

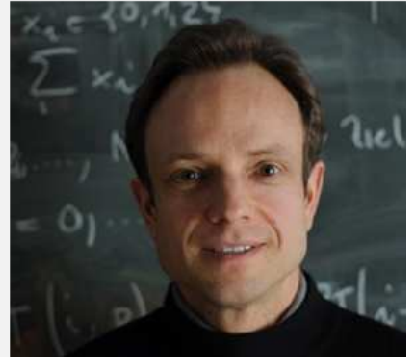
Course Video

Section 6.2.3.3: Chebychev Interpolation:
Computational Aspects

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 13, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- C++ and EIGEN
- Arithmetic in $\mathbb{C}$

Dependencies. Section 6.2.3.1, Section 4.3

Duration: 2 | minutes

VI. Approximation of Functions
in 1D

6.2. Approximation by Global Polynomials

6.2.3. Chebychev Interpolation

6.2.3.3. CI: Computational Aspects

C++ code 6.2.3.31: Definition of class for Chebychev interpolation

```

1 class ChebInterp {
2     private:
3         // various internal data describing Chebychev interpolating polynomial
4         // p
5     public:
6         // Constructor taking function f and degree n as arguments
7         template <typename Function>
8             PolyInterp(const Function &f, unsigned int n);
9         // Evaluation operator: y_j = p(x_j), j = 1, ..., m (m "large")
10        template <typename Vector>
11        void eval(const Vector &x, Vector &y) const;
12    };

```

$$\rightarrow L_f = \sum_{j=0}^n \alpha_j T_j \quad [\text{Chebychev expansion}]$$

(2)

§ 6.2.3.32 (Fast evaluation of Chebyshev expansion)

Idea: Use 3-term recursion

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x), \quad j \geq 2$$

$$\begin{aligned} p(x) &= \sum_{j=0}^{n-1} \alpha_j T_j(x) + \alpha_n T_n(x) \stackrel{(6.2.3.5)}{=} \sum_{j=0}^{n-1} \alpha_j T_j(x) + \alpha_n (2xT_{n-1}(x) - T_{n-2}(x)) \\ &= \sum_{j=0}^{n-3} \alpha_j T_j(x) + (\alpha_{n-2} - \alpha_n) T_{n-2}(x) + (\alpha_{n-1} + 2x\alpha_n) T_{n-1}(x). \end{aligned}$$

 $\hat{=}$ Another Chebyshev expansion, degree $n-1$ $\triangleright$ recursion

C++ code 6.2.3.34: Recursive evaluation of Chebyshev expansion (6.2.3.3)

```

2 // Recursive evaluation of a polynomial  $p = \sum_{j=1}^{n+1} a_j T_{j-1}$  at point  $x$ 
3 // based on (6.2.3.33)
4 // IN : Vector of coefficients  $a$ 
5 // evaluation point  $x$ 
6 // OUT: Value at point  $x$ 
7 double recclenshaw(const VectorXd& a, const double x) {
8     const VectorXd::Index n = a.size() - 1;
9     if (n == 0) return a(0); // Constant polynomial
10    else if (n == 1) return (x*a(1) + a(0)); // Value  $\alpha_1 * x + \alpha_0$ 
11    else {
12        VectorXd new_a(n);
13        new_a << a.head(n-2).a(n-2) - a(n), a(n-1) + 2*x*a(n);
14        return recclenshaw(new_a, x); // recursion
15    }
16 }  $\triangleright$  Cost  $= O(n)$ 

```

§ 6.2.3.36 (Computation of Chebyshev expansion of interpolant)

$$\text{I.C.: } p(t_k) = f(t_k), \quad k = 0, \dots, n, \quad \text{for } t_k := \cos\left(\frac{2k+1}{2(n+1)}\pi\right). \quad (6.2.3.37)$$

Chebyshev nodes

Idea: reduction to DFT by change of variables

$$t \in [-1, 1] \longmapsto s \in \mathbb{R} : t = \cos 2\pi s : 1\text{-periodic}$$

$$q(s) := \overbrace{p(\cos 2\pi s)}^t = \sum_{j=0}^n \alpha_j T_j(\cos 2\pi s) \stackrel{\text{Def. 6.2.3.3}}{=} \sum_{j=0}^n \alpha_j \cos(2\pi js)$$

$$= \sum_{j=0}^n \frac{1}{2} \alpha_j (\exp(2\pi i js) + \exp(-2\pi i js)) \quad [\text{by } \cos z = \frac{1}{2}(e^z + e^{-z})]$$

(6.2.3.38)

$$= \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i js), \quad \text{with } \beta_j := \begin{cases} 0 & , \text{ for } j = n+1, \\ \frac{1}{2} \alpha_j & , \text{ for } j = 1, \dots, n, \\ \alpha_0 & , \text{ for } j = 0, \\ \frac{1}{2} \alpha_{n-j} & , \text{ for } j = -n, \dots, -1. \end{cases}$$

 $\downarrow$
a trigonometric polynomial

$$\text{I.C.} \Rightarrow q(s_k) = f(t_k), \quad s_k = \frac{2k+1}{4(n+1)}, \quad k = 0, \dots, n$$

Chebyshev expansion for Chebyshev interpolants

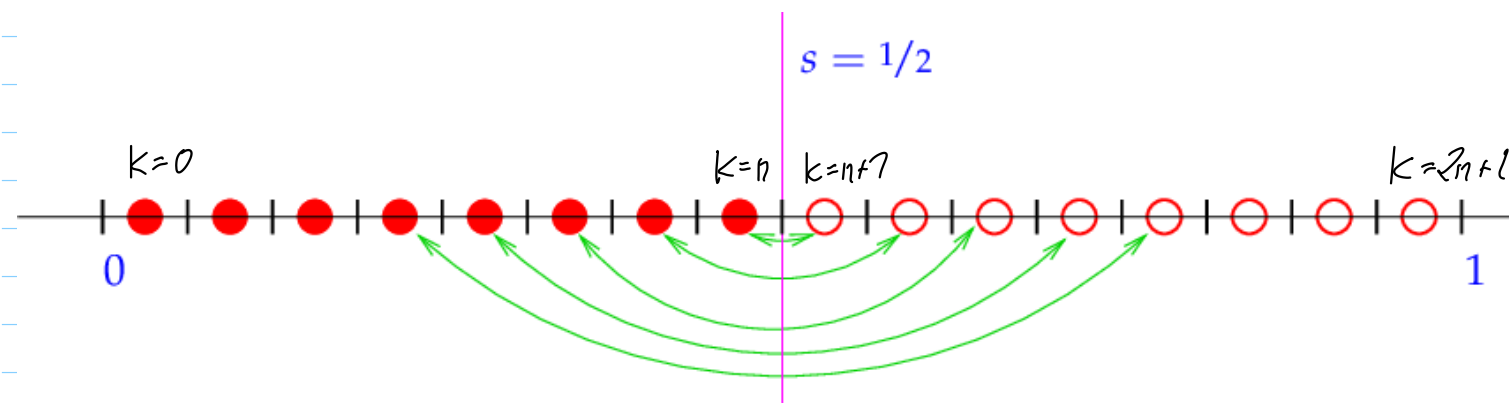
$\Updownarrow$

Trigonometric interpolation at equidistant points $\rightarrow$ Section 5.6.3 $\triangleright$ DFT

$$q(s) \text{ even} \Rightarrow q(s) = q(1-s)$$

$\triangleright$ Equivalent I.C. by reflection:

$$q\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right) = z_k := \begin{cases} y_k & , \text{ for } k = 0, \dots, n, \\ y_{2n+1-k} & , \text{ for } k = n+1, \dots, 2n+1. \end{cases} \quad (6.2.3.40)$$



$$q(s) = \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i s) :$$

$$q\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right) = \sum_{j=-n}^{n+1} \left(\beta_j \exp\left(-\frac{2\pi i j}{4(n+1)}\right) \right) \exp\left(-\frac{2\pi i}{2(n+1)} k j\right) = z_k.$$

$\Updownarrow$ [reindexing: $j \rightarrow j+n$]

$$\sum_{j=0}^{2n+1} \left(\beta_{j-n} \exp\left(-\frac{2\pi i (j-n)}{4(n+1)}\right) \right) \exp\left(-\frac{2\pi i}{2(n+1)} k j\right) = \exp\left(-\pi i \frac{nk}{n+1}\right) z_k, \quad k = 0, \dots, 2n+1.$$

$= \omega_{2(n+1)}^{kj} !$

$\Updownarrow$

$$\mathbf{F}_{2(n+1)} \mathbf{c} = \mathbf{b} \quad \text{with} \quad \mathbf{c} = \left[\beta_{j-n} \exp\left(-\frac{2\pi i (j-n)}{4(n+1)}\right) \right]_{j=0}^{2n+1}, \quad (6.2.3.41)$$

$$\mathbf{b} = \left[\exp\left(-\pi i \frac{nk}{n+1}\right) z_k \right]_{k=0}^{2n+1}.$$

$(2n+2) \times (2n+2)$ Fourier matrix, see (4.2.1.13)

$\triangleright$ Solve w/ FFT, cost = $O(n \log n)$

