A Set of Tools for Computing Recursive Equilibria

Simon Mongey

November 17, 2015

Contents

1 Introduction

- In this note I take a canonical stationary recursive competitive equilibrium model and show how to solve it using a number of techniques combined.
- The main features of the approach include: (i) a collocation approach to solving the value function problem, (ii) a discrete approximation to the law of motion of the distribution of agents over states using Young's (2010) method, (iii) a parallelised bisection method finding the equilibrium variable.
- In the baseline model the state variable of the agent is two dimensional $s = (x, z)$ where x evolves endogenously due to the decisions of the agent and z follows a first order stochastic process $z' \sim \Gamma(z, z')$. The baseline model considers a discrete approximation to Γ using a transition matrix $P(z, z')$ over discrete states $\{z_1, \ldots, z_K\} = \mathcal{Z}$.
- I then extend this model to include (i) discrete choices, (ii) continuously distributed z, (iii) iid shocks to z, (iv) *iid* shocks to x.

- **Note**: *The contents of these notes draw heavily on the tools in Miranda & Fackler's 'Applied Computational Economics and Finance', in particular I use a number of routines from their COMPECON toolbox which can be found at http://www4.ncsu.edu/~pfackler/compecon/toolbox.html*

2 Baseline model

- **Agent's problem**
  - The agent takes the price p as given and solves the following recursive problem given their states (x, z)

$$V(x,z) = \max_{x' \in B(x,z)} F(x,z,x',p) + \beta \sum_{z' \in \mathcal{Z}} P(z,z')V(x',z')$$

  - The solution to this problem is a value function $V(x, z)$ and a policy function $x'(x, z)$.The
- **Equilibrium**
  - Let $\lambda(x, z)$ be the distribution of firms in steady-state then the cumulativeΛ satisfies

$$\Lambda(x', z') = \int_{X \times Z} \mathbf{1}[x'(x,z) \leq x']P(z,z')d\lambda(x,z)$$

  - Alternatively define an operator Q that satisfies the above and maps λ's into λ such that

$$\lambda = Q(\lambda)$$

  - Market clearing requires that

$$\bar{x} = \int_{X \times Z} x'(x,z)d\lambda(x,z)$$

- A small amount of structure yields a unique equilibrium:
  - Regularity conditions on P: Satisfies monotonicity and Feller property (see: Stokey and Lucas)
  - Monotonicity in prices: F is increasing in p so that $x'(x, z)$ is also increasing in p.
  - The set $B(x, z)$ is compact, continuous and convex.
  - The function F, given (x, z) is concave in x'

3 Solving the Bellman equation using collocation

- The collocation approach requires approximating the value function (an object which lives in an infinite dimensional space) with a function defined by a finite set of coefficients (which lives in a finite dimensional space).
- The first question to ask ourselves is what function to approximate.
- Given that we have to solve the optimization problem which requires taking the max over $x' \in B(x, z)$, if we were to approximate V directly then we would have to compute the expectation of V in every loop of our optimization routine for x'.

- This is a useful idea - Approximate the expected continuation value
- The first step is then to re-write the Bellman equation in terms of V^e and V as follows

$$\begin{aligned} V(x,z) &= \max_{x' \in B(x,z)} F(x,z,x',p) + \beta V_e(x',z) \\ V_e(x,z) &= \sum_{z' \in \mathcal{Z}} P(z,z') V(x,z') \end{aligned}$$

- This establishes a mapping $(V, V_e) \to (V', V_e')$. We could then approximate V and V_e separately and solve this system, and in some cases that's the best way to go, I'll describe these cases later.
  - Note that this simple step will drastically speed up any kind of iterative routine, even if we were to stop here and just use value function iteration on a grid.
- Alternatively we could just solve for the expected value function

$$V_e(x,z) = \sum_{z' \in \mathcal{Z}} P(z,z') \left(\max_{x' \in B(x,z')} F(x,z',x',p) + \beta V_e(x',z') \right)$$

- I'll call these **Case I** and **Case II** respectively and for now proceed with **Case I**.
- We'll write this system in notation which will be useful for collocation and computation.
- Let's call a state $s = (x, z)$.
- The first step in approximating the value functions is choosing nodes for collocation.
- Choose a set of N_X collocation notes for x and call this $(x_1, \ldots, x_{N_x})' = \mathbf{x}$
- Choose your favourite method to discretize Γ into the $N_Z \times N_Z$ transition matrix P along with a grid for z and call this $(z_1, \ldots, z_{N_z})' = \mathbf{z}$.
- Now choose an approximant. Throughout I'll assume that we're using splines. I'll comment on why I prefer these later on.
- Given our two sets of collocation nodes in each dimension we generate a set of collocation nodes s where

$$s = [\mathbf{i}_{N_Z} \otimes \mathbf{x}, \mathbf{k} \otimes \mathbf{i}_{N_X}]$$

  so that s is $N \times 2$ where $N = N_X \times N_Z$
- From now on anything without a subscript will refer to a vector and we will try to write most things in vector notation.
- Writing our system of functional equations in this notation

$$\begin{aligned} V(s_i) &= \max_{x' \in B(s_i)} F(s_i,x',p) + \beta V_e([x', s_{i2}]) \\ V_e(s_i) &= \sum_{k=1}^{N_Z} P(z,z') V([s_{i,1}, k]) \end{aligned}$$

- We now replace the functions we want to approximate with interpolants so that

$$V(s_i) = \sum_{j=1}^{N} \phi(s_i) c_j, \qquad V^e(s_i) = \sum_{j=1}^{N} \phi(s_i) c_j^e$$

  where ϕ is a basis function and $c = (c_1, \ldots, c_N)'$ is a vector of coefficients.

