Lecture XI

Global Approximation Methods

Gianluca Violante

New York University

Quantitative Macroeconomics

Global function approximation

- Global methods: function interpolation over the entire domain
- First choice: type of approximation
  1. Interpolation I: spectral methods
     - Orthogonal polynomials (e.g., Chebyshev, Lagrange, etc...)
  2. Interpolation II: finite element methods
     - Splines (e.g., B-splines, cubic splines, Schumaker splines)
- Second choice: how to minimize residuals between true function and interpolant
  1. Collocation
  2. Least squares
  3. Galerkin method

Interpolation

- Today, one dimension.
- We want to represent an real-valued function f on $[a, b]$ with a computationally tractable function
- Interpolation is a form of function approximation in which the approximating (interpolant) and the underlying function must agree at a finite number of points, i.e. they must have same value.
- In some cases additional restrictions are imposed on the interpolant. E.g., f' evaluated at a finite number of points may have to agree with that of the underlying function. Other examples include additional constraints imposed on the interpolant such as monotonicity, convexity, or smoothness requirements.
- In general, interpolation is any method taking information at a finite set of points, X, and finds a function which satisfies that information at X.

Interpolation

- We want to approximate a known function $f(x)$. The interpolant $\tilde{f}$ is chosen to be a linear combination of a collection of basis functions $\{\phi_j(x)\}_{j=0}^{n}$ where n is the order/degree of interpolation

- Basis functions: linearly independent functions that span the family of functions chosen for the interpolation (the function space)

  - any function in the space can be obtained as a linear combination of basis functions, just as every vector in a vector space can be obtained as a linear combination of basis vectors.

- In general we are interested in the space of continuous or continuously differentiable functions.

Interpolation

- Thus, for a given family of basis function we have:

$$f(x) \equiv \sum_{j=0}^{n} w_j \phi_j (x)$$

- We have reduced the problem of characterizing an infinite-dimensional object, f, to the problem of determining the n weights (or coefficients) $\{w_j\}$.

- There is some arbitrariness in interpolation: there are arbitrarily many functions passing through a finite number of points.

- Choices: spectral or finite element methods depending whether the basis functions are nonzero over the entire domain of the true function (except possibly at a finite number of points) or nonzero only on a subinterval of the domain.

Weighted residual methods: collocation

- Since we have $n+1$ undetermined coefficients, we need $n+1$ conditions, at least, to determine these coefficients.

- The most basic conditions imposed are that $\tilde{f}$ interpolates or matches the value of the original function f at $n+1$ selected interpolation nodes $\{x_i\}_{i=0}^{n}$ or:

$$\tilde{f}(x_i) \equiv \sum_{j=0}^{n} w_j \phi_j(x_i) = f(x_i), \quad i = 0, ..., n$$

- Let Φ be the $(n+1 \times n+1)$ matrix with element $\phi_{ij} = \phi_j(x_i)$, w be the $(n+1 \times 1)$ vector of weights with element w_j and y be the $(n+1 \times 1)$ vector of $f(x_i)$ or in matrix notation:

$$\Phi w = y$$

  Note that Φ the interpolation matrix should be nonsingular, so that we can obtain $w = \Phi^{-1} y$.

Weighted residual methods: collocation

- One can also choose to match the value of the function at $n_1 + 1$ points and its derivative at $n_2 + 1$ points. Thus, we solve a system of $(n_1 + n_2 + 2) = n + 1$ linear equations and same number of unknowns defined by:

$$\begin{aligned} \sum_{j=0}^{n} w_j \phi_j (x_i) &= f(x_i), i = 0, ..., n_1 \\ \sum_{j=0}^{n} w_j \phi_j' (x_i) &= f'(x_i), i = 0, ..., n_2 \end{aligned}$$

- In these cases, we set the residuals on the nodes exactly to zero, i.e. we solve:

$$r = y - \Phi w = 0$$

Weighted residual methods: LS

- If we have more interpolation nodes than basis functions, then we have a “curve fitting” problem: we can define residuals

$$r = y - \Phi w$$

- By minimizing the SSR, we get:

$$w = \left(\Phi' \Phi\right)^{-1} \Phi' y$$

  which equals $\Phi^{-1} y$ when Φ is squared and invertible since $\left(\Phi' \Phi\right)^{-1} = \Phi^{-1} \left(\Phi'\right)^{-1}$

- Basically, we set to zero the weighted residuals, with weight $\frac{\partial r}{\partial w}$:

$$\Phi' r = \Phi'(y - \Phi w) = 0$$

Weighted residual methods: Galerkin

- Define residuals:

$$r(x) = f(x) - \sum_{j=0}^{n} w_j \phi_j (x)$$

- Solve the $n+1$ equations:

$$\int_a^b r(x)\phi_j(x)dx = 0, \quad \text{for} j = 0, 1, ..., n$$

  into $n+1$ unknowns $\{w_j\}$

- The Galerkin choice for the weights are the basis functions used to represent $\tilde{f}$

- Idea: a continuous function $(r(x))$ is zero if it is orthogonal to each member of a complete set of functions

Spectral methods

- Basically, it means that we use polynomial basis. Polynomials are generically nonzero. The motivation for using polynomials come from:

- Weierstrass Theorem: If $C[a,b]$ is the set of all continuous function on $[a,b]$ then for all $f \in C[a,b]$ and $\varepsilon > 0$ there exists a polynomial q for which

$$\sup_{x \in [a,b]} |f(x) - q(x))| < \varepsilon$$

- I.e., there exists a polynomial that approximates any continuous function over a compact domain arbitrarily well in a uniform sense (the approximation error as measured by the sup norm is arbitrarily small.)

Spectral methods

- How do we find a good approximating polynomial and the degree of interpolation needed to reach a desired level of accuracy?

- Another theorem comes handy and tells us some properties of such polynomial. Define:

$$\rho_n(f) = \inf ||f - q_n||_\infty$$

  the smallest of the all the supnorm distances between f and the polynomial q of degree n. Then:

- Equioscillation theorem: If $C[a,b]$ is the set of all continuous function on $[a,b]$ and $f \in C[a,b]$, then there is a unique polynomial of degree n, q_n^* that achieves $\rho_n(f)$. In addition, the polynomial q_n^* is also the unique polynomial for which there are at least $n+2$ points $a \leq x_0 < x_1 < ...x_{n+1} \leq b$ such that for $m = 1$ or $m = -1$

$$f(x_j) - q_n^*(x_j) = m(-1)^j \rho_n(f) \quad , j = 0, ..., n+1$$

Equioscillation theorem

- This last equation is called the equioscillation property.
- It means that, if we have the best, say, cubic polynomial approximation of f, the maximum error should be achieved at least five times and that the sign of the error should alternate between these points.
- This is a useful theorem because it tells us what our errors should look like when we are trying to find the best approximation.
- If we plot the error and have a very different shape, we know that it is theoretically possible to do better. A general rule of thumb is the closer our error looks to this shape the better the overall approximation.

Jackson's theorem

- Finally, we have:

- Jackson's theorem: For all k, if $f \in C^k[a,b]$, then for $n \geq k$

$$\rho_n(f) \leq \frac{(n-k)!}{n!}\left(\frac{\pi}{2}\right)^k \left(\frac{b-a}{2}\right)^k ||f^{(k)}||_\infty$$

- It establishes an upper bound for $\rho_n(f)$

- Under the best polynomial approximation (i) the higher the order of the polynomial and (ii) the smoother the function the better the approximation.

Monomial basis

- The most naive procedure would be to use as basis functions the monomials, i.e:

$$\phi_j(x) = x^j, \quad j = 0, 1, ..., n$$

with interpolating polynomials of the form

$$\tilde{f}(x) \equiv p_n(x) \equiv w_0 + w_1 x + w_2 x^2 + ... + w_n x^n$$

and a linear system of $n+1$ equations to solve of the form:

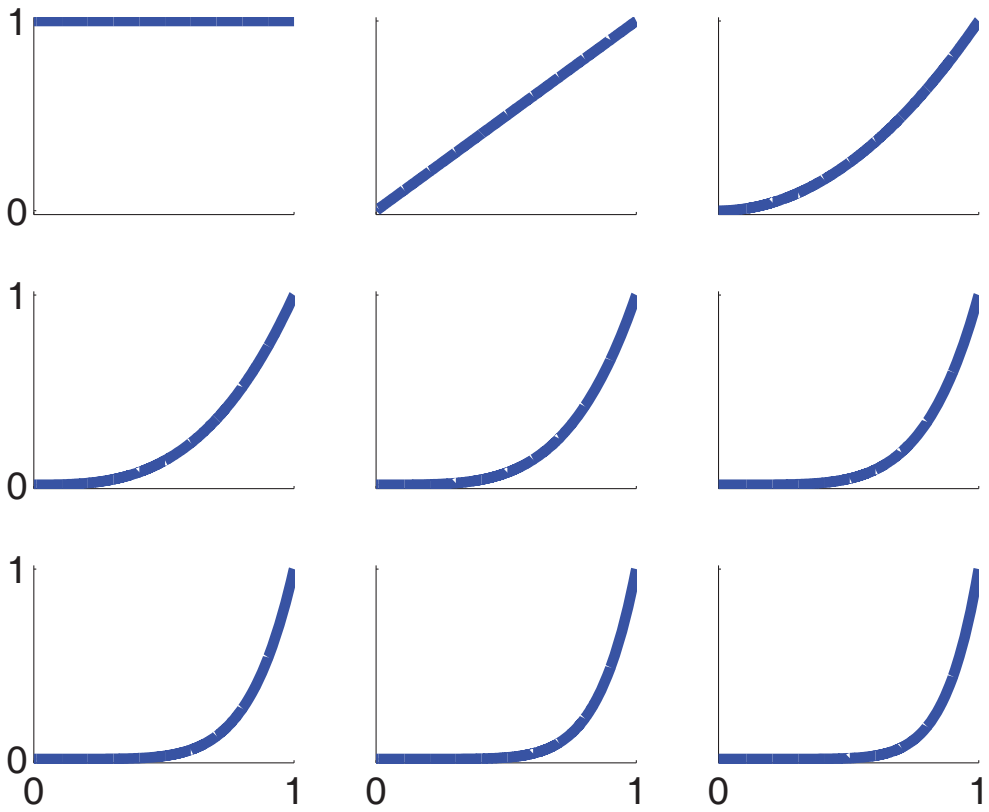
$$\begin{bmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Monomial basis

- They span space of continuous functions, so in theory they work
- Matrix associated with this linear system is called a Vandermonde matrix, i.e. a matrix with the terms of a geometric progression in each row. It can be shown that it's determinant is always non-zero. Thus theoretically, the system is well-defined.
- Unfortunately, in practice, the Vandermonde matrix is known for being ill-conditioned, especially for high-degree polynomials. This is because, as the degree increases, the functions become progressively less distinguishable making the columns of the Vandermonde matrix nearly linearly dependent.
- In fact, if we define $V_{ij} = x_{i-1}^{j-1}$ as the generic element of the matrix, then

$$\det(V) = \Pi_{1 \leq i \leq j \leq n+1} (x_{i-1} - x_{j-1})$$

Monomials in [0,1]



Orthogonal polynomials

- The reason why the monomials are not a good basis for the space of continuous functions is that they are far from being orthogonal.
- Orthogonality is a good property for polynomial basis because (i) they do a good job of sampling the interval (thus the problem is unlikely to be poorly conditioned) and (ii) orthogonal polynomials can be obtained recursively, which is very convenient (Theorem 6.3.1 in Judd)
- Weighting functions. A weighting function, w on $[a, b]$ is a positive function almost everywhere, such that $\int_a^b w(u)\,du < \infty$
- Inner Product. Consider two functions, f and g defined at least on $[a, b]$. The inner product with respect to the weighting function w is given by $\langle f, g\rangle = \int_a^b f(u)g(u)w(u)du$.

