ECON-714: Problem Set II

Problem 1: Steady State

The Bellman equation of the problem is

$$V(k,z) = \max_{c,l}[(\log c - l^2/2) + \beta \mathbb{E} V(k',z')]$$

such that $k' = e^z k^\alpha l^{1-\alpha} - c + (1-\delta)k$

Solving the labor and consumption FOCs, we observe

$$l = (e^z k^\alpha / c)^{1/(1+\alpha)}$$

and as such the problem can be restated as

$$V(k,z) = \max_{k'}[(\log c - l^2/2) + \beta \mathbb{E} V(k',z')]$$

such that $l = (e^z k^\alpha / c)^{1/(1+\alpha)}$

From the Euler equation, by equating $c = c'$ and setting $z = 0$, we observe the capital-labor ratio at the steady state is

$$k^*/l^* = (\alpha/(1/\beta + \delta - 1))$$

And by combining the accounting identity equation with the consumption-labor FOC, we observe the steady state labor can be expressed in terms of model primitives and the steady-state capital-labor ratio:

$$l^* = [((k^*/l^*)^\alpha - \delta k^*/l^*)/((1-\alpha)(k^*/l^*)^\alpha)]^{1/2}$$

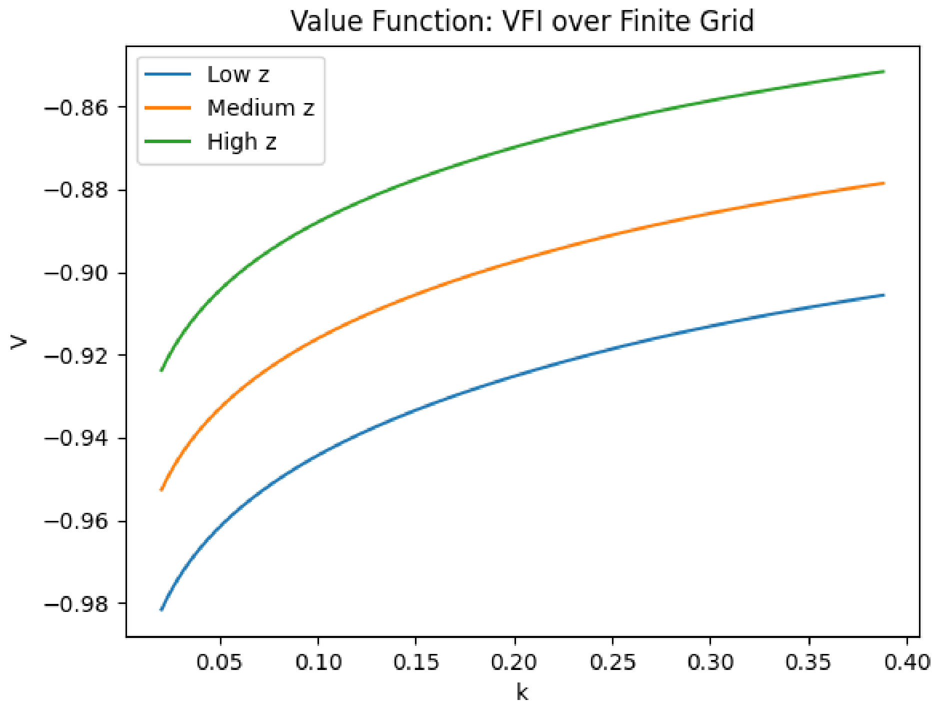
with which we can recover k^*, y^*, c^* by substitution. (See main.py for details.)

