

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 6-2: Piecewise linear interpolation with knots different from nodes

In [Lecture → Ex. 5.1.0.15] we examined piecewise linear interpolation in the case where the interpolation nodes coincide with the abscissas of the corners of the interpolating polygon. However, that one is a very special situation. In this problem we consider a more general setting for piecewise linear interpolation.

Involves moderate implementation in C++.

We are given two ordered sets of real numbers to be read as points on the real line:

- (I) the **knots** $x_0 < x_1 < x_2 < \dots < x_n$,
- (II) the (interpolation) **nodes** $t_0 < t_1 < \dots < t_n$, $n \in \mathbb{N}$, satisfying $x_0 \leq t_j \leq x_n$, $j = 0, \dots, n$.

The space of piecewise linear **continuous** functions with respect to the knot set $\mathcal{N} := \{x_j\}_{j=0}^n$ is defined as:

$$\mathcal{S}_{1,\mathcal{N}} := \{s \in C^0([x_0, x_n]) : s(t) = \gamma_j t + \beta_j \text{ for } t \in]x_{j-1}, x_j], \gamma_j, \beta_j \in \mathbb{R} \ i = 1, \dots, n\}. \quad (6.2.1)$$

Compare [Lecture → Ex. 5.1.0.15] for the special case $x_j := t_j$.

On the function space $\mathcal{S}_{1,\mathcal{N}} \subset C^0(I)$, $I := [x_0, x_n]$, we consider the **interpolation problem**:

Find $s \in \mathcal{S}_{1,\mathcal{N}}$ such that $s(t_i) = y_i$, $i = 0, \dots, n$, for given values $y_i \in \mathbb{R}$.

This fits the general framework of [Lecture → § 5.1.0.21].

Below we set $I_0 := [x_0, x_1]$, $I_j :=]x_{j-1}, x_{j+1}]$, $j = 1, \dots, n-1$, $I_n := [x_{n-1}, x_n]$.

(6-2.a) $\square$ Show that the interpolation problem may not have a solution for some values y_j if a single interval $[x_j, x_{j+1}]$, $j = 0, \dots, n-1$, contains more than 2 nodes t_i .

SOLUTION of (6-2.a):

In general a linear function cannot interpolate three data points.

▲

(6-2.b) $\square$ Show that the interpolation problem can have a unique solution for any data values y_j *only if* each interval I_j , $j = 0, \dots, n$, contains at least one node.

HINT 1 for (6-2.b): You may appeal to the linear system of equations that arises from the above interpolation problem when using the “tent basis function” basis of $\mathcal{S}_{1,\mathcal{N}}$ from [Lecture → Fig. 149]. $\lrcorner$

SOLUTION of (6-2.b):

If the condition is violated, there will be a “tent basis function” whose support does not contain a node. The corresponding column in the system matrix of the LSE [Lecture → Eq. (5.1.0.22)] will vanish and the matrix cannot be invertible.

▲

(6-2.c)  Show that the interpolation problem has a unique solution for all data values y_j if each of the closed intervals $]x_0, x_1[,]x_1, x_2[,]x_2, x_3[, \dots,]x_{n-1}, x_n[$ contains at least one node t_j .

HINT 1 for (6-2.c): Since we have to distribute $n + 1$ nodes on n intervals, there will be a single interval that contains 2 nodes. ┘

SOLUTION of (6-2.c):

The interpolant is fixed in that interval that contains two nodes. Thus, its value will be known at the endpoints of that interval, which, due to continuity, also fixes the values at the endpoints of adjacent intervals. Since they contain one node each, the interpolant is also fixed there. Continue this argument until you reach the endpoints x_0 and x_n .



(6-2.d)  Implement a C++ function

```
VectorXd tentBasCoeff(const VectorXd &x, const VectorXd &t,
                     const VectorXd &y);
```

that returns the vector of values $s(x_j)$ of the interpolant s of the data points (t_i, y_i) , $i = 0, \dots, n$ in $\mathcal{S}_{1\mathcal{N}}$ in the knots x_j , $j = 0, \dots, n$. The argument vectors x , t , and y pass the x_j , t_j , and values y_j , respectively.

The function should first check whether the condition formulated in Sub-problem (6-2.c) is satisfied. You must not take for granted that knots or nodes are sorted already.

HINT 1 for (6-2.d): While you check the condition of Sub-problem (6-2.c), you can also find the interval $]x_j, x_{j+1}[$ which contains 2 nodes. There you can compute γ_j and β_j of (6.2.1). Given that $s(t) = \gamma_j t + \beta_j$, you can compute $s(x_j)$ and $s(x_{j+1})$. You are then able to compute γ_j and β_j in all the other intervals. ┘

SOLUTION of (6-2.d):

C++11-code 6.2.2: Solution of (6-2.d)

```
2 VectorXd PwLinIP::tentBasCoeff(const VectorXd &x, const VectorXd &t,
3                               const VectorXd &y) const
4
5 {
6     // Initialization
7     size_t n = t.size();
8     auto x_indices = order(x);
9     auto t_indices = order(t);
10    // You can also implement a solution which does not need
11    // sorted vectors and e.g. for each knot  $x_j$  looks
12    // for the closest node  $t_{i1}$  and the next closest node  $t_{i2}$ .
13    // However, such solution will not become more efficient
14    // if you give as input already sorted vectors: for each knot  $x_j$ 
15    // you will always have to iterate along the sorted vector  $t$ 
16    // to find the included node  $t_i$ .
17
18    VectorXd s = VectorXd::Zero(n);
19
20    // Check condition of subproblem 5.5.c
```

```

21  size_t i = 0;
22  size_t k = 0;
23  for(size_t j=0; j<(n-1); ++j) {
24
25      bool intervalOK = false;
26      while(i < n) {
27
28          bool inInterval = (x(x_indices[j]) < t(t_indices[i])) &&
29                          (t(t_indices[i]) < x(x_indices[j+1]));
30
31          if(inInterval) {
32              intervalOK = true;
33              if(i == j) { // Index of interval which contains 2
34                          // nodes
35                          //  $t_i$  and  $t_{i+1}$ . After that, we have
36                          //  $i > j$ ...
37                          k = j;
38                      }
39                      break;
40              } else {
41                  ++i;
42              }
43          }
44          if(!intervalOK) {
45              std::exit(EXIT_FAILURE);
46          }
47      }
48
49      // 1. Find slope  $\gamma$  and intercept  $\beta$ 
50      // in interval with 2 nodes  $k$ 
51      // 2. Find  $s_k$  and  $s_{k+1}$ 
52      double gamma = (y(t_indices[k+1]) - y(t_indices[k])) /
53                  (t(t_indices[k+1]) - t(t_indices[k]));
54      double beta = y(t_indices[k]) - gamma * t(t_indices[k]);
55
56      s(x_indices[k]) = gamma * x(x_indices[k]) + beta;
57      s(x_indices[k+1]) = gamma * x(x_indices[k+1]) + beta;
58
59      // Find intercept, slope and value  $s$  at the lower bound  $x$ 
60      // for all intervals on the left of interval with 2 nodes  $k$ 
61      for(int j=k-1; j>=0; --j) {
62          gamma = (s(x_indices[j+1]) - y(t_indices[j])) /
63                  (x(x_indices[j+1]) - t(t_indices[j]));
64          beta = y(t_indices[j]) - gamma * t(t_indices[j]);
65
66          s(x_indices[j]) = gamma * x(x_indices[j]) + beta;
67      }
68      // Find intercept, slope and value  $s$  at the upper bound  $x$ 
69      // for all intervals on the right of interval with 2 nodes  $k$ 
70      for(int j=k+2; j<n; ++j) {
71          gamma = (y(t_indices[j]) - s(x_indices[j-1])) /
72                  (t(t_indices[j]) - x(x_indices[j-1]));
73          beta = s(x_indices[j-1]) - gamma * x(x_indices[j-1]);
74
75          s(x_indices[j]) = gamma * x(x_indices[j]) + beta;
76      }
77
78      return s;
79  }

```

Get it on  GitLab (interleaveLIP.cpp).

(6-2.e) Implement a C++ class

```

class PwLinIP {
public:
    PwLinIP(const VectorXd &x, const VectorXd &t, const VectorXd &y);
    double operator()(double arg) const;
private:
    ...
};

```

that realizes an interpolator class in the spirit of [Lecture → Rem. 5.1.0.11]. The meanings of the arguments of the constructor are the same as for the function `tentBasCoeff` from Sub-problem (6-2.d). Pay attention to the efficient implementation of the evaluation operator!

