Code Documentation for "Financial Fragility with SAM?"

Daniel L. Greenwald, Tim Landvoigt, Stijn Van Nieuwerburgh*

May 26, 2020

Contents

*Greenwald: Massachussetts Institute of Technology Sloan School; email: dlg@mit.edu. Landvoigt: University of Pennsylvania Wharton School, NBER, and CEPR; email: timland@wharton.upenn.edu. Van Nieuwerburgh: Columbia University Graduate School of Business, NBER, and CEPR, 3022 Broadway, Uris Hall 809, New York, NY 10027; email: svnieuwe@gsb.columbia.edu.

1 Fast Steps to Reproduce Benchmark Model

1. In Matlab, execute the script "`main_create_env`".
   - If you have the Matlab Symbolic Math Toolbox and use version 2018b or earlier, leave the flag "`useJacobian`" in line 43 on. Otherwise set to false to use numerical differentiation.
   - Note: generating the analytic Jacobian for the benchmark model takes approximately 5 minutes with version 2018b, and can take longer for the other experiments.
   - "`main_create_env`" will create a file "`env_bench_ini0`" that contains the experiment definition for the benchmark economy.
   - The code archive contain the pre-computed file.
2. Execute script "`main_run_exper`".
   - You can set the number of parallel workers to be started in the separate script "`open_parpool`"
   - Set to zero if you want to run it with a single process.
   - On a computer with sixteen cores (and 16 parallel workers) the benchmark model converges in about 3 hours.
   - "`main_run_exper`" creates a results file named "`res_[current_date_time]`" that contains the converged policy functions.
   - The code archive contain the pre-computed named "`res_20191112_bench`".
2. Simulate the model using "`sim_stationary`" and "`sim_trans_cluster`".
   - "`sim_stationary`" simulates the model policies contained in "`res_20191112_bench`" for 10,000 periods and writes out the resulting time-series and several statistics. The main output is a file named "`sim_res_20191112_bench`".
   - "`sim_trans_cluster`" reads both "`res_20191112_bench`" and "`sim_res_20191112_bench`", and simulates generalized IRFs.
   - To plot IRFs, run "`plot_trans`".

2 Overall Structure

The overall approach of the computational procedure is to represent the economy as one large system of functional equations, with the unknown functions being the choice variables of the household problems and the asset prices. We then use "policy function iteration" (Judd (1998)) to compute the policy and price functions that depend on the state variables.

The key steps are solving of a system on nonlinear equations at each point in the state space given a guess for future policy and price functions, and then updating these approximated functions based on the newly found solution. Iterate on this until convergence. For a detailed description of the algorithm see Appendix B.2 of the paper.

2.1 Algorithm Implementation

At a high level, we can describe the necessary steps for the algorithm as follows. First, define bounds for the endogenous state variables. We are approximating the system in a hypercube, hoping that it is stationary within these bounds. Choose the kind of approximation, e.g. how many spline knots in each dimension. For the implementation in this paper, we use multivariate linear interpolation. Then create an initial guess of the policy and price functions (and other functions required to compute expectations in Euler equations). Using this guess, the algorithm proceeds as follows:

1. Loop over all individual points in the state space, and solve a nonlinear system of N equations (FOCs and constraints) in N unknowns (choices, prices, and Lagrange multipliers), using last guess to compute expectations. Save the solution vector at each point.
2. Update policy and price functions using new solution. This becomes the next guess.
3. If distance between this new guess and last guess is below a certain threshold, stop. Otherwise go back to step 1.

2.2 Transition Functions for Endogenous State Variables

Depending on the specifics of the problem, it may not be possible to compute next period's state variables in closed-form form only based today's state variables, choices, and prices. As an example

in this specific paper, intermediary wealth next period depends on next period's prices, which in turn are a function of next period's intermediary wealth. Since the problem has a first-order Markov structure in its state variables (see Kubler and Schmedders (2003)), we can define a mapping from today's aggregate state variables into tomorrow's states, and approximate these transition functions numerically. These functions are updated between iterations in step 2. of the algorithm above along with policy functions. This variant of policy iteration is called "transition function iteration" and was developed in Elenev, Landvoigt, and Van Nieuwerburgh (2016) and Elenev, Landvoigt, and Van Nieuwerburgh (2020).

3 Preparing the Computation

3.1 Setting Parameters and Defining Numerical Experiments

The file "`experdef_20191112.m`" sets the parameters of the model for all numerical experiments in the paper. The baseline calibration described section III of the paper is the fixed-rate-only benchmark model. All other experiments contained in the paper, for instance aggregate or local indexation, are variations on the parameters of the benchmark model. These are defined in the MATLAB struct "`expers_calib`" in line 268ff of "`experdef_20191112.m`". Other experiments can defined by adding lines to this struct.

3.1.1 Creating an Experiment Definition File

The script "`main_create_env`" reads the experiment definitions from "`experdef_20191112.m`" and creates a MATLAB data file that serves as input for a run of the computational algorithm. The most important inputs to set at the top of the file are name of the experiment definition script, and name of the experiment for which the definition file should be created, e.g. "`experdef_20191112.m`" and "`bench`" to create a definition file for the benchmark economy in the paper.

"`main_create_env`" computes an approximate non-stochastic steady state of the economy (making assumptions on the bindingness of constraints and the wealth distribution). This steady-state serves as initial guess for policies and prices of the dynamic stochastic model. The script performs several

other steps needed to initialize the code and then writes the result into a file that serves as input for the actual computation script.

3.1.2 Running the Computation

Script "`main_run_exper`" reads the contents of the file created by "`main_create_env`", and starts the computation. It either runs until convergence or until it hits the maximum number of iterations. The key inputs that should be set at the top of the file are:

- name of the experiment definition file,
- maximum number of iterations,
- convergence tolerance,
- number of parallel workers to start (see also "`open_parpool`"; set to 0 for no parallel workers),
- number of additional value function iterations.

The actual model computation is in a class file (using MATLAB's object-oriented features), which is called "`SAMIntermediaryModel.m`". The two key methods are "`calcStateTransition`" and "`calcEquations`". The first method uses budget constraints and market clearing conditions, and evaluates the transition functions to calculate the state variables for the next period. It is useful to separate this part of the code, since it is also needed to simulate the model after the solution has been calculated. The second method uses current choices/prices, and the values of next-period state variables as inputs to compute the equilibrium equations listed in Appendix B.1. Note that these functions are called at each point in the state space by MATLAB's nonlinear equation solver 'fsolve' many times.

Once the computation is done, either by convergence or because it reached the maximum number of iterations, it will write final policy and transition functions in a MATLAB data file.

Note that "`main_run_exper`" executes additional iterations after the main loop of the algorithm has converged if the control parameter "`maxit_VF`" is set to a number greater than zero. These additional iterations do not update policy functions, but only iterate forward on the agents' value

functions. This can be useful for welfare comparisons; in the model's baseline calibrations savers' discount factor is 0.998. Convergence of savers' value function to acceptable precision for welfare comparisons – especially for model variations with small welfare differences – requires these additional iterations. One could also run the model for more full iterations during which all model functions are recomputed and updated. However, these functions do not change significantly with additional iterations, yet the saver value function still does. This is because in a model like ours, that contains agents with different preferences, different equilibrium functions have different speed of convergence (e.g., borrower and intermediary value functions converge much faster than the saver value function). The iteration scheme used for all results reported in the paper is pre-configured in "`main_run_exper`": 150 full and 100 value-function-only iterations.

4 Processing the Results

4.1 Running a Long Simulation

"`sim_stationary`" can be used to simulate the economy for many periods and compute moments of the variables. The script also calculates Euler equation errors along the simulated path. The script calls another method in the model-specific class "`SAMIntermediaryModel.m`" that is called "`computeSimulationMoments`". Input for "`sim_stationary`" is the results file produced by "`main_run_exper`". It creates another file that contains the simulated data for the model. It can also create Excel files with model moments.

4.2 Simulating IRFs and Transition Paths

"`sim_trans_cluster`" can be used to simulate paths of the economy after it was hit by a specific shock. The paths are initiated at the ergodic steady state of the stochastic model that is calculated by applying a clustering algorithm to the long time-series of state variables created by "`sim_stationary`". To plot IRFs, use the scripts "`plot_trans`" or "`plot_trans_finrec`". The first script was used to create Figures 3 and 4 in the paper and is for comparison of different shocks within in the same model parameterization. The second script was used to create Figures 5 and 6 in the paper and is for comparison of the same shock across different model parameterizations.

4.3 Computing Welfare Comparisons

To compare compensating-variation welfare measures reported as CEV-welfare in the paper, we first need to compute prices for the consumption streams of agents. This is done in the script "`priceZNS.m`" based on the model's solution file. After "`priceZNS.m`" has finished, "`welfare.m`" creates an Excel sheet that contains the welfare comparison of the different economies for which solution files are present in the main code folder to the benchmark.

References

Elenev, V., T. Landvoigt, and S. Van Nieuwerburgh (2016): "Phasing out the GSEs," Journal of Monetary Economics, 81, 111 – 132.

——— (2020): "A Macroeconomic Model with Financially Constrained Producers and Intermediaries," Working Paper.

Judd, K. L. (1998): Numerical Methods in Economics, The MIT Press.

Kubler, F. and K. Schmedders (2003): "Stationary Equilibria in Asset-Pricing Models with Incomplete Markets and Collateral," Econometrica, 71, 1767–1795.