Hochschule Karlsruhe
University of
Applied Sciences

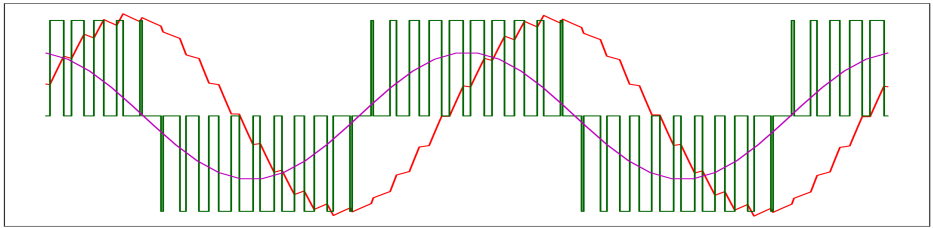
Fakultät für
Elektro- und
Informationstechnik

Labor Modellbildung und Simulation WS 21/22

Präsentation der Lösungswege - LGB04

Daniel Hauser, Dominik Axtmann, Julian Lickert | 26. Januar 2022

FAKULTÄT EIT

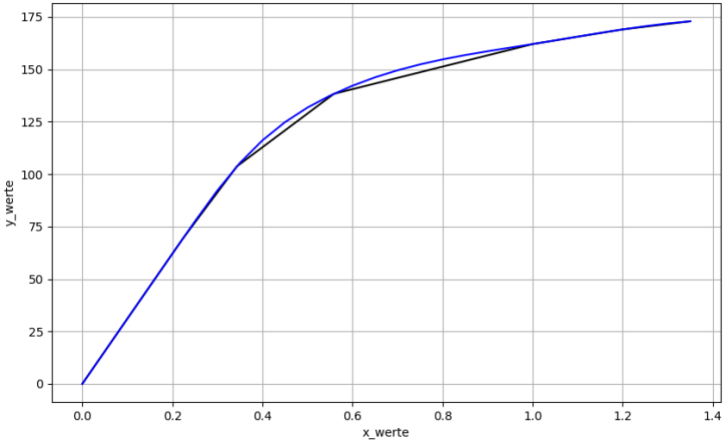


Kapitel 1 - Grundlagen Python

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Aufgabe:

- Nachbilden des gegebenen Diagramms mit Python
- Stützwerte aus .mat-Datei (schwarz)
- Kurvenverlauf interpolieren (blau)



Kapitel 1 - Grundlagen Python

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik

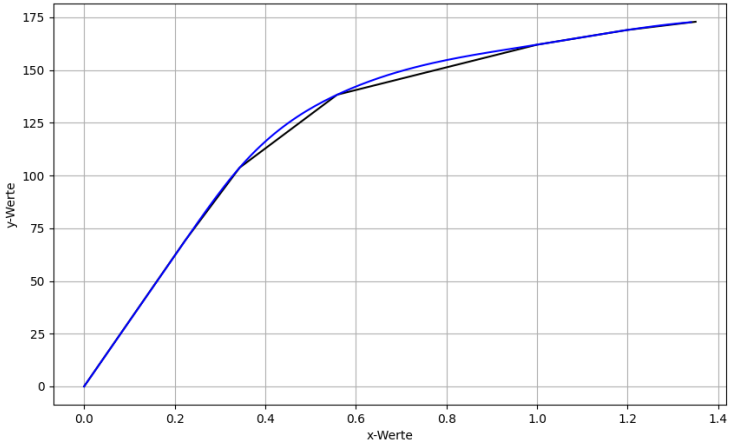
```
import numpy as np
import scipy.io
import scipy.interpolate
import matplotlib.pyplot as plt
from pathlib import Path

data_file = Path(__file__).with_name("mkl.mat")
raw_data = scipy.io.loadmat(data_file)
x_werte = raw_data["x_werte"][0]
y_werte = raw_data["y_werte"][0]

f_inter = scipy.interpolate.interp1d(x_werte, y_werte, kind="cubic")
mehr_x_werte = np.arange(x_werte[0], x_werte[-1], 0.01)
```