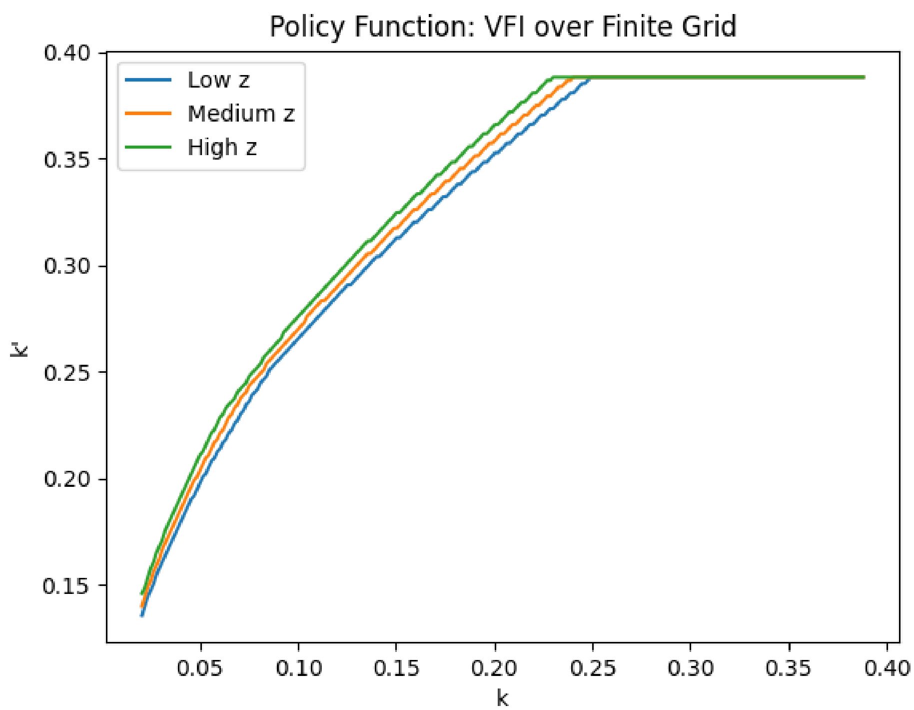
Problem 2: VFI with a fixed grid

I implement a VFI with a fixed grid. I define the policy function to be $k'(k,z)$. Under the policy function and the consumption-labor equation (the constraint to the dynamic programming), the consumption is defined implicitly as

$$-k'(k,z) + e^z k^\alpha l(c;k,z)^{1-\alpha} - c + (1-\delta)k = 0$$

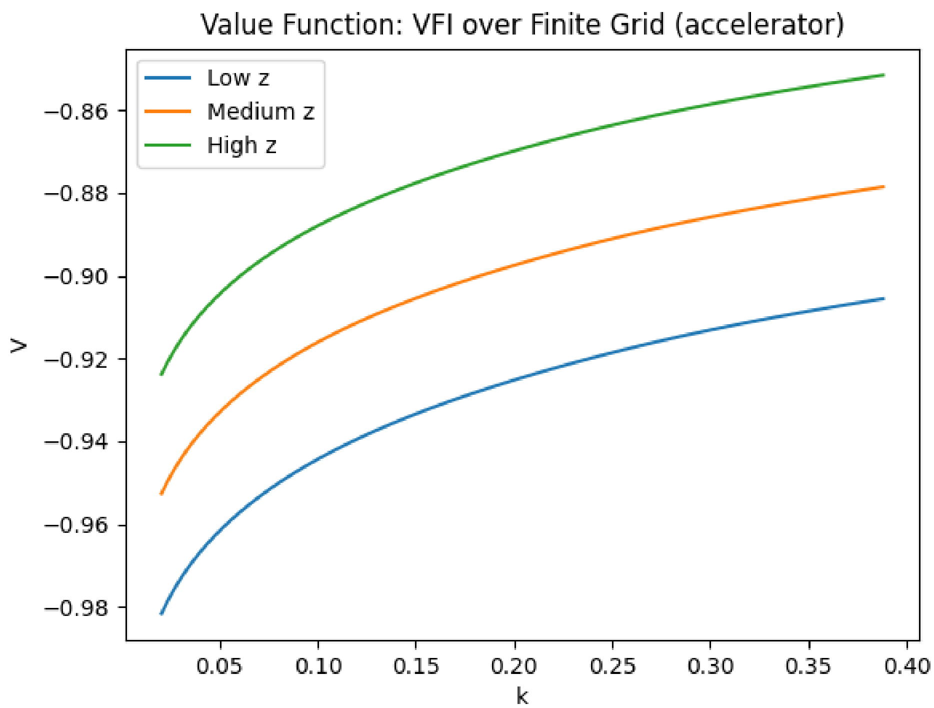
See main.py for implementation details. Below is the value function and the policy function computed from VFI.

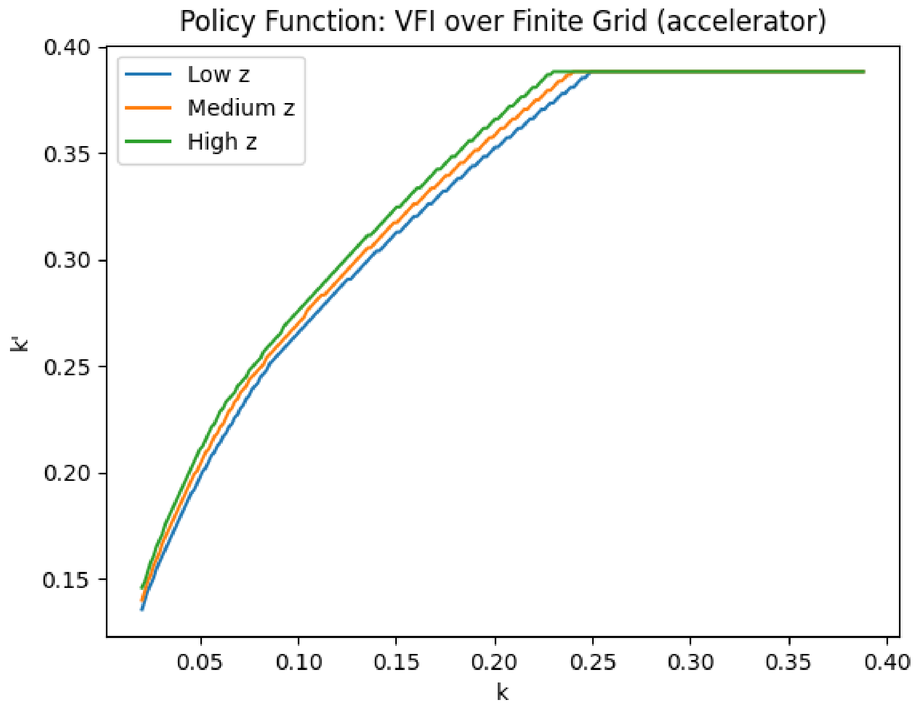

Value Function: VFI over Finite Grid
Low z
Medium z
High z
V
−0.86
−0.88
−0.90
−0.92
−0.94
−0.96
−0.98
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
k


Policy Function: VFI over Finite Grid
Low z
Medium z
High z
k'
0.40
0.35
0.30
0.25
0.20
0.15
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
k

Problem 3: Accelerator

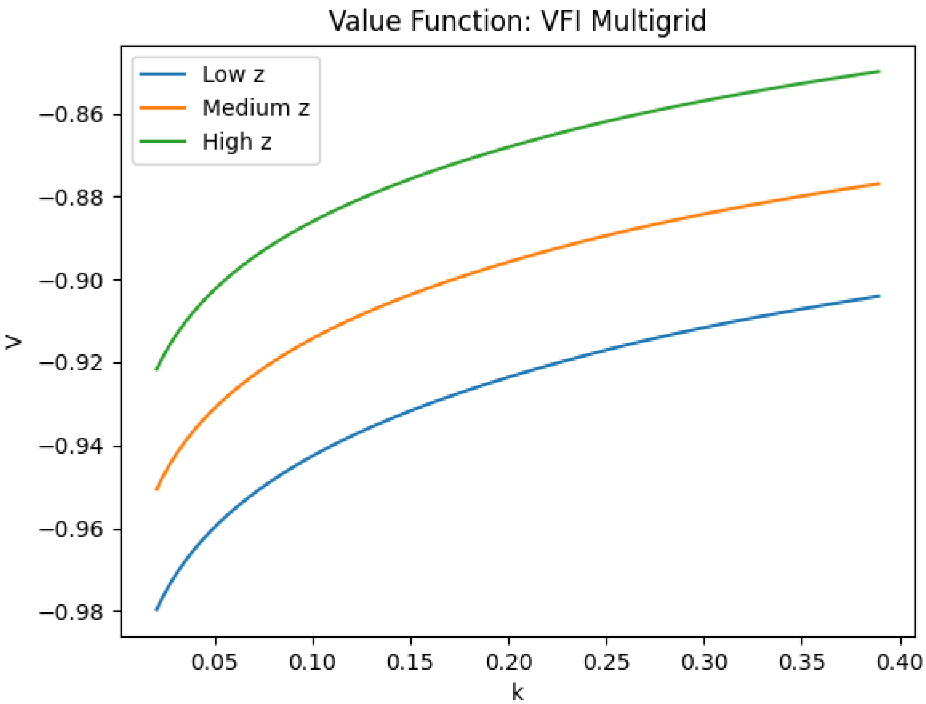
I implement the accelerator and report the results below. Using an accelerator (skipping the max operator 9 out of 10 times) improves the compute time: for 250 grid points for capital at 1e-5 tolerance, the compute time is reduced from 46 seconds to 21 seconds. We observe no perceptible difference between the accelerated results and the non-accelerated ones.

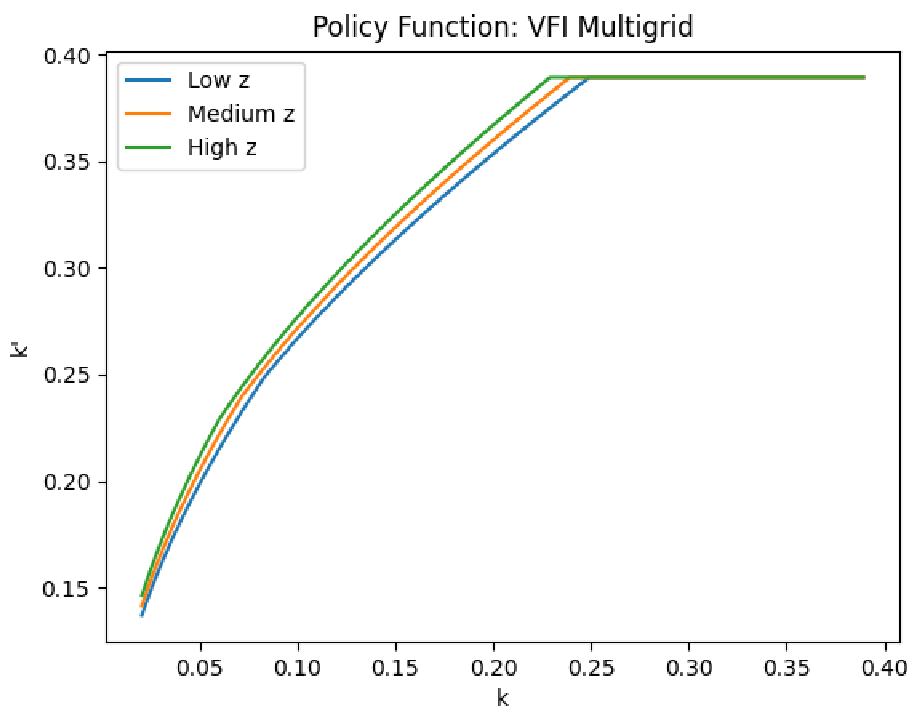




Problem 4: Multigrid

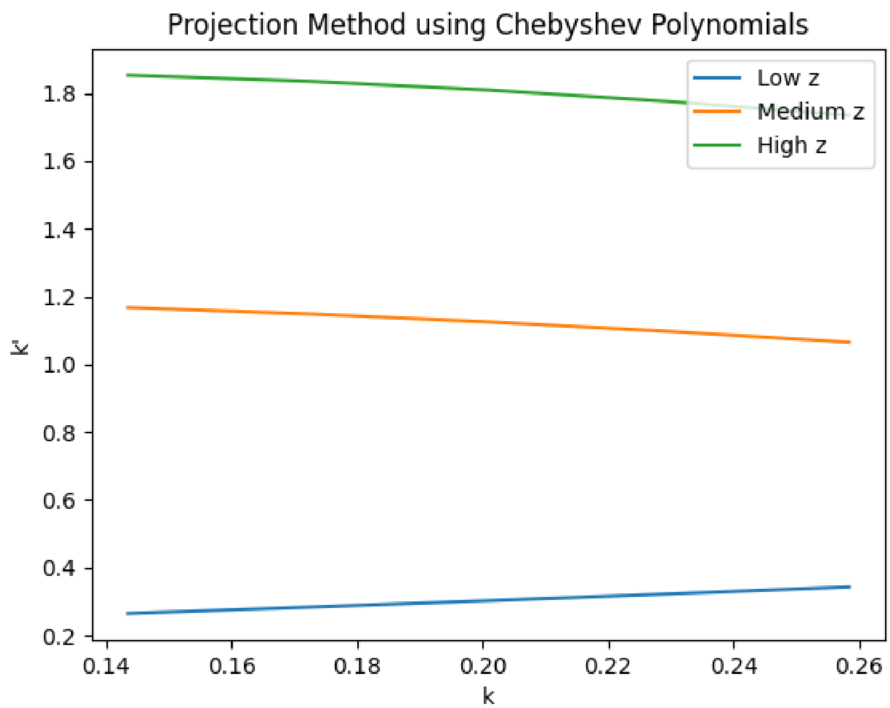
I implement the multigrid algorithm and show the results below. The compute times for the capital grid of 1e3 points and the tolerance of 1e-3 are 79 seconds (VFI without acceleration); 42 seconds (VFI with accleration) and 37 seconds for (VFI with multigrid (1e2 and 1e3 points in order). Observe with more points, the policy functions become smoother.


Value Function: VFI Multigrid
Low z
Medium z
High z
V
k
−0.86
−0.88
−0.90
−0.92
−0.94
−0.96
−0.98
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40


Policy Function: VFI Multigrid
Low z
Medium z
High z
k'
k
0.40
0.35
0.30
0.25
0.20
0.15
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40

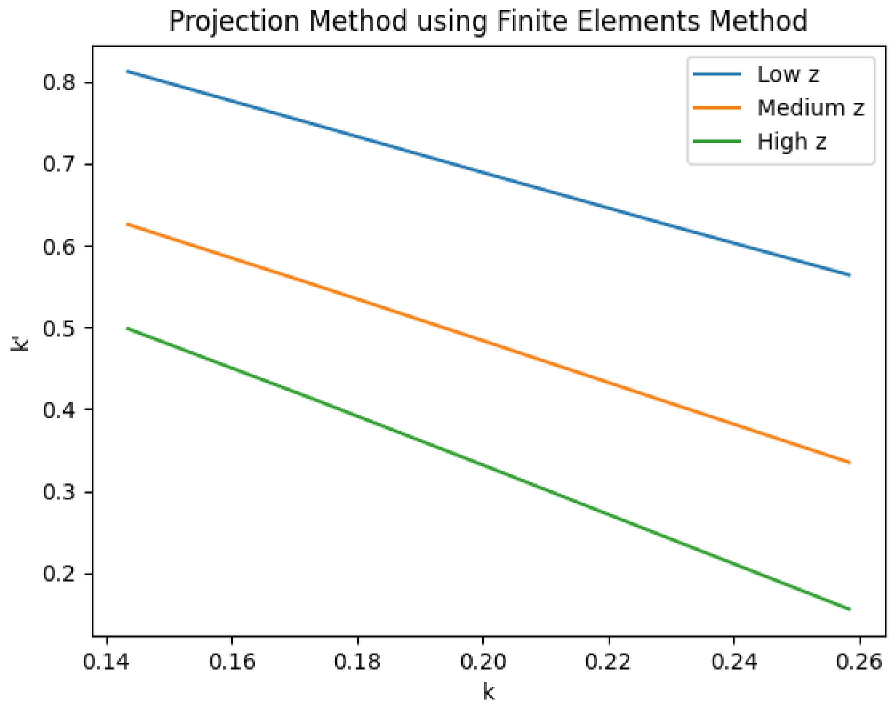
Problem 5: Chebychev Polynomials

I implement the projection method using Chebychev polynomials. I use the collocation method to integrate the Euler equation errors on the grid on capital and productivity. I numerically minimize the residuals with respect to the Chebychev polynomials. I observe depending on initialization the numerical optimizer may fail to optimize. I observe also the Euler equation errors are inadequately optimized given the numerical optimizer. I show below a result from a halted optimization. I observe the general trend of the policy function is also visible from the projection method. Yet I observe also that the results deviate significantly from previous iterations. See main.py form implementation details.



Problem 6: Finite Elements

I implement the projection method using finite elements. As in the previous problem, I use the collocation method to integrate the Euler equation errors. I observe numerical optimization is as before insufficient, yet the results are better optimized using the finite elements method. The results contrast to previous results. Potentially a bug in the implementation of parameterization of the policy function may be culprit.


Projection Method using Finite Elements Method
Low z
Medium z
High z
k'
k
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.14
0.16
0.18
0.20
0.22
0.24
0.26