

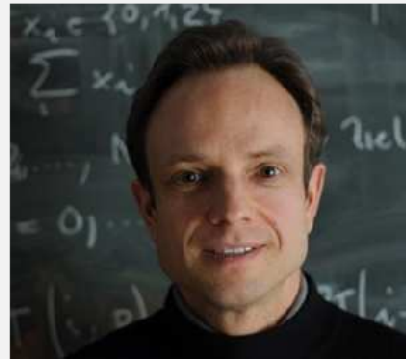
Course Video

Section 5.2.3: Polynomial Interpolation:
Algorithms

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 27, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Knowledge of C++

Dependencies. Section 5.1, Section 5.2.1, Section 5.2.2

Duration: 30 minutes

V. Data Interpolation and
Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.3. Polynomial Interpolation: Algorithms

Goal : Functor class

C++-code 5.1.0.12: C++ class representing an interpolant in 1D

```

1  class Interpolant {
2      private:
3          // Various internal data describing f
4          // Can be the coefficients of a basis representation (5.1.0.14)
5      public:
6          // Constructor: computation of coefficients  $c_j$  of representation
7          // (5.1.0.14)
8          Interpolant(const vector<double>& t, const vector<double>& y);
9          // Evaluation operator for interpolant f
10         double operator() (double t) const;
11     };

```

→ Adapt to different use scenarios

5.2.3.1. Multiple Evaluations

Task: For ❶ a **fixed** set $\{t_0, \dots, t_n\}$ of nodes,
 and ❷ **many** different given data values $y_i, i = 0, \dots, n$
 and ❸ **many** arguments $x_k, k = 1, \dots, N, N \gg 1$,
 efficiently compute all $p(x_k)$ for $p \in \mathcal{P}_n$ interpolating in $(t_i, y_i), i = 0, \dots, n$.

C++-code 5.2.3.1: Polynomial Interpolation

```

1 class PolyInterp {
2   private:
3     // various internal data describing p
4     Eigen::VectorXd t;
5   public:
6     // Constructors taking node vector (t_0, ..., t_n) as argument
7     PolyInterp(const Eigen::VectorXd &t);
8     template <typename SeqContainer>
9       PolyInterp(const SeqContainer &v);
10    // Evaluation operator for data (y_0, ..., y_n); computes
11    // p(x_k) for x_k's passed in x
12    Eigen::VectorXd eval(const Eigen::VectorXd &y,
13                        const Eigen::VectorXd &x) const;
14 };
  
```

Straightforward: $p(x_k) = \sum_{i=0}^n y_i L_i(x_k)$

$$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{(t - t_j)}{(t_i - t_j)} \triangleright \text{Cost} = O(n^2 N) \text{ for } n, N \rightarrow \infty$$

Idea: **Precompute** parts of L_i in the constructor!

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

where $\lambda_i = \frac{1}{(t_i - t_0) \cdots (t_i - t_{i-1})(t_i - t_{i+1}) \cdots (t_i - t_n)}$, $i = 0, \dots, n$: independent of y_i !

$\hookrightarrow$ precompute in constructor, cost = $O(n^2)$

Observe: $y_i = 1 \ \forall i \Rightarrow p(t) = 1$

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

$$\text{Barycentric interpolation formula} \quad p(t) = \frac{\sum_{i=0}^n \frac{\lambda_i}{t - t_i} y_i}{\sum_{i=0}^n \frac{\lambda_i}{t - t_i}}. \quad (5.2.3.3)$$

③

C++-code 5.2.3.7: Interpolation class: multiple point evaluations

```

2 template <typename NODESCALAR>
3 template <typename RESVEC, typename DATAVEC>
4 RESVEC BarycPolyInterp<NODESCALAR>::eval(const DATAVEC &y,
5                                           const nodeVec_t &x) const {
6     const idx_t N = x.size(); // No. of evaluation points
7     RESVEC p(N);              // Output vector
8     // Compute quotient of weighted sums of  $\frac{\lambda_i}{t-t_i}$ ,
9     // effort  $O(n)$ 
10    for (int i = 0; i < N; ++i) {
11        nodeVec_t z = (x[i] * nodeVec_t::Ones(n) - t);  $\rightarrow [x_k - t_i]_{i=0}^n$ 
12
13        // Check if we want to evaluate close to a node
14        const double tref{z.cwiseAbs().maxCoeff()}; // reference size
15        idx_t k;
16        if (z.cwiseAbs().minCoeff(&k) <
17            tref * std::abs(std::numeric_limits<NODESCALAR>::epsilon())) {
18            // evaluation at node  $t_k$ 
19            p[i] = y[k];
20        } else {
21            const nodeVec_t mu = lambda.cwiseQuotient(z);  $\rightarrow$  comb. weight (5.2.3.3)
22            p[i] = (mu.cwiseProduct(y)).sum() / mu.sum();
23        }
24    } // end for
25    return p;
26 }

```

$\rightarrow$ Loop over x_k -values

$$\text{Total cost} = O(n^2 + N \cdot n)$$

$\downarrow$
 constructor

$\downarrow$
 eval()

5.2.3.2. Single evaluation

```
double eval(const Eigen::VectorXd &t, const Eigen::VectorXd &y,
            double x);
```

$$n := (t.size() = x.size()) - 1 \Rightarrow 1$$

Tool: "partial polynomial interpolants"

$$p_{k,\ell} \in \mathcal{P}_{\ell-k} : p_{k,\ell}(t_j) = y_j, \quad j = k, \dots, \ell$$

Idea: recursion in degree

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

$$p_{k,\ell}(x) = \frac{(x - t_k)p_{k+1,\ell}(x) - (x - t_\ell)p_{k,\ell-1}(x)}{t_\ell - t_k} = * \in \mathcal{P}_{\ell-k} \quad (5.2.3.9)$$

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

$$* = \begin{cases} y_j & \text{for } x = t_j, k < j < \ell \\ y_k & \text{for } x = t_k \\ y_\ell & \text{for } x = t_\ell \end{cases}$$

$\downarrow$
satisfies IC

Finally : $p = p_{0,n}$

(4)

Ex: $n = 3$

$n =$	0	1	2	3	
t_0	$y_0 =: p_{0,0}(x)$	$\rightarrow p_{0,1}(x)$	$\rightarrow p_{0,2}(x)$	$\rightarrow p_{0,3}(x)$	$\rightarrow i \text{ loop}$
t_1	$y_1 =: p_{1,1}(x)$	$\rightarrow p_{1,2}(x)$	$\rightarrow p_{1,3}(x)$		$\downarrow k \text{ loop}$
t_2	$y_2 =: p_{2,2}(x)$	$\rightarrow p_{2,3}(x)$			
t_3	$y_3 =: p_{3,3}(x)$				

Aitken-Neville scheme

(ANS)

C++-code 5.2.3.10: Aitken-Neville algorithm

```

2 // Aitken-Neville algorithm for evaluation of interpolating polynomial
3 // IN: t, y: (vectors of) interpolation data points
4 // x: (single) evaluation point
5 // OUT: value of interpolant in x
6 double ANipoleval(const VectorXd& t, VectorXd y, const double x) {
7     for (int i = 0; i < y.size(); ++i) {
8         for (int k = i - 1; k >= 0; --k) {
9             // Recursion (5.2.3.9)
10            y[k] = y[k + 1] + (y[k + 1] - y[k]) * (x - t[i]) / (t[i] - t[k]);
11        }
12    }
13    return y[0];
14 }

```

cost = $O(n^2)$ Memory efficient: $O(n)$

pol. interpolant never computed!

A-N. useful for *simultaneous* evaluations & updates**C++-code 5.2.3.12: Single point evaluation with data updates**

```

1 class PolyEval {
2     private:
3         // evaluation point and various internal data describing the
4         // polynomials
5     public:
6         // Constructor taking the evaluation point as argument
7         PolyEval(double x);
8         // Add another data point and update internal information
9         void addPoint(t, y);
10        // Value of current interpolating polynomial at x
11        double eval(void) const;
12 };

```

data update

Ex: $n = 3$

$n =$	0	1	2	3
t_0	$y_0 =: p_{0,0}(x)$	$\rightarrow p_{0,1}(x)$	$\rightarrow p_{0,2}(x)$	$\rightarrow p_{0,3}(x)$
t_1	$y_1 =: p_{1,1}(x)$	$\rightarrow p_{1,2}(x)$	$\rightarrow p_{1,3}(x)$	
t_2	$y_2 =: p_{2,2}(x)$	$\rightarrow p_{2,3}(x)$		
t_3	$y_3 =: p_{3,3}(x)$			

cost (single update) = $O(k)$, k = current #dataMemory efficient, cost(eval) = $O(1)$

⑤

▷ Review questions 5.2.3.14

7