Lifecycle Model

This note explains how to solve a simple finite-horizon life-cycle consumption-saving model with stochastic labor income in continuous time. Section 1 lays out the model and explains the algorithm. For comparison Section 2 references some cutting-edge methods for solving the same problem in discrete time and shows that the continuous-time method performs favorably.

1 Lifecycle Model in Continuous Time

We focus on the problem of a given individual in partial equilibrium. The model can, in principle, be extended to general equilibrium with a continuum of individuals that form overlapping generations. Denote wealth by a, age by t and idiosyncratic labor productivity by z. The individual lives from age $t = 0$ to age $t = T$ and solves

$$\begin{aligned}\max_{\{c_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^T e^{-\rho t} u(c_t) dt \quad \text{s.t.} \\ \dot{a}_t &= w z_t + r a_t - c_t \\ dz_t &= \mu(z_t)dt + \sigma(z_t) dW_t \\ a_t &\geq 0.\end{aligned}$$

We assume that idiosyncratic productivity z_t follows an Ornstein-Uhlenbeck process in logs, i.e.

$$d \log z_t = -\theta \log z_t dt + \sigma dW_t \quad \Rightarrow \quad \mu(z) = \left(-\theta \log z + \frac{\sigma^2}{2}\right) z, \quad \sigma(z) = \sigma z$$

and we impose reflecting barriers at some lower bound $\underline{z}$ and at some upper bound $\bar{z}$. The algorithm can easily be extended to allow for a lifecycle profile in idiosyncratic productivity but we do not incorporate this here. The HJB equation is

$$\rho v(a,z,t) = \max_c \; u(c) + \partial_a v(a,z,t)(wz + ra - c) + \mu(z)\partial_z v(a,z,t) + \frac{\sigma^2(z)}{2}\partial_{zz} v(a,z,t) + \partial_t v(a,z,t)$$

with terminal condition

$$v(a,z,T) = 0 \quad \text{all } (a,z) \tag{1}$$

and with state constraint

$$a \geq 0.$$

As usual, the state constraint implies a state-constraint boundary condition

$$\partial_a v(a,z,t) \geq u'(wz + ra)$$

Challenge: terminal condition at death T. The terminal condition on the level of the value function (1) implies that its derivative $\partial_a v(a,z,T) = 0$ for all (a,z) which then implies $c(a,z,T) = \infty$. This actually makes sense: people should spend down everything when they are about to die which requires an infinite consumption flow per infinitesimal unit of time. The only problem is that this terminal condition is computationally ill-behaved since it involves infinity.

A possible solution to this challenge is as follows:[1] we impose a terminal condition corresponding to a "warm glow" bequest motif as in papers by Atkinson (1971) and De Nardi (2004). In particular, we replace the individual's objective function by

$$\mathbb{E}_0\left[\int_0^T e^{-\rho t}u(c_t)dt + e^{-\rho T}\phi_\varepsilon(a_T)\right] \quad \text{where} \quad \phi_\varepsilon(a) = \varepsilon\frac{(\kappa+a)^{1-\gamma}}{1-\gamma}$$

and where $\varepsilon, \kappa \geq 0$ are constants. Our original problem corresponds to the limit $\varepsilon \to 0$ so that $\phi_\varepsilon(a) \to 0$ for all a. This problem is solved easily by solving the HJB equation with terminal condition

$$v(a,z,T) = \phi_\varepsilon(a) \quad \text{all } (a,z) \tag{2}$$

in place of (1) and where ε is a very small number, say $\varepsilon = 10^{-8}$. Other approaches are possible as well. One alternative is to impose a terminal condition directly on $c(a,z,T)$ or equivalently $\partial_a v(a,z,T)$ for instance $\partial_a v(a,z,T) = u'(1000)$ for $a > 0$ and $\partial_a v(0,z,T) = u'(wz)$.[2] Another possible solution is to work with an age-dependent discount rate $\rho(t)$. That is, solve the model slightly past age T and set the discount rate equal to a large value $\rho(t) = 1,000$ for $t \geq T$. All of these solutions work nicely in practice.

Algorithm: The Matlab code can be found at `http://www.princeton.edu/~moll/HACTproject/lifecycle.m`. The discretization follows the same steps as in Section 5 of `http://www.princeton.edu/~moll/HACTproject/HACT_Numerical_Appendix.pdf`. In particular, we discretize (a,z,t) as $a_i, i =, ..., I, z_j, j = 1, ..., J$ and $t^n, n = 1, ..., N$ and use the short-hand notation $v^n_{i,j} = v(a_i, z_j, t^n)$. We stack all $v^n_{i,j}$ for a given n into a vector $\mathbf{v}^n$ that is of dimension IJ. The

[1] We thank SeHyoun Ahn for suggesting this approach.

[2] What we know is that people consume everything at T. Let's consider a discrete-time economy with time periods of length Δt and budget constraint $a_{t+\Delta t} = \Delta t(wz_t + ra_t - c_t) + a_t$. Then we know $a_{T+\Delta t} = 0$ which implies $c(a,z,T) = wz + ra + a/\Delta t$. Therefore, we have

$$c(a,z,T) = \begin{cases} \infty, & a > 0 \\ wz, & a = 0 \end{cases}$$

We cannot exactly impose $c(a,z,T) = \infty$ for $a > 0$. So we impose $c(a,z,T) = 1,000$ or some other large number for $a > 0$. The proposed boundary condition is equivalent.

discretized HJB equation is then a system of difference equations

$$\rho \mathbf{v}^n = \mathbf{u}(\mathbf{v}^{n+1}) + \mathbf{A}(\mathbf{v}^{n+1})\mathbf{v}^n + \frac{\mathbf{v}^{n+1} - \mathbf{v}^n}{\Delta t}$$

This system of difference equations is simply solved backwards in time from a terminal condition $\mathbf{v}^N$ given by (2).

1.1 Results and Performance

The figure plots consumption and saving policy functions for particular income types and ages under the assumption that death occurs at age $T = 75$. As expected, individuals decumulate wealth faster as they approach the end of their lives. The speed of the algorithm depends on

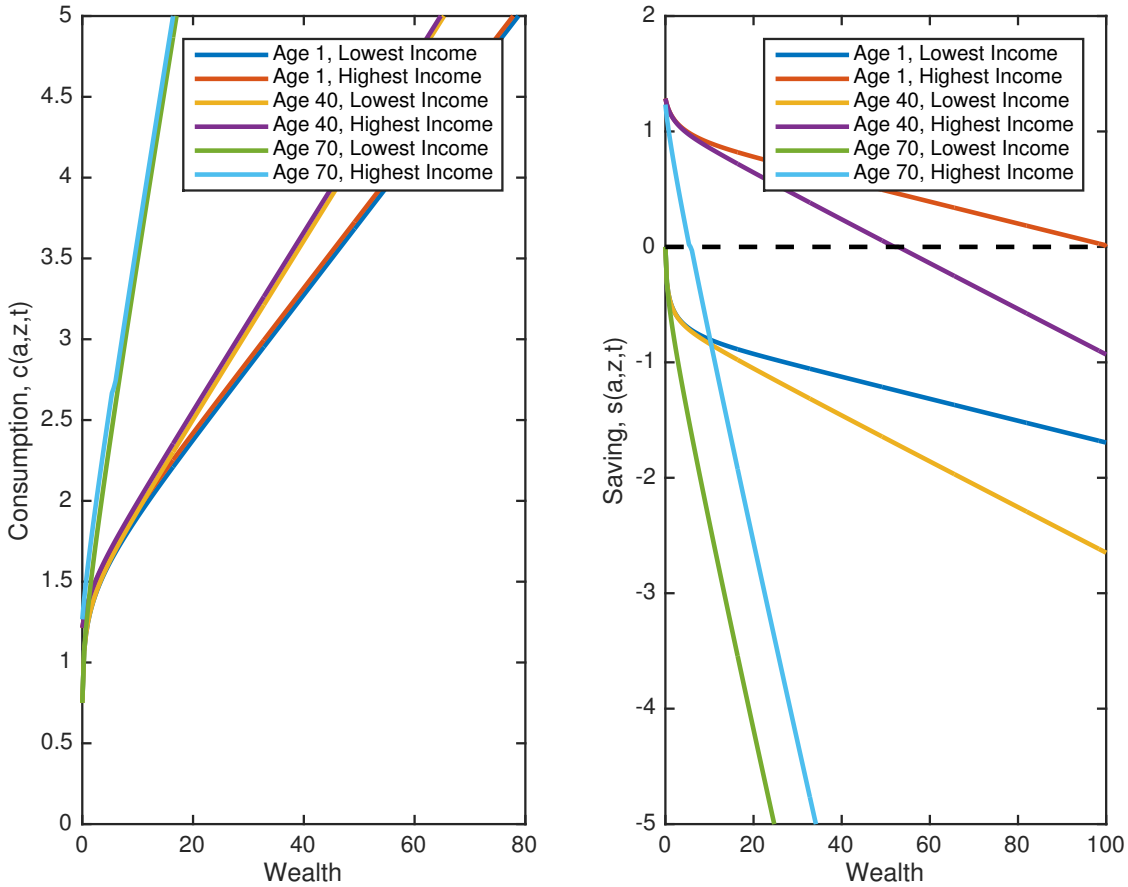

Figure 1: Policy Functions in Lifecycle Model

how finely age is discretized. With $N = 75$ age steps (and $I = 300$ and $J = 15$ grid points for wealth and productivity), it takes 0.82 seconds on a Macbook Pro laptop computer. With $N = 300$ time steps and the same number of grid points for wealth and productivity it takes 3.26 seconds.

2 Discrete Time Version for Comparison

Just to get a sense of the rough performance of the code relative to traditional discrete-time methods, we compare the performance to discrete-time implementations in various programming languages written by Jesus Fernandez-Villaverde and collaborators. In particular the continuous-time model above is the analogue of the model on slide 12 of these lecture notes `https://www.sas.upenn.edu/~jesusfv/Lecture_HPC_9_Paralellization.pdf` for which Jesus & co have also programmed up various numerical implementations. The notation is somewhat different:

1. labor productivity z is e in Jesus' notation
2. wealth a is x in Jesus' notation

Apart from that the two models are basically the same. Jesus & co use the same number of grid points for wealth and labor productivity but considerably less grid points for age. We use 75 or 300 (see above) whereas they use 10.

Slide 38 of `https://www.sas.upenn.edu/~jesusfv/Lecture_HPC_9_Paralellization.pdf` summarizes the performance of a Julia implementation.[3] For example parallelizing with 4 cores takes 22 seconds. This can be compared with 0.82 and 3.26 seconds in our Matlab implementation without parallelization.

Other comparisons:

- `C++` code `https://github.com/davidzarruk/Parallel_Computing/blob/master/Cpp_main.cpp` runs in 0.91 seconds
- `C++` code with `openMPI`, i.e. parallelization, `https://github.com/davidzarruk/Parallel_Computing/blob/master/MPI_main.cpp` takes 0.33 seconds.

Our preliminary conclusion is that our Matlab implementation of the continuous-time life-cycle model without parallelization is competitive with both a `Julia` implementation with parallelization and a `C++` implementation without parallelization of the analogous discrete-time problem. And it is almost competitive with a `C++` implementation using parallelizations (with a coarser discretization of age $N = 10$ the continuous-time code runs in 0.13 seconds, i.e. it is competitive).

[3] The code is here `https://github.com/davidzarruk/Parallel_Computing/blob/master/Julia_main_parallel.jl` or a slightly faster and more compact implementation here `https://github.com/davidzarruk/Parallel_Computing/blob/master/Julia_main_pmap.jl`.