Function Approximation

Wouter J. Den Haan
London School of Economics

© by Wouter J. Den Haan

Goal

Obtain an approximation for

$$f(x)$$

when

- $f(x)$ is unknown, but we have some information, or
- $f(x)$ is known, but too complex to work with

Information available

- **Either** finite set of derivatives
  - usually at one point
- **or** finite set of function values
  - $f_1, \cdots, f_m$ at m nodes, $x_1, \cdots, x_m$

Classes of approximating functions

1. polynomials
   - this still gives lots of flexibility
   - examples of second-order polynomials
     - $a_0 + a_1 x + a_2 x^2$
     - $a_0 + a_1 \ln(x) + a_2 (\ln(x))^2$
     - $\exp\left(a_0 + a_1 \ln(x) + a_2 (\ln(x))^2\right)$

2. splines, e.g., linear interpolation

Classes of approximating functions

- Polynomials and splines can be expressed as

$$f(x) \approx \sum_{i=0}^{n} \alpha_i T_i(x)$$

- $T_i(x)$: the *basis functions* that define the *class* of functions used, e.g., for regular polynomials:

$$T_i(x) = x^i.$$

- α_i : the coefficients that pin down the particular approximation

Reducing the dimensionality

unknown $f\left(x\right)$: infinite dimensional object

$\sum_{i=0}^{n} \alpha_i T_i(x)$: $n+1$ elements

General procedure

- Fix the order of the approximation n
- Find the coefficients $\alpha_0, \cdots, \alpha_n$
- Evaluate the approximation
- If necessary, increase n to get a better approximation

Weierstrass (sloppy definition but true)

Let $f : [a,b] \longrightarrow \mathbb{R}$ be any real-valued function. For large enough n, it is approximated arbitrarily well with the polynomial

$$\sum_{i=0}^{n} \alpha_i x^i.$$

Thus, we can get an accurate approximation if

- f is not a polynomial
- f is discontinuous

How can this be true?

How to find the coefficients of the approximating polynomial?

- With derivatives:
  - use the Taylor expansion
- With a set of points (nodes), $x_0, \cdots, x_m$, and function values, $f_0, \cdots, f_m$?
  - use projection
  - Lagrange way of writing the polynomial (see last part of slides)

Function fitting as a projection

Let

$$Y = \begin{bmatrix} f_0 \\ \vdots \\ f_m \end{bmatrix}, X = \begin{bmatrix} T_0(x_0) & T_1(x_0) & \cdots & T_n(x_0) \\ T_0(x_1) & T_1(x_1) & \cdots & T_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ T_0(x_m) & T_1(x_m) & \cdots & T_n(x_m) \end{bmatrix}$$

then

$$Y \approx X\alpha$$

- We need $m \geq n+1$. Is $m = n+1$ as bad as it is in empirical work?
- What problem do you run into if n increases?

Orthogonal polynomials

- Construct basis functions so that they are orthogonal to each other, i.e.,

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

- This requires a particular weighting function (density), $w(x)$, and range on which variables are defined, $[a,b]$

Chebyshev orthogonal polynomials

- $$[a,b] = [-1,1] \text{ and } w(x) = \frac{1}{(1-x^2)^{1/2}}$$
- What if function of interest is not defined on $[-1,1]$?

Constructing Chebyshev polynomials

- The basis functions of the Chebyshev polynomials are given by

$$\begin{aligned} T_0^c(x) &= 1 \\ T_1^c(x) &= x \\ T_{i+1}^c(x) &= 2xT_i^c(x) - T_{i-1}^c(x) \quad i > 1 \end{aligned}$$

Chebyshev versus regular polynomials

- Chebyshev polynomials, i.e.,

$$f(x) \approx \sum_{j=0}^{n} a_j T_j^c(x),$$

can be rewritten as regular polynomials, i.e.,

$$f(x) \approx \sum_{j=0}^{n} b_j x^j,$$

Chebyshev nodes

- The $n^{\text{th}}-$order Chebyshev basis function has n solutions to

$$T_n^c(x) = 0$$

- These are the n Chebyshev nodes

Discrete orthogonality property

- Evaluated at the Chebyshev nodes, the Chebyshev polynomials satisfy:

$$\sum_{i=1}^{n} T_j^c(x_i) T_k^c(x_i) = 0 \text{ for } j \neq k$$

- Thus, if

$$X = \begin{bmatrix} T_0(x_0) & T_1(x_0) & \cdots & T_n(x_0) \\ T_0(x_1) & T_1(x_1) & \cdots & T_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ T_0(x_m) & T_1(x_m) & \cdots & T_n(x_m) \end{bmatrix}$$

then $X'X$ is a diagonal matrix

Uniform convergence

- Weierstrass $\Longrightarrow$ there is a good polynomial approximation
- Weierstrass $\not\Rightarrow f(x) = \lim_{n\to\infty} p_n(x)$ for every sequence $p_n(x)$
- If polynomials are fitted on Chebyshev nodes$\Longrightarrow$ even *uniform* convergence is guaranteed

Splines

Inputs:

1. $n+1$ nodes, $x_0, \cdots, x_n$
2. $n+1$ function values, $f(x_0) \cdots, f(x_n)$

- nodes are fixed $\Longrightarrow$ the $n+1$ function values are the *coefficients* of the spline

Piece-wise linear

- For $x \in [x_i, x_{i+1}]$

$$f(x) \approx \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right) f_i + \left(\frac{x - x_i}{x_{i+1} - x_i}\right) f_{i+1}.$$

- That is, a separate linear function is fitted on the n intervals
- Still it is easier/better to think of the coefficients of the approximating function as the $n + 1$ function values

Piece-wise linear versus polynomial

- Advantage: Shape preserving
  - in particular monotonicity & concavity (strict?)
- Disadvantage: not differentiable

Extra material

1. Lagrange interpolation
2. Higher dimensional polynomials
3. Higher-order splines

Lagrange interpolation

Let

$$L_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} \text{ and}$$

$$f(x) \approx f_0 L_0(x) + \cdots + f_n L_n(x).$$

- Right-hand side is an n^{th}-order polynomial
- By construction perfect fit at the $n+1$ nodes?
- $\Longrightarrow$ the RHS is the n^{th}-order approximation

Higher-dimensional functions

- second-order *complete* polynomial in x and y:

$$\sum_{0\leq i+j\leq 2} a_{i,j}x^iy^j$$

- second-order *tensor product* polynomial in x and y:

$$\sum_{i=0}^{2}\sum_{j=0}^{2} a_{i,j}x^iy^j$$

Complete versus tensor product

- tensor product can make programming easier
    - simple double loop instead of condition on sum

- n^{th} tensor has higher order term than $(n+1)^{\text{th}}$ complete
    - 2^{nd}-order tensor has fourth-order power
    - at least locally, lower-order powers are more important $\Longrightarrow$ complete polynomial may be more efficient

Higher-order spline

Cubic (for example)

- !!! Same inputs as with linear spline, i.e. $n+1$ function values at $n+1$ nodes which can still be thought of as the $n+1$ coefficients that determine approximating function
- Now fit 3rd-order polynomials on each of the n intervals

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

What conditions can we use to pin down these coefficients?

Cubic spline conditions: levels

- We have $2+2(n-1)$ conditions to ensure that the function values correspond to the given function values at the nodes.
  - For the intermediate nodes we need that the cubic approximations of both adjacent segments give the correct answer. For example, we need that

$$\begin{aligned} f_1 &= a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3 \text{ and} \\ f_1 &= a_2 + b_2 x_1 + c_2 x_1^2 + d_2 x_1^3 \end{aligned}$$

  - For the two endpoints, x_0 and x_{n+1}, we only have one cubic that has to fit it correctly.

Cubic spline conditions: 1st-order derivatives

- To ensure differentiability at the intermediate nodes we need

  $b_i + 2c_i x_i + 3d_i x_i^2 = b_{i+1} + 2c_{i+1} x_i + 3d_{i+1} x_i^2$ for $x_i \in \{x_1, \cdots, x_{n-$

  which gives us $n-1$ conditions.

Cubic spline conditions: 2nd-order derivatives

- To ensure that second derivatives are equal we need

$$2c_i + 6d_i x_i = 2c_{i+1} + 6d_{i+1} x_i \text{ for } x_i \in \{x_1, \cdots, x_{n-1}\}.$$

- We now have $2 + 4(n-1) = 4n - 2$ conditions to find $4n$ unknowns.
- We need two additional conditions; e.g. that 2nd-order derivatives at end points are zero.

Splines - additional issues

- (standard) higher-order splines do not preserve shape
- higher-order difficult for multi-dimensional problems
- first-order trivial for multi-dimensional problems
    - if interval is small then nondifferentiability often doesn't matter

References

- Den Haan, W.J., Numerical Integration, online lecture notes.
- Heer, B., and A. Maussner, 2009, Dynamic General Equilibrium Modeling.
- Judd, K. L., 1998, Numerical Methods in Economics.
- Miranda, M.J, and P.L. Fackler, 2002, Applied Computational Economics and Finance.

Numerical Integration

Wouter J. Den Haan
London School of Economics

© by Wouter J. Den Haan

Quadrature techniques

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) = \sum_{i=1}^{n} w_i f_i$$

- Nodes: x_i
- Weights: w_i

Quadrature techniques

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

Two versions:

- Newton Cotes:
  - equidistant nodes & "best" choice for the weights w_i
- Gaussian Quadrature:
  - "best" choice for both nodes and weights

Monte Carlo techniques

- pseudo:
  - implemetable version of true Monte Carlo
- quasi:
  - looks like Monte Carlo, but is something different
    - name should have been chosen better

Power

- Newton-Cotes: With n nodes you get
  - exact answer if f is $(n-1)^{\text{th}}$-order polynomial
  - accurate answer f is close to an $(n-1)^{\text{th}}$-order polynomial
- Gaussian: With n nodes you get
  - exact answer if f is $(2n-1)^{\text{th}}$-order polynomial
  - accurate answer f is close to a $(2n-1)^{\text{th}}$-order polynomial

Power

- (Pseudo) Monte Carlo: accuracy requires lots of draws
- Quasi Monte Carlo: definitely better than (pseudo) Monte Carlo and dominates quadrature methods for higher-dimensional problems

Idea behind Newton-Cotes

- function values at n nodes $\Longrightarrow$ you can fit a $(n-1)^{\text{th}}$-order polynomial & integrate the approximating polynomial

$$\int_a^b f(x)dx \approx \int_a^b P_2(x)dx$$

- It turns out that this can be standardized
  - (derivation at the end of these slides)

Simpson with 3 nodes

$$\int_a^b f(x)dx \approx \left(\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{1}{3}f_2\right) h$$

Simpson with n+1 nodes

Implement this idea over many (small) intervals we get:

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

$$= \left(\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \frac{2}{3}f_4 + \cdots \frac{2}{3}f_{n-2} + \frac{4}{3}f_{n-1} + \frac{1}{3}f_n\right)h$$

Simpson in Matlab

- Integration routine in Matlab

```
quad(@myfun,A,B)
```

- This is an adaptive procedure that adjusts the length of the interval (by looking at changes in derivatives)

Gaussian quadrature

- Could we do better? That is, get better accuracy with same amount of nodes?
- **Answer:** Yes, if you are smart about choosing the nodes
    - This is Gaussian quadrature

Gauss-Legendre quadrature

- Let $[a,b]$ be $[-1,1]$
  - can always be accomplished by scaling
- Quadrature

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

- **Goal:** Get exact answer if $f(x)$ is a polynomial of order $2n-1$
- That is with 5 nodes you get exact answer even if $f(x)$ is a 9th-order polynomial

Implementing Gauss-Legendre quadrature

- Get n nodes and n weights from a computer program
  - ζ_i, $i = 1, \cdots, n$, ω_i, $i = 1, \cdots, n$
- Calculate the function values at the n nodes, f_i $i = 1, \cdots, n$
- Answer is equal to

$$\sum_{i=1}^{n} \omega_i f_i$$

- Anybody could do this
- How does the computer get the nodes and weights?

2n equations for nodes and weights

- To get right answer for $f(x) = 1$

$$\int_{-1}^{1} 1dx = \sum_{i=1}^{n} \omega_i 1$$

- To get right answer for $f(x) = x$

$$\int_{-1}^{1} xdx = \sum_{i=1}^{n} \omega_i \zeta_i$$

- To get right answer for $f(x) = x^2$

$$\int_{-1}^{1} x^2dx = \sum_{i=1}^{n} \omega_i \zeta_i^2$$

- etc

2n equations for nodes and weights

- To get right answer for $f(x) = x^j$ for $j = 0, \cdots, 2n-1$

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

- This is a system of $2n$ equations in $2n$ unknowns.

