Online Appendix: Numerical Methods for "Income and Wealth Distribution in Macroeconomics: A Continuous-Time Approach"[1]

Yves Achdou, Jiequn Han, Jean-Michel Lasry, Pierre-Louis-Lions, Benjamin Moll

This is an online Appendix to Achdou et al. (2020). It is concerned with the numerical solution using a finite difference method of the continuous time heterogeneous agent models presented in that paper. Also see the discussion in Section 4 of the paper, particulary the material on the conditions of Barles and Souganidis (1991).

Huggett Economy

We start by solving a continuous time version of Huggett (1993) which is arguably the simplest heterogeneous agent model that captures many of the features of richer models. The economy can be represented by the following system of equations which we aim to solve numerically:

$$\rho v_1(a) = \max_c \; u(c) + v_1'(a)(z_1 + ra - c) + \lambda_1(v_2(a) - v_1(a)) \tag{1}$$

$$\rho v_2(a) = \max_c \; u(c) + v_2'(a)(z_2 + ra - c) + \lambda_2(v_1(a) - v_2(a)) \tag{2}$$

$$0 = -\frac{d}{da}[s_1(a)g_1(a)] - \lambda_1 g_1(a) + \lambda_2 g_2(a) \tag{3}$$

$$0 = -\frac{d}{da}[s_2(a)g_2(a)] - \lambda_2 g_2(a) + \lambda_1 g_1(a) \tag{4}$$

$$1 = \int_{\underline{a}}^{\infty} g_1(a)da + \int_{\underline{a}}^{\infty} g_2(a)da \tag{5}$$

$$0 = \int_{\underline{a}}^{\infty} a g_1(a)da + \int_{\underline{a}}^{\infty} a g_2(a)da \equiv S(r) \tag{6}$$

where $z_1 < z_2$ and $s_j(a) = z_j + ra - c_j(a)$ and $c_j(a) = (u')^{-1}(v_j(a))$ are optimal savings and consumption. Finally, there is a state constraint $a \geq \underline{a}$. The first order condition $u'(c_j(\underline{a})) = v_j'(\underline{a})$ still holds at the borrowing constraint. However, in order to respect the constraint we need $s_j(\underline{a}) = z_j + ra - c_j(\underline{a}) \geq 0$. Combining this with the FOC, the state constraint motivates a boundary condition

$$v_j'(\underline{a}) \geq u'(z_j + r\underline{a}), \quad j = 1, 2 \tag{7}$$

[1]We thank SeHyoun Ahn for fantastic research assistance. Matthieu Gomez provided extremely useful comments and in particular showed us how to clearly think about non-uniform grids in Appendix Section 7.

We use a finite difference method. A useful reference is Candler (1999). We first explain how to solve the Hamilton-Jacobi-Bellman (HJB) equation (1) and (2), and then turn to the Kolmogorov Forward (Fokker-Planck) equation (3) and (4).

Section 4 explains how the setup and the solution method can be generalized to an environment where productivity z is continuous and follows a diffusion rather than a two-state Poisson process. Finally, a useful "warm-up problem" is to solve the HJB equation with no uncertainty, $\lambda_j = 0$. See `http://www.princeton.edu/~moll/HACTproject/HACT_Additional_Codes.pdf`. All algorithms are available as Matlab codes from `https://benjaminmoll.com/codes/`. We are especially indebted to SeHyoun Ahn for showing us how to use matlab's sparse matrix routines to increase speed by an order of magnitude.

1 HJB Equation

We use a finite difference method and approximate the functions (v_1, v_2) at I discrete points in the space dimension, $a_i, i = 1, ..., I$. We use equispaced grids, denote by Δa the distance between grid points, and use the short-hand notation $v_{i,j} \equiv v_j(a_i)$ and so on. The derivative $v'_{i,j} = v'_j(a_i)$ is approximated with either a forward or a backward difference approximation

$$\begin{aligned} v'_j(a_i) &\approx \frac{v_{i+1,j} - v_{i,j}}{\Delta a} \equiv v'_{i,j,F} \\ v'_j(a_i) &\approx \frac{v_{i,j} - v_{i-1,j}}{\Delta a} \equiv v'_{i,j,B} \end{aligned} \tag{8}$$

The finite difference approximation to (1) and (2) is

$$\begin{aligned} \rho v_{i,j} &= u(c_{i,j}) + v'_{i,j}(z_j + ra_i - c_{i,j}) + \lambda_j(v_{i,-j} - v_{i,j}), \quad j = 1, 2 \\ c_{i,j} &= (u')^{-1}(v'_{i,j}) \end{aligned} \tag{9}$$

where $v'_{i,j}$ is either the forward or the backward difference approximation. There are two complications. The first question is when to use a forward and when a backward difference approximation. It turns out that this is actually quite important for the stability properties of the scheme. The second is that the HJB equations are highly non-linear, and therefore so is the system of equations (9). It therefore has to be solved using an iterative scheme (rather than simply inverting a matrix).

There are two options that differ in how the value function is updated: a so-called "explicit" method and an "implicit" method. As a general rule, the implicit method is the preferred approach because it is both more efficient and more stable/reliable. However, the explicit method is easier to explain so we turn to it first.

1.1 Explicit Method

See matlab program `HJB_stateconstraint_explicit.m`. One starts with an initial guess $v_j^0 = (v_{1,j}^0, ..., v_{I,j}^0)$, $j = 1, 2$ and then updates $v_j^n, n = 1, ...$ according to

$$\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^n = u(c_{i,j}^n) + (v_{i,j}^n)'(z_j + ra_i - c_{i,j}^n) + \lambda_j(v_{i,-j}^n - v_{i,j}^n) \tag{10}$$

where $c_{i,j}^n = (u')^{-1}[(v_{i,j}^n)']$. The parameter Δ is the step size of the explicit method. It can be shown that the explicit method only converges if Δ is not too large (it has to satisfy the so-called "CFL condition", see e.g. p.181 in Candler (1999)). An advantage of implicit methods discussed in section 1.2 is that the step size Δ can be arbitrarily large.

Upwind Scheme. As already mentioned, it is important whether and when a forward or a backward difference approximation is used. The correct way of doing this is to use a so-called "upwind scheme." The rough idea is to use a forward difference approximation whenever the drift of the state variable (here, savings $s_{i,j}^n = z_j + ra_i - c_{i,j}^n$) is positive and to use a backwards difference whenever it is negative.[2] In practice, this is done as follows: first compute savings according to both the backwards and forward difference approximations $v'_{i,j,F}$ and $v'_{i,j,B}$

$$s_{i,j,F} = z_j + ra_i - (u')^{-1}(v'_{i,j,F}), \quad s_{i,j,B} = z_j + ra_i - (u')^{-1}(v'_{i,j,B})$$

where we suppress n superscripts for notational simplicity. Then use the following approximation for $v'_{i,j}$:

$$v'_{i,j} = v'_{i,j,F}\mathbf{1}_{\{s_{i,j,F}>0\}} + v'_{i,j,B}\mathbf{1}_{\{s_{i,j,B}<0\}} + \bar{v}'_{i,j}\mathbf{1}_{\{s_{i,j,F}\leq 0\leq s_{i,j,B}\}} \tag{11}$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. The meaning of the last term is as follows. First note that since v is concave in a, we have $v'_{i,j,F} < v'_{i,j,B}$ and so $s_{i,j,F} < s_{i,j,B}$. Therefore, for some grid points i, $s_{i,j,F} \leq 0 \leq s_{i,j,B}$. At these grid points, we set savings equal to zero and hence set the derivative of the value function equal to $\bar{v}'_{i,j} = u'(z_j + ra_i)$. The fact that v is concave also means that we do not have to worry about the case where both $s_{i,j,F} > 0$ *and* $s_{i,j,B} < 0$: because concavity implies $s_{i,j,F} < s_{i,j,B}$, this cannot happen.

Under some circumstances (and in more general applications e.g. problems with non-convexities) it can happen that the value function is not concave. The question then arises what to do when both $s_{i,j,F} > 0$ *and* $s_{i,j,B} < 0$. The following upwind scheme works well in practice in this case. Define an indicator for the problematic case in which both $s_{i,j,F} > 0$

[2]Note that treatments of finite difference methods concerned with solving PDEs forward in time usually define an "upwind scheme" in the opposite way (forward difference when the drift is *negative*, backwards difference whenever it is *positive*). See e.g. `https://en.wikipedia.org/wiki/Upwind_scheme`. The difference is that solving HJB equations (even stationary ones) amounts to solving PDEs backwards in time given a terminal condition (rather than forward in time given an initial condition). The two seemingly different definitions of the term "upwind scheme" are the same when taking this difference into account.

and $s_{i,j,B} < 0$, $\mathbf{1}_{i,j}^{both} := \mathbf{1}_{\{s_{i,j,B} \leq 0 \leq s_{i,j,F}\}}$, and an indicator for the unproblematic case $\mathbf{1}_{i,j}^{unique} = \mathbf{1}_{\{s_{i,j,F}<0 \text{ and } s_{i,j,B}>0\}} + \mathbf{1}_{\{s_{i,j,F}<0 \text{ and } s_{i,j,B}<0\}}$. Next, define the forward and backward Hamiltonians $H_{i,j,F} := u(c_{i,j,F}) + v'_{i,j,F} s_{i,j,F}$ and similarly for $H_{i,j,B}$. Finally, use the upwind scheme

$$\begin{aligned} v'_{i,j} = & v'_{i,j,F} \left(\mathbf{1}_{\{s_{i,j,F}>0\}} \mathbf{1}_{i,j}^{unique} + \mathbf{1}_{\{H_{i,j,F} \geq H_{i,j,B}\}} \mathbf{1}_{i,j}^{both}\right) \\ & + v'_{i,j,B} \left(\mathbf{1}_{\{s_{i,j,B}<0\}} \mathbf{1}_{i,j}^{unique} + \mathbf{1}_{\{H_{i,j,F} < H_{i,j,B}\}} \mathbf{1}_{i,j}^{both}\right) \\ & + \bar{v}'_{i,j} \mathbf{1}_{\{s_{i,j,F} \leq 0 \leq s_{i,j,B}\}} \end{aligned} \tag{12}$$

Intuitively, in the problematic case when both $s_{i,j,F} > 0$ *and* $s_{i,j,B} < 0$, this upwind scheme uses as the "tie breaker" the rule to use the derivative in the direction in which the gain according to the Hamiltonians $H_{i,j,B}$ and $H_{i,j,F}$ is larger.

State Constraint. The state constraint (7) is enforced by setting

$$v'_{1,j,B} = u'(z_j + ra_1), \quad j = 1, 2$$

From (11) it can then be seen that the state constraint is imposed whenever the forward difference approximation would result in negative savings $s_{1,j,F} \leq 0$. Otherwise if $s_{1,j,F} > 0$ the forward difference approximation $v'_{1,j,F}$ is used at the boundary, implying that the value function "never sees the state constraint." At the upper end of the state space, the upwind method should make sure that a backward-difference approximation is used. In practice, it can sometimes help stability of the algorithm to simply impose a state constraint $a \leq a_{\max}$ where $a_{\max}$ is the upper end of the bounded state space used for computations (this can be achieved by setting $v'_{I,j,F} = u'(z_j + ra_I)$).

Initial Guess. A natural initial guess is the value function of "staying put"

$$v_{i,j}^0 = \frac{u(z_j + ra_i)}{\rho}.$$

Summary of Algorithm. Summarizing, the algorithm for finding a solution to the HJB equation (1) and (2) is as follows. Guess $v_{i,j}^0, i = 1, ..., I, j = 1, 2$ and for $n = 0, 1, 2, ...$ follow

1. Compute $(v_{i,j}^n)'$ using (8) and (11).
2. Compute c^n from $c_{i,j}^n = (u')^{-1}[(v_{i,j}^n)']$
3. Find v^{n+1} from (10).
4. If v^{n+1} is close enough to v^n: stop. Otherwise, go to step 1.

