Solving continuous time heterogeneous agent models in MATLAB

Ananya Kotia

April 12, 2021

Outline

- Walk through how to solve an HJB equation in MATLAB
- Discuss extension to non-convexities
- Closely follow the numerical appendix in Achdou et al. (2017) available here: *https://benjaminmoll.com/wp-content/uploads/2020/02/HACT_Numerical_Appendix.pdf*
- Start with the Hugget model in partial equilibrium and Poisson income process. Easy extensions I won't discuss (see *https://benjaminmoll.com/codes/*)
  - Aiyagari model with productive capital
  - More general stochastic processes for income e.g. diffusions
  - General equilibrium

Hugget Model

Households are heterogeneous in their wealth a and income y, solve

$$\max_{\{c_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u\left(c_t\right) dt$$

$$\dot{a}_t = z_t + r a_t - c_t$$

$$z_t \in \{z_1, z_2\} \text{ Poisson income process with intensities } \lambda_1, \lambda_2$$

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

- c_t : consumption
- u : utility function, $u' > 0, u'' < 0$
- ρ : discount rate
- r : interest rate
- $\underline{a} \geq -y_1/r$ if $r > 0$: borrowing limit e.g. if $\underline{a} = 0$, can only save

The system of equations to solve

Equations to solve numerically

$$\rho v_1(a) = \max_c u(c) + v_1'(a)\left(z_1 + ra - c\right) + \lambda_1\left(v_2(a) - v_1(a)\right)$$

$$\rho v_2(a) = \max_c u(c) + v_2'(a)\left(z_2 + ra - c\right) + \lambda_2\left(v_1(a) - v_2(a)\right)$$

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

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

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

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

For derivations, see appendix B of Achdou et al. (2017):
https://benjaminmoll.com/wp-content/uploads/2019/07/HACT_appendix.pdf

I. HJB

Intertemporal consumption-saving problem:

$$\rho v_{i,j} = \max_c u\left(c_{i,j}\right) + v'_{i,j}\left(z_j + r a_i - c_{i,j}\right) + \lambda_j\left(v_{i,-j} - v_{i,j}\right), \quad j = 1, 2$$

- Get rid of the $\max$ operator by substituting for optimal c from the FOC (consumption policy function): $u'(c) = v'(a) \implies c = (u')^{-1}(v'(a))$:

$$\rho v_{i,j} = u\left(c_{i,j}\right) + v'_{i,j}\left(z_j + r a_i - c_{i,j}\right) + \lambda_j\left(v_{i,-j} - v_{i,j}\right), \quad j = 1, 2$$

- Discretize the asset grid and compute $v'(a)$ using the upwind scheme
- I'll go through these steps in the code $\iff$ slides
- Later we'll see that the borrowing constraint is easily accommodated while coding this up

Calculating $v'(a)$

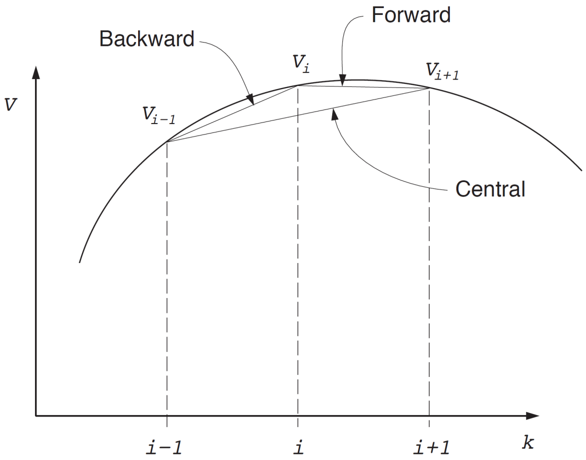
$$v'(k_i) \approx \frac{v_i - v_{i-1}}{\Delta k} = v'_{i,B} \quad \text{backward difference}$$

$$v'(k_i) \approx \frac{v_{i+1} - v_i}{\Delta k} = v'_{i,F} \quad \text{forward difference}$$

$$v'(k_i) \approx \frac{v_{i+1} - v_{i-1}}{2\Delta k} = v'_{i,C} \quad \text{central difference}$$

[See `dVf, dVb` in code]

Calculating $v'(a)$



Which difference to use: upwind scheme

At a given level of a, look at the savings policy function.

- $v'_{ij,F}$ whenever savings > 0 (drift of state variable positive)
- $v'_{ij,B}$ whenever savings < 0 (drift of state variable negative)
- In our example, the drift variable is just savings (obtain from savings policy function):

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

- Approximate derivative as follows

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

  where $\mathbf{1}_{\{\cdot\}}$ is indicator function, and $\bar{v}'_{ij} = u'\left(z_j + ra_i\right)$ (stay put)
- Since v is concave, $v'_{ij,F} < v'_{ij,B}$ (see figure) $\Rightarrow s_{ij,F} < s_{ij,B}$. So will not encounter problematic case where $s_{ij,F}$ tells you to go forward and $s_{ij,B}$ tells you go go backward.

[See `dV_Upwind` in code], ▸ State constraint in upwinding

Iterative algorithm to solve HJB: implicit method

$$\rho v_{i,j} = u\left(c_{i,j}\right) + v'_{i,j}\left(z_j + r a_i - c_{i,j}\right) + \lambda_j\left(v_{i,-j} - v_{i,j}\right), \quad j = 1,2$$

- Start with a guess $v_j^0 = \left(v_{1,j}^0, \ldots, v_{I,j}^0\right), j = 1,2$ and then updates $v_j^n, n = 1, \ldots$ in each step of the iteration.
- Each step n involves solving a linear system of equations

$$\begin{aligned}
\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta} + \rho v_{i,j}^{n+1} &= u\left(c_{i,j}^n\right) + \left(v_{i,j}^{n+1}\right)'\left(z_j + r a_i - c_{i,j}^n\right) + \lambda_j\left(v_{i,-j}^{n+1} - v_{i,j}^{n+1}\right) \\
&= u\left(c_{i,j}^n\right) + \frac{v_{i+1,j}^{n+1} - v_{i,j}^{n+1}}{\Delta a}\left(s_{i,j,F}^n\right)^+ + \frac{v_{i,j}^{n+1} - v_{i-1,j}^{n+1}}{\Delta a}\left(s_{i,j,B}^n\right)^- \\
&+ \lambda_j\left[v_{i,-j}^{n+1} - v_{i,j}^{n+1}\right]
\end{aligned}$$

HJB in vector form

Collect terms on the RHS:

$$\frac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta} + \rho v_{i,j}^{n+1} = u\left(c_{i,j}^{n}\right) + v_{i-1,j}^{n+1} x_{i,j} + v_{i,j}^{n+1} y_{i,j} + v_{i+1,j}^{n+1} z_{i,j} + v_{i,-j}^{n+1} \lambda_j \quad \text{where}$$

$$x_{i,j} = -\frac{\left(s_{i,j,B}^{n}\right)^{-}}{\Delta a}$$

$$y_{i,j} = -\frac{\left(s_{i,j,F}^{n}\right)^{+}}{\Delta a} + \frac{\left(s_{i,j,B}^{n}\right)^{-}}{\Delta a} - \lambda_j$$

$$z_{i,j} = \frac{\left(s_{i,j,F}^{n}\right)^{+}}{\Delta a}$$

This is a system of $2 \times I$ linear equations which is solved in each step n. It can be written in matrix notation as:

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

HJB in vector form

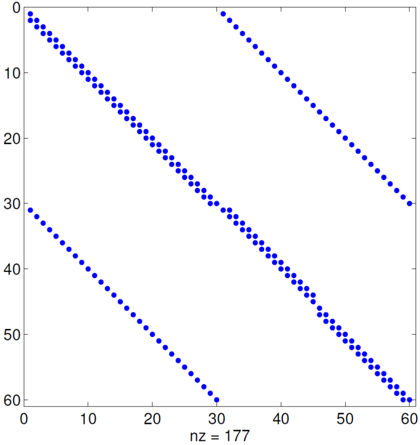
$$\frac{1}{\Delta}\left(v^{n+1} - v^{n}\right) + \rho v^{n+1} =$$

$$\begin{bmatrix} u\left(c^n_{1,1}\right) \\ \vdots \\ \vdots \\ u\left(c^n_{I,1}\right) \\ u\left(c^n_{1,2}\right) \\ \vdots \\ \vdots \\ u\left(c^n_{I,2}\right) \end{bmatrix} + \begin{bmatrix} y_{1,1} & z_{1,1} & 0 & \cdots & 0 & \lambda_1 & 0 & 0 & \cdots & 0 \\ x_{2,1} & y_{2,1} & z_{2,1} & 0 & \cdots & 0 & \lambda_1 & 0 & 0 & \cdots \\ 0 & x_{3,1} & y_{3,1} & z_{3,1} & 0 & \cdots & 0 & \lambda_1 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & x_{I,1} & y_{I,1} & 0 & 0 & 0 & 0 & \lambda_1 \\ \lambda_2 & 0 & 0 & 0 & 0 & y_{1,2} & z_{1,2} & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & x_{2,2} & y_{2,2} & z_{2,2} & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 & 0 & x_{3,2} & y_{3,2} & z_{3,2} & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & \cdots & 0 & \lambda_2 & 0 & \cdots & 0 & x_{I,2} & y_{I,2} \end{bmatrix} \begin{bmatrix} v_1^{n+1} \\ v_2^{n+1} \\ v_3^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ v_I^{n+1} \end{bmatrix}$$

[See *X, Y, Z* in code and use *spdiags* to construct A. Solve this system of equations using the *mldivide* command in *MATLAB* (*https://www.mathworks.com/help/matlab/ref/mldivide.html*)

spy(A)

- $\mathbf{A}^{\mathbf{n}}$ 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 $\mathbf{A}^{\mathbf{n}}$ summarizing the corresponding Poisson intensities.
- Note that $\mathbf{A}^{\mathbf{n}}$ satisfies all the properties a Poisson transition matrix needs to satisfy. In particular, all rows sum to zero (would mean that the state remains fixed over time).



HJB in vector form

- Each step n involves solving a linear system of the form

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

$$\underbrace{\left[\left(\rho + \frac{1}{\Delta}\right) I - \mathbf{A}_n\right]}_{\mathbf{B}^n} v^{n+1} = \underbrace{u + \frac{1}{\Delta} v^{n}}_{b^n}$$

$$\mathbf{B}^n v^{n+1} = b^n$$

[See `B, b, A, Aswitch` in code]

Summary of Algorithm

Guess $v_{i,j}^0, i = 1, \ldots, I, j = 1, 2$ and for $n = 0, 1, 2, \ldots$ follow

1. Compute $\left(v_{i,j}^n\right)'$ using the upwind scheme.
2. Compute c^n from $c_{i,j}^n = (u')^{-1}\left(v_{i,j}^n\right)'$ using the $v_{i,j}^n$ computed in step 1.
3. Find v^{n+1} by solving the system of equations.
4. If v^{n+1} is close enough to v^n : stop. Otherwise, go to step 1 .

Extension: non-convexities

Many applications in macro development

- Convex-concave production function (Skiba 1978)
- Entrepreneurship/occupation choice
- Poverty traps

Butterfly production function (Skiba 1978)

Consider the planning problem in the neoclassical growth model:

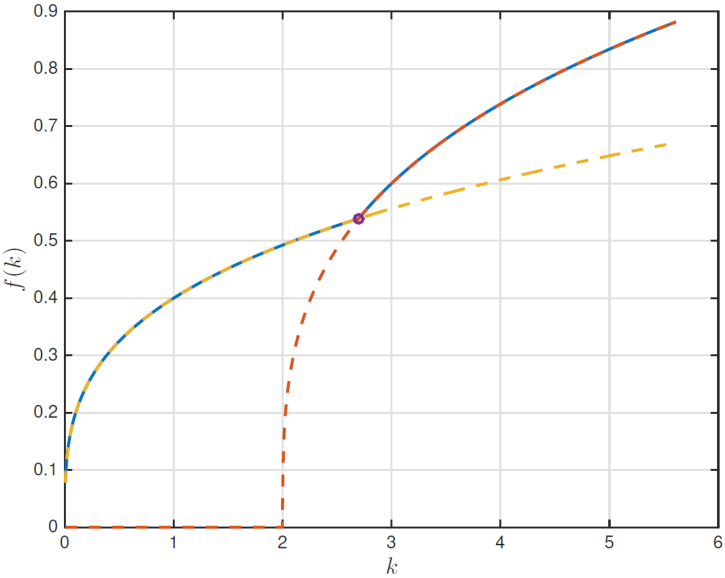
$$v(k_0) = \max_{\{c(t)\}_{t\geq 0}} \int_0^\infty e^{-\rho t} u(c(t))dt \quad \text{s.t.}$$
$$\dot{k}(t) = f(k(t)) - \delta k(t) - c(t), \quad k(0) = k_0$$

But now assume that the production function is not strictly concave everywhere. In particular assume that

$$f(k) = \max\{f_L(k), f_H(k)\}$$
$$f_L(k) = A_L k^\alpha$$
$$f_H(k) = A_H \left((k-\kappa)^+\right)^\alpha$$

with $\kappa > 0$ and $A_H > A_L$.

Planner has costless access to a bad technology with productivity A_L, and can upgrade it to a good technology with productivity $A_H > A_L$ but only by paying a per-period fixed cost κ.


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
$f(k)$
k

Upwinding with non-concave value function

s_F s_B	>0 >0	<0 <0	<0 >0	>0 <0
	use v'_F	use v'_B	use $\bar{v}'$	Ruled out by concavity of v
	$\mathbf{1}^{unique}$			$\mathbf{1}^{both}$

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

- The forward and backward Hamiltonians $H_{i,j,F} := u\left(c_{i,j,F}\right) + v'_{i,j,F}s_{i,j,F}$ and similarly $H_{i,j,B}$ are used as “tie breakers”
- 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. ▸ Viscosity solution

Occupational Choice

- See *https://benjaminmoll.com/wp-content/uploads/2020/06/entrepreneurs_numerical.pdf* and code *http://benjaminmoll.com/wp-content/uploads/2020/06/entrepreneurs.m*

$$\rho v_1(a) = \max_c u(c) + v_1'(a)\left(z_1 + ra - c\right) + \lambda_1 \left(v_2(a) - v_1(a)\right)$$

$$\rho v_2(a) = \max_c u(c) + v_2'(a)\left(z_2 + ra - c\right) + \lambda_2 \left(v_1(a) - v_2(a)\right)$$

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

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

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

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

Kolmogorov forward equation

- Given the wealth distribution today, savings decisions and the random evolution of income, what is the wealth distribution tomorrow?

$$0 = -\frac{d}{da}[s_1(a)g_1(a)] - \lambda_1 g_1(a) + \lambda_2 g_2(a)$$
$$0 = -\frac{d}{da}[s_2(a)g_2(a)] - \lambda_2 g_2(a) + \lambda_1 g_1(a)$$

- Discretized version is simply

$$0 = \mathbf{A}(\mathbf{v})^\top \mathbf{g}$$

- This is an eigenvalue problem.
- get KF for free, one more reason for using implicit scheme
- Why transpose: operator in (HJB) is ”adjoint” of operator in (KF), i.e. an infinite-dimensional analogue of matrix transpose

[see *http://benjaminmoll.com/wp-content/uploads/2020/06/huggett_partialeq.m*]

Appendix

Where did the borrowing constraint go?

- State constraint is $a \geq \underline{a}$.
- The first order condition $u'\left(c_j(\underline{a})\right) = v_j'(\underline{a})$ 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

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

Intuition: at the borrowing constraint, i.e. at $a = \underline{a}$, the agent is always better forgoing some consumption in order to save and increase her wealth. At $\underline{a}$, the marginal value of increasing wealth just a little bit (i.e. $v_j'(\underline{a})$ exceeds the marginal utility of consumption when the agent's entire net worth is devoted to consumption (i.e. when $c = z_j + r\underline{a}$. Back

State constraint and upwinding

1. At the lower end $a = a_1$
   - State constraint is enforced by setting $v'_{1,j,B} = u'(z_j + ra_1)$
   - 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.”
2. 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 $v'_{I,j,F} = u'(z_j + ra_I)$

Back

Viscosity solution

Question: If the value function has kinks because of the non-convexities $\implies v'$ doesn't exist. Then what does it mean for v to satisfy the HJB equation which clearly involves this derivative?

Answer: replace $v'(k)$ at point where it does not exist (because of kink in $v(k)$) with the derivative ϕ' of a smooth function ϕ (a "test function") that "touches v", and to define a viscosity solution as a function v that satisfies an alternative equation that features ϕ' instead of v'.

For details see *https://benjaminmoll.com/wp-content/uploads/2020/02/viscosity_for_dummies.pdf*

Back