 and 13. And also get some key variables that we can use to get some stock and bond moments. Therefore, below is a step-by-step explanation of all parts of the replication file.

1. **Define classes :** In this part of the code, as shown in Figure 1, we define the classes macro1, num1 and asset.

    - **macro1 class :** We define this class as a macro_dyn class, which contains all parameters and methods related to the macro dynamics of the model.

[1] It is necessary to mention that the moments will be exactly the same as in the paper due to both exercises use one simulation of 10000.

[2] This model calibration is the same as in the paper.

- **num1 class :** We define this class as a num_set class, which contains all the settings for the numerical solution of asset prices. And it also calculates the weights used for Gauss-Legendre integration and the grid for the state vector, which we use when solving asset prices.
- **asset class :** We define this class as an asset_p class, which contains all parameters and methods related to asset prices and their properties.

Figure 1: Defining classes

```
%% Define the classes macro1, num1 and asset
macro1  =   macro_dyn;
num1    =   num_set;
asset   =   asset_p;
```

2. **Define parameters for speed and numerical solution :** In this part of the code, as you can see in Figure 2, we define some parameters of the classes asset and num1. Which are necessary to improve the speed of the code and implement the numerical solution, respectively. In addition, we define a random number generator to ensure the exact replicability of the results.

   - **num1 class :** For this class, we use the function "parameters" to define the most important parameters that are necessary for the numerical solution of asset prices. For example, the length of model simulation and the number of gridpoints along each dimension of the state vector $\tilde{Z}$.[3]
   - **asset class :** For this class, we define the dummy variable "risk_neutral_run" that takes the value of 1 when we choose to calculate the risk-neutral part of the code.[4] Also we define the dummy variable "FOMC_run" that takes the value of 1 when we choose to simulate the FOMC effects.

[3]The number of independent model simulations (num1.Nsim) is set equal to 1.

[4]We set asset.risk_neutral_run=0 to speed up the code when we don't need to compute risk-neutral asset prices.

Figure 2: Parameter settings for speed

```
%% Parameter settings that can speed up or slow down code
% Define a dummy variable to run the risk neutral part (=1 for risk neutral)
asset.risk_neutral_run  =   1;

% Define a dummy variable to simulate the FOMC effects (=1 for FOMC effects)
asset.FOMC_run          =   1;

% Parameter settings for numerical solution method, e.g. grid density,
% number of simulations etc.
num1 = num1.parameters;

% Fix random number generator if desired for exact replicability
rng('default');
```

3. **Define calibration :** In this part of the code, as you can see in Figure 3, we define some parameters of the classes macro1 and asset, which are the minimum necessary to run the code. Also, we use the function "update_params" of the macro1 class, which is mainly used to compute implied Euler equation coefficients.

   - **macro1 class :** For this class, we define the most important parameters that are related to the macro model:
     - θ_0: Persistence of the surplus consumption ratio.
     - θ_1: Dependence of the surplus consumption ratio on current output gap.
     - ϕ: Smoothing parameter for consumption.
     - γ: Parameter controlling utility curvature.
     - γ^x: Parameter controlling the reaction of the interest rate to the output gap.
     - γ^π: Parameter controlling the response of the interest rate to inflation relative to its target level.
     - g: Consumption growth rate.
     - $\bar{r}$: steady state real short-term interest rate at $x_t = 0$.
     - δ: Leverage parameter relating the dividend growth to consumption growth.
     - κ: Parameter controlling the sensitivity of inflation to the output gap.

- ρ^i: Parameter controlling the influence of past interest rates on current interest rates (MP equation).
- ρ^π: The parameters $(1-\rho^\pi)$ and ρ^π are, respectively, the coefficient on expected next period output gap and lagged output gap in the PC equation.
- sigma_vec: The variance of MP shock, in particular, is the calibrated value to get empirically relevant values.

- **asset class :** For this class, we define the vector "initialShockVec", which is needed for simulating IRFs of macro dynamics and asset prices to various IRFl shocks (std of FOMC date monetary policy surprise).

