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 ...

4.3) 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

$$\sup_{x \in [a,b]} |f(x) - p_n(x)| \leq \|f^{(n+1)}\|_\infty \frac{1}{(n+1)!} \sup_{x \in [a,b]} \psi(x; x_1, \dots, x_n)$$

"the upper bound on interp errors" — "is smaller than stuff we can't influence" — "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 (P70)

Chebyshev interpolation

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

(which are the zeros of the Tscheby polys $T_n(x)$) (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:

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

Note: In Judd 1998, Numerical methods, Judd also considers Piecewise linear interpolation for the data, (x_i, y_i) as

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

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.

$\to$ orthogonal polynomials usually do not preserve shape.

Piecewise-polynomial approx does.

$\to$ so our discussion of piece-wise linear/polynomial interp/approx is shape-preserving.

But it's not smooth.

$\Rightarrow$ 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 sthg 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 TI etc are less variable under learn vs. RE

- Motivating episodes -> make 'em clear!
  Missing Dot & Iwff.

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

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

or in the case of the feedforward network

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

→ 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

(u: control, x: 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 $\hat{V}$ 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. Zeldes: 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 $k = 0 \ldots n$

    $\text{fitted\_k} = \text{beta\_ols\_k} \cdot \text{varphi}(k, j)$

  end

  $p(x_j) = \text{fitted\_k}$

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 create $\hat{\beta}$, 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 $[a, b]$

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

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

Which we can write as

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

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

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

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

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

4(ngrid-1) = 4ngrid-4

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

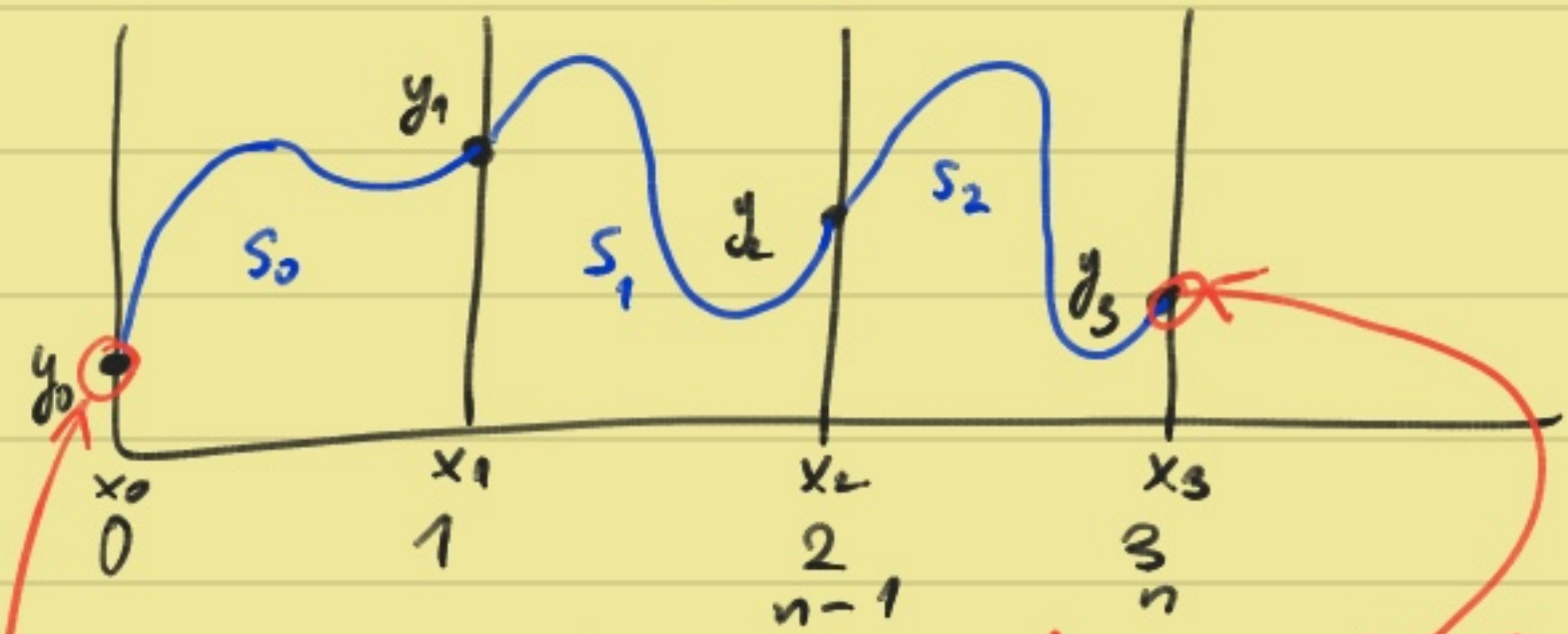
ngrid?

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

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 hit & only described by s_0. And this only by s_2.

