Universidade Estadual Paulista "Júlio de Mesquita Filho"

Programa de Pós-Graduação em Engenharia Elétrica

Trabalho I:

Simulação de Fluxos de Potência (FP) de Redes de Energia e Resolução de Sistema Não-Linear com Método de Newton

Disciplina: Despacho e Pré-Despacho de Geração

Discente: Rafael Pavan

Docente: Prof. Dr. Leonardo Nepomuceno

Bauru, 2020

SUMÁRIO

1. INTRODUÇÃO

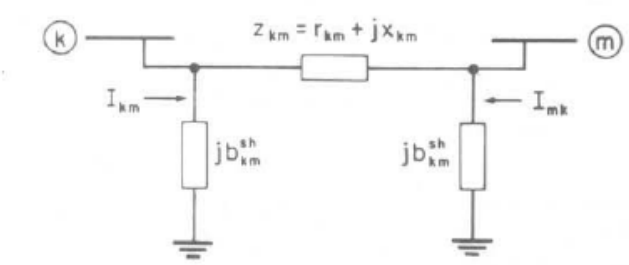
A problemática do fluxo de potência em uma rede de energia consiste em determinar o estado das grandezas que constituem este sistema, tais como: potências ativas e reativas, tensões, ângulos de fase e os parâmetros físicos das linhas como admitâncias e susceptâncias. A modelagem deste sistema tende a ser do tipo estática, onde a rede é representada por um conjunto de equações e inequações algébricas não-lineares que correspondem às leis de Kirchhoff. O cálculo do fluxo de carga é desenvolvido a partir de aplicações computacionais para uma determinada rede.

Um sistema é constituído de barras, que são pontos onde estão conectados subestações de geração ou de carga. As barras podem ser divididas em três tipos:

- Barras de Carga (PQ): barras onde são conhecidas as potências ativas e reativas.
- Barras de Geração (PV): barras onde são conhecidos os valores de potência ativa e módulo da tensão.
- Barras de Referência (Vθ): barras onde são conhecidas as referências de magnitude de tensão e angular.

A linha de transmissão do sistema é modelada através do modelo π. É composta pelos elementos de admitância série (condutância resistiva e susceptância indutiva) e susceptância shunt capacitiva.

Figura 1 – Modelo π da Linha de Transmissão



Fonte: Monticelli [1]

Aplicando as leis de Kirchhoff nas barras, temos que a corrente que passa por uma linha pode ser decomposta em duas componentes: a série e a shunt. Ambas as correntes são função das tensões das barras das extremidades das linhas e dos parâmetros destas, além dos ângulos de fase. Através das correntes, pode-se determinar os fluxos de potência ativa e reativa que fluem pelo sistema, o que gerará um sistema de equações não-lineares que devem ser resolvidas computacionalmente. O restante do trabalho focará na formulação do problema e em sua solução através do método de Newton.

O trabalho está organizado da seguinte maneira: inicialmente foi definida a problemática, e em seguida foi realizado um scrip para cada solução solicitada. Em cada parte é apresentada a resposta encontrada, um embasamento teórico por trás do método e o algoritmo realizado no MATLAB. Ao final, no ANEXO I, encontra-se o algoritmo completo com todas as partes unidas.

2. DEFINIÇÃO DO PROBLEMA

Seja o sistema de 3 barras mostrado a seguir, cujos dados de transmissão são mostrados nas Tabelas 1 e 2, onde todos os dados de barra e de ramos estão em pu:

Figura 2 – Sistema Exemplo de 3 Barras

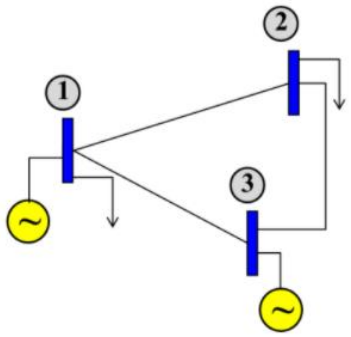


Tabela 1 – Dados de Barra do Sistema da Figura 1

Número	Tipo	V	θ	Pg	Qg	Pc	Qc
1	slack	1.050	0.0	-	-	0.	0
2	PQ	-	-	0	0	0.5	0.2
3	PV	1.000	-	0.3	-	0.	0

Tabela 2 – Dados de Ramo do Sistema da Figura 1

Nó Inicial	Nó Final	r	x	b_{sh}
1	2	0.20	0.40	0.01
1	3	0.20	0.40	0.01
2	3	0.10	0.20	0.01

Para este sistema, deve ser desenvolvido em linguagem MATLAB um trabalho de implementação de fluxos de potência para a solução do sistema teste, nas seguintes etapas:

3. ESCREVER AS EQUAÇÕES DO FLUXO DE POTÊNCIA PARA O PROBLEMA

Através do MATLAB foi criada uma função que monta as equações de potência ativa e reativa de um sistema genérico de N barras, com base nos dados de entrada de: tensão, potência e ângulos de fase de cada barra, além das admitâncias e susceptâncias das linhas. Para o programa realizar o cálculo, há de também ser informado o número de barras total do sistema, bem como discriminar quais são do tipo geração e quais são do tipo carga, além de informar qual é a barra de referência. As equações geradas pelo programa estão descritas abaixo:

- **Equações de Potência Ativa das Barras 2 (Carga) e 3 (Geração):**
    - **Barra 2**

$$\frac{(21*V2*cos(fi2))}{20} - \frac{(21*V2*sin(fi2))}{10} + (2*V2*cos(fi2 - fi3)) - (4*V2*sin(fi2 - fi3)) - (3*V2^2) - \frac{1}{2} = 0$$

    - **Barra 3**

$$\frac{(21*cos(fi3))}{20} - \frac{(21*sin(fi3))}{10} + (2*V2*cos(fi2 - fi3)) + (4*V2*sin(fi2 - fi3)) - \frac{27}{10} = 0$$

- **Equações de Potência Reativa da Barra 2 (Carga):**
    - **Barra 2**

$$\frac{(21*V2*cos(fi2))}{10} + \frac{(21*V2*sin(fi2))}{20} + (4*V2*cos(fi2 - fi3)) + (2*V2*sin(fi2 - fi3)) - \frac{(299*V2^2)}{50} - \frac{1}{5} = 0$$

- **Sustentação Teórica:**

Conforme visto anteriormente através do modelo da linha de transmissão, a corrente pode ser decomposta em duas componentes: série e shunt. Multiplicando-se tal corrente (I) pelo valor de tensão da barra (E), encontramos a expressão da potência aparente que flui pela linha. Vamos considerar duas barras genéricas *k* e *m*. Assim o fluxo de potência (S) entre a barra k e m pode ser dado por:

$$E_k * I_k = S_{km} = P_{km} - jQ_{km}$$

$$S_{km} = y_{km} * V_k e^{-j\theta k}(V_k e^{j\theta_k} - V_m e^{-j\theta_m}) + jb_{km}^{sh} * V_k^{\,2}$$

Separando as equações anteriores em partes reais e imaginárias, encontram-se as expressões para o fluxo de potência reativa (Q) e ativa (P) nas linhas.

$$P_{km} = V_k^{\,2} * g_{km} - V_k * V_m * g_{km} * \cos(\theta_{km}) - V_k * V_m * b_{km} * \sin(\theta_{km})$$

$$Q_{km} = -V_k^{\,2} * (b_{km} + b_{sh}^{km}) + V_k * V_m * b_{km} * \cos(\theta_{km}) + V_k * V_m * g_{km} * \sin(\theta_{km})$$

Para atender ao balanço de potência, tem-se que o somatório das potências nas barras tem que ser igual ao somatório das potências que saem destas barras. Assim, a potência gerada em uma barra, menos a potência consumida nesta e menos a potência que vai desta para as outras barras, tem de ser igual a 0. Assim encontramos as equações de potência ativa e reativa das barras de carga e de potência ativa das barras de geração. As barras de referência não possuem equações. As barras de geração não possuem equações reativas. Logo, tem-se que:

Para barras de Carga e Geração:

$$PG_k - PC_k = \sum_{m \in \Omega_k}^{nbarras} V_k^{\,2} * g_{km} - V_k * V_m * g_{km} * \cos(\theta_{km}) - V_k * V_m * b_{km} * \sin(\theta_{km})$$

Para barras de Carga:

$$QG_k - QC_k = \sum_{m \in \Omega_k} -V_k^{\,2} * (b_{km} + b_{sh}^{km}) + V_k * V_m * b_{km} * \cos(\theta_{km}) - V_k * V_m * g_{km} * \sin(\theta_{km})$$

Assim o número de equações do sistema será dado por:

Total de equações = Nº de Barras de Carga * 2 + Nº de Barras de Geração

- **Código Utilizado Para Gerar as Equações:**

```
% Script para Gerar as Equações de Potência de um Sistema Genérico de
N Barras

% Matriz de Impedâncias Série

Zkm = [0 (0.2 + 0.4i) (0.2 + 0.4i);(0.2 +0.4i) 0 (0.1 + 0.2i);
(0.2+0.4i) (0.1+0.2i) 0 ];
```

```
% Vetor de Susceptâncias Shunt

Bsh = imag([0, 0.01i, 0.01i; 0.01i, 0, 0.01i; 0.01i, 0.01i,0]);

% Calculando a Matriz de Admitâncias

Ykm = 1./Zkm;

% Separando o Vetor de Admitâncias em Dois Vetores: Condutâncias [Gkm]
e
% Susceptâncias [Ykm]

Gkm = real(Ykm);

Bkm = imag(Ykm);

% Criando as Variáveis Simbólicas (Incógnitas)

syms V2 fi2 fi3 Pg1 Qg1 Qg3;

incognitas = [V2, fi2, fi3];

vetordevalores = [1, 1, 1];

% Criando Vetor de Ângulos

fi = [0, fi2, fi3];

% Criando Vetor de Tensões

V = [1.05, V2, 1];

% Criando Vetor de Potências Geradas

Pg = [Pg1, 0, 0.3];
Qg = [Qg1, 0, Qg3];

% Criando Vetor de Potências Consumidas

Pc = [0, 0.5, 0];
Qc = [0, 0.2, 0];

% Define Número Total de Barras

nbarras = [3];
 % Define Quais são Barras de Geração

bger = [3];

% Define Quais são Barras de Carga
```

