

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 3-3: Linear Data Fitting

Abstract linear data fitting and its connection to the linear least squares problem [Lecture → Eq. (3.1.3.7)] is discussed in [Lecture → Ex. 3.0.1.1]. In this problem we practise these techniques for a concrete example.

This problem relies on the methods and EIGEN facilities introduced in [Lecture → Section 3.2] and [Lecture → Section 3.3.4].

For certain points in time t_i we have measured data f_i for a physical quantity f whose time-dependence $t \mapsto f(t)$ is only known qualitatively, that is, up to a few unknown parameters, see [Lecture → Ex. 3.0.1.1]:

t_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
f_i	100	34	17	12	9	6	5	4	4	2

$f(t)$ is assumed to be a linear combination

$$f(t) = \sum_{j=1}^4 \gamma_j \phi_j(t) \quad (3.3.1)$$

of the functions

$$\phi_1(t) = \frac{1}{t}, \quad \phi_2(t) = \frac{1}{t^2}, \quad \phi_3(t) = e^{-(t-1)}, \quad \phi_4(t) = e^{-2(t-1)}. \quad (3.3.2)$$

We aim for calculating the coefficients $\gamma_j, j = 1, 2, 3, 4$, such that

$$\sum_{i=1}^{10} (f(t_i) - f_i)^2 \longrightarrow \min. \quad (3.3.3)$$

(3-3.a)  (30 min.) In the file `linfit.hpp` write a C++ function

```
Eigen::VectorXd data_fit_normal(const Eigen::VectorXd &t, const
    Eigen::VectorXd &f);
```

that computes a least squares estimate for the coefficients γ_i by related to the data (3.3.2) and (3.3.3) by means of solving the *normal equations*.

SOLUTION of (3-3.a):

We have to recast (3.3.3) as a least squares problem arising from an overdetermined linear system of equations $\mathbf{Ax} = \mathbf{b}$. We find

$$\sum_{i=1}^{10} (f(t_i) - f_i)^2 = \sum_{i=1}^{10} \left(\sum_{j=1}^4 \gamma_j \phi_j(t_i) - f_i \right)^2,$$

which means

$$\mathbf{A} = \begin{bmatrix} \phi_1(t_1) & \phi_2(t_1) & \phi_3(t_1) & \phi_4(t_1) \\ \vdots & & & \vdots \\ \phi_1(t_{10}) & \phi_2(t_{10}) & \phi_3(t_{10}) & \phi_4(t_{10}) \end{bmatrix} \in \mathbb{R}^{10,4}.$$

First we look at the helper function `make_A`. This only creates the matrix $\mathbf{A}$ by applying the four model functions to the vector $\mathbf{t}$ and building up the matrix column wise. The argument vector $\mathbf{t}$ passes $[t_i]_{i=1}^{10}$.

C++-code 3.3.4: Creating the matrix A

```

2  Eigen::MatrixXd make_A(const Eigen::VectorXd &t) {
3      size_t n = b.size();
4      Eigen::MatrixXd A(n, 4);
5      // TODO (3-3.a) Build the matrix A
6      // Hint: Evaluate the functions \phi_j at the time points defined in b
7
8      // START
9      for (size_t i = 0; i < n; i++) {
10         double t_vec = t[i];
11         // Initialize row i of the matrix
12         A(i, 0) = 1.0 / t_vec;
13         A(i, 1) = 1.0 / (t_vec * t_vec);
14         A(i, 2) = std::exp(-(t_vec - 1.0));
15         A(i, 3) = std::exp(-2.0 * (t_vec - 1.0));
16     }
17     // END
18
19     return A;
20 }

```

Get it on  [GitLab \(linfit.hpp\)](https://gitlab.com/linfit/linfit.hpp).

In `data_fit_normal`, we use `make_A` to create the matrix. For the **normal equations**

$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b},$$

we calculate $\mathbf{A}^T \mathbf{A}$ and $\mathbf{A}^T \mathbf{b}$ and then solve the system using EIGEN's built-in dense linear system solver, see [Lecture → § 2.5.0.8].

