Build Status

Overview

This repository contains the complete code used to calibrate the model and to compute equilibrium steady-state and transition dynamics of the model in Equilibrium Technology Diffusion, Trade, and Growth by Perla, Tonetti, and Waugh (AER 2020); OpenICPSR project number openicpsr-119393.

The derivation document has the complete set of equations defining the model. All equation numbers in the code refer to this document. General code and derivations for upwind finite difference methods are in the SimpleDifferentialOperators.jl package with detailed derivations.

As in the derivation, the code has a "warmup" model without trade or monopolistic competition to help build an understanding of the transition dynamics in the full model in a related but simpler growth model. The warmup model is also helpful for experimenting with the DAE and finite-difference discretization methods.

Directory of Code

- **/src/full** contains all of the code used to compute the numerical results presented in the paper, given parameter values.
- **/src/calibration** contains all of the code used to create empirical moments and calibrate the model to match those moments. Parameter values are outputs. Readme file describes this part of the code.

Directory of Notebooks for Results in Paper

The following notebooks compute all of the quantitative results and figures presented in the paper. They are organized by Section in accordance with NBER working paper No. 20881 (April 2020 revision).

- **Section 7-1: Calibration—Compute Empirical Moments** jupyter (python) notebook which directly pulls data and constructs empirical moments for the calibration of the PTW model. The resulting output are moments saved as `.csv` files which are then imported into the simulated method of moments calibration routine. **NOTE** the firm-level moments (transition probabilities and firm entry rate) are from the restricted access SynLBD. The Stata code that constructs the SynLBD-derived moments, the resulting output log files, and instructions on how to access the SynLBD are at `/src/calibration/SynLBD/`.
- **Section 7-2: Calibration—SMM and Resulting Parameter Values** jupyter (python) notebook which calls the MATLAB code to implement the calibration procedure, which finds parameter values such that moments simulated from the model best fit moments in data. This procedure has parameter values as output, which are saved at `/parameters/calibration_params.csv`. Headers for each column identify the parameter associated with each value.
- **Section 7.3 The Sources of the Welfare Gains from Trade—A Quantitative Decomposition** jupyter (julia) notebook that performs the local welfare decomposition and associated results contined in the discussion surrounding the decomposition in the paper.

- **Section 7.4 The Welfare Effects of a Reduction in Trade Costs** jupyter (python) notebook which (i) calls Julia notebook TransitionDynamics.ipynb and computes the transition dynamics of the economy and (ii) plots the results to create Figures 1-5 of the paper.
- **Section 7.4 The Welfare Effects of a Reduction in Trade Costs, Further Analysis** jupyter (python) notebook which (i) performs calibration routines by calling MATLAB routines and (ii) computes welfare gains (by calling julia notebook SteadyState.ipynb) for alternatives to the baseline model (ACR, Sampson, Atkenson and Burstein, No GBM).
- **Section 7.5. The Role of Firm Dynamics and Adoption Costs** jupyter (python) notebook (i) which calls MATLAB code to generate results for different GBM and delta shock parameter values and (ii) plots the results corresponding with Figure 6 and 7 of the paper. Also computes related results contained in the text of Section 7-5.

Installation and Use

1. Follow the instructions to install Julia and Jupyter

   1a. **Optional**: A Python installation is required for most of the figures.

   1b. **Optional**: Re-running the calibration requires Matlab.

   1c. **Optional**: Computing the SynLBD-derived moments requires Stata.

2. Open the Julia REPL (see the documentation above) and then install the package (by entering package mode with `]`) with

   ```
   ] add https://github.com/jlperla/PerlaTonettiWaugh.jl.git
   ```

   2a. **Optional, but strongly encouraged**: To install the exact set of packages used here (as opposed to using existing compatible versions on your machine), first run the following in the REPL

   ```
   using PerlaTonettiWaugh # will be slow the first time, due to precompilation
   cd(pkgdir(PerlaTonettiWaugh))
   ```

   Then, enter the package manager in the REPL (**note** that `]` cannot be copy-pasted on OSX; you need to type it to enter the REPL mode.)

   ```
   ] activate .; instantiate
   ```

3. There are several ways you can run the notebooks after installation

   Using the built-in Jupyter is straightforward. In the Julia terminal

```
using PerlaTonettiWaugh, IJulia
jupyterlab(dir=pkgdir(PerlaTonettiWaugh))
```

Note: If this is the first time running a `jupyterlab()` command inside Julia, it may prompt you to install Julia via conda. Hit `yes`. Also, this will **hand over control of your Julia session to the notebook.** To get it back, hit `control+c` in the Julia REPL.

Alternatively, to use a separate Jupyter installation you may have installed with Anaconda,

```
using PerlaTonettiWaugh
cd(pkgdir(PerlaTonettiWaugh))
; jupyter lab
```

where the last step runs your `jupyter lab` in the shell. **The ; cannot be copy-and-pasted**; to access shell mode, you must type it manually (and you will see your prompt go red).

In either case, the first time running the `using` command will be very slow, as all dependencies need to be precompiled.

4. In addition to the notebooks mentioned above, there is also the `simple_transition_dynamics.ipynb` notebook to solve the simple warm-up variation of the model (which does not appear in the paper) as described in the notes.

NOTE: When using the notebooks for the first time, it will be very slow as the package and its dependencies are all compiled.

README for code associated with calibration, plotting of output, and robustness results of Equilibrium Technology Diffusion, Trade, and Growth by Perla, Tonetti, and Waugh (AER 2020)

All in matlab or python with the associated jupyter notebooks.

Create Calibration Moments

The notebook `../../section_7-1.ipynb` pulls data and computes the empirical moments used in calibration. Most data in that notebook is pulled directly from public online sources like FRED or the BLS. Some data is from the Synthetic Longitudinal Business Database.

- See **`/src/calibration/SynLBD`** for the Stata code used to construct the SynLBD-derived moments, output log files, and instructions for how to access the SynLBD.

Data used in the paper was current from these sources as of April 5, 2020. The notebook `../../section_7-1.ipynb` outputs the following `.csv` files to the `src/calibration/data/` folder, which are then used as inputs into the calibration code:

- `data/growth_and_r_moments.csv` are the measured productivity growth and real interest generate
- `data/firm_moments.csv` are the firm transition dynamic moments
- `data/bejk_moments.csv` are the BEJK exporter and relative size moments
- `data/trade_moments.csv` is the trade share moment
- `data/entry_moment.csv` is the entry moment

Master File

The master file `master_calibration.m` produces all calibration and robustness results. In MATLAB, just run

```
>> master_calibration
```

This file runs all other files, prints the results, and generates the appropriate output files used as input when creating figures and numbers that appear in the paper.

Structure of Main Calibration File `calibrate_wrap.m`

The file `calibrate_wrap.m` calibrates the model using code that computes the BGP (balanced growth path) equilibrium. Transition dynamics are computed in Julia and described here.

The file `calibrate_wrap.m` does the following:

1. Reads in the moments that are generated from the python notebook described above in create calibration moments

2. Minimizes the distance between the data moments and model moments. The code that computes the model moments via simulation is organized in the following way:

- `calibrate_growth.m` organizes parameters, passes them to the function to retrieve model moments, then constructs the objective function the calibration is minimizing. The moments are stacked in the following order:
  - g, lambda_{ii}, fraction of exporters, size of exporters vs non exporters, and then the quantile moments (note how these are passed through) into vector form
  - Then line 47 describes the objective function used. This is just the standard GMM quadratic objective with an identity weighting matrix
- `compute_growth_fun_cal.m` takes in parameters, computes the BGP equilibrium, constructs the moments from simulated data, and calculates welfare. The following functions perform these operations:
- `eq_functions/calculate_equilibrium.m` computes the BGP equilibrium
- `eq_functions/selection_gbm_constant_Theta_functions.m` are the equilibrium relationships that map to equations in the paper
- `static_firm_moments.m` Computes the BEJK moments, fraction of exporters, and the relative size of domestic shipments for exporters vs non-exporters
- `markov_chain/generate_transition_matrix.m` constructs the transition matrix across productivity states. **Note** that the code `markov_chain/generate_stencils.m` is required to generate the grid. This is called before the calibration routine since it is kept constant
- `markov_chain/coarsen_to_quintiles_four.m` maps the transition matrix into a four by four matrix over a 5 year horizon to match up with data moments

