Homework 2

Computational Methods

18.06.2025

Megan Hainsworth-Gömann

Question 8.1

Consider the following intertemporal optimization problem:

$$\max_{\{c_v\}_{v=t}^{\infty}} U_t = \sum_{v=t}^{\infty} \beta^{v-t} u(c_v) \quad \text{subject to} \quad \sum_{v=t}^{\infty} c_v = a_t \tag{1}$$

Show that the solution to this problem is equivalent to solving the following sequential (recursive) problem:

$$\max_{c_t} \left[u(c_t) + \beta \max_{\{c_v\}_{v=t+1}^{\infty}} U_{t+1} \right] \tag{2}$$

$$\text{subject to} \quad a_{t+1} = a_t - c_t, \quad \sum_{v=t+1}^{\infty} c_v = a_{t+1} \tag{3}$$

If we split the infinite sum into present and future utility, we obtain:

$$U_t = u(c_t) + \sum_{v=t+1}^{\infty} \beta^{v-t} u(c_v)$$

Reindexing the second sum to start from time *t+1* gives:

$$U_t = u(c_t) + \beta \sum_{v=t+1}^{\infty} \beta^{v-(t+1)} u(c_v)$$

Which implies

$$U_t = u(c_t) + \beta U_{t+1}$$

Maximizing this result gives the solution to the sequential problem.

Question 8.2

Consider an agent who lives for an infinite number of periods indexed by $t = 0, 1, ..., \infty$. In each of the periods he receives a constant income stream w. At each point in time the agent has to decide how much to consume c_t and how much to save for the next period a_t. Negative savings (debt) are also allowed. Having saved an amount of a_t in period t, the agent receives an amount of $(1 + r)a_t$ in the next period. The interest rate therefore is r. Calculate the all-in-one

solution of this problem:

$$\max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\frac{1}{\gamma}}}{1-\frac{1}{\gamma}} \quad \text{subject to} \quad \sum_{t=0}^{\infty} \frac{c_t}{(1+r)^t} = \sum_{t=0}^{\infty} \frac{w}{(1+r)^t} \tag{4}$$

Setting up the Lagrangian and finding the FOC for c_t:

$$\mathcal{L} = \sum_{t=0}^{\infty} \left[\beta^t \frac{c_t^{1-\frac{1}{\gamma}}}{1-\frac{1}{\gamma}} - \lambda \cdot \frac{c_t}{(1+r)^t} \right] + \lambda \sum_{t=0}^{\infty} \frac{w}{(1+r)^t}$$

$$\frac{\partial \mathcal{L}}{\partial c_t} = \beta^t c_t^{-\frac{1}{\gamma}} - \lambda \cdot \frac{1}{(1+r)^t} = 0$$

$$c_t = \left(\lambda \cdot \frac{1}{\beta^t (1+r)^t} \right)^{-\gamma}$$

By taking a ratio of FOCs at time t and $t+1$, we can derive the Euler equation to show how consumption evolves over time:

$$\frac{\beta^{t+1} c_{t+1}^{-\frac{1}{\gamma}}}{\beta^t c_t^{-\frac{1}{\gamma}}} = \frac{(1+r)^t}{(1+r)^{t+1}}$$

$$\beta \left(\frac{c_{t+1}}{c_t} \right)^{-\frac{1}{\gamma}} = \frac{1}{1+r}$$

$$\frac{c_{t+1}}{c_t} = [\beta(1+r)]^{\gamma}$$

Question 8.3

Write the problem in 2 as a dynamic programming problem. Can you derive a closed-form solution for the policy and value function? Assume the value function takes the form:

$$V(a) = \beta \cdot \frac{\left(a + \frac{w}{r}\right)^{1-\frac{1}{\gamma}}}{1 - \frac{1}{\gamma}}$$

Using the utility function from 8.2, defining V(a') and setting a' as $(a + w - c)$, we can define bellman rhs as $u(c_t) + V(a')$, which results in the following Bellman RHS:

$$\frac{c^{1-\frac{1}{\gamma}}}{1 - \frac{1}{\gamma}} + \beta \cdot \frac{\left(a - c + w + \frac{w}{r}\right)^{1-\frac{1}{\gamma}}}{1 - \frac{1}{\gamma}}$$

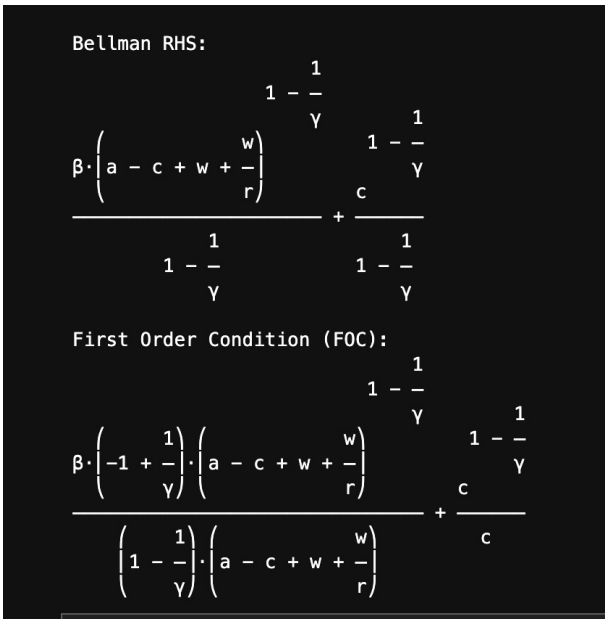
Differentiating that with respect to c gives the FOC:

$$c^{-\frac{1}{\gamma}} = \beta \cdot \left(a - c + w + \frac{w}{r}\right)^{-\frac{1}{\gamma}}$$

Solving for C gives the policy function:

$$c = \frac{a + w + \frac{w}{r}}{1 + \beta^{-\gamma}}$$

The results shown above are simplified from this output:



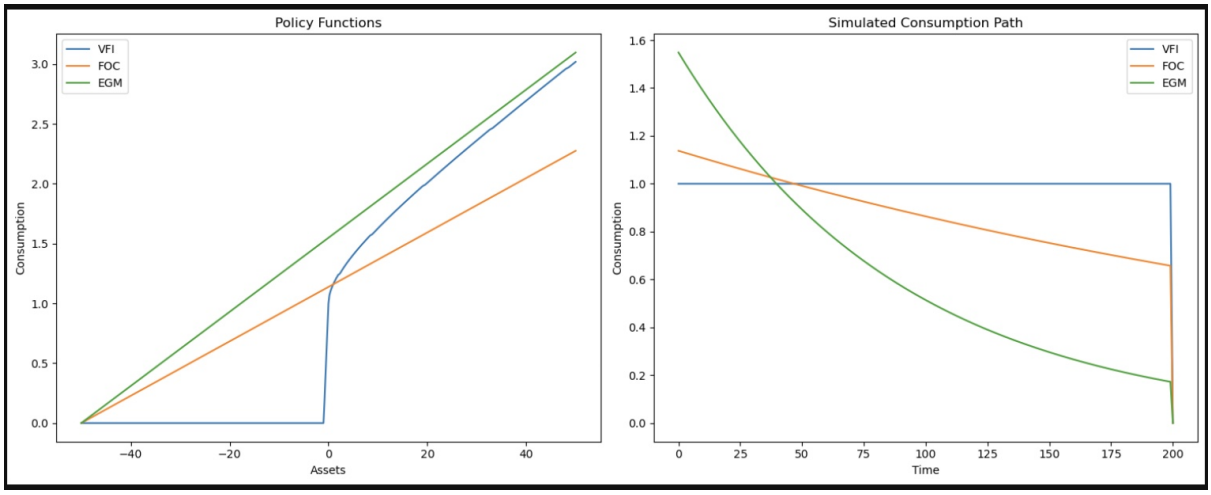


Figure 1: Results in 8.4, 8.5, and 8.6 will be discussed using these graphs.

Question 8.4

Implement the solution to the problem in 2 using minimization and interpolation. Plot the policy and value function for the parameter combination $\gamma = 0.5$, $\beta = 0.975$, $r = 0.02$ *and* $w = 1$. *In addition, simulate a consumption path for 200 periods. Now chance* r *to 0.03. How does the solution change?*

In order to solve the optimization problem using minimization and interpolation, we apply value function interpolation (VFI). We are trying to model the optimal levels of consumption c for every asset level a the will maximize discounted lifetime utility.

