

## Value function iteration

22 April 2020 cont

Judd "Numerical Methods" and Judd 1996 reading this!

The main concern is to find (or approximate) a function  $f$  (e.g. a policy function like the consumption function)  
perturbation  $\rightarrow$  local approx

global methods  $\rightarrow$  " $L^P$  approximation":

find a "nice" function  $g$  which is "close to"  $f$  by a  $L^P$  norm.

### Interpolation

any procedure which finds a "nice" function

$g$  that fits a finite set of  $n$  points

(we only have info about  $f$  at  $n$  points)

Regression: related to interpolation b/c it uses  $n$  points to fit a smooth function to data

using, in the case of least squares, an  $L^2$  norm.

$\rightarrow$  forms the basis of "projection methods" aka "weighted residual methods"

local methods: perturbation

→ find an approx of  $f$  around a particular local point

global methods: projection

→ find a global approx of  $f$

### The mathematics of $L^p$ approximations (udd 1996, Sect 7)

Recall: we wanna find representations of  $f \in \mathcal{F}$   
which are the continuous ft  $g$ .

→ We're in the space of continuous functions.

→ the space of continuous functions is spanned  
by the polynomials,  $x^n$

→ the polynomials  $x^n$  will be the basis of  
the space of continuous functions

→ bases have the property of orthogonality

⇒ we will therefore construct orthogonal polynomials

7.1) DEF. Weighting function  $w(x)$  on  $[a, b]$

a positive function w/ finite integral on  $[a, b]$

DEF. Inner product of integrable functions over  $[a, b]$

$$\langle f, g \rangle := \int_a^b f(x) g(x) w(x) dx$$

"The weighted sum over  $[a, b]$  of the product  $f(x)g(x)$ "

DEF. Orthogonal polynomials

A family of polynomials  $\{q_n(x)\}$  is mutually orthogonal w.r.t. weighting function  $w(x)$  iff

$$\langle q_n, q_m \rangle = 0 \quad n \neq m.$$

The most used families of orthogonal polynomials in economics are

Legendre

Chebyshev

Laguerre

Hermite polynomials.

(!)

## Legendre polynomials

weight function:  $w(x) = 1$  on  $[-1, 1]$

$n^{\text{th}}$  polynomial:  $P_n(x) := \frac{(-1)^n}{2^n n!} \cdot \frac{d^n}{dx^n} [(1-x^2)^n]$

## Chebyshev polynomials

weight function:  $w(x) = (1-x^2)^{-1/2}$  on  $[-1, 1]$

$n^{\text{th}}$  polynomial:  $T_n(x) := \cos(n \cos^{-1} x)$

$\Rightarrow$  these two are useful for problems in compact sets.

## Laguerre polynomials

weight function:  $w(x) = e^{-x}$  on  $[0, \infty)$

$n^{\text{th}}$  polynomial:  $L_n(x) := \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$

useful: when one needs to approx the time paths  
of variables in a deterministic analysis

## Hermite polynomials

weight function:  $w(x) = e^{-x^2}$  on  $(-\infty, \infty)$

$n^{\text{th}}$  polynomial:  $H_n(x) := (-1)^n e^{x^2} \frac{d^n}{dx^n} (e^{-x^2})$

useful: to approx functions of normal random variables

## 7.2) Least-squares orthogonal polynomial approximation

The LS polynomial approx of  $f$  given weighting fn  $w(x)$  is the  $n$ -degree polynomial which solves

$$p(x) = \operatorname{argmin} \int_a^b (f(x) - p(x))^2 w(x) dx$$

Note: Like in GMM, the weighting fn allows us to specify where the approx should be better (by setting a higher weight there).

Now use: orthog polynomials. If  $\{\varphi_n\}_{n=1}^\infty$  is an orthogonal sequence wrt  $w(x)$ , then the least-squares solution is

$$p(x) = \sum_{i=0}^n \frac{\langle f, \varphi_i \rangle}{\langle \varphi_i, \varphi_i \rangle} \varphi_i(x)$$

"Cov(f, x)"  
"Var(x)"

Note: Linear regression is a special case of this, w/  
 $\varphi_i(x)$   $i=0, \dots, n$  interpreted as the  $n+1$  regressor,  $f$  as the regressed.

## Chebyshev approximation theorem (A type of LS approx.)

Assume  $f \in C^k [-1, 1]$  let

$$C_n(x) := \frac{1}{2} c_0 + \sum_{j=1}^n c_j T_j(x)$$

where  $c_j := \frac{2}{\pi} \int_{-1}^1 \frac{f(x) T_j(x)}{\sqrt{1-x^2}} dx$

Then  $\exists a, b$  such that  $\forall a \geq 2$

$$\|f - C_n\|_\infty \leq \frac{b \ln n}{n^a}$$

That is,  $C_n \rightarrow f$  uniformly as  $n \rightarrow \infty$

Also,  $\exists c : |c_j| \leq \frac{c}{j^6} \quad j \geq 1$

This gives us a way to answer if an  $n$ -order Cheby approx is nearly as good as a full approx.  
If the coefficients of the last terms are dropping at the rate  $j^{-6}$ , we can stop; if they are not, we need to add  $n+1$  order,  $n+2$  etc ...

### 4.3) Interpolation

Take a set of  $n$  pointwise restrictions and find a function  $f: R^n \rightarrow R^m$  satisfying those restrictions.

Lagrange interpolation where  $y_i = f(x_i)$

Take  $n$  points  $(x_i, y_i)$   $i=1, \dots, n$ .

Find degree  $n-1$  polynomial,  $p(x)$ , such that

$$y_i = p(x_i), \quad i=1, \dots, n.$$

The polynomial that interpolates the data  $(f(\cdot))$  is

$$p(x) = \sum_{i=1}^n y_i l_i(x), \quad \text{where}$$

$$l_i(x) := \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

Note:  $l_i(x) = 1$  for  $x = x_i$

$l_i(x) = 0$  for  $x = x_j$  for  $i \neq j$

## Hermite interpolation

$$p(x) = \sum_{i=1}^n y_i h_i(x) + \sum_{i=1}^n y'_i \tilde{h}_i(x)$$

is the polynomial that interpolates the data  
where

$$\tilde{h}_i(x) := (x - x_i) l_i(x)^2$$

$$h_i(x) := (1 - 2l'_i(x)(x - x_i)) l_i(x)^2$$

Note  $h_i(x) = 1$  when  $x = x_i$

$h_i(x) = 0$  when  $x = x_j$   $j \neq i$

Q: - it seems to me as if  $\tilde{h}_i(x) = 0 \forall x$ .

Good for the case when we have data:

$$p(x_i) = y_i \quad i = 1, \dots, n$$

$$p'(x_i) = y'_i$$

## 7.4. Approx by interpolation

Powerful! - but can behave perversely, in particular the polynomial  $p_n$  may not converge to  $f$  if the nodes  $x_i$  are uniformly placed. Therefore care is called for:

### 7.4.1. Interpolation error

DEF. Interpolation error

$$\Psi(x; x_0, \dots, x_n) := \prod_{k=1}^n (x - x_k)$$

(at least for the Lagrange polynomial interpolating  $f$  at points  $x_i$ , denoted  $p_n(x)$ )

THM 9 Bounds on errors of Lagrange Interpolant

Ass  $a = x_0 < x_1 < \dots < x_n = b$  Then

$$\sup_{x \in [a, b]} |f(x) - p_n(x)| \leq \underbrace{\|f^{(n+1)}\|_\infty}_{\text{"the upper bound on interp errors"}} \underbrace{\frac{1}{(n+1)!}}_{\text{"is smaller than stuff we can't influence"}} \sup_{x \in [a, b]} \Psi(x; x_0, \dots, x_n) \underbrace{\text{"and stuff we can lower by choosing interp points wisely"}}$$

## Supremum (sup)

The smallest of the elements in set T

that is greater than all elements in set S.

## Infimum (inf)

The greatest of the elements in set T

that is smaller than all elements in set S.

A nice example is  $S = \mathbb{R}^-$  (excluding 0).

