Dirk Praetorius
Michele Ruggeri

Sommersemester 2015
28.05.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. An alternative root-finding algorithm (see also the Bisection method from the lecture) is the so called *secant method.* Let $f : [a, b] \to \mathbb{R}$. Given two initial guesses x_0 and x_1, the appromation x_{n+1} is obtained as the root of the line through $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$, i.e.,

$$x_{n+1} := x_n - f(x_n)\,\frac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)}.$$

Write a function `secant(x0,x1,tau)`, which performs the above iteration until either

$$|f(x_n) - f(x_{n-1})| \le \tau$$

or

$$|f(x_n)| \le \tau \quad \text{and} \quad |x_n - x_{n-1}| \le \begin{cases} \tau & \text{for } |x_n| \le \tau, \\ \tau|x_n| & \text{else.} \end{cases}$$

In the first case, print a warning to inform that the result is presumably wrong. The function returns x_n as the approximation of the root z_0 of f. Test your implementation with a suitable example.Then, write a main program, that reads x_0 and x_1 from the keyboard and displays x_n. Save your source code as `secant.c` into the directory `serie06`.

Aufgabe 6.2. 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

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

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. Write a main-programme to test the function `computeLU` on a suitable example. Save your source code as `computeLU.c` into the directory `serie06`.

Aufgabe 6.3. 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 6.2 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. Test your code on a suitable example. Speichern Sie den Source-Code unter `computeLU3.c` in das Verzeichnis `serie06`.

Aufgabe 6.4. Write a library for *columnwise*(!) stored $m \times n$-matrices. Implement the following functions

- `double* mallocmatrix(int m, int n)`
  Allocates memory for a columnwise stored $m \times n$ matrix.
- `double* freematrix(double* matrix)`
  Frees memory of a matrix.
- `double* reallocmatrix(double* matrix, int m, int n, int mNew, int nNew)`
  Reallocates memory and initializes new entries.

Store the signatures of the functions in the header file `dynamicmatrix.h`. Write also appropriate comments to this functions in the header file. The file `dynamicmatrix.c` should contain the implementations of the above functions. Use dynamical arrays.

Aufgabe 6.5. Expand the library from Exercise 6.4 by the following functions.

- `void printmatrix(double* matrix, int m, int n)`
  Prints the column-wise-saved $m \times n$-Matrix on screen. The 2×3-Matrix `double matrix[6]={1,2,3,4,5,6}` shall look like in the following example:

  ```
  1 3 5
  2 4 6
  ```

- `double* scanmatrix(int m, int n)`
  Allocates memory for a matrix and scans the coefficients from keyboard-entry.
- `double* cutOffRowJ(double* matrix, int m, int n, int j)`
  Cuts off the j-th line from a $m \times n$-Matrix.
- `double* cutOffColK(double* matrix, int m, int n, int k)`
  Cuts off the k-th column from a $m \times n$-Matrix.

Use dynamical arrays. Write a main program, that tests the functions from this exercise and from Exercise 6.4.

Aufgabe 6.6. Write a function `dec2float`, that, for a given decimal number $x \in \mathbb{R}_{>0}$ and a mantissa, $M \in \mathbb{N}$, computes and returns the digits $a_1, \ldots, a_M \in \{0, 1\}$ and the exponent $e \in \mathbb{Z}$ of the normalized floating-point-representation (meaning $a_1 = 1$). Moreover, write a main-programm, which reads x from the keyboard, calls the function `dec2float` and prints the floating-point-representation on screen. Save your source code as `dec2float.c` into the directory `serie06`.

Aufgabe 6.7. Write a program that reads in a word (string) und checks if this word is a *palindrome*. A palindrome is a word which reads the same backward or forward, e.g., radar, level, madam. Save your source code as `palindrome.c` into the directory `serie06`.

Aufgabe 6.8. Where are the errors in the program? Explain why!

```
#include <stdio.h>

void square(double* x)
{
  double* y;
  x=(*y)*(*x);
```

```
}

int main(){
  double x=2.1;
  square(&x);
  printf("x^2=%f\n",x);
  return 0;
}
```

Change *only* the function `square`, such that the output of the code is correct.