Quantitative Macroeconomics

Lecture 8

Simon Mongey

New York University

November 19, 2015

Introduction

- Take a standard problem in quantitative macroeconomics - recursive competitive equilibrium
- Re-write in notation which makes it easy to solve using collocation
  - Based on Miranda and Fackler
  - General notation useful for a lot of problems. Fully vectorized.
  - Type straight into computer
  - Pre-computed integrals (Maliar-Maliar-Judd)
- Robust method for computing stationary distributions
- A simple parallelised bisection method
- Discuss generalisations
- HPC

Simple problem

Agent's problem

The agent takes the price p as given and solves the following given their states (x, z)

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

Solution $V(x,z)$ and $x'(x,z)$.

Simple problem

Agent's problem

The agent takes the price p as given and solves the following given their states (x, z)

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

Solution $V(x,z)$ and $x'(x,z)$.

Extensions

- Markov-switching, for example uncertainty regimes
- Discrete choices
- $z' \sim \Gamma(z,z')$, for example $\log(z') = \rho \log(z) + \varepsilon$
- Next period x' subject to iid shock

Simple problem

Agent's problem

The agent takes the price p as given and solves the following given their states (x, z)

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

Solution $V(x,z)$ and $x'(x,z)$.

Equilibrium

Let $\lambda(x,z)$ be the stationary measure of agents then Λ satisfies

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

Market clearing requires that

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

Concern for robustness - $\max_{\text{strategy}} \min_{\text{what can go wrong}}$

Concern for robustness - $\max_{\text{strategy}} \min_{\text{what can go wrong}}$

- Problem sets
  - Reasonable / already estimated θ, already given p
  - All very well behaved
  - Tend to be solving each piece of a problem separately
  - Ideal Gradient-based solvers for maximization problems, Chebyshev polynomials for approximants.

Concern for robustness - $\max_{\text{strategy}} \min_{\text{what can go wrong}}$

- Problem sets
  - Reasonable / already estimated θ, already given p
  - All very well behaved
  - Tend to be solving each piece of a problem separately
  - Ideal Gradient-based solvers for maximization problems, Chebyshev polynomials for approximants.
- Quantitative work
  - No idea what θ is, always solving for a p
  - Possibly manic
  - All routines are nested
  - Want All routines highly robust. Bisection methods, splines, golden-section search solvers. Tolerances should be highest in the inner-most loop.
  - Want Set-up to be highly flexible. What if I add iid shocks? Uncertainty? Completely different model?

Notation

- Let's call a state $s = (x, z)$

$$V(s) = \max_{x' \in B(s)} F(s, x') + \beta \sum_{z' \in \mathcal{Z}} P(s_2, z') V([x', z'])$$

Notation

- Let's call a state $s = (x, z)$

$$V(s) = \max_{x' \in B(s)} F(s, x') + \beta \sum_{z' \in \mathcal{Z}} P(s_2, z') V([x', z'])$$

- Choose a set of nodes for collocation
- Here focus is on splines, using Chebyshev these are chosen for you.

  See code - Simple way to add curvature to nodes for x

  See code - Way to remove low probability points for z

Notation

- Let's call a state $s = (x, z)$

$$V(s) = \max_{x' \in B(s)} F(s, x') + \beta \sum_{z' \in \mathcal{Z}} P(s_2, z') V([x', z'])$$

- Choose a set of nodes for collocation
- Here focus is on splines, using Chebyshev these are chosen for you.

  See code - Simple way to add curvature to nodes for x

  See code - Way to remove low probability points for z

- Result - N_X nodes for x, N_Z nodes for z, matrix $\mathbf{P}$

$$\begin{aligned} \mathbf{s} &= [\mathbf{i}_{N_Z} \otimes \mathbf{x}, \mathbf{z} \otimes \mathbf{i}_{N_X}] \\ N_S &= N_X \times N_Z \end{aligned}$$

Pre-computing integrals - Maliar, Maliar and Judd

- Take the problem and split it in two

$$\mathbf{V}(s) = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \mathbb{E}\left[\mathbf{V}([x', z'])\right]$$

Pre-computing integrals - Maliar, Maliar and Judd

- Take the problem and split it in two

$$\begin{aligned}
\mathbf{V}(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \mathbb{E}\left[\mathbf{V}([x', z'])\right] \\
&\downarrow \\
\mathbf{V}_1(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \mathbf{V}_2([x', s_2]) \\
\mathbf{V}_2(s) &= \mathbb{E}\left[\mathbf{V}_1([s_1, z'])\right]
\end{aligned}$$

- **Why?**
  1. When solving the $\max$ problem we won't have to compute expectations $\mathbb{E}$
  2. If there are kinks in $\mathbf{V}$ then taking $\mathbb{E}$ can smooth these, allowing us to still use non-linear splines
  3. Given above point it is actual **most robust** to solve for the **expected value function**, as I do in practice **See code**

$$\mathbf{V}_1^{\mathbb{E}}(s) = \mathbb{E}\left[\max_{x' \in \mathbf{B}([s_1, z'])} \mathbf{F}([s_1, z'], x') + \mathbf{V}_1^{\mathbb{E}}([s_1, z'])\right]$$