What has been accomplished so far?

- By construction we get right answer for

$$f(x) = 1,\ f(x) = x,\ \cdots,f(x) = x^{2n-1}$$

- But this is enough to get right answer for *any* polynomial of order $2n-1$

$$f(x) = \sum_{i=0}^{2n-1} a_i x^i$$

- Why?

Gauss-Hermite Quadrature

- Suppose we want to approximate

$$\int_{-\infty}^{\infty} f(x)e^{-x^2}dx \text{ with } \sum_{i=1}^{n} \omega_i f(\zeta_i)$$

- The function e^{-x^2} is the *weighting function*, it is not used in the approximation but is captured by the ω_i coefficients

Gauss-Hermite Quadrature

- We can use the same procedure to find the weights and the nodes, that is we solve them from the system:

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

- Note that $e^{-\zeta_i^2}$ is *not* on the right-hand side

Implementing Gauss-Hermite Quadrature

- Get n nodes, ζ_i, $i = 1, \cdots, n$, and n weights, ω_i, $i = 1, \cdots, n$, from a computer program
- Calculate the function values at the n nodes, f_i $i = 1, \cdots, n$
- Answer is equal to

$$\sum_{i=1}^{n} \omega_i f_i$$

Expectation of Normally distributed variable

- How to calculate

$$\mathsf{E}\left[h(y)\right] \text{ with } y \sim N(\mu, \sigma^2)$$

- That is, we have to calculate

$$\int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} h(y) \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) dy$$

- Unfortunately, this does not exactly fit the Hermite weighting function, but a change in variable will do the trick

Change of variables

- If $y = \phi(x)$ then

$$\int_a^b g(y)dy = \int_{\phi^{-1}(a)}^{\phi^{-1}(b)} g(\phi(x))\phi'(x)dx$$

- Note the Jacobian is added

Change of variables

The transformation we use here is

$$x = \frac{y-\mu}{\sigma\sqrt{2}} \text{ or } y = \sigma\sqrt{2}x + \mu$$

Change of variables

$$\mathsf{E}\left[h(y)\right] = \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} h(y) \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) dy$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} h(\sqrt{2}\sigma x + \mu) \exp\left(-x^2\right) \sigma\sqrt{2} dx$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} h(\sqrt{2}\sigma x + \mu) \exp\left(-x^2\right) dx$$

What to do in practice?

- Obtains n Gauss-Hermite quadrature weights and nodes using a numerical algorithm.
- Calculate the approximation using

$$\mathsf{E}\left[h(y)\right] \approx \sum_{i=1}^{n} \frac{1}{\sqrt{\pi}} \omega_i^{GH} h\left(\sqrt{2}\sigma\zeta_i^{GH} + \mu\right)$$

- Do not forget to divide by $\sqrt{\pi}$!
- Is this amazingly simple or what?

Extra material

- Derivation Simpson formula
- Monte Carlo integration

Lagrange interpolation

Let

$$L_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

$$f(x) \approx f_0 L_0(x) + \cdots + f_n L_n(x).$$

- What is the right-hand side?
- Do I have a perfect fit at the $n+1$ nodes?

Simpson: 2nd-order Newton-Cotes

- $x_0 = a$, $x_1 = (a+b)/2$, $x_2 = b$, or
- $x_1 = x_0 + h$, $x_2 = x_0 + 2h$

Using the Lagrange way of writing the 2$^{\text{nd}}$-order polynomial, we get

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b f_0 L_0(x) + f_1 L_1(x) + f_2 L_2(x) \\ &= f_0 \int_a^b L_0(x)dx + f_1 \int_a^b L_1(x)dx + f_2 \int_a^b L_2(x)dx\end{aligned}$$

Amazing algebra

$$\begin{aligned}
\int_a^b L_0(x)dx &= \frac{1}{3}h \\
\int_a^b L_1(x)dx &= \frac{4}{3}h \\
\int_a^b L_2(x)dx &= \frac{1}{3}h
\end{aligned}$$

- Why amazing?
    - formula only depends on h, not on values x_i and f_i
- Combining gives

$$\int_a^b f(x)dx \approx \int_a^b P_2(x)dx = \left(\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{1}{3}f_2\right)h.$$

True and pseudo Monte Carlo

To calculate an expectation

- Let x be a random variable with CDF $F(x)$
- Monte Carlo integration:

$$\int_a^b h(x)dF(x) \approx \frac{\sum_{t=1}^T h(x_t)}{T},$$

- Use random number generator to implement this in practice

True and pseudo Monte Carlo

What if integral is not an expectation

$$\int_a^b h(x)dx = (b-a)\int_a^b h(x)f_{\mathsf{ab}}(x)dx,$$

where f_{ab} is the density of a random variable with a uniform distribution over $[a,b]$, that is, $f_{\mathsf{ab}} = (b-a)^{-1}$.
Thus, one could approximate the integral with

$$\int_a^b h(x)dx \approx (b-a)\frac{\sum_{t=1}^{T} h(x_t)}{T},$$

where x_t is generated using a random number generator for a variable that is uniform on $[a,b]$.

Quasi Monte Carlo

- Monte Carlo integration has very slow convergence properties
- In higher dimensional problems, however, it does better than quadrature (it seems to avoid the curse of dimensionality)
- But why? Pseudo MC is simply a deterministic way to go through the state space
- Quasi MC takes that idea and improves upon it

Quasi Monte Carlo

- Idea: Fill the space in an *efficient* way
- *Equidistributed* series: A scalar sequence $\{x_t\}_{t=1}^{T}$ is equidistributed over $[a, b]$ iff

$$\lim_{T \longrightarrow \infty} \frac{b-a}{T} \sum_{t=1}^{T} f(x_t) = \int_a^b f(x)dx$$

  for all Rieman-integrable $f(x)$.
- Equidistributed takes the place of uniform

Quasi Monte Carlo

.

- Examples
  - $\xi, 2\xi, 3\xi, 4\xi, \cdots$ is equidistributed modulo 1 for any irrational number ξ.[1]
  - The sequence of prime numbers multiplied by an irrational number $(2\xi, 3\xi, 5\xi, 7\xi, \cdots)$

[1] $Frac(x)$ (or x Modulo 1) means that we subtract the largest integer that is less than x. For example, $frac(3.564) = 0.564$.

Multidimensional

For a d-dimensional problem, an equidistributed sequence $\{x_t\}_{t=1}^T \subset D \subset R^d$ satisfies

$$\lim_{T \longrightarrow \infty} \frac{\mu(D)}{T} \sum_{t=1}^{T} f(x_t) = \int_D f(x) dx,$$

where $\mu(D)$ is the Lebesque measure of D.

Multidimensional equidistributed vectors

Examples for the d-dimensional unit hypercube:

Weyl:

$$x_t = (t\sqrt{p_1}, t\sqrt{p_2}, \cdots, t\sqrt{p_d}) \text{ modulo } 1,$$

where p_i is the i^{th} positive prime number.

Neiderreiter:

$$x_t = (t2^{1/(d+1)}, t2^{2/(d+1)}, \cdots, t2^{d/(d+1)}) \text{ modulo } 1$$

References

- Den Haan, W.J., Numerical Integration, online lecture notes.
- Heer, B., and A. Maussner, 2009, Dynamic General Equilibrium Modeling.
- Judd, K. L., 1998, Numerical Methods in Economics.
- Miranda, M.J, and P.L. Fackler, 2002, Applied Computational Economics and Finance.

Projection

Wouter J. Den Haan
London School of Economics

© by Wouter J. Den Haan

Model

$$\begin{aligned}
c_t^{-\nu} &= \mathsf{E}_t\left[\beta c_{t+1}^{-\nu}\alpha z_{t+1}k_{t+1}^{\alpha-1}\right] \\
c_t + k_{t+1} &= z_t k_t^{\alpha} \\
\ln(z_{t+1}) &= \rho\ln(z_t) + \varepsilon_{t+1} \\
&\quad \varepsilon_{t+1} \sim N(0,\sigma^2) \\
&\quad k_1, z_1 \text{ given}
\end{aligned}$$

Projection Methods

True rational expectations solution:

$$\begin{aligned} c_t &= c(k_t, z_t) \\ k_{t+1} &= k(k_t, z_t) \end{aligned}$$

- Why a difficult problem to find these?

Define error terms

$$e\left(k_t, z_t\right) = -c_t^{-\nu} + \mathsf{E}_t\left[\beta c_{t+1}^{-\nu} \alpha z_{t+1} k_{t+1}^{\alpha-1}\right]$$

At the true solutions, $c(k_t, z_t)$ and $k\left(k_t, z_t\right)$:

$$e\left(k_t, z_t\right) = 0 \ \forall k_t, z_t$$

- Structural parameters $(\alpha, \beta, \rho, \sigma)$ have fixed numerical values (thus not included as arguments in policy function)

$$c_t = c(k_t, z_t) \approx P_n(k_t, z_t; \eta_n)$$

- $P_n(\cdot)$: from class of approximating functions
  - such as polynomials or splines
  - n is fixed $\Longrightarrow$ solve for η_n, a *finite-dimensional* object

Which equations to use?

- goal: solve for $P_n(k_t, z_t; \eta_n) \approx c(k_t, z_t)$,
  - i.e., N_n elements of η_n
  - $k(k_t, z_t)$ implicitly defined by budget constraint
- One first-order equation left, namely Euler equation
  - this is a different equation at each point in the state space
  - $\Longrightarrow$ plenty of equations

Which equations to use?

- At M grid points $\{k_i, z_i\}$ with $M \geq N_n$ we would like the following to equal zero:

$$e(k_i, z_i; \eta_n) = -P_n(k_i, z_i; \eta_n)^{-\nu} +$$

$$\mathsf{E}\begin{bmatrix} \alpha\beta\times \\ P_n(\{\boldsymbol{k}'\}, \{\mathbf{z}'\}; \eta_n)^{-\nu}\times \\ \{\boldsymbol{z}'\}\times \\ (\{\boldsymbol{k}'\})^{\alpha-1} \end{bmatrix}$$

Which equations to use?

- **Goal:** $\forall$ grid point get an expression with η_n as only unknown

$$e(k_i, z_i; \eta_n) = -P_n(k_i, z_i; \eta_n)^{-\nu} + E\left[\begin{array}{c} \alpha\beta\times \\ \\ P_n(\mathbf{k}', \mathbf{z}'; \eta_n)^{-\nu}\times \\ \\ \mathbf{z}'\times \\ \\ (\mathbf{k}')^{\alpha-1} \end{array}\right]$$

- Note that k_i and z_i are known

Which equations to use?

$$
\begin{gathered}
e(k_i, z_i; \eta_n) = -P_n(k_i, z_i; \eta_n)^{-\nu} + \\
\mathrm{E}\left[\begin{array}{c}
\alpha\beta \times \\
P_n(z_i k_i^\alpha - P_n(k_i, z_i; \eta_n), \exp\{\rho \ln(z_i) + \varepsilon'\}; \eta_n)^{-\nu} \times \\
\exp\{\rho \ln(z_i) + \varepsilon'\} \times \\
\left(z_i k_i^\alpha - P_n(k_i, z_i; \eta_n)\right)^{\alpha - 1}
\end{array}\right]
\end{gathered}
$$

How to deal with expectations operator?

Let $\{\omega_j, \zeta_j\}_{j=1}^J$ be the Hermite Gaussian quadrature nodes

$$e(k_i, z_i; \eta_n) = -P_n(k_i, z_i; \eta_n)^{-\nu} +$$

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

Define error terms

$$e(k_i, z_i; \eta_n) = -P_n(k_i, z_i; \eta_n)^{-\nu} +$$

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

How to find coefficients of approximation?

- True rational expect. solution gives zero error term $\forall(k_i, z_i)$
- Thus, choose η_n such that error terms are as small as possible.

- **Collacation** $(M = N_n)$: Use equation solver to get errors exactly equal to zero on grid
- **Galerkin** $(M > N_n)$: Use minimization routine (and possibly smart weighting of error terms)

Different types of approximating functions

- $P_n(k_i, z_i; \eta_n)$ could be polynomial or spline
- dimension η_n usually higher for splines
  - may make eq. solver/minimization less appropriate
  - use iteration scheme instead

How to find coefficients of approximation?

