lektion12

January 13, 2022

Inhalt

1 Gradient, Hessematrix (Wiederholung)

1.1 Extrema (Wiederholung)

1.2 Tangenten

1.3 Gradient als Vektorfeld (quiver) Höhenlinien (contour)

2 Differentialgleichungen

2.1 erstes Beispiel

2.2 Inhomogene lineare DGL

2.3 Variation der Konstanten Formel

2.4 Lösung mit einem Reihenansatz

2.5 Logistische Gleichung

1 Lektion 12

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
init_printing()
%matplotlib inline
```

1.1 Gradient, Hessematrix (Wiederholung)

[2]:
```
x, y, z = symbols('x y z')
var = (x, y, z)
```

[3]:
```
f = Function('f')
f(*var)
```

[3]: $f(x, y, z)$

[4]:
```
G = Matrix([f(*var)]).jacobian(var)
G
```

[4]: $\left[\frac{\partial}{\partial x}f(x,y,z) \quad \frac{\partial}{\partial y}f(x,y,z) \quad \frac{\partial}{\partial z}f(x,y,z)\right]$

[5]:
```
H = hessian(f(*var), var)
H
```

[5]: $\begin{bmatrix} \frac{\partial^2}{\partial x^2}f(x,y,z) & \frac{\partial^2}{\partial y\partial x}f(x,y,z) & \frac{\partial^2}{\partial z\partial x}f(x,y,z) \\ \frac{\partial^2}{\partial y\partial x}f(x,y,z) & \frac{\partial^2}{\partial y^2}f(x,y,z) & \frac{\partial^2}{\partial z\partial y}f(x,y,z) \\ \frac{\partial^2}{\partial z\partial x}f(x,y,z) & \frac{\partial^2}{\partial z\partial y}f(x,y,z) & \frac{\partial^2}{\partial z^2}f(x,y,z) \end{bmatrix}$

[6]:
```
#Matrix([f(*var)]).hessian(var)
```

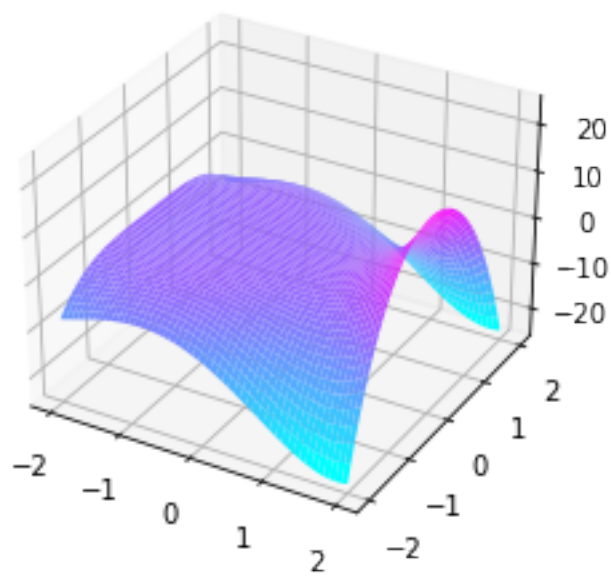
[7]:
```
f = 1/2 * x**4  - x**2*y**2 - 1/2*y**4 + x**3 -2*x*y**2
```

[8]:
```
fn = lambdify((x, y), f)
```

[9]:
```
gn = np.linspace(-2, 2)
X, Y = np.meshgrid(gn ,gn)
F = fn(X, Y)
```

[10]:
```
fig = plt.figure(1)
fig.clf()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, F, linewidth=0, cmap=plt.cm.cool);
