

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 7-3: Numerical Quadrature of Improper Integrals

In analysis you have seen so-called improper integrals over unbounded intervals, which, due to a fast decay of the integrand have a finite value. Occasionally such integrals have to be evaluated numerically and this problem will be concerned with corresponding quadrature rules.

This problem makes extensive use of substitution rules for integrals, implementation in C++ is requested.

We want to devise a numerical method for the computation of improper integrals of the form $\int_{-\infty}^{\infty} f(t) dt$ for continuous functions $f: \mathbb{R} \rightarrow \mathbb{R}$ that decay sufficiently fast for $|t| \rightarrow \infty$ (such that they are integrable on $\mathbb{R}$).

A first option is the truncation of the domain to a bounded interval $[-b, b]$, $b \leq \infty$, that is, we approximate:

$$\int_{-\infty}^{\infty} f(t) dt \approx \int_{-b}^b f(t) dt$$

and then use a standard quadrature rule like Gauss-Legendre quadrature [Lecture $\rightarrow$ Def. 7.4.2.18] on $[-b, b]$.

(7-3.a)  (10 min.) For the integrand $g(t) := 1/(1+t^2)$ determine b such that the truncation error E_T satisfies:

$$E_T := \left| \int_{-\infty}^{\infty} g(t) dt - \int_{-b}^b g(t) dt \right| \leq 10^{-6}. \quad (7.3.1)$$

HINT 1 for (7-3.a): You will probably remember that $\arctan' = g$, where the prime tags a derivative.

┘

SOLUTION of (7-3.a):

An antiderivative of g is $\arctan$. Since the function g is even, we have

$$E_T = 2 \int_b^{\infty} g(t) dt = \lim_{x \rightarrow \infty} 2\arctan(x) - 2\arctan(b) = \pi - 2\arctan(b) \stackrel{!}{<} 10^{-6}, \quad (7.3.2)$$

i.e. $b > \tan((\pi - 10^{-6})/2) = \cot(10^{-6}/2) \approx 2 \cdot 10^6$.

▲

(7-3.b)  (5 min.) What is the algorithmic difficulty faced in the implementation of the truncation approach for a generic integrand?

SOLUTION of (7-3.b):

A good choice of b requires a detailed knowledge about the decay of f , which may not be available for f defined implicitly.

▲

A second option is the transformation of the improper integral to a bounded domain by substitution. For instance, we may use the map $t = \cot(s) := \frac{\cos s}{\sin s}$, $\cot :]0, \pi[\rightarrow]-\infty, \infty[$.

(7-3.c) (15 min.) Into which integral does the substitution $t = \cot(s)$ convert $\int_{-\infty}^{\infty} f(t) dt$?

SOLUTION of (7-3.c):

We have:

$$\frac{dt}{ds} = -\left(1 + \cot^2(s)\right) = -(1 + t^2)$$

$$\int_{-\infty}^{\infty} f(t) dt = - \int_{\pi}^0 f(\cot(s)) \left(1 + \cot^2(s)\right) ds = \int_0^{\pi} \frac{f(\cot(s))}{\sin^2(s)} ds, \quad (7.3.3)$$

because $\sin^2(\theta) = \frac{1}{1 + \cot^2(\theta)}$.

▲

(7-3.d) (10 min.) Write down the transformed integral explicitly for the integrand $g(t) := \frac{1}{1+t^2}$. Simplify the integrand.

HINT 1 for (7-3.d): Use identities involving squares of trigonometric functions, see, for instance the [Pythagorean trigonometric identities](#) online. ┘

SOLUTION of (7-3.d):

Replacing the function $g(t)$, we get:

$$\int_0^{\pi} \frac{1}{1 + \cot^2(s)} \frac{1}{\sin^2(s)} ds = \int_0^{\pi} \frac{1}{\sin^2(s) + \cos^2(s)} ds = \int_0^{\pi} ds = \pi$$

▲

(7-3.e) (20 min.) [depends on Sub-problem (7-3.c)]

Write a C++ function that uses the previous transformation together with the n -point Gauss-Legendre quadrature rule to evaluate $\int_{-\infty}^{\infty} f(t) dt$:

```
template <typename Function>
double quadinf(int n, Function && f);
```

f passes an object that provides an evaluation operator of the form:

```
double operator() (double x) const;
```

You can use the function `golubwelsh()` implemented in the file `golubwelsh.hpp` that computes nodes and weights for Gauss quadrature rules using the Golub-Welsch algorithm discussed in [Lecture → Rem. 7.4.2.23], see also [Lecture → Code 7.4.2.24].

HINT 1 for (7-3.e): A lambda function with signature `(double) -> double` automatically satisfies the requirement for f . ┘

SOLUTION of (7-3.e):

We give an implementation based on an auxiliary function `quad()` that applies a quadrature rule on $[-1, 1]$ to approximate an integral over $[a, b]$. Of course you need not employ such auxiliary functions.

C++-code 7.3.4: Implementation of transformed quadrature rule

```

2  template <class Function>
3  double quad(Function &&f, const Eigen::VectorXd &w, const Eigen::VectorXd &x,
4             double a, double b) {
5      double l = 0.;
6      // TODO: (7-3.e) implement transformation of quadrature rule
7      // START
8      for (unsigned int i = 0; i < w.size(); ++i) {
9          l += f((x(i) + 1) * (b - a) / 2 + a) * w(i);
10     }
11     l *= (b - a) / 2.;
12     // END
13     return l;
14 }
```

Get it on  GitLab ([quadinf.hpp](#)).

C++-code 7.3.5: Implementation of `quadinf()`

```

2  template <class Function>
3  double quadinf(const int n, Function &&f) {
4      double l = 0.;
5      // TODO: (7-3.e) define the function quadinf that integrates the
6      // input
7      // lambda f over the real axis. First, transform the integrand with
8      // a change
9      // of variables and then use an n point Gauss quadrature.
10     // Hint 1: lambda functions can take parameters inside the []
11     // brackets
12     // Hint 2: you may write an auxiliary function to compute the
13     // quadrature over
14     // a bounded interval, see function quad above.
15
16     // START
17     Eigen::VectorXd w, x;
18
19     // Compute nodes and weights of Gauss quadrature rule
20     // using Golub-Welsh algorithm
21     golubwelsh(n, w, x);
22
23     /*! NOTE: no function cot available in c++, need to resort to
24     trigonometric
25     identities. Both lines below are valid, the first computes
26     three
27     trigonometric functions, but the second is prone to
28     cancellation
29     */
30     auto ftilde = [&f](double x) {
31         return f(std::cos(x) / std::sin(x)) / pow(std::sin(x), 2);
32     };
33     /* auto ftilde = [&f](double x) { double cot = std::tan(PI_HALF - x); return
34     * f(cot) * (1. + pow(cot, 2)); }; */
35
36     // integrate ftilde from 0 to PI using Gauss quadrature rules
37     l = quad(ftilde, w, x, 0, PI);
38     // END
```

```

31 |     return l;
32 | }

```

Get it on  GitLab (quadinf.hpp).



(7-3.f)  (15 min.) Implement a C++ function

```
void cvgQuadInf(void);
```

in order to study the convergence for $n \rightarrow \infty$ of the quadrature method implemented in the previous subproblem for the integrand $h(t) := \exp(-(t-1)^2)$ (shifted Gaussian). To compute the error you can use that $\int_{-\infty}^{\infty} h(t) dt = \sqrt{\pi}$.

Use **MATPLOTLIBCPP** to create a suitable plot of the quadrature error versus the number of quadrature nodes. What kind of convergence do you observe?

SOLUTION of (7-3.f):

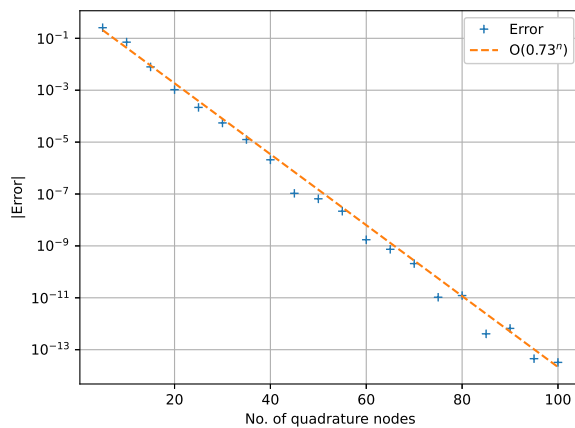
C++-code 7.3.6: Implementation of **cvgQuadInf()**

```

2 | void cvgQuadInf(void) {
3 |     // Number of max Gauss pts.
4 |     constexpr int N = 100;
5 |     plt::figure();
6 |     // TODO: (7-3.f) plot convergence errors against number of
7 |     // integration nodes
8 |     // START
9 |     // variables for the plots
10 |    std::vector<int> num_nodes;
11 |    std::vector<double> err;
12 |    std::vector<double> cvg_rate;
13 |
14 |    // Integrand and exact integral
15 |    auto f = [](double t) { return std::exp(-std::pow((t - 1), 2)); };
16 |    const double l_ex = std::sqrt(PI);
17 |
18 |    for (unsigned int n = 5; n <= N; n += 5) {
19 |        // Approximated value of integral
20 |        const double QS = quadinf(n, f);
21 |        // plot data
22 |        num_nodes.push_back(n);
23 |        err.push_back(std::abs(QS - l_ex));
24 |        // 'straight' line in plot
25 |        cvg_rate.push_back(std::pow(0.73, n));
26 |    }
27 |    plt::semilogy(num_nodes, err, "+", {{ "label", "Error" }});
28 |    plt::semilogy(num_nodes, cvg_rate, "--", {{ "label", "O(0.73^n)" }});
29 |    plt::legend("best");
30 |    plt::xlabel("No. of quadrature nodes");
31 |    plt::ylabel("| Error |");
32 |
33 |    plt::grid("True");
34 |    // END
35 |    plt::savefig("./cx_out/convergence.png");
36 | }

```

Get it on  GitLab (quadinf.hpp).



According to [Lecture → § 6.2.2.9] we observe exponential convergence: The graph of the quadrature error vs. the number of quadrature nodes in **lin-log** scale is approximately a line with negative slope.

Appealing to [Lecture → Def. 6.2.2.7] and recalling [Lecture → § 6.2.2.9], the plot provides empiric evidence of **exponential convergence**.

Fig. 61



End Problem 7-3 , 75 min.