GJLS_readme_model

January 31, 2021

1 Replicating the model results

This notebook describes how to simulate the overlapping-generations model from the paper "Understanding the New Normal: The Role of Demographics" by Etienne Gagnon, David Lopez-Salido, and Benjamin Johannsen. The paper was accepted for publication in IMF Economic Review in January 2021.

Detailed descriptions of the model and solution method are contained in the technical appendix to the paper. The materials below provide additional information on the supporting programs.

1.1 Programming languages

All model files and data cleaning/organizing programs are written in the Julia programming language using the platform Jupyter.

1.1.1 Julia

Julia is a programming language developed for high performance dynamic programming and numerical computing. Its syntax is similar to those of Python, R, and Matlab whereas its performance can approach that of C in some applications. Julia is free and open source. The language can be installed at Julialang.org. Our code is written using version 1.5.

1.1.2 Jupyter

Jupyter provides a framework for creating interactive code as well as allowing seamless integration of formatted text with computational code and results. Unlike a static presentation using a program such as LaTeX, Jupyter's "notebooks" can include both text and code in selected programming languages. These replication materials are created in Jupyter notebooks. To download and install Jupyter, please look here. For a complete list of languages that can be used with Jupyter notebooks, all current kernels are listed at this site. Jupyter is free and open source.

1.2 Key model files

1.2.1 `calibeta.jl`

This is the main function that launches the model simulations. It can take arguments that adjust the model and simulation parameters as desired (see descriptions of our calls at the end of this notebook).

The main steps of this function are:

1. Loads the requisite functions and data types.
2. Loads the demographic and technology data through call of `read_data.jl`.
3. Call `set_targets.jl` to create the target mean r_t value in the 1980s.
4. Set the parameters of the model and simulation through calls of `set_pars.jl` and `set_simpars.jl`.
5. Initiate the search for β and the associated solution through call of `calibeta_bisection.jl` and save the data.
6. Save the simulation results.

1.2.2 `calibeta_bisection.jl`

This file sets up the bisection search for the value of β such that the equilibrium interest rate path in the 1980s matches that of Johannsen and Mertens (2017). A low and a high value of β must be provided when iterating over beta. If the simulation takes β as given, then only the low value needs to be provided. Each guess of β must be accompanied with two pairs of low and high guesses of R_t in the balanced-growth equilibriums consistent with the demographic variables at the beginning and end of the simulation period.

The main steps of this function are:

1. Read the model and simulation parameters.
2. Given the low guess of β, call `GE_search.jl` to find the associated general equilibrium solution and then calculate the mean rate of interest targeted.
3. Proceed similarly for the high guess of β.
4. With the bisection of β seeded, iteratively compute a solution for the mid-point of the range of β values until the targeted mean interest rate has converged.

1.2.3 GE_search.jl

1. Generate the balanced-growth solution for the turn of 1900, which is used to seed the simulation under both perfect foresight and backward-looking expectations.
   - To this end, the program first computes a balanced-growth population and dependency structure through a call of `Compute_population.jl` imposing that the 1900:Q1 demographic parameters last forever.
   - The program then uses a bisection method to search for the balanced-growth value of R^* consistent with the population distribution. In particular, for a given guess of R^*, it computes two measures of the aggregate capital stock.
     - The first measure, K^{mpk}, is the capital stock consistent with R^* and the growth rates of the labor supply and technology based on the firm's first-order condition with respect to capital.
     - The second measure, K^{hh}, is the aggregate capital stock based on aggregating individual household decisions given the population.
     - When seeding the bisection, the low guess of R^* should be such that $K^{mpk} > K^* > K^{hh}$. Intuitively, a low value of R^* encourages capital accumulation by firms but discourages capital accumulation by households. Conversely, the high guess of R^* should have the reverse ordering. If the guesses are both too low or too high, then the values of `pars.seedR1900` should be adjusted accordingly.
   - The balanced-growth solution given by the low and high guesses of R^* is computed through a call of the `BG_search_bisection.jl` function.