- **Important** - There are N coefficients and N points in s, we'll see why now.

- If we substitute these into our system of functional equations we have

$$\begin{aligned}
\sum_{j=1}^{N} \phi(s_i) c_j &= \max_{x' \in B(s_i)} F(s_i, x', p) + \beta \sum_{j=1}^{N} \phi([x', s_{i2}]) c_j^e, \quad i = 1, \ldots, N \\
\sum_{j=1}^{N} \phi(s_i) c_j^e &= \sum_{k=1}^{N_Z} P(z, z') \sum_{j=1}^{N} \phi([s_{i,1}, k]) c_j^e, \quad i = 1, \ldots, N
\end{aligned}$$

- **Key -** This is a $2N$ system of equations (one for each s_i and each main equation) in $2N$ unknowns $[c; c^e]$.

- Crucially, this system is **linear** in the coefficients, which will allow us to easily compute Jacobians and use Newton-methods.

- Notationally the next step is to stack this system.

- First note that we can stack $V(s_i) = \sum_{j=1}^{N} \phi(s_i) c_j$ as

$$\mathbf{V}(s) = \Phi(s) c$$

  where $\Phi(s)$ is the basis matrix of the approximant evaluated at s

- We can also define a vectorized $\mathbf{F}(s, x', p)$ which takes as arguments vectors s and x' and scalar p and gives a vectorized output. Similarly create a vectorized set function $\mathbf{B}$

- In this way we can stack the first equation

$$\Phi(s) c = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x', p) + \beta \Phi([x', s_2]) c^e$$

- We can also stack the second equation. Note that since $s = [\mathbf{i}_{N_Z} \otimes \mathbf{x}, \mathbf{k} \otimes \mathbf{i}_{N_X}]$, then we can write the second equation as

$$\Phi(s) c^e = (P \otimes \mathbf{I}_{N_X}) \Phi(s) c$$

- This may look a little odd, let's confirm what's going on in the case of $N_X = 3$ and $N_Z = 2$

$$V^e\left(\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_1 & 2 \\ x_2 & 2 \\ x_3 & 2 \end{bmatrix}\right) = \begin{bmatrix} p_{11} & 0 & 0 & p_{12} & 0 & 0 \\ 0 & p_{11} & 0 & 0 & p_{12} & 0 \\ 0 & 0 & p_{11} & 0 & 0 & p_{12} \\ p_{21} & 0 & 0 & p_{22} & 0 & 0 \\ 0 & p_{21} & 0 & 0 & p_{22} & 0 \\ 0 & 0 & p_{21} & 0 & 0 & p_{22} \end{bmatrix} V\left(\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_1 & 2 \\ x_2 & 2 \\ x_3 & 2 \end{bmatrix}\right)$$

$$V^e(s) = (P \otimes \mathbf{I}_{N_X})V(s)$$

- **System to solve -** So our $2N$-equation system now can be written neatly as

$$\begin{aligned} \Phi(s)c &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x', p) + \beta\Phi([x', s_2])c^e \\ \Phi(s)c^e &= (P \otimes \mathbf{I}_{N_X})\Phi(s)c \end{aligned}$$

- So what else do we need to know how to do to solve this system? We need to be able to compute the max given a c^e. We need to have a strategy for solving the system for (c, c^e). We need to know how to compute basis functions.

- Let's start with the last. Using the COMPECON toolbox we can use the following

$$\begin{aligned} \Phi(s) &= funbas(fspace, s) \\ \Phi([x', s_2]) &= funbas(fspace, [x', s_2]) \end{aligned}$$

  where $fspace$ is the function space for the approximant. See the code to see how this is set-up. Funbas creates sparse matrices taking vectors as arguments. Note that we use a linear interpolant for $\mathbf{k}$, which is fine since it always remains on the grid points.

- What about the max. There are two approaches here:

  1. **Vectorized golden search**
     - The COMPECON command *goldenx* is a vectorized version of golden search. Its very fast and highly robust to unpleasant features of the objective function which might arise at steps along the way to the solution. Here we would have

$$\begin{aligned} [\underline{x}', \bar{x}'] &= \mathbf{B}(s) \\ obj &= @(x')\mathbf{F}(s, x', p) + \beta\Phi([x', s_2])c^e \\ x' &= goldenx(obj, \underline{x}', \bar{x}') \end{aligned}$$

  2. **Newton-Method**
     - If using a spline of order greater than two, then a Newton method can be used to solve the first-order condition. If a cubic spline, for example, is used then we can define derivatives

easily

$$\begin{aligned}\mathbf{f}(s) &= \Phi(s)c \\ \nabla\mathbf{f}(s) &= \Phi_{[1,0]}(s)c\end{aligned}$$

which are also easily computed using COMPECON with

$$\Phi_1(s) = funbas(fspace, s, [1, 0])$$

given that we are computing the derivative of order 1 in the first dimension and order 0 in the second. The N-first-order conditions, stacked are

$$\mathbf{F}_x(s, x', p) + \beta\Phi_{[1,0]}([x', s_2])c^e = 0$$

Now given a starting guess x'_n we can update as usual

$$x_{n+1} = x_n - \left(\mathbf{F}_{xx}(s, x_n, p) + \beta\Phi_{[2,0]}([x_n, s_2])c\right)^{-1}\left(\mathbf{F}_x(s, x', p) + \beta\Phi_{[2,0]}([x_n, s_2])c^e\right)$$

All the objects here are easily computed.

