```julia
begin
    using LinearAlgebra
    using Printf
end
```

```
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
```

```julia
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]
```

```julia
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
```

```julia
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
```

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
```

```
1 Zabck * 0.621371
```

```
l = 1.893939393939394
```

```
1 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:
```

```julia
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
```

```julia
Zabck_ = Zabck * l
```

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

```julia
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
```

```julia
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
```

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

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

```julia
d = a
```

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

```julia
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
```

```julia
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)
```

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

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

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

```
va = 7199.557856794634
```

```julia
va = 12.47e3 / √3
```

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

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

```julia
dv(Vabc1_)
```

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

```
s = 2.0e6
```

```julia
s = 2.0e6
```

```
Sabc = 3×1 Matrix{ComplexF64}:
        1.8e6 + 871779.7887081346im
        1.8e6 + 871779.7887081346im
        1.8e6 + 871779.7887081346im
```

```julia
Sabc = fill(p(s, acosd(0.9)), (3, 1))
```

```julia
dv(Sabc)
```

```
2000000.00 ∠25.84º
2000000.00 ∠25.84º
2000000.00 ∠25.84º
```

```julia
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 = 0.0548
Iteration 2 absolute error = 0.0033
Iteration 3 absolute error = 0.0002
Iteration 4 absolute error = 0.0000
Iteration 5 absolute error = 0.0000
```

Tensões na carga:

```julia
dv(Vabc2)
```

```
6841.46 ∠-1.77º
6943.18 ∠-121.81º
6898.44 ∠117.93º
```

Correntes na linha:

```julia
dv(Iabc2)
```

```
292.34 ∠-27.61º
288.05 ∠-147.66º
289.92 ∠92.09º
```

Tensões na fonte:

```julia
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}:
 0.9502608053767477
 0.9643901071147718
 0.9581749639213127
```

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

Corrente de desequilíbrio na carga:

```
5.123601929630065
```

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

Corrente no condutor neutro:

```
1×1 Matrix{ComplexF64}:
 2.162565114902449 + 1.8007282766724209im
```

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

```
dv(In2)
```

```
2.81 ∠39.78º
```

Corrente de retorno pela terra:

```
md"Corrente de retorno pela terra:"
```

```
Ig2 = 1×1 Matrix{ComplexF64}:
       -7.284453995566237 - 1.9332085674389439im
```

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

```
dv(Ig2)
```

```
7.54 ∠-165.14º
```

Corrente de Desequilíbrio total:

```
md"Corrente de Desequilíbrio total:"
```

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

```
5.12 ∠-178.52º
```