Dirk Praetorius
Michele Ruggeri

Sommersemester 2015
12.05.2015

**Übungen zur Vorlesung**
**Einführung in das Programmieren für TM**

**Serie 8**

**Aufgabe 8.1.** For a continuous Integrand $f : [a, b] \to \mathbb{R}$ the integral $I := \int_a^b f\,dx$ can be approximated numerically by the so-called *trapezoidal rule*. For a given $n \in \mathbb{N}$ and $h := (b-a)/n$ calculate

$$I_n := \frac{h}{2}\Big(f(a) + 2\sum_{j=1}^{n-1} f(a+jh) + f(b)\Big). \tag{1}$$

In the case of a piecewise affine function $p$ with $p(a+jh) = f(a+jh)$ we have $I_n = I$. Write a function `trapecoidalrule(f,a,b,tau)` that calculates the approximating sequence $I_n$, until

$$|I_n - I_{n-1}| \le \begin{cases} \tau & \text{für } |I_n| \le \tau, \\ \tau\,|I_n| & \text{anderenfalls.} \end{cases}$$

Return the whole approximating sequence. Test your functions with $f(x) = \exp(x)$ on $[0, 10]$. Print a tabular on screen in which you compare $n$ and the corresponding error $|I - I_n|$ and the experimental rate of convergence. Save your source code as `trapezoidal.c` into the directory `serie08`.

**Aufgabe 8.2.** Write a structure `CVector` for the storage of vector with complex coefficients. Use the structure `cdouble` from Exercise 7.1 and Exercise 7.2. Moreover, implement the functions `newCVector`, `delCVector`, `getCVectorLength`, `getCVectorEntry`, `setCVectorEntry`. Save your source code as `cvector.c` into the directory `serie08`.

**Aufgabe 8.3.** Write a function `CVectorVector`, which, given two complex vectors $x, y \in \mathbb{C}^n$, computes the scalar product $x \cdot y := \sum_{j=1}^n x_j \overline{y_j}$ . Use the structure `CVector` from Exercise 7.1 and Exercise 7.2. Then, write a main program, which reads two complex vectors $x, y \in \mathbb{C}^n$ from the keyboard and displays the value of the scalar product $x \cdot y \in \mathbb{C}$. Save your source code as `CVectorVector.c` into the directory `serie08`. Test your code on a suitable example.

**Aufgabe 8.4.** Write a structure `CMatrix` for the storage of $m \times n$-matrices $A \in \mathbb{C}^{m\times n}$ with complex entries. Use the structure `cdouble` from Exercise 7.1 and Exercise 7.2. Furthermore, write the functions `newCMatrix`, `delCMatrix`, `getCMatrixM`, `getCMatrixN`, `getCMatrixCoeff`, `setCMatrixCoeff`. Save your source code as `CMatrix.c` into the directory `serie08`.

**Aufgabe 8.5.** Write a function `cmatrixvector`, which, for given complex matrix $A \in \mathbb{C}^{m\times n}$ and a complex vector $x \in \mathbb{C}^n$, calculates the Matrix-Vector-product $Ax \in \mathbb{C}^m$. For calculating with the coefficients use Exercise 7.1 and Exercise 7.2. Save your source code as `cmatrixvector.c` into the directory `serie08`. Test your code on a suitable example.

**Aufgabe 8.6.** Write a structure `Matrix` to save quadratic $n \times n$ `double` matrices. Distinguish between fully-populated matrices (type `0`), lower triangle matrices (type `'L'`) and upper triangle matrices (type `'U'`). A lower triangular matrix $L$ and an upper triangular matrix $U$ have the following polulation structure:

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix} \qquad L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

We thus have $u_{jk} = 0$, if $j > k$ and $\ell_{jk} = 0$, if $j < k$. A fully populated matrix should by stored in Fortran-Style- therefore columnwise in a dynamical vector with $n \cdot n$ entries. triangle-matrices should be stored in a vector with $\sum_{j=1}^{n} j = n(n+1)/2$ entries. Write all the necessary functions to work with this structure ((`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Save your source code as `matrix.c` into the directory `serie08`. (Hint: The functions `getMatrixEntry` and `setMatrixEntry` depend on the type of the matrix.)

**Aufgabe 8.7.** Write a function `columnsumnorm.c`, which, for a given matrix $A \in \mathbb{R}^{n \times n}$, calculates and returns the absolute column sum norm

$$\|A\|_S := \max_{j=1,\dots,n} \sum_{i=1}^{n} |A_{ij}|$$

$A$ is stored in the structure from Exercise 8.6. if $A$ is a triangular matrix, exploit the population structure of $A$. Save your source code as `columnsumnorm` into the directory `serie08`. Testen Sie Ihren Code an einem geeigneten Beispiel.

**Aufgabe 8.8.** Let `squareMatrix` be a structure data-type for the storage of quadratic matrices $A \in \mathbb{R}^{n \times n}$. The structure contains the dimension $n \in \mathbb{N}$ and the entries given as `double*`, i.e., the entries of the matrix is stored columnwise. The functions `newSquareMatrix, delSquareMatrix, getSquareMatrixDimension, getSquareMatrixEntry` and `setSquareMatrixEntry` are implemented in order to work with the structure `squareMatrix`. (NOTE: You DO NOT have to implement neither the structure `squareMatrix`, nor the corresponding functions!)
What is the function `func` doing, when it is called with the matrix

$$A = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 \\ 1 & 2 & 0 & 2 \\ 17 & 4 & 4 & 1 \end{pmatrix}?$$

Create a table, where you put in the values of all variables at the given time (the comment line in the following code). What is the function `func` doing in general? What is inefficient about this code? Explain how this code can be improved!

```
int func(squareMatrix* mat) {
   double foo = 0;
   int mp, dp, tf;
   mp = 1;
   for (dp = 0; dp < getMatrixDim(mat); ++dp) {
      for (tf = dp+1; tf < getMatrixDim(mat); ++tf) {
         foo = getMatrixEntry(mat,dp,tf);
         if ( foo != 0 ) {
            mp = 0;
         }
      /* VALUE OF VARIABLES AT THIS POINT */
      }
   }
   return mp;
}
```