Figure 3: Calibration

```
%% Input parameters for New Keynesian model with FOMC surprises

% Monetary policy rule
macro1.gamma_x          =   0.5/4;          % MP Response to output (annualized percent)
macro1.gamma_pi         =   1.5;            % MP Response to inflation
macro1.rho_i            =   0.8;            % MP Persistence

% Preference parameters
macro1.theta0           =   0.9658;         % Peristence surplus consumption
macro1.theta1           =   -0.90;          % Backward looking habit
macro1.phi              =   0.9300;         % Consumption-output gap
macro1.gamma            =   2;              % Utility curvature
macro1.g                =   0.004725;       % Consumption growth
macro1.rf               =   0.00235;        % Risk-free rate

% Phillips curve
macro1.kappa            =   0.0062/4;       % Slope of the Phillips curve
macro1.rho_pi           =   0.8;            % Backward-Looking Coefficient

% Leverage parameter
macro1.delta            =   0.6666;

% Variance of quarterly MP shock in natural units.
macro1.sigma_vec(3)     =   8.7767e-06;

% Std of FOMC date monetary policy surprise in annualized percent
asset.initialShockVec  = [0 0 0.0652 0];

% Compute implied Euler equation coefficients and fill in other parameters
% such as the industry portfolio betas
macro1 = macro1.update_params;
```

4. **Solving the macro dynamics :** In this part of the code, as you can see in Figure 4, we use the function "ModelPQ82" of the macro1 class to solve the macro dynamics of the model using the method of generalized eigenvectors and select an equilibrium.

Figure 4: Macro-dynamics solution

```
%% Solve for macroeconomic dynamics of the form Y_t=PY_{t-1}+Qv_t

% Solves for the macro dynamics of the model
macro1 = macro1.ModelPQ82(num1);
```

5. **Prepare for asset price VFI :** In this part of the code, as you can see in Figure 5, we use the function "ScaledStateVector" of the macro1 class to compute the state vector $\tilde{Z} = A\hat{Y}$. And also, we update some specifications of the num1 class, using the function "update_num" of this class. This function is used to:

   - Define the number of burn observations.
   - Generate the grid for surplus consumption ratio $\hat{s}$.
   - Generate the grid for value function iteration.
   - Evaluate the sensitivity function at each of the points listed in $\hat{s}$.
   - Generate transition probabilities i.e. scaled Gauss-Legendre weights.
   - Generate the distribution of surplus consumption ratio $\hat{s}_{t+1}$ at current state vector $\left(\tilde{Z}_t, \hat{s}_t, x_{t-1}\right)$.

Figure 5: Preparation for asset prices solution

```
%% Prepare for asset price value function iteration

% Compute the rotated state vector
macro1  = macro1.ScaledStateVector;

% Numerical settings for value function iterations, some of which depend on
% rotated grid
num1 = num1.update_num(macro1);
```

6. **Solve risk-neutral part :** In this part of the code, as you can see in Figure 6, we solve and simulate risk-neutral asset prices, using the function "risk_neutral_ap" of the asset class. This function contains other functions such as "computeFn21", which implements the value function iteration to solve the asset prices. Another function is "SimulateMoments" which simulates macroeconomic dynamics and asset prices to obtain moments. It should be noted that the assessment of the risk-neutral part only occurs if the dummy variable "risk_neutral_run" is active.[5]

Figure 6: Output while solving and simulating risk-neutral asset prices (line 83)

```
%% Solve and simulate risk-neutral asset prices.

% Solve and simulate risk neutral asset prices. Only executed if
% macro1.risk_neutral_run ==1
disp("Solve and simulate asset prices")
asset = asset.risk_neutral_ap(macro1, num1);
```

7. **Solve risk premia part :** In this part of the code, as you can see in Figure 7, we solve and simulate asset prices with risk premia, using the functions "computeFn21" and "SimulateMoments" of the asset class mentioned above.

