

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 6-7: Piecewise cubic Hermite Interpolation of Functions

The variant of **piecewise cubic Hermite interpolation** studied in [Lecture → Section 6.6.2] was a generalized interpolation scheme in the sense that it required point values of the derivative of the interpoland. Conversely, for piecewise cubic Hermite data interpolation we had to use reconstructed slopes. This problem examines the use of reconstructed slopes also for the purpose of function approximation.

Relies on [Lecture → Section 5.3.3] and [Lecture → Section 6.6.2] and involves C++ coding.

Get it on  [GitLab \(pchi.hpp\)](https://gitlab.com/pchi/hpp).

Piecewise cubic Hermite interpolation of a function $f \in C^1([a, b])$ with exact slopes on a mesh

$$\mathcal{M} := \{a = x_0 < x_1 < \dots < x_m = b\}$$

was defined in [Lecture → Def. 6.6.2.1]. For $f \in C^4([a, b])$ it enjoys h -convergence with rate 4 as we have seen in [Lecture → Exp. 6.6.2.2] and which is stated in [Lecture → Thm. 6.6.2.3].

Theorem [Lecture → Thm. 6.6.2.3]. Convergence of approximation by cubic Hermite interpolation

Let s be the cubic Hermite interpolant of $f \in C^4([a, b])$ on a mesh $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\}$ according to [Lecture → Def. 6.6.2.1]. Then

$$\|f - s\|_{L^\infty([a, b])} \leq \frac{1}{4!} h_{\mathcal{M}}^4 \|f^{(4)}\|_{L^\infty([a, b])},$$

with the meshwidth $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}|$.

Now we consider cases, where perturbed or reconstructed slopes are used. For instance, this was done in the context of monotonicity preserving piecewise cubic Hermite interpolation as discussed in [Lecture → Section 5.3.3.2].

(6-7.a)  (30 min.) Assume that piecewise cubic Hermite interpolation is based on perturbed slopes, that is, the piecewise cubic function s on the mesh $\mathcal{M}$ satisfies:

$$s(x_j) = f(x_j) \quad , \quad s'(x_j) = f'(x_j) + \delta_j, \quad (6.7.1)$$

where the perturbations $\delta_j \in \mathbb{R}$ may depend on $\mathcal{M}$, too.

Now we consider a sequence of meshes with meshwidth $h_{\mathcal{M}} \rightarrow 0$. Which rate of asymptotic h -convergence of the maximum norm $\|f - s\|_{L^\infty([a, b])}$ of the approximation error can be expected, if we know that for all j that

$$|\delta_j| = O(h_{\mathcal{M}}^\beta) \quad \text{as } h_{\mathcal{M}} \rightarrow 0 \quad \text{for some } \beta \in \mathbb{N}_0, \quad (6.7.2)$$

for mesh-width $h_{\mathcal{M}} \rightarrow 0$.

HINT 1 for (6-7.a): Use a local generalized cardinal basis representation, cf. [Lecture → § 5.3.3.3] and [Lecture → Eq. (5.3.3.4)]. Also invoke [Lecture → Thm. 6.6.2.3]. 

SOLUTION of (6-7.a):

Let s be the piecewise cubic polynomial interpolant of f . Using the notations from [Lecture → Eq. (5.3.3.5)]

$$\begin{aligned} H_1(t) &:= \phi\left(\frac{x_i - t}{h_i}\right), \\ H_2(t) &:= \phi\left(\frac{t - x_{i-1}}{h_i}\right), & h_i &:= x_i - x_{i-1}, \quad i = 1, \dots, m, \\ H_3(t) &:= -h_i \psi\left(\frac{x_i - t}{h_i}\right), & \phi(\tau) &:= 3\tau^2 - 2\tau^3, \\ H_4(t) &:= h_i \psi\left(\frac{t - x_{i-1}}{h_i}\right), & \psi(\tau) &:= \tau^3 - \tau^2, \end{aligned}$$

we can rewrite $s|_{[x_{i-1}, x_i]}$ using the local representation with cardinal basis polynomials: for $x_{i-1} \leq t \leq x_i, i = 1, \dots, m$,

$$\begin{aligned} s(t) &= y_{i-1}H_1(t) + y_iH_2(t) + c_{i-1}H_3(t) + c_iH_4(t) \\ &= y_{i-1}H_1(t) + y_iH_2(t) + (f'(t_{i-1}) + \delta_{i-1})H_3(t) + (f'(t_i) + \delta_i)H_4(t) \\ &= y_{i-1}H_1(t) + y_iH_2(t) + f'(t_{i-1})H_3(t) + f'(t_i)H_4(t) + \delta_{i-1}H_3(t) + \delta_iH_4(t). \end{aligned}$$

Hence, if we denote by $\tilde{s}$ the Hermite interpolant with exact slopes, taking the maximum over all mesh cells and using the estimate from [Lecture → Thm. 6.6.2.3]

$$\begin{aligned} \|f - s\|_{L^\infty([a,b])} &\leq \|f - \tilde{s} + \tilde{s} - s\|_{L^\infty([a,b])} \leq \|f - \tilde{s}\|_{L^\infty([a,b])} + \|\tilde{s} - s\|_{L^\infty([a,b])} \\ &\leq O(h_{\mathcal{M}}^4) + \max_i \|\delta_{i-1}H_3(t) + \delta_iH_4(t)\|_{L^\infty([t_{i-1}, t_i])} \\ &= O(h_{\mathcal{M}}^4) + O(h_{\mathcal{M}}^{\beta+1}) = O(h_{\mathcal{M}}^{\min(4, \beta+1)}), \end{aligned}$$

since

$$\|H_3(t)\|_{L^\infty([x_{i-1}, x_i])} = \|H_4(t)\|_{L^\infty([x_{i-1}, x_i])} = O(h).$$

To see this one easily computes that the functions H_3, H_4 attain their maximum at $t = x_i + \frac{2}{3}h_i$ resp. minimum at $t = x_i + \frac{1}{3}h_i$, with value $h_i((\frac{2}{3})^3 - (\frac{2}{3})^2)$. See also [Lecture → Fig. 161].



(6-7.b) We define a “strange” piecewise cubic interpolant:

Definition 6.7.3. Flat-point piecewise cubic interpolant

Given $f \in C^0([a, b])$ and a mesh $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\}$ the **flat-point piecewise cubic interpolant** $s : [a, b] \rightarrow \mathbb{R}$ is defined as

$$s|_{[x_{j-1}, x_j]} \in \mathcal{P}_3, \quad j = 1, \dots, m, \quad s(x_j) = f(x_j), \quad s'(x_j) = 0, \quad j = 0, \dots, m.$$

Implement a C++ function

```
template <typename Function>
Eigen::VectorXd fppchip(Function &&f,
    const Eigen::VectorXd &x, const Eigen::VectorXd &t);
```

that returns the vector $[s(t_k)]_{k=1}^N$, where

- the argument x passes the $m + 1$ (ordered) nodes of a mesh $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\}$,
- s is the flat-point piecewise cubic interpolant of f according to Def. 6.7.3,
- the (sorted!) points $t_k \in [x_0, x_m]$ are supplied as components of the vector t .

Remark. Please refer to [Lecture $\rightarrow$ Thm. 5.3.3.16] to understand the motivation for introducing the “flat-point piecewise cubic interpolant”. This is the only linear interpolation operator into C^1 -functions that is locally monotonicity preserving.

