lektion10

December 17, 2025

Inhalt

1 Lektion 10

1.1 Vektoren

[1]:
```
from sympy import *
init_printing()
```

Es gibt **keine** Vektoren nur $1 \times n$ und $n \times 1$ Matrizen.

[2]:
```
v = Matrix([0 , 1+I, 1-I])
v
```

[2]:
$$\begin{bmatrix} 0 \\ 1+i \\ 1-i \end{bmatrix}$$

[3]:
```
v = Matrix(3, 1, [0, 1+I, 1-I])
v
```

[3]:
$$\begin{bmatrix} 0 \\ 1+i \\ 1-i \end{bmatrix}$$

[4]:
```
u = Matrix(1, 3, [3*I, 1+I, 1])
u
```

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

[5]:
```
display(u)
display(v)
```

$$\begin{bmatrix} 3i & 1+i & 1 \end{bmatrix}$$

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

[6]:
```
v*u
```

[6]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 3i\,(1+i) & (1+i)^2 & 1+i \\ 3i\,(1-i) & (1-i)\,(1+i) & 1-i \end{bmatrix}$$

[7]:
```
u*v # Achtung das ist eine 1x1 Matrix
```

[7]:
$$\begin{bmatrix} 1-i+(1+i)^2 \end{bmatrix}$$

[8]:
```
simplify((u*v)) # simplify wird eintragsweise angewandt, das gilt nicht für␣
↪alle Funktionen
```

[8]:

$$\begin{bmatrix}1 + i\end{bmatrix}$$

[9]:
```
xs = symbols('x:3')
xs
```

[9]:
$$(x_0,\ x_1,\ x_2)$$

[10]:
```
x = Matrix(xs)
x
```

[10]:
$$\begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix}$$

1.1.1 transponieren und (transponieren und komplex konjugieren)

[11]:
```
x.T*x # .T transponieren
```

[11]:
$$\begin{bmatrix}x_0^2 + x_1^2 + x_2^2\end{bmatrix}$$

[12]:
```
x.transpose()*x
```

[12]:
$$\begin{bmatrix}x_0^2 + x_1^2 + x_2^2\end{bmatrix}$$

[13]:
```
simplify((v.T*v)[0])
```

[13]:
$$0$$

[14]:
```
v.conjugate(), v.C # .C konjugieren
```

[14]:
$$\left(\begin{bmatrix}0\\1 - i\\1 + i\end{bmatrix},\ \begin{bmatrix}0\\1 - i\\1 + i\end{bmatrix}\right)$$

[15]:
```
v.H # .H transponieren und komplex konjugieren
```

[15]:
$$\begin{bmatrix}0 & 1 - i & 1 + i\end{bmatrix}$$

[16]:
```
simplify((v.H*v)[0])
```

[16]:
$$4$$

1.1.2 Skalarprodukt

[17]:
```
u, v
```

[17]:
$$\left(\begin{bmatrix}3i & 1 + i & 1\end{bmatrix},\ \begin{bmatrix}0\\1 + i\\1 - i\end{bmatrix}\right)$$

[18]:
```
u.dot(v, hermitian=True).simplify()  # Skalarprodukt .dot gibt einen Ausdruck␣
↪zurueck
```

[18]:
$$3 - i$$

[20]:
```
simplify((u.conjugate() * v))  # Skalarprodukt
```

[20]:
$$\left[3 - i\right]$$

[22]:
```
simplify(u.C * v)  # Skalarprodukt
```

[22]:
$$\left[3 - i\right]$$

[23]:
```
u.dot(v, hermitian=True, conjugate_convention='physics').simplify() #␣
↪Skalarprodukt, wobei v konjugiert wird
```

[23]:
$$3 + i$$

[25]:
```
simplify(u * v.C)
```

[25]:
$$\left[3 + i\right]$$

[26]:
```
u.dot(v).simplify() # das ist kein Skalarprodukt für komplexe Vektorräume
```

