Approximation and Interpolation of Functions

ECON-81500, Part I

Lilia Maliar

Motivation: Solving a Dynamic Economic Model

Neoclassical stochastic growth model with inelastic labor supply

- A one-agent stochastic growth model:

$$\max_{\{k_{t+1},c_t\}_{t=0}^{\infty}} E_0 \sum_{t=0}^{\infty} \beta^t u(c_t)$$

$$\text{s.t.} \quad c_t + k_{t+1} = (1-\delta) k_t + \theta_t f(k_t),$$

$$\ln \theta_{t+1} = \rho \ln \theta_t + \epsilon_{t+1}, \quad \epsilon_{t+1} \sim \mathcal{N}(0, \sigma^2),$$

initial condition (k_0, θ_0) is given;
$f(\cdot)$ = production function;
c_t = consumption; k_{t+1} = capital; θ_t = productivity level;
β = discount factor; δ = depreciation rate of capital;
ρ = autocorrelation coefficient of the productivity level;
σ = standard deviation of the productivity shock ϵ_{t+1}.

- The agent does not value leisure and supplies to the market all her time endowment $\Longrightarrow$ *A model with inelastic labor supply*.

First-order conditions

- We assume that a solution to the model is interior (satisfies the first-order conditions, FOCs):

$$u_1(c_t) = \beta E_t\left[u_1(c_{t+1})\left(1-\delta+\theta_{t+1}f_1(k_{t+1})\right)\right], \quad (1)$$

$$c_t + k_{t+1} = (1-\delta)k_t + \theta_t f(k_t). \quad (2)$$

- FOC (1) is the *Euler equation* or inter-temporal FOC (relates variables of different periods).
- Budget constraint (2) is intra-temporal (relates variables within the same period).

Concept of solution

- *Objective:* to find a recursive Markov solution in which the decisions on next-period capital and consumption are made according to some time invariant state-contingent functions

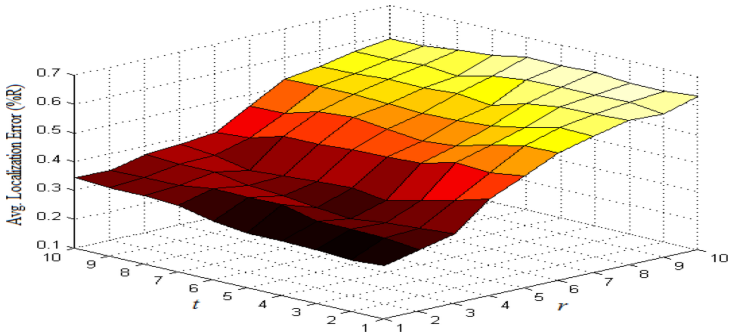
$$k' = K(k, \theta), \quad c = C(k, \theta).$$

- Note that if the capital decision function K is known, the consumption decision function follows from the budget constraint:

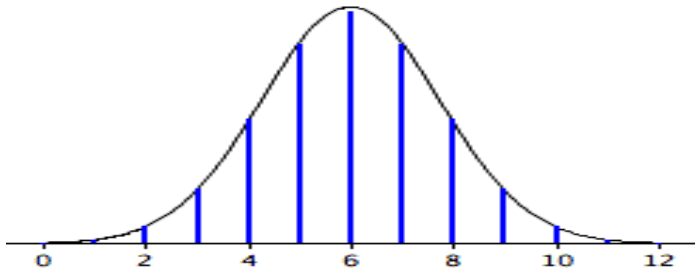
$$C(k, \theta) = (1 - \delta) k_t + \theta_t f(k_t) - K(k, \theta).$$

Projection-style Euler equation method, e.g., Judd (1992)

- A flexible functional form for approximation $\widehat{K}(k, \theta; b) \approx K(k, \theta)$, where b is a vector of parameters, for example, $k' = b_0 + b_1 k + b_2 \theta$.
- A grid of points $\{k_m, \theta_m\}_{m=1,\ldots,M}$ on which K is approximated:



- Deterministic integration methods approximate integrals with weighted average $\int_{-\infty}^{\infty} G(\epsilon) w(\epsilon) d\epsilon \approx \sum_{j=1}^{J} \omega_j G(\epsilon_j)$,



where $\{\epsilon_j\}_{j=1}^{J}$ and $\{\omega_j\}_{j=1}^{J}$ are integration nodes and weights.

Global projection-style Euler equation method

(EEM): A global projection-style Euler equation method.

Step 1. Choose functional form $\widehat{K}(\cdot, b)$ for representing K, where b is the coefficients vector. Choose a grid $\{k_m, \theta_m\}_{m=1,...,M}$ on which $\widehat{K}$ is constructed.

Step 2. Choose nodes, ϵ_j, and weights, ω_j, $j = 1, ..., J$, for approximating integrals. Compute next-period productivity $\theta'_{m,j} = \theta_m^{\rho} \exp(\epsilon_j)$ for all j, m.

Step 3. Solve for b that approximately satisfies the model's equations:

$$u_1(c_m) = \beta \sum_{j=1}^{J} \omega_j \cdot \left[u_1\left(c'_{m,j}\right)\left(1 - \delta + \theta'_{m,j} f_1(k'_m)\right)\right],$$
$$c_m = (1-\delta) k_m + \theta_m f(k_m) - k'_m$$
$$c'_{m,j} = (1-\delta) k'_m + \theta'_{m,j} f(k'_m) - k''_{m,j}$$

We have $J+2$ equations and $2J+2$ unknowns $k'_m, c_m, \left\{k''_{m,j}, c'_{m,j}\right\}_{j=1}^{J}$

$\Rightarrow$ the solution is under-determined.

Counting equations and unknowns

We use **stationarity**: the same decision function $\widehat{K}(\cdot, b)$ is used both at t and $t+1$:

- in the current period: $k'_m = \widehat{K}(k_m, \theta_m; b)$;
- in J possible future states: $k''_{m,j} = \widehat{K}\left(k'_m, \theta'_{m,j}; b\right)$, where future shocks are $\theta'_{m,j} = \theta_m^{\rho} \exp(\epsilon_j)$.

$$u_1(c_m) = \beta \sum_{j=1}^{J} \omega_j \left[u_1\left(c'_{m,j}\right)\left(1 - \delta + \theta'_{m,j} f_1\left(k'_m\right)\right)\right], \tag{3}$$

$$c_m = (1-\delta) k_m + \theta_m f(k_m) - k'_m, \tag{4}$$

$$c'_{m,j} = (1-\delta) k'_m + \theta'_{m,j} f\left(k'_m\right) - \widehat{K}\left(k'_m, \theta'_{m,j}; b\right). \tag{5}$$

- $J+2$ equations and $J+2$ unknowns: $c_m, k'_m, \left\{c'_{m,j}\right\}_{j=1}^{J} \Rightarrow$ the solution is exactly determined.

Discussion

- Substitute $c'_{m,j}$ from (5) into (3).
- Substitute c_m from (4) into (3).
- Substitute $k'_m = \widehat{K}\left(k_m, \theta_m; b\right)$ into the resulting equation.

$$\Downarrow$$

- Get one equation in which the vector of coefficients b is the only unknown.
- If $b = (b_1, b_2)$, we need just two grid points, $m = 1, 2$ to exactly identify b.

Questions arising in the context of this algorithm

- How to approximate the unknown decision functions?
- What families of functions to use for approximation?
- Possible choices: polynomials, splines, trigonometric functions, etc.
- Today, we will only talk about low-dimensional cases (one or few variables).
- High dimensional approximation and interpolation will be studied later (Smolyal sparse grids, ergodic-set methods, model reduction, machine learning, etc.).

Approximation and Interpolation in Numerical Analysis

General idea of approximation

- We consider a generic approximation problem (not in relation to solving dynamic economic models).
- *Objective*: Given data about a function $f(x)$, construct a parametric function $\widehat{f}(x, b)$ that approximately represents $f(x)$, where b is a vector of the parameters.
- Why could we need such an approximation?
  - Maybe $f(x)$ is known only in a finite set of points x_i, $i = 1, ..., n$ or in some interval, $y_i = f(x_i)$.
  - Maybe $f(x)$ is costly to evaluate, so we evaluate it in few points and approximate it in other points.

Questions:

- What *data* should be produced and used (assuming that we have control over where to place x_i, $i = 1, ..., n$)?
- What parametric *family* of functions (i.e., $\widehat{f}(x, b)$) should be used?
- How do we construct the approximation?

General problem

- Formally, the studied problem can be represented as

$$\min_b \left\| f(x) - \widehat{f}(x, b) \right\|,$$

where $\|\cdot\|$ is a norm for measuring approximation errors.

- There many classes of functions that can be used for approximation but we focus on a specific case of linear (in coefficients) approximations

$$\widehat{f}(x, b) = \sum_{j=1}^{m} b_j \phi_j(x),$$

where $\phi_1(x), ..., \phi_m(x)$ are basis functions and $b_1, ..., b_m$ are the corresponding coefficients.

Interpolation problem

- If the number of points is the same as the number of coefficients $n = m$, we have an *interpolation problem*

$$\begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} \phi_1(x_1) & \dots & \phi_n(x_1) \\ & \dots & \\ \phi_1(x_n) & \dots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} b_1 \\ \dots \\ b_n \end{bmatrix}$$

$$y = \Phi b.$$

  Provided that the basis functions are linearly independent, we find a unique inverse solution for the parameters $b = \Phi^{-1} y$.

- By construction the interpolant $\widehat{f}(x, b)$ pass through all the data points.

Approximation problem

- If the number of points is larger than the number of coefficients $n > m$, we have an *approximation problem*

$$\begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} \phi_1(x_1) & \dots & \phi_m(x_1) \\ & \dots & \\ \phi_1(x_n) & \dots & \phi_m(x_n) \end{bmatrix} \begin{bmatrix} b_1 \\ \dots \\ b_m \end{bmatrix},$$

$$y = \Phi b.$$

- Generally, m free parameters do not allow the interpolant to pass through all n points, therefore the left and right sides do not coincide.

Approximation (cont.)

- In this case, we choose the coefficients to minimize the approximation error according to some norm, such as least squares.

$$\min_{b_1,\ldots,b_m} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} b_j \phi_j \left(x_i \right) \right)^2 ,$$

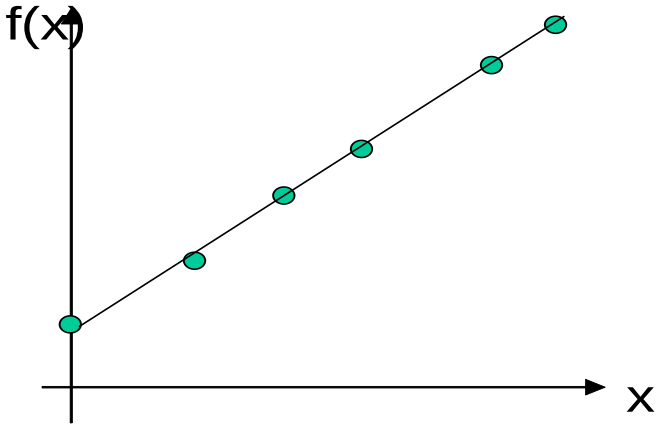
$$\min_{b} \left(y - \Phi b \right)' \left(y - \Phi b \right) .$$

- This is a familiar linear regression and b is the usual least squares coefficient $b = \left(\Phi' \Phi \right)^{-1} \Phi' y$.
- Unlike interpolant, the approximating function $\widehat{f}(x, b)$ does not passes through the data points.

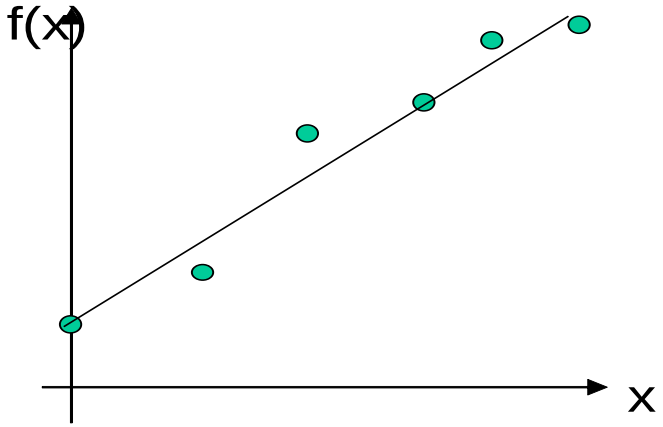
Interpolation versus Approximation

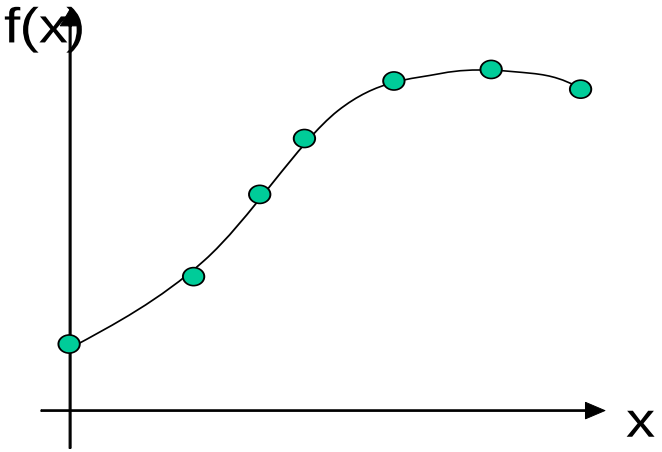
interpolation **approximation**

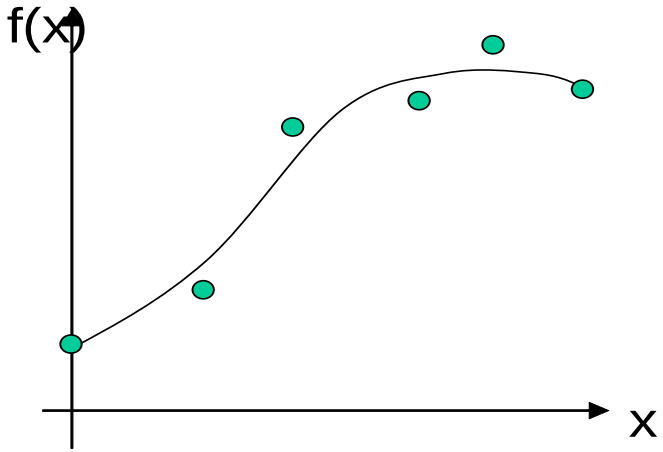
curve must pass through control points



curve is influenced by control points

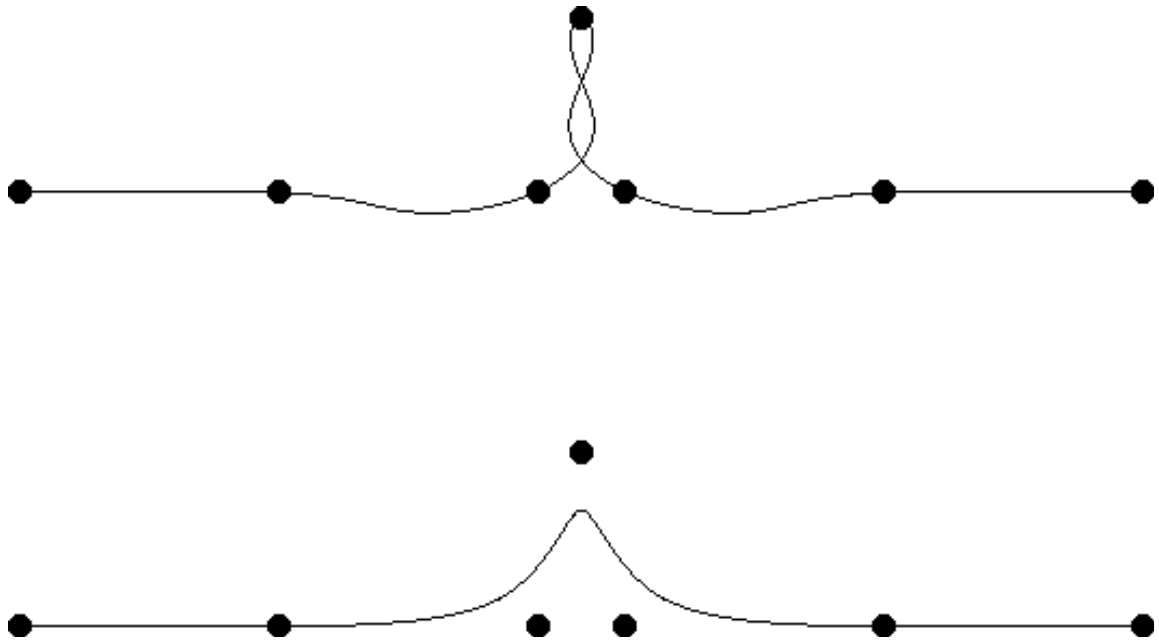






Interpolation versus Approximation

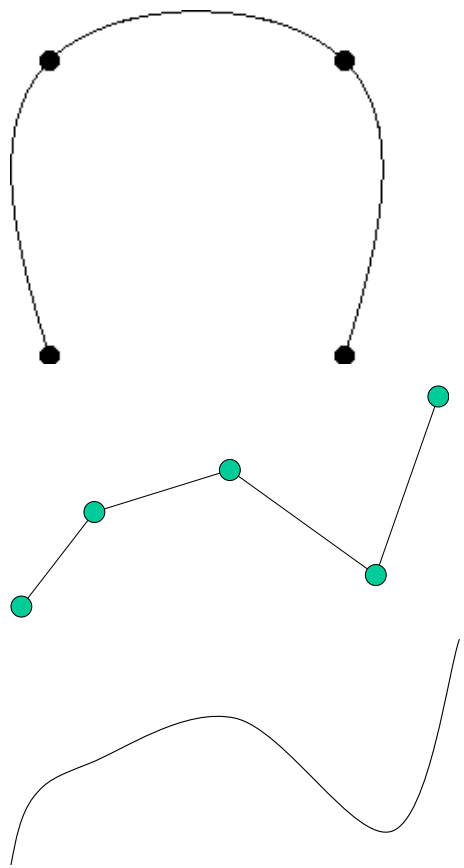
- *Interpolation:* find a function from an n-dimensional family of functions which exactly fits n data points.



