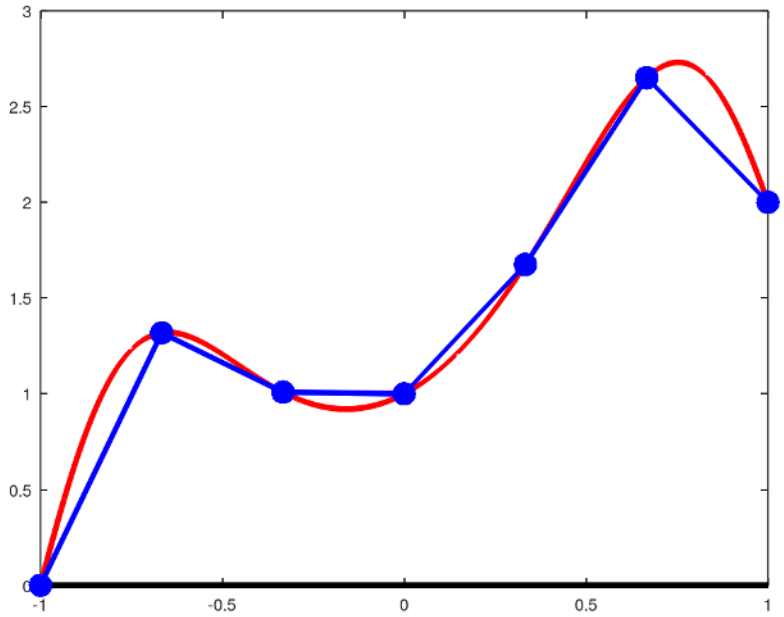
Chapter 7: Interpolation

Topics:

- Background
- Examples
- Polynomial interpolation: bases, error, Chebyshev, piecewise
- Orthogonal polynomials
- Splines: error, end conditions
- Parametric interpolation
- Multivariate interpolation: $f(x, y)$

Classic Interpolation: Piecewise Linear

- Here, we have n interpolation points (blue dots)
- The *interpolant* is the blue line, $p(t) \approx f(t)$ (the red line)



- This is generally the type of interpolation used in look-up tables
- Questions:
  - What is the error (e.g, $\max_x |p(t) - f(t)|$) as a function of n?
  - Is the interpolation error bounded as $n \longrightarrow \infty$?
  - Can we improve the *rate of convergence*? (e.g., reduced error for same n or same error for lower n)
  - Are there properties, such as monotonicity, that we should preserve?
  - etc.

Piecewise Linear

- Error vs h?
- Piecewise constant?

Interpolation Problem Definition

- Basic interpolation problem: *For given data*

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

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

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

(interpolatory condition)

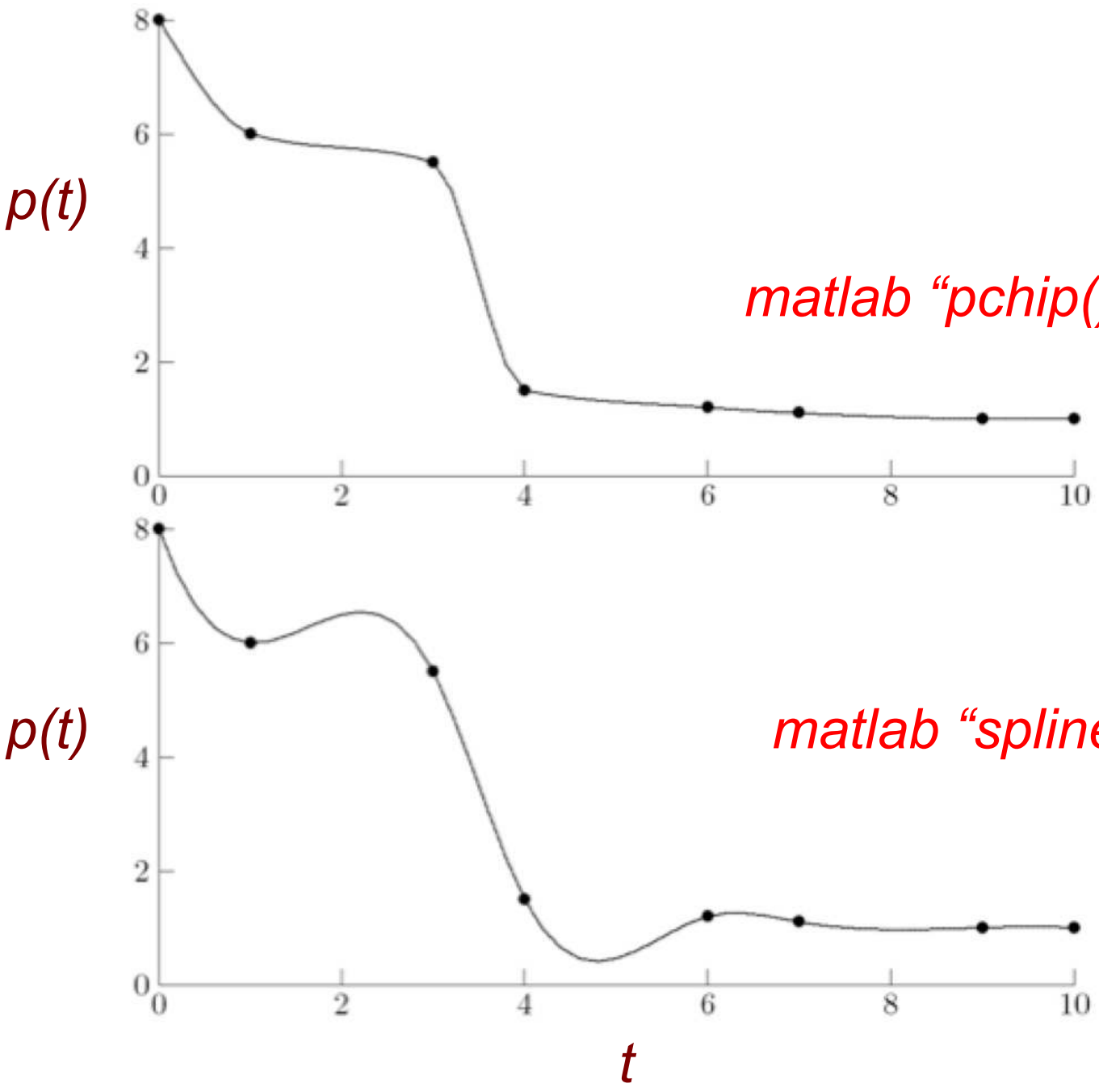
- f *is* interpolating function, *or* interpolant, *for given data*

- *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 a function of more than one variable*

Example



Purposes for Interpolation

- Look-up tables
- Plotting smooth curve through discrete points
- Differentiating or integrating tabular data
- Quick and easy evaluation of mathematical function
- Replacing complicated function by simple one
- Basis functions for approximation in numerical solution of ordinary and partial differential equations (example)
- Basis functions for developing integration rules
- Basis functions for developing differentiation techniques

Interpolation vs Approximation

- *Interpolatory condition,* $f(t_i) = y_i$ implies that interpolant fits given data points exactly
- Interpolation is *not appropriate* 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
- curve_fit.m

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.)

- Is it easy to update the interpolant if new information is provided? (e.g., Newton divided-difference approach)

Functions for Interpolation

- Families of functions commonly used for interpolation include
  - polynomials
  - piecewise polynomials (e.g., piecewise linear)
  - trigonometric functions
  - exponential functions
  - rational functions

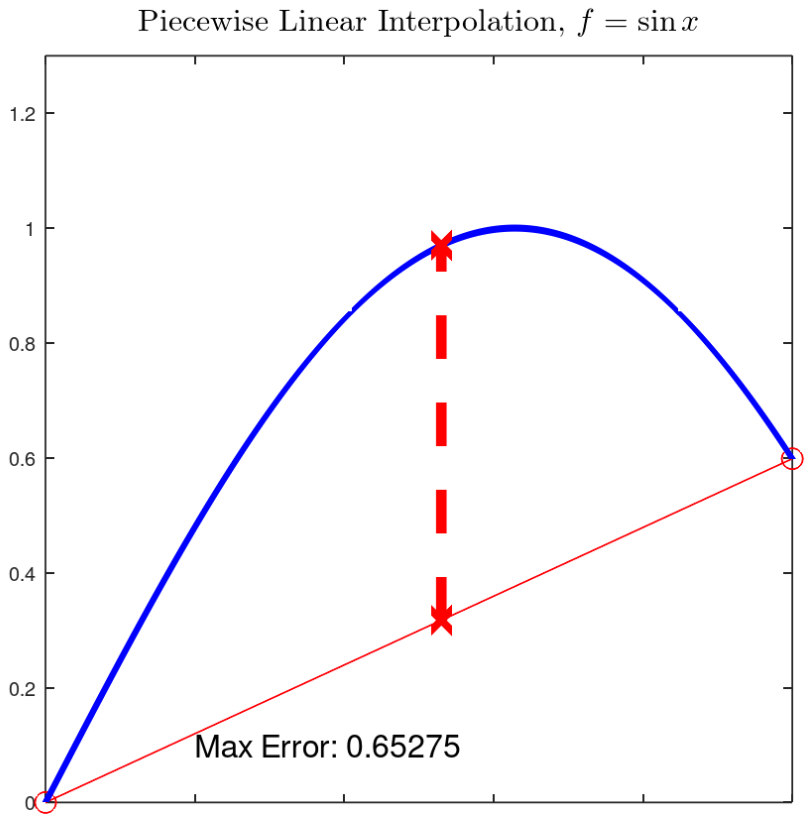
- Primarily focus on polynomials and piecewise polynomials

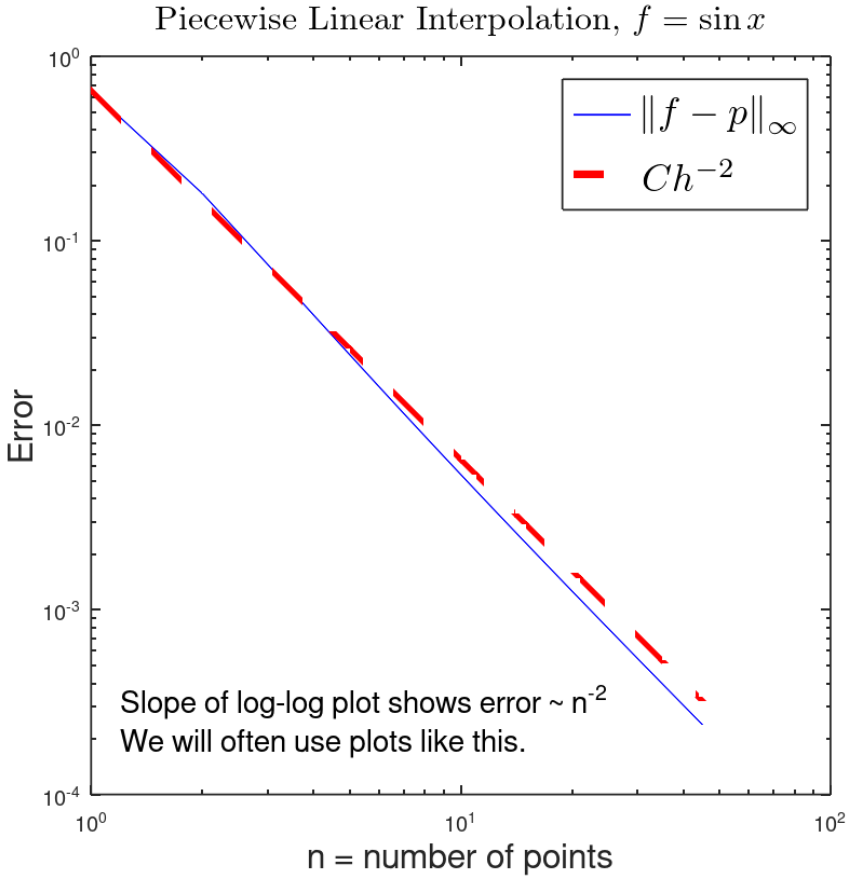
- `interp_lin.m`

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 basis chosen wisely:
  - t_i = roots of orthogonal polynomials
  - Convergence is exponential: $err \sim Ce^{-\sigma n}$, instead of algebraic: $err \sim Cn^{-k}$

Error for Piecewise Linear Interpolation of sin(x) on [0:2.5]





interp_lin.m

Example: Table Look-Up

- Suppose you're tasked with tabulating data such that linear interpolation between tabulated values is correct to 4 digits.

x	$Si(x) = \int_0^x \frac{\sin t}{t} dt$	δ^2	$Ci(x) = \int_\infty^x \frac{\cos t}{t} dt$	δ^2
41.00	1.59494 33514	-23986	-0.00327 89946	+ 4456
.01	9490 34645	23936	0351 95823	4695
.02	9486 11840	23881	0375 97005	4932
.03	9481 65154	23826	0399 93255	5170
.04	9476 94642	23766	0423 84335	5405
41.05	1.59472 00364	-23705	-0.00447 70010	+ 5642
.06	9466 82381	23642	0471 50043	5878
.07	9461 40756	23577	0495 24198	6110
.08	9455 75554	23508	0518 92243	6347
.09	9449 86844	23439	0542 53941	6577
41.10	1.59443 74695	-23365	-0.00566 09062	+ 6811
.11	9437 39181	23290	0589 57372	7042
.12	9430 80377	23214	0612 98640	7273
.13	9423 98359	23134	0636 32635	7502
.14	9416 93207	23053	0659 59128	7732
41.15	1.59409 65002	-22967	-0.00682 77889	+ 7959
.16	9402 13830	22883	0705 88691	8187
.17	9394 39775	22793	0728 91306	8412
.18	9386 42927	22703	0751 85509	8639
.19	9378 23376	22609	0774 71073	8862

- **Q:** How many table entries are required on the interval [0:1]?
- **Q:** How many digits should you have in the tabulated data?
- To answer this question we will need some error estimates

Error for Global Polynomial Interpolant

- Suppose that $f(x) \in C^n$ on interval $[a, b]$ that contains $x_1,\ x_2,\ \ldots, x_n,\ x$

- If $p(x) \in \mathbb{P}_{n-1}$ (the space of polynomials of degree $\leq\ n-1$) and $p(x_j) = f(x_j),\ j = 1, \ldots, n$, then there exists a $\theta \in [a, b]$ 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* on $[x_1 : x_2]$ we have

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

- The L^∞ error bound is

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

where $M := max_{[x_1:x_2]}|f''(x)|$ and $h := x_2 - x_1$

Table Look-Up, continued

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

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

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

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

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

$$h^2 \leq 4 \times 10^{-4}$$

$$h \leq 0.02$$

- The table would look like

x	$\cos(x)$
0.00	1.00000
0.02	0.99980
0.04	0.99920
0.06	0.99820
0.08	0.99680

Piecewise Polynomial Interpolation

- Example – Given the table below,

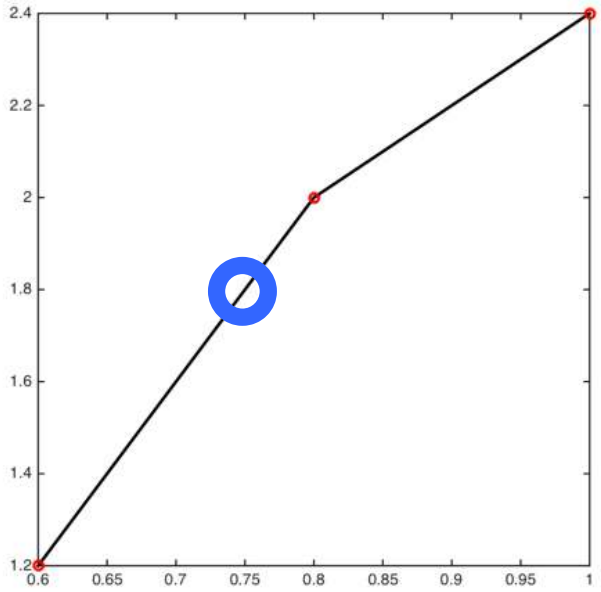
x_j	f_j
0.6	1.2
0.8	2.0
1.0	2.4

- 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 $< (0.2)^2 f'' / 8$.

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 (this is LLSQ case...)
- If $m < n$, interpolant is not unique
- If $m = n$, then basis matrix $\mathbf{A}$ is nonsingular provided data points t_i are distinct and basis functions are linearly independent, so data can be fit exactly
- Sensitivity of parameters $\mathbf{x}$ to perturbations in data depends on $\text{cond}(\mathbf{A})$, which depends on choice of basis functions

Basis Functions

- Family of functions for interpolating given data points is spanned by *basis functions*, $\phi_1(t), \ldots, \phi_n(t)$

- 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

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

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

Basis Function Example: Fourier

- For functions that are periodic on, say, $[0, L]$, we can consider the *Fourier* bases comprising sines and cosines.

- For example, consider $L = 2\pi$ and let

$$[\phi_1(t)\ \ \phi_2(t)\ \ \cdots\ \ \phi_n(t)]\ =$$

$$[\cos 0t\ \ \cos 1t\ \ \cdots\ \cos kt\ \ \sin 1t\ \ \cos 2t\ \ \cdots\ \cos(k-1)t]$$

  be a set of basis functions with $n = 2k$.

- This basis is stably (and optimally) sampled by n uniformly-spaced points with spacing $\Delta t = L/n$, which means we do not include *both* endpoints.

- A typical distribution is $t_i = 2\pi i/n$, for $i = 1, \ldots, n$ with $L = 2\pi$

- We build the Vandermonde matrix with entries $a_{ij} = \phi_j(t_i)$ and solve $\mathbf{Ax} = \mathbf{y}$, where $\mathbf{y}_i = f(t_i)$ holds the function values at the test points

Fourier Example, continued

- Once we have the basis coefficients, $\mathbf{x}$, we can evaluate the interpolant on a *finer set* of points $\{\bar{t}_i\}$ for plotting or other purposes

- Let $\bar{\mathbf{A}}$ be an $m \times n$ matrix with $\bar{a}_{ij} = \phi_j(\bar{t}_i)$

- For plotting purposes, we typically take $m \gg n$ and $\bar{t}_i = 2\pi i/m$ for $i = 0, \ldots, m$

- Unlike the interpolation problem, we do not need to worry about solvability for the reconstrunction matrix, $\bar{\mathbf{A}}$, so we can admit the redundant row pair that results from $t = 0$ and $t = 2\pi$

- The finely-sampled output is then

$$\bar{\mathbf{y}} = \bar{\mathbf{A}}\mathbf{x} = \bar{\mathbf{A}}\mathbf{A}^{-1}\mathbf{y}$$

  and we can plot $(\bar{t}_i, \bar{y}_i)$

drive_cs2.m

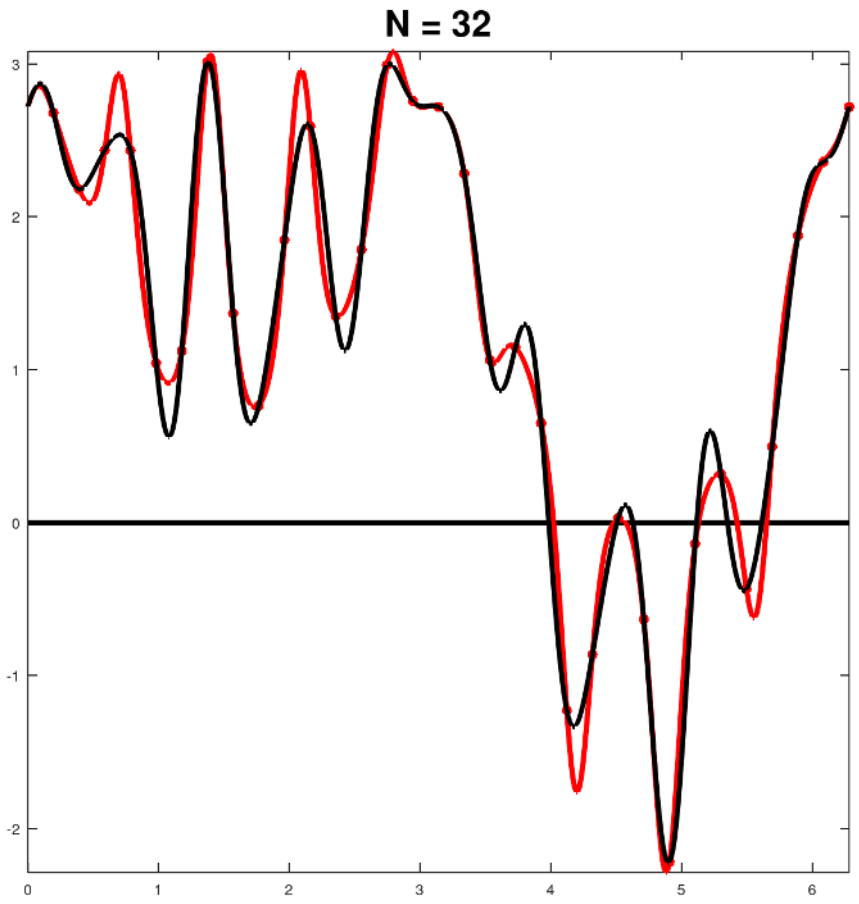
Fourier Example, continued

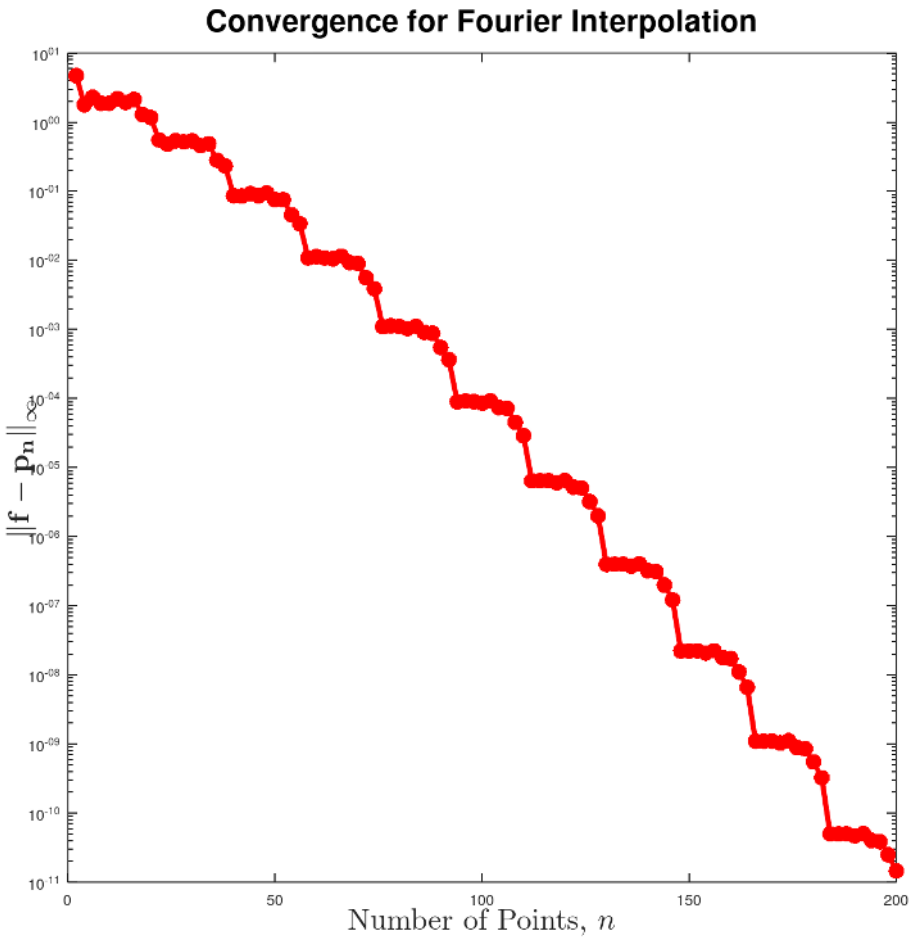
drive_cs2.m

- Here is a code that implements interpolation with the cosine/sine basis on n points

```
%% Generalized Van der Monde example, Fourier
hdr;

if mod(N,2)=1; N=N+1; end; % Force N to be even
Nf=20+18*N;                % Nf >> N, fine points

[z,w]  =zwuni(N);  x  = pi*(z (2:end)+1);   %  retain only one end-point
[zf,wf]=zwuni(Nf); xf = pi*(zf(1:end)+1);   %  retain both, for plotting


wc=[0:N/2]';    % Cosine wavenumbers
ws=[1:N/2-1]';  % Sine wavenumbers


C  = cos(x*wc');  S  = sin(x*ws');  A  = [C S];   %% Interpolation matrix
Cf = cos(xf*wc'); Sf = sin(xf*ws'); Af = [Cf Sf]; %% Fine-mesh reconstructor

f  = exp(cos(2*x))+sin(x).*exp(cos(9*x));      % coarse mesh input
ff = exp(cos(2*xf))+sin(xf).*exp(cos(9*xf));   % fine-mesh solution, verification

fN = Af*(A\f);                                 % interpolant on fine mesh

plot(xf,0*xf,'k-',lw,2,xf,ff,'r-',lw,2,x,f,'r.',ms,12,xf,fN,'k-',lw,1.9);
axis tight;

max(max(abs(fN)));

eold  = enorm; iold=inorm;
enorm = norm(ff-fN)/norm(ff);
inorm = max(abs(ff-fN));
disp([ N enorm eold/enorm inorm iold/inorm])
```

Fourier Example, continued





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
- If $f(t)$ is smooth and t_is are carefully chosen, polynomial interpolant converges rapidly to f
- There are many ways to represent or compute interpolating polynomial but, in theory (i.e., in infinite-precision arithmetic), all must give the same result

Monomial Basis (bad)

- *Monomial basis functions*

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

  give interpolating polynomial of form

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

- Coefficients $\mathbf{x}$ given by the $n \times n$ linear system

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

- Matrix of this form is called *Vandermonde matrix*

- We will also refer to a matrix having entries $\phi_j(t_i)$ as a *generalized* Vandermonde matrix, or simply 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

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

- For these particular data, system is

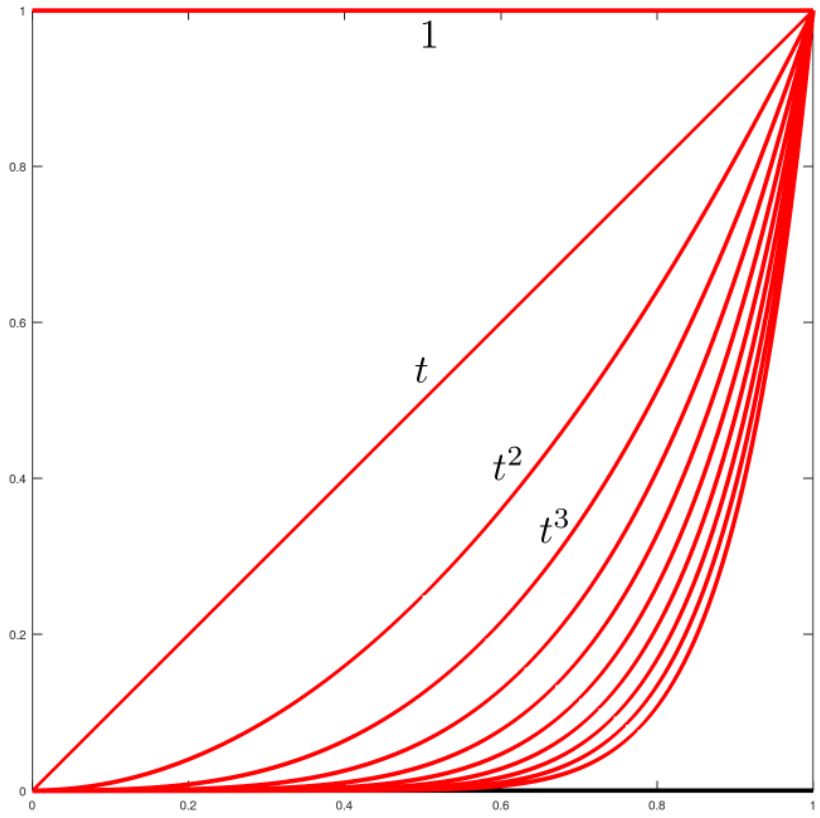
$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

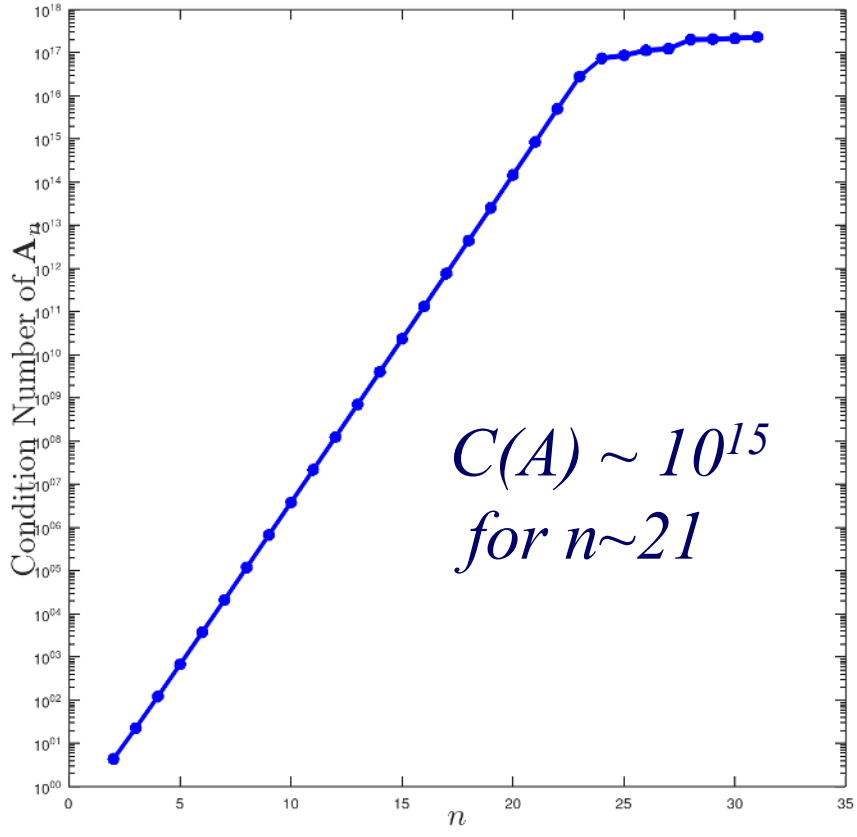
- Solution is $\mathbf{x} = [-1 \;\; 5 \;\; -4]^T$, so interpolating polynomial is

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

Monomial Basis, continued

- Monomial basis functions become similar as n becomes large
- Matrix $\mathbf{A}$ is increasingly *ill-conditioned* as degree increases
- Condition number of $\mathbf{A}$ grows exponentially with n
- Resulting basis coefficients will be poorly determined





Monomial Basis, continued

- Solving system $\mathbf{Ax} = \mathbf{b}$ using standard linear equation to determine basis coefficients $\mathbf{x}$ requires $O(n^3)$ work

- Conditioning and amount of computational work can be improved by using a different basis

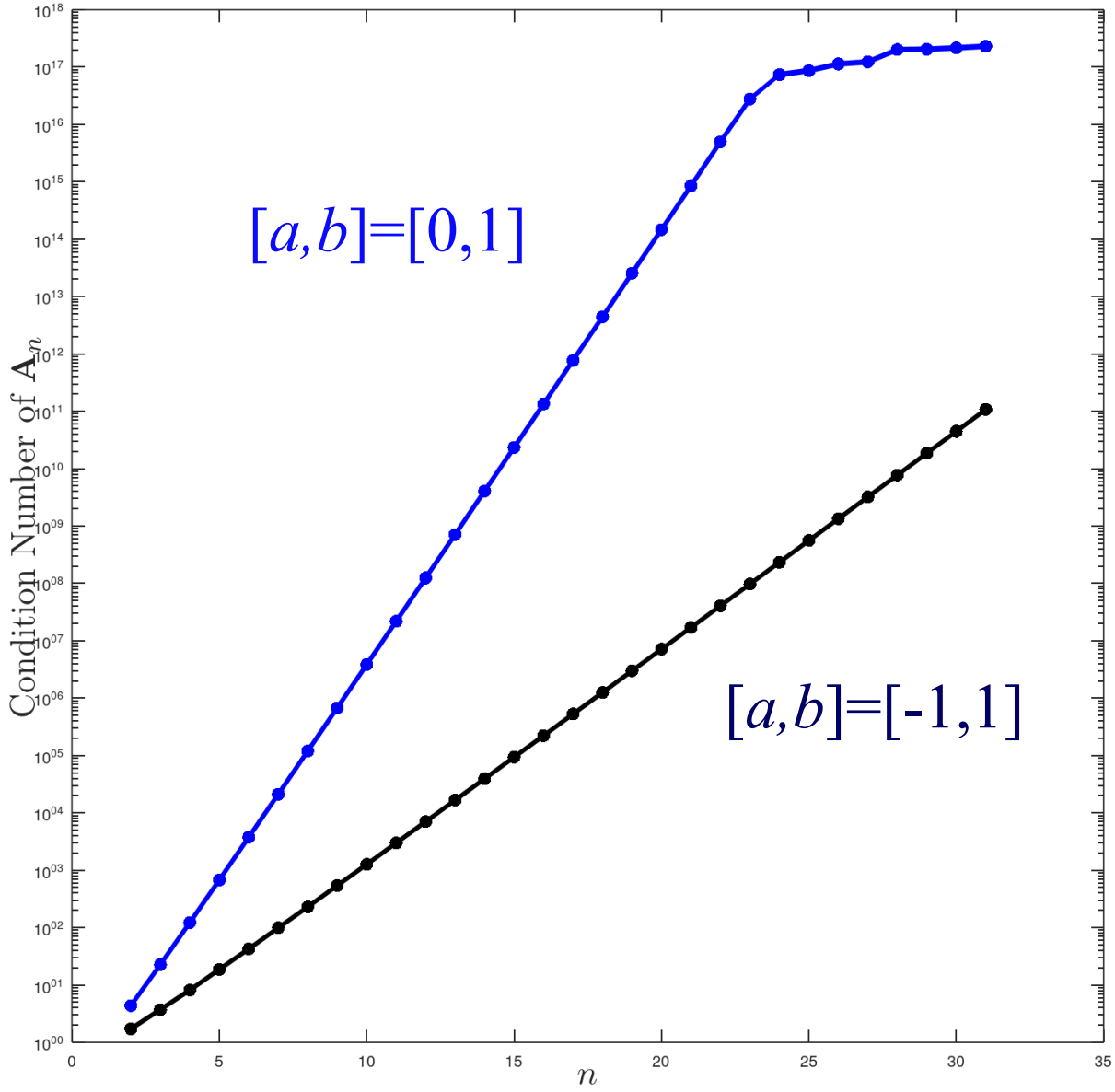
- For example, can improve conditioning by centering and scaling the independent variable from $t \in [a, b]$ to $\xi \in [-1, 1]$ via the transformation

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

  where $a = \min t_j$, $b = \max t_j$, $c = (b + a)/2$ is midpoint and $d = (b - a)/2$ is half of the range of the data

- Condition improves, but still exhibits exponential growth with n

Monomial Basis, continued



Monomial Basis, continued

- More progress can be made by *changing basis* to something other than monomials
- *Orthogonal polynomials* provide a good alternative
- *Lagrange polynomials*, with a good choice of nodal points, are also effective
- Change of basis still gives the same interpolating polynomial for given data, but *representation* of polynomial will be different

Uniqueness of Interpolating Polynomial

- Suppose you have two polynomial interpolants, $p(t)$ and $q(t)$ of degree $n-1$, each satisfying the interpolatory condition

$$p(t_i) \;=\; q(t_i) \;=\; y_i$$

- Then $p(t) \equiv q(t)$.

- That is, assuming exact arithmetic, the interpolating polynomial is unique, no matter how constructed.

- **Proof.**
  - Denote $\mathbb{P}_{n-1}$ as the space of polynomials of degree $\leq\; n-1$.
  - Both p and $q \in \mathbb{P}_{n-1}$
  - Therefore, $d := p - q \in \mathbb{P}_{n-1}$.
  - However, $d(t_i) = p(t_i) - q(t_i) = 0$, $i = 1, \ldots, n$.
  - The only polynomial in $\mathbb{P}_{n-1}$ that has n roots is the zero function, $d \equiv 0$, which establishes the uniqueness result

Lagrange Interpolation

- Because of its convenience and stability, *Lagrange interpolation* is one of the most commonly used polynomial interpolants for science and engineering and applications

- It avoids the ill-conditioning of the Vandermonde matrix by building a polynomial basis that automatically satisfies the interpolatory conditions

Lagrange Bases

- For a given set of interpotion points, t_i, $i = 1, \ldots, n$, the Lagrange polynomial basis functions (also known as *cardinal* functions) are defined as follows,

$$l_j(t) \in \mathbb{P}_{n-1}, \qquad l_j(t_i) = \delta_{ij} = \begin{cases} = 1, \ i = j \\ = 0, \ i \neq j \end{cases}, \quad i, j = 1, \ldots, n$$

- Consequently, the generalized Vandermonde matrix $\mathbf{A}$ has entries $a_{ij} = \delta_{ij}$, which means that $\mathbf{A} = \mathbf{I}$, the $n \times n$ identity matrix

- Moreover, the basis coefficients are *the function values*, y_i

- Thus, Lagrange polynomial interpolating data (t_i, y_i) is

$$p_{n-1}(t) = y_1 l_1(t) + y_2 l_2(t) + \cdots + y_n l_n(t)$$

Construction of the Cardinal Functions

- Lagrange interpolation relies on satisfying the conditions

  1. $l_j(t) \in \mathbb{P}_{n-1}$
  2. $l_j(t_i) = 0,\ i \neq j$
  3. $l_j(t_j) = 1$

- To meet the first two conditions, consider the polynomial

$$l_j(t) = C_j(t - t_1) \cdots (t - t_{j-1})\ (t - t_{j+1}) \cdots (t - t_n),$$

  which contains all terms in the nth-order polynomial

$$q_n := (t - t_1)(t - t_2) \cdots (t - t_n)$$

  save for the jth one.

- Consequently, $l_j(t_i) = 0$ for $i \in [1, \ldots, n],\ i \neq j$.

- If C_j is a constant, then $l_j(t)$ is a polynomial with $n-1$ roots and therefore satisfies Condition 1

Construction of the Cardinal Functions

- Lagrange interpolation relies on satisfying the conditions

  1. $l_j(t) \in \mathbb{P}_{n-1}$
  2. $l_j(t_i) = 0,\ i \neq j$
  3. $l_j(t_j) = 1$

- To satisfy Condition 3, we need to scale C_j so that $l_j(t_j) = 1$.

- This condition is met by setting

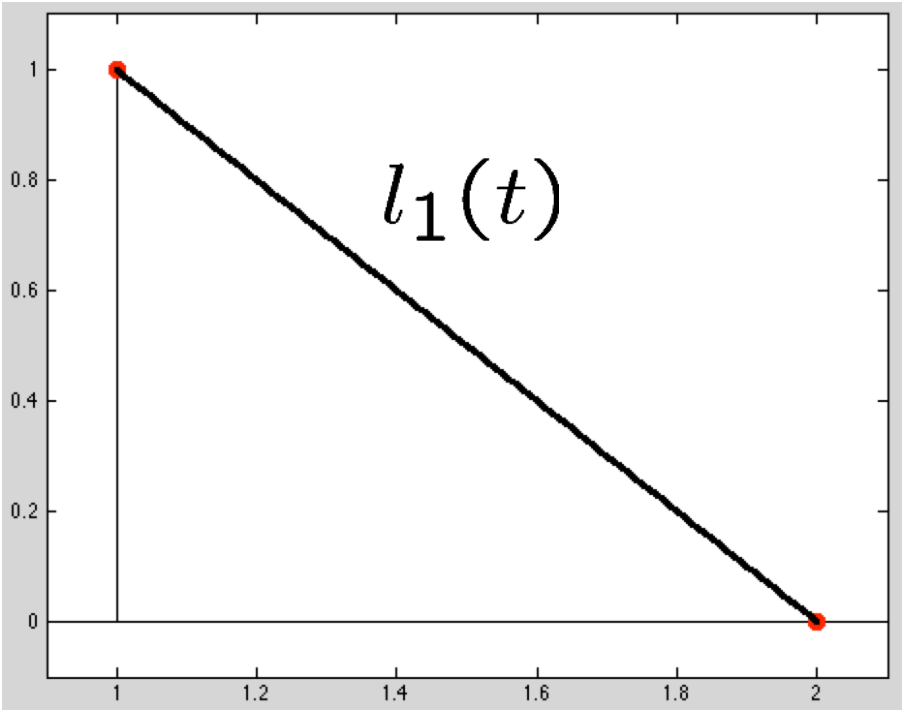
$$C_j = [(t_j - t_1) \cdots (t_j - t_{j-1})\ (t_j - t_{j+1}) \cdots (t_j - t_n)]^{-1}$$

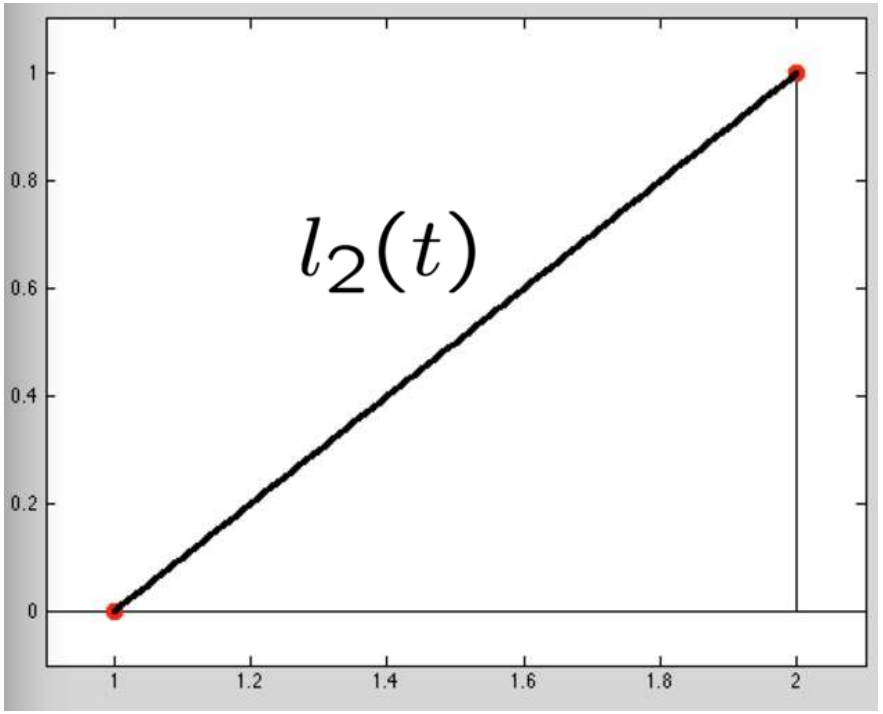
- A compact representation of l_j is

$$l_j(t) = \prod_{i \neq j}^{n} \frac{t - t_i}{t_j - t_i}$$

Lagrange Basis Functions, n=2 (linear)

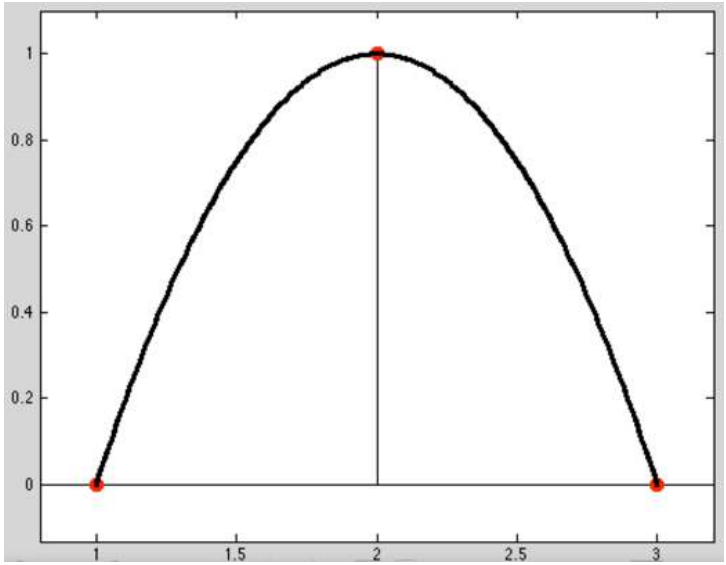
- Here, we have *two* basis functions



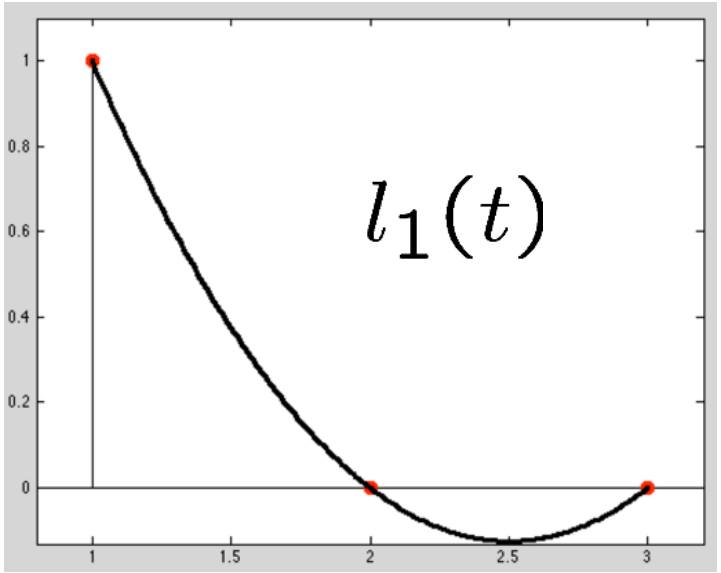


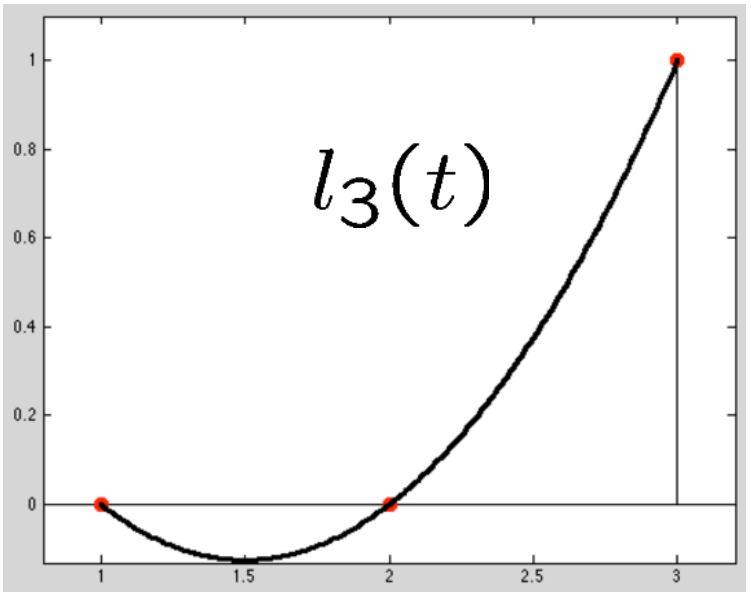
Lagrange Basis Functions, n=3 (quadratic)

- Here, we have *three* basis functions



$l_2(t)$





Fast Lagrange Evaluation

- We construct the Lagrange (cardinal) basis functions as follows:

$$l_j(x) \;=\; \alpha_j\,(x-x_1)(x-x_2)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n),$$

  which clearly is in $\mathbb{P}_{n-1}$ and satisfies $l_j(x_i)=0$ when $i\neq j$.

- To get the scaling condition, $l_j(x_j)=1$, set

$$\alpha_j \;=\; [(x_j-x_1)(x_j-x_2)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)]^{-1}.$$

- A compact form for $l_j(x)$ is

$$l_j(x) \;=\; \prod_{k\neq j}\left(\frac{x-x_k}{x_j-x_k}\right)$$

- However, the first form involving α_j is generally faster for multiple queries (multiple evaluations with different x values).

- To implement the fast form for a given x, first define

$$\begin{array}{lcl} s_1 &=& 1 \\ s_2 &=& s_1\cdot(x-x_1) \\ s_3 &=& s_2\cdot(x-x_2) \\ &\vdots& \\ s_n &=& s_{n-1}\cdot(x-x_{n-1}) \end{array} \qquad \begin{array}{lcl} t_n &=& 1 \\ t_{n-1} &=& t_n\cdot(x-x_n) \\ t_{n-2} &=& t_{n-1}\cdot(x-x_{n-1}) \\ &\vdots& \\ t_1 &=& t_2\cdot(x-x_2) \end{array}$$

  then set $l_j(x)=\alpha_j\cdot s_j\cdot t_j$, for $j=1,\ldots,n$.

Fast Lagrange Evaluation

- Notice that this form requires $O(n)$ operations for n outputs, i.e., the cost is *linear* in n.

- To be precise, the cost to interpolate from n input points to m output values is $O(n^2)$ for generating the full set of α_js, and $O(mn)$ for generating m results.

- For certain choices of interpolation nodes (e.g., Chebyshev points), barycentric formulas eliminate the $O(n^2)$ cost and allow $n = 10^6$ in just seconds on a laptop!
  (See L.N. Trefethen, *Approximation Theory and Approximation Practice*, Oxford.)

- Often, sample points and nodes are unchanged, but $f(x)$ changes.
  In this case, recognize that

$$\begin{aligned} p(\tilde{x}_i) &= \sum_{j=1}^{n} l_j(\tilde{x}_i)\, f_j \\ \mathbf{p} &= J\mathbf{f}, \quad \text{with } J_{ij} := l_j(\tilde{x}_i). \end{aligned}$$

  For moderate m and n (say, < 200) this matrix-vector product approach is ideal, especially if you have $O(n)$ right-hand sides or more.

Stopped Here, 4/3/25

Newton Basis

- Start with $p_n(t) \in \mathbb{P}_{n-1}$ and $p_{n-1}(t_j) = f_j := f(t_j)$, $j = 1, \ldots, n$

- Recursively define

$$\begin{aligned} p_n(t) &= p_{n-1}(t) + C(t-t_1)(t-t_2)\cdots(t-t_n) \\ &= p_{n-1}(t) + C(t-t_1)q_n(t) \end{aligned}$$

  such that $p_n(t_{n+1}) = f_{n+1}$

- Set

$$C = \frac{f_{n+1} - p_{n-1}(t_{n+1})}{q_n(t_{n+1})}$$

- Newton bases are interesting because they are *adaptive*—you can choose/update the points as you increase the order of approximation.

- However, they *do* depend on the f_js, which means you have to recompute everything if $f(t)$ changes, which is not the case for the Lagrange approach.

Unstable and Stable Interpolating Basis Sets

- Examples of *unstable bases* are
  - monomials (modal): $\phi_j = x^{j-1}$
  - high-order Lagrange (nodal) bases on *uniformly-spaced* points

- Examples of *stable bases* are
  - Orthogonal polynomials (modal) such as Chebyshev or Legendre polynomials
  - Lagrange (nodal) polynomials on Gauss quadrature points (e.g., Gauss-Legendre, Gauss-Chebyshev, and Gauss-Lobatto-Legendre), all of which are *clustered* near the interval endpoints

- Can map back and forth between stable nodal/modal bases with *minimal information loss*

Orthogonal Polynomials

- We have seen that the condition number of the Vandermonde matrix associated with the *monomial basis* grows exponentially with n.

- We asserted that the growth was due to near linear dependency of the monomials

- It would make therefore make sense to consider *orthogonal polynomials*

- The question is, *in what sense should they be orthogonal?*

Orthogonal Polynomials, continued

- To define orthogonality for functions, we follow the same approach as for vector norms in which we define an inner product

- It is common to use a weighted L^2 inner-product of the form

$$(f, g)_w := \int_{-1}^{1} w(x)\, f(x)\, g(x)\, dx$$

  and associated norm, $\|f\|_w := \sqrt{(f, f)_w}$

- Other domain choices are possible, but $[-1, 1]$ is the standard for Legendre and Chebyshev polynomials

- If the domain of interest is $x \in [a, b]$, one can use the transformation $z = -1 + 2(x - a)/(b - a)$ to map the problem to $\Omega := [-1, 1]$.

- In the inner product, $w(x) \geq 0$ is a nonnegative *weight function* on Ω

- For Legendre polynomials, $w(x) = 1$, and for first-kind Chebyshev polynomials, $w(x) = (1 - x^2)^{-\frac{1}{2}}$

Orthogonal Polynomials, continued

- Table 7.1 from the text gives parameters for several common orthogonal polynomials

Name	**Symbol**	**Interval**	**Weight function**
Legendre	P_k	$[-1, 1]$	1
Chebyshev (1st kind)	T_k	$[-1, 1]$	$(1-t^2)^{-1/2}$
Chebyshev (2nd kind)	U_k	$[-1, 1]$	$(1-t^2)^{1/2}$
Jacobi	J_k	$[-1, 1]$	$(1-t)^\alpha(1+t)^\beta,\ \alpha, \beta > -1$
Laguerre	L_k	$[0, \infty)$	e^{-t}
Hermite	H_k	$(-\infty, \infty)$	e^{-t^2}

- Chebyshev (1st kind) and Legendre are most widely used for numerical interpolation, differentiation, and integration, and as bases for solving partial differential equations.

Legendre Polynomials

- Legendre polynomials are defined by the orthogonality condition

$$(P_i, P_j) \;=\; \int_{-1}^{1} P_i(x) P_j(x)\, dx \;=\; \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases},$$

  with $P_i(1) = 1$. (An alternative scaling is to require $\|P_i\| = 1$, in which case the polynomials are *orthonormal*.)

- Using Gram-Schmidt orthogonalization, it's easy to show that all orthogonal polynomials satisfy a *three-term recurrence* of the form

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

- If $w(x)$ is symmetric, then β_k=0

- These recurrence formulas provide an *efficient* and *stable* way to evaluate orthogonal polynomials, even for large values of k

Legendre Polynomials, continued

- Starting with $\qquad P_0 = 1, \quad P_1 = x,$

  the recurrence formula for the kth-order Legendre polynomial is

$$P_{k+1}(x) \;=\; \frac{2k+1}{k+1}\, t\, P_k(t) \;-\; \frac{k}{k+1}\, P_{k-1}(t)$$

- The first few Legendre polynomials are

$$\begin{aligned}
P_0 &= 1 \\
P_1 &= x \\
P_2 &= \frac{1}{2}\left(3x^2 - 1\right) \\
P_3 &= \frac{1}{2}\left(5x^3 - 3x\right) \\
P_4 &= \frac{1}{8}\left(35x^4 - 30x^2 + 3\right) \\
P_5 &= \frac{1}{8}\left(63x^5 - 70x^3 + 15x\right)
\end{aligned}$$

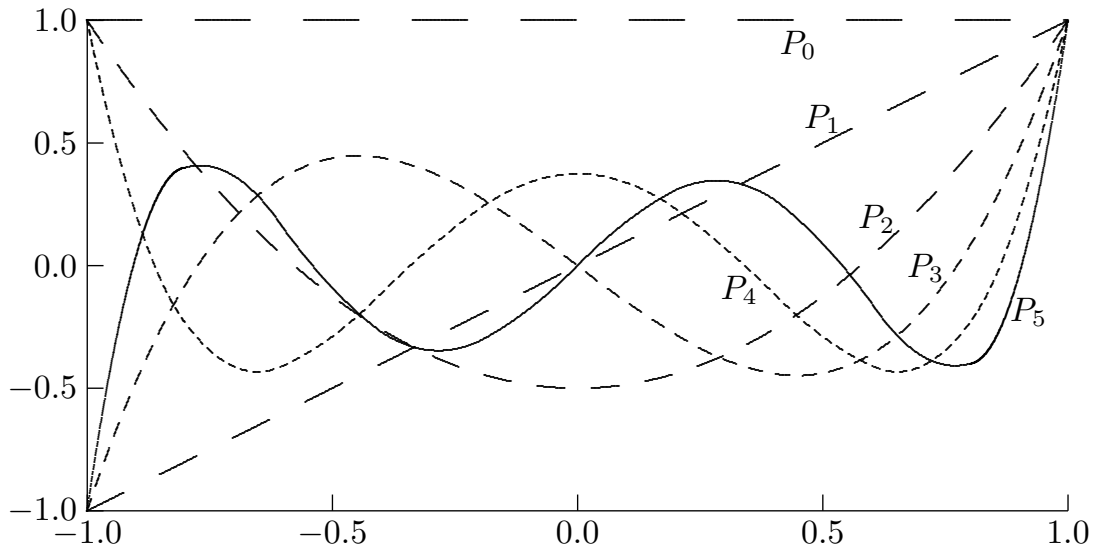


Figure 7.4: First six Legendre polynomials.

Chebyshev Polynomials

- Starting with $T_0 = 1, \quad T_1 = x,$

  recurrence for the kth-order (1st-kind) Chebyshev polynomial is

$$T_{k+1}(x) \;=\; 2xT_k(x) \;-\; T_{k-1}(x)$$

- The first few Chebyshev polynomials are

$$\begin{aligned} T_0 &= 1 \\ T_1 &= x \\ T_2 &= 2x^2 - 1 \\ T_3 &= 4x^3 - 3x \\ T_4 &= 8x^4 - 8x^2 + 1 \\ T_5 &= 16x^5 - 20x^3 + 5x \end{aligned}$$

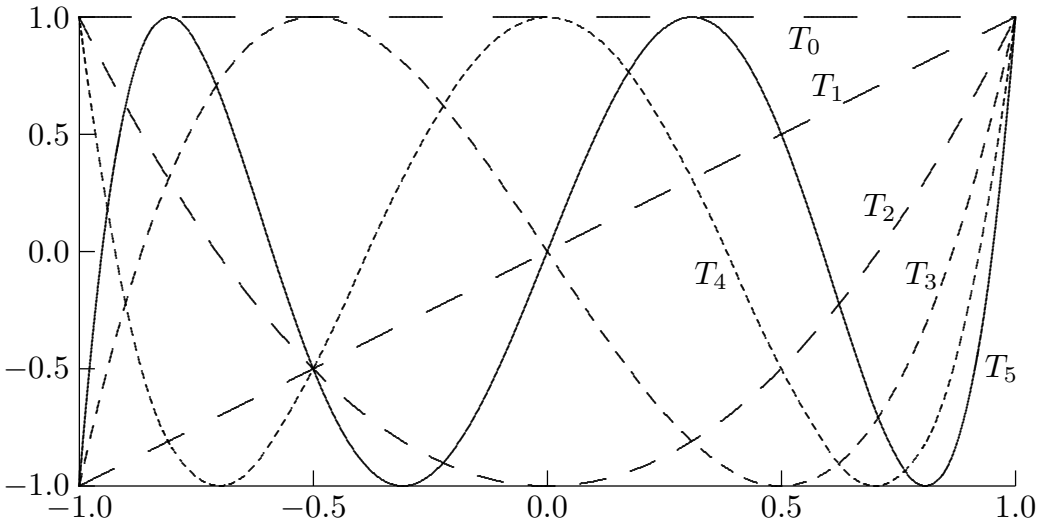


Figure 7.5: First six Chebyshev polynomials.

Interpolating with Orthogonal Polynomials

- Interpolation or weighted least-squares (i.e., *projection*) can be easily implemented with orthogonal polynomials

- Simply set up the Vandermonde matrix and solve for the coefficients

- Here is a 3×3 case with Chebyshev polynomials

$$\mathbf{Ac} = \begin{bmatrix} 1 & T_1(x_1) & T_2(x_1) \\ 1 & T_1(x_2) & T_2(x_2) \\ 1 & T_1(x_3) & T_2(x_3) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

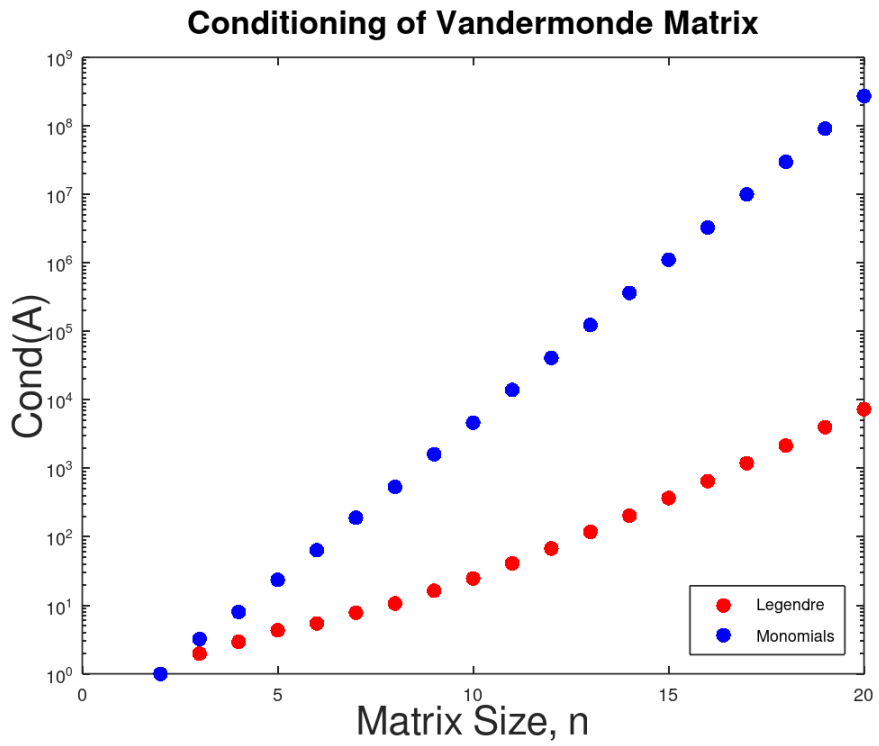
- One then evaluates

$$p_{n-1}(x) = \sum_{j=1} c_j T_j(x)$$

Interpolating with Orthogonal Polynomials

- Here is an example comparing the conditioning of the Vandermonde matrices for the Legendre basis vs. a monomial basis.

```
%% Generalized Van der Monde example, Legendre v. Monomial
hdr;

for kpass=1:2;

   k=0;
   for n=2:20; k=k+1;

      N=n-1;           % Polynomial order is N = n-1

      [z,w]  =zwuni(N);

      if kpass==1;
         A=legendp(z,N);  % Evaluate [P0, P1, ... , PN] at [z1,z2,...,zn]
         clr='r.';
      else;
         A=monop  (z,N);  % Evaluate [P0, P1, ... , PN] at [z1,z2,...,zn]
         clr='b.';
      end;

      cn(k) = cond(A); nn(k) = n;

   end

   semilogy(nn,cn,clr,ms,14);

   xlabel('Matrix Size, n',fs,22)
   ylabel('Cond(A)',fs,22)
   title('Conditioning of Vandermonde Matrix',fs,18)
   pause(2); hold on;
end
legend('Legendre','Monomials','location','southeast')
```



- We see that the Legendre basis *much* better conditioned than the monomials, as expected

Polynomial Interpolation Error

- Without going into all the details, let's see if we can make an educated guess about the form of the polynomial interpolation error, $f(x) - p(x)$, with $p(x) \in \mathbb{P}_{n-1}$ being the interpolant

- Because of interpolatory condition, the error at the nodes vanishes,

$$f(x_i) - p(x_i) = 0, \quad i = 1, \ldots, n$$

- It is reasonable therefore to expect an error of the form

$$f - p = C(x)(x - x_1)(x - x_2) \cdots (x - x_n)$$

- The *projective property*, $f(x) - p(x) \equiv 0$ for all $f \in \mathbb{P}_{n-1}$, suggests that

$$C(x) \;=\; \gamma f^{(n)}(\xi)$$

  where γ is a constant and $\xi = \xi(x)$ depends not only on x but also on x_i

- Dimensional analysis indicates that the constant γ is dimensionless since the units of the product $f^{(n)} q_n(x)$ match those of f itself

Interpolation Error, continued

- To determine γ, consider the case $f(x) = x^n \notin \mathbb{P}_{n-1}$, for which $f^{(n)} = n!$

- In this case, the error is $x^n - p_{n-1}(x)$, which is a *monic* polynomial of degree n (leading coefficient $= 1$) that vanishes at all the nodes, $x_1, \ldots, x_n$.

- Consequently, $x^n - p_{n-1}(x) \equiv q_n(x)$, which is also a monic polynomial of degree n.

- We can see this by noting that their difference is $\in \mathbb{P}_{n-1}$, yet has n zeros and therefore they must be the same

- In this case, $f(x) - p_{n-1}(x) = \gamma f^{(n)}(\xi) q_n(x) = \gamma\, n!\, q_n(x) = q_n(x)$, which indicates that $\gamma = \frac{1}{n!}$.

Interpolation Error, continued

- Therefore, we can expect that form of the error is

$$f - p = \frac{f^{(n)}(\xi)}{n!} q(x),$$

with

$$q(x) := (x - x_1)(x - x_2) \cdots (x - x_n)$$

- The precise location of $\xi(x)$ is in general not known as it depends on f, x, and the x_is.

- However (via repeated application of Rolle's Thm), it can be shown to be in the interval containing $[x_1, x_2, \ldots, x_n, x]$.

- The error formula thus applies even for *extrapolation*: $x \notin$ interval containing $[x_1, x_2, \ldots, x_n]$.

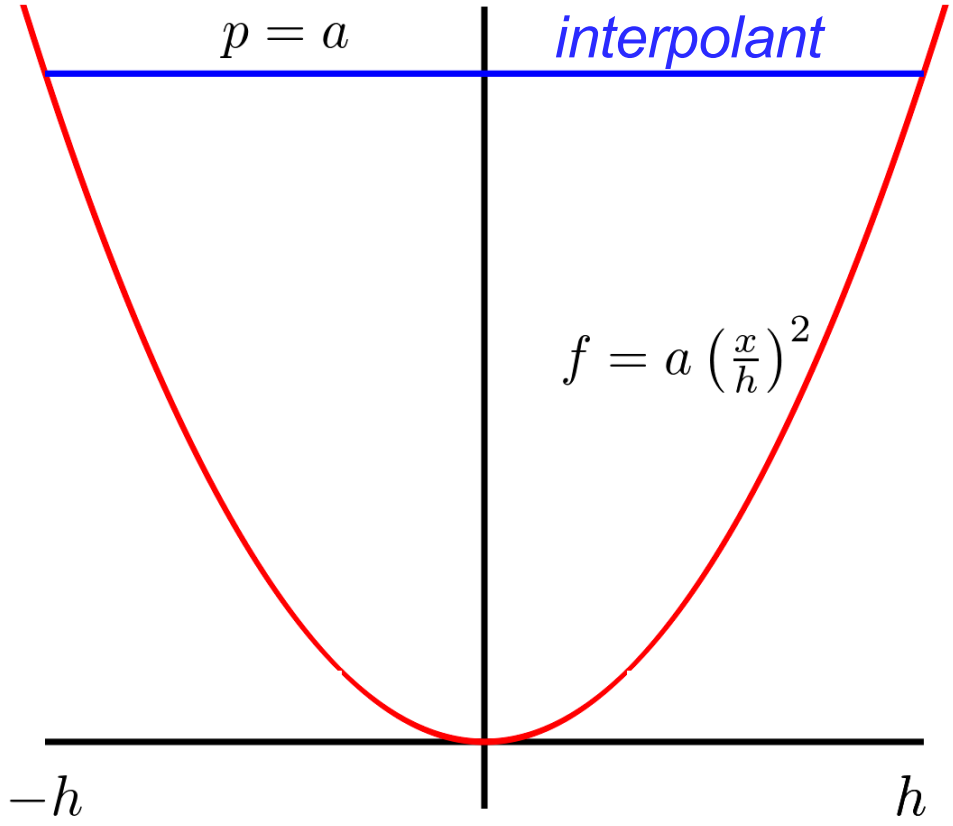
Example

- Consider $f(x) = a\left(\frac{x}{h}\right)^2$, with $x_1 = -h$ and $x_2 = h$.

- Here,

$$n = 2 \longrightarrow p(x) \in \mathbb{P}_{n-1},$$

so $p(x)$ is *linear* and, in this particular case, is a constant, $p(x) = a$

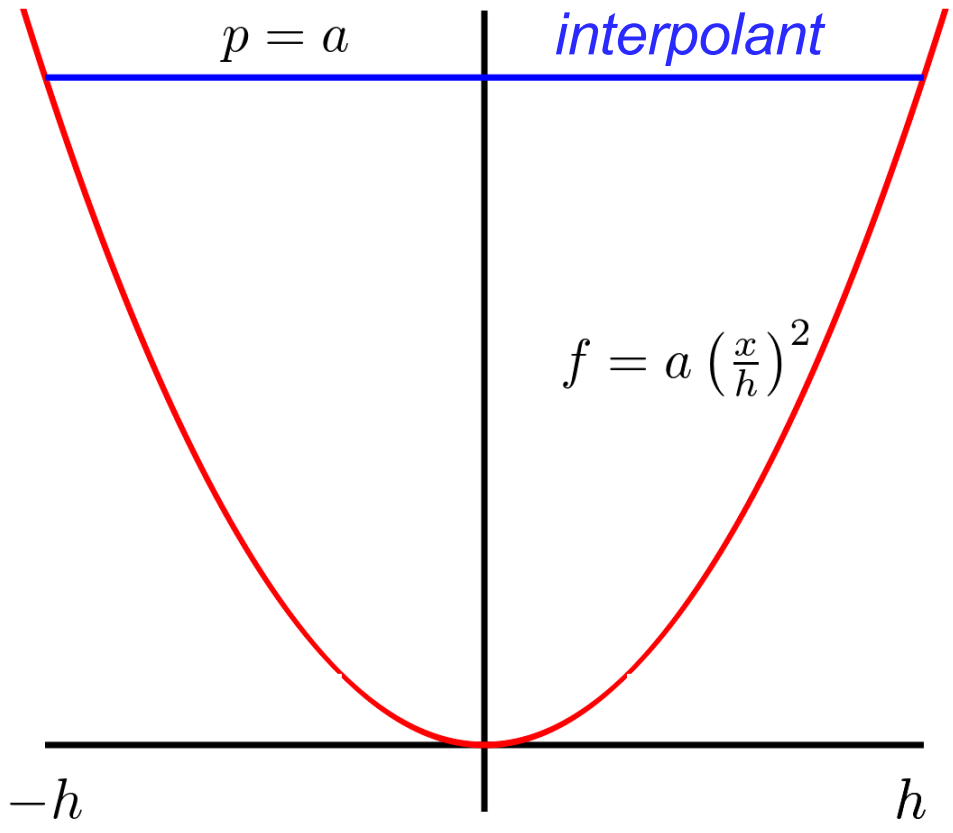


- The error is

$$f - p = a\left[\left(\frac{x}{h}\right)^2 - 1\right] = \frac{a}{h^2}(x+h)(x-h)$$

$$= \frac{a}{h^2}(x-x_1)(x-x_2)$$

- Since $f'' \equiv 2!\frac{a}{h^2}$, the multiplier in front of $(x-x_1)(x-x_2)$ is $f''/2!$



A Crude Error Bound

- Assume $t_1 < t_2 < \cdots < t_n$ and define $\Delta t_i := t_{i+1} - t_i$ for $i = 1, \ldots, n-1$

- If $h := \max_i \Delta t_i$ and $|f^{(n)}| \leq M$ for all $t \in [t_0, t_n]$ then

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

- For n *fixed*, error diminishes as h^n

- For h *fixed*, the error diminishes with increasing n only if $|f(t)|$ does not grow too rapidly with n

- It is important to note that a sharper estimate is possible in the case of the t_is being the Chebyshev nodes.

Controlling the Error

- 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 $\exists\ \xi \in [t_1, t_2, \ldots, t_n, t]$ such that

$$f(t) - p(t) \ = \ \frac{f^n(\xi)}{n!}(t - t_1)(t - t_2) \cdots (t - t_n)$$

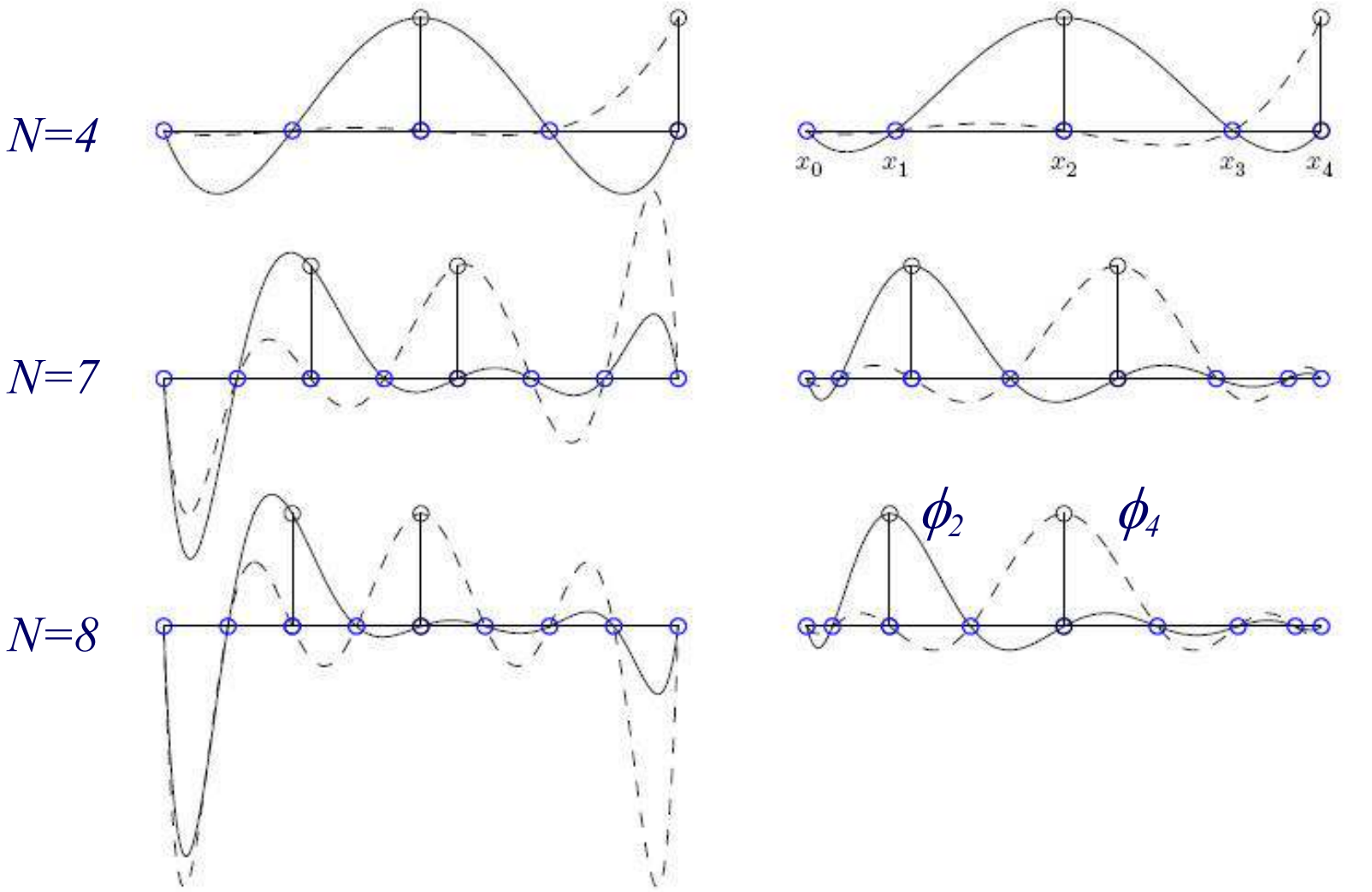
- We generally have no control over $f^n(\xi)$, so instead seek to optimize choice of the t_j in order to minimize

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

- 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 discuss shortly

- First, however, we look at some examples of the basis functions

Lagrange Polynomials: Good and Bad Point Distributions



Optimizing the Choice of Nodes

- The switch from monomial bases to orthogonal polynomials significantly improves the conditioning of the Vandermonde matrix

- In exact arithmetic, however, both will give the *same* result

- We can significantly reduce the interpolation error by abandoning uniformly-distributed nodes in favor of *optimized* nodes

- The idea is to shift the nodes *toward* the endpoints where $q_n(x)$ is large in the case of uniform points

- Another way of posing the problem is: *For n points on $[-1, 1]$, what is the set that will minimize*

$$\max_{x \in [-1,1]} q_n(x) = \max_{x \in [-1,1]} (x - x_1)(x - x_2) \cdots (x - x_n) \, ?$$

- We will see that the optimal set corresponds to the n roots of $T_n(x)$ and that the corresponding maximum will be $1/2^{n-1}$ for $n > 0$.

Optimizing the Choice of Nodes, continued

- For $x \in [-1, 1]$, we can define $x = \cos\theta \iff \theta = \cos^{-1} x$
- Then the trigonometric identity, $\cos(k+1)\theta = 2\cos\theta\ \cos k\theta\ -\ \cos(k-1)\theta$ leads to

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

- Thus, on [-1,1], $T_n(x)$ is a cosine function with maximal absolute extrema value of 1.

- If we define the monic polynomial (for $n > 0$),

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

  then its roots correspond to the roots of $\cos n\theta$, i.e.,

$$\theta_j \;=\; \frac{\pi}{n}\left(j - \frac{1}{2}\right),\; j = 1, \ldots, n$$

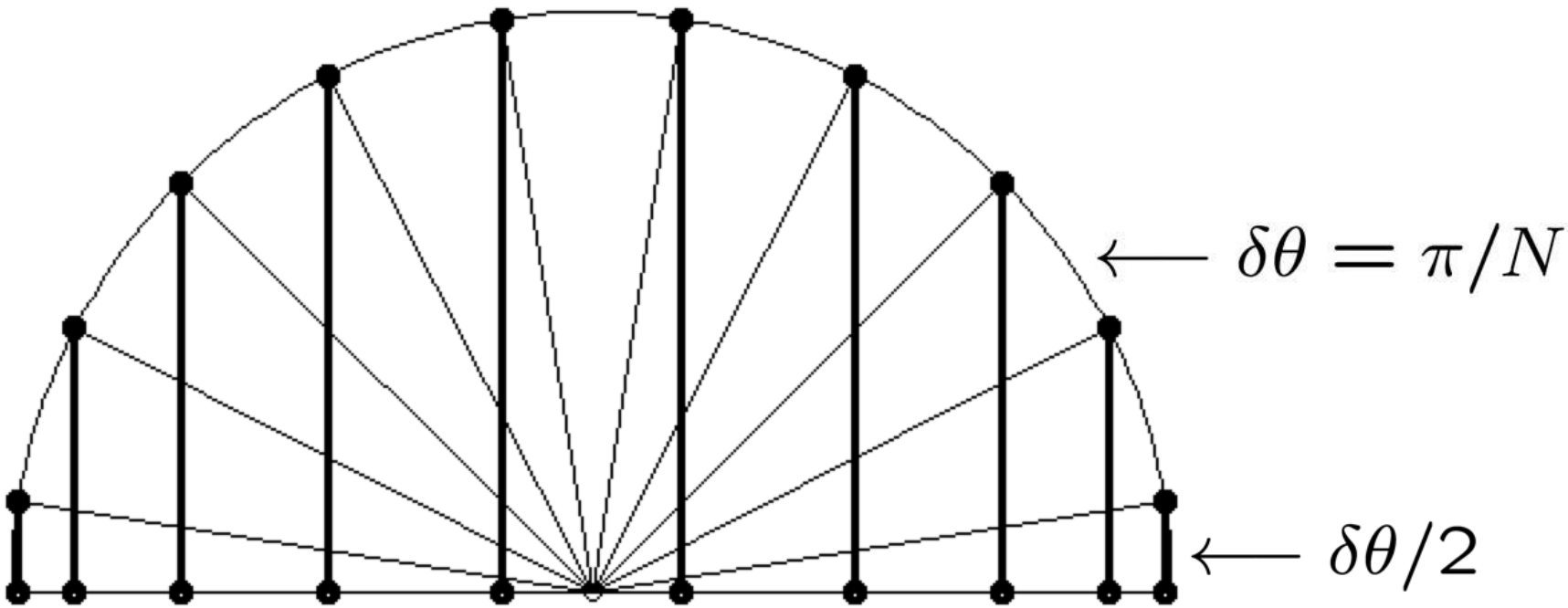
  and we have $x_j = \cos^{-1}\theta_j$

- Notice that

$$\max_{x \in [-1,1]} \hat{T}_n(x) \;=\; \frac{1}{2^{n-1}}$$

Nth-order Chebyshev points

- For $t \in [-1, 1]$, Chebyshev polynomial of degree N is $T_N(t) = \cos(N\theta)$, with $\theta := \cos^{-1}(t)$.
- Zeros occur when $\theta_j = \pi\left(j - \frac{1}{2}\right)$, $j = 1, \ldots, N$

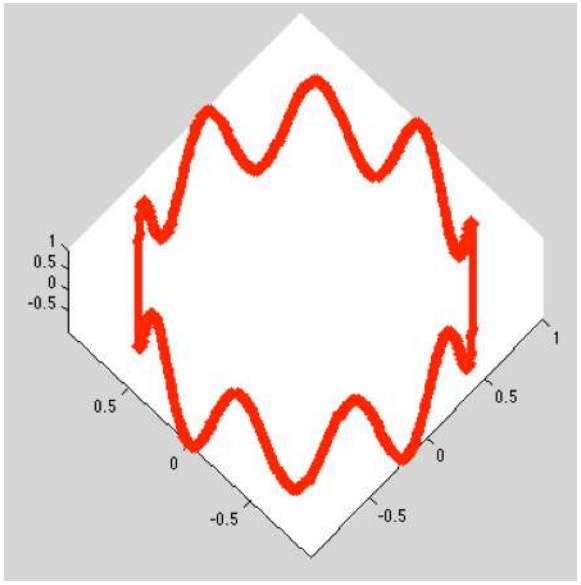


- Distribution is "clustered" near the endpoints of the interval, which controls the wild oscillations seen with uniformly-distributed nodes

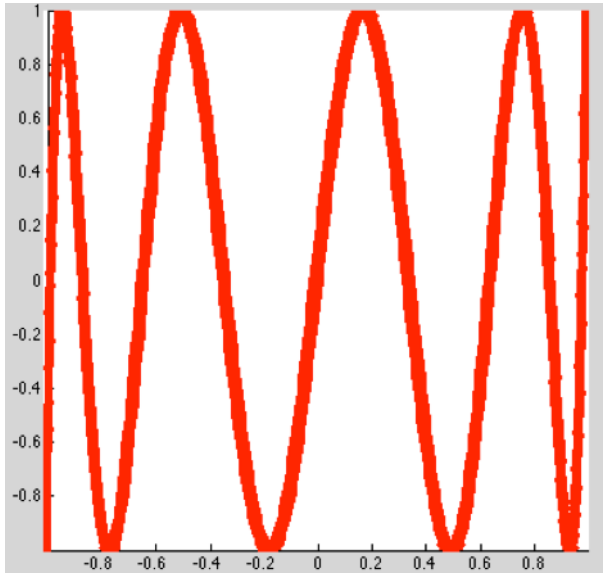
Nth-Order Gauss Chebyshev Points

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
```



$\cos(N\theta)$

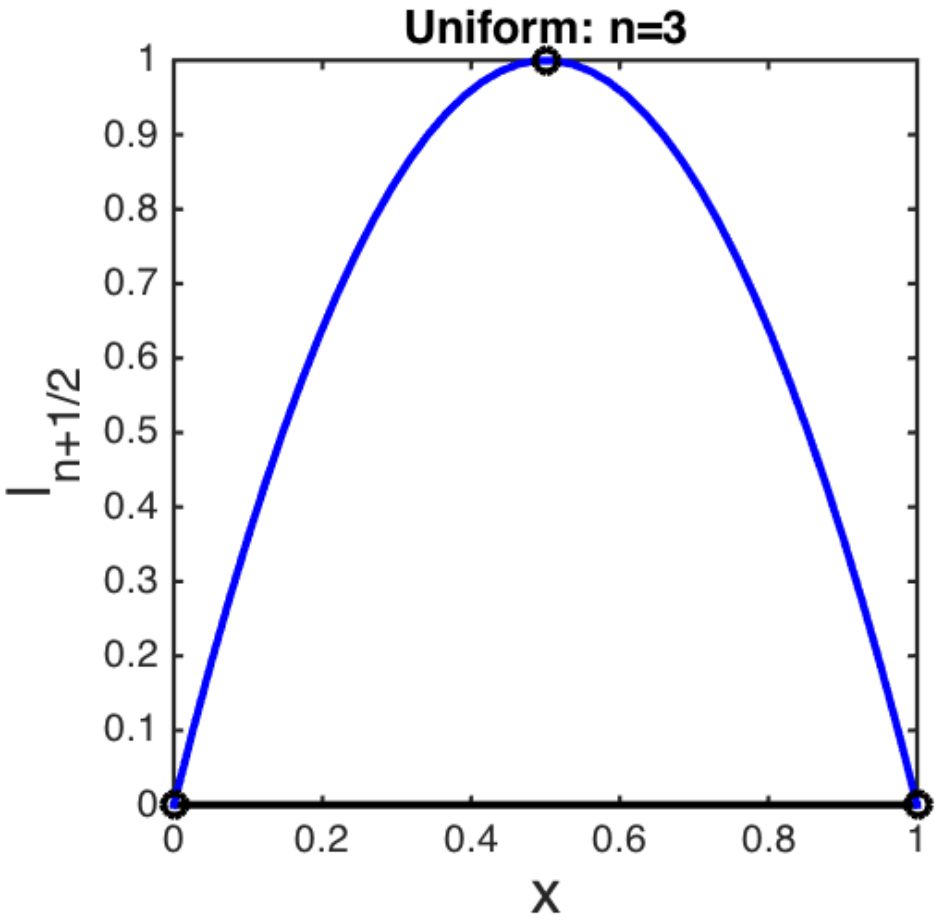


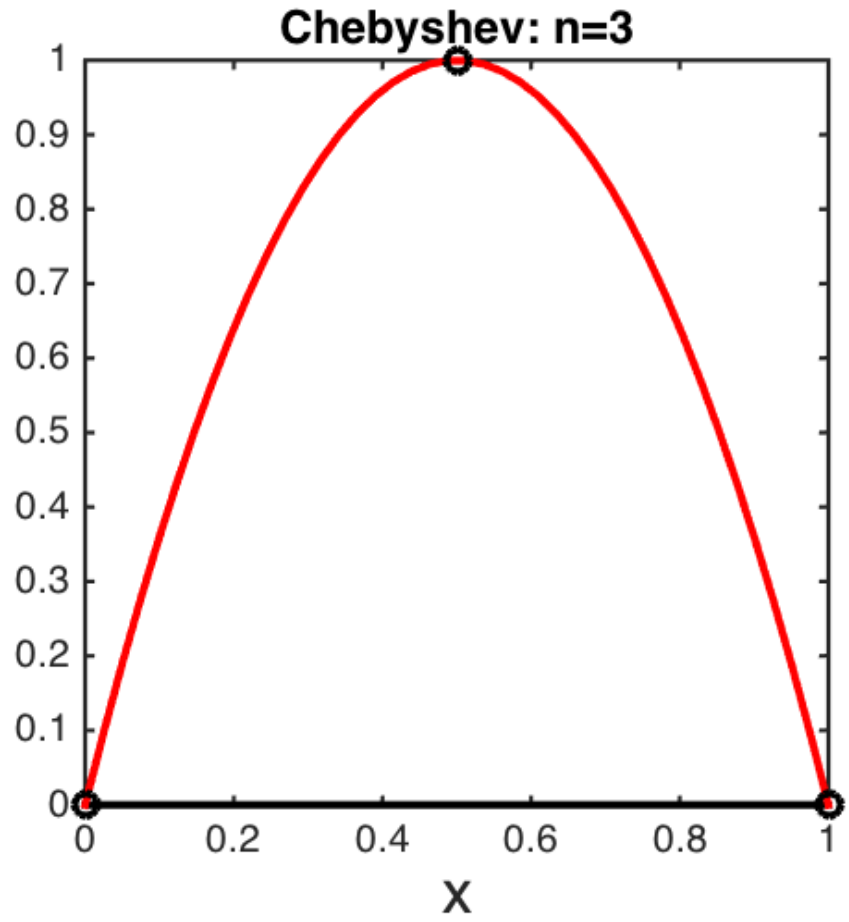
$T_N(x)$

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

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

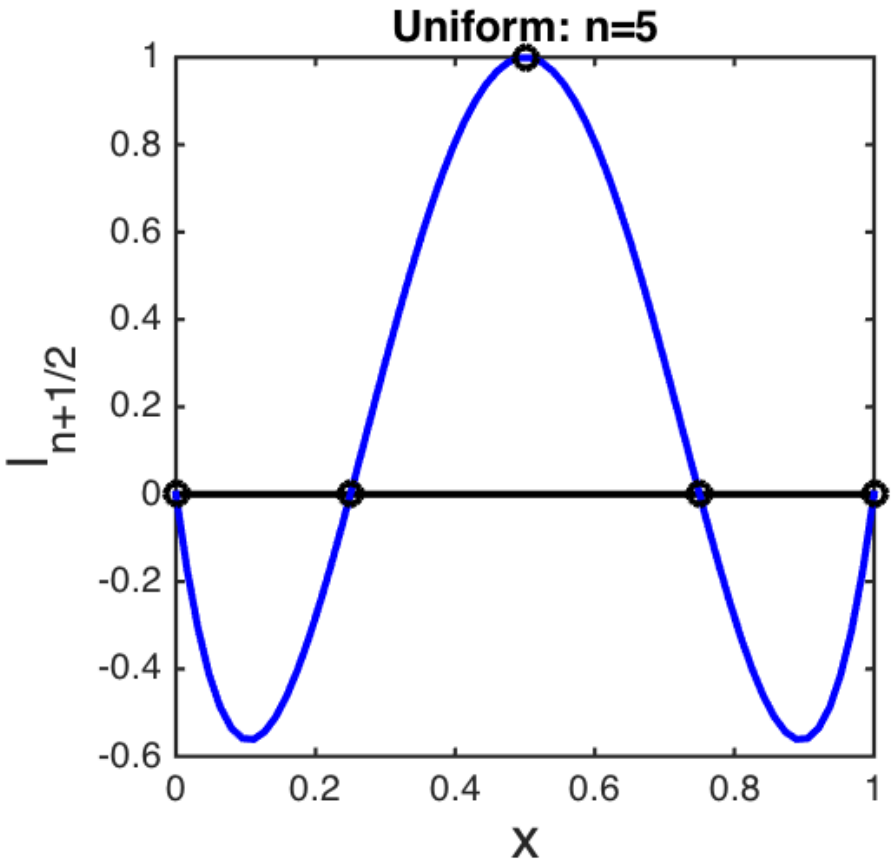
Behavior of Cardinal Functions: Uniform vs. Chebyshev

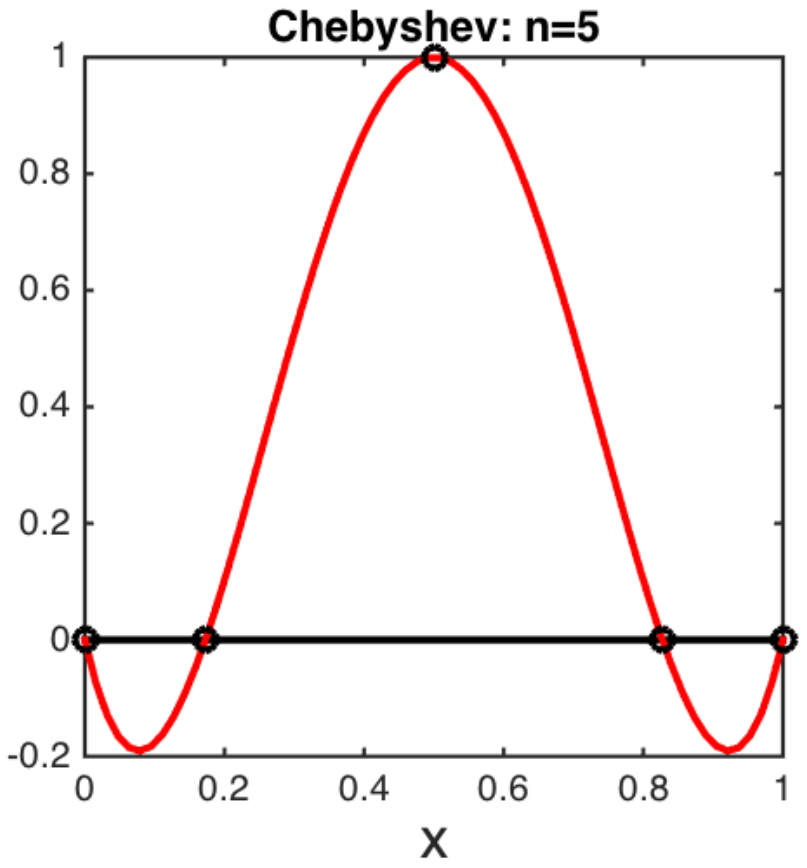




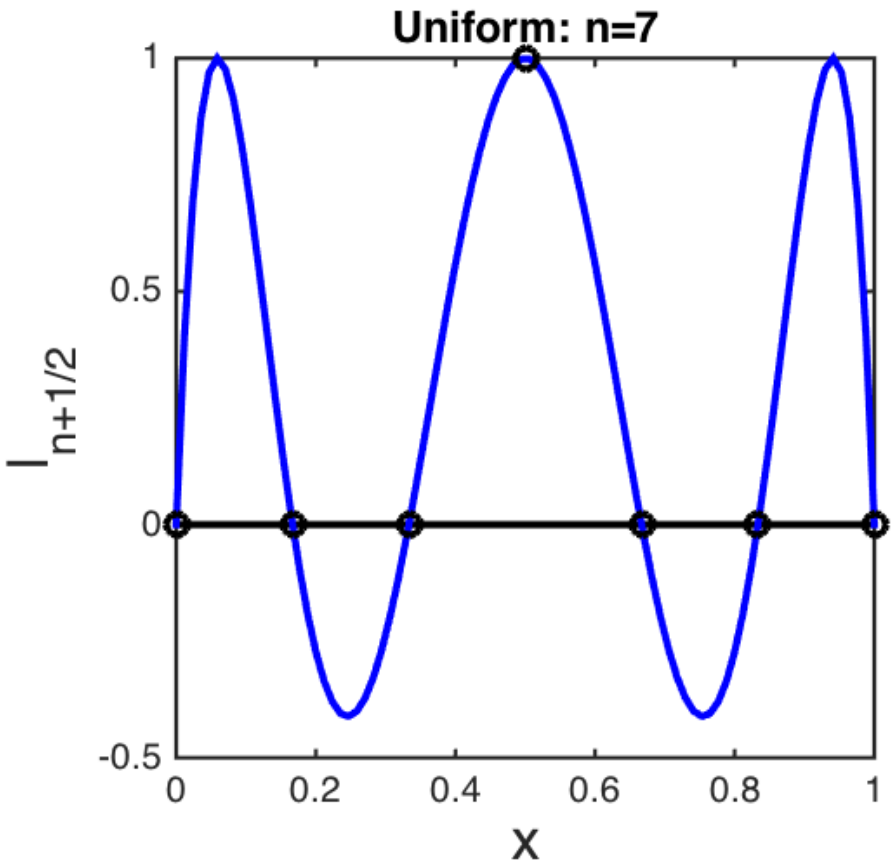
Matlab Demo: lag_subplot.m

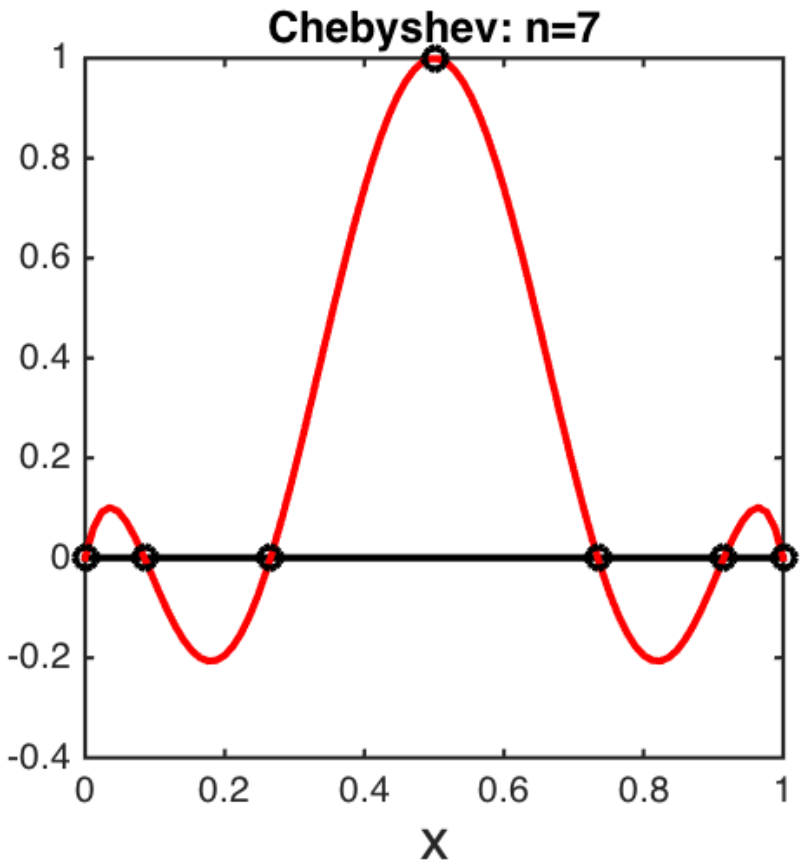
Behavior of Cardinal Functions: Uniform vs. Chebyshev



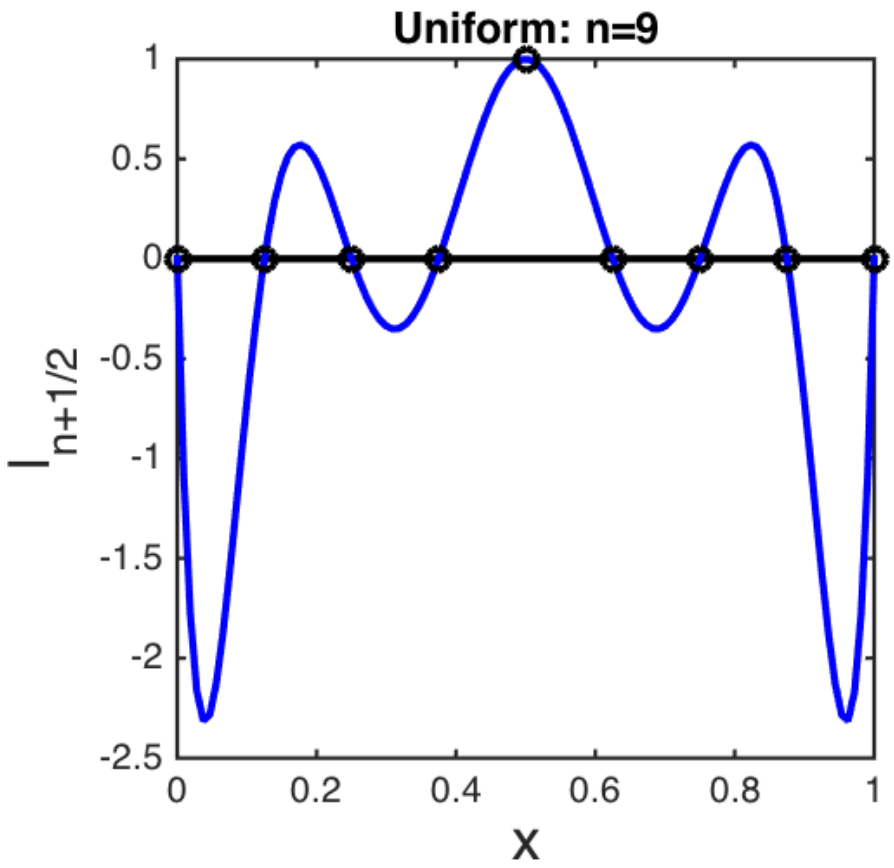


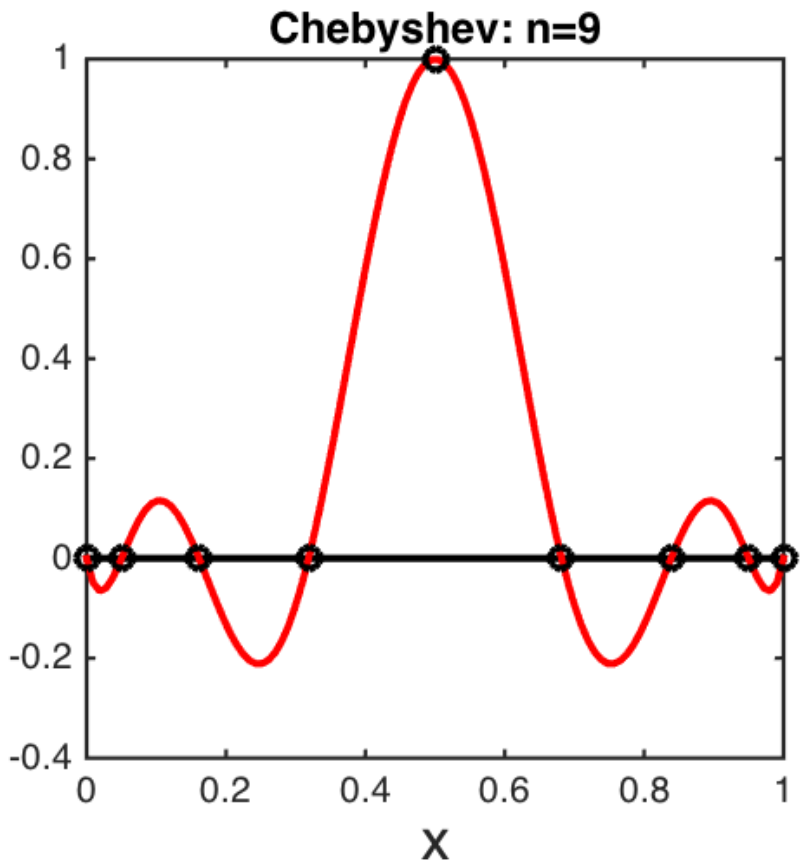
Behavior of Cardinal Functions: Uniform vs. Chebyshev



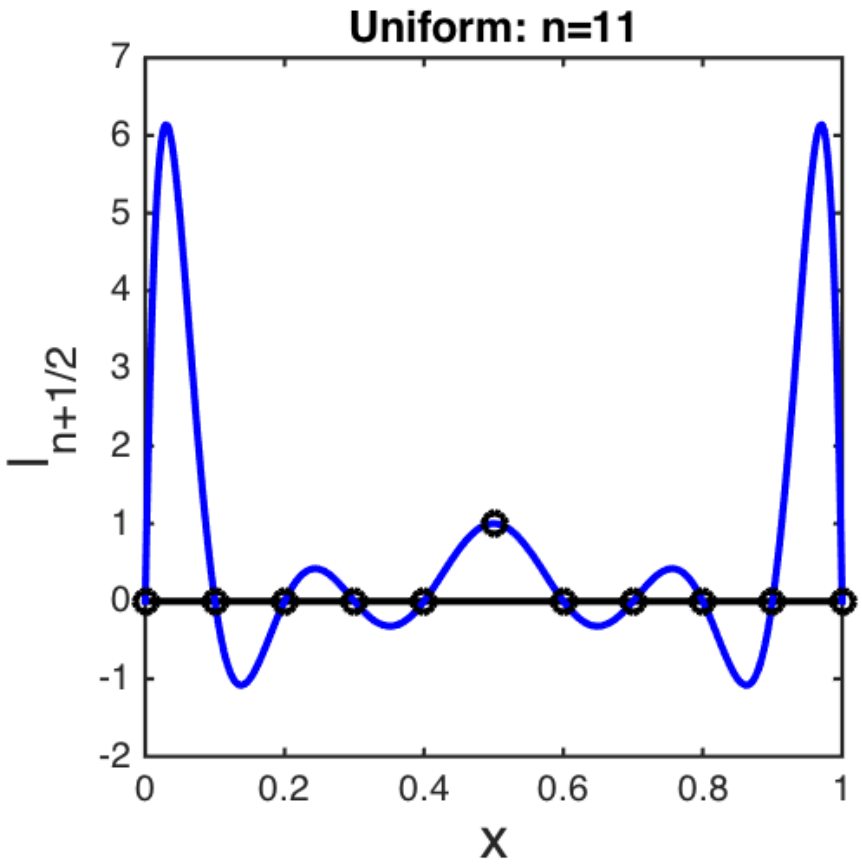


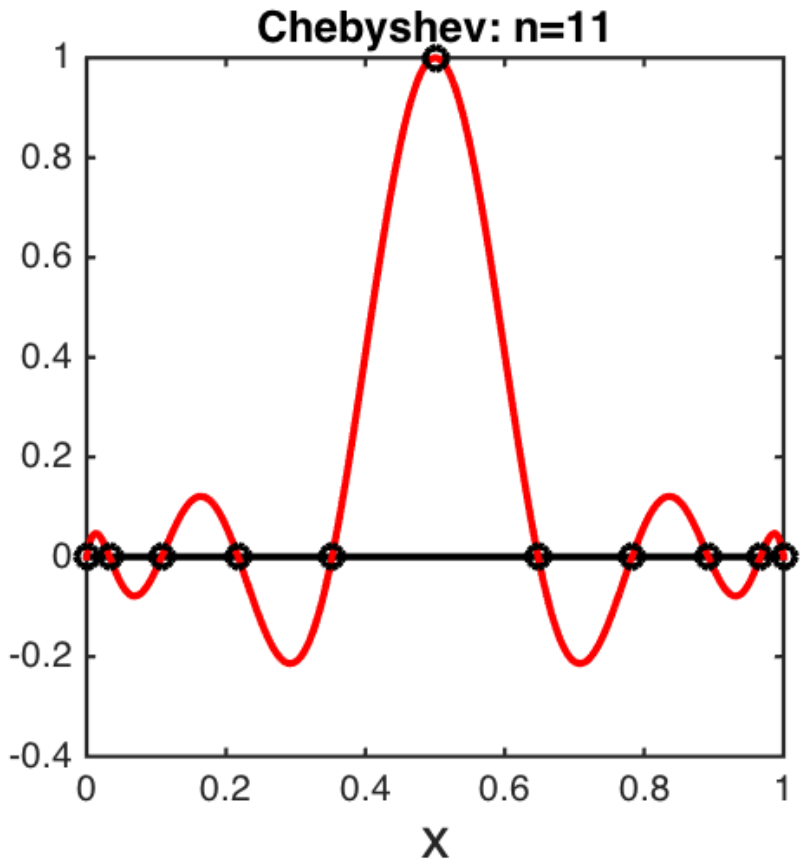
Behavior of Cardinal Functions: Uniform vs. Chebyshev



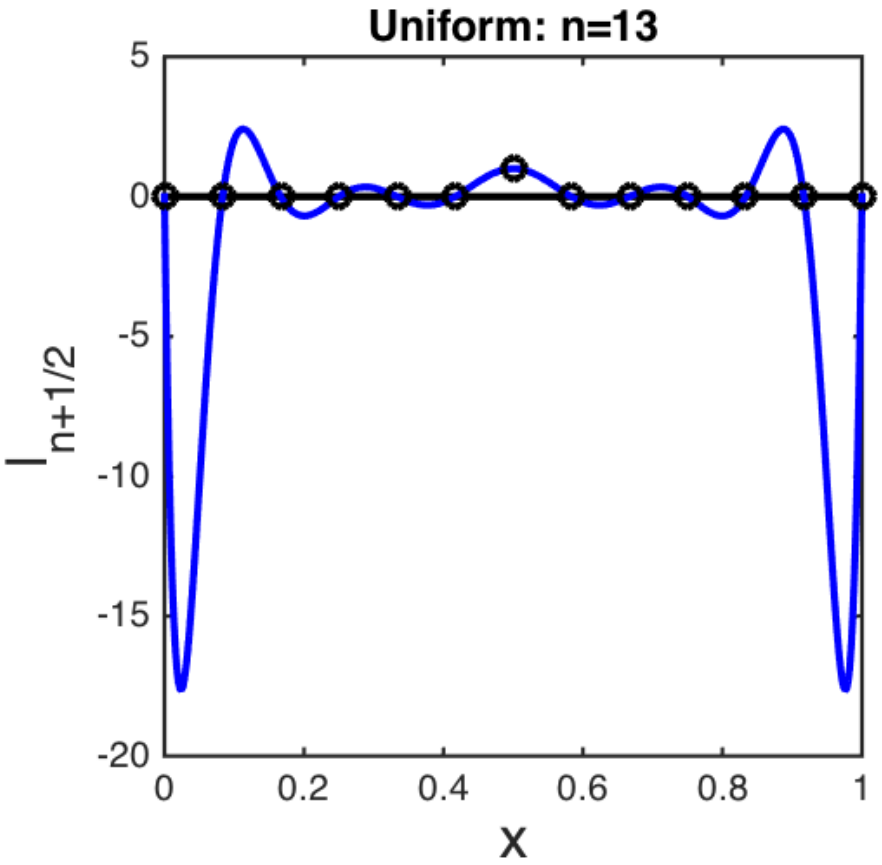


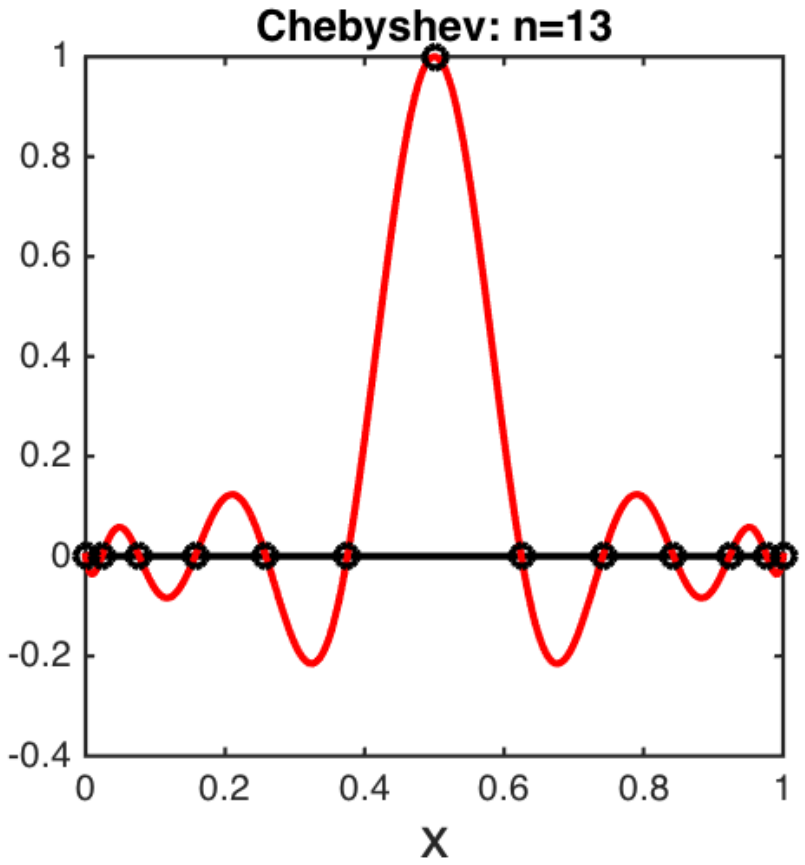
Behavior of Cardinal Functions: Uniform vs. Chebyshev



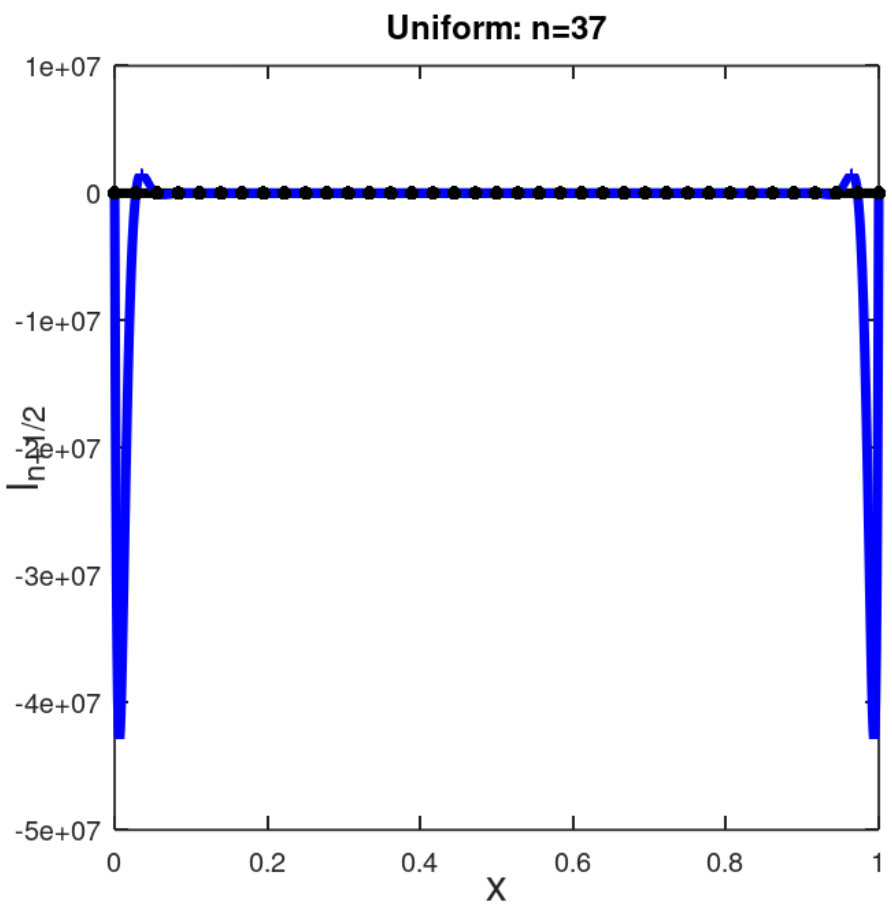


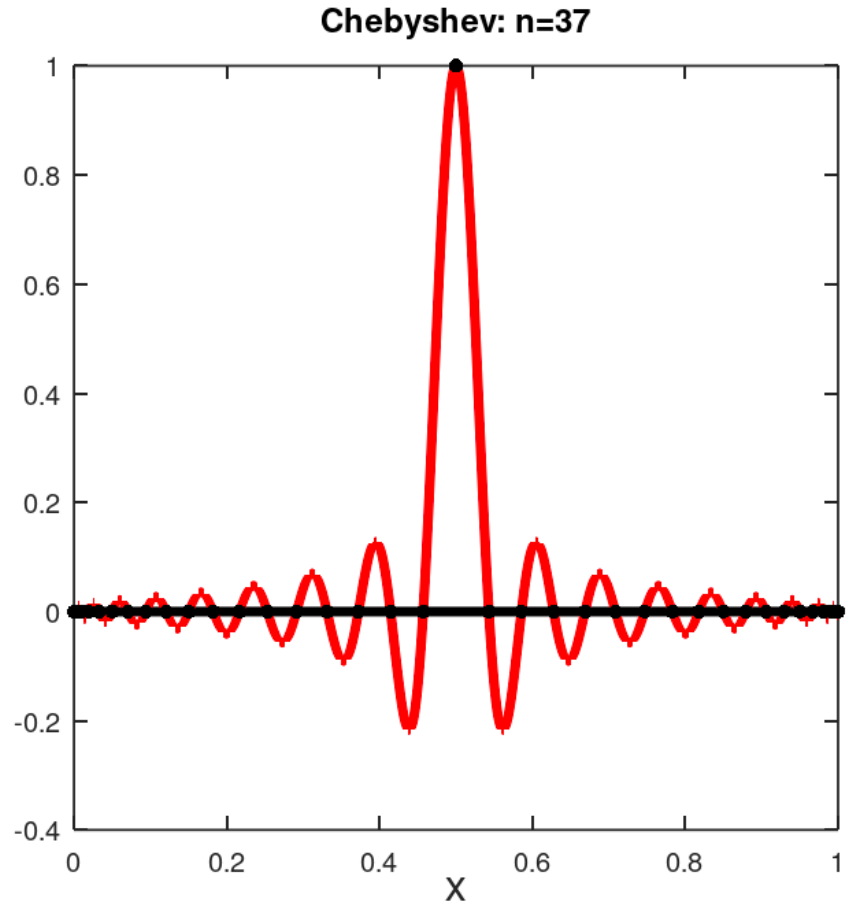
Behavior of Cardinal Functions: Uniform vs. Chebyshev





Behavior of Cardinal Functions: Uniform vs. Chebyshev





Q: What happens to the derivative of $l_i(x)$ in each of these cases? What about the integral?

Impact of Optimal Node Set vs Uniform

- From the preceding slide, we can see that $\max |l_j(x)|$ grows exponentially with n for uniform points.

- Suppose $f(x)$ is analytic (e.g., $\sin x$), for which we have convergence even with uniform points

  - At the nodes, the *floating point representation* will be of the form

$$\hat{f}(x_j) \;=\; f_j \;+\; \epsilon_j$$

  - Suppose the interpolant is computed without further error.
  - The interpolant will be of the form

$$\hat{p}(x) \;=\; \sum f_j l_j(x) \;+\; \sum_j \epsilon_j l_j(x)$$

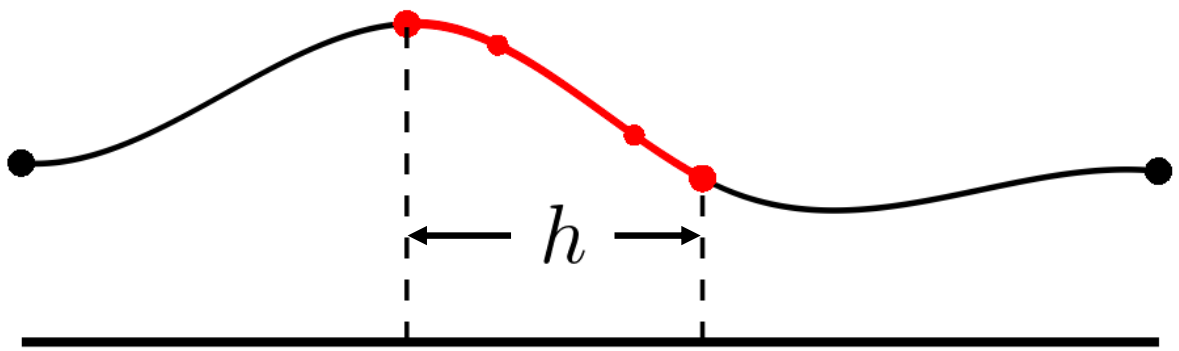
    - The first term on the right will converge, but the second term, with random coefficients of order $|\epsilon_j| \leq \epsilon_M \|f\|$ will lead to variations that scale with $\epsilon_M \|f\| \max |l_j(x)|$, which can be large

Interp_sine.m

STOPPED HERE

Composite Interpolation

- An alternative to high-order polynomial interpolation is to use *piecewise polynomials*

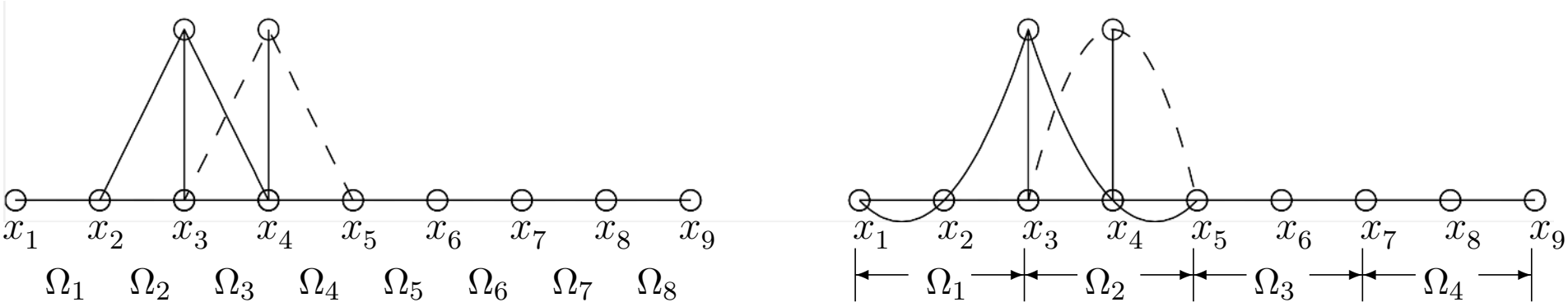


- Scenarios where this approach might be useful:
  - cases where the function is highly oscillatory in some regions and relatively smooth elsewhere
  - situations in which data is available only at *prescribed* points $(x_i,\ f_i)$

- For the latter case, *splines* are of particular interest because they match the data (i.e., are interpolatory) *and* they can be very smooth

- From a practical standpoint, piecewise polynomials of degree $k > 1$ are of interest because they typically converge as $O(h^{k+1})$ as the subdomain width h is reduced.

Piecewise Polynomial Bases: Linear and Quadratic

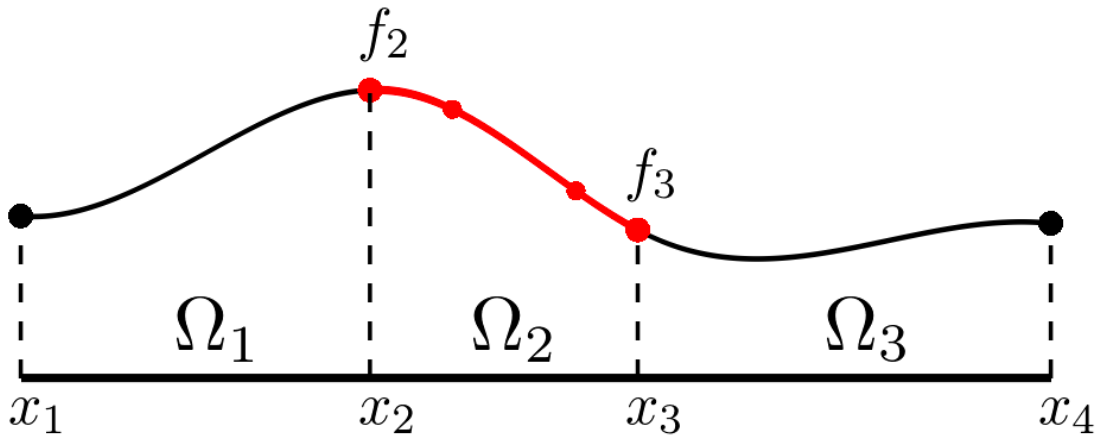


Examples of one-dimensional piecewise linear (left) and piecewise quadratic (right) Lagrangian basis functions, $l_2(x)$ and $l_3(x)$, with subdomains Ω_j.

Splines

- Spline is a piecewise polynomial of degree k that is $k-1$ times differentiable
- For example, linear spline is of degree 1 and has 0 continuous derivatives, i.e., it is continuous but not smooth
- *Cubic spline* is piecewise cubic that is twice continously differentiable
- Interpolating data (n points) and requiring two continous derivatives leaves 2 free parameters at the endpoints
- In the example below, we could write cubic on each domain as

$$s(x)|_{\Omega_j} = a_j + b_j x + c_j x^2 + d_j x^3$$



Cubic Splines, continued

- Continuity leads to a system of equations for 4 coefficients on each interval
  - Interval $\Omega_j = [x_j, x_{j+1}],\ j = 1, \ldots, n-1$

    $s_j(x) \in \mathbb{P}_3(x)$ on Ω_j

    $s_j(x) = a_j + b_j x + c_j x^2 + d_j x^3$

  - $4n - 4$ unknowns

    $s_j(x_j) = f_j,\ \ j = 1, \ldots, n-1$

    $s_j(x_{j+1}) = f_{j+1},\ j = 1, \ldots, n-1$

    $s_j'(x_{j+1}) = s_{j+1}'(x_{j+1}),\ j = 1, \ldots, n-2$

    $s_j''(x_{j+1}) = s_{j+1}''(x_{j+1}),\ j = 1, \ldots, n-2$

    Spline Conditions: *enforce continuous 1st & 2nd derivatives*

  - $4n - 2$ equations

  - Two more equations from boundary conditions on s(x)

Cubic Spline Formulation – 2 Segments

8 Unknowns

$$s_1(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2 + \alpha_4 x^3$$

$$s_2(x) = \beta_1 + \beta_2 x + \beta_3 x^2 + \beta_4 x^3$$

8 Equations

Interpolatory

$$s_1(x_1) = f_1$$

$$s_1(x_2) = f_2$$

$$s_2(x_2) = f_2$$

$$s_2(x_3) = f_3$$

Continuity of Derivatives

$$s_1'(x_2) = s_2'(x_2)$$

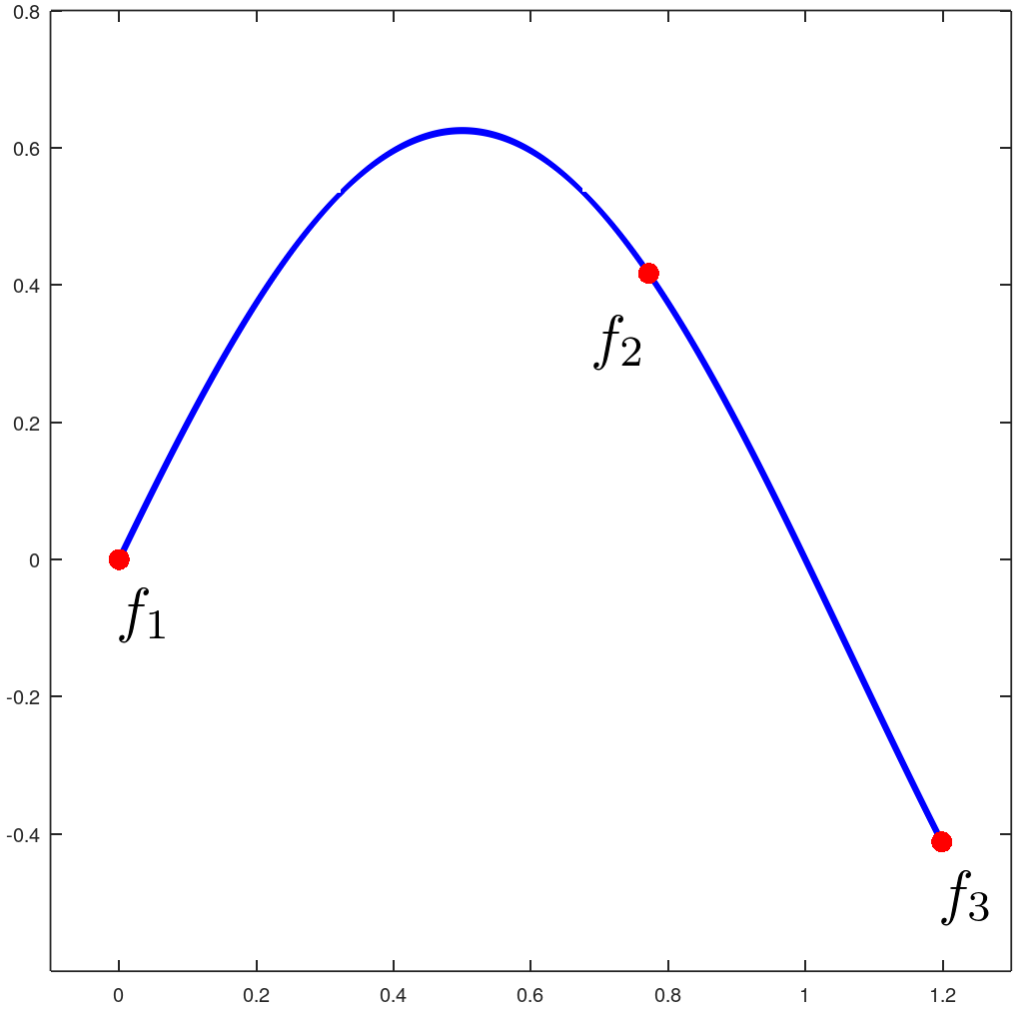
$$s_1''(x_2) = s_2''(x_2)$$

End Conditions

$$s_1''(x_1) = 0$$

$$s_2''(x_3) = 0$$

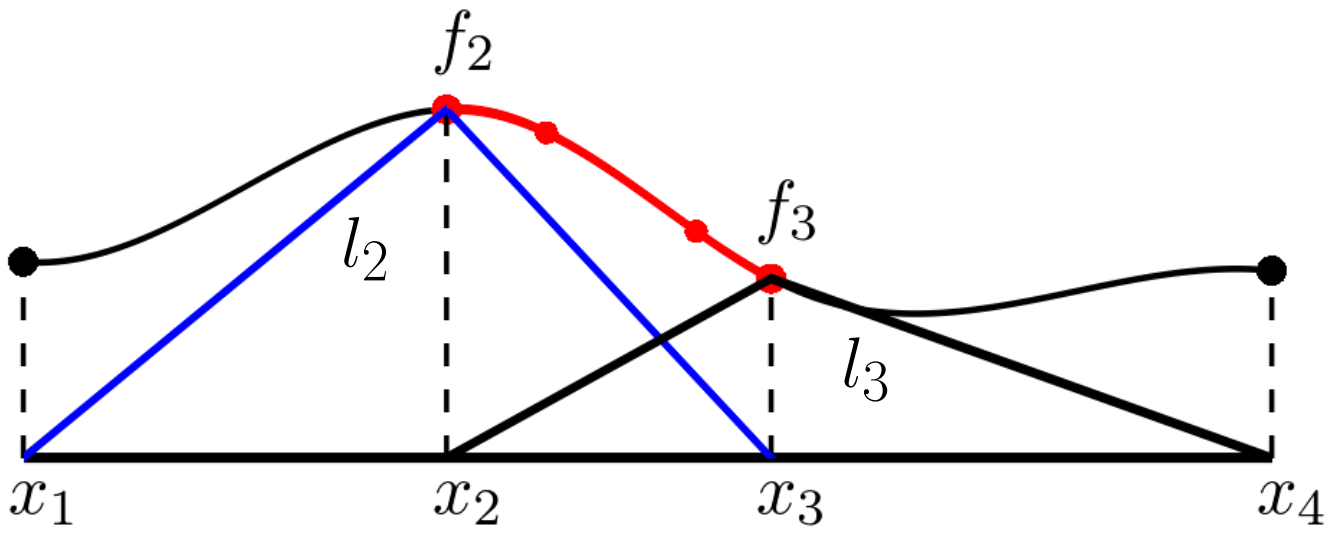
(*Natural Spline*)



Cubic Splines, continued

- It is convenient to write the cubic in terms of the known interpolatory constraints ($s(x_j) = f_j$) and the unknown second derivatives (s''_j and s''_{j+1}) at each endpoint of Ω_j

- This direct approach is easily realized using the 1st-order Lagrange cardinal functions on $l_j(x)$ and $l_{j+1}(x)$, which have overlapping support on Ω_j



- It is clear that we can construct the red cubic on Ω_2 as a linear combination of l_2 and l_3 plus a pair of cubics that vanish at x_2 and x_3

Cubic Splines, continued

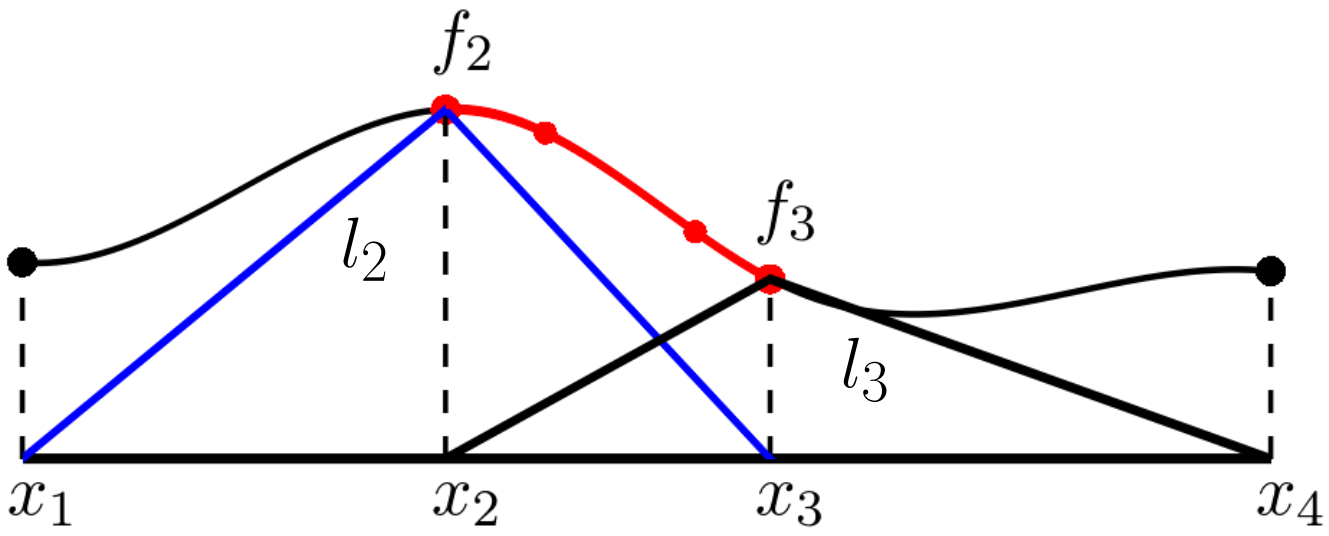
$$s(x)|_{\Omega_j} \;:=\; s_j(x) \;=\; l_j(x)f_j \;+\; l_{j+1}(x)f_{j+1} \;+\; h_j^2[g_j(x)s_j'' + g_{j+1}(x)s_{j+1}'']$$

where

$$g_j(x) \;=\; \frac{1}{6}\left(l_j^3 - l_j\right) \qquad g_{j+1}(x) \;=\; \frac{1}{6}\left(l_{j+1}^3 - l_{j+1}\right)$$

- Here, the l_js are the standard linear Lagrange interpolants on Ω_j that satisfy $l_j(x_i) = \delta_{ij}$, $f_j := f(x_j)$, and $s_j'' := s''(x_j)$.

- In this expression, f_j, f_{j+1}, s_j'', and s_{j+1}'' are four numeric values that uniquely define the cubic polynomial $s_j(x)$ on Ω_j.



Cubic Splines, continued

$$s(x)|_{\Omega_j} := s_j(x) = l_j(x)f_j + l_{j+1}(x)f_{j+1} + h_j^2[g_j(x)s_j'' + g_{j+1}(x)s_{j+1}'']$$

- f_j and f_{j+1} will be the respective interpolation values $f(x_j)$ and $f(x_{j+1})$
- s_j'' and s_{j+1}'' are two unknowns that are to be determined.
- If we knew the second derivative of f it would be appropriate to set $s_j'' = f''(x_j)$
- For cubic splines, we use the continuity conditions on $s'(x)$ and $s''(x)$ to set up a system of equations for these unknown values

Cubic Splines, continued

- Note that

$$\begin{aligned} l_j' &= -\tfrac{1}{h_j} & l_j'' &= 0 \\ l_{j+1}' &= \tfrac{1}{h_j} & l_{j+1}'' &= 0 \\ g_j' &= -\tfrac{1}{2h_j} l_j^2 + \tfrac{1}{h_j} & g_j'' &= \tfrac{1}{h_j^2} l_j \\ g_{j+1}' &= \tfrac{1}{2h_j} l_{j+1}^2 - \tfrac{1}{h_j} & g_{j+1}'' &= \tfrac{1}{h_j^2} l_{j+1} \end{aligned}$$

- Return to $s(x)$ on Ω_j,

$$s_j(x) = l_j(x) f_j + l_{j+1}(x) f_{j+1} + h_j^2 [g_j(x) s_j'' + g_{j+1}(x) s_{j+1}''],$$

and evaluate its *second derivative*

$$\begin{aligned} s_j''(x) &= \frac{d^2}{dx^2} \left(h_j^2 [g_j(x) s_j'' + g_{j+1}(x) s_{j+1}''] \right) \\ &= l_j(x) s_j'' + l_{j+1}(x) s_{j+1}'' \end{aligned}$$

- So, we do indeed have $s_j''(x_j) = s_j''$ and $s_j''(x_{j+1}) = s_{j+1}''$
- Here, we think of s_j'' and s_{j+1}'' as *unknown values* that are to be determined
- They are *basis coefficients* in the expression for $s_j(x)$

Cubic Splines, continued

- Differentiating s on Ω_j, the first derivative of $s(x)$ is

$$s'_j(x) \;=\; \frac{f_{j+1} - f_j}{h_j} + \frac{h_j}{6}\left[\,(3l_{j+1}^2 - 1)s''_{j+1} \;-\; (3l_j^2 - 1)s''_j\,\right]$$

- Rearranging, we have the following equations for $j = 2, \ldots, n-1$

$$\frac{h_{j-1}}{6}\,s''_{j-1} \;+\; \frac{h_{j-1} + h_j}{3}\,s''_j \;+\; \frac{h_j}{6}\,s''_{j+1} \;=\; \frac{\Delta f_j}{h_j} - \frac{\Delta f_{j-1}}{h_{j-1}}$$

- This is an $(n-2)\times(n-2)$ tridiagonal system that can be solved in $\approx 8(n-2)$ operations, which is optimal

- Once the s''_j values are known, the spline requires only a few operations to evaluate, but you must first identify the interval Ω_j that holds the query point x.

- Fortunately, because the x_js are sorted, the interval can be found in $\log_2 n$ time and in fact typically much less if there are many query (sorted) points, because one can always first check if the interval of the preceding spline evaluation holds the new query point

Boundary Conditions for Cubic Splines

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

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

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

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

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

- **"Not-a-Knot" spline ($s'''(a + h_1)$ and $s'''(b - h_n)$ are continuous):**

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

  – Default spline in matlab.

- Can also prescribe periodic boundary conditions on s' and s''

Some Cubic Spline Properties

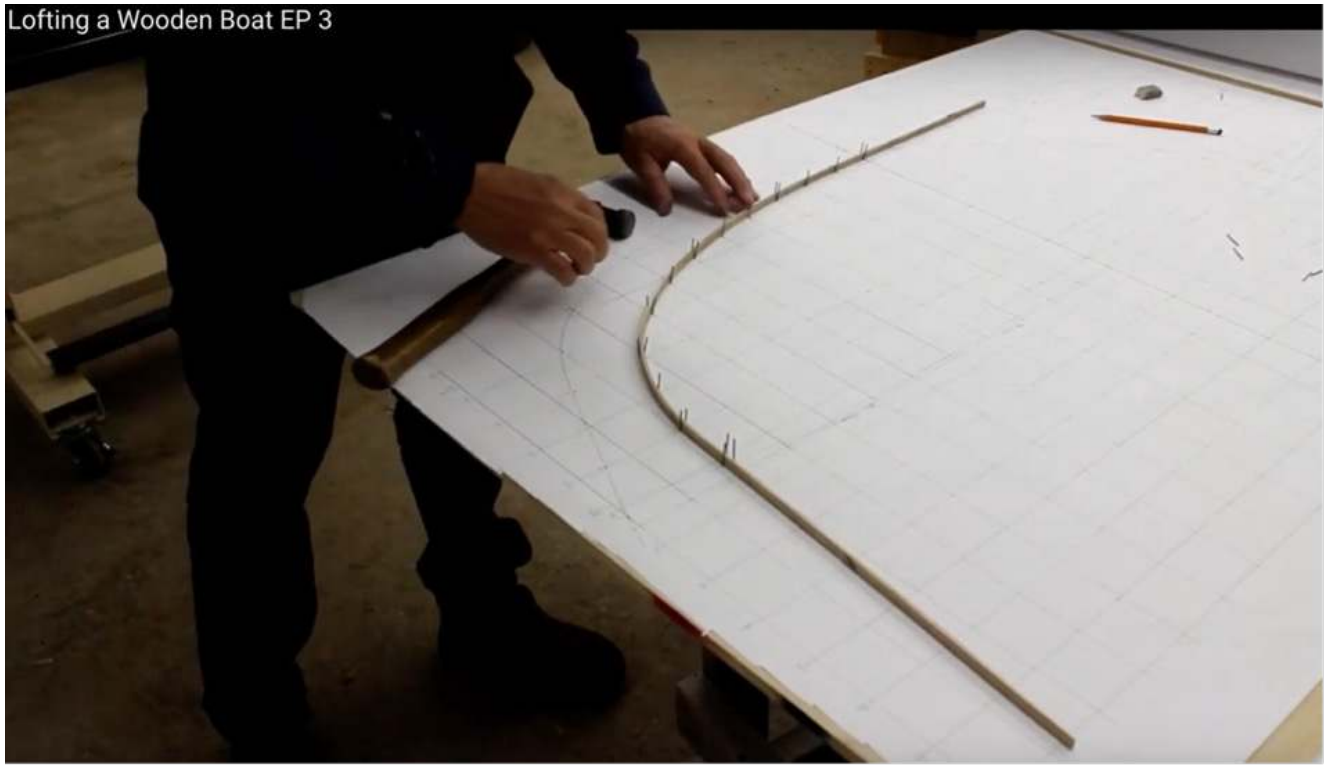
- Continuity
  - 1st derivative: continuous
  - 2nd derivative: continuous

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

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

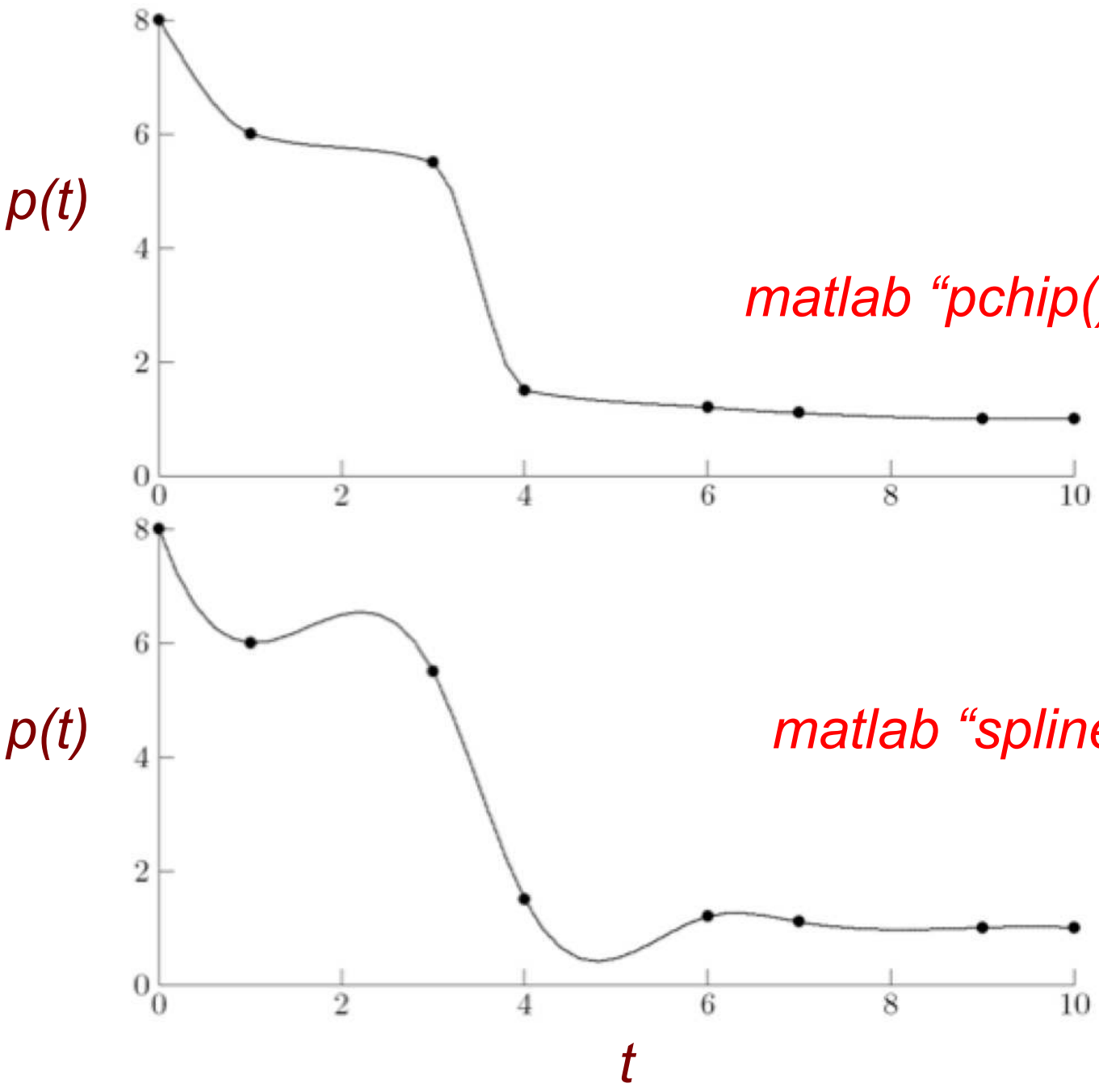
- Robust / Stable (unlike high-order polynomial interpolation)
- Commonly used in computer graphics, CAD software, etc.
- Usually used in parametric form There are other forms, e.g., tension-splines, that are also useful.
- For clamped and not-a-knot boundary conditions, convergence is $O(h^4)$
- For small displacements, natural spline is like a ***physical spline.***

Spline in Practice



- ***Questions:***
- ***What kind of boundary conditions do you see in this photo?***
  - ***Natural?***
  - ***Clamped?***
- ***Is this a function?***
- ***Is it well approximated by our numerical spline?***

Example



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.

Theory: Summary of Key Convergence Results

- **Piecewise linear interpolation:**

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

- **Polynomial interpolation through n points:**

$$\max_{t\in[a,b]} |\,p - f\,| \;\leq\; \frac{|q_n(\theta)|}{n!}\,M, \tag{2}$$

$$\leq\; \frac{h^n}{4n}\,M,\;\; (\text{for } t \in [a,b]), \tag{3}$$

$$\text{with}\;\; M := \max_{\theta\in[a,b,t]} |f^n(\theta)\,|.$$

  - Here, $q_n(\theta) := (\theta - t_1)(\theta - t_2)\cdots(\theta - t_n)$.
  - Recall that $\max|q_n(\theta)|$ is much smaller for Chebyshev points than uniform.
  - The result (3) holds also for *extrapolation*, i.e., $t \notin [a,b]$.

Convergence Results for Piecewise Cubic Splines (presented shortly)

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

$$\max_{t\in[a,b]} |\, p - f \,| \;\leq\; C\, h^2\, M, \quad M = \max_{\theta\in[a,b]} |f''(\theta)\,|, \tag{4}$$

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

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

$$\max_{t\in[a,b]} |\, p - f \,| \;\leq\; C\, h^4\, M, \quad M = \max_{\theta\in[a,b]} |f^{iv}(\theta)\,|. \tag{5}$$

- **"Not-a-Knot" spline ($s'''(a + h_1)$ and $s'''(b - h_n)$ are continuous):**

$$\max_{t\in[a,b]} |\, p - f \,| \;\leq\; C\, h^4\, M, \quad M = \max_{\theta\in[a,b]} |f^{iv}(\theta)\,|. \tag{6}$$

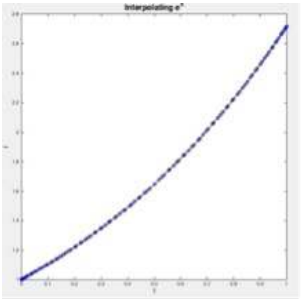
  – Default spline in matlab.

Other Conditions Limiting Accuracy

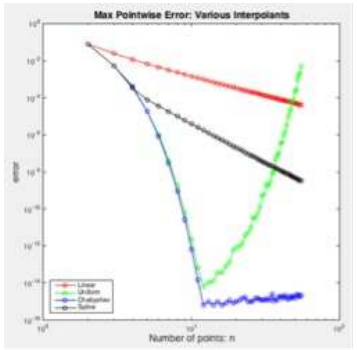
- **Nyquist sampling theorem:**
  - *The maximum frequency that can be resolved with n points is $N = n/2$.*
  - Roughly: You need $h <$ the smallest wavelength before you will observe the asymptotic behaviors given by the above theory.
  - Note that the theory results (1)–(6) are still correct, but the bounds are too conservative in the undersampled case.

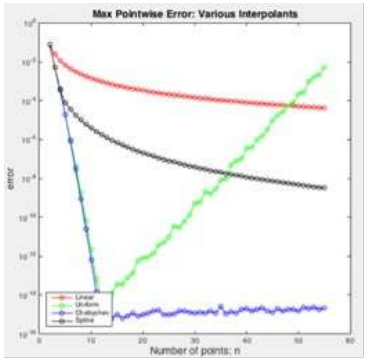
- **Round-Off:**
  - *Stable Methods*: Error will remain flat at $\approx \epsilon_M$.
  - *Unstable Methods*: Error will remain grow at $\approx \epsilon_M$ or earlier.

- **Methods:**
  - piecewise linear
  - polynomial on uniform points
  - polynomial on Chebyshev points
  - knot-a-not 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



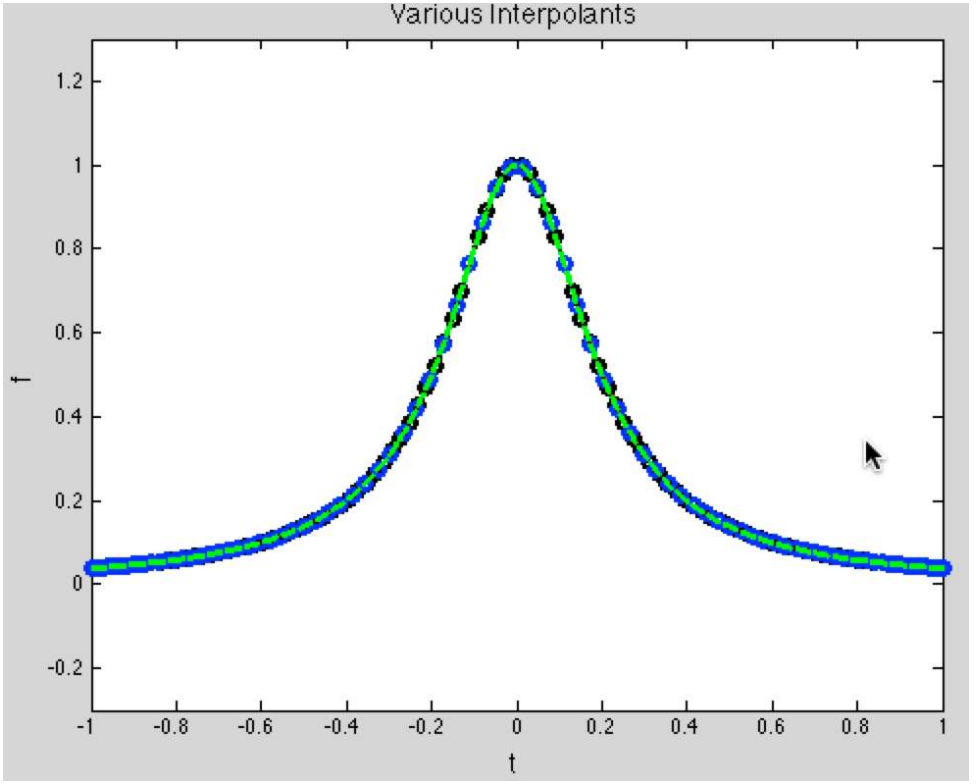


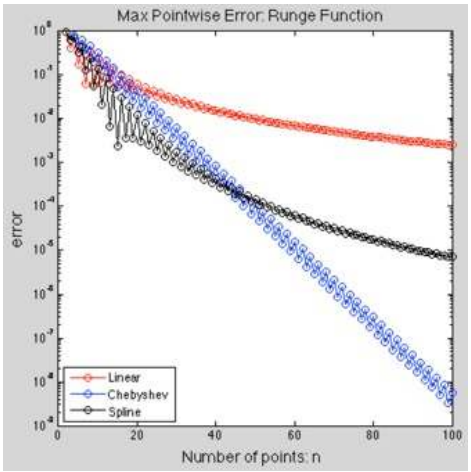
interp_test.m

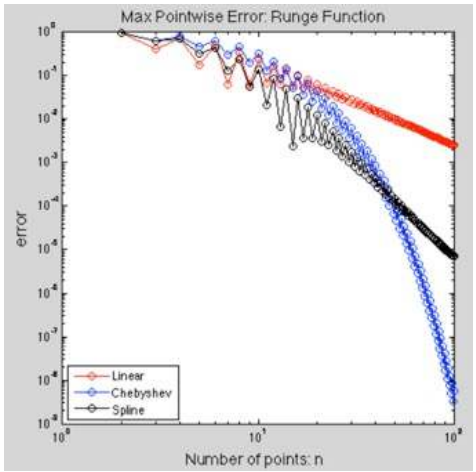
- **Methods:**
  - piecewise linear
  - polynomial on uniform points
  - polynomial on Chebyshev points
  - natural cubic spline

- **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

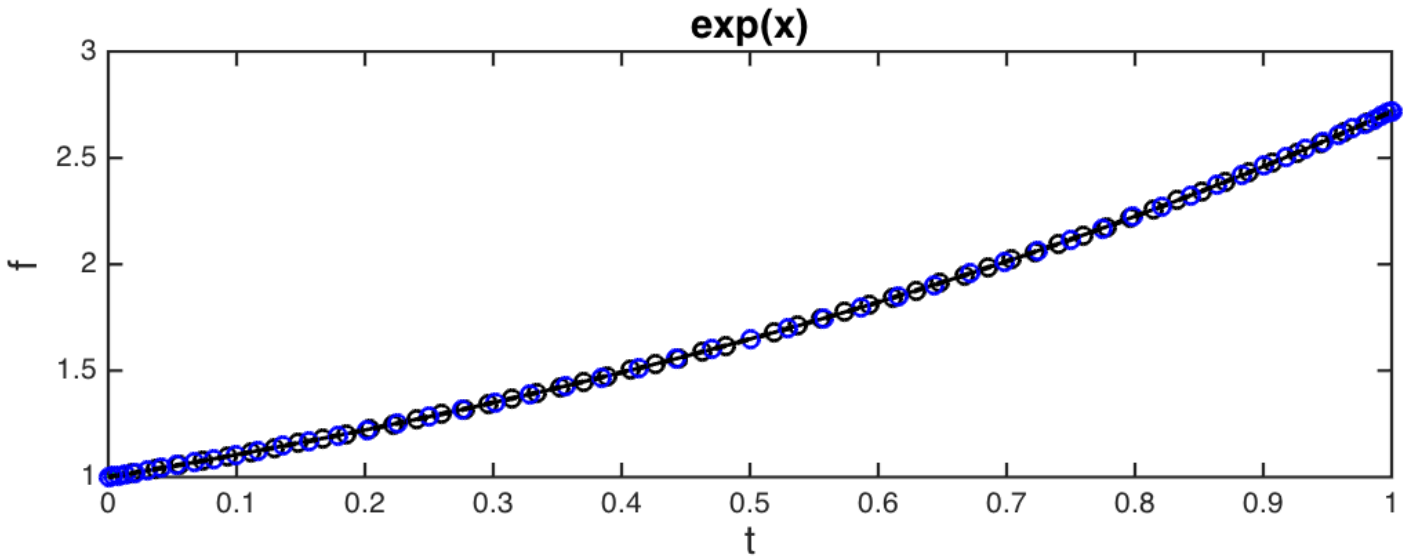
interp_test_runge.m

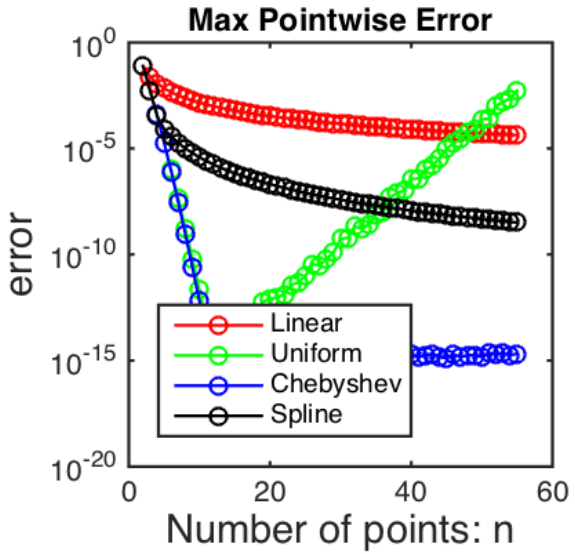


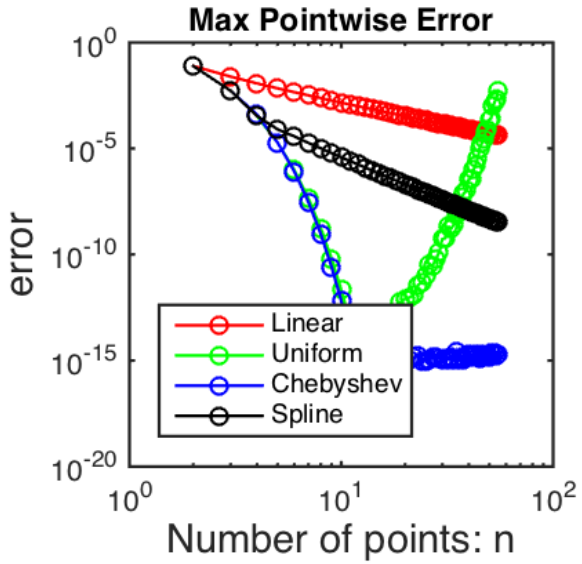




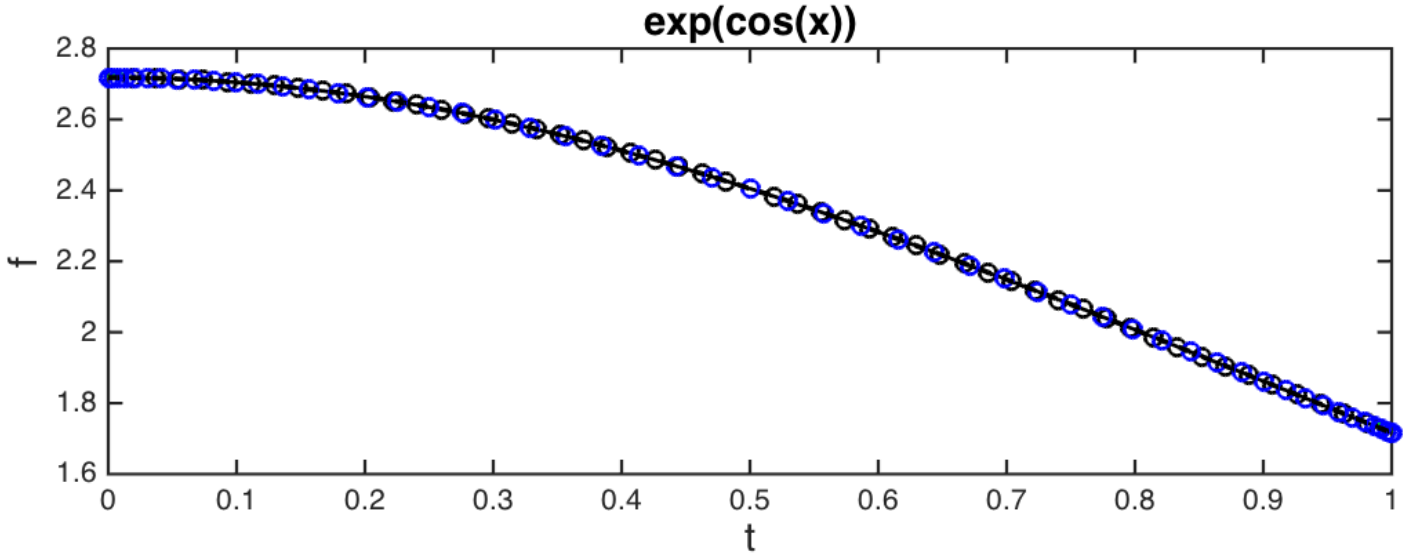
Test Cases Revisited (Pay Attention to Splines)

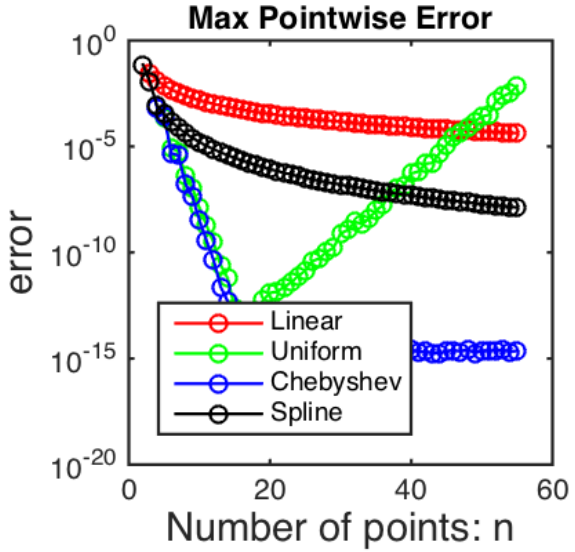


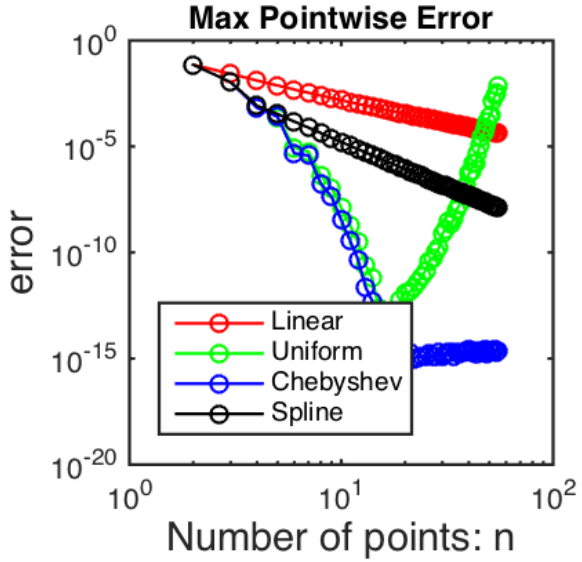




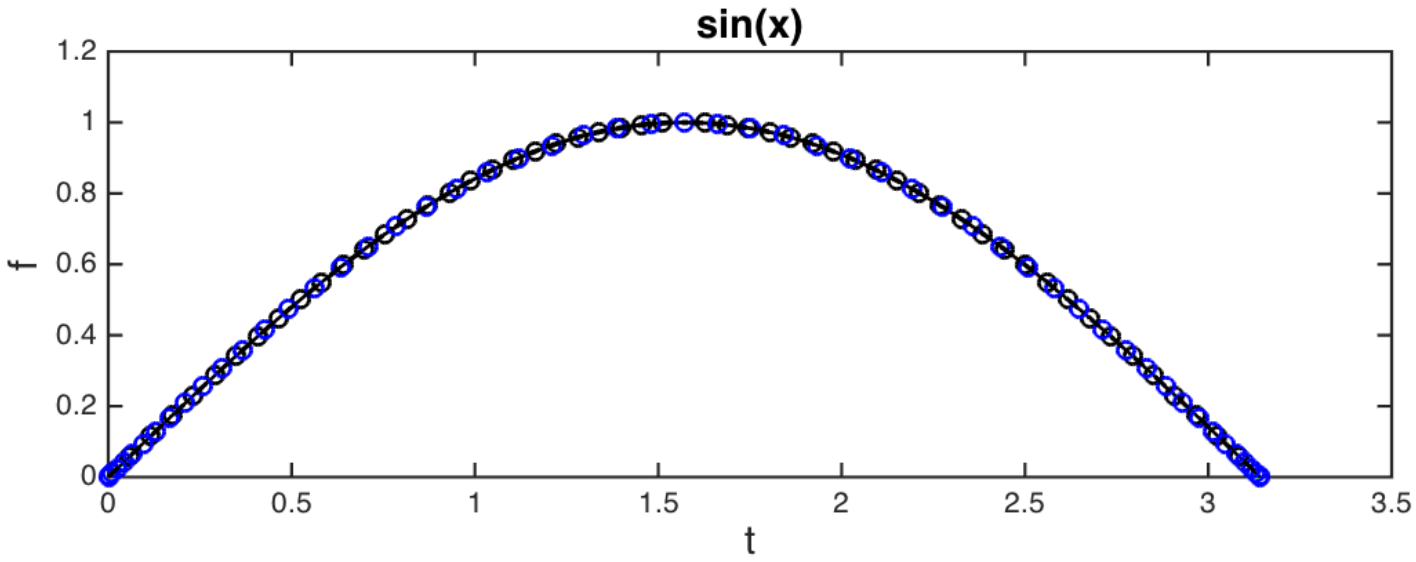
Test Cases Revisited (Pay Attention to Splines)

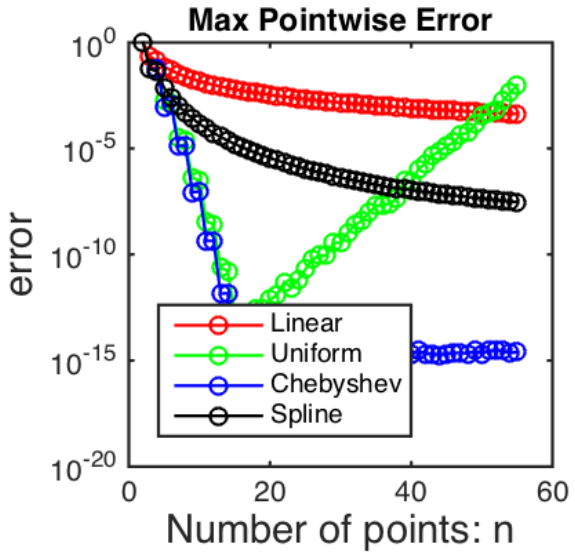


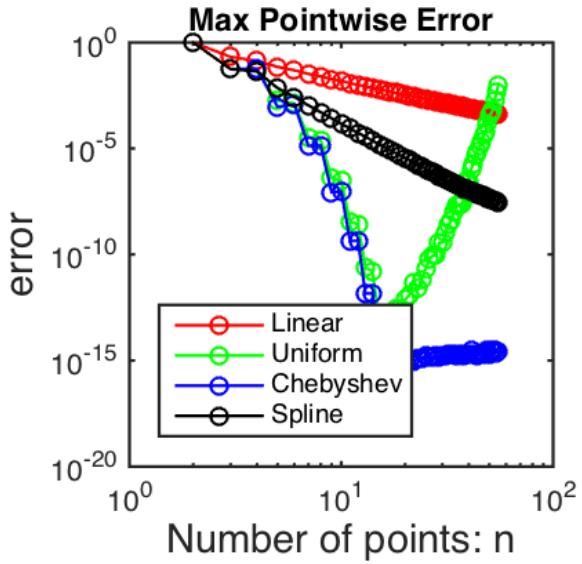




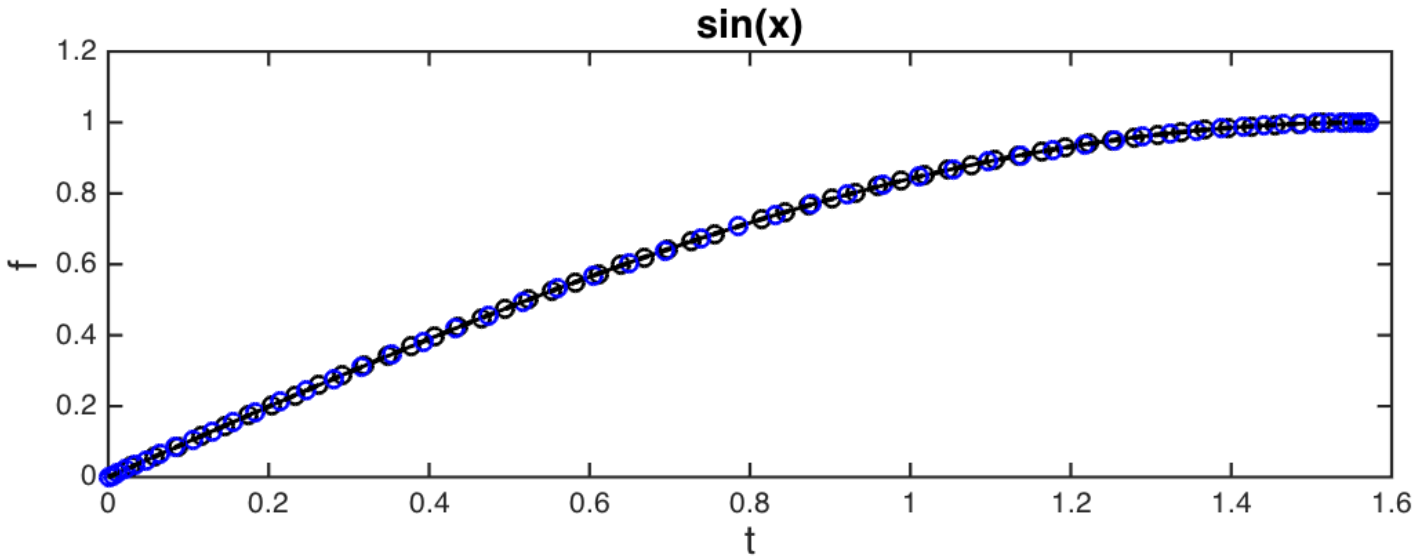
Test Cases Revisited (Pay Attention to Splines)

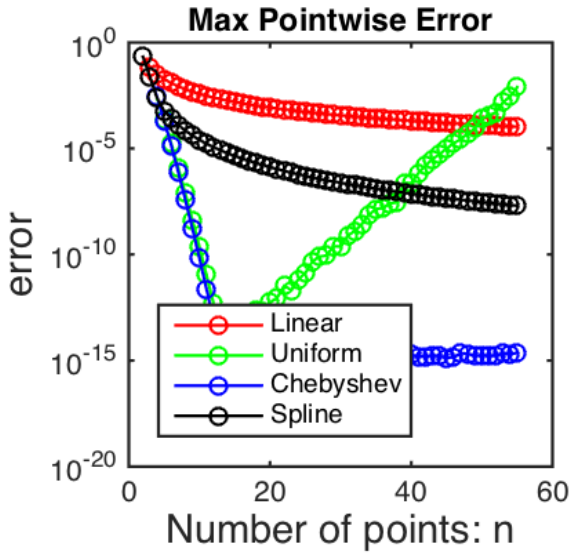


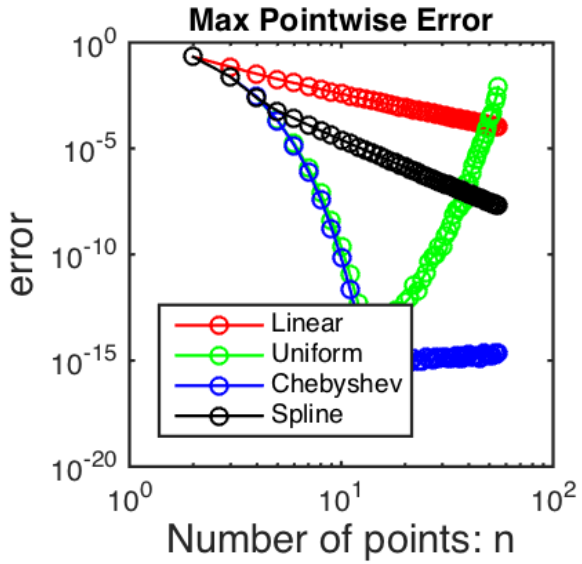




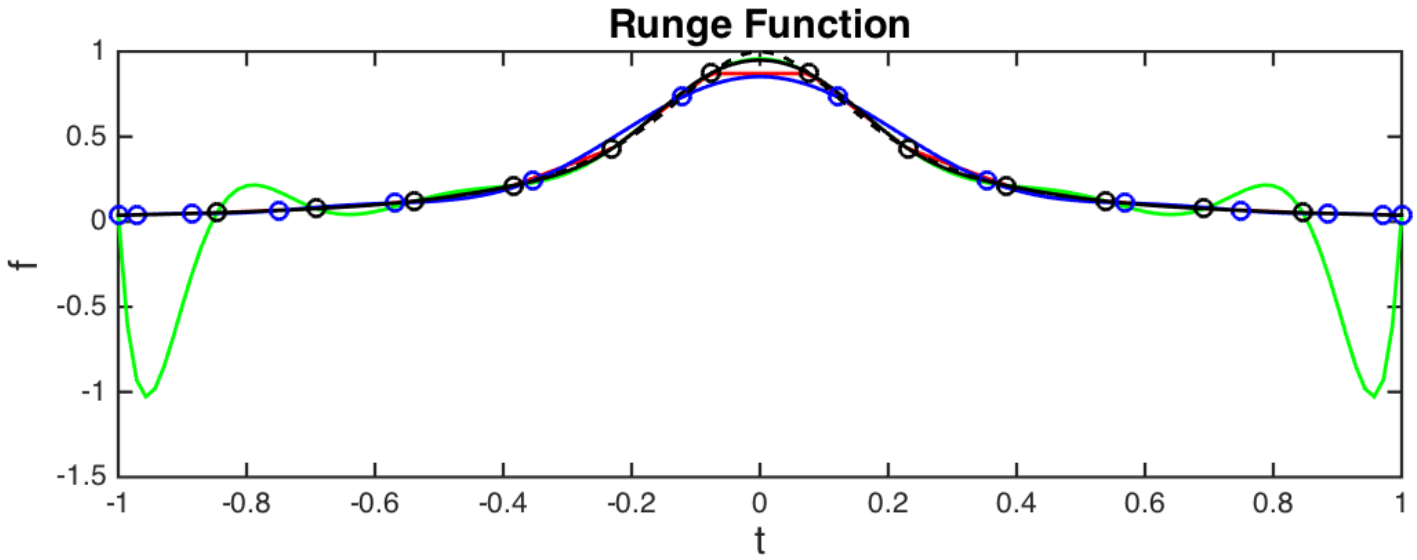
Test Cases Revisited (Pay Attention to Splines)

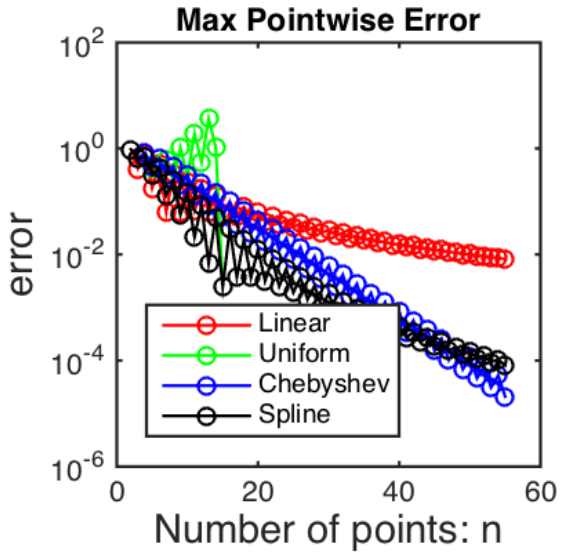


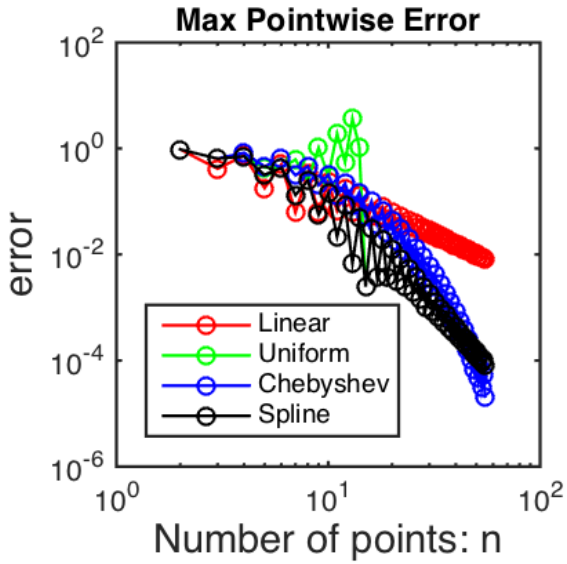




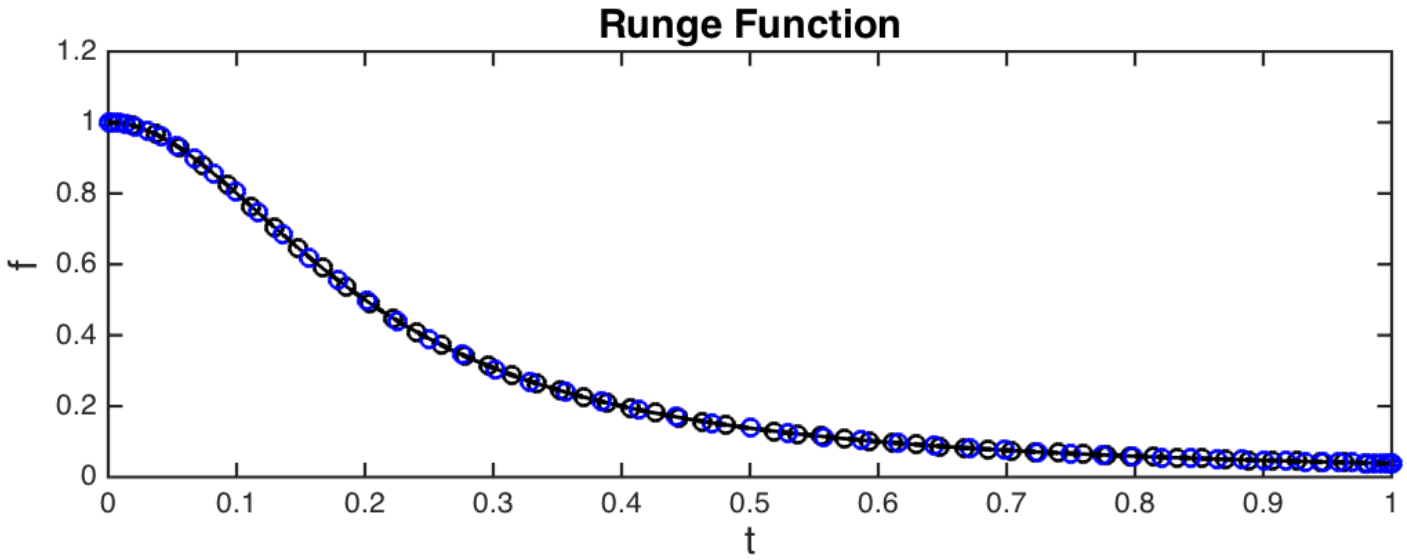
Test Cases Revisited (Pay Attention to Splines)

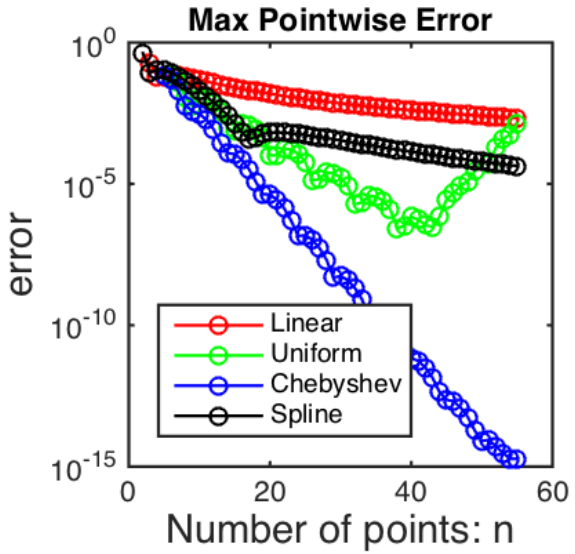


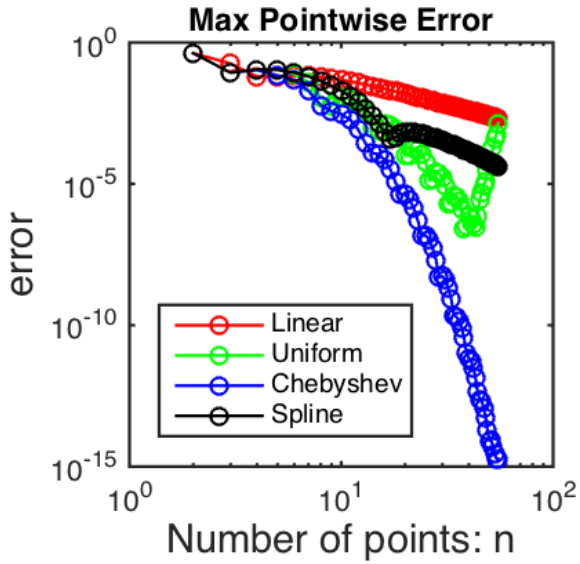




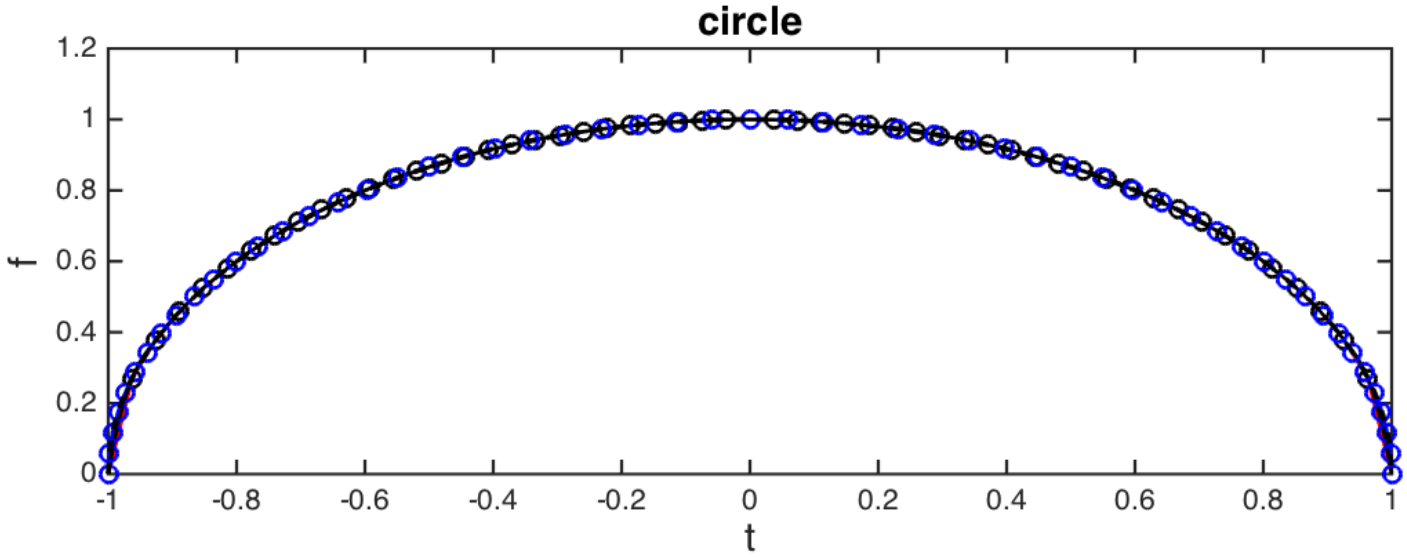
Test Cases Revisited (Pay Attention to Splines)

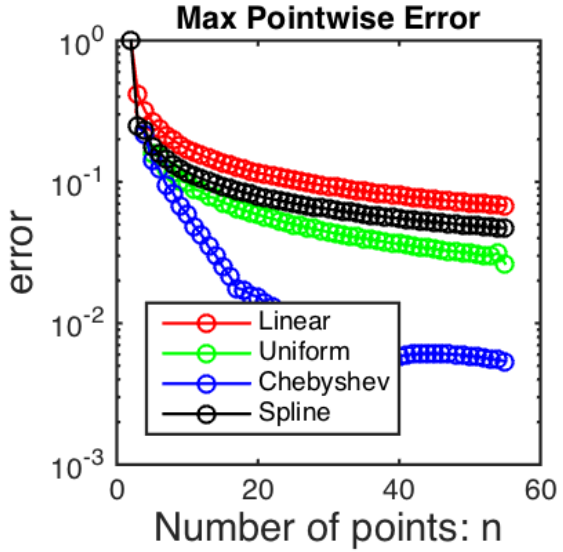


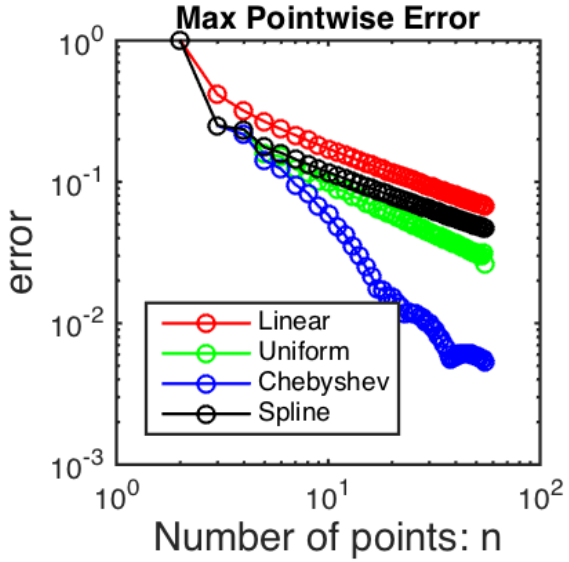




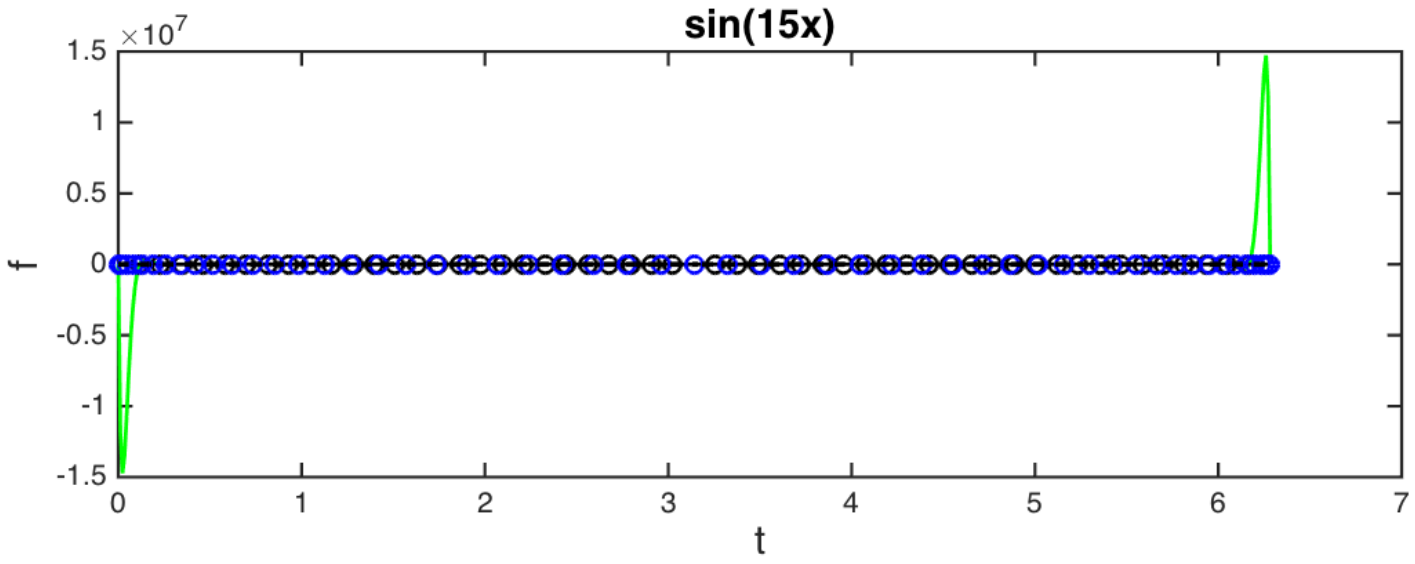
Test Cases Revisited (Pay Attention to Splines)

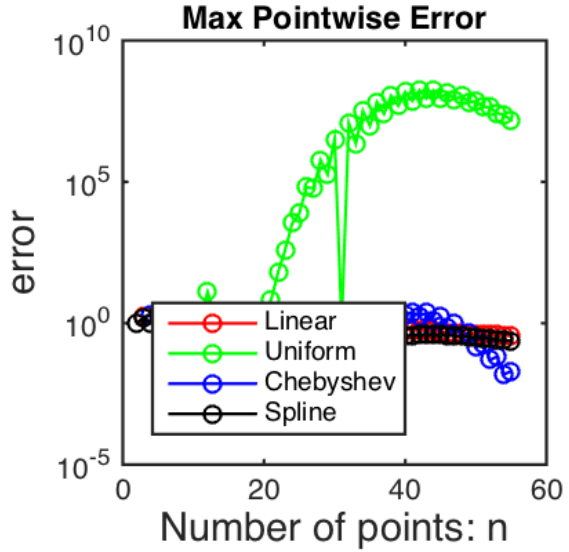


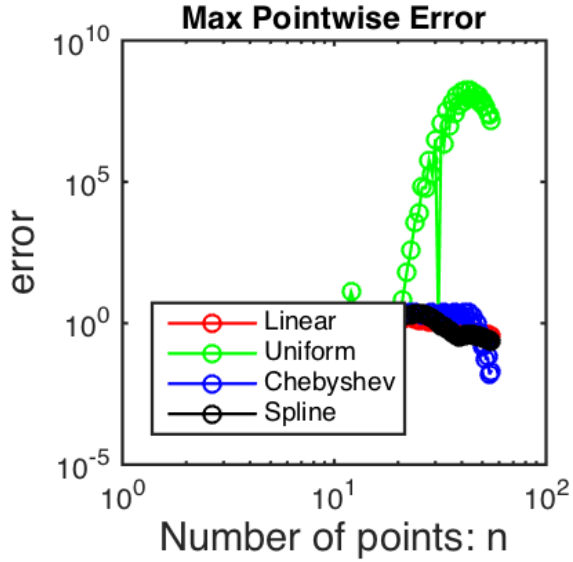




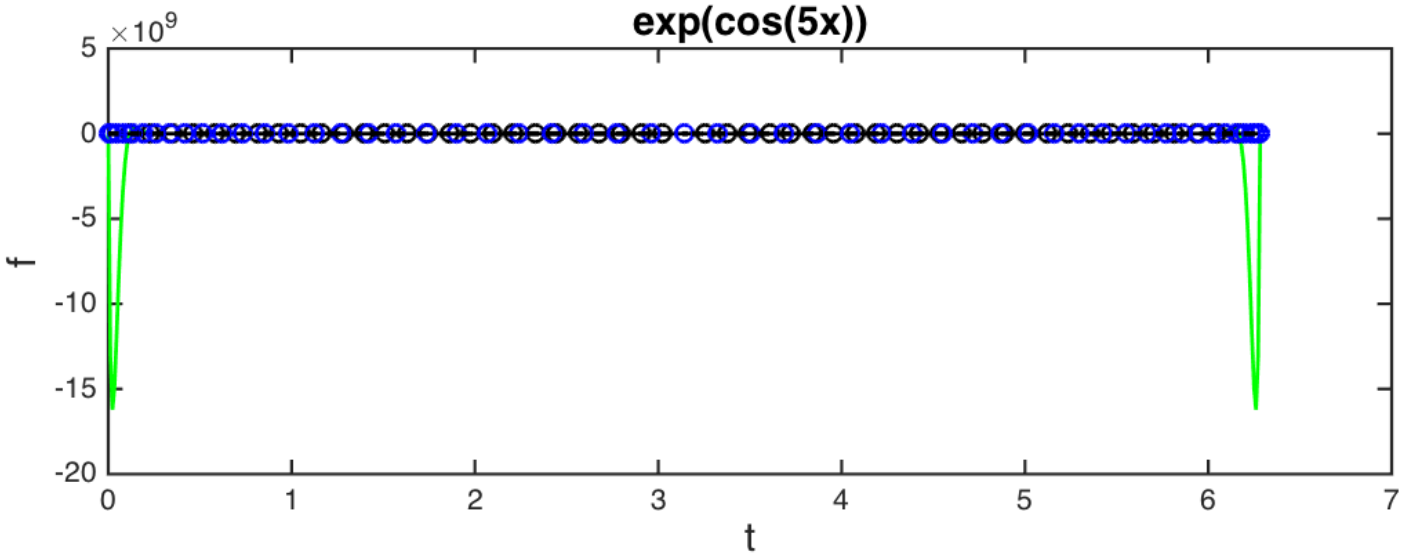
Test Cases Revisited (Pay Attention to Splines)

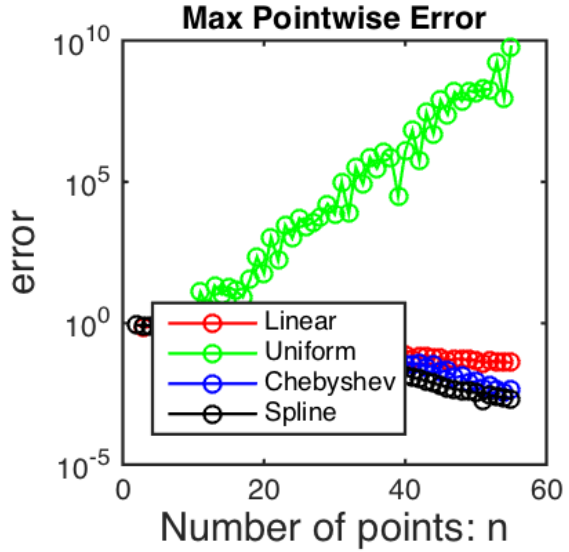


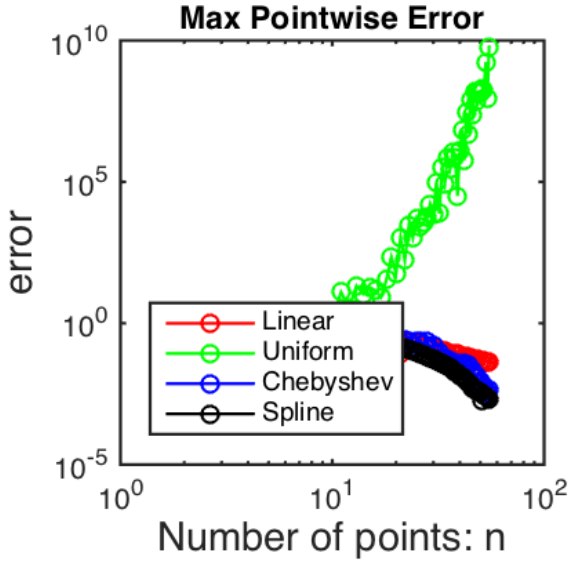




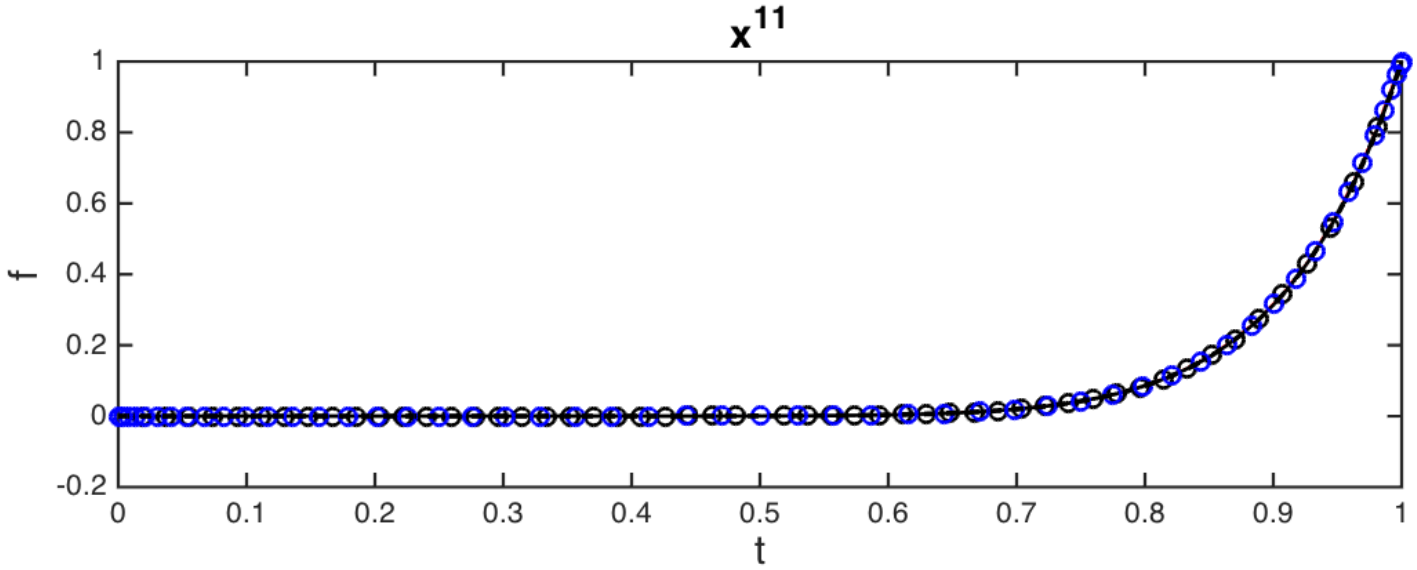
Test Cases Revisited (Pay Attention to Splines)

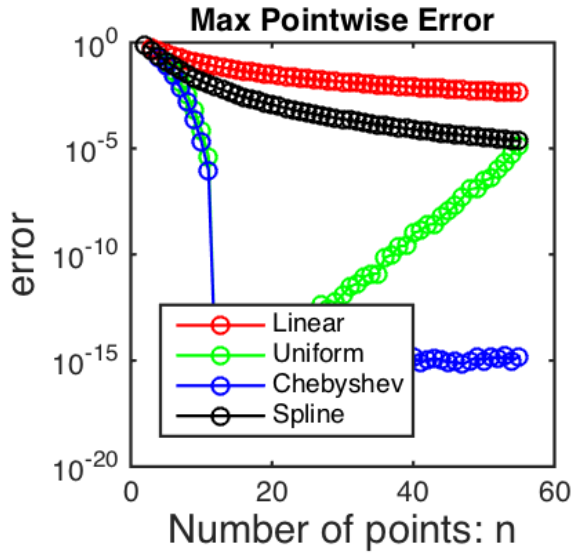


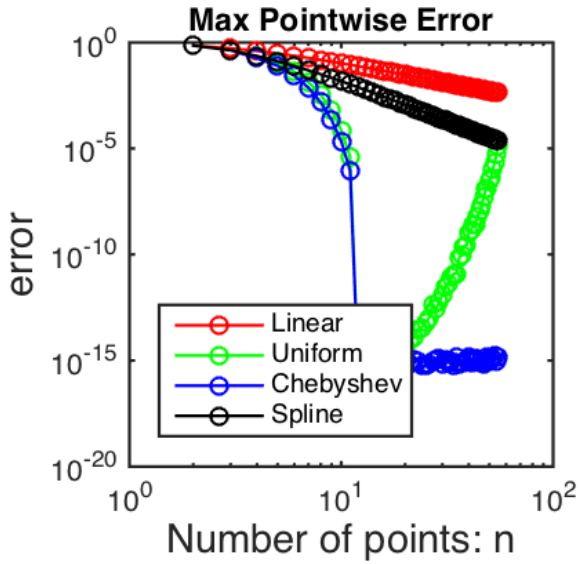




Test Cases Revisited (Pay Attention to Splines)

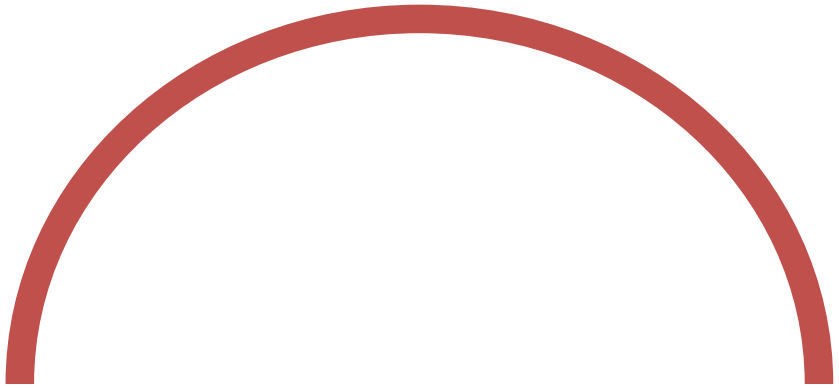






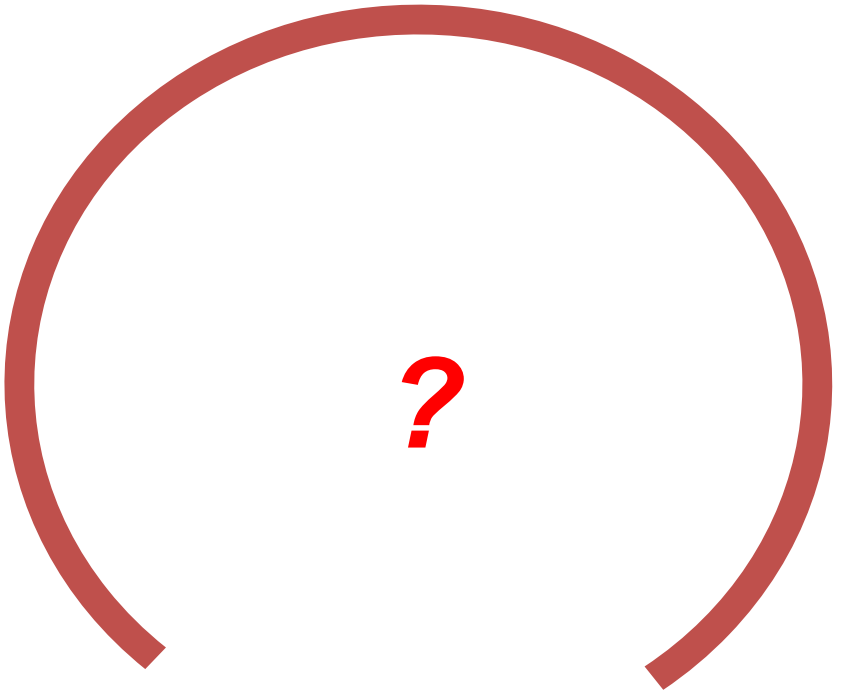
Parametric Interpolation

- It's clear we can interpolate this:



- though maybe not with great accuracy.

But what about this?

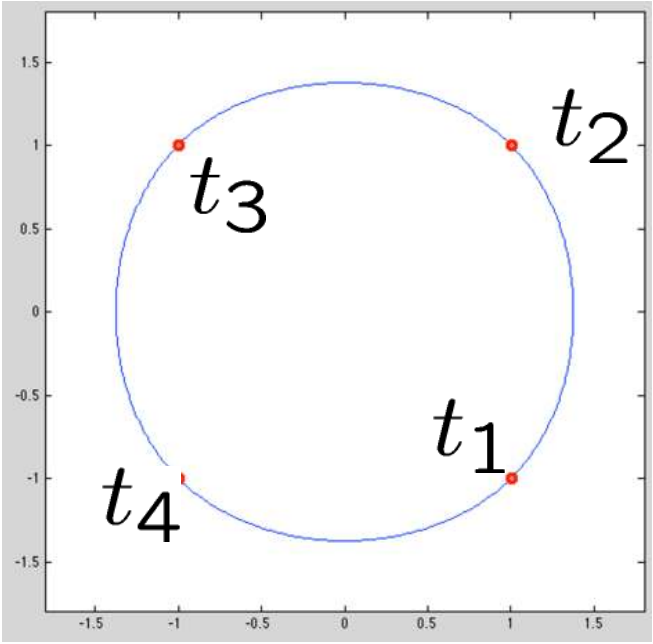


It's not even a function!

Parametric Interpolation

- 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.



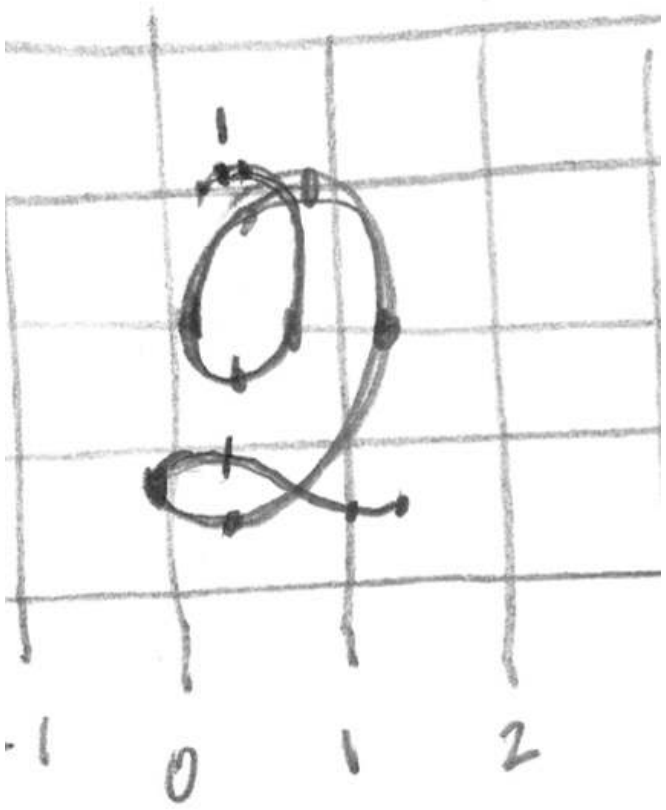

```
%% LAZY WAY TO APPROXIMATE
%% PERIODIC SPLINE

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

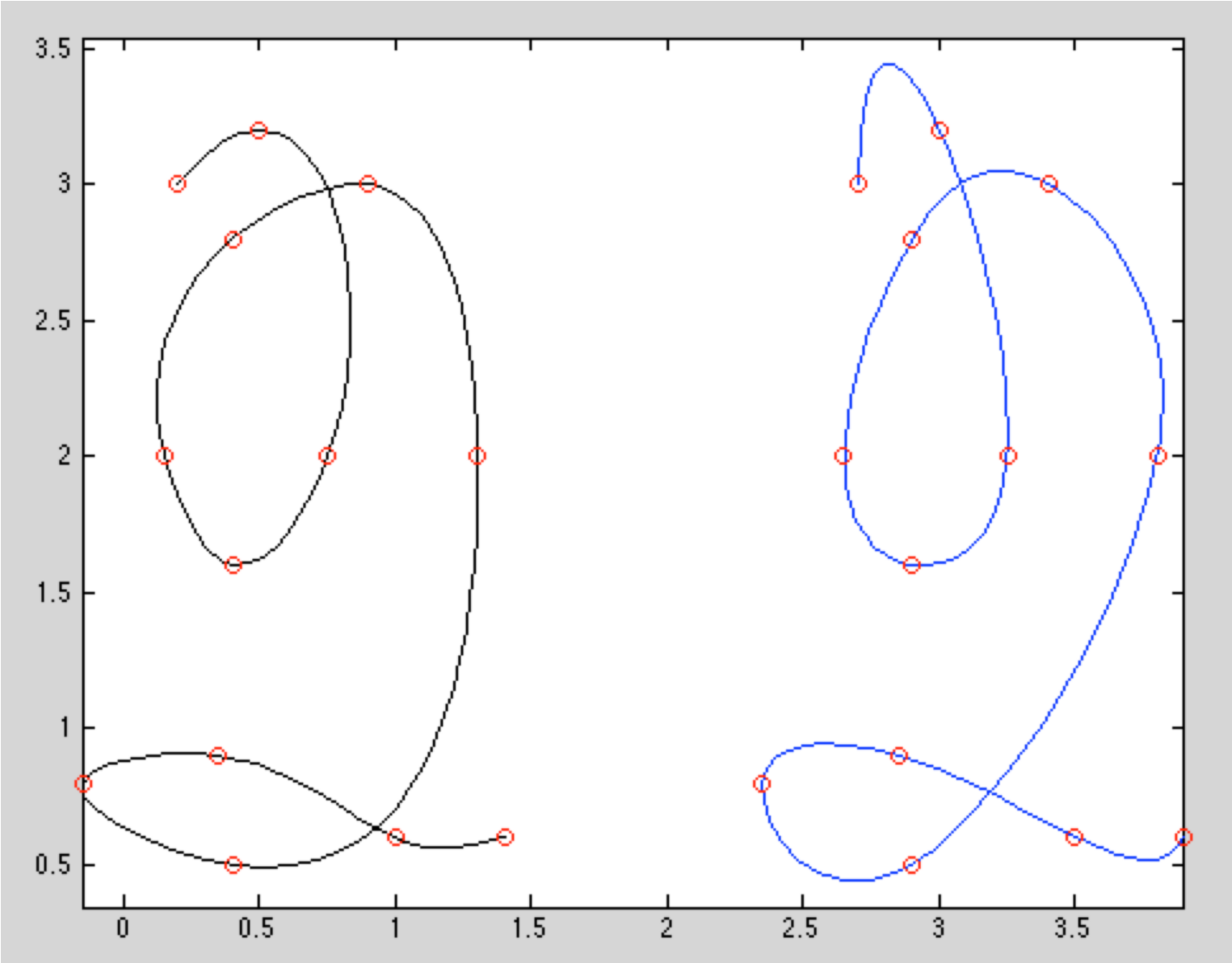


- *Once we have our (x_i , y_i) pairs, we still need to pick t_i .*
- *One possibility: $t_i = i$, but usually it's better to parameterize by arclength, if x and y have the same units.*
- *An approximate arclength is:*

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

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

Parametric Interpolation: Example 2



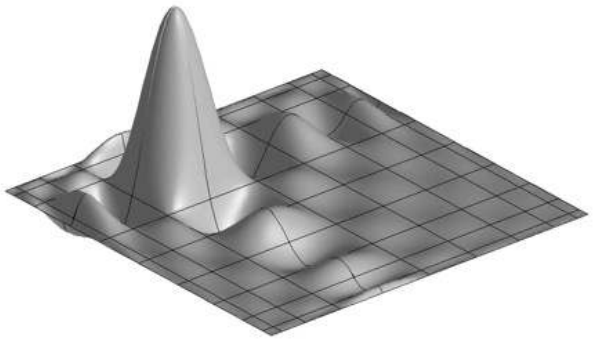
Pseudo-Arclength-Based

Index-Based

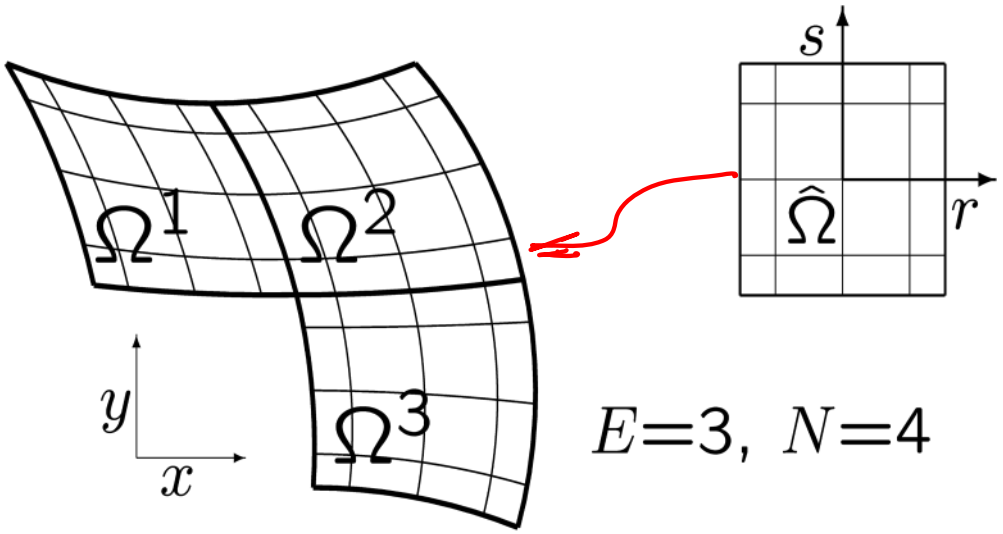
Multidimensional Interpolation

- 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.

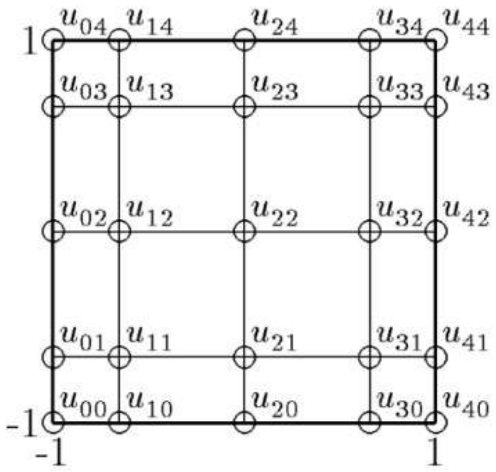


2D basis function, N=10

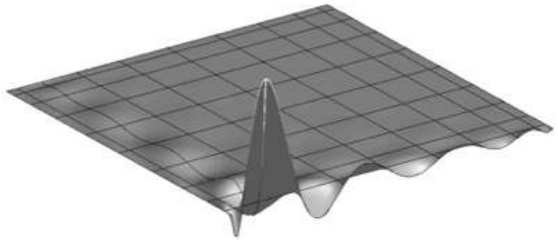


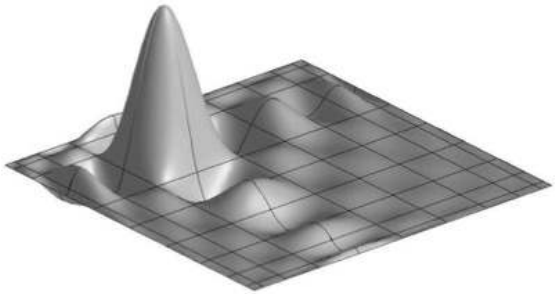
Multidimensional Interpolation

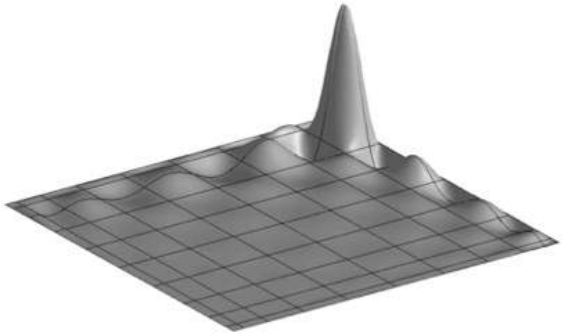
- There are many strategies for interpolating f(x,y) [or f(x,y,z), etc.].
- One easy approach is to use **tensor products** of one-dimensional interpolants, such as bicubic splines or tensor-product Lagrange polynomials.



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







Consider 1D Interpolation

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

$$p(\mathbf{s}) = \sum_{j=1}^{n} l_j(\mathbf{s})\, f_j, \qquad \mathbf{s} = [s_1\ s_2\ \cdots\ s_m]^T$$

$$p_i = \sum_{j=1}^{n} l_{ij}\, f_j, \qquad l_{ij} := l_j(s_i) =: J_{ij}$$

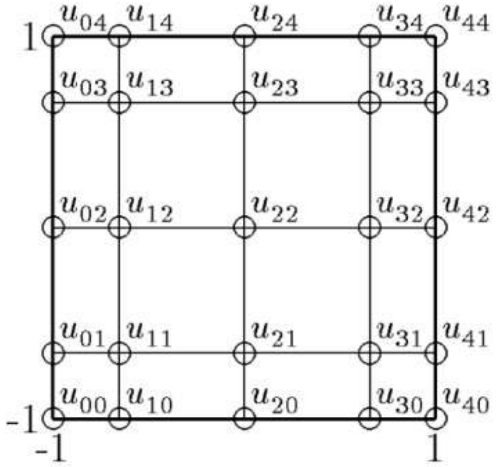
$$\mathbf{p} = J\,\mathbf{f}$$

- s_i – *fine mesh* (i.e., target gridpoints)
- J is the 1D interpolation matrix
- J is the matrix of Lagrange cardinal polynomials evaluated at the target points, s_i, $i = 1, \ldots, m$.

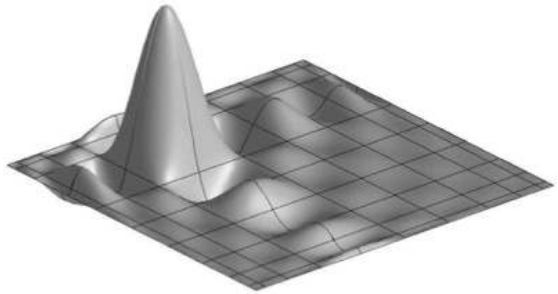
Two-Dimensional Case (say, n x n → m x m)

$$p(s,t) = \sum_{i=1}^{n}\sum_{j=1}^{n} l_i(s)\, l_j(t)\, f_{ij}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} l_i(s)\, f_{ij}\, l_j(t)$$



$$p_{pq} := p(s_p, t_q) = \sum_{i=1}^{n}\sum_{j=1}^{n} l_{pi}\, f_{ij}\, l_{qj}$$



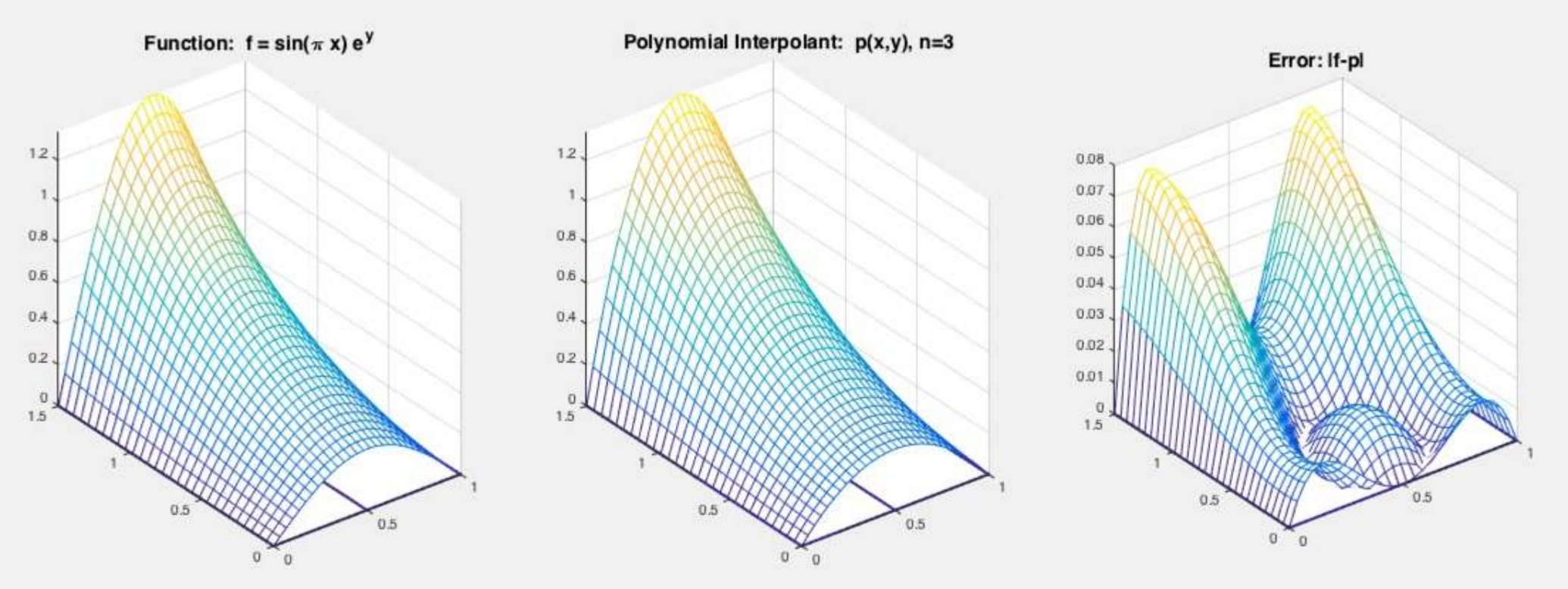
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} l_{pi}\, f_{ij}\, l^{T}_{jq}$$

$$P = JFJ^{T}$$ ← *matrix-matrix product,* ***fast***

Two-Dimensional Case (say, n x n → m x m)

- Note that the storage of J is $mn < m^2 + n^2$, which is the storage of P and F combined.
- That is, in higher space dimensions, the operator cost (J) is less than the data cost (P, F).
- This is even more dramatic in 3D, where the relative cost is mn to $m^3 + n^3$.
- Observation: *It is difficult to assess relevant operator costs based on 1D model problems.*

Two-Dimensional Case (3 x 3 → 30 x 30)



interp2d.m

Two-Dimensional Case (3 x 3 → 30 x 30)

```
%% Interpolate f(x,y) for different values of n

lw='linewidth'; fs='fontsize'; format compact;

ax=0; bx=1;   ay=0; by=1.5;          %% DOMAIN:  [ax,bx] X [ay,by]

m=30;                                %% Number of fine points


%  Set up grids, X_ij and Y_ij

x =xglc(ax,bx,n); y =xglc(ay,by,n); %% Chebyshev nodes and
xt=xuni(ax,bx,m); yt=xuni(ay,by,m); %%    uniform interrogation pts.

[X ,Y ]=ndgrid(x ,y );               %% Map (x,y) to 2D grid
[Xt,Yt]=ndgrid(xt,yt);               %%  ... same for fine points


%  Evaluate function

F = .3*sin(pi*X).*exp(Y);            %% F_ij  - Interpolation values
Ft= .3*sin(pi*Xt).*exp(Yt);          %% Ft_ij - Exact values on fine mesh


%  Construct interpolant

Jx=interp_mat(xt,x);     %% 1-D Interpolation matrices
Jy=interp_mat(yt,y);

Pt = Jx*F*Jy';           %% TENSOR CONTRACTION


subplot(1,3,1);                      %% PLOT ORIGINAL FUNCTION
hold off; mesh(X,Y,0*F,lw,2); hold on; mesh(Xt,Yt,Ft); axis equal;
title('Function:  f = sin(\pi x) e^y',fs,14);

subplot(1,3,2);                      %% PLOT INTERPOLANT
hold off; mesh(X,Y,0*F,lw,2); hold on; mesh(Xt,Yt,Pt); axis equal;
title('Polynomial Interpolant:  p(x,y), n=3',fs,14);

subplot(1,3,3);                      %% PLOT ERROR
hold off; mesh(X,Y,0*F,lw,2); hold on; mesh(Xt,Yt,abs(Ft-Pt));  axis square
title('Error: |f-p|',fs,14);
```