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

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

$2(\text{ngrid}-2)+2 = 2\text{ngrid}-2$

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

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

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

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

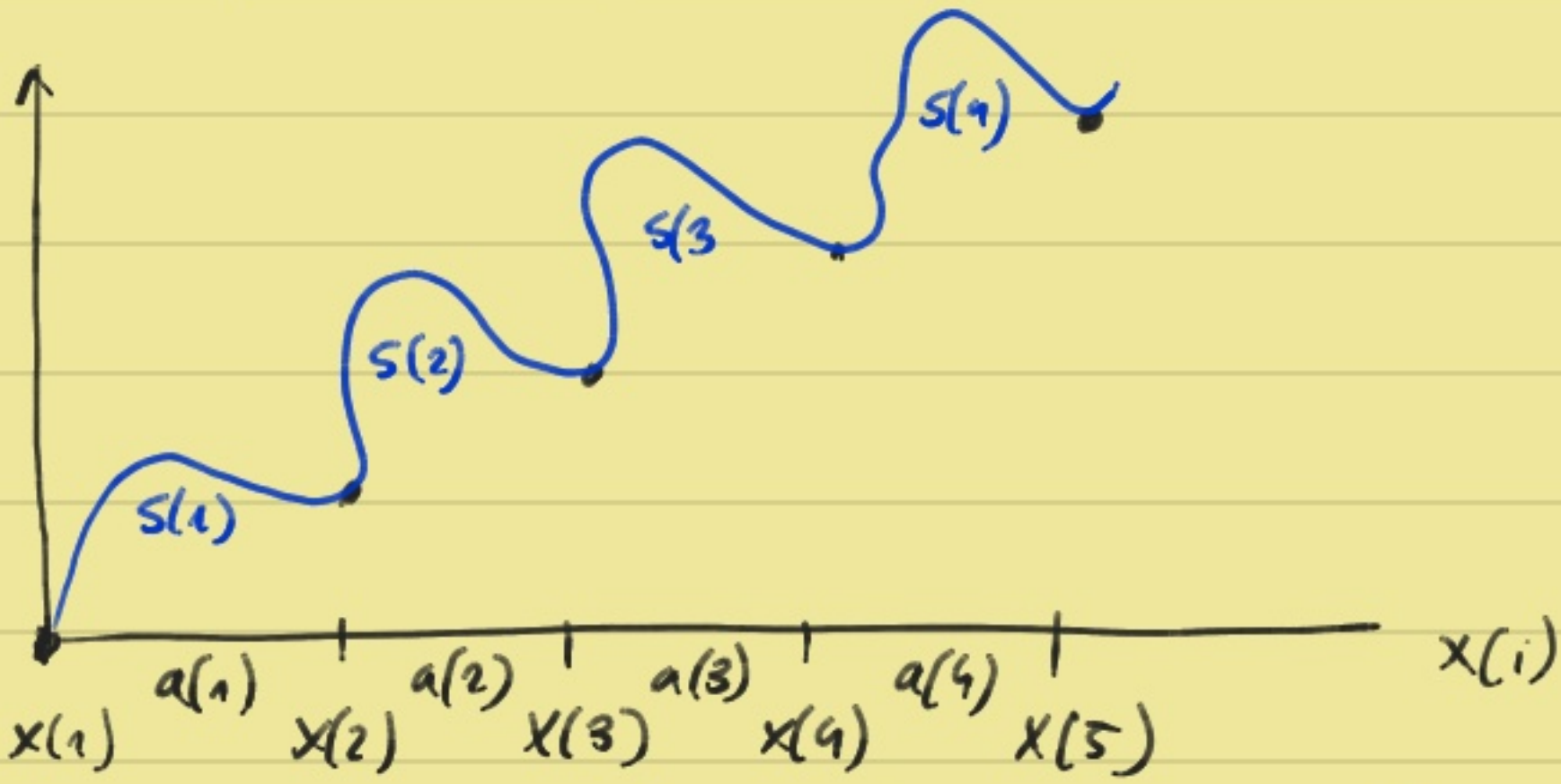
Now we're using 5 points $\longrightarrow$ $ngrid = 5$, I'm gonna call

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

ngrid-1

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

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

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

$i \leftarrow 1 \Rightarrow i-1$

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 fndd, 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 28 April 2020

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 thru 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.

Work after

Trying to find the hypo:

for each k, $\beta_k^{OLS} = \dfrac{\langle f, \varphi_k \rangle}{\langle \varphi_k, \varphi_k \rangle}$

So given φ_k, and f, more concretely,
for x_j = nodes, we're given $\varphi_k(x_j)$, $f(x_j)$.

$$\sum_{j=1}^{n_{grid}} w_j \cdot f(x_j)\,\varphi_k(x_j)\,\omega^{Legendre}(x_j)$$

So:

- nodes are fine
- quadrature weights are fine
- true fun is false
- Legendre polynomials are fine, both during the quadrature, both in the data.

→ The only thing that can be screwed up is the inner products!

Somehow the Legendre least squares method oscillates too much b/c $\beta_k^{OLS} = 0$ for k odd.

This comes from the fact that the inner product $\langle f, \varphi_k \rangle = 0$ for k odd.

I think the reason for this is that the summands of the inner product are mirrors over the x-axis, i.e. they equal each other $\times(-1)$.

And this isn't b/c of the interval $[-1,1]$ b/c a different one $[1,5]$ also leads to way too big Legendre interpolants.

Ah: input $n-1$ in legendre! (in code: $k-1$)

By decreasing the input degree, you can dampen the amplitude of the legendre.

$$(l+1)P_{l+1}(x) - (2l+1)xP_l(x) + lP_{l-1}(x) = 0$$

$$(l+1)P_{l+1}(x) = (2l+1)xP_l(x) - lP_{l-1}(x)$$

$$P_{l+1}(x) = \frac{2l+1}{l+1} \cdot x \cdot P_l(x) - \frac{l}{l+1} P_{l-1}(x)$$

I don't know! Everything seems correct!

Comparing 4th order Legendre P_4 using the recursive formula vs. analytical expression for it from Wolfram reveals that the 2 coincide only if I input $n-1$ instead of n in my recursive relation

↳ that amounts to $k-2$ in the approx.

that does a decent job, but still not as great as before. Fine, I'll leave it here.

Ok, now that approximations of 29 April 2020
functions work, I wanna understand how it's done for value function iteration. Let me see in Cai & Judd, Collard & Eric Sims.

What surprises me somewhat is that in both Collard & Sims, interpolation is presented as a problem that for a control var, the state does not lie on its grid

so that the value function is unknown at this value.

This is a related, but not identical problem.

However, in Collard, there is something called "parametric dynamic programming" which approximates the value function's functional form and iterates on the functional form parameters.

→ and indeed, this is described in Judd, Numerical, p. 122 Mac, and also denoted "parametric approach".

Like Collard says, Judd says that the idea is to not restrict states and controls to take particular values (i.e. not to discretize).

Instead, we approximate $V(x) = \hat{V}(x, a)$

where I'll betyn that a will be coefficients.

The functional form of $\hat{V}$ can be chosen i.e.

it may be a linear combination of polynomials, a neural network or something else, say a spline. Collard does this w/ Chebyshev polynomials. Thank God!

Let's write the "Parametric Dynamic Programming with Value Function Iteration" Algorithm,

Judd, Algorithm 12.5 p. 123 Mac

Collard, p. 20 Mac.

P TO.

ALGORITM: Parametric dynamic programming.

Objective: Solve Bellman equation.

Initialization: Choose a functional form for $\hat{V}(x; a)$ and choose the grid of interpolation nodes $X = \{x_1, \dots, x_n\}$ Choose initial vector of parameters $a^0 \Rightarrow \hat{V}(x, a^0)$ and stopping criterion $\varepsilon > 0$.

Step 1. Maximization step

Compute $v_j = T\hat{V}(\cdot, a^i))(x_j)$

$$= \max_y u(y, x_j) + \beta \hat{V}(x', a^i)$$

$$\text{s.t. } x' = h(y, x_j) \quad \text{for } x_j \in X$$

Step 2. Fitting step

Using your chosen approximation method, compute the next vector of parameters a^{i+1} such that $\hat{V}(x; a^{i+1})$ approximates the (v_i, x_i) data.

(I think this is concise)

Step 3. If $\| \hat{V}(x; a^i) - \hat{V}(x; a^{i+1}) \| < \varepsilon$, STOP.

Else go to Step 1.

Ryan meeting — 29 April 2020

Let's see if implementing the TC via inputting exog. sequences as spelled out in materials 26 was correct or not.

The answer is no. 🙁

Obs.

1.) Initial last period RGDP is not 0

→ and it's = $\bar{i}$(end)

Ryan sets → solved! Zero out $\bar{i}$ before setting e = squeeze(eN)

2) # inputs not in line w/ number of eqs.

↳ # eqs = should be 300

3) There's stuff underidentified at the ends

1st & last obs.

→ you have some vars going in that don't

affect the sol

98 params w/ 100 equations

couldn't be solved if there wasn't also some under ID!

↳ Some eqs are repeating info / or are repeated.

4) options.UseParallel = true

⇒ Point 2) is actually mysterious: Began initially thought: if input i_t $(T=100)$, then shouldn't use reads $\begin{matrix} a_t \\ x_t \\ \vdots \\ i_t \end{matrix}$ $t=1,\dots,100$ b/c then we have 300 eqs for 100 inputs. But strangely, when he ran it w/ only the TR reads (100 eqs) the computer had a harder time.

Now I'm in main_file.m in the input_sequences_for_Ryan folder and I'm adding an fsolve to it.
→ indeed, now even this file can solve it, even if you initialize away from the TR.

Instead of 9 sec, parpool w/ 2 workers is 65 sec.
And I only have 2 cores, so I can't do better.

Oh shit – it was working for Ryan b/c he had set people to intermediate the TR...

- Now the MSUM complains if Agents don't know the TR
- CEMP can't solve w/o knowing the TR — can solve w/ knowing the TR
- smooth cost can't even solve w/ knowing the TR!
- Logsin solves when agents know the TR AND when they don't!

- I tried the scalar cumsum
  - it finds a diff sol when agents know the TR
  - it finds no sol when agents don't know the TR
- Cgain can solve if agents do know the TR
  can't solve if agents don't know the TR
- Cusum can't solve either if agents don't know the TR
  and so explodes, that's why we get the warnings that I've turned off in fk_cusum_vector.

⇒ what I'm learning is that my previous conclusion obtains: the stabilizing effect of knowing the TR is unchallenged, except for cgain b/c there the gain seems to go to zero faster than things could diverge.

Back to VFI w/ approximation — 30 April 2020

Let me write the maximization step as full antes id,

p 129 Mac:

$$v_j = \max_{u \in D(x_j)} \pi(u, x_j) + \beta \int \hat{V}(x', a)\, dF(x' | x_j, u) \quad x_j \in X$$

⇒ "3 kinds of problems we need to handle"

1.) Evaluating the integral: support of states is smooth

→ Discretizing the shock amounts to approx of the integral as a step function.

- If ∫ smooth, Gaussian quadrature.
- If not, low-order Newton-Cotes.
- If ∫ high-dimensional, (quasi-) Monte Carlo method.

2) Optimization problem: Newton's method? ⟶

3) Approx of V: i.e. computation of a.

- If V smooth, orthogonal polynomials
- Else, splines. ⇒ in particular Schumaker shape-pres.!

A detour on numerical optimization

Algorithm 4.2 Newton's Method in R^1 judd, p. 104 Mac

Initialization: Choose initial guess x_0 and stopping parameters $\delta, \varepsilon > 0$

Step 1. $x_{k+1} = x_k - f'(x_k) / f''(x_k)$

Step 2. If $|x_k - x_{k+1}| < \varepsilon(1 + |x_k|)$ and $|f'(x_k)| < \delta$

(step size; gradient)

stop & report success; else go to Step 1.

It seems to me that Newton's method is a curvature-based method.

comparison methods

pick a set of points & choose the largest

gradient methods

use gradient to determine direction of search

↳ **curvature methods**

use curvature info too to determine direction of search.

It's obvious that

- comparison methods are cheap b/c they do not require evaluation of $f'(\cdot)$ or $f''(\cdot)$ but are slow b/c they do not climb smartly
  → suited for non-smooth problems
- gradient & especially curvature methods are faster, but costlier (especially Hessian f'' is costly to compute)
  → suited for smooth problems

Multi-dimensional optimization:

1) Comparison:
   - grid-search: pick a grid, compare values $f(x_1^i, x_2^i)$
     choose the largest $f(\cdot) \to x_1^*, \ldots, x_n^*$.
     I think this is misleading b/c grid search can be implemented w/ gradient methods.
   - polytope methods (ha! never heard of this!)

2) Gradient & curvature methods

Newton's Method for Multivariate Problems

Let

$$\nabla f(x) := \left(\frac{\partial f}{\partial x_1}(x), \dots, \frac{\partial f}{\partial x_n}(x) \right) \quad \text{gradient}$$

$$H(x) := \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right)_{i,j=1}^{n} \quad \text{Hessian}$$

Then

$$x^{k+1} = x^k - H(x^k)^{-1} \left(\nabla f(x^k) \right)' \qquad (4.3.1)$$

See algorithm 4.4, p. 111 Mac.

Time-out: This is why my parametric VFI is so much slower than Collard's b/c I choose a gradient optimization step instead of a comparison one.

- Direction methods → Newton's is a special case
- Coordinate directions } these two sound smarter
- Steepest descent } but are slower than Newton.

Direction search – the general idea ALG. 9.5.

Initialization. Choose initial guess x^0 and stopping params δ and $\varepsilon > 0$

Step 1. Compute search direction s^k

In Newton's method, $s^k = -\text{gradient}/\text{Hessian}$

Step 2. Solve $\lambda_k = \arg\min f(\underbrace{x^k + \lambda s^k}_{\text{a line given by the direction}})$

Step 3. $x^{k+1} = x^k + \lambda_k s^k$

In Newton's method, $\lambda_k = 1 \quad \forall k$.

Step 4. If $\|x^k - x^{k+1}\| < \varepsilon(1 + \|x^k\|)$ go to Step 5, else to 1.

Step 5. If $\|\nabla f(x^k)\| < \delta(1 + f(x^k))$ stop & report success, else stop & report convergence to nonoptimal point.

- Conjugate Gradient method

Essentially, solves 2 problems

1. Coordinate directions & steepest descent are slow b/c they move in a direction that's optimal on one dimension, but not on the other → conjugate directions are directions that are optimal in all dimensions

2. Does not calculate & store Hessians.

4.5 Nonlinear least squares (Isqnonlin)

$$\min_x \frac{1}{2} \sum_{i=1}^{m} f^i(x)^2 \equiv S(x)$$

↙ m-valued function.

Let $J(x)$ be the Jacobian of f. Then the NLS problem has the following nice features:

- the gradient of $S(x) = J(x)^T f(x)$
- the Hessian of $S(x) = J(x)^T J(x) + G(x)$

where $G_{j,l}(x) = \sum_{i=1}^{m} f^i_{j,l}(x) f^i(x)$

Time-out: gradient vs. Jacobian.

When $f: \mathbb{R}^n \to \mathbb{R}$, the matrix of partial derivatives is a $1 \times n$ vector and is called the gradient. "f is a scalar-valued function"

When $f: \mathbb{R}^n \to \mathbb{R}^m$, the matrix of partial derivatives is a $m \times n$ matrix and is called the Jacobian. "f is a vector-valued function".

$\Rightarrow$ The nice thing is we never have to take 2nd order derivatives I guess except in $G(x)$ term. Thus becomes nicer when $f(x) = 0 \to G(x) = 0$ so that we only have to take no 2nd derivs.

$\Rightarrow$ When we're close to the sol ($f(x) = 0$), $J(x)^T J(x)$ is a good (& cheap to compute) approximation of the Hessian

$\Rightarrow$ the Gauss-Newton method uses $J(x)^T J(x)$ as an estimate for the Hessian!

Gauss-Newton method = Newton's method except:

$$s^k = -\left(J(x)^T J(x)\right)^{-1} \left(\nabla f(x^k)\right)^T$$

→ much faster.

Problems:

- $J(x)$ or $J(x)^T J(x)$ ill-conditioned
- s^k may not be a descent direction

  ↳ how on all earth not? Is it b/c $J(x)^T J(x)$ is a poor estimate of Hessian?

→ **Levenberg-Marquardt algorithm**

$$s^k = -\left(J(x)^T J(x) + \lambda I\right)^{-1} \left(\nabla f(x^k)\right)^T$$

(λ: scalar, I: identity)

→ s^k is a (steepest) descent direction for λ large

→ λI reduces the conditioning problems

Constrained optimization (fmincon)

is apparently much harder to program robustly on your own, according to pudd.

Penalty function methods

- Sequential Quadratic Methods
- Reduced Gradient methods
- Active Set Methods

Constrained opt problem has the form

$$\min_x f(x) \qquad \text{s.t.} \quad g(x) = 0$$
$$h(x) \leq 0$$

The solution is a critical point of $\mathcal{L}$

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda' g(x) + \mu' h(x)$$

Penalty function methods consider an unconstrained problem, but alter the objective function so that violating the orig. constraints is extremely painful.

BU GCMM simulation

- Breakout rooms

  ↳ can people enter, return

  can sme enter an existing one

- Can co-hosts create breakout rooms → No, only host.
- A background w/ instructions for screenshare.

⇒ ~~everyone who presented / poster~~

~~make can co-host~~

↳ Let's create a breakout room for every presenter

- Dummy room for non-presenter

- U needs to be co-host in order to be able to brand
- breakout room ~~for~~ all the presentors / posters.
- Main room will be the bar
- Hallway → non-presenters

- Swyppr password to Bot
- Breakout rooms for posters

secretpw

So the password works but you need to send the invite link up to me?
(not including)

Penalty function methods cont.'

So instead of $\min_x f(x)$ s.t. $g(x)=a$ & $h(x)\leq b$

we write

$\min_x f(x) + \frac{1}{2} P(\text{some fct of } g(x)-a \text{ and } h(x)-b)$

As $P \to \infty$, the new problem and the old coincide. In practise, these methods use iteratively larger P_s' (initially small) to overcome the problem that a large P results in an ill-conditioned Hessian.

Sequential quadratic methods (from Matlab sqp)

solves a sequence of quadratic problems:
quadratic approx of Lagrangian, linear
approx of constraints.

Augmented Lagrangian methods

in principle these combine penalty function
& sequential quadratic. (I wonder if
Matlab's 'sqp' is this?) Sounds really
just like penalty.

Reduced Gradient methods

- specify a set of independent vars
- solve for "dependent vars" using constraints
- optimize over the remaining indep. vars
(nonlin) in an approximated setting.

Active set methods (fmincon Active Set)

We don't wanna compute all constraints especially if only a few bind. A.S. therefore considers a subset of the constraints, the "active set" and thus only solves a series of subproblems.

→ it adds & drops constraints as it goes, as it periodically checks to see if there are some that hold or are violated.

In practice, existing solvers usually use sophisticated combos of these.

Now I'm reading the Matlab documentation b/c now I wanna know what Matlab uses when! I wanna know!

Large-scale method: one that does not need to use, operate on, or to store full matrices.

→ uses sparse matrices

Medium-scale: one that uses full/dense matrices

→ so it seems like from a computational perspective you'd wanna use large-scale methods

For many things, like fmincon, the "interior-point" algorithm seems to be the default.

Recap

① Unconstrained opti

Comparison methods

Gradient / curvature methods

- Direction methods → Newton's is a special case
- Coordinate directions
- Steepest descent
- Conjugate gradient (CG)
- Nonlinear least squares
- Gauss-Newton
- Levenberg-Marquardt

fminunc quasi-newton (the other is a trust-region based)

fsolve (the others are trust-region based)

lsqnonlin (the others are trust-region or interior point)

② Constrained opti

- Penalty function methods
- SQP sequential quadratic methods fmincon sqp
- Augmented Lagrangian methods fmincon interior point?
- Reduced gradient
- Active set fmincon active-set

fmincon trust-region seems like a combo of direction methods
fmincon interior point is like a combo of penalty & direction.

Let me discuss trust-region & interior point algorithms since they seem to be the core of Matlab's routines.

Interior point in fmincon

The original problem is

$$\min_x f(x) \quad \text{s.t.} \quad h(x) = 0 \quad \text{and} \quad g(x) \leq 0$$

The approximate problem is:

$$\min_{x,s} f_\mu(x,s) = \min_{x,s} f(x) - \mu \sum_i \ln(s_i) \quad \text{s.t.} \quad h(x) = 0$$
$$g(x) + s = 0$$

→ where $\mu \sum_i \ln(s_i)$ is called a barrier function and it seems to serve to remove the inequality constraint as a penalty.

The algorithm uses 2 kinds of steps to solve the approx. problem:

- direct step / Newton step
- a conjugate gradient step (CG), using a trust-region.

Trust-region in general

The original problem has the same form. But now the idea is to approx f w/ a simpler function, q, which I guess is supposed to approx f well in a neighborhood N around the point x (where we are). This neighborhood N is the trust-region.

The trust-region subproblem is

$$\min_s \{ q(s) \quad s \in N \}$$

$$\Rightarrow \quad x^{n+1} = x^n + s \quad \begin{cases} \text{if} \quad f(x+s) < f(x) \\ \text{else} \quad x^{n+1} = x^n \quad \text{and } N \text{ is} \end{cases}$$

shrunk, the subproblem repeated.

In the std approach, the approx. q is the first two terms of a Taylor approx of f at x.

Ok, I think that's it. Let's return to VFI w/ interpolation

1 May 2020

I've been able to get Collard's parametric VFI to work, but there are some details I don't understand.

- e.g. the coefficients a^i or b^i should really be $(n+1) \times 1$, where n = order of Chebyshevs, but it they actually end up being $(m \times 1)$, where m = # interpolation nodes. (Ha! This is what Judd was yappin' about!)
- The other thing is that he never seems to compute (directly) the coefficients, he infers them from V^{new} instead.

One thing I observe is that Collard's 2 May 2020 way of backing out b^i seems to imply bigger b^i than mine → therefore his value fct is an order of magnitude bigger than mine. When he infers them,

he uses Matt $V = \sum_j b_j^i T_j(z)$

i.e. $V = b\ T(z)$
$(1 \times n+1)(n+1 \times m)$

$V T(z)^{-1} = b$

Except that Collard writes

$\theta = T(z)^{-1} V$
$m \times 1 \quad (m \times n+1)\ (m \times 1)$

You can see both Rhett Collard's approach does require $n+1 = m$

b/c $V = T(z)\ \theta$
$(m \times 1) \quad (m \times n+1)\ (m \times 1)$

What's vexing me is that

$T(z)^{-1} V' \neq V T(z)^{-1}$

and if I use $V T(z)^{-1}$, I get qualitatively wrong things too.

Ok, so I switched back to computing the Cheby coefficients - it's faster, too. The big problem is that k_F^* never really changes.

This is surprising b/c I'm essentially using the same function as Collard.

Silly comment: I'm not sure I know what I changed: I thought I only added kp(j, iter) to fminunc and now it's better:

Things look better but kp is still kinda too small at the tail end, which means that c is too low at the tail end.

Ok - I can see stg more clearly now: somehow the stopping criterion alternates btwn stg c-14 and about 0.05.
→ need to look into that!
But it only alternates if I consider my b', not if I do the correctly computed Collard's.

Hm... at this point I feel I really understand the code and I'm feeling more & more convinced that there's a silly error somewhere in TV, vhat_cheby, compute_cheby_coeffs or chebyshev.

⇒ So next step: Redo the own version w/ the sizes of things like Collard suggests so I can verify where my functions give the same and where not as Collard's.

Speaking points for Green Line Poster 3 May 2020

4 take-home points

① RE (NK) models can't explain deviations from TR or say anything about interplay of mrkpd & expectations

② But empirical evidence → expectations NOT RE: empirical work (e.g. Milani 2014, Ensep & Preston 2011 AER) suggest learning can match
- persistence of TL
- time-varying volatility
- low FEs

③ Instead, this model makes explicit the link between monpol & expectations b/c interest-rate setting affects E(·) then realized inflation, which affect π.

④ the CB needs to respond differently to shocks based how it affects anchoring today & in the future b/c these determine the config of intra- & intertemporal tradeoffs.

Anchoring E(·)-formation allows CB to postpone the intra-temporal tradeoff betwn π & x b/c E(·) don't (cannot) incorporate promises

SR-LR tradeoff in anchoring due to volatility and speed & extent of convergence to RE distortion.

Elevator pitch → Info design

What else I'm working on:

- Communication in monpol only static → dynamic bc endog prior.
  - ↳ Bayesian persuasion. Anchor of fwd determines persuasion ability.
- ICT a GPT in last 20yrs in US? SVAR & IR-matching 2-sector growth model - yes

Who to talk to:	Was there? AM	PM
-Bob King	✓	
-Adam Guren		
-Stephen Terry		
-Tarek Hassan		
- Pablo		
- Rosen		
- Fabio		

Who else was there:

AM.

Jenny Tong → empirical stuff

CB only has i-rate as a tool: (communication would be important)
SR vs LR E(·) interacting

Liyang, Luca, Marco, Carter, Tony Zhang (Ohnstrom)

PM. Tony: Show what's lost (welfare) if you don't account for this friction

Luke, Ryan

Francesco d'Acunto → lot of work on TI-E(·)
w/ Michael Weber
⇒ nice data on cognitive limitation
he's at BC business school (cannot)

Bob also texted me about Sargent's (congress of inflation.

Jenny Tang — 5 May 2020

Schleifer: diagnostic E(·)

E(·)-data on FX-rates (survey)
→ estimated Schleifer's eqs and sign is flipped.
⊖ instead ⊕

Suggestion:
empirical lit & policy makers: unanchoring as

CB losing credibility

→ gain fct of (Ti and target)

⇒ result in similar opt policy

↳ stay even more stark: the target is on

period-by-period-target

in eq. 5: $g(\pi_t - \pi^{target})$

Monetary Policy Framework Review at Fed

↳ price-level-targeting (PLT)

This aspect can introduce a new layer to that discussion.

→ the things you address/promise influences what ppl think

→ one framework Fed considered was to target average π in long-run (instead of π in each period)

↳ if what the Fed states changes (eq 5) is

⇨ formally stable iff target as a LR avg

- Amazon & some private firms might do pooltools

Facebook

Boston Fed has 1-2 students as Diss Fells.
The Board has 12. Apply!

Could do:

historical counterfactuals

Would this policy (i-rate) have been feasible sooner or more frequently?
→ or resulted in much more i-rate volatility?

- There are a bunch of empirics possible for a follow-up paper.

Reach out to Jenny mid-summer.

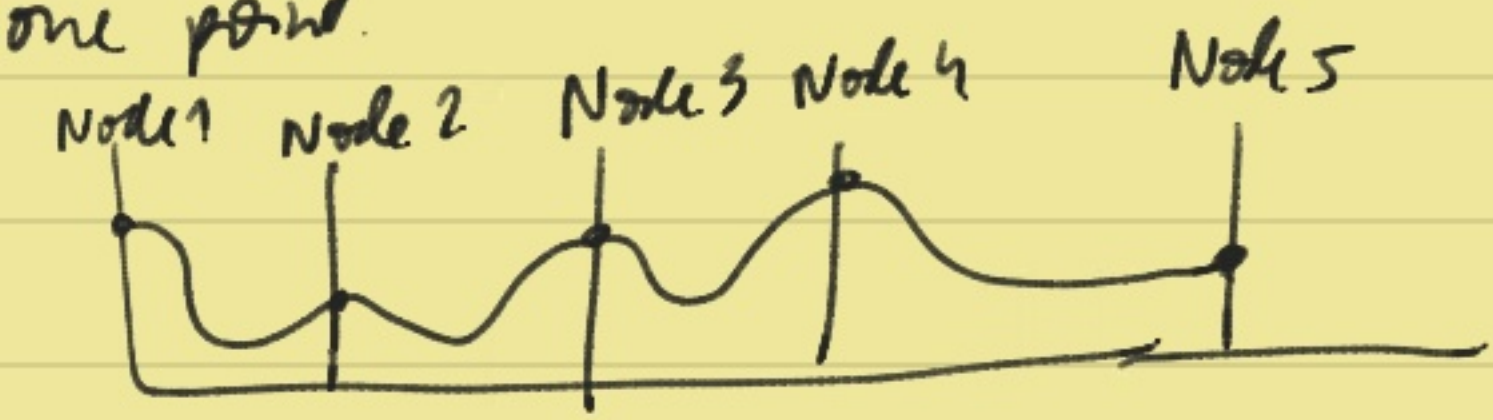
Peter meeting 5 May 2020

- optimal growth model — deterministic
  - w/ interpolation
    - Akeby
    - spline
    - shape-preserving
  - stochastic
- multivariate approx

Back to work

Need to get the spline to work.

The issue is that what_spline always gives back a vector. What I need instead is that it should first evaluate the value function at one point.



Maybe I have a more efficient way of calculating the spline coeffs. 6 May 2020

$$s(x_j) = a_i + b_i x + c_i x^2 + d_i x^3$$

$$\underset{1\times 1}{s(x_j)} = \underset{1\times 4}{\begin{bmatrix} a_i, & b_i, & c_i, & d_i \end{bmatrix}} \underset{4\times 1}{\begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix}}$$

for $x = x_j \quad j = 1, \dots, T$

Suppose for simplicity that $T = m-1$ = # intervals

$$\underset{1\times(m-1)}{S(x)} = \underbrace{\begin{bmatrix} a_1, & b_1, & c_1, & d_1 \\ \vdots & & & \vdots \\ a_{m-1}, & b_{m-1}, & c_{m-1}, & d_{m-1} \end{bmatrix}}_{(m-1)\times 4} \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ x_1 & & x_m \\ x_1^2 & & x_m^2 \\ x_1^3 & & x_m^3 \end{bmatrix}}_{4\times(m-1)}$$

it was 1,

and we'd be good. We need some kind of sparse matrix approach to get the multiplication right.

E.g. suppose we stack, denoting $n := m-1$

$$\underset{1\times 4n}{\left[a_1\ b_1\ c_1\ d_1\ \dots\ a_n, b_n, c_n, d_n\right]} \cdot \underset{4n\times n}{X}$$

where

$$X = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ x_1 & 0 & & \vdots \\ x_1^2 & 0 & & \vdots \\ x_1^3 & 0 & & \vdots \\ 0 & 1 & & \vdots \\ \vdots & x_2 & & 0 \\ \vdots & x_2^2 & & 0\ 1 \\ \vdots & x_2^3 & & 0\ x_n \\ \vdots & 0 & & 0\ x_n^2 \\ 0 & 0 & \cdots & 0\ x_n^3 \end{bmatrix}_{4n\times n}$$

$$\Rightarrow S(x) = \underset{1\times 4}{\text{vec}(\theta)'} \cdot X$$

So given V' and X ... shit... this X isn't invertible.

Wait a sec. This whole thing is like β^{OLS}, where

$1, x_i, x_i^2, x_i^3$ are the 4 regressors for data $i=1,\dots,T$.

and $\underset{4\times 1}{\theta}$ are the β, except that they are "time-varying".

$$\underset{n\times 1}{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}} = \underset{n\times(k+1)}{\begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ 1 & & & & \vdots \\ \vdots & & & & \vdots \\ 1 & x_{1n} & - & - & x_{kn} \end{bmatrix}} \underset{(k+1)\times 1}{\begin{bmatrix} \beta_1 \\ \vdots \\ \beta_{k+1} \end{bmatrix}} + \underset{n\times 1}{\begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}}$$

For me, $k=4$ (incl constant) and so for X to be invertible you need $n=4$.

$$y = X\beta + e$$

What one could do is to write it the other way

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}' = \underset{1\times k}{\begin{bmatrix} \beta_1, & \cdots & \beta_k \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} 1 & & - & - & - & 1 \\ x_{11} & & & & & \\ \vdots & & & & & \\ x_{k-1,1} & & & & x_{k-1,k-1} \end{bmatrix}}$$

which is

$y = RHS'$ which gives us the above.

So btw this was a crucial mistake in the way I "pre-collared" into the Chebyshev polys as regressors.

$$\begin{bmatrix} s_1 \\ \vdots \\ s_T \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ \vdots & & & \vdots \\ 1 & x_T & x_T^2 & x_T^3 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_4 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_T \end{bmatrix}$$

$T \times 1 \qquad T \times 4 \qquad 4 \times 1 \qquad T \times 1$

$\Rightarrow T \times 1$

Now. For the spline β is not 4×1 but it's $4 \times (n-1)$ or $4 \times n$, where n = # of intervals.

Suppose for simplicity that $T = n$, so each row j of X is associated with column j of β.

$$\underset{n \times 1}{\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix}} = \underset{n \times 4}{\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ \vdots & & & \vdots \\ 1 & x_T & x_T^2 & x_T^3 \end{bmatrix}} \underset{4 \times n}{\begin{bmatrix} \beta_1 & \cdots & \beta_{1,n} \\ \vdots & & \vdots \\ \beta_4 & \cdots & \beta_{4,n} \end{bmatrix}}$$

What if we write this as

$$\underset{n \times 1}{s(x)} = \underset{1 \times 4}{\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \end{bmatrix}} \underset{4 \times 1}{\begin{bmatrix} \beta_{1,1} \\ \vdots \\ \beta_{4,1} \end{bmatrix}} + \begin{bmatrix} 1 & x_2 & x_2^2 & x_2^3 \end{bmatrix} \begin{bmatrix} \beta_{1,2} \\ \vdots \\ \beta_{4,2} \end{bmatrix}$$

$+ \ldots$ until n?

Not cool b/c it would give us 1×1.

What we need is that

$$s(x_i) = \begin{bmatrix} 1 & x_i & x_i^2 & x_i^3 \end{bmatrix} \begin{bmatrix} \beta_{1,j} \\ \beta_{2,j} \\ \beta_{3,j} \\ \beta_{4,j} \end{bmatrix}$$

Repeat meeting

6 May 2020

on is identity at one point.

This is before warning.

Want to treat CUSUM - equation as a resid eq.

-> on should be chosen so as to min that resid coming from the CUSUM-test.

Recursive feedback of errors make the sol extremely unstable.

& partor => could save you on order of 20.

Try mex w/ sim_learn LM_clear_given_seg in