

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 7-8: Discretization of an integral operator

In this problem we consider a completely new concept, an **integral operator** of the form

$$(\mathbb{T}f)(x) = \int_I k(x, y) f(y) \, dy, \quad x \in I \subset \mathbb{R}, \quad (7.8.1)$$

where the **kernel** k is continuous on $I \times I$, $I \subset \mathbb{R}$ a closed interval. It maps a function on I to another function on I . Such integral operators are very common in models of continuous physics. Of course, their numerical treatment heavily draws on numerical quadrature.

Requires knowledge about Gauss quadrature from [Lecture → Section 7.4] and involves substantial implementation in C++.

Given $f \in C^0([0, 1])$, the integral

$$g(x) := \int_0^1 e^{|x-y|} f(y) \, dy$$

defines a function $g \in C^0([0, 1])$. We approximate the integral by means of n -point Gauss quadrature, which yields a function $g_n(x)$.

(7-8.a) ☐ Let $\{\xi_j^n\}_{j=1}^n$ be the nodes for the Gauss quadrature on the interval $[0, 1]$. Assume that the nodes are ordered, namely $\xi_j^n < \xi_{j+1}^n$ for every $j = 1, \dots, n-1$. We can write

$$(g_n(\xi_l^n))_{l=1}^n = M(f(\xi_j^n))_{j=1}^n,$$

for a suitable matrix $M \in \mathbb{R}^{n,n}$. Give a formula for the entries of M .

SOLUTION of (7-8.a):

Applying Gauss quadrature to the expression for $g(\xi_l^n)$ we obtain $g_n(\xi_l^n) = \sum_{j=1}^n w_j^n e^{|\xi_l^n - \xi_j^n|} f(\xi_j^n)$, whence $M_{lj} = w_j^n e^{|\xi_l^n - \xi_j^n|}$ for $l, j = 1, \dots, n$.



(7-8.b) ☐ Implement a function

```
template<typename Function>
Eigen::VectorXd comp_g_gausspts(Function f, unsigned int n)
```

that computes the n -vector $(g_n(\xi_l^n))_{l=1}^n$ with optimal complexity $O(n)$ (excluding the computation of the nodes and weights), where f is an object with an evaluation operator, e.g. a lambda function, that represents the function f .

You may use the provided function

```
void gaussrule(int n, Eigen::VectorXd & w, Eigen::VectorXd & xi)
```

that computes the weights w and the ordered nodes xi relative to the n -point Gauss quadrature on the interval $[-1, 1]$.

SOLUTION of (7-8.b):

In order to write a function with optimal complexity, we need to use the structure of the matrix M . Write

$$g_n(\xi_l^n) = \sum_{j=1}^n w_j^n e^{|\xi_l^n - \xi_j^n|} f(\xi_j^n) = \sum_{j=1}^{l-1} w_j^n e^{\xi_l^n} e^{-\xi_j^n} f(\xi_j^n) + \sum_{j=l}^n w_j^n e^{-\xi_l^n} e^{\xi_j^n} f(\xi_j^n).$$

Setting $p_l^n = \sum_{j=1}^{l-1} w_j^n e^{-\xi_j^n} f(\xi_j^n)$ and $q_l^n = \sum_{j=l}^n w_j^n e^{\xi_j^n} f(\xi_j^n)$, this identity can be rewritten as

$$g_n(\xi_l^n) = e^{\xi_l^n} p_l^n + e^{-\xi_l^n} q_l^n.$$

This expression for $l = 1, \dots, n$ can be computed with $O(n)$ operations. It remains to compute p_l^n and q_l^n in $O(n)$ operations. This can be done by using these recursive formulas, that immediately follow from the definition: for every $l = 1, \dots, n-1$ we have

$$\begin{aligned} p_1^n &= 0, & q_n^n &= w_n^n e^{\xi_n^n} f(\xi_n^n), \\ p_{l+1}^n &= p_l^n + w_l^n e^{-\xi_l^n} f(\xi_l^n), & q_l^n &= q_{l+1}^n + w_l^n e^{\xi_l^n} f(\xi_l^n). \end{aligned}$$

C++-code 7.8.2: Implementation of `comp_g_gausspts`.

```

2  template <typename Function>
3  Eigen::VectorXd comp_g_gausspts(Function f, unsigned int n) {
4      Eigen::VectorXd g = Eigen::VectorXd::Zero(n);
5
6      // TODO: (7-8.b) Compute  $(g_n(\xi_l^n))_{l=1}^n$  with optimal complexity
7      //  $O(n)$ . START
8      Eigen::VectorXd w(n), xi(n);
9      gaussrule(n, w, xi); // Compute Gauss nodes and weights relative to
10                          // [-1,1]
11
12      Eigen::VectorXd p(n), q(n), expxi(n), fxi(n);
13      w = w / 2.; // Rescale the weights to [0,1]
14      xi = (xi + Eigen::MatrixXd::Ones(n, 1)) / 2.; // Rescale the nodes to [0,1]
15
16      // Evaluate at points needed later on
17      for (unsigned int l = 0; l < n; l++) {
18          fxi(l) = f(xi(l));
19          expxi(l) = std::exp(xi(l));
20      }
21
22      // Initialize the Eigen::VectorXds p and q; base cases for recursive
23      // formulas
24      p(0) = 0;
25      q(n-1) = w(n-1) * expxi(n-1) * fxi(n-1);
26
27      // Fill in the Eigen::VectorXds p and q ( $O(n)$  complexity)
28      for (unsigned int l = 0; l < n-1; l++) {
29          p(l+1) = p(l) + w(l) / expxi(l) * fxi(l);
30          q(n-2-l) = q(n-1-l) + w(n-2-l) * expxi(n-2-l) * fxi(n-2-l);
31      }
32
33      // Finally, constructing the output ( $O(n)$  complexity)
34      g = (p.array() * expxi.array() + q.array() / expxi.array()).matrix();
35      // END
36      return g;
37  }

```

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



(7-8.c)  Test your implementation and write a C++ function

double testCompGGaussPts (void) ;

that computes $g(\xi_{11}^{21})$ for $f(y) = e^{-|0.5-y|}$. What result do you expect?

SOLUTION of (7-8.c):

For reasons of symmetry, we have $\xi_{11}^{21} = 1/2$, and so $g_{21}(\xi_{11}^{21})$ should be equal to 1 for $f(y) = e^{-|0.5-y|}$, since Gauss quadrature is exact for constant functions.

C++-code 7.8.3: Testing code.

```

2  double testCompGGaussPts () {
3      double g_val = 0;
4
5      // TODO: (7-8.c) Compute  $g(\xi_{11}^{21})$  for  $f(y) = e^{-|0.5-y|}$ .
6      // START
7      constexpr unsigned int n = 21;
8      auto f = [](double y) { return std::exp(-std::abs(.5 - y)); };
9      const Eigen::VectorXd g = comp_g_gausspts(f, n);
10     g_val = g((n + 1) / 2 - 1);
11     // END
12     return g_val;
13 }
```

Get it on  GitLab (gauss_pts.hpp).



End Problem 7-8