

# Scientific Computing: An Introductory Survey

## Chapter 7 – Interpolation

Prof. Michael T. Heath

Department of Computer Science  
University of Illinois at Urbana-Champaign

Copyright © 2002. Reproduction permitted  
for noncommercial, educational use only.



# Outline

- 1 Interpolation
- 2 Polynomial Interpolation
- 3 Piecewise Polynomial Interpolation

# Interpolation

- Basic interpolation problem: for given data

$$(t_1, y_1), (t_2, y_2), \dots (t_m, y_m) \quad \text{with} \quad t_1 < t_2 < \dots < t_m$$

determine function  $f: \mathbb{R} \rightarrow \mathbb{R}$  such that

$$f(t_i) = y_i, \quad i = 1, \dots, m$$

- $f$  is *interpolating function*, or *interpolant*, for given data
- Additional data might be prescribed, such as slope of interpolant at given points
- Additional constraints might be imposed, such as smoothness, monotonicity, or convexity of interpolant
- $f$  could be function of more than one variable, but we will consider only one-dimensional case

# Purposes for Interpolation

- Plotting smooth curve through discrete data points
- Reading between lines of table
- Differentiating or integrating tabular data
- Quick and easy evaluation of mathematical function
- Replacing complicated function by simple one

# Interpolation vs Approximation

- By definition, interpolating function fits given data points exactly
- Interpolation is inappropriate if data points subject to significant errors
- It is usually preferable to smooth noisy data, for example by least squares approximation
- Approximation is also more appropriate for special function libraries

# Issues in Interpolation

Arbitrarily many functions interpolate given set of data points

- What form should interpolating function have?
- How should interpolant behave between data points?
- Should interpolant inherit properties of data, such as monotonicity, convexity, or periodicity?
- Are parameters that define interpolating function meaningful?
- If function and data are plotted, should results be visually pleasing?

# Choosing Interpolant

Choice of function for interpolation based on

- How easy interpolating function is to work with
  - determining its parameters
  - evaluating interpolant
  - differentiating or integrating interpolant
- How well properties of interpolant match properties of data to be fit (smoothness, monotonicity, convexity, periodicity, etc.)

# Functions for Interpolation

- Families of functions commonly used for interpolation include
  - Polynomials
  - Piecewise polynomials
  - Trigonometric functions
  - Exponential functions
  - Rational functions
- For now we will focus on interpolation by polynomials and piecewise polynomials
- We will consider trigonometric interpolation (DFT) later

# Basis Functions

- Family of functions for interpolating given data points is spanned by set of **basis functions**  $\phi_1(t), \dots, \phi_n(t)$
- Interpolating function  $f$  is chosen as linear combination of basis functions,

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

- Requiring  $f$  to interpolate data  $(t_i, y_i)$  means

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

which is system of linear equations  $Ax = y$  for  $n$ -vector  $x$  of parameters  $x_j$ , where entries of  $m \times n$  matrix  $A$  are given by  $a_{ij} = \phi_j(t_i)$

# Existence, Uniqueness, and Conditioning

- Existence and uniqueness of interpolant depend on number of data points  $m$  and number of basis functions  $n$
- If  $m > n$ , interpolant usually doesn't exist
- If  $m < n$ , interpolant is not unique
- If  $m = n$ , then basis matrix  $A$  is nonsingular provided data points  $t_i$  are distinct, so data can be fit exactly
- Sensitivity of parameters  $x$  to perturbations in data depends on  $\text{cond}(A)$ , which depends in turn on choice of basis functions

# Polynomial Interpolation

- Simplest and most common type of interpolation uses polynomials
- Unique polynomial of degree at most  $n - 1$  passes through  $n$  data points  $(t_i, y_i)$ ,  $i = 1, \dots, n$ , where  $t_i$  are distinct
- There are many ways to represent or compute interpolating polynomial, but in theory all must give same result

< interactive example >



# Monomial Basis

- *Monomial basis functions*

$$\phi_j(t) = t^{j-1}, \quad j = 1, \dots, n$$

give interpolating polynomial of form

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

with coefficients  $x$  given by  $n \times n$  linear system

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

- Matrix of this form is called *Vandermonde matrix*



## Example: Monomial Basis

- Determine polynomial of degree two interpolating three data points  $(-2, -27)$ ,  $(0, -1)$ ,  $(1, 0)$
- Using monomial basis, linear system is

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \mathbf{y}$$

- For these particular data, system is

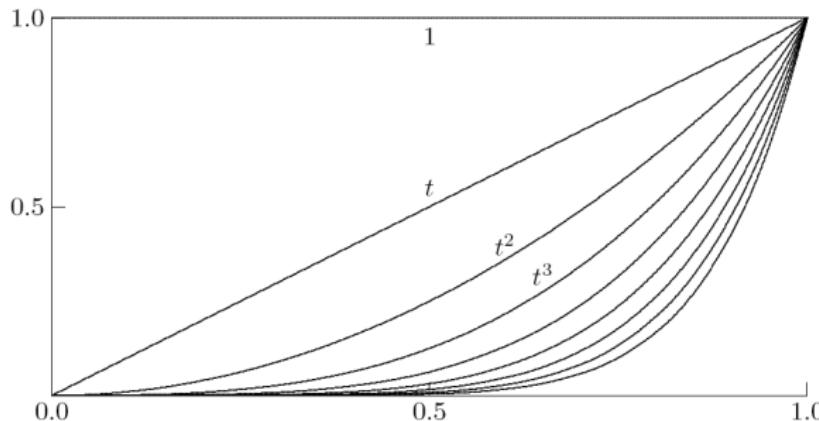
$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

whose solution is  $\mathbf{x} = [-1 \quad 5 \quad -4]^T$ , so interpolating polynomial is

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



# Monomial Basis, continued



< interactive example >

- Solving system  $Ax = y$  using standard linear equation solver to determine coefficients  $x$  of interpolating polynomial requires  $\mathcal{O}(n^3)$  work

# Monomial Basis, continued

- For monomial basis, matrix  $A$  is increasingly ill-conditioned as degree increases
- Ill-conditioning does not prevent fitting data points well, since residual for linear system solution will be small
- But it does mean that values of coefficients are poorly determined
- Both conditioning of linear system and amount of computational work required to solve it can be improved by using different basis
- Change of basis still gives same interpolating polynomial for given data, but *representation* of polynomial will be different

# Monomial Basis, continued

- Conditioning with monomial basis can be improved by shifting and scaling independent variable  $t$

$$\phi_j(t) = \left( \frac{t - c}{d} \right)^{j-1}$$

where,  $c = (t_1 + t_n)/2$  is midpoint and  $d = (t_n - t_1)/2$  is half of range of data

- New independent variable lies in interval  $[-1, 1]$ , which also helps avoid overflow or harmful underflow
- Even with optimal shifting and scaling, monomial basis usually is still poorly conditioned, and we must seek better alternatives

< interactive example >



# Evaluating Polynomials

- When represented in monomial basis, polynomial

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

can be evaluated efficiently using *Horner's nested evaluation* scheme

$$p_{n-1}(t) = x_1 + t(x_2 + t(x_3 + t(\cdots (x_{n-1} + tx_n) \cdots)))$$

which requires only  $n$  additions and  $n$  multiplications

- For example,

$$1 - 4t + 5t^2 - 2t^3 + 3t^4 = 1 + t(-4 + t(5 + t(-2 + 3t)))$$

- Other manipulations of interpolating polynomial, such as differentiation or integration, are also relatively easy with monomial basis representation

# Lagrange Interpolation

- For given set of data points  $(t_i, y_i)$ ,  $i = 1, \dots, n$ , **Lagrange basis functions** are defined by

$$\ell_j(t) = \prod_{k=1, k \neq j}^n (t - t_k) / \prod_{k=1, k \neq j}^n (t_j - t_k), \quad j = 1, \dots, n$$

- For Lagrange basis,

$$\ell_j(t_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \quad i, j = 1, \dots, n$$

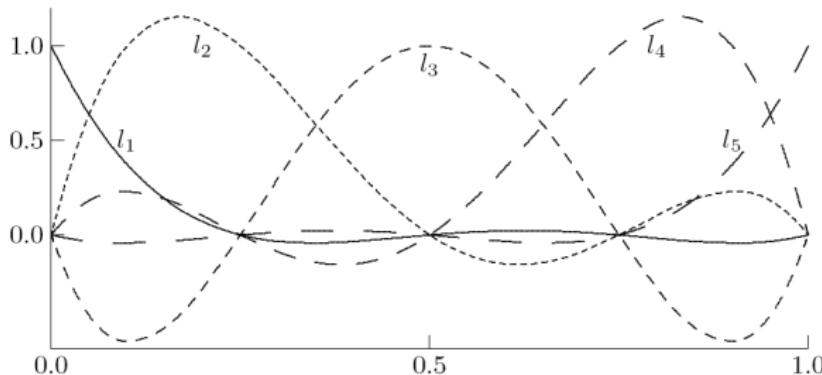
so matrix of linear system  $\mathbf{A}\mathbf{x} = \mathbf{y}$  is identity matrix

- Thus, Lagrange polynomial interpolating data points  $(t_i, y_i)$  is given by

$$p_{n-1}(t) = y_1\ell_1(t) + y_2\ell_2(t) + \cdots + y_n\ell_n(t)$$



# Lagrange Basis Functions



< interactive example >

- Lagrange interpolant is easy to determine but more expensive to evaluate for given argument, compared with monomial basis representation
- Lagrangian form is also more difficult to differentiate, integrate, etc.

# Example: Lagrange Interpolation

- Use Lagrange interpolation to determine interpolating polynomial for three data points  $(-2, -27)$ ,  $(0, -1)$ ,  $(1, 0)$
- Lagrange polynomial of degree two interpolating three points  $(t_1, y_1)$ ,  $(t_2, y_2)$ ,  $(t_3, y_3)$  is given by  $p_2(t) =$

$$y_1 \frac{(t - t_2)(t - t_3)}{(t_1 - t_2)(t_1 - t_3)} + y_2 \frac{(t - t_1)(t - t_3)}{(t_2 - t_1)(t_2 - t_3)} + y_3 \frac{(t - t_1)(t - t_2)}{(t_3 - t_1)(t_3 - t_2)}$$

- For these particular data, this becomes

$$p_2(t) = -27 \frac{t(t - 1)}{(-2)(-2 - 1)} + (-1) \frac{(t + 2)(t - 1)}{(2)(-1)}$$

# Newton Interpolation

- For given set of data points  $(t_i, y_i)$ ,  $i = 1, \dots, n$ , **Newton basis functions** are defined by

$$\pi_j(t) = \prod_{k=1}^{j-1} (t - t_k), \quad j = 1, \dots, n$$

where value of product is taken to be 1 when limits make it vacuous

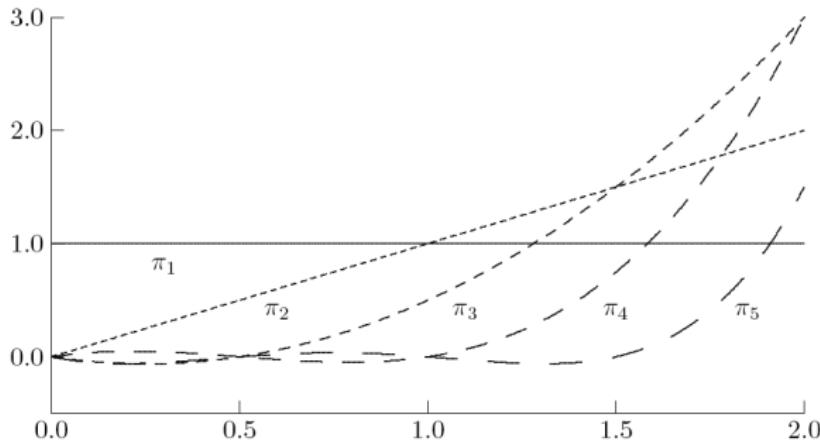
- Newton interpolating polynomial has form

$$\begin{aligned} p_{n-1}(t) &= x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + \\ &\quad \cdots + x_n(t - t_1)(t - t_2) \cdots (t - t_{n-1}) \end{aligned}$$

- For  $i < j$ ,  $\pi_j(t_i) = 0$ , so basis matrix  $A$  is lower triangular, where  $a_{ij} = \pi_j(t_i)$



# Newton Basis Functions



< interactive example >

# Newton Interpolation, continued

- Solution  $x$  to system  $Ax = y$  can be computed by forward-substitution in  $\mathcal{O}(n^2)$  arithmetic operations
- Moreover, resulting interpolant can be evaluated efficiently for any argument by nested evaluation scheme similar to Horner's method
- Newton interpolation has better balance between cost of computing interpolant and cost of evaluating it

# Example: Newton Interpolation

- Use Newton interpolation to determine interpolating polynomial for three data points  $(-2, -27)$ ,  $(0, -1)$ ,  $(1, 0)$
- Using Newton basis, linear system is

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & t_2 - t_1 & 0 \\ 1 & t_3 - t_1 & (t_3 - t_1)(t_3 - t_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

- For these particular data, system is

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

whose solution by forward substitution is

$x = [-27 \quad 13 \quad -4]^T$ , so interpolating polynomial is

$$p(t) = -27 + 13(t + 2) - 4(t + 2)t$$



# Newton Interpolation, continued

- If  $p_j(t)$  is polynomial of degree  $j - 1$  interpolating  $j$  given points, then for any constant  $x_{j+1}$ ,

$$p_{j+1}(t) = p_j(t) + x_{j+1}\pi_{j+1}(t)$$

is polynomial of degree  $j$  that also interpolates same  $j$  points

- Free parameter  $x_{j+1}$  can then be chosen so that  $p_{j+1}(t)$  interpolates  $y_{j+1}$ ,

$$x_{j+1} = \frac{y_{j+1} - p_j(t_{j+1})}{\pi_{j+1}(t_{j+1})}$$

- Newton interpolation begins with constant polynomial  $p_1(t) = y_1$  interpolating first data point and then successively incorporates each remaining data point into interpolant

< interactive example >



# Divided Differences

- Given data points  $(t_i, y_i)$ ,  $i = 1, \dots, n$ , **divided differences**, denoted by  $f[ ]$ , are defined recursively by

$$f[t_1, t_2, \dots, t_k] = \frac{f[t_2, t_3, \dots, t_k] - f[t_1, t_2, \dots, t_{k-1}]}{t_k - t_1}$$

where recursion begins with  $f[t_k] = y_k$ ,  $k = 1, \dots, n$

- Coefficient of  $j$ th basis function in Newton interpolant is given by

$$x_j = f[t_1, t_2, \dots, t_j]$$

- Recursion requires  $\mathcal{O}(n^2)$  arithmetic operations to compute coefficients of Newton interpolant, but is less prone to overflow or underflow than direct formation of triangular Newton basis matrix



# Orthogonal Polynomials

- Inner product can be defined on space of polynomials on interval  $[a, b]$  by taking

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

where  $w(t)$  is nonnegative *weight function*

- Two polynomials  $p$  and  $q$  are *orthogonal* if  $\langle p, q \rangle = 0$
- Set of polynomials  $\{p_i\}$  is *orthonormal* if

$$\langle p_i, p_j \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- Given set of polynomials, Gram-Schmidt orthogonalization can be used to generate orthonormal set spanning same space

# Orthogonal Polynomials, continued

- For example, with inner product given by weight function  $w(t) \equiv 1$  on interval  $[-1, 1]$ , applying Gram-Schmidt process to set of monomials  $1, t, t^2, t^3, \dots$  yields *Legendre polynomials*

$$1, \quad t, \quad (3t^2 - 1)/2, \quad (5t^3 - 3t)/2, \quad (35t^4 - 30t^2 + 3)/8,$$

$$(63t^5 - 70t^3 + 15t)/8, \dots$$

first  $n$  of which form an orthogonal basis for space of polynomials of degree at most  $n - 1$

- Other choices of weight functions and intervals yield other orthogonal polynomials, such as Chebyshev, Jacobi, Laguerre, and Hermite

# Orthogonal Polynomials, continued

- Orthogonal polynomials have many useful properties
- They satisfy three-term recurrence relation of form

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

which makes them very efficient to generate and evaluate

- Orthogonality makes them very natural for least squares approximation, and they are also useful for generating Gaussian quadrature rules, which we will see later

# Chebyshev Polynomials

- $k$ th *Chebyshev polynomial* of first kind, defined on interval  $[-1, 1]$  by

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

are orthogonal with respect to weight function  $(1 - t^2)^{-1/2}$

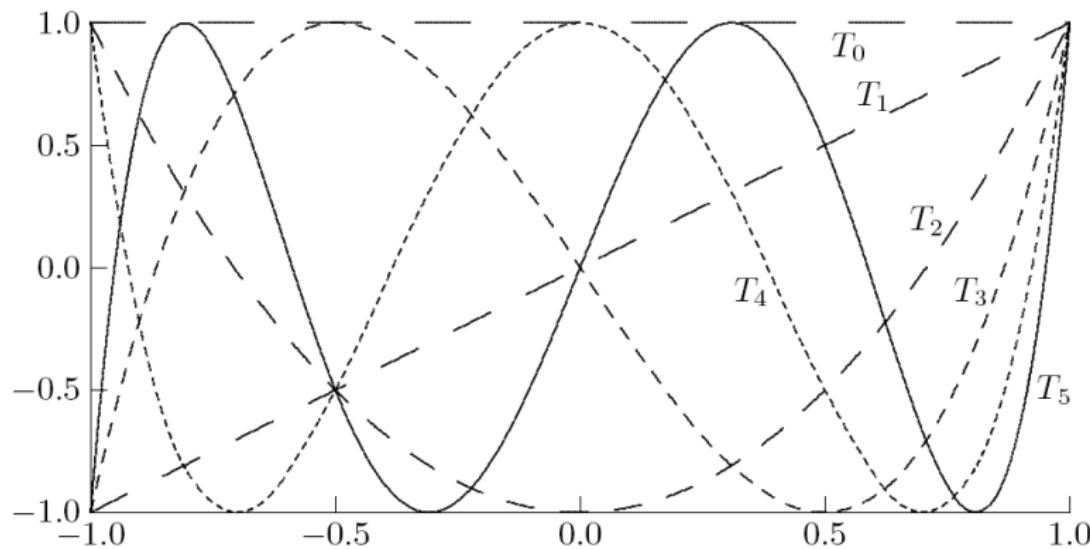
- First few Chebyshev polynomials are given by

$$1, \quad t, \quad 2t^2 - 1, \quad 4t^3 - 3t, \quad 8t^4 - 8t^2 + 1, \quad 16t^5 - 20t^3 + 5t, \quad \dots$$

- *Equi-oscillation property*: successive extrema of  $T_k$  are equal in magnitude and alternate in sign, which distributes error uniformly when approximating arbitrary continuous function



# Chebyshev Basis Functions



< interactive example >



# Chebyshev Points

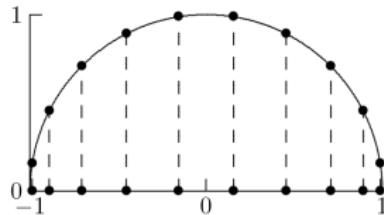
- *Chebyshev points* are zeros of  $T_k$ , given by

$$t_i = \cos\left(\frac{(2i-1)\pi}{2k}\right), \quad i = 1, \dots, k$$

or extrema of  $T_k$ , given by

$$t_i = \cos\left(\frac{i\pi}{k}\right), \quad i = 0, 1, \dots, k$$

- Chebyshev points are abscissas of points equally spaced around unit circle in  $\mathbb{R}^2$



- Chebyshev points have attractive properties for interpolation and other problems

# Interpolating Continuous Functions

- If data points are discrete sample of continuous function, how well does interpolant approximate that function between sample points?
- If  $f$  is smooth function, and  $p_{n-1}$  is polynomial of degree at most  $n - 1$  interpolating  $f$  at  $n$  points  $t_1, \dots, t_n$ , then

$$f(t) - p_{n-1}(t) = \frac{f^{(n)}(\theta)}{n!} (t - t_1)(t - t_2) \cdots (t - t_n)$$

where  $\theta$  is some (unknown) point in interval  $[t_1, t_n]$

- Since point  $\theta$  is unknown, this result is not particularly useful unless bound on appropriate derivative of  $f$  is known

# Interpolating Continuous Functions, continued

- If  $|f^{(n)}(t)| \leq M$  for all  $t \in [t_1, t_n]$ , and  $h = \max\{t_{i+1} - t_i : i = 1, \dots, n-1\}$ , then

$$\max_{t \in [t_1, t_n]} |f(t) - p_{n-1}(t)| \leq \frac{Mh^n}{4n}$$

- Error diminishes with increasing  $n$  and decreasing  $h$ , but only if  $|f^{(n)}(t)|$  does not grow too rapidly with  $n$

< interactive example >



# High-Degree Polynomial Interpolation

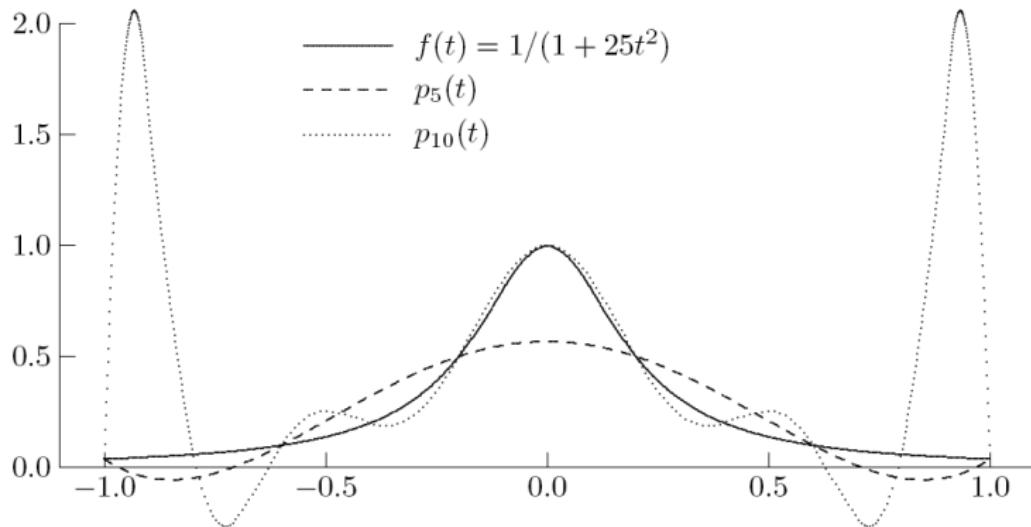
- Interpolating polynomials of high degree are expensive to determine and evaluate
- In some bases, coefficients of polynomial may be poorly determined due to ill-conditioning of linear system to be solved
- High-degree polynomial necessarily has lots of “wiggles,” which may bear no relation to data to be fit
- Polynomial passes through required data points, but it may oscillate wildly between data points

# Convergence

- Polynomial interpolating continuous function may not converge to function as number of data points and polynomial degree increases
- Equally spaced interpolation points often yield unsatisfactory results near ends of interval
- If points are bunched near ends of interval, more satisfactory results are likely to be obtained with polynomial interpolation
- Use of Chebyshev points distributes error evenly and yields convergence throughout interval for any sufficiently smooth function

# Example: Runge's Function

- Polynomial interpolants of Runge's function at *equally spaced points* ***do not*** converge

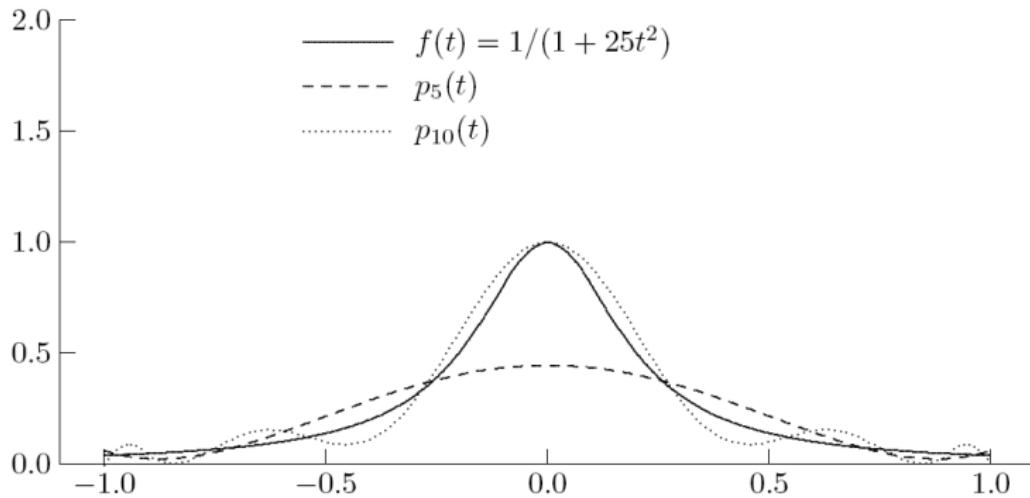


< interactive example >



# Example: Runge's Function

- Polynomial interpolants of Runge's function at *Chebyshev* points *do* converge



< interactive example >



# Taylor Polynomial

- Another useful form of polynomial interpolation for smooth function  $f$  is polynomial given by truncated Taylor series

$$p_n(t) = f(a) + f'(a)(t-a) + \frac{f''(a)}{2}(t-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(t-a)^n$$

- Polynomial interpolates  $f$  in that values of  $p_n$  and its first  $n$  derivatives match those of  $f$  and its first  $n$  derivatives evaluated at  $t = a$ , so  $p_n(t)$  is good approximation to  $f(t)$  for  $t$  near  $a$
- We have already seen examples in Newton's method for nonlinear equations and optimization

< interactive example >



# Piecewise Polynomial Interpolation

- Fitting single polynomial to large number of data points is likely to yield unsatisfactory oscillating behavior in interpolant
- Piecewise polynomials provide alternative to practical and theoretical difficulties with high-degree polynomial interpolation
- Main advantage of piecewise polynomial interpolation is that large number of data points can be fit with low-degree polynomials
- In piecewise interpolation of given data points  $(t_i, y_i)$ , *different* function is used in each subinterval  $[t_i, t_{i+1}]$
- Abscissas  $t_i$  are called *knots* or *breakpoints*, at which interpolant changes from one function to another

# Piecewise Interpolation, continued

- Simplest example is piecewise linear interpolation, in which successive pairs of data points are connected by straight lines
- Although piecewise interpolation eliminates excessive oscillation and nonconvergence, it appears to sacrifice smoothness of interpolating function
- We have many degrees of freedom in choosing piecewise polynomial interpolant, however, which can be exploited to obtain smooth interpolating function despite its piecewise nature

< interactive example >



# Hermite Interpolation

- In *Hermite interpolation*, derivatives as well as values of interpolating function are taken into account
- Including derivative values adds more equations to linear system that determines parameters of interpolating function
- To have unique solution, number of equations must equal number of parameters to be determined
- Piecewise cubic polynomials are typical choice for Hermite interpolation, providing flexibility, simplicity, and efficiency



# Hermite Cubic Interpolation

- *Hermite cubic interpolant* is piecewise cubic polynomial interpolant with continuous first derivative
- Piecewise cubic polynomial with  $n$  knots has  $4(n - 1)$  parameters to be determined
- Requiring that it interpolate given data gives  $2(n - 1)$  equations
- Requiring that it have one continuous derivative gives  $n - 2$  additional equations, or total of  $3n - 4$ , which still leaves  $n$  free parameters
- Thus, Hermite cubic interpolant is not unique, and remaining free parameters can be chosen so that result satisfies additional constraints

# Cubic Spline Interpolation

- *Spline* is piecewise polynomial of degree  $k$  that is  $k - 1$  times continuously differentiable
- For example, linear spline is of degree 1 and has 0 continuous derivatives, i.e., it is continuous, but not smooth, and could be described as “broken line”
- *Cubic spline* is piecewise cubic polynomial that is twice continuously differentiable
- As with Hermite cubic, interpolating given data and requiring one continuous derivative imposes  $3n - 4$  constraints on cubic spline
- Requiring continuous second derivative imposes  $n - 2$  additional constraints, leaving 2 remaining free parameters

# Cubic Splines, continued

Final two parameters can be fixed in various ways

- Specify first derivative at endpoints  $t_1$  and  $t_n$
- Force second derivative to be zero at endpoints, which gives *natural spline*
- Enforce “not-a-knot” condition, which forces two consecutive cubic pieces to be same
- Force first derivatives, as well as second derivatives, to match at endpoints  $t_1$  and  $t_n$  (if spline is to be periodic)

# Example: Cubic Spline Interpolation

- Determine natural cubic spline interpolating three data points  $(t_i, y_i)$ ,  $i = 1, 2, 3$
- Required interpolant is piecewise cubic function defined by separate cubic polynomials in each of two intervals  $[t_1, t_2]$  and  $[t_2, t_3]$
- Denote these two polynomials by

$$p_1(t) = \alpha_1 + \alpha_2 t + \alpha_3 t^2 + \alpha_4 t^3$$

$$p_2(t) = \beta_1 + \beta_2 t + \beta_3 t^2 + \beta_4 t^3$$

- Eight parameters are to be determined, so we need eight equations

## Example, continued

- Requiring first cubic to interpolate data at end points of first interval  $[t_1, t_2]$  gives two equations

$$\alpha_1 + \alpha_2 t_1 + \alpha_3 t_1^2 + \alpha_4 t_1^3 = y_1$$

$$\alpha_1 + \alpha_2 t_2 + \alpha_3 t_2^2 + \alpha_4 t_2^3 = y_2$$

- Requiring second cubic to interpolate data at end points of second interval  $[t_2, t_3]$  gives two equations

$$\beta_1 + \beta_2 t_2 + \beta_3 t_2^2 + \beta_4 t_2^3 = y_2$$

$$\beta_1 + \beta_2 t_3 + \beta_3 t_3^2 + \beta_4 t_3^3 = y_3$$

- Requiring first derivative of interpolant to be continuous at  $t_2$  gives equation

$$\alpha_2 + 2\alpha_3 t_2 + 3\alpha_4 t_2^2 = \beta_2 + 2\beta_3 t_2 + 3\beta_4 t_2^2$$



## Example, continued

- Requiring second derivative of interpolant function to be continuous at  $t_2$  gives equation

$$2\alpha_3 + 6\alpha_4 t_2 = 2\beta_3 + 6\beta_4 t_2$$

- Finally, by definition natural spline has second derivative equal to zero at endpoints, which gives two equations

$$2\alpha_3 + 6\alpha_4 t_1 = 0$$

$$2\beta_3 + 6\beta_4 t_3 = 0$$

- When particular data values are substituted for  $t_i$  and  $y_i$ , system of eight linear equations can be solved for eight unknown parameters  $\alpha_i$  and  $\beta_i$



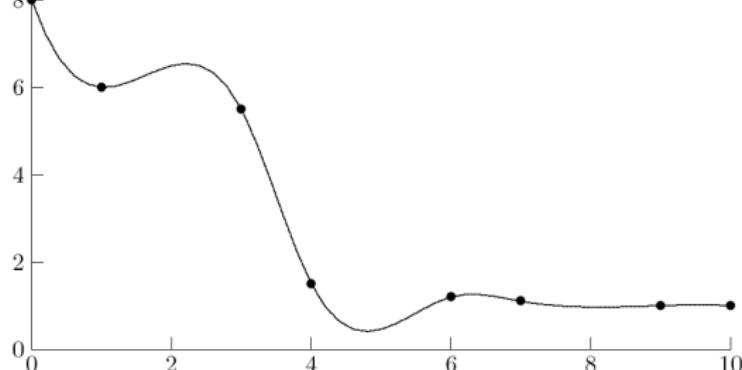
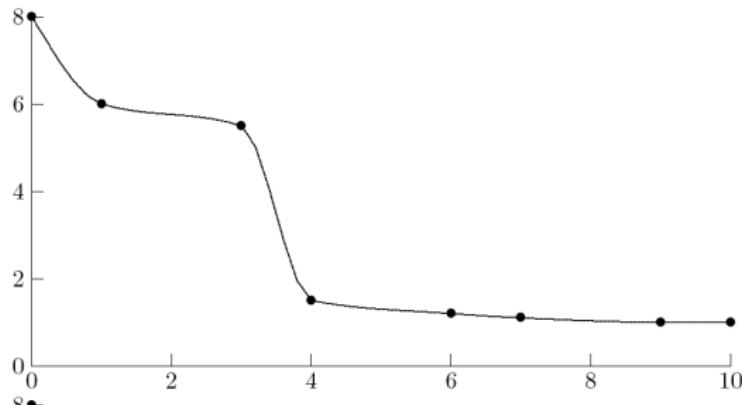
# Hermite Cubic vs Spline Interpolation

- Choice between Hermite cubic and spline interpolation depends on data to be fit and on purpose for doing interpolation
- If smoothness is of paramount importance, then spline interpolation may be most appropriate
- But Hermite cubic interpolant may have more pleasing visual appearance and allows flexibility to preserve monotonicity if original data are monotonic
- In any case, it is advisable to plot interpolant and data to help assess how well interpolating function captures behavior of original data

< interactive example >



# Hermite Cubic vs Spline Interpolation



# B-splines

- *B-splines* form basis for family of spline functions of given degree
- B-splines can be defined in various ways, including recursion (which we will use), convolution, and divided differences
- Although in practice we use only finite set of knots  $t_1, \dots, t_n$ , for notational convenience we will assume infinite set of knots

$$\cdots < t_{-2} < t_{-1} < t_0 < t_1 < t_2 < \cdots$$

Additional knots can be taken as arbitrarily defined points outside interval  $[t_1, t_n]$

- We will also use linear functions

$$v_i^k(t) = (t - t_i)/(t_{i+k} - t_i)$$



# B-splines, continued

- To start recursion, define B-splines of degree 0 by

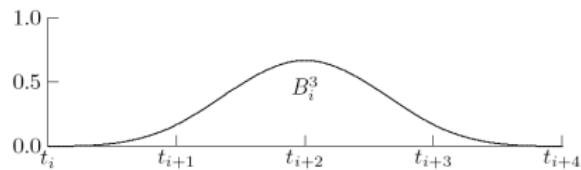
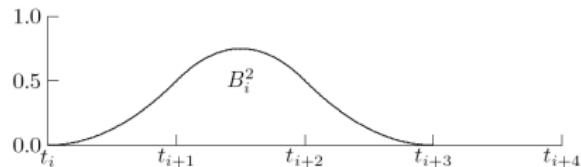
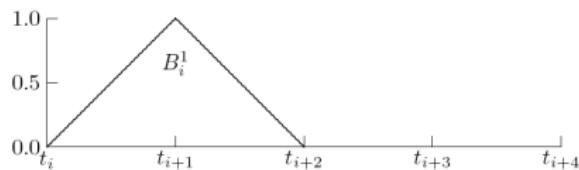
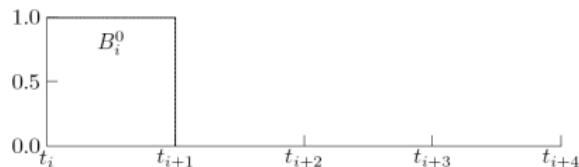
$$B_i^0(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and then for  $k > 0$  define B-splines of degree  $k$  by

$$B_i^k(t) = v_i^k(t)B_i^{k-1}(t) + (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t)$$

- Since  $B_i^0$  is piecewise constant and  $v_i^k$  is linear,  $B_i^1$  is piecewise linear
- Similarly,  $B_i^2$  is in turn piecewise quadratic, and in general,  $B_i^k$  is piecewise polynomial of degree  $k$

# B-splines, continued



< interactive example >

# B-splines, continued

Important properties of B-spline functions  $B_i^k$

- ① For  $t < t_i$  or  $t > t_{i+k+1}$ ,  $B_i^k(t) = 0$
- ② For  $t_i < t < t_{i+k+1}$ ,  $B_i^k(t) > 0$
- ③ For all  $t$ ,  $\sum_{i=-\infty}^{\infty} B_i^k(t) = 1$
- ④ For  $k \geq 1$ ,  $B_i^k$  has  $k - 1$  continuous derivatives
- ⑤ Set of functions  $\{B_{1-k}^k, \dots, B_{n-1}^k\}$  is linearly independent on interval  $[t_1, t_n]$  and spans space of all splines of degree  $k$  having knots  $t_i$

# B-splines, continued

- Properties 1 and 2 together say that B-spline functions have local support
- Property 3 gives normalization
- Property 4 says that they are indeed splines
- Property 5 says that for given  $k$ , these functions form basis for set of all splines of degree  $k$

## B-splines, continued

- If we use B-spline basis, linear system to be solved for spline coefficients will be nonsingular and banded
- Use of B-spline basis yields efficient and stable methods for determining and evaluating spline interpolants, and many library routines for spline interpolation are based on this approach
- B-splines are also useful in many other contexts, such as numerical solution of differential equations, as we will see later