One can show that, for a small enough Δ, this algorithm satisfies the three conditions of Barles and Souganidis (1991) (monotonicity, consistency, stability). See the discussion in Section 4 of Achdou et al. (2020).

1.2 Implicit Method

See matlab program `HJB_stateconstraint_implicit.m`. Relative to the explicit scheme in (10), an implicit differs in how v^n is updated. In particular, v^{n+1} is now implicitly defined by the equation

$$\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^{n+1} = u(c_{i,j}^n) + (v_{i,j}^{n+1})'(z_j + ra_i - c_{i,j}^n) + \lambda_j(v_{i,-j}^{n+1} - v_{i,j}^{n+1})$$

Note the $n+1$ superscripts on the right-hand side of the equation.[3] The main advantage of the implicit scheme is that the step size Δ can be arbitrarily large.

Upwind Scheme. As was the case for the explicit method, we need to use an "upwind scheme." As above, the idea is still to use the forward difference approximation whenever the drift of the state variable is positive and the backward difference approximation whenever it is negative. We use the following finite difference approximation to (1) and (2).

$$\begin{aligned}\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^{n+1} =& u(c_{i,j}^n) + (v_{i,j,F}^{n+1})'[z_j + ra_i - c_{i,j,F}^n]^+ + (v_{i,j,B}^{n+1})'[z_j + ra_i - c_{i,j,B}^n]^- \\ &+ \lambda_j[v_{i,-j}^{n+1} - v_{i,j}^{n+1}]\end{aligned} \tag{13}$$

where $c_{i,j}^n = (u')^{-1}[(v_{i,j}^n)']$ and $(v_{i,j}^n)'$ is given by (11).[4] For any number x, the notation x^+ means "the positive part of x", i.e. $x^+ = \max\{x, 0\}$ and analogously $x^- = \min\{x, 0\}$, i.e. $[z_j + ra_i - c_{i,j,F}^n]^+ = \max\{z_j + ra_i - c_{i,j,F}^n, 0\}$ and $[z_j + ra_i - c_{i,j,B}^n]^- = \min\{z_j + ra_i - c_{i,j,B}^n, 0\}$.

Equation (13) constitutes a system of $2 \times I$ linear equations, and it can be written in matrix notation using the following steps. Substituting the definition of the derivatives (8), and defining $s_{i,j,F}^n = z_j + ra_i - c_{i,j,F}^n$ and similarly for $s_{i,j,B}^n$, (13) is

$$\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^{n+1} = u(c_{i,j}^n) + \frac{v_{i+1,j}^{n+1} - v_{i,j}^{n+1}}{\Delta a}(s_{i,j,F}^n)^+ + \frac{v_{i,j}^{n+1} - v_{i-1,j}^{n+1}}{\Delta a}(s_{i,j,B}^n)^- + \lambda_j[v_{i,-j}^{n+1} - v_{i,j}^{n+1}]$$

[3] Strictly speaking, the present method is a "semi-implicit method." A fully implicit method would feature $n+1$ superscripts also on $c_{i,j}$. Such a fully implicit scheme can be solved using a Newton method, which ends up looking very similar to the iterative scheme outlined here.

[4] As noted in the discussion of the explicit scheme in Section 1.1, this works well when the value function is concave. When the value function is not concave, we can again use the scheme (12) instead of (11).

Collecting terms with the same subscripts on the right-hand side:

$$\begin{aligned}
&\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta}+\rho v_{i,j}^{n+1}=u(c_{i,j}^{n})+v_{i-1,j}^{n+1}x_{i,j}+v_{i,j}^{n+1}y_{i,j}+v_{i+1,j}^{n+1}z_{i,j}+v_{i,-j}^{n+1}\lambda_j \quad \text{where}\\
&x_{i,j}=-\frac{(s_{i,j,B}^{n})^{-}}{\Delta a},\\
&y_{i,j}=-\frac{(s_{i,j,F}^{n})^{+}}{\Delta a}+\frac{(s_{i,j,B}^{n})^{-}}{\Delta a}-\lambda_j,\\
&z_{i,j}=\frac{(s_{i,j,F}^{n})^{+}}{\Delta a}
\end{aligned} \tag{14}$$

Note that importantly $x_{1,j}=z_{I,j}=0, j=1,2$ so $v_{0,j}^{n+1}$ and $v_{I+1,j}^{n+1}$ are never used. Equation (14) is a system of $2\times I$ linear equations which can be written in matrix notation as:

$$\frac{1}{\Delta}(v^{n+1}-v^{n})+\rho v^{n+1}=u^{n}+\mathbf{A}^{n}v^{n+1} \tag{15}$$

where

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

This system can in turn be written as

$$\mathbf{B}^{n}v^{n+1}=b^{n},\qquad \mathbf{B}^{n}=\left(\frac{1}{\Delta}+\rho\right)\mathbf{I}-\mathbf{A}^{n},\quad b^{n}=u^{n}+\frac{1}{\Delta}v^{n} \tag{16}$$

Equation (16) can be solved very efficiently in matlab using sparse matrix routines. To check that one has constructed the intensity matrix $\mathbf{A}$ correctly, the matlab function `spy` is a convenient tool. Figure 1 plots an example of `spy`'s output with 30×2 grid points (we usually use many more).

Finally, it is instructive to consider the case with an infinite updating step size $\frac{1}{\Delta}=0$ and to write the linear system (15) as

$$\rho v^{n+1}=u^{n}+\mathbf{A}^{n}v^{n+1} \tag{17}$$

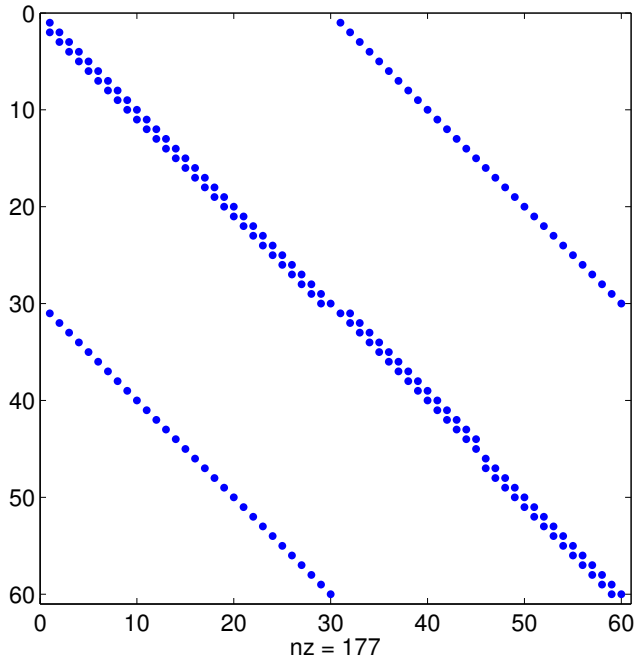


Figure 1: Visualization of the matrix $\mathbf{A}$ using Matlab's spy function

It can be seen that (17) is just a way of writing the discretized version of the HJB equations (1) and (2) in matrix form. In particular the matrix $\mathbf{A}^n$ encodes the evolution of the stochastic process (a_t, z_t). The finite difference method basically approximates this process with a discrete Poisson process with a transition matrix $\mathbf{A}^n$ summarizing the corresponding Poisson intensities. Note that $\mathbf{A}^n$ satisfies all the properties a Poisson transition matrix needs to satisfy. In particular, all rows sum to zero, diagonal elements are non-positive and off-diagonal elements are non-negative (all entries in a row being zero would mean that the state remains fixed over time). We will therefore sometimes refer to $\mathbf{A}^n$ as "Poisson transition matrix" or "intensity matrix." All this will be useful in section 2 below when we solve the Kolmogorov Forward equation (3) and (4).

Summary of Algorithm. The algorithm is exactly the same as above, except that the updating step uses (13) or equivalently (16). Guess $v^0_{i,j}, i = 1, ..., I, j = 1, 2$ and for $n = 0, 1, 2, ...$ follow

1. Compute $(v^n_{i,j})'$ using (8) and (11).
2. Compute c^n from $c^n_{i,j} = (u')^{-1}[(v^n_{i,j})']$
3. Find v^{n+1} from (16).
4. If v^{n+1} is close enough to v^n: stop. Otherwise, go to step 1.

One can show that this algorithm satisfies the three conditions of Barles and Souganidis (1991) (monotonicity, consistency, stability) *regardless of the size of* Δ. See the discussion in Section 4 of Achdou et al. (2020). This is the big advantage of an implicit scheme over an explicit scheme.

2 Kolmogorov Forward (Fokker-Planck) Equation

See matlab code `huggett_partialeq.m` We now turn to the solution of (3) and (4), which also have to satisfy (5). The rough idea is to discretize these as

$$0 = -[s_{i,j}g_{i,j}]' - \lambda_j g_{i,j} + \lambda_{-j} g_{i,-j} \tag{18}$$

$$1 = \sum_{i=1}^{I} g_{i,1}\Delta a + \sum_{i=1}^{I} g_{i,2}\Delta a \tag{19}$$

(Instead of (19), one could also use a slightly more accurate trapezoidal rule, but results are virtually identical given the fine grid size.) Because (3) and (4) are linear in g_1 and g_2 so is the finite difference approximation. As a result, no iterative procedure like the one for the HJB equation is needed and the equation can be solved in one step.

Upwind Scheme. There is again a question when to use a forward and a backward approximation for the derivative $[s_{i,j}g_{i,j}]'$. It turns out that the most convenient/correct approximation is as follows:

$$-\frac{(s^n_{i,j,F})^+ g_{i,j} - g_{i-1,j}(s^n_{i-1,j,F})^+}{\Delta a} - \frac{g_{i+1,j}(s^n_{i+1,j,B})^- - g_{i,j}(s^n_{i,j,B})^-}{\Delta a} - g_{i,j}\lambda_j + g_{i,-j}\lambda_{-j} = 0$$

Note that because $g_{0,j}$ and $g_{I+1,j}$ are outside the state space, the density at these points is zero and so $(s_{0,j,F})^+$ and $(s_{I+1,j,B})^-$ are never used. The reason why the approximation above is desirable is as follows. Collecting terms, we can write

$$\begin{aligned}
&g_{i-1,j}z_{i-1,j} + g_{i,j}y_{i,j} + g_{i+1,j}x_{i+1,j} + g_{i,-j}\lambda_{-j} = 0 \\
&x_{i+1,j} = -\frac{(s^n_{i,j+1,B})^-}{\Delta a} \\
&y_{i,j} = -\frac{(s^n_{i,j,F})^+}{\Delta a} + \frac{(s^n_{i,j,B})^-}{\Delta a} - \lambda_j \\
&z_{i-1,j} = \frac{(s^n_{i,j-1,F})^+}{\Delta a}
\end{aligned}$$

The reason this is the preferred approximation is that it can be written in matrix form in a way that is closely related to the approximation used for the HJB equation

$$\mathbf{A}^{\mathrm{T}} g = 0 \tag{20}$$

where $\mathbf{A}^{\mathrm{T}}$ is the transpose of the intensity matrix $\mathbf{A}$ from the HJB equation (17) ($\mathbf{A}^n$ from the final HJB iteration). This makes sense: the operation is exactly the same as that used for finding the stationary distribution of a discrete Poisson process (continuous-time Markov chain). The matrix $\mathbf{A}$ captures the evolution of the stochastic process and to find the stationary distribution, one solves the eigenvalue problem $\mathbf{A}^{\mathrm{T}} g = 0$. There is therefore a deep reason why one wants to use the transpose of the intensity matrix $\mathbf{A}$. For interested readers, this can be made more precise using some tools from the theory of differential operators: one can write the HJB equations (1) and (2) in terms of a differential operator $\mathcal{A}$, the so-called "infinitesimal generator" of the process. Similarly, the Kolmogorov Forward equations (3) and (4) can be written in terms of an operator $\mathcal{A}^*$. An operator is the infinite-dimensional analogue of a matrix. And the analogue of a matrix transpose is the so-called "adjoint" of an operator. It turns out that the operator in the Kolmogorov Forward equation $\mathcal{A}^*$ is the "adjoint" of the operator in the HJB equation $\mathcal{A}$. Putting things together, $\mathbf{A}$ is simply the discretized infinitesimal generator whereas $\mathbf{A}^{\mathrm{T}}$ is the discretized version of its adjoint, the "Kolmogorov Forward operator."

