Métodos numéricos - Lista 2

João Vitor Dias Gonçalves

1º trimestre 2023

Question 1

The planner problem can be written as:

$$V(k,z) = \max_{c,k'} u(c) + \beta \mathbb{E} V(k',z') \tag{1}$$

$$s.t. \;\; c + k' = zk^{\alpha} + (1-\delta)k \tag{2}$$

$$\log z' = \rho \log z + \epsilon, \quad \epsilon \sim N(0,\sigma^2)$$

Where I transformed the production function to a capital per capita function.

Question 2

Substituting the restriction in the problem I have:

$$V(k,z) = \max_{c,k'} u(zk^{\alpha} + (1-\delta)k - k') + \beta \mathbb{E} V(k',z') \tag{3}$$

Deriving in k' I find the FOC:

$$-u'(zk^{\alpha} + (1-\delta)k - k') + \beta \mathbb{E} V'(k',z') \tag{4}$$

and then applying the envelope theorem

$$V'(k',z') = (\alpha z' k'^{\alpha-1} + (1-\delta)) u'(z'k'^{\alpha} + (1-\delta)k' - k'') \tag{5}$$

Uniting (4) and (5)

$$u'(zk^{\alpha} + (1-\delta)k - k') + \beta \mathbb{E} V'(k',z') = \beta \mathbb{E}(\alpha z' k'^{\alpha-1} + (1-\delta)) u'(z'k'^{\alpha} + (1-\delta)k' - k'')$$

We know that $u'(c) = c^{-\mu}$ and using the fact that in the steady state $k = k' = k''$ we finally have:

$$k_{ss} = \frac{\beta\alpha}{1-\beta(1-\delta)}^{\frac{1}{1-\alpha}} \tag{6}$$

Question 3

As can be seen in my code, auxiliary functions, such Tauchen, consumption and utility are in the $function$ archive. Following I present both the state grid and the transition probability matrix.

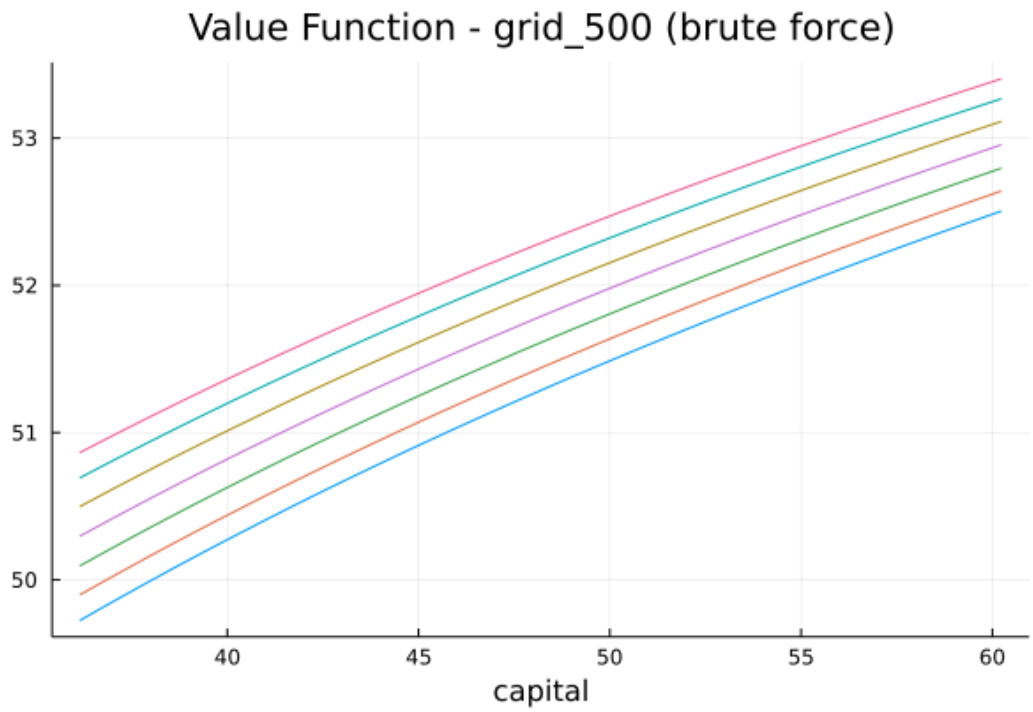
$$\begin{bmatrix} 0.86883 & 0.13116 & 1.0e-5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.02733 & 0.87258 & 0.10009 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.03908 & 0.88614 & 0.07477 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.05466 & 0.89069 & 0.05466 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.07477 & 0.88614 & 0.03908 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.10009 & 0.87258 & 0.02733 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0e-5 & 0.13116 & 0.86883 \end{bmatrix}$$