Orthogonal polynomials

- Orthogonal Polynomials. The collection of polynomials $\{\phi_j\}_{j=0}^{n}$ is orthogonal on $[a, b]$ with respect to the weighting function w if and only if $\langle \phi_i, \phi_k \rangle = 0$ for all $i \neq k$ and they are orthonormal if $\langle \phi_j, \phi_j \rangle = 1$ for all j.

- Thus a basis is orthonormal if

$$\int_a^b \phi_i(u)\,\phi_k(u)\,w(u)\,du = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$$

- Common families of orthogonal polynomials: Chebyshev, Lagendre, Laguerre, Hermite

- Easy to integrate through quadrature!

Chebyshev polynomials

- They are trigonometric polynomials defined by

$$\begin{aligned} T_k &: [-1,1] \to [-1,1] \\ T_k &= \cos\left(n \cos^{-1}(x)\right) \end{aligned}$$

  and can be simply constructed recursively as

$$\begin{aligned} T_0(x) &= \cos(0) = 1, \\ T_1(x) &= \cos\left(\cos^{-1}(x)\right) = x, \\ T_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x), \quad \text{for } k \geq 1. \end{aligned}$$

- Therefore:

$$\begin{aligned} T_2(x) &= 2xT_1(x) - T_0(x) = 2x^2 - 1 \\ T_3(x) &= 2xT_2(x) - T_1(x) = 4x^3 - 3x \\ T_4(x) &= 2xT_3(x) - T_2(x) = 8x^4 - 8x^2 + 1 \end{aligned}$$

Chebyshev polynomials

- $T_n(x)$ has n distinct roots in $[-1, 1]$ with expression:

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

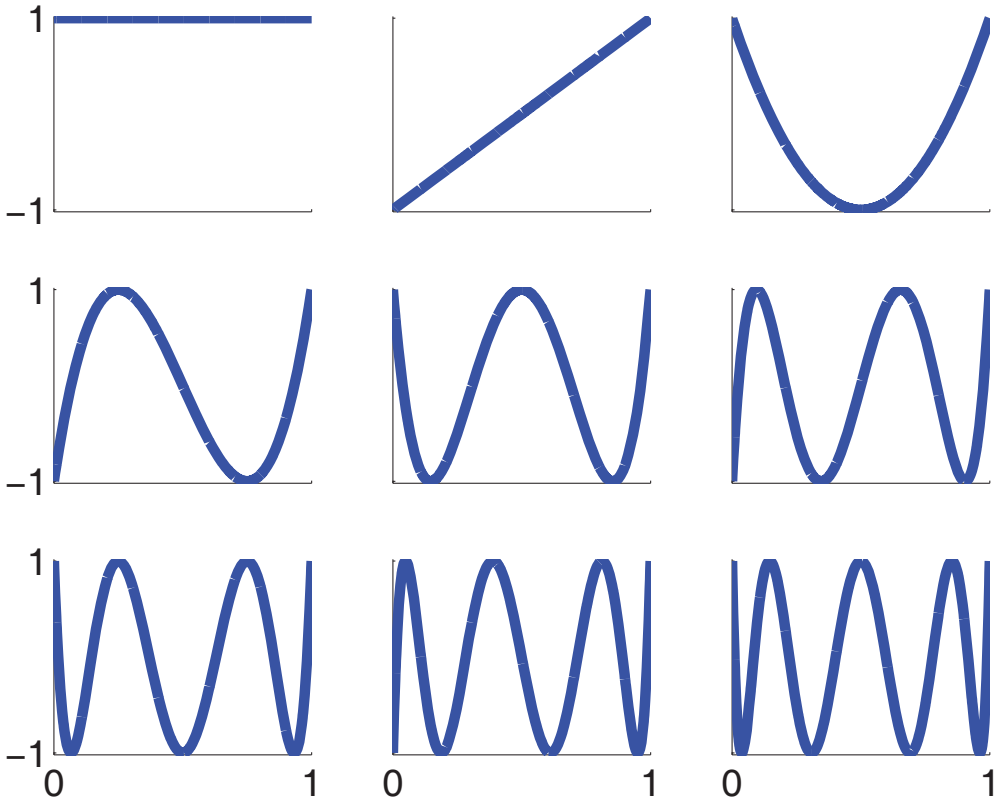
  and it can be shown that the zeros are clustered quadratically towards -1 and $+1$

- $T_n(x)$ has $n+1$ distinct extrema with value of either -1 or 1. Check for $k = 2$ for example.

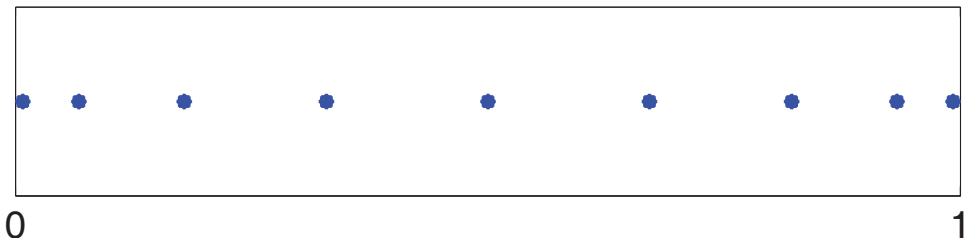
- They are orthogonal with respect to the weighting function

$$w(x) = \left(1 - x^2\right)^{-1/2}.$$

Chebyshev basis functions in [0,1]



Chebyshev zeros on [0,1]



- One can show that the interpolation nodes that minimize the error of the Chebyshev interpolation are the zeros of the Chebyshev polynomials themselves.
- More Cheybshev points near the endpoints of the interval where we see larger errors when using an evenly-spaced grid.

Chebyshev polynomials: quality of appx

- Rivlin Theorem: If $C_n(x)$ is the nth degree Chebyshev approximation of $f \in C^1[-1,1]$ then:

$$||f - C_n||_\infty \leq \left(4 + \frac{4}{\pi^2}\ln n\right)\rho_n(f)$$

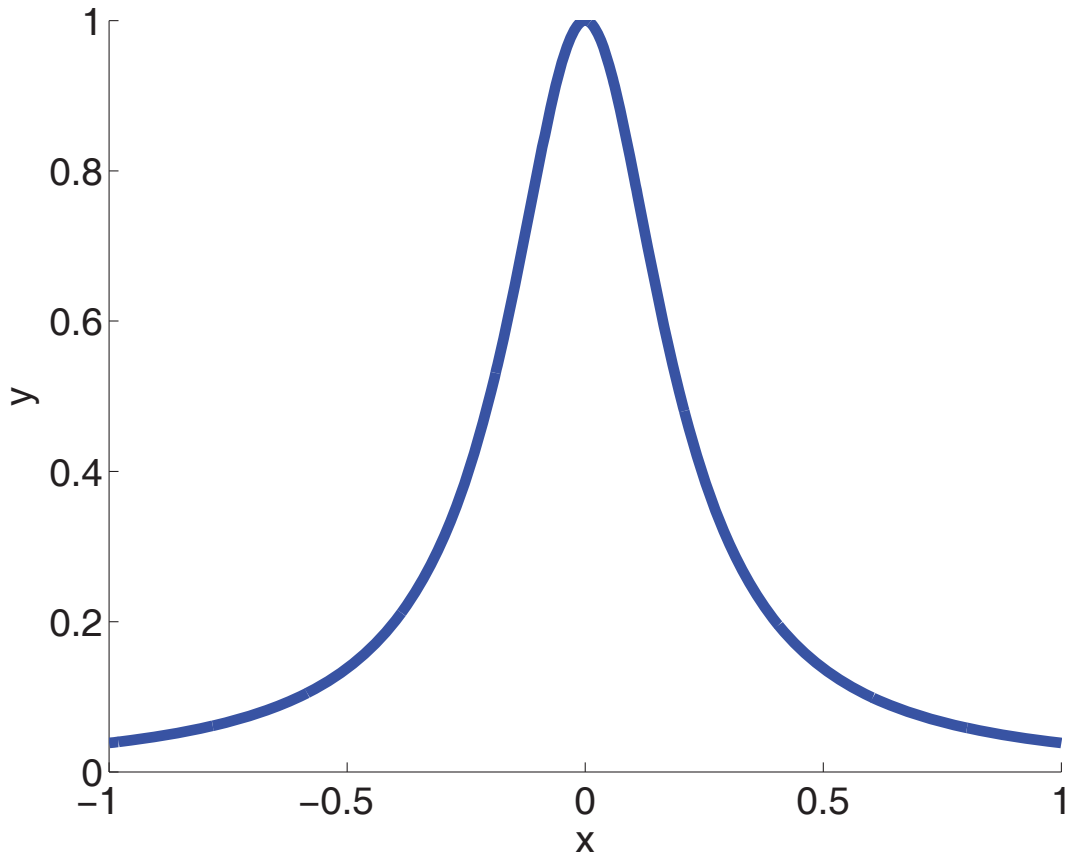
  and therefore the Chebyshev approximation is not much worse than the best polynomial appx of the same order in L^∞.