To do this, we create a grid over current asset values:

```
grid_a = np.linspace(a_min, a_max, 501)
```

Initialize value and policy functions:

```
V = np.zeros_like(grid_a)
policy = np.zeros_like(grid_a)
```

Iterate over the Bellman equation:

```
def bellman(value_func_interp, a):
    income = (1 + r) * a + w
    def obj(c):
        a_next = income - c
```

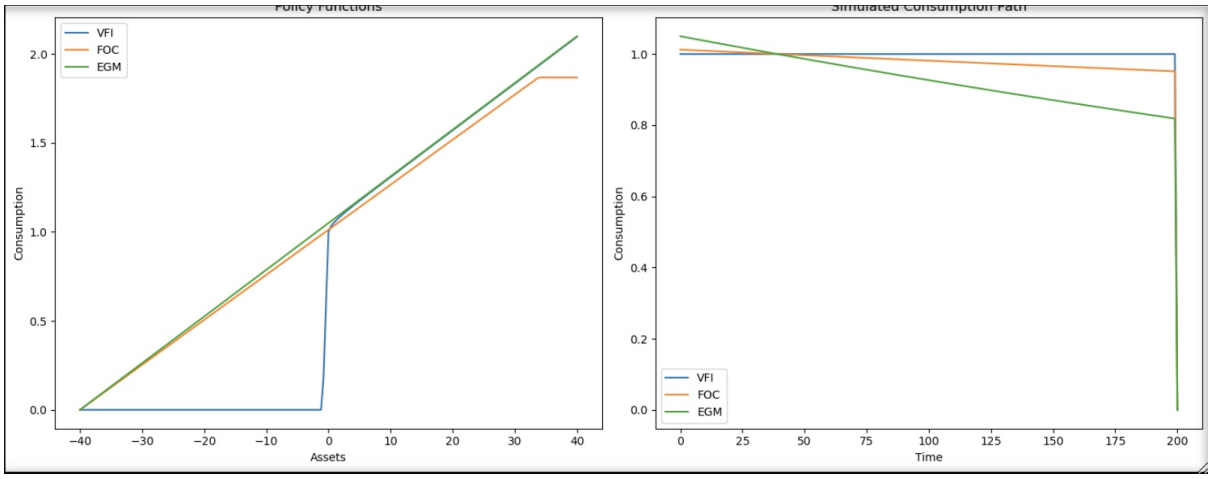
```
        return -(utility(c) + beta * value_func_interp(a_next))
    res = minimize_scalar(obj, bounds=(1e-10, income), method='bounded')
    return -res.fun, res.x
```

Interpolate:

```
value_func_interp = interp1d(grid_a, V, kind='linear', fill_value='
    extrapolate')
```

And repeat until convergence.

Results: The VFI converged in 575 iterations. As we can see in the resulting policy function (right graph), optimal consumption increases with assets, but is concave and has a binding borrowing constraint where households cannot go below zero assets. We see a sharp increase in consumption when assets become positive. Because this method solves the Bellman equation directly, it maximizes current utility plus discounted future value, leading to a stable consumption path.



To test an increase in interest, $r = 0.03$ was not compatible with the previous code, so it was tested with $r = 0.025$ instead. This still showed clear results that are compatible with expectations for higher returns to savings, as all models show a higher propensity to save, consuming less today than tomorrow. In addition, all of them caused an increase in overall asset levels over time.

Question 8.5

Implement the same solution using root-finding and interpolation.

To find the optimal consumption levels for all asset levels, we find the asset choice a' that satisfies the FOC (the Euler equation).

$$u'(c_t) = \beta(1+r)\,u'(c_{t+1}) \tag{5}$$

In order to do this, we iterate over the Euler equation:

```
def foc(x, a_i, spline_c):
    c_plus = spline_c(min(x, grid_a[-1]))
    rhs = (beta * (1 + r)) ** (-gamma) * c_plus
    lhs = (1 + r) * grid_a[a_i] + w - x
    return lhs - rhs
```

We then update the policy function and repeat until convergence:

```
sol = root_scalar(foc, args=(i, spline_c), bracket=(a_min, income), method=
    'brentq')
c_new[i] = income - sol.root
```

Results: The FOC methods converged in 456 iterations. We see a gradual decline in the consumption path of the FOC, as the agent will optimally consume less in the future in order to have more consumption now as $\beta(1+r)$ in this case is < 1.

Question 8.6

Implement the same solution using the method of endogenous grid points.

In this method, we invert the Euler equation to construct the grid for future assets a' and then work backwards to find what today's consumption choices should optimally be.

$$c_t = [\beta(1+r)\,u'(c_{t+1})]^{-1} \tag{6}$$

In order to do this, we compute marginal utility and invert:

```
mu_next = marg_utility(c_next)
c_now = inv_marg_utility(beta * (1 + r) * mu_next)
```

Back out current assets (`a_t`) from future values (`grid_a`):

```
a_now = (c_now + grid_a - w) / (1 + r)
```

Interpolate to get the consumption function over the regular grid and repeat until convergence:

```
c_new = np.interp(grid_a, a_now_sorted, c_now_sorted)
```

Results: EGM converged in 360 iterations. This was found to be the fastest of all the methods. The simulated consumption path shows that it doesn't lend itself to consuming constantly over all 200 periods, but instead has high consumption initially that lowers drastically over time.

Question 8.7

Assume that the agent can save as much as he wants, but is only allowed to borrow against future income to a certain limit $\bar{a}$. Consequently as an additional restriction $a_t \geq -\bar{a}$ has to hold for all t. Try to implement this in your solution with minimization and interpolation. Test your program for different values of $\bar{a}$ and r. How does it behave?

When implementing the borrowing restriction $\bar{a}$, we are ensuring that the agent can only borrow a certain amound and that next period's assets stay above a certain floor. In order to do this, we adjust the asset grid to start at the borrowing limit:

```
gamma = 0.5
beta = 0.975
r = 0.028
w = 1
a_bar = 4 #borrowing limit, assets must be => -a_bar

# grid setup
a_min = -a_bar
a_max = w / r
grid_a = np.linspace(a_min, a_max, 501)
tol = 1e-6
max_iter = 1000
T_sim = 200
```

Then, in the Bellman function we restricted the feasible choices for a':

```python
def bellman(value_func_interp, a):
    income = (1 + r) * a + w
    c_min = 1e-10
    c_max = income - a_min  # ensure a'     - a


    def obj(c):
        if c <= 0 or c > c_max:
            return 1e10
        a_next = income - c
        if a_next < a_min:
            return 1e10  # enforce borrowing constraint
        return -(utility(c) + beta * value_func_interp(a_next))

    res = minimize_scalar(obj, bounds=(c_min, c_max), method='bounded')
    return -res.fun, res.x
```

Higher Interest Rate: When we increase the interest to $r = 0.025$, savings become more rewarding and consumption in the short term is decreased in order to increase savings. This is reflected in the Simulated Consumption Path plot below, where the figure with $r = 0.025$ starts at a lower rate consumption and decreases that consumption slowly over time.

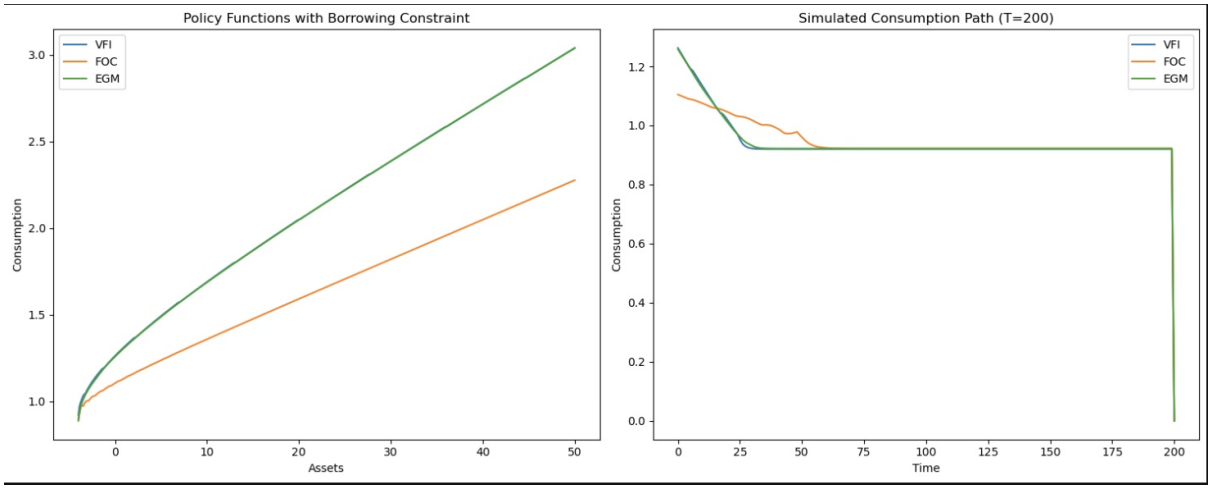


Figure 2: $r = 0.02$

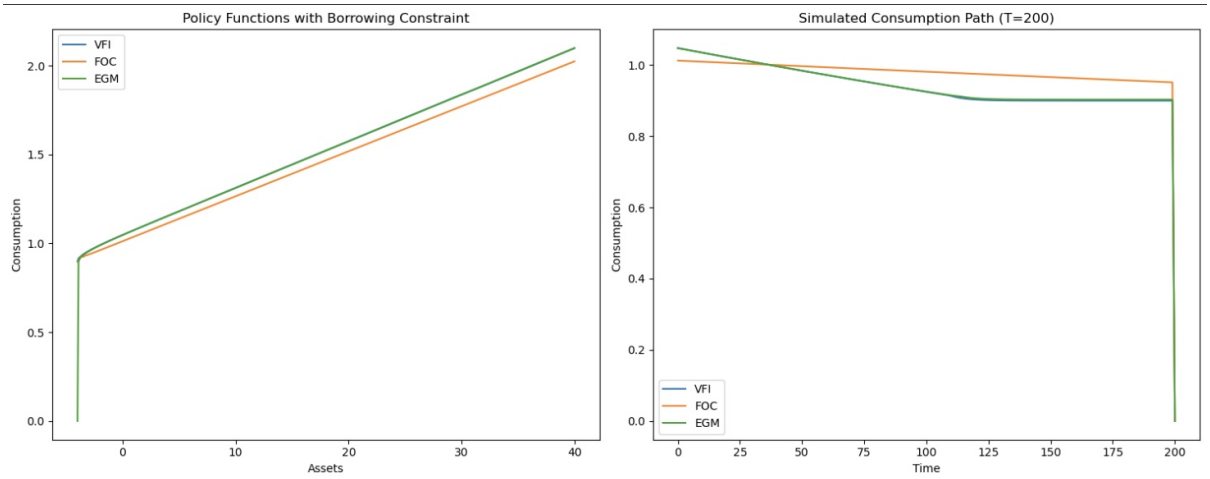


Figure 3: $r = 0.025$

Borrowing Limit: When the borrowing limit is increased, we can expect to see a shift up in consumption at lower levels of assets. These expectations are reflected in the graphs displayed below, of which the first has $\bar{a} = 4$ and the second $\bar{a} = 10$.

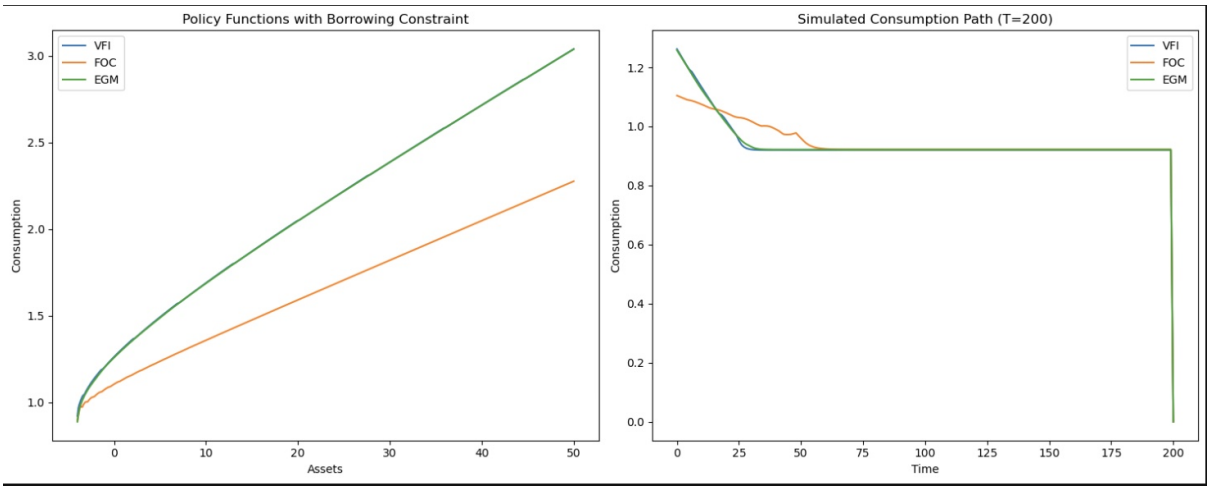


Figure 4: $\bar{a} = 4$

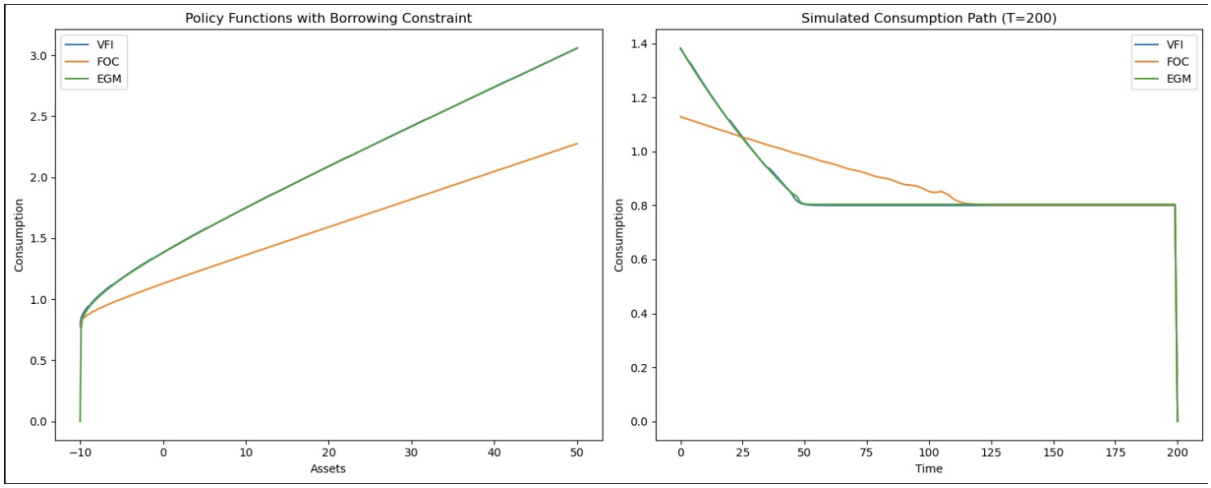


Figure 5: $\bar{a} = 10$

Question 8.8

Can you think of a way of implementing the constraint $\bar{a}$ in the root-finding and interpolation solution?

With the borrowing constraint, the agent can choose at most for each a:

$$c \leq (1+r)a + w + \bar{a}$$

Therefore, we restrict the root-finding bracket:

```
for i in range(len(grid_a)):
        income = (1 + r) * grid_a[i] + w
        a_next_min = a_min
        a_next_max = income
        try:
            sol = root_scalar(foc, args=(i, spline_c),
                              bracket=(a_next_min, a_next_max), method='
   brentq')
            c_new[i] = income - sol.root
        except ValueError:
            c_new[i] = 1e-10
```

Question 8.9

Can you think of a way of implementing the constraint $\bar{a}$ in the solution that uses the method of endogenous gridpoints?

In our previous model, we inverted built the implicit asset grid on the inverted Euler equation. In order to achieve implement the borrowing constraint, we first set

```
a_bar = 4
a_min = -a_bar
```

For computing `a_now`, clip values to be feasible:

```
    a_now = (c_now + grid_a - w) / (1 + r)
    a_now = np.maximum(a_now, a_min)  # enforce constraint
```

And then interpolate.