Abgabe - Übungsblatt [9]

Angewandte Mathematik: Numerik

[Felix Lehmann] [Markus Menke]

21. Januar 2021

Aufgabe 1

Cholesky

1. Berechne Cholesky Zerlegung von A
2. Löse unteres Dreieckssystem $R^*w = A^*b$ nach w
3. Löse oberes Dreieckssystem $Rx = w$ nach x

Instabil, nur für kleine Probleme
nur für hermitesche positiv-definite Matrizen

QR

1. Berechne QR Zerlegung von A
2. Berechne Q^*b
3. Löse oberes Dreieckssystem $Rx = Q^*b$ nach x

Stabiler, Standardverfahren

SVD

1. Berechne SVD Zerlegung von A
2. Berechne U^*b
3. Löse Diagonalsystem $\Sigma w = U^*b$ nach w
4. Berechne $x = Vw$

Ähnliche effizient wie QR wenn $m >> n$, sonst teurer aber auch noch stabiler

Aufgabe 2

a)

$$\begin{pmatrix}1 & 2 & 1\\ 2 & 8 & 2\\ 1 & 2 & 2\end{pmatrix} = \begin{pmatrix}l_{1,1} & 0 & 0\\ l_{2,1} & l_{2,2} & 0\\ l_{3,1} & l_{3,2} & l_{3,3}\end{pmatrix} \cdot \begin{pmatrix}l_{1,1} & l_{2,1} & l_{3,1}\\ 0 & l_{2,2} & l_{3,2}\\ 0 & 0 & l_{3,3}\end{pmatrix} = \begin{pmatrix}l_{1,1}^2 & l_{1,1}l_{2,1} & l_{1,1}l_{3,1}\\ l_{1,1}l_{2,1} & l_{2,1}^2 + l_{2,2}^2 & l_{2,1}l_{3,1} + l_{2,2}l_{3,2}\\ l_{1,1}l_{3,1} & l_{2,1}l_{3,1} + l_{2,2}l_{3,2} & l_{3,1}^2 + l_{3,2}^2 + l_{3,3}^2\end{pmatrix}$$

$$\begin{aligned}
l_{1,1}^2 &= 1 \rightarrow l_{1,1} = 1\\
l_{1,1}l_{2,1} &= 2 \rightarrow l_{2,1} = 2\\
l_{1,1}l_{3,1} &= 1 \rightarrow l_{3,1} = 1\\
l_{2,1}^2 + l_{2,2}^2 &= 8 \rightarrow l_{2,2} = 2\\
l_{2,1}l_{3,1} + l_{2,2}l_{3,2} &= 2 \rightarrow l_{3,2} = 0\\
l_{3,1}^2 + l_{3,2}^2 + l_{3,3}^2 &= 2 \rightarrow l_{3,3} = 1
\end{aligned}$$

$$L = \begin{pmatrix}1 & 0 & 0\\ 2 & 2 & 0\\ 1 & 0 & 1\end{pmatrix} \; R = \begin{pmatrix}1 & 2 & 1\\ 0 & 2 & 0\\ 0 & 0 & 1\end{pmatrix}$$

b)

$$\begin{aligned}
R^* w &= b\\
\begin{pmatrix}1 & 0 & 0\\ 2 & 2 & 0\\ 1 & 0 & 1\end{pmatrix} w &= \begin{pmatrix}\\ 2\\ 1\end{pmatrix} Rx & &= w
\end{aligned}$$

Aufgabe 3

```
import scipy.io
import numpy as np
from matplotlib import pyplot

iterations = 100

def PowerIteration(A, v):
    """Given a square matrix A and a column vector v of
        matching size to initialize
       the iteration this function implements the power
           iteration to approximate the
       largest Eigenvalue of A. It returns a list of
           successive approximations where
       the last approximation is the best one."""
    approx = []
    vector = v
    Av = A.dot(vector)
    for _ in range(iterations):
        vector = Av / np.linalg.norm(Av)
```

```
        Av = A.dot(vector)
        val = vector.flatten().dot(Av)
        approx.append(val)
    return approx


def RayleighQuotientIteration(A, v):
    """Given a square matrix A and a column vector v of
        matching size to initialize
       the iteration this function implements the
           Rayleigh quotient iteration to
       approximate the Eigenvalue of A whose Eigenvector
           is closest to v. It returns
       a list of successive approximations where the last
            approximation is the best
       one."""
    approx = []
    vector = v
    I = np.eye(A.shape[0])
    for _ in range(iterations):
        u = vector/np.linalg.norm(vector)
        val = np.dot(u.flatten(), np.dot(A, u))
        approx.append(val)
        vector = np.linalg.solve(A-val*I, u)
    u = vector/np.linalg.norm(vector)
    val = np.dot(u.flatten(), np.dot(A, u))
    approx.append(val)
    return approx


Matrizen = scipy.io.loadmat("Matrizen.mat")
A1 = Matrizen["A1"]
A2 = Matrizen["A2"]
Startvektor = Matrizen["Startvektor"]

pyplot.suptitle('Power Iteration')
LambdaList = PowerIteration(A1, Startvektor)
pyplot.plot(LambdaList, color="g")
LambdaList = PowerIteration(A2, Startvektor)
pyplot.plot(LambdaList, color="r")

pyplot.figure()
pyplot.suptitle("Rayleigh Quotient Iteration")
LambdaList = RayleighQuotientIteration(A1, Startvektor)
pyplot.plot(LambdaList, "g")
LambdaList = RayleighQuotientIteration(A2, Startvektor)
pyplot.plot(LambdaList, "r")

pyplot.show()
```