Kapitel 1 - Grundlagen Python


```
fig = plt.figure(1, figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1)
ax.grid()
ax.set_xlabel("x_werte")
ax.set_ylabel("y_werte")
ax.plot(x_werte, y_werte, color="black")
ax.plot(mehr_x_werte, f_inter(mehr_x_werte), color="blue")

plt.show()
```

Kapitel 1 - Grundlagen Python

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik

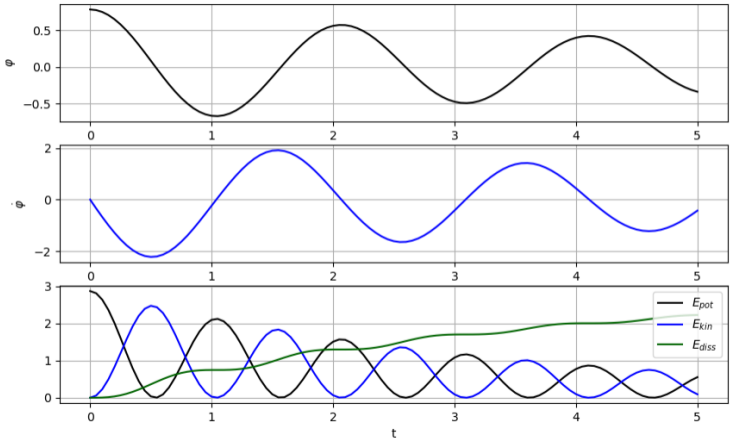


Kapitel 2 - Lösen von gewöhnlichen DGLs

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Aufgabe:

- Energetische Analyse eines mathematischen Pendels mit Dämpfung
- Nachbilden des gegebenen Diagramms mit Python
- Überlegung zu linearisierten DGLs



Kapitel 2 - Lösen von gewöhnlichen DGLs

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

$$\ddot{\varphi} + d * \dot{\varphi} + \frac{g}{l} * \varphi = 0$$

mit:

$g = 9,81 \frac{m}{s^2}$

$l = 1m$

$d = 0,3$

$\varphi_{(t=0)} = \frac{\pi}{4}$

```
t_max = 5
phi_0 = pi / 4
l = 1
d = 0.3

t = np.linspace(0, t_max, 101)

def fkt(t, x, *args):
    return [x[1], -g/l*np.sin(x[0])-d*x[1]]
```

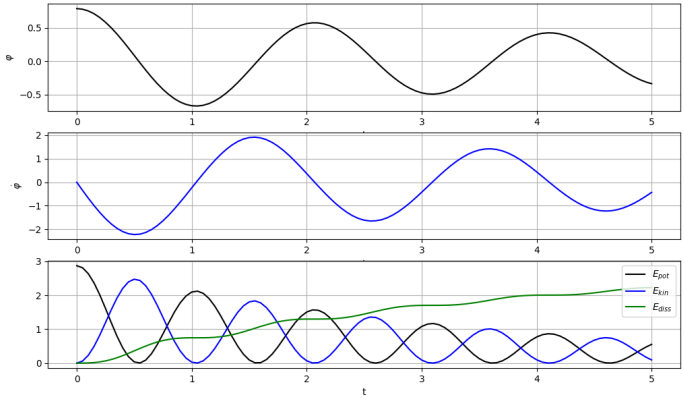
Kapitel 2 - Lösen von gewöhnlichen DGLs

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

```
x_0 = [phi_0, 0]
sol = solve_ivp(fkt, [0, t_max], x_0, t_eval=t, method='LSODA',
                args=(g, l))
t = sol.t
phi = sol.y[0]
phi_dot = sol.y[1]

Epot = g * l * (1 - np.cos(phi))
Ekin = 1/2 * phi_dot**2
Ediss = Epot[0] - (Epot + Ekin)
```

Kapitel 2 - Lösen von gewöhnlichen DGLs

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

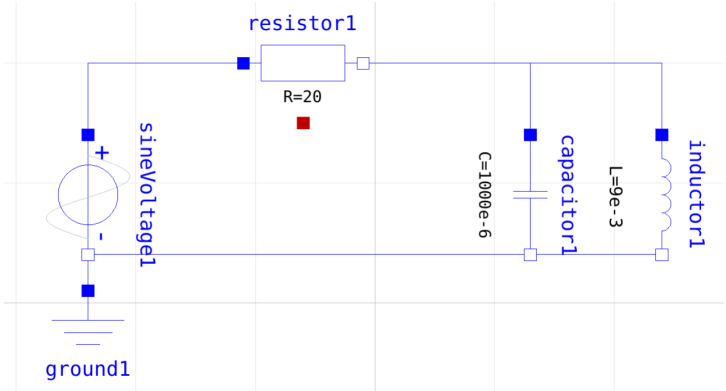


Welches Problem tritt auf, wenn Sie mit der linearisierten Differentialgleichung rechnen?

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Aufgabe:

- Aufstellen der Übertragungs-funktion G(s)
- Pole von G und Eigenwerte der Systemmatrix berechnen
- Gedämpfte Eigenfrequenz berechnen
- Bode-Diagramm zeichnen



Kapitel 3 - Simulationstechniken

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Übertragungsfunktion:

$$G(s) = \frac{U_a(s)}{Ue(s)} = \frac{Z_C(s)||Z_L(s)}{R+(Z_C(s)||Z_L(s))} \qquad mit\ Z_C(s)\,||Z_L(s) = \frac{\frac{1}{s*C}*s*L}{\frac{1}{s*C}+s*L} = \frac{s*L}{1+s^2*L*C}$$

$$= \frac{\frac{s*L}{1+s^2*L*C}}{R+\frac{s*L}{1+s^2*L*C}} = \frac{s*L}{R+s^2*R*L*C+s*L}$$

$$= \frac{\frac{L}{R}*s}{L*C*s^2+\frac{L}{R}*s+1} \qquad \text{mit } R = 20\,\Omega, C = 1\,\mathrm{mF}, L = 9\,\mathrm{mH}$$

Kapitel 3 - Simulationstechniken

Pole:

$$G(s) = \frac{\frac{L}{R} * s}{L * C * s^2 + \frac{L}{R} * s + 1}$$

$$s_{1,2} = \frac{-\frac{L}{R} \pm \sqrt{\frac{L^2}{R^2} - 4 * L * C}}{2 * L * C} \quad \text{mit } R = 20\,\Omega, C = 1\,\mathrm{mF}, L = 9\,\mathrm{mH}$$

$$s_1 = -25 + 332,4i$$
$$s_2 = -25 - 332,4i$$

Kapitel 3 - Simulationstechniken

Eigenwerte der Systemmatrix:

$$0 \stackrel{!}{=} \begin{vmatrix} -\frac{1}{R \cdot C} - \lambda & -\frac{1}{C} \\ \frac{1}{L} & 0 - \lambda \end{vmatrix}$$

$$0 = \left(\frac{-1}{R \cdot C} - \lambda\right)(-\lambda) - \left(\frac{1}{L}\right)\left(\frac{-1}{C}\right)$$

$$\lambda_{1,2} = \frac{-1}{2RC} \pm \sqrt{\frac{1}{4R^2C^2} - \frac{1}{LC}}$$

$$\lambda_1 = -25 + 332,4i$$

$$\lambda_2 = -25 - 332,4i$$

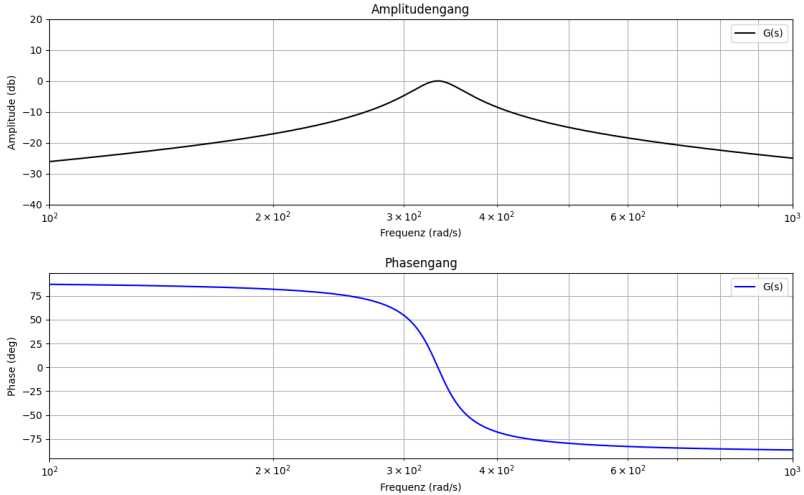
```
num = [L/R, 0]
den = [L*C,L/R,1]
sys = signal.TransferFunction(num, den)
w, mag, phase = signal.bode(sys, n = 10000)
ss= sys.to_ss()
sysmatrix = ss.A
```

Kapitel 3 - Simulationstechniken


```
print("Pole: " + str(sys.poles))
>>> Pole: [-25.+332.39451125j -25.-332.39451125j]

print("Systemmatrix: \n" + str(sysmatrix))
>>> Systemmatrix:
    [[-5.00000000e+01 -1.11111111e+05]
    [ 1.00000000e+00  0.00000000e+00]]

print("Eigenwerte: " + str(linalg.eigvals(sysmatrix)))
>>> Eigenwerte: [-25.+332.39451125j -25.-332.39451125j]

print("Eigenfrequenz: " + str(round((1/(2*pi*np.sqrt(L*C))), 2)) + " Hz")
>>> Eigenfrequenz: 53.05 Hz
```

Kapitel 3 - Simulationstechniken

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik

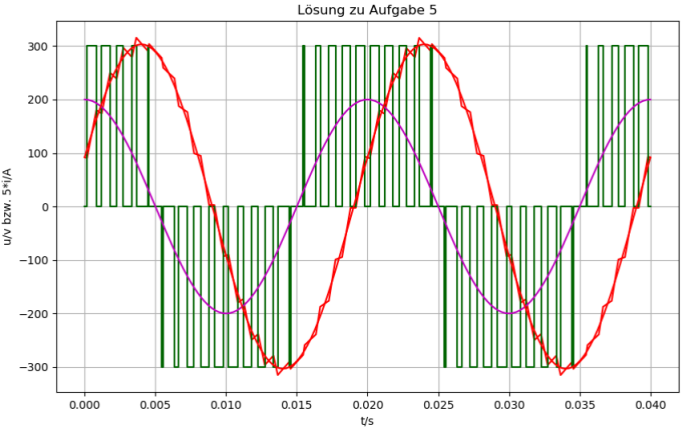


Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Aufgabe 1:

- Vergleichen des Stromverlaufs bei kontinuierlichem Spannungsverlauf und PWM
- Nachbilden des gegebenen Diagramms mit Python



Kapitel 4 - Vierquadrantensteller


```
def RL_circuit(duty, T_p, N, u_0, i_0, R, L):
    T = L/R
    N_1 = round((1-duty)/2*(N-1))
    if N_1 < 1:
        N_1 = 1
    N_2 = (N-1)-2*N_1
    if N_2 < 1:
        if N % 2:
            N_2 = 2
        else:
            N_2 = 1
    N_1 = (N-1-N_2)/2
    t_ein = duty*T_p
    t_0_halbe = (T_p-t_ein)/2
    Delta_t_1 = t_0_halbe/N_1
    Delta_t_2 = t_ein/N_2
```

Kapitel 4 - Vierquadrantensteller


```
t_a_vec = Delta_t_1*np.arange(0, N_1+1)
t_b_vec = Delta_t_2*np.arange(1, N_2)
t_c_vec = t_a_vec
i_0_neu = i_0
i_a_vec = i_0_neu*np.exp(-t_a_vec/T)
i_8 = u_0/R
i_0_neu = i_a_vec[-1]
i_b_vec = i_8+(i_0_neu-i_8)*np.exp(-t_b_vec/T)
i_0_neu = i_8+(i_0_neu-i_8)*np.exp(-t_ein/T)
i_c_vec = i_0_neu*np.exp(-t_c_vec/T)
t = t_a_vec
t = np.append(t, t_b_vec+t_0_halbe)
t = np.append(t, t_c_vec+t_0_halbe+t_ein)
i = i_a_vec
i = np.append(i, i_b_vec)
i = np.append(i, i_c_vec)
return i, t
```

Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

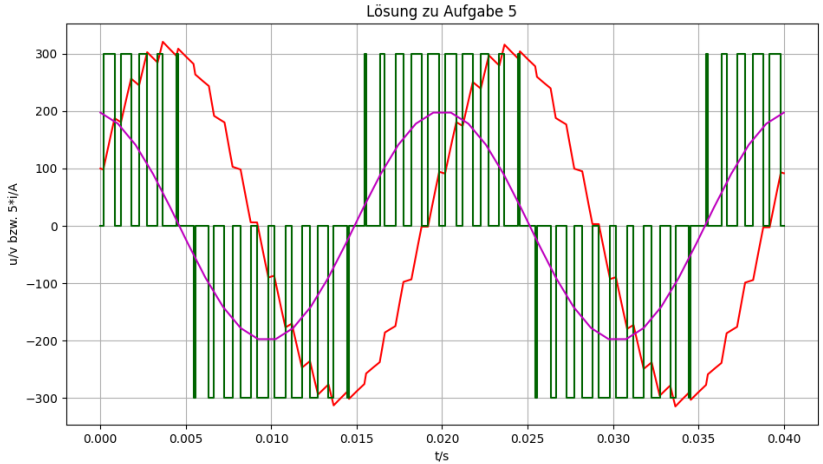
```
def u_verlauf(duty, T_p, u_0):
    t_ein = duty*T_p
    t_0_halbe = (T_p-t_ein)/2
    t_u = np.array([0, t_0_halbe, t_0_halbe+t_ein, T_p])
    u = np.array([0, u_0, 0, 0])
    return u, t_u
```

Kapitel 4 - Vierquadrantensteller


```
kk = np.arange(0, int(t_sim/T_p))
t_sin = np.linspace(0, t_sim, len(kk))
u_mittel_vec = 200*np.cos(2*np.pi*50*(kk*T_p+T_p/2))

for k in kk:
    t_akt = k*T_p+T_p/2
    u_mittel = u_mittel_vec[k]
    duty = abs(u_mittel/u_0)
    u_abs = - u_0 if u_mittel < 0 else u_0
    i, t = RL_circuit(duty, T_p, N, u_abs, i_0_neu, R, L)
    t_vec = np.append(t_vec, t+t_0_neu)
    i_vec = np.append(i_vec, i)
    u, t_u = u_verlauf(duty, T_p, u_abs)
    t_u_vec = np.append(t_u_vec, t_u+t_0_neu)
    u_vec = np.append(u_vec, u)
    i_0_neu = i[-1]
    t_0_neu = t_vec[-1]
```

Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik

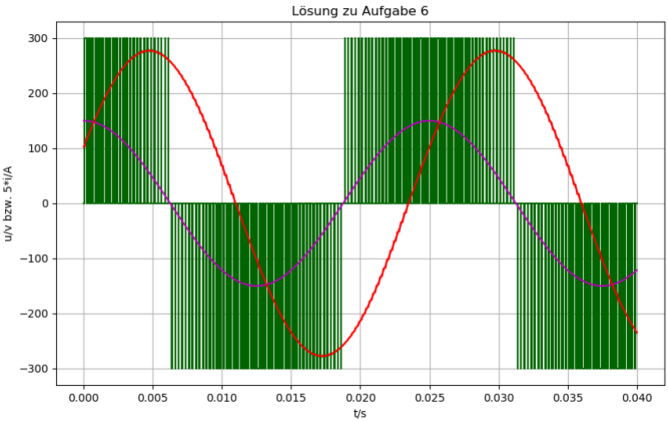


Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

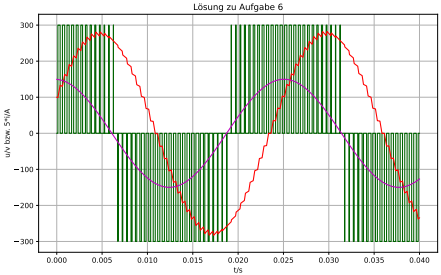
Aufgabe 2:

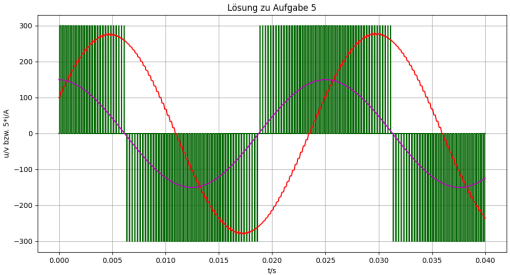
- Auswirkungen der Parameter auf den resultierenden Stromverlauf



Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik



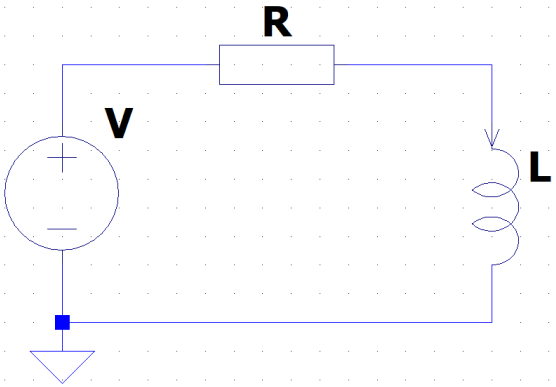


Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Aufgabe 3:

- Lösung der DGL für das RL-Glied mit Hilfe der Laplacetransformation



Kapitel 4 - Vierquadrantensteller

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

$$i(t) = C_1 e^{-\frac{Rt}{L}} + \frac{u_0}{R}$$

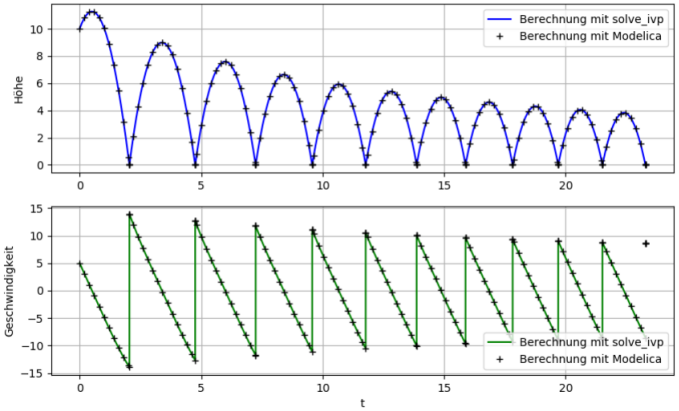
```
import sympy as sym

# Symbole für SymPy erzeugen
l, r, t, u0 = sym.symbols('L R t u0')
i = sym.Function('i')

#DGL aufstellen
dgl = sym.Eq(i(t).diff(t)*l + r*i(t),u0)

#DGL lösen
dgl_sol = sym.dsolve(dgl, i(t))

#DGL als tex
print(sym.latex(dgl_sol))
```

Kapitel 5 - Numerische Lösungsverfahren

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Aufgabe:

- Nachbilden des gegebenen Diagramms
- Warum bewegt sich der Ball für t=0 nach oben?



Kapitel 5 - Numerische Lösungsverfahren

$$m\dot{v} + mg + cv^3 = 0 \longrightarrow -g - \alpha * v^3 = \dot{v}$$
$$h = 0 \Longrightarrow v := -v$$

```
def xdot_fkt(t, x, *args):
    theta, omega = x
    xdot= [x[1], -g-alpha*x[1]**3]
    return xdot

def hit_ground(t, y, *args):
    return y[0]
```

```
equation
  v=der(h);
  -g-alpha*v^3=der(v);
  when {h <= 0.0} then
    reinit(v, -pre(v));
  end when;
```

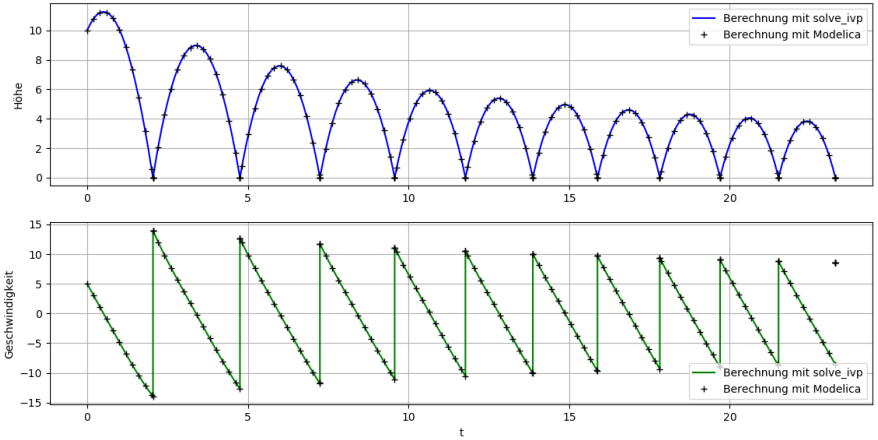
Kapitel 6 - Eigene Modelle in Modelica

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik

```
modelname = 'bounce'
mod = ModelicaSystem(modelname+'.mo', modelname)
mod.setSimulationOptions('stopTime={}'.format(t_ges[-1]))
mod.simulate()
[t_vec_mo] = mod.getSolutions('time')
[h_mo] = mod.getSolutions('h')
[v_mo] = mod.getSolutions('v')
delete_OM_files(modelname)
```

Kapitel 5 - Numerische Lösungsverfahren

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik

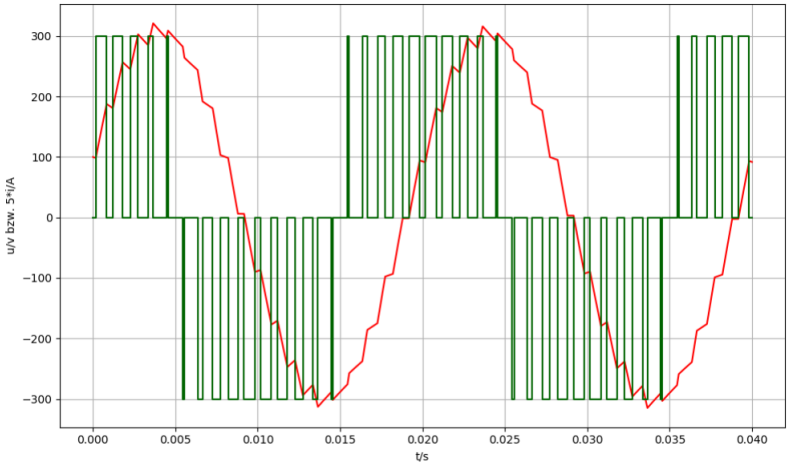


Kapitel 6 - Eigene Modelle in Modelica

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

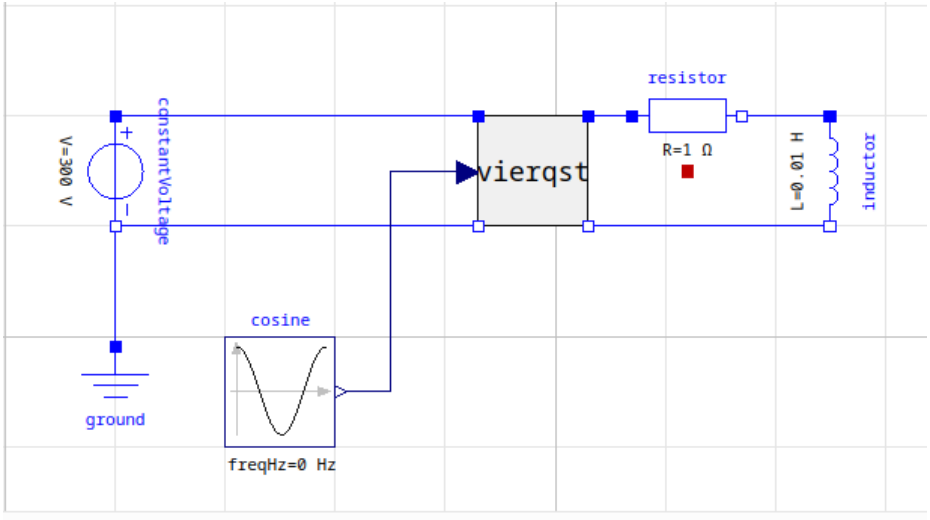
Aufgabe:

- Nachbilden des gegebenen Diagramms mit Modelica
- Bauteil Parameter aus Kapitel 4



Kapitel 6 - Eigene Modelle in Modelica

Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

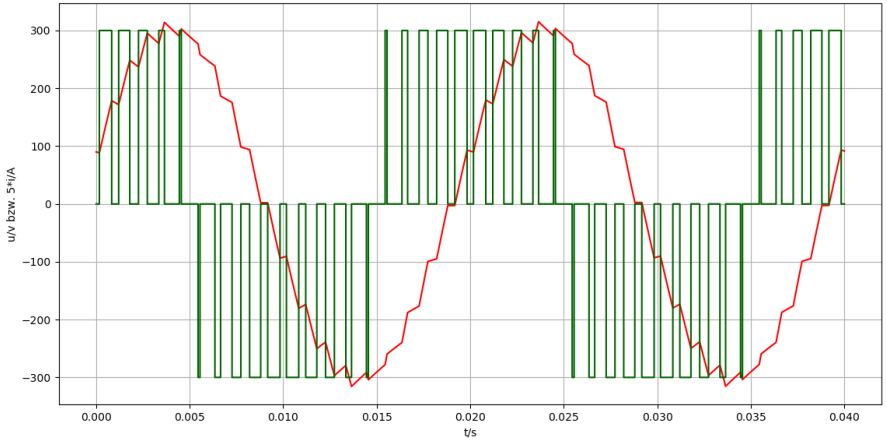


Kapitel 6 - Eigene Modelle in Modelica


```
f = 50
a = 200
L = 10e-3
R = 1

mod.setSimulationOptions('stopTime={}'.format(t_ges[-1]))
mod.setParameters(f'f={f}')
mod.setParameters(f'cosine.amplitude={a}')
mod.setParameters(f'inductor.L={L}')
mod.setParameters(f'resistor.R={R}')
```

Kapitel 6 - Eigene Modelle in Modelica

Hochschule Karlsruhe
University of
Applied Sciences
Fakultät für
Elektro- und
Informationstechnik



Vielen Dank für Ihre Aufmerksamkeit