lektion13_i

January 22, 2026

Inhalt

1 Lektion 13

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
init_printing()
%matplotlib inline
plt.rcParams['figure.figsize'] = (5, 3)
```

2 Differentialgleichungen (Teil 2)

$$y'(t) = f(y,t), \quad y(t_0) = y_0$$

2.1 Logistische Gleichung (Wiederholung)

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

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

[2]:
```
y = Function('y')
t, t0, tau, y0 = symbols('t t_0 tau y0', real=True)
yy = symbols('yy')
dgl = Eq(y(t).diff(t), (1 - y(t)) * y(t))
dgl
```

[2]:
$$\frac{d}{dt}y(t) = (1 - y(t))\, y(t)$$

[3]:
```
sol = dsolve(dgl, y(t))
sol
```

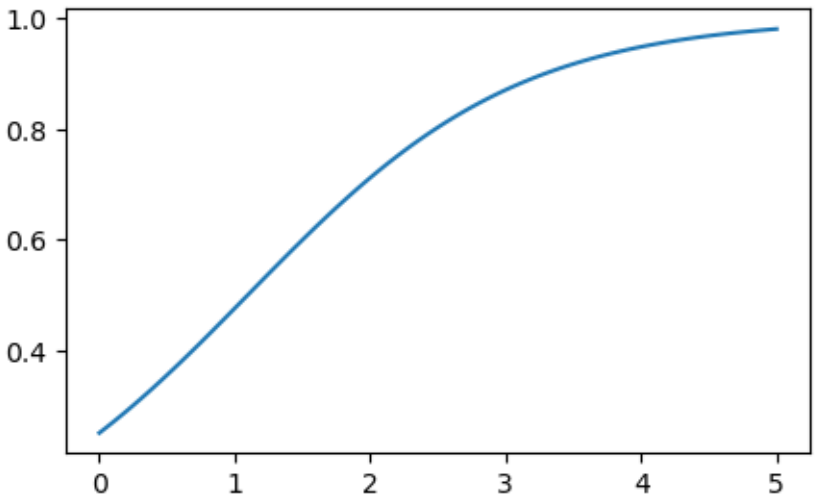
[3]:
$$y(t) = \frac{1}{C_1 e^{-t} + 1}$$

[4]:
```
aw = {y(t0): y0}
sol_aw = dsolve(dgl, y(t), ics=aw)
sol_aw
```

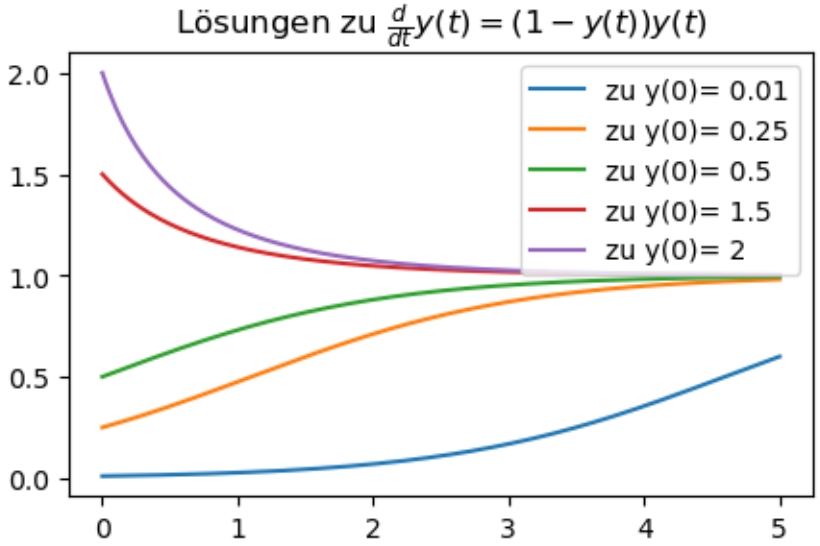
[4]:
$$y(t) = \frac{1}{1 + \frac{(1-y_0)e^{-t}e^{t_0}}{y_0}}$$

[5]:
```
fig = plt.figure(1)
fig.clf()
ax = fig.gca()
tn = np.linspace(0, 5, 101)
y_ = sol_aw.rhs.subs(t0, 0).subs(y0, .25) # Anfangsbedingung y(0)= 0.1
yn = lambdify(t, y_)
ax.plot(tn, yn(tn))
```

[5]:
```
[<matplotlib.lines.Line2D at 0x7f65da162600>]
```



[6]:
```
fig = plt.figure(2)
ax = fig.gca()
tn = np.linspace(0, 5, 201)
for y_0 in [.01, .25, .5, 1.5, 2]: # mehrere Anfangsbedingungen
    y_ = sol_aw.rhs.subs(t0, 0).subs(y0, y_0)
    yn = lambdify(t, y_)
    ax.plot(tn, yn(tn), label=f"zu y(0)= {y_0}")
ax.legend()
ax.set_title(f"Lösungen zu ${latex(dgl)}$");
```



[7]:
```
f = dgl.rhs.subs(y(t), yy)
f
```

[7]:
$$yy\,(1 - yy)$$

[8]:
```
fn = lambdify(yy, f)
tg = np.linspace(0, 5, 10)
yg = np.linspace(0.1, 1.9, 20)
TT, YY = np.meshgrid(tg, yg)
ax.quiver(TT, YY, np.ones_like(YY), fn(YY), angles='xy');
# angles='xy' ist wichtig für korrekte Richtungen der Vektoren!
```

2.1.1 Anpassung an Anfangswert

[9]:
```
sol
```

[9]:
$$y(t) = \frac{1}{C_1 e^{-t} + 1}$$

[10]:
```
c1_s = S('C1')
c1_s
```

[10]:
$$C_1$$

[11]:
```
C1 = solve(sol, c1_s)[0].subs(t, t0).replace(y(t0),y0)
C1
```

[11]:
$$-e^{t_0} + \frac{e^{t_0}}{y_0}$$

[12]:
```
ys = sol.subs(c1_s, C1)
ys
```

[12]:
$$y(t) = \frac{1}{\left(-e^{t_0} + \frac{e^{t_0}}{y_0}\right) e^{-t} + 1}$$

2.2 Systeme von DGLen

2.2.1 Lineare Systeme - Matrixexponentialfunktion

Für eine Matrix $n \times n$ A suchen wir eine Lösung $u : [t_0, T] \to \mathbb{R}^n$ von

$$u'(t) = Au(t), \quad u(0) = u_0$$

[13]:
```
t = symbols('t', real=True)
```

[14]:
```
A = Matrix(3, 3, [2, 1, 0,\
                  0, 2, 1,\
                  0, 0, 2])
A
```

[14]:
$$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

[15]:
```
exp(t*A)
```

[15]:
$$\begin{bmatrix} e^{2t} & te^{2t} & \frac{t^2 e^{2t}}{2} \\ 0 & e^{2t} & te^{2t} \\ 0 & 0 & e^{2t} \end{bmatrix}$$

[16]:
```
u0 = Matrix(3, 1, [1, 2, 3]) # Anfangswert
u0
```

[16]:
$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

[17]:
```
u = exp(t*A)*u0
u
```

[17]:
$$\begin{bmatrix} \frac{3t^2e^{2t}}{2} + 2te^{2t} + e^{2t} \\ 3te^{2t} + 2e^{2t} \\ 3e^{2t} \end{bmatrix}$$

[18]:
```
u.diff(t) - A*u
```

[18]:
$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

2.2.2 Zweite Ordnung DGL

Harmonischer Oszillator

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

---

[19]:
```
dgl = Eq(y(t).diff(t, 2), -y(t))
dgl
```

[19]:
$$\frac{d^2}{dt^2}y(t) = -y(t)$$

[20]:
```
dsolve(dgl, y(t))
```

[20]:
$$y(t) = C_1 \sin(t) + C_2 \cos(t)$$

äquivalentes System 1. Ordnung

$$\begin{aligned} u_1'(t) &= u_2(t) \\ u_2'(t) &= -u_1(t) \end{aligned}$$

in Matrixschreibweise

$$u'(t) = Au(t) := f(u(t))$$

[21]:
```
A = Matrix(2, 2, [0, 1, -1, 0])
A
```

[21]:
$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

[22]:
```
u0 = Matrix(2, 1, [1, 0])
u0
```

[22]:
$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

[23]:
```
MexpA = lambda t: simplify((t*A).exp())
MexpA(t)
```

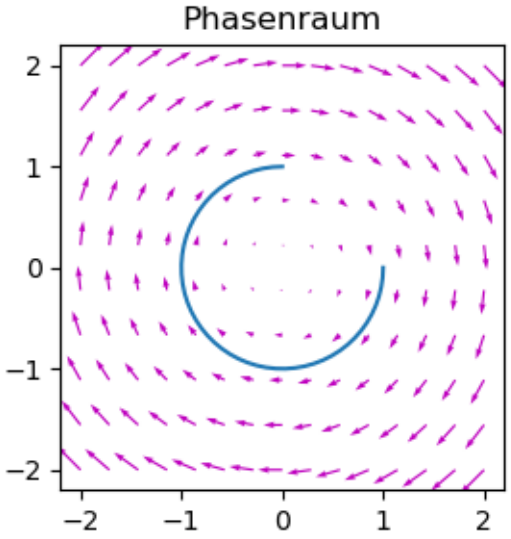
[23]:
$$\begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix}$$

[24]:
```
u = MexpA(t)*u0
un = lambdify(t,u)
```

[25]:
```
def f(u):
    return u[1], -u[0]
```

[26]:
```
u1 = np.linspace(-2.0, 2.0, 15)
u2 = np.linspace(-2.0, 2.0, 10)
tn = np.linspace(0, 3 / 2 * np.pi)
U1, U2 = np.meshgrid(u1, u2)
U, V = f([U1, U2])
```

[27]:
```
fig = plt.figure(3)
ax = fig.add_subplot(111)
ax.plot(un(tn)[0][0], un(tn)[1][0])
ax.set_title('Phasenraum')
ax.quiver(U1, U2, U, V, angles='xy', color='m')
ax.set_aspect('equal');
```



2.2.3 Pendelgleichung (physikalisches Pendel)

Nichtlineare Differentialgleichung 2. Ordnung:

$$\alpha''(t) = -\sin(\alpha(t)) \quad (m = g = l = 1)$$

äquivalent zu System 1. Ordnung

$$y_1(t) := \alpha(t),\ y_2(t) := \frac{d}{dt}\alpha(t)$$

$$\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= -\sin(y_1(t)) \end{aligned}$$

y_1: Winkel

y_2: Winkelgeschwindigkeit

[28]:
```
dgl = Eq(y(t).diff(t, 2), -sin(y(t)))
dgl
```

[28]:
$$\frac{d^2}{dt^2}y(t) = -\sin\left(y(t)\right)$$

[29]:
```
#sol = dsolve(dgl, y(t)) # das kann sympy nicht
#sol
```

2.2.4 Phasenraum und Trajektorien

vgl. Ana II Kurzskript Kap. 20 http://www.math.uni-duesseldorf.de/~internet/ana2_19/vorlesung.pdf https://www.math.uni-duesseldorf.de/~adams/static/analysis2_ws2025/analysis2_skript.pdf

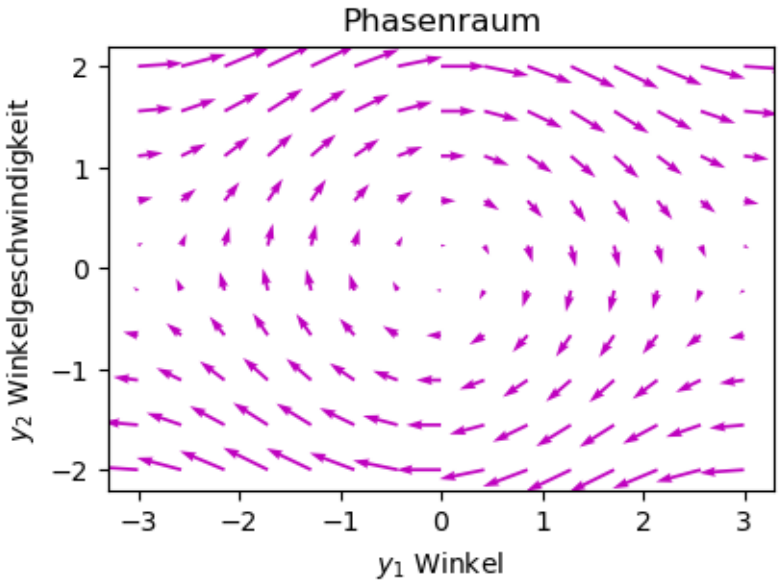
[30]:
```
def f(y, t):
    y1 = y[0]
    y2 = y[1]
    return y2, -np.sin(y1)
```

[31]:
```
y1 = np.linspace(-3.0, 3.0, 15)
y2 = np.linspace(-2.0, 2.0, 10)
Y1, Y2 = np.meshgrid(y1, y2)
t0 = 0
```

[32]:
```
U, V = f([Y1, Y2], t0)
```

[33]:
```
fig = plt.figure(4)
ax = fig.add_subplot(111)
ax.set_title('Phasenraum')
ax.quiver(Y1, Y2, U, V, angles='xy', color='m')
ax.set_aspect('equal')
ax.set_xlabel('$y_1$ Winkel')
```

```
ax.set_ylabel('$y_2$ Winkelgeschwindigkeit');
```

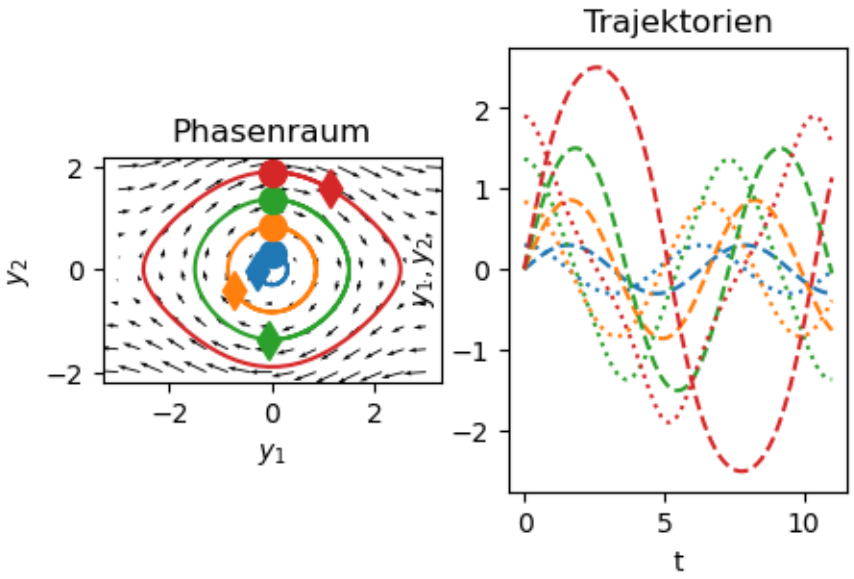


[34]:
```
from scipy.integrate import odeint  # numerische Integration (numerisches␣
 ↪Lösungsverfahren für DGL)
# Das kommt sicher nicht in der Klausur dran!
tn = np.linspace(0, 2.5, 10)
y0 = np.array([0.0, 1]) # Anfangswert
yn = odeint(f, y0, tn)  #berechnet Näherungslösung y für jeden Wert in tn
yn
```

[34]:
```
array([[ 0.        ,  1.        ],
       [ 0.27423279,  0.96190773],
       [ 0.52781962,  0.85312045],
       [ 0.74293751,  0.68772761],
       [ 0.90626302,  0.48309797],
       [ 1.0091793 ,  0.25516877],
       [ 1.04703987,  0.01652653],
       [ 1.01830076, -0.22276961],
       [ 0.92412561, -0.45284143],
       [ 0.76868903, -0.66154768]])
```

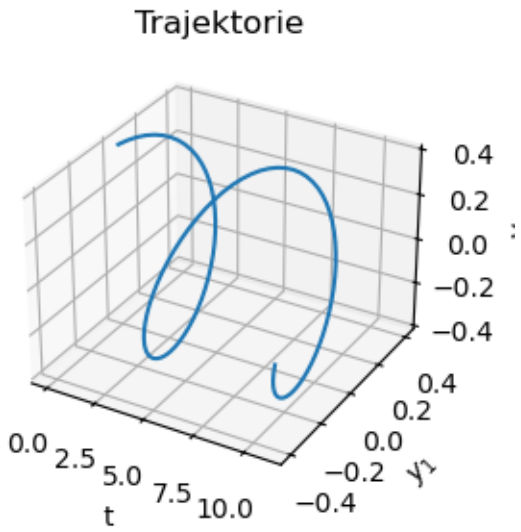
[35]:
```
fig = plt.figure(5)

ax1 = fig.add_subplot(121)
ax1.quiver(Y1, Y2, U, V, angles='xy', color='k')
ax1.set_aspect('equal');
ax1.set_xlabel('$y_1$');
```

```
ax1.set_ylabel('$y_2$');
ax1.set_title('Phasenraum')

ax2 = fig.add_subplot(122)
ax2.set_title('Trajektorien')

tn = np.linspace(0, 11, 100)
for y20 in np.linspace(0.3, 1.9, 4):
    y0 = [0.0, y20]

    yn = odeint(f, y0, tn)

    line, = ax1.plot(yn[:,0], yn[:,1], '-') # Phasenporträt (Lösung im␣
 ↪Phasenraum)
    ax1.plot(yn[0,0], yn[0,1], 'o', color=line.get_color(), markersize=10)  #␣
 ↪Startwert
    ax1.plot(yn[-1,0], yn[-1,1], 'd', color=line.get_color(), markersize=10) #␣
 ↪Wert zum Endzeitpunkt

    ax2.plot(tn, yn[:,0], '--', color=line.get_color()) # y_1
    ax2.plot(tn, yn[:,1], ':', color=line.get_color())  # y_2
    ax2.set_xlabel('t')
    ax2.set_ylabel('$y_1, y_2$')
```



[36]:
```
y0 = [0.0, .4]
yn = odeint(f, y0, tn)
```

```
fig = plt.figure(6)
ax = fig.add_subplot(111, projection='3d')
ax.plot3D(tn, yn[:, 0], yn[:, 1])
ax.set_xlabel('t')
ax.set_ylabel('$y_1$')
ax.set_zlabel('$y_2$')
ax.set_title('Trajektorie');
```



[37]:
```
ax.view_init(0, 0)
ax.set_proj_type('ortho')  # focal_lenght = oo,  FOV = 0 deg
#ax.set_proj_type('persp')  # focal_lenght 1 ,FOV = 90 deg
#ax.set_proj_type('persp', focal_length=0.2)  # FOV = 157.4 deg
plt.show()
```

[38]:
```
ax.view_init(0, 90)
plt.show()
```

[39]:
```
ax.view_init(90,-90)
plt.show()
```

falls der Winkel klein ist, ist

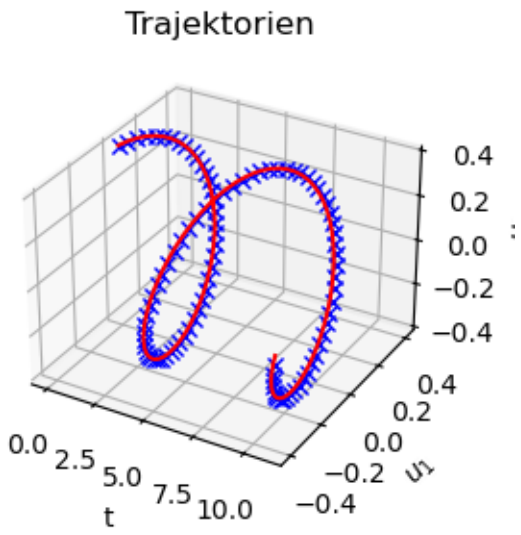
$$\sin(\alpha) \approx \alpha$$

und wir erhalten als Approximation den harmonischen Oszillator

$$u_1'(t) = u_2(t) \tag{1}$$
$$u_2'(t) = -u_1(t) \tag{2}$$

[40]:
```
dgl3 = Eq(y(t).diff(t, 2), -y(t))
awe = {y(0): y0[0], y(t).diff(t).subs(t, 0): y0[1]}
sol3 = dsolve(dgl3, y(t), ics=awe)
```

[41]:
```
fig = plt.figure(7)
ax = fig.add_subplot(111, projection='3d')
ax.plot3D(tn,yn[:,0], yn[:,1],'bx')
ax.plot3D(tn,lambdify(t, sol3.rhs)(tn), lambdify(t, sol3.rhs.diff(t))(tn), 'r')
ax.set_xlabel('t')
ax.set_ylabel('$u_1$')
ax.set_zlabel('$u_2$')
ax.set_title('Trajektorien')
plt.show()
```



2.2.5 Zwei Massen Feder System

System von linearen Differentialgleichungen 2. Ordnung

$$q_0''(t) = -k_0 q_0(t) - k_1(q_0(t) - q_1(t)) \tag{3}$$
$$q_1''(t) = k_1(q_0(t) - q_1(t)) \tag{4}$$

Anfangswerte

$$q_0(0) = 0, q_0'(0) = 1, q_1(0) = 0, q_1'(0) = 0$$

[42]:
```
q0 = Function('q_0')
q1 = Function('q_1')
t, k0, k1 = symbols('t k_0 k_1', real=True)
```

[43]:
```
dgl_orig = [
    Eq(q0(t).diff(t, 2), -k0 * q0(t) - k1 * (q0(t) - q1(t))),
    Eq(q1(t).diff(t, 2), k1 * (q0(t) - q1(t)))
]
dgl_orig
```

[43]:
$$\left[\frac{d^2}{dt^2}q_0(t) = -k_0 q_0(t) - k_1\left(q_0(t) - q_1(t)\right),\ \frac{d^2}{dt^2}q_1(t) = k_1\left(q_0(t) - q_1(t)\right)\right]$$

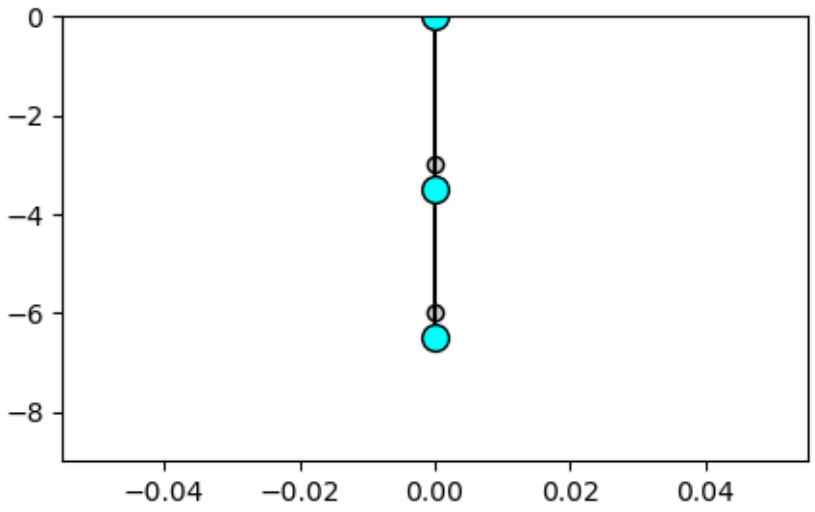
[44]:
```
dgl = [dgl_.subs({k0: 9, k1: 16}) for dgl_ in dgl_orig]
dgl
```

[44]:
$$\left[\frac{d^2}{dt^2}q_0(t) = -25q_0(t) + 16q_1(t),\ \frac{d^2}{dt^2}q_1(t) = 16q_0(t) - 16q_1(t)\right]$$

[45]:
```
awe = {
    q0(0): Rational(1, 2),
    q0(t).diff(t).subs(t, 0): 0,
    q1(0): Rational(1, 2),
    q1(t).diff(t).subs(t, 0): 0
}
sol = dsolve(dgl, [q0(t), q1(t)], ics=awe)
sol
```

[45]:
$$\left[q_0(t) = \frac{\sqrt{41-\sqrt{1105}}\sqrt{\sqrt{1105}+41}\left(23\sqrt{1105}+1105\right)\cos\left(\frac{\sqrt{2}t\sqrt{41-\sqrt{1105}}}{2}\right)}{106080} + \frac{\left(1105-23\sqrt{1105}\right)\cos\left(\frac{\sqrt{2}t\sqrt{\sqrt{1105}+}}{2}\right.}{4420}\right.$$

[46]:
```
fig, ax = plt.subplots()
q0n = lambdify(t, sol[0].rhs)
q1n = lambdify(t, sol[1].rhs)
tn = np.linspace(0, 50, 2000)
ax.set_ylim((-9, 0))
L1, L2 = 3, 6
r, = ax.plot([0, 0, 0], [0, -L1, -L2],'ko')
r.set_markerfacecolor('lightgray')
l, = ax.plot([0, 0, 0], [0, -(L1 + q0n(0)), -(L2 + q1n(0))], 'ok-')
l.set_markersize(10)
l.set_markerfacecolor('cyan')
plt.pause(1)
for t_ in tn:
    l.set_ydata([0, -(L1 + q0n(t_)), -(L2 + q1n(t_))])
    plt.pause(0.001)
```



2.3 Weitere Beispiele

[47]:
```
y = Function('y')
t, t0 = symbols('t t_0', real = True)
yy = symbols('yy')
```

2.3.1 DGL mit endlichen maximalen Definitionsbereich

[48]:
```
dgl1 = Eq(y(t).diff(t), exp(y(t))*sin(t))
dgl1
```

[48]:
$$\frac{d}{dt}y(t) = e^{y(t)}\sin(t)$$

[49]:
```
sol1 = dsolve(dgl1, ics={y(0):-S(1)/2})
sol1
```

[49]:
$$y(t) = \log\left(-\frac{1}{-\cos(t) - e^{\frac{1}{2}} + 1}\right)$$

Diese Lösung ist nur auf dem Intervall $[-\pi + \arccos(\exp(1/2) - 1), \pi - \arccos(\exp(1/2) - 1)]$ definiert

[50]:
```
lsg = dsolve(dgl1)
lsg
```

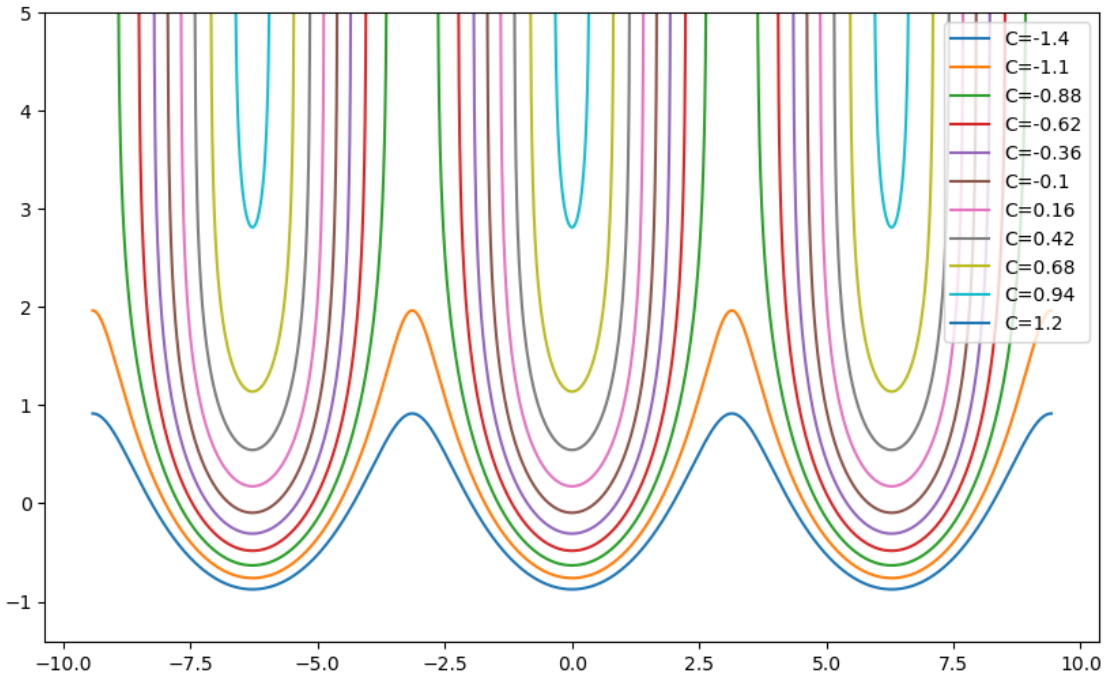
[50]:
$$y(t) = \log\left(-\frac{1}{C_1 - \cos(t)}\right)$$

[51]:
```
tn = np.linspace(-3*np.pi, 3*np.pi, 3000)
yn = lambdify((t, S('C1')), lsg.rhs)
```

[52]:
```
fig = plt.figure()
ax = fig.add_subplot()
for C in np.linspace(-1.4, 1.2, 11):
    ax.plot(tn, yn(tn, C), label=f"C={C:1.2}")
ax.axis(ymax=5)
plt.legend()
```

```
<lambdifygenerated-13>:2: RuntimeWarning: invalid value encountered in log
  return log(-1/(C1 - cos(t)))
```

[52]:
```
<matplotlib.legend.Legend at 0x7f65ce062e10>
```



[53]:
```
tq = np.linspace(tn[0], tn[-1], 20)
yq = np.linspace(-1, 5, 20)
X, Y = np.meshgrid(tq, yq)
r_n = lambdify((t, y(t)), dgl1.rhs)
V = r_n(X, Y)
```

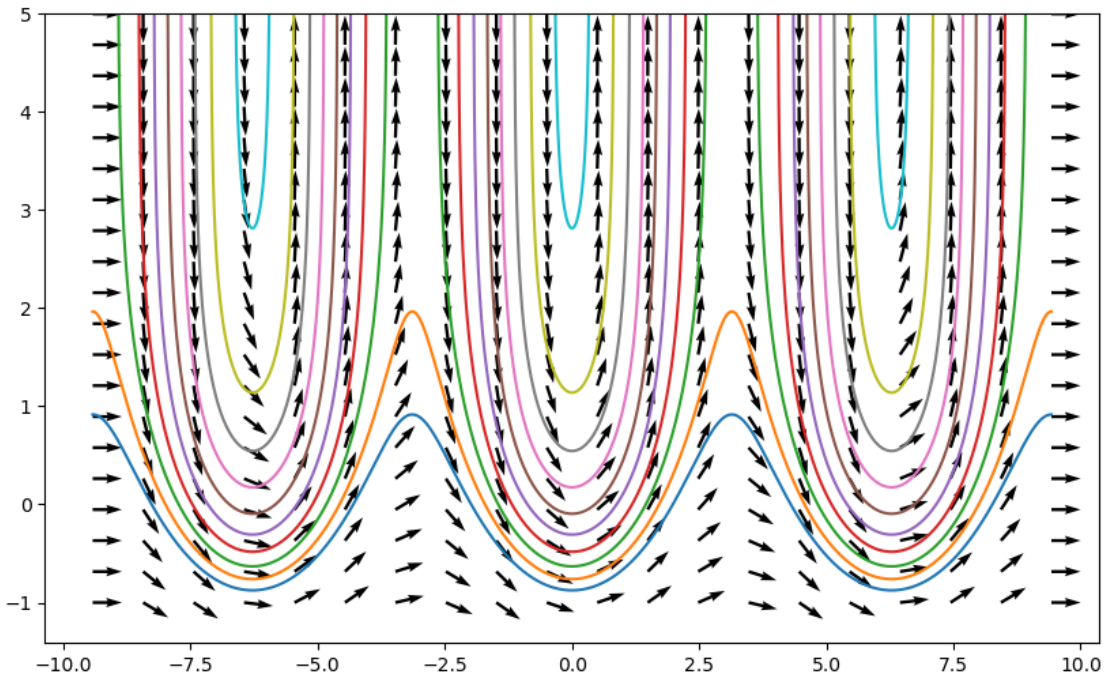
[54]:
```
U = np.ones_like(X)
ax.quiver(X, Y, U, V, angles='xy')
```

[54]:
```
<matplotlib.quiver.Quiver at 0x7f65cdee7ef0>
```

Hier ist es besser, die Pfeile zu normieren.

[55]:
```
nv = np.sqrt(1+V**2)
V /= nv
U /= nv
```

[56]:
```
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111)
for C in np.linspace(-1.4, 1.2, 11):
    ax.plot(tn, yn(tn, C))
ax.axis(ymax=5)
ax.quiver(X, Y, U, V, angles='xy');
```



maximaler Definitionsbereich Für welche y_0 ist die Lösung mit der Anfangsbedingung $y(0) = y_0$ auf ganz $\mathbb{R}$ definiert?

[57]:
```
y0 = S('y0')
ics = {y(0): y0}
```

[58]:
```
lsg = dsolve(dgl1, ics=ics)
lsg
```

[58]:
$$y(t) = \log\left(-\frac{1}{-\cos(t) + 1 - e^{-y_0}}\right)$$

Dazu muss der Nenner immer negativ sein.

Wir isolieren den Nenner.

[59]:
```
lsg.rhs.args[0].args[1].args[0]
```

[59]:
$$-\cos(t) + 1 - e^{-y_0}$$

Wird am größten für $t = \pi$

[60]:
```
b = lsg.rhs.args[0].args[1].args[0].subs(t, pi)
b
```

[60]:
$$2 - e^{-y_0}$$

[61]:
```
I1 = solveset(b < 0, domain=Reals)
I1
```

[61]:
$$(-\infty, -\log(2))$$

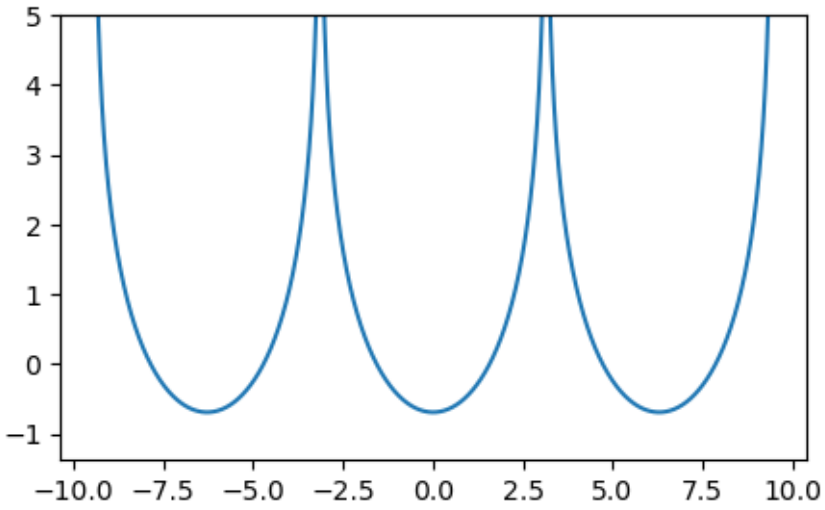
[62]:
```
y_ = lsg.rhs.subs(y0, I1.end)
y_
```

[62]:
$$\log\left(-\frac{1}{-\cos(t) - 1}\right)$$

[63]:
```
y_n = lambdify(t, y_)
plt.plot(tn, y_n(tn))
plt.axis(ymax=5)
```

```
<lambdifygenerated-15>:2: RuntimeWarning: divide by zero encountered in divide
  return log(-1/(-cos(t) - 1))
<lambdifygenerated-15>:2: RuntimeWarning: invalid value encountered in log
  return log(-1/(-cos(t) - 1))
```

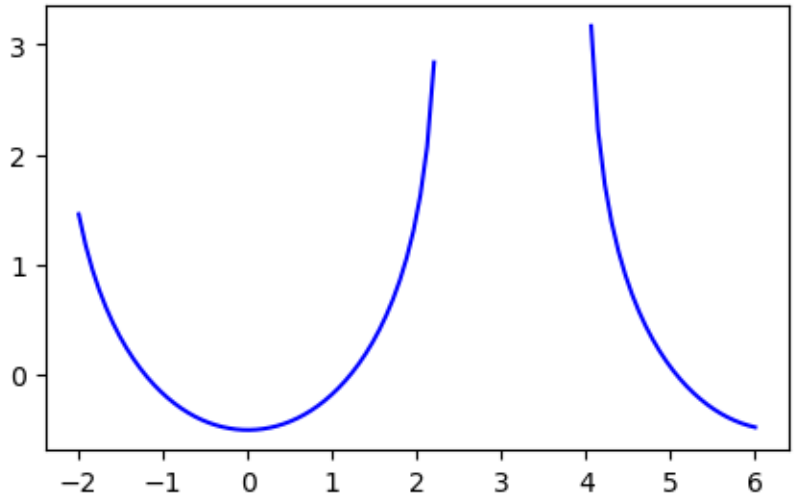
[63]:
$$(-10.3603419484056,\ 10.3603419484056,\ -1.379277340083,\ 5.0)$$



[64]:
```
tn = np.linspace(-2, 6, 100)
plt.figure(8)
plt.plot(tn, lambdify(t, sol1.rhs)(tn), 'b')
(pi - acos(exp(1 / 2) - 1)).n()
```

