Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática – CEEI
Circuitos Elétricos II

Projeto de simulação de uma linha de transmissão por associação de quadripolos

Aluno: Carlos Renan Correia Fidelis — Matrícula: 121110703
Aluno: José Ívines Matos Silva — Matrícula: 123211061
Aluno: Lucas Martins Bezerra — Matrícula: 120110422
Aluno: Lucila Maria Orestino Pereira — Matrícula: 119111041
Aluno: Lucivaldo Barbosa de Aguiar Júnior — Matrícula: 119210605
Professor: Dr. Luiz Augusto Medeiros Martins Nóbrega — Turma: 01

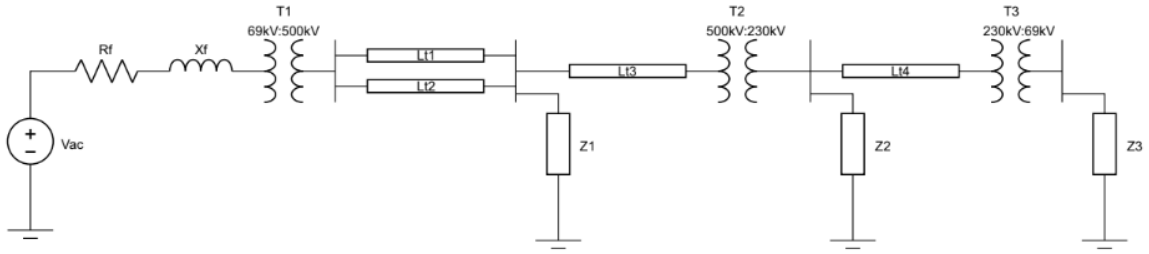
Campina Grande, 2024.

Sumário

1 Introdução

O nosso principal objetivo é desenvolver um software capaz de simular um sistema elétrico que representa uma linha de transmissão conforme Figura 4.

Figura 1: Sistema elétrico



Criaremos funções em Python para associar as matrizes dos quadripolos de cada um dos estágios da linha, sejam quadripolos de cargas "shunt", linha de transmissão em paralelo, transformadores, impedâncias em série e quadripolos no geral associados em cascata.

Iremos também ajustar as tensões nas cargas que são alteradas devido à presença de indutores e capacitores no sistema elétrico a fim de alcançarmos os valores desejados. Serão feitos ajustes de tap nos transformadores e associação de reatores ou banco de capacitores em paralelo com as cargas conforme a necessidade, seja subir ou diminuir as tensões nas cargas.

Além disso, o sistema elétrico foi simulado no orcad Capture cis Lite e PSpice para conferir os resultados. No código escrito em Python foram geradas as matrizes de transmissão em cada um dos pontos em que houve necessidade de ajuste de tensão.

Os sistemas de equações serão solucionados considerando que a tensão na fonte de entrada é igual a $69kV_{RMS}$

2 Código Python

2.1 Variáveis características gerais

O trecho de código 3 importa a biblioteca numpy, e define valores de impedância Thévenin, impedâncias dos modelos do transformador e impedância das cargas.

```
import numpy as np

w  = 120*np.pi

Rf, Xf = 4, 0.38j #par metros de imped ncia em s rie com o
    transformador T1

Zth = Rf + Xf #imped ncia  s rie Thevenin

R1, X1, = 7.6e-3, 3.8e-3j #parametros gerais do transformador

R2, X2 = 33.9e-3, 0.85e-3j #parametros gerais do transformador

Z1 = R1 + X1 #impedancia 1 do transformador

Z2 = R2 + X2 #impedancia 2 do transformador

#impedancias "shunt" dos transformadores
ZT1 =  (4320 * 5050j) / (4320 + 5050j)

ZT2 = (432000 * 505000j) / (432000 + 505000j)

ZT3 = (402000 * 607000j) / (402000 + 607000j)

#Cargas
Zc1 = 8400 + (1j* w * 46)

Zc2 = 1175.55 + (1j* w * 6.43)

Zc3 = 529 + (1j * w * 2.9)
```

Source Code 1: Declaração das variáveis caracteríticas

Em seguida foram definidas funções que modelam os principais quadripolos que serão utilizados na simulação. Foi decidido seguir a teoria de quadripolos definida por (GLOVER et al., 2015), ou seja, a corrente entra pelos terminais à esquerda e sai pelo terminal à direita. Foram definidas funções para os principais quadripolos de impedância, admitância, associação em cascata e as topologias pi e t mostrados por (GLOVER et al.,2015).

Além disso, foi definido uma função de quadripolo para as linhas de transmissão, um transformador ideal e uma função que multiplica matrizes de forma recursiva, além da função de associação em paralelo que foi escrita baseada em notas de aula do professor Luiz.

```
#-=-=-=-=-=-=-=-=-=Modelos de Transmissao-=-=-=-=-=-=-=-=-=#

def TransformadorIdeal(N1,N2):
    #esta funcao retorna a matriz de transmissao de um transformador ideal
    #considerando que I2 sai do pelo a diretia

    matriz_Transformador = np.array([[ N1/N2 ,   0   ],
                        [   0   , N2/N1 ]])

    return matriz_Transformador

def ImpedanciaSerie(Z):

    matriz_Z = np.array([[ 1 , Z ],
                         [ 0 , 1 ]])

    return matriz_Z

def AdmitanciaShunt(Y):

    matriz_Y = np.array([[ 1 , 0 ],
                         [ Y , 1 ]])
    return matriz_Y

def CircuitoT(Z1, Z2, Y):

    matriz_T = np.array([[ 1 + (Y * Z1) , Z1 + Z2 + (Y * Z1 * Z2) ],
                         [      Y       ,       1 + (Y * Z2)      ]])

    return matriz_T

def CircuitoPI(Z, Y1, Y2):

    matriz_Pi = np.array([[ 1 + (Y2 * Z)          ,       Z     ],
                         [ Y1 + Y2 + (Y1* Y2 * Z) , 1 + (Y1 * Z)]])

    return matriz_Pi
```

```
38
39
40 def LinhaDeTransmissao(Comprimento):
41
42     #a entrada de comprimento deve ser feita em Km
43
44     R = 0.182 * Comprimento
45
46     L = 1j*w * 2.28e-3  * Comprimento
47
48     C1 =  0.0140e-6 * Comprimento
49
50     C = C1/2
51
52     C_fasorial = 1 / (1j*120* np.pi * C)
53
54     Z = R + L
55
56     Y1 = 1 / C_fasorial
57
58     Y2 = 1 / C_fasorial
59     return CircuitoPI(Z, Y1, Y2)
60
61 def Cascata(*matrizes):
62     resultado = matrizes[0]
63     for matriz in matrizes[1:]:
64         resultado = np.dot(resultado, matriz)
65     return resultado
66
67
68 def QuadripoloParalelo(matriz1, matriz2):
69
70     Aa, Ba, Ca, Da = matriz1[0][0], matriz1[0][1], matriz1[1][0], matriz1
       [1][1]
71
72     Ab, Bb, Cb, Db = matriz2[0][0], matriz2[0][1], matriz2[1][0], matriz2
       [1][1]
73
74     den = Ba + Bb
75
76     A = (( Aa * Bb ) + (Ab * Ba ) ) / den
77
78     B = ( Ba * Bb ) / den
79
80     C = ( Ca + Cb + ( ( Aa - Ab )*( Db - Da ) / den ) )
81
82     D = ( (Bb * Da ) + ( Ba * Db )) / den
```

```
83
84      matriz_Paralelo = np.array( [[ A , B],
85                                  [ C , D]])
86
87      return matriz_Paralelo
```

Source Code 2: Definição das funções que retornam as matrizes de quadripolo

Em seguida, as funções criadas foram utilizadas para modelar os quadripolos adequadamente.

```
1  #-=-=-=-=-=-=-=-=-=Quadripolos-=-=-=-=-=-=-=-=-=#
2
3  #impedancia em serie com a fonte
4
5  serie = ImpedanciaSerie(Zth)
6
7  #transformadores
8
9  T1 = np.dot(CircuitoT(Z1, Z2, 1/ZT1), TransformadorIdeal(69, 500))
10
11 T2 = np.dot(CircuitoT(Z1, Z2, 1/ZT2), TransformadorIdeal(500, 230))
12
13 T3 = np.dot(CircuitoT(Z1, Z2, 1/ZT3), TransformadorIdeal(230, 69))
14
15 #Linhas de transmissao
16
17 LT1 = LinhaDeTransmissao(100)
18
19 LT2 = LinhaDeTransmissao(100)
20
21 LT3 = LinhaDeTransmissao(100)
22
23 LT4 = LinhaDeTransmissao(80)
24
25 #Cargas
26
27 CargaZ1 = AdmitanciaShunt(1/Zc1)
28
29 CargaZ2 = AdmitanciaShunt(1/Zc2)
30
31 CargaZ3 = AdmitanciaShunt(1/Zc3)
```

Source Code 3: Declaração das variáveis caracteríticas

No trecho de código 4 a função cascata é chamada na linha 3 com todas as matrizes de quadripolo da linha como argumento da função que retornará a matriz de transmissão do sistema elétrico como um todo, em seguida são utilizados elementos da matriz de transmissão para modelar o sistema de equações que descrevem o sistema elétrico que podemos resolver para encontra a tensão e a corrente na carga Z3.

A lógica é repetida nas linhas 31 e 53 a fim de obter os valores de tensão e corrente para Z2 e Z1 respectivamente.

```
1  #-=-=-=-=-=-=-=-=-=Matriz de transmissao-=-=-=-=-=-=-=-=-=-=#
2
3  ABCD = Cascata(serie, T1, QuadripoloParalelo(LT1, LT2,),
4                            CargaZ1, LT3, T2, CargaZ2, LT4, T3, CargaZ3)
5  print('-=-'*10, 'Linha de transmiss o original', '-=-'*10, '\n')
6
7  print('Matriz da linha de transmiss o: \n', ABCD,'\n')
8
9  #-=-=-=-=-=-=-=-=-=-=solucao do sitema para Z3 -=-=-=-=-=-=-=-=-=-=#
10
11 A = ABCD[0][0]
12
13 B = ABCD[0][1]
14
15 C = ABCD[1][0]
16
17 D = ABCD[1][1]
18
19 Eqs = np.array([[A + (B /Zc3), 0], [-(C+(D/Zc3)), 1]])
20
21 Igualdade = np.array([69e3*np.sqrt(2), 0])
22
23 solucao = np.linalg.solve(Eqs, Igualdade)
24
25 print(f'Para a carga Z3, V = {round(np.abs(solucao[0]),4)}     {np.angle(
       solucao[0])} V\n')
26
27 print(f'Para a carga Z3, I = {np.abs(solucao[0]/Zc3)}      {np.angle(
       solucao[0]/Zc3)} A \n')
28
29 #-=-=-=-=-=-=-=-=-=-=solucao do sitema para Z2 -=-=-=-=-=-=-=-=-=-=#
30
31 ABCD_Z2 = Cascata(serie, T1, QuadripoloParalelo(LT1, LT2,),
32                            CargaZ1, LT3, T2, CargaZ2)
33
34 A2 = ABCD_Z2[0][0]
35
36 B2 = ABCD_Z2[0][1]
```

```
37
38 C2 = ABCD_Z2[1][0]
39
40 D2 = ABCD_Z2[1][1]
41
42 Eqs2 = np.array([[A2 + (B2 /Zc2), 0], [-(C2+(D2/Zc2)), 1]])
43
44 solucao2 = np.linalg.solve(Eqs2, Igualdade)
45
46 print(f'Para a carga Z2, V = {np.abs(solucao2[0])}     {np.angle(solucao2
       [0])} V \n')
47
48 print(f'Para a carga Z2, I = {np.abs(solucao2[0]/Zc2)}      {np.angle(
       solucao2[0]/Zc2)} A \n') #
49
50
51 #-=-=-=-=-=-=-=-=-=solucao do sitema para Z1 -=-=-=-=-=-=-=-=-=#
52
53 ABCD_Z3 = Cascata(serie, T1, QuadripoloParalelo(LT1, LT2,),
54                         CargaZ1)
55
56 A3 = ABCD_Z3[0][0]
57
58 B3 = ABCD_Z3[0][1]
59
60 C3 = ABCD_Z3[1][0]
61
62 D3 = ABCD_Z3[1][1]
63
64 Eqs3 = np.array([[A3 + (B3 /Zc1), 0], [-(C3+(D3/Zc1)), 1]])
65
66 solucao3 = np.linalg.solve(Eqs3, Igualdade)
67
68 print(f'Para a carga Z1, V = {np.abs(solucao3[0])}     {np.angle(solucao3
       [0])} V \n')
69
70 print(f'Para a carga Z1, I = {np.abs(solucao3[0]/Zc1)}      {np.angle(
       solucao3[0]/Zc1)} A \n')
```

Source Code 4: Primeira simulação da linha

A saída mostrada no terminal pode ser encontrada no trecho de código a seguir:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- Linha de transmiss o original
   -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

Matriz da linha de transmiss o:
 [[9.06957856e-01+0.33756796j 3.17500329e+00+8.98746237j]
 [2.42680295e-03+0.07729014j 3.99493400e-01+0.14592859j]]

Para a carga Z3, V = 100046.1519 ∠ -0.35435835243136216 V

Para a carga Z3, I = 82.37419674241151 ∠ -1.4744960703640038 A

Para a carga Z2, V = 327206.1903227826 ∠ -0.31710718725111886 V

Para a carga Z2, I = 121.45475399662357 ∠ -1.4363663787495777 A

Para a carga Z1, V = 711033.7345088145 ∠-0.21092543569162386 V

Para a carga Z1, I = 36.900573460725724 ∠  -1.3306443232281835 A
```

Source Code 5: Saída no terminal

A lógica de uso da função Cascata e solução do sistema foi reutilizada para o ajuste de tap do transformador, ou seja, apenas foram mudados os valores de N2 na função TransformadorIdeal que está sendo multiplicada com o circuito T que representa a entrada do transformador. A implementação é apresentada no trecho de código 6 a seguir

```


T1_tap = np.dot(CircuitoT(Z1, Z2, 1/ZT1), TransformadorIdeal(69, 347.65))

T2_tap = np.dot(CircuitoT(Z1, Z2, 1/ZT2), TransformadorIdeal(500, 225.16))

T3_tap = np.dot(CircuitoT(Z1, Z2, 1/ZT3), TransformadorIdeal(230, 67.45))

ABCD_tap = Cascata(serie, T1_tap, QuadripoloParalelo(LT1, LT2,),
                        CargaZ1, LT3, T2_tap, CargaZ2, LT4, T3_tap,
   CargaZ3)

print('-=-'*10, 'Linha de transmiss o com ajuste de tap', '-=-'*10, '\n')

print('Matriz da linha de transmiss o: \n', ABCD_tap,'\n')

A_tap = ABCD_tap[0][0]

B_tap = ABCD_tap[0][1]

```

```
C_tap = ABCD_tap[1][0]

D_tap = ABCD_tap[1][1]

Eqs_tap = np.array([[A_tap + (B_tap /Zc3), 0], [-(C_tap+(D_tap/Zc3)), 1]])

solucao_tap = np.linalg.solve(Eqs_tap, Igualdade)

print(f'Para a carga Z3 com ajuste de tap, V = {np.abs(solucao_tap[0])}
       {np.angle(solucao_tap[0])} V \n')

print(f'Para a carga Z3 com ajuste de tap, I = {np.abs(solucao_tap[0]/Zc3)
    }      {np.angle(solucao_tap[0]/Zc3)} A \n')

#-=-=-=-=-=-=-=-= Z2 tap -=-=-=-=-=-=-=-=-=-=-=-=-=-

ABCD_tap_Z2 = Cascata(serie, T1_tap, QuadripoloParalelo(LT1, LT2,),
                          CargaZ1, LT3, T2_tap, CargaZ2)

A_tap_Z2 = ABCD_tap_Z2[0][0]

B_tap_Z2 = ABCD_tap_Z2[0][1]

C_tap_Z2 = ABCD_tap_Z2[1][0]

D_tap_Z2 = ABCD_tap_Z2[1][1]

Eqs_tap_Z2 = np.array([[A_tap_Z2 + (B_tap_Z2 /Zc2), 0], [-(C_tap_Z2+(
    D_tap_Z2/Zc2)), 1]])

solucao_tap_Z2 = np.linalg.solve(Eqs_tap_Z2, Igualdade)

print(f'Para a carga Z2 com ajuste de tap, V = {np.abs(solucao_tap_Z2[0])}
       {np.angle(solucao_tap_Z2[0])} V \n')

print(f'Para a carga Z2 com ajuste de tap, I = {np.abs(solucao_tap_Z2[0]/
    Zc2)}      {np.angle(solucao_tap_Z2[0]/Zc2)} A \n')

# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Z1 tap -=-=-=-=-=-=-=-=-=

ABCD_tap_Z1 = Cascata(serie, T1_tap, QuadripoloParalelo(LT1, LT2,),
                          CargaZ1)

A_tap_Z1 = ABCD_tap_Z1[0][0]

B_tap_Z1 = ABCD_tap_Z1[0][1]

```

```
C_tap_Z1 = ABCD_tap_Z1[1][0]

D_tap_Z1 = ABCD_tap_Z1[1][1]

Eqs_tap_Z1 = np.array([[A_tap_Z1 + (B_tap_Z1 /Zc1), 0], [-(C_tap_Z1+(
    D_tap_Z1/Zc1)), 1]])

solucao_tap_Z1 = np.linalg.solve(Eqs_tap_Z1, Igualdade)

print(f'Para a carga Z1 com ajuste de tap, V = {np.abs(solucao_tap_Z1[0])}
        {np.angle(solucao_tap_Z1[0])} V \n')

print(f'Para a carga Z1 com ajuste de tap, I = {np.abs(solucao_tap_Z1[0]/
    Zc1)}      {np.angle(solucao_tap_Z1[0]/Zc1)} A \n')
```

Source Code 6: Simulação com ajuste de tap

A saída mostrada no terminal pode ser encontrada no trecho de código a seguir:

```

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Linha de transmiss o com ajuste de tap
    -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

Matriz da linha de transmiss o:
 [[1.37854497e+00 +0.26313076j 3.27810226e+00+12.2010381j ]
 [1.86871665e-03 +0.05601705j 2.53408209e-01 +0.1013753j ]]

Para a carga Z3 com ajuste de tap, V = 69018.47103655945 ∠
    -0.1886038961016099 V

Para a carga Z3 com ajuste de tap, I = 56.82718428471337 ∠
    -1.3087416140342514 A

Para a carga Z2 com ajuste de tap, V = 230004.40544753004 ∠
    -0.16704496550770748 V

Para a carga Z2 com ajuste de tap, I = 85.37469433023855 ∠
    -1.2863041570061662 A

Para a carga Z1 com ajuste de tap, V = 500050.99287435267 ∠
    -0.10530903842102245 V

Para a carga Z1 com ajuste de tap, I = 25.951185578298517 ∠
    -1.225027925957582 A
```

Source Code 7: Saída no terminal

A lógica de uso da função Cascata foi um pouco alterada, isso tendo em vista que a topologia do circuito foi alteara associando um reator shunt em paralelo com cada uma das cargas para solucionar o problema de sobretensão apresentado no resultado do código 4. Assim, houve uma pequena alteração no argumento da função Cascata, mas a lógica de solução do sistema para encontrar as tensões e corrente se manteve. A implementação mencionada pode ser vista no trecho de Código 8

```

reator1 = AdmitanciaShunt( 1/(1j * w * 600e-3))

reator2 = AdmitanciaShunt( 1/(0.001 + 1j * w * 3))

reator3 = AdmitanciaShunt( 1/(0.001+ 1j * w * 1.25))

ABCD_reator = Cascata(serie, T1, QuadripoloParalelo(LT1, LT2,),
                          CargaZ1, reator1, LT3, T2, CargaZ2, reator2, LT4,
    T3, CargaZ3, reator3)

print('-=-'*10, 'Linha de transmiss o com associa  o de reatores em
   paralelo com a carga', '-=-'*10, '\n')

print('Matriz da linha de transmiss o: \n', ABCD_reator,'\n')

A_reator = ABCD_reator[0][0]

B_reator = ABCD_reator[0][1]

C_reator = ABCD_reator[1][0]

D_reator = ABCD_reator[1][1]

Eqs_reator = np.array([[A_reator + (B_reator /Zc3), 0], [-(C_reator+(
   D_reator/Zc3)), 1]])

Igualdade_reator = np.array([69e3*np.sqrt(2), 0])

solucao_reator = np.linalg.solve(Eqs_reator, Igualdade_reator)

print(f'Para a carga Z3 com reator em paralelo, V = {round(np.abs(
   solucao_reator[0]),4)}     {np.angle(solucao_reator[0])} V \n')

print(f'Para a carga Z3 com reator paralelo, I = {np.abs(solucao_reator
   [0]/Zc3)}      {np.angle(solucao_reator[0]/Zc3)} A \n')

ABCD_reator_Z2 = Cascata(serie, T1, QuadripoloParalelo(LT1, LT2,),
                          CargaZ1, reator1, LT3, T2, CargaZ2, reator2)

```

```
A_reator_Z2 = ABCD_reator_Z2[0][0]

B_reator_Z2 = ABCD_reator_Z2[0][1]

C_reator_Z2 = ABCD_reator_Z2[1][0]

D_reator_Z2 = ABCD_reator_Z2[1][1]

Eqs_reator_Z2 = np.array([[A_reator_Z2 + (B_reator_Z2 /Zc2), 0], [-(
    C_reator_Z2+(D_reator_Z2/Zc2)), 1]])

Igualdade_reator_Z2 = np.array([69e3*np.sqrt(2), 0])

solucao_reator_Z2 = np.linalg.solve(Eqs_reator_Z2, Igualdade_reator_Z2)

print(f'Para a carga Z2 com reator em paralelo, V = {round(np.abs(
    solucao_reator_Z2[0]),4)}     {np.angle(solucao_reator_Z2[0])} V \n')

print(f'Para a carga Z2 com reator paralelo, I = {np.abs(solucao_reator_Z2
    [0]/Zc2)}      {np.angle(solucao_reator_Z2[0]/Zc2)} A \n')

ABCD_reator_Z1 = Cascata(serie, T1, QuadripoloParalelo(LT1, LT2,),
                         CargaZ1, reator1)

A_reator_Z1 = ABCD_reator_Z1[0][0]

B_reator_Z1 = ABCD_reator_Z1[0][1]

C_reator_Z1 = ABCD_reator_Z1[1][0]

D_reator_Z1 = ABCD_reator_Z1[1][1]

Eqs_reator_Z1 = np.array([[A_reator_Z1 + (B_reator_Z1 /Zc2), 0], [-(
    C_reator_Z1+(D_reator_Z1/Zc1)), 1]])

Igualdade_reator_Z1 = np.array([69e3*np.sqrt(2), 0])

solucao_reator_Z1 = np.linalg.solve(Eqs_reator_Z1, Igualdade_reator_Z1)

print(f'Para a carga Z1 com reator em paralelo, V = {round(np.abs(
    solucao_reator_Z1[0]),4)}     {np.angle(solucao_reator_Z1[0])} V \n')

print(f'Para a carga Z1 com reator paralelo, I = {np.abs(solucao_reator_Z1
    [0]/Zc1)}      {np.angle(solucao_reator_Z1[0]/Zc1)} A \n')
```

Source Code 8: Simulação com associação de reatores em paralelo

Cuja saída pode ser vista no trecho de código a seguir:

```
-=-------=-------=-------=-------=- Linha de transmiss o com assocla  o de
   reatores em paralelo com a carga -=------=-------=-------=-------=------=-

Matriz da linha de transmiss o:
 [[1.24250088e+00-0.66501041j 1.13885133e+01+9.74993948j]
 [5.59294159e-03-0.15935321j 2.22688099e+00-0.22484187j]]

Para a carga Z3 com reator em paralelo, V = 68642.5842 ∠
   0.49073640629023046 V

Para a carga Z3 com reator paralelo, I = 56.51769335020412 ∠
   -0.6294013116424112 A

Para a carga Z2 com reator em paralelo, V = 226136.4425 ∠
   0.5015946903895784 V

Para a carga Z2 com reator paralelo, I = 83.93895594910657 ∠
   -0.6176645011088804 A

Para a carga Z1 com reator em paralelo, V = 462187.9373 ∠
   0.5517939798586868 V

Para a carga Z1 com reator paralelo, I = 23.98620361679111 ∠
   -0.5679249076778728 A
```

Source Code 9: Saída no terminal

3 Simulação no PSpice

Devido ao tamanho da linha, o circuito que simula o sistema elétrico será mostrado em figuras diferentes.

Figura 2: Sistema elétrico parte 1

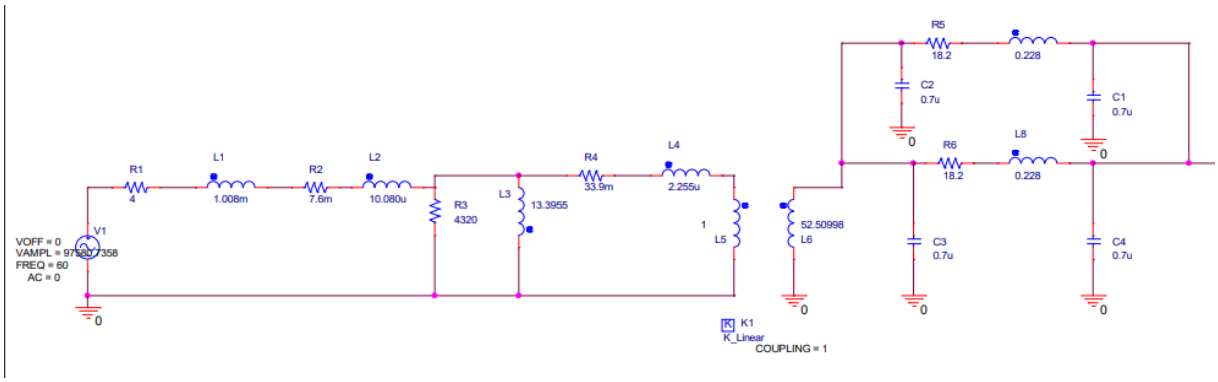


Figura 3: Sistema elétrico parte 2

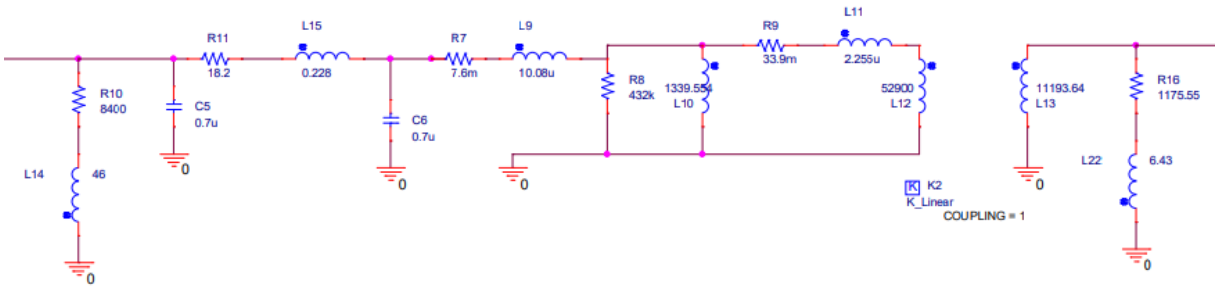
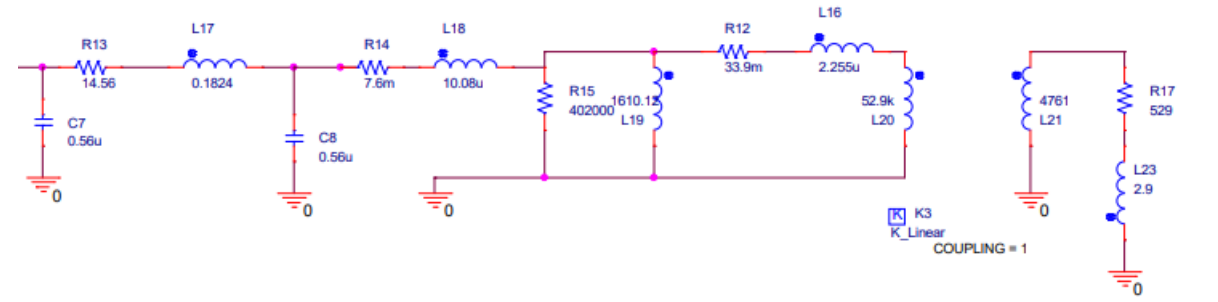
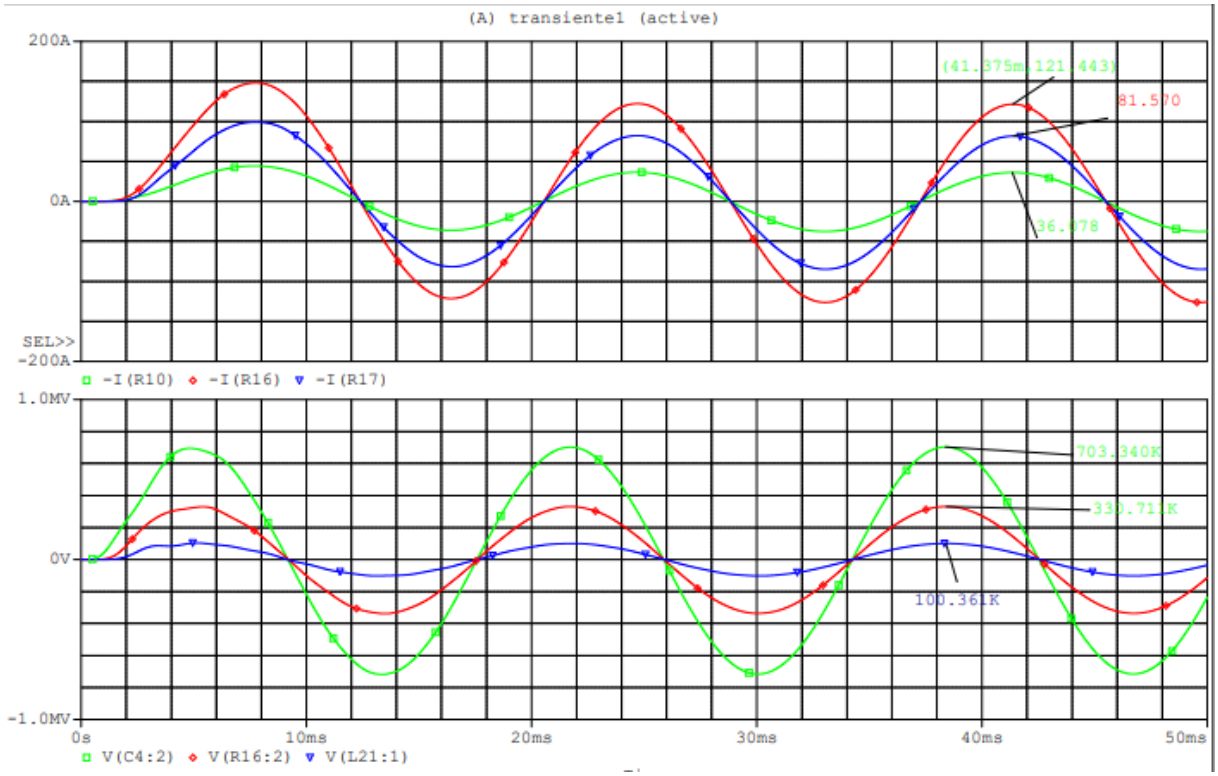


Figura 4: Sistema elétrico parte 3



Com o circuito dessa forma, ou seja, sem nenhum ajuste, podemos medir a tensão e a corrente na carga. Tais leituras podem ser vistas na Figura 5 considerando uma entrada de $69kV$.

Figura 5: Saída



Fonte: PSpice

3.1 Ajuste do Tap

Em seguida foi feito o ajuste de tap alterando o valor de L da segunda bobina de cada um dos transformadores. Os valores alterados podem ser vistos nas Figuras 6, 7, 8.

Figura 6: Ajuste de tap 1

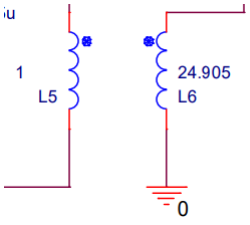


Figura 7: Ajuste de tap 2

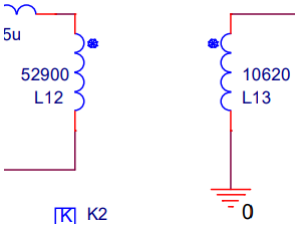
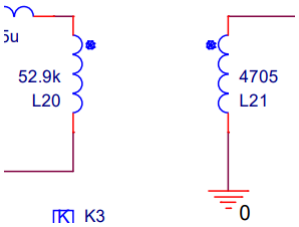
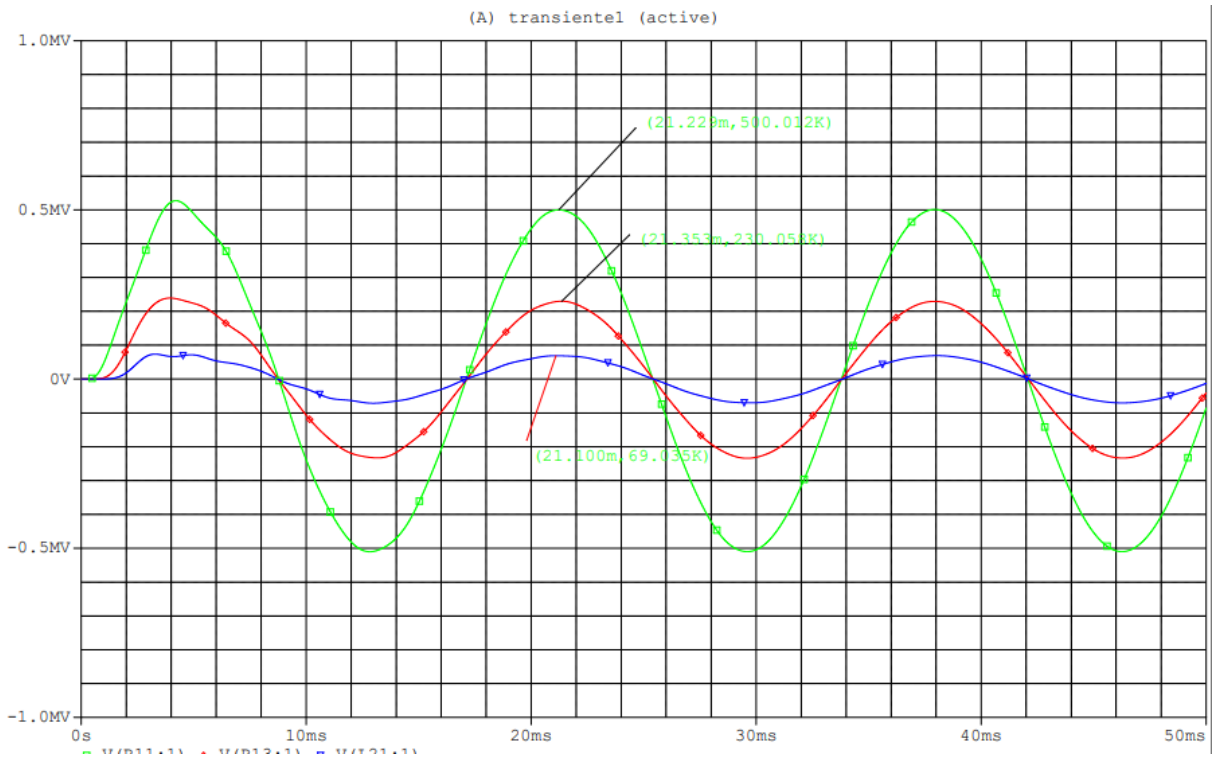


Figura 8: Ajuste de tap 3



A saída com os ajustes pode ser vista na Figura 9

Figura 9: Saída com ajuste de tap



3.2 Associação de reatores paralelos às cargas

Outra forma de ajustar o problema de sobretensão é associar em paralelo às cargas reatores que compensem adequadamente a sobretensão a fim de termos a tensão adequada na saída.

Figura 10: Reator associado à carga Z1

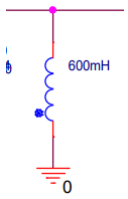


Figura 11: Reator associado à carga Z2

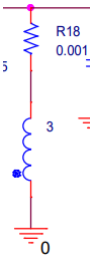
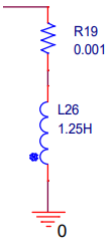
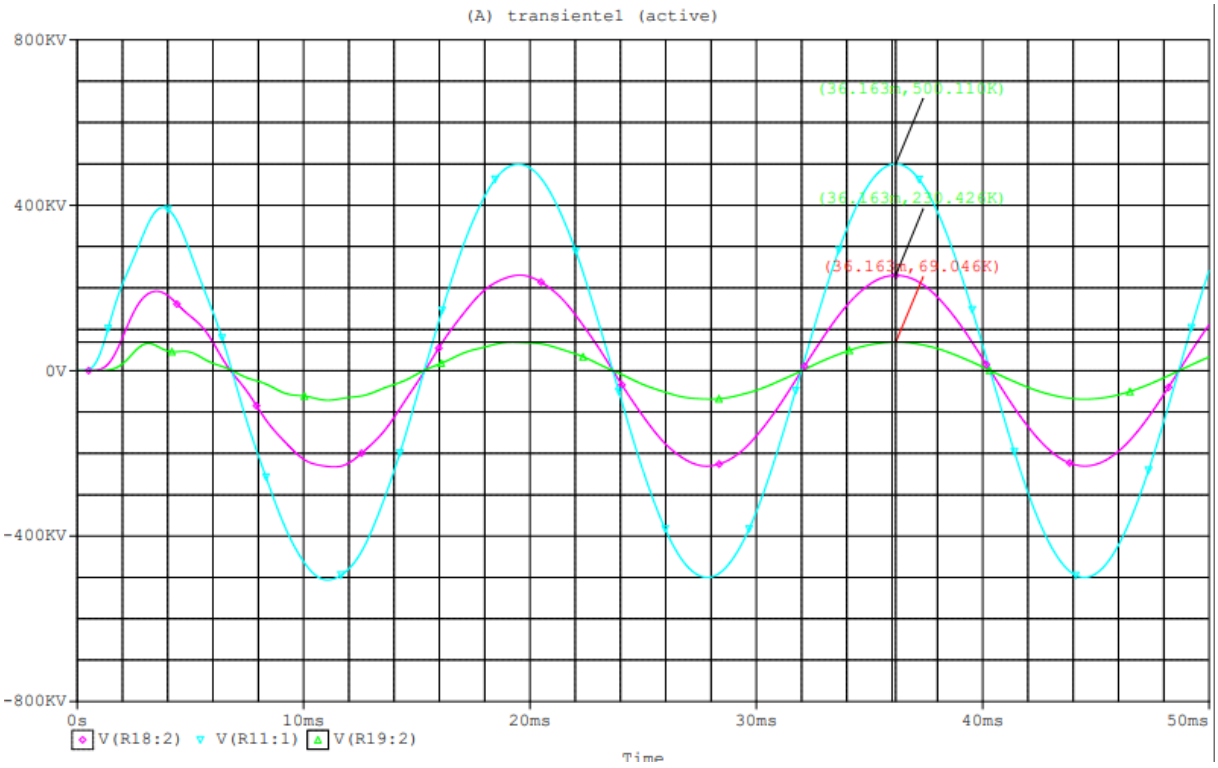


Figura 12: Reator associado à carga Z3



A saída com os ajustes pode ser vista na Figura 13

Figura 13: Saída com associação dos reatores



4 Conclusão

Embora não tenhamos obtido valores exatamente iguais, nas saídas do código e da simulação pelo PSpice, foram obtidos resultados muito próximos conforme o que foi exposto nos capítulos anteriores.

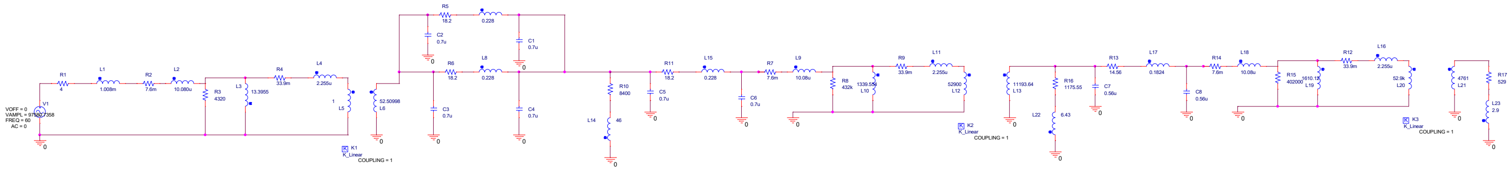
Fomos capazes de simular o comportamento de uma linha de transmissão através de um código desenvolvido em Python, que nos permitiu compreender melhor os efeitos eletromagnéticos presentes no processo de transmissão de energia elétrica.

Não só isso, também fomos capazes de simular maneiras de resolver problemas relacionados aos efeitos eletromagnéticos em uma linha de transmissão por meio de duas técnicas, e ambas mostraram-se muito eficazes.

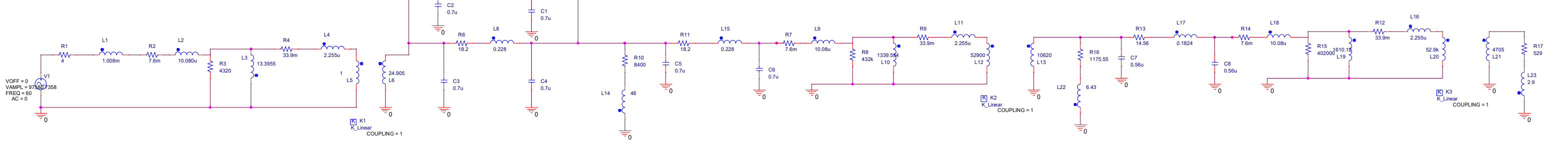
5 Referências

GLOVER, J. Duncan; OVERBYE, Thomas J; SARMA, S. Mulukutla. **Power System Analysis and Design**. 6th edition. Printed in the United States of America. CENGAGE Learning, 2015.

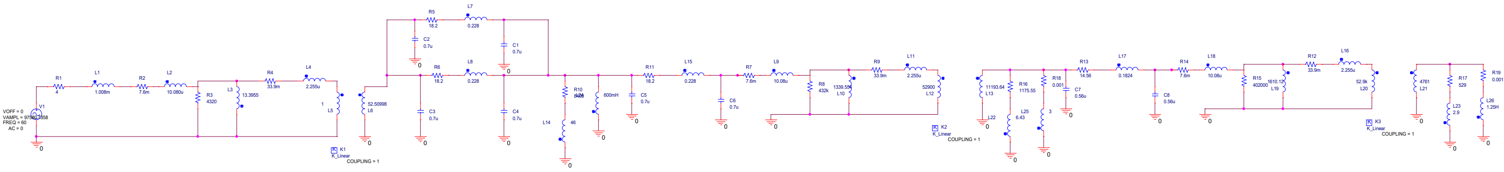
6 Anexos



VOFF = 0
VAMPL = 97580.7358
FREQ = 60
AC = 0
V1
R1 4
L1 1.008m
R2 7.6m
L2 10.080u
R3 4320
L3 13.3955
R4 33.9m
L4 2.255u
1 L5
52.50998 L6
K1 K_Linear COUPLING = 1
R5 18.2
0.228
C2 0.7u
C1 0.7u
R6 18.2
L8 0.228
C3 0.7u
C4 0.7u
R10 8400
L14 46
R11 18.2
C5 0.7u
L15 0.228
C6 0.7u
R7 7.6m
L9 10.08u
R8 432k
1339.554 L10
R9 33.9m
L11 2.255u
52900 L12
K2 K_Linear COUPLING = 1
11193.64 L13
R16 1175.55
L22 6.43
C7 0.56u
R13 14.56
L17 0.1824
C8 0.56u
R14 7.6m
L18 10.08u
R15 402000
1610.12 L19
R12 33.9m
L16 2.255u
52.9k L20
K3 K_Linear COUPLING = 1
4761 L21
R17 529
L23 2.9
0

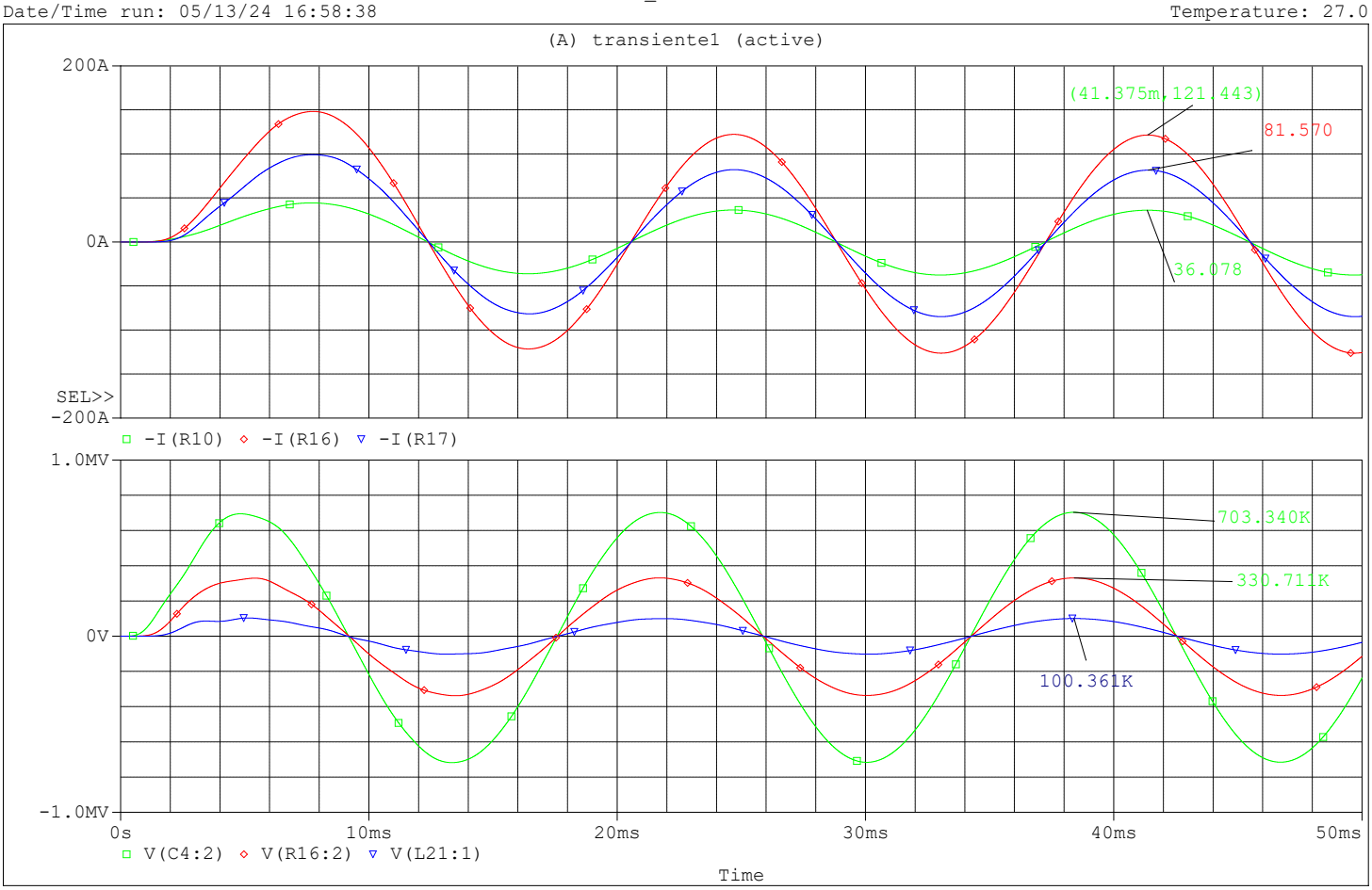


V1
VOFF = 0
VAMPL = 97580.7358
FREQ = 60
AC = 0
R1
4
L1
1.008m
R2
7.6m
L2
10.080u
R3
4320
L3
13.3955
R4
33.9m
L4
2.255u
L5
1
L6
24.905
K1
K_Linear
COUPLING = 1
R5
18.2
L7
0.228
C2
0.7u
C1
0.7u
R6
18.2
L8
0.228
C3
0.7u
C4
0.7u
R10
8400
L14
46
C5
0.7u
R11
18.2
L15
0.228
C6
0.7u
R7
7.6m
L9
10.08u
R8
432k
1339.554
L10
R9
33.9m
L11
2.255u
52900
L12
K2
K_Linear
COUPLING = 1
10620
L13
R16
1175.55
L22
6.43
C7
0.56u
R13
14.56
L17
0.1824
C8
0.56u
R14
7.6m
L18
10.08u
R15
402000
1610.12
L19
R12
33.9m
L16
2.255u
52.9k
L20
K3
K_Linear
COUPLING = 1
4705
L21
R17
529
L23
2.9
0

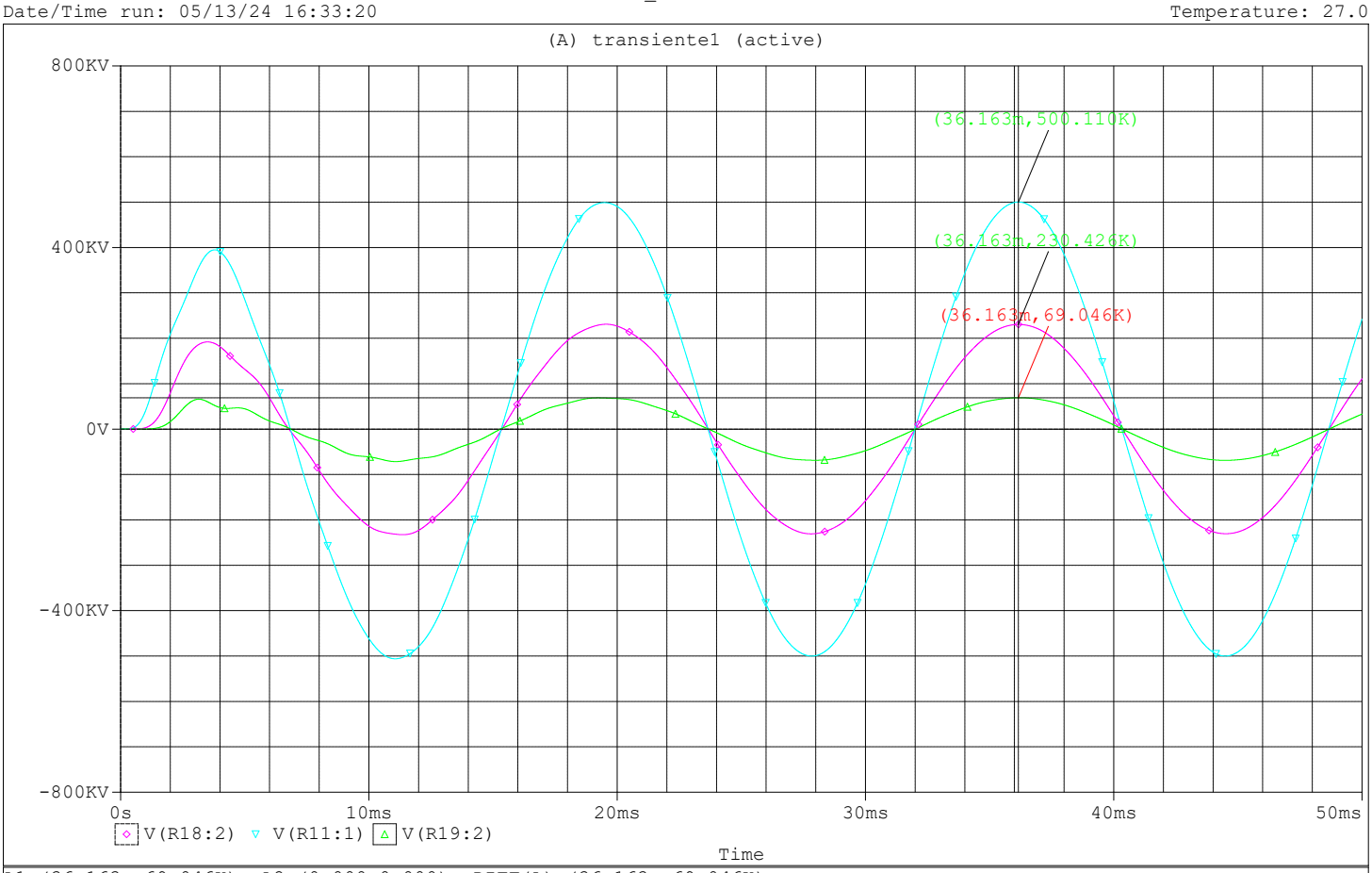


VOFF = 0
FREQ = 60
AC = 0
COUPLING = 1
K_Linear
COUPLING = 1
COUPLING = 1

** Profile: "SCHEMATIC1-transiente1" [c:\cadence\spb_17.2\tools\capture\1\projetoquadripolos-PSpiceFiles\SCHEM...
Date/Time run: 05/13/24 16:58:38 Temperature: 27.0



** Profile: "SCHEMATIC1-transiente1" [c:\cadence\spb_17.2\tools\capture\1\projetoquadripolos-PSpiceFiles\SCHEM...
Date/Time run: 05/13/24 16:33:20 Temperature: 27.0



A1:(36.163m,69.046K) A2:(0.000,0.000) DIFF(A):(36.163m,69.046K)

** Profile: "SCHEMATIC1-transiente1" [c:\cadence\spb_17.2\tools\capture\1\projetoquadripolos-pspicefiles\schem...
Date/Time run: 05/13/24 22:52:41 Temperature: 27.0