Code Documentation for "Financial Regulation in a Quantitative Model of the Modern Banking System"

Juliane Begenau and Tim Landvoigt*

September 11, 2021

Contents

*Begenau: Stanford University, Graduate School of Business; email: begenau@stanford.edu. Landvoigt: University of Pennsylvania Wharton School; email: timland@wharton.upenn.edu.

This folder contains all the code to replicate the results of Begenau and Landvoigt "Financial Regulation in a Quantitative Model of the Modern Banking System", forthcoming at Review of Economic Studies. This code is also available online at:
https://github.com/timlandvoigt/FinRegModernBanking.

1 Hardware and Software Requirements

All results in the paper were produced by running Matlab version 2020a. Toolboxes required are: Parallel Computing, Statistics, Curve Fitting, Optimization. The code should run fine on older and newer versions of Matlab, but we did not test this.

The code can run on a regular Windows PC or Mac with 16GB RAM. With 4 cores, the running time for baseline economy is about 4 hours. All experiments reported in the paper were run on a Linux High Performance Computing Cluster with 16 cores per experiment. This cuts the running time approximately in half.

2 Fast Steps to Reproduce Benchmark Model

1. In Matlab, execute the script `mainSB_create_env`.
   - `mainSB_create_env` will create a file `env_base` that contains the experiment definition for the benchmark economy.
2. Execute script `main_run_exper` to run the base economy.
   - You can set the number of parallel workers to be started in line 29.
   - Set to zero if you want to run it with a single process.
   - On a computer with sixteen cores (and 16 parallel workers) this should take about 90 minutes.
   - `main_run_exper` creates a results file named `res_[current_date_time]` that contains the converged policy functions.
   - Rename this file to `res_20201117_base.mat`.

3. Simulate the model using `sim_stationary` and `sim_trans`.

   – `sim_stationary` simulates the model policies contained in `res_20201117_base.mat` for 5,000 periods and writes out the resulting time-series and several statistics. The main output is a file named `sim_res_20201117_base.mat`.

   – `sim_trans` reads both `res_20201117_base.mat` and `sim_res_20201117_base.mat`, and simulates generalized IRFs.

   – To plot IRFs, run `plot_trans`.

3 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 an algorithm that is an implementation of Judd (1998)'s "policy function iteration", 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 Online Appendix B of the paper.

3.1 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.

3.2 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, see Kubler and Schmedders (2003). This is not the case for this paper; all state variables of the model have "natural" transition laws that can be fully expressed as function of objects know in the current period. Thus, the algorithm in this paper does not make explicit use of "transition functions" introduced in Elenev, Landvoigt, and Van Nieuwerburgh (2016). Instead, this is a simpler algorithm that computes the explicit transition of all endogenous state variable as function of current policies (i.e., the algorithm computes the left-hand and right-hand side of all Euler equations during the solution step, see also Elenev, Landvoigt, and Van Nieuwerburgh (2021).)

4 Preparing the Computation

4.1 Setting Parameters and Defining Numerical Experiments

The file `experdef_2020117.m` sets the parameters of the model for all numerical experiments in the paper. The baseline calibration is described in Section 4 of the paper. All other experiments contained in the paper, for instance the variations of the capital requirement for Table 5, are variations on the parameters of the base model. These are defined in the MATLAB struct `allexpers` in line 332 of `experdef_20201117.m`. Other experiments can defined by adding lines to this struct. Note that state variable grids are part of each experiment definition. If no custom grid is specified for a given experiment, it defaults to the `basegrid`.

4.1.1 Creating an Experiment Definition File

The script `mainSB_create_env` reads the experiment definitions from `experdef_20201117.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_2020117.m` and `base` to create a definition file for the benchmark economy in the paper.

`mainSB_create_env` computes an approximate deterministic steady state of the economy (making assumptions on the bindingness of constraints). 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.

4.1.2 Running the Computation

Script `main_run_exper` reads the contents of the file created by `mainSB_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 (set to 0 for no parallel workers),

The actual model computation is in a class file (using MATLAB's object-oriented features), which is called `SBModel_CD.m`. The two key methods are `calcStateTransition` and `calcEquations`. The first method uses budget constraints and market clearing conditions, and calculates 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. 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.

5 Processing the Results

5.1 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 `SBModel_CD.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; the code archive contains the output of the simulation for the paper's benchmark economy in `sim_res_20201117_base.mat`. It can also create Excel files with model moments.

5.2 Simulating IRFs and Transition Paths

`sim_trans` 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 script `plot_trans`. This script was used to create Figures 1 in the paper and is for comparison of different shocks within in the same model parameterization. To generate the transition paths in Figure 2 of the paper, `sim_trans_GFC` simulates a path that evolves through several economies with parameter changes ("MIT shock") until reaching a final economy. These paths can be plotted using `plot_trans_GFC`.

6 Step-by-Step Detailed Replication Instructions

This section lists all steps required to reproduce all the computational results in the paper. Note that this requires computing the model solution for all parameter combinations specified in `experdef_20201117.m`. It is impractical to compute all of these experiments locally; rather, we run all experiments as a batch job on a high-performance computing cluster (HPCC). The code repository contains sample scripts for batch execution on a Linux-based system with a job scheduler (`hpcc_script_aws_array.sh`). Depending on the specific HPCC environment, the implementation of batch execution will of course

have to be adapted. However, `experdef_20201117.m` fully specifies all aspects of experiment definitions required for the Matlab code, irrespective of the HPCC environment. Below is the content of file `readme.txt` also in the repository.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Part 1: Solve and Simulate the Model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

This part can be done locally or on a HPCC cluster. Instructions below are for local execution.

------------------------------
Part 1a: Computing locally
------------------------------

Repeat the steps below for each economy in experdef_20201117.txt. The definitions of these econ

Run the economy for up to 300 iterations.
1.  Configure mainSB_create_env.m as follows (leave other variables as is):
- expername = 'econ';
- guess_mode = 'no_guess';
2.  Run mainSB_create_env.m and it will produce a file called env_econ.mat
3.  Configure main_run_exper.m as follows (leave other variables as is):
- no_par_processes = XX; % where XX is the number of processors on the machine you're running i
- exper_path = 'env_econ.mat';
- maxit = 300;
4.  Run main_run_exper.m. On a machine with 16 cores, this should take about 1 hour. It will cr
5.  Rename the res_TIMESTAMP.mat file to res_2020117_econ.mat.

Next, simulate. The model must be solved and res* file must exist.
```

6. Configure sim_stationary.m as follows (leave other variables as is):

- resfile = 'res_20201117_econ';

7. Run sim_stationary.m. If you are running sim_stationary having run it before during the curr

- sim_res_20201117_econ.mat: all simulation results incl. full series, statistics, errors, and

- Results/statsexog_res_20201117_econ.xls: statistics using exogenous subsampling to define cri

- Results/errstats_res_20201117_econ.xls: statistics of EE errors

Next, compute impulse response functions. Model must be solved and simulated. Both res* and sim

8. Configure sim_trans.m as follows (leave other variables as is):

- resfile_list = {'res_20201117_econ'};

9. Run sim_trans.m. On a machine with 16 cores, this should take about 10 min. It will create t

- GR_res_20201117_econ.mat: mean, median, and sd of IRF paths for each of 4 shocks (no shock, r

---

Part 1b: Computing on the cluster

The following instructions work for a HPCC cluster that uses a job scheduling engine for batch

The configuration needed to run the numerical experiments as batch job on a cluster will vary s

%%%

Part 2: Compute and Save Results

%%%

This part pre-supposes that the res* and sim_res* for each economy created by Part 1 exist. NOT

The steps below re-create all the results used in the paper.

1. Run writeStats.m. This will create Results/simres.mat, which contains a variable called "ma
map of every simulation and IRF statistic and model parameter. They can be retrieved by running
[command] - [econ] - [subsmaple] - [variable] - [statistic]

command:
- sim: simulation statistic
- comp: simulation statistic relative to bench (either level or percent change)

econ: name of the economy (as in experdef_20200910.txt)

subsample: if command is "sim" or "comp"
- u: unconditional
- e: expansions
- r: recessions
- c: crises (ie run recessions)

variable: all variables reported by sim_stationary (as well as welfare "VHcons" for unconditior

statistic: for command = "sim" and "comp", all statistics reported by sim_stationary e.g. "mear

2. After simres.mat has been created in step 1, use build.m script in subfolder "paper_tables"

build('BLrev_v2021_tables','../Results/simres.mat',[])

This will create a new file "BLrev_v2021_tables_filled.tex", in which placeholders are replaced

3. Compute base economy IRF graphs, Figure 1 in the paper.
a. Configure sim_trans.m as follows:
- resfile_list = {'res_2020117_base'};

- no_par_processes = XX; % where XX is the number of processors on the machine you're running

b. Run sim_trans.m. This will create GTR_res_20201117_base.mat.

c. Configure plot_trans.m as follows:

- resfile = 'res_20201117_base';

Running plot_trans.m will create Figure 1.(pdf|eps).

4. Compute financial crisis simulations graph, Figure 2 in the paper.

a. Configure sim_trans_GFC.m as follows:

- resfile_start = 'res_20201117_precrisis';

- resfile_end = 'res_20201117_postcrisis';

- resfile_steps={'res_20201117_postcrisis085',...

'res_20201117_postcrisis085',...

'res_20201117_postcrisis09',...

'res_20201117_postcrisis09',...

'res_20201117_postcrisis095',...

'res_20201117_postcrisis095',...

'res_20201117_postcrisis10',...

'res_20201117_postcrisis10',...

'res_20201117_postcrisis105',...

'res_20201117_postcrisis105'};

- no_par_processes = XX; % where XX is the number of processors on the machine you're running

b. Run sim_trans_GFC.m. This will create GFC_res_20201117_postcrisis.mat.

c. Configure sim_trans_GFC.m as follows:

- resfile_start = 'res_20201117_precrisis';

- resfile_end = 'res_20201117_postcrisis08';

- resfile_steps={'res_20201117_postcrisisc1',...

'res_20201117_postcrisisc1',...

'res_20201117_postcrisisc2',...

'res_20201117_postcrisisc2',...

'res_20201117_postcrisisc3',...

'res_20201117_postcrisisc3',...

```
'res_20201117_postcrisisc4',...
'res_20201117_postcrisisc4',...
'res_20201117_postcrisisc5',...
'res_20201117_postcrisisc5'};
- no_par_processes = XX;  % where XX is the number of processors on the machine you're running
d. Run sim_trans_GFC.m. This will create GFC_res_20201117_postcrisis08.mat.
e. Run plot_trans_GFC.m. This scripts reads the contents of the two GFC_ files generated above
```

References

Vadim Elenev, Tim Landvoigt, and Stijn Van Nieuwerburgh. Phasing out the gses. Journal of Monetary Economics, 81:111–132, 2016.

Vadim Elenev, Tim Landvoigt, and Stijn Van Nieuwerburgh. A macroeconomic model with financially constrained producers and intermediaries. Econometrica, 89(3):1361–1418, 2021.

Kenneth L. Judd. Numerical Methods in Economics. The MIT Press, 1998. URL http://ideas.repec.org/b/mtp/titles/0262100711.html.

Felix Kubler and Karl Schmedders. Stationary equilibria in asset-pricing models with incomplete markets and collateral. Econometrica, 71:1767–1795, 2003.