

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

Chapter 7

Numerical Quadrature

Problem 7-1: Smooth integrand by transformation

In [Lecture → Rem. 7.4.3.10] we saw how knowledge about the structure of a non-smooth integrand can be used to restore its smoothness by transformation. The very same idea can be put to use in this problem.

This problem practises the use of Gauss-Legendre quadrature and of the techniques introduced in [Lecture → Rem. 7.4.3.10].

Given a smooth, odd function $f : [-1, 1] \rightarrow \mathbb{R}$, consider the integral

$$I := \int_{-1}^1 \arcsin(t) f(t) dt. \quad (7.1.1)$$

We want to approximate this integral using global Gauss-Legendre quadrature as introduced in [Lecture → Section 7.4.2].

The nodes and the weights of the n -point Gauss-Legendre quadrature rule on $[-1, 1]$ can be computed using the provided C++ function

```
QuadRule gaussquad(const unsigned n);
```

which initializes a data structure describing a quadrature rule

```
struct QuadRule {
    Eigen::VectorXd nodes_;
    Eigen::VectorXd weights_;
};
```

The names of the fields tell their function. The function `gaussquad()` can be found in `gaussquad.hpp`.

(7-1.a) 🕒 (30 min.) Based on `MATPLOTLIBCPP` write a C++ function

```
template <class Function>
double gaussConv(const Function& fh, const double I_ex, const
    unsigned N);
```

that produces an *appropriate* plot of the quadrature error of the Gauss-Legendre quadrature rules versus the number $n = 1, \dots, 50$ of quadrature points. The function should output the best approximation of

the integral, i.e. the approximated value at the maximum number of quadrature points. Here f is a functor object [Lecture → Section 0.3.3] providing the function f in (7.1.1). I_{ex} is the exact solution of the integral and N is the maximum number of quadrature points.

Save your convergence plot for $f(t) = \sinh(t)$ as `GaussConv.png`.

HINT 1 for (7-1.a): Use as exact value of the integral $I \approx 0.870267525725852642$. ┘

SOLUTION of (7-1.a):

In our implementation we defined an auxiliary function that approximates the integral for a given function and quadrature rule according to [Lecture → Def. 7.2.0.1].

C++-code 7.1.2: Implementation of the auxiliary function `integrate()`

```

2  template <class Function>
3  double integrate(const QuadRule& qr, const Function& f) {
4      double I = 0;
5
6      // TODO: (7-1.a) optional: compute the integral with this helper
7      // function
8      // START
9
10     // Compute weighted sum of function values according to
11     // the concept of a generic quadrature formula
12     for (unsigned i = 0; i < qr.weights_.size(); ++i) {
13         I += qr.weights_(i) * f(qr.nodes_(i));
14     }
15
16     // END
17     return I;
18 }
```

C++-code 7.1.3: Implementation of the function `gaussConv()`

```

2  template <class Function>
3  double gaussConv(const Function& fh, const double I_ex, const unsigned N) {
4      std::vector<double> evals, // used to save no. of quad nodes
5      error, // used to save the error
6      evals_ref; // used to plot reference line of order of convergence
7
8      double I_N = 0.; // best approx. of integral
9      plt::figure();
10
11     // TODO: (7-1.a) write code that plots the no. of nodes versus the
12     // error
13     // using matplotlibcpp and returns the best approximation
14     // START
15
16     // Build integrand
17     auto f = [fh](double x) { return std::asin(x) * fh(x); };
18
19     double I;
20     for (unsigned n = 1; n <= N; ++n) {
21         QuadRule qr = gaussquad(n); // Create quadrature rule
22
23         I = integrate(qr, f); // Compute integral
24
25         evals.push_back(n); // Save no. of quadrature nodes
26     }
```

```

25     evals_ref.push_back(
26         std::pow(n, -3)); // used for plotting to show convergence rate
27     error.push_back(std::abs(I - I_ex)); // Save error
28 }
29 // save best approx. of integral
30 I_N = I;
31
32 // Create convergence plots
33 plt::title("Gauss quadrature convergence");
34 plt::loglog(evals, error, "+r", {"label", "Error"}); // plot error
35 plt::loglog(evals, evals_ref, "k--",
36             {"label", "O(n-3)"}); // reference line
37 plt::xlabel("No. of quadrature nodes");
38 plt::ylabel("|Error|");
39 plt::legend("best");
40 // END
41
42 plt::savefig("./cx_out/GaussConv.png");
43 // return the best approximation
44 return I_N;
45 }

```

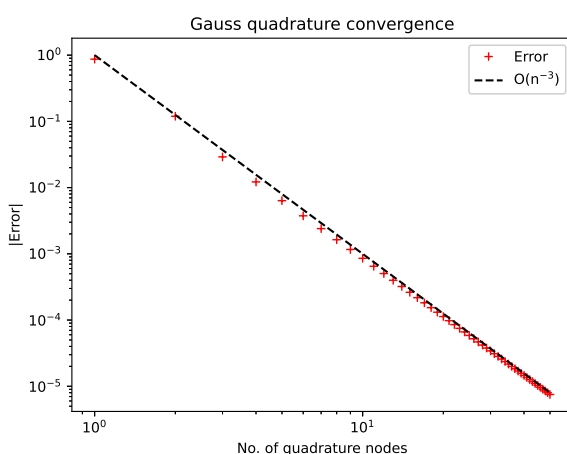
Get it on  GitLab (gaussConvergence.hpp).

