lektion5

November 12, 2025

Inhalt

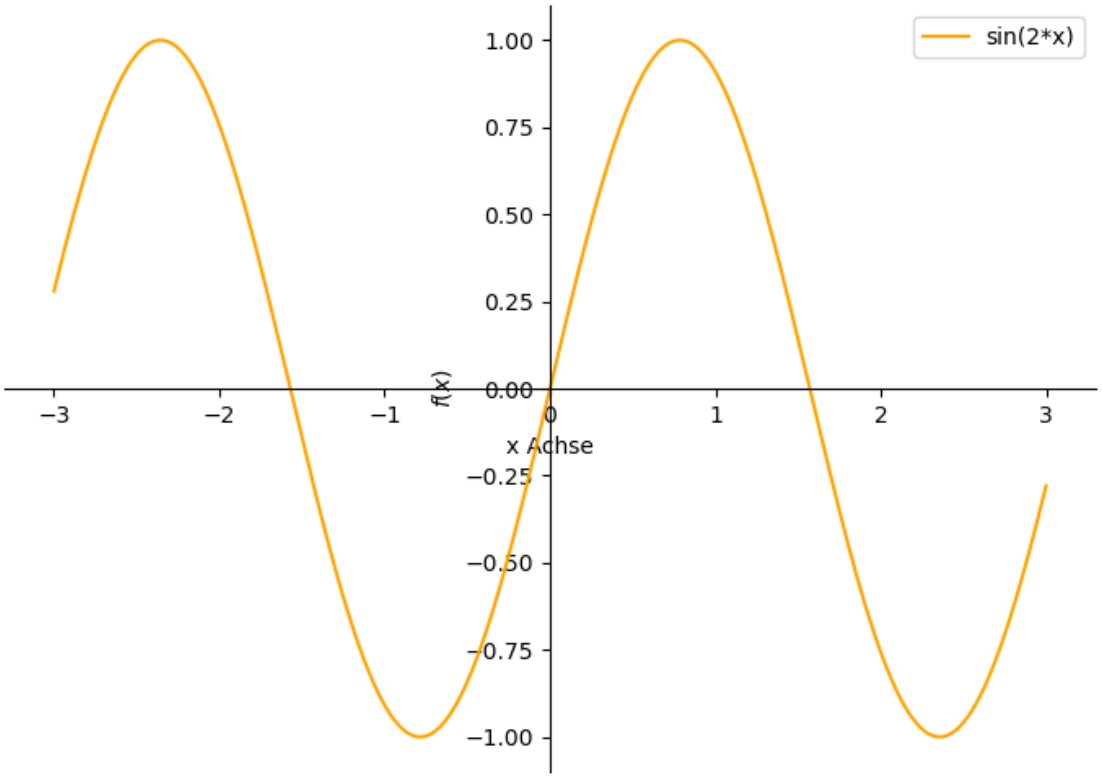
1 Lektion 5

1.1 Wiederholung

[1]:
```
%matplotlib inline

import matplotlib  # das ist nur fuer die Vorlesung
matplotlib.rcParams['figure.figsize'] = (7, 5)

import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

sp.init_printing()
```

[2]:
```
x = sp.symbols('x')
f = sp.sin(2*x)

p1 = sp.plot(f, (x, -3, 3), show=False)
```

```
p1[0].line_color = 'orange'   # Attribut des Graphen
p1.legend = True              # Attribut der Bildes/Koordinatensystems
p1.xlabel = 'x Achse'
p1.show()                     # Methode des Bildes
```



1.2 Lambdifizierung

numpy Fuktionen können auf np.arrays ausgewertet werden und sind schnell.

[3]:
```
fn = sp.lambdify(x, f)
gn = lambda x: 5*np.cos(x)
gn(2.)
```

[3]:
-2.08073418273571

[4]:
```
xn = np.linspace(-np.pi, np.pi, 10) # 10 äquidistante Punkte in [-pi,pi]
xn
```

[4]:
```
array([-3.14159265, -2.44346095, -1.74532925, -1.04719755, -0.34906585,
        0.34906585,  1.04719755,  1.74532925,  2.44346095,  3.14159265])
```

[5]:
```
fn(xn)
```

[5]:
```
array([ 2.44929360e-16,  9.84807753e-01,  3.42020143e-01, -8.66025404e-01,
       -6.42787610e-01,  6.42787610e-01,  8.66025404e-01, -3.42020143e-01,
       -9.84807753e-01, -2.44929360e-16])
```

[6]:
```
gn(xn)
```

[6]:
```
array([-5.        , -3.83022222, -0.86824089,  2.5       ,  4.6984631 ,
        4.6984631 ,  2.5       , -0.86824089, -3.83022222, -5.        ])
```

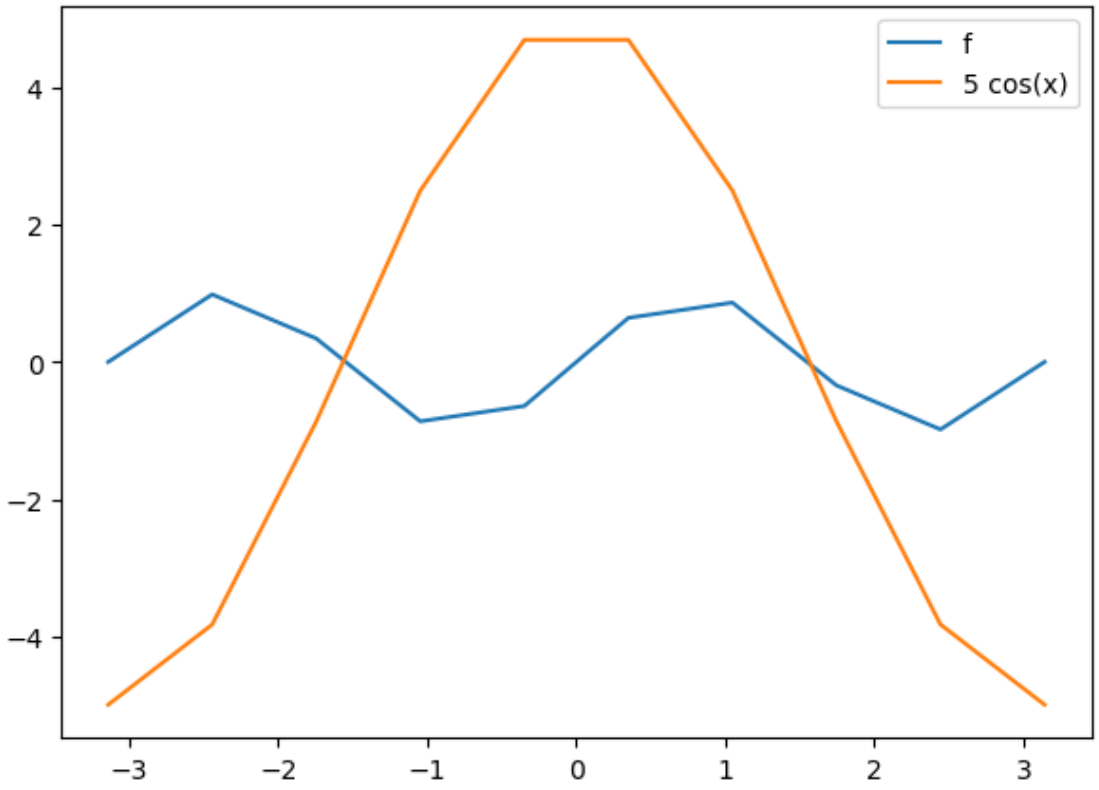
1.3 Matplotlib Graphik 2D

1.3.1 Graphen von Funktionen (plot)

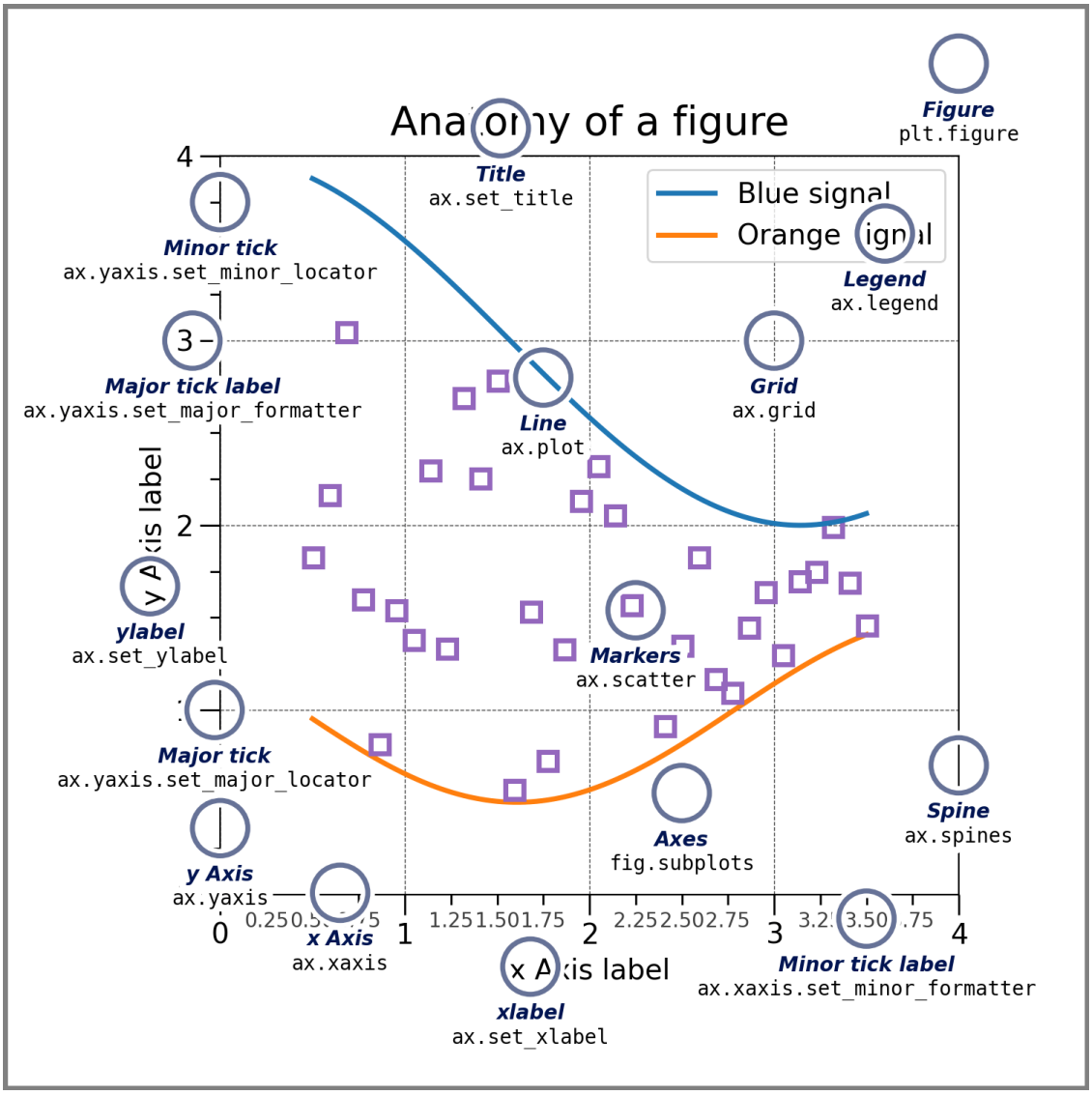
[7]:
```
#%matplotlib qt
plt.figure('mein erstes Bild');   # erzeugt ein Bild/Fenster mit qt backend
```

```
<Figure size 700x500 with 0 Axes>
```

