Non-Linear Solution Methods

Solution Methods for Macroeconomic Models

Petr Sedláček

Solution Methods for Macroeconomic Models

- Monday - Tuesday: Solving models with "representative agents"
  - ~~Linearization in theory and practice: Dynare~~
  - Non-linear solutions methods: value function iteration, projection
  - Analyzing models: parameterization/estimation, simulation/IRFs
- Wednesday - Thursday: Solving models with "heterogeneous agents"
  - Models without aggregate uncertainty: basic algorithm
  - Models with aggregate uncertainty: key issues and alternatives
- Friday: "Final assignment"
  - Solve/estimate model with heterogeneous firms and aggregate uncertainty

Overview for today

Non-linear solution methods

- higher-order perturbation
- projection
- value function iteration

Analyzing DSGE models

- parameterization/estimation, simulation/IRFs

Overview for today

1. Higher-order perturbation
2. Projection
3. Value function iteration

Higher-Order Perturbation

Getting 1st order approximations

Recall that we can write our model as

$$\mathbb{E}_t F\left(g(h(x_t,\sigma)+\sigma\widetilde{\epsilon}_{t+1},\sigma), g(x_t,\sigma), h(x_t,\sigma)+\sigma\widetilde{\epsilon}_{t+1}, x_t \right) = 0$$

Deriving coefficients of Taylor polynomial

For simplicity, substitute out consumption to get $F[x_{t+2}, x_{t+1}, x_t] = 0$

$$F_x = \frac{\partial F}{\partial x_{t+2}} \frac{\partial x_{t+2}}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial x_t} + \frac{\partial F}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial x_t} + \frac{\partial F}{\partial x_t}$$

$$= \overline{F}_1 \frac{\partial x_{t+2}}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial x_t} + \overline{F}_2 \frac{\partial x_{t+1}}{\partial x_t} + \overline{F}_3$$

$$= \overline{F}_1 h_x^2 + \overline{F}_2 h_x + \overline{F}_3 = 0$$

- $\frac{\partial F(x_{t+2}, x_{t+1}, x_t, \sigma)}{\partial x_{t+i}} |_{x_{t+2} = x_{t+1} = x_t = \bar{x}, \sigma = 0} = \overline{F}_{3-i}$
- $\frac{\partial h(x_t, \sigma)}{\partial x_t} |_{x_t = \bar{x}, \sigma = 0 \ \forall t} = h_x$

Then, approximating polynomial: $h(x, \sigma) \approx h(\bar{x}, 0) + h_x(\bar{x}, 0)(x - \bar{x}) + h_\sigma(\bar{x}, 0)\sigma$

Getting 2nd-order approximations

$$
\begin{aligned}
h(x,\sigma) =& h(\bar{x},\bar{\sigma}) + h_x(\bar{x},\bar{\sigma})(x-\bar{x}) + h_\sigma(\bar{x},\bar{\sigma})(\sigma-\bar{\sigma}) \\
&+ 1/2[h_{xx}(\bar{x},\bar{\sigma})(x-\bar{x})^2 + 2h_{x\sigma}(\bar{x},\bar{\sigma})(x-\bar{x})(\sigma-\bar{\sigma}) \\
&+ h_{\sigma\sigma}(\bar{x},\bar{\sigma})(\sigma-\bar{\sigma})^2]
\end{aligned}
$$

Getting 2nd-order derivative w.r.t. x_t

$$
\begin{aligned}
F_{xx} = & \frac{\partial F_x}{\partial x_t} = h_x^2(\bar{F}_{11} h_x^2 + \bar{F}_{12} h_x + \bar{F}_{13}) \\
& + \bar{F}_1 2 h_x h_{xx} \\
& + h_x(\bar{F}_{21} h_x^2 + \bar{F}_{22} h_x + \bar{F}_{23}) \\
& + \bar{F}_2 h_{xx} \\
& + (\bar{F}_{31} h_x^2 + \bar{F}_{32} h_x + \bar{F}_{33}) = 0
\end{aligned}
$$

- $\frac{\partial^2 F(x_{t+2}, x_{t+1}, x_t, \sigma)}{\partial x_{t+i} \partial x_{t+j}}\Big|_{x_{t+2}=x_{t+1}=x_t=\bar{x}, \sigma=0} = \bar{F}_{3-i,3-j}$
- $\frac{\partial h(x_t, \sigma)}{\partial x_t^2}\Big|_{x_t=\bar{x}, \sigma=0 \ \forall t} = h_{xx}$

GETTING 2ND-ORDER DERIVATIVE W.R.T. x_t

- the above is *linear* in h_{xx}
- the same holds for higher-order derivatives
- i.e. easy to solve for coefficients of approximating polynomial
- but, careful with accuracy and in simulation (pruning)

