ETH Lecture 401-0663-00L Numerical Methods for CSE

Homework Problems

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2016

URL: https://www.sam.math.ethz.ch/ grsam/HS16/NumCSE/NCSEProblems.pdf

0.1 Current Assignment

Christmas homework problems

- Problem 11.4
- Problem 12.4 (**Core problem**)
- Problem 12.5
- Problem 12.6 (**Core problem**)
- Problem 12.7

(0.1.2) Homework problems 9.12.2016 - 19.12.2016

- Problem 11.2 (**Core problem**)
- Problem 11.3 (**Core problem**)
- Problem 11.5
- Problem 11.1

(0.1.3) Homework problems 2.12.2016 - 12.12.2016

- Problem 8.4 (**Core problem**)
- Problem 8.5 (**Core problem**)
- Problem 8.7 (**Core problem**)

- Problem 7.1
- Problem 8.9

(0.1.4) Homework problems 25.11.2016 - 05.12.2016

- Problem 7.3 (**Core problem**)
- Problem 7.5 (**Core problem**)
- Problem 7.4
- Problem 7.6

(0.1.5) Homework problems 17.11.2016 - 28.11.2016

- Problem 5.6
- Problem 6.2 (**Core problem**)
- Problem 6.3 (**Core problem**)
- Problem 6.4

(0.1.6) Homework problems 11.11.2016 - 21.11.2016

- Problem 5.3 (**Core problem**)
- Problem 5.1 (**Core problem**)
- Problem 5.5
- Problem 5.4

(0.1.7) Homework problems 4.11.2016 - 14.11.2016

- Problem 4.3 (**Core problem**)
- Problem 4.4 (**Core problem**)
- Problem 4.2
- Problem 4.1

(0.1.8) Homework problems 28.10.2016 - 07.11.2016

- Problem 3.7 (**Core problem**)
- Problem 3.8
- Problem 3.9 (**Core problem**)

- Problem 3.10 (**Core problem**)

(0.1.9) Homework problems 21.10.2016 - 31.10.2016

- Problem 3.5 (**Core problem**)
- Problem 3.4 (**Core problem**)
- Problem 3.2
- Problem 3.1

(0.1.10) Homework problems 14.10.2016 - 24.10.2016

- Problem 2.9 (**Core problem**)
- Problem 2.10
- Problem 2.11
- Problem 2.13 (**Core problem**)
- Problem 2.12
- Problem 2.8

(0.1.11) Homework problems 8.10.2016 - 17.10.2016

- Problem 2.3
- Problem 2.5 (**Core problem**)
- Problem 2.4
- Problem 2.6 (**Core problem**)

(0.1.12) Homework problems 30.9.2016 - 10.10.2016

- Problem 1.8 (**Core problem**)
- Problem 1.10
- Problem 1.11
- Problem 2.2 (**Core problem**)

(0.1.13) Homework problems 22.9.2016 - 3.10.2016

- Problem 1.2 [discussed on Monday, Sep 26]
- Problem 1.1 [discussed on Monday, Sep 26]

- Problem 1.5
- Problem 1.3 (**Core problem**)
- Problem 1.7 (**Core problem**)

Contents

0.2 General Information

0.2.1 Weekly Homework Assignments

All problems will be published in this single ".pdf" file. Every week, we publish a list of problems (cf. Section Section 0.1) that should be solved. We denote by "**Core problem**" problems that you should really try to solve, whenever you do not have much time to dedicate to the homework.

0.2.2 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 in a difficult problem and you tried to find a solution for a sufficient amount of time.

Make sure to practise with the `Linux` environment. During the exam you will have to work on `Fedora` with `Gnome3`: there will be no time to become familiar with this environment.

0.2.3 Corrections and Grading of Assignments

Place your handwritten solutions by Thursday night in the pigeon-holes in front of HG G 53.2. You can submit your codes to the assistants using the online submission interface. You should submit your solutions even if those are incomplete and/or incorrect. Feedback from the assistants is most useful when incomplete/incorrect solutions are returned.

Please do not submit solutions you copied from others or from the published solutions. Assignments are not graded.

You will obtain feedback about your submissions on the subsequent Monday, during the exercise class.

0.2.4 Codes and Templates

For each problem involving coding you will find templates on GitLab (the specific link is provided in the problem sheet). Templates may be used as starting point for your solutions. All solutions we provide will be based on the templates. You are not forced to use the templates, but using them will allow you to focus on the problems. This will also simplify the correction from the assistants. A similar workflow will also be used during the exam.

All templates come with a self-contained `CMake` file that can be used to compile only the problem you are working on.

There are many ways to obtain templates and solutions:

1. Clone the entire GitLab repository and navigate to the folders:
   - `Assigmnents/Codes/<Chapter>/<ProblemName>/templates_nolabels` for templates.
   - `Assigmnents/Codes/<Chapter>/<ProblemName>/solutions_nolabels` for solutions.
2. Follow the link you find in this ".pdf" and use the button "Download zip".
3. Use the ".zip" files we provide on the website.

Choose the way you prefer.

0.2.5 Hints and Solutions

Hints and solutions are kept in separate ".pdf" files.

You can download the ".pdf" files with hints and solutions from the website.

Chapter 1

Computing with Matrices and Vectors

Problem 1.1: Arrow matrix×vector multiplication

Innocent looking linear algebra operation can burn considerable CPU power when implemented carelessly, see Code 1.3.11. In this problem we study operations involving so-called arrow matrices, that is, matrices, for which only a few rows/columns and the diagonal are populated, see the `spy`-plots in Rem. 1.3.5.

Templates: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Let $n \in \mathbb{N}, n > 0$. An "arrow matrix" $\mathbf{A} \in \mathbb{R}^{n \times n}$ is constructed given two vectors $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{d} \in \mathbb{R}^n$. The matrix is then squared and multiplied with a vector $\mathbf{x} \in \mathbb{R}^n$. This is implemented in the following C++ function:

C++11-code 1.0.1: Computing $\mathbf{A}^2\mathbf{y}$ for an arrow matrix $\mathbf{A}$

```
2  void arrow_matrix_2_times_x(const VectorXd &d, const VectorXd &a,
3                              const VectorXd &x, VectorXd &y) {
4      assert(d.size() == a.size() && a.size() == x.size() &&
5             "Vector size must be the same!");
6      int n = d.size();
7
8      VectorXd d_head = d.head(n-1);
9      VectorXd a_head = a.head(n-1);
10     MatrixXd d_diag = d_head.asDiagonal();
11
12     MatrixXd A(n,n);
13
14     A << d_diag,              a_head,
15          a_head.transpose(), d(n-1);
16
17     y = A*A*x;
18 }
```

Get it on GitLab (arrowmatvec.cpp).

(1.1.a) ⚀ For general vectors $\mathbf{d} = [d_1, \ldots, d_n]^\top$ and $\mathbf{a} = [a_1, \ldots, a_n]^\top$ sketch the pattern, that is, the `spy`-plot, of the matrix $\mathbf{A}$ created in the function `arrow_matrix_2_times_x` in Code 1.0.1.

HIDDEN HINT 1 for (1.1.a) → `1.1.1:arrmatha.pdf`

SOLUTION for (1.1.a) → `1.1.1:arrws.pdf` ▲

(1.1.b) ⚁

We measure the runtime of the function `arrow_matrix_2_times_x` with respect to the matrix size n and plot the results in Figure 1. (Get it on GitLab (arrowmatvec.cpp).)
The plot shows timings for `arrow_matrix_2_times_x` (log-log plot).

Machine details: *Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz. Compiled with: gcc 6.2.1 (flags: -O3).* ▷

Give a detailed description of the behavior of the measured runtimes and an explanation for it.

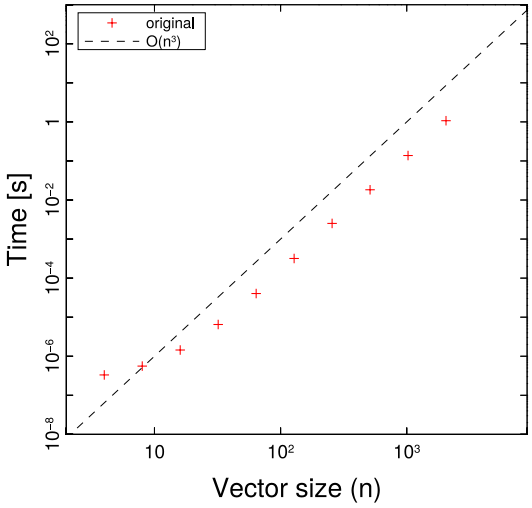


Fig. 1

SOLUTION for (1.1.b) → `1.1.2:arrwc.pdf` ▲

(1.1.c) ⚂ Write an *efficient* C++ function

```
void efficient_arrow_matrix_2_times_x(const VectorXd &d,
                                      const VectorXd &a,
                                      const VectorXd &x,
                                      VectorXd &y)
```

that computes the same product as in Code 1.0.1, but with optimal asymptotic complexity with respect to n. Compute the complexity of your code and explain why you obtain such result.

Here `d` passes the vector $[d_1, \ldots, d_n]^\top$ and `a` passes the vector $[a_1, \ldots, a_n]^\top$.

SOLUTION for (1.1.c) → `1.1.3:arrwi.pdf` ▲

(1.1.d) ⚃ What is the asymptotic complexity of your algorithm from sub-problem (1.1.c) (with respect to matrix size n)?

SOLUTION for (1.1.d) → `1.1.4:arrwce.pdf`

▲

(1.1.e) ⚄ Compare the runtime of your implementation and the implementation given in Code Code 1.0.1 for $n = 2^5, \ldots, 2^{12}$. Beware, for large n ($n > 2048$) the computations may take a long time.

Output the times in seconds, using 3 decimal digits in scientific notation. You can use `std::setw`, `std::precision(int)` and `std::scientific` from the standard library to output formatted text (include `iomanip`). For example:

```
1 std::cout << std::setw(8) << 1./3e4
2           << std::scientific << std::setprecision(3)
3           << std::setw(15) << 1./3e-9;
```

Remark: run your code using optimization flags (`-O3`). With `CMake`, you can achieve this using `-DCMAKE_BUILD_TYPE=Release` as a `CMake` option.

(1.0.6) Measuring runtimes in a C++ code

In order to measure runtimes you have two options: either you use `std::chrono` or use the **Timer** class.

How to use `Timer`

If you want to time a code, include `timer.h`, create a new `Timer` object, as demonstrated in the following code.

C++11-code 1.0.7: Usage of Timer

```
1  Timer t;
2  t.start();
3  // HERE CODE TO TIME
4  t.stop();
5
6  // Now you can get the time passed between start and stop using
7  t.duration();
8  // You can start() and stop() again, a number of times
9  // Ideally: repeat experiment many times and use min() to obtain
       the
10 // fastest run
11 t.min();
12 // You can also obtain the mean():
13 t.mean();
```

All times will be outputted in seconds. `Timer` is simply a wrapper to `std::chrono`.

Advanced user: how to use `std::chrono`

Find the documentation of `chrono` on C++ reference.

First include the `chrono` STL header file (note: this requires C++11). The `chrono` header provides the function:

C++11-code 1.0.8: `now()` function

```
1 std::chrono::high_resolution_clock::time_point
2     std::chrono::high_resolution_clock::now();
```

that returns the current time using the highest possible precision offered by the machine. For simplicity, we rename the return type:

C++11-code 1.0.9: `now()` function

```
1 using time_point_t = std::chrono::high_resolution_clock::time_point;
2 // Declare a starting point and a termination time
3 time_point_t start, end;
```

The difference between two objects of type `time_point_t` (e.g. `end - start`) is an object of type `std::chrono::high_resolution_clock::duration`. In order to convert the `chrono`'s duration type (which, in principle, can be anything: seconds, milliseconds, ...), to a fixed duration (say nanoseconds, `std::chrono::nanoseconds`), use the `duration_cast`:

```
using duration_t = std::chrono::nanoseconds;
duration_t elapsed = std::chrono::duration_cast<duration_t>(end - start);
```

To obtain the actual number (as integer) used to represent `elasped`, use `elapsed.count()`.

Note: the data type used to represent `std::chorno::seconds`, `std::chorno::milliseconds`, etc. is of integer type. Thus, you cannot obtain fractions of this units. Make sure you use a sufficiently "refined" unit of measure (e.g. `std::chrono::nanoseconds`).

SOLUTION for (1.1.e) → `1.1.5:arrwc.pdf` ▲

End Problem 1.1

Problem 1.2: Gram-Schmidt orthonormalization with EIGEN

In ➜Ex. 0.2.41 and ➜Code 0.2.42 you can find an implementation of the famous Gram-Schmidt orthonormalization algorithm from linear algebra. That code makes use of a rather simple (and inefficient) implementation of vector and matrix classes.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

(1.2.a) ⚀ Determine the asymptotic complexity of the function `gramschmidt()` from ➜Code 0.2.42 in terms of the matrix dimension n, if the argument matrix `A` has size $n \times n$ and no premature termination occurs.

SOLUTION for (1.2.a) → `1.2.1:grsi.pdf` ▲

(1.2.b) ⚁ Based on the C++ linear algebra library EIGEN, implement a function that performs the same computations as ➜Code 0.2.42:

```
1 MatrixXd gram_schmidt(const MatrixXd &A);
```

The output vectors should be returned as the columns of a matrix.

Use EIGENs block operations see Eigen documentation.

SOLUTION for (1.2.b) → `1.2.2:grsi.pdf` ▲

(1.2.c) ⚂ Test your implementation by applying the function `gram_schmidt` to a small random matrix and checking the orthonormality of the columns of the output matrix.

HIDDEN HINT 1 for (1.2.c) → `1.2.3:sp2mn.pdf`

HIDDEN HINT 2 for (1.2.c) → `1.2.3:sp2QQ.pdf`

SOLUTION for (1.2.c) → `1.2.3:grst.pdf`

▲

End Problem 1.2

Problem 1.3: Kronecker product

In Def. 1.4.17 we learned about the so-called Kronecker product. In this problem we revisit the discussion of Ex. 1.4.18.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

(1.3.a) ⚀ Compute the Kronecker product $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ of the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

SOLUTION for (1.3.a) → `1.3.1:cmpm.pdf` ▲

(1.3.b) ⚁ Implement a C++ function

```
void kron(const MatrixXd & A, const MatrixXd & B, MatrixXd & C);
```

that computes the Kronecker product of the argument matrices `A` and `B` and stores the result in the matrix `C`. You can use Eigen "block" operations, see the Eigen documentation.

SOLUTION for (1.3.b) → `1.3.2:cmpk.pdf` ▲

(1.3.c) ⚁ What is the asymptotic complexity (Def. 1.4.4) in terms of the problem size parameter $n \to \infty$ of your implementation of `kron` from (1.3.b), when we pass $n \times n$ square matrices as arguments.

You may use the *Landau* symbol from Def. 1.4.5 to state your answer.

SOLUTION for (1.3.c) → `1.3.3:cmpk.pdf` ▲

(1.3.d) ⚂ In general, computing the entire matrix is unnecessary. Devise an *efficient* implementation of an EIGEN-based C++ function

```
void kron_mult(const MatrixXd & A, const MatrixXd & B,1
const VectorXd & x, VectorXd & y);
```

for the computation of $\mathbf{y} = (\mathbf{A} \otimes \mathbf{B})\mathbf{x}$ for *square matrices* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,n}$. Do not use reshaping through the `Map` method of EIGEN's matrix classes; use access to sub-vectors by the `segment` method instead. The meaning of the arguments of `kron_mult` should be self-explanatory.

HIDDEN HINT 1 for (1.3.d) → `1.3.4:h1.pdf`

SOLUTION for (1.3.d) → `1.3.4:cmpk.pdf` ▲

(1.3.e) ⚂ Devise another efficient implementation of a C++ code for the computation of $\mathbf{y} = (\mathbf{A} \otimes \mathbf{B})\mathbf{x}$, $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,n}$. This time use Eigen "reshaping" functions. The function to implement is:

```
void kron_reshape(const MatrixXd &A, const MatrixXd &B,
                  const VectorXd &x, VectorXd &y);
```

Study Rem. 1.2.27 about "reshaping" matrices in EIGEN. See also the Eigen documentation.

HIDDEN HINT 1 for (1.3.e) → `1.3.5:.pdf`

SOLUTION for (1.3.e) → `1.3.5:cmpk.pdf` ▲

(1.3.f) ⚀ Compare the runtimes of your implementations in sub-problems (1.3.b), (1.3.d) and (1.3.e). To this end, measure the runtime of the functions `kron`, `kron_mult`, `kron_reshape`, with $n = 2^1, \ldots, 2^8$. Repeat the experiments 10 times and consider only the smallest runtime (skip the computations with `kron` for $n > 2^6$). You can use the class `Timer` or the STL class `std::chrono` as explained in Problem 1.1, § 1.0.6.

Report the measurements in seconds with scientific notation using 3 decimal digits.

SOLUTION for (1.3.f) → `1.3.6:cmpk.pdf` ▲

End Problem 1.3

Problem 1.4: Fast matrix multiplication with EIGEN

➜Rem. 1.4.10 presents Strassen's algorithm that can achieve the multiplication of two dense square matrices of size $n = 2^k$, $k \in \mathbb{N}$, with an asymptotic complexity better than $O(n^3)$ (which is the complexity of a loop-based matrix multiplication). This problem centers around the implementation of this algorithm in EIGEN.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

(1.4.a) ⚀ Using EIGEN , implement a recursive function

```
MatrixXd strassenMatMult(const MatrixXd &A, const MatrixXd &B);
```

that uses Strassen's algorithm (➜Rem. 1.4.10) to multiply the two square matrices $\mathbf{A}$ and $\mathbf{B}$ of size $n = 2^k$, $k \in \mathbb{N}$, and returns the result as output.

You can use the template we provide. Get it on GitLab (strassen.hpp).

SOLUTION for (1.4.a) → `1.4.1:fastmi.pdf` ▲

(1.4.b) ⚁ Validate the correctness of your code by comparing the result with EIGEN's built-in matrix multiplication.

You can use the template. Get it on GitLab (test.cpp).

HIDDEN HINT 1 for (1.4.b) → `1.4.2:hs1.pdf`

SOLUTION for (1.4.b) → `1.4.2:fastmv.pdf` ▲

(1.4.c) ⚁ Measure the runtime of your function `strassenMatMult` for random matrices of sizes 2^k, $k = 4, \ldots, 9$, and compare with the matrix multiplication offered by the $*$-operator of EIGEN.

You can use the class `Timer` or the STL class `std::chrono` as explained in Problem 1.1, § 1.0.6. Base your code on the supplied template. Get it on GitLab (test.cpp).

Recommendation: Use the optimization capabilities of your C++ compiler. With `CMake`, this can be done by appending `-DCMAKE_BUILD_TYPE=Release` to the `CMake` invocation. Please note that this also disables various integrity checks.

Warning: For big values of n, the computations may take some time. Consider reducing n while debugging.

SOLUTION for (1.4.c) → `1.4.3:fastmt.pdf` ▲

End Problem 1.4

Problem 1.5: Householder reflections

This problem is a supplement to ➜Section 1.5.1 and is related to Gram-Schmidt orthogonalization, see ➜Code 1.5.4.
For solving this problem, it is useful to remember the QR-decomposition of a matrix from linear algebra, see also ➜Thm. 3.3.9. This problem is meant to practise some (advanced) linear algebra skills.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

(1.5.a) ⚀

Let $\mathbf{v} \in \mathbb{R}^n$, $n \in \mathbb{N}, n > 0$. Consider the following algorithm:

Algorithm 1.0.20: Householder reflection

$\mathbf{w} \leftarrow \mathbf{v}/\|\mathbf{v}\|_2$
$\mathbf{u} \leftarrow \mathbf{w}$
$\mathbf{u}_1 \leftarrow \mathbf{u}_1 + 1$
$\mathbf{q} \leftarrow \mathbf{u}/\|\mathbf{u}\|_2$
$\mathbf{X} \leftarrow \mathbf{I} - 2\mathbf{q}\mathbf{q}^\top$
$\mathbf{Z} \leftarrow (\mathbf{X})_{:,2:n}$

Write a C++ function with signature:

```
void houserefl(const VectorXd &v, MatrixXd &Z);
```

that implements Code 1.0.20. Use data types from EIGEN.

HIDDEN HINT 1 for (1.5.a) → `1.5.1:hrfH1.pdf`

SOLUTION for (1.5.a) → `1.5.1:impl.pdf` ▲

(1.5.b) ⚁ Show that the matrix $\mathbf{X}$, defined in Code 1.0.20, satisfies:

$$\mathbf{X}^\top \mathbf{X} = \mathbf{I}_n$$

Here $\mathbf{I}_n$ is the identity matrix of size n.

HIDDEN HINT 1 for (1.5.b) → `1.5.2:norm.pdf`

SOLUTION for (1.5.b) → `1.5.2:orth.pdf` ▲

(1.5.c) ⚂ Show that the first column of $\mathbf{X}$, in Code 1.0.20, is a multiple of the vector $\mathbf{v}$.

HIDDEN HINT 1 for (1.5.c) → `1.5.3:mulh.pdf`

SOLUTION for (1.5.c) → `1.5.3:muls.pdf` ▲

(1.5.d) ⚁ What property does the set of columns of the matrix $\mathbf{Z}$ have? What is the purpose of the function `houserefl`?

SOLUTION for (1.5.d) → `1.5.4:propertyimpl.pdf` ▲

(1.5.e) ⊡ What is the asymptotic complexity of the function `houserefl` as the length n of the input vector $\mathbf{v}$ tends to ∞?

Specify it in leading order using the Landau symbol O.

SOLUTION for (1.5.e) → `1.5.5:impl.pdf` ▲

End Problem 1.5

Problem 1.6: Matrix powers

This problems studies a (moderately) efficient way to compute large integer powers of matrices in EIGEN.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

This problems deals with integer powers of square matrices, defined as:

$$\mathbf{A}^k := \overbrace{\mathbf{A}\cdot\ldots\cdot\mathbf{A}}^{k \text{ times}}, \quad k \in \mathbb{N}$$

for $\mathbf{A} \in \mathbb{C}^{n\times n}$ and $n \in \mathbb{N}$, $n > 0$.

(1.6.a) ⚄ Implement a C++ function

```
void matPow(MatrixXcd & A, unsigned int k);
```

that, using only basic linear algebra operations (including matrix-vector or matrix-matrix multiplications), computes the k^{th} power of the $n \times n$ matrix $\mathbf{A}$ as efficiently as possible. You can use the provided template. Get it on GitLab (matPow.cpp).

Here, `MatrixXcd` is a *complex matrix*. A complex matrix is similar to `MatrixXd`, and differs only by the fact that if contains elements of $\mathbb{C}$.

HIDDEN HINT 1 for (1.6.a) → `1.6.1:.pdf`

Remark: a `MatrixXcd` is defined internally (in EIGEN), as:

```
using MatrixXcd = Matrix<std::complex, Dynamic, Dynamic>;
```

where `Dynamic` is an `enum` type with value -1. A generic matrix can be defined in EIGEN as

```
Matrix<T, rows, cols> M;
```

where `rows` and `cols` are integers (number of rows resp. columns) and `T` is the underlying type. If `rows` (resp. `cols`) is -1 (or `Dynamic`), the Matrix will have a variable number of rows (resp. columns), see ➜§ 1.2.11.

Remark: Matrix multiplication in EIGEN is not affected by aliasing issues, cf. Eigen documentation. Therefore you can safely write `A = A*A`).

Remark: For code validatioin the EIGEN implementation of Matrix power (`MatrixXcd::pow(unsigned int)`) can be used writing:

```
#include <unsupported/Eigen/MatrixFunctions>
```

SOLUTION for (1.6.a) → `1.6.1:powi.pdf` ▲

(1.6.b) ⚄ Find (in leading order and state using the Landau symbol O) the asymptotic complexity in k (and n) of your implementation of `matPow()` taking into account that in EIGEN a matrix-matrix multiplication requires a $O(n^3)$ computational effort.

SOLUTION for (1.6.b) → `1.6.2:powc.pdf` ▲

(1.6.c) ⚀ Compute the runtime of the EIGEN power function (`MatrixXxd::pow(int)`) and find out the complexity w.r.t k. Compare this with the function `matPow` from (1.6.a). For $n > 0$, use the $n \times n$ matrix defined by

$$(\mathbf{A}_{j,k}) := \frac{1}{\sqrt{n}} \exp\left(\frac{2\pi\imath\, jk}{n}\right)$$

to test the two functions ($\imath$ is the complex imaginary unit).

You should use the class `Timer` or the STL class `std::chrono` as explained in Problem 1.1, § 1.0.6.

Compute and output the runtime of the computation of the power for a matrix of size $N = 3$. The runtime should be the minimum runtime of 10 trial runs for $k = 2, \ldots, 2^{31}$. Output the time is seconds, using 3 decimal digits in scientific notation.

SOLUTION for (1.6.c) → `1.6.3:powr.pdf` ▲

End Problem 1.6

Problem 1.7: Structured matrix–vector product

In ➜Ex. 1.4.15 we saw how the particular structure of a matrix can be exploited to compute a matrix-vector product with substantially reduced computational effort. This problem presents a similar case.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Let $n \in \mathbb{N}, n > 0$ and consider the real $n \times n$ matrix $\mathbf{A}$ defined by

$$(\mathbf{A})_{i,j} = a_{i,j} = \min\{i,j\},\ i,j = 1,\ldots,n. \tag{1.0.24}$$

The matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{x}$ can be implemented in C++ as

C++11-code 1.0.25: Computing $\mathbf{A}\mathbf{x}$ for $\mathbf{A}$ from 1.0.24

```
2     VectorXd one = VectorXd::Ones(n);
3     VectorXd linsp = VectorXd::LinSpaced(n,1,n);
4     y = ( ( one * linsp.transpose() )
5           .cwiseMin( linsp * one.transpose()) ) * x;
```

Get it on GitLab (multAmin.cpp).

(1.7.a) ⚀ What is the asymptotic complexity (for $n \to \infty$) of the evaluation of the C++ code displayed above, with respect to the problem size parameter n?

SOLUTION for (1.7.a) → 1.7.1:strc.pdf ▲

(1.7.b) ⚄ Write an *efficient* C++ function

```
void multAmin(const VectorXd &x, VectorXd &y);
```

that computes the same multiplication as Code 1.0.29 but with a better asymptotic complexity with respect to n (you can use the template). Get it on GitLab (multAmin.cpp).

You can test your implementation by comparing the returned values with the ones obtained with Code 1.0.29.

HIDDEN HINT 1 for (1.7.b) → 1.7.2:smv:h1.pdf

SOLUTION for (1.7.b) → 1.7.2:stri.pdf ▲

(1.7.c) ⚀ What is the asymptotic complexity (in terms of problem size parameter n) of your function `multAmin`?

SOLUTION for (1.7.c) → 1.7.3:cmpk.pdf ▲

(1.7.d) ⚂ Compare the runtimes of your implementation and the implementation given in Code 1.0.29 for $n = 2^5, \ldots, 2^{12}$.

You can use the class `Timer` or the STL class `std::chrono` as explained in Problem 1.1, § 1.0.6.

Report the measurements in seconds with scientific notation using 3 decimal digits.

SOLUTION for (1.7.d) → 1.7.4:cmpk.pdf ▲

(1.7.e) ⚂ Consider the following C++ snippet:

C++11-code 1.0.29: Initializing $\mathbf{B}$

```
2   MatrixXd B = MatrixXd::Zero(n,n);
3   for(unsigned int i = 0; i < n; ++i) {
4     B(i,i) = 2;
5     if(i < n-1) B(i+1,i) = -1;
6     if(i > 0) B(i-1,i) = -1;
7   }
8   B(n-1,n-1) = 1;
```

Get it on GitLab (multAmin.cpp).

Sketch the matrix $\mathbf{B}$ created by these lines.

SOLUTION for (1.7.e) → `1.7.5:cmpk.pdf` ▲

(1.7.f) ⚄ With $\mathbf{A}$ from (1.0.24) and $\mathbf{B}$ from (1.7.e) write a short EIGEN-based C++ program that, for $n = 10$, computes $\mathbf{ABe}_j$, $\mathbf{e}_j$ the j-th unit vector in $\mathbb{R}^n$, $j = 1, \ldots, n$, and prints the result.

Formulate a conjecture based on you observations.

SOLUTION for (1.7.f) → `1.7.6:cmpkinv.pdf`

▲

End Problem 1.7

Problem 1.8: Avoiding cancellation

In ➜Section 1.5.4 we saw that the so-called *cancellation phenomenon* is a major cause of numerical instability, *cf.* ➜§ 1.5.43. Cancellation is the massive amplification of *relative errors* when subtracting two real numbers in floating point representation of about the same value.
Fortunately, expressions vulnerable to cancellation can often be recast in a mathematically equivalent form that is no longer affected by cancellation, see ➜§ 1.5.55. There, we studied several examples, and this problem gives some more.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

(1.8.a) ⊡ We consider the function

$$f_1(x_0,h) := \sin(x_0+h) - \sin(x_0)\ . \tag{1.0.30}$$

1. Derive an equivalent expression $f_2(x_0,h) = f_1(x_0,h)$, which no longer involves the difference of return values of trigonometric functions.
2. Suggest a formula that avoids cancellation errors for computing the approximation
$$f'(x) \approx \frac{f(x_0+h)-f(x_0)}{h}$$
of the derivative of $f(x) := \sin(x)$ at $x = x_0$.
3. Write a C++ program that implements your formula and computes an approximation of $f'(1.2)$, for $h = 1\cdot 10^{-20}, 1\cdot 10^{-19}, \ldots, 1$. Tabulate the relative error of the result using $\cos(1.2)$ as exact value. Plot the error of the approximation of $f'(x)$ at $x = 1.2$. Use the `mgl::Figure` class as explained in the lecture notes.

   For background information refer to Ex. 1.5.45.
4. Explain the observed behaviour of the error.

HIDDEN HINT 1 for (1.8.a) → `1.8.1:cnch1.pdf`

SOLUTION for (1.8.a) → `1.8.1:cancsin.pdf` ▲

(1.8.b) ⚄ Rewrite function $f(x) := \ln(x-\sqrt{x^2-1})$, $x > 1$, into a mathematically equivalent expression that is more suitable for numerical evaluation for any $x > 1$. Explain why, and provide a numerical example, which highlights the superiority of your new formula.

HIDDEN HINT 1 for (1.8.b) → `1.8.2:logcnch.pdf`

SOLUTION for (1.8.b) → `1.8.2:canclog.pdf` ▲

(1.8.c) ⊡ For the following expressions, state the numerical difficulties that might affect a straightforward implementation for certain values of x (which ones?), and rewrite the formulas in a way that is more suitable for numerical computation.

1. $\sqrt{x+\frac{1}{x}} - \sqrt{x-\frac{1}{x}}$, for $x > 1$.
2. $\sqrt{\frac{1}{a^2}+\frac{1}{b^2}}$, for $a, b > 0$.

SOLUTION for (1.8.c) → `1.8.3:cancsqr.pdf` ▲

End Problem 1.8

Problem 1.9: Complexity of a C++ function

In this problem we recall a concept from linear algebra: the diagonalization of a square matrix. Unless you can still remember what this means, please look up the chapter on "eigenvalues" in your linear algebra lecture notes. This problem also has a subtle relationship with Problem 1.6.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We consider the C++ function defined in `getit.cpp` (cf. Code 1.0.32). Get it on GitLab (getit.cpp).

C++11-code 1.0.32: Code for `getit`.

```
2  VectorXd getit(const MatrixXd &A, const VectorXd &x, unsigned int k) {
3
4      EigenSolver<MatrixXd> eig = EigenSolver<MatrixXd>(A);
5      const VectorXcd & V = eig.eigenvalues();
6      const MatrixXcd & W = eig.eigenvectors();
7
8      VectorXcd cx = x.cast<std::complex<double>>();
9
10     VectorXcd ret = W *
11         (
12           V.array().pow(k) *
13           (W.partialPivLu().solve(cx)).array()
14         ).matrix();
15
16     return ret.real();
17 }
```

Get it on GitLab (getit.cpp).

See the Eigen documentation for eigenvalue decompositions in Eigen.

The operator `array()` for $v \in \mathbb{R}^n$ transforms the vector to an EIGEN array, on which operations are performed componentwise. The function `matrix()` is the "inverse" of `array()`, transforming an array to a matrix, on which vector/matrix operations are performed.

The code `W.partialPivLU().solve(cx)` performs a LU decomposition and solves the system `W*x = cx` for `x`. See the Eigen documentation.

The `MatrixBase::cast<T>()` method applied to a matrix, will perform a componentwise cast of the matrix to a matrix of type `T`. See the Eigen documentation.

(1.9.a) ⚄ What is the output of `getit`, when A is a diagonalizable $n \times n$ matrix, $x \in \mathbb{R}^n$ and $k \in \mathbb{N}$?

SOLUTION for (1.9.a) → `1.9.1:eigpoww.pdf` ▲

(1.9.b) ⚀ Fix $k \in \mathbb{N}$. Discuss (in detail) the asymptotic complexity of `getit` for $n \to \infty$.

You may use the fact that computing eigenvalues and eigenvectors of an $n \times n$ matrix has asymptotic complexity $O(n^3)$ for $n \to \infty$.

SOLUTION for (1.9.b) → `1.9.2:eigpowc.pdf` ▲

End Problem 1.9

Problem 1.10: Approximating the Hyperbolic Sine

In this problem, we study how Taylor expansions can be used to avoid cancellation errors in the approximation of the hyperbolic sine, *cf.* the discussion in ➜Ex. 1.5.65.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Consider the C++ code given in Listing 1.0.33.

C++11-code 1.0.33: Implementation of sinh

```
2   auto sinh_unstable = [] (double x) {
3       double t = std::exp(x);
4       return .5 * (t - 1./t);
5   };
```

Get it on GitLab (sinh.cpp).

(1.10.a) ⚀ Explain why the function given in Listing 1.0.33 may not give a good approximation of the hyperbolic sine for small values of x, and compute the relative error

$$\epsilon_{rel} := \frac{|\texttt{sinh\_unstable}(x) - \sinh(x)|}{|\sinh(x)|}$$

with C++ for $x = 10^{-k}$, $k = 1, 2, \ldots, 10$. As "exact value" use the result of the C++ built-in function `std::sinh` (include `cmath`).

SOLUTION for (1.10.a) → `1.10.1:sinhex.pdf` ▲

(1.10.b) ⚀ Write the Taylor expansion with m terms around $x = 0$ of the function e^x. Specify the remainder, cf. ➜Ex. 1.5.65.

SOLUTION for (1.10.b) → `1.10.2:sinhwrt.pdf` ▲

(1.10.c) ⚁ Prove that, for every $x \geq 0$ the following inequality holds true:

$$\sinh x \geq x. \tag{1.0.35}$$

SOLUTION for (1.10.c) → `1.10.3:sinhleq.pdf` ▲

(1.10.d) ⚂ Based on the Taylor expansion, find an approximation for $\sinh(x)$, for every $0 \leq x \leq 10^{-3}$, so that the relative approximation error ϵ_{rel} is smaller than 10^{-15}. Follow the considerations of ➜Ex. 1.5.65.

SOLUTION for (1.10.d) → `1.10.4:sinhty.pdf` ▲

End Problem 1.10

Problem 1.11: Complex roots

This problem deals with complex numbers and cancellation errors.

Templates: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Given a complex number $w = u + iv$, $u, v \in \mathbb{R}$ with $v \geq 0$, its root $\sqrt{w} = x + iy$ can be defined by

$$x := \sqrt{(\sqrt{u^2 + v^2} + u)/2}, \tag{1.0.36}$$

$$y := \sqrt{(\sqrt{u^2 + v^2} - u)/2}. \tag{1.0.37}$$

Here, $\sqrt{-1} =: i$.

(1.11.a) ⊡ For what $w \in \mathbb{C}$ will the direct implementation of (1.0.36) and (1.0.37) be vulnerable to cancellation?

SOLUTION for (1.11.a) → `1.11.1:root1.pdf` ▲

(1.11.b) ⊡ Compute xy as an expression of u and v.

SOLUTION for (1.11.b) → `1.11.2:root2.pdf` ▲

(1.11.c) ⊡ Implement a function

```
std::complex<double> myroot( std::complex<double> w );
```

that computes the root of `w` as given by (1.0.36) and (1.0.37) without the risk of cancellation. You may use only real arithmetic: for instance, you may not apply the function `std::sqrt` with *complex* arguments.

Test your implementation with $w = 10^{20} + 5i$ and with $w = -5 + 10^{20}i$.

SOLUTION for (1.11.c) → `1.11.3:root3.pdf` ▲

End Problem 1.11

Problem 1.12: Symmetric Gauss-Seidel iteration

For a square matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, define $\mathbf{D_A}, \mathbf{L_A}, \mathbf{U_A} \in \mathbb{R}^{n,n}$ by:

$$(\mathbf{D_A})_{i,j} := \begin{cases} (\mathbf{A})_{i,j}, & i = j, \\ 0, & i \neq j \end{cases}, (\mathbf{L_A})_{i,j} := \begin{cases} (\mathbf{A})_{i,j}, & i > j, \\ 0, & i \leq j \end{cases}, (\mathbf{U_A})_{i,j} := \begin{cases} (\mathbf{A})_{i,j}, & i < j, \\ 0, & i \geq j \end{cases}. \quad (1.0.39)$$

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

The symmetric Gauss-Seidel iteration associated with the linear system of equations $\mathbf{Ax} = \mathbf{b}$ is defined as

$$\mathbf{x}^{(k+1)} = (\mathbf{U_A} + \mathbf{D_A})^{-1}\mathbf{b} - (\mathbf{U_A} + \mathbf{D_A})^{-1}\mathbf{L_A}(\mathbf{L_A} + \mathbf{D_A})^{-1}(\mathbf{b} - \mathbf{U_A}\mathbf{x}^{(k)}). \quad (1.0.40)$$

(1.12.a) Give a necessary and sufficient condition on $\mathbf{A}$, such that the iteration (1.0.40) is well-defined.

SOLUTION for (1.12.a) → `1.12.1:gsitw.pdf` ▲

(1.12.b) Assume that (1.0.40) is well-defined. Show that a fixed point $\mathbf{x}$ of the iteration (1.0.40) is a solution of the linear system of equations $\mathbf{Ax} = \mathbf{b}$.

SOLUTION for (1.12.b) → `1.12.2:sinhex.pdf` ▲

(1.12.c) Implement a C++ function

```
void GSIt(const MatrixXd & A, const VectorXd & b,
          VectorXd & x, double rtol);
```

solving the linear system $\mathbf{Ax} = \mathbf{b}$ using the iterative scheme (1.0.40). To that end, apply the iterative scheme to an initial guess $\mathbf{x}^{(0)}$ passed trough `x`. The approximated solution given by the final iterate is then stored in `x` as an output.

Use a correction based termination criterion with relative tolerance `rtol` based on the Euclidean vector norm.

SOLUTION for (1.12.c) → `1.12.3:gsitf.pdf` ▲

(1.12.d) Test your implementation ($n = 9, rtol = 10e-8$) with the linear system given by

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 0 & \dots & 0 \\ 2 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & 2 & 3 \end{bmatrix} \in \mathbb{R}^{n,n}, \mathbf{b} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n \quad (1.0.49)$$

output the l^2 norm of the residual of the approximated solution. Use $\mathbf{b}$ as your initial data.