- *Approximation* (rather than interpolation): there may be errors in the original measurements.
  – We don't have to interpolate original data points exactly.
  – We only need to approximate them.

Interpolation Curves

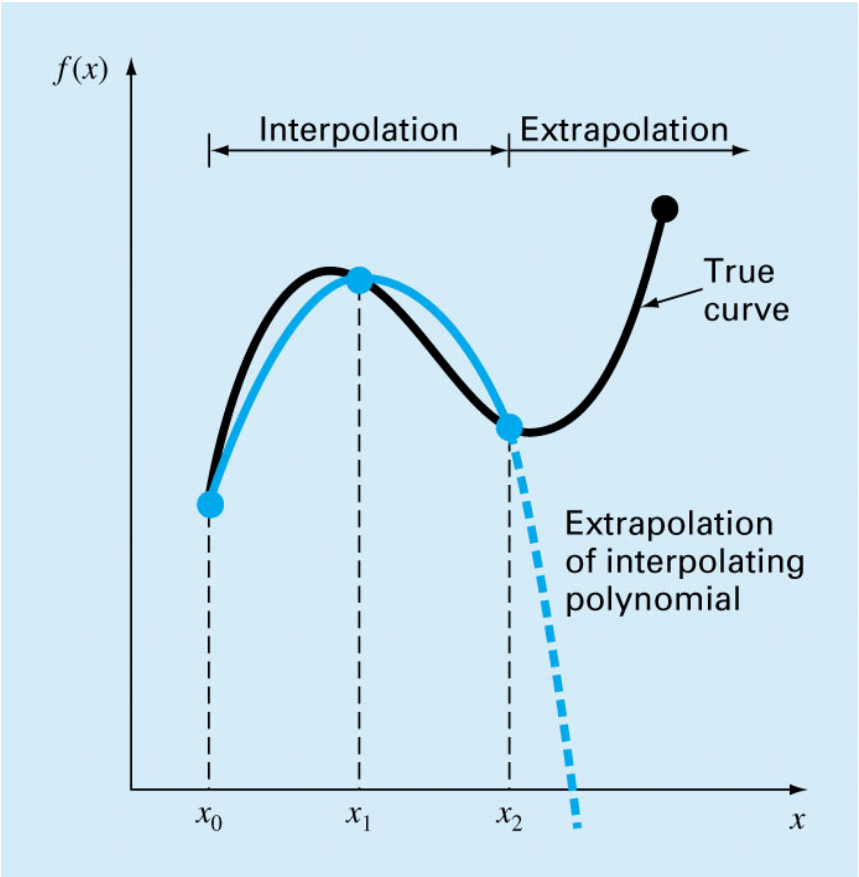
- Curve is constrained to pass through all control points.
- Piecewise linear: curve defined by multiple segments (linear).
- Piecewise polynomial: segments defined by polynomial functions; most common polynomial used is cubic (3rd order).



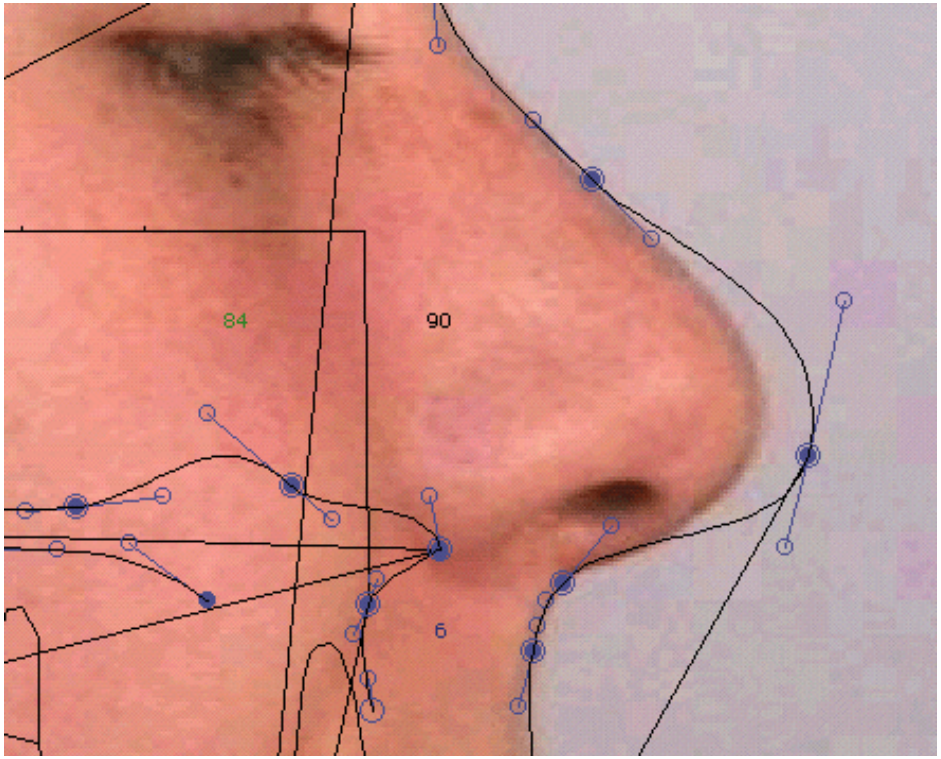
Extrapolation

- *Interpolation:* given a sequence of n unique points, (x_i, y_i), we want to construct a function $f(x)$ that passes through all the given points so that we can use $f(x)$ to estimate the value of y for any x inside the range of the known base points.
- *Extrapolation:* the process of estimating a value of $f(x)$ that lies outside the range of the known base points.

Interpolation is inside range; **extra**polation is outside.



Extrapolation: practical example



Example of approximation in economics

Example

Suppose we approximate a capital policy function K with an ordinary polynomial function

$$K(\cdot, b) = b_0 + b_1 k + b_2 a + ... + b_m a^p$$

on a set of simulated points $\{k_t, a_t\}$, $t = 1, ..., T$.

– Generally, we have more simulated points T than the polynomial coefficients in b.

– We compute the coefficients using a regression method. This is an example of approximation problem.

Example of interpolation in economics

Example

Consider an algorithm for solving an optimal growth model on the grid $\{k_i, a_i\}$, $i = 1, ..., m$.

– Suppose we defined a capital policy function for each points of the grid, i.e., we have $k_i' = K(k_i, a_i)$.

– To implement iteration on the Euler equation, we need to compute future capital $k_i'' = K(k_i', a_i')$.

– To do so, we need a method that allows to infer $K(k, a)$ for any point (k, a) on the domain given the value of this function in a finite number of grid points.

– This is an example of interpolation problem.

Plan of the lecture

1. Polynomial Interpolation.
2. Orthogonal Polynomials.
3. Approximation (Regression).
4. Splines.
5. Multidimensional Methods.

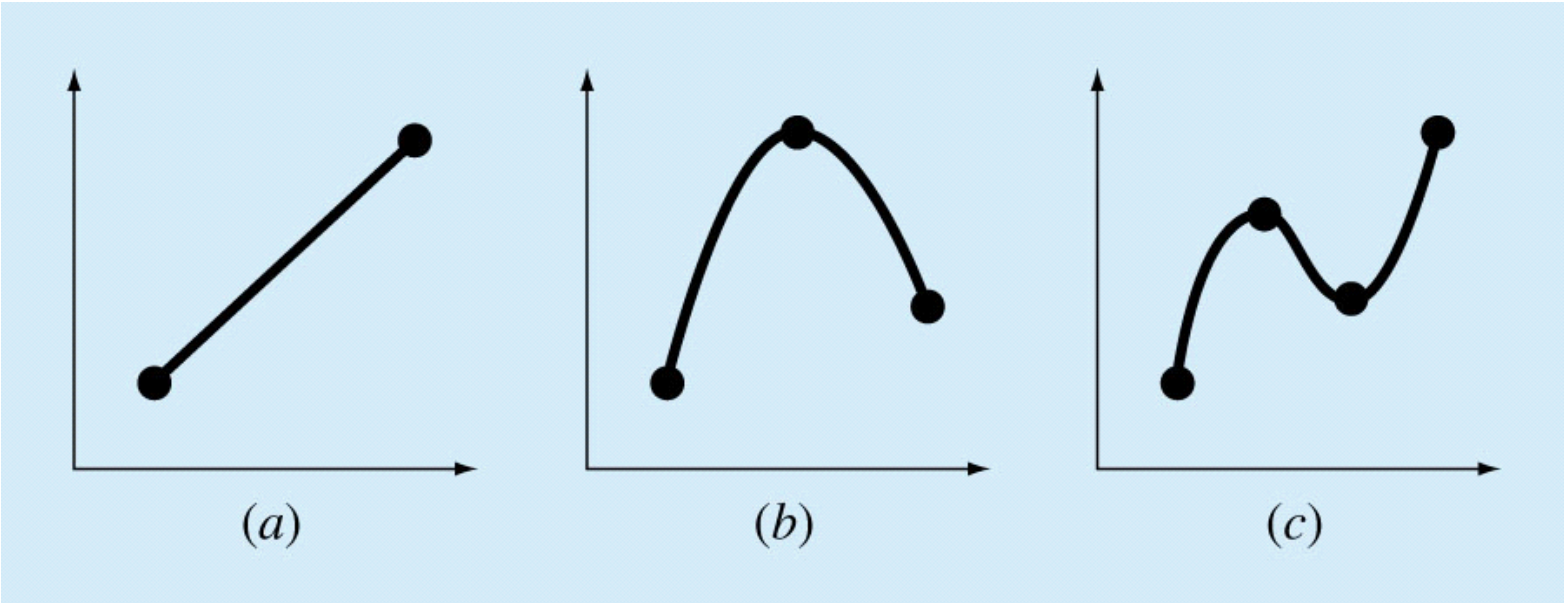
Polynomial Interpolation

Polynomial Interpolation

- Any continuous function can be represented as a sum of monomials, x^n, $n = 0, 1, 2, ...$
- Monomials are bases for continuous functions.
- *Objective:* Given n points, we want to find the *polynomial* of order $n-1$

$$p_n(x) = b_0 + b_1 x + b_2 x^2 + ... + b_{n-1} x^{n-1}$$

that passes through all the points.



Polynomial Interpolation (cont.)

- The $(n-1)$-th-order polynomial that passes through n points is unique, but it can be written in different mathematical formats:
  - Conventional form;
  - Lagrange Form;
  - Hermite Form (see the appendix); etc.
- Useful characteristics of polynomials
  - Infinitely differentiable
  - Can be easily integrated
  - Easy to evaluate

Conventional Form Polynomial

- One can compute the polynomial directly, $p_n(x) = \sum_{i=0}^{n-1} b_i x^i$. This is done by solving a linear system,

$$\begin{cases} y_1 = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_{n-1} x_1^{n-1} \\ y_2 = b_0 + b_1 x_2 + b_2 x_2^2 + \ldots + b_{n-1} x_2^{n-1} \\ \vdots \\ y_n = b_0 + b_1 x_n + b_2 x_n^2 + \ldots + b_{n-1} x_n^{n-1} \end{cases}.$$

- In matrix notation, $Ab = y$, where $b = (b_0, b_1, \ldots, b_{n-1})^\top$, and A is the so-called *Vandermonde* matrix for x_i, $i = 1, \ldots, n$

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_2^2 & \cdots & x_n^{n-1} \end{pmatrix}$$

- Result: If the (x_i) are distinct, there is a unique solution.

Conventional Form Polynomial

- What is the shortcoming of finding the polynomial using this method?

$$\begin{cases} y_1 = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_{n-1} x_1^{n-1} \\ y_2 = b_0 + b_1 x_2 + b_2 x_2^2 + \ldots + b_{n-1} x_2^{n-1} \\ \vdots \\ y_n = b_0 + b_1 x_n + b_2 x_n^2 + \ldots + b_{n-1} x_n^{n-1} \end{cases}.$$

- This system is typically ill-conditioned.
  – The resulting coefficients can be highly inaccurate when n is large (if finite-precision computers are used).
  – Talk about it later.
- If our objective is to determine the intermediate values between points, we can construct and represent the polynomials in Lagrange form.

Lagrange Polynomial Interpolation

- Data: $(x_i, y_i)\,, i = 1, .., n.$
- *Objective:* Find a polynomial of degree $n-1$, $p_n(x)$, which interpolates the sample (x_i, y_i) with $y_i = f(x_i)$,

$$y\,(x) = p_n(x) = \sum_{i=1}^{n} b_i l_i\,(x)\,,$$

$l_i\,(x)$ = weighting function, defined by

$$l_i\,(x) = \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}, \quad \text{for } i = 1, .., n$$

with the property

$$l_i\,(x_j) = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

- Note

$$l_i\,(x) = \frac{x - x_1}{x_i - x_1} \cdot \ldots \cdot \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdot \frac{x - x_n}{x_i - x_n}.$$

Lagrange Polynomial Interpolation (cont.)

- Example: quadratic Lagrange interpolation ($n-1=2$) includes $l_1(x)$, $l_2(x)$ and $l_3(x)$:

$$y(x) = b_1 l_1(x) + b_2 l_2(x) + b_3 l_3(x).$$

- Recall $l_i(x_j) = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$. Evaluate it at x_1:

$$y(x_1) = b_1 \underbrace{l_1(x_1)}_{=1} + b_2 \underbrace{l_2(x_1)}_{=0} + b_3 \underbrace{l_3(x_1)}_{=0} = b_1.$$

- In general, $b_i = y_i = f(x_i)$.
- Thus, interpolation is given by

$$p_n(x) = \sum_{i=1}^{n} f(x_i) l_i(x)$$

- Example: $p_1(x) = f(x_0)\frac{x-x_1}{x_0-x_1} + f(x_1)\frac{x-x_0}{x_1-x_0}$.

Example

- Construct a 4th order polynomial in Lagrange form that passes through the following points:

i	1	2	3	4	5
x_i	0	1	-1	2	-2
$f(x_i)$	-5	-3	-15	39	-9

- We can construct the polynomial as

$$p_1(x) = -5l_1(x) - 3l_2(x) - 15l_3(x) + 39l_4(x) - 9l_5(x)$$

- Recall

$$l_i(x) = \frac{x - x_1}{x_i - x_1} \cdot \ldots \cdot \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdot \frac{x - x_n}{x_i - x_n}.$$

- For example, $l_1(x) = \frac{(x-1)(x+1)(x-2)(x+2)}{(0-1)(0+1)(0-2)(0+2)} = \frac{(x-1)(x+1)(x-2)(x+2)}{4}$.

Lagrange Polynomial Interpolation (cont.)

Problems

- If the number of data points is large, this type of interpolation can be expansive.
- To compute a single $l_i(x)$ we need to compute $2(n-1)$ subtractions, n multiplications.
- This has to be constructed for the n data points to compute all needed $l_i(x)$.
- Then to compute $p_n(x)$ we need n additions and n multiplications.