```
<lambdifygenerated-16>:2: RuntimeWarning: invalid value encountered in log
  return log(-1/(-cos(t) - exp(1/2) + 1))
```

[64]:
2.27669928768065



2.3.2 AWP mit zwei Lösungen

[65]:
```
dgl2 = Eq((t**2 + 2 * t) * y(t) * exp(y(t)**2) * y(t).diff(t), 1)
dgl2
```

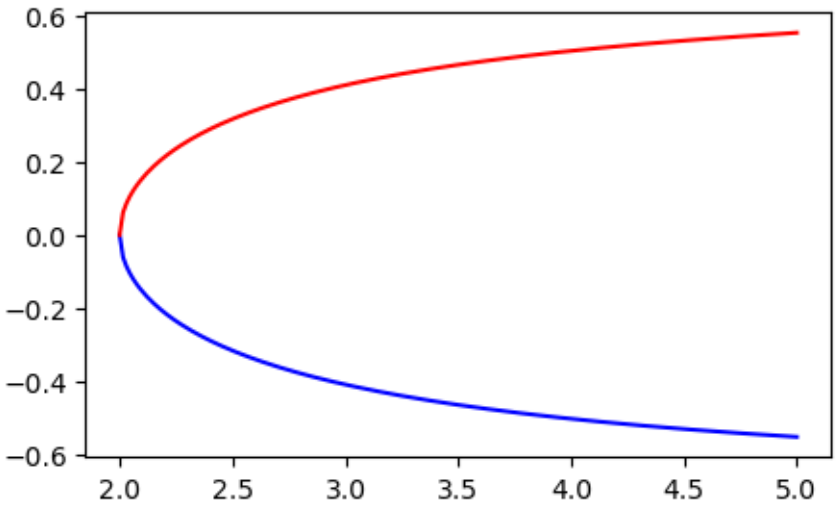
[65]:
$$(t^2 + 2t)\, y(t) e^{y^2(t)} \frac{d}{dt} y(t) = 1$$

[66]:
```
sol2 = dsolve(dgl2, ics={y(2): 0})
sol2
```

[66]:
$$\left[y(t) = -\sqrt{\log\left(\log\left(t\right) - \log\left(t+2\right) + \log\left(2\right) + 1\right)},\ y(t) = \sqrt{\log\left(\log\left(t\right) - \log\left(t+2\right) + \log\left(2\right) + 1\right)}\right]$$

Hier gibt es zwei Lösungen zum Anfangswert y(2)=0

[67]:
```
tn = np.linspace(2, 5, 200)
plt.figure()
plt.plot(tn,
         lambdify(t, sol2[0].rhs)(tn), 'b', tn,
         lambdify(t, sol2[1].rhs)(tn), 'r');
```



2.3.3 AWP mit unendlich vielen Lösungen

[68]:
```
dgl3 = Eq( y(t).diff(t)*sin(t), 2*y(t)*cos(t))
dgl3
```

[68]:
$$\sin(t)\frac{d}{dt}y(t) = 2y(t)\cos(t)$$

[69]:
```
sol3 = dsolve(dgl3)
sol3
```

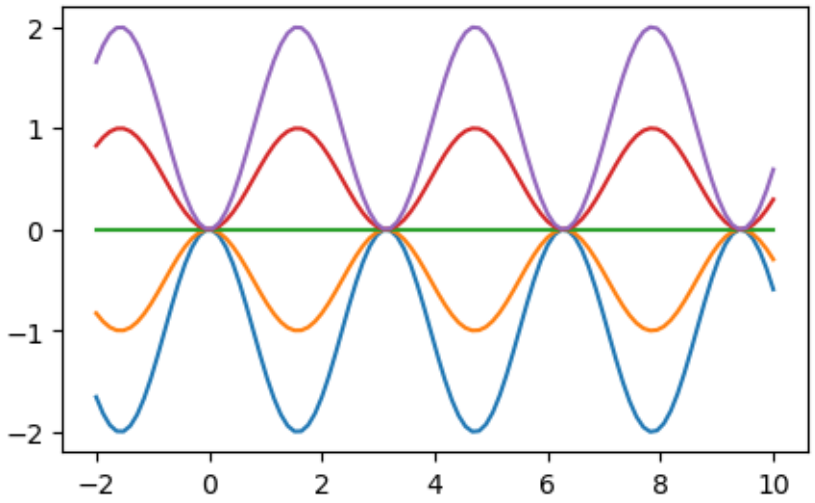
[69]:
$$y(t) = C_1 \sin^2(t)$$

Da y(0) = 0 unabhängig von C1 gilt, gibt es unendlich viele Lösungen

[70]:
```
sol3aw = dsolve(dgl3, ics={y(1):1})
sol3aw
```

[70]:
$$y(t) = \frac{\sin^2(t)}{\sin^2(1)}$$

[71]:
```
tn = np.linspace(-2, 10, 100)
plt.figure()
for c1 in [-2, -1, 1e-10, 1, 2]:
    plt.plot(tn, lambdify(t, sol3.rhs.subs(S('C1'), c1))(tn))
```


2
1
0
−1
−2
−2
0
2
4
6
8
10

[ ]: