Computerphysik WS 2015/2016

Prof. Dr. Roland Netz, FU Berlin

Übungsblatt 3: Lineare Algebra

Alexander Schlaich, Johann Hansing
29. Oktober 2015

Allgemeine Hinweise

Abgabetermin für die Lösungen ist

- **Sonntag, 08.11., 24:00 Uhr.**

Aufgabe 3.1: Lineare Gleichungssysteme: Gauß-Eliminierung (10 Punkte)

Gegeben sei das Gleichungssystem $\mathbf{Ax} = \mathbf{b}$ mit

$$\mathbf{A} = \begin{pmatrix} 3.0 & 1.0 & 4.0 & -1.0 \\ 2.0 & -2.0 & -1.0 & 2.0 \\ 5.0 & 7.0 & 14.0 & -8.0 \\ 1.0 & 3.0 & 2.0 & 4.0 \end{pmatrix} \text{ und } \mathbf{b} = \begin{pmatrix} 7.0 \\ 1.0 \\ 20.0 \\ -4.0 \end{pmatrix}.$$

- 3.1.1 (4 Punkte): Schreiben Sie eine Funktion, die den Gauß-Algorithmus implementiert. Als Argumente sollte die Funktion ein $(N \times N)$-`numpy.array` **A** und einen $(N \times 1)$-`numpy.array` **b** nehmen und den Lösungsvektor **x** als `numpy.array` zurückgeben. Prüfen Sie in jedem Schritt ob die Eliminierung durchgeführt werden kann und vertauschen Sie ggf. mit einer darunter liegenden Zeile.

Hinweis: Beachten Sie, dass

```
a = np.array([1,2,3])
b = a
b += 1
print (a)
```

die Ausgabe `[2 3 4]` erzeugt, NumPy also nur sogenannte flache Kopien anlegt. Um *echte* Kopien anzulegen, können Sie `np.copy(a)` verwenden. Achten Sie außerdem darauf Ihre Arrays mit dem Datentyp `float` anzulegen. Hierzu können Sie entweder die Kommastellen angeben wie oben oder explizit `dtype=float` verwenden.

Achten Sie außerdem darauf, dass in allen Fällen während des Eliminationsverfahrens geprüft werden muss, ob ein Diagonalelement Null ist.

- 3.1.2 (2 Punkte): Erweitern Sie die Funktion so, dass die Rechnungen mit n-stelliger Genauigkeit ausgeführt werden, d.h. nach jeder arithmetischen Operation wird die Zahl auf n Dezimalstellen gerundet. Verwenden Sie hierzu wieder die Hilfsfunktion `mround` von Übungsblatt 1 (siehe auch nächste Seite). Passen Sie Ihre Funktion so an, dass n als Argument übergeben wird.

- 3.1.3 (4 Punkte): Bei der Konstruktion der rechten oberen Dreiecksmatrix können Rundungsfehler bei der Multiplikation der Matrix- und Vektorelemente mit a_{ii}^{-1} verstärkt werden. Durch geeignete Zeilenvertauschung kann $\left|\frac{a_{ji}}{a_{ii}}\right| \leq 1$ mit $j > i$ bei jedem Zwischenschritt erreicht werden.

  Erweitern Sie das Programm so, dass Spaltenpivotisierung benutzt wird, wenn ein weiteres Argument `pivot=True` übergeben wird.

  Berechnen Sie $\mathbf{x}$ mithilfe der geschriebenen Funktionen in folgenden Variationen:

  – Für $n = 7$ mit und ohne Spaltenpivotisierung.

  – Mit `float`-Genauigkeit mit und ohne Spaltenpivotisierung.

  Vergleichen Sie die Ergebnisse untereinander und außerdem mit denen von `numpy.linalg.solve()`.

```
def mround(x,N):
    if (x==0):
        return x
    return round(x, int(N - math.ceil(math.log10(abs(x)))))
```

Aufgabe 3.2: LR Zerlegung Matrix-Inversion (10 Punkte)

- 3.2.1 (4 Punkte):

  Implementieren Sie eine Funktion, welche für eine als Argument übergegebene $(N \times N)$-Matrix $\mathbf{A}$ die in der Vorlesung definierte Matrix G zurückgibt (Zur Erinnerung: $\mathbf{G}$ erfüllt $\mathbf{G} \cdot \mathbf{A} = \mathbf{R}$ wobei $\mathbf{R}$ eine rechte obere Dreiecksmatrix ist). Nehmen Sie an, dass keine Zeilenvertauschungen notwendig sind.

- 3.2.1 (2 Punkte): Betrachten Sie nun die Matrix

$$\mathbf{A} = \begin{bmatrix} 2.0 & 1.0 & 0 \\ 1.0 & -1.0 & 1.0 \\ 3.0 & 2.0 & 1.0 \end{bmatrix}. \tag{1}$$

  Nutzen Sie ihre Funktion aus Aufgabe 3.2.1 um die dazugehörige Matrix $\mathbf{G}$ zu berechnen und überprüfen Sie das Ergebnis durch Berechnen von $\mathbf{G} \cdot \mathbf{A}$ mithilfe von `numpy.dot()`: Liefert die Matrixmultiplikation eine rechte obere Dreiecksmatrix? Hinweis: Auch hier bleibt es wichtig, die numpy arrays mit dem Datentyp float anzulegen!

Sobald $\mathbf{G}$ berechnet wurde, ist es einfach, die Gleichung $\mathbf{Ax} = \mathbf{b}$ zu lösen. Zunächst wird die Gleichung auf beiden Seiten mit $\mathbf{G}$ multipliziert, sodass sich unter Benutzung von $\mathbf{G} \cdot \mathbf{A} = \mathbf{R}$ die Gleichung

$$\mathbf{Rx} = \mathbf{Gb} \tag{2}$$

ergibt. Nachdem die rechte Seite ausgerechnet wurde, kann die Gleichung nun durch Rückwärtssubstitution nach $\mathbf{x}$ aufgelöst werden.

- 3.2.3 (1 Punkte): Benutzen Sie $\mathbf{G}$, um die Gleichung $\mathbf{Ax} = \mathbf{b}$ mit $\mathbf{b} = [-8.0, 15.0, 34.0]^T$ zu lösen. Vergleichen Sie das Ergebnis mit dem von `numpy.linalg.solve()`.

- 3.2.4 (3 Punkte): Nun können Sie die Inverse der Matrix $\mathbf{A}$ berechnen. Ermitteln Sie dazu die Lösung von

$$\mathbf{A}\mathbf{x_i} = \mathbf{e_i} \tag{3}$$

wobei $\mathbf{e_i}$ die Einheitsvektoren $\mathbf{e_1} = [1.0, 0, 0]^T$, $\mathbf{e_2} = [0, 1.0, 0]^T$ und $\mathbf{e_3} = [0, 0, 1.0]^T$ sind. Die Vektoren $\mathbf{x_i}$ sind dann die Spalten der inversen $\mathbf{A^{-1}} = (\mathbf{x_1}\ \mathbf{x_2}\ \mathbf{x_3})$. Überprüfen Sie ihr Ergebnis, indem Sie $\mathbf{A} \cdot \mathbf{A^{-1}} = \mathbf{1}$ berechnen. $\mathbf{1}$ ist die Einheitsmatrix. Hinweis: Beachten Sie, ob Sie im letzten Schritt die Inverse, oder die transponierte der Inverse berechnet haben.

Gilt $\mathbf{A} \cdot \mathbf{A^{-1}} = \mathbf{A^{-1}} \cdot \mathbf{A}$ im Allgemeinen und in Ihrer Implementierung?