Dirk Praetorius
Michele Ruggeri

Sommersemester 2015
28.05.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Alternativ zum Bisektionsverfahren aus der Vorlesung kann eine Nullstelle von $f : [a, b] \to \mathbb{R}$ auch mit dem *Sekantenverfahren* berechnet werden. Dabei sind x_0 und x_1 gegebene Startwerte und man definiert induktiv x_{n+1} als Nullstelle der Geraden durch $(x_{n-1}, f(x_{n-1}))$ und $(x_n, f(x_n))$, d.h.

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

Schreiben Sie eine Funktion `sekante(x0,x1,tau)` die die Folge der Iterierten berechnet, bis entweder

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

oder

$$|f(x_n)| \le \tau \quad \text{und} \quad |x_n - x_{n-1}| \le \begin{cases} \tau & \text{für } |x_n| \le \tau, \\ \tau|x_n| & \text{sonst} \end{cases}$$

gilt. Es werde dann x_n als Approximation einer Nullstelle z_0 von f zurückgegeben. Im ersten Fall gebe man zusätzlich eine Warnung aus, dass das numerische Ergebnis vermutlich falsch ist.
Verwenden Sie ein geeignetes Beispiel zum Test. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x_0 und x_1 eingelesen werden und x_n ausgegeben wird. Speichern Sie den Source-Code unter `sekante.c` in das Verzeichnis `serie06`.

Aufgabe 6.2. Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

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

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

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

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

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

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null. Schreiben Sie eine Funktion `computeLU`, die die LU-Zerlegung von A berechnet und zurückgibt. Dazu überlege man, in welcher Reihenfolge man die Einträge von L und U berechnen muss, damit die angegebenen Formeln wohldefiniert sind (d.h. alles was benötigt wird, ist bereits zuvor berechnet worden). Schreiben Sie ein main-Programm, in dem Sie die Funktion `computeLU` an einen geeigneten Beispiel testen. Speichern Sie den Source-Code unter `computeLU.c` in das Verzeichnis `serie06`.

Aufgabe 6.3. Die Matrix $A \in \mathbb{R}^{n\times n}$ sei eine Tridiagonalmatrix, d.h.

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

wobei alle anderen Einträge von A Null sind, und besitze eine LU-Zerlegung. Mit Hilfe der Formeln in Aufgabe 6.2 überlege man sich Formeln für die Einträge von L und U in diesem speziellen Fall. Dann schreibe man eine Funktion `computeLU3` bei der keine unnötigen Rechenoperationen (d.h. Additionen/Multiplikationen von Null) durchgeführt werden und nur Einträge von L und U berechnet werden, die nicht Null sind. Testen Sie Ihren Code an einem geeigneten Beispiel.
Speichern Sie den Source-Code unter `computeLU3.c` in das Verzeichnis `serie06`.

Aufgabe 6.4. Schreiben Sie eine Bibliothek zur Verwaltung von *spaltenweise* gespeicherten $m \times n$-Matrizen. Implementieren Sie die folgenden Funktionen

- `double* mallocmatrix(int m, int n)`
  Allokieren von Speicher für eine spaltenweise gespeicherte $m \times n$- Matrix.
- `double* freematrix(double* matrix)`
  Freigeben des allokierten Speichers einer Matrix.
- `double* reallocmatrix(double* matrix, int m, int n, int mNew, int nNew)`
  Reallokieren und initialisieren von neuen Einträgen.

Speichern Sie die Funktionssignaturen in das Header-File `dynamicmatrix.h`. Schreiben Sie auch entsprechende Kommentare zu den Funktionen in das Header-File. In die Datei `dynamicmatrix.c` kommt dann die Implementierung der Funktionen. Verwenden Sie dynamische Arrays.

Aufgabe 6.5. Erweitern Sie die Bibliothek aus Aufgabe 6.4 um folgende Funktionalitäten

- `void printmatrix(double* matrix, int m, int n)`
  Gibt eine spaltenweise gespeicherte $m \times n$-Matrix als Matrix am Bildschirm aus. Die 2×3-Matrix `double matrix[6]={1,2,3,4,5,6}` soll wie folgt ausgegeben werden:

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

- `double* scanmatrix(int m, int n)`
  Allokiert Speicher für eine Matrix und liest die Koeffizienten der Matrix von der Tastatur ein.
- `double* cutOffRowJ(double* matrix, int m, int n, int j)`
  Schneidet die j-te Zeile aus einer $m \times n$-Matrix heraus.
- `double* cutOffColK(double* matrix, int m, int n, int k)`
  Schneidet die k-te Spalte aus einer $m \times n$-Matrix heraus.

Verwenden Sie dynamische Arrays. Schreiben Sie ein main-Programm um die Funktionen aus dieser Aufgabe und aus Aufgabe 6.4 zu testen.

Aufgabe 6.6. Schreiben Sie eine Funktion `dec2float`, die für eine gegebene Dezimalzahl $x \in \mathbb{R}_{>0}$ und eine Mantissenlänge $M \in \mathbb{N}$ die Ziffern $a_1, \ldots, a_M \in \{0, 1\}$ und den Exponenten $e \in \mathbb{Z}$ der normalisierten Gleitkommadarstellung (d.h. $a_1 = 1$) berechnet und zurückgibt. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x eingelesen und die Gleitkommadarstellung von x ausgegeben wird. Speichern Sie den Source-Code unter `dec2float.c` in das Verzeichnis `serie06`.

Aufgabe 6.7. Schreiben Sie ein Programm, welches ein Wort einliest und überprüft ob es sich bei dem eingegeben Wort um ein Palindrom handelt. Ein Palindrom ist ein Wort, welches von vorne und hinten gelesen gleich lautet, z.B.: Anna, Otto, Reliefpfeiler. Speichern Sie den Source-Code unter `palindrom.c` in das Verzeichnis `serie06`.

Aufgabe 6.8. Wo liegen die Fehler im folgenden Programm?

```
#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;
}
```

Verändern Sie *nur* die Funktion `square`, so dass der Output des Codes den Erwartungen entspricht.