HINT 1 for (6-2.e): Sorting of nodes and knots should be performed in the constructor using `std::sort`. To sort the data points, either wrap them into suitable objects for which you supply a comparison operator or exploit function `order` supplied in the code, which returns the indices of `std::sort`. Sorting is essential for the efficient implementation of the evaluation operator in the case $n \gg 1$. ┘

HINT 2 for (6-2.e): The function `tentBasCoeff` can be reused as a private method of the class. ┘

SOLUTION of (6-2.e):

C++11-code 6.2.3: Full declaration of class PwLinIP

```

2  class PwLinIP {
3  public:
4      /*!
5      * \brief
6      * \param[in] x Vector of knots
7      * \param[in] t Vector of nodes
8      * \param[in] y Vector of values of interpolant in nodes
9      */
10     PwLinIP(const VectorXd &x, const VectorXd &t, const VectorXd &y);
11
12     /*!
13     * \brief operator() evaluate interpolant at arg.
14     */
15     double operator()(double arg) const;
16 private:
17     VectorXd x_;
18     VectorXd t_;
19     VectorXd y_;
20     VectorXd s_;
21     /*!
22     * \brief Compute values of interpolant in knots x from (ti, yi)
23     * \param[in] x Vector of knots
24     * \param[in] t Vector of nodes
25     * \param[in] y Vector of values of interpolant in nodes t
26     * \param[out] s Vector of values of interpolant in knots x
27     */
28     VectorXd tentBasCoeff(const VectorXd &x, const VectorXd &t,
29                          const VectorXd &y) const;

```

```
30 };
```

Get it on  [GitLab \(interleaveLIP.cpp\)](#).

C++11-code 6.2.4: Implementation of `PwLinIP::PwLinIP` (constructor)

```
2 PwLinIP::PwLinIP(const VectorXd &x, const VectorXd &t,
3                 const VectorXd &y)
4 {
5     assert(t.size() == y.size() && t.size() == x.size() &&
6           "x, t, y must have same size!");
7
8     size_t n = t.size();
9     x_.resize(n);
10    t_.resize(n);
11    y_.resize(n);
12
13    auto x_indices = order(x);
14    for(size_t i=0; i<n; ++i) {
15        x_(i) = x[x_indices[i]];
16    }
17
18    auto t_indices = order(t);
19    for(size_t i=0; i<n; ++i) {
20        t_(i) = t[t_indices[i]];
21        y_(i) = y[t_indices[i]];
22    }
23
24    s_ = tentBasCoeff(x_, t_, y_);
25 }
```

Get it on  [GitLab \(interleaveLIP.cpp\)](#).

C++11-code 6.2.5: Implementation of `PwLinIP::operator()`

```
2 double PwLinIP::operator()(double arg) const
3 {
4     if(arg < x_(0) || arg > x_(x_.size()-1)) {
5
6         return 0;
7     } else {
8         std::vector<double> x(x_.size()-1); // Exclude x0 from search of
9         interval including arg
10        VectorXd::Map(&x.front(), x_.size()-1) = x_.tail(x_.size()-1);
11        size_t j = std::lower_bound(x.begin(), x.end(), arg) - x.begin() + 1; //
12        Binary search
13        // '+1' is needed to restore indexing from x0
14
15        double gamma = (s_(j) - s_(j-1)) / (x_(j) - x_(j-1));
16        double beta = s_(j-1) - gamma * x_(j-1);
17
18        return gamma * arg + beta;
19    }
20 }
```

Get it on  [GitLab \(interleaveLIP.cpp\)](#).



(6-2.f) ☹ Implement a C++ code that creates plots of the **cardinal basis functions** for interpolation in $\mathcal{S}_{1,\mathcal{N}}$

- with know set $\mathcal{N} := \{0, 1, 2, 3, \dots, 9, 10\}$
- and for interpolation nodes $t_0 := 0, t_j = j - \frac{1}{2}, j = 1, \dots, 10$.