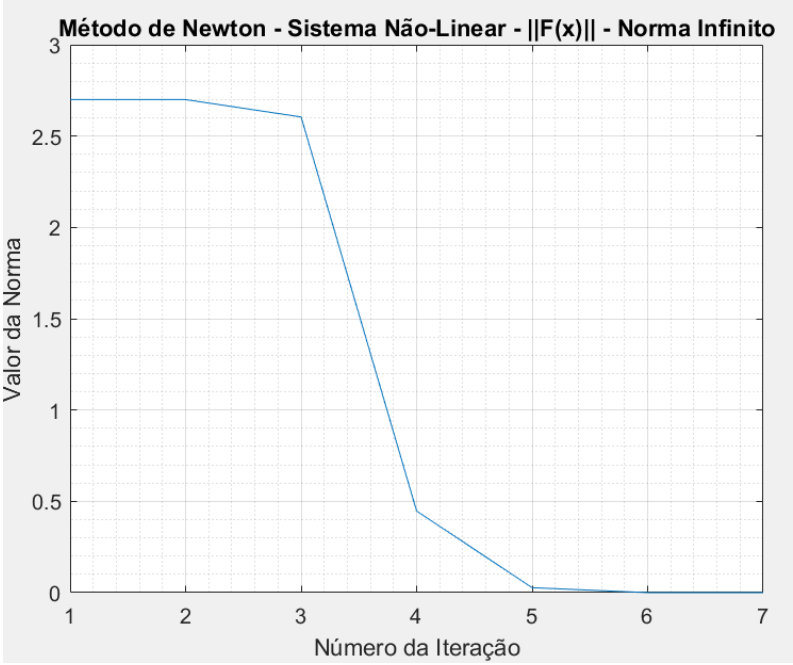
```
bcarga = [2];

% Define Qual a Barra de Referência

bref= [1];

% Inicializa a variável que vai armazenar as equaçãoes

eqqc = sym(zeros(1,numel(bcarga)));
eqpc = sym(zeros(1,numel(bcarga)));
eqpg = sym(zeros(1,numel(bger)));

% Inicializa Variáveis de Controle dos Loops

k = 1;
j=1;
l=1;
m=1;

% Equações Barras de Carga

for j=1:numel(bcarga)

    eqqc(j) = Qg(bcarga(j))-Qc(bcarga(j));
    eqpc(j) = Pg(bcarga(j))-Pc(bcarga(j));

       while (k<nbarras)

            if k == bcarga(j)
                k=k+1;
            end

            eqpc(j) = eqpc(j) + ((-
V(bcarga(j))*V(bcarga(j))*(Gkm(bcarga(j),k)))+(V(bcarga(j))*V(k)*Gkm(b
carga(j),k)*cos(fi(bcarga(j))-
fi(k)))+(V(bcarga(j))*V(k)*Bkm(bcarga(j),k)*sin(fi(bcarga(j))-
fi(k))));
            eqqc(j) = eqqc(j) -((-
V(bcarga(j))*V(bcarga(j))*(Bkm(bcarga(j),k)+Bsh(bcarga(j),k)))+(V(bcar
ga(j))*V(k)*Bkm(bcarga(j),k)*cos(fi(bcarga(j))-fi(k)))-
(V(bcarga(j))*V(k)*Gkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            k=k+1;

       end


end


 % Equações Barras de Geração
```

```
for l=1:numel(bger)

    eqpg(l) = Pg(bger(l))-Pc(bger(l));

      while (m<nbarras)

            if m == bger(l)
                 m=m+1;
            end

        eqpg(l) = eqpg(l) -((V(bger(l))*V(bger(l))*(Gkm(bger(l),m)))-
(V(bger(l))*V(m)*Gkm(bger(l),m)*cos(((fi(bger(l)))-(fi(m)))))-
(V(bger(l))*V(m)*Bkm(bger(l),m)*sin((fi(bger(l))-fi(m)))));
        m=m+1;

      end

end


% Salva as Equações em um Vetor

equations = [eqpc, eqqc, eqpg]
```

4. DESENVOLVER UM PROGRAMA COMPUTACIONAL PARA A SOLUÇÃO DO FLUXO DE POTÊNCIA DO SISTEMA

Com base nas equações montadas anteriormente, também foi criado um *script* que resolve um sistema genérico de N equações não-lineares, utilizando o Método de Newton. Para isso, o programa em questão precisa receber como dados de entrada as equações, as variáveis que são incógnitas, bem como o vetor de valores iniciais. Além de resolver o sistema, o programa também realiza a plotagem de um gráfico informando o valor da norma infinito para cada iteração realizada. Como podemos ver pela Figura 3, o sistema convergiu na 7ª iteração para um valor de norma inferior a 0.00001.

Figura 3 – Valor da Norma para Cada Iteração Realizada



A Solução do Sistema dada pelo Algoritmo foi de:

V2 = 0.954 **fi2** = -0.048 **fi3** = 0.012

- **Sustentação Teórica:**

O método de newton é um algoritmo utilizado para a solução de sistemas não-lineares. Ele funciona da seguinte maneira:

1. Substitui-se um vetor de valores iniciais nas incógnitas do problema, e calcula-se o vetor com o resultado de cada equação.
2. Considera-se o maior valor absoluto entre os resultados como sendo a norma infinito.
3. Verifica-se se a norma infinito atende ao critério para a solução do problema, ou seja, que seja nula.
4. Caso contrário, realiza-se o cálculo da matriz jacobiana (derivadas parciais de primeira ordem das equações do problema).
5. Multiplica-se a inversa negativa da matriz jacobiana pelo vetor de resultados calculado anteriormente, na etapa 1.
6. Soma-se o resultado da etapa 5 ao vetor de valores iniciais da etapa 1, atualizando-o
7. Volta a etapa 1 e repete todos os procedimentos até que o critério de parada seja atingido.

O método tem por objetivo estimar as raízes de uma função, através da derivada desta em um ponto e da intersecção da reta tangente com o eixo da abcissa. O processo é repetido até que seja encontrada a solução. O método é generalizável para sistemas de várias funções, que é o caso do problema proposto.

- **Código Utilizado Para Resolver um Sistema Não-Linear de N Equações pelo Método de Newton:**

```
% Script para Resolver o Sistema Não-Linear Genérico de N Equações
pelo Método de Newton

% Parâmetros necessários: Vetor de Equações, Vetor de Valores Iniciais
e
```

```
% Vetor de Incógnitas

equations = [eqpc, eqqc, eqpg];
incognitas = [V2, fi2, fi3];
vetordevalores = [1, 1, 1];

% Inicializa Variáveis de Controle dos Loops

iteracoes = 1;
t=1;
u=1;

% Realiza Cálculo da Matriz Jacobiana com as Equações

for t = 1:length(equations)

       for u = 1:length(equations)

          J(u,t) = diff(equations(u),incognitas(t));

       end
end

% Substitui o vetor de valores iniciais nas equações

F = vpa(subs(equations, incognitas, vetordevalores));

% Calcula a Norma Infinito

error(iteracoes)=max(abs(F));

% Enquanto a Norma Infinito for Superior ao Critério de Parada
% Estabelecido (0.00001), Realiza o Procedimento Abaixo

while(error(iteracoes)>0.00001)

    iteracoes=iteracoes+1;

    % Calcula a Matriz Jacobiana e Sua Inversa, e Múltiplica por -1

    Jsub = -1*vpa(subs(J, incognitas, vetordevalores));
    IJsub = inv(Jsub);

    % Calcula o Valor da Norma Infinito

    F = vpa(subs(equations, incognitas, vetordevalores));
    error(iteracoes)=max(abs(F));

    % Calcula o Vetor de Acréscimo

    dx=IJsub*transpose(F);
```

```
    % Atualiza o Vetor de Valores Inicial
    vetordevalores=vetordevalores+transpose(dx);
    resultado = vetordevalores;

end

% Realiza a Plotagem do Gráfico com o Valor da Norma Infinito para
Cada
% Iteração

plot(1:iteracoes,error);

title('Método de Newton - Sistema Não-Linear - ||F(x)|| - Norma
Infinito');
xlabel('Número da Iteração');
ylabel('Valor da Norma');
grid on;
grid minor;

% Solução do Sistema

v2=resultado(1)
fi2=resultado(2)
fi3=resultado(3)
```

8. CALCULAR AS TENSÕES FASORIAIS EM CADA BARRA

Através das informações dadas na definição do problema e da solução encontrada pelo método de Newton, tem-se a seguinte configuração de tensão e ângulo de fase em cada barra:

Tabela 3 – Tensões Fasoriais em cada Barra

Barras	Tensões [PU]	Fase [Rad]
1	1.05	0
2	**0.954**	**-0.048**
3	1	**0.012**

9. CALCULAR OS FLUXOS DE POTÊNCIA ATIVA E REATIVA NAS LINHAS

Para se calcular o fluxo de potência, serão utilizadas as equações de fluxo de potência de linhas de transmissão, demonstradas anteriormente na sustentação teórica do item 3.

$$P_{km} = V_k^{\,2} * g_{km} - V_k * V_m * g_{km} * \cos(\theta_{km}) - V_k * V_m * b_{km} * \sin(\theta_{km})$$

$$Q_{km} = -V_k^{\,2} * (b_{km} + b_{sh}^{km}) + V_k * V_m * b_{km} * \cos(\theta_{km}) + V_k * V_m * g_{km} * \sin(\theta_{km})$$

Após implementação de um *script* genérico no MATLAB, foram encontrados os seguintes fluxos:

- **Fluxos de Potência Ativa [PU]**
    - P(1,2) = 0.1979
    - P(1,3) = 0.0271
    - P(2,1) = - 0.1865
    - P(2,3) = - 0.3134
    - P(3,1) = - 0.0245
    - P(3,2) = 0.3245
- **Fluxos de Potência Reativa [PU]**
    - Q(1,2) = 0.1429
    - Q(1,3) = 0.1068
    - Q(2,1) = - 0.1402
    - Q(2,3) = - 0.0597
    - Q(3,1) = - 0.1225
    - Q(3,2) = 0.0627

- **Código Utilizado Para Calcular os Fluxos de Potência Ativa e Reativa Entre Cada Barra**

```
% Script para Calcular o Fluxo de Potência Ativa e Reativa Entre as
Barras
% Necessita dos Vetores de Tensões e Ângulos de Fase como Parâmetros

Peq = sym(zeros(1,nbarras));
Qeq = sym(zeros(1,nbarras));

VN = subs(V, [incognitas(1)],[resultado(1)])
fin = subs(fi,
[incognitas(2),incognitas(3)],[resultado(2),resultado(3)])

y = 1;
c = 1;
d = 1;

for y=1:nbarras

    for d=1:nbarras

         if (y == d)

            Peq(y,d)=0.0;
            Qeq(y,d)=0.0;

         end

     Peq(y,d) = (VN(y)*VN(y))*Gkm(y,d)-
VN(y)*VN(d)*Gkm(y,d)*cos(fin(y)-fin(d)) -
VN(y)*VN(d)*Bkm(y,d)*sin(fin(y)-fin(d));
     Qeq(y,d) = -(VN(y)^2)*(Bkm(y,d)+Bsh(y,d)) +
VN(y)*VN(d)*Bkm(y,d)*cos(fin(y)-fin(d)) -
VN(y)*VN(d)*Gkm(y,d)*sin(fin(y)-fin(d));
     d=d+1;

    end

      y=y+1;

end

fluxoentrelinhas = Peq+Qeq*i;
fluxo12=fluxoentrelinhas(1,2)
fluxo21=fluxoentrelinhas(2,1)
fluxo13=fluxoentrelinhas(1,3)
fluxo31=fluxoentrelinhas(3,1)
fluxo32=fluxoentrelinhas(3,2)
fluxo23=fluxoentrelinhas(2,3)
```

10. CALCULAR AS PERDAS NAS LINHAS

- **Sustentação Teórica:**

Segundo Monticelli [1], para calcular as perdas nas linhas de transmissão basta somar os fluxos que saem das barras que estão em suas extremidades. Por exemplo, para calcular as perdas na linha que interliga a barra k com a barra m, basta somar os fluxos de potências Pkm com Pmk.

$$Perdas_{km} = P_{km} + P_{mk}$$

$$Perdas_{km} = Q_{km} + Q_{mk}$$

Implementando um algoritmo no MATLAB, encontra-se que:

- Perdas (1,2) = 0.0114 + 0.0027i
- Perdas (1,3) = 0.0027 – 0.0157i
- Perdas (2,3) = 0.0111 + 0.0030i
- Total de Perdas de Potência = 0.0251 – 0.010i

- **Código Utilizado Para Calcular as Perdas nas Linhas**

```
% Script que Calcula as Perdas nas Linhas

n=1;
p=1;

for n=1:nbarras

    for p=1:nbarras

     if (n == p)
```

```
            Peql(n,p)=0.0;
            Qeql(n,p)=0.0;

        end

        Peql(n,p) = Peq(n,p)+Peq(p,n);
        Qeql(n,p) = Qeq(n,p)+Qeq(p,n);
        p=p+1;

      end

        n=n+1;

end

perdasnalinha=Peql+Qeql*i


% Perdas nas Linhas

perdas12 = perdasnalinha(1,2)
perdas13 = perdasnalinha(1,3)
perdas32 = perdasnalinha(3,2)

perdastotais = perdas12+perdas13+perdas32
```

11. MOSTRAR QUE O BALANÇO DE POTÊNCIA (ATIVA E REATIVA) É SATISFEITO EM CADA BARRA

- **Sustentação Teórica**

Segundo Monticelli [1], a injeção de potência em uma barra deve ser igual a soma dos fluxos que saem da mesma. Desta forma, precisaremos calcular as incógnitas de potência ativa e reativa geradas na barra 1, e a potência reativa consumida na barra 3, para posteriormente demonstrar o balanço.

Para calcularmos a potência gerada na barra 1, utilizaremos o conceito de que toda potência gerada nas barras tem que ser igual à soma entre as potências consumidas e as perdas do sistema. Para calcularmos os valores de potência reativa gerada na barra 1 e consumida na barra 3, utilizaremos o conceito de que a potência que entra na barra, tem que ser igual a potência que sai da mesma.

$$\sum_{1}^{n°debarras} Potência_{Gerada} - \sum_{1}^{n°debarras} Potência_{Consumida} = \sum_{1}^{n°barras} Potência_{fluxos}$$

$$\sum_{1}^{n°debarras} Potência_{Gerada} - \sum_{1}^{n°debarras} Potência_{Consumida} - \sum_{1}^{n°delinhas} Potência_{Perdas} = 0$$

Encontra-se que:

- Potência Ativa Gerada na Barra 1 [PU] = 0.2251
- Potência Reativa Gerada na Barra 1 [PU] = 0.2497
- Potência Reativa Gerada na Barra 3 [PU] = -0.0598

Assim, o somatório da potência aparente que entra em cada barra tem de ser igual ao somatório da potência aparente que sai de cada barra, para que o balanço de potência seja satisfeito:

$$\sum_{1}^{n^{o}debarras} S_{Gerada} - \sum_{1}^{n^{o}debarras} S_{Consumida} - \sum_{1}^{n^{o}barras} S_{fluxos} = 0$$

Implementando no MATLAB, encontra-se que:

- Balanço de Potência Aparente na Barra 1 = 0
- Balanço de Potência Aparente na Barra 2 = 0
- Balanço de Potência Aparente na Barra 3 = 0

- **Código Utilizado Para Calcular as Incógnitas de Potência**

```
% Calcula as Potências Ativas e Reativas Geradas na Barra 1, e a
Potência
% Reativa Gerada na Barra 3

PG1=vpa(-((Pg(2)+Pg(3)) - (Pc(1)+Pc(2)+Pc(3)) -
(perdasnalinha(1,2)+perdasnalinha(1,3)+perdasnalinha(3,2))))

QG1=vpa(-(-Qc(1)-Qeq(1,2)-Qeq(1,3)))

QG3=vpa(-(-Qc(3)-Qeq(3,1)-Qeq(3,2)))


% Novos Vetores de Potências Geradas
PgN = subs(Pg,Pg(1),PG1)
QgN = subs(Qg,[Qg(1),Qg(3)],[QG1,QG3])
```

- **Código Utilizado Para Provar o Balanço de Potência**

```
% Cálculo do Balanço de Potência Aparente nas Barras


SbalancoB3 = round(PgN(3)+QgN(3)*i-
(Pc(3)+Qc(3)*i+Peq(3,1)+Peq(3,2)+Qeq(3,1)*i+Qeq(3,2)*i))
```

```
SbalancoB2 = round(PgN(2)+QgN(2)*i-
(Pc(2)+Qc(2)*i+Peq(2,1)+Peq(2,3)+Qeq(2,1)*i+Qeq(2,3)*i))

SbalancoB1 = round(PgN(1)+QgN(1)*i-
(Pc(1)+Qc(1)*i+Peq(1,2)+Peq(1,3)+Qeq(1,2)*i+Qeq(1,3)*i))
```

12. SIMULAR E ANALISAR AS SEGUINTES SITUAÇÕES

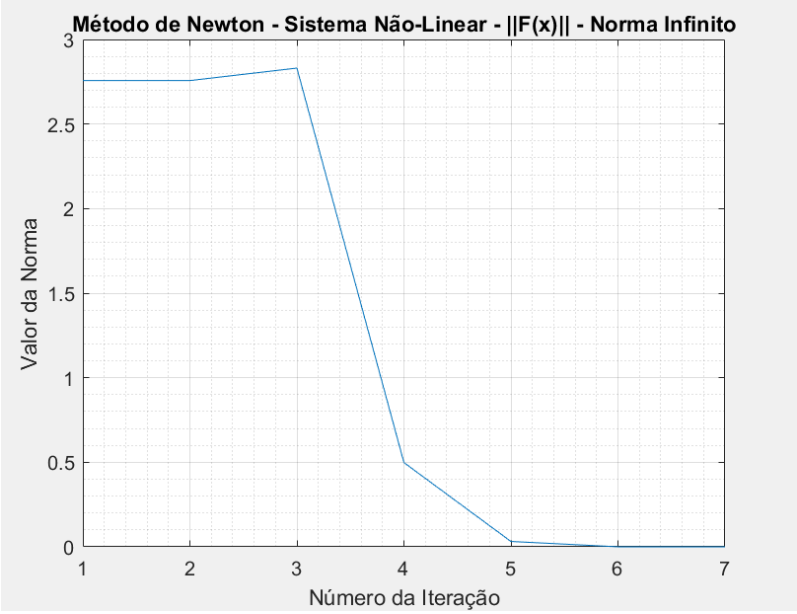
Os Códigos utilizados nesta parte do Trabalho estarão disponíveis no Anexo II, III e IV. Utilizando o mesmo script do Anexo I, apenas modificando as constantes conforme proposto, foram simulados as mesmas problemáticas. O resultado encontra-se na seção abaixo.

12.1 AUMENTO DA TENSÃO DA BARRA 1 PARA 1.1 PU

Código no Anexo II.

- **Resolução do Sistema Não-Linear Pelo Método de Newton:**

Figura 5 – Valor da Norma para Cada Iteração Realizada



Solução do Sistema Não Linear pelo Método de Newton:

V2 = 0.9725 **Fi2** = -0.0311 **Fi3** = 0.0359

Nota-se que o incremento na tensão da barra 1 alterou os valores de V2, Fi2 e Fi3. Também foram realizadas sete iterações para que o resultado convergisse. A tensão na barra 2 teve um aumento, enquanto os ângulos diminuíram.

