Hochschule München
Studiengang Informatik

Angewandte Mathematik
Übungsblatt 3

Bastian Kersting, Michael Schober, Elena Lilova
IF2B

28. Mai 2019

Inhaltsverzeichnis

1 Aufgabe: Analytisch **2**
1.1 Aufgabenteil a 2
1.2 Aufgabenteil b 4

2 Aufgabe: Numerisch **6**
2.1 Aufgabenteil a 6
2.2 Aufgabenteil b 8
2.3 Aufgabenteil c 10
2.4 Aufgabenteil d 11

1 Aufgabe: Analytisch

1.1 Aufgabenteil a

Zu untersuchen ist die Abhängigkeit der Fallgeschwindigkeit ν von der von der Fallzeit t einer frei fallenden Masse unter Berücksichtigung des Luftwiederstandes. Die auf die Masse wirkenden Kräfte lassen sich durch folgende Ortsgleichung beschreiben:

$$-gm + k\left(\frac{d}{dt}x(t)\right)^2 + m\frac{d^2}{dt^2}x(t) = 0$$

Daraus lässt sich folgende Geschwindigkeitsgleichung bilden:

$$-gm + kv^2(t) + m\frac{d}{dt}v(t) = 0$$

Die Funktion v(t) wird mittels 'dsolve' ermittelt:

$$v(t) = -\frac{\sqrt{g}\sqrt{m}}{\sqrt{k}\tanh\left(\frac{\sqrt{g}\sqrt{k}\log\left(C_1 e^{-2t}\right)}{2\sqrt{m}}\right)}$$

Zur Vereinfachung wird die Gleichung umgestellt:

$$v(t) = -\sqrt{\frac{k}{gm}}\tanh\left(0.5\sqrt{\frac{gk}{m}}\left(-2t + \log\left(C_1\right)\right)\right)$$

Zum Zeitpunkt null gilt v(t) = t = 0. Umstellen ergibt für die Konstante C1 = 1. Zuletzt wird die Funktion v(t) auf t abgebildet. Dabei gilt: Erdbeschleunigung g = 9.81, Masse m = 1, Wiederstandswert k = 1.

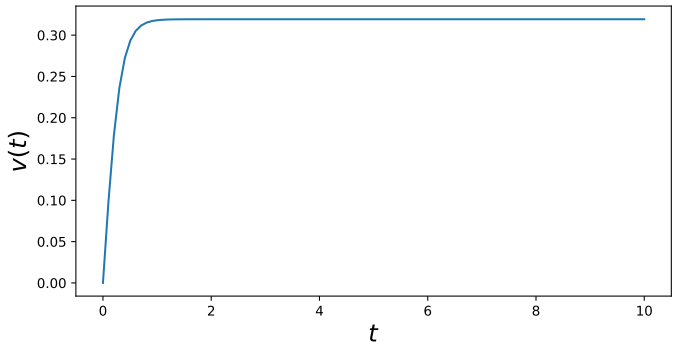

Abbildung 1: Masse im freien Fall im Zeitraum von 0 - 10 Sekunden.

Im folgenden der Quellcode:

```
from sympy import symbols
from sympy import Function, dsolve, Eq, sin, cos, symbols, solve,latex
from sympy import init_printing, sqrt, tanh, log, Derivative
init_printing(use_latex=True)


k, m, g, t = symbols('k m g t')

x = Function("x")
free_fall1 = Derivative(x(t),t,2)*m + k* Derivative(x(t),t)**2 - g*m
#print(latex(eval('free_fall1')))

v = Function('v')
free_fall2 = Derivative(v(t),t)*m + k* v(t)**2  - g*m
#print(latex(eval('free_fall2')))

sol = dsolve(free_fall2)
#print(latex(eval('sol')))


fig, ax = plt.subplots(figsize=(8, 4))

x = linspace(0,10,100)

newEq = Function('newEq')
newEq = -sqrt(1/9.81*1) * tanh (0.5*sqrt(9.81*1/1) * (-2*t+log(1)))
#print(latex(eval('newEq')))

ys = []
for xs in x:
    l = newEq.subs(t,xs)
    ys.append(l)

plt.plot(x,ys)
ax.set_xlabel(r"$t$", fontsize=18)
ax.set_ylabel(r"$v(t)$", fontsize=18)
plt.show()
fig.savefig('v_on_t.pdf')
```