Figure 7: Output while solving and simulating full asset prices (lines 89 and 93)

```
%% Solve and simulate full asset prices.

% Implements the value function iteration and simulations
disp('Computing prices')
asset = asset.computeFn21(num1,macro1);

% Simulate path for macroeconomic dynamics and asset prices
disp('Simulate moments')
asset = asset.SimulateMoments(num1,macro1);
```

[5]The dummy variable "risk_neutral_run" is active when "risk_neutral_run==1". If "risk_neutral_run==0" risk-neutral asset prices are not calculated and any simulations of risk-neutral asset prices are not meaningful. In order to obtain meaningful simulated risk-neutral asset prices set "risk_neutral_run=1".

8. **Simulate structural IRFs :** In this part of the code, as you can see in Figure 8, we define structural shock for structural impulse response set parameters for impulse response functions. And then we use the function "simulateStructuralIRF" to obtain the simulated structural IRFs.

Figure 8: Simulation of impulse response functions

```
%% Structural Impulse responses. This computes the consumption moments in Table 3

% Macroeconomic impulse responses do not need simulation. Set num.Nsim=2
% and asset.testirf=1 for speed.
num1.Nsim = 2;
num1.Tirf = 150;
asset.testirf=1;

% Simulations to build impulse responses for asset prices
asset =  asset.simulateStructuralIRF(macro1,num1);

% Lag trough consumption response
lag_trough=find(asset.Irftemp.c==min(asset.Irftemp.c))-1;
```

9. **Reproduce moments :** In this part of the code, as you can see in Figures 9, 10 and 11, we reproduce the results of Tables 3 and 4, and Figure 3 of the paper. We reproduce the results of these tables using only the structures "stocks", "nominalBonds", "macroDynamics" and "AP_responses". Furthermore, it's important to mention that Figure 3, contains the model impulse responses to monetary policy shock and the results for all the variables are in log points.

Figure 9: Finding moments of the model

```
%% Table 3: Quarterly moments

% Collect simulated model moments
Table3      =   [asset.stocks.equityPremium, asset.stocks.vol, asset.stocks.sharpeRatio, asset.stocks.rhoDP, asset.stocks.coeffRegRetOnPD1y, asset.stocks.R2RegRetOnPD1y]';
Table3      =   [Table3; 0;  [asset.nominalBonds.meanLogYieldSpread, asset.nominalBonds.vol, asset.nominalBonds.coeffRegRetOnYS1y, asset.nominalBonds.R2RegRetOnYS1y]';...
                0;asset.macroDynamics.consGrowthVol;asset.macroDynamics.iChangeVol];
Table3      =   [Table3; 0; min(asset.Irftemp.c)/max(asset.Irftemp.i); lag_trough];

% Data moments
Table3Data  = [7.89;
               16.79;
               0.47;
               0.92;
               -0.38;
               0.23;
               0;
               1.87;
               9.35;
               2.69;
               0.14
               0;
               1.50;
               1.35;
               0;
               -0.7;
               4];


% Create Table 3 to compare data and simulated model moments
Table3 = [Table3, Table3Data];
Table3 = array2table(round(Table3,2));
Table3.Properties.RowNames = {'Equity Premium'...
'Equity Vol'...
'Equity SR'...
'AR(1) pd'...
'1 YR Excess Returns on pd'...
'1 YR Excess Returns on pd (R^2)'...
'-'...
'Yield Spread'...
'Return Vol.'...
'1-YR Excess REturns on Yield Spread'...
'1-YR Excess REturns on Yield Spread (R^2)'...
'--'...
'Std. Annual Cons. Growth'...
'Std Annual Change Fed Funds Rate'...
'---'...
'Trough Effect Consumption'...
'Lag Trough'
};
Table3.Properties.VariableNames = {'Model', 'Data'};
Table3
```