1. Equation solver or minimization routine
2. Iteration procedures
    1. fixed-point iteration
    2. time iteration

Iterating versus eq. solver/minimization

- Advantage:
    - less of a black box
    - can deal with many coefficients
        - e.g. when spline is used
    - some iteration schemes are guaranteed to converge
        - under some regularity conditions
- Disadvantage:
    - does not use information on how best to update

Iteration procedure: Construct Grid

- Construct a grid with nodes for k and z
- At the nodes construct the basis functions of $P_n(k, z; \eta_n)$.
- For example, if

$$P_n(k, z; \eta_n) = \eta_{0,n} + \eta_{k,n}k + \eta_{z,n}z + \eta_{kk}k^2 + \eta_{kz}kz + \eta_{zz}z^2$$

then construct the matrix (where subscripts denote grid numbers)

$$X = \begin{bmatrix} 1 & k_1 & z_1 & k_1^2 & k_1 z_1 & z_1^2 \\ 1 & k_2 & z_2 & k_2^2 & k_2 z_2 & z_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & k_M & z_M & k_M^2 & k_M z_M & z_M^2 \end{bmatrix}$$

and calculate $(X'X)^{-1} X'$

Iteration procedure: Construct Grid

- **Chebyshev nodes:** Using Chebyshev nodes is important. This ensures uniform convergence. With equidistant nodes it is possible that the oscillations between grid point explode as the order of the polynomial increases.
- **Chebyshev polynomials**: If you have (i) no problems finding initial conditions and (ii) only low-order appoximations so that calculating the inverse of $X'X$ can be done accurately, then you can use regular polynomials. Orthogonal Chebyshev polynomials can overcome these problems. They ensure that $X'X$ is diagonal (and trivial to invert). This does require scaling of the state variables so they are between -1 and 1.

Fixed-point Iteration

The value of η_n used in the q^{th} iteration is referred to as η_n^q. Follow the following iteration scheme until convergence

- At each grid point:
    - Calculate the RHS of the Euler equation using the latest value for η_n, i.e., η_n^{q-1}
    - Use RHS to calculate c_i, value for c at i^{th} grid point
- Use values for c_i to obtain an estimate for η_n, $\hat{\eta}_n^q$
    - Polynomial: run a regression to get $\hat{\eta}_n^q$
    - Spline: the values of c at the nodes are the new values of η_n
- Let $\eta_n^q = \lambda \hat{\eta}_n^q + (1-\lambda)\eta_n^{q-1}$

Fixed-point Iteration

- **Step 1: Calculate current consumption values implied by η_n^{j-1} at each grid point**
  - Use η_n^{q-1} to calculate $k' = z_i k_i^\alpha - P_n(k_i, z_i; \eta_n^{q-1})$
  - Use η_n^{q-1} to calculate $c' = P_n(k', z'; \eta_n^{q-1})$
  - Then, get c_i from

$$(c_i)^{-\nu} =$$

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

Fixed-point iteration

Step 2: Get new estimate for η_n by running a projection step

- Let $Y = [c_1, c_2, \cdots, c_M]'$
- If

$$P_n(k, z; \eta_n) = \eta_{0,n} + \eta_{k,n}k + \eta_{z,n}z + \eta_{kk}k^2 + \eta_{kz}kz + \eta_{zz}z^2$$

then

$$\hat{\eta}_n^q = \left(X'X\right)^{-1} X'Y$$

Fixed-point iteration

Step 2: Get new estimate for η_n by running a projection step

- If

  $$P_n(k,z;\eta_n) = \exp\left(\eta_{0,n} + \eta_{k,n}k + \eta_{z,n}z + \eta_{kk}k^2 + \eta_{kz}kz + \eta_{zz}z^2\right)$$

  then

  $$\widehat{\eta}_n^q = \left(X'X\right)^{-1} X' \ln(Y)$$

- no stochastic error term $\Longrightarrow$ ok to take ln of LHS & RHS

Fixed-point iteration

Step 3: Update η_n

$$\eta_n^q = \lambda \widehat{\eta}_n^q + (1-\lambda)\eta_n^{q-1} \text{ for } 0 < \lambda \leq 1$$

- Fixed-point iteration does not always converge
    - Choosing a lower value of λ:
        - convergence more likely
        - slows down algorithm if lower value not needed for convergence
- Alternative is **time iteration**

Time Iteration

- At each grid point use η_n^{q-1} only for *next period's* choices
- Again solve for c_i at each grid point
  - this is now a bit trickier (non-linear problem)
- Get n_n^q as with fixed-point iteration
  - guaranteed to converge without dampening (under regularity conditions)

Time Iteration - solving for c

Solve c_i from following non-linear equation

$$(\boldsymbol{c_i})^{-\nu} = \sum_{j=1}^{J} \left[\begin{array}{c} \alpha\beta\times \\ P_n(z_i k_i^{\alpha} - \boldsymbol{c_i}, \exp\{\rho \ln(z_i) + \sqrt{2}\sigma\zeta_j\}; \eta_n^{q-1})^{-\nu} \times \\ \exp\{\rho \ln(z_i) + \sqrt{2}\sigma\zeta_j\} \times \\ \left(z_i k_i^{\alpha} - \boldsymbol{c_i})\right)^{\alpha-1} \\ \omega_j / \sqrt{\pi} \end{array} \right]$$

Time Iteration

- Natural interpretation for η_n^{q-1} and η_n^q, namely
  - η_n^{q-1} is tomorrow's policy function and
  - η_n^q is today's policy function
- Time iteration is reliable and convergent
  - (the proof is related to the convergence of value function iteration, which uses the same idea)

Fixed-point versus time iteration

- Fixed-point iteration uses η_n^{q-1} for *all* terms on the RHS, i.e., both next period's consumption choice and today's capital choice
- Time iteration uses η_n^{q-1} only to evaluate next period's consumption
- The structure of time iteration mimics the choice of value function iteration:
    - next period's behavior described by previous solution for value function
    - Bellman equation used to solve for choice of c and k *simultaneously*

Endogenous grid points

- Simple idea: construct grid for k' instead of a grid for k
- Instead of solving for the choice k' given k, we now solve for the value of k that would have led to the choice k'
- In both cases you end up at each grid point with a set of values for k and a set of corresponding values for k'.
- Terminology is a bit confusing: the grid itself is exogenous and fixed but it is for an endogenous variable
- You can use endogenous grid points both with fixed-point and with time iteration
- The added value with time iteration lies in getting rid of the non-linear problem of solving for today's choices

Endogenous grid points and time iteration

- Time iteration $\Longrightarrow$
  - use η_n^{q-1} for tomorrow's choices and
  - use η_n^q only for today's choices (which show up on both sides of the policy function
- Then, get c_i from

Endogenous grid points and time iteration

$$(c_i)^{-\nu} = \sum_{j=1}^{J} \begin{bmatrix} \alpha\beta \times \\ P_n(k_i', \exp\{\rho \ln(z_i) + \sqrt{2}\sigma\zeta_j\}; \eta_n^{q-1})^{-\nu} \times \\ \exp\{\rho \ln(z_i) + \sqrt{2}\sigma\zeta_j\} \times \\ (k_i')^{\alpha-1} \\ \omega_j / \sqrt{\pi} \end{bmatrix}$$

and k_i from

$$k_i' + c_i = z_i k^\alpha$$

Perturbation versus projection

- Nondifferentiabilities
  - impossible for perturbation
- Large number of state variables
  - difficult for projection
- Constructing the grid can be difficult
  - apriori hard to know what sensible points are
  - some calculations may not be well defined everywhere

Perturbation versus projection

- Global versus local
  - Projection designed to be global method
  - Perturbation designed to be local method
    - but could give accurate global approximation
    - question is whether (lower-order) derivatives at perturbation point capture global behavior

When can't you use projection methods?

- Not all solutions to optimization problems can be characterized by first-order conditions
    - e.g. when objective function is not concave or budget set not convex
    - then you have no choice but to use Value Function Iteration

When can't you use projection methods?

- Constructing a grid where all calculations are well defined may be tough
    - e.g., not get negative consumption/unemployment
    - this can be tough even at the true solution
    - calculations should be possible also on path towards solution
- Solutions
    - Simply exclude problematic grid points (works for Galerkin)
    - Endogenize grid using simulations (Parameterized expectations)
        - but simulated points cluster so you are likely to get worse convergence properties

References

- Heer, B., and A. Maussner, 2009, Dynamic General Equilibrium Modeling.
- Judd, K. L., 1998, Numerical Methods in Economics.
- Rendahl, P., 2006, Inequality constraints in recursive economies.
  - shows that time-iteration converges even in the presence of inequality constraints