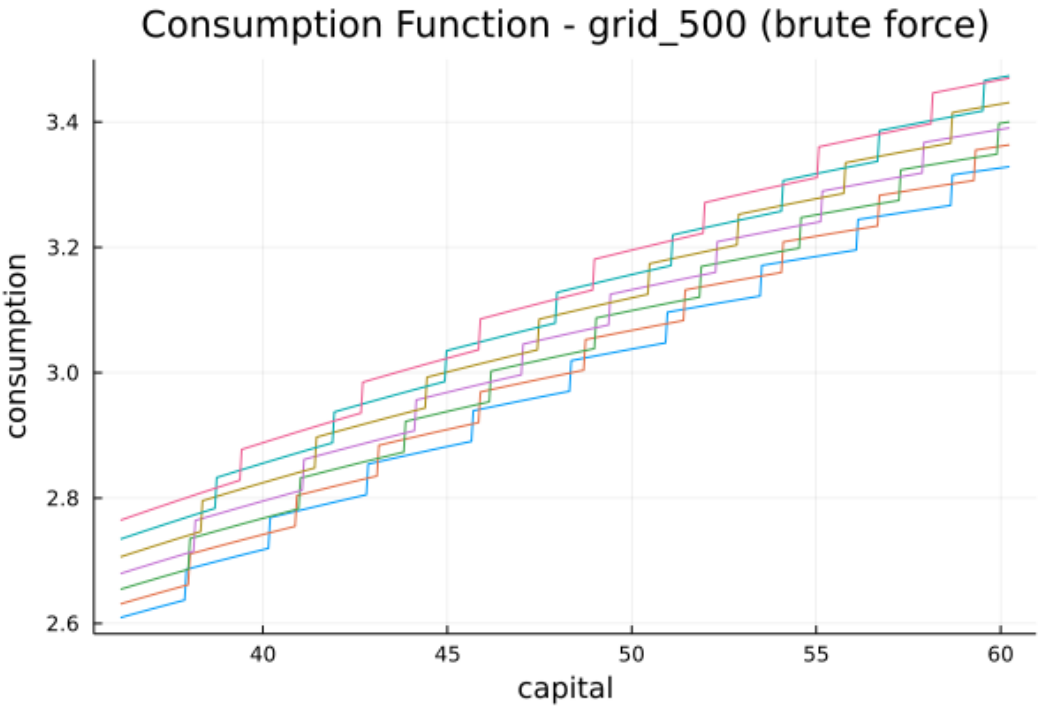
$$\begin{bmatrix} -0.06725 \\ -0.04484 \\ -0.02242 \\ 0.0 \\ 0.02242 \\ 0.04484 \\ 0.06725 \end{bmatrix}$$

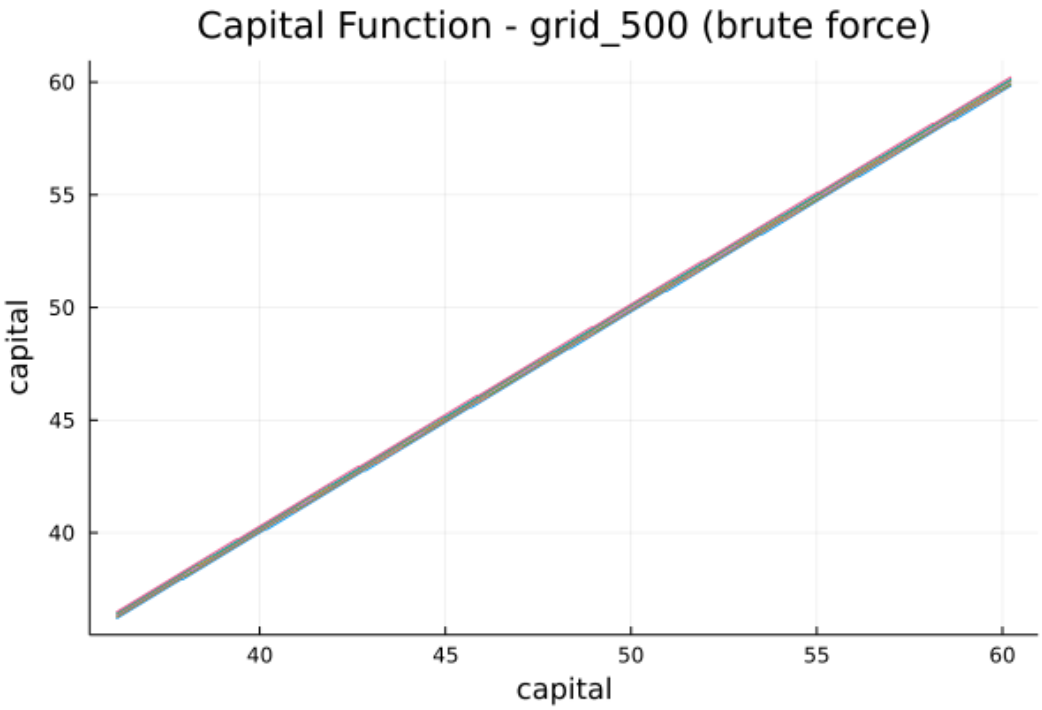
So I could compare the time and result I opted to construct two different functions to solve the model. The first one, called $brute\ force$, where I constructed a 3d array for the utility function and using a matrix of zeros as initial guess, it converged in 848 iterations taking 34 seconds. The second one I used the fact that the value function is monotone and concave. I created a monotonicity counter, so in the k' loop it is bounded below by the last capital chosen. for concavity I just check if the function is bigger than the one from the last capital. This one took 6 seconds to run the same 848 iterations. Since the result for both of them are the same, I will show the plots just for the *concave mon* function. It is important to notice that I had to exponentiate the state grid generated by the Tauchen function, since we have a log normal processes for the error. At the end of this work there is a table with the summary of all methods tested with the time and mean EEE of each one. When the grid has 500 points the mean EEE is of -3, what means one dollar of error for each 1000 dollar spent. It reduces to -3.5 when we have 5000 points.

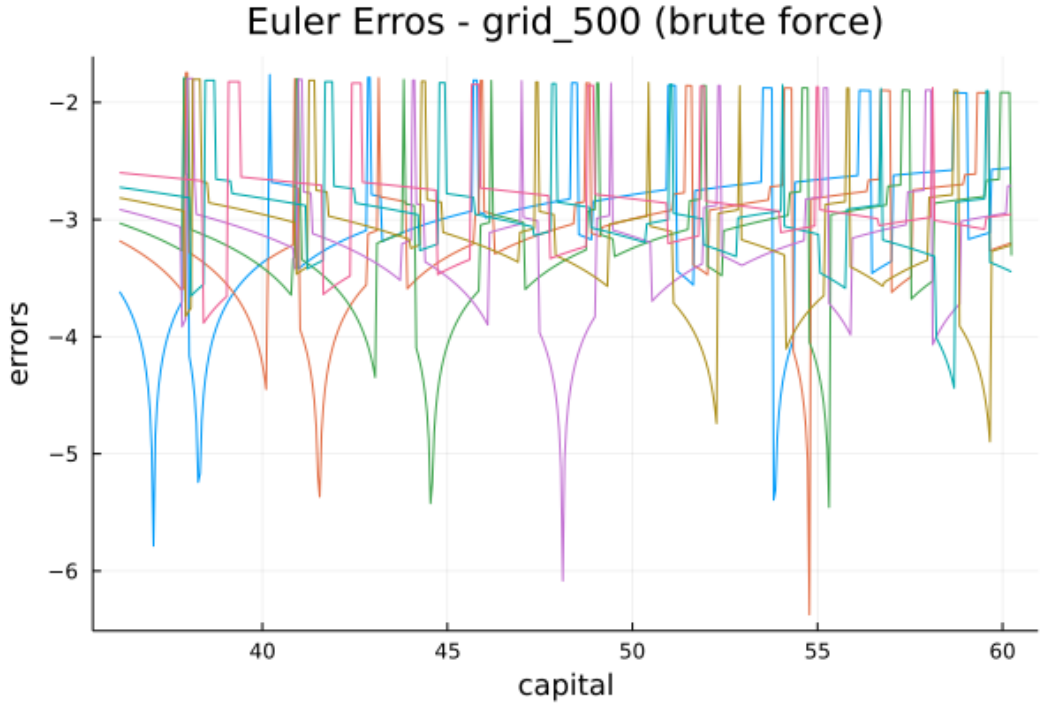
Question 4

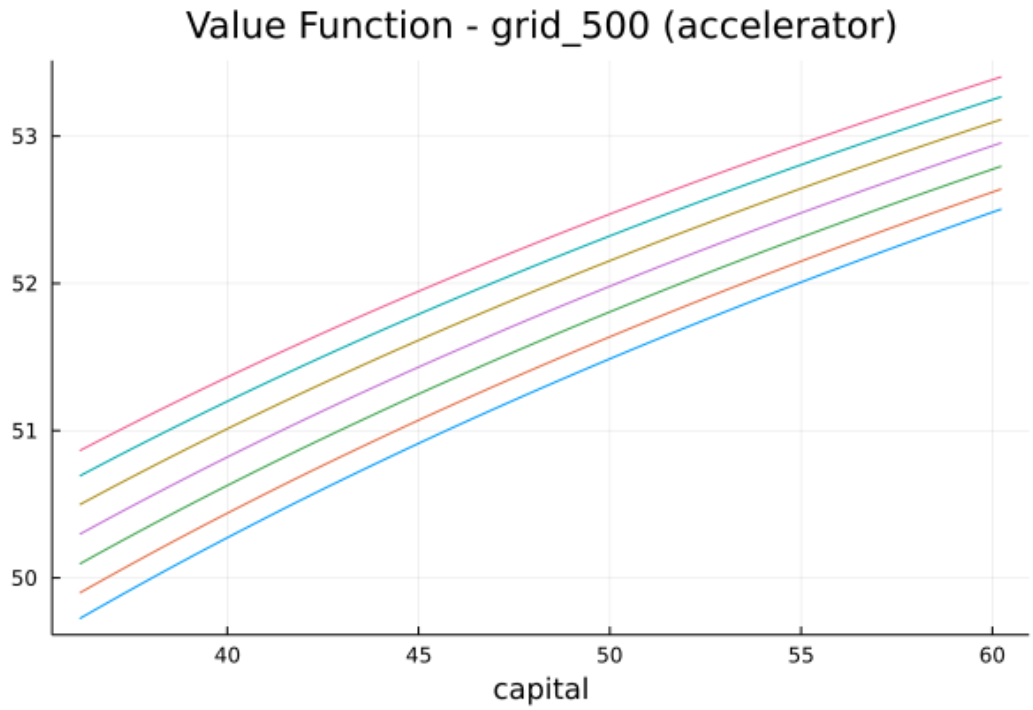
Again I created two functions, using the ones from the first exercise. Since we are maximizing just 10% of the time and in the beginning of the iterations is where the value functions changes more, I opted to maximize more in the firsts iterations. So the function has a default vector, that can be changed, that has 5% of the grid length as a sequence without holes. The other half of the vector has equally distanced numbers finishing in the last number of iterations, in our case 10.000. Saying in other words, the vector has length 1.000, beginning with a sequence from 1 to 500, then the following 500 numbers have a interval of 19 numbers between them. The function just do the maximization if the iteration counter is in this vector, otherwise it will just add the utility, using the last capital chosen, and the last value function calculated. Both functions use the same accelerator mechanism in my code. The brute force one took 28 seconds and the other one took 5 seconds. Naturally it was faster than in the last exercise, but the the difference wasn't that big. Again I'm posting just the plots for the concave and monotonic, since they are the same.

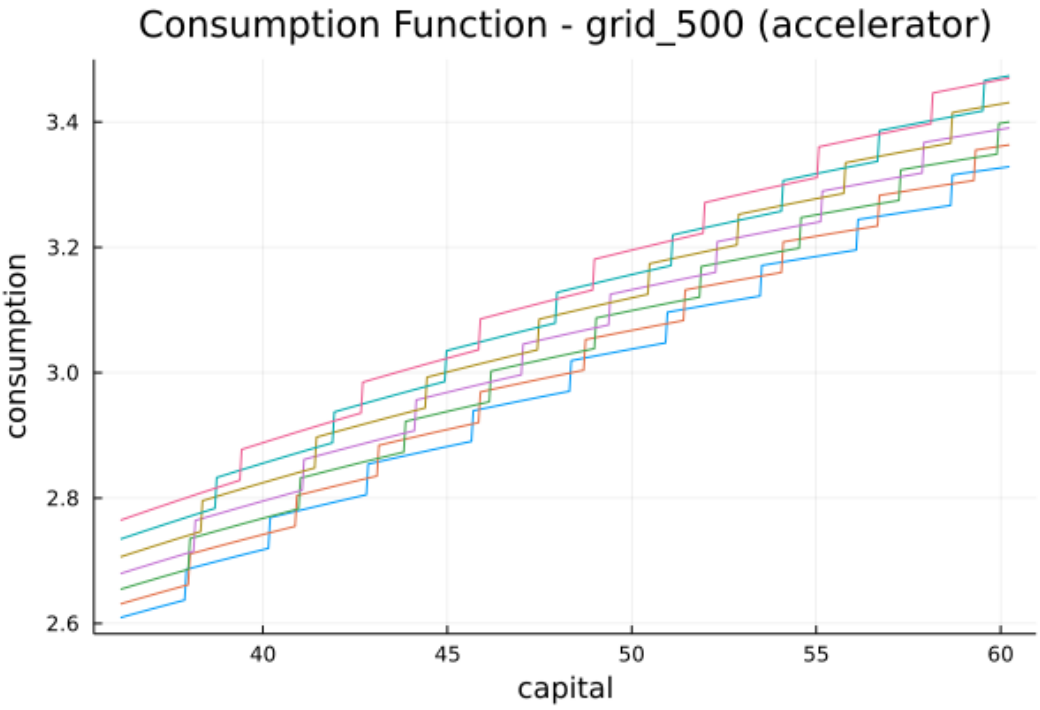

Value Function - grid_500 (brute force)
53
52
51
50
40
45
50
55
60
capital

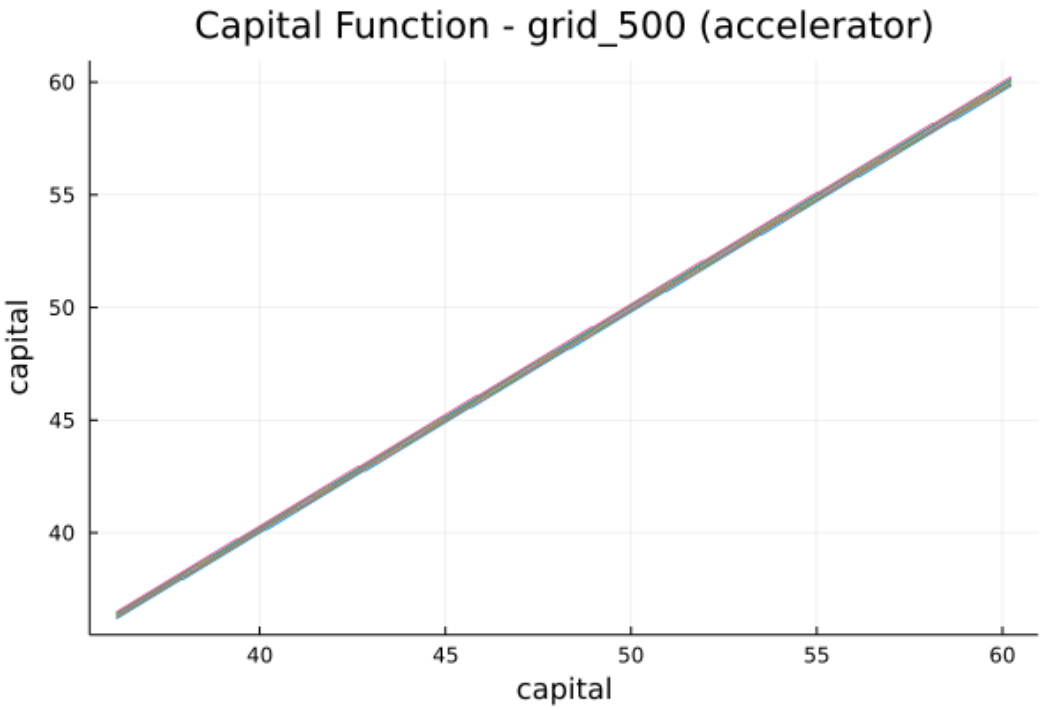

Consumption Function - grid_500 (brute force)
3.4
3.2
3.0
2.8
2.6
consumption
40
45
50
55
60
capital

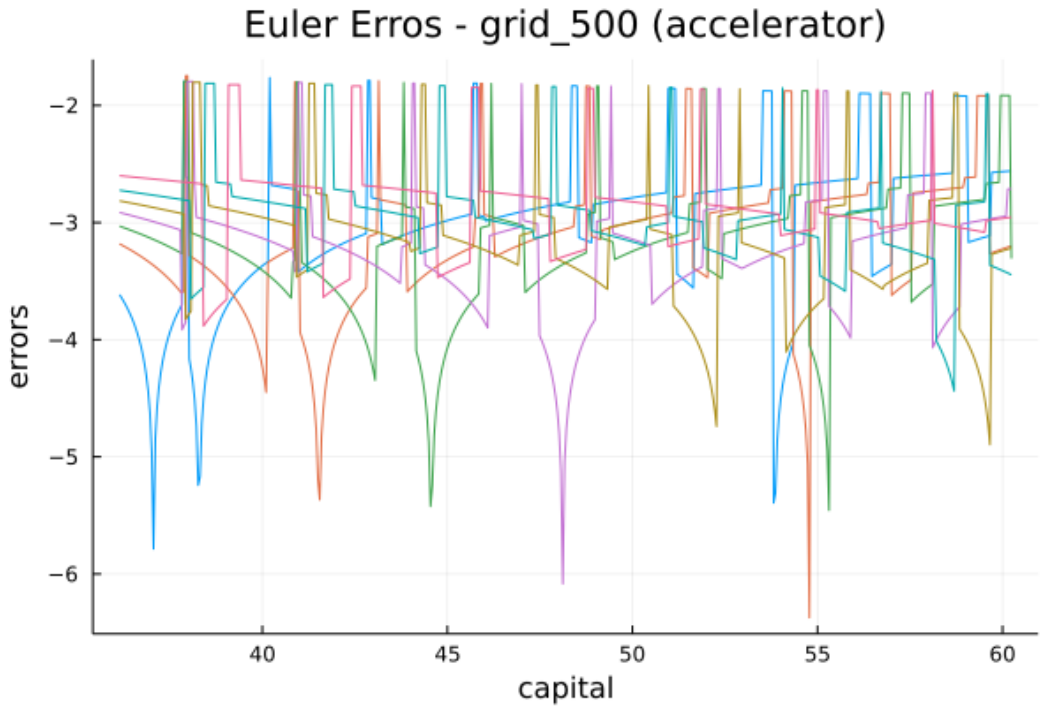

Capital Function - grid_500 (brute force)
capital
60
55
50
45
40
40
45
50
55
60
capital


Euler Erros - grid_500 (brute force)
errors
−2
−3
−4
−5
−6
40
45
50
55
60
capital


Value Function - grid_500 (accelerator)
53
52
51
50
40
45
50
55
60
capital

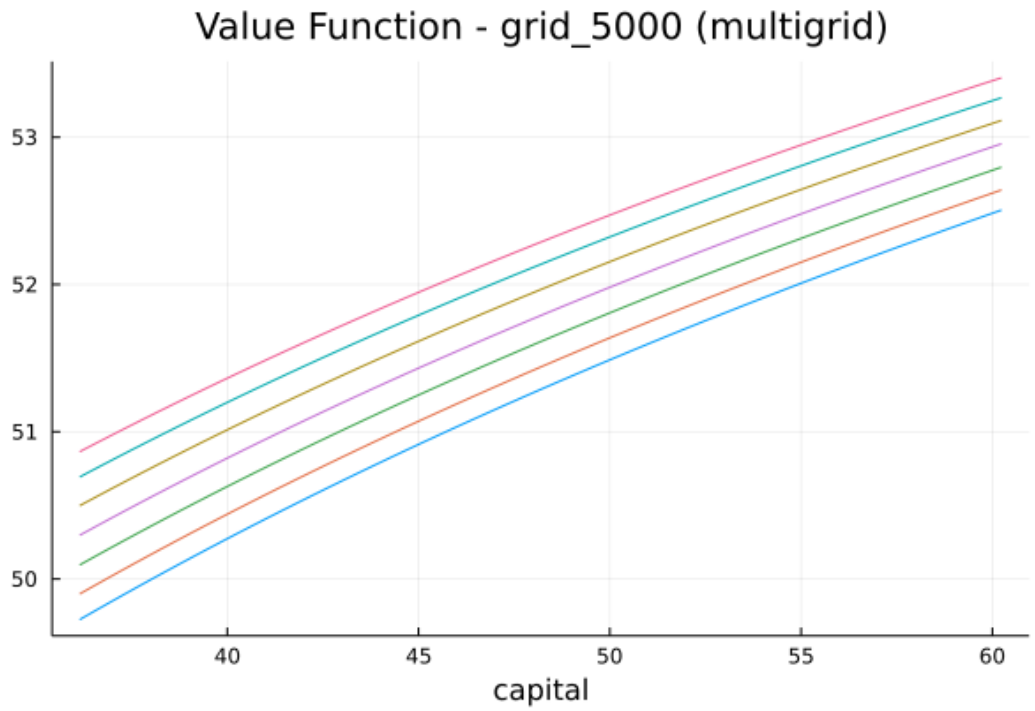

Consumption Function - grid_500 (accelerator)
3.4
3.2
3.0
2.8
2.6
consumption
40
45
50
55
60
capital

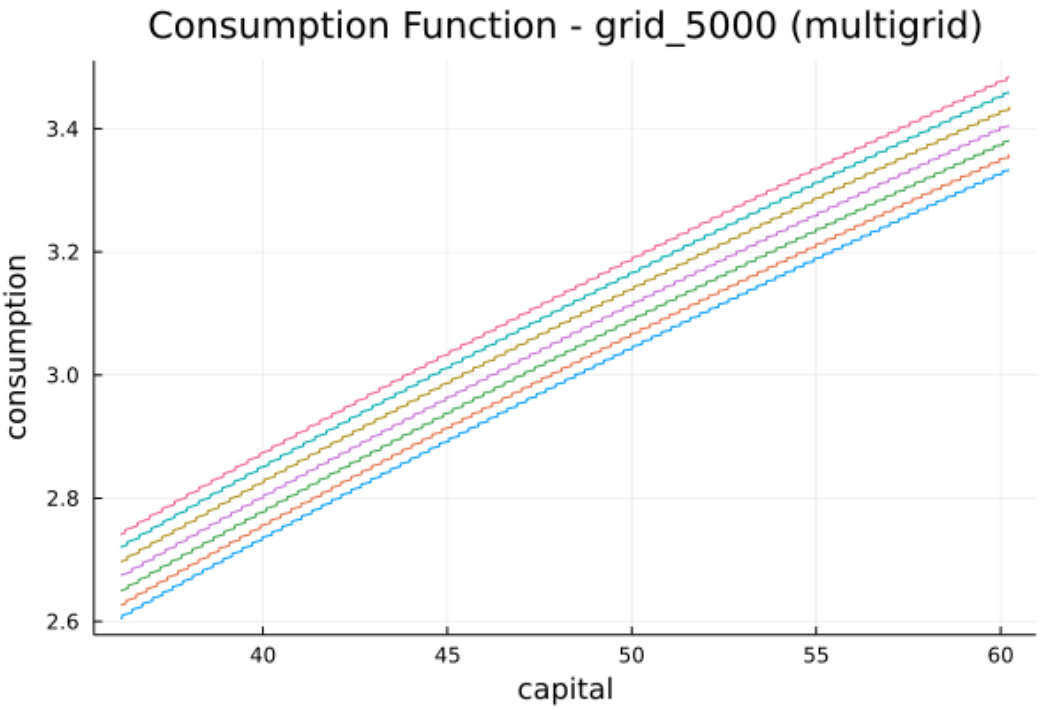

Capital Function - grid_500 (accelerator)
capital
60
55
50
45
40
40
45
50
55
60
capital

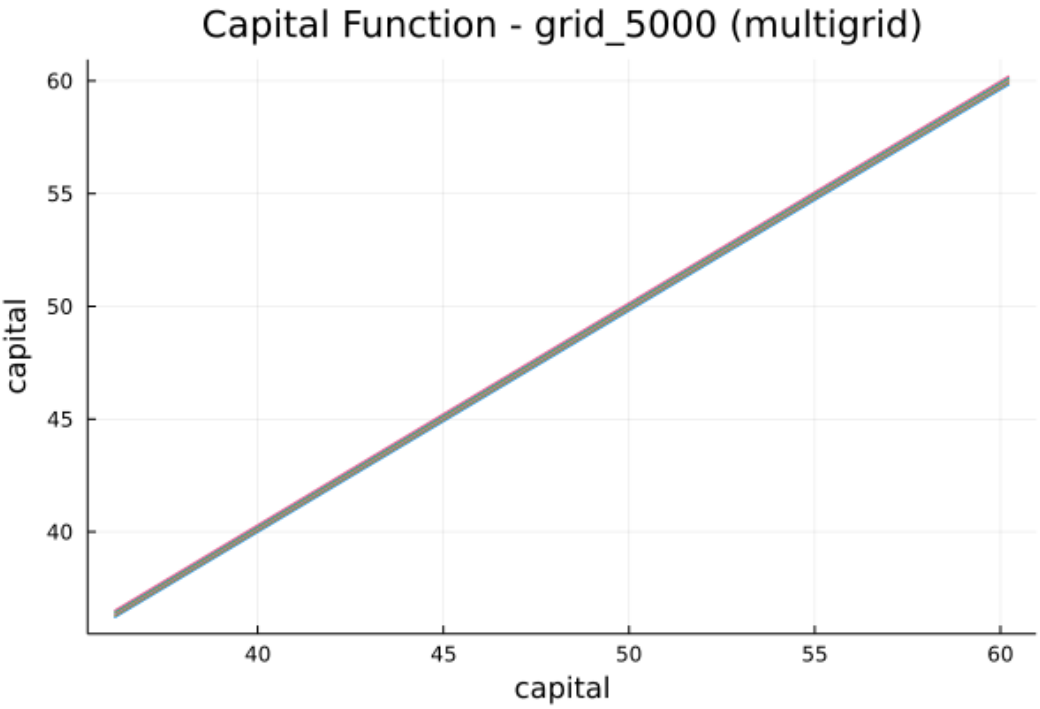

Euler Erros - grid_500 (accelerator)
errors
−2
−3
−4
−5
−6
40
45
50
55
60
capital

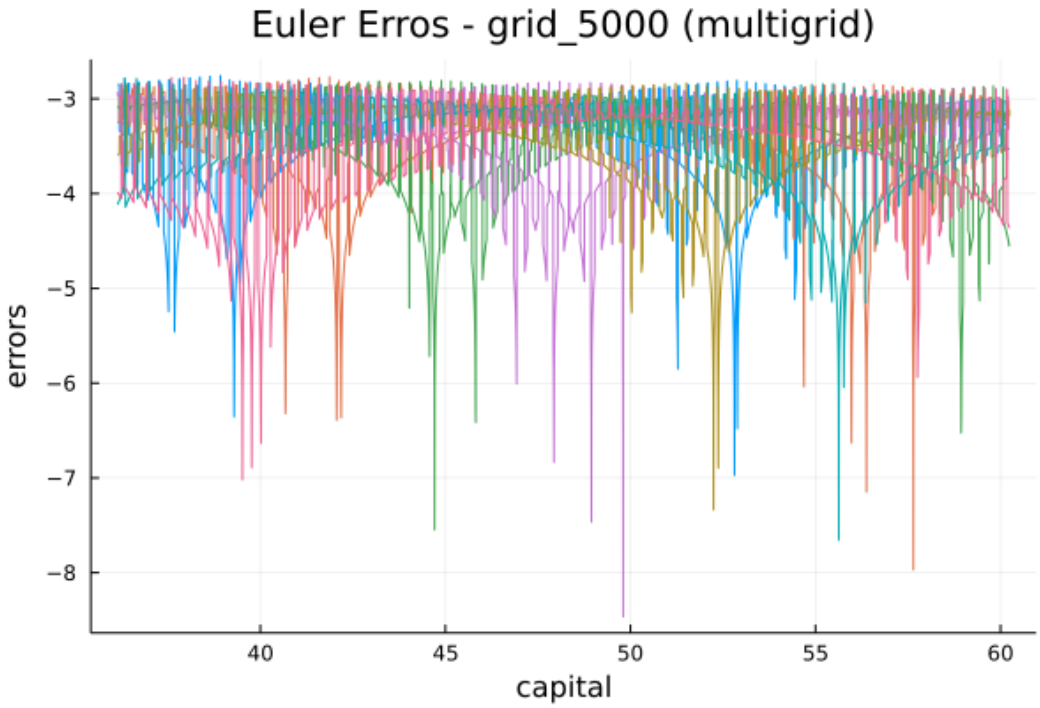
Question 5

For the multigrid function I created a function that basically do a loop and use the function I created in the last exercise, so I can easily combine the multigrid and accelerator. I did 2 functions again, but since I'm using a 3d array in the simple brute force method, for the multigrid this function is too slow. I took more than 10 minutes to converge the 5000 capital grid. I opted to let it this way because that function is not the core of this exercise, neither from the last ones, I used it just to have a comparison for the time. The function using the concavity and monotonicity took 6.5 seconds to converge the 5000 capital grid. When using just the 100 and 500 capital grids it took 3 seconds, so half the original function from the first exercise. To do the interpolation I used the function linear_interpolation() from the package Interpolation.jl. When using both the multigrid and the accelerator the function took that same 6.5 seconds to converge, so probably there is something, that I couldn't find, that can be optimized in my function. To compare I used the accelerator function from the previous exercise to calculate with the 5000 capital grid, it took 95 seconds, so the multigrid method helped a lot with the speed. I'm posting just the plots from the multigrid alone, since they are all the same. I checked the Euler Errors and for the multigrid alone and the combination with the accelerator they are exactly the same.
We can notice that now the capital grid is more precise, the consumption function is smoother, closer to what we expected.



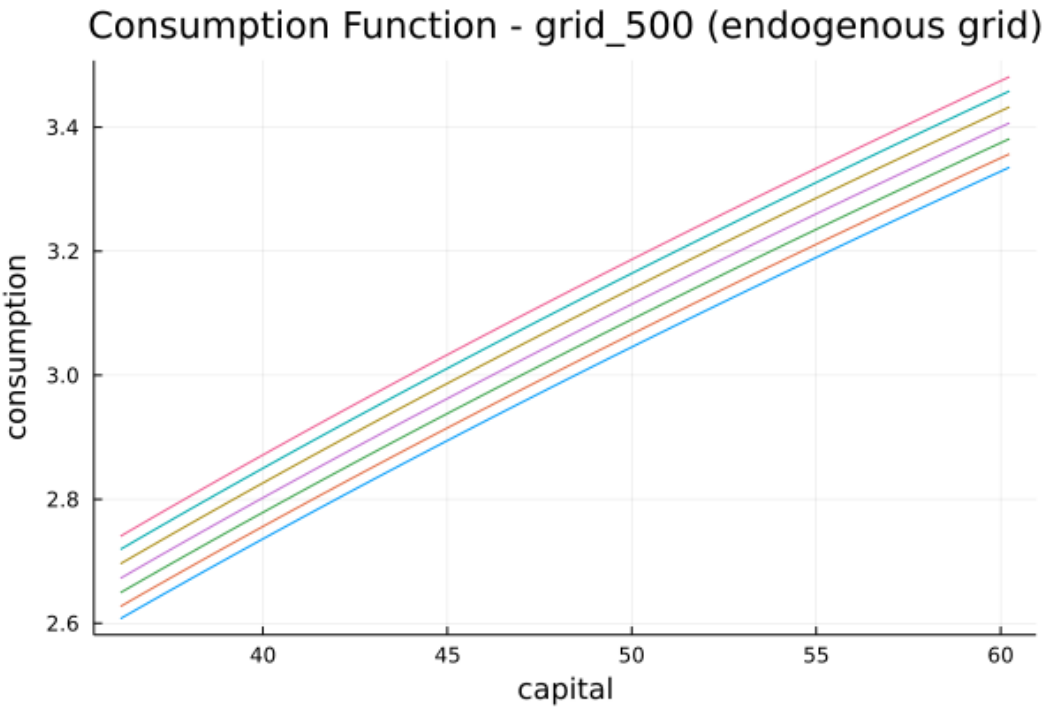

Consumption Function - grid_5000 (multigrid)
consumption
3.4
3.2
3.0
2.8
2.6
40
45
50
55
60
capital

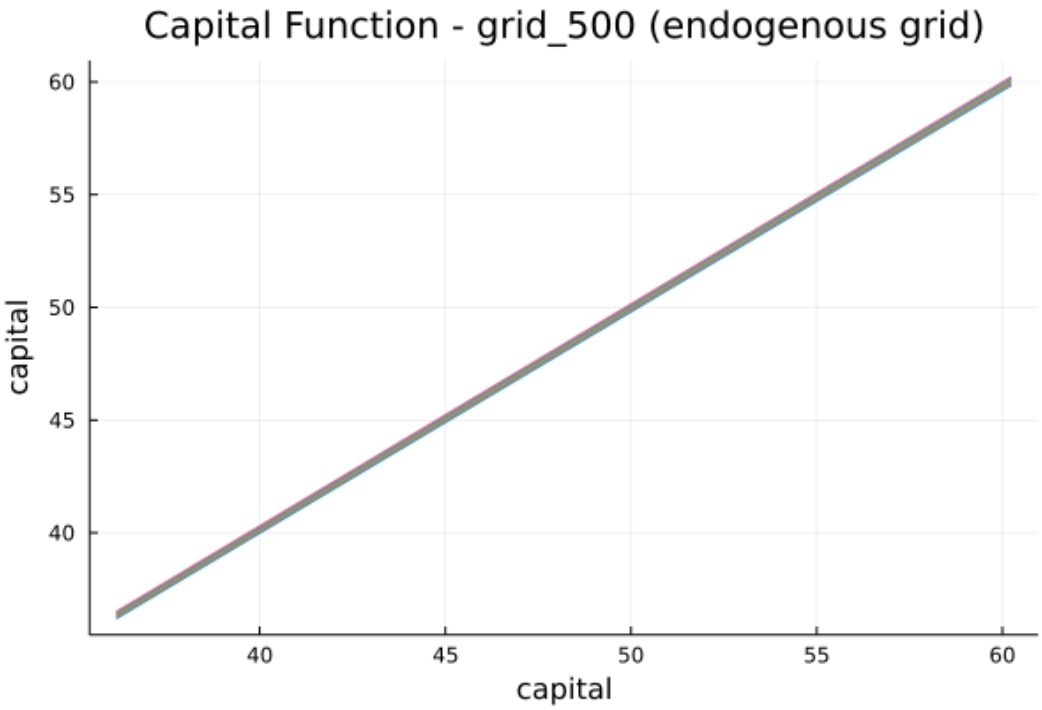

Capital Function - grid_5000 (multigrid)
capital
60
55
50
45
40
40
45
50
55
60
capital

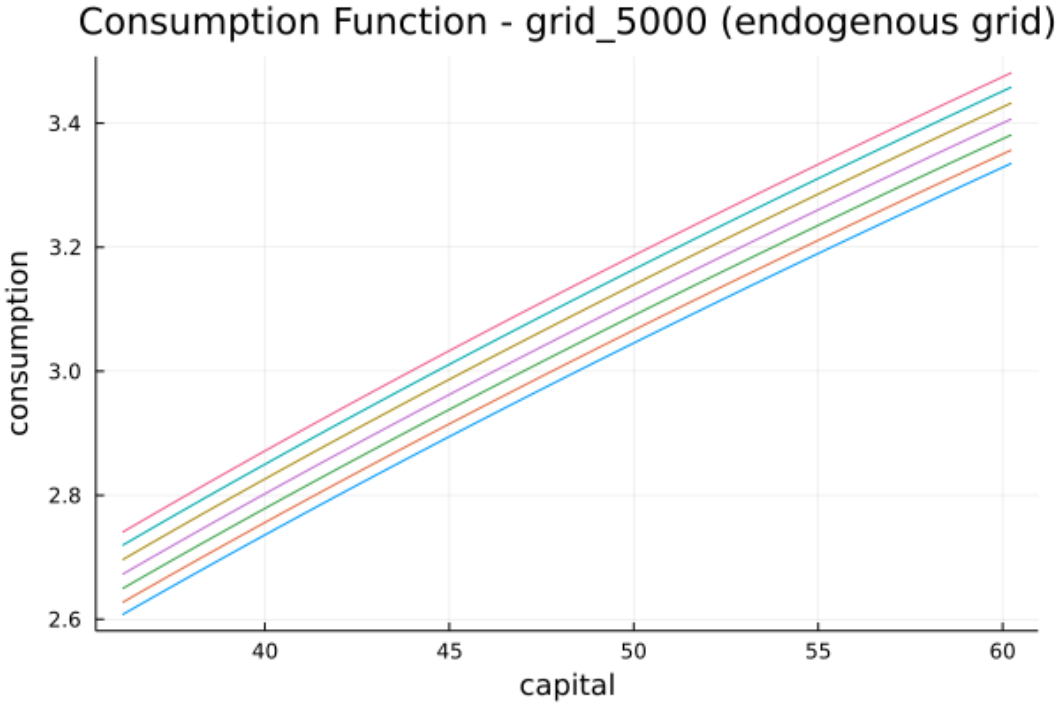

Euler Erros - grid_5000 (multigrid)
errors
−3
−4
−5
−6
−7
−8
40
45
50
55
60
capital

Question 6

My function first create a initial consumption matrix, as initial guess I used that $k = k'$. After that I constructed a matrix with the inverse of the marginal utility. After that I constructed another matrix saving the capital from today that minimizes the difference between the consumption and the last matrix, I used the capital grid as capital for tomorrow, and to minimize I used the function find_zero() from the package Roots.jl. After that I did the interpolation, using again the linear_interpolation() function. The function took 2.5 seconds to converge with the 500 capital grid and 17 seconds to run the 5000 capital grid. The function can be easily adjusted so I can use both the multigrid and the accelerator method. I could not calculate the EEE for the endogenous grid, once the political function for capital is not in our exogenous grid.




Capital Function - grid_500 (endogenous grid)
capital
60
55
50
45
40
40
45
50
55
60
capital


Consumption Function - grid_5000 (endogenous grid)
consumption
3.4
3.2
3.0
2.8
2.6
40
45
50
55
60
capital

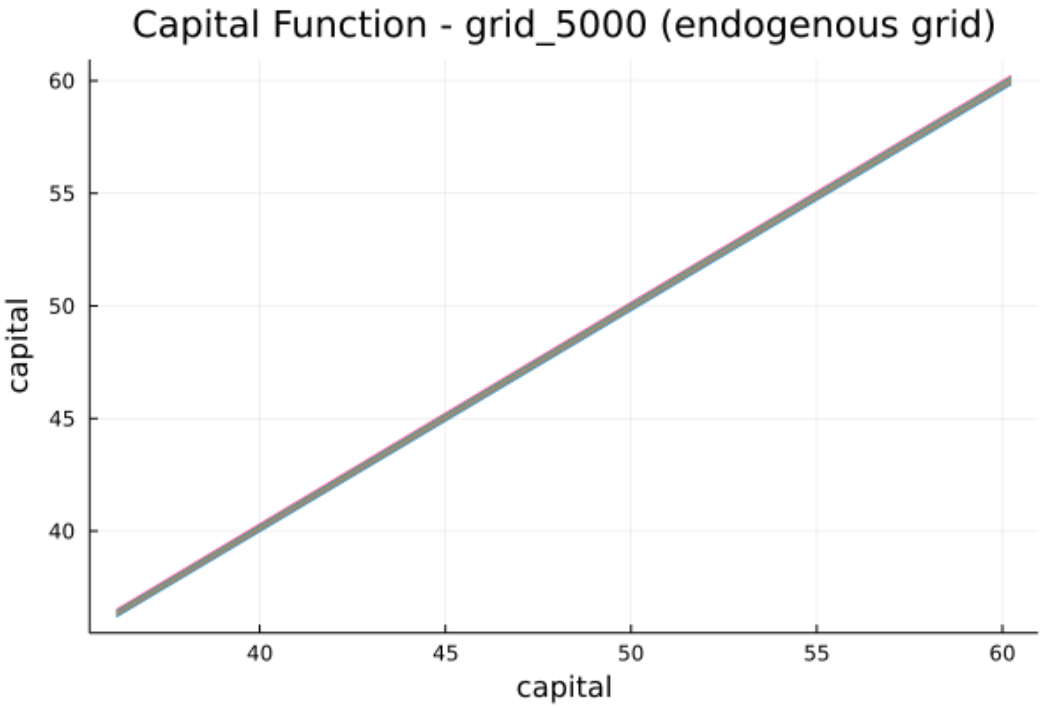

Capital Function - grid_5000 (endogenous grid)
capital
60
55
50
45
40
40
45
50
55
60
capital

Table comparing for the 500 capital grid:

Method	Time	mean EEE
Brute force	34	-3
Concavity and Mon.	6	-3
BF accelerator	28	-3
CM accelerator	5	-3
BF multigrid	10	-3
CM multigrid	3	-3
BF mult. and acc.	8	-3
CM mult. and acc.	1.5	-3
Endogenous grid	3.5	X

Table comparing for the 5000 capital grid:

Method	Time	mean EEE
Concavity and Mon.	62	-3.5
CM accelerator	94	-3.5
CM multigrid	7	-3.5
CM mult. and acc.	6.5	-3.5
Endogenous grid	17	X