Besides making sense, this approximation is also convenient: once one has constructed the matrix $\mathbf{A}$ for solving the HJB equation using an implicit method, almost no extra work is needed.

To solve the eigenvalue problem (20) while imposing (19), the simplest procedure is as follows. Fix $g_{i,j} = 0.1$ (any other number will do as well) for an arbitrary (i, j), to then solve the system for some $\tilde{g}$ and then to renormalize $g_{i,j} = \tilde{g}_{i,j}/(\sum_{i=1}^{I} \tilde{g}_{i,1}\Delta a + \sum_{i=1}^{I} \tilde{g}_{i,2}\Delta a)$. Fixing $g_{i,j} = 0.1$ is achieved by replacing the corresponding entry of the zero vector in (20) by 0.1, and the corresponding row of $\mathbf{A}^{\mathrm{T}}$ by a row of zeros everywhere except for one on the diagonal. Without this "dirty fix," the matrix $\mathbf{A}^{\mathrm{T}}$ is singular and so cannot be inverted.

Alternatively, the eigenvalue problem (20) can be solved using a pre-built routine for numerical eigenvalue problems. For example, MATLAB's `eigs` function is well suited.[5]

As shown in Achdou et al. (2020), the wealth distribution of the low-income type g_1 features a Dirac mass at the borrowing constraint $\underline{a}$ (the left boundary of the state space). When discretizing the distribution using a finite difference method, there is technically a Dirac mass at every point in the state space. The algorithm therefore simply ignores the Dirac mass at the boundary and treats it like any other point. Nevertheless, as we will see below, the numerical solution has a clearly visible spike at $\underline{a}$.

[5]The particular command `[g,val]=eigs(A',1,'lr')` seems to work well. It returns the eigenvalue `v` and eigenvector `val` of $\mathbf{A}^{\mathrm{T}}$ with the largest real part (which is `v`$= 0$). Finally, for very large problems (e.g. with three or four state variables) iterative methods such as `bicgstab` may be preferable.

2.1 Results

Figure 2 (a) plots the functions $s_1(a)$ (solid blue line), $s_2(a)$ (solid green line). Note that $s_1'(a) \to -\infty$ as $a \to \underline{a}$ as expected. Figure 2 (b) plots the associated densities $g_1(a)$ and $g_2(a)$.

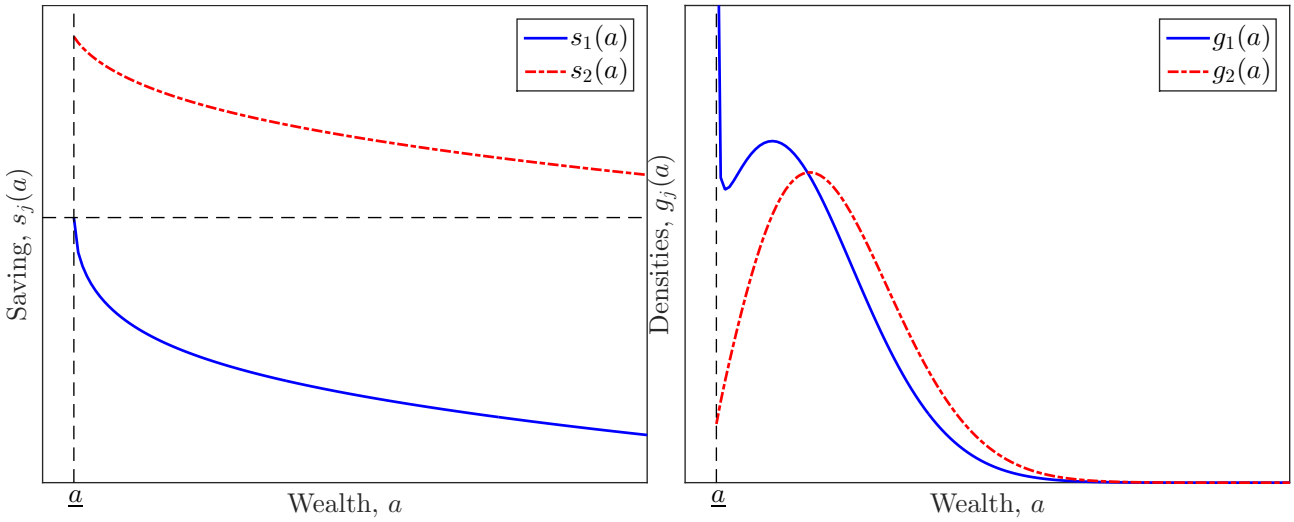


Figure 2: Savings Policy Function and Implied Wealth Distribution

3 Equilibrium

3.1 Asset Supply

See matlab code `huggett_asset_supply.m`. After having solved (1) to (5), the asset supply function $S(r)$ defined in (6) can be easily computed. We approximate it as

$$S(r) \approx \sum_{i=1}^{I} a_i g_{i,1} \Delta a + \sum_{i=1}^{I} a_i g_{i,2} \Delta a$$

Figure 3 plots asset supply as a function of the interest rate. It looks as expected: in particular, supply is bounded below by the borrowing constraint and $S(r) \to \infty$ as $r \to \rho$.

3.2 Finding the Equilibrium Interest Rate

See matlab code `huggett_equilibrium_iterate.m`. The equilibrium interest rate can easily be found using a bisection method: the obvious idea is to increase r whenever $S(r) < 0$ and decrease r whenever $S(r) > 0$. See the code for details.

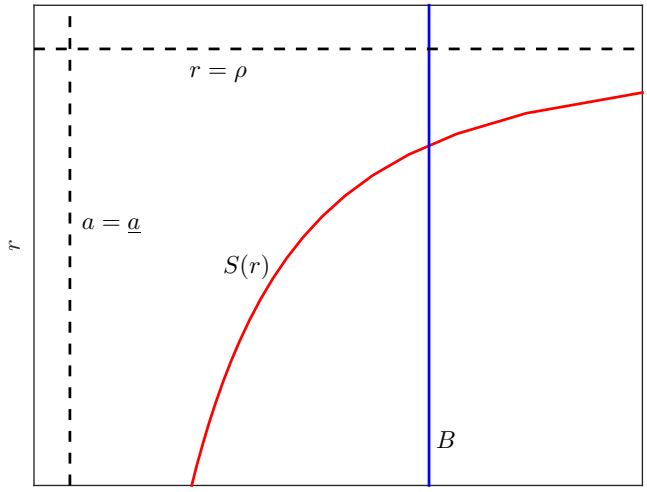


Figure 3: Asset Supply $S(r)$

4 Transition Dynamics and "MIT Shocks"

See matlab codes `huggett_transition.m` which needs input from `huggett_initial.m` and `huggett_terminal.m` (the initial and terminal conditions). Besides solving transition dynamics from an arbitrary initial condition, the same algorithm can be used to study the economy's impulse response after an "MIT shock," i.e. an unanticipated (zero probability) shock followed by a deterministic transition. We provide an example in Section 6.3.

The system to be solved is:

$$0 = S(r(t)) = \int_{\underline{a}}^{\infty} a g_1(a,t) da + \int_{\underline{a}}^{\infty} a g_2(a,t) da \tag{21}$$

$$\rho v_j(a,t) = \max_c \; u(c) + \partial_a v_j(a,t)[z_j + r(t)a - c] + \lambda_j[v_{-j}(a,t) - v_j(a,t)] + \partial_t v_j(a,t), \tag{22}$$

$$\partial_t g_j(a,t) = -\partial_a[s_j(a,t) g_j(a,t)] - \lambda_j g_j(a,t) + \lambda_{-j} g_{-j}(a,t) \tag{23}$$

$$s_j(a,t) = z_j + r(t)a - c_j(a,t), \quad c_j(a,t) = (u')^{-1}(\partial_a v_j(a,t)) \tag{24}$$

The bond market clearing condition can be written as

$$0 = \int_{\underline{a}}^{\infty} s_1(a,t) g_1(a,t) da + \int_{\underline{a}}^{\infty} s_2(a,t) g_2(a,t) da$$

We solve this system using the following algorithm. Guess a function $r^0(t)$ and then for $\ell = 0, 1, 2, ...$ follow

1. Given $r^\ell(t)$, solve the HJB equation (22) with terminal condition $v_j^\ell(a,T) = v_j(a)$ backward in time to compute the time path of $v_j^\ell(a,t)$. Also compute the implied saving policy function $s_j^\ell(a,t)$

2. Given $s_j^\ell(a,t)$, solve the Kolmogorov Forward equation (23) with initial condition $g_j(a,0) =$

$g_{j0}(a)$ forward in time to calculate the time path for $g_j^\ell(a,t)$.

3. Given $s_j^\ell(a,t)$ and $g_j^\ell(a,t)$ calculate

$$S^\ell(t) = \int_{\underline{a}}^{\infty} a g_1^\ell(a,t)da + \int_{\underline{a}}^{\infty} a g_2^\ell(a,t)da$$

4. Update $r^{\ell+1}(t) = r^\ell(t) - \xi \frac{dS^\ell(t)}{dt}$, where $\xi > 0$.

5. Stop when $r^{\ell+1}$ is sufficiently close to $r^\ell(t)$.

4.1 Solving the Time-Dependent HJB Equation (Step 1)

Approximate the value function at I discrete points in the wealth dimension and N discrete points in the time dimension, and use the shorthand notation $v_{i,j}^n = v_j(a_i, t^n)$. The discrete approximation to the time-dependent HJB (22) is

$$\rho v_{i,j}^n = u(c_{i,j}^{n+1}) + (v_{i,j}^n)'[z_j + r^{n+1}a_i - c_{i,j}^{n+1}] + \lambda_j[v_{i,-j}^n - v_{i,j}^n] + \frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta t} \tag{25}$$

with terminal condition $v_{i,j}^N = v_j(a_i)$. Given v^{n+1}, this system can be solved for v^n exactly as in Section 1.2. In particular, one can write this in matrix notation as

$$\rho v^n = u^{n+1} + \mathbf{A}^{n+1}v^n + \frac{1}{\Delta t}(v^{n+1} - v^n) \tag{26}$$

where $\mathbf{A}^{n+1}$ is defined in an analogous fashion to Section 1.2 and still has the interpretation of the transition matrix of the discretized stochastic process for (a_t, z_t). Now each n has the interpretation of a time step instead of an iteration on the stationary value function. The reason for this similarity in the algorithm is that intuitively a stationary value function can be found by solving a time-dependent problem and going far enough back in time, i.e. as $t \to -\infty$.

4.2 Solving the Time-Dependent Kolmogorov Forward Eq. (Step 2)

Analogously to the value function, we approximate the density at J discrete points in the wealth dimension and N discrete points in the time dimension, and use the shorthand notation $g_{i,j}^n = g_j(a_i, t^n)$. Similarly to Section 2 one can directly make use of the transition matrix $\mathbf{A}^n$ defined when solving the time-dependent HJB equation (Section 4.1). Given an initial condition $g_{i,j}^0 = g_{j,0}(a_i)$, the Kolmogorov Forward equation (23) is then easily solved. One here has the option of using either an explicit method

$$\frac{g^{n+1} - g^n}{\Delta t} = (\mathbf{A}^n)^{\mathrm{T}} g^n \quad \Rightarrow \quad g^{n+1} = \Delta t(\mathbf{A}^n)^{\mathrm{T}} g^n + g^n \tag{27}$$

or an implicit method

$$\frac{g^{n+1} - g^n}{\Delta t} = (\mathbf{A}^n)^{\mathrm{T}} g^{n+1} \quad \Rightarrow \quad g^{n+1} = (\mathbf{I} - \Delta t(\mathbf{A}^n)^{\mathrm{T}})^{-1} g^n.$$

Note that these schemes preserve mass: starting from any initial distribution g^0 that sums to one, all future g^n's also sum to one. This follows from the fact that the rows of the intensity matrices $\mathbf{A}^n$ sum to zero. The implicit scheme is also guaranteed to preserve the positivity of g for arbitrary time steps Δt.

4.3 Results: Transition Dynamics

Figure 4 plots the time path for the equilibrium interest rate in response to a permanent increase in "unemployment risk", λ_2. Figure 5 plots the densities $g_1(a,t)$ and $g_2(a,t)$ at various points in time during the transition. For comparison, the Figure also plots densities in the initial steady state (dashed lines).

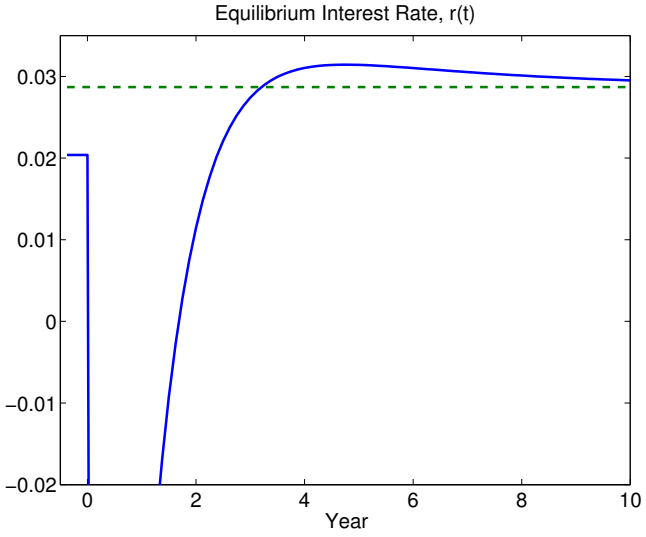


Figure 4: Time Path of Equilibrium Interest Rate

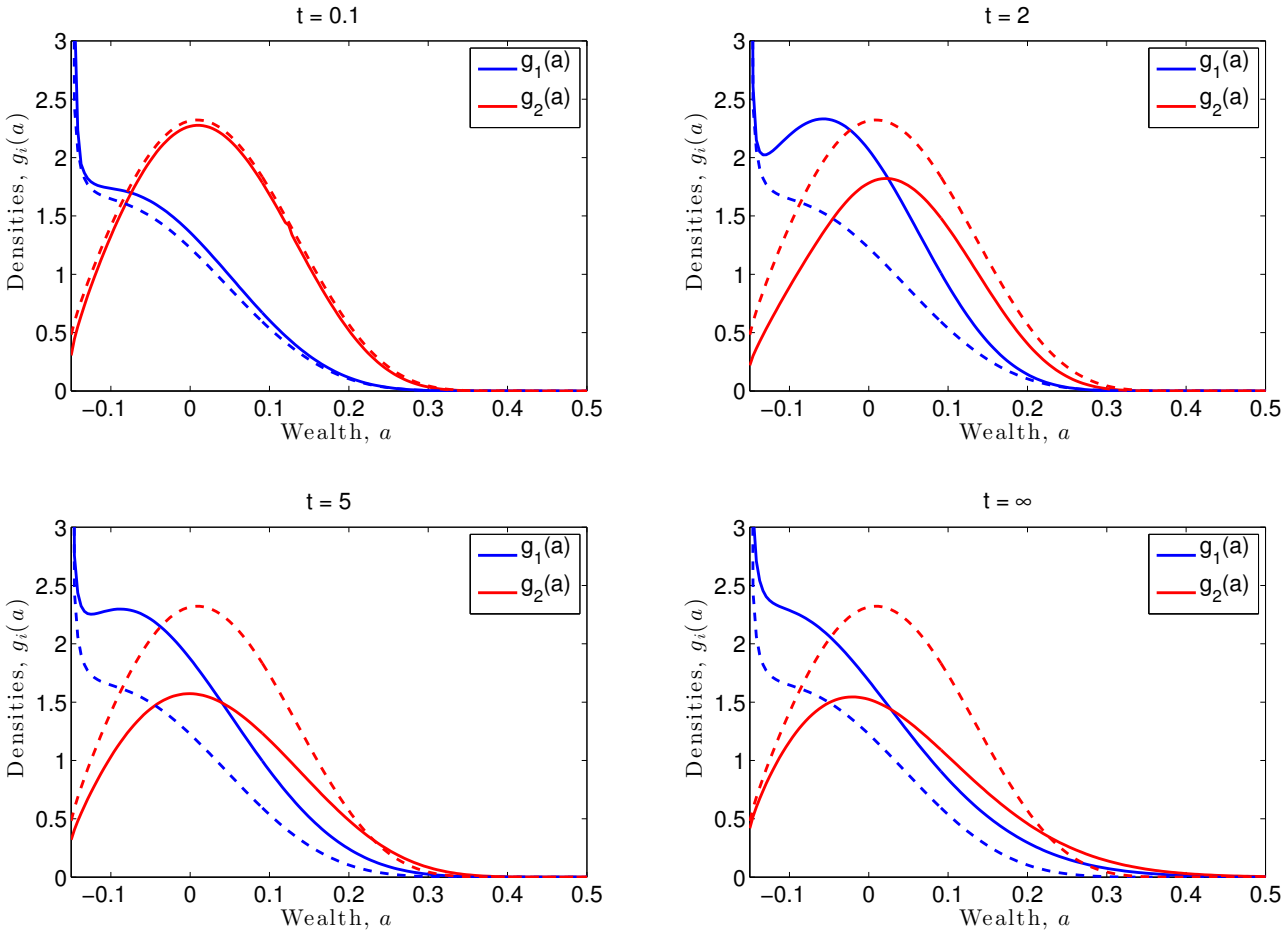


Figure 5: Dynamics of Wealth Distribution

5 Generalization to Diffusion Process

The system to be solved is:

$$\rho v(a,z) = \max_{c} \; u(c) + \partial_a v(a,z)[z + ra - c] + \mu(z)\partial_z v(a,z) + \frac{\sigma^2(z)}{2}\partial_{zz} v(a,z) \tag{28}$$

$$0 = -\partial_a[s(a,z)g(a,z)] - \partial_z[\mu(z)g(a,z)] + \frac{1}{2}\partial_{zz}[\sigma^2(z)g(a,z)] \tag{29}$$

$$1 = \int_0^\infty \int_{\underline{a}}^\infty g(a,z)dadz \tag{30}$$

$$0 = \int_0^\infty \int_{\underline{a}}^\infty ag(a,z)dadz \equiv S(r) \tag{31}$$

We assume that the z-process gets reflected at some $\underline{z}$ and $\bar{z}$. One can show that this gives rise to the following boundary conditions for v:[6]

$$0 = \partial_z v(a, \underline{z}) = \partial_z v(a, \bar{z})$$

Finally, we have the state constraint boundary condition:

$$\partial_a v(\underline{a}, z) \geq u'(z + r\underline{a}), \quad \text{all } z. \tag{32}$$

We again use a finite difference method and use the short-hand notation $v(a_i, z_j) = v_{i,j}$. Note that we changed notation slightly and i now indexes wealth and j indexes productivity.

5.1 HJB Equation

See matlab program `HJB_diffusion_implicit.m`. With a diffusion process, an explicit method becomes extremely inefficient so we here only explain the solution of the HJB with an implicit method. The derivative in the a dimension is again approximated using an upwind method, i.e. using either a forward or a backward difference approximation depending on the sign of the drift:

$$\begin{aligned} \partial_{a,B} v_{i,j} &= \frac{v_{i,j} - v_{i-1,j}}{\Delta a} \\ \partial_{a,F} v_{i,j} &= \frac{v_{i+1,j} - v_{i,j}}{\Delta a} \end{aligned} \tag{33}$$

Similarly, we also use an upwind method in the z-direction. For the second-order derivative, we use a central difference approximation. Hence:

$$\begin{aligned} \partial_{z,B} v_{i,j} &= \frac{v_{i,j} - v_{i,j-1}}{\Delta z} \\ \partial_{z,F} v_{i,j} &= \frac{v_{i,j+1} - v_{i,j}}{\Delta z} \\ \partial_{zz} v_{i,j} &= \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{(\Delta z)^2} \end{aligned}$$

Analogously to the model with Poisson shocks, v^{n+1} is now implicitly defined by the equation

$$\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^{n+1} = u(c_{i,j}^n) + \partial_a v_{i,j}^{n+1}[z_j + ra_i - c_{i,j}^n] + \mu_j \partial_z v_{i,j}^{n+1} + \frac{\sigma_j^2}{2} \partial_{zz} v_{i,j}^{n+1}$$

Note the $n+1$ superscripts on the right-hand side of the equation. The main advantage of the implicit scheme is that the step size Δ can be arbitrarily large.

[6] See e.g. Section 3.5 in Dixit (1993).

