Using Value Function Iteration to Solve an Aiyagari Model with Training

Michael Bennett
Yale University

May 8, 2023

1 Model Description

I consider a partial equilibrium Aiyagari model, extended to include skill acquisition. I consider the optimisation decision of an agent with infinite planning horizon who dies with constant positive probability in each period over how much to consumer and how much schooling to take in each period. The agent prefers to consume more and to work less, where both market labour and training are considered work. The agent has control over how much schooling they undertake and how much to save at the prevailing market interest rate r, which abstracts from the production side of the model.

Workers move between employment ($e_t = 1$) and unemployment ($e_t = 0$) exogenously, but can choose to train in order to improve their productivity no matter their employment status. Higher productivity workers earn more when they are employed but receive the same unemployment benefits z as lower productivity workers. Agents face no liquidity constraints.

By analysing this model under different parameters, we will be able to investigate how consumption and training interact when there is idiosyncratic unemployment risk. We will be able to study how the distribution of training varies across potentially heterogeneous individuals who face different idiosyncratic shocks. We will also be able to investigate how the model responds to changes in the utility of leisure, the return to training, and the utility of consumption.

This is a model with two deterministic continuous state variables, one stochastic discrete state variable, and two choice variables. We will phrase this model in discrete time, and solve it using Value Function Iteration.

1.1 Sequential Problem

The model is precisely stated as follows. The agent solves the following problem.

$$\max_{c,h} U(c,l) = \mathbb{E}_0 \left\{ \sum_{t=0}^{\infty} \beta^t u(c_t, l_t) \right\} \tag{1.1}$$

subject to the constraints

$$a_{t+1} = z_t + (h_t - z_t)e_t(1 - s_t) + (1 + r)a_t - c_t \tag{1.2}$$

$$h_{t+1} = h(s_t, h_t) \tag{1.3}$$

$$l_t = e_t + (1 - e_t)s_t \tag{1.4}$$

$$a_t \geq \underline{a} \tag{1.5}$$

$$s_t \in [0, 1] \tag{1.6}$$

$$e_t \in \{0, 1\} \tag{1.7}$$

and a no-ponzi condition, where consumption is c_t, human capital is h_t, employment is e_t, work is l_t, schooling is s_t and assets are a_t. Employment e_t follows a two-state poisson process with exogenous job finding rate λ_f and job loss rate λ_l. We assume that $\lambda_l = 0.05$ and $\lambda_f = 0.95$, which ensures a steady state frictional unemployment rate of

$$u = \frac{\lambda_l}{\lambda_l + \lambda_f} = 0.05. \tag{1.8}$$

For concreteness, we will restrict the utility function to come from the family

$$u(c,l) = \frac{c^{1-\theta}}{1-\theta} - \gamma \frac{\varepsilon}{1+\varepsilon} l^{\frac{1+\varepsilon}{\varepsilon}} \tag{1.9}$$

which is a common form for such a utility function. It adds an isoelastic consumption utility function to a labour disutility function. In particular, note that the parameter ε is the Frisch elasticity of labour supply. Based on Chetty et al. (2011), we will use a value of $\varepsilon = 0.5$. Moreover, we will assume that θ, the degree of relative unevenness aversion in consumption, is equal to $\theta = 1.5$, as in Groom and Pr. (2019).

We will let the human capital production function to come from the family

$$h(s,h) = As^{\alpha}h^{\eta} + (1 - \delta)h. \tag{1.10}$$

We assume that partial training is possible to ensure that the problem is convex.

We can provide a lower bound $\underline{a}$ for a' using the natural borrowing limit, which arises from applying the no-ponzi condition to the worst case scenario for consumer income ($y = z$) when the consumer does all

they can to pay the debt back. This is

$$a' \geq -\frac{z}{r} = \underline{a}. \tag{1.11}$$

We will use $\underline{a}$ to inform our grid for a.

Moreover, we can also provide bounds for h. Given that schooling $s_t \in [0, 1]$, it is mechanically impossible for schooling to exceed a certain level, given positive δ. This is

$$h \leq \overline{h} = \left(\frac{A}{\delta}\right)^{\frac{1}{1-\eta}} \tag{1.12}$$

Moreover, from economic theory we know that a worker will never let their level of human capital (which is isomorphic to their wage) fall below the level of unemployment benefits z. This provides a lower bound for h of

$$h \geq \underline{h} = z \tag{1.13}$$

1.2 Recursive Form

To solve this problem using value function iteration, we need to write it in its recursive form. This is:

$$V(h, a, e) = \max_{h', a'} \{u(c, l) + \beta \mathbb{E}[V(h', a', e') \,|e]\} \tag{1.14}$$

such that

$$\begin{aligned} a' &= z + (h - z)e(1 - s) + (1 + r)a - c \\ h' &= h(s, h) = As^{\alpha}h^{\eta} + (1 - \delta)h \\ l &= e + (1 - e)s \end{aligned} \tag{1.15}$$

