Problem Set 3

Joe Emmens

October 29, 2020

Recursive Formulation

At first the labour supply is normalised to 1 and supplied inelastically.

$$E_0 \left\{ \sum_{t=0}^{\infty} \beta^t u\left(c_t\right) \right\}$$

$$u\left(c_t\right) = \ln c_t, \text{subject to}$$

$$c_t + i_t = y_t$$
$$y_t = k_t^{1-\theta}$$
$$i_t = k_{t+1} - (1-\delta)k_t$$

From the constraints,

$$c_t = k_t^{1-\theta} + (1-\delta)k_t - k_{t+1}$$

In recursive formulation,

$$V(k) = \max_{k' \in \Gamma(k)} ln(k^{1-\theta} + (1-\delta)k - k') + \beta v(k')$$
$$\Gamma(k) = \left\{k' \in \mathbb{R}_+ : k' \leq k^{1-\theta} + (1-\delta)k\right\}$$

Attaching the Lagrange multiplier on the constraint we have that,

$$\frac{\partial V(k)}{\partial k'} = -\frac{1}{k^{1-\theta} + (1-\delta)k - k'} - \lambda_k + \beta \frac{\partial V(k')}{\partial k'}$$

With both the following conditions,

- Complementary slackness $\lambda_k(k' \leq k^{1-\theta} + (1-\delta)k)$
- Dual feasibility $\lambda_k > 0$

By applying the envelope theorem, we know that the derivative of the value function with respect to a state variable is the derivative of the utility with respect to that state variable.

$$\frac{\partial V(k')}{\partial k'} = u_k'(k')$$

We can assume a standard that the constraint is not binding, else consumption will go to zero which the agent will not choose thanks to the Inada conditions. Therefore the Euler equation is,

$$\frac{1}{k^{1-\theta} + (1-\delta)k - k'} = \beta \frac{(1-\theta)k^{\theta} + (1-\delta)}{k^{1-\theta} + (1-\delta)k - k'}$$

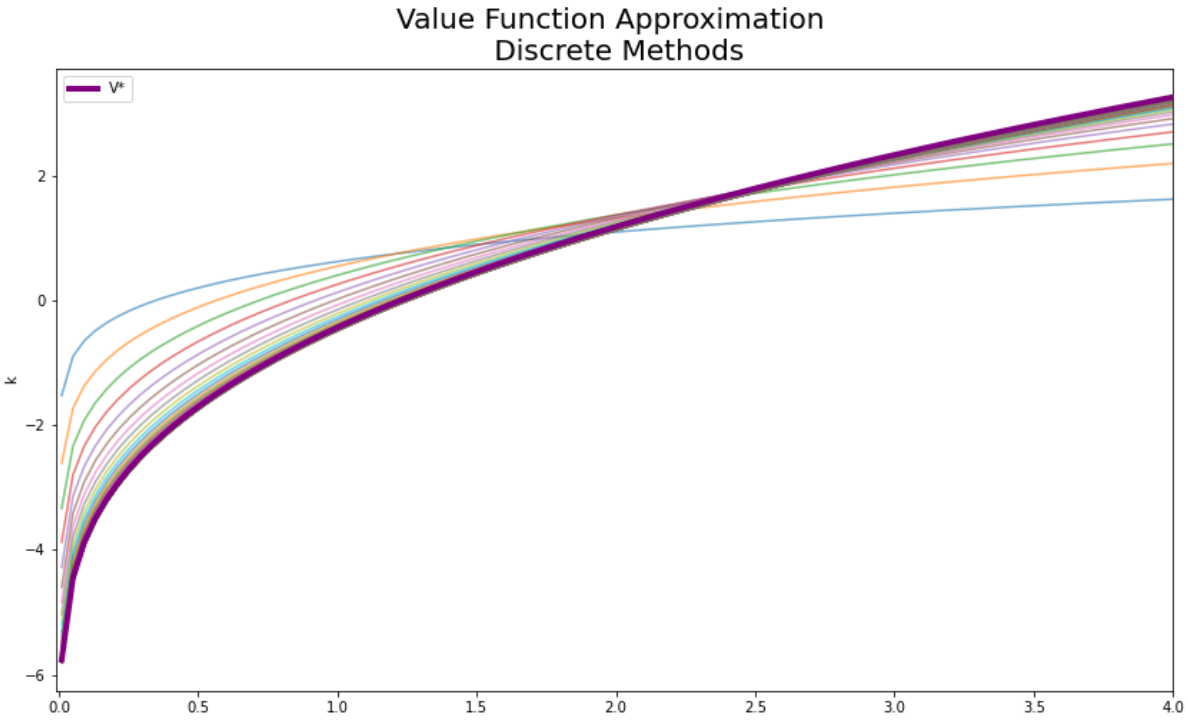
As before, solving for capital in the steady state we find that,

$$k^{ss} = \left(\frac{\beta(1-\theta)}{1-\beta(1-\delta)}\right)^{\frac{1}{\theta}}$$

Given the following exogenous parameters, $\begin{bmatrix}\theta & \beta & \delta & h_t\end{bmatrix} = \begin{bmatrix}0.679 & 0.988 & 0.13 & 1\end{bmatrix}$,

$$k^{ss} \approx 3.231$$

To solve the first problem I have set the maximum capital level to 4, slightly above the analytically found steady state. My initial guess in all stages is the null vector, $V^{s=0}(k_i) = 0, \forall i \in \{1, .., p\}$.



As to be expected the value function is increasing in capital. The curvature in the value function is greater at lower levels of capital which is consistent with a theory of diminishing marginal utility.

Speeding Up

I have run numerous versions of the algorithm which in theory utilise properties of the value and policy functions. For all the methods presented I have kept the following specifications constant,

- $\epsilon = 0.0001$
- grid size = 100
- max iterations = 275

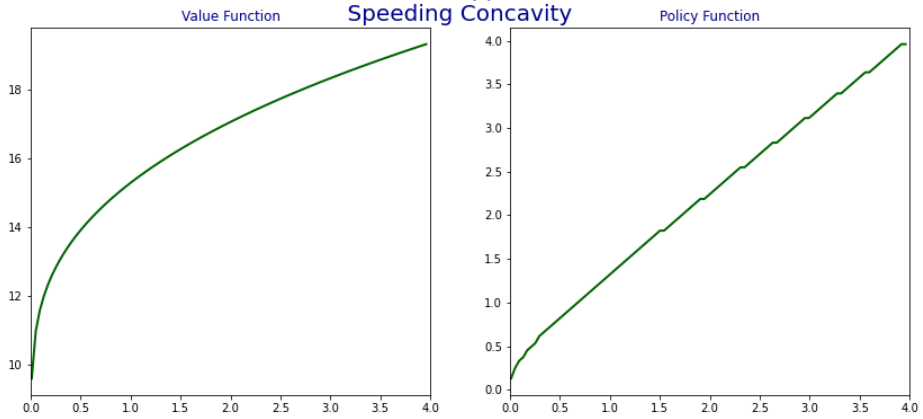
The results are presented in the table below and displayed to the nearest hundredth of a second.

Method	Time
Brute Force	1.32s
Concavity	3.66s
Monotonicity	3.61s
Concavity and Monotonicity	2.42s
Local Search	3.25s

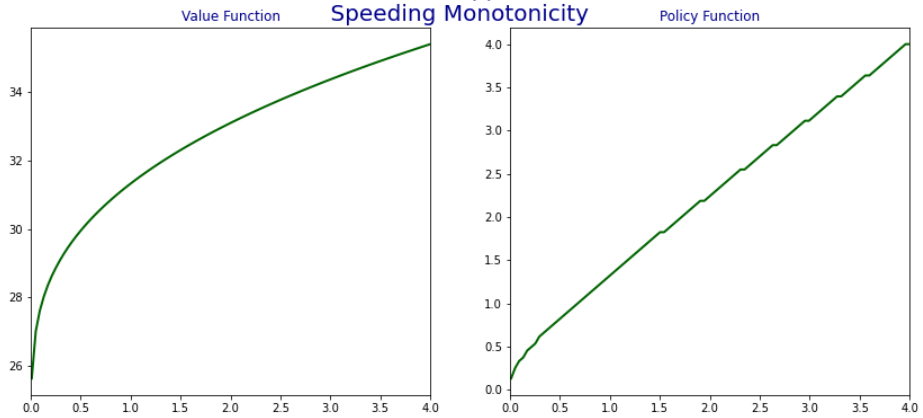
The results are both surprising and interesting. Firstly, the fastest is easily the brute force method. This questions the validity of whether these methods do in fact speed up the calculations. My theory for this is the following. In order to utilise both the monotonicity of the policy function and concavity of the value function i had to insert a second loop into the model. While when calculating the brute force method, since we have only one state variable you can simply iterate over the choice for k', in order to determine whether for example $k_j \geq g^k(k_i)$, you must iterate over both k and k'. Loops are notoriously slow and this removes any possible gain from implementing such methods.

Furthermore, while both the concavity and monotonicity methods provide similar results, the combination of the both is gives a time saving of around 33%. The local search method provides similar results.

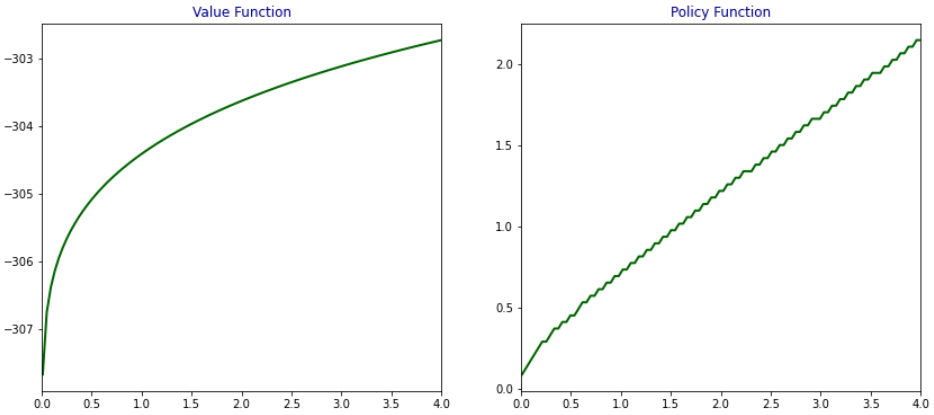
Value Function Approximation
Speeding Concavity



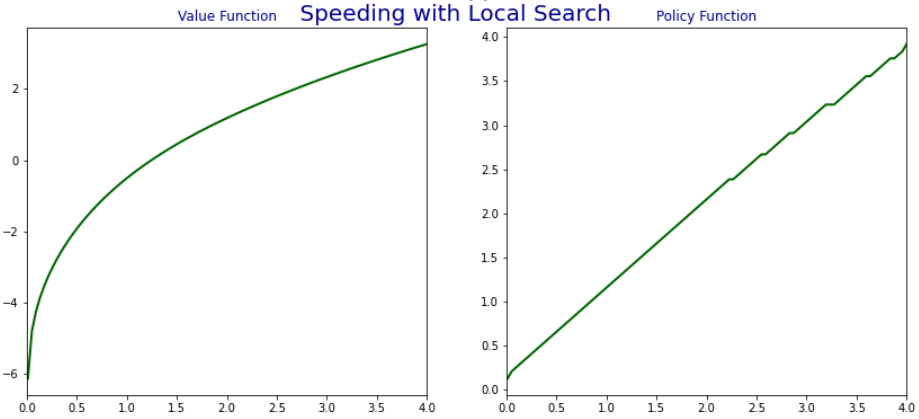
Value Function Approximation
Speeding Monotonicity



Speeding with Monotonicity & Concavity



Value Function Approximation
Speeding with Local Search



The policy functions are monotonic but not strictly monotonic. Overall the results mirror the main findings nicely. These findings are robust to changing both the boundaries on the grid for k, the grid size and stopping parameter.

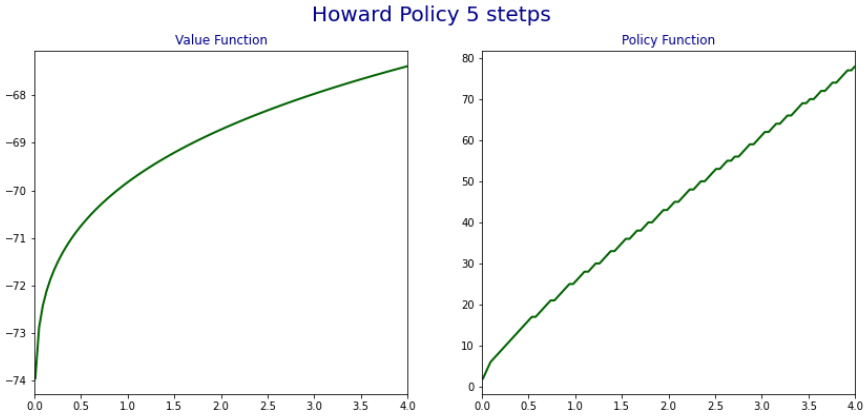
Howard Policy Improvement

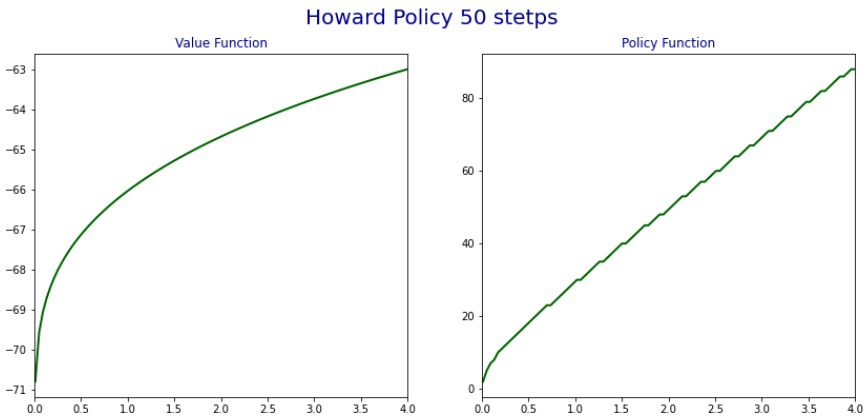
I have attempted to implement the Howard policy improvement algorithm below. First I calculated the optimal decision rule given the current V^s. Then proceed to iterate over this decision rule and the value function. I present my answers below,

Method	Time
Policy Iterations	
5	0.211s
10	0.198s
20	0.198s
50	0.203s

As you can see, when running the brute force method the time required was around 1.32 seconds. Therefore we are able to make serious time savings from implementing the Howard policy improvement despite the introduction of a new loop over the policy function. Calculating the max for each return matrix X requires iterating over every column, this is time consuming.

The results are presented in the graphs below for both 5 and 50 iterations. You can find the others in the accompanying jupyter notebook.



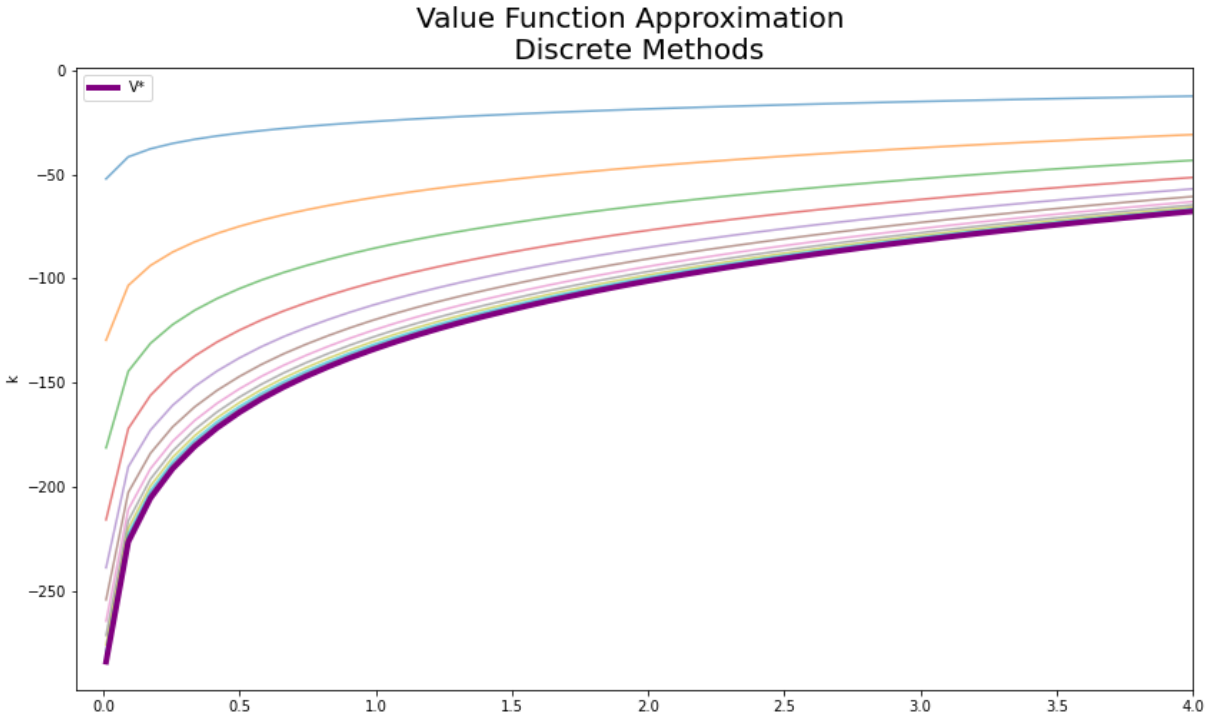


Endogenous Labour

We now introduce endogenous labour choice into the model. The utility for the agent is now given by,

$$u\left(c_t\right) = \ln c_t - \kappa \frac{h_t^{1+\frac{1}{\nu}}}{1+\frac{1}{\nu}}$$

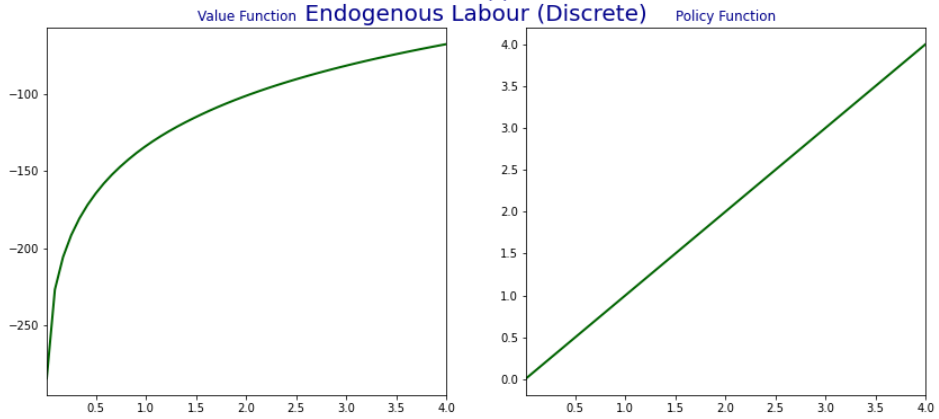
The rest of the parameters of the model remain the same. Labour supply is an additional choice variable, therefore given each state of the world of capital, the agent maximises utility by choosing both how much to work today and their savings today. The value function therefore remains a uni-variate function of capital today. The value function is presented in the graph below.



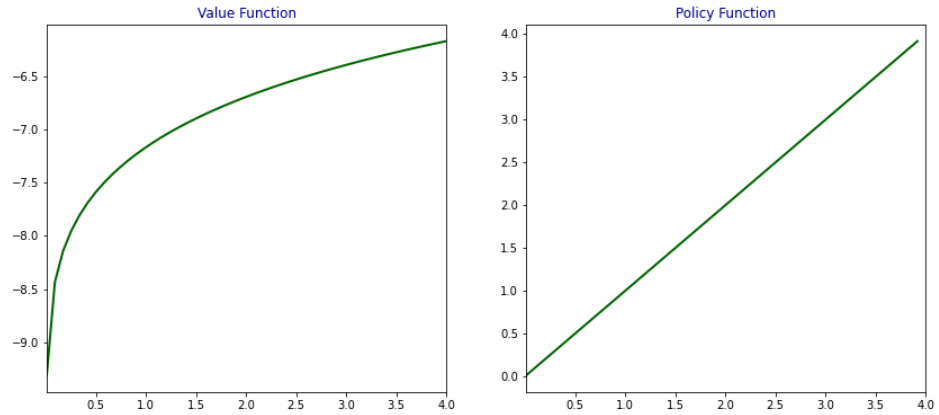
Again the value function displays the standard form. Introducing a second choice variable has increased the dimension of the problem. This is captured in the speed at which the program is solved. Below I present three extensions from before to demonstrate how when the dimensionality increases, speed savings are possible.

Value Function Approximation

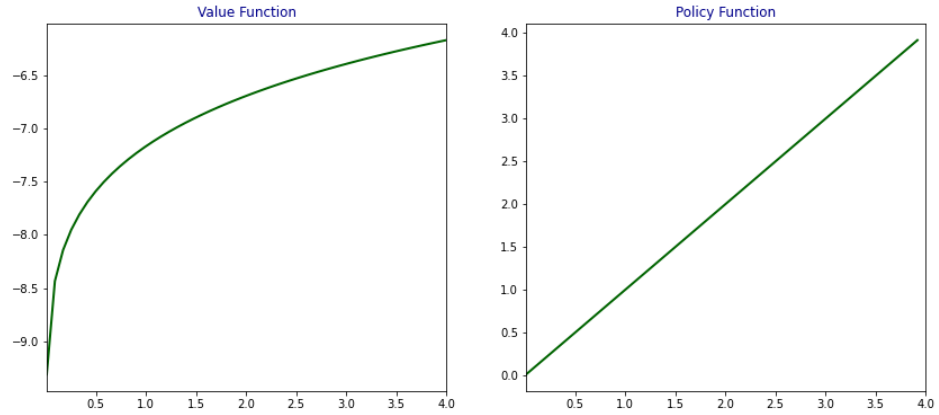
Endogenous Labour (Discrete)



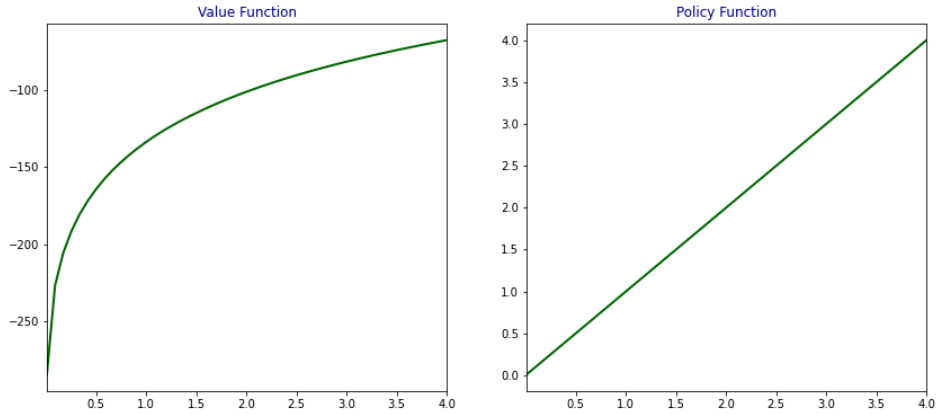
Endogenous Labour (Concavity)



Endogenous Labour (Concavity & Monotonicity)



Endogenous Labour (Monotonicity)



Method		Time
	$k = 50$	$k = 100$
Brute Force	23.2	4m18s
Concavity	9s	4m21s
Monotonicity	25s	52s
Local Search	7.8	53.7s

We can see that we are now seeing the speed improvements we expected to see in the previous section. Using the monotonicity of the policy function appears not to make too much difference, however the concavity of the value function makes a huge difference. The speed required for 50 runs drops from around 23 seconds to 9. The combination of both methods needs only 7.8 seconds to solve the program. These results are even more exaggerated when we increase the density of the grid. We can conclude then that for the speed improvements to take effect, we need to increase the dimensionality of the problem above the baseline case.

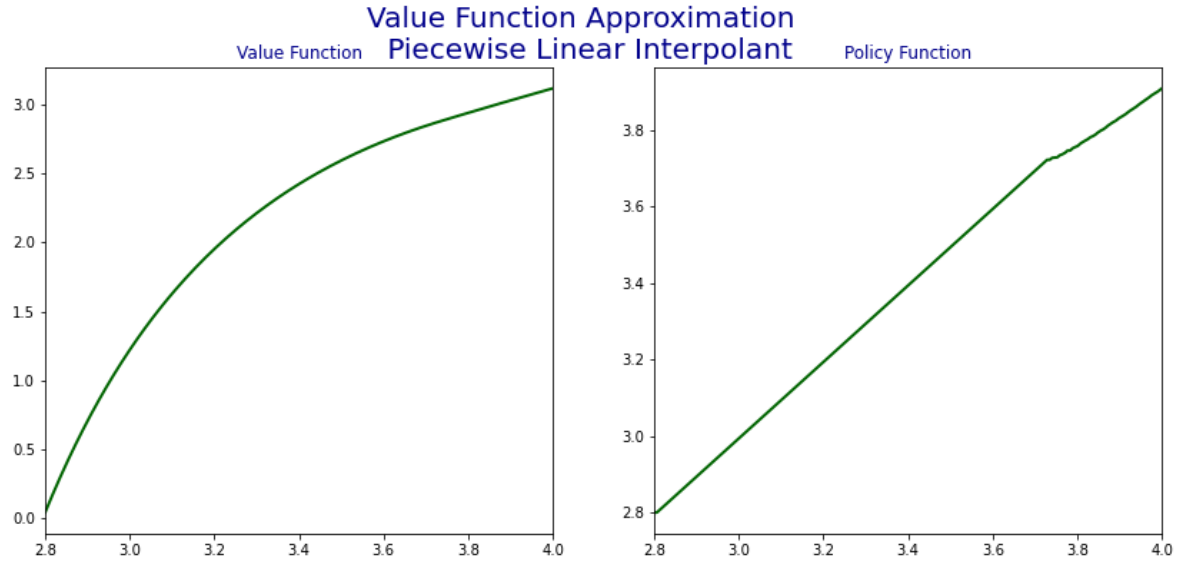
Again, these findings are robust to changing both the boundaries on the grid for k, the grid size and stopping parameter.

Continuous Approximation

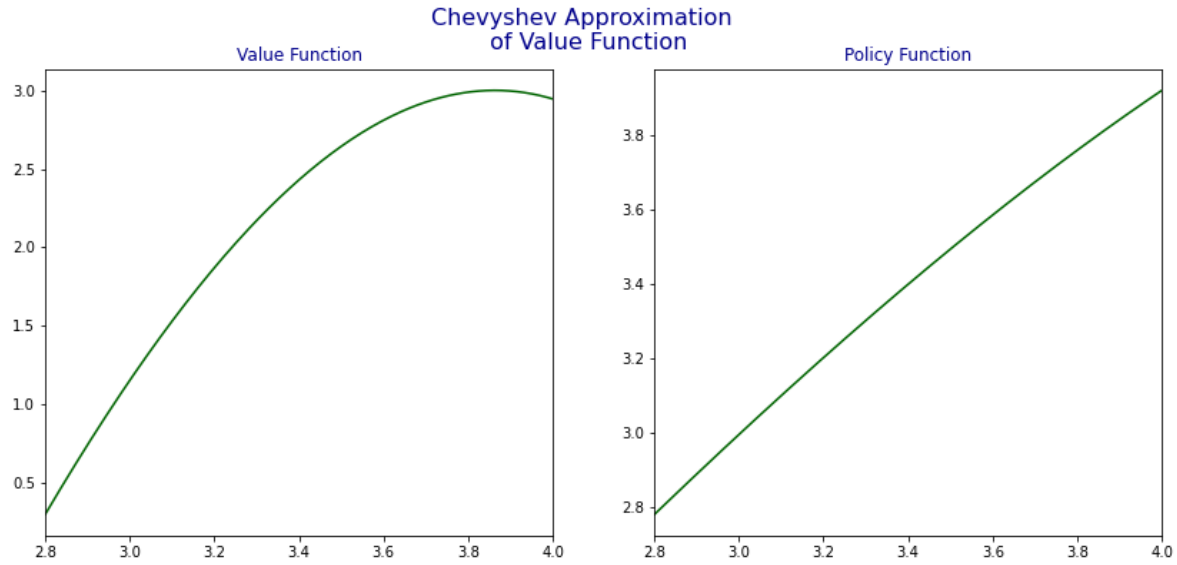
Finally, I have extended the problem to include a continuous approximation. Interestingly, interpolating the value function and storing it in it's functional form increased the speed of solving the problem greatly. The reason for this is that the computer no longer has to store and loop over the return matrix. As can be seen in the table below, the program now only takes 722 milliseconds to run. This is a huge gain in speed.

Method	Time
Continuous	
$k = 200$	722ms

This gain in speed however does come at a cost of stability. Overall, when working with continuous methods the output was more prone to wild fluctuations. I present two approaches below, the first is using a standard piece wise linear interpolate method. I then proceed to maximise using the non-linear solve fminbound to find the optimal policy function. There appears to be a slight error in the approximation of the policy function as k approaches the upper end of the boundaries.



In addition I present below a Chevyshev approximation of the final value function V^*



The Chevyshev approximation is produced through the Chevyshev regression algorithm. It appears to not suffer the same instability around the end of the policy function. The curvature of both value functions doesn't appear to match perfectly.

In conclusion I have presented a range of value function methods and how they can be optimised. All of the technical details are clearly outlined in the accompanying jupyter notebook files.