- Worried about the $\log n$ term? You should not be, because remember that by Jackson theorem:

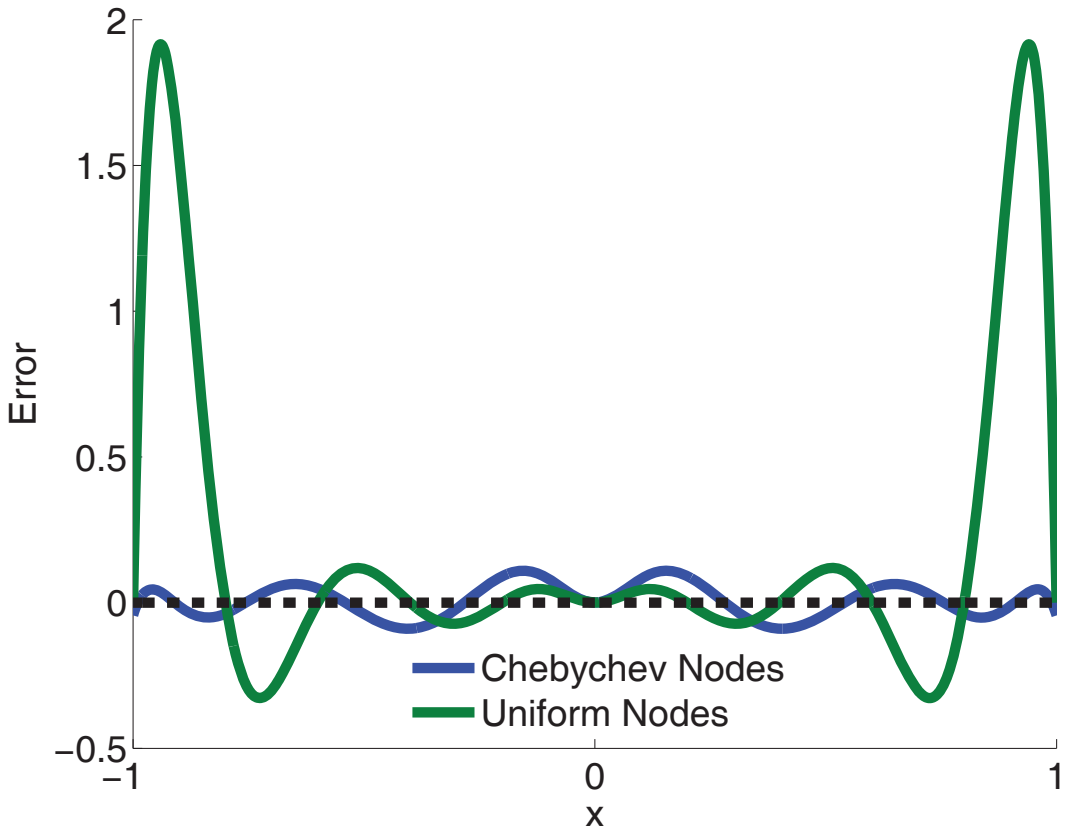
$$\left(4 + \frac{4}{\pi^2}\ln n\right)\rho_n(f) \leq \left(4 + \frac{4}{\pi^2}\ln n\right)\frac{(n-k)!}{n!}\left(\frac{\pi}{2}\right)^k\left(\frac{b-a}{2}\right)^k |f^{(k)}||_\infty$$

  and thus as n grows the error goes to zero uniformly under the Chebyshev approximation

Runge function on [-1,1]: $f(x) = 1/(1 + 25x^2)$



Approx. errors with Chebyshev poly (11th order)



Chebyshev polynomials

- The Chebyshev nodes are always internal thus these optimal nodes exclude the endpoints.
- In practice, we need to have a grid point at endpoint it because we expect a mass of agents on the constraint for example.
- Then if we want the first gridpoint to be -1

$$-1 = -\cos\left(\frac{\pi}{2n}\right)\kappa \rightarrow \kappa = \frac{1}{\cos\left(\frac{\pi}{2n}\right)} = \sec\left(\frac{\pi}{2n}\right)$$

- Then, one should use the expanded Chebyshev grid:

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

Chebyshev polynomials

- Also note that even though the Chebyshev polynomials are defined over the interval $[-1, 1]$ this interval can be easily converted to $[a, b]$ by linearly transforming the data.

- If $x \in [-1, 1]$ and $z \in [a, b]$ then the nodes become:

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

- Theorem 8.2.3 in Heer and Maussner: If $f \in C^k [-1, 1]$ has a Chebyshev representation $f = \sum_{j=0}^{\infty} w_j T_j (x)$ then there is a constant c s.t.:

$$|w_j| \leq \frac{c}{j^k}, j \geq 1$$

Coefficients of Chebyshev poly rapidly decreasing in the order of polynomials and we can confidently ignore higher order basis

How to use Chebyshev poly in practice

- Here comes handy the discrete orthogonality property.
- Let $\{x_k\}_{k=0}^{m}$ be the first $m+1$ roots of the Chebyshev polynomial. Then for all $i, j < m$:

$$\sum_{k=0}^{m} T_i(x_k)\, T_j(x_k) = \begin{cases} 0 & \text{if } i \neq j \\ (m+1)/2 & \text{if } i = j \neq 0 \\ (m+1) & \text{if } i = j = 0 \end{cases}$$

- This property allows us to derive simple expressions for the interpolation weights when the grid points used are the Cheybshev nodes.

How to use Chebyshev poly in practice

- Suppose we want to approximate a function f on $[-1, 1]$ with an n^{th} order Chebyshev polynomial.
- Compute $m + 1 \geq n + 1$ Chebyshev interpolation nodes indexed by $k = 0, ..., m$
- Evaluate the function:

$$y_k = f(x_k)$$

- The problem is to find the $(n + 1)$ weights $\{w_j\}_{j=0}^{n}$ such that:

$$\sum_{j=0}^{n} w_j T_j(x_k) = y_k, \quad k = 0, ..., m \geq n$$

i.e., collocation or LS (depending whether $m = n$ or $m > n$)

How to use Chebyshev poly in practice

- First pick an $i \in \{0, 1, ..., n\}$ and multiply both sides by $T_i(x_k)$ and then sum across k

$$\begin{aligned}
\sum_{k=0}^{m}\sum_{j=0}^{n} w_j T_i(x_k) T_j(x_k) &= \sum_{k=1}^{m} T_i(x_k) y_k \\
\sum_{j=0}^{n} w_j \sum_{k=0}^{m} T_i(x_k) T_j(x_k) &= \sum_{k=1}^{m} T_i(x_k) y_k
\end{aligned}$$

From the discrete orthogonality the terms in $i \neq j$ are zero, so

$$\begin{aligned}
w_j \sum_{k=0}^{m} T_j(x_k) T_j(x_k) &= \sum_{k=1}^{m} T_i(x_k) y_k \\
w_j &= \frac{\sum_{k=0}^{m} T_j(x_k) y_k}{\sum_{k=0}^{m} T_j(x_k) T_j(x_k)}
\end{aligned}$$

How to use Chebyshev poly in practice

- Note that the expression above for w_j is that of an OLS estimator that solves

$$\min_{\{w_j\}_{j=0}^{n}} \left\{ \sum_{k=0}^{m} \left[f(x_k) - \sum_{j=0}^{n} w_j T_j(x_k) \right]^2 \right\}$$

  i.e. the vector of $\{w_j\}$ minimizes the sum of squared residuals between the true function and the approximated function.

How to use Chebyshev poly in practice

- The vector of $\{w_j\}$ can be obtained in one step as:

$$\begin{bmatrix} T_0(x_0) & T_1(x_0) & \dots & T_n(x_0) \\ T_0(x_1) & T_1(x_1) & \dots & T_1(x_1) \\ & \vdots & \ddots & \vdots \\ T_0(x_m) & T_1(x_m) & \dots & T_n(x_m) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_m \end{bmatrix}$$

  where the $\mathbf{T}$ matrix is $(m+1 \times n+1)$ and

$$\mathbf{w} = \left(\mathbf{T}'\mathbf{T}\right)^{-1}\mathbf{T}'\mathbf{y}$$

- Finally, the polynomial approximation of f is

$$f(x) = \sum_{j=0}^{n} w_j T_j(x)$$

Finite element methods

- Use local basis, instead of global basis.
- These are called splines
- An order n spline consists of a series of n^{th} order polynomial segments spliced together so as to preserve continuity of derivatives of order $n-1$ or less.
- The points at which the pieces are spliced together are called knots or breakpoints.
- By convention the first and last knots are the two extremes of the function domain.

Splines

- An order n spline with p knots is characterized by $(p-1)(n+1)$ parameters.
- The simplest way to proceed (Judd calls it the “kindergarten procedure of connecting the dots”) is to use linear B-splines, where B stands for basic.
- Suppose we have a grid with knots $x_0 < x_1 < ...x_k < ... < x_n$ and we want to approximate a function $f(x)$.
- B^0 splines are defined as

$$B_k^0(x) = \begin{cases} 1 & \text{if } x_k \leq x < x_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

so they are right-continuous step functions. They implement step-function interpolations... not recommended

Linear B-splines

- B^1 splines or piecewise linear splines implement piece-wise linear interpolation

$$B_k^1(x) = \begin{cases} \frac{x - x_{k-1}}{x_k - x_{k-1}} & \text{if } x_{k-1} \leq x < x_k \\ \frac{x_{k+1} - x}{x_{k+1} - x_k} & \text{if } x_k \leq x < x_{k+1} \\ 0 & \text{elsewhere} \end{cases}$$

  and it looks like a “tent-functions” with peak at x_k equal to one, thus the weight of the interpolant must be $f(x_k)$

