

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 3-7: 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.

This problem practises the solution of a linear least squares problem based on the normal-equation method, see [Lecture → Section 3.2].

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 $\mathbf{p}_i \in \mathbb{R}^2, i = 1, \dots, n$, we have to decide whether those point represent a stop sign, or a “priority road sign”. In this exercise we consider $n = 8$.

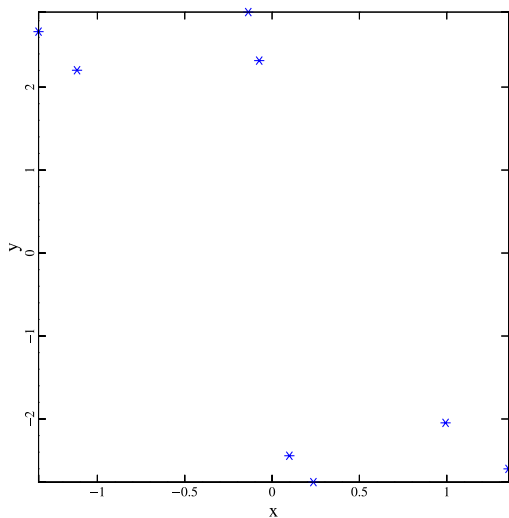


Fig. 11

A set of possible input points $\mathbf{p}^i$: corners detected in the photo.

Which traffic sign had been in the focus of the camera?



Fig. 12

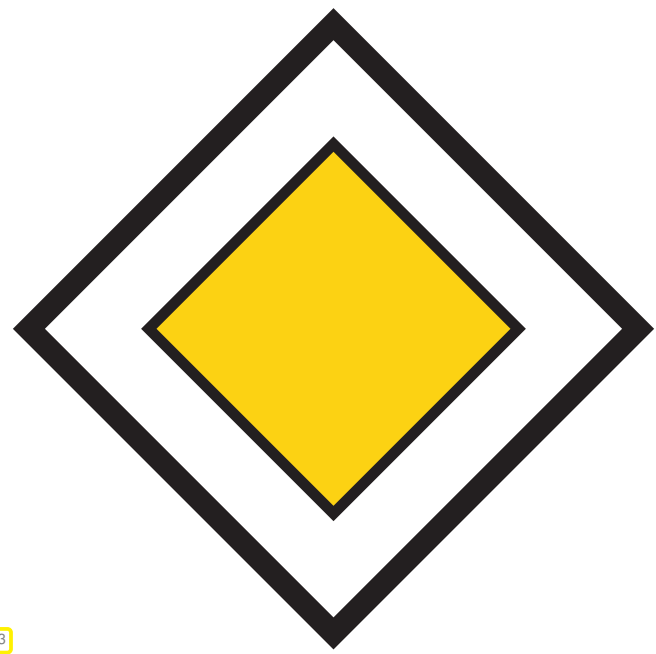


Fig. 13

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) $\mathbf{x}_{stop}^i \in \mathbb{R}^2$ resp. $\mathbf{x}_{priority}^i \in \mathbb{R}^2$ for $i = 1, \dots, 8$.

