```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# """
# Created on Mon Dec 18 14:52:29 2017
#
# @author: christianehelzel
# """
```

1 11. Vorlesung: Singulärwertzerlegung, Teil 1

```
import numpy as np
import numpy.linalg as npl
from matplotlib import pyplot as plt

plt.close('all')
```

1.1 Singulärwertzerlegung in numpy.linalg

full SVD

```
A = np.random.rand(9, 6);
U, sigma, Vh = npl.svd(A);
print(U.shape, sigma.shape, Vh.shape)
V = Vh.T
```

```
(9, 9) (6,) (6, 6)
```

Erzeuge mxn Diagonalmatrix Sigma:

```
Sigma = np.zeros((9,6),dtype=complex)
Sigma[:6, :6] = np.diag(sigma)
```

Überprüfe: A = U Sigma V^T

```
print(np.max(np.abs(A-U@Sigma@V.T)))
```

```
1.66533453694e-15
```

reduced SVD

```
A = np.random.rand(9, 6);
U, sigma, Vh = npl.svd(A, full_matrices=False);
print(U.shape, sigma.shape, Vh.shape)
V = Vh.T
```

```
(9, 6) (6,) (6, 6)
```

Erzeuge nxn Diagonalmatrix Sigma:

```
Sigma = np.diag(sigma)
```

Überprüfe: A = U Sigma V^T

```
print(np.max(np.abs(A-U@Sigma@V.T)))
```

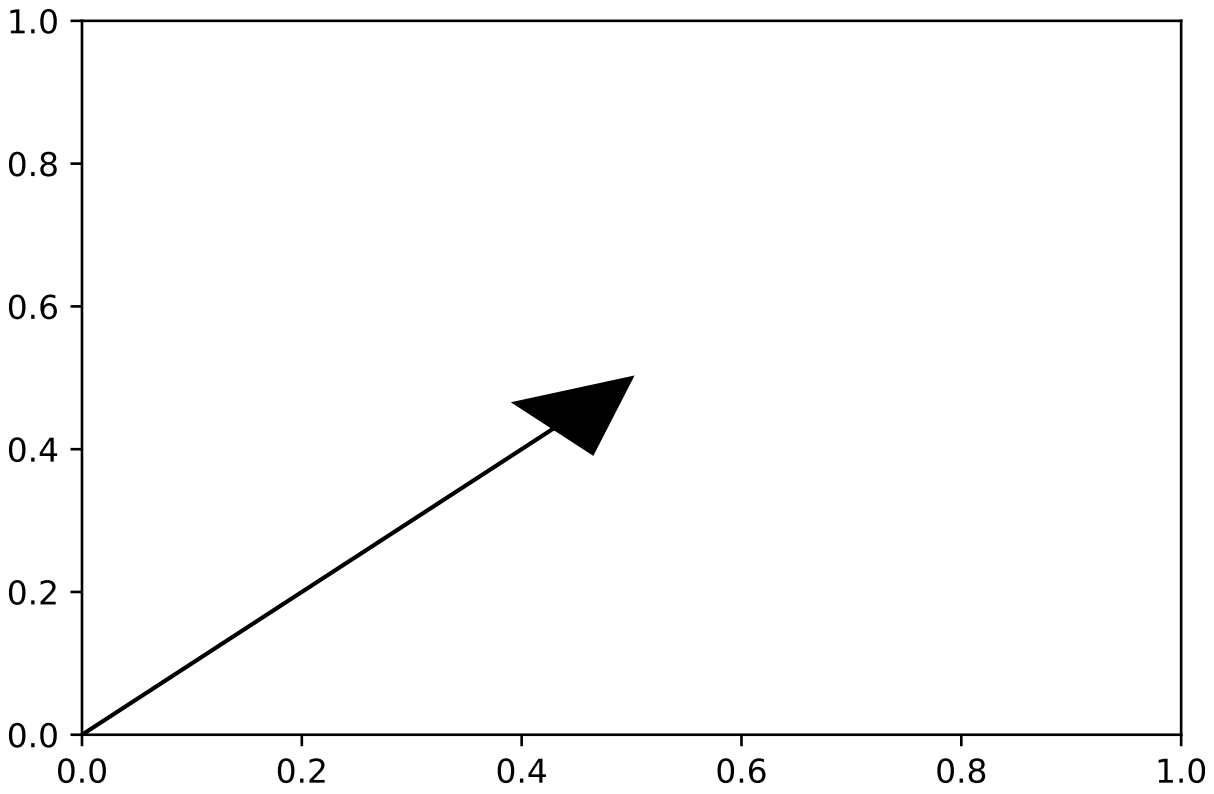
```
6.80011602583e-16
```

Beachte: als default wird full_matrix = True benutzt

1.2 Zeichne Vektor (Pfeil)

```
plt.figure();
plt.arrow(0, 0, 0.5, 0.5, head_width=0.1, head_length=0.1, facecolor='k',\
          edgecolor='k', length_includes_head='True')
```

```
<matplotlib.patches.FancyArrow at 0x7f3c6c5b07f0>
```



Da wir demnächst mehrere Vektoren zeichnen wollen, führen wir eine Variable (ein Dictionary) für die optionalen Parameter ein

```
arrowProps = {
    "head_length": 0.1,
```

```
    "head_width":  0.1,
    "edgecolor":   'k',
    "facecolor":   'k',
    "length_includes_head": True,
}
```

Anmerkung: Dictionarys weisen einem Schlüssel (z.B. ein String) ein anderes Objekt zu. Beispiel:

```
print(arrowProps['edgecolor'])
```

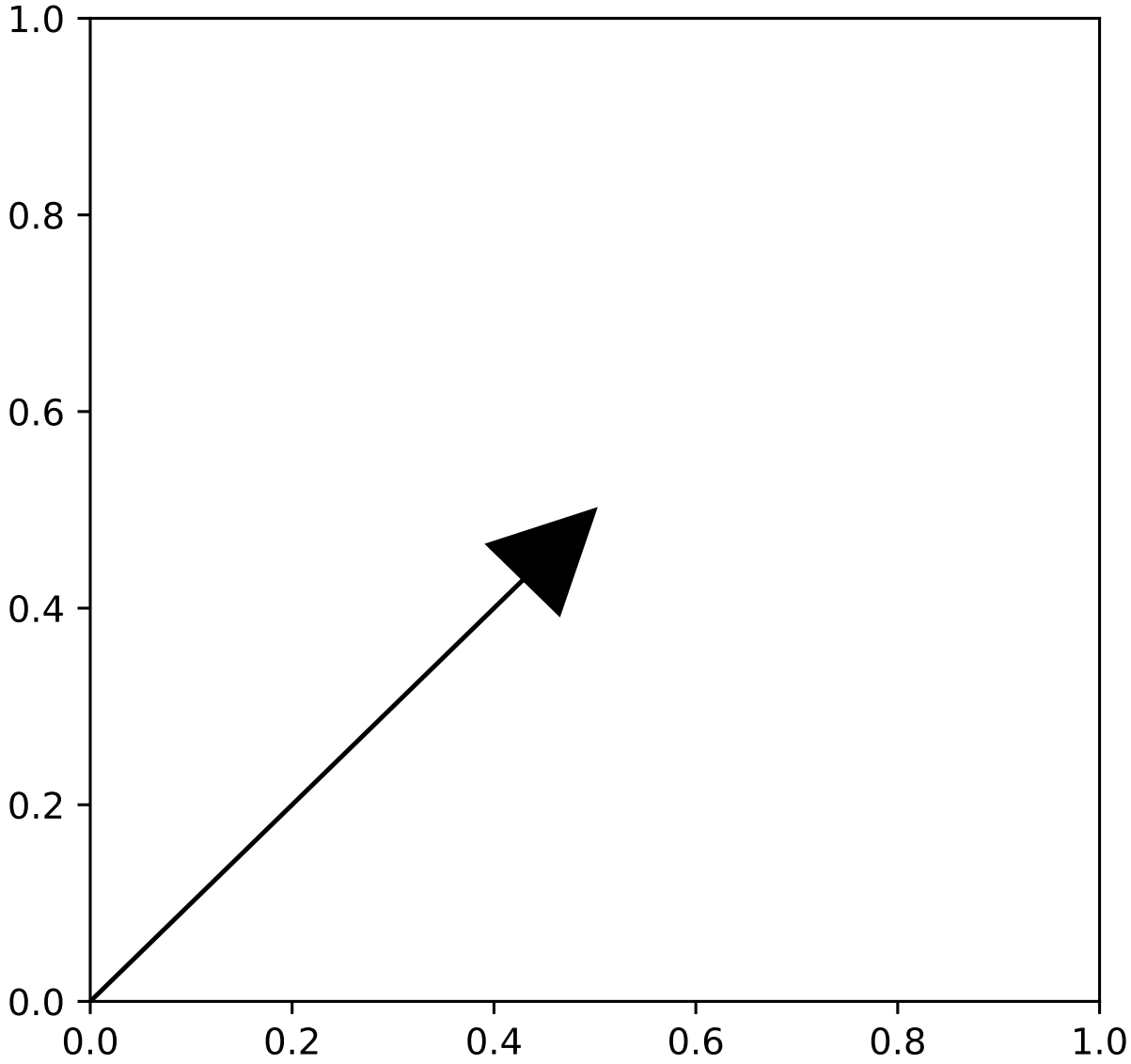
```
k
```

Es ist also so was wie ein Liste, nur dass man statt einer Zahl als Index auch andere Dinge verwenden kann. Weiter Infos unter https://www.python-kurs.eu/python3_dictionaries.php

Zeichnen

```
plt.figure(figsize = (5,5));
plt.arrow(0, 0, 0.5, 0.5, **arrowProps)
# **arrowProps: entpackt das Dictionary für den Funktionsaufruf
```

```
<matplotlib.patches.FancyArrow at 0x7f3c6c23b780>
```

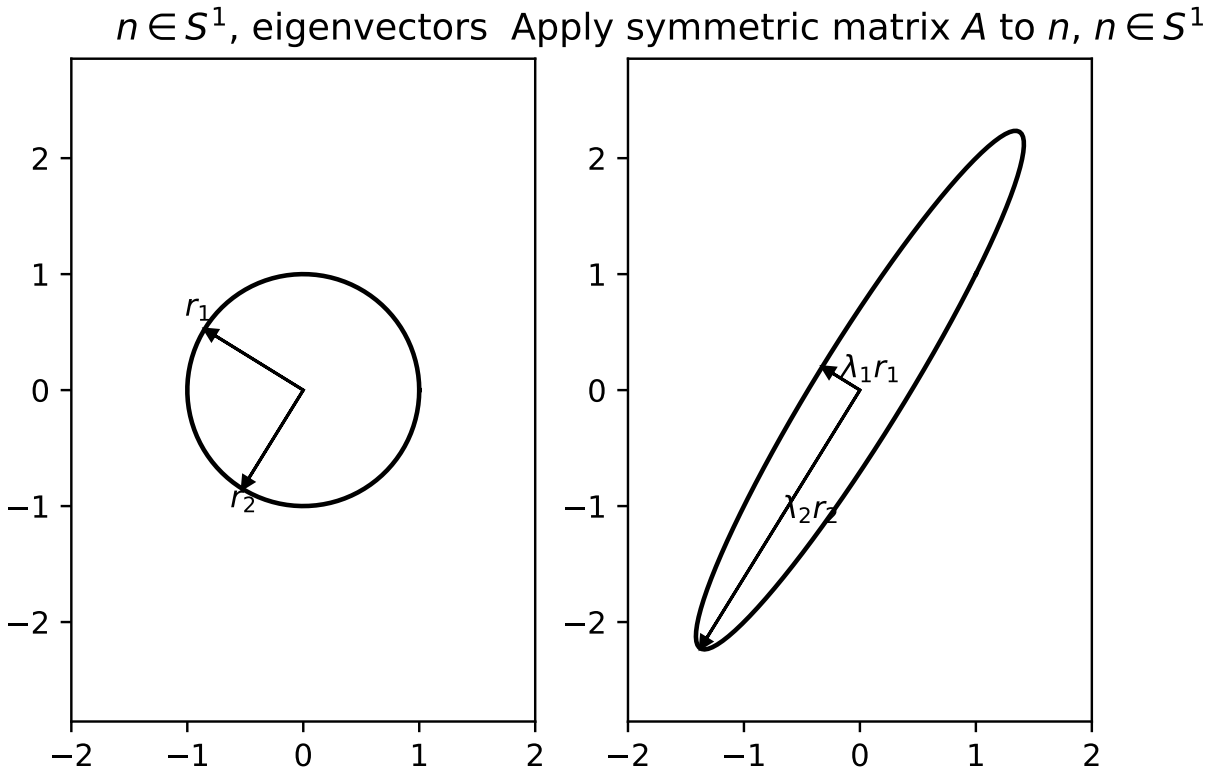


Geometrische Interpretation für symmetrische Matrizen: Die Einheitssphäre wird auf eine Ellipse abgebildet, lamda_1 r_1 und lambda_2 r_2 beschreiben die Hauptachsen der Ellipse

Betrachte zunächt eine symmetrische Matrix

```
A = np.array([[1.,1.],[1.,2.]])
eig, R = npl.eig(A)

plt.figure()
plt.subplot(121)

# Definiere Vektoren n auf der Einheitssphäre S1
npt = 100
phi = np.linspace(0, 2 * np.pi,npt);
n = np.zeros((2,npt))
n[0,:] = np.cos(phi)
n[1,:] = np.sin(phi)

# Zeichne Punkte auf Einheitssphäre S1
plt.plot(*n,'k') # * entpackt das Array
plt.axis('equal')
```

```
# Zeichne die Vektoren v_1 und v_2
plt.arrow(0, 0, *R[:,0], **arrowProps) # r_1
plt.arrow(0, 0, *R[:,1], **arrowProps) # r_2
plt.text(*1.2*R[:, 0], "$r_1$");
plt.text(*1.2*R[:, 1], "$r_2$");
plt.axis('equal')
plt.axis([-2, 2, -3, 3])
plt.title('$n \in S^1$, eigenvectors')

plt.subplot(122)

# Berechne die Vektoren A@n und zeichne die Werte als Punkte auf Ellipse
b=np.zeros((2,npt))
b = A@n

plt.plot(*b,'k')

# Zeichne die Vektoren lambda1*r1 und lambda2*r2
plt.arrow(0, 0, *A@R[:,0], **arrowProps)
plt.arrow(0, 0, *A@R[:,1], **arrowProps)
plt.text(*0.5*eig[0]*R[:, 0], "$\lambda_1 r_1$");
plt.text(*0.5*eig[1]*R[:, 1], "$\lambda_2 r_2$");
plt.axis('equal')
plt.axis([-2, 2, -3, 3])
plt.title('Apply symmetric matrix $A$ to $n$, $n \in S^1$')
plt.show()
```



Beachte: *R[:,0] entpackt Array / Tupel

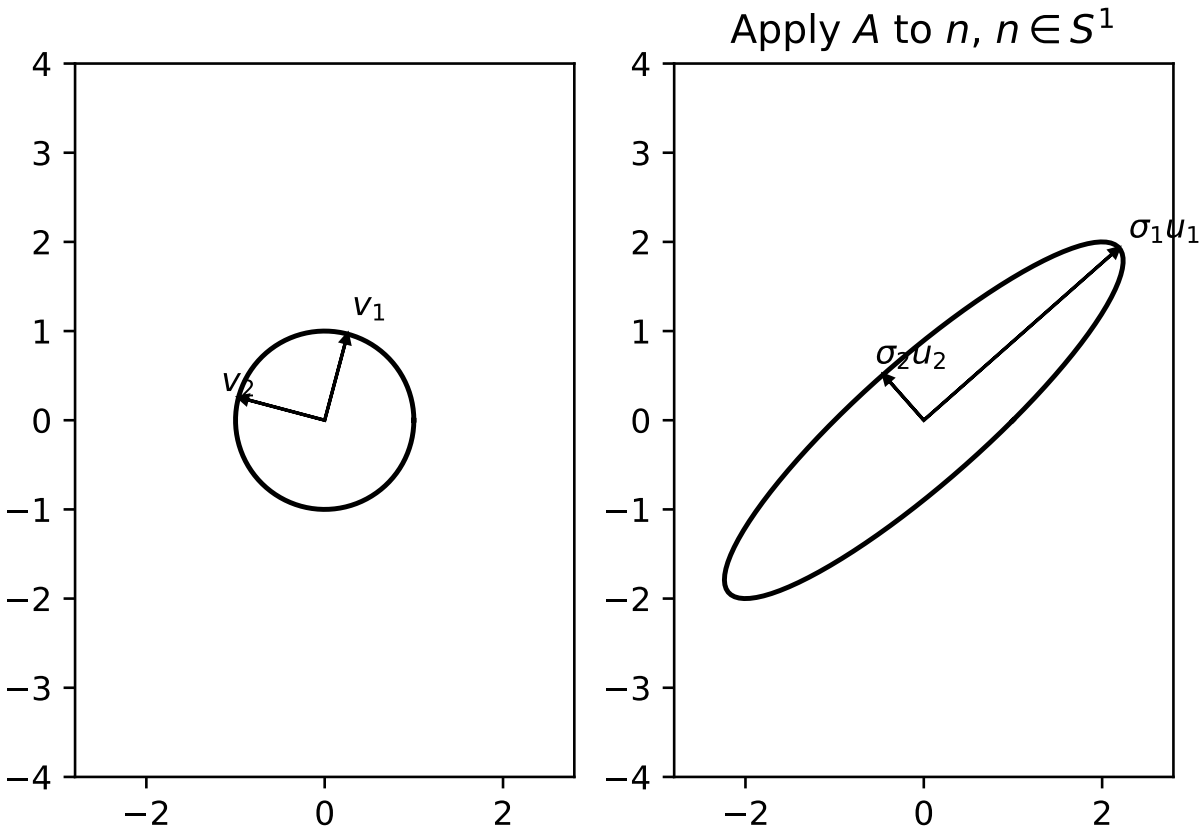
Geometrische Interpretation der SVD: Das Bild der Einheitskugel ist für jede mxn Matrix eine Hyperellipse Singulärwerte beschreiben die Hauptachsen der Ellipse

```
from matplotlib import pyplot as plt
#import matplotlib.pyplot as plt ist das selbe
A = np.array([[1.,2.],[0.,2.]])
U,s,Vh = npl.svd(A);
S = np.diag(s)
V = Vh.T
```

und nun zeichnen

```
plt.figure()
plt.subplot(121)

# Zeichne Punkte auf Einheitssphäre S1
plt.plot(*n,'k')
plt.axis('equal')

# Zeichne die Vektoren v_1 und v_2
plt.arrow(0, 0, *V[:,0], **arrowProps) # v_1
plt.arrow(0, 0, *V[:,1], **arrowProps) # v_2
plt.text(*1.2*V[:, 0], "$v_1$");
plt.text(*1.2*V[:, 1], "$v_2$");
plt.axis('equal')
plt.axis([-3, 3, -4, 4])

plt.subplot(122)
# Berechne die Vektoren A@n und zeichne Punkte auf Ellipse
b = A@n
plt.plot(*b,'k')

# Zeichne die Vektoren sigma1*u1 und sigma2*u2
plt.arrow(0, 0, *A@V[:,0], **arrowProps)
plt.arrow(0, 0, *A@V[:,1], **arrowProps)
plt.text(*1.05*s[0]*U[:, 0], "$\sigma_1 u_1$");
plt.text(*1.2*s[1]*U[:, 1], "$\sigma_2 u_2$");
plt.axis('equal')
plt.axis([-3, 3, -4, 4])
plt.title('Apply $A$ to $n$, $n \in S^1$')
```

```
Text(0.5,1,'Apply $A$ to $n$, $n                    in S^1$')
```



Es gilt A = U @ S @ V.T

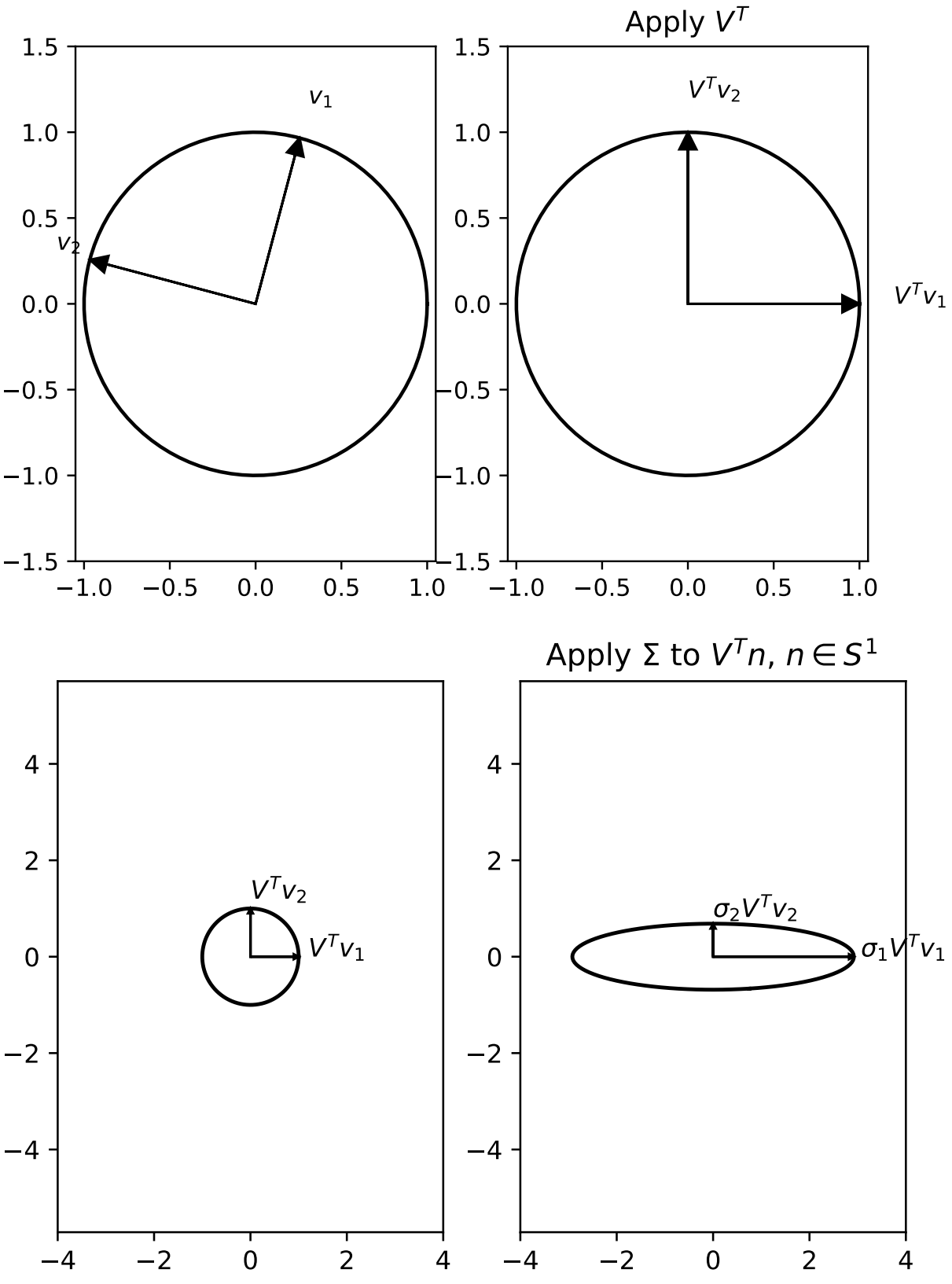
Wir betrachten die Wirkung der einzelnen Matrizen separat

1. Multiplikation mit V.T:

```
plt.figure()
plt.subplot(121)
plt.plot(*n,'k')

plt.arrow(0, 0, *V[:,0], **arrowProps)
plt.arrow(0, 0, *V[:,1], **arrowProps)
plt.text(*1.2*V[:, 0], "$v_1$", color = "black");
plt.text(*1.2*V[:, 1], "$v_2$", color = "black");
plt.axis('equal')
plt.axis([-1.5, 1.5, -1.5, 1.5])

plt.subplot(122)
plt.plot(*n,'k')

plt.arrow(0,0,*V.T@V[:,0], **arrowProps)
plt.arrow(0,0,*V.T@V[:,1], **arrowProps)
plt.text(*1.2*V.T@V[:, 0], "$V^T v_1$");
plt.text(*1.2*V.T@V[:, 1], "$V^T v_2$");
plt.axis('equal')
plt.axis([-1.5, 1.5, -1.5, 1.5])

plt.title('Apply $V^T$')
# 2. Multiplikation mit Diagonalmatrix:

plt.figure()
```

```
plt.subplot(121)
plt.plot(*n,'k')

plt.arrow(0,0,*V.T@V[:,0], **arrowProps)
plt.arrow(0,0,*V.T@V[:,1], **arrowProps)
plt.text(*1.2*V.T@V[:, 0], "$V^T v_1$");
plt.text(*1.2*V.T@V[:, 1], "$V^T v_2$");
plt.axis('equal')
plt.axis([-4, 4, -6, 6])

plt.subplot(122)

d=np.zeros((2,npt))

d = S@V.T@n

plt.plot(*d,'k')
plt.arrow(0,0,*s[0]*V.T@V[:,0], **arrowProps)
plt.arrow(0,0,*s[1]*V.T@V[:,1], **arrowProps)
plt.text(*1.05*s[0]*V.T@V[:, 0], '$\sigma_1 V^T v_1$');
plt.text(*1.2*s[1]*V.T@V[:, 1], '$\sigma_2 V^T v_2$');
plt.axis('equal')
plt.axis([-4, 4, -6, 6])
plt.title('Apply $\Sigma$ to $V^T n$, $n \in S^1$')
```

```
Text(0.5,1,'Apply $          Sigma$ to $V^T n$, $n          in S^1$')
```



3. Multiplikation mit U:

```
plt.figure()
plt.subplot(121)
plt.plot(*d,'k')
plt.arrow(0,0,*s[0]*V.T@V[:,0], **arrowProps)
```

```
plt.arrow(0,0,*s[1]*V.T@V[:,1], **arrowProps)
plt.text(*1.05*s[0]*V.T@V[:, 0], '$\sigma_1 V^T v_1$');
plt.text(*1.2*s[1]*V.T@V[:, 1], '$\sigma_2 V^T v_2$');
plt.axis('equal')
plt.axis([-4, 4, -6, 6])

plt.subplot(122)
Ud = U@d
plt.plot(*Ud,'k')

# Zeichne die Vektoren sigma1*u1 und sigma2*u2
plt.arrow(0, 0, *U@(s[0]*V.T@V[:,0]), **arrowProps)
plt.arrow(0, 0, *U@(s[1]*V.T@V[:,1]), **arrowProps)
plt.text(*1.05*s[0]*U[:, 0], '$\sigma_1 u_1$');
plt.text(*1.2*s[1]*U[:, 1], '$\sigma_2 u_2$');
plt.title('Apply $U$ to $V^T \Sigma n$, $n\in S^1$')
plt.axis('equal')
plt.axis([-4, 4, -6, 6])
```

```
[-4, 4, -6, 6]
```