lektion7_inline

November 19, 2026

Table of Contents

1 Matplotlib (Teil 2)

1.1 Wiederholung

1.2 Achsen und Gitterlinien

1.3 Reihenfolge der Elemente festlegen

1.4 Bilder abspeichern

1.5 Logarithmische Skalierung der Achsen

1.6 Scatter Plots

1.7 Vektorpfeile mit quiver

1.8 Flächen füllen

1.9 Zeichnen von 2d reellwertigen Daten (imshow und contour)

2 3D Plots

2.1 Polygonzug in 3D

2.2 Fläche in 3d mit plot_surface

2.3 Vektorpfeile in 3d mit quiver

1 Matplotlib (Teil 2)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
#modus für Vorlesung
#%matplotlib qt5
#plt.rcParams['figure.figsize'] = (4., 4.)
#plt.rcParams['lines.linewidth'] = 3
#modus für Export nach pdf
%matplotlib inline
plt.rcParams["figure.figsize"] = (3.0, 2.0) #default figsize für Export nach pdf
```

[ ]:
```
import matplotlib
from matplotlib.backends import backend_registry
```

```
# interactive backends
#backend_registry.list_builtin(matplotlib.backends.BackendFilter.INTERACTIVE)
# noninteractive backends
backend_registry.list_builtin(matplotlib.backends.BackendFilter.NON_INTERACTIVE)
# all backends built into Matplotlib
# backend_registry.list_builtin()
# all registered backends
#backend_registry.list_all()
```

[ ]:
```
#plt.rcParams.keys()
```

https://matplotlib.org/stable/gallery/color/named_colors.html

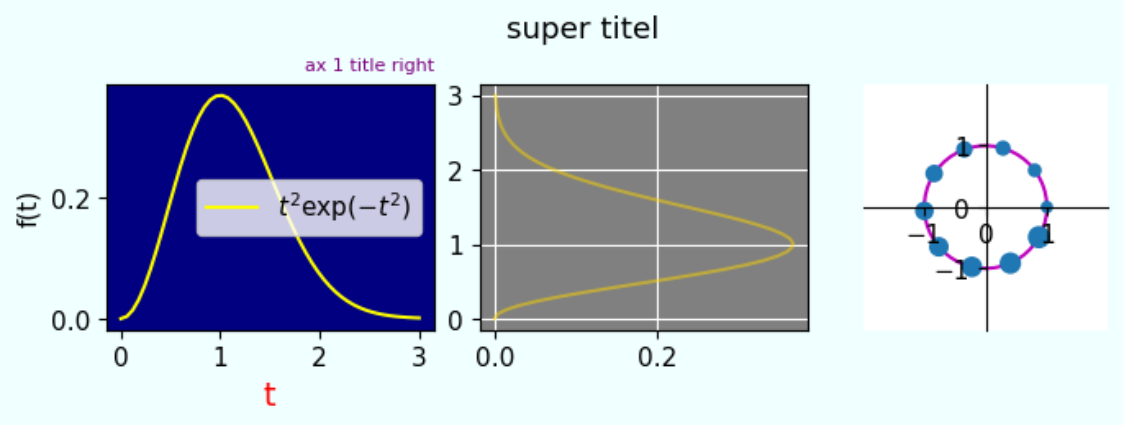
1.1 Wiederholung

1.2 Achsen und Gitterlinien

Hinzufügen von Achsenbezeichnungen. **layout='constrained'** in plt.subplots sorgt dafür, dass alles sichtbar bleibt.

[2]:
```
fig, ax = plt.subplots(1, 3, layout='constrained', figsize=(7, 2.5))
fig.suptitle('super titel')
fig.set_facecolor('azure')

ax[0].set_title('ax 1 title right', loc='right', color='purple', fontsize=8)
ax[0].set_facecolor('navy')

def f(t):
    return t**2 * np.exp(-t**2)

t = np.linspace(0, 3, 51)
y = f(t)

ax[0].plot(t, y, color='yellow', label='$t^2\\exp(-t^2)$')
ax[0].legend();

ax[0].set_xlabel('t', color='red', fontsize=14)
ax[0].set_ylabel('f(t)');

ax[1].set_facecolor('grey')
ax[1].grid(color='white')
ax[1].plot(y, t, color='gold', alpha=.5);

phi = np.linspace(0, 2 * np.pi)
ax[2].plot(np.cos(phi), np.sin(phi), 'm', zorder=1)
ax[2].scatter(np.cos(phi[::5]), np.sin(phi[::5]), s=20 + 10 * phi[::5])
ax[2].set_aspect('equal')
# Setze die bottom und left Achsen (engl. spines) als x und y Achse
```

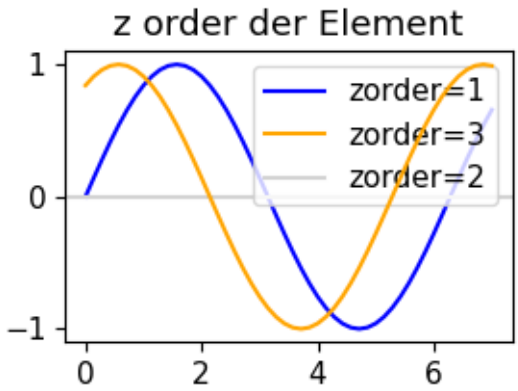
```
ax[2].spines['bottom'].set_position('zero')
ax[2].spines['left'].set_position('zero')
# Entferne top und right spines
ax[2].spines['top'].set_visible(False)
ax[2].spines['right'].set_visible(False)

ax[2].set_xlim((-2, 2))
ax[2].set_ylim((-2, 2))
ax[2].set_xticks([-1, 0, 1])
ax[2].set_yticks([-1, 0, 1]);
```



1.3 Reihenfolge der Elemente festlegen

zorder legt fest, in welcher Reihenfolge die Elemente gezeichnet werden.

[3]:
```
x = np.linspace(0, 7)

plt.figure()
plt.plot(x, np.sin(x), label='zorder=1', color='blue', zorder=1)
plt.plot(x, np.sin(x + 1), label='zorder=3', color='orange', zorder=3)
plt.axhline(0, label='zorder=2', color='lightgrey', zorder=2)

plt.title('z order der Element')
l = plt.legend(loc='upper right')
l.set_zorder(2)
plt.show()
```



Position der Legende

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

1.4 Bilder abspeichern

Bild in einer Datei (pdf, jpeg, ...) speichern:

savefig

[ ]:
```
fig.savefig('bild1.pdf')  # das format wird anhand der Endung geraten
```

[ ]:
```
fig.savefig('bild1.jpg', format='jpg')
```

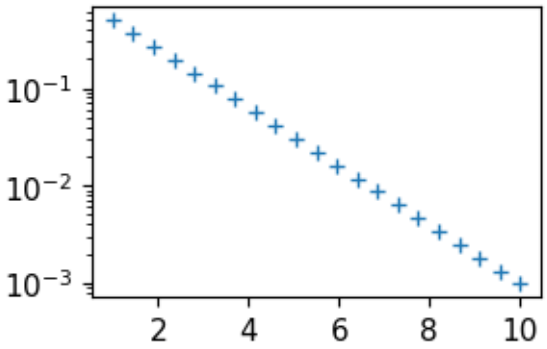
[ ]:
```
fig.canvas.get_supported_filetypes_grouped()
```

1.5 Logarithmische Skalierung der Achsen

Verwendung logarithmischer Achsen:

[4]:
```
x = np.linspace(1, 10, 21)
y = 2**(-x)
```

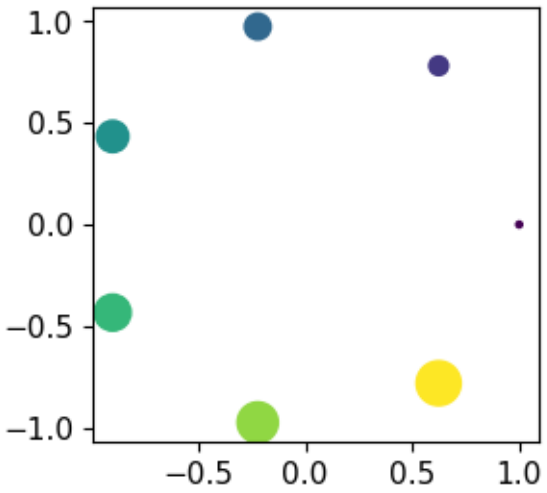
```
fig, ax = plt.subplots(1)
ax.semilogy(x, y, '+');
```



Siehe auch semilogx() und loglog(), wenn die x-Achse bzw. beide Achsen logarithmisch skaliert werden sollen.

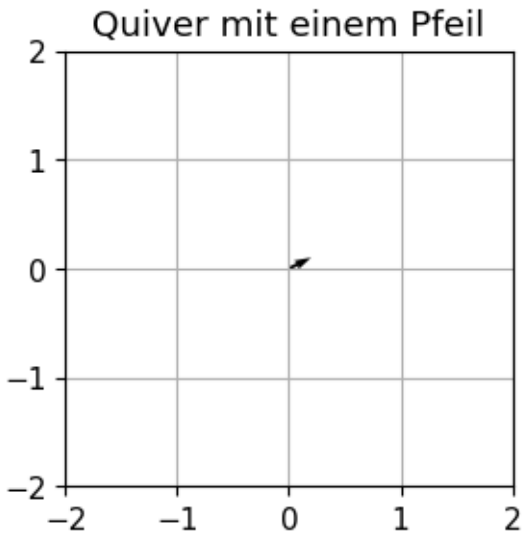
1.6 Scatter Plots

[5]:
```
fig, ax = plt.subplots(figsize=(3, 3))
n = 7
t = np.linspace(0, 2 * np.pi, n, endpoint=False)
x = np.cos(t)
y = np.sin(t)
color = t
# Punkte an Koordinate (x_i, y_i) s: Größe  c: Farbe
ax.scatter(x, y, s=50 * t + 5, c=color);
```

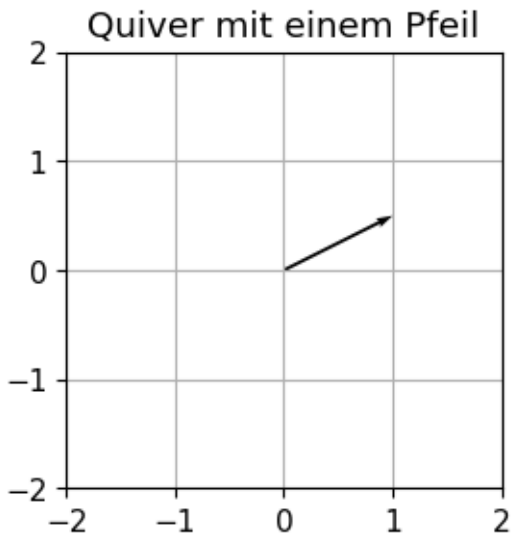


1.7 Vektorpfeile mit quiver

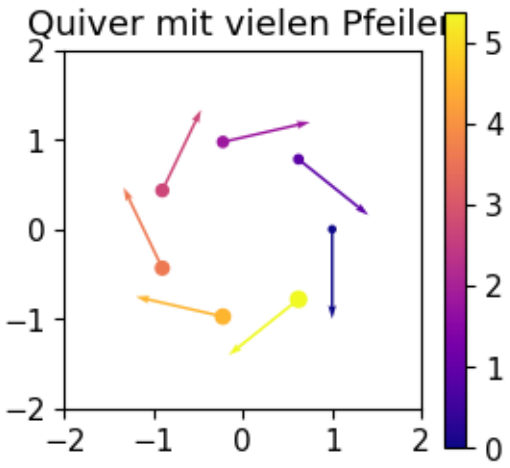
[6]:
```
fig, ax = plt.subplots(figsize=(3, 3))
x, y, u, v = 0, 0, 1, .5

ax.quiver(x, y, u, v)  # (x,y) Fusspunkt (u,v) Richtung
ax.set_title('Quiver mit einem Pfeil')
ax.axis([-2, 2, -2, 2])
ax.grid()
```



[7]:
```
fig, ax = plt.subplots(figsize=(3, 3))
x, y, u, v = 0, 0, 1, .5

ax.quiver(x, y, u, v, angles='xy', scale_units='xy', scale=1)
ax.set_title('Quiver mit einem Pfeil')
ax.axis([-2, 2, -2, 2])
ax.grid()
ax.set_aspect('equal')
```



[9]:
```
fig, ax = plt.subplots(figsize=(3, 3))
n = 7
t = np.linspace(0, 2 * np.pi, n, endpoint=False)
x = np.cos(t)
y = np.sin(t)
u = np.sin(t)
v = -np.cos(t)
color = t
# Punkte an Koordinate (x_i, y_i) s: Größe  c: Farbe
ax.scatter(x, y, s=5 * t + 5, c=color, cmap='plasma');

qv = ax.quiver(x, y, u, v, color, angles='xy', scale_units='xy',
               scale=1, cmap='plasma')
ax.set_title('Quiver mit vielen Pfeilen')
ax.axis([-2, 2, -2, 2])
ax.set_aspect('equal')

fig.colorbar(qv);
```

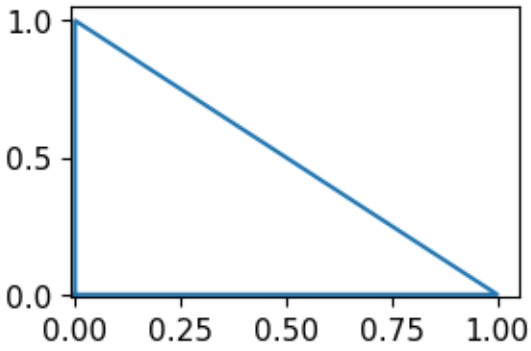


Der Parameter **cmap** wählt die Colormap aus.
Link zu Colormaps (cmap) https://matplotlib.org/stable/tutorials/colors/colormaps.html

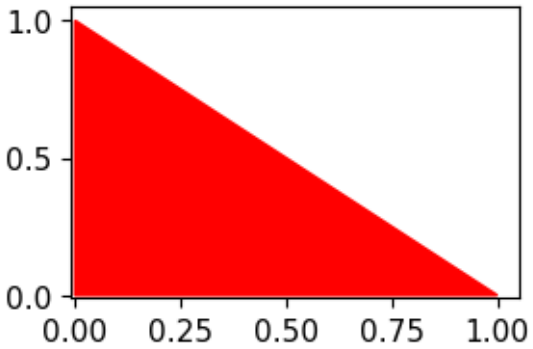
1.8 Flächen füllen

fill

[11]:
```
fig, ax = plt.subplots(1)
ax.plot([0, 1, 0, 0], [0, 0, 1, 0])
ax.axis(xmin=-.01, ymin=-.01);
```

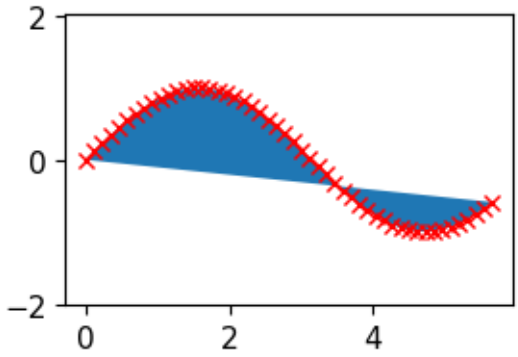


[12]:
```
fig, ax = plt.subplots(1)
ax.fill([0, 1, 0, 0], [0, 0, 1, 0], color='red')
ax.axis(xmin=-.01, ymin=-.01);
```



[13]:
```
x = np.linspace(0, 1.8 * np.pi)
y1 = np.sin(x)
```

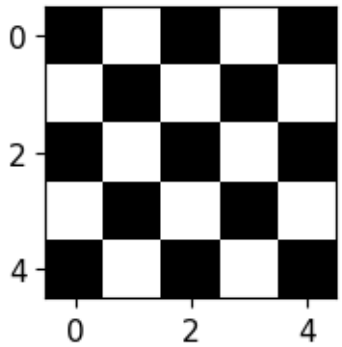
[14]:
```
fig, ax = plt.subplots(1)
ax.fill(x, y1)
ax.plot(x, y1, 'xr')
ax.axis('equal');
```



1.9 Zeichnen von 2d reellwertigen Daten (imshow und contour)

imshow zum darstellen von gepixelten Bildern

[15]:
```
i = np.arange(5)
M = (-1)**(i[:, np.newaxis] + i)
fig, ax = plt.subplots(1)
ax.imshow(M, cmap='Greys');
```



[16]:
```
M
```

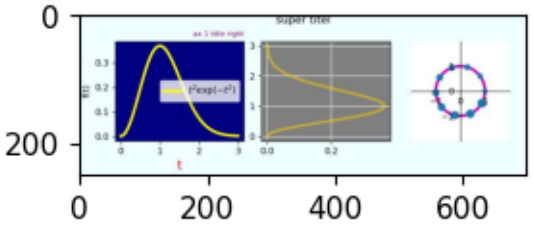
[16]:
```
array([[ 1, -1,  1, -1,  1],
       [-1,  1, -1,  1, -1],
       [ 1, -1,  1, -1,  1],
       [-1,  1, -1,  1, -1],
       [ 1, -1,  1, -1,  1]])
```

[ ]:
```
plt.close('all')
```

[17]:
```
from matplotlib.image import imread

img = imread('bild1.jpg')
img.shape
```

[17]:
```
(250, 700, 3)
```

[19]:
```
plt.figure()
plt.imshow(img);
```



contour berechnet und zeichnet Höhenlinien auf einem kartesichen Gitter

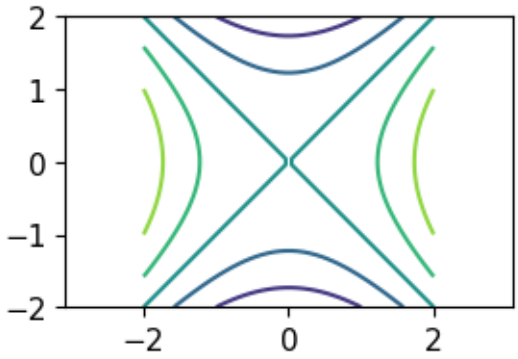
```
contour(x, y, z, levels)
```

x, y : Arrays mit den Koordinaten für den Wert z

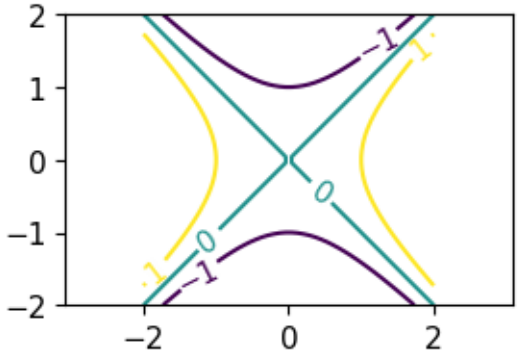
levels : Anzahl der Höhenlinine oder Werte für die Höhenlinien

[20]:
```
x = np.linspace(-2, 2)
y = np.linspace(-2, 2)
xx, yy = np.meshgrid(x, y)
zz = xx**2 - yy**2
```

[21]:
```
fig, ax = plt.subplots(1)
cs = ax.contour(x, y, zz, 5) # 5 Höhenlinien ausgewählt von matplotlib
#cs.clabel()
ax.axis('equal');
```



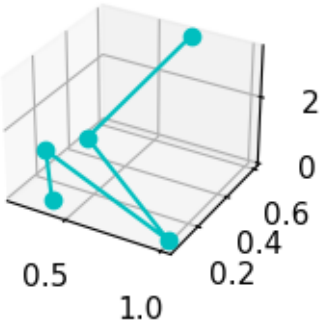
[23]:
```
fig, ax = plt.subplots(1)
cs = ax.contour(x, y, zz, [-1, 0, 1]) # 3 Höhenlinien zu Werten -1, 0 und 1
cs.clabel()
ax.axis('equal');
```



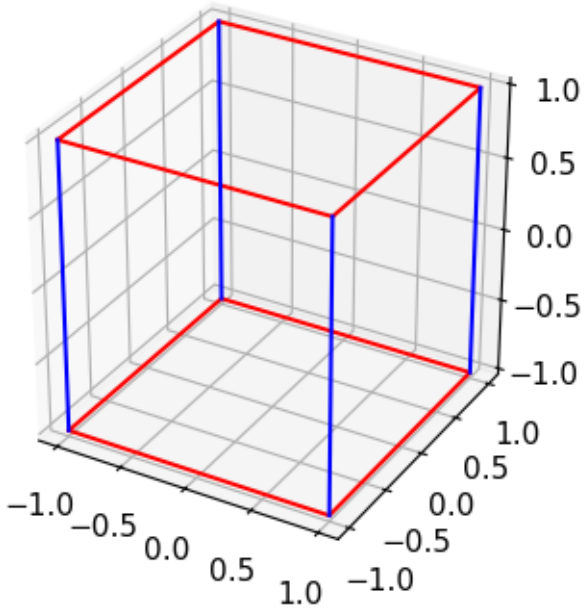
2 3D Plots

2.1 Polygonzug in 3D

[24]:
```
# erzeugen ein 3D Koordinatensystem
ax = plt.figure().add_subplot(projection='3d')

x = np.random.rand(5)
y = np.random.rand(5)
z = np.random.randn(5)

ax.plot(x, y, z, 'co-');
```



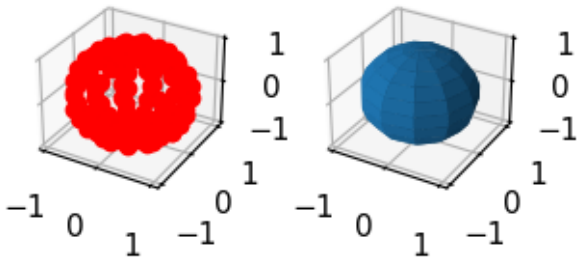
[25]:
```
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(projection='3d')

x, y = [-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1]
for z in (-1, 1):                          # Zwei Quadrate
    ax.plot(x, y, [z] * 5, color="r")
for a, b in zip(x[:4], y[:4]):             # verbinde die vier Ecken der Quadrate
    ax.plot([a, a], [b, b], [-1, 1], color="b")

ax.set_box_aspect((1, 1, 1))
```



2.2 Fläche in 3d mit plot_surface

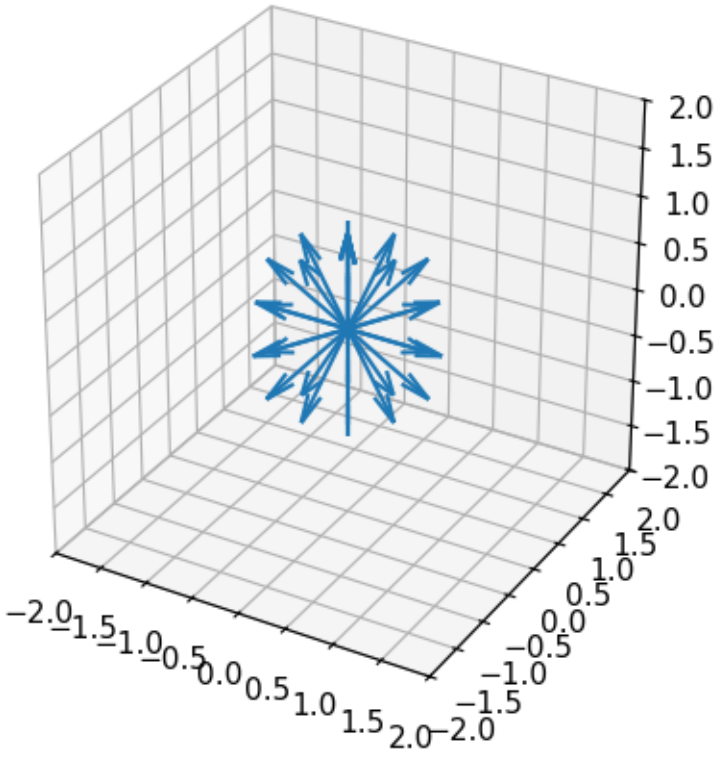
[26]:
```
fig, ax = plt.subplots(1,
                       2,
                       layout='constrained',
                       subplot_kw={'projection': '3d'})

phi = np.linspace(0, 2 * np.pi, 12).reshape(-1, 1)
theta = np.linspace(0, np.pi, 12).reshape(1, -1)
theta = np.append(theta, np.nan) #nan Trick um ungewünschte Linien zu entfernen

# Parametrisierung der Kugelfläche siehe https://de.wikipedia.org/wiki/Kugel
x = np.cos(phi) * np.sin(theta)  # durch broadcasting sind x, y, z hier 2Darrays
y = np.sin(phi) * np.sin(theta)
z = (phi * 0 + 1) * np.cos(theta)

ax[0].plot(x.flatten(), y.flatten(), z.flatten(), 'ro-', linewidth=1)
ax[1].plot_surface(x, y, z)
```

[26]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7a529e241eb0>
```



2.3 Vektorpfeile in 3d mit quiver

[28]:
```
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(projection='3d')

phi = np.linspace(0, 2 * np.pi, 6, endpoint=False).reshape(-1, 1)
theta = np.linspace(0, np.pi, 6, endpoint=False).reshape(1, -1)

# durch broadcasting sind u, v, w hier 2D arrays
u = np.cos(phi) * np.sin(theta)
v = np.sin(phi) * np.sin(theta)
w = np.ones_like(phi) * np.cos(theta)

ax.quiver(0, 0, 0, u, v, w)
ax.set_box_aspect((1, 1, 1))

ax.set_xlim(-2, 2)
ax.set_ylim(-2, 2)
ax.set_zlim(-2, 2);
```



[ ]:
```
plt.close('all')
```