2. Calculate the population and dependency structures in the actual population via a call to `Compute_population.jl`.
3. The program next computes the dynamic solution conditional on the capital holdings and interest rate carried into the initial period. This step differs between the solutions under perfect foresight and backwards-looking expectations.
    - Perfect foresight: Pass demographic and technology data to the function `GE_search_solution.jl`.
    - Backward-looking expectations:
        - First, create a set of counterfactual demographic expectations from the point of view of the first period in the simulation. For a rolling window of `pars.hhexpectations` periods, the algorithm computes the mean demographic variables for the current and past `pars.hhexpectations`−1 periods. 1900:Q1 values are used for periods in the window that fall before the start of the simulation. The actual demographic variables are always used for the current period to ensure that the solution to the household and firm problems are consistent with each other.
        - Second, compute the general-quilibrium solution under the counterfactual demographic variables through a call of `GE_search_solution.jl`.
        - Third, save the aggregate solution for the current period as well as the capital holdings carried into the next period. Then move the start of the simulation by one period and repeat these three steps.
        - Continue to iterate on these steps until a solution conditional on backward-looking expectations has been computed from the point of view of every period to be displayed. Because the general-equilibrium solution is computed hundreds of times, it takes up to a day to obtain a solution.

1.2.4 `GE_search_solution.jl`

This function computes the general-equilibrium solution given demographic data and initial conditions for the distribution of household asset holdings and the previous-period interest rate. It starts with a guess of the aggregate object paths and then gradually adjusts those paths in the direction that makes them consistent with the aggregation of household decisions.

The main steps of this function are:

1. Compute the population and dependency structure consistent with the assumed path of the demographic variables through a call of `Compute_population.jl`.
2. Compute the balanced-growth solution for the last period in the simulation.
3. Use the balanced-growth solutions for the first and last periods of the simulation to initialize the path of aggregate variables.
4. Given the path of aggregate variables, solve the household problem for each period in the simulation and aggregate the decisions to derive new paths.
5. Measure the distance between the new paths and the old paths. Stop if the difference meets the numerical convergence criterion. Otherwise update the path of aggregate variables in the direction of the new paths. Keep updating until numerical convergence.

In some instances, it is found that the update toward the new paths should be slow to ensure convergenge to the general-equilibrium solution.

1.2.5 `Compute_population.jl`

This function computes the resident population, measured at the end of each period, between 1900:Q1 and 2399:Q4 under specific demographic assumptions. It takes the baseline demographic information for the 1900:Q1-2099:Q4 period contained in the structure `data` and, where relevant, extends those data through 2399:Q4. In this extension, the variables guiding the computation of the population in 2099:Q4 are carried foward for all subsequent periods, so that population dynamics are consistent with the economy reaching a balanced growth state.

Four key parameters determine how the population is recursively created:

- `use_birth`: If true, the function grows the population through exogenous births. Otherwise, it grows the population through exogenous fertility rates.
- `fixed_birth`: If true and `use_birth==true`, then live births are assumed to grow at the preset rate `pars.n`.
- `migration`: If true, then the net migration values in `data.net_migration_Q` are added to the population at the beginning of every period. Otherwise, the function sets net migration to zero for all ages and periods.
- `extra_birthgr`: If true, this option allows the user to add a fixed number of extra basis points to the exogenous growth rate of live births to all periods.

During execution, the function creates the following variables:

- `data.fitted_age_marriage`: Median age difference in years between men and women at time of first marriage (vector, 2000x1).
- `data.share_births_mothers`: Share of births in period accruing to mothers aged 14 to 49 years of age (matrix, 144x2000).
- `data.net_migration_Q`: Net number of migrants of a given age at beginning of period (matrix, 480x2000).
- `data.births_interpolated`: Annualized number of births at beginning of period (vector, 2000x1).
- `data.death_rate`: Death rate affecting resident population and net migrants alive at beginning of period (matrix, 480x2000).
- `Population`: Population by age alive at end of period (matrix, 480x2000).
- `Parent_child`: Number of kids by age of their parent at end of period (matrix, 480x2000).
- `Dependents`: Number of kids of a given age dependent on parent of a given age at end of period (matrix, 480x2000).
- `Parent_newborns`: Number of newborns per parent of a given age at end of period (matrix, 480x2000).
- `Parent_fertility`: Fertility rate by age of parent, measured at end of period (matrix, 480x2000).

When using live births to grow the population, the function returns the (end-of-period) fertility rates consistent with the population. Conversely, when using fertility rates, the function returns a birth series. Also, the function leaves the original 'data' series unchanged because model simulations may use these demographic data in several manners.

The function provides the option of fixing the fertility rates, the mortality rates, or both to the values observed in the period immediately before a specified date. For example, by setting `pars.per_fix_frate=320`, the population computations from 1980:Q1 onward would be based on the assumption that the fertility rates are unchanged at their 1979:Q4 values. Similarly, setting

`pars.per_fix_grate=320` would leave the mortality rates constant from 1980:Q1 onward.

When computing a counterfactual population under backward-looking expectations, the `data` structure input should contain the demographic variables expected by households. To ensure that the counterfactual population aligns with the actual population in the initial period, the demographic variables for that period should match the actual ones. If that is not the case, then the aggregation of the household and firm decisions would not align with the general equilibrium values for that initial period.

