Lecture I
Appendix: Numerical Recipes

Adam Hal Spencer

The University of Nottingham

Applied Computational Economics 2020

Roadmap

1. Adda & Cooper (2003) Discretisation
2. Interpolation
3. Root-Finding Methods

Adda & Cooper (2003) AR(1) Approximation

- We'll follows the Adda & Cooper (2003) approach.
- The procedure is:
  (1) Discretise process into $N \in \mathbb{N}$ intervals,
  (2) Get the conditional mean of each interval (discretised y_t values),
  (3) Find the conditional transition probability of moving from one interval to the next, (transition matrix).

Adda & Cooper (2003): Step (1)

- Denote the limits of each of the N intervals of y_t as $y^1, y^2, y^3, ..., y^{N+1}$.
- See that $y^1 = -\infty$ and $y^{N+1} = \infty$.

Adda & Cooper (2003): Step (1)

- Cut-off points are then defined as (F denotes the normal CDF)

$$F\left(\frac{y^{i+1}-\mu}{\sigma_y}\right)-F\left(\frac{y^{i}-\mu}{\sigma_y}\right)=\frac{1}{N}$$

for $i = 1, 2, ..., N$ and where

$$\sigma_y^2=\frac{\sigma^2}{1-\rho^2}$$

is the unconditional variance of y_t. This all follows from the normality of ϵ_t.

Adda & Cooper (2003): Step (1)

- Working recursively, we can then write

$$y^i = \sigma_y F^{-1}\left(\frac{i-1}{N}\right) + \mu$$

Adda & Cooper (2003): Step (2)

- Denote the conditional mean of interval i as z^i. See that

$$\begin{aligned} z^i &= \mathbb{E}\{y_t | y_t \in [y^i, y^{i+1}]\} \\ &= \frac{1}{N} \frac{1}{\sqrt{2\pi\sigma_y^2}} \int_{y^i}^{y^{i+1}} y \exp\left(\frac{-[y-\mu]^2}{2\sigma_y^2} \right) dy \\ &= N\sigma_y \left[f\left(\frac{y^i - \mu}{\sigma_y} \right) - f\left(\frac{y^{i+1} - \mu}{\sigma_y} \right) \right] + \mu \end{aligned}$$

  where f is the normal distribution PDF.

- The last line comes from changing variables and a lot of painful manipulations.

Adda & Cooper (2003): Step (3)

- Denote the transition probability then as π_{ij} where

$$
\begin{aligned}
\pi_{ij} &= \Pr(y_t \in [y^j, y^{j+1}] | y_{t-1} \in [y^i, y^{i+1}]) \\
&= \frac{N}{\sqrt{2\pi\sigma_y^2}} \int_{y^i}^{y^{i+1}} \exp\left(\frac{-[y_{t-1} - \mu]^2}{2\sigma_y^2} \right) \times
\end{aligned}
$$

$$
\left\{ F\left(\frac{y^{j+1} - \mu(1-\rho)\rho y_{t-1}}{\sigma} \right) - F\left(\frac{y^j - \mu(1-\rho) - \rho y_{t-1}}{\sigma} \right) \right\} dy_{t-1}
$$

where you can use numerical integration to evaluate π_{ij}, (built-in Matlab function).

Roadmap

Cubic Splines

- We've talked about linear interpolation in class.
- We can take things up another level: approximate using cubic functions.
- Construct a function $s(x)$ such that $s(x_i) = y_i$ at each of the cut-offs and on each connecting interval

$$s_{[x_i, x_{i+1}]}(x) = a_i + b_i x + c_i x^2 + d_i x^3$$

meaning that we have $N - 1$ intervals and N datapoints.

Cubic Splines

- How do we identify all these coefficients in the approximation?
- We have to deal with $\{a_i, b_i, c_i, d_i\}_{i=1}^{N-1}$.
- That is: $4(N-1)$ unknowns.

Cubic Splines

- We have the cut-offs: provides us with $2(N-1)$ equations:

$$\begin{aligned} y_1 &= s_{[x_1,x_2]}(x_1) \;\text{ for }\; i = 1 \\ s_{[x_{i-1},x_i]}(x_i) = y_i &= s_{[x_i,x_{i+1}]}(x_i) \;\text{ for }\; i = 2, ..., (N-1) \\ y_N &= s_{[x_{N-1},x_N]}(x_N) \;\text{ for }\; i = N \end{aligned}$$

where notice that the middle line says that we want consistency on either side of the non-endpoint cut-offs.

Cubic Splines

- So we still have $2(N-1)$ equations to find.
- What else can we do?
- Impose continuity of the first and second derivatives.

Cubic Splines

- For the first derivative

$$s'_{[x_{i-1},x_i]}(x_i) = s'_{[x_i,x_{i+1}]}(x_i)$$
$$\Rightarrow b_{i-1} + 2c_{i-1}x_i + 3d_{i-1}x_i^2 = b_i + 2c_ix_i + 3d_ix_i^2$$

for $i = 2, ..., (N-1)$. For the second derivative

$$s''_{[x_{i-1},x_i]}(x_i) = s''_{[x_i,x_{i+1}]}(x_i)$$
$$\Rightarrow 2c_{i-1} + 6d_{i-1}x_i = 2c_i + 6d_ix_i$$

for $i = 2, ..., (N-1)$.

Cubic Splines

- Almost there. But we're still missing two equations.
- If we knew what the true function was — if we knew $f'(x_1)$ and $f'(x_N)$, then we could impose

$$s'_{[x_1,x_2]}(x_1) = f'(x_1)$$
$$s'_{[x_{N-1},x_N]}(x_N) = f'(x_N)$$

but why are we approximating this function if we knew that? We don't generally.

Cubic Splines

- You could assume that

$$
\begin{aligned}
s'_{[x_1,x_2]}(x_1) &= 0 \\
s'_{[x_{N-1},x_N]}(x_N) &= 0
\end{aligned}
$$

a bit arbitrary though, no?

Cubic Splines

- Could also use the first and final secants to approximate the first and final derivatives

$$s'_{[x_1,x_2]}(x_1) = \frac{s_{[x_1,x_2]}(x_2) - s_{[x_1,x_2]}(x_1)}{x_2 - x_1}$$
$$s'_{[x_{N-1},x_N]}(x_N) = \frac{s_{[x_{N-1},x_N]}(x_N) - s_{[x_{N-1},x_N]}(x_{N-1})}{x_N - x_{N-1}}$$

- Whatever you choose, the additional two points complete the system.
- It's a bunch of equations you can solve on the computer.

The Need for Orthogonality

- The space of continuous functions is spanned by monomials of the form x^n for $n = 0, 1, 2, \ldots$.
- But these things all start to look alike after a while: collinear.
- Adding more an more of these things isn't going to help us achieve a better fit.
- Enter: orthogonal polynomials.

Orthogonal Polynomials

- Orthogonality generalises the idea of perpendicular objects.
- Two polynomials are orthogonal on some vector space if their inner product is zero.
- Inner products just generalise the idea of measuring angles between vectors.

Orthogonal Polynomials

- If two polynomials are orthogonal, it means that their shapes do not really resemble each other.
- Exactly what we're after with these approximation techniques: overcomes the collinearity issue with monomials.
- You can construct orthogonal polynomials using the Gram-Schmidt procedure.
- You can just Chebyschev polynomials: a special case.

Chebyschev Polynomials

- These are polynomials defined on the interval $T_n : [-1, +1] \rightarrow [-1, +1]$ that can be expressed recursively as

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \text{ for } n \geqslant 2$$

where

$$T_0(x) = 1$$
$$T_1(x) = x.$$

- See then that

$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 2x(2x^2 - 1) - x$$

Chebyschev Polynomials: Implementation

- Judd's algorithm: choose m nodes and use them to construct a degree $n < m$ polynomial approximation of $f(x)$ on the interval $[a, b]$.

(1) Compute the $m \geqslant n + 1$ nodes on [-1,1].

$$z_k = -\cos\left(\frac{2k-1}{2m}\pi\right) \text{ for } k = 1, ..., m$$

(2) Adjust the nodes to your interval $[a, b]$

$$x_k = (z_k + 1)\left(\frac{b-a}{2}\right) + a \text{ for } k = 1, ..., m$$

(3) Evaluate the function at the approximation nodes

$$y_k = f(x_k) \text{ for } k = 1, ..., m$$

Chebyschev Polynomials: Implementation

(4) Compute the Chebyschev coefficients c_i using

$$c_i = \frac{\sum_{k=1}^{m} y_k T_i(z_k)}{\sum_{k=1}^{m} T_i(z_k)^2} \text{ for } i = 0, 1, ..., n$$

to get your approximation given as

$$\hat{f}(x) = \sum_{i=0}^{n} c_i T_i \left(2\frac{x-a}{b-a} - 1 \right)$$

Interpolation v.s. Grid Search Example

- Consider a simple Bellman equation of the form

$$v(x) = \max_{x' \in X} \; u(x - x') + \beta v(x')$$

where u is just some function.

- What does it look like to implement grid search here relative to interpolation methods?

Example: Grid Search

- Start by discretising the space X into $\{x_i\}_{i=1}^{N}$ where $x_1 < x_2 < ... < x_N$.
- We have some $N \times 1$ vector from the j^{th} iteration.
- For each x_i, we aim to solve for v^{j+1} using

$$v^{j+1}(x_i) = \max_{x' \in X} \; u(x_i - x') + \beta v^j(x')$$

Example: Grid Search

- The grid search algorithm is then of the form:
  - While $\sup_x |v^j(x) - v^{j-1}(x)| > \epsilon$,
  - Evaluate the vector of the form

$$V_i^{j+1} = \begin{bmatrix} u(x_i - x_1) + \beta v^j(x_1) \\ u(x_i - x_2) + \beta v^j(x_2) \\ ... \\ u(x_i - x_N) + \beta v^j(x_N) \end{bmatrix}$$

    for each $i = 1, ..., N$.

  - That is: for each gridpoint x_i in the state space, compute this column vector of numbers.
  - The optimal choice of x' for a given x_i is the one that gives you the largest number in vector V_i^{j+1}.

Example: Linear Interpolation

- Utilise the same gridpoint setup, but now we have intervals

Interval	**Value** $v^j(x)$
$x \in [x_1, x_2]$	$a^j_{1,2} + b^j_{1,2}x$
$x \in [x_2, x_3]$	$a^j_{2,3} + b^j_{2,3}x$
...	...
$x \in [x_{N-1}, x_N]$	$a^j_{N-1,N} + b^j_{N-1,N}x$

- Then for each x_i, we aim to solve for v^{+1} using

$$v^{j+1}(x) = \max_{x' \in [x_1, x_N]} \quad u(x_i - x') + \beta v^j(x')$$

where see we are now looking at the entire interval $[x_1, x_N]$ rather than just in the set $\{x_i\}_{i=1}^N$.

Example: Linear Interpolation

- For the piecewise linear interpolation, the algorithm looks like the following
  - While $\sup_x |v^j(x) - v^{j-1}(x)| > \epsilon$,
  - Use a maximisation routine to maximise V_i^{j+1} where

$$V_i^{j+1} = \begin{bmatrix} u(x_i - x') + \beta[a^j_{1,2} + b^j_{1,2}x'] \text{ for } x' \in [x_1, x_2] \\ u(x_i - x') + \beta[a^j_{2,3} + b^j_{2,3}x'] \text{ for } x' \in [x_2, x_3] \\ \dots \\ u(x_i - x') + \beta[a^j_{N-1,N} + b^j_{N-1,N}x'] \text{ for } x' \in [x_{N-1}, x_N] \end{bmatrix}$$

    call the minimised value $v^{j+1}(x_i)$.

  - With the new $v^{j+1}(x_i)$, recompute $\{a^{j+1}_{i,i+1}, b^{j+1}_{i,i+1}\}_{i=1}^{N-1}$.

Roadmap

Non-Linear Equations

- So many economic models look like the following

$$f(x, y) = 0$$

where f is a function.

- That is: $\exists y = g(x)$ such that

$$f(x, g(x)) = 0 \ \forall x \in X$$

- E.g. an Euler equation.

- Searching for some policy function $g(x)$ of the current state x.

Bisection

- Remember the **Intermediate Value Theorem**?
  - If we have a continuous function over some interval $[a, b]$ and takes the values $f(a)$ and $f(b)$ at these points, then $f(x)$ takes any value between $f(a)$ and $f(b)$ for $x \in [a, b]$.
- **Bolzano's Theorem**:
  - If $f(a)$ and $f(b)$ have opposite signs with f continuous on $[a, b]$, then there must be a root such that $f(x) = 0$ for some $x \in [a, b]$.

Bisection

- $f(a)$ and $f(b)$ with opposite signs...what does this mean in an economic context?
- Excess demand, anyone?
- If the price is too high, then excess demand is negative.
- If the price is too low, then excess demand is positive.

Bisection

- Say we think $f(x)$ is continuous on $x \in [a, b]$.
- Method of bisection follows the procedure
  - Guess x' such that $x' = \frac{a+b}{2}$.
  - If $f(x')$ and $f(b)$ have opposite signs, then replace a with x'.
  - If $f(x')$ and $f(a)$ have opposite sings, then set b as x'.
  - Repeat until $|a - b| < \epsilon$.

Bisection

- E.g. if $ED(p)$ is excess demand as a function of price $p \in [a, b]$.
- Set $x' = \frac{a+b}{2}$ then
  - Increase the price if $ED(p) > 0$: i.e. set $a = x'$.
  - Decrease the price if $ED(p) < 0$: i.e. set $b = x'$.

Bisection

- Obviously many other procedures you can use.