Problems with Interpolation

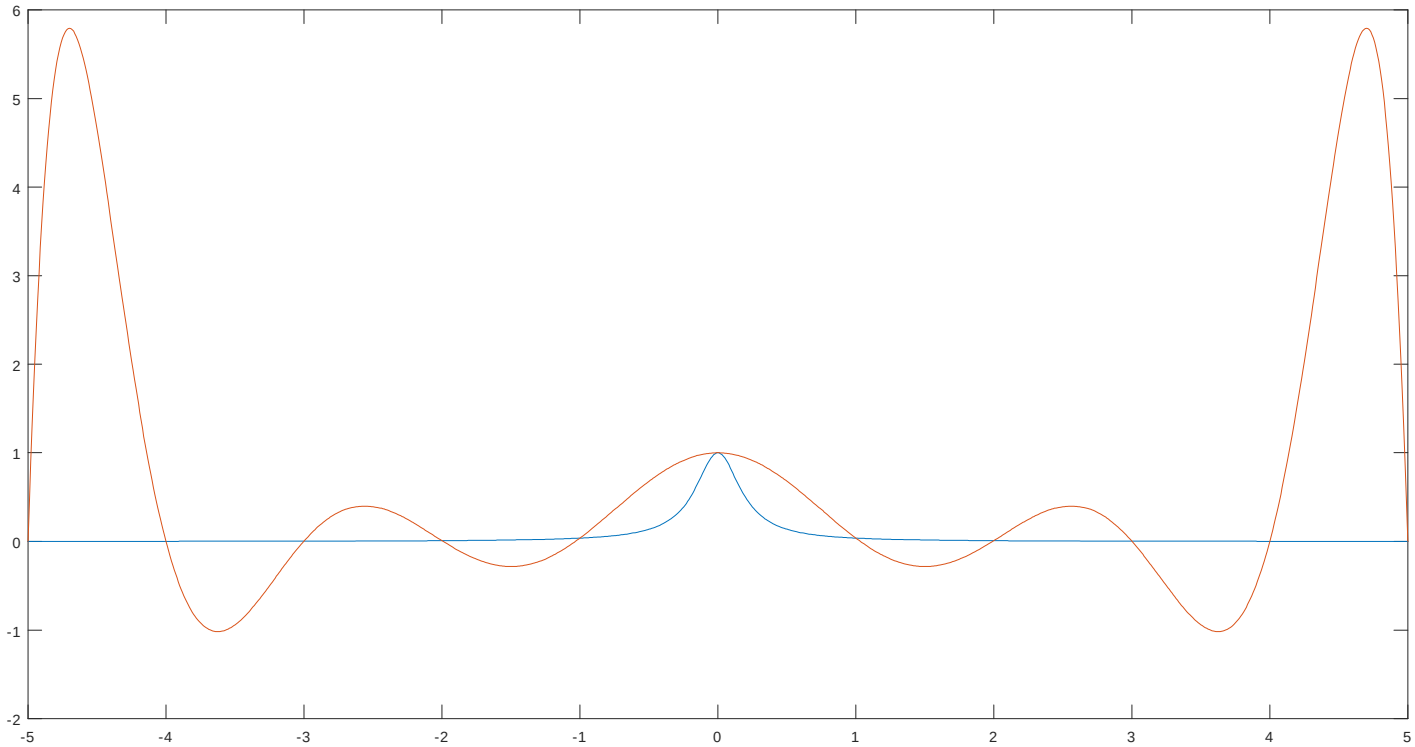
- Interpolation does not always work even for well-behaved functions.
- Does $p_n(x)$ converge to $f(x)$ as we use more points? No!
- Consider a so-called Runge function

$$\begin{aligned} f(x) &= \frac{1}{1+x^2}, \\ x_i &= -5, -4, \ldots, 3, 4, 5 \end{aligned}$$

  i.e., 11 uniformly sampled points.
- Use Lagrange interpolation (fit degree 10 polynomial).
- *Conclusion:* under $(n-1)$ interpolation at n uniformly spaced points, $p_n(x)$ gets worse as we use more points.

Problems with Interpolation (cont.)

Runge function



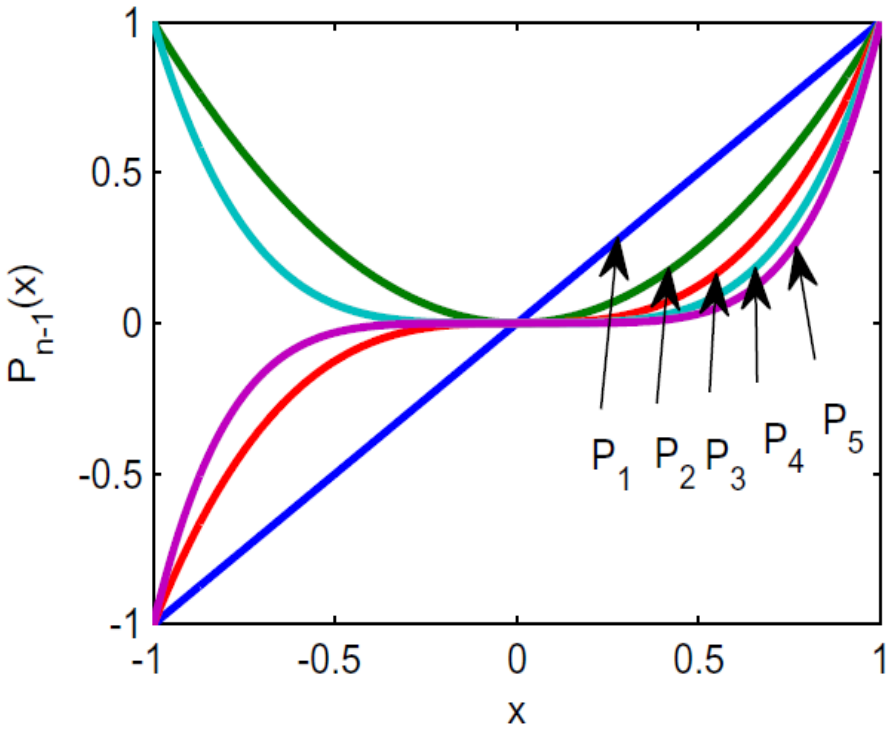
Highly oscillatory. Hard to control away from measured points.

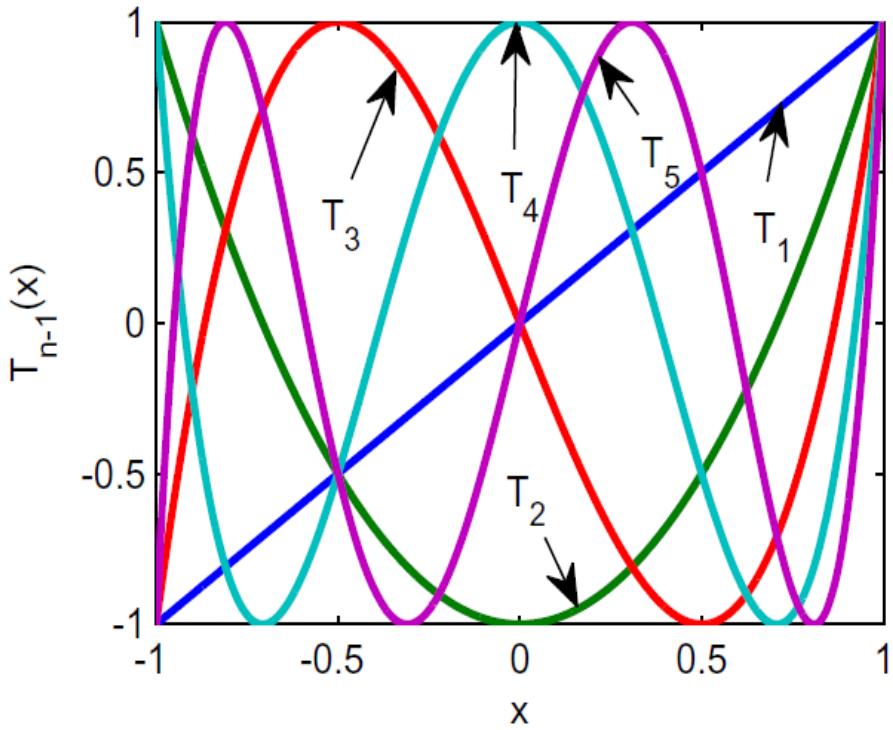
Orthogonal Polynomials

Polynomials

- Ordinary
- Chebyshev
- Legendre
- Laguere
- Hermite, etc.

Ordinary and Chebyshev Polynomials (cont.)



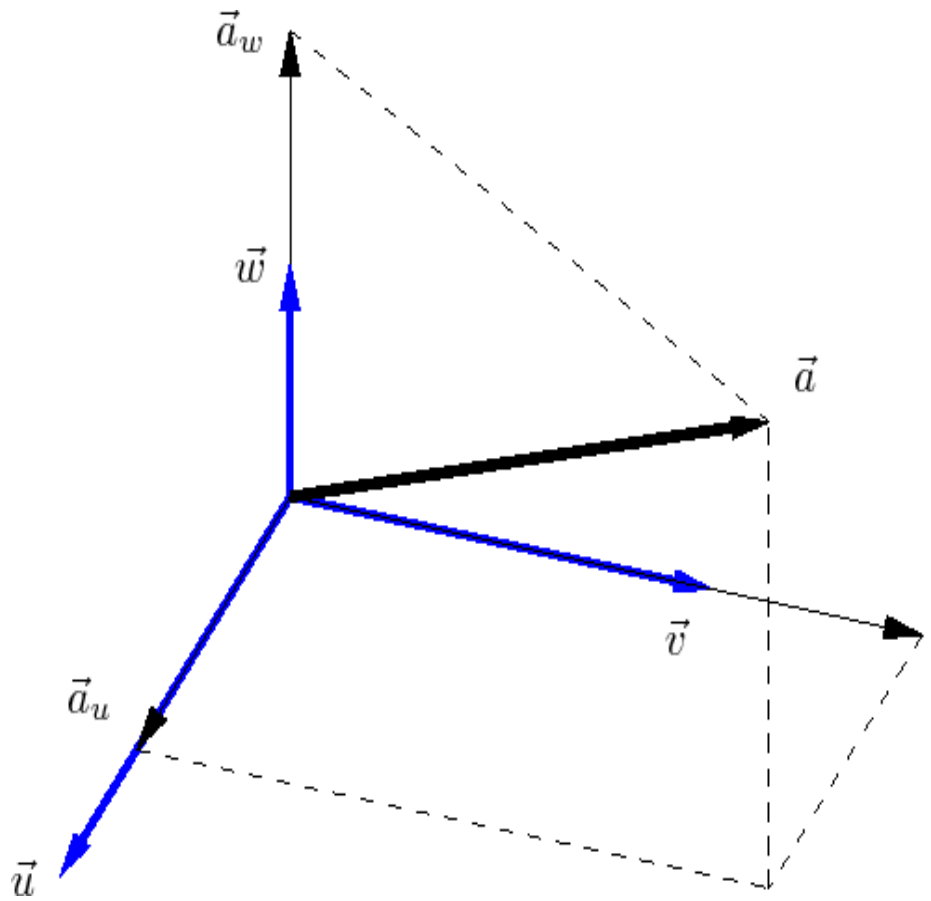


Comments about Ordinary and Chebyshev Polynomials

- For the ordinary polynomial family, the basis functions look very similar on $\mathbb{R}_+$.
- Approximation methods using ordinary polynomials may fail because they cannot distinguish between similarly shaped polynomial terms such as x^2 and x^4.
- In contrast, for the Chebyshev polynomial family, basis functions have very different shapes and are easy to distinguish.
- Chebyshev polynomials are orthogonal.
- Good bases have an orthogonality property.
- Why? Analogy with orthogonal vectors.

Analogy with Orthogonal Vectors

- Two vectors x and y are said to be *orthogonal* if their scalar product is zero, $(x, y) = 0$.
- To represent any vector in a 3-dimensional space, we usually use 3 orthogonal vectors.
- We can also use 3 non-orthogonal vectors but it can happen that they are close to each other and we have very big coefficients.



Orthogonal versus Non-orthogonal Vectors

Example

Consider a vector $d = (1, 1, 1)$. We have an orthogonal basis from 3 vectors $a = (1, 0, 0)$, $b = (0, 1, 0)$ and $c = (0, 0, 1)$. Note that a scalar product of (a, b), (c, b), $(a, c) = 0$.

$$(a, b) = (1, 0, 0) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = 0$$

Then, $d = 1 \cdot a + 1 \cdot b + 1 \cdot c$.

Orthogonal versus Non-orthogonal Vectors (cont.)

Example

Suppose now that the basis vectors are $a = (1, 0, 0)$, $b = (1, 0.01, 0)$ and $c = (1, 0, 0.01)$.

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \alpha a + \beta b + \gamma c = \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ 0.01 \\ 0 \end{bmatrix} + \gamma \begin{bmatrix} 1 \\ 0 \\ 0.01 \end{bmatrix}$$

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -199.0 \\ 100.0 \\ 100.0 \end{bmatrix}$$

So, instead of $(1, 1, 1)$, we have coefficients (α, β, γ) that are very big $(-199, 100, 100)$.

Orthogonal Polynomials for Interpolation

- Coming back to the model, suppose that we would like to approximate a function by some family of functions $\{f_0(x), f_1(x), f_2(x), f_3(x), ...\}$,

$$F(x) \cong \alpha f_0(x) + \beta f_1(x) + \gamma f_2(x) + ...$$

- It is like to approximate a vector d by vectors a, b, c.
- We would like functions $\{f_0(x), f_1(x), f_2(x), f_3(x), ...\}$ to be orthogonal, i.e., their scalar product is zero, $\langle f_0(x), f_1(x) \rangle = 0$.
- Consider monomials $\{1, x, x^2, x^3, ...\}$. They are not orthogonal and very close to each other (see the figure).
- So, some coefficients can be very large which leads to the loss of digits in the finite arithmetic precision.

Orthogonal Polynomials

General orthogonal polynomials

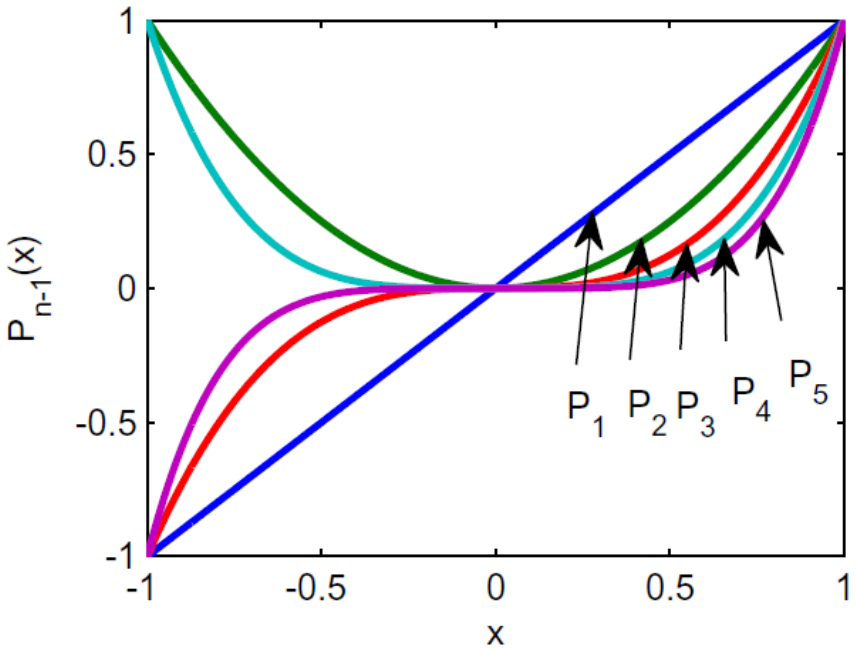
- Space: polynomials over domain D
- Weighting function: $w(x) > 0$
- Inner product of two functions f and g: $\langle f, g \rangle \equiv \int_D f(x)g(x)w(x)dx$

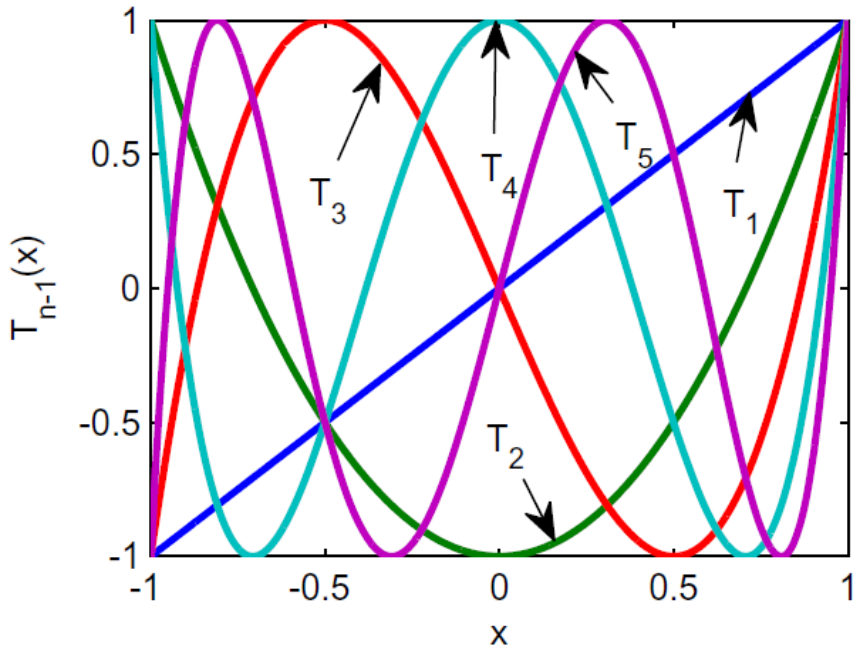
Definition

$\{\phi_i\}$ is a family of orthogonal polynomials w.r.t $w(x)$ iff

$$\left\langle \phi_i, \phi_j \right\rangle = 0, \ i \neq j$$

Orthogonal Polynomials (cont.)





Orthogonal Polynomials (cont.)

- Recurrence formulas

$$\begin{aligned} \phi_0(x) &= 1 \\ \phi_1(x) &= x \\ \phi_{k+1}(x) &= (a_{k+1}x + b_k)\,\phi_k(x) + c_{k+1}\phi_{k-1}(x) \end{aligned}$$

Example

$\phi_2(x) = (a_2 x + b_1)\,\phi_1(x) + c_2\phi_0(x)$

$\langle \phi_2, \phi_0 \rangle = a_2 \langle x\phi_1, \phi_0 \rangle + b_1 \langle \phi_1, \phi_0 \rangle + c_2 \langle \phi_0, \phi_0 \rangle$

$0 = a_2 \langle x\phi_1, \phi_0 \rangle + 0 + c_2 \langle \phi_0, \phi_0 \rangle \quad \Longrightarrow \quad c_2 = \frac{a_2 \langle x\phi_1, \phi_0 \rangle}{\langle \phi_0, \phi_0 \rangle}$

Similarly, $\langle \phi_2, \phi_1 \rangle = a_2 \langle x\phi_1, \phi_1 \rangle + b_1 \langle \phi_1, \phi_1 \rangle + c_2 \langle \phi_0, \phi_1 \rangle$

$0 = a_2 \langle x\phi_1, \phi_1 \rangle + b_1 \langle \phi_1, \phi_1 \rangle + 0 \quad \Longrightarrow \quad b_1 = \frac{a_2 \langle x\phi_1, \phi_1 \rangle}{\langle \phi_1, \phi_1 \rangle}.$

Chebyshev Polynomials

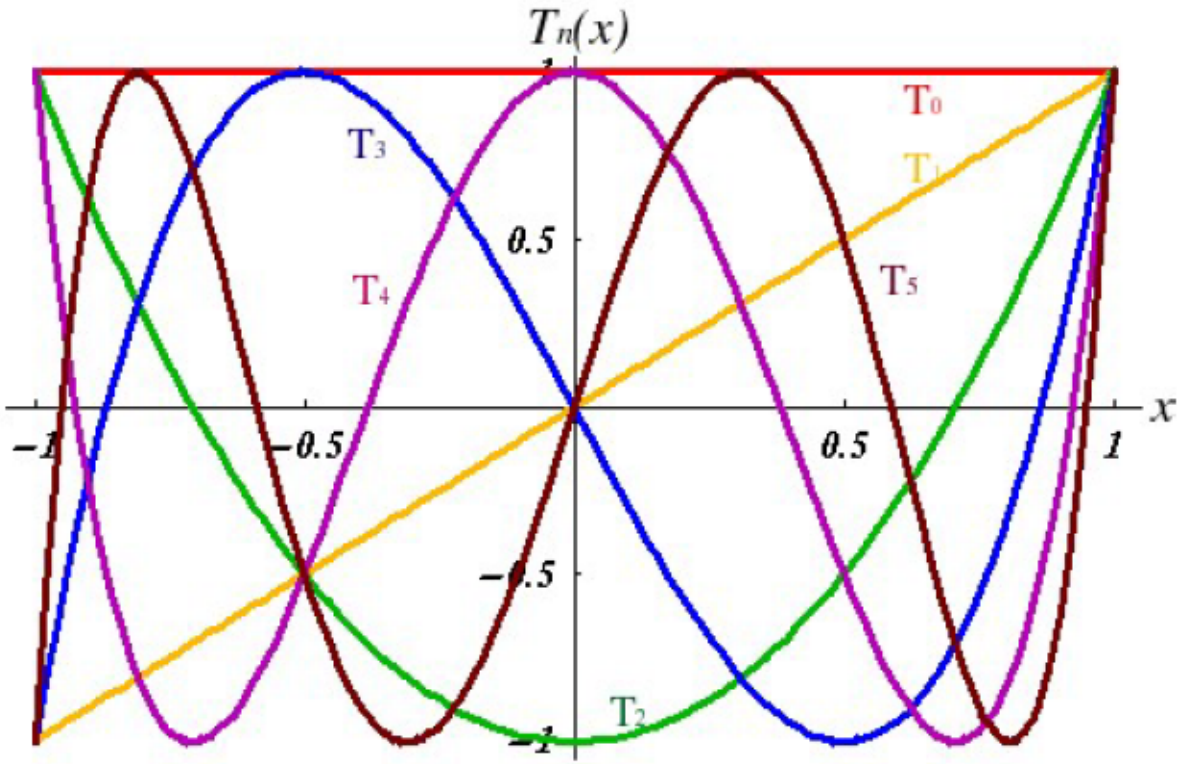
- $D = [a, b] = [-1, 1]$
- $w(x) = \left(1 - x^2\right)^{-1/2}$
- $T_n(x) = \cos(n \arccos(x))$
- Recurrence formula:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \\ &\dots \\ T_{n+1}(x) &= 2x\, T_n(x) - T_{n-1}(x). \end{aligned}$$

- This recurrence formula follows from the recurrence relation for cosines:

$$\cos((n+1)\theta) = 2\cos(\theta)\cos(n\theta) - \cos((n-1)\theta).$$

Chebyshev Polynomials (cont.)



- Again, $T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x)$.
- For example, $T_0(x) = 0$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$, and $T_3(x) = 4x^3 - 2x^2 - 2x^2 + 1 = 4x^3 - 4x^2 + 1$.

Orthogonality of Chebyshev Polynomials (cont.)

- Check the orthogonality $\int_{-1}^{1} T_i(x)\,T_j(x)\,w(x)\,dx = 0$:

$$
\begin{aligned}
0 &= \int_{-1}^{1} T_0(x)\,T_1(x)\,w(x)\,dx \\
&= \int_{-1}^{1} 1 \cdot x \cdot \left(1-x^2\right)^{-1/2} dx = \int_{-1}^{1} 1 \cdot \left(1-x^2\right)^{-1/2} \frac{dx^2}{2} \\
&\underset{y=x^2}{=} \frac{1}{2} \int_{1}^{1} (1-y)^{-1/2}\, dy = 0.
\end{aligned}
$$

Discrete Orthogonality of Chebyshev Polynomials

- Chebyshev polynomials satisfy a discrete orthogonality relation too.
- That is, if x_k $(k = 1, ..., m)$ are the m zeros of $T_m(x)$, and if i, $j < m$, then

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

Discrete Orthogonality of Chebyshev Polynomials (cont.)

Example (1)

Let $m = 3$; $T_3(x) = 4x^3 - 3x$ has 3 zeros: $x_1 = 0$, $x_2 = \frac{\sqrt{3}}{2}$, $x_3 = -\frac{\sqrt{3}}{2}$.

$$\sum_{k=1}^{3} T_0(x_k)\, T_1(x_k) = \underset{=1}{T_0(x_1)}\underset{=0}{T_1(x_1)} + \underset{=1}{T_0(x_2)}\underset{=\frac{\sqrt{3}}{2}}{T_1(x_2)} + \underset{=1}{T_0(x_3)}\underset{=-\frac{\sqrt{3}}{2}}{T_1(x_3)} = 0$$

$$\sum_{k=1}^{3} T_1(x_k)\, T_2(x_k) = \underset{=0}{T_1(x_1)}\underset{=-1}{T_2(x_1)} + \underset{=\frac{\sqrt{3}}{2}}{T_1(x_2)}\underset{=\frac{1}{2}}{T_2(x_2)} + \underset{=-\frac{\sqrt{3}}{2}}{T_1(x_3)}\underset{=\frac{1}{2}}{T_2(x_3)} = 0$$

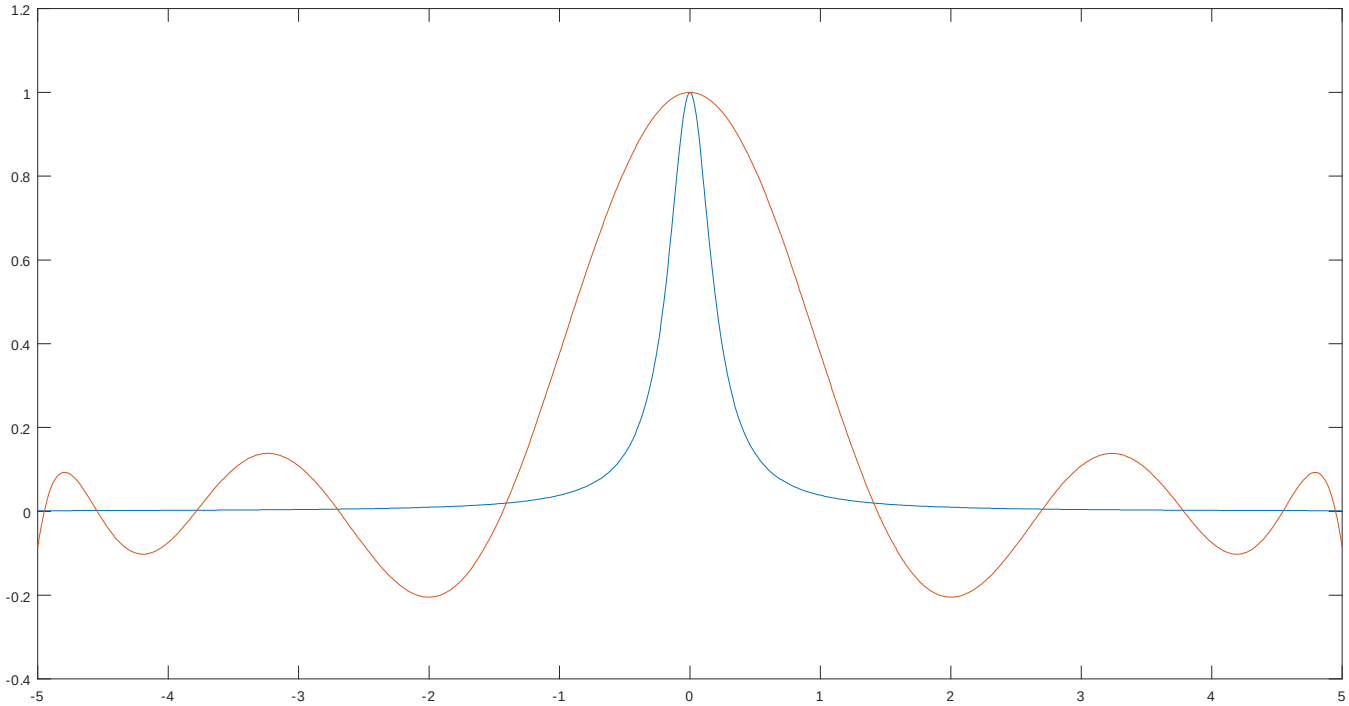
$$\sum_{k=1}^{3} T_0(x_k)\, T_0(x_k) = \underset{=1}{T_0(x_1)}\underset{=1}{T_0(x_1)} + T_0(x_2)\, T_0(x_2) + T_0(x_3)\, T_0(x_3) =$$

$$\sum_{k=1}^{3} T_1(x_k)\, T_1(x_k) = \underset{=0}{T_1(x_1)}\underset{=0}{T_1(x_1)} + \underset{=\frac{\sqrt{3}}{2}}{T_1(x_2)}\underset{=\frac{\sqrt{3}}{2}}{T_1(x_2)} + \underset{=-\frac{\sqrt{3}}{2}}{T_1(x_3)}\underset{=-\frac{\sqrt{3}}{2}}{T_1(x_3)} = \frac{3}{2}$$

$$\sum_{k=1}^{3} T_2(x_k)\, T_2(x_k) = T_2(x_1)\, T_2(x_1) + T_2(x_2)\, T_2(x_2) + T_2(x_2)\, T_2(x_2) = \frac{3}{2}$$

Chebyshev Polynomials and Runge Function

- Using Chebyshev nodes to approximate the Runge function improves approximation (but the result is not completely good).



Chebyshev Polynomials for Interpolation

Let us illustrate the use of Chebyshev polynomials for interpolation by way of example.

Example

Let $f(x)$ be a function defined on an interval $[-1,1]$, and let us approximate this function with a Chebyshev polynomial function of degree two, i.e.,

$$f(x) \approx \widehat{f}(x;b) = b_1 + b_2 x + b_3\left(2x^2 - 1\right).$$

We compute $b \equiv (b_1, b_2, b_3)$ so that $\widehat{f}(\cdot;b)$ and f coincide in three extrema of Chebyshev polynomials, namely, $\{-1,0,1\}$,

$$\begin{aligned}
\widehat{f}(-1;b) &= b_1 + b_2\cdot(-1) + b_3\left(2\cdot(-1)^2 - 1\right) = f(-1)\\
\widehat{f}(0;b) &= b_1 + b_2\cdot 0 + b_3\left(2\cdot 0^2 - 1\right) = f(0)\\
\widehat{f}(1;b) &= b_1 + b_2\cdot 1 + b_3\left(2\cdot 1^2 - 1\right) = f(1).
\end{aligned}$$

Chebyshev Polynomials for Interpolation (cont.)

Example (cont.)

This leads us to a system of three linear equations with three unknowns that has a unique solution

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} f(-1) \\ f(0) \\ f(1) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} f(-1) \\ f(0) \\ f(1) \end{bmatrix} = \begin{bmatrix} \frac{f(-1)}{4} + \frac{f(0)}{2} + \frac{f(1)}{4} \\ -\frac{f(-1)}{2} + \frac{f(1)}{2} \\ \frac{f(-1)}{4} - \frac{f(0)}{2} + \frac{f(1)}{4} \end{bmatrix}.$$

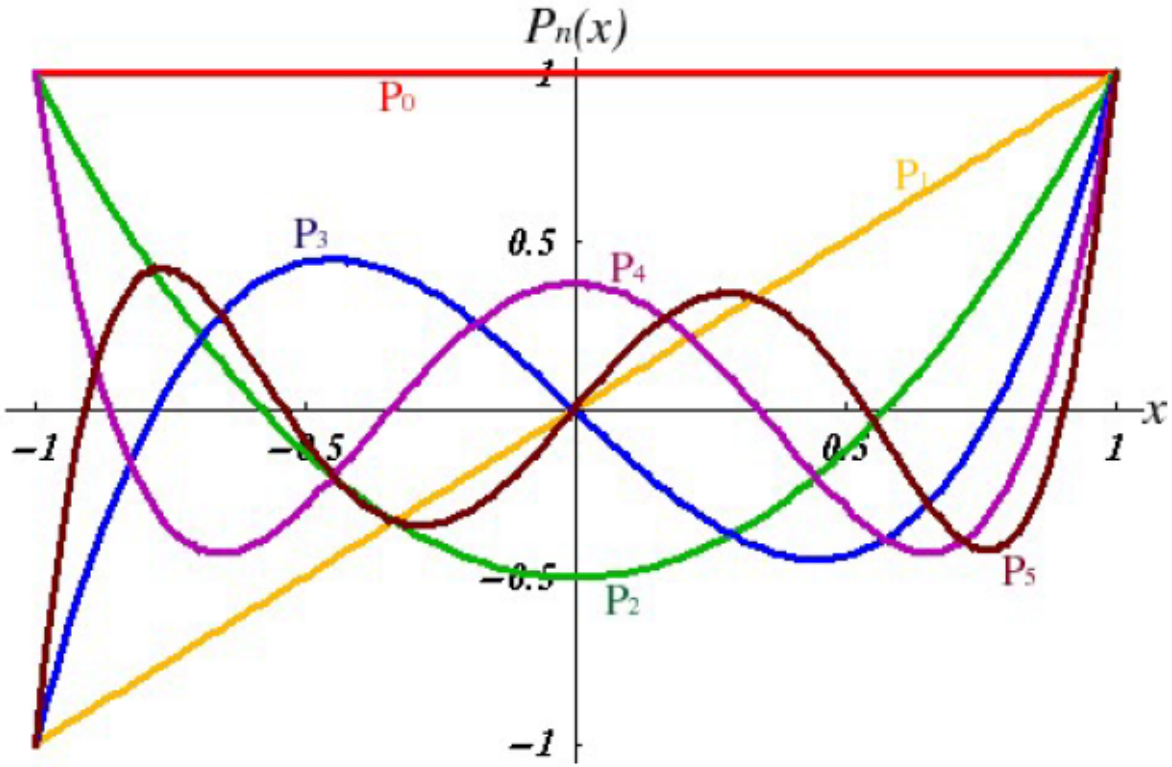
- It is possible to use Chebyshev polynomials with other grids, but the grid of extrema (or zeros) of Chebyshev polynomials is a perfect match.

Legendre Polynomials

- $D = [a, b] = [-1, 1]$
- $w(x) = 1$
- $P_n(x) = \frac{(-1)^n}{2^n n!} \frac{d^n}{dx^n} \left[(1 - x^2)^n\right]$
- Recurrence formula:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_{n+1}(x) &= \frac{2n+1}{n+1}\, x\, P_n(x) - \frac{n}{n+1}\, P_{n-1}(x), \end{aligned}$$

Legendre Polynomials (cont.)

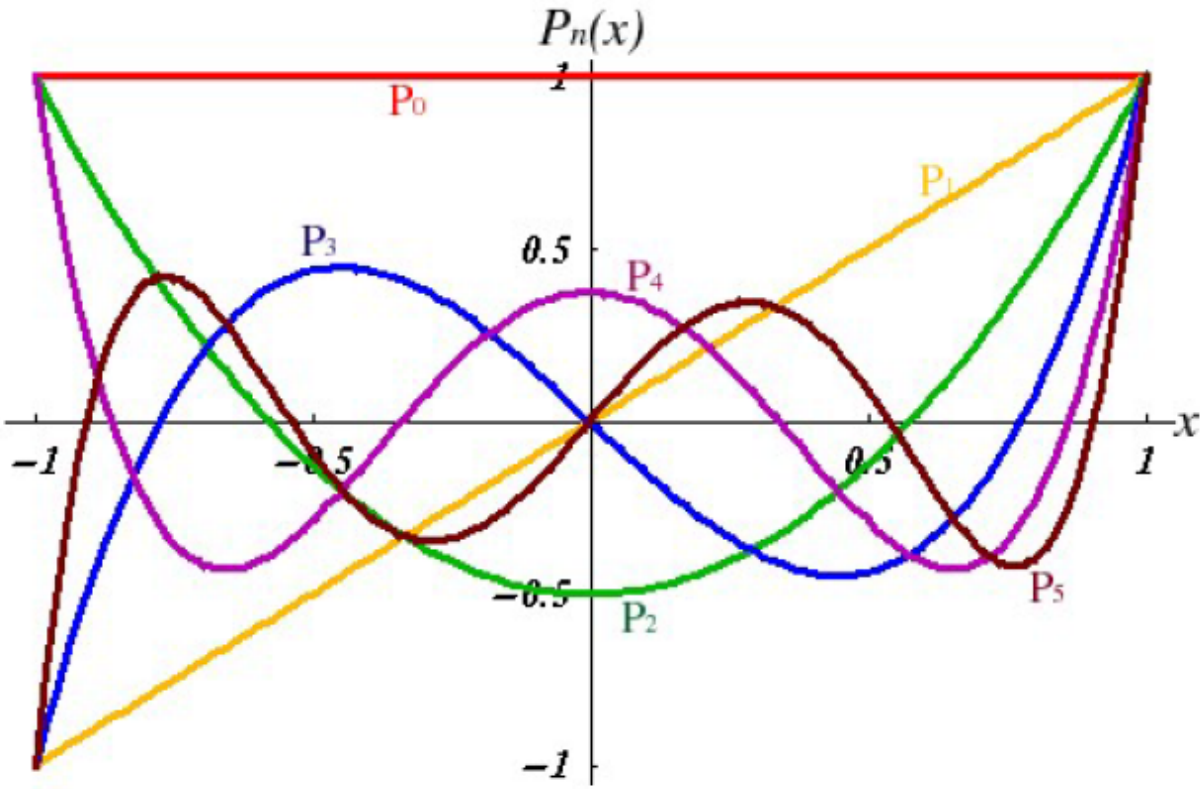


Laguerre Polynomials

- $D = [a, b] = [0, \infty)$
- $w(x) = e^{-x}$
- $L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$
- Recurrence formula:

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= 1 - x \\ L_{n+1}(x) &= \frac{1}{n+1} (2n + 1 - x) L_n(x) - \frac{n}{n+1} L_{n-1}(x), \end{aligned}$$

Laguerre Polynomials (cont.)

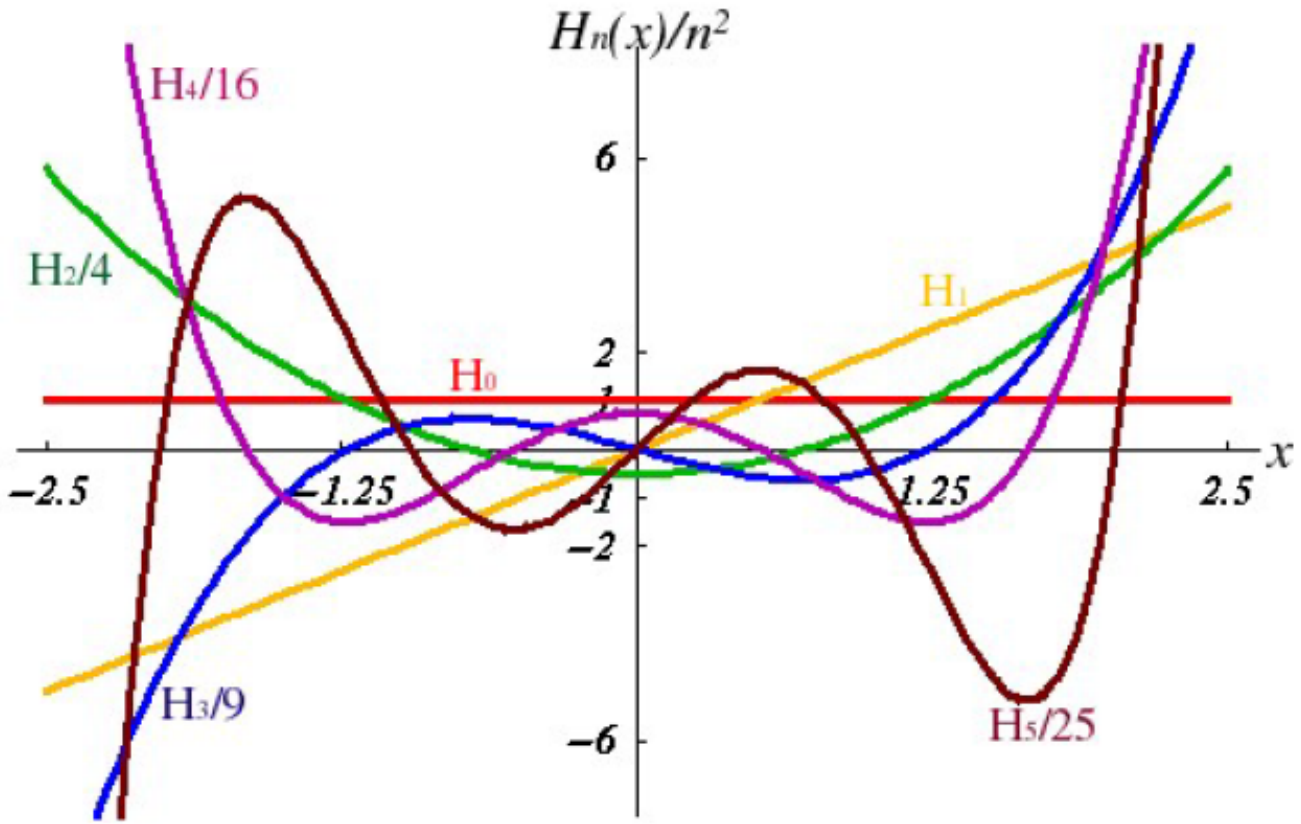


Hermite Polynomials

- $D = [a, b] = (-\infty, \infty)$
- $w(x) = e^{-x^2}$
- $H_n(x) = (-1)^n e^{x^2} \ \frac{d^n}{dx^n} \ (e^{-x^2})$
- Recurrence formula:

$$\begin{aligned} H_0(x) &= 1 \\ H_1(x) &= 2x \\ H_{n+1}(x) &= 2x\, H_n(x) - 2n\ H_{n-1}(x). \end{aligned}$$

Hermite Polynomials (cont.)



Note that the Hermite Polynomials are scaled down by a factor of n^2 in order to be fit on the same plot.

General Orthogonal Polynomials

- Few problems have the specific intervals and weights used in definitions.
- One must adapt interval through linear change of variables (COV).
- If compact interval $[a, b]$ is mapped to $[-1, 1]$ by

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

  where $x \in [a, b]$ and $y \in [-1, 1]$.
- Then $\phi_i\left(-1 + 2\frac{x-a}{b-a}\right)$ are orthogonal over $x \in [a, b]$ with respect to $w\left(-1 + 2\frac{x-a}{b-a}\right)$ iff $\phi_i(y)$ are orthogonal over $y \in [-1, 1]$ w.r.t. $w(y)$.
- There are also COV for mapping a half-infinite interval $[a, \infty]$ to $[0, \infty]$ or for mapping an infinite $[-\infty, \infty]$ to $[-\infty, \infty]$.

Approximation (Regression)

Regression

- When the number of data points n is larger than the number of unknown coefficients m, least-squares regression can be used to find an approximation.
- Data: $(x_i, y_i)\,, i = 1, .., n$.
- *Objective:* Find a function $f(x; \beta)$ with $\beta \in R^m$, $m \leq n$, with $y_i \doteq f(x_i), i = 1, .., n$.
- Least Squares regression:

$$\min_{\beta \in R^m} \sum \left(y_i - f\left(x_i; \beta\right)\right)^2$$

Chebyshev Regression (Approximation)

- Approximation function: degree m polynomial
- Data: n data points, $(x_i, y_i), i = 1, .., n$ are the n zeroes of $T_n(x)$ adapted to $[a, b]$
- More data than unknown coefficients: $n > m + 1$
- Objective: minimize unweighted sum of errors at nodes
- Chebyshev interpolation is a special case of regression with $n = m + 1$.

Orthogonal Polynomials

- Approximation (assuming $\langle \phi_i, \phi_i \rangle \equiv \| \phi_i \| = 1$):

$$\begin{aligned} f(x) &\cong \sum_{i=0}^{\infty} a_i \phi_i \\ a_i &= \langle f, \phi_i \rangle = \int_D f(x) \phi_i(x) w(x) dx \end{aligned}$$

- To find a_i's, explicit formulas are derived to avoid OLS.

Chebyshev Approximation Algorithm

General algorithm in $\mathbb{R}^1$:

- Objective: Given $f(x)$ defined on $[a, b]$, find a m-point degree-n Chebyshev polynomial approximation $p(x)$
- Step 1: Compute the $m \geq n+1$ Chebyshev interpolation nodes on $[-1, 1]$:

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

- Step 2: Adjust nodes to $[a, b]$ interval (back transformation, i.e., $z_k \in [-1, 1]$ and $x_k \in [a, b]$:

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

- Step 3: Evaluate f at approximation nodes:

$$w_k = f(x_k)\ ,\ k = 1, \cdots, m,$$

where w_k are values of the function f.

Chebyshev Approximation Algorithm (cont.)

- Step 4: Compute Chebyshev coefficients, $a_i, i = 0, \cdots, n$:

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

to arrive at approximation of $f(x, y)$ on $[a, b]$:

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

- Alternatively, one can solve for a using OLS formula $a = (X'X)^{-1} X'y$ where

$$X = \begin{pmatrix} T_0(z_1) & \ldots & T_n(z_1) \\ \ldots & & \ldots \\ T_0(z_m) & \ldots & T_n(z_m) \end{pmatrix}, \; y = \begin{pmatrix} f(x_1) \\ \ldots \\ f(x_m) \end{pmatrix}$$

Chebyshev Approximation Algorithm (cont.)

- For example, in our Example 1, we have

$$X'X = \begin{pmatrix} \sum_{k=1}^{3} T_0(z_k)\, T_0(z_k) & \sum_{k=1}^{3} T_0(z_k)\, T_1(z_k) & \sum_{k=1}^{3} T_0(z_k)\, T_2(z_k) \\ \sum_{k=1}^{3} T_0(z_k)\, T_1(z_k) & \sum_{k=1}^{3} T_1(z_k)\, T_1(z_k) & \sum_{k=1}^{3} T_2(z_k)\, T_1(z_k) \\ \sum_{k=1}^{3} T_0(z_k)\, T_2(z_k) & \sum_{k=1}^{3} T_2(z_k)\, T_1(z_k) & \sum_{k=1}^{3} T_2(z_k)\, T_2(z_k) \end{pmatrix}$$

- Note that $(X'X)^{-1}$ is a diagonal matrix due to the discrete orthogonality relation:

$$X'X = \begin{pmatrix} 3 & 0 & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & 0 & \frac{3}{2} \end{pmatrix}$$

- In general, $a_i = \begin{cases} \frac{1}{m} \sum_{k=1}^{m} w_k\, T_i(z_k) \text{ for } i = 0 \\ \frac{2}{m} \sum_{k=1}^{m} w_k\, T_i(z_k) \text{for } i \geq 1 \end{cases}$

Chebyshev Approximation Algorithm: Example

We are given $f(x)$ defined on $[a, b]$. Let $[a, b] = [1, 2]$ and $f(x) = \ln(x)$.

- Step 1: Compute the zeros of Chebyshev polynomials, $z_k \in [-1, 1]$, from

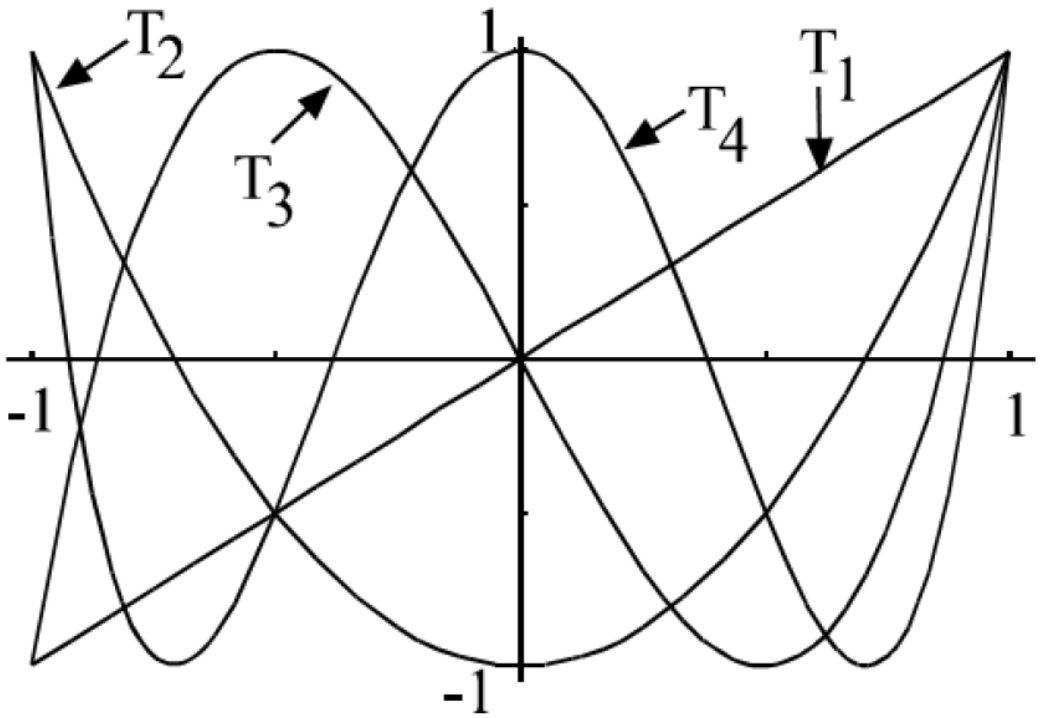
$$z_k = -\cos\left(\frac{2k-1}{2m}\ \pi\right)\ ,\ k = 1, \cdots, m.$$

For $m = 3$: $k = 1$, $z_1 = -0.87$; $k = 2$, $z_2 = 0$; $k = 3$, $z_3 = 0.87$.

- Step 2: Adjust nodes to $[1, 2]$ interval:

$$\begin{aligned} x_1 &= (-0.87 + 1)\left(\frac{2-1}{2}\right) + 1 = 1.07, \\ x_2 &= (0 + 1)\left(\frac{2-1}{2}\right) + 1 = \frac{3}{2}, \\ x_3 &= (0.87 + 1)\left(\frac{2-1}{2}\right) + 1 = 1.93. \end{aligned}$$

Chebyshev Approximation Algorithm: Example (cont.)



Chebyshev Approximation Algorithm: Example (cont.)

- Step 3: Evaluate f at approximation nodes:

$$\begin{aligned} w_1 &= \ln x_1 = \ln(1.07), \\ w_2 &= \ln x_2 = \ln\left(\frac{3}{2}\right), \\ w_3 &= \ln x_3 = \ln(1.93). \end{aligned}$$

Chebyshev Approximation Algorithm: Example (cont.)

- Step 4: Compute Chebyshev coefficients, a_0, a_1, a_2 on $T_0(z) = 1$, $T_1(z) = z$, $T_2(z) = 2z^2 - 1$ in $p(x)$:

$$
\begin{aligned}
a_0 &= \frac{w_1 T_0(z_1) + w_2 T_0(z_2) + w_3 T_0(z_3)}{T_0(z_1)^2 + T_0(z_2)^2 + T_0(z_3)^2} \\
&= \frac{\ln(1.07) \cdot 1 + \ln\left(\frac{3}{2}\right) \cdot 1 + \ln(1.93) \cdot 1}{1+1+1} = 0.38 \\
a_1 &= \frac{w_1 T_1(z_1) + w_2 T_1(z_2) + w_3 T_1(z_3)}{T_1(z_1)^2 + T_1(z_2)^2 + T_1(z_3)^2} \\
&= \frac{\ln(1.07) \cdot (-0.87) + \ln\left(\frac{3}{2}\right) \cdot 0 + \ln(1.93) \cdot (0.87)}{(-0.87)^2 + 0 + (0.87)^2} = 2 \\
a_2 &= \frac{w_1 T_2(z_1) + w_2 T_2(z_2) + w_3 T_2(z_3)}{T_2(z_1)^2 + T_2(z_2)^2 + T_2(z_3)^2} = \ldots = 0.389
\end{aligned}
$$

Chebyshev Approximation Algorithm: Example (cont.)

