02a Two period household problem

Textbook

- Miao, section 9.4. but too difficult
- Niepelt ch.2 is on Moodle.

Two period problem

- β is discount factor (割引因子).
- $u(c)$ is utility function (効用関数).
- w_t is income at time t
- R_t is the interest rate at the beginning of period t
- c_t is the consumption at period t
- a_t is the asset holding at the beginning of period t
- discrete time $t = 0, 1$ (離散時間).
- Household two period optimization problem

$$\max \ u(c_0) + \beta u(c_1)$$
$$\text{s.t. } a_1 = w_0 - c_0, \quad c_1 = a_1 R_1 + w_1$$

- intertemporal budget constraint (異時点間予算制約).

$$c_0 + \frac{c_1}{R_1} = w_0 + \frac{w_1}{R_1} = W$$

  W is called the permanent income (恒常所得) or life-time income (生涯所得).

- Lagrangian method

$$\mathcal{L} = u(c_0) + \beta u(c_1) + \lambda \left[W - \left(c_0 + \frac{c_1}{R_1} \right) \right]$$

- Tips: consumption cannot exceed the income.
  $W - \left(c_0 + \frac{c_1}{R_1} \right) \geq 0$.
  The bracket after λ should be something ≥ 0.
  Then add $+$ before λ.
  It is a trick to keep $\lambda \geq 0$ in maximization problems.
- The first-order conditions

$$u'(c_0) - \lambda = 0, \quad \beta u'(c_1) - \frac{\lambda}{R_1} = 0$$

- Euler equation, or the intertemporal marginal rate of substitution (異時点間限界代替率).

$$\frac{u'(c_0)}{u'(c_1)} = \beta R_1 \quad u'(c_0) = \beta R_1 u'(c_1)$$

- Question: what happens to the relative consumption c_1/c_0 if R_1 increases?

- The solution is the system of non-linear equations.

$$c_0 + \frac{c_1}{R_1} = W, \quad u'(c_0) = \beta R_1 u'(c_1)$$

- Suppose $u(c) = \ln(c)$. ln means the natural log.

$$c_1 = \beta R_1 c_0 \;\Rightarrow\; c_0 = \left(\frac{1}{1+\beta}\right) W, \;\; c_1 = \left(\frac{\beta}{1+\beta}\right) W$$

  The total wealth W is divided by the relative share. It is similar to Cobb-Douglas. $u(c_0, c_1) = (c_0)^{1/(1+\beta)} \cdot (c_0)^{\beta/(1+\beta)}$

- constant intertemporal elasticity of substitution (CIES, or CES, 異時点間代替の弾力性が一定) or constant relative risk aversion (CRRA, 相対的リスク回避度が一定).

$$u(c) = \frac{c^{1-\sigma}}{1-\sigma} \text{ or } u(c) = \frac{c^{1-\sigma} - 1}{1-\sigma}$$

  - $u(c) = c$ as $\sigma \to 0$
  - $u(c) = \ln(c)$ as $\sigma \to 1$
  - $u(c_0) + \beta u(c_1) \to \min\{c_0, \beta c_1\}$ as $\sigma \to \infty$

- Niepelt says,

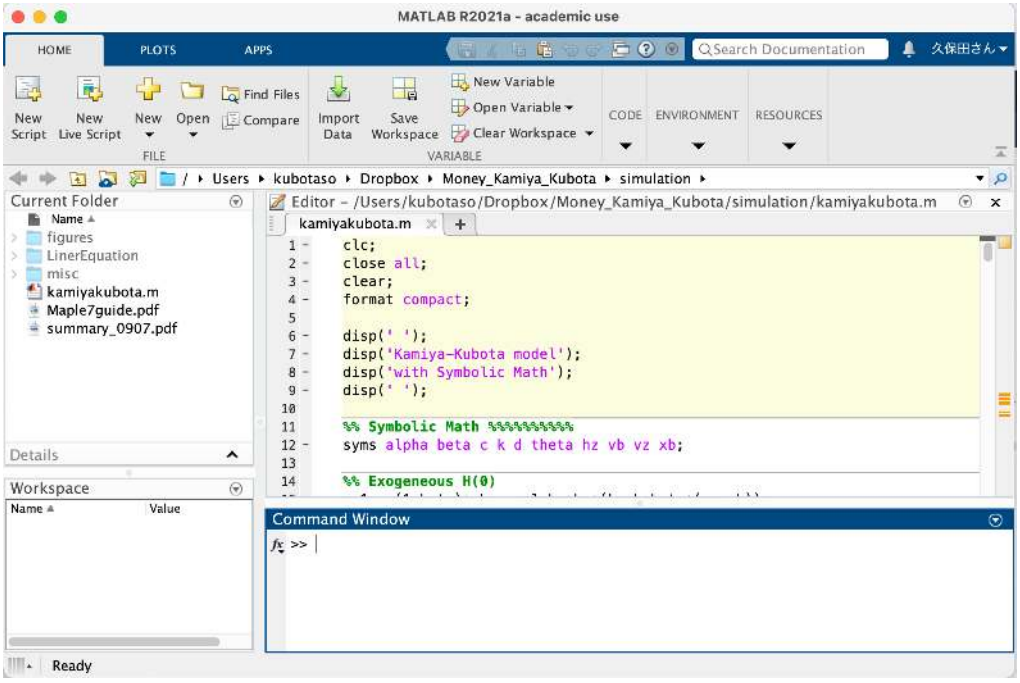
$$c_0 = \left(\frac{1}{1 + \frac{(\beta R)^{1/\sigma}}{R_1}}\right) W$$

Numerical solution of the system of non-linear equation

- In general, we cannot find explicit c_0 and c_1. Numerical solution is helpful.
- Let's begin with the one non-linear equation and one variable case.

$$u'(c_0) = \beta R_1 u'\big(R_1(W - c_0)\big)$$

02b Matlab basics



- simple calculation.

  `10+4    10-4   10*4   10/4`

- simple functions.

  `sin(pi/2)    log(1)   exp(1)`

- variables.

  `x=10   y=4   x+y`

- matrix

  ```
  a=[1,2,3;4,5,6]
  b=[9,8,7;6,5,4]
  a+b   a*b   a.*b
  zeros(3)   ones(2,3)   a=1:0.5:3   a=linspace(1,3,5)
  ```

- plot.

  ```
  a=linspace(1,3,5)
  b = a.^2
  plot(a,b)
  ```

- m-script. write a simple program on editor. save as ”test.m”.

```
x=10
y=4;
```

- Editor tab, Run
- semicolon? Do not show the result.
- Usually, semicolon to all. To show the results, `disp(y);`
- write comment. %
- sections, use %%, Run Sections

```
%% Calculation section
a=1:0.5:3;
b=a.*a; % multiplicate each element
c=3*b; % I'm hungry
disp(c); % display the results
%% plot section
plot(a,c) % plot the results
```

- clean up at the beginning of m-file.

```
clc;
close all;
clear;
```

- if-else

```
testscore = 95; % try other numbers
if testscore >= 90
    disp('Your grade is A');
elseif testscore < 90 && testscore >= 80
% && means logical and, || means logical or
    disp('Your grade is B');
else
    disp('Your grade is C or lower');
end
```

- loop

```
a=zeros(1,10);   % make a container
a(1) = 1;   % initial value
for i = 1:9   % i is for loop
    a(i+1) = a(i)+i;  % "=" means substitution
end
disp(a);
```

- function.
  - Your own function. Call from main program.
  - function is defined at the end of the main program.
  - Definition: `function y=f(x)`
  - program inside `{ }`

- y should be at the last line in the function.
- You can also make a separate file "sumup.m" and put the function in it.

```
%% main program
length = 5;
disp(sumup(length));

%% function
function result = sumup(maxn);
    a=zeros(1,maxn);   % make a container
    a(1) = 1;   % initial value
    for i = 1:maxn   % i is for loop
        a(i+1) = a(i)+i;
    end
    result = a;
end
```

- local vs. global variable
  - local variable is defined inside function.
  - global variable is defined in the main program and all functions.
- Which one should you use?
  - Use global variable for constants in your whole analysis
  - Use local variable if you change the values inside the program.

```
xl = 3;  % local in the main program, not shared in f(x)
global xg;  % define global variable, shared in f(x)
f(5);
disp([xl xg]);
xg = 6;
f(5);

function f(x)  % no return, void function
    global xg  % receive global variable's value
    xl = x;  % xl is the same name, but defined only inside function
    disp([xl xg]);
end
```

02c Newton Method

- One-dimensional root-finding problem: $f(x)$ is a nonlinear function. Find x^* which satisfied $f(x^*) = 0$.
- For example: $f(x) = u'(x) - \beta R_1 u'\big(R_1(W - x)\big)$
- Newton method
  1. Take an initial value x_0
  2. Calculate $f'(x_0)$. Take a tangent line
  3. Extend tengent line to 0. $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$
  4. Calculate $f'(x_1)$. Take a tangent line gain.
  5. Extend the new tangent line. $x_2 = x_1 - \dfrac{f(x_1)}{f'(x_1)}$
  6. repeat the process. ith to $i + 1$th
  7. stop if the process stops down sufficiently. $|x_{i+1} - x_i| < \varepsilon$.
- How to take the derivative $f'(x)$ on the computer?
- analytical derivative may be possible, but not available in general.
  - If yes, you can use special toolbox of software. In Matlab "Symbolic Math Toolbox". There are two more famous and advanced software: Mathematica & Maple
  - $f(x)$ may not be differentiable. In may economic models, $f(x)$ is not defined mathematically. It may include loop and if-else.
  - We usually uses numerical derivative: $\dfrac{f(x+\varepsilon) - f(x)}{\varepsilon}$ with a tiny ε
- you can write Newton method by yourself, but let Matlab solve it.
  - Define $f(x)$ as a *Matlab* function
  - Specify function *handle*. It is a label of a function. Matlab can interpret function f as a variable, and give f to another function.
  - Give f to a `fzero` function, which runs Newton method. `fzero` is for one equation. `fsolve` for multiple equations
- $f(x) = x^2 - 3x - 2$.
- `f_h` is f's function handle.
- 0.5 is the initial value of Newton Method.

```
%  function f & function handling f_h
f_h = @f;
xstar = fzero(f_h,0.5);
disp(xstar);

% one line definition using semicolon
function y = f(x); y = x^2-3*x-2; end
```

02d Numerical solution of household problem: One equation case

- Let's solve the two-period problem!

$$0 = u'(c_0) - \beta R_1 u'\left(R_1\left(w_0 + \frac{w_1}{R_1} - c_0\right)\right)$$

- Assume $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$. Then, $u'(c) = c^{-\sigma}$.
- Method 1: Define parameters inside the function

```
cstar = fzero(@f,100);
disp(cstar);

function y = f(cz)
    sig = 0.5;  % sigma
    bt = 0.95;  % beta
    R = 1.02;  % R1, gross interest rate
    w0 = 120;  % wage at period 0
    w1 = 90;  % wage at period 1
    y = c0^(-sig) - bt*R*(R*w0+w1-R*c0)^(-sig);
end
```

- Method 2: Define parameters as global variables

```
global sig bt R w0 w1;
sig = 0.5;  % sigma
bt = 0.95;  % beta
R = 1.02;  % R1, gross interest rate
w0 = 120;  % wage at period 0
w1 = 90;  % wage at period 1

cstar = fzero(@f,100);
disp(cstar);

function y = f(c0)
    global sig bt R w0 w1;  % use global inside function
    y = c0^(-sig) - bt*R*(R*w0+w1-R*c0)^(-sig);
end
```

- Method 3: Pass parameters to the function
  - **My Recommendation**
  - Separately define two types of inputs: variables and parameters. Each one can be a vector
  - Define function handle as the function of only variable

```
sig=0.5;bt=0.95;R=1.02;w0=120;w1=90;

% packing
parameters = [sig bt R w0 w1];
% Fix parameters. Define function handle where x is the only input
f_h = @(x) f(x,parameters);
```

```
cstar = fzero(f_h,100);
disp(cstar);

function y = f(c0,p)
    % p contains parameters. unpacking
    sig=p(1);bt=p(2);R=p(3);w0=p(4);w1=p(5);
    y = c0^(-sig) - bt*R*(R*w0+w1-R*c0)^(-sig);
end
```

- An advantage of Method 3 is comparative statics.
- Here, change R_1 from 0.8 to 1.2 and calculate c_0 for each R_1.

```
sig=0.5;bt=0.95;w0=120;w1=90;

R_vec = linspace(0.8,1.2,100);  % vector of possible R1
c0_vec = zeros(1,100);  % container to save results

for i=1:100
    R = R_vec(i);  % R1 for each iteration
    parameters = [sig bt R w0 w1];  % different R for each interation
    f_h = @(x) f(x,parameters);  % different function hadle for each iteration
    c0_vec(i) = fzero(f_h,100);  % save c0 for each iteration
end

plot(R_vec,c0_vec);

function y = f(c0,p)
    sig=p(1);bt=p(2);R=p(3);w0=p(4);w1=p(5);
    y = c0^(-sig) - bt*R*(R*w0+w1-R*c0)^(-sig);
end
```

02e Numerical solution of household problem: Two equation case

- It is originally a system of two non-linear equations.

$$0 = u'(c_0) - \beta R_1 u'(c_1)$$
$$0 = w_0 + \frac{w_1}{R_1} - c_0 - \frac{c_1}{R_1}$$

- Use `fsolve` instead of `fzero`
- output is also a vector of two variables `y(1), y(2)`

```
sig=0.5;bt=0.95;R=1.02;w0=120;w1=90;

parameters = [sig bt R w0 w1];
```

```
f_h = @(x) f(x,parameters);  % x is a vector, [c0 c1]

% initial value [100 100] and c_vec are vectors with two variables
c_vec = fsolve(f_h,[100 100]);
disp(c_vec);

function y = f(x,p)
    sig=p(1);bt=p(2);R=p(3);w0=p(4);w1=p(5);  % unpacking parameters
    c0 = x(1); c1 = x(2);  % unpacking variables
    y(1) = c0^(-sig) - bt*R*(c1)^(-sig);
    y(2) = w0 + (w1/R) - c0 - (c1/R);
end
```