Interpolation

Things you should know

- Lagrange vs. Hermite interpolation
- Conditioning of interpolation
- Uniform vs. non-uniform points, Lebesgue constant
- Polynomial bases: Lagrange, Newton, Monomial
Classical polynomial interpolation
Newton polynomial basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(t) := \prod_{j=0}^{i-1} (t - t_j) \in P_i.$$ 

The leading coefficient $a_n$ of the interpolation polynomial of $f$

$$P(f|t_0, \ldots, t_n) = a_n x^n + \ldots$$

is called the $n$-th divided difference, $[t_0, \ldots, t_n]f := a_n$. 
Theorem: For \( f \in C^n \), the interpolation polynomial \( P(f|t_0, \ldots, t_n) \) is given by

\[
P(t) = \sum_{i=0}^{n} [t_0, \ldots, t_i] f \omega_i(t).
\]

If \( f \in C^{n+1} \), then

\[
f(t) = P(t) + [t_0, \ldots, t_n, t] f \omega_{n+1}(t).
\]

This property allows to estimate the interpolation error.
The divided differences \([t_0, \ldots, t_n]f\) satisfy the following properties:

- \([t_0, \ldots, t_n]P = 0\) for all \(P \in P_{n-1}\).

- If \(t_0 = \ldots = t_n\):

\[ [t_0, \ldots, t_n]f = \frac{f^{(n)}(t_0)}{n!} \]

nodes.
Classical polynomial interpolation

Divided differences

- The following recurrence relation holds for \( t_i \neq t_j \) (nodes with a hat are removed):

\[
[t_0, \ldots, t_n] f = \frac{([t_0, \ldots, \hat{t}_i, \ldots, t_n] f - [t_0, \ldots, \hat{t}_j, \ldots, t_n] f)}{t_j - t_i}
\]

- If \( f \in C^n \) \( \left[ t_0, \ldots, t_n \right] f = \frac{1}{n!} f^{(n)}(\tau) \) with \( a \leq \tau \leq b \), and the divided differences depend continuously on the nodes.
Let us use divided differences to compute the coefficients for the Newton basis for the cubic interpolation polynomial \( p \) that satisfies \( p(0) = 1, \ p(0.5) = 2, \ p(1) = 0, \ p(2) = 3 \).

| \( t_i \) | \( [t_0]f = 1 \) | \( [t_1]f = 2 \) | \( [t_0t_1]f = \frac{[t_1]f - [t_0]f}{t_1 - t_0} = 2 \) | \( [t_2]f = 0 \) | \( [t_1t_2]f = \frac{[t_2]f - [t_1]f}{t_2 - t_1} = -4 \) | \( [t_0t_1t_2]f = -6 \) | \( [t_2t_3]f = \frac{[t_3]f - [t_2]f}{t_3 - t_2} = 3 \) | \( [t_1t_2t_3]f = \frac{14}{3} \) | \( \frac{16}{3} \) |

Thus, the interpolating polynomial is

\[
p(t) = 1 + 2t + (-6)t(t - 0.5) + \frac{16}{3}t(t - 0.5)(t - 1).\]
Let us now use divided differences to compute the coefficients for
the Newton basis for the cubic interpolation polynomial \( p \) that
satisfies \( p(0) = 1, p'(0) = 2, p''(0) = 1, p(1) = 3. \)

\[
\begin{array}{c|ccc}
 t_i & [t_0]f = 1 & [t_0 t_1]f = p'(0) = 2 & [t_0 t_1 t_2]f = \frac{p''(0)}{2!} = \frac{1}{2} \\
0 & [t_0]f = 1 & [t_1 t_2]f = p'(0) = 2 & 0 \\
0 & [t_0]f = 1 & 0 & \frac{1}{2} \\
1 & [t_3]f = 3 & [t_2 t_3]f = \frac{[t_3]f-[t_0]f}{t_3-t_0} = 2 & -\frac{1}{2}
\end{array}
\]

Thus, the interpolating polynomial is

\[
p(t) = 1 + 2t + \frac{1}{2} t^2 + (-\frac{1}{2})t^3
\]
Classical polynomial interpolation

Approximation error

If $f \in C^{(n+1)}$, then

$$f(t) - P(f|t_0, \ldots, t_n)(t) = \frac{f^{(n+1)}(\tau)}{(n + 1)!} \omega_{n+1}(t)$$

for an appropriate $\tau = \tau(t)$, $a < \tau < b$.

In particular, the error depends on the choice of the nodes.

$$f(t) - P(f|t_0, \ldots, t_n)(t) = \left[ t_0, t_1, \ldots, t_{n+1} \right] f \omega_{n+1}(t)$$

$$= \frac{f^{(n+1)}(\tau)}{(n+1)!} \omega_{n+1}(t), \quad \tau \in (a,b)$$

For Taylor interpolation, i.e., $t_0 = \ldots = t_n$, this results in:

$$f(t) - P(f|t_0, \ldots, t_n)(t) = \frac{f^{(n+1)}(\tau)}{(n + 1)!} (t - t_0)^{n+1}$$
Consider functions

\[ \{ f \in C^{n+1}([a, b]) : \sup_{\tau \in [a, b]} |f^{n+1}(\tau)| \leq M(n + 1)! \} \]

for some \( M > 0 \), then the approximation error depends on \( \omega_n(t) \), and thus on \( t_0, \ldots, t_n \).

Thus, one can try to minimize

\[ \max_{a \leq t \leq b} |\omega_{n+1}(t)|, \]

which is achieved by choosing the nodes as the roots of the Chebyshev polynomial of order \((n + 1)\).
Summary on pointwise convergence:

- If an interpolating polynomial is close/converges to the original function depends on the regularity of the function and the choice of interpolation nodes.

- For a good choice of interpolation nodes, fast convergence can be obtained for almost all functions.
Classical polynomial interpolation

- Polynomial interpolation
- Least squares with polynomials
- Splines (i.e., piecewise polynomial interpolation):
Assume \((l + 2)\) pairwise disjoint nodes:

\[ a = t_0 < t_1 < \ldots < t_{l+1} = b. \]

A spline of degree \(k - 1\) (order \(k\)) is a function in \(C^{k-2}\) which on each interval \([t_i, t_{i+1}]\) coincides with a polynomial in \(P_{k-1}\).

Most important examples:

- linear splines, \(k = 2\)
- cubic splines, \(k = 4\)
Splines

Cubic splines look smooth:

\[ s_i \text{ polynomials of degree } 3 \]
(4 degrees of freedom)

\[ \text{overall: } 4 \times 4 \text{ unknowns} \]

Constraints:
- Each \( s_i \) is required to interpolate at 2 points
- \( 4 \times 2 \) conditions
- First & second derivatives coincide at intersection points

\[ 3 \times 2 = 6 \text{ cond} \]

\[ (s_0'(t) = s_i'(t), \quad s_0''(t) = s_i''(t), \quad \ldots) \]

\[ \rightarrow 2 \text{ free degrees of freedom, e.g. } s_0''(a) = s_2''(b) = 0 \]
B-splines are a basis in the spline space that:

- has local support
- satisfies a 3-term recursion
- non-negative

\[ N_{i1} \quad N_{i2} \quad N_{i3} \]

\[ \tau_i \quad \tau_{i+1} \quad \tau_i \quad \tau_{i+1} \quad \tau_{i+2} \quad \tau_i \quad \tau_{i+1} \quad \tau_{i+2} \quad \tau_{i+3} \]
Splines

B-splines

- Coefficients for interpolation with the B-spline basis can be computed efficiently using the De Boor algorithm.
- Splines are essential in Computer Aided Design (CAD).
- Also important in CAD: Bezier curves (these do not interpolate points and have useful geometrical properties).
Trigonometric Interpolation

For periodic functions

Instead of polynomials, use \( \sin(jt), \cos(jt) \) for different \( j \in \mathbb{N} \).

For \( N \geq 1 \), we define the set of complex trigonometric polynomials of degree \( \leq N - 1 \) as

\[
T_{N-1} := \left\{ \sum_{j=0}^{N-1} c_j e^{ijt}, c_j \in \mathbb{C} \right\},
\]

where \( i = \sqrt{-1} \).

**Complex interpolation problem:** Given pairwise distinct nodes \( t_0, \ldots, t_{N-1} \in [0, 2\pi) \) and corresponding nodal values \( f_0, \ldots, f_{N-1} \in \mathbb{C} \), find a trigonometric polynomial \( p \in T_{N-1} \) such that \( p(t_i) = f_i \), for \( i = 0, \ldots, N - 1 \).
Trigonometric Interpolation

There exists exactly one $p \in T_{N-1}$, which solves this interpolation problem.

Choose the equidistant nodes $t_k := \frac{2\pi k}{N}$ for $k = 0, \ldots, N - 1$. Then, the trigonometric polynomial that satisfies $p(t_i) = f_i$ for $i = 0, \ldots, N - 1$ has the coefficients

$$c_j = \frac{1}{N} \sum_{k=0}^{N-1} e^{-\frac{2\pi i j k}{N}} f_k.$$ 

For equidistant nodes, the linear map from $\mathbb{C}^N \to \mathbb{C}^N$ defined by $(f_0, \ldots, f_{N-1}) \mapsto (c_0, \ldots, c_{N-1})$ is called the discrete Fourier transformation (DFT).

Invert DFT:

$$\sum_{j=0}^{N-1} c_j e^{i \omega j t} \bigg|_{t_0, t_1, \ldots, t_N} \quad \mapsto \quad (f_0, \ldots, f_{N-1}).$$
Discrete Fourier transform

The interpolation problem \((f_0, \ldots, f_{N-1}) \mapsto (c_0, \ldots, c_{N-1})\) and its inverse require the multiplication or solution with a dense \(n \times n\) system, i.e., at least \(O(n^2)\) flops.

However, the special structure of the system matrix allows performing those operations using a much faster algorithm, the Fast Fourier Transform (FFT).
The Fast Fourier Transform (FFT) is a (very famous!) algorithm that computes the DFT and its inverse in $O(n)$ flops.

- Note that uniform nodes are used (and even required for the FFT).
- Tensor products on square domains can be used for two dimensional approximations, i.e., $p(x)p(y)$.
- Can be used to approximate and solve differential equations (see Numerical Methods II).