We can rewrite these constraints:

$$\begin{aligned} c &= z + (h - z)e(1 - s(h')) + (1 + r)a - a' \\ l &= e + (1 - e)s(h') \\ s(h') &= \left(\frac{h' - (1 - \delta)h}{Ah^{\eta}}\right)^{\frac{1}{\alpha}} \end{aligned} \tag{1.16}$$

Thus, we can rewrite the problem:

$$\begin{aligned} V(h, a, e) = \max_{h', a'} \{ & u\left[z + (h - z)\, e\, (1 - s(h')) + (1 + r)a - a', e + (1 - e)s(h')\right] \\ & + \beta \mathbb{E}\left[V\left(h', a', e' | e\right)\right] \} \end{aligned} \tag{1.17}$$

such that

$$\begin{aligned} s(h') &= \left(\frac{h' - (1 - \delta)h}{Ah^{\alpha}}\right)^{\frac{1}{\alpha}} \\ h' &\in [(1 - \delta)h, (1 - \delta)h + Ah^{\eta}] \\ a' &\geq \underline{a} \end{aligned} \tag{1.18}$$

Which can be written

$$V(h,a,e) = \max_{h',a'} \{v(h,a,e,h',a') + \beta \mathbb{E}\left[V\left(h',a',e'|e\right)\right]\} \tag{1.19}$$

such that

$$\begin{aligned} h' &\in [(1-\delta)h, (1-\delta)h + Ah^{\eta}] \\ a' &\geq \underline{a} \end{aligned} \tag{1.20}$$

Which in fact defines the system of equations

$$V(h,a,0) = \max_{h'_0,a'_0} \{v(h,a,0,h'_0,a'_0) + \beta \mathbb{E}\left[V\left(h'_0,a'_0,e'_0|e\right)\right]\} \tag{1.21}$$

$$V(h,a,1) = \max_{h'_1,a'_1} \{v(h,a,e,h'_1,a'_1) + \beta \mathbb{E}\left[V\left(h'_1,a'_1,e'|e\right)\right]\}. \tag{1.22}$$

This representation is the basis for our solution method of value function iteration.

1.3 Parameter Values

To solve this model, we need to choose value for the parameters. Some, discussed above, are broadly informed by economic theory and evidence, but I will also set some values somewhat arbitrarily. The chosen parameter values are listed below.

$$\begin{aligned} \beta &= 0.9 \\ \delta &= 0.1 \\ \alpha &= 1 \\ \eta &= 0 \\ A &= 1.5 \\ \gamma &= 1 \\ \theta &= 1.5 \\ \varepsilon &= 0.5 \\ z &= 1 \\ r &= 0.1 \\ \lambda_f &= 0.95 \\ \lambda_l &= 0.05 \end{aligned} \tag{1.23}$$

I have already discussed the justification for θ, ε, λ_f and λ_l. I chose $z = 1$ as it is essentially a numeraire, $\eta = 0$ to ensure and I chose A and δ to get an interesting value of $\overline{h}$, which represents the difference in productive capacity between those with the most education and those with no education, at around 15.

2 Method: Value Function Iteration

I tried several different techniques. I found the most basic method I tried, numerical value function iteration without interpolation, to be most robust. In the appendix, I will discuss the other methods I attempted.

2.1 Algorithm

Consider our recursive form problem from (1.19). We can write the Bellman operator $T: \mathscr{C}(\mathbb{R}^2) \to \mathscr{C}(\mathbb{R}^2)$ as

$$TV(h,a,e) = \max_{h',a'} \left\{ v(h,a,e,h',a') + \beta \mathbb{E}\left[V\left(h',a',e'|e\right)\right]\right\}. \tag{2.1}$$

By Blackwell's sufficient conditions, T is a contraction mapping, so repeatedly applying T to any candidate function V will converge to a unique fixed point V^*. We apply this technique on a discretised grid of points in the state space.

For h and a, we define a grid of N_h and N_a evenly spaced points between pre-determined minimum and maximum values of h and a respectively. For h, I define the grid between $\underline{h}$ and $\overline{h}$ as discussed prior. For a, I define the grid between $\underline{a}$, as discussed above, and $-\underline{a}$, which ends up working out. Since e is discrete and binary, we can simply consider the grid $[0,1]$. We also define a 2×2 0-indexed matrix of transition probabilities that takes the form

$$\Lambda = \begin{pmatrix} 1-\lambda_f & \lambda_f \\ \lambda_l & 1-\lambda_l \end{pmatrix} \tag{2.2}$$

with the interpretation that Λ_{ij} is the probability of transitioning from $e = i$ to $e' = j$.

Having initialised the grids, we can calculate the flow values for each combination of (h,a,e,h',a') and store these in a five-dimensional array. If the tuple (h,a,e,h',a') is infeasible, we simply let the flow value at that point be $-\infty$. Then, we start with an initial guess of V, which is a $N_h \times N_a \times 2$ array. Somewhat arbitrarily, I take this to be equal to the maximum value of the flow value over (h',a') at each grid point (h,a,e).

Starting with this initial guess, we maximise the right hand side of the bellman equation over h' and a' at each grid point (h,a,e) by calculating the candidate value

$$V_{\text{cand}} = v(h,a,e,h',a') + \beta \mathbb{E}\left[V\left(h',a',e'|e\right)\right] \tag{2.3}$$

at each grid point and then choosing the largest value.

We then repeat this process until the difference between the new value of V and the old value of V is sufficiently small. This is the value function iteration algorithm.

2.2 Performance

Since this problem has two continuous state and choice variables each, we start to run into the curse of dimensionality in Dynamic Programming. That is, assuming that $N_a = N_h = n$, both the runtime and memory usage of this VFI algorithm scale at $O(n^4)$ in the best case. That means that doubling N will lead to a 16-fold increase in runtime and memory usage. Increasing N by an order of magnitude will increase runtime and memory usage by 10,000-fold. In other words, this algorithm scales incredibly poorly.

Such poor performance at scale represents a problem for this algorithm because it means that we need to dramatically reduce the number of gridpoints in order to achieve a solution in a reasonable amount of time. However, reducing the number of gridpoints reduces the accuracy of the solution, potentially dramatically.

There are two potential ways around this. One method is to optimise the performance of this algorithm as much as possible to reduce the runtime and hence allow for a larger number of gridpoints. However, the gains to this are ultimately limited by the $O(n^4)$ scaling of the algorithm, which means that even substantial performance optimisations will only allow for a modest increase in the number of gridpoints. The other method is to seek alternative algorithms with better scaling properties or that produce more accurate results with fewer gridpoints.

Ultimately, I attempted both methods, but was unsuccessful at getting any other method to work for this problem in the time available. Thus, I will discuss the other methods I attempted and the problems I encountered in the appendix, and in the following section I will discuss the performance optimisations I implemented for the algorithm described above.

2.3 Optimisations

To improve the runtime of the algorithm, I have implemented several optimisations which have allowed me to increase the number of gridpoints which achieving a solution in a given time. These optimisations come in two forms: some are based on the structure of the problem, and some are purely computational.

2.3.1 Problem Structure

The following optimisations are based on the structure of the problem.

- Lowering β to 0.9 to increase the rate of convergence of the Bellman operator.
- Since consumption is required to be positive and schooling must be in the unit range, many gridpoints are not feasible. When we pre-compute the array of flow values, we set the value to $-\infty$ if the gridpoint is not feasible. Then, when optimising the value function within each iteration, we know that an (h', a') pair for (h, a, e) cannot possibly be optimal if that point is not finite in the flow value matrix, so we can skip all such points in the maximisation. This is an optimisation that leverages the sparsity of the problem to dramatically reduce the number of points that need to be considered. I found that this lead to an approximate 50-fold speedup.

2.3.2 Computational Optimisations

The following optimisations are purely computational.

- Julia supports multithreading using the `Threads.@threads` macro. This allows multiple 'threads' to access the same memory, which I use to parallelise the computations required to update the value function within each iteration. By setting the environment variable `JULIA_NUM_THREADS=auto` in my `.zshrc` (on MacOS), Julia automatically uses all eight threads available on my machine, leading to slightly less than 8-fold speedup, allowing for thread management overhead, multiple concurrent memory demand and kernel prioritisation.
- Precise numerical calculations are expensive. We can relax the degree of precision without overly impacting the accuracy of the solution, thereby improving both runtime and memory usage. To reduce precision, I used 32-bit floats instead of 64-bit floats, enabled the `@fastmath` macro in the inner loop to enable faster floating point arithmetic, and set `set_zero_subnormals(true)` to set very small floating point numbers to zero. None of these changes had a significant affect on computational accuracy when applied to small test grids, but did lead to a speedup of approximately 2-fold in total.
- Similarly, Julia also checks that any array element you are attempting to index is actually in the array. Since my code makes heavy use of array indexing, I turned this check off using the `@inbounds` macro after verifying that all array indexing was valid on a smaller test grid.

2.4 Implementation

With the above optimisations implemented, I was able to run the VFI algorithm on a grid of $N_h = N_a = 200$ points in approximately six hours on my machine. While 200 grid points is not ideal, it still allowed me to get some interesting results which I discuss below.

3 Response Functions

First, I will present and discuss the value and policy functions produced by the model. These show the optimal responses of the agent to the state variables.

Alongside the following plots, which are static views of one angle of a 3D plot, you can view interactive 3D plots of the value and policy functions provided on my GitHub Pages site or as in the folder `code/plots` in my submission.

3.1 Value Function

The value function shows the welfare of an agent for each combination of the state variables.

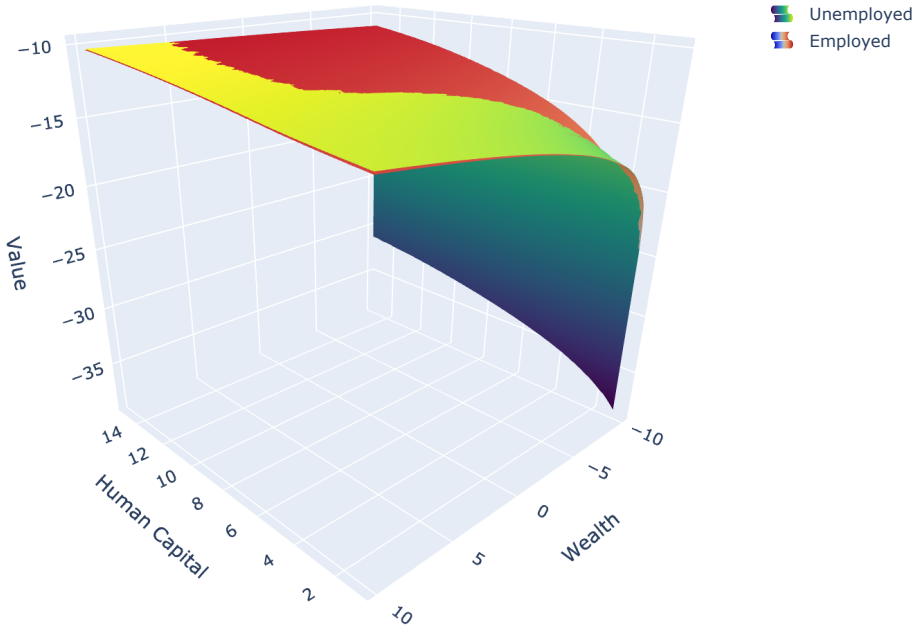

Figure 1: Value Function Plot

We see that welfare is strongly increasing and concave in wealth, but not increasing very much in human capital. This is because it is relatively easy to acquire human capital in this model. Since the agent gets unemployment benefits even if they could be working, the opportunity cost of schooling is low. Thus, an agent with low human capital will soon have high human capital, and so the value function is not very sensitive to human capital.

Also notice that the value of employment and unemployment is very similar. This is because the job finding rate is high, so the expected duration of unemployment is low. It is sometimes better to be unemployed because it allows the agent to take some leisure. This is an artifact of the exogenous job search and matching process, but I decided to spare my computer the effort that another endogenous state variable would require.

3.2 Schooling Policy Function

The schooling policy function shows the optimal level of schooling for each combination of the state variables.

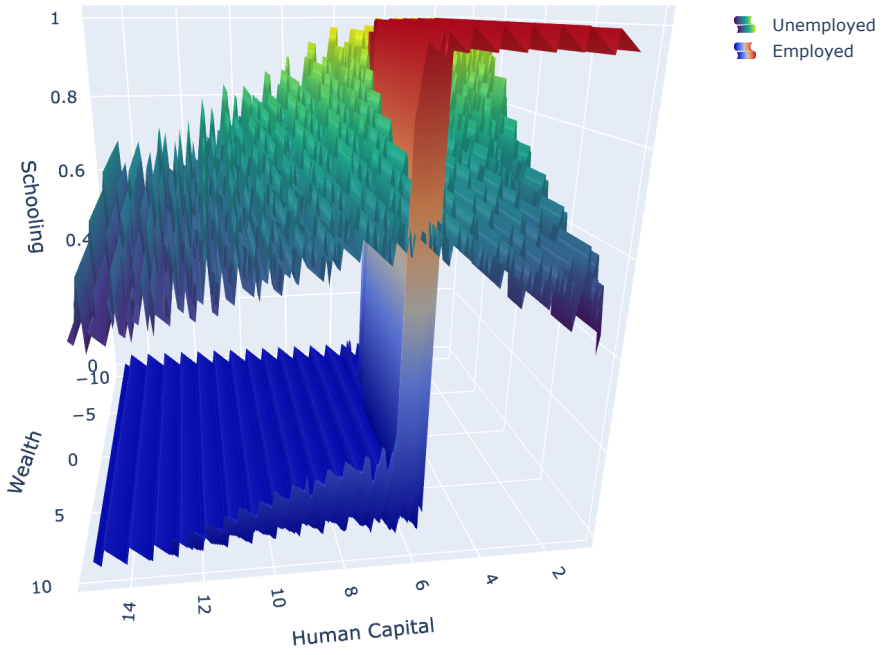

Figure 2: Schooling Policy Function Plot

The jaggedness of this plot is an artifact of the sparseness of the grid. We will ignore this and look at the general shape of the function.

When the agent is employed, we see that the constraint on schooling introduces a jump in the policy function. Optimally, an employed agent wants to have about 6 units of human capital. If schooling was unlimited, they would want to get all this schooling at once since there are no adjustment costs to human capital accumulation. However, since schooling cannot go above a certain value, the agent will instead get as much schooling as they can when their human capital is low, and none when their human capital is high.

Meanwhile, unemployed agents optimise compare the value of leisure and human capital. Agents with low assets and low human capital want more schooling to increase their ability to pay off their financial obligations and increase their level of human capital. Interestingly, when assets are high and human capital is low, optimal schooling is actually decreasing in human capital. This is because rich agents value leisure relatively highly, and so prefer to take leisure now and use their savings to study later when they are employed again. The lower the level of existing human capital, the lower is the cost of taking schooling over work once one is employed again.

3.3 Consumption Policy Function

The consumption policy function shows the optimal level of consumption for each combination of the state variables.

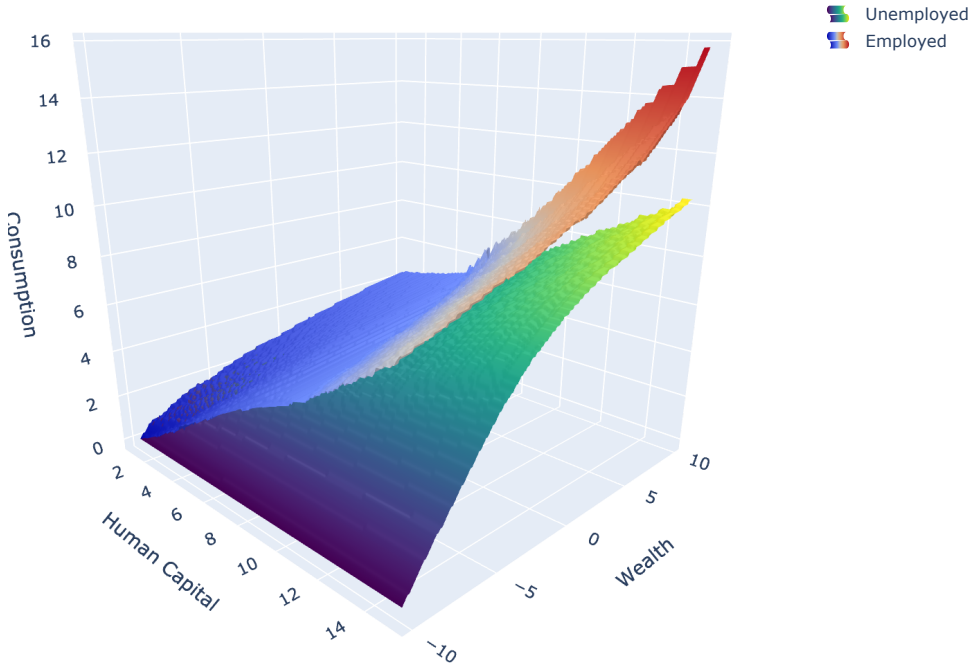

Figure 3: Consumption Policy Function Plot

As we would expect, we see that consumption is increasing in both wealth and human capital, and that employed people consume more than unemployed people.

3.4 Savings Policy Function

The consumption policy function shows the optimal level of consumption for each combination of the state variables.

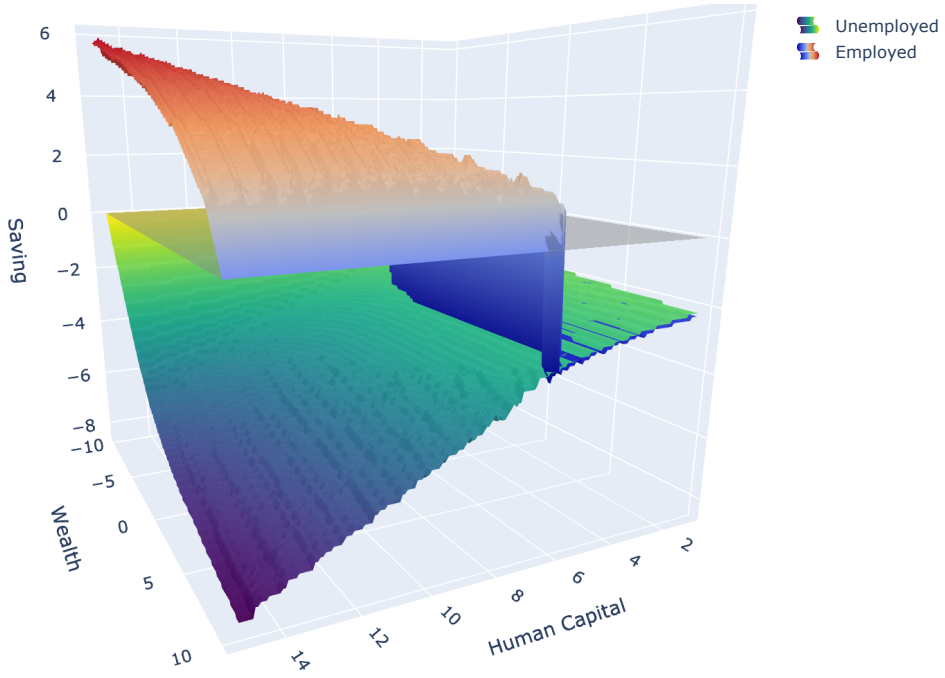

Figure 4: Savings Policy Function Plot

We see that those who are employed and have high human capital are generally savers and those who are unemployed or have low human capital are generally spenders.

For the employed, those who have high human capital are working at high wage rates, and so can afford to save more. Meanwhile, those who are unemployed or have low human capital are earning z per period, but know they will have higher income in the future. Thus, they borrow against their future income to finance consumption now. The schooling cliff we saw above is what causes this savings cliff here, as individuals smooth their consumption over time.

4 Asset Distributions

In the Aiyagari model, we can also simulate time paths of the state variables for an agent under random employment shocks. This allows us to see how randomness can cause the outcomes of identical individuals to diverge over time.

4.1 Simulation

I simulated the model for 100 periods 10000 times. I set initial $h_0 = 1$ to 1 and $a_0 = 0$ for all agents, reflecting being born with no assets and the replacement level of human capital. I interpolated used a bicubic spline to interpolate the policy function for each choice variable for each possible value of employment separately, and then I used these functions and random draws of employment to calculate the relevant quantities for each agent.

4.2 Results

To analyse the results of my simulation, I plotted the bivariate distribution of assets and human capital for each t. The distribution stabilised, so I present here only the results for $t = 100$.

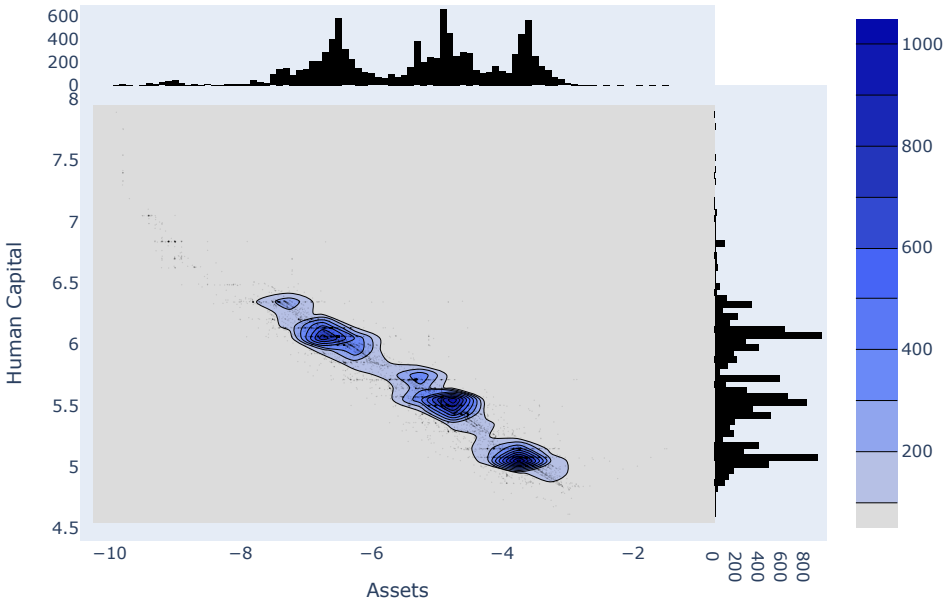

Figure 5: Bivariate Distribution of Assets and Human Capital at $t = 100$

This plot shows that in the long run there is significant variation in the levels of assets and human capital that agents have. Moreover, there is a very strong negative relationship between assets and human capital. This is because agents who are made unemployed increase their human capital, but borrow against their future consumption. Thus, agents who are unemployed are likely to have higher human capital and low assets, while agents who are employed are likely to have low human capital and high assets.

5 Conclusions

I have solved an Aiyagari model with human capital accumulation and exogenous job search and matching using Value Function Iteration. I have characterised the best responses of a consumer to this partial equilibrium problem, and I have found that the model has a unique stationary distribution of assets and human capital, and that assets and human capital are negatively correlated in this distribution.

A Extended Solution Methods

I attempted to implement several alternative solution methods, but I was not able to get them working in the time available. Here, I will describe my attempts, what went wrong, how any problems might be fixed, and what the advantages of each method is.

A.1 Interpolation

I attempted to use interpolation to approximate the value function over (h', a') for each (h, a, e) in order to use a root-finding algorithm to find the optimal policy function precisely. I used a bicubic spline to interpolate the value function over (h', a'), and I hoped that this would improve the performance of the method by allowing me to get accurate, smooth estimates of the value and policy functions even over a very sparse grid.

One problem with this method was that the set of feasible (h', a') for each (h, a, e) was a non-rectangular subarray of the grid of (h', a') values. One method I tried to solve this was to find the largest rectangular subarray of the grid that was feasible, and then to interpolate over this subarray. Unfortunately, this method faced issues with singular jacobians, which seemed to be caused by problems around the edge of the box. These issues were slowing me down substantially, so I decided to move on.

However, I think there is still promise for this method given more time to investigate the issues. Sadly, it was time I did not have.

A.2 Policy Function Iteration

I also tried to do approximate numerical policy function iteration. To do this, I started with a guess of the policy function, and then used repeated evaluation of the value function for this policy to approximate the value function. I then maximised over the right hand side of the bellman equation to find a new policy function and repeated this process until the new policy function and the old policy function were identical.

The promise of this method was that it would require less iterations to converge than value function iteration, as is a known trait of policy function iteration. Indeed this was true. Policy function iteration converged very quickly. However, the error in the policy and value functions was substantial, likely due to the coarseness of the grid and imprecision in the value function I was calculating.

In reality, it is likely that this method requires first order information from the Euler equations for this model to be accurate.

A.3 First Order Information

As such, I decided to derive the optimality conditions for this problem. I present the derivations below, but I was not ultimately able to make any progress on implementations using the four resulting functional equations due to lack of time. However, this problem has a form that is mostly analytically quite tractable, apart from the slight complication introduced by the constraint on feasible schooling. It is likely that an implementation of PFI using this method would be faster and more accurate than my naïve VFI implementation.

We restate the problem in our two-state case as

$$\begin{aligned} V(h,a,e) = \max_{h',a'} \{ & u\left[z + (h-z)\, e\, (1-s(h')) + (1+r)a - a', e + (1-e)s(h')\right] \\ & + \beta \mathbb{E}\left[V\left(h', a', e' | e\right)\right] \} \end{aligned} \tag{A.1}$$

such that

$$\begin{aligned} s(h') &= \left(\frac{h' - (1-\delta)h}{Ah^\eta} \right)^{\frac{1}{\alpha}} \\ h' &\in [(1-\delta)h, (1-\delta)h + Ah^\eta] \\ a' &\geq \underline{a} \end{aligned} \tag{A.2}$$

For ease, I search only for interior solutions, ignoring the constraint on schooling. Let us first note that

$$\begin{aligned} u_c'(c,l) &= c^{-\theta} \\ u_l'(c,l) &= -\gamma l^{\frac{1}{\varepsilon}} \\ s'(h') &= \frac{1}{\alpha} \left(\frac{h' - (1-\delta)h}{Ah^\eta} \right)^{\frac{1-\alpha}{\alpha}} \cdot \frac{1}{Ah^\eta} = \frac{s(h')}{\alpha(h' - (1-\delta)h)} \\ \frac{\partial s(h')}{h} &= \frac{1}{\alpha} \left(\frac{h' - (1-\delta)h}{Ah^\eta} \right)^{\frac{1-\alpha}{\alpha}} \cdot \frac{-(1-\delta)Ah^\alpha - \eta Ah^{\eta-1}(h' - (1-\delta)h)}{(Ah^\eta)^2} \end{aligned} \tag{A.3}$$

Then, the FOCs for the Bellman Equation are

$$\frac{\partial V(h,a,e)}{\partial h'} = -u_c'(c,l)(h-z)es'(h') + u_l'(c,l)(1-e)s'(h') + \beta \mathbb{E}\left[V_h'(h',a',e')|e\right] =^* 0 \tag{A.4}$$

$$\frac{\partial V(h,a,e)}{\partial a'} = -u_c'(c,l) + \beta \mathbb{E}\left[V_a'(h',a',e')|e\right] =^* 0 \implies u_c'(c,l) = \beta \mathbb{E}\left[V_a'(h',a',e')|e\right] \tag{A.5}$$

Moreover, the envelope theorem gives us

$$V_h'(h,a,e) = u_c'(c,l) \cdot \left[e(1-s(h')) - (h-z)e\frac{\partial s(h')}{h} \right] + u_l'(c,l) \cdot (1-e)\frac{\partial s(h')}{h} \tag{A.6}$$

$$V_a'(h,a,e) = (1+r)u_c'(c,l) \tag{A.7}$$

Thus, now considering the states $e = 1$ and $e = 0$ separately, we have

$$\frac{\partial V(h,a,0)}{\partial h_0'} = u_l'(c_0,l_0)s'(h_0') + \beta\mathbb{E}\left[V_h'(h_0',a_0',e')|0\right] =^* 0 \tag{A.8}$$

$$\frac{\partial V(h,a,1)}{\partial h_1'} = -u_c'(c_1,l_1)(h-z)s'(h_1') + \beta\mathbb{E}\left[V_h'(h_1',a_1',e')|1\right] =^* 0 \tag{A.9}$$

$$\frac{\partial V(h,a,0)}{\partial a_0'} = -u_c'(c_0,l_0) + \beta\mathbb{E}\left[V_a'(h_0',a_0',e')|0\right] =^* 0 \implies u_c'(c_0,l_0) = \beta\mathbb{E}\left[V_a'(h_0',a_0',e')|0\right] \tag{A.10}$$

$$\frac{\partial V(h,a,1)}{\partial a_1'} = -u_c'(c_1,l_1) + \beta\mathbb{E}\left[V_a'(h_1',a_1',e')|1\right] =^* 0 \implies u_c'(c_1,l_1) = \beta\mathbb{E}\left[V_a'(h_1',a_1',e')|1\right] \tag{A.11}$$

$$V_h'(h,a,0) = u_l'(c_0,l_0)\frac{\partial s(h_0')}{\partial h} \tag{A.12}$$

$$V_h'(h,a,1) = u_c'(c_1,l_1)\cdot\left[1 - s(h_1') - (h-z)\frac{\partial s(h_1')}{h}\right] \tag{A.13}$$

$$V_a'(h,a,0) = (1+r)u_c'(c_0,l_0) \tag{A.14}$$

$$V_a'(h,a,1) = (1+r)u_c'(c_1,l_1) \tag{A.15}$$

Iterating the envelope conditions one period forward and substituting into the first order conditions, we get four optimality conditions:

$$u_l'(c_0,l_0)s'(h_0') + \beta\left[\pi_{00}u_l'(c_0',l_0')\frac{\partial s(h_0'')}{\partial h_0'} + \pi_{01}u_c'(c_0',l_0')\left[1 - s(h_1'') - (h_0'-z)\frac{\partial s(h_1'')}{h_0'}\right]\right] =^* 0 \tag{A.16}$$

$$-u_c'(c_1,l_1)(h-z)s'(h_1') + \beta\left[\pi_{10}u_l'(c_1',l_1')\frac{\partial s(h_0'')}{\partial h'} + \pi_{11}u_c'(c_1',l_1')\left[1 - s(h_1'') - (h'-z)\frac{\partial s(h_1'')}{h'}\right]\right] =^* 0 \tag{A.17}$$

$$u_c'(c_0,l_0) = \beta(1+r)\left[\pi_{00}u_c'(c_0',l_0') + \pi_{01}u_c'(c_0',l_0')\right] \tag{A.18}$$

$$u_c'(c_1,l_1) = \beta(1+r)\left[\pi_{10}u_c'(c_1',l_1') + \pi_{11}u_c'(c_1',l_1')\right]. \tag{A.19}$$

Then, letting μ_{he} and μ_{ae} be the policy functions for h and a respectively in each state of nature, and μ_{ce}, μ_{le} be the derived policy functions for c and l, we have

$$\begin{aligned} &u_l'(\mu_{c0},\mu_{l0})s'(\mu_{h0}) \\ &\quad+\beta\Bigg[\pi_{00}u_l'(\mu_{c0}\circ\mu_0,\mu_{l0}\circ\mu_0)\frac{\partial s(\mu_{h0}\circ\mu_{h0})}{\partial\mu_{h0}} \\ &\qquad+\pi_{01}u_c'(\mu_{c1}\circ\mu_0,\mu_{l1}\circ\mu_0)\left[1 - s(\mu_{h1}\circ\mu_{h0}) - (\mu_{h0}-z)\frac{\partial s(\mu_{h1}\circ\mu_{h0})}{\mu_{h0}}\right]\Bigg] =^* 0 \end{aligned} \tag{A.20}$$

This is a functional equation in $\mu_{h1}, \mu_{h0}, \mu_{a1}, \mu_{a0}$. If we did the above for the remaining three optimality conditions, we would have a set of four functional equations in four unknowns. We could then solve for the policy functions using policy function iteration via the coleman operator.

References

Chetty, R., Guren, A., Manoli, D. and Weber, A. (2011), 'Are micro and macro labor supply elasticities consistent? a review of evidence on the intensive and extensive margins', *American Economic Review* **101**(3), 471–75.
URL: *https://www.aeaweb.org/articles?id=10.1257/aer.101.3.471*

Groom, B. and Pr., D. M. (2019), 'New estimates of the elasticity of marginal utility for the uk', *Environmental and Resource Economics* **72**, 1155–1182.