

Online Appendix to “Applications of Markov Chain Approximation Methods to Optimal Control Problems in Economics”

Thomas Phelan
Federal Reserve Bank of Cleveland

Keyvan Eslami
Ryerson University

April 3, 2022

1 Introduction

This document provides a guide to the code used to produce the results in the paper. All code is written in Python and may be found at https://github.com/tphelanECON/EslamiPhelan_MCA. If you spot errors or have suggestions for improvements please email the second author at tom.phelan@clev.frb.org.

2 Guide to code

In the following we use the abbreviations PFI, MPFI and GPFI for “policy function iteration”, “modified policy function iteration” and “generalized policy function iteration”, respectively. We also write IFP for “income fluctuation problem”. There are three folders: IFP, LQ and macro.finance. Section 2.1 begins with some general remarks on the code common to all folders. Subsequent sections then describe each folder.

2.1 General remarks

We hope that most of the methods within the class constructors are self-explanatory. If not please email the second author at tom.phelan@clev.frb.org. Some methods that are repeatedly used in many of the classes:

- `self.trans_keys`. A list indicating the transitions used within the class. E.g. the list $[(1,0,0),(-1,0,0),(0,1,0),(0,-1,0),(0,0,1),(0,0,-1)]$ for IFP_3D indicates that in the 3D IFP, the transitions are up/down in each dimension.
- `self.p_func`. A dictionary of probabilities, the keys of which are the transitions in `self.trans_keys`. Takes as argument a policy vector and the arrays of indices where the probabilities will be evaluated.

- `self.P_tran`. Creates sparse transition matrix using probabilities in `self.p_func`.
- `self.tran_func` is like `self.p_func` for the generalized algorithm, i.e. a dictionary of probabilities divided by Δ_t . `self.T_tran` is like `self.P_tran` for the generalized algorithm.
- `self.norm_func` is a function defining the constant C used in the generalized policy function iteration. For convenience we also define $H = I + \tilde{T}$.
- `self.P_func` and `self.T_func` reshape sparse matrices for the modified and generalized policy function algorithms.
- `self.mesh` takes a list of integers and returns a meshgrid for the rows and columns corresponding to these transitions. E.g. in a 2D problem with transition $(1,0)$, (a shift up in the first component and unchanged second component), `mesh[(1,0)]` returns the indices used to populate the parts of the associated probability matrix.

2.2 IFP

Code for income fluctuation problems and growth model. Contains the following scripts:

- `classes.py`. Contains the following class constructors:
 - (i) `IFP_2D`: IFP with 1D income process and non-durable consumption. Takes a fixed, positive, grid-dependent timestep and contains methods for PFI and MPFI.
 - (ii) `GIFP_2D`: IFP with 1D income process and non-durable consumption. Sends timestep to zero and contains methods for PFI and GPFI.
 - (iii) `IFP_3D`: Analogue of `IFP_2D` for IFP with 2D income process and non-durable consumption.
 - (iv) `GIFP_3D`: Analogue of `GIFP_2D` with 2D income process and non-durable consumption.
 - (v) `DuraCons`: Analogue of `GIFP_2D` with 1D income process and both durable and non-durable consumption. Only contains PFI and GPFI (since the timestep necessary to use MPFI is minuscule).
- `main_2D.py`: tests PFI, MPFI and GPFI for the IFP with a 1D income process and non-durable consumption. Produces tables (?)
- `main_3D.py`: tests PFI, MPFI and GPFI for the IFP with a 2D income process and non-durable consumption. Produces tables (?)
- `main_durable.py`: tests PFI, MPFI and GPFI for the IFP with a 1D income process and both durable and non-durable consumption. Produces tables (?)
- `NCGex.py`. Non-concave growth example from first section of the paper. Contains class constructor and creates example.
- `durable_ex.py`: solves the durable consumption example in the paper and creates figures.

2.3 LQ

Code for linear-quadratic problems. These are not covered in the main text of the paper but are included in the appendix to check the accuracy and speed of the algorithms. Contains the following scripts:

- `LQ_classes.py`. Contains the following class constructors:
 - (i) `LQ_2D`: 2D problem with two controls.
 - (ii) `LQ_3D`: 3D problem with one control.
 - (iii) `LQ_3D_SD`: 3D problem with state-dependent timestep.
 - (iv) `LQ_3D_GEN`: generalized normalized policy function.
- `LQ_degen_classes.py`. Contains the following class constructors relevant for the linear quadratic problems with degenerate covariance matrices:
 - (i) `LQ_degen3.py`: three-point algorithm described in main text.
 - (ii) `LQ_degen5.py`: five-point algorithm described in main text.
 - (iii) `BOZ`: implements algorithm of [Bonnans et al. \(2004\)](#) for linear quadratic problem.
- `LQ_3D.py`: tests speed and accuracy of code in `LQ_classes`.
- `LQ_degen.py`: tests speed and accuracy of code in `LQ_degen_classes`.

2.4 macro_finance

Code for macrofinance section. Contains the following scripts:

- MF_classes.py. Contains the following class constructors:
 - (i) MF_corr: class constructor for the macrofinance problem in the main text with perfectly correlated noise.
 - (ii) MF_ind: class constructor for a variation of the macrofinance problem in the main text with independent noise.
 - (iii) MF_corr_var_dt: class constructor for a variation of the macrofinance problem in the main text in which transition probabilities for the non-local transitions are constant and the timestep varies (in the main text the opposite choice was made).
- main.py. Uses MF_corr from MF_classes to compute the competitive equilibrium for example in the macrofinance part of the text. Also creates figures for this section.
- main_ind.py. Uses MF_ind from MF_classes to compute an example (not included in main text though).
- FT_PI.py. Compares the policy iteration and false transient approach and produces the final figure in the paper.
- checks.py. Performs four checks on the algorithms:
 - (i) check 1: approximate recovery of log utility values for $\gamma \approx 1$.
 - (ii) check 2: recovery of known boundary values in absence of mean reversion.
 - (iii) check 3: approximate agreement of “constant dt” and “variable dt” algorithms using MF_corr and MF_corr_var_dt.
 - (iv) check 4: check false transient approach remains constant if one begins at the initial values given by the policy iteration algorithm.
- draw.py. Creates Figure 3 that describes the algorithm for selecting the non-local transitions.

Methods in these folders that are not in others and common to the MF classes:

- con_E: creates constants used in the policy functions and updating of aggregate law of motion.
- log_quant: calculates aggregate laws in the case of logarithmic utility, in which case everything is attainable in closed-form.

- `agg_update`: takes pair of value functions and computes new aggregate laws. Needs non-zero timestep and hence E terms.
- `solve_PFI`: solve for the competitive equilibrium using policy iteration.
- `solve_FT`: solve for the competitive equilibrium using the false transient approach.
- `bound_adj`: function that ensures transitions do not leave grid.

References

Bonnans, J. F., Ottenwaelter, É., and Zidani, H. A fast algorithm for the two dimensional HJB equation of stochastic control. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, 38(4):723–735, 2004. doi:[10.1051/m2an:2004034](https://doi.org/10.1051/m2an:2004034).