Prof. R. Hiptmair
G. Alberti,
F. Leonardi

AS 2015

Numerical Methods for CSE

ETH Zürich
D-MATH

Problem Sheet 6

Problem 1. Evaluating the derivatives of interpolating polynomials (core problem)

In [1, Section 3.2.3.2] we learned about an efficient and "update-friendly" scheme for evaluating Lagrange interpolants at a single or a few points. This so-called Aitken-Neville algorithm, see [1, Code 3.2.31], can be extended to return the derivative value of the polynomial interpolant as well. This will be explored in this problem.

(1a) ⚀ Study the Aitken-Neville scheme introduced in [1, § 3.2.29].

(1b) ⚂ Write an efficient MATLAB functi7on

```
dp = dipoleval(t,y,x)
```

that returns the row vector $(p'(x_1), \ldots, p'(x_m))$, when the argument x passes $(x_1, \ldots, x_m)$, $m \in \mathbb{N}$ small. Here, p' denotes the *derivative* of the polynomial $p \in \mathcal{P}_n$ interpolating the data points (t_i, y_i), $i = 0, \ldots, n$, for pairwise different $t_i \in \mathbb{R}$ and data values $y_i \in \mathbb{R}$.

HINT: Differentiate the recursion formula [1, Eq. (3.2.30)] and devise an algorithm in the spirit of the Aitken-Neville algorithm implemented in [1, Code 3.2.31].

(1c) ⚁ For validation purposes devise an alternative, less efficient, implementation of `dipoleval` (call it `dipoleval_alt`) based on the following steps:

1. Use MATLAB's `polyfit` function to compute the monomial coefficients of the Lagrange interpolant.
2. Compute the monomial coefficients of the derivative.
3. Use `polyval` to evaluate the derivative at a number of points.

Use `dipoleval_alt` to verify the correctness of your implementation of `dipoleval` with `t = linspace(0,1,10)`, `y = rand(1,n)` and `x = linspace(0,1,100)`.

Problem 2. Piecewise linear interpolation

[1, Ex. 3.1.8] introduced piecewise linear interpolation as a simple linear interpolation scheme. It finds an interpolant in the space spanned by the so-called tent functions, which are *cardinal basis functions*. Formulas are given in [1, Eq. (3.1.9)].

(2a) Write a C++ class `LinearInterpolant` representing the piecewise linear interpolant. Make sure your class has an efficient internal representation of a basis. Provide a constructor and an evaluation `operator()` as described int he following template:

```
class LinearInterpolant {
    public:
        LinearInterpolant( /*  TODO: pass pairs */) {
         // TODO: construct your data from  (t_i, y_i)'s
        }

        double operator() (double x) {
            // TODO: return I(x)
        }
    private:
        // Your data here
};
```

HINT: Recall that C++ provides containers such as `std::vector` and `std::pair`.

(2b) Test the correctness of your code.

Problem 3. Evaluating the derivatives of interpolating polynomials (core problem)

This problem is about the Horner scheme, that is a way to efficiently evaluate a polynomial in a given point, see [1, Rem. 3.2.5].

(3a) Using the Horner scheme, write an efficient C++ implementation of a function

```
template <typename CoeffVec>
std::pair<double,double> evaldp ( const CoeffVec & c,
    double x )
```

which returns the pair $(p(x), p'(x))$, where p is the polynomial with coefficients in `c`. The vector `c` contains the coefficient of the polynomial in the monomial basis, using Matlab convention (leading coefficient in `c[0]`).

(3b) For the sake of testing, write a naive C++ implementation of the above function

```
template <typename CoeffVec>
std::pair<double,double> evaldp_naive ( const CoeffVec &
    c, double x )
```

which returns the same pair $(p(x), p'(x))$. This time, $p(x)$ and $p'(x)$ should be calculated with the simple sums of the monomials constituting the polynomial.

(3c) What are the asymptotic complexities of the two implementations?

(3d) Check the validity of the two functions and compare the runtimes for polynomials of degree up to $2^{20} - 1$.

Problem 4. Lagrange interpolant

Given data points $(t_i, y_i)_{i=1}^n$, show that the Lagrange interpolant

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

is given by:

$$p(x) = \omega(x) \sum_{j=0}^{n} \frac{y_j}{(x - t_j)\omega'(t_j)}$$

with $\omega(x) = \prod_{j=0}^{n}(x - t_j)$.

Issue date: 22.10.2015

Hand-in: 29.10.2015 (in the boxes in front of HG G 53/54).

Version compiled on: October 22, 2015 (v. 1.0).

References

[1] R. Hiptmair. *Lecture slides for course "Numerical Methods for CSE"*. http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf. 2015.