Übungen zur Vorlesung „Physik auf dem Computer"
Sommersemester 2018

Übungsgruppenleiter:
Robin Bardakcioglu – rhb@itp1.uni-stuttgart.de; Do. 14:00 - 15:30 Uhr, 5.331
Johannes Reiff – jreiff@itp1.uni-stuttgart.de; Di. 14:00 - 15:30 Uhr, 4.141
Matthias Feldmaier – fem@itp1.uni-stuttgart.de; Do. 14:00 - 15:30 Uhr, 4.141

Übungsblatt 10 Ausgabe: 27. Juni 2018

Dienstagsübung: schriftliche Abgabe 01.07.18, Besprechung 03.07.18
Donnerstagsübungen: schriftliche Abgabe 03.07.18, Besprechung 05.07.18

Aufgabe 24: LU-Zerlegung und lineare Gleichungssysteme

In der Vorlesung wurde die LU-Zerlegung durch eine Gauß-Elimination vorgestellt. In einer Programmierer-freundlichen Darstellung kann der Algorithmus dafür mit den folgenden Gleichungen angegeben werden. Einen ersten Schritt, der für eine $n \times n$-Matrix die Transformation

$$\boldsymbol{A}^{(0)} = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \cdots & a_{2n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \cdots & a_{nn}^{(0)} \end{pmatrix} \quad \longrightarrow \quad \boldsymbol{A}^{(1)} = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

ausführt, erreicht man mit den Operationen

$$a_{ij}^{(1)} = a_{ij}^{(0)} - \ell_i^{(1)} a_{1j}^{(0)} \,, \qquad \text{mit} \quad \ell_i^{(1)} = \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} \,, \qquad \text{für} \quad i = 2, \ldots, n \,, \quad j = 1, \ldots, n$$

$$a_{1j}^{(1)} = a_{1j}^{(0)} \,, \qquad \text{für} \quad j = 1, \ldots, n \,.$$

Sukzessive erhält man dann die Matrizen $\boldsymbol{A}^{(k)}$ mit den Operationen

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \ell_i^{(k)} a_{kj}^{(k-1)} \,, \qquad \text{mit} \quad \ell_i^{(k)} \quad = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \,, \qquad \text{für} \quad i = k+1, \ldots, n \,, \quad j = 1, \ldots, n$$

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} \,, \qquad \text{sonst} \,.$$

Die Matrix $\boldsymbol{U} = \boldsymbol{A}^{(n-1)}$ ist die gesuchte obere rechte Dreiecksmatrix und die Matrix

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_2^{(1)} & 1 & 0 & \cdots & 0 \\ \ell_3^{(1)} & \ell_3^{(2)} & 1 & & 0 \\ \vdots & & \ddots & \ddots & \\ \ell_n^{(1)} & \cdots & \cdots & \ell_n^{(n-1)} & 1 \end{pmatrix}$$

bildet die gesuchte untere linke Dreiecksmatrix.

a) Laden Sie aus dem Webangebot zur Vorlesung das Programmpaket zu dieser Aufgabe und öffnen Sie die Datei `LU_Gauss.cc`. Sie enthält ein Programm, das zur LU-Zerlegung einer reellen $n \times n$-Matrix, die mit Zufallszahlen gefüllt wurde, vorbereitet ist.

ToDo: Der Algorithmus zur LU-Zerlegung fehlt noch. Implementieren Sie ihn so, dass die Matrix $\boldsymbol{U}$ am Ende in $\boldsymbol{A}$ abgespeichert ist (überschreiben Sie also $\boldsymbol{A}$ in jedem der oben aufgelisteten Schritte) und für $\boldsymbol{L}$ die vorbereitete Einheitsmatrix verwendet wird. Überzeugen Sie sich, dass der Algorithmus funktioniert, indem Sie die Ausgabe des Programms für mehrere verschiedene Matrixgrößen beobachten. Die beiliegende Datei `Makefile` ist so vorbereitet, dass sie zum Compilieren des Programms genutzt werden kann. Stellen Sie Ihren Algorithmus in Pseudocode vor.

(4 Punkt(e), Votier)

b) Die Matrix $\boldsymbol{L}$ ist im Programm prinzipiell überflüssig, da die trivialen Werte 0 in den unbesetzten Dreiecken und 1 auf der Diagonalen von $\boldsymbol{L}$ bekannt sind und nicht gespeichert werden müssen. Schreiben Sie das Programm so um, dass auf $\boldsymbol{L}$ komplett verzichtet wird und im Schritt $n-1$ die Matrix

$$\boldsymbol{A}^{(n-1)} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ \ell_2^{(1)} & u_{22} & u_{23} & \cdots & u_{2n} \\ \ell_3^{(1)} & \ell_3^{(2)} & u_{33} & & u_{3n} \\ \vdots & & \ddots & \ddots & \vdots \\ \ell_n^{(1)} & \cdots & \cdots & \ell_n^{(n-1)} & u_{nn} \end{pmatrix} \tag{1}$$

vorliegt, in der also die gesamte Information über $\boldsymbol{U}$ *und* $\boldsymbol{L}$ abgespeichert ist.

ToDo: Schreiben Sie den Algorithmus so, dass Sie auch während des Aufbaus der Matrizen $\boldsymbol{A}^{(k)}$ bis auf Summationsindizes keine weitere Variable benötigen. Verwenden Sie im gesamten restlichen Programm nur diese Matrix. Überprüfen Sie wieder anhand der Ausgabe der Matrix $\boldsymbol{LU}$, ob Ihr Algorithmus funktioniert. Stellen Sie auch diesen neuen Algorithmus in Pseudocode dar.

(4 Punkt(e), Votier)

c) Legen Sie eine Kopie des Programms an. Öffnen Sie die Datei `Makefile` und versuchen Sie das dort angeführte Beispiel impliziter Regeln zu verstehen. Es ermöglicht das Compilieren Ihrer neuen Datei, nachdem nur eine Zeile in `Makefile` geändert wurde. Führen Sie diese Änderung durch. Sie können aber auch ohne Verwendung von `make` weiterarbeiten.

d) Lösen Sie in einer Kopie des Programms (das darf die aus dem vorherigen Aufgabenteil sein) das lineare Gleichungssystem $\boldsymbol{Ax} = \boldsymbol{b}$ mit Hilfe der Matrix aus (1), indem Sie zuerst mit

$$y_i = b_i - \sum_{j=1}^{i-1} \ell_i^{(j)} y_j$$

das Teilsystem $\boldsymbol{Ly} = \boldsymbol{b}$ und dann mit

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right)$$

das Teilsystem $\boldsymbol{Ux} = \boldsymbol{y}$ lösen. Schreiben Sie den Algorithmus so, dass Sie außer der Matrix $\boldsymbol{A}^{(n-1)}$ und dem Vektor $\boldsymbol{b}$ keine weitere Variable benötigen.

Zur Kontrolle Ihres Ergebnisses können Sie die Eigen-Bibliothek verwenden:

```
/* Anlegen einer Kopie von A und b fuer die Berechnung mit dem
Eigen-Paket, _bevor_ sie veraendert werden. */
MatrixXd A_Kopie = A;
VectorXd b_Kopie = b;
...
// Zum Vergleich: Loesung mit der Eigen-Bibliothek
VectorXd x = A_Kopie.fullPivLu().solve(b_Kopie);
```

ToDo: Stellen Sie Ihren Quelltext zum Lösen des linearen Gleichungssystems vor.

(4 Punkt(e), Votier)

e) Bauen Sie Anweisungen zum Messen der Laufzeit Ihres selbstprogrammierten Algorithmus und des Aufrufs der Lösung des Gleichungssystems mit der Eigen-Bibliothek ein. Dies kann z.B. nach folgendem Muster geschehen.

```
#include <ctime>
...
main {
  ...
  // Zeitpunkt des Starts des selbst programmierten Algorithmus
  double t_start = clock();
  ...
  // Zeit fuer den selbst programmierten Teil messen
  cout << "Laufzeit selbstprogrammiert (in Sekunden): "
       << (clock() - t_start)/CLOCKS_PER_SEC << endl;
  ...
}
```

Betrachten Sie die Entwicklung der Laufzeit beider Algorithmen für Matrizen der Größen $n = 5, 10, 100, 500, 1000, 2000, 5000$. Berücksichtigen Sie dabei, dass der Aufruf der Eigen-Bibliothek eine LU-Zerlegung *mit* vollständiger Pivotierung in jedem Schritt enthält, der also wesentlich stabiler als Ihrer ist und deutlich mehr Operationen enthalten sollte.

ToDo: Welche Laufzeiten messen Sie für alle genannten Fälle?

(3 Punkt(e), Votier)

f) Da das Ergebnis der Eigen-Bibliothek aus einem vollständig pivotierten Algorithmus stammt, ist es zuverlässiger als die oben beschriebene Implementierung. Deren Qualität soll untersucht werden. Bauen Sie das Programm so um, dass Sie den Differenzvektor zwischen beiden Lösungen berechnen und geben Sie den relativen Betrag dieses Vektors (also normiert auf den Betrag des Lösungsvektors aus der Eigen-Bibliothek) für die Matrixgrößen $n = 5, 10, 100, 500, 1000$ aus. Wiederholen Sie diese Untersuchung für Matrizen, in denen sich die Beträge der Elemente stärker unterscheiden. Dies erreichen Sie z.B. durch:

```
#include <unsupported/Eigen/MatrixFunctions>
...
main {
  ...
  MatrixXd A = MatrixXd::Random(n,n);
  A = A.exp();
  ...
}
```

ToDo: Stellen Sie Ihre Änderungen des Programms vor und geben Sie die Ergebnisse Ihrer Untersuchung an. Was lernen Sie daraus über die Verwendung nicht-pivotierter Algorithmen?

(4 Punkt(e), Votier)

Aufgabe 25: Schlecht konditionierte Matrizen

a) Laden Sie das Programmpaket zu dieser Aufgabe und öffnen Sie die Datei `Invertiere.cc`. Sie enthält ein Programm zum Invertieren einer Matrix $\boldsymbol{M}$, die zuvor aufgebaut wird. Das Produkt $\boldsymbol{M}^{-1}\boldsymbol{M}$ wird anschließend ausgegeben. Compilieren Sie das Programm (`make` reicht) und lassen Sie es laufen. Die zu invertierende Matrix wird aus einer sehr einfachen Form aufgebaut und dann durch Ähnlichkeitstransformationen (Rotationen, orthogonale Matrix!) in eine voll besetzte Gestalt gebracht.

ToDo: Lesen Sie die einfache Form vor dem Rotieren aus dem Programm ab und invertieren Sie diese von Hand. Was fällt Ihnen auf? Kommentieren Sie in diesem Zusammenhang das numerische Ergebnis. Stellen Sie Vermutungen an, warum die Numerik im Gegensatz zu Ihnen zu einem Ergebnis kommt.

(3 Punkt(e), Votier)

b) Erweitern Sie das Programm so, dass Sie den „Abstand zur singulären Matrix“, also die Variable `diff` aus dem Matrixaufbau, in 1000 Schritten von 10^{-12} bis zu 10^{-15} durchlaufen lassen.

ToDo: Berechnen Sie für alle Fälle die Abweichung

$$a = \sum_{i,j=1}^{n} |(\boldsymbol{M}^{-1}\boldsymbol{M} - \mathbf{1})_{i,j}|^2$$

zur Einheitsmatrix und schreiben Sie die Werte in eine Datei. Erstellen Sie ein Diagramm, in dem Sie a logarithmisch über `diff` auftragen. Prinzipiell sind alle diese Matrizen invertierbar. Kommentieren Sie die Qualität der numerischen Invertierung.

(4 Punkt(e), Votier)

c) **Bonusaufgabe:** Lagern Sie die Funktion zum Matrixaufbau in eine eigene Datei aus, die Sie als `.o`-Datei compilieren. Erstellen Sie dafür eine implizite Regel in der Datei `Makefile` und ändern Sie die implizite Regel für `.exe`-Dateien so ab, dass alle von Ihrer neuen Datei zum Matrixaufbau abhängen. Beobachten Sie, wie das Compilieren funktioniert, insbesondere das Löschen der `.o`-Datei nach dem Erstellen des Programms, weil das „Zwischenergebnis“ dann nicht mehr benötigt wird. Stellen Sie fest, wie dieses Löschen nicht mehr stattfindet, wenn Sie ähnlich wie bei der Verwendung von `.PHONY` über

```
1 .PRECIOUS: %.o
```

ein besonderes Verhalten von `make` erzwingen.