ZS 5

Freitag, 11. Januar 2019 15:42

5. Interpolation

5.1 Curve Fitting.

You have given n data points and want to fit a function over these points.

Introduction: • The aim is to find a function that goes through all data points. (= Curve fitting) → Data points are accurate.

• If we have d+1 interpolation points and get via Interpolation a polynomial p of degree d, then p is exactly determined. Hence it doesn't matter with what Basis you did the Interpolation, you will get the same polynomial. The reason we will look at different methods is because the way they get to p is faster, more stable etc.

• Interpolating with a Basis is an $Bc = y$ Problem, with B = Basis(f) $c = ?$ y= data values.

$$\begin{bmatrix} b_0(t_0) & \cdots & b_n(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \cdots & b_n(t_n) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}$$

Piecewise linear Interpolation

$$\text{Basis: } b_j(t) := \begin{cases} 1 - \frac{t_j - t}{t_j - t_{j-1}} & \text{on } [t_{j-1}, t_j] \\ 1 - \frac{t - t_j}{t_{j+1} - t_j} & \text{on } [t_j, t_{j+1}] \\ 0 & \text{otherwise} \end{cases} \qquad j = 1, \dots, n-1$$

$$b_0(t) := \begin{cases} 1 - \frac{t - t_0}{t_1 - t_0} & \text{on } [t_0, t_1] \\ 0 & \text{otherwise} \end{cases}$$

$$b_n(t) := \begin{cases} 1 - \frac{t_n - t}{t_n - t_{n-1}} & \text{on } [t_{n-1}, t_n] \\ 0 & \text{otherwise} \end{cases}$$

Note: $B = I$ since it's a cardinal Basis

Complexity: To evaluate for an arbitrary t, we have:

$$\sum_{i=0}^{n} c_i \cdot \overset{B=I}{=} \sum_{i=0}^{n} y_i \cdot b_i(t)$$

To comp $b_i(t)$ we only need $O(1)$
Need to do it n-times: ⇒ $O(n)$

Problem: Not really a nice Interpolation function:  (not smooth)
This is not a Polynomial!

Global Polynomial Interpolation

Trys to actually fit a polynomial. Advantages of polynomials:
- differentiation & integration easy to compute
- efficient evaluation (Horner-scheme) $O(n)$ S.127
- approximation property of polynomials

Note: Higher degree of Polynom $\not\Rightarrow$ Better Approximation

Monomial Basis $(1, t, t^2, \dots)$
Not a cardinal Basis. B is a normal Matrix. (not a cardinal Basis)
Need to do back and forward substitution: $O(n^3)$

Lagrange Basis

⊖ Numerically unstable with close nodes

$$\text{Basis: } L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} \qquad i = 0, \dots, n$$

Complexity: To evaluate at an arbitrary t we have:

$$\sum_{i=0}^{n} c_i \cdot L_i(t) \overset{B=I}{=} \sum_{i=0}^{n} y_i \cdot \underbrace{L_i(t)}_{O(n) \text{ (with Horner Scheme)}}$$

$\Rightarrow n \cdot O(n) = O(n^2)$

$\Rightarrow$ For N points: $O(n^2 N)$

Barycentric Interpolation.

Just an other way of computing Lagrange Interpolation so that for N points we have: $O(n^2 + N \cdot n)$

$$p(t) = \frac{\sum_{i=0}^{n} y_i \frac{\lambda_i}{t - t_i}}{\sum \frac{\lambda_i}{t - t_i}} \qquad \lambda_i = \prod_{j=0} \frac{1}{t_i - t_j}$$

5.2. Approximation of a function by Interpolation.

Now we can choose the nodes ourself.
Now we can fight Runge's effect.

Since we now can choose the nodes we then can choose them clever, such that:
- Runges' effect is fought
- We can compute $p(t)$ fast and numerically stable
- Very good convergence

Algebraic convergence: $\| f - I_T f \| = O(q^n)$ // linear bei log
Exponential convergence: $\| f - I_T f \| = O(n^t)$ // linear bei lin-

$$\text{Error: } \| f - L_T f \|_{L^\infty} \leq \frac{\| f^{(n+1)} \|_{L^\infty}}{(n+1)!} \max_{t \in I} |(t - t_0) \cdots (t - t_n)|$$

Global polynomial Interpolation

We have one polynom over the whole Interval $I = [a, b]$

Chebyshev Interpolation

Interpolation with Chebyshev Basis and Chebyshev nodes.

Nodes: $t_j := a + \frac{1}{2}(b-a)\left(\cos\left(\frac{2j+1}{2(n+1)}\pi\right) + 1\right) \qquad t_j \in [a, b]$

Basis: $T_n(t) := \cos(n \cdot \arccos(t))$

or
$$\begin{cases} T_0(t) = 1 \\ T_1(t) = t \\ T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t) \end{cases}$$

$$\| f - L_T f \|_{L^\infty} \leq \frac{2^{-n}}{(n+1)!} \cdot |I|^{n+1} \| f^{(n+1)} \|_{L^\infty([-1,1])}$$

Convergence: – Exponential if $f \in C^\infty$
– Algebraic if $f \in C^0, C^1$

Computation: To find the coefficients c ($Bc = y$) we use FFT and difficult Indexing Variables to get: $O(n \cdot \log$
To actually calculate $p(t) = \sum_{j=0}^{n} \alpha_j T_j(t)$ we use Clenshaws' algo; Devide and Conquer: $O(n)$

Piecewise polynomial Interpolation

We devide the Intervall into different parts. This can be Idea because if you look at the interpolation error you change two factors: Nodes and Intervall length.

Both following Interpolation methods will decrease the interpolation in mesh width. (algebraictly)

Cubic Splines

In each Subintervall we fit a polynomial of degree 3 and require that our overall function will be in C^2

→ We need to determine $4n$ coefficients.
↳ With $4n$ conditions we solve a sparse $(n-1)\times(n-1)$ LSE for a vector σ. With σ we can compute the coefficients.
↳ S.156 ! $s_j(t) = \tilde{a}_j + \tilde{b}_j(t - t_{j-1}) + \tilde{c}_j(t - t_{j-1})^2 + \tilde{d}_j(t - t_j)^3$!

Complexity: Solve LSE of Sparse Matrice ⇒ ~$O(n^2)$

⊕ Global smoothness
⊖ Need to solve LSE

Piecewise polynomial Lagrange interpolation

Just split I in subintervals and do global lagrange interp on the subinterval. In order to have a continuous f you always have start and end of the interval as noc

Note: Higher degree polynomial provides only faster algebraic But not exponential!

⊕ No need to solve LSE
⊖ No global smoothness

$i=0$ $\quad j \neq i$

Newton Basis

$$N_0(t) = 1$$

$$N_i(t) = \prod_{j=0}^{i-1} (t - t_j) \qquad \Rightarrow \quad p(t) = \sum_{i=0}^{n} c_i \cdot N_i(t)$$

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & N_1(t_1) & 0 & \ddots & \vdots \\ 1 & N_1(t_2) & N_2(t_2) & \ddots & \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & N_1(t_n) & N_2(t_n) & \cdots & N_n(t_n) \end{bmatrix}$$

Complexity: $\mathcal{O}(\underbrace{n^2}_{\text{Assembling } B \ (= \text{Evaluating } N_i(t_j))} + \underbrace{Nn^2}_{\text{Solving}})$

+ More or less stable
− Runges effect