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 fct)

perturbation → local approx

global methods → "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.

→ 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 Judd 1996, Sect 7

Recall: we wanna find representations of fct f which are the continuous fct 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 $\{\varphi_n(x)\}$ is mutually orthogonal w.r.t. weighting function $w(x)$ iff

$$\langle \varphi_n, \varphi_m \rangle = 0 \qquad n \neq m.$$

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

Chebyshev polynomials

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

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

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

Laguerre polynomials

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

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

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

useful: to approx functions of normal random variables

7.2) Least-squares orthogonal polynomial approximation

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

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

Note: Like in GMM, the weighting fct 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(Y,X)" — $\langle f, \varphi_i \rangle$

"Var(X)" — $\langle \varphi_i, \varphi_i \rangle$

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 regressand.

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, $C_n \to f$ uniformly as $n \to \infty$

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

This gives us a way to assess 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^{-k}, we can stop; if they are not, we need to add n+1 order, n+2 etc ...

7.3) Interpolation

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

Lagrange interpolation

Take n points (x_i, y_i) $i = 1, \dots, n$ / where $y_i = f(x_i)$

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

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

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

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

$$h_i(x) := \left(1 - 2 l_i'(x)\,(x - x_i)\right) l_i(x)^2$$

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

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

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

Good for the case when we have data:

$$p(x_i) = y_i \qquad i = 1, \ldots, n.$$

$$p'(x_i) = y_i'$$

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. <u>Interpolation error</u>

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

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

<u>THM 9</u> Bounds on errors of Lagrange interpolant

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

$$\underbrace{\sup_{x \in [a,b]} |f(x) - p_n(x)|}_{\text{"the upper bound on interp errors"}} \leq \underbrace{\|f^{(n+1)}\|_\infty \frac{1}{(n+1)!}}_{\text{"is smaller than stuff we can't influence"}} \underbrace{\sup_{x \in [a,b]} \psi(x; x_1, \dots, x_n)}_{\text{"and stuff we can lower by choosing interp. points wisely"}}$$

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

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

is the Chebyshev interpolation (PTO)

Chebyshev interpolation

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

(which are the zeros of the Tscheby polyns $T_n(x)$) (general Cheby (Judd, 1998 p. 209 Mac))

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-nonparametric 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

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

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:

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

Note: In Judd 1998, Numerical methods, Judd also considers <u>Piecewise linear interpolation</u>

for the data, (x_i, y_i) as

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

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 piecewise 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 knots at the nodes ↓

Splines alleviates 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. Suppose 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.

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

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)

$$i = 0, \ldots, n-1.$$

Splines are great!

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

2.) good fits even for ill-behaved functions (not C^∞)

→ 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 $x_{-k} < \dots < x_{-1} < x_0 < \dots x_{n+k}$.
Order 1 splines implement a step function interpolation and are spanned by B^0-splines:

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

A step function.

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

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

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

Shape-preserving interpolation (!)

The point is: you will shape-preserve automatically as the # nodes $\to \infty$. But you never have ∞ 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.

Piecewise-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

→ filter out liq. premium ⇒ paper.

Peter Fortune at Boston Fed 15 years ago

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

x_{t+1} is summed in π_{t+1}

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

↳ Need to explain that!

Solve nonlinear diff eq numerically.

"Automaton" is the right way to put it, "bw-looking" isn't.

Woodford's timeless 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 ctg 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 TVC are satisfied under $\hat{E}$
→ for the HHs. Useful to know if it is!

- show on IRFs that τ_1 etc are less variable under learn vs. RE

- Motivating episodes -> make 'em clear!
  Missing Defl & Infl.

shape-preserving quadratic splines — 24 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 in-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.
→ Numerical also has an algorithm for solving for ξ.

7.8. Multidimensional approximation

when we approx functions of many variables

→ Tensor product bases

- ⊕: easy extension of orthogonal polys 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 lit, 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

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

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

DEF. Single hidden-layer feedforward network

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

You fit neural networks by finding

$$\beta = \operatorname{argmin} \sum_{0} \left(y_0 - F(x^0;\beta)\right)^2$$

or in the case of the feedforward network

$$(\beta,\gamma) = \operatorname{argmin} \sum_{0} \left(y_0 - F(x^1;\beta,\gamma)\right)^2$$

→ Both are just cases of nonlinear least squares.

A little more in Judd, Numerical.