1.2.6 `PE_family_growth_minc`

This file, called by `GE_search_solution.jl`, computes the solution to each household's partial-equilibrium life-cycle problem taking as given the path of all aggregate variables and demographic data. This file would need to be modified if the structure of preferences, bequests, or production were to change.

1.2.7 `PE_family_growth_minc_SS`

This file computes the ergodic distribution of the population and the aggregate objects that are consistent with a given rate of population growth and the individual household and firm decisions that would be observed for a given value of R^*. These partial-equilibrium objects are used in the search for the balanced growth equilibrim of the model in the function `BG_search_bisection.jl`. This file would need to be modified if the structure of preferences, bequests, or production were to change.

1.2.8 `GJLS_model_IRF.ipynb` and `GJLS_plot_IRF.ipynb`

These notebooks create impulse reponses to shocks to fertility and life expectancy and plot the results, respectively. The baseline simulation of the model must have been run before executing these files to provide a no-shock baseline.

1.3 Other model files

1.3.1 `functionincludes.jl`

This file, called by `calibeta.jl`, loads all the functions that are used in the performance of the simulations.

1.3.2 `late_data_start.jl`

This function, called by `calibeta.jl`, shifts the demographic data so that the simulation starts in a period later than 1900:Q1, the default for all simulations in the paper. A later start allows us to check the sensitivity of our findings to using later initialization periods, as other authors such as Eggerston et al. (2018) do. The function requires estimates of the population in period immediately before the start of the simulation, which we read from the output of `FertilityRates.pynb`. This function has not been adapted to update the family dependency structure. Therefore, it should not be called with versions of the model in which children enter the utility function of households.

1.3.3 `matrix_prep.jl`

This function is called by `GE_search.jl` to create the household's expectations of demographic variables under backward-looking expectations. The function can handle both vectors and matrices. It also fills in missing historical data when the window to form expectations includes periods from before 1900.

1.3.4 `plotting_setup.jl`

After running all of the model simulations, this file can be run to export the Julia data output to CSV files that can then be read by R programs to replicate the related figures in the paper.

1.3.5 `read_data.jl`

This function reads the demographic and technology time-series data for the 1900-2099 period. These data were created by the Jupyter notebook `GJLS_readme_data.ipynb`. The data should be located in the folder `../Data/`. The series are: * `marriageAgeDiff_Q.csv`: The median age difference between men and women at first marriage. * `interp_death_rate_1900_2220_Q.csv`: Estimate of the annualized quarterly death rates by age and birth cohort. * `births_Q_annualized.csv`: Estimate of the annualized quarterly number of live births. * `population_1899_Q.csv`: Estimate of the population by quarter of age at the end of 1899. * `share_births_mothers_Q.csv`: Share of live births in each quarterly period that accrue to mothers of a given age. * `netmigration_Q.csv`: Estimate of the net migration by quarterly period and age of migrant that is needed to reconcile our population estimates with our live births and mortality rate series. * `parent_child_1899_Q.csv`: Estimate of the family structure at the end of 1899 by the age of the parents and children. * `dependents_1899_Q.csv`: Estimate of the parent-children dependency structure at the end of 1899 by the age of the responsible adults and children. * `epr_trend_1900_2100_Q.csv`: Estimate of the employment rates by age and period.

1.3.6 `reset_initial_conditions.jl`

This function is called by `GE_search.jl` in the computation of a general-equilibrium solution under backward-looking expectations. As a precaution, it assigns missing values to demographic data that should not play a role in the computation of a counterfactual population under backward-looking expectations.

1.3.7 `run_model.jl`

This function contains the main calls to replicate our findings. These calls can also be found at the bottom of this notebook, along with explanations. To run this function directly from the Julia REPL, make sure that to uncomment only the simulations of interest. The simulation can be executed through the command `include("run_model.jl")`.

1.3.8 `set_pars.jl`

This function is called by `calibeta.jl` to create a `pars_t` object that holds model and and simulation parameters values. The call of this function assigns several values by default, which can be overridden in the call to `run_model.jl` or if the values are altered in other parts of the code. Therefore, the user should check that the simulations proceed under the desired parameters.

1.3.9 set_simpars.jl

Similar to set_pars.jl, this function called by calibeta.jl creates a sims_t object holding simulation parameter values.

1.3.10 set_targets.jl

This function reads the average long-run estimates of the neutral rate in the 1980s computed by Johannsen and Mertens (2018).

