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 5 Ausgabe: 16. Mai 2018

Dienstagsübung: schriftliche Abgabe 27.05.18, Besprechung 29.05.18
Donnerstagsübungen: schriftliche Abgabe 29.05.18, Besprechung 07.06.18 zusammen mit Blatt 6

Aufgabe 13: Monte-Carlo-Integration

Die Zahl π kann über das zweidimensionale Integral

$$\pi = 4 \int_0^1 \mathrm{d}x \int_0^1 \mathrm{d}y \, \Theta(1 - x^2 - y^2)$$

mit der Stufenfunktion

$$\Theta(x) = \begin{cases} 1 & \text{für } x \geq 0\,, \\ 0 & \text{sonst}\,, \end{cases}$$

das den Flächeninhalt des Einheitskreises berechnet, bestimmt werden. Dies soll hier mit einer Monte-Carlo-Integration durchgeführt werden.

In C und C++ lassen sich Zufallszahlen mit Hilfe der Funktion `rand()` bestimmen. Diese bestehen aus ganzen Zahlen im Intervall 0 bis `RAND_MAX`. Mit

```
1 x = double(rand())/RAND_MAX;
2 y = double(rand())/RAND_MAX;
```

erhalten Sie Zufallskoordinaten, die im Einheitsquadrat gleichverteilt sind. Zur Verwendung benötigen Sie im Programmkopf die Eingabe:

```
1 #include <cstdlib>
```

a) Schreiben Sie ein Programm, das die Monte-Carlo-Integration

$$\pi \approx \frac{4}{n} \sum_{i=1}^{n} \Theta(1 - x_i^2 - y_i^2)$$

ausführt. Die Zahl n der Zufallskoordinaten soll ein Eingabeparameter des Programms sein.

ToDo: Vergleichen Sie den Integralwert mit dem Wert `M_PI` für verschiedene Werte von n. Zeigen Sie, dass der Fehler für größere Werte von n kleiner wird. Geben Sie den relativen Fehler an. Erklären Sie in eigenen Worten den Grundgedanken und die Funktionsweise der Monte-Carlo-Integration (hierbei ist keine strenge mathematische Herleitung verlangt).

(5 Punkt(e), Votier)

Aufgabe 14: Verwendung von Bibliotheken und `make`

In der Praxis begegnen uns häufig dieselben numerischen Problemstellungen, für die es daher effizient implementierte und bequem zu bedienende Algorithmen in Bibliotheken gibt. Als Beispiel möchten wir uns in dieser Aufgabe anschauen, wie man das Bibliothekspaket Eigen, eine Sammlung von C++-Bibliotheken zu Fragestellungen aus der linearen Algebra, in einem Programm einbinden kann.

Eigen muss auf dem für diese Aufgabe verwendeten Rechner vorhanden sein, d. h. Sie müssen Eigen ggf. erst auf Ihrem Rechner installieren. In vielen Linux-Distributionen gibt es dazu vorbereitete Pakete (z.B. `libeigen3-dev` in der Debian-Familie). Eine Installationsanleitung finden Sie auch im Webangebot der Bibliothek:
`http://eigen.tuxfamily.org`

a) Laden Sie das Programmpaket zu Aufgabe 14 und entpacken Sie es. Sie finden darin mehrere Dateien. Öffnen Sie zunächst die Datei `Beispiel.cc` und versuchen Sie, den Inhalt zu verstehen. Um sie compilieren zu können, ist folgende Eingabe nötig:

```
1 clang++ -o Beispiel.exe Beispiel.cc -I /opt/local/include/eigen3/
```

Dabei benötigen wir die Option `-I`, um dem Linker, der vom Compiler aufgerufen wird, zu sagen, wo er die Eigen-Bibliothek auf dem Rechner findet. Dies ist wichtig, weil unser Programm auf Funktionen, Definitionen, ... aus der Eigen-Bibliothek zurückgreift und unsere ausführbare Datei den entsprechenden Inhalt finden muss. Auch die Kopfdateien aus `Eigen/Dense` müssen beim compilieren gelesen werden können.

ToDo: Lassen Sie das Programm laufen. Ändern Sie es so ab, dass Sie eine 4×4-Diagonalmatrix erstellen und auf dem Bildschirm ausgeben, bei der das neue Diagonalelement sich aus dem Produkt der vorherigen ergibt. Präsentieren Sie den hierfür nötigen Quelltext und die Funktionsweise Ihres Programms.

(2 Punkt(e), Votier)

b) Wie oben zu erkennen ist, ist die notwendige Eingabe zum Compilieren des Programms recht lang. Bei der Verwendung mehrerer Bibliotheken oder benötigter Compiler-Optionen wird das noch aufwendiger. Es geht aber auch einfacher. Öffnen Sie die Datei `Makefile` und gehen Sie den Inhalt durch.

Das Programm `make` ist in der Lage, anhand von Regeln, die wir in der Datei `Makefile` definieren, das Programm zu erstellen. Die Einstellungen sind so vorbereitet, dass der folgende Aufruf alle Programme des Verzeichnisses erstellt:

```
1 make
```

Beobachten Sie, wie das funktioniert.

Insbesondere das zweite Beispiel, das Programm `Multiplikation.cc`, ist interessant. Eine Funktion dieses Programms wurde in die eigene Datei `Matrixaufbau.cc` ausgelagert. Die Regeln in `Makefile` sind so definiert, dass diese Datei unabhängig vom Hauptprogramm compiliert und das ausführbare Ergebnis in der Datei `Matrixaufbau.o` (einem „object file“) abgespeichert wird. Wenn das Hauptprogramm compiliert wird, wird der Inhalt der Datei `Matrixaufbau.o` in den von `Multiplikation.exe` aufgenommen. Lohnenswert ist das insbesondere bei großen Programmen, deren Compilier-Vorgänge lange dauern. So kann man erreichen, dass immer nur das compiliert wird, was sich wirklich geändert hat.

Das Programm `make` führt immer nur benötigte Aktionen aus. Beobachten Sie, wie `make` nichts unternimmt, wenn Sie es ein zweites Mal aufrufen. Üben Sie auch das Verwenden von `make clean` und `make new`. Täuschen Sie `make` vor, dass sich die Dateien `Multiplikation.cc` und/oder `Matrixaufbau.cc` geändert hätten, indem Sie sie z.B. in einem Editor öffnen und neu speichern. Machen Sie das einzeln für die Dateien, rufen Sie danach `make` auf und beobachten Sie, wie nur benötigte Schritte ausgeführt werden, um alle Änderungen im Endergebnis `Multiplikation.exe` zu berücksichtigen.

c) Das Programm `Multiplikation.cc` enthält einen primitiven Algorithmus zur Bestimmung des betragsmäßig größten Eigenwerts und des zugehörigen Eigenvektors einer Matrix, der eine Grundidee iterativer Verfahren verdeutlicht. Dazu starten wir mit einem zufällig gewählten Vektor $\boldsymbol{v}$, und stellen uns seine Zerlegung in Eigenvektoren $\boldsymbol{x}_i$ der Matrix $\boldsymbol{M}$ vor,

$$\boldsymbol{v} = \alpha_1 \boldsymbol{x}_1 + \alpha_2 \boldsymbol{x}_2 + \dots \ .$$

Wenn wir diesen Vektor n mal auf die Matrix $\boldsymbol{M}$ multiplizieren, erhalten wir mit den Eigenwerten λ_i das Ergebnis

$$\boldsymbol{M}^n \boldsymbol{v} = \alpha_1 \lambda_1^n \boldsymbol{x}_1 + \alpha_2 \lambda_2^n \boldsymbol{x}_2 + \dots \ .$$

Damit sieht man leicht ein, dass dieses Verfahren gegen den Eigenvektor mit dem betragsmäßig größten Eigenwert konvergiert, sofern er im Anfangsvektor enthalten ist (Numerisch gibt es keine exakte Orthogonalität!).

Betrachten Sie, wie der Algorithmus umgesetzt ist und wie alle Eigenwerte und -vektoren der Matrix $\boldsymbol{M}$ anschließend mit Funktionen aus Eigen zum Vergleich berechnet werden.

d) Die Eigenwerte und Eigenvektoren, die mit dem Eigen-Paket berechnet wurden, sind nicht sortiert, allerdings entspricht die Spalte i aus der Matrix der Eigenvektoren dem Eigenvektor zum i-ten Eigenwert aus dem Vektor der Eigenwerte. Eine Sortierung nach aufsteigendem Realteil der Eigenwerte wird durch die beigelegte Funktion `Sortiere.cc` erreicht. Binden Sie diese Funktion analog zum Beispiel `Matrixaufbau.cc` so ein, dass die sortierten Eigenwerte und Eigenvektoren auf dem Bildschirm ausgegeben werden. Verändern sie die Datei `Makefile` so, dass die Datei auch beim Compilieren richtig verwendet wird.