The convergence plot is included in the solution of the next sub-problem.

(7-1.b)  (15 min.) Describe qualitatively (type of convergence) and quantitatively (speed of convergence) the asymptotic (for $n \rightarrow \infty$) convergence of the quadrature error you observe in (7-1.a).

HINT 1 for (7-1.b): Types of convergence have been introduced in [Lecture $\rightarrow$ Def. 6.2.2.7] for approximation errors. They apply to quadrature errors as well.

SOLUTION of (7-1.b):



◁ *Doubly logarithmic* plot of the quadrature error.

Using other axes scales yields plots from which the type of convergence is not immediately obvious.

Fig. 59

The plot indicates **algebraic convergence** (being a straight line in log-log mode), with a rate of about 3: **error = $O(n^{-3})$** for $n \rightarrow \infty$.

(7-1.c) 🕒 (20 min.) With a suitable change of variable transform the integral of (7.1.1) into an equivalent one, so that Gauss-Legendre quadrature applied to the transformed integral can be expected to *converge exponentially* in the number of quadrature nodes, if f possesses an analytic extension beyond $[-1, 1]$.

HINT 1 for (7-1.c): Combine the results of [Lecture → § 7.4.3.2], [Lecture → Lemma 7.4.3.1], and [Lecture → Rem. 6.2.3.26].

HINT 2 for (7-1.c): Use the transformation $t = \sin(x)$.

SOLUTION of (7-1.c):

With the change of variable $t = \sin(x)$, $dt = \cos(x) dx$, we obtain

$$I = \int_{-1}^1 \arcsin(t) f(t) dt = \int_{-\pi/2}^{\pi/2} x f(\sin(x)) \cos(x) dx. \quad (7.1.4)$$

The change of variable has to provide an analytic integrand on the integration interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. The integrand is analytic, provided f has this property on $[-1, 1]$, because the trigonometric functions $\sin$ and $\cos$ are *entire functions*, that is, analytic on $\mathbb{C}$. Then invoke [Lecture → Thm. 6.2.2.68].

If the integrand possesses an analytic extension beyond $[-\frac{\pi}{2}, \frac{\pi}{2}]$ we can appeal to the results of [Lecture → Rem. 6.2.3.26] and [Lecture → Thm. 7.4.3.3] to conclude exponential convergence of the quadrature errors.



(7-1.d) 🕒 (25 min.) [depends on Sub-problem (7-1.c)]

Now, write a C++ function

```
template <class Function>
double gaussConvCV(const Function& f, const double I_ex, const
    unsigned N);
```

which plots (employing `MATPLOTLIBCPP`) the quadrature error versus the number $n = 1, \dots, 50$ of quadrature points for the integral obtained in the previous subtask and returns the best approximation of the integral, i.e. the approximated value at the maximum number of quadrature points.

Again, as in Sub-problem (7-1.a), choose $f(t) = \sinh(t)$ and save your convergence plot as `GaussConvCV.png`.

SOLUTION of (7-1.d):

The Gauss quadrature nodes and weights are defined on the interval $[-1, 1]$. We need to transform them to the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Consider the following general affine transformation:

$$\begin{cases} \Phi : [-1, 1] & \rightarrow [a, b] \\ \hat{x} & \rightarrow a + \frac{b-a}{2}(\hat{x} + 1) = \frac{b-a}{2}\hat{x} + \frac{a+b}{2} \end{cases} \quad (7.1.5)$$

Then we can transform $\int_a^b g(x) dx$, the integral which needs to be computed over the domain $[a, b]$, as the following:

$$\int_a^b g(x) dx = \frac{b-a}{2} \int_{-1}^1 g\left(\frac{b-a}{2}\hat{x} + \frac{a+b}{2}\right) d\hat{x} \approx \frac{b-a}{2} \sum_{i=1}^n \hat{w}_i f\left(\frac{b-a}{2}\hat{x} + \frac{a+b}{2}\right) \quad (7.1.6)$$

The nodes on $[a, b]$, $\{x_i\}_{i=1}^n$, can be directly obtained from the affine transformation above.

The weights on $[a, b]$, $\{w_i\}_{i=1}^n$, can be obtained by:

$$w_i = \frac{b-a}{2} \hat{w}_i \quad (7.1.7)$$

In our example, this means that $w_i = \frac{\pi}{2} \hat{w}_i$.

C++-code 7.1.8: Implementation of `gaussConvCV()`

```

2  template <class Function>
3  double gaussConvCV(const Function& f, const double l_ex, const unsigned N) {
4      std::vector<double> evals, // Used to save no. of quad nodes
5                          error; // Used to save the error
6
7      double l_N = 0.; // best approx. of integral
8      plt::figure();
9
10     // TODO: (7-1.d) complete to perform Gauss-Legendre quadrature with
11     // the
12     // variable transform of the integral derived in 8-1.c)
13     // write code that plots the no. of nodes versus the error
14     // using matplotlibcpp and returns the best approximation
15     // START
16
17     // Lambda function providing transformed integrand
18     auto g = [f](double x) { return x * f(std::sin(x)) * std::cos(x); };
19
20     double l;
21     for (unsigned n = 1; n <= N; ++n) {
22         QuadRule qr = gaussquad(n); // Obtain quadrature rule
23
24         // Transform nodes and weights to new interval
25         Eigen::VectorXd w = qr.weights_ * M_PI / 2;
26         Eigen::VectorXd c =
27             (-M_PI / 2 + M_PI / 2 * (qr.nodes_.array() + 1)).matrix();
28
29         // Evaluate g at quadrature nodes c
30         Eigen::VectorXd gc = c.unaryExpr(g);
31
32         // Same as  $I = \sum_{i=1}^n w_i g(c_i)$ 
33         l = w.dot(gc);
34
35         evals.push_back(n); // Save no. of quadrature nodes
36         error.push_back(std::abs(l - l_ex)); // Save error
37     }
38
39     // save approximation of integral computed with highest-order rule
40     l_N = l;
41
42     // Create convergence plots
43     plt::title("Gauss quadrature convergence");
44     plt::semilogy(

```

```

44     evals, error, "+r",
45     [{"label", "Error"}]); // plot error as lin-log plot to see
                             convergence
46 plt::xlabel("No. of quadrature nodes");
47 plt::ylabel("|Error|");
48 plt::legend("best");
49 // END
50
51 plt::savefig("./cx_out/GaussConvCV.png");
52 // return the best approximation
53 return I_N;
54 }

```

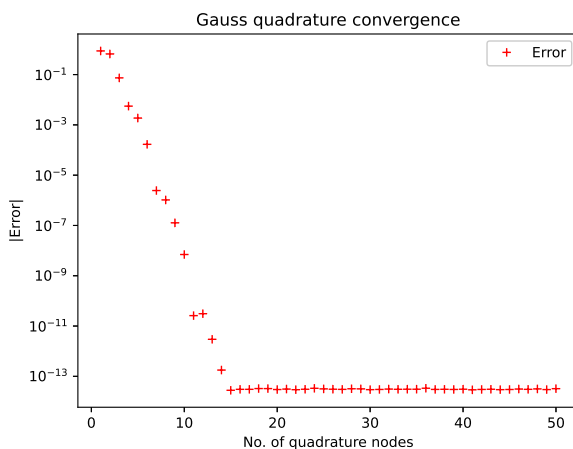
Get it on 🐙 [GitLab](#) ([gaussConvergence.hpp](#)).



(7-1.e) 🕒 (10 min.) [depends on Sub-problem (7-1.d)]

Analogous question as in Sub-problem (7-1.b): describe qualitatively the asymptotic (for $n \rightarrow \infty$) convergence of the quadrature error you observe in Sub-problem (7-1.d).

SOLUTION of (7-1.e):



◁ *Linear-logarithmic plot* of the quadrature error of Gauss-Legendre quadrature rules applied to the transformed integral as a function of the number of quadrature nodes.

Fig. 60

The plot indicates **exponential convergence** (being a straight line in **lin-log** scale).



(7-1.f) 🕒 [depends on Sub-problem (7-1.b) and Sub-problem (7-1.d)]

Explain the difference between observations made in Sub-problem (7-1.b) and Sub-problem (7-1.d).

SOLUTION of (7-1.f):

In Sub-problem (7-1.b) we found merely asymptotic **algebraic convergence** of the quadrature error of Gauss-Legendre quadrature in terms of the number of quadrature nodes, whereas in Sub-problem (7-1.d) we observe **exponential convergence**.

The reason is that the function $t \mapsto \arcsin t$ merely belongs to $C^0([-1, 1])$, but its derivative $t \mapsto \frac{1}{\sqrt{1-t^2}}$

has singularities in the points $t = \pm 1$. Hence, it and the entire integrand in (7.1.1) has finite smoothness only and *cannot be extended analytically* beyond the interval $[-1, 1]$, which rules out exponential convergence.

Conversely, in Sub-problem (7-1.d) the change of variable provides a perfectly smooth integrand: $x \mapsto x \cos(x) \sinh(\sin x)$ that possesses an analytic extension to some complex neighborhood of $[-\frac{\pi}{2}, \frac{\pi}{2}]$. This makes it possible to apply the estimated of [Lecture $\rightarrow$ Thm. 7.4.3.3] and [Lecture $\rightarrow$ Eq. (6.2.3.28)] and conclude exponential convergence in the number of quadrature nodes.



End Problem 7-1 , 100 min.