SOLUTION for (1.12.d) → `1.12.4:gsitim.pdf` ▲

(1.12.e) Using the same matrix $\mathbf{A}$ and the same r.h.s. vector $\mathbf{b}$ as above (1.12.d), we have tabulated the quantity $\|\mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}\|_2$ for $k = 1, \dots, 20$.

k	$\|\mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}\|_2$	k	$\|\mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}\|_2$
1	0.172631	11	0.00341563
2	0.0623049	12	0.00234255
3	0.0464605	13	0.00160173
4	0.0360226	14	0.00109288
5	0.0272568	15	0.000744595
6	0.0200715	16	0.000506784
7	0.0144525	17	0.000344682
8	0.0102313	18	0.000234316
9	0.00715417	19	0.000159234
10	0.00495865	20	0.000108185

Describe qualitatively and quantitatively the convergence of the iterative scheme with respect to the number of iterations k.

SOLUTION for (1.12.e) → `1.12.5:gsitc.pdf` ▲

End Problem 1.12

Chapter 2

Direct Methods for Linear Systems of Equations

Problem 2.1: Resistance to impedance map

In ➜§ 2.6.13, we learned about the Sherman-Morrison-Woodbury update formula ➜Lemma 2.6.22, which allows the efficient solution of a linear system of equations after a low-rank update according to ➜Eq. (2.6.17), provided that the setup phase of an elimination (→ ➜§ 2.3.30) solver has already been done for the system matrix.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

In this problem, we examine the concrete application from ➜Ex. 2.6.25, where the update formula is key to efficient implementation. This application is the computation of the impedance of the circuit drawn in Fig. 5 as a function of a variable resistance of a *single* circuit element.
This circuit contain only identical linear resistors with the resistance R, that is, the relationship between branch currents and voltages is $I = RU$ throughout. Excitation is provided by a voltage V imposed at node 16.
Here we consider DC operation (stationary setting), that is, all currents and voltages are real-valued.

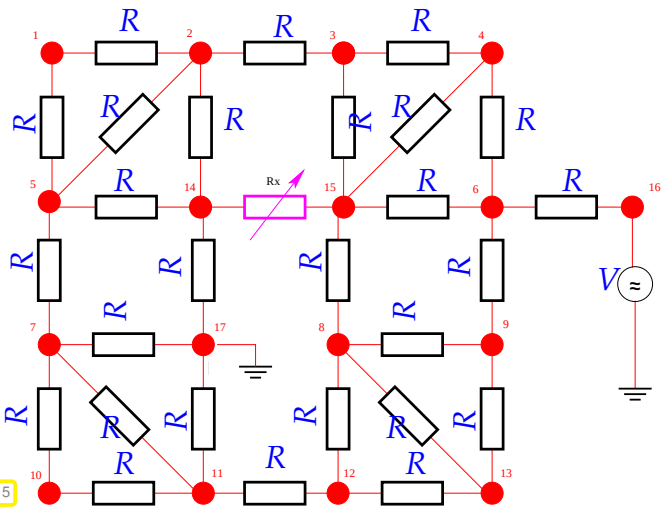


Fig. 5

(2.1.a) ⚀ Study ➜Ex. 2.1.3 that explains how to compute voltages and currents in a linear circuit by means of nodal analysis. Understand how this leads to a linear system of equations for the unknown nodal potentials. The fundamental laws of circuit analysis should be known from physics as well as the principles of nodal analysis. ▲

(2.1.b) ⚁ Use nodal analysis to derive the linear system of equations $\mathbf{A}_{R_x}\mathbf{x} = \mathbf{b}$ satisfied by the nodal potentials of the circuit from Figure 5. Here, $\mathbf{x}$ denotes the unknown voltages at the nodes. The voltage V is applied to node #16. Node #17 is grounded (set voltage to 0V). All resistors except for the controlled one (R_x, colored magenta) have the same resistance R. Use the numbering of nodes indicated in Figure 5.

Optionally, you can make the computer work for you and find a fast way to build a matrix providing only

the essential data. This is less tedious, less error prone and more flexible than specifying each entry individually. For this you can use auxiliary data structures.

SOLUTION for (2.1.b) → `2.1.2:circmat.pdf` ▲

(2.1.c) ⚄ Characterise the change in the circuit matrix $\mathbf{A}_{R_x}$ derived in sub-problem (2.1.b) induced by a change in the value of R_x as a low-rank modification of the circuit matrix $\mathbf{A}_0$. Use the matrix $\mathbf{A}_0$ (with $R_x = 0$) as your "base state".

HIDDEN HINT 1 for (2.1.c) → `2.1.3:circh.pdf`

SOLUTION for (2.1.c) → `2.1.3:circsmw.pdf` ▲

(2.1.d) ⚄ Based on the EIGEN library, implement a C++ class

C++11-code 2.0.2: `ImpedanceMap` Class signature

```
2  class ImpedanceMap {
3      std::size_t nnodes; //< Number of nodes in the circuit
4  public:
5      /* \brief Build system matrix and r.h.s. and perform a LU
          decomposition
6       * The LU decomposition is stored in 'lu' and can be
7       * reused in the SMW formula
8       * to avoid expensive matrix solves
9       * for repeated usages of the operator()
10      * \param R Resistance (in Ohm) value of R
11      * \param V Source voltage V at node 16 (in Volt),
12      *          ground is set to 0V at node 17
13      */
14     ImpedanceMap(double R, double V) : R(R), V(V) {
15         // TODO: build the matrix A_0.
16         // Compute lu factorization of A_0.
17         // Store LU factorization in 'lu'.
18         // Compute the right hand side and store it in 'b'.
19     };
20
21     /* \brief Compute the impedance given the resistance R_x.
22      * Use SMW formula for low rank perturbations to reuse LU
23      * factorization.
24      * \param Rx Resistence R_x > 0 between node 14 and 15
25      * \return Impedance V/I of the system A_{R_x}
26      */
27     double operator()(double Rx) {
28         // TODO: use SMW formula to compute the solution of A_{R_x} x = b
29         // Compute and return the impedance of the system.
30     }
31 private:
32     MatrixXd lu; //< Store LU decomp. of matrix A.
33     double R, V; //< Resistance R and source voltage W.
34     VectorXd b; //< R.h.s vector prescribing sink/source voltages.
35 };
```

Get it on GitLab (impedancemap.cpp).

whose `operator()` returns the impedance of the circuit from Figure 5, when supplied with a concrete value for R_x. This function should be implemented efficiently using ➜Lemma 2.6.22. The setup phase of Gaussian elimination should be carried out in the constructor.

Test your class using $R = 1, V = 1$ and $R_x = 1, 2, 4, \ldots, 1024$.

For $R_x = 1024$, we obtain an impedance of 2.65744.

See the file `impedancemap.cpp`.

The impedance of the circuit is the quotient of the voltage at the input node #16 and the current through the voltage source.

SOLUTION for (2.1.d) → `2.1.4:circimp.pdf` ▲

End Problem 2.1

Problem 2.2: Partitioned Matrix

Based on the block view of matrix multiplication presented in ➜§ 1.3.15, we looked a *block elimination* for the solution of block partitioned linear systems of equations in ➜§ 2.6.2. Also of interest are ➜Rem. 2.3.34 and ➜Rem. 2.3.32 where LU-factorisation is viewed from a block perspective. Closely related to this problem is ➜Ex. 2.6.5, which you should study again as warm-up to this problem.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Let the matrix $\mathbf{A} \in \mathbb{R}^{n+1,n+1}$ be partitioned according to

$$\mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}, \tag{2.0.5}$$

where $\mathbf{v} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^n$, and $\mathbf{R} \in \mathbb{R}^{n \times n}$ is *upper triangular* and *regular*.

(2.2.a) ⚀ Give a necessary and sufficient condition for the triangular matrix $\mathbf{R}$ to be invertible.

SOLUTION for (2.2.a) → 2.2.1:partinv.pdf ▲

(2.2.b) ⚁ Determine expressions for the sub-vectors $\mathbf{z} \in \mathbb{R}^n, \zeta \in \mathbb{R}$ of the solution vector of the linear system of equations

$$\begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \zeta \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \beta \end{bmatrix}$$

for arbitrary $\mathbf{b} \in \mathbb{R}^n, \beta \in \mathbb{R}$.

Use block-wise Gaussian elimination as presented in ➜§ 2.6.2.

SOLUTION for (2.2.b) → 2.2.2:partgauss.pdf ▲

(2.2.c) ⚀ Show that $\mathbf{A}$ is regular if and only if $\mathbf{u}^T\mathbf{R}^{-1}\mathbf{v} \neq 0$.

SOLUTION for (2.2.c) → 2.2.3:partareg.pdf ▲

(2.2.d) ⚂ Implement the C++ function

```
void solvelse(const MatrixXd & R,
              const VectorXd & v, const VectorXd & u,
              const VectorXd & b, VectorXd & x);
```

for the efficient computation of the solution of $\mathbf{Ax} = \mathbf{b}$ (with $\mathbf{A}$ as in (2.0.5)). Perform size check on input matrices and vectors.

Use the decomposition from (2.2.b).

You can rely on the `triangularView()` function to instruct EIGEN of the triangular structure of $\mathbf{R}$, see ➜Code 1.2.16.

SOLUTION for (2.2.d) → 2.2.4:partimpl.pdf ▲

(2.2.e) Test your implementation by comparing with a standard LU-solver provided by EIGEN.

Check the Eigen documentation.

SOLUTION for (2.2.e) → 2.2.5:parttest.pdf ▲

(2.2.f) What is the asymptotic complexity of your implementation of `solvelse()` in terms of problem size parameter $n \to \infty$?

SOLUTION for (2.2.f) → `2.2.6:partcmp.pdf` ▲

End Problem 2.2

Problem 2.3: Banded matrix

Banded matrices are an important class of structured matrices; see ➜Section 2.7.6 and Def. 2.7.55. We will study ways to exploit during computations the knowledge that only some (sub)diagonals of a matrix are nonzero.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

For $n \in \mathbb{N}$, consider the following matrix ($a_i, b_i \in \mathbb{R}$):

$$\mathbf{A} := \begin{bmatrix} 2 & a_1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 2 & a_2 & 0 & \dots & \dots & 0 \\ b_1 & 0 & \ddots & \ddots & \ddots & & \vdots \\ 0 & b_2 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & a_{n-1} \\ 0 & 0 & \dots & 0 & b_{n-2} & 0 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{2.0.8}$$

This matrix is an instance of a banded matrix.

(2.3.a) ⚀ Implement an efficient C++ function `multAx` for the computation of $\mathbf{y} = \mathbf{A}\mathbf{x}$:

```
template <class Vector>
void multAx(const Vector & a, const Vector & b,
            const Vector & x, Vector & y);
```

`Vector a` and `Vector b` contain the nonzero entries along the subdiagonals of matrix $\mathbf{A}$ (Eq. (2.0.8)). `Vector y` and `Vector x` are the vectors of equation $\mathbf{y} = \mathbf{A}\mathbf{x}$ (`Vector y` is the output).

SOLUTION for (2.3.a) → `2.3.1:effbandimpl.pdf` ▲

(2.3.b) ⚁ Show that $\mathbf{A}$ is invertible if $a_i, b_i \in [0,1]$.

HIDDEN HINT 1 for (2.3.b) → `2.3.2:effbandinvh.pdf`

SOLUTION for (2.3.b) → `2.3.2:effbandinv.pdf` ▲

(2.3.c) ⚂ Assume that $b_i = 0,\ \forall i = 1,\dots,n-2$. Implement an efficient C++ function solving $\mathbf{A}\mathbf{x} = \mathbf{r}$:

```
template <class Vector>
void solvelseAupper(const Vector & a, const Vector & r, Vector &
   x);
```

`Vector a` contains the nonzero entries along the upper subdiagonal of matrix $\mathbf{A}$ (Eq. (2.0.8)). `Vector r` and `Vector x` are the vectors of the equation $\mathbf{r} = \mathbf{A}\mathbf{x}$ (`Vector x` is the output).

HIDDEN HINT 1 for (2.3.c) → `2.3.3:effbandimplh.pdf`

SOLUTION for (2.3.c) → `2.3.3:effbandimplsol.pdf` ▲

(2.3.d) ⚄ For general $a_i, b_i \in [0,1]$, implement an efficient C++ function that computes the solution of $\mathbf{A}\mathbf{x} = \mathbf{r}$ by Gaussian elimination:

```
template <class Vector>
void solvelseA(const Vector & a, const Vector & b, const Vector &
   r, Vector & x);
```

You must not use any high-level solver routines of EIGEN.

HIDDEN HINT 1 for (2.3.d) → `2.3.4:effbandgaussh.pdf`

SOLUTION for (2.3.d) → `2.3.4:effbandhausssol.pdf` ▲

(2.3.e) ⚀ What is the asymptotic complexity of your implementation of `solvelseA` for $n \to \infty$?

HIDDEN HINT 1 for (2.3.e) → `2.3.5:effbandcompl.pdf`

SOLUTION for (2.3.e) → `2.3.5:effbandcompls.pdf` ▲

(2.3.f) ⚁ Implement `solvelseAEigen` as in (2.3.d), but this time using EIGEN's sparse elimination solver.

HIDDEN HINT 1 for (2.3.f) → `2.3.6:effbandsprs.pdf`

SOLUTION for (2.3.f) → `2.3.6:effbandsprssol.pdf` ▲

End Problem 2.3

Problem 2.4: Sequential linear systems

Consider a sequence of linear systems when all the linear systems share the same matrix $\mathbf{A}$: $\mathbf{A}\mathbf{x} = \mathbf{b}_i$. The computational cost of this problem may be reduced by performing an LU decomposition of $\mathbf{A}$ only once and reusing it for the different systems.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Consider the following pseudocode with input data $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$:

Algorithm 2.0.13: Code `solvepermb`
Input: Matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, vector $\mathbf{b} \in \mathbb{R}^n$
Output: Matrix $\mathbf{X} = [\mathbf{X}_1, \ldots, \mathbf{X}_n] \in \mathbb{R}^{n\times n}$, $\mathbf{X}_i \in \mathbb{R}^n$

```
while not all cyclic permutations of b tested yet do
    b ← [b_n, b_1, b_2, ..., b_{n-1}]^T
    X_i = A^{-1} b
end while
```

(2.4.a) ⚀ What is the worst case asymptotic complexity of the function `solvepermb` for $n \to \infty$?

HIDDEN HINT 1 for (2.4.a) → `2.4.1:permch.pdf`

SOLUTION for (2.4.a) → `2.4.1:permcs.pdf` ▲

(2.4.b) ⚁ Port the function `solvepermb` (Code 2.0.13) to C++ using EIGEN.

HIDDEN HINT 1 for (2.4.b) → `2.4.2:permph.pdf`

SOLUTION for (2.4.b) → `2.4.2:permps.pdf` ▲

(2.4.c) ⚂ Design an efficient C++ implementation of function `solvepermb` using EIGEN with asymptotic complexity $O(n^3)$.

HIDDEN HINT 1 for (2.4.c) → `2.4.3:permeh.pdf`

SOLUTION for (2.4.c) → `2.4.3:permes.pdf` ▲

End Problem 2.4

Problem 2.5: Rank-one perturbations

We consider another application of the Sherman-Morrison-Woodbury formula (see ➜Lemma 2.6.22), after Problem 2.1. Please carefully revise ➜§ 2.6.13 in the lecture notes.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Consider the following pseudocode:

Algorithm 2.0.16: Code `rankoneinvit`
Input: $\mathbf{d} \in \mathbb{R}^n, tol \in \mathbb{R}, tol > 0$
Output: l_{min}

```
ev ← d
l_min ← 0
l_new ← min|d|
while |l_new − l_min| > tol · l_min do
    l_min ← l_new
    M ← diag(d) + ev · ev^⊤
    ev ← M^{-1} ev
    ev ← ev/|ev|
    l_new ← ev^⊤ M ev
end while
l_min ← l_new
```

(2.5.a) ⚀ Port the function `rankoneinvit` (Code 2.0.16) to C++ using EIGEN.

The C++ code should perform exactly the same computations. In EIGEN the `asDiagonal()` method converts a vector into a diagonal matrix.

Do not expect to understand the purpose of this function.

SOLUTION for (2.5.a) → `2.5.1:smwps.pdf` ▲

(2.5.b) ⚁ What is the asymptotic complexity of the body of the loop in function `rankoneinvit`?

HIDDEN HINT 1 for (2.5.b) → `2.5.2:smwch.pdf`

SOLUTION for (2.5.b) → `2.5.2:smwcs.pdf` ▲

(2.5.c) ⚂ Design an efficient C++ implementation of the loop in function `rankoneinvit` using EIGEN, possibly with optimal asymptotic complexity. Compare the asymptotic complexity with the previous naive implementation in C++.

HIDDEN HINT 1 for (2.5.c) → `2.5.3:smweh.pdf`

SOLUTION for (2.5.c) → `2.5.3:smwes.pdf` ▲

(2.5.d) ⚁ What is the asymptotic complexity of the new version of the loop?

SOLUTION for (2.5.d) → `2.5.4:smwecs.pdf` ▲

(2.5.e) ⚁ Tabulate the runtimes of the two C++ implementations with different vector sizes $n = 2^p$, with $p = 2, 3, \ldots, 9$. As test vector use:

```
VectorXd::LinSpaced(n,1,2)
```

How can you show the different asymptotic complexity of the two implementations using these data?

HIDDEN HINT 1 for (2.5.e) → `2.5.5:smwcdh.pdf`

SOLUTION for (2.5.e) → `2.5.5:smwcds.pdf` ▲

End Problem 2.5

Problem 2.6: Lyapunov equation

Any linear system of equations with a finite number of unknowns can be written in the "canonical form" $\mathbf{Ax} = \mathbf{b}$ with a system matrix $\mathbf{A}$ and a right hand side vector $\mathbf{b}$. However, the linear system may be given in a different form and it may not be obvious how to extract the system matrix. We propose an intriguing example and also present an important *matrix equation*, the so-called Lyapunov equation.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Given $\mathbf{A} \in \mathbb{R}^{n\times n}$, consider the equation:

$$\mathbf{AX} + \mathbf{XA}^\top = \mathbf{I} \tag{2.0.19}$$

$\mathbf{X} \in \mathbb{R}^{n\times n}$ is unknown.

(2.6.a) ⚀ Show that for a fixed matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ the following mapping is linear:

$$L : \begin{cases} \mathbb{R}^{n,n} & \to & \mathbb{R}^{n,n} \\ \mathbf{X} & \mapsto & \mathbf{AX} + \mathbf{XA}^T \end{cases}$$

HIDDEN HINT 1 for (2.6.a) → `2.6.1:ly1h.pdf`

SOLUTION for (2.6.a) → `2.6.1:ly1s.pdf` ▲

In the following let $\mathrm{vec}(\mathbf{M}) \in \mathbb{R}^{n^2}$ denote the column vector obtained by storing the internal coefficient array of a matrix $\mathbf{M} \in \mathbb{R}^{n,n}$ in column major format, i.e. a data array with n^2 components (➜Rem. 1.2.23). In MATLAB, $\mathrm{vec}(\mathbf{M})$ would be the column vector obtained by `reshape(M,n*n,1)` or by `M(:)`. See ➜Rem. 1.2.27 for the implementation with EIGEN.

Equation (2.0.19) is equivalent to the following linear system of equations:

$$\mathbf{C}\,\mathrm{vec}(\mathbf{X}) = \mathbf{b} \tag{2.0.20}$$

The system matrix is $\mathbf{C} \in \mathbb{R}^{n^2,n^2}$ and the right-hand side vector $\mathbf{b} \in \mathbb{R}^{n^2}$.

(2.6.b) ⚀ Recall the notion of *sparse matrix*: see ➜Section 2.7 and, in particular, ➜Notion 2.7.1 and ➜Def. 2.7.3. ▲

(2.6.c) ⚁ Determine $\mathbf{C}$ and $\mathbf{b}$ from Eq. (2.0.20) for $n = 2$ and

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 3 \end{bmatrix}$$

HIDDEN HINT 1 for (2.6.c) → `2.6.3:ly3h.pdf`

SOLUTION for (2.6.c) → `2.6.3:ly3s.pdf` ▲

(2.6.d) ⚂ Use the Kronecker product to find a general expression for $\mathbf{C}$ in terms of $\mathbf{A}$.

SOLUTION for (2.6.d) → `2.6.4:ly4s.pdf` ▲

(2.6.e) ⚃ Implement a C++ function that builds the EIGEN matrix $\mathbf{C}$ from $\mathbf{A}$:

```
Eigen::SparseMatrix<double> buildC(const MatrixXd &A)
```

Make sure that the initialisation is done efficiently using an intermediate triplet format.

HIDDEN HINT 1 for (2.6.e) → 2.6.5:arrmatha.pdf

SOLUTION for (2.6.e) → 2.6.5:arrwc.pdf ▲

(2.6.f) ⚀ Implement a C++ function that returns the solution of Eq. (2.0.19), i.e. the $n \times n$-matrix $\mathbf{X}$ if $\mathbf{A} \in \mathbb{R}^{n,n}$:

```
void solveLyapunov(const MatrixXd & A, MatrixXd & X)
```

HIDDEN HINT 1 for (2.6.f) → 2.6.6:ly6h.pdf

SOLUTION for (2.6.f) → 2.6.6:ly6s.pdf ▲

(2.6.g) ⚁ Validate your C++ implementation of `buildC` and `solveLyapunov` for $n = 5$ and:

$$\mathbf{A} = \begin{bmatrix} 10 & 2 & 3 & 4 & 5 \\ 6 & 20 & 8 & 9 & 1 \\ 1 & 2 & 30 & 4 & 5 \\ 6 & 7 & 8 & 20 & 0 \\ 1 & 2 & 3 & 4 & 10 \end{bmatrix}$$

For reference, the 2-norm of the solution X is 0.102809.

SOLUTION for (2.6.g) → 2.6.7:ly7s.pdf ▲

(2.6.h) ⚂ Give an upper bound (as sharp as possible) for $\mathrm{nnz}(\mathbf{C})$ in terms of $\mathrm{nnz}(\mathbf{A})$ (nnz is the number of nonzero elements). Can $\mathbf{C}$ be legitimately regarded as a sparse matrix for large n even if $\mathbf{A}$ is dense?

HIDDEN HINT 1 for (2.6.h) → 2.6.8:ly8h.pdf

SOLUTION for (2.6.h) → 2.6.8:ly8s.pdf ▲

End Problem 2.6

Problem 2.7: Structured linear systems with pivoting

This problem deals with a block structured system and ways to efficiently implement the solution of such system. For this problem, you should have understood Gauss elimination with partial pivoting for the solution of a linear system, see ➜Section 2.3.3.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

We consider a block partitioned linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}\,, \quad \mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{C} \\ \mathbf{C} & \mathbf{D}_2 \end{bmatrix} \in \mathbb{R}^{2n,2n}\,,$$

where all the $n \times n$-matrices $\mathbf{D}_1$, $\mathbf{D}_2$ and $\mathbf{C}$ are *diagonal*. Hence, the matrix $\mathbf{A}$ can be described through three n-vectors $\mathbf{d}_1$, $\mathbf{c}$ and $\mathbf{d}_2$, which provide the diagonals of the matrix blocks. These vectors will be passed as arguments `d1`, `c`, and `d2` to the C++ codes below.

(2.7.a) ⚀ Which permutation of rows and columns converts the matrix into a tridiagonal matrix?

SOLUTION for (2.7.a) → `2.7.1:blockrnt.pdf` ▲

(2.7.b) ⚀ Write an efficient C++ function

```
VectorXd multA(const VectorXd & d1, const VectorXd & d2,
               const VectorXd & c, const VectorXd & x);
```

that returns $\mathbf{y} := \mathbf{A}\mathbf{x}$. The argument x passes a column vector $\mathbf{x} \in \mathbb{R}^{2n}$.

SOLUTION for (2.7.b) → `2.7.2:blockim.pdf` ▲

(2.7.c) ⚁

Compute the LU-factors of $\mathbf{A}$, where you may assume that they exist.

HIDDEN HINT 1 for (2.7.c) → `2.7.3:hblocklu.pdf`

SOLUTION for (2.7.c) → `2.7.3:blocklu.pdf` ▲

(2.7.d) ⚁ Write an efficient C++ function

```
VectorXd solveA(const VectorXd & d1, const VectorXd & d2,
                const VectorXd & c, const VectorXd & x);
```

that solves $\mathbf{A}\mathbf{x} = \mathbf{b}$ with Gaussian elimination.

Do not use EIGEN built-in linear solvers. Test for "near singularity" of the matrix.

Test your code with the arguments given in sub-problem Sub-problem (2.7.f). Compare with reference solution obtained in C++ using EIGEN linear solvers.

HIDDEN HINT 1 for (2.7.d) → `2.7.4:hsmalllse.pdf`

SOLUTION for (2.7.d) → `2.7.4:blockim.pdf` ▲

(2.7.e) ⚀ Analyse the asymptotic complexity of your implementation of `solveA` in term of the problem size parameter $n \to \infty$.

SOLUTION for (2.7.e) → `2.7.5:blockco.pdf` ▲

(2.7.f) ⊡ Determine the asymptotic complexity of `solveA` in a numerical experiment. As test case use $\mathbf{d}_1 = [1,\ldots,n]^\top = -\mathbf{d}_2$, $\mathbf{c} = \mathbf{1}$ and $b = [d1; d1]$. Tabulate the runtimes (in seconds, 3 decimal digit, scientific notation) for meaningful values of n.

SOLUTION for (2.7.f) → `2.7.6:blockrnt.pdf` ▲

End Problem 2.7

Problem 2.8: Structured linear systems

In this problem we come across the example of a structured matrix, for which a linear system can be solved very efficiently, though this is not obvious.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Consider the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, where the $n \times n$ matrix $\mathbf{A}$ has the following structure:

$$\mathbf{A} = \begin{bmatrix} a_1 & 0 & 0 & \dots & 0 \\ a_1 & a_2 & 0 & \ddots & \vdots \\ a_1 & a_2 & a_3 & \ddots & \\ \vdots & \vdots & & \ddots & 0 \\ a_1 & a_2 & a_3 & \dots & a_n \end{bmatrix}$$

(2.8.a) ⊡ Give necessary and sufficient conditions for the vector $\mathbf{a} = (a_1, \dots, a_n)^\top \in \mathbb{R}^n$ such that the matrix $\mathbf{A}$ is non-singular.

HIDDEN HINT 1 for (2.8.a) → 2.8.1:structuredLSE1h.pdf

SOLUTION for (2.8.a) → 2.8.1:structuredLSE1s.pdf ▲

(2.8.b) ⚀ Write a C++ function that builds the matrix $\mathbf{A}$ given the vector $\mathbf{a}$:

```
MatrixXd buildA(const VectorXd & a);
```

You can use EIGEN classes for your code.

HIDDEN HINT 1 for (2.8.b) → 2.8.2:structuredLSE2h.pdf

SOLUTION for (2.8.b) → 2.8.2:structuredLSE2s.pdf ▲

(2.8.c) ⊡ Implement a function in C++ which solves the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ by means of "structure oblivious" Gaussian elimination, for which EIGEN's dedicated classes and methods should be used.

```
void solveA(const VectorXd & a, const VectorXd & b,
  VectorXd & x);
```

The input is composed of vectors $\mathbf{a}$ and $\mathbf{b}$.

Run this function with $\mathbf{a}$ and $\mathbf{b}$ made of random elements and explore the cases $n = 2^k$, $k = 4, \dots, 12$. What is the asymptotic complexity in n of this naive implementation?

HIDDEN HINT 1 for (2.8.c) → 2.8.3:structuredLSE3h.pdf

SOLUTION for (2.8.c) → 2.8.3:structuredLSE3s.pdf ▲

(2.8.d) ⚀ Given a generic vector $\mathbf{a} \in \mathbb{R}^n$, compute symbolically (i.e. *by hand*) the inverse of matrix $\mathbf{A}$ and the solution $\mathbf{x}$ of $\mathbf{A}\mathbf{x} = \mathbf{b}$.

HIDDEN HINT 1 for (2.8.d) → 2.8.4:structuredLSE4h.pdf

SOLUTION for (2.8.d) → 2.8.4:structuredLSE4s.pdf ▲

(2.8.e) ⚀ Implement a function in C++ which efficiently solves the linear system $\mathbf{Ax} = \mathbf{b}$, using EIGEN classes. The input is consists of vectors $\mathbf{a}$ and $\mathbf{b}$.

HIDDEN HINT 1 for (2.8.e) → `2.8.5:structuredLSE5h.pdf`

SOLUTION for (2.8.e) → `2.8.5:structuredLSE5s.pdf` ▲

(2.8.f) ⚁ Compare the timing of the naive implementation of (2.8.c) with the efficient solution of (2.8.e).

HIDDEN HINT 1 for (2.8.f) → `2.8.6:structuredLSE6h.pdf`

SOLUTION for (2.8.f) → `2.8.6:structuredLSE6s.pdf` ▲

End Problem 2.8

Problem 2.9: Triplet format to CRS format

This exercise is about sparse matrices. Make sure you are prepared on the subject by reading ➜Section 2.7 in the lecture notes. In particular, read through the various *sparse storage formats* discussed in class (cf. ➜Section 2.7.1).
The ultimate goal is to devise a function that converts a matrix given in *triplet (or COOrdinate) list format* (COO, see ➜§ 2.7.6) to the *compressed row storage format* (CRS, see ➜Ex. 2.7.9). You do not have to follow the subproblems, if you devise a suitable conversion function and data structures on your own. You do not need to rely on EIGEN to solve this problem.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

The COO format stores a collection of triplets (i,j,v), with $i,j \in \mathbb{N},\ i,j \geq 0$ (the indices) and $v \in \mathbb{R}$ (the value in cell (i,j)). Multiple triplets corresponding to the same cell (i,j) are allowed, meaning that multiple values with the same indices (i,j) should be summed together when fully expressing the matrix.

The CRS format uses three vectors:

1. `val`, which stores the values of the nonzero cells, one row after the other.
2. `col_ind`, which stores the column indices of the nonzero cells, one row after the other.
3. `row_ptr`, which stores the index of the entry in `val`/`col_ind` containing the first element of each row.

The case of rows only made by zero elements require special consideration. The usual convention is that `row_ptr` stores two consecutive entries with the same values, meaning that in vectors `val` and `col_ind` the next row begins at the same index of the previous row (which implies that the previous row must be empty). This convention should be taken into account when e.g. iterating through `row_ptr`. However, to simplify things, we always consider *matrices without rows only made by zeros.*

(2.9.a) ⚀ Define a C++ structure that stores a matrix of type `scalar` (template parameter) in COO format:

```
template <class scalar>
struct TripletMatrix;
```

You should store sizes and indices as **std::size_t**.

HIDDEN HINT 1 for (2.9.a) → `2.9.1:triplettoCRS1h.pdf`

SOLUTION for (2.9.a) → `2.9.1:triplettoCRS1s.pdf` ▲

(2.9.b) ⚁ Define a C++ structure that stores a matrix of type `scalar` (template parameter) in CRS format:

```
template <class scalar>
struct CRSMatrix;
```

You can store sizes and indices as **std::size_t**.

HIDDEN HINT 1 for (2.9.b) → `2.9.2:triplettoCRS2h.pdf`

SOLUTION for (2.9.b) → `2.9.2:triplettoCRS2s.pdf` ▲

(2.9.c) ⚂ *This subproblem is optional.* Implement C++ member functions for `TripletMatrix` ((2.9.a)) and `CRSMatrix` ((2.9.b)) that convert your structures to EIGEN dense matrices types:

```
Eigen::Matrix<scalar, Eigen::Dynamic, Eigen::Dynamic>
   TripletMatrix<scalar>::densify();
Eigen::Matrix<scalar, Eigen::Dynamic, Eigen::Dynamic>
   CRSMatrix<scalar>::densify();
```

HIDDEN HINT 1 for (2.9.c) → 2.9.3:triplettoCRS3h.pdf

SOLUTION for (2.9.c) → 2.9.3:triplettoCRS3s.pdf ▲

(2.9.d) ⚄ Write a C++ function that converts a matrix **T** in COO format to a matrix **C** in CRS format:

```
template <class scalar>
void tripletToCRS(const TripletMatrix<scalar>& T,
   CRSMatrix<scalar>& C);
```

Try to be as efficient as possible.

HIDDEN HINT 1 for (2.9.d) → 2.9.4:triplettoCRS4h.pdf

SOLUTION for (2.9.d) → 2.9.4:triplettoCRS4s.pdf ▲

(2.9.e) ⚄ What is the worst-case complexity of your function `tripletToCRS` ((2.9.d)) in the number of triplets?

HIDDEN HINT 1 for (2.9.e) → 2.9.5:triplettoCRS5h.pdf

SOLUTION for (2.9.e) → 2.9.5:triplettoCRS5s.pdf ▲

(2.9.f) ⚀ Test the correctness and runtime of your function `tripletToCRS`.

HIDDEN HINT 1 for (2.9.f) → 2.9.6:triplettoCRS6h.pdf

SOLUTION for (2.9.f) → 2.9.6:triplettoCRS6s.pdf ▲

End Problem 2.9

Problem 2.10: Sparse matrices in CCS format

Internally, sparse matrices have to be encoded in special formats. It is essential to be familiar with them when using one of the many routines for handling sparse matrices.
In ➜Ex. 2.7.9 the so-called *CRS format* (Compressed Row Storage) was introduced. It uses three arrays (`val`, `col_ind` and `row_ptr`) to store a sparse matrix.
EIGEN can also handle the *CCS format* (Compressed Column Storage), which is like CRS for the transposed matrix. It relies on the arrays `val`, `row_ind` and `col_ptr`.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Similarly to Problem 2.9, you can assume that the considered matrices do not have columns with all elements equal to 0. (Otherwise, it may become problematic to update `col_ptr`.)

(2.10.a) ⚄ Implement a C++ function which returns the given square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ in CCS format:

```
void CCS(const MatrixXd & A, VectorXd & val,
         VectorXd & row_ind, VectorXd & col_ptr);
```

While you can use EIGEN classes such as `MatrixXd`, do not use EIGEN methods to directly access `val`, `row_ind` and `col_ptr`. In other words, you must not use EIGEN's class `Eigen::SparseMatrix` and simply run `makeCompressed()`.

In this problem you may test for exact equality with 0.0, because zero matrix entries are not supposed to be results of floating point computations.

HIDDEN HINT 1 for (2.10.a) → `2.10.1:smwch.pdf`

SOLUTION for (2.10.a) → `2.10.1:smwps.pdf` ▲

(2.10.b) ⚄ What is the computational complexity of your `CCS` function:

- with respect to the matrix size n, where $\mathbf{A} \in \mathbb{R}^{n\times n}$?
- with respect to nnz, which denotes the number of nonzero elements in $\mathbf{A}$?

SOLUTION for (2.10.b) → `2.10.2:smwps.pdf` ▲

End Problem 2.10

Problem 2.11: Ellpack sparse matrix format

The number of processing cores of high-performance computers has seen an exponential growth. However, the increase in *memory bandwidth*, which is the rate at which data can be read from or written into memory by a processor, has been slower. Hence, memory bandwidth is a bottleneck for several algorithms.
Several papers present storage formats to improve the performance of sparse matrix-vector multiplications. One example is the *Ellpack* format, where the number of nonzero entries per row is bounded and shorter rows are padded with zeros.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Consider the template file `ellpack.cpp`. The following `EllpackMat` class implements the Ellpack sparse matrix format for a generic matrix $\mathbf{A} \in \mathbb{R}^{m,n}$:

C++11-code 2.0.42: Declaration of `EllpackMat` class

```
2  class EllpackMat {
3  public:
4      EllpackMat(const Triplets & triplets, index_t m, index_t n);
5
6      double operator() (index_t i, index_t j) const;
7
8      void mvmult(const VectorXd &x, VectorXd &y) const;
9
10     void mtvmult(const VectorXd &x, VectorXd &y) const;
11
12 private:
13     std::vector<double>  val; //< Vector containing values
14     // corresponding to entries in 'col'
15     std::vector<index_t> col; //< Vector containing column
16     // indices of the entries in 'val'.
17     // The position of a cell in 'val' and 'col'
18     // is determined by its row number and original position in
           'triplets'
19
20     index_t maxcols; //< Number of non-empty columns
21     index_t m,n; //< Number of rows, number of columns
22 };
```

Get it on GitLab (ellpack.cpp).

In the code above, `maxcols` is defined as:

$$\texttt{maxcols} := \max\{\#\big[(i,j) \mid \mathbf{A}_{i,j} \neq 0,\ j = 1,\ldots,n\big],\ i = 1,\ldots,m\}$$

The **operator** `()` method is implemented as follows:

C++11-code 2.0.43: Declaration of operator() method

```
2 void EllpackMat::mvmult(const VectorXd &x, VectorXd &y) const {
3     assert(x.size() == n && "Incompatible vector x size!");
4     assert(y.size() == m && "Incompatible vector y size!");
5
6     // TODO: implement operation y = Ax
7 }
```

Get it on GitLab (ellpack.cpp).

It is well defined for $i = 0,\ldots,m-1$ and for $j = 0,\ldots,n-1$.

(2.11.a) ⚀ Implement an efficient constructor that builds an object of type `EllpackMat` from data forming a matrix in triplet format. In other words, implement the constructor with the following signature:

```
EllpackMat(const Triplets & triplets, index_t m, index_t
    n);
```

`triplets` is a **std::vector** of EIGEN triplets (see ➜Section 2.7.3). The arguments m and n represent the number of rows and columns of the matrix $\mathbf{A}$.

The data in the triplet vector must be compatible with the matrix size provided to the constructor. No assumption is made on the ordering of the triplets. Values belonging to multiple occurrences of the same index pair are to be summed up. However, in this subproblem you may assume that duplicate index pairs do not occur in the triplets vector.

HIDDEN HINT 1 for (2.11.a) → `2.11.1:ellpack1h.pdf`

SOLUTION for (2.11.a) → `2.11.1:ellpack1s.pdf` ▲

(2.11.b) ⚁ Implement an efficient method of class `EllpackMat` that, given an input n-vector $\mathbf{x}$, returns the m-vector $\mathbf{y}$ from the matrix-vector product $\mathbf{Ax}$:

```
void mvmult(const Vector &x, Vector &y) const;
```

$\mathbf{A}$ is the matrix represented by ⋆**this**. The implementation must run with the optimal complexity of $O(\mathrm{nnz}(\mathbf{A}))$.

HIDDEN HINT 1 for (2.11.b) → `2.11.2:ellpack2h.pdf`

SOLUTION for (2.11.b) → `2.11.2:ellpack2s.pdf` ▲

(2.11.c) ⚂ Similarly to (2.11.b), implement now the method `mtvmult` that computes the matrix-vector product $\mathbf{A}^\top\mathbf{x}$.

HIDDEN HINT 1 for (2.11.c) → `2.11.3:ellpack3h.pdf`

SOLUTION for (2.11.c) → `2.11.3:ellpack3s.pdf` ▲

(2.11.d) ⚂ You can test your solution of (2.11.a), (2.11.b) and (2.11.c) using the code in `main()`. A 3×6 matrix $\mathbf{A}$ of type `Eigen::SparseMatrix<double>` is built from the following triplets:

$$\{(1,2,4),(0,0,5),(1,2,6),(2,5,7),(0,4,8),(1,3,9),(2,2,10),(2,1,11),(1,0,12)\}$$

An equivalent matrix $\mathbf{E}$ of type `EllpackMat` is also built. We then use vectors $\mathbf{x}_1 = [4,5,6,7,8,9]^\top$, $\mathbf{x}_2 = [1,2,3]^\top$ and compute products $\mathbf{Ax}_1$, $\mathbf{Ex}_1$ and $\mathbf{A}^\top\mathbf{x}_2$, $\mathbf{E}^\top\mathbf{x}_2$.

Finally, we return the l^2-norm of the differences.

SOLUTION for (2.11.d) → `2.11.4:ellpack4s.pdf` ▲

End Problem 2.11

Problem 2.12: Grid functions ⊡

This exercise deals with construction of sparse matrices from triplet format, see ➜Section 2.7.3. This task is relevant for applications like image processing and the numerical solution of partial differential equations.

[This problem involves implementation in C++]

Consider the following matrix $\mathbf{S} \in \mathbb{R}^{3,3}$:

$$\mathbf{S} := \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Denote by $s_{i,j}$ the entries of $\mathbf{S}$. Consider the following *linear* operator from $\mathbb{R}^{n,m}$ to $\mathbb{R}^{n,m}$:

$$L : \mathbb{R}^{n,m} \to \mathbb{R}^{n,m},$$
$$x_{i,j} \mapsto \begin{cases} \sum_{k,l=1}^{3} s_{k,l} x_{i+k-2,j+l-2} & \text{if well-defined} \\ x_{i,j} & \text{otherwise} \end{cases}$$

(2.12.a) ⊡ Show that L is a linear operator.

SOLUTION for (2.12.a) → `2.12.1:grfl.pdf` ▲

(2.12.b) ⊡ The space $\mathbb{R}^{n,m}$, as a vector-space, is isomorphic to $\mathbb{R}^{n\cdot m}$, via the mapping $\mathbf{X} \mapsto \mathrm{vec}(\mathbf{X})$, see ➜Eq. (1.2.24).

From linear algebra, we know that any linear operator over a finite dimensional (real) vector space can be represented by a (real) matrix multiplication. We define the matrix $\mathbf{A} \in \mathbb{R}^{nm,nm}$ s.t.

$$\mathbf{A} \cdot \mathrm{vec}(\mathbf{X}) = \mathrm{vec}(L(\mathbf{X})).$$

Write down the matrix $\mathbf{A}$ for $n = m = 3$.

SOLUTION for (2.12.b) → `2.12.2:gfrwa.pdf` ▲

(2.12.c) ⊡ Write a function:

```
void eval(MatrixXd & X, std::function<double(index_t,index_t)> f);
```

which, given a matrix $\mathbf{X} \in \mathbb{R}^{n,m}$ and a function $f : \mathbb{N}^2 \to \mathbb{R}$, fills the matrix $\mathbf{X}$ s.t. $(\mathbf{X})_{i,j} = f(i,j)$.

Here `index_t` denotes an integer type for the indices.

Additionally, in `main()`, define a lambda function f, with signature `double(index_t,index_t)`, implementing:

$$f : \mathbb{N}^2 \to \mathbb{R},$$
$$(i,j) \mapsto \begin{cases} 1 & i > n/4 \wedge i < 3n/4 \wedge j > m/4 \wedge j < 3m/4 \\ 0 & \text{otherwise} \end{cases}$$

SOLUTION for (2.12.c) → `2.12.3:gfrwa.pdf` ▲

(2.12.d) ⚂ Implement a function

```
SparseMatrix<double> build_matrix(const Matrix3d & S,
                                  const shape_t & size);
```

which, given the stencil matrix $\mathbf{S} \in \mathbb{R}^{3,3}$ and the size of the matrix $\mathbf{X}$, builds and returns the sparse matrix $\mathbf{A}$ in CRS format. The matrix $\mathbf{A}$ is built using intermediate triplet format.

The type `shape_t` is a type (equivalent to a tuple) representing the size $(rows, cols)$ of the a matrix.

To help you in the endeavour, you can optionally implement a function

```
inline index_t to_vector_index(index_t i,
                               index_t j,
                               const shape_t & size);
```

which, given an index pair $(i,j) \in \mathbb{N}^2$ returns an index $I \in \mathbb{N}$ s.t. $(\mathbf{X})_{i,j} = (\mathrm{vec}(\mathbf{X}))_I$. The variable `size` passes the size of $\mathbf{X}$.

SOLUTION for (2.12.d) → `2.12.4:gfrwb.pdf` ▲

(2.12.e) ⚂ Implement a function

```
void mult(const SparseMatrix<double> & A,
          const MatrixXd & X, MatrixXd & Y);
```

which, given a matrix $\mathbf{X}$ and the sparse matrix $\mathbf{A}$, returns the matrix $\mathbf{Y}$ s.t. $\mathrm{vec}(\mathbf{Y}) := \mathbf{A}\,\mathrm{vec}(\mathbf{X})$. You can use objects of type `Eigen::Map` to "reshape" a matrix into a column vector.

SOLUTION for (2.12.e) → `2.12.5:gfrwm.pdf` ▲

(2.12.f) ⚄

Implement a function

```
void solve(const SparseMatrix<double> & A,
           const MatrixXd & Y, MatrixXd & X);
```

which, given a matrix $\mathbf{Y}$ and the sparse matrix $\mathbf{A}$, returns the matrix $\mathbf{X}$ s.t. $\mathrm{vec}(\mathbf{Y}) := \mathbf{A}\,\mathrm{vec}(\mathbf{X})$. You can use objects of type `Eigen::Map` to "reshape" a matrix into a column vector. Object of this type are read and write compatible.

SOLUTION for (2.12.f) → `2.12.6:gfrwi.pdf` ▲

End Problem 2.12

Problem 2.13: Efficient sparse matrix-matrix multiplication in COO format

The *triplet (or COOrdinate) list format* works very well for defining a matrix or adding/modifying its elements. This is however not so true for matrix operations such as matrix-matrix multiplications, unlike the CSC/CSR storage.
This problem is about (sparse) matrix-matrix multiplication in COO format. We will consider the following items:

1. The worst case of sparse matrix-matrix multiplication, when one wants to use sparse matrix storage formats.
2. The most efficient way to tackle this problem with the COO format.
3. The asymptotic complexity of such algorithm (which is, by definition, the complexity under the worst-case scenario).

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

For simplicity, in the following we will only deal with *binary* matrices. Binary matrices are only made of 0 or 1. At the same time, we will allow for duplicates in our implementations of the triplet format (see ➜§ 2.7.6). The matrix entry associated to the pair (i,j) is defined to be the sum of all triplets corresponding to it.

For the subproblems involving coding, we define types `Trip = Triplet<double>` and `TripVec = std::vector<trip>`.

(2.13.a) ⚀ Consider multiplications between sparse matrices: $\mathbf{AB} = \mathbf{C}$. Is the product $\mathbf{C}$ guaranteed to be sparse? Make an example of two sparse matrices which, when multiplied, return a dense matrix.

HIDDEN HINT 1 for (2.13.a) → `2.13.1:matmatCOO1h.pdf`

SOLUTION for (2.13.a) → `2.13.1:matmatCOO1s.pdf` ▲

(2.13.b) ⚁ Implement a C++ function which returns the input matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ in COO format:

```
TripVec Mat2COO(const MatrixXd &A);
```

You can use EIGEN classes.

SOLUTION for (2.13.b) → `2.13.2:matmatCOO2s.pdf` ▲

(2.13.c) ⚂ Implement a C++ function which computes the product between two matrices in COO format:

```
TripVec COOprod_naive(const TripVec &A, const TripVec &B);
```

You can use EIGEN classes.

HIDDEN HINT 1 for (2.13.c) → `2.13.3:matmatCOO3h.pdf`

SOLUTION for (2.13.c) → `2.13.3:matmatCOO3s.pdf` ▲

(2.13.d) ⚀ What is the asymptotic complexity of your naive implementation in (2.13.c)?

HIDDEN HINT 1 for (2.13.d) → `2.13.4:matmatCOO4h.pdf`

SOLUTION for (2.13.d) → `2.13.4:matmatCOO4s.pdf` ▲

(2.13.e) ⚃ Implement a C++ function which computes the product between two matrices in COO format in an efficient way:

```
TripVec COOprod_effic(TripVec &A, TripVec &B);
```

You can use EIGEN classes.

Can you do better than (2.13.c)?

HIDDEN HINT 1 for (2.13.e) → `2.13.5:matmatCOO5h.pdf`

SOLUTION for (2.13.e) → `2.13.5:matmatCOO5s.pdf` ▲

(2.13.f) ⚂ What is the asymptotic complexity of your efficient implementation in (2.13.e)?

HIDDEN HINT 1 for (2.13.f) → `2.13.6:matmatCOO6h.pdf`

SOLUTION for (2.13.f) → `2.13.6:matmatCOO6s.pdf` ▲

(2.13.g) ⊡ Compare the timing of `COOprod_naive` (2.13.c) and `COOprod_effic` (2.13.e) for random matrices with different dimensions n. Perform the comparison twice: first only for products between sparse matrices, then for any kind of matrix.

HIDDEN HINT 1 for (2.13.g) → `2.13.7:matmatCOO7h.pdf`

SOLUTION for (2.13.g) → `2.13.7:matmatCOO7s.pdf` ▲

End Problem 2.13

Chapter 3

Direct Methods for Linear Least Squares Problems

Problem 3.1: Matrix least squares in Frobenius norm

In this problem we look at a particular *constrained* linear least squares problem. In particular, we will study the augmented normal equation approach to solve such type of least squares problem, see ➜Section 3.6.1. In this context we refresh our understanding of the Lagrange multiplier technique.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Consider the following problem:

$$\text{given } \mathbf{z} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^n, \quad \text{find} \quad \mathbf{M}^* = \operatorname*{argmin}_{\mathbf{M}\in\mathbb{R}^{n,n},\, \mathbf{Mz}=\mathbf{g}} \|\mathbf{M}\|_F \,, \tag{3.0.1}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix as introduced in ➜Def. 3.4.46.

(3.1.a) ⚀ Reformulate the problem as an equivalent standard linearly constrained least squares problem

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^N,\, \mathbf{Cx}=\mathbf{d}} \|\mathbf{Ax}-\mathbf{b}\|_2 \,,$$

for suitable matrices $\mathbf{A}$, $\mathbf{C}$ and vectors $\mathbf{b}$ and $\mathbf{d}$, see ➜Eq. (3.6.1). These matrices and vectors have to be specified based on $\mathbf{z}$ and $\mathbf{g}$.

SOLUTION for (3.1.a) → `3.1.1:lsqfrobe.pdf` ▲

(3.1.b) ⚁ [depends on Sub-problem (3.1.a)]

State a necessary and sufficient condition for the matrix $\mathbf{C}$ found in Sub-problem (3.1.a) to possess full rank.

SOLUTION for (3.1.b) → `3.1.2:lsqf1.pdf` ▲

(3.1.c) ⚂ [depends on Sub-problem (3.1.a)]

State the augmented normal equations corresponding to the constrained linear least squares problem found in Sub-problem (3.1.a) and give necessary and sufficient conditions on $\mathbf{g}$ and $\mathbf{z}$ that ensure existence and uniqueness of solutions.

HIDDEN HINT 1 for (3.1.c) → `3.1.3:lsqfh1.pdf`

SOLUTION for (3.1.c) → `3.1.3:.pdf` ▲

(3.1.d) ⚄ [depends on Sub-problem (3.1.c)]

Write a C++ function

```
MatrixXd min_frob(const VectorXd & Z, const VectorXd & g);
```

that computes the solution $\mathbf{M}^*$ of the minimization problem Eq. (3.0.1) given the vectors $\mathbf{z}, \mathbf{g} \in \mathbb{R}^n$. Use the augmented normal equations ➜Eq. (3.6.7) derived in Sub-problem (3.1.c).

You may use

```
#include <unsupported/Eigen/KroneckerProduct>
```

SOLUTION for (3.1.d) → 3.1.4:lsqfrobi.pdf

▲

(3.1.e) ⚄ Using random vectors $\mathbf{z}$ and $\mathbf{g}$, check *in a numerical experiment* that

$$\mathbf{M} = \frac{\mathbf{g}\mathbf{z}^\top}{\|\mathbf{z}\|_2^2} \tag{3.0.5}$$

is a solution to Eq. (3.0.1). To that end write a short code that calls the function `min_frob()` implemented in Sub-problem (3.1.d), conducts the test and prints the result to `stdout`.

Run your code and report the output.

SOLUTION for (3.1.e) → 3.1.5:lsqfrobc.pdf

▲

(3.1.f) ⚄ [depends on Sub-problem (3.1.c)]

Now, give a rigorous proof that Eq. (3.0.5) gives a solution of Eq. (3.0.1), provided that $\mathbf{z} \neq \mathbf{0}$.

SOLUTION for (3.1.f) → 3.1.6:lsqfrx.pdf ▲

So far, we have simply applied the formulas from ➜Section 3.6.1 to Eq. (3.0.1) and its reformulation as a constrained linear least squares problem. Now we take a closer look at the method of Lagrangian multipliers, which was used to derive these equations in class.

(3.1.g) ⚄ As in ➜Eq. (3.6.3) and ➜Eq. (3.6.4) from the lecture notes, find a Lagrangian functional $L : \mathbb{R}^{n,n} \times \mathbb{R}^n \longrightarrow \mathbb{R}$ such that:

$$\mathbf{M}^* = \operatorname*{argmin}_{\mathbf{M}\in\mathbb{R}^{n,n}} \left\{ \max_{\mathbf{m}\in\mathbb{R}^n} [L(\mathbf{M},\mathbf{m})] \right\}. \tag{3.0.8}$$

The matrix $\mathbf{M}^*$ should provide the solution of (3.0.20).

HIDDEN HINT 1 for (3.1.g) → 3.1.7:LSQFrobLagr1h.pdf

SOLUTION for (3.1.g) → 3.1.7:LSQFrobLagr1s.pdf ▲

(3.1.h) ⚄ Find the derivative $\mathbf{grad}\,\Phi \in \mathbb{R}^{n,n}$ of the function Φ defined as:

$$\Phi : \mathbb{R}^{n,n} \to \mathbb{R}, \quad \Phi(\mathbf{X}) = \|\mathbf{X}\|_F^2 \tag{3.0.9}$$

HIDDEN HINT 1 for (3.1.h) → 3.1.8:LSQFrobLagr2h.pdf

SOLUTION for (3.1.h) → 3.1.8:LSQFrobLagr2s.pdf ▲

(3.1.i) ⚀ Use the result of Sub-problem (3.1.h) to derive saddle point equations for $\mathbf{grad}\, L(\mathbf{M},\mathbf{m}) = 0$, for the L obtained in Sub-problem (3.1.g).

HIDDEN HINT 1 for (3.1.i) → `3.1.9:LSQFrobLagr3h.pdf`

SOLUTION for (3.1.i) → `3.1.9:LSQFrobLagr3s.pdf` ▲

(3.1.j) ⚀ Solve the saddle point equations obtained in (3.1.i) in symbolic form.

HIDDEN HINT 1 for (3.1.j) → `3.1.10:LSQFrobLagr4h.pdf`

SOLUTION for (3.1.j) → `3.1.10:LSQFrobLagr4s.pdf` ▲

End Problem 3.1

Problem 3.2: Sparse Approximate Inverse (SPAI)

This problem studies the least squares aspects of the SPAI method, a technique used in the numerical solution of partial differential equations. We encounter an "exotic" sparse matrix technique and rather non-standard least squares problems. Please note that the matrices to which SPAI techniques are applied will usually be huge and extremely sparse, say, of dimension $10^7 \times 10^7$ with only 10^8 non-zero entries. Therefore sparse matrix techniques must be applied.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Let $\mathbf{A} \in \mathbb{R}^{N,N}$, $N \in \mathbb{N}$, be a regular sparse matrix with at most $n \ll N$ non-zero entries per row and column. We define the space of matrices with the same pattern as $\mathbf{A}$:

$$\mathcal{P}(\mathbf{A}) := \{\mathbf{X} \in \mathbb{R}^{N,N} : \ (\mathbf{A})_{ij} = 0 \ \Rightarrow \ (\mathbf{X})_{ij} = 0\} \ . \tag{3.0.14}$$

The "primitive" SPAI (sparse approximate inverse) $\mathbf{B}$ of $\mathbf{A}$ is defined as

$$\mathbf{B} := \operatorname*{argmin}_{\mathbf{X} \in \mathcal{P}(\mathbf{A})} \|\mathbf{I} - \mathbf{A}\mathbf{X}\|_F \ , \tag{3.0.15}$$

where $\|\cdot\|_F$ stands for the Frobenius norm. The solution of (3.0.14) can be used as a so-called preconditioner for the acceleration of iterative methods for the solution of linear systems of equations, see ➜Chapter 10.

An extended "self-learning" variant of the SPAI method is presented in

> M. J. GROTE AND T. HUCKLE, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

(3.2.a) ⚀ Show that the columns of $\mathbf{B}$ can be computed independently of each other by solving linear least squares problems. In the statement of these linear least squares problems write $\mathbf{b}_i$ for the columns of $\mathbf{B}$.

HIDDEN HINT 1 for (3.2.a) → `3.2.1:spaih1.pdf`

SOLUTION for (3.2.a) → `3.2.1:spai1.pdf` ▲

(3.2.b) ⚄ Implement an efficient C++ function

```
SparseMatrix<double> spai(SparseMatrix<double> & A);
```

for the computation of $\mathbf{B}$ according to (3.0.15). You may rely on the normal equations associated with the linear least squares problems for the columns of $\mathbf{B}$ or you may simply invoke the least squares solver of EIGEN.

HIDDEN HINT 1 for (3.2.b) → `3.2.2:spai2h.pdf`

SOLUTION for (3.2.b) → `3.2.2:spai2s.pdf` ▲

(3.2.c) ⚂ What is the total asymptotic computational effort of `spai` in terms of the problem size parameters N and n.

SOLUTION for (3.2.c) → `3.2.3:SPAI3.pdf`

▲

End Problem 3.2

Problem 3.3: Constrained least squares and Lagrange multipliers

In ➜Section 3.6.1 we saw how to solve a constrained least squares problem via Lagrange multipliers. In this problem, we will think about the ideas behind the Lagrange multipliers and derivate the equations solving such problems.

Template: Get it on GitLab.
Solution: Get it on GitLab.

Consider the following linearly constrained quadratic minimization problem that was already considered in Problem 3.1:

$$\text{Given} \quad \mathbf{z} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^n, \quad \text{find} \quad \mathbf{M}^* = \operatorname*{argmin}_{\mathbf{M}\in\mathbb{R}^{n,n},\, \mathbf{Mz}=\mathbf{g}} \{\|\mathbf{M}\|_F\}. \tag{3.0.20}$$

Here, $\|\cdot\|_F$ denotes the *Frobenius norm* of a matrix (see ➜Def. 3.4.46).

We tackle (3.0.20) using Lagrange multipliers (see ➜Section 3.6.1) to take into account the n linear constraints imposed by $\mathbf{Mz} = \mathbf{g}$.

(3.3.a) ⚀ As in ➜Eq. (3.6.3) and ➜Eq. (3.6.4) from the lecture notes, find a Lagrangian functional $L : \mathbb{R}^{n,n} \times \mathbb{R}^n \longrightarrow \mathbb{R}$ such that:

$$\mathbf{M}^* = \operatorname*{argmin}_{\mathbf{M}\in\mathbb{R}^{n,n}} \left\{ \max_{\mathbf{m}\in\mathbb{R}^n} [L(\mathbf{M},\mathbf{m})] \right\}. \tag{3.0.21}$$

The matrix $\mathbf{M}^*$ should be the same of (3.0.20).

HIDDEN HINT 1 for (3.3.a) → 3.3.1:LSQFrobLagr1h.pdf

SOLUTION for (3.3.a) → 3.3.1:LSQFrobLagr1s.pdf ▲

(3.3.b) ⚁ Find the derivative $\mathbf{grad}\,\Phi \in \mathbb{R}^{n,n}$ of the function Φ defined as:

$$\Phi : \mathbb{R}^{n,n} \to \mathbb{R}, \quad \Phi(\mathbf{X}) = \|\mathbf{X}\|_F^2 \tag{3.0.22}$$

HIDDEN HINT 1 for (3.3.b) → 3.3.2:LSQFrobLagr2h.pdf

SOLUTION for (3.3.b) → 3.3.2:LSQFrobLagr2s.pdf ▲

(3.3.c) ⚁ Use the result of (3.3.b) to derive saddle point equations for $\mathbf{grad}\, L(\mathbf{M},\mathbf{m}) = 0$, for the L obtained in (3.3.a).

HIDDEN HINT 1 for (3.3.c) → 3.3.3:LSQFrobLagr3h.pdf

SOLUTION for (3.3.c) → 3.3.3:LSQFrobLagr3s.pdf ▲

(3.3.d) ⚀ Solve the saddle point equations obtained in (3.3.c) in symbolic form.

HIDDEN HINT 1 for (3.3.d) → 3.3.4:LSQFrobLagr4h.pdf

SOLUTION for (3.3.d) → 3.3.4:LSQFrobLagr4s.pdf ▲

End Problem 3.3

Problem 3.4: Hidden linear regression

This problem is about a hidden linear regression: in fact, at first glance it will seem that we are dealing with a nonlinear system of equations. However, we will be able to reduce the system to a linear form.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

We consider the function $f(t) = \alpha e^{\beta t}$ with unknown parameters $\alpha > 0$, $\beta \in \mathbb{R}$. Given are measurements (t_i, y_i), $i = 1, \ldots, n$, $2 < n \in \mathbb{N}$. We want to determine $\alpha, \beta \in \mathbb{R}$ such that $f(t)$ fulfills the following condition "to the best of its ability" (in the least square sense):

$$f(t_i) = y_i, \quad \text{with} \quad y_i > 0,\ i = 1, \ldots, n\,. \tag{3.0.27}$$

(3.4.a) ⚀ Which overdetermined system of *nonlinear* equations directly stems from (3.0.27)?

SOLUTION for (3.4.a) → `3.4.1:AdaptedLinReg1s.pdf` ▲

(3.4.b) ⚁ In which overdetermined system of *linear* equations can you transform the nonlinear system derived in (3.4.a)?

HIDDEN HINT 1 for (3.4.b) → `3.4.2:AdaptedLinReg2h.pdf`

SOLUTION for (3.4.b) → `3.4.2:AdaptedLinReg2s.pdf` ▲

(3.4.c) ⚂ Determine the normal equations for the overdetermined linear system of (3.4.b).

HIDDEN HINT 1 for (3.4.c) → `3.4.3:AdaptedLinReg3h.pdf`

SOLUTION for (3.4.c) → `3.4.3:AdaptedLinReg3s.pdf` ▲

(3.4.d) ⚃ Consider the data (t_i, y_i), $i = 1, \ldots, m$, stored in two vectors `t` and `y` of equal length. Implement a C++ function that solves the linear regression problem in the sense of ➜§ 5.7.7 and ➜Ex. 3.1.5 for $\mathbf{t}$ and $\mathbf{y}$:

```
VectorXd linReg(const VectorXd &t, const VectorXd &y);
```

This function should return the estimated parameters in a vector of length 2. You can use dedicated EIGEN classes for the solution of linear least squares problems or simply implement the normal equations from ➜Thm. 3.1.10.

SOLUTION for (3.4.d) → `3.4.4:AdaptedLinReg4s.pdf` ▲

(3.4.e) ⚃ Consider the data (t_i, y_i) stored in two vectors `t` and `y` of equal length. Implement a C++ function that returns a least squares estimate of α and β, given the nonlinear relationship between $\mathbf{t}$ and $\mathbf{y}$ (3.0.27):

```
VectorXd expFit(const VectorXd &t, const VectorXd &y);
```

You can use the function `linReg()` implemented in (3.4.d).

HIDDEN HINT 1 for (3.4.e) → `3.4.5:AdaptedLinReg5h.pdf`

SOLUTION for (3.4.e) → `3.4.5:AdaptedLinReg5s.pdf` ▲

End Problem 3.4

Problem 3.5: Estimating a Tridiagonal Matrix

To determine the least squares solution of an overdetermined linear system of equations $\mathbf{Ax} = \mathbf{b}$ we minimize the residual norm $\|\mathbf{Ax} - \mathbf{b}\|_2$ w.r.t. $\mathbf{x}$. However, we also face a linear least squares problem when minimizing the residual norm w.r.t. the entries of $\mathbf{A}$.
This is the situation considered in this problem.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Let two vectors $\mathbf{z}, \mathbf{c} \in \mathbb{R}^n$, $n > 2 \in \mathbb{N}$ of measurements be given. α^* and β^* are defined as:

$$(\alpha^*, \beta^*) = \operatorname*{argmin}_{\alpha,\beta\in\mathbb{R}} \|\mathbf{T}_{\alpha,\beta}\mathbf{z} - \mathbf{c}\|_2 \tag{3.0.30}$$

$\mathbf{T}_{\alpha,\beta} \in \mathbb{R}^{n\times n}$ is the following tridiagonal matrix:

$$\mathbf{T}_{\alpha,\beta} = \begin{bmatrix} \alpha & \beta & 0 & \dots & 0 \\ \beta & \alpha & \beta & \ddots & \vdots \\ 0 & \beta & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \alpha & \beta \\ 0 & \dots & 0 & \beta & \alpha \end{bmatrix} \tag{3.0.31}$$

(3.5.a) ⚀ Reformulate Eq. (3.0.30) as a linear least squares problem in the usual form:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^k} \|\mathbf{Ax} - \mathbf{b}\|_2 \tag{3.0.32}$$

Define suitable $\mathbf{A} \in \mathbb{R}^{m,k}$, $\mathbf{x} \in \mathbb{R}^k$ and $\mathbf{b} \in \mathbb{R}^m$, with $m, k \in \mathbb{N}$.

HIDDEN HINT 1 for (3.5.a) → `3.5.1:TridiagLeastSquares1h.pdf`

SOLUTION for (3.5.a) → `3.5.1:TridiagLeastSquares1s.pdf` ▲

(3.5.b) ⚄ Write a C++ function that computes the optimal parameters α^* and β^* according to Eq. (3.0.30) from data vectors `z` and `c` (i.e. $\mathbf{z}$ and $\mathbf{c}$ from Eq. (3.0.30)).

```
VectorXd lsqEst(const VectorXd &z, const VectorXd &c);
```

You can use EIGEN classes.

HIDDEN HINT 1 for (3.5.b) → `3.5.2:TridiagLeastSquares2h.pdf`

SOLUTION for (3.5.b) → `3.5.2:TridiagLeastSquares2s.pdf` ▲

End Problem 3.5

Problem 3.6: Approximation of a circle

In this problem we study how a fitting problem arising in computational geometry can be solved using least squares in several ways, leading to different results.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Let us consider a sequence of N points approximately located on a circle ($N = 8$):

x_i	0.7	3.3	5.6	7.5	6.4	4.4	0.3	-1.1
y_i	4.0	4.7	4.0	1.3	-1.1	-3.0	-2.5	1.3

A circle with center (m_1, m_2) and radius r is described by

$$(x - m_1)^2 + (y - m_2)^2 = r^2. \tag{3.0.35}$$

3.6.I: linear algebraic fit

(3.6.a) ⚀ Plugging the point coordinates (x_i, y_i) into (3.0.35) we obtain an overdetermined linear system with three unknowns

$$m_1,\ m_2,\ c := r^2 - m_1^2 - m_2^2.$$

Specify the system matrix $\mathbf{A}$ and the right-hand side vector $\mathbf{b}$ of the corresponding *linear* least squares problem ➜Eq. (3.1.38).

SOLUTION for (3.6.a) → `3.6.1:cala1.pdf` ▲

(3.6.b) ⚁

Write a C++ function

```
Vector3d circl_alg_fit(const VectorXd &x, const VectorXd & y);
```

that receives point coordinates in the vectors `x` and `y` and solves the overdetermined system in least squares sense and returns a 3-dimensional vector (m_1, m_2, r).

SOLUTION for (3.6.b) → `3.6.2:cala2.pdf` ▲

3.6.II: geometric fit

The algebraic approach lacks an intuitive geometrical meaning: minimising the equation residual of (3.0.35) in least squares sense does not necessarily yield the best circle fit in aesthetic sense.

A more appealing approach consist in the minimization of the distances between the data points and the (unknown) circle

$$d_i = \left|\sqrt{(m_1 - x_i)^2 + (m_2 - y_i)^2} - r\right| \qquad i = 1, \ldots, N,$$

in the sense of least squares, i.e., determine m_1, m_2 and r such that the sum $\sum_{i=1}^{n} d_i^2$ is minimal. This is a *non-linear* least squares problem of the form

$$\mathbf{z}^* = \operatorname*{argmin}_{\mathbf{z}} \|\mathbf{F}(\mathbf{z})\|^2,$$

see ➜Section 8.6.

(3.6.c) ⚀ Write down the concrete function $\mathbf{F} : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ for this non-linear least squares problem.

You can use the auxiliary values

$$R_i = \sqrt{(x_i - m_1)^2 + (y_i - m_2)^2}\,, \quad i = 1, \ldots, N.$$

SOLUTION for (3.6.c) → 3.6.3:cage1.pdf ▲

(3.6.d) ⚁

Define the functional

$$\Phi(\mathbf{z}) := \frac{1}{2}\|\mathbf{F}(\mathbf{z})\|^2.$$

Compute the Jacobian $\mathbf{DF}$ of $\mathbf{F}$, the gradient $\mathbf{grad}\,\Phi(\mathbf{z})$ of Φ and the Hessian $\mathbf{H}\Phi(\mathbf{z})$.

SOLUTION for (3.6.d) → 3.6.4:cage2.pdf ▲

(3.6.e) ⚁ Use C++ to find the circle that fit best the data given in the table above, according to the distances d_i. In order to do this, implement a C++ function

```
Vector3d circl_geo_fit(const VectorXd &x, const VectorXd & y);
```

that uses the Gauss-Newton method introduced in ➜Section 8.6.2 to minimize the functional Φ.

SOLUTION for (3.6.e) → 3.6.5:cage3.pdf ▲

(3.6.f) ⚁ Now solve the same problem using the Newton method for least squares equations as described in ➜Section 8.6.1.

SOLUTION for (3.6.f) → 3.6.6:cage4.pdf ▲

(3.6.g) ⚀ Compare the convergence of Gauss-Newton and Newton methods implemented in the previous sub-problems. You can use the parameters determined by the algebraic fit as initial guess.

Measure the error in the parameters in the maximum norm.

SOLUTION for (3.6.g) → 3.6.7:cage5.pdf ▲

3.6.III: constrained fit using SVD

As you may know, for $a \neq 0$ the solution set of the quadratic equation

$$a\mathbf{x}^T\mathbf{x} + \mathbf{b}^T\mathbf{x} + c = 0, \quad \mathbf{b} \in \mathbb{R}^2,\ a, c \in \mathbb{R}, \tag{3.0.41}$$

(solved with respect to $\mathbf{x} \in \mathbb{R}^2$) describes a circle.

(3.6.h) ⚀ Derive an expression in terms of $a, \mathbf{b}, c$ for the center $\mathbf{m}$ and the radius r of the circle defined by (3.0.41).

SOLUTION for (3.6.h) → 3.6.8:casvd1.pdf ▲

(3.6.i) ⚁ According to equation (3.0.41), the same circle can be defined by different parameter vectors $\mathbf{v} = (a, b_1, b_2, c)^T$ and $\mathbf{v}' = (a', b_1', b_2', c')^T$, when $\mathbf{v}' = \lambda\mathbf{v}$, for every $\lambda \in \mathbb{R}$, $\lambda \neq 0$. Thus, this equation, for different data values (x_i, y_i), can be solved in a least squares sense if supplemented by a *non-linear* constraint:

$$a\mathbf{x}_i^T\mathbf{x}_i + \mathbf{b}^T\mathbf{x}_i + c = 0 \quad i = 1, \ldots, N, \qquad \left\|(a, b_1, b_2, c)^T\right\|_2^2 = 1\,.$$

Write a MATLAB function

```
[m,r] = circ_svd_fit(x,y)
```

that solves this constrained overdetermined linear system of equations in least squares sense. Use the data in the table from the previous subtasks.

To learn how to use SVD to solve this problem, study carefully the hyperplane fitting problem ➜Ex. 3.4.35 and ➜Eq. (3.4.31).

SOLUTION for (3.6.i) → `3.6.9:casvd2.pdf` ▲

3.6.IV: comparison of the results

(3.6.j) ⚀ Draw the data points and the fitted circles computed in the previous subtasks. Compare the centers and the radii.

SOLUTION for (3.6.j) → `3.6.10:caco.pdf` ▲

End Problem 3.6

Problem 3.7: Shape identification

This problem deals with pattern recognition, a central problem in image processing (e.g. in aerial photography, self-driving cars, and recognition of pictures of cats). We will deal with a very simplistic problem.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

A routine within the program of a self driving-car is tasked with the job of identifying road signs. The task is the following: given a collection of points $\mathbf{p}_i \in \mathbb{R}^2, i = 1, \ldots, n$, we have to decide whether those point represent a stop sign, or a "priority road sign". In this exercise we consider $n = 8$.

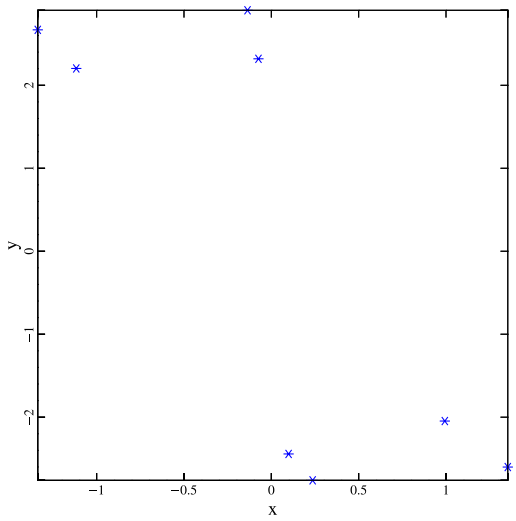


Fig. 11

Example of input points $\mathbf{p}^i$.

The shape of the sign can be represented by a 2×8 matrix with the 8 coordinates in $\mathbb{R}^2$ defining the shape of the sign. We assume that the stop sign resp. the priority road sign are defined by the "model" points (known a priori) $\mathbf{x}^i_{stop} \in \mathbb{R}^2$ resp. $\mathbf{x}^i_{priority} \in \mathbb{R}^2$ for $i = 2, \ldots, 15$.

Stop sign

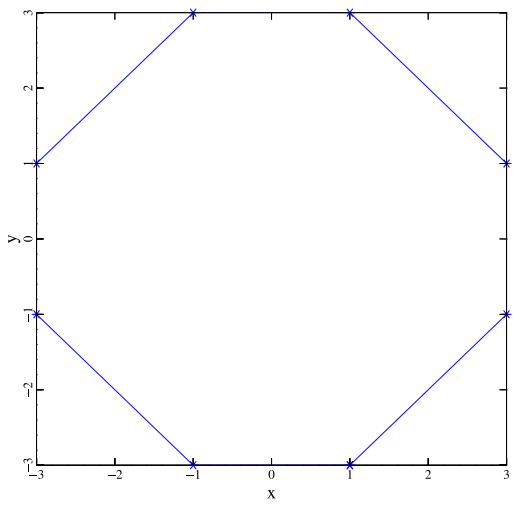


Fig. 12

The 8-points $\mathbf{x}^i_{stop}$ defining the model of a stop sign.

Priority road sign

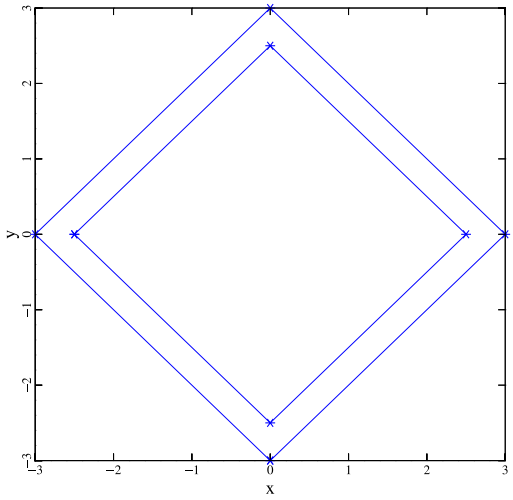


Fig. 13

The 8-points $\mathbf{x}^i_{priority}$ defining the model of a priority road sign.

However, in a real case scenario, one can imagine that the photographed shape is not exactly congruent to the one specified by the "model" points. Instead, one can assume that the points on the photo ($\mathbf{p}^i \in \mathbb{R}^2$) are the result of a *linear transformation* of the original points $\mathbf{x}^i \in \mathbb{R}^2$ for $i = 1, \ldots, 15$; i.e. we can assume that there exists a matrix $\mathbf{A} \in \mathbb{R}^{2,2}$, s.t.

$$\mathbf{p}^i = \mathbf{A}\mathbf{x}^i, i = 1, \ldots, n. \tag{3.0.42}$$

We do not know whether $\mathbf{x}^i = \mathbf{x}^i_{stop}$ or $\mathbf{x}^i = \mathbf{x}^i_{priority}$.

Moreover, we have some error in the data, i.e. our points do not represent exactly one of the linearly transformed shapes (it is plausible to imagine that there is some measurement error, and that the photographed shape is not exactly the same as our "model" shape), i.e. (3.0.42) is satisfied only "approximately".

With this problem, we will try to use the least square method to find the matrix $\mathbf{A}$ and to *classify* our points, i.e. to decide whether they represent a stop or a priority road sign.

(3.7.a) ⊡ For the moment, assume we know the points $\mathbf{x}^i$ (i.e. we know the shape of our sign). Explicitly write (3.0.42) as an overdetermined linear system:

$$\mathbf{w} = \mathbf{B}\mathbf{v},$$

whose least square solution will allow to determine the "best" linear transformation $\mathbf{A}$ (in the least square sense). What is the size and what are the unknown of the system?

SOLUTION for (3.7.a) → `3.7.1:ShapeIdent1.pdf` ▲

(3.7.b) ⊡ When does the matrix $\mathbf{B}$ have full rank? Give a geometric interpretation of this condition.

SOLUTION for (3.7.b) → `3.7.2:si2s.pdf` ▲

(3.7.c) ⊡ Implement a C++ function

```
MatrixXd shape_ident_matrix(const MatrixXd & X);
```

that returns the matrix $\mathbf{B}$. Pass the vectors $\mathbf{x}^i$ as a $2 \times n$ EIGEN matrix `X`.

SOLUTION for (3.7.c) → `3.7.3:si3s.pdf` ▲

(3.7.d) ⊡ Implement a C++ function

```
double solve_lsq(const MatrixXd & X,
                 const MatrixXd & P,
                 MatrixXd & A);
```

that computes the matrix $\mathbf{A}$ by solving the least squares problem based on the normal equations, see ➜Section 3.2. It should return the Euclidean norm of the residual of the least square solution. Pass the vectors $\mathbf{x}^i$ and the vectors $\mathbf{p}^i$ as a $2 \times n$ EIGEN matrix `X` resp. `P`.

SOLUTION for (3.7.d) → `3.7.4:si4s.pdf` ▲

(3.7.e) ⊡ Explain how the norm of the residual of the least square solution can be used to identify the shape defined by the points $\mathbf{p}^i$. Implement a function

```
enum Shape { Stop, Priority };

Shape identify(const MatrixXd Xstop,
               const MatrixXd Xpriority,
               const MatrixXd & P,
               MatrixXd & A);
```

that indentifies the shape (either Stop or Priority sign) of the input points $\mathbf{p}^i$. The function returns an `enum` that classifies the shape of points specified by `P`. Return the linear transformation in the matrix `A`. The "model points" $\mathbf{x}^i_{stop}$ resp. $\mathbf{x}^i_{priority}$ are passed trough `Xstop` resp. `Xpriority`.

SOLUTION for (3.7.e) → `3.7.5:si5s.pdf` ▲

(3.7.f) ⊡ Use the points provided in the variables `P1`, `P2`, and `P3` in the `main()` function to identify the shape (or: which shape is best suited) of the objects defined by those points.

SOLUTION for (3.7.f) → `3.7.6:si6s.pdf` ▲

End Problem 3.7

Problem 3.8: Properties of Householder reflections

➜Section 3.3.3 describes an orthogonal transformation that can be used to map a vector onto another vector of the same length: the Householder transformation (3.3.16). In this problem we examine properties of the "Householder matrices" and construct some of them.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Householder transformation are described by square matrices of the form

$$\mathbf{H} = \mathbf{I} - 2\mathbf{v}\mathbf{v}^H \qquad \text{with} \quad \mathbf{v}^H\mathbf{v} = 1, \quad \mathbf{v} \in \mathbb{C}^n.$$

We study a few important properties of this class of matrices and delve into the geometric meaning of Householder reflections.

(3.8.a) ⚀ Prove the following properties:

i) $\mathbf{H}\mathbf{H} = \mathbf{I}$.

ii) $\mathbf{H}\mathbf{H}^H = \mathbf{I}$.

iii) $|\det(\mathbf{H})| = 1$.

iv) For every $\mathbf{s} \in \mathbb{C}^n$ perpendicular to $\mathbf{v}$ (i.e. $\mathbf{v}^H\mathbf{s} = 0$): $\mathbf{H}\mathbf{s} = \mathbf{s}$.

v) For every $\mathbf{z} \in \mathbb{C}^n$ collinear to $\mathbf{v}$ (i.e. $\mathbf{z} = c\mathbf{v}$): $\mathbf{H}\mathbf{z} = -\mathbf{z}$.

SOLUTION for (3.8.a) → `3.8.1:householder1.pdf` ▲

(3.8.b) ⚀

In real space the transformation $\mathbf{H}$ can be understood as a reflection across a hyperplane perpendicular to $\mathbf{v}$. Compute the matrix $\mathbf{H}$ and sketch the corresponding transformation for $\mathbf{v} = \frac{1}{\sqrt{10}}[1,3]^T$, $n = 2$. What is the line of reflection?

SOLUTION for (3.8.b) → `3.8.2:householder2.pdf` ▲

(3.8.c) ⚀

The matrix

$$\mathbf{C} = \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}$$

should be transformed to an upper triangular matrix using the Householder transformation ($\mathbf{H}\mathbf{C} = \mathbf{R}$).

Compute:

- the vector $\mathbf{v}$ that defines $\mathbf{H}$ (is it unique?);
- the matrix $\mathbf{H}$;
- the images of the columns of $\mathbf{C}$ under the transformation $\mathbf{H}$.

SOLUTION for (3.8.c) → `3.8.3:householder3.pdf` ▲

(3.8.d) ⚀ Implement a function

```
void applyHouseholder(VectorXd &x, const VectorXd &v)
```

that efficiently computes $\mathbf{x} \leftarrow \mathbf{H}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^n$, for the Householder matrix ($\mathbf{v} \in \mathbb{R}^n$)

$$\mathbf{H} = \mathbf{I} - 2\mathbf{w}\mathbf{w}^\top, \quad \mathbf{w} := \frac{\mathbf{v}}{\|\mathbf{v}\|_2}.$$

What is the asymptotic complexity of your implementation for $n \to \infty$?

SOLUTION for (3.8.d) → 3.8.4:householder2.pdf ▲

(3.8.e) ⚀ As we learned in ➜Rem. 3.3.25, the sequence of Householder transformations $\mathbf{H}_1, \ldots, \mathbf{H}_{n-1}$ effecting the conversion of an $n \times n$ matrix into upper triangular form (→ ➜§ 3.3.15) is represented by the sequence of their defining *unit vectors* $\mathbf{v}_i \in \mathbb{R}^n$, also following the convention that $(\mathbf{v}_i)_i \geq 0$: $\mathbf{H}_i := \mathbf{I} - 2\mathbf{v}_i\mathbf{v}_i^\top$. The vectors $\mathbf{v}_i$ are stored in the *strictly lower triangular* part of another $n \times n$ matrix $\mathbf{V} \in \mathbb{R}^{n,n}$. More precisely, we have

$$\mathbf{v}_i = [0, \ldots, 0, (\mathbf{v}_i)_i, (\mathbf{V})_{i+1:n,i}]^\top \in \mathbb{R}^n \,, \quad i = 1, \ldots, n-1 \,.$$

Write a C++/EIGEN function

```
template <typename Scalar>
void applyHouseholder(VectorXd Xd &x,
                      const MatrixBase<Scalar> &V)
```

that computes $\mathbf{x} \leftarrow \mathbf{H}_{n-1}^{-1} \ldots \mathbf{H}_1^{-1}\mathbf{x}$ based on Householder vectors stored in $\mathbf{V}$ as described above.

HIDDEN HINT 1 for (3.8.e) → 3.8.5:house:mb.pdf

HIDDEN HINT 2 for (3.8.e) → 3.8.5:house:h1.pdf

SOLUTION for (3.8.e) → 3.8.5:householder2.pdf ▲

End Problem 3.8

Problem 3.9: Cholesky and QR decomposition

This problem is about the Cholesky and QR decomposition and the relationship between them. The Cholesky decomposition is explained in ➜§ 2.8.13. The QR decomposition is explained in ➜Section 3.3.3. Please study these topics before tackling this problem.

Template: Get it on GitLab.
Solution: Get it on GitLab.

(3.9.a) ⚀ Show that, for every matrix $\mathbf{A} \in \mathbb{R}^{m,n}$ such that $\operatorname{rank}(\mathbf{A}) = n$, the product matrix $\mathbf{A}^\top\mathbf{A}$ admits a Cholesky decomposition.

HIDDEN HINT 1 for (3.9.a) → `3.9.1:CholeskyQR1h.pdf`

SOLUTION for (3.9.a) → `3.9.1:CholeskyQR1s.pdf` ▲

(3.9.b) ⚄ The following C++ functions with EIGEN are given:

C++11-code 3.0.50: QR-decomposition via Cholesky decomposition

```
2  void CholeskyQR(const MatrixXd & A, MatrixXd & R, MatrixXd & Q) {
3
4      MatrixXd AtA = A.transpose() * A;
5      LLT<MatrixXd> L = AtA.llt();
6      R = L.matrixL().transpose();
7      Q =
           R.transpose().triangularView<Lower>().solve(A.transpose()).transpose()
8      // .triangularView() template member only accesses the
           triangular part
9      // of a dense matrix and allows to easily solve linear problem
10 }
```

Get it on GitLab (choleskyQR.cpp).

C++11-code 3.0.51: Standard way to do QR-decomposition in EIGEN

```
2  void DirectQR(const MatrixXd & A, MatrixXd & R, MatrixXd & Q) {
3
4      size_t m = A.rows();
5      size_t n = A.cols();
6
7      HouseholderQR<MatrixXd> QR = A.householderQr();
8      Q = QR.householderQ() * MatrixXd::Identity(m, std::min(m, n));
9      R = MatrixXd::Identity(std::min(m, n), m) *
           QR.matrixQR().triangularView<Upper>();
10     // If A: m x n, then Q: m x m and R: m x n.
11     // If m > n, however, the extra columns of Q and extra rows of R
           are not needed.
12     // Matlab returns this "economy-size" format calling "qr(A,0)",
13     // which does not compute these extra entries.
14     // With the code above, Eigen is smart enough to not compute the
           discarded vectors.
15 }
```

Get it on GitLab (choleskyQR.cpp).

Prove that, for every matrix $\mathbf{A}$, `CholeskyQR` and `DirectQR` produce the *same* output matrices $\mathbf{Q}$ and $\mathbf{R}$, if there were no roundoff errors (Such functions are called algebraically equivalent).

HIDDEN HINT 1 for (3.9.b) → `3.9.2:CholeskyQR2h.pdf`

SOLUTION for (3.9.b) → `3.9.2:CholeskyQR2s.pdf` ▲

(3.9.c) ⚄ Let `EPS` denote the machine precision. Why does the function `CholeskyQR` from Subproblem (3.9.b) fail to return the correct result for $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ \frac{1}{2}\mathrm{EPS} & 0 \\ 0 & \frac{1}{2}\mathrm{EPS} \end{bmatrix}$?

HIDDEN HINT 1 for (3.9.c) → `3.9.3:cencel.pdf`

HIDDEN HINT 2 for (3.9.c) → `3.9.3:CholeskyQR3h.pdf`

SOLUTION for (3.9.c) → `3.9.3:CholeskyQR3s.pdf` ▲

End Problem 3.9

Problem 3.10: Low rank approximation of matrices

As explained in the course, large $m \times n$ matrices of low rank $k \ll \min\{m,n\}$ can be stored using only $k(m+n)$ real numbers when using their SVD factorization/representation as sum of tensor products ➜Eq. (3.4.8). Thus, low rank matrices are of considerable interest for *matrix compression* (see ➜Ex. 3.4.51). Unfortunately, adding two low rank matrices usually leads to an increase of the rank and entails "recompression" by computing a low-rank best approximation of the sum. This problems demonstrates an efficient approach to recompression.
This problem discusses how this can be done in an efficient way.

Template: Get it on GitLab.
Solution: Get it on GitLab.

(3.10.a) ⚀ Show that for a matrix $\mathbf{X} \in \mathbb{R}^{m,n}$ the following statements are equivalent:

(i) $\operatorname{rank}(\mathbf{X}) = k$

(ii) $\mathbf{X} = \mathbf{A}\mathbf{B}^\top$ for $\mathbf{A} \in \mathbb{R}^{m,k}$, $\mathbf{B} \in \mathbb{R}^{n,k}$, $k \le \min\{m,n\}$, both full rank.

HIDDEN HINT 1 for (3.10.a) → 3.10.1:LowRankRep1h.pdf

HIDDEN HINT 2 for (3.10.a) → 3.10.1:LowRankRep1h.pdf

SOLUTION for (3.10.a) → 3.10.1:LowRankRep1s.pdf ▲

(3.10.b) ⚁ Write a C++ function that factorizes the matrix $\mathbf{X}$ with $\operatorname{rank}(\mathbf{X}) = k$ into $\mathbf{A}\mathbf{B}^\top$, as in (3.10.a):

```
void factorize_X_AB(const MatrixXd & X, size_t k, MatrixXd
  & A, MatrixXd & B);
```

The function should issue a warning in case $\operatorname{rank}(\mathbf{X}) = k$ is in doubt. You can use EIGEN classes.

HIDDEN HINT 1 for (3.10.b) → 3.10.2:LowRankRep2h.pdf

SOLUTION for (3.10.b) → 3.10.2:LowRankRep2s.pdf ▲

(3.10.c) ⚂ Let $\mathbf{A} \in \mathbb{R}^{m,k}$, $\mathbf{B} \in \mathbb{R}^{n,k}$, with $k \ll m,n$. Write an efficient C++ function that calculates a singular value decomposition of the product $\mathbf{A}\mathbf{B}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where orthogonal $\mathbf{U},\mathbf{V} \in \mathbb{R}^{n,k}$ and $\mathbf{\Sigma} \in \mathbb{R}^{k,k}$:

```
void svd_AB(const MatrixXd & A, const MatrixXd & B, MatrixXd & U,
  MatrixXd & S, MatrixXd & V);
```

HIDDEN HINT 1 for (3.10.c) → 3.10.3:LowRankRep3h.pdf

HIDDEN HINT 2 for (3.10.c) → 3.10.3:LRRh.pdf

SOLUTION for (3.10.c) → 3.10.3:LowRankRep3s.pdf ▲

(3.10.d) ⚀ What is the asymptotic computational cost of the function `svd_AB` for a small k and $m = n \to \infty$? Discuss the effort required by the different steps of your algorithm.

SOLUTION for (3.10.d) → 3.10.4:LowRankRep4s.pdf ▲

(3.10.e) ⚀ If $\mathbf{X},\mathbf{Y} \in \mathbb{R}^{m,n}$ satisfy $\operatorname{rank}(\mathbf{Y}) = \operatorname{rank}(\mathbf{X}) = k$, show that $\operatorname{rank}(\mathbf{Y}+\mathbf{X}) \le 2k$.

SOLUTION for (3.10.e) → 3.10.5:LowRankRep5s.pdf ▲

(3.10.f) ⚀ Consider $\mathbf{A}_X, \mathbf{A}_Y \in \mathbb{R}^{m,k}$, $\mathbf{B}_X, \mathbf{B}_Y \in \mathbb{R}^{n,k}$, $\mathbf{X} = \mathbf{A}_X\mathbf{B}_X^\top$ and $\mathbf{Y} = \mathbf{A}_Y\mathbf{B}_Y^\top$. Find a factorization of sum $\mathbf{X}+\mathbf{Y}$ as $\mathbf{X}+\mathbf{Y} = \mathbf{A}\mathbf{B}^\top$, with $\mathbf{A} \in \mathbb{R}^{m,2k}$ and $\mathbf{B} \in \mathbb{R}^{n,2k}$.

HIDDEN HINT 1 for (3.10.f) → 3.10.6:LowRankRep6h.pdf

SOLUTION for (3.10.f) → 3.10.6:LowRankRep6s.pdf ▲

(3.10.g) ⊡ Rely on the previous subproblems to write the efficient C++ function:

```
void rank_k_approx(const MatrixXd & Ax, const MatrixXd &
  Ay, const MatrixXd & Bx, const MatrixXd & By, MatrixXd
  & Az, MatrixXd & Bz);
```

$\mathbf{A}_Z \in \mathbb{R}^{m,k}$ and $\mathbf{B}_Z \in \mathbb{R}^{n,k}$ are the two terms of the decomposition of $\mathbf{Z} = \mathbf{A}_Z\mathbf{B}_Z^\top$, the rank-$k$ best approximation of the sum $\mathbf{X}+\mathbf{Y} = \mathbf{A}_X\mathbf{B}_X^\top + \mathbf{A}_Y\mathbf{B}_Y^\top$:

$$\mathbf{Z} = \operatorname*{argmin}_{\substack{\mathbf{M}\in\mathbb{R}^{m,n}\\ \operatorname{rank}(\mathbf{M})\le k}} \left\| \mathbf{A}_X\mathbf{B}_X^\top + \mathbf{A}_Y\mathbf{B}_Y^\top - \mathbf{M}\right\|_F$$

Here $\mathbf{A}_X, \mathbf{A}_Y \in \mathbb{R}^{m,k}$ and $\mathbf{B}_X, \mathbf{B}_Y \in \mathbb{R}^{n,k}$.

HIDDEN HINT 1 for (3.10.g) → 3.10.7:LowRankRep7h.pdf

SOLUTION for (3.10.g) → 3.10.7:LowRankRep7s.pdf ▲

(3.10.h) ⊡ What is the asymptotic computational cost of the function `rank_k_approx` for a small k and $m = n \to \infty$?

SOLUTION for (3.10.h) → 3.10.8:LowRankRep8s.pdf ▲

End Problem 3.10

Chapter 4

Filtering Algorithms

Problem 4.1: Autofocus with FFT

In this problem, we will use 2D frequency analysis to find the "best focused" image among a collection of out-of-focus photos. To that end, we will implement an algorithm based on 2D DFT as introduced in ➜Section 4.2.4.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Let a grey-scale image consisting of $n \times m$ pixels be given as a matrix $\mathbf{P} \in \mathbb{R}^{n,m}$ as in ➜Ex. 4.2.56; each element of the matrix indicates the gray-value of the pixel as a number between 0 and v_{max}. This image is regarded as the "perfect image".

If a camera is poorly focused due to an inadequate arrangement of the lenses, it will record a blurred image $\mathbf{B} \in \mathbb{R}^{n,m}$. As before ➜Eq. (4.2.57), the blurring operation can be modeled through the 2D (periodic) discrete convolution and can be undone provided that the point spread function (PSF) is known. However, in this problem we must not assume any knowledge of the PSF.

Assume that the only data available to us is the "black-box" C++ function (declared in `autofocus.hpp`):

```
MatrixXd set_focus(double f);
```

which returns the potentially blurred image $\mathbf{B}(f)$, when the focus parameter f is set to a particular value. The actual operation of `set_focus` is utterly obscure. The focus parameter can be read as the distance of the lenses of a camera that can be changed through turning on and off a stepper motor.

The problem of *autofocusing* is to determine the value of the focus parameter f, which yields an image $\mathbf{B}(f)$ with the least blur. The idea of the autofocusing algorithm is the following: *The less the image is marred by blur, the larger its "high frequency content"*. The latter can found out based on the discrete Fourier transform, more precisely, its 2D version.

To translate the autofocus idea into an algorithm we have to give a quantitative meaning to the notion of "high frequency content" of a (blurred) image $\mathbf{C}$, which is done by looking at the second moments of its 2D discrete Fourier transform as supplied by the function `fft2r` found in "`FFT/fft.hpp`", see also ➜Code 4.2.48 and Code 4.2.49.

$$V(\mathbf{C}) = \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \left(\left(\frac{n}{2} - \left| k_1 - \frac{n}{2} \right| \right)^2 + \left(\frac{m}{2} - \left| k_2 - \frac{m}{2} \right| \right)^2 \right) |\hat{\mathbf{C}}_{k_1,k_2}|^2 .$$

Here, $\hat{\mathbf{C}} \in \mathbb{C}^{n,m}$ stand for the 2D DFT of the image $\mathbf{C}$. Hence, the autofocusing policy can be rephrased as

find $f \in \mathbb{R}$ such that $V(\mathbf{B}(f))$ becomes maximal.

(4.1.a) ⚀ Thus sub-problem supplies us with a tool to write an image file from a C++ code. Write a C++ function

```
void save_image(double focus);
```

that saves the image $\mathbf{B}(f)$ returned by `set_focus` for $f = 0, 1, 2, 3$ in **P**ortable **G**ray**m**ap (PGM) format.

For this you can use objects of type `PGMObject` (found in "`pgm.hpp`"). You can find example of usages of this class in "`examples/pgm_example.cpp`".

SOLUTION for (4.1.a) → `4.1.1:render.pdf`

▲

(4.1.b) ⚁ The two-dimensional discrete Fourier transform can be performed using the C++function `fft2r`. Write a C++script:

```
void plot_freq(double focus);
```

that creates 3D plots of the (modulus of the) 2D DFTs of the images obtained in sub-problem (4.1.a). Clamp the data between 0 and 8000.

SOLUTION for (4.1.b) → `4.1.2:plotfft2.pdf`

▲

(4.1.c) ⚂

In plots from sub-problem (4.1.b), mark (or explain) the regions corresponding to the high and low frequencies.

HIDDEN HINT 1 for (4.1.c) → `4.1.3:afh1.pdf`

SOLUTION for (4.1.c) → `4.1.3:explain.pdf`

▲

(4.1.d) ⚂ Write a C++function

```
double void high_frequency_content(const MatrixXd & C);
```

that returns the value $V(\mathbf{C})$ for a given matrix $\mathbf{C}$. Write a C++function

```
void plotV();
```

that plots the function $V(\mathbf{B}(f))$ in terms of the focus parameter f. Use 100 equidistantly spaced sampling points in the interval $[0, 5]$.

SOLUTION for (4.1.d) → `4.1.4:plotV.pdf`

▲

(4.1.e) ⚃ Write an *efficient* (you want the auto-focus procedure to take not longer than a few seconds!) C++function

```
double autofocus();
```

that numerically determines optimal focus parameter $f_0 \in [0, 5]$ for which the 2nd moment $V(\mathbf{B}(f))$ is maximized. What is the resulting focus parameter f_0?

To locate the change of slope of $V(\mathbf{B}(f))$ in $[0,5]$ you should use the *bisection algorithm* ➜Section 8.3.1 for finding a zero of the derivative $\partial V(\mathbf{B}(f))/\partial f$, which will, hopefully, mark the location of the maximum of $V(f)$.

The bisection algorithm for finding a zero of a continuous function $\varphi : [a,b] \to \mathbb{R}$ with $\varphi(a) \cdot \varphi(b) < 0$ is rather simple and lucidly demonstrated in ➜Code 8.3.2.

Of course, the derivative is not available, so that we have to approximate it crudely by means of a difference quotient

$$\frac{\partial V(f)}{\partial f} \approx \frac{V(\mathbf{B}(f+\delta f)) - V(\mathbf{B}(f-\delta f))}{2\delta f} \,.$$

You can assume that the smallest step of the auto-focus mechanism is 0.05 and so the natural choice for δf is $\delta f = 0.05$. This maximal resolution also tells you a priori how many bisection steps should be performed.

Use the structure of the function $V(\mathbf{B}(f))$ that you observed in sub-problem (4.1.d). Use as few evaluations of $V(\mathbf{B}(f))$ as possible!

SOLUTION for (4.1.e) $\to$ `4.1.5:bisect.pdf`

▲

End Problem 4.1

Problem 4.2: FFT and least squares

This problem deals both with an application of fast Fourier transformation in the context of a least squares (data fitting) problems.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

A planet orbits the sun in a closed, planar curve. For equispaced polar angles $\varphi_j = 2\pi\frac{j}{n}$, $j = 0,\ldots,n$, $n \in \mathbb{N}$ its distance from the sun is measured and found to be d_j, $j = 0,\ldots,n-1$. An approximation of the planet's trajectory can be found as follows.

Write $\mathcal{P}_m^C$, $m \in \mathbb{N}$ for the space of real trigonometric polynomials of the form

$$p(t) = \sum_{k=0}^{m} c_k \cos(2\pi kt), \quad c_k \in \mathbb{R}. \tag{4.0.6}$$

The aim is to approximate the distance from the sun as a function of the angle with a trigonometric polynomial $p^* \in \mathcal{P}_m^C$ that solves

$$p^* = \operatorname*{argmin}_{p\in\mathcal{P}_m^C} \sum_{j=0}^{n-1} \left| p\left(\frac{\varphi_j}{2\pi}\right) - d_j \right|^2.$$

Throughout, we assume $m < n$.

(4.2.a) ⚀ Recast this task as a standard linear least squares problem

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$$

with suitable matrix $\mathbf{A}$ and vectors $\mathbf{b}, \mathbf{x}$.

SOLUTION for (4.2.a) → `4.2.1:relsq.pdf`

▲

(4.2.b) ⚁ State the normal equations for this linear least squares problem in explicit form.

With $i^2 = -1$ use the identity $\cos(2\pi x) = \frac{1}{2}(e^{2\pi ix} + e^{-2\pi ix})$ and the geometric series formula

$$\sum_{\ell=0}^{n-1} q^\ell = \frac{1-q^n}{1-q}.$$

SOLUTION for (4.2.b) → `4.2.2:normleq.pdf`

▲

(4.2.c) ⚂ Write a C++function

```
VectorXd find_c(const VectorXd & d, unsigned int m);
```

that computes the coefficients c_k, $k = 0,\ldots,m$ of p^* in the form (4.0.6) for arbitrary inputs $d_j \in \mathbb{R}$, $j = 0,\ldots,n-1$. These are passed as vector `d`. The function must not have asymptotic complexity worse than $O(n\log n)$ in terms of the problem size parameter n.

Use the provided function `VectorXd fftr(const VectorXd &)` that realises the discrete Fourier transform (defined in "`FFT/fft.hpp`"). Also note the identity

$$\sum_{k=0}^{n-1} e^{\frac{2\pi i}{n}d_k} = \overline{\sum_{k=0}^{n-1} e^{-\frac{2\pi i}{n}jk} d_k},$$

where the over-line denotes the complex conjugation.

SOLUTION for (4.2.c) → `4.2.3:implem.pdf`

▲

(4.2.d) ⊡ Test your implementation. To that end, test your routine by fitting a trigonometric polynomial of degree 3 with the data values `d` found in `main()` of `fftlsq.cpp`. (Get it on GitLab (fftlsq.cpp).). Store the result in the vector `g`.

As comparison, we obtain the vector $g = [0.984988, -0.00113849, -0.00141515]$ when running our code. ▲

End Problem 4.2

Problem 4.3: Multiplication and division of polynomials based on FFT

In ➜Rem. 4.1.24 we saw that the formula of discrete convolution ➜Def. 4.1.22 also gives the coefficients of products of polynomials. Thus, in light of the realization of discrete convolutions by means of DFT ➜Section 4.2.1, the Fast Fourier Transform (FFT) becomes relevant for efficiently multiplying polynomials of very high degree.

Template: Get it on GitLab.
Solution: Get it on GitLab.

Let two large numbers $m, n \gg 1$ and two polynomials

$$u(x) = \sum_{j=0}^{m-1} \alpha_j x^j, \quad v(x) = \sum_{j=0}^{n-1} \beta_j x^j, \quad \alpha_j, \beta_j \in \mathbb{C} \tag{4.0.8}$$

be given. We consider their product

$$uv(x) = u(x)v(x) \tag{4.0.9}$$

The tasks are:

1. Efficiently compute the coefficients of the polynomial uv (with degree $m+n-1$) from (4.0.9) is fulfilled.
2. Given uv and u in terms of their coefficients, find the polynomial v again, if it exists (polynomial division).

(4.3.a) ⚀ Given polynomials u and v with degrees $m-1$ and $n-1$, respectively, their coefficients can be stored as C++ vectors `Eigen::VectorXd u` and `Eigen::VectorXd v` of size m and n (the known term is stored in position 0).

Write a C++ function that "naively", i.e. using a simple loop-based implementation, computes the vector of coefficients of the polynomial which is the multiplication between polynomials u and v:

```
VectorXd polyMult_naive(const VectorXd & u, const VectorXd
    & v);
```

You can use EIGEN classes. Also determine the asymptotic complexity of your algorithm for $mn \to \infty$ (separately).

HIDDEN HINT 1 for (4.3.a) → `4.3.1:PolyDiv1h.pdf`

SOLUTION for (4.3.a) → `4.3.1:PolyDiv1s.pdf` ▲

(4.3.b) ⚀ Write a C++ function that efficiently computes the vector of coefficients of the polynomial which is the multiplication between polynomials u and v:

```
VectorXd polyMult_fast(const VectorXd & u, const VectorXd
    & v);
```

You can use EIGEN classes.

HIDDEN HINT 1 for (4.3.b) → `4.3.2:PolyDiv2h.pdf`

SOLUTION for (4.3.b) → `4.3.2:PolyDiv2s.pdf` ▲

(4.3.c) ⚀ Determine the complexity of your algorithm in (4.3.b).

HIDDEN HINT 1 for (4.3.c) → `4.3.3:PolyDiv3h.pdf`

SOLUTION for (4.3.c) → `4.3.3:PolyDiv3s.pdf` ▲

(4.3.d) ⊡ What does it mean to apply a discrete Fourier transform to the coefficients of a polynomial? In other words, what is an equivalent operation that can be performed on a polynomial which leads to the same result?

HIDDEN HINT 1 for (4.3.d) → `4.3.4:PolyDiv4h.pdf`

SOLUTION for (4.3.d) → `4.3.4:PolyDiv4s.pdf` ▲

(4.3.e) ⊡ Now we will handle with polynomial division.

Consider the division between polynomials uv and u. How can you check whether u divides uv?

HIDDEN HINT 1 for (4.3.e) → `4.3.5:PolyDiv5h.pdf`

SOLUTION for (4.3.e) → `4.3.5:PolyDiv5s.pdf` ▲

(4.3.f) ⚂ Write a C++ function that efficiently computes the vector of coefficients of the polynomial v, division between polynomials uv and u from the problems above:

```
VectorXd polyDiv(const VectorXd & uv, const VectorXd & u);
```

You can use EIGEN classes.

HIDDEN HINT 1 for (4.3.f) → `4.3.6:PolyDiv6h.pdf`

SOLUTION for (4.3.f) → `4.3.6:PolyDiv6s.pdf` ▲

End Problem 4.3

Problem 4.4: Solving triangular Toeplitz systems

In ➜Section 4.5 we learned about Toeplitz matrices, the class of matrices with constant diagonals ➜Def. 4.5.8. Obviously, $m \times n$ Toeplitz matrices are *data-sparse* in the sense that it takes only $m+n-1$ to encode them completely. Therefore, operations with Toeplitz matrices can often be done with asymptotic computational cost significantly lower than that of the same operations for a generic matrix of the same size. In ➜Section 4.5.1 we have seen this for FFT-based matrix×vector multiplication for Toeplitz matrices.
In this problem we study an efficient FFT-based algorithm for solving triangular linear systems of equations whose coefficient matrix is Toeplitz. Such linear systems are faced, for instance, when inverting a finite, linear, time-invariant, causal channel (LT-FIR) as introduced in ➜Section 4.1.

Template: Get it on GitLab.
Solution: Get it on GitLab.

In ➜§ 4.1.1 we found that the output operator of a causal finite linear time-invariant filter with impulse response $\mathbf{h} = (h_0, \ldots, h_{n-1})^\top \in \mathbb{R}^n$ can be obtained by discrete convolution ➜Eq. (4.1.14); see also ➜Def. 4.1.22. Given an input signal $\mathbf{x} := (x_0, \ldots, x_{n-1})^\top \in \mathbb{R}^n$, the first n components $(y_0, \ldots, y_{n-1})$ of the output are obtained by:

$$\begin{bmatrix} y_0 \\ \vdots \\ \vdots \\ y_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} h_0 & 0 & \cdots & & \cdots & 0 \\ h_1 & h_0 & 0 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ h_{n-1} & \cdots & & \cdots & h_1 & h_0 \end{bmatrix}}_{=:\mathbf{H}_n \in \mathbb{R}^{n,n}} \begin{bmatrix} x_0 \\ \vdots \\ \vdots \\ x_{n-1} \end{bmatrix} \tag{4.0.18}$$

The matrix $\mathbf{H}_n \in \mathbb{R}^{n,n}$ is a lower triangular matrix with constant diagonals:

$$(H)_{l,j} = \begin{cases} h_{l-j} & \text{if } l \geq j \\ 0 & \text{else} \end{cases}$$

This matrix is clearly not circulant: see ➜Def. 4.1.38. At the same time, it still belongs to the class of *Toeplitz matrices*, see ➜Def. 4.5.8, which generalizes the class of circulant matrices. (Recall ➜Def. 4.5.8: $\mathbf{T} \in \mathbb{K}^{m,n}$ is a Toeplitz matrix if there is a vector $\mathbf{u} = (u_{-m+1}, \ldots, u_{n-1}) \in \mathbb{K}^{m+n-1}$ such that $t_{ij} = u_{j-i}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.)

If you want to recover the input signal from the output (*deconvolution*), you will face the following crucial task for digital signal processing: *solve a lower triangular LSE with a Toeplitz system matrix*. Of course, forward elimination, see ➜Section 2.3, can achieve this with asymptotic complexity $\mathcal{O}(n^2)$: see ➜Eq. (2.3.6). However, since a Toeplitz matrix merely has an information content of $\mathcal{O}(n)$ numbers, there might be a more efficient way – which ought to be explored by you in this problem.

(4.4.a) ⊡ Given a Toeplitz matrix $\mathbf{T} \in \mathbb{R}^{n,n}$, find another matrix $\mathbf{S} \in \mathbb{R}^{n,n}$ such that the following composed matrix is a circulant matrix:

$$\mathbf{C} = \begin{bmatrix} \mathbf{T} & \mathbf{S} \\ \mathbf{S} & \mathbf{T} \end{bmatrix} \in \mathbb{R}^{2n,2n}$$

HIDDEN HINT 1 for (4.4.a) → `4.4.1:Toeplitz1h.pdf`

SOLUTION for (4.4.a) → 4.4.1:Toeplitz1s.pdf ▲

(4.4.b) ⚀ Write the following C++ function:

```
Eigen::MatrixXd toeplitz(const VectorXd &c, const VectorXd
    &r)
```

This function should take as input column vectors $\mathbf{c} \in \mathbb{R}^m$ and $\mathbf{r} \in \mathbb{R}^n$ and return a Toeplitz matrix $\mathbf{T} \in \mathbb{R}^{m,n}$ such that:

$$(T)_{i,j} = \begin{cases} (c)_{i-j+1} & \text{, if } i \geq j \\ (r)_{j-i+1} & \text{, if } j > i \end{cases} \quad 1 \leq i \leq m,\ 1 \leq j \leq n$$

SOLUTION for (4.4.b) → 4.4.2:tplcpp.pdf ▲

(4.4.c) ⚀ Show that the following two C++ functions realize the same linear mapping $\mathbf{x} \mapsto \mathbf{y}$, when supplied with the same arguments.

C++11-code 4.0.20: Function `toepmatmult`

```
2  VectorXd toepmatmult(const VectorXd & c, const VectorXd & r,
3                       const VectorXd & x)
4  {
5      assert(c.size() == r.size() &&
6             c.size() == x.size() &&
7             "c, r, x have different lengths!");
8
9      MatrixXd T = toeplitz(c,r);
10
11     VectorXd y = T*x;
12
13     return y;
14 }
```

Get it on GitLab (toeplitz.cpp).

C++11-code 4.0.21: Function `toepmult`

```
2  VectorXd toepmult(const VectorXd & c, const VectorXd & r,
3                    const VectorXd & x)
4  {
5      assert(c.size() == r.size() &&
6             c.size() == x.size() &&
7             "c, r, x have different lengths!");
8      int n = c.size();
9
10     VectorXcd cr_tmp = c.cast<std::complex<double>>();
11     cr_tmp.conservativeResize(2*n); cr_tmp.tail(n) =
           VectorXcd::Zero(n);
12     cr_tmp.tail(n-1).real() = r.tail(n-1).reverse();
13
14     VectorXcd  x_tmp = x.cast<std::complex<double>>();
15     x_tmp.conservativeResize(2*n);   x_tmp.tail(n) =
```

```
        VectorXcd::Zero(n);

    VectorXd y = pconvfft(cr_tmp, x_tmp).real();
    y.conservativeResize(n);

    return y;
}
```

Get it on GitLab (toeplitz.cpp).

Function `pconvfft` is defined in ➜Section 4.2.1, ➜Code 4.2.25, and `toeplitz` is the simple C++ function from Sub-problem (4.4.b).

HIDDEN HINT 1 for (4.4.c) → `4.4.3:Toeplitz1h.pdf`

SOLUTION for (4.4.c) → `4.4.3:Toeplitz2s.pdf` ▲

(4.4.d) ⚀ What is the asymptotic complexity of either `toepmatmult` or `toepmult`?

SOLUTION for (4.4.d) → `4.4.4:Toeplitz3s.pdf` ▲

(4.4.e) ⚄ Explain in detail what the following C++ functions are meant for and why they are algebraically equivalent (i.e. equivalent when ignoring roundoff errors), provided that `h.size()= 2^l`.

C++11-code 4.0.23: Function `ttmatsolve`

```
VectorXd ttmatsolve(const VectorXd & h, const VectorXd & y)
{
    assert(h.size() == y.size() &&
           "h and y have different lengths!");
    int n = h.size();

    VectorXd h_tmp = VectorXd::Zero(n);
    h_tmp(0) = h(0);

    MatrixXd T = toeplitz(h,h_tmp);

    VectorXd x = T.fullPivLu().solve(y);

    return x;
}
```

Get it on GitLab (toeplitz.cpp).

C++11-code 4.0.24: Function `ttrecsolve`

```
VectorXd ttrecsolve(const VectorXd & h, const VectorXd & y, int l)
{
    assert(h.size() == y.size() &&
           "h and y have different lengths!");

    VectorXd x;

    if(l == 0) {
```

```
10            x.resize(1);
11            x(0) = y(0)/h(0);
12        } else {
13            int n = std::pow(2,l);
14            int m = n/2;
15
16            assert(h.size() == n && y.size() == n &&
17                   "h and y have length different from 2^l!");
18
19            VectorXd x1 = ttrecsolve(h.head(m), y.head(m), l-1);
20            VectorXd y2 = y.segment(m,m) - toepmult(h.segment(m,m),
21                 h.segment(1,m).reverse(), x1);
22            VectorXd x2 = ttrecsolve(h.head(m), y2, l-1);
23
24            x.resize(n);
25            x.head(m) = x1;
26            x.tail(m) = x2;
27        }
28
29        return x;
30    }
```

Get it on GitLab (toeplitz.cpp).

HIDDEN HINT 1 for (4.4.e) → 4.4.5:Toeplitz4h.pdf

SOLUTION for (4.4.e) → 4.4.5:Toeplitz4s.pdf ▲

(4.4.f) ⚀ Why is the algorithm implemented in `ttrecsolve` called a "divide & conquer" method?

SOLUTION for (4.4.f) → 4.4.6:Toeplitz5s.pdf ▲

(4.4.g) ⚂ Perform a timing comparison of `ttmatsolve` and `ttrecsolve` for

```
h = VectorXd::LinSpaced(n,1,n).cwiseInverse();
```

and $n = 2^l$, $l = 3, \ldots, 11$. Plot the timing results for both functions in double logarithmic scale.

SOLUTION for (4.4.g) → 4.4.7:Toeplitz6s.pdf ▲

(4.4.h) ⚀ Derive the asymptotic complexity of both functions `ttmatsolve` and `ttrecsolve` in terms of the problem size parameter n.

SOLUTION for (4.4.h) → 4.4.8:Toeplitz7s.pdf ▲

(4.4.i) ⚄ For the case that `n = h.size()` is not a power of 2, implement a wrapper function for `ttrecsolve` that, in the absence of roundoff errors, would behave exactly like `ttmatsolve`, i.e. `ttsolve(h,y)= ttmatsolve(h,y)`.

```
VectorXd ttsolve(const VectorXd & h, const VectorXd & y);
```

HIDDEN HINT 1 for (4.4.i) → 4.4.9:tplhext.pdf

SOLUTION for (4.4.i) → 4.4.9:Toeplitz8s.pdf ▲

End Problem 4.4

Chapter 5

Data Interpolation in 1D

Problem 5.1: Evaluating the derivatives of interpolating polynomials

In ➜Ex. 5.1.5 we learned about the importance of data interpolation for obtaining functor representations of constitutive relationships $t \mapsto f(t)$. Numerical methods like Newton's method ➜Code 8.3.5 often require information about the derivative f' as well. Therfore, we need efficient ways to evaluate the derivatives of interpolants. In this problem we discuss this issue for polynomial interpolation for (i) monomial representation and (ii) "update-friendly" point evaluation. We generalize the Horner scheme ➜Rem. 5.2.5 and the Aitken-Neville algorithm ➜§ 5.2.33.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We first deal with polynomial in monomial representation ➜Rem. 5.2.4.

(5.1.a) ⚀ Using the Horner scheme, write an efficient C++ implementation of a template function which returns the pair $(p(x), p'(x))$, where p is a polynomial with coefficients in `c`:

```
template <typename CoeffVec>
std::pair<double,double> evaldp(const CoeffVec& c, double
   x);
```

Vector `c` contains the coefficient of the polynomial in the monomial basis, expressed with the MATLAB convention (leading coefficient in `c(0)`).

SOLUTION for (5.1.a) → `5.1.1:Horner1s.pdf` ▲

(5.1.b) ⚁ For the sake of testing, write a naive C++ implementation of function in (5.1.a) which returns the same pair $(p(x), p'(x))$. However, this time $p(x)$ and $p'(x)$ should be calculated with the simple sums of the monomials constituting the polynomial:

```
template <typename CoeffVec>
std::pair<double,double> evaldp_naive(const CoeffVec& c,
   double x);
```

SOLUTION for (5.1.b) → `5.1.2:Horner2s.pdf` ▲

(5.1.c) ⚂ What are the asymptotic complexities of the two functions in (5.1.a) and (5.1.b)?

SOLUTION for (5.1.c) → `5.1.3:Horner3s.pdf` ▲

(5.1.d) ⚀ Check the validity of the two functions in (5.1.a) and (5.1.b) and compare the runtimes for polynomials of degree up to $2^{20} - 1$.

HIDDEN HINT 1 for (5.1.d) → `5.1.4:Horner4h.pdf`

SOLUTION for (5.1.d) → `5.1.4:Horner4s.pdf` ▲

In ➜Section 5.2.3.2 we learned about an efficient and "update-friendly" scheme for evaluating Lagrange interpolants at a single or a few points. This so-called Aitken-Neville algorithm, see ➜Code 5.2.35, can be extended to return the derivative value of the polynomial interpolant as well. This will be explored next.

(5.1.e) ⚁ Write an efficient C++ function

```
VectorXd dipoleval(const VectorXd & t,
                   const VectorXd & y,
                   const VectorXd & x);
```

that returns the row vector $(p'(x_1), \ldots, p'(x_m))$, when the argument `x` passes $(x_1, \ldots, x_m)$, $m \in \mathbb{N}$ small. Here, p' denotes the *derivative* of the polynomial $p \in \mathcal{P}_n$ interpolating the data points (t_i, y_i), $i = 0, \ldots, n$, for pairwise different $t_i \in \mathbb{R}$ and data values $y_i \in \mathbb{R}$.

Differentiate the recursion formula ➜Eq. (5.2.34) and devise an algorithm in the spirit of the Aitken-Neville algorithm implemented in ➜Code 5.2.35.

SOLUTION for (5.1.e) → `5.1.5:dipoleval.pdf`

▲

(5.1.f) ⚁ For validation purposes devise an alternative, less efficient, implementation of `dipoleval` (call it `dipoleval_alt`) based on the following steps:

1. Use the `polyfit` (find it in "`polyfit.hpp`") function to compute the monomial coefficients of the Lagrange interpolant.
2. Compute the monomial coefficients of the derivative.
3. Use `polyval` (defined in "`polyval.hpp`") to evaluate the derivative at a number of points.

Use `dipoleval_alt` to verify the correctness of your implementation of `dipoleval` by computing the derivative of the 10-th order polynomial interpolating the values $(t_i, sin(t_i))$, where t_i are equidistant points in $[0, 3]$. Evaluate the Euclidean norm of the error of the resulting polynomials, evaluated at 100 equidistant points $x_i \in [0, 3]$.

SOLUTION for (5.1.f) → `5.1.6:test.pdf`

▲

End Problem 5.1

Problem 5.2: Piecewise linear interpolation

➜Ex. 5.1.10 introduced piecewise linear interpolation as a simple linear interpolation scheme. It finds an interpolant in the space spanned by the so-called tent functions, which are *cardinal basis functions*. Formulas are given in ➜Eq. (5.1.11).

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

(5.2.a) ⚄

Write a C++ class `LinearInterpolant` representing the piecewise linear interpolant. Make sure your class has an efficient internal representation of a basis. Provide a constructor and an evaluation `operator()` as described in the following template:

C++11-code 5.0.6:

```
2  class LinearInterpolant {
3  public:
4
5      /*!
6       * \brief LinearInterpolant builds interpolant from data.
7       * Sort the array for the first time:
8       * the data is not assumed to be sorted
9       * sorting is necessary for binary search
10      * \param TODO
11      */
12     LinearInterpolant(/* TODO: pass data here */);
13
14     /*!
15      * \brief operator () Evaluation operator.
16      * Return the value of I at x, i.e. I(x).
17      * Performs bound checks (i.e. if x < t_0 or x >= t_n ).
18      * \param x Value x ∈ ℝ.
19      * \return Value I(x).
20      */
21     double operator() (double x);
22 private:
23     // TODO: your data there
24 };
```

Get it on GitLab (linearinterpolant.cpp).

SOLUTION for (5.2.a) → `5.2.1:impl.pdf`

▲

End Problem 5.2

Problem 5.3: Lagrange interpolant

This short problem studies a particular aspect of the Lagrange polynomials introduced in ➜§ 5.2.10.

Template: Get it on GitLab.
Solution: Get it on GitLab.

As in ➜Eq. (5.2.11) we denote by L_i the i-th Lagrange polynomial for given nodes $t_j \in \mathbb{R}$, $j = 0, \ldots, n$, $t_i \neq t_j$, if $i \neq j$. Then the polynomial Lagrange interpolant p through the data points $(t_i, y_i)_{i=0}^n$ has the representation

$$p(x) = \sum_{i=0}^{n} y_i L_i(x) \quad \text{with} \quad L_i(x) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - t_j}{t_i - t_j} \,. \tag{5.0.9}$$

(5.3.a) ⚀ Show that

$$\sum_{i=0}^{n} L_i(t) = 1 \quad \forall t \in \mathbb{R} \,. \tag{5.0.10}$$

HIDDEN HINT 1 for (5.3.a) → 5.3.1:glgh1.pdf

SOLUTION for (5.3.a) → 5.3.1:glgs1.pdf ▲

(5.3.b) ⚁ Show that

$$\sum_{i=0}^{n} L_i(0) t_i^j = \begin{cases} 1 & \text{for } j = 0 \,, \\ 0 & \text{for } j = 1, \ldots, n \,. \end{cases} \tag{5.0.11}$$

(Read t_i riased to the power j.)

HIDDEN HINT 1 for (5.3.b) → 5.3.2:glgh1.pdf

SOLUTION for (5.3.b) → 5.3.2:glgs1.pdf ▲

(5.3.c) ⚂ Show that $p(x)$ can also be written as

$$p(x) = \omega(x) \sum_{i=0}^{n} \frac{y_i}{(x - t_i)\omega'(t_i)} \quad \text{with} \quad \omega(t) := \prod_{j=0}^{n} (t - t_j) \tag{5.0.12}$$

HIDDEN HINT 1 for (5.3.c) → 5.3.3:LagrangePoly1h.pdf

SOLUTION for (5.3.c) → 5.3.3:LagrangePoly1s.pdf ▲

End Problem 5.3

Problem 5.4: Generalized Lagrange polynomials for Hermite interpolation

➜Rem. 5.2.21 addressed a particular generalization of the Lagrange polynomial interpolation considered in ➜Section 5.2.2: Hermite interpolation, where beside the usual "nodal" interpolation conditions also the derivative of the interpolating polynomials is prescribed in the nodes, see ➜Eq. (5.2.22) (for $l_j = 1$ throughout). The associated generalized Lagrange polynomials were defined in ➜Def. 5.2.24 and in this problem we aim to find concrete formulas for them in the spirit of ➜Eq. (5.2.11).

Let $n+1$ different nodes t_i, $i = 0, \ldots, n$, $n \in \mathbb{N}$, be given. For numbers $y_j, c_j \in \mathbb{R}$, $j = 0, \ldots, n$, Hermite interpolation seeks a polynomial $p \in \mathcal{P}_{2n+1}$ satisfying $p(t_i) = y_i$ and $p'(t_i) = c_i$, $i = 0, \ldots, n$. Here p' designates the derivative of p.

The generalized Lagrange polynomials for Hermite interpolation $L_i, K_i \in \mathcal{P}_{2n+1}$ satisfy

$$L_i(t_j) = \delta_{ij} \,, \qquad L_i'(t_j) = 0 \,, \tag{5.0.13a}$$

$$K_i(t_j) = 0 \,, \qquad K_i'(t_j) = \delta_{ij} \,, \tag{5.0.13b}$$

for $i, j \in \{0, \ldots, n\}$.

(5.4.a) ⚀ Give explicitly the linear system of equations for the basis expansion coefficients of the Hermite interpolant with respect to the *monomial basis* of $\mathcal{P}_{2n+1}$.

HIDDEN HINT 1 for (5.4.a) → `5.4.1:gagh0.pdf`

SOLUTION for (5.4.a) → `5.4.1:glgs1.pdf` ▲

(5.4.b) ⚁ Give concrete formulas for K_0, K_1, L_0, L_1 for $n = 1$, $t_0 = -1$, $t_1 = 1$.

HIDDEN HINT 1 for (5.4.b) → `5.4.2:glgh1.pdf`

SOLUTION for (5.4.b) → `5.4.2:glgs1.pdf` ▲

(5.4.c) ⚁ Find the general formula describing all polynomials in t

1. of degree 2 that have a double zero in $t = a$, $a \in \mathbb{R}$,
2. of degree $n > 2$ that have a double zero in $t = a$, $a \in \mathbb{R}$.

A polynomial p is said to have a double zero in $t = a$, if $p(a) = 0$ *and* $p'(a) = 0$.

SOLUTION for (5.4.c) → `5.4.3:glgs1.pdf` ▲

(5.4.d) ⚁ Let $g_1, \ldots, g_m$, $m \in \mathbb{N}$, be functions in $C^1(I)$ (space of continuously differentiable functions). Find a formula for the derivative of $h(t) = \prod_{k=1}^{m} g(t)$.

SOLUTION for (5.4.d) → `5.4.4:glgs1.pdf`

▲

(5.4.e) ⚂ Find general formulas for the polynomials L_i, K_i as defined through (5.0.13). Of course, these formulas will involve the nodes t_i.

HIDDEN HINT 1 for (5.4.e) → `5.4.5:glgh1.pdf`

HIDDEN HINT 2 for (5.4.e) → `5.4.5:glghx.pdf`

SOLUTION for (5.4.e) → `5.4.5:glgs1.pdf`

▲

End Problem 5.4

Problem 5.5: Piecewise linear interpolation with knots different from nodes

In ➜Ex. 5.1.10 we examined piecewise linear interpolation in the case where the interpolation nodes coincide with the abscissas of the corners of the interpolating polygon. However, that one is a very special situation. In this problem we consider a more general setting for piecewise linear interpolation.

We are given two ordered sets of real numbers to be read as points on the real line:

(I) the knots $x_0 < x_1 < x_2 < \cdots < x_n$,

(II) the (interpolation) nodes $t_0 < t_1 < \cdots < t_n$, $n \in \mathbb{N}$, satisfying $x_0 \le t_j \le x_n$, $j = 0, \ldots, n$.

The space of piecewise linear continuous functions with respect to the knot set $\mathcal{N} := \{x_j\}_{j=0}^{n}$ is defined as:

$$\mathcal{S}_{1,\mathcal{N}} := \{s \in C^0([x_0, x_n]) : \ s(t) = \gamma_j t + \beta_j \text{ for } t \in]x_{j-1}, x_j], \ \gamma_j, \beta_j \in \mathbb{R} \ i = 1, \ldots, n\} \ . \tag{5.0.25}$$

Compare ➜Ex. 5.1.10 for the special case $x_j := t_j$.

On the function space $\mathcal{S}_{1,\mathcal{N}} \subset C^0(I)$, $I := [x_0, x_n]$, we consider the **interpolation problem**:

Find $s \in \mathcal{S}_{1,\mathcal{N}}$ such that $s(t_i) = y_i$, $i = 0,\ldots,n$, for given values $y_i \in \mathbb{R}$.

This fits the general framework of ➜§ 5.1.13.

Below we set $I_0 := [x_0, x_1]$, $I_j :=]x_{j-1}, x_{j+1}]$, $j = 1,\ldots,n-1$, $I_n := [x_{n-1}, x_n]$.

(5.5.a) ⚀ Show that the interpolation problem may not have a solution for some values y_j if a single interval $[x_j, x_{j+1}]$, $j = 0,\ldots,n-1$, contains more than 2 nodes t_i.

SOLUTION for (5.5.a) → 5.5.1:lips1.pdf ▲

(5.5.b) ⚁ Show that the interpolation problem can have a unique solution for any data values y_j *only if* each interval I_j, $j = 0,\ldots,n$, contains at least one node.

HIDDEN HINT 1 for (5.5.b) → 5.5.2:glgh1.pdf

SOLUTION for (5.5.b) → 5.5.2:lips2.pdf ▲

(5.5.c) ⚂ Show that the interpolation problem has a unique solution for all data values y_j if each of the closed intervals $]x_0, x_1[,]x_1, x_2[,]x_2, x_3[, \ldots,]x_{n-1}, x_n[$ contains at least one node t_j.

HIDDEN HINT 1 for (5.5.c) → 5.5.3:glgh1.pdf

SOLUTION for (5.5.c) → 5.5.3:lips3.pdf ▲

(5.5.d) ⚄ Implement a C++ function

```
VectorXd tentBasCoeff(const VectorXd &x,const VectorXd &t,
                      const VectorXd &y);
```

that returns the vector of values $s(x_j)$ of the interpolant s of the data points (t_i, y_i), $i = 0,\ldots,n$ in $\mathcal{S}_{1,\mathcal{N}}$ in the knots x_j, $j = 0,\ldots,n$. The argument vectors `x`, `t`, and `y` pass the x_j, t_j, and values y_j, respectively.

The function should first check whether the condition formulated in Sub-problem (5.5.c) is satisified. You must not take for granted that knots or nodes are sorted already.

HIDDEN HINT 1 for (5.5.d) → 5.5.4:glgh1.pdf

SOLUTION for (5.5.d) → 5.5.4:glgs1.pdf ▲

(5.5.e) ⚂ Implement a C++ class

```
class PwLinIP {
public:
  PwLinIP(const VectorXd &x, const VectorXd &t, const VectorXd &y);
  double operator()(double arg) const;
private:
  ...
};
```

that realizes an interpolator class in the spirit of ➜Rem. 5.1.19. The meanings of the arguments of the constructor are the same as for the function `tentBasCoeff` from Sub-problem (5.5.d). Pay attention to the efficient implementation of the evaluation operator!

HIDDEN HINT 1 for (5.5.e) → 5.5.5:lipx1.pdf

HIDDEN HINT 2 for (5.5.e) → 5.5.5:lipzz.pdf

SOLUTION for (5.5.e) → 5.5.5:dsfg1.pdf ▲

(5.5.f) ⚄ Implement a C++ code that creates plots of the cardinal basis functions for interpolation in $\mathcal{S}_{1,\mathcal{N}}$

- with know set $\mathcal{N} := \{0,1,2,3,\ldots,9,10\}$
- and for interpolation nodes $t_0 := 0$, $t_j = j - \frac{1}{2}$, $j = 1,\ldots,10$.

Recall that the cardinal basis function $b_k \in \mathcal{S}_{1,\mathcal{N}}$ is characterized by the conditions:

$$b_k(t_j) = \delta_{kj}\,, \quad k,j = 0,\ldots,10 \tag{5.0.30}$$

HIDDEN HINT 1 for (5.5.f) → `5.5.6:liphx.pdf`

SOLUTION for (5.5.f) → `5.5.6:glgs1.pdf` ▲

End Problem 5.5

Problem 5.6: Cardinal basis for trigonometric interpolation

In ➜Section 5.6 we learned about interpolation into the space $\mathcal{P}_{2n}^T$ or trigonometric polynomials from ➜Def. 5.6.3. In this task we investigate the cardinal basis functions for this interpolation operator and will also obtain a result about its stability.

The (ordered) real trigonometric basis of $\mathcal{P}_{2n}^T$ is

$$\begin{aligned}\mathcal{B}_{\mathrm{Re}} := \Big\{ & C_0 := t \mapsto 1, S_1 := t \mapsto \sin(2\pi t), C_1 := t \mapsto \cos(2\pi t), S_2 := t \mapsto \sin(4\pi t), \\ & C_2 t \mapsto \cos(4\pi t), \ldots, S_n : t \mapsto \sin(2n\pi t), C_n := t \mapsto \cos(2n\pi t) \Big\}\,.\end{aligned} \tag{5.0.32}$$

Another (ordered) basis of $\mathcal{P}_{2n}^T$ is

$$\mathcal{B}_{\exp} := \{B_k := t \mapsto \exp(2\pi k \imath t) : k = -n,\ldots,n\}\,. \tag{5.0.33}$$

Both bases have $2n+1$ elements, which agrees with the dimension of $\mathcal{P}_{2n}^T$ Cor. 5.6.8.

(5.6.a) ⚂ Give the matrix $\mathbf{S} \in \mathbb{C}^{2n+1,2n+1}$ that effects the transformation of a coefficient representation with respect to $\mathcal{B}_{\mathrm{Re}}$ into a coefficient representation with respect to $\mathcal{B}_{\exp}$.

HIDDEN HINT 1 for (5.6.a) → `5.6.1:trh1.pdf`

SOLUTION for (5.6.a) → `5.6.1:trisol1.pdf` ▲

(5.6.b) ⚂ The shift operator $S_\tau : C^0(\mathbb{R}) \to C^0(\mathbb{R})$ acts on a function according to $S_\tau f(t) = f(t-\tau)$, for $\tau \in \mathbb{R}$. If $\mathbf{c} \in \mathbb{C}^{2n+1}$ is the coefficient vector of $p \in \mathcal{P}_{2n}^T$ with respect to $\mathcal{B}_{\exp}$, what is the coefficient vector $\tilde{\mathbf{c}} \in \mathbb{C}^{2n+1}$ of $S_\tau p$?

HIDDEN HINT 1 for (5.6.b) → `5.6.2:tch2.pdf`

SOLUTION for (5.6.b) → `5.6.2:trisol2.pdf` ▲

Now we consider the set of interpolation nodes

$$\mathcal{T}_n := \left\{ \frac{j+1/2}{2n+1} : j = 0,\ldots,2n \right\}\,. \tag{5.0.34}$$

The cardinal basis functions $b_0,\ldots,b_{2n} \in \mathcal{P}_{2n}^T$ of $\mathcal{P}_{2n}^T$ with respect to $\mathcal{T}_n$ are defined by

$$b_j(t_k) = \delta_{kj} \quad \text{[Kronecker symbol]}\,, \quad j,k = 0,\ldots,2n \tag{5.0.35}$$

(5.6.c) ⚂ Use the function `trigpolyvalequid`, see ➜Code 5.6.21 (can be included from "`trigpolyvaleq`
to plot the cardinal basis function $t \mapsto b_0(t)$ in $[0,1]$ for $n = 5$. To that end evaluate $b_0(t)$ in 1000 equidistant points $\frac{k}{1000}$, $k = 0,\ldots,999$.

SOLUTION for (5.6.c) → `5.6.3:plotsol.pdf` ▲

(5.6.d) ⚂ Show that

$$b_{k+1} = \mathsf{S}_\delta b_k \quad \text{with} \quad \delta := \frac{1}{2n+1}\,. \tag{5.0.37}$$

HIDDEN HINT 1 for (5.6.d) → `5.6.4:h1.pdf`

SOLUTION for (5.6.d) → `5.6.4:trisol1.pdf` ▲

(5.6.e) ⚂ Describe the entries of the "interpolation matrix", that is, the system matrix of ➜Eq. (5.1.15), for the trigonometric interpolation problem with node set $\mathcal{T}_n$ and with respect to the basis $\mathcal{B}_{\exp}$ of $\mathcal{P}_{2n}^T$.

HIDDEN HINT 1 for (5.6.e) → `5.6.5:tklh1.pdf`

HIDDEN HINT 2 for (5.6.e) → `5.6.5:tmp.pdf`

SOLUTION for (5.6.e) → `5.6.5:trisol1.pdf` ▲

(5.6.f) ⚂ What is the inverse of the interpolation matrix found in the previous problem?

HIDDEN HINT 1 for (5.6.f) → `5.6.6:h1.pdf`

SOLUTION for (5.6.f) → `5.6.6:trisol1.pdf`

▲

(5.6.g) ⚃ Give a closed-form expression for the cardinal basis functions b_k defined in (5.0.35).

HIDDEN HINT 1 for (5.6.g) → `5.6.7:h1.pdf`

SOLUTION for (5.6.g) → `5.6.7:trisol1.pdf` ▲

(5.6.h) ⚃ Write a function

```
double trigIpL(std::size_t n);
```

that approximately computes the Lebesgue constant $\lambda(n)$, see ➜§ 5.2.69, for trigonometric interpolation based on the node set $\mathcal{T}_n$. The Lebesgue constant is available through the formula

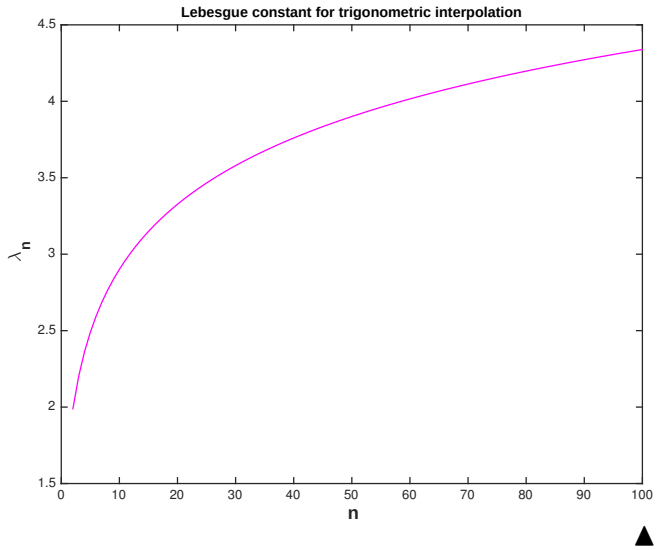
$$\lambda(n) = \sup_{\mathbf{y}\in\mathbb{C}^{2n+1}\setminus\{\mathbf{0}\}} \frac{\|\mathsf{I}_n\mathbf{y}\|_{L^\infty([0,1])}}{\|\mathbf{y}\|_\infty} = \sup_{t\in[0,1]} \sum_{k=0}^{2n} |b_k(t)|\,, \tag{5.0.40}$$

where $\mathsf{I}_n : \mathbb{C}^{2n+1} \to C^0(\mathbb{R})$ is the trigonometric interpolation operator.

Use sampling in 10^4 equidistant points to obtain a good guess for the supremum norm of a function on $[0,1]$. Maximum efficiency of the implementation need not be a concern. Use the formula involving sum of cosines.

SOLUTION for (5.6.h) → `5.6.8:trisol1.pdf`

Graph of $n \mapsto \lambda(n)$



▲

(5.6.i) ⚂ The following table gives the values $\lambda(n)$ for $n = 2^k$, $k = 2,3,\ldots,8$.

$^k k$	$\lambda(2^k)$
4	2.7
8	3.07
16	3.46
32	3.89
64	4.32
128	4.75
256	5.18
512	5.62

From these numbers predict the asymptotic behavior of $\lambda(n)$ for $n \to \infty$. Is it $O(\log n)$, $O(n)$, $O(n \log n)$, or $O(n^2)$?

SOLUTION for (5.6.i) → `5.6.9:xtrsol1.pdf` ▲

End Problem 5.6

Chapter 6

Approximation of Functions in 1D

Problem 6.1: Adaptive polynomial interpolation

In ➜Section 6.1.3 we have seen that the *a priori* placement of interpolation nodes is key to a good approximation by a polynomial interpolant. This problem deals with an *a posteriori* adaptive strategy that controls the placement of interpolation nodes depending on the interpolant. It employs a *greedy algorithm* to grow the node set based on an intermediate interpolant.
This strategy has recently gained prominence for a variety prominence for a wide range of approximation problems, see V. N. TEMLYAKOV, *Greedy approximation*, Acta Numer., 17 (2008), pp. 235–409.

[This problem involves implementation in C++]

A description of the greedy algorithm for adaptive polynomial interpolation is as follows:

Given a function $f : [a,b] \mapsto \mathbb{R}$ one starts $\mathcal{T}_0 := \{\frac{1}{2}(b+a)\}$. Based on a fixed finite set $\mathcal{S} \subset [a,b]$ of *sampling points* one augments the set of nodes according to

$$\mathcal{T}_{n+1} = \mathcal{T}_n \cup \left\{ \operatorname*{argmax}_{t\in\mathcal{S}} |f(t) - \mathrm{I}_{\mathcal{T}_n}(t)| \right\}, \tag{6.0.1}$$

where $\mathrm{I}_{\mathcal{T}_n}$ is the polynomial interpolation operator for the node set $\mathcal{T}_n$, until

$$\max_{t\in\mathcal{S}} |f(t) - \mathrm{I}_{\mathcal{T}_n}(t)| \leq \texttt{tol} \cdot \max_{t\in\mathcal{S}} |f(t)| \,. \tag{6.0.2}$$

(6.1.a) ⚂ Write a C++ function

```
template <class Function>
void adaptivepolyintp(const Function& f,double a, double b,
                      double tol, int N,
                      Eigen::VectorXd& adaptive_nodes);
```

that implements the algorithm described above.

The function arguments are: the function handle `f`, the interval bounds `a`, `b`, the relative tolerance `tol`, the number `N` of *equidistant* sampling points to compute the error (in the interval $[a,b]$), that is,

$$\mathcal{S} := \left\{ a + (b-a)\frac{j}{N},\ j = 0,\ldots,N \right\}.$$

and `t` will be used as return parameter for interpolation nodes (i.e. the final $\mathcal{T}_n$).

The type `Function` defines a function and is supposed to provide an operator `double operator()(doubl` A suitable lambda function can satisfy this requirement.

A function `intpolyval` (used in the lecture document) is provided and may be used (though it may not be the most efficient way to implement the function).

You might need to convert between the types `std::vector` and `Eigen::VectorXd`. This can be done as follows using EIGEN's **Map** class:

For `Eigen::VectorXd` to `std::vector<double>`, use:

```
Eigen::VectorXd v;
std::vector<double> to_std(v.data(), v.data() + v.size());
```

and the other way round (using `Eigen::Map`):

```
std::vector<double> v;
Eigen::Map<Eigen::VectorXd> to_eigen(v.data(), v.size());
```

SOLUTION for (6.1.a) → `6.1.1:solfile.pdf`

▲

(6.1.b) ⚀ Extend the function from the previous sub-problem so that it reports the quantity

$$\epsilon_n := \max_{t\in\mathcal{S}} |f(t) - \mathrm{T}_{\mathcal{T}_n}(t)| \tag{6.0.3}$$

for each intermediate set $\mathcal{T}_n$.

SOLUTION for (6.1.b) → `6.1.2:solfile.pdf`

▲

(6.1.c) ⚀ For $f_1(t) := \sin(e^{2t})$ and $f_2(t) = \frac{\sqrt{t}}{1+16t^2}$ plot ϵ_n versus n (the number of interpolation nodes). Choose plotting styles that reveal the qualitative decay (types of convergence as given in ➜Def. 6.1.38) of this error norm as the number of interpolation nodes is increased. Use interval $[a,b] = [0,1]$, `N=1000` sampling points, tolerance `tol = 1e-6`.

SOLUTION for (6.1.c) → `6.1.3:solfile.pdf`

▲

End Problem 6.1

Problem 6.2: Piecewise Cubic Hermite Interpolation

In this problem, we will consider (piecewise) cubic Hermite interpolation, which is a (local) generalized polynomial interpolation scheme ➜Rem. 5.2.21.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We consider the cubic Hermite interpolation operator $\mathcal{H}$ acting on continuously differentiable function defined on an interval $[a,b]$ and mapping into the space $\mathcal{P}_3$ of polynomials of degree at most 3r:

$$\mathcal{H} : C^1([a,b]) \to \mathcal{P}_3 .$$

It is defined by the generalized interpolation conditions, see also ➜Def. 5.4.1,

$$(\mathcal{H}f)(a) = f(a) \quad , \quad (\mathcal{H}f)(b) = f(b) \, , \tag{6.0.6}$$
$$(\mathcal{H}f)'(a) = f'(a) \quad , \quad (\mathcal{H}f)'(b) = f'(b) \, . \tag{6.0.7}$$

(6.2.a) ⚀ Assume $f \in C^4([a,b])$. Show that for every $x \in]a,b[$ there exists a $\tau \in [a,b]$ such that:

$$(f - \mathcal{H}f)(x) = \frac{1}{24} f^{(4)}(\tau)(x-a)^2(x-b)^2 \, . \tag{6.0.8}$$

HIDDEN HINT 1 for (6.2.a) → 6.2.1:pchh.pdf

SOLUTION for (6.2.a) → 6.2.1:errpr.pdf ▲

Now, we consider a *piecewise cubic Hermite interpolation* with exact slopes on a mesh

$$\mathcal{M} := \{a = x_0 < x_1 < \cdots < x_n = b\}$$

as defined in ➜Section 6.5.2, ➜Def. 6.5.14. By slopes we mean the values of the derivative of the interpolant imposed at the nodes of the mesh $\mathcal{M}$.

(6.2.b) ⚁ Based on (6.0.8) predict the rate of algebraic convergence of the maximum norm of the interpolation error of piecewise cubic Hermite interpolation of a function $f \in C^4([a,b])$ in terms of the mesh width $h_{\mathcal{M}}$ of $\mathcal{M}$ as $h_{\mathcal{M}} \to 0$ (h-convergence).

HIDDEN HINT 1 for (6.2.b) → 6.2.2:pch0.pdf

SOLUTION for (6.2.b) → 6.2.2:pchol2.pdf ▲

(6.2.c) ⚁ Now we consider cases, where perturbed or reconstructed slopes are used. For instance, this was done in the context of monotonicity preserving piecewise cubic Hermite interpolation as discussed in ➜Section 5.4.2.

Assume that piecewise cubic Hermite interpolation is based on perturbed slopes, that is, the piecewise cubic function s on $\mathcal{M}$ satisfies:

$$s(x_j) = f(x_j) \quad , \quad s'(x_j) = f'(x_j) + \delta_j ,$$

where the δ_j may depends on $\mathcal{M}$, too.

Which rate of asymptotic h-convergence of the sup-norm of the approximation error can be expected, if we know that for all j:

$$|\delta_j| = O(h^\beta) \, , \quad \beta \in \mathbb{N}_0$$

for mesh-width $h \to 0$.

HIDDEN HINT 1 for (6.2.c) → 6.2.3:pchhx.pdf

SOLUTION for (6.2.c) → 6.2.3:pchi1.pdf

▲

(6.2.d) ⚂ Consider the following class declaration that implements a piecewise cubic Hermite interpolant and provides a corresponding point evaluation operator **operator** `()`:

C++11-code 6.0.10: Declaration of the class `PCHI`.

```
/*!
 * \brief Implements a piecewise cubic Hermite interpolation.
 * Uses equidistant meshes and various methods of slope
    reconstruction.
 *
 */
class PCHI {
public:
    /*!
     *! \brief Construct the slopes from the data.
     *! Use finite-differences or setting s'(x_j) = 0.
     *! \param[in] t Vector of nodes (assumed equidistant and
        sorted).
     *! \param[in] y Vector of values at nodes t.
     *! \param[in] s Flag to set if you want to reconstruct or set
        the slopes to zero.
     */
    PCHI(const VectorXd & t,
         const VectorXd & y,
         Slope s = Slope::Reconstructed);

    /*!
     *! \brief Evaluate the intepolant at the nodes x.
     *! The input is assumed sorted, unique and inside the interval
     *! \param[in] x The vector of points x_i where to compute s(x_i).
     *! \return Values of interpolant at x (vector)
     */
    VectorXd operator() (const VectorXd & x) const;

private:
    // Provided nodes and values (t,y) to compute spline,
    // Eigen vectors, c contains slopes
    // All have the same size n
    VectorXd t, y, c;
    // Difference t(i) - t(i-1)
    double h;
    // Size of t, y and c.
    int n;
};
```

Get it on GitLab (pchi.cpp).

The `enum` type `Slope` will be used later to specify the slope reconstruction for the interpolant.

Give an efficient implementation the evaluation operator.

SOLUTION for (6.2.d) → `6.2.4:pchi2.pdf`

▲

(6.2.e) ⚀ Implement a strange piecewise cubic interpolation scheme in C++ that satisfies:

$$s(x_j) = f(x_j) \quad , \quad s'(x_j) = 0$$

and empirically determine its convergence on a sequence of equidistant meshes of $[-5,5]$ with mesh-widths $h = 2^{-l}, l = 0,\ldots,8$ and for the interpoland $f(t) := \frac{1}{1+t^2}$.

To that end, implement the correct reconstruction of the slope `c` in the constructor of the class `PCHI`, when the value of `s` is `Slope::Zero`.

Compare with the insight gained in (6.2.c).

SOLUTION for (6.2.e) → `6.2.5:pchi2.pdf`

▲

(6.2.f) ⚁ Assume equidistant meshes and reconstruction of slopes by a particular averaging. More precisely, the $\mathcal{M}$-piecewise cubic function s is to satisfy the generalized interpolation conditions

$$\begin{aligned} s(x_j) &= f(x_j), \\ s'(x_j) &= \begin{cases} \frac{-f(x_2)+4f(x_1)-3f(x_0)}{2h} & \text{for } j = 0\,, \\ \frac{f(x_{j+1})-f(x_{j-1})}{2h} & \text{for } j = 1,\ldots,n-1\,, \\ \frac{3f(x_n)-4f(x_{n-1})+f(x_{n-2})}{2h} & \text{for } j = n\,. \end{cases} \end{aligned} \tag{6.0.13}$$

What will be the rate of h-convergence of this scheme (in sup-norm)?

You can use what you have found in (6.2.c). To find perturbation bounds, rely on the Taylor expansion formula with remainder, see ➜Ex. 1.5.65.

SOLUTION for (6.2.f) → `6.2.6:pchi3.pdf`

▲

(6.2.g) ⚀ Implement the correct reconstruction of the slope `c` defined in Sub-problem (6.2.f), (6.0.13) in the constructor of the class `PCHI`, when the value of `s` is `Slope::Reconstructed`.

SOLUTION for (6.2.g) → `6.2.7:pchi4.pdf`

▲

(6.2.h) ⚀ In a numerical experiment determine qualitatively and quantitatively the h-convergence of the maximum norm of the interpolation error for piecewise cubic Hermite interpolation of a function $f \in C^4([a,b])$ based on the slopes from Sub-problem (6.2.f) as given in (6.0.13).

As test case use the setting of Sub-problem (6.2.e) and make sure that your observation matches what you have found in Sub-problem (6.2.f).

SOLUTION for (6.2.h) → `6.2.8:xtrsol1.pdf` ▲

End Problem 6.2

Problem 6.3: Piecewise linear approximation on graded meshes

One of the main point made in ➜Section 6.1.3 is that the quality of an interpolant depends heavily on the choice of the interpolation nodes. If the function to be interpolated has a "bad behavior" in a small part of the domain, for instance it has very large derivatives of high order, more interpolation points are required in that area of the domain. Commonly used tools to cope with this task are *graded meshes*, which will be the topic of this problem.

Template: Get it on GitLab.
Solution: Get it on GitLab.

Given a mesh $\mathcal{T} = \{0 \le t_0 < t_1 < \cdots < t_n \le 1\}$ on the unit interval $I = [0,1]$, $n \in \mathbb{N}$, we define the *piecewise linear interpolant*:

$$\mathsf{I}_\mathcal{T} : C^0(I) \to \mathcal{P}_{1,\mathcal{T}} = \{s \in C^0(I),\ s_{|[t_{j-1},t_j]} \in \mathcal{P}_1\ \forall\, j\}, \quad \text{s.t.} \quad (\mathsf{I}_\mathcal{T} f)(t_j) = f(t_j), \quad j = 0,\ldots,n \tag{6.0.16}$$

See also ➜Section 5.3.2.

(6.3.a) ⚀ If we choose the uniform mesh $\mathcal{T} = \{t_j\}_{j=0}^n$ with $t_j = j/n$, given a function $f \in C^2(I)$ what is the asymptotic behavior of the error $\|f - \mathsf{I}_\mathcal{T} f\|_{L^\infty(I)}$ when $n \to \infty$?

HIDDEN HINT 1 for (6.3.a) → `6.3.1:GradedMeshes1h.pdf`

SOLUTION for (6.3.a) → `6.3.1:GradedMeshes1s.pdf` ▲

(6.3.b) ⚁ What is the regularity of the function $f : I \to \mathbb{R},\ f(t) = t^\alpha,\ 0 < \alpha < 2$? In other words, for which $k \in \mathbb{N}$ do we have $f \in C^k(I)$?

HIDDEN HINT 1 for (6.3.b) → `6.3.2:GradedMeshes2h.pdf`

SOLUTION for (6.3.b) → `6.3.2:GradedMeshes2s.pdf` ▲

(6.3.c) ⚂ Study numerically the h-convergence of the piecewise linear approximation of $f(t) = t^\alpha$ ($0 < \alpha < 2$) on uniform meshes. Determine the order of convergence using 1D linear regression, see ➜Rem. 6.1.40 which is implemented, as a special case, in the auxiliary function `polyfit`, see ➜Code 5.7.20.

SOLUTION for (6.3.c) → `6.3.3:GradedMeshes3s.pdf` ▲

(6.3.d) ⚂ In which mesh interval do you expect $|f - \mathsf{I}_\mathcal{T} f|$ to attain its maximum?

HIDDEN HINT 1 for (6.3.d) → `6.3.4:GradedMeshes4ha.pdf`

HIDDEN HINT 2 for (6.3.d) → `6.3.4:GradedMeshes4hb.pdf`

SOLUTION for (6.3.d) → `6.3.4:GradedMeshes4s.pdf` ▲

(6.3.e) ⚂ Compute by hand the exact value of $\|f - \mathsf{I}_\mathcal{T} f\|_{L^\infty(I)}$.

Compare the order of convergence obtained with the one observed numerically in (6.3.b).

HIDDEN HINT 1 for (6.3.e) → `6.3.5:GradedMeshes5h.pdf`

SOLUTION for (6.3.e) → `6.3.5:GradedMeshes5s.pdf` ▲

(6.3.f) ⚂ Since the interpolation error is concentrated in the left part of the domain, it seems reasonable to use a finer mesh only in this part. A common choice is an *algebraically graded mesh*, defined as $\mathcal{G} = \left\{t_j = \left(\frac{j}{n}\right)^\beta,\quad j = 0,\ldots,n\right\}$ for a parameter $\beta > 1$. An example is depicted in Fig. 26 for $\beta = 2$.

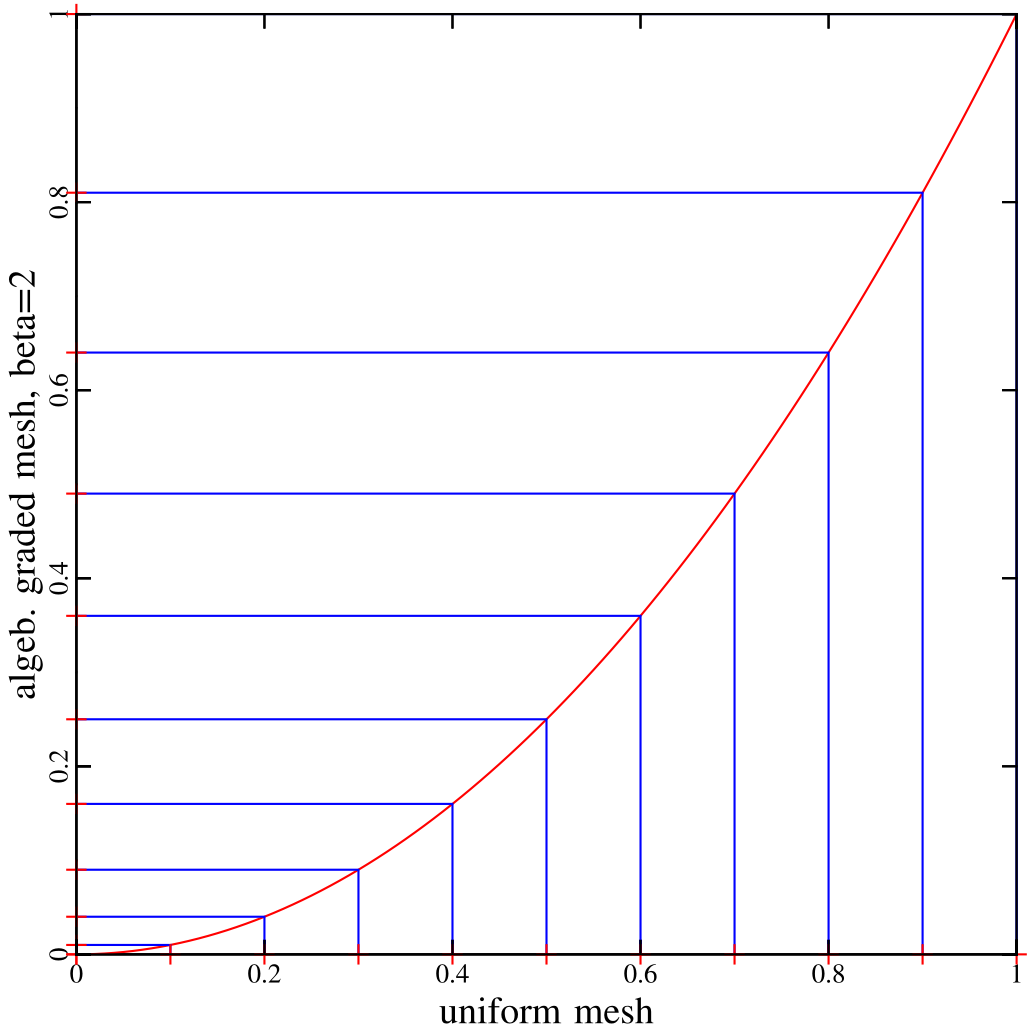


Fig. 26

For a fixed parameter α in the definition of f, numerically determine the rate of convergence of the piecewise linear interpolant $I_{\mathcal{G}}$ on the graded mesh $\mathcal{G}$ as a function of the parameter β. Try for instance $\alpha = 1/2$, $\alpha = 3/4$ or $\alpha = 4/3$.

How do you have to choose β in order to recover the optimal rate $\mathcal{O}(n^{-2})$ (if possible)?

SOLUTION for (6.3.f) → `6.3.6:GradedMeshes6s.pdf` ▲

End Problem 6.3

Problem 6.4: Chebyshev polynomials and their properties

Chebyshev polynomials as defined in ➜Def. 6.1.76 are a sequence of orthogonal polynomials, which can be defined recursively, see ➜Thm. 6.1.77. In this problem we will examine these polynomials and a few of their many properties.

Template: Get it on GitLab.
Solution: Get it on GitLab.

Let $T_n \in \mathcal{P}_n$ be the n-th Chebyshev polynomial, as defined in ➜Def. 6.1.76 and $\xi_0^{(n)}, \ldots, \xi_{n-1}^{(n)}$ be the n

zeros of T_n. According to ➜Eq. (6.1.84), these are given by:

$$\xi_j^{(n)} = \cos\left(\frac{2j+1}{2n}\pi\right), \quad j = 0,\ldots,n-1 \tag{6.0.22}$$

We define the family of discrete L^2 semi-inner products (i.e. not conjugate symmetric), *cf.* ➜Eq. (6.2.22):

$$(f,g)_n := \sum_{j=0}^{n-1} f(\xi_j^{(n)})g(\xi_j^{(n)}), \quad f,g \in C^0([-1,1]) \tag{6.0.23}$$

We also define the special weighted L^2 semi-inner product:

$$(f,g)_w := \int_{-1}^{1} \frac{1}{\sqrt{1-t^2}} f(t)g(t)\,dt \quad f,g \in C^0([-1,1]) \tag{6.0.24}$$

(6.4.a) ⚀ Show that the Chebyshev polynomials are an orthogonal family of polynomials with respect to the inner product defined in Eq. (6.0.24) according to ➜Def. 6.2.25, namely $(T_k, T_l)_w = 0$ for every $k \neq l$.

HIDDEN HINT 1 for (6.4.a) → 6.4.1:ChebPolyProperties1h.pdf

SOLUTION for (6.4.a) → 6.4.1:ChebPolyProperties1s.pdf ▲

Consider the following statement:

Theorem 6.0.25.

The family of polynomials $\{T_0,\ldots,T_n\}$ is an orthogonal basis (➜Def. 6.2.14) of $\mathcal{P}_n$ with respect to the inner product $(\ ,\ )_{n+1}$ defined in Eq. (6.0.23).

(6.4.b) ⚀ Write a C++ code to test the assertion of Thm. 6.0.25.

HIDDEN HINT 1 for (6.4.b) → 6.4.2:ChebPolyProperties2h.pdf

SOLUTION for (6.4.b) → 6.4.2:ChebPolyProperties2s.pdf ▲

(6.4.c) ⚁ Prove Thm. 6.0.25.

HIDDEN HINT 1 for (6.4.c) → 6.4.3:ChebPolyProperties3h.pdf

SOLUTION for (6.4.c) → 6.4.3:ChebPolyProperties3s.pdf ▲

(6.4.d) ⚀ Given a function $f \in C^0([-1,1])$, find an expression for the best approximant $q_n \in \mathcal{P}_n$ of f in the discrete L^2-norm:

$$q_n = \operatorname*{argmin}_{p\in\mathcal{P}_n} |f-p|_{n+1}\,,$$

$|\ |_{n+1}$ is the norm induced by the scalar product $(\ ,\ )_{n+1}$. You should represent q_n through an expansion in Chebychev polynomials of the form:

$$q_n = \sum_{j=0}^{n} \alpha_j T_j\,, \tag{6.0.33}$$

for suitable coefficients $\alpha_j \in \mathbb{R}$, see also ➜Eq. (6.1.101).

HIDDEN HINT 1 for (6.4.d) → 6.4.4:ChebPolyProperties4h.pdf

SOLUTION for (6.4.d) → 6.4.4:ChebPolyProperties4s.pdf ▲

(6.4.e) ⚀ Write a C++ function that returns the vector of coefficients $(\alpha_j)_j$ in Eq. (6.0.33) given a function f:

```
template <typename Function>
void bestpolchebnodes(const Function &f, Eigen::VectorXd
   &alpha)
```

Note that the degree of the polynomial is indirectly passed with the length of the output `alpha`. The input `f` is a lambda-function, e.g.:

```
auto f = [] (double & x) {return 1/(pow(5*x,2)+1);};
```

SOLUTION for (6.4.e) → `6.4.5:ChebPolyProperties5s.pdf` ▲

(6.4.f) ⚀ Test `bestpolchebnodes` with the function $f(x) = \frac{1}{(5x)^2+1}$ and $n = 20$. Approximate the supremum norm of the approximation error by sampling on an equidistant grid with 10^6 points.

HIDDEN HINT 1 for (6.4.f) → `6.4.6:ChebPolyProperties6h.pdf`

SOLUTION for (6.4.f) → `6.4.6:ChebPolyProperties6s.pdf` ▲

(6.4.g) ⚀ Let L_j, $j = 0,\ldots,n$, be the Lagrange polynomials associated with the nodes $t_j = \xi_j^{(n+1)}$ of the Chebyshev interpolation with $n+1$ nodes on $[-1,1]$ (see ➜Eq. (6.1.84)). Show that:

$$L_j = \frac{1}{n+1} + \frac{2}{n+1}\sum_{l=1}^{n} T_l(\xi_j^{(n+1)})T_l$$

HIDDEN HINT 1 for (6.4.g) → `6.4.7:ChebPolyProperties7h.pdf`

SOLUTION for (6.4.g) → `6.4.7:ChebPolyProperties7s.pdf` ▲

End Problem 6.4

Chapter 7

Numerical Quadrature

Problem 7.1: Zeros of orthogonal polynomials

This problem combines elementary methods for finding zeros from ➜Section 8.3 and 3-term recursions satisfied by orthogonal polynomials, *cf.* ➜Thm. 6.2.32. This will permit us to find the zeros of Legendre polynomials, the so-called Gauss nodes (see ➜Def. 7.3.29).

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

The zeros of the Legendre polynomial P_n (see Def. 7.3.27) are the n Gauss points ξ_j^n, $j = 1, \ldots, n$. In this problem we compute the Gauss points by zero-finding methods applied to P_n. The 3-term recursion ➜Eq. (7.3.33) for Legendre polynomials will play an essential role. Moreover, recall that, by definition, the Legendre polynomials are $L^2(]-1,1[)$-orthogonal.

(7.1.a) ⚄ Prove the following interleaving property of the zeros of the Legendre polynomials. For all $n \in \mathbb{N}_0$ we have:

$$-1 < \xi_j^n < \xi_j^{n-1} < \xi_{j+1}^n < 1, \qquad j = 1, \ldots, n-1$$

HIDDEN HINT 1 for (7.1.a) → `7.1.1:ZerosLegendre1h.pdf`

SOLUTION for (7.1.a) → `7.1.1:ZerosLegendre1s.pdf`

▲

(7.1.b) ⚀ By differentiating ➜Eq. (7.3.33) derive a combined 3-term recursion for the sequences $(P_n)_n$ and $(P_n')_n$.

SOLUTION for (7.1.b) → `7.1.2:ZerosLegendre2s.pdf`

▲

(7.1.c) ⚄ Use the recursions obtained in (7.1.b) to write a C++ function that fills the matrices `Lx` and `DLx` in $\mathbb{R}^{N\times(n+1)}$ with the values $\{P_k(x_j)\}_{jk}$ and $\{P_k'(x_j)\}_{jk}$, $j = 0, \ldots, N-1$ and $k = 0, \ldots, n$, for an input vector $x \in \mathbb{R}^N$ (passed in `x`):

```
void legvals(const Eigen::VectorXd& x, Eigen::MatrixXd& Lx,
  Eigen::MatrixXd& DLx)
```

SOLUTION for (7.1.c) → 7.1.3:ZerosLegendre3s.pdf

▲

(7.1.d) ⚀ We can compute the zeros of P_k, $k = 1,\ldots,n$, by means of the secant rule (see § 8.3.22) using the endpoints $\{-1,1\}$ of the interval and the zeros of the previous Legendre polynomial as initial guesses; see (7.1.a). We opt for a correction-based termination criterion (see Section 8.1.2) based on prescribed relative and absolute tolerance (see Code 8.3.25).

Write a C++ function that computes the Gauss points $\zeta_j^k \in [-1,1]$, $j = 1,\ldots,k$ and $k = 1,\ldots,n$, using the zero finding approach outlined above:

```
Eigen::MatrixXd gaussPts(const int n, const double rtol=1e-10,
   const double atol=1e-12)
```

The Gauss points should be returned in an upper triangular $n \times n$-matrix.

HIDDEN HINT 1 for (7.1.d) → 7.1.4:ZerosLegendre4h.pdf

SOLUTION for (7.1.d) → 7.1.4:ZerosLegendre4s.pdf

▲

(7.1.e) ⚀ Validate your implementation of the function `gaussPts` with $n = 8$ by computing the values of the Legendre polynomials in the zeros obtained (use the function `legvals`). Explain the failure of the method.

HIDDEN HINT 1 for (7.1.e) → 7.1.5:ZerosLegendre5h.pdf

SOLUTION for (7.1.e) → 7.1.5:ZerosLegendre5s.pdf

▲

(7.1.f) ⚀ Fix your function `gaussPts` taking into account the above considerations. You should use the *regula falsi*, that is a variant of the secant method in which, at each step, we choose the old iterate to keep depending on the signs of the function. More precisely, given two approximations $x^{(k)}$, $x^{(k-1)}$ of a zero in which the function f has different signs, compute another approximation $x^{(k+1)}$ as zero of the secant. Use this as the next iterate, but then chose as $x^{(k)}$ the value $z \in \{x^{(k)}, x^{(k-1)}\}$, for which $\operatorname{sign}\left[f\left(x^{(k+1)}\right)\right] \neq \operatorname{sign}[f(z)]$. This ensures that f has always a different sign in the last two iterates.

The regula falsi variation of the secant method can be easily implemented with a little modification, as done in the MATLAB code `secant_falsi.m` on GitLab.

SOLUTION for (7.1.f) → 7.1.6:ZerosLegendre6s.pdf ▲

End Problem 7.1

Problem 7.2: Efficient quadrature of singular integrands

This problem deals with efficient numerical quadrature of non-smooth integrands with a special structure. Before you tackle this problem, read about regularization of integrands by transformation, cf. Rem. 7.3.46. For other problems on the same topic see Problem 7.8 and Problem 7.3.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

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

$$W(f) := \int_{-1}^{1} \sqrt{1-t^2}\, f(t)dt, \tag{7.0.8}$$

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

(7.2.a) ⊡ The function

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

(contained in `gauleg.hpp`) returns a structure `QuadRule` containing nodes (x_j) and weights (w_j) of a Gauss-Legendre quadrature (Def. 7.3.29) on $[-1,1]$ with n nodes. ▲

(7.2.b) ⊡ Study § 7.3.38 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 for (7.2.b) → `7.2.2:effs1.pdf`

▲

(7.2.c) ⚀ 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 $n \to \infty$, the error of your implementation has asymptotic exponential convergence to zero.

HIDDEN HINT 1 for (7.2.c) → `7.2.3:h21.pdf`

HIDDEN HINT 2 for (7.2.c) → `7.2.3:h23.pdf`

SOLUTION for (7.2.c) → `7.2.3:effs2.pdf` ▲

(7.2.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 for (7.2.d) → `7.2.4:effs3.pdf` ▲

(7.2.e) 2 Tabulate the quadrature error:

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

for $f(t) := \frac{1}{2+\exp(3t)}$ and $n = 1,2,...,25$. Then describe the type of convergence observed, see ➜Def. 6.1.38.

SOLUTION for (7.2.e) → `7.2.5:effs5.pdf` ▲

End Problem 7.2

Problem 7.3: Smooth integrand by transformation

In ➜Rem. 7.3.46 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. Related problems are Problem 7.2 and Problem 7.8.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

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

$$I := \int_{-1}^{1} \arcsin(t)\, f(t)\, \mathrm{d}t. \tag{7.0.11}$$

We want to approximate this integral using global Gauss quadrature. The nodes (vector `x`) and the weights (vector `w`) of an n-point Gaussian quadrature on $[-1,1]$ can be computed using the provided C++ routine `gaussquad` (found in `gaussquad.hpp`).

We have the following function:

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

`QuadRule` is a `struct` containing the quadrature weights `w` and nodes `x`:

```
struct QuadRule {
  Eigen::VectorXd nodes, weight;
};
```

(7.3.a) ⚀ Write a C++ routine

```
template <class Function>
void gaussConv(const Function & f);
```

that produces an appropriate convergence plot of the quadrature error versus the number $n = 1,\ldots,50$ of quadrature points. Here `f` is a handle to the function f.

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

HIDDEN HINT 1 for (7.3.a) → `7.3.1:hcv.pdf`

SOLUTION for (7.3.a) → `7.3.1:gauq1.pdf` ▲

(7.3.b) ⚁ Describe qualitatively and quantitatively the asymptotic (for $n \to \infty$) convergence of the quadrature error you observe in (7.3.a).

HIDDEN HINT 1 for (7.3.b) → `7.3.2:h1.pdf`

SOLUTION for (7.3.b) → `7.3.2:gauq2.pdf` ▲

(7.3.c) ⚂ Transform the previous integral into an equivalent one, with a suitable change of variable, so that the Gauss quadrature applied to the transformed integral can be expected to converge exponentially if $f \in C^\infty([-1,1])$.

Use the transformation $t = \sin(x)$.

SOLUTION for (7.3.c) → `7.3.3:gauq3.pdf`

▲

(7.3.d) ⚁ Now, write a C++ function

```
template <class Function>
void gaussConvCV(const Function & f);
```

which plots the quadrature error versus the number $n = 1, \ldots, 50$ of quadrature points for the integral obtained in the previous subtask.

Again, as in Sub-problem (7.3.a), choose $f(t) = \sinh(t)$ and save your convergence plot as `GaussConvCV.ep`

SOLUTION for (7.3.d) → `7.3.4:gauq4.pdf`

▲

(7.3.e) ⚁ Similar question as in Sub-problem (7.3.b): describe qualitatively the asymptotic (for $n \to \infty$) convergence of the quadrature error you observe in Sub-problem (7.3.d).

SOLUTION for (7.3.e) → `7.3.5:gauq2.pdf` ▲

(7.3.f) ⚁ Explain the difference between the results obtained in Sub-problem (7.3.a) and Sub-problem (7.3.d).

SOLUTION for (7.3.f) → `7.3.6:gauq5.pdf` ▲

End Problem 7.3

Problem 7.4: Generalize "Hermite-type" quadrature formula

In this exercise we will construct a new quadrature formula and then use it in an application.
You may want to have a look at the methods used in ➜Ex. 7.3.13, as they will come in handy.

(7.4.a) ⚂ Determine $A, B, C, x_1 \in \mathbb{R}$ such that the quadrature formula

$$\int_0^1 f(x)dx \approx Af(0) + Bf'(0) + Cf(x_1) \tag{7.0.18}$$

is exact for polynomials of highest possible degree.

SOLUTION for (7.4.a) → `7.4.1:herm1s.pdf` ▲

(7.4.b) ⚂ Compute an approximation of $z(2)$ using the quadrature rule of Eq. (7.0.18), where the function z is defined as the solution of the initial value problem

$$z'(t) = \frac{t}{1+t^2} \quad , \quad z(1) = 1\,. \tag{7.0.19}$$

HIDDEN HINT 1 for (7.4.b) → `7.4.2:herm2h.pdf`

SOLUTION for (7.4.b) → `7.4.2:herm2s.pdf` ▲

End Problem 7.4

Problem 7.5: Numerical integration of improper integrals

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 $f : \mathbb{R} \to \mathbb{R}$ that decay sufficiently fast for $|t| \to \infty$ (such that they are integrable on $\mathbb{R}$).

[This problem involves implementation in C++]

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) on $[-b,b]$.

(7.5.a) ⚀ For the integrand $g(t) := 1/(1+t^2)$ determine b such that the truncation error E_T satisfies:

$$E_T := \left| \int_{-\infty}^{\infty} g(t)dt - \int_{-b}^{b} g(t)dt \right| \leq 10^{-6} \tag{7.0.22}$$

SOLUTION for (7.5.a) → `7.5.1:op1sol.pdf` ▲

(7.5.b) ⚀ What is the algorithmic difficulty faced in the implementation of the truncation approach for a generic integrand?

SOLUTION for (7.5.b) → `7.5.2:op1dif.pdf` ▲

A second option is the transformation of the improper integral to a bounded domain by substitution. For instance, we may use the map $t = \cot(s)$.

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

SOLUTION for (7.5.c) → `7.5.3:op2sub.pdf` ▲

(7.5.d) ⚀ Write down the transformed integral explicitly for $g(t) := \frac{1}{1+t^2}$. Simplify the integrand.

HIDDEN HINT 1 for (7.5.d) → `7.5.4:h1ti.pdf`

SOLUTION for (7.5.d) → `7.5.4:op2com.pdf` ▲

(7.5.e) ⚀ 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, const Function & f);
```

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

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

HIDDEN HINT 1 for (7.5.e) → `7.5.5:hlf1.pdf`

SOLUTION for (7.5.e) → `7.5.5:op2imp.pdf` ▲

(7.5.f) ⚀ Study the convergence for $n \to \infty$ of the quadrature method implemented in the previous subproblem for the integrand $h(t) := \exp(-(t-1)^2)$ (shifted Gaussian). What kind of convergence do you observe? For the error you can use that $\int_{-\infty}^{\infty} h(t)dt = \sqrt{\pi}$.

SOLUTION for (7.5.f) → `7.5.6:testsol.pdf` ▲

End Problem 7.5

Problem 7.6: Nested numerical quadrature

This problem addresses numerical quadrature over a two-dimensional domain. It turns out that this problem can be reduced to two one-dimensional integrals, which are amenable to the techniques covered in ➜Chapter 7.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

A laser beam has intensity

$$I(x,y) := \exp(-\alpha((x-p)^2+(y-q)^2)) \,, \quad x,y \in \mathbb{R}$$

on the plane orthogonal to the direction of the beam.

(7.6.a) ⚀ Write down the radiant power absorbed by the triangle

$$\Delta := \{(x,y)^T \in \mathbb{R}^2 \mid x \geq 0,\, y \geq 0,\, x+y \leq 1\}$$

as a double integral.

HIDDEN HINT 1 for (7.6.a) → `7.6.1:neq1h.pdf`

SOLUTION for (7.6.a) → `7.6.1:neq1s.pdf`

▲

(7.6.b) ⚁ Write a C++ function

```
template <class Function>
double evalgaussquad(const double a, const double b,
                     const Function & f, const QuadRule & Q);
```

that evaluates the n-point Gaussian quadrature as introduced in ➜Section 7.3 for an integrand passed in `f` on domain $[a,b]$. It should rely on the quadrature rule on the reference interval $[-1,1]$ that is supplied through an object of type `QuadRule`:

```
struct QuadRule { Eigen::VectorXd nodes, weight; };
```

(The vectors `weights` and `nodes` denote the weights and nodes of the reference quadrature rule, respectively.)

Use the function `gaussquad` from `gaussquad.hpp` to compute weights and nodes in $[-1,1]$.

SOLUTION for (7.6.b) → `7.6.2:neq2s.pdf` ▲

(7.6.c) ⚂ Write a C++ function

```
template <class Function>
double gaussquadtriangle(const Function & f, const unsigned N)
```

for the computation of the integral

$$\int_{\Delta} f(x,y)\, dx\, dy \tag{7.0.28}$$

using nested n-point, 1D Gauss quadratures (using the function `evalgaussquad` of (7.6.b)).

HIDDEN HINT 1 for (7.6.c) → `7.6.3:neq3h1.pdf`

HIDDEN HINT 2 for (7.6.c) → `7.6.3:neq3h2.pdf`

SOLUTION for (7.6.c) → `7.6.3:neq3s.pdf` ▲

(7.6.d) ⊡ Apply the function `gaussquadtriangle` of (7.6.c) using the parameters $\alpha = 1$, $p = 0$, $q = 0$. Compute the error w.r.t. to the number of nodes n using the "exact" value of the integral $I \approx 0.366046550000405$. What kind of convergence do you observe and why?

HIDDEN HINT 1 for (7.6.d) → `7.6.4:neq4h.pdf`

SOLUTION for (7.6.d) → `7.6.4:neq4s.pdf`

▲

End Problem 7.6

Problem 7.7: Quadrature plots

In this problem, we will try and deduce the type of quadrature and the integrand from an error plot.

We consider three different functions on the interval $I = [0,1]$:

$$\begin{aligned} &\text{function A:} \quad f_A \in C^\infty(I)\,, \quad f_A \notin \mathcal{P}_k \;\forall\, k \in \mathbb{N}\,; \\ &\text{function B:} \quad f_B \in C^0(I)\,, \quad f_B \notin C^1(I)\,; \\ &\text{function C:} \quad f_C \in \mathcal{P}_{12}\,, \end{aligned}$$

where $\mathcal{P}_k$ is the space of the polynomials of degree at most k defined on I. The following quadrature rules are applied to these functions:

- quadrature rule A is a global Gauss quadrature;
- quadrature rule B is a composite trapezoidal rule;
- quadrature rule C is a composite 2-point Gauss quadrature.

The corresponding absolute values of the quadrature errors are plotted against the number of function evaluations in the figure below. Notice that only the quadrature errors obtained with an even number of function evaluations are shown.

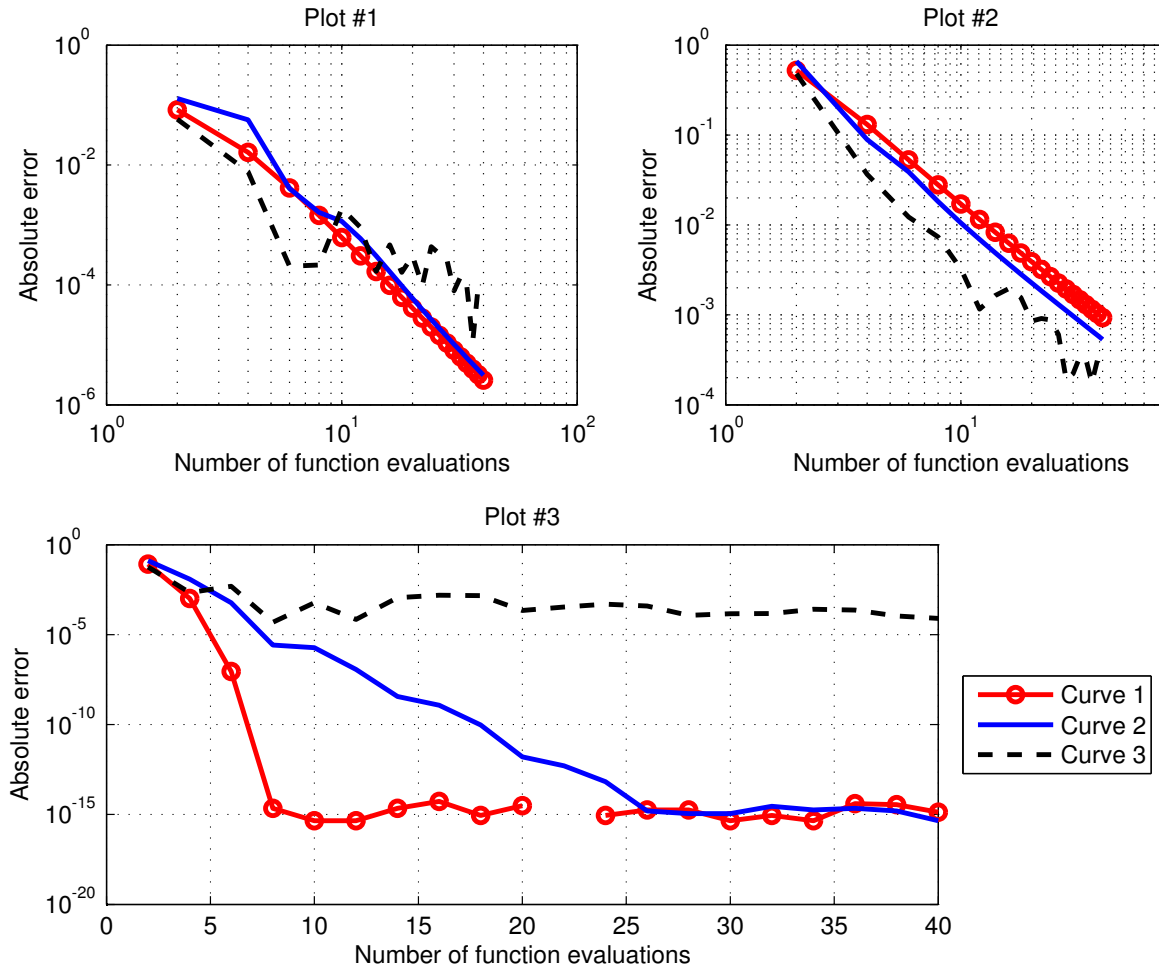


Fig. 34

(7.7.a) ⚀ Match the three plots (plot #1, #2 and #3) with the three quadrature rules (quadrature rule A, B, and C). Justify your answer.

HIDDEN HINT 1 for (7.7.a) → `7.7.1:plot1h.pdf`

SOLUTION for (7.7.a) → `7.7.1:plot1s.pdf` ▲

(7.7.b) ⚁ The quadrature error curves for a particular function f_A, f_B and f_C are plotted in the same style (curve 1 as red line with small circles, curve 2 means the blue solid line, curve 3 is the black dashed line). Which curve corresponds to which function (f_A, f_B, f_C)? Justify your answer.

SOLUTION for (7.7.b) → `7.7.2:plot2s.pdf` ▲

End Problem 7.7

Problem 7.8: Quadrature by transformation

In ➜Rem. 7.3.46 a suitable transformation converted a singular integrand into a smooth one. In this problem we see another example. Also Problem 7.2 and Problem 7.3 cover this topic.

For $f \in C([0,2])$ we consider the definite integral with singular integrand

$$I(f) := \int_0^2 \sqrt{\frac{2-t}{t}} f(t)\,\mathrm{d}t\,.$$

(7.8.a) ⚂

For $n \in \mathbb{N}^*$, derive a family of $2n$-point quadrature formulas

$$Q_n(f) = \sum_{j=1}^{2n} w_j^n f(c_j^n), \qquad f \in C([0,2])$$

for which $Q_n(f)$ converges to $I(f)$ exponentially as $n \to \infty$, if $f \in C^\infty([0,2])$, more precisely, if it has an analytic extension to a complex neighborhood of $[0,2]$.

HIDDEN HINT 1 for (7.8.a) → `7.8.1:hsi1.pdf`

HIDDEN HINT 2 for (7.8.a) → `7.8.1:hsi5.pdf`

SOLUTION for (7.8.a) → `7.8.1:qtf1.pdf`

▲

(7.8.b) ⚀ Given $n > 0$, determine the maximal $d \in \mathbb{N}$ such that $I(p) = Q_n(p)$ for every polynomial $p \in \mathcal{P}_{d-1}$.

SOLUTION for (7.8.b) → `7.8.2:1tf2ZerosLegendre5s.pdf` ▲

End Problem 7.8

Problem 7.9: Discretization of the integral operator

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

$$(Tf)(x) = \int_I k(x,y) f(y)\,\mathrm{d}y\,, \quad x \in I \subset \mathbb{R}\,, \tag{7.0.33}$$

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.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves 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 $g_n(x)$.

(7.9.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 $j = 1,\ldots,n-1$. 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 for (7.9.a) → `7.9.1:ongp1.pdf` ▲

(7.9.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 weigths `w` and the ordered nodes `xi` relative to the `n`-point Gauss quadrature on the interval $[-1,1]$.

SOLUTION for (7.9.b) → `7.9.2:ongp2.pdf` ▲

(7.9.c) ⚄ Test your implementation by computing $g(\xi_{11}^{21})$ for $f(y) = e^{-|0.5-y|}$. What result do you expect?

SOLUTION for (7.9.c) → `7.9.3:ongp3.pdf` ▲

End Problem 7.9

Chapter 8

Iterative Methods for Non-Linear Systems of Equations

Problem 8.1: Order of convergence from error recursion

In ➜Exp. 8.3.26 we have observed *fractional* orders of convergence (➜Def. 8.1.17) for both the secant method and the quadratic inverse interpolation method. This is fairly typical for 2-point methods in 1D and arises from the underlying recursions for error bounds. The analysis is elaborated for the secant method in ➜Rem. 8.3.27, where a linearised error recursion is given in ➜Eq. (8.3.31). This problem addresses how to determine the order of convergence from an abstract error recursion.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

(8.1.a) ⊡ Now we suppose the recursive bound for the norms of the iteration errors to be

$$\|e^{(n+1)}\| \leq \|e^{(n)}\| \sqrt{\|e^{(n-1)}\|}\,, \tag{8.0.1}$$

where $e^{(n)} = x^{(n)} - x^*$ is the error of n-th iterate.

Guess the maximal order of convergence of the method from a numerical experiment conducted in C++.

HIDDEN HINT 1 for (8.1.a) → `8.1.1:RecursionOrder1h.pdf`

SOLUTION for (8.1.a) → `8.1.1:RecursionOrder1s.pdf` ▲

(8.1.b) ⚄ Find the maximal guaranteed order of convergence of this method through analytical considerations.

HIDDEN HINT 1 for (8.1.b) → `8.1.2:RecursionOrder2ha.pdf`

HIDDEN HINT 2 for (8.1.b) → `8.1.2:RecursionOrder2hb.pdf`

HIDDEN HINT 3 for (8.1.b) → `8.1.2:RecursionOrder2hc.pdf`

SOLUTION for (8.1.b) → `8.1.2:RecursionOrder2s.pdf` ▲

End Problem 8.1

Problem 8.2: Code quiz

A frequently encountered drudgery in scientific computing is the use and modification of poorly documented code. This makes it necessary to understand the ideas behind the code first. Now we practice this in the case of a simple iterative method.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

(8.2.a) ⚀ What is the purpose of the following C++ code?

C++11-code 8.0.6: Undocumented function.

```
2  double myfunction(double x) {
3      double log2=0.693147180559945;
4      double y=0;
5      while(x>std::sqrt(2)){x/=2; y+=log2;} //
6      while(x<1./std::sqrt(2)){x*=2; y-=log2;} //
7      double z=x-1; //
8      double dz=x*std::exp(-z)-1.0;
9      while(std::abs(dz/z)>std::numeric_limits<double>::epsilon()) {
10         z+=dz; dz=x*std::exp(-z)-1.0;
11     }
12     return y+z+dz; //
13 }
```

Get it on GitLab (codequiz.cpp).

▲

HIDDEN HINT 1 for (8.2.a) → 8.2.1:Code1ha.pdf

HIDDEN HINT 2 for (8.2.a) → 8.2.1:Code1hb.pdf

SOLUTION for (8.2.a) → 8.2.1:Code1s.pdf

(8.2.b) ⚁ Explain the rationale behind the first two `while` loops in the code, Line 5, Line 6, preceding the main iteration.

SOLUTION for (8.2.b) → 8.2.2:Code2s.pdf ▲

(8.2.c) ⚂ Explain the last loop body does.

SOLUTION for (8.2.c) → 8.2.3:Code3s.pdf ▲

(8.2.d) ⚂ Explain the conditional expression in the last `while` loop.

SOLUTION for (8.2.d) → 8.2.4:Code4s.pdf ▲

(8.2.e) ⚃ Replace the last `while`-loop with a fixed number of iterations that, nevertheless, guarantee that the result has a relative accuracy `eps`.

SOLUTION for (8.2.e) → 8.2.5:Code5s.pdf ▲

End Problem 8.2

Problem 8.3: Convergent Newton iteration

As explained in ➜Section 8.3.2.1, the convergence of Newton's method in 1D may only be local. This problem investigates a particular setting, in which global convergence can be expected.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We recall the notion of a *convex function* ➜Def. 5.3.5 and its geometric meaning ➜Fig. 176: A differentiable function $f : [a,b] \mapsto \mathbb{R}$ is convex if and only if its graph lies on or above its tangent at any point. Equivalently, differentiable function $f : [a,b] \mapsto \mathbb{R}$ is convex, if and only if its derivative is non-decreasing.

Give a "graphical proof" of the following statement:

Theorem 8.0.10. Global convergence of Newton's method in 1D for convex monotone functions

If $F(x)$ belongs to $C^2(\mathbb{R})$, is strictly increasing, is convex, and has a unique zero, then the Newton iteration ➜Eq. (8.3.4) for $F(x) = 0$ is well defined and will converge to the zero of $F(x)$ for any initial guess $x^{(0)} \in \mathbb{R}$.

SOLUTION for (8.3.) → `8.3.0:Conv1s.pdf`

End Problem 8.3

Problem 8.4: Modified Newton method

The following problem consists in EIGEN implementation of a modified version of the Newton method (in one dimension ➜Section 8.3.2.1 and many dimensions ➜Section 8.4) for the solution of a non-linear system. Refresh yourself on stopping criteria for iterative methods ➜Section 8.1.2.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

For the solution of the non-linear system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ (with $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$), the following iterative method can be used:

$$\begin{aligned} \mathbf{y}^{(k)} &= \mathbf{x}^{(k)} + D\mathbf{F}(\mathbf{x}^{(k)})^{-1}\,\mathbf{F}(\mathbf{x}^{(k)})\,, \\ \mathbf{x}^{(k+1)} &= \mathbf{y}^{(k)} - D\mathbf{F}(\mathbf{x}^{(k)})^{-1}\,\mathbf{F}(\mathbf{y}^{(k)})\,, \end{aligned}$$

where $D\mathbf{F}(\mathbf{x}) \in \mathbb{R}^{n,n}$ is the Jacobian matrix of $\mathbf{F}$ evaluated in the point $\mathbf{x}$.

(8.4.a) ⚀ Show that the iteration is consistent with $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ in the sense of ➜Def. 8.2.1, that is, show that $\mathbf{x}^{(k)} = \mathbf{x}^{(0)}$ for every $k \in \mathbb{N}$ if and only if $\mathbf{F}(\mathbf{x}^{(0)}) = \mathbf{0}$ and $D\mathbf{F}(\mathbf{x}^{(0)})$ is regular.

SOLUTION for (8.4.a) → `8.4.1:Modi1s.pdf` ▲

(8.4.b) ⚁ Implement a C++ function

```
template <typename Scalar, class Function, class Jacobian>
Scalar mod_newt_step_scalar(const Scalar& x,
                            const Function& f,
                            const Jacobian& df);
```

that computes a step of the modified Newton method for a *scalar* function $\mathbf{F}$, that is, for the case $n = 1$.

Here, `f` is a function object of type `Function` passing the function $F : \mathbb{R} \mapsto \mathbb{R}$ and `df` a function object of type `Jacobian` passing the derivative $F' : \mathbb{R} \mapsto \mathbb{R}$. Both require an appropriate `operator()`.

SOLUTION for (8.4.b) → `8.4.2:Modi2s.pdf` ▲

(8.4.c) ⚀ What is the order of convergence of the method?
To investigate it, write a C++function `void mod_newt_ord()` that:

- uses the function `mod_newt_step` to the following scalar equation
$$\arctan(x) - 0.123 = 0\,;$$
- determines empirically the order of convergence, in the sense of ➜Rem. 8.1.19 of the course slides;
- implements meaningful stopping criteria (➜Section 8.1.2).

Use $x_0 = 5$ as initial guess.

HIDDEN HINT 1 for (8.4.c) → `8.4.3:Modi3ha.pdf`

HIDDEN HINT 2 for (8.4.c) → `8.4.3:Modi3hb.pdf`

SOLUTION for (8.4.c) → `8.4.3:Modi3s.pdf` ▲

(8.4.d) ⚄ Write a C++ function `void mod_newt_sys()` that provides an efficient implementation of the modified Newton method for the non-linear *system*
$$\mathbf{F}(\mathbf{x}) := \mathbf{A}\mathbf{x} + \begin{pmatrix} c_1 e^{x_1} \\ \vdots \\ c_n e^{x_n} \end{pmatrix} = \mathbf{0}\,,$$
where $\mathbf{A} \in \mathbb{R}^{n,n}$ is symmetric positive definite and $c_i \geq 0,\ i = 1,\ldots,n$. Stop the iteration when the Euclidean norm of the increment $\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ relative to the norm of $\mathbf{x}^{(k+1)}$ is smaller than the tolerance passed in `tol`. Use the zero vector as initial guess. Test your code and report the experimental order of convergence.

SOLUTION for (8.4.d) → `8.4.4:Modi4s.pdf` ▲

End Problem 8.4

Problem 8.5: The order of convergence of an iterative scheme

➜Rem. 8.1.19 shows how to detect the order of convergence of an iterative method from a numerical experiment. In this problem we study the so-called Steffensen's method, which is a derivative-free iterative method for finding zeros of functions in 1D.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Let $f : [a,b] \mapsto \mathbb{R}$ be twice continuously differentiable with $f(x^*) = 0$ and $f'(x^*) \neq 0$. Consider the iteration defined by
$$x^{(n+1)} := x^{(n)} - \frac{f(x^{(n)})}{g(x^{(n)})}\,, \quad \text{where} \quad g(x) = \frac{f(x + f(x)) - f(x)}{f(x)}\,. \tag{8.0.16}$$

(8.5.a) ⚀ Write a C++ function for the Steffensen's method:

```
template <class Function>
VectorXd steffensen(Function& f, double x0);
```

`f` is a handle to function f, `x0` is the initial guess $x^{(0)}$.

HIDDEN HINT 1 for (8.5.a) → `8.5.1:Quad1h.pdf`

SOLUTION for (8.5.a) → `8.5.1:Quad1s.pdf` ▲

(8.5.b) ⚀ Function $g(x)$ contains a term like e^{xe^x}, therefore it grows very fast in x and the method cannot start for a large $x^{(0)}$. How can you modify the function f (keeping the same zero) in order to allow the choice of a larger initial guess?

HIDDEN HINT 1 for (8.5.b) → `8.5.2:Quad2h.pdf`

SOLUTION for (8.5.b) → `8.5.2:Quad2s.pdf` ▲

End Problem 8.5

Problem 8.6: Newton's method for $F(x) := \arctan x$

The merely local convergence of Newton's method is notorious, see ➜Section 8.4.2 and ➜Ex. 8.4.54. The failure of the convergence is often caused by the overshooting of Newton correction. In this problem we try to understand the observations made in ➜Ex. 8.4.54.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

(8.6.a) ⚁ Find an equation satisfied by the smallest positive initial guess $x^{(0)}$ for which Newton's method does not converge when it is applied to $F(x) = \arctan x$.

HIDDEN HINT 1 for (8.6.a) → `8.6.1:NewtonArctan1ha.pdf`

HIDDEN HINT 2 for (8.6.a) → `8.6.1:NewtonArctan1hb.pdf`

SOLUTION for (8.6.a) → `8.6.1:NewtonArctan1s.pdf` ▲

(8.6.b) ⚀ Use Newton's method to find an approximation of such $x^{(0)}$ and implement it in C++:

```
double newton_arctan(double x0_);
```

SOLUTION for (8.6.b) → `8.6.2:NewtonArctan2s.pdf` ▲

End Problem 8.6

Problem 8.7: Order-p convergent iterations

In ➜Section 8.1.1 we investigated the speed of convergence of iterative methods for the solution of a general non-linear problem $F(\mathbf{x}) = 0$ and introduced the notion of convergence of order $p \geq 1$, see ➜Def. 8.1.17. This problem highlights the fact that for $p > 1$ convergence may not be guaranteed, even if the error norm estimate of ➜Def. 8.1.17 may hold for some $\mathbf{x}^* \in \mathbb{R}^n$ and all iterates $\mathbf{x}^{(k)} \in \mathbb{R}^n$.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Given $\mathbf{x}^* \in \mathbb{R}^n$, suppose that a sequence $\mathbf{x}^{(k)}$ satisfies ➜Def. 8.1.17:

$$\exists C > 0: \quad \|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq C\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^p \qquad \forall k \quad \text{and} \quad p > 1\,. \tag{8.0.19}$$

(8.7.a) ⚀ Determine $\epsilon_0 > 0$ as large as possible such that

$$\|\mathbf{x}^{(0)} - \mathbf{x}^*\| \leq \epsilon_0 \quad \Longrightarrow \quad \lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*\,. \tag{8.0.20}$$

In other words, ϵ_0 tells us which distance of the initial guess from $\mathbf{x}^*$ still guarantees local convergence.

SOLUTION for (8.7.a) → `8.7.1:OrdC1s.pdf` ▲

(8.7.b) ⚁ Provided that $\|\mathbf{x}^{(0)} - \mathbf{x}^*\| < \epsilon_0$ is satisfied with ϵ_0 from Sub-problem (8.7.a), determine the minimal iteration step $k_{\min} = k_{\min}(\epsilon_0, C, p, \tau)$ such that $\|\mathbf{x}^{(k)} - \mathbf{x}^*\| < \tau$.

SOLUTION for (8.7.b) → `8.7.2:OrdC2s.pdf` ▲

(8.7.c) ⚂ Complete the `TODO`s in the supplied `ordconviter.cpp` code and plot $k_{\min}(\epsilon_0, \tau)$ for the values $p = 1.5$, $C = 2$. Test you implementation in the `main` function for every $(\epsilon_0, \tau) \in$ `linspace` $\left(\left[0, C^{\frac{1}{1-p}}\right]^2\right)$.

SOLUTION for (8.7.c) → `8.7.3:OrdC3s.pdf` ▲

End Problem 8.7

Problem 8.8: Nonlinear electric circuit

In the previous exercises we have discussed electric circuits with elements that give rise to linear voltage–current dependence, see ➜Ex. 2.1.3 and ➜Ex. 2.8.1. The principles of nodal analysis were explained in these cases.
However, the electrical circuits encountered in practise usually feature elements with a *non-linear* current-voltage characteristic. Then nodal analysis leads to non-linear systems of equations as was elaborated in ➜Ex. 8.0.1. Please note that transformation to frequency domain is not possible for non-linear circuits so that we will always study the direct current (DC) situation.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

In this problem we deal with a very simple non-linear circuit element, a diode. The current through a diode as a function of the applied voltage can be modelled by the relationship $I_{kj} = \alpha\left(e^{\beta\frac{U_k-U_j}{U_T}} - 1\right)$, with suitable parameters α, β and the thermal voltage U_T. Now we consider the circuit depicted in Fig. 41 and assume that all resistors have resistance $R = 1$.

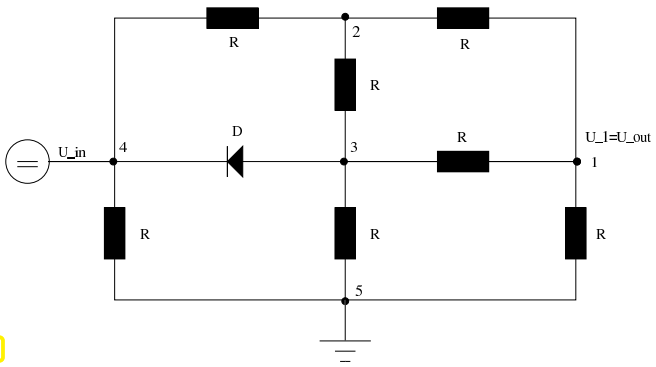


Fig. 41

(8.8.a) ⚀ Carry out the nodal analysis of the electric circuit and derive the corresponding non-linear system of equations $\mathbf{F}(\mathbf{u}) = \mathbf{0}$ for the voltages in nodes 1, 2 and 3, *cf.* ➜Eq. (8.0.2). Note that the voltages in nodes 4 and 5 are known (input voltage and ground voltage 0).

SOLUTION for (8.8.a) → `8.8.1:NonL1s.pdf` ▲

(8.8.b) ⚁ Write an EIGEN function

```
void circuit(const double & alpha, const double & beta, const
  VectorXd & Uin, VectorXd & Uout)
```

that computes the output voltages `Uout` (at node 1 in Fig. 41) for a *sorted* vector of input voltages `Uin` (at node 4) for a thermal voltage $U_T = 0.5$. The parameters `alpha`, `beta` pass the (non-dimensional) diode parameters.

Use Newton's method to solve $\mathbf{F}(\mathbf{u}) = \mathbf{0}$ with a tolerance of $\tau = 10^{-6}$.

SOLUTION for (8.8.b) → `8.8.2:NonL2s.pdf` ▲

(8.8.c) ⚂ We are interested in the nonlinear effects introduced by the diode. Calculate $U_{\text{out}} = U_{\text{out}}(U_{\text{in}})$ as a function of the variable input voltage $U_{\text{in}} \in [0, 20]$ (for non-dimensional parameters $\alpha = 8$, $\beta = 1$ and for a thermal voltage $U_T = 0.5$) and infer the nonlinear effects from the results.

SOLUTION for (8.8.c) → `8.8.3:NonL3s.pdf` ▲

End Problem 8.8

Problem 8.9: Julia Set

Julia sets are famous fractal shapes in the complex plane. They are constructed from the basins of attraction of zeros of complex functions when the Newton method is applied to find them.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

In the space $\mathbb{C}$ of complex numbers the equation

$$z^3 = 1 \tag{8.0.21}$$

has three solutions: $z_1 = 1$, $z_2 = -\frac{1}{2} + \frac{1}{2}\sqrt{3}i$, $z_3 = -\frac{1}{2} - \frac{1}{2}\sqrt{3}i$ (the cubic roots of unity).

(8.9.a) ⚀ As you know from the analysis course, the complex plane $\mathbb{C}$ can be identified with $\mathbb{R}^2$ via $(x, y) \mapsto z = x + iy$. Using this identification, convert Eq. (8.0.21) into a system of equations $\mathbf{F}(x, y) = \mathbf{0}$ for a suitable function $\mathbf{F} : \mathbb{R}^2 \mapsto \mathbb{R}^2$.

SOLUTION for (8.9.a) → `8.9.1:JuliaSet1s.pdf` ▲

(8.9.b) ⚁ Formulate Newton iteration ➜Eq. (8.4.1) for the non-linear equation $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ with $\mathbf{x} = (x,y)^T$ and $\mathbf{F}$ from the previous subproblem.

SOLUTION for (8.9.b) → `8.9.2:JuliaSet2s.pdf` ▲

(8.9.c) ⚄ Denote by $\mathbf{x}^{(k)}$ the iterates produced by Newton method from the previous subproblem with some initial vector $\mathbf{x}^{(0)} \in \mathbb{R}^2$. Depending on $\mathbf{x}^{(0)}$, the sequence $\mathbf{x}^{(k)}$ will either diverge or converge to one of the three cubic roots of unity.

Analyze the behavior of Newton iterations in C++ using the following procedure:

- Use equally spaced points on the domain $[-2,2]^2 \subset \mathbb{R}^2$ as starting points of Newton iterations.
- Color the starting points differently depending on which of the three roots is the limit of the sequence $\mathbf{x}^{(k)}$.

Complete the `TODO`s in the supplied `julia.cpp` code and plot the results running the `main` function.

HIDDEN HINT 1 for (8.9.c) → `8.9.3:JuliaSet3h.pdf`

SOLUTION for (8.9.c) → `8.9.3:JuliaSet3s.pdf` ▲

End Problem 8.9

Problem 8.10: Solving a quasi-linear system

In ➜§ 8.4.18 we studied Newton's method for a so-called quasi-linear system of equations, see ➜Eq. (8.4.19). In ➜Ex. 8.4.23 we then dealt with concrete quasi-linear system of equations and in this problem we will supplement the theoretical considerations from class by implementation in EIGEN. We will also learn about a simple fixed point iteration for that system, see ➜Section 8.2. Refresh yourself about the relevant parts of the lecture. You should also try to recall the Sherman-Morrison-Woodbury formula ➜Lemma 2.6.22.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Consider the *nonlinear* (quasi-linear) system:

$$\mathbf{A}(\mathbf{x})\mathbf{x} = \mathbf{b}\,, \tag{8.0.23}$$

as in ➜Ex. 8.4.23. Here, $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}^{n,n}$ is a matrix-valued function:

$$\mathbf{A}(\mathbf{x}) := \begin{bmatrix} \gamma(\mathbf{x}) & 1 & & & & \\ 1 & \gamma(\mathbf{x}) & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & \gamma(\mathbf{x}) & 1 \\ & & & & 1 & \gamma(\mathbf{x}) \end{bmatrix}, \quad \gamma(\mathbf{x}) := 3 + \|\mathbf{x}\|_2$$

where $\|\cdot\|_2$ is the Euclidean norm.

(8.10.a) ⚀ A fixed point iteration can be obtained from Eq. (8.0.23) by the "frozen argument technique"; in a step we take the argument to the matrix valued function from the previous step and just solve a linear system for the next iterate. State the defining recursion and iteration function for the resulting fixed point iteration. ▲

(8.10.b) ⚀ We consider the fixed point iteration derived in Eq. (8.0.23). Implement a function computing the iterate $\mathbf{x}^{(k+1)}$ from $\mathbf{x}^{(k)}$ in EIGEN.

HIDDEN HINT 1 for (8.10.b) → `8.10.2:QuasiLinear2h.pdf`

SOLUTION for (8.10.b) → `8.10.2:QuasiLinear2s.pdf` ▲

(8.10.c) ⚀ Write a routine that finds the solution $\mathbf{x}^*$ with the fixed point method applied to the previous quasi-linear system. Use $\mathbf{x}^{(0)} = \mathbf{b}$ as initial guess. Supply it with a suitable correction based stopping criterion as discussed in ➜Section 8.1.2 and pass absolute and relative tolerance as arguments.

SOLUTION for (8.10.c) → `8.10.3:QuasiLinear3s.pdf` ▲

(8.10.d) ⚀ Let $\mathbf{b} \in \mathbb{R}^n$ be given. Write the recursion formula for the solution of

$$\mathbf{A}(\mathbf{x})\mathbf{x} = \mathbf{b} \tag{8.0.26}$$

with the Newton method.

SOLUTION for (8.10.d) → `8.10.4:QuasiLinear4s.pdf` ▲

(8.10.e) ⚀ The matrix $\mathbf{A}(\mathbf{x})$, being symmetric and tri-diagonal, is cheap to invert. Rewrite the previous iteration efficiently, exploiting, the Sherman-Morrison-Woodbury inversion formula for rank-one modifications ➜Lemma 2.6.22.

SOLUTION for (8.10.e) → `8.10.5:QuasiLinear5s.pdf` ▲

(8.10.f) ⚁ Implement the above step of Newton method in EIGEN.

HIDDEN HINT 1 for (8.10.f) → `8.10.6:QuasiLinear6ha.pdf`

HIDDEN HINT 2 for (8.10.f) → `8.10.6:QuasiLinear6hb.pdf`

SOLUTION for (8.10.f) → `8.10.6:QuasiLinear6s.pdf` ▲

(8.10.g) ⚀ Repeat (8.10.f) for the Newton method. As initial guess use $\mathbf{x}^{(0)} = \mathbf{b}$.

SOLUTION for (8.10.g) → `8.10.7:QuasiLinear7s.pdf` ▲

End Problem 8.10

Chapter 9

Eigenvalues

Chapter 10

Krylov Methods for Linear Systems of Equations

Chapter 11

Numerical Integration – Single Step Methods

Problem 11.1: Integrating ODEs using the Taylor expansion method

In ➜Chapter 11 of the course we studied single step methods for the integration of initial value problems for ordinary differential equations $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, ➜Def. 11.3.5. Explicit single step methods have the advantage that they only rely on point evaluations of the right hand side $\mathbf{f}$.
This problem examines another class of methods that is obtained by the following reasoning: if the right hand side $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ of an autonomous initial value problem

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) , \qquad \mathbf{y}(0) = \mathbf{y}_0 , \tag{11.0.1}$$

with solution $\mathbf{y} : \mathbb{R} \to \mathbb{R}^n$ is smooth, the solution $\mathbf{y}(t)$ will also be regular and it is possible to expand it into a Taylor sum at $t = 0$, see ➜Thm. 8.2.15,

$$\mathbf{y}(t) = \sum_{n=0}^{m} \frac{\mathbf{y}^{(n)}(0)}{n!} t^n + R_m(t) , \tag{11.0.2}$$

with remainder term $R_m(t) = O(t^{m+1})$ for $t \to 0$.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

A single step method for the numerical integration of Eq. (11.0.1) can be obtained by choosing $m = 3$ in Eq. (11.0.2), neglecting the remainder term and taking the remaining sum as an approximation of $\mathbf{y}(h)$, that is

$$\mathbf{y}(h) \approx \mathbf{y}_1 := \mathbf{y}(0) + \frac{d\mathbf{y}(0)}{dt} h + \frac{1}{2} \frac{d^2\mathbf{y}(0)}{dt^2} h^2 + \frac{1}{6} \frac{d^3\mathbf{y}(0)}{dt^3} h^3 .$$

Subsequently, one uses the ODE and the initial condition to replace the temporal derivatives $\frac{d^i\mathbf{y}}{dt^i}$ with expressions in terms of (derivatives of) $\mathbf{f}$. This yields a single step integration method called *Taylor (expansion) method*.

(11.1.a) ⚄ Express $\frac{d\mathbf{y}}{dt}(t)$ and $\frac{d^2\mathbf{y}}{dt^2}(t)$ in terms of $\mathbf{f}$ and its Jacobian $\mathbf{D}\mathbf{f}$.

HIDDEN HINT 1 for (11.1.a) → `11.1.1:Tayl1h.pdf`

SOLUTION for (11.1.a) → `11.1.1:Tayl1s.pdf` ▲

(11.1.b) ⚄ Verify the formula

$$\frac{d^3\mathbf{y}}{dt^3}(0) = \mathbf{D}^2\mathbf{f}(\mathbf{y}_0)(\mathbf{f}(\mathbf{y}_0),\mathbf{f}(\mathbf{y}_0)) + \mathbf{D}\mathbf{f}(\mathbf{y}_0)^2\mathbf{f}(\mathbf{y}_0)\,. \tag{11.0.3}$$

HIDDEN HINT 1 for (11.1.b) → `11.1.2:Tayl0h.pdf`

HIDDEN HINT 2 for (11.1.b) → `11.1.2:Tayl2h.pdf`

HIDDEN HINT 3 for (11.1.b) → `11.1.2:Tayl24h.pdf`

SOLUTION for (11.1.b) → `11.1.2:Tayl2s.pdf` ▲

(11.1.c) ⚄ We now apply the Taylor expansion method introduced above to the following *predator-prey* model, already introduced in ➜Ex. 11.1.9:

$$\dot{y}_1(t) = (\alpha_1 - \beta_1 y_2(t))y_1(t) \tag{11.0.4}$$
$$\dot{y}_2(t) = (\beta_2 y_1(t) - \alpha_2)y_2(t) \tag{11.0.5}$$
$$\mathbf{y}(0) = [100,5]^\top \tag{11.0.6}$$

To this aim, in the template file `taylorintegrator.hpp` write a header-only C++ class `TaylorIntegrato` for the integration of the autonomous ODE of Eq. (11.0.4) using the Taylor expansion method with uniform timesteps on the temporal interval $[0,10]$.

HIDDEN HINT 1 for (11.1.c) → `11.1.3:Tayl3h.pdf`

SOLUTION for (11.1.c) → `11.1.3:Tayl3s.pdf` ▲

(11.1.d) ⚀ With the template file `taylorprey.cpp`, experimentally determine the order of convergence of the considered Taylor expansion method when it is applied to solve Eq. (11.0.4). Study the behaviour of the error at final time $t = 10$ for the initial data $\mathbf{y}(0) = [100,5]^\top$.

SOLUTION for (11.1.d) → `11.1.4:Tayl4s.pdf` ▲

(11.1.e) ⚀ What is the disadvantage of the Taylor's method compared with a Runge-Kutta method?

SOLUTION for (11.1.e) → `11.1.5:Tayl5s.pdf` ▲

End Problem 11.1

Problem 11.2: Linear ODE in spaces of matrices

In this problem we consider initial value problems (IVPs) for linear ordinary differential equations (ODEs) whose state space is a vector space of $n \times n$ matrices. Such ODEs arise when modelling the dynamics of rigid bodies in classical mechanics.
A related problem is Problem 11.4.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

First we consider the *linear* matrix differential equation

$$\dot{\mathbf{Y}} = \mathbf{A}\mathbf{Y} =: \mathbf{f}(\mathbf{Y}) \quad \text{with} \quad \mathbf{A} \in \mathbb{R}^{n\times n}. \tag{11.0.9}$$

whose solutions are *matrix-valued functions* $\mathbf{Y} : \mathbb{R} \to \mathbb{R}^{n\times n}$.

(11.2.a) ⊡ Show that for *skew-symmetric* $\mathbf{A}$, i.e. $\mathbf{A} = -\mathbf{A}^\top$ we have:

$$\mathbf{Y}(0) \text{ orthogonal} \quad \Longrightarrow \quad \mathbf{Y}(t) \text{ orthogonal} \quad \forall t\,.$$

HIDDEN HINT 1 for (11.2.a) → `11.2.1:MatO1h1.pdf`

HIDDEN HINT 2 for (11.2.a) → `11.2.1:MatO1h2.pdf`

SOLUTION for (11.2.a) → `11.2.1:MatO1s.pdf` ▲

(11.2.b) ⊡ Implement three C++ functions

1. a single step of the explicit Euler method, see ➜Section 11.2.1:

```
MatrixXd eeulstep(const Eigen::MatrixXd & A,
                  const MatrixXd & Y0, double h);
```

2. a single step of the implicit Euler method, see ➜Section 11.2.2,

```
MatrixXd ieulstep(const MatrixXd & A,
                  const MatrixXd & Y0, double h);
```

3. a single step of the implicit mid-point method, see ➜Section 11.2.3.

```
MatrixXd impstep(const MatrixXd & A,
                 const MatrixXd & Y0, double h);
```

which compute, for a given initial value $\mathbf{Y}(t_0) = \mathbf{Y}_0$ and for given step size h, approximations for $\mathbf{Y}(t_0 + h)$ using one step of the corresponding method for the approximation of the ODE (11.0.9)

HIDDEN HINT 1 for (11.2.b) → `11.2.2:MatO2h.pdf`

SOLUTION for (11.2.b) → `11.2.2:MatO2s.pdf` ▲

(11.2.c) ⊡ Investigate numerically, which one of the implemented methods preserves orthogonality for the ODE (11.0.9) and which one doesn't. To that end, consider the matrix

$$\mathbf{M} := \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 9 & 9 & 2 \end{bmatrix}$$

and use the matrix $\mathbf{Q}$ arising from the QR-decomposition of $\mathbf{M}$ as initial data $\mathbf{Y}_0$. As matrix A, use the skew-symmetric matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}.$$

To that end, perform $n = 20$ time steps of size $h = 0.01$ with each method and compute the Frobenius norm of $\mathbf{Y}(T)^\top \mathbf{Y}(T) - \mathbf{I}$.

SOLUTION for (11.2.c) → `11.2.3:MatO3s.pdf` ▲

End Problem 11.2

Problem 11.3: Explicit Runge-Kutta methods

The most widely used class of numerical integrators for IVPs is that of *explicit* Runge-Kutta (RK) methods as defined in ➜Def. 11.4.9. They are usually described by giving their coefficients in the form of a Butcher scheme ➜Eq. (12.0.18).

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

(11.3.a) ⚀ In the template file `rkintegrator.hpp`, code a header-only C++ class `RKIntegrator` which implements a generic RK method given by a Butcher scheme to solve the autonomous initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}),\ \mathbf{y}(t_0) = \mathbf{y}_0$:

```
template <class State>
class RKIntegrator;
```

SOLUTION for (11.3.a) → `11.3.1:RK3P1s.pdf` ▲

(11.3.b) ⚁ With the template file `rk3prey.cpp`, test your implementation of the RK methods with the following data. As autonomous initial value problem, consider the predator/prey model (*cf.* ➜Ex. 11.1.9):

$$\dot{y}_1(t) = (\alpha_1 - \beta_1 y_2(t))y_1(t) \tag{11.0.15}$$
$$\dot{y}_2(t) = (\beta_2 y_1(t) - \alpha_2)y_2(t) \tag{11.0.16}$$
$$\mathbf{y}(0) = [100, 5]^\top \tag{11.0.17}$$

with coefficients $\alpha_1 = 3, \alpha_2 = 2, \beta_1 = \beta_2 = 0.1$.

Use a Runge-Kutta single step method described by the following *Butcher scheme* (*cf.* ➜Def. 11.4.9):

$$\begin{array}{c|ccc} 0 & 0 & & \\ \frac{1}{3} & \frac{1}{3} & 0 & \\ \frac{2}{3} & 0 & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & 0 & \frac{3}{4} \end{array} \tag{11.0.18}$$

Compute an approximated solution up to time $T = 10$ for the number of steps $N = 2^j,\ j = 7, \ldots, 14$.

As reference solution, use $\mathbf{y}(10) = [0.319465882659820, 9.730809352326228]^\top$.

Tabulate the error and compute the experimental order of algebraic convergence of the method.

SOLUTION for (11.3.b) → `11.3.2:RK3P2s.pdf`

▲

End Problem 11.3

Problem 11.4: Non-linear Evolutions in Spaces of Matrices

In this problem we consider initial value problems (IVPs) for ordinary differential equations whose state space is a vector space of $n \times n$ matrices. Such IVPs occur in mathematical models of discrete mechanical systems.
Related to this problem is Problem 11.2.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We consider a *non-linear* ODE in matrix space and study the associated initial value problem

$$\dot{\mathbf{Y}} = -(\mathbf{Y} - \mathbf{Y}^\top)\mathbf{Y} =: f(\mathbf{Y}) \quad , \quad \mathbf{Y}(0) = \mathbf{Y}_0 \in \mathbb{R}^{n,n}, \tag{11.0.20}$$

whose solution is given by a *matrix-valued function* $t \mapsto \mathbf{Y}(t) \in \mathbb{R}^{n\times n}$.

(11.4.a) ⊡ Write a C++ function

```
MatrixXd matode(const MatrixXd & Y0, double T)
```

which solves (11.0.20) on $[0, T]$ using the C++ header-only class `ode45` (in the file `ode45.hpp`). The initial value should be given by a $n \times n$ EIGEN matrix `Y0`. Set the absolute tolerance to 10^{-10} and the relative tolerance to 10^{-8}. The output should be an approximation of $\mathbf{Y}(T) \in \mathbb{R}^{n\times n}$.

The `ode45` class works as follows:

1. Call the constructor, and specify the r.h.s. function f and the type for the solution and the initial data in `RhsType`, example:

   ```
   ode45<RhsType> O(f);
   ```

   with, for instance, `Eigen::VectorXd` as `StateType`.

2. (optional) Set custom options, modifying the `struct options` inside `ode45`, for instance:

   ```
   O.options.<option_you_want_to_change> = <value>;
   ```

3. Solve the IVP and store the solution, e.g.:

   ```
   std::vector<std::pair<Eigen::VectorXd, double>> sol = O.solve(y0, T);
   ```

Relative and absolute tolerances for `ode45` are defined as `rtol` resp. `atol` variables in the `struct options`. The return value is a sequence of states and times computed by the adaptive single step method.

The type `RhsType` needs a vector space structure implemented with operators `*`, `*`, `*=`, `+=` and assignment/copy operators. Moreover a norm method must be available. Eigen vector and matrix types, as well as fundamental types are eligible as `RhsType`.

SOLUTION for (11.4.a) → `11.4.1:NMatO1s.pdf` ▲

(11.4.b) ⊡ Show that the function $t \mapsto \mathbf{Y}^\top(t)\mathbf{Y}(t)$ is constant for the exact solution $\mathbf{Y}(t)$ of (11.0.20).

HIDDEN HINT 1 for (11.4.b) → `11.4.2:NMatO2h1.pdf`

HIDDEN HINT 2 for (11.4.b) → `11.4.2:NMatO2h2.pdf`

SOLUTION for (11.4.b) → `11.4.2:NMatO2s.pdf` ▲

(11.4.c) ⚀ Write a C++ function

```
bool checkinvariant(const Eigen::MatrixXd & M, double T);
```

which (numerically) determines if the invariant is preserved, for $t = T$ and for the output of `matode`. You must take into account round-off errors. The function's input should be the same as that of `matode`.

SOLUTION for (11.4.c) → `11.4.3:NMatO3s.pdf` ▲

(11.4.d) ⚀ Use the function `checkinvariant` to test whether the invariant is preserved by `ode45` or not. Use the matrix $\mathbf{M}$ defined above and and $T = 1$.

SOLUTION for (11.4.d) → `11.4.4:NMatO4s.pdf` ▲

End Problem 11.4

Problem 11.5: System of second-order ODEs

In this problem we practise the conversion of a second-order ODE into a first-order system in the case of a large linear system of ODEs.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Consider the following initial value problem for an (implicit) second-order system of ordinary differential equations in the time interval $[0, T]$:

$$2\ddot{u}_1 - \ddot{u}_2 = u_1(u_2 + u_1)\,, \tag{11.0.26}$$

$$-\ddot{u}_{i-1} + 2\ddot{u}_i - \ddot{u}_{i+1} = u_i(u_{i-1} + u_{i+1})\,, \quad i = 2, \ldots, n-1\,, \tag{11.0.27}$$

$$2\ddot{u}_n - \ddot{u}_{n-1} = u_n(u_n + u_{n-1})\,, \tag{11.0.28}$$

$$u_i(0) = u_{0,i} \quad i = 1, \ldots, n\,, \tag{11.0.29}$$

$$\dot{u}_i(0) = v_{0,i} \quad i = 1, \ldots, n\,. \tag{11.0.30}$$

Here the notation $\ddot{w}$ designates the second derivative of a time-dependent function $t \mapsto w(t)$.

(11.5.a) ⚁ Write Eq. (11.0.26) as a first-order IVP of the form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$.

HIDDEN HINT 1 for (11.5.a) → `11.5.1:Syst1h1.pdf`

HIDDEN HINT 2 for (11.5.a) → `11.5.1:Syst1h2.pdf`

HIDDEN HINT 3 for (11.5.a) → `11.5.1:Syst1h3.pdf`

SOLUTION for (11.5.a) → `11.5.1:Syst1s.pdf` ▲

(11.5.b) ⚁ Write down the equations for the Runge-Kutta increments $\mathbf{k}_i$ as stated in ➜Def. 11.4.9 for the concrete case of the linear ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{R}^{d,d}$, and the classical Runge-Kutta method of order 4, whose Butcher scheme is given in ➜Ex. 11.4.13.

SOLUTION for (11.5.b) → `11.5.2:Syst2s.pdf` ▲

(11.5.c) ⚁ Implement a C++ function

```
template <class Function, class State>
void rk4step(const Function &odefun, double h, const State & y0,
   State & y1);
```

that performs a single step of stepsize h of the classical Runge-Kutta method of order 4 for the first-order ODE obtained in Sub-problem (11.5.a) (passed via `odefun`). `y0` contains the state before the step and the function has to return the state after the step in `y1`.

SOLUTION for (11.5.c) → `11.5.3:Syst3s.pdf` ▲

(11.5.d) ⚀ Apply the function `errors` included in utility header file `errors.hpp` to the IVP obtained in the previous subproblem. Use

$$n = 5\,, \qquad u_{0,i} = i/n\,, \qquad v_{0,i} = -1\,, \qquad T = 1\,,$$

and the classical RK method of order 4. Construct any sparse matrix encountered as a sparse matrix in EIGEN. Comment on the order of convergence observed.

Function `errors` is defined as follows:

```
template <class Function>
void errors(const Function &f, const double &T, const VectorXd
   &y0, const MatrixXd &A, const VectorXd &b);
```

This template function approximates the order of convergence of the RK scheme defined by `A` and `b` when applied to the first-order system $\dot{y} = f(y)$, $y(0) = y_0$. We are interested in the error of the solution at time `T`.

SOLUTION for (11.5.d) → `11.5.4:Syst4s.pdf` ▲

End Problem 11.5

Problem 11.6: Order is not everything

In ➜Section 11.3.2 we have seen that Runge-Kutta single step methods when applied to initial value problems with sufficiently smooth solutions will converge algebraically (with respect to the maximum error in the mesh points) with a rate given by their intrinsic order, see ➜Def. 11.3.21.
This problem relies on a class implemented in Problem 11.3.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

In this problem we perform empiric investigations of orders of convergence of several explicit Runge-Kutta single step methods. We rely on two IVPs, one of which has a perfectly smooth solution, whereas the second has a solution that is merely piecewise smooth. Thus in the second case the smoothness assumptions of the convergence theory for RK-SSMs might be violated and it is interesting to study the consequences.

In order to use the class `RKIntegrator`, you first need to construct an object of this class, passing as arguments the Butcher tableau matrices `A` and `b`, for instance:

```
RKIntegrator<VectorXd> rk(A,b);
```

After that, call the methods `solve`, with parameters: r.h.s. function, fi nal time, initial value and number of steps. For instance:

```
rk.solve(f,T,y0,n);
```

The output of this function will be `std::vector<VectorXd>` containing the solution at each equidistant time step.

(11.6.a) ⚀

Consider the autonomous ODE

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0, \tag{11.0.33}$$

where $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^n$ and $\mathbf{y}_0 \in \mathbb{R}^n$. Using the class `RKIntegrate` write a C++ function

```
template <class Function>
void errors(const Function &f, double T,
            const VectorXd &y0,
            const MatrixXd &A, const VectorXd &b)
```

that computes an approximated solution $\mathbf{y}_N$ of (11.0.33) up to time T by means of an explicit Runge-Kutta method with $N = 2^k$, $k = 1,\ldots,15$, uniform timesteps. The method is defined by the Butcher scheme described by the inputs `A` and `b`. The input `f` is an object with an evaluation operator (e.g. a lambda function) for arguments of type const VectorXd & representing $\mathbf{f}$. The input `y0` passes the initial value $\mathbf{y}_0$.

For each k, the function should show the error at the final point $E_N = |\mathbf{y}_N(T) - \mathbf{y}_{2^{15}}(T)|$, $N = 2^k$, $k = 1,\ldots,13$, accepting $\mathbf{y}_{2^{15}}(T)$ as exact value. Assuming algebraic convergence for $E_N \approx CN^{-r}$, at each step show an approximation of the order of convergence r_k (recall that $N = 2^k$). This will be an expression involving E_N and $E_{N/2}$.

Finally, compute and show an approximate order of convergence by averaging the relevant r_Ns (namely, you should take into account the cases before machine precision is reached in the components of $\mathbf{y}_N(T) - \mathbf{y}_{2^{15}}(T)$).

SOLUTION for (11.6.a) → `11.6.1:OrdN1h.pdf` ▲

(11.6.b) ⚀ Calculate the analytical solutions of the logistic ODE (see ➜Ex. 11.1.5)

$$\dot{y} = (1-y)y, \quad y(0) = 1/2, \tag{11.0.36}$$

and of the initial value problem

$$\dot{y} = |1.1 - y| + 1, \quad y(0) = 1. \tag{11.0.37}$$

HIDDEN HINT 1 for (11.6.b) → `11.6.2:ONA2s.pdf`

HIDDEN HINT 2 for (11.6.b) → `11.6.2:OMA6s.pdf`

HIDDEN HINT 3 for (11.6.b) → `11.6.2:OMA8s.pdf`

SOLUTION for (11.6.b) → `11.6.2:OrdN2h.pdf` ▲

(11.6.c) ⚀ Use the function `errors` with the ODEs (11.0.36) and (11.0.37) and the methods:

- the explicit Euler method ➜Eq. (11.2.7), a RK single step method of order 1,
- the explicit trapezoidal rule ➜Eq. (11.4.6), a RK single step method of order 2,
- an RK method of order 3 given by the Butcher tableau ➜Eq. (12.0.18)

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & -1 & 2 & \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

- the classical RK method of order 4, see ➜Ex. 11.4.13 for details.

Use final time $T = 0.1$ and initial value $y_0 = 0.5$.

Comment on the calculated order of convergence for the different methods and the two different initial value problems.

SOLUTION for (11.6.c) → `11.6.3:OrdN3h.pdf` ▲

End Problem 11.6

Problem 11.7: Initial Condition for Lotka-Volterra ODE

In this problem we will face a situation, where we need to compute the derivative of the solution of an initial value problem with respect to the initial state.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We consider IVPs for the autonomous ODE

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \tag{11.0.41}$$

with smooth right hand side $\mathbf{f}\colon D \to \mathbb{R}^d$, where $D \subseteq \mathbb{R}^d$ is the state space. We take for granted that for all initial states, solutions exist for all times (global solutions, see ➜Ass. 11.1.38).

By its very definition given in ➜Def. 11.1.39, the evolution operator

$$\boldsymbol{\Phi}\colon \mathbb{R} \times D \to D, \quad (t, \mathbf{y}) \mapsto \boldsymbol{\Phi}(t, \mathbf{y})$$

satisfies

$$\frac{\partial \boldsymbol{\Phi}}{\partial t}(t, \mathbf{y}) = \mathbf{f}(\boldsymbol{\Phi}(t, \mathbf{y})).$$

Next, we can differentiate this identity with respect to the state variable $\mathbf{y}$. We assume that all derivatives can be interchanged, which can be justified by rigorous arguments (which we won't do here). Thus, by the chain rule, we obtain, after swapping partial derivatives $\frac{\partial}{\partial t}$ and $\mathrm{D}_{\mathbf{y}}$,

$$\frac{\partial\, \mathrm{D}_{\mathbf{y}}\, \boldsymbol{\Phi}}{\partial t}(t, \mathbf{y}) = \mathrm{D}_{\mathbf{y}} \frac{\partial \boldsymbol{\Phi}}{\partial t}(t, \mathbf{y}) = \mathrm{D}_{\mathbf{y}}(\mathbf{f}(\boldsymbol{\Phi}(t, \mathbf{y}))) = \mathrm{D}\,\mathbf{f}(\boldsymbol{\Phi}(t, \mathbf{y}))\, \mathrm{D}_{\mathbf{y}}\, \boldsymbol{\Phi}(t, \mathbf{y}).$$

Abbreviating $\mathbf{W}(t, \mathbf{y}) := \mathrm{D}_{\mathbf{y}}\, \boldsymbol{\Phi}(t, \mathbf{y})$ we can rewrite this as the non-autonomous ODE

$$\dot{\mathbf{W}} = \mathrm{D}\,\mathbf{f}(\boldsymbol{\Phi}(t, \mathbf{y}))\mathbf{W}. \tag{11.0.42}$$

Here, the state $\mathbf{y}$ can be regarded as a parameter. Since $\boldsymbol{\Phi}(0, \mathbf{y}) = \mathbf{y}$, we also know $\mathbf{W}(0, \mathbf{y}) = \mathbf{I}$ (identity matrix), which supplies an initial condition for (11.0.42). In fact, we can even merge (11.0.41) and (11.0.42) into the ODE

$$\frac{d}{dt}[\mathbf{y}(\cdot)\,,\ \mathbf{W}(\cdot, \mathbf{y}_0)] = [\mathbf{f}(\mathbf{y}(t))\,,\ \mathrm{D}\,\mathbf{f}(\mathbf{y}(t))\mathbf{W}(t, \mathbf{y}_0)]\,, \tag{11.0.43}$$

which is autonomous again.

Now let us apply (11.0.42)/(11.0.43). As in ➜Ex. 11.1.9, we consider the following autonomous Lotka-Volterra differential equation of a predator-prey model

$$\begin{aligned} \dot{u} &= (2 - v)u \\ \dot{v} &= (u - 1)v \end{aligned} \tag{11.0.44}$$

on the state space $D = \mathbb{R}^2_+$, $\mathbb{R}_+ = \{\xi \in \mathbb{R} : \xi > 0\}$. All the solutions of (11.0.44) are periodic and their period depends on the initial state $[u(0), v(0)]^T$. In this exercise we want to develop a numerical method which computes a suitable initial condition for a given period.

(11.7.a) ⚀ For fixed state $\mathbf{y} \in D$, (11.0.42) represents an ODE. What is its state space?

SOLUTION for (11.7.a) → `11.7.1:Init1h.pdf` ▲

(11.7.b) ⚁ What is the right hand side function for the ODE (11.0.42), in the case of the $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ given by the Lotka-Volterra ODE (11.0.44)? You may write $u(t), v(t)$ for solutions of (11.0.44).

SOLUTION for (11.7.b) → `11.7.2:Init2h.pdf` ▲

(11.7.c) ⚁ From now on we write $\boldsymbol{\Phi}: \mathbb{R} \times \mathbb{R}^2_+ \to \mathbb{R}^2_+$ for the evolution operator associated with (11.0.44). Based on $\boldsymbol{\Phi}$ derive a function $\mathbf{F} : \mathbb{R}^2_+ \to \mathbb{R}^2_+$ which evaluates to zero for the input $\mathbf{y}_0$ if the period of the solution of system (11.0.44) with initial value

$$\mathbf{y}_0 = \begin{bmatrix} u(0) \\ v(0) \end{bmatrix}$$

is equal to a given value T_P.

SOLUTION for (11.7.c) → `11.7.3:Init4h.pdf` ▲

(11.7.d) ⚁ We write $\mathbf{W}(T, \mathbf{y}_0)$, $T \geq 0$, $\mathbf{y}_0 \in \mathbb{R}^2_+$ for the solution of (11.0.42) for the underlying ODE (11.0.44). Express the Jacobian of $\mathbf{F}$ by means of $\mathbf{W}$.

SOLUTION for (11.7.d) → `11.7.4:Init5h.pdf` ▲

(11.7.e) ⚂ Argue, why the solution of $\mathbf{F}(\mathbf{y}) = 0$ will, in gneneral, not be unique. When will it be unique?

HIDDEN HINT 1 for (11.7.e) → `11.7.5:Init1s.pdf`

SOLUTION for (11.7.e) → `11.7.5:Init6h.pdf` ▲

A C++ implementation of an adaptive embedded Runge-Kutta method is available, with a functionality similar to MATLAB's `ode45` see Problem 11.2).

Instructions on the usage of `ode45`.

The class `ode45` is header-only, meaning you just include the file and use it right away (no linking required). The file `ode45.hpp` defines class `ode45` implementing an emulation of Matlab's Rosenbrock method of order 4(3): `ode45`.

1. Construct an object of `ode45` type: create an instance of the class, passing the r.h.s. function f to constructor:

   ```
   ode45<StateType> = O(f);
   ```

   Template parameters:

   - StateType: type of initial data and solution (state space), the only requirement is that the tpye possesses a normed vector-space structure;
   - RhsType: type of rhs function (automatically deduced).

   The function `f` must be a function handle with

   ```
   operator()(const StateType & vec) -> StateType
   ```

2. (optional) Set the integration options: set members of struct `ode45.options` to configure the solver:

   ```
   O.options.<option_you_want_to_set> = <value>
   ```

   Examples:

- `rtol`: relative tolerance for error control (default is `10e-6`)
- `atol`: absolute tolerance for error control (default is `10e-8`)

e.g.:

```
O.options.rtol = 10e-5;
```

3. Solve stage: call the solver:

   ```
   std::vectro<std::pair<StateType> sol = O.solve(y0, T, norm)
   ```

   Template parameters:

   - `NormType`: type of norm function, automatically deduced

   Arguments:

   - `y0`: initial value in StateType ($y(0) = \mathtt{y0}$)
   - `T`: final time of integration
   - `norm`: (optional) norm function to call on member of StateType, for the computation of the error

   The function returns the solution of the IVP, as a `std::vector` of `std::pair` $(y(t), t)$ for every snapshot.

For more documentation, consult the in-class documentation or the file `NumCSE/Utils/README.md`.

(11.7.f) ⚀

Relying on `ode45`, implement a C++ function

```
std::pair<Vector2d,Matrix2d> PhiAndW(double u0, double v0, double T)
```

that computes $\mathbf{\Phi}(T, [u_0, v_0]^T)$ and $\mathbf{W}(T, [u_0, v_0]^T)$. The first component of the output pair should contain $\mathbf{\Phi}(T, [u_0, v_0]^T)$ and the second component the matrix $\mathbf{W}(T, [u_0, v_0]^T)$.

HIDDEN HINT 1 for (11.7.f) → `11.7.6:Init2s.pdf`

SOLUTION for (11.7.f) → `11.7.6:Init7h.pdf` ▲

(11.7.g) ⚁ Using `PhiAndW`, write a C++ routine that determines initial conditions $u(0)$ and $v(0)$ such that the solution of the system (11.0.44) has period $T = 5$. Use the multi-dimensional *Newton method* for $\mathbf{F}(\mathbf{y}) = 0$ with $\mathbf{F}$. As your initial approximation, use $[3, 2]^T$. Terminate the *Newton method* as soon as $|\mathbf{F}(\mathbf{y})| \leq 10^{-5}$. Validate your implementation by comparing the obtained initial data $\mathbf{y}$ with $\mathbf{\Phi}(100, \mathbf{y})$.

Set relative and absolute tolerances of `ode45` to 10^{-14} and 10^{-12}, respectively.

Remark. The residual based termination criterion recommended above ➜§ 8.1.26 is appropriate for this particular application and, in general, should not be used for Newton's method. Better termination criteria are proposed in ➜Section 8.4.3.

The correct solutions are $u(0) \approx 3.110$ and $v(0) = 2.081$.

SOLUTION for (11.7.g) → `11.7.7:Init8h.pdf` ▲

End Problem 11.7

Chapter 12

Single Step Methods for Stiff Initial Value Problems

Problem 12.1: Semi-implicit Runge-Kutta SSM

General implicit Runge-Kutta methods as introduced in ➜Section 12.3.3 entail solving systems of non-linear equations for the increments, see ➜Rem. 12.3.24. Semi-implicit Runge-Kutta single step methods, also known as Rosenbrock-Wanner (ROW) methods ➜Eq. (12.4.6) just require the solution of linear systems of equations. This problem deals with a concrete ROW method, its stability and aspects of implementation.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We consider the following autonomous ODE:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \tag{12.0.1}$$

and discretize it with a *semi-implicit* Runge-Kutta SSM (*Rosenbrock method*):

$$\begin{aligned} \mathbf{W}\mathbf{k}_1 &= \mathbf{f}(\mathbf{y}_0)\,, \\ \mathbf{W}\mathbf{k}_2 &= \mathbf{f}(\mathbf{y}_0 + \frac{1}{2}h\mathbf{k}_1) - ah\mathbf{J}\mathbf{k}_1\,, \\ \mathbf{y}_1 &= \mathbf{y}_0 + h\mathbf{k}_2\,, \end{aligned} \tag{12.0.2}$$

where

$$\mathbf{J} = D\mathbf{f}(\mathbf{y}_0)\,, \quad \mathbf{W} = \mathbf{I} - ah\mathbf{J}\,, \quad a = \frac{1}{2+\sqrt{2}}\,.$$

(12.1.a) ⚀ Compute the stability function S of the Rosenbrock method (12.0.2), that is, compute the (rational) function $S(z)$, such that

$$y_1 = S(z)y_0, \quad z := h\lambda,$$

when we apply the method to perform one step of size h, starting from y_0, of the linear scalar model ODE $\dot{y} = \lambda y, \lambda \in \mathbb{C}$.

SOLUTION for (12.1.a) → `12.1.1:SemI1s.pdf` ▲

(12.1.b) ⚀ Compute the first 4 terms of the Taylor expansion of $S(z)$ around $z = 0$. What is the maximal $q \in \mathbb{N}$ such that

$$|S(z) - \exp(z)| = O(|z|^q)$$

for $|z| \to 0$? Deduce the maximal possible order of the method Eq. (12.0.2).

HIDDEN HINT 1 for (12.1.b) → `12.1.2:SemI2h.pdf`

SOLUTION for (12.1.b) → `12.1.2:SemI2s.pdf` ▲

(12.1.c) ⚀ Implement a C++ function:

```
template <class Function, class Jacobian, class StateType>
std::vector<StateType> solveRosenbrock(const Function & f,
                                       const Jacobian & df,
                                       const StateType & y0,
                                       unsigned int N, double T);
```

taking as input function handles for $\mathbf{f}$ and $D\mathbf{f}$ (e.g., as lambda functions), an initial data (vector or scalar) `y0` $= \mathbf{y}(0)$, a number of steps N and a final time T. The function returns the sequence of states generated by the single step method up to $t = T$, using N equidistant steps of the Rosenbrock method.

SOLUTION for (12.1.c) → `12.1.3:SemI3s.pdf` ▲

(12.1.d) ⚀ Explore the order of the method Eq. (12.0.2) empirically by applying it to the IVP for the limit cycle ➜Ex. 12.2.5:

$$\mathbf{f}(\mathbf{y}) := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2)\mathbf{y}\,, \tag{12.0.4}$$

with $\lambda = 1$ and initial state $\mathbf{y}_0 = [1,1]^\top$ on $[0,10]$. Use uniform timesteps of size $h = 2^{-k}, k = 4,\ldots,10$ and compute a reference solution $\mathbf{y}_{\mathrm{ref}}$ with timestep size $h = 2^{-12}$. Monitor the maximal error on the temporal mesh

$$\max_j \|\mathbf{y}_j - \mathbf{y}_{\mathrm{ref}}(t_j)\|_2\,.$$

SOLUTION for (12.1.d) → `12.1.4:SemI4s.pdf` ▲

In complex analysis you might have hear about the maximum principle for holomorphic/analytic functions. The following is a special version of it.

Theorem 12.0.5. Maximum principle for holomorphic functions

Let

$$\mathbb{C}^- := \{z \in \mathbb{C} \mid Re(z) < 0\}\,.$$

Let $f : D \subset \mathbb{C} \to \mathbb{C}$ be non-constant, defined on $\overline{\mathbb{C}^-}$, and analytic in $\mathbb{C}^-$. Furthermore, assume that $w := \lim_{|z|\to\infty} f(z)$ exists and $w \in \mathbb{C}$, then:

$$\forall z \in \mathbb{C}^-: \quad |f(z)| < \sup_{\tau\in\mathbb{R}} |f(i\tau)|\,.$$

(12.1.e) ⚀

Appealing to Thm. 12.0.5 show that the method (12.0.2) is L-stable (*cf.* ➜§ 12.3.37).

HIDDEN HINT 1 for (12.1.e) → `12.1.5:SemI5h.pdf`

SOLUTION for (12.1.e) → `12.1.5:SemI5s.pdf` ▲

End Problem 12.1

Problem 12.2: Exponential integrator

The exponential integrators are a modern class of single step methods developed for special initial value problems (problems that can be regarded as perturbed linear ODEs), see

M. HOCHBRUCK AND A. OSTERMANN, *Exponential integrators*, Acta Numerica, 19 (2010), pp. 209–286.

These methods fit the concept of single step methods as introduced in ➜Def. 11.3.5 and, usually, converge algebraically according to ➜Eq. (11.3.20).

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

A step with size h of the so-called *exponential Euler* single step method for the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ with continuously differentiable $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$ reads:

$$\mathbf{y}_1 = \mathbf{y}_0 + h\, \varphi\big(h\, \mathrm{D}\,\mathbf{f}(\mathbf{y}_0)\big)\, \mathbf{f}(\mathbf{y}_0), \tag{12.0.6}$$

where $\mathrm{D}\,\mathbf{f}(\mathbf{y}) \in \mathbb{R}^{d,d}$ is the Jacobian of $\mathbf{f}$ at $\mathbf{y} \in \mathbb{R}^d$, and the matrix function $\varphi : \mathbb{R}^{d,d} \to \mathbb{R}^{d,d}$ is defined as $\varphi(\mathbf{Z}) = (\exp(\mathbf{Z}) - \mathrm{Id})\, \mathbf{Z}^{-1}$. Here, $\exp(\mathbf{Z})$ is the matrix exponential of $\mathbf{Z}$, a special function $\exp : \mathbb{R}^{d,d} \to \mathbb{R}^{d,d}$, see ➜Eq. (12.1.32).

The function φ is implemented in the template file as the function `MatrixXd phim(const MatrixXd & Z)`.

When plugging in the exponential series, it is clear that the function $z \mapsto \varphi(z) := \frac{\exp(z)-1}{z}$ is analytic on $\mathbb{C}$. Thus, $\varphi(\mathbf{Z})$ is well defined for all matrices $\mathbf{Z} \in \mathbb{R}^{d,d}$.

(12.2.a) ⚁ Is the exponential Euler single step method defined in (12.0.6) consistent with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ (see ➜Def. 11.3.10)? Explain your answer.

SOLUTION for (12.2.a) → `12.2.1:Expo1h.pdf` ▲

(12.2.b) ⚁ Show that the exponential Euler single step method defined in (12.0.6) solves the linear initial value problem

$$\dot{\mathbf{y}} = \mathbf{A}\,\mathbf{y}\,, \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^d\,, \qquad \mathbf{A} \in \mathbb{R}^{d,d}\,,$$

exactly.

HIDDEN HINT 1 for (12.2.b) → `12.2.2:Expo1s.pdf`

SOLUTION for (12.2.b) → `12.2.2:Expo2h.pdf` ▲

(12.2.c) ⚁ Determine the region of stability of the exponential Euler single step method defined in (12.0.6) (see ➜Def. 12.1.49).

SOLUTION for (12.2.c) → `12.2.3:Expo3h.pdf` ▲

(12.2.d) ⚁ Write a C++ function

```
template <class Function, class Jacobian>
VectorXd ExpEulStep(const VectorXd & y0,
                    const Function& f, const Jacobian & df,
                    double h);
```

that implements (12.0.6). Here `f` and `df` are objects with evaluation operators representing the ODE right-hand side function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$ and its Jacobian, respectively.

SOLUTION for (12.2.d) → `12.2.4:Expo4h.pdf` ▲

(12.2.e) ⚀ What is the order of the single step method (12.0.6)?

To investigate it, write a C++ routine that applies the method to the scalar logistic ODE

$$\dot{y} = y\,(1-y)\,, \quad y(0) = 0.1\,,$$

in the time interval $[0,1]$. Show the error at the final time against the stepsize $h = T/N$, $N = 2^k$ for $k = 1,\ldots,15$.For each k compute and show an approximate order of convergence.

HIDDEN HINT 1 for (12.2.e) → `12.2.5:Expo3s.pdf`

SOLUTION for (12.2.e) → `12.2.5:Expo5h.pdf` ▲

End Problem 12.2

Problem 12.3: Damped precession of a magnetic needle

This problem deals with a dynamical system from mechanics describing the movement of a rod-like magnet in a strong magnetic field. This can be modelled by an ODE with a particular invariant that can become stiff in the case of large friction.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We consider the initial value problem

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) := \mathbf{a} \times \mathbf{y} + c\mathbf{y} \times (\mathbf{a} \times \mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0 = [1,1,1]^\top, \tag{12.0.9}$$

where $c > 0$ and $\mathbf{a} \in \mathbb{R}^3$, $|a|_2 = 1$.

Note: $\mathbf{x} \times \mathbf{y}$ denotes the cross product between the vectors $\mathbf{x}$ and $\mathbf{y}$. It is defined by

$$\mathbf{x} \times \mathbf{y} = [x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1]^\top.$$

It satisfies $\mathbf{x} \times \mathbf{y} \perp \mathbf{x}$. In Eigen, it is available as `x.cross(y)`.

(12.3.a) ⚀ Show that $|\mathbf{y}(t)|_2 = |\mathbf{y}_0|_2$ for every solution $\mathbf{y}$ of the ODE.

HIDDEN HINT 1 for (12.3.a) → `12.3.1:Cros1s.pdf`

SOLUTION for (12.3.a) → `12.3.1:Cros1h.pdf` ▲

(12.3.b) ⚀ Compute the Jacobian $D\mathbf{f}(\mathbf{y})$. Compute also the spectrum $\sigma(D\mathbf{f}(\mathbf{y}))$ in the stationary state $\mathbf{y} = \mathbf{a}$, for which $\mathbf{f}(\mathbf{y}) = 0$. For simplicity, you may consider only the case $\mathbf{a} = [1,0,0]^\top$.

SOLUTION for (12.3.b) → `12.3.2:Cros2h.pdf` ▲

(12.3.c) ⚀

For $\mathbf{a} = [1,0,0]^\top$, (12.0.9) was solved with the standard MATLAB integrators `ode45` and `ode23s` up to the point $T = 10$ (default Tolerances). Explain the different dependence of the total number of steps from the parameter c observed in the figure.

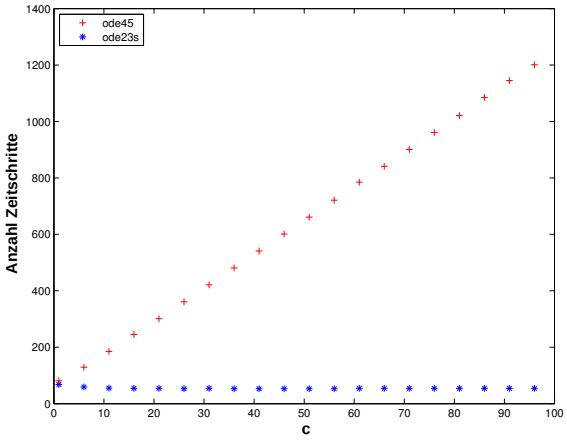


Fig. 44

SOLUTION for (12.3.c) → `12.3.3:Cros3h.pdf` ▲

(12.3.d) ⚀ Formulate the non-linear equation given by the implicit mid-point rule for the initial value problem (12.0.9).

SOLUTION for (12.3.d) → `12.3.4:Cros4h.pdf` ▲

(12.3.e) ⚀ Solve (12.0.9) with $\mathbf{a} = [1,0,0]^\top$, $c = 1$ up to $T = 10$. Use $N = 128$ uniform time steps of the implicit mid-point rule. Tabulate $|\mathbf{y}_k|_2$ for the sequence of approximate states generated by the implicit midpoint method. What do you observe?

HIDDEN HINT 1 for (12.3.e) → `12.3.5:CP1h.pdf`

SOLUTION for (12.3.e) → `12.3.5:Cros5h.pdf` ▲

(12.3.f) ⚁ The linear-implicit mid-point rule can be obtained by a simple linearization of the incremental equation of the implicit mid-point rule around the current solution value.

Give the defining equation of the linear-implicit mid-point rule for the general autonomous differential equation

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$$

with smooth f.

SOLUTION for (12.3.f) → `12.3.6:Cros6h.pdf` ▲

(12.3.g) ⚁ Implement the linear–implicit midpoint rule in the function:

```
std::vector<VectorXd> solve_lin_mid(const Function &f,
                                    const Jacobian &Jf,
                                    double T,
                                    const VectorXd & y0,
                                    unsigned int N);
```

Use this method to solve (12.0.9) with $\mathbf{a} = [1,0,0]^\top$, $c = 1$ up to $T = 10$ and $N = 128$. Tabulate $|\mathbf{y}_k|_2$ for the sequence of approximate states generated by the linear implicit midpoint method. What do you observe?

SOLUTION for (12.3.g) → `12.3.7:Cros7h.pdf` ▲

End Problem 12.3

Problem 12.4: Implicit Runge-Kutta method

This problem addresses the implementation of general implicit Runge-Kutta methods ➜Def. 12.3.18. We will adapt all routines developed for the explicit method to the implicit case. This problem assumes familiarity with ➜Section 12.3, and, especially, ➜Section 12.3.3 and ➜Rem. 12.3.24.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

Problem 11.6 introduced the class `RKIntegrator` that implemented the timestepping for a general explicit Runge-Kutta method according to ➜Def. 11.4.9. Keeping the interface we now extend this class so that it realizes a general Runge-Kutta timestepping method as given in ➜Def. 12.3.18 for solving the autonomous initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \mathbf{y}(0) = \mathbf{y}_0$.

(12.4.a) ⚀ Rederive the stage form ➜Eq. (12.3.23) of a general (implicit) Runge-Kutta method from the increment equations as given in ➜Def. 12.3.18.

SOLUTION for (12.4.a) → `12.4.1:0s.pdf` ▲

(12.4.b) ⚁ Let $\mathbf{g}_i$, $i = 1,\ldots,s$, be the so-called stages of a general Runge-Kutta method as defined in ➜Eq. (12.3.22). In ➜Rem. 12.3.24 the stages are recovered by solving a non-linear system of equations $F(\mathbf{g}) = \mathbf{0}$ with a suitable $F : \mathbb{R}^{sd} \to \mathbb{R}^{sd}$. Formulate the Newton iteration for it when only autonomous ODEs $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ with differentiable right-hand-side functions $\mathbf{f}$ are considered.

SOLUTION for (12.4.b) → `12.4.2:0as.pdf` ▲

(12.4.c) ⚂ By modifying the class `RKIntegrator` for the implementation of explicit Runge-Kutta methods, design a similar header-only C++ class `implicit_RKIntegrator` which implements a general implicit RK method given through a Butcher scheme ➜Eq. (12.3.20) to solve the autonomous initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \mathbf{y}(0) = \mathbf{y}_0$. The stages $\mathbf{g}_i$ as introduced in Sub-problem (12.4.a) are to be computed with the damped Newton method (see ➜Section 8.4.4) applied to the nonlinear system of equations satisfied by the stages (see ➜Rem. 12.3.21 and ➜Rem. 12.3.24). Use the provided code `dampnewton.hpp`, that is a simplified version of ➜Code 8.4.58. Note that we do not use the simplified Newton method as discussed in ➜Rem. 12.3.24.

In the code template you will find all the parts that you should implement. In fact, you only have to write the method `step` for the implicit RK.

SOLUTION for (12.4.c) → `12.4.3:Impl1h.pdf` ▲

(12.4.d) ⚁ Examine the code in `implicit_rk3prey.cpp`. Write down the complete Butcher scheme according to ➜Eq. (12.3.20) for the implicit Runge-Kutta method defined there. Which method is it? Is it A-stable ➜Def. 12.3.32, L-stable ➜Def. 12.3.38?

HIDDEN HINT 1 for (12.4.d) → `12.4.4:Impl1s.pdf` ▲

SOLUTION for (12.4.d) → `12.4.4:Impl2h.pdf`

(12.4.e) ⚀ Test your implementation `implicit_RKintegrator` of general implicit RK SSMs with the routine provided in the file `implicit_rk3prey.cpp` and comment on the observed order of convergence.

SOLUTION for (12.4.e) → `12.4.5:Impl3h.pdf` ▲

End Problem 12.4

Problem 12.5: Singly Diagonally Implicit Runge-Kutta Method

SDIRK methods (**S**ingly **D**iagonally **I**mplicit **R**unge-**K**utta methods) are distinguished by Butcher schemes of the particular form

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|ccccc} c_1 & \gamma & & \cdots & & 0 \\ c_2 & a_{21} & \ddots & & & \vdots \\ \vdots & \vdots & & & & \vdots \\ \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ c_s & a_{s1} & \cdots & & a_{s,s-1} & \gamma \\ \hline & b_1 & \cdots & & b_{s-1} & b_s \end{array}, \tag{12.0.18}$$

with $\gamma \neq 0$. In other words, the matrix $\mathfrak{A}$ of the Butcher scheme is lower triangular with constant diagonal.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

In this problem the scalar linear initial value problem of second order

$$\ddot{y} + \dot{y} + y = 0, \qquad y(0) = 1, \quad \dot{y}(0) = 0 \tag{12.0.19}$$

should be solved numerically using the SDIRK method described by the Butcher scheme

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1-\gamma & 1-2\gamma & \gamma \\ \hline & 1/2 & 1/2 \end{array}. \tag{12.0.20}$$

(12.5.a) ⚀ Explain the benefit of using SDIRK-SSMs compared to using Gauss-Radau RK-SSMs as introduced in ➜Ex. 12.3.44. In what situations will this benefit matter much?

HIDDEN HINT 1 for (12.5.a) → `12.5.1:SDIR0h.pdf` ▲

(12.5.b) ⚀ State the equations for the increments $\mathbf{k}_1$ and $\mathbf{k}_2$ of the Runge-Kutta method Eq. (12.0.20) applied to the initial value problem corresponding to the differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$.

SOLUTION for (12.5.b) → `12.5.2:SDIR1s.pdf` ▲

(12.5.c) ⚁ Show that the stability function $S(z)$ of the SDIRK method Eq. (12.0.20) is given by

$$S(z) = \frac{1 + z(1-2\gamma) + z^2(1/2 - 2\gamma + \gamma^2)}{(1-\gamma z)^2}$$

and plot the stability domain using the supplied MATLAB's function `stabdomSDIRK.m` or C++ code `stabdomSDIRK.cpp`.

SOLUTION for (12.5.c) → `12.5.3:SDIR2s.pdf` ▲

(12.5.d) ⚁ Find out whether for $\gamma = 1$ the SDIRK RK-SSM (12.0.20) is

- A–stable,
- L–stable.

HIDDEN HINT 1 for (12.5.d) → 12.5.4:SDIRxh.pdf

SOLUTION for (12.5.d) → 12.5.4:SDIRxs.pdf ▲

(12.5.e) ⊡ Formulate Eq. (12.0.19) as an initial value problem for a linear first order system for the function $\boldsymbol{z}(t) = (y(t), \dot{y}(t))^\top$.

SOLUTION for (12.5.e) → 12.5.5:SDIR3s.pdf ▲

(12.5.f) ⊡ Implement a C++ function

```
template <class StateType>
StateType sdirtkStep(const StateType & z0, double h, double gamma);
```

that realizes the numerical evolution of one step of the method Eq. (12.0.20) for the differential equation Eq. (12.0.19), starting from the value `z0` and returning the value of the next step of size `h`.

SOLUTION for (12.5.f) → 12.5.6:SDIR4s.pdf ▲

(12.5.g) ⊡ Use your C++ code to conduct a numerical experiment, which gives an indication of the order of the method (with $\gamma = \frac{3+\sqrt{3}}{6}$) for the initial value problem from Eq. (12.0.20). Choose $\boldsymbol{y}_0 = [1,0]^\top$ as initial value, `T=10` as end time and `N=20,40,80,...,10240` as steps.

SOLUTION for (12.5.g) → 12.5.7:SDIR5s.pdf ▲

End Problem 12.5

Problem 12.6: Stability of a Runge-Kutta method

This problem is devoted to an empirical study of stability problems haunting explicit Runge-Kutta single-step methods, recall the discussion in ➜Section 12.1.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

We focus on a 3-stage Runge-Kutta single step method described by the following Butcher-Tableau:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1/2 & 1/4 & 1/4 & 0 \\ \hline & 1/6 & 1/6 & 2/3 \end{array} \tag{12.0.25}$$

We also consider the following concrete case of the prey/predator model as introduced in ➜Ex. 11.1.9:

$$\begin{aligned} \dot{y}_1(t) &= (1 - y_2(t))y_1(t)\,, \\ \dot{y}_2(t) &= (y_1(t) - 1)y_2(t)\,. \end{aligned} \tag{12.0.26}$$

(12.6.a) ⚂ Create a C++ function

```
Vector2d predprey(Vector2d y0, double T, unsigned N)
```

that uses the RK-SSM (12.0.25) to solve an initial value problem for (12.0.26) with initial value $\mathbf{y}_0$ and N equidistant timestep up to final time $T > 0$. It should return $\mathbf{y}_N \approx \mathbf{y}(T)$.

SOLUTION for (12.6.a) → 12.6.1:Stab0s.pdf ▲

(12.6.b) ⊡

Write a C++ code to approximate the solution up to time $T = 1$ of the IVP for (12.0.26) with initial value $\mathbf{y}_0 = [100, 1]^\top$. Use the RK-SSM (12.0.25) and the function from Sub-problem (12.6.a).

Numerically determine the convergence order of the method for uniform steps of size $2^{-j}, j = 2, \ldots, 13$. As a reference solution, use an approximation with 2^{14} steps.

What do you notice for big step sizes? Try to find the maximum step size for which blow-up of the numerical solution can still be avoided.

HIDDEN HINT 1 for (12.6.b) → `12.6.2:Stab1h1.pdf`

HIDDEN HINT 2 for (12.6.b) → `12.6.2:Stab1h2.pdf`

SOLUTION for (12.6.b) → `12.6.2:Stab1s.pdf` ▲

(12.6.c) ⊡ Calculate the stability function $S(z)$, with $z = h\lambda$ and $\lambda \in \mathbb{C}$, of the method given by the table Eq. (12.0.25).

SOLUTION for (12.6.c) → `12.6.3:Stab2s.pdf` ▲

End Problem 12.6

Problem 12.7: Mono-implicit Runge-Kutta single step method

A so-called s-stage mono-implicit Runge-Kutta single step method (MIRK) for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^d \to \mathbb{R}^d$, is defined as:

$$\begin{aligned} \mathbf{g}_i &:= (1 - v_i)\mathbf{y}_0 + v_i\mathbf{y}_1 + h\sum_{j=1}^{i-1} d_{i,j}\mathbf{f}(\mathbf{g}_j), \quad i = 1, \ldots s\,, \\ \mathbf{y}_1 &:= \mathbf{y}_0 + h\sum_{j=1}^{s} b_j\mathbf{f}(\mathbf{g}_j) \end{aligned} \tag{12.0.29}$$

for suitable coefficients $v_i, b_i, d_{i,j} \in \mathbb{R}$, fixed to achieve a desired order.

Template: Get it on GitLab. **Solution:** Get it on GitLab.

[This problem involves implementation in C++]

(12.7.a) Single step methods defined as in Eq. (12.0.29) belong to the class of implicit Runge-Kutta methods ➜Def. 12.3.18. Write down the corresponding Butcher scheme ➜Eq. (12.3.20) in terms of the coefficients $v_i, b_i, d_{i,j}$.

HIDDEN HINT 1 for (12.7.a) → `12.7.1:mirk0.pdf`

SOLUTION for (12.7.a) → `12.7.1:MIRK1h.pdf`

▲

(12.7.b) Compute the stability function of a MIRK scheme defined as in Eq. (12.0.29) for $s = 2$.

HIDDEN HINT 1 for (12.7.b) → `12.7.2:MIRK1.pdf`

SOLUTION for (12.7.b) → `12.7.2:MIRK2h.pdf`

▲

(12.7.c) Now, we consider the special case of a scalar ODE ($d = 1$) and $s = 2$. Abbreviating $\mathbf{z} := [g_1, g_2, y_1]^\top$, rewrite Eq. (12.0.29) as a non-linear system of equations in the form $\mathbf{F}(\mathbf{z}) = \mathbf{0}$ for an explicitly specified suitable function $\mathbf{F} : \mathbb{R}^3 \to \mathbb{R}^3$.

SOLUTION for (12.7.c) → `12.7.3:MIRK3h.pdf`

▲

(12.7.d) Find the Jacobian $D\mathbf{F}(\mathbf{z})$ of the function $\mathbf{F}$ (in terms of $d_{i,j}, v_i, b_i$ and the derivative of f) from the previous sub-problem.

SOLUTION for (12.7.d) → `12.7.4:MIRK4h.pdf` ▲

In the next steps, we will implement the MIRK scheme for $d = 1, s = 2$.

(12.7.e) Implement a function

```
template <class Func, class Jac>
void newton2steps(const Func & F, const Jac & DF,
                  VectorXd & z);
```

that approximates the solution of $\mathbf{F}(\mathbf{z}) = \mathbf{0}$ by performing two steps of the Newton method applied to $\mathbf{F}(\mathbf{z}) = \mathbf{0}$. Use z to pass the initial guess and to return the approximated solution. Objects of type `Func` and `Jac` have to supply suitable evaluation operators `operator()`.

SOLUTION for (12.7.e) → `12.7.5:MIRK5h.pdf`

▲

(12.7.f) Consider the particular MIRK scheme given by the coefficients:

$$v_1 = 1, \quad v_2 = \frac{344}{2025}, \quad d_{21} = -\frac{164}{2025}, \quad b_1 = \frac{37}{82}, \quad b_2 = \frac{45}{82}. \tag{12.0.32}$$

Using the function `newton2steps`, implement a function

```
template <class Func, class Jac>
double MIRKstep(const Func & f, const Jac & df,
                double y0, double h);
```

that realizes one step of the MIRK scheme defined by Eq. (12.0.29) for $s = 2$ and a scalar ODE, that is, $d = 1$. The right hand side function $\mathbf{f}$ and its Jacobian are passed through f and df. The solution of the nonlinear system arising from Eq. (12.0.29) is approximated using two Newton steps, namely by using the function `newton2steps`. The initial guess has to be chosen appropriately!

SOLUTION for (12.7.f) → `12.7.6:MIRK6h.pdf` ▲

(12.7.g) Implement a function

```
template <class Func, class Jac>
double MIRKsolve(const Func & f, const Jac & df,
                 double y0, double T, unsigned int N);
```

for the solution of a scalar ODE up to time T, using N equidistant steps of the mono-implicit Runge-Kutta single step method defined by (12.0.32). The initial value is passed in y0.

SOLUTION for (12.7.g) → `12.7.7:MIRK7h.pdf` ▲

(12.7.h) Apply your implementation to the IVP

$$\dot{y} = 1 + y^2, y(0) = 0 \tag{12.0.35}$$

on $[0,1]$. The exact solution of (12.0.35) is $y_{ex}(t) := \tan t$. Compute the solution y_n at $T = 1$ with a sequence of uniform temporal meshes with $n = 4,\ldots,512$ intervals. Compute and output the error $|y_n(1) - y_{ex}(1)|$ and determine the rate of convergence of the scheme.

SOLUTION for (12.7.h) → `12.7.8:MIRK8h.pdf` ▲

End Problem 12.7

Problem 12.8: Extrapolation of evolution operators

In ➜§ 11.3.1 we have seen how discrete evolution operators can describe single-step methods for the numerical integration of ODEs. This task will study a way to combine discrete evolution operators of known order in order to build a single-step method with increased order. The method can be generalized to an extrapolation construction of higher-order single-step methods. These can be used for time-local stepsize control following the policy of ➜Rem. 11.5.16.

Template: Get it on GitLab.
Solution: Get it on GitLab.

[This problem involves implementation in C++]

Let Ψ^h define the discrete evolution of an order p Runge-Kutta single step methods for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$. We define a new evolution operator:

$$\tilde{\Psi}^h := \frac{1}{1-2^p}\left(\Psi^h - 2^p \cdot (\Psi^{h/2} \circ \Psi^{h/2})\right), \tag{12.0.37}$$

where $\circ$ denotes the composition of mappings.

(12.8.a) ⊡ If Ψ^h belongs to the explicit Euler methods, give the explicit formulas for $\tilde{\Psi}^h$.

SOLUTION for (12.8.a) → `12.8.1:ODESolve1s.pdf` ▲

(12.8.b) ⊡ Templates relevant for this problem are found in the template file `odesolve.cpp`. Implement a C++ function

```
using Vector = Eigen::VectorXd;

template <class DiscEvlOp>
Vector psitilde(const DiscEvlOp& Psi, unsigned int p,
          double h, const Vector & y0);
```

that returns $\tilde{\Psi}^h \mathbf{y}_0$ when given the underlying Ψ.

Objects of type `DiscEvlOp` must provide an evaluation operator:

```
Vector operator()(double h, const Vector &y);
```

providing the evaluation of $\Psi^h(\mathbf{y})$. A suitable C++ lambda function satisfies this requirement.

SOLUTION for (12.8.b) → `12.8.2:ODESolve2s.pdf` ▲

(12.8.c) ⊡ Implement a C++ function

```
template <class DiscEvlOp>
std::vector<Vector> odeintequi(const DiscEvlOp& Psi,
     double T, const Vector &y0, unsigned int N);
```

for the computation of an approximated solution given by the application of the evolution operator Ψ (given as `Psi`) on N equidistant time steps and with final time $T > 0$. The function returns the approximated value at each step (including y_0: $y_0, y_1, \ldots$) in a `std::vector<Vector>`.

HIDDEN HINT 1 for (12.8.c) → `12.8.3:ODESolve3h.pdf`

SOLUTION for (12.8.c) → `12.8.3:ODESolve3s.pdf` ▲

(12.8.d) ⊡ In the case of the IVP

$$\dot{y} = 1 + y^2, \quad y(0) = 0\,, \tag{12.0.40}$$

with exact solution $y(t) = \tan(t)$, determine empirically (using `odeintequi`) the order of the single step method induced by $\tilde{\Psi}^h$, when Ψ arises from the explicit Euler method. Monitor the error $|y_n(1) - y_{ex}(1)|$ at final time $T = 1$ for N uniforms steps with $N = 2^q, q = 2, \ldots, 12$.

HIDDEN HINT 1 for (12.8.d) → `12.8.4:ODESolve4h.pdf`

SOLUTION for (12.8.d) → `12.8.4:ODESolve4s.pdf` ▲

(12.8.e) ⊡ In general, the method defined by $\tilde{\Psi}^h$ has order $p + 1$. Thus, it can be used for adaptive timestep control and prediction.

Complete the implementation of a function

```
template <class DiscEvlOp>
std::vector<Vector> odeintssctrl(const DiscEvlOp& Psi,
                      double T, const Vector &y0,
                      double h0, unsigned int p,
                      double reltol, double abstol,
                      double hmin);
```

for the approximation of the solution of the IVP by means of adaptive timestepping based on Ψ and $\tilde{\Psi}$, where Ψ is passed through the argument `Psi`. Step rejection and stepsize correction and prediction is to be employed. The argument `T` supplies the final time, `y0` the initial state, `h0` an initial stepsize, `p` the order of the discrete evolution Ψ, `reltol` and `abstol` the respective tolerances, and `hmin` a minimal stepsize that will trigger premature termination. Compute the solution obtained using this function applied to the IVP (12.0.40) up to time $T = 1$ with the following data: `h0` $= 1/100$, `reltol` $= 10e-4$, `abstol` $= 10e-4$, `hmin` $= 10e-5$. Output the approximated solution at time $T = 1$.

HIDDEN HINT 1 for (12.8.e) → `12.8.5:EPh.pdf`

SOLUTION for (12.8.e) → `12.8.5:ODESolve5s.pdf` ▲

End Problem 12.8

Chapter 13

Structure Preserving Integration

Bibliography

[1] W. Hackbusch. *Hierarchische Matrizen. Algorithmen und Analysis*. Springer, Heidelberg, 2009.