Dirk Praetorius
Michele Ruggeri

Summer semester 2019
June 21, 2019

Scientific programming in mathematics

Exercise sheet 13

Inheritance

The following exercises consider the class `Matrix` from the lecture (slides 343–350) and its derived classes `Vector`, `SquareMatrix`, and `LowerTriangularMatrix`. The corresponding source code can be downloaded from TUWEL.

Exercise 13.1. A diagonal matrix is a square matrix in which all entries outside the main diagonal are zero, i.e., $a_{ij} = 0$ for all $i \neq j$. Derive the class `DiagonalMatrix` from the class `SquareMatrix`. Only the diagonal entries of the matrix must be stored. Implement constructors, type casting and a suitable access to the coefficients. For each entry a_{ij} with $i \neq j$ proceed in the same way as done for the class `LowerTriangularMatrix` from the lecture: Save `double zero` and `double const_zero` and use it when you need to access the coefficients. Test your implementation appropriately!

Exercise 13.2. A matrix $S \in \mathbb{R}^{n\times n}$ is symmetric if it holds that $S^T = S$. Derive from the class `SquareMatrix` from the lecture the class `SymmetricMatrix`. 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. Test your implementation appropriately!

Exercise 13.3. A matrix $S \in \mathbb{R}^{n\times n}$ is skewsymmetric if it holds that $S^T = -S$. Derive from the class `SquareMatrix` from the lecture 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 appropriately!

Exercise 13.4. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric (i.e., $A^T = A$) and positive definite matrix (i.e., it holds that $Ax \cdot x > 0$ for any $x \in \mathbb{R}^n \setminus \{0\}$). Then, there exists a unique lower triangular matrix $L \in \mathbb{R}^{n\times n}$ with diagonal entries $\ell_{jj} > 0$ for all $j = 1 \dots, n$ such that $A = LL^T$. This decomposition is usually referred to as *Cholesky decomposition*. Use the formula of the matrix-matrix multiplication $A = LL^T$ to derive a formula for the entries of L. Extend the class `SymmetricMatrix` from Exercise 13.2 by the method

```
const LowerTriangularMatrix computeCholesky() const
```

which computes and returns the Cholesky decomposition of a symmetric and positive definie matrix $A \in \mathbb{R}^{n\times n}$. Note that the return value $L \in \mathbb{R}^{n\times n}$ should be of type `LowerTriangularMatrix`. Test your implementation appropriately!

Exercise 13.5. The *Gaussian elimination* method is an important algorithm for solving systems of linear equations. Let $A \in \mathbb{R}^{n\times n}$ be a matrix and $b \in \mathbb{R}^n$ a vector. The computation of the solution $x \in \mathbb{R}^n$ of the linear system $Ax = b$ via Gaussian elimination consists of two main steps:

- First of all, the matrix A is converted into an equivalent upper triangular matrix. Note that also the right-hand side vector b must be modified accordingly. More precisely, at the beginning, the matrix is in general fully populated

$$A = A^{(0)} = \begin{pmatrix} a_{1,1}^{(0)} & \dots & a_{1,n}^{(0)} \\ \vdots & \vdots & \vdots \\ a_{n,1}^{(0)} & \dots & a_{n,n}^{(0)} \end{pmatrix}.$$

During the first elimination step, an appropriate multiple of the first row of the matrix is subtracted from the remaining rows in order to obtain a matrix of the form

$$A^{(1)} = \begin{pmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \dots & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(1)} & \dots & a_{2,n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n,2}^{(1)} & \dots & a_{n,n}^{(1)} \end{pmatrix}.$$

In the second elimination step, an appropriate multiple of the second row of the matrix is subtracted from the remaining rows in order to obtain a matrix of the form

$$A^{(2)} = \begin{pmatrix} a_{1,1}^{(2)} & a_{1,2}^{(2)} & a_{1,3}^{(2)} & \dots & a_{1,n}^{(2)} \\ 0 & a_{2,2}^{(2)} & a_{2,3}^{(2)} & \dots & a_{2,n}^{(2)} \\ 0 & 0 & a_{3,3}^{(2)} & \dots & a_{2,n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n,3}^{(2)} & \dots & a_{n,n}^{(2)} \end{pmatrix}.$$

After $n-1$ elimination steps, one obtains an upper triangular matrix

$$A^{(n-1)} = \begin{pmatrix} a_{1,1}^{(n-1)} & \dots & \dots & \dots & a_{1,n}^{(n-1)} \\ 0 & a_{2,2}^{(n-1)} & \dots & \dots & a_{2,n}^{(n-1)} \\ 0 & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & a_{n-1,n-1}^{(n-1)} & a_{n-1,n}^{(n-1)} \\ 0 & 0 & \dots & 0 & a_{n,n}^{(n-1)} \end{pmatrix}.$$

- The resulting system, characterized by an upper triangular matrix A, is then solved directly.

