Dirk Praetorius
Michele Ruggeri

Wintersemester 2014/15
02.12.2014

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Aufgabe 10.1. The *bubblesort* algorithm is an inefficient, but short sorting algorithm which works as follows: You run through the entries of a given vector $x \in \mathbb{R}^n$ several times. For every run, each entry x_j of x is compared to its successor x_{j+1}. If $x_j > x_{j+1}$, then the two entries x_j and x_{j+1} are swapped. After the first complete run through the vector, one knows that (at least) the last element is sorted correctly, i.e. the last element x_n is the maximum of the vector. Thus, in the next run one only has to go up-to the last-but-one entry of the vector. How many loops do you need for this algorithm? Write a function `bubblesort` which sorts a given vector $x \in \mathbb{R}^n$ with this algorithm. Additionally, write a main program that reads in $x \in \mathbb{R}^n$ and sorts it. The length n should be constant. However, your function `bubblesort` should be programmed for arbitrary lengths n. Save your source code as `bubblesort.c` into the directory `serie10`.

Aufgabe 10.2. An upper triangular matrix $U \in \mathbb{R}^{n\times n}$ has at most $\frac{n(n+1)}{2} = \sum_{j=1}^n j$ nontrivial coefficients. Write a structure `matrixU` to save the dimension $n \in \mathbb{N}$ and the coefficients U_{ij} (in a dynamical vector of length $\frac{n(n+1)}{2}$). Write all necessary functions to work with the structure (`newMatrixU`, `delMatrixU`, `getMatrixUDimension`, `getMatrixUij`, `setMatrixUij`). In which entry u_ℓ of the dynamical vector should the coefficient U_{ij} be saved? Hint: Save U columnwise.

Aufgabe 10.3. Write a function `mvmU` which, given a vector $x \in \mathbb{R}^n$, performs the matrix-vector multiplication with an upper triangular matrix $U \in \mathbb{R}^{n\times n}$. For the matrix U use the structure from Exercise 10.2, for the vector $x \in \mathbb{R}^n$ and for the product, use the structure from the lecture. Exploit the special nature of the matrix U, i.e., unnecessary products with trivial coefficients of the matrix must be avoided.

Aufgabe 10.4. Let $U \in \mathbb{R}^{n\times n}$ be an upper triangular Matrix with $U_{jj} \neq 0$ for all $j = 1, \dots, n$. Given a vector $b \in \mathbb{R}^n$, there exists a unique solution $x \in \mathbb{R}^n$ of the system $Ux = b$. Starting from the formula for the matrix-vector multiplication, derive a formula for x (Hint: Write the formula of the matrix-vector product $b = Ux$ for b_j, $j = 1, \dots, n$, as a sum, and observe how the nature of U simplifies the indices of the sum). Write a function `solveU`, which, given U and b, computes and returns the vector x. For the matrix U use the structure from Exercise 10.2. For the vectors $b, x \in \mathbb{R}^n$, use the structure introduced in the lecture. To check you inplementation, write a main program which reads an upper triangular matrix U and a vector x and performs the matrix-vector multiplication $b = Ux$ by using the function from Exercise 10.3. With the computed right-hand side b, the program then calls the function `solveU` to solve the system $U\widetilde{x} = b$. The solution $\widetilde{x}$ should agree with the initial vector x.

Aufgabe 10.5. A matrix $A \in \mathbb{R}^{n\times n}$ admits a normalized a normalized LU-factorization $A = LU$ if

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

If A admits a normalized LU-factorization, it holds

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{for } i = 1, \dots, n, \quad k = i, \dots, n,$$

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

$$\ell_{ii} = 1 \quad \text{for } i = 1, \dots, n.$$

The remaining coefficients of $L, U \in \mathbb{R}^{n\times n}$ are zero. This can be easily shown from the matrix-matrix multiplication formula. Write a function `computeLU`, which computes and returns the LU-factorization of A. To use the above formulae, compute the coefficients of L and U in an appropriate order. Save your source code as `computeLU.c` into the directory `serie10`.

Aufgabe 10.6. Let $A \in \mathbb{R}^{n\times n}$ be a tridiagonal matrix, i.e.,

$$\begin{pmatrix} a_{1,1} & a_{1,2} & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & \\ & a_{3,2} & a_{3,3} & \ddots & \\ & & \ddots & \ddots & a_{n-1,n} \\ & & & a_{n,n-1} & a_{n,n}, \end{pmatrix}$$

for which there exists a LU-factorization. Determine how the formulae for the coefficients L and U from Exercise 10.5 simplifies in this special case. Then, write a function `computeLU3` which computes the LU-factorization of A without unnecessary operations, i.e., unnecessary sums/products of trivials coefficients must be avoided and only the nontrivial coefficients of L and U must be computed.

Aufgabe 10.7. The Laplace formula states that for each $j \in \{1, \dots, n\}$ it holds

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

where A_{ij} is the $(n-1)\times(n-1)$-submatrix of A obtained by removing the i-th row and the j-th column from A. Write a function `detlaplace`, which applies the Laplace formula to compute the determinant $\det(A)$ of a matrix $A \in \mathbb{R}^{n\times n}$. Save your source code as `detlaplace.c` into the directory `serie10`.

Aufgabe 10.8. To compute the determinant of a matrix $A \in \mathbb{R}^{n\times n}$, using the Laplace formula from Exercise 10.7 is the best idea (Why? Try it!). It is better to compute the normalized LU-factorization from Exercise 10.5. Indeed, it holds $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Write a function `det(A)`, which computes the determinant of a matrix A exploiting its normalized LU-factorization.