Function Approximation

David Florian Hoyle

UCSC

[11/14] November 2014

Objective

- Obtain an approximation for $f(x)$ by another function $\widehat{f}(x)$
- **Two cases**:
  - $f(x)$ is known in all its domain, but it is very expensive to calculate it.
  - $f(x)$ is known only in a finite set of points: Interpolation.

Outline

Approximation methods

- We want to approximate $f(x) : [a, b] \to \mathbb{R}$ by a linear combination of polynomials

$$f(x) \approx \sum_{j=1}^{n} c_j \varphi_j(x)$$

where

  - $x \in [a, b]$
  - n : Degree of interpolation
  - c_j : Basis coefficients
  - $\varphi_j(x)$: Basis functions which are polynomials of degree $\leq n$.

Approximation methods: Introduction

- When trying to approximate $f(x) : C \subset \mathbb{R} \to \mathbb{R}$ by $f(x) \approx \sum_{j=1}^{n} c_j \varphi_j(x)$
- We need to rely on a **concept of distance** between $f(x)$ and $\sum_{j=1}^{n} c_j \varphi_j(x)$ at the points we choose to make the approximation.
- We are dealing with normed vector spaces of functions which can be finite or infinite dimensional.
    - The space of continuos functions in $\mathbb{R}^n$, the space of infinite sequences l^p, the space of measurable functions L^p and so on

Approximation methods: Introduction

- If we define an inner product in this spaces then we have the induced norm L^2 :

$$\|f(x)\|_{L^2} = \left(\int_a^b f(x)^2 w(x)\, dx \right)^{\frac{1}{2}}$$

  it is usefull for least squares approximation.

- Approximation theory, is based on uniform convergence of $f(x)$ to $\sum_{j=1}^{n} c_j \varphi_j(x)$. In this case, we use the supreme norm

$$\|f(x)\|_{\infty} = \max_{x \in [a,b]} |f(x)|$$

Approximation methods: Local vs Global

- Local approximations are based on the **Taylor approximation theorem.**
- Global approximations are based on the **Weierstrass approximation theorem.**

Approximation methods: Weierstrass approximation theorem

Theorem

Let $f(x)$ be a continuous function on $[a,b]$ (i.e., $f \in C[a,b]$), then for all $\epsilon > 0$, there exists a sequence of polynomials $p_n(x)$ of degree $\leq n$ that converges uniformly to $f(x)$ on $[a,b]$. That is, $\forall \epsilon > 0$, $\exists N \in \mathbb{N}$ and polynomials $p_n(x)$ such that

$$\forall n \geq N, \forall x \in [a,b] \text{ then } \|f(x) - p_n(x)\|_\infty \leq \varepsilon$$

- Where $\|\cdot\|_\infty$ is the sup norm or L^∞ norm:

$$\|f(x) - p_n(x)\|_\infty = \max_{x \in [a,b]} |f(x) - p_n(x)|$$

- In other words,

$$\lim_{n \longrightarrow \infty} p_n(x) = f(x) \quad \text{for all } x \in [a,b]$$

Approximation methods: Weierstrass approximation theorem

- **Another version**

Theorem

if $f \in C[a, b]$,*then for all* $\epsilon > 0$ *there exists a polynomial* $p(x)$ *such that*

$$\|f(x) - p_n(x)\|_\infty \leq \varepsilon \quad \forall x \in [a, b]$$

- This theorem tell us that any continuous function on a compact interval can be approximated arbitrarily well by polynomials of any degree.

Approximation methods: Weierstrass approximation theorem

- A constructive proof of the theorem is based on the Bernstein polynomials defined on $[0,1]$

$$p_n(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right)\left[\binom{n}{k} x^k (1-x)^{n-k}\right]$$

  such that

$$\lim_{n\longrightarrow\infty} p_n(x) = f(x) \quad \text{for all } x \in [0,1]$$

  uniformly.

- **Weierstrass theorem is conceptually valuable but it is not practical.** From a computational point perspective it is not efficient to work with Bernstein polynomials.
  - Bernstein polynomials converge very slowly!

Approximation methods: Minimax approximation

- Also called **the Best polynomial approximation**
- We want to find the polynomial $p_n(x)$ of degree $\leq n$ that best approximates a function $f(x)$ uniformly with $f(x) \in C[a,b]$:
  - For this, we are going to look for **the infimun of the distance between f and all possible degree $\leq n$ polynomial approximations**
  - Recall that the uniform error term (i.e., using the L^∞ norm) is

$$\|f(x) - p_n(x)\|_\infty = \max_{x\in[a,b]} |f(x) - p_n(x)|$$

- Define $d_n(x)$ as **the infimun of the distance between f and all possible** $p_n(x)$ approximations of f

$$\begin{aligned} d_n(f) &= \inf_{p_n} \|f - p_n\|_\infty \\ &= \inf_{p_n} \left(\max_{x\in[a,b]} |f(x) - p_n(x)| \right) \end{aligned}$$

Approximation methods: Minimax approximation

- Let $p_n^*(x)$ be the polynomial for which the infimum is obtained

$$d_n(f) = \|f - p_n^*\|_\infty$$

- It can be shown that p_n^* **exists, it is unique** and it is characterized by a property called the **equioscillation property.**
  - Algorithms to compute p_n^* are difficult/complex/not efficients.
  - Example: Remez algorithm.
- **Standard solution: Chebyshev least squares polynomial approximation closely approximate the minmax or best polynomial.**
  - This strategy is called: **Near minmax approximation.**

Approximation methods: Minimax approximation

- Existence and uniqueness of the minimax polynomial:

Theorem

Let $f \in C[a,b]$. Then for any $n \in \mathbb{N}$ there exists a unique $p_n^*(x)$ that minimizes $\|f - p_n\|_\infty$ among all polynomials of degree $\leq n$.

- Sadly, the proof is not constructive so we have to rely on an algorithm that help us compute $p_n^*(x)$.

Approximation methods: Minimax approximation

- But we can **characterize** the error generated by the minmax polynomial, $p_n^*(x)$ in terms of its **oscillation property**:

Theorem

Let $f \in C[a,b]$. The polynomial $p_n^*(x)$ is the minimax polynomial of degree n that approximates $f(x)$ in $[a,b]$, if and only if, the error $f(x) - p_n^*(x)$, assumes the values $\pm \|f - p_n^*\|_\infty$ with an alternating change of sign in at least $n+2$ points in $[a,b]$. That is, in $a \leq x_0 < x_1 < \ldots < x_{n+1} \leq b$ we have the following

$$f(x) - p_n^*(x) = (-1)^j \|f - p_n^*\|_\infty \quad \text{for } j = 0, 1, 2, \ldots n+1$$

Approximation methods: Minimax approximation

- The theorem give us the desired shape of the error when we want L^∞ approximation (the best approximation).
- For example it says that the maximum error of a cubic approximation should be achieved at least five times and that the sign of the error should alternate between these points.

Approximation methods: Minimax approximation

- **Example 1**: Consider the function $f(x) = x^2$ in the interval $[0, 2]$. Find the minimax linear approximation $(n = 1)$ of $f(x)$.
- Consider the approximating polynomial $p_1(x) = ax + b$ and the errors

$$e(x) = \left| x^2 - ax - b \right|$$

- Notice that the function $e(x) = x^2 - ax - b$ has a maximun when $2x - a = 0$, that is when $x = \frac{a}{2}$.
  - $x = \frac{a}{2}$ belongs to the interval $[0, 2]$ when $0 \leq a \leq 4$.
- Evaluating $e\left(\frac{a}{2}\right) = \left(\frac{a}{2}\right)^2 - a\left(\frac{a}{2}\right) - b = \frac{a^2}{4} - \frac{a^2}{2} - b = -\frac{a^2}{4} - b$
- According to the oscillation property we need **two more extremun points**:
  - Let's take $x = 0$ and $x = 2$. Thus $e(2) = 4 - 2a - b$ and $e(0) = -b$.

Approximation methods: Minimax approximation

- **Example 1 (cont.)**:
- According to the oscillation property we must have $h(0) = -h(\frac{a}{2}) = h(2)$ which implies that

$$
\begin{aligned}
h(0) &= h(2) \\
-b &= 4 - 2a - b \\
a &= 2
\end{aligned}
$$

and

$$
\begin{aligned}
h(0) &= -h\left(\frac{a}{2}\right) \\
-b &= \frac{a^2}{4} + b
\end{aligned}
$$

but we already know that $a = 2$ thus

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

Approximation methods: Minimax approximation

- **Example 1 (cont.)**:
- Then , the approximating polynomial is

$$p_1^*(x) = 2x - \frac{1}{2}$$

  and the maximun error

$$\max_{[0,2]} \left| x^2 - 2x - \frac{1}{2} \right| = \frac{1}{2}$$

- If you try to find the quadratic minmax approximation to $f(x) = x^2$, you will notice that things start to get complicated.
- **There is no general characterization-based algorithm to compute the minimax polynomial approximation.**
- **The Remez algorithm is based on known optimal approximation for certain $f(x)$.**

Approximation methods: Minimax approximation

- **Example 1 (cont.)**:Oscillating property of the error for a minmax approximation $e(x) = x^2 - 2x - \frac{1}{2}$ for $x \in [0, 2]$

v

Approximation methods: Least squares

- **Some basic concepts**:
  - **Weighting function**: $w(x)$ on $[a, b]$ is any function that is positive and has a finite integral over $[a, b]$,that is $\int_a^b w(x)\,dx < \infty$
  - **Inner product relative to $w(x)$**: Let $f, g \in \mathbb{C}[a, b]$ then the inner product is

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

  - The inner product induces a norm given by:

$$\|f(x)\|_{L^2} = \left(\int_a^b f(x)^2\, w(x)\, dx \right)^{\frac{1}{2}}$$

Approximation methods: Least squares

- Assume a **general family of polynomials** $\{\varphi_j(x)\}_{j=0}^{n}$ that consititute a basis for $C(X)$.
- Assume that $f(x)$ is defined on $[a,b]$. We want to approximate f by a linear combination of polynomials: $p_n(x) = \sum_{j=0}^{n} c_j \varphi_j(x)$
- Define the error as

$$E(x) = f(x) - p_n(x)$$

- The **least squares approximation problem is to find the polynomial of degree** $\leq n$ that is closest to $f(x)$ in the L^2-norm among all the polynomials of degree $\leq n$.

$$\min_{\widehat{f}} \|E(x)\|_{L^2} = \min_{\widehat{f}} \left(\int_a^b E(x)^2 w(x)\, dx \right)^{\frac{1}{2}}$$

Approximation methods: Least squares

- Notice that minimizing the L^2-distance between f and $p_n(x)$ (i.e., $\|E(x)\|_{L^2}$) is equivalent to minimizing the square of the L^2-distance, given by $\|E(x)\|_{L^2}^2$:

$$\min_{\widehat{f}} \|E(x)\|_{L^2}^2 = \min_{\widehat{f}} \int_a^b E(x)^2 w(x)\, dx$$
$$= \min_{\widehat{f}} \int_a^b (f(x) - p_n(x))^2 w(x)\, dx$$

- The above problem is the continuos least square problem

Approximation methods: Least squares

- Since $p_n(x) = \sum_{j=0}^{n} c_j \varphi_j(x)$ then solve the following problem

$$\min_{\{c_j\}_{j=0}^{n}} \int_a^b w(x) \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right)^2 dx$$

- FOC's w.r.t c_k :

$$2 \int_a^b w(x) \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right) \varphi_k(x)\, dx = 0 \quad \forall k = 0, 1, 2, \ldots n$$

Approximation methods: Least squares

- The normal equations are given by

$$\sum_{j=0}^{n} c_j \int_a^b \varphi_k(x)\, \varphi_j(x)\, w(x)\, dx = \int_a^b f(x)\, \varphi_k(x)\, w(x)\, dx \quad \forall k = 0, 1, 2, \ldots n$$

- Or in terms of inner product

$$\sum_{j=0}^{n} c_j \left\langle \varphi_k, \varphi_j \right\rangle = \left\langle f, \varphi_k \right\rangle \quad \forall k = 0, 1, 2, \ldots n$$

Approximation methods: Least squares

- More explicitely, the normal equations are a system of n linear equations in n unknowns:

$$
\begin{aligned}
c_0 \langle \varphi_0, \varphi_0 \rangle + c_1 \langle \varphi_0, \varphi_1 \rangle + c_2 \langle \varphi_0, \varphi_1 \rangle + \ldots + c_n \langle \varphi_0, \varphi_n \rangle &= \langle f, \varphi_0 \rangle \\
c_0 \langle \varphi_1, \varphi_0 \rangle + c_1 \langle \varphi_1, \varphi_1 \rangle + c_2 \langle \varphi_1, \varphi_2 \rangle + \ldots + c_n \langle \varphi_1, \varphi_n \rangle &= \langle f, \varphi_1 \rangle \\
&\ldots \\
c_0 \langle \varphi_n, \varphi_0 \rangle + c_1 \langle \varphi_n, \varphi_1 \rangle + c_2 \langle \varphi_n, \varphi_1 \rangle + \ldots + c_n \langle \varphi_n, \varphi_n \rangle &= \langle f, \varphi_n \rangle
\end{aligned}
$$

Approximation methods: Least squares

- In matrix form

$$\begin{bmatrix} \langle \varphi_0, \varphi_0 \rangle & \langle \varphi_0, \varphi_1 \rangle & \dots & \langle \varphi_0, \varphi_n \rangle \\ \langle \varphi_1, \varphi_0 \rangle & \langle \varphi_1, \varphi_1 \rangle & \dots & \langle \varphi_1, \varphi_n \rangle \\ \dots & \dots & \dots & \dots \\ \langle \varphi_n, \varphi_0 \rangle & \langle \varphi_n, \varphi_1 \rangle & \dots & \langle \varphi_n, \varphi_n \rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle f, \varphi_0 \rangle \\ \langle f, \varphi_1 \rangle \\ \dots \\ \langle f, \varphi_n \rangle \end{bmatrix}$$

- In more compact form

$$Hc = b$$

Approximation methods: Least squares

- **Candidates for** $\varphi_j(x)$:
- **Monomials** constitute a basis for $\mathbb{C}(X)$:

$$1, x, x^2, x^3, ..., x^n$$

- **Orthogonal polynomials** relative to $w(x)$:

$$\int_a^b \varphi_k(x)\,\varphi_j(x)\,w(x)\,dx = 0 \quad \text{for } k \neq j$$

Approximation methods: Least squares

- If we use **Monomials** the approximation for $f(x)$ can be written as

$$p_n(x) = \sum_{j=0}^{n} c_j x^j$$

- The least squares problem is

$$\min_{\{c_j\}_{j=0}^{n}} \int_a^b \left(f(x) - \sum_{j=0}^{n} c_j x^j \right)^2 dx$$

- The FOC w.r.t c_k:

$$-2\int_a^b f(x)\, x^k dx + 2\int_a^b \left(\sum_{j=0}^{n} c_j x^j \right) x^k dx = 0 \quad \forall k = 0, 1, 2, \ldots n$$

Approximation methods: Least squares

- FOC's are expressed as

$$\int_a^b \left(\sum_{j=0}^{n} c_j x^{j+k} \right) dx = \int_a^b f(x)\, x^k dx \quad \forall k = 0, 1, 2, \ldots n$$

- Where $\int_a^b \left(\sum_{j=0}^{n} c_j x^{j+k} \right) dx = \sum_{j=0}^{n} c_j \int_a^b x^{j+k} dx$ and $\int_a^b x^{j+k} dx = \left| \frac{x^{j+k+1}}{j+k+1} \right|_a^b = \frac{b^{j+k+1} - a^{j+k+1}}{j+k+1}$ thus,

$$\sum_{j=0}^{n} \left(\frac{b^{j+k+1} - a^{j+k+1}}{j+k+1} \right) c_j = \int_a^b f(x)\, x^k dx \quad \forall k = 0, 1, 2, \ldots n$$

Approximation methods: Least squares

- The above is a linear system of equations with $n+1$ unknowks ($\{c_j\}_{j=0}^{n}$) and $n+1$ equations.
- If we assume that $[a, b] = [0, 1]$ then the system of equations to determine each of the $\{c_j\}_{j=0}^{n}$ becomes

$$\sum_{j=0}^{n} \left(\frac{1}{j+k+1} \right) c_j = \int_{0}^{1} f(x)\, x^k dx \quad \forall k = 0, 1, 2, \ldots n$$

Approximation methods: Least squares

- Explicitely the system is given by

$$\left(\frac{1}{k+1}\right)c_0+\left(\frac{1}{k+2}\right)c_1+\left(\frac{1}{k+3}\right)c_3+\ldots+\left(\frac{1}{k+n+1}\right)c_n$$
$$=\int_0^1 f\left(x\right)x^k dx \quad \forall k=0,1,2,\ldots n$$

Approximation methods: Least squares

for $k = 0$

$$c_0 + \left(\frac{1}{2}\right) c_1 + \left(\frac{1}{3}\right) c_3 + ... + \left(\frac{1}{n+1}\right) c_n = \int_0^1 f(x)\, dx$$

for $k = 1$

$$\left(\frac{1}{2}\right) c_0 + \left(\frac{1}{3}\right) c_1 + \left(\frac{1}{4}\right) c_3 + ... + \left(\frac{1}{n+2}\right) c_n = \int_0^1 f(x)\, xdx$$

for $k = 2$

$$\left(\frac{1}{3}\right) c_0 + \left(\frac{1}{4}\right) c_1 + \left(\frac{1}{5}\right) c_3 + ... + \left(\frac{1}{n+3}\right) c_n = \int_0^1 f(x)\, x^2 dx$$

for $k = n$

$$\left(\frac{1}{n+1}\right) c_0 + \left(\frac{1}{n+2}\right) c_1 + \left(\frac{1}{n+3}\right) c_3 + ... + \left(\frac{1}{n+n+1}\right) c_n = \int_0^1 f(x)\, x^n dx$$

Approximation methods: Least squares

- In matrix form, the system canbe written as

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+1} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+2} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n+3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{1}{n+1} & \frac{1}{n+2} & \frac{1}{n+3} & \cdots & \frac{1}{n+n+1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_n \end{bmatrix} = \begin{bmatrix} \int_0^1 f(x)\,dx \\ \int_0^1 f(x)\,xdx \\ \int_0^1 f(x)\,x^2 dx \\ \cdots \\ \int_0^1 f(x)\,x^n dx \end{bmatrix}$$

or

$$Hc = b$$

- H is a **HILBERT MATRIX**

Approximation methods: Least squares

- Problems with linear systems of equations with Hilbert Matrices: $Hc = b$
  - **H** is an ill-conditioned matrix: Increasing rounding errors as we increase n.
  - Condition number increases as n increases.

Approximation methods: Least squares

- If the polynomial family $\left\{\varphi_j\left(x\right)\right\}_{j=0}^{n}$ is orthogonal, that is $\langle\varphi_n,\varphi_m\rangle = 0$ for $n \neq m$ then the system becomes

$$\begin{bmatrix} \langle\varphi_0,\varphi_0\rangle & 0 & 0 & \dots & 0 \\ 0 & \langle\varphi_1,\varphi_1\rangle & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \langle\varphi_n,\varphi_n\rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle f,\varphi_0\rangle \\ \langle f,\varphi_1\rangle \\ \dots \\ \langle f,\varphi_n\rangle \end{bmatrix}$$

- In this case, the coefficients are given by

$$c_k = \frac{\langle f,\varphi_k\rangle}{\langle\varphi_k,\varphi_k\rangle} \quad \forall k = 0, 1, 2, \dots n$$

- Thus

$$p_n\left(x\right) = \sum_{j=0}^{n} \frac{\langle f,\varphi_k\rangle}{\langle\varphi_k,\varphi_k\rangle}\varphi_j\left(x\right)$$

Approximation methods: Least squares

- Computations for finding the coefficients $\{c_k\}_{k=0}^{n}$ are easy to perform when using a family of orthogonal polynomials to approximate a function.
- A family of orthogonal polynomials $\{\varphi_j(x)\}_{j=0}^{n}$ have always a recursive representation which make computations even faster.

Orthogonal polynomials: Most common families

- There are many families of orthogonal polynomials that are basis for function spaces:
  - Chebyshev: For $x \in [-1, 1]$
  $$T_n(x) = \cos(n \arccos x)$$
  - Legendre: For $x \in [-1, 1]$
  $$P_n(x) = \frac{(-1)^n}{2^n n!} \frac{d^n}{dx^n} [\left(1 - x^2\right)^n]$$
  - Laguerre: For $x \in [0, \infty]$
  $$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} \left(x^n e^{-x}\right)$$
  - Hermite: For $x \in [-\infty, \infty]$
  $$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} \left(e^{-x^2}\right)$$
- Most of this polynomials come from the soultion of "important" difference equations.

Orthogonal polynomials: Most common families

`go to inneficient matlab script!`

`orthogonal_families.m`

Orthogonal polynomials: Recursive representation

Theorem

Let $\{\varphi_n(x)\}_{n=0}^{\infty}$ *be an orthogonal family of monic polynomials on* $[a, b]$ *relative to an inner product* $\langle \cdot, \cdot \rangle$,
$\varphi_0(x) = 1, \varphi_1(x) = x - \frac{\langle x\varphi_0(x), \varphi_0(x)\rangle}{\langle \varphi_0(x), \varphi_0(x)\rangle}$. *Then* $\{\varphi_n(x)\}_{n=0}^{\infty}$ *satisfies the recursive scheme*

$$\varphi_{n+1}(x) = (x - \delta_n)\varphi_n(x) - \gamma_n \varphi_{n-1}(x)$$

where

$$\delta_n = \frac{\langle x\varphi_n, \varphi_n\rangle}{\langle \varphi_n, \varphi_n\rangle} \quad \text{and} \quad \gamma_n = \frac{\langle \varphi_n, \varphi_n\rangle}{\langle \varphi_{n-1}, \varphi_{n-1}\rangle}$$

- The above theorem is the so colled Gram-Schimdt process to find an orthogonal family of **monic** polynomials.

Orthogonal polynomials: Recursive representation

- A Chebyshev polynomial is given by

$$T_i(x) = \cos(i \arccos x) \quad \text{for } i : 0, 1, 2, ..., n$$

  and it is defined for $x \in [-1, 1]$.

- Chebyshev polynomials have the follwoing **recursive representation**:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x) \quad \text{for } i = 2, 3, ..., n$$

Orthogonal polynomials: Recursive representation

- **Recursive representation of Chebyshev polinomials:** $T_i(x)$ can be wrriten as

$$T_i(x) = \cos(i\theta) \quad \text{for } i : 1, 2, ..., n$$

where $\theta = \arccos x$.

- Using trigonometric identities we have

$$T_{i+1}(x) = \cos((i+1)\theta) = \cos(i\theta)\cos(\theta) - \sin(i\theta)\sin(\theta)$$
$$T_{i-1}(x) = \cos((i-1)\theta) = \cos(i\theta)\cos(\theta) + \sin(i\theta)\sin(\theta)$$

thus

$$\begin{aligned} T_{i+1}(x) &= \cos(i\theta)\cos(\theta) - T_{i-1}(x) + \cos(i\theta)\cos(\theta) \\ &= 2(\cos(i\theta)\cos(\theta)) - T_{i-1}(x) \end{aligned}$$

- Notice that $\cos(\theta) = \cos(\arccos x) \Longrightarrow \cos(\theta) = x$ and that $\cos(i\theta) = T_i(x)$. Therefore

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x) \quad \text{for } i = 2, 3, ..., n$$

Orthogonal polynomials: Recursive representation

- **Recursive representation of Chebyshev polinomials (cont.):**
- The first five Chebyshev polynomials are

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_2(x) = 2xT_1(x) - T_0(x) = 2x^2 - 1$$

$$\begin{aligned} T_3(x) &= 2xT_2(x) - T_1(x) = 2x\left(2x^2 - 1\right) - x \\ &= 4x^3 - 3x \end{aligned}$$

$$\begin{aligned} T_4(x) &= 2xT_3(x) - T_2(x) = 2x\left(4x^3 - 3x\right) - \left(2x^2 - 1\right) \\ &= 8x^4 - 8x^2 + 1 \end{aligned}$$

- Chebyshev polynomials are polynomials with leading coefficient of 2^{n-1}

Orthogonal polynomials: Recursive representation

- **Recursive representation of Chebyshev monic polinomials (cont.):**
- The monic Chebyshev polynomial is defined as

$$\widetilde{T}_n(x) = \frac{\cos(n \arccos x)}{2^{n-1}} \quad \text{for } n > 1$$

- Using the general recursive formula for orthogonal monic polynomials we obtain the recursion shceme for the monic Chebyshev polynomial

$$\widetilde{T}_0(x) = 0$$
$$\widetilde{T}_1(x) = x$$

$$\widetilde{T}_2(x) = x\widetilde{T}_1(x) - \frac{1}{2}\widetilde{T}_0(x)$$

and

$$\widetilde{T}_{n+1}(x) = x\widetilde{T}_n(x) - \frac{1}{4}\widetilde{T}_{n-1}(x) \quad \text{for } n \geq 2$$

Orthogonal polynomials: Recursive representation

- **Going from the monic recursive representation to the original Chebyshev representation:**

$$\frac{2^n}{2^n}\widetilde{T}_{n+1}(x) = x\frac{2^{n-1}}{2^{n-1}}\widetilde{T}_n(x) - \frac{1}{4}\frac{2^{n-2}}{2^{n-2}}\widetilde{T}_{n-1}(x) \quad \text{for } n \geq 2$$

$$\frac{1}{2^n}T_{n+1}(x) = x\frac{1}{2^{n-1}}T_n(x) - \frac{1}{4}\frac{1}{2^{n-2}}T_{n-1}(x)$$

$$\begin{aligned} T_{n+1}(x) &= x\frac{2^n}{2^{n-1}}T_n(x) - \frac{1}{4}\frac{2^n}{2^{n-2}}T_{n-1}(x) \\ &= 2xT_n(x) - \frac{1}{4}2^2 T_{n-1}(x) \\ &= 2xT_n(x) - T_{n-1}(x) \end{aligned}$$

Orthogonal polynomials

- **The weigthing function** for a Chebyshev polynomial must satisfy the orthogonality condition

$$\int_{-1}^{1} T_i(x)\, T_j(x)\, w(x)\, dx = \begin{cases} 0 & \text{for } i \neq j \\ C_j & \text{for } i = j \end{cases}$$

  where C_j is a constant
- Integrating by substitution and solving for $w(x)$ yields

$$w(x) = \frac{1}{\sqrt{1-x^2}}$$

Orthogonal polynomials

- **Orthogonality property of Chebyshev polynomials**:

$$\int_{-1}^{1} T_i(x)\, T_j(x)\, w(x)\, dx = \begin{cases} 0 & \text{for } i \neq j \\ \frac{\pi}{2} & \text{for } i = j \neq 0 \\ \pi & \text{for } i = j = 0 \end{cases}$$

Orthogonal polynomials: Chebyshev least squares

- Approximate $f \in \mathbb{C}([-1,1])$ with Chebyshev polinomials.
- Coefficients of the continuous least squares are given by

$$c_k = \frac{\langle f, T_k \rangle}{\langle T_k, T_k \rangle} \quad \forall k = 0, 1, 2, \ldots n$$

$$= \frac{\int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} dx}{\int_{-1}^{1} \frac{T_k(x)^2}{\sqrt{1-x^2}} dx}$$

where $T_k(x) = \cos(k \arccos x)$ or it is given by its recursive version.

- Notice that the denominator has two cases: For $k = 0 \rightarrow \int_{-1}^{1} \frac{T_0(x)^2}{\sqrt{1-x^2}} dx = \pi$ and for $k = 1, 2, \ldots, n \rightarrow \int_{-1}^{1} \frac{T_k(x)^2}{\sqrt{1-x^2}} dx = \frac{\pi}{2}$.

Orthogonal polynomials: Chebyshev least squares

- Therefore

$$c_0 = \frac{1}{\pi} \int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx$$

and

$$c_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)\, T_k(x)}{\sqrt{1-x^2}} dx \quad \forall k = 1, 2, \ldots n$$

- The Chebyshev polynomial approximation is

$$T_n(x) = c_0 + \sum_{k=1}^{n} c_k T_k(x)$$

Orthogonal polynomials: Chebyshev least squares

- Near minimax property of Chebyshev least squares:

Theorem

Let $p_n^*(x)$ be the minmax polynomial approximation to $f \in C[-1,1]$. If $T_n^*(x)$ is the nth degree Chebyshev least square approximation to $f \in C[-1,1]$ then

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

and

$$\lim_{n\to\infty} \|f - T_n^*\|_\infty = 0 \quad \textit{uniformly}$$

- Chebyshev least squares approximation is nearly the same as the minmax polynomial approximation and that as $n \to \infty$, then T_n^* converges to f uniformly.

Orthogonal polynomials: Chebyshev least squares

- **Chebyshev least squares (cont.):**EXAMPLE
- Approximate $f(x) = \exp(x)$ for $x \in [-1, 1]$
- **First order polynomial** approximation of $\exp(x)$ is

$$\exp(x) \approx c_0 T_0(x) + c_1 T_1(x)$$

where

$$T_0(x) = \cos((0)(\arccos x)) = 1$$

$$T_1(x) = \cos(\arccos x) = x$$

and

$$c_0 = \frac{1}{\pi} \int_{-1}^{1} \frac{\exp(x)}{\sqrt{1-x^2}} dx = 1.2661$$

$$c_1 = \frac{2}{\pi} \int_{-1}^{1} \frac{\exp(x)\, x}{\sqrt{1-x^2}} dx = 1.1303$$

Orthogonal polynomials: Chebyshev least squares

- **Chebyshev least squares (cont.):**EXAMPLE
- Second order polynomial approximation of $\exp(x)$ is

$$\exp(x) \approx c_0 T_0(x) + c_1 T_1(x) + c_2 T_2(x)$$

where c_0, c_1, T_0 and T_1 are the same as before and

$$T_2(x) = \cos(2 \arccos x)$$

and$c$

$$c_2 = \frac{2}{\pi} \int_{-1}^{1} \frac{\exp(x)\ T_2(x)}{\sqrt{1-x^2}} dx = 0.2715$$

Orthogonal polynomials: Chebyshev least squares

- **Chebyshev least squares (cont.):**EXAMPLE

`go to inneficient matlab script!`

`cheby_least_squares.m`

- Type CC for each marix of coefficients....

Interpolation: Basics

- Usually we don't have the value of $f(x)$ for all its domain.
- We only have the value of $f(x)$ at some finite set of points:

$$(x_0, f(x_0)), (x_1, f(x_1)), ..., (x_n, f(x_n))$$

  - Interplation nodes or points: $x_0, x_1, ..., x_n$

- **Interpolation problem**: Find the degree $\leq n$ polynomial $p_n(x)$ that passes through these points:

$$f(x_i) = p_n(x_i) \quad \forall i : 0, ...n$$

Interpolation: Basics

- Existence and uniqueness of the interpolating polynomial

Theorem

If $x_0, \ldots, x_n$ are distinct, then for any $f\left(x_0\right), \ldots, f\left(x_n\right)$ there existis a unique polynomial $p_n\left(x_i\right)$ of degree $\leq n$ such that the interpolation conditions

$$f\left(x_i\right) = p_n\left(x_i\right) \quad \forall i : 0s, \ldots n$$

are satisfied.

Linear interpolation

- The simplest case is **linear interpolation** (i.e., $n = 1$) with two data points

$$\left(x_0, f\left(x_0\right)\right), \left(x_1, f\left(x_1\right)\right)$$

- The interpolation conditions are:

$$\begin{aligned} f\left(x_0\right) &= p_1\left(x_0\right) \\ &= a_0 + a_1 x_0 \end{aligned}$$

$$\begin{aligned} f\left(x_1\right) &= p_1\left(x_1\right) \\ &= a_0 + a_1 x_1 \end{aligned}$$

Linear interpolation

- Solving the above system yields

$$a_0 = f(x_0) - \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x_0$$

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

- Thus, the interpolating polynomial is

$$p_1(x) = \left(f(x_0) - \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x_0 \right) + \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x$$

Linear interpolation

- Notice that the interpolating polynomial can be written as
  - **Power form**

$$p_1(x) = \left(f(x_0) - \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x_0 \right) + \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x$$

  - **Newton form**

$$p_1(x) = f(x_0) + \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) (x - x_0)$$

  - **Lagrange form**

$$p_1(x) = \left(\frac{x - x_1}{x_0 - x_1} \right) f(x_0) + \left(\frac{x - x_0}{x_1 - x_0} \right) f(x_1)$$

- We have the same interpolating polynomial $p_1(x)$ written in two different forms.

Quadratic interpolation

- If we assume $n = 2$ and three data points

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$

Quadratic interpolation

- The interpolation conditions are

$$
\begin{aligned}
f(x_0) &= p_2(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 \\
f(x_1) &= p_2(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 \\
f(x_2) &= p_2(x_2) = a_0 + a_1 x_2 + a_2 x_2^2
\end{aligned}
$$

Quadratic interpolation

- In matrix form the interpolation conditions are

$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{bmatrix}$$

  or im more compact form

$$Va = b$$

- Notice that V is a **Vandermonde** matrix which is ill-conditioned.
  - The condition number of V is large so it is better to compute the $a's$ by using another form of writing the interpolating polynomial.

Quadratic interpolation

- But we can still do it by hand since this is a 3by3 matrix!
- Solving the above system yields

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \frac{x_1 x_2}{(x_0-x_1)(x_0-x_2)} & \frac{-x_0 x_2}{(x_0-x_1)(x_1-x_2)} & \frac{x_0 x_1}{(x_0-x_2)(x_1-x_2)} \\ \frac{-(x_1+x_2)}{(x_0-x_1)(x_0-x_2)} & \frac{x_0+x_2}{(x_0-x_1)(x_1-x_2)} & \frac{-(x_0+x_1)}{(x_0-x_2)(x_1-x_2)} \\ \frac{1}{(x_0-x_1)(x_0-x_2)} & \frac{-1}{(x_0-x_1)(x_1-x_2)} & \frac{1}{(x_0-x_2)(x_1-x_2)} \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{bmatrix}$$

Quadratic interpolation

- Or by using the *Matlab* symbolic toolbox

```
>> syms a b c
>> A = [1 a a^2; 1 b b^2; 1 c c^2];
>> inv(A)
ans =
[ (b*c)/((a - b)*(a - c)), -(a*c)/((a - b)*(b - c)),
(a*b)/((a - c)*(b - c))]
[ -(b + c)/((a - b)*(a - c)), (a + c)/((a - b)*(b - c)),
-(a + b)/((a - c)*(b - c))]
[ 1/((a - b)*(a - c)), -1/((a - b)*(b - c)), 1/((a - c)*(b
- c))]
```

Quadratic interpolation

- Solving the above system yields the coefficients:

$$a_0 = \left(\frac{x_1 x_2}{(x_0 - x_1)(x_0 - x_2)}\right) f(x_0) + \left(\frac{-x_0 x_2}{(x_0 - x_1)(x_1 - x_2)}\right) f(x_1) + \left(\frac{x_0 x_1}{(x_0 - x_2)(x_1 - x_2)}\right) f(x_2)$$

$$a_1 = \left(\frac{-(x_1 + x_2)}{(x_0 - x_1)(x_0 - x_2)}\right) f(x_0) + \left(\frac{x_0 + x_2}{(x_0 - x_1)(x_1 - x_2)}\right) f(x_1) + \left(\frac{-(x_0 + x_1)}{(x_0 - x_2)(x_1 - x_2)}\right) f(x_2)$$

$$a_2 = \left(\frac{1}{(x_0 - x_1)(x_0 - x_2)}\right) f(x_0) + \left(\frac{-1}{(x_0 - x_1)(x_1 - x_2)}\right) f(x_1) + \left(\frac{1}{(x_0 - x_2)(x_1 - x_2)}\right) f(x_2)$$

Quadratic interpolation

- The approximating second order polynomial in "power" form is

$$p_2\left(x\right) = a_0 + a_1 x + a_2 x^2$$

  where a_0, a_1 and a_2 are defined above.
- Notice that $p_2\left(x\right)$ is a linear combination of $n+1=3$ monomials each of degree $0, 1, 2$.

Quadratic interpolation

- After "some" algebra we can write $p_2(x)$ in different forms:
- **Lagrange form**

$$p_2(x) = f(x_0)\left(\frac{(x-x_1)}{(x_0-x_1)}\frac{(x-x_2)}{(x_0-x_2)}\right) + f(x_1)\left(\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\right) + f(x_2)\left(\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\right)$$

- The above is a linear combination of $n+1=3$ polynomials of degree $n=2$. The coefficients are the interpolated values $f(x_0), f(x_1)$ and $f(x_2)$.

Quadratic interpolation

- By doing "some" algebra we can write $p_2(x)$ in different forms:
- **Newton form**

$$p_2(x) = f(x_0) + \left(\frac{f(x_1) - f(x_0)}{x_1 - x_0}\right)(x - x_0) + \left(\frac{\left(\frac{f(x_2)-f(x_1)}{(x_2-x_1)}\right) - \left(\frac{f(x_1)-f(x_0)}{(x_1-x_0)}\right)}{(x_2 - x_0)}\right)(x - x_0)(x - x_1)$$

- The above is a linear combination of $n + 1 = 3$ polynomials each of degree $0, 1, 2$. The coefficients are the called **divided differences.**

Interpolation: The general case

- The interpolation conditions when we have $n+1$ data points: $\{(x_0, f(x_0)), (x_1, f(x_1)), ..., (x_n, f(x_n))\}$

$$f(x_i) = p_n(x_i) \quad \forall i : 0, \ldots n$$

- $p_n(x_i)$ written in "power" form is

$$p_n(x_i) = \sum_{j=0}^{n} a_j x^j$$

Interpolation: The general case

- The interpolation conditions can be written as

$$f\left(x_i\right) = \sum_{j=0}^{n} a_j x_i^j \quad \forall i : 0, \dots n$$

or

$$f\left(x_0\right) = a_0 + a_1 x_0 + \dots + a_n x_0^n$$

$$f\left(x_1\right) = a_0 + a_1 x_1 + \dots + a_n x_1^n$$

$$\dots$$

$$f\left(x_n\right) = a_0 + a_1 x_n + \dots + a_n x_n^n$$

Interpolation: The general case

- In matrix form

$$\begin{bmatrix} 1 & x_0 & \ldots & x_0^n \\ 1 & x_1 & \ldots & x_1^n \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_n & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \ldots \\ f(x_n) \end{bmatrix}$$

- The matrix to be inverted is a Vandermonde matrix: Also an ill-conditioned matrix.

Interpolation: The general case

- We can also generalize the **lagrange form** of the interpolating polynomial:

$$p_n(x) = f(x_0)\, l_{n,0}(x) + f(x_1)\, l_{n,1}(x) + \ldots + f(x_n)\, l_{n,n}(x)$$

where $\left\{ l_{n,j}(x) \right\}_{j=0}^{n}$ are a family of $n+1$ polynomials of degree n given by

$$l_{n,j}(x) = \frac{(x - x_0)\ldots(x - x_{j-1})(x - x_{j+1})\ldots(x - x_n)}{(x_j - x_0)\ldots(x_j - x_{j-1})(x_j - x_{j+1})\ldots(x_j - x_n)} \quad \forall 0 \leq j \leq n$$

- More compactly

$$p_n(x) = \sum_{j=0}^{n} f(x_j)\, l_{n,j}(x)$$

Interpolation: The general case

- For $j = 0$

$$l_{n,0}(x) = \frac{(x - x_1)\ldots(x - x_n)}{(x_0 - x_1)\ldots(x_0 - x_n)} = \prod_{\substack{j=0\\ j\neq 0}}^{n} \frac{x - x_j}{x_0 - x_j}$$

- For $j = 1$

$$l_{n,1}(x) = \frac{(x - x_0)(x - x_2)\ldots(x - x_n)}{(x_1 - x_0)(x_1 - x_2)\ldots(x_1 - x_n)} = \prod_{\substack{j=0\\ j\neq 1}}^{n} \frac{x - x_j}{x_1 - x_j}$$

- For $j = n$

$$l_{n,n}(x) = \frac{(x - x_0)(x - x_2)\ldots(x - x_{n-1})}{(x_n - x_0)(x_n - x_2)\ldots(x_n - x_{n-1})} = \prod_{\substack{j=0\\ j\neq 2}}^{n} \frac{x - x_j}{x_2 - x_j}$$

Interpolation: The general case

- For any $\forall 0 \leq j \leq n$

$$l_{n,j}(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

- The lagrange form of the interpolating polynomial is

$$p_n(x) = \sum_{j=0}^{n} f(x_j)\, l_{n,j}(x)$$

  with $l_{n,j}(x)$ defined above.
- It turns out that computing the lagrange polynomial is more efficient than solving the vandermonde matrix!

Interpolation: The general case

- We can also generalize the newton form of the interpolating polynomial

  $$p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) \ldots + c_n(x - x_0)(x - x_1) \ldots$$

  where the coefficients $c_0, c_1, ..., c_n$ are called the divided difference and are denoted by

  $$c_0 = d(x_0)$$

  $$c_1 = d(x_1, x_0)$$

  $$c_2 = d(x_2, x_1, x_0)$$

  ...

  $$c_n = d(x_n, ..., x_1, x_0)$$

Interpolation: The general case

- The divided differences are defined as

$$d(x_0) = f(x_0)$$

$$d(x_1, x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$\begin{aligned} d(x_2, x_1, x_0) &= \frac{d(x_2, x_1) - d(x_1, x_0)}{x_2 - x_0} \\ &= \frac{\left(\frac{f(x_2)-f(x_1)}{x_2-x_1}\right) - \left(\frac{f(x_1)-f(x_0)}{x_1-x_2}\right)}{x_2 - x_0} \end{aligned}$$

Interpolation: The general case

- The divided differences are defined as (Cont.)

$$
\begin{aligned}
d\left(x_3, x_2, x_1, x_0\right) &= \frac{d\left(x_3, x_2, x_1\right) - d\left(x_2, x_1, x_0\right)}{x_3 - x_0} \\
&= \frac{\left(\frac{\left(\frac{f(x_3)-f(x_2)}{x_3-x_2}\right)-\left(\frac{f(x_2)-f(x_1)}{x_2-x_1}\right)}{x_3-x_2}\right) - \left(\frac{\left(\frac{f(x_2)-f(x_1)}{x_2-x_1}\right)-\left(\frac{f(x_1)-f(x_0)}{x_1-x_2}\right)}{x_2-x_0}\right)}{x_3 - x_0}
\end{aligned}
$$

$$
d\left(x_n, \ldots, x_1, x_0\right) = \frac{d\left(x_n, \ldots, x_1, x_0\right) - d\left(x_{n-1}, \ldots, x_1, x_0\right)}{x_n - x_0}
$$

Interpolation: The general case

- The generalization of the Newton form of the interpolating polynomial is

$$
\begin{aligned}
p_n(x) = {} & d(x_0) + d(x_1, x_0)(x - x_0) + d(x_2, x_1, x_0)(x - x_0)(x - x_1) + \ldots \\
& + d(x_n, \ldots, x_1, x_0)(x - x_0)(x - x_1)\ldots(x - x_{n-1})
\end{aligned}
$$

or

$$
p_n(x) = d(x_0) + \sum_{j=1}^{n} d(x_j, \ldots, x_1, x_0) \prod_{k=0}^{j-1} (x - x_k)
$$

Interpolation: The interpolation error

Theorem

Let $f(x) \in \mathbb{C}^{n+1}[a, b]$. Let $p_n(x)$ a polynomial of degree $\leq n$ such that it interpolates $f(x)$ at the $n+1$ **distinct** *nodes $\{x_0, x_1, ..., x_n\}$. Then $\forall x \in [a, b]$, there exists a $\xi_n \in (a, b)$ such that*

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_n) \prod_{k=0}^{n} (x - x_k)$$

Fact

The error term for the nth Taylor approximation around the point x_0 is

$$\frac{f^{(n+1)}(\xi_n)}{(n+1)!} (x - x_0)^{n+1}$$

Interpolation: The interpolation error

- Notice that applying the supremum norm to the interpolation error yields

$$\left\| f(x) - p_n(x) \right\|_\infty \leq \frac{1}{(n+1)!} \left\| f^{(n+1)}(\xi_n) \right\|_\infty \left\| \prod_{k=0}^{n} (x - x_k) \right\|_\infty$$

or

$$\max_{x\in[a,b]} \left| f(x) - p_n(x) \right| \leq \frac{1}{(n+1)!} \left(\max_{\xi_n\in[a,b]} \left| f^{(n+1)}(\xi_n) \right| \right) \left(\max_{x\in[a,b]} \left| \prod_{k=0}^{n} (x - x_k) \right| \right)$$

- The R.H.S is an **upper bound for the interpolation error**.

Interpolation: The interpolation error

- We would like to have

$$\lim_{n\to\infty}\left\{\frac{1}{(n+1)!}\left(\max_{\xi_n\in[a,b]}\left|f^{(n+1)}(\xi_n)\right|\right)\left(\max_{x\in[a,b]}\left|\prod_{k=0}^{n}(x-x_k)\right|\right)\right\}=0$$

  thus

$$\lim_{n\to\infty}\left(f(x)-p_n(x)\right)=0$$

- But nothing guarantees convergence (neither point or uniform convergence).

Interpolation: The interpolation error

- The maximun error depends on the interpolation nodes $\{x_0, x_1, ..., x_n\}$ through the term $\left(\max_{x\in[a,b]}\left|\prod_{k=0}^{n}(x-x_k)\right|\right)$.
- Notice that no other term depends on the interpolating nodes once we look for the **maximum error**.
- We can choose the nodes in order to **minimize the interpolation error:**

$$\min_{\{x_0,...,x_n\}}\left\{\max_{x\in[a,b]}\left|\prod_{k=0}^{n}(x-x_k)\right|\right\}$$

Interpolation: Choosing the nodes

- What happens if we work with uniformly spaced nodes?, that is with nodes such that

$$x_i = a + \left(\frac{i-1}{n-1}\right)(b-a) \quad \text{for } i : 1, .., n$$

- Recall that:
  - We want to interpolate a function $f(x) : [a, b] \to \mathbb{R}$
  - n is the degree of the interpolating polynomial: $p_n(x)$
  - The interpolation conditions are

$$f(x_i) = p_n(x_i) \text{ ...for } i : 0, .., n$$

so, if $n = 10$, we need $n + 1 = 11$ data points.

Interpolation: Choosing the nodes

- **Runge's example**: Let $f(x) = \frac{1}{1+x^2}$ defined on the interval $[-5, 5]$.
- Find the lagrange polnomial approximation for $n = 11$.

`go to inneficient matlab script!`

`runge_example_lagrange_interpolation_UN.m`

- Play increasing the degree of interpolation and see that there is no convergence.
- The graph replicates figure 6.6 of Judd's book.

Interpolation: Choosing the nodes

- But we can choose the nodes in order to obtain the smallest value for

$$\left\| \prod_{k=0}^{n} (x - x_k) \right\|_{\infty} = \max_{x \in [a,b]} \left| \prod_{k=0}^{n} (x - x_k) \right|$$

- **Chebyshev polynomials one more time:** Recall the monic Chebyshev polynomial is

$$\widetilde{T}_j(x) = \frac{\cos(j \arccos x)}{2^{j-1}}$$

with $x \in [-1, 1]$ and $j = 1, 2, ..., n$

- Then, the zeros of $\widetilde{T}_n(x)$ are given by the soltution to

$$\begin{aligned} \widetilde{T}_n(x) &= 0 \\ \cos(n \arccos x) &= 0 \\ \cos(n\theta) &= 0 \end{aligned}$$

where $\theta = \arccos x$. Thus $\theta \in [0, \pi]$.

Interpolation: Choosing the nodes

- Zeros occur when $\cos(n\theta) = 0$, that is

$$n\theta = \left(\frac{2k-1}{2}\right)\pi \quad \text{for } k = 1, 2, ..n$$

- Notice that

$$\begin{aligned}
&\cos\left(\frac{2k-1}{2}\pi\right) \\
&= \cos\left(k\pi - \frac{\pi}{2}\right) \\
&= \cos(k\pi)\underbrace{\cos\left(\frac{\pi}{2}\right)}_{=0} - \underbrace{\sin(k\pi)}_{=0}\sin\left(\frac{\pi}{2}\right) \\
&= 0
\end{aligned}$$

Interpolation: Choosing the nodes

- The equation $\widetilde{T}_n(x) = 0$ has n different roots given by

$$n\theta = \left(k - \frac{1}{2}\right)\pi \quad \text{for } k = 1, 2, ..n$$

- This means that
  - For $k = 1$

$$\theta_1 = \frac{\pi}{2n}$$

  - For $k = 2$

$$\theta_2 = \left(\frac{3}{2}\right)\frac{\pi}{n}$$

  - For $k = n$

$$\theta_n = \left(\frac{2n-1}{2}\right)\frac{\pi}{n}$$

Interpolation: Choosing the nodes

- Roots for $\cos(n\theta)$ where $\theta \in [0, \pi]$:

	$n = 1$	$n = 2$	$n = 3$	...	n
$k = 1$	$\frac{\pi}{2}$	$\frac{\pi}{4}$	$\frac{\pi}{6}$		$\frac{\pi}{2n}$
$k = 2$		$\frac{3}{4}\pi$	$\frac{3}{6}\pi$		$\frac{3}{2n}\pi$
$k = 3$			$\frac{5}{6}\pi$		$\frac{5}{2n}\pi$
...					
$k = n$					$\left(\frac{2n-1}{2n}\right)\pi$

Interpolation: Choosing the nodes

- Plooting $\cos(j\theta)$ for $\theta \in [0, \pi]$ and $j = 1, 2, ..., n$

`go to inneficient matlab script!`

`cheby_nodes.m`

Interpolation: Choosing the nodes

- We want the roots of the monic chebyshev and we have the roots of the cosine function:

$$n\theta = \left(k - \frac{1}{2}\right)\pi \quad \text{for } k = 1, 2, ..n$$

- but $\theta = \arccos x$, thus

$$\arccos x = \left(\frac{2k-1}{2n}\right)\pi \quad \text{for } k = 1, 2, ..n$$

then the roots of the chebyshev polynomials are

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

- Notice that the roots of $\widetilde{T}_n(x)$ are the same as the roots of $T_n(x)$.

Interpolation: Choosing the nodes

- Ploting chebyshev nodes

`go to inneficient matlab script!`

`cheby_nodes.m`

Interpolation: Choosing the nodes

- The following theorem summarizes some characteristics of chebyshev polynomials

Theorem

The Chebyshev polynomial $T_n(x)$ of degree $n \geq 1$ has n zeros in $[-1, 1]$ at

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

Moreover, $T_n(x)$ assumes its extremum at

$$x_k^* = \cos\left(\frac{k\pi}{n}\right) \quad \textit{for } k = 0, 1, .., n$$

with

$$T_n(x_k^*) = (-1)^k \quad \textit{for } k = 0, 1, .., n$$

Interpolation: Choosing the nodes

Corollary

The monic Chebyshev polynomial $\widetilde{T}_n(x)$ has the same zeros and extremum points as $T_n(x)$ but with extremum values given by

$$\widetilde{T}_n(x_k^*) = \frac{(-1)^k}{2^{n-1}} \text{ for } k = 0, 1, ..., n$$

Interpolation: Choosing the nodes

- Extrema of Chebyshev polynomials:

$$T_n(x) = \cos(n \arccos x)$$

then

$$\begin{aligned}\frac{dT_n(x)}{dx} &= T_n'(x) \\ &= -\sin(n \arccos x)\left(-\frac{n}{\sqrt{1-x^2}}\right) \\ &= \frac{n \sin(n \arccos x)}{\sqrt{1-x^2}}\end{aligned}$$

- Notice that $T_n'(x)$ is a polynomial of degree $n-1$ with zeros given by

$$T_n'(x) = 0$$

Interpolation: Choosing the nodes

- Excluding the endpoints of the domain, $x = -1$ or $x = 1$, then, extremum points occurs when

  $$\sin\left(n \arccos x\right) = 0$$

  or when

  $$\sin\left(n\theta\right) = 0$$

  for $\theta \in (0, \pi)$. Thus

  $$n\theta_k = k\pi \quad \text{for all } k = 1, 2, ..., n-1$$

- Solving for x yields

  $$\begin{aligned} \theta = \arccos x &= \frac{k\pi}{n} \\ \Longrightarrow x_k^* &= \cos\left(\frac{k\pi}{n}\right) \quad \text{for } k = 1, 2, ..., n-1 \end{aligned}$$

- Obviusly, extrema also occur at the **endpoints** of the domain (i.e, $x = -1$ or $x = 1$), that is when $k = 0$ or when $k = n$.

Interpolation: Choosing the nodes

- The extremum values of $T_n(x)$ occurs when

$$\begin{aligned} T_n(x^*) &= \cos(n \arccos x^*) \\ &= \cos\left(n \arccos\left(\cos\left(\frac{k\pi}{n}\right)\right)\right) \\ &= \cos\left(n \frac{k\pi}{n}\right) \\ &= \cos(k\pi) \\ &= (-1)^k \quad \text{for } k = 0, 1, ..., n \end{aligned}$$

  Notice that we are including the endpoints of the domain.

- The above result implies

$$\max_{x \in [-1,1]} |T_n(x)| = 1$$

Interpolation: Choosing the nodes

- Extrema for monic Chebyshev polynomials are characterized by the same points since

$$\widetilde{T}_n(x) = \frac{T_n(x)}{2^{n-1}}$$

thus

$$\widetilde{T}'_n(x_k^*) = T'_n(x_k^*) = 0 \quad \text{for } k = 0, 1, ..., n$$

- But the extremum values of $\widetilde{T}_n(x)$ are given by

$$\begin{aligned}\widetilde{T}_n(x_k^*) &= \frac{T_n(x_k^*)}{2^{n-1}} \quad \text{for } k = 0, 1, ..., n \\ &= \frac{(-1)^k}{2^{n-1}} \quad \text{for } k = 0, 1, ..., n\end{aligned}$$

Therefore

$$\max_{x \in [-1,1]} \left| \widetilde{T}_n(x) \right| = \frac{1}{2^{n-1}}$$

Interpolation: Choosing the nodes

- An important property of monic Chebyshev polynomials is given by the following theorem

Theorem

If $\widetilde{p}_n(x)$ is a monic polynomial of degree n defined on $[-1, 1]$, then

$$\max_{x\in[-1,1]} \left|\widetilde{T}_n(x)\right| = \frac{1}{2^{n-1}} \leq \max_{x\in[-1,1]} \left|\widetilde{p}_n(x)\right|$$

for all monic polynomials of degree n.

Interpolation: Choosing the nodes

- Recall that we want to choose the interpolation nodes $\{x_0, ..., x_n\}$ in order to solve

$$\min_{\{x_0,\dots,x_n\}} \left\{ \max_{x\in[a,b]} \left| \prod_{k=0}^{n} (x - x_k) \right| \right\}$$

- Choosing the interpolation nodes is the same as choosing the zeros of $\prod_{k=0}^{n} (x - x_k)$.

- Notice that $\prod_{k=0}^{n} (x - x_k)$ is a monic polynomial of degree $n+1$. Therefore it must be the case that

$$\max_{x\in[-1,1]} \left| \widetilde{T}_{n+1}(x) \right| = \frac{1}{2^n} \leq \max_{x\in[-1,1]} \left| \prod_{k=0}^{n} (x - x_k) \right|$$

Interpolation: Choosing the nodes

- The smallest value that $\max_{x\in[-1,1]}\left|\prod_{k=0}^{n}(x-x_k)\right|$ can take is $\frac{1}{2^n}$.

  Therefore

$$\max_{x\in[-1,1]}\left|\prod_{k=0}^{n}(x-x_k)\right| = \frac{1}{2^n}$$
$$= \max_{x\in[-1,1]}\left|\widetilde{T}_{n+1}(x)\right|$$

  which implies that

$$\prod_{k=0}^{n}(x-x_k) = \widetilde{T}_{n+1}(x)$$

- Therefore the zeros of $\prod_{k=0}^{n}(x-x_k)$ must be the zeros of $\widetilde{T}_{n+1}(x)$ which are given by

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

Interpolation: Choosing the nodes

- Since $\max_{x\in[-1,1]} \left| \prod_{k=0}^{n} (x - x_k) \right| = \frac{1}{2^n}$ then the maximum interpolation error becomes

$$\max_{x\in[a,b]} |f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \left(\max_{\xi_n\in[a,b]} \left| f^{(n+1)}(\xi_n) \right| \right) \left(\max_{x\in[a,b]} \left| \prod_{k=0}^{n} (x \right.\right.$$

$$\max_{x\in[a,b]} |f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \left(\max_{\xi_n\in[a,b]} \left| f^{(n+1)}(\xi_n) \right| \right) \left(\frac{1}{2^n} \right)$$

- Chebyshev nodes eliminate violent oscillations for the error term compared to uniform spaced nodes.
- Interpolation with Chebyshev nodes has better convergence properties.
- It is possible to show that $p_n(x) \to f(x)$ as $n \to \infty$ uniformly. This is not guaranteed under uniform spaced nodes.

Interpolation: Choosing the nodes

- **Runge's example with chebyshev nodes**

`runge_example_cheby_nodes.m`

Interpolation: Choosing the nodes

- Figure 6.2 of Miranda and Flecker book:
- Comparing the interpolation errors of $f(x) = \exp(-x)$ defined in $x \in [-5, 5]$ with 10-node polynomial approximation

`example_miranda_cheby_nodes`

Interpolation through regresssion

- The interpolation conditions require to have the same number of data points (interpolation data) and unknown coefficients in order to proceed.
- But we can also have the case, where the data points exceed the number of unknow coefficients.
- For this case, we can use **discrete least squares**: Use m interplation points to find $n < m$ coefficients.
  - The omitted terms are high degree polynomials that may produce undesirable oscillations.
  - The result is a smoother function that approximates the data.

Interpolation through regresssion

- **Objective:** Construct a degree n polynomial , $\widehat{f}(x)$, that approximates the function f for $x \in [a, b]$ using $m > n$ interpolation nodes.

$$\widehat{f}(x) = \sum_{j=0}^{n} c_j T_j(x_k)$$

Interpolation through regresssion

- **Algorithm:**
- **Step 1: Compute the m Chebyshev interpolation nodes on** $[-1, 1]$:

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

  - As if we want an $m-$degree Chebyshev interpolation.
- **Step 2: Adjust the nodes to the interval** $[a, b]$ **:**

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

- **Step 3: Evaluate f at the nodes:**

$$y_k = f(x_k) \text{ ...for } k = 1, ..., m$$

Interpolation through regresssion

- **Algorithm (Cont.):**
- **Step 4: Compute the Chebyshev least squares coefficients**
  - The coefficients that solve the discrete LS problem

$$\min \sum_{k=1}^{m} \left[y_k - \sum_{j=0}^{n} c_j T_j\left(z_k\right) \right]^2$$

are given by

$$c_j = \frac{\sum_{k=1}^{m} y_k T_j\left(z_k\right)}{\sum_{k=1}^{m} \left(T_j\left(z_k\right)\right)^2} \quad \text{for } j = 0, 1, ..., n$$

where z_k is the inverse transformation of x_k :

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

Interpolation through regresssion

- Finally, the LS Chebyshev approximating polynomial is given by

$$\widehat{f}(x) = \sum_{j=0}^{n} c_j T_j(z)$$

where $z \in [-1, 1]$ and it is given by

$$z = \frac{2x - (a + b)}{b - a}$$

and the c_j are estimated using the LS coefficients

$$c_j = \frac{\sum_{k=1}^{m} y_k T_j(z_k)}{\sum_{k=1}^{m} \left(T_j(z_k)\right)^2} \quad \text{for } j = 0, 1, ..., n$$

Piecewise linear approximation

- If we have interpolation data given by $\{(x_0, f(x_0)), (x_1, f(x_1)), ..., (x_n, f(x_n))\}$
- We can divide the interpolation nodes in subintervals of the form

$$[x_i, x_{i+1}] \quad \text{for } i = 0, 1, ..., n-1$$

- Then we can perform linear interpolation in each subinterval:
  - Interpolation conditions for each subinterval:

$$\begin{aligned} f(x_i) &= a_0 + a_1 x_i \\ f(x_{i+1}) &= a_0 + a_1 x_{i+1} \end{aligned}$$

Piecewise linear approximation

- Linear interpolation in each subinterval yields $[x_i, x_{i+1}]$:
  - The interpolating coefficients:

$$a_0 = f(x_i) - \left(\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \right) x_i$$

$$a_1 = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

  - Piecewise linear interpoland:

$$p_i(x) = f(x_i) + \left(\frac{x - x_i}{x_{i+1} - x_i} \right) (f(x_{i+1}) - f(x_i))$$

Piecewise linear approximationPiecewise polynomial approximation: Splines

- Example: $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$
- Then we have two subintervals

$$[x_0, x_1] \quad \text{and } .[x_1, x_2]$$

- The interpolating function is given by:

$$\widehat{f}(x) = \begin{cases} f(x_0) + \left(\frac{x-x_0}{x_1-x_0}\right)(f(x_1) - f(x_0)) & \text{for } x \in [x_0, x_1] \\ f(x_1) + \left(\frac{x-x_1}{x_2-x_1}\right)(f(x_2) - f(x_1)) & \text{for } x \in [x_1, x_2] \end{cases}$$

Piecewise polynomial approximation: Splines

- A spline is any smooth function that is piecewise polynomial but also smooth where the polynomial pieces connect.
- Assume that the interpolation data is given by $\{(x_0, f(x_0)), (x_1, f(x_1)), ..., (x_m, f(x_m))\}$.
- A function $s(x)$ defined on $[a, b]$ is a spline of order n if:
  - s is C^{n-2} on $[a, b]$
  - $s(x)$ is a polynomial of degree $n-1$ on each subinterval $[x_i, x_{i+1}]$ for $i = 0, 1, .., m-1$
- Notice that an order 2-spline is the piecewise linear interpolant equation.

Piecewise polynomial approximation: Splines

- A cubic spline is a spline of order 4:
  - s is C^2 on $[a, b]$
  - $s(x)$ is a polynomial of degree $n - 1 = 3$ on each subinterval $[x_i, x_{i+1}]$ for $i = 0, 1, .., m - 1$

$$s(x) = a_i + b_i x + c_i x^2 + d_i x^3 \quad \text{for } x \in [x_i, x_{i+1}], i = 0, 1, .., m - 1$$

Piecewise polynomial approximation: Splines

- Example of cubic spline: Assume that we have the following 3 data points: $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$
- There are two subintervals: $[x_0, x_1]$ and .$[x_1, x_2]$.
- A cubic spline is a function s such that
  - s is C^2 on $[a, b]$
  - $s(x)$ is a polynomial of degree 3 on each subinterval:

$$s(x) = \begin{cases} s_0(x) = a_0 + b_0 x + c_0 x^2 + d_0 x^3 & \text{for } x \in [x_0, x_1] \\ s_1(x) = a_1 + b_1 x + c_1 x^2 + d_1 x^3 & \text{for } x \in [x_1, x_2] \end{cases}$$

- Notice that in this case we have 8 unknowns: $a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1$

Piecewise polynomial approximation: Splines

- Example (Cont.): We need 8 conditions
  - Interpolation and continuity at interior nodes conditions

$$
\begin{aligned}
y_0 &= s_0(x_0) = a_0 + b_0 x_0 + c_0 x_0^2 + d_0 x_0^3 \\
y_1 &= s_0(x_1) = a_0 + b_0 x_1 + c_0 x_1^2 + d_0 x_1^3 \\
y_1 &= s_1(x_1) = a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3 \\
y_2 &= s_1(x_2) = a_1 + b_1 x_2 + c_1 x_2^2 + d_1 x_2^3
\end{aligned}
$$

Piecewise polynomial approximation: Splines

- Example (Cont.): We need 8 conditions
  - First and second derivatives must agree at the interior nodes

$$s_0'(x_1) = s_1'(x_1)$$
$$b_0 + 2c_0x_1 + 3d_0x_1^2 = b_1 + 2c_1x_1 + 3d_1x_1^2$$

$$s_0''(x_1) = s_1''(x_1)$$
$$2c_0 + 6d_0x_1 = 2c_1 + 6d_1x_1$$

Piecewise polynomial approximation: Splines

- Up to know we have 6 conditions, we need two more conditions
- 3 ways to obtain the additional conditions:
  - **Natural spline:** $s'(x_0) = s'(x_2) = 0$
  - **Hermite spline:** If we have information on the slope of the original function at the end points:

$$f'(x_0) = s'(x_0)$$

$$f'(x_2) = s'(x_2)$$

  - **Secant Hermite spline:** use the secant to estimate the slope at the end points

$$s'(x_0) = \frac{s(x_1) - s(x_0)}{x_1 - x_0}$$

$$s'(x_2) = \frac{s(x_2) - s(x_1)}{x_2 - x_1}$$

Piecewise polynomial approximation: Splines

- Generalization of cubic splines:

`See Judd's book!!!`