ax.set_zlim(-25, 25)
```

[10]: $(-25.0,\ 25.0)$

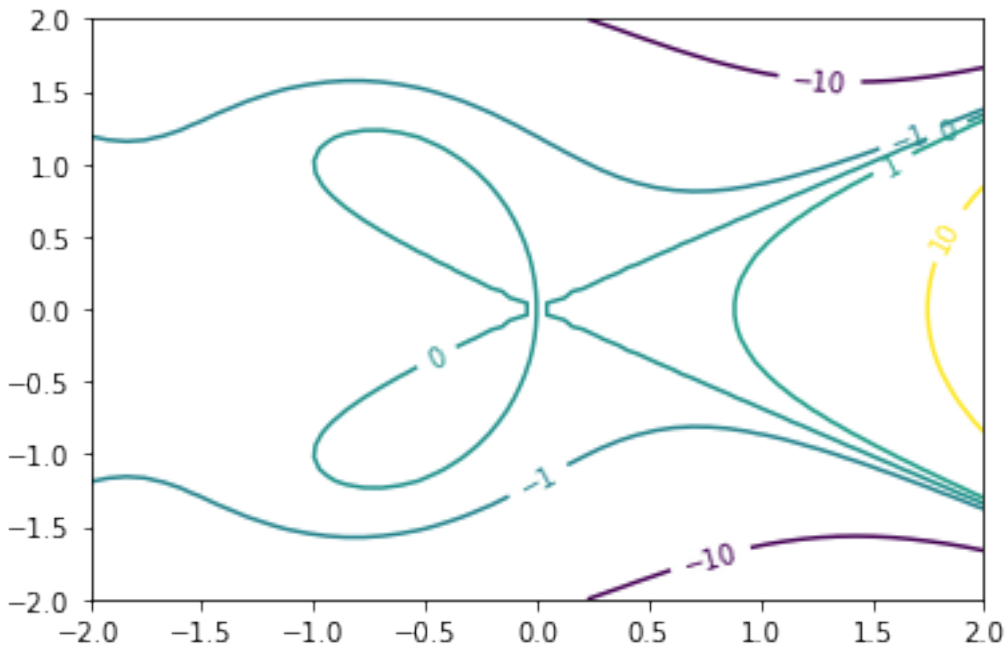


[11]:
```
ax = fig.gca()
ax.contour(X, Y, F, np.array([-10, -1, 0, 1, 10]), offset=-25)
```

[11]: <matplotlib.contour.QuadContourSet at 0x7f550b792880>

[12]:
```
fig = plt.figure(2)
fig.clf()
ax = fig.gca()
cs = ax.contour(X, Y, F, np.array([-10, -1, 0, 1, 10]))
plt.clabel(cs)
```

[12]: <a list of 8 text.Text objects>



1.1.1 Extrema (Wiederholung)

[13]:
```
ext = solve({Eq(f.diff(y), 0), Eq(f.diff(x), 0)}) # Kandidaten für Extrema␣
↪(grad(f) = 0)
```

[14]:
```
ext # Liste von Lösungen jeweils als Dictionary
```

[14]: $[\{x : -1.64038820320221,\ y : -0.768051397498532\},\ \{x : -1.64038820320221,\ y : 0.768051397498532\},\ \{x : -1.5$

[15]:
```
extl = nonlinsolve((Eq(f.diff(y), 0), Eq(f.diff(x), 0)), (x, y))
```

[16]:
```
extl # Lösungsmenge
```

[16]:
$$\{(-1.64038820320221,\ -0.768051397498532), (-1.64038820320221,\ 0.768051397498532), (-1.5,\ 0), (-0.6096117$$

[17]:
```
ext = [{x:ex[0], y:ex[1]} for ex in extl] # Liste von Lösungen jeweils als␣
↪Dictionary
```

[18]:
```
H = hessian(f, (x, y))
H
```

[18]:
$$\begin{bmatrix} 6.0x^2 + 6x - 2y^2 & -4xy - 4y \\ -4xy - 4y & -2x^2 - 4x - 6.0y^2 \end{bmatrix}$$

Hurwitz Kriterium

Eine Matrix A ist genau dann positiv definit, falls alle Hauptminoren von A positiv sind.

Eine Matrix A ist genau dann negativ definit, falls $-A$ positiv definit ist.

[19]:
```
H0 = H.subs(ext[0])
H0, (H0.det() > 0) & (H0[0, 0] > 0)
```

[19]:
$$\left(\begin{bmatrix} 5.12310562561766 & -1.96740421764412 \\ -1.96740421764412 & -2.35961179679779 \end{bmatrix}, \text{False}\right)$$

[20]:
```
for lsg in ext:
    H0 = H.subs(lsg)
    print(lsg, H0.det(), H0[0, 0])
```

