Numerical Methods I
Polynomial Interpolation

Aleksandar Donev
Courant Institute, NYU

donev@courant.nyu.edu

\footnote{MATH-GA 2011.003 / CSCI-GA 2945.003, Fall 2014}

October 30th, 2014
1. Function spaces

2. Polynomial Interpolation in 1D

3. Piecewise Polynomial Interpolation

4. Higher Dimensions
Function spaces are the equivalent of finite vector spaces for functions (space of polynomial functions $\mathcal{P}$, space of smoothly twice-differentiable functions $C^2$, 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**: $\|f(x)\|_w = \left[ \int_a^b |f(x)|^2 w(x) \, dx \right]^{1/2}$

An **inner or scalar product** (equivalent of dot product for vectors):

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

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

which is equivalent to thinking of the function as being the vector

\[
f_{\mathcal{X}} = y = \{f(x_0), f(x_1), \ldots, f(x_m)\}.
\]

**Finite representations** lead to **semi-norms**, but this is not that important.

**A discrete dot product** can be just the vector product:

\[
(f, g)^{\mathcal{X}} = f_{\mathcal{X}} \cdot g_{\mathcal{X}} = \sum_{i=0}^{m} f(x_i) g^*(x_i)
\]
Think of a function as a vector of coefficients in terms of a set of $n$ basis functions:

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

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

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

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

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

$$c^* = \text{arg min}_c \left\| f(x) - \tilde{f}(x) \right\|_2 ,$$

which gives the orthogonal projection of $f(x)$ onto the finite-dimensional basis.
Discrete case: Think of fitting a straight line or quadratic through experimental data points.

The function becomes the vector $y = f(x)$, and the approximation is

$$y_i = \sum_{j=1}^{n} c_j \phi_j(x_i) \Rightarrow y = \Phi c,$$

where $\Phi_{ij} = \phi_j(x_i)$.

This means that finding the approximation consists of solving an overdetermined linear system

$$\Phi c = y.$$ 

Note that for $m = n$ this is equivalent to interpolation. MATLAB’s `polyfit` works for $m \geq n$. 

Recall that one way to solve this is via the normal equations:

\[(\Phi^* \Phi) c^* = \Phi^* y\]

A basis set is an **orthonormal basis** if

\[
(\phi_i, \phi_j) = \sum_{k=0}^{m} \phi_i(x_k)\phi_j(x_k) = \delta_{ij} = \begin{cases} 
1 & \text{if } i = j \\
0 & \text{if } i \neq j
\end{cases}
\]

\[\Phi^* \Phi = I \text{ (unitary or orthogonal matrix)} \Rightarrow \]

\[c^* = \Phi^* y \Rightarrow c_i = \phi_i^\chi \cdot f^\chi = \sum_{k=0}^{m} f(x_k)\phi_i(x_k)\]
Interpolation in 1D (Cleve Moler)

Figure 3.8. *Four interpolants.*
The task of interpolation is to find an interpolating function $\phi(x)$ which passes through $m + 1$ data points $(x_i, y_i)$:

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

where $x_i$ are given nodes.

The type of interpolation is classified based on the form of $\phi(x)$:

- Full-degree **polynomial** interpolation if $\phi(x)$ is globally polynomial.
- **Piecewise polynomial** if $\phi(x)$ is a collection of local polynomials:
  - Piecewise linear or quadratic
  - **Hermite** interpolation
  - **Spline** interpolation
- **Trigonometric** if $\phi(x)$ is a trigonometric polynomial (polynomial of sines and cosines).
- **Orthogonal polynomial** interpolation (Chebyshev, Legendre, etc.).

As for root finding, in dimensions higher than one things are more complicated!
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 $a = \{a_1, \ldots, a_m\}$ are solutions to the square linear system:

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

In matrix notation, if we start indexing at zero:

$$
[V(x_0, x_1, \ldots, x_m)] a = y
$$

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

$$
v_{i,j} = x_i^j.
$$
The Vandermonde approach

\[ V a = x \]

One can prove by induction that

\[ \det V = \prod_{j<k} (x_k - x_j) \]

which means that the Vandermonde system is non-singular and thus: The interpolating polynomial is **unique if the nodes are distinct**.

- Polynomial interpolation is thus equivalent to solving a linear system.
- However, it is easily seen that the Vandermonde matrix can be very **ill-conditioned**.
- 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:

\[ x^2 - 2x + 4 = (x - 2)^2. \]

- 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

\[ \phi(x_j) = \sum_{i=0}^{m} a_i \phi_i(x_j) = y_j \quad \Rightarrow \quad \Phi a = y \]
Lagrange basis

\[ \Phi a = y \]

- This linear system will be trivial to solve if \( \Phi = I \), i.e., if

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

- The \( \phi_i(x) \) is itself a polynomial interpolant on the same nodes but with function values \( \delta_{ij} \), and is thus unique.

- Note that the nodal polynomial

\[ w_{m+1}(x) = \prod_{i=0}^{m} (x - x_i) \]

vanishes at all of the nodes but has degree \( m + 1 \).
It can easily be seen that the following characteristic polynomial provides the desired basis:

$$\phi_i(x) = \frac{\prod_{j \neq i} (x - x_j)}{\prod_{j \neq i} (x_i - x_j)} = \frac{w_{m+1}(x)}{(x - x_i)w'_{m+1}(x_i)}$$

The resulting Lagrange interpolation formula is

$$\phi(x) = \sum_{i=0}^{m} y_i \phi_i(x) = \sum_{i=0}^{m} \left[ \frac{y_i}{\prod_{j \neq i} (x_i - x_j)} \right] \prod_{j \neq i} (x - x_j)$$

This is useful analytically but expensive and cumbersome to use computationally!
Lagrange basis on 10 nodes

A few Lagrange basis functions for 10 nodes
Newton’s interpolation formula

- By choosing a different basis we get different representations, and Newton’s choice is:

\[ \phi_i(x) = w_i(x) = \prod_{j=0}^{i-1} (x - x_j) \]

- There is a simple recursive formula to calculate the coefficients \( a \) in this basis, using Newton’s **divided differences**

\[ D^0_i f = f(x_i) = y_i \]

\[ D^k_i = \frac{D^{k-1}_{i+1} - D^{k-1}_i}{x_{i+1} - x_i} \]

- Note that the first divided difference is

\[ D^1_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \approx f'(x_i) \]

and \( D^2_i \) corresponds to second-order derivatives, etc.
Convergence and stability

- 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 is not hard to show that

$$\exists \xi \in I \text{ such that } E_m(x) = f(x) - \phi(x) = \frac{f^{(m+1)}(\xi)}{(m+1)!} \left[ \prod_{i=0}^{m} (x - x_i) \right].$$

Question: Does $\|E_m(x)\|_\infty = \max_{x \in I} |f(x)| \to 0$ as $m \to \infty$.

- For equi-spaced nodes, $x_{i+1} = x_i + h$, a bound is

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

- The problem is that higher-order derivatives of seemingly nice functions can be unbounded!
Runge’s counter-example: \( f(x) = (1 + x^2)^{-1} \)
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 \( \tilde{y} = y + \delta y \),

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

- It is possible to improve the situation by using specially-chosen nodes (e.g., Chebyshev nodes), or by interpolating derivatives (Hermite interpolation).
- In general however, we conclude that interpolating using high-degree polynomials is a bad idea!
Figure 3.8. *Four 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: $\phi_i^{(0)}(x) = y_i$.

**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 and stable:

$$\left\| f(x) - \phi_i^{(1)}(x) \right\|_\infty \leq \frac{h^2}{8} \left\| f^{(2)}(x) \right\|_\infty$$
If we are given not just the function values but also the first derivatives at the nodes:

\[ z_i = f'(x_i), \]

we can find a cubic polynomial on every interval that interpolates both the function and the derivatives at the endpoints:

\[ \phi_i(x_i) = y_i \text{ and } \phi'_i(x_i) = z_i \]

\[ \phi_i(x_{i+1}) = y_{i+1} \text{ and } \phi'_i(x_{i+1}) = z_{i+1}. \]

This is called the **piecewise cubic Hermite interpolant**.

If the derivatives are not available we can try to estimate \( z_i \approx \phi'_i(x_i) \) (see MATLAB’s *pchip*).
Note that in piecewise Hermite interpolation $\phi(x)$ has is continuously differentiable, $\phi(x) \in C^1_I$:
Both $\phi(x)$ and $\phi'(x)$ are continuous across the internal nodes.

We can make this even stronger, $\phi(x) \in C^2_I$, leading to piecewise 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: $\phi_i(x_i) = \phi_{i-1}(x_i) = y_i$.
  This gives $m + 1$ conditions plus $m - 1$ conditions at interior nodes.
- The first and second derivatives are continuous at the interior nodes:
  
  $$\phi'_i(x_i) = \phi'_{i-1}(x_i) \quad \text{and} \quad \phi''_i(x_i) = \phi''_{i-1}(x_i) \quad \text{for} \quad i = 1, 2, \ldots, m - 1,$$

  which gives $2(m - 1)$ equations, for a total of $4m - 2$ conditions.
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:
    \[
    \phi'(x_0) = \phi'(x_m) \quad \text{and} \quad \phi''(x_0) = \phi''(x_m)
    \]
  - **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 **tridiagonal linear system**, which can be done very fast ($O(m)$).
Nice properties of splines

- Minimum curvature property:

\[ \int_I [\phi''(x)]^2 \, dx \leq \int_I [f''(x)]^2 \, dx \]

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

\[ \| f(x) - \phi(x) \|_\infty \leq \frac{5}{384} \cdot h^4 \cdot \| f^{(4)}(x) \|_\infty \]

\[ \| f'(x) - \phi'(x) \|_\infty \leq \frac{1}{24} \cdot h^3 \cdot \| f^{(4)}(x) \|_\infty \]

\[ \| f''(x) - \phi''(x) \|_\infty \leq \frac{3}{8} \cdot h^2 \cdot \| f^{(4)}(x) \|_\infty \]
In MATLAB

- \( c = \text{polyfit}(x, y, n) \) does least-squares polynomial of degree \( n \) which is interpolating if \( n = \text{length}(x) \).
- Note that MATLAB stores the coefficients in reverse order, i.e., \( c(1) \) is the coefficient of \( x^n \).
- \( y = \text{polyval}(c, x) \) evaluates the interpolant at new points.
- \( y1 = \text{interp1}(x, y, x_{\text{new}}, 'method') \) or if \( x \) is ordered use \( \text{interp1q} \). Method is one of 'linear', 'spline', 'cubic'.
- The actual piecewise polynomial can be obtained and evaluated using \( \text{ppval} \).
Interpolating $(1 + x^2)^{-1}$ in MATLAB

```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');
plot(x_fine,y_interp,'k−−');

% Or equivalently:
pp=spline(x,y);
y_interp=ppval(pp,x_fine)
```
Runge’s function with spline interpolation
Two Dimensions
Now $\mathbf{x} = \{x_1, \ldots, x_n\} \in \mathbb{R}^n$ is a multidimensional data point. Focus on 2D since 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

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

We can use separable basis functions:

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

We can directly apply Lagrange interpolation to each coordinate separately:

\[
\phi(x) = \sum_{i,j} f_{i,j} \phi_{i,j}(x, y) = \sum_{i,j} f_{i,j} \phi_i(x) \phi_j(y),
\]

but this still suffers from Runge’s phenomenon:
Piecewise-Polynomial Interpolation

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

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

- For separable polynomials, the equivalent of piecewise linear interpolation in 1D is the **piecewise bilinear interpolation**

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

- There are 4 unknown coefficients in $\phi_{i,j}$ that can be found from the 4 data (function) values at the corners of rectangle $\Omega_{i,j}$.

- 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.
Higher Dimensions

Bilinear Interpolation

- It is better to think in terms of a basis set \( \{ \phi_{i,j}(x, y) \} \), where each basis function \( \phi_{i,j} \) is itself piecewise bilinear, one at the node \((i,j)\)-th node of the grid, zero elsewhere:

\[
\phi(x) = \sum_{i,j} f_{i,j} \phi_{i,j}(x, y).
\]

- 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:

\[
B_{i,j} \hat{\Omega} + b_{i,j} = \Omega_{i,j},
\]

and the same transformation can then be applied to the interpolation function:

\[
\phi_{i,j}(x) = \hat{\phi}(B_{i,j}\hat{x} + b_{i,j}).
\]
Bilinear Basis Functions

- Consider one of the corners \((0, 0)\) of the reference rectangle and the corresponding basis \(\hat{\phi}_{0,0}\) restricted to \(\hat{\Omega}\):

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

- For an actual grid, the basis function corresponding to a given interior node is simply a composite of 4 such bilinear terms, one for each rectangle that has that interior node as a vertex: Often called a tent function.

- If higher smoothness is required one can consider, for example, bicubic Hermite interpolation (when derivatives \(f_x, f_y\) and \(f_{xy}\) are known at the nodes as well).

- Generalization of bilinear to 3D is trilinear interpolation

\[
\phi(x, y, z) = axyz + bxy + cxz + dyz + ex + fy + gz + h,
\]

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

- Bilinear basis function $\phi_{0,0}$ on reference rectangle
- Bilinear basis function $\phi_{3,3}$ on a 5x5 grid
Bicubic basis functions

Bicubic basis function $\phi_{3,3}$ on a 5x5 grid
Irregular (Simplicial) Meshes

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

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 linear function we need 3 coefficients $(x, y, \text{const})$, for quadratic 6 $(x, y, x^2, y^2, xy, \text{const})$:

![Diagram showing basis functions on a triangle grid](image)
Fig. 8.7. Characteristic piecewise Lagrange polynomial, in two and one space dimensions. Left, $k = 0$; right, $k = 1$. 
For regular grids the function

\[ qz = \text{interp2}(x, y, z, qx, qy, 'linear') \]

will evaluate the piecewise bilinear interpolant of the data \( x, y, z = f(x, y) \) at the points \( (qx, qy) \).

Other methods are 'spline' and 'cubic', and there is also \textit{interp3} for 3D.

For irregular grids one can use the old function \textit{griddata} which will generate its own triangulation or there are more sophisticated routines to manipulate triangulations also.
Regular grids

\[
[x, y] = \text{meshgrid}(-2:.5:2, -2:.5:2);
\]
\[
z = x.*\exp(-x.^2-y.^2);
\]
\[
ti = -2:.1:2;
\]
\[
[qx, qy] = \text{meshgrid}(ti, ti);
\]
\[
qz = \text{interp2}(x, y, z, qx, qy, 'cubic');
\]
\[
\text{mesh}(qx, qy, qz); \text{ hold on;}
\]
\[
\text{plot3}(x, y, z, 'o'); \text{ hold off;}
\]
MATLAB's interp2
Irregular grids

\[
x = \text{rand}(100,1) \times 4 - 2; \quad y = \text{rand}(100,1) \times 4 - 2;
\]
\[
z = x \times \exp(-x^2 - y^2);
\]
\[
t_i = -2:.1:2;
\]
\[
[qx, qy] = \text{meshgrid}(t_i, t_i);
\]
\[
qz = \text{griddata}(x, y, z, qx, qy, 'cubic');
\]
\[
\text{mesh}(qx, qy, qz); \quad \text{hold on};
\]
\[
\text{plot3}(x, y, z, 'o'); \quad \text{hold off};
\]
MATLAB’s \texttt{griddata}
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).