Overview for today

1. ~~Higher-order perturbation~~
2. Projection
3. Value function iteration

Projection

Let's return to our favorite DSGE model

$$c_t^{-\gamma} = \beta \mathbb{E}_t c_{t+1}^{-1} \left(\alpha z_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta \right)$$

$$c_t + k_{t+1} = z_t k_t^{\alpha} + (1-\delta) k_t$$

$$\ln z_t = \rho \ln z_{t-1} + \epsilon_t$$

Policy rules

What are the policy rules?

$$c_t = c(k_t, Z_t)$$

$$k_{t+1} = k(k_t, Z_t)$$

How are they determined?

$$u_c(c_t) = \beta \mathbb{E}_t u_c(c_{t+1}) \left(\alpha Z_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta \right)$$

$$c_t + k_{t+1} = y_t + (1 - \delta) k_t$$

But we don't know what policy rules look like!

Function approximation

- analytical solutions rarely exist
- $\rightarrow$ need to approximate the policy functions

$$c_t \approx \widetilde{c}(k_t, Z_t; \psi_c)$$

$$k_{t+1} \approx \widetilde{k}(k_t, Z_t; \psi_k)$$

- what are we solving for?
  - the coefficients of the approximations: ψ_c and ψ_k

But we don't know what policy rules look like!

Numerical integration

- analytical solutions rarely exist
- $\rightarrow$ need to calculate expectations of (unknown) functions

$$u_c(c_t) = \beta \mathbb{E}_t u_c(c_{t+1}) \left[\alpha Z_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta \right]$$

But we don't know what policy rules look like!

Numerical integration

- analytical solutions rarely exist
- $\rightarrow$ need to calculate expectations of (unknown) functions

$$u_c(c_t) = \beta \int u_c(c_{t+1}) \left[\alpha Z_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta\right] dF(\epsilon)$$

Projection

Non-linear (global) solution method: "brute force" use of

- function approximation
- numerical integration

Projection

Function Approximation

Why function approximation?

- we are after policy rules
  - these are functions of state variables
  - moreover, closed form solutions rarely exist
  - $\rightarrow$ work with approximations of true functions
- let's think about this problem more generally first
- later we'll talk about how to implement it with DSGE models

Main idea of function approximation

Consider we want to approximate a function

$$y = f(x)$$

1. choose a family of functions to use as interpolants
   - popular choice is the family of polynomials
   - but others also exist
     - trigonometric functions (fourier approximation)
     - rational functions (pade approximation)

2. find coefficients of interpolant
   - such that interpolant and true function agree at certain points

Main idea of function approximation

We've already made enormous progress at this point:

- we have reduced the problem to a finite dimension!

$$y \approx \bar{f}(x) = a_0 T_0(x) + a_1 T_1(x) + ... + a_n T_n(x)$$

- where a_j are coefficients of the polynomial for $j = 0, .., n$
- and $T_j(x)$ are basis functions

a_j's solve the above at each chosen node, $i = 1, ..., n+1$

$$y_i = f(x_i) = a_0 + \sum_{j=1}^{n} a_j T_j(x_i)$$

$$y = T(x)a$$

Main idea of function approximation

- the above (interpolation) method is a special case
- regression is a method of interpolation!
  - when the number of grid points i
  - is generally larger than the number of basis functions
  - what if you run a regression of n points on n regressors?

Weierstrass' Approximation Theorem

Theorem *Let f be a continous real-valued function defined on the real interval $[a, b]$. For every $\epsilon > 0$, there exists a polynomial p such that for all $x \in [a, b]$ we have $|f(x) - p(x)| < \epsilon$.*

In other words

- there exists a polynomial
- that approximates any continuous function
- arbitrarily well

Problem with Weierstrass...

- it is useless from a practical point of view
- because it gives no guidance on how to find p

There are (at least) 2 important choices to be made

- what type of polynomial to use
- and where to evaluate it (grid points)

An example is using monomials and equidistant nodes

- turns out to be a bad idea

Projection

Function Approximation: Basis Functions

Why not monomials?

- the choice of monomial basis functions implies

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix}$$

- the first matrix on the RHS is a Vandermonde matrix
- even though it is non-singular, it is often ill-behaved
- intuition from regression?

Orthogonal polynomials

- monomials often suffer from high correlation
- orthogonal polynomials are constructed
- to have orthogonal basis functions (w.r.t. some measure)

$$\int_a^b T_i(x)T_j(x)w(x) = 0 \quad \forall i,j \quad i \neq j$$

  - $w(x)$ is some weighting function

Popular orthogonal polynomials are Chebyshev polynomials

Chebyshev polynomials

- defined on interval $[-1, 1]$
- weighting function

$$w(x) = \frac{1}{(1 - x^2)^{1/2}}$$

- basis functions

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{j+1}(x) = 2xT_j(x) - T_{j-1}(x) \quad j > 1$$

Projection

Function Approximation: Nodes

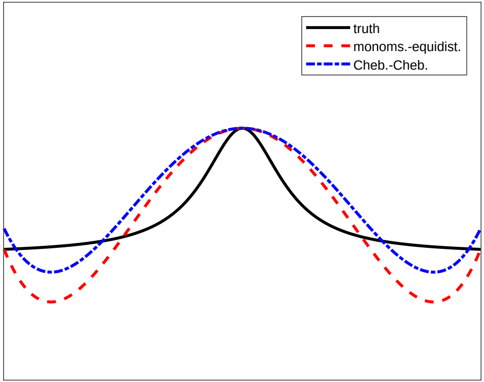
Why not an equidistant grid?