```
{x: -1.64038820320221, y: -0.768051397498532} -15.9592198260524 5.12310562561766
{x: -1.64038820320221, y: 0.768051397498532} -15.9592198260524 5.12310562561766
{x: -1.50000000000000, y: 0} 6.75000000000000 4.50000000000000
{x: -0.609611796797793, y: -0.920650341226544} 8.52171982605243
-3.12310562561766
{x: -0.609611796797793, y: 0.920650341226544} 8.52171982605243 -3.12310562561766
{x: 0, y: 0} 0 0
```

[21]:
```
for ex in ext:
    H0 = H.subs(ex)
    if (H0.det() > 0) & (H0[0, 0] > 0):
        print('Minimum bei x {0} y {1}'.format(ex[x], ex[y]))
    elif ((-H0).det() > 0) & (-H0[0, 0] > 0):
        print('Maximum bei x {0} y {1}'.format(ex[x], ex[y]))
    elif (H0.det() < 0): #Spezialfall für reellwertige Funktione auf R^2
        print('Sattelpunkt bei x {0} y {1}'.format(ex[x], ex[y]))
    else:
        print('Keine Aussage bei x {0} y {1}'.format(ex[x], ex[y]))
```

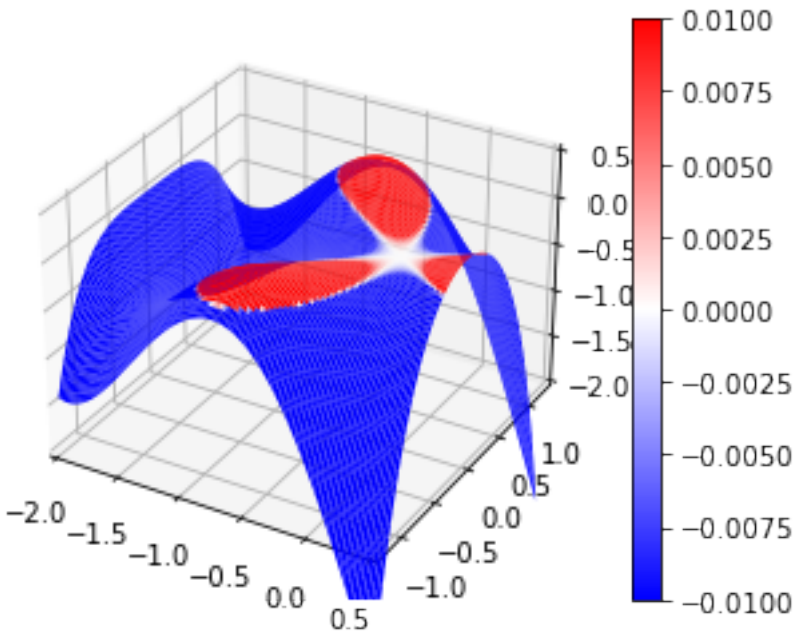
```
Sattelpunkt bei x -1.64038820320221 y -0.768051397498532
Sattelpunkt bei x -1.64038820320221 y 0.768051397498532
```

```
Minimum bei x -1.50000000000000 y 0
Maximum bei x -0.609611796797793 y -0.920650341226544
Maximum bei x -0.609611796797793 y 0.920650341226544
Keine Aussage bei x 0 y 0
```

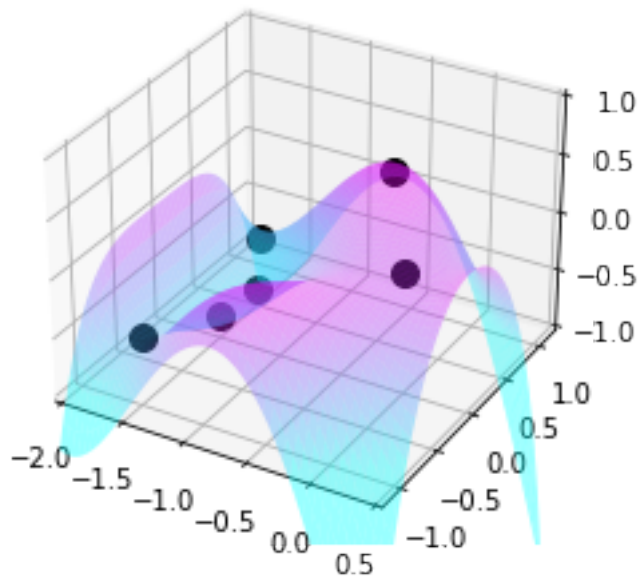
[22]:
```
xv = []
yv = []
fv = []
for ex in ext:
    fw = f.subs(ex)
    print('x', ex[x].n(), 'y', ex[y].n(), 'f', fw.n())
    xv.append(ex[x].n())
    yv.append(ex[y].n())
    fv.append(fw.n())
```

```
x -1.64038820320221 y -0.768051397498532 f -0.619684349426759
x -1.64038820320221 y 0.768051397498532 f -0.619684349426759
x -1.50000000000000 y 0 f -0.843750000000000
x -0.609611796797793 y -0.920650341226544 f 0.201715599426759
x -0.609611796797793 y 0.920650341226544 f 0.201715599426759
x 0 y 0 f 0
```

[23]:
```
xn = np.linspace(-2, .5, 100)
yn = np.linspace(-1.3, 1.3, 100)
X, Y = np.meshgrid(xn, yn)
F = fn(X, Y)

from matplotlib.colors import Normalize
norm = Normalize(fv[5]-.01, fv[5]+.01) # legt die Farbskalierung um den Wert␣
 ↪f(0,0)

fig = plt.figure(3)
fig.clf()

ax = fig.add_subplot(111, projection='3d')
pl = ax.plot_surface(X, Y, F, linewidth=0, cmap=plt.cm.bwr, norm=norm,␣
 ↪cstride=1, rstride=1)

ax.set_xlim(-2, .5)
ax.set_ylim(-1.3, 1.3)
ax.set_zlim(-2, .5)
fig.colorbar(pl);
```

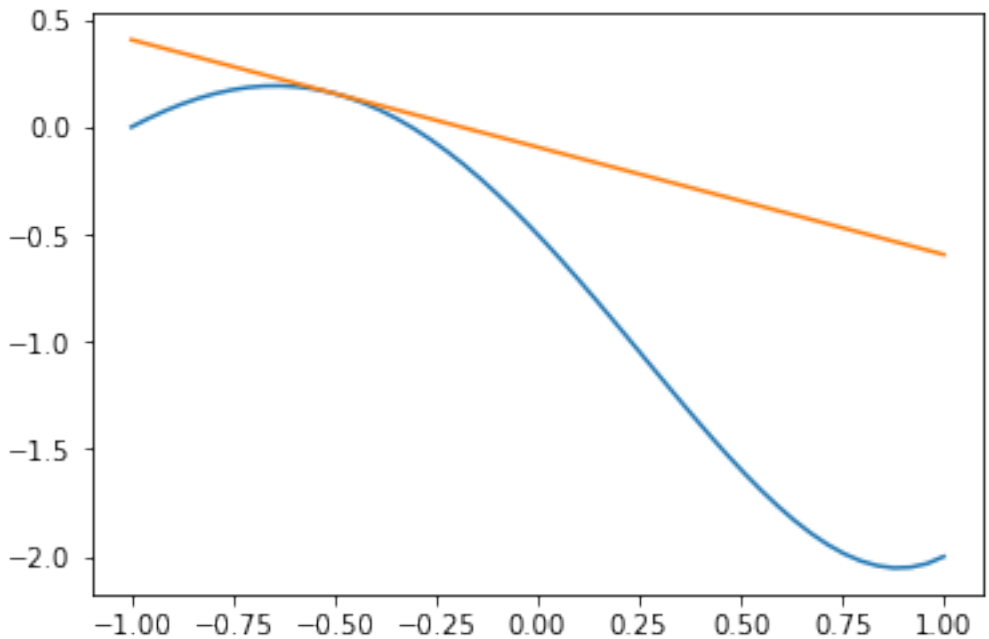


[24]:
```
norm = Normalize(-1, 0.25)

fig = plt.figure(4)
fig.clf()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, F, linewidth=0, cmap=plt.cm.cool, norm=norm, alpha=0.4)
ax.plot3D(xv, yv, fv, 'ko', markersize=10)
ax.set_ylim(-1.2, 1.2)
ax.set_xlim(-2, 0.5)
ax.set_zlim(-1, 1);
```



1.1.2 Tangenten

[25]:
```
xt = np.linspace(-1, 1)
fig2 = plt.figure(6)
ax2 = fig2.gca()
x0, y0 = -.5, -1
ax2.plot(xt, fn(xt, y0))
ax2.plot(xt, fn(x0, y0)+f.diff(x).subs({x:x0, y:y0})*(xt-x0));
```



[26]:
```
tangent = lambda xt: fn(x0, y0) + f.diff(x).subs({x:x0, y:y0})*(xt-x0)
ax.plot(xt, 0*xt-1, tangent(xt));
```

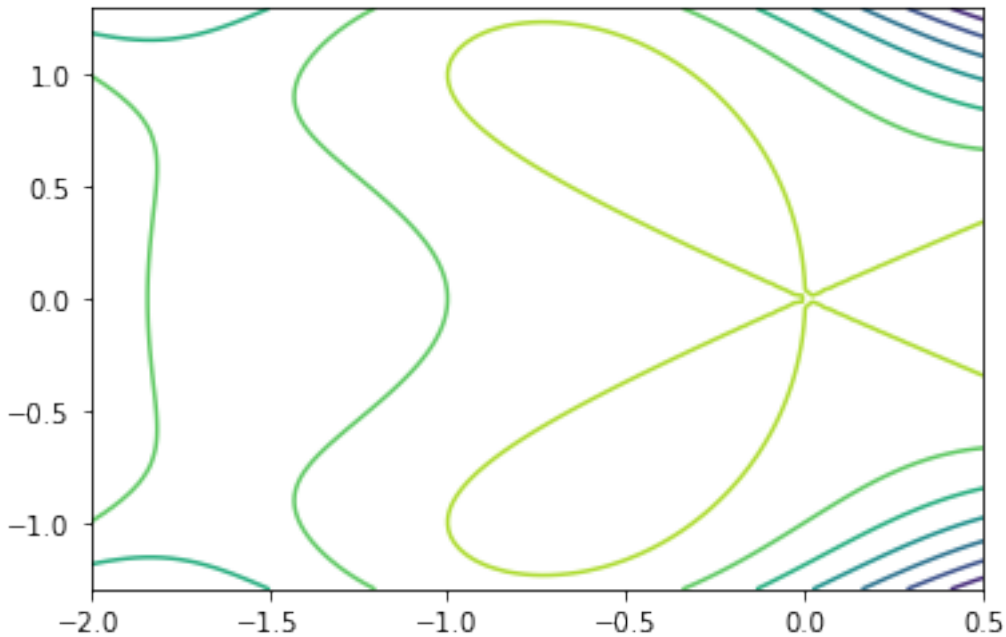
[27]:
```
ax.plot(xt, 0*xt+y0, fn(xt, y0));
```

1.1.3 Gradient als Vektorfeld (quiver) Höhenlinien (contour)

[28]:
```
xn = np.linspace(-1.9, 0.4, 12)
yn = np.linspace(-1.1, 1.1, 11)
XG, YG = np.meshgrid(xn, yn)
grad = Matrix([f]).jacobian([x, y])
gradn = lambdify((x, y), grad)
GR = gradn(XG, YG)[0]
```

[29]:
```
fig = plt.figure(70)
ax = fig.gca()
ax.contour(X, Y, F)
```

[29]: <matplotlib.contour.QuadContourSet at 0x7f54c9015fd0>

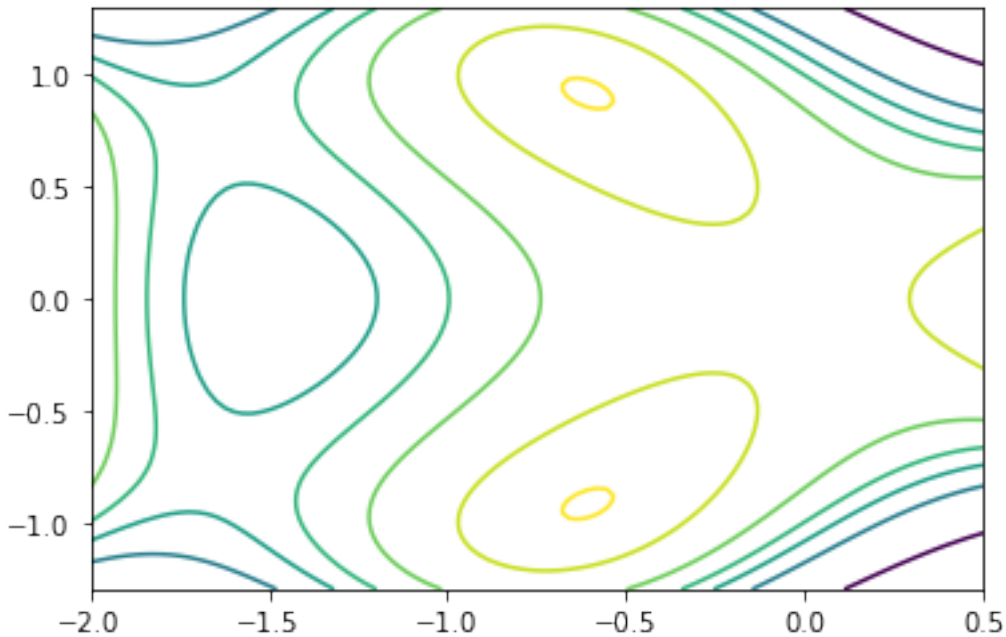


[30]:
```
FG = fn(XG.flatten(), YG.flatten())
dig = np.digitize(FG, np.linspace(-1.5, .3, 16)) # verteile die Funktionswerte␣
↪in 16 'bins'
u, ind = np.unique(dig, return_index=True)
min(FG), max(FG), FG[ind[::2]]
```

[30]:
```
(-1.8168500000000005,
 0.19508708852674006,
 array([-1.81685   , -0.96647168, -0.68845596, -0.49566168, -0.25495528,
        0.02858919,  0.19508709]))
```

[31]:
```
fig = plt.figure(7)
ax = fig.gca()
ax.contour(X, Y, F, FG[ind[::2]])
```

[31]:
```
<matplotlib.contour.QuadContourSet at 0x7f54c0019400>
```

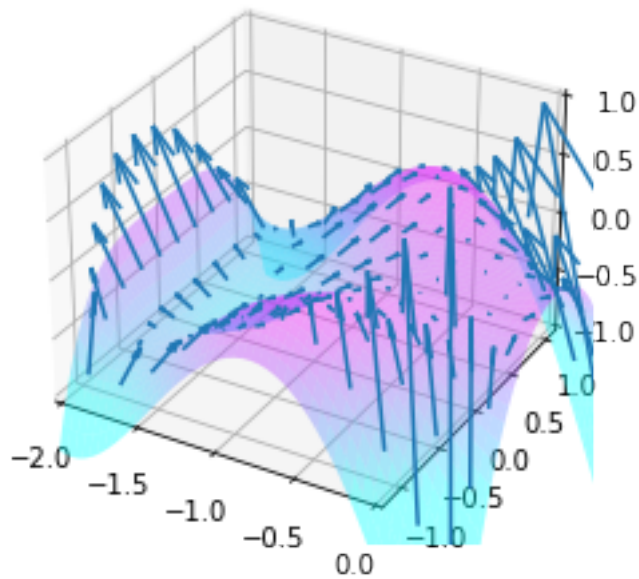


[32]:
```
ax.quiver(XG, YG, GR[0], GR[1], angles='xy', scale_units='xy', scale=5,␣
↪color='red')
ax.set_aspect('equal')
```

[33]:
```
GR = gradn(XG, YG)[0]
nG = np.sqrt(GR[0, :, :]**2 + GR[1, :, :]**2)
```

[34]:
```
fig = plt.figure(9)
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, F, linewidth=0, cmap=plt.cm.cool, norm=norm, alpha=.4)
ax.quiver(XG, YG, fn(XG, YG), GR[0, :, :]/10, GR[1, :, :]/10, (nG**2)/10)

ax.set_ylim(-1.2, 1.1)
ax.set_xlim(-2, 0)
ax.set_zlim(-1, 1)
```

[34]: $(-1.0,\ 1.0)$



1.2 Differentialgleichungen

vgl. Analysis II

1.2.1 erstes Beispiel

[35]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
init_printing()
%matplotlib notebook
```

Gesucht ist eine differenzierbare Funktion $y : \mathbb{R} \to \mathbb{R}$ mit

$$y(t)' = y(t).$$

[36]:
```
y = Function('y')
t, tau = symbols('t tau', real = True)
dgl = Eq(y(t).diff(t), y(t))
dgl
```

[36]: $\frac{d}{dt}y(t) = y(t)$

[37]:
```
sol = dsolve(dgl, y(t))
sol
```

[37]:

$$y(t) = C_1 e^t$$

[38]:
```
sol.subs(S('C1'), 1)
```

[38]:

$$y(t) = e^t$$

[39]:
```
c1_ = sol.atoms(Symbol).difference(dgl.atoms(Symbol))
c1_
```

[39]:

$$\{C_1\}$$

[40]:
```
c1 = list(c1_)[0]
c1
```

[40]:

$$C_1$$

[41]:
```
c1 = S('C1')
```

Wir ergänzen die Differentialgleichunng mit einer Anfangsbedingung zur einem Anfangswertproblem. Gesucht ist $y : \mathbb{R} \to \mathbb{R}$ mit

$$\begin{aligned} \frac{d}{dt} y(t) &= y(t) \\ y(t_0) &= y_0 \end{aligned}$$

[42]:
```
t0, y0 = symbols('t_0 y_0')
sol.subs({t:t0 , y(t0): y0})
```

[42]:

$$y(t_0) = C_1 e^{t_0}$$

[43]:
```
solve(sol.subs({t:t0 , y(t0): y0}), c1)
```

[43]:

$$\left[y(t_0)e^{-t_0}\right]$$

[44]:
```
checkodesol(dgl, sol)
```

[44]: (True, 0)

[45]:
```
aw = {y(t0): y0} # Anfangswerte (initial conditions)
```

[46]:
```
dsolve(dgl, y(t), ics=aw)
```

[46]:

$$y(t) = y_0 e^t e^{-t_0}$$

1.2.2 Inhomogene lineare DGL

$$u'(t) = u(t) + \sin(t)$$

[47]:
```
u = Function('u')
t = symbols('t', real = True)
dgl = Eq(u(t).diff(t)-u(t)-sin(t), 0)
sol = dsolve(dgl,u(t))
sol
```

[47]:

$$u(t) = C_1 e^t - \frac{\sin(t)}{2} - \frac{\cos(t)}{2}$$

[57]:
```
c1_s = S('C1')
c1 = solve(sol, c1_s).pop().subs(t, 0) # hier ist  t_0 = 0
c1
```

[57]:

$$\left(u(t) + \frac{\sqrt{2}\sin\left(t + \frac{\pi}{4}\right)}{2}\right) e^{-t}$$

[58]:
```
w = sol.subs(c1_s, c1)
w = w.rhs
```

1.2.3 Variation der Konstanten Formel

Die Lösung von

$$\frac{d}{dt}u(t) = au(t) + g(t, u(t)), \quad u(0) = u_0$$

ist

$$u(t) = \mathrm{e}^{at}u_0 + \int_0^t \mathrm{e}^{a(t-\tau)} g(\tau, u(\tau))d\tau$$

Für das Beispiel oben ist $a = 1$ und $g(t, u) = \sin(t)$. Das ergibt:

[59]:
```
v = u(0)*exp(t) + integrate(exp(t-tau)*(sin(tau)), (tau, 0, t))
v
```

[59]:

$$u(0)e^t + \frac{e^t}{2} - \frac{\sin(t)}{2} - \frac{\cos(t)}{2}$$

[60]:
```
simplify(v-w)
```

[60]:

$$u(0)e^t - u(t) + \frac{e^t}{2} - \frac{\sqrt{2}\sin\left(t + \frac{\pi}{4}\right)}{2}$$

1.2.4 Lösung mit einem Reihenansatz

$$y'(t) = y(t), \quad y(0) = 1$$

mit einem Reihenansatz

[61]:
```
y0, t, C = symbols('y0 t C')
a = symbols('a:8')
y = Function('y')
```

[62]:
```
dgl = Eq(y(t).diff(t), y(t))
dgl
```

[62]:
$$\frac{d}{dt}y(t) = y(t)$$

[63]:
```
ys = sum([a[i]*t**i for i in range(8)])
ys = ys.subs(a[0],1)  # y(0) = 1
ys
```

[63]:
$$a_1t + a_2t^2 + a_3t^3 + a_4t^4 + a_5t^5 + a_6t^6 + a_7t^7 + 1$$

[64]:
```
gl = dgl.subs(y(t), ys).doit()
gl
```

[64]:
$$a_1 + 2a_2t + 3a_3t^2 + 4a_4t^3 + 5a_5t^4 + 6a_6t^5 + 7a_7t^6 = a_1t + a_2t^2 + a_3t^3 + a_4t^4 + a_5t^5 + a_6t^6 + a_7t^7 + 1$$

[65]:
```
gl.coeff(t) # das klappt so nicht
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_143359/37395411.py in <module>
----> 1 gl.coeff(t) # das klappt so nicht

AttributeError: 'Equality' object has no attribute 'coeff'
```

[66]:
```
gl1 = (gl.lhs - gl.rhs).expand()
gl1
```

[66]:
$$-a_1t + a_1 - a_2t^2 + 2a_2t - a_3t^3 + 3a_3t^2 - a_4t^4 + 4a_4t^3 - a_5t^5 + 5a_5t^4 - a_6t^6 + 6a_6t^5 - a_7t^7 + 7a_7t^6 - 1$$

[67]:
```
gl1.coeff(t)
```

[67]:
$$-a_1 + 2a_2$$

[68]:
```
gls = gl1.as_poly(t).all_coeffs()
gls[1:]
```

[68]:
$$[-a_6 + 7a_7,\ -a_5 + 6a_6,\ -a_4 + 5a_5,\ -a_3 + 4a_4,\ -a_2 + 3a_3,\ -a_1 + 2a_2,\ a_1 - 1]$$

[69]:
```
ac = solve(gls[1:])
ac
```

[69]:
$$\left\{a_1 : 1,\ a_2 : \frac{1}{2},\ a_3 : \frac{1}{6},\ a_4 : \frac{1}{24},\ a_5 : \frac{1}{120},\ a_6 : \frac{1}{720},\ a_7 : \frac{1}{5040}\right\}$$

[70]:
```
ac[a[0]] = 1  # wir raten das Bildungsgesetz der a's
acc = [ac[j] for j in a]
acc
```

[70]: $\left[1,\ 1,\ \frac{1}{2},\ \frac{1}{6},\ \frac{1}{24},\ \frac{1}{120},\ \frac{1}{720},\ \frac{1}{5040}\right]$

[71]:
```
[ acc[j]/acc[j+1] for j in range(7)]
```

[71]: $[1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7]$

[72]:
```
[ acc[j] - 1/factorial(j) for j in range(8)]
```

[72]: $[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]$

[73]:
```
n = symbols('n')
yr = Sum(t**n/factorial(n) ,(n,0,oo))
yr
```

[73]: $\sum_{n=0}^{\infty} \frac{t^n}{n!}$

[74]:
```
yr.doit()
```

[74]: e^t

1.2.5 Logistische Gleichung

einfaches Modell zur Beschreibung von Wachstum in einem Habitat mit beschränkten Resourcen

$$y'(t) = (1 - y(t))y(t)$$

[75]:
```
y = Function('y')
t = symbols('t', real = True)
dgl = Eq(y(t).diff(t), (1-y(t))*y(t))
dgl
```

[75]: $\frac{d}{dt}y(t) = (1 - y(t))\, y(t)$

[76]:
```
sol = dsolve(dgl, y(t))
sol
```

[76]: $y(t) = \frac{1}{C_1 e^{-t} + 1}$

[77]:
```
c1_s = sol.atoms(Symbol).difference(dgl.atoms(Symbol)).pop() # das geht␣
↪einfacher :)
c1_s
```

[77]: C_1

[78]:
```
c1_s = S('C1')
```

[79]:
```
C1 = solve(sol, c1_s)
C1
```

[79]:

$$\left[-e^{t} + \frac{e^{t}}{y(t)}\right]$$

[80]:
```
C1 = solve(sol, c1_s).pop().subs(t, 0)
C1
```

[80]:

$$-1 + \frac{1}{y(0)}$$

[81]:
```
sol.subs(c1_s, C1)
```

[81]:

$$y(t) = \frac{1}{\left(-1 + \frac{1}{y(0)}\right) e^{-t} + 1}$$

[ ]:

[ ]: