What Happened to U.S. Business Dynamism?
Readme for the numerical solution and exercises[*]

This file explains the Julia code that is used to compute the stationary BGP and the transition equilibrium of the model, calibrate the parameters, and conduct further exercises. All functions and other related scripts are divided into subfiles according to their objectives so that it is easier to categorize and read them. Functions to calibrate model parameters are also outlined below. In order to replicate the results in the paper such as figures, tables and moments, it is recommended to begin with the second part titled "Part 2 - Replication" (see Step 4 of "main.jl" below).

1 Structure of "main.jl"

Julia file named "main.jl" is the script that contains all the workflow. Its structure is as follows.

1. It imports necessary Julia packages at the beginning. These are:

   Parameters.jl: For defining parameters as structs with keywords.

   NLsolve.jl: Nonlinear solver "nlsolve" function is imported when firm market revenue shares are numerically derived as a result of Bertrand price competition.

   StatsBase.jl: This package provides necessary functions for sector simulations.

   Plots.jl: Plotting package to replicate figures in the paper.

   Optim.jl: An optimization package which provides the function "optimize" with Nelder-Mead algorithm, when we calibrate both initial BGP and transition equilibria of the model.

2. It includes several other Julia files that contain equilibrium solvers and other related functions. These files and their content are explained below.

3. Part 1 deals with the calibration of initial BGP and transition. Functions used in this part are defined under related script files, and they are discussed below.

4. Part 2 focuses on the replication of the paper. Exact parameter values used in the paper are provided in this part. After solving for initial BGP and transition equilibria, figures 3, 4, 5,

[*]We thank Furkan Kilic for preparing this document.

6, 7, and D.4 in the paper are replicated. Similarly, tables 3, 6, 7, and D1 in the paper are also reproduced.

2 File "parameters.jl"

This file defines parameter structures.

1. Struct "StrParam": Structural parameters of the model
2. Struct "TransParam": This object governs how structural parameters of the model evolved over time. In particular, 2015 value of a parameter is obtained by multiplying 1980 value of the parameter with the corresponding value stored in TransParam objects.
3. Struct "NumParam": These objects store numerical parameters used in iterations, simulations, and other related calculations.

3 File "fundamentals.jl"

This file contains several functions to solve for static Bertrand equilibrium of sectors in terms of market revenue shares of firms.

1. Function "Shares": It solves for revenue shares of leaders, followers, and neck-and-neck firms by using a nonlinear solver.
2. Function "Markups": Given market revenue shares, markups are returned.
3. Function "TimeVaryingSharesMarkups": Above functions assume constant underlying parameters in calculating shares and markups. In the benchmark model, parameters that govern shares and markups are indeed constant. However, in some robustness checks, these parameters do vary over time. When this is the case, the function "TimeVarying-SharesMarkups" must be used.

4 File "bgp_solver.jl"

This file starts with the definition of a stationary BGP equilibrium and the endogenous variables. These variables (vectors and floats) can be stored in a Julia struct called BGP. Then, several solver functions follow. These functions can be grouped as below.

4.1 Struct "BGP"

1. "p": Structural parameters used (StrParam type).
2. "numericp": Numerical parameters used in iterations, simulations and discretizations (NumParam type).

3. "ω": Equilibrium labor share, or equivalently normalized wage rate. The most outer loop of the solver runs over possible ω values.

4. "μ and μnn": μ is a vector that stores density of unleveled sectors with gap $m > 0$. Its length is $\bar{m}$ defined under "numericp". An element of this vector indexed by i corresponds to the density of sectors with $m = i$. Similarly, μnn is the density of neck-and-neck (leveled) sectors.

5. "$xl, xf, xnn, xe, xenn$": Vectors or single numbers for equilibrium innovation rates. l stands for leaders, f stands for followers, nn stands for neck-and-neck firms, e stands for entrants in unleveled sectors, enn stands for entrants in neck-and-neck sectors.

6. "vl, vf, vnn": Equilibrium value functions. Similar convention as above for leaders, followers, and neck-and-neck firms.

7. "sl, sf, snn": Market revenue shares of firms.

8. "zl, zf, znn": Markups of firms.

Note that each element of a vector represents the value under that particular productivity gap of the sector. For example, the equilibrium innovation rate of a leader firm in a sector where the productivity gap is $m > 0$ can be accessed as "bgp.xl[m]". Or similarly, the density of such sectors would be "bgp.μ[m]".

4.2 Remaining functions

1. First group calculates innovation rates given value functions. These are "InnL!", "InnF!", "InnNN", "InnE!", "InnENN".

2. After a proper discretization of time, functions in the second group iterate Bellman equations a single time in the sense that they solve for current value functions (vectors without primes) given a future value function and the associated innovation rates (vectors with primes). This is basically backward iteration. These functions include "LeaderBellman!", "FollowerBellman!", and "NNBellman".

3. Function "ValueIteration!": This function performs a standard value function iteration algorithm given a normalized wage rate ω. Beginning with guesses for value functions (vl, vf, vnn), which are typically zero vectors, it iterates until value functions are close enough under some tolerance "valuetol". As a convention, vectors without primes correspond objects "today", while the ones with primes stand for objects "tomorrow".

4. Function "DistributionIteration!": After we converged on values and corresponding innovation rates, we can now iterate sector distribution forward in time. Again given a normalized wage rate ω, and innovation rates, this function solves for corresponding firm distribution μ and μnn, beginning from a guess which is typically taken as a uniform distribution.

These functions do not return new vectors every time they are called. Instead, they overwrite their arguments to preserve memory and efficiency. Note that objects of type Float64 cannot be overwritten. They have to be instead returned each time.

4.3 Function "BgpSolver"

This function solves for the stationary BGP equilibrium of the model given parameters "p" and other numerical parameters "numericp", such as dt or tolerances. It returns a BGP object. Steps are outlined as below.

1. First solve for revenue shares and markups of firms characterized by productivity gap m, and firm status, i.e. whether the firm is a leader, follower or neck-and-neck.
2. Preallocate necessary objects of a BGP in order to fill them during loops and iterations.
3. Guess a normalized wage rate ω as 0.9 as a candidate for equilibrium. Outer loop starts at this point.
4. Given ω, solve for value functions and innovation rates by a standard backward value function iteration procedure using the function "ValueIteration!" described above.
5. Given innovation rates, solve for the firm distribution by a forward iteration.
6. Finally update the wage rate ω. In particular, under just solved innovation rates and firm distribution, calculate implied wage rate $\omega\prime$ in order to clear the labor market.
7. If $\omega\prime$ and ω are close enough, exit the outer loop and return the BGP equilibrium just calculated by the most recent iteration. If they are not close enough, update ω and begin the outer loop again from step 4 with this updated wage rate guess.

5 File "transition_solver.jl"

This file contains the definition of a struct called "Transition" in order to store the transition equilibrium objects, together with the main solver function "TransitionSolver". When solving the transition equilibrium, continuous time is discretized with respect to predefined value of dt. Each year is divided into periods, and the number of periods in a year is $1/dt$. The transition starting from an initial BGP is computed over a long period of time that is enough to ensure the economy converges to the new steady before the terminal point of the transition is reached. In the calculations, this duration is chosen to be 150 years.

5.1 Struct "Transition"

Ingredients of this struct is the same as the struct "BGP" described above, except that there is an extra dimension for discretized time periods. Therefore vectors become matrices, and numbers become vectors. In matrices, columns represent time, while rows are productivity gaps. For example, $transition.vl[m,t]$ is equal to the value of a leader firm with productivity gap m in period t. For vectors such as "vnn", each element corresponds to the value at that time. For instance, $transition.vnn[t]$ is the value of neck-and-neck firms in period t.

5.2 Function "TransitionSolver"

This function solves for the transition given an initial BGP equilibrium called "initial", transition parameters "transitionp", and numerical parameters "numericp". Initial BGP "initial" is required, because the productivity gap distribution is an aggregate state of the model, and its initial value must be provided. It is assumed that "initial" represents the stationary equilibrium before 1980, and economy enters a transition period after that.

Steps:

1. Revenue shares and markups do not change in the benchmark model over the transition, since structural parameters governing them, λ, β, ψ, remain constant. These objects are read from initial BGP "initial".
2. A sequence of parameters is created. In particular, "sequencep" is a vector of "StrParam" objects that define the transition of the model over time.
3. The stationary distribution in distant future, "terminal", is found by calling "BgpSolver" function.
4. Matrices and vectors are preallocated for endogenous objects of transition equilibrium.
5. Unlike the BGP equilibrium, which is characterized by a single constant wage rate, transition equilibrium is characterized by a sequence of normalized wage rates for the whole transition period. In particular, a vector ω of guessed wage path stores normalized wage rates over time such that $\omega[t]$ is the normalized wage rate in period t. The guess is simply the normalized wage rate in initial BGP.
6. Given the vector of normalized wage rates ω, the algorithm iterates value functions only one time for the latest period $t = tperiods$ starting from the value functions of the terminal BGP calculated above. Implied innovation rates for the latest period are also found.
7. Between periods $t = tperiods - 1$ and $t = 1$, a backward iteration of value functions is performed given the wage rates. In each time period, the implied innovation rates are also stored.
8. Next, a forward iteration of productivity gap distribution over time is performed starting from the initial condition, which is given by the sector distribution in initial BGP.
9. The distribution of productivity gaps and innovation rates over time help obtain the sequence of wage rates that clear the labor market, denoted by "$\omega\prime$".

10. If vectors ω and $\omega\prime$ are close enough, the outer loop ends and returns the "Transition" object. If not, the initial guess ω is updated, the algorithm goes back to step 6.

6 File "simulation.jl"

Not all moments can be calculated analytically. Therefore, a large group of firms is simulated over both BGP and transition equilibria in order to obtain the full set of moments, including the employment share of young firms, firm growth rate dispersion, etc. Main simulation functions (wrappers) are called "InitialBgpSimulation!", "TransitionSimulation!", and "TerminalBgpSimulation!". In order to keep track of entrants and exiters over time, each firm is assigned a unique id number. The simulated number of sectors is 10,000. Therefore, the total number of firms is 20,000. Simulating firms with their productivity levels in every discretized period yields a panel dataset that is basically the snapshot of active firms with their current states at the end of each year. Firm ages can be calculated from this panel dataset with the help of unique firm ids assigned at the time of entry.

For every period and every product line, an event is randomly assigned based on probabilities implied by equilibrium innovation rates. These events include leader innovation, follower innovation with or without drastic innovation, entry with or without drastic innovation, knowledge diffusion, and a final event in which nothing happens. Results of these simulations are stored in a matrix called "sectordata". A single sector stored in element $sectordata[i,t]$ is the one with $id = 1, \ldots, 10,000$, in period t. After constructing "sectordata", a matrix called "panel" is constructed with each column representing a variable, while rows are firm observations. This is an annual panel data which is the snapshot of the economy at the end of each year. Columns are year, firm id, productivity step, labor employment, firm age, and productivity level, respectively.

7 File "moments.jl"

This file contains several functions to calculate model moments. Importantly the function "Moments" takes either a "BGP" or a "Transition" object as its argument, depending on what type of equilibrium is handled. A "BgpMoments" or a "TransitionMoments" object is then returned. Transition moments are not stationary, and importantly they are annual. Therefore, they are represented as vectors whose single elements correspond a year over transition.

8 File "calibration.jl"

This file contains functions related to the calibration/estimation procedure. Importantly, "BgpCalibration" and "TransitionCalibration" are two functions that run the whole calibration procedure.

8.1 Function "BgpCalibration"

Numerical parameters "numericp", moment targets "targets", and initial parameter guesses for the minimization routine "guess" are arguments of this function. Firstly, parameter ranges are created. These are lower and upper bounds for the parameter space over which the minimization searches for a local minimum. Secondly, moment weights are created and attached to each target moments. Finally, minimization function optimize is called. For this routine, Nelder-Mead algorithm in the Optim.jl package is adopted.

8.2 Function "TransitionCalibration"

The idea of this function is similar to the previous one. This function takes the initial BGP as an argument. For the sake of consistency across the initial BGP and transition simulations, simulated firms in the last period and last year of the initial BGP along with the latest entrants are also provided as function arguments.

After defining parameter ranges and moment weights as before, the same minimization routine is adopted. In every round, the transition equilibrium is solved, simulated, and necessary moments are calculated. These moments are then compared to their target values, and a difference measure between them is returned. The function "optimize" from Optim.jl package minimizes this returned value.

9 File "exercises.jl"

This file contains several functions to perform different exercises of Section 5, 6, and 7 together with a sensitivity analysis in initial BGP, and alternative mechanisms of Section 8.