CSC349A Numerical Analysis

Lecture 15

Rich Little

University of Victoria

2018

Table of Contents I

Error term of polynomial interpolation

Theorem:
Let $x_0, x_1, \ldots x_n$ be any distinct points in $[a, b]$. Let $f(x) \in C^{n+1}[a, b]$ and let $P(x)$ interpolate $f(x)$ at x_i.
Then for each $\hat{x} \in [a, b]$, there exists a value ξ in (a, b) such that

$$f(\hat{x}) = P(\hat{x}) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (\hat{x} - x_i)$$

for example for $n = 3$

$$f(\hat{x}) = P(\hat{x}) + \frac{f^{(4)}(\xi)}{24} (\hat{x} - x_0)(\hat{x} - x_1)(\hat{x} - x_2)(\hat{x} - x_3)$$

Table of Contents I

The Runge Phenonmenon

The following example is the classical example to illustrate the oscillatory nature and thus the unsuitability of high order interpolating polynomials.

Example: Consider the problem of interpolating

$$f(x) = \frac{1}{1 + 25x^2}$$

on the interval $[-1, 1]$ at $n + 1$ equally-spaced points x_i by the interpolating polynmial $P_n(x)$.

Graphs of $f(x)$, $P_5(x)$, and $P_{20}(x)$

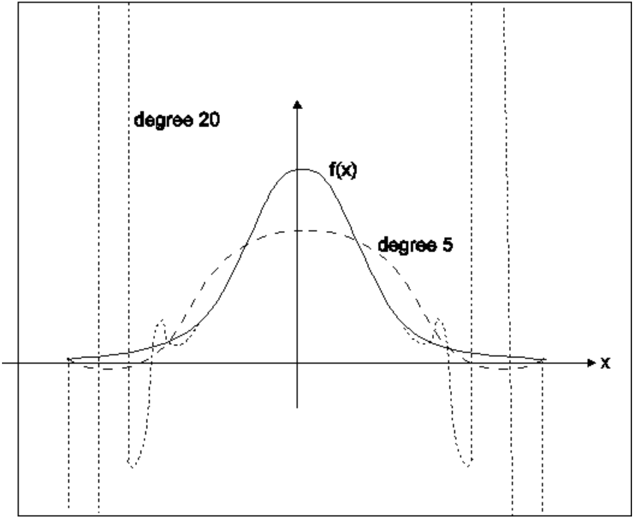

Figure: The Runge function

Runge's Theorem

- Runge proved that as $n \to \infty$, $P_n(x)$ diverges from $f(x)$ for all values of x such that $0.726 \le |x| < 1$ (except for the points of interpolation x_i).
- The interpolating polynomials do approximate $f(x)$ well for $|x| < 0.726$.
- One way to see that the difference between $f(x)$ and $P_n(x)$ becomes arbitrarily large as n becomes large is to consider the error term for polynomial interpolation

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

As $n \to \infty$, it can be shown that $f(x) - P_n(x) \to \infty$ (at some points x in $[-1, 1]$).

Table of Contents I

Piecewise interpolation

An alternative to polynomial interpolation use "piecewise" polynomials.
Given $x_0, x_1, \ldots, x_n$ and $f(x_0), f(x_1), \ldots, f(x_n)$ construct a different interpolating polynomial on each subinterval:

$$[x_0, x_1], [x_1, x_2], \ldots, [x_{n-1}, x_n]$$

For example piecewise linear interpolation: construct a linear polynomial on each subinterval $[x_i, x_{i+1}]$.

Linear splines

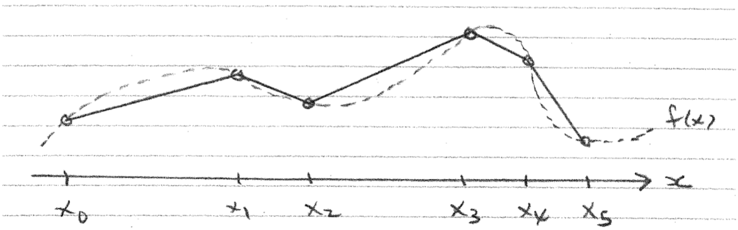

Figure: Example of linear spline

Disadvantage of piecewise linear polynomials: not differentiable (at points x_i, the knots).

Quadratic splines

Differentiability can be obtained by using quadratic (instead of linear) polynomials on each $[x_i, x_{i+1}]$.

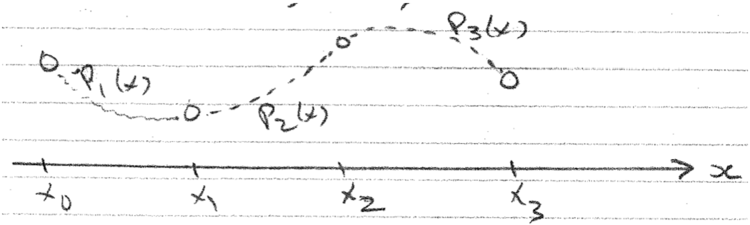

Figure: Example of quadratic spline

Quadratic splines

- Each $P_i(x)$ is a quadratic (and is not uniquely determined)
- The piecewise polynomial can be made differentiable on $[x_0, x_n]$
- If differentiable, this is an example of a spline function

Spline Defintion

Definition: $S(x)$ is a spline function on $[x_0, x_n]$ if for some $q \geq 1$

1. $S(x)$ is a polynomial of degree q on each subinterval $[x_i, x_{i+1}]$
2. $S(x)$ and its first $q - 1$ derivatives are continuous on $[x_0, x_n]$

Spline types:

- Linear spline, $q = 1$
- Quadratic spline, $q = 2$
- Cubic spline, $q = 3$

Physical splines

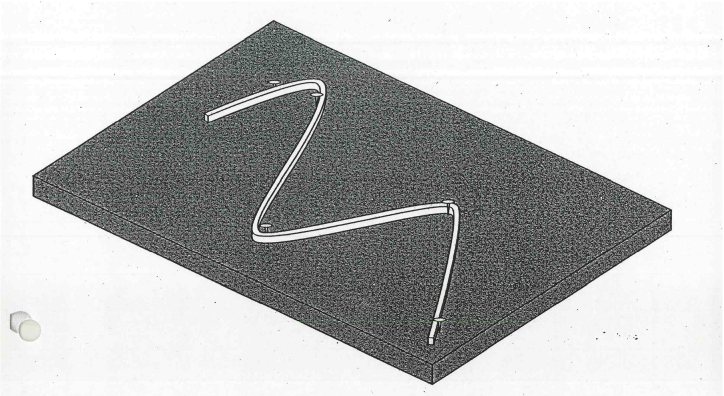

Figure: Drafting technique of using a spline to draw smooth curves through a series of points

History

Splines were first defined by Schoenberg in 1946. Note that the definition of a spline function does **not** require that it interpolates some given function $f(x)$. But splines are often used as interpolating functions (a spline interpolant):

- They do not have the osciallatory nature of high degree interpolating polynomials
- They require no derivatives of $f(x)$, except possibly at the end points x_0 and x_n.

The most common spline interpolant is **cubic**.

Applications

- Graphics
  - Smooth curves (continuity)
- Animation
  - Modeling = specifying shape
  - Animation = specifying shape over time
  - Real objects don't move in straight lines - Video
- Motion Control
  - embedded systems
  - automated motion
  - robotics

Table of Contents I

Cubic Spline Interpolants

Definition: Given $x_0, x_1, ..., x_n$ with $x_i < x_{i+1}$ for each i, and $f(x_0), f(x_1), ..., f(x_n)$, then $S(x)$ is a **cubic spline interpolant** for $f(x)$ if,

(a) $S(x)$ is a cubic polynomial, denoted by $S_j(x)$, on each subinterval $[x_j, x_{j+1}]$, for $j = 0, ..., n-1$

(b) $S_j(x_j) = f(x_j)$, for $j = 0, ..., n-1$ and $S_{n-1}(x_n) = f(x_n)$

(c) $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$, for $j = 0, ..., n-2$

(d) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$, for $j = 0, ..., n-2$

(e) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$, for $j = 0, ..., n-2$

(f) either one of the following hold:

  (i) $S''(x_0) = S''(x_n) = 0$ (natural bounds), or
  (ii) $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (clamped bounds)

Cubic Spline Interpolants II

Notes:

- for any $f(x)$, there exist an infinite number of cubic splines satisfying conditions (a) - (e). Why?
- There are n cubic polynomials $S_j(x)$ to specify, each one is defined by 4 coefficients, giving a total of $4n$ unknowns to be specified.
- However, condition (b) gives $n+1$ conditions to be satisfied, and (c), (d) and (e) each give $n-1$ conditions to be satisfied.
- Thus, there are $(n+1)+3(n-1)=4n-2$ conditions (equations) to be satisfied in $4n$ unknowns.
- But if either (i) or (ii) is also required to be satisfied, then there are $4n$ conditions in $4n$ unknowns and there exists a <u>unique</u> cubic spline interpolant satisfying (a) - (f).

Example: Quadratic Spline

Determine a, b, c, d, and e so that

$$Q(x) = \begin{cases} ax^2 + x + b, & -1 \le x \le 0 \\ cx^2 + dx + e, & 0 \le x \le 1 \end{cases}$$

is a quadratic spline function that interpolates $f(x)$ where $f(-1) = 1, f(0) = 1, f(1) = 1$.

Example: Cubic Spline

Determine $a_0, b_0, d_0, a_1, b_1, c_1,$ and d_1 so that

$$S(x) = \begin{cases} a_0 + b_0 x - 3x^2 + d_0 x^3, & -1 \leq x \leq 0 \\ a_1 + b_1 x + c_1 x^2 + d_1 x^3, & 0 \leq x \leq 1 \end{cases}$$

is the natural cubic spline function such that
$S(-1) = 1, S(0) = 2, S(1) = -1.$

Cubic Splines in MATLAB

There is an algorithm for spline computation given in the text but it has a different derivation than what we have done and different from MATLAB. In MATLAB they use a different form for the splines. For example, when $n = 3$, MATLAB uses the following form for the cubic polynomials:

$$S_0(x) = a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3$$
$$S_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3$$
$$S_2(x) = a_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3$$

Note that, with this form, $a_0 = f(x_0), a_1 = f(x_1)$, and $a_2 = f(x_2)$. This simplifies the system somewhat.