Univerzitet u Beogradu — Elektrotehnički fakultet

Rad

iz predmeta

Teorija električnih kola

# IZVOĐENJE BRANINOVIH JEDNAČINA

Katarina Stanković
2018/0183
Odsek: Računarska tehnika i informatika

decembar 2020.

# 1 Uvod

SymPy [1] je besplatna (*free open-source software*) Python [2][3] biblioteka za simboličku matematiku. Ovaj alat teži da postane potpun CAS (*computer algebra system*) paket, napisan isključivo u Python-u, koji je jedan od najpopularnijih programskih jezika današnjice. Nekoliko prednosti SymPy alata koji kreatori navode su zavisnost od samo Python-a i jedne njegove biblioteke, samim tim i malo memorijsko zauzeće (netipično za CAS pakete), kao i mogućnost korišćenja SymPy biblioteke u bilo kom okruženju koji podržava Python. Za praćenje ovog rada preporučena je Anaconda [4], besplatna distribucija (*free open-source distribution*) programskog jezika Python. U okviru nje se nalazi preporučeno okruženje Jupyter Notebook [5].
Korišćenjem SymPy biblioteke možemo izvesti Braninove jednačine [6].

# 2 Izvođenje Braninovih jednačina

Idealan vod je uniforman vod bez gubitaka sa homogenim dielektrikom i savršenim provodnikom na kome se prostire transverzalan elektromagnetski talas.

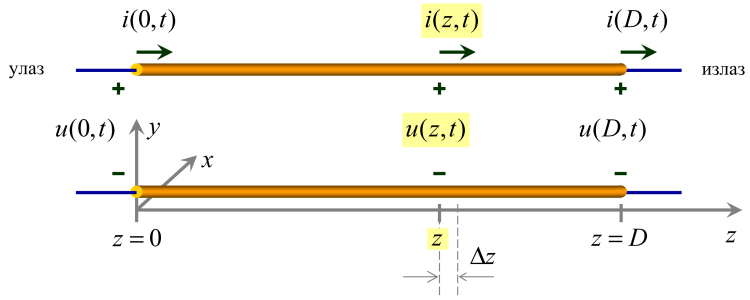


Figure 1: Idealan vod

Jednačine telegrafičara idealnog voda su:

$$\left\{ \begin{array}{l} -\dfrac{\partial u(z,t)}{\partial z} = L' \dfrac{\partial i(z,t)}{\partial t} \\ -\dfrac{\partial i(z,t)}{\partial z} = C' \dfrac{\partial u(z,t)}{\partial t} \end{array} \right\}$$

a njegove talasne jednačine napona i struje su:

$$\left\{ \begin{array}{l} \dfrac{\partial^2 u(z,t)}{\partial z^2} = L'C' \dfrac{\partial^2 u(z,t)}{\partial t^2} \\ \dfrac{\partial^2 i(z,t)}{\partial z^2} = C'L' \dfrac{\partial^2 i(z,t)}{\partial t^2} \end{array} \right\}$$

Proizvod $C'L'$ možemo izraziti preko podužnog kašnjenja voda $T' = \sqrt{C'L'}$.

Primenom Laplasove transformacije, smatrajući da nema početne energije, dobijaju se kompleksne jednačine telegrafičara:

$$\left\{ \begin{array}{l} -\dfrac{d\underline{U}(z)}{dz} = \underline{s}L'\underline{I}(z) \\ -\dfrac{d\underline{I}(z)}{dz} = \underline{s}C'\underline{U}(z) \end{array} \right\}$$

i kompleksne talasne jednačine:

$$\left\{ \begin{array}{l} \dfrac{d^2\underline{U}(z)}{dz^2} = (\underline{s}T')^2\underline{U}(z) \\ \dfrac{d^2\underline{I}(z)}{dz^2} = (\underline{s}T')^2\underline{I}(z) \end{array} \right\}$$

Definisaćemo neke od simbola koje ćemo dalje koristiti. Ako ne navedemo osobine simbola, podrazumevano je da predstavljaju kompleksne brojeve.

```
from sympy import *
init_printing()
A1, A2, Uz, Iz, Tprim, Lprim = symbols('A1, A2, Uz, Iz, T\', L\'')
s,z = symbols('s,z')
```

Rešavanjem obične homogene linearne diferencijalne jednačine drugog reda sa konstantnim koeficijentima dobija se opšte rešenje za kompleksan napon i kompleksnu struju voda u preseku na koordinati $z$.

```
Uz = A1*exp(-s*Tprim*z)+A2*exp(s*Tprim*z)
Iz = -diff(Uz,z)/(s*Lprim)
```

Uvodimo karakterističnu impedansu voda $Z_c = \dfrac{T'}{L'} = \sqrt{\dfrac{L'}{C'}}$.

```
Zc = symbols('Z_c', positive = True)
```

Izrazićemo $L'$ preko karakteristične impedanse voda i uvrstiti u $Iz$, koristeći funkciju `subs`. Funkcija `expand` će predstaviti izraz kao sumu.

```
Iz = expand(Iz.subs({Lprim:Zc*Tprim}))
```

Opšte rešenje postaje:

```
Uz, Iz
```

$$\left(A_1 e^{-T'sz} + A_2 e^{T'sz},\ \frac{A_1 e^{-T'sz}}{Z_c} - \frac{A_2 e^{T'sz}}{Z_c}\right)$$

Uzimajući koordinatu $z = 0$ možemo dobiti kompleksan napon i kompleksnu struju na početku voda, a uvođenjem kašnjenje voda $\tau = T'd$ možemo izraziti kompleksan napon i kompleksnu struju na kraju voda, ali za neusaglašene smerove.

```
U0, I0, Ud, Id = symbols('U0, I0, Ud, Id')
d = symbols('d')
tau = symbols(r'\tau')
```

Pravimo jednačine menjajući koordinatu $z$ i $T'$ i postavljamo atribut `simultaneous` takav da zabrani optimizacije izraza pre nego što se uvrste i $z$ i $T'$.

```
jednacine = []
jednacine.append(Eq(U0, Uz.subs({z:0, Tprim:tau/d}, simultaneous=True)))
jednacine.append(Eq(I0, Iz.subs({z:0, Tprim:tau/d}, simultaneous=True)))
jednacine.append(Eq(Ud, Uz.subs({z:d, Tprim:tau/d}, simultaneous=True)))
jednacine.append(Eq(Id, Iz.subs({z:d, Tprim:tau/d}, simultaneous=True)))
```

Nalazimo $A_1$ iz prve jednačine tako što rešimo tu jednačinu po promenljivoj $A_1$ koristeći funkciju `solveset`, dok nalazimo $A_1$ i $A_2$ iz druge i treće jednačine, koje predstavljaju sistem nelinearnih jednačina koje rešavamo funkcijom `nonlinsolve`.

```
A1_iz_J1 = simplify(list(solveset(jednacine[1], A1))[0])
A1A2_iz_J2J3 = list(nonlinsolve(jednacine[2:], A1, A2))[0]
```

Uvrstićemo $A_1$ iz prve jednačine i $A_2$ izvučenog iz sistema jednačina, u nultu jednačinu ($U_0$).

```
U0jednacina = expand(jednacine[0].subs({A1:A1_iz_J1,A2:A1A2_iz_J2J3[1]}))
```

Sada nalazimo $A_2$ iz treće jednačine tako što rešimo tu jednačinu po promenljivoj $A_2$ koristeći funkciju `solveset`, dok nalazimo $A_1$ i $A_2$ iz nulte i prve jednačine, koje predstavljaju sistem nelinearnih jednačina koje rešavamo funkcijom `nonlinsolve`.

```
A2_iz_J3 = simplify(list(solveset(jednacine[3], A2))[0])
A1A2_iz_J0J1 = list(nonlinsolve(jednacine[:2], A1, A2))[0]
```

Uvrstićemo $A_2$ iz treće jednačine i $A_1$ izvučenog iz sistema jednačina, u drugu jednačinu ($U_d$).

```
Udjednacina = jednacine[2].subs({A2:A2_iz_J3})
Udjednacina = expand(Udjednacina.subs({A1:A1A2_iz_J0J1[0]}))
```

Usvajamo oznake uobičajene za elemente sa dva pristupa, za standardne smerove:

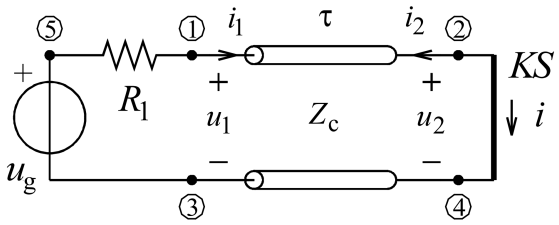


Figure 2: Usaglašeni smerovi voda

```
U1, I1, U2, I2 = symbols('U1, I1, U2, I2')
```

Uvrstićemo uobičajene oznake u jednačine $U_0$ i $U_d$, gde je $I_1 = I_0$, $U_1 = U_0$, $I_2 = -I_d$ i $U_2 = U_d$.

```
idealanVodLT = []
idealanVodLT.append(expand(U0jednacina.subs({I0:I1, Id:-I2, Ud:U2, U0:U1})))
idealanVodLT.append(expand(Udjednacina.subs({I0:I1, Id:-I2, Ud:U2, U0:U1})))
```

Sada možemo videti kompleksne Braninove jednačine:

```
for jednacina in idealanVodLT:
    display(jednacina)
```

$$U_1 = Z_c I_1 + Z_c I_2 e^{-\tau s} + U_2 e^{-\tau s}$$
$$U_2 = Z_c I_1 e^{-\tau s} + Z_c I_2 + U_1 e^{-\tau s}$$

Primenjujući $\underline{U} e^{-\underline{s}\tau} \leftrightarrow u(t-\tau)$ (inverzna Lapslasova transformacija), dobijaju se jednačine idealnog voda.

```
e = Symbol('e')
```

Prvo ćemo zameniti exponencijalnu funkciju $e^{-s\tau}$ simbolom $e$ funkcijom `replace`, pa onda uvrstiti transformacije.

```
for i in range(len(idealanVodLT)):
    idealanVodLT[i] = idealanVodLT[i].replace(exp(-tau*s), e).subs( \
                                        { \
                                        U1*e:Symbol(r'u_{1}(t-\tau)'), \
                                        U2*e:Symbol(r'u_{2}(t-\tau)'), \
                                        I1*e:Symbol(r'i_{1}(t-\tau)'), \
                                        I2*e:Symbol(r'i_{2}(t-\tau)'), \
                                        U1:Symbol(r'u_{1}(t)'), \
                                        U2:Symbol(r'u_{2}(t)'), \
                                        I1:Symbol(r'i_{1}(t)'), \
                                        I2:Symbol(r'i_{2}(t)') \
                                        })
```

```
display(idealanVodLT[0])
display(idealanVodLT[1])
```

$$u_1(t) = Z_c i_1(t) + Z_c i_2(t-\tau) + u_2(t-\tau)$$
$$u_2(t) = Z_c i_1(t-\tau) + Z_c i_2(t) + u_1(t-\tau)$$

Ove jednačine su poznate kao Braninove jednačine.

# 3 Primena Braninovih jednačina

Braninovim jednačinama možemo da modelujemo idealne vodove ekvivalentnim šemama [6].

U softverskom alatu SPICE (*Simulation Program with Integrated Circuit Emphasis*), idealni vod se implementira kao vremenski zakašnjeni kontrolisani generatori na osnovu Braninovih jednačina [7].

Grafičko programsko okruženje zasnovan na MATLAB-u, Simulink, ima više vrsti vodova na raspolaganju. Idealni vod je modelovan Braninovim jednačinama i pruža najefikasniju simulaciju od ponuđenih vrsta vodova [8].

# Reference

[1] Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. (2017) SymPy: symbolic computing in Python. PeerJ Computer Science 3:e103 https://doi.org/10.7717/peerj-cs.103

[2] Allen Downey, Think Python: How to Think Like a Computer Scientist, Green Tea Press, 2008.

[3] Paul Gerrard, Lean Python: Learn Just Enough Python to Build Useful Tools, Apress, 2016.

[4] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Web. <https://anaconda.com>.

[5] Thomas Kluyver and Benjamin Ragan-Kelley and Fernando Pérez and Brian Granger and Matthias Bussonnier and Jonathan Frederic and Kyle Kelley and Jessica Hamrick and Jason Grout and Sylvain Corlay and Paul Ivanov and Damián Avila and Safia Abdalla and Carol Willing and Jupyter development team. Jupyter Notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Scmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, Netherlands, 2016. IOS Press.

[6] F. H. Branin, "Transient Analysis of Lossless Transmission Lines", Proceedings of the IEEE, Volume 55, Issue 11, Pages 2012-2013, Nov. 1967. DOI: 10.1109/PROC.1967.6033

[7] Paul R. Clayton, Analysis of Multiconductor Transmission Lines, 2/e, Wiley IEEE Press, 2008.

[8] Transmission Line. Retrieved from https://www.mathworks.com/help/physmod/sps/ref/transmissionline.html