Approximation

- System to be solved on nodes s

$$\begin{aligned}
\mathbf{V}_1(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \mathbf{V}_2([x', s_2]) \\
\mathbf{V}_2(s) &= \mathbb{E}\left[\mathbf{V}_1([s_1, z'])\right]
\end{aligned}$$

Approximation

- System to be solved on nodes s

$$\begin{aligned}\mathbf{V}_1(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \mathbf{V}_2([x', s_2]) \\ \mathbf{V}_2(s) &= \mathbb{E}\left[\mathbf{V}_1([s_1, z'])\right]\end{aligned}$$

- Replace functions with approximants

$$\begin{aligned}V_1(s_i) &= \sum_{i=1}^{N_S} \phi(s_i) c_j^1 \qquad V_2(s_i) = \sum_{i=1}^{N_S} \phi(s_i) c_j^2 \\ \mathbf{V}_1(s) &= \Phi(s) c^1 \qquad \mathbf{V}_2(s) = \Phi(s) c^2\end{aligned}$$

Approximation

- System to be solved on nodes s

$$\begin{aligned}\Phi(s)c^1 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 &= \mathbb{E}\left[\Phi([s_1, z'])c^1\right]\end{aligned}$$

- Replace functions with approximants

$$\begin{aligned}V_1(s_i) &= \sum_{i=1}^{N_S} \phi(s_i)c_j^1 \qquad V_2(s_i) = \sum_{i=1}^{N_S} \phi(s_i)c_j^2 \\ \mathbf{V}_1(s) &= \Phi(s)c^1 \qquad \mathbf{V}_2(s) = \Phi(s)c^2\end{aligned}$$

Pre-computing integrals - Maliar, Maliar and Judd

$$\Phi(s)c^2 = \mathbb{E}\left[\Phi([s_1, z'])c^1\right]$$

$$\Phi\left(\begin{bmatrix} x_1 & z_1 \\ x_2 & z_1 \\ x_3 & z_1 \\ x_1 & z_2 \\ x_2 & z_2 \\ x_3 & z_2 \end{bmatrix}\right) c^2 = \begin{bmatrix} p_{11} & 0 & 0 & p_{12} & 0 & 0 \\ 0 & p_{11} & 0 & 0 & p_{12} & 0 \\ 0 & 0 & p_{11} & 0 & 0 & p_{12} \\ p_{21} & 0 & 0 & p_{22} & 0 & 0 \\ 0 & p_{21} & 0 & 0 & p_{22} & 0 \\ 0 & 0 & p_{21} & 0 & 0 & p_{22} \end{bmatrix} \Phi\left(\begin{bmatrix} x_1 & z_1 \\ x_2 & z_1 \\ x_3 & z_1 \\ x_1 & z_2 \\ x_2 & z_2 \\ x_3 & z_2 \end{bmatrix}\right) c^1$$

$$\Longrightarrow \Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

Solution

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\Phi(s)c^1 = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2$$

$$\Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

Solution

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\begin{aligned}\Phi(s)c^1 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 &= (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1\end{aligned}$$

- Contains a $\max$ so any solution scheme must be iterative

1. Contraction - Easy to implement Howard improvement, and faster than usual since using basis matrices)
   - Fixing a policy x' HI reduces to matrix algebra

$$\begin{aligned}\Phi(s)c^1 &= \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 &= (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1\end{aligned}$$

Solution

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\Phi(s)c^1 = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2$$

$$\Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

- Contains a $\max$ so any solution scheme must be iterative

1. Contraction - Easy to implement Howard improvement, and faster than usual since using basis matrices)
   - Fixing a policy x' HI reduces to matrix algebra

$$\begin{bmatrix} \Phi(s) & 0 \\ 0 & \Phi(s) \end{bmatrix} \begin{bmatrix} c^1 \\ c^2 \end{bmatrix} = \begin{bmatrix} \mathbf{F}(s, x') \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & \beta\Phi([x', s_2]) \\ (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s) & 0 \end{bmatrix} \begin{bmatrix} c^1 \\ c^2 \end{bmatrix}$$

Solution

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\Phi(s)c^1 = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2$$

$$\Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

- Contains a $\max$ so any solution scheme must be iterative

1. Contraction - Easy to implement Howard improvement, and faster than usual since using basis matrices)
   - Fixing a policy x' HI reduces to matrix algebra

$$\begin{bmatrix} c^1 \\ c^2 \end{bmatrix} = \begin{bmatrix} \Phi(s) & -\beta\Phi([x', s_2]) \\ -(\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s) & \Phi(s) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{F}(s, x') \\ 0 \end{bmatrix}$$

Solution

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\Phi(s)c^1 = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2$$

$$\Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

- Contains a $\max$ so any solution scheme must be iterative

2. Newton-Raphson - What we're aiming for, incredibly fast

Solution - Contraction

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\Phi(s)c^1 = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2$$

$$\Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

1. Guess a $c_0 = (c_0^{1\prime}, c_0^{2\prime})$
2. Given c_k update to c_{k+1} by

$$c_{k+1}^1 = \Phi(s)^{-1}\left[\max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c_k^2\right]$$

$$c_{k+1}^2 = \Phi(s)^{-1}(\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c_k^1$$

