Minimum working example for programming package

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

August 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 Macroeconomic Drivers of Bond and Equity Risks (JPE, 2020)[1]. Specifically, in this code, we want to replicate some results from the first calibration of the model presented in Tables 2, 3, and 4 of the paper.[2] 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 not be exactly the same as in the paper due to simulation noise because the published code uses one simulation instead of two simulations of length 10000 as in the paper.

[2]This model calibration is used to analyze the period from 1979Q3 to 2001Q1.

- **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]

[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;

% 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 macro1 class, 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.
     - θ_2: Dependence of the surplus consumption ratio on lagged output gap.
     - ϕ: Smoothing parameter for consumption.
     - γ: Parameter controlling utility curvature.
     - 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.

     Also, we define some important matrices as:

     - corr_vNew: Correlations of model shocks (v_t). In the paper, these parameters are called $\rho_{\pi i}$, $\rho_{\pi *}$ and ρ_{i*}.
     - P: It's one of the matrices that determine the solution for the dynamics of Y_t, in the form $Y_t = PY_{t-1} + Qu_t$.

– sigma_vNew: Standard deviations of model shocks (v_t). In the paper, these parameters are called σ_π, σ_i and σ_*.

Figure 3: Calibration

```
%% Input parameters for Model

% Preference parameters
macro1.theta0 = 0.9658;         % Peristence surplus consumption
macro1.theta1 = -0.0500;        % Dependence Output Gap
macro1.theta2 = 0.0200;         % Dependence Lagged Output Gap
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

% Leverage parameter
macro1.delta  = 0.5000;

% Correlations of model shocks (v_t)
macro1.corr_vNew = [ 1.0000,    0.1100,    0.1100;
                     0.1100,    1.0000,   -0.1100;
                     0.1100,   -0.1100,   1.0000];

% Matrix needed to determine the solution for the dynamics of the model
macro1.P = [  0.8134,   -0.7781,   -0.1682;
             -0.1111,    0.5556,   -0.5556;
             -0.1111,   -0.1111,    0.3333];

% Standard deviations of model shocks (v_t)
macro1.sigma_vNew = [0.0017    0.0017    0.0014];

% Compute implied Euler equation coefficients
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]

[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".

Figure 6: Output while solving and simulating risk-neutral asset prices (line 76)

```
%% 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 for period 1:")
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 82 and 86)

```
%% 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);
```

8. **Reproduce moments :** In this part of the code, as you can see in Figure 8, we reproduce the results of Tables 2, 3 and 4 of the paper using only the structures "stocks", "nominalBonds" and "crossAsset".

Figure 8: Finding moments of the model

```
%% Table 2
disp("----------------------Table 2 - Stocks-----------------------------")
disp("Period 1 Model Moments")
asset.stocks
%% Table 3
disp("----------------------Table 3 - Bonds-----------------------------")
disp("Period 1 Model Moments")
asset.nominalBonds
%% Table 4
disp("----------------------Table 4 - Bonds and Stocks--------------------")
disp("Period 1 Model Moments")
asset.crossAsset
```

9. **Compare Outputs :** In Figure 9 we present the results of the minimum working example code and in Figure 10 we show the results from tables 2, 3 and 4 of the paper (these are the highlighted values). 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.

Figure 9: Minimum working example results

```
----------------------Table 2 - Stocks------------------------------
Period 1 Model Moments
ans =
  struct with fields:

        equityPremium: 11.0305
                  vol: 22.5234
          sharpeRatio: 0.4897
            meanPDlev: 14.1376
                stdDP: 0.1552
                rhoDP: 0.9555
    coeffRegRetOnPD1y: -0.3175
       R2RegRetOnPD1y: 0.0497
----------------------Table 3 - Bonds-------------------------------
Period 1 Model Moments
ans =
  struct with fields:

                termPremium: 1.7122
                        vol: 6.3142
                sharpeRatio: 0.2712
          meanLogYieldSpread: 0.9311
           volLogYieldSpread: 0.8143
    persistenceLogYieldSpread: 0.8224
            coeffRegRetOnYS1y: -0.1775
               R2RegRetOnYS1y: 5.1477e-04
----------------------Table 4 - Bonds and Stocks--------------------
Period 1 Model Moments
ans =
  struct with fields:

    corrNomStock: 0.4977
         betaNom: 0.1395
```

Figure 10: Paper results

Table 2: Stocks

	79Q3-01Q1		01Q2-11Q4	
	Empirical	Model	Empirical	Model
Excess Returns				
Equity Premium	7.97	10.99	4.03	8.11
Volatility	16.42	21.92	20.00	15.95
Sharpe Ratio	0.49	0.50	0.20	0.51
Log Price-Dividend Ratio				
Mean ($exp(mean(pd))$)	34.04	14.43	53.73	20.33
Volatility	0.46	0.14	0.20	0.16
AR(1) Coefficient	1.00	0.95	0.86	0.98
Predictability				
1-YR Excess Return on pd	-0.01	-0.33	-0.43	-0.22
R^2	0.00	0.05	0.22	0.05

Table 3: Bonds

	79Q3-01Q1		01Q2-11Q4	
	Empirical	Model	Empirical	Model
Excess Returns				
Term Premium	2.31	1.68	3.23	-1.41
Volatility	8.37	6.26	5.98	3.78
Sharpe Ratio	0.28	0.27	0.54	-0.37
Yields				
Mean log Yield Spread	1.16	0.94	1.40	-0.77
Volatility	1.29	0.82	0.93	0.47
AR(1) Coefficient	0.70	0.82	0.79	0.89
Predictability				
1-YR Excess Returns on log Yield Spread	2.78	-0.15	0.39	0.89
R^2	0.19	0.00	0.01	0.01

Table 4: Bonds and Stocks

	79Q3-01Q1		01Q2-11Q4	
	Empirical	Model	Empirical	Model
Bond-Stock Comovement				
Correlation Bond and Stock Returns	0.21	0.50	-0.64	-0.66
Beta Bond Returns on Stock Returns	0.11	0.14	-0.19	-0.16
Nominal-Real Comovement				
Correlation Quarterly Inflation and Output Gap	-0.28	-0.37	0.65	0.35
Correlation 5-Year Average Inflation and Output Gap	-0.15	-0.05	0.20	0.14
Correlation 5-Year Average Federal Funds Rate and Output Gap	-0.38	-0.38	0.57	0.57
Predictability				
1-YR Excess Stock Return on Output Gap	-1.05	-1.56	-4.71	-0.54
R^2	0.02	0.02	0.21	0.04
1-YR Excess Bond Return on Output Gap	-0.89	-0.20	-0.11	0.05
R^2	0.05	0.00	0.00	0.01

10. **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 replicate the results we show in Figure 9. So, in Figures 11, 12 and 13, we have the replication of the same variables that we have in tables 2, 3 and 4, respectively. Thus in Figure 14 we have the results of this replication, which are the same as in Figure 9.

[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.

Figure 11: Replication of Table 2 with simulated series

```
%% Replicate Table 2

% 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;

% Mean of the price-dividend ratio
meanPDlev                    = exp(mean(asset.simulated.pdlev));

% Volatility of the price-dividend ratio
stdDP                        = std(asset.simulated.pdlev);

% 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);


% Replicating Table 2
disp("-----------------------Replicate Table 2 - Stocks-----------------------------")
var_names2                   = ["equityPremium"; "vol"; "sharpeRatio"; "meanPDlev"; "stdDP"; "rhoDP"; ...
                                "coeffRegRetOnPD1y"; "R2RegRetOnPD1y"];
var_vals2                    = [equityPremium vol_stocks sharpeRatio_stocks meanPDlev stdDP rhoDP(1,2) ...
                                coeffRegRetOnPD1y R2RegRetOnPD1y]' ;
for i=1:size(var_names2)
    disp(sprintf('%s: %.4f ',var_names2(i),var_vals2(i)));
end
```

Figure 12: Replication of Table 3 with simulated series

```
%% Replicate Table 3

% BONDS
% Term Premium: The average log 5-year nominal bond return in excess of the log nominal
% 3-month T-bill plus one-half times the log excess return variance
termPremium                   = 4*(mean(asset.simulated.retnom) + .5*std(asset.simulated.retnom).^2/100);

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

% Sharpe Ratio: Ratio between the term premium and bonds volatility
sharpeRatio_nbonds            = termPremium /vol_nbond;

% Yield Spread: The log 5-year bond yield minus the log nominal 3-month Treasury bill
spreadNom                     = asset.simulated.y5nom'-asset.simulated.rfr_nom;
meanLogYieldSpread            = mean(spreadNom);

% Volatility Yield Spread: The standard deviation of bond yield spreads
volLogYieldSpread             = std(spreadNom);

% AR(1) coeff. ys: Persistence of the yield spread
persistenceLogYieldSpread     = corrcoef(spreadNom(2:end), spreadNom(1:end-1));

% Coefficient and R^2 of 1-YR Excess Returns on log ys: 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);


% Replicating Table 3
disp("------------------------Replicate Table 3 - Bonds--------------------------------")
var_names3                    = ["termPremium"; "vol"; "sharpeRatio"; "meanLogYieldSpread"; "volLogYieldSpread";...
                                 "persistenceLogYieldSpread"; "coeffRegRetOnYS1y"; "R2RegRetOnYS1y"];
var_vals3                     = [termPremium vol_nbond sharpeRatio_nbonds meanLogYieldSpread volLogYieldSpread ...
                                 persistenceLogYieldSpread(1,2) coeffRegRetOnYS1y R2RegRetOnYS1y]' ;
for i=1:size(var_names3)
    disp(sprintf('%s: %.4f ',var_names3(i),var_vals3(i)));
end
```

Figure 13: Replication of Table 4 with simulated series

```
%% Replicate Table 4

% BOND-STOCK COMOVEMENT
% Correlation Bond and Stock Returns: The correlation of quarterly log bond excess returns
% with log stock excess returns
bondstock_corr_temp           = corrcoef(asset.simulated.retnom, asset.simulated.reteq);
tipsstock_corr_temp           = corrcoef(asset.simulated.retreal, asset.simulated.reteq);
correlations                  = [bondstock_corr_temp; tipsstock_corr_temp];
corrNomStock                  = correlations(1,2);

% Beta Bond Returns on Stock Returns: The slope coefficient from regressing quarterly
% log bond excess returns onto log stock excess returns
beta_temp                     = regress(asset.simulated.retnom, [ones(size(asset.simulated.retnom,1),1), ...
                                asset.simulated.reteq(:,1)]);
betaNom                       = beta_temp(2);

% Replicating Table 4
disp("-----------------------Replicate Table 4 - Bonds and Stocks----------------------")
var_names4                    = ["corrNomStock"; "betaNom"];
var_vals4                     = [corrNomStock betaNom]' ;
for i=1:size(var_names4)
    disp(sprintf('%s: %.4f ',var_names4(i),var_vals4(i)));
end
```

Figure 14: Minimum working example results with simulated series

```
-----------------------Replicate Table 2 - Stocks---------------------------------
equityPremium: 11.0305
vol: 22.5234
sharpeRatio: 0.4897
meanPDlev: 14.1376
stdDP: 0.1552
rhoDP: 0.9555
coeffRegRetOnPD1y: -0.3175
R2RegRetOnPD1y: 0.0497
-----------------------Replicate Table 3 - Bonds----------------------------------
termPremium: 1.7122
vol: 6.3142
sharpeRatio: 0.2712
meanLogYieldSpread: 0.9311
volLogYieldSpread: 0.8143
persistenceLogYieldSpread: 0.8224
coeffRegRetOnYS1y: -0.1775
R2RegRetOnYS1y: 0.0005
-----------------------Replicate Table 4 - Bonds and Stocks----------------------
corrNomStock: 0.4977
betaNom: 0.1395
```

11. **Additional results :** As we mentioned above, we can change the value of some parameters. For example, in Figure 15 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 Figure 16, 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 15 and 16 are very similar to the original results of the paper and the minimum working example.[7]

    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.[8] So the average speed of the minimum working example's original code is 111.13 seconds, which is faster than the versions with num1.N=3 and num1.sfast=8 , which have an average speed of 402.96 and 2165.72 seconds, respectively.[9]

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

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

[9] 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 R2020a.

Figure 15: Minimum working example results with num1.N=3

```
--------------------Table 2 - Stocks-----------------------------
Period 1 Model Moments
ans =
  struct with fields:

        equityPremium: 11.0306
                  vol: 22.5225
          sharpeRatio: 0.4898
            meanPDlev: 14.1372
                stdDP: 0.1552
                rhoDP: 0.9555
    coeffRegRetOnPD1y: -0.3175
       R2RegRetOnPD1y: 0.0497
----------------------Table 3 - Bonds------------------------------
Period 1 Model Moments
ans =
  struct with fields:

                 termPremium: 1.7123
                         vol: 6.3142
                 sharpeRatio: 0.2712
           meanLogYieldSpread: 0.9311
            volLogYieldSpread: 0.8143
    persistenceLogYieldSpread: 0.8224
            coeffRegRetOnYS1y: -0.1775
               R2RegRetOnYS1y: 5.1477e-04
---------------------Table 4 - Bonds and Stocks--------------------
Period 1 Model Moments
ans =
  struct with fields:

    corrNomStock: 0.4977
         betaNom: 0.1395
```

Figure 16: Minimum working example results with num1.sfast=8

```
----------------------Table 2 - Stocks------------------------------
Period 1 Model Moments
ans =
  struct with fields:

        equityPremium: 11.1718
                  vol: 22.6441
          sharpeRatio: 0.4934
            meanPDlev: 13.9225
                stdDP: 0.1570
                rhoDP: 0.9557
    coeffRegRetOnPD1y: -0.3177
       R2RegRetOnPD1y: 0.0504
----------------------Table 3 - Bonds-------------------------------
Period 1 Model Moments
ans =
  struct with fields:

                 termPremium: 1.7315
                         vol: 6.3111
                 sharpeRatio: 0.2744
           meanLogYieldSpread: 0.9401
            volLogYieldSpread: 0.8154
    persistenceLogYieldSpread: 0.8226
            coeffRegRetOnYS1y: -0.1731
               R2RegRetOnYS1y: 4.9147e-04
----------------------Table 4 - Bonds and Stocks--------------------
Period 1 Model Moments
ans =
  struct with fields:

    corrNomStock: 0.4972
         betaNom: 0.1386
```