Outline

Chapter 7: Interpolation

- Topics:
  - Examples
  - Polynomial Interpolation – bases, error, Chebyshev, piecewise
  - Orthogonal Polynomials
  - Splines – error, end conditions
  - Parametric interpolation
  - Multivariate interpolation: f(x,y)

Interpolation

- Basic interpolation problem: for given data

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

determine function $f\colon \mathbb{R} \to \mathbb{R}$ 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

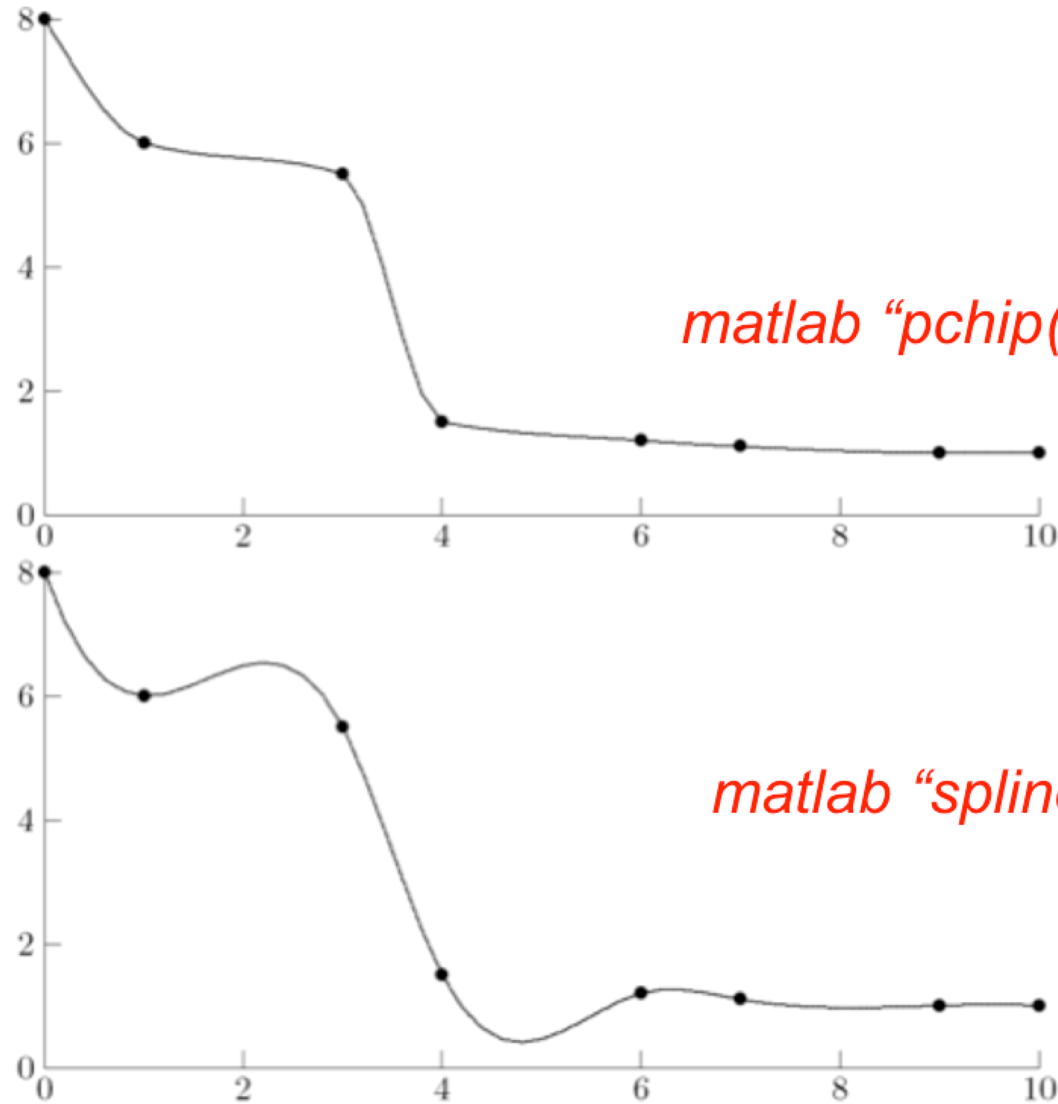
- What form should interpolating function have?
- How should interpolant behave between data points?
- Should interpolant inherit properties of data, such as monotonicity, convexity, or periodicity?
- Are parameters that define interpolating function meaningful? ***For example, function values, slopes, etc. ?***
- 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.)

Example



p(t)

matlab "pchip()" function

p(t)

matlab "spline()" function

t

A Classic Interpolation Problem

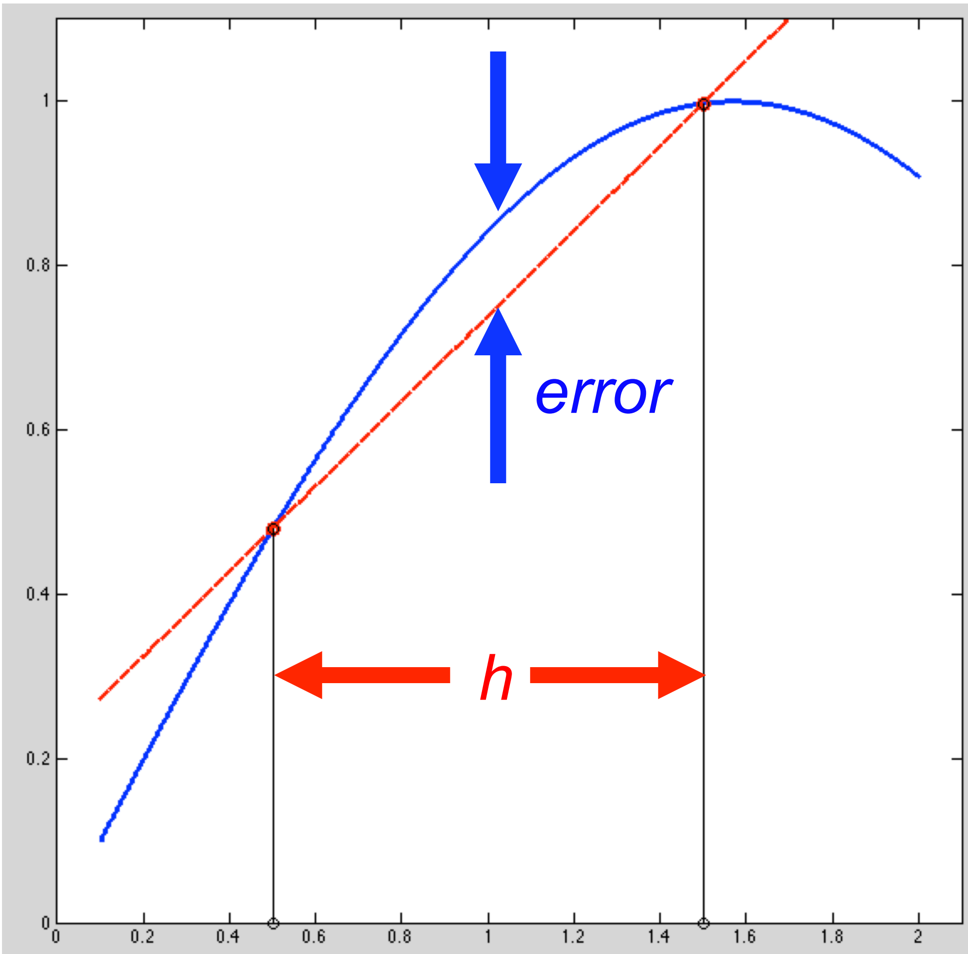
- Suppose you're asked to tabulate data such that linear interpolation between tabulated values is correct to 4 digits.

- How many entries are required on, say, [0,1]?

- How many digits should you have in the tabulated data?

x	$Si(x) = \int_0^x \frac{\sin t}{t}\,dt$
41.00	1.59494 33514
.01	9490 34645
.02	9486 11840
.03	9481 65154
.04	9476 94642
41.05	1.59472 00364
.06	9466 82381
.07	9461 40756
.08	9455 75554
.09	9449 86844
41.10	1.59443 74695
.11	9437 39181
.12	9430 80377
.13	9423 98359
.14	9416 93207
41.15	1.59409 65002
.16	9402 13830
.17	9394 39775
.18	9386 42927
.19	9378 23376

Error in Linear Interpolation



$$|f(x) - p(x)| \;\leq\; \max_{[x_1:x_2]} \frac{|f''|}{2}\frac{h^2}{4} = \max_{[x_1:x_2]} \frac{h^2|f''|}{8}$$

Polynomial Interpolation Example

- Given the table below,

x_j	f_j
0.6	1.2
0.8	2.0
1.0	2.4

estimate f(x=0.75).

Polynomial Interpolation Example

- Given the table below,

x_j	f_j
0.6	1.2
0.8	2.0
1.0	2.4

estimate f(x=0.75).

- A: 1.8 --- You've just done (piecewise) linear interpolation.

- Moreover, you know the error is $\leq$ $(0.2)^2$ f'' / 8.
  - Estimate the error…

A Classic Polynomial Interpolation Problem

Example: $f(x) = \cos(x)$

We know that $|f''| \leq 1$ and thus, for linear interpolation

$$|f(x) - p(x)| \leq \frac{h^2}{8}.$$

If we want 4 decimal places of accuracy, accounting for rounding, we need

$$\begin{aligned}|f(x) - p(x)| &\leq \frac{h^2}{8} \leq \frac{1}{2} \times 10^{-4} \\ h^2 &\leq 4 \times 10^{-4} \\ h &\leq 0.02\end{aligned}$$

x	cos x
0.00	1.00000
0.02	0.99980
0.04	0.99920
0.06	0.99820
0.08	0.99680

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 $\boldsymbol{Ax} = \boldsymbol{y}$ for n-vector $\boldsymbol{x}$ of parameters x_j, where entries of $m \times n$ matrix $\boldsymbol{A}$ 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 ***(linear least squares)***
- If $m < n$, interpolant is not unique
- If $m = n$, then basis matrix $\boldsymbol{A}$ is nonsingular provided data points t_i are distinct, so data can be fit exactly
- Sensitivity of parameters $\boldsymbol{x}$ to perturbations in data depends on $\text{cond}(\boldsymbol{A})$, 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), $i = 1, \ldots, n$, where t_i are distinct
- There are many ways to represent or compute interpolating polynomial, but in theory all must give same result

Monomial Basis

- *Monomial basis functions*

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

give interpolating polynomial of form

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

with coefficients $\boldsymbol{x}$ given by $n \times n$ linear system

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

- 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

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

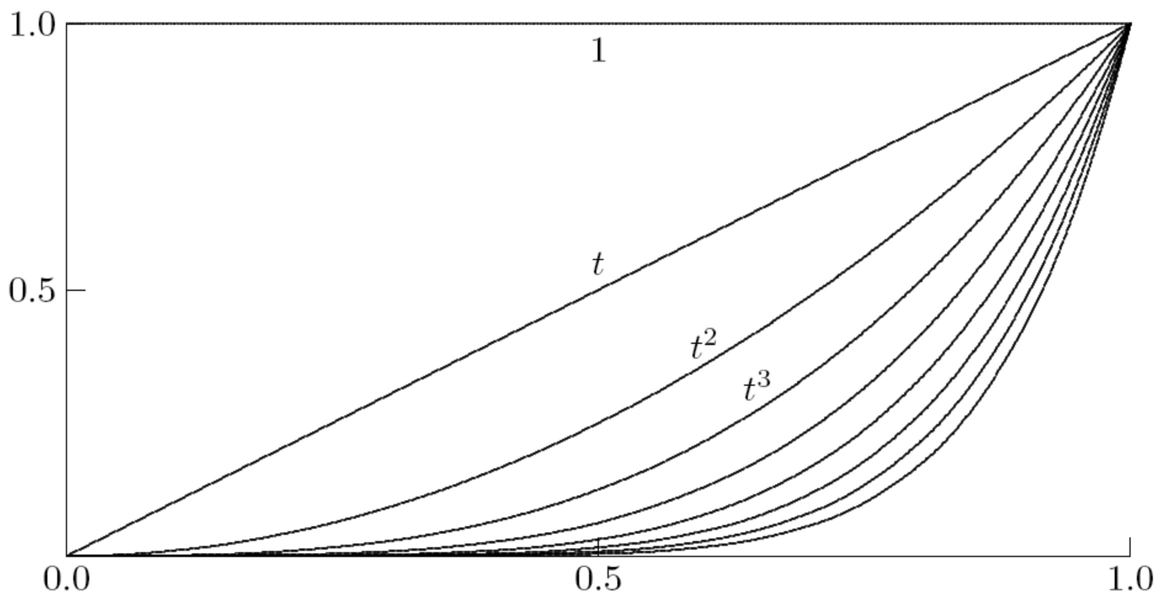
- For these particular data, system is

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

whose solution is $x = \begin{bmatrix} -1 & 5 & -4 \end{bmatrix}^T$, so interpolating polynomial is

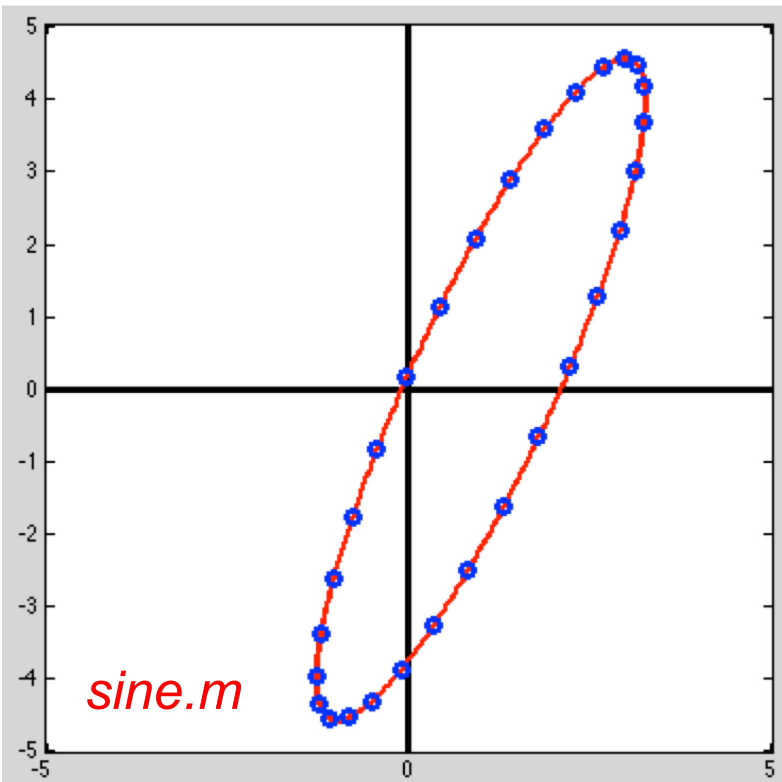
$$p_2(t) = -1 + 5t - 4t^2$$

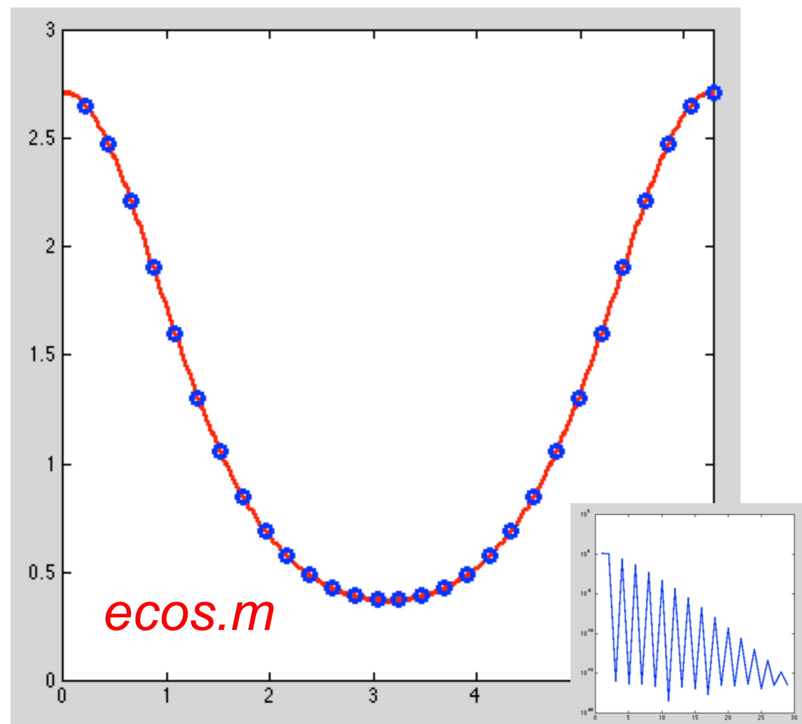
Monomial Basis, continued



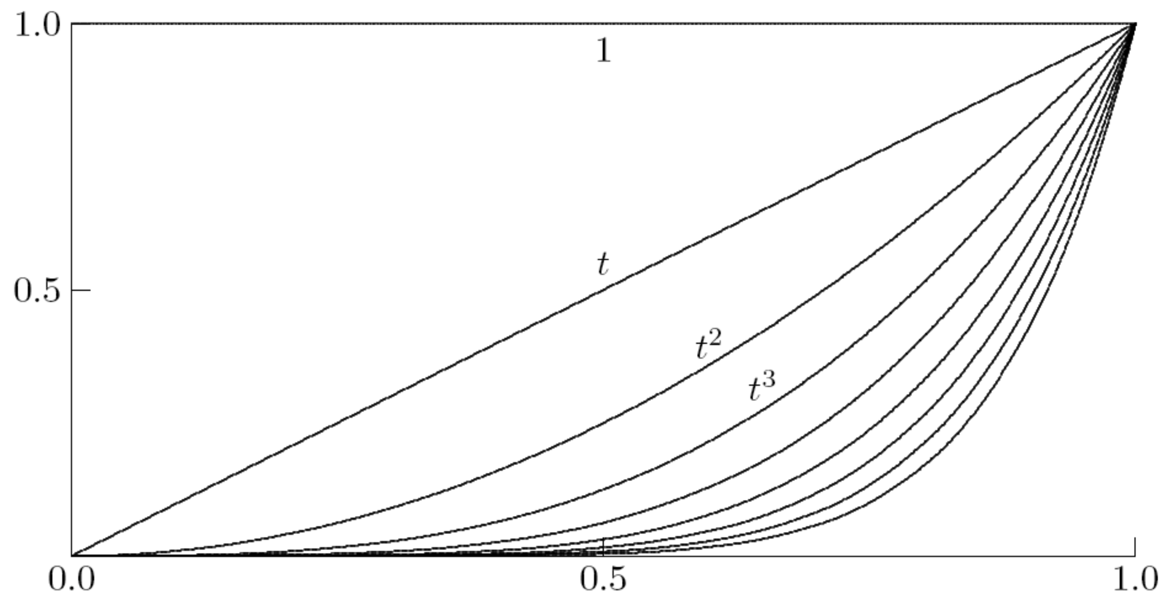
- Solving system $\boldsymbol{Ax} = \boldsymbol{y}$ using standard linear equation solver to determine coefficients $\boldsymbol{x}$ of interpolating polynomial requires $\mathcal{O}(n^3)$ work

Example: Sinusoidal Bases for Periodic Functions





Monomial Basis, continued



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

Monomial Basis, continued

- For monomial basis, matrix $\boldsymbol{A}$ 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

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

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

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), $i = 1, \ldots, n$, *Lagrange basis functions* are defined by

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

- For Lagrange basis,

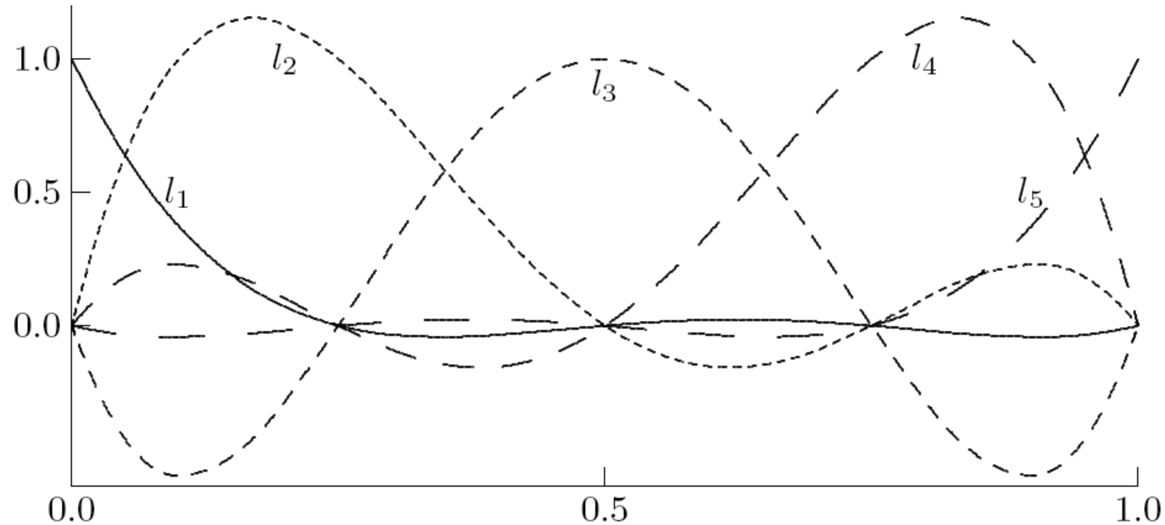
$$\ell_j(t_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \quad i, j = 1, \ldots, n$$

so matrix of linear system $\boldsymbol{Ax} = \boldsymbol{y}$ 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



- 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

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

General Polynomial Interpolation

- Whether interpolating on segments or globally, error formula applies over the interval.
  If $p(t) \in \mathbb{P}_{n-1}$ and $p(t_j) = f(t_j)$, $j = 1, \ldots, n$, then there exists a $\theta \in [t_1, t_2, \ldots, t_n, t]$ such that

$$\begin{aligned} f(t) - p(t) &= \frac{f^n(\theta)}{n!}(t - t_1)(t - t_2)\cdots(t - t_n) \\ &= \frac{f^n(\theta)}{n!} q_n(t), \qquad q_n(t) \in \mathbb{P}_n. \end{aligned}$$

- We generally have no control over $f^n(\theta)$, so instead seek to optimize choice of the t_j in order to minimize

$$\max_{t \in [t_1, t_n]} |q_n(t)| \, .$$

- Such a problem is called a *minimax* problem and the solution is given by the t_js being the roots of a Chebyshev polynomial, as we will discuss shortly.

- First, however, we turn to the problem of constructing $p(t) \in \mathbb{P}_{n-1}(t)$.

Constructing High-Order Polynomial Interpolants

- *Lagrange Polynomials*

$$p(t) = \sum_{j=1}^{n} f_j \, l_j(t)$$

$$\left.\begin{aligned} l_j(t) &= 1 \qquad t = t_j \\ l_j(t_i) &= 0 \qquad t = t_i, \; i \neq j \end{aligned}\right\} \; l_j(t_i) = \delta_{ij}$$

$$l_j(t) \in \mathbb{P}_{n-1}(t)$$

The $l_j(t)$ polynomials are chosen so that $p(t_j) = f(t_j) := f_j$

The $l_j(t)$s are sometimes called the Lagrange cardinal functions.

Constructing High-Order Polynomial Interpolants

- *Lagrange Polynomials*

$$
\begin{aligned}
p(t) &= \sum_{j=1}^{n} f_j \, l_j(t) \\
l_j(t) &= 1 \qquad t = t_j \\
l_j(t_i) &= 0 \qquad t = t_i, \; i \neq j \\
l_j(t) &\in \mathbb{P}_{n-1}(t)
\end{aligned}
$$

$$
l_j(t) = \frac{1}{C}(t - t_1)(t - t_2) \cdots (t - t_{j-1})(t - t_{j+1}) \cdots (t - t_n)
$$

- $l_j(t)$ is a polynomial of degree $n-1$
- It is zero at $t = t_i$, $i \neq j$.
- Choose C so that it is 1 at $t = t_j$.

Constructing High-Order Polynomial Interpolants

- $l_j(t)$ is a polynomial of degree $n-1$
- It is zero at $t = t_i$, $i \neq j$.
- Choose C so that it is 1 at $t = t_j$.

$$
\begin{aligned}
l_j(t) &= \frac{1}{C}(t-t_1)(t-t_2)\cdots(t-t_{j-1})(t-t_{j+1})\cdots(t-t_n) \\
C &= (t_j-t_1)(t_j-t_2)\cdots(t_j-t_{j-1})(t_j-t_{j+1})\cdots(t_j-t_n)
\end{aligned}
$$

Constructing High-Order Polynomial Interpolants

- $l_j(t)$ is a polynomial of degree $n-1$
- It is zero at $t = t_i$, $i \neq j$.
- Choose C so that it is 1 at $t = t_j$.

$$\begin{aligned} l_j(t) &= \frac{1}{C}(t-t_1)(t-t_2)\cdots(t-t_{j-1})(t-t_{j+1})\cdots(t-t_n) \\ C &= (t_j-t_1)(t_j-t_2)\cdots(t_j-t_{j-1})(t_j-t_{j+1})\cdots(t_j-t_n) \end{aligned}$$

$$l_j(t) = \left(\frac{t-t_1}{t_j-t_1}\right)\left(\frac{t-t_2}{t_j-t_2}\right)\cdots\left(\frac{t-t_{j-1}}{t_j-t_{j-1}}\right)\left(\frac{t-t_{j+1}}{t_j-t_{j+1}}\right)\cdots\left(\frac{t-t_n}{t_j-t_n}\right).$$

Constructing High-Order Polynomial Interpolants

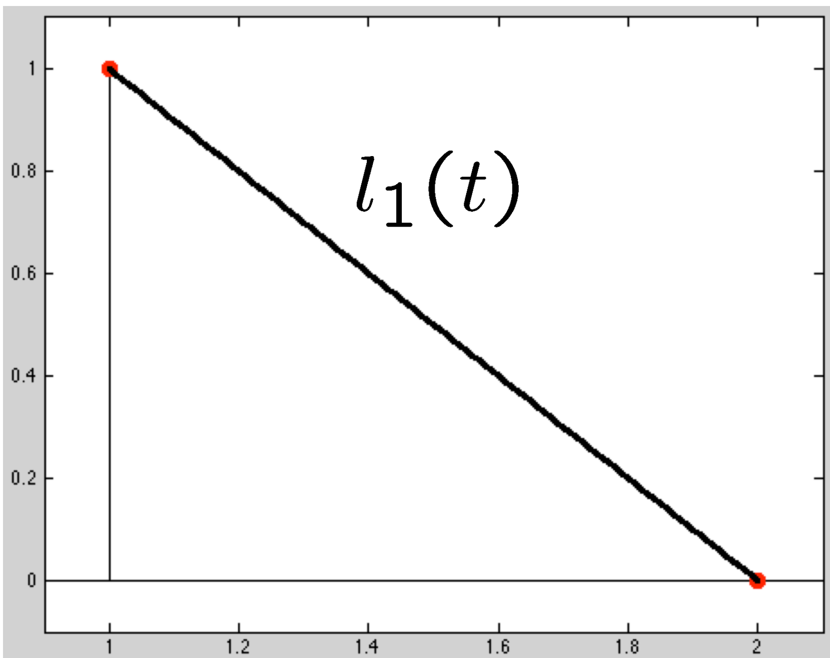
$$l_j(t) \;=\; \left(\frac{t-t_1}{t_j-t_1}\right)\left(\frac{t-t_2}{t_j-t_2}\right)\cdots\left(\frac{t-t_{j-1}}{t_j-t_{j-1}}\right)\left(\frac{t-t_{j+1}}{t_j-t_{j+1}}\right)\cdots\left(\frac{t-t_n}{t_j-t_n}\right).$$

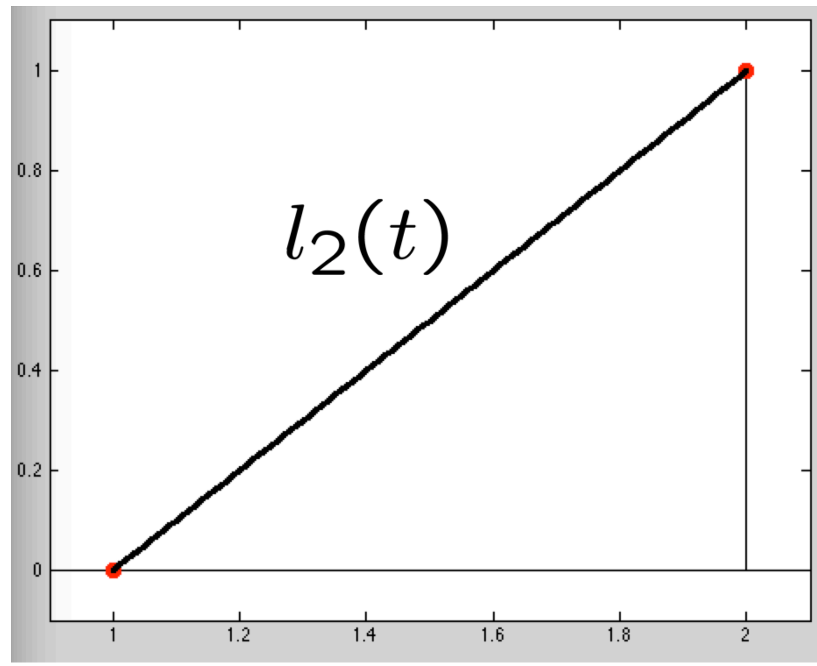
- Although a bit tedious to do by hand, these formulas are relatively easy to evaluate with a computer.

- So, to recap – Lagrange polynomial interpolation:

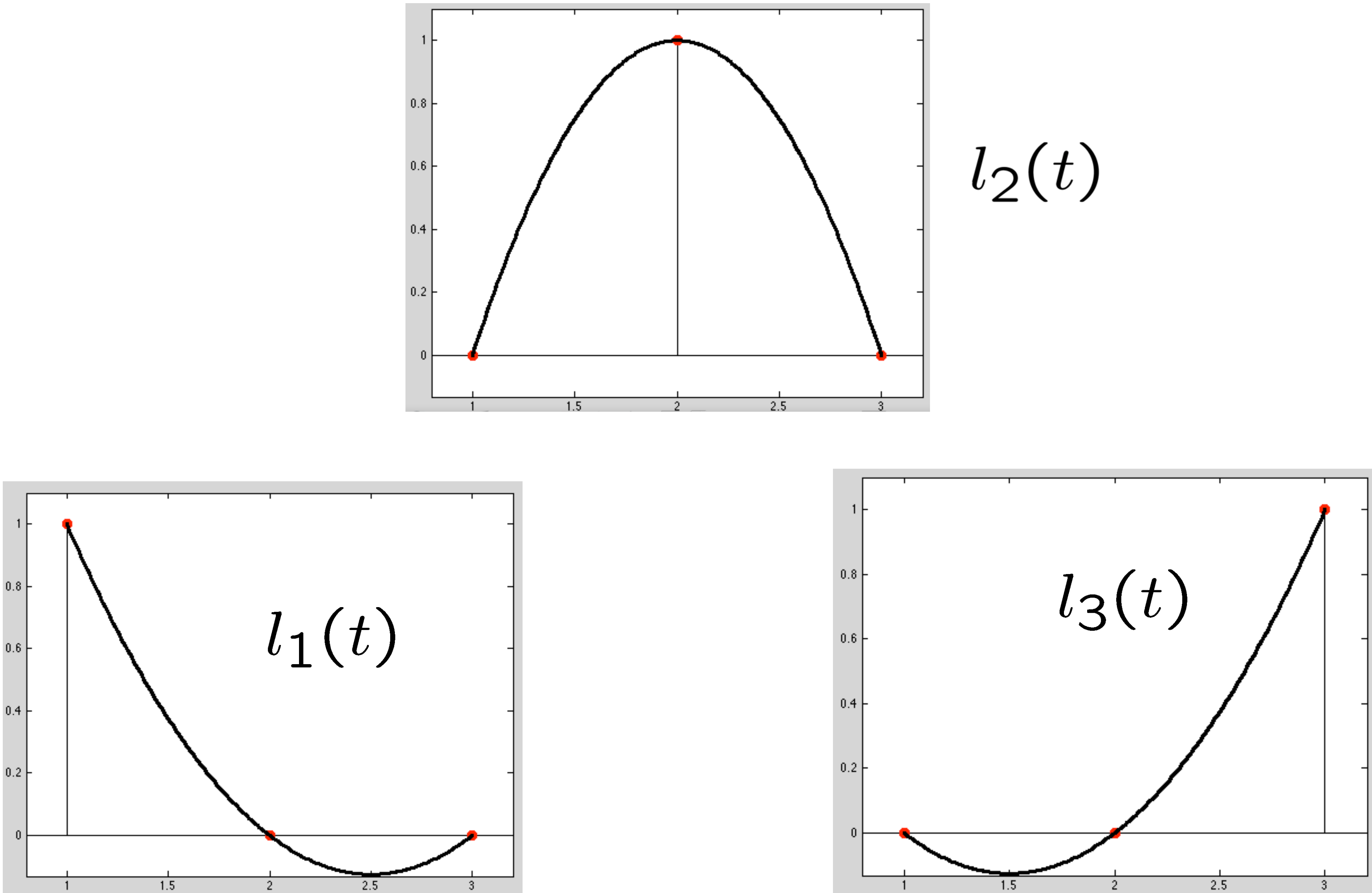
  - Construct $p(t) = \sum_j f_j l_j(t)$.
  - $l_j(t)$ given by above.
  - Error formula $f(t) - p(t)$ given as before.
  - Can choose t_j's to minimize error polynomial $q_n(t)$.

Lagrange Basis Functions, n=2 (linear)





Lagrange Basis Functions, n=3 (quadratic)



Comment on Costs

- Two parts:
  - A. Finding coefficients
  - B. Evaluating interpolant.

Costs for Lagrange Interpolation

Fast evaluation...

- Consider the following scheme, which is $O(n)$ per evaluation:

$$
\begin{aligned}
p(t) &= \sum_{j=1}^{n} l_j(t) f_j && (1)\\
l_j(t) &= c_j\, q_j(t)\, r_j(t) && (2)\\
c_j &:= \left[\prod_{i \neq j} (x_j - x_i)\right]^{-1} && (3)\\
q_j(t) &:= (t-t_1)(t-t_2)\cdots(t-t_{j-1}) \;=\; q_{j-1}(t)\cdot(t-t_{j-1}) && (4)\\
r_j(t) &:= (t-t_{j+1})(t-t_{j+2})\cdots(t-t_n) \;=\; (t-t_{j+1})\cdot r_{j+1}. && (5)
\end{aligned}
$$

  - The cost of (1) is $2n$.
  - The cost of (2) is 2, for j=1...n.
  - The cost of (3) is $O(n^2)$, but *one-time only*.
  - The cost of (4) is 2, for j=1...n.
  - The cost of (5) is 2, for j=1...n.

- The total evaluation cost for $m \gg n$ evaluations is $O(nm)$.
- There are no $O(n^3)$ costs.

Newton Interpolation

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

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

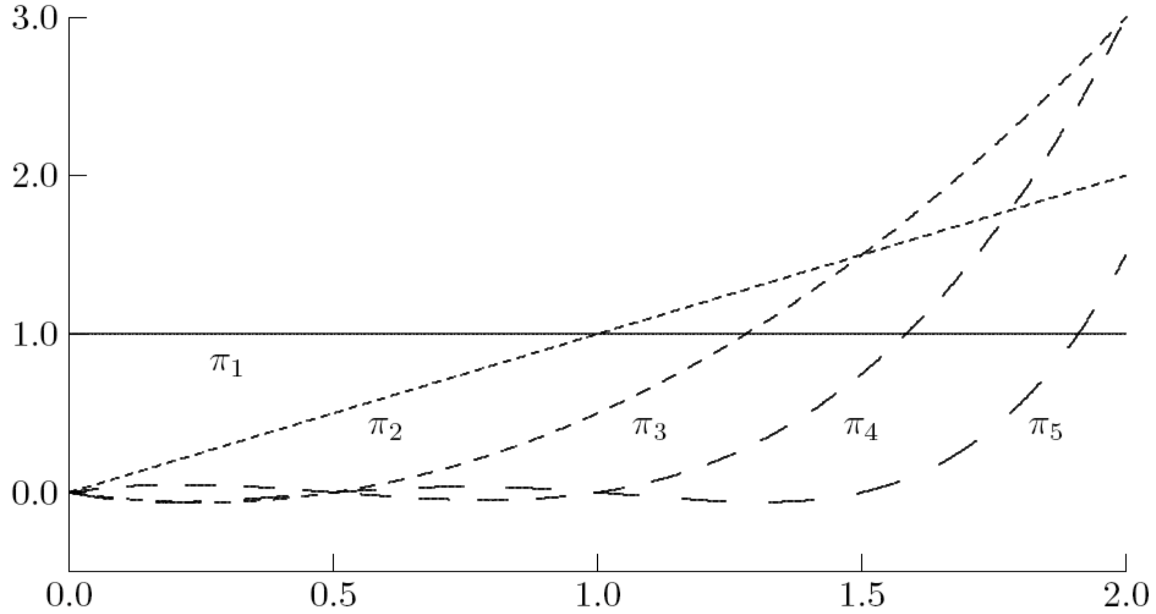
  where value of product is taken to be 1 when limits make it vacuous

- 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 $\boldsymbol{A}$ is lower triangular, where $a_{ij} = \pi_j(t_i)$

Newton Basis Functions



Newton Interpolation, continued

- Solution $\boldsymbol{x}$ to system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ 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 $x = \begin{bmatrix} -27 & 13 & -4 \end{bmatrix}^T$, so interpolating polynomial is

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

Newton Interpolation, continued

- Solution $\boldsymbol{x}$ to system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ 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

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

Divided Differences

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

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

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

- Coefficient of jth basis function in Newton interpolant is given by

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

- 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 $1, t, t^2, t^3, \ldots$ yields *Legendre polynomials*

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

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

  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

- kth *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

$$1, \;\; t, \;\; 2t^2 - 1, \;\; 4t^3 - 3t, \;\; 8t^4 - 8t^2 + 1, \;\; 16t^5 - 20t^3 + 5t, \;\; \ldots$$

- *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 Polynomials

$$
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1
\end{aligned}
$$

Legendre Polynomials

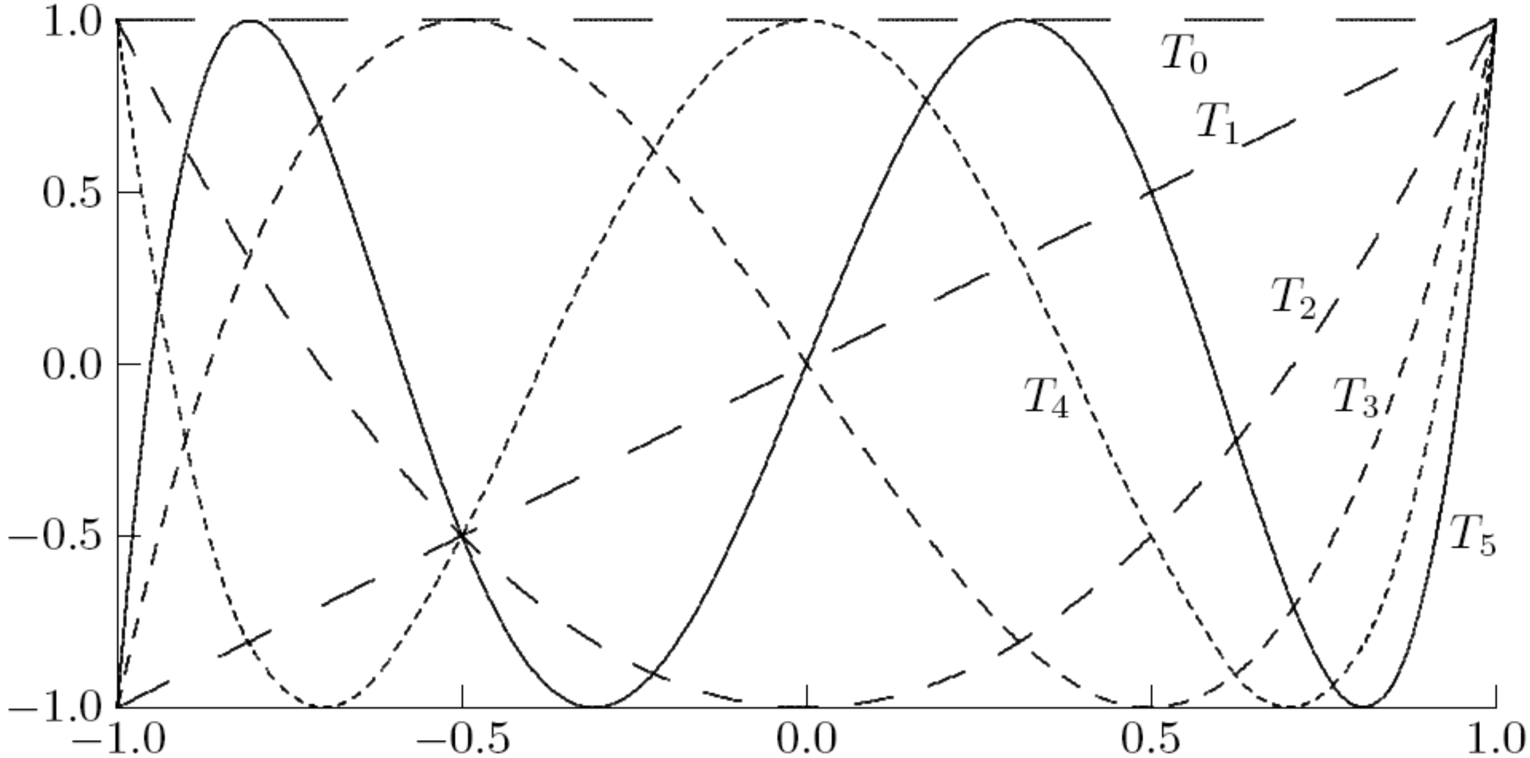
$$
\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= \tfrac{1}{2}\left(3x^2 - 1\right) \\
P_3(x) &= \tfrac{1}{2}\left(5x^3 - 3x\right) \\
P_4(x) &= \tfrac{1}{8}\left(35x^4 - 30x^3 + 3\right) \\
P_5(x) &= \tfrac{1}{8}\left(63x^5 - 70x^3 + 15x\right) \\
P_6(x) &= \tfrac{1}{16}\left(231x^6 - 315x^4 + 105x^2 - 5\right).
\end{aligned}
$$

- Recursion relationships:

$$
\begin{aligned}
\textit{Chebyshev:} \quad T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \\
\textit{Legendre:} \quad (n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x).
\end{aligned}
$$

- Chebyshev polynomials orthogonal with respect to $w(x) = (1 - x^2)^{-\frac{1}{2}}$.
- Legendre polynomials orthogonal with respect to $w(x) = 1$.
- Chebyshev polynomials important for minimax problems (e.g., minimize maximum of $p(t) - f(t)$).
- Legendre polynomials important for Gauss quadrature rules.
- These and other orthogonal polynomials have other important uses.

Chebyshev Basis Functions

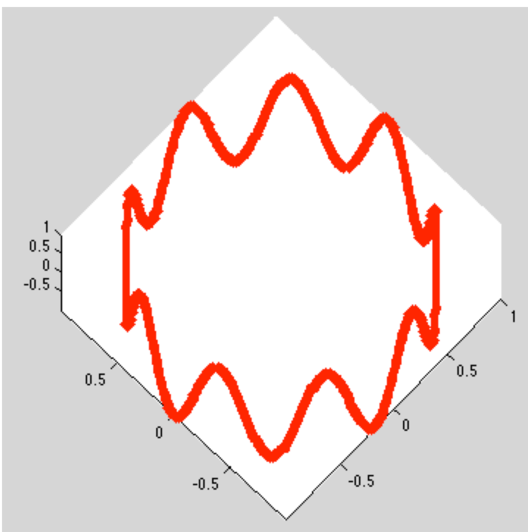


< interactive example >

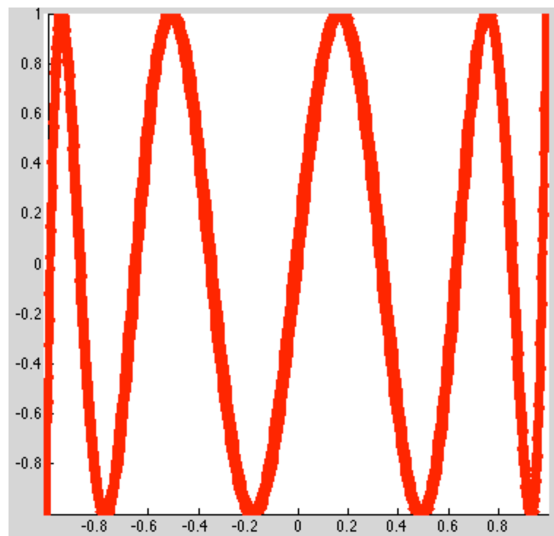
Nth-Order Gauss Chebyshev Points

Matlab Demo: cheb_fun_demo.m

```
t=0:.01:(2*pi); t=t'; x=cos(t); y=sin(t);

n=9; z=cos(n*t);

plot3(x,y,z,'r','LineWidth',5); axis equal
```



$\cos(N\theta)$



$T_N(x)$

$$T_N(x) = \cos(N\theta)$$

$$x = \cos(\theta)$$

Chebyshev Points

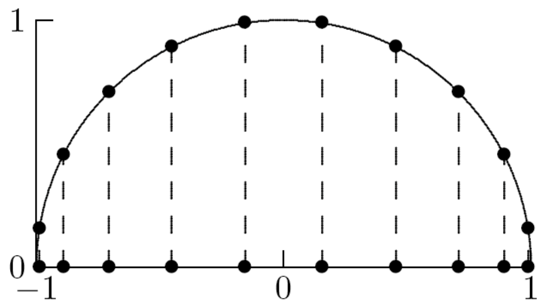
- *Chebyshev points* are zeros of T_k, given by

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

or extrema of T_k, given by

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

- 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 $t_1, \ldots, t_n$, then

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

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

- Since point θ 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 $h = \max\{t_{i+1} - t_i : \; i = 1, \ldots, n-1\}$, 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

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

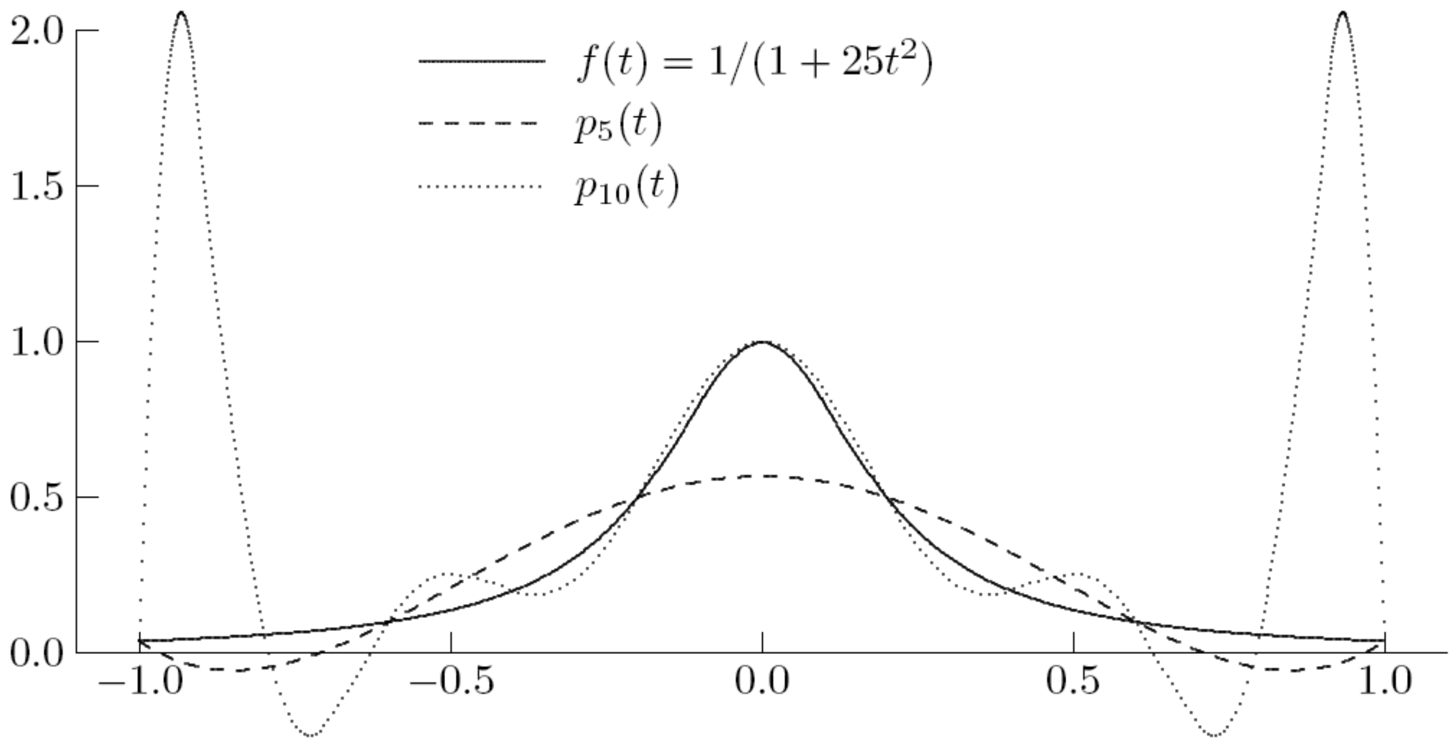
Not Always True!

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

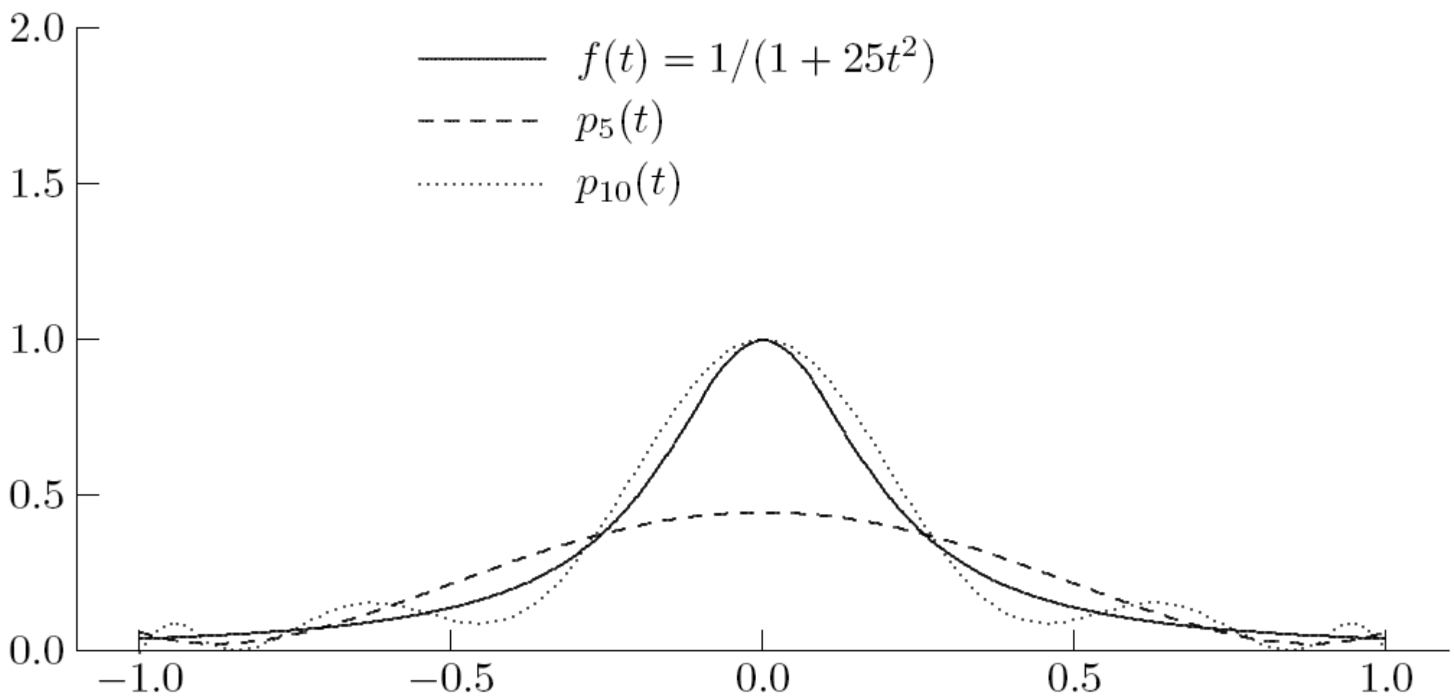
Example: Runge's Function

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



Example: Runge's Function

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



Chebyshev Convergence is exponential for smooth f(t).

Important Polynomial Interpolation Result

If $p(x) \in \mathbb{P}_{n-1}$ and $p(x_j) = f(x_j)$, $j = 1, \ldots, n$, then there exists a $\theta \in [x_1, x_2, \ldots, x_n, x]$ such that

$$\begin{aligned} f(x) - p(x) &= \frac{f^n(\theta)}{n!}(x - x_1)(x - x_2) \cdots (x - x_n) \\ |f(x) - p(x)| &\leq \frac{M}{n!} \left|(x - x_1)(x - x_2) \cdots (x - x_n)\right|, \end{aligned}$$

where $M = \max_{\theta} |f^n(\theta)|$.

- Note that $f(x_j) - p(x_j) = 0$, $j = 1, \ldots, n$, as should be the case.
- The formula applies to extrapolation ($x \notin [x_1, \ldots, x_n]$) as well as interpolation ($x \in [x_1, \ldots, x_n]$).
- The error is contolled by the maximum of $|f^n|$ on the interval of interest, which is the smallest interval containing the x_js and x.
- Notice that if $f \in \mathbb{P}_n$ then f^n is a constant.
  In particular, if $f(x) = x^n$, the error is simply
  $$f(x) - p(x) \;=\; (x - x_1)(x - x_2) \cdots (x - x_n).$$
- On [-1,1], the Chebyshev points minimize
  $$q(x) \;:=\; (x - x_1)(x - x_2) \cdots (x - x_n).$$

An important polynomial interpolation result for $f(x) \in C^n$:

If $p(x) \in \mathbb{P}_{n-1}$ and $p(x_j) = f(x_j)$, $j = 1, \ldots, n$, then there exists a $\theta \in [x_1, x_2, \ldots, x_n, x]$ such that

$$f(x) - p(x) \;=\; \frac{f^n(\theta)}{n!}(x - x_1)(x - x_2)\cdots(x - x_n).$$

In particular, for *linear interpolation*, we have

$$f(x) - p(x) \;=\; \frac{f''(\theta)}{2}(x - x_1)(x - x_2)$$

$$|f(x) - p(x)| \;\leq\; \max_{[x_1:x_2]} \frac{|f''|}{2}\frac{h^2}{4} \;=\; \max_{[x_1:x_2]} \frac{h^2|f''|}{8}$$

where the latter result pertains to $x \in [x_1, x_2]$.

Examples: Application of Error Formula, etc.

- What is the sum of the Lagrange cardinal functions at any given x?

- Assume that $0 < t_1 < t_2 < \ldots < t_n < 1$ and that polynomial interpolation is used to interpolate cos t on [0,1].

  Show that for any 5 points on [0,1]

$$| \cos(t) - p(t) | < .01$$

A Classic Interpolation Problem

- **Q:** What accuracy can we expect when interpolating from the attached table, using piecewise linear interpolation?
  - **A:** What do we need to estimate the error?
    - h
    - f''
    - Use finite difference to estimate f'' …

x	$Si(x) = \int_0^x \frac{\sin t}{t}\,dt$
41.00	1.59494 33514
.01	9490 34645
.02	9486 11840
.03	9481 65154
.04	9476 94642
41.05	1.59472 00364
.06	9466 82381
.07	9461 40756
.08	9455 75554
.09	9449 86844
41.10	1.59443 74695
.11	9437 39181
.12	9430 80377
.13	9423 98359
.14	9416 93207
41.15	1.59409 65002
.16	9402 13830
.17	9394 39775
.18	9386 42927
.19	9378 23376

Unstable and Stable Interpolating Basis Sets

- *Examples of unstable bases are:*
  - Monomials (modal): $\phi_i = x^i$
  - High-order Lagrange interpolants (nodal) on *uniformly-spaced* points.

- Examples of ***stable*** bases are:
  - Orthogonal polynomials (modal), e.g.,
    - Legendre polynomials: $L_k(x)$, or
    - bubble functions: $\phi_k(x) := L_{k+1}(x) – L_{k-1}(x)$.
  - Lagrange (nodal) polynomials based on Gauss quadrature points (e.g., Gauss-Legendre, Gauss-Chebyshev, Gauss-Lobatto-Legendre, etc.)

- Can map back and forth between stable nodal bases and Legendre or bubble function modal bases, *with minimal information loss*.

Unstable and Stable Interpolating Basis Sets

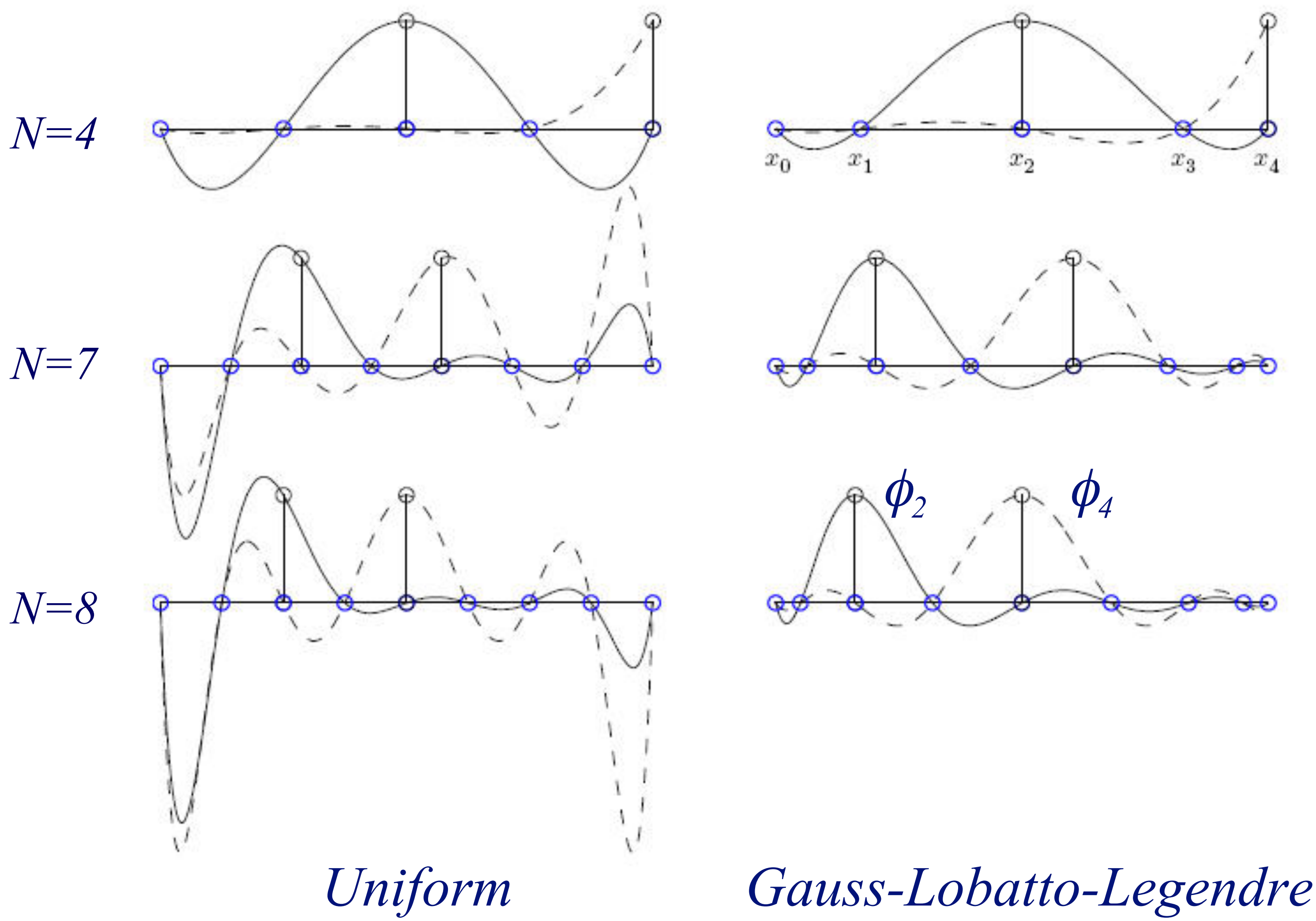
- Key idea for Chebyshev interpolation is to choose points that minimize $\max |q_{n+1}(x)|$ on interval $\mathcal{I} := [-1, 1]$.

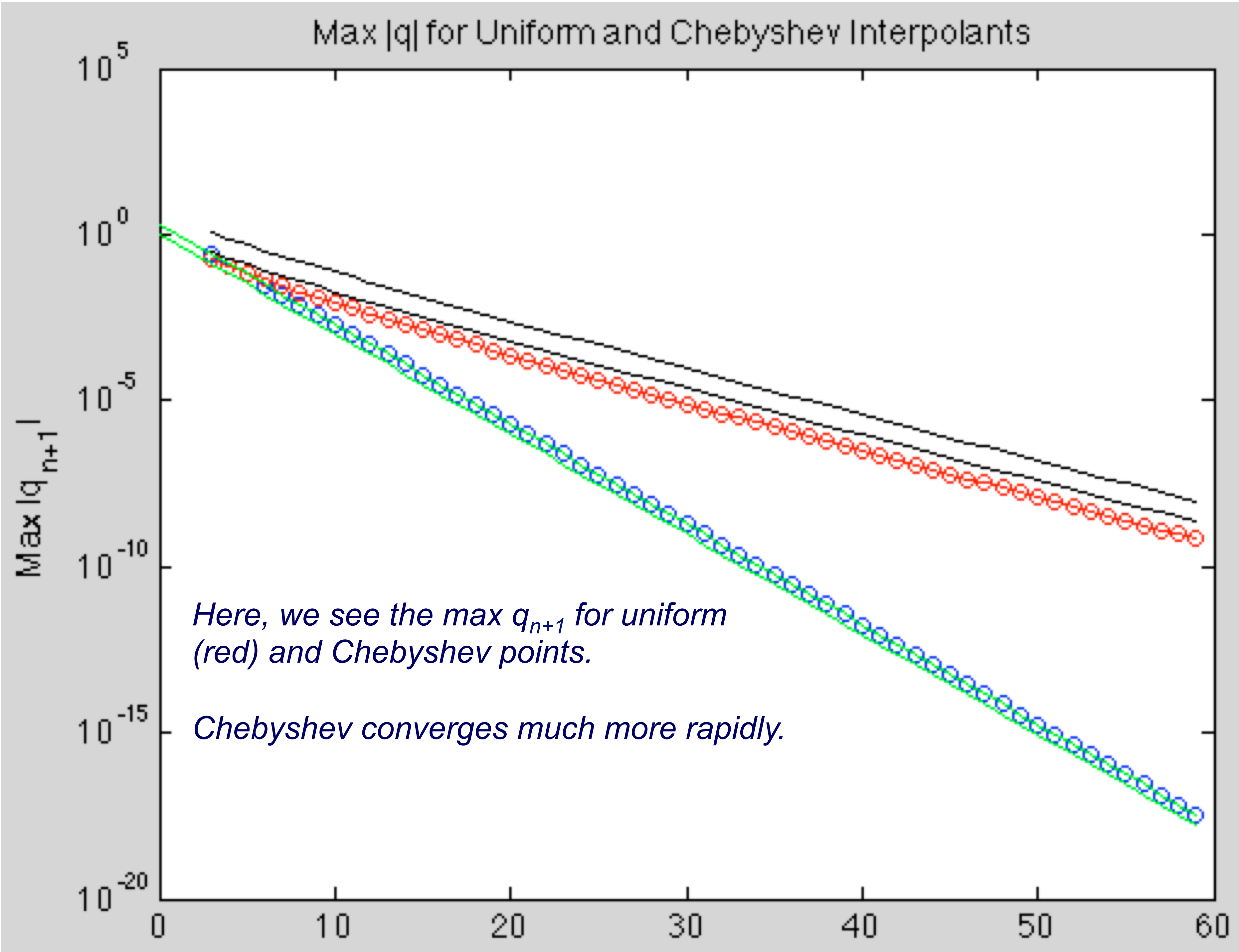
$$\begin{aligned} q_{n+1}(x) &:= (x - x_0)(x - x_1) \dots (x - x_n) \\ &:= x^n + c_{n-1}x^{n-1} + \dots + c_0 \end{aligned}$$

  which is a *monic polynomial* of degree $n + 1$.

- The roots of the Chebyshev polynomial $T_{n+1}(x)$ yield such a set of points by clustering near the endpoints.

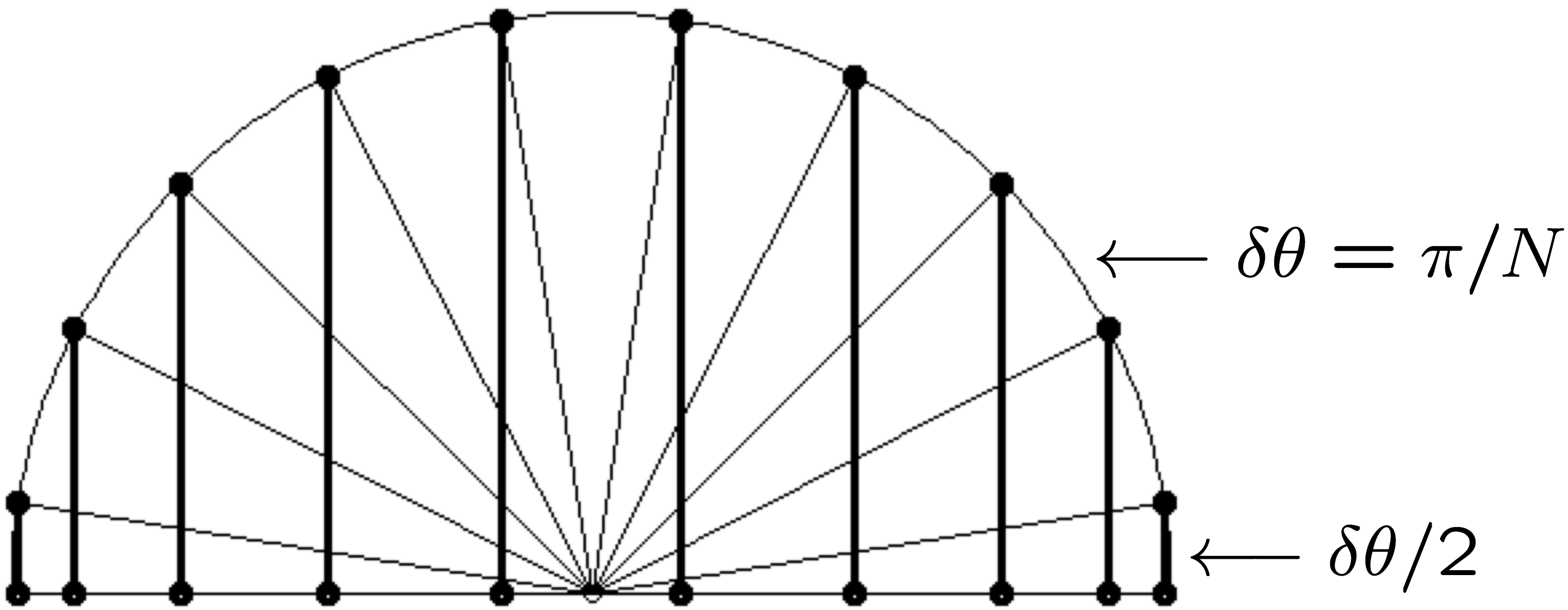
Lagrange Polynomials: Good and Bad Point Distributions




Max |q| for Uniform and Chebyshev Interpolants
Max $|q_{n+1}|$
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
0
10
20
30
40
50
60
Here, we see the max q_{n+1} for uniform (red) and Chebyshev points.
Chebyshev converges much more rapidly.

Nth-order Gauss-Chebyshev Points

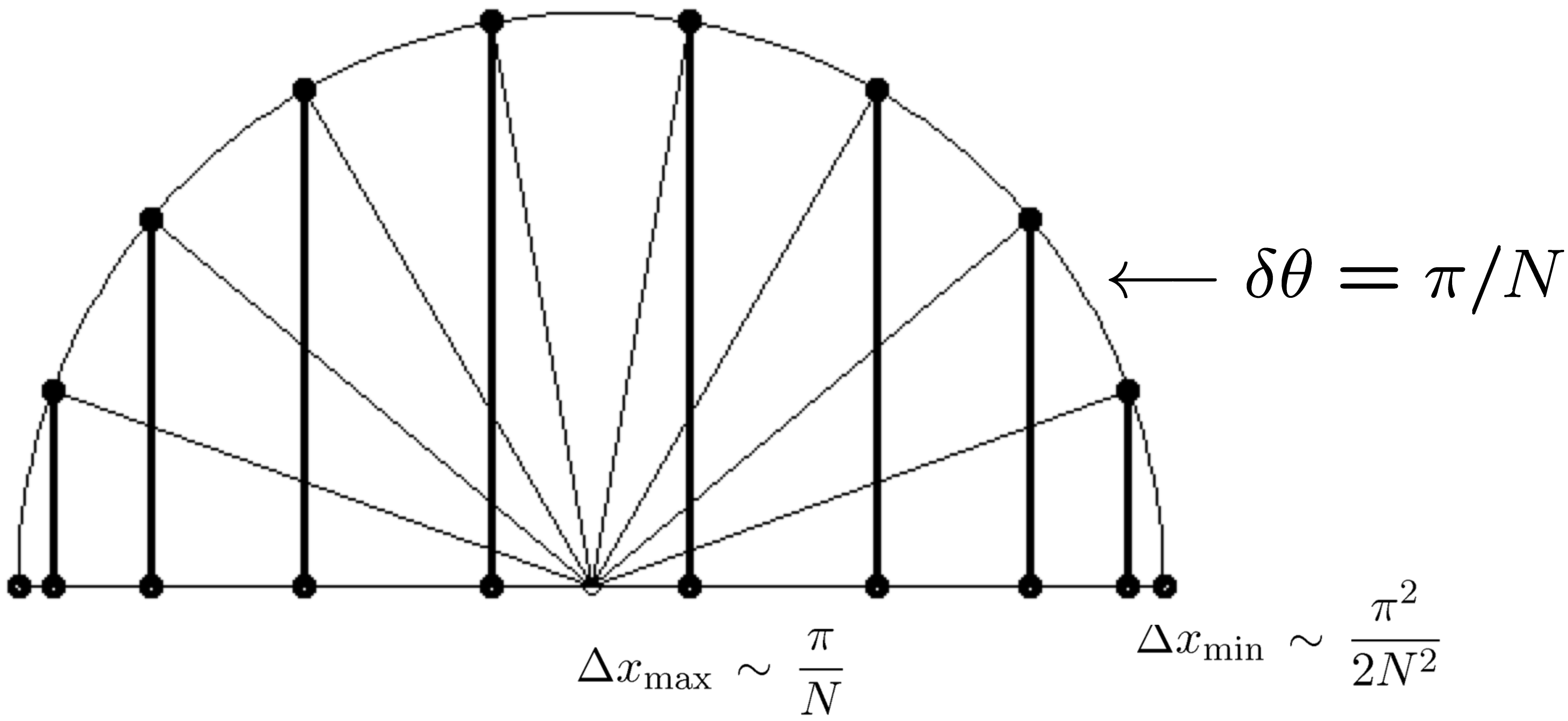
- Roots of Nth-order Chebyshev polynomial are projections of equispaced points on the circle, starting with $\theta = \delta\theta/2$, then $\theta = 3\delta\theta/2,\ldots,\pi-\delta\theta/2$.




```
ti=(0:N)/(N); ti=pi*ti'; %%% Gauss Lobatto Chebyshev Point generator
ti=(0:N); ti=(ti+.5)/(N+1); ti=pi*ti'; %%% Gauss Chebyshev Point generator
xi=cos(ti); yi=sin(ti);

close all; figure('Color',[1.0 1.0 1.0]);
for i=1:N+1;
    plot([xi(i) xi(i)],[0 yi(i)],'ko-','LineWidth',2); hold on;
    plot([0 xi(i)],[0 yi(i)],'ko-');
end;

N=100;  %% Draw Circle and x-axis:
ti=(0:N)/(N);  ti=pi*ti';  % theta in [0,pi]
xi=cos(ti); yi=sin(ti); plot(xi,0*xi,'k-',xi,yi,'k-');
```

N+1 Gauss-Lobatto Chebyshev Points

- N+1 GLC points are projections of equispaced points on the circle, starting with θ = 0, then θ = π/N, 2π/N, … , kπ/N, … , π.



Interpolation Testing

- Try a variety of *methods* for a variety of *functions.*
- Inspect by plotting the function and the interpolant.
- Compare with theoretical bounds. (Which *are* accurate!)

Typical Interpolation Experiment

- Given $f(t)$, evaluate $f_j := f(t_j)$, $j = 1, \ldots, n$.
- Construct interpolant:

$$p(t) = \sum_{j=1}^{n} \hat{p}_j \, \phi_j(t).$$

- Evaluate $p(t)$ at $\tilde{t}_i$, $i = 1, \ldots, m$, $m \gg n$. (Fine mesh, for plotting, say.)
- To check error, compare with original function on fine mesh, $\tilde{t}_i$.

$$\begin{aligned} e_i &:= p(\tilde{t}_i) - f(\tilde{t}_i) \\ e_{\max} &:= \frac{\max_i |e_i|}{\max_i |f_i|} \\ &\approx \frac{\max |p - f|}{\max |f|}. \end{aligned}$$

(Remember, it's an *experiment*.)

- Preceding description is for one *trial*.
- Repeat for increasing n and plot $e_{\max}(n)$ on a log-log or semilog plot.
- Compare with other methods *and* with theory:
  - **methods** – identify best method for given function / requirements
  - **theory** – verify that experiment is correctly implemented
- Repeat with a different function.

Summary of Key Theoretical Results

- **Piecewise linear interpolation:**

$$\max_{t\in[a,b]} |\, p - f\,| \;\leq\; \frac{h^2}{8} M, \qquad \begin{cases} M := \max_{\theta\in[a,b]} \; |f''(\theta)\,| \\ h := \max_{j\in[2,\ldots,n]}(t_j - t_{j-1}), \;\; t_{j-1} < t_j \end{cases}$$

- **Polynomial interpolation through n points:**

$$\begin{aligned} \max_{t\in[a,b]} |\, p - f\,| \quad &\leq \quad \frac{q_n(\theta)}{n!} M, \\ &\leq \quad \frac{h^n}{4n} M, \;\; (\text{for } t \in [a,b]), \\ \text{with} \quad & M := \max_{\theta\in[a,b,t]} |f^n(\theta)\,|. \end{aligned}$$

  – Here, $q_n(\theta) := (\theta - t_1)(\theta - t_2)\cdots(\theta - t_n)$.

  – The first result also holds true for *extrapolation*, i.e., $t \notin [a,b]$.

- **Natural cubic spline ($s''(a) = s''(b) = 0$):**

$$\max_{t\in[a,b]} |\,p - f\,| \;\le\; C\,h^2\,M, \quad M = \max_{\theta\in[a,b]} |f''(\theta)\,|,$$

unless $f''(a) = f''(b) = 0$, or other lucky circumstances.

- **Clamped cubic spline ($s'(a) = f'(a)$, $s'(b) = f'(b)$):**

$$\max_{t\in[a,b]} |\,p - f\,| \;\le\; C\,h^4\,M, \quad M = \max_{\theta\in[a,b]} |f^{iv}(\theta)\,|.$$

- **Nyquist sampling theorem:**

Roughly: *The maximum frequency that can be resolved with n points is $N = n/2$.*

There are other conditions, such as limits on the spacing of the sampling.

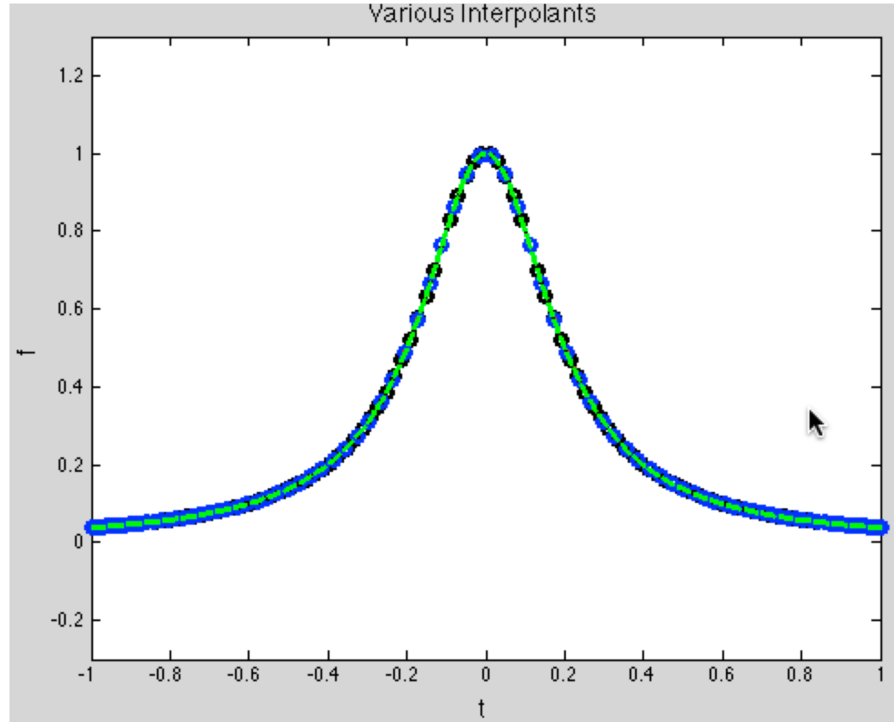
- **Methods:**
  - piecewise linear
  - polynomial on uniform points
  - polynomial on Chebyshev points
  - natural cubic spline

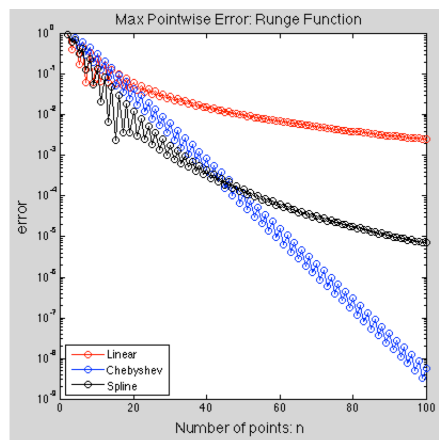
- **Tests:**
  - e^t
  - $e^{\cos t}$
  - $\sin t$ on $[0, \pi]$
  - $\sin t$ on $[0, \frac{\pi}{2}]$
  - $\sin 15t$ on $[0, 2\pi]$
  - $e^{\cos 11t}$on $[0, 2\pi]$
  - Runge function: $\frac{1}{1+25t^2}$ on $[0, 1]$
  - Runge function: $\frac{1}{1+25t^2}$ on $[-1, 1]$
  - Semi-circle: $\sqrt{1-t^2}$ on $[-1, 1]$
  - Polynomial: t^n
  - Extrapolation
  - Other

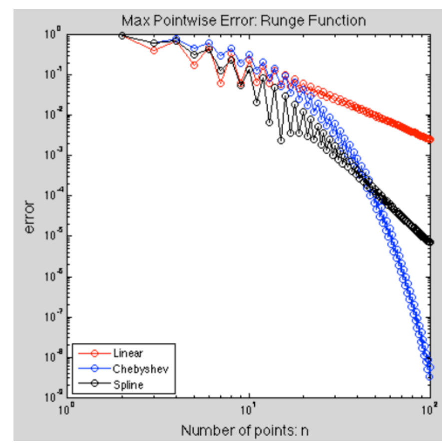
interp_test.m *interp_test_runge.m*

- **Methods:**
  - piecewise linear
  - polynomial on uniform points
  - polynomial on Chebyshev points
  - natural cubic spline

- **Tests:**
  - e^t
  - $e^{\cos t}$
  - $\sin t$ on $[0, \pi]$
  - $\sin t$ on $[0, \frac{\pi}{2}]$
  - $\sin 15t$ on $[0, 2\pi]$
  - $e^{\cos 11t}$on $[0, 2\pi]$
  - Runge function: $\frac{1}{1+25t^2}$ on $[0, 1]$
  - Runge function: $\frac{1}{1+25t^2}$ on $[-1, 1]$
  - Semi-circle: $\sqrt{1-t^2}$ on $[-1, 1]$
  - Polynomial: t^n
  - Extrapolation
  - Other







interp_test.m   *interp_test_runge.m*

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

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

Polynomial Interpolation

- Two types: *Global* *or Piecewise*
- Two scenarios:
  - A: points are given to you
  - B: you choose the points
- Case A: ***piecewise*** polynomials are most common – ***STABLE***.
  - Piecewise linear
  - Splines
  - Hermite (matlab "pchip" – piecewise cubic Hermite int. polynomial)
- Case B: high-order polynomials are OK if points chosen wisely
  - Roots of orthogonal polynomials
  - Convergence is exponential: err ~ $Ce^{-\sigma n}$, instead of algebraic: err~Cn^{-k}

Polynomial Interpolation Example

- Given the table below,

x_j	f_j
0.6	1.2
0.8	2.0
1.0	2.4

estimate f(x=0.75).

Polynomial Interpolation Example

- Given the table below,

x_j	f_j
0.6	1.2
0.8	2.0
1.0	2.4

estimate f(x=0.75).

- A: 1.8 --- You've just done (piecewise) linear interpolation.
- Moreover, you know the error is $\leq$ $(0.2)^2$ f'' / 8.
  - Estimate the error…

quick_spline.m

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

Cubic Spline Formulation – 2 Segments

8 Unknowns

$$\begin{aligned} p_1(t) &= \alpha_1 + \alpha_2 t + \alpha_3 t^2 + \alpha_4 t^3 \\ p_2(t) &= \beta_1 + \beta_2 t + \beta_3 t^2 + \beta_4 t^3 \end{aligned}$$

8 Equations

Interpolatory

$$\begin{aligned} p_1(t_1) &= y_1 \\ p_1(t_2) &= y_2 \\ p_2(t_2) &= y_2 \\ p_2(t_3) &= y_3 \end{aligned}$$

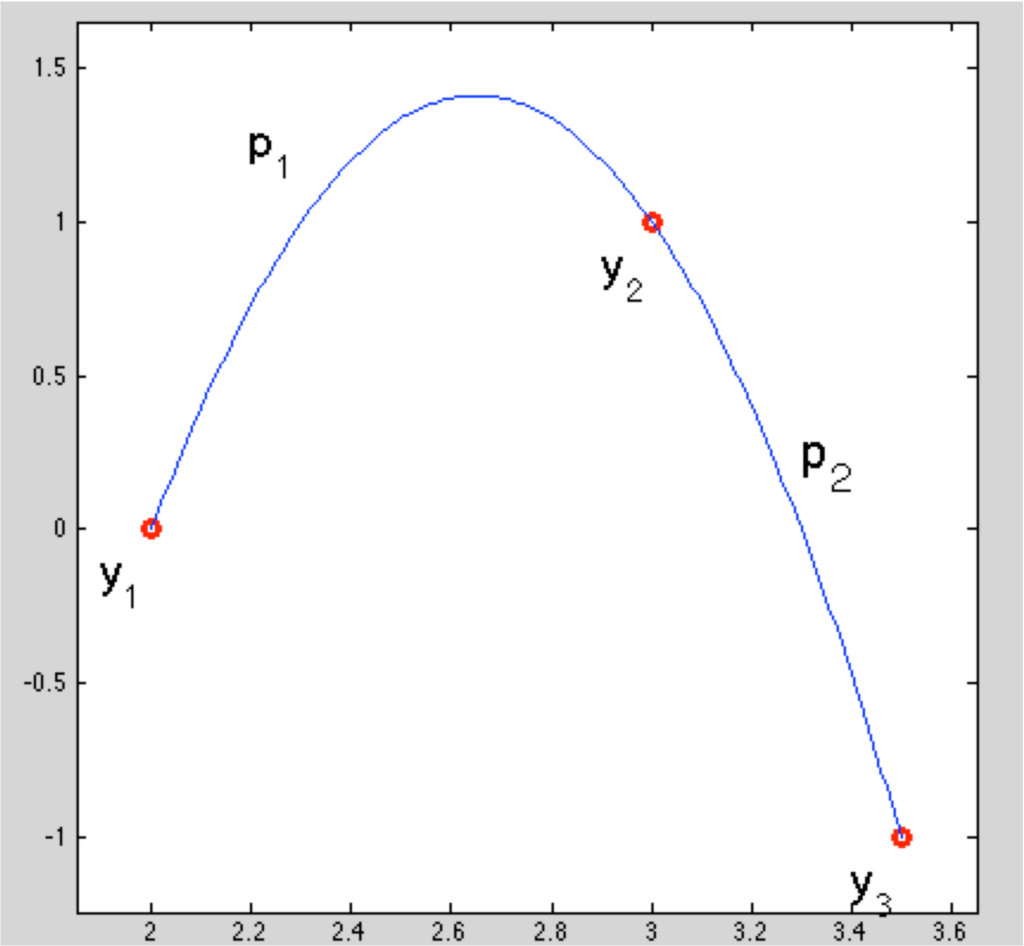
Continuity of Derivatives

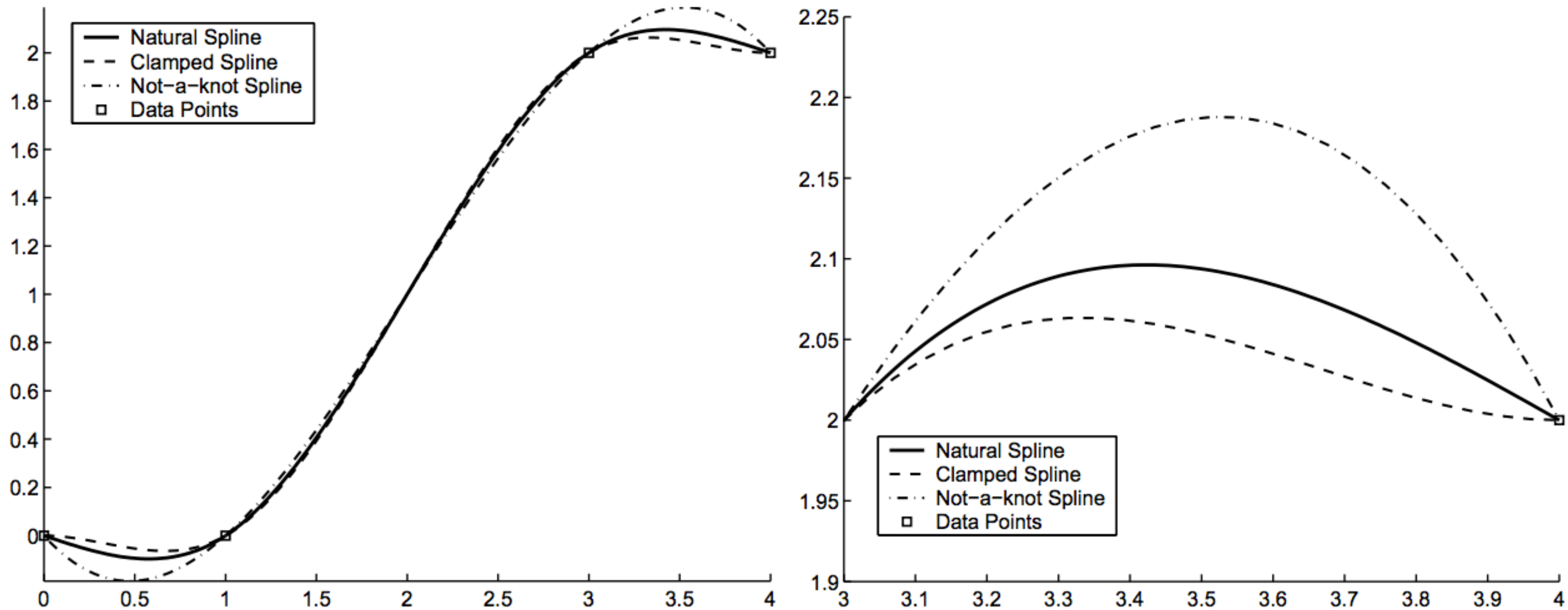
$$\begin{aligned} p_1'(t_2) &= p_2'(t_2) \\ p_1''(t_2) &= p_2''(t_2) \end{aligned}$$

End Conditions

$$\begin{aligned} p_1''(t_1) &= 0 \\ p_2''(t_3) &= 0 \end{aligned}$$

$(\mathit{Natural\ Spline})$





Note that the `spline` function in MATLAB computes a not-a-knot spline by default. If `X = [0 1 3 4]` and `Y = [0 0 2 2]`, the not-a-knot spline can be computed and plotted in MATLAB with `plot(x,ppval(spline(X,Y),x))`. Specifying additional data points at the beginning and end of the interval will give a clamped spline with those extra values as the slopes at the endpoints of the intervals. The command `plot(x,ppval(spline(X,[0 Y 0]),x))` would give the clamped spline plotted here with $f'(0) = f'(4) = 0$.

Some Cubic Spline Properties

- Continuity
  - 1^{st} derivative: continuous
  - 2^{nd} derivative: continuous

- "Natural Spline" minimizes integrated curvature:
  over all twice-differentiable f(x)
  passing through (x_j,f_j), j=1,…,n.

$$\int_{x_1}^{x_n} |S''(x)|^2\,dx \leq \int_{x_1}^{x_n} |f''(x)|^2\,dx$$

- Robust / Stable (unlike high-order polynomial interpolation)
- Commonly used in computer graphics, CAD software, etc.
- Usually used in parametric form (DEMO)
- There are other forms, e.g., tension-splines, that are also useful.
- For clamped boundary conditions, convergence is $O(h^4)$
- For small displacements, natural spline is like a ***physical spline.*** ***(DEMO)***

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

Piecewise Polynomial Bases: Linear and Quadratic

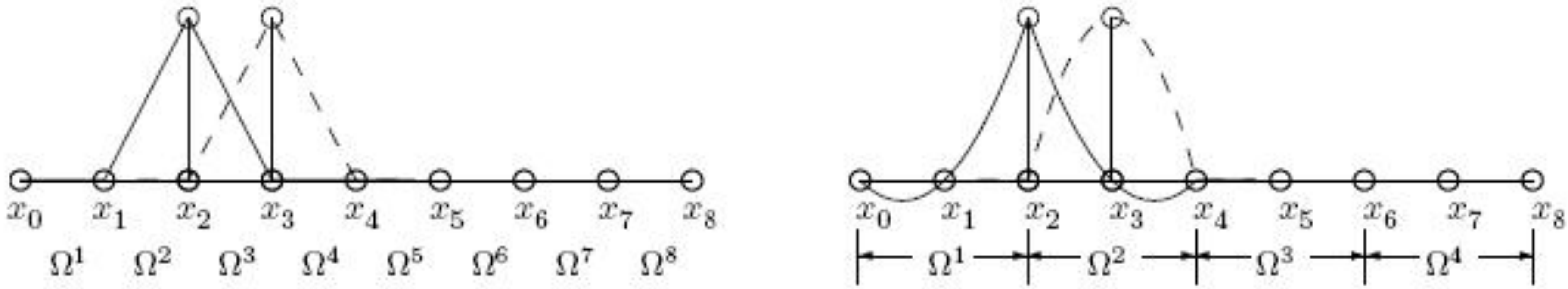


Figure 2: Examples of one-dimensional piecewise linear (left) and piecewise quadratic (right) Lagrangian basis functions, $\phi_2(x)$ and $\phi_3(x)$, with associated element support, Ω^e, $e = 1, \ldots, E$.

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

Piecewise cubics:

- Interval $\mathcal{I}_j \;=\; [x_{j-1}, x_j],\; j = 1, \ldots, n$

$$p_j(x) \in \mathbb{P}_3(x) \text{ on } \mathcal{I}_j$$

$$p_j(x) \;=\; a_j \;+\; b_j x \;+\; c_j x^2 \;+\; d_j x^3$$

- $4n$ unknowns

$$p_j(x_{j-1}) \;=\; f_{j-1},\; j = 1, \ldots, n$$

$$p_j(x_j) \;=\; f_j,\; j = 1, \ldots, n$$

$$\left.\begin{array}{l} p_j'(x_j) \;=\; p_{j+1}'(x_j),\; j = 1, \ldots, n-1 \\ p_j''(x_j) \;=\; p_{j+1}''(x_j),\; j = 1, \ldots, n-1 \end{array}\right\} \textit{Spline conditions}$$

- $4n - 2$ equations

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)
- Force first derivatives at endpoints to match y'(x) – *clamped spline.*

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

Cubic Spline Formulation – 2 Segments

8 Unknowns

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

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

8 Equations

Interpolatory

$$p_1(t_1) = y_1$$

$$p_1(t_2) = y_2$$

$$p_2(t_2) = y_2$$

$$p_2(t_3) = y_3$$

Continuity of Derivatives

$$p_1'(t_2) = p_2'(t_2)$$

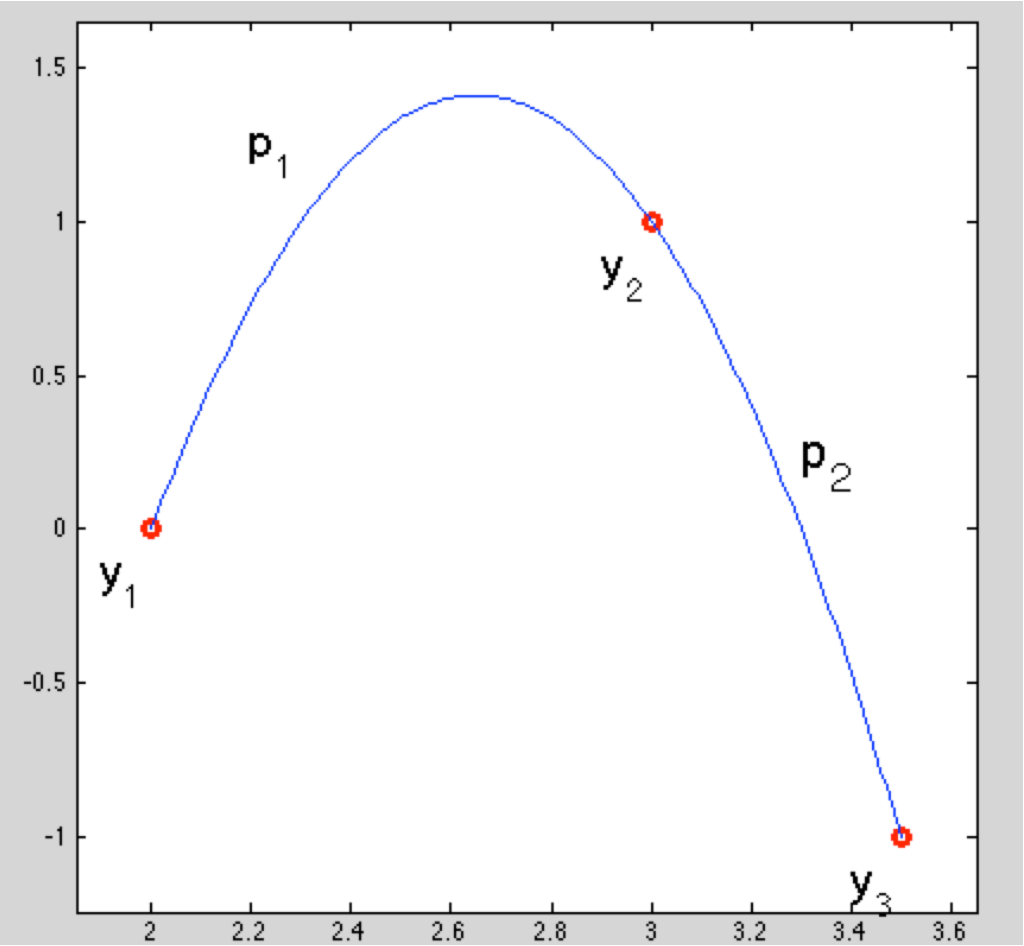
$$p_1''(t_2) = p_2''(t_2)$$

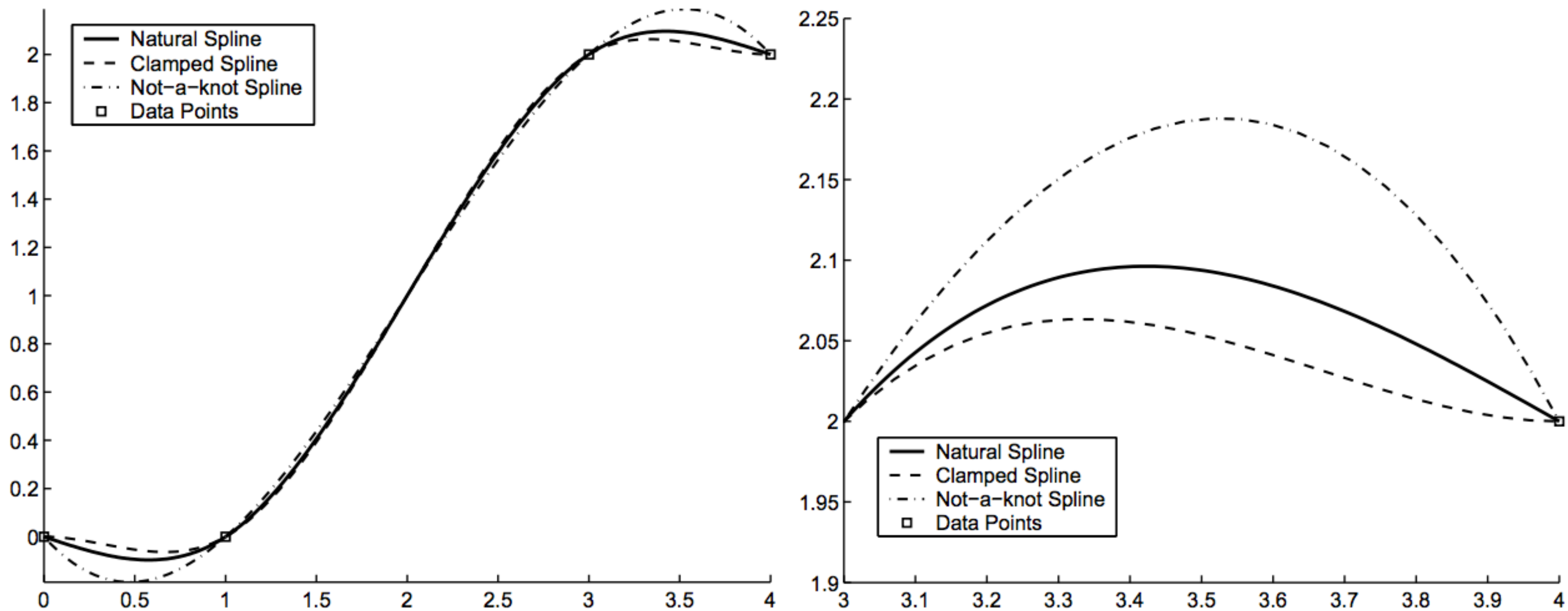
End Conditions

$$p_1''(t_1) = 0$$

$$p_2''(t_3) = 0$$

$(Natural\ Spline)$





Note that the `spline` function in MATLAB computes a not-a-knot spline by default. If `X = [0 1 3 4]` and `Y = [0 0 2 2]`, the not-a-knot spline can be computed and plotted in MATLAB with `plot(x,ppval(spline(X,Y),x))`. Specifying additional data points at the beginning and end of the interval will give a clamped spline with those extra values as the slopes at the endpoints of the intervals. The command `plot(x,ppval(spline(X,[0 Y 0]),x))` would give the clamped spline plotted here with $f'(0) = f'(4) = 0$.

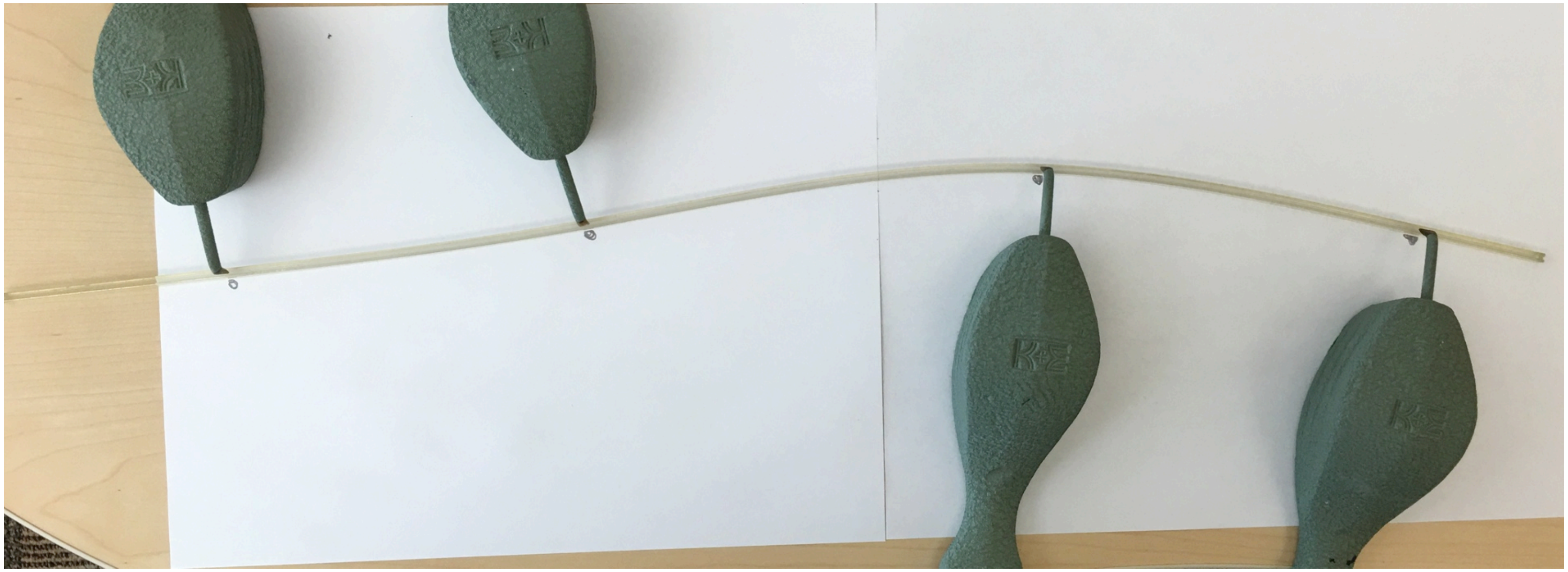
Some Cubic Spline Properties

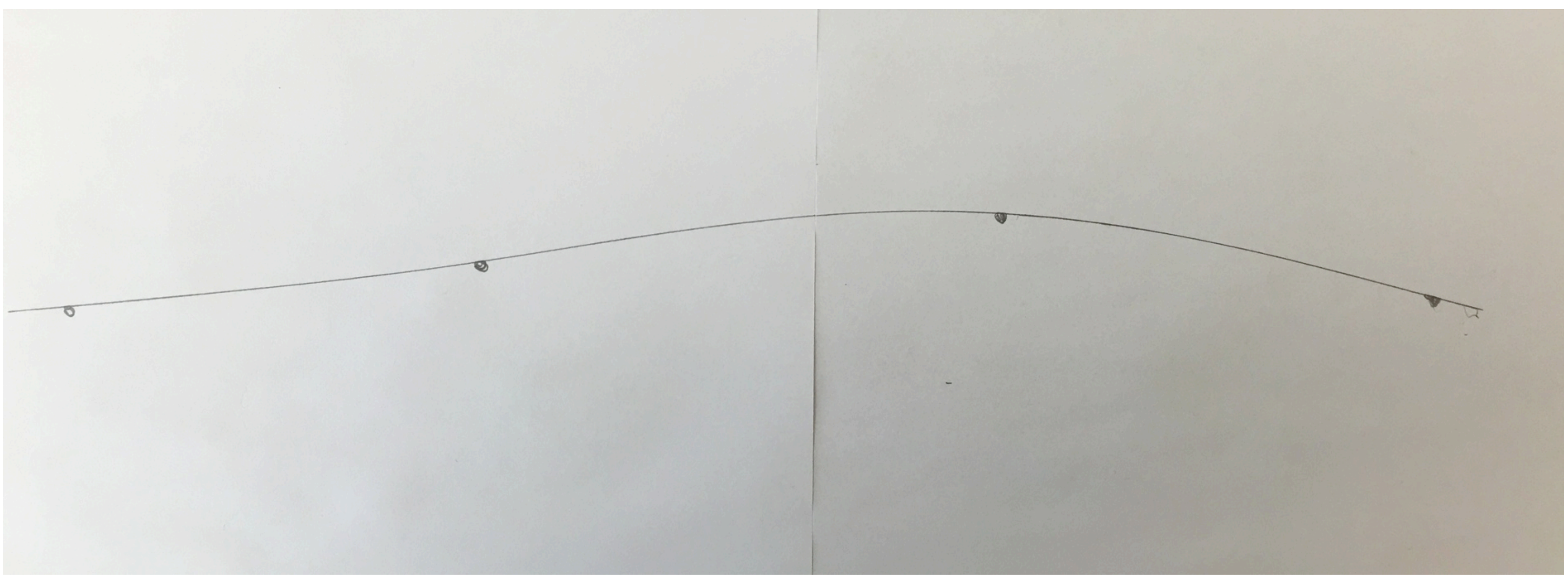
- Continuity
  - 1st derivative: continuous
  - 2nd derivative: continuous
- "Natural Spline" minimizes integrated curvature:

  over all twice-differentiable f(x) passing through (x_j,f_j), j=1,…,n.

$$\int_{x_1}^{x_n} |S''(x)|^2\,dx \leq \int_{x_1}^{x_n} |f''(x)|^2\,dx$$

- Robust / Stable (unlike high-order polynomial interpolation)
- Commonly used in computer graphics, CAD software, etc.
- Usually used in parametric form (DEMO)
- There are other forms, e.g., tension-splines, that are also useful.
- For clamped boundary conditions, convergence is $O(h^4)$
- For small displacements, natural spline is like a ***physical spline.*** ***(DEMO)***

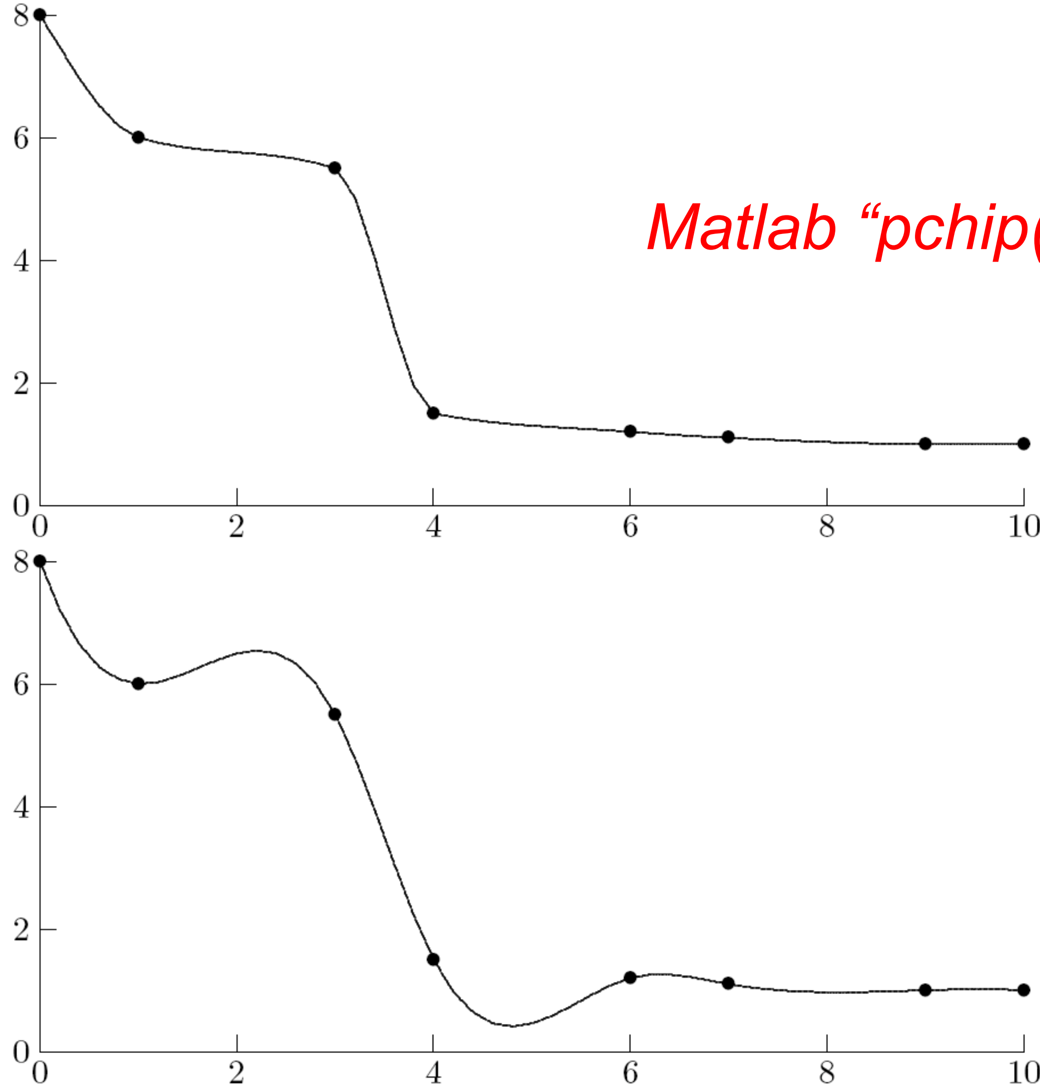




Hermite Cubic vs Spline Interpolation

- 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

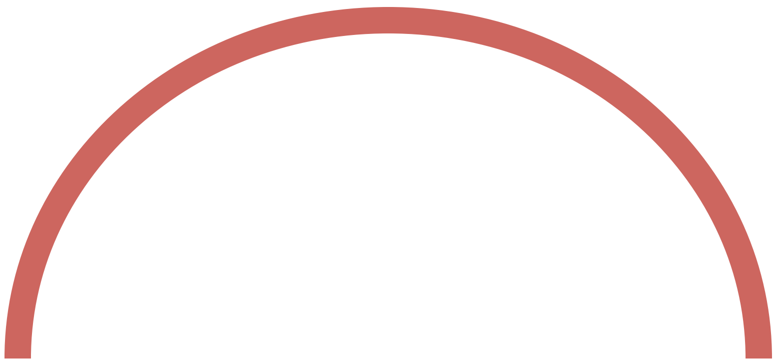
Hermite Cubic vs Spline Interpolation



Matlab "pchip()" function

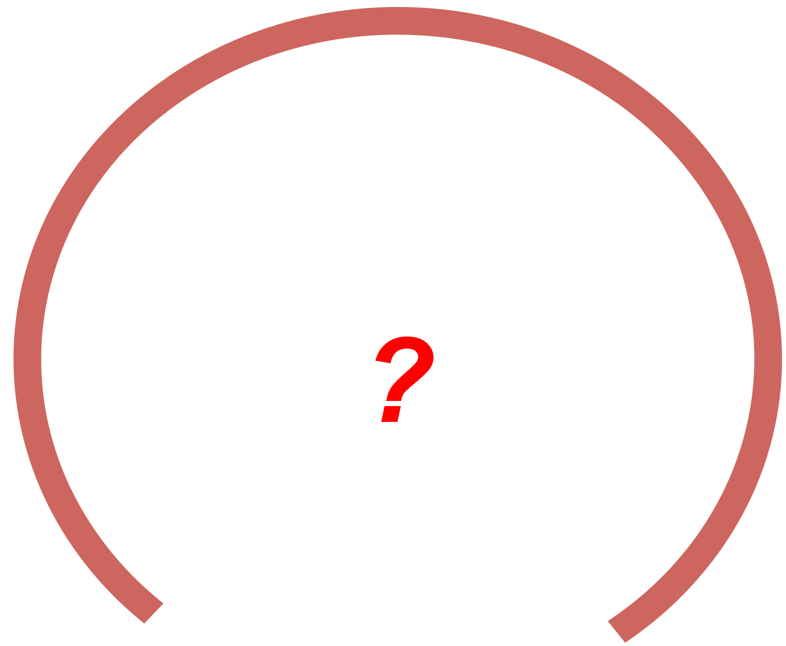
Parametric Interpolation

- It’s clear we can interpolate this:



though maybe not with great accuracy.

But what about this?

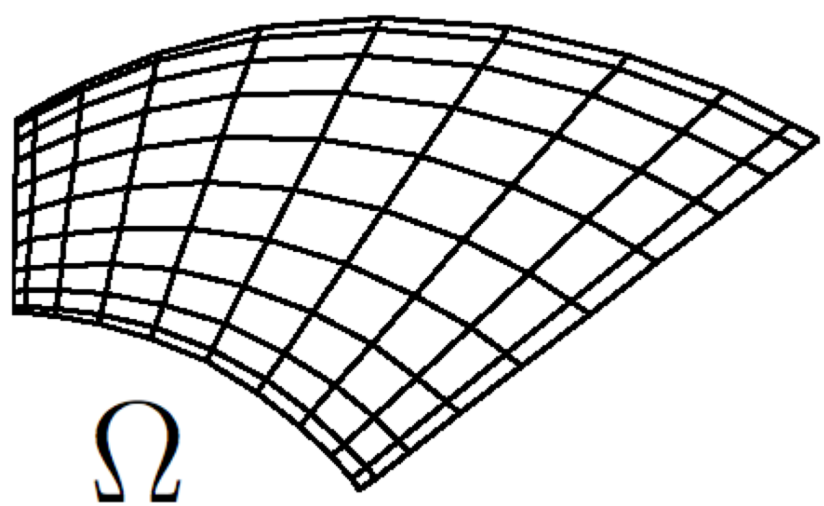


It’s not even a function!

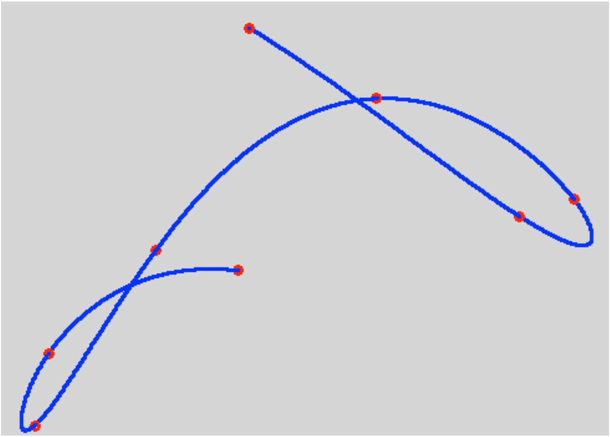
Parametric Interpolation

Two common use cases:

- $y \neq f(x)$ ($y(x)$ is not a function)
- Multidimensional interpolation: $f = f(x, y)$, x, $y \in \Omega$



- If $y \neq f(x)$



- Define a progress variable that is monotonically increasing.
- Build a table:

t_j	x_j	y_j
1	x_1	y_1
2	x_2	y_2
3	x_3	y_3
$\vdots$	$\vdots$	$\vdots$
n	x_n	y_n

- Construct two interpolants:

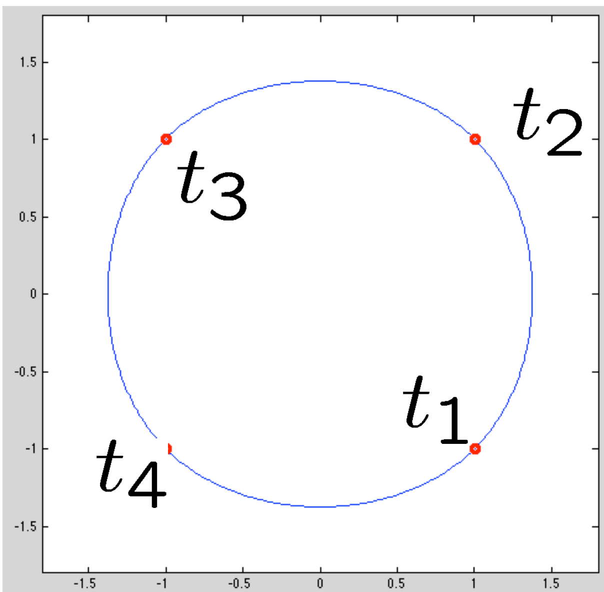
$$\begin{aligned} p_x(t) &= \text{interpolant of } x(t), \quad p_x(t_j) = x_j \\ p_y(t) &= \text{interpolant of } y(t), \quad p_y(t_j) = y_j. \end{aligned}$$

- Then, plot (x_j, y_j) and (p_x, p_y).

Parametric Interpolation

- Important when y(x) is not a function of x; then, define [x(t), y(t)] such that both are (preferably smooth) functions of t.

- Example 1: a circle.



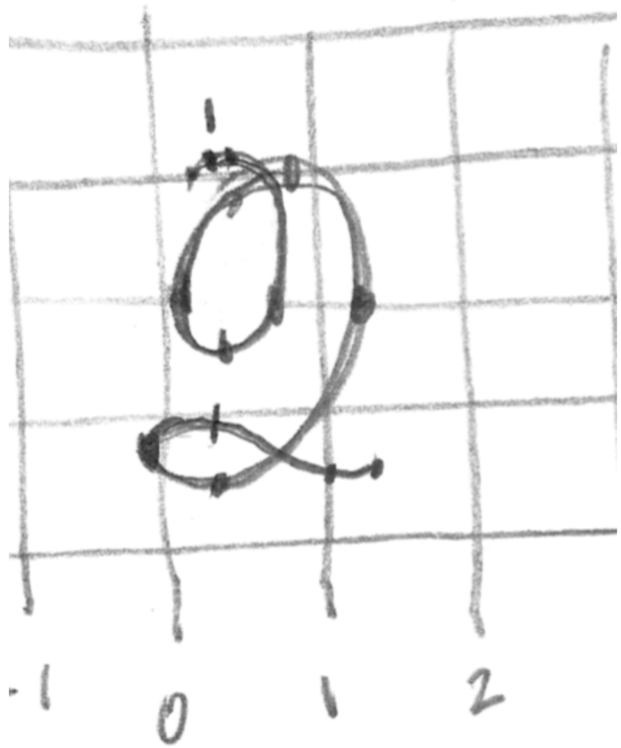

```
%% LAZY WAY TO APPROXIMATE
%% PERIODIC SPLINE

t = -7:8; t=t';

x = [ 1 1 -1 -1 1 1 -1 -1 ]; x=[ x x ];
y = [ -1 1 1 -1 -1 1 1 -1 ]; y=[ y y ];

tt=-2:.01:2;
xx=spline(t,x,tt);
yy=spline(t,y,tt);


hold off;
plot(xx,yy,'b-','LineWidth',1.0); hold on;
plot(x,y,'ro','LineWidth',2.0);
axis equal
axis ([-1.8  1.8  -1.8  1.8 ])
```

Parametric Interpolation: Example 2

- Suppose we want to approximate a cursive letter.
- Use (minimally curvy) splines, parameterized.

Parametric Interpolation: Example 2

i	x_i	y_i
1	.2	3
2	.5	3.2
3	.75	2
	.4	1.6
	.15	2
	.4	2.8
	.9	3
	1.3	2
	.4	.5
	-.15	.8
	.35	.9
	1	.6
	1.4	.6

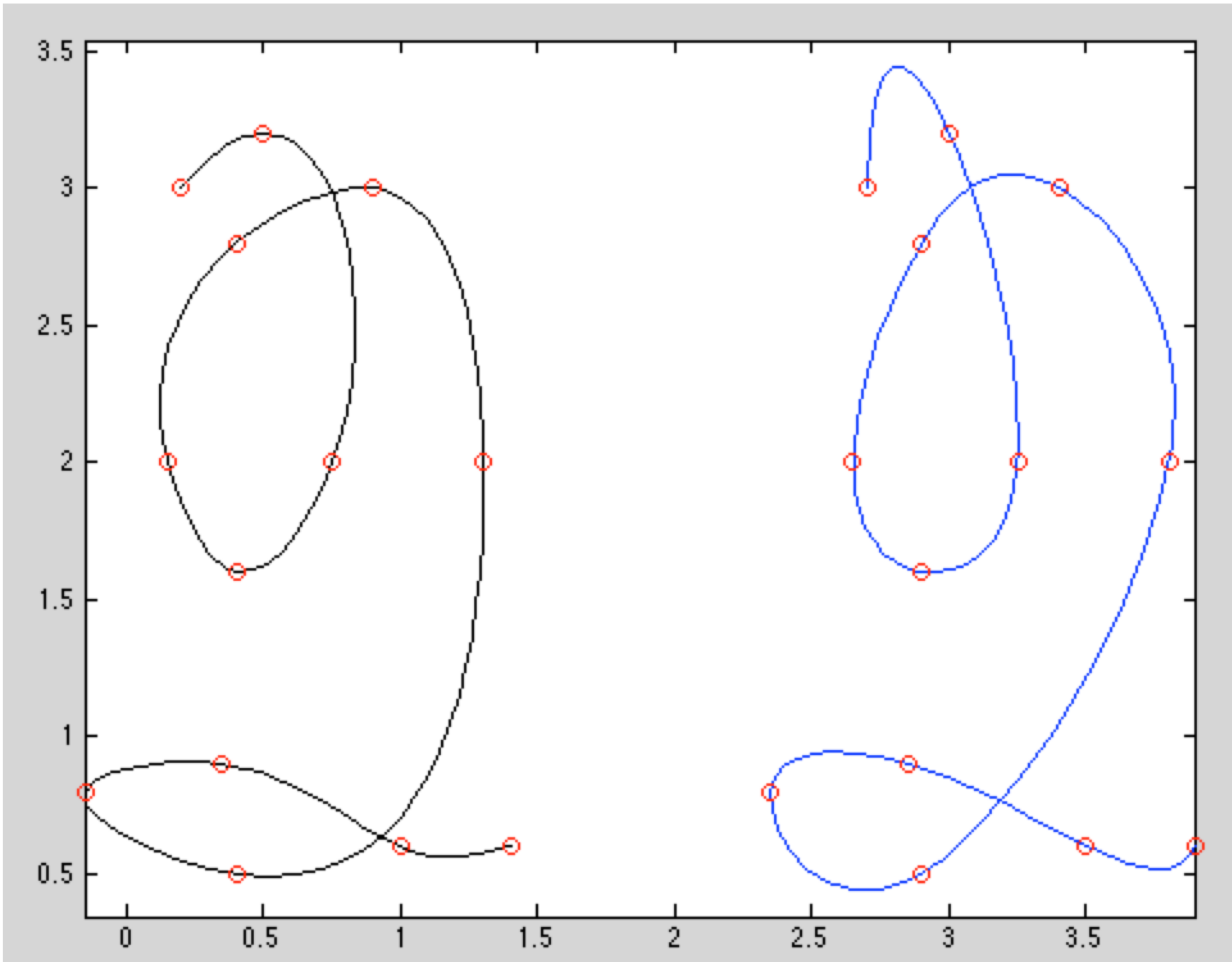


- *Once we have our (x_i , y_i) pairs, we still need to pick t_i .*
- *One possibility: $t_i = i$, but usually it's better to parameterize by arclength, if x and y have the same units.*
- *An approximate arclength is:*

$$s_i \;=\; \sum_{j=0}^{i} ds_j, \quad ds_i := ||\mathbf{x}_i - \mathbf{x}_{i-1}||_2$$

- *Note – can also have Lagrange parametric interpolation…but splines are generally preferable*

Parametric Interpolation: Example 2



Pseudo-Arclength-Based *Index-Based*

Multidimensional Case

- Start with $f(r,s)$, r, $s \in [-1,1]^2$.
- Suppose we know function values $f_{ij} = f(r_i, s_j)$.
- Construct Lagrange interpolant,

$$\begin{aligned} p(r,s) &\in \mathbb{P}_{n-1}(r,s) \\ &= \sum_{j=1}^{n} \sum_{i=1}^{n} l_i(r)\, l_j(s)\, f_{ij}, \quad l_i(r_k) = \delta_{ik}. \end{aligned}$$

  If r_i and s_j are Gauss quadrature points (e.g., zeros of an orthogonal polynomial such as Chebyshev or Legendre), interpolation is stable.

- Example: $f = e^{x \cos 5y}$.

Parametric Interpolation

- Define

$$\begin{aligned} x &= x(r,s) := \sum_{j=1}^{n}\sum_{i=1}^{n} l_i(r)\, l_j(s)\, x_{ij}, \\ y &= y(r,s) := \sum_{j=1}^{n}\sum_{i=1}^{n} l_i(r)\, l_j(s)\, y_{ij}. \end{aligned}$$

- Plot $(x, y, p) = (x(r,s),\ y(r,s),\ p(r,s)\ )$.

Fast Evaluation:

- Suppose:

$$\begin{aligned}\mathbf{r} &= [r_1,\, r_2, \ldots, r_n]^T \quad \text{(on GLC points)}\\ \mathbf{s} &= [s_1,\, s_2, \ldots, s_n]^T\\ \tilde{\mathbf{r}} &= \text{linspace}(-1, 1, m \gg n), \quad \tilde{\mathbf{s}} = \tilde{\mathbf{r}}.\end{aligned}$$

- Define $\mathbf{x},\ \mathbf{y},\ \mathbf{f}\ \in\ \mathbb{R}^{n\times n}$: *source interpolation nodes.*
- Define $\tilde{\mathbf{x}},\ \tilde{\mathbf{y}},\ \tilde{\mathbf{f}}\ \in\ \mathbb{R}^{m\times m}$: *target interpolation points.*

- It appears we need m^2 interrogations of $l_i(r)l_j(s)$, $i, j\ \in\ [1, \ldots, n]^2$.
- However, tensor-product forms allow this to be done in $O(nm^2)$ time, rather than $O(n^2m^2)$ time.

- Form $J_{ij} \; := \; l_j(\tilde{r}_i)$.
- Then,

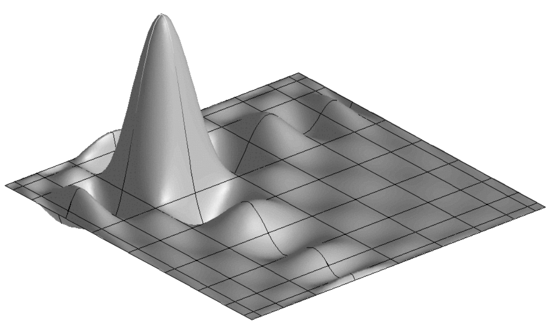
$$
\begin{aligned}
x_{ij} \; &= \; \sum_{q=1}^{n} \sum_{p=1}^{n} l_p(\tilde{r}_i) l_q(\tilde{s}_j)\, x_{pq} \\
&= \; \sum_{q=1}^{n} \sum_{p=1}^{n} J_{ip}\, x_{pq}\, J^T_{qj}.
\end{aligned}
$$

- In matrix form, $\tilde{X} \; = \; J X J^T$.
- In vector form, $\tilde{\mathbf{x}} \; = \; (J \, \otimes \, J)\mathbf{x}$.
- *A lot faster than evaluating m^2 matrix exponentials!*

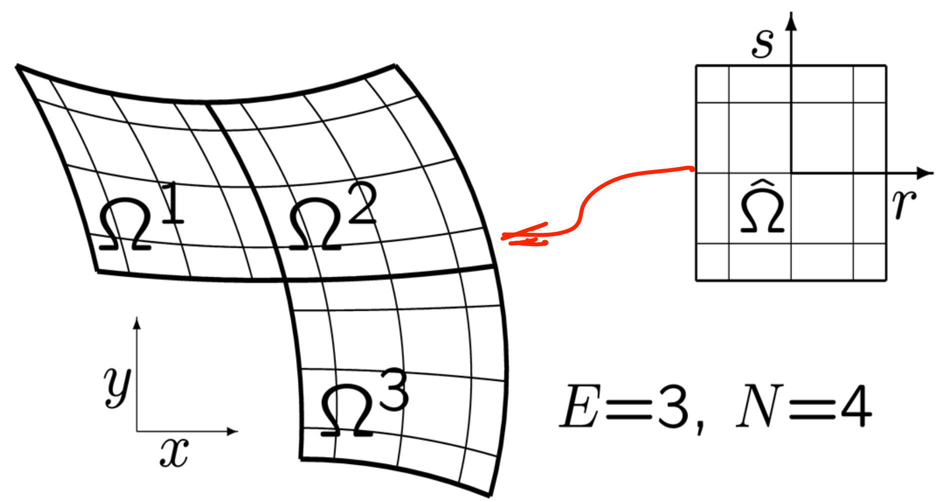
Multidimensional Interpolation

- Multidimensional interpolation has many applications in computer aided design (CAD), partial differential equations, high-parameter data fitting/assimilation.

- Costs considerations can be dramatically different (and of course, higher) than in the 1D case.



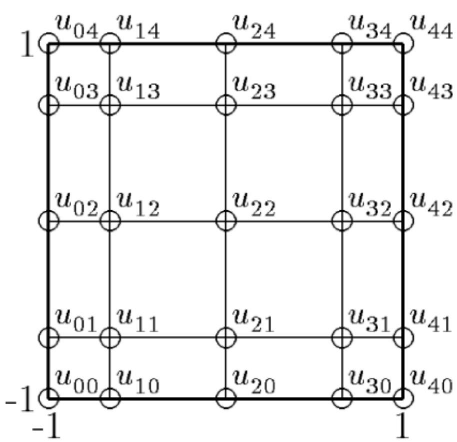
2D basis function, N=10

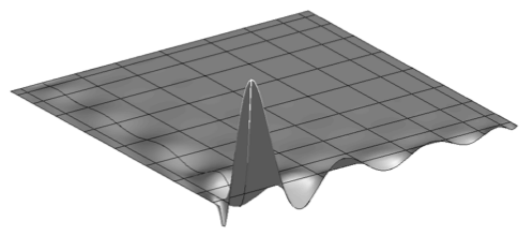


Multidimensional Interpolation

- There are many strategies for interpolating f(x,y) [or f(x,y,z), etc.].
- One easy one is to use tensor products of one-dimensional interpolants, such as bicubic splines or tensor-product Lagrange polynomials.

$$p_n(s,t) \;=\; \sum_{i=0}^{n}\sum_{j=0}^{n} l_i(s)\, l_j(t)\, f_{ij}$$





2D Example: n=2