lektion4

November 7, 2019

Table of Contents

1 Lektion 4

2 Graphik

[1]:
```
import matplotlib.pyplot as plt
import sympy as sp
import numpy as np
```

[2]:
```
#%matplotlib notebook
%matplotlib inline
#%matplotlib qt
```

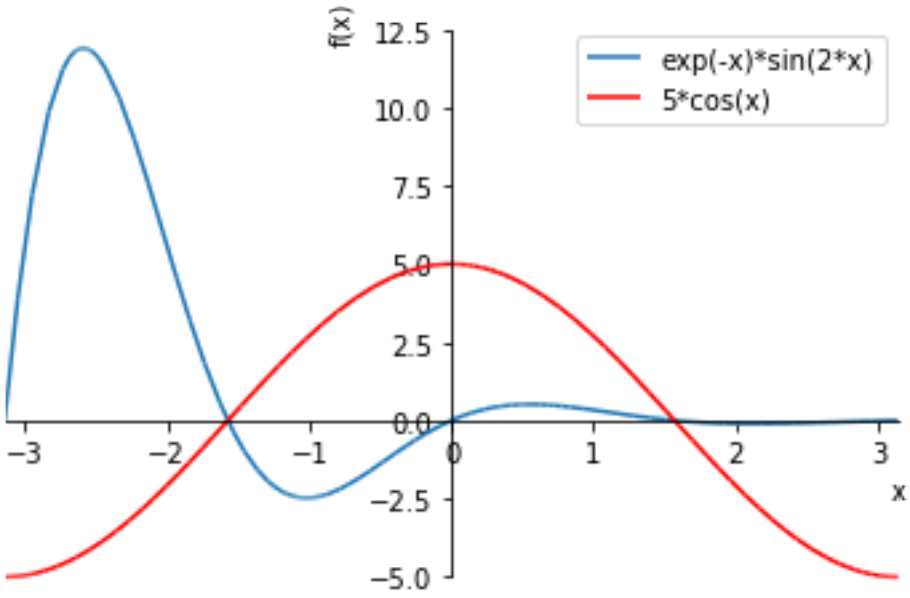
2.1 Sympy Graphik 2D

2.1.1 Einfache Graphen von Funktionen

Graph: Alle Punkte (x, y) mit

$$y = f(x)$$

[3]:
```
x = sp.symbols('x')
f = sp.exp(-x)*sp.sin(2*x)
p1 = sp.plot(f,(x,-sp.pi,sp.pi),show=False)
p2 = sp.plot(5*sp.cos(x),(x,-sp.pi,sp.pi),show=False)
p1.extend(p2)
p1.legend=True
p1[1].line_color='red'
p1.show()
```



[4]:
```
dir(p1)
```

[4]:
```
['__class__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
```

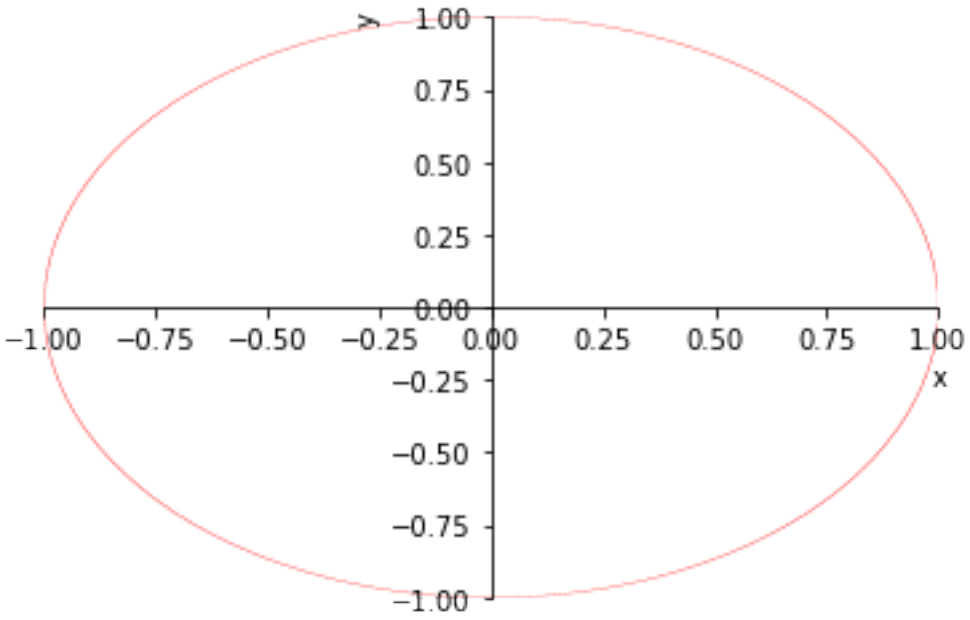
```
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_backend',
 '_series',
 'append',
 'aspect_ratio',
 'autoscale',
 'axis',
 'axis_center',
 'backend',
 'extend',
 'legend',
 'margin',
 'save',
 'show',
 'title',
 'xlabel',
 'xlim',
 'xscale',
 'ylabel',
 'ylim',
 'yscale']
```

2.1.2 Implizit gegebene Kurven

Alle Punkte (x, y) mit

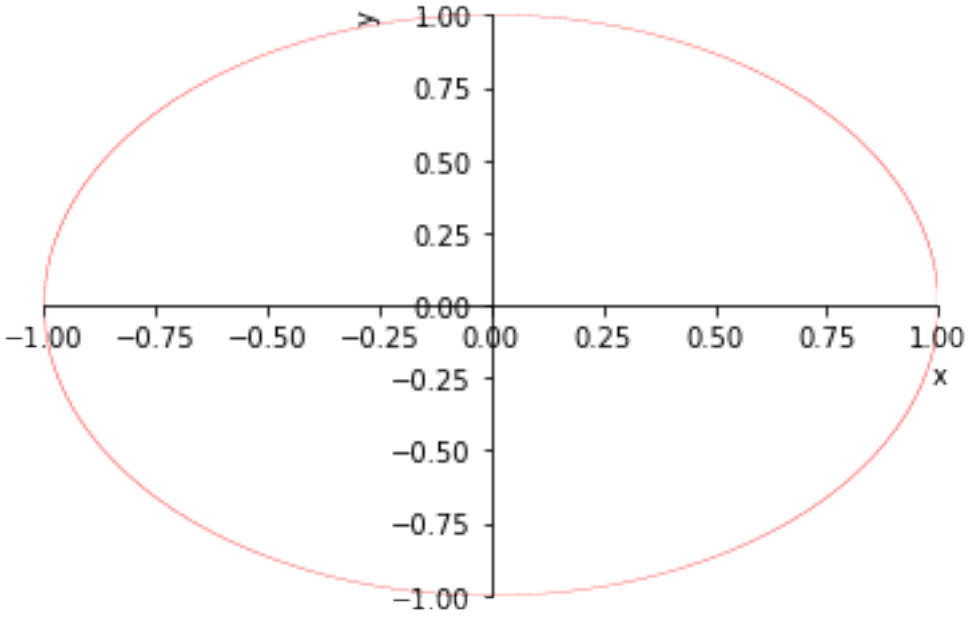
$$g(x, y) = 0$$

[5]:
```
x,y,z =sp.symbols('x y z')
p3 = sp.plotting.plot_implicit(sp.
 ↪Eq(x**2+y**2,1),(x,-1,1),(y,-1,1),line_color='red')
```



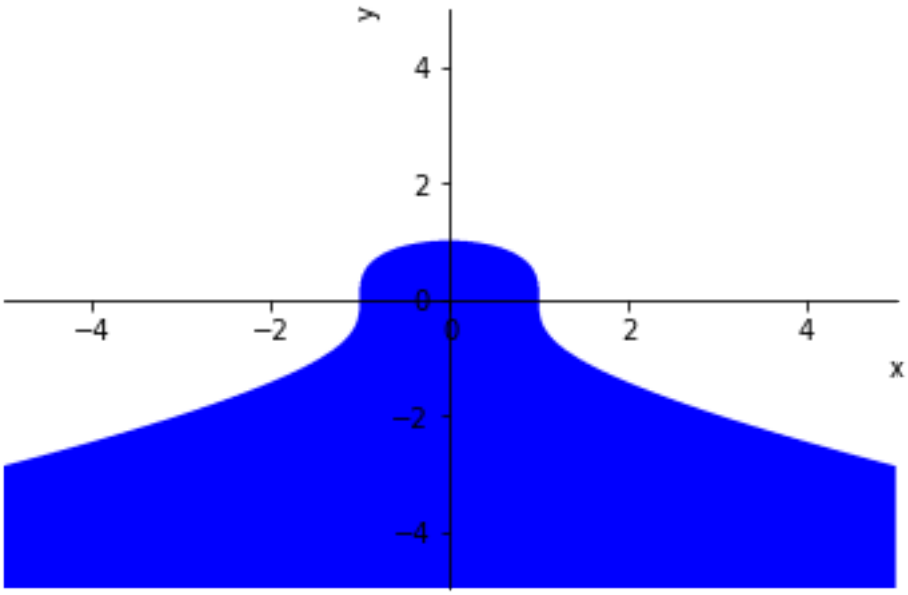
Implizit gegebene Kurven

[6]:
```
x,y,z =sp.symbols('x y z')
p3 = sp.plotting.plot_implicit(sp.
 ↪Eq(x**2+y**2,1),(x,-1,1),(y,-1,1),line_color='red')
ax = p3._backend.ax
ax.set_aspect('equal') # funktioniert nicht im inline Modus
```