3. Given the calibration results, the following output is created:

- A file `/parameters/calibration_params.csv` is created with the calibrated parameters
- An associated `.mat` file `cal_params.mat` is generated for use within matlab. The variable `new_cal` has the values in the in the following order: d, theta, kappa, 1/chi, mu, upsilon, sigma
- Comparisons of BGP equilibria associated with different parameter values are presented. To compute BGP results for different parameter values the code `compute_growth_fun_cal` is called with the appropriately changed parameter values

---

Robustness Exercises

In the paper Figures 6 and 7 display how changes to the GBM parameters μ and υ and to the BGP entry rate δ affect key outcomes.

- **Changes in δ** The code that creates the underlying data for Figure 7 is:
    - `robust_no_recalibrate_delta.m`. This reads in `cal_params.mat` generated from the calibration routine and then works through a several loops. First it changes χ by a scale value, then for the given χ it changes delta. The main file it calls is `compute_growth_fun_cal.m`. The output are the following `.mat` files:
    - `output/robust/delta/param_values_delta_xx` where `xx` is the scale value used for the given run of different δs. Each row shows the growth rate, baseline parameter values, then μ, υ, σ, and δ for that given run
    - `output/robust/delta/norecalibrate_values_delta_xx` where `xx` is the scale value for the given run. Each row shows the growth rate, counterfactual growth rate, difference, welfare gain, and the associated δ
- **Changes in μ and υ** The code that creates the underlying data for Figure 6 is:
    - `robust_no_recalibrate_gbm.m`. This reads in `cal_params.mat` generated from the calibration routine and then works through a several loops. First it changes χ by a scale value, then for the given χ it changes μ and υ by a common scale value. The main file it calls is `compute_growth_fun_cal.m`. The output are the following `.mat` files:
    - `output/robust/gbm/param_values_gbm_xx.mat` where `xx` is the scale value used for the given run of different δs. Each row shows the growth rate, baseline parameter values, then μ, υ, σ, and δ for that given run
    - `output/robust/gbm/norecalibrate_values_gbm_xx.mat` where `xx` is the scale value for the given run. Each row shows the growth rate, counterfactual growth rate, difference, welfare gain, and the associated δ
    - `/parameters/calibration_chi_xx.csv`, where `xx` is the value for χ at which the code is run. This file computes the parameter values which, for the given χ, result in a growth rate which is closest to the targeted baseline growth rate of 0.79.
- **Robustness to changes in scaling parameter ζ** This code is exactly the same as `calibrate_wrap.m` with the only difference being that alternative ζs are fixed and a loop over different calibrations is performed for the different ζs. The output is:
    - `/parameters/calibration_zeta_xx.csv` where `xx` is the ζ value for the given run and each `.mat` files contains the parameter values
- **Comparison to Sampson (2016)** This code recalibrates the model to target moments related to Sampson (2016 QJE), in part by setting GBM parameters to zero. The code `calibrate_wrap_sampson.m` performs the calibration. The underlying code and structure is exactly the same as in the main calibration code `calibrate_wrap.m`.
    - `/parameters/calibration_sampson.csv` reports the resulting parameter values

- **No Firm Dynamics** This code recalibrates the model with GBM parameters set to zero. The code `calibrate_wrap_no_firm_dynamics.m` performs the calibration. The underlying code and structure is exactly the same as in the main calibration code `calibrate_wrap.m`.
    - `/parameters/calibration_no_firm_dynamics.csv` reports the resulting parameter values
- **Comparison to Atkeson and Burstein (2010)** The base of this code was downloaded from Ariel Burstein's website at: http://www.econ.ucla.edu/arielb/innovcodes.zip. Starting from that code we set the discount rate, elasticity of substitution, the exit rate of firms, and the aggregate import share to be the same as in our baseline calibration. The code, located in `/src/calibration/ABmodel/Master_ptw.m`, runs the routine and returns the welfare gains from the same reduction in trade costs as studied in the baseline exercise of PTW, inclusive of the transition path.

README for code associated with solving the transition dynamics and steady state of of Equilibrium Technology Diffusion, Trade, and Growth by Perla, Tonetti, and Waugh (AER 2020)

Julia Source

The files in this package are installed and executed through installation in the top-level README.md. The files are:

- `params.jl` organizes parameters, settings, and initial conditions.
  - All of these may be swapped out using the named tuples.
  - To speed up loading pre-solved versions of the model, we store a cache in the `/data` folder where the name is defined by `model_cachename(...)` function in this file. The cachename is calculated by hashing all parameters and settings.
  - `load_parameters(..)` takes a CSV file with calibrated parameters (generated from the `/src/calibration` etc.) and creates the necessary structure for the Julia files.
- `static.jl` directly maps equations from the paper for the static equilibrium calculations from the parameters and intermediate values.
- `stationary.jl` calculates the stationary equilibrium and associated welfare analysis.
  - In particular `stationary_algebraic(parameters, settings)` solves the model as the system of equations specified in the main paper, and using the `static.jl` functions
  - `stationary_numerical(parameters, settings)` solves for the equilibrium using upwind-finite difference methods for the ODEs rather than solving as a system of equations. This is primarily used as the initial condition for the transition dynamics (which require ODEs).
  - `steady_state_from_g` is used by the `total_derivative` the welfare analysis
- `dynamic.jl` calculates the transition dynamics of the equilibrium between the two steady-states
  - The main entry-point is `solve_transition(parameters, settings)` which calculates the two steady states for the system of DAEs.
  - That, in turn iterates on the stock of varieties, `Ω` and solves the system of DAEs conditional on that sequence with `solve_dynamics(...)`. Convergence occurs when the entry residual is minimized, while the adoption decision is encapsulated in the DAE.
  - Finally, `prepare_results` generates a dataframe from the results.

How to access the SLBD

Overall information on the SLBD is here with the link to the application here:

- Apply for SLBD access.

Once this process is approved. Below are the instructions to log into the server:

Cornell's instructions on how to get setup: http://www.vrdc.cornell.edu/sds/first-login/

Code book for the file: http://www2.ncrn.cornell.edu/ced2ar-web/codebooks/synlbd

How to get setup.

- Install the `nomachine` application. This is found here https://www2.vrdc.cornell.edu/news/synthetic-data-server/step-3-setting-up-access-to-sds/
- You need to downlad a configuration file. This configuration file can be found at the link above. Download this to your local machine storage. When you run the nomachine program, import this configuration. Once you do that, a login will appear asking for a user id and password. Use those obtained from above instructions.

The programs and how to access them.

The file directory is organized in the following way: the path `rdcprojects/tr/tr00612/programs/users` will take you to folder that contains folders for each user. You want to access Michael E. Waugh's folder, which is under username `spec676`. Opening this folder will display the `PTW_AER` subfolder.

In `rdcprojects/tr/tr00612/programs/users/spec676/PTW_AER` is the code that constructs the empirical firm dynamics moments from the SLBD. It contains the following STATA `.do` files.

- `trmatrix_g20_loop.do` computes the transition matrix moments. It pulls the appropriate SLBD files and then constructs the transition matrix for each year. It reports the time averaged cells for each element of the transition matrix. It uses the file `quartile.do`. The output is `temp_trmatrix_g20.dta` which contains the transition matrix for each year in the SLBD.
- `quartile_code.do` computes the quartile positions of establishments in each year.
- `entry_loop.do` computes the entry rate moment. It pulls the SLBD files, computes the fraction of employment that is associated with entrants, then averages this across all years.