- The interpolant is therefore defined as:

$$\hat{f}(x) = \sum_{k=0}^{n} f(x_k) B_k^1(x)$$

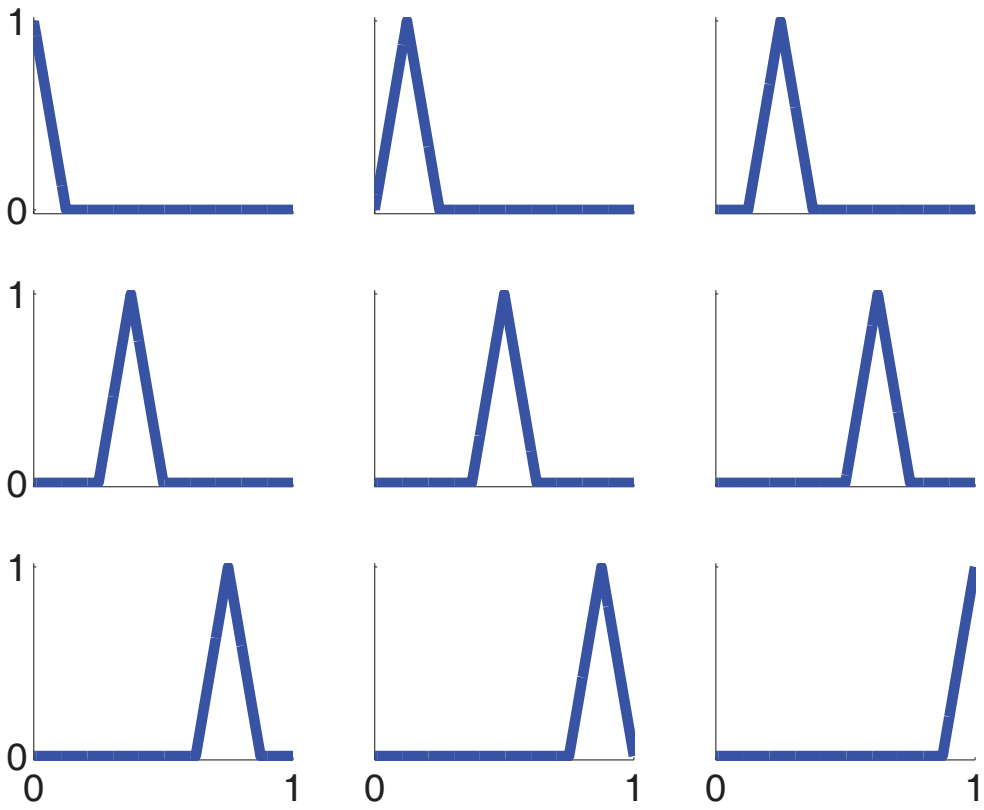
Linear B-splines

- It is easy to see that for any given point x on an interval $[x_k, x_{k+1}]$ there are only two nonzero basis at x, $B_k(x)$ and $B_{k+1}(x)$.
- To find k given x you use a locating algorithm that essentially proceeds by bisection.
- Thus

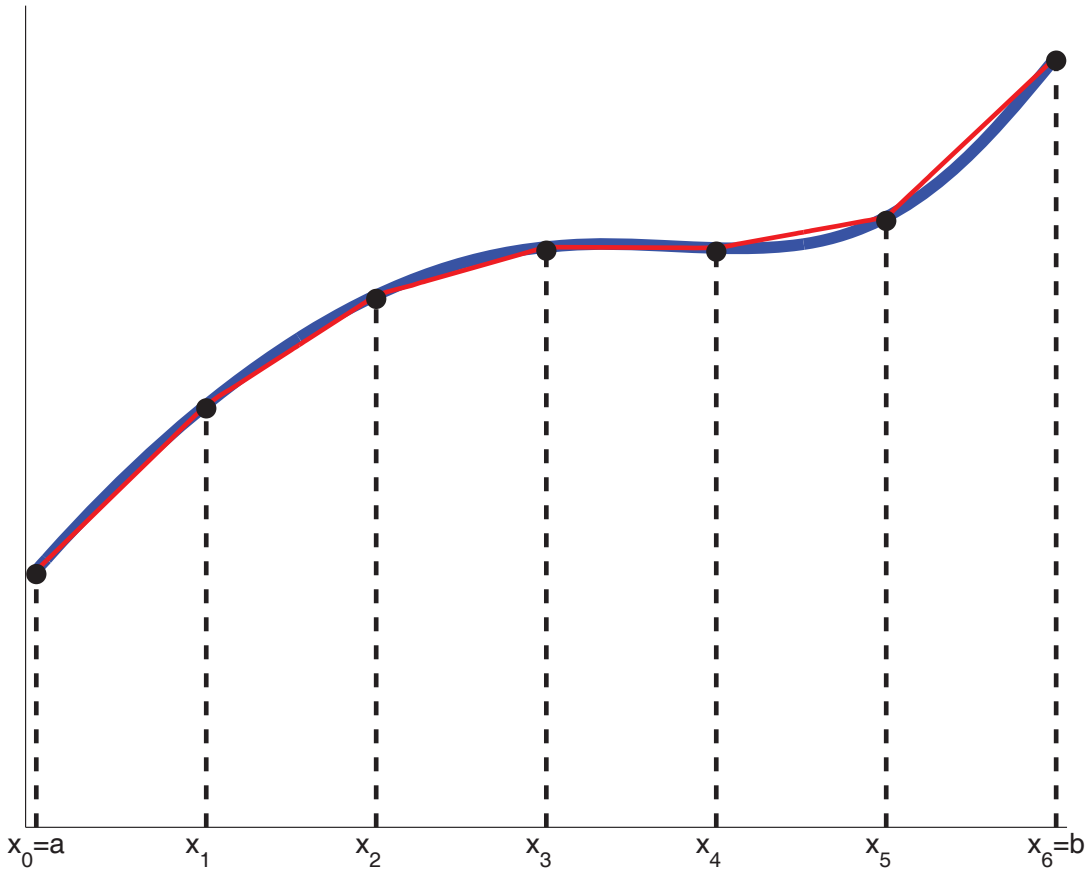
$$\begin{aligned}
\hat{f}(x) &= f(x_k)\frac{x_{k+1}-x}{x_{k+1}-x_k} + f(x_{k+1})\frac{x-x_k}{x_{k+1}-x_k} \\
&= f(x_k)\frac{x_{k+1}-x-x_k+x_k}{x_{k+1}-x_k} + f(x_{k+1})\frac{x-x_k}{x_{k+1}-x_k} \\
&= f(x_k) + [f(x_{k+1})-f(x_k)]\frac{x-x_k}{x_{k+1}-x_k}
\end{aligned}$$

that is a familiar formula for linear interpolation of decision rules (see later...)

Linear spline Basis Functions on [0, 1].



Linear Spline approximation (7 knots)



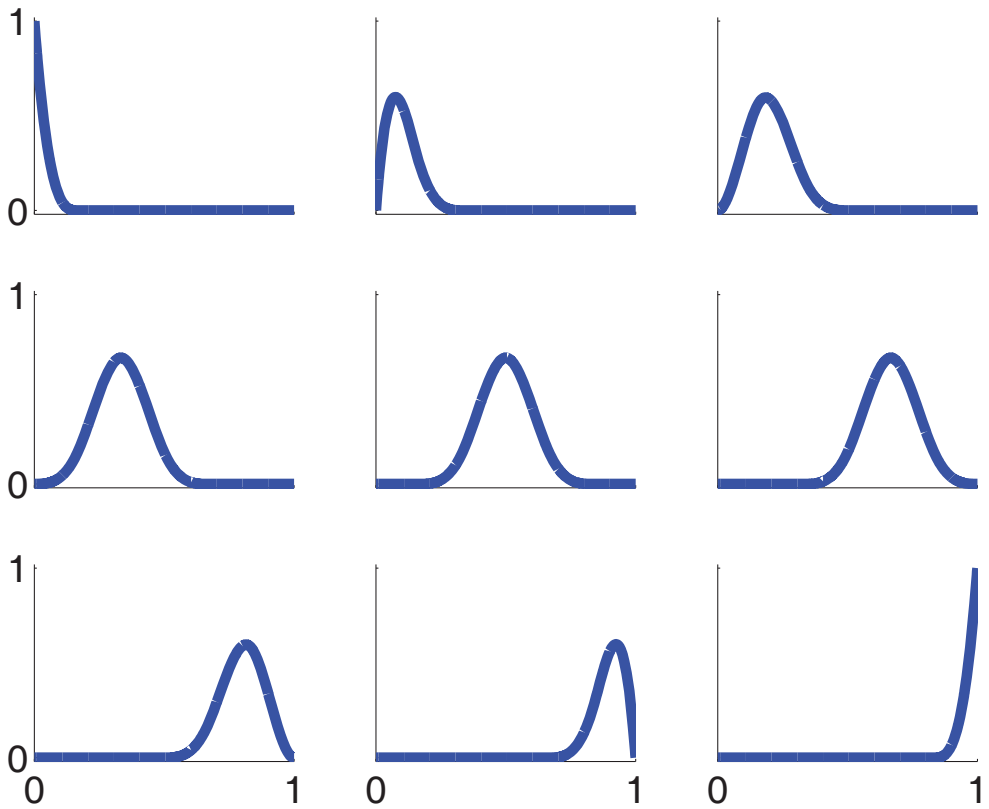
Pros and cons of linear splines

- Pros:
  - Preserves monotonicity and concavity of f.
  - We can exploit information about f clustering points more closely together in areas of high curvature or areas where we know a kink exists in order to increase our accuracy.
  - We can capture binding inequality constraints very well.
- Cons:
  - The approximated function is not differentiable at the knots (they become kinks).
  - $\tilde{f}'' = 0$ where it exists (it does not exist at the knots). Function not smooth.

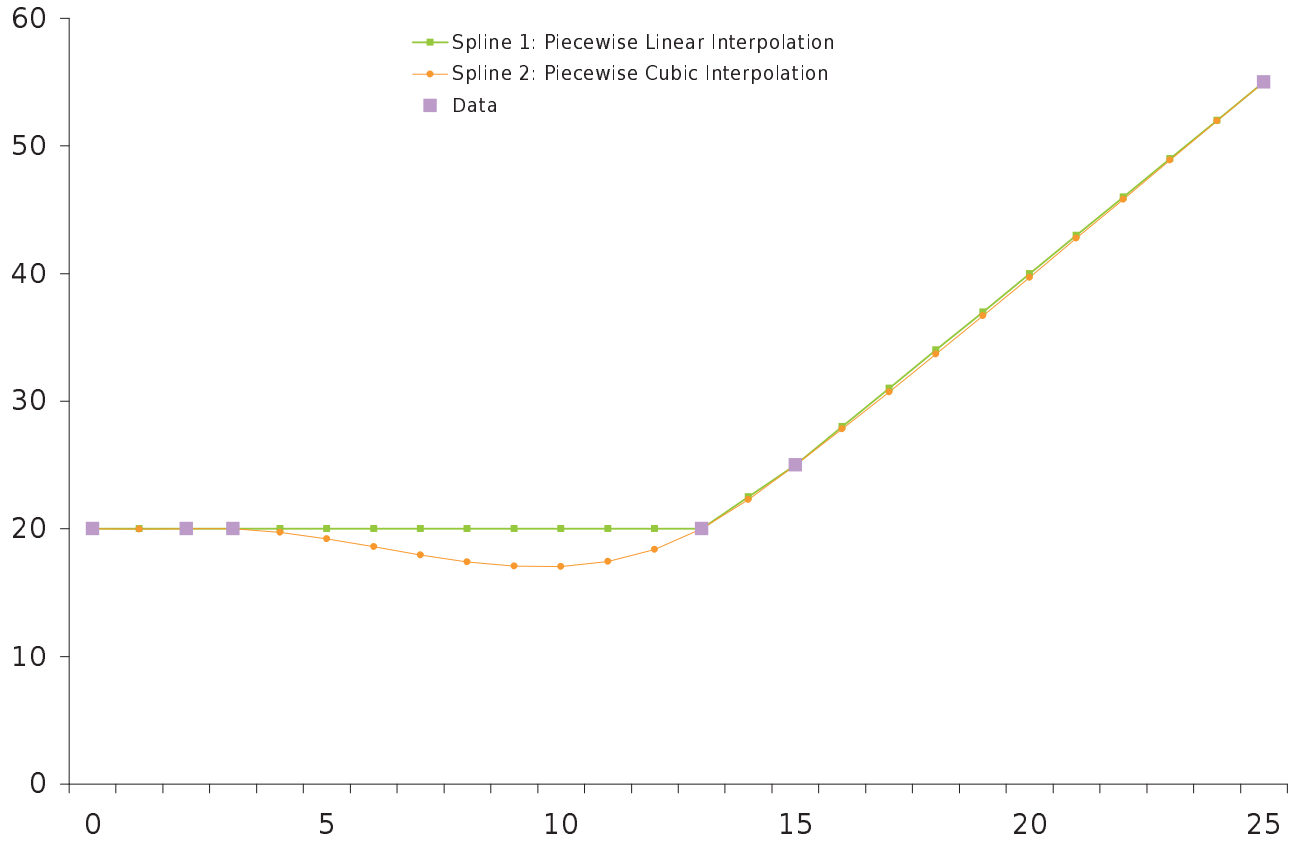
Cubic splines

- Cubic spline $(n = 3)$, has $4(p-1)$ coefficients to be determined
  - $\hat{f}(x_i) = y_i$ (p conditions)
  - Continuity of $\hat{f}$ at interior points (p-2 conditions)
  - Continuity of $\hat{f}'$ at interior points (p-2 conditions)
  - Continuity of $\hat{f}''$ at interior points (p-2 conditions)
  - $\hat{f}''(x_1) = \hat{f}''(x_p) = 0$ (2 conditions)
- These last two can be modified to something else depending on what is suitable in the particular case at hand, they are ad-hoc and they can be more than 2.

Cubic spline Basis Functions on [0,1].



Cubic spline Basis Functions on [0,1].



Pros and cons of cubic splines

- Pros:
  - Easy to compute: interpolant matrix very sparse, easy to invert.
  - Smooth approximation
- Cons:
  - It may not be able to handle constraints well
  - It does not preserve monotonicity and concavity of f

Schumaker splines

- So far: linear interpolation preserves shape (monotonicity and concavity), but not differentiability. Cubic splines preserves differentiability, but not shape.
- Schumaker quadratic splines preserve both (Schumaker, 1983, SIAM Journal on Numerical Analysis).
- By shape we mean that in those intervals where the data is monotonically increasing or decreasing, the spline has the same property. Similarly for convexity or concavity.
- Idea: add knots to a subinterval by means of an algorithm

Application when function is unknown

- In practice, function f is unknown: value function or decision rule
- How do we adapt this method? Easy.
- Represent a decision rule by interpolation, e.g.

$$x' = g(x) = \sum_{k=0}^{n} \phi_k T_k(x)$$

- Build a residual function from Euler (or Bellman) equation

$$\mathcal{R}(x, \phi) \equiv \mathcal{R}(x, g(x), g(g(x)))$$

- Choose a weighted residual method to solve for $\{\phi\}$ s.t.:

$$\int_{x_{\min}}^{x_{\max}} \omega(x)\mathcal{R}(x, \phi)dx = 0$$

Solving the income fluctuation problem

$$\begin{aligned} V(a, y_j) &= \max_{\{c, a'\}} u(c) + \beta \sum_{y_i \in Y} \pi(y_i|y_j) V(a', y_i) \\ & \quad s.t. \\ c + a' &\leq Ra + y_j \\ a' &\geq \underline{a} \end{aligned}$$

The Euler equation, once we substitute in the budget constraint, reads:

$$u_c(Ra + y_j - a') - \beta R \sum_{y_i \in Y} \pi(y_i|y_j) u_c(Ra' + y_i - a'') \geq 0.$$

where the strict inequality holds when the constraint binds.

Solving the income fluctuation problem

1. Choose order n of the Chebyshev polynomial for approximation

2. Recall the expression for Chebyshev nodes on $[-1, 1]$ as

$$z_i = \cos\left(\frac{2i+1}{2\,(m+1)}\pi\right) \qquad i = 0, ..., m \geq n$$

3. Set the upper bound $\bar{a}$. Define the grid on $[\underline{a}, \bar{a}]$ as:

$$a_i = \frac{\bar{a}+\underline{a}}{2} - \sec\left(\frac{\pi}{2\,(m+1)}\right) \cdot z_i \cdot \left(\frac{\bar{a}-\underline{a}}{2}\right) \qquad i = 0, ..., m$$

   Note that we have no control over point allocation on $[a, \bar{a}]$. Note that for $i = 0, a_i = \underline{a}$ and for $i = m, a_i \simeq \bar{a}$

4. Let $t\,(a_i)$ be the function that converts a value a_i on $[a, \bar{a}]$ back to a value z_i on $[-1, 1]$

Solving the income fluctuation problem

1. Express the saving decision rule as:

$$a' = g\left(a, \phi_j\right) = \sum_{k=0}^{n} \phi_{jk} T_k\left(t\left(a\right)\right)$$

   where T_k are Chebyshev basis of order $k = 0, 1, ..., n$.

2. There is one decision rule for each value of the shock $j = 1, ..., J$, i.e., the Chebyshev coefficients depend on the value of y_j

3. Define the residual function from the Euler equation

$$\mathcal{R}_j\left(a, \phi_j\right) \equiv u_c\left(Ra + y_j - \sum_{k=0}^{n} \phi_{jk} T_k\left(t\left(a\right)\right)\right)$$

$$-\beta R \sum_{y_i \in Y} \pi\left(y_i | y_j\right) u_c\left(R \sum_{k=0}^{n} \phi_{jk} T_k\left(t\left(a\right)\right) + y_i - \sum_{k=0}^{n} \phi_{ik} T_k\left(t\left(\sum_{k=0}^{n} \phi_{jk} T_k\left(t\left(a\right)\right)\right)\right)\right)$$

Solving the income fluctuation problem

4. Choose your favorite weighted residual method. Set:

$$\int_{\underline{a}}^{\bar{a}} \omega_j(a)\,\mathcal{R}_j(a,\phi_j) = 0 \quad j = 1,2,...,J$$

4.a Collocation: Set $m = n$ and residuals equal to zero on the $n+1$ grid points. Equivalent to choosing Dirac as weighting function:

$$\omega_j(a) = \delta(a - a_i) = \begin{cases} 1 & \text{if } a = a_i \\ 0 & \text{otherwise} \end{cases}$$

i.e., appx. exact on the nodes but nothing imposed off nodes.

In practice, you solve a $(n+1)\,J$ eqns into $(n+1)\,J$ unknowns ϕ:

$$\mathcal{R}_j(a_i,\phi_j) = 0,\ i = 0,1,...,n \text{ and } j = 1,2,...,J$$

by using a multidimensional rootfinding algorithm

Solving the income fluctuation problem

4.2 Galerkin method:

$$\int_{\underline{a}}^{\bar{a}} \frac{T_k(a)}{\sqrt{1-t(a_i)^2}} \mathcal{R}_j(a, \phi_j)\, da = 0 \quad \text{for all } k = 0, 1, ..., n \text{ and } j = 1, 2, ..., J$$

To solve the integral, we use quadrature methods. Fix $m+1$ quadrature points.

Use nodes and weights from Chebyshev quadrature to integrate and rewrite the above equation as:

$$\sum_{i=0}^{m} \left(T_k(t(a_i))\right) \mathcal{R}_j(a_i, \phi_j) = 0 \quad \text{for all } j, k$$

and note that, for each j you can rewrite it in matrix form.

Solving the income fluctuation problem

Define:

$$T\left(t(a)\right)=\begin{bmatrix} T_0\left(t\left(a_0\right)\right) & T_0\left(t\left(a_1\right)\right) & ... & T_0\left(t\left(a_m\right)\right) \\ T_1\left(t\left(a_0\right)\right) & T_1\left(t\left(a_1\right)\right) & \vdots & T_1\left(t\left(a_m\right)\right) \\ \vdots & \vdots & \vdots & \vdots \\ T_n\left(t\left(a_0\right)\right) & T_n\left(t\left(a_1\right)\right) & & T_n\left(t\left(a_m\right)\right) \end{bmatrix} \text{ and } \mathcal{R}_j\left(a,\phi_j\right)=\begin{bmatrix} R_j\left(a_0,\phi_j\right) \\ R_j\left(a_1,\phi_j\right) \\ \vdots \\ R_j\left(a_m,\phi_j\right) \end{bmatrix}$$

and write the system of $n+1$ equations as

$$T\left(t\left(a\right)\right)\mathcal{R}_j\left(a,\phi_j\right)=0 \quad \text{for } j=1,2,...,J$$

Once again, use a quasi-Newton method to solve for the vector of Chebyshev coefficients. You can do it separately for each j.

For each j you have a system of $n+1$ equations.

How do you check the constraint is binding?

- Tell the nonlinear solver that looks for the coefficients to find a vector ϕ_j such that:

$$a'(a_i, y_j) = \sum_{k=0}^{n} \phi_{jk} T_k(t(a_i)) > 0 \quad \text{for all } i, j$$

- You can even impose monotonicity. For all i, j:

$$a'(a_i, y_j) \quad > \quad a'(a_{i-1}, y_j) \rightarrow \sum_{k=0}^{n} \phi_{jk} \left[T_k(t(a_i)) - T_k(t(a_{i-1})) \right] > 0$$

- E.g., in Matlab the function `fmincon` allows you to do that

Policy function iteration with linear interpolation

1. Construct a grid on the asset space $\{a_0, a_2, ..., a_m\}$ with $a_0 = \underline{a} = 0$.

2. Guess an initial vector of decision rules for a'' on the grid points, call it $\hat{a}_0(a_i, y_j)$.

3. For each point (a_i, y_j) on the grid, check whether the borrowing constraint binds. I.e. check whether:

$$u_c(Ra_i + y_j - a_0) - \beta R \sum_{y' \in Y} \pi(y'|y_j)\, u_c(Ra_0 + y' - \hat{a}_0(a_0, y')) > 0.$$

4. If this inequality holds, the borrowing constraint binds. Then, set $\hat{a}_0(a_i, y_j) = a_0$ and repeat this check for the next grid point. If the equation instead holds with the $<$ inequality, we have an interior solution (it is optimal to save for the household) and we proceed to the next step.

Policy function iteration with linear interpolation

5. For each point (a_i, y_j) on the grid, use a nonlinear equation solver to find the solution a^* of the nonlinear equation

$$u_c\left(Ra_i + y_j - a^*\right) - \beta R \sum_{y' \in Y} \pi\left(y'|y_j\right) u_c\left(Ra^* + y' - \hat{a}_0\left(a^*, y'\right)\right) = 0$$

   (a) Need to evaluate the function $\hat{a}_0\left(a, y'\right)$ outside grid points: assume it is piecewise linear.

   (b) Every time the solver calls an a^* which lies between grid points, do as follows. First, find the pair of adjacent grid points $\{a_i, a_{i+1}\}$ such that $a_i < a^* < a_{i+1}$, and then compute

$$\hat{a}_0\left(a^*, y'\right) = \hat{a}_0\left(a_i, y'\right) + \left(a^* - a_i\right)\left(\frac{\hat{a}_0\left(a_{i+1}, y'\right) - \hat{a}_0\left(a_i, y'\right)}{a_{i+1} - a_i}\right)$$

   (c) If the solution of the nonlinear equation is a^*, then set $a_0'\left(a_i, y_j\right) = a^*$ and iterate on the next grid point.

Policy function iteration with linear interpolation

6. Check convergence by comparing $a_0'(a_i, y_j) - \hat{a}_0(a_i, y_j)$ through some pre-specified norm. For example, declare convergence at iteration n when

$$\max_{i,j} \{|a_n'(a_i, y_j) - \hat{a}_n(a_i, y_j)|\} < \varepsilon$$

   for some small number ε which determines the degree of tolerance in the solution algorithm.

7. If convergence is achieved, stop. Otherwise, go back to point 3 with the new guess $\hat{a}_1(a_i, y_j) = a_0'(a_i, y_j)$.

Note that the most time-consuming step in this procedure is 5, the root-finding problem. We now discuss how to avoid it.

Endogenous grid method (EGM)

- EGM is much faster than the traditional method because it does not require the use of a nonlinear equation solver
- The essential idea of the method is to construct a grid on a', next period's asset holdings, rather than on a, as is done in the standard algorithm.
- The method also requires the policy function to be at least weakly monotonic in the state
- Recall the Euler equation:

$$u_c\left(c\left(a,y\right)\right) \geq \beta R \sum_{y' \in Y} \pi\left(y'|y\right) u_c\left(c\left(a',y'\right)\right).$$

- As usual, we start from a guess $\hat{c}_0\left(a,y\right)$ and iterate on the Euler Equation until the decision rule for consumption that we solve for is essentially identical to the one in the previous iteration.

EGM: Algorithm

1. Construct a grid for (a, y)

2. Guess a policy function $\hat{c}_0(a_i, y_j)$. If y is persistent, a good initial guess is to set $\hat{c}_0(a_i, y_j) = ra_i + y_j$ which is the solution under quadratic utility if income follows a random walk.

3. Fix y_j. Instead of iterating over $\{a_i\}$, we iterate over $\{a'_i\}$. For any pair $\{a'_i, y_j\}$ on the mesh construct the RHS of the Euler equation [call it $B(a'_i, y_j)$]

$$B(a'_i, y_j) \equiv \beta R \sum_{y' \in Y} \pi(y'|y_j)\, u_c(\hat{c}_0(a'_i, y'))$$

   where the RHS of this equation uses the guess $\hat{c}_0$.

EGM: Algorithm

4. Use the Euler equation to solve for the value $\tilde{c}\left(a_i', y_j\right)$ that satisfies

$$u_c\left(\tilde{c}\left(a_i', y_j\right)\right) = B\left(a_i', y_j\right)$$

and note that it can be done analytically, e.g. for $u_c\left(c\right) = c^{-\gamma}$ we have $\tilde{c}\left(a_i', y_j\right) = \left[B\left(a_i', y_j\right)\right]^{-\frac{1}{\gamma}}$. Here algorithm becomes much more efficient because it does not require a nonlinear solver.

5. From the budget constraint:

$$\tilde{c}\left(a_i', y_j\right) + a_i' = Ra_i^* + y_j$$

solve for $a^*\left(a_i', y_j\right)$ the value of assets today that would lead the consumer to have a_i' assets tomorrow if her income shock was y_j today. This yields $c\left(a_i^*, y_j\right) = \tilde{c}\left(a_i', y_j\right)$, a function not defined on the grid points. This is the endogenous grid and it changes on each iteration.

EGM: Algorithm

6. Let a_0^* be the value of asset holdings that induces the borrowing constraint to bind next period, i.e., the value for a^* that solves that equation at the point a_0', the lower bound of the grid.

7. Now we need to update our guess defined on the original grid.

   - To get new guess $\hat{c}_1(a_i, y_j)$ on grid points $a_i > a_0^*$ we can use simple linear interpolation methods using values for $\left\{c(a_n^*, y_j), c\left(a_{n+1}^*, y_j\right)\right\}$ on the two most adjacent values $\left\{a_n^*, a_{n+1}^*\right\}$ that bracket a_i.

   - If some points a_i are beyond a_m^*, the upper bound of the endogenous grid, just extend linearly the function

EGM: Algorithm

- To update the consumption policy function on grid values $a_i < a_0^*$, we use the budget constraint:

$$\hat{c}_1(a_i, y_j) = Ra_i + y_j - a_0'$$

since we cannot use the Euler equation as the borrowing constraint is binding for sure next period: the reason is that we found it was binding at a_0^*, therefore a fortiori it will be binding for $a_i < a_0^*$.

8. Check convergence as before.

Envelope condition method (ECM)

It is an alternative method that, in some cases, avoids the use of nonlinear solvers

1. Construct a grid on a with $n+1$ points: call it $\mathcal{A}^V$.

2. Guess a value function $\hat{V}_0(a, y_j) = \sum_{k=0}^{n} w_{kj}^0 B_k(a)$ where B_k are piecewise linear splines.

3. Define another grid on a of size $n+1$: call it $\mathcal{A}^D$

4. Compute the derivative of the value function on the nodes of $\mathcal{A}^D$:

$$\hat{V}_0(a_i, y_j) = \sum_{k=0}^{n} w_{kj}^0 B_k'(a_i)$$

   You need to use a different grid because $\hat{V}_0$ is not differentiable on the nodes of the original grid $\mathcal{A}^V$

Envelope Condition Method (ECM)

5. For any pair $\{a_i, y_j\}$ on $\mathcal{A}^D \times \mathcal{Y}$ construct the envelope condition:

$$\hat{V}_0'(a_i, y_j) = u_c\left(Ra_i + y_j - \hat{a}_0(a_i, y_j)\right) R$$

from which we can obtain:

$$\hat{a}_0(a_i, y_j) = Ra_i + y_j - u_c^{-1}\left(\frac{\hat{V}_0'(a_i, y_j)}{R}\right)$$

6. Update the value function from the Bellman equation. For each point of the grid $\mathcal{A}^V$ compute:

$$\hat{V}_1(a_i, y_j) = u\left(Ra_i + y_j - \hat{a}_0(a_i, y_j)\right) + \beta \sum_{y_{j'} \in Y} \pi(y_{j'}|y_j)\, \hat{V}_0\left(\hat{a}_0(a_i, y_j), y_{j'}\right)$$

where a piece-wise linear interpolation is used to evaluate $\hat{a}_0(a_i, y_j)$ on the grid points a_i of $\mathcal{A}^V$ ($\neq$ grid points of $\mathcal{A}^D$)

Envelope Condition Method (ECM)

7. On each point of the grid $\mathcal{A}^V$ we must verify if the borrowing constraint binds and if it does, we set $\hat{a}_0(a_i, y_j) = 0$.

   - In practice, need to solve $(n+1)J$ equations in $(n+1)J$ coefficients $\left\{w_{kj}^1\right\}$ of the type:

$$\sum_{k=0}^{n} w_{kj}^1 B_k(a_i) = u\left(Ra_i + y_j - \hat{a}_0(a_i, y_j)\right) + \beta \sum_{y_{j'} \in Y} \pi(y_{j'}|y_j) \sum_{k=0}^{n} w_{kj'}^0 B_k\left(\hat{a}_0(a_i, y_j)\right)$$

     for all $i = 0, 1, ...n$ and $j = 1, 2, ..., J$

   - It is a linear system of equations

Checking accuracy of the numerical solution

- When using splines, how do we determine when the solution is accurate enough that no more points in the grid are needed?
- Or, when using the Chebishev approximation method, how do we determine that increasing the order of the polynomial would not lead to any significant improvement in accuracy?
- The numerically approximated Euler equation is

$$u_c(c_t) - \beta R E_t\left[u_c(c_{t+1})\right] \simeq 0.$$

- The above equation will be very close to zero when evaluated on grid points, but it may be quite far from zero outside the grid.
- Idea: compute Euler equation errors

Euler equation errors

- One can define the relative approximation error ε_t as that value such that the EE from simulated data holds exactly at t:

$$u_c\left(c_t(1-\varepsilon_t)\right) = \beta R E_t\left[u_c\left(c_{t+1}\right)\right]$$

- Then we have:

$$\varepsilon_t = 1 - \frac{u_c^{-1}\left(\beta R E_t\left[u_c\left(c_{t+1}\right)\right]\right)}{c_t}.$$

- For example, an error of 0.01 means that the agent is making a mistake equivalent to \$1 for every \$100 consumed in period t.

