

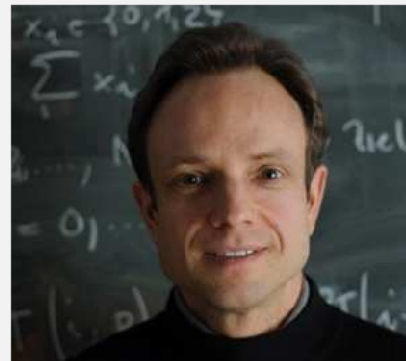
Course Video

Section 5.2.3.4: Newton Basis and Divided Differences

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 28, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.1, Section 5.2.2, Section 5.2.3.1, Section 5.2.3.2

Duration: 30 minutes

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.3. Polynomial Interpolation: Algorithms

5.2.3.4. Newton Basis and Divided Differences

C++code 5.2.3.21: Polynomial evaluation

```

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

```

evaluation in $N \gg 1$ points

(2)

Idea: Use of an "update-friendly" basis of $\mathcal{P}_n$

→ **Newton basis**

$$N_0(t) \equiv 1, \dots, N_k(t) = \prod_{i=0}^{k-1} (t - t_i), \quad k=1, \dots, n$$

$\{N_0, \dots, N_n\}$ is a basis of $\mathcal{P}_n$

$\in \mathcal{P}_k$ w/ leading coefficient=1

$$\triangleright N_k(t_j) = 0, \quad j=0, \dots, k-1 \quad (*)$$

$$IC: \quad a_j \in \mathbb{R}: a_0 N_0(t_j) + a_1 N_1(t_j) + \dots + a_n N_n(t_j) = y_j, \quad j=0, \dots, n. \quad (5.2.3.24)$$

⇔ **triangular** linear system

$$LSE: \quad \begin{bmatrix} N_0(t_0) & N_1(t_0) & \dots & N_n(t_0) \\ N_0(t_1) & N_1(t_1) & \dots & N_n(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ N_0(t_n) & N_1(t_n) & \dots & N_n(t_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

triangular LSE

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & (t_1 - t_0) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & (t_n - t_0) & \dots & \prod_{i=0}^{n-1} (t_n - t_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \quad (5.2.3.25)$$

$\triangleright a_k$ depends only on $(t_j, y_j) : 0 \leq j \leq k$
No change of $a_0, \dots, a_n$ when (t_{n+1}, y_{n+1}) added

§ 5.2.3.33 : Efficient evaluation of a polynomial in Newton form

$$p(t) = a_0 N_0(t) + a_1 N_1(t) + \dots + a_n N_n(t)$$

M.B.

$$= a_0 + a_1(t - t_0) + a_2(t - t_0)(t - t_1) + \dots + a_n \prod_{j=0}^{n-1} (t - t_j), \quad t \in \mathbb{R},$$

$$= a_0 + (\dots ((a_n(t - t_n) + a_{n-1})(t - t_{n-1}) + a_{n-2}) \dots + a_1)(t - t_1)$$

→ Horner-like scheme

C++-code 5.2.3.34: Divided differences evaluation by modified Horner scheme

```
// Evaluation of a polynomial in Newton form, that is, represented
// through the
// vector of its basis expansion coefficients with respect to the Newton
// basis
// (5.2.3.23).
Eigen::VectorXd evalNewtonForm(const Eigen::VectorXd &t, // nodes
                                const Eigen::VectorXd &a, // basis exp. coeffs
                                const Eigen::VectorXd &x) { // eval pts.

    const unsigned int n = a.size() - 1;
    const Eigen::VectorXd ones = VectorXd::Ones(x.size());
    Eigen::VectorXd p{a[n] * ones};
    for (int j = n - 1; j >= 0; --j) {
        p = (x - t[j] * ones).cwiseProduct(p) + a[j] * ones;
    }
    return p;
}
```

Cost = $O(n)$

§ 5.2.3.26 : Divided differences $\rightarrow$ compute a_j

$$p(t) = \sum_{j=0}^n a_j N_j(t)$$

The Newton basis polynomial $N_j(t)$ has degree j and leading coefficient 1

► a_j is the leading coefficient of the interpolating polynomial $p_{0,j}$.

"Partial polynomial interpolants"

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

$$p_{0,m}(t) = \sum_{j=0}^m a_j N_j(t) \quad \leftarrow$$

Notation : $\gamma[t_\ell, \dots, t_m] \triangleq$ leading coeff. of $p_{\ell,m}$

$$a_j = \gamma[t_0, \dots, t_j] \quad (*)$$

Remember : Aitken-Neville recursion

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

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

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

▷ Recursion for leading coefficients

$$y[t_i] = y_i$$

$$y[t_i, \dots, t_{i+k}] = \frac{y[t_{i+1}, \dots, t_{i+k}] - y[t_i, \dots, t_{i+k-1}]}{t_{i+k} - t_i} \quad (\text{recursion}). \quad (5.2.3.28)$$

↳ divided differences

A.N. - like scheme :

$$\begin{array}{l} t_0 \quad y[t_0] \\ t_1 \quad y[t_1] \\ t_2 \quad y[t_2] \\ t_3 \quad y[t_3] \end{array} \begin{array}{l} > y[t_0, t_1] \\ > y[t_1, t_2] \\ > y[t_2, t_3] \end{array} \begin{array}{l} > y[t_0, t_1, t_2] \\ > y[t_1, t_2, t_3] \end{array} > y[t_0, t_1, t_2, t_3]$$

$= a_j$ by $(*)$

↳ in place : $\gamma_j \rightarrow a_j$

(5.2.3.30)

C++ code 5.2.3.31: In-situ computation of divided differences

```
2 // IN: t = node set (mutually different)
3 // y = nodal values
4 // OUT: y = coefficients of polynomial in Newton basis
5 void divdiff(const VectorXd &t, VectorXd &y) {
6     const int n = y.size() - 1;
7     // Follow scheme (5.2.3.30), recursion (5.2.3.27)
8     for (int l = 1; l < n; ++l)
9         for (int j = n - l; j < n; ++j)
10             y[j] = (y[j] - y[j - 1]) / (t[j] - t[n - 1 - l]);
11 }
```

▷ Cost = $O(n^2)$

5

Example 5.2.3.35 : Class PolyEval

→ Combined : • adding data

• evaluation in many points

depend only on $t_0, \dots, t_{n-1}$

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & (t_1 - t_0) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & (t_n - t_0) & \dots & \prod_{i=0}^{n-1} (t_n - t_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}. \quad (5.2.3.25)$$

$$\begin{aligned} a_n &= \prod_{i=0}^{n-1} (t_n - t_i)^{-1} \left(y_n - \sum_{k=0}^{n-1} \prod_{i=0}^{k-1} (t_n - t_i) a_k \right) \\ &= \prod_{i=0}^{n-1} (t_n - t_i)^{-1} y_n - \sum_{k=0}^{n-1} \prod_{i=k}^{n-1} (t_n - t_i)^{-1} a_k \\ &= (\dots ((y_n - a_0) / (t_n - t_0) - a_1) / (t_n - t_1) - a_2) / \dots - a_{n-1} / (t_n - t_{n-1}). \end{aligned}$$

→ Horner-like formula

C++-code 5.2.3.37: Implementation of class PolyEval

```

1 PolyEval::PolyEval() {}
2
3 void PolyEval::addPoint(double td, double yd) {
4     t.push_back(td);
5     y.push_back(yd);
6     int n = t.size();
7     for (int j = 0; j < n - 1; j++)
8         y[n - 1] = ((y[n - 1] - y[j]) / (t[n - 1] - t[j]));
9 }
10
11 double PolyEval::operator()(double x) const {
12     double s = y.back();
13     for (int i = y.size() - 2; i >= 0; --i)
14         s = s * (x - t[i]) + y[i];
15     return s;
16 }

```

▷ cost = $O(n)$

→ a_n

▷ Review questions 5.2.3.39

⑦