I don't think this has an inf, but it does have

a sup:  $\sup(\mathbb{R}^-) = 0$  since 0 is the smallest number that's bigger than all negative real numbers.

→ it turns out that the best way to choose interpolation nodes (i.e. the sol to the problem

$$\min_{x_1, \dots, x_n} \max_x \prod_{k=1}^n (x - x_k) = \Psi(x; x_1, \dots, x_n)$$

is the Chebyshev interpolation (PTD)

## Chebyshev interpolation

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right) \quad k = 1, \dots, n$$

general Cheby (Judd, 1858 p.  $\frac{703}{744}$ )

(which are the zeros of the Tscheby polyns  $T_n(x)$ )

Interpretation: The interpolation points  $x_i$  (or nodes) that minimize the interpolation error are the zeros of the Chebyshev polynomial (adapted to the interval).

Thm 10 Chebyshev interpolation theorem (p. 554) (Mac 46)

"Chebyshev interpolation keeps the max error acceptably small"

I think what this is saying is that suppose you approx f using the Lagrange polynomial  $P_n(x)$  w/ gridpoints  $x_i$ ,  $i=1, \dots, n$  which are zeros of the Chebyshev polynomial  $T_n(x)$ , then you can be sure that the interpolation is good enough.

"Valuable whenever the approximated function is smooth."

## 7.5 Approximation through regression

Goal is still to approx  $f$ . When using regression instead of interpolation:

1. Evaluate  $f(x)$  at  $m$  points
  2. Use them to choose a parametric approx w/  
 $n \ll m$  parameters which minimizes some  
loss function
- "semi-parametric methods" w/ "random"  
choices for the nodes  $x_i$ .

## 7.6 Piecewise polynomial interpolation

An alternative that computes a function which is  
only piecewise smooth. 2 common schemes:  
Hermite polynomials and splines.

### 7.6.1 Step function approximations ...

... on  $[a,b]$  are generated by a basis of step functions,

$y_i$ ,  $i = 1, \dots, n$ , where  $h = \frac{a-b}{n}$  and

the basis is

$$\varphi_i(x) = \begin{cases} 0 & a \leq x \leq a + (i-1)h \\ 1 & a + (i-1)h \leq x < a + ih \\ 0 & a + ih \leq x \leq b \end{cases}$$

If the data are  $(x_i, y_i)$  and  $\varphi_i(x_i) = 1$ , then  
the step function  $\sum_{i=1}^n y_i \varphi_i(x)$  interpolates the  
data.

### 7.6.2. Piecewise linear approximation (!)

1) Take a sequence of data  $(x_i, y_i)$

2) Create a piecewise linear function which  
interpolates the data

I'm not sure, but I think that like the step function  
approach,  $\sum_{i=1}^n y_i \varphi_i(x)$  interpolates the data. The  
difference is that now the basis  $\varphi_i(x)$  is a tent function:

$$y_i(x) = \begin{cases} 0 & a \leq x \leq a + (i-1)h \\ \frac{x - (a + (i-1)h)}{h} & a + (i-1)h \leq x \leq a + ih \\ 1 - \frac{x - (a + ih)}{h} & a + ih \leq x \leq a + (i+1)h \\ 0 & a + (i+1)h \leq x \leq b \end{cases}$$

Note: In Judd 1998, Numerical methods, Judd also considers Picewise linear interpolation

for the data,  $(x_i, y_i)$  as

$$\hat{f}(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i} (y_{i+1} - y_i) \quad x \in [x_i, x_{i+1}] \quad (6.8.1)$$

which he calls the "kindergarten procedure of 'connecting the dots'".

The basis of this is a B-spline (see later)

I do think this is the same as the picewise linear approx.

### 7.6.3. Hermite interpolation polynomials

- Supp.  $x_1, \dots, x_n$  give us both level ( $y_i$ ) info as well as slope ( $f'(x_i) = y'_i$ ) info
- For each  $[x_i, x_{i+1}]$  interval, construct the Hermite interpolation polynomial
- The result is a collection of (mostly cubic) Hermite polynomials : This is a piecewise polynomial function which has kinks at the nodes

$\downarrow$   
splines all over this

### 7.6.4. Splines (!)

Another piecewise smooth scheme, which is not only smooth between, but also at the nodes.

#### DEF. Spline

A smooth function  $s(x)$  on  $[a, b]$  is a spline of order  $k$  if  $s$  is  $C^{k-2}$  on  $[a, b]$  and there is a grid of points

(called nodes)  $a = x_0 < x_1 < \dots < x_n = b$   
such that  $s$  is a polynomial of degree at most  
 $k-1$  on each subinterval  $[x_i, x_{i+1}]$   $i=0, \dots, n-1$ .

Note: order 2 splines are just the common  
piecewise linear interpolant in (6.8.1).

### The cubic spline (of order 4)

23 April 2020

1. Supp we have data  $(x_i, y_i)$   $i=0, \dots, n$
2.  $x_i$  will be the nodes
3. Construct a spline such that  $s(x_i) = y_i$   $i=0, \dots, n$   
and on each interval  $[x_i, x_{i+1}]$ ,  $s(x)$  will be a  
cubic  $a_i + b_i x + c_i x^2 + d_i x^3$ .
4. We have  $n$  intervals, so  $4n$  coefficients. Somehow  
I should be able to tell that we only have  
 $4n-2$  conditions on those  $4n$  coefficients.
5. Need to impose two additional constraints to pin down  
the spline:

natural splines: the 2 additional constraints are

$$s'(x_0) = 0 = s'(x_n)$$

Natural splines also have the property that

it minimizes the total curvature,  $\int_{x_0}^{x_n} s''(x)^2 dx$   
of all cubic splines that interpolate the data.

Hermite spline: if we know the derivative (the true slope) of the approximand at the endpoints, we can impose those as the 2 additional constraints.

$$\begin{aligned} s'(x_0) &= y'_0 \quad \text{and} \quad s'(x_n) = y'_n \\ &= f'(x_0) \quad \quad \quad = f'(x_n) \end{aligned}$$

Judd, Numerical Methods, p. 230-231 Mac gives  
an overview for the equation system you need to  
solve to obtain the coefficients  $(a_i, b_i, c_i, d_i)$

$$i = 0, \dots, n-1$$

Splines are great!

1.) b/c they are very cheap to evaluate

2.) good fits even for ill-behaved functions ( $\text{not } C^\infty$ )

→ vs. orthogonal polynomials, which work well if the approximand is well-behaved.

Judd, Numerical methods p. 231 Mac has more on the computation of splines. And on: B-Splines

### B-Splines

the space of splines w/ nodes on a prescribed grid form a finite vector space. The B-splines form a basis for splines.

Supp we have a grid of nodes  $x_k < \dots < x_i < x_0 < \dots < x_{n+k}$ . Order 1 splines implement a step function interpolation and are spanned by  $B^0$ -splines:

$$B_i^0(x) = \begin{cases} 0 & x < x_i \\ 1 & x_i \leq x \leq x_{i+1} \\ 0 & x_{i+1} < x \end{cases}$$

A step function.

Linear splines (order 2) implement piecewise linear interpolation and are spanned by  $B^1$ -splines

$$B_i^1(x) = \begin{cases} 0 & x \leq x_i \\ \frac{x - x_i}{x_{i+1} - x_i} & x_i \leq x \leq x_{i+1} \\ \frac{x_{i+2} - x}{x_{i+2} - x_{i+1}} & x_{i+1} \leq x \leq x_{i+2} \\ 0 & x_{i+2} \leq x \end{cases}$$

A tent function.

Higher-order B-splines are defined by the recursive relation

$$B_i^k(x) = \frac{x - x_i}{x_{i+k} - x_i} B_i^{k-1}(x) + \frac{x_{i+k+1} - x}{x_{i+k+1} - x_{i+1}} B_{i+1}^{k-1}(x)$$

For theory of B-splines, see de Boor (1978).

Work more examples in Judd, Numerical, p. 233 Mac, "Examples"?

## Shape-preserving interpolation (!)

The point is: you will shape-preserve automatically as the # nodes  $\rightarrow \infty$ . But you never have  $\infty$  points (and also, recall that more points may actually screw up convergence if they're not placed smartly). So we want methods that preserve the shape of the approximand even for a small # of points, and in particular, between nodes.

→ orthogonal polynomials usually do not preserve shape.

Picard - polynomial approx does.

→ so our discussion of piece-wise linear / polynomial interp / approx is shape-preserving.  
But it's not smooth.

⇒ shape-preserving (quadratic) spline of Schumaker (1983)

Susanto meeting

23 April 2020

↳ liquidity problem of TIPS market

problem: TIPS market is relatively thin

→ large bid-ask spreads

or prices move a lot in particular directions

→ filters out big premium  $\Rightarrow$  paper

Peter Fortune at Boston Fed 15 years ago

AM slide 13  $E_t \pi_{t+1}$  in (13), in RE

$K_{t+1}$  is summed in  $\pi_{t+1}$

↳  $x_{t+1}$  is a qualitatively new term.

↳ Need to explain that!

Solve nonlinear diff eq numerically

"Automation" is the right way to put it; "bw-looking" isn't

Woodford's timelss perspective is stationary  
whereas the learning model isn't

don't call a commitment device a "comm. device"  
call it "punishment"

↳ Commitment: what does it mean?

The ability to carry out what you  
announced even though it's not  
ex post optimal. (not subgame perfect)

↳ timeless perspective is like ass.

that the planner has this power

→ check w/ Ryan!

Check that TRC are satisfied under  $\hat{E}$

→ for the HMs. Useful to know if it is!

- Show on IRFs that  $\tau_1$  etc are less variable under Mean vs RE
- Motivating episodes  $\rightarrow$  make 'em clear!  
Missing Difl & Infl.

shape-preserving quadratic splines

29 April 2020

Schumaker (1983)

The idea is to find a function  $s \in C^1[t_1, t_2]$  that has the same shape as the data (concave, convex, monotonic, nonnegative etc). The idea is to add an  $i$ -between interpolation node  $\xi_i \in [t_i, t_{i+1}]$ . Then, if we want  $s$  to be concave, we construct concave quadratic functions over  $[t_i, \xi_i]$  and over  $[\xi_i, t_{i+1}]$  which together make a concave  $C^1$  function  $s$  on  $[t_i, t_{i+1}]$ .

Again, Judd, Numerical, has more on this.  
 $\rightarrow$  Numerical also has an algorithm for solving for  $\xi_i$ .

## 7.8. Multidimensional approximation

when we approx. functions of many variables

→ Tensor product bases

- ⊕: easy extension of orthogonal polynomials and splines

- ⊖: curse of dimensionality

→ Complete polynomials

- ⊕ avoids curse of dim. as it drops those tensor elements that add little to the approx.

→ finite element approaches

? large lib., see Burnett 1978

- bases that are zero at most places

→ neural networks

- previous methods: linear combos of polynomials and trigonometric functions
- NN is inherently nonlinear

DEF. Single-layer neural network

A function of the form

$$F(x; \beta) := h\left(\sum_{i=1}^n \beta_i g(x_i)\right)$$

$x \in \mathbb{R}^n$  is a vector of inputs,  $h$  &  $g$  scalar functions

DEF. Single hidden-layer feedforward network

$$F(x; \beta, \gamma) := f\left(\sum_{j=1}^m \gamma_j h\left(\sum_{i=1}^n \beta_{ij} g(x_i)\right)\right)$$

You fit neural networks by finding

$$\beta = \operatorname{argmin} \sum_j (y_j - F(x^0; \beta))^2$$

or in the case of the feedforward network

$$(\beta, \gamma) = \operatorname{argmin} \sum_j (y_j - F(x^j; \beta, \gamma))^2$$

→ both are just cases of nonlinear least squares.

A little more in Prob, Numerical.

## 8. Applications of approximation to dynamic programming

$$\pi(u, x) = \text{payoff}$$

↑ control state

$$x_{t+1} = g(x_t, u_t) \quad \text{tom state (endog)}$$

then the value function solves

$$V(x) = \max_u \pi(u, x) + \beta V(g(x, u)) =: (TV)(x)$$

(45)

"If we could handle arbitrary functions", we'd start w/ a given  $V_0$  and then compute the sequence  $\{V_n\}$  generated by

$$V_n = TV_{n-1} \quad (\text{makes sense!})$$

"On the computer, however, one cannot store arbitrary functions." We will therefore approximate  $V(x)$  as a finite linear sum of basis functions.

$$\rightarrow V(x) \approx \sum_{i=1}^n a_i \varphi_i(x) =: \hat{V}(x) \quad (51)$$

and we do this using numerical procedures that have  $\hat{V}(x)$  approximately satisfy the Bellman eq,  
(49).

$\Rightarrow$  Objective: find vector  $\vec{a} \in \mathbb{R}^N$  such that  $V$  solves  
(49) as closely as possible.

The task is to replace  $T$ , a functional operator,  
w/ a finite dims approx,  $\tilde{T}$ , which maps functions  
of the form (51) to functions of the same form.

Construct  $\tilde{T}$ :

### 1) maximization step

i) choose a collection of points  $x$

ii) evaluate  $(\tilde{T}\hat{V})(x)$  at each  $x$

$\rightarrow$  the resulting values are points on the function  $\tilde{V}$ .

### 2) approximation step

i) use the resulting values to choose a value function of

the form (51) which best summarizes the info we just generated concerning  $\hat{TV}$ .

→ result is  $\hat{f}\hat{V}$ .

Options for the approximation step:

Curse of  
↑ dim.

- 8.1. Discretization: replace by grids. Inefficient.
- 8.2. Multilinear approx: e.g. Zilles: piecewise linear.
- 8.3. Polynomial approx: e.g.

Bellman et al., polynomials

Daniel, splines

Fudl, tensor-product basis of Chebychev

to solve a 3-dim optimal growth

- ⊕ needs fewer nodes: more efficient and smooth!
- ⊖ do not shape-preserve: screw up convergence.  
⇒ can use the shape-preserving version!

## Ryan meeting

24 April 2020

- 1.) You were right about time consistency  
→ I still think though that the timeless perspective is consistent (maybe?)
- 2.) Susan's def of commitment: the intrinsic power to stick to a committed plan even though it's inconsistent (not subgame perfect)  
Commitment device + punishment strategy  
→ timeless perspective amounts to this

---

## The meeting:

commit. device: a mechanism that makes a time inconsistent problem time consistent  
↳ + punishment b/c that comes from interaction w/ the other player(s).

Reputation in Berns & Gordon is a punishment.

PS wouldn't internalize the promises. →

- timelens is stationary: it's like the ergodic dist'n of to-optimal commitment.
- There's nothing in it mathematically to suggest that the authority has that intrinsic power. But it still has exactly that interpretation: it's a credible way for the policymaker to signal its intention to stick to this plan b/c it's already striking to it today.

to-committment: "We'll let bygones-be-bygones today, and never again"

timelens: "We'll never let bygones-be-bygones. We mean it, so we won't let it happen today either."

→ What I mean is that in learning, commitment doesn't arise b/c even if I promised the commitment plan, the PS wouldn't internalize it.

Lagged multipliers have the interpretation of a promise (but not that I'll keep it); in learning, you don't promise b/c PS isn't able to understand promises.

For GLMM:

- Need to check new term in NKPC
- Sharpen answers (positive & normative theory)
- Need to decide TBC

for draft:

- Need to add lit on correcting TIPS.

Ok, now I wanna work them Examples in  
"Numerical" (p. 233 Mac)

- Spline, Legendre, Chebyshev
  - five-parameter approxes:
    - 4<sup>th</sup> degree least-squares approx/introp at 5 points
    - cubic spline, secant Hermite, 1 guess at 5 points
- ⇒ Legendre least squares
- Chebyshev interpolation
  - spline
- (• polynomial interpolation on a uniform grid - wtf  
is this? I'll ignore it)

First step is to construct recursively the polynomials.

Take some  $x$  as given. Then the recursion formulae  
I take from Judd, Numerical, p. 211 Mac.