Recall that the cardinal basis function $b_k \in \mathcal{S}_{1,\mathcal{N}}$ is characterized by the conditions:

$$b_k(t_j) = \delta_{kj}, \quad k, j = 0, \dots, 10 \quad (6.2.6)$$

HINT 1 for (6-2.f): You should use the **Figure** class for plotting. ┘

SOLUTION of (6-2.f):

C++11-code 6.2.7: Solution of (6-2.f)

```

2  int main() {
3      // Initialization
4      size_t n = 11;
5
6      // Nodes
7      VectorXd x = VectorXd::LinSpaced(n, 0, 10);
8      VectorXd t(n);
9      t(0) = 0; t.tail(n-1).setLinSpaced(n-1, 0.5, 9.5);
10
11     mgl::Figure fig;
12     fig.xlabel("t");
13     fig.ylabel("y");
14
15     // Plot colors
16     const unsigned int M = 3;
17     std::vector<std::string> C = {"b", "r", "m"};
18
19     // Loop over basis functions
20     for(unsigned int i = 0; i < M; ++i) {
21         // Basis function  $e_i$ 
22         VectorXd y = VectorXd::Zero(n);
23         y(i) = 1;
24
25         // Create interpoland with values y
26         PwLinIP cardinalBasis(x, t, y);
27
28         // Plotting values
29         unsigned int N = 1000;
30         VectorXd xval = VectorXd::LinSpaced(N, 0, 10);
31         VectorXd s(N);
32         for(size_t j=0; j<N; ++j) {
33             s(j) = cardinalBasis(xval(j));
34         }
35
36         std::stringstream ss;
37         ss << "Basis k = " << i;
38         fig.plot(xval, s, C[i].c_str()).label(ss.str().c_str());
39         fig.legend();
40
41         fig.title("Tent basis functions");
42         fig.save("tent_basis_functions.eps");
43     }
44 }
```

45 }

Get it on  GitLab (interleaveLIP.cpp).

Tent basis functions

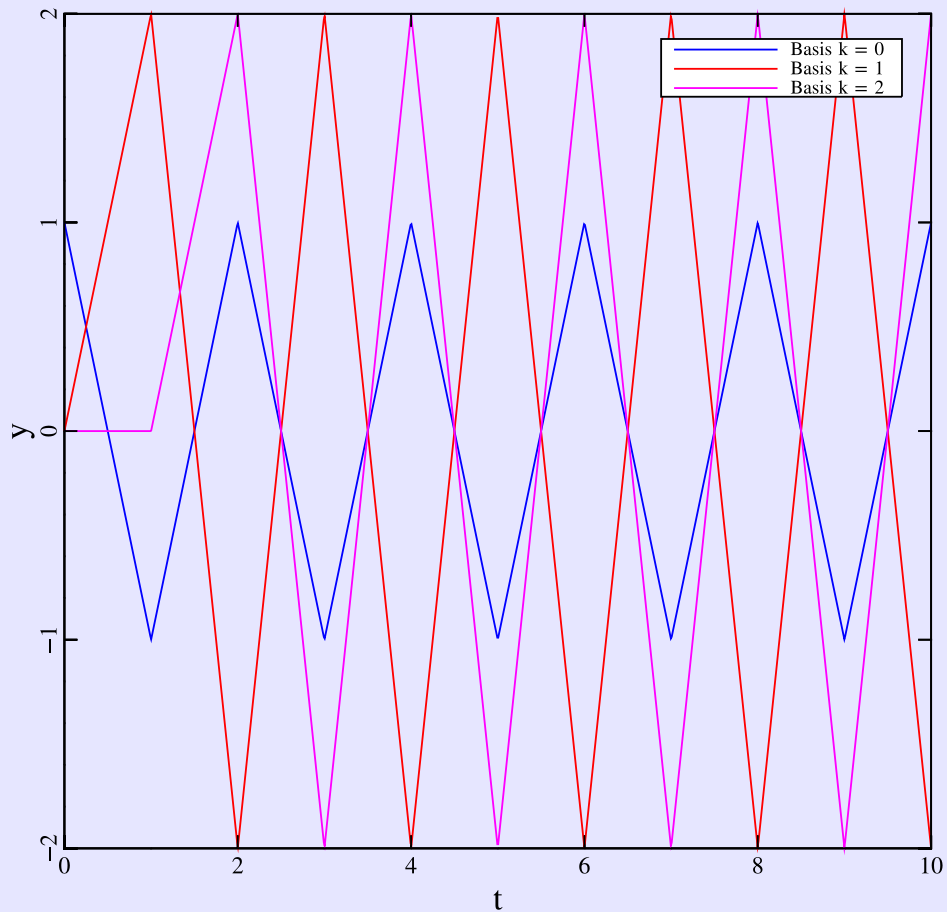


Fig. 55



End Problem 6-2