lektion11

January 7, 2026

Table of Contents

1 Lektion 11

Lineare Algebra Teil 2

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
```

```
init_printing()
x, y, z = symbols('x y z')
#%matplotlib notebook
%matplotlib inline
```

1.1 Wiederholung und Ergänzung

[2]:
```
v = Matrix([1 ,-1, 1])
v
```

[2]:
$$\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

[3]:
```
A = Matrix(3, 3, range(9))
A
```

[3]:
$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

[4]:
```
A*v, A@v # Matrix-Matrixprodukt (Vektoren sind Matrizen)
```

[4]:
$$\left(\begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} \right)$$

[5]:
```
v*v.T
```

[5]:
$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

1.1.1 Matrixoperationen

Addition und Skalarmultiplikation

[6]:
```
v + I * A[:,0]
```

[6]:
$$\begin{bmatrix} 1 \\ -1 + 3i \\ 1 + 6i \end{bmatrix}$$

[7]:
```
a = symbols('a')
a*v
```

[7]:
$$\begin{bmatrix} a \\ -a \\ a \end{bmatrix}$$

[8]:
```
A -  2*(v*v.T)
```

[8]:
$$\begin{bmatrix} -2 & 3 & 0 \\ 5 & 2 & 7 \\ 4 & 9 & 6 \end{bmatrix}$$

[9]:
```
matrix_multiply_elementwise(v*v.T, A) # Hadamard- oder Schurprodukt oder elementweises Produkt
                                      # (entspricht * in numpy)
```

[9]:
$$\begin{bmatrix} 0 & -1 & 2 \\ -3 & 4 & -5 \\ 6 & -7 & 8 \end{bmatrix}$$

[10]:
```
x = symbols('x0:10')
x #Tupel mit Symbolen
```

[10]:
$$(x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7,\ x_8,\ x_9)$$

[11]:
```
x[7]
```

[11]:
$$x_7$$

[12]:
```
M = Matrix(2, 5, x)
M
```

[12]:
$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 & x_9 \end{bmatrix}$$

Das ist nicht schön.

[13]:
```
a = symbols('a0:2_0:5')
A = Matrix(2, 5, a)
```

[14]:
```
A
```

[14]:
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \end{bmatrix}$$

1.2 Eigenwerte, Eigenvektoren

[15]:
```
A = Matrix(3, 3, lambda i, j : abs(i-j))
A[-1,0] = 0 # letzte Zeile, erste Spalte
A
```

[15]:
$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

[16]:
```
A.eigenvals() # Eigenwerte mit ihrer algebraischen Vielfachheit
```

[16]:
$$\left\{\left(-\frac{1}{2}-\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{57}}{9}+1}+\frac{2}{3\left(-\frac{1}{2}-\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{57}}{9}+1}}:1,\ \frac{2}{3\left(-\frac{1}{2}+\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{57}}{9}+1}}+\left(-\frac{1}{2}+\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{57}}{9}+1}\right.$$

[17]:
```
A.eigenvals(simplify=True)
```

[17]:
$$\left\{\frac{\sqrt[3]{3}\left(2\sqrt[3]{3}+\left(\sqrt{57}+9\right)^{\frac{2}{3}}\right)}{3\sqrt[3]{\sqrt{57}+9}}:1,\ \frac{\sqrt[3]{3}\left(-8\sqrt[3]{3}-\left(1-\sqrt{3}i\right)^{2}\left(\sqrt{57}+9\right)^{\frac{2}{3}}\right)}{6\left(1-\sqrt{3}i\right)\sqrt[3]{\sqrt{57}+9}}:1,\ \frac{\sqrt[3]{3}\left(-8\sqrt[3]{3}-\left(1+\sqrt{3}i\right)^{2}\left(\sqrt{57}+9\right)^{\frac{2}{3}}\right.}{6\left(1+\sqrt{3}i\right)\sqrt[3]{\sqrt{57}+9}}\right.$$

[18]:
```
A[-1, 0] = 1 # etwas übersichtlicher
A.eigenvals()
```

[18]:
$$\left\{-1:1,\ \frac{1}{2}-\frac{\sqrt{13}}{2}:1,\ \frac{1}{2}+\frac{\sqrt{13}}{2}:1\right\}$$

[19]:
```
evs = A.eigenvects()
evs # Tripel: Eigenwert, algebraische Vielfachheit, Basis des Eigenraums
```

[19]:
$$\left[\left(-1,\ 1,\ \left[\begin{bmatrix}-1\\1\\0\end{bmatrix}\right]\right),\ \left(\frac{1}{2}-\frac{\sqrt{13}}{2},\ 1,\ \left[\begin{bmatrix}-\frac{\sqrt{13}}{2}-\frac{1}{2}\\1\\1\end{bmatrix}\right]\right),\ \left(\frac{1}{2}+\frac{\sqrt{13}}{2},\ 1,\ \left[\begin{bmatrix}-\frac{1}{2}+\frac{\sqrt{13}}{2}\\1\\1\end{bmatrix}\right]\right)\right]$$

[20]:
```
evs[0][2][0]
```

[20]:
$$\begin{bmatrix}-1\\1\\0\end{bmatrix}$$

[21]:
```
D = diag(*[ev_[0] for ev_ in evs])
T = Matrix([[ev_[2][0] for ev_ in evs]])
D, T
```

[21]:
$$\left(\begin{bmatrix}-1 & 0 & 0\\0 & \frac{1}{2}-\frac{\sqrt{13}}{2} & 0\\0 & 0 & \frac{1}{2}+\frac{\sqrt{13}}{2}\end{bmatrix},\ \begin{bmatrix}-1 & -\frac{\sqrt{13}}{2}-\frac{1}{2} & -\frac{1}{2}+\frac{\sqrt{13}}{2}\\1 & 1 & 1\\0 & 1 & 1\end{bmatrix}\right)$$

[22]:
```
simplify(A@T - T@D)
```

[22]:
$$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$$

[23]:
```
# das geht auch kürzer
A.diagonalize()
```

[23]:

$$\left(\begin{bmatrix} -1 & -\frac{\sqrt{13}}{2} - \frac{1}{2} & -\frac{1}{2} + \frac{\sqrt{13}}{2} \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{2} - \frac{\sqrt{13}}{2} & 0 \\ 0 & 0 & \frac{1}{2} + \frac{\sqrt{13}}{2} \end{bmatrix} \right)$$

[24]:
```
A = Matrix(3, 3, [-4, -2, -3, 5, 3, 3, 5, 2, 4])
A
```

[24]:
$$\begin{bmatrix} -4 & -2 & -3 \\ 5 & 3 & 3 \\ 5 & 2 & 4 \end{bmatrix}$$

[25]:
```
A.eigenvals()
```

[25]:
$$\{1 : 3\}$$

[26]:
```
A.eigenvects()
```

[26]:
$$\left[\left(1,\ 3,\ \left[\begin{bmatrix} -\frac{2}{5} \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -\frac{3}{5} \\ 0 \\ 1 \end{bmatrix} \right] \right) \right]$$

[27]:
```
A.left_eigenvects() # linke Eigenvektoren
```

[27]:
$$\left[\left(1,\ 3,\ \left[\begin{bmatrix} 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \right] \right) \right]$$

[28]:
```
A.diagonalize()
```

```
---------------------------------------------------------------------------
MatrixError                               Traceback (most recent call last)
Cell In[28], line 1
----> 1 A.diagonalize()

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/matrices/matrixbase.py:3341, in MatrixBase.diagonalize(self, reals_only␣
 ↪sort, normalize)
   3340 def diagonalize(self, reals_only=False, sort=False, normalize=False):
-> 3341     return _diagonalize(self, reals_only=reals_only, sort=sort,
   3342                         normalize=normalize)

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/matrices/eigen.py:700, in _diagonalize(M, reals_only, sort, normalize)
    696 is_diagonalizable, eigenvecs = _is_diagonalizable_with_eigen(M,
    697             reals_only=reals_only)
    699 if not is_diagonalizable:
--> 700     raise MatrixError("Matrix is not diagonalizable")
    702 if sort:
    703     eigenvecs = sorted(eigenvecs, key=default_sort_key)
```

```
MatrixError: Matrix is not diagonalizable
```

[29]:
```
evs = A.eigenvects()[0]# hier gibt es nur einen Eigenwert
T = Matrix([evs[2]])
D = diag(*([evs[0]] * len(evs[2])))
T, D
```

[29]:
$$\left(\begin{bmatrix} -\frac{2}{5} & -\frac{3}{5} \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

[30]:
```
A @ T - T @ D
```

[30]:
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

1.3 Jordannormalform

[31]:
```
T, J = A.jordan_form()
T, J
```

[31]:
$$\left(\begin{bmatrix} -5 & 1 & -\frac{2}{5} \\ 5 & 0 & 1 \\ 5 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right)$$

[32]:
```
T * J * T.inv() == A
```

[32]:
```
True
```

[33]:
```
A @ T - T @ J
```

[33]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

1.4 Kreuzprodukt

[34]:
```
w = Matrix(1, 3, [1, -2, 1])
v, w
```

[34]:
$$\left(\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}\right)$$

[35]:
```
z = v.cross(w)
z
```

[35]:

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

[36]:
```
w.cross(v)
```

[36]:
$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

1.5 Berechung des Rangs

[37]:
```
M = Matrix(3, 3, range(1, 10))
M
```

[37]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

[38]:
```
M.rank()
```

[38]:
2

[39]:
```
x, y = symbols('x y')
M = Matrix(3, 2, [2*x+2, 2*y-2, 2*x+2, -2*y+2, y-1, x+1])
M
```

[39]:
$$\begin{bmatrix} 2x+2 & 2y-2 \\ 2x+2 & 2-2y \\ y-1 & x+1 \end{bmatrix}$$

[40]:
```
M.rank() # ?
```

[40]:
2

[41]:
```
M1 = M.elementary_row_op('n->kn', row=2, k=2 * x + 2)
# für row=2 ersetze row-te Zeile durch k-fache der row-ten Zeil
M1
```

[41]:
$$\begin{bmatrix} 2x+2 & 2y-2 \\ 2x+2 & 2-2y \\ (2x+2)(y-1) & (x+1)(2x+2) \end{bmatrix}$$

Das darf ich aber nur, wenn $2x + 2 \neq 0$.

[42]:
```
M2 = M1.elementary_row_op('n->n+km', row1=2, row2=0, k=1 - y).expand()
# ersetze row1-te Zeile durch Summe aus row1-ter Zeile und k-fache der row2-ten␣
↪Zeile
M2
```

[42]:
$$\begin{bmatrix} 2x+2 & 2y-2 \\ 2x+2 & 2-2y \\ 0 & 2x^2+4x-2y^2+4y \end{bmatrix}$$

[43]:
```
M3 = M2.elementary_row_op('n->n+km', row1=1, row2=0, k=-1)
M3
```

[43]:
$$\begin{bmatrix} 2x+2 & 2y-2 \\ 0 & 4-4y \\ 0 & 2x^2+4x-2y^2+4y \end{bmatrix}$$

[44]:
```
M4 = M3.elementary_row_op('n->n+km', row1=2, row2=1, k=-M3[2, 1] / M3[1, 1])
M4
```

[44]:
$$\begin{bmatrix} 2x+2 & 2y-2 \\ 0 & 4-4y \\ 0 & 0 \end{bmatrix}$$

