Dirk Praetorius
Michele Ruggeri

Sommersemester 2015
12.05.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 8

Aufgabe 8.1. Für einen stetigen Integranden $f : [a, b] \to \mathbb{R}$ berechnet man das Integral $I := \int_a^b f\,dx$ numerisch über geeignete Summen. Bei der *summierten Trapezregel* berechnet man für gegebenes $n \in \mathbb{N}$ und $h := (b-a)/n$ zum Beispiel

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

Dies ist gerade das Integral über die stetige und stückweise affine Funktion p mit $p(a+jh) = f(a+jh)$. Schreiben Sie eine Funktion `trapezregel(f,a,b,tau)`, die die Folge der Approximationen I_n berechnet, bis gilt

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

In diesem Fall gebe man die vollständige Folge $(I_1, \dots, I_n)$ der Approximationen zurück. Man teste die numerische Integration am Beispiel $f(x) = \exp(x)$ auf dem Intervall $[0, 10]$ und gebe abhängig von n neben dem Fehler $|I - I_n|$ auch die experimentelle Konvergenzordnung tabellarisch aus. Speichern Sie den Source-Code unter `trapezregel.c` in das Verzeichnis `serie08`.

Aufgabe 8.2. Schreiben Sie eine Struktur `CVector`, zur Speicherung von Vektoren mit komplexwertigen Koeffizienten. Benutzen Sie hierzu den Strukturdatentyp `cdouble` aus Aufgabe 7.1 und Aufgabe 7.2. Ferner schreibe man die zugehörigen Funktionen `newCVector`, `delCVector`, `getCVectorLength`, `getCVectorEntry`, `setCVectorEntry`. Speichern Sie den Source-Code unter `cvector.c` in das Verzeichnis `serie08`.

Aufgabe 8.3. Schreiben Sie eine Funktion `CVectorVector`, die das Skalarprodukt $x \cdot y := \sum_{j=1}^n x_j \overline{y_j}$ zweier komplexwertiger Vektoren $x, y \in \mathbb{C}^n$ berechnet. Benutzen Sie hierzu den Strukturdatentyp `CVector` aus Aufgabe 7.1 und Aufgabe 7.2. Schreiben Sie ferner ein aufrufendes Hauptprogramm, indem die Vektoren x, y eingelesen und der Wert $x \cdot y \in \mathbb{C}$ ausgegeben werden. Speichern Sie den Source-Code unter `CVectorVector.c` in das Verzeichnis `serie08`. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 8.4. Schreiben Sie eine Struktur `CMatrix`, zur Speicherung von $(m \times n)$-Matrizen $A \in \mathbb{C}^{m\times n}$ mit komplexwertigen Koeffizienten. Benutzen Sie hierzu den Strukturdatentyp `cdouble` aus Aufgabe 7.1 und Aufgabe 7.2. Ferner schreibe man die zugehörigen Funktionen `newCMatrix`, `delCMatrix`, `getCMatrixM`, `getCMatrixN`, `getCMatrixCoeff`, `setCMatrixCoeff`.

Aufgabe 8.5. Schreiben Sie eine Funktion `cmatrixvector`, die die Matrix-Vektor-Multiplication $Ax \in \mathbb{C}^m$ mit einer Matrix $A \in \mathbb{C}^{m\times n}$ und einem Vektor $x \in \mathbb{C}^n$ realisiert. Verwenden Sie für die Koeffizienten die komplexe Arithmetik aus Aufgabe 7.1 und Aufgabe 7.2. Speichern Sie den Source-Code unter `cmatrixvector.c` in das Verzeichnis `serie08`. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 8.6. Schreiben Sie eine Struktur `Matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `0`) auch untere (Typ `'L'`) und obere (Typ `'U'`)

Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

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

als obere bzw. untere Dreiecksmatrix. Mathematisch formuliert, gilt also $u_{jk} = 0$ für $j > k$ bzw. $\ell_{jk} = 0$ für $j < k$. Eine vollbesetzte Matrix werde im Fortran-Format spaltenweise als dynamischer Vektor der Länge $n \cdot n$ gespeichert. Dreiecksmatrizen sollen in einem Vektor der Länge $\sum_{j=1}^{n} j = n(n+1)/2$ gespeichert werden. Schreiben Sie die Funktionen, um mit dieser Struktur arbeiten zu können (`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Speichern Sie den Source-Code unter `matrix.c` in das Verzeichnis `serie08`. Dabei hängen insbesondere die Funktionen `getMatrixEntry` und `setMatrixEntry` vom Matrixtyp (Dreiecksmatrix!) ab.

Aufgabe 8.7. Schreiben Sie eine Funktion `spaltensummennorm`, die die Spaltensummennorm

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

einer Matrix $A \in \mathbb{R}^{n \times n}$ zurückgibt. Die Matrix A sei dabei in der Datenstruktur aus Aufgabe 8.6 gespeichert, und etwaige Struktur (Dreiecksmatrix!) von A soll ausgenutzt werden. Speichern Sie den Source-Code unter `spaltensummennorm` in das Verzeichnis `serie08`. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 8.8. Gegeben ist der Strukturdatentyp `squareMatrix` zum Speichern quadratische Matrizen $A \in \mathbb{R}^{n \times n}$. Hierbei werden die Einträge der Matrix spaltenweise als `double*`, sowie die Größe $n \in \mathbb{N}$ abgespeichert. Der Eintrag A_{ij} wird also an der Stelle `[i+j*n]` gespeichert. Darüberhinaus verfügen Sie über die üblichen Funktionen `newSquareMatrix, delSquareMatrix, getSquareMatrixDimension, getSquareMatrixEntry`. und `setSquareMatrixEntry` um mit der Struktur `squareMatrix` arbeiten zu können. (ACHTUNG: Sie müssen diese Struktur und die zugehörigen Funktionen NICHT programmieren!)
Was tut nun die folgende Funktion `func` bei Übergabe der Matrix

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

Geben Sie tabellarisch wieder, welchen Wert die Variablen zum angegebenen Zeitpunkt haben. Welche Funktionalität wird durch `func` bereitgestellt? Was ist an dieser Lösung ineffizient realisiert? Erklären Sie, wie man das effizienter gestalten könnte?

```
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;
         }
      /* WERT DER VARIABLEN ZU DIESEM ZEITPUNKT */
      }
```

```
    }
    return mp;
}
```