Projekt 2: "schnelle" Matrix-Matrixmultiplikation

Der Standardalgorithmus zur Multiplikation zweier Matrizen $A, B \in \mathbb{R}^{n\times n}$ hat die Komplexität $\mathcal{O}(n^3)$. Der vom Mathematiker Volker Strassen 1969 publizierte *Strassen-Algorithmus* zur Matrixmultiplikation reduziert die Komplexität (d.h. die Anzahl arithmetischer Operationen) auf $\mathcal{O}(n^{\log_2(7)})$. Der Algorithmus wird hier kurz beschrieben.
Sei der Einfachheit halber $n = 2^m$. Berechnet werden soll

$$C = AB. \tag{1}$$

Die Matrizen werden in vier Blöcke der Größe 2^{m-1} zerlegt

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \text{ und } C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \tag{2}$$

Die Standardmultiplikation läßt sich schreiben als

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{21} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}. \tag{3}$$

Eine andere – äquivalente – Darstellung gewinnt man durch Definition folgender Matrizen

$$\begin{aligned} M_1 &:= (A_{11} + A_{22})(B_{11} + B_{22}) && \text{(4a)}\\ M_2 &:= (A_{21} + A_{22})B_{11} && \text{(4b)}\\ M_3 &:= A_{11}(B_{12} - B_{22}) && \text{(4c)}\\ M_4 &:= A_{22}(B_{21} - B_{11}) && \text{(4d)}\\ M_5 &:= (A_{11} + A_{12})B_{22} && \text{(4e)}\\ M_6 &:= (A_{21} - A_{11})(B_{11} + B_{12}) && \text{(4f)}\\ M_7 &:= (A_{12} - A_{22})(B_{21} + B_{22}) && \text{(4g)} \end{aligned}$$

und der Beobachtung, daß die Gleichungen

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7,\\ C_{12} &= M_3 + M_5,\\ C_{21} &= M_2 + M_4, \quad \text{und}\\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

gelten. Man geht nun rekursiv für die Multiplikationen in (4) vor bis nur noch 1×1-Matrizen zu multiplizieren sind.

1. Überzeugen Sie sich von der Richtigkeit des oben beschriebenen Algorithmus, und beweisen Sie die Komplexität.

2. Programmieren Sie folgende Funktionen für quadratische Matrizen der Größe 2^m des Typs **float** in C oder C++:

a) das Standardverfahren zur Multiplikation zweier Matrizen

b) den Strassen-Algorithmus zur Multiplikation zweier Matrizen

Die Funktion für den Strassen-Algorithmus ist rekursiv und sollte daher keinen Speicher anfordern. Geben Sie ihr eine vierte Matrix als „Arbeitsmatrix" mit, z.B

```
float * A    = new float[n*n]; FillWithSomething(A);
float * B    = new float[n*n]; FillWithSomething(B);
float * C    = new float[n*n];
float * Work = new float[n*n];  // ab hier nie wieder 'new'
ComputeStrassen(n,n,A,B,C,Work);// A*B=C unter Verwendung von Work
```

Dokumentieren Sie ihren Quellcode ausreichend. Testen Sie Ihre Funktionen mit einfachen Beipielmatrizen für kleines m, bis Sie sicher sind, daß die Funktionen das Gewünschte tun. Testen Sie dazu zunächst die Funktion zur Standardmultiplikation, gegebenenfalls unter Verwendung von MATLAB. Testen Sie dann die Strassen-Funktion, indem Sie die Ergebnisse mit der Standardmultiplikation vergleichen. Dokumentieren Sie Ihre Testbeispiele und Ergebnisse angemessen.

3. Die Komplexität und das Laufzeitverhalten der beiden Verfahren sollen verglichen werden. Schreiben Sie dazu ein Programm, dass die in Aufgabe 2 erstellten Funktionen verwendet und die von ihnen verwendete CPU-Zeit mißt. Verwenden Sie dazu die Funktion clock(), z.B.

```
#include <ctime>
...
  clock_t start = clock();
  ComputeStrassen(...);
  clock_t end = clock();

  took = difftime(end,start)*1000.0/CLOCKS_PER_SEC;
  std::cout << "Took " << took << " [millisec]" << std::endl;
```

Erstellen Sie in dem Programm zu gegebenem m, das von der Komandozeile eingelesen wird, zwei Matrizen A und B der Größe $n = 2^m$, die zufällige Einträge in $[0, 1]$ enthalten. Legen Sie alle weiteren Matrizen an. Führen Sie dann die Multiplikation auf beiden Arten aus, messen die benötigten Zeiten und geben Sie diese aus. In etwa:

```
  clock_t sStrassen = clock();
  ComputeStrassen(...);
  clock_t eStrassen = clock();

  clock_t sStandard = clock();
  ComputeStandard(...);
  clock_t eStandard = clock();

  std::cout << "Zeiten fuer n = " << n << std::endl
            << "Strassen: "
            << difftime(eStrassen,sStrassen)/CLOCKS_PER_SEC
            << " [sec]" << std::endl
            << "Standard: "
            << difftime(eStandard,sStandard)/CLOCKS_PER_SEC
            << " [sec]" << std::endl;
```

Rufen Sie Ihr Programm für $m = 6, \ldots, 11$ (oder größer, wenn es der Speicher zuläßt) auf. Übertragen Sie die gewonnenen Zeiten in eine Tabelle. Plotten Sie die Zeiten in MATLAB. Was beobachten Sie?

Hinweis: Verwenden Sie nicht die Optimierungsmöglichkeiten Ihres Compilers, da sie das Laufzeitverhalten für Spezialfälle enorm verbessern können und Ihre Ergebnisse verfälschen.

4. Die Stabilität der Verfahren soll ebenfalls untersucht werden. Dazu werden Sie genauere Ergebnisse unter Verwendung des **double** Typs und seiner Arithmetik erzeugen und mit den **float** Ergebnissen vergleichen. Schreiben Sie dazu weitere Funktionen

a) die Standardmultiplikation für **double**-Matrizen,

b) die Konvertierung von **float**- in **double**-Matrizen

c) die Frobeniusnorm der Differenz zweier **double**-Matrizen, also $\|A-B\|_F := \sqrt{\sum_{i,j}(A_{ij} - B_{ij})^2}$.

Wenn Sie C++ verwenden und mit Templates vertraut sind, bietet es sich an, die Standardmultiplikation für Matrizen allgemeinen Typs zu implementieren.

Schreiben Sie ein Programm, das zu gegebem m

a) eine **float**-Matrix A der Größe 2^m mit Zufallswerten in $[0, 1]$ erzeugt,

b) eine **float**-Matrix B der Größe 2^m mit alternierenden Werten 1, -1 anlegt,

c) davon das Strassenprodukt C^f_{Strassen} berechnet,

d) davon das Standardprodukt C^f_{Standard} berechnet,

e) A und B in **double**-Matrizen A_d und B_d konvertiert,

f) das **double**-Standardprodukt C^d_{Standard} von A_d und B_d berechnet

g) C^f_{Strassen} und C^f_{Standard} in **double**-Matrizen $C^f_{d,\text{Strassen}}$ und $C^f_{d,\text{Standard}}$ konvertiert, und

h) die Fehler $\|C^d_{\text{Standard}} - C^f_{d,\text{Standard}}\|_F$ und $\|C^d_{\text{Standard}} - C^f_{d,\text{Strassen}}\|_F$ berechnet.

Rufen Sie Ihr Programm für $m = 4, \ldots, 11$ (oder größer, wenn es der Speicher zuläßt) auf und notieren Sie die Fehler. Was beobachten Sie? Woran könnte das liegen?

5. Überlegen Sie sich einen Algorithmus, der in ähnlicher Weise (d.h. rekursiv) die normalisierte LU-Zerlegung einer Matrix berechnet. Geben Sie die Komplexität an!