Overload the operator `|` so that typing `x = A | b` for a square matrix $A \in \mathbb{R}^{n \times n}$ (type `SquareMatrix`) and a vector $b \in \mathbb{R}^n$ (type `Vector`) computes and returns the solution $x \in \mathbb{R}^n$ of the system $Ax = b$ (type `Vector`). Use `assert` to ensure that, in the k-th elimination step, the condition $a_{kk}^{(k)} \neq 0$ is satisfied. Do not forget that also the right-hand side vector $b \in \mathbb{R}^n$ must be modified accordingly. Solve the system for an upper triangular matrix directly (derive an appropriate formula using the formula for the matrix-vector product and the simplifications thereof which follow from the triangular structure of the matrix). Test your implementation appropriately!
Optional extension (1 bonus point). The Gaussian elimination algorithm fails when it happens that $a_{kk}^{(k)} = 0$ in the k-th elimination step. This can happen even if the linear system $Ax = b$ admits a unique solution x. To avoid this, the algorithm is usually extended by the so-called *pivoting*:

- During the k-th step, choose amongst $a_{kk}^{(k)}, \dots, a_{nk}^{(k)}$ the element $a_{pk}^{(k)}$ with the largest absolute value (the so-called pivot).
- Swap the k-th and the p-th row of $A^{(k)}$ (and $b^{(k)}$).
- Perform the elimination step as before.

Improve your implementation of the Gaussian elimination method by adding pivoting to the elimination step.

Exercise 13.6. For lower triangular matrices (resp., diagonal matrices) $A \in \mathbb{R}^{n \times n}$, it is possible to derive direct formulae for the computation of the solution $x \in \mathbb{R}^n$ of the linear system of equations $Ax = b$ with $b \in \mathbb{R}^n$. The derivation of such formulae exploits the triangular (resp., diagonal) structure of the matrix. In Exercise 13.5 you have implemented the capability to compute the solution $x \in \mathbb{R}^n$ (stored in an object `x` of type `Vector`) of the linear system of equations $Ax = b$ for a square matrix $A \in \mathbb{R}^{n \times n}$ (stored in an object `A` of type `SquareMatrix`) and a vector $b \in \mathbb{R}^n$ (stored in an object `b` of

type `Vector`) via `x = A | b`. The solution of the system was based on the Gaussian elimination method. Overload the operator `|` so that, for a lower triangular matrix $A \in \mathbb{R}^{n \times n}$ (stored in an object `A` of type `LowerTriangularMatrix` from the lecture) and for a diagonal matrix $A \in \mathbb{R}^{n \times n}$ (stored in an object `A` of type `DiagonalMatrix` from Exercise 13.1), the computation of the solution via `x = A | b` is not based on the Gaussian elimination method, but rather on appropriate direct formulae. Test your implementation appropriately! What was the computational complexity of your implementation from Exercise 13.5 for a general square matrix $A \in \mathbb{R}^{n \times n}$? What is the computational complexity of the current implementation for a lower triangular matrix or a diagonal matrix $A \in \mathbb{R}^{n \times n}$? Use the $\mathcal{O}$ notation to write the results and justify your answers.

Exercise 13.7. Let $A \in \mathbb{R}^{n \times n}$ be a square matrix. The Laplace formula for determinants states that, for each $j \in \{1, \ldots, n\}$, it holds that

$$\det A = \sum_{i=1}^{n} (-1)^{i+j} \cdot a_{ij} \cdot \det A_{ij},$$

where A_{ij} denotes the $(n-1) \times (n-1)$-submatrix of A obtained by removing from A its i-th row and its j-th column. Extend the class `SquareMatrix` by the recursive method `double det() const`, which computes and returns the determinant of a square matrix using the Laplace formula. Test your implementation appropriately!

Exercise 13.8. Derive a simple formula for the computation of the determinant of a lower triangular (resp., diagonal) matrix using the Laplace formula from Exercise 13.7 and the simplifications thereof which follow from the triangular (resp., diagonal) structure of the matrix. Redefine the method `det` from Exercise 13.7 for objects of type `LowerTriangularMatrix` from the lecture and `DiagonalMatrix` from Exercise 13.1 so that for lower triangular and diagonal matrices the computation of the determinant does not use the recursive implementation of Exercise 13.7, but rather the simple formula you derived. Test your implementation appropriately!