lektion4

November 4, 2021

Table of Contents

1 Lektion 4

1.1 For Schleife und List Comprehension

[1]:
```
L = []
for i in range(6):
    L.append(i**2)
L
```

[1]:
```
[0, 1, 4, 9, 16, 25]
```

[2]:
```
L = [i**2 for i in range(6)]  # Liste der Quadratzahlen
L
```

[2]:
```
[0, 1, 4, 9, 16, 25]
```

[3]:
```
L = [i**2 for i in range(6) if i%2 == 1] # Liste der ungeraden Quadratzahlen
L
```

[3]:
```
[1, 9, 25]
```

[4]:
```
L = [i**2 if i%2==1 else -i for i in range(6)]
L
```

[4]:
```
[0, 1, -2, 9, -4, 25]
```

1.2 Python Funktionen

[5]:
```
pf = lambda x : x**2
pf(2)
```

[5]:
```
4
```

das gleiche ausführlicher

[6]:
```
def mysqr2(x):
    """ berechnet x Quadrat"""
    y = x**2
    return y
```

[7]:
```
mysqr2(2)
```

[7]:
```
4
```

[8]:
```
?mysqr2
```

[9]:
```
def mypow(x, n=2):
    """ Berechnet x**n und falls n nicht gegeben ist das Quadrat von x"""
    y = x**n
    return y
```

[10]:
```
mypow(3)
```

[10]:
```
9
```

[11]:
```
mypow(3, 3)
```

[11]:
```
27
```

Achtung

[12]:
```
def f(a, L=[]):
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(3))

print(f(4, [2, 3, 1]))
```

```
[1]
[1, 2]
[1, 2, 3]
[2, 3, 1, 4]
```

Der default wird nur einmal ausgewertet und [] ist ein veränderbares Objekt.

Eine Lösung ist:

[13]:
```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(4, [2, 3, 1]))
```

```
[1]
[2]
[2, 3, 1, 4]
```

[14]:
```
b = [7, 2]
def f(a):
    c = b[0] + a
    b[1] = 1
    return c
print(b, f(2))
```

```
[7, 1] 9
```

Ein Funktion kann auf Variablen zugreifen, die auf 'höherer' Ebene definiert wurden. Und im Fall von Listen diese auch verändern.

[15]:
```
b = 7
def f(a):
    b = 2
    c = b + a
```

```
    return c
print(b, f(2))
```

```
7 4
```

[16]:
```
b = 7
def f(a):
    b = 21
    c = b + a
    return c
print(b, f(2))
```

```
7 23
```

Hier ist b zwei verschiedene Variablen. Einmal eine Variable, die auf die Zahl 7 verweist und andererseits eine lokale Variable innerhalb der Funktion f, die den Wert 21 hat, die auf die Variable b, die eine Ebene höher existiert, keine Auswirkung hat.

[17]:
```
b = 7
def f(a):
    erg = b + a
    return erg
```

1.3 Kopieren von Listen

[18]:
```
a = [1, 2]
b = [3, 4, a]
b
```

[18]:
```
[3, 4, [1, 2]]
```

[19]:
```
a[0] = 11
```

[20]:
```
b
```

[20]:
```
[3, 4, [11, 2]]
```

[21]:
```
bb = [3, 4, a.copy()]
bb
```

[21]:
```
[3, 4, [11, 2]]
```

[22]:
```
a[0] = 22
```

[23]:
```
bb
```

[23]:
```
[3, 4, [11, 2]]
```

[24]:
```
b
```

[24]:
```
[3, 4, [22, 2]]
```

1.4 Sympy Funktionen

[25]:
```
from sympy import *
init_printing()
x, y, z = symbols('x y z')
f = Lambda(x, x**2) # vgl. lambda x : expr
f(23)
```

[25]:
$$529$$

[26]:
```
f = Lambda((x, y, z), x*y+y-2*z**2)
f
```

[26]:
$$\left((x,\ y,\ z) \mapsto xy + y - 2z^{2} \right)$$

[27]:
```
f(1, 2, 3)
```

[27]:
$$-14$$

[28]:
```
param = x, y, z
f = Lambda(param, x+y-z)
type(param), f, f(*param) # * ist hier der "argument unpacking operator"
```

[28]:
```
(tuple, Lambda((x, y, z), x + y - z), x + y - z)
```

[29]:
```
f = Function('f')
g = Function('g')
```

[30]:
```
f(g(x))
```

[30]:
$$f(g(x))$$

[31]:
```
k = f(x)+g(1/x)
k
```

[31]:
$$f(x) + g\left(\frac{1}{x}\right)$$

[32]:
```
k.diff(x)
```

[32]:
$$\frac{d}{dx} f(x) - \frac{\frac{d}{d\xi_1} g(\xi_1)\Big|_{\xi_1=\frac{1}{x}}}{x^{2}}$$

1.5 Numpy Funktionen

[33]:
```
import numpy as np
```

[34]:
```
xn = np.linspace(0 , 1, 4)
xn
```

[34]:
```
array([0.        , 0.33333333, 0.66666667, 1.        ])
```

[35]:
```
np.sin(xn)
```

[35]:
```
array([0.        , 0.3271947 , 0.6183698 , 0.84147098])
```

[36]:
```
np.sin(np.pi*xn)
```

[36]:
```
array([0.00000000e+00, 8.66025404e-01, 8.66025404e-01, 1.22464680e-16])
```

1.6 Lamdifizierung (sympy -> numpy/scipy)

[37]:
```
f = x**2 * sin(x)
f
```

[37]:
$$x^2 \sin(x)$$

[38]:
```
fn = lambdify(x, f)
```

[39]:
```
xn = np.linspace(0, 11, 5)
fn(xn)
```

[39]:
```
array([   0.        ,    2.88631125,  -21.34259485,   62.79474906,
       -120.99881499])
```

[40]:
```
f = Integral(exp(-x**2), (x, 1, y))
F = f.doit()
F            # erf: gaußsche Fehlerfunktion (engl. error function)
```

[40]:
$$\frac{\sqrt{\pi}\,\mathrm{erf}(y)}{2} - \frac{\sqrt{\pi}\,\mathrm{erf}(1)}{2}$$

[41]:
```
Fn = lambdify(y, F)
Fn(2)
```

[41]:
$$0.135257257949995$$

[42]:
```
%%timeit
Fn(2)
```

```
2.93 µs ± 31.7 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

[43]:
```
%%timeit
F.subs(y, 2).n()
```

```
380 µs ± 9.77 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

[44]:
```
Fn(xn)
```

[44]:
```
array([-0.74682413,  0.13931362,  0.13940279,  0.13940279,  0.13940279])
```

[45]:
```
f = sin(pi/2*x**2)
F = Integral(f, (x, 0, y))
F
```

[45]:
$$\int_0^y \sin\left(\frac{\pi x^2}{2}\right) dx$$

[46]:
```
F.doit().simplify()
```

[46]:
$$S\left(y\right)$$

[47]:
```
type(_)
```

[47]:
```
fresnels
```

[48]:
```
fresnels(2).n()
```

[48]:
0.343415678363698

[49]:
```
Fn = lambdify(y, F.doit().simplify())
Fn(2)
```

[49]:
0.343415678363698

Mehr zu FresnelS hier: https://en.wikipedia.org/wiki/Fresnel_integral

[50]:
```
import sympy as sp
import numpy as np
```

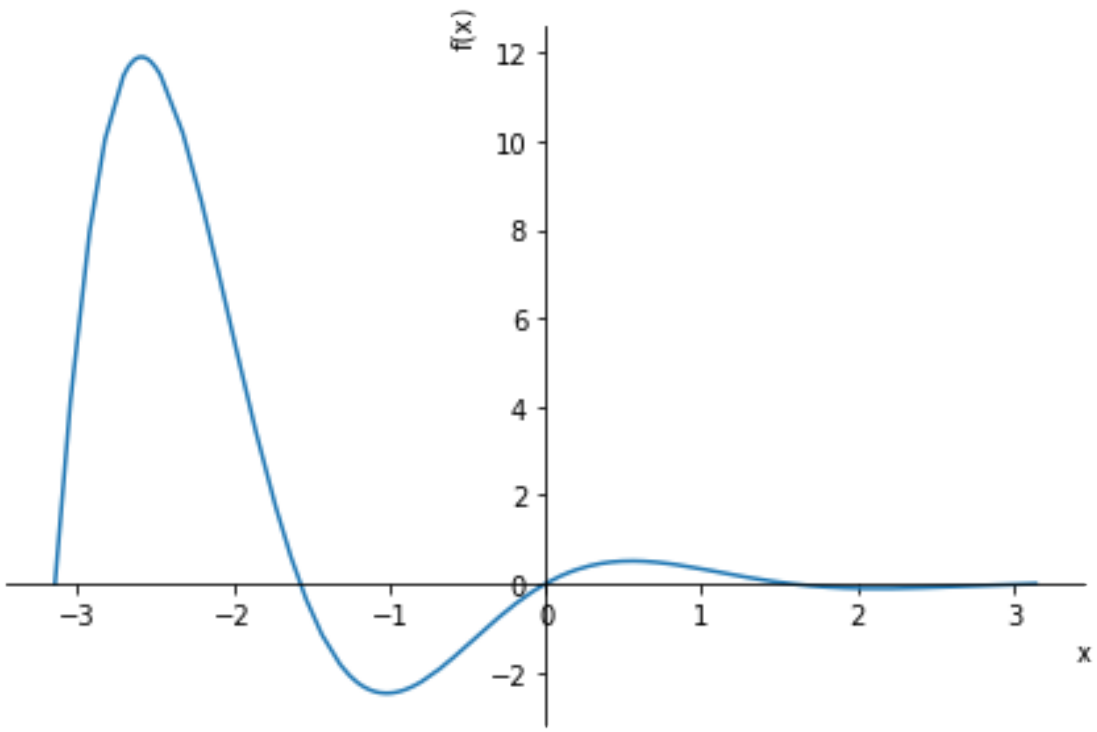
1.7 Sympy Graphik 2D

1.7.1 Einfache Graphen von Funktionen

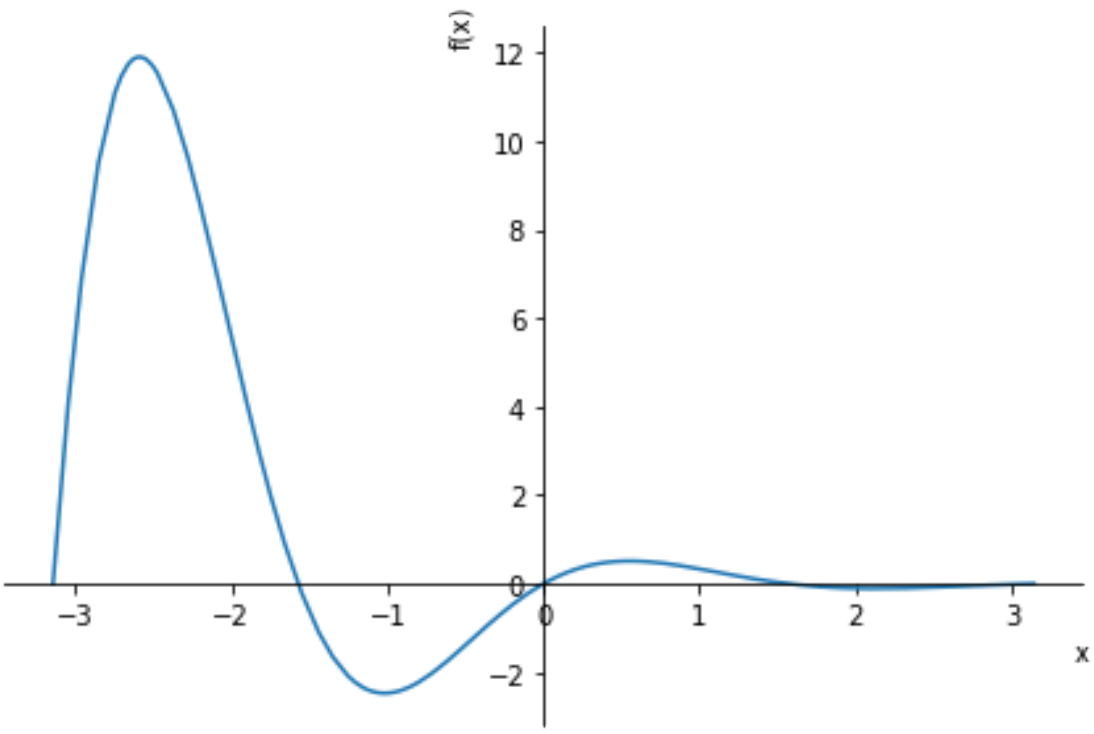
Graph: Alle Punkte (x, y) mit

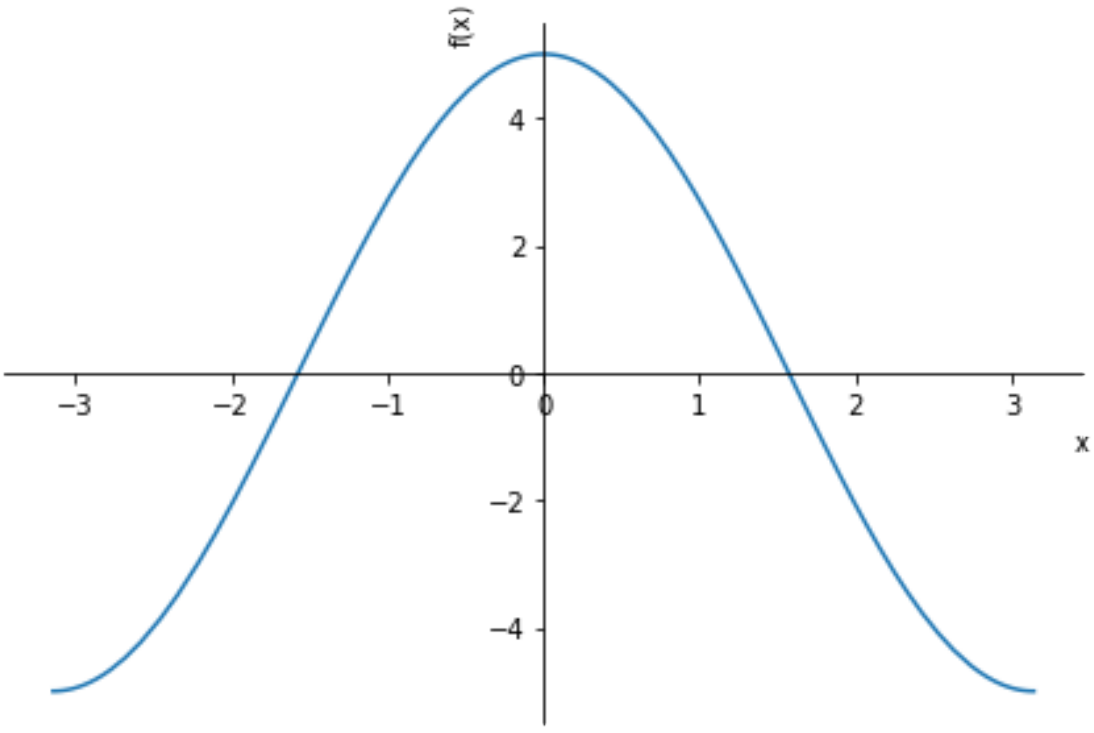
$$y = f(x)$$

[51]:
```
x = sp.symbols('x')
f = sp.exp(-x)*sp.sin(2*x)
p1 = sp.plot(f, (x, -sp.pi, sp.pi))
```

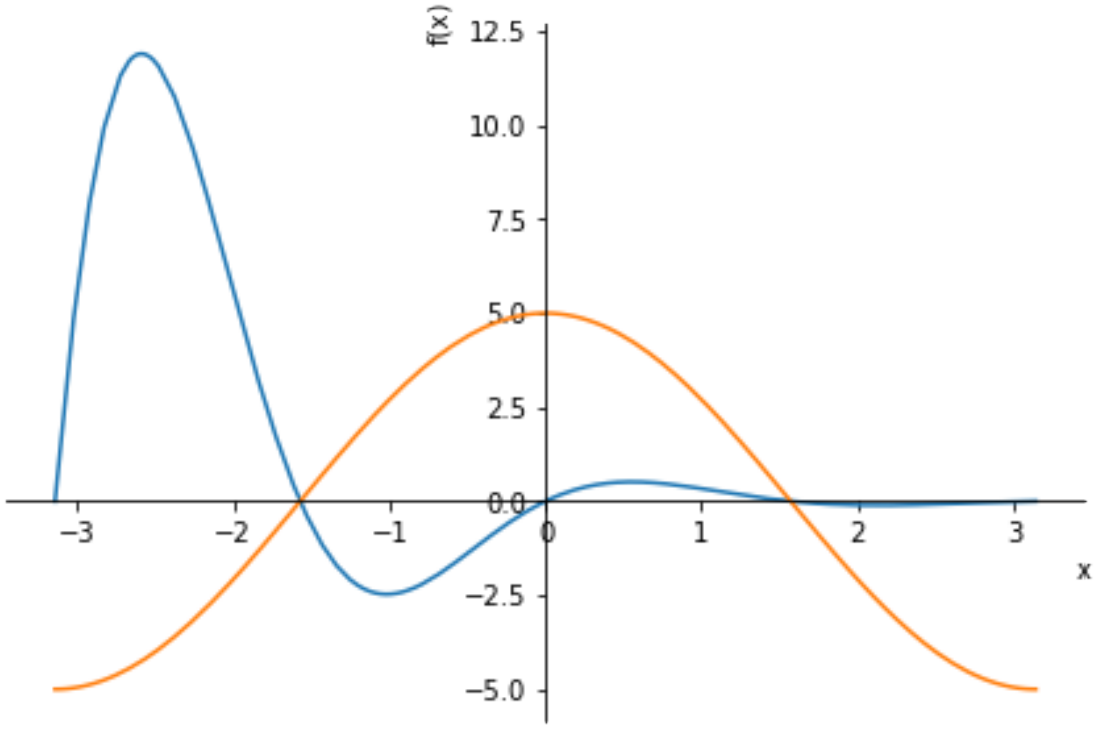


[52]:
```
x = sp.symbols('x')
f = sp.exp(-x)*sp.sin(2*x)
p1 = sp.plot(f,(x, -sp.pi, sp.pi))
p2 = sp.plot(5*sp.cos(x), (x, -sp.pi, sp.pi))
```

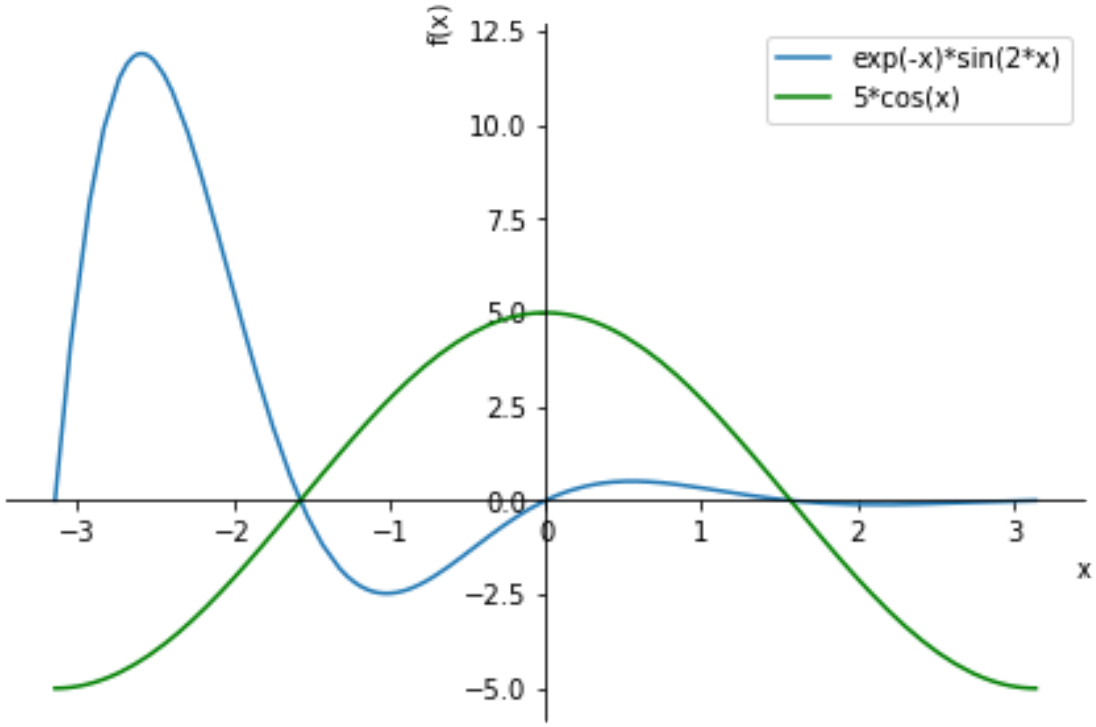

f(x)
12
10
8
6
4
2
0
−2
−3
−2
−1
0
1
2
3
x


f(x)
4
2
0
−2
−4
−3
−2
−1
0
1
2
3
x

[53]:
```
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.show()
```



[54]:
```
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='green'
p1.show()
```



1.7.2 Implizit gegebene Kurven

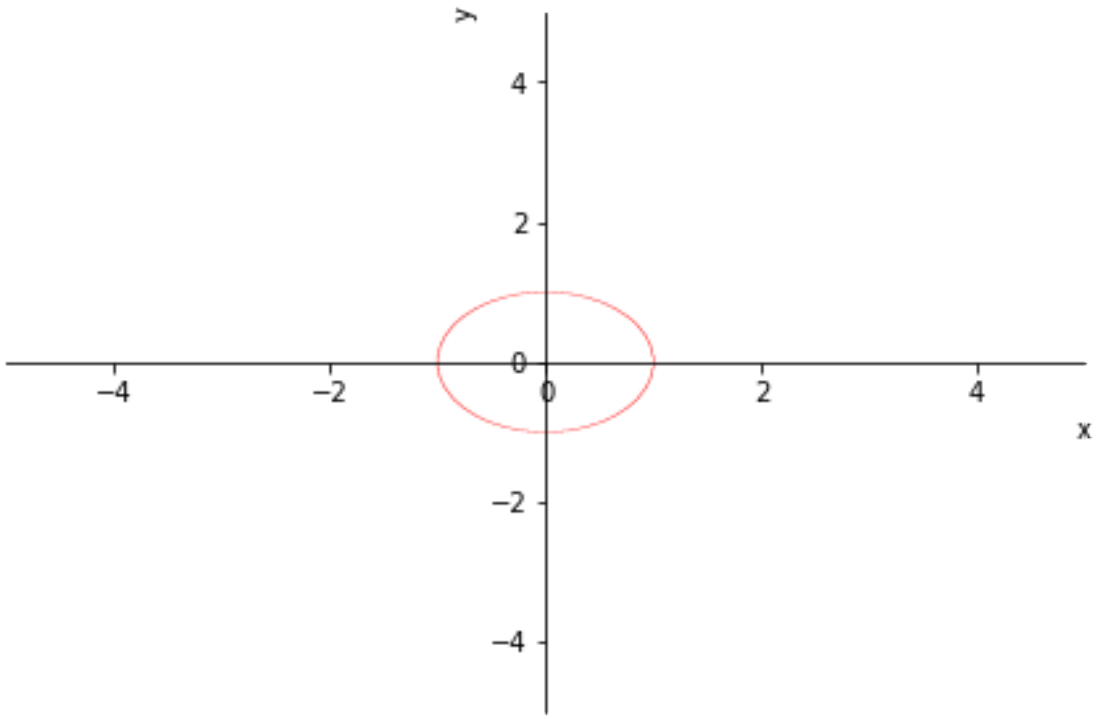
Alle Punkte (x, y) mit

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

[55]:
```
sp.Eq(x**2-y, 0) # Gleichung in Sympy, das mathematische =
```

[55]: $x^2 - y = 0$

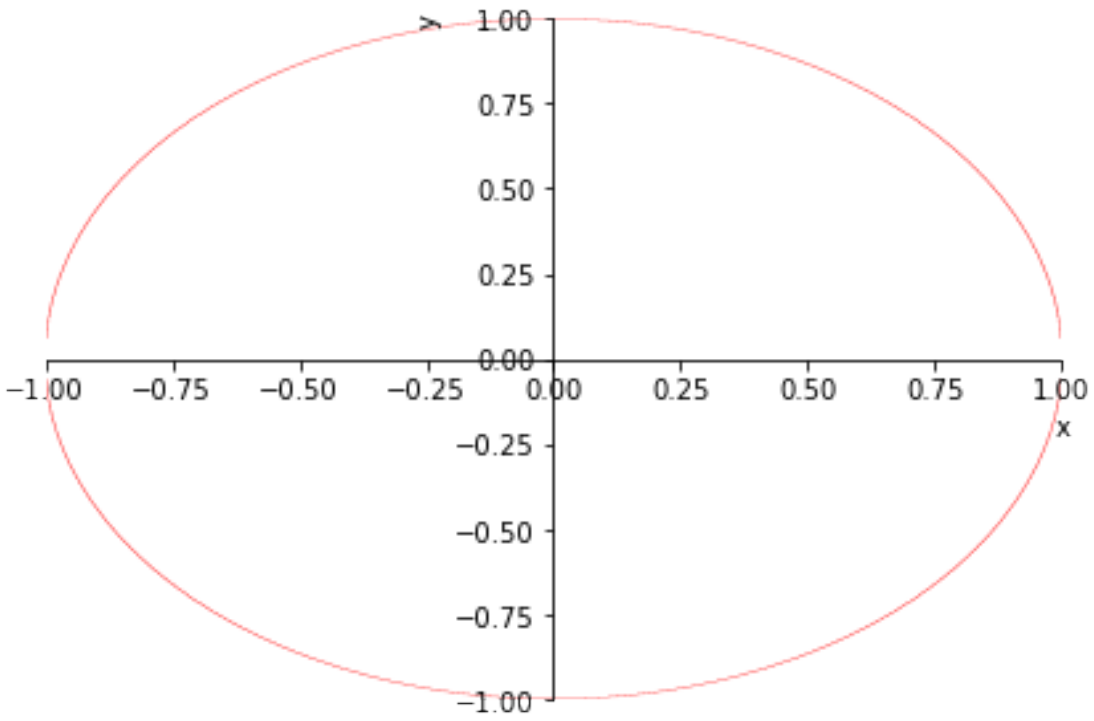
[56]:
```
x, y, z = sp.symbols('x y z')
p3 = sp.plotting.plot_implicit(sp.Eq(x**2+y**2, 1), line_color='red')
```



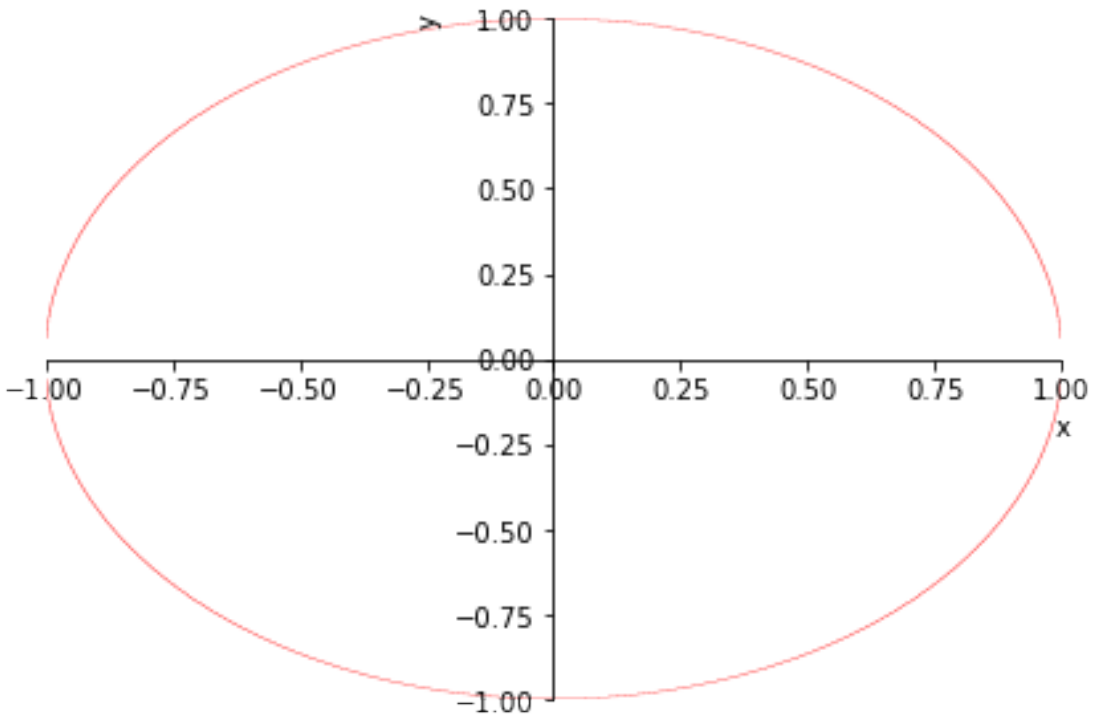
Implizit gegebene Kurven

[57]:
```
#%matplotlib notebook # Graphik backend wechseln
#%matplotlib inline # Graphik backend wechseln

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[0].set_aspect('equal') # funktioniert nicht im inline Modus
```

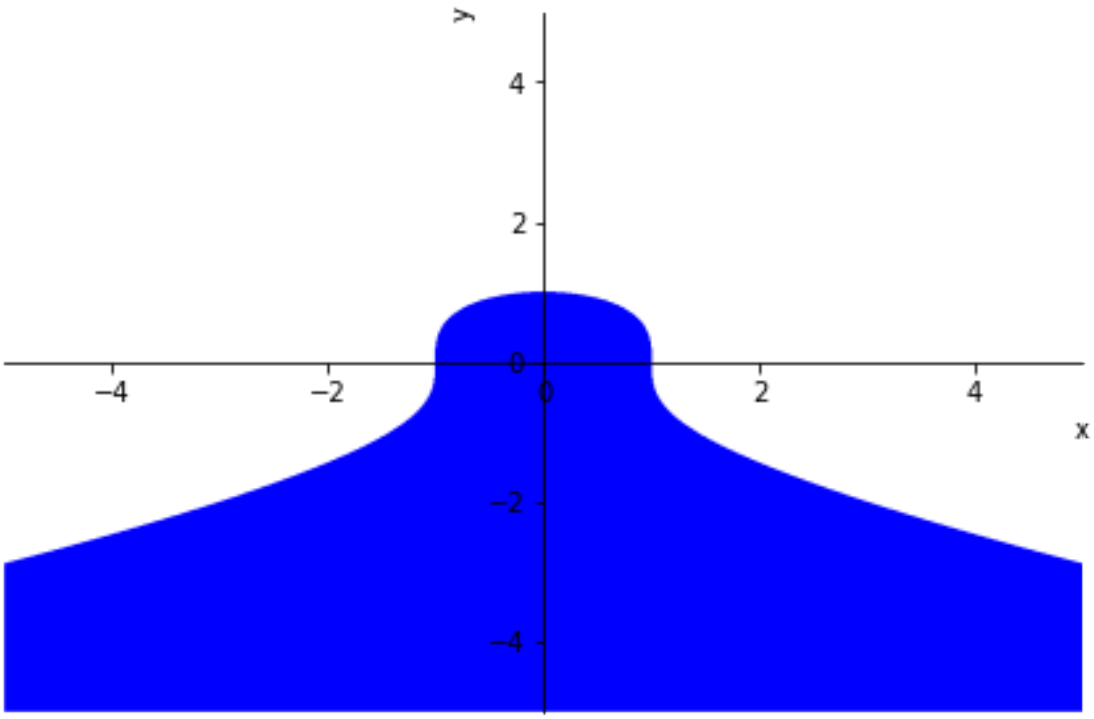


[58]:
```
x, y, z = sp.symbols('x y z')
p3 = sp.plotting.plot_implicit(sp.Eq(x**2+y**2, 1), (x, -1, 1), (y, -1, 1),␣
 ↪line_color='red')
ax = p3._backend.ax
ax[0].set_aspect('equal') # funktioniert nicht im inline Modus
```



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

[59]:
```
sp.plotting.plot_implicit(sp.Le(x**2+y**3, 1));
```

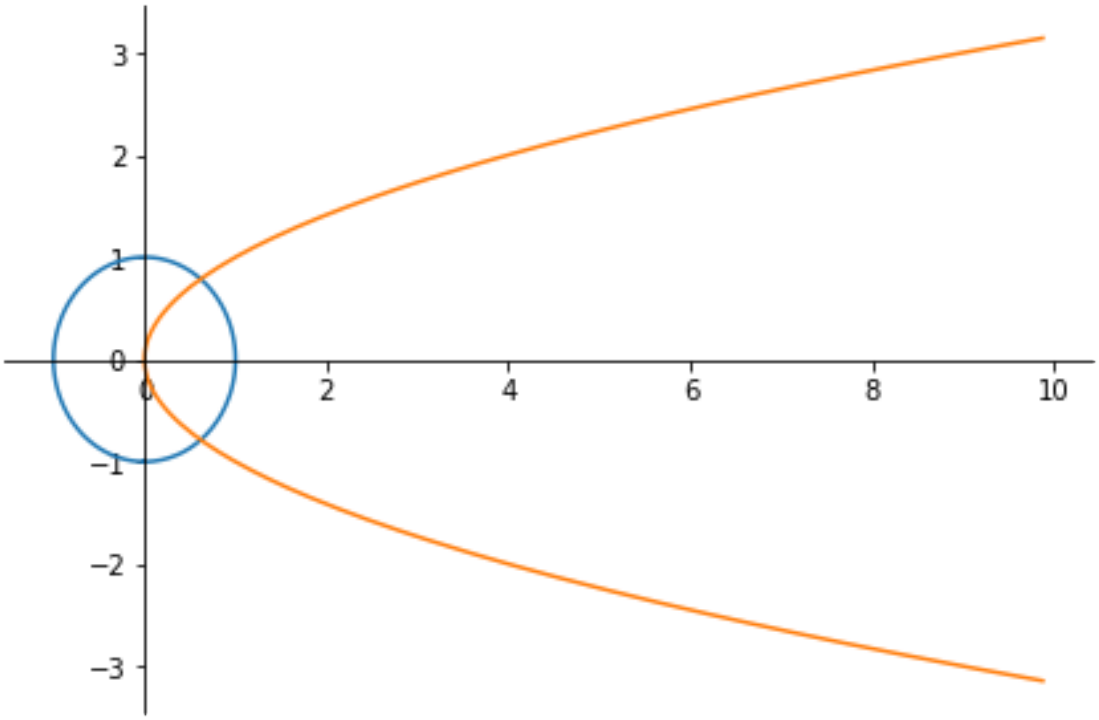


1.7.3 Kurven in Parameterdarstellung in der Ebene

Punkte (x, y) in der Ebene mit

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

[60]:
```
t = sp.symbols('t')
p1 = sp.plotting.plot_parametric((sp.sin(t), sp.cos(t)), (t**2, t), (t, -sp.pi,␣
↪sp.pi))
ax = p1._backend.ax
ax[0].set_aspect('equal')
```



Was gibt es noch?

[61]:
```
?plotting.
```

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

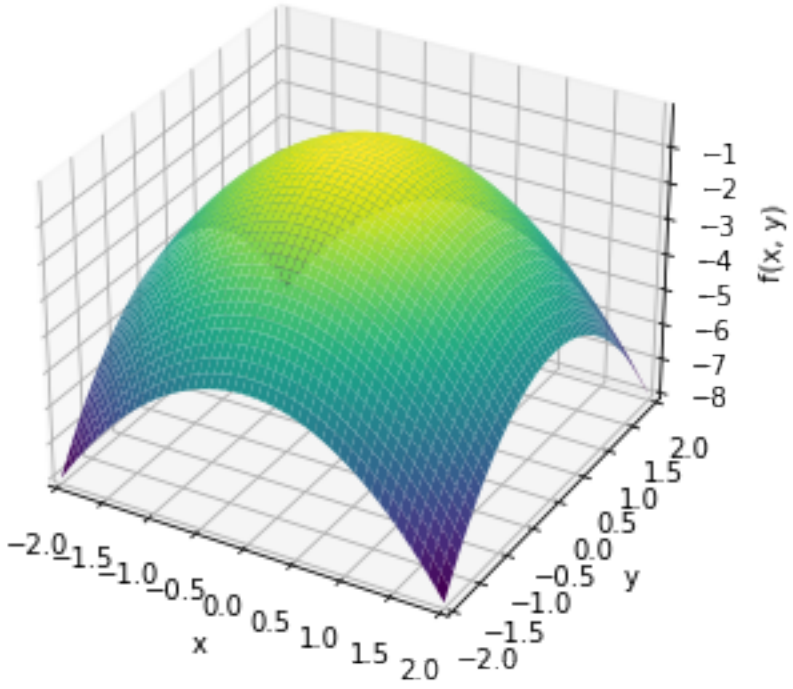
1.8 Sympy Graphik 3D

1.8.1 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 $x \in (x_1, x_2),\ y \in (y_1, y_2)$

[62]:
```
sp.plotting.plot3d((-x**2-y**2), (x, -2, 2), (y, -2, 2));
```



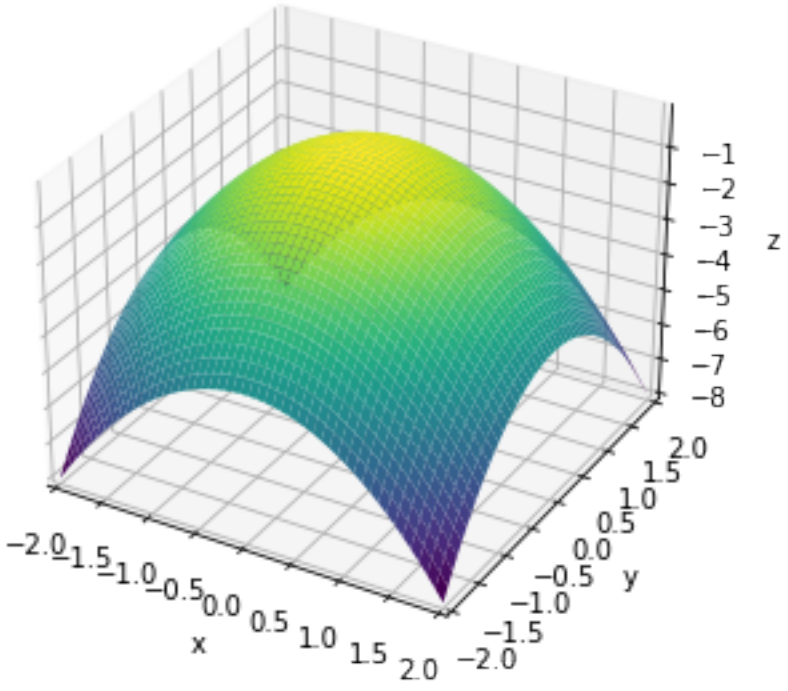
1.8.2 Parametrische Flächen

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 $f_j : \mathbb{R}^2 \to \mathbb{R}$.

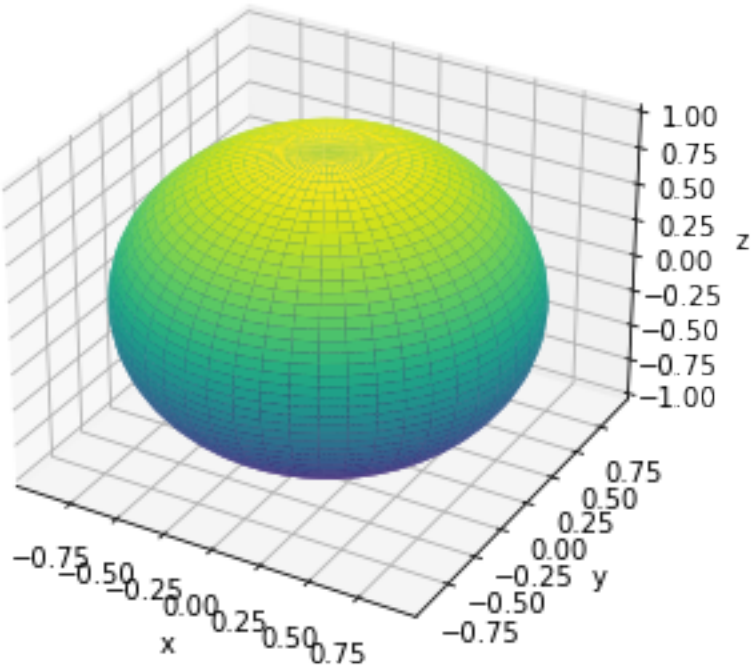
[63]:
```
u, v = sp.symbols('u v')
sp.plotting.plot3d_parametric_surface(u, v, -u**2-v**2, (u, -2, 2), (v, -2, 2))
```



[63]: <sympy.plotting.plot.Plot at 0x7f0198c49490>

[64]:
```
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')
```



[65]:
```
#alternativ: (unperfekte Lösung)
ax.set_xlim3d((-1.2, 1.2))
ax.set_ylim3d((-1.2, 1.2))
ax.set_zlim3d((-1, 1))
```

[65]: $(-1.0,\ 1.0)$

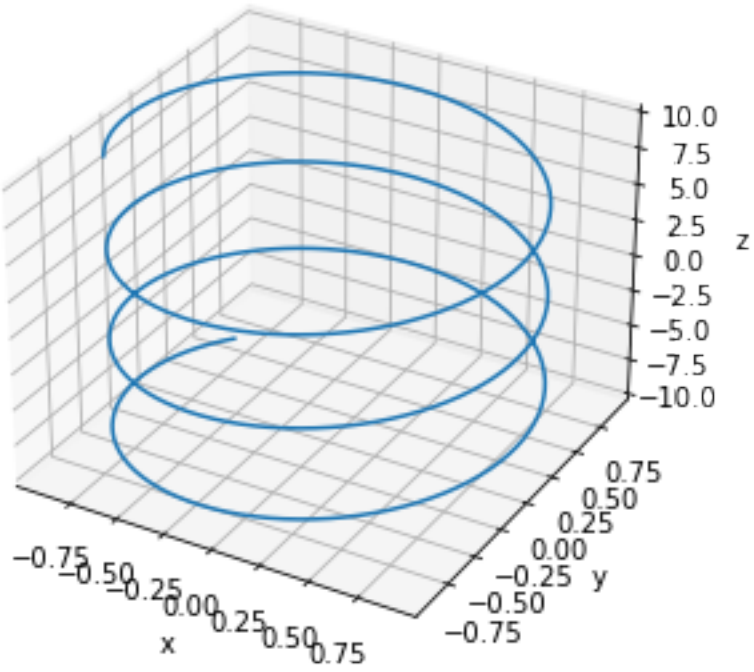
1.8.3 Kurven in Parameterdarstellung

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.

[66]:
```
sp.plotting.plot3d_parametric_line(sp.cos(u), sp.sin(u), u, (u, -10, 10))
```



[66]: <sympy.plotting.plot.Plot at 0x7f0198e5e490>