Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

Computerphysik

Übungsblatt 8

SS 2013

Website: http://www.thp.uni-koeln.de/trebst/Lectures/2013-CompPhys.html
Abgabedatum: Montag, 17. Juni 2013 vor Beginn der Vorlesung

Auf diesem Übungsblatt wollen wir uns Problemstellungen der **linearen Algebra** widmen.

29. Lineare Algebra mit Python

Programmiertechniken

Bevor wir uns auf den folgenden Zetteln mit den Algorithmen der **linearen Algebra** beschäftigen, führen wir die wichtigsten Funktionen ein, die im **numpy**-Paket enthalten sind und die wir für die folgenden Aufgaben benötigen.

Eine **Matrix** mit leeren Einträgen erzeugen Sie mit

```
a_matrix = np.empty((rows, cols))
```

wobei *rows* und *cols* die Anzahl der Zeilen und Spalten angeben. Eine Matrix, die bereits mit Nullen gefüllt ist, wird mit

```
a_matrix = np.zeros((rows, cols))
```

initialisiert. Einen **Vektor** können Sie ganz genauso initialisieren, indem Sie nur eine Dimension angeben:

```
a_vector = np.zeros((entries))
```

Ob Sie einen Spalten- oder Zeilenvektor erstellen wollen ist egal, denn die entsprechenden Funktionen erkennen dies automatisch.

Nachdem Sie nun die für Ihren Algorithmus benötigten Matrizen und Vektoren erstellt haben, müssen Sie diese auch mit Werten füllen und manipulieren. Der Zugriff auf ein Element geschieht mit dem von Arrays bekannten eckigen Klammern:

```
print a_matrix[2, 0]
print a_vector[10]
```

Besonders praktisch ist die Möglichkeit, gleich einen ganzen Bereich zu manipulieren, den sie mit der Notation *Start:Ende* angeben können. Angenommen Sie haben eine 10x10 Matrix und wollen die obere rechte 5x5 Matrix ausgeben, dann würden Sie schreiben

```
print a_matrix[0:5, 5:10]
```

denn der Bereich wird durch die Zeilen 0 und 5 und durch die Spalten 5 und 10 begrenzt. Beachten Sie, dass der erste Wert zum Bereich gehört, der Zweite aber nicht. In der Mathematik entspricht dies Intervallen des Typs $[a,b)$. Erwähnt seien noch zwei praktische Schreibweisen. Wenn Sie in einer der Dimensionen alle Werte bis zu oder ab einem Wert benutzen wollen, dann müssen Sie nur einen Index angeben. Konkret sieht dies so aus:

```
print a_matrix[:5, 5:]
```

Wenn Sie alle Einträge bis zum Vorletzten, dem Vorvorletzten etc. auswählen möchten, so geben Sie negative Werte an. Unser vorheriges Beispiel wird dann zu

```
print a_matrix[:-5, -5:]
```

und liest sich: Wähle den Bereich aus, der begrenzt wird durch alle Zeilen bis auf die fünf Letzten und die fünf letzten Spalten.

Widmen wir uns nun den benötigten **Operationen**. Beachten Sie, dass Ausdrücke wie

```
matrix_3 = matrix_1 * matrix_2
another_vector = matrix * vector
```

keine Matrix-Matrix oder Matrix-Vektor Multiplikationen sondern elementweise Operationen sind. Stattdessen verwenden wir die Funktion *dot*

```
matrix_3 = np.dot(matrix_1, matrix_2)
another_vector = np.dot(matrix, vector)
```

Sie werden außerdem lineare **Gleichungssysteme lösen** müssen, was denkbar einfach mit der Funktion *solve* erledigt wird. Sie ist sogar in der Lage allgemeine Gleichungssysteme des Typs $A \cdot X = B$, wobei sowohl A als auch B und X Matrizen sind, zu lösen. Der Aufruf sieht folgendermaßen aus:

```
import numpy.linalg as npl

# solves AX = B for X
X = npl.solve(A, B)
```

Um Algorithmen zu testen ist es von Vorteil **Zufallsmatrizen** als Eingabe zu benutzen um viele verschiedene Testfälle zu generieren. Auch dazu bietet numpy mit dem Modul *numpy.random* die Möglichkeit. Die Funktion *rand* erzeugt einen n-dimensionalen Array mit Zufallszahlen zwischen 0 und 1. Wie dies genau funktioniert, werden wir auf einem der späteren Übungsblätter noch sehen. Die gewünschten Dimensionen übergeben wir als Parameter:

```
import numpy.random as npr

# a 6x8 random matrix
a_random_matrix = npr.rand(6, 8)
```

Alle diese Funktionen werden Sie in den nachfolgenden Aufgaben anwenden. Insbesondere die Möglichkeit ganze Bereiche auf einmal zu manipulieren, kann Ihre Programme erheblich beschleunigen. Verändern Sie Ihr Programm um das Potential eines Plattenkondensators zu berechnen so, dass statt der zwei Schleifen über die Gitterpunkte in x- und y-Richtung nur eine Zeile mit Bereichsnotation übrig bleibt und vergleichen Sie die Geschwindigkeit.

30. Matrix halb und halb *5 Punkte*

Eine praktische Erfahrung, die wir immer wieder machen ist, daß nicht jeder Algorithmus, der auf dem Papier wohldefiniert ist, auch auf dem Computer die gewünschten Resultate liefert. Ein "funktionierender" Algorithmus mag etwa nicht die nötige Geschwindigkeit besitzen oder in numerische Instabilitäten laufen. Derartige Überlegungen sind speziell für die Verfahren der linearen Algebra oft von besonderer Bedeutung.

Ein Lösungsweg, einen Algorithmus weiter zu optimieren, ist dabei häufig, die **Darstellung der Matrizen** zu wechseln. Eine Möglichkeit ist etwa, eine Matrix M als das **Produkt** zweier Matrizen A und B zu schreiben, d.h. $M = A \cdot B$. Wir werden in dieser Aufgabe sehen, daß dies sehr vorteilhafte Eigenschaften produzieren kann.

Untersuchen Sie dazu beispielhaft einen Algorithmus, dessen Implementation wir Ihnen als decomposition.py für diese Übung zur Verfügung stellen. Lesen Sie den Quellcode, ergänzen Sie Kommentare im Code und beschreiben Sie für jede Zeile den konzeptionellen Schritt. Fassen Sie schließlich in knapper Form (drei Sätze) zusammen, was der Algorithmus bewerkstelligt.

Erzeugen Sie dann eine quadratische Zufallsmatrix M, wie Sie es im oben stehenden Python-Crashkurs gelernt haben, und führen Sie die Funktion mit

```
A, B = decomposition(M)
```

aus. Welche Form besitzen die Matrizen A und B? Untersuchen Sie diese Matrizen außerdem auf Symmetrie und Orthogonalität.

31. Der heiße Draht 2.0 *5 Punkte*

Auf Übungsblatt 6 haben Sie die **Wärmeleitungsgleichung** als Anfangswertproblem mit der Euler-Methode bearbeitet. Dabei mussten Sie feststellen, dass die Stabilität des Algorithmus stark von der Diskretisierung dx in der räumlichen und dt in der zeitlichen Dimension abhing.

Diese Einschränkung wird durch das **Crank-Nicolson-Verfahren**, welches Sie in der Vorlesung besprochen haben, aufgehoben. Wir wollen dies in dieser Aufgabe überprüfen. Die Iterationsschritte des Algorithmus haben Sie in der Vorlesung ausführlich besprochen. Wie Sie sich erinnern, müssen Sie an einem Schritt ein **lineares Gleichungssystem** lösen, was Sie mit der in unserem Crashkurs erwähnten Funktion *solve* tun können.

Implementieren Sie das Crank-Nicolson-Verfahren für die eindimensionale Wärmeleitungsgleichung

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

mit Anfangsverteilung bei $t = 0$

$$u(x,0) = \sin(\pi x), \quad x \in (0,1)$$

und untersuchen Sie die Stabilität indem Sie $20, 200$, und 2000 räumliche Diskretisierungsschritte verwenden.