Abgabe - Übungsblatt [8]

Angewandte Mathematik: Numerik

[Felix Lehmann] [Markus Menke]

15. Januar 2021

Aufgabe 1

a)

$$U = \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 1.4 & 1.1 & -0.7 \\ 0.8 & 4.3 & 2.1 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Zeilen sind mit römischen Ziffern aufsteigend zu interpretieren:

$II = II - (10/23 * 14/10)I$

$$U = \begin{pmatrix} 23/10 & 9/5 & 1 \\ 0 & 1/230 & -301/230 \\ 4/5 & 43/10 & 21/10 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 14/23 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$III = III - (10/23 * 8/10)I$

$$U = \begin{pmatrix} 23/10 & 9/5 & 1 \\ 0 & 1/230 & -301/230 \\ 0 & 169/46 & 403/230 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 14/23 & 1 & 0 \\ 8/23 & 0 & 1 \end{pmatrix}$$

$III = III - (230 * 169/46)II$

$$U = \begin{pmatrix} 23/10 & 9/5 & 1 \\ 0 & 1/230 & -301/230 \\ 0 & 0 & 5538/5 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 14/23 & 1 & 0 \\ 8/23 & 845 & 1 \end{pmatrix}$$

L ist nun untere Dreiecksmatrix und U ist obere Dreiecksmatrix. Der Ergebnis entspricht dem erwarteten Verhalten.

Wir errechnen nun die Lösung des LGS:

$L * (U * x) = b$

Vorwärtseinsetzen (y = Ux):

$$\begin{pmatrix} 1 & 0 & 0 \\ 14/23 & 1 & 0 \\ 8/23 & 845 & 1 \end{pmatrix} * \begin{pmatrix} y1 \\ y2 \\ y3 \end{pmatrix} = \begin{pmatrix} 1.2 \\ -2.1 \\ 0.6 \end{pmatrix}$$

$\Rightarrow y = \begin{pmatrix} 6/5 \\ -651/230 \\ 23919/10 \end{pmatrix}$ Jetzt lösen wir das verbleibende LGS:

$U * x = y$

Welches sich durch Rückwärtseinsetzen lösen lässt:

$$x = \begin{pmatrix} 323/923 \\ -3619/3692 \\ 7973/3691 \end{pmatrix}$$

Da ich die Aufgabe nicht richtig gelesen habe und genau gerechnet habe, ist der relative und absolute Fehler = 0. Ich vermute an dieser Stelle, dass der Fehler recht hoch ist, da die LR Zerlegung numerisch instabil ist. Der relative Fehler wäre an dieser Stelle durch $(|x_{gegeben} - x_{berechnet}|/x_{gegeben}$ berechnet worden.

b)

$U = \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 1.4 & 1.1 & -0.7 \\ 0.8 & 4.3 & 2.1 \end{pmatrix}$ $L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Maximales $|u_{i1}| = 2.3 \Rightarrow$ Keine Zeile muss getauscht werden.
Maximales $|u_{i2}| = 4.3 \Rightarrow$ Tausche Zeilen II und III in U und P.

$U = \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 0.8 & 4.3 & 2.1 \\ 1.4 & 1.1 & -0.7 \end{pmatrix}$

$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Jetzt lösen wir: $P * A = L * U$
Analog zu Teilaufgabe b) können wir nun L und U berechnen.
0en erste Spalte:

$U = \begin{pmatrix} 2.3 & 1.8 & 1 \\ 0 & 3.67 & 1.75 \\ 0 & 0 & -1.31 \end{pmatrix}$ $L = \begin{pmatrix} 1 & 0 & 0 \\ 0.34 & 1 & 0 \\ 0.61 & 0 & 1 \end{pmatrix}$

U ist bereits eine obere Dreiecksmatrix
Wir lösen $P * L * U * x = b$
Sei $y = Ux$
Durch Vorwärtseinsetzen erhalten wir y:

$y = \begin{pmatrix} 1.2 \\ -2.51 \\ -0.13 \end{pmatrix}$

Durch Rückwärtseinsetzen erhalten wir nun x:

$x = \begin{pmatrix} 1.05 \\ -0.73 \\ 0.1 \end{pmatrix}$

Jetzt muss $P * L * U * x$ b sein
Da ich mich irgendwo massiv verrechnet habe stimmt das leider nicht ganz....
Der relative Fehler sollte bei Aufgabeteil b deutlich kleiner sein. Grund dafür ist, dass wir nach weniger Schritten (d.h. auch weniger Rundungen) bereits zum Ergebnis kommen, da U schneller in eine obere Dreiecksmatrix umgewandelt werden kann.

Aufgabe 2

```
import numpy as np
import scipy.linalg as la
```

```
def Pmatrix(A):
    m = len(A)

    id_mat = np.identity(m,dtype=float)

    for j in range(m):
        row = max(range(j, m), key=lambda i: abs(A[i,j]))
        if j != row:
            id_mat[[j, row],:] = id_mat[[row, j],:]
            A[[j, row],:] = A[[row, j],:]

    return id_mat

def LUP(A):
    """Computes and returns an LU decomposition with
        pivoting. The return value
        is a tuple (L,U,P) and fulfills L*U=P*A (* is
            matrix-multiplication)."""
    n = len(A)
    U = np.zeros_like(A)
    L = np.zeros_like(A)
    P = Pmatrix(A)
    PA = P@A

    for j in range(n):
        L[j,j] = 1.0

        for i in range(j+1):
            s1 = sum(U[k,j] * L[i,k] for k in range(i))
            U[i,j] = PA[i,j] - s1

        for i in range(j, n):
            s2 = sum(U[k,j] * L[i,k] for k in range(j))
            L[i,j] = (PA[i,j] - s2) / U[j,j]

    return (L, U, P)

def ForwardSubstitution(L,b):
    """Solves the linear system of equations L*x=b
        assuming that L is a left lower
        triangular matrix. It returns x as column vector.
            """
    x = np.zeros_like(b)
    x[0] = b[0] / L[0,0]
    for i in range(1,len(b)):
        x[i] = b[i] - sum([L[i,j]*x[j] for j in range(1,i
            )])
    return x

def BackSubstitution(U,b):
```

```
    """Solves the linear system of equations U*x=b
        assuming that U is a right upper
        triangular matrix. It returns x as column vector.
            """
    x = np.zeros_like(b)
    n = len(x)
    x[n-1] = b[n-1] / U[n-1,n-1]

    for i in range(0,n-2):
        x[i] = (b[i] - sum([U[i,j]*x[j] for j in range(i
            +1,n-1)])) / U[i,i]
    return x

def SolveLinearSystemLUP(A,b):
    """Given a square array A and a matching vector b
        this function solves the
        linear system of equations A*x=b using a pivoted
            LU decomposition and returns
        x."""
    L,U,_ = LUP(A)
    y = ForwardSubstitution(L,b)
    x = BackSubstitution(U,y)
    return x

def LeastSquares(A,b):
    """Given a matrix A and a vector b this function
        solves the least squares
        problem of minimizing |A*x-b| and returns the
            optimal x."""



if(__name__=="__main__"):
    # A test matrix where LU fails but LUP works fine
    A=np.array([[1,2, 6],
                [4,8,-1],
                [2,3, 5]],dtype=np.double)
    b=np.array([1,2,3],dtype=np.double)
    # Test the LUP-decomposition
    L,U,P=LUP(A)
    print("L")
    print(L)
    print("U")
    print(U)
    print("P")
    print(P)
    print("Zero (LUP sanity check): "+str(np.linalg.norm(
        np.dot(L,U)-np.dot(P,A))))
    # Test the method for solving a system of linear
        equations
```

```
    print("Zero (SolveLinearSystemLUP sanity check): "+
        str(np.linalg.norm(np.dot(A,SolveLinearSystemLUP(A
        ,b))-b)))
    # Test the method for solving linear least squares
    A=np.random.rand(6,4)
    b=np.random.rand(6)
    print("Zero (LeastSquares sanity check): "+str(np.
        linalg.norm(LeastSquares(A,b).flat-np.linalg.lstsq
        (A,b)[0])))
```

Aufgabe 3

```
from math import sqrt
import numpy as np

def cholesky(A):
    n = len(A)
    L = np.zeros_like(A)

    for i in range(n):
        for k in range(i+1):
            tmp_sum = sum(L[i,j] * L[k,j] for j in range(
                k))

            if (i == k):
                L[i,k] = sqrt(A[i,i] - tmp_sum)
            else:
                L[i,k] = (1.0 / L[k,k] * (A[i,k] -
                    tmp_sum))
    return L

A = np.array( [[6, 3, 4, 8], [3, 6, 5, 1], [4, 5, 10, 7],
      [8, 1, 7, 25]],dtype=np.double)
L = cholesky(A)
R = L.transpose()
print("A:")
print(A)

print("L:")
print(L)

print("R:")
print(R)

print("R* R")
print(R.transpose()@R)
```