LR-Zerlegung

October 26, 2018

In [1]:
```
%%latex
\parindent0cm
```

1 LR-Zerlegung

Demo-Implementierung zur LR-Zerlegung. Hier geht es nicht um Geschwindigkeit, sondern um Verständnis. Wir beginnen mit einer symbolischen Zerlegung und ziehen dabei die Überlegungen der Vorlesung nach. Auf Pivotsuche gehen wir nicht ein.
Hinweis für dieses Arbeitsblatt: Alle Indizierungen beginnen bei 0.
Wir importieren zunächst die benötigten Bibliotheken.

In [2]:
```
import sympy
from sympy import Matrix
sympy.init_printing(use_unicode=True)
import numpy as np
```

1.1 Elementar- oder Frobenius-Matrizen

Wir starten mit den in der Vorlesung besprochenen Elementarmatrizen und demonstrieren die dort gezeigten Sätze an symbolischen Matrizen. Wir erzeugen eine volle A- und L-Matrix.

In [3]:
```
N=4
A=Matrix(sympy.MatrixSymbol('A',N,N))
L=Matrix(sympy.MatrixSymbol('L',N,N))
A
```

Out[3]:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}$$

Elementarmatrizen sind Einheitsmatrizen mit Einträgen unterhalb der Hauptdiagonalen in der i. Spalte. Wir wählen ein i.

In [4]:
```
i=1
Li=sympy.eye(N)
Li[i+1:N,i]=L[i+1:N,i]
Li
```

Out[4]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & L_{2,1} & 1 & 0 \\ 0 & L_{3,1} & 0 & 1 \end{bmatrix}$$

Wir testen die Wirkung von L_i auf die Matrix A.

In [5]:
```
Li*A
```

Out[5]:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{1,0}L_{2,1} + A_{2,0} & A_{1,1}L_{2,1} + A_{2,1} & A_{1,2}L_{2,1} + A_{2,2} & A_{1,3}L_{2,1} + A_{2,3} \\ A_{1,0}L_{3,1} + A_{3,0} & A_{1,1}L_{3,1} + A_{3,1} & A_{1,2}L_{3,1} + A_{3,2} & A_{1,3}L_{3,1} + A_{3,3} \end{bmatrix}$$

Das haben wir erwartet: In den Zeilen $i+1 \ldots N-1$ wird ein Vielfaches der i. Zeile addiert.
Um diese Operation rückgängig zu machen, müssen wir einfach nur dieses Vielfache wieder abziehen. Die Inverse ist also leicht angebbar:

In [6]:
```
Li_invers=sympy.eye(N)
Li_invers[i+1:N,i]=-L[i+1:N,i]
Li_invers
```

Out[6]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -L_{2,1} & 1 & 0 \\ 0 & -L_{3,1} & 0 & 1 \end{bmatrix}$$

In [7]:
```
Li*Li_invers
```

Out[7]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Wir erzeugen eine zweite Elementarmatrix in einer anderen Spalte.

In [8]:
```
k=0
Lk=sympy.eye(N)
Lk[k+1:N,k]=L[k+1:N,k]
Lk
```

Out[8]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{1,0} & 1 & 0 & 0 \\ L_{2,0} & 0 & 1 & 0 \\ L_{3,0} & 0 & 0 & 1 \end{bmatrix}$$

$L_k\, L_i$ für $k < i$ ist einfach nur die Überlagerung der Elementarmatrizen. Entsprechend für weitere Multiplikationen.

In [9]:
```
Lk*Li
```

Out[9]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{1,0} & 1 & 0 & 0 \\ L_{2,0} & L_{2,1} & 1 & 0 \\ L_{3,0} & L_{3,1} & 0 & 1 \end{bmatrix}$$

Übrigens ist es für $k > i$ **nicht** die Überlagerung.

In [10]:
```
Li*Lk
```

Out[10]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{1,0} & 1 & 0 & 0 \\ L_{1,0}L_{2,1} + L_{2,0} & L_{2,1} & 1 & 0 \\ L_{1,0}L_{3,1} + L_{3,0} & L_{3,1} & 0 & 1 \end{bmatrix}$$

Aufgabe: Probieren Sie dies mit anderen Werten von N, i, k.

1.2 Permutationsmatrizen

Wir schauen kurz auf Permutationsmatrizen. Wir erzeugen zwei Permutationen mit numpy.

In [11]:
```
N=4
sigma1=np.random.permutation(N)
sigma2=np.random.permutation(N)
print(sigma1,sigma2)
```

```
[0 3 1 2] [3 1 0 2]
```

Wir erzeugen die zugehörigen Permutationsmatrizen, wie in der Vorlesung definiert.

In [12]:
```
P1=sympy.zeros(N)
P2=sympy.zeros(N)
for i in sigma1:
    P1[i,sigma1[i]]=1
for i in sigma2:
    P2[i,sigma2[i]]=1
P1
```

Out[12]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Permutationsmatrizen sind unitär, d.h. $P^t P = I$.

In [13]: P1.T*P1

Out[13]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Wir testen die Wirkung einer Permutationsmatrix auf eine Matrix.

In [14]: P1*A

Out[14]:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \end{bmatrix}$$

Das bringt die Zeilen in die permutierte Reihenfolge. Nun von rechts mit der Transponierten:

In [15]: A*P1.T

Out[15]:

$$\begin{bmatrix} A_{0,0} & A_{0,3} & A_{0,1} & A_{0,2} \\ A_{1,0} & A_{1,3} & A_{1,1} & A_{1,2} \\ A_{2,0} & A_{2,3} & A_{2,1} & A_{2,2} \\ A_{3,0} & A_{3,3} & A_{3,1} & A_{3,2} \end{bmatrix}$$

Das bringt die Spalten in die permutierte Reihenfolge. Wir testen die Multiplikation von Permutationsmatrizen:

In [16]: P1*P2

Out[16]:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Dies ist wieder eine Permutationsmatrix. Wir erzeugen nun eine Permutationsmatrix, die die Zeilen bis i nicht ändert, und die Elementarmatrix L_i.

```
In [17]: i=0
         sigma3=list(range(0,i+1))+list(np.random.permutation(N-1-i)+1)
         print(sigma3)
         P3=sympy.zeros(N)
         for k in sigma3:
             P3[k,sigma3[k]]=1
         Li=sympy.eye(N)
         Li[i+1:N,i]=L[i+1:N,i]
         P3
```

```
[0, 1, 2, 3]
```

Out[17]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Wir testen die Wirkung von PL_iP^t.

```
In [18]: P3*Li*P3.T
```

Out[18]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{1,0} & 1 & 0 & 0 \\ L_{2,0} & 0 & 1 & 0 \\ L_{3,0} & 0 & 0 & 1 \end{bmatrix}$$

Wie in der Vorlesung: Dies ist wieder eine Elementarmatrix, und die Einträge sind einfach nur die der alten Elementarmatrix, in der Zeilenreihenfolge σ.
Aufgabe: Ändern Sie die Parameter und lassen Sie das Programm mehrmals durchlaufen (permutation erzeugt dann andere Werte).

1.3 Berechnung der LR-Zerlegung (ohne Permutation, Spaltenpivotsuche)

Wir erzeugen zunächst wieder eine symbolische Matrix.

```
In [19]: N=4
         A=Matrix(sympy.MatrixSymbol('A',N,N))
         A
```

Out[19]:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}$$

Wir erzeugen eine allgemeine Frobeniusmatrix zur Spalte i.

```
In [20]: L=sympy.eye(N)
         L1=sympy.MatrixSymbol('L',N,N)
         i=1
         for k in range(0,i):
             for j in range(k+1,N):
                 A[j,k]=0
         for k in range(i+1,N):
             L[k,i]=L1[k,i]
         L,A
```

Out[20]:

$$\left(\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & L_{2,1} & 1 & 0\\ 0 & L_{3,1} & 0 & 1\end{bmatrix}, \quad \begin{bmatrix}A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3}\\ 0 & A_{1,1} & A_{1,2} & A_{1,3}\\ 0 & A_{2,1} & A_{2,2} & A_{2,3}\\ 0 & A_{3,1} & A_{3,2} & A_{3,3}\end{bmatrix}\right)$$

Jetzt wenden wir diese Elementarmatrix auf eine Matrix A an, die in den ersten Spalten bereits Nullen aufweist unterhalb der Hauptdiagonalen. Nach der Diskussion oben erwarten wir: Diese Matrix lässt bis zur i. Zeile alles konstant, und addiert Vielfache der ersten Zeile auf die Zeilen darunter.

```
In [21]: Matrix(L*A)
```

Out[21]:

$$\begin{bmatrix}A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3}\\ 0 & A_{1,1} & A_{1,2} & A_{1,3}\\ 0 & A_{1,1}L_{2,1}+A_{2,1} & A_{1,2}L_{2,1}+A_{2,2} & A_{1,3}L_{2,1}+A_{2,3}\\ 0 & A_{1,1}L_{3,1}+A_{3,1} & A_{1,2}L_{3,1}+A_{3,2} & A_{1,3}L_{3,1}+A_{3,3}\end{bmatrix}$$

Wir wählen nun die Einträge unterhalb der Hauptdiagonalen wie in der Vorlesung. Wir erwarten: Dadurch werden in der i. Spalte unterhalb der Hauptdiagonalen Nullen erzeugt.

```
In [22]: L[i+1:N,i]=-A[i+1:N,i]/A[i,i]
         L*A
```

Out[22]:

$$\begin{bmatrix}A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3}\\ 0 & A_{1,1} & A_{1,2} & A_{1,3}\\ 0 & 0 & A_{2,2}-\frac{A_{1,2}A_{2,1}}{A_{1,1}} & A_{2,3}-\frac{A_{1,3}A_{2,1}}{A_{1,1}}\\ 0 & 0 & A_{3,2}-\frac{A_{1,2}A_{3,1}}{A_{1,1}} & A_{3,3}-\frac{A_{1,3}A_{3,1}}{A_{1,1}}\end{bmatrix}$$

Dies tun wir nun für die Spalten 0 bis $N-2$. Der Einfachheit halber ziehen wir die Einheitsmatrix aus der Elementarmatrix heraus. Wie in der Vorlesung bilden wir die L-Matrix durch Überlagerung.

```
In [23]: N=3
         A=Matrix(sympy.MatrixSymbol('A',N,N))
         L1=sympy.eye(N)
```

```
for i in range(0,N-1):
    L=sympy.zeros(N)
    L[i+1:N,i]=-A[i+1:N,i]/A[i,i]
    A=A+L*A
    L1=L1-L
sympy.simplify(A)
```

Out[23]:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} \\ 0 & A_{1,1}-\frac{A_{0,1}A_{1,0}}{A_{0,0}} & A_{1,2}-\frac{A_{0,2}A_{1,0}}{A_{0,0}} \\ 0 & 0 & \frac{A_{0,0}A_{1,1}A_{2,2}-A_{0,0}A_{1,2}A_{2,1}-A_{0,1}A_{1,0}A_{2,2}+A_{0,1}A_{1,2}A_{2,0}+A_{0,2}A_{1,0}A_{2,1}-A_{0,2}A_{1,1}A_{2,0}}{A_{0,0}A_{1,1}-A_{0,1}A_{1,0}} \end{bmatrix}$$

Wie erwartet, erhalten wir eine obere rechte Dreiecksmatrix als Ergebnis der Elimination. Wir checken, ob tatsächlich L*R=A.

In [24]:
```
L1*A
```

Out[24]:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} \\ A_{1,0} & A_{1,1} & A_{1,2} \\ A_{2,0} & A_{2,1} & A_{2,2} \end{bmatrix}$$

Symbolisch klappt das schon mal. Dies wiederholen wir nun numerisch:

In [25]:
```
def lr(A):
    R=A
    L=np.eye(N)
    for i in range(0,N-1):
        L0=np.zeros([N,N])
        L0[i+1:N,i]=-R[i+1:N,i]/R[i,i]
        R=R+np.matmul(L0,R)
        L=L-L0
    return (L,R)
```

Kurzer Test, wir berechnen die LR-Zerlegung einer zufälligen Matrix und testen ob

$$|| A - LR || \sim 0.$$

In [26]:
```
N=4
A=np.random.uniform(0,1,(N,N))
(L,R)=lr(A)
print(np.linalg.norm(np.matmul(L,R)-A))
```

```
1.7554167342883506e-16
```

Abschließend testen wir noch, ob die Formen von L und R korrekt sind.

```
In [27]: print(L)
         print(R)
```

```
[[1.         0.         0.         0.        ]
 [1.53135538 1.         0.         0.        ]
 [0.22479153 1.65556075 1.         0.        ]
 [0.90027458 1.66121788 1.33929371 1.        ]]
[[ 5.71399181e-01  1.34150317e-01  4.53618776e-01  4.11046685e-02]
 [-1.11022302e-16  2.69251866e-01 -2.34898289e-01  3.54745747e-01]
 [ 1.83804166e-16  0.00000000e+00  5.02830353e-01 -8.97815237e-02]
 [-6.17355298e-17  0.00000000e+00  0.00000000e+00 -3.66524933e-01]]
```

Aufgabe: Wiederholen Sie dies mit Matrizen anderer Größe und testen Sie, ob es auch dann funktioniert.