- suppose we have an equidistant grid
- it turns out that the higher the order of polynomial
- the larger the “swings” between grid points
- these oscillations become more dramatic at the end points!

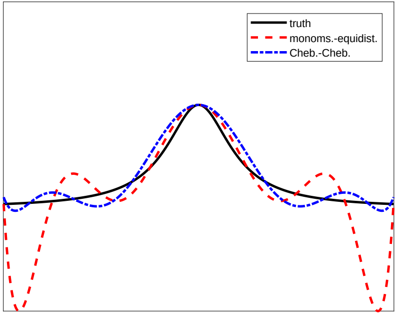
Weierstraas’ theorem

- there exists a uniformly converging polynomial approximation
- to find it, however, we have to be smart about the nodes

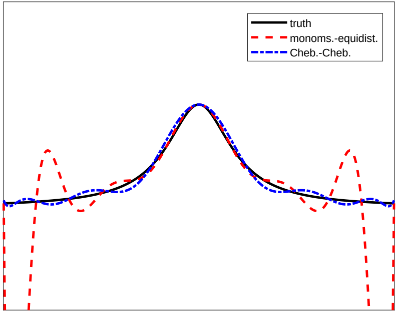
Why not an equidistant grid?



Why not an equidistant grid?



Why not an equidistant grid?



Intuition

Chebyshev nodes

Chebyshev nodes ensure uniform convergence

- the roots (z_j) at which the basis functions are equal to 0
- e.g. $T_2(x) = 2x^2 - 1 \rightarrow$ nodes of $z_1 = -1/2$ and $z_2 = 1/2$
- i.e. get n Chebyshev nodes by solving the nth basis function
- this is the reason for the popularity of Chebyshev polynomials
- Chebyshev nodes can be computed according to

$$z_j = -cos\left(\frac{(2j-1)\pi}{2n}\right)$$

Interval conversion

- to approximate on an interval $[a, b]$
- we must rescale the Chebyshev nodes
- find α and β for which

$$x = \alpha z + \beta$$

$$\beta - \alpha = a$$

$$\beta + \alpha = b$$

- then if $z \in [-1, 1]$ and $x \in [a, b]$ then

$$x = \frac{b-a}{2} z + \frac{a+b}{2}$$

Projection

Function Approximation: Splines

Splines - main idea

- polynomials approximate over entire domain
  - spectral method
- splines split support into sections
  - finite element method
- splines can be expressed as linear combinations of basis fces
- but they are not polynomials
  - basis functions are zero for most of the domain

Piece-wise linear splines

- the easiest type is piece-wise linear

$$f(x) \approx \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right) f_i + \left(\frac{x - x_i}{x_{i+1} - x_i}\right) f_{i+1} \quad x \in [x_i, x_{i+1}]$$

- in general not differentiable at nodes
- could be problematic $\rightarrow$ use higher-order polynomials

Cubic splines

- fit a 3-rd order polynomial in each segment

$$f(x) \approx a_i + b_i x + c_i x^2 + d_i x^3 \quad x \in [x_i, x_{i+1}]$$

$$S(x) = \begin{cases} S_1(x) & x_0 \leq x \leq x_1 \\ S_i(x) & x_{i-1} \leq x \leq x_i \\ S_n(x) & x_{n-1} \leq x \leq x_n \end{cases}$$

- i.e. we have n separate cubic splines for $n+1$ nodes

Cubic splines

- n splines give $4n$ coefficients to determine
- what conditions pin down the $4n$ coefficients?
  - $2 + 2(n-1)$ function values at the nodes
  - $2(n-1)$ smoothness conditions (for $0 < i < n$)

$$S_i'(x_i) = S_{i+1}'(x_i)$$

$$S_i''(x_i) = S_{i+1}''(x_i)$$

  - 2 boundary conditions
    - "natural (simple)" $S_1''(x_0) = 0$ and $S_n''(x_n) = 0$
    - or "clamped" $S_1'(x_0) = 0$ and $S_n'(x_n) = 0$

Before we move on...

- sovling DSGE models means getting policy functions
- so how does function approximation work in DSGE models?

Projection

Numerical integration

Why numerical integration?

- in economics, there are plenty of integrals
    - expectations
- evaluating integrals can be a tough problem
    - the functional form may be nasty
    - we may not even have the functional form
    - we may be able to evaluate it, but not draw from it
        - as in e.g. Bayesian estimation
- therefore, we need a way around this...

