Numerical Methods for Computational Science and Engineering

Fall Semester 2017 (HS17)

Prof. Rima Alaifari, SAM, ETH Zurich

Polynomial interpolation algorithms:

Consider: • data points $(\underbrace{t_i}_{\in I}, y_i)$ $i \in \{0, \dots, n\}$

• find interpolant $p(t) \in \mathcal{P}_n$ and evaluate $p(x)$ for some $x \in I$

• using the monomial basis

Find coefficients $\alpha_0, \dots, \alpha_n$ in $p(t) = \sum_{i=0}^{n} \alpha_i t^i$

$$\underbrace{\begin{bmatrix} 1 & t_0 & t_0^2 & \dots & t_0^n \\ 1 & t_1 & t_1^2 & \dots & t_1^n \\ \vdots & & & & \\ 1 & t_n & t_n^2 & \dots & t_n^n \end{bmatrix}}_{\text{Vandermonde matrix}} \begin{bmatrix} \alpha_0 \\ \vdots \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

Effort of finding $\alpha_0, \dots, \alpha_n$: $\underline{O(n^3)}$

• Using Lagrange polynomials:

$$p(x) = \sum_{i=0}^{n} y_i L_i(x)$$

Evaluating $L_i(x)$ is $O(n)$ [Horner scheme]

$\Rightarrow$ Evaluating $p(x)$ is $\underline{O(n^2)}$

Series of interpolation problems:

- Fixed nodes $t_0, \dots, t_n \in I$
- N different data value sets

  $\{y_0^k, \dots, y_n^k\}$, $k \in \{1, \dots, N\}$
- For every k: find interpolant $p_k \in \mathcal{P}_n$

  and evaluate $p_k(x_k)$, $x_k \in I$, $k \in \{1, \dots, N\}$

Complexity of monomial basis approach:

$$\underline{\Theta(n^3 N)} \quad (+\Theta(nN))$$

evaluating N polynomials of deg n

Complexity of Lagrange basis approach:

$$\underline{\Theta(n^2 N)}$$

5.2.27 Barycentric interpolation approach

$$p(t) = \sum_{i=0}^{n} y_i L_i(t) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-t_j}{t_i-t_j}$$

$$= \sum_{i=0}^{n} y_i \lambda_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (t-t_j)$$

$$\lambda_i := \frac{1}{(t_i-t_0)(t_i-t_1)\dots(t_i-t_{i-1})(t_i-t_{i+1})\dots(t_i-t_n)} \qquad i=0,\dots,n$$

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

For $p_1(t) \equiv 1$ $\quad y_{i,1} = 1$

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

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

(*)

$$\Rightarrow \quad p(t) = \frac{\sum_{i=0}^{n} y_i \frac{\lambda_i}{t-t_i}}{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}}$$

- Computing $\lambda_0, \dots, \lambda_n$: $\underline{\Theta(n^2)}$
- Evaluating $p(x_k) = \dfrac{\sum_{i=0}^{n} y_i^k \frac{\lambda_i}{x_k - t_i}}{\sum_{i=0}^{n} \frac{\lambda_i}{x_k - t_i}}$

Effort for each k: $\underline{\Theta(n)}$ $\quad k \in \{1, \dots, N\}$

Total computational complexity: $\underline{\Theta(n^2 + nN)}$

<u>Note</u>: If nodes t_i are close to each other: numerical instability in computing λ_i's (also in evaluating Lagrange polynomials)

Different basis approach: Newton basis

$$N_0(t) \equiv 1, \quad N_i(t) = \prod_{j=0}^{i-1}(t-t_j), \quad i = 1, \dots, n$$

i-degree polynomial $\rightarrow$ linearly independent set $\{N_0, \dots, N_n\}$

$\underline{N_i(t_\ell) = 0}$ $\quad \forall \ell < i$

Find interpolant $p(t) = \sum_{i=0}^{n} a_i N_i(t)$

$$p(t_0) = a_0 \quad + 0$$
$$p(t_1) = a_0 + a_1 N_1(t_1) \quad + 0$$
$$p(t_2) = a_0 + a_1 N_1(t_2) + a_2 N_2(t_2) + 0$$
$$\vdots$$

$\Rightarrow$ lower triangular system for $[a_0, \dots, a_n]^T$

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 1 & N_1(t_1) & 0 & \cdots & 0 \\ \vdots & N_1(t_2) & N_2(t_2) & 0 \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & N_1(t_n) & N_2(t_n) & \cdots & N_n(t_n) \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

Effort for computing entries of the matrix: $\Theta(n^2)$

Solving for $[a_0, \dots, a_n]^T$: $\Theta(n^2)$ [forward substitution]

N evaluations for $[y_0^k, \dots, y_n^k]$ $\quad k \in \{1, \dots, n\}$

and computing $p_k(x_k)$: $\Theta(n^2 + n^2 N) = \Theta(n^2 N)$

building system matrix once

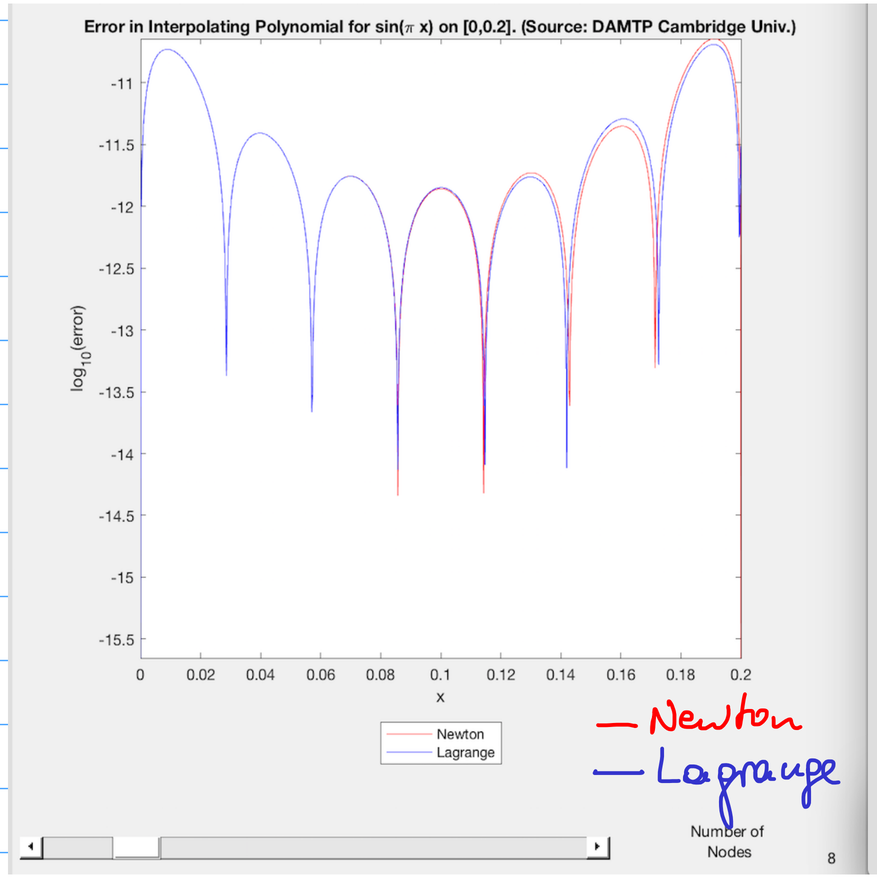
N forward substitutions

<u>Note</u>: The interpolant is unique (algebraically, Vandermonde, Lagrange & Newton interpolation are equivalent)

Example: $\sin(\pi x)$ on $[0, 0.2]$

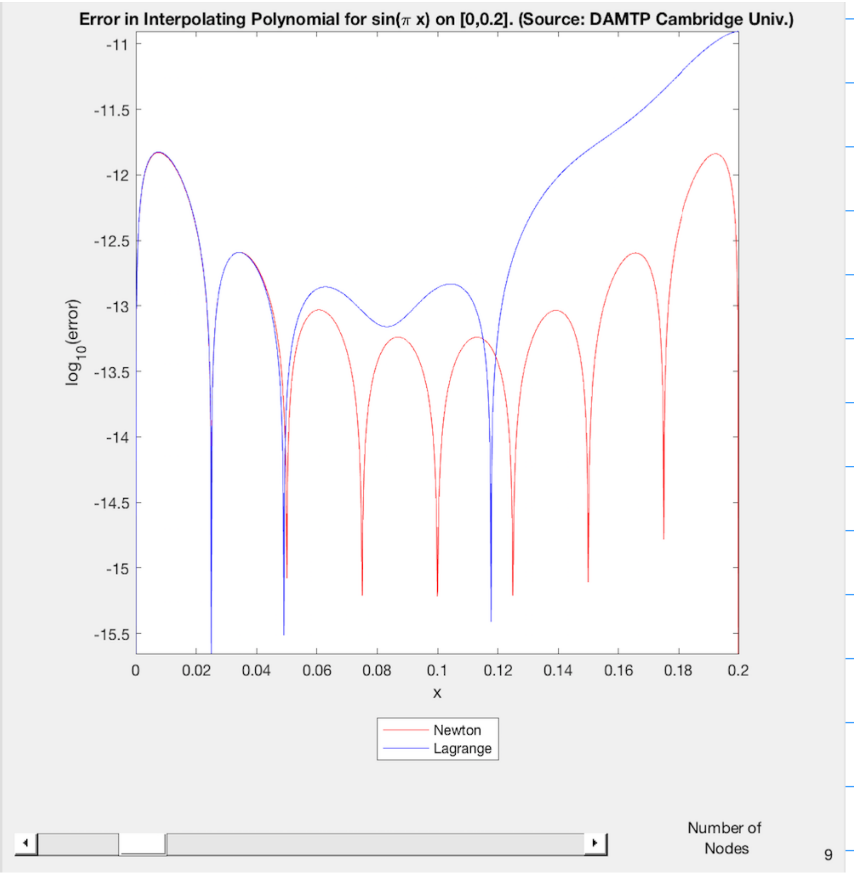
error in interpolating with Lagrange vs. Newton; nodes: equidistant

8 nodes

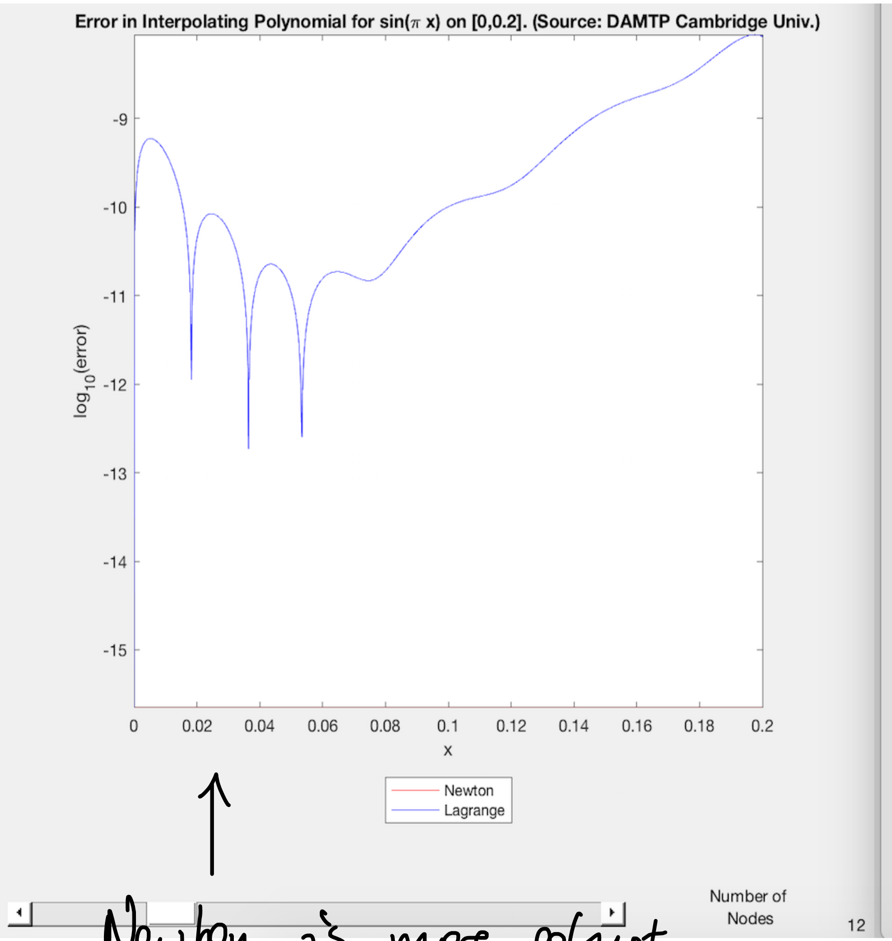


— Newton
— Lagrange

9 nodes

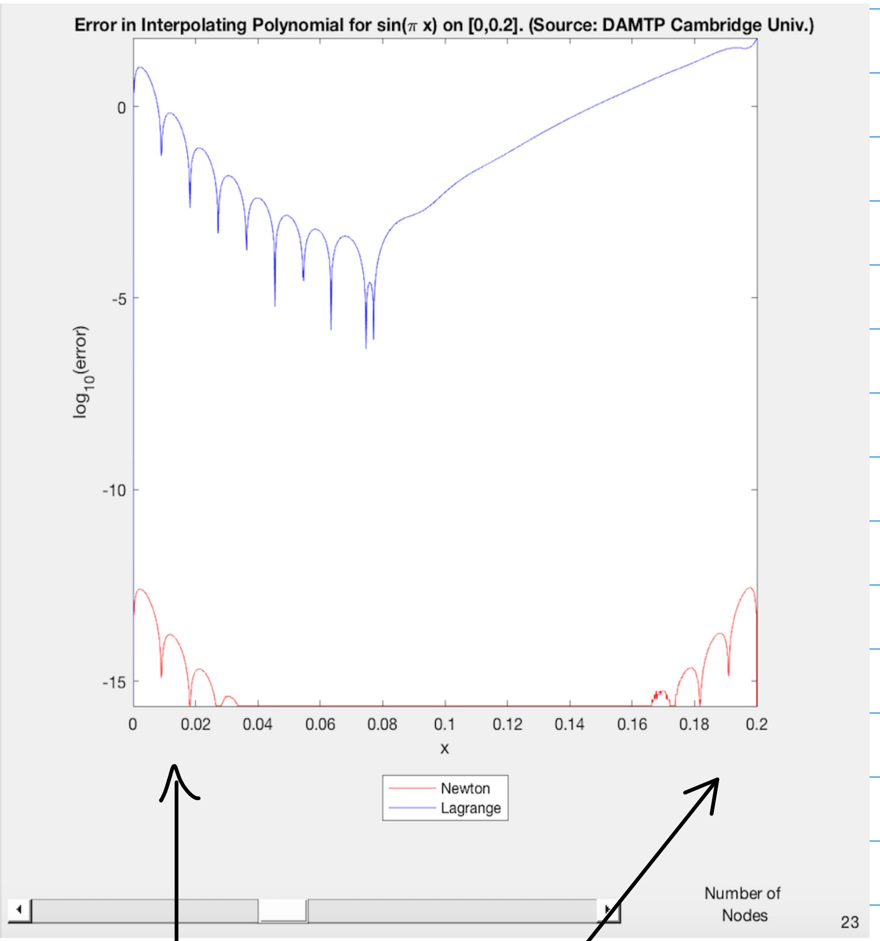


12 nodes



Newton is more robust

23 nodes



higher degree
$\neq$
better approximation

Runge's phenomenon
interpolating with high deg.
polynomials leads to osc./
artifacts at the endpoints
[when equidistant nodes used]

Different ways to avoid Runge's phenomenon:

- points more densely distributed close to the endpoints (cf. Chebychev nodes)
- piecewise polynomial interpolation

"Update-friendly" implementation

What if a single data point is added or changed?

The Aitken–Neville Scheme

introducing partial interpolating polynomials:

$p_{k,\ell}$:= unique interpolating polynomial of degree $\ell - k$ through $(t_k, y_k), \dots, (t_\ell, y_\ell)$,

$p_{k,k}(x) \equiv y_k$ ("constant polynomial"), $k = 0, \dots, n$,

$$p_{k,\ell}(x) = \frac{(x - t_k)p_{k+1,\ell}(x) - (x - t_\ell)p_{k,\ell-1}(x)}{t_\ell - t_k} \quad (=: \tilde{p}_{k,\ell}(x)) \quad \text{degree } \ell - k$$

$$= p_{k+1,\ell}(x) + \frac{x - t_\ell}{t_\ell - t_k}(p_{k+1,\ell}(x) - p_{k,\ell-1}(x)), \quad 0 \le k \le \ell \le n, \tag{5.2.34}$$

$p_{k+1,\ell}$: interpolant $(t_{k+1}, y_{k+1}), \dots, (t_\ell, y_\ell)$

$p_{k,\ell-1}$: interpolant $(t_k, y_k), \dots, (t_{\ell-1}, y_{\ell-1})$

$\rightarrow$ degree $\ell - k - 1$

Check: $\tilde{p}_{k,\ell}(t_i) = y_i$. $i = k, \dots, \ell$ (exercise)

is interpolating through $(t_k, y_k), \dots, (t_\ell, y_\ell)$

By uniqueness: $\tilde{p}_{k,\ell} = p_{k,\ell}$

Recursive computation of $p_{0,n}(x)$:

		interpolation through 2 points	interp. through 3 points	through 4 points	
$n =$	0	1	2	3	
t_0	$y_0 =: p_{0,0}(x)$	$\rightarrow p_{0,1}(x)$	$\rightarrow p_{0,2}(x)$	$\rightarrow p_{0,3}(x)$	$\rightarrow p_{0,4}(x)$
t_1	$y_1 =: p_{1,1}(x)$	$\rightarrow p_{1,2}(x)$	$\rightarrow p_{1,3}(x)$	$\rightarrow p_{1,4}(x)$	
t_2	$y_2 =: p_{2,2}(x)$	$\rightarrow p_{2,3}(x)$	$\rightarrow p_{2,4}(x)$		
t_3	$y_3 =: p_{3,3}(x)$	$\rightarrow p_{3,4}(x)$			
new data point	$y_4 =: p_{4,4}(x)$				

"update-friendly"

C++-code 5.2.35: Aitken-Neville algorithm

```cpp
// Aitken-Neville algorithm for evaluation of interpolating polynomial
// IN: t, y:  (vectors of) interpolation data points
// x:  (single) evaluation point
// OUT: value of interpolant in x
double ANipoleval(const VectorXd& t, VectorXd y, const double x) {
  for (int i = 0; i < y.size(); ++i) {
    for (int k = i - 1; k >= 0; --k) {
      // Recursion (5.2.34)
      y(k) = y(k + 1) + (y(k + 1) - y(k))*(x - t(i))/(t(i) - t(k));
    }
  }
  return y(0);
}
```

Two nested loops: $O(n^2)$

direct evaluation of $p_{0,n}(x)$ at given point x

Update-friendly computation of coefficients in Newton basis interpolation?

Recall:

$$a_j \in \mathbb{R}: \quad a_0 N_0(t_j) + a_1 N_1(t_j) + \cdots + a_n N_n(t_j) = y_j, \quad j = 0, \ldots, n\,.$$

$\Leftrightarrow$ triangular linear system

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & (t_1 - t_0) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & (t_n - t_0) & \cdots & \prod_{i=0}^{n-1}(t_n - t_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

with some calculations:

$$a_0 = y_0$$

$$a_1 = \frac{y_1 - y_0}{t_1 - t_0}$$

$$a_2 = \frac{\dfrac{y_2 - y_1}{t_2 - t_1} - \dfrac{y_1 - y_0}{t_1 - t_0}}{t_2 - t_0}$$

$$a_3 = \frac{\dfrac{\frac{y_3 - y_2}{t_3 - t_2} - \frac{y_2 - y_1}{t_2 - t_1}}{t_3 - t_1} - \dfrac{\frac{y_2 - y_1}{t_2 - t_1} - \frac{y_1 - y_0}{t_1 - t_0}}{t_2 - t_0}}{t_3 - t_0}$$

Divided difference scheme:

$$\begin{aligned} y[t_i] &= y_i \\ y[t_i, \ldots, t_{i+k}] &= \frac{y[t_{i+1}, \ldots, t_{i+k}] - y[t_i, \ldots, t_{i+k-1}]}{t_{i+k} - t_i} \quad \text{(recursion)} \end{aligned} \tag{5.2.52}$$

$$\begin{array}{c|lllll}
t_0 & \overset{=: a_0}{y[t_0]} & & & & \\
 & & > \overset{=: a_1}{y[t_0, t_1]} & & & \\
t_1 & y[t_1] & & > \overset{=: a_3}{y[t_0, t_1, t_2]} & & \\
 & & > y[t_1, t_2] & & > \overset{=: a_4}{y[t_0, t_1, t_2, t_3]}, & \\
t_2 & y[t_2] & & > y[t_1, t_2, t_3] & & \\
 & & > y[t_2, t_3] & & & \\
t_3 & y[t_3] & & & &
\end{array} \tag{5.2.54}$$

C++-code 5.2.55: Divided differences, recursive implementation, in situ computation

```
// IN: t = node set (mutually different)
// y = nodal values
// OUT: y = coefficients of polynomial in Newton basis
void divdiff(const VectorXd& t, VectorXd& y) {
  const unsigned n = y.size() - 1;
  // Follow scheme (5.2.54), recursion (5.2.51)
  for (unsigned l = 0; l < n; ++l)
    for (unsigned j = l; j < n; ++j)
      y(j+1) = (y(j+1)-y(l))/(t(j+1)-t(l));
}
```

$$p(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)(t-t_1) + \cdots + a_n \prod_{j=0}^{n-1}(t-t_j) \tag{5.2.56}$$

$$a_0 = y[t_0],\ a_1 = y[t_0,t_1],\ a_2 = y[t_0,t_1,t_2],\ \ldots .$$

Adding new data point (t_n, y_n) :

calculate $y[t_n],\ y[t_{n-1}, t_n], \ldots, y[t_0, \ldots, t_n]$

"update-friendly" $\quad O(n)$

Evaluation of polynomial in Newton form:

based on Horner scheme

$$p \leftarrow a_n,\quad p \leftarrow (t-t_{n-1})p + a_{n-1},\quad p \leftarrow (t-t_{n-2})p + a_{n-2},\quad \ldots .$$

C++-code 5.2.58: Divided differences evaluation by modified Horner scheme

```
// Evaluation of polynomial in Newton basis (divided differences)
// IN: t = nodes (mutually different)
// y = values in t
// x = evaluation points (as Eigen::Vector)
// OUT: p = values in x */
void evaldivdiff(const VectorXd& t, const VectorXd& y, const
    VectorXd& x, VectorXd& p) {
  const unsigned n = y.size() - 1;

  // get Newton coefficients of polynomial (non in-situ
      implementation!)
  VectorXd coeffs; divdiff(t, y, coeffs);

  // evaluate
  VectorXd ones = VectorXd::Ones(x.size());
  p = coeffs(n)*ones;
  for (int j = n - 1; j >= 0; --j) {
    p = (x - t(j)*ones).cwiseProduct(p) + coeffs(j)*ones;
  }
}
```

$O(n)$ for evaluating $p(t)$.

5.5 Splines

Piecewise polynomial interpolation