

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 5-8: Approximating π by extrapolation

In [Lecture → Section 5.2.3.3] we learned about the approximation of a limit $\lim_{h \rightarrow 0} \psi(h)$ based on the evaluation of the function ψ “far away” from 0. In this exercise we will practice the underlying technique of “Lagrangian polynomial extrapolation” for a geometric approximation problem.

Study [Lecture → Section 5.2.3.3] and, in particular [Lecture → Code 5.2.3.19], before you tackle this problem.

In this problem we encounter the situation that a quantity of interest is defined as a limit of a sequence

$$x^* = \lim_{n \rightarrow \infty} T(n),$$

where the function $T : \{n_{\min}, n_{\min} + 1, \dots\} \mapsto \mathbb{R}$ may be given in procedural form as `double T(int n)` only. However, invoking `T(Inf)` will usually result in an error and for very large arguments n the implementation of `T()` may not yield reliable results, cf. [Lecture → Ex. 1.5.4.7] and [Lecture → § 5.2.3.16].

The idea of **extrapolation** is, firstly, to compute a few values $T(n_0), T(n_1), \dots, T(n_k)$, $k \in \mathbb{N}$, and to consider them as the values $g(1/n_0), g(1/n_1), \dots, g(1/n_k)$ of a continuous function

$$g : (0, 1/n_{\min}] \mapsto \mathbb{R}, \quad \text{for which, obviously } x^* = \lim_{h \rightarrow 0} g(h).$$

Secondly, the function g is approximated by an interpolating polynomial $p_k \in \mathcal{P}_k$ with $p_k(n_j^{-1}) = T(n_j)$, $j = 0, \dots, k$. In many cases we can expect that $p_k(0)$ will provide a good approximation for x^* .

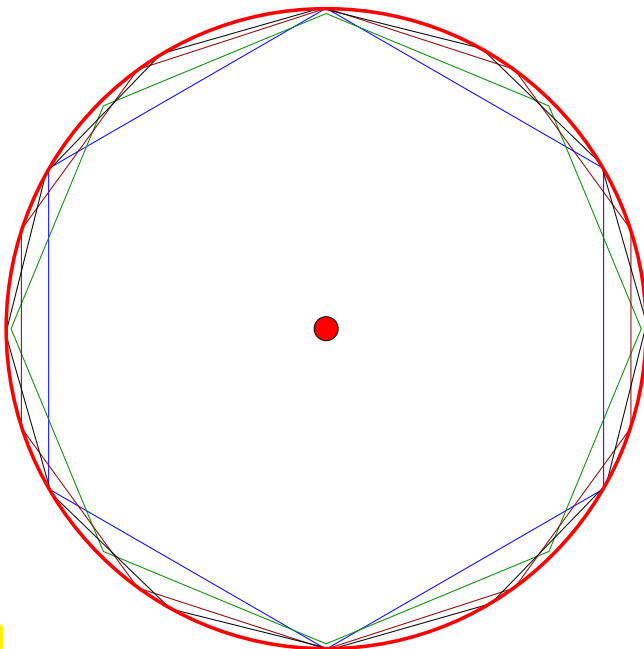


Fig. 36

The unit circle can be approximated by inscribed regular polygons with k edges. Thus, the length of half circumference (i.e., π) can be approximated by the half the length of the perimeters s_k of such polygons. These values $s_k/2$ can be calculated by elementary trigonometry and a closed form formula is

$$\frac{1}{2}s_k = k \sin(\pi/k). \quad (5.8.1)$$

(5-8.a) (10 min.) Show that for any $L \in \mathbb{N}$ and for $k \rightarrow \infty$

$$\frac{1}{2}s_k = \pi + \sum_{\ell=1}^L c_\ell k^{-2\ell} + O(k^{-2(L+1)}) \quad \text{for some } c_\ell \in \mathbb{R} \text{ independent of } L. \quad (5.8.2)$$

We say that s_k has an **asymptotic expansion** in k^{-2} .

Remark. The heuristics behind polynomial extrapolation in this example is that the existence of an asymptotic expansion (5.8.2) suggests that $\frac{1}{2}s_\infty$ can be well approximated by $p(0)$, where p is an interpolating polynomial in k^{-1} or k^{-2} .

HINT 1 for (5-8.a): Remember the power series for `sin`:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots = \sum_{\ell=0}^{\infty} \frac{(-1)^\ell x^{2\ell+1}}{(2\ell+1)!}, \quad x \in \mathbb{R}. \quad (5.8.3)$$

SOLUTION of (5-8.a):

Using the power series for `sin` (5.8.3) we obtain

$$k \sin(\pi/k) = \pi + \sum_{\ell=1}^{\infty} \frac{(-1)^\ell \pi^{2\ell+1} k^{-2\ell}}{(2\ell+1)!}.$$

Since this series converges absolutely for all k , we have

$$\sum_{\ell=L+1}^{\infty} \frac{(-1)^\ell \pi^{2\ell+1} k^{-2\ell}}{(2\ell+1)!} \leq C k^{-2(L+1)} \quad \text{for } C > 0 \text{ independent of } k.$$

(5-8.b) (45 min.) Implement a C++ function

double extrapolate_to_pi(const unsigned int k);

that uses the *Aitken-Neville scheme*, see [Lecture → Code 5.2.3.10], to approximate π by extrapolation from the data points $(j^{-1}, \frac{1}{2}s_j)$, for $j = 2, \dots, k$. Use the values $\frac{1}{2}s_j$ as given in (5.8.1).

SOLUTION of (5-8.b):

C++ code 5.8.4: Implementation of `extrapolate_to_pi()`

```

2  double extrapolate_to_pi(const unsigned int k) {
3      double pi = 0.0;
4
5      // TODO (5-8.b): Use the Aitken-Neville scheme to approximate
6      // pi by extrapolation.
7      // Hint: You can use the constant M_PI and Eigen's sin()
8      // when calculating s_j for j = 2, ..., k.
9
10     // START
11
12     // Data for interpolation.
13     // Note: We are using eigen arrays because they allow for coefficient-wise
14     // operations while eigen matrices/vectors are used for linear algebra
15     // operations.
16     Eigen::ArrayXd J = Eigen::ArrayXd::LinSpaced(k - 1, 2, k);
17     Eigen::ArrayXd T = 1.0 / J; // t_j = 1/j
18     Eigen::ArrayXd S = J * sin(M_PI * T);
19
20     // Aitken-Neville scheme for evaluating the interpolating
21     // polynomial of (t_j, s_j) at x = 0.
22     for (int l = 0; l < S.size(); l++) {
23         for (int m = l - 1; m >= 0; m--) {
24             S(m) = S(m + 1) + (S(m + 1) - S(m)) * (0.0 - T(l)) / (T(l) - T(m));

```

```

25     }
26 }
27 // Now S(0) has contributions from all data points.
28 pi = S(0);
29 // END
30
31 return pi;
32 }

```

Get it on  GitLab (extrapolpi.hpp).



(5-8.c)  (30 min.) Write a C++ function

void plotExtrapolationError(const unsigned int kmax);

that generates a *linear-logarithmic* plot of the extrapolation errors

$$\text{err}(k) := |\pi - p_k(0)|, \quad k = 2, \dots, k_{\max},$$

versus k and also tabulates the errors. To create the plot use the functions of **MATPLOTLIBCPP**. Do not forget axis annotations and a meaningful title for the plot.

In `main.cpp`, `plotExtrapolationError(10)` is called for $k_{\max} = 10$.

SOLUTION of (5-8.c):

C++ code 5.8.5: Implementation of `plotExtrapolationError()`

```

2  void plotExtrapolationError(const unsigned int kmax) {
3      plt::figure();
4      // TODO (5-8.c): Plot the error made by extrapolate_to_pi(k) for
5      // k = 2, 3, ..., 10. Use a log-scale for the y-axis, and a linear x-axis.
6      // Also, tabulate the results and errors of extrapolate_to_pi().
7      // Hint: Use the constant M_PI to compute the error made by
8      // extrapolate_to_pi().
9      // Hint: matplotlibcpp (here = plt) implements Python's
10     // matplotlib functions such as figure(), plot(), xlabel(), title(), ...
11     // You can use main.cpp of the LinearDataFit problem as a reference.
12
13     // START
14     Eigen::VectorXd K = Eigen::VectorXd::LinSpaced(kmax - 1, 2, kmax);
15     Eigen::VectorXd Err(K.size());
16
17     std::cout << std::setw(5) << " k" << std::setw(15) << "approx. pi"
18               << std::setw(15) << "error" << std::endl;
19
20     for (int l = 0; l < K.size(); l++) {
21         double pi = extrapolate_to_pi(K(l));
22         Err(l) = std::abs(pi - M_PI);
23         std::cout << std::setw(5) << K(l) << std::setw(15) << pi << std::setw(15)
24               << Err(l) << std::endl;
25     }
26
27     // Number of data points is k-1, so polynomial degree is k-2.
28     Eigen::VectorXd Deg = K - 2 * Eigen::VectorXd::Ones(kmax - 1);
29     plt::semilogy(Deg, Err);

```

```

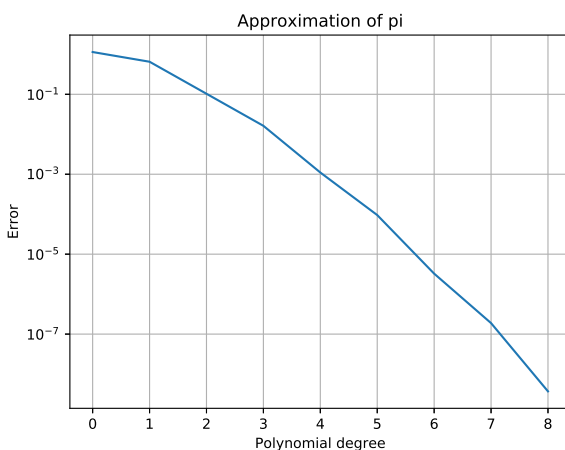
30 plt::title("Approximation of pi");
31 plt::xlabel("Polynomial degree");
32 plt::ylabel("Error");
33 plt::grid();
34 // END
35
36 std::string path = "./cx_out/pi_error_" + std::to_string(kmax) + ".png";
37 plt::savefig(path);
38 }

```

Get it on  GitLab (extrapolpi.hpp).

The tabulated errors should be:

num. data points,	approximation of pi,	error:
1	2.00000000	1.1415926
2	3.79422863	0.6526359
3	3.24473110	0.1031384
4	3.12532464	0.0162680
5	3.14048823	0.0011044
6	3.14167629	0.0000836
7	3.14159488	0.0000022



Linear-logarithmic plot of error in the approximation of π by extrapolation.

Fig. 37

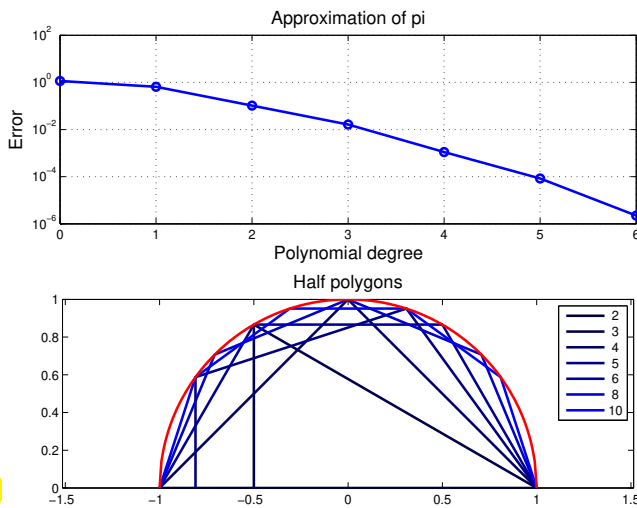


(5-8.d)  (15 min.) [depends on Sub-problem (5-8.c)]

Which kind of convergence of $\text{err}(k) \rightarrow 0$ for $k \rightarrow \infty$ can you conclude from the data generated in Sub-problem (5-8.c)? The main types of asymptotic convergence to 0 are introduced in [Lecture $\rightarrow$ Def. 6.2.2.7].

HINT 1 for (5-8.d): Study [Lecture $\rightarrow$ § 6.2.2.9] to learn how to determine the type of convergence from error data.

SOLUTION of (5-8.d):



The convergence is **exponential** with respect to the degree k of the polynomial used for extrapolation. We deduce this from the observation that the points in the error plot are nearly located on a line.

At second glance, the error points are located on a slightly downward-bending curve, which even hints at **superexponential convergence**, that is, faster than exponential.

Fig. 38



(5-8.e) (15 min.) In `main.cpp`, `plotExtrapolationError(30)` is called which plots and tabulates err_k for k up to 30. Describe and explain your observations of the plot found in `cx_out/pi_error_30.png`.

SOLUTION of (5-8.e):

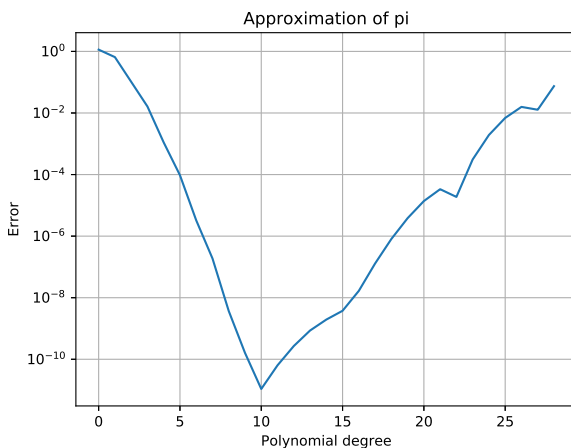


Fig. 39

For large k the error starts increasing again!

Two possible explanations:

1. High-degree polynomial interpolants display strong oscillations between the data points, which affect $p(0)$.
2. The Aitken-Neville algorithm is based on the computation of differences of similar quantities, therefore, ultimately, it will be vulnerable to cancellation errors.

Both effects may play a role here.



End Problem 5-8 , 115 min.