Intuitive integration method

Consider you want to compute the integral $\int_a^b g(x)dH(x)$

- where x is a random variable with CDF $H(x)$

Monte-Carlo integration uses the following approximation

$$\int_a^b g(x)dH(x) \approx \frac{\sum_{t=1}^T g(x_t)}{T}$$

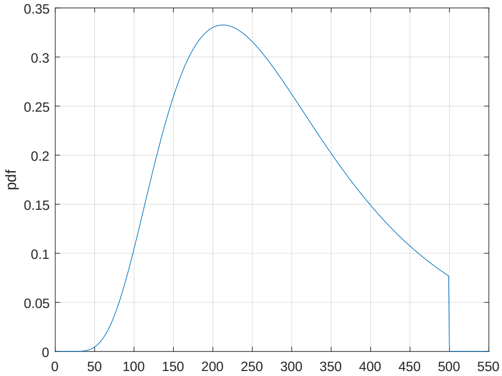
- $\{x_t\}_{t=1}^T$ is a series drawn from a random number generator

This procedure is very simple and intuitive but

- it is not very accurate (fast)
- more powerful procedures available $\rightarrow$ numerical integration

Main idea of numerical integration

- we want to calculate $I = \int_a^b f(x)dx$
- the basic idea is to approximate it with $I \approx \sum_{i=1}^{n} \omega_i f(x_i)$



Main idea of numerical integration

$$I \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

- therefore, we face (at least) three choices
  - choice of quadrature weights, ω_i's
  - choice of quadrature nodes, x_i's
  - choice of number of evaluations, n

Types of quadrature methods

Newton-Cotes quadrature methods

- break interval into equidistant intervals
- approximate $f(x)$ with a low-order polynomial
- use integrals of polynomials as the approximations

Gaussain quadrature methods

- same idea as with Newton-Cotes
- more clever in choosing quadrature nodes

Projection

Numerical Integration: Newton-Cotes

Newton-Cotes

Types of Newton-Cotes quadrature methods

- mid-point rule
  - interpolant is a constant
- trapezoid rule
  - interpolant is linear
- Simpson's rule
  - interpolant is quadratic

Simpson's quadrature

$$I \approx \sum_i \omega_i P_2(x_i)$$

- choose 3 equidistant nodes in (each) interval $[a, b]$:
    - $x_0 = a, x_1 = a + h$ and $x_2 = a + 2h = b$
- choose polynomial type to approximate $f(x)$
    - Simpson's quadrature uses Lagrange polynomials
    - the beauty is that Simpson's rule can be standardized!

Lagrange basis functions

$$P_n(x) = a_0 L_0(x) + \ldots + a_n L_n(x)$$
$$L_j(x) = \frac{(x - x_0)\ldots(x - x_{j-1})(x - x_{j+1})\ldots(x - x_n)}{(x_j - x_0)\ldots(x_j - x_{j-1})(x_j - x_{j+1})\ldots(x_j - x_n)}$$

- L_j's are polynomials so the approximation is a polynomial
- the approximation gives an exact fit at the $n+1$ nodes

$$L_j(x) = \begin{cases} 1 & \text{if} \quad x = x_j \\ 0 & \text{if} \quad x \in \{x_0, \ldots, x_n\} \setminus x_j \end{cases}$$

- what are the coefficients of the polynomial?

Simpson's quadrature

$$I \approx \int_a^b \left(f_0 L_0(x) + f_1 L_1(x) + f_2 L_2(x)\right) dx$$
$$= f_0 \int_a^b L_0(x)dx + f_1 \int_a^b L_1(x)dx + f_2 \int_a^b L_2(x)dx$$

- doing the integration results in

$$\int_a^b L_0(x)dx = 1/3h \quad \int_a^b L_1(x)dx = 4/3h \quad \int_a^b L_2(x)dx = 1/3h$$

  - i.e. you can find quadrature weights
  - *independent* of the functional form of f!

The above can be easily extended to $n+1$ equidistant nodes

- total number of nodes must be odd
- this gives us $n/2$ segments of lenght h
- apply the above idea for each of the segments

$$\int_a^b f(x)dx \approx \left(\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \frac{2}{3}f_4 + \cdots + \frac{2}{3}f_{n-2} + \frac{4}{3}f_{n-1} + \frac{1}{3}f_n\right)h$$

Projection

Numerical Integration: Gaussian Quadrature

Gaussian quadrature

- Newton-Cotes formulas are simple due to equidistant nodes
- moreover, one can show that with Newton-Cotes
  - we get the exact answer when the true function
  - is a polynomial of order $n-1$
- but we can get more accuracy by choosing nodes cleverly
  - we get the exact answer if the true function
  - is a polynomial of order $2n-1$!
  - i.e. 5 nodes give exact (accurate) answers for true functions
  - which are (approximated by) a 9th order polynomial!

