Lecture_3

November 15, 2021

1 Lecture 3

2 A Partial-equilibrium Life-cycle Model with Income Risk

- First, we set up and solve a dynamic model with income risk using dynamic programming methods (backward induction)
- Second, we use the solution to the model to simulate a sample of individuals over an entire lifetime

2.1 1. The Model

- We consider a finite-horizon model where individuals live for T periods and work for r periods.
- While working, the individual earns stochastic labor income, y_t, defined by transitory income shocks.
  - One could also consider persistent/permanent income shocks (the canonical Buffer-Stock Model, Caroll (1997)).
  - To maintain parsimony, we abstract from such dependencies across time.
- Out of all available resources (cash-on-hand), w_t, the individual consumes c_t and saves a_t at the gross real interest rate, R.
  - For future use, we term w_t a state-variable and c_t a choice variable.
  - It is possible to introduce more state and choice variables. However, computation time often increases dramatically.
  - Our simple model serves merely to show some common practices and principles.
- Individuals are borrowing constrained. Thus, they must have non-negative savings in any period.
  - A borrowing constraint is often necessary to describe empirical observations on savings and wealth.
  - Implies the existence of hand-to-mouth consumers who consume all available resources and have no wealth.

2.1.1 1.1 The Bellman Equation

Letting w_t denote cash-on-hand and defining l_t as an indicator for labor-force participation, the recursive formulation of the decision problem - the Bellman equation - reads:

$$
\begin{aligned}
V_t\left(w_t\right) = & \max_{c_t}\left\{u\left(c_t\right)+\beta\cdot\mathbb{E}\left[V_t\left(w_{t+1}\right)\right]\right\} \\
& s.t. \\
a_t = & \, w_t - c_t \\
w_{t+1} = & \, y_{t+1}l_{t+1} + R\cdot a_t \\
y_{t+1} = & \, exp\left(z_{t+1}\right) \\
z_{t+1} \sim & \, \mathcal{N}\left(\mu,\sigma\right) \\
a_t \geq & \, 0
\end{aligned}
$$

2.2 2. Solving the Model

We solve the maximization problem for policy functions (optimal current consumption given the current state), $c_t^*\left(w_t\right)$, using backward induction as described in Section 2.2 and coded in Model.

2.2.1 2.1 Discretizing the State Space

Computers do not deal well with continuous distributions. Hence, we discretize the state space (a continuous line) by assuming an exogenous and time-independent grid (a vector of discrete points on that line) for cash-on-hand, $w \in \mathcal{G}_w$. Thus, we only know the policy function at specific gridpoints. Due to random shocks, the value of cash-on-hand does generally not coincide with a specific gridpoint. Thus, we interpolate over levels of the value function at the closest gridpoints. For details, see Tools > inter_linear_1d. If cash-on-hand is larger (smaller) than the last (first) gridpoint, we approximate the level of the value function by extrapolation.

2.2.2 2.2 Backward Induction

To initialize the backward induction procedure, we utilize that the continuation value is zero, $V_{T+1} = 0$. Thus, the decision problem in the final period of life reduces to:

$$
\begin{aligned}
V_T^*\left(w_T\right) &= u\left(w_T\right) \\
c_T^*\left(w_T\right) &= w_T
\end{aligned}
$$

for all gridpoints, $w_T \in \mathcal{G}_w$. Nothing is left on the table in the final period, T.

In the backward-induction spirit, we then go back one period and solve:

$$
V_{T-1}\left(w_{T-1}\right) = \max_{c_{T-1}}\left\{u\left(c_{T-1}\right)+\beta\cdot\mathbb{E}\left[u\left(w_T\right)\right]\right\}
$$

for optimal consumption in period $T-1$, to obtain new policy functions, $c_{T-1}^*\left(w_{T-1}\right)$ at every gridpoint, $w_{T-1} \in \mathcal{G}_w$.

We repeat this process period-by-period. Thus, in a generalized form, we sequentially solve:

$$V_t(w_t) = \max_{c_t} \{u(c_t) + \beta \cdot \mathbb{E}[V_t^*(y_{t+1} l_{t+1} + R \cdot (w_t - c_t^*(w_t)))]\}$$

for optimal consumption in period t to obtain new policy functions, $c_t^*(w_t)$, in every period at every gridpoint, $w_t \in \mathcal{G}_w$.

The only remaining challenges in solving the problem are to: 1. Evaluate the expectation operator over the value function tomorrow. 2. To maximize the value function today for given future policy functions.

To evaluate the expectation, we discretize the shock distribution by Gaussian Quadrature (specifically Gauss-Hermite) as shown in Tools > gauss_hermite(). This produces a number, S, of nodes and weights that can be used to approximate the expectation (an integral) with great accuracy even for a small number of nodes.

To maximize the value function for given future policy functions, we loop over cash-on-hand and utilize standard numerical solution tools (scipy.optimize) to find optimal consumption given every level of cash-on-hand.

2.3 3. Simulating the Model

Having solved for the policy functions, we now simulate a sample of $N = 100000$ individual paths as coded in Model > simulation().

As we discretize the state space, we only know levels of the policy function at certain gridpoints. However, for a continuous income distribution, any individual cash-on-hand will generally not coincide with one of these gridpoints. Accordingly, we again use linear interpolation/extrapolation over the value of policy functions at known gridpoints to determine the optimal level of consumption.

2.4 4. Calibration

In the numerical examples, we assume a CRRA utility function:

$$u(c_t) = \frac{c_t^{1-\rho}}{1-\rho}$$

We then assume the a standard parameter value for risk-aversion, $\rho = 2$. To weed out trending variation in consumption coming from the Euler-equation, we assume that $R = \beta^{-1}$. To limit computational time we consider a life-cycle that consists of $T = 6$ periods with no mortality risk before the final period. To model retirement in a reasonable way, the individual works for $r = 4$ periods earning wage income. Wage income is stochastic and follows a log-normal distribution with $\mu = 0$ and $\sigma = 0.5$. This is somewhat arbitrary but serves merely to illustrate the signifigance of risk for saving and consumption.

Given this shock distribution, we know that $\mathbb{E}[y_t] = exp\left(\mu + \frac{\sigma^2}{2}\right) \approx 1.13$. To avoid extrapolation but simultaneously preserve accuracy, we assume that the last gridpoint is $w(\#) = 5$. Finally, we assume $\# = 10000$ discrete gridpoints in the grid for cash-on-hand.

3 5. Implementation

[ ]:
```
# Toolkit on coding multiperiod OLG models
# Iowa State Univeristy, Fall semester 2021
# Authors: Frederik Bjørn Christensen, Tim Dominik Maurer

# Importing relevant libraries
import numpy as np
import Tools as tools
import ModelFunctions as model
import matplotlib.pyplot as plt
import time
plt.rcParams['figure.figsize'] = [12, 8]
from matplotlib.ticker import StrMethodFormatter

# Setting parameters
par = dict()

# Setting model-specific parameters
par['T'] = 6
par['r'] = 4
par['R'] = 1.2
par[' '] = 1/par['R']
par[' '] = 2
par['l'] = np.concatenate((np.ones(par['r']),np.zeros(par['T'] - par['r'])))
par[' '] = 0
par[' '] = 0.5

# Load Gauss-Hermite weights and nodes
par['S'] = 7
par['x'],par['wi'] = tools.gauss_hermite(par['S'])
par[' '] = par['wi']/np.sqrt(np.pi)

# Compute the discretized shock vector
par['Y'] = np.exp(par[' ']*np.sqrt(2)*par['x'])
par['Y'] = par['Y'].flatten()

# Creating a grid for cash-on-hand
par['w_max'] =  5
par['gridsize_w'] = 1000 # State variable: Cash-on-hand
par[' _w'] = np.linspace(10e-6,par['w_max'],par['gridsize_w'])
```

We solve the model as shown in Model. For future reference, we keep track of computation time:

[ ]:
```
""" Solving the model """
t0 = time.time()
Vstar,Cstar,Astar = model.solve(par)
```

```
t1 = time.time()
print('Time to solve:',t1-t0)
```

Next, we plot our policy functions, first for periods in the workforce and second for periods in retirement:

[ ]:
```
plt.plot(par[' _w'],Cstar[0:par['r'],:].T)
plt.title('Working Periods')
plt.xlabel("Cash-on-hand")
plt.ylabel("Consumption")
plt.gca().legend(('Period 1','Period 2','Period 3','Period 4'))
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}')) # 2 decimal␣
 ↪places
plt.show()

plt.plot(par[' _w'],Cstar[par['r']:par['T'],:].T)
plt.title('Retired Periods')
plt.xlabel("Cash-on-hand")
plt.ylabel("Consumption")
plt.gca().legend(('Period 5','Period 6'))
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}')) # 2 decimal␣
 ↪places
plt.show()
```

For given policy functions, we simulate the model as shown in Model.py:

[ ]:
```
""" Simulating the model """
par['N'] = 100000 # Number of individuals
simW,simA,simC,simY = model.simulation(par,Cstar)
```

For our simulated results, we plot the profile of average savings over the life-cycle and a sample of individual paths:

[ ]:
```
plt.plot(np.mean(simA,axis=1))
plt.xlabel("Periods")
plt.ylabel("Average Savings")
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}')) # 2 decimal␣
 ↪places
plt.show()

plt.plot(np.mean(simA,axis=1),label='Average Savings')
plt.legend(loc="upper left")
plt.plot(simA[:,0:10])
plt.xlabel("Periods")
plt.ylabel("Individual Savings")
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}')) # 2 decimal␣
 ↪places
plt.show()
```

3.1 6. Precautionary Savings

To highlight the importance of risk for saving, we re-run the model with zero risk, but update the mean of the underlying shock process to maintain the same level of average income as in the baseline calibration.

[ ]:
```
# Restating parameters such that mean is preseved but standard deviation is zero
par[' '] = par[' ']**2/2
par[' '] = 0

# Restating the shock vector for deterministic income
par['Y'] = np.repeat(np.exp(par[' ']),par['S'])
par['Y'] = par['Y'].flatten()

# Solving againg
Vstar_nr,Cstar_nr,Astar_nr = model.solve(par)
plt.plot(par[' _w'],Astar[0,:])
plt.plot(par[' _w'],Astar_nr[0,:])
plt.title('First-period Policy Functions with and without Precautionary␣
 ↪Savings')
plt.xlabel("Cah-on-hand")
plt.ylabel("Optimal Average")
plt.gca().legend(('With Risk','Without Risk'))
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}')) # 2 decimal␣
 ↪places
plt.show()

# Simulating again
simW_nr,simA_nr,simC_nr,simY_nr = model.simulation(par,Cstar_nr)
plt.plot(np.mean(simA,axis=1))
plt.plot(np.mean(simA_nr,axis=1))
plt.title('Dynamic Illustration of Precautionary Savings Motive')
plt.xlabel("Periods")
plt.ylabel("Average Savings")
plt.gca().legend(('With Risk','Without Risk'))
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}')) # 2 decimal␣
 ↪places
plt.show()
```

4 6. Potential Extensions

1. Mortality Risk
   - Easily done if no uncertainty about future mortality rates
2. A Pension System
   - PAYG
   - FF
3. Persistent/Permanent Income Shocks

- AR(1)
- Markov processes
- Permanent skill types

4. Accidental or Voluntary Bequest
   - Exogenous
   - Endogenous but on aggregate income
   - Endogenous and family-linked → extra shocks and state variables + convergence of wealth distribution
5. General Equilibrium
   - Convergence of all endogenous distributions

Every extension increases computation time! Thus, we could be forced to find more efficient solution methods. This is the topic of Lecture 4.