

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 7-14: Aspects of Numerical Quadrature

A quadrature rule approximates the integral of a (continuous) function by means of a weighted sum of function values at so-called quadrature nodes/points. Numerical analysis studies the order of quadrature rules and the asymptotic convergence of the quadrature error as a function of the number of quadrature points.

This problem is linked with [Lecture → Section 7.2], [Lecture → Section 7.4.1], and [Lecture → Section 7.5] and also requires familiarity with [Lecture → § 7.5.0.18].

(7-14.a) 🕒 (4 min.)

Complete the definition of the order of a quadrature rule:

Definition cf. [Lecture → Def. 7.4.1.1]. Order of a quadrature rule

The **order** of a quadrature rule $Q_n : C^0([a, b]) \rightarrow \mathbb{R}$ is defined as

$$\text{order}(Q_n) := \left[\text{ } \right] \left\{ m \in \mathbb{N}_0 : Q_n(\text{ }) = \int_a^b \text{ } dt \quad \forall p \in \text{ } \right\}.$$

SOLUTION of (7-14.a):

Definition cf. [Lecture → Def. 7.4.1.1]. Order of a quadrature rule

The **order** of a quadrature rule $Q_n : C^0([a, b]) \rightarrow \mathbb{R}$ is defined as

$$\text{order}(Q_n) := \max \left\{ m \in \mathbb{N}_0 : Q_n(p) = \int_a^b p(t) dt \quad \forall p \in \mathcal{P}_{m-1} \right\}.$$



(7-14.b) 🕒 (4 min.)

The C++ function

```
unsigned int checkQuadOrder (
    const Eigen::VectorXd &c, const Eigen::VectorXd &w,
    const double tol = 1.0E-10);
```

expects that the input vectors `c` and `w` have the same length and contain the weights and nodes of a quadrature formula on $[-1, 1]$. It returns the **order** of that quadrature formula. Supplement the missing parts of the following listing by writing valid C++ code in the boxes.

```
unsigned int checkQuadOrder (
    const Eigen::VectorXd &c, const Eigen::VectorXd &w,
    const double tol = 1.0E-10) {
    const unsigned long N = c.size();
    assert(N == w.size());

    for (unsigned int d = 0; d < [ ] ; ++d) {
        double s = 0.0;
        for (int j = 0; j < [ ] ; ++j) {
```

```

    s +=  * std::pow(, d);
}

double val = (d % 2 == 0) ?  : ;

if (std::abs(s - val) > tol) {
    return ;
}

return ;
}

```

HINT 1 for (7-14.b):

- The C++ operator % gives the remainder of integer division: $p \% q \triangleq p \bmod q$.
- `std::pow(x, n)` computes x^n for $n \in \mathbb{N}_0$, when x is a floating-point type, and n a cardinal number. By specification `std::pow(0.0, 0) == 1.0`.

Writing $Q(f)$ for the evaluation of the quadrature formula for a function f , we have to find the largest $d \in \mathbb{N}_0$ such that

$$Q(\{t \mapsto t^d\}) = \int_{-1}^1 t^d dt = \begin{cases} 2/d+1 & \text{for even } d, \\ 0 & \text{for odd } d. \end{cases}$$

SOLUTION of (7-14.b):

C++ code 7.14.1: Implementation of function `checkQuadOrder()`

```

2  unsigned int checkQuadOrder(const Eigen::VectorXd &c, const Eigen::VectorXd &w,
3                             const double tol = 1.0E-10) {
4      const unsigned long N = c.size(); // Number of quadrature points
5      assert(N == w.size());           // Numbers of nodes and weights must
6      // Check for exactness for polynomials of degree d
7      // Note that the highest possible order is 2N!
8      for (unsigned int d = 0; d < 2 * N; ++d) {
9          // j-loop performs the evaluation of the quadrature formula
10         // for the monomial  $t \mapsto t^d$ .
11         double s = 0.0;
12         for (int j = 0; j < N; ++j) {
13             s += w[j] * std::pow(c[j], d);
14         }
15         // Exact value:  $2/(d+1)$  for even degree, zero else.
16         double val = (d % 2 == 0) ? 2.0 / (d + 1) : 0.0;
17         // Safe test for equality of computed floating point numbers
18         if (std::abs(s - val) > tol) {
19             return d;
20         }
21     }
22     // The quadrature rule has the highest possible order
23     return 2 * N;
24 }

```



(7-14.c) (3 min.) The C++ function

```
template <typename FUNCTOR>
double trapezoidalRule (
    FUNCTOR &&f, double a, double b, unsigned int N);
```

evaluates the **equidistant trapezoidal quadrature rule** with N nodes on the interval $[a, b]$ for a (continuous) function passed through the functor object f .

Write valid C++ code in the boxes of the following listing so that the function meets this specification.

```
template <typename FUNCTOR>
double trapezoidalRule (
    FUNCTOR &&f, double a, double b, unsigned int N) {
    assert (a < b);
    assert (N >= 2);

    const double h =  ;

    double t = a+h;
    double s =  ;

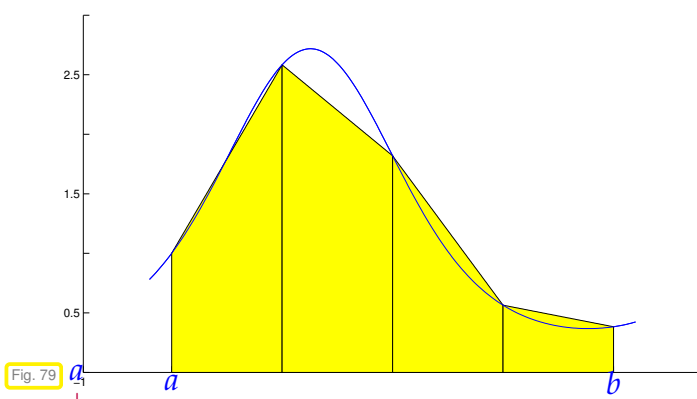
    for (int j = 1; j <  ; ++j) {

        s +=  ;
        t += h;
    }

    s +=  ;

    return  ;
}
```

HINT 1 for (7-14.c):



Recall that the trapezoidal rule approximates the integral $\int_a^b f(t) dt$ by the area under an interpolating polygonal line.

SOLUTION of (7-14.c):

C++ code 7.14.2: Implementation of function `trapezoidalRule()`

```

2  template <typename FUNCTOR>
3  double trapezoidalRule(FUNCTOR &&f, double a, double b, unsigned int N) {
4      assert(a < b);
5      assert(N >= 2);
6      // Spacing of the N equidistant quadrature nodes
7      const double h = (b - a) / (N - 1);
8      // Variable containing location of current quadrature node
9      double t = a + h;
10     // Left endpoint has weight 0.5*h
11     double s = 0.5 * f(a);
12     // Summing contributions of N-2 interior quadrature nodes with
        weight h
13     for (int j = 1; j < N - 1; ++j) {
14         s += f(t);
15         t += h;
16     }
17     // Right endpoint has weight 0.5*h
18     s += 0.5 * f(b);
19     // Scaling with h is performed in the end. Can also be done when
        accumulating
20     // value of quadrature formula.
21     return h * s;
22 }

```



(7-14.d) 🕒 (4 min.)

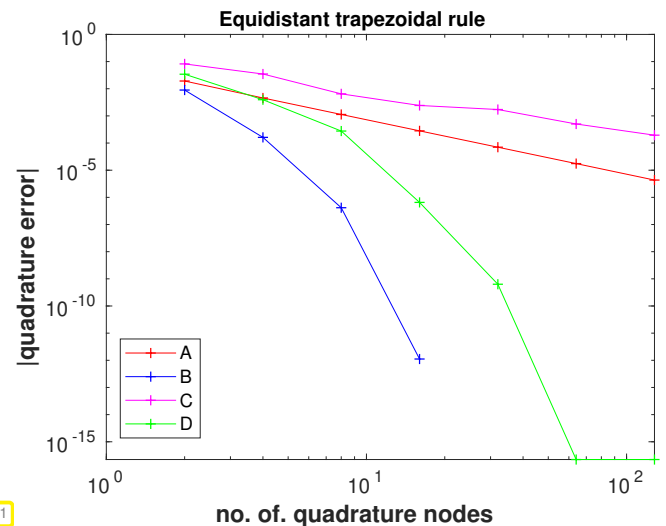
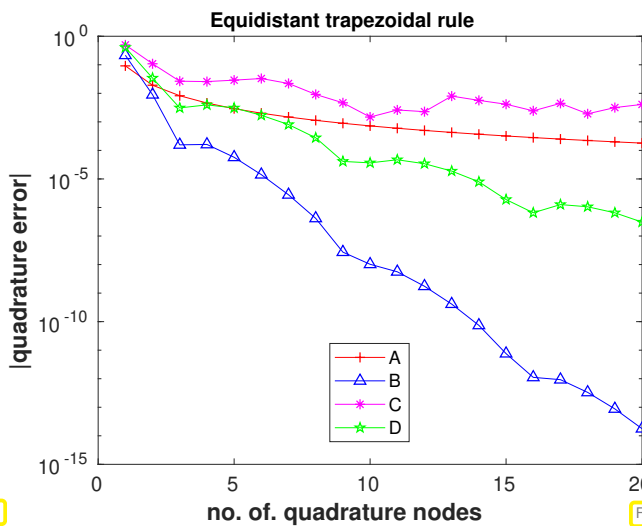
The **equidistant trapezoidal quadrature rule** as implemented in Subproblem (7-14.c) is used for the approximate evaluation of the integrals

$$\int_a^b f_{\alpha,\beta}(t) dt, \quad a = 0.33, b = 1.33,$$

where the family of functions $f_{\alpha,\beta} : \mathbb{R} \rightarrow \mathbb{R}$ is defined as

$$f_{\alpha,\beta}(t) := \sqrt{|1 + \alpha \sin(\beta t)|}, \quad \alpha, \beta, t \in \mathbb{R}.$$

The following plots display the modulus of the quadrature error for certain members of this family of functions as a function of the number of quadrature nodes.

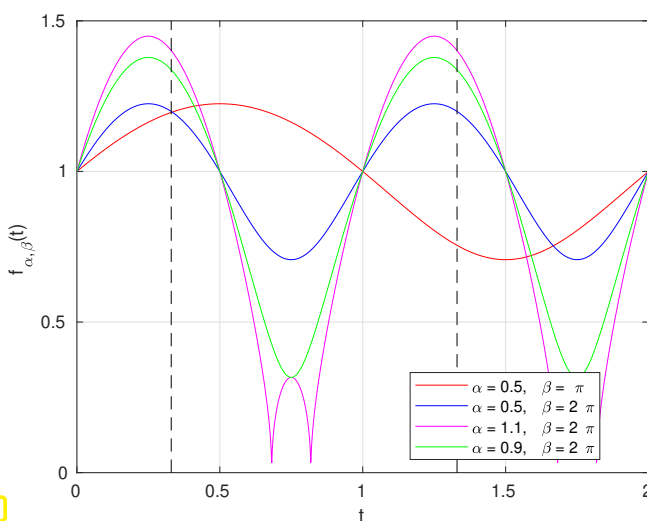


State, which graph (A,B,C, or D) belongs to which pair of parameters α, β (defining the integrand $f_{\alpha,\beta}$).

Parameters	$\leftrightarrow$	Graph
(1) $\alpha = 0.9, \beta = 2\pi$	$\leftrightarrow$	
(2) $\alpha = 0.5, \beta = \pi$	$\leftrightarrow$	
(3) $\alpha = 1.1, \beta = 2\pi$	$\leftrightarrow$	
(4) $\alpha = 0.5, \beta = 2\pi$	$\leftrightarrow$	

SOLUTION of (7-14.d):

This sub-problem is closely related to [Lecture $\rightarrow$ Exp. 7.5.0.16] and [Lecture $\rightarrow$ § 7.5.0.18]. See there for in-depth explanations.



◁ The four functions from the $f_{\alpha,\beta}$ family, for whom plots of the quadrature errors have been provided.

The colors used correspond to the colors of the associated error graphs.

In the sequel, n stands for the number of quadrature nodes.

- (1) $\alpha = 0.9, \beta = 2\pi$: the function is 1-periodic and analytic in a strip along the real axis. We expect exponential asymptotic convergence for $n \rightarrow \infty$, reflected by aligned error points in a linear-

logarithmic plot Fig. 80. This expectation is matched by graphs B or D. Since B is already claimed, just **D** remains.

- (2) $\alpha = 0.5$, $\beta = \pi$: the function is smooth, but **not** 1-periodic. Thus we can just expect regular $O(n^{-2})$ asymptotic convergence of the trapezoidal rule for $n \rightarrow \infty$. By exclusion we conclude that this must correspond to graph **A**.
- (3) $\alpha = 1.1$, $\beta = 2\pi$: The function has root-type singularities where $1 + 1.1 \sin(2\pi t) = 0$. Hence for this non-smooth integrand we cannot expect full $O(n^{-2})$ algebraic convergence, let alone exponential convergence. This function will result in the slowest observed convergence of the quadrature error, hence graph **C**.
- (4) $\alpha = 0.5$, $\beta = 2\pi$: Same as for (1) but with a wider strip of analyticity along the real axis, hence faster exponential convergence than (1), graph **B**.

The best way to approach this sub-problem is to rank the different functions according to the expected speed of convergence of the quadrature error. Appealing to the properties of the functions discussed above we obtain:

$$(3) < (2) < (1) < (4),$$

where “ $<$ ” means “expected to converge to zero more slowly than”. A corresponding ranking of the error graphs is evident:

$$(C) < (A) < (D) < (B).$$

This yields the right association of functions and graphs.



End Problem 7-14, 15 min.