Procedure of Gaussian quadrature

Using n nodes, we can approximate $\int_{-1}^{1} f(x)dx$ as

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} \omega_i f(\zeta_i)$$

- i.e. we have $2n$ parameters
- need $2n$ conditions to pin down our parameters
- $\rightarrow$ ensure correct answer for first $2n$ basis functions

$$1, x, x^2, \cdots, x^{2n-1}$$

Procedure of Gaussian quadrature

How to choose weights and nodes?

- solve for ω_i and $\zeta_i \quad \forall i$ s.t.

$$\int_{-1}^{1} x^j dx = \sum_{i=1}^{n} \omega_i \zeta_i^j \quad j = 0, 1, \cdots, 2n-1$$

- i.e. solve a system of $2n$ equations in $2n$ unknowns
- note that the solution is independent of f!
    - i.e. choice of nodes and weights is independent of f

Gauss-Legendre quadrature

- the above method (for interval between -1 and 1)
- is called Gauss-Legendre quadrature
- nodes (ζ_i^{GL}) and weights (ω_i^{GL}) satisfy above $2n$ conditions
- the approximation is then given by

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} \omega_i^{GL} f(\zeta_i^{GL})$$

Gauss-Hermite quadrature

- when the true function is given by $f(x) = g(x)W(x)$ where
  - $g(x)$ can be approximated well by a polynomial
  - but $f(x)$ cannot
- then adjust the quadrature procedure depending on $W(x)$
- Gauss-Hermite quadrature is used when $W(x) = \exp(-x^2)$
- why is this an interesting case?

Gauss-Hermite quadrature

- nodes and weights chosen s.t.

$$\int_{-\infty}^{\infty} x^j \exp(-x^2)dx = \sum_{i=1}^{n} \omega_i \zeta_i^j \quad j = 0, 1, \cdots, 2n-1$$

- the approximation is then given by

$$\int_{-\infty}^{\infty} g(x) \exp(-x^2)dx \approx \sum_{i=1}^{n} \omega_i^{GH} g(\zeta_i^{GH})$$

- often, we need to revert to “change of variable” to convert original problem
    - why?

Projection

Main Idea

Projection: main idea

Policy rules are

- (unknown) functions of state variables
- $\rightarrow$ use function approximation and numerical integration

True rational expectations solution given by:

$$c_t = c(k_t, Z_t)$$

$$k_{t+1} = k(k_t, Z_t)$$

Approximate $c(k_t, Z_t)$ with polynomial $P_n(k_t, Z_t; \psi_n)$

- what about $k(k_t, Z_t)$?

Projection: main idea

- what are we solving for?
- how do we do it?

Define error terms

$$e(k_t, Z_t) = -c_t^{-\gamma} + \mathbb{E}_t[\beta c_{t+1}^{-\gamma} \alpha Z_{t+1} k_{t+1}^{\alpha-1}]$$

- substitute c_t with $P_n(k_t, Z_t; \psi_n)$
- there is N_n elements of ψ_n but only one Euler equation...

Projection

Details

Details of the setup

Define M grid points $\{k_i, Z_i\}$

$$e(k_i, Z_i; \psi_n) = -P_n(k_i, Z_i; \psi_n)^{-\gamma} + \alpha\beta\mathbb{E}\left[\begin{array}{c} P_n(k', Z'; \psi_n)^{-\gamma} \times \\ Z' \times \\ (k')^{\alpha-1} \end{array}\right]$$

- but what about k' and Z'?

Details of the setup

$$
e(k_i, Z_i; \psi_n) = -P_n(k_i, Z_i; \psi_n)^{-\gamma} +
$$

$$
\mathbb{E}\left[\begin{array}{c}
\alpha\beta\times \\
P_n\left(Z_i k_i^{\alpha} - P_n(k_i, Z_i; \psi_n), \exp(\rho \ln(Z_i) + \epsilon'); \psi_n\right)^{-\gamma} \times \\
\exp(\rho \ln(Z_i) + \epsilon')\times \\
(Z_i k_i^{\alpha} - P_n(k_i, Z_i; \psi_n))^{\alpha-1}
\end{array}\right]
$$

- but what about ϵ'?

Details of the setup

$$e(k_i, Z_i; \psi_n) = -P_n(k_i, Z_i; \psi_n)^{-\gamma} +$$

$$\sum_{j=1}^{J} \frac{\omega_j}{\sqrt{\pi}} \left[\begin{array}{c} \alpha\beta\times \\ P_n\left(Z_i k_i^{\alpha} - P_n(k_i, Z_i; \psi_n), \exp(\rho \ln(Z_i) + \sqrt{2}\sigma\zeta_j); \psi_n\right)^{-\gamma} \times \\ \exp(\rho \ln(Z_i) + \sqrt{2}\sigma\zeta_j)\times \\ (Z_i k_i^{\alpha} - P_n(k_i, Z_i; \psi_n))^{\alpha-1} \end{array} \right]$$

- ω_j and ζ_j are Gauss-Hermite quadrature weights and nodes

Computation vs iteration

How to solve for coefficients of P_n?