- Form the polynomial function:

$$
\begin{aligned}
p(x) &= a_0 \underbrace{T_0\left(2\frac{x-1}{2-1}-1\right)}_{=1} + a_1 T_1\left(2\frac{x-1}{2-1}-1\right) \\
&\quad + a_2 T_2\left(2\frac{x-1}{2-1}-1\right) \\
&= 0.38 + 2 \cdot T_1\left[2\left(x-1\right)-1\right] + 0.389 \cdot \left(\left[2\left(x-1\right)-1\right]^2 - 1\right)
\end{aligned}
$$

Minmax Approximation

- Data: (x_i, y_i), $i = 1, .., n$.
- Objective: L^∞ fit

$$\min_{\beta \in R^m} \max_i \|y_i - f(x_i; \beta)\|,$$

  where $f(x_i; \beta)$ is a polynomial function with β being parameters.
- $\forall i$, compute $y_i - f(x_i; \beta) =$ error of approximation.
- Find a maximum error across all i.
- Choose such a β that minimizes such maximum error.

Chebyshev Minmax Property

Theorem

Suppose $f : [-1, 1] \to R$ *is* C^k *for some* $k \geq 1$, *and let* I_n *be the n-point (degree* $n-1$*) polynomial interpolation of* f *based at the zeroes of* $T_n(x)$. *Then*

$$\| f - I_n \|_\infty \leq \left(\frac{2}{\pi} \log(n+1) + 1 \right) \times \frac{(n-k)!}{n!} \left(\frac{\pi}{2} \right)^k \left(\frac{b-a}{2} \right)^k \| f^{(k)} \|_\infty$$

Decompose the error bound

- $\frac{2}{\pi} \log(n+1)$: grows very slowly in n; ignore it
- $\left(\frac{\pi}{2}\right)^k \left(\frac{b-a}{2}\right)^k$: independent of n and f
- $\| f^{(k)} \|_\infty$: a measure of k'th order curvature
- $\frac{(n-k)!}{n!}$: essentially $\frac{1}{n^k}$ (decreases very rapidly).

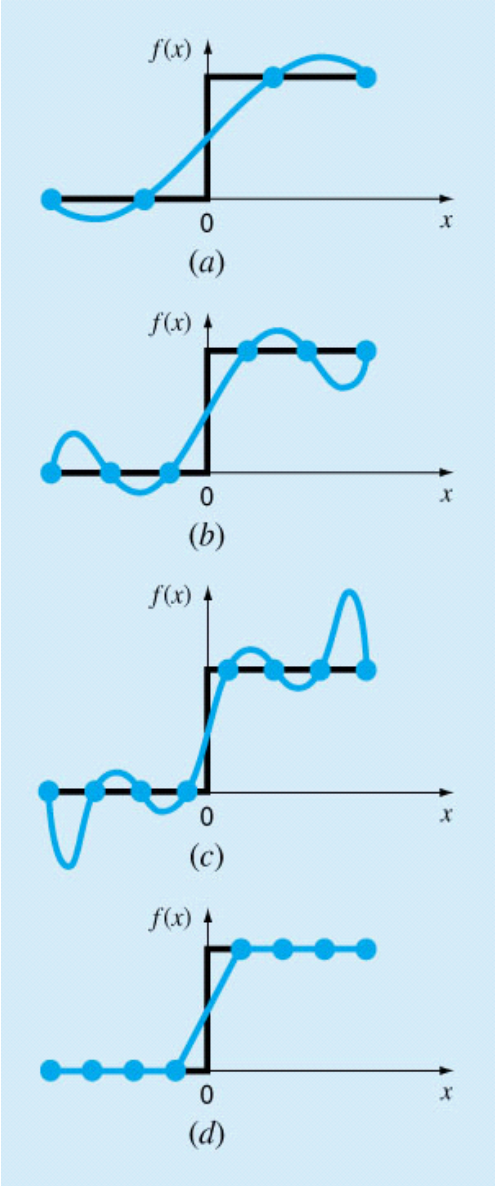
Chebyshev Minmax Property (cont.)

- Thus, Chebyshev interpolation converges in L^∞, essentially achieves minmax approximation, easy to compute.
- *Caution*
  - does *not* necessarily approximate f'
  - if $\| f^{(k)} \|_\infty$ is large then the error may be large for moderate n.
- Chebyshev polynomials do not guarantee good approximations (they are wiggly) but they guarantee accuracy bounds.
  – With other polynomials, one cannot get this error bounds.

Splines

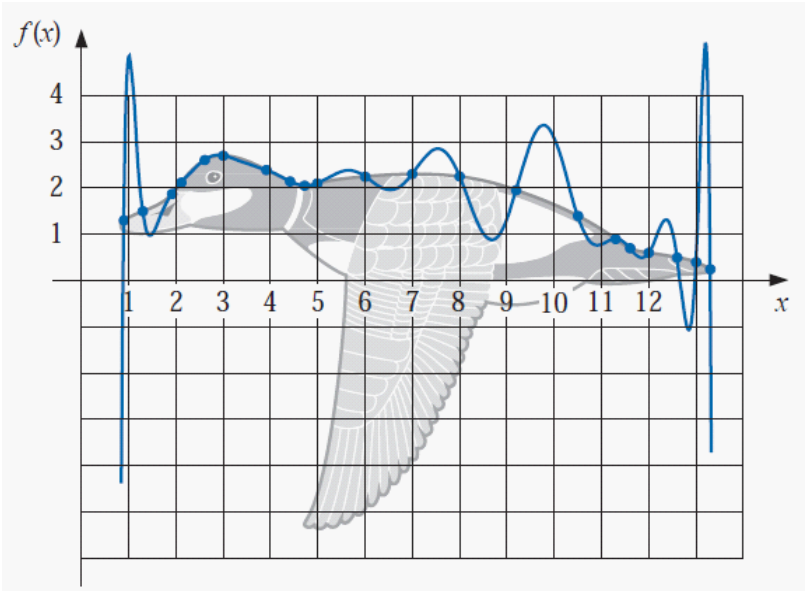
Spline Interpolation

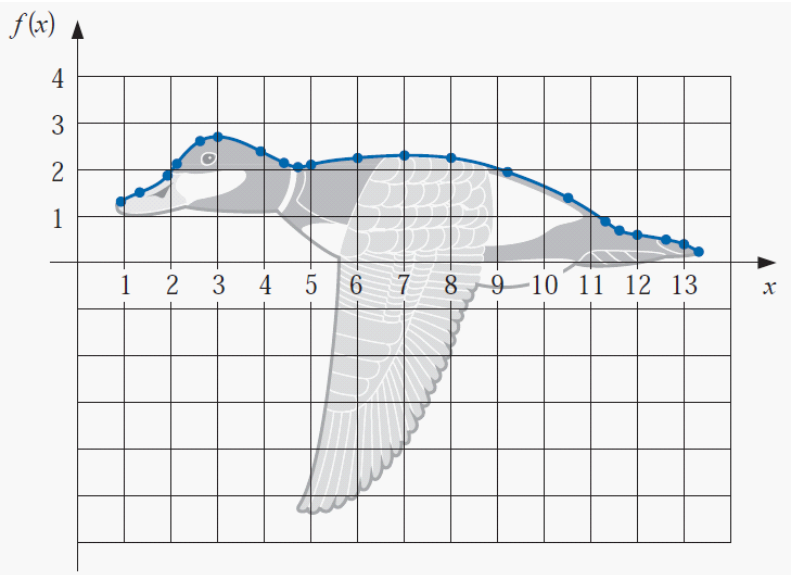
- For some cases, polynomials can lead to erroneous results because of round off error and overshoot.
- Alternative approach is to apply lower-order polynomials to subsets of data points. Such connecting polynomials are called spline functions.



Spline in economics

- Splines are implemented in all computer languages.
- Splines are the best choice in economic models with
  – borrowing constraints;
  – kinks (due to, e.g., zero lower bound on nominal interest rates);
  – few state variables.

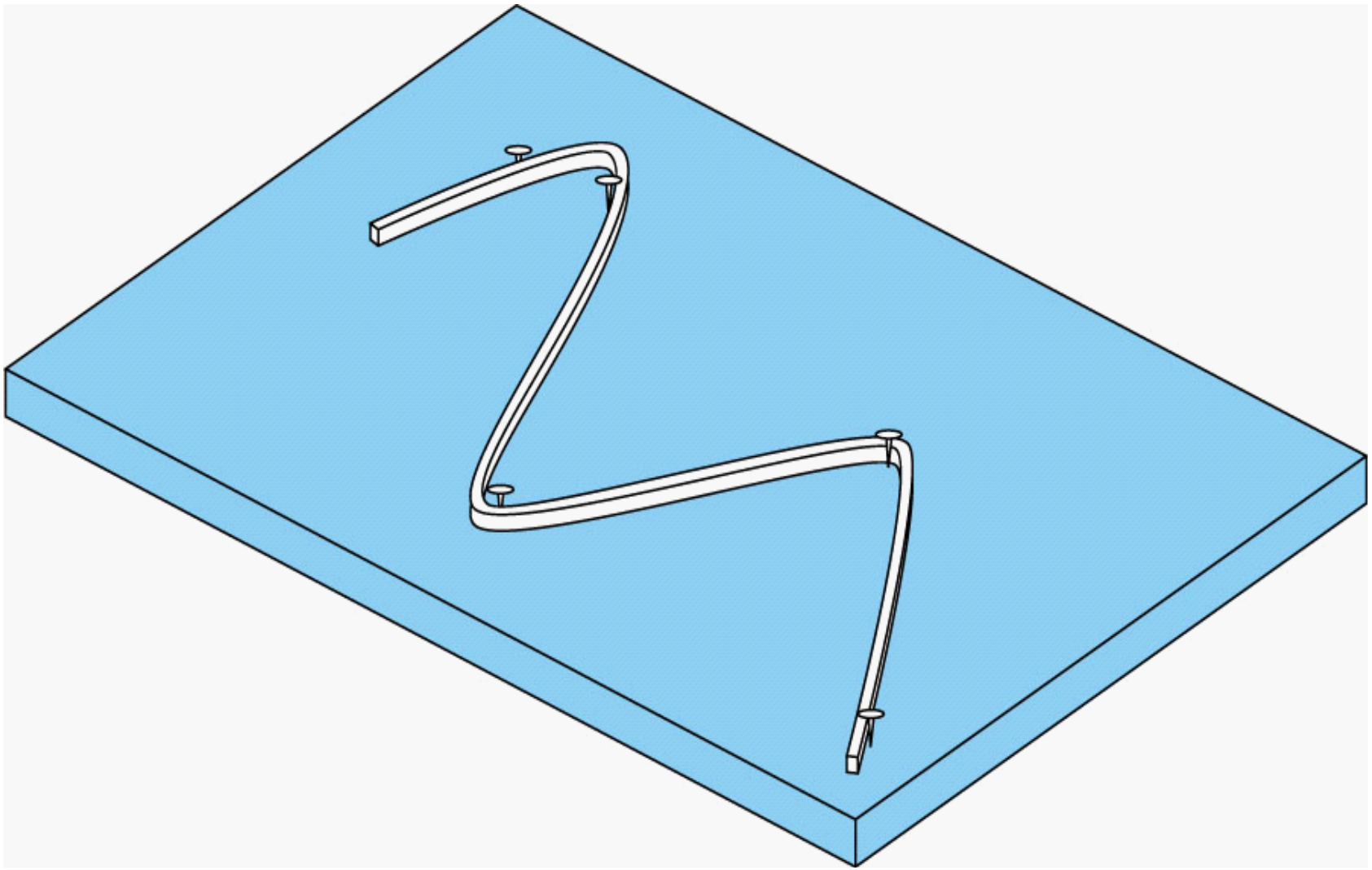




Splines

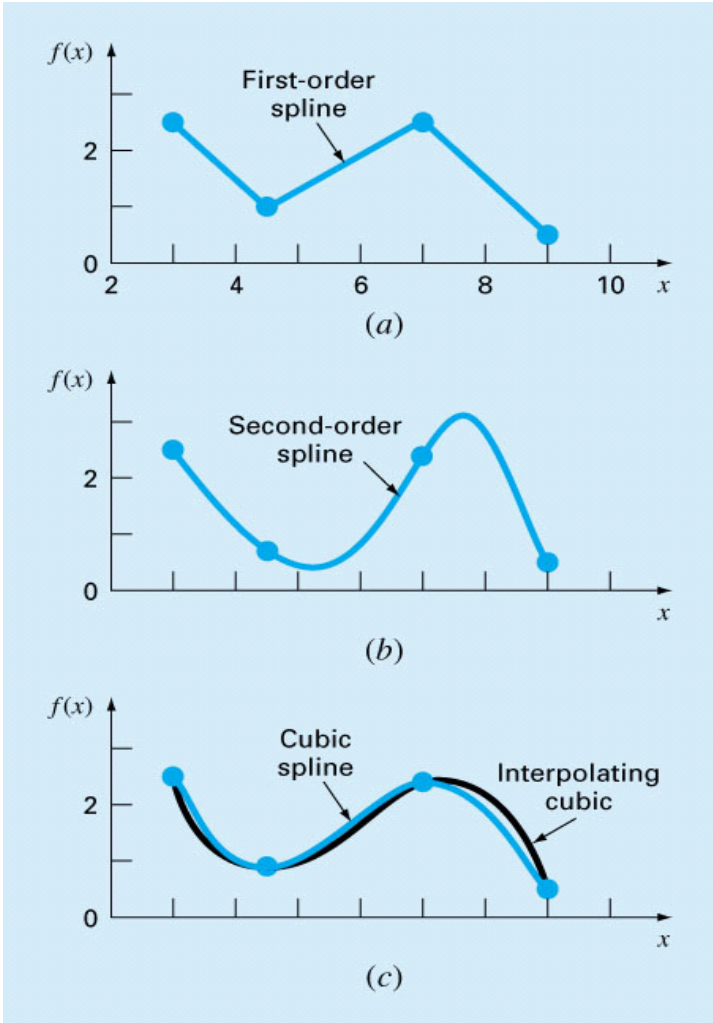
- Piecewise interpolation: does not fit a single function to all the data but separate functions.
- Splines: piecewise interpolation of a particular form.
- Namely, spline is a smooth function:
  - it is piecewise polynomial;
  - smooth where the polynomial pieces connect.

Spline Interpolation



Spline Interpolation (cont.)

- (a) Linear spline
  – Derivatives are not continuous
  – Not smooth
- (b) Quadratic spline
  – Continuous 1st derivatives
- (c) Cubic spline
  – Continuous
  – 1st & 2nd derivatives
  – Smoother



Exercise

Which of the following is a quadratic spline?

$$A(x) = \begin{cases} x^2 & x \in [-2,0] \\ x^2 + 2x & x \in [0,1] \\ x^2 + x + 1 & x \in [1,2] \end{cases}$$

$$B(x) = \begin{cases} x^2 & x \in [-2,0] \\ -x^2 & x \in [0,1] \\ 1 - 2x & x \in [1,2] \end{cases}$$

Exercise: Solution

$$A(x) = \begin{cases} x^2 & x \in [-2,0] \\ x^2 + 2x & x \in [0,1] \\ x^2 + x + 1 & x \in [1,2] \end{cases}$$

$$A_0'(x) = 2x$$

$$A_1'(x) = 2x + 2$$

At $x = 0$, $A_0'(x) \neq A_1'(x)$.

Thus $A(x)$ is not a quadratic spline

$$B(x) = \begin{cases} x^2 & x \in [-2,0] \\ -x^2 & x \in [0,1] \\ 1 - 2x & x \in [1,2] \end{cases}$$

$$B_0(0) = 0 = B_1(0)$$

$$B_1(1) = -1 = B_2(1)$$

$$B_0'(0) = 0 = B_1'(0)$$

$$B_1'(1) = -2 = B_2'(1)$$

Since B and B' are continuous for all x in [-2,2], and since B_i's are all polynomial of degree ≤ 2, B is a quadratic spline.

Definition of splines

Definition

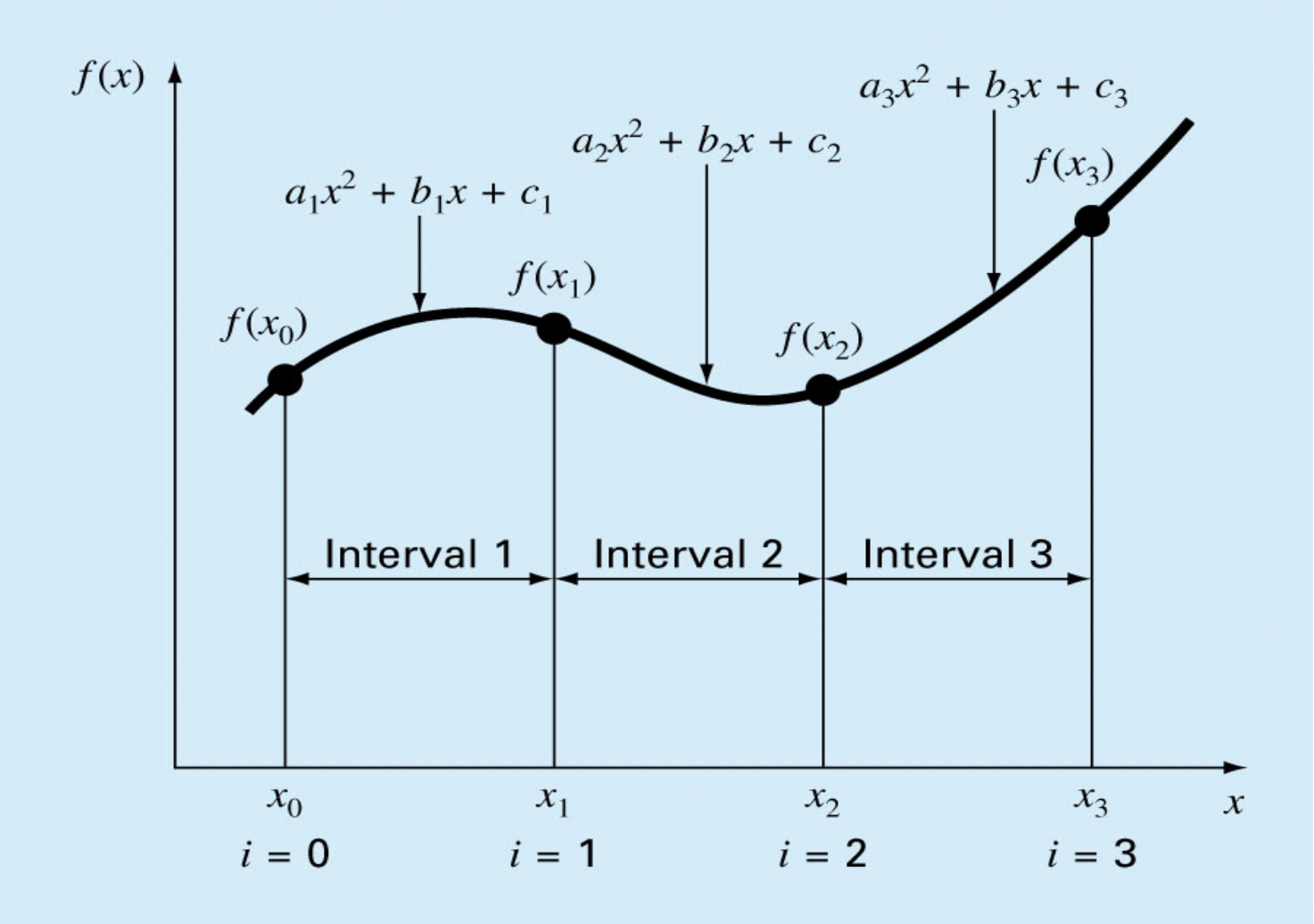
A function $s(x)$ on $[a, b]$ is a *spline of order* n iff

1. s is C^{n-2} on $[a, b]$, and
2. there is a grid of points (called nodes) $a = x_0 < x_1 < \cdots < x_m = b$ such that $s(x)$ is a polynomial of degree $n - 1$ on each subinterval $[x_i, x_{i+1}]$, $i = 0, \ldots, m - 1$.

Note: an order 2 spline is the piecewise linear interpolant.

Quadratic Splines

Spline of order 3.



Quadratic Splines (cont.)

- A quadratic spline is a continuously differentiable piecewise quadratic function (spline of order 3).
- A function s is called a spline of order 3 if
  - The domain of s is an interval $[a, b]$.
  - s and s' are continuous functions on $[a, b]$.
  - There are points x_i (called knots) such that $a = x_0 < x_1 < \ldots < x_n = b$ and s is a polynomial of degree at most 2 on each subinterval $[x_i, x_{i+1}]$.

Quadratic Interpolation (3n conditions)

1. Interpolating conditions

- On each sub interval $[x_i, x_{i+1}]$, the function $s_i(x)$ must satisfy the conditions
$$s_i(x_i) = f(x_i) \text{ and } s_i(x_{i+1}) = f(x_{i+1}).$$
- These conditions yield $2n$ equations
$$a_i x_i^2 + b_i x_i + c_i = f(x_i),$$
$$a_i x_{i+1}^2 + b_i x_{i+1} + c_i = f(x_{i+1}),$$
$$i = 0, \ldots, n-1.$$

Quadratic Interpolation (3n conditions) (cont.)

2. Continuous first derivatives

- The first derivatives at the interior knots must be equal.
- This adds $n-1$ more equations:

$$2a_i x_i + b_i = 2a_{i+1} x_i + b_{i+1}, \quad i = 1, \ldots, n-1.$$

- We now have $2n + (n-1) = 3n - 1$ equations.
- We need one more equation.

Quadratic Interpolation (3n conditions) (cont.)

3. Assume the 2nd derivatives is zero at the first point.

- This gives us the last condition as

$$2a_1 = 0 \Longrightarrow a_1 = 0$$

- With this condition selected, the first two points are connected by a straight line.
- *Note:* This is not the only possible choice or assumption we can make.

Example

Fit quadratic splines to the following data points:

i	0	1	2	3
x_i	3	4.5	7	9
$f(x_i)$	2.5	1	2.5	0.5

Example (Solution)

1. Interpolating conditions:

$$
\begin{aligned}
9a_1 &+ 3b_1 &+ c_1 &= 2.5\\
20.25a_1 &+ 4.5b_1 &+ c_1 &= 1.0\\
20.25a_2 &+ 4.5b_2 &+ c_2 &= 1.0\\
49a_2 &+ 7b_2 &+ c_2 &= 2.5\\
49a_3 &+ 7b_3 &+ c_3 &= 2.5\\
81a_3 &+ 9b_3 &+ c_3 &= 0.5
\end{aligned}
$$

2. Continuous first derivatives:

$$
\begin{aligned}
9a_1 &+ b_1 &= 9a_2 &+ b_2\\
14a_2 &+ b_2 &= 14a_3 &+ b_3
\end{aligned}
$$

3. Assume the 2nd derivatives is zero at the first point

$$a_1 = 0$$

Example (Solution) (cont.)

We can write the system of equations in matrix form as

$$\begin{bmatrix} 9 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 20.25 & 4.5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20.25 & 4.5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 49 & 7 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 49 & 7 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 81 & 9 & 1 \\ 9 & 1 & 0 & -9 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 14 & 1 & 0 & -14 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1 \\ 1 \\ 2.5 \\ 2.5 \\ 0.5 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Notice that the coefficient matrix is sparse.

Example (Solution) (cont.)

- The system of equations can be solved to yield

$$\begin{array}{lll} a_1 = 0 & b_1 = -1 & c_1 = 5.5 \\ a_1 = 0.64 & b_1 = -6.76 & c_1 = 18.46 \\ a_1 = -1.6 & b_1 = 24.6 & c_1 = -91.3 \end{array}$$

- Thus, the quadratic spline that interpolates the given points is

$$s(x) = \begin{cases} -x + 5.5 & x \in [3, 4.5] \\ 0.64x^2 - 6.76x + 18.46 & x \in [4.5, 7] \\ -1.6x^2 + 24.6x - 91.3 & x \in [7, 9] \end{cases}$$

Cubic Splines

- Lagrange data set: $\{(x_i, y_i) \mid i = 0, \cdots, n\}$.
- Nodes: The x_i are the nodes of the spline.
- Functional form: $s(x) = a_i + b_i\, x + c_i\, x^2 + d_i\, x^3$ on $[x_{i-1}, x_i]$.
- Unknowns: $4n$ unknown coefficients, a_i, b_i, c_i, d_i, $i = 1, \cdots n$.
- **Conditions:**
  - $2n$ interpolation and continuity conditions:

$$\begin{aligned} y_i &= a_i + b_i x_i + c_i x_i^2 + d_i x_i^3, \\ & \quad i = 1, ..., n \\ y_i &= a_{i+1} + b_{i+1} x_i + c_{i+1} x_i^2 + d_{i+1} x_i^3, \\ & \quad i = 0, ..., n-1 \end{aligned}$$

  - $2n-2$ conditions from C^2 at the interior: for $i = 1, \cdots n-1$,

$$\begin{aligned} b_i + 2c_i x_i + 3d_i x_i^2 &= b_{i+1} + 2c_{i+1}\, x_i + 3d_{i+1} x_i^2 \\ 2c_i + 6d_i x_i &= 2c_{i+1} + 6d_{i+1} x_i \end{aligned}$$

Cubic Splines (cont.)

- Equations (1–4) are $4n - 2$ linear equations in $4n$ unknown parameters, a, b, c, and d.
- Construct 2 side conditions
  - *natural spline:* $s'(x_0) = 0 = s'(x_n)$.
  - *Hermite spline*: $s'(x_0) = y'_0$ and $s'(x_n) = y'_n$ (assumes extra data)
- Solve system by special (sparse) methods.

Splines: Example

- Let $i = 0, 1, 2, 3$ (4 points): $\{(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)\}$.
- For example, the points are $\{(0, 0), (1, 1), (2, 4), (3, 5)\}$.
- The $2n = 6$ interpolation and continuity equations are:
  - $i = 1, 2, 3$

$$\begin{aligned} y_1 &= a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3 \\ y_2 &= a_2 + b_2 x_2 + c_2 x_2^2 + d_2 x_2^3 \\ y_3 &= a_3 + b_3 x_3 + c_3 x_3^2 + d_3 x_3^3 \end{aligned}$$

  - $i = 0, 1, 2$

$$\begin{aligned} y_0 &= a_1 + b_1 x_0 + c_1 x_0^2 + d_1 x_0^3 \\ y_1 &= a_2 + b_2 x_1 + c_2 x_1^2 + d_2 x_1^3 \\ y_2 &= a_3 + b_3 x_2 + c_3 x_2^2 + d_3 x_2^3 \end{aligned}$$

Splines: Example (cont.)

- The $2n-2$ conditions at the interior for $i = 1, 2$
  - first derivatives match

$$\begin{aligned} b_1 + 2c_1x_1 + 3d_1x_1^2 &= b_2 + 2c_2x_1 + 3d_2x_1^2 \\ b_2 + 2c_2x_2 + 3d_2x_2 &= b_3 + 2c_3x_2 + 3d_3x_2^2 \end{aligned}$$

  - second derivatives match

$$\begin{aligned} 2c_1 + 6d_1x_1 &= 2c_2 + 6d_2x_1 \\ 2c_2 + 6d_2x_2 &= 2c_3 + 6d_3x_2 \end{aligned}$$

- Terminal conditions:

$$\begin{aligned} b_1 + 2c_1x_0 + 3d_1x_0^2 &= 0 \\ b_3 + 2c_3x_3 + 3d_3x_3^2 &= 0. \end{aligned}$$

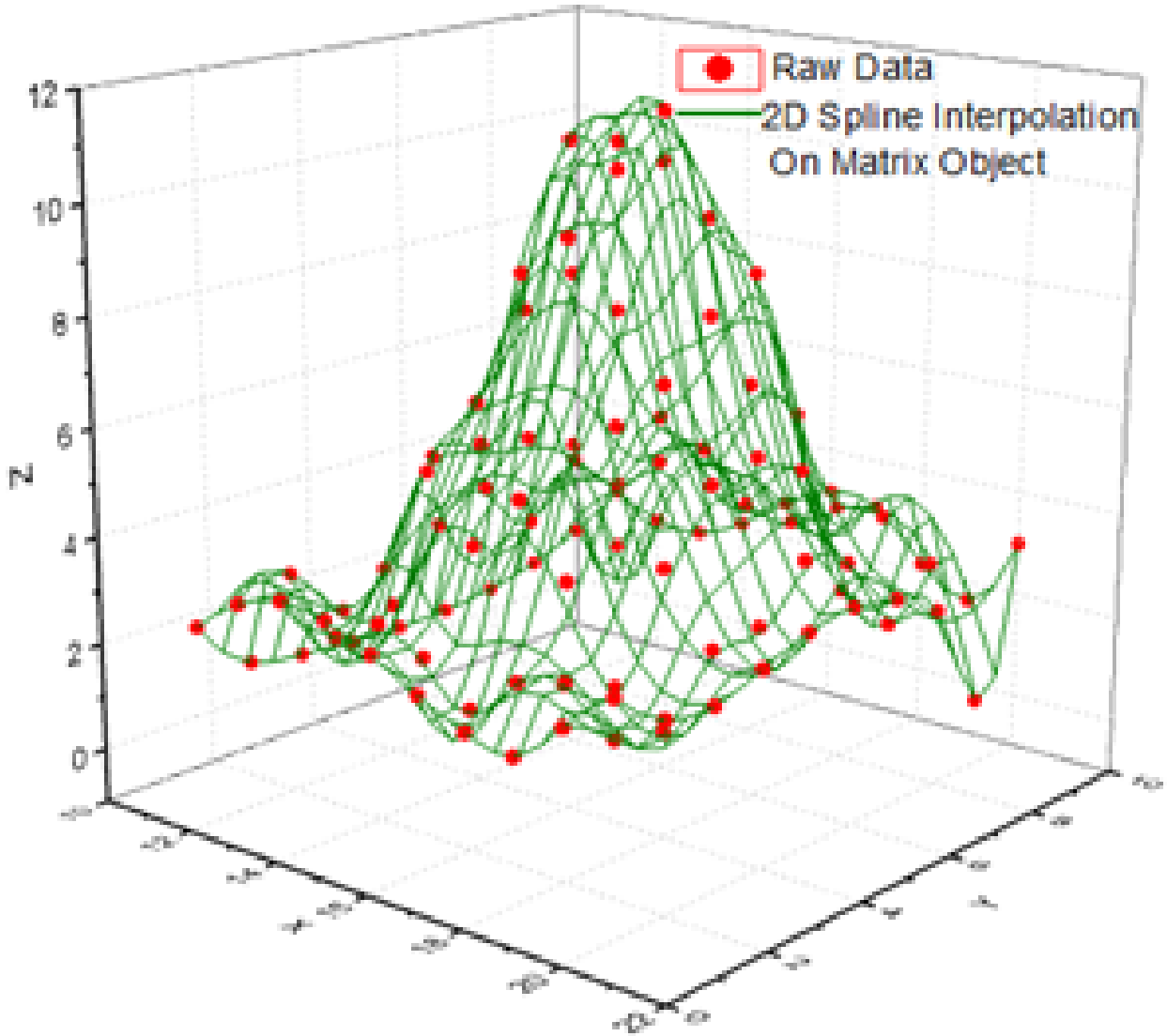
Multidimensional Methods

Multidimensional Methods

- So far, we study interpolation and approximation in one dimension.
- But the problems we study in economics have multiple dimensions.
- For example, for our growth model, we need to approximate / interpolate decision function of 2 state variables $k' = K(k, \theta)$.
- We now show how to extend our one-dimensional analysis to multiple dimension.

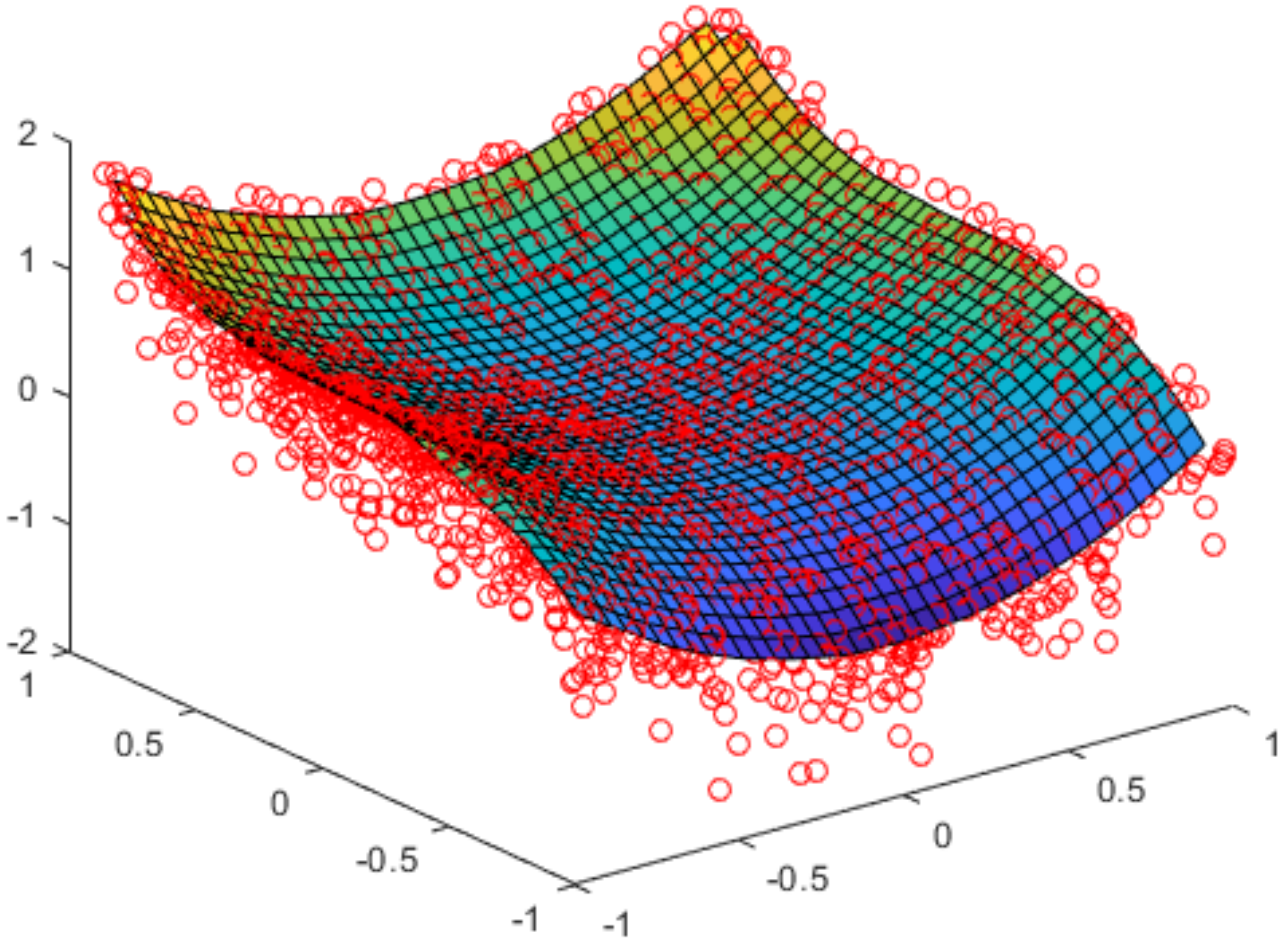
Interpolation in 2d

Interpolation in 2d passes through all grid points



Approximation in 2d

Approximation in 2d passes "the best" plane through the given points, for example, least squares



Multidimensional Methods

- Data: $D \equiv \{(x_i, z_i)\}_{i=1}^{N} \subset R^{n+m}$, where $x_i \in R^n$ and $z_i \in R^m$
- Objective: find $f : R^n \longrightarrow R^m$ such that $z_i = f(x_i)$.
- Need to choose nodes carefully.
- Task: Find combinations of interpolation nodes and spanning functions to produce a nonsingular (well-conditioned) interpolation matrix.

Multidimensional Methods (cont.)

Example that DOES NOT work

- Interpolation nodes:

$$\{P_1, P_2, P_3, P_4\} \equiv \{(1,0), (-1,0), (0,1), (0,-1)\}$$

- Use linear combinations of $\{1, x, y, xy\}$.
- Data: $z_i = f(P_i), i = 1, 2, 3, 4$.
- Interpolation form $f(x,y) = a + bx + cy + dxy$.
- For example, $P_1 = (x,y) = (1,0)$.
  $z_1 = f(1,0) = a + b \cdot 1 + c \cdot 0 + d \cdot 1 \cdot 0 = a + b$.
- Defining conditions form the singular system

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}.$$

Tensor Products

General Approach:

- If A and B are sets of functions over $x \in R^n$, $y \in R^m$, their tensor product is
$$A \otimes B = \{\varphi(x)\psi(y) \mid \varphi \in A, \psi \in B\}.$$
- Given a basis for functions of x_i, $\Phi^i = \{\varphi_k^i(x_i)\}_{k=0}^{\infty}$, the *n-fold tensor product* basis for functions of $(x_1, x_2, \ldots, x_n)$ is
$$\Phi = \left\{ \prod_{i=1}^{n} \varphi_{k_i}^i(x_i) \mid k_i = 0, 1, \cdots, \ i = 1, \ldots, n \right\}$$

Tensor Products (cont.)

Tensor products of unidimensional Chebyshev polynomial basis for the two-dimensional case

	Dimension y		
Dimension x	1	y	$2y^2 - 1$
1	1	y	$2y^2 - 1$
x	x	xy	$x\left(2y^2 - 1\right)$
$2x^2 - 1$	$2x^2 - 1$	$\left(2x^2 - 1\right) y$	$\left(2x^2 - 1\right)\left(2y^2 - 1\right)$

Tensor products (cont.)

Orthogonal polynomials and Least-square approximation

- Suppose Φ^i are orthogonal with respect to $w_i(x_i)$ over $[a_i, b_i]$
- Least squares approximation of $f(x_1, \cdots, x_n)$ in Φ is

$$\sum_{\varphi \in \Phi} \frac{\langle \varphi, f \rangle}{\langle \varphi, \varphi \rangle} \varphi,$$

where the product weighting function

$$W(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} w_i(x_i)$$

defines $\langle \cdot, \cdot \rangle$ over $D = \prod_i [a_i, b_i]$ in

$$\langle f(x), g(x) \rangle = \int_D f(x) g(x) W(x) dx.$$

Example: Second-order Chebyshev polynomial

- Data: $\left(\left(x_i, y_i\right), f\left(x_i, y_i\right)\right)$.
- $\left(x_i, y_i\right) \in \left\{(1,1), (1,-1), (-1,1), (-1,-1)\right\}$.
- $f\left(x_i, y_i\right) \in \left\{f(1,1), f(1,-1), f(-1,1), f(-1,-1)\right\}$

$$\underbrace{\begin{matrix} T_0(1)\,T_0(1) & T_1(1)\,T_0(1) & T_0(1)\,T_1(1) & T_1(1)\,T_1(1 \\ T_0(1)\,T_0(-1) & T_1(1)\,T_0(-1) & T_0(1)\,T_1(-1) & T_1(1)\,T_1(- \\ T_0(-1)\,T_0(1) & T_1(-1)\,T_0(1) & T_0(-1)\,T_1(1) & T_1(-1)\,T_1(\\ T_0(-1)\,T_0(-1) & T_1(-1)\,T_0(-1) & T_0(-1)\,T_1(-1) & T_1(-1)\,T_1(- \end{matrix}}_{A}$$

- We are to find $b = (b_1, b_2, b_3, b_4)$ by solving $Ab = \left[f(1,1), f(1,-1), f(-1,1), f(-1,-1)\right]^\top$.
- For Chebyshev polynomials, evaluate A in the corresponding T_0 and T_1 to get

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} f(1,1) \\ f(1,-1) \\ f(-1,1) \\ f(-1,-1) \end{pmatrix}.$$

Algorithm: Chebyshev Approximation Algorithm

Algorithm in $\mathbb{R}^2$

- Objective: Given $f(x, y)$ defined on $[a, b] \times [c, d]$, find the m-point degree n Chebyshev polynomial approximation $p(x, y)$
- Step 1: Compute the $m \geq n + 1$ Chebyshev interpolation nodes on $[-1, 1]$:

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

- Step 2: Adjust nodes to $[a, b]$ and $[c, d]$ intervals:

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

$$y_k = (z_k + 1)\left(\frac{d-c}{2}\right) + c, k = 1, ..., m.$$

Algorithm: Chebyshev Approximation Algorithm (cont.)

- Step 3: Evaluate f at approximation nodes:

$$w_{k,\ell} = f(x_k, y_\ell) \ , \ k = 1, \cdots, m. \ , \ \ell = 1, \cdots, m.$$

- Step 4: Compute Chebyshev coefficients, $a_{ij}, i, j = 0, \cdots, n$:

$$a_{ij} = \frac{\sum_{k=1}^{m} \sum_{\ell=1}^{m} w_{k,\ell} \, T_i(z_k) \, T_j(z_\ell)}{\left(\sum_{k=1}^{m} T_i(z_k)^2\right) \left(\sum_{\ell=1}^{m} T_j(z_\ell)^2\right)}$$

to arrive at approximation of $f(x, y)$ on $[a, b] \times [c, d]$:

$$p(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{ij} \, T_i \left(2 \frac{x - a}{b - a} - 1 \right) T_j \left(2 \frac{y - c}{d - c} - 1 \right)$$

Complete polynomials

- **Motivation:** Tensor product contains all possible tensor terms $A \otimes B = \{\varphi(x)\psi(y) \mid \varphi \in A,\ \psi \in B\}$ but not all terms are equally important.
- Taylor's theorem for $\mathbb{R}^n$ produce complete polynomials by removing higher-order less important terms

$$\begin{aligned} f(x) \doteq \quad & f(x^0) \\ & + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x^0)\ (x_i - x_i^0) \\ & + \frac{1}{2} \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \frac{\partial^2 f}{\partial x_{i_1} \partial x_{i_k}}(x_0)(x_{i_1} - x_{i_1}^0)(x_{i_k} - x_{i_k}^0)\ldots \end{aligned}$$

- For $k = 1$, Taylor's theorem for n dimensions used the linear functions

$$\mathcal{P}_1^n \equiv \{1, x_1, x_2, \cdots, x_n\}$$

- For $k = 2$, Taylor's theorem uses

$$\mathcal{P}_2^n \equiv \mathcal{P}_1^n \cup \{x_1^2, \cdots, x_n^2, x_1x_2, x_1x_3, \cdots, x_{n-1}x_n\}.$$

$\mathcal{P}_2^n$ contains some product terms, but not all; for example, $x_1x_2x_3$ is not in $\mathcal{P}_2^n$.

Tensor Products versus Complete Polynomials

Tensor products of unidimensional Chebyshev polynomial basis for 2d case

Dimension x	Dimension y		
	1	y	$2y^2-1$
1	1	y	$2y^2-1$
x	x	xy	$x\left(2y^2-1\right)$
$2x^2-1$	$2x^2-1$	$\left(2x^2-1\right)y$	$\left(2x^2-1\right)\left(2y^2-1\right)$

Complete 2d Chebyshev polynomial basis

Dimension x	Dimension y		
	1	y	$2y^2-1$
1	1	y	$2y^2-1$
x	x	xy	
$2x^2-1$	$2x^2-1$		

Complete polynomial (cont.)

- In general, the kth degree expansion uses the *complete set of polynomials of total degree k in n variables*.

$$\mathcal{P}_k^n \equiv \{x_1^{i_1} \cdots x_n^{i_n} \mid \sum_{\ell=1}^{n} i_\ell \leq k,\ 0 \leq i_1, \cdots, i_n\}$$

- Complete orthogonal basis includes only terms with total degree k or less.
- Sizes of alternative bases

degree k	$\mathcal{P}_k^n$	Tensor Prod.
2	$1 + n + n(n+1)/2$	3^n
3	$1 + n + \frac{n(n+1)}{2} + n^2 + \frac{n(n-1)(n-2)}{6}$	4^n

  - Complete polynomial bases contains fewer elements than tensor products.
  - Asymptotically, complete polynomial bases are as good as tensor products.
  - For smooth n-dimensional functions, complete polynomials are more efficient approximations

Complete polynomials (cont.)

Construction

- Compute tensor product approximation, as in Algorithm above.
- Drop terms not in complete polynomial basis (or, just compute coefficients for polynomials in complete basis).
- Complete polynomial version is faster to compute since it involves fewer terms

Multidimensional splines

- One-dimensional splines can be extended to two and higher dimensions using tensor-product approaches.
- The multi-dimensional splines are more cumbersome to describe, in particular, connecting smoothly the values on the boundaries.
- However, multi-dimensional splies are available and easy to use in many languages.

Summary

- Interpolation versus regression
  - Lagrange data uses level information only
  - Regression uses more points than coefficients
- One-dimensional problems
  - Smooth approximations
    – Orthogonal polynomial methods for nonperiodic functions
  - Less smooth approximations
    – Splines
- Multidimensional data
  - Tensor product methods have curse of dimension
  - Complete polynomials are more efficient
- Tensor product approaches are subject to severe curse of dimensionality.
- Later, we will show how to deal with that: Smolyak sparse grids, dimensionality reduction, ergodic-set methods, machine learning

Appendix: Linear Interpolation

Linear Interpolation

- Recall: interpolation is when we find a function from an n-dimensional family of functions which exactly fits n data items.
- An interpolation method starts with discrete data, and defines a continuos function.
- *Linear interpolation*: draw a straight line between two neighboring points and return the appropriate point along that line.
- Given a collection of data (x_i, y_i), $i = 1, \ldots, n$, for any interval $x \in [x_{i-1}, x_i]$, we can compute y.
- It delivers a collection of approximations for each interval.
- It can perform badly for non-linear functions.

Linear interpolation: Example

Consider two data points, (x_1, y_1) and (x_2, y_2).

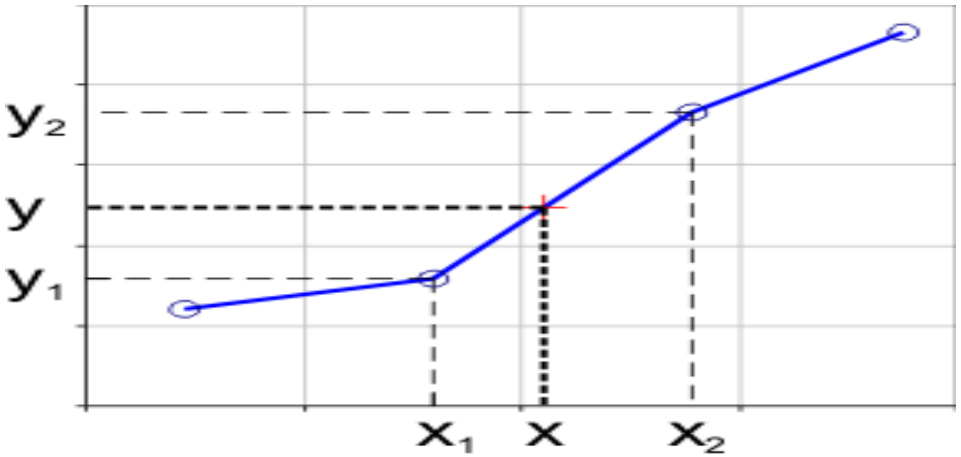
$$Slope = \frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1} \equiv b.$$

Then, $y = a + bx$, so that

$$a = y_1 - bx_1 = y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$$

$$y = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} + \frac{y_2 - y_1}{x_2 - x_1} \cdot x.$$

Linear interpolation: Example (cont.)



Appendix: Hermite Interpolation

Hermite Polynomial Interpolation

- Suppose we want to find polynomial that fits both the *slope* and the *level* of the function that we want to approximate.
- Data: $\left(x_i, y_i, y_i'\right), i = 1, .., n$.
- *Objective:* Find a polynomial of degree $2n-1$, $p(x)$, which agrees with the data, i.e.,

$$\begin{aligned} y_i &= h(x_i),\ i = 1, .., n \\ y_i' &= h'(x_i),\ i = 1, .., n, \end{aligned}$$

  where $h'(\cdot)$ is a derivative of polynomial.
- Result: If the (x_i) are distinct, there is a unique interpolating polynomial.

Hermite Polynomial Interpolation: Example

- Consider a third-degree polynomial:

$$h(x) = H_3 x^3 + H_2 x^2 + H_1 x + H_0$$

- Suppose that we want to match 2 points, (x_0, y_0) and (x_1, y_1) and that we have 2 derivatives, y_0', y_1', at the points.
- We can compute 4 coefficients, H_3, H_2, H_1, H_0 by solving:

$$\begin{aligned}
h(x_0) &= H_3 x_0^3 + H_2 x_0^2 + H_1 x_0 + H_0 = y_0, \\
h(x_1) &= H_3 x_1^3 + H_2 x_1^2 + H_1 x_1 + H_0 = y_1, \\
h'(x_0) &= 3H_3 x_0^2 + 2H_2 x_0 + H_1 = y_0', \\
h'(x_1) &= 3H_3 x_1^2 + 2H_2 x_1 + H_1 = y_1'.
\end{aligned}$$

- Cai and Judd, (2015). "Dynamic Programming with Hermite Approximation".

Appendix: Trigonometric Polynomials

Trigonometric Polynomials and Fourier Series

- $\{\cos(n\theta), \sin(m\theta)\}$ are orthogonal on $[-\pi, \pi]$.
- If f is continuous on $[-\pi, \pi]$ and $f(-\pi) = f(\pi)$, then

$$f(\theta) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(n\theta) + \sum_{n=1}^{\infty} b_n \sin(n\theta) \tag{6}$$

where the *Fourier coefficients* are

$$\begin{aligned} a_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(\theta)\, \cos(n\theta) d\theta \\ b_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(\theta) \sin(n\theta)\, d\theta, \end{aligned}$$

Trigonometric Polynomials and Fourier Series (cont.)

- A *trigonometric polynomial* is any function of the form in (6).
- Convergence is uniform for periodic functions.
- Excellent for approximating a smooth *periodic function*, i.e., $f : R \to R$ such that for some ω, $f(x) = f(x + \omega)$.
- Not good for nonperiodic functions
  - Convergence is not uniform
  - Many terms are needed

Appendix: B-splines

B-splines

- Put knots at $\{x_{-k}, \cdots, x_{-1}, x_0, \cdots, x_n\}$.
- Order 1 splines: step function interpolation spanned by

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

  - Order 2 splines: piecewise linear interpolation and are spanned by

$$B_i^1(x) = \begin{cases} 0\ , & x \le x_i \text{ or } x \ge x_{i+2}, \\ \frac{x - x_i}{x_{i+1} - x_i}\ , & x_i \le x \le x_{i+1}, \\ \frac{x_{i+2} - x}{x_{i+2} - x_{i+1}}\ , & x_{i+1} \le x \le x_{i+2}. \end{cases}$$

    The B_i^1-spline is the tent function with peak at x_{i+1} and is zero for $x \le x_i$ and $x \ge x_{i+2}$.
- Both B^0 and B^1 splines form cardinal bases for interpolation at the x_i's.
- Higher-order B-splines are defined by the recursive relation