Introduction To Computational Economics Using ~~Fortran~~ Python

Cagri S. Kumru[1] Dr. Kumru's PhD students[2]

[1]Australian National University

[2]Khademul Chowdhury, Jiu Lian, Dr. Yurui Zhang

QuantEcon Seminar

Introduction

Introduction to Computational Economics Using FORTRAN.

By: Hans Fehr and Fabian Kinderman

- github.com/fabiankindermann/ce-fortran

This Book..

- Part I: Introduces to FORTRAN and Numerical Methods
- Part II: Computational Economics for Beginners
- Part III: Advanced Computational Economics (DP).

Within Each Topic:

- Explain theoretical background
- Show how to implement the problem in computer
- Discuss simulation results.

Chapters

1. FORTRAN90: A simple programming language
2. Numerical solutions method

3. The static general equilibrium
4. Topics in finance and risk management
5. The life-cycle model and inter-temporal choice
6. The overlapping generations model
7. Extending the OLG model

8. Introduction to dynamic programming
9. Infinite horizon model
10. Life-cycle choices and risk
11. The stochastic OLG Model

FORTRAN Vs Python

FORTRAN Vs Python

- User Friendliness
- Compilation
- Library Resources
- Speed

Our Project:

Computational Economics in Python

Life-Cycle Model (Chapter 10.1)

Model Settings:

- Households' life starts at age 1 and ends at age J. The variable j denotes the agent's actual age. ψ_j is the conditional survival probability.
- Agents start their working life at age 1 and retire at age j_r.
- Agents' utility function is given by: $u(c) = \frac{c^{1-\frac{1}{\gamma}}}{1-\frac{1}{\gamma}}$.

Life-Cycle Model (Chapter 10.1)

Model Settings:

- Each period, agents receive an endowment of productive efficiency units h_j which they can supply to the market at the wage w.

$$h_j = \begin{cases} e_j \cdot exp(\theta + \eta_j) & \textbf{if} \quad j < j_r \\ 0 & \textbf{if} \quad j \geqslant j_r \end{cases},$$

  where e_j is a deterministic age-profile, θ is a fixed productivity effect that is drawn at the beginning of the life cycle, and η_j evolves stochastically: $\eta_j = \rho\eta_{j-1} + \epsilon_j$ with $\epsilon \sim N(0, \sigma_\epsilon^2)$.

- The pension system is given by $pen_j = \kappa \cdot \frac{w}{j_r - 1} \cdot \sum_{j=1}^{j_r-1} e_j$ if $j \geqslant j_r$ ($pen_j = 0$ if not retire).

Life-Cycle Model (Chapter 10.1)

Households' problem:

$z = (j, a, \theta, \eta),\ z^+ = (j+1, a^+, \theta, \eta^+)$

$$V(z) = \max_{c, a^+} u(c) + \beta \psi_{j+1} \mathbb{E}[V(z^+)|\eta]$$

$$\textbf{s.t.} \quad a^+ + c = (1+r)a + wh + pen,\ a^+ \geqslant \underline{a}(j+1, \theta)$$

$$\eta_j = \rho \eta_{j-1} + \epsilon_j, \quad \epsilon \sim N(0, \sigma_\epsilon^2)$$

F.O.C:

$$c = (\beta \psi_{j+1}(1+r) \cdot \mathbb{E}[c(z^+)^{-\frac{1}{\gamma}}|\eta])^{-\gamma}$$

Life-Cycle Model (Chapter 10.1)

Fehr and Kindermann's codes style:

- Lots of subroutines, divide problems into pieces and solve them using specific subroutines.
- Each program has two .f90 files plus one toolbox.f90 file.
  prog10_01.f90: main program, including running directives and important subroutines.
  prog10_01m.f90: define global variables, including frequently used functions.
  toolbox.f90: useful toolkit, including rouwenhorst method, time meter, grid discretizing with constant and growing space, and root-finding, etc.

Life-Cycle Model (Chapter 10.1)

Some interesting computational techniques in this chapter/book:

- Dealing with liquidity constraint: We have state-dependent borrowing constraint $\underline{a}(j+1,\theta) \leqslant 0$.

  Normal way: Discretize the grid points contingent on age j and the fixed effects θ_i. That is, specify grid points as $\{\hat{a}_{v,j,i}\}$. Then specify a lower bar for each j and θ_i level: $\hat{a}_{0,j,i} = \underline{a}(j,\theta_i)$. However, this increases the computational burden massively! If we have 1-80 ages, 2 θ levels, then we can have at most 160 different asset grids!

Life-Cycle Model (Chapter 10.1)

Some interesting computational techniques in this chapter/book:

- Dealing with liquidity constraint: We have state-dependent borrowing constraint $\underline{a}(j+1,\theta) \leqslant 0$.

  Nice way: Define a variable $\tilde{a}^+ = a^+ - \underline{a}(j+1,\theta_i)$. $\tilde{a}^+$ denotes the savings of a household in excess of the borrowing limit $\underline{a}(j+1,\theta_i)$. Then the budget constraint can be transformed to:

$$[\tilde{a}^+ + \underline{a}(j+1,\theta)] + c = (1+r)[\tilde{a} + \underline{a}(j,\theta)] + wh + pen$$
$$\tilde{a}^+ \geqslant 0$$

  Using this method, we only need to discretize $\tilde{a}$ and make slight changes to the budget constraint.

Life-Cycle Model (Chapter 10.1)

Some interesting computational techniques in this chapter/book:

- Interpolation:

$$c(j, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = \left[\beta \psi_{j+1}(1+r) \sum_{g^+=1}^{m} \pi_{gg^+} \cdot c(j+1, a^+, \hat{\theta}_i, \hat{\eta}_{g^+})^{-\frac{1}{\gamma}} \right]^{-\gamma}$$

$$a^+ + \underline{a}(j+1, \theta) + c(j, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = (1+r)[\hat{a}_v + \underline{a}(j, \theta)] + wh + pen$$

Normal way: We have calculated the policy function $c(j+1, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g)$. Using the interpolation method, we can get a smooth continuous function: $S(j+1, a, \hat{\theta}_i, \eta)$. We can evaluate $S(j+1, a^+, \hat{\theta}_i, \hat{\eta}_{g^+})$ and then solve the equation system.
However, we need to evaluate $S(j+1, a^+, \hat{\theta}_i, \hat{\eta}_{g^+})$ m times for each different a^+ (the numbers could be huge if we use root-finding method that requires to try many different values of a^+).

Life-Cycle Model (Chapter 10.1)

Some interesting computational techniques in this chapter/book:

- Interpolation:

  Nice way: Instead of interpolate $c(j+1, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g)$, we interpolate the whole Right Hand Side equation:

$$RHS(j, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = \left[\beta \psi_j (1+r) \sum_{g^+=1}^{m} \pi_{gg^+} \cdot c(j, \hat{a}_v, \hat{\theta}_i, \eta_{g^+})^{-\frac{1}{\gamma}} \right]^{-\gamma}$$

  Thus, for each different a^+, we only need to evaluate $RHS(j+1, a^+, \hat{\theta}_i, \hat{\eta}_g)$ once. This reduces the computational time massively.

$$c(j, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = RHS(j+1, a^+, \hat{\theta}_i, \hat{\eta}_g)$$

Life-Cycle Model (Chapter 10.1)

Some interesting computational techniques in this chapter/book:

- Transformed value function:

  The expectation of the next period value function is given by:

$$EV(j+1, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = \sum_{g^+=1}^{m} \pi_{gg^+} \cdot V(j+1, \hat{a}_v, \hat{\theta}_i, \eta_{g^+})$$

  Then the current period value function is given by:

$$V(j, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = u(c(j, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g)) + EV(j+1, a^+, \hat{\theta}_i, \hat{\eta}_g)$$

  We only need to interpolate the $EV(j+1, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g)$ with respect to a^+.

Life-Cycle Model (Chapter 10.1)

Some interesting computational techniques in this chapter/book:

- Transformed value function:
  However, the problem is when a is approaching 0, the value function is diverging to $-\infty$ (recall that the utility function is: $u(c) = \frac{c^{1-\frac{1}{\gamma}}}{1-\frac{1}{\gamma}}$).
  This leads to larger error when a is approaching 0 for interpolating $EV(j+1, a^+, \hat{\theta}_i, \hat{\eta}_g)$.
  Normal way: Increase the grid points number.
  Nice way: Transform the expected value function:

$$EV(j+1, \hat{a}_v, \hat{\theta}_i, \hat{\eta}_g) = \left[(1-\frac{1}{\gamma}) \sum_{g^+=1}^{m} \pi_{gg^+} \cdot V(j+1, \hat{a}_v, \hat{\theta}_i, \eta_{g^+}) \right]^{1-\frac{1}{\gamma}}$$

  The advantage of this is now the interpolation only has to approximate a function that is linear in a and does not diverge to $-\infty$ when a is approaching 0.

Life-Cycle Model (Chapter 10.1)

Other interesting computational techniques in this chapter/book:

- Endogenous gridpoints method (used in Kindermann and Krueger (2022))
- Instead of discretizing today's asset level a, we discretize tomorrow's asset level a^+.
- Advantage: It does not need a root-finding or minimization routine at all, but only relies on an analytical solution of the first-order condition.

Life-Cycle Model (Chapter 10.1)

My code's style:

Life-Cycle Model (Chapter 10.1)

My code's style:

- Exactly the same as the book ...

Life-Cycle Model (Chapter 10.1)

My code's style:

- Exactly the same as the book for some chapters.

Life-Cycle Model (Chapter 10.1)

My code's style:

- Exactly the same as the book for some chapters.
- Write my own toolkits, including:

  simulate.py: simulate_AR

  DiscretizeTools.py: NormalDiscrete1, NormalDiscrete2, log_normal_discrete, rouwenhorst (sources from internet), grow_grid

  linear.py: LinintGrow, LinintEqui

  ToolBox.py: SortOrder

Life-Cycle Model (Chapter 10.1)

My code's style:

- Following the coding logic as the book for Chapters 1-8.
- Try to use Object-oriented programming (OOP) from Chapter 9 (Thanks to Professor Fedor Iskhakov's ECON8014 course).

Female labour-force participation (Chapter 10.3)

Model setting:

- Household's state variables become: $z = (j, a, h_f, \theta, \eta_m, \eta_f)$, where h_f is the human capital of women and $\{\eta_m, \eta_f\}$ are the autoregressive shock of male and female respectively. The $h_{f,j}$ evolves according to:

$$\log(h_{f,j+1}) = \max[\log(h_{f,j}) + (\xi_1 + \xi_w \cdot j) l_j - \delta_h(1 - l_j), \log(h_{f,1})],$$

  where l_j is the female labour-force choice ($l_j \in \{0, 1\}$).
- Exogenous child care cost is included.

Female labor-force participation (Chapter 10.3)

Time compare:

- Added state variables increase the computational time massively. Huge differences between using Fortran and Python.
- Using parallel programming from Chapter 10: '`from joblib import Parallel, delayed`'
- Python is still much slower than Fortran even using 18 parallel threads. Fortran only needs 3 seconds to finish running a model. Python needs almost 1 hour to finish (Hope to get some ideas from QuantEcon).

Remarks

Fortran:

- Advantages:
  Faster speed: speed is the most important advantage of Fortran;
  Free choice of index: you can choose index starts and ends freely;
  Rigorous variable declaration: you won't mess up with your variables;
- Disadvantages:
  Compiler sensitive: nightmare for programming beginners, **ifort** and **gfortran** has different directive style;
  Few reliable IDE choices: I tried **Geany** and **Visual Studio**, but ends with the command line;
  Hard to debug: could be even more difficult for complex programs.
  Variables declaration: it is not easy to change variables' size within the program (need deallocate and reallocate).

Remarks

Python (from a beginner perspective):

- Advantages:
  Jupyter Notebook: lower learning cost, friendly for beginners;
  Flexible variables: do not need to declare variables' name and size at the beginning of the program;
  Packages: lots of useful packages available.
  Support: Python is popular and can get support easily from others (peers, StackOverflow, and QuantEcon)
- Disadvantages:
  Slower Speed: need to figure out how to speed up (parallel programming, Numba, GPU, or supercomputer);
  Index: all the index needs to start from 0 (???), which could be annoying when you have ages starting from 1.

Remarks

At the beginning stage of learning Python, I haven't gotten many ideas of how to fully utilize Python's advantages. Plus, based on my current project which is related to some previous research that used Fortran, Fortran is preferable for me at this stage.

Current Fortran compiler I use: `Intel Fortran Compiler (ifort)`.
Parallelization library I use: `OpenMp`.

Thanks

Appreciate any criticism/advice/suggestions.