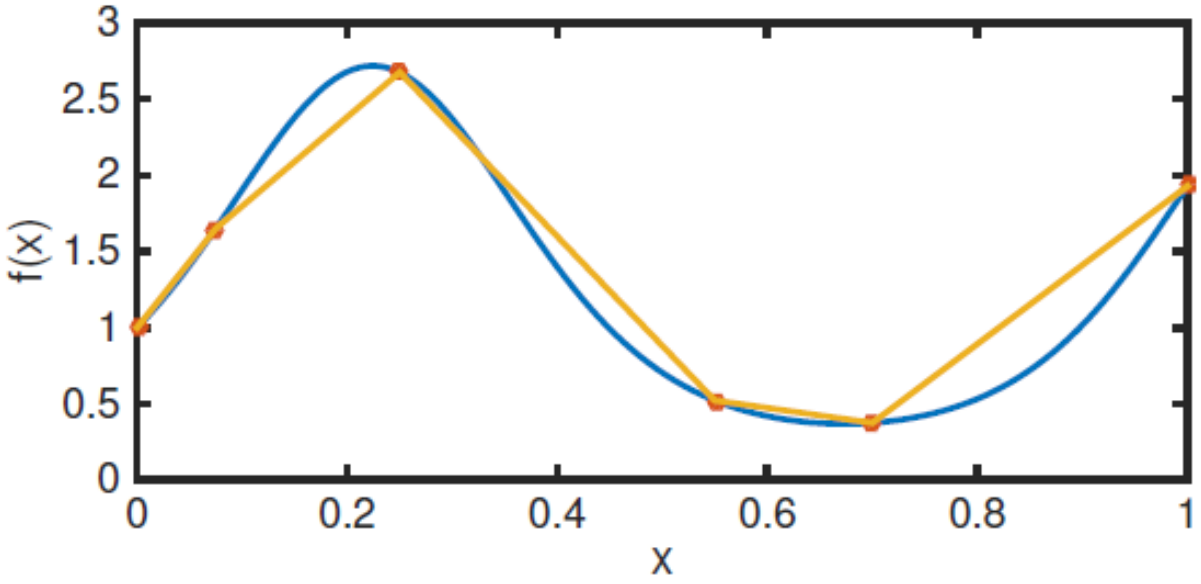
Interpolation

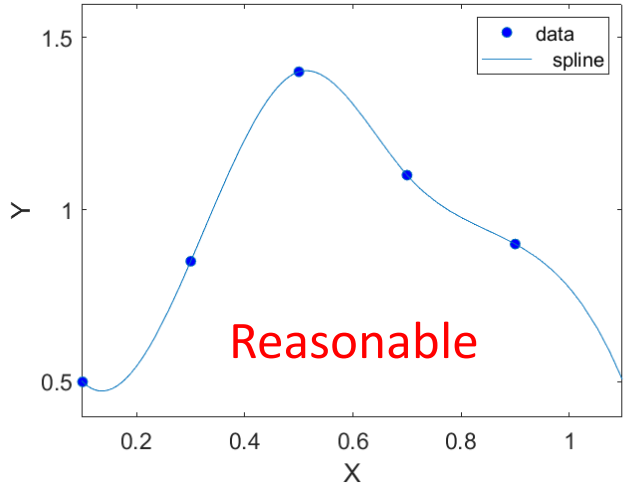
Function Approximation

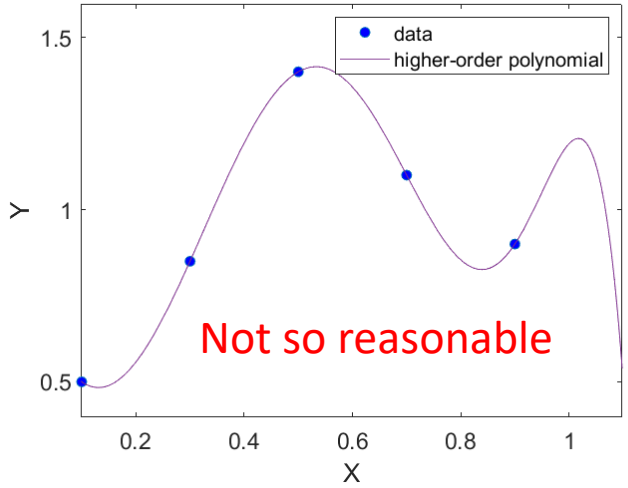


Interpolation

- Building block for various complex algorithms
  - Differentiation
  - Integration
  - Solution of differential equations
  - Signal processing
    - Trigonometric interpolation
  - Approximation theory
    - Approximate discrete data
    - Complex function
- Key idea
  - Find a curve that passes through data points
  - Data might come from any field
- We have two approaches to interpolation
  - Polynomial interpolation
  - Piecewise interpolation

In interpolation, function (interpolant) must match the data exactly





$(x_i, y_i), \qquad i = 1, \ldots m, \qquad$ with $x_1 < x_2 < \cdots x_m$

we seek a function $f: \mathbb{R} \to \mathbb{R}$ such that $\quad f(x_i) = y_i, \qquad i = 1, \ldots m$

We call function *f* the interpolant.

For complex interpolation problems additional information might be included

- Slope of interpolant at data points
- Monotonicity
- Convexity

Function (Curve) Fitting

Fitting the data exactly is not always desirable

- When data have uncertainty or sources of error

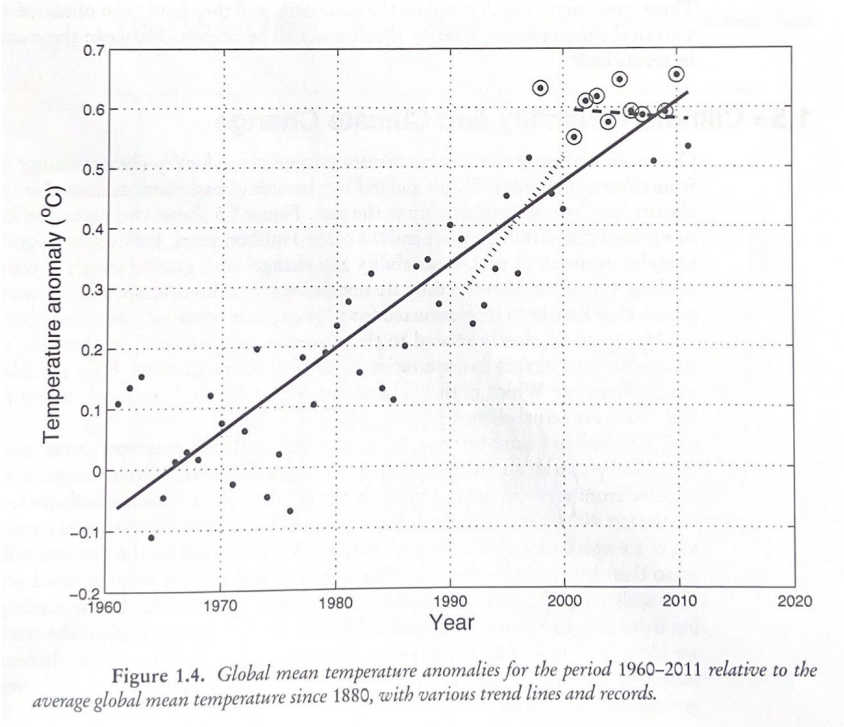
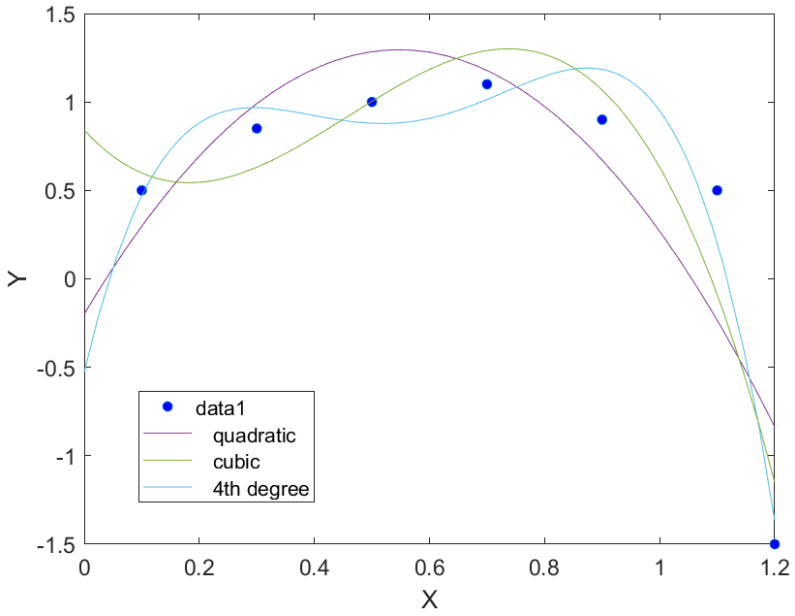


Figure 1.4. *Global mean temperature anomalies for the period* 1960–2011 *relative to the average global mean temperature since* 1880, *with various trend lines and records.*



A common approach is method of least squares (will be covered separately)

Unique Interpolating Polynomial

Theorem: For any real data points $\{(x_i, y_i)\}_{i=1}^{n}$ with distinct abscissae x_i there exists a unique polynomial interpolant $p(x)$ of degree *n-1* which satisfies the interpolation conditions

$$p(x_i) = y_i, \qquad i = 1,2, \dots n$$

In other words, no matter which method or basis is used to obtain the interpolating polynomial, the same result is obtained.

Existence & Uniqueness of Interpolants

For a given set of data points (t_i, y_i), $\quad i = 1, \ldots m,$

an interpolant is chosen from the space of functions spanned by a suitable set of *basis functions* $\phi_1(t), \ldots \phi_n(t)$

Interpolating function *f* is chosen to be a linear combination of the basis functions (a set of pre-defined shapes or patterns)

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t),$$

Number of basis functions is equal to the number of data

where the parameters x_j are to be determined. Additionally, we require *f* to interpolate the data (t_i, y_i)

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i, \qquad i = 1, \ldots m,$$

which is a system of linear equations that can be written in matrix form as

$$\mathbf{A}x = y$$

$$\mathbf{A}x = y$$

A is called the Vandermonde matrix

y (a vector of size *m*) composed of the known data values y_i

x (a vector of size *n*) are the unknown parameters to be determined

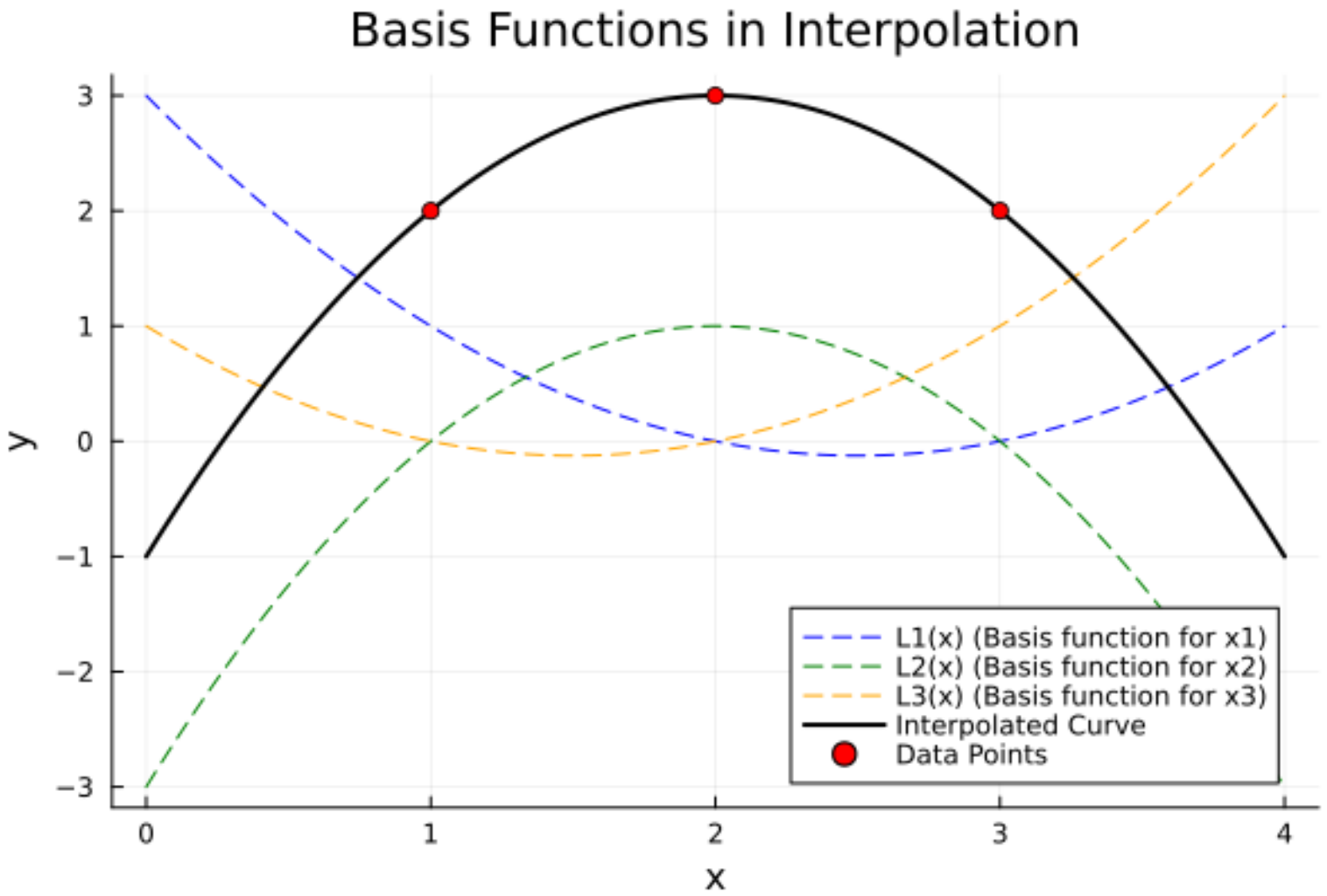
A is an $m \times n$ *basis* matrix where the entries are given by $a_{ij} = \phi_j(t_i)$. In other words, a_{ij} is the value of the *j*th *basis* function evaluated at the *i*th data point

If $n = m$ data points can be fit exactly. Need to solve a system of linear equations. Assuming A is a nonsingular matrix

If $n < m$ overdetermined system (more data than the parameters) → method of least squares. Data cannot be fitted exactly.

If $n > m$ underdetermined system (more parameters than the data) → Interpolant is nonunique. Additional properties can be useful (e.g., degree of smoothness, monotonicity, convexity)

Sensitivity of the parameters x to perturbations in the data depends on the conditioning of **A**. Choice of basis functions determines the conditioning of **A**


Basis Functions in Interpolation
y
x
3
2
1
0
−1
−2
−3
0
1
2
3
4
L1(x) (Basis function for x1)
L2(x) (Basis function for x2)
L3(x) (Basis function for x3)
Interpolated Curve
Data Points

Example: Monomial Basis

Basis function: $\phi_j(t) = t^{j-1}, \qquad j = 1, \ldots n$

Interpolant: $p_{n-1}(t) = x_1 + x_2 t + \cdots + x_n t^{n-1}$

$$\mathbf{A}x = \begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & & t_2^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & t_n & & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = y$$

Three data points: (-2,-27), (0,-1), (1,0) → m=3

$$\phi_j(t) = t^{j-1}, \quad j = 1, \ldots 3$$

$$f(t_i) = \sum_{j=1}^{m} x_j \phi_j(t_i) = y_i, \qquad i = 1, \ldots m,$$

$$f(t_i) = \sum_{j=1}^{3} x_j t^{j-1} = y_i, \qquad i = 1, \ldots 3,$$

$$p_2(t) = x_1 + x_2 t + x_2 t^2$$

$$Ax = \begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

Solve by Gauss elimination to find
$x = [-1 \quad 5 \quad -4]^T$

$$p_2(t) = -1 + 5t - 4t^2$$

Issues with $\mathbf{A}x = y$

- Polynomial order grows with more data
- Solving system of linear equations is $\mathcal{O}(n^3)$ operations
- Matrix **A** is often *ill-conditioned* for high-degree polynomials
  - As the order of polynomial increase, each polynomial becomes less distinguishable (i.e., 9th order looks like 10th order)

Are there other polynomial basis where $x_j = y_j$

Lagrange Interpolation

For a given set of data points (t_i, y_i), $\quad i = 1, \dots n$, let

$$l(t) = \prod_{k=1}^{n} (t - t_k) = (t - t_1)(t - t_2) \dots (t - t_n)$$

Barycentric formulation is more suitable for computation

Define barycentric weights $\quad w_j = \dfrac{1}{l'(t_j)} = \dfrac{1}{\prod_{k=1, k\neq j}^{n} (t_j - t_k)}, \quad j = 1,2, \dots n.$

Lagrange basis functions for $\mathbb{P}_{n-1}$, also fundamental polynomials are given by

Lagrange polynomials (basis functions): $l_j(t) = l(t) \dfrac{w_j}{t - t_j}, \quad j = 1,2, \dots n.$

Basis matrix $\boldsymbol{A}x = y$ turns out to be an identity matrix $\boldsymbol{I}$, meaning that diagonal entries are the data values y

$$p_{n-1}(t) = \sum_{j=1}^{n} y_j l_j(t) = \sum_{j=1}^{n} y_j l(t) \frac{w_j}{t - t_j} = l(t) \sum_{j=1}^{n} y_j \frac{w_j}{t - t_j}$$

Three data points: (-2,-27), (0,-1), (1,0) → n=3

$$l(t) = (t - t_1)(t - t_2)(t - t_3) = (t + 2)(t - 0)(t - 1)$$

$$w_1 = \frac{1}{(t_1 - t_2)(t_1 - t_3)} = \frac{1}{(-2 - 0)(-2 - 1)} = \frac{1}{6}$$

$$w_2 = \frac{1}{(t_2 - t_1)(t_2 - t_3)} = \frac{1}{(0 - (-2))(0 - 1)} = -\frac{1}{2}$$

$$w_3 = \frac{1}{(t_3 - t_1)(t_3 - t_2)} = \frac{1}{(1 - (-2))(1 - 0)} = \frac{1}{3}$$

$$p_2(t) = (t + 2)(t - 0)(t - 1)\left(-27\frac{1/6}{t + 2} - 1\frac{-1/2}{t} + 0\frac{1/3}{t - 1}\right)$$

In-class exercise

Evaluate and plot the following polynomial functions for any t and comment on your results

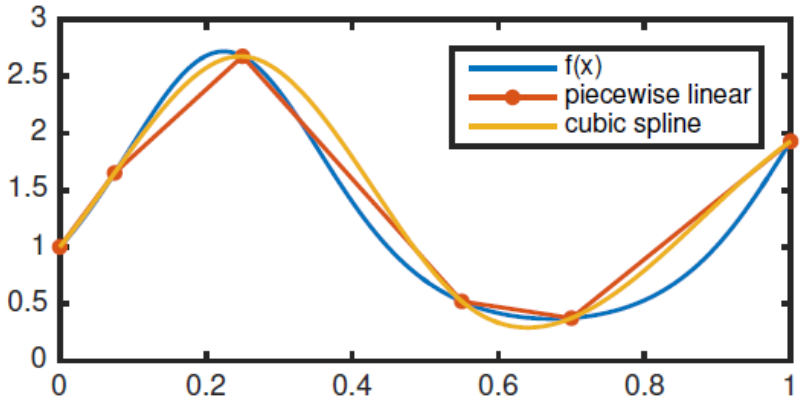
$$p_2(t) = -1 + 5t - 4t^2$$

$$p_2(t) = (t+2)(t-0)(t-1)\left(-27\frac{1/6}{t+2} - 1\frac{-1/2}{t} + 0\frac{1/3}{t-1}\right)$$

Piecewise Polynomial Interpolation

A single polynomial of degree *n* fitted to the entire data can behave poorly as data size grows. Lagrange or monomial basis.

One can use *piecewise* Lagrange polynomials with lower order to sections of the data



The problem with *piecewise* Lagrange polynomials is that it has discontinuous slopes at boundaries of the sections, which causes problems when calculating derivatives at the data points

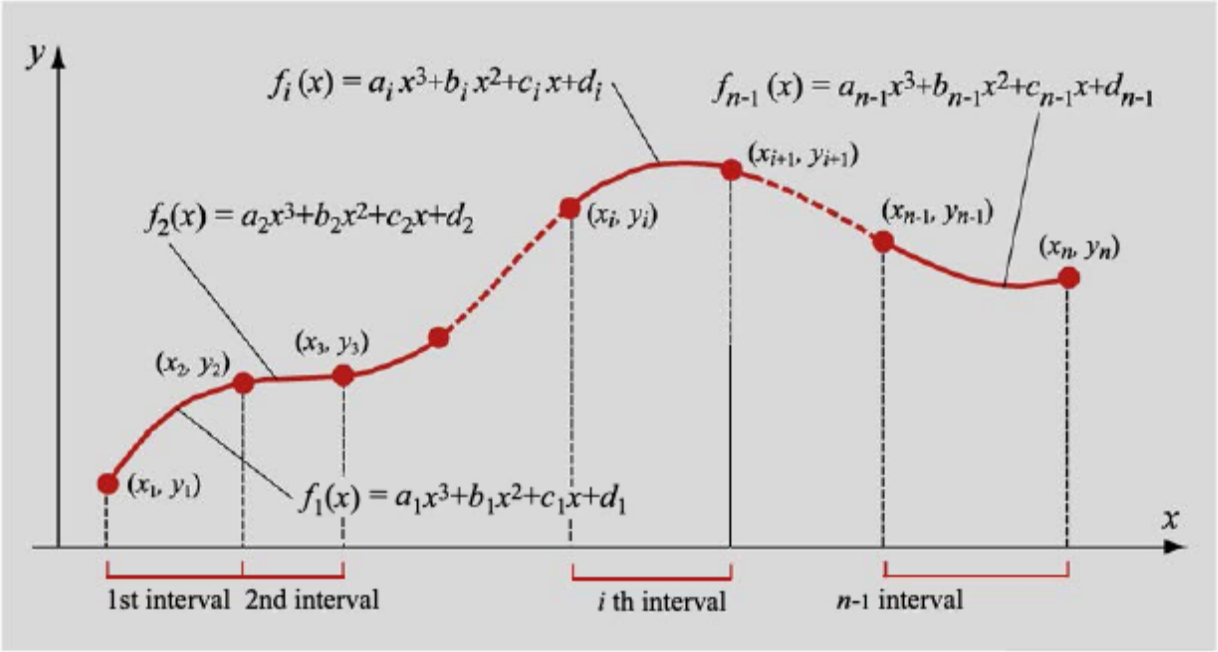
We can address this issue with cubic splines

Cubic Spline Interpolation

A *spline* is a piecewise polynomial of degree *k* that is continuously differentiable *k*-1 times.

A *linear* spline can be continuous, but it is NOT differentiable at data points

A *cubic* spline is continuous and twice differentiable at data points



Cubic Spline Interpolation

Consider $n+1$ distinct data points (nodes) with $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ with $x_0 < x_1 < \cdots < x_n$

Note: index starts from 0

Derive a third order polynomial for each interval $f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$

We have $n+1$ points and n intervals → 4n unknown constants.

We need $4n$ equations or conditions

Cubic Spline Interpolation

1. Function values must be equal at the interior nodes: 2*(n-1) equations

$$f(x_{i-1}) = a_i x^3 + b_i x^2 + c_i x + d_i$$

$$f(x_{i-1}) = a_{i-1} x_{i-1}^3 + b_{i-1} x_{i-1}^2 + c_{i-1} x_{i-1} + d_{i-1}$$

2. First and last function must pass through the end points: 2 equations

$$f(x_0) = y_o \qquad f(x_n) = y_n$$

3. First derivatives at the interior nodes are equal: (n-1) equations

$$f'(x_{i-1}) = f'(x_i)$$

4. Second derivatives at the interior nodes are equal: (n-1) equations

$$f''(x_{i-1}) = f''(x_i)$$

5. Second derivatives at the end nodes are zero: 2 equations

This is known as the *natural* splines

$$f''(x_0) = 0 \qquad f''(x_n) = 0$$

Total of 4n equations/conditions

The fifth condition can be replaced with a *not-a-knot* spline

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

$$f_1'''(x_1) = f_2'''(x_1) \qquad f_{n-1}'''(x_{n-1}) = f_n'''(x_{n-1})$$

$$a_1 = a_2 \qquad a_{n-1} = a_n$$

2 equations!

The not-a-knot condition reduces the influence of boundary points on the overall spline shape, making the curve more flexible and less biased by endpoint conditions.

Orthogonal Polynomials

- Orthogonal polynomials have many useful properties.
  - The elegant theory behind it is beyond the scope of this class.
  - We will make use of some of those properties
  - Orthogonal polynomials are particularly convenient for least squares approximation of a given function
  - Orthogonal polynomials are useful for generating Gaussian quadrature rules
  - Can represent an arbitrary but smooth function efficiently
  - Pretty much necessary when the degree of polynomial is high

Example: *three-term recurrence* property makes generation and evaluation of polynomials very efficient

$$p_{k+1}(t) = (\alpha_k t + \beta_k)p_k(t) - \gamma p_{k-1}(t)$$

Orthogonal Polynomials

Inner product (generalization of dot product) of two polynomials *p* and *q* on an interval [*a*,*b*]

Inner product: $\langle p, q \rangle = \int_a^b p(t)q(t)w(t)dt$

where $w(t)$ is nonnegative weight function

Polynomials *p* and *q* are said to be orthogonal if $\langle p, q \rangle = 0$

A set of polynomials $\{p_i\}$ is said to be orthonormal if

$$\langle p_i, p_j \rangle = \begin{cases} 1 & \text{for} \quad i = j \\ 0 & \text{for} \quad i \neq j \end{cases}$$

Methods, such as Gram-Schmidt, process can be used for orthogonalization

Example: Legendre polynomials are derived by orthogonalization of monomial polynomials $(1, t, t^2, t^3, \ldots)$

Orthogonal Polynomials

Recall $\langle p, q \rangle = \int_a^b p(t)q(t)w(t)dt$

Name	Symbol	Interval	Weight function w(t)
Legendre	P_k	[-1,1]	1
Chebyshev, 1st kind	T_k	[-1,1]	$(1-t^2)^{-1/2}$
Chebyshev, 1st kind	U_k	[-1,1]	$(1-t^2)^{1/2}$
Jacobi	J_k	[-1,1]	$(1-t)^{\alpha}(1+t)^{\beta}, \quad \alpha, \beta > 1$
Laguerre	L_k	[-1,1]	e^{-t}
Hermite	H_k	[-1,1]	e^{-t^2}

Chebyshev Polynomials

- Orthogonal polynomials
- Useful for interpolation of continuous functions
- Close to the best polynomial approximation to continuous function

Recall $\langle p, q \rangle = \int_a^b p(t)q(t)w(t)dt$

The kth Chebyshev polynomial of the 1st kind on the interval [-1,1]

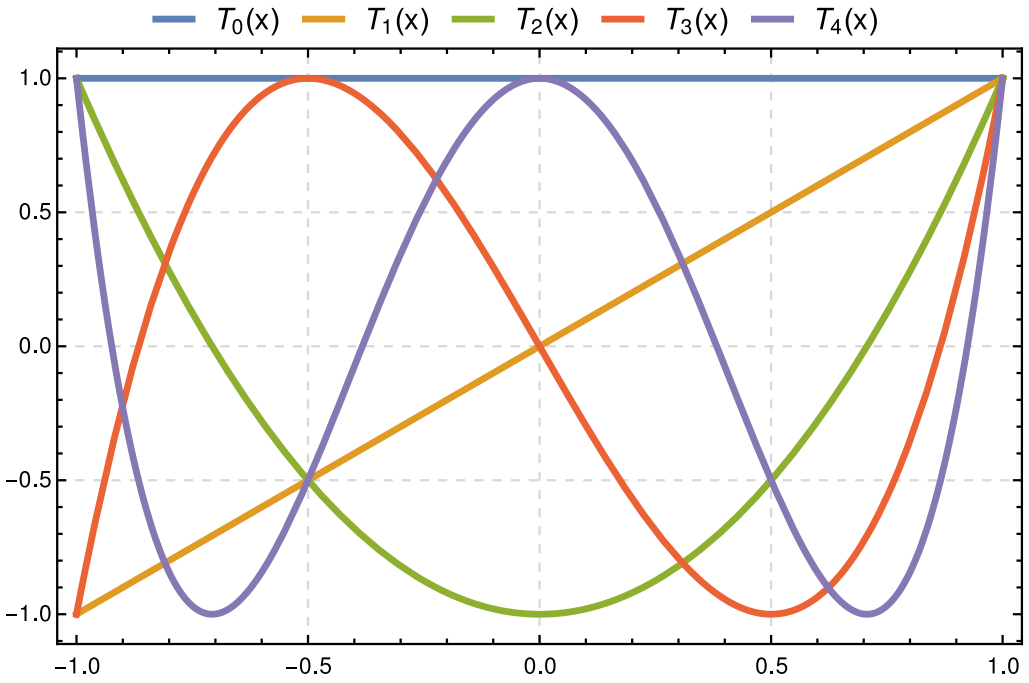
$$T_k(t) = \cos(k \arccos t)$$

With the three-term recurrence property, we can construct higher degree polynomials

$$T_{k+1}(t) = 2\,t\,T_k(t) - T_{k-1}(t) \qquad k=0,1,2,\ldots$$

In-class exercise: Form the first six Chebyshev polynomials and plot them on the interval [-1,1]

T_o	1
T_1	t
T_2	
T_3	
T_4	
T_5	



Observations:

- Degree k polynomial contains only terms of even degree if k is even, odd terms if k is odd
- Highest degree term has a coefficient 2^{k-1}
- Equi-alternation or equi-oscillation: successive extrema are equal in magnitude and alternate in sign
  - Distribute error uniformly when used to approximate continuous functions
    - Maximum error over an interval is minimized if interpolation points are the extrema of a Chebyshev polynomial (Chebyshev points)

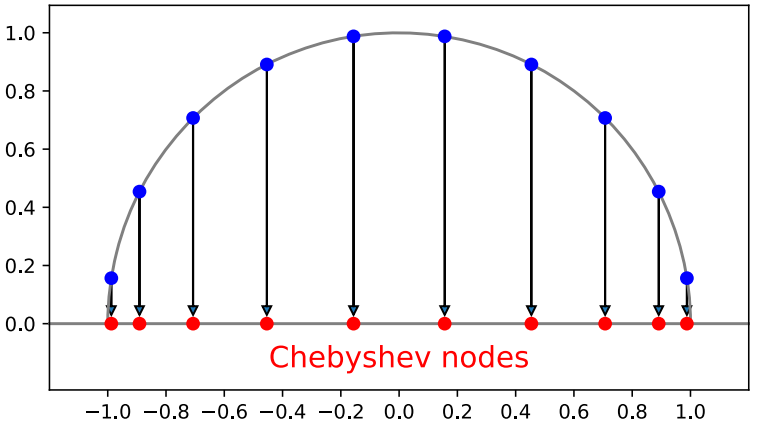
Chebyshev Points (Nodes)

k^{th} Chebyshev polynomial $\quad T_k(t) = \cos(k \arccos t)$

Its *k* zeros are given by $\quad t_i = \cos\left(\frac{(2i-1)\pi}{2k}\right), \quad i = 1,2,\dots,k$

Its *k+1* extrema (including end points) are given by $\quad t_i = \cos\left(\frac{i\pi}{k}\right), \quad i = 0,1,2,\dots k$

Chebyshev points are the abscissas of points in the plane that are equally spaced around the unit circle



Interpolation of Continuous Functions

Let us assume the interpolating points (data) are obtained from a continuous function

Let us assume that we fit a polynomial to those points

How closely the interpolant approximates the given function between the points?

n points represented by a polynomial of *n-1* degree, the error is bounded as

$$\max|f(t) - p_{n-1}(t)| \leq \frac{Mh}{n},$$

where *M* is an upper bound on the n^{th} derivative of *f(t)*

Basis Splines (B-splines)

Goal is to represent an arbitrary spline with linear combination of basis functions

B-splines can be defined through recursion, convolution, or divided differences

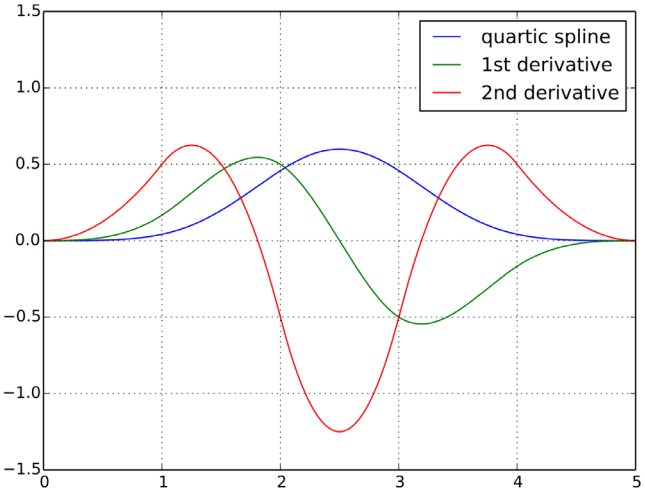
B-spline of degree *k* is *k-1* times continuously differentiable

B-splines (and Bézier curves) are used to create and manage complex shapes and surfaces using several points.

B-splines are extensively used in shape optimization

In computer aided design (CAD), computer aided manufacturing (CAM), and computer graphics, a powerful extension of B-splines is non-uniform rational B-splines (NURBS).

B-splines



$B_i^4(t)$

Linear functions: $$v_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i}$$

B-spline of degree zero: $$B_i^o(t) = \begin{cases} 1 & \text{if } t_i \le t \le t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

B-spline of degree *k* $$B_i^k(t) = v_i^k(t)\, B_i^{k-1}(t) + \left(1 - v_{i+1}^k(t)\right)\, B_{i+1}^{k-1}(t)$$

$B_i^k(t)$ is a piecewise polynomial of degree *k*

When data value at a given knot changes, coefficients of only a few basis functions change, whereas in standard polynomial, the entire interpolant is affected

Horner's Rule

$$p(x) = c_1 + c_2 x + \cdots c_n x^{n-1}$$

An efficient way to evaluate such polynomials is through nested multiplication instead of calling the function pow(x,n)

$$p(x) = \Big(\big((c_n x + c_{n-1})x + c_{n-2}\big)x + \cdots + c_2\Big)x + c_1$$

```
"""
    horner(c,x)

Evaluate a polynomial whose coefficients are given in ascending
order in `c`, at the point `x`, using Horner's rule.
"""
function horner(c,x)
    n = length(c)
    y = c[n]
    for k in n-1:-1:1
        y = x*y + c[k]
    end
    return y
end
```

Multi-dimensional Interpolation

What if the function we want to interpolate represents a surface?

Example: Consider a photograph with many pixels.

$$p(x_i, y_i) = f_i, \qquad i = 0,1,2, \ldots n$$

$$p(x, y) = \sum_{k=0}^{N_x} \sum_{l=0}^{N_y} c_k \phi_k(x) \psi_l(y)$$

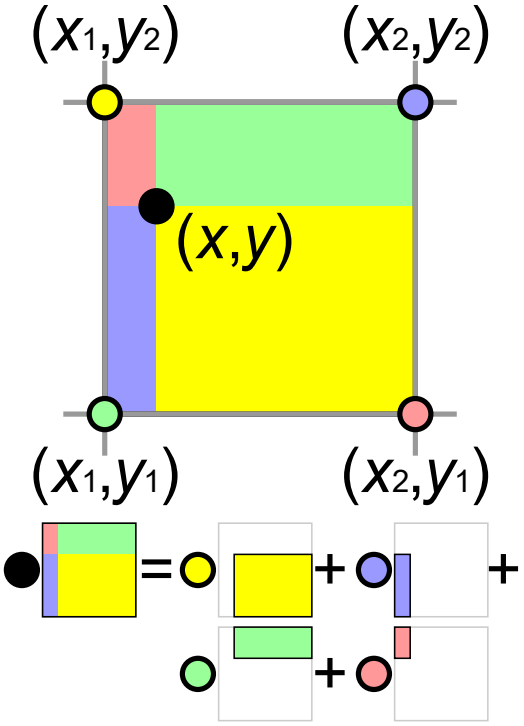
$\phi(x)$ & $\psi\,(y)$ are one-dimensional basis functions

Bilinear Interpolation

$$p(x, y) = c_0 + c_1 x + c_2 y + c_3 xy$$

Can be calculated through different ways

1. Polynomial fit
2. Repeated linear interpolation
3. Weighted mean



Multivariate Interpolation

- Trilinear interpolation
  - *n*-linear interpolation for any dimensions
- Nearest neighbor
- Kriging
- Inverse distance weighting
- Radial basis functions
- Delaunay triangulation
  - Construct a triangulation for the scattered data
  - Construct a piecewise polynomial interpolant for each triangle
- and several other methods …