Figure 10: Stock market response to high-frequency monetary policy shocks

```
%% Table 4: Model Stock Returns on FOMC Dates

% Collect simulated model moments
Table4              =   [asset.AP_responses.bernankeKuttner(1), asset.AP_responses.bernankeKuttner_dummy(1,1), asset.AP_responses.bernankeKuttner(2),...
                         asset.AP_responses.bernankeKuttner_dummy(1,2), asset.AP_responses.bernankeKuttner(3), asset.AP_responses.bernankeKuttner_dummy(1,3);
                         0, asset.AP_responses.bernankeKuttner_dummy(2,1), 0, asset.AP_responses.bernankeKuttner_dummy(2,2), 0, asset.AP_responses.bernankeKuttner_dummy(2,3)];

% Create Table 4 to show simulated model moments
Table4              =   array2table(round(Table4,2));
Table4.Properties.RowNames = {
 'FF_Shock'...
 'FF_Shock_x_(FF_Shock>0)'
 };
Table4.Properties.VariableNames = {'Overall'...
 'Overall_'...
 'RN'...
 'RN_'...
 'RP'...
 'RP_'
 };
Table4
```

Figure 11: Impulse response to monetary policy shocks

```
%% Figure 3: Asset price impulse responses with simulations
% Set the parameters for simulations as the number of simulations and
% the length of IRFs
num1.Nsim = 5000;
num1.Tirf = 150;
asset.testirf=0;

% Simulations to build impulse responses for asset prices
asset =  asset.simulateStructuralIRF(macro1,num1);

% Save third structural IRF
asset.Irf3=asset.Irftemp;

% Plotting function for Figure 3
plot_StructuralIRF_STMP(asset);
```

10. **Compare Outputs :** In Figures 12 and 13 we present the results of the minimum working example code. As can be seen, the results of the minimum working example code are very similar to those of the paper, which ensures its correct implementation. It's necessary to mention that these results are for the case of asset prices with risk premia.

11. **Replicate Outputs with simulated series :** In this exercise, we create some simulated series for the risky version, saved in asset class and called "asset.simulated".[6] Which will help us to replicate the results that we show in Figure 12. So, in Figures 14 and 15, we have the replication of the same variables that we have in tables 3 and 4, respectively. Thus in Figure 16 we have the results of this replication, which are the same as in Figure 12.

[6]It should be noted that in the case of the risk-neutral asset prices, we have the same series but saved in the structure "simulated_rn" of the asset class.

[7]The impulse response for equity returns in Figure 3 in the original paper is slightly different due to a computational error in the original figure. The results are, however, substantively unchanged.

Figure 12: Minimum working example results - Tables 3 and 4

Table3 =

17×2 table

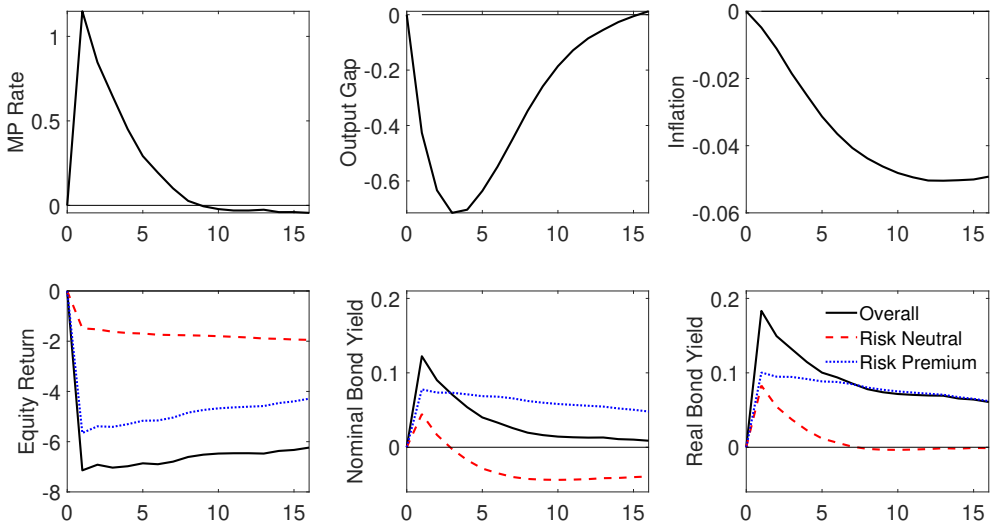
	Model	Data
Equity Premium	7.29	7.89
Equity Vol	14.87	16.79
Equity SR	0.49	0.47
AR(1) pd	0.93	0.92
1 YR Excess Returns on pd	-0.34	-0.38
1 YR Excess Returns on pd (R^2)	0.07	0.23
-	0	0
Yield Spread	0.94	1.87
Return Vol.	2.57	9.35
1-YR Excess REturns on Yield Spread	-0.18	2.69
1-YR Excess REturns on Yield Spread (R^2)	0.01	0.14
--	0	0
Std. Annual Cons. Growth	1.55	1.5
Std Annual Change Fed Funds Rate	2.13	1.35
---	0	0
Trough Effect Consumption	-0.71	-0.7
Lag Trough	4	4

