MATHEMATISCHES INSTITUT
PROF. DR. CHRISTIANE HELZEL
ANDREAS TROLL

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

14.12.2017

Computergestützte Mathematik zur linearen Algebra – 10. Übungsblatt

WICHTIG: Kommentieren Sie Ihren Quelltext. Ihre Skripte müssen durch ausführen des „run"-Befehls (grünes Dreieck bzw. F5) lauffähig sein.

Aufgabe 34: (*Kompakte LR-Zerlegung*)

Skriptname: *lr_zerlegung_in_place.py*
`P,L,R=PLR(A)` (siehe Vorlesung 9) gibt als Rückgabewert die Matrizen für die LR-Zerlegung $PA = LR$ zurück. In Aufgabe 30 haben Sie schon eine speicherschonendere Variante implementiert, welche statt `P` nur einen Vektor `p` mit den Permutationen zurückgibt. Es geht aber noch effizienter:
Implementieren Sie eine Funktion `p=PLReff(A)`, welche die LR-Zerlegung in der Eingabematrix `A` speichert (A also „in-place" überschreibt) und nur den Permutationsvektor `P` zurückgibt.
Hinweise:

- R ist eine obere Dreiecksmatrix und L eine untere Dreiecksmatrix. Wenn man voraussetzt, dass L im Algorithmus so normiert wird, dass die Diagonalelemente 1 sind, kann man R und L ohne zusätzlichen Speicherplatz in der Variable A speichern.
- Sie sollen die Funktion auch speichereffizient implementieren. Erstellen Sie in Ihrer Funktion also NICHT Matrizen P, L und R und überschreiben dann am Ende A.
- L und R erhält man nach Anwendung von `PLReff(A)` durch: `L=np.tril(A,-1)+np.eye(A.shape[0])` und `R=np.triu(A)`.
- Kommentieren Sie Ihren Quelltext und testen Sie Ihre Funktion an geeigneten Beispielen. Wir werden Sie fragen was Sie getan haben.

Aufgabe 35: (*Schleife zu Slices*)

Skriptname: *Gramschmidt_ohne_schleife.py*
Ersetzen Sie die innere Schleife aus `GramSchmidt(A)` (Vorlesung 10) durch Slices. Der neue Funktionsname soll `GramSchmidtSlices` sein. Testen Sie Ihre Funktion an geeigneten Beispielen und vergleichen Sie sie mit der Ursprungsversion.

Aufgabe 36: (*QR-Zerlegung bei Ausgleichsrechnung*)

Skriptname: *Ausgleich_QR.py*

(a) Lesen Sie die Daten aus der Textdatei `daten.txt` (siehe Vorlesungseite) ein. Die erste Zahl jeder Zeile ist ein x-Wert und die zweite Zahl ist der zugehörige y-Wert. Beachten Sie, dass die einzelnen Datenpunkte diesmal durch Kommas (und nicht durch Leerzeichen) getrennt sind. Gucken Sie sich also die Hilfe der Stringfunktion `split` oder von `np.loadtxt` an.

(b) Schreiben Sie eine Funktion `A=erstelle_matrix(x,n)`, die zu gegebenen Punkten `x` die Matrix A für das Ausgleichsproblem mit einem Polynom von Grad `n` erstellt.

(c) Erstellen Sie Ausgleichspolynome mit den beiden Methoden aus Vorlesung 6 und 10 der Grade n für $n = 5, 10, 15, 20, 25, 30$. Zeichnen Sie jeweils die Lösung der A^TA- und der QR-Variante zusammen mit den Datenpunkten in einen gemeinsamen Plot (eine Schleife erspart Ihnen hier viel Arbeit).

(d) Machen Sie sich klar, dass beide Varianten (A^TA und QR) mathematisch gesehen die selben Lösungen berechnen sollten. Wieso tun sie das in dieser Aufgabe nicht?

Aufgabe 37: (*Fehlersuche*)

Finde zu einer aufsteigend sortierten Folge von Zahlen $v_0 \leq v_2 \leq \ldots \leq v_n \in \mathbb{Z}$ (Datenbasis) und einer Zahl $x \in \mathbb{Z}$ den kleinsten Index i, so dass $v_i = x$ gilt.

Der folgende Code beschreibt die sogenannte "Binäre Suche" zur Lösung dieser Aufgabe. Durch Vergleich von x mit dem mittleren Element $v_{\lfloor \frac{n}{2} \rfloor}$ der Folge ($\lfloor \frac{n}{2} \rfloor \in \mathbb{N}$ ist $\frac{n}{2}$ abgerundet) kann die Suche auf eine Folge der halben Länge beschränkt werden. Indem man diesen Vorgang *rekursiv* wiederholt, benötigt man nur $\log_2(n)$ Vergleiche um festzustellen, ob und wo x in der Folge zu finden ist.

Das folgende PYTHON-Programm `index = bisu(v,x)` findet mittels binärer Suche das kleinste `index`, so dass für die aufsteigend sortierte Liste `v v[index]==x` gilt. Sollte `x` nicht in `v` enthalten sein, so wird `float('nan')` zurückgegeben.

Es haben sich jedoch mindestens 6 Fehler im Code versteckt. Finden und berichtigen Sie diese (den Quellcode können Sie sich von der Vorlesungsseite herunterladen).

```
1  def bisu(v,x):
2      n=v.len()
3      if n==1:
4          if v[0]==x:
5              return 0
6      else: return float('nan')
7      else:
8          m=int((n-1)/2)
9          if x <=v[n]:
10             return bisu(v[:m],x)
11         else:
12             return m-1+bisu(v[m+1:],x)
```

Sie können Ihre korrigierte Version z.B. testen mit

```
v = [ 0, 1, 1, 2, 2, 4, 4, 4, 5, 6, 7, 7, 7, 8, 9 ];
[ bisu(v, kk) for kk in range(10) ]
```

Besprechung in den Übungen vom 18.-22.12. 2017.