Replication of the Aiyagari Model with Fat-tailed Wealth Distribution by Achdou et. al. 2014

Carlos Lizama

June 3, 2016

1 The Model

1.1 Description of the Model

This is a partial equilibrium model. Agents are heterogeneous in their productivity z and their wealth a. The state of the economy is the joint distribution of income and wealth. Individual's preferences are given by

$$\mathbb{E}_0 \int_0^\infty e^{-\rho} u(c_t) dt$$

where c_t is the consumption flow at time t, ρ is the discount rate and u is strictly increasing and strictly concave. In particular, $u(c_t) = \frac{c^{1-\gamma}}{1-\gamma}$.

The productivity process follows a two-state Poisson process $z_t \in \{z_1, z_2\}$. The process jumps from state 1 to state 2 with intensity λ_1 and vice versa with intensity λ_2.

There are two assets in the economy. A riskless bond in zero net-supply b_t and a risky asset k_t. The budget constraint is

$$db_t + dk_t = (z_t + \tilde{R}_t k_t + r_t b_t - c_t)dt$$

where r_t is the return on the riskless asset and $\tilde{R}_t$ is the return on the risky asset which follows the following stochastic process

$$\tilde{R}_t = R + \sigma dW_t$$

Negative position are not allowed for k_t and there is a borrowing constraint on the riskless bond $b_t \geq -\phi$. It is easy to show that the budget constraint in terms of wealth $a_t = b_t + k_t$ reads

$$da_t = (z_t + r_t a_t + (R - r)k_t - c_t)dt + \sigma k_t dW_t$$

1.2 Stationary Equilibrium

The individuals' saving decisions and the joint distribution of income and wealth can be summarized by the Hamilton-Jacobi-Bellman equation and the Kolmogorov Forward equation:

$$\rho v_i(a) = \max_{c, 0 \leq k \leq a+\phi} u(c) + v_i'(a)(z_i + ra + (R-r)k - c) + \frac{1}{2} v_i''(a)\sigma^2 k^2 + \lambda_i(v_{-i}(a) - v_i(a))$$

$$0 = -\frac{d}{da}[s_i(a)g_i(a)] + \frac{1}{2}\frac{d^2}{da^2}[\sigma^2 k_i(a)^2 g_i(a)] - \lambda_i g_i(a) + \lambda_{-i} g_{-i}(a)$$

where $s_i(a) = z_i + ra + (R-a)k - c$ and $g_i(a)$ is the stationary distribution of assets for type i agents.

The interest rate r is determined in equilibrium by the fact that bonds are in zero net supply. Hence, the bond market condition is[1]

$$\int_{\underline{a}}^{\infty} k_1(a)g_1(a)da + \int_{\underline{a}}^{\infty} k_2(a)g_2(a)da = \int_{\underline{a}}^{\infty} ag_1(a)da + \int_{\underline{a}}^{\infty} ag_2(a)da$$

2 The Algorithm

As in Achdou et. al., I will solve the partial equilibrium for this economy. A natural extension is the general equilibrium, although some issues arise.

The system to be solved is

$$\rho v_i(a) = \max_{c, 0\le k \le a+\phi} u(c) + v_i'(a)(z_i + ra + (R-r)k - c) + \frac{1}{2}v_i''(a)\sigma^2 k^2 + \lambda_i(v_{-i}(a) - v_i(a)) \tag{1}$$

$$0 = -\frac{d}{da}[s_i(a)g_i(a)] + \frac{1}{2}\frac{d^2}{da^2}[\sigma^2 k_i(a)^2 g_i(a)] - \lambda_i g_i(a) + \lambda_{-i}g_{-i}(a) \tag{2}$$

From the first order conditions, the optimal decisions of consumption and risky assets are

$$c_i(a) = v_i'(a)^{-\frac{1}{\gamma}} \tag{3}$$

$$k_i(a) = \min\left\{-\frac{v_i'(a)}{v_i''(a)}\frac{(R-r)}{\sigma^2}, a+\phi\right\} \tag{4}$$

2.1 Finite Difference Method for HJB equation with non-uniform grid

Let i be the index of the type of the agent and j the index for asset, where $i \in \{1,2\}$ and $j \in \{1,\dots,J\}$ where J is the number of points in the asset grid. Thus v_{ij} is the value function of the agent of type i with assets a_j.

Denote by $\Delta a_{j,+} = a_{j+1} - a_j$ and $\Delta a_{j,-} = a_j - a_{j-1}$.

Define the forward and backward difference approximation for the derivative $v_{i,j}' = v_i(a_j)$ as:

$$v_{i,j,F}' \approx \frac{v_{i,j+1} - v_{i,j}}{a_{j+1} - a_j} = \frac{v_{i,j+1} - v_{i,j}}{\Delta a_{j,+}}$$

$$v_{i,j,B}' \approx \frac{v_{i,j} - v_{i,j-1}}{a_j - a_{j-1}} = \frac{v_{i,j} - v_{i,j-1}}{\Delta a_{j,-}}$$

The approximation for the second derivative is[2]

$$v_{i,j}'' \approx \frac{\Delta a_{j,-}v_{i,j+1} - (\Delta a_{j,-} + \Delta a_{j,+})v_{i,j} + \Delta a_{j,+}v_{i,j-1}}{\frac{1}{2}(\Delta a_{j,+} + \Delta a_{j,-})\Delta a_{j,-}\Delta a_{j,+}}$$

Brief description of the algorithm

Given an initial guess $v_{i,j}^0$, the algorithm to solve the HJB equation is.

1. Compute $v_{i,j}^{n\,\prime}$ using an upwind scheme. The upwind scheme helps us to decide which approximation is best, forward or backward, in every grid point.

[1] This condition should be used for the general equilibrium. However, in the codes only the partial equilibrium version is solved.
[2] See the paper appendix for an argument about why this is a good approximation.

2. Compute optimal decisions c^n and k^n.
3. Update v^{n+1}.
4. If v^{n+1} is close enough to v^n stop. Otherwise, go to step 1.

A natural initial guess is the value of "staying put", $v^0_{i,j} = \frac{u(z_i + ra_j)}{\rho}$

In what follows I explain in details each step of the algorithm.

Step 1: Upwind scheme

The basic idea of the upwind scheme is to use the forward difference whenever the drift of state variable is positive (ie, positive savings) and the backward difference when the drift is negative. Hence, $v'_{i,j}$ is approximated by

$$v'_{i,j} = v'_{i,j,F}\mathbf{1}_{\{s_{i,j,F}>0\}} + v'_{i,j,B}\mathbf{1}_{\{s_{i,j,B}<0\}} + \bar{v}'_{i,j,F}\mathbf{1}_{\{s_{i,j,F}\le 0 \le s_{i,j,B}\}} \tag{5}$$

Note that since v is concave in a, then $v'_{i,j,F} < v'_{i,j,B}$ and so $s_{i,j,F} < s_{i,j,B}$. The last term in the previous equation is used for some grid points where $s_{i,j,F} \le 0 \le s_{i,j,B}$ and savings are set to zeros and thus $\bar{v}'_{i,j} = u'(z_i + ra_j + (R-r)k_{i,j})$.

Step 2: Compute c and k

From equations (3) and (4), we can write c and k as:

$$c_{i,j} = (v_{i,j})^{-1/\gamma}$$
$$k_{i,j} = \max\left\{ -\frac{v'_{i,j}}{v_{i,j}}\frac{(R-r)}{\sigma^2}, a_j + \phi \right\}$$

Step 3: Update v^{n+1}

To update v^{n+1} use an implicit method[3]. For a given guess v^n, v^{n+1} is implicitly defined by

$$\frac{v^{n+1}_{i,j} - v^n_{i,j}}{\Delta} + \rho v^n_{i,j} = u(c^n_{i,j}) + (v^{n+1}_{i,j,F})' s^+_{i,j,F} + (v^{n+1}_{i,j,B})' s^-_{i,j,B} + \frac{\sigma^2}{2} k^2_{i,j} (v^{n+1}_{i,j})'' + \lambda_i (v_{-i,j} - v_{i,j}) \tag{6}$$

where

$$s^+_{i,j,F} = \max\{z_i + ra_j + (R-r)a_{i,j,F} - c_{i,j,F}, 0\}$$
$$s^+_{i,j,F} = \min\{z_i + ra_j + (R-r)a_{i,j,B} - c_{i,j,B}, 0\}$$

Using the approximation for the first and second derivatives, (6) can be written as

$$\begin{aligned}\frac{v^{n+1}_{i,j} - v^n_{i,j}}{\Delta} + \rho v^n_{i,j} =& u(c^n_{i,j}) + \left(\frac{v^{n+1}_{i,j+1} - v^{n+1}_{i,j}}{\Delta a_{j,+}}\right) s^+_{i,j,F} + \left(\frac{v^{n+1}_{i,j} - v^{n+1}_{i,j-1}}{\Delta a_{j,-}}\right) s^-_{i,j,B} + \\ & \frac{\sigma^2}{2} k^2_{i,j} \frac{\Delta a_{j,-} v_{i,j+1} - (\Delta a_{j,-} + \Delta a_{j,+}) v_{i,j} + \Delta a_{j,+} v_{i,j-1}}{\frac{1}{2}(\Delta a_{j,+} + \Delta a_{j,-})\Delta a_{j,-}\Delta a_{j,+}} + \lambda_i (v^{n+1}_{-i,j} - v^{n+1}_{i,j})\end{aligned}$$

[3] For a discussion between the explicit and implicit method see the paper's appendix. In general, an implicit method is preferable. In particular, the parameter Δ can be arbitrarily large when using this method, while in the explicit method it has to be sufficiently low.

Collecting terms

$$\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta}+\rho v_{i,j}^{n}=u(c_{i,j}^{n})+v_{i,j-1}^{n+1}x_{i,j}+v_{i,j}^{n+1}y_{i,j}+v_{i,j+1}^{n+1}z_{i,j}+v_{-i,j}^{n+1}\lambda_i$$

where

$$\begin{aligned}
x_{i,j}&=-\frac{(s_{i,j,B})^-}{\Delta a_{j,-}}+\frac{\sigma^2}{2}k_{i,j}^2\frac{\Delta a_{j,+}}{\frac{1}{2}(\Delta a_{j,+}+\Delta a_{j,-})\Delta a_{j,+}\Delta a_{j,-}}\\
y_{i,j}&=-\frac{(s_{i,j,F})^+}{\Delta a_{j,+}}+\frac{(s_{i,j,B})^-}{\Delta a_{j,-}}-\frac{\sigma^2}{2}k_{i,j}^2\frac{(\Delta a_{j,-}+\Delta a_{j,+})}{\frac{1}{2}(\Delta a_{j,+}+\Delta a_{j,-})\Delta a_{j,+}\Delta a_{j,-}}-\lambda_i\\
z_{i,j}&=\frac{(s_{i,j,F})^+}{\Delta a_{j,+}}+\frac{\sigma^2}{2}k_{i,j}^2\frac{\Delta a_{j,-}}{\frac{1}{2}(\Delta a_{j,+}+\Delta a_{j,-})\Delta a_{j,+}\Delta a_{j,-}}
\end{aligned}$$

The system of equations of dimension $2\times J$ can be written as

$$\frac{1}{\Delta}(v^{n+1}-v^{n})+\rho v^{n+1}=u^{n}+A^{n}v^{n+1} \tag{7}$$

where

$$A^n=\begin{pmatrix}
y_{1,1} & z_{1,1} & 0 & \dots & 0 & \lambda_1 & 0 & 0 & \dots & 0\\
x_{1,2} & y_{1,2} & z_{1,2} & 0 & \dots & 0 & \lambda_1 & 0 & 0 & \dots\\
0 & x_{1,3} & y_{1,3} & z_{1,3} & 0 & \dots & 0 & \lambda_1 & 0 & 0\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
0 & \ddots & \ddots & x_{1,I} & y_{1,I} & 0 & 0 & 0 & 0 & \lambda_1\\
\lambda_2 & 0 & 0 & 0 & 0 & y_{2,1} & z_{2,1} & 0 & 0 & 0\\
0 & \lambda_2 & 0 & 0 & 0 & x_{2,2} & y_{2,2} & z_{2,2} & 0 & 0\\
0 & 0 & \lambda_2 & 0 & 0 & 0 & x_{2,3} & y_{2,3} & z_{2,3} & 0\\
0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots\\
0 & \cdots & \cdots & 0 & \lambda_2 & 0 & \cdots & 0 & x_{2,I} & y_{2,I}
\end{pmatrix},\qquad u^n=\begin{pmatrix}
u(c_{1,1}^n)\\ \vdots\\ \vdots\\ u(c_{1,J}^n)\\ u(c_{2,1}^n)\\ \vdots\\ \vdots\\ u(c_{2,J}^n)
\end{pmatrix}$$

and it can be solved efficiently. In particular, it can be simplified even further to

$$\mathbf{B}^n v_{n+1}=b^n,\qquad \mathbf{B}^n=\left(\frac{1}{\Delta}+\rho\right)\mathbf{I}-A^n,\qquad b^n=u^n+\frac{1}{\Delta}v^n$$

Boundary conditions

- **Lower bound**

  There is a state constraint $a\geq \underline{a}=-\phi$. The FOC for consumption $u'(c_i(a))=v_i'(a)$ still holds at the borrowing constraint. In order to respect the boundary constraint, the following condition must hold at the boundary $s_i(a)=wz_i+r\underline{a}-c_i(a)\geq 0$. Combining this condition with the FOC, the boundary condition states $v_i'(\underline{a})\geq u'(wz_i+r\underline{a})$.

  In order to enforce this constraint, the following condition is imposed

  $$v_{i,1,B}'=u'(wz_i+ra_1)$$

  This constraint is used whenever $s_{i,1,F}<0$ (see (5)). If $s_{i,1,F}>0$ the forward difference approximation is used and therefore the value function does not hit this constraint.

- **Upper bound**

  In theory, the HJB equation is defined on $(\underline{a}, \infty)$ but in practice it is solved on a bounded interval $(\underline{a}, a_{max})$. A boundary condition must be imposed at a_{max}, a state constraint to guarantee that $a \leq a_{max}$. Furthermore, there has also to be a boundary condition of the term $\frac{\sigma^2}{2} v_i''(a) k(a)^2$.

  Hence, we will impose a boundary condition on $v_i'(a_{max})$, $v_i''(a_{max})$ and $\frac{\sigma^2}{2} v_i''(a) k(a)^2$

  Taking derivatives of the FOC with respect to c, we get $c'(a) = -\frac{1}{\gamma} v_i'(a)^{-\frac{1}{\gamma}-1} v_i''(a)$. In the paper it is shown that the policy function for c is asymptotically linear, hence $c_i'(a) = \bar{c}$ for a sufficiently large. Using these two equation, the following boundary condition is imposed at a_{max}

$$v_i''(a_{max}) = -\gamma v_i'(a_{max})^{1+\frac{1}{\gamma}} \bar{c}$$

  To bound the term $\frac{\sigma^2}{2} v_i''(a) k(a)^2$, we use the FOC with respect to k and the previous condition to obtain

$$k_i(a_{max}) = -\frac{v_i'(a)}{v_i''(a)} \frac{(R-r)}{\sigma^2} = \frac{v_i(a_{max})^{-\frac{1}{\gamma}}}{\bar{c}} \frac{(R-r)}{\gamma\sigma^2}$$

  After a few steps of algebra, it is easy to bound the term we want

$$\frac{\sigma^2}{2} v_i''(a) k(a)^2 = v_i'(a_{max}) \xi$$

  with $\xi = -\frac{v_i'(a_{max})^{-\frac{1}{\gamma}}}{\bar{c}} \frac{(R-r)^2}{2\gamma\sigma^2}$

  Finally, to impose $a \leq a_{max}$, we use $s(a_{max}) \leq 0$ and the previous conditions to get:

$$\begin{aligned} c_i(a_{max}) &\geq wz_i + ra_{max} + (R-r)k(a_{max}) \\ v_i'(a_{max})^{-\frac{1}{\gamma}} = c_i(a_{max}) &\geq wz_i + ra_{max} + v_i'(a_{max})^{-\frac{1}{\gamma}} \frac{(R-r)^2}{\bar{c}\gamma\sigma^2} \\ v_i'(a_{max}) &\geq (wz_i + ra_{max})^{-\gamma} \left(1 - \frac{(R-r)^2}{\bar{c}\gamma\sigma^2}\right)^{\gamma} \end{aligned}$$

Because of the boundary conditions at the top, the associated entries of the matrix A change to[4]

$$\begin{aligned} x_{i,J} &= -\frac{(s_{i,J,B})^-}{\Delta a_{J,-}} - \frac{\xi}{\Delta a_{J,-}} \\ y_{i,J} &= -\frac{(s_{i,J,F})^+}{\Delta a_{J,-}} + \frac{(s_{i,J,B})^-}{\Delta a_{J,-}} + \frac{\xi}{\Delta a_{J,-}} - \lambda_i \\ z_{i,J} &= \frac{(s_{i,J,F})^+}{\Delta a_{J,-}} \end{aligned}$$

2.2 Finite Difference Method for Kolmogorov Forward equation

In the case with uniform grids, to solve the Kolmogorov Forward equation there is only need to solve the system $A^T g = 0$. However, as discussed in the appendix of the paper, with non-uniform grids the matrix A does not preserve mass. Therefore, some adjustments must be made. See the paper's appendix for discussion and derivations.

[4]The point $z_{i,J}$ refers to the non-existent grid point $J+1$. In practice (in the codes), this point is summed to $y_{i,J}$ so then $y_{i,J} = \frac{(s_{i,J,B})^-}{\Delta a_{J,-}} + \frac{\xi}{\Delta a_{J,-}} - \lambda_i$ and $z_{i,J} = 0$

In particular, the matrix A is replaced by $\tilde{A}$

$$\tilde{A} = DAD^{-1}$$

which in this particular case is $D = diag\{\tilde{\Delta}a_1, \tilde{\Delta}a_2, \ldots, \tilde{\Delta}a_J, \tilde{\Delta}a_1, \tilde{\Delta}a_2, \ldots, \tilde{\Delta}a_J\}$

where

$$\tilde{\Delta}a_j = \begin{cases} \frac{1}{2}\Delta a_{j,+} & j = 1 \\ \frac{1}{2}(\Delta a_{j,+} + \Delta a_{j,-}) & j = 2, \ldots, J-1 \\ \frac{1}{2}\Delta a_{j,-} & j = J \end{cases}$$

Note also that the first J terms are the same as the last J terms. This is because in this version we have two types of agents (and of course both use the same grid).

Finally, the matrix A used for the Kolmogorov Forward equation is slightly different from the one used to solve the HJB equation. In particular, the only adjustment made is that the mass associated to the (non-existent) grid point $J+1$ is "reflected" to the point J. Hence

$$\begin{aligned} \tilde{x}_{i,J} &= x_{i,J} = -\frac{(s_{i,J,B})^+}{\Delta a_{j,-}} + \frac{\sigma^2}{2} \frac{k_{i,J}^2}{(\Delta a_{J,-})^2} \\ \tilde{y}_{i,J} &= y_{i,J} + z_{i,J} = \frac{(s_{i,J,B})^-}{\Delta a_{j,-}} - \frac{\sigma^2}{2} \frac{k_{i,J}^2}{(\Delta a_{J,-})^2} - \lambda_i \\ \tilde{z}_{i,J} &= 0 \end{aligned}$$

3 Results

In this section I show the graphs delivered by the Matlab version of the codes developed by Achdou et. al. and I compare them with the ones delivered by my Julia version.

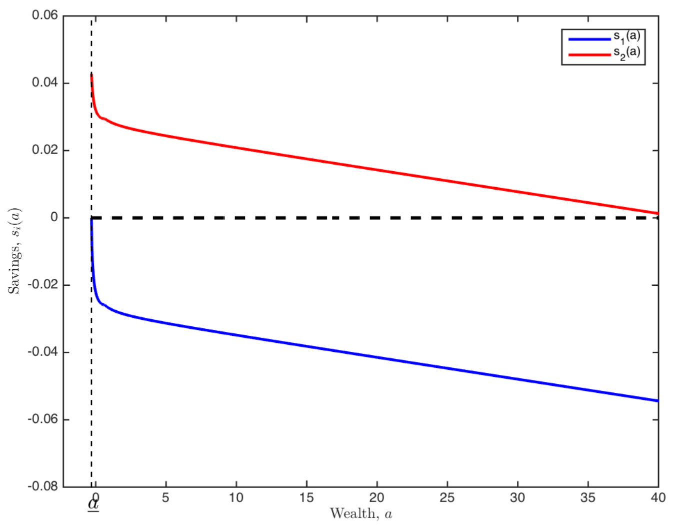

Figure 1: Savings functions from original codes

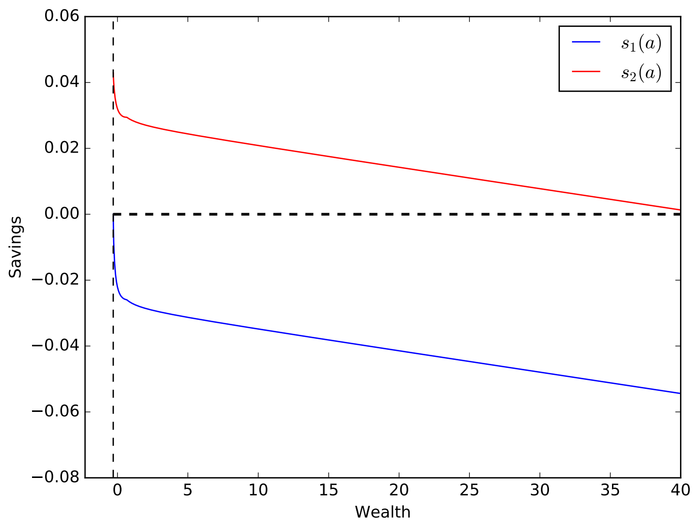

Figure 2: Savings functions from Julia codes

Figure 1 and 2 show the savings functions for each type of agents. I plot the same graph as in the original paper and it is clear that the results are quite similar[5].

[5]Actually, in order to make the results clearer, I added the ticks in the graphs by Achdou et. at. If you run the original Matlab codes, you'll see that the x and y axis do not have any ticks.

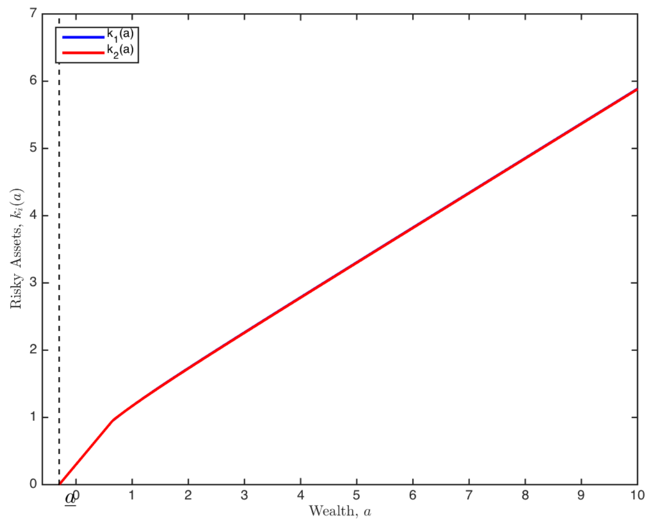

Figure 3: Risky asset from original codes

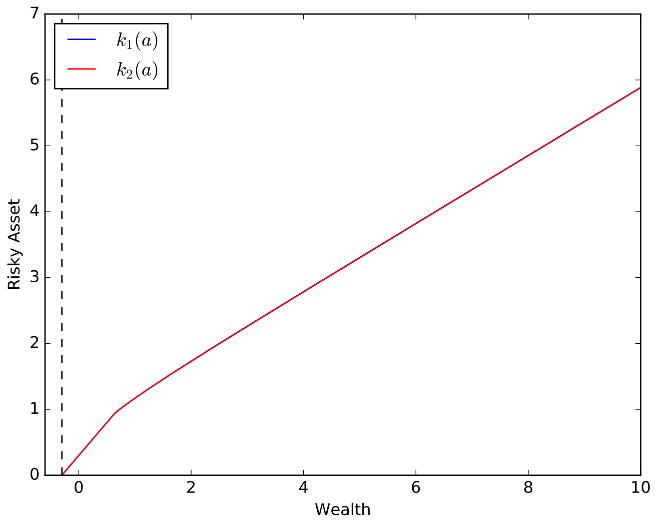

Figure 4: Risky asset from Julia codes

Figure 3 and 4 show the risky asset holding. Again, the graphs look very alike.

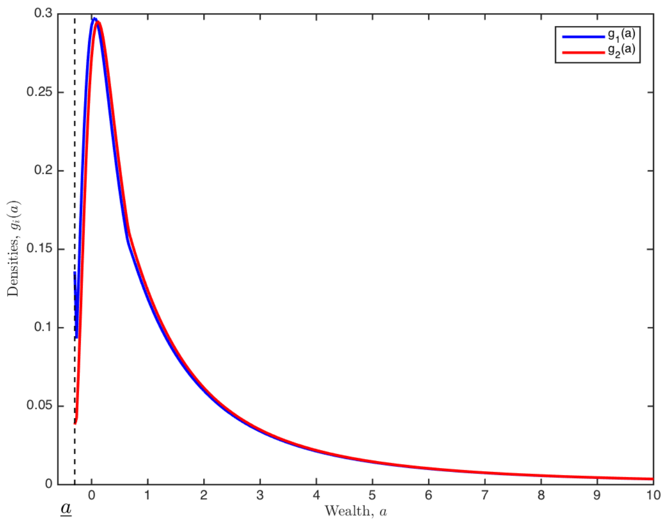

Figure 5: Densities from original codes

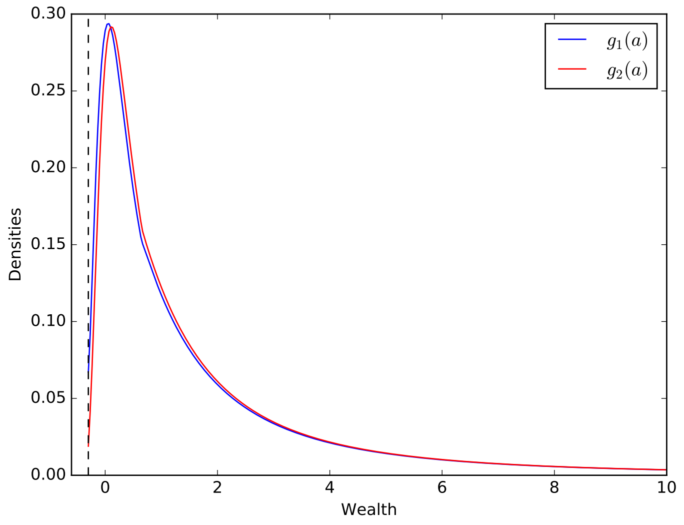

Figure 6: Densities from Julia codes

Finally, Figure 5 and 6 show the distribution of each type of agents. From all the graphs above we can conclude that the replication is quite successful.

4 Comparison

In general, the codes run pretty fast both in Matlab and in Julia. Depending on the version of the Julia codes I use, the performance is different. Furthermore, Julia saves a compilated version of the code/functions being called after being ran once, and hence if the code is ran again it is way faster.

I wrote two version of the codes, one with a procedural type of algorithm that emulates the matlab codes, *Aiyagari_Fat_Tails_Matlab.ipynb*, and another version in which I create a julia type with different functions *Aiyagari_Fat_Tails.ipynb*.

The following table summarizes the performance of the codes in solving the Hamilton-Jacobi-Bellman equation (HJB) and the Kolmogorov Forward equation (KF)

Code	Time (is seconds) HJB	Time (is seconds) FK
Matlab (original codes)	0.570594	0.041999
Julia, no types, 1st run	1.820382	2.354478
Julia, no types, 2st run	0.000003	0.074941
Julia, with types, 1st run	3.216733	1.941737
Julia, with types, 2st run	0.257379	0.040830

Table 1: Performance of the codes

In general, there are no much gains of doing the computation in Julia. The HJB equation is solved faster in both version when there is already a compilated version of the code, but when the code is ran for the first time it actually takes more time to solve it in Julia. Neither version is significantly faster in Julia to solve the KF equation.

Recall that the HJB equation is solved iteratively, while the KF equation is solved in one step. The julia version with no types is super fast when the code is ran more than once, in particular for the Hamilton-Jacobi-Bellman equation. The Julia version with types is also faster than Matlab solving the HJB equation. This shows that Julia is in fact way faster than Matlab when solving loops.

On the other hand, neither version of the codes is faster when solving the Kolmogorov Forward equation. The reason is that there are not much gains in using Julia since this equation is solved only in one step.

The reason why Julia version with no types is slower than the one with types in solving the Kolmogorov Forward equation might be that in the version with Julia types, there are actually more calculations to be done. In fact, the julia type that I define contains two matrices, one to solve the HJB equation and the other to solve the KF equation. There are very similar, they only differ in the upper bound. However, in the iterative algorithm in the version with julia types I update each matrix in each iteration while in the procedural algorithm this is done only once.

5 Extension: Continuum of types

In this section I extend the model to allow for a continuum of productivity types[6]. In particular, productivity follows an Ornstein-Uhlenbeck process

$$d \log z_t = -\theta \log z_t + \sigma d W_t$$

Applying Ito's lemma, the process can be written in levels as

[6]In this section I make a small change in notation. In particular, consider the index i for the asset space, ie $\{a_i\}_{i=1}^{I}$ and j the index for productivity types $\{z\}_{j=1}^{J}$. The same change is made in Achdou et. al. I'm just trying to be consistent with the notation in their paper.

$$dz_t = \left(-\theta z_t \log z_t + \frac{\sigma^2}{2} z_t\right) dt + \sigma z_t dW_t$$

The process gets reflected at a lower and upper bound (z_{min}, z_{max}). The following boundary conditions must be imposed:

$$0 = \partial_z v(a, z_{min}) = \partial_z v(a, z_{max}) \qquad \forall a$$

As discussed in Achdou et. al., it is not necessary to use an upwind scheme for the diffusion process z. Therefore, I use the forward difference to approximate the first derivative of the value function with respect to z. Furthermore, I use an equally spaced grid for the productivity process for simplicity and hence there is no need to distinguish between $\Delta z_{j,+}$ and $\Delta z_{j,-}$. The space between two consecutive grid points in the productivity type space is just Δz.

It is worth noting that this extension considers two exogenous process, one for the risky asset and another one for the idiosyncratic productivity. In Achdou et. at. there is no model with more than one stochastic shock, so I have to combine many of the techniques they show.

The system to be solved is the following:

$$\rho v(a,z) = \max_{c, 0\leq k\leq a+\phi} u(c) + v_a(a,z)(z + ra + (R-r)k - c) + \frac{1}{2}\partial_{aa} v(a,z)\sigma^2 k^2 + \mu(z)\partial_z v(a,z) + \frac{\sigma_z^2}{2}\partial_{zz} v(a,z)$$
$$0 = -\frac{d}{da}[s(a)g(a,z)] + \frac{1}{2}\frac{d^2}{da^2}[\sigma^2 k(a,z)^2 g(a,z)] - \partial_z[\mu(z)g(a,z)] + \frac{1}{2}\partial_{zz}[\sigma_z^2 g(a,z)]$$

The discretized version of the HJB equation is:

$$\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^n = u(c_{i,j}^n) + \partial_{a,F}(v_{i,j,F}^{n+1}) s_{i,j,F}^+ + \partial_{a,B}(v_{i,j,B}^{n+1}) s_{i,j,B}^- + \frac{\sigma^2}{2} k_{i,j}^2 \partial_{aa} v_{i,j}^{n+1} + \partial_z v_{i,j}^{n+1} \mu_j + \frac{\sigma_{z_j}^2}{2}\partial_{zz} v_{i,j}^{n+1} \tag{8}$$

Substituting the approximations of value function with respect to a and z and collecting terms, the previous equation can be written as:

$$\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^n = u(c_{i,j}^n) + v_{i-1,j}^{n+1} x_{i,j} + v_{i,j}^{n+1}(y_{i,j} + \upsilon_j) + v_{i+1,j}^{n+1} z_{i,j} + v_{i,j-1}^{n+1}\chi_j + v_{i,j+1}^{n+1}\zeta_j$$

where $x_{i,j}, y_{i,j}$ and $z_{i,j}$ are defined in the same way as in the previous section[7] and χ_j, υ_j and ζ_j are defined as follows:

$$\chi_j = \frac{\sigma_j^2}{2(\Delta z)^2}$$
$$\upsilon_j = -\frac{\mu_j}{\Delta z} - \frac{\sigma_j^2}{(\Delta z)^2}$$
$$\zeta_j = \frac{\mu_j}{\Delta z} + \frac{\sigma_j^2}{2(\Delta z)^2}$$

To impose the boundary conditions at the z–boundaries, I replace the equations at those nodes with the

[7] To be precise, $x_{i,j}$ and $z_{i,j}$ are exactly the same and $y_{i,j}$ is the same but without the $-\lambda_i$ term.

following:

$$\frac{v_{i,1}^{n+1}-v_{i,1}^{n}}{\Delta}+\rho v_{i,1}^{n}=u(c_{i,1}^{n})+v_{i,1}^{n+1}x_{i,1}+v_{i,1}^{n+1}(y_{i,1}+v_1+\chi_1)+v_{i+1,1}^{n+1}z_{i,1}++v_{i,2}^{n+1}\zeta_1$$
$$\frac{v_{i,J}^{n+1}-v_{i,J}^{n}}{\Delta}+\rho v_{i,J}^{n}=u(c_{i,J}^{n})+v_{i-1,J}^{n+1}x_{i,J}+v_{i,J}^{n+1}(y_{i,J}+v_J+\zeta_J)+v_{i+1,J}^{n+1}z_{i,J}+v_{i,J-1}^{n+1}\chi_J$$

Hence, to solve the HJB equation I have to solve the following system of $I\times J$ equations:

$$\frac{1}{\Delta}(v^{n+1}-v^{n})+\rho v^{n+1}=u^{n}+A^{n}v^{n+1} \tag{9}$$

where u^n is defined as in the previous case and $A^n=B^n+C$ where

$$B^n=\begin{pmatrix}
y_{1,1} & z_{1,1} & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0\\
x_{1,2} & y_{1,2} & z_{1,2} & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & x_{1,I} & y_{1,I} & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & 0 & y_{2,1} & z_{2,1} & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & 0 & x_{2,2} & y_{2,2} & z_{2,2} & 0 & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 & x_{2,I} & y_{2,I} & 0 & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & y_{1,J} & z_{1,J} & 0 & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & x_{1,J} & y_{1,J} & z_{1,J} & 0\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 & x_{I,J} & y_{I,J}
\end{pmatrix}$$

and

$$C=\begin{pmatrix}
v_1+\chi_1 & 0 & \dots & \dots & 0 & \zeta_1 & 0 & \dots & \dots & \dots & \dots & \dots & 0\\
0 & v_1+\chi_1 & 0 & \ddots & \ddots & 0 & \zeta_1 & 0 & \ddots & \ddots & \ddots & \ddots & 0\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
0 & \ddots & 0 & v_1+\chi_1 & 0 & \ddots & \ddots & 0 & \zeta_1 & 0 & \ddots & \ddots & \vdots\\
\chi_2 & 0 & \ddots & 0 & v_2 & 0 & \ddots & \ddots & 0 & \zeta_2 & 0 & \ddots & \vdots\\
0 & \chi_2 & 0 & \ddots & 0 & v_2 & 0 & \ddots & \ddots & 0 & \zeta_2 & 0 & \vdots\\
\vdots & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & \ddots & \vdots\\
\vdots & \ddots & 0 & \chi_2 & 0 & \ddots & 0 & v_2 & 0 & \ddots & \ddots & 0 & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & 0 & \chi_J & 0 & \ddots & \ddots & v_J+\zeta_J & 0 & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 & \chi_J & 0 & \ddots & \ddots & v_J+\zeta_J & 0 & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 & \chi_J & 0 & \dots & 0 & v_J+\zeta_J
\end{pmatrix}$$

There is a small difference in the way this type of model is solved in Achdou et. al. and the way I'm solving it this time. In particular, they use two versions of the matrix to solve the HJB equation and the KF equation, they only differ in the boundary condition at the top. This is done only in fat-tailed kind of models, and in the standard Aiyagari model both matrices are exactly the same.

After solving the the FK with the same matrix of the HJB, I realized that the results do not change significantly and hence I used the same matrix this time. When I solve the Fat Tailed model with only two types, I used the two different matrices because they did it that way and I wanted to replicate their results, but there are no quantitatively important changes when this part is not done. After all, the difference is only in the upper bound of the grid[8].

The following plot shows the distribution over the states wealth a and productivity z

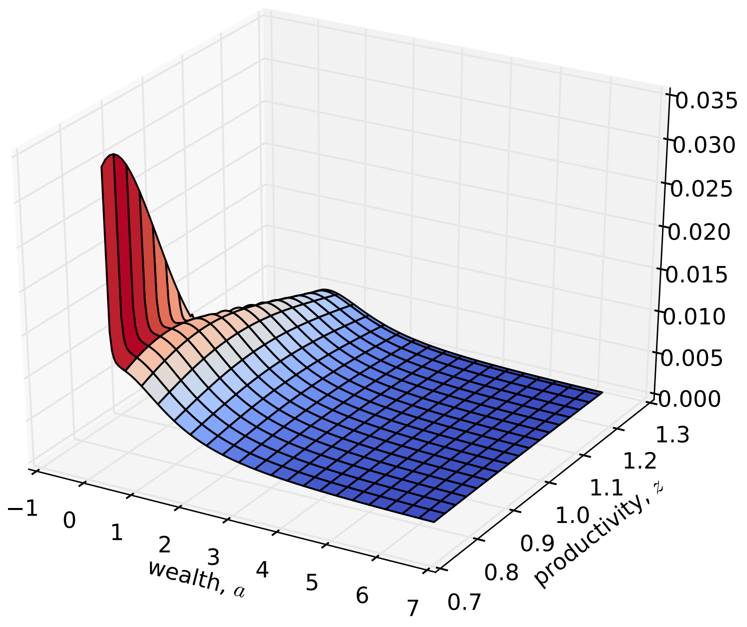

Figure 7: Distribution over (a, z)

Finally, in order to see this graph in a different way, I create a movie which plots the wealth distribution for a given z and then z increases as the movie is played. In order to see it, play *plot_wealth.mp4*

[8]In the two types of agents model they solve, this means that in a total of 2000 points, only two points differ.