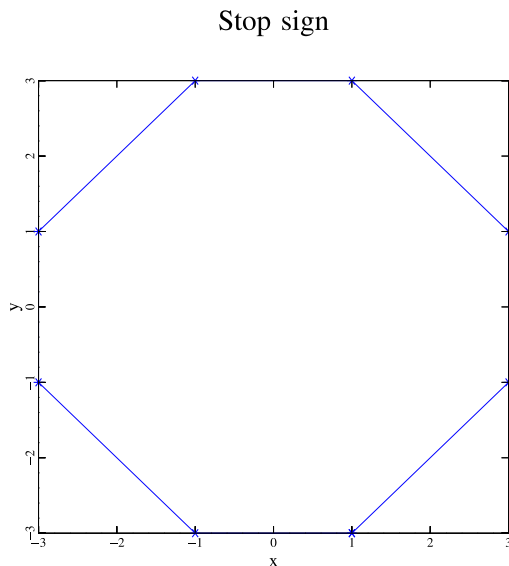


Fig. 14

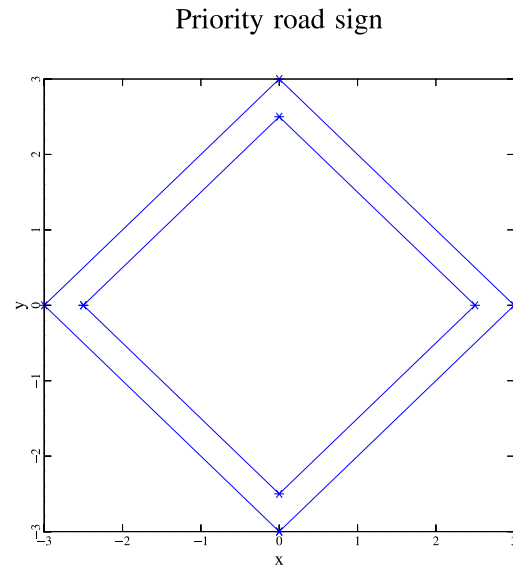


Fig. 15

The 8-points $\mathbf{x}_{stop}^i$ defining the model of a stop sign. The 8-points $\mathbf{x}_{priority}^i$ 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 $i = 1, \dots, 8$; i.e. we can assume that there exists a matrix $\mathbf{A} \in \mathbb{R}^{2,2}$, s.t.

$$\mathbf{p}^i = \mathbf{A}\mathbf{x}^i, i = 1, \dots, n. \quad (3.7.1)$$

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

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. (3.7.1) 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) ☹️ (20 min.) For the moment, assume we know the points $\mathbf{x}^i$ (i.e. we know the shape of our sign). Explicitly write (3.7.1) as an overdetermined linear system:

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

whose least-squares solution $\mathbf{v}^*$ will allow to determine the “best” linear transformation $\mathbf{A}$ (in the least-squares sense). How is the vector of unknowns related to $\mathbf{A}$?

SOLUTION of (3-7.a):

Let $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$. Then, for every $i = 1, \dots, n$:

$$\begin{bmatrix} p_1^i \\ p_2^i \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1^i \\ x_2^i \end{bmatrix} = \begin{bmatrix} a_{11}x_1^i + a_{12}x_2^i \\ a_{21}x_1^i + a_{22}x_2^i \end{bmatrix} \quad (3.7.2)$$

Since all entries of $\mathbf{A}$ are unknown, we want to write the above system as $\mathbf{p}^i = \mathbf{X}^i \mathbf{v}$, where $\mathbf{X}^i$ is given and:

$$\mathbf{v} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{bmatrix} \quad (3.7.3)$$

Rewriting (3.7.2):

$$\begin{bmatrix} p_1^i \\ p_2^i \end{bmatrix} = \begin{bmatrix} x_1^i & x_2^i & 0 & 0 \\ 0 & 0 & x_1^i & x_2^i \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{bmatrix} \quad (3.7.4)$$

Vertically stacking all the equations, for $i = 1, \dots, n$:

$$\underbrace{\begin{bmatrix} p_1^1 \\ p_2^1 \\ p_1^2 \\ p_2^2 \\ \vdots \\ p_1^n \\ p_2^n \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} x_1^1 & x_2^1 & 0 & 0 \\ 0 & 0 & x_1^1 & x_2^1 \\ x_1^2 & x_2^2 & 0 & 0 \\ 0 & 0 & x_1^2 & x_2^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^n & x_2^n & 0 & 0 \\ 0 & 0 & x_1^n & x_2^n \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{bmatrix}}_{\mathbf{v}}.$$

The matrix $\mathbf{B}$ is a $2n \times 4$ matrix, $\mathbf{w}$ is a $2n \times 1$ vector and $\mathbf{v}$ a 4×1 vector.

Note that the order of the matrix entries in the vector $\mathbf{v}$ can be different, of course, leading to a matrix $\mathbf{B}$ with permuted columns.



(3-7.b) (10 min.) When does the matrix $\mathbf{B}$ have full rank? Give a geometric interpretation of this condition.

SOLUTION of (3-7.b):

The matrix has $\text{rank}(\mathbf{B}) \leq 4$. The matrix has not full rank, if and only if its columns are linearly dependent, which is the case if and only if there exists a $c \in \mathbb{R}$ s.t. $x_2^i = cx_1^i$. This is equivalent to $\mathbf{x}^i = x_1^i [1, c]^\top$, i.e. all points $\mathbf{x}^i$ lie on the same line.



(3-7.c) (15 min.) [depends on Sub-problem (3-7.a)]

In the file `shape_ident.hpp` implement a C++ function

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

that returns the matrix **B**. Pass the vectors $\mathbf{x}^i$ in the columns of a $2 \times n$ EIGEN matrix **X**.

SOLUTION of (3-7.c):

C++-code 3.7.5: Building the system matrix **B**.

```

2  Eigen::MatrixXd shape_ident_matrix(const Eigen::MatrixXd &X) {
3      assert(X.rows() == 2 && "X must have 2 rows!");
4
5      unsigned int n = X.cols();
6      Eigen::MatrixXd B = Eigen::MatrixXd::Zero(2 * n, 4);
7
8      // TODO: (3-7.c) Build system matrix B.
9
10     // START
11     for (unsigned int row = 0; row < n; ++row) {
12         // Odd row, first two columns
13         B(2 * row, 0) = X(0, row);
14         B(2 * row, 1) = X(1, row);
15         // Even row, last two columns
16         B(2 * row + 1, 2) = X(0, row);
17         B(2 * row + 1, 3) = X(1, row);
18     }
19     // END
20
21     return B;
22 }

```

Get it on  GitLab (shape_ident.hpp).



(3-7.d) (30 min.) [depends on Sub-problem (3-7.c)]

In the file `shape_ident.hpp` implement a C++ function

```

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

```

that computes the matrix **A** by solving the least squares problem based on the **normal equations**, see [Lecture → 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 of (3-7.d):

The following code demonstrates advanced use of EIGEN's facilities. Of course, everything can be done "in a pedestrian way". Nevertheless it is worth while trying to understand that very compact computation of the matrix **A**.

C++-cde 3.7.6: Solving the least square problem.

```

2  double solve_lsq(const Eigen::MatrixXd &X, const Eigen::MatrixXd &P,
3                  Eigen::MatrixXd &A) {
4      assert(X.rows() == 2 && "X must have 2 rows!");

```

```

5  assert(P.rows() == 2 && "P must have 2 rows!");
6  assert(X.cols() == P.cols() && "P and X must have same size!");
7
8  unsigned n = X.cols();
9  double norm_of_residual = 0.;
10
11 // TODO: (3-7.d) solve LSQ problem, return best linear approximation
12 // and
13 // residual
14
15 // START
16 // Build system matrix
17 Eigen::MatrixXd B = shape_ident_matrix(X);
18
19 // Solve LSQ system using normal equation
20 // We need to do some reshaping in order to properly set up the
21 // system
22 A = Eigen::Map<Eigen::MatrixXd>(
23     Eigen::MatrixXd(
24         // Solve LSQ problem using normal equation
25         (B.transpose() * B)
26         .ldlt()
27         .solve(B.transpose() *
28             // Need to vectorize matrix
29             Eigen::Map<const Eigen::MatrixXd>(P.data(), 2 * n, 1))
30         // Pass an array to "Eigen::Map"
31         )
32         .data(),
33         // Need to reshape vector to 2x2 matrix
34         2, 2)
35         .transpose();
36
37 // Residual: must reshape P
38 norm_of_residual =
39     (Eigen::Map<const Eigen::MatrixXd>(P.data(), 2, n) - A * X).norm();
40 // END
41
42 return norm_of_residual;
43 }

```

Get it on  GitLab (shape_ident.hpp).



(3-7.e) (30 min.) [depends on Sub-problem (3-7.d)]

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 Eigen::MatrixXd Xstop,
              const Eigen::MatrixXd Xpriority,
              const Eigen::MatrixXd & P,
              Eigen::MatrixXd & A);

```

that identifies 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” $\mathbf{x}_{stop}^i$ resp. $\mathbf{x}_{priority}^i$ are passed through Xstop resp. Xpriority.

SOLUTION of (3-7.e):

We can solve the least square problem for both sets of points $\mathbf{x}_{stop}^i$ and $\mathbf{x}_{priority}^i$. It is reasonable to assume that the solution with the smallest residual error is the correct one. Of course this is just an heuristic argument.

C++-code 3.7.7: Code for shape identification.

```

2 Shape identify(const Eigen::MatrixXd Xstop, const Eigen::MatrixXd Xpriority,
3               const Eigen::MatrixXd &P, Eigen::MatrixXd &A) {
4     // TODO: (3-7.e) Use residual do decide wether shape defines stop or
5     // priority
6     // road sign.
7
8     // START
9     // Best linear transformation for both "model" shape
10    Eigen::MatrixXd Astop, Apriority;
11
12    // Residuals and linear transform for both "models"
13    double res_stop = solve_lsq(Xstop, P, Astop);
14    double res_priority = solve_lsq(Xpriority, P, Apriority);
15
16    // If residual with stop Model is bigger than the one with priority
17    // sign,
18    // probably it it is priority sign, otherwise is a stop sign.
19    if (res_stop <= res_priority) {
20        A = Astop;
21        return Stop;
22    } else {
23        A = Apriority;
24        return Priority;
25    }
26    // END
27 }

```

Get it on  GitLab (shape_ident.hpp).



End Problem 3-7 , 105 min.