

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 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](#).

We consider the C++ function defined in `getit.cpp` (cf. Code 1.9.1). [Get it on 🐙 GitLab \(getit.cpp\)](#).

C++11-code 1.9.1: 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 VectorXd &V = eig.eigenvalues();
6      const MatrixXd &W = eig.eigenvectors();
7
8      VectorXd cx = x.cast<std::complex<double>>();
9
10     VectorXd 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 $Wx = cx$ 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 of (1-9.a):

The output is y s.t. $y = A^k x$. In fact, the eigenvalue decomposition of the matrix $A = SDW^{-1}$ (where D is diagonal and W invertible), allows us to write:

$$A^k = (WDW^{-1})^k = WD^k W^{-1}$$

D^k can be computed efficiently (component-wise) for diagonal matrices.

The code `getit.cpp` provides a line-by-line explanation. [Get it on 🐙 GitLab \(getit.cpp\)](#).



(1-9.b)  Fix $k \in \mathbb{N}$. Discuss (in detail) the asymptotic complexity of `getit` for $n \rightarrow \infty$.

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

SOLUTION of (1-9.b):

The algorithm comprises the following operations:

- Diagonalization of a full-rank matrix $\mathbf{A}$ is $O(n^3)$.
- Matrix-vector multiplication is $O(n^2)$.
- Vector-vector componentwise multiplication is $O(n)$.
- Raising a vector in $\mathbb{R}^n$ for the power k has complexity $O(n)$.
- Solve a fully determined linear system: $O(n^3)$.

The complexity of the algorithm is dominated by the operations with higher exponent. Therefore the total complexity of the algorithm is $O(n^3)$ for $n \rightarrow \infty$.



End Problem 1-9