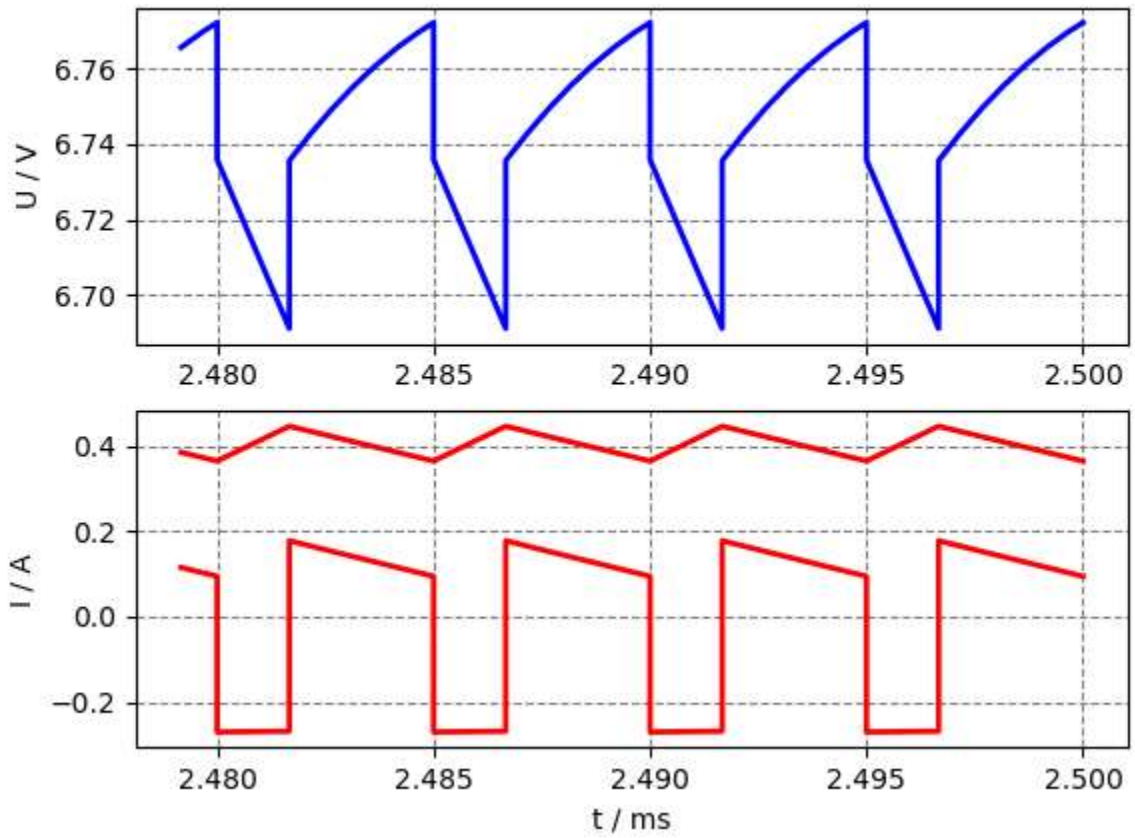
Simulation eines Aufwärtswandlers.

```
"""Das Programm simuliert n = 1 ... N_SIM Schaltzyklen eines Aufwärtswandlers,
welcher in jedem Schaltzyklus zwei Zustände durchläuft:

- Zustand 0, von (n * T) bis (n * T + DUTY * T): MOSFET leitet
- Zustand 1, (n * T + DUTY * T) bis (n * T + T): MOSFET sperrt

Im oberen Teilfenster wird die Ausgangsspannung des Wandlers dargestellt,
im unteren Teilfenster finden sich Spulen- und Kondensatorströme.

Es wird vorausgesetzt, dass kein lückender Betrieb vorliegt. Dazu müsste
ein weiterer Zustand zur Simulation hinzugefügt werden (der MOSFET sperrt,
es fließt aber kein Spulenstrom mehr). Aus demselben Grund werden auch die
Startwerte für Spulenstrom (IL_S) und Kondensatorspannung (UC_S) zu Beginn
der Simulation nicht auf null gesetzt: Die Einschwingvorgänge nach einem
"Kaltstart" des Wandlers würden in der Simulation zu negativen Spulenströmen
führen, die in der Realität aufgrund der Diode so nicht möglich wären.

Für weitere Details zur Schaltungssimulation mit Python siehe:
https://kuepper.userweb.mwn.de/elektronik/formelsammlung-elektronik.pdf

Tilman Küpper, 2022-05-26
"""

from scipy.integrate import solve_ivp
from numpy import concatenate
import matplotlib.pyplot as plt

T = 1 / 200e+3      # Periodendauer (in Sekunden)
DUTY = 0.333333     # Tastgrad (0...1)
T_ON = T * DUTY     # MOSFET-Einschaltdauer (in Sekunden)
RA = 25.0           # Lastwiderstand (in Ohm)
UE = 5.0            # Eingangsspannung (in Volt)
C = 10e-6           # Kapazität (in F)
L = 100e-6          # Induktivität (in H)
RL = 0.2            # Widerstand der Spule (in Ohm)
RC = 0.1            # Widerstand des Kondensators (in Ohm)
US = 0.4            # Schwellenspannung der Diode (in V)
RF = 0.5            # diff. Widerstand der Diode (in Ohm)
RDSON = 0.1         # Widerstand des eingeschalteten MOSFETs (in Ohm)
IL_S = 0.5          # Startwert Spulenstrom (in A)
UC_S = 6.5          # Startwert Kondensatorspannung (in V)
N_SIM = 500         # Anzahl der simulierten Schaltzyklen
POS = -55           # Nur die letzten POS Stützpunkte werden geplottet


def ua_z0(_il, uc):
    """Berechne Ausgangsspannung in Zustand 0 (MOSFET leitet)."""
    return RA / (RA + RC) * uc


def ua_z1(il, uc):
    """Berechne Ausgangsspannung in Zustand 1 (MOSFET sperrt)."""
    return RA / (RA + RC) * (RC * il + uc)


def ic_z0(il, uc):
    """Berechne Kondensatorstrom in Zustand 0 (MOSFET leitet)."""
    return -ua_z0(il, uc) / RA


def ic_z1(il, uc):
    """Berechne Kondensatorstrom in Zustand 1 (MOSFET sperrt)."""
    return il - ua_z0(il, uc) / RA
```

```
def dgl_z0(_t, y):
    """Differentialgleichungssystem in Zustand 0 (MOSFET leitet)."""
    il, uc = y[0], y[1]  # Zustandsvektor y: zwei Zustandsvariablen
    ddt_il = (UE - il * (RL + RDSON)) / L
    ddt_uc = -uc / (C * (RA + RC))
    return [ddt_il, ddt_uc]


def dgl_z1(_t, y):
    """Differentialgleichungssystem in Zustand 1 (MOSFET sperrt)."""
    il, uc = y[0], y[1]  # Zustandsvektor y: zwei Zustandsvariablen
    ddt_il = (UE - US - ua_z1(il, uc) - il * (RL + RF)) / L
    ddt_uc = (ua_z1(il, uc) - uc) / (C * RC)
    return [ddt_il, ddt_uc]


def main():
    """Simulation durchführen, Ergebnisse plotten."""
    il, uc = IL_S, UC_S  # Startwerte Spulenstrom/Kondensatorspannung
    t_, il_, ic_, ua_ = [], [], [], []  # Listen für Ergebnisse

    for i in range(N_SIM):
        # Neuer Schaltzyklus beginnt, MOSFET ist eingeschaltet
        t = i * T
        sol = solve_ivp(dgl_z0, (t, t + T_ON), [il, uc], max_step=T/10)
        sol_il, sol_uc = sol.y[0], sol.y[1]
        t_.append(sol.t * 1000)  # Für die Ausgabe: Zeit in ms umrechnen
        il_.append(sol_il)       # Ergebnisse in Listen speichern
        ic_.append(ic_z0(sol_il, sol_uc))
        ua_.append(ua_z0(sol_il, sol_uc))

        # Endwerte Spulenstrom/Kondensatorspg = Startwerte nächster Schritt
        il, uc = sol_il[-1], sol_uc[-1]

        # Ab jetzt sperrt der MOSFET
        sol = solve_ivp(dgl_z1, (t + T_ON, t + T), [il, uc], max_step=T/10)
        sol_il, sol_uc = sol.y[0], sol.y[1]
        t_.append(sol.t * 1000)  # Für die Ausgabe: Zeit in ms umrechnen
        il_.append(sol_il)       # Ergebnisse in Listen speichern
        ic_.append(ic_z1(sol_il, sol_uc))
        ua_.append(ua_z1(sol_il, sol_uc))

        # Endwerte Spulenstrom/Kondensatorspg = Startwerte nächster Schritt
        il, uc = sol_il[-1], sol_uc[-1]

    # Ausgangsspannung (oben), Spulen- und Kondensatorstrom (unten) plotten
    _fig, (a1, a2) = plt.subplots(2, 1)
    a1.plot(concatenate(t_)[POS:], concatenate(ua_)[POS:], "b-", linewidth=2)
    a2.plot(concatenate(t_)[POS:], concatenate(il_)[POS:], "r-", linewidth=2)
    a2.plot(concatenate(t_)[POS:], concatenate(ic_)[POS:], "r-", linewidth=2)
    a1.grid(True, color='gray', linestyle='dashed')
    a2.grid(True, color='gray', linestyle='dashed')
    a1.set_ylabel('U / V')
    a2.set_ylabel('I / A')
    a2.set_xlabel('t / ms')
    plt.show()


# Simulation durchführen, Ergebnisse plotten.
if __name__ == "__main__":
    main()
```


6.76
6.74
6.72
6.70
U / V
2.480
2.485
2.490
2.495
2.500
0.4
0.2
0.0
−0.2
I / A
2.480
2.485
2.490
2.495
2.500
t / ms