Quantitative Macroeconomics & Numerical Methods

Assignment 1

Louis Brune

ID: 8514718

Luca Seibert

ID: 8511270

João Cordeiro

ID: 8509509

October 30, 2025

Exercise 1 - Value and Policy Function Iteration

Solution of the full discretized Ramsey problem computationally. For the first question, we fixed the following parameters:

- $\beta = 0.99$
- $\alpha = 0.3$
- No. grid points = 500

For the grid we first compute the steady state value of capital (=0.4335510497247844) and locate our grid from 10e-10 to 1.5 $\times$ the steady state value of capital (`k_ss`).

The entire code was run on a machine with the following hardware configurations:

- Intel(R) Core(TM) Ultra 7 155H (1.40 GHz)
- 32GB RAM

a) Analytical Solution

i) Verifying Optimal Policy Directly via Euler

We can leverage the type of the problem by plugging the suggested policy for capital into the Euler equation. Because of the log utility we get a nice closed form expression:

$$\frac{1}{(1-a)k^{\alpha}} = \beta\alpha k_{prime}^{\alpha-1}\frac{1}{(1-a)k_{prime}^{\alpha}} \tag{1}$$

Which we can rewrite into:

$$k_{prime} = \beta\alpha k^{\alpha} \tag{2}$$

and hence verifies our guess with a = $\alpha\beta$

ii) Analytical Solution: Guess and Verify

The representative agent solves:

$$V(k) = \max_{k'}\Big\{\log(k^{\alpha} - k') + \beta V(k')\Big\}. \tag{3}$$

From the Euler equation, the optimal policy is known to be:

$$k' = \beta\alpha k^{\alpha}, \tag{4}$$

so that consumption is given by:

$$c = (1 - \beta\alpha)k^{\alpha}. \tag{5}$$

Substituting the optimal policy into the Bellman equation yields:

$$V(k) = \log\Big((1 - \beta\alpha)k^{\alpha}\Big) + \beta V\Big(\beta\alpha k^{\alpha}\Big). \tag{6}$$

Guess the Functional Form

We conjecture that the value function takes the log-linear form:

$$V(k) = b_0 + b_1 \log k, \tag{7}$$

where b_0 and b_1 are constants to be determined.

Substitute the Guess into the Bellman Equation

Substituting $V(k) = b_0 + b_1 \log k$ and $V(k') = b_0 + b_1 \log k'$:

$$b_0 + b_1 \log k = \log\Big((1 - \beta\alpha)k^{\alpha}\Big) + \beta\Big[b_0 + b_1 \log\Big(\beta\alpha k^{\alpha}\Big)\Big]. \tag{8}$$

Expand the Logarithms

Using $\log(ab) = \log a + \log b$:

$$\log\Big((1 - \beta\alpha)k^{\alpha}\Big) = \log(1 - \beta\alpha) + \alpha \log k, \tag{9}$$

$$\log\Big(\beta\alpha k^{\alpha}\Big) = \log(\beta\alpha) + \alpha \log k. \tag{10}$$

Substituting these back gives:

$$b_0 + b_1 \log k = \Big[\log(1 - \beta\alpha) + \alpha \log k\Big] + \beta\Big[b_0 + b_1(\log(\beta\alpha) + \alpha \log k)\Big]. \tag{11}$$

Expand and Collect Terms

Expanding the right-hand side:

$$b_0 + b_1 \log k = \log(1 - \beta\alpha) + \alpha \log k + \beta b_0 + \beta b_1 \log(\beta\alpha) + \beta b_1 \alpha \log k. \tag{12}$$

Collect terms with and without $\log k$.

(i) Coefficient on $\log k$

$$b_1 = \alpha + \beta b_1 \alpha. \tag{13}$$

Solving for b_1:

$$b_1(1 - \beta\alpha) = \alpha \quad \Rightarrow \quad b_1 = \frac{\alpha}{1 - \beta\alpha} \tag{14}$$

(ii) Constant Terms

$$b_0 = \log(1 - \beta\alpha) + \beta b_0 + \beta b_1 \log(\beta\alpha). \tag{15}$$

Rearranging:

$$b_0(1 - \beta) = \log(1 - \beta\alpha) + \beta b_1 \log(\beta\alpha), \tag{16}$$

so that:

$$b_0 = \frac{\log(1 - \beta\alpha) + \beta b_1 \log(\beta\alpha)}{1 - \beta} \tag{17}$$

Substitute b_1 **into** b_0

Replacing $b_1 = \dfrac{\alpha}{1 - \beta\alpha}$:

$$b_0 = \frac{\log(1 - \beta\alpha) + \dfrac{\beta\alpha}{1 - \beta\alpha} \log(\beta\alpha)}{1 - \beta}. \tag{18}$$

Final Analytical Solution

$$\boxed{\begin{aligned} V(k) &= b_0 + b_1 \log k, \\ b_1 &= \frac{\alpha}{1 - \beta\alpha}, \\ b_0 &= \frac{\log(1 - \beta\alpha) + \beta b_1 \log(\beta\alpha)}{1 - \beta}, \\ k' &= \beta\alpha k^{\alpha}, \\ c &= (1 - \beta\alpha) k^{\alpha}. \end{aligned}} \tag{19}$$

Numerical Example

For $\beta = 0.99$ and $\alpha = 0.3$:

$$b_1 = \frac{0.3}{1 - 0.99 \times 0.3} = 0.4267, \tag{20}$$

$$b_0 = \frac{\log(1 - 0.297) + 0.99 \times 0.4267 \times \log(0.297)}{1 - 0.99} \approx -86.53. \tag{21}$$

Hence:

$$V(k) \approx -86.53 + 0.4267 \log k. \tag{22}$$

b) Value Function Iteration with Full Discretization

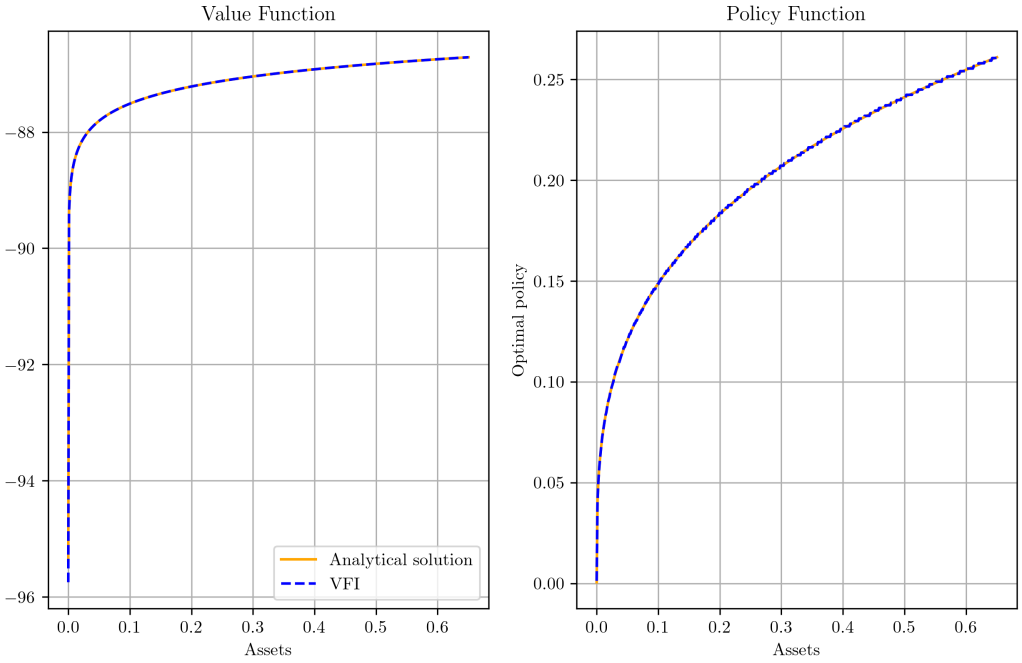

Figure 1: Comparing VFI to the analytical solution

c) Log-spaced grid & d) Power grid

Detour Asset Grid

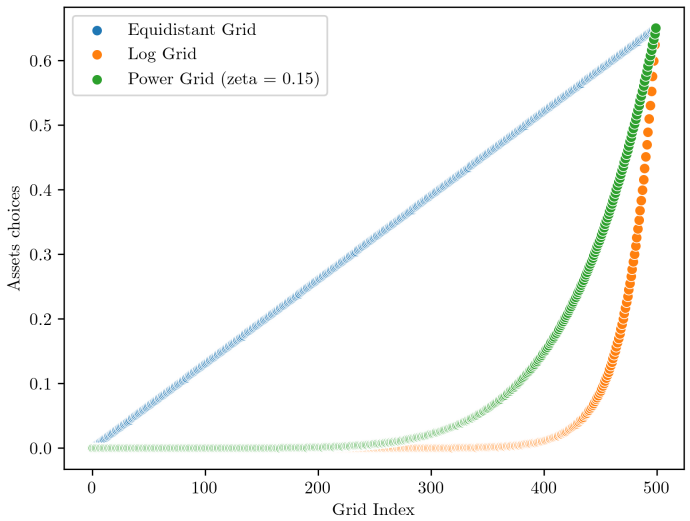

Figure 2: Location of different Grid Methods

Here, we can clearly see the locations of the three grid methods. The log grid allocates the most points at the beginning, while the power method allocates more towards the end compared to the log grid. Both methods allocate more grid points closer to the beginning of the asset space than the equidistant grid does.

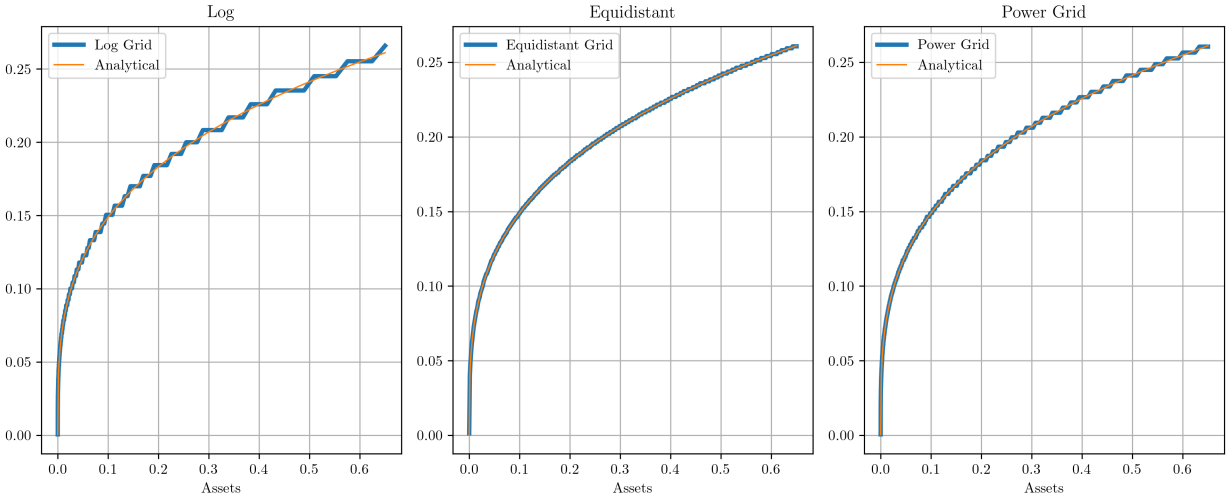

Figure 3: Visualization of policy functions produced by different grids

Judging from Figure 3, the log grid is not a higher quality approximation. As seen on Figure 2, log and power grid have more points located towards the beginning of the grid.
However, it is hard to judge based on the chart alone. Let us analyze numerically:
Deviation from analytical solution:

- Equidistant grid: mean: 1.2939873823916904e-05, max: 0.0008030432816054089
- Log grid: mean: 0.15766414962406053, max: 0.004779946168767646
- Power grid: mean: 0.11349537563391897, max: 9.110183680892652e-06

All in all, the equidistant grid minimizes the mean loss - as expected since we distribute the grid points evenly. In terms of maximum deviation, the power grid performs best. Here, the idea is that we allocate more grid points to the region where the value function has a steeper curve (where the VFI is more prone to numerical estimation error) and help with more grid points. As a result, we minimize the maximum deviation of all three methods. The log grid does not seem to work well in this scenario, being the worst performer in both in terms of mean and max deviation.

The reason for that could be that the log grid allocates too many points towards the error-prone region of the value function, and that in turn allocates few points towards the end of the value function. We can see in Figure 3 that the log grid produces a policy function that is very wiggly at the end of the asset grid. While also not yielding a significant improvement at the beginning of the grid.

e) Economic Intuition (Concavity & Monotonicity)

We can make use of concavity and monotonicity of the value function in the following version:

- **Concavity:** given the current capital space, for each iteration, break the code as soon as the value function starts decreasing for the choice grid. The idea here is that by concavity of the value function, declining values imply that we passed already the maximum, hence capital choice values after that will not be optimal.
- **Monotonicity of the policy:** after finding an optimal policy for the first capital value, by monotonicity, the next capital state value must have at least a higher savings value. Thus, while iterating, we can bound the search grid from below.

Nevertheless, the great disadvantage is that we rely on for loops and cannot vectorize the process. Hence, we expect a runtime loss compared to the VFI with vectorization.

```
VFI: 1.3430853000027128 s
VFI_exploited: 2.6614179000025615 s
VFI_loop: 3.0867252000025474 s
```

To summarize, we see a slight increase in time for the exploited VFI; however, the vectorized VFI still performs the best in this scenario.

f) Howard's Improvement

When computing Howard's improvement, there are two possible ways:

- After computing an optimal policy, iterate a fixed number of times on the value function, maintain the policy fixed, and then optimize the policy again.
- Or, invert the system of equations after computing the optimal policy (instead of iterating on it). The latter follows the approach discussed in Heer and Maussner (2024), and utilizes that solving the system of equations is analogously to iterating infinitely many times on the value function.

We shall compare both methods and their respective runtimes. Note that the method where we solve the system of equations (known as the exact Howard method) must indeed solve the system, which can be computationally expensive. However, it obtains the full benefit of the fully iterated value function (a partial mix of policy and value function iteration is also possible, but this is neglected here). Comparing Runtimes:

```
VFI: 1.3430853000027128 s
VFI_exploited: 2.6614179000025615 s
VFI_loop: 3.0867252000025474
Howard: 0.1431596000038553 s
Howard_exact: 0.1610899000079371 s
```

Comparing Iterations:

```
VFI: 1376
VFI_exploited: 1376
VFI_loop: 1376
Howard: 9
Howard_exact: 9
```

Note that the iteration count for Howard's improvement algorithm only takes the outer loop into account and not the policy iteration (150 for Howard). The time stamp measures both. Comparing across all methods, Howard's improvement with 150 policy iteration is the fastest algorithm. This indicates that iterating 150 times on the Value function with the running policy is faster than inverting the system of equations (exact Howard).

Exercise 2 - Euler Equation Methods

We now turn to a different family of solution methods based on the optimality conditions given by the Euler Equation, and compare them to the classical VFI. For the Euler methods we use equidistant grids of n = 200 and for the VFI equidistant of n = 2000. Here we our aim is to analyze runtime and accuracy of the policy function.

Note that we fixed the guesses for the methods in the following way:

- The policy guess for the Euler methods is to save 10% of output.
- For the VFI to be comparable, we start with a naive, but better guess then zero, assuming that k is equal to k_{prime}.

We can then rewrite the value function into our guess:

$$V(k) = \frac{U(f(k) - k)}{1 - \beta} \tag{23}$$

a) Time Iteration

For this question, we choose cubic interpolation connected with Newton's method for root finding.

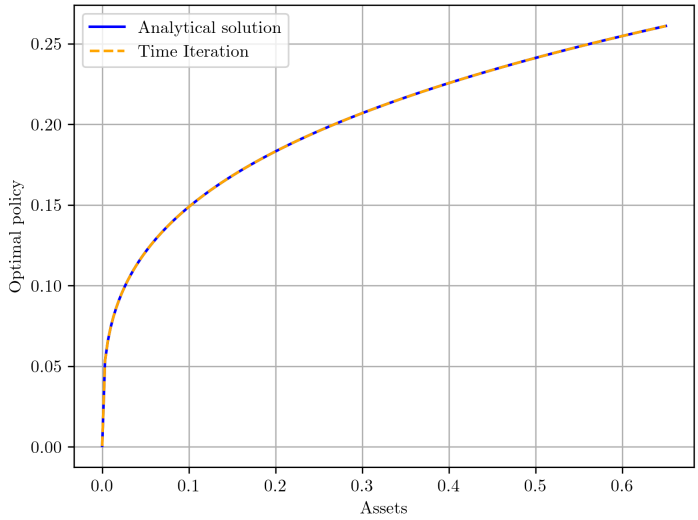

Figure 4: Comparing Time Iteration to the analytical solution

Maximum deviation from analytical solution for time iteration policy: 4.981229317913005e-11

For comparison, the deviation from the analytical solution for the VFI was: 0.0008030432816054089.

Caveat: We are comparing two different methods on two different grids. The VFI was defined on a 500 point grid and thus has potentially more errors. Nevertheless, an extreme improvement.

b) Fixed-Point Iteration

Using a dampening factor of 0.7

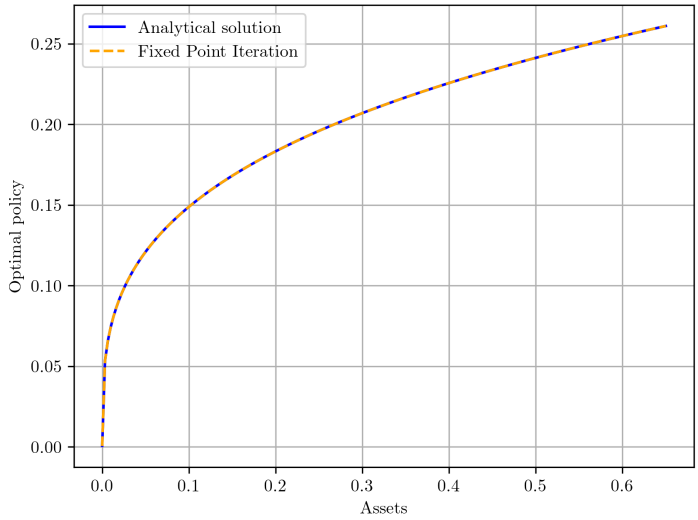

Figure 5: Comparing Fixed Point Interation to the analytical solution

Maximum deviation from analytical solution for fixed point iteration policy: 3.091669206756187e-08

c) Endogenous Grid Points

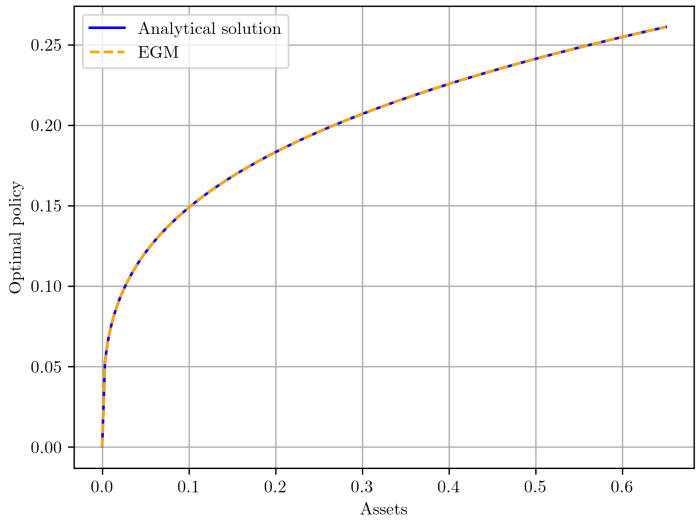

Figure 6: Comparing EGM to the analytical solution

Maximum deviation from analytical solution for EGM policy: 3.294334741110738e-07

d) Speed and Accuracy

Method	Time (s)	Iter	MeanEE	MaxEE
VFI	0.2016	7	7.17e-04	1.27e-01
TI	0.4354	14	6.37e-09	6.39e-09
FPI	0.1150	13	2.49e-04	4.92e-02
EGM	0.1204	22	3.42e-01	6.84e+01

Table 1: Comparison between the different methods

Note that the VFI uses a naive guess that k is equal to k_{prime}. This guess works quite well. Using the naive zero value function guess leads to a computation time of: 18.669102699961513s and 1376 iterations to converge.

e) Euler Equation Errors

The Euler Error reduces significantly in both the mean and max computation:

```
equidistant meanEE: 0.3420452278738818, maxEE: 68.40356099216702
log-grid meanEE: 0.0007439862086958013, maxEE: 0.001434840537035588
```

References

Heer, B. and Maussner, A. (2024). *Dynamic General Equilibrium Modeling: Computational Methods and Applications*. Springer Texts in Business and Economics. Springer, 3rd edition.