

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 5-4: Piecewise linear interpolation

[Lecture → Ex. 5.1.0.15] 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 [Lecture → Eq. (5.1.0.17)].

Study [Lecture → Ex. 5.1.0.15] before tackling this problem. Problem involves coding in C++.

(5-4.a) (20 min.)

Realize a C++ class

```
class LinearInterpolant {
public:
    LinearInterpolant(const Eigen::VectorXd &t, const
        Eigen::VectorXd &y);
    double operator(double x) const;
private:
    // Necessary data members here
};
```

representing the **piecewise linear interpolant** of the data points (t_i, y_i) passed to the constructor through two vectors $\mathbf{t}$ and $\mathbf{y}$ of equal length. In addition the class should provide the evaluation operator **double operator(double x) const**, which returns the value of the piecewise linear interpolant at $x \in \mathbb{R}$. Pay attention to efficient implementation.

Note that you must not assume that the nodes passed in $\mathbf{t}$ are sorted.

SOLUTION of (5-4.a):

Key to efficiency is the **sorting** of the data point according to their first coordinates (the nodes). This paves the way for using **binary search** for locating the relevant node interval for the evaluation.

C++11-code 5.4.1:

```
2  LinearInterpolant::LinearInterpolant(const Eigen::VectorXd &t,
3                                     const Eigen::VectorXd &y) {
4      assert(t.size() == y.size() && "Input sizes must match");
5      assert(t.size() > 1 && "Must specify at least two nodes");
6
7      // TODO: (6-4.a) Implement the constructor of the class. t and y
8      // may not be sorted. START
9
10     // helper lambda to transform two Eigen::VectorXd into one
11     // std::vector<std::pair<double, double>
12     auto VecToVecPair = [](const double &t_i, const double &y_i) -> pair {
13         return std::make_pair(t_i, y_i);
14     };
15
16     // perform transformation towards std::vector<std::pair<double,
17     // double>
18     i_points.resize(t.size());
19     std::transform(t.data(), t.data() + t.size(), y.data(), i_points.begin(),
20                   VecToVecPair);
21     // This is needed in std::sort
```

```

22 auto Ordering = [](const pair &P, const pair &Q) -> bool {
23     return P.first < Q.first;
24 };
25 // Sort the array
26 std::sort(i_points.begin(), i_points.end(), Ordering);
27 // END
28 }

```

Get it on  [GitLab \(linearinterpolant.hpp\)](#).

We leave the binary search to the standard library function `std::lower_bound`.

C++11-code 5.4.2:

```

2 double LinearInterpolant::operator()(double x) {
3     double l_x = 0.;
4     // TODO: (6-4.a) Implement an efficient evaluation operator.
5     // START
6
7     // Lambda for comparison of pair type,
8     // we want to compare only the
9     // first element of the pair, i.e.  $t_i$ 
10    auto Compare = [](const pair &P, double V) -> bool { return P.first < V; };
11
12    // Find the place  $i$  (as iterator), where  $t_i \geq x$ , notice that  $i\_points$ 
13    // *must* be sorted (this is needed in std::lower_bound)
14    // IMPORTANT: "lower_bound" performs a binary search
15    // on the data, provided the data has a random access iterator
16    // http://www.cplusplus.com/reference/iterator/RandomAccessIterator/
17    // this is crucial here, since evaluation operators must be fast
18    // this allows to reduce the complexity,
19    // from linear complexity to a logarithmic
20    auto it = std::lower_bound(i_points.begin(), i_points.end(), x, Compare);
21
22    // Bound checks, if  $i=0$  and  $t_0 \neq x$  (we are before the first node)
23    // or if  $x > t_n$  (we are after the last node).
24    // In such cases return 0 (*it and *(it-1) would be undefined in
25    // such cases)
26    if ((it == i_points.begin() && it->first != x) || it == i_points.end())
27        return 0;
28
29    // Actually compute the interpolated value, dist_ratio
30    // contains the distance from  $t_{i-1}$  to  $x$  (as a ratio)
31    double dist_ratio = (x - (it - 1)->first) / (it->first - (it - 1)->first);
32    l_x = (it - 1)->second * (1 - dist_ratio) + it->second * dist_ratio;
33    // END
34    return l_x;
35 }

```

Get it on  [GitLab \(linearinterpolant.hpp\)](#).

(5-4.b)  (10 min.) Discuss the asymptotic effort involved in the constructor and the evaluation operator of the class **LinearInterpolant** as the number n of data points becomes large.

SOLUTION of (5-4.b):

- The constructor involves sorting a sequence of length n , which requires an effort of $O(n \log n)$ for $n \rightarrow \infty$.
- The evaluation operator has to find the node interval $[t_i, t_{i+1}]$ containing x . This can be done with a binary search at an asymptotic cost of $O(\log n)$ for $n \rightarrow \infty$.



End Problem 5-4 , 30 min.