Function Approximation

David Florian Hoyle
Raul Cruz Tadle

UCSC

December 8, 2014
Objective

- Obtain an approximation for $f(x)$ by another function $\hat{f}(x)$
- **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

1 Approximation theory
   1 Weierstrass approximation theorem
   2 Minimax approximation
   3 Orthogonal polynomials and least squares
   4 Near minimax approximation

2 Interpolation
   1 The interpolation problem
   2 Different representations for the interpolating polynomial
   3 The error term
   4 Minimizing the error term with Chebyshev nodes
   5 Discrete least squares
   6 Piecewise polynomial interpolation: splines
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:
  \[
  (x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))
  \]
- Interpolation nodes or points: \( x_0, x_1, \ldots, x_n \)

**Interpolation problem**: Find the polynomial that has a maximum degree that is less than or equal to the polynomial degree \( n \) of \( p_n(x) \). Note that \( p_n(x) \) passes through the interpolation points:

\[
f(x_i) = p_n(x_i) \quad \forall i : 0, \ldots n
\]
Existence and uniqueness of the interpolating polynomial

**Theorem**

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

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

are satisfied.
Linear interpolation

- The simplest case is **linear interpolation** (i.e., \( n = 1 \)) with two data points
  \[
  (x_0, f(x_0)), (x_1, f(x_1))
  \]
- The interpolation conditions are:
  \[
  f(x_0) = p_1(x_0) = a_0 + a_1 x_0 \\
  f(x_1) = p_1(x_1) = a_0 + a_1 x_1
  \]
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

\[
p_1(x) = \left( f(x_0) - \left( \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x_0 \right) + \left( \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) x
\]
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 three different forms.
Quadratic interpolation

- If we assume \( n = 2 \) and three data points

\[
(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))
\]
The interpolation conditions are

\[ f(x_0) = p_2(x_0) = a_0 + a_1x_0 + a_2x_0^2 \]

\[ f(x_1) = p_2(x_1) = a_0 + a_1x_1 + a_2x_1^2 \]

\[ f(x_2) = p_2(x_2) = a_0 + a_1x_2 + a_2x_2^2 \]
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 in a more compact form

\[Va = b\]

Notice that \(V\) is a **Vandermonde** matrix.
Quadratic interpolation

- But we can still do it by hand since this is a 3x3 matrix!
- We need the inverse of the Vandermonde matrix. Using the *Matlab* symbolic toolbox, we have

\[
\text{syms a b c}
\]
\[
\text{A = [1 a a^2; 1 b b^2; 1 c c^2];}
\]
\[
\text{inv(A)}
\]
\[
\text{ans =}
\]
\[
\begin{bmatrix}
\frac{(b*c)}{(a - b)*(a - c)}, & -\frac{(a*c)}{(a - b)*(b - c)}, & \frac{(a*b)}{(a - c)*(b - c)} \\
-\frac{(b + c)}{(a - b)*(a - c)}, & \frac{(a + c)}{(a - b)*(b - c)}, & -\frac{(a + b)}{(a - c)*(b - c)} \\
1/((a - b)*(a - c)), & -1/((a - b)*(b - c)), & 1/((a - c)*(b - c))
\end{bmatrix}
\]
Quadratic interpolation

Incorporating the Matlab results and manipulating the system yields

\[
\begin{bmatrix}
  a_0 \\
  a_1 \\
  a_2
\end{bmatrix}
= 
\begin{bmatrix}
  \frac{x_1 x_2}{(x_0-x_1)(x_0-x_2)} & -\frac{x_0 x_2}{(x_0-x_1)(x_1-x_2)} & \frac{x_0 x_1}{(x_0-x_2)(x_1-x_2)} \\
  -\frac{x_0 x_2}{(x_1+x_2)} & \frac{x_0 + x_2}{(x_0-x_1)(x_1-x_2)} & -\frac{(x_0 + x_1)}{(x_0-x_2)(x_1-x_2)} \\
  \frac{1}{(x_0-x_1)(x_0-x_2)} & -\frac{1}{(x_0-x_1)(x_1-x_2)} & \frac{1}{(x_0-x_2)(x_1-x_2)}
\end{bmatrix}
\begin{bmatrix}
  f(x_0) \\
  f(x_1) \\
  f(x_2)
\end{bmatrix}
\]
Solving the above system yields the coefficients:

\[
\begin{align*}
    a_0 &= \left( \frac{x_1 x_2}{(x_0 - x_1) (x_0 - x_2)} \right) f(x_0) + \left( \frac{-x_0 x_2}{(x_0 - x_1) (x_1 - x_2)} \right) f(x_1) \\
    &\quad + \left( \frac{x_0 x_1}{(x_0 - x_2) (x_1 - x_2)} \right) f(x_2) \\
    a_1 &= \left( \frac{- (x_1 + x_2)}{(x_0 - x_1) (x_0 - x_2)} \right) f(x_0) + \left( \frac{x_0 + x_2}{(x_0 - x_1) (x_1 - x_2)} \right) f(x_1) \\
    &\quad + \left( \frac{- (x_0 + x_1)}{(x_0 - x_2) (x_1 - x_2)} \right) f(x_2) \\
    a_2 &= \left( \frac{1}{(x_0 - x_1) (x_0 - x_2)} \right) f(x_0) + \left( \frac{-1}{(x_0 - x_1) (x_1 - x_2)} \right) f(x_1) \\
    &\quad + \left( \frac{1}{(x_0 - x_2) (x_1 - x_2)} \right) f(x_2)
\end{align*}
\]
However, the Vandermonde matrix 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.

- We prefer a different method, if possible.
Quadratic interpolation

- The approximating second order polynomial in “power” form is

\[ p_2(x) = a_0 + a_1x + a_2x^2 \]

where \( a_0, a_1 \) and \( a_2 \) are defined above.

- Notice that \( p_2(x) \) is a linear combination of \( n + 1 = 3 \) monomials each of degree 0, 1, and 2, respectively.
Quadratic interpolation

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

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

- 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 $f(x_2)$. 
Newton form of \( p_2(x) \):

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

The above is a linear combination of \( n + 1 = 3 \) polynomials each of degree 0, 1, and 2. The coefficients are what are called \textit{divided differences}.
The interpolation conditions when we have \( n + 1 \) data points:
\[
\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}
\]

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

\( p_n(x_i) \) written in “power” form is
\[
p_n(x_i) = \sum_{j=0}^{n} a_j x^j
\]
The interpolation conditions can be written as

\[ f(x_i) = \sum_{j=0}^{n} a_j x_i^j \quad \forall i : 0, \ldots, n \]

or

\[ f(x_0) = a_0 + a_1 x_0 + \ldots + a_n x_0^n \]
\[ f(x_1) = a_0 + a_1 x_1 + \ldots + a_n x_1^n \]
\[ \ldots \]
\[ f(x_n) = a_0 + a_1 x_n + \ldots + a_n x_n^n \]
Interpolation: The general case

- In matrix form

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

- The matrix to be inverted is a Vandermonde matrix (which we said earlier is an ill-conditioned matrix.)
Interpolation: The general case

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

\[
p_n(x) = f(x_0) l_{n,0}(x) + f(x_1) l_{n,1}(x) + ... + f(x_n) l_{n,n}(x)
\]

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

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

- More compactly,

\[
p_n(x) = \sum_{j=0}^{n} f(x_j) l_{n,j}(x)
\]
Interpolation: The general case

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

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

- For \( j = n \)
  \[
  l_{n,n}(x) = \frac{(x - x_0) (x - x_2) \ldots (x - x_{n-1})}{(x_n - x_0) (x_n - x_2) \ldots (x_n - x_{n-1})} = \prod_{j=0, j \neq 2}^{n} \frac{x - x_j}{x_2 - x_j}
  \]
Interpolation: The general case

- For all $0 \leq j \leq n$, 
  \[ l_{n,j}(x) = \prod_{\substack{j=0 \atop j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \]

- The Lagrange form of the interpolating polynomial is 
  \[ p_n(x) = \sum_{j=0}^{n} f(x_j) l_{n,j}(x) \]

- 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 \( c_0, c_1, \ldots, c_n \) are called the divided difference and are denoted by

\[
\begin{align*}
    c_0 &= d(x_0) \\
    c_1 &= d(x_1, x_0) \\
    c_2 &= d(x_2, x_1, x_0) \\
    & \ldots \\
    c_n &= d(x_n, \ldots, x_1, x_0)
\end{align*}
\]
The divided differences are defined as

\[ d(x_0) = f(x_0) \]

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

\[ d(x_2, x_1, x_0) = \frac{d(x_2, x_1) - d(x_1, x_0)}{x_2 - x_0} \]

\[ = \left( \frac{f(x_2) - f(x_1)}{x_2 - x_1} \right) - \left( \frac{f(x_1) - f(x_0)}{x_1 - x_2} \right) \]

\[ = \frac{x_2 - x_0}{x_2 - x_0} \]
The divided differences are defined as (Cont.)

\[
d(x_3, x_2, x_1, x_0) = \frac{d(x_3, x_2, x_1) - d(x_2, x_1, x_0)}{x_3 - x_0} \\
= \frac{\left( f(x_3) - f(x_2) \right) - \left( f(x_2) - f(x_1) \right)}{x_3 - x_2} - \frac{\left( f(x_2) - f(x_1) \right) - \left( f(x_1) - f(x_0) \right)}{x_2 - x_0} \\
= \frac{d(x_n, ..., x_2, x_1) - d(x_{n-1}, ..., x_1, x_0)}{x_n - x_0}
\]
The generalization of the Newton form of the interpolating polynomial is

\[ p_n(x) = d(x_0) + d(x_1, x_0)(x - x_0) + d(x_2, x_1, x_0)(x - x_0)(x - x_1) + \ldots + d(x_n, \ldots, x_1, x_0)(x - x_0)(x - x_1) \ldots (x - x_{n-1}) \]

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**
Assume $f(x) \in \mathbb{C}^{n+1}[a, b]$. Let $p_n(x)$ be 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 $\zeta_n \in [a, b]$ such that

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

**Fact**
The error term for the $n$th Taylor approximation around the point $x_0$ is

$$\frac{f^{(n+1)}(\zeta_n)}{(n+1)!} (x - x_0)^{n+1}$$
Notice that applying the supremum norm to the interpolation error yields

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

or

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

The R.H.S is an upper bound for the interpolation error.
We again note that \( n \) is the degree of the interpolating polynomial, \( p_n(x) \).

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

\[
\lim_{n \to \infty} (f(x) - p_n(x)) = 0
\]

But nothing guarantees convergence (neither point or uniform convergence).
The maximum error depends on the interpolation nodes \( \{x_0, x_1, \ldots, x_n\} \) through the term 
\[
\max_{x \in [a, b]} \left| \prod_{k=0}^{n} (x - x_k) \right|
\].

Note 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:

\[
\min_{\{x_0, \ldots, x_n\}} \left\{ \max_{x \in [a, b]} \left| \prod_{k=0}^{n} (x - x_k) \right| \right\}
\]
Interpolation: Choosing the nodes

- **Runge's example**: Let \( f(x) = \frac{1}{1+x^2} \) defined on the interval \([-5, 5]\). The approximation errors for Runge's function when \( n = 9 \) is shown in the **matlab** file (Runge example of Lagrange Interpolation).

- Increasing the degree of interpolation does not lead to convergence.

- This is the same as Figure 6.6 in Judd's book.
Interpolation: Choosing the nodes

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

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

Recall that:

- We want to interpolate a function $f(x) : [a, b] \rightarrow \mathbb{R}$
- The interpolation conditions are

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

so, if $n = 10$, we need $n + 1 = 11$ data points.
Interpolation: Choosing the nodes

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

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

We can do so by using **Chebyshev polynomials**. Recall the monic Chebyshev polynomial given by

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

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

Then, the zeros of $\tilde{T}_n (x)$ are given by the solution to

$$\tilde{T}_n (x) = 0$$

$$\cos (n \arccos x) = 0$$

$$\cos (n \theta) = 0$$

where $\theta = \arccos x$. Thus, $\theta \in [0, \pi]$. 
Interpolation: Choosing the nodes

- Zeros occur when \( \cos(n\theta) = 0 \). We know that this occurs when

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

- Also note that

\[
\cos\left(\frac{2k - 1}{2}\pi\right)
= \cos\left(k\pi - \frac{\pi}{2}\right)
= \cos(k\pi)\cos\left(\frac{\pi}{2}\right) - \sin(k\pi)\sin\left(\frac{\pi}{2}\right)
\]

\[
= 0
\]
The equation $\tilde{T}_n(x) = 0$ has $n$ different roots given by

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

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 $\theta \in [0, \pi]$

<table>
<thead>
<tr>
<th>$k = 1$</th>
<th>$n = 1$</th>
<th>$n = 2$</th>
<th>$n = 3$</th>
<th>...</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\frac{\pi}{2}$</td>
<td>$\frac{\pi}{4}$</td>
<td>$\frac{\pi}{6}$</td>
<td>$\frac{\pi}{2n}$</td>
<td></td>
</tr>
<tr>
<td>$k = 2$</td>
<td></td>
<td>$\frac{3\pi}{4}$</td>
<td>$\frac{\pi}{3}$</td>
<td>$\frac{3\pi}{2n}$</td>
<td></td>
</tr>
<tr>
<td>$k = 3$</td>
<td></td>
<td></td>
<td>$\frac{5\pi}{6}$</td>
<td>$\frac{5\pi}{2n}$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = n$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\left(\frac{2n-1}{2n}\right)\pi$</td>
</tr>
</tbody>
</table>
Interpolation: Choosing the nodes

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

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

- but \( \theta = \arccos x \). Thus,

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

Then the roots of the Chebyshev polynomials are

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

- Notice that the roots of \( \tilde{T}_n (x) \) are the same as the roots of \( T_n (x) \).
Interpolation: Choosing the nodes

- Plotting \( \cos(j\theta) \) for \( \theta \in [0, \pi] \) and \( j = 1, 2, \ldots, n \) in \textbf{Matlab} (Chebyshev nodes)
Interpolation: Choosing the nodes

- The following theorem summarizes some characteristics of the Chebyshev polynomials

**Theorem**

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

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

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

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

with

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

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

$$\tilde{T}_n(x_k^*) = \frac{(-1)^k}{2^{n-1}} \text{ for } k = 0, 1, ..., n$$
Interpolation: Choosing the nodes

- Extrema of Chebyshev polynomials:

\[ T_n(x) = \cos(n \arccos x) \]

then

\[ \frac{dT_n(x)}{dx} = T'_n(x) \]

\[ = -\sin(n \arccos x) \left( -\frac{n}{\sqrt{1-x^2}} \right) \]

\[ = \frac{n \sin(n \arccos x)}{\sqrt{1-x^2}} \]

- Notice that \( T'_n(x) \) is a polynomial of degree \( n - 1 \) with zeros given by

\[ T'_n(x) = 0 \]
Interpolation: Choosing the nodes

- When excluding the endpoints of the domain \((x = -1 \text{ or } x = 1)\), the extremum points occurs when

\[
\sin (n \arccos x) = 0
\]

or when

\[
\sin (n\theta) = 0
\]

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

\[
n\theta_k = k\pi \quad \text{for all } k = 1, 2, \ldots, n - 1
\]

- Solving for \(x\) yields

\[
\theta = \arccos x = \frac{k\pi}{n}
\]

\[
\implies x^*_k = \cos \left(\frac{k\pi}{n}\right) \quad \text{for } k = 1, 2, \ldots, n - 1
\]

- Obviously, extrema also occur at the endpoints of the domain (i.e., \(x = -1 \text{ or } x = 1\)). That is when \(k = 0\) or when \(k = n\).
The extremum values of \( T_n(x) \) occurs when

\[
T_n(x^*) = \cos(n \arccos x^*) \\
= \cos \left( n \arccos \left( \cos \left( \frac{k \pi}{n} \right) \right) \right) \\
= \cos \left( \frac{k \pi}{n} \right) \\
= \cos(k \pi) \\
= (-1)^k \quad \text{for } k = 0, 1, \ldots, n
\]

Notice that we are including the endpoints of the domain.

The above result implies that

\[
\max_{x \in [-1,1]} |T_n(x)| = 1
\]
Extrema for monic Chebyshev polynomials are characterized by the same points since

\[ \tilde{T}_n(x) = \frac{T_n(x)}{2^{n-1}} \]

Thus,

\[ \tilde{T}_n'(x_k^*) = T_n'(x_k^*) = 0 \quad \text{for } k = 0, 1, \ldots, n \]

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

\[ \tilde{T}_n(x_k^*) = \frac{T_n(x_k^*)}{2^{n-1}} \quad \text{for } k = 0, 1, \ldots, n \]

\[ = \frac{(-1)^k}{2^{n-1}} \quad \text{for } k = 0, 1, \ldots, n \]

Therefore

\[ \max_{x \in [-1, 1]} \left| \tilde{T}_n(x) \right| = \frac{1}{2^{n-1}} \]
An important property of monic Chebyshev polynomials is given by the following theorem:

**Theorem**

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

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

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

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

- 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

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

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

which implies that

\[
\prod_{k=0}^{n} (x - x_k) = \tilde{T}_{n+1}(x)
\]

- Therefore, the zeros of \( \prod_{k=0}^{n} (x - x_k) \) must be the zeros of \( \tilde{T}_{n+1}(x) \)

  which are given by

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

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

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

- 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

- Comparing the interpolation errors of $f(x) = \exp(-x)$ defined in $x \in [-5, 5]$ with 10-node polynomial approximation (example_miranda_cheby_nodes)

- This contains Figure 6.2 of the Miranda and Flecker book.
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 unknown coefficients.

For this case, we can use the **discrete least squares**. To do this, use \( m \) interpolation 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.
Objective: Construct a degree $n$ polynomial, $\hat{f}(x)$, that approximates the function $f$ for $x \in [a, b]$ using $m > n$ interpolation nodes.

$$\hat{f}(x) = \sum_{j=0}^{n} c_j T_j(x_k)$$
Interpolation through regression

- **Algorithm:**
  - **Step 1:** Compute the $m$ Chebyshev interpolation nodes on $[-1, 1]$:  
    \[ z_k = \cos \left( \left( \frac{2k - 1}{2m} \right) \pi \right) \quad \text{for} \quad k = 1, \ldots, m \]
    Do this as if we want an $m$–degree Chebyshev interpolation.
  - **Step 2:** Adjust the nodes to the interval $[a, b]$:  
    \[ x_k = (z_k + 1) \left( \frac{b - a}{2} \right) + a \quad \text{for} \quad k = 1, \ldots, m \]
  - **Step 3:** Evaluate $f$ at the nodes:  
    \[ y_k = f(x_k) \quad \text{for} \quad k = 1, \ldots, m \]
Algorithm (Cont.):

**Step 4: Compute the Chebyshev least squares coefficients**

The coefficients that solve the discrete LS problem

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

are given by

\[
c_j = \frac{\sum_{k=1}^{m} y_k T_j(z_k)}{\sum_{k=1}^{m} (T_j(z_k))^2} \quad \text{for } j = 0, 1, \ldots, n
\]

where \( z_k \) is the inverse transformation of \( x_k \):

\[
z_k = \frac{2x_k - (a + b)}{b - a}
\]
Finally, the Least Squares (LS) Chebyshev approximating polynomial is given by

\[ \hat{f}(x) = \sum_{j=0}^{n} c_j T_j(z) \]

where \( z \in [-1, 1] \) and is given by

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

Furthermore, \( c_j \) is estimated using the LS coefficients

\[ c_j = \frac{\sum_{k=1}^{m} y_k T_j(z_k)}{\sum_{k=1}^{m} \left(T_j(z_k)\right)^2} \quad \text{for } j = 0, 1, \ldots, n \]
Piecewise linear approximation

- If we have interpolation data given by
  \[\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}\]

- we can divide the interpolation nodes in subintervals of the form
  \[[x_i, x_{i+1}] \text{ for } i = 0, 1, \ldots, n - 1\]

- Afterwards, we can perform linear interpolation in each subinterval:
  - Interpolation conditions for each subinterval:
    \[f(x_i) = a_0 + a_1 x_i\]
    \[f(x_{i+1}) = a_0 + a_1 x_{i+1}\]
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:
    \[
    p_i(x) = f(x_i) + \left( \frac{x - x_i}{x_{i+1} - x_i} \right) (f(x_{i+1}) - f(x_i))
    \]
Example: \((x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))\)

Then we have two subintervals

\([x_0, x_1]\) and \([x_1, x_2]\)

The interpolating function is given by:

\[
\hat{f}(x) = \begin{cases} 
  f(x_0) + \left( \frac{x-x_0}{x_1-x_0} \right) (f(x_1) - f(x_0)) & \text{for } x \in [x_0, x_1] \\
  f(x_1) + \left( \frac{x-x_1}{x_2-x_1} \right) (f(x_2) - f(x_1)) & \text{for } x \in [x_1, x_2]
\end{cases}
\]
A spline is any smooth function that is a piecewise polynomial but is also smooth where the polynomial pieces connect.

Assume that the interpolation data is given by 
\( \{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_m, f(x_m))\} \).

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 \( i = 0, 1, \ldots, m - 1 \)

Notice that an order 2 spline is the piecewise linear interpolant equation.
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 \( i = 0, 1, \ldots, m - 1 \)

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

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

Notice that in this case we have 8 unknowns: \(a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1\).
Example (Cont.): We need 8 conditions

- Interpolation and continuity at interior nodes conditions

\[ y_0 = s_0 (x_0) = a_0 + b_0 x_0 + c_0 x_0^2 + d_0 x_0^3 \]
\[ y_1 = s_0 (x_1) = a_0 + b_0 x_1 + c_0 x_1^2 + d_0 x_1^3 \]
\[ y_1 = s_1 (x_1) = a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3 \]
\[ y_2 = s_1 (x_2) = a_1 + b_1 x_2 + c_1 x_2^2 + d_1 x_2^3 \]
Example (Cont.): We need 8 conditions

First and second derivatives must agree at the interior nodes

\[ s'_0(x_1) = s'_1(x_1) \]
\[ b_0 + 2c_0 x_1 + 3d_0 x_1^2 = b_1 + 2c_1 x_1 + 3d_1 x_1^2 \]

\[ s''_0(x_1) = s''_1(x_1) \]
\[ 2c_0 + 6d_0 x_1 = 2c_1 + 6d_1 x_1 \]
Up to now, 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
  \[
  s'(x_0) = \frac{s(x_1) - s(x_0)}{x_1 - x_0} \\
  s'(x_2) = \frac{s(x_2) - s(x_1)}{x_2 - x_1}
  \]
Different ways to approximate a function

Increasing the degree of interpolation does not guarantee convergence in Chebyshev nodes.

Several standards can be used. Some are easier to implement and are also less computationally costly but are not as accurate as others.

Judd’s book was published in 1998 and Miranda and Fackler’s book was published in 2002. More than a decade has passed since these books were published. There are many methods that have built on the foundations we have just discussed.
Thank you for your time! Please let me know if you have any questions.