1.3.11 typedefs.jl

For convenience and efficiency, we define five data types that encompass the variables used in the model. The definitions are made in calibeta.jl. The data types are: * data_t: Structure containing the time-series demographic and technology information. * calibeta_t: Structure containing the targets of the calibration of β. * simpars_t: Structure containing many of the parameters controlling the model simulations. * pars_t: Structure containing the main parameters of the model, along with a few controlling the simulations. * Track_SS_t: Structure that tracks the information allowing us to assess convergence to the β value consistent with the Johannsen and Mertens (2018) estimate of the mean long-run equilibrium rate in the 1980s.

2 Launching the simulations

The set of instructions below show the commands that we run to produce the key results in the paper. The solution to the version of the model with backward-looking expectations is especially time consuming. It is preferable to launch each such specification separately.

[ ]:
```
# Baseline model (perfect foresight, no technology growth, no kids)
# betaL  R[1900] =0.036708 R[2400] =0.026709
# betaH  R[1900] =0.026197 R[2400] =0.015543
# beta* = 0.9971010742187502
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(; simsuf="_base", nu = 2.0, seedR1900L = [0.030;0.042], seedR1900H =␣
↪[0.022;0.030],
          seedR2400L = [0.021;0.030],seedR2400H = [0.013;0.018],␣
↪DS_AGG_maxiter=200, DS_AGG_adjfactor=0.25)
```

[ ]:
```
# Baseline model with historical TFP growth (Fernald)
# betaL  R[1900] =0.050807 R[2400] =0.034434
# betaH  R[1900] =0.035464 R[2400] =0.017737

using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(simsuf="_TFP", nu = 2,
          betaL = 0.992, betaH = 1.006, seedR1900L = [0.049;0.053],seedR1900H␣
↪= [0.034;0.038],
          seedR2400L = [0.033;0.036], seedR2400H = [0.016;0.020],
```

```
          dZ = log.(exp.(readdlm("../Data/CleanData/dlogtfp_Q.csv", ',')[:,1]).
↪^(1.0/0.65)),
          DS_AGG_adjfactor=0.2)
```

[ ]:
```
# Baseline model with historical human quality growth (Fernald)
# betaL  R[1900] =0.037432 R[2400] =0.027334
# betaH  R[1900] =0.023910 R[2400] =0.013771
pwd()
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(simsuf="_HQ", nu = 2, betaL = 0.992, betaH = 1.006,
            seedR1900L = [0.028;0.040],
            seedR1900H = [0.019;0.025],
            seedR2400L = [0.023;0.033],
            seedR2400H = [0.012;0.022],
            dZ = 0.65*(readdlm("../Data/CleanData/dlogLQ_Q.csv", ',')[:,1]),
            DS_AGG_adjfactor=0.25)
```

[ ]:
```
# Baseline model with historical human quality growth (Penn World Table)
# betaL  R[1900] =0.039756 R[2400] =0.029861
# betaH  R[1900] =0.025792 R[2400] =0.015081
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(simsuf="_HQPWT", nu = 2, betaL = 0.992, betaH = 1.006,
            seedR1900L = [0.038;0.041],
            seedR1900H = [0.027;0.032],
            seedR2400L = [0.023;0.030],
            seedR2400H = [0.012;0.018],
            dZ = (1.0/0.65)*(readdlm("../Data/dlogLQPWT_Q.csv", ',')[:,1]),
            DS_AGG_adjfactor=0.2)
```

[ ]:
```
# Baseline model with dependents
# betaL  R[1900] =0.037144 R[2400] =0.029661
# betaH  R[1900] =0.028229 R[2400] =0.017814
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(simsuf="_dep", nu = 2, epsilon = 0.65, eta = 0.76,
        betaL = 0.991, betaH = 1.0015,
        seedR1900L = [0.030;0.040],
        seedR1900H = [0.026;0.030],
        seedR2400L = [0.025;0.032],
        seedR2400H = [0.016;0.021],
        DS_AGG_adjfactor=0.25)
```

[ ]:
```
# Baseline model with dependents and alternative calibration of epsilon and eta
# betaL  R[1900] =0.036963 R[2400] =0.028554
# betaH  R[1900] =0.028554 R[2400] =0.017814
```

```
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(simsuf="_depalt", nu = 2, epsilon = 0.251252473522698, eta = 0.9225,
        betaL = 0.991, betaH = 1.0015,
        seedR1900L = [0.030;0.040],
        seedR1900H = [0.026;0.030],
        seedR2400L = [0.026;0.032],
        seedR2400H = [0.016;0.021],
        DS_AGG_adjfactor=0.25)
```