Solution - Contraction

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\Phi(s)c^1 = \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2$$

$$\Phi(s)c^2 = (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1$$

1. Guess a $c_0 = (c_0^{1\prime}, c_0^{2\prime})$
2. Given c_k update to c_{k+1} by

$$c_{k+1}^1 = \Phi(s)^{-1}\left[\max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c_k^2\right]$$

$$c_{k+1}^2 = \Phi(s)^{-1}(\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c_k^1$$

- Debug - Should *always* check this leads to convergence. It should if the problem is a contraction and approximation is reasonable.

Solution - Newton Raphson

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\begin{aligned} \mathbf{g}(c) &= 0 \\ \begin{bmatrix} g_1\left(c^1, c^2\right) \\ g_2\left(c^1, c^2\right) \end{bmatrix} &= \begin{bmatrix} \Phi(s)c^1 - \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 - (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1 \end{bmatrix} \end{aligned}$$

Solution - Newton Raphson

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\begin{aligned} \mathbf{g}(c) &= 0 \\ \begin{bmatrix} g_1\left(c^1, c^2\right) \\ g_2\left(c^1, c^2\right) \end{bmatrix} &= \begin{bmatrix} \Phi(s)c^1 - \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 - (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1 \end{bmatrix} \end{aligned}$$

1. Guess a $c_0 = (c_0^{1\prime}, c_0^{2\prime})$
2. Given c_k update to c_{k+1} by

$$c_{k+1} = c_k - \left(\frac{\partial \mathbf{g}(c)}{\partial c}\bigg|_{c=c_k} \right)^{-1} \mathbf{g}(c_k)$$

Solution - Newton Raphson

- $2N_S$ equations in $2N_S$ unknowns: $c = (c^{1\prime}, c^{2\prime})'$

$$\begin{aligned} \mathbf{g}(c) &= 0 \\ \begin{bmatrix} g_1\left(c^1, c^2\right) \\ g_2\left(c^1, c^2\right) \end{bmatrix} &= \begin{bmatrix} \Phi(s)c^1 - \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 - (\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s)c^1 \end{bmatrix} \end{aligned}$$

1. Guess a $c_0 = (c_0^{1\prime}, c_0^{2\prime})$
2. Given c_k update to c_{k+1} by

$$\begin{aligned} c_{k+1} &= c_k - \left(\frac{\partial \mathbf{g}(c)}{\partial c}\bigg|_{c=c_k} \right)^{-1} \mathbf{g}(c_k) \\ \frac{\partial \mathbf{g}(c)}{\partial c}\bigg|_{c=c_k} &= \begin{bmatrix} \Phi(s) & -\beta\Phi([x'(s), s_2]) \\ -(\mathbf{P} \otimes \mathbf{I}_{N_X})\Phi(s) & \Phi(s) \end{bmatrix} \end{aligned}$$

Filling in the gaps - How to solve $\max$?

- Given a c we need to compute

$$\max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \Phi([x', s_2]) c^2$$

- Compecon function `goldenx` does vectorized golden-search. Robust. Fast.
- Simple to handle constraints $x' \in \mathbf{B}(s)$

Filling in the gaps - How to solve $\max$?

- Given a c we need to compute

$$\max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \Phi([x', s_2]) c^2$$

- Compecon function `goldenx` does vectorized golden-search. Robust. Fast.
- Simple to handle constraints $x' \in \mathbf{B}(s)$
- What if x is multi-dimensional?

  Option 1 - Relatively straight-forward to write a Nelder-Mead algorithm. Can still keep everything vectorized.

  Option 2 - Use a FOC to pin down x_2 in terms of x_1, use golden-search on x_1.

  Option 3 - Use nested golden-search, one variable then the other.

Filling in the gaps - How to compute $\Phi(s)$?

- Compecon function `fundef` establishes a `fspace` object, for example ...
- Can then use the function `funbas` to compute basis matrices given `fspace` and any length vector x
- Compecon is nice because it's suitably general. Can jump back and forth between different approximation schemes (splines of different orders, Chebyshev etc).
- ...or write your own code. The main task in any collocation algorithm is computing basis matrices, over and over again.
- Can we avoid some of these computations?

Filling in the gaps - How to compute $\Phi(s)$?

- In the case of linear splines, the basis matrix $\Phi([x, z])$ is a tensor product.

$$\Phi([x, z]) = \Phi_Z(z) \circledast \Phi_X(x)$$

  where Φ_X is N_X wide and Φ_Z is N_Z wide.
- Looking at our problem note that we only have to compute $\Phi_Z(s_2)$ once.
- Each iteration of the solver for $\max_x'$ we need to do two things, (i) compute $\Phi_X(x')$ then take the tensor product with $\Phi_Z(s_2)$ to form $\Phi(s)$.
- Compecon function `dprod` will do this, written in C++, super fast. Code spends 80% of its time in `dprod`

Filling in the gaps - How to compute $\Phi(s)$?

- In the case of linear splines, the basis matrix $\Phi([x,z])$ is a tensor product.

$$\Phi([x,z]) = \Phi_Z(z) \circledast \Phi_X(x)$$

  where Φ_X is N_X wide and Φ_Z is N_Z wide.
- Looking at our problem note that we only have to compute $\Phi_Z(s_2)$ once.
- Each iteration of the solver for $\max_x'$ we need to do two things, (i) compute $\Phi_X(x')$ then take the tensor product with $\Phi_Z(s_2)$ to form $\Phi(s)$.
- Compecon function `dprod` will do this, written in C++, super fast. Code spends 80% of its time in `dprod`
- Warning The tensor products work *backwards*

$$\Phi([x,y,z]) = \Phi_Z(z) \circledast \Phi_Y(y) \circledast \Phi_X(x) = \text{dprod}(\Phi_Z(z), \text{dprod}(\Phi_Y(y), \Phi_X(x)))$$

Filling in the gaps - $x' \in \mathbf{B}(s)$

- Write a function which takes s as an argument and gives a vector of upper and lower bounds for x' depending on your problem
- These bounds then passed to solver

$$
\begin{aligned}
[\underline{x}, \bar{x}] &= \text{menufun}(\text{'bounds'}, s, \text{param}, \text{glob}, \text{options}) \\
\text{obj} &= @(x')\text{valfunc}(x', c, s, \text{param}, \text{glob}, \text{options}) \\
x' &= \text{goldenx}(\text{obj}, \underline{x}, \bar{x})
\end{aligned}
$$

Extensions

Basic firm problem - $F(k, z, k')$ may contain adjustment costs etc

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

where P is a stochastic matrix

Extensions

Basic firm problem - $F(k, z, k')$ may contain adjustment costs etc

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

where P is a stochastic matrix

Extensions - Work into our framework

1. iid shocks to endogenous variable $k' = k'^* - \eta, \quad \eta \sim F(\eta)$
2. Markov 'uncertainty'* shocks P transitions between P_l and P_h
3. Discrete choice F depends on whether $\uparrow k$ or $\downarrow k$
4. Continuous process for z $\log z' = \rho \log z + \varepsilon, \quad \varepsilon \sim N(0, \sigma)$

* Not really *uncertainty* shocks at all but this is what the literature (regrettably) calls them

Extension 1 - iid shocks to endogenous variable

1. Firm problem

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

Extension 1 - iid shocks to endogenous variable

1. Firm problem

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

2. Re-write $s = (k,z)$ and split up

$$\begin{aligned} \mathbf{V}_1(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta \mathbf{V}_2([x', s_2]) \\ \mathbf{V}_2(s) &= \mathbb{E}\left[\int_{\underline{\eta}}^{\bar{\eta}} \mathbf{V}_1([s_1 - \eta, z']) dF(\eta) \right] \end{aligned}$$

Extension 1 - iid shocks to endogenous variable

1. Firm problem

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

2. Re-write $s = (k,z)$ and split up

$$\begin{aligned} \mathbf{V}_1(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta \mathbf{V}_2([x', s_2]) \\ \mathbf{V}_2(s) &= \mathbb{E}\left[\int_{\underline{\eta}}^{\bar{\eta}} \mathbf{V}_1([s_1 - \eta, z']) dF(\eta) \right] \end{aligned}$$

3. Approximate iid shock with N_η nodes η with weights w

$$\mathbf{V}_2(s) = \mathbb{E}\left[\sum_{i=1}^{N_\eta} w_i \mathbf{V}_1([s_1 - \eta_i, z']) \right]$$

Extension 1 - iid shocks to endogenous variable

4. Approximate value functions

$$\begin{aligned}
\Phi(s)c^1 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \Phi([x', s_2])c^2 \\
\Phi(s)c^2 &= \mathbb{E}\left[\sum_{i=1}^{N_\eta} w_i \Phi([s_1 - \eta_i, z'])c^1\right]
\end{aligned}$$

Extension 1 - iid shocks to endogenous variable

4. Approximate value functions

$$\begin{aligned}\Phi(s)c^1 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta\Phi([x', s_2])c^2 \\ \Phi(s)c^2 &= \mathbb{E}\left[\sum_{i=1}^{N_\eta} w_i \Phi([s_1 - \eta_i, z'])c^1\right]\end{aligned}$$

5. Write expectations as matrix operation

$$\Phi(s)c^2 = (P \otimes \mathbf{I}_{N_X})(\mathbf{I}_{N_S} \otimes w')\Phi([s_1 \otimes \mathbf{i}_{N_\eta} - \mathbf{i}_{N_S} \otimes \eta\ ,\ s_2 \otimes \mathbf{i}_{N_\eta}])c^1$$

Extension 2 - Markov 'uncertainty' shocks

Firm faces Markov-switching stochastic matrix P_i with $i \in \{l, h\}$

$$V_i(k,z) = \max_{a' \in B(k,z)} F(k,z,z') + \beta \sum_{z' \in \mathcal{E}} P_i(z,z') \left[p_i V_i(k',z') + (1-p_i) V_j(k',z') \right]$$

Extension 2 - Markov 'uncertainty' shocks

Firm faces Markov-switching stochastic matrix P_i with $i \in \{l, h\}$

$$V_i(k,z) = \max_{a' \in B(k,z)} F(k,z,z') + \beta \sum_{z' \in \mathcal{E}} P_i(z,z') \left[p_i V_i(k',z') + (1-p_i) V_j(k',z') \right]$$

Re-write $s = (k,z)$ and split up

$$\begin{aligned}
V_1(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta V_3([x', s_2]) \\
V_2(s) &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta V_4([x', s_2]) \\
V_3(s) &= \sum_{z' \in \mathcal{Z}} P_l(s_2,z') \left[p_l V_1([s_1,z']) + (1-p_l) V_2([s_1,z']) \right] \\
V_4(s) &= \sum_{z' \in \mathcal{Z}} P_h(s_2,z') \left[p_h V_2([s_1,z']) + (1-p_h) V_1([s_1,z']) \right]
\end{aligned}$$

Extension 2 - Markov 'uncertainty' shocks

Approximate value functions, write expectation as a matrix operation

$$
\begin{aligned}
\Phi(s)c^1 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \Phi(s) c^3([x', s_2]) \\
\Phi(s)c^2 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s, x') + \beta \Phi(s) c^4([x', s_2]) \\
\Phi(s)c^3 &= (\mathbf{P}_l \otimes \mathbf{I}_{N_X}) \left[p_l \Phi(s) c^1 + (1 - p_l) \Phi(s) c^2 \right] \\
\Phi(s)c^4 &= (\mathbf{P}_h \otimes \mathbf{I}_{N_X}) \left[p_h \Phi(s) c^2 + (1 - p_h) \Phi(s) c^1 \right]
\end{aligned}
$$

Proceed as above, slightly more complicated Jacobian

Extension 3 - Discrete choice

Firm problem for $i \in \{d, u\}$

$$V_i(k,z) = \max_{k' \in B_i(k,z)} F_i(k,z,k') + \beta \sum_{z' \in \mathcal{Z}} P(z,z') \max\{V_d(k',z'), V_u(k',z')\}$$

Extension 3 - Discrete choice

Firm problem for $i \in \{d, u\}$

$$V_i(k,z) = \max_{k' \in B_i(k,z)} F_i(k,z,k') + \beta \sum_{z' \in \mathcal{Z}} P(z,z') \max\{V_d(k',z'), V_u(k',z')\}$$

Re-write $s = (k,z)$ and split up

$$\begin{aligned}
V_1(s) &= \max_{x' \in \mathbf{B}_d(s)} \mathbf{F}_d(s,x') + \beta V_3([x', s_2]) \\
V_2(s) &= \max_{x' \in \mathbf{B}_u(s)} \mathbf{F}_u(s,x') + \beta V_3([x', s_2]) \\
V_3(s) &= \sum_{z' \in \mathcal{Z}} P(s_2,z') \max\{V_1([s_1,z']), V_2([s_1,z'])\}
\end{aligned}$$

Extension 3 - Discrete choice

Approximate value functions, write expectation as a matrix operation

$$\begin{aligned}
\Phi(s)c^1 &= \max_{x' \in \mathbf{B}_d(s)} \mathbf{F}_d(s,x') + \beta\Phi(s)([x', s_2])c^3 \\
\Phi(s)c^2 &= \max_{x' \in \mathbf{B}_u(s)} \mathbf{F}_u(s,x') + \beta\Phi(s)([x', s_2])c^3 \\
\Phi(s)c^3 &= (P \otimes \mathbf{I}_{N_X}) \max\left\{\Phi([s_1, z'])c^1, \Phi([s_1, z'])c^2\right\}
\end{aligned}$$

Extension 4 - AR(1) process for z

Firm problem

$$
\begin{aligned}
V(k,z) &= \max_{k' \in B(k,z)} F(k,z,k') + \beta \int V(k', g(z,\varepsilon)) d\Gamma(\varepsilon) \\
g(z,\varepsilon) &= \exp\left(\rho \log z + \varepsilon\right) \\
\varepsilon &\sim N(0, \sigma_\varepsilon)
\end{aligned}
$$

Extension 4 - AR(1) process for z

Firm problem

$$\begin{aligned} V(k,z) &= \max_{k' \in B(k,z)} F(k,z,k') + \beta \int V(k', g(z,\varepsilon)) d\Gamma(\varepsilon) \\ g(z,\varepsilon) &= \exp\left(\rho \log z + \varepsilon\right) \\ \varepsilon &\sim N(0, \sigma_\varepsilon) \end{aligned}$$

Proceed exactly as above for the iid case but approximating $\Gamma(\varepsilon)$ with a Gaussian quadrature scheme with N_ε nodes ε and weights w, arrive at

$$\begin{aligned} \Phi(s)c^1 &= \max_{x' \in \mathbf{B}(s)} \mathbf{F}(s,x') + \beta \Phi([x', s_2])c^2 \\ \Phi(s)c^2 &= (\mathbf{I}_{N_S} \otimes w')\Phi([s_1 \otimes \mathbf{i}_{N_\varepsilon},\ g(s_2 \otimes \mathbf{i}_{N_\varepsilon}, \mathbf{i}_{N_S} \otimes \varepsilon)])c^1 \end{aligned}$$

Stationary distribution

- Method of Young (2010), Gianluca discussed this last week
- Approximate the distribution λ with a fine histogram L
- Having solved the value function problem we have the expected value function
- Now given any (notation warning) s we can solve for $x'(s)$

- Choose a fine grid of s for approximation of the distribution
- Solve for $x'(s)$ by computing $\max_{x'} \ \mathbf{F}(s, x') + \beta\Phi([x', s_2])c^1$
- Let L be the approximate distribution, we want to use $x'(s)$ and P to generate a matrix Q such that the law of motion can be represented

$$L = Q'L$$

- Then either take out principle eigen vector of Q or iterate given a guess L^0

Stationary distribution - Q

- Construct Q by taking the tensor product of two matrices Q_X and Q_Z

$$Q = Q_Z \circledast Q_X$$

- The $N_S \times N_X$ matrix Q_X matrix uses $x'(s)$ to shift mass in terms of endogenous transitions
- The $N_S \times N_Z$ matrix Q_Z matrix *then* uses P to shift mass in terms of exogenous transitions

Stationary distribution - Q_X

- Given the approximating grid $x = (x_1, \ldots, x_{N_X})$, we compute

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

Stationary distribution - Q_X

- Given the approximating grid $x = (x_1, \ldots, x_{N_X})$, we compute

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

- Note Useful in that aggregates will be unbiased!

Stationary distribution - Q_X

- Given the approximating grid $x = (x_1, \ldots, x_{N_X})$, we compute

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

- Note Useful in that aggregates will be unbiased!

- Note When stacked, $Q_X = \Phi(x'(s))$ where Φ is basis matrix of linear-spline

$$\begin{aligned} fspaceQ_X &= fundef(\{spli, x, 0, 1\}) \\ Q_X &= funbas(fspaceQ_X, x'(s)) \end{aligned}$$

Stationary distribution - Q_Z

- The matrix Q_Z is easy to compute in the case of discrete transitions

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

Stationary distribution - Q_Z

- The matrix Q_Z is easy to compute in the case of discrete transitions

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

- In the case of 'continuous' shocks, say due to an AR(1) process then we proceed as follows

Stationary distribution - Q_Z

- The matrix Q_Z is easy to compute in the case of discrete transitions

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

- In the case of 'continuous' shocks, say due to an AR(1) process then we proceed as follows
- We approximate the iid term using N_ε nodes ε and weights w

Stationary distribution - Q_Z

- The matrix Q_Z is easy to compute in the case of discrete transitions

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

- In the case of 'continuous' shocks, say due to an AR(1) process then we proceed as follows
- We approximate the iid term using N_ε nodes ε and weights w
- Then we have

$$Q_Z = \sum_{i=1}^{N_\varepsilon} w_i Q_{Z,i}$$

where

$$Q_{Z,i} = \Phi(g(s_2, \varepsilon_i)) \quad , \quad g(s_2, \varepsilon_i) = \exp\left(\rho \log s_2 + \varepsilon_i\right)$$

Stationary distribution - Q_Z

- The matrix Q_Z is easy to compute in the case of discrete transitions

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

- In the case of 'continuous' shocks, say due to an AR(1) process then we proceed as follows
- We approximate the iid term using N_ε nodes ε and weights w
- Then we have

$$Q_Z = \sum_{i=1}^{N_\varepsilon} w_i Q_{Z,i}$$

where

$$Q_{Z,i} = \Phi(g(s_2, \varepsilon_i)) \quad , \quad g(s_2, \varepsilon_i) = \exp\left(\rho \log s_2 + \varepsilon_i\right)$$

- Something along these lines also works for iid shocks to $x'(s)$

Equilibrium - Bisection

- Given Q can compute approximation of stationary distribution L
- Then aggregate $\bar{x}(p)$ is

$$\bar{x}(p) = L's_1$$

Equilibrium - Bisection

- Given Q can compute approximation of stationary distribution L
- Then aggregate $\bar{x}(p)$ is

$$\bar{x}(p) = L's_1$$

- Bisection
  - Fix $(\underline{p}, \bar{p})$ and $p_0 = \frac{\underline{p}+\bar{p}}{2}$
  - If $\bar{x}(p_k) > \bar{x}$ set $(\underline{p}, \bar{p}) = (p_k, \bar{p})$
  - If $\bar{x}(p_k) < \bar{x}$ set $(\underline{p}, \bar{p}) = (\underline{p}, p_k)$

Equilibrium - Parallel Bisection

- Haven’t used parallel computing anywhere - only used one CPU
- If have more than one CPU then have idle resources
- This is the cheapest way to speed up computation

Equilibrium - Parallel Bisection

- Haven't used parallel computing anywhere - only used one CPU
- If have more than one CPU then have idle resources
- This is the cheapest way to speed up computation
- If have $h > 1$ CPUs then separate $(\underline{p}, \bar{p})$ into $h + 1$ equally sized intervals

$$\left[\underline{p}, p_1, \ldots, p_h, \bar{p}\right]$$

- Proceed as in bisection with a parallelised loop over $\{p_1, \ldots, p_h\}$
- Find the interval in which $\bar{x}(p) = \bar{x}$ and then set the lower and upper bounds to the end points of that interval

Equilibrium - Parallel Bisection

- Haven't used parallel computing anywhere - only used one CPU
- If have more than one CPU then have idle resources
- This is the cheapest way to speed up computation
- If have $h > 1$ CPUs then separate $(\underline{p}, \bar{p})$ into $h + 1$ equally sized intervals

$$\left[\underline{p}, p_1, \ldots, p_h, \bar{p}\right]$$

- Proceed as in bisection with a parallelised loop over $\{p_1, \ldots, p_h\}$
- Find the interval in which $\bar{x}(p) = \bar{x}$ and then set the lower and upper bounds to the end points of that interval
- Note Particularly useful when the structure of the problem prevents you parallelising anything else, for example when computing period t equilibria for transition dynamics / Krussell-Smith

Mongey-Williams

- The key to the exercise is being to estimate the parameters associated with the aggregate shocks via maximum-likelihood
- Two problems
  - Solving heterogeneous agent models with aggregate shocks
  - Generating a linearized state-space system
- Both solved by the method of Reitter (JEDC, 2009)
  - Alisdair McKay: McKay & Reis (2015), McKay Nakamura Steinsson (2015), McKay (2013) & co-authors, Winberry (JMP)
  - Winberry (JMP, 2015): *Lumpy Investment, Business Cycles, and Stimulus Policy*
  - Terry (WP, 2014) - *Alternative Methods for Solving Heterogeneous Firm Models*, shows this to be an excellent and fast approximation method

Reitter Method - Krusell-Smith steady-state

- The household problem is

$$\begin{aligned} V(a,e) &= \max_{a'} u(c) + \beta \sum_{e'} P(e,e') V(a',e') \\ \text{s.t.} \quad c &= (1+r-\tau)a + we - a' + T \\ a' &\geq \underline{a} = 0 \end{aligned}$$

- Let the measure of households over (a,e) be $\lambda(a,e)$
- The market clearing condition is

$$K = \int a\lambda(a,e)$$

- Prices are due to perfect competition and the aggregate technology $Y = F(K,\bar{N}) = ZK^{\alpha}\bar{N}^{1-\alpha}$.
- Setting $\bar{N} = 1$ as a normalisation then

$$\begin{aligned} r &= \alpha Z K^{\alpha-1} - \delta \\ w &= (1-\alpha) Z K^{\alpha} \\ T &= \tau K \end{aligned}$$

Reitter Method - Krusell-Smith steady-state

1. Approximating $V(a, e)$
   - Consider a set of *collocation nodes* $s = (a_i, e_i)_{i=1,...N_i}$
   - Consider an *interpolant* on these nodes such that an interpolated function $f(s) = \Phi(s)c$

$$\Phi(s)c = \max_{a'} u\left((1 + r - \tau)s_1 + ws_2 - a' + T\right) + \beta(\mathbf{I} \otimes \mathbf{P})\Phi([a', s_2])c$$

   - An approximation of $V(a, e)$ is therefore the N_i vector c
2. Approximating $\lambda(a, e)$
   - Consider a set of *distribution nodes* $s = (a_j, e_j)_{j=1,...N_j}$
   - Using the policy functions from the problem approximate the law of motion of masses over s with a $N_j \times N_j$ matrix Q
   - Then an approximation of $\lambda(a, e)$ is therefore the N_j vector L, with $L = Q'L$
3. Remaining conditions (MC) $K = L's_1$, (Prices) $r = \alpha Z K^{\alpha-1} - \delta$, $w = (1 - \alpha)ZK^{\alpha}$

Reitter Method - Krusell-Smith steady-state

1. Approximating $V(a, e)$
   - Consider a set of *collocation nodes* $s = (a_i, e_i)_{i=1,...N_i}$
   - Consider an *interpolant* on these nodes such that an interpolated function $f(s) = \Phi(s)c$

$$\Phi(s)c_t = \max_{a'} u\left((1 + r_t - \tau_t)s_1 + w_t s_2 - a' + T\right) + \beta\mathbb{E}\left[(\mathbf{I} \otimes \mathbf{P})\Phi([a', s_2])c_{t+1}\right]$$

   - An approximation of $V_t(a, e)$ is therefore the N_i vector c

2. Approximating $\lambda(a, e)$
   - Consider a set of *distribution nodes* $s = (a_j, e_j)_{j=1,...N_j}$
   - Using the policy functions from the problem approximate the law of motion of masses over s with a $N_j \times N_j$ matrix Q
   - Then an approximation of $\lambda_t(a, e)$ is therefore the N_j vector L, with $L_{t+1} = Q_t' L_t$

3. Remaining conditions (MC) $K_t = L_t' s_1$, (Prices) $r_t = \alpha Z_t K_t^{\alpha-1} - \delta$, $w_t = (1 - \alpha)Z_t K_t^{\alpha}$

Reitter Method - System of equations

$$
\begin{aligned}
\Phi(s)c_t &= \max_{a'} \; u\left((1+r_t-\tau_t)s_1 + w_t s_2 - a' + T_t\right) \\
&\qquad +\beta\Phi([a', s_2])c_{t+1} + \eta_{t+1} \\
L_{t+1} &= Q_t' L_t \\
r_t &= \alpha Z_t K_t^{\alpha-1} - \delta \\
w_t &= (1-\alpha) Z_t K_t^{\alpha} \\
T_t &= \tau K_t \\
\log Z_{t+1} &= \rho \log Z_t + \varepsilon_{t+1}
\end{aligned}
$$

NYU - HPC

Resources

+ NYU HPC Wiki (Google it)
+ Some useful shell-script - HPCnotes.sh
+ Example pbs file - RUNsimple.pbs

NYU HPC

- What is it?
  - A cluster of nodes, each node has up to 20 CPUs
  - Think of a node as a large desktop (show pbstop)

NYU HPC

- What is it?
  - A cluster of nodes, each node has up to 20 CPUs
  - Think of a node as a large desktop (show pbstop)
- What's it good for?
  - Non-parallelisable code that takes a lot of memory / time

    Example - Transition dynamics / Krusell-Smith
  - Parallelised code that takes a lot of time

    Example - Calibration exercises

    Guvenen - Macro with Heterogeneity: A Practical Guide, section 8

NYU HPC

- What is it?
  - A cluster of nodes, each node has up to 20 CPUs
  - Think of a node as a large desktop (show pbstop)
- What's it good for?
  - Non-parallelisable code that takes a lot of memory / time

    Example - Transition dynamics / Krusell-Smith
  - Parallelised code that takes a lot of time

    Example - Calibration exercises

    Guvenen - Macro with Heterogeneity: A Practical Guide, section 8

Example - Model is indexed by $\theta \in \Theta$. **Aim** $\min_{\theta \in \Theta} \mathcal{C}(\theta)$

1. A hill-climbing algorithm (1-node $\times$ 1-CPU)
2. A random-search algorithm (20-node $\times$ 20-CPU)
3. MCMC (20-node $\times$ 1-CPU)

   See: Appendix B4 of Jarosh (2014)

NYU HPC - Access

- Get your adviser to approve access
- Operating systems
  - Mac - Can access directly from terminal
  - Windows - Download Cygwin, add packages (openssh, vi, ...)
- Logging in
  - PC-to-HPC: ssh ab1234@hpc.nyu.edu
  - HPC-to-Mercer: ssh ab1234@mercer.es.its.nyu.edu
- Structure - /scratch /home
- Submitting jobs - Interactive mode / Batch processing
  - Work in interactive mode
  - Write a *.pbs file

NYU HPC - Access

- Resources
  - Matlab, Fortran, Git, Python, OpenMPI, Julia, R, Stata, Dynare, Nag, Lapack, Pandas, NumPy, SciPy ...

NYU HPC - Access

- Resources
  - Matlab, Fortran, Git, Python, OpenMPI, Julia, R, Stata, Dynare, Nag, Lapack, Pandas, NumPy, SciPy ...
  - breakdancer, homer, jellyfish, introspection
  - See what is available: module avail
  - Load a module: module load introspection

NYU HPC - Access

- Resources
  - Matlab, Fortran, Git, Python, OpenMPI, Julia, R, Stata, Dynare, Nag, Lapack, Pandas, NumPy, SciPy ...
  - breakdancer, homer, jellyfish, introspection
  - See what is available: module avail
  - Load a module: module load introspection
- Move directory of Compecon files from PC-to-HPC-Mercer
  - From PC: scp -r Compecon sm4125@hpc.nyu.edu
  - From Mercer: scp sm4125@hpc.nyu.edu: /Compecon /scratch/sm4125
  - Don’t use WinSCP, it’s horrible
  - Use git to sync

NYU HPC - Access

- Resources
  - Matlab, Fortran, Git, Python, OpenMPI, Julia, R, Stata, Dynare, Nag, Lapack, Pandas, NumPy, SciPy ...
  - breakdancer, homer, jellyfish, introspection
  - See what is available: module avail
  - Load a module: module load introspection
- Move directory of Compecon files from PC-to-HPC-Mercer
  - From PC: scp -r Compecon sm4125@hpc.nyu.edu
  - From Mercer: scp sm4125@hpc.nyu.edu: /Compecon /scratch/sm4125
  - Don't use WinSCP, it's horrible
  - Use git to sync
- Tip Since coding at the bash-script can be a pain, work from a text. Then copy over to prompt. Notepad++ marks up a .sh file nicely.