- Santos (2000) proves that the implied cost in terms of household welfare is the square of the Euler equation error, i.e., the error in the value function is of the order ε^2.

Euler equation errors

- By running a long simulation, and selecting states where the EE holds with equality, one can compute a string of EE errors and report either the maximum or the average of the errors in absolute value. This procedure may not explore the entire state space, but it will explore the region of the state space where the agent is more likely to find herself.

- Alternatively, one can explore the entire state space by choosing a grid for assets different from the one used in the computation and, from the decision rule, calculate for each point a_i on this new grid, and for every y_j the Euler equation error $\varepsilon\left(a_i, y_j\right)$.

- Usually, absolute errors are reported in base 10 logarithm, so a value of -2 means an error of \$1 for every \$100, a value of -3 means an error of \$1 for every \$1,000, and so on. Acceptable average errors should be around values of 4 and above.

Den Haan-Marcet statistics

- Den Haan and Marcet (1994) devise a simple test of numerical accuracy based on Hansen J test of overidentifying restrictions

- The consumption Euler equation

$$u_c(c_t) = \beta R E_t [u_c(c_{t+1})]$$

  implies that the residual

$$\varepsilon_{t+1} = u_c(c_t) - \beta R u_c(c_{t+1})$$

  should not be correlated with any variable dated t and earlier, since the expectation at time t is conditional on everything observable up to then.

Den Haan-Marcet statistics

- Therefore, we should have, for every t:

$$\mathbb{E}_t \left[\varepsilon_{t+1} \otimes \mathbf{z}_t \right] = 0,$$

  where $\otimes$ denotes element-by-element product. The vector $\mathbf{z}_t$ is a $(r \times 1)$ vector which can include all the variables in the information set of the agent at time t e.g. $\{y_j, c_j, a_j\}_{j=0}^{t}$.

- A badly approximated solution will not satisfy that property. This is the key idea of the test proposed by den Haan and Marcet.

- One can obtain an estimate of the LHS above through a simulation of length S of the model and the construction of

$$\mathbf{q}_S = \frac{\sum_{t=1}^{S} \hat{\varepsilon}_{t+1} \otimes \hat{\mathbf{z}}_t}{S},$$

  where $\hat{\varepsilon}_{t+1}$ and $\hat{\mathbf{z}}_t$ are the simulated counterparts of ε_{t+1} and $\mathbf{z}_t$.

Den Haan-Marcet statistics

- It can be shown that, under mild conditions, $\sqrt{S}\mathbf{q}_S \xrightarrow{d} N(0, V)$, and the appropriate quadratic form for the test statistics is:

$$S\mathbf{q}_S'\hat{V}_S^{-1}\mathbf{q}_S \xrightarrow{d} \chi_r^2$$

  where $\hat{V}_S^{-1}$ is the inverse of the matrix

$$V_S = \frac{\sum_{t=1}^{S} \hat{\varepsilon}_{t+1} \otimes \hat{\mathbf{z}}_t\hat{\mathbf{z}}_t' \otimes \hat{\varepsilon}_{t+1}}{S}$$

- Given a certain level of approximation in the solution, we can always find a number S large enough so that the approximation fails the accuracy test.

- This is not a problem when comparing solution methods, since one can determine the smallest S such that the method fails the test and compare these thresholds.

Den Haan-Marcet statistics

- When we want to judge the accuracy of our unique solution, DM suggest the following procedure.

  1. Draw a very long simulation path of the model (of length $S = 5,000$ with a burn-in period of $1,000$) and compute the statistic.

  2. Record whether the statistic is above the upper or below the lower 2.5% critical value given by the χ^2_r distribution.

  3. Repeat $N = 500$ times and calculate the fraction of times it fails. If it is significantly above the theoretical 5% then the test indicates an inaccurate solution.

Multidimensional interpolation

- Let's start from bilinear interpolation.
- You wish to interpolate $f(x,y)$ on a square $[\underline{x},\bar{x}]\times\left[\underline{y},\bar{y}\right]$ in $\mathbb{R}^2$
- Suppose you set your mesh and have obtained the value of the function at the four grid points $\{x_1,y_1\}$,$\{x_1,y_2\}$, $\{x_2,y_1\}$,$\{x_2,y_2\}$.
- What is the value of the function on a point (x,y) that belongs to the square defined by those four vertices?
- We first do linear interpolation in the x-direction

$$
\begin{aligned}
f(x,y_1) &= \frac{x_2-x}{x_2-x_1}f(x_1,y_1)+\frac{x-x_1}{x_2-x_1}f(x_2,y_1)\\
f(x,y_2) &= \frac{x_2-x}{x_2-x_1}f(x_1,y_2)+\frac{x-x_1}{x_2-x_1}f(x_2,y_2)
\end{aligned}
$$

and note that this gives expected results when $x=x_1$ and $x=x_2$

Multidimensional interpolation

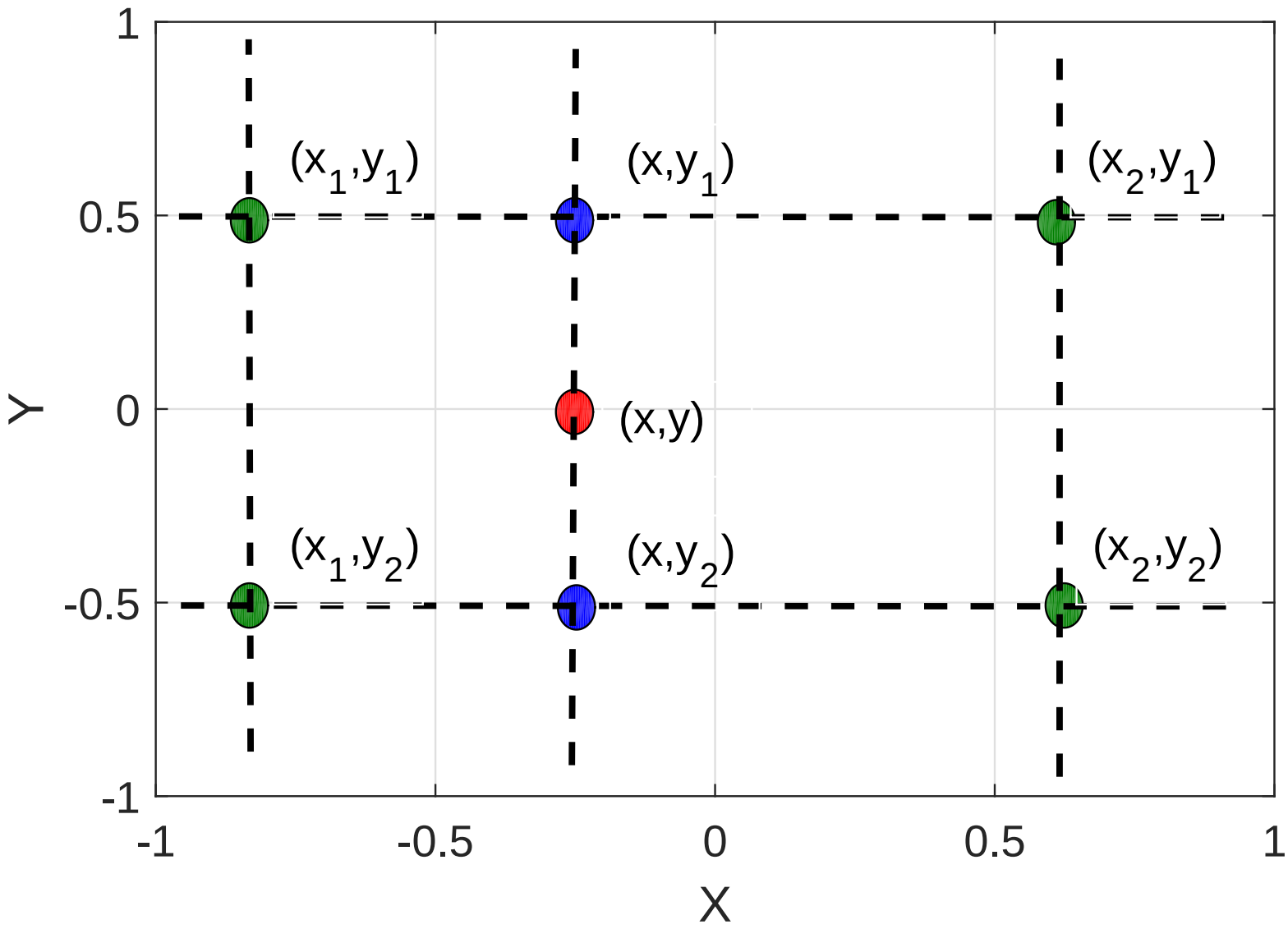
- Next, we interpolate in the y direction:

$$f(x,y) = \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

- This is the piecewise linear representation of the function in two dimensions

- Easy to figure out how to incorporate it into a more general algorithm to determine the policy function.

Bilinear interpolation



Tensor products for Chebyshev basis

- Let $x = (x_1, x_2, ..., x_d) \in [-1, 1]^d$
- Suppose we wish to approximate the function $f : [-1, 1]^d \to \mathbb{R}$ with Chebyshev polynomials of total degree n
- Let $T_k(x_i)$ denote the Chebyshev polynomial of degree $k = 0, 1, ..., n$ defined over the $i - th$ dimension of the state-space

$$f(x) \simeq \hat{f}(x) = \sum_{k_1=0}^{n} ... \sum_{k_i=0}^{n} ... \sum_{k_d=0}^{n} w_{k_1,...,k_n} \cdot T_{k_1}(x_1) \cdot ... \cdot T_{k_i}(x_i) \cdot ... \cdot T_{k_d}(x_d)$$

- Advantage: (i) simplicity, (ii) if the one-dim. basis is orthogonal in a norm, the tensor basis is orthogonal in the product norm
- Disadvantage: number of elements increases exponentially. We end up having terms of the type $x_1^n x_2^n \cdots x_d^n$: very high order!

Complete polynomials

- Solution: eliminate some elements of the tensor in such a way that there is not much numerical degradation

- This is the idea of complete polynomials. It originates from $n-th$ order Taylor expansion of a function defined on $\mathbb{R}^d$:

$$f(x) \simeq f(x^0) + \sum_{i=1}^{d} \frac{\partial f(x^0)}{\partial x_i} (x_i - x_i^0) + ...$$

$$+ \frac{1}{n!} \sum_{i_1=1}^{d} \cdots \sum_{i_n=1}^{d} \frac{\partial f(x^0)}{\partial x_{i_1} \cdots \partial x_{i_n}} (x_{i_1} - x_{i_1}^0) \cdots (x_{i_n} - x_{i_n}^0)$$

where there are no element of order higher than n

Complete polynomials

- Complete orthogonal basis use only terms of the tensor product with total degree n or less:

$$\mathcal{P}_n^d \equiv \left\{ x_1^{i_1} \cdots x_d^{i_d} \quad \text{s.t.} \quad \sum_{p=1}^{d} i_p \leq n \right\}$$

- Comparison of number of elements:

Degree of appx n	$\mathcal{P}_n^d$	Tensor product
$n = 2$	$1 + d + \frac{d(d+1)}{2}$	$(n+1)^d = 3^d$
$n = 3$	$1 + d + \frac{d(d+1)}{2} + d^2 + \frac{d(d-1)(d-2)}{6}$	$(n+1)^d = 4^d$

- Number of elements in complete polynomial basis increases geometrically vs tensor basis increases exponentially.

- Big gain in efficiency with little loss... often still too many elements

Smolyak algorithm for high dimensional interpolation

- Let $x = (x_1, x_2, ..., x_d) \in [-1, 1]^d$ with typical element x_i
- Suppose we wish to approximate the function $f : [-1, 1]^d \to \mathbb{R}$ with Chebyshev polynomials
- Let the degree of the interpolating polynomial in dimension i be $n_i - 1$, where $n_1 = 1$ and $n_i = 2^{i-1} + 1$ for $i = 2, ..., d$ so by constructions only polynomials of order $2, 4, 8$ etc. can be used for the interpolation on each dimension
- Define $\mathcal{G}^i = \left\{\zeta_1^i, \zeta_2^i, ..., \zeta_{n_i}^i\right\} \in [-1, 1]$ with $\mathcal{G}^1 = \{0\}$ the set of extrema of the Chebyshev polynomials, where

$$\zeta_j^i = -\cos\left(\frac{j-1}{n_i - 1}\pi\right) \quad j = 1, ..., n_i$$

Smolyak algorithm for high dimensional interpolation

- These are called Gauss-Lobotto nodes and interpolation over these nodes has nice properties as well.

- A crucial advantage of using Chebyshev extrema as opposed to Chebyshev nodes is that $\mathcal{G}^i \subset \mathcal{G}^{i+1}$ and this property is used to save on points.

- For an integer $q > d$ that measures the sparsity of the grid, we can define a Smolyak grid in $[-1,1]^d$ as

$$\mathcal{H}(q,d) = \bigcup_{q-d+1 \leq \sum_{p=1}^{d} i_p \leq q} \left(\mathcal{G}^{i_1} \times ... \times \mathcal{G}^{i_p} \times ... \times \mathcal{G}^{i_d} \right)$$

Smolyak algorithm for high dimensional interpolation

- By definition of $n_i = 2^{i-1} + 1$ and since $\sum_{p=1}^{d} i_p \geq q - d + 1$ the maximum number of points in a set $\mathcal{G}^i$ is $2^{q-d} + 1$ and it is along dimension $i = q - d + 1$, so the total degree of the interpolating Chebyshev polynomial is 2^{q-d}

- Very sparse and efficient grid. For example, if $d = 10$ and $q = 12$ the number of gridpoints is 221 and the total order of the polynomial is $2^2 = 4$, so quite flexible

- Kubler and Krueger (JEDC 2004) explain how to use it in conjunction with policy function iteration methods to solve for the Chebyshev coefficients

Example of Smolyak sparse grid in 2D

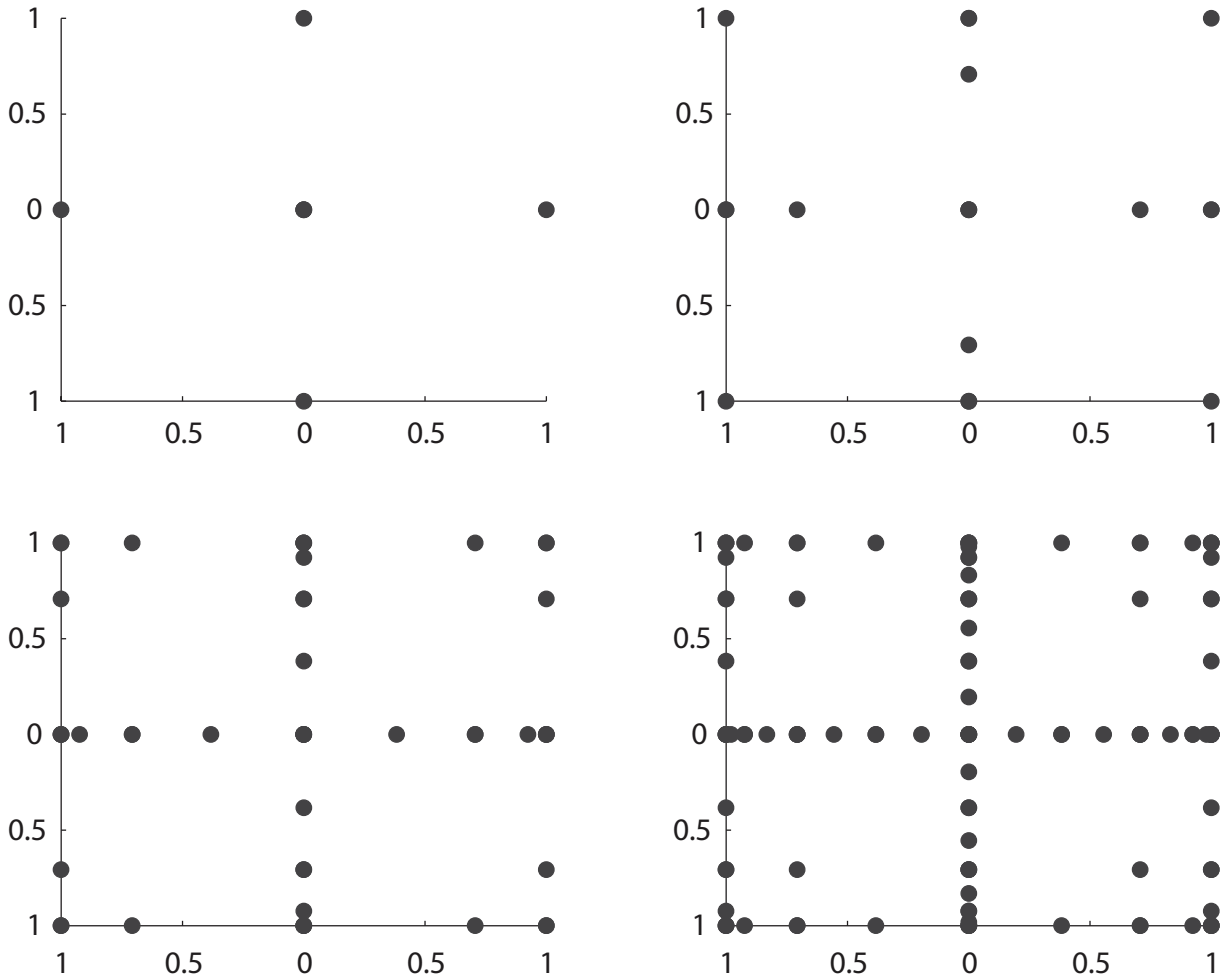
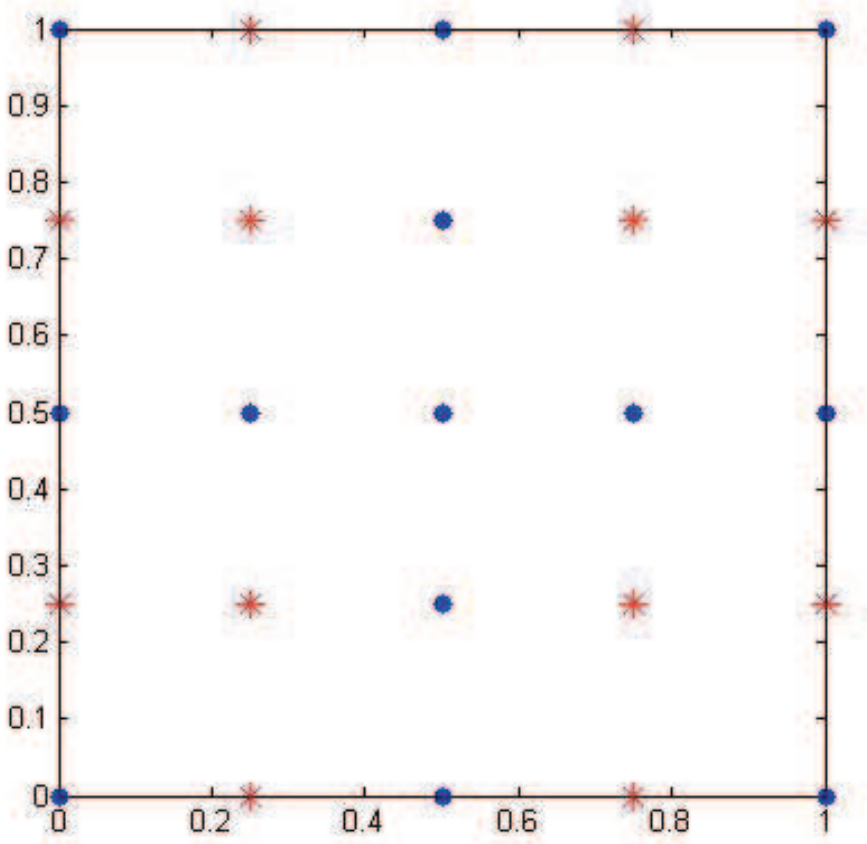


Fig. 1. The Grids $\mathscr{H}(3,2)$, $\mathscr{H}(4,2)$, $\mathscr{H}(5,2)$ and $\mathscr{H}(6,2)$.

Example of Smolyak sparse grid vs Tensor grid in 2D



Example of Smolyak sparse grid in 3D