Goethe-Center for Scientific Computing (G-CSC)
Goethe-Universität Frankfurt am Main

Modellierung und Simulation II

(Praktikum ModSim2, WiSe 20/21)

Dr. A. Nägel, L. Larisch, M. Stepniewski

Blatt 1 (Abgabe: Mi., 2.12.2020, 10h)

Auf dem ersten Aufgabenblatt werden Sie einige grundlegende Klassen implementieren, die im weiteren Verlauf des Praktikums benötigt werden. Eine erste Anwendung wird in der Diskretisierung einer Poisson-Gleichung bestehen.
Für alle zu implementierenden Klassen steht ein Code-Gerüst zur Verfügung, in das Sie Ihre Implementierung an den markierten Stellen einfügen können. Sie dürfen die Klassen auch anders als vorgeschlagen implementieren, sofern das vorgegebene Interface dabei erfüllt wird.

Aufgabe 1 (Vektor, 4 Punkte)
Implementieren Sie eine Klasse `Vector`, die mindestens das in der Datei »vector.h« vorgegebene und beschriebene Interface erfüllt.
Tipp: Sie können einige der Anforderungen erfüllen, indem Sie von der STL-Klasse `std::vector<double>` ableiten.

Aufgabe 2 (Dicht besetzte Matrix, 6 Punkte)
Schreiben Sie eine Klasse `Matrix`, die eine dicht besetzte Matrix realisiert. Dabei soll jeder Eintrag der Matrix gespeichert werden. Implementieren Sie mindestens das in der Datei »matrix.h« vorgegebene Interface, insb. auch die Iteratoren über die Matrix.

Aufgabe 3 (Strukturiertes Gitter, 8 Punkte)
Implementieren Sie eine Klasse `StructuredGrid`, die mindestens das in der Datei »structured_grid.h« vorgegebene und beschriebene Interface erfüllt.
Die Klasse soll ein strukturiertes, in jede Dimension (1, 2 oder 3) separat äquidistantes Vertexgitter beschreiben, das durch die unteren und oberen Ränder sowie durch die Anzahl der Vertices in jeder Dimension definiert ist. Die Vertices sollten am besten erst in x-Richtung, dann in y- und z-Richtung nummeriert sein (lexikographische Anordnung).

Aufgabe 4 (Poisson-Diskretisierung, 10 Punkte)
Gegeben ist das Poisson-Problem auf dem d-dimensionalen Gebiet $\Omega = (0,1)^d$:

$$\begin{aligned} -\Delta u &= f \quad \text{auf } \Omega, \\ u &= g \quad \text{auf } \Gamma_D \subset \partial\Omega. \end{aligned}$$

mit ggf. von der Raumkoordinaten $\mathbf{x}$ abhängigen Funktionen f und g. Schreiben Sie eine Klasse `PoissonDisc`, die eine Finite-Differenzen-Diskretisierung für das Poisson-Problem (mit $d \in \{1,2,3\}$) durchführt, d.h. für ein gegebenes strukturiertes Gitter die Matrix und den Rechte-Seite-Vektor des entstehenden Gleichungssystems erzeugt.

Aufgabe 5 (LU-Solver, (12 Punkte)
Die *LU-Zerlegung* kann zum Lösen von Linearen Gleichungssystemen der Form

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow \mathbf{x} = A^{-1}\mathbf{b} \tag{1}$$

benutzt werden, wobei $A \in \mathbb{R}^{N\times N}$ eine invertierbare Matrix sei, $\mathbf{b} \in \mathbb{R}^N$ ein gegebener Vektor und $\mathbf{x} \in \mathbb{R}^N$ der Vektor mit der unbekannten Lösung.

Mit Hilfe der *LU-Zerlegung* wird die Matrix A in eine rechte obere (*upper*) Matrix U und eine linke untere (*lower*) Matrix L aufgespalten, sodass

$$A = L \cdot U \tag{2}$$

mit

$$L_{ij} = 0, \text{ für } i < j, \qquad L_{ii} = 1, \tag{3}$$
$$U_{ij} = 0, \text{ für } j < i, \tag{4}$$

D.h. alle Nulleinträge von U befinden sich auf dem rechten oberen Teil der Matrix und alle Nulleinträge von L auf dem unteren linken. Da die Diagonale von L eindeutig durch Einseinträge definiert ist (und daher nicht explizit gespeichert werden muss), lässt sich der Speicher von A für beide Matrizen L und U gleichzeitig wie folgt benutzen:

$$A := \begin{pmatrix} U_{11} & U_{12} & U_{12} & \dots & U_{1N} \\ L_{21} & U_{22} & U_{23} & \dots & U_{2N} \\ L_{31} & L_{32} & U_{33} & \dots & U_{3N} \\ \vdots & \dots & \ddots & \ddots & \vdots \\ L_{N1} & L_{N2} & \dots & L_{N,N-1} & U_{NN} \end{pmatrix} \tag{5}$$

Nachfolgend finden Sie den Algorithmus zur Berechnung der *LU-Zerlegung* in Pseudo-Code:

Algorithm 1 $A = LU$ factorization

Input: $A \in \mathbb{R}^{N \times N}$ regular
for $k = 1, \ldots, N-1$ **do**
  for $i = k+1, \ldots, N$ **do**
    $A_{ik} := A_{ik}/A_{kk}$
    for $j = k+1, \ldots, N$ **do**
      $A_{ij} := A_{ij} - A_{ik} \cdot A_{kj}$
    end for
  end for
end for
Output: $A := LU$ (using the same storage as the input matrix!)

Nach der *LU-Zerlegung* der Matrix A wird das Lösen des Linearen Gleichungssystems $Ax = b \Leftrightarrow LUx = b$ nun in zwei Stufen realisiert:

1. **Vorwärtseinsetzen**:
   $L\mathbf{y} = \mathbf{b}$ mit
   $y_i := b_i - \sum_{j=1}^{i-1} L_{ij} y_j, \;\; i = 1, .., N$

2. **Rückwärtseinsetzen**:
   $U\mathbf{x} = \mathbf{y}$ mit
   $x_i := \frac{1}{U_{ii}} \left(y_i - \sum_{j=i+1}^{N} U_{ij} x_j \right), \;\; i = N, .., 1$

Implementieren Sie die Klasse `LUSolver`, welche die LU-Zerlegung einer Matrix A durchführt und mit deren Hilfe lineare Gleichungssysteme mit A löst. Speichern Sie die Dreiecksmatrizen L und U in einer einzigen gemeinsamen Matrix (ohne die Diagonaleinträge von L, welche allesamt 1 sind), die eine Member-Variable der Klasse ist.

Testen Sie Ihre `LUSolver` Implementierung für die sog. Hilbert-Matrix $A = (a_{ij})_{i,j=1}^{n}$, $a_{ij} = \frac{1}{i+j-1}$ sowie die rechte Seite $\mathbf{b} = (1, \ldots, n)$ für die 4 Fälle $n \in \{2, 5, 10, 30\}$.

Hinweis (zu allen Aufgaben): Sie können Ihre Implementierungen testen, indem Sie den Code in »main01_20.cpp« (oder Teile davon) kompilieren und ausführen. Sollten Ihre Resultate von den vorgegebenen abweichen, enthält Ihr Code mit großer Wahrscheinlichkeit noch Fehler.
Zum Kompilieren nutzen Sie bitte – soweit möglich – cmake. Angenommen, Ihre Dateien liegen im Ordner »code«. Dann wechseln Sie in der Konsole/Terminal (Linux/Mac/Windows?) in diesen Ordner und geben Sie folgenden Befehl ein:
`mkdir build && cd build && cmake .. && make`
Wenn Sie alles vollständig und syntaktisch korrekt implementiert haben, sollte so in »code/bin« ein Executable mit dem Namen »execMain01« entstehen, das Sie zum Testen ausführen können.

Hinweis (zu Aufgaben 3 und 4): Für die Realisierung der Klassen in den Aufgaben 3 und 4 steht ihnen neben dem jeweiligen Code-Gerüst auch noch die bereits implementierte Klasse `CoordVector` zur effizienten Speicherung von Koordinaten fester Dimension in der Datei »coord_vector.h« zur Verfügung.

Abgabe: Senden Sie Ihren Code sowie sonstige Antworten als Text, PDF oder Scans bitte per E-Mail an die beiden folgenden Adressen:
`lukas.larisch@gcsc.uni-frankfurt.de`
`martin.stepniewski@gcsc.uni-frankfurt.de`
An diese Adressen können Sie sich auch bei Fragen zu den Aufgaben wenden.