8. Applications of approximation to dynamic programming

$\pi(u, x)$ = payoff

↑ control ← state

$x_{t+1} = g(x_t, u_t)$ Law state (endog)

then the value function solves

$$V(x) = \max_u \pi(u, x) + \beta V(g(x, u)) =: (TV)(x) \tag{49}$$

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

$V_n = T V_{n-1}$ (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.

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

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

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

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

Construct $\hat{T}$:

1) <u>maximization step</u>

i) choose a collection of points x

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

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

2) <u>approximation step</u>

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

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

$\Rightarrow$ result is $\hat{T}\hat{V}$.

Options for the approximation step: curse of ↑ dim.

8.1. Discretization: replace by grids. Inefficient.

8.2. Multilinear approx: e.g. Zelles: piecewise linear.

8.3. Polynomial approx: e.g.

Bellman et al, polynomials

Daniel, splines

Judd, tensor-product basis of Chebyshev

do solve a 3-dim optimal growth

⊕ needs fewer nodes: more efficient and smooth!

⊖ do not shape-preserve: screw up convergence.

$\Rightarrow$ 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) Susanto'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 Barro & Gordon is a punishment.

PS wouldn't internalize the promises. →

- timeless is stationary: it's like the ergodic distrib of to-optimal commitment.
- There's nothing in it mathematically to suggest that the authority has that intrinsic power. But it STILL has <u>exactly that</u> interpretation: it's a credible way for the policymaker to signal its intention to stick to this plan b/c it's already sticking to it today.

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

timeless: "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 check TVC

For draft:

- Need to check list on correcting TIPS.

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

- Spline, Legendre, Chebyshev
- five-parameter approxes:
  - 4th degree least-squares approx/interp at 5 points
  - cubic spline, secant Hermite, I 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 formulas I take from Judd, Numerical, p. 211 Mac.

Cont. w/ Exercises

25 April 2020

1) Legendre least squares.

Need to compute

$$p(x) = \sum_{k=0}^{n} \frac{\langle f, \varphi_k \rangle}{\langle \varphi_k, \varphi_k \rangle} \varphi_k(x) \qquad (6.4.2)$$

β_k^{ols} — k^{th} regressor

Numerical

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

Damn - to evaluate the inner product I need to numerically compute this integral...

... but I think that's where the 5 gridpoints come into play.

Ok, listen: Newton-Cotes is one way to eval integrals numerically. Gauss-based methods choose the nodes smartly!

$$\int_0^1 x^{1/4}\,dx = \left|_0^1 \frac{4}{5}x^{5/4}\right. = \frac{4}{5} - 0 = \underline{\underline{0.8}}$$

I think I can choose the nodes 26 April 2020

ngrid vs N = max order of polynomial (here 4)
(5)

Calculate nodes x_j for $j = 1, \ldots, ngrid$
and quadrature weights $w^q(j)$.

Evaluate $\psi(k,j)$ (= Legendre polynomials of order $1..k$)
at the nodes $x(j)$.

Evaluate $w^L(j)$ at $x(j)$ (Legendre weights)

Evaluate $f^{true}(x_j)$

→ compute inner_product_num_k

compute inner_product_den_k

→ compute bet_ds(k) = inner_product_num_k / inner_product_den_k

for $x_j = a : \ldots b$

for $k = 0 \ldots n$

fitted_k = beta_ols_k · varphi(k, j)

end

$p(x_j)$ = fitted_k

end

Normally: $\hat{Y} = X\beta \qquad \beta' x'$

$T \times 1 \quad T \times k \; k \times 1 \qquad 1 \times k \; k \times T$

this T can (and I think should!) differ from ngrid b/c we use ngrid to locate $\hat{\beta}$, but then we use more data to plot the true function and the fitted value.

Ok, we're getting there. A new reading of Judd Numerical makes me think that least squares approx uses <u>all</u> "datapoints" to construct $\hat{\beta}^{ols}$, while interpolation only uses ngrid points.

=> which might mean that I need to compute the inner product integrals using a different numerical integration procedure.

→ I want to see what integration procedure Judd recommends "for computing the coefficients of orthogonal polynomial approximations" which is EXACTLY my concern here.

Ok, I think that all should work. I'm just doing some detail wrong. The main difference between the two kinds of "non-sampling" quadratures (i.e. non-Monte-Carlo) is whether they compute the nodes or not:

- Newton-Cotes formulas: do not compute nodes
- Gaussian quadratures: compute nodes & weights

  Idea: roots of orthogonal polynomial family are the quadrature nodes; I guess their weights are the quadrature weights

Newton-Cotes:

- 1. Midpoint rule
- 2. Trapezoid rule
- 3. Simpson rule → done composite version!