ToDo: Präsentieren Sie den Quelltext des veränderten Programms und das geänderte `Makefile`.
(5 Punkt(e), Votier)

e) Erstellen Sie eine neue Kopie des Programms und definieren Sie eine Regel dafür in der Datei `Makefile`. Erweitern Sie die Kopie so, dass Sie mit demselben Algorithmus nun den Eigenwert

mit dem zweitgrößten Betrag und seinen normierten Eigenvektor berechnen können, wobei Sie den zuerst gefundenen Eigenvektor mit dem größten Betrag des Eigenwerts verwenden müssen.

Informieren Sie sich in der Dokumentation von Eigen, falls Sie auf bereits vorhandene Implementierungen von neu benötigten Rechenoperationen zurückgreifen möchten:
`http://eigen.tuxfamily.org`

ToDo: Präsentieren Sie die Funktionsweise Ihres neuen Algorithmus in Pseudocode.

(6 Punkt(e), Votier)

Aufgabe 15: Harmonischer Oszillator im elektrischen Feld

Der dimensionslose Hamiltonoperator eines Elektrons in einem harmonischen Potential, das zusätzlich einem elektrischen Feld der dimensionslosen Feldstärke f ausgesetzt ist, lautet

$$H = \frac{p^2}{2} + \frac{x^2}{2} - fx \; .$$

a) Zeigen Sie, dass sich das Problem mit einer Variablensubstitution $y = x + a$ und geeignetem a auf die Form eines harmonischen Oszillators mit einer konstanten Energieverschiebung bringen und somit exakt lösen lässt.

ToDo: Geben Sie Ihre Rechnung und das Ergebnis für das exakte quantenmechanische Spektrum ab.

(2 Punkt(e), Schriftlich)

b) Mit den bosonischen Erzeugern $b^\dagger$ und Vernichtern b und

$$x = \frac{1}{\sqrt{2}} \left(b^\dagger + b \right) \; , \qquad p = \frac{\mathrm{i}}{\sqrt{2}} \left(b^\dagger - b \right)$$

lautet der Hamiltonoperator

$$H = \left(b^\dagger b + \frac{1}{2} \right) - \frac{f}{\sqrt{2}} \left(b^\dagger + b \right)$$

und hat in der Basis der Energieeigenzustände $|n\rangle$ des harmonischen Oszillators die Matrixdarstellung

$$\langle n|H|m\rangle = \left(m + \frac{1}{2} \right) \delta_{n,m} - \frac{f}{\sqrt{2}} \left(\sqrt{m+1}\, \delta_{n,m+1} + \sqrt{m}\, \delta_{n,m-1} \right) \; .$$

ToDo: Schreiben Sie sich eine 3×3-Matrix dieser Form auf und überzeugen Sie sich, dass sie symmetrisch ist.

(4 Punkt(e), Schriftlich)

c) Implementieren Sie den Matrixaufbau für n Dimensionen, wobei n variabel sein soll und speichern Sie die Matrix in einem Objekt vom Typ `MatrixXd` aus dem Eigen-Paket ab.

ToDo: Präsentieren Sie den Quelltext zu Ihrem Matrixaufbau.

(3 Punkt(e), Votier)

d) Schreiben Sie ein Programm, das die dimensionslose Feldstärke f und die Matrixgröße n als Eingabewerte entgegennimmt und die Eigenwerte mit dem Eigen-Bibliothekspaket berechnet. Anschließend soll das Programm eine Wertetabelle mit den Eigenwerten der maximal 10 tiefstliegenden Zustände ausgeben.

Durch die endliche Matrixgröße erhalten wir nur eine Näherung der korrekten Eigenwerte. Lassen Sie das Programm mit $f = 2$ und $n = 5, 10, 20, 100$ durchlaufen und vergleichen Sie mit dem exakten Ergebnis. Wie zufrieden sind Sie mit der Konvergenz der 10 tiefstliegenden Zustände?

ToDo: Erstellen Sie eine Wertetabelle zu den genannten Fällen. Diskutieren Sie die Konvergenz.
(3 Punkt(e), Votier)