- equation solver/minimization routine
- iteration procedures
    - fixed-point iteration
    - time-iteration

How to solve for coefficients of P_n?

How to choose polynomial and grid points?

- use Chebyshev nodes
  - guaranteed uniform convergence
- use Chebyshev polynomials
  - especially useful for iteration procedures
  - rescaling is needed (defined only between -1 and 1)

Solvers and minimization routines

Smart in updating ψ_n, but high-dimensions costly

- Collocation: $M = N_n$
  - use equation solver to obtain ψ_n at which $e(k_i, Z_i; \psi_n) = 0 \quad \forall i$
- Galerkin: $M > N_n$
  - use minimization routine to obtain ψ_n
  - minimize $e(k_i, Z_i; \psi_n)$

Can deal with high N_n, sometimes guaranteed to converge

- in both fixed-point and time-iteration
  1. use latest "guess" of ψ_n in Eurler equation and compute implied c_i
  2. use c_i values from 1 to get new guess of ψ_n
  3. update your guess of coefficients ψ_n
- difference between fixed-point and time-iteration
- is in implementation of 1

Fixed-point iteration

Define ψ_n^q as value of ψ_n in qth iteration

1. At each grid point calculate c_i using ψ_n^{q-1}

$$c_i^{-\gamma} = \sum_{j=1}^{J} \frac{\omega_j}{\sqrt{2}} \begin{bmatrix} \alpha\beta\times \\ P_n\left(Z_i k_i^\alpha - P_n(k_i, Z_i; \psi_n^{q-1}), \exp(\rho \ln(Z_i) + \sqrt{2}\sigma\zeta_j); \psi_n^{q-1}\right)^{-\gamma} \times \\ \exp(\rho \ln(Z_i) + \sqrt{2}\sigma\zeta_j)\times \\ (Z_i k_i^\alpha - P_n(k_i, Z_i; \psi_n^{q-1}))^{\alpha-1} \end{bmatrix}$$

2. Use obtained c_i's to get new guess of ψ_n
   - e.g. for $n = 2$, define

$$X = \begin{bmatrix} 1 & k_1 & Z_1 & k_1^2 & k_1 Z_1 & Z_1^2 \\ 1 & k_2 & Z_2 & k_2^2 & k_2 Z_2 & Z_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & k_M & Z_M & k_M^2 & k_M Z_M & Z_M^2 \end{bmatrix}$$

   - compute $\hat{\psi}_n^q = (X'X)^{-1}X'Y$, where
   - $Y = (c_1, c_2, ..., c_M)$ from step 1

Fixed-point iteration

3. Use past guess and newly estimated values for ψ_n for new guess
   - typically making slower steps is more stable:

$$\psi_n^q = \lambda \psi_n^{q-1} + (1-\lambda)\hat{\psi}_n^q$$

   - $\lambda \in [0, 1)$
     - high values of λ increase chances of convergence
     - but they also slow things down

Time-iteration

Basic idea is the same as with fixed-point iteration

- 1. use latest "guess" of coefficients ψ_n in Eurler equation $\rightarrow c_i$
- 2. use c_i values from 1 to get new guess of ψ_n
- 3. update your guess of coefficients ψ_n

But this time use latest guess of ψ_n only

- for next period's choices
  - makes the solution of c_i trickier
  - guarantees convergence (under conditions similar to VFI)

There is something slightly inconsistent with fixed-point iteration:

$$c_i^{-\gamma} = \sum_{j=1}^{J} \frac{\omega_j}{\sqrt{2}} \left[\begin{array}{c} \alpha\beta\times \\ P_n\left(Z_i k_i^{\alpha} - P_n(k_i, Z_i; \psi_n^{q-1}), \exp(\rho \ln(Z_i) + \sqrt{2}\sigma\zeta_j); \psi_n^{q-1}\right)^{-\gamma} \times \\ \exp(\rho \ln(Z_i) + \sqrt{2}\sigma\zeta_j)\times \\ (Z_i k_i^{\alpha} - P_n(k_i, Z_i; \psi_n^{q-1}))^{\alpha-1} \end{array} \right]$$

Time iteration uses ψ_n^{q-1} *only for next period's choices*!

$$c_i^{-\gamma} = \sum_{j=1}^{J} \frac{\omega_j}{\sqrt{2}} \left[\begin{array}{c} \alpha\beta\times \\ P_n\left(Z_i k_i^\alpha - c_i, \exp(\rho\ln(Z_i) + \sqrt{2}\sigma\zeta_j); \psi_n^{q-1}\right)^{-\gamma} \times \\ \exp(\rho\ln(Z_i) + \sqrt{2}\sigma\zeta_j)\times \\ (Z_i k_i^\alpha - c_i)^{\alpha-1} \end{array} \right]$$

Homotopy

It's always important to have good starting conditions

- begin with a point with good starting values
- solve for that setup
- adjust slowly towards desired (final) setup
- use solution from previous step as new starting values

Taking Stock

Projection is a brute-force application of

- function approximation
  - careful choice of nodes and polynomial types
- numerical integration
  - careful choice of nodes, type of quadrature
- time vs fixed-point iteration

Overview for today

1. ~~Higher-order perturbation~~
2. ~~Projection~~
3. Value function iteration

Value Function Iteration

Our model in “Bellman form”

We can write our neoclassical growth model as

$$V(z,k) = \max_{c,k'} u(c) + \beta \mathbb{E} V(z',k')$$

$$c + k' = zk^{\alpha}$$

$$z' = 1 - \rho + \rho z + \epsilon$$

$$k_0, z_0 \ \text{ given}, \epsilon \sim N(0,\sigma^2)$$

Our model in "Bellman form"

More generally, we're looking for a value function, V(x), i.e. the solution to

$$V(x) = \max_u r(x, u) + \beta V(x')$$

$$u = g(x)$$

$$x' = h(x, u)$$

- $r(x, u)$: payoff function
- $g(x)$: policy rule which maps states x into controls (u)
- $h(x, u)$: law of motion for states
- we've ignored uncertainty, but it all carries over also to stochastic case

Solving for V

$$V(x) = \max_u r(x, u) + \beta V(h(x, u))$$

Define the Bellman operator B

- maps any function V into a new function BV
- $BV(x) = \max_u r(x, u) + \beta V(h(x, u))$

If $V(x)$ is the solution to the Bellman equation

- then it is the fixed point of B
- i.e. B maps V into V

Solving for V

The name suggests that we will repeatedly apply B

$$
\begin{aligned}
V_1(x) =& BV_0(x) = \max_u r(x,u) + \beta V_0(h(x,u)) \\
V_2(x) =& B(BV_0)(x) = \max_u r(x,u) + \beta V_1(h(x,u)) \\
& \dots \\
V_n(x) =& B^n V_0(x) = \max_u r(x,u) + \beta V_{n-1}(h(x,u))
\end{aligned}
$$

If $V(x)$ is the solution to the Bellman equation

- then it is the fixed point of B
- V_n will converge to the true value function V!
- $\lim_{n\to\infty} B^n V_0 = V$
- this happens if B is a contraction mapping

Underlying theory

Dynamic programming comes with some powerful theory

- unlike many other solution methods, VFI comes with theoretical results
- existence, uniqueness, convergence etc (of course, under certain conditions)

Value Function Iteration

Implementation

Different ways of solving for V

1. Guess and verify
2. Value function iteration
    - Basic algorithm
    - Some speed improvements

Guess and verify

As the name suggests, not greatly sophisticated

- But can still be powerful

General steps

1. Set up Bellman equation
2. Derive optimality conditions
3. Guess function form of value function
4. Verify guess in optimality conditions (and derive coefficients)

Usually, closed form solutions to Bellman equation don't exist

1. discrete-state approximations
    - force state vector to lie on a finite and discrete grid
    - solve numerically for value function
2. smooth approximations
    - use function approximation (e.g. polynomials)
    - to numerically solve for the value function

Consider the neoclassical growth model

$$V(k,z) = \max_{c,k'} U(c) + \beta \mathbb{E} V(k', z')$$

$$\text{s.t.} c + k' = zf(k) + (1-\delta)k$$

$$z' = (1-\rho)\bar{z} + \rho z + \epsilon'$$

$$\epsilon \sim N(0, \sigma^2)$$

$$c, k \geq 0$$

$$k_0 \quad \text{given}$$

- with anything but log-utility and $\delta = 1$
- $\rightarrow$ need to approximate $V(k)$ numerically

Consider the neoclassical growth model

$$V(k,z) = \max_{k'} U(zf(k) + (1-\delta)k - k') + \beta \mathbb{E} V(k', z')$$

$$z' = (1-\rho)\bar{z} + \rho z + \epsilon'$$

$$\epsilon \sim N(0, \sigma^2)$$

$$c, k \geq 0$$

$$k_0 \quad \text{given}$$

- with anything but log-utility and $\delta = 1$
- $\rightarrow$ need to approximate $V(k)$ numerically

Discrete-state approximations: grid

Approximate value function V with N function values

- how to choose grid points for k?
  - ideally, choose high N, but time is finite!
  - moreover, tougher with more dimensions (state variables)
- what are the bounds $\underline{k}$ and $\overline{k}$?
- equidistant vs other spacing?
  - where to put denser grid?
  - grid in levels or logs of capital?

Discrete-state approximations: shocks

Approximate value function V with N function values

- in computing value function, we need

$$\mathbb{E}V(k', z') = \int V(k', z')h(z'|z)dz'$$

- how to discretize stochastic process of z?
- replace continuous Markov chain with a discrete one $\widetilde{z}$
  - takes on values from finite set $Z = \{z_1, z_2, ..., z_m\}$ with
  - transition matrix P with elements $p_{i,j} = Prob(\widetilde{z}' = z_j | \widetilde{z} = z_i)$

$$\mathbb{E}V(k', \widetilde{z}') = \sum_{j=1}^{m} p_{i,j} V(k', z_j')$$

But how to choose Z and P?

Discretizing shocks: Tauchen (1986)

Based on fact that given z_i

$$z' \sim N(\mu_{z_i}, \sigma^2)$$

$$\mu_{z_i} = (1-\rho)\bar{z} + \rho z_i$$

- choose m equally spaced values between
- $z_1 = \bar{z} - k\sigma_z$ and $z_m = \bar{z} + k\sigma_z$
  - $\sigma_z = \sigma/\sqrt{1-\rho^2}$ is the unconditional st. deviation of z
- in interior, define $w = z_j - z_{j-1}$, and set

$$p_{i,j} = Pr[z_j - w/2 \leq \mu_{z_i} + \epsilon \leq z_j + w/2]$$

- at end-points set

$$p_{i,1} = Pr[\mu_{z_i} + \epsilon \leq z_1 + w/2], \quad p_{i,m} = 1 - Pr[z_m - w/2 \leq \mu_{z_i} + \epsilon]$$

Discretizing shocks: Tauchen (1986)

This procedure amounts to setting

$$
p_{i,j} = \begin{cases} \Phi\left(\frac{z_1 - w/2 - \mu_{z_i}}{\sigma}\right) & \text{for} j = 1, \\ \Phi\left(\frac{z_j + w/2 - \mu_{z_i}}{\sigma}\right) - \Phi\left(\frac{z_j - w/2 - \mu_{z_i}}{\sigma}\right) & \text{for} 1 < j < m, \\ 1 - \Phi\left(\frac{z_m - w/2 - \mu_{z_i}}{\sigma}\right) & \text{for} j = m. \end{cases}
$$

Clearly, the precision of this discrete approximation rises with m

Value Function Iteration

Algorithm

Value function iteration algorithm

1. choose an error tolerance **e**
2. discretize state space
   - $k = \{k_1, k_2, ..., k_n\}$, $z = \{z_1, z_2, ..., z_m\}$
3. guess initial value function $V^{(0)}(k, z)$
   - function values at grid pairs $\{k_i, z_j\}$, $i = 1, ..., n$, $j = 1, ..., m$
4. update value function using

$$V^{(l+1)}(k, z) = \max_{k'} U(zf(k) + (1-\delta)k - k') + \beta \mathbb{E} V^{(l)}(k', z')$$

   - for each grid pair i, j store max of RHS as new guess
   - remember to enforce $c \geq 0$
5. compute distance, e.g. $d = \max_{i,j} |V_{i,j}^{(l+1)} - V_{i,j}^{(l)}|$
6. stop if $d \leq \mathbf{e}$, otherwise go back to 4 with new guess.

Value function iteration algorithm

Things to keep in mind and avoid

- is the grid too constrictive?
- is the error tolerance too large?
- is the number of grid points too small?

And remember, a good initial guess always goes a long way!

Value Function Iteration

Some speed improvements

There are several ways to increase computation speed

- utilize concavity of value function
    - $\rightarrow$ unique maximum
    - once you find a maximum, stop looking!
- monotonicity of the policy function $k(k_i, z_j)$
    - $\rightarrow k(k_i, z_j) \leq k(k_{i+1}, z_j)$ for $k_i < k_{i+1}$
    - don't look at unnecessary grid points!

Speed improvements: Howard's algorithm

Policy function tends to converge faster than the value function

- use this fact in speeding up VFI

For a given value function guess $V^{(l)}$

- $k^*(k_i, z_j) = \underset{k'}{\arg\max}\; U(zf(k) + (1-\delta)k - k') + \beta \mathbb{E} V^{(l)}(k', z')$
- $c^*(k_i, z_j) = zf(k) + (1-\delta)k - k^*(k_i, z_j)$

Speed improvements: Howard's algorithm ... continued

In between steps 4 and 5 above

- keep the same policy function and iterate on:

$$V^{(l+1)} = U(c^*(k,z)) + \beta \mathbb{E} V^{(l)}(k^*(k,z),z')$$

- notice there is no maximization! (most computationally expensive part)
- can solve in one step as $V^{(\infty)} = (I - \beta P)^{-1} U(c^*(k,z))$

The above is called Howard's Improvement Algorithm

Taking Stock

Value function iteration

- powerful, global, solution method
- has theory to back its convergence (under some conditions)
- can handle various non-linearities, but curse of dimensionality

Overview for today

1. ~~Higher-order perturbation~~
2. ~~Projection~~
3. ~~Value function iteration~~


That's all Folks!