durch Ungleichung z.B. $\leq$ gegebenen Bereiche

[7]:
```
sp.plotting.plot_implicit(sp.Le(x**2+y**3,1))
```



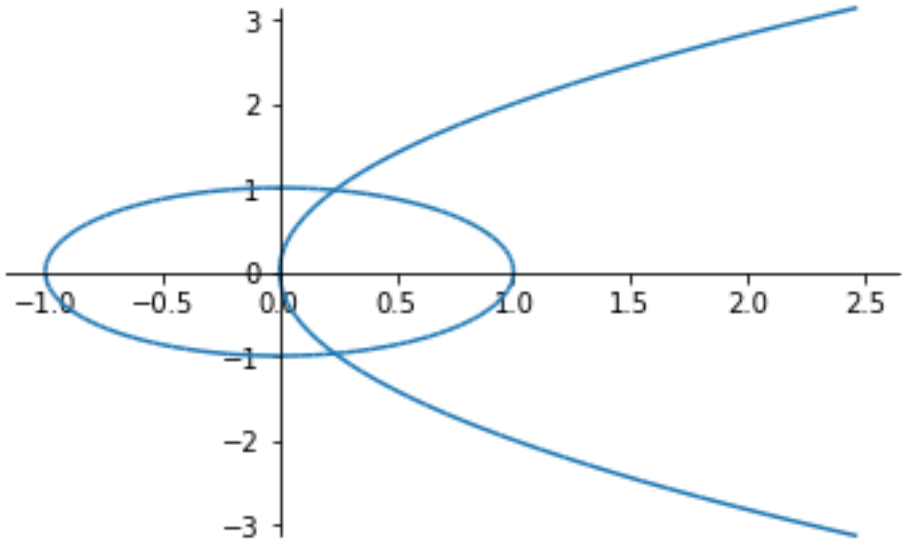
[7]:
```
<sympy.plotting.plot.Plot at 0x7fe8d8e01d50>
```

2.1.3 Kurven in Parameterdarstellung in der Ebene

Punkte (x, y) in der Ebene mit

$$x = f(t),\ y = g(t) \quad t \in [a, b]$$

[8]:
```
t = sp.symbols('t')
p1 = sp.plotting.plot_parametric((sp.sin(t),sp.cos(t)),((t/2)**2,t),(t,-sp.
 ↪pi,sp.pi))
ax = p1._backend.ax
ax.set_aspect('equal')
lc = p1._backend
```



Was gibt es noch?

[9]:
```
?sp.plotting.
```

```
Object `sp.plotting. ` not found.
```

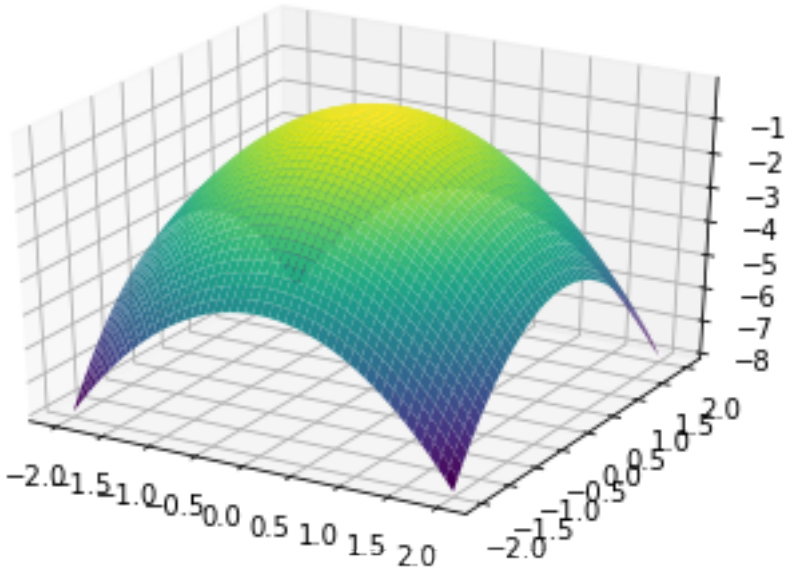
2.2 Sympy Graphik 3D

2.2.1 Flächen

als Graph einer Funktion $f : \mathbb{R}^2 \to \mathbb{R} : (x, y) \mapsto f(x, y)$

Alle Punkte $(x, y, f(x, y))$ mit $x \in (x_1, x_2),\ y \in (y_1, y_2)$

[10]:
```
sp.plotting.plot3d((-x**2-y**2),(x,-2,2), (y,-2,2));
```



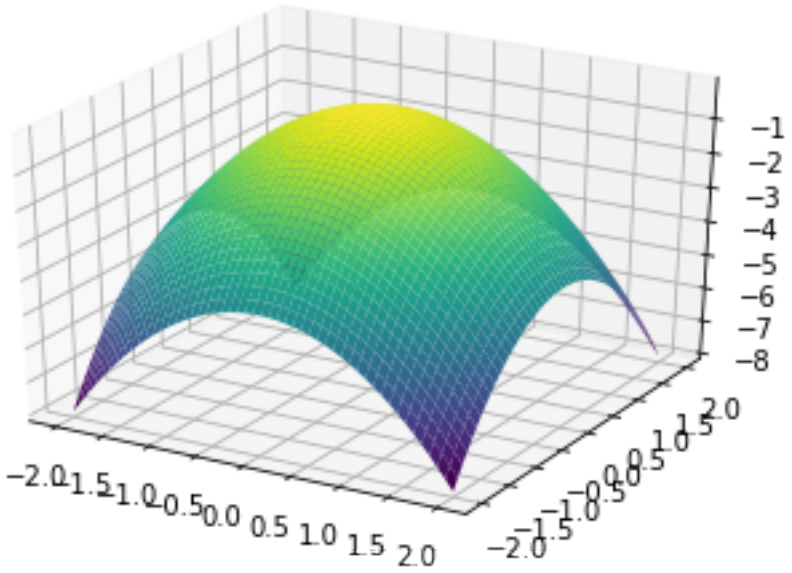
2.2.2 Parametrische Flächen

Alle Punkte $(x, y, z) \in \mathbb{R}^3$ so, dass

$$\begin{aligned} x &= f_1(u, v) \\ y &= f_2(u, v) \\ z &= f_3(u, v) \end{aligned}$$

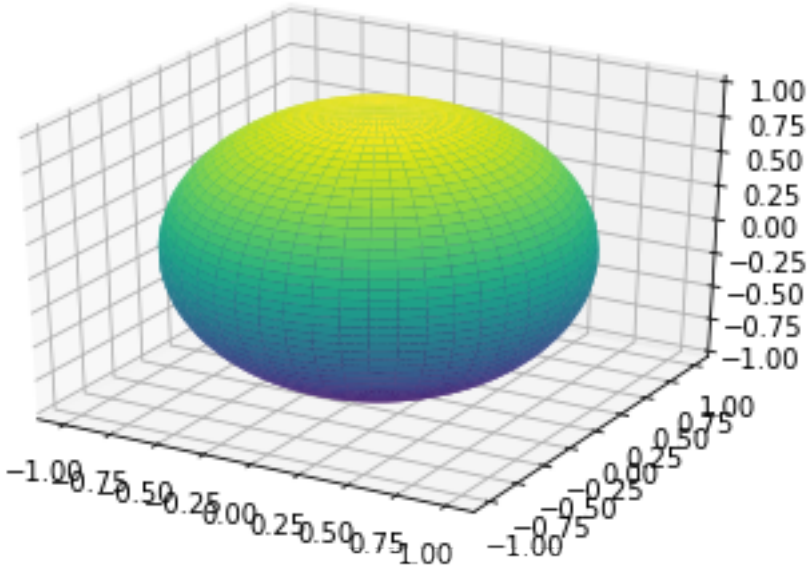
für (u, v) in einem Bereich in $\mathbb{R}^2$ und $f_j : \mathbb{R}^2 \to \mathbb{R}$.