[8]:
```
fig, ax = plt.subplots()  # Bild 'fig' mit Koordinatensystem 'ax'
ax.plot(xn, fn(xn), label='f')
#  zeichnet in das 'ax' Koordinatensystem
ax.plot(xn, gn(xn), label='5 cos(x)')
ax.legend();
# fügt den Inhalt der labels als Legende hinzu
```



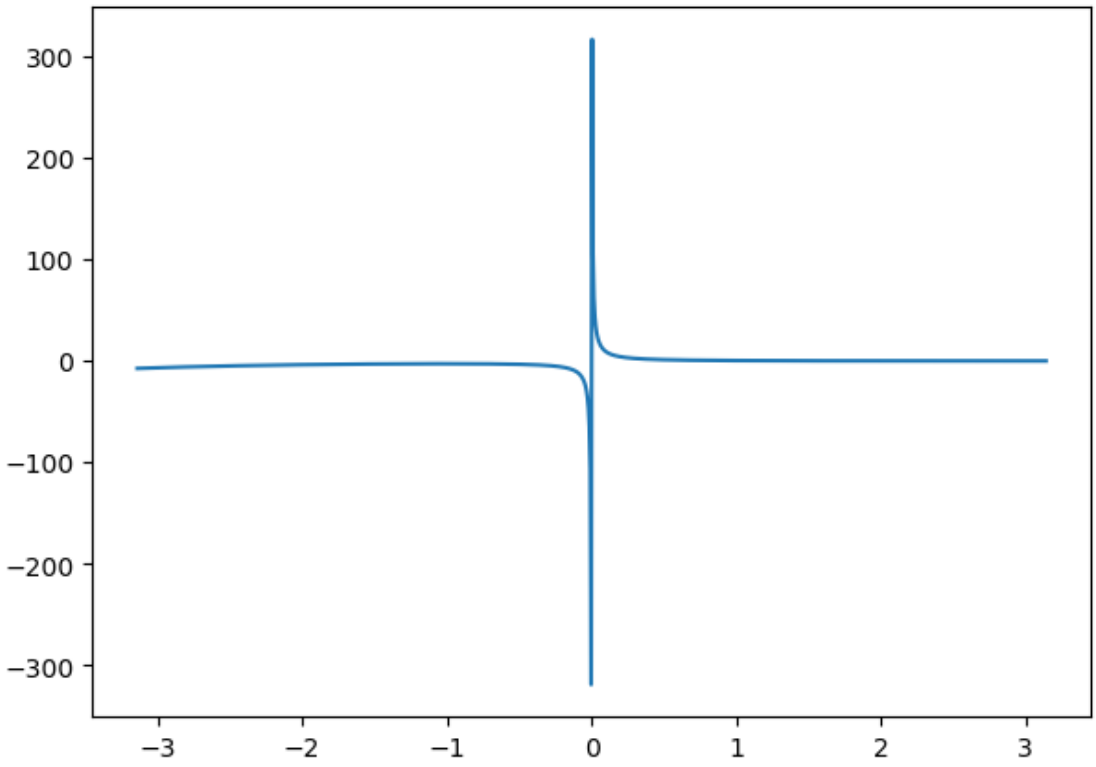
Von https://matplotlib.org/stable/users/explain/quick_start.html



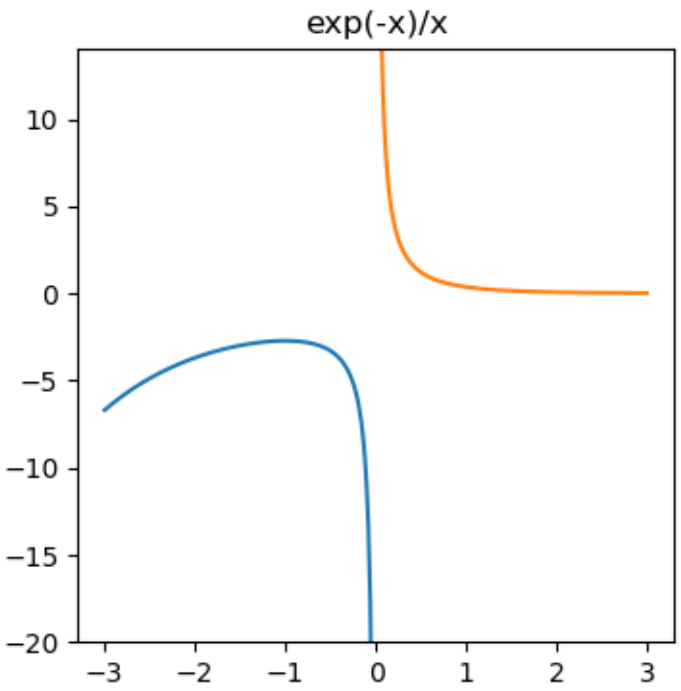
[9]:
```
x = sp.Symbol('x')
h = sp.exp(-x)/x
xn = np.linspace(-np.pi, np.pi, 1000)
hn = sp.lambdify(x, h)
```

[10]:
```
fig, ax = plt.subplots()
ax.plot(xn, hn(xn))
```

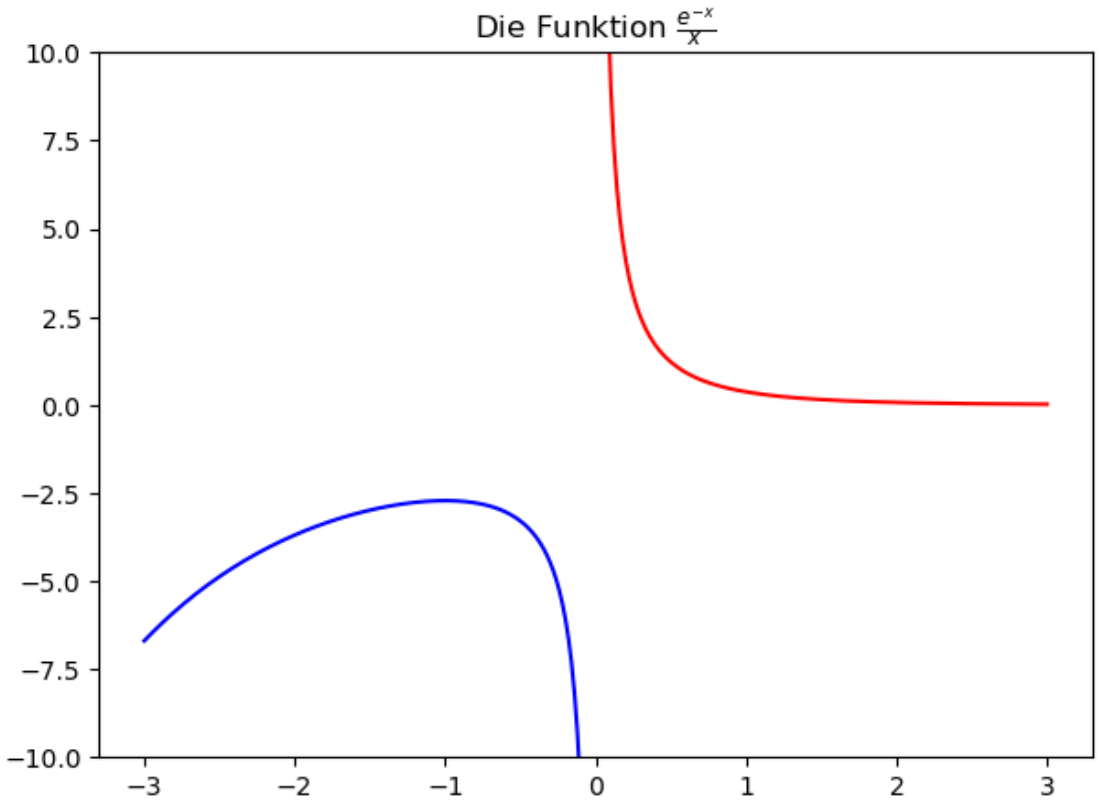
[10]:
```
[<matplotlib.lines.Line2D at 0x7ff59ad29d30>]
```