Das darf ich nur für $4-4y \neq 0$, weil ich durch diesen Wert geteilt habe.

Bis jetzt gesehen:

Für $x \neq -1$ und $y \neq 1$ ist der Rang gleich 2.

[45]:
```
M.subs(x, -1)
```

[45]:
$$\begin{bmatrix} 0 & 2y-2 \\ 0 & 2-2y \\ y-1 & 0 \end{bmatrix}$$

[46]:
```
M.subs(y, 1)
```

[46]:
$$\begin{bmatrix} 2x+2 & 0 \\ 2x+2 & 0 \\ 0 & x+1 \end{bmatrix}$$

[47]:
```
M.subs({x: -1, y: 1})
```

[47]:
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\text{Rang}(M) = \begin{cases} 0, & x = -1 \wedge y = 1, \\ 2, & \text{in allen anderen Fällen.} \end{cases}$$

1.6 Normen

1.6.1 Vektornormen

[48]:
```
v = Matrix(1, 3, range(1, 4))
v
```

[48]:
$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

[49]:
```
v.norm() # euklidische Norm
```

[49]:
$$\sqrt{14}$$

[50]:
```
v.norm(1)
```

[50]:
$$6$$

[51]:
```
v.norm(oo)
```

[51]:
$$3$$

1.6.2 Matrixnormen

[52]:
```
A
```

[52]:
$$\begin{bmatrix} -4 & -2 & -3 \\ 5 & 3 & 3 \\ 5 & 2 & 4 \end{bmatrix}$$

[53]:
```
A.norm()
```

[53]:
$$3\sqrt{13}$$

Definition Die Frobenius Norm einer Matrix A, A.norm(), ist

$$||A||_F = \sqrt{\sum_{i,j} |a_{i,j}|^2}$$

[54]:
```
sqrt(trace(A * A.H))
```

[54]:
$$3\sqrt{13}$$

Definition Mit $||A||_2$ wird die 2-Norm von A bezeichnet

$$||A||_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

[55]:
```
A.norm(2)
```

[55]:
$$\sqrt{\sqrt{3363} + 58}$$

Man kann zeigen, dass $||A||_2$ die Wurzel des größten Eigenwerts von $A^H A$ ist.

[56]:
```
Max(*(A*A.H).eigenvals().keys())
```

[56]:
$$\sqrt{3363} + 58$$

[57]:
```
A
```

[57]:

$$\begin{bmatrix} -4 & -2 & -3 \\ 5 & 3 & 3 \\ 5 & 2 & 4 \end{bmatrix}$$

[58]:
```
A.norm(1) # Spaltensummennorm
```

[58]:
14

[59]:
```
A.norm(oo) # Zeilensummennorm
```

[59]:
11

1.7 Matrixzerlegungen (nicht besprochen)

Dieser Abschnitt ist nur von Interesse, wenn Sie eine symbolische Implementierung der Algorithmen aus CompLA brauchen. ### LU Zerlegung

vgl. CompLA VL9

Satz: Jede invertierbare $n \times n$ Matrix A besitzt eine Zerlegung der Form

$$PA = LU,$$

mit einer Permutationsmatrix P, einer unteren Dreiecksmatrix L und einer oberen Dreiecksmatrix U.

Eine Zerlegung der Form $A = LR$ existiert, falls die Matrix A ohne Zeilentausch in obere Dreiecksgestallt überführt werden kann, d.h. falls während der Rechnung keine Division durch Null auftritt.

[60]:
```
A = Matrix([[1, -1, 2], [0, 0, -1], [0, 2, -1]])
A
```

[60]:
$$\begin{bmatrix} 1 & -1 & 2 \\ 0 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix}$$

[61]:
```
L, U, perm = A.LUdecomposition() # Zeile 2 und 3 muessen vertauscht werden
L, U, perm
```

[61]:
$$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 & 2 \\ 0 & 2 & -1 \\ 0 & 0 & -1 \end{bmatrix}, [[1, 2]] \right)$$

[62]:
```
P = eye(A.shape[0]).permuteBkwd(perm)
P, L, U
```

[62]:
$$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 & 2 \\ 0 & 2 & -1 \\ 0 & 0 & -1 \end{bmatrix} \right)$$

[63]:
```
P@A == L@U
```

[63]:
```
True
```

[64]:
```
def element(i, j):
    return 1/(i+j+1)
H = Matrix(3, 3, element) # Hilbertmatrix
H
```

[64]:
$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

1.7.1 LDL- und Choleskyzerlegung

vgl. CompLA Übungen

Die LU-Zerlegung einer hermiteschen, positiv definiten Matrix $A \in \mathbb{C}^{n\times n}$ ist ohne Zeilentausch berechenbar.

Satz: Jede hermitesche, positiv definite Matrix $A \in \mathbb{C}^{n\times n}$ besitzt eine Zerlegung der Form