SOLUTION of (6-7.b):

C++ code 6.7.4: Sub-problem (6-7.b): Computation of cubic Hermite interpolant

```

2  template <typename Function>
3  Eigen::VectorXd fppchip(Function &&f, const Eigen::VectorXd &t,
4                          const Eigen::VectorXd &x) {
5      Eigen::VectorXd ret{x.size()};
6      // TO DO: compute ret.
7      // START
8
9      Eigen::VectorXd y = t.unaryExpr(f);
10     int i_star = 0;
11     double tmp, h_tmp, t1, t2, y1, y2, a1, a2, a3;
12     for (int j = 0; j < x.size(); j++) {
13         // Stores the current interval index and some temporary variable
14         // Find the interval for x_j while assuming that x values are sorted
15         // and within [t_min, t_max]
16         if (t(i_star) <= x(j) && x(j) <= t(i_star + 1)) {
17             t1 = t(i_star);
18             t2 = t(i_star + 1);
19             h_tmp = t2 - t1; // h_tmp = h_ for equidistant grid
20             y1 = y(i_star);
21             y2 = y(i_star + 1);
22             a1 = y2 - y1;
23             a2 = a1;
24             a3 = -a1 - a2;
25
26             // Compute s(x(j))
27             tmp = (x(j) - t1) / h_tmp;
28             ret(j) = y1 + (a1 + (a2 + a3 * tmp) * (tmp - 1.)) * tmp;
29
30             ++j;
31         } else {
32             // Otherwise, go to next interval
33             ++i_star;
34             // Terminate if outside any interval
35             if (i_star >= t.size() - 1) break;
36             continue;
37         }
38     }
39
40     // END
41
42     return ret;
43 }

```

Get it on  GitLab (pchi.hpp).



(6-7.c) 📄 (20 min.) [depends on Sub-problem (6-7.b)]

For the function $f(t) = \frac{1}{1+t^2}$ and the interval $[-5, 5]$ empirically determine (qualitatively and quantitatively) the asymptotic convergence (in terms of meshwidth $h_M \rightarrow 0$) of the flat-point piecewise cubic interpolants according to Def. 6.7.3 on sequences of *equidistant meshes*.

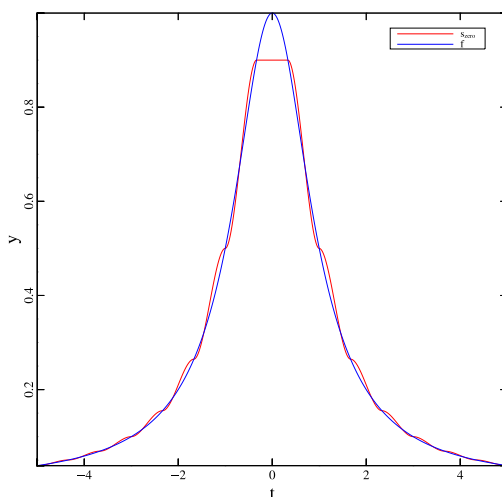
To that end study the maximum norm of the interpolation error in numerical experiments. Approximate those maximum norms by sampling in 10000 equidistant points. Implement a C++ function

std::vector<double> fppchipConvergence(void);

that tabulates and returns the error norms for meshes with 4, 8, 16, 32, 64, 128, 256, 512 nodes.

SOLUTION of (6-7.c):

Interpolant with zero slope



◁ Flat-point piecewise cubic interpolant of the Runge function $f(t) = \frac{1}{1+t^2}$ in $[-5, 5]$.

Obviously this interpolant provides a pathetic approximation.