Table4 =

2×6 table

	Overall	Overall_	RN	RN_	RP	RP_
FF_Shock	-6.37	-6.31	-1.23	-1.22	-5.13	-5.08
FF_Shock_x_(FF_Shock>0)	0	-0.07	0	-0.02	0	-0.06

Figure 13: Minimum working example results - Figure 3[7]



12. **Additional results :** As we mentioned above, we can change the value of some parameters. For example, in Figures 17 and 18 we can see the results when we change the number of grid points along each dimension of $\tilde{Z}$ (num1.N) from 2 to 3. Also in Figures 19 and 20, we can see the results when we change the length of the grid for surplus consumption ratio (num1.sfast) from 6 to 8 (which uses a bigger grid). It's necessary to mention that the results shown in Figures 17 - 20 are very similar to the original results of the paper and the minimum working example.[8]

    Finally, we can compare the average speed of the minimum working example's original code with versions where we used num1.N=3 and num1.sfast=8.[9] So the average speed of the minimum working example's original code is 326.36 seconds, which is faster than the versions with num1.N=3 and num1.sfast=8 , which have an average speed of 636.30 and 2787.50 seconds, respectively.[10]

13. **Switching off Test Mode (Careful - Slows down Code) :** It's important to mention that if we want to run the code faster, we can turn off the test mode with background noise when we simulate the impulse responses. To do this, we need to set "asset.testirf=1", also we can increase the speed of the code by reducing the number of simulations. For example, we can change the number of simulations from 5000 to 100, and even then we would obtain very similar results for the IRFs as you can see in Figure 21 (these changes can be made in lines 274 and 276 of the code).

[8] It's not a good choice to use the values of 4 and 5 for num1.sfast as they give meaningless results.

[9] To calculate the average speed of each version we use 100 iterations.

[10] The processor of the PC where I did these calculations is: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz. And has 6 cores and 12 logical processors. And I used MATLAB R2023a.

Figure 14: Replication of Table 3 with simulated series

```
%% This section shows how to replicate Table 3 from the simulated macro and asset pricing series directly

% STOCKS
% Equity Premium: Quarterly log return on average stock return in excess of the log 3-month Treasury bill
% plus one-half times the log excess return variance
equityPremium                = 4*(mean(asset.simulated.reteq) + .5*std(asset.simulated.reteq).^2/100);

% Stocks Volatility: Log excess stock return standard deviation (in annualized percent)
vol_stocks                   = std(asset.simulated.reteq)*2;

% Sharpe Ratio: Ratio between the equity premium and stocks volatility
sharpeRatio_stocks           = equityPremium/vol_stocks;

% AR(1) coeff. pd: Persistence of the price-dividend ratio
rhoDP                        = corrcoef(asset.simulated.pdlev(2:end,:), asset.simulated.pdlev(1:end-1,:));

% Coefficient and R^2 of 1-YR Excess Returns on pd: Predictability of annual stock returns from the lagged
% price-dividend ratio (in annualized percent)
ret1yr                       = conv(asset.simulated.reteq,ones(1,4),'valid')/100;
[re1_coef,~,~,~,R2_re1]      = regress(ret1yr, [ones(size(asset.simulated.pdlev(1:end-4))), ...
                               asset.simulated.pdlev(1:end-4)]);
coeffRegRetOnPD1y            = re1_coef(2);
R2RegRetOnPD1y               = R2_re1(1);


% 10-YEAR NOMINAL BONDS
% Yield Spread: The log 10-year bond yield minus the log nominal 3-month Treasury bill
spreadNom                    = asset.simulated.y10nom'-asset.simulated.rfr_nom;
meanLogYieldSpread           = mean(spreadNom);

% Volatility Excess Returns: Log excess bond return standard deviation (in annualized percent)
vol_nbond                    = std(asset.simulated.retnom)*2;

% Coefficient and R^2 of 1-YR Excess Returns on Yield Spread: Predictability of annual stock returns from
% the lagged Yield Spread (in annualized percent)
ret1yrNom                    = asset.simulated.retnom(4:end)+asset.simulated.retnom(3:end-1)...
                               +asset.simulated.retnom(2:end-2)+asset.simulated.retnom(1:end-3);
ret1yrNom                    = ret1yrNom/100;
[ys1_coef,~,~,~,R2_ys1]      = regress(100*ret1yrNom, [ones(size(spreadNom(1:end-4)')), spreadNom(1:end-4)']);
coeffRegRetOnYS1y            = ys1_coef(2);
R2RegRetOnYS1y               = R2_ys1(1);


% MACROECONOMIC DYNAMICS
% Std. Annual Consumption Growth: Standard deviation of 4-quarter real consumption growth (in percent)
consGrowthVol                = 100*std(asset.simulated.csim(5:end)-asset.simulated.csim(1:end-4));

% Std. Annual Change fed funds Rate: Standard deviation of 4-quarter change fed funds rate
iChangeVol                   = std(asset.simulated.rfr_nom(5:end)-asset.simulated.rfr_nom(1:end-4));


% Replica_Table3: Replicating Table 3
Replica_Table3  =   [equityPremium, vol_stocks, sharpeRatio_stocks, rhoDP(1,2), coeffRegRetOnPD1y, R2RegRetOnPD1y]';
Replica_Table3  =   [Replica_Table3; 0;  [meanLogYieldSpread, vol_nbond, coeffRegRetOnYS1y, R2RegRetOnYS1y]';0;consGrowthVol;iChangeVol];
Replica_Table3  =   [Replica_Table3; 0; min(asset.Irftemp.c)/max(asset.Irftemp.i); lag_trough];
Replica_Table3  =   [Replica_Table3, Table3Data];
Replica_Table3  =   array2table(round(Replica_Table3,2));

Replica_Table3.Properties.RowNames  =   Table3.Properties.RowNames;
Replica_Table3.Properties.VariableNames =   Table3.Properties.VariableNames;
Replica_Table3
```

Figure 15: Replication of Table 4 with simulated series

```
%% This section shows how to replicate Table 4 from the simulated macro and asset pricing series directly

% FF Shock: Is the stock market response to monetary policy shocks in model-simulated data
bernankeKuttnerTemp1        =   regress(asset.simulated.reteq_FOMC,[ones(size(asset.simulated.rfr_nom_FOMC)); asset.simulated.rfr_nom_FOMC]');
bernankeKuttnerTemp2        =   regress(asset.simulated.reteq_rn_FOMC,[ones(size(asset.simulated.rfr_nom_FOMC)); asset.simulated.rfr_nom_FOMC]');
bernankeKuttner             =   [bernankeKuttnerTemp1(2), bernankeKuttnerTemp2(2), bernankeKuttnerTemp1(2)-bernankeKuttnerTemp2(2)];


% FF Shock × (FF Shock>0): Is the stock market response to monetary policy shocks in model-simulated data when the shock is positive
bernankeKuttnerTemp1_dummy  =   regress(asset.simulated.reteq_FOMC,[asset.simulated.rfr_nom_FOMC; asset.simulated.rfr_nom_FOMC.*(asset.simulated.rfr_nom_FOMC>0);...
                                (asset.simulated.rfr_nom_FOMC>0); ones(size(asset.simulated.rfr_nom_FOMC))]');
bernankeKuttnerTemp2_dummy  =   regress(asset.simulated.reteq_rn_FOMC,[ asset.simulated.rfr_nom_FOMC; asset.simulated.rfr_nom_FOMC.*(asset.simulated.rfr_nom_FOMC>0);...
                                (asset.simulated.rfr_nom_FOMC>0); ones(size(asset.simulated.rfr_nom_FOMC))]');
bernankeKuttner_dummy       =   [bernankeKuttnerTemp1_dummy, bernankeKuttnerTemp2_dummy, bernankeKuttnerTemp1_dummy-bernankeKuttnerTemp2_dummy];


% Replica_Table4: Replicating Table 4
Replica_Table4              =   [bernankeKuttner(1), bernankeKuttner_dummy(1,1), bernankeKuttner(2), bernankeKuttner_dummy(1,2), bernankeKuttner(3), bernankeKuttner_dummy(1,3);
                                 0, bernankeKuttner_dummy(2,1), 0, bernankeKuttner_dummy(2,2), 0, bernankeKuttner_dummy(2,3)];

Replica_Table4              =   array2table(round(Replica_Table4,2));

Replica_Table4.Properties.RowNames      =   Table4.Properties.RowNames;
Replica_Table4.Properties.VariableNames =   Table4.Properties.VariableNames;
Replica_Table4
```

Figure 16: Minimum working example results with simulated series

Replica_Table3 =

17×2 table

	Model	Data
Equity Premium	7.29	7.89
Equity Vol	14.87	16.79
Equity SR	0.49	0.47
AR(1) pd	0.93	0.92
1 YR Excess Returns on pd	-0.34	-0.38
1 YR Excess Returns on pd (R^2)	0.07	0.23
-	0	0
Yield Spread	0.94	1.87
Return Vol.	2.57	9.35
1-YR Excess REturns on Yield Spread	-0.18	2.69
1-YR Excess REturns on Yield Spread (R^2)	0.01	0.14
--	0	0
Std. Annual Cons. Growth	1.55	1.5
Std Annual Change Fed Funds Rate	2.13	1.35
---	0	0
Trough Effect Consumption	-0.71	-0.7
Lag Trough	4	4

Replica_Table4 =

2×6 table

	Overall	Overall_	RN	RN_	RP	RP_
FF_Shock	-6.37	-6.31	-1.23	-1.22	-5.13	-5.08
FF_Shock_x_(FF_Shock>0)	0	-0.07	0	-0.02	0	-0.06

Figure 17: Minimum working example results with num1.N=3 - Tables 3 and 4

Table3 =

17×2 table

	Model	Data
Equity Premium	7.29	7.89
Equity Vol	14.86	16.79
Equity SR	0.49	0.47
AR(1) pd	0.93	0.92
1 YR Excess Returns on pd	-0.34	-0.38
1 YR Excess Returns on pd (R^2)	0.07	0.23
-	0	0
Yield Spread	0.94	1.87
Return Vol.	2.56	9.35
1-YR Excess REturns on Yield Spread	-0.18	2.69
1-YR Excess REturns on Yield Spread (R^2)	0.01	0.14
--	0	0
Std. Annual Cons. Growth	1.55	1.5
Std Annual Change Fed Funds Rate	2.13	1.35
---	0	0
Trough Effect Consumption	-0.71	-0.7
Lag Trough	4	4

Table4 =

2×6 table

	Overall	Overall_	RN	RN_	RP	RP_
FF_Shock	-6.36	-6.3	-1.23	-1.22	-5.13	-5.08
FF_Shock_x_(FF_Shock>0)	0	-0.07	0	-0.02	0	-0.06

