lektion5

November 11, 2021

Inhalt

1 Lektion 5

1.1 Wiederholung

[1]:
```
#%matplotlib notebook
%matplotlib inline
#%matplotlib qt
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)
fn = sp.lambdify(x, f)
fn(2)
```

[2]:
-0.756802495307928

[3]:
```
gn = lambda x: 5*np.cos(x)
```

[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])
```

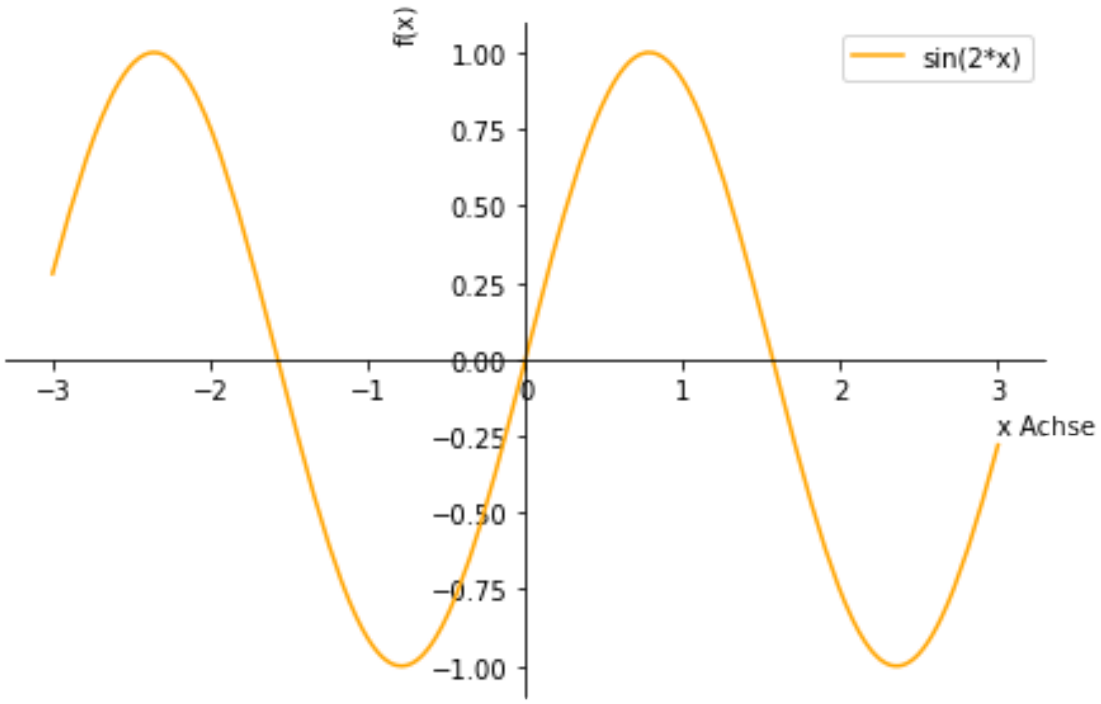
[5]:
```
fn(xn)
```

[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.        ])
```

[7]:
```
p1 = sp.plot(sp.sin(2*x), (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 Matplotlib Graphik 2D

1.2.1 Graphen von Funktionen (plot)

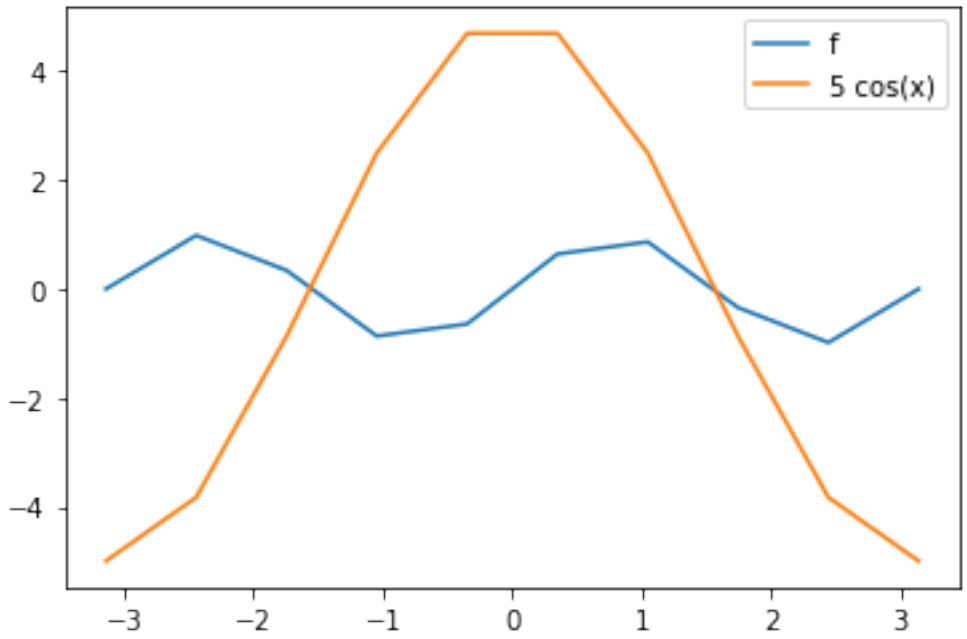
[8]:
```
plt.figure('mein erstes Bild')   # erzeugt ein Bild/Fenster mit qt backend
```

[8]:
```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

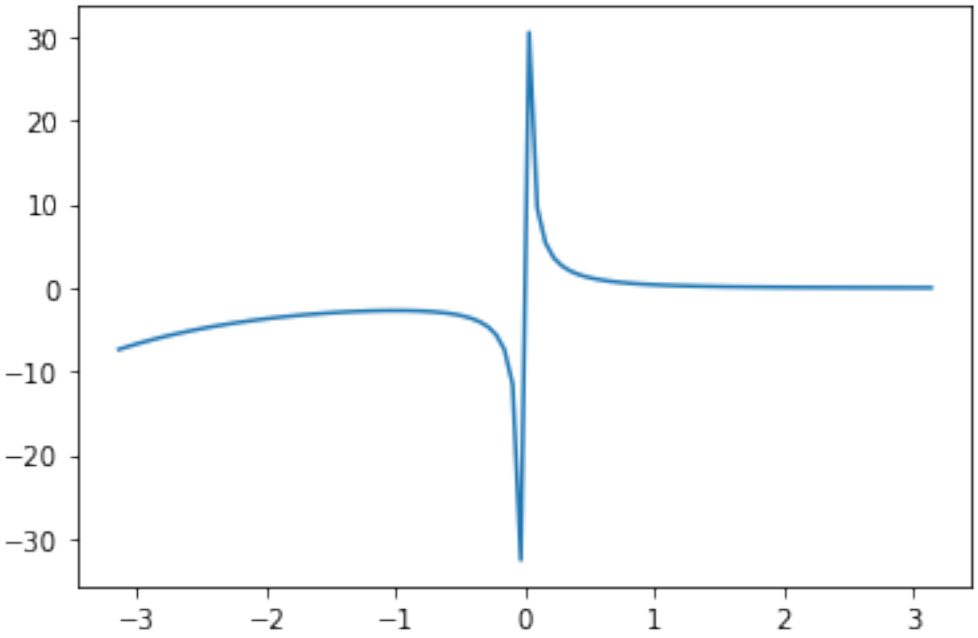
[9]:
```
plt.figure(100)
plt.plot(xn, fn(xn), label='f')
#  zeichnet in das aktive Koordinatensystem des aktiven Bildes
#  und erzeugt diese falls nicht vorhanden

plt.plot(xn, gn(xn), label='5 cos(x)')
# zeichnet in das aktive Koordinatensystem des aktiven Bildes

plt.legend()
# fügt den Inhalt der labels als Legende hinzu
```

[9]:
```
<matplotlib.legend.Legend at 0x7fb274d3f3d0>
```



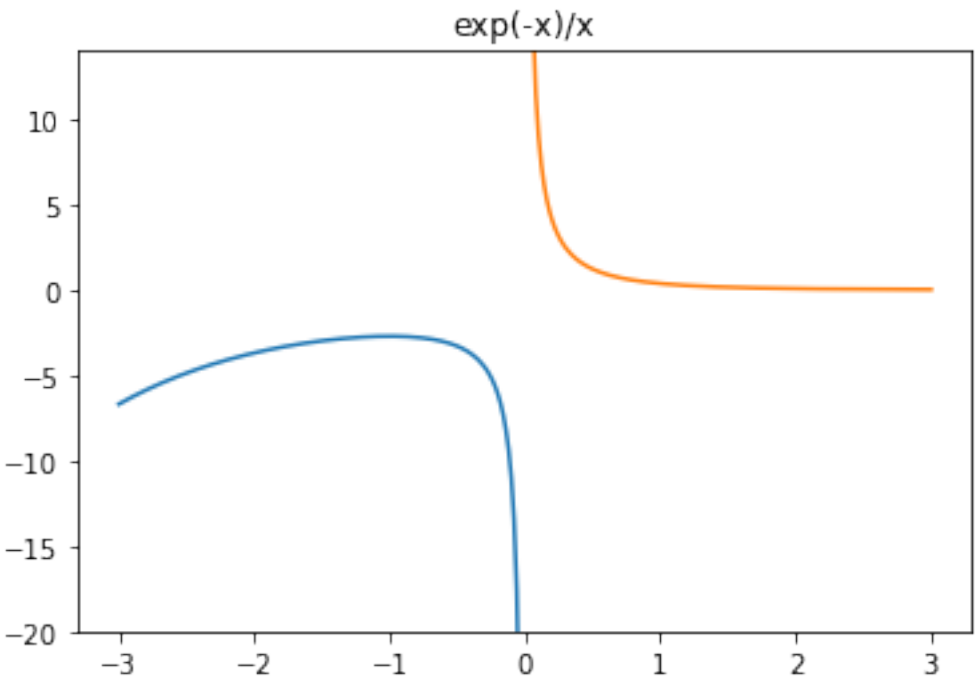
[10]:
```
x = sp.Symbol('x')
h = sp.exp(-x)/x
xn = np.linspace(-np.pi, np.pi, 100)
hn = sp.lambdify(x, h)
```

[11]:
```
plt.figure()
plt.plot(xn,hn(xn));
```

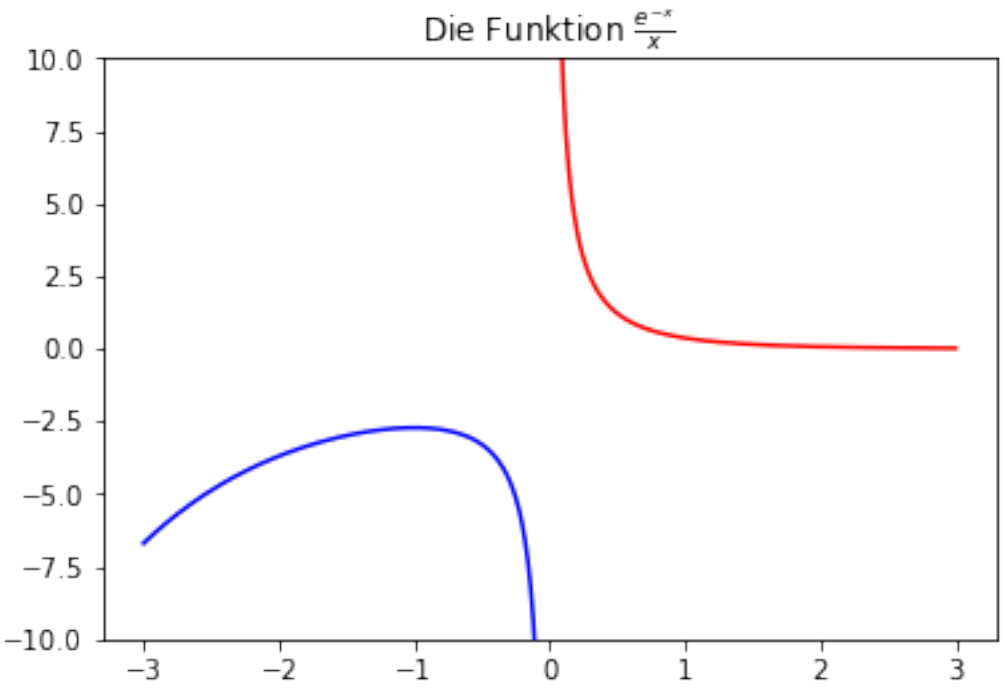


[12]:
```
xm = np.linspace(-3, -0.0001, 150)
xp = -xm
plt.figure()
plt.plot(xm, hn(xm))
plt.plot(xp, hn(xp))
#plt.ylim((-10,10))
plt.axis(ymin=-20,ymax=14)
plt.title(h)
```

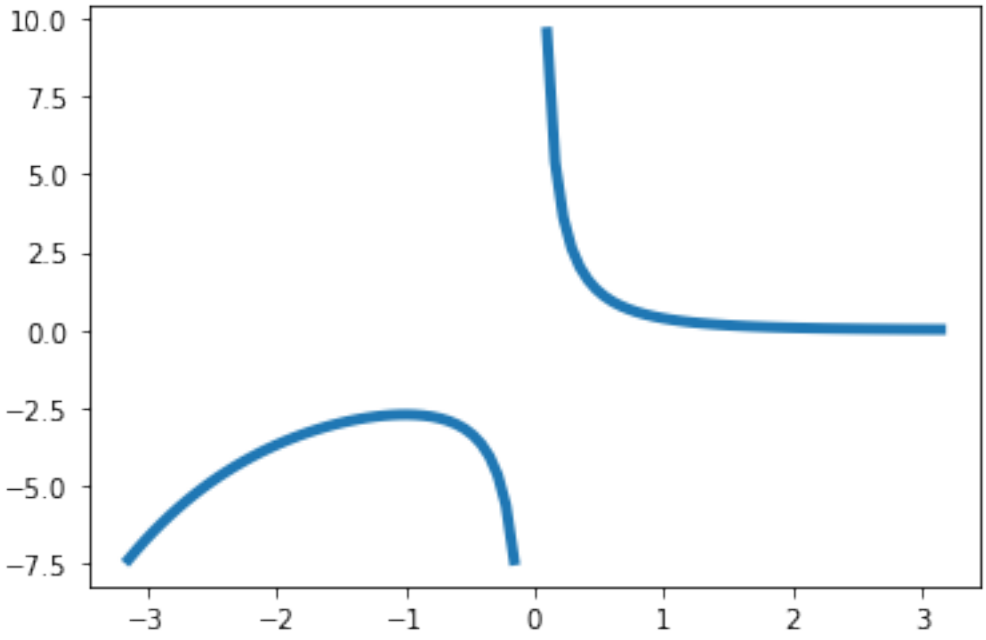
[12]:
```
Text(0.5, 1.0, 'exp(-x)/x')
```



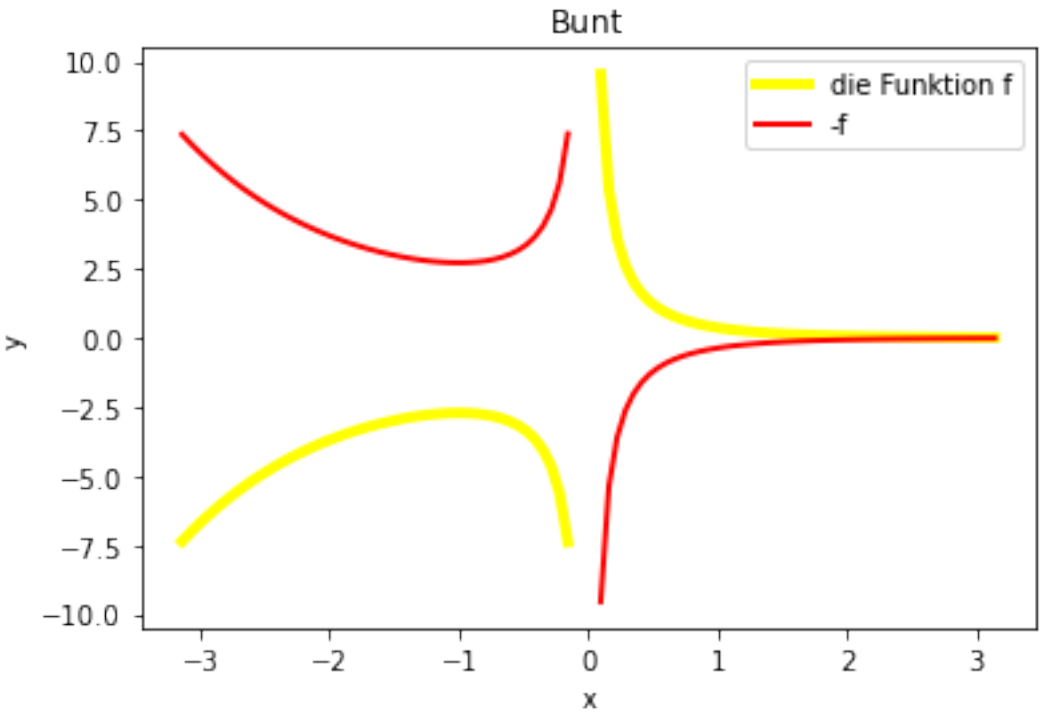
[13]:
```
xm = np.linspace(-3, -0.0001, 150)
xp = -xm
plt.figure()
plt.plot(xm, hn(xm), 'b')
plt.plot(xp, hn(xp), 'r')
plt.axis(ymin=-10, ymax=10)
txt = sp.latex(h)
plt.title(f"Die Funktion ${txt}$");
```



[14]:
```
plt.figure()
yn = hn(xn)
yn[abs(yn)>10] = np.nan  # 'not a number' Trick
plt.plot(xn, yn, linewidth=4);
```

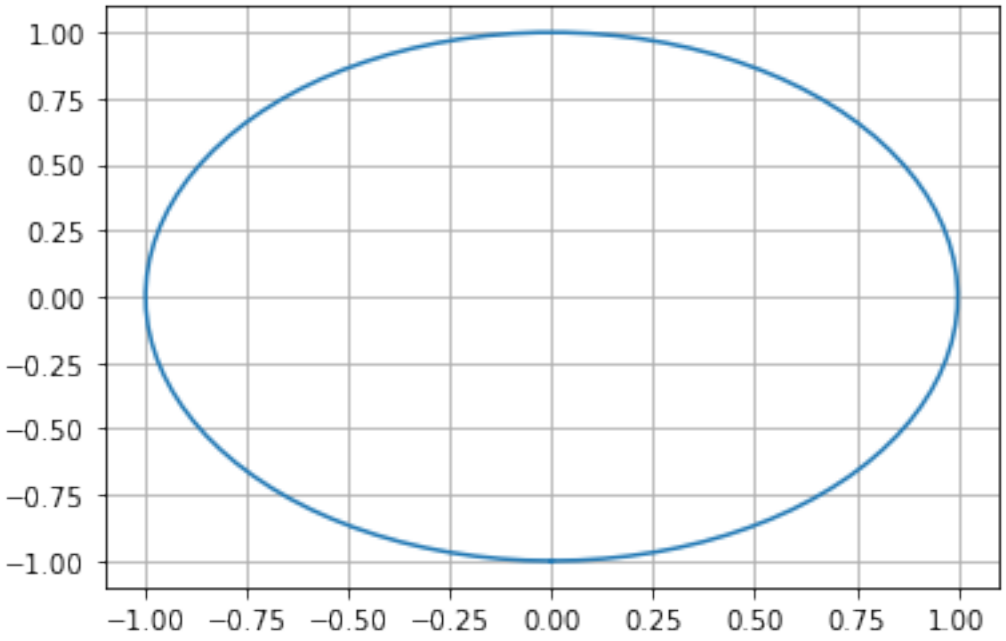


[15]:
```
plt.figure()
plt.plot(xn, yn, color='yellow', linewidth=4, label='die Funktion f') # plot in␣
 ↪das aktuelle Koordinatensystem
plt.plot(xn, -yn, color='red', linewidth=2, label='-f')
plt.ylabel('y')
plt.xlabel('x')
plt.title('Bunt')
plt.legend()
plt.show()
```



1.2.2 Parametrische Kurven (plot)

[16]:
```
fig = plt.figure()  # erzeugt ein neues Bild und darin ein Koordinatensystem
ax = fig.gca()      # Zugriff auf aktives Koordinatensystem (gca: get current␣
 ↪axes)
ax.plot(np.sin(xn), np.cos(xn))
plt.grid()
```

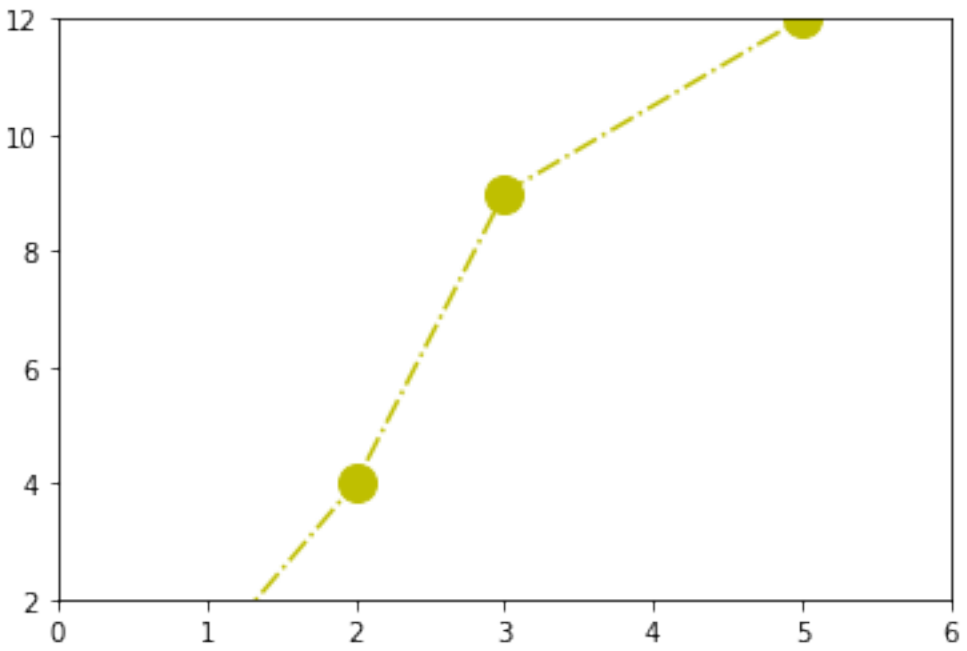


[17]:
```
ax.plot(np.sin(xn[0]), np.cos(xn[0]), 'rd', markersize=24)
ax.set_aspect('equal')
```

[18]:
```
ax.cla()   # clear axis
```

[19]:
```
fig.clf()  # clear figure
```

[20]:
```
plt.figure()
plt.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 -, :,  --, -.
plt.axis([0, 6, 2, 12])
?plt.plot
```



1.2.3 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
```

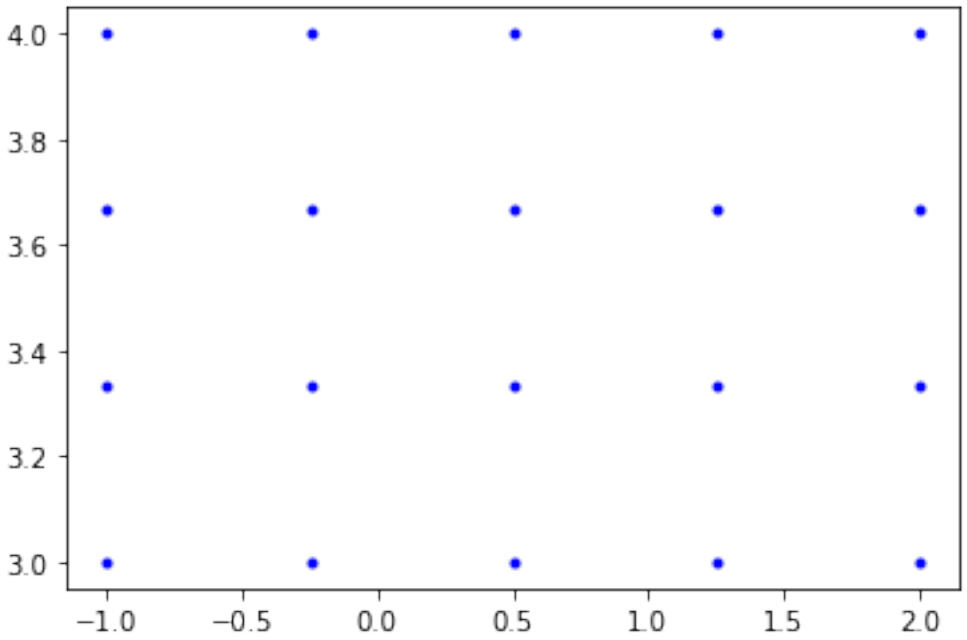
[22]:
```
xn = np.linspace(-2, 2, 100)
yn = np.linspace(-1.5, 2, 50)
X, Y = np.meshgrid(xn, yn) # x,y Koordinaten, der Gitterpunkte eines␣
 ↪kartesischen Gitters
Z = f(X, Y)                # Auswertung von f auf den Gitterpunkten
```

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

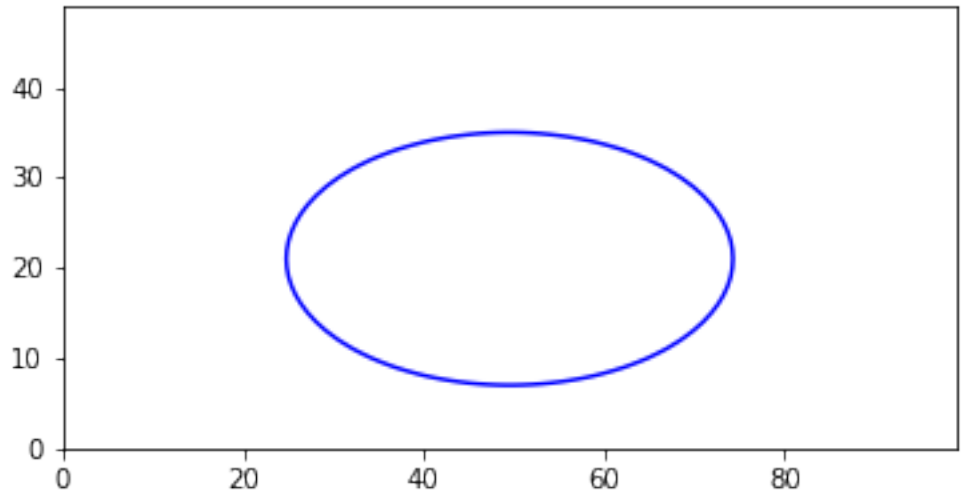
[23]:
```
(array([[-1.  , -0.25,  0.5 ,  1.25,  2.  ],
        [-1.  , -0.25,  0.5 ,  1.25,  2.  ],
        [-1.  , -0.25,  0.5 ,  1.25,  2.  ],
        [-1.  , -0.25,  0.5 ,  1.25,  2.  ]]),
 array([[3.        , 3.        , 3.        , 3.        , 3.        ],
        [3.33333333, 3.33333333, 3.33333333,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

```
       [3.66666667, 3.66666667, 3.66666667, 3.66666667, 3.66666667],
       [4.        , 4.        , 4.        , 4.        , 4.        ]]))
```

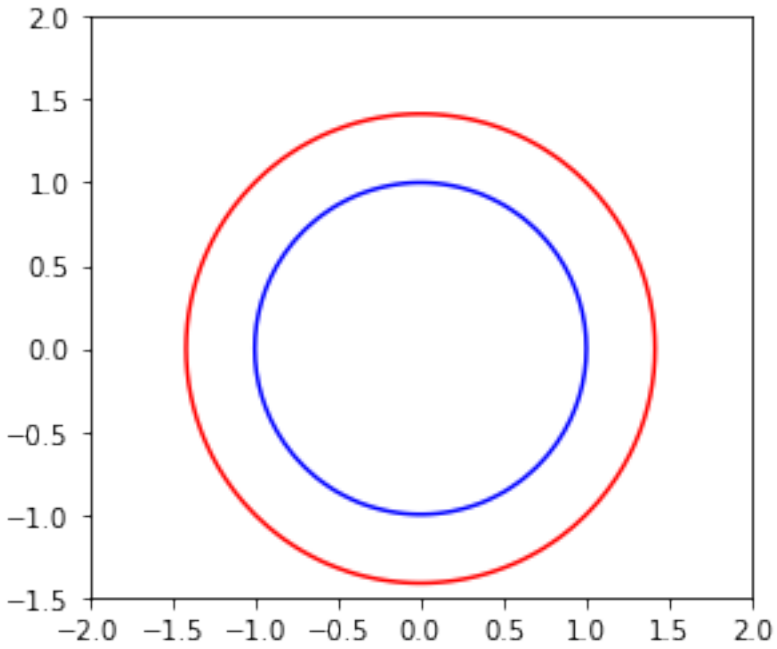
[24]:
```
plt.figure()
plt.plot(A,B, 'b.');
```



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



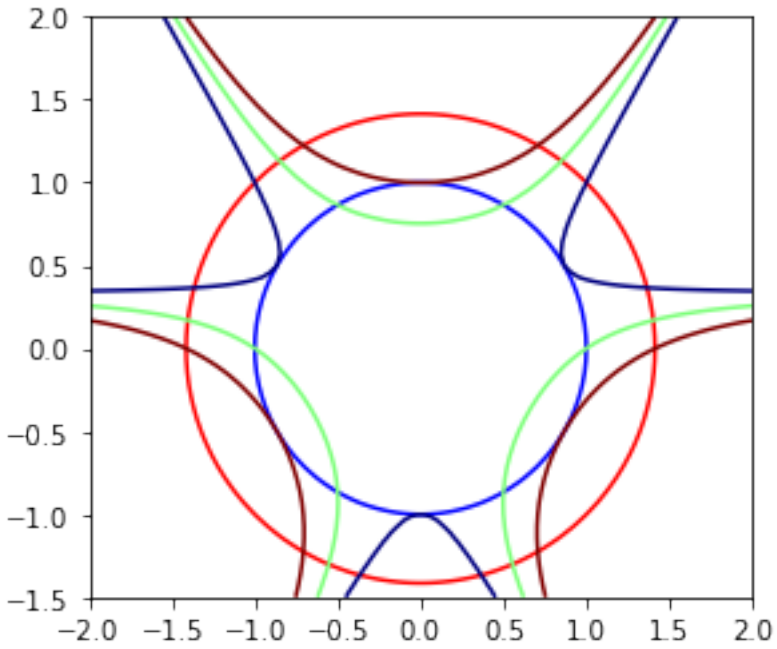
[26]:
```
# besser
fig = plt.figure()
ax = fig.gca()
ax.contour(xn, yn, Z, [0, 1], colors=['blue', 'red'])
# ax.contour(X, X, Z, [0, 1], colors=['blue', 'red']) # alternativ
# 0 und 1 Höhenline in blau und rot, jetzt mit x,y Koordinaten
ax.set_aspect('equal')
```



[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')
```

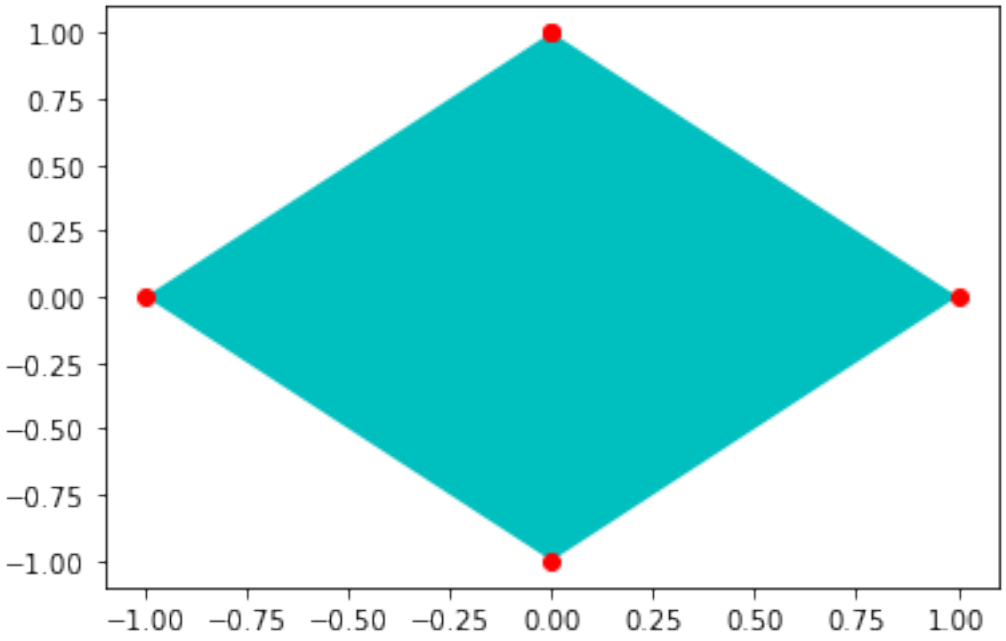


Alle Colormaps findet man auf https://matplotlib.org/stable/gallery/color/colormap_reference.html

1.2.4 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');
```



1.3 Matplotlib 3D Graphik

1.3.1 Graphen von Funktionen (plot_surface)

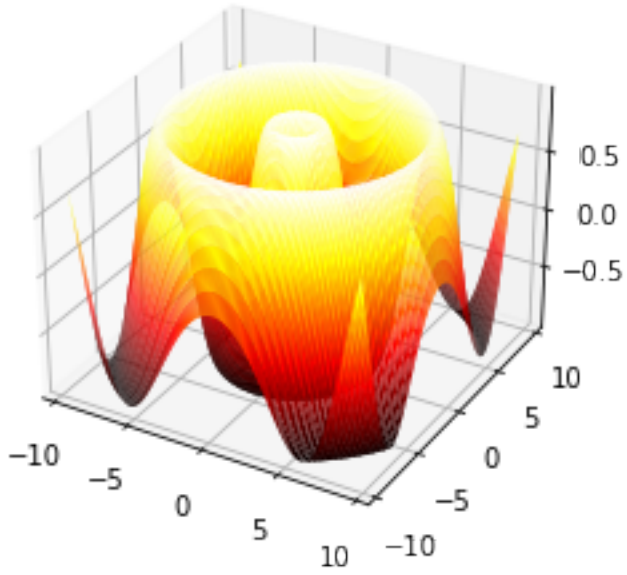
[32]:
```
x = sp.Symbol('x')
y = sp.Symbol('y')
f = sp.sin(sp.sqrt(x**2+y**2))
f
```

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

[33]:
```
fn = sp.lambdify((x, y), f)
```

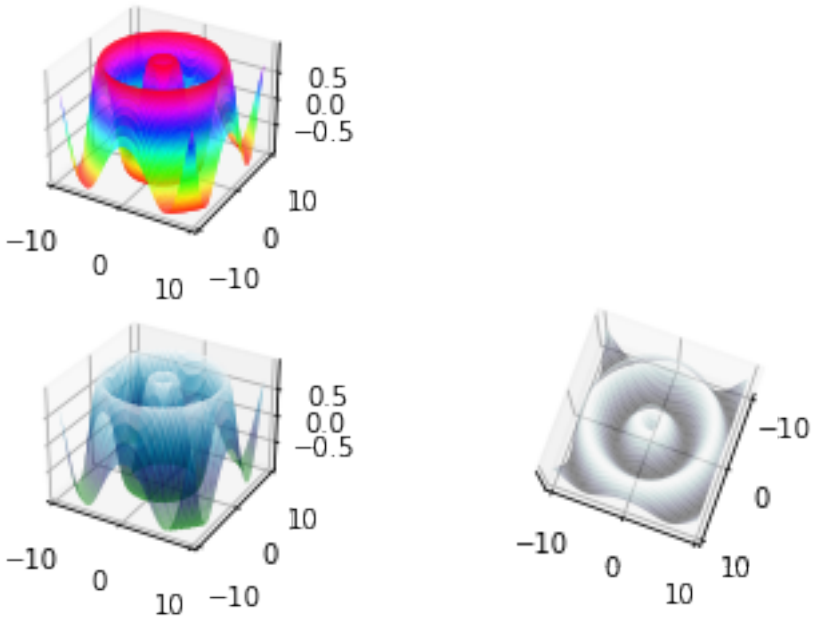
[34]:
```
xn = np.linspace(-3*np.pi, 3*np.pi, 100)
yn = np.linspace(-3*np.pi, 3*np.pi, 100)
X, Y = np.meshgrid(xn, yn)
```

[35]:
```
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);
```



[36]:
```
fig = plt.figure()
ax1 = fig.add_subplot(221, projection='3d')
ax1.plot_surface(X, Y, Z, rstride=1, cstride=1,
                 cmap=plt.cm.hsv, linewidth=1,
                 alpha=1);   # alpha Transparenzwert (1 undurchsichtig)
ax2 = fig.add_subplot(223, projection='3d')
ax2.plot_surface(X, Y, Z,
                 cmap=plt.cm.ocean, linewidth=1,
                 alpha=0.5);
ax4 = fig.add_subplot(224, projection='3d')
ax4.plot_surface(X, Y, Z,
                 cmap=plt.cm.bone, linewidth=1,
                 alpha=0.5);
ax4.view_init(80,20)          # Blickwickel auf das Bild im Koordinatensystem ax4
ax4.set_zticks([])
```

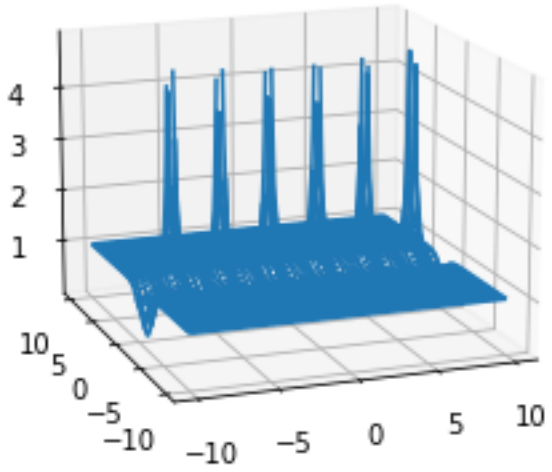
[36]:
```
[]
```



[37]:
```
f = abs(sp.tan(x+sp.I*y))   # Betrag des Tangens in der komplexen Ebene
fn = sp.lambdify((x, y), f)
```

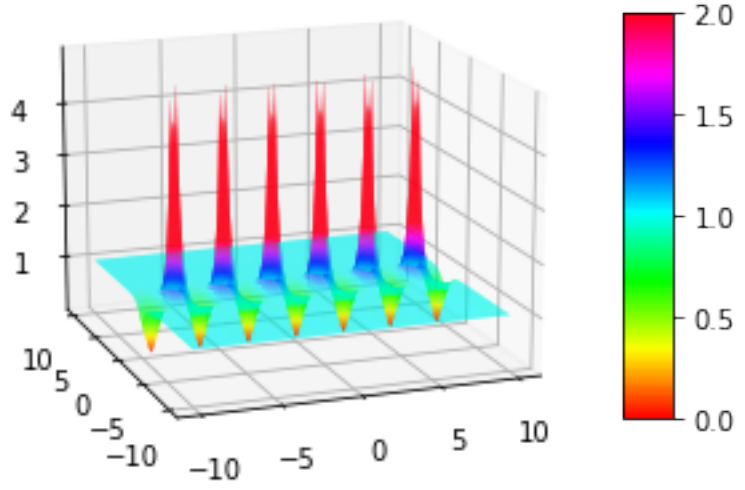
[38]:
```
xn = np.linspace(-10, 10, 300)
yn = xn
X, Y = np.meshgrid(xn,  yn) # Gitter in der komplexen Ebene X: Realteil Y:␣
 ↪Imaginaerteil
Z = fn(X, Y)
Z[Z>5] = np.nan  # Werte größer als 5 werden ignoriert
```

[39]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z)
ax.view_init(15, -110)
```



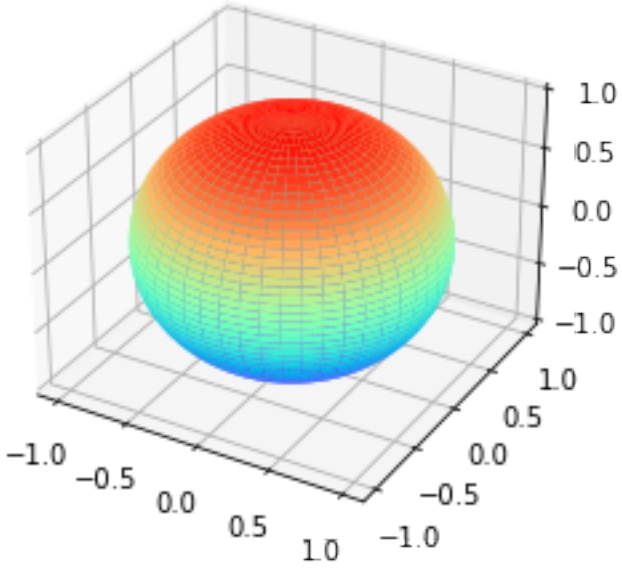
[40]:
```
from matplotlib.colors import Normalize
Normierung = Normalize(0, 2)
#Werte zwischen 0 und 2 sollen eingefärbt werden Werte größer 2 werden wie 2␣
 ↪eingefärbt

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);
```

```
/tmp/ipykernel_145337/3711892616.py:8: UserWarning: Z contains NaN values. This
may result in rendering artifacts.
  surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.hsv, linewidth=0, rstride=1,
cstride=1, norm=Normierung)
```



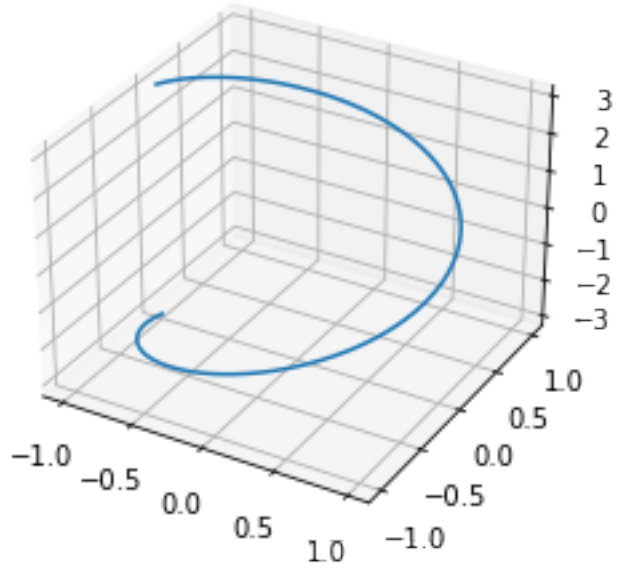
1.3.2 Parametrische Flächen (plot_surface)

[41]:
```
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);
```



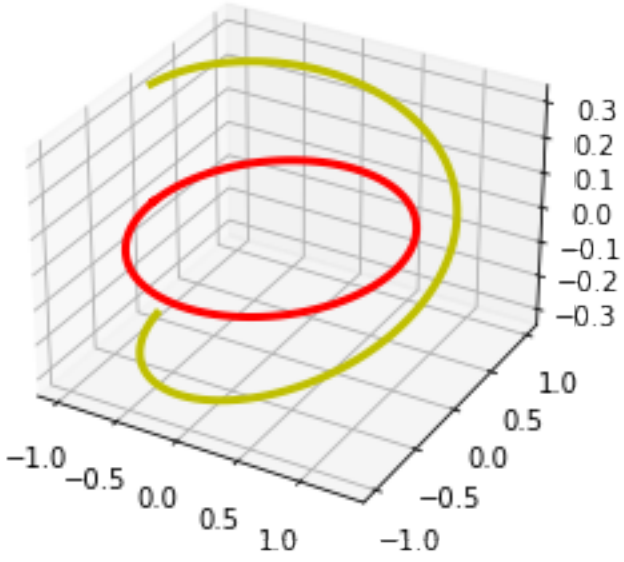
1.3.3 Raumkurven in Parameterdarstellung (plot)

[42]:
```
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);
```

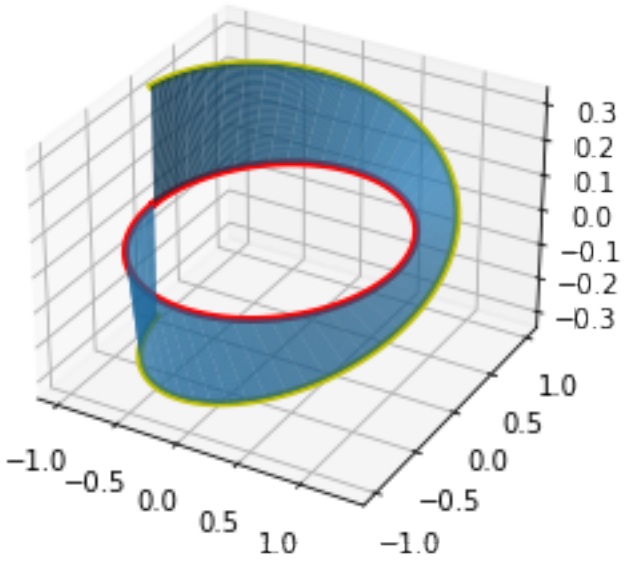


ein gewundenes Band

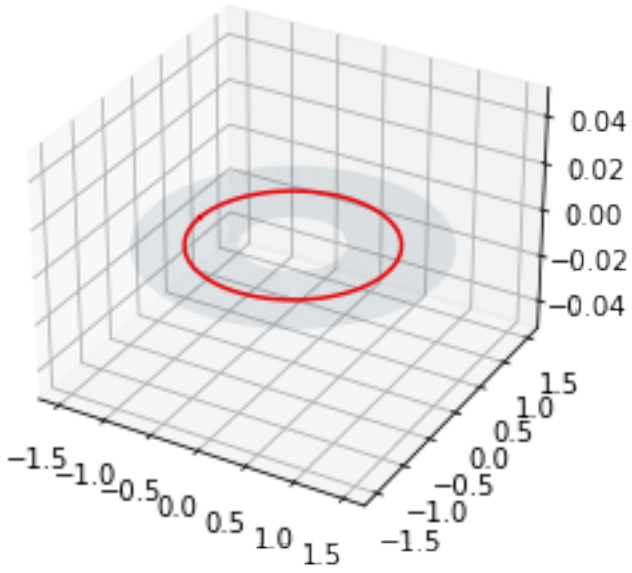
[43]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

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);
```



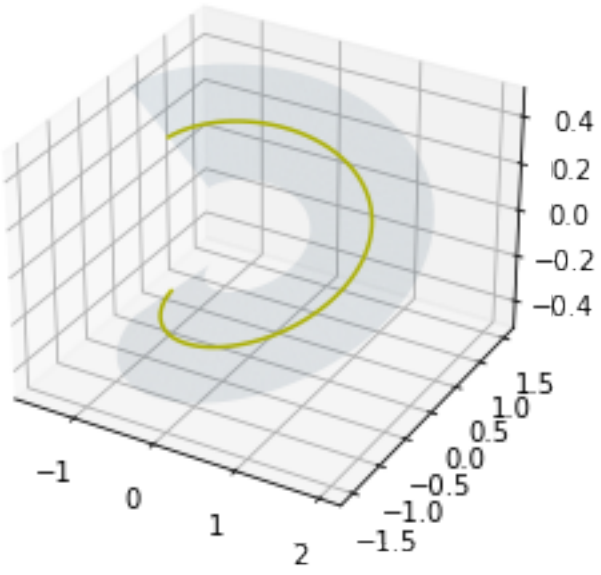
[44]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

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);
```



[45]:
```
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');
```



[46]:
```
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
0
1
2

[ ]: