Institut für Grundlagen und Theorie der Elektrotechnik

Elektrische Netzwerke und Mehrtore
Übung

Wintersemester 2020

Protokoll Übung 6: Fourieranalyse

Gruppe: 04

Gruppenteilnehmer:

1. Matthias Fottner
2. David Keller
3. Moritz Woltron

Vortragende: Helena Grabner

Graz, am 4. Dezember 2020

Inhaltsverzeichnis

1 Numerische Bestimmung der Fourierreihenkoeffizienten von $u_{in}(t)$ 3

2 Plot der ersten 5 Schwingungen 4

3 Plot der ersten 250 Wellen 5

4 Bestimmen der Induktivität L 5

5 Bestimmen des Ausgangssignals $u_{out}(t)$ 7

6 Python-Skripte 8

6.1 plot_1.py 8

6.2 plot_2.py 10

6.3 plot_3.py 12

1 Numerische Bestimmung der Fourierreihenkoeffizienten von $u_{in}(t)$

Die Fourierreihe einer Funktion lässt sich folgendermaßen notieren:

$$a_0 + \sum_{k=1}^{\infty} [a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)]$$

Da es keinen Gleichanteil gibt, ist $a_0 = 0$. Aufgrund der ungeraden Symmetrie sind alle Koeffizienten $a_k = 0$. Die Koeffizienten b_k werden in Matlab numerisch mit der Trapezmethode angenähert. Mit der Funktion `b_k` (Kapitel 6.1, Z. 50-56) erhält man für die ersten Koeffiziente:

```
b_1 = 41.64106231
b_2 = -4.87735577
b_3 = 2.25902828
b_4 = -3.78545704
b_5 = 2.54658425
```

Abbildung 1: Die Grundschwingung b_1 und die ersten 4 Oberschwingungen b_{2-5}

2 Plot der ersten 5 Schwingungen

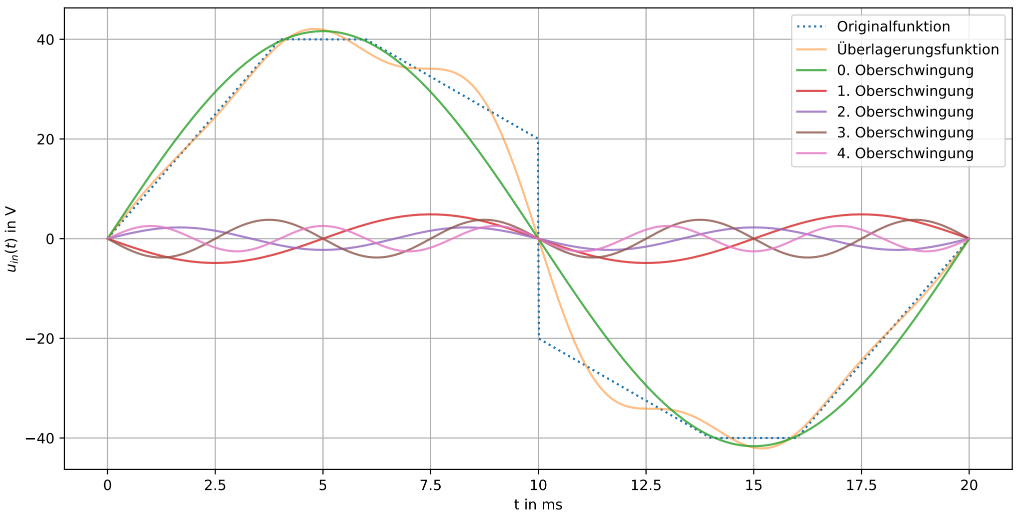

Abbildung 2: Plot der Grundschwingung und der ersten 4 Oberschwingungen, sowie deren Überlagerung

3 Plot der ersten 250 Wellen

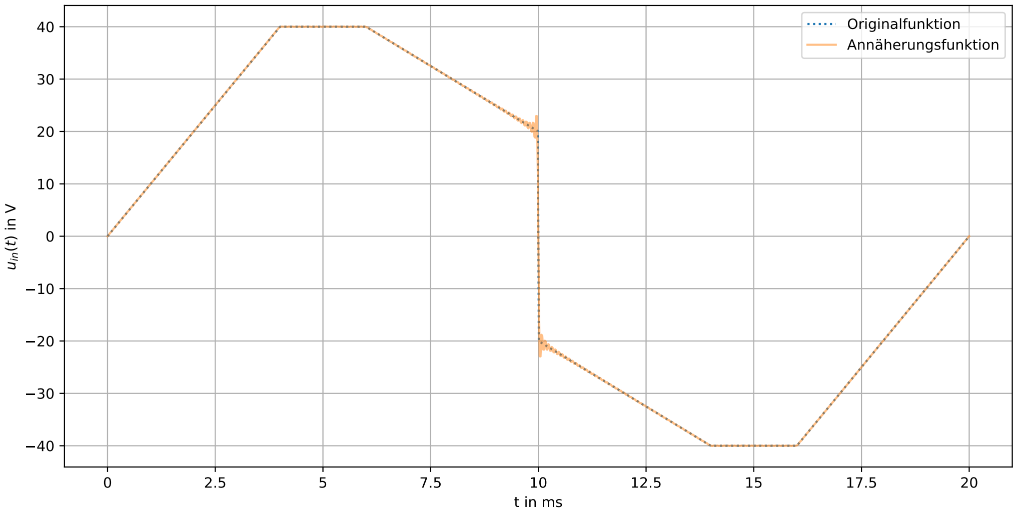

Abbildung 3: Plot der Fourierreihe mit den ersten 250 Wellen

Wie in Abbildung 3 ersichtlich, strebt die Fourierreihe(orange) gegen die Originalfunktion(blau). Da bei $t = \frac{T}{2}$ eine Sprungstelle existiert, nimmt die Fourierreihe dort den Mittelwert der Sprungstelle an. In diesem Fall ist das 0. Somit ist $u_{in}(t = \frac{T}{2}) = 0$.

4 Bestimmen der Induktivität L

Um die Induktivität der Spule zu bestimmen, ist es sinnvoll zuerst die Übertragungsfunktion $H(j\omega)$ zu bestimmen:

$$\begin{aligned}
\underline{H}(j\omega) &= \frac{\underline{U}_{out}}{\underline{U}_{in}} \\
&= \frac{R}{R + j\omega L} \\
&= \frac{1}{1 + \frac{j\omega}{\frac{R}{L}}} \\
\implies \Omega &= \frac{R}{L}
\end{aligned}$$

Die dritte Oberschwingung ω_3 soll bereits um $-3\,\text{dB}$ verstärkt werden. Das ist der Fall, wenn $\omega_3 \stackrel{!}{=} \Omega$ gilt.

$$\Omega = \omega_3 = 4 \cdot \omega_0 = 4 \cdot \frac{2\pi}{T} = 400\pi\ \frac{1}{\mathrm{s}}$$

$$\Omega = \frac{R}{L}$$

$$L = \frac{R}{\Omega} = \frac{2\pi\,\Omega}{400\pi\ \frac{1}{\mathrm{s}}}$$

$$= 6{,}67\,\mathrm{mH}$$

5 Bestimmen des Ausgangssignals $u_{out}(t)$

Jede Oberschwingung $u_{in,k}$ wird mit dem Betrag der Übertragungsfunktion multipliziert, weiterhin wird der Winkel im Argument addiert. Es ergibt sich:

$$u_{out,k} = b_k \cdot |H(j\omega)| \cdot \cos(k\omega_0 t + arg(H(j\omega)))$$

u_{out} ergibt sich aus der Summe der Schwingungen $u_{out,k}$:

$$u_{out} = \sum_{k=1}^{\infty} u_{out,k}$$

Dies wurde in der Funktion `u_out()` (Kapitel 6.3, ab Z. 78) implementiert und in Abbildung 4 geplottet.

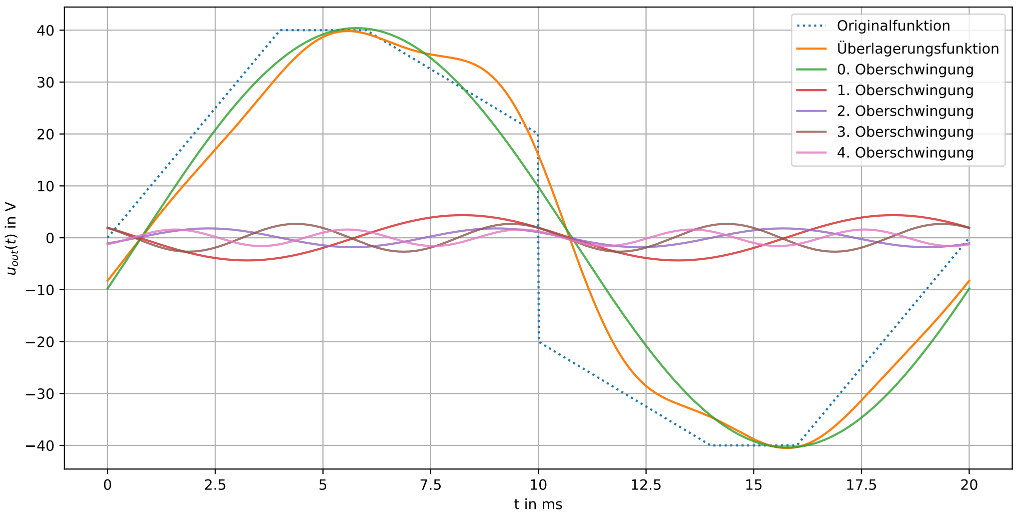

Abbildung 4: Plot Ausgangsspannung $u_{out}(t)$ und dessen Oberschwingungen

Betrag und Phasenwinkel wurden in Funktion `u_out()` (Kapitel 6.3, Z. 89) als print ausgegeben und anschließend in Abbildung 5 aufgelistet. Es fällt auf, dass der Betrag der Wellen mit steigendem ω abnimmt. Deshalb handelt es sich auch wirklich um einen Tiefpass-Filter. Weiterhin fällt der Phasenwinkel von $-45°$ bei der 3. Oberschwingung auf. Da diese Oberschwingung eine Verstärkung von $-3\,\text{dB}$ haben sollte, dient diese als Grenzfrequenz. Die Grenzfrequenz hat die Eigenschaft, einen Phasenwinkel von $45°$ zu besitzen. Dadurch bestätigen sich die Berechnungen.

```
0. Oberschwingung: |H(jw)|: 0.9701,      arg(H(jw)): -14.0362
1. Oberschwingung: |H(jw)|: 0.8944,      arg(H(jw)): -26.5651
2. Oberschwingung: |H(jw)|: 0.8,         arg(H(jw)): -36.8699
3. Oberschwingung: |H(jw)|: 0.7071,      arg(H(jw)): -45.0
4. Oberschwingung: |H(jw)|: 0.6247,      arg(H(jw)): -51.3402
```

Abbildung 5: Betrag und Phasenwinkel der ersten 5 Oberschwingungen

6 Python-Skripte

6.1 plot_1.py

```


# importieren von den ben tigten Bibliotheken

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker

# bekannte Werte

U_0 = 40
R = 2 * np.pi
L = (2 * np.pi) / (400 * np.pi)
leaps = [0.004, 0.006, 0.01, 0.014, 0.016]
T = 0.02
omega_0 = (2*np.pi)/(T)

# Definition des Eingangssignals

def u_in(value, leap, U_0):

    u_ins = np.array([])

    for x in value:

        if(x<leap[0]):
            u_ins = np.append(u_ins, (10000 * x));

        elif((x>leap[0]) and (x<leap[1])):
            u_ins = np.append(u_ins, (U_0))

        elif((x>leap[1]) and (x<leap[2])):
            u_ins = np.append(u_ins, (-5000 * x + 70))

        elif((x>leap[2]) and (x<leap[3])):
            u_ins = np.append(u_ins, (-5000 * x + 30))

        elif((x>leap[3]) and (x<leap[4])):
```

```
            u_ins = np.append(u_ins, (-U_0))

        elif((x>leap[4])):
            u_ins = np.append(u_ins, (10000 * x - 200))

    return u_ins


# Berechnung der b_ks

def b_k(u_in, x, omega_0, a, b, k_max, n=1000):
    b_k = np.array([])

    for k in range(1, k_max+1):
        b_k = np.append(b_k, (2/b)*np.trapz(u_in*np.sin(k*omega_0*x), x,
    (b-a)/n))

    return b_k

# Definieren der Synthesegleichung

def synth_u_in(b_k, x, omega_0):
    u_in_synth = np.zeros(len(x))
    u_in_sin = []

    for k, b in enumerate(b_k):
        u_in_synth += b * np.sin((k+1)*omega_0*x)
        u_in_sin.append(np.sin((k+1)*omega_0*x))

    return u_in_synth, u_in_sin

# Definieren der  bertragungsfunktion

def H(omega, R, L):
    z = (1)/(1 + (1j*omega*L) / (R))
    return abs(z), np.angle(z)

# Definieren der Ausgangsignals

def u_out(b_k, R, L, omega_0, x):

    u_out = []

    for k, bk in enumerate(b_k):

        abs_H, ang_H = H(omega_0*(k+1), R, L)

        u_out.append(bk * abs_H * np.sin((k+1)*x*omega_0 + ang_H))

    return u_out


######### 1. Plot #########
x = np.linspace(0,0.02,1000)
```

```python
y = u_in(x, leaps, U_0);

b_k_5 = b_k(y, x, omega_0, 0, T, 5)
u_in_synth_5, u_in_sin_5 = synth_u_in(b_k_5, x, omega_0)

fig, ax = plt.subplots(figsize=(12,6), dpi=300)
ax.plot(x, y, label="Originalfunktion", linestyle="dotted")

for i, sin in enumerate(u_in_sin_5):
    ax.plot(x, b_k_5[i] * sin, alpha=0.8, label = f"{i}. Oberschwingung")

scale_x = 1e-3
ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_x))
ax.xaxis.set_major_formatter(ticks_x)

ax.set(xlabel='t in ms', ylabel='$u_{in}(t)$ in V')

ax.grid()
plt.legend()
#plt.savefig("plot_1.pdf")
plt.show()
```

6.2 plot_2.py

```python

# importieren von den ben tigten Bibliotheken

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker

# bekannte Werte

U_0 = 40
R = 2 * np.pi
L = (2 * np.pi) / (400 * np.pi)
leaps = [0.004, 0.006, 0.01, 0.014, 0.016]
T = 0.02
omega_0 = (2*np.pi)/(T)

# Definition des Eingangssignals

def u_in(value, leap, U_0):

    u_ins = np.array([])

    for x in value:

        if(x<leap[0]):
            u_ins = np.append(u_ins, (10000 * x));

        elif((x>leap[0]) and (x<leap[1])):
            u_ins = np.append(u_ins, (U_0))
```

```
        elif((x>leap[1]) and (x<leap[2])):
            u_ins = np.append(u_ins, (-5000 * x + 70))

        elif((x>leap[2]) and (x<leap[3])):
            u_ins = np.append(u_ins, (-5000 * x + 30))

        elif((x>leap[3]) and (x<leap[4])):
            u_ins = np.append(u_ins, (-U_0))

        elif((x>leap[4])):
            u_ins = np.append(u_ins, (10000 * x - 200))

    return u_ins


# Berechnung der b_ks

def b_k(u_in, x, omega_0, a, b, k_max, n=1000):
    b_k = np.array([])

    for k in range(1, k_max+1):
        b_k = np.append(b_k, (2/b)*np.trapz(u_in*np.sin(k*omega_0*x), x,
    (b-a)/n))

    return b_k

# Definieren der Synthesegleichung

def synth_u_in(b_k, x, omega_0):
    u_in_synth = np.zeros(len(x))
    u_in_sin = []

    for k, b in enumerate(b_k):
        u_in_synth += b * np.sin((k+1)*omega_0*x)
        u_in_sin.append(np.sin((k+1)*omega_0*x))

    return u_in_synth, u_in_sin

# Definieren der  bertragungsfunktion

def H(omega, R, L):
    z = (1)/(1 + (1j*omega*L) / (R))
    return abs(z), np.angle(z)

# Definieren der Ausgangsignals

def u_out(b_k, R, L, omega_0, x):

    u_out = []

    for k, bk in enumerate(b_k):

        abs_H, ang_H = H(omega_0*(k+1), R, L)
```

```python

        u_out.append(bk * abs_H * np.sin((k+1)*x*omega_0 + ang_H))

    return u_out


######### 2. Plot #########

x = np.linspace(0,0.02,1000)
y = u_in(x, leaps, U_0);

b_k_250 = b_k(y, x, omega_0, 0, T, 250)
u_in_synth_250, u_in_sin_250 = synth_u_in(b_k_250, x, omega_0)


fig, ax = plt.subplots(figsize=(12,6), dpi=300)
ax.plot(x, y, label="Originalfunktion", linestyle="dotted")

ax.plot(x, u_in_synth_250, label = "Ann herungsfunktion", alpha=0.5)

# for i, sin in enumerate(u_in_sin_5):

#     ax.plot(x, b_k_5[i] * sin, alpha=0.8, label = f"{i}. Oberschwingung")

scale_x = 1e-3
ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_x))
ax.xaxis.set_major_formatter(ticks_x)

ax.set(xlabel='t in ms', ylabel='$u_{in}(t)$ in V')

ax.grid()
plt.legend()
#plt.savefig("plot_2.pdf")
plt.show()
```

6.3 plot_3.py

```python

# importieren von den ben tigten Bibliotheken

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker

# bekannte Werte

U_0 = 40
R = 2 * np.pi
L = (2 * np.pi) / (400 * np.pi)
leaps = [0.004, 0.006, 0.01, 0.014, 0.016]
T = 0.02
omega_0 = (2*np.pi)/(T)
```

```python

# Definition des Eingangssignals

def u_in(value, leap, U_0):

    u_ins = np.array([])

    for x in value:

        if(x<leap[0]):
            u_ins = np.append(u_ins, (10000 * x));

        elif((x>leap[0]) and (x<leap[1])):
            u_ins = np.append(u_ins, (U_0))

        elif((x>leap[1]) and (x<leap[2])):
            u_ins = np.append(u_ins, (-5000 * x + 70))

        elif((x>leap[2]) and (x<leap[3])):
            u_ins = np.append(u_ins, (-5000 * x + 30))

        elif((x>leap[3]) and (x<leap[4])):
            u_ins = np.append(u_ins, (-U_0))

        elif((x>leap[4])):
            u_ins = np.append(u_ins, (10000 * x - 200))

    return u_ins


# Berechnung der b_ks

def b_k(u_in, x, omega_0, a, b, k_max, n=1000):
    b_k = np.array([])

    for k in range(1, k_max+1):
        b_k = np.append(b_k, (2/b)*np.trapz(u_in*np.sin(k*omega_0*x), x,
    (b-a)/n))

    return b_k


# Definieren der Synthesegleichung

def synth_u_in(b_k, x, omega_0):
    u_in_synth = np.zeros(len(x))
    u_in_sin = []

    for k, b in enumerate(b_k):
        u_in_synth += b * np.sin((k+1)*omega_0*x)
        u_in_sin.append(np.sin((k+1)*omega_0*x))

    return u_in_synth, u_in_sin

```

```
# Definieren der    bertragungsfunktion

def H(omega, R, L):
    z = (1)/(1 + (1j*omega*L) / (R))
    return abs(z), np.angle(z)

# Definieren der Ausgangsignals

def u_out(b_k, R, L, omega_0, x):

    u_out = []

    for k, bk in enumerate(b_k):

        abs_H, ang_H = H(omega_0*(k+1), R, L)


        # Ausgabe des Betrags + Phase der    bertragungsfunktion

        #print(f"{k}. Oberschwingung: |H(jw)|: {round(abs_H, 4)}, \t arg(H(jw)): {round(ang_H*360/(2*np.pi), 4)}  ")

        u_out.append(bk * abs_H * np.sin((k+1)*x*omega_0 + ang_H))

    return u_out

######### 3. Plot #########

x = np.linspace(0,0.02,1000)
y = u_in(x, leaps, U_0);

b_k_5 = b_k(y, x, omega_0, 0, T, 5)
u_in_synth_5, u_in_sin_5 = synth_u_in(b_k_5, x, omega_0)


u_out = u_out(b_k_5, R, L, omega_0, x)


fig, ax = plt.subplots(figsize=(12,6), dpi=300)
ax.plot(x, y, label="Originalfunktion", linestyle="dotted")


ax.plot(x, np.sum(u_out, axis=0), label="   berlagerungsfunktion   ")

for k, u in enumerate(u_out):
    ax.plot(x, u, label=f"{k}. Oberschwingung", alpha=0.8)

scale_x = 1e-3
ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_x))
ax.xaxis.set_major_formatter(ticks_x)

ax.set(xlabel='t in ms', ylabel='$u_{out}(t)$ in V')

ax.grid()
```

```
plt.legend()
#plt.savefig("plot_3.pdf")
plt.show()
```