C++-code 3.3.5: Solving the Normal Equations

```

2  Eigen::VectorXd data_fit_normal(const Eigen::VectorXd &t_vec,
3                                  const Eigen::VectorXd &b) {
4      // TODO (3-3.a) Solve normal equations to find the coefficients of the
5      // linear fitting
6      Eigen::VectorXd gamma(4);
7
8      // START
9      // Create matrix A
10     Eigen::MatrixXd A = make_A(t_vec);
11
12     // Transpose matrix A
13     Eigen::MatrixXd At = A.transpose();
14
15     // Compute LHS (A^T*A) of the normal equation
16     Eigen::MatrixXd AtA = At * A;
17
18     // Compute RHS (A^T*b) of the normal equation
19     Eigen::VectorXd Atb = At * b;
20
21     // Solve normal equation to get coefficients
22     gamma = AtA.ldlt().solve(Atb);
23     // END
24
25     return gamma;
26 }

```

Get it on  [GitLab \(linfit.hpp\)](https://gitlab.com/linfit/linfit.hpp).



(3-3.b) 🕒 (30 min.) Write a C++ function

```
Eigen::VectorXd data_fit_qr(const Eigen::VectorXd &t, const
    Eigen::VectorXd &f);
```

that uses *orthogonal transformation techniques* as introduced in [Lecture → Section 3.3.4] for the same task as in the previous sub-problem.

SOLUTION of (3-3.b):

In `data_fit_qr`, the first few steps are the same as in `data_fit_normal`. But instead of solving the normal equation, we compute the QR-decomposition and use this to solve the problem, see also [Lecture → Code 3.3.4.2].

C++-code 3.3.6: Solving with QR-Decomposition

```
2 Eigen::VectorXd data_fit_qr(const Eigen::VectorXd &t_vec,
3                             const Eigen::VectorXd &b) {
4     // TODO (3-3.b) Find the coefficients for the linear
5     // fitting by means of the QR decomposition of A
6     Eigen::VectorXd gamma(4);
7
8     // START
9     // Make matrix A
10    Eigen::MatrixXd A = make_A(t_vec);
11
12    // Compute QR decomposition
13    gamma = A.colPivHouseholderQr().solve(b);
14    // END
15
16    return gamma;
17 }
```

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



(3-3.c) 🕒 (15 min.) Write a C++ function

```
Eigen::VectorXd fitted_function(const Eigen::VectorXd &gamma,
    const Eigen::VectorXd &t_vec)
```

that takes the estimated parameters γ_i , $i = 1, 2, 3, 4$, as argument vector `gamma`, a vector `t_vec` of times τ_j , and returns the vector of values $f(\tau_j)$.

SOLUTION of (3-3.c):

We use the following code for this and the next part:

C++-code 3.3.7: `fitted_function()`: computing values of $f(t)$ using the estimated parameters

```

2  Eigen::VectorXd fitted_function(const Eigen::VectorXd &gamma,
3                                const Eigen::VectorXd &t_vec) {
4      // TODO (3-3.c): Define the data for the first plot by
5      // evaluating the function f at the grid defined by t_vec
6      unsigned int n = t_vec.size();
7      Eigen::VectorXd y(n);
8
9      // START
10     Eigen::MatrixXd A = make_A(t_vec);
11     y = A * gamma;
12     // END
13
14     return y;
15 }

```

Get it on  GitLab (linfit.hpp).

(3-3.d)  (15 min.) Using EIGEN write a C++ function

```
Eigen::VectorXd fitting_error(const Eigen::VectorXd &gamma);
```

that returns the vector

$$\left[(f(t_i) - f_i)^2 \right]_{i=1}^{10} \in \mathbb{R}^{10}$$

where f is defined according to (3.3.1) using the values for the γ_j -coefficients passed in the `gamma` argument.

SOLUTION of (3-3.d):

C++-code 3.3.8: `fitting_error()`: computing the deviation from measured data

```

2  Eigen::VectorXd fitting_error(const Eigen::VectorXd &gamma) {
3      // TODO (3-3.c): Compute the vector of squared errors of your
4      // fit at the provided sample points
5      Eigen::VectorXd err(10);
6
7      // START
8      Eigen::VectorXd t_vec = Eigen::VectorXd::LinSpaced(10, 0.1, 1.0);
9      Eigen::VectorXd f(10);
10     f << 100., 34., 17., 12., 9., 6., 5., 4., 4., 2.;
11
12     // Compute error
13     err = fitted_function(gamma, t_vec) - f;
14     err = err.cwiseProduct(err);
15     // END
16
17     return err;
18 }

```

Get it on  GitLab (linfit.hpp).

End Problem 3-3 , 90 min.