$$A = LDL^T,$$

wobei L eine untere Dreiecksmatrix mit Diagonaleinträgen $l_{i,i} = 1$, $i = 0, \dots, n-1$ und D eine Diagonalmatrix mit positiven Einträgen ist.

Mit

$$D^{1/2} := \begin{pmatrix} \sqrt{d_0} & & & \\ & \sqrt{d_1} & & \\ & & \ddots & \\ & & & \sqrt{d_{n-1}} \end{pmatrix}$$

erhält man eine Zerlegung der Form

$$A = LDL^T = \left(LD^{1/2}\right)\left(D^{1/2}L^T\right) = \tilde{L}\tilde{L}^T$$

mit einer unteren Dreiecksmatrix $\tilde{L}$. Diese Zerlegung nennt man **Cholesky Zerlegung** .

[65]:
```
HC = H.copy()
HC[0,-1] = (1+I)/3
HC[-1,0] = (1-I)/3
HC[-1,-1] = 1
HC
```

[65]:
$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3}+\frac{i}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3}-\frac{i}{3} & \frac{1}{4} & 1 \end{bmatrix}$$

[66]:
```
HC.is_hermitian
```

[66]:
```
True
```

[67]:
```
[simplify(re(l))>0 for l in HC.eigenvals()]
```

[67]: [True, True, True]

[68]:
```
[expand(_) for _ in HC.LDLdecomposition()]
```

[68]: $$\left[\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} - \frac{i}{3} & 1 + 2i & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{12} & 0 \\ 0 & 0 & \frac{13}{36} \end{bmatrix}\right]$$

[69]:
```
HC.cholesky().expand()
```

[69]: $$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 0 \\ \frac{1}{3} - \frac{i}{3} & \frac{\sqrt{3}}{6} + \frac{\sqrt{3}i}{3} & \frac{\sqrt{13}}{6} \end{bmatrix}$$

1.7.2 QR Zerlegung

vgl. CompLA VL 10

Die QR Zerlegung einer $n \times m$ Matrix $ A $ ist eine Zerlegung

$$A = QR,$$

mit einer Matrix Q mit orthonormalen Spalten und einer (verallgemeinerten) oberen Dreiecksmatrix R.

Satz: Jede $n \times m$ Matrix A mit $n \geq m$ und vollem Rang besitzt eine QR Zerlegung.

[70]:
```
Q, R = A.QRdecomposition()
Q, R
```

[70]: $$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & -1 & 2 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}\right)$$

[71]:
```
Q.H@Q, Q@R-A
```

[71]: $$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right)$$

[72]:
```
B = Matrix(4, 3, [3, 1 , 1, 1, 4, -1, 1, -1, 4, 1, 1, 1])
B
```

[72]: $$\begin{bmatrix} 3 & 1 & 1 \\ 1 & 4 & -1 \\ 1 & -1 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

[73]:
```
Q, R = B.QRdecomposition()
Q, R
```

[73]:
$$\left(\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{3\sqrt{537}}{358} & -\frac{45\sqrt{10382}}{10382} \\ \frac{\sqrt{3}}{6} & \frac{41\sqrt{537}}{1074} & \frac{13\sqrt{10382}}{5191} \\ \frac{\sqrt{3}}{6} & -\frac{19\sqrt{537}}{1074} & \frac{42\sqrt{10382}}{5191} \\ \frac{\sqrt{3}}{6} & \frac{5\sqrt{537}}{1074} & \frac{25\sqrt{10382}}{10382} \end{bmatrix}, \begin{bmatrix} 2\sqrt{3} & \frac{7\sqrt{3}}{6} & \frac{7\sqrt{3}}{6} \\ 0 & \frac{\sqrt{537}}{6} & -\frac{121\sqrt{537}}{1074} \\ 0 & 0 & \frac{5\sqrt{10382}}{179} \end{bmatrix}\right)$$

[74]:
```
Q.T@Q, Q@R-B
```

[74]:
$$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right)$$

1.8 Matrixfunktionen

1.8.1 Matrixwurzel

[4]:
```
B = Matrix(3, 3, [4, 1 , 1, 1, 4, -1, 1, -1, 4])
B
```

[4]:
$$\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & -1 \\ 1 & -1 & 4 \end{bmatrix}$$

[76]:
```
sqrtB = B**(S(1)/2)
sqrtB
```

[76]:
$$\begin{bmatrix} \frac{\sqrt{2}}{3} + \frac{2\sqrt{5}}{3} & -\frac{\sqrt{2}}{3} + \frac{\sqrt{5}}{3} & -\frac{\sqrt{2}}{3} + \frac{\sqrt{5}}{3} \\ -\frac{\sqrt{2}}{3} + \frac{\sqrt{5}}{3} & \frac{\sqrt{2}}{3} + \frac{2\sqrt{5}}{3} & -\frac{\sqrt{5}}{3} + \frac{\sqrt{2}}{3} \\ -\frac{\sqrt{2}}{3} + \frac{\sqrt{5}}{3} & -\frac{\sqrt{5}}{3} + \frac{\sqrt{2}}{3} & \frac{\sqrt{2}}{3} + \frac{2\sqrt{5}}{3} \end{bmatrix}$$

[77]:
```
erg = sqrtB @ sqrtB  - B
simplify(erg)
```

[77]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

1.8.2 Matrixexponentialfunktion

Die Exponentialfunktion $z \mapsto e^z$ ist als Reihe definiert

$$e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!}.$$

Setzt man hier formal für z eine Matrix ein, so erhält man die Matrixexponentilafunktion (vgl. Ana II).

[78]:
```
exp(B)
```

[78]:

$$\begin{bmatrix} \frac{e^2}{3} + \frac{2e^5}{3} & -\frac{e^2}{3} + \frac{e^5}{3} & -\frac{e^2}{3} + \frac{e^5}{3} \\ -\frac{e^2}{3} + \frac{e^5}{3} & \frac{e^2}{3} + \frac{2e^5}{3} & -\frac{e^5}{3} + \frac{e^2}{3} \\ -\frac{e^2}{3} + \frac{e^5}{3} & -\frac{e^5}{3} + \frac{e^2}{3} & \frac{e^2}{3} + \frac{2e^5}{3} \end{bmatrix}$$

$t \mapsto \exp(tB)$ löst die Differentialgleichung

$$\frac{d}{dt}y(t) = By(t)$$

[5]:
```
t = symbols('t')
diff(exp(t*B), t) - B*exp(t*B)
```

[5]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

exp erfülltÖ $\exp(B)\exp(-B) = I$

[80]:
```
exp(t*B) * exp(-t*B)
```

[80]:
$$\begin{bmatrix} 2\left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(\frac{e^{5t}}{3} - \frac{e^{2t}}{3}\right) + \left(\frac{e^{-2t}}{3} + \frac{2e^{-5t}}{3}\right)\left(\frac{2e^{5t}}{3} + \frac{e^{2t}}{3}\right) & \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(\frac{2e^{5t}}{3} + \\ \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(-\frac{e^{5t}}{3} + \frac{e^{2t}}{3}\right) + \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(\frac{2e^{5t}}{3} + \frac{e^{2t}}{3}\right) + \left(\frac{e^{-2t}}{3} + \frac{2e^{-5t}}{3}\right)\left(\frac{e^{5t}}{3} - \frac{e^{2t}}{3}\right) & \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(\frac{e^{5t}}{3} - \frac{e^{2t}}{3}\right) \\ \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(-\frac{e^{5t}}{3} + \frac{e^{2t}}{3}\right) + \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(\frac{2e^{5t}}{3} + \frac{e^{2t}}{3}\right) + \left(\frac{e^{-2t}}{3} + \frac{2e^{-5t}}{3}\right)\left(\frac{e^{5t}}{3} - \frac{e^{2t}}{3}\right) & \left(-\frac{e^{-2t}}{3} + \frac{e^{-5t}}{3}\right)\left(\frac{e^{5t}}{3} - \frac{e^{2t}}{3}\right) \end{bmatrix}$$

[81]:
```
B.applyfunc(exp) # eintragsweise Anwendung der exp Funktion
```

[81]:
$$\begin{bmatrix} e^4 & e & e \\ e & e^4 & e^{-1} \\ e & e^{-1} & e^4 \end{bmatrix}$$

1.9 Gradient

[82]:
```
x, y, z = symbols('x y z')
var = (x, y, z)
```

[83]:
```
f = exp(x**2 + y**2 + z**2)
f
```

[83]:
$$e^{x^2+y^2+z^2}$$

[84]:
```
gr = Matrix([[f.diff(t) for t in var]])
gr
```

[84]:
$$\begin{bmatrix} 2xe^{x^2+y^2+z^2} & 2ye^{x^2+y^2+z^2} & 2ze^{x^2+y^2+z^2} \end{bmatrix}$$

[85]:
```
J = Matrix([f]).jacobian(var)
J
```

[85]:
$$\begin{bmatrix} 2xe^{x^2+y^2+z^2} & 2ye^{x^2+y^2+z^2} & 2ze^{x^2+y^2+z^2} \end{bmatrix}$$

1.10 Hessematrix

[86]:
```
H = J.jacobian(var)
H
```

[86]:
$$\begin{bmatrix} 4x^2e^{x^2+y^2+z^2} + 2e^{x^2+y^2+z^2} & 4xye^{x^2+y^2+z^2} & 4xze^{x^2+y^2+z^2} \\ 4xye^{x^2+y^2+z^2} & 4y^2e^{x^2+y^2+z^2} + 2e^{x^2+y^2+z^2} & 4yze^{x^2+y^2+z^2} \\ 4xze^{x^2+y^2+z^2} & 4yze^{x^2+y^2+z^2} & 4z^2e^{x^2+y^2+z^2} + 2e^{x^2+y^2+z^2} \end{bmatrix}$$

[87]:
```
hessian(f, var)
```

[87]:
$$\begin{bmatrix} 4x^2e^{x^2+y^2+z^2} + 2e^{x^2+y^2+z^2} & 4xye^{x^2+y^2+z^2} & 4xze^{x^2+y^2+z^2} \\ 4xye^{x^2+y^2+z^2} & 4y^2e^{x^2+y^2+z^2} + 2e^{x^2+y^2+z^2} & 4yze^{x^2+y^2+z^2} \\ 4xze^{x^2+y^2+z^2} & 4yze^{x^2+y^2+z^2} & 4z^2e^{x^2+y^2+z^2} + 2e^{x^2+y^2+z^2} \end{bmatrix}$$

[88]:
```
hessian(f, var) - H
```

[88]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

[89]:
```
f = Function('f')
f(*var)
```

[89]:
$$f(x, y, z)$$

[90]:
```
hessian(f(x, y, z), var)
```

[90]:
$$\begin{bmatrix} \frac{\partial^2}{\partial x^2} f(x,y,z) & \frac{\partial^2}{\partial y \partial x} f(x,y,z) & \frac{\partial^2}{\partial z \partial x} f(x,y,z) \\ \frac{\partial^2}{\partial y \partial x} f(x,y,z) & \frac{\partial^2}{\partial y^2} f(x,y,z) & \frac{\partial^2}{\partial z \partial y} f(x,y,z) \\ \frac{\partial^2}{\partial z \partial x} f(x,y,z) & \frac{\partial^2}{\partial z \partial y} f(x,y,z) & \frac{\partial^2}{\partial z^2} f(x,y,z) \end{bmatrix}$$

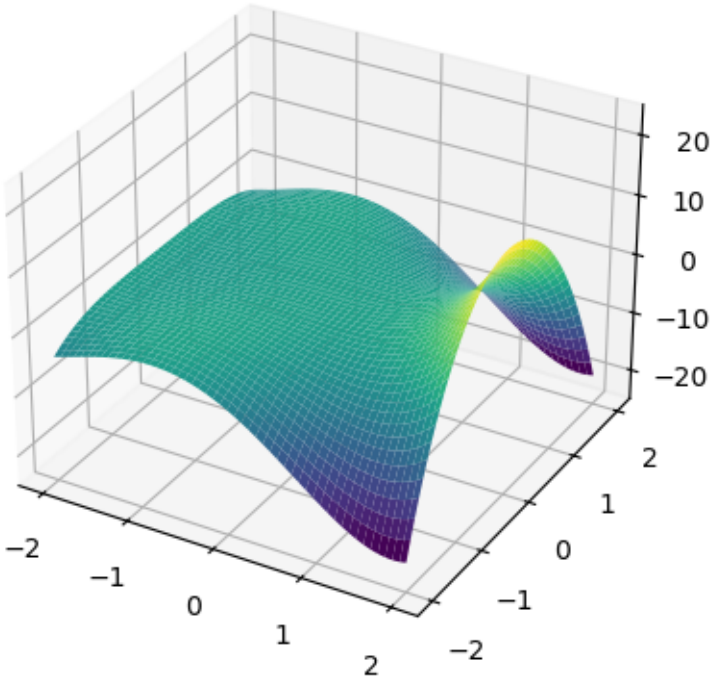
[91]:
```
f = x**4 / 2 - x**2 * y**2 - y**4 / 4 + x**3 - 2 * x * y**2
f.factor()
```

[91]:
$$\frac{2x^4 + 4x^3 - 4x^2y^2 - 8xy^2 - y^4}{4}$$

[124]:
```
fn = lambdify((x, y), f)
```

[125]:
```
gn = np.linspace(-2, 2, 500)
X, Y = np.meshgrid(gn ,gn)
F = fn(X, Y)
```

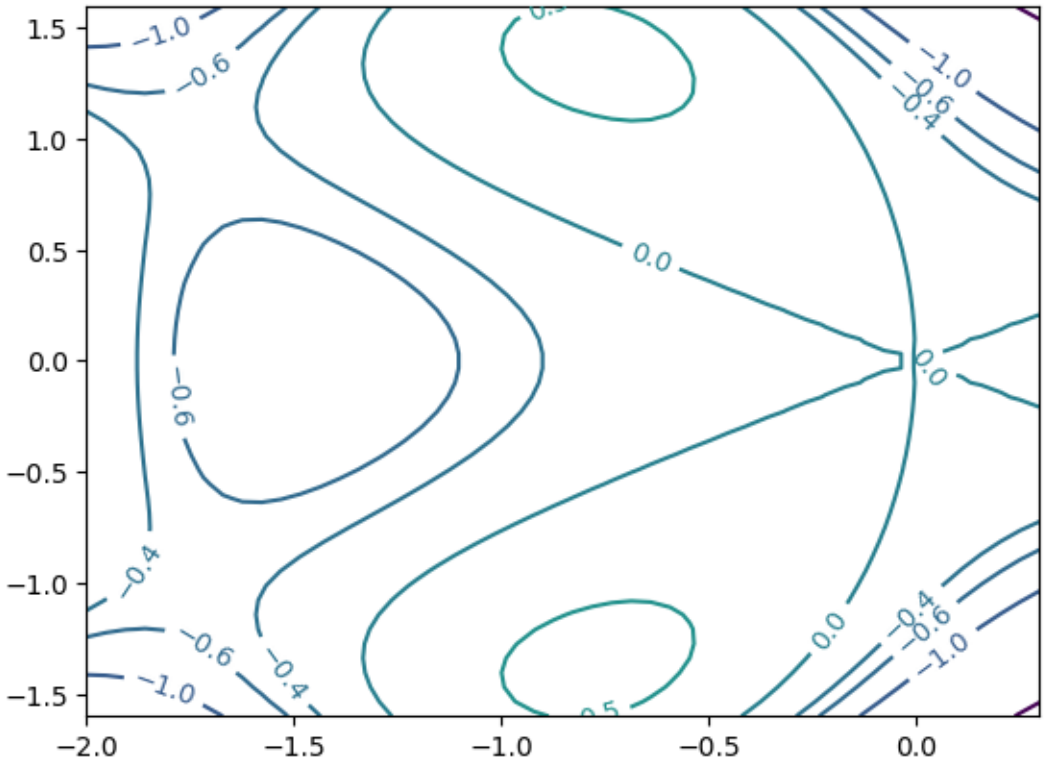
[126]:
```
fig = plt.figure(6)
fig.clf()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, F, lw=1, cmap=plt.cm.viridis);
ax.set_zlim(-25, 25);
```



[127]:
```
ax.contour(X, Y, F, [-10, -1, 0, 1, 10]exp( ) , offset=-25, cmap=plt.cm.
↪viridis);
```

[120]:
```
fig = plt.figure(7)
ax = fig.gca()
cs = ax.contour(X, Y, F, [-3, -1, -.6, -.4, 0, .5, 1, 4], cmap=plt.cm.viridis)
plt.clabel(cs)
```

[120]:
```
<a list of 18 text.Text objects>
```



[97]:
```
ax.contour?
```

1.11 Extrema und Sattelpunkte

[98]:
```
ext = solve(Matrix([f]).jacobian((x, y)))
```

[99]:
```
ext # Liste von Lösungen jeweils als Dictionary
```

[99]:
$$\left[\left\{x : -\frac{3}{2},\ y : 0\right\},\ \{x : 0,\ y : 0\},\ \left\{x : -\frac{5}{4} - \frac{\sqrt{33}}{12},\ y : -\sqrt{\frac{17}{12} - \frac{\sqrt{33}}{12}}\right\},\ \left\{x : -\frac{5}{4} - \frac{\sqrt{33}}{12},\ y : \sqrt{\frac{17}{12} - \frac{\sqrt{33}}{12}}\right\},\ \{\right.$$

[100]:
```
H = hessian(f, (x, y))
H
```

[100]:
$$\begin{bmatrix} 6x^2 + 6x - 2y^2 & -4xy - 4y \\ -4xy - 4y & -2x^2 - 4x - 3y^2 \end{bmatrix}$$

Hurwitz Kriterium

Eine Matrix A ist genau dann positiv definit, falls alle Hauptminoren von A positiv sind.

Eine Matrix A ist genau dann negativ definit, falls $-A$ positiv definit ist.

[101]:
```
H0 = H.subs(ext[0])
H0, (H0.det() > 0) & (H0[0,0] > 0)
```

[101]:
$$\left(\begin{bmatrix} \frac{9}{2} & 0 \\ 0 & \frac{3}{2} \end{bmatrix}, \text{ True}\right)$$

[265]:
```
for ex in ext:
    H0 = H.subs(ex)
    stri = f' bei x={ex[x]} \t y={ex[y]}'
    if (H0.det() > 0) & (H0[0, 0] > 0): # H ist hier positiv definit
        print('Minimum \t' + stri)
    elif ((-H0).det() > 0) & (-H0[0, 0] > 0): # -H ist hier positiv definit
        print('Maximum \t' + stri)
    elif (H0.det() < 0):  #Spezialfall für reellwertige Funktionen ueber R^2
        print('Sattelpunkt\t' + stri)
    else:
        print('Keine Aussage\t' + stri + '\t mit Hilfe der Hessematrix␣
 ↪moeglich')
```

```
Minimum          bei x=-3/2      y=0
Keine Aussage    bei x=0         y=0     mit Hilfe der Hessematrix moeglich
Sattelpunkt      bei x=-5/4 - sqrt(33)/12        y=-sqrt(17/12 - sqrt(33)/12)
Sattelpunkt      bei x=-5/4 - sqrt(33)/12        y=sqrt(17/12 - sqrt(33)/12)
Maximum          bei x=-5/4 + sqrt(33)/12        y=-sqrt(sqrt(33)/12 + 17/12)
Maximum          bei x=-5/4 + sqrt(33)/12        y=sqrt(sqrt(33)/12 + 17/12)
```

[299]:
```
n = len(ext)
extd = np.empty(n,
                dtype=[('x', float), ('y', float), ('f', float), ('size', int),
                       ('color', str, 7), ('type', str, 7)])  # structured␣
 ↪datatype
for i, ex in enumerate(ext):
    extd[i]['x'] = float(ex[x].n())
    extd[i]['y'] = float(ex[y].n())
    extd[i]['f'] = float(f.subs(ex).n())
    H0 = H.subs(ex)
    if (H0.det() > 0) & (H0[0, 0] > 0):
        extd[i]['color'] = 'orange'; extd[i]['size'] = 49
        extd[i]['type'] = 'Minimum'
    elif ((-H0).det() > 0) & (-H0[0, 0] > 0):
        extd[i]['color'] = 'crimson'; extd[i]['size'] = 50
        extd[i]['type'] = 'Maximum'
    elif (H0.det() < 0):
        extd[i]['color'] = 'lime'; extd[i]['size'] = 51
        extd[i]['type'] = 'Sattel'
    else:
        extd[i]['color'] = 'black'; extd[i]['size'] = 52
```

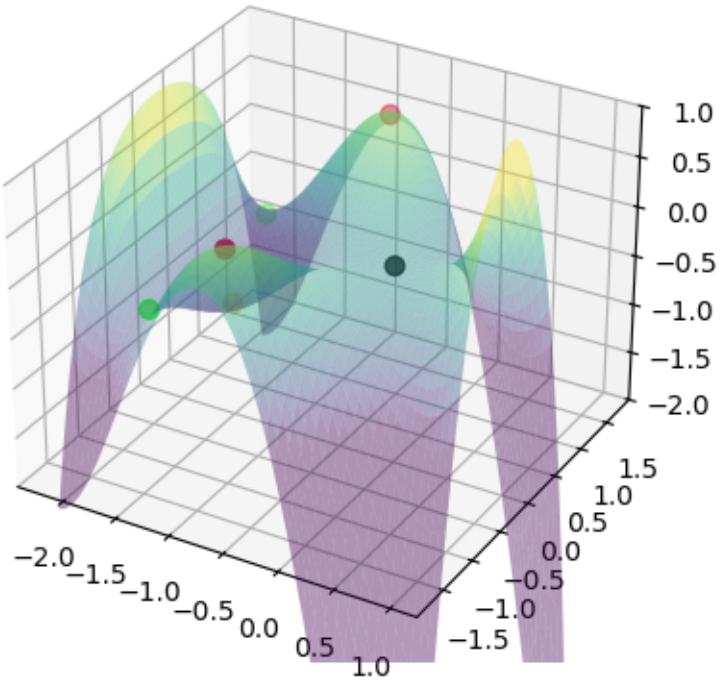
```
        extd[i]['type'] = '?'
```

[300]:
```
extd
```

[300]:
```
array([(-1.5       ,  0.        , -0.84375   , 49, 'orange', 'Minimum'),
       ( 0.        ,  0.        ,  0.        , 52, 'black', '?'),
       (-1.72871355, -0.96847979, -0.48081761, 51, 'lime', 'Sattel'),
       (-1.72871355,  0.96847979, -0.48081761, 51, 'lime', 'Sattel'),
       (-0.77128645, -1.37672809,  0.61623428, 50, 'crimson', 'Maximum'),
       (-0.77128645,  1.37672809,  0.61623428, 50, 'crimson', 'Maximum')],
      dtype=[('x', '<f8'), ('y', '<f8'), ('f', '<f8'), ('size', '<i8'),
('color', '<U7'), ('type', '<U7')])
```

[301]:
```
from matplotlib.colors import Normalize