Figure 18: Minimum working example results with num1.N=3 - Figure 3

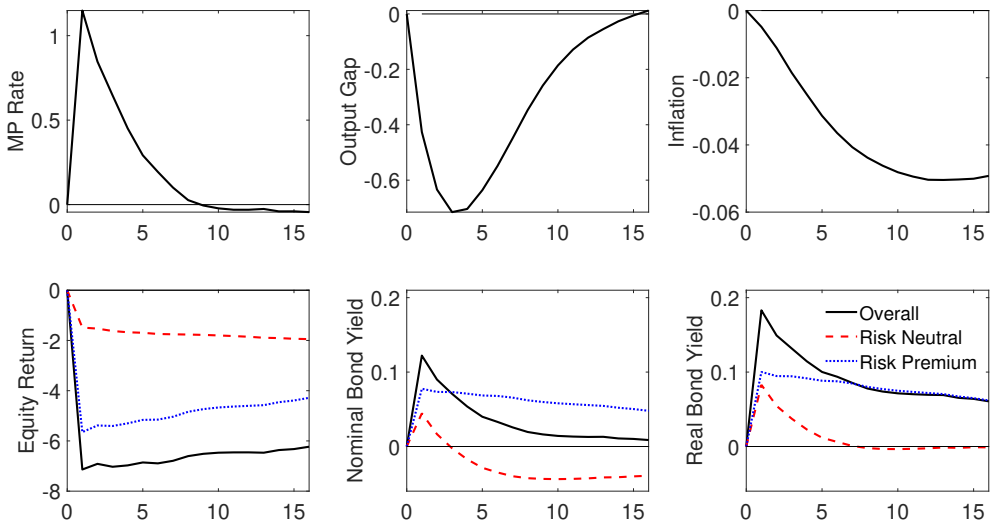


Figure 19: Minimum working example results with num1.sfast=8 - Tables 3 and 4

Table3 =

17×2 table

	Model	Data
Equity Premium	7.37	7.89
Equity Vol	14.93	16.79
Equity SR	0.49	0.47
AR(1) pd	0.93	0.92
1 YR Excess Returns on pd	-0.34	-0.38
1 YR Excess Returns on pd (R^2)	0.07	0.23
-	0	0
Yield Spread	0.95	1.87
Return Vol.	2.59	9.35
1-YR Excess REturns on Yield Spread	-0.18	2.69
1-YR Excess REturns on Yield Spread (R^2)	0.01	0.14
--	0	0
Std. Annual Cons. Growth	1.55	1.5
Std Annual Change Fed Funds Rate	2.13	1.35
---	0	0
Trough Effect Consumption	-0.71	-0.7
Lag Trough	4	4

Table4 =

2×6 table

	Overall	Overall_	RN	RN_	RP	RP_
FF_Shock	-6.39	-6.33	-1.23	-1.22	-5.16	-5.11
FF_Shock_x_(FF_Shock>0)	0	-0.07	0	-0.02	0	-0.06

Figure 20: Minimum working example results with num1.sfast=8 - Figure 3

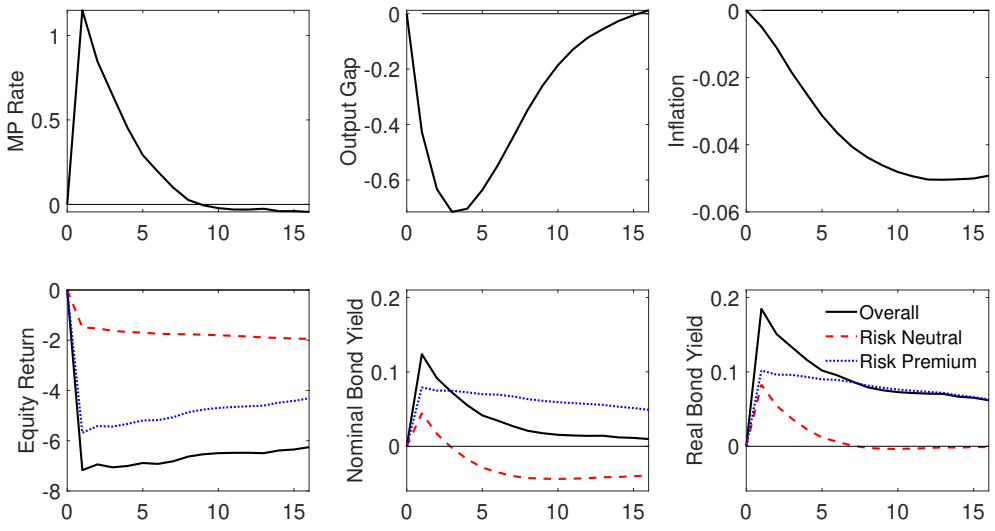


Figure 21: Minimum working example results with asset.testirf=1 - Figure 3