Repository for "Markov-Chain Approximations for Life-Cycle Models"

This repository includes all the source code and input for generating the results in *Markov-Chain Approximations for Life-Cycle Models* (Giulio Fella, Giovanni Gallipoli, and Jutong Pan).

The code is written in **Fortran** and **R**. Most of the computations are done in Fortran. R is used for some of the statistical tasks.

Structure

Ordered in their relative importance, the list of files is:

- *main.f90* defines the main program.
- *discretization.f90* defines the subroutines for the discretization methods, including our extensions of Rouwenhorst (1995) and Tauchen (1986) methods in life-cycle models (canonical AR(1) process and AR(1) with non-normal innovations), as well as our implementation of the methods in Huggett (1996) and Storesletten et al. (2004).
- *adda_cooper.f90* defines the subroutines for our extension of Adda and Cooper (2003) method. It calls two R scripts *panel_to_markov.R* and *panel_to_markov_nmar.R* described below.
- *endogrid.f90* defines the subroutines for implementing the endogenous grid method to solve the model.
- *simulation.f90* defines the subroutines for simulating the panel of individual life cycles using the continuous or discretized income process and the policy functions solved.
- *common_tools.f90* defines a few global parameters such as π and the project path, and several small subroutines commonly used across other modules.
- *sub_grid.f90* includes the subroutine for generating various types of grid.
- *gaussquad.f90* implements Gaussian quadrature, from Christopher N. Gilbreth.
- *csv_file.f90*, *csv_file_1d.f90*, *csv_file_2d.f90* include subroutines for exporting and importing data between Fortran and local csv files, from Arjen Markus.
- Sub-directory *R* includes the following files:
  - *panel_to_markov.R* implements the Adda and Cooper method using a simulation approach, based on the paper "Nonlinear household earnings dynamics, self-insurance, and welfare," (Mariacristina De Nardi, Giulio Fella and Gonzalo Paz-Pardo) forthcoming in the Journal of the European Economic Association. It is called by the Fortran subroutine `adda_cooper` in *adda_cooper.f90*.
  - *panel_to_markov_nmar.R* is similar, but applies to the processes with non-normal innovations. It is called by the Fortran subroutine `adda_cooper_nmar` in *adda_cooper.f90*.
  - *KellySkew_CSKurt.R* is used for computing the Kelly's measure of skewness and Crow-Siddiqui kurtosis.
- Sub-directory *input* includes the following files:
  - *rho_sigma_by_age.csv* stores the time-varying conditional second moments used in Section 4.2 of the paper.

    - *Omega_calibrated.csv* is for the robustness-check exercise of using a different parameterization for Tauchen method, described in Appendix A.3.
- Sub-directory *normalized* includes another set of Fortran code. It is used for implementing the alternative quasi-exact solution of the normalized problem in the random walk case, described in Appendix A.1.
- Sub-directories *output* and *temp* are placeholders. Several files will be generated by the program and saved there. At the end of each run, the results of the run will be appended as a new row into *output/Results.csv*.

The code is commented extensively, and the naming of each subroutine indicates clearly what the subroutine is used for. One general note is that the suffix of `_nmar` indicates that subroutine is for AR(1) processes with non-normal innovations.

Preparation

A Fortran (90 or higher) compiler (we tested on Intel Fortran compiler) and R (>= 3.3) are required. In addition, Intel MKL library for Fortran is necessary. R packages *data.table* and *statar* are also used, which will be installed automatically by the scripts if you haven't installed them yet.

Before compiling and running the code, please perform the following steps to adapt it for your setup:

1. Change the following paths defined in *common_tools.f90*:
    - `projectpath`: the local path to this repo
    - `rscriptpath`: the path to Rscript.exe (e.g., "C:/Program Files/R/bin/x64/Rscript.exe")
2. Change the working directory for the R scripts. To do so, in *panel_to_markov.R*, *panel_to_markov_nmar.R* and *KellySkew_CSKurt.R*, change the following line:

    ```
    setwd("C:/Users/jpan/Documents/repos/fgp/")
    ```

    to

    ```
    setwd(projectpath)
    ```

    where `projectpath` should be the same as the path you specify `projectpath` to be in common_tools.f90 above, i.e., the local path to this repo.
3. Include the Intel MKL library for compilation of the Fortran code. If you are using an integration of Visual Studio and Intel Fortran compiler, change the setting in "Project Configuration Properties" - "Fortran" - "Libraries" - "Using Intel Math Kernel Library" to `Cluster`. If you are compiling on the command line, add the option `/Qmkl:cluster` in the compilation line.
4. Set Heap Arrays to 0 (to avoid stack overflow error). If you are using an integration of Visual Studio and Intel Fortran compiler, change the setting in "Project Configuration Properties" - "Fortran" - "Optimization" - "Heap Arrays" to `0`. If you are compiling on the command line, add the option `/heap-arrays0` in the compilation line.

Replication

Note: The paper presents a rich set of numerical results, and some of them (especially the benchmark solutions) are obtained through intensive computations. This means that for those runs, the runtime can be very long. The code is thus structured in a way that, instead of running all scenarios and solutions and generating all results in one shot (which could take days), it allows the user to specify which model setting and which method to run.

Parameters

To replicate the tables of results in the paper, you need to change the values of the following parameters in the code depending on the scenario you would like to run:

- `mode`: defined in main.f90, and accepts one of the following values
  - `0`: Income process is the canonical AR(1) process as described in Section 4.1 of the paper
  - `1`: Income process is the age-dependent AR(1) process as described in Section 4.2
  - `2`: Income process is the AR(1) process with non-normal innovations as described in Section 4.3
  - `3`: Income process is the canonical AR(1) process but parameterized for comparison with Huggett (1996), as described in Section 4.1.1
- `rho`: defined in main.f90 (on the line `rho = 0.95d0`). It is the persistence of AR(1) process used in mode 1. In the paper, we used three different values for `rho`: 0.95, 0.98, 1.0
- `rho_nmar`: defined in main.f90. It is the persistence of AR(1) with mixture of normals innovations, used in mode 3. In the paper, we used three different values for `rho_nmar`: 0.95, 0.98, 1.0

In addition, you can select the solution method and the simulation method **interactively** when running the program. The values for the following parameters are chosen by the user interactively instead of being modified in the code:

- `method`: takes one of the following values, with their corresponding solution method
  - `0`: benchmark solution as described in Section 3 of the paper
  - `1`: using Rouwenhorst (1995) method for Markov chain approximation, as described in Section 2.3
  - `2`: using Tauchen (1986) method, as described in Section 2.1
  - `3`: using Adda and Cooper (2003) method, as described in Section 2.2
  - `4`: using Huggett (1996) method, as described in Section 2.4. Applicable in mode 3 only
  - `5`: using Storesletten et al. (2004) method, as described in Section 2.4. Applicable only in mode 0 and when `rho = 1`
- `nygrid`: the number of Markov chain states (denoted by N in the paper).
- `method_sim`: specify which simulation method (described in Section 3) to use
  - `1`: Markov chain simulation
  - `2`: continuous process simulation

Replication of tables in the main text

The combinations of different values for these parameters generate different sets of results in the paper. In detail:

- **Table 1**: set `mode = 0`, varying `rho` among $\{0.95, 0.98, 1.0\}$, and select solution methods from `0` to `3`. For simulation method, select `1`.
- **Table 2**: same as Table 1 except for choosing simulation method `2`
- **Table 3**: set `mode = 3`, and select solution methods from `0` to `4`

- **Table 4**: set `mode = 0` and `rho = 1.0`, and select solution methods from `0` to `3` as well as `5`
- **Table 5**: set `mode = 1`, select solution methods from `0` to `3`, and select simulation methods `1` and `2`
- **Table 6**: set `mode = 2` and `rho` $\in \{0.95, 0.98, 1.0\}$, select solution methods from `0` to `3`, and select simulation method `1`
- **Table 7**: same as Table 6 except for choosing simulation method `2`
- In addition, for all the tables we report results for different numbers of states of Markov chain ($N = 5, 10, 25$). This parameter can be selected interactively before each run as well.

The output file *output/Results.csv* will record the parameters and results of each run, one row per run. Please note that the output are the **raw** values (of unconditional means and standard deviations of income, consumption, and assets), whereas in the paper we report the percent **deviations from the benchmark values** for each discretization method.

The skewness and kurtosis measures in Tables 6 and 7 require some extra work. First, uncomment the following lines in *main.f90*:

```
    ! IF (mode == 2) THEN
    !     write (filename_rho, "(F4.2)") rho_nmar
    !     write (filename_sim, "(I1)") method_sim
    !     write (filename_method, "(I1)") method
    !     ...
    !     OPEN (unit=333, file=TRIM(outputfile), action='write', position='REWIND')
    !         CALL csv_write(333, a)
    !     CLOSE (unit=333)
    ! END IF
```

This part does the job of exporting the full panel of income, consumption, and assets to csv files. Please note that this part takes fairly long due to I/O speed of Fortran, which is also why this part is commented out by default. After exporting, run the R script *KellySkew_CSKurt.R* (also time-consuming). At the end, it will generate a file *skew_kurt.csv* in the *output* folder.

Replication of tables in appendices

Table 9 is simple: First, uncomment the following lines in the subroutine `Tauchen` defined in *discretization.f90*:

```
    ! OPEN (unit=38, file=TRIM(projectpath)//"input/Omega_calibrated.csv", action='read',
position='REWIND')
    ! READ(38, *) omega_from_file(:,1)
    ! READ(38, *) omega_from_file(:,2)
    ! READ(38, *) omega_from_file(:,3)
    ! CLOSE(unit=38)
    ! IF (N==5) omega = omega_from_file(:,1)
    ! IF (N==10) omega = omega_from_file(:,2)
    ! IF (N==25) omega = omega_from_file(:,3)
```

Then set `mode = 0` and `rho = 1.0`, and run solution method `2` to obtain the results of Tauchen with calibrated values of Ω. The other results (benchmark, Rouwenhorst, and original Tauchen) were obtained when running the unit root scenario in Table 1 and 2.

Table 8 provides a comparison of the benchmark solution and the alternative quasi-exact solution in the special case of random walk process. Thus its replication requires

- Getting the benchmark solution result: set `mode = 0` and `rho = 1.0`, as well as setting `sigma_u = 0` in main.f90.
- Getting the alternative quasi-exact solution result: compile and run the code in the sub-directory *normalized*. The requirement and setup for compiling this code are the same as the main code above (need to set `projectpath` in *main.f90*). The results will be written to *output/Results_normalized.txt*.

Contacts

For any questions encountered during replication of the paper or adaptation of certain parts of the code, please email Jutong Pan, Giulio Fella, or Giovanni Gallipoli. Comments and suggestions are welcome.