xn = np.linspace(-2.2, 1)
yn = np.linspace(-1.7, 1.7)
X, Y = np.meshgrid(xn, yn)
F = fn(X, Y)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, F, lw=0, cmap=plt.cm.viridis, alpha=0.4, vmin=-1, vmax=1)
ax.scatter(extd['x'], extd['y'], extd['f'], s=50, c=extd['color'])
ax.set_zlim(-2, 1);
```

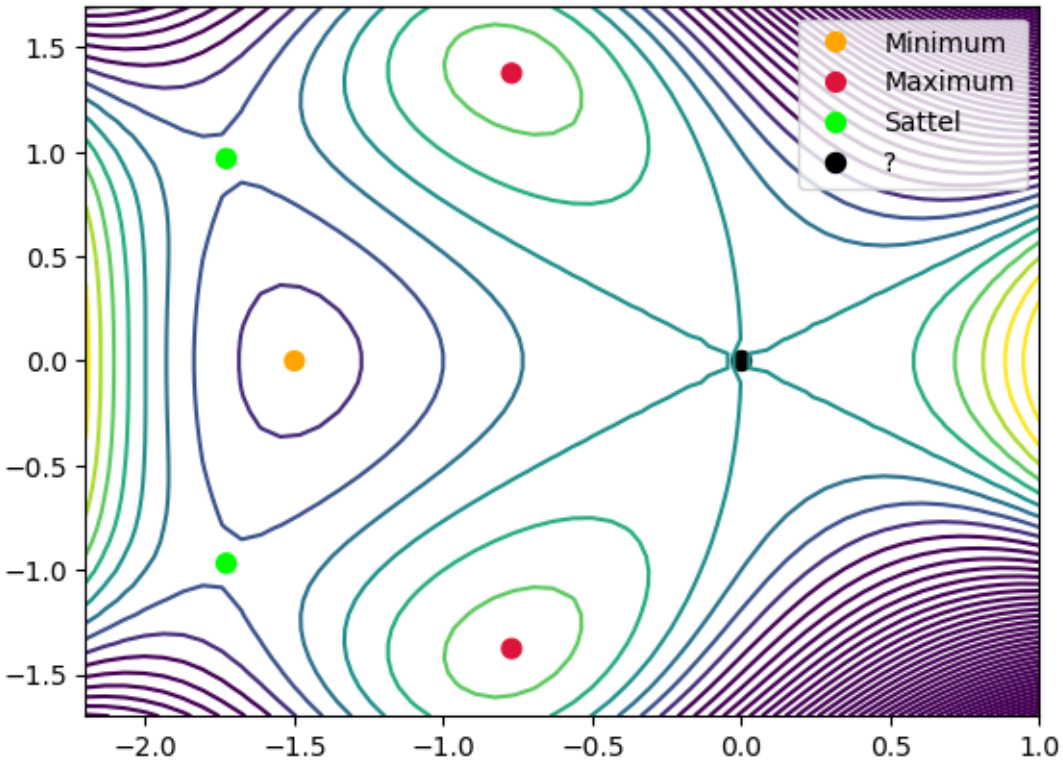


[312]:
```
import matplotlib
fig = plt.figure()
ax = fig.add_subplot(111)
ax.contour(X, Y, F, 50, cmap=plt.cm.viridis, vmin=-1, vmax=1)
scatter = ax.scatter(extd['x'], extd['y'], s=extd['size'], c=extd['color'])

handles, _ = scatter.legend_elements(prop='sizes') # leider können wir nicht␣
 ↪auf die 'colors' zugreifen

# Trick um Legende zu beschriften
sizes, index = np.unique(extd['size'], return_index=True)
for i, ind in enumerate(index):
    handles[i].set_color(extd['color'][ind])

ax.legend(handles, extd['type'][index]);
```



[ ]: