Universidade Federal de Santa Catarina

Programa de Pós-Graduação em Engenharia Elétrica

Modelagem Dinâmica de Conversores Estáticos

Resolução da Tarefa 4

Data: 11/06/2022

Discente: Eduardo Eller Behr

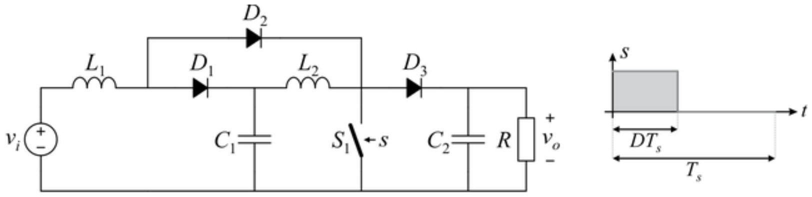
Tarefa 4

Para o conversor boost quadrático ilustrado abaixo, utilizando a metodologia por espaço de estados, pede-se:

I) A função de transferência que relaciona a tensão de saída com a razão cíclica;

II) A função de transferência que relaciona a tensão de saída com a razão cíclica considerando perdas nos indutores, capacitores, diodos e interruptores.

Valide os modelos por simulações.




```
In [ ]:
from matplotlib import pyplot as plt
import control as ctl
import numpy as np
import sympy as sp
```

I) Função de Transferência $\frac{v_o(s)}{d(s)}$ (**sem** perdas)

```
In [ ]:
iL1, iL2, vC1, vC2 = sp.symbols("i_{L1} i_{L2} v_{C1} v_{C2}")
d, vi, s = sp.symbols("d v_i s")
R, L1, L2, C1, C2 = sp.symbols("R L_1 L_2 C_1 C_2")

U = sp.Matrix([
    [vi]
])

# Considerando que vo = 1*vC2
C = sp.Matrix([[0, 0, 0, 1]])

# display('X=', X);
display('U=', U); display('C=', C)
```

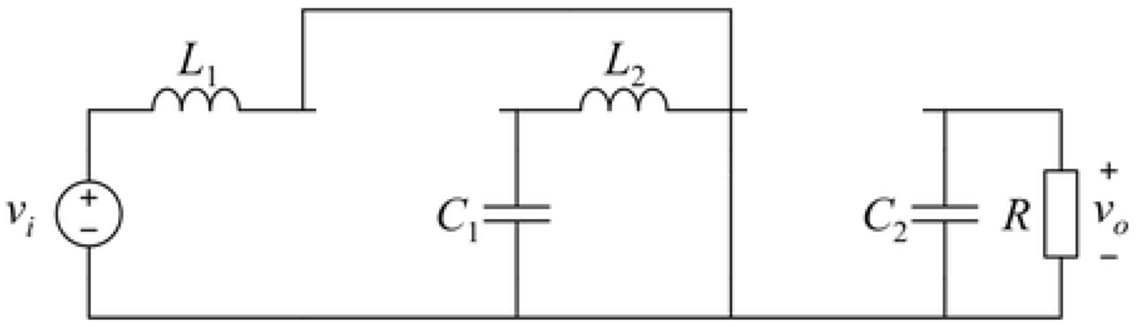
'U='

$$\begin{bmatrix} v_i \end{bmatrix}$$

'C='

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

a) 1ª etapa de operação



In [ ]:
```
A1 = sp.Matrix([
        [0,              0,               0,              0
        [0,              0,               1/L2,   0                          ],
        [0,              -1/C1,   0,              0                          ],
        [0,              0,               0,              -1/(R*C2)       ],
#    L1          L2               C1              C2
])

B1 = sp.Matrix([
        [1/L1    ], # iL1
        [0              ],     # iL2
        [0              ],     # vC1
        [0              ],     # vC2
])

display('A1=', A1)
display('B1=', B1)
```

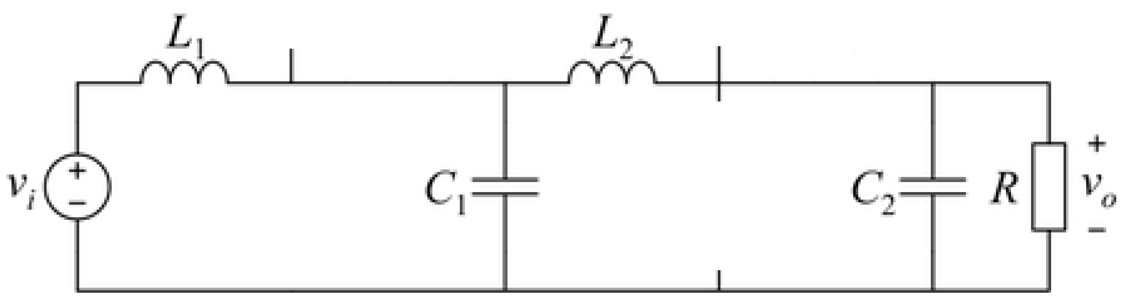
'A1='

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{L_2} & 0 \\ 0 & -\frac{1}{C_1} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{C_2 R} \end{bmatrix}$$

'B1='

$$\begin{bmatrix} \frac{1}{L_1} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

b) 2ª Etapa de operação



In [ ]:
```
A2 = sp.Matrix([
        [0,             0,              -1/L1,  0               ],
        [0,             0,              1/L2,   -1/L2           ], # iL2
        [1/C1,   -1/C1, 0,              0                       ], # vC1
        [0,             1/C2,   0,              -1/(R*C2)       ], # vC2
#    L1         L2              C1              C2
])

B2 = sp.Matrix([
        [1/L1   ],      # iL1
        [0              ],      # iL2
        [0              ],      # vC1
        [0              ],      # vC2
])

display('A2=', A2)
display('B2=', B2)
```

'A2='

$$\begin{bmatrix} 0 & 0 & -\frac{1}{L_1} & 0 \\ 0 & 0 & \frac{1}{L_2} & -\frac{1}{L_2} \\ \frac{1}{C_1} & -\frac{1}{C_1} & 0 & 0 \\ 0 & \frac{1}{C_2} & 0 & -\frac{1}{C_2 R} \end{bmatrix}$$

'B2='

$$\begin{bmatrix} \frac{1}{L_1} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

c) Média das duas etapas

O ponto de operação em regime permanente é dado por:

$$\boldsymbol{X} = -\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{U}$$

$$\boldsymbol{Y} = (-\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B} + \boldsymbol{E})\boldsymbol{U}$$

A função de transferencia é obtida com:

$$\boldsymbol{G}(s) = \frac{\hat{\boldsymbol{y}}(s)}{\hat{d}(s)} = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\left[(\boldsymbol{A}_1 - \boldsymbol{A}_2)\boldsymbol{X} + (\boldsymbol{B}_1 - \boldsymbol{B}_2)\boldsymbol{U}\right]$$

In [ ]:
```
A = d*A1 + (1-d)*A2
B = d*B1 + (1-d)*B2
X = -A**-1*B*U

G = C*((s*sp.eye(4)-A)**-1)*( (A1-A2)*X + (B1-B2)*U )
```

In [ ]:
```
display('Gvd=', G[0].factor(s))
```

'Gvd='

$$\frac{v_i\left(-d^7+7d^6-21d^5+35d^4-35d^3+21d^2-7d+1\right)\left(C_1L_1L_2s^3-2Rd^4+8Rd^3-\right.}{\begin{array}{l}\left(1-d\right)\left(d^2-2d+1\right)\left(d^3-3d^2+3d-1\right)\left(d^4-4d^3+6d^2-4d+1\right)\left(C_1C_2L_1L_2Rs\right. \\ -4Rd^3+6Rd^2-4Rd+R \\ +s^2\left(C_1L_1Rd^2-2C_1L_1Rd+C_1L_1R+C_2L_1R+C_2L_2Rd^2-2C_2L_2Rd+C_2L_2R\right) \\ +s\left(L_1+L_2d^2-2L_2d+L_2\right)\right)\end{array}}$$

$$\left.+s^2\left(-C_1L_1Rd^2+2C_1L_1Rd-C_1L_1R\right)+s\left(2L_1+L_2d^2-2L_2d+L_2\right)\right)$$

Substituindo os valores numéricos

In [ ]:

```
params = {
    L1: 1000e-6,
    L2: 100e-6,
    C1: 2e-6,
    C2: 20e-6,
    R:  50,
    d: 0.55,
    vi: 48
}

Gvd = G.subs(params)

Gvd = Gvd[0].factor(s).simplify()

display('Gvd=',Gvd.evalf(3)); display('X=', X.subs(params).evalf(6))
```

'Gvd='

$$\frac{-5.14\cdot10^{-11}s^3+5.2\cdot10^{-6}s^2-0.519s+1.05\cdot10^3}{9.76\cdot10^{-17}s^4+9.75\cdot10^{-14}s^3+5.07\cdot10^{-7}s^2+0.000498s+1.0}$$

'X='

$$\begin{bmatrix}23.4111\\10.535\\106.667\\237.037\end{bmatrix}$$

Convertendo para objeto TransferFunction

In [ ]:

```
def sympy_rational_to_control_tf(rational):
    numerator, denominator = rational.as_numer_denom()
    num, den = sp.Poly(numerator).all_coeffs(), sp.Poly(denominator).all_co

    for i, _n in enumerate(num):
        num[i] = float(num[i])

    for i, _d in enumerate(den):
        den[i] = float(den[i])

    return ctl.TransferFunction(num, den)

Gvd_tf = sympy_rational_to_control_tf(Gvd)
Gvd_tf
```

Out[ ]:

$$\frac{-5.138 \times 10^{-11}s^3 + 5.202 \times 10^{-6}s^2 - 0.519s + 1053}{9.755 \times 10^{-17}s^4 + 9.755 \times 10^{-14}s^3 + 5.075 \times 10^{-7}s^2 + 0.0004976s + 1}$$

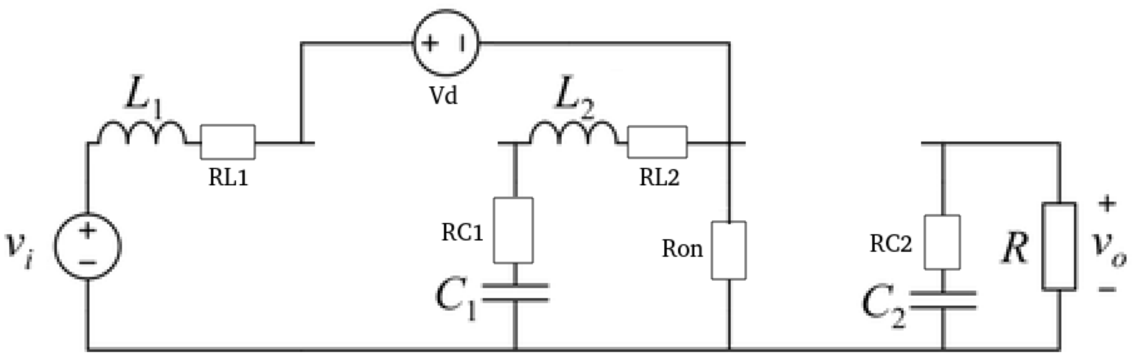
II) Função de Transferência $\frac{v_o(s)}{d(s)}$ (**com** perdas)

In [ ]:
```
Vd, RL1, RL2, Ron, RC1, RC2 = sp.symbols('V_d R_{L1} R_{L2} R_{on} R_{C1} R

Up = sp.Matrix([
    vi,
    Vd
])

display('Up=', Up)
```

'Up='

$$\begin{bmatrix} v_i \\ V_d \end{bmatrix}$$

a) 1ª Etapa de operação



In [ ]:
```
A1p = sp.Matrix([
        [-(RL1+Ron)/L1, 0,                        0,
        [-Ron/L2,                -(RC1+RL2+Ron)/L2,      1/L2,              0
        [0,                              -1/C1,                     0,
        [0,                              0,                                 0,
#    L1                          L2                              C1
])

# entradas:  vi e vd
B1p = sp.Matrix([
        [1/L1,   -1/L1    ],       # iL1
        [0,             0               ],       # iL2
        [0,             0               ],       # vC1
        [0,             0               ],       # vC2
#       vi              vd
])

display('A1p=', A1p)
display('B1p=', B1p)
```

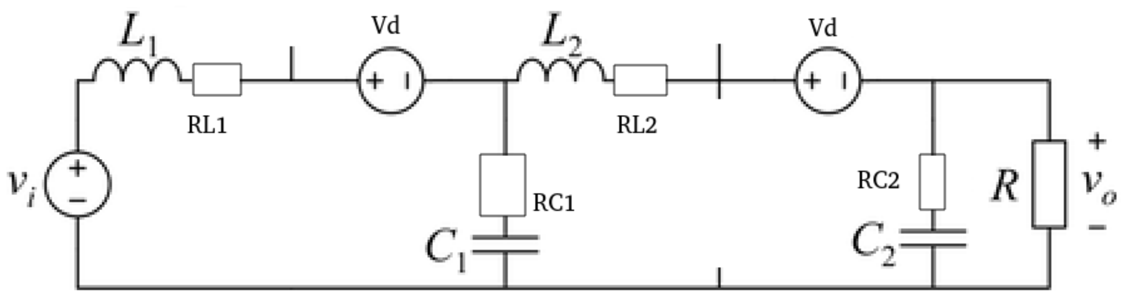
'A1p='

$$\left[\begin{matrix}\frac{- R_{L1} - R_{on}}{L_{1}} & 0 & 0 & 0\\- \frac{R_{on}}{L_{2}} & \frac{- R_{C1} - R_{L2} - R_{on}}{L_{2}} & \frac{1}{L_{2}} & 0\\0 & - \frac{1}{C_{1}} & 0 & 0\\0 & 0 & 0 & - \frac{1}{C_{2} \left(R + R_{C2}\right)}\end{matrix}\right]$$

'B1p='

$$\left[\begin{matrix}\frac{1}{L_{1}} & - \frac{1}{L_{1}}\\0 & 0\\0 & 0\\0 & 0\end{matrix}\right]$$

b) 2ª Etapa de operação



In [ ]:
```
# vo=(vC2+RC2*iL2)/(1-RC2/R)
# iC2 = (iL2*(R - 2*RC2) - vC2)/(R - RC2)
# dvC2/dt = (iL2*(R - 2*RC2)/C2 - vC2/C2)/(R - RC2)
# dvC2/dt = iL2*(R - 2*RC2)/(C2(R - RC2))   - vC2)/(C2(R - RC2))

A2p = sp.Matrix([
        [-(RL1+RC1)/L1,         RC1/L1,
        [RC1/L2,                        -(RC1+RL2)/L2,
        [1/C1,                          -1/C1,
        [0,                                     (R - 2*RC2)/(C2*(R - RC2)),
#    L1                                 L2                              C1
])

B2p = sp.Matrix([
        [1/L1,  -1/L1   ],      # iL1
        [0,             -1/L2   ],      # iL2
        [0,             0               ],      # vC1
        [0,             0               ],      # vC2
])

display('A2p=', A2p)
display('B2p=', B2p)
```

'A2p='

$$\left[\begin{matrix}\frac{- R_{C1} - R_{L1}}{L_{1}} & \frac{R_{C1}}{L_{1}} & - \frac{1}{L_{1}} & 0\\\frac{R_{C1}}{L_{2}} & \frac{- R_{C1} - R_{L2}}{L_{2}} & \frac{1}{L_{2}} & - \frac{1}{L_{2}}\\\frac{1}{C_{1}} & - \frac{1}{C_{1}} & 0 & 0\\0 & \frac{R - 2 R_{C2}}{C_{2} \left(R - R_{C2}\right)} & 0 & - \frac{1}{C_{2} \left(R - R_{C2}\right)}\end{matrix}\right]$$

'B2p='

$$\left[\begin{matrix}\frac{1}{L_1} & -\frac{1}{L_1}\\0 & -\frac{1}{L_2}\\0 & 0\\0 & 0\end{matrix}\right]$$

Definindo valores das variáveis de perdas

In [ ]:
```
params.update({
        RL1: 1,
        RL2: 0.1,
        RC1: 0.01,
        RC2: 0.1,
        Vd: 1,
        Ron: 0.05
});
```

c) Média das duas etapas

In [ ]:
```
Ap = (d*A1p + (1-d)*A2p).subs(params)
Bp = (d*B1p + (1-d)*B2p).subs(params)
Xp = -Ap**-1*Bp*Up.subs(params)
display('Xp=', Xp.evalf(6))

# Obs: vo=(vC2 + RC2*iL2)/(1-RC2/R)
Cp = sp.Matrix([
    [0,      RC2/(1-RC2/R),      0,      1/(1-RC2/R)]
]).subs(params)

Gvdp = (Cp*((s*sp.eye(4)-Ap)**-1)*( (A1p-A2p).subs(params)*Xp + (B1p-B2p).s
Gvdp = Gvdp.factor(s).simplify()
display('Gvdp=', Gvdp.evalf(3))

Gvdp_tf = sympy_rational_to_control_tf(Gvdp)
```

'Xp='

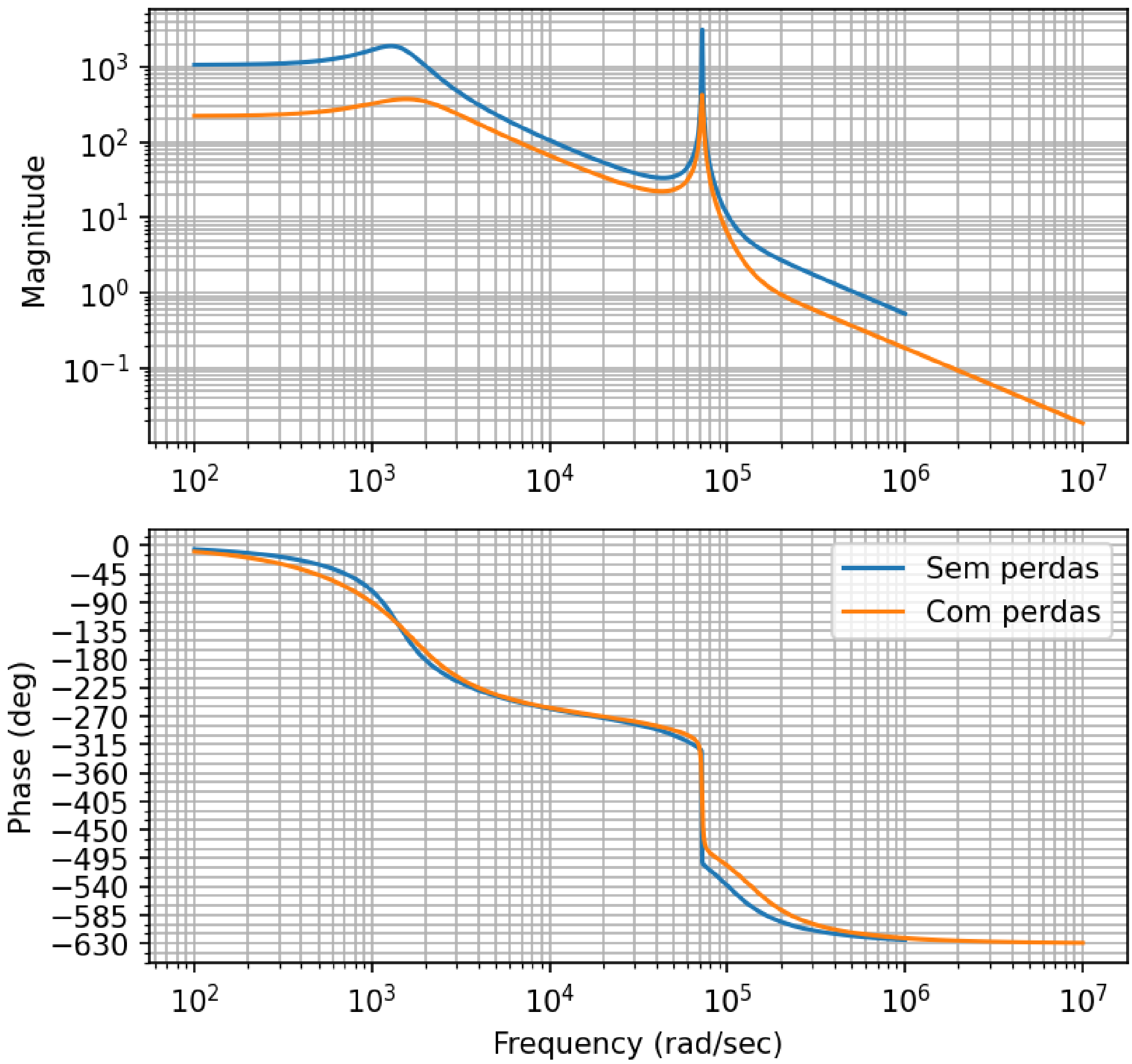
$$\left[\begin{matrix}15.0244\\6.76097\\70.0562\\151.847\end{matrix}\right]$$

'Gvdp='

$$\frac{-7.65\cdot 10^{-28}s^7 + 1.06\cdot 10^{-22}s^6 - 1.75\cdot 10^{-17}s^5 + 5.22\cdot 10^{-13}s^4 - 7.1\cdot 10^{-8}s^3 - 7.\ldots + 221.0}{(6.42\cdot 10^{-17}s^4 + 2.19\cdot 10^{-13}s^3 + 3.34\cdot 10^{-7}s^2 + 0.000677s + 1.\ldots}$$

Comparação entre modelos com e sem perdas

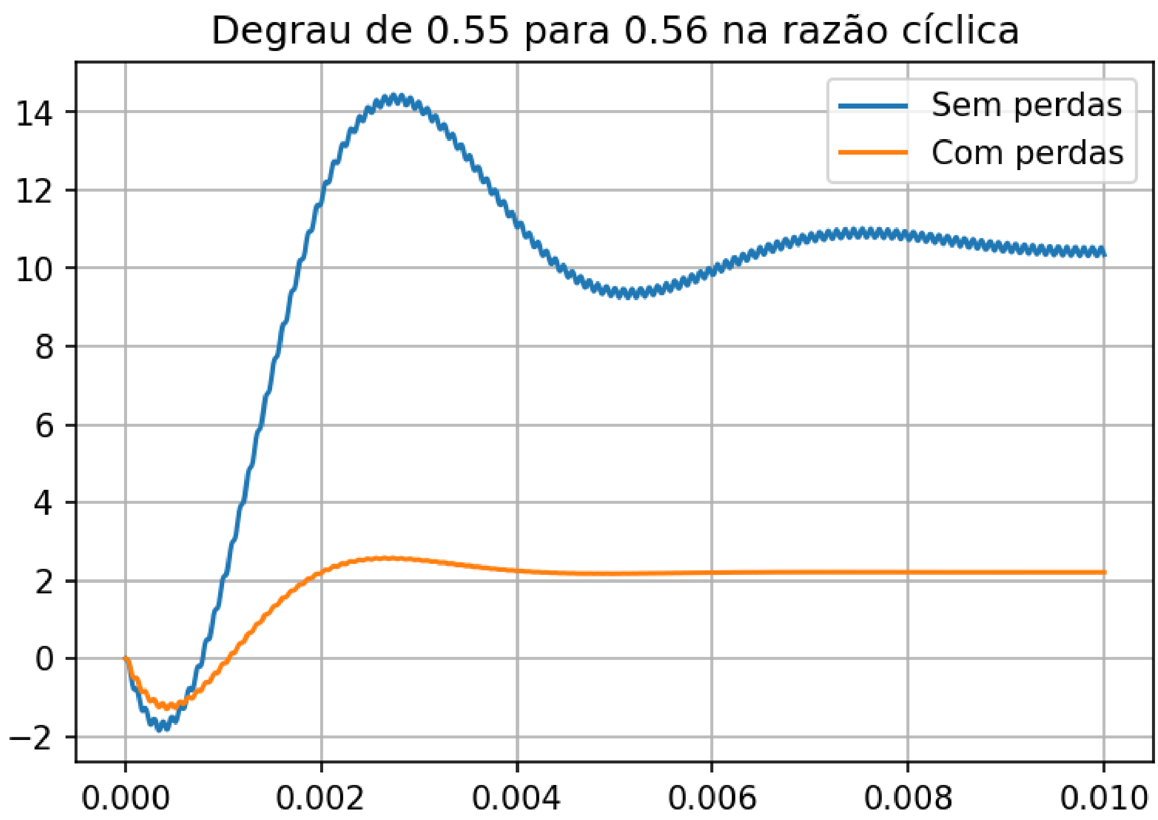
In [ ]:
```
plt.figure(dpi=150, figsize=(6,6))
ctl.bode(Gvd_tf);
ctl.bode(Gvdp_tf);
plt.legend(["Sem perdas", "Com perdas"]);
```



In [ ]:

```
plt.figure(dpi=150)

t=np.linspace(0,10e-3, 1000)

Vi = 48
D = 0.55

_, x = ctl.step_response(Gvd_tf,T=t, X0=0);
_, xp = ctl.step_response(Gvdp_tf,T=t, X0=0);
plt.plot(t,x/100); plt.grid(True);
plt.plot(t,xp/100); plt.grid(True);
plt.title("Degrau de 0.55 para 0.56 na razão cíclica");
plt.legend(["Sem perdas", "Com perdas"]);
```

```
/home/eduardo/.local/lib/python3.10/site-packages/scipy/sparse/linalg/_matf
uncs.py:708: LinAlgWarning: Ill-conditioned matrix (rcond=6.09441e-21): res
ult may not be accurate.
  return solve(Q, P)
/home/eduardo/.local/lib/python3.10/site-packages/scipy/sparse/linalg/_matf
uncs.py:708: LinAlgWarning: Ill-conditioned matrix (rcond=1.78027e-49): res
ult may not be accurate.
  return solve(Q, P)
```

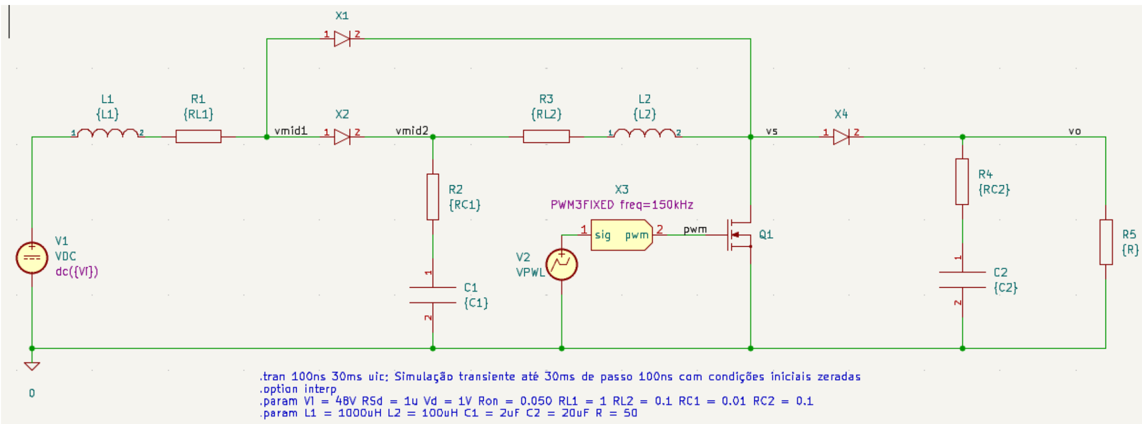


III) Simulações para validação

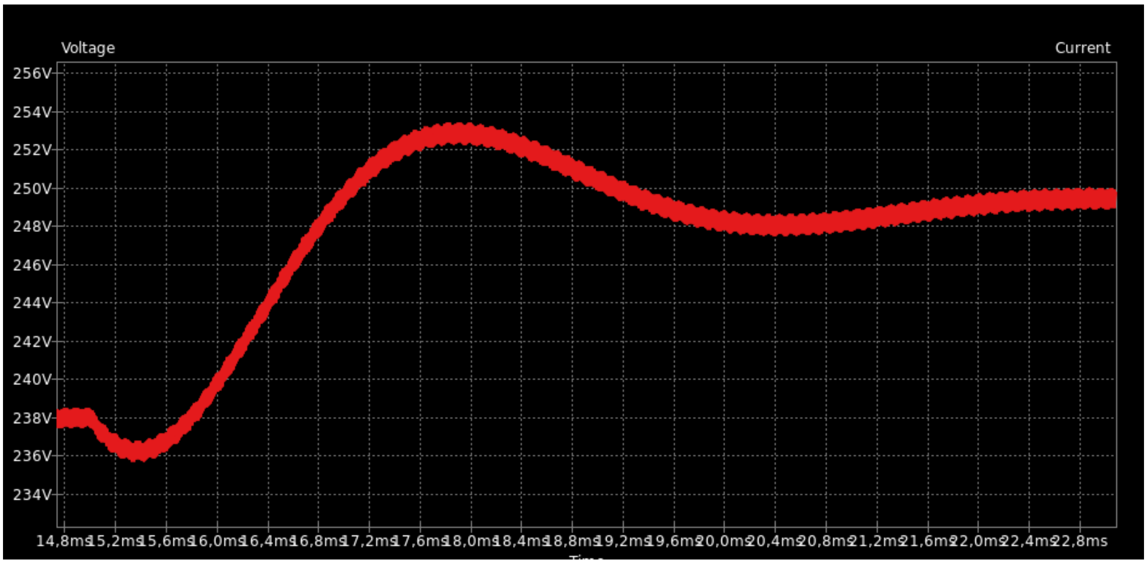
Para fins de comparação, foram simulados os seguintes modelos em Ngspice com captura de esquemático e geração de gráficos do KiCad:

1. Circuito comutado (sem perdas)
2. Circuito comutado (com perdas)

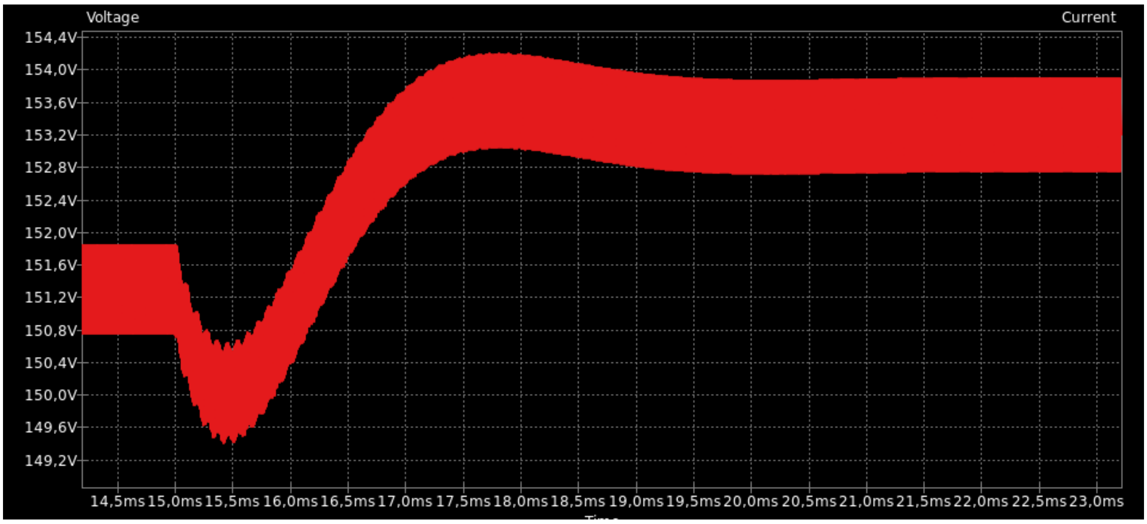
Abaixo está ilustrado o modelo comutado:



A figura abaixo ilustra o resultado do degrau de $d = 0.55$ para $d = 0.56$ aos $15ms$ da análise transiente para o modelo sem perdas.



Ao considerar as perdas, a resposta ao degrau pode ser visualizada abaixo



Conclusão

A simulação demonstra que a mesma variação da tensão de saída v_o a partir do ponto de operação é obtida pelas funções de transferências encontradas:

- Para o caso sem perdas, o ganho estático de d para v_o é em torno de 1000 (aumento de 0.01 da razão cíclica causa um aumento de 10V na saída)
- Já para o caso com perdas, o ganho estático ficou próximo de 200 (aumento de 0.01 da razão cíclica causa um aumento de 2V na saída)