

Elektrische Netzwerke und Mehrport 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:

```
1  b_1 = 41.64106231
2  b_2 = -4.87735577
3  b_3 = 2.25902828
4  b_4 = -3.78545704
5  b_5 = 2.54658425
```

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

2 Plot der ersten 5 Schwingungen

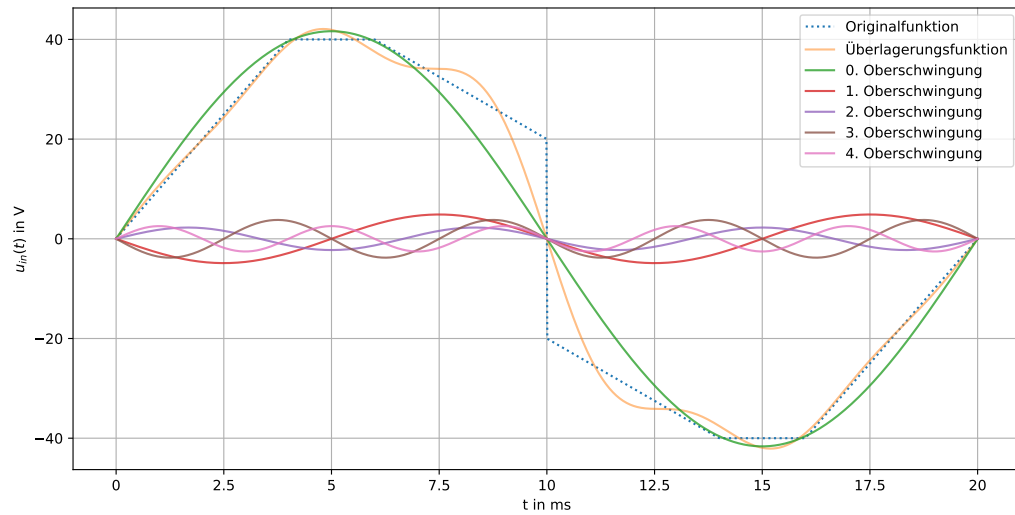


Abbildung 2: Plot der Grundschiwingung und der ersten 4 Oberschwingungen, sowie deren Überlagerung

3 Plot der ersten 250 Wellen

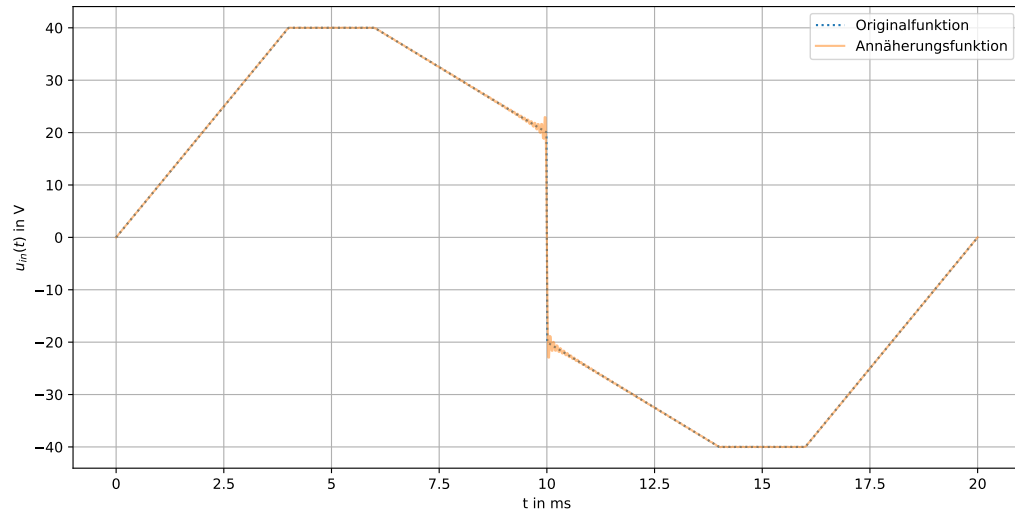


Abbildung 3: Plot der Fourierreihe mit den ersten 250 Wellen

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 $u_{in}(t = \frac{T}{2}) = 0$.

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 dB verstärkt werden. Das ist der Fall, wenn $\omega_3 \stackrel{!}{=} \Omega$ gilt.

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

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

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

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

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.

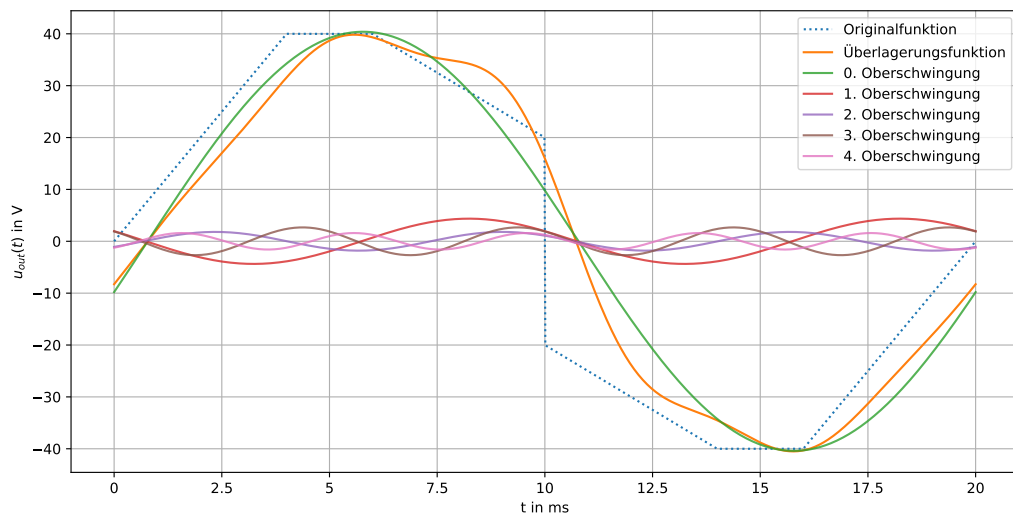


Abbildung 4: Plot Ausgangsspannung $u_{out}(t)$ und dessen Oberschwingungen

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 -3dB haben sollte, dient diese als Grenzfrequenz. Die Grenzfrequenz hat die Eigenschaft, einen Phasenwinkel von 45° zu besitzen. Dadurch bestätigen sich die Berechnungen.

1	0. Oberschwingung:	$ H(j\omega) $:	0.9701,	$\arg(H(j\omega))$:	-14.0362
2	1. Oberschwingung:	$ H(j\omega) $:	0.8944,	$\arg(H(j\omega))$:	-26.5651
3	2. Oberschwingung:	$ H(j\omega) $:	0.8,	$\arg(H(j\omega))$:	-36.8699
4	3. Oberschwingung:	$ H(j\omega) $:	0.7071,	$\arg(H(j\omega))$:	-45.0
5	4. Oberschwingung:	$ H(j\omega) $:	0.6247,	$\arg(H(j\omega))$:	-51.3402

Abbildung 5: Betrag und Phasenwinkel der ersten 5 Oberschwingungen

6 Python-Skripte

6.1 plot_1.py

```

1
2
3 # importieren von den ben tigten Bibliotheken
4
5 import matplotlib
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import matplotlib.ticker as ticker
9
10 # bekannte Werte
11
12 U_0 = 40
13 R = 2 * np.pi
14 L = (2 * np.pi) / (400 * np.pi)
15 leaps = [0.004, 0.006, 0.01, 0.014, 0.016]
16 T = 0.02
17 omega_0 = (2*np.pi)/(T)
18
19 # Definition des Eingangssignals
20
21 def u_in(value, leap, U_0):
22
23     u_ins = np.array([])
24
25     for x in value:
26
27         if(x<leap[0]):
28             u_ins = np.append(u_ins, (10000 * x));
29
30         elif((x>leap[0]) and (x<leap[1])):
31             u_ins = np.append(u_ins, (U_0))
32
33         elif((x>leap[1]) and (x<leap[2])):
34             u_ins = np.append(u_ins, (-5000 * x + 70))
35
36         elif((x>leap[2]) and (x<leap[3])):
37             u_ins = np.append(u_ins, (-5000 * x + 30))
38
39         elif((x>leap[3]) and (x<leap[4])):

```



```

40         u_ins = np.append(u_ins, (-U_0))
41
42         elif((x>leap[4])):
43             u_ins = np.append(u_ins, (10000 * x - 200))
44
45     return u_ins
46
47
48 # Berechnung der b_ks
49
50 def b_k(u_in, x, omega_0, a, b, k_max, n=1000):
51     b_k = np.array([])
52
53     for k in range(1, k_max+1):
54         b_k = np.append(b_k, (2/b)*np.trapz(u_in*np.sin(k*omega_0*x), x,
55         (b-a)/n))
56
57     return b_k
58
59 # Definieren der Synthesegleichung
60
61 def synth_u_in(b_k, x, omega_0):
62     u_in_synth = np.zeros(len(x))
63     u_in_sin = []
64
65     for k, b in enumerate(b_k):
66         u_in_synth += b * np.sin((k+1)*omega_0*x)
67         u_in_sin.append(np.sin((k+1)*omega_0*x))
68
69     return u_in_synth, u_in_sin
70
71 # Definieren der   bertragungsfunktion
72
73 def H(omega, R, L):
74     z = (1)/(1 + (1j*omega*L) / (R))
75     return abs(z), np.angle(z)
76
77 # Definieren der Ausgangsignals
78
79 def u_out(b_k, R, L, omega_0, x):
80     u_out = []
81
82     for k, bk in enumerate(b_k):
83
84         abs_H, ang_H = H(omega_0*(k+1), R, L)
85
86         u_out.append(bk * abs_H * np.sin((k+1)*x*omega_0 + ang_H))
87
88     return u_out
89
90
91 ##### 1. Plot #####
92 x = np.linspace(0,0.02,1000)

```

```

93 y = u_in(x, leaps, U_0);
94
95 b_k_5 = b_k(y, x, omega_0, 0, T, 5)
96 u_in_synth_5, u_in_sin_5 = synth_u_in(b_k_5, x, omega_0)
97
98 fig, ax = plt.subplots(figsize=(12,6), dpi=300)
99 ax.plot(x, y, label="Originalfunktion", linestyle="dotted")
100
101 for i, sin in enumerate(u_in_sin_5):
102     ax.plot(x, b_k_5[i] * sin, alpha=0.8, label = f"{i}. Oberschwingung")
103
104 scale_x = 1e-3
105 ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_x))
106 ax.xaxis.set_major_formatter(ticks_x)
107
108 ax.set(xlabel='t in ms', ylabel='$u_{in}(t)$ in V')
109
110 ax.grid()
111 plt.legend()
112 #plt.savefig("plot_1.pdf")
113 plt.show()

```

6.2 plot_2.py

```

1
2 # importieren von den ben tigten Bibliotheken
3
4 import matplotlib
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import matplotlib.ticker as ticker
8
9 # bekannte Werte
10
11 U_0 = 40
12 R = 2 * np.pi
13 L = (2 * np.pi) / (400 * np.pi)
14 leaps = [0.004, 0.006, 0.01, 0.014, 0.016]
15 T = 0.02
16 omega_0 = (2*np.pi)/(T)
17
18 # Definition des Eingangssignals
19
20 def u_in(value, leap, U_0):
21
22     u_ins = np.array([])
23
24     for x in value:
25
26         if(x<leap[0]):
27             u_ins = np.append(u_ins, (10000 * x));
28
29         elif((x>leap[0]) and (x<leap[1])):
30             u_ins = np.append(u_ins, (U_0))

```

```

31
32     elif((x>leap[1]) and (x<leap[2])):
33         u_ins = np.append(u_ins, (-5000 * x + 70))
34
35     elif((x>leap[2]) and (x<leap[3])):
36         u_ins = np.append(u_ins, (-5000 * x + 30))
37
38     elif((x>leap[3]) and (x<leap[4])):
39         u_ins = np.append(u_ins, (-U_0))
40
41     elif((x>leap[4])):
42         u_ins = np.append(u_ins, (10000 * x - 200))
43
44     return u_ins
45
46
47 # Berechnung der b_ks
48
49 def b_k(u_in, x, omega_0, a, b, k_max, n=1000):
50     b_k = np.array([])
51
52     for k in range(1, k_max+1):
53         b_k = np.append(b_k, (2/b)*np.trapz(u_in*np.sin(k*omega_0*x), x,
54         (b-a)/n))
55
56     return b_k
57
58 # Definieren der Synthesgleichung
59
60 def synth_u_in(b_k, x, omega_0):
61     u_in_synth = np.zeros(len(x))
62     u_in_sin = []
63
64     for k, b in enumerate(b_k):
65         u_in_synth += b * np.sin((k+1)*omega_0*x)
66         u_in_sin.append(np.sin((k+1)*omega_0*x))
67
68     return u_in_synth, u_in_sin
69
70 # Definieren der   bertragungsfunktion
71
72 def H(omega, R, L):
73     z = (1)/(1 + (1j*omega*L) / (R))
74     return abs(z), np.angle(z)
75
76 # Definieren der Ausgangsignals
77
78 def u_out(b_k, R, L, omega_0, x):
79     u_out = []
80
81     for k, bk in enumerate(b_k):
82
83         abs_H, ang_H = H(omega_0*(k+1), R, L)

```

```

84         u_out.append(bk * abs_H * np.sin((k+1)*x*omega_0 + ang_H))
85
86
87     return u_out
88
89
90 ##### 2. Plot #####
91
92 x = np.linspace(0,0.02,1000)
93 y = u_in(x, leaps, U_0);
94
95 b_k_250 = b_k(y, x, omega_0, 0, T, 250)
96 u_in_synth_250, u_in_sin_250 = synth_u_in(b_k_250, x, omega_0)
97
98
99 fig, ax = plt.subplots(figsize=(12,6), dpi=300)
100 ax.plot(x, y, label="Originalfunktion", linestyle="dotted")
101
102 ax.plot(x, u_in_synth_250, label = "Annherungsfunktion", alpha=0.5)
103
104 # for i, sin in enumerate(u_in_sin_5):
105
106 #     ax.plot(x, b_k_5[i] * sin, alpha=0.8, label = f"{i}. Oberschwingung")
107
108 scale_x = 1e-3
109 ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_x))
110 ax.xaxis.set_major_formatter(ticks_x)
111
112 ax.set(xlabel='t in ms', ylabel='$u_{in}(t)$ in V')
113
114 ax.grid()
115 plt.legend()
116 #plt.savefig("plot_2.pdf")
117 plt.show()

```

6.3 plot_3.py

```

1
2 # importieren von den benutzten Bibliotheken
3
4 import matplotlib
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import matplotlib.ticker as ticker
8
9 # bekannte Werte
10
11 U_0 = 40
12 R = 2 * np.pi
13 L = (2 * np.pi) / (400 * np.pi)
14 leaps = [0.004, 0.006, 0.01, 0.014, 0.016]
15 T = 0.02
16 omega_0 = (2*np.pi)/(T)

```

```

17
18 # Definition des Eingangssignals
19
20 def u_in(value, leap, U_0):
21
22     u_ins = np.array([])
23
24     for x in value:
25
26         if(x<leap[0]):
27             u_ins = np.append(u_ins, (10000 * x));
28
29         elif((x>leap[0]) and (x<leap[1])):
30             u_ins = np.append(u_ins, (U_0))
31
32         elif((x>leap[1]) and (x<leap[2])):
33             u_ins = np.append(u_ins, (-5000 * x + 70))
34
35         elif((x>leap[2]) and (x<leap[3])):
36             u_ins = np.append(u_ins, (-5000 * x + 30))
37
38         elif((x>leap[3]) and (x<leap[4])):
39             u_ins = np.append(u_ins, (-U_0))
40
41         elif((x>leap[4])):
42             u_ins = np.append(u_ins, (10000 * x - 200))
43
44     return u_ins
45
46
47 # Berechnung der b_ks
48
49 def b_k(u_in, x, omega_0, a, b, k_max, n=1000):
50     b_k = np.array([])
51
52     for k in range(1, k_max+1):
53         b_k = np.append(b_k, (2/b)*np.trapz(u_in*np.sin(k*omega_0*x), x,
54         (b-a)/n))
55
56     return b_k
57
58 # Definieren der Synthesgleichung
59
60 def synth_u_in(b_k, x, omega_0):
61     u_in_synth = np.zeros(len(x))
62     u_in_sin = []
63
64     for k, b in enumerate(b_k):
65         u_in_synth += b * np.sin((k+1)*omega_0*x)
66         u_in_sin.append(np.sin((k+1)*omega_0*x))
67
68     return u_in_synth, u_in_sin
69

```

```

70 # Definieren der   bertragungsfunktion
71
72 def H(omega, R, L):
73     z = (1)/(1 + (1j*omega*L) / (R))
74     return abs(z), np.angle(z)
75
76 # Definieren der Ausgangsignals
77
78 def u_out(b_k, R, L, omega_0, x):
79
80     u_out = []
81
82     for k, bk in enumerate(b_k):
83
84         abs_H, ang_H = H(omega_0*(k+1), R, L)
85
86
87         # Ausgabe des Betrags + Phase der   bertragungsfunktion
88
89         #print(f"{k}. Oberschwingung: |H(jw)|: {round(abs_H, 4)}, \t arg(
H(jw)): {round(ang_H*360/(2*np.pi), 4)}  ")
90
91         u_out.append(bk * abs_H * np.sin((k+1)*x*omega_0 + ang_H))
92
93     return u_out
94
95 ##### 3. Plot #####
96
97 x = np.linspace(0,0.02,1000)
98 y = u_in(x, leaps, U_0);
99
100 b_k_5 = b_k(y, x, omega_0, 0, T, 5)
101 u_in_synth_5, u_in_sin_5 = synth_u_in(b_k_5, x, omega_0)
102
103
104 u_out = u_out(b_k_5, R, L, omega_0, x)
105
106
107 fig, ax = plt.subplots(figsize=(12,6), dpi=300)
108 ax.plot(x, y, label="Originalfunktion", linestyle="dotted")
109
110
111 ax.plot(x, np.sum(u_out, axis=0), label="   berlagerungsfunktion   ")
112
113 for k, u in enumerate(u_out):
114     ax.plot(x, u, label=f"{k}. Oberschwingung", alpha=0.8)
115
116 scale_x = 1e-3
117 ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_x))
118 ax.xaxis.set_major_formatter(ticks_x)
119
120 ax.set(xlabel='t in ms', ylabel='$u_{out}(t)$ in V')
121
122 ax.grid()

```

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