[ ]:
```
# Baseline model with dependents but beta of no-dependent calibration
# betaL  R[1900] =0.031836 R[2400] =0.022329
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(; simsuf="_dep_betafixed", nu = 2, betaL=0.9974086914062501,␣
 ↪beta_maxiter=0,
        epsilon = 0.65, eta = 0.76,
        seedR1900L = [0.025;0.037],
        seedR2400L = [0.020;0.030])
```

[ ]:
```
# Low elasticity in production (it helps to set DS_AGG_adjfactor=0.15)
# betaL  R[1900] =0.037637 R[2400] =0.026844
# betaH  R[1900] =0.026275 R[2400] =0.014040
# beta* =0.997137
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(simsuf="_lowrho", nu = 2,
        betaL = 0.991, betaH = 1.0015,
        seedR1900L = [0.03;0.04],
        seedR1900H = [0.021;0.03],
        seedR2400L = [0.023;0.028],
        seedR2400H = [0.012;0.016],
        rho = -1.0/3.0,
        DS_AGG_adjfactor=0.25)
```

[ ]:
```
# Counterfactual: Constant demographics from 1960 onward (all, employment only,␣
 ↪fertility only, mortality only)
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
# betaL  R[1900] =0.030357 R[2400] =0.028147
calibeta(; simsuf="_1960", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.028;0.033], seedR2400L = [0.023;0.032], per_fix_erate=240,␣
 ↪per_fix_frate=240, per_fix_grate=240)
# betaL  R[1900] =0.030357 R[2400] =0.019329
calibeta(; simsuf="_1960e", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.028;0.033], seedR2400L = [0.017;0.020], per_fix_erate=240,␣
 ↪per_fix_frate=800, per_fix_grate=1280)
```

```
# betaL  R[1900] =0.030357 R[2400] =0.022334
calibeta(; simsuf="_1960f", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.026;0.033], seedR2400L = [0.020;0.026], per_fix_erate=800,␣
 ↪per_fix_frate=240, per_fix_grate=1280)
# betaL  R[1900] =0.030357 R[2400] =0.026841
calibeta(; simsuf="_1960g", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.026;0.033], seedR2400L = [0.022;0.029], per_fix_erate=800,␣
 ↪per_fix_frate=800, per_fix_grate=240)
```

[ ]:
```
# Counterfactual: Constant demographics from 1960 onward (all, employment only,␣
 ↪fertility only, mortality only)
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
# betaL  R[1900] =0.030357 R[2400] =0.022640
calibeta(; simsuf="_1980", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.026;0.033], seedR2400L = [0.018;0.025], per_fix_erate=320,␣
 ↪per_fix_frate=320, per_fix_grate=320)
# betaL  R[1900] =0.030357 R[2400] =0.018569
calibeta(; simsuf="_1980e", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.026;0.033], seedR2400L = [0.015;0.020], per_fix_erate=320,␣
 ↪per_fix_frate=800, per_fix_grate=1280)
# betaL  R[1900] =0.030357 R[2400] =0.019513
calibeta(; simsuf="_1980f", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.026;0.033], seedR2400L = [0.015;0.021], per_fix_erate=800,␣
 ↪per_fix_frate=320, per_fix_grate=1280)
# betaL  R[1900] =0.030357 R[2400] =0.025358
calibeta(; simsuf="_1980g", nu = 2, betaL=0.9974086914062501, beta_maxiter=0,␣
 ↪seedR1900L = [0.026;0.033], seedR2400L = [0.021;0.027], per_fix_erate=800,␣
 ↪per_fix_frate=800, per_fix_grate=320, DS_AGG_maxiter=200)
```

[ ]:
```
# Backward-looking model
# betaL  R[1900] =0.029445 R[2400] =0.029438
# betaH  R[1900] = R[2400] =
using JLD, Statistics, Printf, DelimitedFiles, LinearAlgebra
include("functionincludes.jl")
calibeta(;simsuf="_bwd20", hhexpectations=20, betaL=0.9974086914062501,␣
 ↪seedR1900L = [0.028;0.031], seedR2400L = [0.028;0.037], beta_maxiter=0,␣
 ↪DS_AGG_adjfactor=0.30)
#calibeta(;simsuf="_bwd40", hhexpectations=40, betaL=0.9974086914062501,␣
 ↪seedR1900L = [0.024;0.027], seedR2400L = [0.025;0.027], beta_maxiter=0)
```

[ ]:
```
# Converting the simulations' output files from JLD to CSV files that can be␣
 ↪read in the creation of the figures
include("plotting_setup.jl")
```