Fig. 58

C++ code 6.7.5: Sub-problem (6-7.c): Convergence plot for flat-point piecewise cubic interpolant

```
2  std::vector<double> fppchipConvergence() {
3      // Interpoland
4      auto f = [](double x) { return 1. / (1. + x * x); };
5
6      double a = 5; // Interval bounds will be (-a,a)
7      int M = 1000; // Number of points in which to evaluate the interpoland
8
9      // Precompute values at which evaluate f
10     Eigen::VectorXd x = Eigen::VectorXd::LinSpaced(M, -a, a);
11     Eigen::VectorXd fx = x.unaryExpr(f);
12
13     // Store error and number of nodes
14     std::vector<double> N_nodes, err_zero;
15
16     // TO DO: compute err_zero.
17     // START
```

```

18
19 for (int i = 4; i <= 512; i = i << 1) {
20     // Define subintervals and evaluate f there (find pairs (t,y))
21     Eigen::VectorXd t = Eigen::VectorXd::LinSpaced(i, -a, a);
22
23     // compute reconstructed slopes
24     Eigen::VectorXd s_zero_x = fppchip(f, t, x);
25
26     // Compute infinity norm of error
27     err_zero.push_back((s_zero_x - fx).lpNorm<Eigen::Infinity>());
28     N_nodes.push_back(i);
29
30     if (i == 16) {
31         plt::figure();
32         plt::title("Interpolant with zero slope");
33         plt::xlabel("t");
34         plt::ylabel("y");
35         plt::plot(x, s_zero_x, "r", {"label", "s_zero"});
36         plt::plot(x, fx, "b--", {"label", "f"});
37         plt::legend();
38         plt::savefig("./cx_out/p_zero.png");
39     }
40 }
41
42 // END
43
44 return err_zero;
45 }

```

Get it on  GitLab (pchi.hpp).

No. of nodes	error norm
4	0.735269
8	0.337813
16	0.099975
32	0.0253299
64	0.0100745
128	0.00494491
256	0.00244481
512	0.00119703

The table shows the corresponding error for each node of the flat-point piecewise cubic interpolation.

Doubling the numbers of mesh nodes, that is, halving the mesh width, leads to a reduction of the maximum norm of the interpolation error by a factor of 2. Thus, we observe asymptotic algebraic h -convergence of $\|f - s\|_{L^\infty([a,b])}$ with rate 1.



(6-7.d)  (10 min.) [depends on Sub-problem (6-7.a) and Sub-problem (6-7.c)]

Invoke the result of Sub-problem (6-7.a) to give a theoretical explanation for the observations made in Sub-problem (6-7.c).

SOLUTION of (6-7.d):

According to Sub-problem (6-7.a), since $s'(x_j) = f'(x_j) - f'(x_j)$, i.e. $|\delta_j| = O(1)$, $\beta = 0$, the rate of asymptotic algebraic convergence is limited to $O(h)$.



We now define a piecewise cubic Hermite interpolant with slopes reconstructed from sampled function values.

Definition 6.7.6. Piecewise cubic Hermite interpolant with reconstructed slopes

Given $f \in C^0([a, b])$ and a mesh $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\}$ the **piecewise cubic Hermite interpolant with reconstructed slopes** $s : [a, b] \rightarrow \mathbb{R}$ is defined as

$$s|_{[x_{j-1}, x_j]} \in \mathcal{P}_3, \quad j = 1, \dots, m, \quad s(x_j) = f(x_j), \quad j = 0, \dots, m,$$

$$s'(x_j) = \begin{cases} \frac{-f(x_2) + 4f(x_1) - 3f(x_0)}{2h} & \text{for } j = 0, \\ \frac{f(x_{j+1}) - f(x_{j-1}))}{2h} & \text{for } j = 1, \dots, m-1, \\ \frac{3f(x_m) - 4f(x_{m-1}) + f(x_{m-2}))}{2h} & \text{for } j = m. \end{cases}$$

A class for piecewise cubic Hermite interpolation of continuous functions according to Def. 6.7.6 is defined as follows:

```
class CubicHermiteInterpolant {
public:
    template <typename Function>
    CubicHermiteInterpolant(Function &&f, const Eigen::VectorXd &t);
    virtual ~CubicHermiteInterpolant(void) = default;
    Eigen::VectorXd eval(const Eigen::VectorXd &x) const;
private:
    Eigen::VectorXd t_; // sorted mesh nodes
    Eigen::VectorXd y_; // function values at mesh nodes
    Eigen::VectorXd c_; // slopes at mesh nodes
};
```

The constructor expects a functor f with signature `std::function<double(double)>` and a (sorted!) vector t of nodes. Its evaluation method `CubicHermiteInterpolant::eval()` returns $s(x_k)$ for the (sorted!) components of the vector passed in x . Here $s \in C^1([a, b])$ is the piecewise cubic Hermite interpolant as defined in Def. 6.7.6.

(6-7.e)  (25 min.) Provide an implementation of the class **CubicHermiteInterpolant**.

SOLUTION of (6-7.e):

C++ code 6.7.7: Sub-problem (6-7.e): Definition of Hermite interpolant-class

```
2 class CubicHermiteInterpolant {
3 public:
4     /*!
5      * \brief Construct the slopes from the data.
6      * Use finite-differences or setting  $s'(x_j) = 0$ .
7      * \param[in] t Vector of nodes (assumed equidistant and sorted).
8      * \param[in] y Vector of values at nodes  $t$ .
9      * \param[in] s Flag to set if you want to reconstruct or set the slopes to
10     *zero.
11     */
12     template <typename Function>
13     CubicHermiteInterpolant(Function &&f, const Eigen::VectorXd &t);
```

```

14  virtual ~CubicHermiteInterpolant() = default;
15
16  /*!
17   * \brief Evaluate the interpolant at the nodes x.
18   * The input is assumed sorted, unique and inside the interval
19   * \param[in] x The vector of points  $x_i$  where to compute  $s(x_i)$ .
20   * \return Values of interpolant at x (vector)
21   */
22  Eigen::VectorXd eval(const Eigen::VectorXd &x) const;
23
24  private:
25  // Provided nodes and values (t,y) to compute spline,
26  // Eigen vectors, c contains slopes
27  // All have the same size n
28  Eigen::VectorXd t_, y_, c_;
29  // Difference  $t(i) - t(i-1)$ 
30  double h_;
31  // Size of t, y and c.
32  int n_;
33 };

```

Get it on  GitLab (pchi.hpp).

C++ code 6.7.8: Sub-problem (6-7.e): Constructor of Hermite interpolant-class

```

2  template <typename Function>
3  CubicHermiteInterpolant::CubicHermiteInterpolant(Function &f,
4  const Eigen::VectorXd &t)
5  : t_(t), y_(t.unaryExpr(f)), c_(t.size()) {
6  // Sanity check
7  n_ = t_.size();
8  assert(n_ == y_.size() && "t and y must have same dimension.");
9  assert(n_ >= 3 && "Need at least three nodes.");
10 h_ = t_(1) - t_(0);
11
12 // Reconstruction of the slope,
13 // TODO: implement reconstruction of slopes.
14 // START
15 // First order alternative:
16 c_(0) = (-1 * y_(2) + 4 * y_(1) - 3 * y_(0)) / 2 / h_;
17 for (int i = 1; i < n_ - 1; ++i) {
18     c_(i) = (y_(i + 1) - y_(i - 1)) / 2 / h_;
19 }
20
21 // First order alternative:
22 c_(n_ - 1) = (3 * y_(n_ - 1) - 4 * y_(n_ - 2) + 1 * y_(n_ - 3)) / 2 / h_;
23 // END
24 }

```

Get it on  GitLab (pchi.hpp).

C++ code 6.7.9: Sub-problem (6-7.e): Implementation of constructor of evaluation operator

```

2  Eigen::VectorXd CubicHermiteInterpolant::eval(const Eigen::VectorXd &x) const {
3  Eigen::VectorXd ret(x.size());
4
5  // TO DO: implement evaluation function.
6  // START

```

```

7
8  int i_star = 0;
9  double tmp, h_tmp, t1, t2, y1, y2, c1, c2, a1, a2, a3;
10 for (int j = 0; j < x.size(); ) {
11     // Stores the current interval index and some temporary variable
12     // Find the interval for x; while assuming that x values are sorted
13     // and within [t_min, t_max]
14     if (t_(i_star) <= x(j) && x(j) <= t_(i_star + 1)) {
15         t1 = t_(i_star);
16         t2 = t_(i_star + 1);
17         h_tmp = t2 - t1; // h_tmp = h_ for equidistant grid
18         y1 = y_(i_star);
19         y2 = y_(i_star + 1);
20         c1 = c_(i_star);
21         c2 = c_(i_star + 1);
22         a1 = y2 - y1;
23         a2 = a1 - h_tmp * c1;
24         a3 = h_tmp * c2 - a1 - a2;
25
26         // Compute s(x(j))
27         tmp = (x(j) - t1) / h_tmp;
28         ret(j) = y1 + (a1 + (a2 + a3 * tmp) * (tmp - 1.)) * tmp;
29
30         ++j;
31     } else {
32         // Otherwise, go to next interval
33         ++i_star;
34         // Terminate if outside any interval
35         if (i_star >= t_.size() - 1) break;
36         continue;
37     }
38 }
39
40 // END
41
42 return ret;
43 }

```

Get it on  GitLab (pchi.hpp).



(6-7.f)  (10 min.) [depends on Sub-problem (6-7.e)]

Based on the class **CubicHermiteInterpolant** realize a C++ function

```
std::vector<double> rspchipConvergence (void);
```

that answers the questions of Sub-problem (6-7.c) for the interpolant according to Def. 6.7.6.

SOLUTION of (6-7.f):

C++ code 6.7.10: Sub-problem (6-7.f): Convergence of Hermite interpolation with reconstructed slopes

```

2  std::vector<double> rspchipConvergence() {
3      // Interpoland

```

```

4  auto f = [](double x) { return 1. / (1. + x * x); };
5
6  double a = 5; // Interval bounds will be (-a,a)
7  int M = 1000; // Number of points in which to evaluate the interpoland
8
9  // Precompute values at which evaluate f
10 Eigen::VectorXd x = Eigen::VectorXd::LinSpaced(M, -a, a);
11 Eigen::VectorXd fx = x.unaryExpr(f);
12
13 // Store error and number of nodes
14 std::vector<double> N_nodes, err_reconstr;
15
16 // TO DO: compute err_reconstr.
17 // START
18
19 for (int i = 4; i <= 512; i = i << 1) {
20     // Define subintervals and evaluate f there (find pairs (t,y))
21     Eigen::VectorXd t = Eigen::VectorXd::LinSpaced(i, -a, a);
22
23     // Construct PCHI with zero and reconstructed slopes
24     CubicHermiteInterpolant s_reconstr(f, t);
25
26     // Evaluate interpolant
27     Eigen::VectorXd s_reconstr_x = s_reconstr.eval(x);
28
29     // Compute infinity norm of error
30     err_reconstr.push_back((s_reconstr_x - fx).lpNorm<Eigen::Infinity>());
31     N_nodes.push_back(i);
32
33     if (i == 16) {
34         plt::figure();
35         plt::title("Interpolant with reconstructed slope");
36         plt::xlabel("t");
37         plt::ylabel("y");
38         plt::plot(x, s_reconstr_x, "r", {{ "label", "s_reconstr" }});
39         plt::plot(x, fx, "b--", {{ "label", "f" }});
40         plt::legend();
41         plt::savefig("./cx_out/p_reconstr.png");
42     }
43 }
44 // END
45
46 return err_reconstr;
47 }

```

Get it on  GitLab (pchi.hpp).

No. of nodes	error norm
4	0.706989
8	0.277399
16	0.0499862
32	0.00480491
64	0.000355225
128	3.72701×10^{-5}
256	4.53776×10^{-6}
512	5.46835×10^{-7}

The table shows the corresponding error for each node of the Hermite interpolation with reconstructed slopes.

Halving the meshwidth engenders a reduction of the maximum norm of the interpolation error by roughly a factor of 8. Thus, in this case we observe algebraic h -convergence of $\|t - s\|_{L^\infty([a,b])}$ with rate 3 as $h_{\mathcal{M}} \rightarrow 0$.



(6-7.g) ☒ (30 min.) Write $I_H : C^0([a, b]) \rightarrow C^1([a, b])$ for the piecewise cubic Hermite interpolation operator with reconstructed slopes introduced in Def. 6.7.6.

Based on the result of Sub-problem (6-7.a) prove that

$$f \in C^3([a, b]) \Rightarrow \exists C > 0: \|f - I_H f\|_{L^\infty([a, b])} \leq C h_{\mathcal{M}}^3,$$

with $C > 0$ independent of the mesh and $h_{\mathcal{M}}$ standing for the mesh width of $\mathcal{M}$.

HINT 1 for (6-7.g): Use Taylor expansion [Lecture $\rightarrow$ Eq. (1.5.4.28)] of f in x_j to express $f(x_{j\pm 1})$ and $f(x_{j\pm 2})$ as a “perturbation” of $f(x_j)$. ┘

SOLUTION of (6-7.g):

For the sake of simplicity this solution assumes an equidistant mesh with meshwidth $h = x_j - x_{j-1}$, $j = 1, \dots, m$. The same approach works for more general meshes, but incurs distracting notational complexity. Throughout we write $s := I_H f$.

First, we show that the approximation $s'(x_j) = f'(x_j) + O(h^2)$. This follows from **Taylor expansion**:

$$f(x) = f(x_j) + f'(x_j)(x - x_j) + f''(x_j)(x - x_j)^2/2 + O(|x - x_j|^3) \quad \text{as } x \rightarrow x_j. \quad (6.7.11)$$

Using $x = x_{j-1}$ and $x = x_{j+1}$ (and $h = x_{j+1} - x_j$):

$$\begin{aligned} \frac{f(x_{j+1}) - f(x_j)}{h} &= f'(x_j) + f''(x_j)h/2 + O(h^2), \\ \frac{f(x_{j-1}) - f(x_j)}{h} &= -f'(x_j) + f''(x_j)h/2 + O(h^2). \end{aligned}$$

Subtracting the second equation from the first equation we get

$$\frac{f(x_{j+1}) - f(x_{j-1})}{h} = 2f'(x_j) + O(h^2).$$

For the one-sided difference we expand at $x = x_{j+2}$ and $x = x_{j+1}$ ($h = x_{j+1} - x_j = x_{j+2} - x_{j+1}$):

$$\begin{aligned} \frac{f(x_{j+1}) - f(x_j)}{h} &= f'(x_j) + f''(x_j)h/2 + O(h^2), \\ \frac{f(x_{j+2}) - f(x_j)}{2h} &= f'(x_j) + f''(x_j)h + O(h^2). \end{aligned} \quad (6.7.12)$$

In light of the definition of the slope in Def. 6.7.6 this yields for the slopes at interior mesh nodes

$$|s'(x_j) - f'(x_j)| = O(h^2) \quad \text{as } h \rightarrow 0 \quad \text{for } j = 1, \dots, m-1. \quad (6.7.13)$$

Subtracting the first equation of (6.7.12) from half of the second equation:

$$\begin{aligned} \frac{f(x_{j+2}) - f(x_j)}{4h} - \frac{f(x_{j+1}) - f(x_j)}{h} &= f'(x_j)(1/2 - 1) + O(h^2), \\ \frac{f(x_{j+1}) - f(x_j) - 4f(x_{j+1}) + 4f(x_j)}{4h} &= f'(x_j)(1/2 - 1) + O(h^2), \\ \frac{-f(x_{j+2}) + 4f(x_{j+1}) - 3f(x_j)}{2h} &= f'(x_j) + O(h^2). \end{aligned}$$

We apply this to x_0 and x_m , which gives (6.7.13) also for $j = 1, m$.

Finally we combine (6.7.13) with the result of Sub-problem (6-7.a) and find $\|f - s\|_{L^\infty([a,b])} = O(h^3)$ for $h \rightarrow 0$. The rate of convergence is limited to 3.



End Problem 6-7 , 125 min.