[26]:
$$1 + i$$

[27]:
```
simplify((u*v))
```

[27]:
$$\left[1 + i\right]$$

1.2 Matrizen

[28]:
```
A = Matrix([[1, 2], [3, 4]])
A
```

[28]:
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

[29]:
```
A = Matrix(3, 3, range(1, 10))
A
```

[29]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

[30]:
```
A[1, 2]
```

[30]:
$$6$$

[31]:
```
A.shape
```

[31]: $(3,\ 3)$

1.2.1 Slicing von Matrizen

[32]:
```
A[:, 0] # erste Spalte
```

[32]: $\begin{bmatrix}1\\4\\7\end{bmatrix}$

[33]:
```
A.col(0)
```

[33]: $\begin{bmatrix}1\\4\\7\end{bmatrix}$

[34]:
```
# vier Arten die letzte Zeile in der 3x3 Matrix A zu erhalten
A[A.shape[1] - 1, :], A[-1, :], A.row(A.shape[1] - 1), A.row(-1)
```

[34]: $\left(\begin{bmatrix}7 & 8 & 9\end{bmatrix}, \begin{bmatrix}7 & 8 & 9\end{bmatrix}, \begin{bmatrix}7 & 8 & 9\end{bmatrix}, \begin{bmatrix}7 & 8 & 9\end{bmatrix}\right)$

[35]:
```
A[0:2, 1:3]
```

[35]: $\begin{bmatrix}2 & 3\\5 & 6\end{bmatrix}$

Beim Entfernen 'delete' von Spalten oder Zeilen wird die Matrix **in place** modifiziert

[36]:
```
A.col_del(0)
A
```

[36]: $\begin{bmatrix}2 & 3\\5 & 6\\8 & 9\end{bmatrix}$

[37]:
```
A.row_del(-1)
A
```

[37]: $\begin{bmatrix}2 & 3\\5 & 6\end{bmatrix}$

Das Hinzufügen 'insert' von Spalten oder Zeilen erfolgt **nicht** in place.

[38]:
```
A = A.row_insert(1, Matrix([[3, 4]]))
A
```

[38]: $\begin{bmatrix}2 & 3\\3 & 4\\5 & 6\end{bmatrix}$

[39]:
```
A = A.col_insert(-1, Matrix([1, 6, 1])) # unerwartet
A
```

[39]:
$$\begin{bmatrix} 2 & 1 & 3 \\ 3 & 6 & 4 \\ 5 & 1 & 6 \end{bmatrix}$$

[40]:
```
A.col_del(1)
```

[41]:
```
A = A.row_join(Matrix(3, 1, [1, 6, 1]))
A
```

[41]:
$$\begin{bmatrix} 2 & 3 & 1 \\ 3 & 4 & 6 \\ 5 & 6 & 1 \end{bmatrix}$$

1.2.2 Werte eintragen

[42]:
```
A[0, 0] = 10
```

[43]:
```
A
```

[43]:
$$\begin{bmatrix} 10 & 3 & 1 \\ 3 & 4 & 6 \\ 5 & 6 & 1 \end{bmatrix}$$

[44]:
```
A[0, :] = u
A
```

[44]:
$$\begin{bmatrix} 3i & 1+i & 1 \\ 3 & 4 & 6 \\ 5 & 6 & 1 \end{bmatrix}$$

1.2.3 Numpy Arrays

Achtung !

Numpy Arrays haben Einträge von einem bestimmten Datentyp. Der Datentyp kann nach der Erstellung des Arrays nicht geändert werden.

[45]:
```
import numpy as np

An = np.array([[1, 2, 3], [4, 5, 6], [6, 7, 8]])
An
```

[45]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [6, 7, 8]])
```

[46]:
```
An = np.array(np.arange(1, 10)).reshape(3, 3)
An
```

[46]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[47]:
```
An[0, 0] = 1.4
An
```

[47]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[48]:
```
An = An.astype(float) # astype erzeugt eine Kopie
An
```

[48]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

[49]:
```
An[0, 0] = 1.4
An
```

[49]:
```
array([[1.4, 2. , 3. ],
       [4. , 5. , 6. ],
       [7. , 8. , 9. ]])
```

[50]:
```
An[1, 1] = 1 + 1j
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[50], line 1
----> 1 An[1, 1] = 1 + 1j

TypeError: float() argument must be a string or a real number, not 'complex'
```

[51]:
```
Ac = An.astype(complex)
Ac[1, 1] = 1 + 1j
Ac
```

[51]:
```
array([[1.4+0.j, 2. +0.j, 3. +0.j],
       [4. +0.j, 1. +1.j, 6. +0.j],
       [7. +0.j, 8. +0.j, 9. +0.j]])
```

1.2.4 Matrixvektorprodukt und Matrixmatrixprodukt

[52]:
```
A*v
```

[52]:
$$\begin{bmatrix} 1-i+(1+i)^2 \\ 10-2i \\ 7+5i \end{bmatrix}$$

[53]:
```
A*u
```

```
---------------------------------------------------------------------------
ShapeError                                 Traceback (most recent call last)
Cell In[53], line 1
----> 1 A*u

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/core/decorators.py:118, in call_highest_priority.<locals>.
 ↪priority_decorator.<locals>.binary_op_wrapper(self, other)
    116         if f is not None:
    117             return f(self)
--> 118 return func(self, other)

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/matrices/matrixbase.py:2819, in MatrixBase.__mul__(self, other)
   2790 @call_highest_priority('__rmul__')
   2791 def __mul__(self, other):
   2792     """Return self*other where other is either a scalar or a matrix
   2793     of compatible dimensions.
   2794
   (…)
   2816     matrix_multiply_elementwise
   2817     """
-> 2819     return self.multiply(other)

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/matrices/matrixbase.py:2846, in MatrixBase.multiply(self, other,␣
 ↪dotprodsimp)
   2843 elif T == "is_matrix":
   2845     if self.shape[1] != other.shape[0]:
-> 2846         raise ShapeError(f"Matrix size mismatch: {self.shape} * {other.
 ↪shape}.")
   2848     m = self._eval_matrix_mul(other)
   2850     if isimpbool:

ShapeError: Matrix size mismatch: (3, 3) * (1, 3).
```

[54]:
```
A*x
```

[54]:
$$\begin{bmatrix} 3ix_0 + x_1\,(1+i) + x_2 \\ 3x_0 + 4x_1 + 6x_2 \\ 5x_0 + 6x_1 + x_2 \end{bmatrix}$$

[55]:
```
B = Matrix(3, 3, [0, 1, 0, 1, 0, 0, 0, 0, 1])
B
```

[55]:
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[56]:
```
A*B
```

[56]:
$$\begin{bmatrix} 1+i & 3i & 1 \\ 4 & 3 & 6 \\ 6 & 5 & 1 \end{bmatrix}$$

[57]:
```
B*A
```

[57]:
$$\begin{bmatrix} 3 & 4 & 6 \\ 3i & 1+i & 1 \\ 5 & 6 & 1 \end{bmatrix}$$

1.2.5 spezielle Matrizen (Identität, Nullmatrix, Diagonalmatrix, Ones)

[58]:
```
eye(4)
```

[58]:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[59]:
```
C = zeros(4, 3)
C
```

[59]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

[60]:
```
C[1, 1] = 1
C
```

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

[61]:
```
D = diag(*[1, 2, 3])
D
```

[61]:
$$\begin{bmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 3\end{bmatrix}$$

[62]:
```
diag(1, 2, 3)
```

[62]:
$$\begin{bmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 3\end{bmatrix}$$

[63]:
```
diag(A, B)
```

[63]:
$$\begin{bmatrix}3i & 1+i & 1 & 0 & 0 & 0\\3 & 4 & 6 & 0 & 0 & 0\\5 & 6 & 1 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 0\\0 & 0 & 0 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}$$

[64]:
```
ones(2, 2)
```

[64]:
$$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$$

1.2.6 Matrizen zusammenfügen

[65]:
```
Matrix([[A, B]])
```

[65]:
$$\begin{bmatrix}3i & 1+i & 1 & 0 & 1 & 0\\3 & 4 & 6 & 1 & 0 & 0\\5 & 6 & 1 & 0 & 0 & 1\end{bmatrix}$$

[66]:
```
Matrix.hstack(A, B)
```

[66]:
$$\begin{bmatrix}3i & 1+i & 1 & 0 & 1 & 0\\3 & 4 & 6 & 1 & 0 & 0\\5 & 6 & 1 & 0 & 0 & 1\end{bmatrix}$$

[67]:
```
Matrix.vstack(A, B)
```

[67]:
$$\begin{bmatrix}3i & 1+i & 1\\3 & 4 & 6\\5 & 6 & 1\\0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}$$

[68]:
```
AB = Matrix([A, B])
AB
```

[68]:
$$\begin{bmatrix} 3i & 1+i & 1 \\ 3 & 4 & 6 \\ 5 & 6 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[69]:
```
AB.reshape(1, len(AB))
```

[69]:
$$\begin{bmatrix} 3i & 1+i & 1 & 3 & 4 & 6 & 5 & 6 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2.7 Matrizen mit Iterator

[70]:
```
def hilb(i, j):
    return 1/(i+j+1)
```

[71]:
```
H = Matrix(3, 3, hilb) # Hilbertmatrix
H
```

[71]:
$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

[72]:
```
def superdiag(i, j, a):
    if i == j-1:
        return a[i]
    else:
        return 0
```

[73]:
```
Sup = Matrix(4, 4, lambda i, j : superdiag(i, j, [1, 2, 3]))
Sup
```

[73]:
$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

1.3 Kopieren von Matrizen

[83]:
```
B = Matrix(2, 2, range(4))
BB = B
```

[84]:
```
BB, B
```

[84]:
$$\left(\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}\right)$$

[85]:
```
BB[0, 0] = 9
```

[86]:
```
BB, B
```

[86]:
$$\left(\begin{bmatrix} 9 & 1 \\ 2 & 3 \end{bmatrix}, \begin{bmatrix} 9 & 1 \\ 2 & 3 \end{bmatrix}\right)$$

[87]:
```
BBB = B.copy()
BBBB = B[:, :]
```

[88]:
```
BBB[1, 1] = 99
BBBB[1, 1] = 99
```

[89]:
```
BBBB, BBB, B
```

[89]:
$$\left(\begin{bmatrix} 9 & 1 \\ 2 & 99 \end{bmatrix}, \begin{bmatrix} 9 & 1 \\ 2 & 99 \end{bmatrix}, \begin{bmatrix} 9 & 1 \\ 2 & 3 \end{bmatrix}\right)$$

1.4 Lineare Algebra

1.4.1 Determinate und Inverse

[90]:
```
A = Matrix(3, 3, range(-4, 5)) + eye(3)
A, A.det()
```

[90]:
$$\left(\begin{bmatrix} -3 & -3 & -2 \\ -1 & 1 & 1 \\ 2 & 3 & 5 \end{bmatrix}, -17\right)$$

[91]:
```
Ainv = A**(-1)
Ainv
```

[91]:
$$\begin{bmatrix} -\frac{2}{17} & -\frac{9}{17} & \frac{1}{17} \\ -\frac{7}{17} & \frac{11}{17} & -\frac{5}{17} \\ \frac{5}{17} & -\frac{3}{17} & \frac{6}{17} \end{bmatrix}$$

[92]:
```
A.inv()
```

[92]:
$$\begin{bmatrix} -\frac{2}{17} & -\frac{9}{17} & \frac{1}{17} \\ -\frac{7}{17} & \frac{11}{17} & -\frac{5}{17} \\ \frac{5}{17} & -\frac{3}{17} & \frac{6}{17} \end{bmatrix}$$

[93]:
```
Ainv*A, A, Ainv
```

[93]:
$$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -3 & -3 & -2 \\ -1 & 1 & 1 \\ 2 & 3 & 5 \end{bmatrix}, \begin{bmatrix} -\frac{2}{17} & -\frac{9}{17} & \frac{1}{17} \\ -\frac{7}{17} & \frac{11}{17} & -\frac{5}{17} \\ \frac{5}{17} & -\frac{3}{17} & \frac{6}{17} \end{bmatrix}\right)$$

1.4.2 Funktionen angewandt auf eine Matrix

Elementweise Anwendung einer Funktion

[94]:
```
A.applyfunc(lambda x: exp(x))
```

[94]:
$$\begin{bmatrix} e^{-3} & e^{-3} & e^{-2} \\ e^{-1} & e & e \\ e^{2} & e^{3} & e^{5} \end{bmatrix}$$

Matrixfunktionen, definiert etwa über Potenzreihe

[95]:
```
expA = A.exp()
expA
```

[95]:
$$\begin{bmatrix} -\frac{(78701-55650\sqrt{2})(11\sqrt{2}+16)e^{1+3\sqrt{2}}}{(-246684+174432\sqrt{2})(31\sqrt{2}+44)} - \frac{(-3527+2494\sqrt{2})(-16+11\sqrt{2})}{(-23148+16368\sqrt{2})(-44+31\sqrt{2})e^{-1+3\sqrt{2}}} + \frac{e}{6} & -\frac{(49-35\sqrt{2})(11\sqrt{2}+16)e^{1+3\sqrt{2}}}{(-264+186\sqrt{2})(31\sqrt{2}+44)} - \frac{e}{3} - \frac{(-16-}{(-44+31\sqrt{} \\ -\frac{e}{3} - \frac{(-3527+2494\sqrt{2})(2-\sqrt{2})}{(-23148+16368\sqrt{2})(-5+4\sqrt{2})e^{-1+3\sqrt{2}}} - \frac{(-2-\sqrt{2})(78701-55650\sqrt{2})e^{1+3\sqrt{2}}}{(-246684+174432\sqrt{2})(5+4\sqrt{2})} & -\frac{(-13+9\sqrt{2})(2-\sqrt{2})}{(-24+18\sqrt{2})(-5+4\sqrt{2})e^{-1+3\sqrt{2}}} + \frac{2e}{3} - \frac{(-2-}{(-2} \\ \frac{-3527+2494\sqrt{2}}{(-23148+16368\sqrt{2})e^{-1+3\sqrt{2}}} + \frac{e}{6} + \frac{(78701-55650\sqrt{2})e^{1+3\sqrt{2}}}{-246684+174432\sqrt{2}} & -\frac{e}{3} + \frac{-13+9\sqrt{2}}{(-24+18\sqrt{2})e^{-1+3\sqrt{2}}} + \frac{(49-3}{-26} \end{bmatrix}$$

[96]:
```
expA.applyfunc(simplify) # das macht es nicht besser
```

[96]:
$$\begin{bmatrix} \frac{-845180841e^{6\sqrt{2}}-158617322e^{3\sqrt{2}}-36836184\sqrt{2}+52094231+112159384\sqrt{2}e^{3\sqrt{2}}+597633104\sqrt{2}e^{6\sqrt{2}}}{12(-79308661+56079692\sqrt{2})e^{-1+3\sqrt{2}}} & \frac{-13e^{6\sqrt{2}}-6\sqrt{2}e^{3\sqrt{2}}-3\sqrt{2}+5+8e^{3\sqrt{}}}{6(-4+3\sqrt{2})e^{-1+3\sqrt{2}}} \\ \frac{-270776706e^{6\sqrt{2}}-224318768\sqrt{2}e^{3\sqrt{2}}-46457938+32850723\sqrt{2}+317234644e^{3\sqrt{2}}+191468045\sqrt{2}e^{6\sqrt{2}}}{12(-79308661+56079692\sqrt{2})e^{-1+3\sqrt{2}}} & \frac{1+4e^{3\sqrt{2}}+e^{6\sqrt{2}}}{6e^{-1+3\sqrt{2}}} \\ \frac{-214697014\sqrt{2}e^{6\sqrt{2}}-158617322e^{3\sqrt{2}}-145010107+102537630\sqrt{2}+112159384\sqrt{2}e^{3\sqrt{2}}+303627429e^{6\sqrt{2}}}{12(-79308661+56079692\sqrt{2})e^{-1+3\sqrt{2}}} & \frac{-287\sqrt{2}e^{6\sqrt{2}}-512\sqrt{2}e^{3\sqrt{2}}-1130+799\sqrt{2}+}{12(-181+128\sqrt{2})e^{-1+3\sqrt{}}} \end{bmatrix}$$

1.4.3 charakteristisches Polynom

[97]:
```
H.charpoly()
```

[97]:
$$\text{PurePoly}\left(\lambda^3 - \frac{23}{15}\lambda^2 + \frac{127}{720}\lambda - \frac{1}{2160}, \lambda, domain = \mathbb{Q}\right)$$

[98]:
```
H.det()
```

[98]:
$$\frac{1}{2160}$$

[99]:
```
H.trace()
```

[99]:
$$\frac{23}{15}$$

1.4.4 Kern und Rang

[100]:
```
H.nullspace(), H.rank()
```

[100]:
$$([],\ 3)$$

[101]:
```
CS = H.columnspace() # Basis des Bildes von H, wenn H von links operiert
RS = H.rowspace()
CS, RS
```

[101]:
$$\left(\left[\begin{bmatrix}1\\ \frac{1}{2}\\ \frac{1}{3}\end{bmatrix}, \begin{bmatrix}\frac{1}{2}\\ \frac{1}{3}\\ \frac{1}{4}\end{bmatrix}, \begin{bmatrix}\frac{1}{3}\\ \frac{1}{4}\\ \frac{1}{5}\end{bmatrix}\right], \left[\begin{bmatrix}1 & \frac{1}{2} & \frac{1}{3}\end{bmatrix}, \begin{bmatrix}0 & \frac{1}{12} & \frac{1}{12}\end{bmatrix}, \begin{bmatrix}0 & 0 & \frac{1}{2160}\end{bmatrix}\right]\right)$$

1.4.5 Gram Schmidt Orthogonalisierung

[102]:
```
CSo = GramSchmidt(CS) # Orthogonalisierung
CSo
```

[102]:
$$\left[\begin{bmatrix}1\\ \frac{1}{2}\\ \frac{1}{3}\end{bmatrix}, \begin{bmatrix}-\frac{5}{98}\\ \frac{17}{294}\\ \frac{13}{196}\end{bmatrix}, \begin{bmatrix}\frac{1}{2190}\\ -\frac{1}{365}\\ \frac{1}{365}\end{bmatrix}\right]$$

[103]:
```
n = len(CSo)
[(CSo[i].H * CSo[j])[0] for i in range(n) for j in range(n)]
```

[103]:
$$\left[\frac{49}{36},\ 0,\ 0,\ 0,\ \frac{73}{7056},\ 0,\ 0,\ 0,\ \frac{1}{65700}\right]$$

[109]:
```
B = Matrix(CSo).reshape(3,3).T
B.H * B
```

[109]:
$$\begin{bmatrix}\frac{49}{36} & 0 & 0\\ 0 & \frac{73}{7056} & 0\\ 0 & 0 & \frac{1}{65700}\end{bmatrix}$$

1.5 Lösung linearer Gleichungssysteme

1.5.1 Lösung mit speziellen Verfahren,

- LR Zerlegung (engl. LU decomposition) (mehr dazu in CompLA und Numerik I)
- QR Zerlegung (engl. QR decomposition) (mehr dazu in CompLa und Numerik I)
- Gaußelimination Schule/Lineare Algebra, hier wird das erweiterte lineare System auf Zeilenstufenform (engl. row echelon form) gebracht. https://en.wikipedia.org/wiki/Gaussian_elimination

Diese Verfahren sind für 'symbolische' Matrizen und Vektoren gedacht.

[110]:
```
A
```

[110]:
$$\begin{bmatrix}-3 & -3 & -2\\ -1 & 1 & 1\\ 2 & 3 & 5\end{bmatrix}$$

[111]:
```
A.LUsolve(v)
```

[111]:

$$\begin{bmatrix} -\frac{8}{17} - \frac{10i}{17} \\ \frac{6}{17} + \frac{16i}{17} \\ \frac{3}{17} - \frac{9i}{17} \end{bmatrix}$$

[112]:
```
A.QRsolve(v)
```

[112]:
$$\begin{bmatrix} \frac{\sqrt{14}\left(-\frac{8\sqrt{14}}{17} - \frac{10\sqrt{14}i}{17}\right)}{14} \\ \frac{\sqrt{5}\left(\frac{6\sqrt{5}}{17} + \frac{16\sqrt{5}i}{17}\right)}{5} \\ \frac{\sqrt{70}\left(\frac{3\sqrt{70}}{70} - \frac{9\sqrt{70}i}{70}\right)}{17} \end{bmatrix}$$

[114]:
```
A.gauss_jordan_solve(v)
```

[114]:
$$\left(\begin{bmatrix} -\frac{8}{17} - \frac{10i}{17} \\ \frac{6}{17} + \frac{16i}{17} \\ \frac{3}{17} - \frac{9i}{17} \end{bmatrix}, \right)$$

[115]:
```
A.inv()*v
```

[115]:
$$\begin{bmatrix} -\frac{8}{17} - \frac{10i}{17} \\ \frac{6}{17} + \frac{16i}{17} \\ \frac{3}{17} - \frac{9i}{17} \end{bmatrix}$$

[123]:
```
B = Matrix(3, 4, range(12))
B.gauss_jordan_solve(Matrix(3, 1, [0, 1, 2])) # allgemeiner Lösungsraum, falls␣
↪B mehr Spalten als Zeilen hat
```

[123]:
$$\left(\begin{bmatrix} \tau_0 + 2\tau_1 + \frac{1}{4} \\ -2\tau_0 - 3\tau_1 \\ \tau_0 \\ \tau_1 \end{bmatrix}, \begin{bmatrix} \tau_0 \\ \tau_1 \end{bmatrix} \right)$$

1.5.2 Lösung mit ‘solve’ und ‘linsolve’

linsolve nutzt intern gauss_jordan_solve

[127]:
```
x = Matrix(3, 1, xs)
x
```

[127]:
$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

[128]:
```
lgs = Eq(A*x, v)
lgs
```

[128]:
$$\begin{bmatrix} -3x_0 - 3x_1 - 2x_2 \\ -x_0 + x_1 + x_2 \\ 2x_0 + 3x_1 + 5x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1+i \\ 1-i \end{bmatrix}$$

[129]:
```
solve(lgs, x)
```

[129]:
$$\left\{x_0 : -\frac{8}{17} - \frac{10i}{17},\ x_1 : \frac{6}{17} + \frac{16i}{17},\ x_2 : \frac{3}{17} - \frac{9i}{17}\right\}$$

[130]:
```
linsolve((A, v), xs)  # erweiterte Matrix
```

[130]:
$$\left\{\left(-\frac{8}{17} - \frac{10i}{17},\ \frac{6}{17} + \frac{16i}{17},\ \frac{3}{17} - \frac{9i}{17}\right)\right\}$$

[131]:
```
linsolve(A*x-v, xs) # Vektor/Liste mit  Gleichungen
```

[131]:
$$\left\{\left(-\frac{8}{17} - \frac{10i}{17},\ \frac{6}{17} + \frac{16i}{17},\ \frac{3}{17} - \frac{9i}{17}\right)\right\}$$

[132]:
```
linsolve([Eq(lgs.rhs[j], lgs.lhs[j]) for j in range(3)], xs)
```

[132]:
$$\left\{\left(-\frac{8}{17} - \frac{10i}{17},\ \frac{6}{17} + \frac{16i}{17},\ \frac{3}{17} - \frac{9i}{17}\right)\right\}$$

1.5.3 Matrizen mit Parameter/Ausdrücken

[133]:
```
a = symbols("a")
A[0, 1] = a
A
```

[133]:
$$\begin{bmatrix} -3 & a & -2 \\ -1 & 1 & 1 \\ 2 & 3 & 5 \end{bmatrix}$$

[134]:
```
A.LUsolve(v).applyfunc(cancel)
```

[134]:
$$\begin{bmatrix} \frac{a(-4-6i)-4-8i}{7a+4} \\ \frac{-6-16i}{7a+4} \\ \frac{a(3+i)+6+12i}{7a+4} \end{bmatrix}$$

[135]:
```
linsolve((A, v), xs)
```

[135]:
$$\left\{\left(\frac{a\left(-4-6i\right)-4-8i}{7a+4},\ \frac{-6-16i}{7a+4},\ \frac{a\left(3+i\right)+6+12i}{7a+4}\right)\right\}$$

[139]:
```
M = Matrix(2, 2, [a**2, 1 / (1 + a), a + 1, a - 1])
u = Matrix(2, 1, [1, 2])
M.gauss_jordan_solve(u)[0].applyfunc(cancel)
```

[139]:
$$\begin{bmatrix} \frac{a^2-3}{a^4-a^2-a-1} \\ \frac{2a^2-a-1}{a^3-a^2-1} \end{bmatrix}$$

1.6 Polygone (Farbe, Marker, Liniendicke)

[140]:
```
%matplotlib qt
import matplotlib.pyplot as plt
```

[141]:
```
dreieck = np.array([[0, 0], [1, 0], [0, 1], [0, 0]])
```

[142]:
```
fig, ax = plt.subplots()
ax.plot(dreieck[:, 0], dreieck[:, 1])
ax.axis('equal');
```

[145]:
```
def n_eck(n):
    theta = np.linspace(0, 2*np.pi, n+1)
    return np.sin(theta), np.cos(theta)
```

[147]:
```
fig, ax = plt.subplots()
ax.plot(n_eck(5))
ax.axis('equal');
```

[148]:
```
fig, ax = plt.subplots()
for n in range(3, 9):
    farbe = plt.cm.hot((n - 3) / 6) # das Argument von hot soll in [0 1] liegen
    ax.plot(*n_eck(n), 'o-', color=farbe)
ax.axis('equal');
```

[149]:
```
x = symbols('x')
f = S(1) / 7 * x**(S(3) / 2) * ((3 / 2)**(sqrt(x)) - floor((3 / 2)**(sqrt(x))))
f
```

[149]:
$$\frac{x^{\frac{3}{2}}\left(1.5^{\sqrt{x}} - \left\lfloor 1.5^{\sqrt{x}} \right\rfloor\right)}{7}$$

[150]:
```
fn = lambdify(x, f)
xn = np.linspace(0, 19.7, 500)
fig, ax = plt.subplots()
ax.plot(xn, fn(xn))
```

[150]:
```
[<matplotlib.lines.Line2D at 0x7f038a0e3bc0>]
```

[151]:
```
ax.plot(xn, fn(xn), xn, -fn(xn), linewidth=3, color='green');
```

[152]:
```
fig, ax = plt.subplots()
ax.plot(fn(xn), -xn, -fn(xn), -xn, linewidth=4, color='green');
ax.axis('equal');
```

[153]:
```
ax.plot([-.5, .5, .5, -.5, -.5],[-19.5, -19.5, -22, -22, -19.5], color='brown',␣
↪linewidth='4');
```

[154]:
```
def n_stern(n):
    x = [(0.5+ j % 2)*np.sin(np.pi*2*j/n) for j in range(2*n+1)]
    y = [(0.5+ j % 2)*np.cos(np.pi*2*j/n) for j in range(2*n+1)]
    return x, y
```

[156]:
```
ax.plot(*n_stern(7), color='gold', linewidth=2.5);
```