

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 7-9: Efficient quadrature of singular integrands

We know that smoothness of the integrand is essential for fast convergence of high-order numerical quadrature like Gauss quadrature rules. In some cases smart transformation can convert the integral into another with a smooth integrand. This problem uses this trick for the sake of efficient numerical quadrature of non-smooth integrands with a special structure.

Before you tackle this problem, read about regularization of integrands by transformation, cf. [Lecture → Rem. 7.4.3.10]. For other problems on the same topic see Problem 7-7 and Problem 7-1.

Our task is to develop quadrature formulas for integrals of the form:

$$W(f) := \int_{-1}^1 \sqrt{1-t^2} f(t) dt, \quad (7.9.1)$$

where f possesses an analytic extension to a complex neighbourhood of $[-1, 1]$.

(7-9.a)  (10 min.) Understand how to use the C++ function

```
QuadRule gauleg(unsigned int n);
```

(contained in `gauleg.hpp`), which returns a structure `QuadRule` containing nodes (x_j) and weights (w_j) of a Gauss-Legendre quadrature [Lecture → Def. 7.4.2.18] on $[-1, 1]$ with n nodes. ▲

(7-9.b)  (15 min.) Study [Lecture → § 7.4.3.2] in order to learn about the convergence of Gauss-Legendre quadrature. What can you say about the least expected convergence for an integrand $f \in C^r([a, b])$? What about convergence in the case $f \in C^\infty([a, b])$?

SOLUTION of (7-9.b):

If $f \in C^r([a, b])$ then we get algebraic convergence with rate r .

If f has an analytic extension to a complex neighbourhood of $[a, b]$ we get exponential convergence. Almost all integrands $\in C^\infty([a, b])$ will allow such an extension.

▲

(7-9.c)  (40 min.) Based on the function `gauleg()`, implement a C++ function

```
template <class Function>
double quadsingint(const Function& f, const unsigned n);
```

that approximately evaluates $W(f)$ using $2n$ evaluations of f . An object of type `Function` must provide an evaluation operator

```
double operator (double t) const;
```

Ensure that, as $n \rightarrow \infty$, the error of your implementation has asymptotic exponential convergence to zero.

HINT 1 for (7-9.c): A C++ lambda function provides such operator, e.g. `auto f = [](double t) { return t*t; };` for $f(t) = t^2$. ┘

HINT 2 for (7-9.c): You may use $\sqrt{1-t^2} = \sqrt{1-t}\sqrt{1+t}$. ┘

SOLUTION of (7-9.c):

Exploiting the hint, we see that the integrand is non-smooth in ± 1 .

There are two possible solutions to the problem.

Method 1: we partition the integration domain $[-1, 1] = [0, 1] \cup [-1, 0]$. Applying the substitution $s = \sqrt{1 \pm t}$ (sign depending on which part of the integrals considered), $t = \pm(s^2 - 1)$:

$$\begin{aligned}\frac{dt}{ds} &= \pm 2s \\ W(f) &:= \int_{-1}^1 \sqrt{1-t^2} f(t) dt = \int_{-1}^0 \sqrt{1-t^2} f(t) dt + \int_0^1 \sqrt{1-t^2} f(t) dt \\ &= \int_0^1 2 \cdot s^2 \sqrt{2-s^2} f(-s^2+1) ds + \int_0^1 2 \cdot s^2 \sqrt{2-s^2} f(s^2-1) ds.\end{aligned}$$

Notice how the resulting integrand is analytic in a neighbourhood of the domain of integration because, for instance, $t \mapsto \sqrt{1+t}$ is analytic in a neighborhood of $[0, 1]$.

Method 2: one may use the trigonometric substitution $t = \sin s$, obtaining

$$\begin{aligned}\frac{dt}{ds} &= \cos s \\ W(f) &:= \int_{-1}^1 \sqrt{1-t^2} f(t) dt \\ &= \int_{-\pi/2}^{\pi/2} \sqrt{1-\sin^2 s} f(\sin s) \cos s ds = \int_{-\pi/2}^{\pi/2} \cos^2 s f(\sin s) ds.\end{aligned}$$

This integrand is also analytic.

C++-code 7.9.2: Implementation of quadsingint.

```

2  template <class Function>
3  double quadsingint(const Function& f, const unsigned n) {
4      double l = 0.;
5
6      // TODO: (7-9.c) Compute  $\int_{-1}^1 \sqrt{1-t^2} f(t) dt$  using
7      // Gauss-Legendre quadrature. Ensure asymptotic exponential convergence. START
8      // Method 1:  $s = \sqrt{1 \pm t}$ 
9      // Will use same node twice
10     QuadRule Q = gauleg(n);
11
12     // quadrature done on interval [0, 1], i.e. nodes and weights need to be
13     // shifted
14     for (unsigned i = 0; i < n; ++i) {
15         // Transform nodes
16         double x = (Q.nodes(i) + 1.) / 2.;
17         // Weights
18         double w = Q.weights(i) * x * x * std::sqrt(2. - x * x);
19         // Symmetric summation
20         l += w * (f(x * x - 1) + f(-x * x + 1));
21     }
22
23     // Method 2:  $t = \cos(s)$ 
24     // We are actually using twice the number of nodes than Method 1
25     // QuadRule Q = gauleg(2 * n);
26     //

```

```

27 // quadrature done on interval [-pi/2, pi/2], i.e. nodes and weights need to
28 // be shifted
29 // for(unsigned i = 0; i < 2 * n; ++i) {
30 // Evaluate transformation
31 //     double x = sin(Q.nodes(i) * M_PI_2);
32 // Weights
33 //     double w = Q.weights(i) * cos(Q.nodes(i) * M_PI_2) * cos(Q.nodes(i) *
34 //     M_PI_2); I += w * f(x) * M_PI_2;
35 // }
36 // END
37 return I;
38 }

```

Get it on  GitLab (quadsingint.hpp).



(7-9.d)  Give formulas for the nodes c_j and weights $\tilde{w}_j$ of a $2n$ -point quadrature rule on $[-1, 1]$, whose application to the integrand f will produce the same results as the function `quadsingint` that you implemented in the previous subproblem.

SOLUTION of (7-9.d):

Using Method 1:

Let (x_j, w_j) , $j = 1, \dots, n$ be the Gauss nodes and weights relative to the Gauss quadrature of order n in the interval $[0, 1]$. The nodes are mapped from x_j in $[0, 1]$ to c_l for $l \in 1, \dots, 2n$ in $[-1, 1]$ as follows:

$$c_{2j-i} = (-1)^i (1 - x_j^2), \quad j = 1, \dots, n, \quad i = 0, 1.$$

The weights $\tilde{w}_l$, $l = 1, \dots, 2n$, become:

$$\tilde{w}_{2j-i} = 2w_j x_j^2 \sqrt{2 - x_j^2}, \quad j = 1, \dots, n, \quad i = 0, 1.$$

Using Method 2:

Let (x_j, w_j) , $j = 1, \dots, n$ be the Gauss nodes and weights relative to the Gauss quadrature of order n in the interval $[-1, 1]$. The nodes are mapped from x_j to c_j as follows:

$$c_j = \sin(x_j \pi / 2), \quad j = 1, \dots, n$$

The weights $\tilde{w}_j$, $j = 1, \dots, n$, become:

$$\tilde{w}_j = w_j \cos^2(x_j \pi / 2) \pi / 2.$$



(7-9.e) 2 Implement a C++ function

```
void tabQuadErr(void)
```

that tabulates the quadrature error:

$$\epsilon_i := |W(f) - \text{quadsingint}(f, n)|$$

for $f(t) := \frac{1}{2 + \exp(3t)}$ and $n = 1, 2, \dots, 25$. Then describe the type of convergence observed, see [Lecture → Def. 6.2.2.7].

SOLUTION of (7-9.e):

The convergence is exponential with both methods.

N	<code>quadsingint(f, n)</code>	ϵ_i
1	0.37174	0.111556
2	0.490984	0.00768718
3	0.484771	0.0014743
4	0.483206	9.06151e-05
5	0.483276	2.11519e-05
6	0.483298	1.6518e-06
7	0.483297	2.42568e-07
8	0.483297	2.41621e-08
9	0.483297	1.89626e-09
10	0.483297	3.19954e-10
11	0.483297	4.24399e-12
12	0.483297	3.67528e-12
13	0.483297	2.03004e-13
14	0.483297	3.41949e-14
15	0.483297	4.88498e-15
16	0.483297	1.16573e-15
17	0.483297	3.88578e-16
18	0.483297	3.88578e-16
19	0.483297	6.66134e-16
20	0.483297	4.996e-16
21	0.483297	3.88578e-16
22	0.483297	4.996e-16
23	0.483297	5.55112e-17
24	0.483297	1.11022e-16

C++-code 7.9.3: Implementation of convergence test.

```

2 void tabAndPlotQuadErr() {
3     // Max num of Gauss points to use (in each direction)
4     constexpr unsigned max_N = 25;
5     // "Exact" integral
6     constexpr double I_ex = 0.483296828976607;
7
8     auto f = [](double t) { return 1. / (2. + std::exp(3. * t)); };
9
10    plt::figure();
11    // TODO: (7-9.e) Tabulate and plot quadrature error for the given function.
12    // START
13    std::vector<int> num_nodes(max_N); // number of nodes on x axis
14    std::vector<double> err(max_N);    // error on y axis
15
16    // Table header

```

```

17 std::cout << std::setw(3) << "N" << std::setw(15) << "I_approx"
18     << std::setw(15) << "error" << std::endl;
19
20 for (unsigned N = 1; N <= max_N; ++N) {
21     const double I_approx = quadsingint(f, N);
22     num_nodes[N - 1] = N;
23     err[N - 1] = std::abs(I_ex - I_approx);
24
25     std::cout << std::setw(3)
26         << N
27         // Value of integral
28         << std::setw(15)
29         << I_approx
30         // Error
31         << std::setw(15) << err[N - 1] << std::endl;
32 }
33 plt::semilogy(num_nodes, err, "+", {"label", "Error"});
34 plt::xlabel("No. of quadrature nodes");
35 plt::ylabel("|Error|");
36 plt::title("Convergence of quadsingint");
37 plt::grid("True");
38 // END
39 plt::savefig("./cx_out/convergence.png");
40 }

```

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



End Problem 7-9 , 65 min.