Dirk Praetorius,
Gregor Gantner

Sommersemester 2018
26.06.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 13

Hint. We consider the class `Matrix` from the lecture (slide 357) together with its derived class `SquareMatrix` (slide 366 resp. Exercise 12.3). From the `SquareMatrix`, we have later derived the classes `LowerTriangularMatrix` (slide 373) and `DiagonalMatrix` (Exercise 12.6).

Aufgabe 13.1. Not every matrix $A \in \mathbb{R}^{n\times n}$ has a normalized LU factorization $A = LU$, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

If such a factorization exists, it holds that

$$\begin{aligned} u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{for } i = 1, \dots, n, \quad k = i, \dots, n, \\ \ell_{ki} &= \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{for } i = 1, \dots, n, \quad k = i+1, \dots, n, \\ \ell_{ii} &= 1 \quad \text{for } i = 1, \dots, n, \end{aligned}$$

Use the formula for the matrix-matrix multiplication to derive these formulae. Implement for the class `SquareMatrix` the method `computeLU`, which computes and returns the LU factorization of a square matrix $A \in \mathbb{R}^{n\times n}$. The method returns a matrix $R \in \mathbb{R}^{n\times n}$ of type `SquareMatrix`, whose upper and lower triangular parts contain the entries of L and U

$$R = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ \ell_{21} & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ \ell_{n1} & \dots & \ell_{n,n-1} & u_{nn} \end{pmatrix}.$$

The diagonal of L does not need to be stored. Why? Test your implementation appropriately!

Aufgabe 13.2. What is the computational cost of your implementation of the LU factorization from Exercise 13.1? Use the $\mathcal{O}$-notation to write the result and justify your answer.

Aufgabe 13.3. The determinant of a matrix $A \in \mathbb{R}^{n\times n}$ can be computed exploiting the normalized LU factorization from Exercise 13.1: Indeed, it holds that $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Extend the class `SquareMatrix` by the method `det`, which computes and returns the determinant. The matrix A should not be overwritten. Test your implementation in an appropriate way!

Aufgabe 13.4. Extend the class `SquareMatrix` by the method `solve`, which computes the solution of a linear system of equations of the form $Ax = b$ according to the following strategy: First, compute the LU factorization $A = LU$. Then, then solve the system $Ly = b$ and finally $Ux = y$. Test your implementation accurately!

Aufgabe 13.5. A matrix $S \in \mathbb{R}^{n \times n}$ is skewsymmetric if it holds that $S^T = -S$. Derive from the class `SquareMatrix` the class `SkewSymmetricMatrix`. Use a vector of length $n(n-1)/2$ to store the matrix entries. Why is this sufficient? Implement constructors, type casting and a suitable access to the coefficients. For the diagonal entries, proceed as done for the class `LowerTriangularMatrix` from the lecture: Store `double zero` and `double const_zero` and use them for the coefficient access. Test your implementation accurately!

Aufgabe 13.6. Write a template function `pot(T x, int n)`, which computes for an arbitrary datatype (which supports `operator*` and `operator/`) the function x^n. Test your example with varying datatypes. Save your source code as `pot.cpp` into the directory `serie13`..
Hint: Note that x^n should also work for negative n. You may use that `x/x` corresponds to the neutral element of `operator*` of the type `T`.

Aufgabe 13.7. Explain the differences between `public`-, `private`-, und `protected`-inheritance on the basis of a suitable exapmle.

Aufgabe 13.8. What is the system of floating-point numbers? Which parts does a floating-point number consist of? How can you determine its value? What is the meaning of `Inf`, `-Inf`, and `NaN`? What is a normalized floating-point number? What is an implicit leading bit? Which are the values of the largest and the smallest positive normalized floating point number in the `float`-system $\mathbb{F}(2, 24, -126, 127)$?