1.2 Aufgabenteil b

In diesem Aufgabenteil wird ein schwach gedämpftes, schwingungsfähiges System betrachtet. Folgende Differentialgleichung gilt es zu lösen:

$$\ddot{x} + 2\delta\dot{x} + \omega_0^2 x = \frac{F_0}{m}\cos\omega_0 t$$

Der rechte Teil der Gleichung beschreibt die Schwingung, wobei δ den Dämpfungsfaktor und ω_0 die Eigenkreisfrequenz darstellt. Der rechte Teil der Gleichung beschreibt eine konstant anregende Kraft.
Wir lösen die Gleichung in Python mithilfe der Sympy Bibliotheksmethode dsolve() analytisch und kommen zu folgendem Ergebnis:

$$x(t) = 1.0C_1 e^{-\gamma t - t\sqrt{\gamma-\omega}\sqrt{\gamma+\omega}} + 1.0C_2 e^{-\gamma t + t\sqrt{\gamma-\omega}\sqrt{\gamma+\omega}} + \frac{1.25\sin(\omega t)}{\gamma\omega}$$

Die Gleichung wurde mithilfe der $e^{\lambda t}$ Methode gelöst, wobei λ für die Eigenvektoren des Systems steht. Um die zwei Unbekannten C_1 und C_2 zu eliminieren, werden die Anfangsbedingungen $x(0) = \dot{x}(0) = 0$ eingesetzt. Es ergibt sich folgendes Ergebnis:

$$x(t) = \frac{0.625e^{-\gamma t - t\sqrt{\gamma-\omega}\sqrt{\gamma+\omega}}}{\gamma\sqrt{\gamma^2-\omega^2}} - \frac{0.625e^{-\gamma t + t\sqrt{\gamma-\omega}\sqrt{\gamma+\omega}}}{\gamma\sqrt{\gamma^2-\omega^2}} + \frac{1.25\sin(\omega t)}{\gamma\omega}$$

Zuletzt wird die Gleichung noch für verschiedene γ in einem Bereich von 0 bis 5 geplottet:

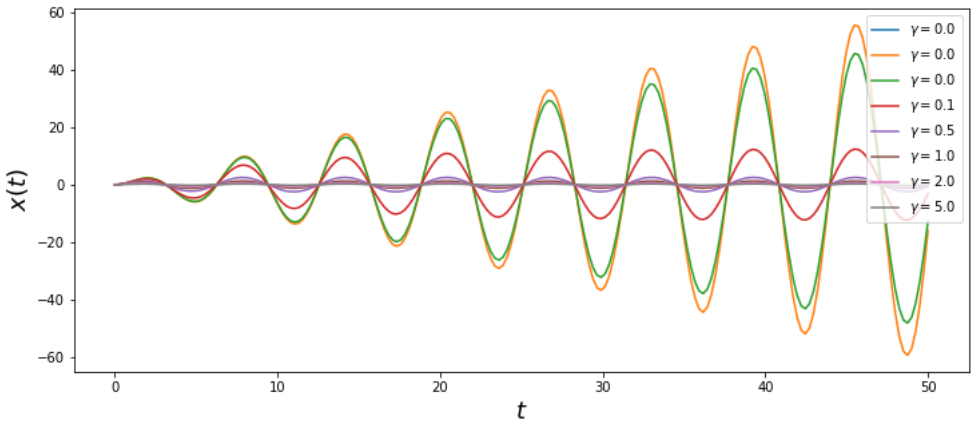

Abbildung 2: Schwach gedämpfte Schwingung mit Parameter γ zwischen 0 und 5

Im folgenden der Quellcode:

```
def apply_ics(sol, ics, x, known_params):
    free_params = sol.free_symbols - set(known_params)
    eqs = [(sol.lhs.diff(x, n) - sol.rhs.diff(x, n)).subs(x, 0).subs(ics)
           for n in range(len(ics))]
    sol_params = sympy.solve(eqs, free_params)
    return sol.subs(sol_params)

t, gamma, omega = sympy.symbols("t gamma omega", positive = True)
F , m = 5,2
x = sympy.Function("x")

dgl = x(t).diff(t,2)+ 2 * gamma * x(t).diff(t,1)
        + omega**2 * x(t) - F / m * sympy.cos(omega * t)

ics = {x(0): 0,x(t).diff(t).subs(t, 0): 0}
dgl_sol = sympy.simplify(sympy.dsolve(dgl))
sol = apply_ics(dgl_sol, ics, t, [omega, gamma])
x_critical = sympy.limit(sol.rhs, gamma, omega)

fig, ax = plt.subplots(figsize=(12, 5))

x = np.linspace(0,50,200)
for g in [0,0.001,0.01,0.1, 0.5,1.0, 2.0, 5.0]:
    if g == 1:
        x_t = sympy.lambdify(t,
            x_critical.subs({omega: 1.0}), 'numpy')
    else:
        x_t = sympy.lambdify(t, sol.rhs.subs({gamma: g,
            omega: 1.0}),"numpy")
    ax.plot(x, x_t(x).real, label=r"$\gamma = %.1f$" % g)

ax.set_xlabel(r"$t$", fontsize=18)
ax.set_ylabel(r"$x(t)$", fontsize=18)
ax.legend()
```

2 Aufgabe: Numerisch

2.1 Aufgabenteil a

In diese Aufgabe muss die Wurfbewegung eines Objekts durch numerisches lösen der Gleichung $\vec{a} = \vec{g}$ bestimmt werden, wobei g die Gravitationskonstante g = 9,81$\frac{m}{s^2}$ ist. Beim waagerechten Wurf, ist die Anfangsgeschwindigkeit in horizontaler und vertikaler Richtung ungleich Null.

$$\vec{V_0} = \begin{pmatrix} V_0, x \\ V_0, y \end{pmatrix}$$

Für unseren Beispiel haben wir als Anfangsgeschwindigkeit den Vektor $\vec{V_0} = \begin{pmatrix} 30 \\ 40 \end{pmatrix}$ genommen. Für die Geschwindigkeit in Abhängigkeit von der Zeit gilt:

$$\vec{V_{(\mathrm{t})}} = \begin{pmatrix} V_0, x \\ -gt + V_0, y \end{pmatrix}$$

Und für die Position in Abhängigkeit von der Zeit:

$$\vec{r_{(\mathrm{t})}} = \begin{pmatrix} V_{0,\mathrm{x}}, t + x_0 \\ -\frac{1}{2}gt^2 + V_{0,\mathrm{y}}t + Y_0 \end{pmatrix}$$

Um die Lösung numerisch zu bestimmen, wird die 'odeint' Funktion benutzt. Die Startwerte zum Zeitpunkt t = 0 sind dabei x = 0, y = 0 für den Ort, die Anfangsgeschwindigkeit in unseren Fall (30, 40) und die Werte der Gleichung zweiter Ordnung: $\vec{a} = 0$ und $\vec{g} =$ - g
Die Gleichung wird für eine Periode von 0 bis 10 Sekunden in numerischer und analytischer Darstellung geplottet:

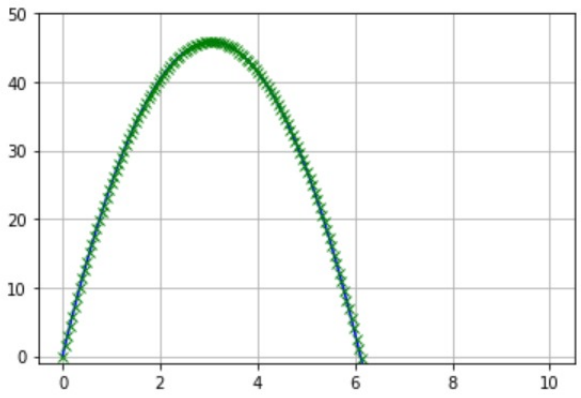

Abbildung 3: Blau: Analytisch erechnete Kurve, Gepunktet: Numerische Lösung

Nun galt es die Wurfweite in Abhängigkeit des Abwurfwinkels zu bestimmen. Dazu wurde der Geschwindigkeitsvektor für Werte zwischen null und neunzig Grad berechnet und dann geschaut, wo es eine Nullstelle gibt, also wie weit das Objekt in x-Richtung geworfen wurde.

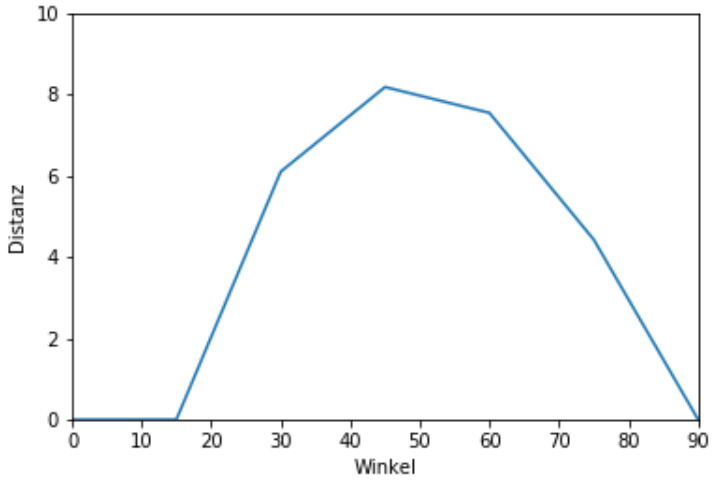

Abbildung 4: Wurfweite als Funktion des Abwurfwinkels

Wie zu erwarten sind die Weiten bei null bzw. neunzig Grad sehr gering und im Bereich dazwischen höher.

Im folgenden der Quellcode:

```
z0 = [0,30,0,40]
g = 9.81
t0 = np.linspace(0,10,200)

def x(z,t):
    x0 = z[0]
    x1 = z[1]
    x2 = z[2]
    x3 = z[3]
    return [x1,-g,x3,0]

solx = odeint(x,z0,t0)

plt.plot(t0, (30*np.sin(np.radians(90))*t0-0.5*g*t0**2)
,label="analytisch", color = "blue")
plt.plot(t0, solx[:,0], "rx", label="odeint", color = "green")
plt.ylim(-1,50)
plt.grid()

def get_root(y):  # Array der y-Werte
    index = 0
```

```
    y = y[5:]
    for elem in y:
        nullpoint = -1
        if elem < 0.4 and elem > -0.4:
            nullpoint = index
            break
        index += 1
    return nullpoint

roots = []
for alpha in range(0,105,15):
    delta_x = 10 * np.cos(np.radians(alpha))
    delta_y = 10 * np.sin(np.radians(alpha))

    inter_sol = odeint(func = x, y0 = [0,delta_y,0,delta_x],t = t0)
    y_range = inter_sol[:,0]
    x_range = inter_sol[:,2]
    y_nullpoint = get_root(y_range)
    x_nullpoint = x_range[y_nullpoint]
    roots.append(x_nullpoint)


plt.plot(range(0,105,15),roots)
plt.ylim(0,10)
plt.xlim(0,90)
plt.xlabel("Winkel")
plt.ylabel("Distanz")
```

2.2 Aufgabenteil b

Eine Schwingung dissipiert durch Reibung Energie, wodurch die Amplitude im Zeitverlauf abnimmt. Betrachtet wird das Verhalten eines Federpendels zu verschiedenen Reibungsstärken k.

Die Schwingung lässt sich mit folgender Gleichung beschreiben:

$$Dx(t) + k\left|\frac{d}{dt}x(t)\right|^n \operatorname{sign}\left(\frac{d}{dt}x(t)\right) + m\frac{d^2}{dt^2}x(t) = 0$$

Um die Lösung numerisch zu bestimmen, wird das Gleichungssystem 2. Ordnung in einen Vektor mit zwei Gleichungssystemen erster Ordnung überführt. Dieser wird der 'odeint' Funktion übergeben. Die Startwerte zum Zeitpunkt t = 0 sind dabei x = 1 für den Ort und $\dot{x} = 0$ für momentane Steigung. Für die Variablen gilt: Masse m = 0.2, Fedekonstante D = 2.

Für verschiedene Werte k und n werden folgende Graphen geplottet:

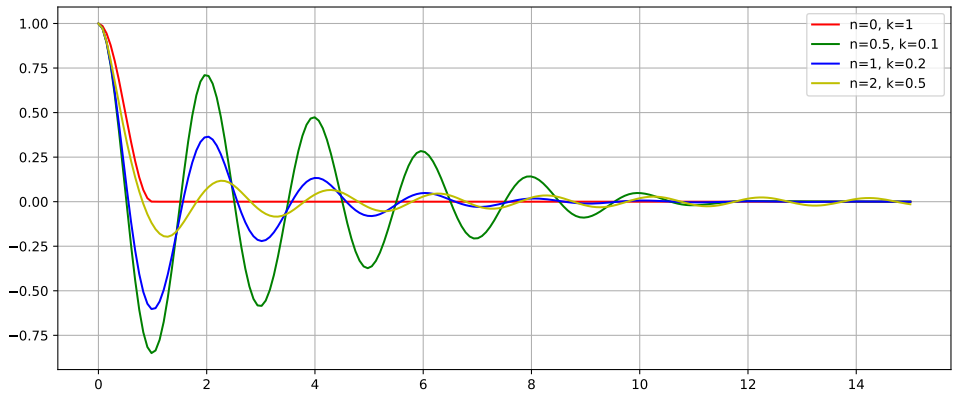

Abbildung 5: Amplitude über die Zeit t = 0 - 15 Sekunden für verschiedene Reibungstärken k.

Im folgenden der Quellcode:

```
fig, ax = plt.subplots(figsize=(12, 5))

m = 0.2
D = 2
z0 = [1,0]

t = np.linspace(0,15,200)

def dF(z,t):
    return[ z[1], -(k/m)* abs(z[1])** n * sign(z[1]) - (D/m)*z[0] ]

n = 0
k = 1
f1 = odeint(func=dF, y0=z0, t=t)
n = 0.5
k = 0.1
f2 = odeint(func=dF, y0=z0, t=t)
n = 1
k = 0.2
f3 = odeint(func=dF, y0=z0, t=t)
n = 2
k = 0.5
f4 = odeint(func=dF, y0=z0, t=t)

plt.plot(t, f1[:,0],"r", label="n=0, k=1")
plt.plot(t, f2[:,0],"g", label="n=0.5, k=0.1")
```

```
plt.plot(t, f3[:,0],"b", label="n=1, k=0.2")
plt.plot(t, f4[:,0],"y", label="n=2, k=0.5")

plt.legend()
plt.grid(True)
plt.show()
fig.savefig('x_to_t.pdf')
```

2.3 Aufgabenteil c

In dieser Aufgabe wird eine nichtlineare Schwingung durch eine vorgegeben Differentialgleichung dargestellt. Die Differentialgleichung lautet:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0$$

mit g = 9,81$\frac{m}{s^2}$ und l = 10 cm. Die Gleichung beschreibt also ein Pendel, welches abhängig vom Auslenkungswinkel θ schwingt. Plottet man das System für Auslenkungen zwischen null und neunzig Grad erhält man folgenden Plot:

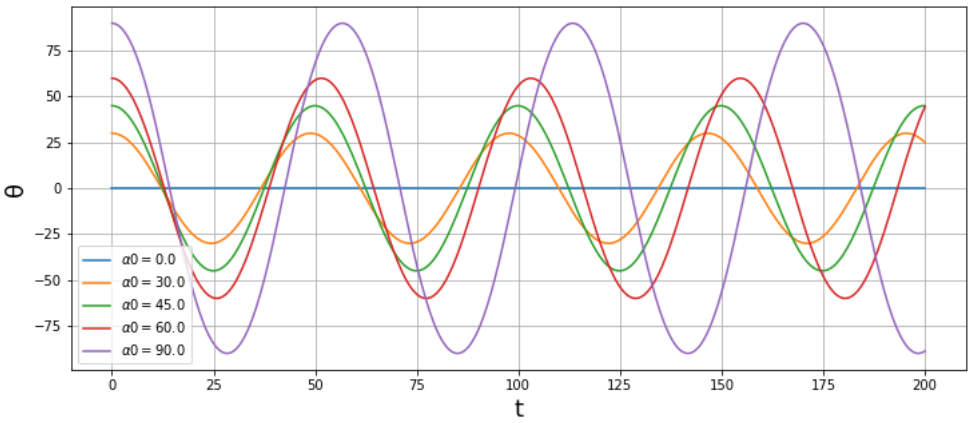

Abbildung 6: Schwingung, abhängig vom Auslenkungswinkel θ

Betrachtet man nun die Schwingung, so fällt auf, dass die Amplitude proportional zum Auslenkungswinkel steigt.
Gelöst wurde die Gleichung numerisch, dazu wurde die Methode odeint, aus der Bibliothek scipy benutzt. Die Lösung wurde mit folgendem Python Code gelöst:

```
fig, ax = plt.subplots(figsize=(12, 5))

g = 9.81
l = 10
```

```
t = np.linspace(0,200,400)

def dg(z,t):
    return [z[1], -g/l * np.sin(np.radians(z[0]))]

for alpha0 in [0, 30,45,60, 90]:
    z0 = [alpha0, 0]
    z = odeint(func=dg, y0=z0, t=t)
    ax.plot(t,z[:,0],label=r"$\alpha0 = %.1f$" % alpha0 )

ax.set_xlabel("t",fontsize=18)
ax.set_ylabel("$\theta$",fontsize=18)
plt.legend()
plt.grid()
plt.show()
```

2.4 Aufgabenteil d

In dieser Aufgabe ging es darum, eine erzwungene Schwingung numerisch zu beschreiben. Die Differentialgleichung zur Schwingung war gegeben:

$$m\ddot{x} + c\dot{x} + Dx = F_0 \cos \Omega t$$

Wie schon in Aufgabe 1b steht der rechte Teil der Gleichung für eine konstant anregende Kraft. Gelöst wurde das Gleichungssystem, wie auch in den vorherigen Aufgaben, mit der odeint Methode. Setzt man für $m = 0.5, c = d = F_0 = 1$ und Ω=20 ein ergibt sich der folgende Plot:

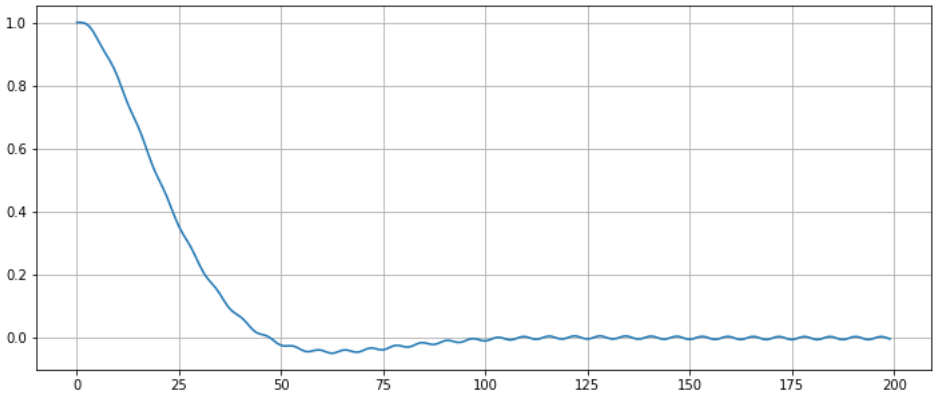

Abbildung 7: Erzwungene Schwingung eines gedämpften harmonischen Oszillators

Nun wurde die Amplitude im eingeschwungenen Zustand in Abhängigkeit zur Frequenz Ω dargestellt. Als Wertebereich für Ω wurde das Intervall $I = [0, 20]$ festgelegt.

Da aus den Plots für veschiedene Frequenzen < 20 abzulesen war, dass sich der eingeschwungene Zustand weiter in positive x-Richtung verschiebt wurde als Beginn des eingeschwungenen Zustands der Punkt x = 125 festgelegt. Er garantiert auch, dass bei $\Omega = 20$ das System eingeschwungen ist (siehe 7). Geplottet ergab sich folgendes Bild:

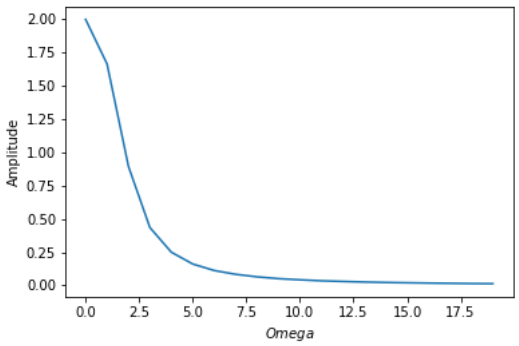

Abbildung 8: Erzwungene Schwingung eines gedämpften harmonischen Oszillators

Mit steigender Frequenz verringert sich die Amplitude unseres Systems im eingeschwungenem Zustand.

Im folgendem der Quelltext:

```
m = 0.5
c = 1
D = 1
om = 20
F = 1
t = np.linspace(0,10,200)

def dgl(z,t,om):
    z0 = z[0]
    z1 = z[1]
    z2 = (F * np.cos(om * t) - c * z1 - D * z0) / m
    return z1, z2

def get_amplitude(arr):
    # Parameter arr: Schwingung im eingeschwungenen Zustand
    return abs(arr.min()) + abs(arr.max())
amp = []
omega_list = range(0,20,1)
for omega in omega_list:
    func = odeint(func=dgl,t=t,y0=[1,0],args=(omega,))
    amp.append(get_amplitude(func[125:,0]))

plt.plot(omega_list,amp)
```