A Simple Two Period OLG Model

Prof. Lutz Hendricks

Econ821

January 25, 2016

Contents

Introduction

We set up a simple 2 period OLG model and compute it.
We later extend the model to

- many periods
- random earnings

Then we can study the wealth distribution, earnings distribution, etc.
The main goal for now: figure out how to compute a simple model.

Deterministic Two Period OLG Model

We start with the simplest model:

- Households live for 2 periods.
- There is no uncertainty.

We compute the model in blocks:

- Household
- Firm
- Market clearing

The Environment

Demographics:

- in each period, mass 1 of households are born
- each lives for 2 periods

Endowments:

- the initial old own K_1 units of capital
- each young has 1 unit of work time

Preferences:

$$\max\ u(c_t^y) + \beta\, u(c_{t+1}^o)$$

Technologies:

$$Y = F\,(K, L) = C + K' - (1-\delta)\,K$$

Household problem

Household solves

$$\max\ u(c_t^y) + \beta\, u(c_{t+1}^o)$$

subject to

$$c_{t+1}^o - w_{t+1}^o = (1 + r_{t+1})\ (w_t^y - c_t^y)$$

Solution:

- Euler equation

$$u'(c_t^y) = \beta\,(1 + r_{t+1})\, u'(c_{t+1}^o)$$

- Lifetime budget constraint

$$W_t = c_t^y + \frac{c_{t+1}^o}{1 + r_{t+1}}$$

Definition: Lifetime (permanent) income:

$$W_t \equiv w_t^y + \frac{w_{t+1}^o}{1 + r_{t+1}}$$

Implications for consumption behavior

Permanent Income Hypothesis:

Consumption at each date only depends on W_t, not on the timing of income over the life-cycle.

Consumption growth does not depend on income growth.

In the data: Consumption tracks income over the life-cycle (Carroll and Summers 1991).

Computing the Household Problem

Set of equations to be solved:

$$u'(c_t^y) = \beta\,(1 + r_{t+1})\,u'(c_{t+1}^o)$$

$$W_t = c_t^y + \frac{c_{t+1}^o}{1 + r_{t+1}}$$

Simplify: Solve for a zero of

$$u'(c_t^y) = \beta\,(1 + r_{t+1})\,u'\left([1 + r_{t+1}]\;[W_t - c_t^y]\right)$$

Then use budget constraint to compute c_{t+1}^o.

Closed form solution

We could solve the household problem in closed form for isoelastic utility:

$$c^y = \left[\beta\,(1+r)\,\left\{(1+r)\,(W - c^y)\right\}^{-\sigma}\right]^{-1/\sigma}$$

$$c^y = \frac{\beta^{-1/\sigma}\,(1+r)^{1-1/\sigma}\,W}{1+\beta^{-1/\sigma}\,(1+r)^{1-1/\sigma}}$$

Instead, we will use a general numerical algorithm that searches for a zero of the Euler equation deviation.

Setting Parameters

We use a simple calibration approach.

Model period: $\lambda = 30$ years per period.

Preferences:

- $u(c) = c^{1-\sigma}/(1-\sigma)$.
- $\sigma = 2$ based on micro-evidence.
- $\beta = 0.97$ (better: set to match K/Y).

Interest rate:

- What is "the" interest rate? No good answer.
- Set $\tilde{r} = 0.05$ per year (Cooley and Prescott). Then $(1+r) = 1.05^{\lambda}$.

Earnings:

- Normalize $w_t^y = 1$. Physical ages 21-50.
- Think of w_{t+1}^o as non-capital income of "elderly" (ages 51-80). Set $w_{t+1}^o = 0.6$.

Exogenous and calibrated parameters

Model parameters are either exogenous or calibrated.

Exogenous parameters include:

- Fixed preference parameters (σ).
- Fixed technology parameters (α).
- Calibration targets (K/Y).
- These are set by `const_olg2d.m`.

Calibrated parameters include:

- Discount factor β.
- Depreciation rate δ.

Code organization

- suffix for unique names: `_olg2d`
- startup routine: init_821 + go_olg2d
  - puts shared progs on path
  - switches to directory with programs
- we have a program that runs everything in sequence: `run_all_olg2d`
  - general rule: you should be able to go from nothing to all results with a single command
  - this also serves as documentation
- exogenous model parameters are set by `const_olg2d`
  - We solve the models for different **parameter combinations**.
  - They are indexed by `calNo`.

Programs: Household Problem

We now go over the household programs in detail.
Hint: Always write down the algorithm in "pseudo code" before you start writing programs.

Steps:

1. Set constants: `const_olg2d`
2. Iterate over guesses for c_t^y in the feasible interval $c_t^y \in [0, w_t^y]$ (`hh_solve_olg2d.m`)
3. For each guess of c_t^y compute the deviation from the Euler equation (`hh_dev_olg2d.m`).
4. Stop when Euler equation deviation is sufficiently small.

Now we write the code, typically inside-out.

Euler Equation deviations

`hh_dev_olg2d.m`

Compute c^o from budget constraint

$$c^o = (1+r)\ (w - c^y)$$

Compute right-hand-side of Euler equation

$$\beta\ (1+r)\ u'(c^o)$$

Return deviation

$$dev = u'(c^y) - \beta\ (1+r)\ u'(c^o)$$

Problem: this deviation is not well behaved.

CES Marginal Utility

Marginal utility is very non-linear for low values of consumption.
To avoid strong non-linearity, use inverse marginal utility function:

$$\left(u'\right)^{-1}(x) = x^{-1/\sigma}$$

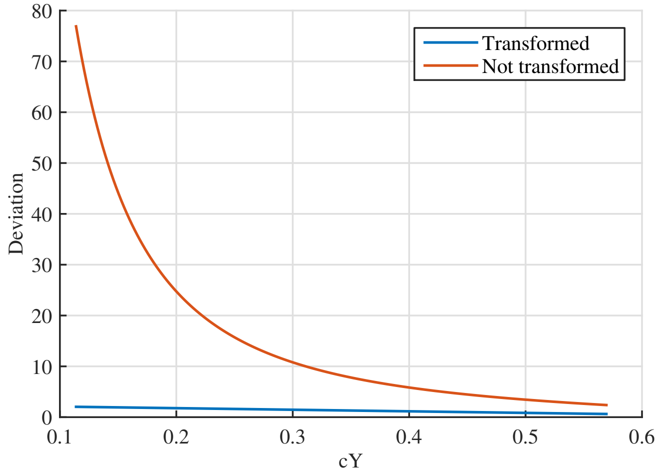
Find the deviation of the transformed Euler Equation

$$c_t^y = \left[\beta\left(1 + r_{t+1}\right)\left(\left[1 + r_{t+1}\right]\left[W_t - c_t^y\right]\right)^{-\sigma}\right]^{-1/\sigma}$$

Drawback: Now it's harder to swap out the utility function

- unless we bundle the Euler equation deviation into the utility function class.

Euler equation deviation as a function of c_t^y



Solving the household problem

`hh_solve_olg2d.m`.

Steps:

1. Set up a range of possible c values.
2. For each c, compute the Euler equation deviation (`hh_dev_olg2d.m`).
3. Search for a c with a small enough deviation using `fzero`.

Digression: Finding Zeros

For single variable problems use the built-in function `fzero`.

Example:

Find the solution to the equation $f(x) = \ln(x) - 5 = 0$.

Set up a function that returns the deviation $f(x)$:

```
function dev = dev1_821(x);
dev = log(x) - 5;
end
```

Use fzero to search for the solution:

```
fzero(@dev1_821, [0.1 100])
ans =
 7.3891
```

Note: `@dev1_821` is a **function handle**.

It essentially passes the name of the function (really: a pointer to the function) to another function.

How `fzero` works:

- Start with two values of x where $f(x)$ is of opposite sign.
- Interpolate between the two points to find a new guess for x.
- Narrow the interval and iterate.

What if the deviation function needs other inputs?

In the OLG model, the deviation function must know all model constants.

But `fzero` cannot pass additional arguments to the objective function.

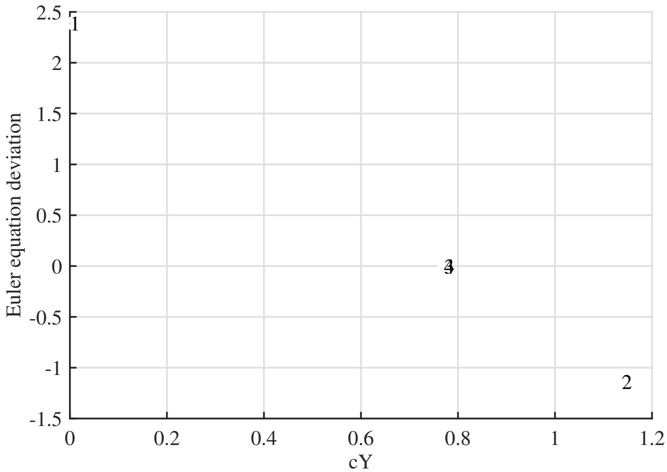
The solution: a **nested function**.

- it sits entirely inside another function

  - here: inside `hh_solve_olg2d`

- it can see all the variables defined in the surrounding function
- it contains a single line
  `dev = hh_dev_olg2d(guess, inputS);`

Search Steps of Household Algorithm



Things to Come

Later we will solve the General Equilibrium for this model.