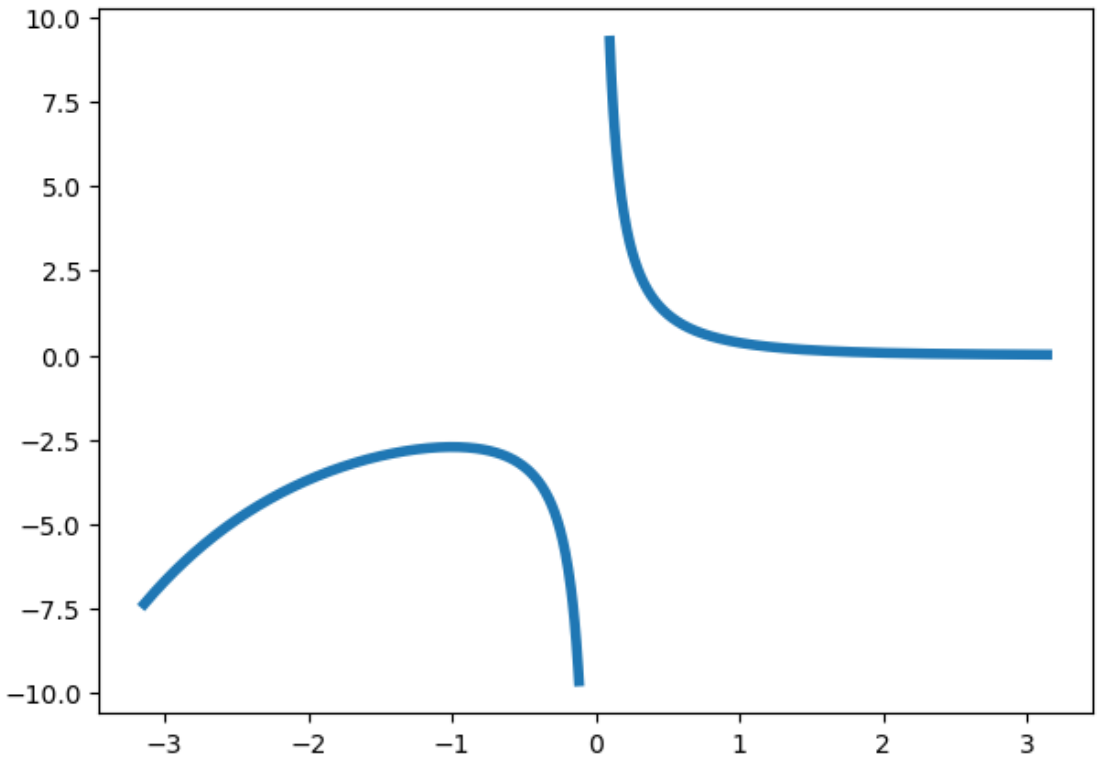
[11]:
```
xm = np.linspace(-3, -0.0001, 150)
xp = -xm
fig, ax = plt.subplots(figsize=(4, 4))
ax.plot(xm, hn(xm))
ax.plot(xp, hn(xp))
ax.axis(ymin=-20, ymax=14)
ax.set_title(h);
```



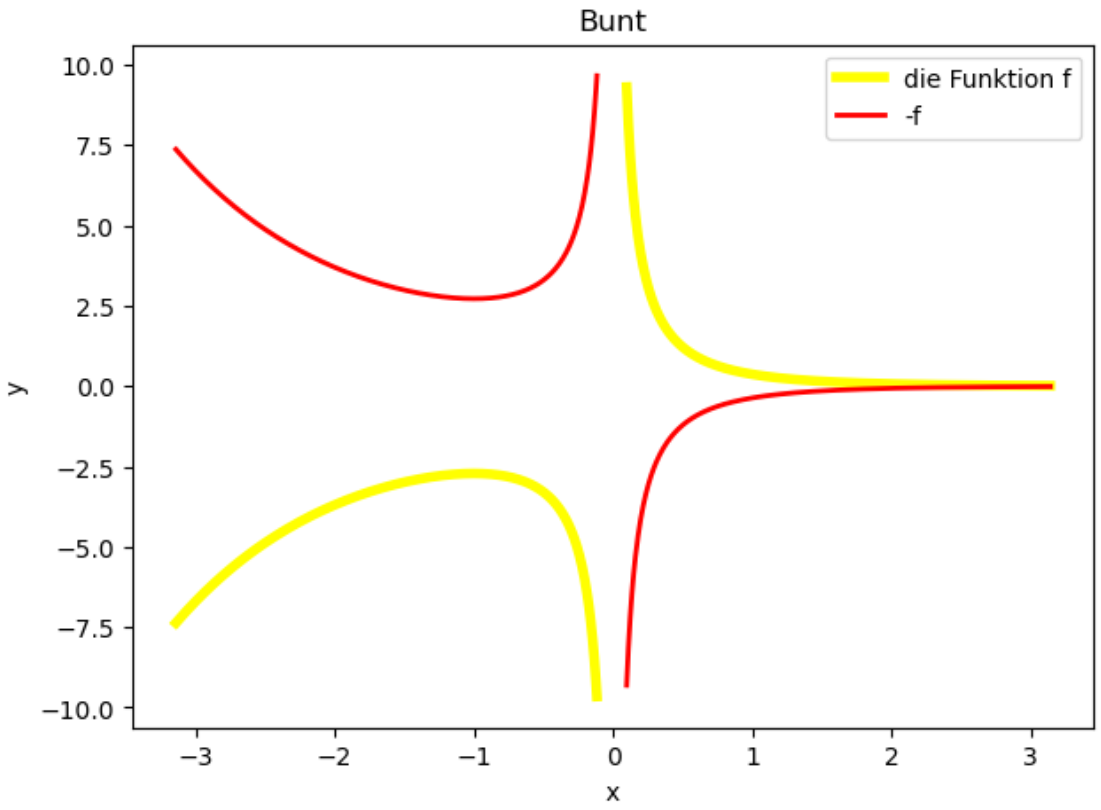
[12]:
```
xm = np.linspace(-3, -0.0001, 150)
xp = -xm
fig2, ax2 = plt.subplots()
ax2.plot(xm, hn(xm), 'b')
ax2.plot(xp, hn(xp), 'r')
ax2.axis(ymin=-10, ymax=10)
txt = sp.latex(h)
ax2.set_title(f"Die Funktion ${txt}$");
```



[13]:
```
fig, ax = plt.subplots()
yn = hn(xn)
yn[abs(yn) > 10] = np.nan  # 'not a number' Trick
ax.plot(xn, yn, linewidth=4);
```

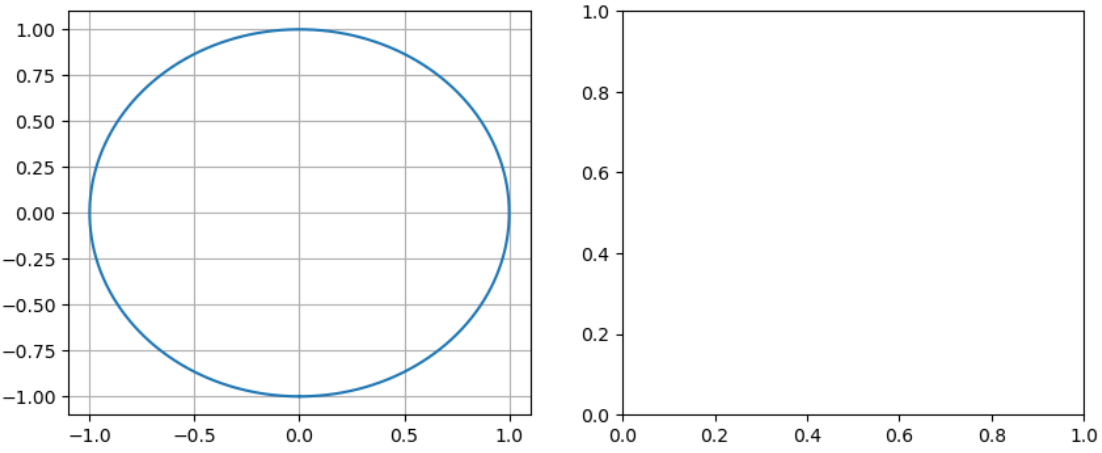


[14]:
```
fig, ax = plt.subplots()
ax.plot(xn, yn, color='yellow', linewidth=4,
        label='die Funktion f')  # plot in das aktuelle Koordinatensystem
ax.plot(xn, -yn, color='red', linewidth=2, label='-f')
ax.set_ylabel('y')
ax.set_xlabel('x')
ax.set_title('Bunt')
ax.legend();
```



1.3.2 Parametrische Kurven (plot)

[15]:
```
#%matplotlib qt
fig, axs = plt.subplots(1,2, figsize=(10,4))
axs[0].plot(np.sin(xn), np.cos(xn))
axs[0].grid()
```

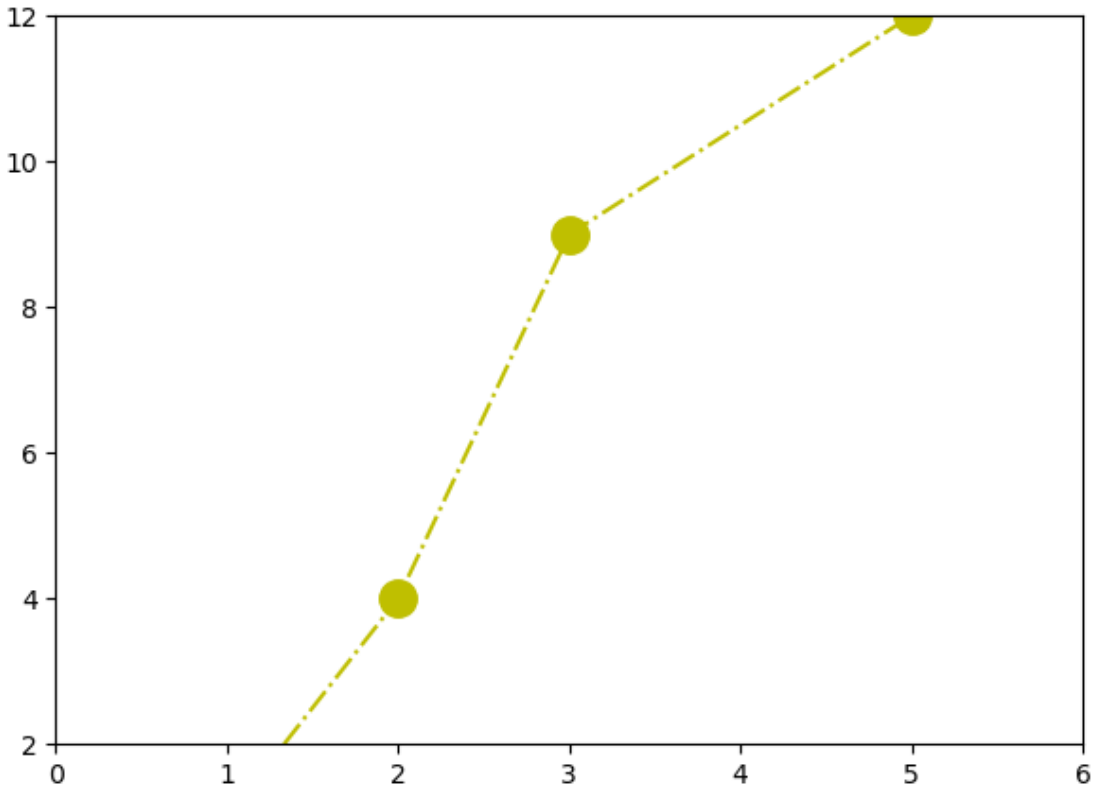


[16]:
```
axs[1].plot(np.sin(xn), np.cos(xn), 'rd', markersize=2)
axs[1].set_aspect('equal')
```

[17]:
```
axs[1].cla()    # clear axis
```

[18]:
```
fig.clf()  # clear figure
```

[19]:
```
fig = plt.figure()
ax = fig.gca()
ax.plot([1, 2, 3, 5], [1, 4, 9, 12],'yo-.', markersize=14)
# Farbe: r, g, b, c, m  ... rot, gruen, blau, cyan, magenta
# Marker: x, o, +, *, ....
# Linienstil -, :,  --, -.
ax.axis([0, 6, 2, 12]);
```



[20]:
```
?ax.plot
```

1.3.3 Implizit gegebene Kurven / Höhenlinien (contour)

Punkte (x, y) in der Ebene, sodass $f(x, y) = const$.

[21]:
```
def f(x, y):
    return x**2 + y**2 - 1
```

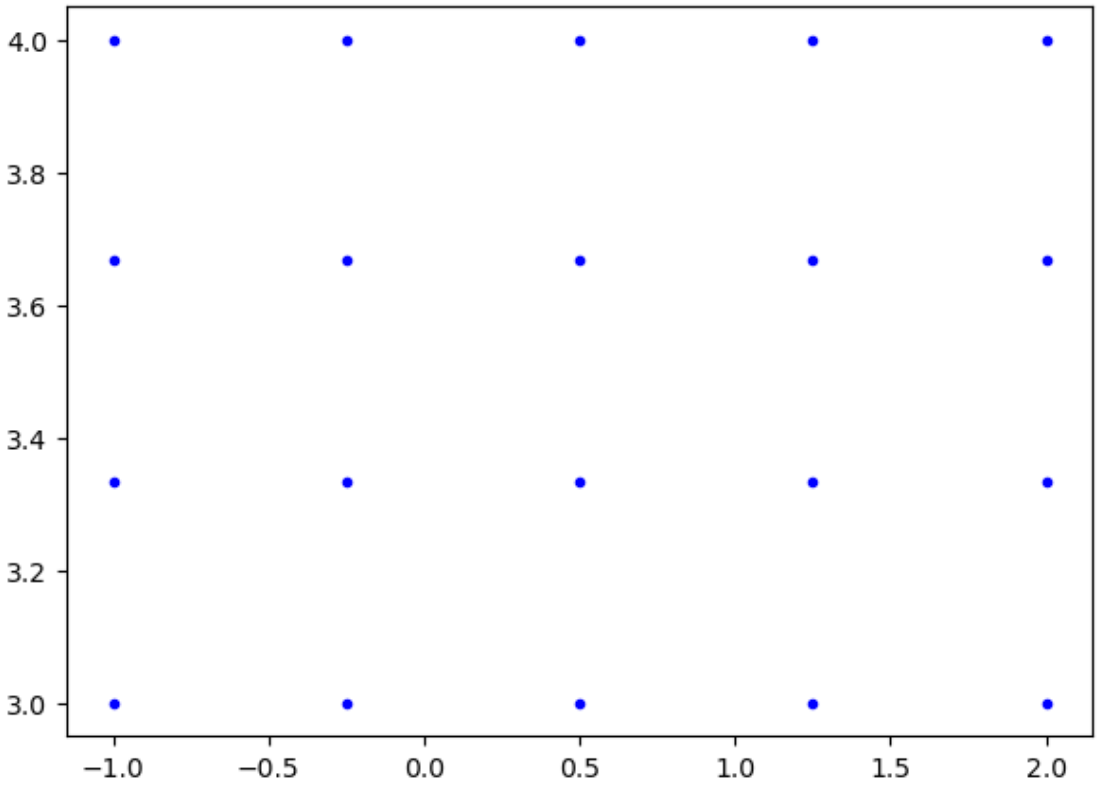
[22]:
```
xn = np.linspace(-2, 2, 100)
yn = np.linspace(-1.5, 2, 50)
X, Y = np.meshgrid(xn, yn) # x,y Koordinaten, der Gitterpunkte eines␣
 ↪kartesischen Gitters
Z = f(X, Y)                # Auswertung von f auf den Gitterpunkten
```

[23]:
```
# linspace meshgrid
a = np.linspace(-1, 2, 5)
b = np.linspace(3, 4, 4)
A, B = np.meshgrid(a, b)
A, B
```

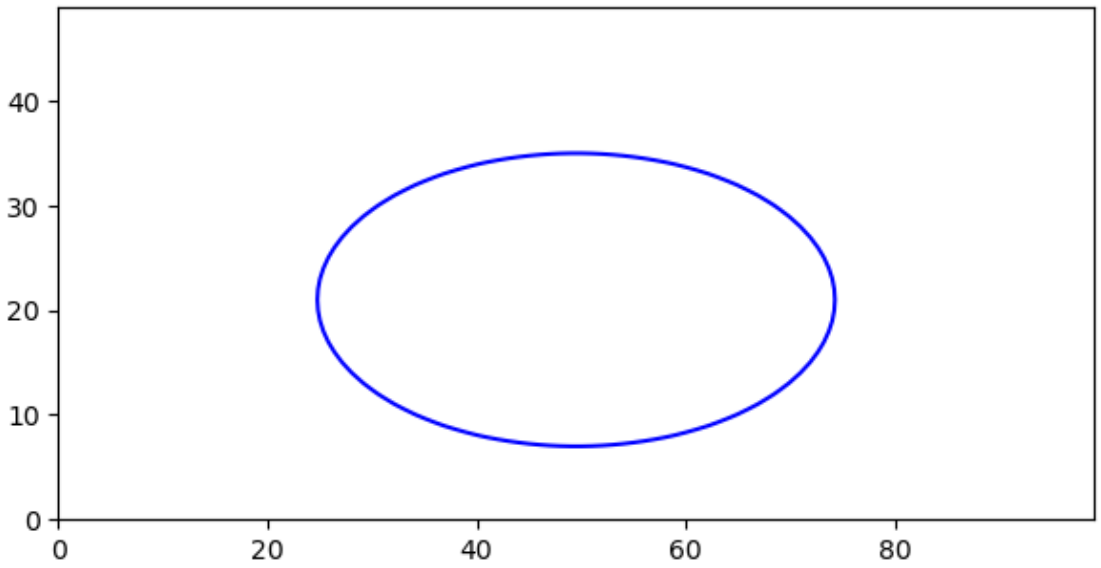
[23]:
```
(array([[-1.  , -0.25,  0.5 ,  1.25,  2.  ],
        [-1.  , -0.25,  0.5 ,  1.25,  2.  ],
        [-1.  , -0.25,  0.5 ,  1.25,  2.  ],
        [-1.  , -0.25,  0.5 ,  1.25,  2.  ]]),
 array([[3.        , 3.        , 3.        , 3.        , 3.        ],
        [3.33333333, 3.33333333, 3.33333333, 3.33333333, 3.33333333],
        [3.66666667, 3.66666667, 3.66666667, 3.66666667, 3.66666667],
        [4.        , 4.        , 4.        , 4.        , 4.        ]]))
```

[24]:
```
plt.figure()
plt.plot(A, B, 'b.')
```

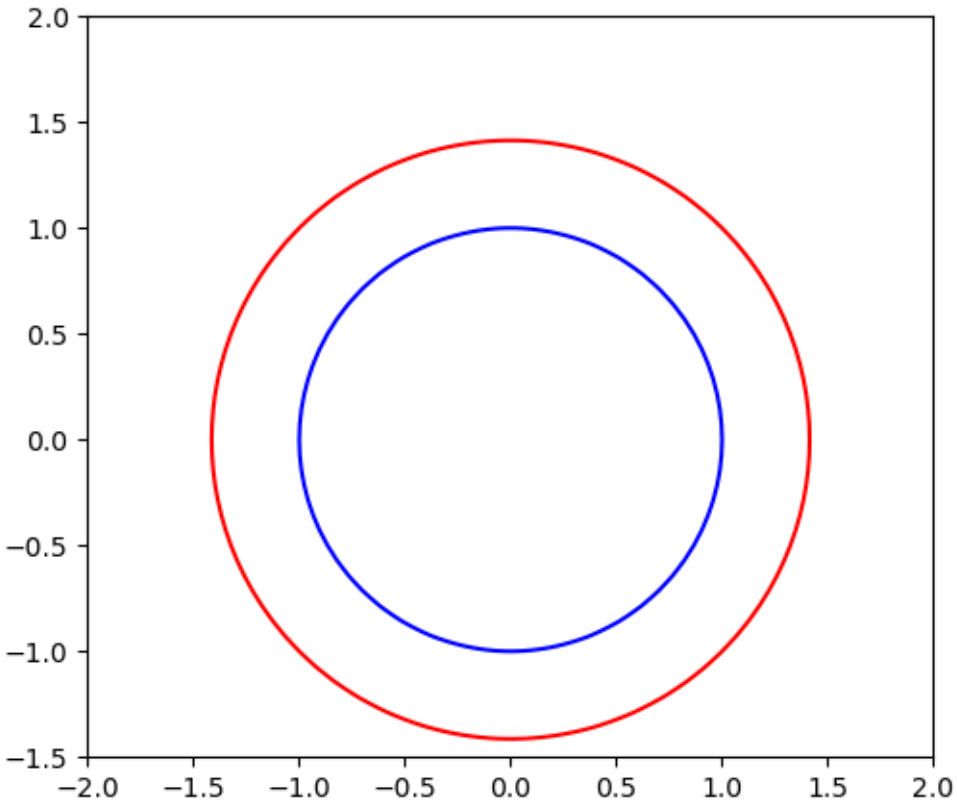
[24]:
```
[<matplotlib.lines.Line2D at 0x7ff5926c7740>,
 <matplotlib.lines.Line2D at 0x7ff5926d8770>,
 <matplotlib.lines.Line2D at 0x7ff5926c7830>,
 <matplotlib.lines.Line2D at 0x7ff5926c78f0>,
 <matplotlib.lines.Line2D at 0x7ff5926c79b0>]
```



[25]:
```
fig = plt.figure()
ax = fig.gca()
ax.contour(Z, [0], colors='blue')  # 0 Höhenlinie in der Farbe blau
ax.set_aspect('equal')
```



[26]:
```
# besser
fig = plt.figure()
ax = fig.gca()
ax.contour(xn, yn, Z, [0, 1], colors=['blue', 'red'])
# ax.contour(X, X, Z, [0, 1], colors=['blue', 'red']) # alternativ
# 0 und 1 Höhenline in blau und rot, jetzt mit x,y Koordinaten
ax.set_aspect('equal')
```

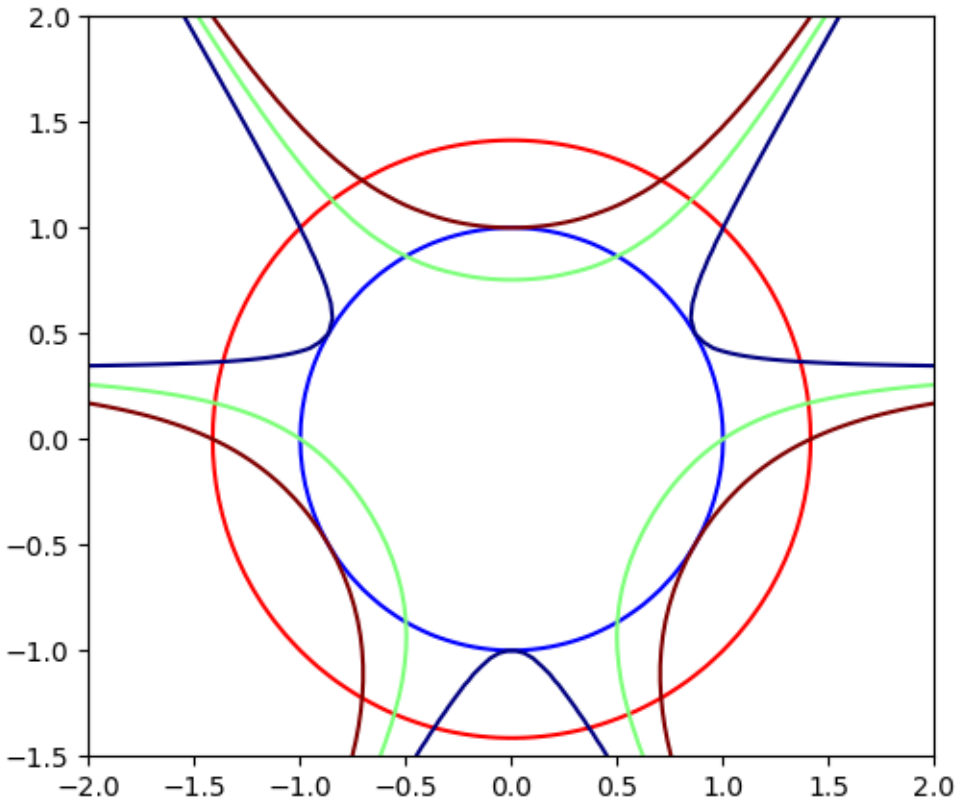


[27]:
```
def g(x, y):
    return x**2+y**2-3*x**2*y+y**3
```

[28]:
```
G = g(X, Y)
```

[29]:
```
fig = plt.figure()
ax = fig.gca()
ax.contour(X, Y, Z, [0, 1], colors=[
    'blue', 'red'])  # 0 und 1 Höhenline in blau und rot, jetzt mit x,y␣
 ↪Koordinaten
```

```
ax.contour( X, Y, G, [0, 1, 2],
    cmap=plt.cm.jet)  # 0, 1 und 2 Höhenlinie mit Farben aus colormap jet
ax.set_aspect('equal')
```

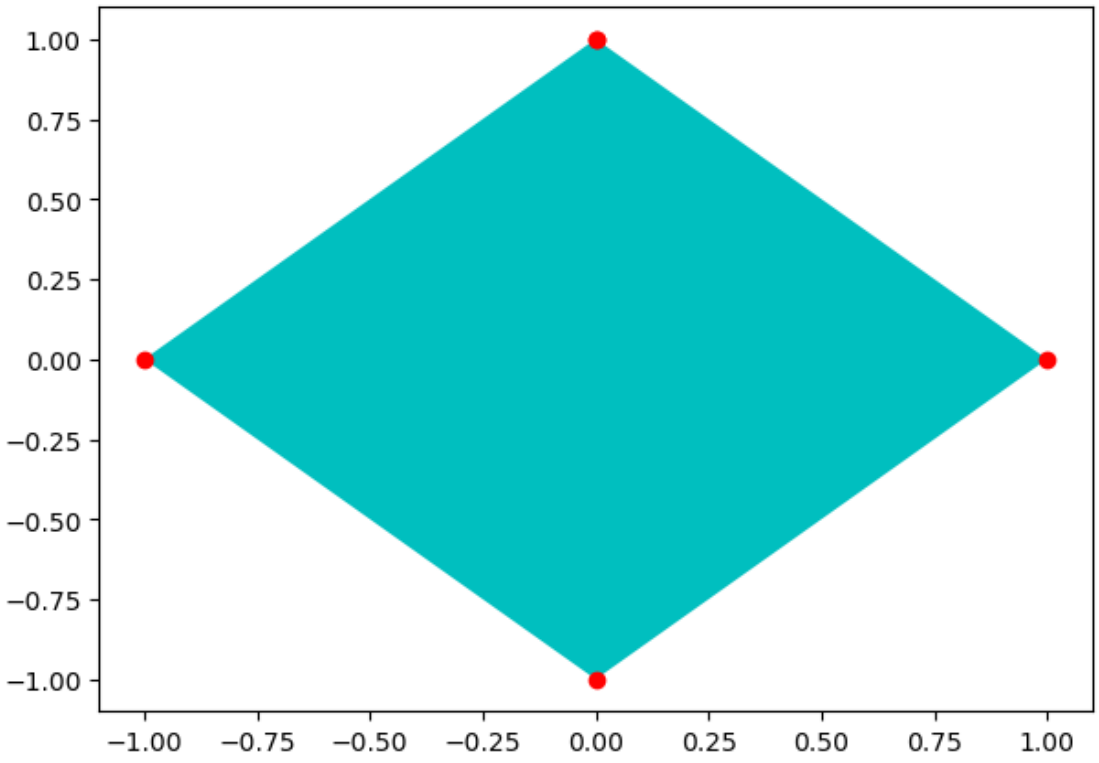


Alle Colormaps findet man auf https://matplotlib.org/stable/gallery/color/colormap_reference.html

1.3.4 Ausgefüllte Flächen

[30]:
```
t = np.linspace(0, 2 * np.pi, 5)
x = np.sin(t)
y = np.cos(t)
```

[31]:
```
fig = plt.figure()
ax = fig.gca()
ax.fill(x, y, 'c')  # ausgefülltes Polygon mit Eckpunten *x, *y
ax.plot(x, y, 'ro');
```



Eine komplexe Zahl $c = x + iy$ gehört zur Mandelbrotmenge mit Schranke s, s eine natürliche Zahl, falls die rekursiv definierte Folge

$$z_{i+1}(c) = z_i^2(c) + c \quad z_0 = 0$$

innerhalb der ersten s Elemente kein Folgenglied enthält für das $|z_i| > 2$ gilt. Man kann zusätzlich den kleinsten Index n, $n < s$, ansehen, bei dem $|z_n| > 2$ zuerst erreicht wird.

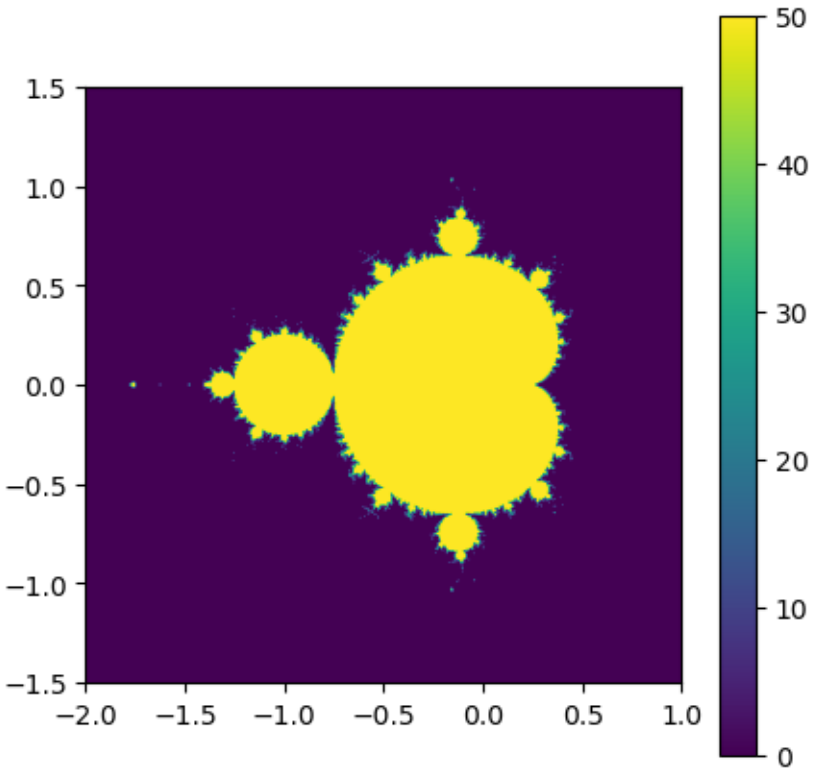
[32]:
```
def mandelbrot(c, s):
    z = 0
    for n in range(s):
        z = z**2 + c
        if abs(z) > 2:
            return n
    return s  # s wird zurückgegeben, wenn c in der Mandelbrotmenge mit Schranke␣
↪s enthalten ist.
```

[33]:
```
#%matplotlib qt
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

ppu = 256  # Anzahl Pixel pro Einheit
```

```
# Vektoren mit Real und Imaginaerteil von c
x_vec = np.linspace(-2, 1, 3 * ppu)
y_vec = np.linspace(-1.5, 1.5, 3 * ppu)

s = 50
M = np.zeros((len(x_vec), len(y_vec)))

for xi, x in enumerate(x_vec):
    for xj, y in enumerate(y_vec):
        M[xi, xj] = mandelbrot(x + y * 1j, s)
```

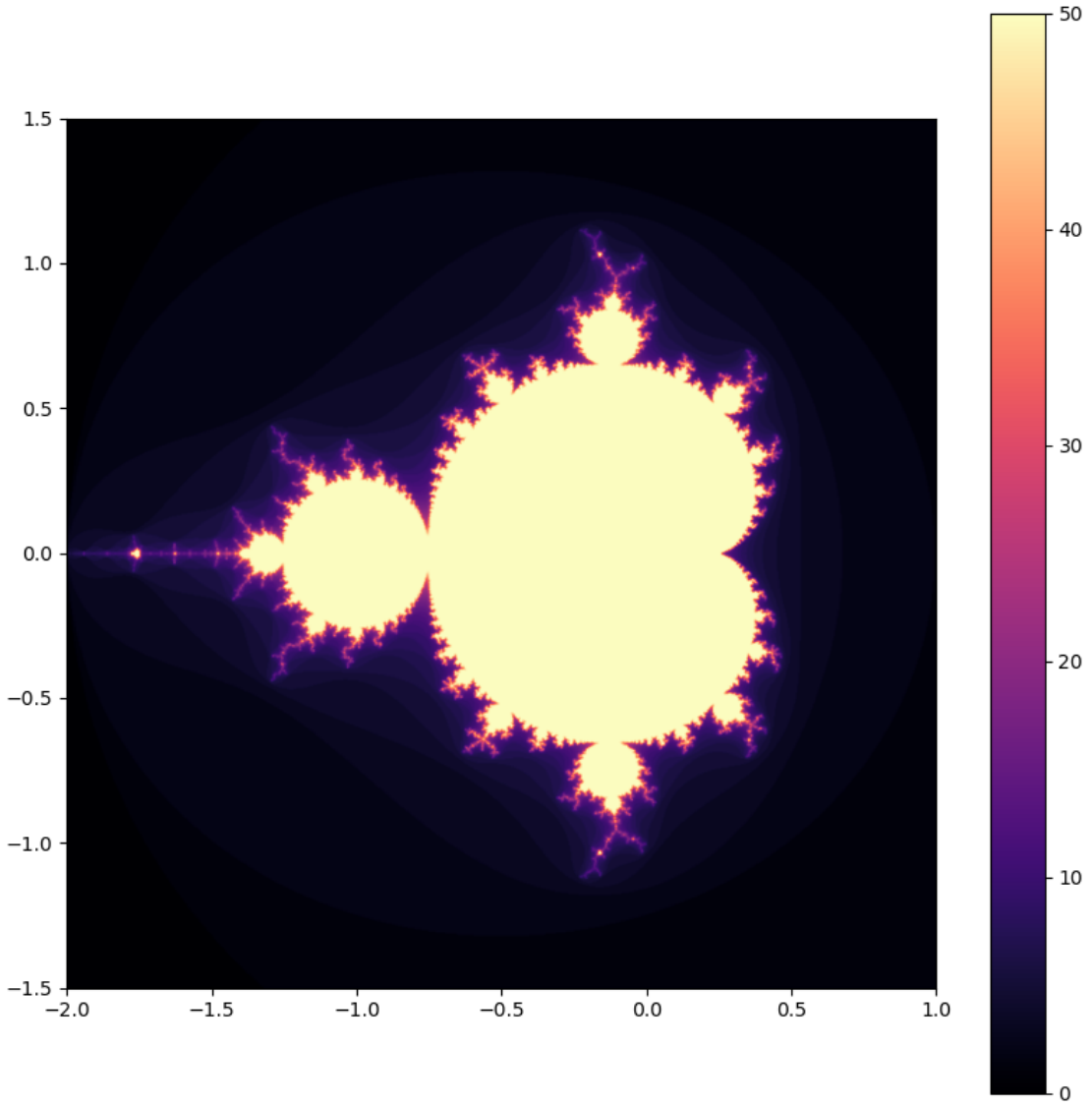
[34]:
```
fig, ax = plt.subplots(1, 1, figsize=(5, 5))
M_ = M.copy()
M_[M_ < s] = 0
# s sind die Werte in der Mandelbrotmenge und 0 sind die Werte ausserhalb
im = ax.imshow(M_.T, extent=[min(x_vec), max(x_vec), min(y_vec), max(y_vec)])
fig.colorbar(im)
```

[34]: <matplotlib.colorbar.Colorbar at 0x7ff5928f22d0>



[35]:
```
fig, ax = plt.subplots(figsize=(10, 10))  # instantiate a figure to draw
im = ax.imshow(M.T,
               interpolation="bicubic",
               cmap='magma',
               extent=[min(x_vec),
                       max(x_vec),
                       min(y_vec),
                       max(y_vec)])
fig.colorbar(im, ax=ax)
```

[35]:
```
<matplotlib.colorbar.Colorbar at 0x7ff5925d5820>
```



1.4 Matplotlib 3D Graphik

1.4.1 Graphen von Funktionen (plot_surface)

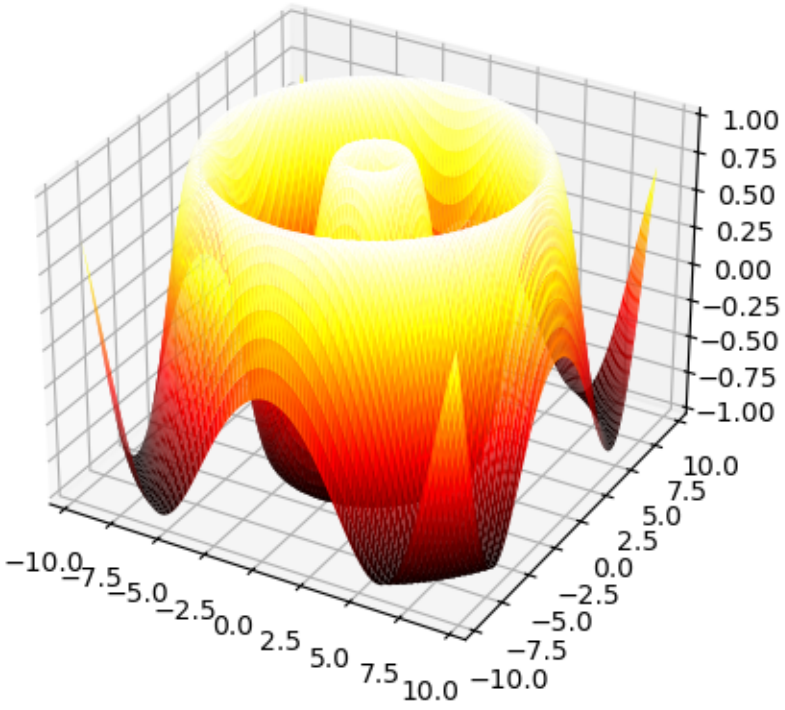
[40]:
```
%matplotlib inline
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
x, y = sp.symbols('x y')
f = sp.sin(sp.sqrt(x**2 + y**2))
f
```

[40]: $\sin\left(\sqrt{x^2 + y^2}\right)$

[41]:
```
fn = sp.lambdify((x, y), f)
```

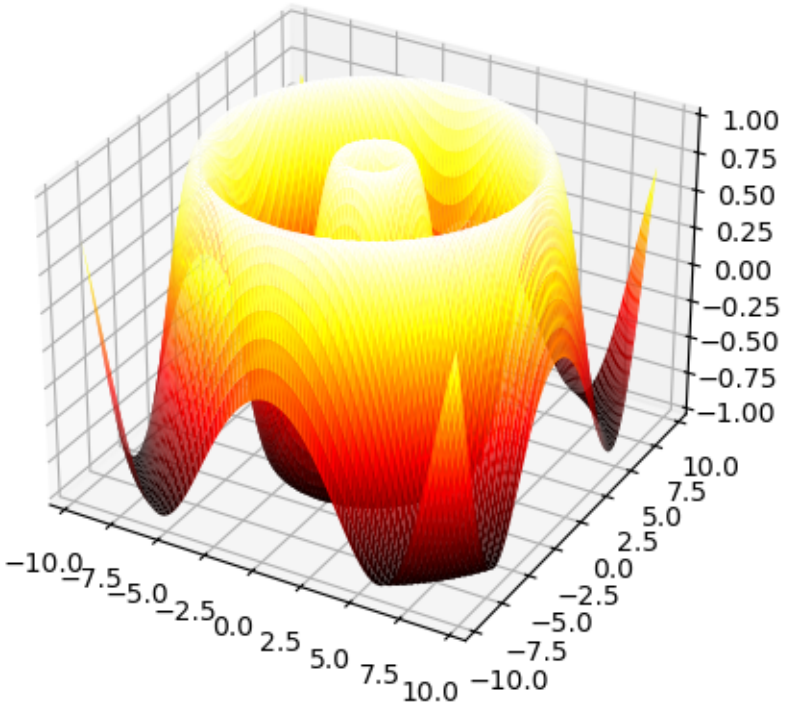
[42]:
```
xn = np.linspace(-3*np.pi, 3*np.pi, 100)
yn = np.linspace(-3*np.pi, 3*np.pi, 100)
X, Y = np.meshgrid(xn, yn)
```

[43]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
Z = fn(X, Y)
s1 = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                     cmap=plt.cm.hot, linewidth=0)
```

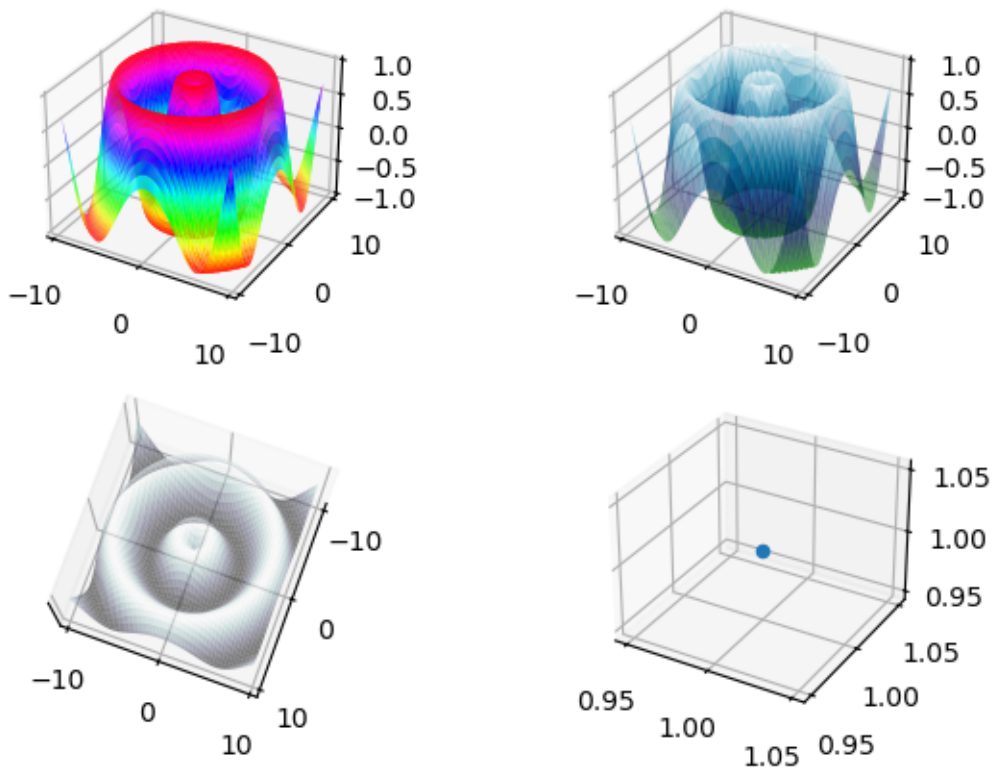


alternativ

[44]:
```
fig, ax = plt.subplots(1, 1, subplot_kw={'projection': '3d'})
Z = fn(X, Y)
s1 = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                     cmap=plt.cm.hot, linewidth=0)
```



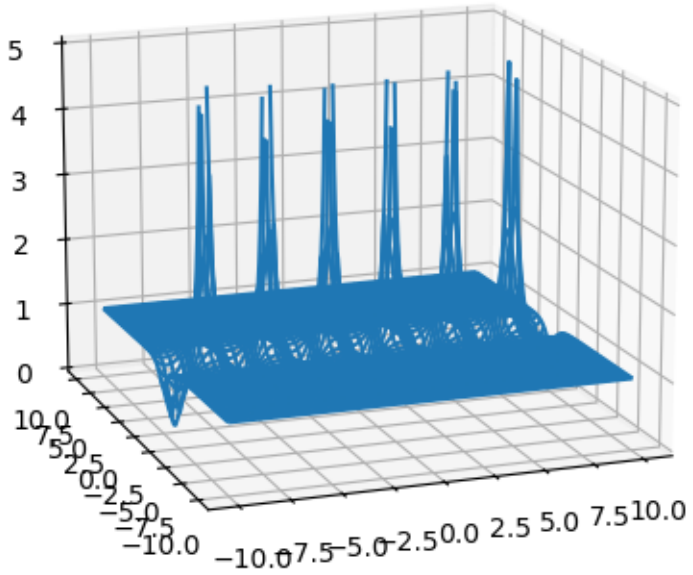
[47]:
```
fig, ax = plt.subplots(2, 2, subplot_kw={'projection': '3d'})
ax

ax[0, 0].plot_surface(X, Y, Z, rstride=1, cstride=1,
                      cmap=plt.cm.hsv, linewidth=1, alpha=1)
# alpha Transparenzwert (1 undurchsichtig)
ax[0, 1].plot_surface(X, Y, Z, cmap=plt.cm.ocean, linewidth=1, alpha=0.5)
ax[1, 0].plot_surface(X, Y, Z, cmap=plt.cm.bone, linewidth=1, alpha=0.5)
ax[1, 0].view_init(80, 20)  # Blickwickel auf das Bild im Koordinatensystem ax4
ax[1, 0].set_zticks([]);
ax[1, 1].scatter(1, 1 , 1, 'or');
```



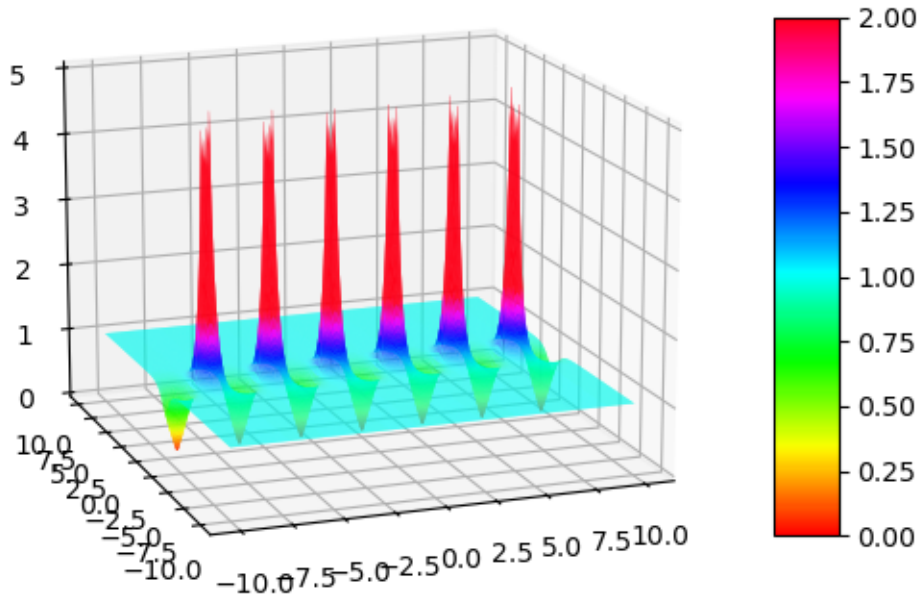
[48]:
```
f = abs(sp.tan(x+sp.I*y))    # Betrag des Tangens in der komplexen Ebene
fn = sp.lambdify((x, y), f)
```

[49]:
```
xn = np.linspace(-10, 10, 300)
yn = xn
X, Y = np.meshgrid(xn,  yn) # Gitter in der komplexen Ebene X: Realteil Y:␣
 ↪Imaginaerteil
Z = fn(X, Y)
Z[Z>5] = np.nan  # Werte größer als 5 werden auf nan (not a number) gesetzt und␣
 ↪ignoriert
```

[50]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z)
ax.view_init(15, -110)
```

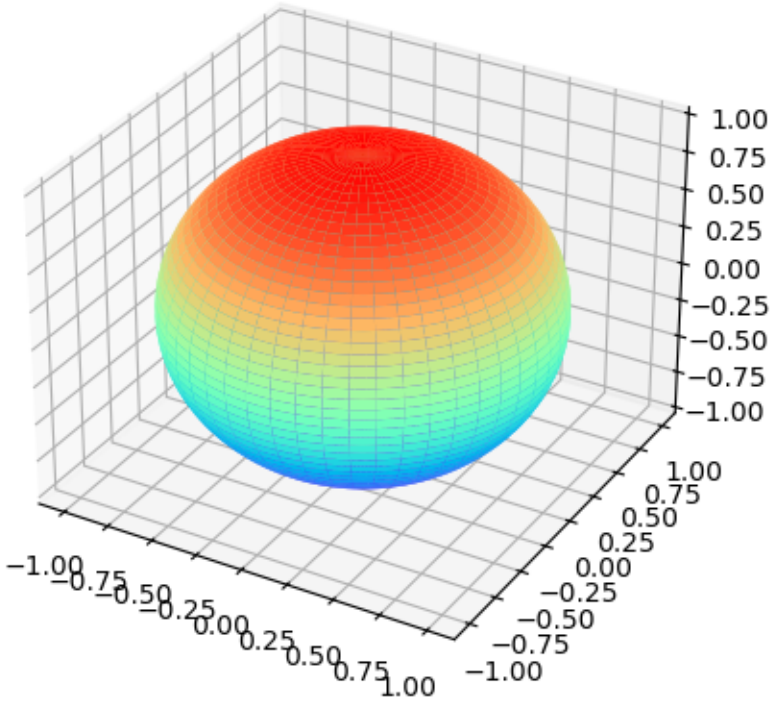


[51]:
```
from matplotlib.colors import Normalize
Normierung = Normalize(0, 2)
#Werte zwischen 0 und 2 sollen eingefärbt werden Werte größer 2 werden wie 2␣
 ↪eingefärbt

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.hsv, linewidth=0, rstride=1,␣
 ↪cstride=1, norm=Normierung)
ax.view_init(15, -110)

fig.colorbar(surf, shrink=0.7, aspect=8);
```



1.4.2 Parametrische Flächen (plot_surface)

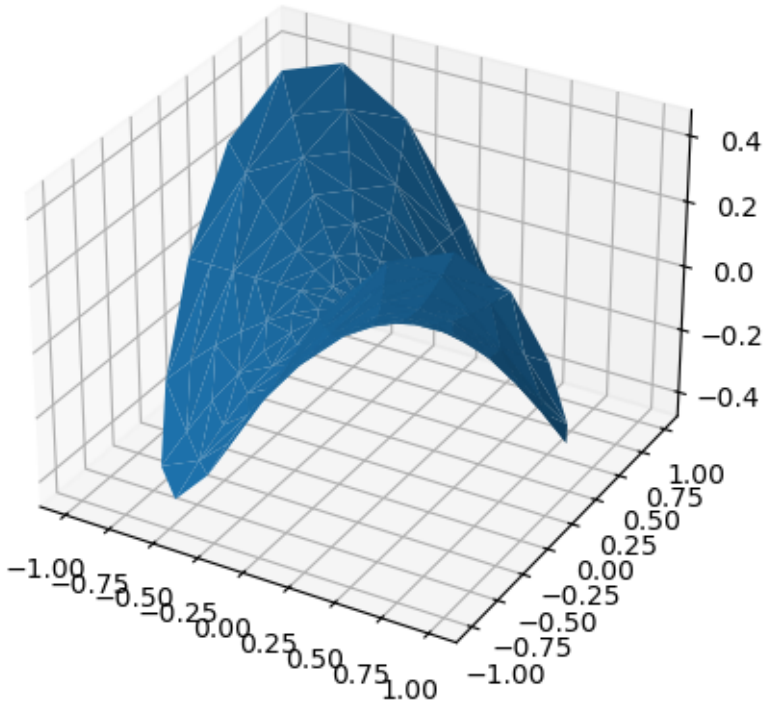
[52]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

az = np.linspace(-np.pi, np.pi, 100)
po = np.linspace(0, np.pi, 50)
AZ, PO = np.meshgrid(az, po)

X = np.sin(PO)*np.cos(AZ)
Y = np.sin(PO)*np.sin(AZ)
Z = np.cos(PO)

ax.plot_surface(X, Y, Z, cmap=plt.cm.rainbow);
```



[53]:
```
r_ = np.linspace(.1, 1, 8)
phi_ = np.linspace(0, 2 * np.pi, 20, endpoint=False)
r, phi = np.meshgrid(r_, phi_)
x = r * np.cos(phi)
y = r * np.sin(phi)
z = np.sin(-x * y)
```

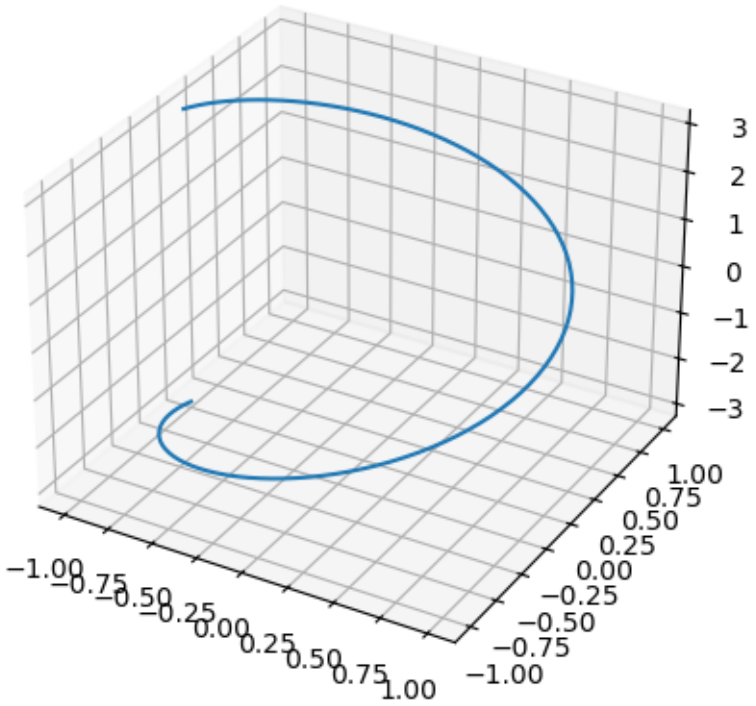
[54]:
```
ax = plt.figure().add_subplot(projection='3d')
ax.plot_trisurf(x.flatten(), y.flatten(), z.flatten(), linewidth=0.2)
```

[54]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7ff587b31220>
```



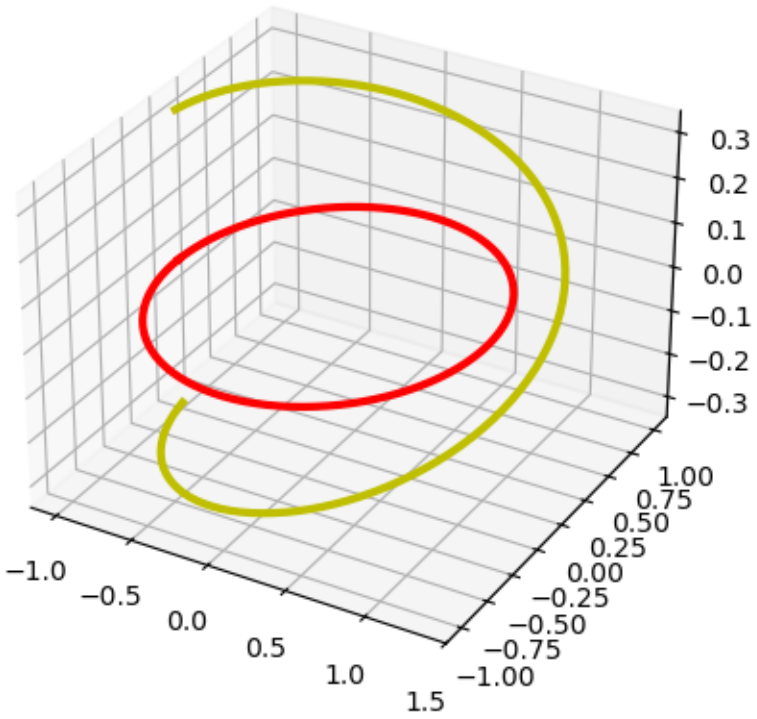
1.4.3 Raumkurven in Parameterdarstellung (plot)

[55]:
```
fig = plt.figure()
xn = np.linspace(-np.pi, np.pi, 100)
ax = fig.add_subplot(111, projection='3d')
ax.plot(np.cos(xn), np.sin(xn), xn);
```

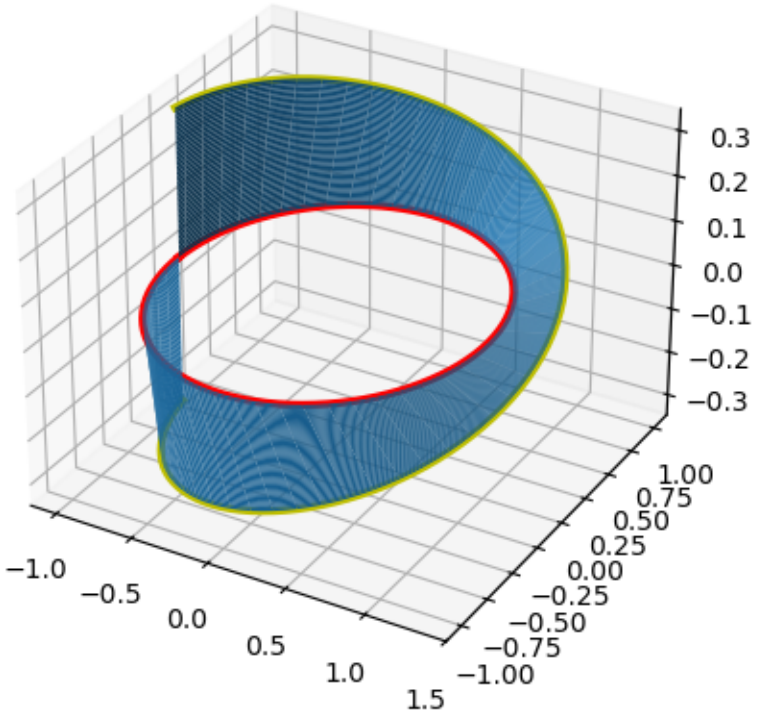


ein gewundenes Band

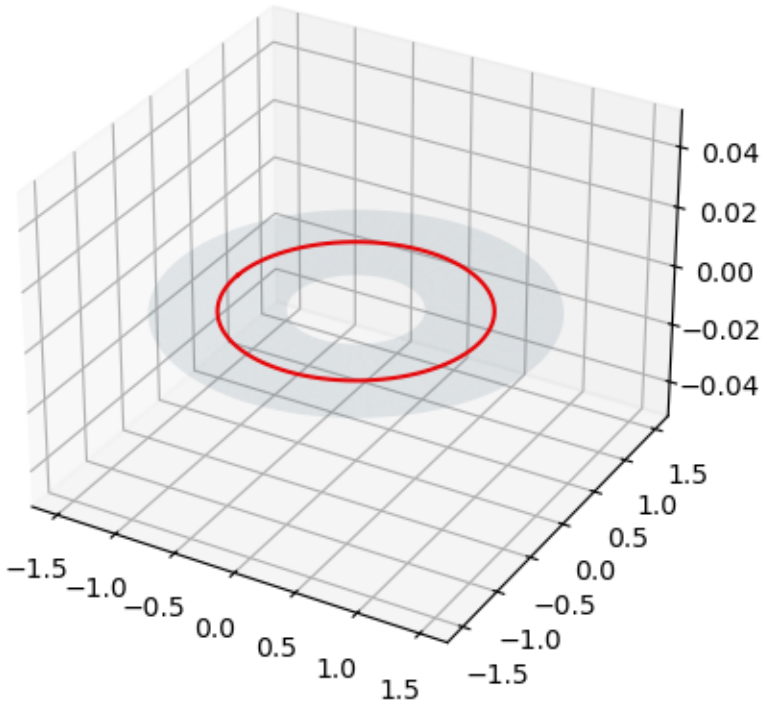
[56]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xn = np.linspace(-np.pi, np.pi, 100)
yn = np.linspace(0, 1, 50)
X, Y = np.meshgrid(xn, yn)

ax.plot(np.cos(xn), np.sin(xn), 0*xn, 'r', linewidth=3)
#ax.plot(np.cos(xn/2)/3, 0*xn, np.sin(xn/2)/3, 'b', linewidth=3)
ax.plot(np.cos(xn)+np.cos(xn/2)/3, np.sin(xn), np.sin(xn/2)/3, 'y', linewidth=3);
```



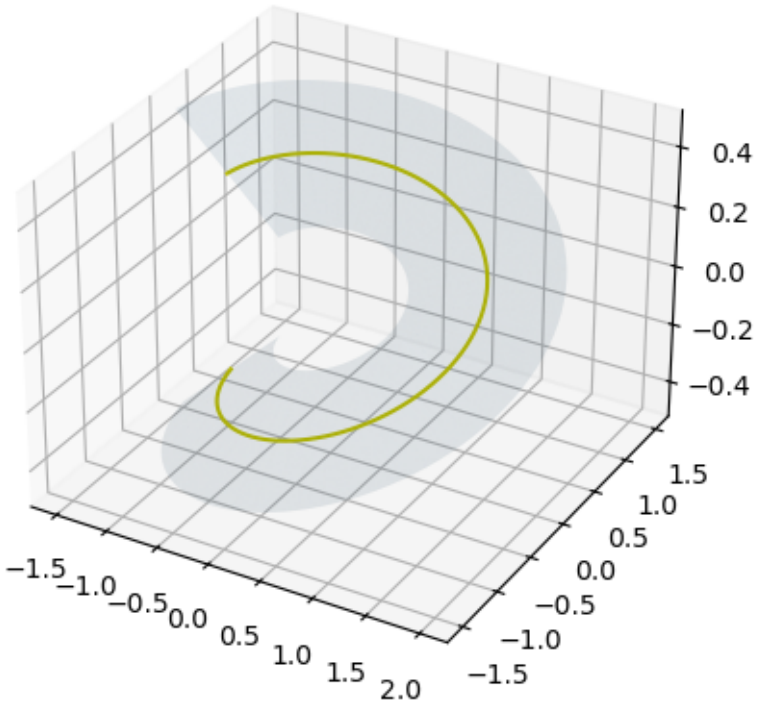
[57]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.plot(np.cos(xn), np.sin(xn), 0*xn, 'r', linewidth=3)
ax.plot(np.cos(xn)+np.cos(xn/2)/3, np.sin(xn), np.sin(xn/2)/3, 'y', linewidth=3)

ax.plot_surface(Y*np.cos(X) + (1-Y)*(np.cos(X)+np.cos(X/2)/3),\
        Y*np.sin(X) + (1-Y)*np.sin(X),\
        Y*0 + (1-Y)*np.sin(X/2)/3);
```



[58]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xn = np.linspace(-np.pi, np.pi, 100)
yn = np.linspace(0,1,50)
X, Y = np.meshgrid(xn, yn)

ax.plot_surface((Y+.5)*np.cos(X),\
        (Y+.5)*np.sin(X),\
        (Y+.5)*0*X, alpha=0.1)
ax.plot(np.cos(xn), np.sin(xn), 0*xn, 'r');
```



[59]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xn = np.linspace(-np.pi, np.pi, 100)
yn = np.linspace(0, 1, 50)
X, Y = np.meshgrid(xn, yn)

ax.plot_surface((.5+Y)*(np.cos(X)+np.cos(X/2)/3),\
        (.5+Y)*np.sin(X),\
        (.5+Y)*np.sin(X/2)/3, alpha=.1)
ax.plot(np.cos(xn)+np.cos(xn/2)/3, np.sin(xn), np.sin(xn/2)/3, 'y');
```


0.4
0.2
0.0
−0.2
−0.4
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

[ ]: