```
begin
    using LinearAlgebra
    using Statistics
    using Printf
end
```

Montagem da Matriz de Impedâncias Primitiva

```
D = 4×4 Matrix{Float64}:
 0.0244  2.5     7.0     5.6569
 2.5     0.0244  4.5     4.272
 7.0     4.5     0.0244  5.0
 5.6569  4.272   5.0     0.0081
```

```
D = [0.0244 2.5 7.0 5.6569;
     2.5 0.0244 4.5 4.272;
     7.0 4.5 0.0244 5.0;
     5.6569 4.272 5.0 0.0081]
```

```
begin
    Zabc = zeros(Complex, 4, 4)
    for i in 1:4
        for j in 1:4
            if i != j
                Zabc[i, j] = 0.0953 + 0.12134 * (log(1.0 / D[i, j]) +
                7.93402)im
            elseif i == 4
                Zabc[i, j] = 0.592 + 0.0953 + 0.12134 * (log(1.0 / D[i,
                j]) + 7.93402)im
            else
                Zabc[i, j] = 0.306 + 0.0953 + 0.12134 * (log(1.0 / D[i,
                j]) + 7.93402)im
            end
        end
    end
end
```

```
4×4 Matrix{Complex}:
 0.4013+1.41327im   0.0953+0.851531im  0.0953+0.726597im  0.0953+0.752447im
 0.0953+0.851531im  0.4013+1.41327im   0.0953+0.780209im  0.0953+0.786518im
 0.0953+0.726597im  0.0953+0.780209im  0.4013+1.41327im   0.0953+0.767425im
 0.0953+0.752447im  0.0953+0.786518im  0.0953+0.767425im  0.6873+1.54707im
```

```
Zabc
```

Conversão da matriz de impedâncias primitiva de Ohms por milha para Ohms por metro:

```
4×4 Matrix{ComplexF64}:
 0.000249356+0.000878165im   5.92167e-5+0.000529117im  …   5.92167e-5+0.00046
  5.92167e-5+0.000529117im  0.000249356+0.000878165im      5.92167e-5+0.00048
  5.92167e-5+0.000451486im   5.92167e-5+0.000484799im      5.92167e-5+0.00047
  5.92167e-5+0.000467549im   5.92167e-5+0.00048872im      0.000427068+0.00096
```

```
1 Zabc * 0.000621371
```

Conversão da matriz de impedâncias primitiva de Ohms por milha para Ohms por quilômetro:

```
4×4 Matrix{ComplexF64}:
  0.249356+0.878165im  0.0592167+0.529117im  …  0.0592167+0.467549im
 0.0592167+0.529117im   0.249356+0.878165im     0.0592167+0.48872im
 0.0592167+0.451486im  0.0592167+0.484799im     0.0592167+0.476856im
 0.0592167+0.467549im  0.0592167+0.48872im       0.427068+0.961307im
```

```
1 Zabc * 0.621371
```

Montagem da Matriz de Impedâncias de Fase (Redução de Kron)

Redução de Kron da matriz de impedâncias primitiva

```
Zabck = 3×3 Matrix{ComplexF64}:
        0.457485+1.07814im    0.15588+0.501778im  0.153417+0.385036im
         0.15588+0.501778im  0.466554+1.04827im   0.157935+0.423754im
        0.153417+0.385036im  0.157935+0.423754im  0.461403+1.06516im
```

```
1 Zabck = Zabc[1:3, 1:3] - Zabc[1:3, 4:4] * Zabc[4:4, 1:3] / Zabc[4, 4]
```

Conversão da matriz de impedâncias de fase de Ohms por milha para Ohms por metro:

```
3×3 Matrix{ComplexF64}:
 0.000284268+0.000669922im   9.68594e-5+0.00031179im    9.53288e-5+0.00023925
  9.68594e-5+0.00031179im   0.000289903+0.000651366im   9.81362e-5+0.00026330
  9.53288e-5+0.00023925im    9.81362e-5+0.000263308im  0.000286702+0.00066186
```

```
1 Zabck * 0.000621371
```

Conversão da matriz de impedâncias de fase de Ohms por milha para Ohms por quilômetro:

```
3×3 Matrix{ComplexF64}:
  0.284268+0.669922im  0.0968594+0.31179im   0.0953288+0.23925im
 0.0968594+0.31179im    0.289903+0.651366im  0.0981362+0.263308im
 0.0953288+0.23925im   0.0981362+0.263308im   0.286702+0.661861im
```

```
Zabck * 0.621371
```

```
l = 1.893939393939394
```

```
l = 10e3 / 5280.0
```

Matriz de Impedâncias Primitiva em Ohms:

```
Zabc_ =
4×4 Matrix{ComplexF64}:
 0.760038+2.67665im  0.180492+1.61275im  0.180492+1.37613im  0.180492+1.42509
 0.180492+1.61275im  0.760038+2.67665im  0.180492+1.47767im  0.180492+1.48962
 0.180492+1.37613im  0.180492+1.47767im  0.760038+2.67665im  0.180492+1.45346
 0.180492+1.42509im  0.180492+1.48962im  0.180492+1.45346im    1.3017+2.93006
```

```
Zabc_ = Zabc * l
```

Matriz de Impedâncias de Fase Reduzida de Kron em Ohms:

```
Zabck_ = 3×3 Matrix{ComplexF64}:
          0.866448+2.04192im   0.295228+0.950337im  0.290562+0.729235im
          0.295228+0.950337im  0.883625+1.98536im   0.299119+0.802565im
          0.290562+0.729235im  0.299119+0.802565im  0.873869+2.01735im
```

```
Zabck_ = Zabck * l
```

Montagem da Matriz de Impedâncias de Sequência

```
md"""
## Montagem da Matriz de Impedâncias de Sequência
"""
```

```
as_ = -0.4999999999999998 + 0.8660254037844387im
```

```
as_ = p(1.0, 120.0)
```

```
As = 3×3 Matrix{ComplexF64}:
      1.0+0.0im   1.0+0.0im         1.0+0.0im
      1.0+0.0im  -0.5-0.866025im   -0.5+0.866025im
      1.0+0.0im  -0.5+0.866025im   -0.5-0.866025im
```

```
As = [1.0 1.0 1.0; 1.0 as_^2 as_; 1.0 as_ as_^2]
```

```
Z012 =
3×3 Matrix{ComplexF64}:
   0.773302+1.93757im    0.0255647+0.0114925im   -0.0320849+0.0158887im
 -0.0320849+0.0158887im    0.30607+0.627im       -0.0722504-0.00602722im
  0.0255647+0.0114925im  0.0723029-0.00589754im     0.30607+0.627im
```

```
Z012 = inv(As) * Zabck *  As
```

```
3×3 Matrix{ComplexF64}:
 0.461814+1.06386im   0.155059+0.462864im  0.155059+0.462864im
 0.155059+0.462864im  0.461814+1.06386im   0.155059+0.462864im
 0.155059+0.462864im  0.155059+0.462864im  0.461814+1.06386im
```

```
begin
    zs = (Zabck[1, 1] + Zabck[2, 2] + Zabck[3, 3]) / 3.0
    zm  = (Zabck[1, 2] + Zabck[1, 3] + Zabck[2, 1]) / 3.0
    Zabcks = [zs zm zm; zm zs zm; zm zm zs]
end
```

```
Z012s =
3×3 Matrix{ComplexF64}:
     0.771932+1.98958im      -1.11022e-16+8.32667e-17im  -2.77556e-16+0.0im
 -1.38778e-17+2.77556e-17im      0.306755+0.600992im     -1.11022e-16-1.38778e
          0.0+5.55112e-17im   6.93889e-17-2.77556e-17im      0.306755+0.60099
```

```
Z012s = inv(As) * Zabcks *  As
```

Cálculo das Matrizes características da Linha

```
md"""
## Cálculo das Matrizes características da Linha
"""
```

```
a = UniformScaling{Bool}
    true*I
```

```
a = I
```

```
b = 3×3 Matrix{ComplexF64}:
     0.866448+2.04192im   0.295228+0.950337im  0.290562+0.729235im
     0.295228+0.950337im  0.883625+1.98536im   0.299119+0.802565im
     0.290562+0.729235im  0.299119+0.802565im  0.873869+2.01735im
```

```
b = Zabck_
```

```
c = 3×3 Matrix{Complex}:
     0+0im  0+0im  0+0im
     0+0im  0+0im  0+0im
     0+0im  0+0im  0+0im
```

```
c = zeros(Complex, 3, 3)
```

```
d = UniformScaling{Bool}
    true*I
```

```
1 d = a
```

```
A = UniformScaling{Float64}
    1.0*I
```

```
1 A = inv(a)
```

```
B = 3×3 Matrix{ComplexF64}:
     0.866448+2.04192im   0.295228+0.950337im  0.290562+0.729235im
     0.295228+0.950337im  0.883625+1.98536im   0.299119+0.802565im
     0.290562+0.729235im  0.299119+0.802565im  0.873869+2.01735im
```

```
1 B = A * b
```

Execução do Fluxo de carga usando Matriz Reduzida de Kron

Definição de duas funções úteis:

- p(m, a)
- dv(v)

```
p (generic function with 1 method)
```

```
p(m, a) = m * cis(deg2rad(a))
```

```
dv (generic function with 1 method)
```

```
function dv(v)
    for i in v
        m = abs(i)
        a = rad2deg(angle(i))
        @printf "%.2f ∠%.2f°\n" m a
    end
end
```

```
va = 7199.557856794634
```

```
va = 12.47e3 / √3
```

```
Vabc1_ = 3×1 Matrix{ComplexF64}:
          7199.557856794634 + 0.0im
         -3599.778928397315 - 6235.000000000001im
         -3599.778928397315 + 6235.000000000001im
```

```
Vabc1_ = [p(va, 0.0); p(va, -120.0); p(va, 120.0);;]
```

```
dv(Vabc1_)
```

```
7199.56 ∠0.00°
7199.56 ∠-120.00°
7199.56 ∠120.00°
```

```
Sabc = [2.25e6+1.08972e6im, 1.7e6+1.05357e6im, 1.425e6+4.68375e5im]
```

```
Sabc = [p(2.5e6, acosd(0.9)); p(2.0e6, acosd(0.85)); p(1.5e6,
acosd(0.95))]
```

```
dv(Sabc)
```

```
2500000.00 ∠25.84º
2000000.00 ∠31.79º
1500000.00 ∠18.19º
```

Execução de Fluxo de Carga utilizando método de varredura Direta-Inversa

Varredura Inversa (<—):

$$I_{abc}^{(n)} = \mathbf{c}\, V_{abc}^{(m)} + \mathbf{d}\, I_{abc}^{(m)}$$

Varredura Direta (—>):

$$V_{abc}^{(m)} = \mathbf{A}\, V_{abc}^{(n)} - \mathbf{B}\, I_{abc}^{(m)}$$

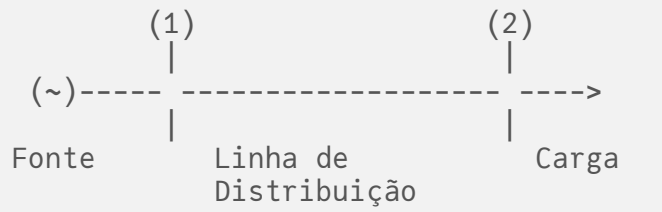
Em que:

$$\mathbf{A} = \mathbf{a}^{-1}$$

e

$$\mathbf{B} = \mathbf{a}^{-1}\mathbf{b}$$

Algoritmo de fluxo de carga de varredura direta inversa:



```
begin
    Vabc2 = Vabc1_
    n = 0
    while n < 10
        global Vabc1, Iabc2, Vabc2, n
        n = n + 1
        Iabc2 = conj.(Sabc ./ Vabc2)

        # Backward Sweep
        Vabc1 = a * Vabc2 + b * Iabc2
        Iabc1 = c * Vabc2 + d * Iabc2

        error = maximum(abs.((Vabc1 - Vabc1_) ./ Vabc1_))
        @printf "Iteration %i absolute error = %.4f\n" n error
        if error < 1e-5
            break
        end

        # Forward Sweep
        Vabc1 = Vabc1_
        Vabc2 = A * Vabc1 - B * Iabc2
    end
end
```

```
Iteration 1 absolute error = 0.0750
Iteration 2 absolute error = 0.0070
Iteration 3 absolute error = 0.0008
Iteration 4 absolute error = 0.0001
Iteration 5 absolute error = 0.0000
Iteration 6 absolute error = 0.0000
```

Tensões na carga:

```
dv(Vabc2)
```

```
6678.23 ∠-2.33º
6972.79 ∠-122.13º
7055.48 ∠118.74º
```

Correntes na linha:

```
dv(Iabc2)
```

```
374.35 ∠-28.17º
286.83 ∠-153.92º
212.60 ∠100.55º
```

Tensões na fonte:

```
dv(Vabc1)
```

```
7199.57 ∠0.00º
7199.55 ∠-120.00º
7199.56 ∠120.00º
```

Razão entre as tensões na carga e na fonte:

$$V_{abc}^{(2)} / V_{abc}^{(1)}$$

```
3×1 Matrix{Float64}:
 0.9275880434454953
 0.96850320272011
 0.9799878879471249
```

```
abs.(Vabc2) ./ abs.(Vabc1)
```

Corrente de desequilíbrio na carga:

```
99.60201289042335
```

```
abs(sum(Iabc2))
```

Corrente no condutor neutro:

```
1×1 Matrix{ComplexF64}:
 -21.68875782114493 + 38.806367750097934im
```

```
begin
    tn = - Zabc[4:4, 1:3] / Zabc[4, 4]
    In2 = tn * Iabc2
end
```

```
dv(In2)
```

```
44.46 ∠119.20º
```

Corrente de retorno pela terra:

```
Ig2 = 1×1 Matrix{ComplexF64}:
       -11.779713448755171 + 55.00419883320296im
```

```
Ig2 = -[sum(Iabc2)] - In2
```

```
dv(Ig2)
```

```
56.25 ∠102.09º
```

```
dv(In2 + Ig2)
```

```
99.60 ∠109.63º
```

Fluxo de carga com regulador de tensão na fonte

Relação de transformação do TP:

```
NPT = 60.0
```

```
NPT = 7200.0 / 120
```

Relação de transformação do TC:

```
120.0
```

```
begin
    CTp = 600.0
    CTs = 5.0
    CT = CTp/CTs
end
```

As tensões encontradas no fluxo de carga sem regulador de tensão na base de 120V serão:

```
dv(Vabc2/NPT)
```

```
111.30 ∠-2.33º
116.21 ∠-122.13º
117.59 ∠118.74º
```

```
VL = 120.0
```

```
VL = 120.0 # voltage level
```

```
BW = 2.0
```

```
BW = 2.0 # bandwidth
```

```
V_target = 119.0
```

```
V_target = VL - BW/2.0
```

Impedância da linha em Ω:

```
Zline = 3×1 Matrix{ComplexF64}:
          0.8988809626273322 + 1.3026207619397003im
         0.16568491838114568 + 1.2004972145691717im
          0.4044224436921688 + 0.9138985432761497im
```

```
1 Zline = (Vabc1 - Vabc2) ./ Iabc2
```

```
1 dv(Zline)
```

```
1.58 ∠55.39º
1.21 ∠82.14º
1.00 ∠66.13º
```

Impedância média:

```
Zline_avg = 0.4896627749002156 + 1.1390055065950071im
```

```
1 Zline_avg = mean(Zline)
```

Cálculo do ajuste do **compensador de linha** em *Volts*:

```
ZLC_volts = 4.896627749002156 + 11.390055065950071im
```

```
1 ZLC_volts = Zline_avg * CTp / NPT
```

```
taps = 3×1 Matrix{Float64}:
       10.0
        4.0
        2.0
```

```
1 taps = round.((V_target .- abs.(Vabc2)/NPT) / 0.75)
```

Por que a divisão por 0,75?

Se considerarmos a base de 120V e um ajuste de +/- 10% na tensão em 16 passos, teremos a seguinte razão:

$$\frac{0,1 \times 120}{16} = \frac{12}{16} = 0,75$$

Definição das matrizes características do regulador de tensão:

```
areg = 3×3 Matrix{Float64}:
        0.9375  -0.0    -0.0
       -0.0      0.975  -0.0
       -0.0     -0.0     0.9875
```

```
1 areg = I - 0.00625 * diagm(vec(taps))
```

```
breg = 3×3 Matrix{Float64}:
       0.0  0.0  0.0
       0.0  0.0  0.0
       0.0  0.0  0.0
```

```
breg = creg = zeros(3, 3)
```

```
dreg = 3×3 Matrix{Float64}:
       1.06667  0.0      0.0
       0.0      1.02564  0.0
       0.0      0.0      1.01266
```

```
dreg = Areg = inv(areg)
```

```
Breg = 3×3 Matrix{Float64}:
       0.0  0.0  0.0
       0.0  0.0  0.0
       0.0  0.0  0.0
```

```
Breg = zeros(3, 3)
```

```
dv(Vabc1_)
```

```
7199.56 ∠0.00º
7199.56 ∠-120.00º
7199.56 ∠120.00º
```

Algoritmo de fluxo de carga de varredura direta inversa com inclusão do regulador de tensão:

```
begin
    Vabc3k = Vabc1_
    Vabc2ar = Vabc1_
    nk = 0

    while nk < 100
        global Vabc1k, Iabc1k, Vabc2ar, Iabc2ar, Vabc3k, Iabc3k, nk
        nk = nk + 1
        Iabc3k = conj.(Sabc ./ Vabc3k)

        # Backward Sweep
        Vabc2ar = a * Vabc3k + b * Iabc3k
        Iabc2ar = c * Vabc3k + d * Iabc3k

        Vabc1k = areg * Vabc2ar + breg * Iabc2ar
        Iabc1k = creg * Vabc2ar + dreg * Iabc2ar


        error = maximum(abs.((Vabc1k - Vabc1_) ./ Vabc1_))
        @printf "Iteration %i absolute error = %.4f\n" nk error
        if error < 1e-5
            break
        end

        # Forward Sweep
        Vabc1k = Vabc1_
        Vabc2ar = Areg * Vabc1k - Breg * Iabc2ar
        Vabc3k = A * Vabc2ar - B * Iabc3k
    end
end
```

```
Iteration 1 absolute error = 0.0346
Iteration 2 absolute error = 0.0030
Iteration 3 absolute error = 0.0002
Iteration 4 absolute error = 0.0000
Iteration 5 absolute error = 0.0000
```

```
dv(Vabc3k / NPT)
```

```
120.09 ∠-1.94º
119.10 ∠-121.97º
119.12 ∠118.65º
```

```
dv(Vabc1_)
```

```
7199.56 ∠0.00º
7199.56 ∠-120.00º
7199.56 ∠120.00º
```

```
dv(Iabc3k)
```

```
346.96 ∠-27.78º
279.88 ∠-153.76º
209.87 ∠100.46º
```

```
dv(Iabc1k)
```

```
370.09 ∠-27.78º
287.06 ∠-153.76º
212.53 ∠100.46º
```

Importante!

Note que em nenhum momento o valor da **impedância do compensador de linha** foi utilizado neste exemplo!

Esse fato gera uma **imprecisão nos cálculos**, conforme demonstrado abaixo.

```
Zc = 0.9793255498004312 + 2.2780110131900014im
```

```
Zc = ZLC_volts / CTs
```

```
V_reg_out = 3×1 Matrix{ComplexF64}:
             7199.557856794634 + 0.0im
            -3599.778928397315 - 6235.000000000001im
            -3599.778928397315 + 6235.000000000001im
```

```
# Para os taps do regulador inicialmente na posição 0
# areg = I
V_reg_out = I * Vabc1_
```

```
dv(V_reg_out)
```

```
7199.56 ∠0.00º
7199.56 ∠-120.00º
7199.56 ∠120.00º
```

```
I_reg_out = 3×1 Matrix{ComplexF64}:
             330.01537833786955 - 176.71493673892792im
            -257.6220938666228 - 126.10266822676691im
            -38.92481320134667 + 209.00703838239394im
```

```
# Para os taps do regulador inicialmente na posição 0
# dreg = inv(I)
I_reg_out = inv(I) * Iabc2
```

```
dv(I_reg_out)
```

```
374.35 ∠-28.17º
286.83 ∠-153.92º
212.60 ∠100.55º
```

```
V_reg_s = 3×1 Matrix{ComplexF64}:
          119.99263094657724 + 0.0im
          -59.99631547328859 - 103.91666666666669im
          -59.99631547328859 + 103.91666666666669im
```

```
V_reg_s = V_reg_out / NPT
```

```
dv(V_reg_s)
```

```
119.99 ∠0.00º
119.99 ∠-120.00º
119.99 ∠120.00º
```

```
I_reg_s = 3×1 Matrix{ComplexF64}:
           2.7501281528155794 - 1.4726244728243993im
          -2.1468507822218563 - 1.050855568556391im
          -0.3243734433445556 + 1.7417253198532 83im
```

```
I_reg_s = I_reg_out / CT
```

```
dv(I_reg_s)
```

```
3.12 ∠-28.17º
2.39 ∠-153.92º
1.77 ∠100.55º
```

```
Zcomp_matrix = 3×3 Diagonal{ComplexF64, Vector{ComplexF64}}:
                0.979326+2.27801im                   ⋅                          ⋅
                         ⋅                  0.979326+2.27801im                 ⋅
                         ⋅                           ⋅                 0.979326+2.27801im
```

```
Zcomp_matrix = Zc * I(3)
```

```
V_relay = 3×1 Matrix{ComplexF64}:
          113.94470541391236 - 4.822643448299475im
           -60.2877102090932 - 97.99698723365236im
          -55.71097881196685 + 102.9498768365253im
```

```
V_relay = V_reg_s - Zcomp_matrix * I_reg_s
```

```
dv(V_relay)
```

```
114.05 ∠-2.42º
115.06 ∠-121.60º
117.06 ∠118.42º
```

```
taps_ = 3×1 Matrix{Float64}:
        7.0
        5.0
        3.0
```

```
taps_ = round.((V_target .- abs.(V_relay)) / 0.75)
```

Importante!

Veja como esses valores de tap são diferentes dos valores utilizados acima para o cálculo do fluxo de carga!

```
# TODO: Executar fluxo de carga com os valores de tap
# do regulador de tensão presnetes na variável taps_
```

Cálculo de fluxo de carga com análise dos taps um por vez

```
begin

    taps_h = [0; 0; 0]

    h = 0
    while h < 16
        global Vabc1_h, Vabc2ar_h, Vabc3_h, h, nh, taps_h
        h = h + 1

        areg_h = I - 0.00625 * diagm(vec(taps_h))
        breg_h = creg_h = zeros(3, 3)
        dreg_h = Areg_h = inv(areg_h)
        Breg_h = zeros(3, 3)

        Vabc3_h = Vabc1_
        Vabc2ar_h = Vabc1_
        Vabc1_h = Vabc1_

        nh = 0

        while nh < 15
            global Vabc1_h, Vabc2ar_h, Vabc3_h, Iabc1_h, Iabc2ar_h,
            Iabc3_h, nh
            nh = nh + 1
            Iabc3_h = conj.(Sabc ./ Vabc3_h)

            # Backward Sweep
            Vabc2ar_h = a * Vabc3_h + b * Iabc3_h
            Iabc2ar_h = c * Vabc3_h + d * Iabc3_h

            Vabc1_h = areg_h * Vabc2ar_h + breg_h * Iabc2ar_h
            Iabc1_h = creg_h * Vabc2ar_h + dreg_h * Iabc2ar_h


            error = maximum(abs.((Vabc1_h - Vabc1_) ./ Vabc1_))
            @printf "Iteration %i absolute error = %.4f\n" nh error
            if error < 1e-5
                break
            end

            # Forward Sweep
            Vabc1_h = Vabc1_
            Vabc2ar_h = Areg_h * Vabc1_h - Breg_h * Iabc2ar_h
            Vabc3_h = A * Vabc2ar_h - B * Iabc3_h
        end

        # Cálculo da tensão do relé de atuação do LDC no
        # circuito de comando do Regulador de Tensão
        I_reg_out_h = Iabc2ar_h

        I_reg_s_h = I_reg_out_h / CT
        V_reg_s_h  = Vabc2ar_h / NPT
```

```
        V_relay_h = V_reg_s_h - Zcomp_matrix * I_reg_s_h

        if sum(abs.(V_relay_h) .< V_target) == 0.0
            println("Taps: ", taps_h)
            dv(V_relay_h)
            break
        else
            println("Taps: ", taps_h)
            taps_h = taps_h + (abs.(V_relay_h) .< V_target)
            dv(V_relay_h)
        end
    end
end
```

```
Iteration 1 absolute error = 0.0750
Iteration 2 absolute error = 0.0070
Iteration 3 absolute error = 0.0008
Iteration 4 absolute error = 0.0001
Iteration 5 absolute error = 0.0000
Iteration 6 absolute error = 0.0000
Taps: [0, 0, 0]
114.05 ∠-2.42º
115.06 ∠-121.60º
117.06 ∠118.42º
Iteration 1 absolute error = 0.0692
Iteration 2 absolute error = 0.0064
Iteration 3 absolute error = 0.0007
Iteration 4 absolute error = 0.0001
Iteration 5 absolute error = 0.0000
Iteration 6 absolute error = 0.0000
Taps: [1; 1; 1;;]
114.85 ∠-2.39º
115.85 ∠-121.58º
117.83 ∠118.44º
Iteration 1 absolute error = 0.0635
Iteration 2 absolute error = 0.0059
Iteration 3 absolute error = 0.0006
Iteration 4 absolute error = 0.0001
Iteration 5 absolute error = 0.0000
Taps: [2; 2; 2;;]
115.66 ∠-2.36º
116.64 ∠-121.56º
118.61 ∠118.46º
Iteration 1 absolute error = 0.0580
Iteration 2 absolute error = 0.0053
Iteration 3 absolute error = 0.0006
```

```
3×1 Matrix{Int64}:
 7
 5
 3
```

```
taps_h
```

Tensão na carga após a atuação do regulador

Para o ajuste implementado, em decorrência da situação de carga elevada na fase A, a tensão nesta fase ficará fora da faixa de ajuste, conforme exibido abaixo:

```
dv(Vabc3_h/NPT)
```

```
117.33 ∠-2.08º
120.09 ∠-121.96º
119.89 ∠118.73º
```

Dica do autor para ajuste do regulador de tensão

Because the three line currents are all different, it means the heavily loaded phase (a) voltage will not represent what is actually happening on the system. Once again, this is a problem that occurs because of the unbalanced loading.

One way to raise the load voltages is: **to specify a higher voltage level by increasing the voltage level to 122 V**.