Gaussian Quadrature:

- 1. Gauss-Legendre
- 2. Gauss-Chebyshev
- 3. Gauss-Laguerre
- 4. Gauss-Hermite → Normal random vars

Let me look at Gauss-Chebyshev for the simple reason that it seems easy to compute

Judd Numerical p 263 Mac & Pablo lecture-notes2 slide 160

$$\int_{-1}^{1} f(x) \underbrace{w(x)}_{(1-x^2)^{-1/2}} dx \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

$$\underbrace{\omega_i = \frac{\pi}{n}}_{\text{quadrature weights}}, \quad \underbrace{x_i = \cos\left(\frac{2i-1}{2n}\pi\right)}_{\text{quadrature nodes}} \quad i = 1, \dots, n$$

For the interval $[a, b]$

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\frac{\pi}{2}\sum_{i=1}^{n} f\left(a + \frac{(b-a)(x_i+1)}{2}\right)\sqrt{1-x_i^2}$$

where $x_i = \cos\left(\frac{2i-1}{2n}\pi\right)$ are the quadrature nodes like before.

Which we can write as

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

$$x_i^{adapted} = a + \frac{(b-a)\left(\cos\left(\frac{2i-1}{2n}\pi\right)+1\right)}{2}$$

$$w_i^{adapted} = \frac{b-a}{2}\frac{\pi}{2}\sqrt{1-\cos^2\left(\frac{2i-1}{2n}\pi\right)}$$

which is exactly the expression in Pablo's lecture-notes p. 162 mac.

Honestly, I kinda think I'm doing things correctly

now. I'm not getting exactly what pedal gets but I don't know if it's b/c there's some obscure detail I'm doing wrong or what.

(What was wrong btw? My Gauss-Chebyshev node expression had a typo.)

Moving to interpolation, in particular Chebyshev interpolation:

Judd, Numerical, p. 223 Mac

We have set of interpolation nodes: x_i and linearly independent functions $\phi_i(x)$, $i = 1, \dots, n$

w/

$$\phi_i(x_j) = \delta_i^j \equiv \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

↳ cardinal function basis

With a cardinal basis, we can write an interpolating fcn directly as $p(x) = \sum_i y_i \phi_i(x)$

w/ Lagrange interpolation)

$$\phi_i(x) = \prod_{j \neq i} (x - x_j) / (x_i - x_j)$$

Chebyshev Interpolation also chooses the nodes as

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

which are the zeros of $T_n(x)$.

Algorithm 6.2. Chebyshev Regression

(which, if $m = n+1$, is Chebyshev interpolation)

Step 1. Compute the m Chebyshev $[-1,1]$ interpolation nodes

$$z_k = -\cos\left(\frac{2k-1}{2m}\pi\right) \qquad k = 1, \ldots, m$$

↑

I wonder if this "−" is a typo.

I think so!

Step 2. Adjust the nodes to the $[a,b]$ interval

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

Step 3. Evaluate f at the nodes

$$y_k = f(x_k) \qquad k = 1, \dots, m$$

Step 4 Compute Chebyshev coefficients, a_i

$$a_i = \frac{\sum_{k=1}^{m} y_k T_i(z_k)}{\sum_{k=1}^{m} T_i(z_k)^2} \qquad i = 0, \dots, n$$

← this thing is questionable

Step 5 Get approximation

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

Splines in the Examples

27 April 2020

Here we take the interpolation points as given:

in fact, they equal the data points x_i.

ngrid ngrid-1

So we have n+1 points, n intervals, on each of

which we have $s(x) = a_i + b_i x + c_i x^2 + d_i x^2 \Rightarrow 4n$ coeffs

For $i = 1, \dots, n-1$ $i = 2, \dots, ngrid-1$ $4(ngrid-1) = 4ngrid-4$

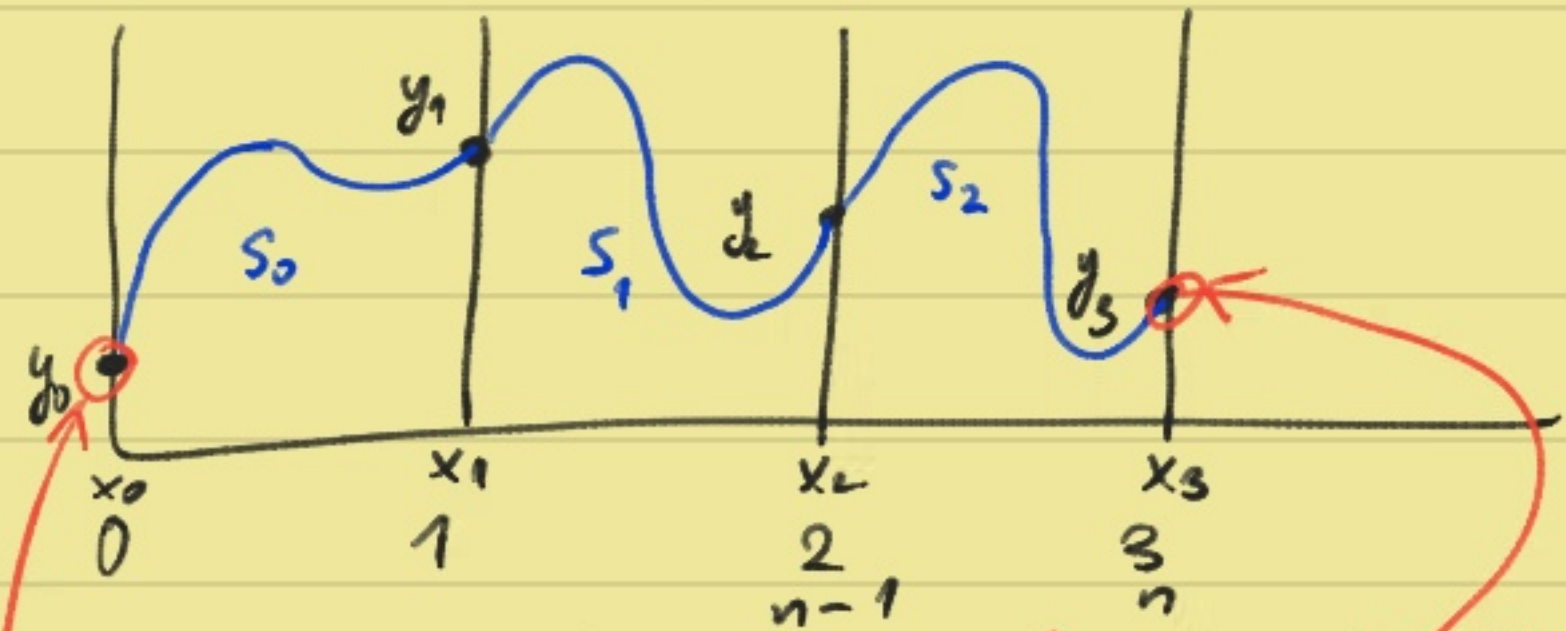
ngrid?

$$y_i = a_i + b_i x_i + c_i x_i^2 + d_i x_i^3$$

For $i = 0, \dots, n-1$ $i = 1, \dots, ngrid-2$

$$y_i = a_{i+1} + b_{i+1} x_i + c_{i+1} x_i^2 + d_{i+1} x_i^3$$

$n = 3$



The point is that is only described by s_0. And that only by s_2.

But all other points (n-1 points) are described by 2 segments.

$\Rightarrow 2(n-1)+2 = 2n$ conditions.

$2(ngrid-2)+2 = 2ngrid-2$

Need more conditions: for $s(x)$ to be C^2 at interior nodes, we impose $2(n-1) = 2n-2$ more conditions: for $i=1,\dots,n-1$ $\quad i=2,\dots,ngrid-2$

1st deriv: $b_i + 2c_i x_i + 3d_i x_i^2 = b_{i+1} + 2c_{i+1} x_i + 3d_{i+1} x_i^2$

2nd deriv: $2c_i + 6d_i x_i = 2c_{i+1} + 6d_{i+1} x_i$

Ok, so we have $2n + 2n-2 = 4n-2$ equations in $4n$ unknowns. Need 2 more eqs.

Secant Hermite spline imposes:

$+1$: $s'(x_0) = \dfrac{s(x_1) - s(x_0)}{x_1 - x_0}$

$+2$: $s'(x_n) = \dfrac{s(x_n) - s(x_{n-1})}{x_n - x_{n-1}}$, which means

$\Leftrightarrow \quad b_0^{1} + 2c_0 x_0 = \dfrac{a_1 + b_1 x_1 + c_1 x_1^{2} + d_1 x_1^{3} - \left(a_0 + b_0 x_0^{1} + c_0 x_0^2 + d_0 x_0^3\right)}{x_1 - x_0}$

and $ngrid-1$ $\qquad ngrid-2$

$$b_n + 2c_n x_n = \frac{a_n + b_n x_n + c_n x_n^2 + d_n x_n^3 - \left(a_{n-1} + b_{n-1} x_{n-1} + c_{n-1} x_{n-1}^2 + d_{n-1} x_{n-1}^3\right)}{x_n - x_{n-1}}$$

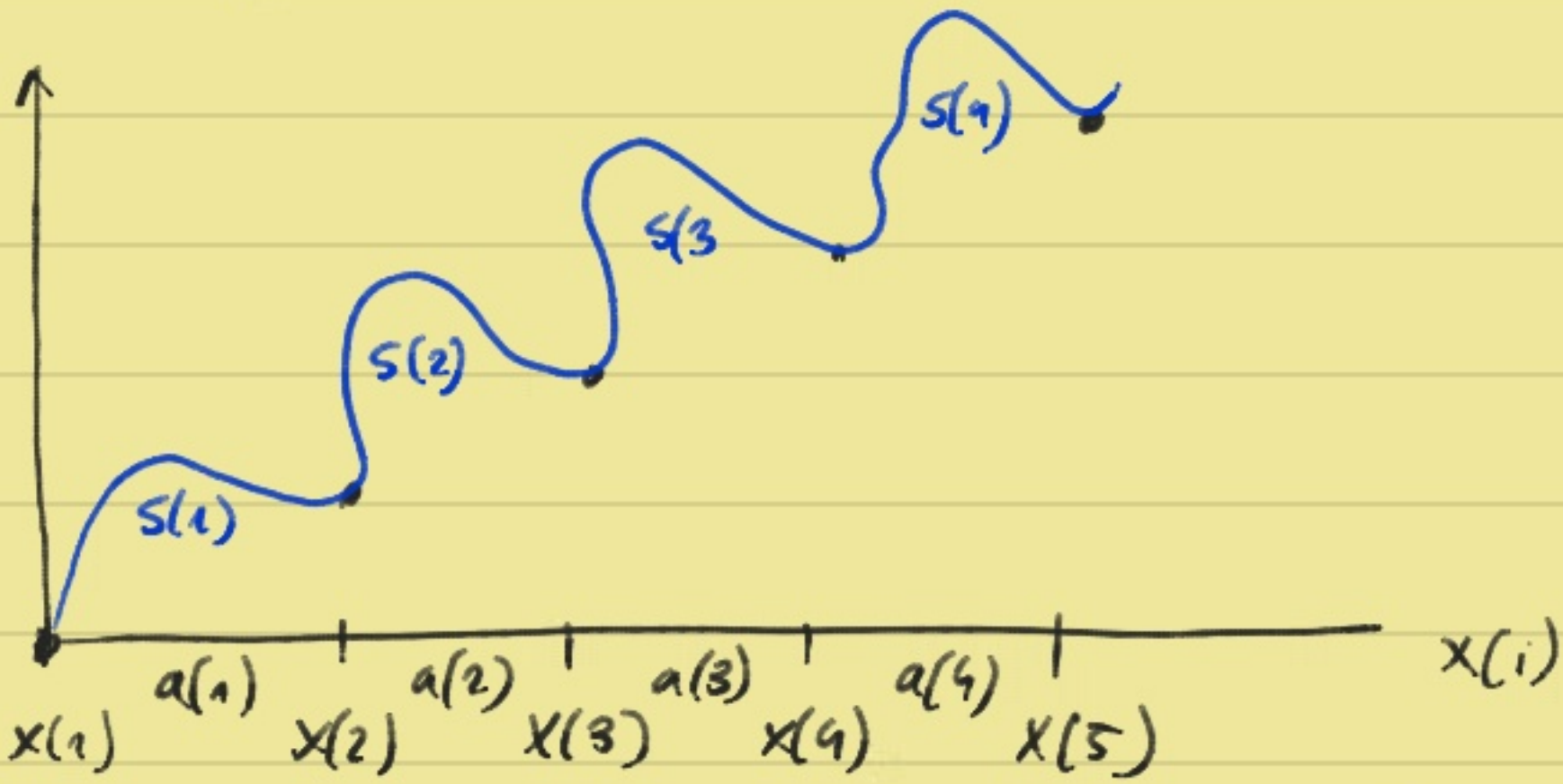
Now we're using 5 points $\longrightarrow$ $ngrid = 5$, I'm gonna call

the first point 1, so we have ngrid - 1 intervals.

Let me redraw:

ngrid = 5 → 5 nodes, i = 1, ..., 5

n = 4 (4 intervals)



Interpolating conditions + continuity at inner nodes

$y(i) = a(i) \dots x(i)$ $\quad i = 1, \dots, 4$ $\quad i = 1, \dots, n$ (ngrid - 1)

$y(i) = a(i-1) \dots x(i)$ $\quad i = 2, \dots, 5$ $\quad i = 2, \dots, n+1$ (ngrid)

Twice differentiability at inner nodes: $i = 2 \dots, 4$ i.e. $i = 1, \dots, n$ (ngrid - 1)

$i+1 \Rightarrow i-1$

Guess what: my pretty little spline is working. They are all doing a decent job, except for the Chebyshev interpolation which is fluctuating a bit too much, also the Legendre least squares, but not as much.

If z_k is the Chebyshev node on $[-1,1]$, 28 April 2020

then $x_k = (z_k+1)\left(\frac{b-a}{2}\right) + a$ is the adapted node.

Express z_k: $x_k - a = \frac{b-a}{2} z_k + \frac{b-a}{2}$

$$\Rightarrow z_k = \left(x_k - a - \frac{b-a}{2}\right)\frac{2}{b-a} = \frac{2x_k - 2a - (b-a)}{b-a}$$

$$= \frac{2x_k - a - b}{b-a} = \frac{2x_k - b - a \pm a}{b-a} = \frac{2x_k - (b-a) - 2a}{b-a}$$

$$= 2\frac{x-a}{b-a} - 1$$

Note that when you compute the Chebyshev coeffs as in Judd, Numerical, vs. as in Cai & Judd 2014, you get the SAME coeffs! I.e.

m = # nodes, $n+1$ = deg T.

$$a_i = \frac{\sum_{k=1}^{m} y_k T_i(z_k)}{\sum_{k=1}^{m} T_i(z_k)^2}$$

$$= b_j = \frac{2}{m} \sum_{i=1}^{m} y_i \, T_j(z_i)$$

which is kind of suggesting that

$$\frac{2}{m} = \frac{1}{\sum_{k=1}^{m} T_i(z_k)^2} = \frac{1}{\frac{m}{2}} \quad \text{i.e that}$$

$$\sum_{k=1}^{m} T_i(z_k)^2 = \frac{m}{2}$$

"Sum of squared Tschebys up to order m equals $m/2$."

I don't find anything on the web on this, but it seems intuitive.

Peter meeting

quadrature: Riemann sums

→ a more efficient way of doing it?

Yes!

Replace dx w/ w_j is what you're doing!

↳ Locate error by trying this w/ STG simples:

like not $(x+1)^{1/4}$, but a standard polynomial.

Interpolation vs. Approximation are synonymous

b/c the unknown object has uncountably ∞ dims

1. Eval $(x+1)^{1/4}$ for $x \in (0,1)$

Do it for 100 diff vals by increment 0.01

for $x = 0.011$, I'll report $f(0.01)$

→ that's an approx

But it's also an interp: b/c it takes it as

a step function
⇒ represent it as a finite-dim vector.

⇒ can do this for VFI where the Bellman eq doesn't hold exactly, but can min w/ eg a least squares loss.

↳ For more states, curse of dim.

⇒ so "are there other ways of describing the ∞-dim objects?"

→ we'd think Taylor-approx.

⇒ a most efficient way is then polynomials.

Uniqueness of interp polynomial is important b/c you want the Bellman eq to be satisfied

$$V(\cdot) = \max u + V'(\cdot)$$

→ you want $\hat{V}$ chebyshev not to toggle back &

forth between two approxes.
→ so "uniqueness wrt a basis".

Interpret spline

$$b_i = c_i = d_i = 0$$

→ f is constant on each of the intervals
→ "discretizing the state-space"

(a_i, b_i, c_i, d_i) $i = 1, \dots, n$

→ so w/ a vector of length $4n$, I'm approx a continuous-valued fct.
⇒ that's the analogy to the polynomials.