[11]:
```
u,v = sp.symbols('u v')
sp.plotting.plot3d_parametric_surface(u,v,-u**2-v**2,(u,-2,2),(v,-2,2))
```



[11]: <sympy.plotting.plot.Plot at 0x7fe8d82c5cd0>

[12]:
```
azimuth, polar, radius = sp.symbols('azimuth polar radius') # Kugel
radius = 1
p4 = sp.plotting.plot3d_parametric_surface(radius*sp.sin(polar)*sp.
 ↪cos(azimuth), \
                                           radius*sp.sin(polar)*sp.sin(azimuth), \
                                           radius*sp.cos(polar),(polar,0,sp.
 ↪pi),(azimuth,-sp.pi,sp.pi))
ax = p4._backend.ax

#ax.set_aspect('equal')
#alternativ: (unperfekte Lösung)
ax.set_xlim3d((-1,1))
ax.set_ylim3d((-1,1))
ax.set_zlim3d((-1,1))
```



[12]:
```
(-1, 1)
```

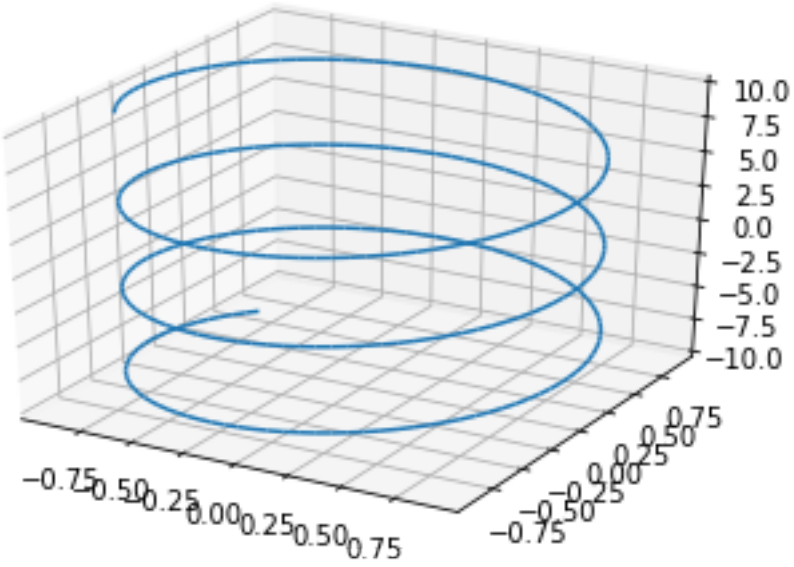
2.2.3 Kurven in Parameterdarstellung

Alle Punkte $(x, y, z) \in \mathbb{R}^3$ sodass

$$\begin{aligned} x &= f_1(u) \\ y &= f_2(u) \\ z &= f_3(u) \end{aligned}$$

für u in einem Intervall.

[13]:
```
sp.plotting.plot3d_parametric_line(sp.cos(u),sp.sin(u),u,(u,-10,10))
```



[13]: <sympy.plotting.plot.Plot at 0x7fe8d82c52d0>

2.3 Matplotlib Graphik 2D

2.3.1 Graphen von Funktionen (plot)

[14]:
```
x = sp.symbols('x')
f = sp.sin(2*x)
fn = sp.lambdify(x,f)
```

[15]:
```
gn = lambda x: 5*np.cos(x)
```

[16]:
```
xn = np.linspace(-np.pi,np.pi,100) # 100 äquidistante Punkte in [-pi,pi]
```

[17]:
```
plt.figure('mein erstes Bild')   # erzeugt ein Bild/Fenster
```

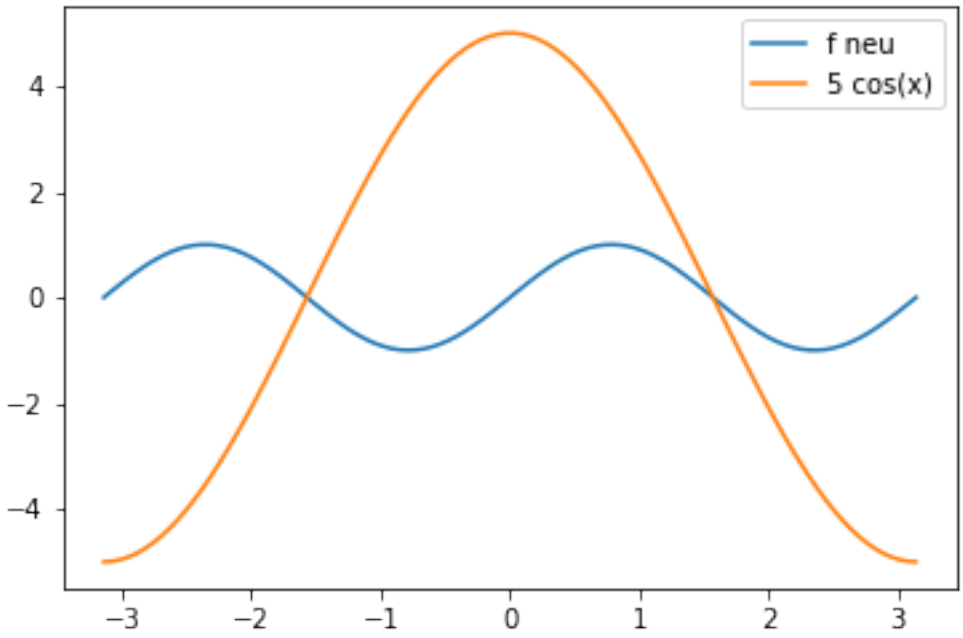
[17]: <Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

[18]:
```
#plt.figure(100)
plt.plot(xn,fn(xn),label='f neu')
#  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
```

[18]: <matplotlib.legend.Legend at 0x7fe8d8cbd890>

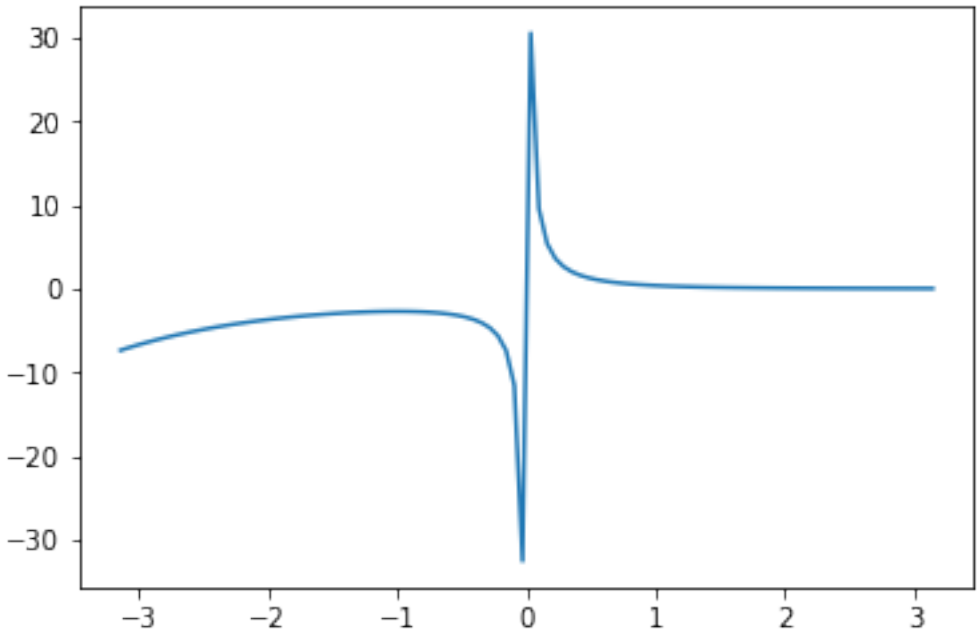


[19]:
```
# Restart
import matplotlib.pyplot as plt
import sympy as sp
import numpy as np
```

[20]:
```
x = sp.Symbol('x')
h = sp.exp(-x)/x
xn = np.linspace(-np.pi,np.pi,100)
hn = sp.lambdify(x,h)
```

[21]:
```
%matplotlib inline
#plt.figure()
plt.plot(xn,hn(xn))
```

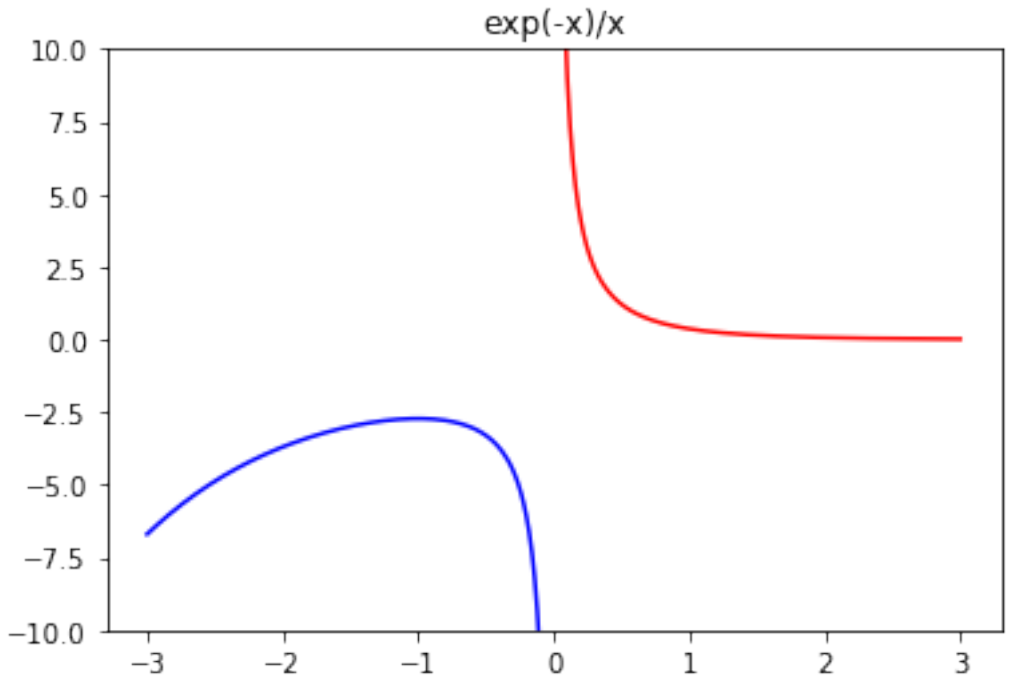
[21]: [<matplotlib.lines.Line2D at 0x7fe8d1af6650>]



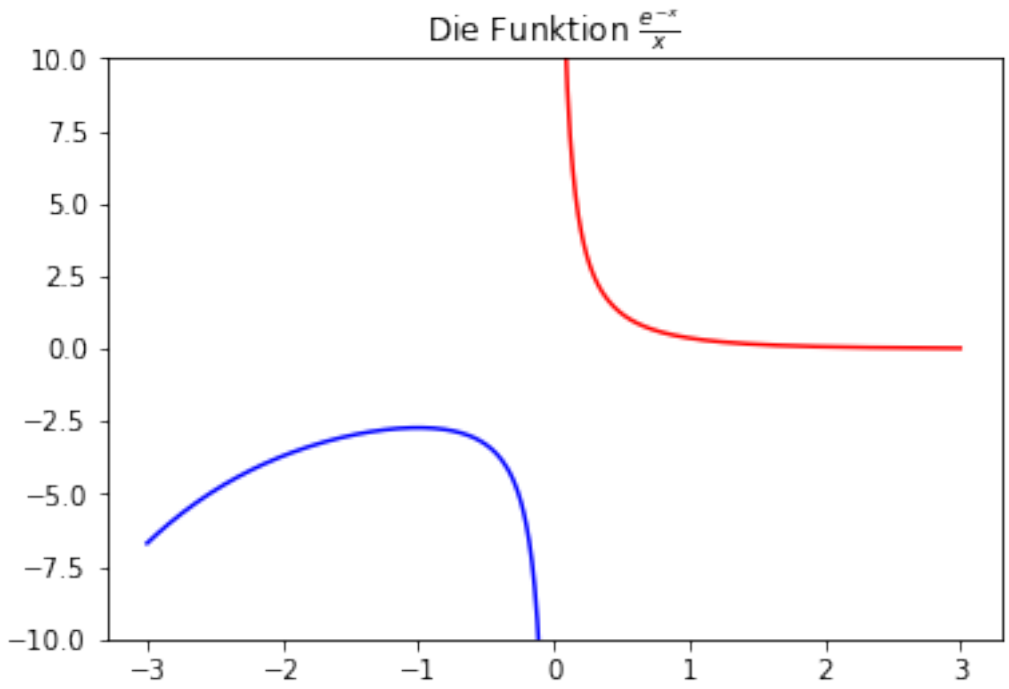
[22]:
```
# Restart
import matplotlib.pyplot as plt
import sympy as sp
import numpy as np
x = sp.Symbol('x')
h = sp.exp(-x)/x
xn = np.linspace(-np.pi,np.pi,100)
hn = sp.lambdify(x,h)
```

[23]:
```
#%matplotlib qt
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)
# alternativ
plt.ylim((-10,10))
plt.title(str(h))
```

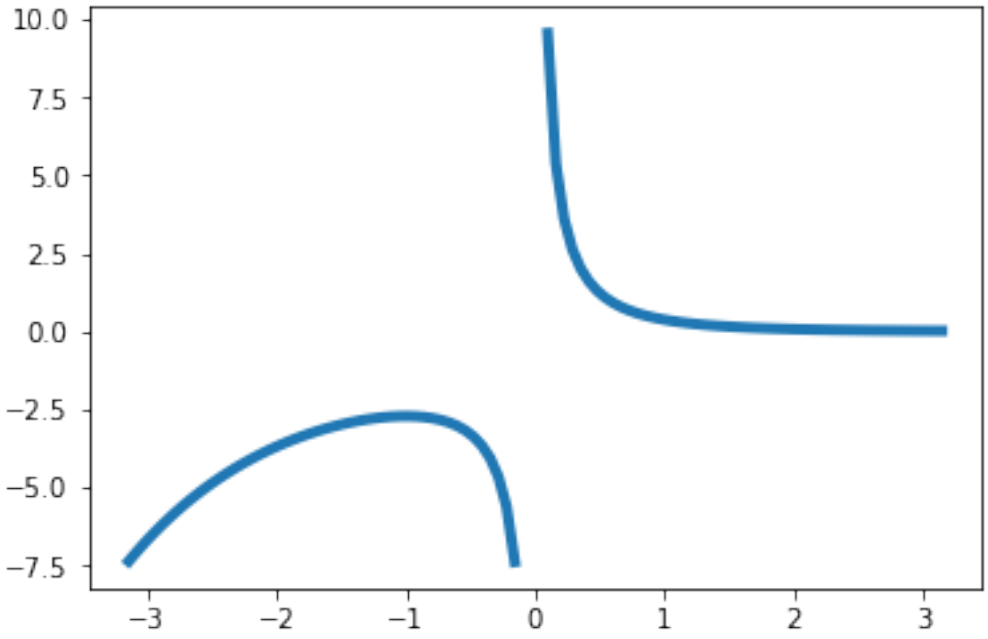
[23]:
```
Text(0.5, 1.0, 'exp(-x)/x')
```



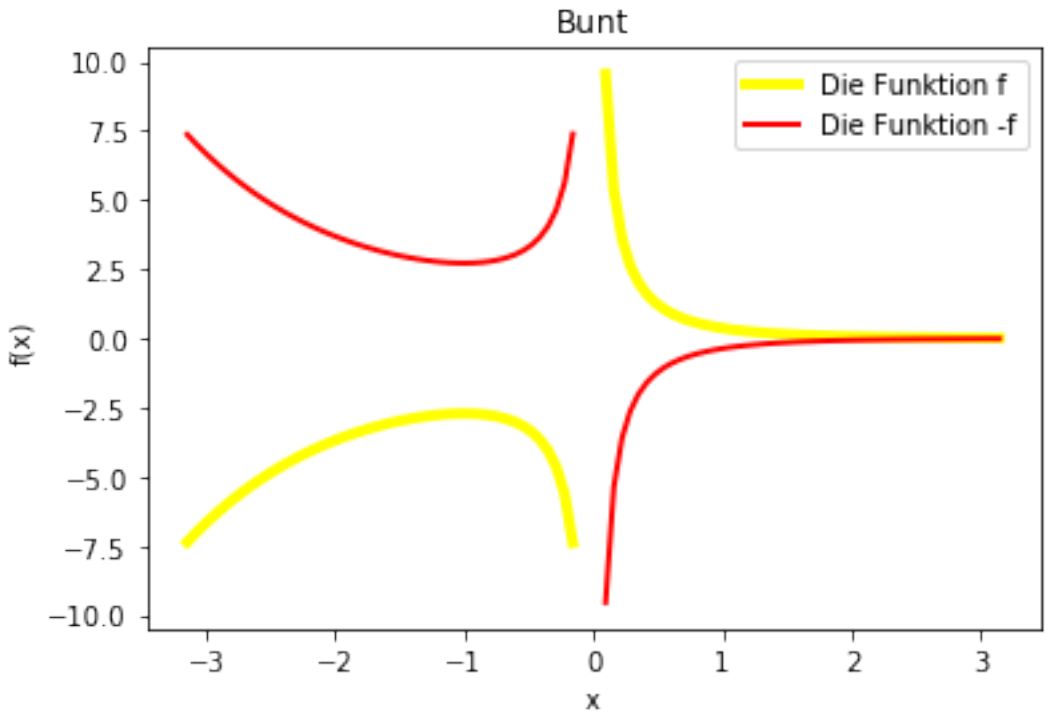
[24]:
```
xm = np.linspace(-3,-0.0001,150)
xp = -xm
plt.figure()
plt.plot(xm,hn(xm),'b')
plt.plot(xp,hn(xp),'r')
plt.axis(ymin=-10,ymax=10)
txt = sp.latex(h)
plt.title(r'Die Funktion ${0:s}$'.format(txt));
```



[25]:
```
plt.figure()
yn = hn(xn)
yn[abs(yn)>10] = np.nan  # Not a number Trick
plt.plot(xn,yn,linewidth=4);
```

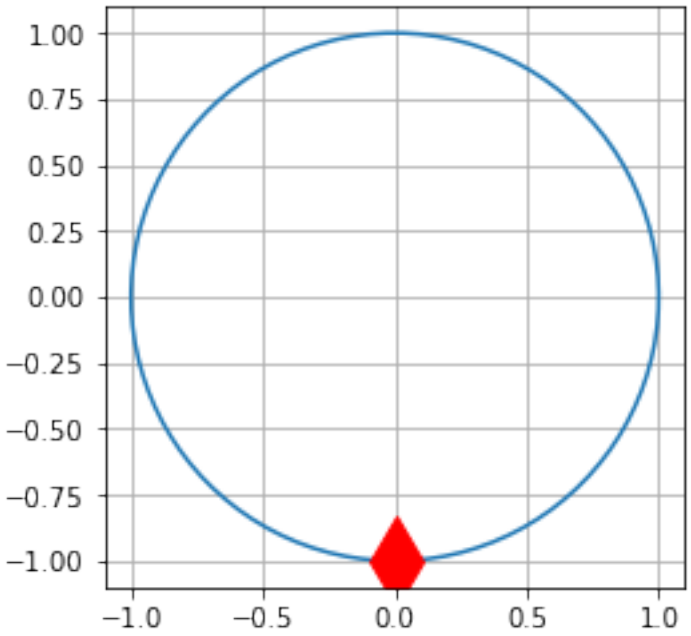


[26]:
```
fig = plt.figure()
plt.plot(xn,yn,color='yellow',linewidth=4,label='Die Funktion f') # plot in das␣
 ↪aktuelle Koordinatensystem
plt.plot(xn,-yn,color='red',linewidth=2,label='Die Funktion -f')
plt.ylabel('f(x)')
plt.xlabel('x')
plt.title('Bunt')
plt.legend()
plt.show()
```

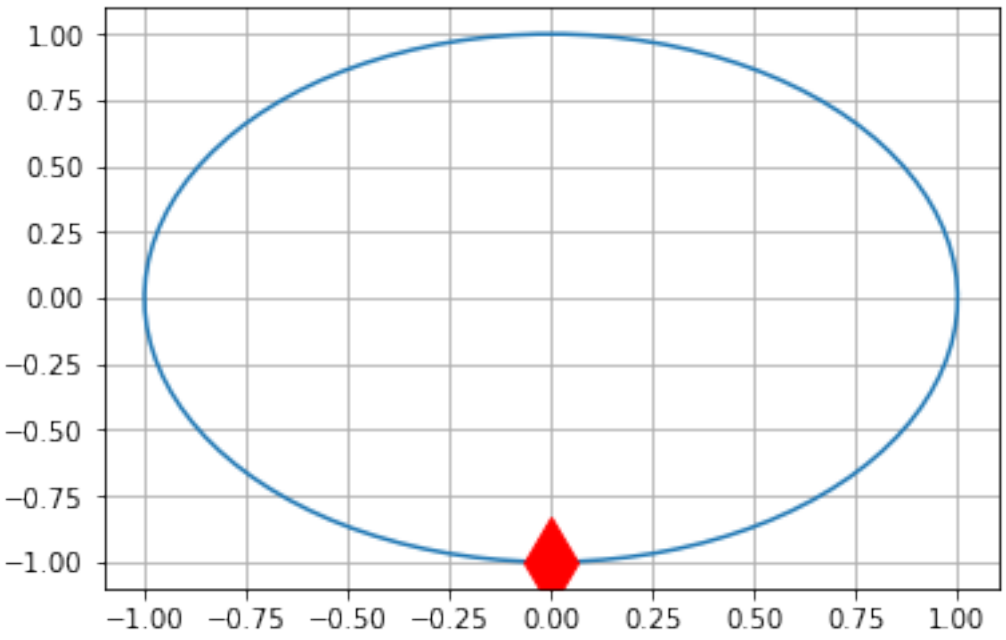


2.3.2 Parametrische Kurven (plot)

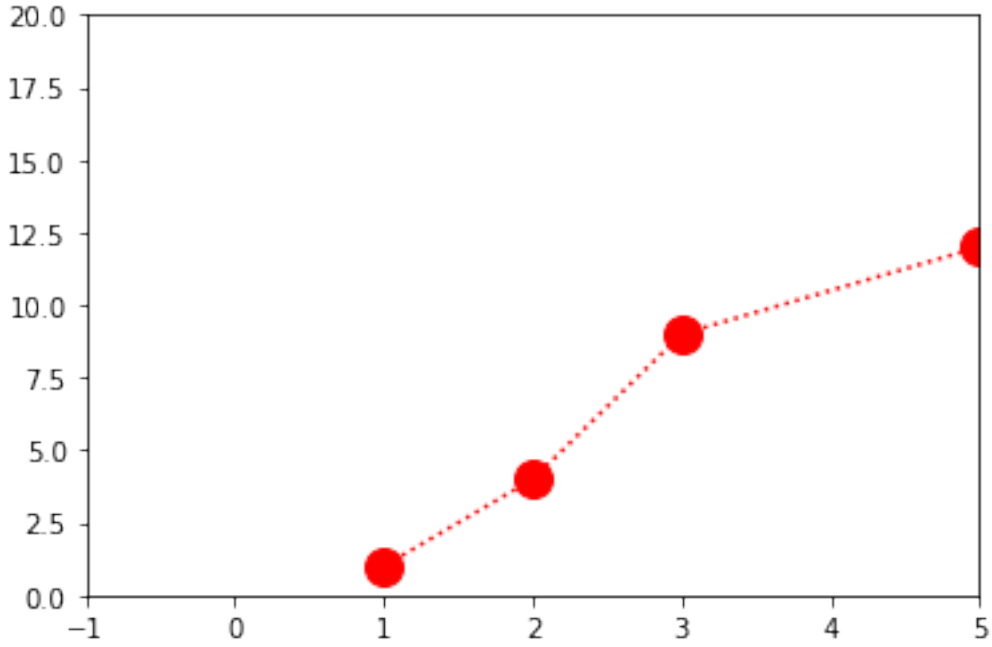
[27]:
```
fig = plt.figure()  # erzeugt ein neues Bild und darin ein Koordinatensystem
ax = fig.gca()      # Zugriff auf aktives Koordinatensystem (gca: get current␣
 ↪axes)
p1 = ax.plot(np.sin(xn),np.cos(xn))
p2 = ax.plot(np.sin(xn[0]),np.cos(xn[0]),'rd',MarkerSize=24)
ax.set_aspect('equal')
plt.grid()
```



[28]:
```
fig = plt.figure(100) # neues Bild
plt.plot(np.sin(xn),np.cos(xn),np.sin(xn[0]),np.cos(xn[0]),'rd',MarkerSize=24)
plt.grid()
```



[29]:
```
plt.figure()
plt.plot([1, 2, 3, 5],[1, 4, 9, 12],'ro:',MarkerSize=14)
# Farbe: r, g, b, c, m  ... rot, gruen, blau, cyan, magenta
# Marker: x, o, +, *, ....
plt.axis([-1,5,0,20])
?plt.plot
```



2.3.3 Implizit gegebene Kurven / Höhenlinien (contour)

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

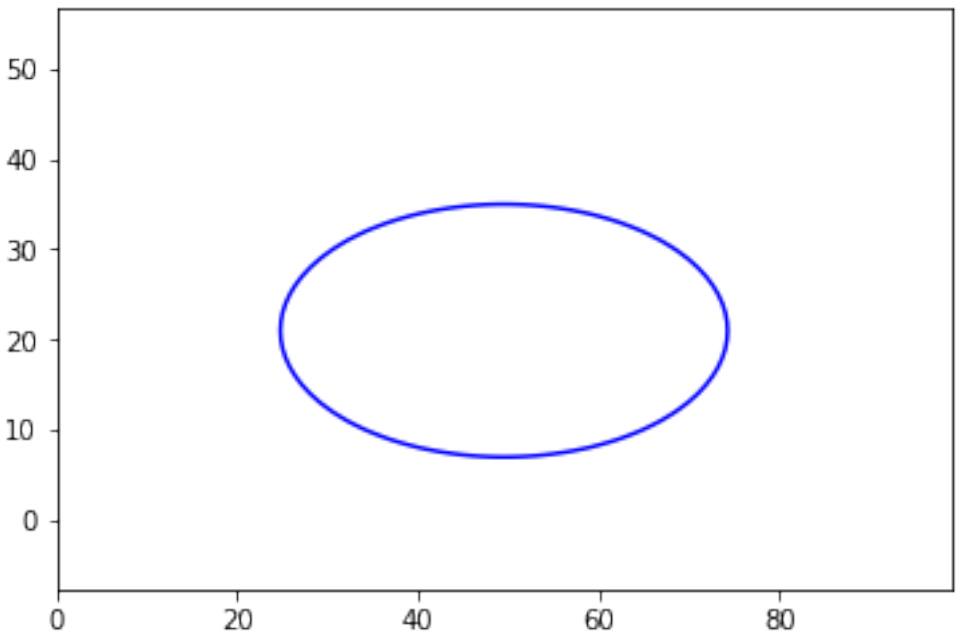
[30]:
```
def f(x,y):
    return x**2 + y**2 - 1
```

[31]:
```
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
```

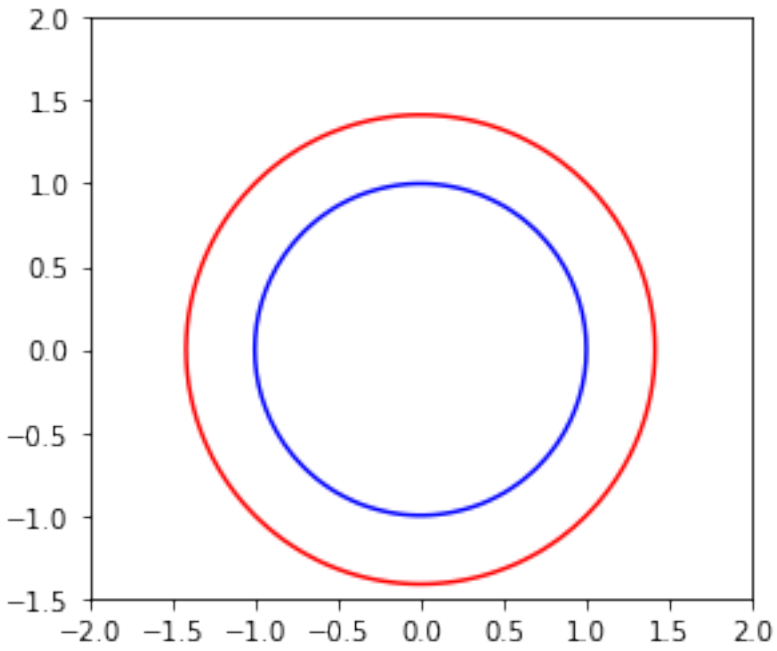
[32]:
```
fig = plt.figure()
ax = fig.gca()
ax.contour(Z, [0], colors='blue') # 0 Höhenlinie in der Farbe blau
```

```
ax.axis('equal')                    # das klappt nicht, da die x,y Koordinaten␣
 ↪nicht bekannt sind
```

[32]: (0.0, 99.0, 0.0, 49.0)



[33]:
```
# besser
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.set_aspect('equal')
```

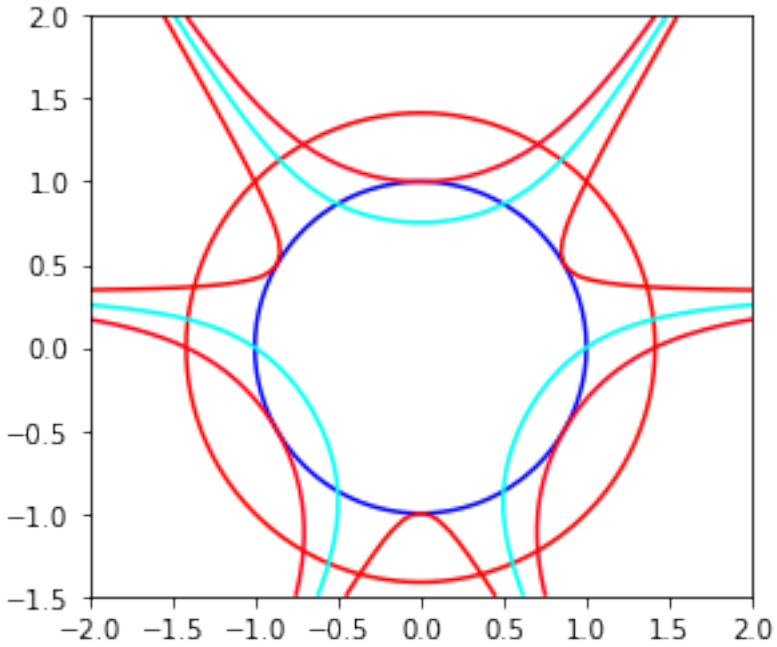


[34]:
```
def g(x, y):
    return x**2+y**2-3*x**2*y+y**3
```

[35]:
```
G = g(X, Y)
```

[36]:
```
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.set_aspect('equal')
ax.contour(X, Y, G, [0, 1, 2], cmap=plt.cm.hsv) # 0, 1 und 2 Höhenlinie mit␣
 ↪Farben aus colormap hsv
```

[36]:
```
<matplotlib.contour.QuadContourSet at 0x7fe8d1873810>
```



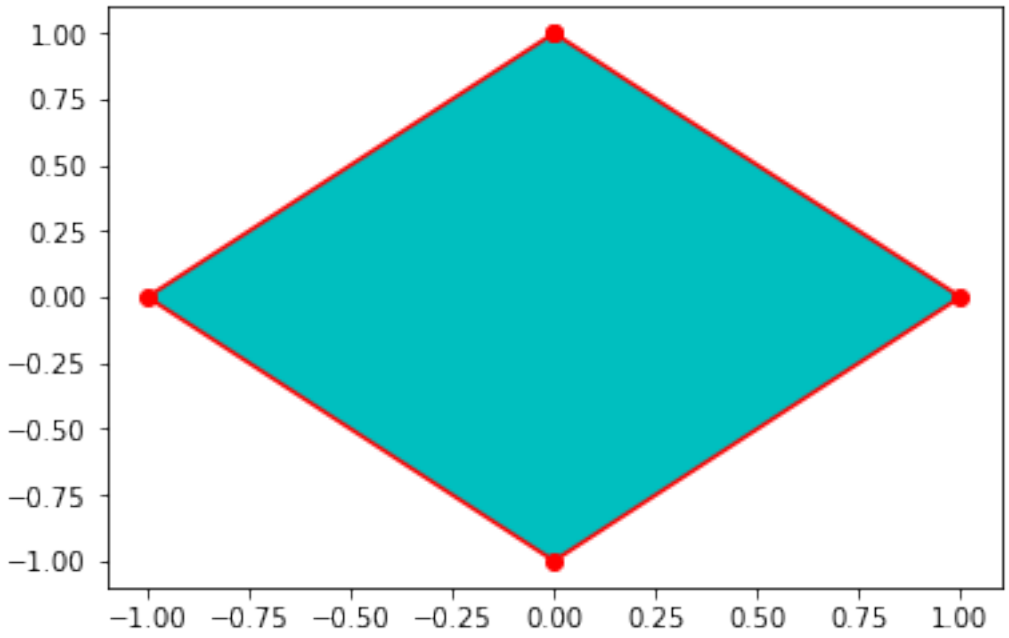
Alle Colormaps findet man auf https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

2.3.4 Ausgefüllte Flächen

[37]:
```
t = np.linspace(0, 2*np.pi, 5)
x = np.sin(t)
y = np.cos(t)
```

[38]:
```
fig = plt.figure()
ax = fig.gca()
ax.fill(x,y,'c')    # ausgefülltes Polygon mit Eckpunten *x, *y
ax.plot(x,y,'ro-')
```

[38]:
```
[<matplotlib.lines.Line2D at 0x7fe8d1471ed0>]
```



2.4 Matplotlib 3D Graphik

2.4.1 Graphen von Funktionen (plot_surface)

[39]:
```
import matplotlib.pyplot as plt
import sympy as sp
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
%matplotlib qt
```

[40]:
```
x = sp.Symbol('x')
y = sp.Symbol('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)
f.subs(x,1.).subs(y,2.), fn(1,2)
```

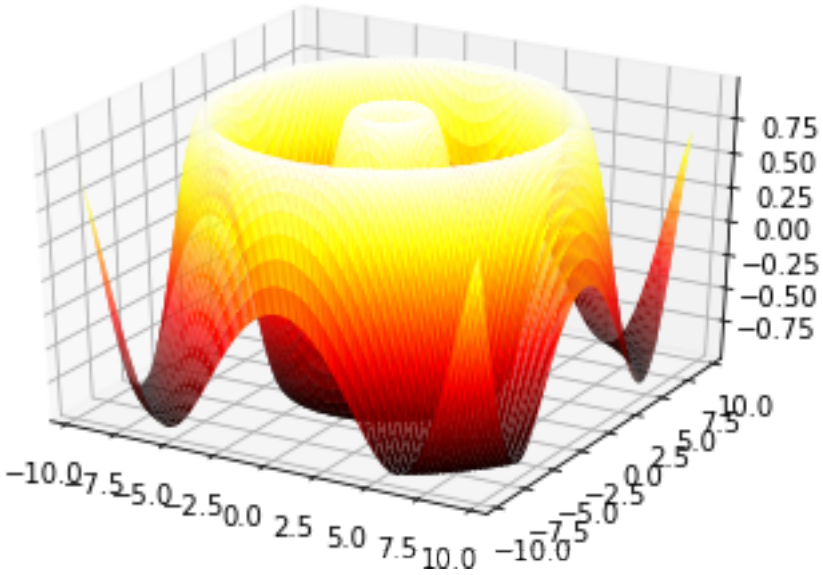
[41]: (0.786749131547214, 0.786749131547214)

[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)

a = np.linspace(-1,1,3)
b = np.linspace(-1,1,5)
a
AA, BB = np.meshgrid(a,b)  # Koordinaten des durch a, b beschriebenen Gitters
display(AA,BB,a,b)
```

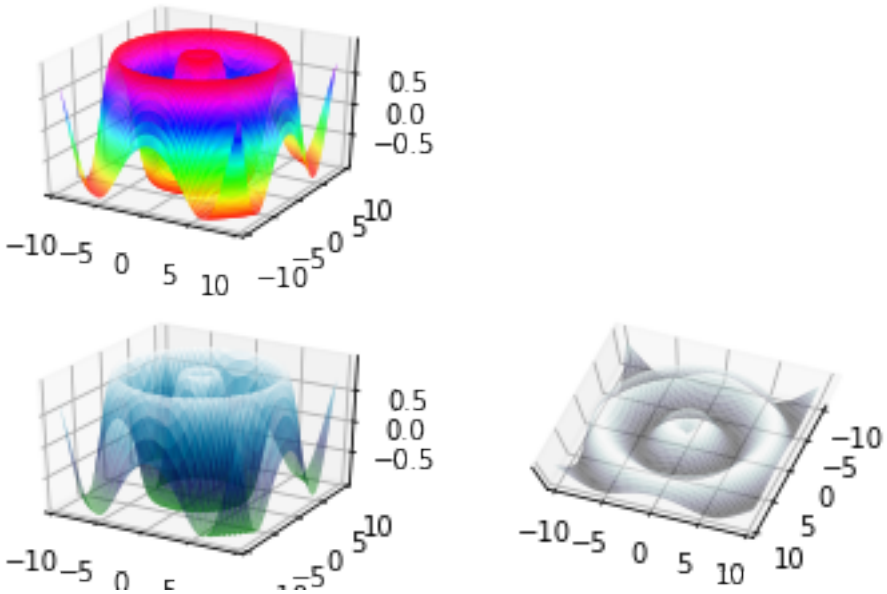
```
array([[-1.,  0.,  1.],
       [-1.,  0.,  1.],
       [-1.,  0.,  1.],
       [-1.,  0.,  1.],
       [-1.,  0.,  1.]])


array([[-1. , -1. , -1. ],
       [-0.5, -0.5, -0.5],
       [ 0. ,  0. ,  0. ],
       [ 0.5,  0.5,  0.5],
       [ 1. ,  1. ,  1. ]])


array([-1.,  0.,  1.])


array([-1. , -0.5,  0. ,  0.5,  1. ])
```

[43]:
```
fig = plt.figure()                              # Bild
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=1);
```



[44]:
```
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([])
```

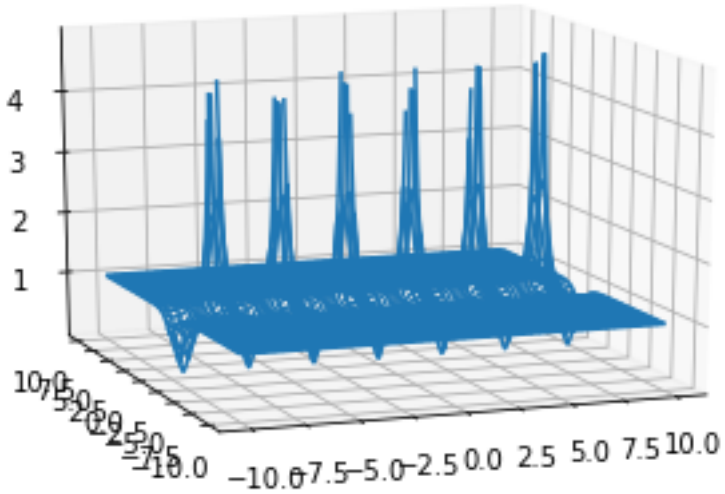
[44]: []



[60]:
```
f = abs(sp.tan(x+sp.I*y))   # Betrag des Tangens in der komplexen Ebene
fn = sp.lambdify((x,y),f)
```

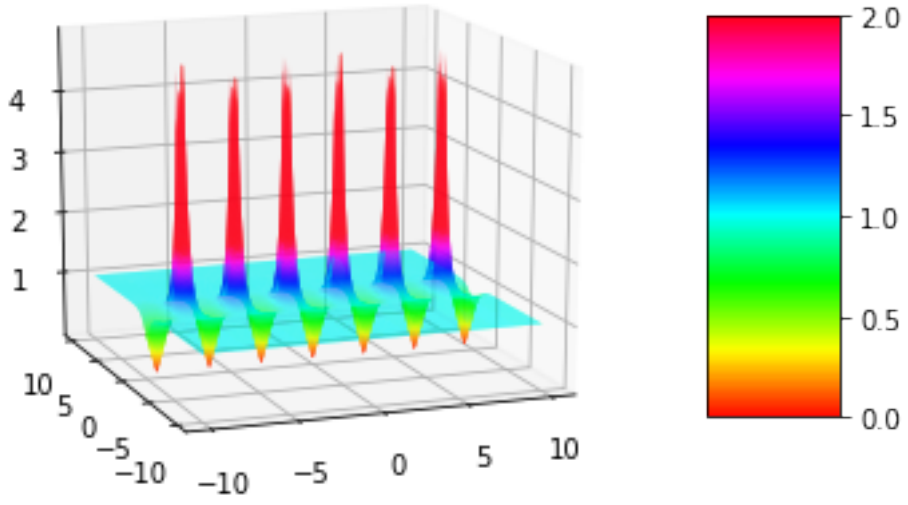
[61]:
```
xn = np.linspace(-10, 10, 400)
yn = xn
X, Y = np.meshgrid(xn,  yn) # Gitter in der komplexen Ebene X: Realteil Y:␣
 ↪Imaginaerteil
Z=fn(X,Y)
Z[Z>5]=np.nan  # Werte größer als 5 werden ignoriert
```

[62]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z)
ax.view_init(15,-110)
```



[63]:
```
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=.8,aspect=3)
```

[63]:
```
<matplotlib.colorbar.Colorbar at 0x7fe8b2caab90>
```



[64]:
```
fig.colorbar(surf,shrink=0.5)
```

[64]:
```
<matplotlib.colorbar.Colorbar at 0x7fe8b7e117d0>
```

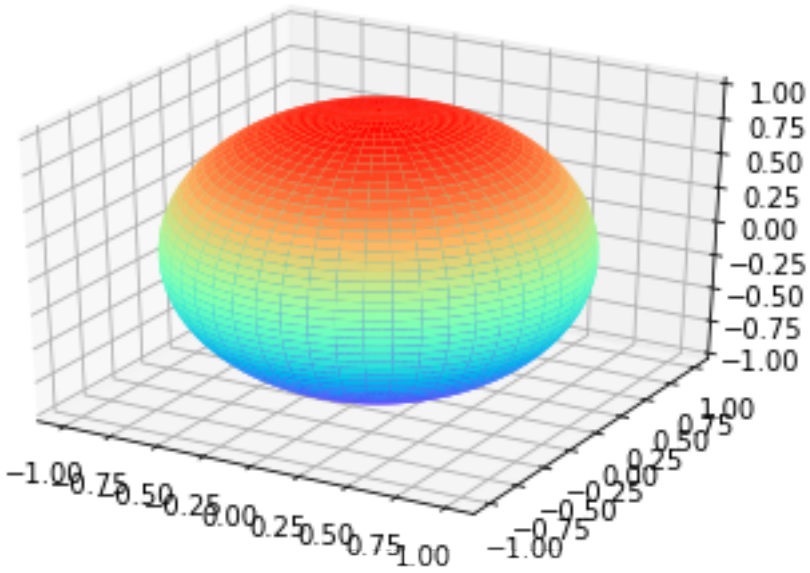
[50]:
```
fig.colorbar(surf,aspect=5)
```

[50]:
```
<matplotlib.colorbar.Colorbar at 0x7fe8bdcbb3d0>
```

2.4.2 Parametrische Flächen (plot_surface)

[65]:
```
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)
```

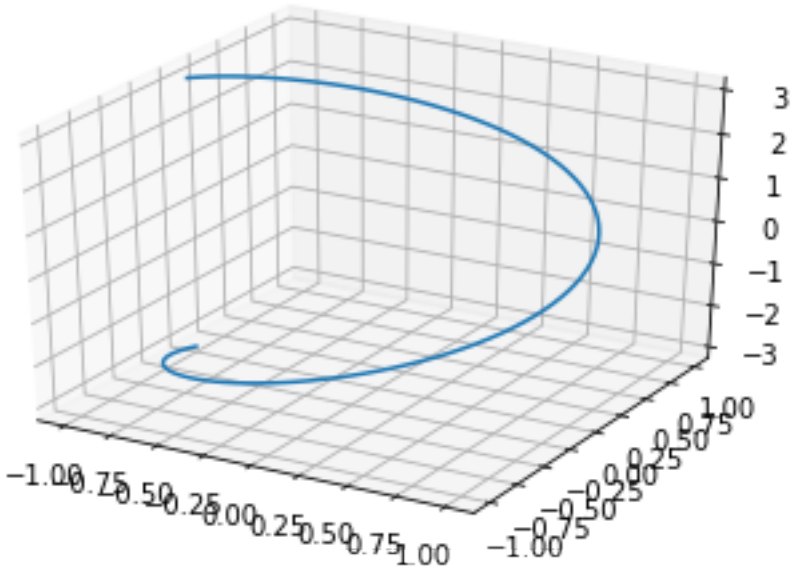
[65]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7fe8b7fee0d0>
```



2.4.3 Raumkurven in Parameterdarstellung (plot)

[66]:
```
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)
```

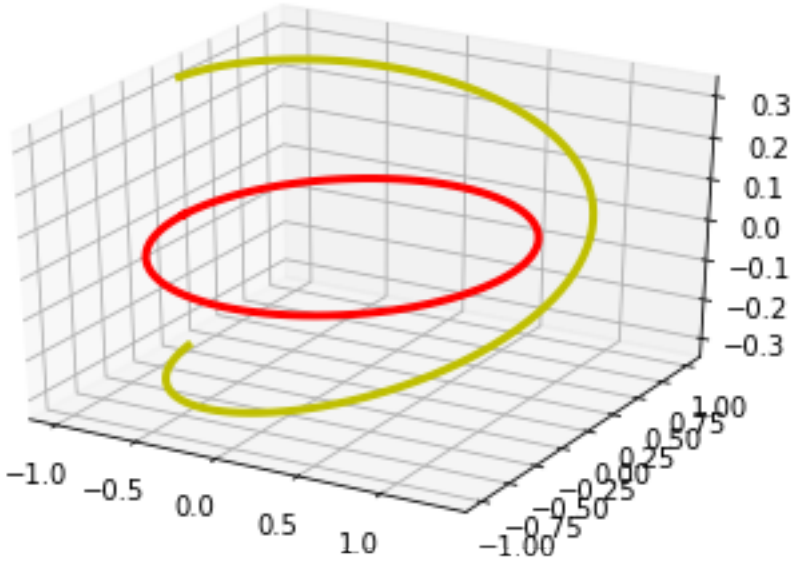
[66]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x7fe8b2c87190>]



ein gewundenes Band

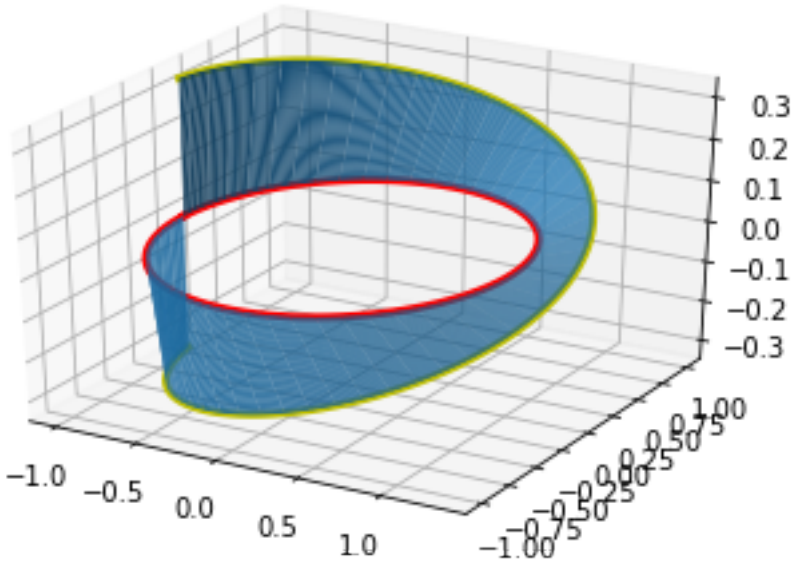
[67]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

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)+np.cos(xn/2)/3, np.sin(xn), np.sin(xn/2)/3, 'y', linewidth=3)
```

[67]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x7fe8a7522b50>]



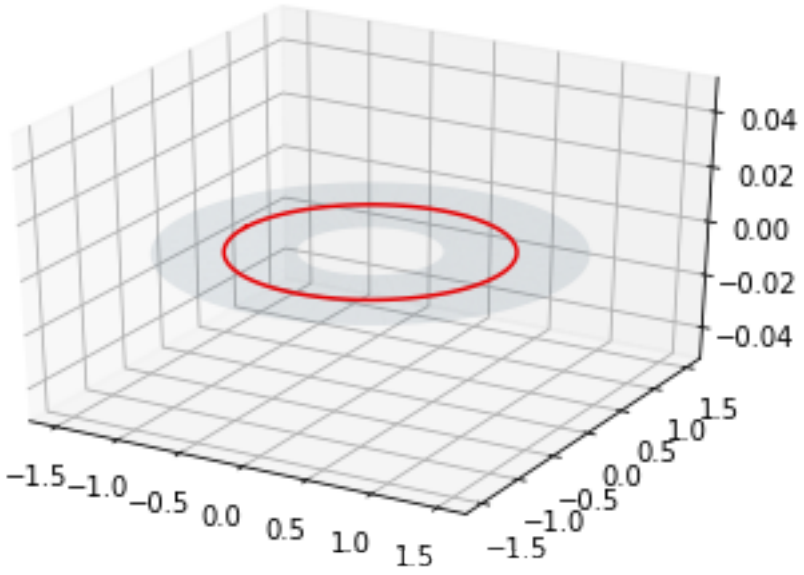
[68]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

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)
```

[68]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7fe8a7466790>



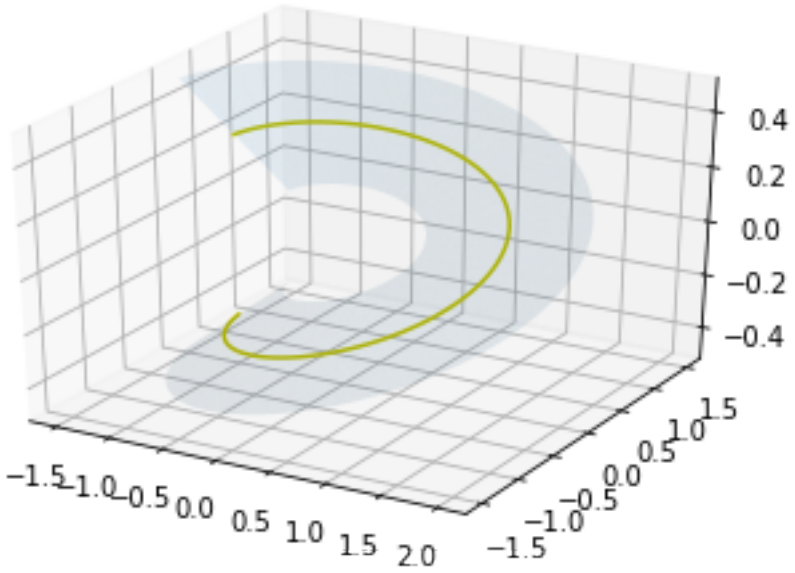
[69]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

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')
```

[69]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x7fe8bc690150>]



[70]:
```
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')
```

[70]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x7fe8a6af0b10>]
```



2.4.4 Implizit gegebene Flächen Isoflächen (marching_cube + plot_trisurf)

Alle Punkte $(x, y, z) \in \mathbb{R}^3$ derart, dass

$$f(x, y, z) = 0 \quad \text{für} \quad f : \mathbb{R}^3 \to \mathbb{R}$$

[71]:
```
from skimage import measure
from mpl_toolkits.mplot3d import Axes3D

xn=np.linspace(-2,2,100)
scale = lambda x: xn[0]+(xn[-1]-xn[0])*x/(len(xn)-1)

X,Y,Z = np.meshgrid(xn,xn,xn)
vol = X**4+Y**4+Z**4+1000*(X**4+Y**4)*(X**4+Z**4)*(Y**4+Z**4)-10
#vol = X**2+Y**2+Z**2-4                    # Kugel
#vol = (np.sqrt(X**2+Y**2)-1)**2+Z**2-1/4  # Torus

verts, faces, _, __ = measure.marching_cubes_lewiner(vol, 0)
# 0-Isofläche beschreiben durch Dreiecksflächen

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_trisurf(scale(verts[:,0]), scale(verts[:,1]), scale(verts[:,2]), \
                triangles = faces, cmap = 'hsv', alpha=0.5)

ax.set_xlim3d((xn[0], xn[-1]))
```

```
ax.set_ylim3d((xn[0], xn[-1]))
ax.set_zlim3d((xn[0], xn[-1]))
```

[71]: (-2.0, 2.0)