# Scientific Computing: Interpolation

**Aleksandar Donev**
*Courant Institute, NYU*[1]
*donev@courant.nyu.edu*

[1] Course MATH-GA.2043 or CSCI-GA.2112, Fall 2020

October 22nd, 2020

# Outline

# Outline

# Function Spaces

- **Function spaces** are the equivalent of finite vector spaces for functions (space of polynomial functions $\mathcal{P}$, space of smoothly twice-differentiable functions $\mathcal{C}^2$, etc.).
- Consider a one-dimensional interval $I = [a, b]$. Standard norms for functions similar to the usual vector norms:
  - **Maximum norm**: $\|f(x)\|_\infty = \max_{x \in I} |f(x)|$
  - $L_1$ **norm**: $\|f(x)\|_1 = \int_a^b |f(x)| \, dx$
  - **Euclidian** $L_2$ **norm**: $\|f(x)\|_2 = \left[\int_a^b |f(x)|^2 \, dx\right]^{1/2}$
  - **Weighted norm**: $\|f(x)\|_w = \left[\int_a^b |f(x)|^2 \, w(x) dx\right]^{1/2}$

  Orthogonal polynomials
- An $L_2$ **inner or scalar product** (equivalent of dot product for vectors):

$$(f, g) = \int_a^b f(x) g^\star(x) dx$$

$$\int f(x)\,dx \approx h \sum_{i=0}^{m} f(x_i)$$

# Finite-Dimensional Function Spaces

Approximations

- Formally, function spaces are **infinite-dimensional linear spaces**. Numerically we always **truncate and use a finite basis**.
- Consider a set of $m+1$ **nodes** $x_i \in \mathcal{X} \subset I$, $i = 0, \ldots, m$, and define:

$x_i = m \cdot h$

$h = \frac{b-a}{m}$

$$\|f(x)\|_2^{\mathcal{X}} = \left[ h \sum_{i=0}^{m} |f(x_i)|^2 \right]^{1/2}, \quad = h^{1/2} \|f_{\mathcal{X}}\|_2$$

$\|f(x)\|_1 = h \|f_{\mathcal{X}}\|_1$

which is equivalent to thinking of the function as being the vector $\mathbf{f}_{\mathcal{X}} = \mathbf{y} = \{f(x_0), f(x_1), \cdots, f(x_m)\}$.

- **Finite representations** lead to **semi-norms**, but this is not that important.
- A **discrete dot product** can be just the vector product:

$$(f,g)^{\mathcal{X}} \overset{h}{=} \mathbf{f}_{\mathcal{X}} \cdot \mathbf{g}_{\mathcal{X}} = h \sum_{i=0}^{m} f(x_i) g^{\star}(x_i)$$

# Function Space Basis

- Think of a function as a vector of coefficients in terms of a set of $n$ **basis functions**:

script \varphi or phi

$$\{\phi_0(x), \phi_1(x), \ldots, \phi_n(x)\},$$

what is a good choice?

for example, the monomial basis $\phi_k(x) = x^k$ for polynomials.

- A finite-dimensional approximation to a given function $f(x)$:

$\vec{c}$ is finite-dimensional

How accurate is this?

$$\tilde{f}(x) = \sum_{i=1}^{n} c_i \phi_i(x) \approx f(x)$$

- **Least-squares approximation** for $m > n$ (usually $m \gg n$):

$$\mathbf{c}^\star = \arg\min_{\mathbf{c}} \left\| \underline{f(x)} - \underline{\tilde{f}(x)} \right\|_2,$$

how do we do this numerically efficiently & stable

which gives the **orthogonal projection** of $f(x)$ onto the finite-dimensional basis.

# Outline

# Interpolation in 1D (Cleve Moler)

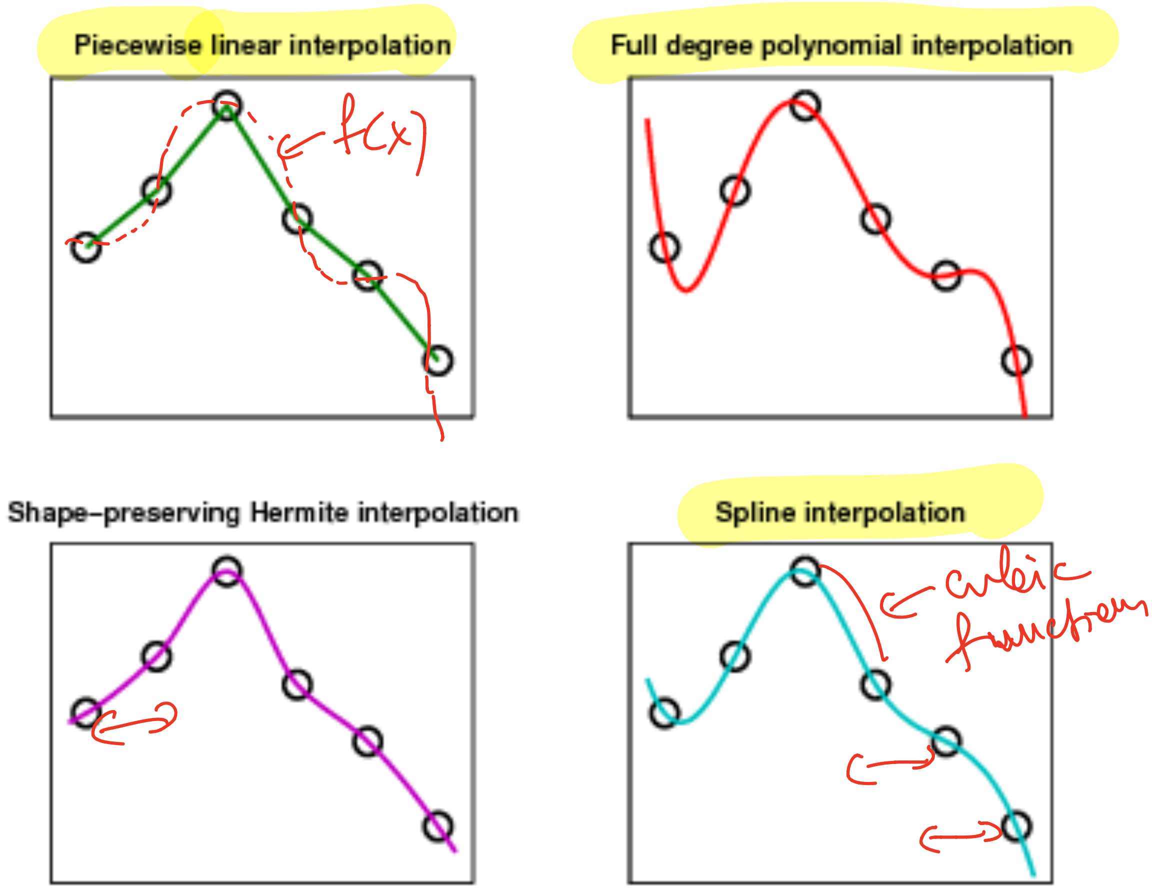



**Figure 3.8.** *Four interpolants.*

# Interpolation

what space

- The task of interpolation is to find an **interpolating function** $\phi(\mathbf{x})$ which passes through $m+1$ **data points** $(\mathbf{x}_i, y_i)$:

$f''$

$$\phi(\mathbf{x}_i) = y_i = f(\mathbf{x}_i) \text{ for } i = 0, 2, \ldots, m,$$

where $\mathbf{x}_i$ are given **nodes**.

$C_2$, $L_2$, $C^\infty$, analytic

- The type of interpolation is classified based on the form of $\phi(\mathbf{x})$:
  - Full-degree **polynomial** interpolation if $\phi(\mathbf{x})$ is globally polynomial.
  - **Piecewise polynomial** if $\phi(\mathbf{x})$ is a collection of local polynomials:
    - Piecewise linear or quadratic
    - **Hermite** interpolation
    - **Spline** interpolation

    } piecewise cubic

  Fast Fourier Transform

  - **Trigonometric** if $\phi(\mathbf{x})$ is a trigonometric polynomial (polynomial of sines and cosines), leading to the Fast Fourier Transform.
- As for root finding, in dimensions higher than one things are more complicated! Large dimensions → NN as function approximations

# Polynomial interpolation in 1D

- The **interpolating polynomial** is degree at most $m$

$$\phi(x) = \sum_{i=0}^{m} a_i x^i = \sum_{i=0}^{m} a_i p_i(x),$$

  where the **monomials** $p_i(x) = x^i$ form a basis for the **space of polynomial functions**.

- The coefficients $\mathbf{a} = \{a_1, \dots, a_m\}$ are solutions to the square linear system:

$$\phi(x_i) = \sum_{j=0}^{m} a_j x_i^j = y_i \text{ for } i = 0, 2, \dots, m$$

square linear system

- In matrix notation, if we start indexing at zero:

$$[\mathbf{V}(x_0, x_1, \dots, x_m)]\,\mathbf{a} = \mathbf{y}$$

  where the **Vandermonde matrix** $\mathbf{V} = \{v_{i,j}\}$ is given by

$$v_{i,j} = x_i^j.$$

# The Vandermonde approach

$$\mathbf{V}\mathbf{a} = \mathbf{x}$$

- One can prove by induction that

$$\det \mathbf{V} = \prod_{j<k} (x_k - x_j)$$

  which means that the Vandermonde system is non-singular and thus: The intepolating polynomial is **unique if the nodes are distinct**.
- Polynomail interpolation is thus equivalent to solving a linear system.
- However, it is easily seen that the Vandermonde matrix can be very **ill-conditioned.** for high degree polynomials.
- Solving a full linear system is also not very efficient because of the special form of the matrix.

# Choosing the right basis functions

- There are many mathematically equivalent ways to rewrite the unique interpolating polynomial:

$$x^2 - 2x + 4 = (x-2)^2.$$

$(x-2)(x-3)$

- One can think of this as choosing a different **polynomial basis** $\{\phi_0(x), \phi_1(x), \ldots, \phi_m(x)\}$ for the function space of polynomials of degree at most $m$:

$$\phi(x) = \sum_{i=0}^{m} a_i \phi_i(x)$$

- For a given basis, the coefficients **a** can easily be found by solving the linear system

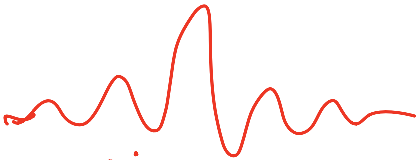
$\Phi_{ij} = \varphi_i(x_j)$

$$\phi(x_j) = \sum_{i=0}^{m} a_i \phi_i(x_j) = y_j \quad \Rightarrow \quad \mathbf{\Phi a} = \mathbf{y}$$

Best: $\Phi = I \Rightarrow a = y$ costs nothing

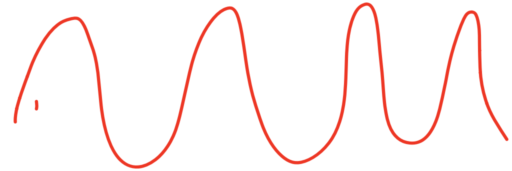
# Lagrange basis

- Instead of writing polynomials as sums of monomials, let's consider a more general **polynomial basis** $\{\phi_0(x), \phi_1(x), \ldots, \phi_m(x)\}$:



sinc

$$\phi(x) = \sum_{i=0}^{m} a_i \phi_i(x),$$



sin

as in $x^2 - 2x + 4 = (x-2)^2$.

- In particular let's consider the **Lagrange basis** which consists of polynomials that vanish at all but exactly one of the nodes, where they are unity:

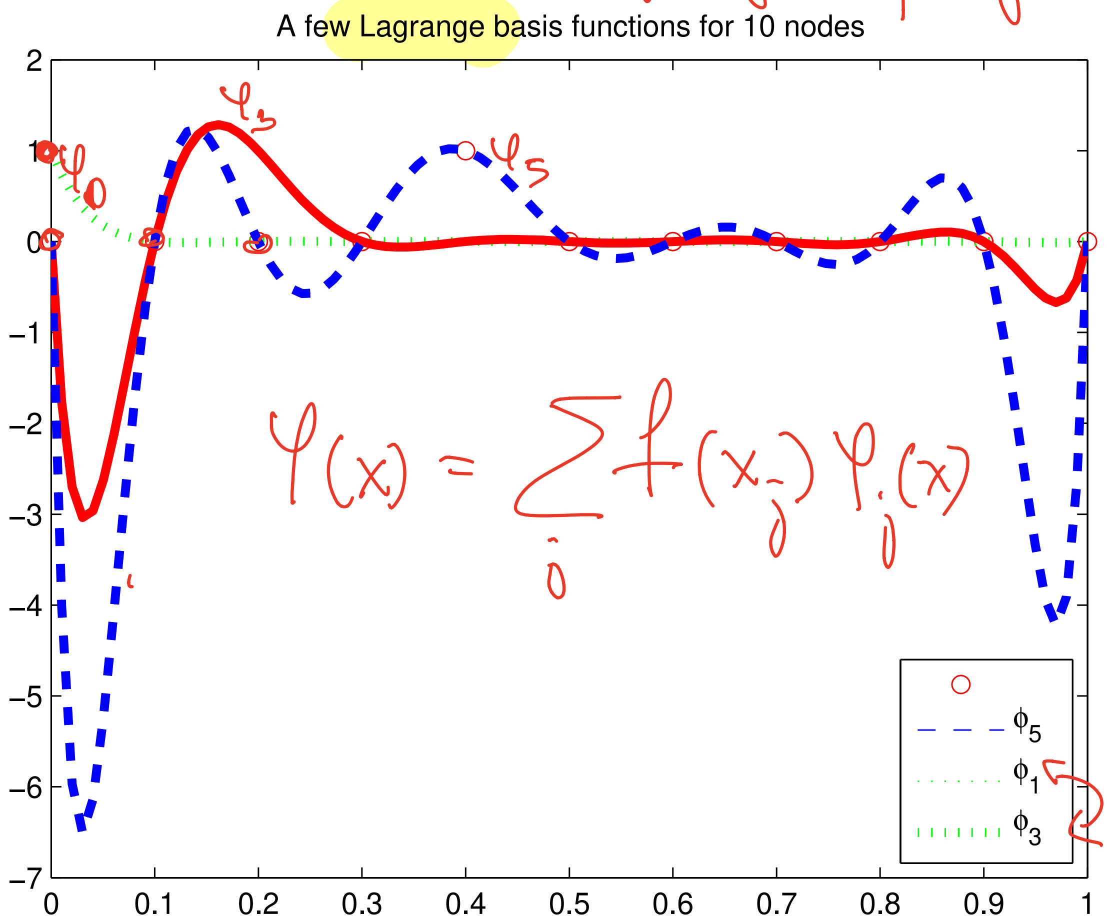
$$\phi_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}.$$

- The following **characteristic polynomial** provides the desired basis:

$$\phi_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \Rightarrow \varphi_i(x_i) = 1$$

# Lagrange basis on 10 nodes

Lagrange polynomials




$$\varphi(x) = \sum_j f(x_j)\,\varphi_j(x)$$

# Convergence, stability, etc.

- We have lost track of our goal: How good is polynomial interpolation?
- Assume we have a function $f(x)$ that we are trying to **approximate** over an interval $I = [x_0, x_m]$ using a polynomial interpolant.
- Using Taylor series type analysis it can be shown that for **equi-spaced nodes**, $x_{i+1} = x_i + h$, where $h$ is a **grid spacing**,

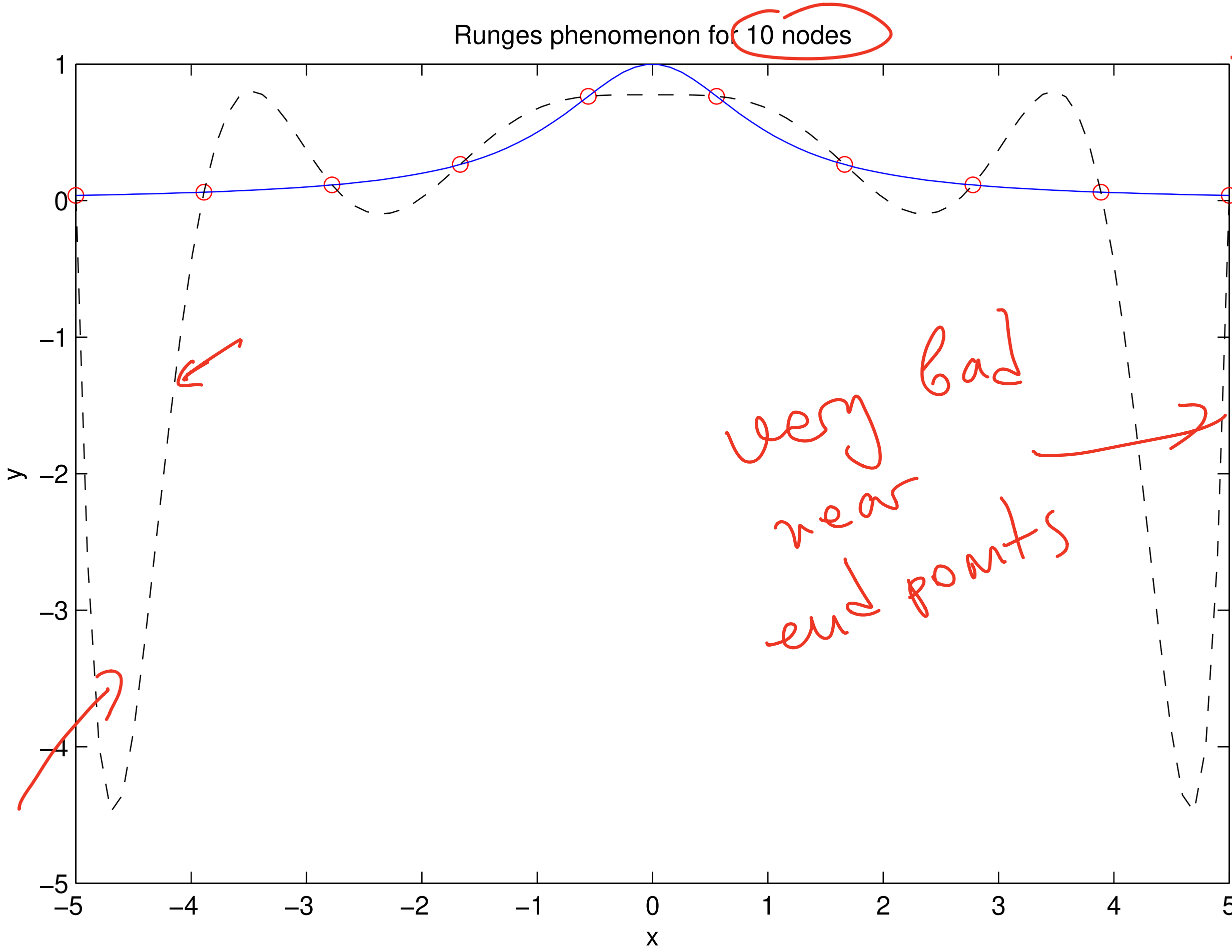
$$\|E_m(x)\|_\infty = max_{x \in I} |f(x) - \phi(x)| \leq \frac{h^{n+1}}{4(m+1)} \left\| f^{(m+1)}(x) \right\|_\infty .$$

Question: Does $\|E_m(x)\|_\infty \to 0$ as $m \to \infty$?

- In practice we may be dealing with **non-smooth functions**, e.g., discontinuous function or derivatives.
  Furthermore, higher-order derivatives of seemingly nice functions can be very large!

# Runge's counter-example: $f(x) = (1 + x^2)^{-1}$

$\frac{1}{1+x^2}$



# Uniformly-spaced nodes

- Not all functions can be approximated well by an interpolating polynomial with **equally-spaced nodes** over an interval.
- Interpolating polynomials of higher degree tend to be **very oscillatory** and **peaked**, especially near the endpoints of the interval.
- Even worse, the **interpolation is unstable**, under small perturbations of the points $\tilde{\mathbf{y}} = \mathbf{y} + \delta\mathbf{y}$,

$$\|\delta\phi(x)\|_\infty \leq \frac{2^{m+1}}{m \log m} \|\delta\mathbf{y}\|_\infty$$

- It is possible to vastly improve the situation by using **specially-chosen non-equispaced nodes** (e.g., Chebyshev nodes), or by **interpolating derivatives** (Hermite interpolation).
- A true understanding would require developing **approximation theory** and looking into **orthogonal polynomials**, which we will not do here.

# Chebyshev Nodes

- A simple but good alternative to equally-spaced nodes are the **Chebyshev nodes**,

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

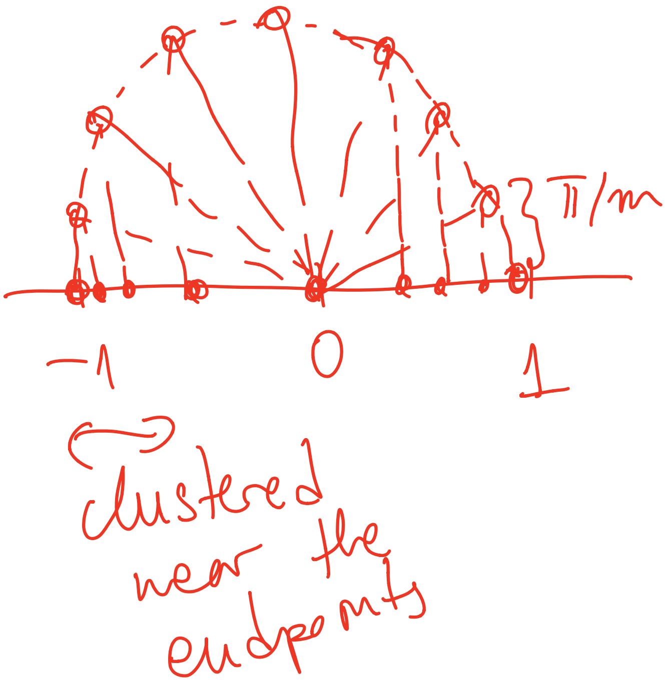
  which have a simple geometric interpretation as the projection of uniformly spaced points on the unit circle.
- Polynomial interpolation using the Chebyshev nodes **eliminates Runge's phenomenon**.
- Furthermore, such polynomial interpolation gives **spectral accuracy**, which approximately means that for **sufficiently smooth functions** the error decays **exponentially in the number of points**, faster than any power law (fixed order of accuracy).

later

- There are very fast and robust numerical methods to actually perform the interpolation (function approximation) on Chebyshev nodes, see for example the package **chebfun** from Nick Trefethen.

# Outline





# Interpolation in 1D (Cleve Moler)

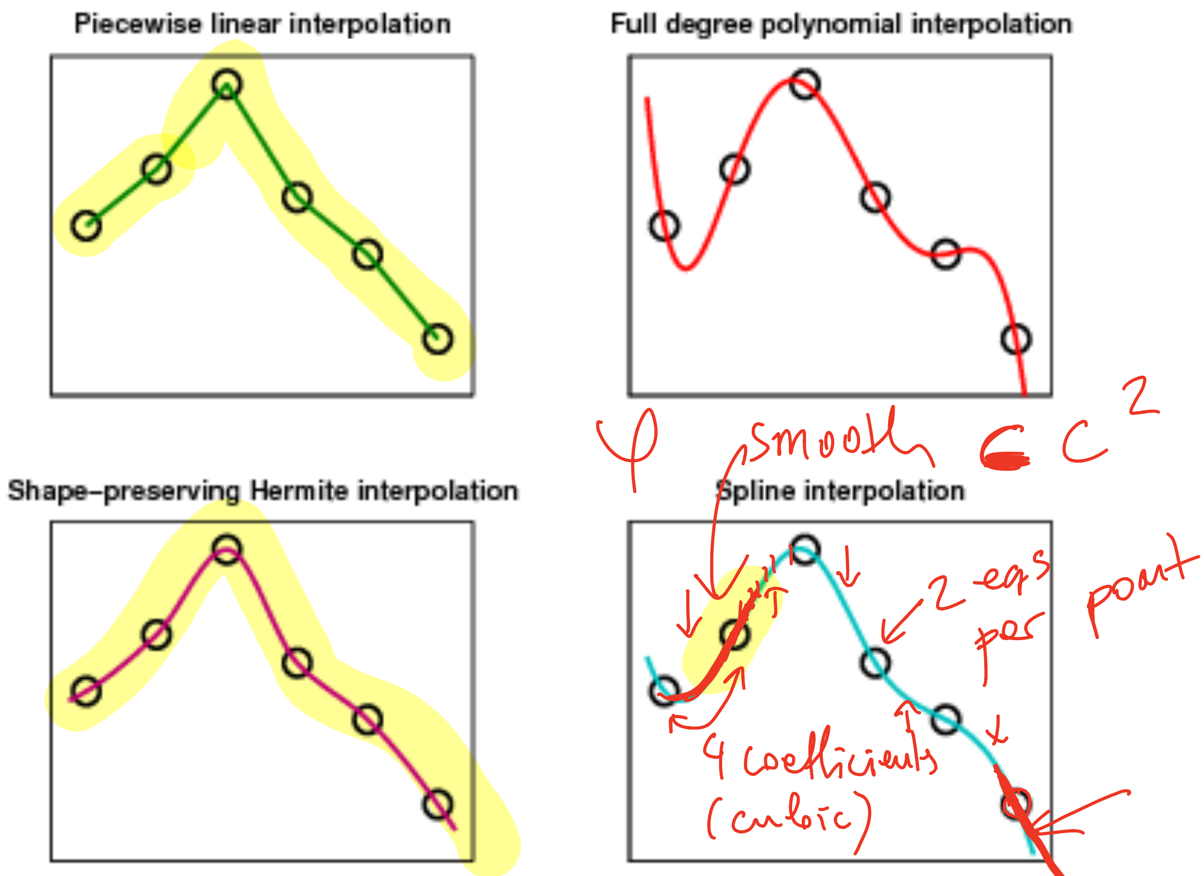
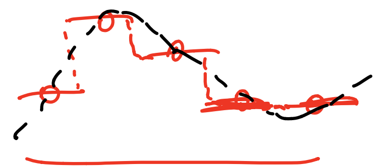



**Figure 3.8.** *Four interpolants.*

# Piecewise interpolants

- The idea is to use a **different low-degree polynomial** function $\phi_i(x)$ in each interval $I_i = [x_i, x_{i+1}]$.
- **Piecewise-constant** interpolation: $\phi_i^{(0)}(x) = y_i$, which is **first-order accurate**:

$$\left\| f(x) - \phi^{(0)}(x) \right\|_\infty \leq h \left\| f^{(1)}(x) \right\|_\infty$$



- **Piecewise-linear** interpolation:

$$\phi_i^{(1)}(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \text{ for } x \in I_i$$

For node spacing $h$ the error estimate is now bounded but only **second-order accurate**

$$\left\| f(x) - \phi^{(1)}(x) \right\|_\infty \leq \frac{h^2}{8} \left\| f^{(2)}(x) \right\|_\infty$$

# Cubic Splines

- One can think about **piecewise-quadratic** interpolants but even better are **piecewise-cubic** interpolants.
- Going after **twice continuously-differentiable** interpolant, $\phi(x) \in C_I^2$, leads us to **cubic spline interpolation**:
  - The function $\phi_i(x)$ is **cubic** in each interval $I_i = [x_i, x_{i+1}]$ (requires $4m$ coefficients).
  - We **interpolate** the function at the nodes: $\phi_i(x_i) = \phi_{i-1}(x_i) = y_i$. 2 eqs
    This gives $m+1$ conditions plus $m-1$ conditions at **interior nodes**.
  - The **first and second derivatives are continuous** at the interior nodes:

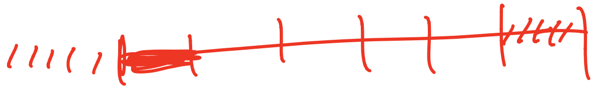
2 eqs
$$\phi_i'(x_i) = \phi_{i-1}'(x_i) \text{ and } \phi_i''(x_i) = \phi_{i-1}''(x_i) \text{ for } i = 1, 2, \ldots, m-1,$$

which gives $2(m-1)$ equations.

equation

- Now we have $(m+1) + (m-1) + 2(m-1) = \boxed{4m-2}$ conditions for $4m$ unknowns.

# Types of Splines



- We need to specify two more conditions arbitrarily (for splines of order $k \geq 3$, there are $k-1$ arbitrary conditions).
- The most appropriate choice depends on the problem, e.g.:
  - **Periodic** splines, we think of node 0 and node $m$ as one interior node and add the two conditions:

$$\phi_0'(x_0) = \phi_m'(x_m) \text{ and } \phi_0''(x_0) = \phi_m''(x_m).$$

  - **Natural** spline: Two conditions $\phi''(x_0) = \phi''(x_m) = 0$.
- Once the type of spline is chosen, finding the coefficients of the cubic polynomials requires solving a **sparse tridiagonal linear system**, which can be done very fast ($O(m)$).

# Nice properties of splines

fourth-order interpolation

- The spline approximation converges for zeroth, first and second derivatives and even third derivatives (for equi-spaced nodes):

$$\|f(x) - \phi(x)\|_\infty \le \frac{5}{384} \cdot h^4 \cdot \left\|f^{(4)}(x)\right\|_\infty$$

$$\|f'(x) - \phi'(x)\|_\infty \le \frac{1}{24} \cdot h^3 \cdot \left\|f^{(4)}(x)\right\|_\infty$$

$$\|f''(x) - \phi''(x)\|_\infty \le \frac{3}{8} \cdot h^2 \cdot \left\|f^{(4)}(x)\right\|_\infty$$

- We see that cubic spline interpolants are **fourth-order accurate** for functions. For **each derivative** we **loose one order of accuracy** (this is typical of all interpolants).

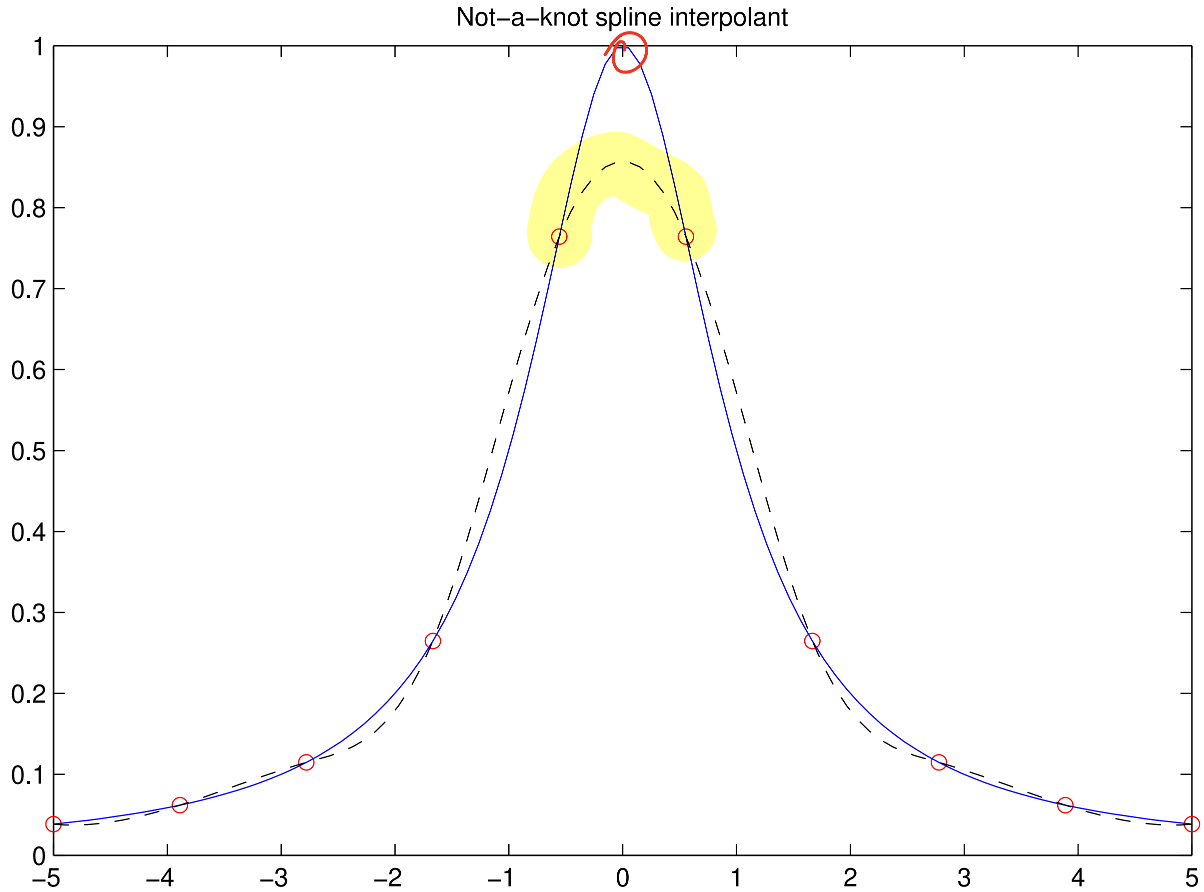
# In MATLAB

- $c = polyfit(x, y, n)$ does least-squares polynomial of degree $n$ which is interpolating if $n = length(x)$.
- Note that MATLAB stores the coefficients in reverse order, i.e., $c(1)$ is the coefficient of $x^n$.
- $y = polyval(c, x)$ evaluates the interpolant at new points.
- $y1 = interp1(x, y, x_{new}, 'method')$ or if $x$ is ordered use $interp1q$. Method is one of 'linear', 'spline', 'cubic'.
- The actual piecewise polynomial can be obtained and evaluated using $ppval$.

# Interpolating $(1+x^2)^{-1}$ in MATLAB

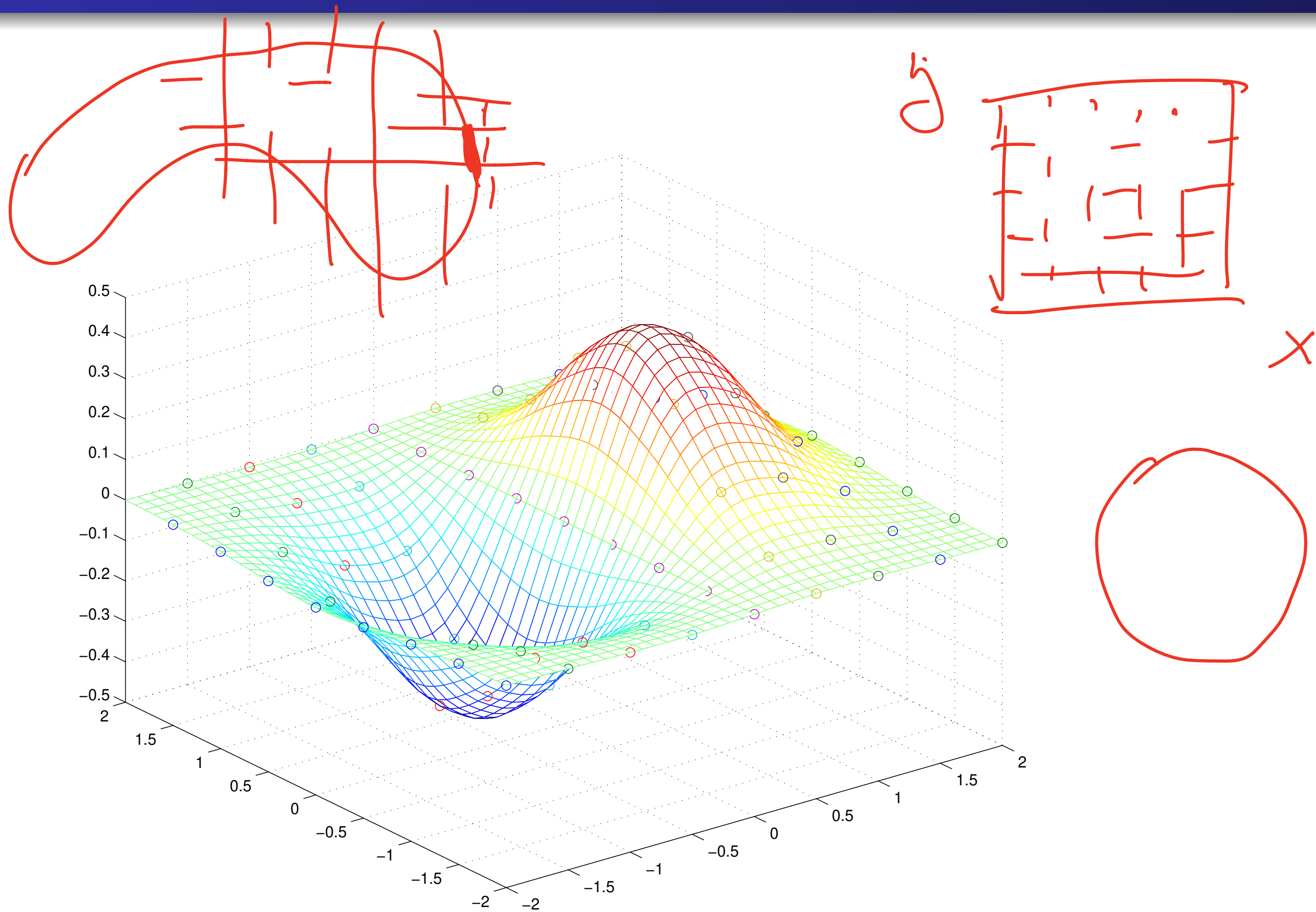
```
n=10;
x=linspace(-5,5,n);
y=(1+x.^2).^(-1);
plot(x,y,'ro'); hold on;

x_fine=linspace(-5,5,100);
y_fine=(1+x_fine.^2).^(-1);
plot(x_fine,y_fine,'b-');

c=polyfit(x,y,n);
y_interp=polyval(c,x_fine);
plot(x_fine,y_interp,'k--');

y_interp=interp1(x,y,x_fine,'spline');
% Or: pp=spline(x,y); y_interp=ppval(pp,x_fine)
plot(x_fine,y_interp,'k--');
```

# Runge's function with spline



# Outline

# Two Dimensions



# Regular grids

- Now $\mathbf{x} = \{x_1, \dots, x_n\} \in \mathbf{R}^n$ is a multidimensional data point. Focus on **two-dimensions** (2D) since **three-dimensions** (3D) is similar.
- The easiest case is when the data points are all inside a **rectangle**

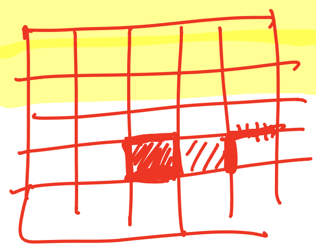
$$\Omega = [x_0, x_{m_x}] \times [y_0, y_{m_y}]$$

where the $m = (m_x + 1)(m_y + 1)$ nodes lie on a **regular grid**

$$\mathbf{x}_{i,j} = \{x_i, y_j\}, \quad f_{i,j} = f(\mathbf{x}_{i,j}).$$

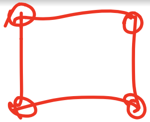
- Just as in 1D, one can use a different interpolation function $\phi_{i,j} : \Omega_{i,j} \to \mathbb{R}$ in each rectangle of the grid (pixel)

$$\Omega_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}].$$



# Bilinear Interpolation

$d + cy + bx$



Equivalent of piecewise linear in 1D

- The equivalent of piecewise linear interpolation for 1D in 2D is the **piecewise bilinear interpolation**

$$\phi_{i,j}(x, y) = (\alpha x + \beta)(\gamma y + \delta) = a_{i,j}xy + b_{i,j}x + c_{i,j}y + d_{i,j}.$$

- There are 4 unknown coefficients in $\phi_{i,j}$ that can be found from the 4 data (function) values at the corners of rectangle $\Omega_{i,j}$. This requires solving a small $4 \times 4$ linear system inside each pixel independently.
- Note that the pieces of the interpolating function $\phi_{i,j}(x, y)$ are **not linear** (but also **not quadratic** since no $x^2$ or $y^2$) since they contain quadratic product terms $xy$: **bilinear functions**.
  This is because there is not a plane that passes through 4 generic points in 3D.

# Piecewise-Polynomial Interpolation

- The key distinction about **regular grids** is that we can use **separable basis** functions:

$$\phi_{i,j}(\mathbf{x}) = \phi_i(x)\phi_j(y).$$

- Furthermore, it is sufficient to look at a **unit reference rectangle** $\hat{\Omega} = [0,1] \times [0,1]$ since any other rectangle or even **parallelogram** can be obtained from the reference one via a linear transformation.
- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

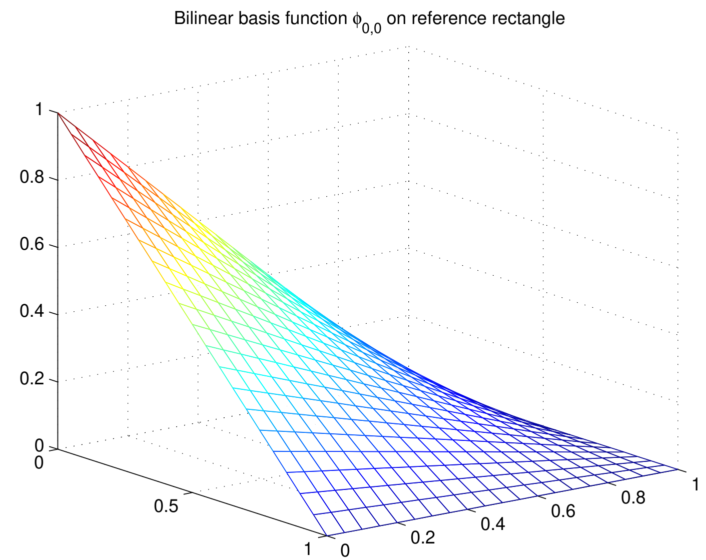
$$\hat{\phi}_{0,0}(\hat{x},\hat{y}) = (1-\hat{x})(1-\hat{y})$$

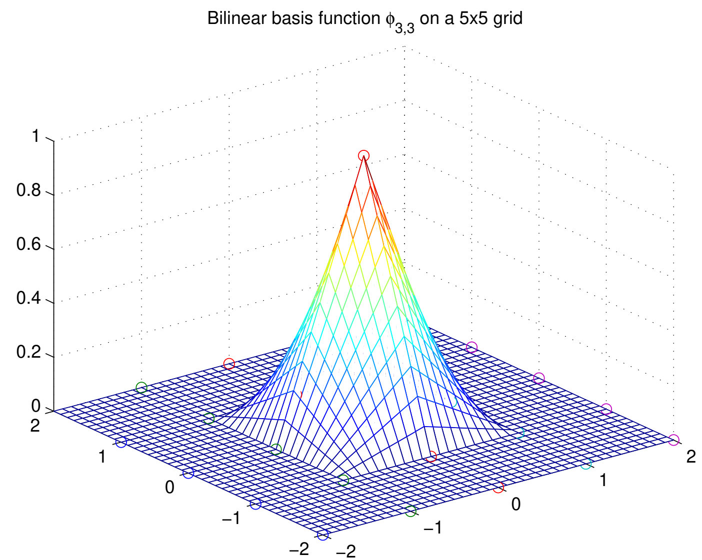
- Generalization of bilinear to 3D is **trilinear interpolation**

$$\phi_{i,j,k} = a_{i,j,k}xyz + b_{i,j,k}xy + c_{i,j,k}xz + d_{i,j,k}yz + e_{i,j,k}x + f_{i,j,k}y + g_{i,j,k}z + h_{i,j,k}$$

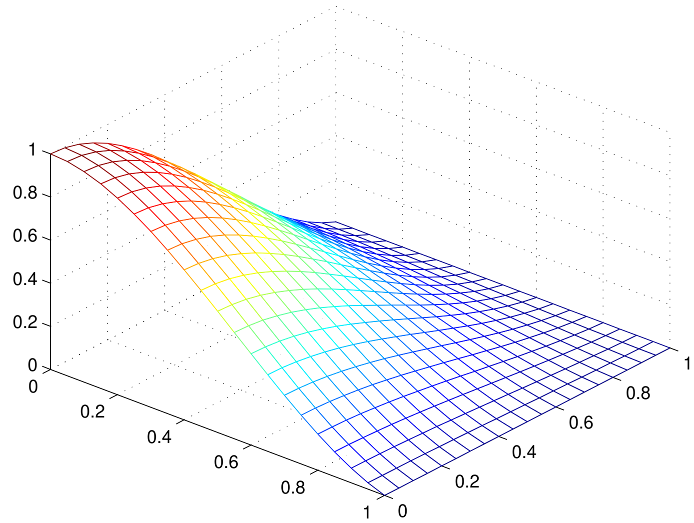
which has 8 coefficients which can be solved for given the 8 values at the vertices of the cube.

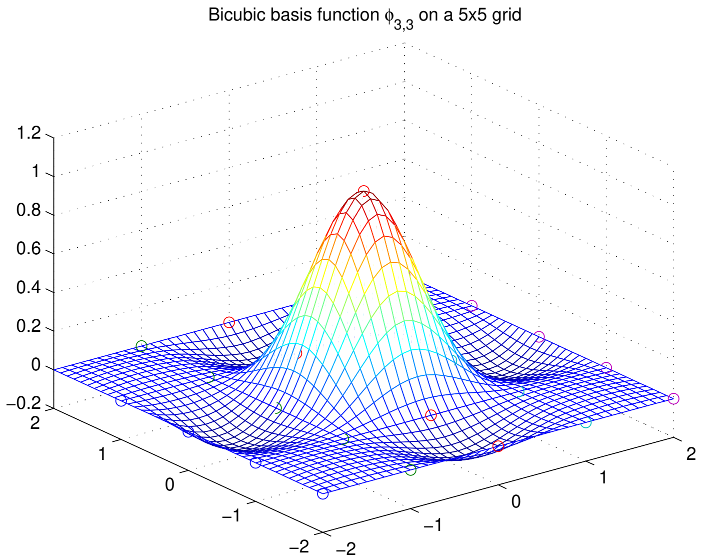
# Bilinear basis functions



# Bicubic basis functions

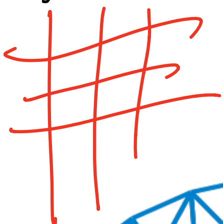


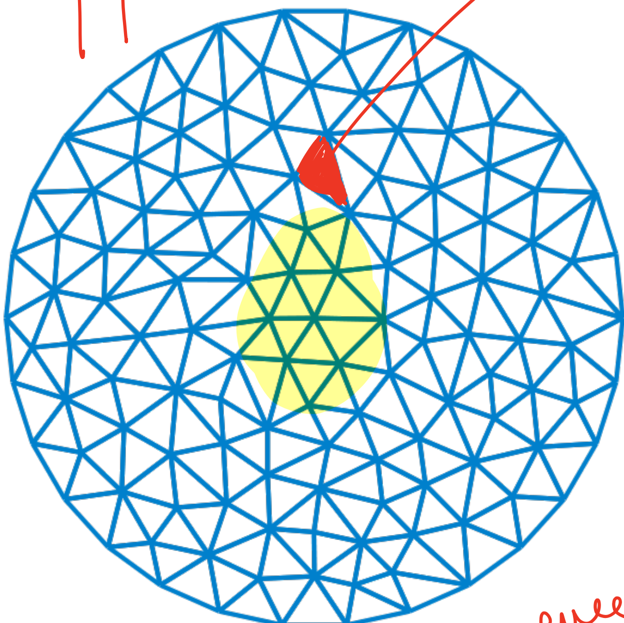



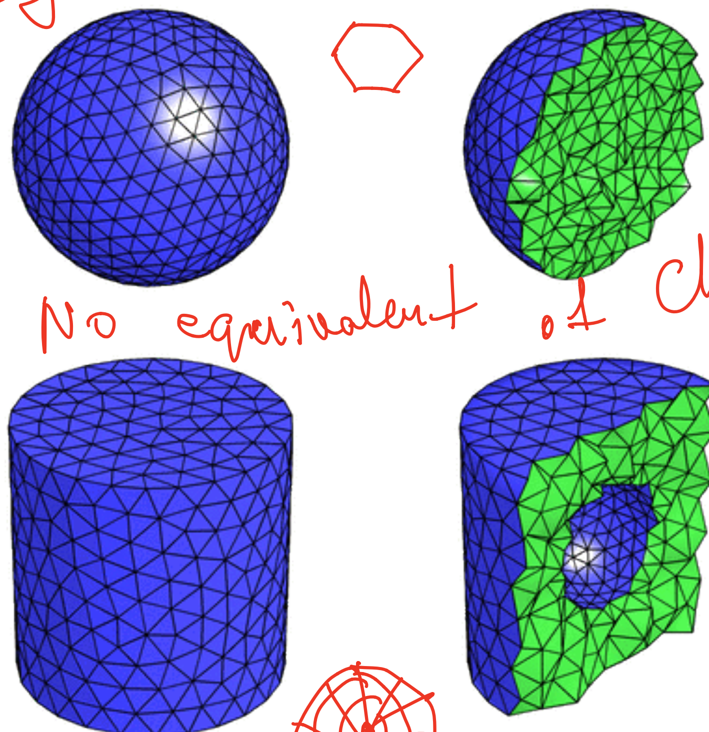
# Irregular (Simplicial) Meshes

Any polygon can be triangulated into arbitrarily many **disjoint triangles**.
Similarly **tetrahedral meshes** in 3D.

a + cy + dx piecewise linear





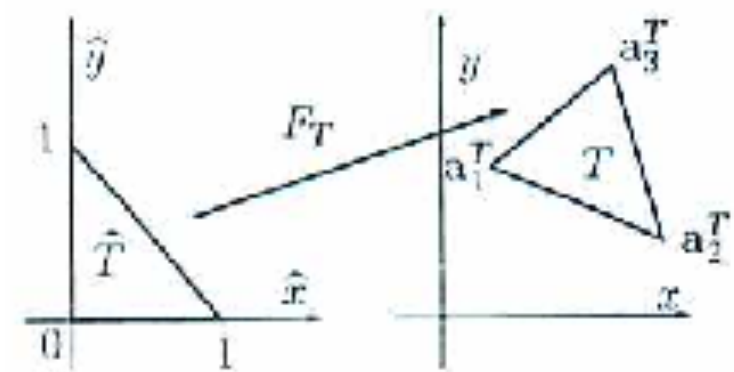


No equivalent of Chebyshev

Computer science

# Basis functions on triangles

- For irregular grids the $x$ and $y$ directions are no longer separable.
- But the idea of using basis functions $\phi_{i,j}$, a **reference triangle**, and **piecewise polynomial interpolants** still applies.
- For a piecewise constant function we need one coefficient per triangle, for a linear function we need 3 coefficients $(x, y, \text{const})$, for quadratic 6 $(x, y, x^2, y^2, xy, \text{const})$, so we choose the **reference nodes**:



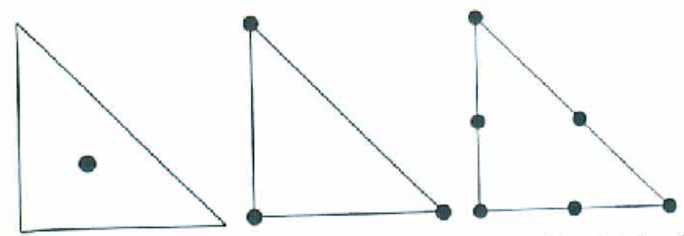



Fig. 8.8. Local interpolation nodes on $\hat{T}$ for $k = 0$ (*left*), $k = 1$ (*center*), $k = 2$ (*right*)

# In MATLAB

- For regular grids the function

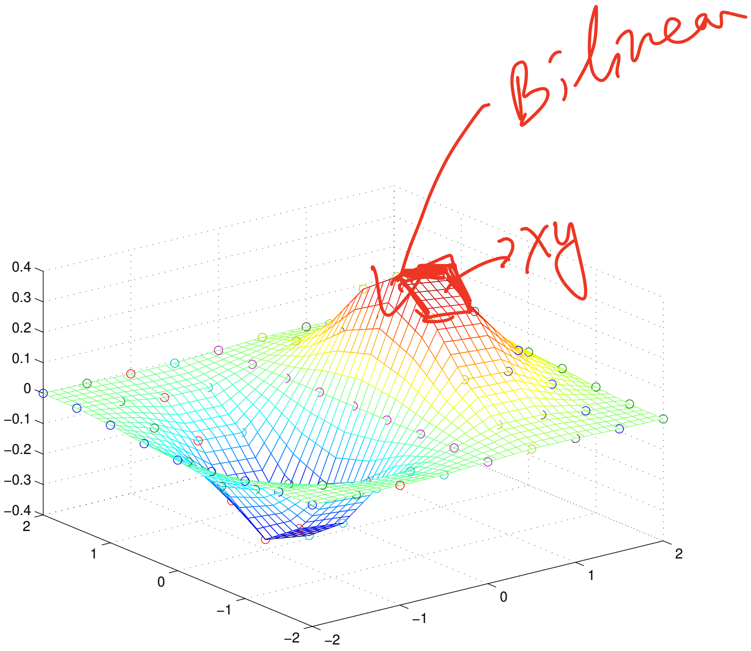
$$qz = interp2(x, y, z, qx, qy, 'linear')$$

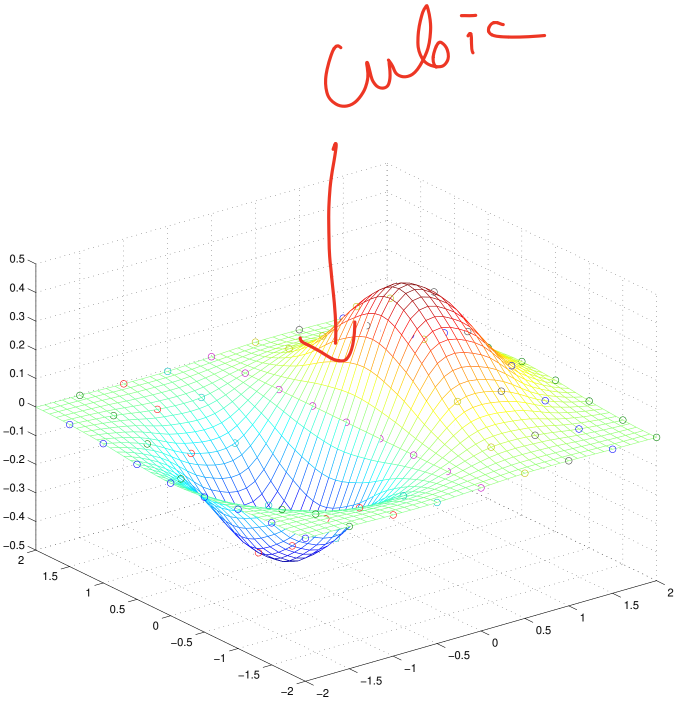
  will evaluate the piecewise bilinear interpolant of the data $x, y, z = f(x, y)$ at the points $(qx, qy)$.
- Other method are 'spline' and 'cubic', and there is also *interp*3 for 3D.
- For irregular grids one can use the old function *griddata* which will generate its own triangulation or there are more sophisticated routines to manipulate triangulations also.

# Regular grids

```
[x,y] = meshgrid(-2:.5:2, -2:.5:2);
z = x.*exp(-x.^2-y.^2);

ti = -2:.1:2;
[qx,qy] = meshgrid(ti,ti);

qz= interp2(x,y,z,qx,qy,'cubic');

mesh(qx,qy,qz); hold on;
plot3(x,y,z,'o'); hold off;
```

# MATLAB's *interp2*

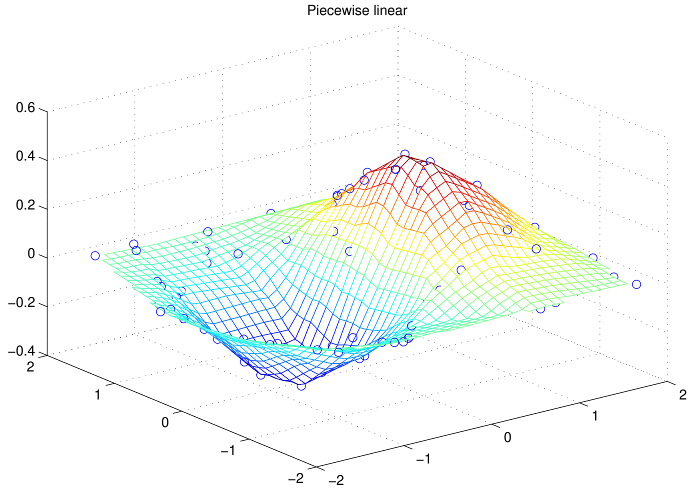


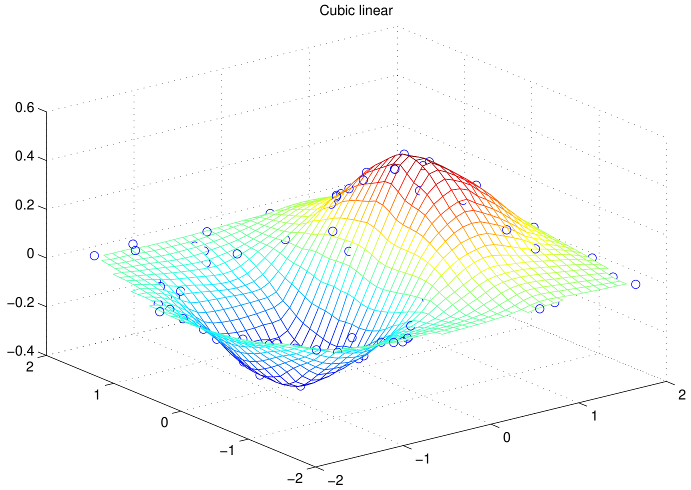



# Irregular grids

```
x = rand(100,1)*4-2; y = rand(100,1)*4-2;
z = x.*exp(-x.^2-y.^2);

ti = -2:.1:2;
[qx,qy] = meshgrid(ti,ti);

qz= griddata(x,y,z,qx,qy,'cubic');

mesh(qx,qy,qz); hold on;
plot3(x,y,z,'o'); hold off;
```

# MATLAB's *griddata*



# Outline




FFT trig

$\sin(n\pi\varphi)$

# Advanced optional material: Orthogonal Polynomials

- Any finite interval $[a, b]$ can be transformed to $I = [-1, 1]$ by a simple transformation.
- Using a **weight function** $w(x)$, define a **function dot product** as:

$$(f, g) = \int_a^b w(x) \left[ f(x) g(x) \right] dx$$

- For different choices of the weight $w(x)$, one can explicitly construct **basis of orthogonal polynomials** where $\phi_k(x)$ is a polynomial of degree $k$ (**triangular basis**):

$$(\phi_i, \phi_j) = \int_a^b w(x) \left[ \phi_i(x) \phi_j(x) \right] dx = \delta_{ij} \left\| \phi_i \right\|^2 .$$

- For **Chebyshev polynomials** we set $w = (1 - x^2)^{-1/2}$ and this gives

$$\phi_k(x) = \cos\left( k \arccos x \right).$$

# Legendre Polynomials

- For equal weighting $w(x) = 1$, the resulting triangular family of of polynomials are called **Legendre polynomials**:

$$
\begin{aligned}
\phi_0(x) &= 1 \\
\phi_1(x) &= x \\
\phi_2(x) &= \frac{1}{2}(3x^2 - 1) \\
\phi_3(x) &= \frac{1}{2}(5x^3 - 3x) \\
\phi_{k+1}(x) &= \frac{2k+1}{k+1} x\phi_k(x) - \frac{k}{k+1}\phi_{k-1}(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2 - 1\right)^n\right]
\end{aligned}
$$

- These are orthogonal on $I = [-1, 1]$:

$$
\int_{-1}^{-1} \phi_i(x)\phi_j(x)dx = \delta_{ij} \cdot \frac{2}{2i+1}.
$$

# Interpolation using Orthogonal Polynomials

- Let's look at the **interpolating polynomial** $\phi(x)$ of a function $f(x)$ on a set of $m+1$ **nodes** $\{x_0, \ldots, x_m\} \in I$, expressed in an orthogonal basis:

$$\phi(x) = \sum_{i=0}^{m} a_i \phi_i(x)$$

- Due to orthogonality, taking a dot product with $\phi_j$ (**weak formulation**):

$$(\phi, \phi_j) = \sum_{i=0}^{m} a_i (\phi_i, \phi_j) = \sum_{i=0}^{m} a_i \delta_{ij} \|\phi_i\|^2 = a_j \|\phi_j\|^2$$

- This is **equivalent to normal equations** if we use the right dot product:

$$(\boldsymbol{\Phi}^\star \boldsymbol{\Phi})_{ij} = (\phi_i, \phi_j) = \delta_{ij} \|\phi_i\|^2 \text{ and } \boldsymbol{\Phi}^\star \mathbf{y} = (\phi, \phi_j)$$

# Gauss Integration

$$a_j \left\|\phi_j\right\|^2 = (\phi, \phi_j) \quad \Rightarrow \quad a_j = \left(\left\|\phi_j\right\|^2\right)^{-1} (\phi, \phi_j)$$

- Question: Can we easily compute

$$(\phi, \phi_j) = \int_a^b w(x) \left[\phi(x)\phi_j(x)\right] dx = \int_a^b w(x) p_{2m}(x) dx$$

for a polynomial $p_{2m}(x) = \phi(x)\phi_j(x)$ of degree at most $2m$?

# Gauss nodes

- If we choose the **nodes to be zeros of** $\phi_{m+1}(x)$, then we can **quickly project any polynomial** onto the basis of orthogonal polynomials:

$$(\phi, \phi_j) = \sum_{i=0}^{m} w_i \phi(x_i) \phi_j(x_i) = \sum_{i=0}^{m} w_i f(x_i) \phi_j(x_i)$$

where the **Gauss weights w** are given by

$$w_i = \int_a^b w(x) \phi_i(x) dx.$$

- The orthogonality relation can be expressed as a **sum instead of integral**:

$$(\phi_i, \phi_j) = \sum_{i=0}^{m} w_i \phi_i(x_i) \phi_j(x_i) = \delta_{ij} \left\| \phi_i \right\|^2$$

# Gauss-Legendre polynomials

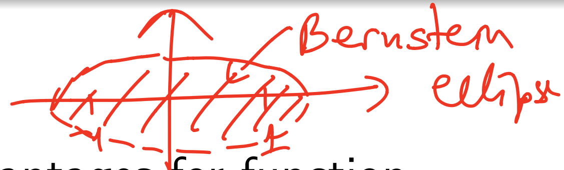
- For any weighting function the polynomial $\phi_k(x)$ has $k$ simple zeros all of which are in $(-1, 1)$, called the (order $k$) **Gauss nodes**, $\phi_{m+1}(x_i) = 0$.
Chebyshev nodes
- The interpolating polynomial $\phi(x_i) = f(x_i)$ on the Gauss nodes is the **Gauss-Legendre interpolant** $\phi_{GL}(x)$.
Chebyshev
- We can thus define a new weighted **discrete dot product**

$$\mathbf{f} \cdot \mathbf{g} = \sum_{i=0}^{m} w_i f_i g_i$$

The Gauss-Legendre interpolant is thus easy to compute:

$$\phi_{GL}(x) = \sum_{i=0}^{m} \frac{\mathbf{f} \cdot \phi_i}{\phi_i \cdot \phi_i} \phi_i(x).$$

# Discrete spectral approximation

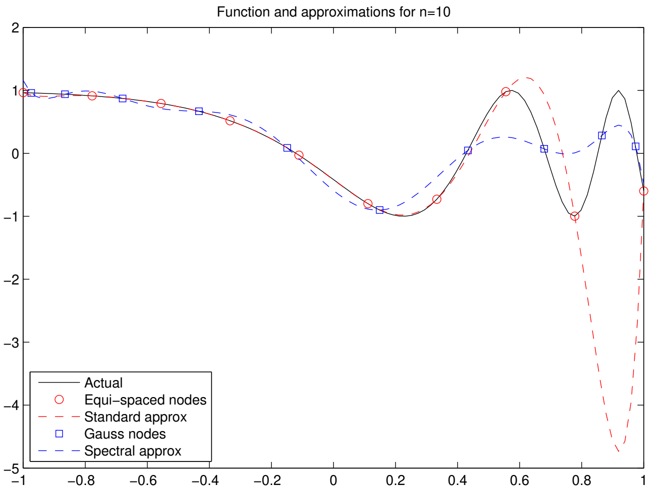


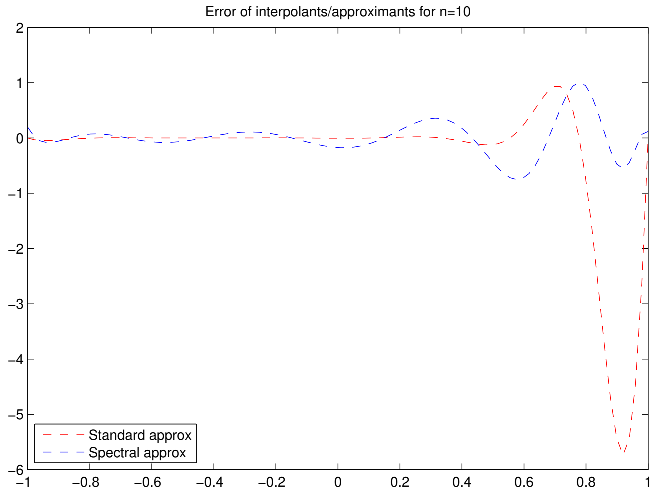

- Using orthogonal polynomails has many advantages for function approximation: **stability**, **rapid convergence**, and **computational efficiency**.
- The convergence, for sufficiently smooth (nice) functions (analytic in the neighborhood of $[-1,1]$ in the complex plane), is **more rapid than any power law**

$$\|f(x) - \phi_{GL}(x)\| \sim C^{-m},$$

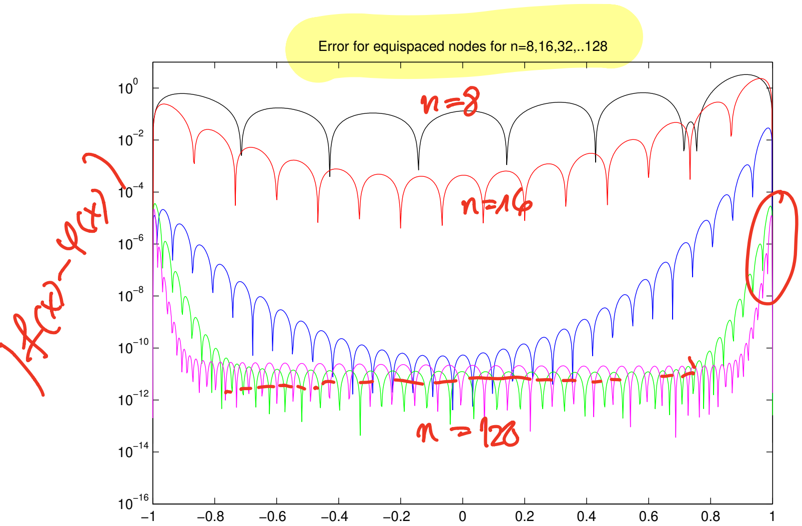
- This so-called **spectral accuracy** (limited by smoothness only) cannot be achived by piecewise, i.e., local, approximations (limited by order of local approximation).

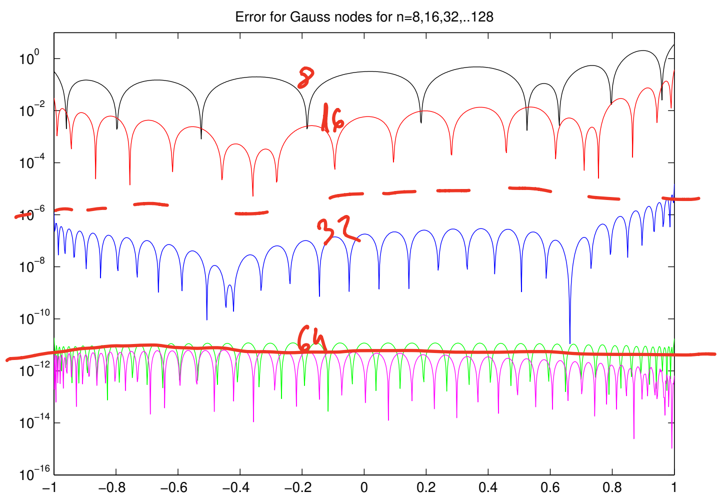
# Gauss-Legendre Interpolation



# Global polynomial interpolation error



# Conclusions/Summary

- Interpolation means approximating function values in the interior of a domain when there are **known samples** of the function at a set of **interior and boundary nodes**.
- Given a **basis set** for the **interpolating functions**, interpolation amounts to solving a linear system for the coefficients of the basis functions.
- Polynomial interpolants in 1D can be constructed using several basis.
- Using polynomial interpolants of **high order is a bad idea**: Not accurate and not stable!
- Instead, it is better to use **piecewise polynomial** interpolation: constant, linear, Hermite cubic, cubic spline interpolant on each **interval**.
- In higher dimensions one must be more careful about how the domain is split into disjoint **elements** (analogues of intervals in 1D): **regular grids** (separable basis such as bilinear), or **simplicial meshes** (triangular or tetrahedral).