lektion9

November 28, 2024

1 numpy.linalg: Lineare Algebra mit NumPy

Viele grundlegende Methoden aus der linearen Algebra müssen wir nicht immer wieder mühsam selbst implementieren, sondern wir können auf Funktionen aus dem Paket numpy.linalg zurückgreifen. Hier ein paar nützliche Beispiele:

[1]:
```
import numpy as np
np.set_printoptions(legacy = '1.25')
```

[2]:
```
A = np.array([[1, 1, 1],
              [1, 2, 2],
              [1, 2, 3]])

b = np.array([[1],
              [2],
              [3]])
```

1.1 Rang einer Matrix

[3]:
```
rk = np.linalg.matrix_rank(A)
rk
```

[3]:
```
3
```

1.2 Determinante einer Matrix

[4]:
```
detA = np.linalg.det(A)
detA
```

[4]:
```
1.0
```

1.3 Norm eines Vektors

Berechnung der Euklidischen Norm eines Vektors $x \in \mathbb{C}^n$:

$$\|x\| = \sqrt{\sum_{i=1}^{n} |x_i|^2} \tag{1}$$

[5]:
```
# per "Hand"
n1 = np.sqrt(sum(b * b))
n1[0]
```

[5]:
```
3.7416573867739413
```

[6]:
```
# mit numpy.linalg
n2 = np.linalg.norm(b)
n2
```

[6]:
```
3.7416573867739413
```

1.4 Inverse einer Matrix

[7]:
```
Ainv = np.linalg.inv(A)
Ainv
```

[7]:
```
array([[ 2., -1.,  0.],
       [-1.,  2., -1.],
       [ 0., -1.,  1.]])
```

[8]:
```
# Probe:
A @ Ainv
```

[8]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[9]:
```
np.linalg.norm(A @ Ainv - np.eye(3))
```

[9]:
```
0.0
```

1.5 Eigenwerte und Eigenvektoren

[10]:
```
ew, ev = np.linalg.eig(A)
```

[11]:
```
ew # i-ter Eigenwert an i-ter Stelle
```

[11]:
```
array([5.04891734, 0.64310413, 0.30797853])
```

[12]:
```
ev # Eigenvektor zum i-ten Eigenwert in i-ter Spalte
```

[12]:
```
array([[-0.32798528, -0.73697623,  0.59100905],
       [-0.59100905, -0.32798528, -0.73697623],
       [-0.73697623,  0.59100905,  0.32798528]])
```

Sind die Eigenvektoren schon normiert?

[13]:
```
[np.linalg.norm(ev[:, j]) for j in range(ev.shape[0])]
```

[13]:
```
[1.0, 1.0, 1.0]
```

Wird die Eigenwert-Gleichung $Av = \lambda v$ erfüllt?

[14]:
```
# Probe für den ersten Eigenvektor:
A @ ev[:, 0] - ew[0] * ev[:, 0]
```

[14]:
```
array([-3.10862447e-15, -5.32907052e-15, -4.88498131e-15])
```

[15]:
```
np.linalg.norm(A @ ev[:, 0] - ew[0] * ev[:, 0])
```

[15]:
```
7.869280847691828e-15
```

[16]:
```
# Probe für alle Eigenwerte und Eigenvektoren gleichzeitig:
np.linalg.norm(A @ ev - ev @ np.diag(ew))
```

[16]:
```
7.931650484759927e-15
```

[17]:
```
# Orthogonalität von ev
np.linalg.norm(ev @ ev.T - np.eye(3))
```

[17]:
```
7.076311083754595e-16
```

[18]:
```
# da alle Eigenwerte verschieden:
# Spektralzerlegung A = U D U^T
np.linalg.norm(A - ev @ np.diag(ew) @ ev.T)
```

[18]:
```
8.030527940416976e-15
```

Wenn man nur die Eigenwerte benötigt, kann man auch numpy.linalg.eigvals() benutzen. Für symmetrisch bzw. hermitesche Matrizen gibt es die Funktion numpy.linalg.eigh().

1.6 Lösung von $Ax = b$

Für gegebenen Vektor b und invertierbare Matrix A passender Dimensionen wird ein Vektor x gesucht, sodass $Ax = b$ gilt.

[19]:
```
# Variante 1 ("Hau drauf"-Methode, NICHT empfohlen!)
x = np.linalg.inv(A) @ b
x
```

[19]:
```
array([[0.],
       [0.],
       [1.]])
```

[20]:
```
# Variante 2 (empfohlen)
x = np.linalg.solve(A, b)
x
```

[20]:
```
array([[0.],
       [0.],
       [1.]])
```

[21]:
```
# Probe:
np.linalg.norm(A @ x - b)
```

[21]:
```
0.0
```

1.7 Sonstiges

[22]:
```
# Manche mögen es auch numpy.linalg einen eigenen Namen zu geben:
import numpy.linalg as la
```

[23]:
```
# Dann werden obige Befehle alle etwas kürzer, z.B.
la.det(A)
```

[23]:
```
1.0
```

[24]:
```
# Alternative Überprüfung, ob z.B. Eigenwertgleichung erfüllt ist:
A @ ev[:, 0] == ew[0] * ev[:, 0] # hier nicht zufriedenstellend
```

[24]:
```
array([False, False, False])
```

[25]:
```
np.allclose(A @ ev[:, 0], ew[0] * ev[:, 0]) # hier gute Alternative
```

[25]:
```
True
```

[26]:
```
# Probleme mit ganzzahligen Matrizen
A
```

[26]:
```
array([[1, 1, 1],
       [1, 2, 2],
       [1, 2, 3]])
```

[27]:
```
A[1, 0] = 2.2
```

[28]:
```
A # Eintrag ist 1 statt 1.2 (es wurde also gerundet)
```

[28]:
```
array([[1, 1, 1],
       [2, 2, 2],
       [1, 2, 3]])
```

[29]:
```
# Variante 1
A = A.astype('float')
A
```

[29]:
```
array([[1., 1., 1.],
       [2., 2., 2.],
       [1., 2., 3.]])
```

[30]:
```
A[1, 0] = 2.2
A
```

[30]:
```
array([[1. , 1. , 1. ],
       [2.2, 2. , 2. ],
       [1. , 2. , 3. ]])
```

[31]:
```
# besser:
A = np.array([[1, 1, 1],
              [1, 2, 2],
              [1, 2, 3]], dtype = 'float')
A
```

[31]:
```
array([[1., 1., 1.],
       [1., 2., 2.],
       [1., 2., 3.]])
```

[32]:
```
A[1, 0] = 2.2
A
```

[32]:
```
array([[1. , 1. , 1. ],
       [2.2, 2. , 2. ],
       [1. , 2. , 3. ]])
```

[ ]:
```

```