- Now let's consider solving for c and c^e . Since there is an optimization step then the solution strategy is going to have to be iterative. That is we start of with (c_0, c_0^e) and then by some scheme choose a new (c_1, c_1^e) and so on until convergence.
- Given a guess of c, c^e the first step will be to solve the maximization problem for $x'(s)$ which we substitute in throughout.
- There are two ways to proceed

1. **Bellman iteration**

   - Given a guess (c_k, c_k^e) we simply iterate as we would in value function iteration on the system

   $$\begin{aligned}\Phi(s)c_{k+1} &= \mathbf{F}(s, x'(s), p) + \beta\Phi([x'(s), s_2])c_k^e \\ \Phi(s)c_{k+1}^e &= (P \otimes \mathbf{I}_{N_X})\Phi(s)c_k\end{aligned}$$

   - So we first compute the RHS and then compute c_{k+1} and c_{k+1}^e simply by inverting $\Phi(s)$

   $$\begin{aligned}c_{k+1} &= \Phi(s)^{-1}\left(\mathbf{F}(s, x'(s), p) + \beta\Phi([x'(s), s_2])c_k^e\right) \\ c_{k+1}^e &= \Phi(s)^{-1}\left((P \otimes \mathbf{I}_{N_X})\Phi(s)c_k\right)\end{aligned}$$

   - Given an intial guess I find it useful to do this type of iteration for a couple of steps to give the approximant the correct shape.
   - **Debugging** - Doing 'Bellman' iterations by themselves should lead to convergence of the coefficients, even if it is very slow. This is useful for debugging to check that the problem does define a contraction.

2. **Newton iteration**

   - Here we view the problem as root-finding rather than using the contraction property.
   - We can re-write the system as a zero-system which is linear in c and c^e

$$\begin{aligned}\mathbf{g}_1(c, c^e) &= \Phi(s)c - (\mathbf{F}(s, x'(s), p) + \beta\Phi([x'(s), s_2])c^e) \\ \mathbf{g}_2(c, c^e) &= \Phi(s)c^e - (P \otimes \mathbf{I}_{N_X})\Phi(s)c\end{aligned}$$

   - The Jacobian of this system is

$$\begin{aligned}\mathbf{D}(c, c^e) &= \begin{bmatrix} \frac{\partial \mathbf{g}_1(c,c^e)}{\partial c} & \frac{\partial \mathbf{g}_1(c,c^e)}{\partial c^e} \\ \frac{\partial \mathbf{g}_2(c,c^e)}{\partial c} & \frac{\partial \mathbf{g}_2(c,c^e)}{\partial c^e} \end{bmatrix} \\ \mathbf{D}(c, c^e) &= \begin{bmatrix} \Phi(s) & -\beta\Phi([x'(s), s_2]) \\ -(P \otimes \mathbf{I}_{N_X})\Phi(s) & \Phi(s) \end{bmatrix}\end{aligned}$$

   - Note how $\frac{\partial \mathbf{g}_1(c,c^e)}{\partial c^e}$ uses the Envelope Theorem. Computing the derivative of the objective function with respect to coefficients (which are parameters of the objective function which computing $\max_{x'}$) can be done using $x'(s)$.
   - This suggests the updating scheme

$$\begin{bmatrix} c_{k+1} \\ c^e_{k+1} \end{bmatrix} = \begin{bmatrix} c_k \\ c^e_k \end{bmatrix} - \mathbf{D}(c_k, c^e_k)^{-1} \begin{bmatrix} \mathbf{g}_1(c_k, c^e_k) \\ \mathbf{g}_2(c_k, c^e_k) \end{bmatrix}$$

- It turns out that Newton-iterations are of order of magnitudes faster at reaching convergence than the Bellman iterations and given the way that the collocation approach has set-up the problem as linear in the coefficient vectors then the Jacobians can be computed *exactly* and in closed form.

3.1 Storage

- So what's left that takes time to compute?
- Let's return to our system of equations that we aim to solve using collocation

$$\begin{aligned}\Phi(s)c &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x', p) + \beta\Phi([x', s_2])c^e \\ \Phi(s)c^e &= (P \otimes \mathbf{I}_{N_X})\Phi(s)c\end{aligned}$$

- So it looks like what we'll be doing a lot of is computing basis matrices. But note that a number of these can be computed once and stored at the start of the computation. In fact the following can be stored: $\Phi(s)$ and $(P \otimes \mathbf{I}_{N_X})\Phi(s)$. Note that this makes total sense, taking expectations, in this discretized framework should be able to be represented by multiplying by a matrix. And it is.
- We can also do a little better. The way that basis matrices are made is by taking *tensor products.* Specifically if $s = (a, b, c)$ and I have splines in each variable then

$$\Phi([a, b, c]) = \Phi_c(c)\Phi_b(b)\Phi_a(a)$$

where Φ_j is $N_j \times N_j$. From COMPECON the C+ written function *dprod* handles this so

$$\Phi([a,b,c]) = dprod(\Phi_c(c), dprod(\Phi_b(b), \Phi_a(a)))$$

- Returning to our problem note that we can compute

$$\Phi([x', s_2]) = dprod(\Phi_2(s_2), \Phi_1(x'))$$

where $\Phi_2(s_2)$ is always the same. Therefore if we store this basis matrix we halve the number of basis matrices that need to be computed for each guess of x' by the solver.

- Inituitively, the transitions regarding the exogenous state are always the same. And we simply impose this everywhere.

3.2 Computing the stationary distribution

- We first choose a very fine grid on which we wish to compute an approximation of the stationary distribution. We choose N_x^d points (d for 'distribution')
- Since z is already discretized we only have to choose a finer grid for x.
- Call this $(x_1, \ldots, x_{N_x^d})' = \mathbf{x}^d$ and for z and call this $(z_1, \ldots, z_{N_z})' = \mathbf{z}$.
- We then have this set of points on which we discretive the distribution s^d

$$s_d = [\mathbf{i}_{N_Z} \otimes \mathbf{x}^d, \mathbf{k} \otimes \mathbf{i}_{N_X^d}]$$

- Given that we have solved for the approximation of the expected continuation value we can now solve for policies on *any* point of the state space. Taking some (x, z) we have

$$x'(x,z) = \arg \max_{x' \in B(x,z)} F(x,z,x',p) + \beta \Phi([x',z]) c^e$$

- Again we can stack everything and compute

$$x'_d = \arg \max_{x' \in \mathbf{B}(s_d)} \mathbf{F}(s_d, x', p) + \beta \Phi([x', s_{d,2}]) c^e$$

- Now we will use the policy x'_d and the grid x_d to approximate the law-of-motion of the distribution $\lambda(x,z)$ of agents over (x,z).
- Define $L(s_i^d)$ as the approximation of λ on point s_i^d of s^d. Then $L(s_i^d)$ can be stacked into a $N^d = N_X^d \times N_Z$ vector.
- We seek to write-down a law of motion for L on the basis of the policies $x'(s_d)$ and the exogenous transitions $P(z, z')$.
- From here I'll drop the additional d notation.

- We construct a $N \times N$ matrix Q such that

$$L = Q'L$$

- We construct Q by taking the tensor product of two matrices Q_X and Q_Z. The $N \times N_X$ matrix Q_X uses $x'(s)$ to shift mass in terms of the endogenous transitions from a point s_i to a point in the x-dimension, x_j. The $N \times N_Z$ matrix Q_Z uses $P(z, z')$ to then shift that mass around in terms of the exogenous transitions.

- Elements of Q_X are given by

$$Q_X(s_i, x_j) = \left[\mathbf{1}_{x'(s_i) \in [x_{j-1}, x_j]} \frac{x'(s_i) - x_j}{x_j - x_{j-1}} + \mathbf{1}_{x'(s_i) \in [x_j, x_{j+1}]} \frac{x_{j+1} - x'(s_i)}{x_{j+1} - x_j} \right]$$

- We read this as follows. Take a point s_i and the policy $x'(s_i)$. Suppose $x'(s_i) \in [x_{j-1}, x_j]$ and is 3/4 of the way across the interval, that is $\frac{x'(s_i) - x_j}{x_j - x_{j-1}} = \frac{3}{4}$, so close to x_j. Then 3/4 of the mass is assigned to x_j and writing out $Q_X(s_i, x_{j-1})$ would reveal that 1/4 of the mass is assigned to x_{j-1}.

- This is particularly useful in that *aggregates* will be unbiased. For example if the true process has a mass of one at 5 and I have grid points at 1 and 10. Then half the mass will be assigned to the point at one and half to the point at 10. When using the discrete approximation to compute aggregates I'll compute $(1/2) \times 1 + (1/2) \times 10 = 5$ which is the true aggregate.

- Rather neatly, we can observe that Q_X is simply the basis matrix of a linear interpolant. That is we can compute it very simply by

$$\begin{aligned} fspaceerg &= fundef\left(\{'spli', 0, 1, \mathbf{x}^d\}\right) \\ Q_X &= funbas(fspaceerg, x'(s_d)) \end{aligned}$$

- The matrix Q_Z is simple to compute since the transitions are independent of x.

$$Q_Z = P \otimes \mathbf{i}_{N_X^f}$$

- We then compute Q simply by

$$Q = dprod(Q_X, Q_P)$$

- The stationary distribution can either be found by proposing an L^k and then iterating using Q to L^{k+1} until convergence (which is recommended for **debugging**). Or simply by solving $L = Q'L$ by taking L equal to the eigen vector of Q' corresponding to the largest eigen value.

- **Debugging** - Check the proposed solution is correct by confirming that it is a fixed point of $L = Q'L$.

3.2.1 Extension - multiple states

- What if we had two endogenous states and one exogenous state?

- Suppose $s = (x, a, z)$ where both x and a are chosen or in some way endogenous.

- Let $x'(x,a,z)$ and $a'(x,a,z)$ be the optimal policies.
- Then we can simply compute Q by
$$Q = Q_X \odot Q_A \odot Q_Z$$
where Q_A and Q_X are computed as was Q_X in the case of two states.
- This makes perfect sense. Q_A looks after the transitions on the A dimension, Q_X on the X dimension and Q_Z the exogenous transitions.

3.3 Aggregation and equilibrium

- In our case the aggregate equilibrium variable we wish to compute is
$$X = \int x'(x,z) d\lambda(x,z)$$
- Given our approximation this can be computed as
$$X = \sum_{i=1}^{N^d} x'(s_i) L(s_i) = L^T x'(s)$$
- We can now compute X for any p. Denote this $X(p)$.
- An equilibrium is one in which $X(p) = \bar{X}$.
- Since we know that $X(p)$ is monontone and increasing in p we can proceed by a bisection method.
- Choose $\underline{p}^0$ and $\bar{p}^0$ and take $p^0 = \frac{1}{2}(\underline{p}^0 + \bar{p}^0)$. Compute $X(p)$. If $X(p) > \bar{X}$, then set $\bar{p}^1 = p$ and $\underline{p}^1 = \underline{p}^0$. If $X(p) < p$, then set $\underline{p}^1 = p$ and $\bar{p}^1 = \bar{p}^0$. Continue until $|X(p) - \bar{X}| < \varepsilon_X$.
- **Bisection with parallel processing.**
  - If parallel processing with 3 or more CPUs is available to you then a very easy and much faster way to solve the problem is as follows.
  - Set $\underline{p}^0$ and $\bar{p}^0$ as above.
  - Let M be the number of processors at your disposal.
  - Let p_{grid} be an evenly spaced grid of M points starting at $\underline{p}$ and ending at $\bar{p}$.
  - Now in parallel compute X on p_{grid}. This can be accomplished easily in MATLAB using $parfor$ and writing a function $eq = solveeq(p, glob, param)$ which outputs as part of the structure eq the value $X(p)$.

$$\begin{aligned} parfor \quad & i = 1 : M \\ & p = pgrid(i); \\ & eq = solveeq(p, glob, param); \\ & X(i) = eq \cdot X; \\ end & \end{aligned}$$

– Find the points on the grid for which $X(p) - \bar{X}$ changes from negative to positive and set these as the new lower and upper bounds. If this does not occur then widen bounds in correct direction.

– On the next, and further iterations, indexed k, the grid can be computed between $\underline{p}^k$ and $\bar{p}^k$ but with $M+2$ points, then removing the first and last points, since $X(\underline{p}^k)$ and $X(\bar{p}^k)$ have already been computed and stored during iteration $k-1$.

4 Extensions

4.1 Case II - Solving the expected value function only

- If we were solving only for the expected value function then the system we aim to solve would be

$$\Phi(s)c^e = (P \otimes \mathbf{I}_{N_X}) \left(\max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x', p) + \beta \Phi([x'(s), s_2])c^e \right)$$

with Jacobian under $x'(s)$ of

$$\mathbf{D}(c^e) = \Phi(s) - \beta(P \otimes \mathbf{I}_{N_X})\Phi([x'(s), s_2])c^e$$

- We can then proceed to Newton-iterations exactly as above.

4.2 Case with discrete choices

- Consider a case of combined continuous and discrete choice which can generally be written as follows

$$\begin{aligned} V_1(x,z) &= \max_{x' \in B_1(x,z)} F_1(x,z,x',p) + \beta \sum_{z' \in \mathcal{Z}} P(z,z') \max\{V_1(x',z'), V_2(x',z')\} \\ V_2(x,z) &= \max_{x' \in B_2(x,z)} F_2(x,z,x',p) + \beta \sum_{z' \in \mathcal{Z}} P(z,z') \max\{V_1(x',z'), V_2(x',z')\} \end{aligned}$$

- **Example** - At the beginning of each period a household decides whether to rent or own a house.

- Proceeding as above, approximating V_1, V_2 and the expected continuation value V_e, which is the same in both cases, we would have the following system

$$\begin{aligned} \Phi(s)c^1 &= \max_{x_1' \in \mathbf{B}_1(s)} \mathbf{F}(s, x_1', p) + \beta \Phi([x_1', s_2])c^e \\ \Phi(s)c^2 &= \max_{x_2' \in \mathbf{B}_1(s)} \mathbf{F}(s, x_2', p) + \beta \Phi([x_2', s_2])c^e \\ \Phi(s)c^e &= (P \otimes \mathbf{I}_{N_X}) \max\left\{\Phi(s)c^1, \Phi(s)c^2\right\} \end{aligned}$$

- Replacing the optimal policies $x_1'(s)$, $x_2'(s)$ and the discrete choice $I(s)$ where $I(s) = 1$ if choose 1 and 0 if choose 2 we have

$$\begin{aligned} \Phi(s)c^1 &= \mathbf{F}(s, x_1'(s), p) + \beta \Phi([x_1'(s), s_2])c^e \\ \Phi(s)c^2 &= \mathbf{F}(s, x_2'(s), p) + \beta \Phi([x_2'(s), s_2])c^e \\ \Phi(s)c^e &= (P \otimes \mathbf{I}_{N_X}) \left[I(s) \odot \Phi(s)c^1 + (1 - I(s)) \odot \Phi(s)c^2\right] \end{aligned}$$

where $\odot$ is a single-expansion which can be computed $dprod(I(s), \Phi(s))$.

- The Jacobian is then easily computed as

$$\mathbf{D}(c^1, c^2, c^e) = \begin{bmatrix} \Phi(s) & 0 & -\beta\Phi([x_1'(s), s_2]) \\ 0 & \Phi(s) & -\beta\Phi([x_2'(s), s_2]) \\ -(P \otimes \mathbf{I}_{N_X}) [I(s) \odot \Phi(s)] & -(P \otimes \mathbf{I}_{N_X}) \left[(1 - I(s)) \odot \Phi(s)c^2\right] & \Phi(s) \end{bmatrix}$$

- This again could be computed simply in terms of the expected value function, in which case our system would be

$$\Phi(s)c^e = (P \otimes \mathbf{I}_{N_X}) \max \left\{ \left(\max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x', p) + \beta\Phi([x'(s), s_2])c^e \right), \left(\max_{x_2' \in \mathbf{B}_1(s)} \mathbf{F}(s, x_2', p) + \beta\Phi([x_2', s_2])c^e \right) \right\}$$

with Jacobian

$$\mathbf{D}(c^e) = \Phi(s) - \beta(P \otimes \mathbf{I}_{N_X}) [I(s) \odot \Phi([x_1'(s), s_2]) + (1 - I(s)) \odot \Phi([x_2'(s), s_2])]$$

4.3 Case with different P's and / or regime switching

- Consider a case with different transition matrices for z and regime switching

$$\begin{aligned} V_1(x, z) &= \max_{x' \in B(x,z)} F(x, z, x', p) + \beta \sum_{z' \in \mathcal{Z}} P_1(z, z')(\theta_1 V_1(x', z') + (1 - \theta_1)V_2(x', z')) \\ V_2(x, z) &= \max_{x' \in B(x,z)} F(x, z, x', p) + \beta \sum_{z' \in \mathcal{Z}} P_2(z, z')(\theta_2 V_1(x', z') + (1 - \theta_2)V_2(x', z')) \end{aligned}$$

where $\theta_i \in [0, 1]$.

- **Example -** This nests a large set of models in which the economy can switch into regimes of high 'uncertainty' (imbodied in a more volatile P_2) with some state-dependent probability. See: Gourio.

- **Example -** At the begining of the period the worker becomes unemployed with probability $(1 - \theta_1)$ and employed with probability $(1 - \theta_2)$ and remains in each state with probability θ_i. When employed productivity evolves according to P_1 and when unemployed according to P_2.

- Proceeding as above, approximating V_1, V_2 and the **two** expected continuation value $V_{e,1}$ and $V_{e,2}$, which is the same in both cases, we would have the following system

$$\begin{aligned} \Phi(s)c^1 &= \max_{x_1' \in \mathbf{B}_1(s)} \mathbf{F}(s, x_1', p) + \beta\Phi([x_1', s_2])c^{e,1} \\ \Phi(s)c^2 &= \max_{x_2' \in \mathbf{B}_1(s)} \mathbf{F}(s, x_2', p) + \beta\Phi([x_2', s_2])c^{e,2} \\ \Phi(s)c^{e,1} &= (P_1 \otimes \mathbf{I}_{N_X})(\theta_1\Phi(s)c^1 + (1 - \theta_1)\Phi(s)c^2) \\ \Phi(s)c^{e,2} &= (P_2 \otimes \mathbf{I}_{N_X})(\theta_2\Phi(s)c^1 + (1 - \theta_2)\Phi(s)c^2) \end{aligned}$$

- In this case, to economize on the number of coefficients we're solving for and so crucially reducing the size of the Jacobian that must be inverted, we would definitely solve only for $c^{e,1}$ and $c^{e,2}$.

- When computing L there are now two additional states, the agent could be in state 1 or state 2.

- So L is now $2N \times 1$ and the $2N \times 2N$ matrix Q is given by

$$Q = \begin{bmatrix} \theta_1 Q_1 & (1-\theta_1)Q_1 \\ (1-\theta_2)Q_2 & \theta_2 Q_2 \end{bmatrix}$$

  where Q_1 and Q_2 are given exactly as above using the policies $x_i'(s)$ and transitions $P_i(z, z')$ for each $i \in \{1, 2\}$.

4.4 Additional *iid* shocks to x

- Suppose our basic model is as follows

$$V(x, z) = \max_{x' \in B(x,z)} F(x, z, x', p) + \beta \int_{\underline{\eta}}^{\bar{\eta}} \sum_{z' \in \mathcal{Z}} P(z, z') V(x' + \eta, z') dF(\eta)$$

- **Example** - This nests a model of a firm with stochastic productivity and *iid* capital destruction shocks.
- **Example -** Firm operating subject to *iid* cost shocks
- Again we split up the problem

$$\begin{aligned} V(x, z) &= \max_{x' \in B(x,z)} F(x, z, x', p) + \beta V_e(x', z) \\ V_e(x, z) &= \int_{\underline{\eta}}^{\bar{\eta}} \sum_{z' \in \mathcal{Z}} P(z, z') V(x' + \eta, z') dF(\eta) \end{aligned}$$

- Now we **approximate** the distribution for F using whatever our favourite method is. Maybe its a couple of points on a uniform. Maybe we're serious about it and $\eta \sim N(\mu_\eta, \sigma_\eta)$ so we use a quadrature method. Regardless, we emerge with two vectors, one a discrete pdf for the shock $\mathbf{f}$ and the other the discrete support η, both are $N_A \times 1$.
  - **COMPECON** Note that COMPECON has a set of routines for computing quadrature nodes and weights such as *lnwnorm* etc.
- Substituting in the approximation of the shock

$$\begin{aligned} V(x, z) &= \max_{x' \in B(x,z)} F(x, z, x', p) + \beta V_e(x', z) \\ V_e(x, z) &= \sum_a f_a \sum_{z' \in \mathcal{Z}} P(z, z') V(x' + \eta_a, z') \end{aligned}$$

- Now we proceed exactly as above, approximating V and V_e.
- The first equation is identical to before

$$\Phi(s)c = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x', p) + \beta \Phi([x', s_2])c^e$$

- The second equation needs a bit more work though it ends up being

$$\Phi(s)c^e = \underbrace{(\mathbf{P}\otimes\mathbf{I}_{N_X})}_{N\times N}\underbrace{(\mathbf{I}_N\otimes\mathbf{f}')}_{N\times N_A N}\underbrace{\Phi([s_1\otimes\mathbf{i}_{N_A}+\mathbf{i}_N\otimes\eta\,,\;s_2\otimes\mathbf{i}_{N_A}])}_{N_A N\times N}\underbrace{c}_{N\times 1}$$

- Again, you should check this by writing it out in full with $N_A = 2$, $N_X = 3$, $N_Z = 2$.
- **Newton -** We can then proceed exactly as above using Newton-methods to solve the system

$$\begin{aligned}\Phi(s)c &= \max_{x'\in\mathbf{B}(s)} \mathbf{F}(s,x',p) + \beta\Phi([x',s_2])c^e \\ \Phi(s)c^e &= \underbrace{(\mathbf{P}\otimes\mathbf{I}_{N_X})(\mathbf{I}_N\otimes\mathbf{f}')\Phi([s_1\otimes\mathbf{i}_{N_A}+\mathbf{i}_N\otimes\eta\,,\;s_2\otimes\mathbf{i}_{N_A}])}_{H}c^1\end{aligned}$$

- **Pre-compute -** Again note that the matrix H can be pre-computed.
- We've now solved for the coefficient vectors so let's turn to computing the stationary distribution.
- Here we have to include the effect of the η shocks on the law of motion for x.
- To compute Q proceed as follows. The matrix Q_Z is as above. The matrix Q_X is given as follows.
  - For $a = 1,\ldots,N_A$, compute $Q_X^a = funbas(fspaceerg, s_1+\eta_a)$
  - Then $Q_X = \sum_{a=1}^{N_A} f_a Q_X^a$.
- See how this works. Take an s_i. There are N_A different $\eta_a's$ that can shock s_i. We first compute where the masses should go under each shock. And then add these all together weighted by the probabilities of the shocks.

4.5 Continuously distributed shocks

- Suppose our basic model is as follows

$$V(x,z) = \max_{x'\in B(x,z)} F(x,z,x',p) + \beta\int V(x',z')d\Gamma(z,z')$$

- Suppose that $\Gamma(z,z')$ is governed by the following

$$\log z' = \rho\log z + \eta, \quad \eta\sim N(\mu,\sigma)$$

- **Example** - The model in Kehoe, Midrigan, Pastorino (2015) can be represented as a combination of this model and that of regime switching as detailed above. In their model a worker that is unemployed has a productivity processes that is AR(1) in logs but subject to a negative drift μ. This is not possible to discretize.
- **Note -** A more complicated process with additional *iid* shocks can easily be figured out from combining this and the previous section.

- We can the write the problem as

$$\begin{aligned} V(x,z) &= \max_{x' \in B(x,z)} F(x,z,x',p) + \beta \int V(x', g(z,\eta)) dF(\eta) \\ g(z,\varepsilon) &= \exp\left(\rho \log z + \eta\right) \end{aligned}$$

- We can now proceed exactly as in the *iid* case. First approximating the *iid* shock according to discrete pdf $\mathbf{f}$ and discrete support η with N_A points. In this case we would use a Gauss-Hermite quadtrature method to pick $\mathbf{f}$ and η.

- Again we split up the problem

$$\begin{aligned} V(x,z) &= \max_{x' \in B(x,z)} F(x,z,x',p) + \beta V_e(x',z) \\ V_e(x,z) &= \sum_{a=1}^{N_A} f_a V(x, g(z,\eta_a)) \end{aligned}$$

- Using approximants for V and V^e we have the system

$$\begin{aligned} \Phi(s)c &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x',p) + \beta \Phi([x', s_2])c^e \\ \Phi(s)c^e &= (\mathbf{I}_N \otimes \mathbf{f}')\Phi([s_1 \otimes \mathbf{i}_{N_A}\ ,\ g(s_2 \otimes \mathbf{i}_{N_A}, \mathbf{i}_N \otimes \eta)\ ])c^1 \end{aligned}$$

- **Issue** - For any grid of z it will be the case that at the maximum point on the grid $\bar{z}$ that $g(\bar{z},\eta) > \bar{z}$ for some large η. Same is true but in reverse at $\underline{z}$. It's well known that approximants might fail off the grid, but It's worth first trying things out and seeing if this is the case. However even if the solution converges under one set of parameters and prices it might not under all. A robust approach is therefore to proceed as follows:

  1. Choose the grid for z such that under the stationary distribution of the process the probability of $z > \bar{z}$ or $z < \underline{z}$ is less than some cut-off. These can be computed by hand in the case of a log-normal process.
  2. Constrain the function g such that we use $\tilde{g}$ where

$$\tilde{g}(z,\eta) = \max(\min(g(z,\eta), \bar{z}), \underline{z})$$

- Having solved the coefficients we then compute the stationary distribution.

- Here we could make a finer grid for z than used in solving the value function problem. Then proceed as standard and compute the policies $x'(x,z)$ where (x,z) are on the fine grids x_d and z_d

- We compute Q in the same way as we did the *iid* shock but this time for the second state.

- To compute Q proceed as follows. The matrix Q_X is as above. The matrix Q_Z is given as follows.

  - For $a = 1, \ldots, N_A$, compute $Q_Z^a = funbas(fspaceergZ, g(s_2, \eta_a))$
  - Then $Q_Z = \sum_{a=1}^{N_A} f_a Q_Z^a$.

4.6 Endogenous price of debt

- Suppose the basic model is as follows

$$V(x,z) = \max_{x' \in B(x,z)} F(x,z,x',q(x',z)) + \beta \sum_{z'} V(x',z') d\Gamma(z,z')$$

but F is also a function of some endogenous price Q which depends on the actions of the agent today x' and their productivity z'.

- I drop the dependence of F on p for ease of notation

- **Example** - F is dividends, the endogenous state $x = (k,b)$ is capital and debt. The firm faces a price $q(b',k',z)$ for debt and dividends. The firm can invest I and $k' = (1-\delta)k + i$ while dividends are

$$d = zk^{\alpha} - b + Q(b',k',z)b' - i$$

- The firm takes the price as given but in equilibrium it is determined as some function

$$q(x',z) = \sum_{z'} g(x'(x',z),z')$$

- **Example** - The price of debt $q(b',k',z)$ depends on the probability of default next period

$$q(x',z) = \beta \text{Pr}[\text{Repay next period}|x',z] = \beta \sum_{z'} \mathbf{1}_{[R(x',z')=1]} \Gamma(z,z')$$

where $R(x',z') = 1$ indicates the *optimal policy* of repayment next period. The firm defaults if $V(x',z') > 0$ so $R(x',z') = \mathbf{1}_{[V(x',z')>0]}$

- Note that g is a function of the firm's optimal decisions *in the next period.* It is a function of a max. This may be unnecessary notation but is worth keeping around for now.

- We can solve this problem by approximating both unknown functions at the same time.

- Instead of approximating the unknown price we approximate whatever object enters F linearly in the states. For example we approximate $H(b',k',z) = q(b',k',z)b'$ instead of q and then back-out q once we have solved for H. The reason for this will become clear shortly.

- So our system is

$$\begin{aligned} V(x,z) &= \max_{x' \in B(x,z)} F(x,z,x,H(x',z)) + \beta \sum_{z'} V(x',z') \Gamma(z,z') \\ H(x,z) &= x \sum_{z'} g(x'(x,z),z) d\Gamma(z,z') \end{aligned}$$

- For ease of notation I'll approximate V directly but note that a better method for computation is to approximate the expected value

- Let's approximate H and V as above, substituting in the optimal policies $x'(s)$

$$\begin{aligned}\Phi(s)c_V &= \mathbf{F}(s, x'(s), \Phi([x'(s), s_2])c_H) + \beta(\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi([x'(s), s_2])c_V \\ \Phi(s)c_H &= s_2 \odot [(\mathbf{P} \otimes \mathbf{I}_{N_X})_{z'} g([x'(s), s_2])]\end{aligned}$$

- Clearly we can solve this system by iteration. Given a c_H and c_V we can solve the first equation for c_V' and the policies $x'(s)$. Using the policies we can determine a new c_H' from the equilibrium condition.
- Now consider deriving the Jacobian of the system with respect to $c = (c_V', c_H')'$.
- A subtle point here, as always, is that a marginal change in c does not effect the derivatives of the system *through* the policy $x'(s)$ as the envelope theorem holds
- Note also that

$$\frac{\partial}{\partial c_H}\mathbf{F}(s, x'(s), \Phi([x'(s), s_2])c_H) = \mathbf{F}_H(s, x'(s), \Phi([x'(s), s_2])c_H)\Phi([x'(s), s_2])$$

- So the Jacobian is

$$\mathbf{D}(c_V, c_H) = \begin{bmatrix} \Phi(s) - \beta(\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi([x'(s), s_2]) & -\mathbf{F}_H(s, x'(s), \Phi([x'(s), s_2])c_H)\Phi([x'(s), s_2]) \\ 0 & \Phi(s) \end{bmatrix}$$

- We can then proceed as usual using a Newton method.