Quantitative Macroeconomics.
Homework 3.

Sebastian Zalas

October 30, 2020

Each method is coded in separate python file, with corresponding name to subject of the subsection. Supplied codes should be enough to replicate presented results.

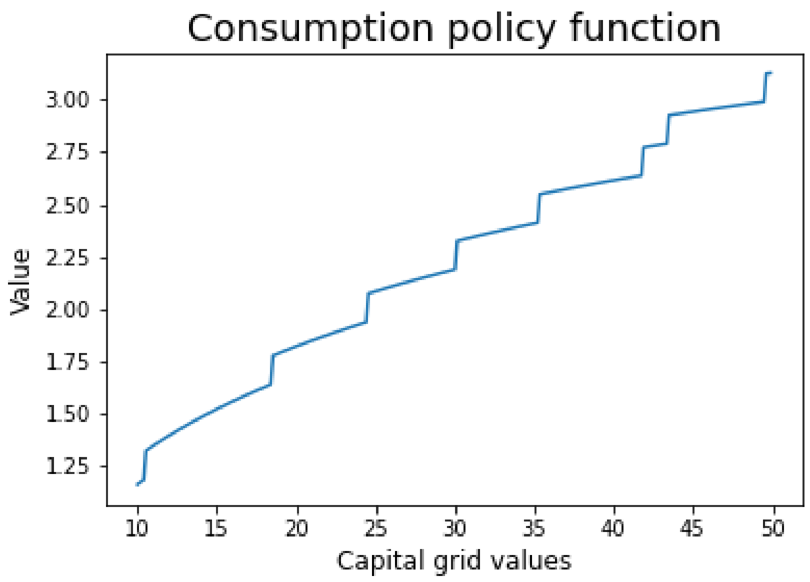
Value Function Iteration with inelastic labor supply.

In each subsection I post value function and policy function plot. At the end of this section I present table with computation time for each method.

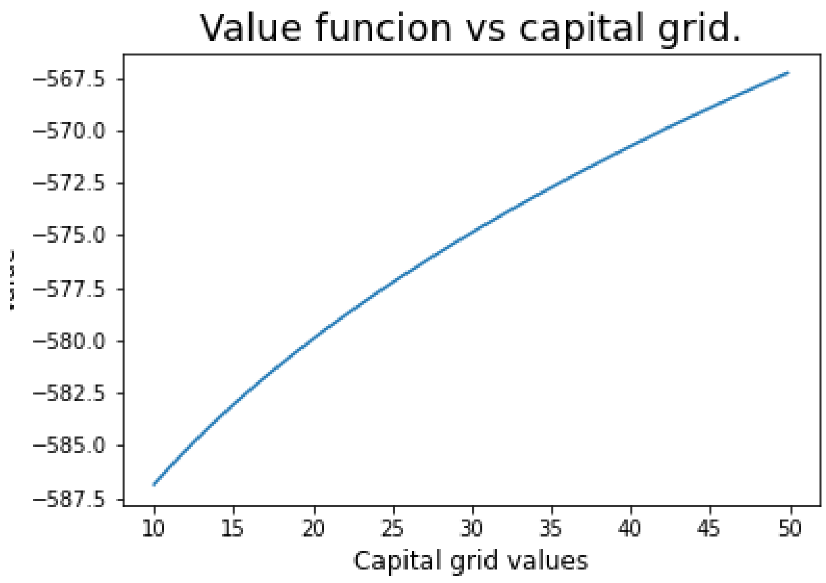
a). Brute force VF

For this task I prepared two codes. First one computes matrix χ with a loop, second one utilizes vectorized operations. I present results for both.

Figure 1: Brute force VFI.

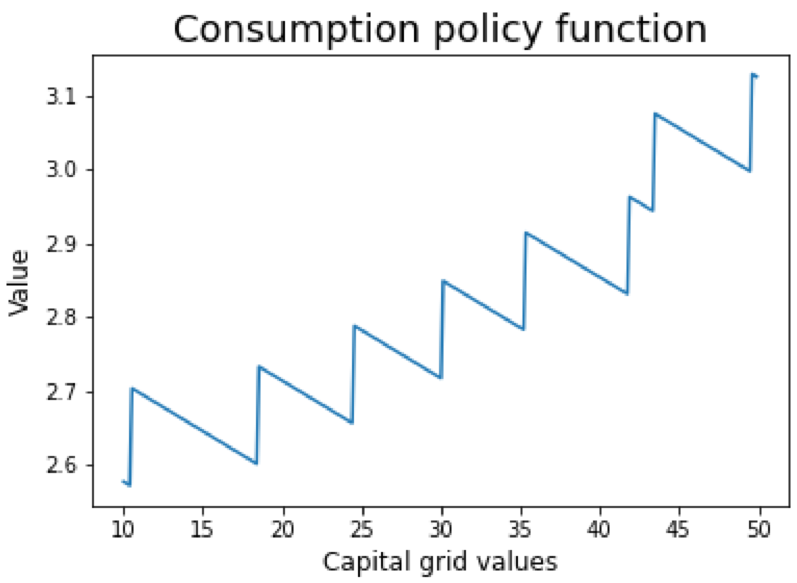


(a) Consumption policy function.

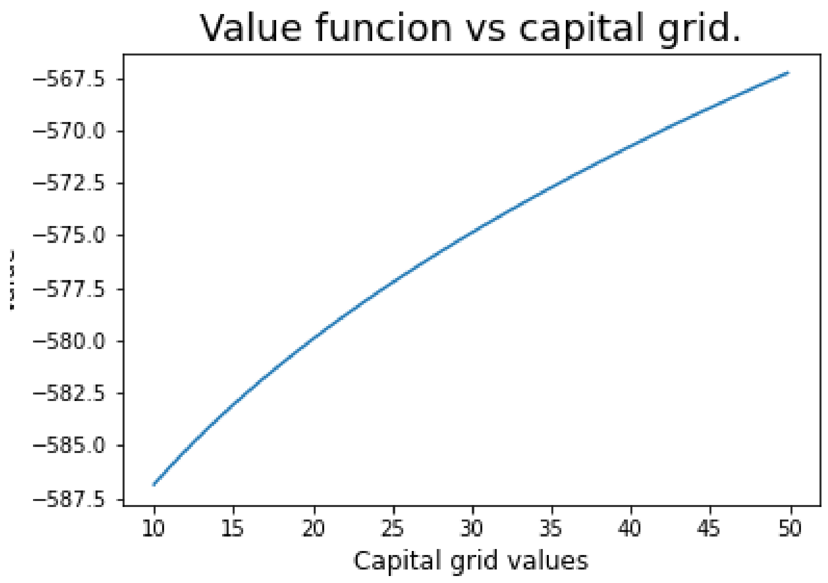


(b) Value function.

Figure 2: VFI with vectorized operations.



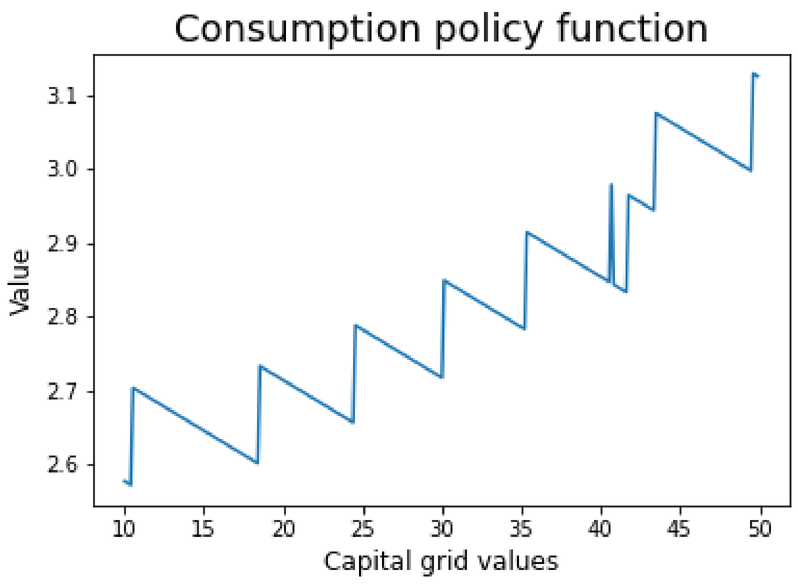
(a) Consumption policy function.



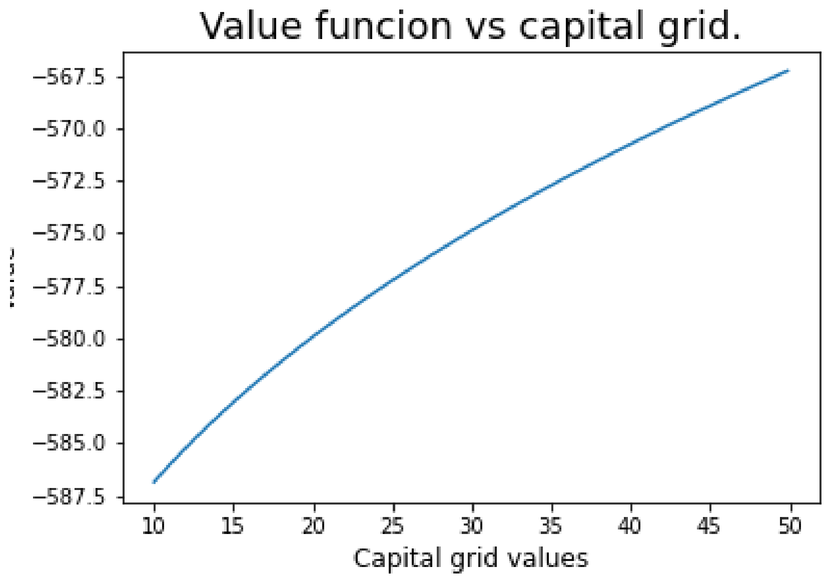
(b) Value function.

b). Value function taking into account monotonicity of the optimal decision rule.

Figure 3: VFI + monotonicity.



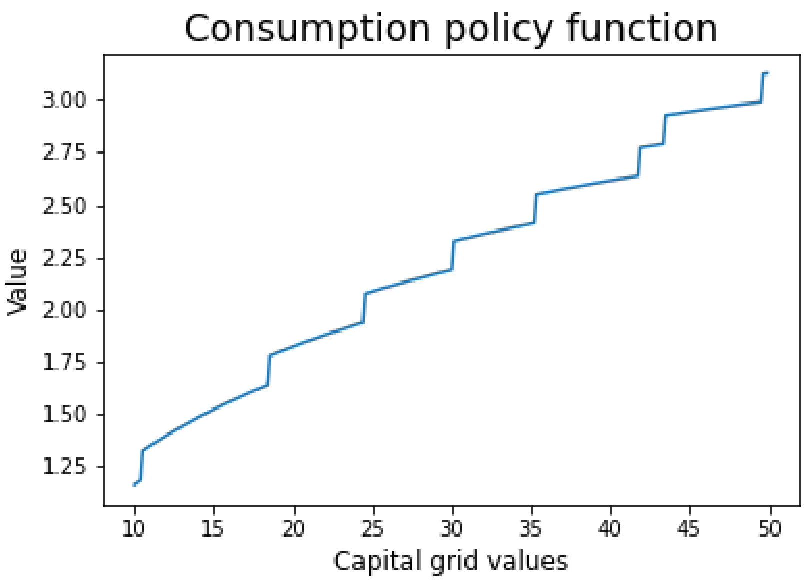
(a) Consumption policy function.



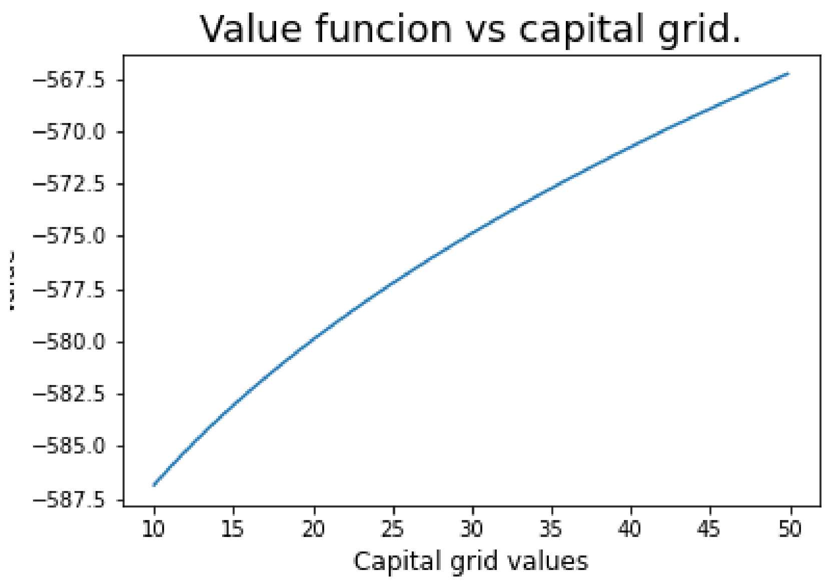
(b) Value function.

c). Value function taking into account concavity of the value function.

Figure 4: VFI + concavity.



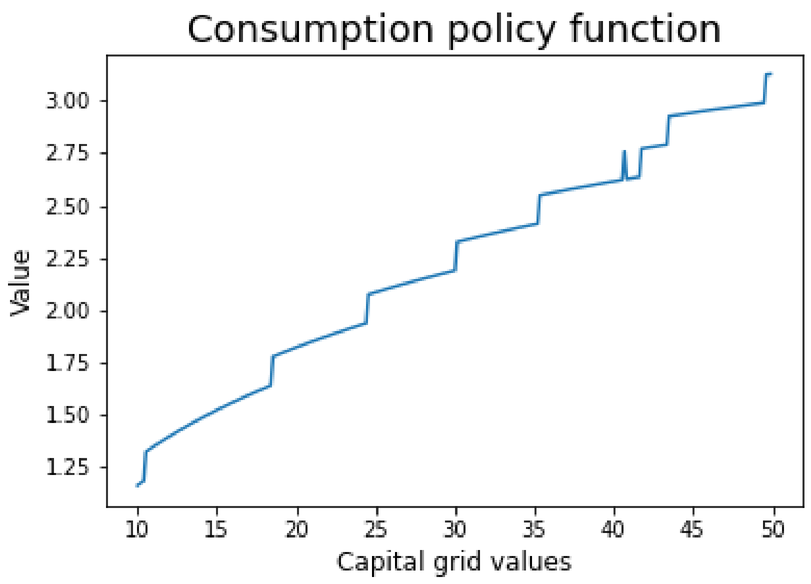
(a) Consumption policy function.



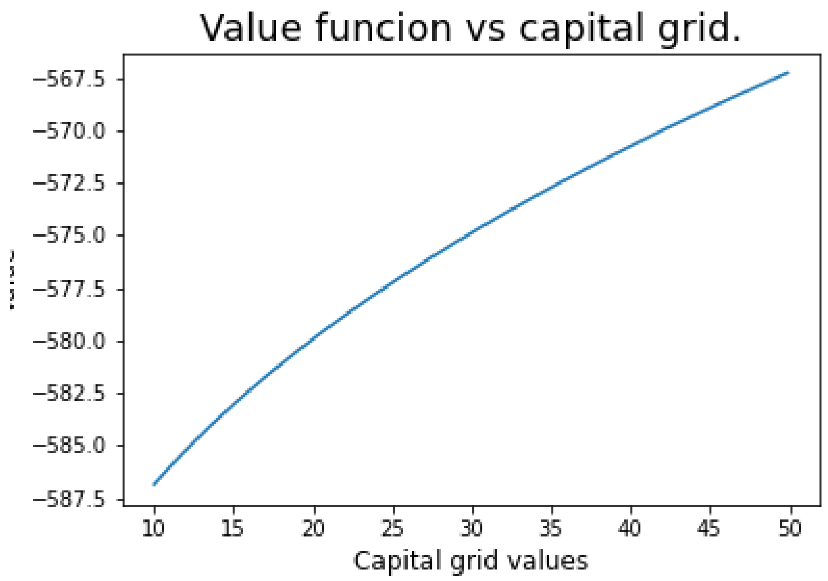
(b) Value function.

e). Value function taking into account both concavity of the value function and monotonicity of the decision rule.

Figure 5: VFI + concavity and monotonicity.



(a) Consumption policy function.

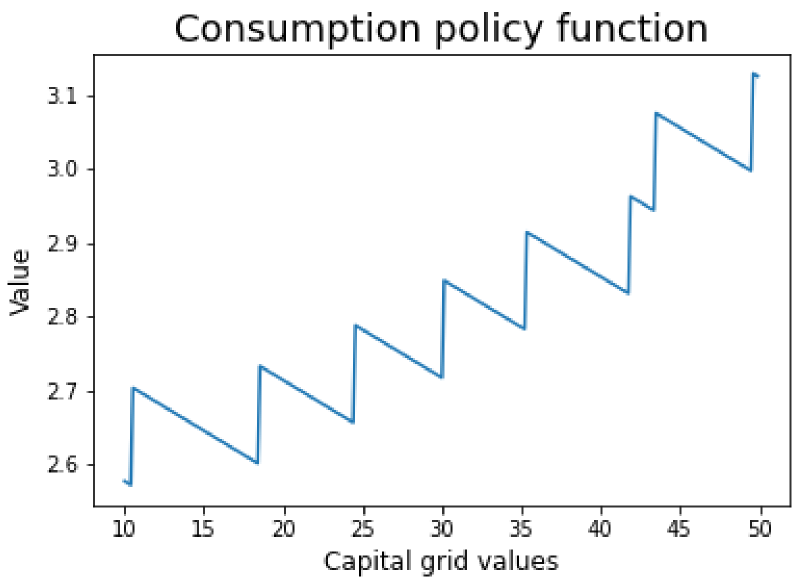


(b) Value function.

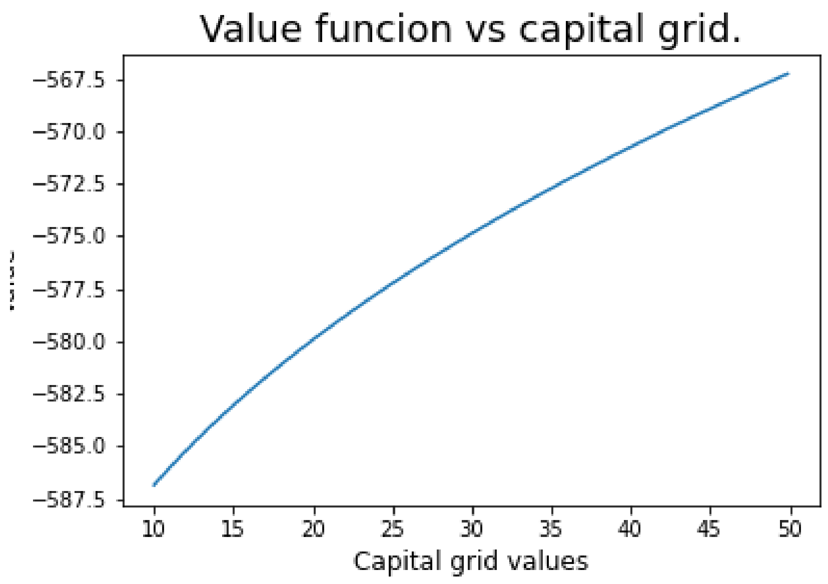
f). Howard Improvement

For the sake of speed, I perform Howard improvement with vectorized value function code. I consider different numbers of steps in Howard improvement, but value function and consumption policy behaves in the same way, therefore I present plot for only one variant.

Figure 6: Howard improvement VFI.



(a) Consumption policy function.



(b) Value function.

Comparison of computation time.

Table 1: Time of convergence for VFI with inelastic labor supply.

Method	Time (seconds)
VFI - brute force loop	267.0094735622406
VFI (Vectorized)	1.0744149684906006
VFI + monotonicity	651.280086517334
VFI + concavity	459.5858016014099
VFI + concavity + monotonicity	1057.7483694553375
VFI + 5 Howard iterations	0.4263460636138916
VFI + 10 Howard iterations	0.37359118461608887
VFI + 20 Howard iterations	0.3264732360839844
VFI + 50 Howard iterations	0.31513166427612305

Grid size was set to 300 and convergence criterion to 1.e-6.

The result show that the best way to improve speed of convergence is to omit loops in codes. Additionally, Howard improvement really makes code faster. Methods that explore concavity and monotonicity of value function are much slower because operations have to be implemented with loops. However it is weird that improving schemes really did not make convergence faster. It may be issue with the code, or computer, as it depends on the moment of execution when cpu is less or more occupied. Moreover, I did not managed to code local search variant.

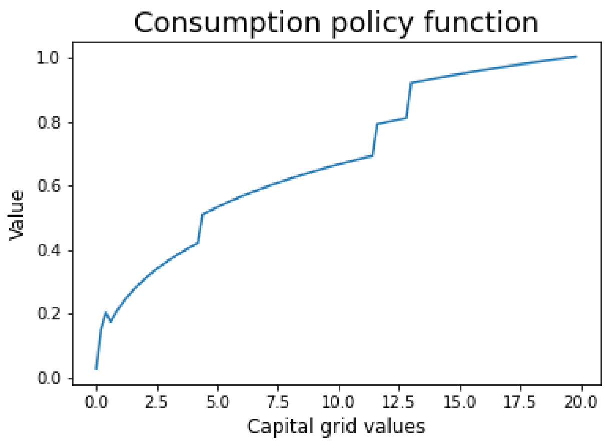
Value Function Iteration with elastic labour supply.

In each subsection I post value function and policy function plot, in this section also for labour. At the end of this section I present table with computation time for each method.

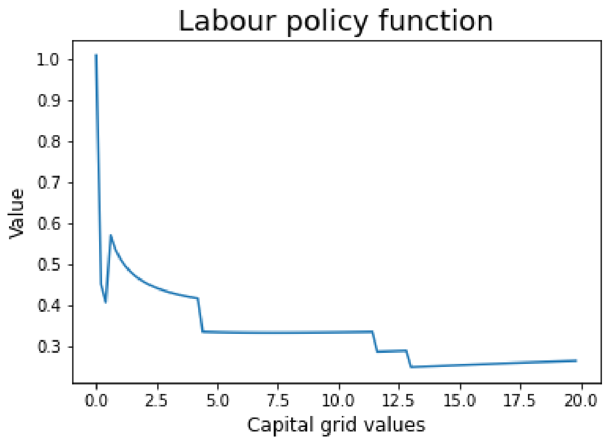
a). Brute force VF with elastic labour.

As in first part, for this task I prepared two codes. First one computes matrix χ with a loop, second one utilizes vectorized operations. I present results for both.

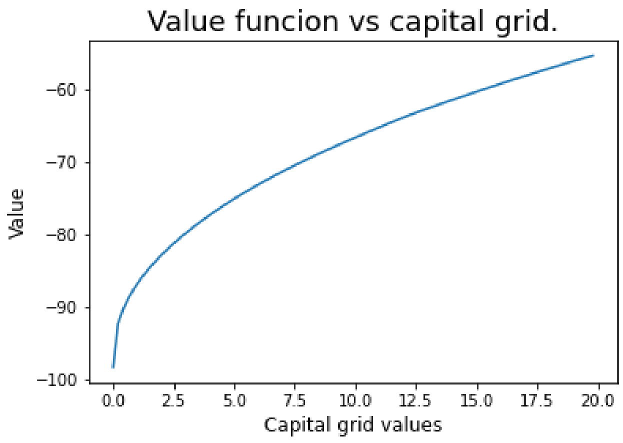
Figure 7: Brute force VF with labour choice.



(a) Consumption policy function.

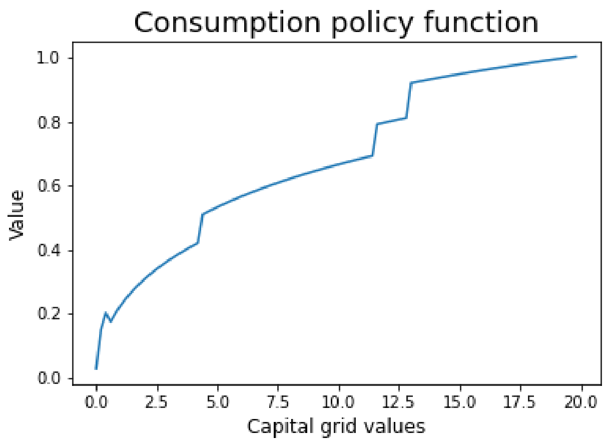


(b) Labour policy function.

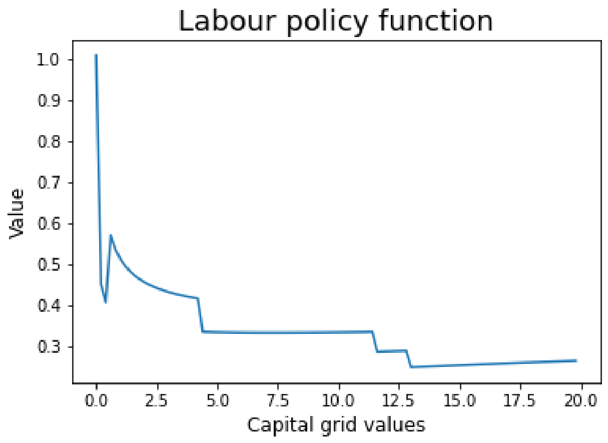


(c) Value function.

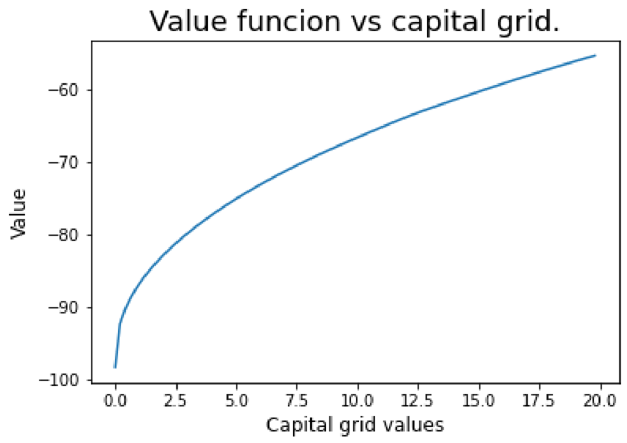
Figure 8: Vectorized VFI with labour choice.



(a) Consumption policy function.



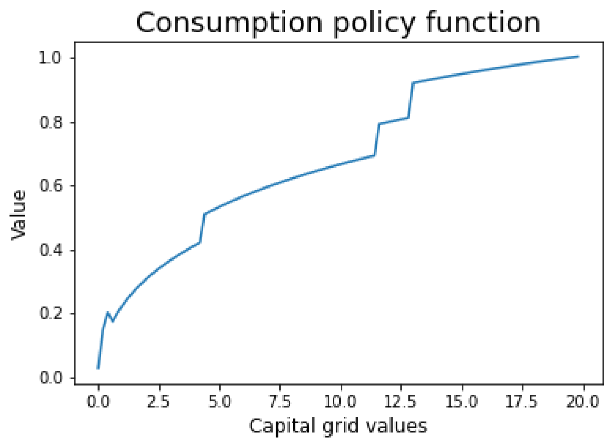
(b) Labour policy function.



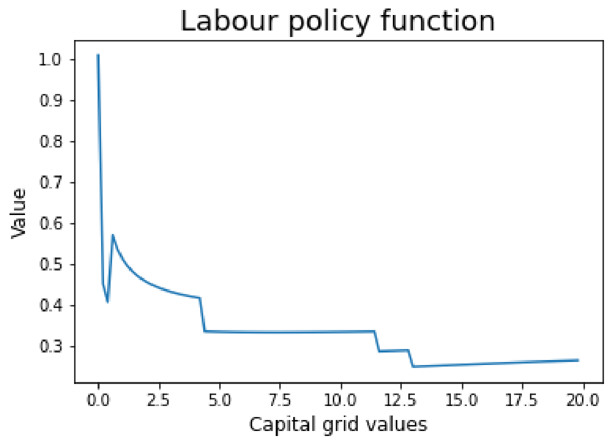
(c) Value function.

b). Value function taking into account monotonicity of the optimal decision rule.

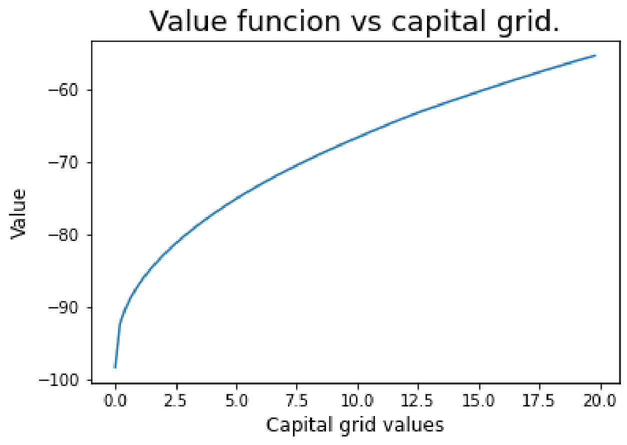
Figure 9: VFI + monotonicity property + labour choice.



(a) Consumption policy function.



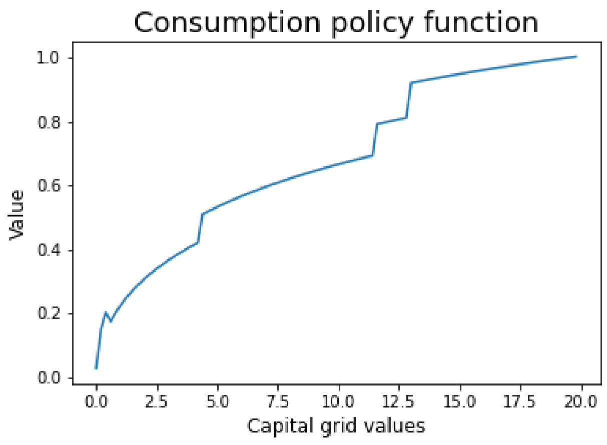
(b) Labour policy function.



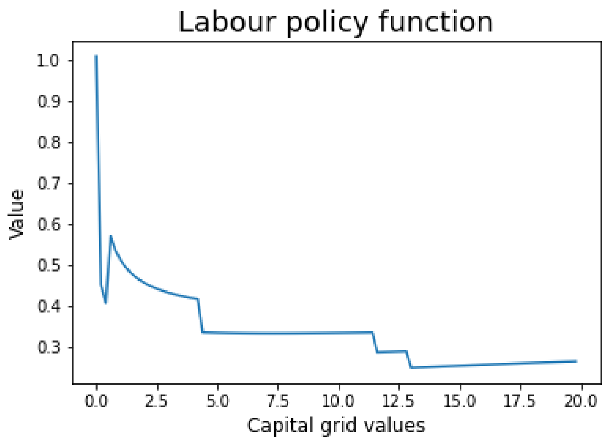
(c) Value function.

c). Value function taking into account concavity of the optimal decision rule.

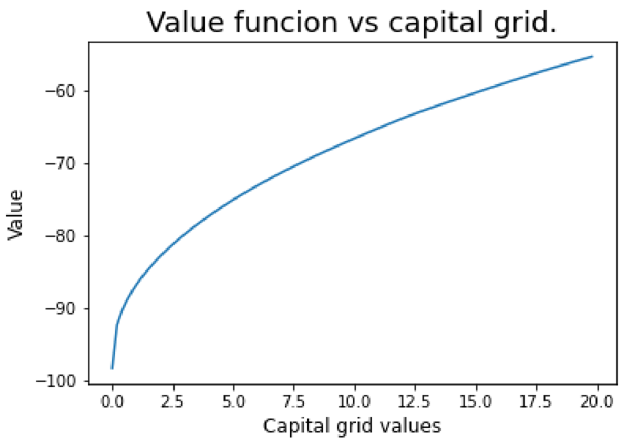
Figure 10: VFI + concavity property + labour choice.



(a) Consumption policy function.



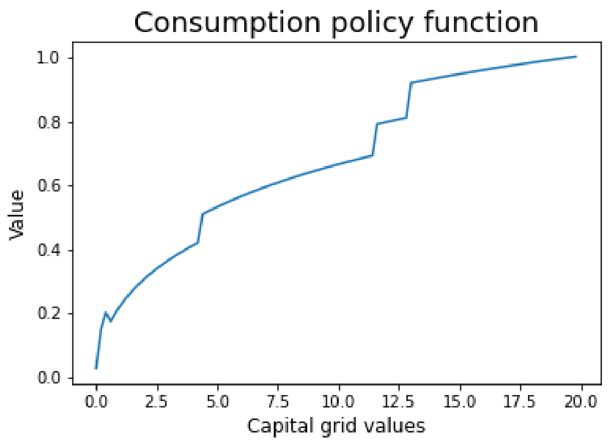
(b) Labour policy function.



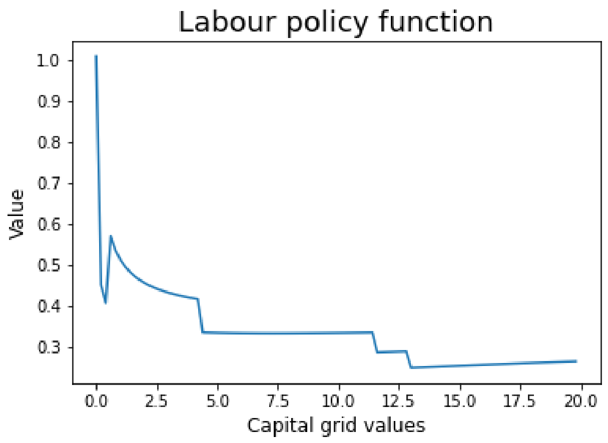
(c) Value function.

e). Value function taking into account both concavity of the value function and monotonicity of the decision rule.

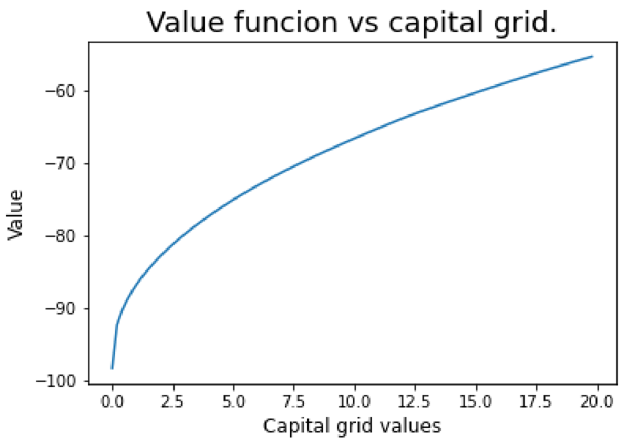
Figure 11: VFI + concavity property + labour choice.



(a) Consumption policy function.



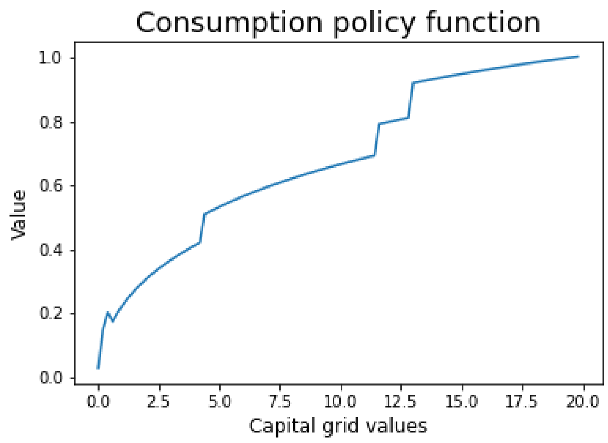
(b) Labour policy function.



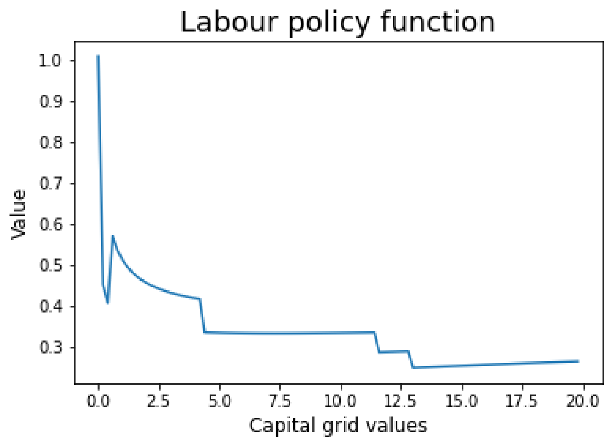
(c) Value function.

f). Howard Improvement

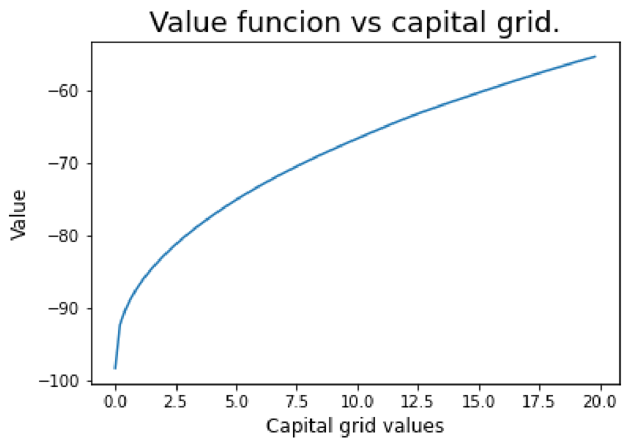
Figure 12: VFI + Howard improvement and labour choice.



(a) Consumption policy function.



(b) Labour policy function.



(c) Value function.

For the sake of speed, I perform Howard improvement with vectorized value function code. I consider different numbers of steps in Howard improvement, but value function and consumption policy behaves in the same way, therefore I present plot for only one variant.

Comparison of computation time.

Table 2: Time of convervence for VF with elastic labor supply.

Method	Time (seconds)
VFI - brute force loop	28.459978580474854
VFI (Vectorized)	1.7818748950958252
VFI + monotonicity	29.87951683998108
VFI + concavity	49.94881510734558
VFI + concavity + monotonicity	49.60693287849426
VFI + 5 Howard iterations	1.9451911449432373
VFI + 10 Howard iterations	1.9105210304260254
VFI + 20 Howard iterations	1.8011295795440674
VFI + 50 Howard iterations	1.827261209487915

Grid size was set to 100 and convergence criterion to 1.e-6.

Here I diminished number of gridpoints in comparison to exercises with inelastic labour. Results here are similar, vectorized code is the fastest. Also different improvements did not shorten convergence time. Now even Howard improvement does not make convergence faster. It may be caused by relatively low number of gridpoints - whit more gridpoints maybe it would be the case that Howard algorithm would make it faster.

Value Function Iteration - Chebyshev approximation.

I did not managed to fix all the mistakes from the code, so I do not report any output as it makes no sense (for instance, negative consumption) but I post on repository my tentative code.