Upwind Scheme. We again need to use an "upwind scheme." As above, the idea is still to use the forward difference approximation whenever the drift of the state variable is positive and the backward difference approximation whenever it is negative. We use the following finite difference approximation to (28).

$$\begin{aligned}\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta}+\rho v_{i,j}^{n+1} =& u(c_{i,j}^{n})+\partial_{a,F}v_{i,j}^{n+1}[z_j+ra_i-c_{i,j,F}^{n}]^{+}+\partial_{a,B}v_{i,j}^{n+1}[z_j+ra_i-c_{i,j,B}^{n}]^{-} \\ &+\partial_{z,F}v_{i,j}^{n+1}\mu_j^{+}+\partial_{z,B}v_{i,j}^{n+1}\mu_j^{-}+\frac{\sigma_j^2}{2}\partial_{zz}v_{i,j}^{n+1}\end{aligned} \tag{34}$$

Equation (34) constitutes a system of $I \times J$ linear equations, and it can be written in matrix notation using the following steps. Substituting the definition of the derivatives (33), and defining $s_{i,j,F}^{n}=z_j+ra_i-c_{i,j,F}^{n}$ and similarly for $s_{i,j,B}^{n}$, (34) is

$$\begin{aligned}\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta}+\rho v_{i,j}^{n+1} =& u(c_{i,j}^{n})+\frac{v_{i+1,j}^{n+1}-v_{i,j}^{n+1}}{\Delta a}(s_{i,j,F}^{n})^{+}+\frac{v_{i,j}^{n+1}-v_{i-1,j}^{n+1}}{\Delta a}(s_{i,j,B}^{n})^{-} \\ &+\frac{v_{i,j+1}^{n+1}-v_{i,j}^{n+1}}{\Delta z}\mu_j^{+}+\frac{v_{i,j}^{n+1}-v_{i,j-1}^{n+1}}{\Delta z}\mu_j^{-}+\frac{\sigma_j^2}{2}\frac{v_{i,j+1}^{n+1}-2v_{i,j}^{n+1}+v_{i,j-1}^{n+1}}{(\Delta z)^2}\end{aligned}$$

Collecting terms with the same subscripts on the right-hand side

$$\begin{aligned}&\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta}+\rho v_{i,j}^{n+1}=u(c_{i,j}^{n})+v_{i-1,j}^{n+1}x_{i,j}+v_{i,j}^{n+1}(y_{i,j}+\upsilon_j)+v_{i+1,j}^{n+1}z_{i,j}+v_{i,j-1}^{n+1}\chi_j+v_{i,j+1}^{n+1}\zeta_j \\ &x_{i,j}=-\frac{(s_{i,j,B}^{n})^{-}}{\Delta a}, \\ &y_{i,j}=-\frac{(s_{i,j,F}^{n})^{+}}{\Delta a}+\frac{(s_{i,j,B}^{n})^{-}}{\Delta a}, \\ &z_{i,j}=\frac{(s_{i,j,F}^{n})^{+}}{\Delta a} \\ &\chi_j=-\frac{\mu_j^{-}}{\Delta z}+\frac{\sigma_j^2}{2(\Delta z)^2} \\ &\upsilon_j=\frac{\mu_j^{-}}{\Delta z}-\frac{\mu_j^{+}}{\Delta z}-\frac{\sigma_j^2}{(\Delta z)^2} \\ &\zeta_j=\frac{\mu_j^{+}}{\Delta z}+\frac{\sigma_j^2}{2(\Delta z)^2}\end{aligned} \tag{35}$$

Note that importantly $x_{1,j}=z_{I,j}=0$ for all j so $v_{0,j}^{n+1}$ and $v_{I+1,j}^{n+1}$ are never used. At the boundaries in the j dimension, the equations become

$$\begin{aligned}&\frac{v_{i,1}^{n+1}-v_{i,1}^{n}}{\Delta}+\rho v_{i,1}^{n+1}=u(c_{i,1}^{n})+v_{i-1,1}^{n+1}x_{i,1}+v_{i,1}^{n+1}(y_{i,1}+\upsilon_1+\chi_1)+v_{i+1,1}^{n+1}z_{i,1}+v_{i,2}^{n+1}\zeta_1 \\ &\frac{v_{i,J}^{n+1}-v_{i,J}^{n}}{\Delta}+\rho v_{i,J}^{n+1}=u(c_{i,J}^{n})+v_{i-1,J}^{n+1}x_{i,J}+v_{i,J}^{n+1}(y_{i,J}+\upsilon_J+\zeta_J)+v_{i+1,J}^{n+1}z_{i,J}+v_{i,J-1}^{n+1}\chi_J\end{aligned}$$

where, in the first equation, we have used that $\partial_{z,B} v_{i,1} = \frac{v_{i,1} - v_{i,0}}{\Delta z} = 0$ and hence $v_{i,0} = v_{i,1}$. Similarly, in the second equation, $\partial_{z,F} v_{i,J} = \frac{v_{i,J+1} - v_{i,J}}{\Delta z} = 0$ and hence $v_{i,J+1} = v_{i,J}$. Note that we here defined the boundary conditions relative to the points $j = 0$ and $j = J + 1$ and used the values $v_{i,0}$ and $v_{i,J+1}$. These points are sometimes called "ghost nodes". Equation (35) is a system of $I \times J$ linear equations which can be written in matrix notation as:

$$\frac{1}{\Delta}(v^{n+1} - v^n) + \rho v^{n+1} = u^n + \mathbf{A}^n v^{n+1} \tag{36}$$

where v^n is a vector of length $I \times J$ with entries $(v_{1,1}, ..., v_{I,1}, v_{1,2}, ..., v_{I,2}, ..., v_{I,J})$ and $\mathbf{A}^n = \widetilde{\mathbf{A}}^n + \mathbf{C}$ where the $(I \times J) \times (I \times J)$ matrices $\widetilde{\mathbf{A}}^n$ and $\mathbf{C}$ are

$$\widetilde{\mathbf{A}}^n = \begin{bmatrix}
y_{1,1} & z_{1,1} & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
x_{2,1} & y_{2,1} & z_{2,1} & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & x_{I,1} & y_{I,1} & 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & 0 & 0 & y_{1,2} & z_{1,2} & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & 0 & x_{2,2} & y_{2,2} & z_{2,2} & 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 & x_{I,2} & y_{I,2} & 0 & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & y_{1,J} & z_{1,J} & 0 & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & x_{2,J} & y_{2,J} & z_{2,J} & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & x_{I,J} & y_{I,J}
\end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix}
\upsilon_1+\chi_1 & 0 & \cdots & \cdots & 0 & \zeta_1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \upsilon_1+\chi_1 & 0 & \ddots & \ddots & 0 & \zeta_1 & 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ddots & 0 & \upsilon_1+\chi_1 & 0 & \ddots & \ddots & 0 & \zeta_1 & 0 & \ddots & \ddots & \vdots \\
\chi_2 & 0 & \ddots & 0 & \upsilon_2 & 0 & \ddots & \ddots & 0 & \zeta_2 & 0 & \ddots & \vdots \\
0 & \chi_2 & 0 & \ddots & 0 & \upsilon_2 & 0 & \ddots & \ddots & 0 & \zeta_2 & 0 & \vdots \\
\vdots & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & \ddots & \vdots \\
\vdots & \ddots & 0 & \chi_2 & 0 & \ddots & 0 & \upsilon_2 & 0 & \ddots & \ddots & 0 & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & 0 & \chi_J & 0 & \ddots & \ddots & \upsilon_J+\zeta_J & 0 & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 & \chi_J & 0 & \ddots & 0 & \upsilon_J+\zeta_J & 0 & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \chi_J & 0 & \cdots & 0 & \upsilon_J+\zeta_J
\end{bmatrix}$$

Equation (36) can again be solved very efficiently in matlab. Figure 6 again plots a visualization of the intensity matrix $\mathbf{A}$ in practice (using Matlab's `spy` function).

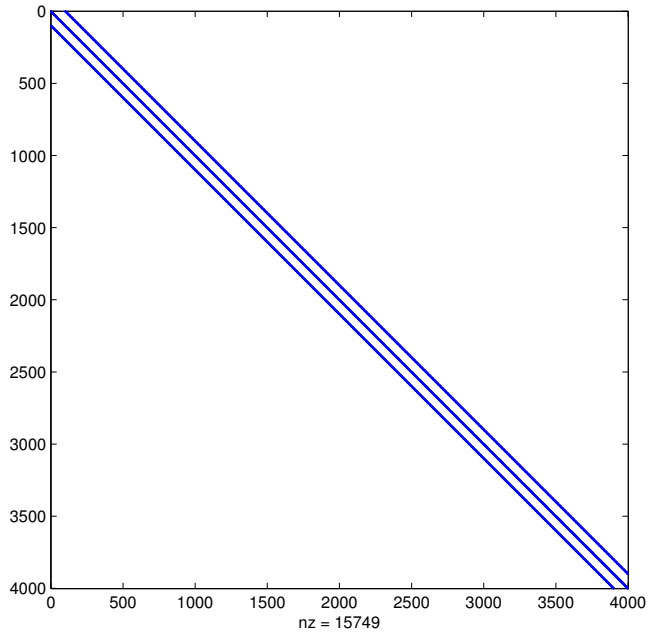


Figure 6: Visualization of the matrix $\mathbf{A}$ using spy in model with diffusion process

5.2 Kolmogorov Forward Equation and Equilibrium

See matlab program `huggett_diffusion_partialeq.m`. The Kolmogorov Forward equation is solved exactly as in section 2, and the equilibrium is found as in section 3.

5.3 Results

Figures 7 and 8 plot the functions $s(a, z)$ and $g(a, z)$.

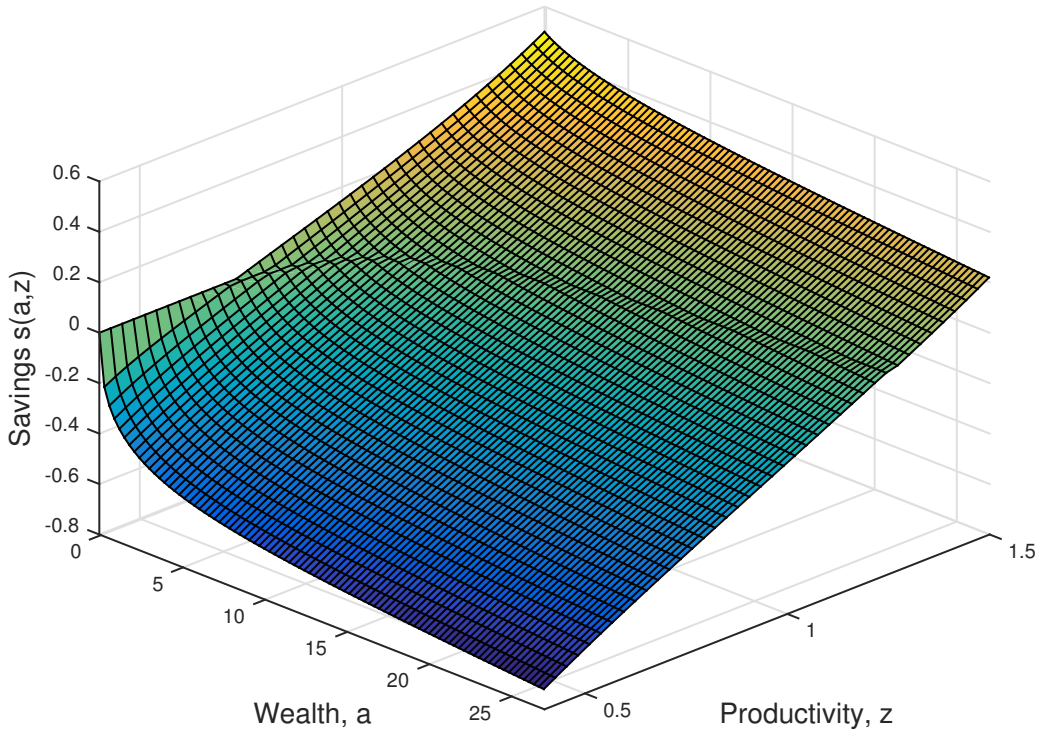


Figure 7: Savings Policy Function in Huggett Model with Diffusion

6 Aiyagari Model

We now briefly explain how to solve the Aiyagari model in Section 6 of Achdou et al. (2020). As in the paper we focus on the case where productivity follows a diffusion process so as to explain how to handle that case. Of course, it is also straightforward to solve an Aiyagari model in which income follows a two-state Poisson process. Codes for this case are also available: `aiyagari_poisson_steadystate.m`, `aiyagari_poisson_asset_supply.m` and `aiyagari_poisson_MITshock.m`.

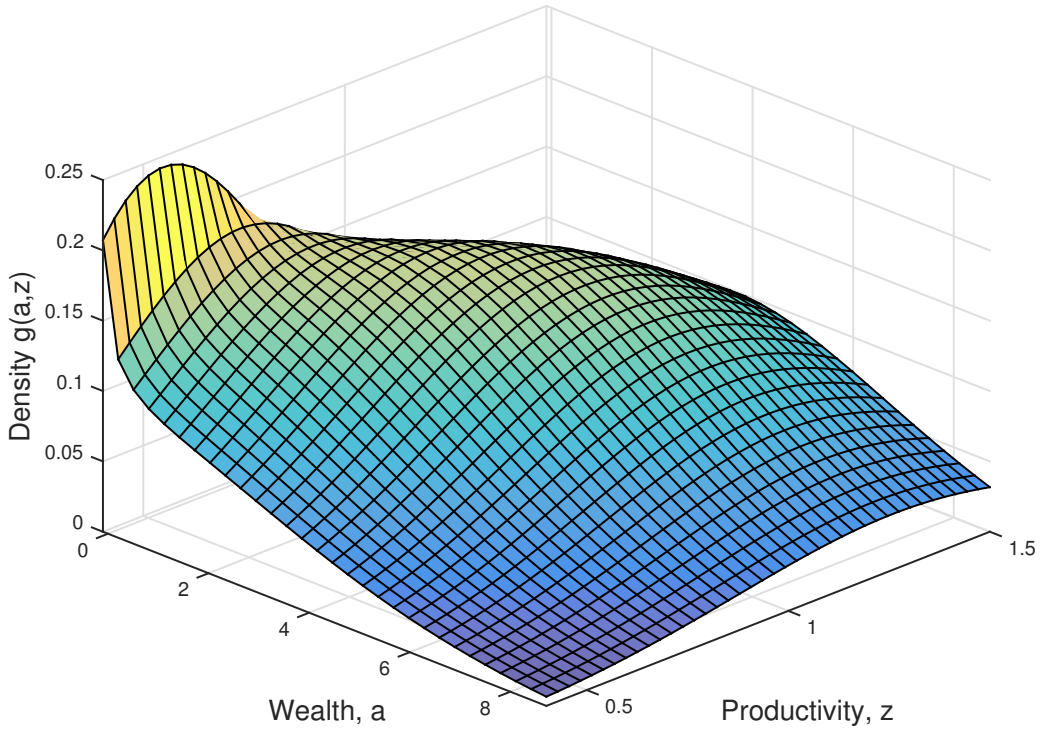


Figure 8: Wealth Distribution in Huggett Model with Diffusion

6.1 Steady State

See matlab program `aiyagari_diffusion_equilibrium.m`, Julia program `aiyagari_diffusion_equilibrium.jl` and C++ program `aiyagari_diffusion_equilibrium.cpp`. A steady state or stationary equilibrium can be represented by the following system of equations which we aim to solve numerically:

$$\rho v(a,z) = \max_c \; u(c) + \partial_a v(a,z)(wz + ra - c) + \partial_z v(a,z)\mu(z) + \frac{1}{2}\partial_{zz} v(a,z)\sigma^2(z) \tag{37}$$

$$0 = \partial_a(s(a,z)g(a,z)) - \partial_z(\mu(z)g(a,z)) + \frac{1}{2}\partial_{zz}(\sigma^2(z)g(a,z)) \tag{38}$$

$$r = \partial_K F(K,1) - \delta, \quad w = \partial_L F(K,1), \tag{39}$$

$$K = \int_{\underline{z}}^{\bar{z}} \int_{\underline{a}}^{\infty} a g(a,z) da dz \tag{40}$$

on $(\underline{a},\infty)\times(\underline{z},\bar{z})$, where $s(a,z) = wz+ra-c(a,z), c(a,z) = (u')^{-1}(\partial_a v(a,z))$ and with boundary conditions

$$u'(wz + r\underline{a}) \geq \partial_a v(\underline{a},z), \quad \text{all } z$$
$$\partial_z v(a,\underline{z}) = 0, \qquad \partial_z v(a,\bar{z}) = 0, \quad \text{all } a$$

The algorithm for solving the HJB and KF equations is the same as in Sections 5.1 and 5.2. To find the equilibrium wage and interest rate w and r, we use a fixed point algorithm on the scalar K. Alternatively, one can express w as a function of r using (39) and (40) and use a bisection method to solve for the equilibrium r.

6.2 Transition Dynamics

See matlab program `aiyagari_diffusion_transition.m`. The algorithm for solving the HJB and KF equations is the natural generalization to the time-dependent case of that outlined in Sections 5.1 and 5.2. To solve for the equilibrium time paths of the wage and interest rate, we use a fixed point algorithm on the function $K(t)$.

6.3 "MIT Shocks"

See matlab program `aiyagari_poisson_MITshock.m`. We consider the version of the Aiyagari model with Poisson income shocks and compute the impulse response to a negative aggregate productivity shock that mean reverts over time. This shock is modeled as an "MIT shock," i.e. an unexpected (zero probability) shock followed by a deterministic transition. More precisely, we assume that the aggregate production function is $Y_t = F_t(K, L) = A_t K^\alpha L^{1-\alpha}$ and aggregate productivity follows a deterministic version of an Ornstein-Uhlenbeck process (the continuous-time analogue of an AR(1) process):

$$dA_t = \nu(\bar{A} - A_t)dt.$$

The parameter ν governs the speed of mean reversion (one can show that $Corr(A_t, A_{t+s}) = e^{-\nu s}$). Figure 9 plots the impulse response to this productivity MIT-shock. In particular note the measures of income and wealth inequality in the last two panels.

6.4 Visualizing Evolution of Wealth Distribution as Movie

After running `aiyagari_diffusion_transition.m` or `aiyagari_poisson_MITshock.m`, you can run `make_movie.m` to make the movie of the transition of the distribution.

After you run `make_movie.m`, you will have `distribution.avi` file. One can use `ffmpeg` (available at `https://www.ffmpeg.org/`) to convert from `avi` file to `mp4` file. In particular type

```
ffmpeg -i distribution.avi -c:v libx264 -g 30 -pix_fmt yuv420p distribution.mp4
```

and this will create a `mp4` file that is accepted by pretty much anything (and it works on website with `html5`).[7]

[7] "`-pix_fmtyuv420p`" is only necessary while browsers and media players are being updated. It is using old format because ffmpeg was updated recently, and most media players have not been.

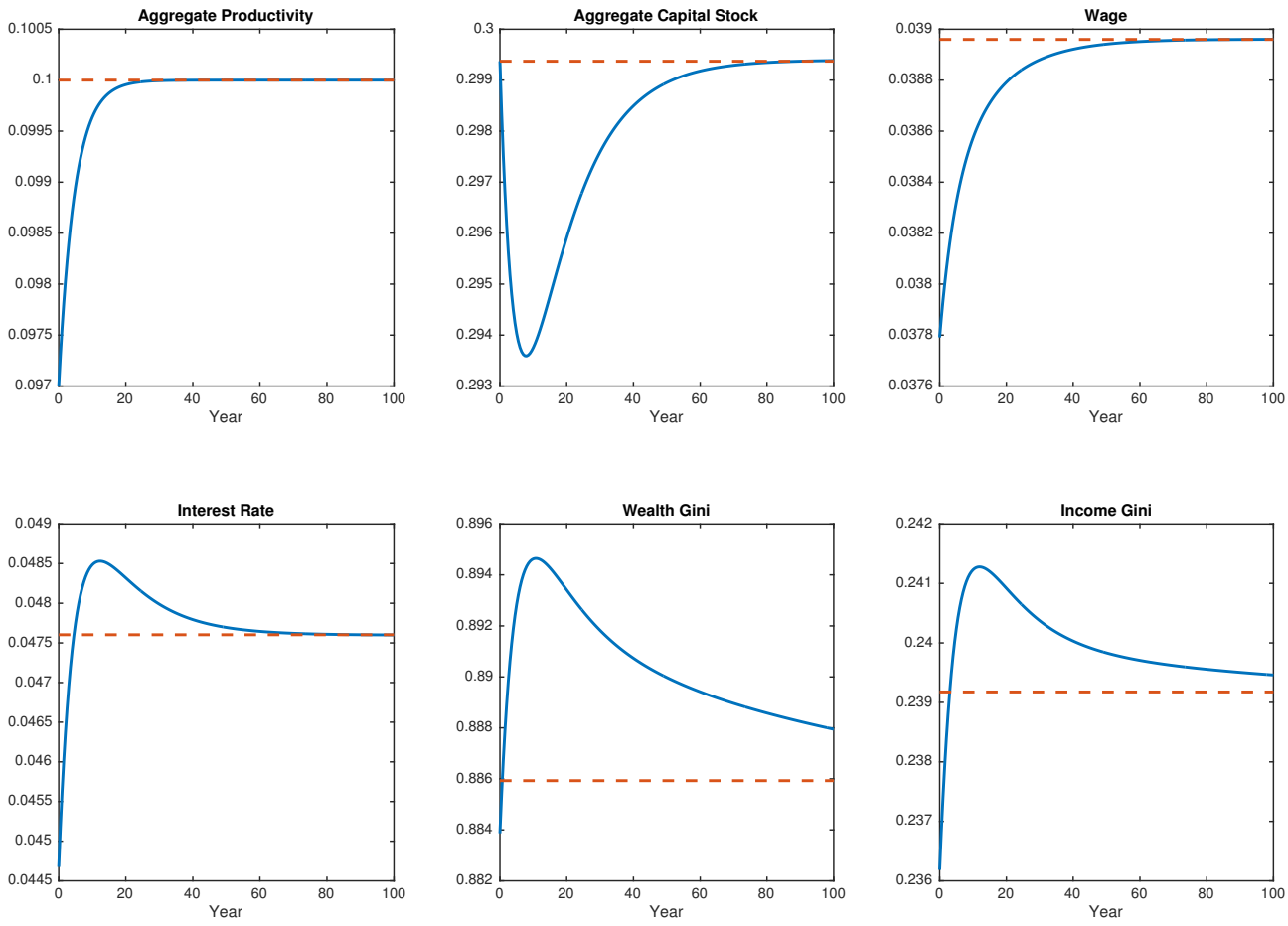


Figure 9: Impulse Response to Negative Productivity MIT-shock in Aiyagari Model

If you do not want to go through the pain of compiling your own `ffmpeg` binary and have a Mac, you can download a pre-compiled binary at `http://ffmpegmac.net/`. Copy the binary `ffmpeg` into the same directory as your `avi` file and run (don't forget the "`./`" in the beginning)

```
./ffmpeg -i distribution.avi -c:v libx264 -g 30 -pix_fmt yuv420p distribution.mp4
```

The resulting movie is at `http://www.princeton.edu/~moll/HACTproject/distribution.mp4`.

7 Non-uniform Grids

In all previous sections, we worked with uniformly spaced grids. But for many applications, we may want to economize on grid points. This can be achieved by working with non-uniform grids and "putting grid points" at points in the state space where the value and density functions have the most curvature.

7.1 HJB equation with non-uniform grid

Extending our algorithm for the HJB equation to the case of non-uniform grids is straightforward. Denoting by $\Delta a_{i,+} = a_{i+1} - a_i$ and $\Delta a_{i,-} = a_i - a_{i-1}$, the forward and backward distance between two grid points, we simply change (8) to

$$\begin{aligned} v_j'(a_i) &\approx \frac{v_{i+1,j} - v_{i,j}}{a_{i+1} - a_i} = \frac{v_{i+1,j} - v_{i,j}}{\Delta a_{i,+}} \equiv v_{i,j,F}' \\ v_j'(a_i) &\approx \frac{v_{i,j} - v_{i-1,j}}{a_i - a_{i-1}} = \frac{v_{i,j} - v_{i-1,j}}{\Delta a_{i,-}} \equiv v_{i,j,B}' \end{aligned} \tag{41}$$

Following the same steps as above, we end up with

$$\begin{aligned} &\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^{n+1} = u(c_{i,j}^n) + v_{i-1,j}^{n+1} x_{i,j} + v_{i,j}^{n+1} y_{i,j} + v_{i+1,j}^{n+1} z_{i,j} + v_{i,-j}^{n+1} \lambda_j \quad \text{where} \\ &x_{i,j} = -\frac{(s_{i,j,B}^n)^-}{\Delta a_{i,-}}, \quad y_{i,j} = -\frac{(s_{i,j,F}^n)^+}{\Delta a_{i,+}} + \frac{(s_{i,j,B}^n)^-}{\Delta a_{i,-}} - \lambda_j, \quad z_{i,j} = \frac{(s_{i,j,F}^n)^+}{\Delta a_{i,+}} \end{aligned} \tag{42}$$

This can again be written in matrix form (15) where the intensity matrix $\mathbf{A}^n$ has the same structure as above but with the entries in (42). The rest of the algorithm is unchanged.

If an approximation to the second derivative of v_j is needed as in Section 8, a good candidate approximation is

$$v_j''(a_i) \approx \frac{\Delta a_{i,-} v_{i+1,j} - (\Delta a_{i,-} + \Delta a_{i,+}) v_{i,j} + \Delta a_{i,+} v_{i-1,j}}{\frac{1}{2}(\Delta a_{i,+} + \Delta a_{i,-}) \Delta a_{i,-} \Delta a_{i,+}} \tag{43}$$

This approximation can be derived from a Taylor approximation to v_j[8] and it can be seen that with $\Delta a_{i,-} = \Delta a_{i,+} = \Delta a$, it reduces to the standard second-derivative approximation in the case with uniform grids:

$$v_j''(a_i) \approx \frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{(\Delta a)^2}$$

[8]Consider a second-order Taylor approximation to v_j around a_i:

$$\begin{aligned} v_{i+1,j} &\approx v_{i,j} + \Delta a_{i,+} v_j'(a_i) + \frac{1}{2}(\Delta a_{i,+})^2 v_j''(a_i) \\ v_{i-1,j} &\approx v_{i,j} - \Delta a_{i,-} v_j'(a_i) + \frac{1}{2}(\Delta a_{i,-})^2 v_j''(a_i) \end{aligned}$$

Multiply the first equation by $\Delta a_{i,-}$ and the second equation by $\Delta a_{i,+}$ and add the two equations

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

Rearranging yields (43).

7.2 Kolmogorov Forward equation with non-uniform grid

Extending the Kolmogorov Forward equation to the case of a non-uniform grid requires more work. Section 2 suggests working with the transpose of the intensity matrix $\mathbf{A}$ and to solve

$$\frac{g^{n+1}-g^n}{\Delta t} = \mathbf{A}^{\mathrm{T}} g^n$$

or the implicit analogue. The problem with this scheme is that it is not guaranteed to preserve mass: starting with an initial distribution g that integrates/sums to one, the total mass may converge to a different number over time (including zero or infinity).

To see this consider a simplified example *with one income type only* $z_1 = z_2$ in which case the intensity matrix is of size $I \times I$ with entries $A_{i,i'}$ given by

$$A_{i,i-1} = x_i = -\frac{(s^n_{i,B})^-}{\Delta a_{i,-}}, \quad A_{i,i} = y_i = -\frac{(s^n_{i,F})^+}{\Delta a_{i,+}} + \frac{(s^n_{i,B})^-}{\Delta a_{i,-}}, \quad A_{i,i+1} = z_i = \frac{(s^n_{i,F})^+}{\Delta a_{i,+}} \tag{44}$$

From (44), we can see that the rows of $\mathbf{A}$ still sum to zero, $\sum_{i'} A_{i,i'} = 0$. Therefore

$$\sum_{i=1}^{I} \frac{g_i^{n+1}-g_i^n}{\Delta t} = \sum_{i'=1}^{I} \left(\sum_{i=1}^{I} A_{i',i} \right) g_{i'} = 0$$

In the case of a *uniform* grid, this also implies mass preservation $\sum_{i=1}^{I} \frac{g_i^{n+1}-g_i^n}{\Delta t} \Delta a = 0$. However, in the case of a *non-uniform* grid what we want instead is that an appropriate approximation of the integral equals zero, i.e. something like $\sum_{i=1}^{I} \frac{g_i^{n+1}-g_i^n}{\Delta t} \Delta a_i = 0$ (where the Δa_i's depend on the precise integral approximation method).

The following solution to this problem seems to work well in practice, in particular it preserves mass and the positivity of g.[9] First, approximate the integral of g with the trapezoidal rule [10]

$$\int_{\underline{a}}^{a_{\max}} g(a,t^n)da \approx \frac{1}{2}\sum_{i=1}^{I-1}(a_{i+1}-a_i)(g_{i+1}+g_i) = \sum_{i=1}^{I} g_i \tilde{\Delta} a_i$$

$$\tilde{\Delta} a_i = \begin{cases} \frac{1}{2}\Delta a_{i,+}, & i = 1 \\ \frac{1}{2}(\Delta a_{i,+} + \Delta a_{i,-}), & i = 2, ..., I-1 \\ \frac{1}{2}\Delta a_{i,-}, & i = I \end{cases}$$

The key idea is now that, rather than working with the vector g with elements g_i, we work directly with a vector $\tilde{g}$ whose elements are $\tilde{g}_i = g_i \tilde{\Delta} a_i$ and which must therefore satisfy

[9] We have not yet been able to derive a rigorous theoretical justification for the proposed scheme.

[10] See `http://en.wikipedia.org/wiki/Trapezoidal_rule#Non-uniform_grid`

$\sum_{i=1}^{I} \tilde{g}_i = 1$. Given an initial condition $\tilde{g}^1$ we simply solve the following analogue of (27):

$$\frac{\tilde{g}^{n+1} - \tilde{g}^n}{\Delta t} = \mathbf{A}^{\mathrm{T}} \tilde{g}^n.$$

Following the same logic as above, the condition $\sum_{i'=1}^{I} A_{i,i'} = 0$ guarantees mass preservation $\sum_{i=1}^{I} \tilde{g}_i = \sum_{i=1}^{I} g_i \tilde{\Delta} a_i = 1$. We can then always back out the true distribution g^n from $g_i = \tilde{g}_i/(\tilde{\Delta} a_i)$. In matrix form, we have $\tilde{g} = Dg$ where D is a diagonal matrix with elements $\Delta \tilde{a}_i, i = 1, ..., I$. Therefore underlying distribution g can also be found from $g = D^{-1}\tilde{g}$.[11]

It is also straightforward to show that the same approach – working with the rescaled density $\tilde{g} = Dg$ – also applies to the case with multiple income states (or a continuum).

8 Aiyagari Model with Fat-tailed Wealth Distribution

See matlab code `fat_tail_partialeq.m`. This section shows how to extend our computational methods to the model with two assets and a fat-tailed wealth distribution from Section 6 of Achdou et al. (2020).

8.1 Model Setup

The system of equations to be solved is

$$\rho v_j(a) = \max_{c, k \leq a + \phi} u(c) + v_j'(a)(z_j + ra + (R - r)k - c) + \frac{1}{2} v_j''(a)\sigma^2 k^2 + \lambda_j(v_{-j}(a) - v_j(a)) \quad (45)$$

$$0 = -\frac{d}{da}[s_j(a)g_j(a)] + \frac{1}{2}\frac{d^2}{da^2}[\sigma^2 k_j(a)^2 g_j(a)] - \lambda_j g_j(a) + \lambda_{-j} g_{-j}(a) \quad (46)$$

$$\int_{\underline{a}}^{\infty} k_1(a)g_1(a)da + \int_{\underline{a}}^{\infty} k_2(a)g_2(a)da = \int_{\underline{a}}^{\infty} a g_1(a)da + \int_{\underline{a}}^{\infty} a g_2(a)da \quad (47)$$

[11] A previous version of this Appendix advocated working with a rescaled version of the intensity matrix $\mathbf{A}$ rather than a rescaled version of the density g: replace the intensity matrix $\mathbf{A}$ in (20) or (27) with an alternative intensity matrix $\tilde{\mathbf{A}}$ that satisfies $\sum_{i'=1}^{I} \tilde{A}_{i,i'} \tilde{\Delta} a_{i'} = 0$ and then solve

$$\frac{g^{n+1} - g^n}{\Delta t} = \tilde{\mathbf{A}}^{\mathrm{T}} g^n.$$

The rescaled intensity matrix was given by $\tilde{\mathbf{A}} = D\mathbf{A}D^{-1}$ where again D is a diagonal matrix with elements $\Delta \tilde{a}_i, i = 1, ..., I$. The two approaches are equivalent given that $\tilde{g} = Dg$. However, we prefer the approach in the text because it is much more transparent and easier to explain. We are grateful to Matthieu Gomez for suggesting the approach in the current version.

Optimal consumption and choice of risky assets are

$$c_j(a) = v_j'(a)^{-1/\gamma} \tag{48}$$

$$k_j(a) = \min\left\{\frac{v_j'(a)}{-v_j''(a)}\frac{R-r}{\sigma^2}, a+\phi\right\}. \tag{49}$$

Boundary Conditions In theory, the HJB equation (45) is defined on $(\underline{a}, \infty)$ but in practice it has to be solved on a bounded interval $(\underline{a}, a_{\max})$. A non-trivial issue concerns the question what boundary condition to impose at $a_{\max}$. We use the asymptotic behavior of the value function in Lemma 2 in the paper to motivate boundary conditions as follows.[12] For large a, we have

$$v_j(a) = \tilde{v}_{0,j} + \tilde{v}_{1,j}a^{1-\gamma}$$

for unknown constants $\tilde{v}_{0,j}$ and $\tilde{v}_{1,j}$. Hence, we impose the following boundary condition

$$v_j''(a_{\max}) = -\gamma v_j'(a_{\max})/a_{\max}. \tag{50}$$

To solve (45), what we really need is a boundary condition for the term $\frac{\sigma^2}{2}v_j''(a)k(a)^2$. From (49) and (50)

$$k_j(a_{\max}) = \frac{R-r}{\gamma\sigma^2}a_{\max} \tag{51}$$

$$\begin{aligned}\frac{\sigma^2}{2}k_j(a_{\max})^2 v_j''(a_{\max}) &= -\frac{\sigma^2}{2}k_j(a_{\max})^2\gamma v_j'(a_{\max})/a_{\max} \\ &= v_j'(a_{\max})\xi, \qquad \xi = -\frac{(R-r)^2}{2\gamma\sigma^2}a_{\max}\end{aligned} \tag{52}$$

We will condition (52) below when solving (45) using a finite difference method. Finally, it sometimes helps numerical stability to impose a state constraint $a \leq a_{\max}$. This is equivalent to $c_j(a_{\max}) \geq z_j + ra_{\max} + (R-r)k_j(a_{\max})$ or using (51)

$$v_j'(a_{\max}) \leq \left(z_j + ra_{\max} + \frac{(R-r)^2}{\gamma\sigma^2}a_{\max}\right)^{-\gamma}.$$

8.2 Finite Difference Method for HJB Equation

The steps follow closely the solution method of the one-asset model. We therefore only outline the main differences. As before we use an implicit upwind method. In contrast to before, the HJB equation (45) now involves the second derivative of the value function. For sake of transparency, we here explain our finite difference method for a uniform grid. The code in `fat_tail_partialeq.m` instead uses a non-uniform grid as explained in Section 7 so as to be

[12] We thank Matthieu Gomez for suggesting this boundary condition.

able to put more points in the region of the state space where policy functions have a lot of curvature, i.e. close to the borrowing constraint.

Defining $s^n_{i,j,F} = z_j + (R-r)k_{i,j} + ra_i - c^n_{i,j,F}$ and similarly for $s^n_{i,j,B}$, the discretization of (45) is

$$\begin{aligned}\frac{v^{n+1}_{i,j} - v^n_{i,j}}{\Delta} + \rho v^{n+1}_{i,j} = u(c^n_{i,j}) &+ \frac{v^{n+1}_{i+1,j} - v^{n+1}_{i,j}}{\Delta a}(s^n_{i,j,F})^+ + \frac{v^{n+1}_{i,j} - v^{n+1}_{i-1,j}}{\Delta a}(s^n_{i,j,B})^- \\ &+ \lambda_j[v^{n+1}_{i,-j} - v^{n+1}_{i,j}] + \frac{\sigma^2}{2}k^2_{i,j}\frac{v^{n+1}_{i+1,j} - 2v^{n+1}_{i,j} + v^{n+1}_{i-1,j}}{(\Delta a)^2}\end{aligned}$$

Collecting terms with the same subscripts on the right-hand side:

$$\begin{aligned}&\frac{v^{n+1}_{i,j} - v^n_{i,j}}{\Delta} + \rho v^{n+1}_{i,j} = u(c^n_{i,j}) + v^{n+1}_{i-1,j}x_{i,j} + v^{n+1}_{i,j}y_{i,j} + v^{n+1}_{i+1,j}z_{i,j} + v^{n+1}_{i,-j}\lambda_j \quad \text{where} \\ &x_{i,j} = -\frac{(s^n_{i,j,B})^-}{\Delta a} + \frac{\sigma^2}{2}\frac{k^2_{i,j}}{(\Delta a)^2}, \\ &y_{i,j} = -\frac{(s^n_{i,j,F})^+}{\Delta a} + \frac{(s^n_{i,j,B})^-}{\Delta a} - \sigma^2\frac{k^2_{i,j}}{(\Delta a)^2} - \lambda_j, \\ &z_{i,j} = \frac{(s^n_{i,j,F})^+}{\Delta a} + \frac{\sigma^2}{2}\frac{k^2_{i,j}}{(\Delta a)^2}\end{aligned} \tag{53}$$

At the upper boundary $a = a_{\max} = a_I$, we make use of (52) and write the approximation as

$$\begin{aligned}\frac{v^{n+1}_{I,j} - v^n_{I,j}}{\Delta} + \rho v^{n+1}_{I,j} = u(c^n_{I,j}) &+ \frac{v^{n+1}_{I+1,j} - v^{n+1}_{I,j}}{\Delta a}(s^n_{I,j,F})^+ + \frac{v^{n+1}_{I,j} - v^{n+1}_{I-1,j}}{\Delta a}(s^n_{I,j,B})^- \\ &+ \lambda_j[v^{n+1}_{I,-j} - v^{n+1}_{I,j}] + \frac{v^{n+1}_{I,j} - v^{n+1}_{I-1,j}}{\Delta a}\xi\end{aligned}$$

so that the corresponding entries of (53) become.

$$\begin{aligned}&x_{I,j} = -\frac{(s^n_{I,j,B})^-}{\Delta a} - \frac{\xi}{\Delta a}, \\ &y_{I,j} = -\frac{(s^n_{I,j,F})^+}{\Delta a} + \frac{(s^n_{I,j,B})^-}{\Delta a} + \frac{\xi}{\Delta a} - \lambda_j, \\ &z_{I,j} = \frac{(s^n_{I,j,F})^+}{\Delta a}\end{aligned} \tag{54}$$

Equations (53) and (54) is a system of $2 \times I$ linear equations which can be written in matrix notation like equation (15)

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

and that can be solved efficiently.

8.3 Finite Difference for Kolmogorov Forward Equation

The solution of the Kolmogorov Forward equation (46) is exactly as in Section 2, with one difference: because of the second-order term one has to decide what to do at the upper end of the state space $a_{\max}$. The cleanest solution is to impose an artificial reflecting barrier. To this end, consider the "intensity matrix" $\mathbf{A}$. Rather than using (54) at the upper end of the state space, construct the transition matrix according to (53). But then move all entries corresponding to the (non-existent) grid point $I+1$ to the entry corresponding to I:

$$\begin{aligned}
\tilde{x}_{I,j} &= x_{I,j} = -\frac{(s^n_{I,j,B})^-}{\Delta a} + \frac{\sigma^2}{2}\frac{k^2_{I,j}}{(\Delta a)^2}, \\
\tilde{y}_{I,j} &= y_{I,j} + z_{I,j} = \frac{(s^n_{I,j,B})^-}{\Delta a} - \frac{\sigma^2}{2}\frac{k^2_{I,j}}{(\Delta a)^2} - \lambda_j, \\
\tilde{z}_{I,j} &= 0.
\end{aligned} \tag{55}$$

The interpretation is that whenever the process would leave the state space according to the discretized law of motion (if it would go to point $I+1$), it is "reflected" back in (back down to point I).[13]

8.4 Results

See Figure 10.

9 Accuracy of Finite Difference Method

See Appendix F.1 of Achdou et al. (2020) available at `https://benjaminmoll.com/HACT_appendix/` for various accuracy checks.

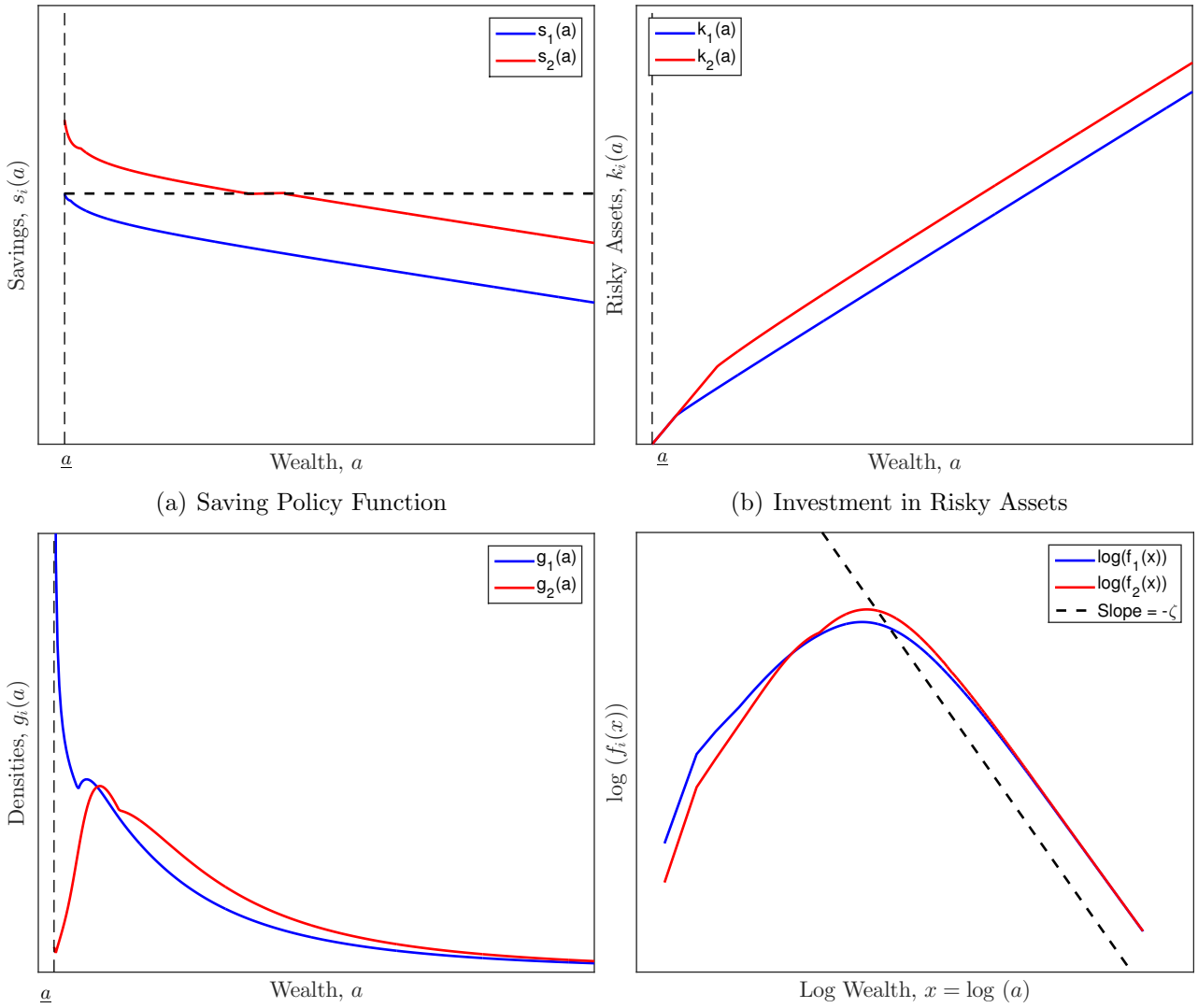
[13] This condition can be derived more rigorously as follows. For simplicity consider the case without productivity shocks $z_1 = z_2$ so that the process for wealth is $da_t = s(a_t)dt + \sigma k(a_t)dW_t$. Impose a reflecting barrier at $a = a_{\max}$. Then the infinitesimal generator corresponding to this process is given by

$$(\mathcal{A}f)(a) = s(a)f'(a) + \frac{\sigma^2}{2}k(a)^2 f''(a)$$

with the boundary condition corresponding to a reflecting barrier: $f'(a_{\max}) = 0$. Its discrete version is:

$$\begin{aligned}
(\mathbf{A}f)_i &= s_i^+ \frac{f_{i+1} - f_i}{\Delta a} + s_i^- \frac{f_i - f_{i-1}}{\Delta a} + \frac{\sigma^2}{2}k_i^2 \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta a)^2} \\
&= x_i f_{i-1} + y_i f_i + z_i f_{i-1}
\end{aligned}$$

where x_i, y_i, z_i are analogous to (53). The discretized boundary condition is $f'(a_I) \approx (f_{I+1} - f_I)/(\Delta a) = 0$ or $f_{I+1} = f_I$. Therefore $(\mathbf{A}f)_I = x_I f_{I-1} + (y_I + z_I)f_I = \tilde{x}_I f_I + \tilde{y}_I f_I$ with $\tilde{y}_I = y_I + z_I$ i.e. just like in (55).



(a) Saving Policy Function

(b) Investment in Risky Assets

(c) Stationary Densities

(d) Pareto Tail of Stationary Distribution

Figure 10: Optimal Choices and Pareto Tail of Wealth Distribution in Two-Asset Model

References

Achdou, Yves, Jiequn Han, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. 2020. "Income and Wealth Distribution in Macroeconomics: A Continuous-Time Approach." London School of Economics Working Paper.

Barles, G., and P. E. Souganidis. 1991. "Convergence of approximation schemes for fully nonlinear second order equations." *Asymptotic Analysis*, 4: 271–283.

Candler, Graham V. 1999. "Finite-Difference Methods for Dynamic Programming Problems." In *Computational Methods for the Study of Dynamic Economies*. Cambridge, England:Cambridge University Press.

Dixit, Avinash. 1993. *The Art of Smooth Pasting.* Fundamentals of Pure and Applied Economics 55, The Routledge.

Huggett, Mark. 1993. "The risk-free rate in heterogeneous-agent incomplete-insurance economies." *Journal of Economic Dynamics and Control*, 17(5-6): 953–969.