- **Cálculo Das Tensões e Ângulos de Fase em Cada Barra:**

Tabela 4 – Tensões Fasoriais em cada Barra

Barras	Tensões [PU]	Fase [Rad]
1	1.05	0
2	**0.9725**	**-0.0311**
3	1	**0.0359**

- **Fluxos de Potência Entre as Barras:**

  - **Fluxos de Potência Ativa [PU]**

    - P(1,2) = 0.2073
    - P(1,3) = 0.0316
    - P(2,1) = - 0.1900
    - P(2,3) = - 0.3099
    - P(3,1) = - 0.0201
    - P(3,2) = 0.3201

  - **Fluxos de Potência Reativa [PU]**

    - Q(1,2) = 0.2359
    - Q(1,3) = 0.2488
    - Q(2,1) = - 0.2229
    - Q(2,3) = 0.0229
    - Q(3,1) = - 0.2481

        - Q(3,2) = -0.0219

- **Perdas Nas Linhas [PU]:**

    - Perdas (1,2) = 0.0173 + 0.0130i
    - Perdas (1,3) = 0.0114 + 0.0007i
    - Perdas (2,3) = 0.0103 + 0.0011i
    - Total de Perdas de Potência = 0.039 + 0.0148i

O Novo ajuste fez com que o total de perdas de potência nas linhas aumentasse em relação à configuração proposta inicialmente. O aumento foi de 0.0193 PU.

- **Balanço de Potência:**

    - **Potências Geradas e Consumidas [PU]:**

        - Potência Ativa Gerada na Barra 1 = 0.2389
        - Potência Reativa Gerada na Barra 1 = 0.4848
        - Potência Reativa Gerada na Barra 3 = -0.2700

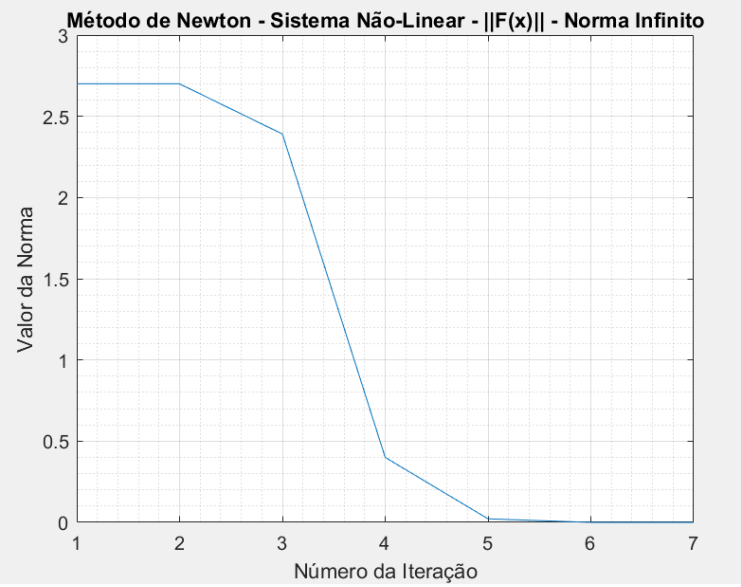
Também verificou-se que o balanço de potência aparente nas barras foi atendido, sendo nulo o resultado da diferença entre as potências que entram e que saem de cada barra. Em comparação com o sistema inicialmente proposto, nota-se um aumento na potência ativa e reativa gerada na barra 1 e um decréscimo da potência reativa gerada na 3. O aumento na tensão provocou um aumento na potência ativa e reativa da respectiva barra.

12.2 AUMENTO DA GERAÇÃO DE POTÊNCIA ATIVA DA BARRA 3 PARA 0.5 PU

Código no Anexo III.

- **Resolução do Sistema Não-Linear Pelo Método de Newton:**

Figura 6 – Valor da Norma para Cada Iteração Realizada



Solução do Sistema Não Linear pelo Método de Newton:

V2 = 0.954 **Fi2** = -0.0110 **Fi3** = 0.0689

Nota-se que o incremento de potência na barra 3 acarretou na modificação dos ângulos de fase do sistema, e manteve constante o módulo da tensão V2 em relação ao problema inicial proposto. Também foram realizadas sete iterações para que o resultado convergisse.

- **Cálculo Das Tensões e Ângulos de Fase em Cada Barra:**

Tabela 5 – Tensões Fasoriais em cada Barra

Barras	Tensões [PU]	Fase [Rad]
1	1.05	0
2	**0.954**	**-0.0110**
3	1	**0.0689**

- **Fluxos de Potência Entre as Barras:**

  - **Fluxos de Potência Ativa [PU]**

    - P(1,2) = 0.1228
    - P(1,3) = - 0.089
    - P(2,1) = - 0.1135
    - P(2,3) = - 0.3864
    - P(3,1) = - 0.0971
    - P(3,2) = 0.4028

  - **Fluxos de Potência Reativa [PU]**

    - Q(1,2) = 0.1792
    - Q(1,3) = 0.1712
    - Q(2,1) = - 0.1807
    - Q(2,3) = - 0.0192
    - Q(3,1) = - 0.1773
    - Q(3,2) = 0.0329

- **Perdas Nas Linhas [PU]:**

    - Perdas (1,2) = 0.0093 - 0.0015i
    - Perdas (1,3) = 0.0075 – 0.0060i
    - Perdas (2,3) = 0.0164 + 0.0137i
    - Total de Perdas de Potência = 0.0332 + 0.0062i

O Novo ajuste fez com que o total de perdas de potência nas linhas aumentasse em relação ao problema inicial proposto. O aumento foi de 0.008 PU.

- **Balanço de Potência:**

    - **Potências Geradas e Consumidas [PU]:**

        - Potência Ativa Gerada na Barra 1 = 0.033
        - Potência Reativa Gerada na Barra 1 = 0.3505
        - Potência Reativa Gerada na Barra 3 = -0.1443

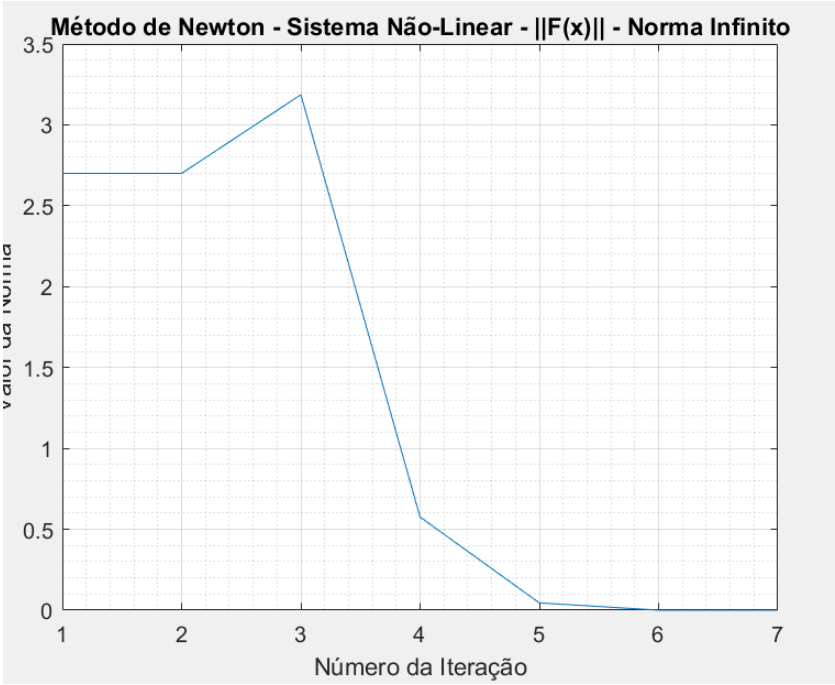
Também verificou-se que o balanço de potência aparente nas barras foi atendido, sendo nulo o resultado da diferença entre as potências que entram e que saem de cada barra. Em comparação com o sistema proposto inicialmente, nota-se uma diminuição na potência ativa gerada na barra 1, uma diminuição na potência reativa gerada na barra 3 e um aumento da potência reativa gerada na barra 1.

12.3 AUMENTO DA CARGA DE POTÊNCIA ATIVA DA BARRA 3 PARA 0.5 PU

Código no Anexo IV.

- **Resolução do Sistema Não-Linear Pelo Método de Newton:**

Figura 7 – Valor da Norma para Cada Iteração Realizada



Solução do Sistema Não Linear pelo Método de Newton:

V2 = 0.9523 **Fi2** = -0.1471 **Fi3** = -0.138

Nota-se que o incremento de potência de carga na barra 3 acarretou na modificação dos ângulos de fase do sistema e da tensão V2 em relação ao problema inicial proposto. Também foram realizadas sete iterações para que o resultado convergisse. A tensão V2 teve um decréscimo muito pequeno, e os ângulos, quando analisados em módulo, aumentaram.

- **Cálculo Das Tensões e Ângulos de Fase em Cada Barra:**

Tabela 6 – Tensões Fasoriais em cada Barra

Barras	Tensões [PU]	Fase [Rad]
1	1.05	0
2	**0.9523**	**-0.1471**
3	1	**-0.138**

- **Fluxos de Potência Entre as Barras:**

  - **Fluxos de Potência Ativa [PU]**

    - P(1,2) = 0.4065
    - P(1,3) = 0.3518
    - P(2,1) = - 0.3753
    - P(2,3) = - 0.1246
    - P(3,1) = - 0.3293
    - P(3,2) = 0.1293

  - **Fluxos de Potência Reativa [PU]**

    - Q(1,2) = 0.0689
    - Q(1,3) = -0.0306
    - Q(2,1) = -0.02675
    - Q(2,3) = - 0.17324
    - Q(3,1) = 0.0546
    - Q(3,2) = 0.1635

- **Perdas Nas Linhas [PU]:**

    - Perdas (1,2) = 0.0311 + 0.0422i
    - Perdas (1,3) = 0.0225 + 0.0240i
    - Perdas (2,3) = 0.0047 – 0.0097i
    - Total de Perdas de Potência = 0.0583 + 0.0565i

O Novo ajuste fez com que o total de perdas de potência nas linhas aumentasse em relação ao problema inicial proposto . O aumento foi de 0.033 PU.

- **Balanço de Potência:**

    - **Potências Geradas e Consumidas [PU]:**

        - Potência Ativa Gerada na Barra 1 = 0.758
        - Potência Reativa Gerada na Barra 1 = 0.038
        - Potência Reativa Gerada na Barra 3 = 0.218

Também se verificou que o balanço de potência aparente nas barras foi atendido, sendo nulo o resultado da diferença entre as potências que entram e que saem de cada barra. Em comparação com o sistema proposto inicialmente, nota-se um aumento na potência ativa gerada na barra 1, um aumento na potência reativa gerada na barra 3 e uma diminuição na potência reativa gerada na barra 1.

13. RESULTADOS E DISCUSSÕES

Através do problema proposto, pode-se colocar em prática os conceitos de fluxo de potência entre as barras de um sistema elétrico, bem como implementar um algoritmo baseado no método de newton para resolução de sistemas de equações não-lineares.

Por meio das modificações realizadas, notou-se a existência de um acoplamento forte entre as variáveis Pθ e QV, de forma que a sensibilidade entre estas variáveis é superior à sensibilidade entre PV e Qθ. Com o incremento da tensão na barra 1, observa-se que as potências reativas geradas apresentaram uma maior taxa de variação em relação ao problema inicial. Com o incremento de potência na barra 3, observa-se que os ângulos das tensões das barras se alteraram enquanto que a tensão na barra 2 se manteve praticamente constante.

14. REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. J. Monticelli, ***Fluxo de Carga em Redes de Energia Elétrica.*** São Paulo, Brasil: Edgard Blücher Ltda, 1983.

ANEXO I – CÓDIGO UTILIZADO PARA O PROBLEMA INICIAL PROPOSTO

```
% Matriz de Impedâncias Série

Zkm = [0 (0.2 + 0.4i) (0.2 + 0.4i);(0.2 +0.4i) 0 (0.1 + 0.2i); (0.2+0.4i)
(0.1+0.2i) 0 ];

% Vetor de Susceptâncias Shunt

Bsh = imag([0, 0.01i, 0.01i; 0.01i, 0, 0.01i; 0.01i, 0.01i,0]);

% Calculando a Matriz de Admitâncias

Ykm = 1./Zkm;

% Separando o Vetor de Admitâncias em Dois Vetores: Condutâncias [Gkm] e
% Susceptâncias [Ykm]

Gkm = real(Ykm);

Bkm = imag(Ykm);

% Criando as Variáveis Simbólicas (Incógnitas)

syms V2 fi2 fi3 Pg1 Qg1 Qg3;

incognitas = [V2, fi2, fi3];

% Criando Vetor de Ângulos

fi = [0, fi2, fi3];

% Criando Vetor de Tensões

V = [1.05, V2, 1];

% Criando Vetor de Potências Geradas

Pg = [Pg1, 0, 0.3];
Qg = [Qg1, 0, Qg3];

% Criando Vetor de Potências Consumidas

Pc = [0, 0.5, 0];
Qc = [0, 0.2, 0];

%Define Número Total de Barras

nbarras = [3];

% Define Quais são Barras de Geração

bger = [3];
```

```
% Define Quais são Barras de Carga

bcarga = [2];

% Define Qual a Barra de Referência

bref= [1];

% Inicializa a variável que vai armazenar as equaçãoes

eqqc = sym(zeros(1,numel(bcarga)));
eqpc = sym(zeros(1,numel(bcarga)));
eqpg = sym(zeros(1,numel(bger)));

% Inicializa Variáveis de Controle dos Loops

k=1;
j=1;
l=1;
m=1;

% Equações Barras de Carga

for j=1:numel(bcarga)

    eqqc(j) = Qg(bcarga(j))-Qc(bcarga(j));
    eqpc(j) = Pg(bcarga(j))-Pc(bcarga(j));

       while (k<nbarras)

            if k == bcarga(j)
                k=k+1;
            end

            eqpc(j) = eqpc(j) + ((-
V(bcarga(j))*V(bcarga(j))*(Gkm(bcarga(j),k)))+(V(bcarga(j))*V(k)*Gkm(bcarga(j
),k)*cos(fi(bcarga(j))-
fi(k)))+(V(bcarga(j))*V(k)*Bkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            eqqc(j) = eqqc(j) -((-
V(bcarga(j))*V(bcarga(j))*(Bkm(bcarga(j),k)+Bsh(bcarga(j),k)))+(V(bcarga(j))*
V(k)*Bkm(bcarga(j),k)*cos(fi(bcarga(j))-fi(k)))-
(V(bcarga(j))*V(k)*Gkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            k=k+1;

       end


end


% Equações Barras de Geração

for l=1:numel(bger)

    eqpg(l) = Pg(bger(l))-Pc(bger(l));

      while (m<nbarras)
```

```
            if m == bger(l)
                 m=m+1;
            end

        eqpg(l) = eqpg(l) -((V(bger(l))*V(bger(l))*(Gkm(bger(l),m)))-
(V(bger(l))*V(m)*Gkm(bger(l),m)*cos(((fi(bger(l)))-(fi(m)))))-
(V(bger(l))*V(m)*Bkm(bger(l),m)*sin((fi(bger(l))-fi(m)))));
        m=m+1;

      end

end

% Salva as Equações em um Vetor

equations = [eqpc, eqqc, eqpg]


% Script para Resolver o Sistema Não-Linear de N Equações pelo Método de
Newton

% Parâmetros necessários: Vetor de Equações, Vetor de Valores Iniciais e
% Vetor de Incógnitas

equations = [eqpc, eqqc, eqpg];
incognitas = [V2, fi2, fi3];
vetordevalores = [1, 1, 1];

% Inicializa Variáveis de Controle dos Loops

iteracoes = 1;
t=1;
u=1;

% Realiza Cálculo da Matriz Jacobiana com as Equações

for t = 1:length(equations)

       for u = 1:length(equations)

          J(u,t) = diff(equations(u),incognitas(t));

       end
end

% Substitui o vetor de valores iniciais nas equações

F = vpa(subs(equations, incognitas, vetordevalores));

% Calcula a Norma Infinito

error(iteracoes)=max(abs(F));

% Enquanto a Norma Infinito for Superior ao Critério de Parada
% Estabelecido, Realiza o Procedimento Abaixo
```

```
while(error(iteracoes)>0.00001)

    iteracoes=iteracoes+1;

    % Calcula a Matriz Jacobiana e Sua Inversa, e Múltiplica por -1

    Jsub = -1*vpa(subs(J, incognitas, vetordevalores));
    IJsub = inv(Jsub);

    % Calcula o Valor da Norma Infinito

    F = vpa(subs(equations, incognitas, vetordevalores));
    error(iteracoes)=max(abs(F));

    % Calcula o Vetor de Acréscimo

    dx=IJsub*transpose(F);

    % Atualiza o Vetor de Valores Inicial
    vetordevalores=vetordevalores+transpose(dx);
    resultado = vetordevalores;

end

% Realiza a Plotagem do Gráfico com o Valor da Norma Infinito para Cada
% Iteração

plot(1:iteracoes,error);

title('Método de Newton - Sistema Não-Linear - ||F(x)|| - Norma Infinito');
xlabel('Número da Iteração');
ylabel('Valor da Norma');
grid on;
grid minor;

% Solução do Sistema

v2=resultado(1)
fi2=resultado(2)
fi3=resultado(3)

% Script para Calcular o Fluxo de Potência Ativa e Reativa Entre as Barras
% Necessita dos Vetores de Tensões e Ângulos de Fase como Parâmetros

VN = subs(V, [incognitas(1)],[resultado(1)])
fin = subs(fi, [incognitas(2),incognitas(3)],[resultado(2),resultado(3)])

y = 1;
c = 1;
d = 1;

for y=1:nbarras

    for d=1:nbarras

         if (y == d)
```

```
            Peq(y,d)=0.0;
            Qeq(y,d)=0.0;

         end

     Peq(y,d) = (VN(y)*VN(y))*Gkm(y,d)- VN(y)*VN(d)*Gkm(y,d)*cos(fin(y)-
fin(d)) - VN(y)*VN(d)*Bkm(y,d)*sin(fin(y)-fin(d));
     Qeq(y,d) = -(VN(y)^2)*(Bkm(y,d)+Bsh(y,d)) +
VN(y)*VN(d)*Bkm(y,d)*cos(fin(y)-fin(d)) - VN(y)*VN(d)*Gkm(y,d)*sin(fin(y)-
fin(d));
     d=d+1;

    end

      y=y+1;

end

fluxoentrelinhas = Peq+Qeq*i;
fluxo12=fluxoentrelinhas(1,2)
fluxo21=fluxoentrelinhas(2,1)
fluxo13=fluxoentrelinhas(1,3)
fluxo31=fluxoentrelinhas(3,1)
fluxo32=fluxoentrelinhas(3,2)
fluxo23=fluxoentrelinhas(2,3)



% Script que Calcula as Perdas nas Linhas

n=1;
p=1;

for n=1:nbarras

    for p=1:nbarras

     if (n == p)

         Peql(n,p)=0.0;
         Qeql(n,p)=0.0;

     end

     Peql(n,p) = Peq(n,p)+Peq(p,n);
     Qeql(n,p) = Qeq(n,p)+Qeq(p,n);
     p=p+1;

    end

      n=n+1;

end

perdasnalinha=Peql+Qeql*i;

% Perdas nas Linhas
```

```
perdas12 = perdasnalinha(1,2)
perdas13 = perdasnalinha(1,3)
perdas32 = perdasnalinha(3,2)

perdastotais = perdas12+perdas13+perdas32

% Calcula as Potências Ativas e Reativas Geradas na Barra 1, e a Potência
% Reativa Gerada na Barra 3

PG1=vpa(-((Pg(2)+Pg(3)) - (Pc(1)+Pc(2)+Pc(3)) -
real((perdasnalinha(1,2)+perdasnalinha(1,3)+perdasnalinha(3,2)))))

QG1=vpa(-(-Qc(1)-Qeq(1,2)-Qeq(1,3)))

QG3=vpa(-(-Qc(3)-Qeq(3,1)-Qeq(3,2)))

% Novos Vetores de Potências Geradas nas Barras

PgN = subs(Pg,Pg(1),PG1);
QgN = subs(Qg,[Qg(1),Qg(3)],[QG1,QG3]);

% Cálculo do Balanço de Potência Aparente nas Barras

SbalancoB3 = round(PgN(3)+QgN(3)*i-
(Pc(3)+Qc(3)*i+Peq(3,1)+Peq(3,2)+Qeq(3,1)*i+Qeq(3,2)*i))

SbalancoB2 = round(PgN(2)+QgN(2)*i-
(Pc(2)+Qc(2)*i+Peq(2,1)+Peq(2,3)+Qeq(2,1)*i+Qeq(2,3)*i))

SbalancoB1 = round(PgN(1)+QgN(1)*i-
(Pc(1)+Qc(1)*i+Peq(1,2)+Peq(1,3)+Qeq(1,2)*i+Qeq(1,3)*i))
```

ANEXO II – CÓDIGO UTILIZADO PARA A ALTERAÇÃO DA TENSÃO NA BARRA 1

```
% Matriz de Impedâncias Série

Zkm = [0 (0.2 + 0.4i) (0.2 + 0.4i);(0.2 +0.4i) 0 (0.1 + 0.2i); (0.2+0.4i)
(0.1+0.2i) 0 ];

% Vetor de Susceptâncias Shunt

Bsh = imag([0, 0.01i, 0.01i; 0.01i, 0, 0.01i; 0.01i, 0.01i,0]);

% Calculando a Matriz de Admitâncias

Ykm = 1./Zkm;

% Separando o Vetor de Admitâncias em Dois Vetores: Condutâncias [Gkm] e
% Susceptâncias [Ykm]

Gkm = real(Ykm);

Bkm = imag(Ykm);

% Criando as Variáveis Simbólicas (Incógnitas)

syms V2 fi2 fi3 Pg1 Qg1 Qg3;

incognitas = [V2, fi2, fi3];

% Criando Vetor de Ângulos

fi = [0, fi2, fi3];

% Criando Vetor de Tensões

V = [1.1, V2, 1];

% Criando Vetor de Potências Geradas

Pg = [Pg1, 0, 0.3];
Qg = [Qg1, 0, Qg3];

% Criando Vetor de Potências Consumidas

Pc = [0, 0.5, 0];
Qc = [0, 0.2, 0];


%Define Número Total de Barras

nbarras = [3];

% Define Quais são Barras de Geração

bger = [3];
```

```
% Define Quais são Barras de Carga

bcarga = [2];

% Define Qual a Barra de Referência

bref= [1];

% Inicializa a variável que vai armazenar as equaçãoes

eqqc = sym(zeros(1,numel(bcarga)));
eqpc = sym(zeros(1,numel(bcarga)));
eqpg = sym(zeros(1,numel(bger)));

% Inicializa Variáveis de Controle dos Loops

k=1;
j=1;
l=1;
m=1;

% Equações Barras de Carga

for j=1:numel(bcarga)

    eqqc(j) = Qg(bcarga(j))-Qc(bcarga(j));
    eqpc(j) = Pg(bcarga(j))-Pc(bcarga(j));

       while (k<nbarras)

            if k == bcarga(j)
                k=k+1;
            end

            eqpc(j) = eqpc(j) + ((-
V(bcarga(j))*V(bcarga(j))*(Gkm(bcarga(j),k)))+(V(bcarga(j))*V(k)*Gkm(bcarga(j
),k)*cos(fi(bcarga(j))-
fi(k)))+(V(bcarga(j))*V(k)*Bkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            eqqc(j) = eqqc(j) -((-
V(bcarga(j))*V(bcarga(j))*(Bkm(bcarga(j),k)+Bsh(bcarga(j),k)))+(V(bcarga(j))*
V(k)*Bkm(bcarga(j),k)*cos(fi(bcarga(j))-fi(k)))-
(V(bcarga(j))*V(k)*Gkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            k=k+1;

       end


end


% Equações Barras de Geração

for l=1:numel(bger)

    eqpg(l) = Pg(bger(l))-Pc(bger(l));
```

```
      while (m<nbarras)

            if m == bger(l)
                 m=m+1;
            end

        eqpg(l) = eqpg(l) -((V(bger(l))*V(bger(l))*(Gkm(bger(l),m)))-
(V(bger(l))*V(m)*Gkm(bger(l),m)*cos(((fi(bger(l)))-(fi(m)))))-
(V(bger(l))*V(m)*Bkm(bger(l),m)*sin((fi(bger(l))-fi(m)))));
        m=m+1;

      end

end

% Salva as Equações em um Vetor

equations = [eqpc, eqqc, eqpg]


% Script para Resolver o Sistema Não-Linear de N Equações pelo Método de
Newton

% Parâmetros necessários: Vetor de Equações, Vetor de Valores Iniciais e
% Vetor de Incógnitas

equations = [eqpc, eqqc, eqpg];
incognitas = [V2, fi2, fi3];
vetordevalores = [1, 1, 1];

% Inicializa Variáveis de Controle dos Loops

iteracoes = 1;
t=1;
u=1;

% Realiza Cálculo da Matriz Jacobiana com as Equações

for t = 1:length(equations)

       for u = 1:length(equations)

          J(u,t) = diff(equations(u),incognitas(t));

       end
end

% Substitui o vetor de valores iniciais nas equações

F = vpa(subs(equations, incognitas, vetordevalores));

% Calcula a Norma Infinito

error(iteracoes)=max(abs(F));

% Enquanto a Norma Infinito for Superior ao Critério de Parada
% Estabelecido, Realiza o Procedimento Abaixo
```

```
while(error(iteracoes)>0.00001)

    iteracoes=iteracoes+1;

    % Calcula a Matriz Jacobiana e Sua Inversa, e Múltiplica por -1

    Jsub = -1*vpa(subs(J, incognitas, vetordevalores));
    IJsub = inv(Jsub);

    % Calcula o Valor da Norma Infinito

    F = vpa(subs(equations, incognitas, vetordevalores));
    error(iteracoes)=max(abs(F));

    % Calcula o Vetor de Acréscimo

    dx=IJsub*transpose(F);

    % Atualiza o Vetor de Valores Inicial
    vetordevalores=vetordevalores+transpose(dx);
    resultado = vetordevalores;

end

% Realiza a Plotagem do Gráfico com o Valor da Norma Infinito para Cada
% Iteração

plot(1:iteracoes,error);

title('Método de Newton - Sistema Não-Linear - ||F(x)|| - Norma Infinito');
xlabel('Número da Iteração');
ylabel('Valor da Norma');
grid on;
grid minor;

% Solução do Sistema

v2=resultado(1)
fi2=resultado(2)
fi3=resultado(3)

% Script para Calcular o Fluxo de Potência Ativa e Reativa Entre as Barras
% Necessita dos Vetores de Tensões e Ângulos de Fase como Parâmetros

VN = subs(V, [incognitas(1)],[resultado(1)])
fin = subs(fi, [incognitas(2),incognitas(3)],[resultado(2),resultado(3)])

y = 1;
c = 1;
d = 1;

for y=1:nbarras

    for d=1:nbarras

         if (y == d)
```

```
            Peq(y,d)=0.0;
            Qeq(y,d)=0.0;

         end

     Peq(y,d) = (VN(y)*VN(y))*Gkm(y,d)- VN(y)*VN(d)*Gkm(y,d)*cos(fin(y)-
fin(d)) - VN(y)*VN(d)*Bkm(y,d)*sin(fin(y)-fin(d));
     Qeq(y,d) = -(VN(y)^2)*(Bkm(y,d)+Bsh(y,d)) +
VN(y)*VN(d)*Bkm(y,d)*cos(fin(y)-fin(d)) - VN(y)*VN(d)*Gkm(y,d)*sin(fin(y)-
fin(d));
     d=d+1;

    end

      y=y+1;

end

fluxoentrelinhas = Peq+Qeq*i;
fluxo12=fluxoentrelinhas(1,2)
fluxo21=fluxoentrelinhas(2,1)
fluxo13=fluxoentrelinhas(1,3)
fluxo31=fluxoentrelinhas(3,1)
fluxo32=fluxoentrelinhas(3,2)
fluxo23=fluxoentrelinhas(2,3)



% Script que Calcula as Perdas nas Linhas

n=1;
p=1;

for n=1:nbarras

    for p=1:nbarras

     if (n == p)

         Peql(n,p)=0.0;
         Qeql(n,p)=0.0;

     end

     Peql(n,p) = Peq(n,p)+Peq(p,n);
     Qeql(n,p) = Qeq(n,p)+Qeq(p,n);
     p=p+1;

    end

      n=n+1;

end

perdasnalinha=Peql+Qeql*i;
```

```
% Perdas nas Linhas

perdas12 = perdasnalinha(1,2)
perdas13 = perdasnalinha(1,3)
perdas32 = perdasnalinha(3,2)

perdastotais = perdas12+perdas13+perdas32

% Calcula as Potências Ativas e Reativas Geradas na Barra 1, e a Potência
% Reativa Gerada na Barra 3

PG1=vpa(-((Pg(2)+Pg(3)) - (Pc(1)+Pc(2)+Pc(3)) -
real((perdasnalinha(1,2)+perdasnalinha(1,3)+perdasnalinha(3,2)))))

QG1=vpa(-(-Qc(1)-Qeq(1,2)-Qeq(1,3)))

QG3=vpa(-(-Qc(3)-Qeq(3,1)-Qeq(3,2)))

% Novos Vetores de Potências Geradas nas Barras

PgN = subs(Pg,Pg(1),PG1);
QgN = subs(Qg,[Qg(1),Qg(3)],[QG1,QG3]);

% Cálculo do Balanço de Potência Aparente nas Barras

SbalancoB3 = round(PgN(3)+QgN(3)*i-
(Pc(3)+Qc(3)*i+Peq(3,1)+Peq(3,2)+Qeq(3,1)*i+Qeq(3,2)*i))

SbalancoB2 = round(PgN(2)+QgN(2)*i-
(Pc(2)+Qc(2)*i+Peq(2,1)+Peq(2,3)+Qeq(2,1)*i+Qeq(2,3)*i))

SbalancoB1 = round(PgN(1)+QgN(1)*i-
(Pc(1)+Qc(1)*i+Peq(1,2)+Peq(1,3)+Qeq(1,2)*i+Qeq(1,3)*i))
```

ANEXO III – CÓDIGO UTILIZADO PARA A ALTERAÇÃO DA POTÊNCIA ATIVA NA BARRA 3

```
% Matriz de Impedâncias Série

Zkm = [0 (0.2 + 0.4i) (0.2 + 0.4i);(0.2 +0.4i) 0 (0.1 + 0.2i); (0.2+0.4i)
(0.1+0.2i) 0 ];

% Vetor de Susceptâncias Shunt

Bsh = imag([0, 0.01i, 0.01i; 0.01i, 0, 0.01i; 0.01i, 0.01i,0]);

% Calculando a Matriz de Admitâncias

Ykm = 1./Zkm;

% Separando o Vetor de Admitâncias em Dois Vetores: Condutâncias [Gkm] e
% Susceptâncias [Ykm]

Gkm = real(Ykm);

Bkm = imag(Ykm);

% Criando as Variáveis Simbólicas (Incógnitas)

syms V2 fi2 fi3 Pg1 Qg1 Qg3;

incognitas = [V2, fi2, fi3];

% Criando Vetor de Ângulos

fi = [0, fi2, fi3];

% Criando Vetor de Tensões

V = [1.05, V2, 1];

% Criando Vetor de Potências Geradas

Pg = [Pg1, 0, 0.5];
Qg = [Qg1, 0, Qg3];

% Criando Vetor de Potências Consumidas

Pc = [0, 0.5, 0];
Qc = [0, 0.2, 0];
%Define Número Total de Barras

nbarras = [3];

    % Define Quais são Barras de Geração

    bger = [3];
```

```
    % Define Quais são Barras de Carga

    bcarga = [2];
% Define Qual a Barra de Referência

bref= [1];

% Inicializa a variável que vai armazenar as equaçãoes

eqqc = sym(zeros(1,numel(bcarga)));
eqpc = sym(zeros(1,numel(bcarga)));
eqpg = sym(zeros(1,numel(bger)));

% Inicializa Variáveis de Controle dos Loops

k=1;
j=1;
l=1;
m=1;

% Equações Barras de Carga

for j=1:numel(bcarga)

    eqqc(j) = Qg(bcarga(j))-Qc(bcarga(j));
    eqpc(j) = Pg(bcarga(j))-Pc(bcarga(j));

       while (k<nbarras)

            if k == bcarga(j)
                k=k+1;
            end

            eqpc(j) = eqpc(j) + ((-
V(bcarga(j))*V(bcarga(j))*(Gkm(bcarga(j),k)))+(V(bcarga(j))*V(k)*Gkm(bcarga(j
),k)*cos(fi(bcarga(j))-
fi(k)))+(V(bcarga(j))*V(k)*Bkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            eqqc(j) = eqqc(j) -((-
V(bcarga(j))*V(bcarga(j))*(Bkm(bcarga(j),k)+Bsh(bcarga(j),k)))+(V(bcarga(j))*
V(k)*Bkm(bcarga(j),k)*cos(fi(bcarga(j))-fi(k)))-
(V(bcarga(j))*V(k)*Gkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            k=k+1;

       end


end


% Equações Barras de Geração

for l=1:numel(bger)

    eqpg(l) = Pg(bger(l))-Pc(bger(l));

      while (m<nbarras)
```

```
            if m == bger(l)
                 m=m+1;
            end

        eqpg(l) = eqpg(l) -((V(bger(l))*V(bger(l))*(Gkm(bger(l),m)))-
(V(bger(l))*V(m)*Gkm(bger(l),m)*cos(((fi(bger(l)))-(fi(m)))))-
(V(bger(l))*V(m)*Bkm(bger(l),m)*sin((fi(bger(l))-fi(m)))));
        m=m+1;

      end

end

% Salva as Equações em um Vetor

equations = [eqpc, eqqc, eqpg]


% Script para Resolver o Sistema Não-Linear de N Equações pelo Método de
Newton

% Parâmetros necessários: Vetor de Equações, Vetor de Valores Iniciais e
% Vetor de Incógnitas

equations = [eqpc, eqqc, eqpg];
incognitas = [V2, fi2, fi3];
vetordevalores = [1, 1, 1];

% Inicializa Variáveis de Controle dos Loops

iteracoes = 1;
t=1;
u=1;

% Realiza Cálculo da Matriz Jacobiana com as Equações

for t = 1:length(equations)

       for u = 1:length(equations)

          J(u,t) = diff(equations(u),incognitas(t));

       end
end

% Substitui o vetor de valores iniciais nas equações

F = vpa(subs(equations, incognitas, vetordevalores));

% Calcula a Norma Infinito

error(iteracoes)=max(abs(F));

% Enquanto a Norma Infinito for Superior ao Critério de Parada
% Estabelecido, Realiza o Procedimento Abaixo

while(error(iteracoes)>0.00001)
```

```
    iteracoes=iteracoes+1;

    % Calcula a Matriz Jacobiana e Sua Inversa, e Múltiplica por -1

    Jsub = -1*vpa(subs(J, incognitas, vetordevalores));
    IJsub = inv(Jsub);

    % Calcula o Valor da Norma Infinito

    F = vpa(subs(equations, incognitas, vetordevalores));
    error(iteracoes)=max(abs(F));

    % Calcula o Vetor de Acréscimo

    dx=IJsub*transpose(F);

    % Atualiza o Vetor de Valores Inicial
    vetordevalores=vetordevalores+transpose(dx);
    resultado = vetordevalores;

end

% Realiza a Plotagem do Gráfico com o Valor da Norma Infinito para Cada
% Iteração

plot(1:iteracoes,error);

title('Método de Newton - Sistema Não-Linear - ||F(x)|| - Norma Infinito');
xlabel('Número da Iteração');
ylabel('Valor da Norma');
grid on;
grid minor;

% Solução do Sistema

v2=resultado(1)
fi2=resultado(2)
fi3=resultado(3)

% Script para Calcular o Fluxo de Potência Ativa e Reativa Entre as Barras
% Necessita dos Vetores de Tensões e Ângulos de Fase como Parâmetros

VN = subs(V, [incognitas(1)],[resultado(1)])
fin = subs(fi, [incognitas(2),incognitas(3)],[resultado(2),resultado(3)])

y = 1;
c = 1;
d = 1;

for y=1:nbarras

    for d=1:nbarras

         if (y == d)

            Peq(y,d)=0.0;
```

```
            Qeq(y,d)=0.0;

          end

     Peq(y,d) = (VN(y)*VN(y))*Gkm(y,d)- VN(y)*VN(d)*Gkm(y,d)*cos(fin(y)-
fin(d)) - VN(y)*VN(d)*Bkm(y,d)*sin(fin(y)-fin(d));
     Qeq(y,d) = -(VN(y)^2)*(Bkm(y,d)+Bsh(y,d)) +
VN(y)*VN(d)*Bkm(y,d)*cos(fin(y)-fin(d)) - VN(y)*VN(d)*Gkm(y,d)*sin(fin(y)-
fin(d));
     d=d+1;

    end

      y=y+1;

end

fluxoentrelinhas = Peq+Qeq*i;
fluxo12=fluxoentrelinhas(1,2)
fluxo21=fluxoentrelinhas(2,1)
fluxo13=fluxoentrelinhas(1,3)
fluxo31=fluxoentrelinhas(3,1)
fluxo32=fluxoentrelinhas(3,2)
fluxo23=fluxoentrelinhas(2,3)



% Script que Calcula as Perdas nas Linhas

n=1;
p=1;

for n=1:nbarras

    for p=1:nbarras

     if (n == p)

         Peql(n,p)=0.0;
         Qeql(n,p)=0.0;

     end

     Peql(n,p) = Peq(n,p)+Peq(p,n);
     Qeql(n,p) = Qeq(n,p)+Qeq(p,n);
     p=p+1;

    end

      n=n+1;

end

perdasnalinha=Peql+Qeql*i;

% Perdas nas Linhas
```

```
perdas12 = perdasnalinha(1,2)
perdas13 = perdasnalinha(1,3)
perdas32 = perdasnalinha(3,2)

perdastotais = perdas12+perdas13+perdas32

% Calcula as Potências Ativas e Reativas Geradas na Barra 1, e a Potência
% Reativa Gerada na Barra 3

PG1=vpa(-((Pg(2)+Pg(3)) - (Pc(1)+Pc(2)+Pc(3)) -
real((perdasnalinha(1,2)+perdasnalinha(1,3)+perdasnalinha(3,2)))))

QG1=vpa(-(-Qc(1)-Qeq(1,2)-Qeq(1,3)))

QG3=vpa(-(-Qc(3)-Qeq(3,1)-Qeq(3,2)))

% Novos Vetores de Potências Geradas nas Barras

PgN = subs(Pg,Pg(1),PG1);
QgN = subs(Qg,[Qg(1),Qg(3)],[QG1,QG3]);

% Cálculo do Balanço de Potência Aparente nas Barras

SbalancoB3 = round(PgN(3)+QgN(3)*i-
(Pc(3)+Qc(3)*i+Peq(3,1)+Peq(3,2)+Qeq(3,1)*i+Qeq(3,2)*i))

SbalancoB2 = round(PgN(2)+QgN(2)*i-
(Pc(2)+Qc(2)*i+Peq(2,1)+Peq(2,3)+Qeq(2,1)*i+Qeq(2,3)*i))

SbalancoB1 = round(PgN(1)+QgN(1)*i-
(Pc(1)+Qc(1)*i+Peq(1,2)+Peq(1,3)+Qeq(1,2)*i+Qeq(1,3)*i))
```

ANEXO IV – CÓDIGO UTILIZADO PARA A ALTERAÇÃO DA POTÊNCIA ATIVA DE CARGA NA BARRA 3

```
    % Matriz de Impedâncias Série

    Zkm = [0 (0.2 + 0.4i) (0.2 + 0.4i);(0.2 +0.4i) 0 (0.1 + 0.2i); (0.2+0.4i)
(0.1+0.2i) 0 ];

    % Vetor de Susceptâncias Shunt

    Bsh = imag([0, 0.01i, 0.01i; 0.01i, 0, 0.01i; 0.01i, 0.01i,0]);

    % Calculando a Matriz de Admitâncias

    Ykm = 1./Zkm;

    % Separando o Vetor de Admitâncias em Dois Vetores: Condutâncias [Gkm] e
    % Susceptâncias [Ykm]

    Gkm = real(Ykm);

    Bkm = imag(Ykm);

    % Criando as Variáveis Simbólicas (Incógnitas)

    syms V2 fi2 fi3 Pg1 Qg1 Qg3;

    incognitas = [V2, fi2, fi3];

    % Criando Vetor de Ângulos

    fi = [0, fi2, fi3];

    % Criando Vetor de Tensões

    V = [1.05, V2, 1];

    % Criando Vetor de Potências Geradas

    Pg = [Pg1, 0, 0.3];
    Qg = [Qg1, 0, Qg3];

    % Criando Vetor de Potências Consumidas

    Pc = [0, 0.5, 0.5];
    Qc = [0, 0.2, 0];

    %Define Número Total de Barras

    nbarras = [3];

    % Define Quais são Barras de Geração

    bger = [3];
```

```
    % Define Quais são Barras de Carga

    bcarga = [2];
% Define Qual a Barra de Referência

bref= [1];

% Inicializa a variável que vai armazenar as equaçãoes

eqqc = sym(zeros(1,numel(bcarga)));
eqpc = sym(zeros(1,numel(bcarga)));
eqpg = sym(zeros(1,numel(bger)));

% Inicializa Variáveis de Controle dos Loops

k=1;
j=1;
l=1;
m=1;

% Equações Barras de Carga

for j=1:numel(bcarga)

    eqqc(j) = Qg(bcarga(j))-Qc(bcarga(j));
    eqpc(j) = Pg(bcarga(j))-Pc(bcarga(j));

       while (k<nbarras)

            if k == bcarga(j)
                k=k+1;
            end

            eqpc(j) = eqpc(j) + ((-
V(bcarga(j))*V(bcarga(j))*(Gkm(bcarga(j),k)))+(V(bcarga(j))*V(k)*Gkm(bcarga(j
),k)*cos(fi(bcarga(j))-
fi(k)))+(V(bcarga(j))*V(k)*Bkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            eqqc(j) = eqqc(j) -((-
V(bcarga(j))*V(bcarga(j))*(Bkm(bcarga(j),k)+Bsh(bcarga(j),k)))+(V(bcarga(j))*
V(k)*Bkm(bcarga(j),k)*cos(fi(bcarga(j))-fi(k)))-
(V(bcarga(j))*V(k)*Gkm(bcarga(j),k)*sin(fi(bcarga(j))-fi(k))));
            k=k+1;

       end


end


% Equações Barras de Geração

for l=1:numel(bger)

    eqpg(l) = Pg(bger(l))-Pc(bger(l));

      while (m<nbarras)
```

```
            if m == bger(l)
                 m=m+1;
            end

        eqpg(l) = eqpg(l) -((V(bger(l))*V(bger(l))*(Gkm(bger(l),m)))-
(V(bger(l))*V(m)*Gkm(bger(l),m)*cos(((fi(bger(l)))-(fi(m)))))-
(V(bger(l))*V(m)*Bkm(bger(l),m)*sin((fi(bger(l))-fi(m)))));
        m=m+1;

      end

end

% Salva as Equações em um Vetor

equations = [eqpc, eqqc, eqpg]


% Script para Resolver o Sistema Não-Linear de N Equações pelo Método de
Newton

% Parâmetros necessários: Vetor de Equações, Vetor de Valores Iniciais e
% Vetor de Incógnitas

equations = [eqpc, eqqc, eqpg];
incognitas = [V2, fi2, fi3];
vetordevalores = [1, 1, 1];

% Inicializa Variáveis de Controle dos Loops

iteracoes = 1;
t=1;
u=1;

% Realiza Cálculo da Matriz Jacobiana com as Equações

for t = 1:length(equations)

       for u = 1:length(equations)

          J(u,t) = diff(equations(u),incognitas(t));

       end
end

% Substitui o vetor de valores iniciais nas equações

F = vpa(subs(equations, incognitas, vetordevalores));

% Calcula a Norma Infinito

error(iteracoes)=max(abs(F));

% Enquanto a Norma Infinito for Superior ao Critério de Parada
% Estabelecido, Realiza o Procedimento Abaixo
```

```
while(error(iteracoes)>0.00001)

    iteracoes=iteracoes+1;

    % Calcula a Matriz Jacobiana e Sua Inversa, e Múltiplica por -1

    Jsub = -1*vpa(subs(J, incognitas, vetordevalores));
    IJsub = inv(Jsub);

    % Calcula o Valor da Norma Infinito

    F = vpa(subs(equations, incognitas, vetordevalores));
    error(iteracoes)=max(abs(F));

    % Calcula o Vetor de Acréscimo

    dx=IJsub*transpose(F);

    % Atualiza o Vetor de Valores Inicial
    vetordevalores=vetordevalores+transpose(dx);
    resultado = vetordevalores;

end

% Realiza a Plotagem do Gráfico com o Valor da Norma Infinito para Cada
% Iteração

plot(1:iteracoes,error);

title('Método de Newton - Sistema Não-Linear - ||F(x)|| - Norma Infinito');
xlabel('Número da Iteração');
ylabel('Valor da Norma');
grid on;
grid minor;

% Solução do Sistema

v2=resultado(1)
fi2=resultado(2)
fi3=resultado(3)

% Script para Calcular o Fluxo de Potência Ativa e Reativa Entre as Barras
% Necessita dos Vetores de Tensões e Ângulos de Fase como Parâmetros

VN = subs(V, [incognitas(1)],[resultado(1)])
fin = subs(fi, [incognitas(2),incognitas(3)],[resultado(2),resultado(3)])

y = 1;
c = 1;
d = 1;

for y=1:nbarras

    for d=1:nbarras

         if (y == d)
```

```
            Peq(y,d)=0.0;
            Qeq(y,d)=0.0;

         end

     Peq(y,d) = (VN(y)*VN(y))*Gkm(y,d)- VN(y)*VN(d)*Gkm(y,d)*cos(fin(y)-
fin(d)) - VN(y)*VN(d)*Bkm(y,d)*sin(fin(y)-fin(d));
     Qeq(y,d) = -(VN(y)^2)*(Bkm(y,d)+Bsh(y,d)) +
VN(y)*VN(d)*Bkm(y,d)*cos(fin(y)-fin(d)) - VN(y)*VN(d)*Gkm(y,d)*sin(fin(y)-
fin(d));
     d=d+1;

    end

      y=y+1;

end

fluxoentrelinhas = Peq+Qeq*i;
fluxo12=fluxoentrelinhas(1,2)
fluxo21=fluxoentrelinhas(2,1)
fluxo13=fluxoentrelinhas(1,3)
fluxo31=fluxoentrelinhas(3,1)
fluxo32=fluxoentrelinhas(3,2)
fluxo23=fluxoentrelinhas(2,3)



% Script que Calcula as Perdas nas Linhas

n=1;
p=1;

for n=1:nbarras

    for p=1:nbarras

     if (n == p)

         Peql(n,p)=0.0;
         Qeql(n,p)=0.0;

     end

     Peql(n,p) = Peq(n,p)+Peq(p,n);
     Qeql(n,p) = Qeq(n,p)+Qeq(p,n);
     p=p+1;

    end

      n=n+1;

end

perdasnalinha=Peql+Qeql*i;

% Perdas nas Linhas
```

```
perdas12 = perdasnalinha(1,2)
perdas13 = perdasnalinha(1,3)
perdas32 = perdasnalinha(3,2)

perdastotais = perdas12+perdas13+perdas32

% Calcula as Potências Ativas e Reativas Geradas na Barra 1, e a Potência
% Reativa Gerada na Barra 3

PG1=vpa(-((Pg(2)+Pg(3)) - (Pc(1)+Pc(2)+Pc(3)) -
real((perdasnalinha(1,2)+perdasnalinha(1,3)+perdasnalinha(3,2)))))

QG1=vpa(-(-Qc(1)-Qeq(1,2)-Qeq(1,3)))

QG3=vpa(-(-Qc(3)-Qeq(3,1)-Qeq(3,2)))

% Novos Vetores de Potências Geradas nas Barras

PgN = subs(Pg,Pg(1),PG1);
QgN = subs(Qg,[Qg(1),Qg(3)],[QG1,QG3]);

% Cálculo do Balanço de Potência Aparente nas Barras

SbalancoB3 = round(PgN(3)+QgN(3)*i-
(Pc(3)+Qc(3)*i+Peq(3,1)+Peq(3,2)+Qeq(3,1)*i+Qeq(3,2)*i))

SbalancoB2 = round(PgN(2)+QgN(2)*i-
(Pc(2)+Qc(2)*i+Peq(2,1)+Peq(2,3)+Qeq(2,1)*i+Qeq(2,3)*i))

SbalancoB1 = round(PgN(1)+QgN(1)*i-
(Pc(1)+Qc(1)*i+Peq(1,2)+Peq(1,3)+Qeq(1,2)*i+Qeq(1,3)*i))
```