Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

01 de Dezembro de 2011

FLUXO DE POTÊNCIA ATIVA LINEAR CONSIDERANDO AS PERDAS

Natalie Granadeiro Mintz

PUC RIO

www.ele.puc-rio.br

Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FLUXO DE POTÊNCIA ATIVA LINEAR CONSIDERANDO AS PERDAS

Aluna: Natalie Granadeiro Mintz

Orientador: Eduardo J. S. Pires de Souza

Trabalho apresentado como requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Agradeço, primeiramente, a Deus, não só por ter me sustentado durante todos esses anos de Engenharia, mas por ser uma presença concreta e paternal desde o início da minha vida.

Aos meus pais e à minha irmã, pelo amor, apoio e incentivo que sempre deles recebi.

Ao meu querido esposo Bruno, pelo amor, amizade, paciência e encorajamento que todos os dias dele recebo.

A toda a minha família e, em especial, à avó Neuza, pelo carinho e pela ajuda material que me permitiu dar continuidade ao curso de Engenharia na PUC.

Aos colegas de trabalho Rosana Werneck, Denise Guimarães, Paulo Fraga, Cláudia Spitz e outros, por terem me incentivado e ajudado a dar continuidade ao curso.

Aos meus amigos da fraternidade de Comunhão e Libertação, que me acompanharam de perto durante toda a graduação.

Aos professores do departamento de Engenharia Elétrica, em especial aos professores Eduardo Pires, Ana Pavani, Delberis Lima e Eduardo Pacheco, pela dedicação e disponibilidade, principalmente nos momentos de dúvidas e dificuldades.

Ao professor Eduardo Pires, pela orientação e correções atenciosas no desenvolvimento deste trabalho.

Resumo

Os estudos de fluxo de potência têm fundamental importância no planejamento e na operação de um sistema de energia elétrica. Consistem em obter valores de potência ativa e reativa nas linhas e nos transformadores, valores de módulo e ângulo da tensão em cada barra e algumas outras grandezas de interesse, possibilitando assim determinar o estado em que a rede está operando. Estes estudos são realizados por meio de programas computacionais que se baseiam em alguns métodos de cálculo, como o de Gauss, Gauss-Seidel, Newton-Raphson, Newton-Raphson com Jacobiano constante, os métodos desacoplados e o do Fluxo de Potência Ativa Linear (Modelo CC).

Este projeto enfatiza o método do Fluxo de Potência Ativa Linear (Modelo CC). Uma característica importante desse modelo linearizado é o fato dele sempre fornecer uma solução, mesmo para os problemas que não podem ser resolvidos pelos métodos convencionais de cálculo do fluxo de carga.

O objetivo deste trabalho foi desenvolver um programa computacional baseado no Modelo CC para o cálculo do fluxo de potência ativa linear considerando as perdas. Os resultados obtidos foram comparados com os resultados de um programa convencional de fluxo de potência.

Palavras-chave: Fluxo de potência, fluxo de carga, modelo linear, potência ativa, análise de contingências

Linear Active Power Flow Considering the Losses

Abstract

Power flow studies are of fundamental importance in the planning and operation of an electric power system. They consist in obtaining values of active and reactive power in lines and transformers, magnitude and angle values of the voltage in each bus, and some other quantities of interest, which allow checks on the network status. These studies are performed using computer programs that are based on calculation methods such as Gauss, Gauss-Seidel, Newton-Raphson, Newton-Raphson with constant Jacobian, decoupled methods and the Linear Active Power Flow method (DC Model).

This project emphasizes the Linear Active Power Flow method (DC Model). An important property of this linearized model is that it always provides a solution, even for problems that cannot be solved by conventional methods for calculating the load flow.

The purpose of this work was to develop a computer program based on the dc model for calculating the linear active power flow considering the losses. The results obtained were compared with those ones of a conventional power flow program.

Keywords: Power-flow, load flow, linear model, active power, contingency analysis.

Sumário

1. Introdução

1.1 Aspectos gerais do fluxo de potência

Os estudos de fluxo de potência têm fundamental importância no planejamento e na operação de um sistema de energia elétrica. Consistem em obter valores de potência ativa e reativa nas linhas e nos transformadores, valores de módulo e ângulo das tensões em cada barra e algumas outras grandezas de interesse, possibilitando assim determinar o estado em que a rede está operando. Este estudo utiliza uma modelagem estática do sistema, ou seja, é suposto que ele esteja em equilíbrio. Além disso, pode ser formulado por sistemas de equações e inequações algébricas não-lineares, onde não são consideradas as variações no tempo, ou efeitos transitórios.

Um sistema elétrico está condicionado às restrições de carga, representadas pelas injeções de potência ativa e reativa especificadas nas barras de carga e pela injeção de potência ativa nas barras de geração; e às restrições de operação, que são os limites impostos às magnitudes de tensão nodais, aos fluxos de potência aparente nos transformadores e nas linhas de transmissão e às injeções de potência reativa pelos geradores [1]. Para que um sistema de energia opere dentro dessas restrições, são realizados estudos do fluxo de carga utilizando-se programas computacionais. Estes se baseiam em alguns métodos de cálculo, como o de Gauss, Gauss-Seidel, Newton-Raphson, Newton-Raphson com Jacobiano constante, os métodos desacoplados e o Modelo CC (Fluxo de Potência Ativa Linear), ao qual daremos ênfase neste trabalho.

1.2 Formulação básica do problema

A resolução do problema de fluxo de carga pode ser efetuado através de equações baseadas nas leis de Kirchhoff e inequações algébricas relacionadas com as restrições operacionais da rede e seus componentes [1] . As variáveis de um sistema de energia podem ser classificadas em três tipos:

1. Variáveis de perturbação ou não controláveis, que referem-se às potências ativa e reativa demandadas na rede (P_D e Q_D);

2. Variáveis de controle, que são as potências ativa e reativa geradas do sistema (P_G e Q_G);

3. Variáveis de estado, que são as magnitudes e ângulos de tensão nas barras da rede (V e θ).

As variáveis de estado devem obedecer às seguintes restrições práticas operacionais:

$$| \theta_i - \theta_j | < \theta_{ij}^{(max)}$$

$$V_i^{(min)} \leq V_i \leq V_i^{(max)} \quad (\Delta V_i \approx \pm 0{,}05 \text{ pu ou } \pm 5\ \%)$$

As variáveis de controle devem obedecer às seguintes restrições:

$$0 \leq P_{Gi} \leq P_{Gi}^{(max)}$$

$$Q_{Gi}^{(min)} \leq Q_{Gi} \leq Q_{Gi}^{(max)}$$

No cálculo do fluxo de carga são associados quatro parâmetros para cada barra do sistema, sendo que dois deles entram como incógnitas nas equações:

V_i magnitude de tensão na barra *i*;

θ_i ângulo da tensão na barra *i*;

P_i injeção líquida de potência ativa
(potência ativa gerada menos potência ativa demandada na barra *i*);

Q_i injeção líquida de potência reativa
(potência reativa gerada menos potência reativa demandada na barra *i*).

Quando V_i e θ_i já foram especificados e P_i e Q_i são as incógnitas da equação, esta barra é chamada barra de referência ou barra *swing.* É assim chamada por fornecer a referência de tensão do sistema e fechar o balanço de potência, considerando as perdas.

Quando P_i e V_i são fornecidos e o ângulo θ_i e a injeção líquida de potência reativa Q_i são calculados, esta barra é chamada barra de tensão controlada ou barra PV.

Por último, quando P_i e Q_i são fornecidos, tem-se uma barra de carga, ou tipo PQ, onde V_i e θ_i serão as incógnitas do problema.

2. Alguns Métodos para Cálculo do Fluxo de Carga

Algumas soluções computacionais dos problemas de fluxo de carga são obtidas por um processo iterativo de equações algébricas não-lineares. As equações utilizadas nesses métodos se relacionam com a primeira Lei de Kirchhoff, pois a potência líquida injetada deve ser igual à soma dos fluxos que deixam a barra através das linhas de transmissão e transformadores:

$$S_i^* = P_i - jQ_i = Y_{i1}\dot{V}_1\dot{V}_i^* + Y_{i2}\dot{V}_2\dot{V}_i^* + \ \ldots \ + Y_{in}\dot{V}_n\dot{V}_i^* \tag{1}$$

Assim,

$$P_i - jQ_i = \dot{V}_i^* \sum_{k=1}^{n} Y_{ik}\dot{V}_k \tag{2}$$

2.1 Método de Gauss

Neste método, inicialmente são estimados valores para as tensões desconhecidas. Estes valores são utilizados juntamente com as potências ativa e reativa especificadas para calcular um novo valor de tensão para cada barra [2]. Assim, dentro de uma iteração, temos:

$$x_1^{(v+1)} = F_1\,(x_1^{(v)}, x_2^{(v)}, \ldots, x_n^{(v)})$$
$$x_2^{(v+1)} = F_2\,(x_1^{(v)}, x_2^{(v)}, \ldots, x_n^{(v)})$$
$$\vdots$$
$$x_n^{(v+1)} = F_n\,(x_1^{(v)}, x_2^{(v)}, \ldots, x_n^{(v)})$$

As iterações são realizadas até que a diferença entre o resultado de uma iteração anterior e o da atual chegue a um mínimo valor desejado, ou seja,

$$|x^{(v+1)} - x^{(v)}| < \varepsilon$$

Para se obter a equação utilizada neste método, tomamos a equação (2) e separamos o termo relacionado com a barra i:

$$P_i - jQ_i = \dot{V}_i^* \left(\sum_{k=1\neq i}^{n} Y_{ik}\dot{V}_k + \dot{V}_i Y_{ii} \right) \tag{3}$$

Após algumas manipulações algébricas, temos:

$$\dot{V}_i^{(v+1)} = \frac{1}{Y_{ii}} \left[\frac{P_i - jQ_i}{\dot{V}_i^{*(v)}} - \sum_{k\neq i} Y_{ik}\dot{V}_k^{(v)} \right] \tag{4}$$

onde

P_i	potência ativa líquida na barra local
Q_i	potência reativa líquida na barra local

$\dot{V}_i^{(v+1)}$	tensão (complexa) na barra local, na iteração seguinte
$\dot{V}_i^{*(v)}$	tensão (complexa conjugada) na barra local, na iteração atual
$\dot{V}_k^{(v)}$	tensão (complexa) na barra vizinha, na iteração atual
Y_{ii}	admitância própria
Y_{ik}	admitância mútua

Caso a barra seja do tipo PV, calcula-se o valor de Q_i através da equação (5) abaixo e verifica-se a compatibilidade do valor obtido com a faixa de geração de potência reativa, ou seja, Q_i deve estar entre os valores de $Q_i^{(min)}$ e $Q_i^{(max)}$.

$$Q_i = -Im\{ Y_{i1}\dot{V}_1\dot{V}_i^* + Y_{i2}\dot{V}_2\dot{V}_i^* + \ \ldots\ + Y_{in}\dot{V}_n\dot{V}_i^* \} \tag{5}$$

sendo n o número de barras do sistema.

Caso Q_i encontre-se fora dos limites, o valor do limite excedido passa a ser utilizado no cálculo, a magnitude de tensão originalmente especificada é desconsiderada e a barra é tratada como sendo do tipo PQ.

2.2 Método de Gauss-Seidel

Com este método de iteração consegue-se uma maior velocidade de convergência em relação ao método de Gauss graças a uma simples modificação, que consiste em usar a iteração seguinte assim que ela for obtida [7]. Nestas condições, ao calcularmos o valor de $x_2^{(v+1)}$, por exemplo, não usamos $x_1^{(v)}$, e sim $x_1^{(v+1)}$.

2.3 Método de Newton-Raphson

O método de Newton-Raphson é o mais empregado no cálculo de fluxo de carga, pois sempre fornece um resultado, se houver solução, e tem uma convergência relativamente rápida. Como nos outros métodos, consiste em calcular as variáveis de estado para todas as barras da rede, tornando possível assim o cálculo de outras variáveis de interesse, como, por exemplo, os fluxos de potência nas linhas de transmissão e transformadores [1].

Utilizando-se a série de Taylor truncada nos seus dois primeiros termos, tem-se:

$$y = f(x) = f(x_0) + f'(x_0)\,\Delta x \tag{6}$$

Reorganizando os termos e adaptando a equação (6) para um sistema com várias variáveis, temos:

$$\boldsymbol{Y} - \boldsymbol{f}(x^{(v)}) = \boldsymbol{J}(x^{(v)})\,\Delta\,\boldsymbol{x}^{(v)} \tag{7}$$

onde $\boldsymbol{J}(x^{(v)}) = \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$.

Neste método, o termo $\boldsymbol{Y}$ - $\boldsymbol{f}$ $(x^{(\nu)})$ representa ΔP e ΔQ, e $\Delta \boldsymbol{x}^{(\nu)}$, $\Delta\theta$ e ΔV. Reescrevendo a equação (7), temos

$$\begin{bmatrix}\Delta \boldsymbol{P}\\ \Delta \boldsymbol{Q}\end{bmatrix}=\begin{bmatrix}\boldsymbol{J}_1 & \boldsymbol{J}_2\\ \boldsymbol{J}_3 & \boldsymbol{J}_4\end{bmatrix}\begin{bmatrix}\Delta \boldsymbol{\theta}\\ \Delta \boldsymbol{V}\end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix}\Delta \boldsymbol{\theta}\\ \Delta \boldsymbol{V}\end{bmatrix}=\begin{bmatrix}\boldsymbol{J}_1 & \boldsymbol{J}_2\\ \boldsymbol{J}_3 & \boldsymbol{J}_4\end{bmatrix}^{-1}\begin{bmatrix}\Delta \boldsymbol{P}\\ \Delta \boldsymbol{Q}\end{bmatrix} \tag{8}$$

onde

$$\Delta P_i = P_i^{esp} - P_i^{calc} \tag{9}$$

$$\Delta Q_i = Q_i^{esp} - Q_i^{calc} \tag{10}$$

Em uma rede com n barras, o vetor $\Delta\boldsymbol{P}$ é composto por n linhas menos uma (referente à barra *swing)* e $\Delta\boldsymbol{Q}$ pela quantidade de barras do tipo PQ.

Os valores P_i^{esp} e Q_i^{esp} são as potências líquidas injetadas na barra i.

Para obter os valores de P_i^{calc} e Q_i^{calc},tomamos a equação (2) e fazemos as seguintes substituições:

$$\begin{cases} Y_{ik} = G_{ik} + jB_{ik} \\ \dot{V}_i = V_i e^{j\theta_i} \\ \dot{V}_k = V_k e^{j\theta_k} \end{cases}$$

Após algumas manipulações algébricas, temos:

$$P_i^{calc} = V_i \sum_{k=1}^{n} V_k (G_{ik} cos\,\theta_{ik} + B_{ik}\, sen\,\theta_{ik}) \tag{11}$$

$$Q_i^{calc} = V_i \sum_{k=1}^{n} V_k (G_{ik}\, sen\,\theta_{ik} - B_{ik} cos\,\theta_{ik}) \tag{12}$$

A partir das equações (11) e (12) obtemos as equações referentes às submatrizes jacobianas $\boldsymbol{J}_1$, $\boldsymbol{J}_2$, $\boldsymbol{J}_3$ e $\boldsymbol{J}_4$, demonstradas abaixo. As expressões onde j é igual a i são usadas no cálculo dos valores da diagonal. Os demais valores fora da diagonal são obtidos pelas expressões onde j é igual a k.

$$J_1 = \frac{\partial P_i}{\partial \theta_j} \quad \begin{cases} j = k: & \dfrac{\partial P_i}{\partial \theta_k} = V_i V_k (G_{ik}\, sen\,\theta_{ik} - B_{ik} cos\,\theta_{ik}) \qquad (13) \\ j = i: & \dfrac{\partial P_i}{\partial \theta_i} = V_i \sum_{k \neq i} V_k (-G_{ik}\, sen\,\theta_{ik} + B_{ik} cos\,\theta_{ik}) \qquad (14) \end{cases}$$

$$J_2 = \frac{\partial P_i}{\partial V_j} \begin{cases} j = k: & \frac{\partial P_i}{\partial V_k} = V_i (G_{ik} \cos\theta_{ik} + B_{ik} \operatorname{sen}\theta_{ik}) \quad (15) \\ j = i: & \frac{\partial P_i}{\partial V_i} = 2 G_{ii} V_i + \sum_{k \neq i} V_k (G_{ik} \cos\theta_{ik} + B_{ik} \operatorname{sen}\theta_{ik}) \quad (16) \end{cases}$$

$$J_3 = \frac{\partial Q_i}{\partial \theta_j} \begin{cases} j = k: & \frac{\partial Q_i}{\partial \theta_k} = V_i V_k (-G_{ik} \cos\theta_{ik} - B_{ik} \operatorname{sen}\theta_{ik}) \quad (17) \\ j = i: & \frac{\partial Q_i}{\partial \theta_i} = V_i \sum_{k \neq i} V_k (G_{ik} \cos\theta_{ik} + B_{ik} \operatorname{sen}\theta_{ik}) \quad (18) \end{cases}$$

$$J_4 = \frac{\partial Q_i}{\partial V_j} \begin{cases} j = k: & \frac{\partial Q_i}{\partial V_k} = V_i (G_{ik} \operatorname{sen}\theta_{ik} - B_{ik} \cos\theta_{ik}) \quad (19) \\ j = i: & \frac{\partial Q_i}{\partial V_i} = -2 B_{ii} V_i + \sum_{k \neq i} V_k (G_{ik} \operatorname{sen}\theta_{ik} - B_{ik} \cos\theta_{ik}) \quad (20) \end{cases}$$

O objetivo é calcular as variáveis de estado de forma que

$$\Delta P_i < \varepsilon_P$$
$$\Delta Q_i < \varepsilon_Q$$

sendo ε uma tolerância especificada.

A resolução do problema segue então os seguintes passos:

i) Estimar um valor inicial para as variáveis de estado θ_i e V_i ou tomar as variáveis de estado obtidas em uma iteração anterior;

ii) Calcular ΔP_i e ΔQ_i, considerando as equações (9) e (10), respectivamente, e comparar com a tolerância ε especificada. Caso esteja fora da tolerância, o processo deverá prosseguir;

iii) Calcular J_1, J_2, J_3 e J_4 utilizando as equações (13) a (20), conforme o caso, substituindo os valores estimados das variáveis de estado. Inverter a matriz obtida;

iv) Utilizar os valores obtidos de ΔP_i e ΔQ_i para calcular $\Delta\theta_i$ e ΔV_i através da equação (8). As novas estimativas das variáveis de estado passam a ser

$$\theta_i^{(\nu+1)} = \theta_i^{(\nu)} + \Delta\theta_i^{(\nu)}$$
$$V_i^{(\nu+1)} = V_i^{(\nu)} + \Delta V_i^{(\nu)}$$

sendo (ν) a iteração atual e $(\nu + 1)$ a iteração seguinte.

v) Voltar ao passo ii para realizar uma nova iteração.

2.4 Método de Newton-Raphson com Jacobiano constante

No método anterior, a matriz jacobiana se altera a cada iteração. Nesta versão, mantém-se a matriz jacobiana original durante todo o processo. Assim, o número de iterações, para uma dada tolerância de convergência, em geral é maior que no método anterior, mas cada iteração se torna mais rápida por não se calcular e inverter a matriz jacobiana a cada passo.

2.5 Métodos desacoplados

Os métodos desacoplados consideram o fato de as sensibilidades $\partial P/\partial\theta$ e $\partial Q/\partial V$ serem mais intensas que $\partial P/\partial V$ e $\partial Q/\partial\theta$. Estes métodos aproximam o cálculo das derivadas, mas mantêm a integridade do modelo da rede e, por isso, não afetam a solução final do fluxo de carga [1]. O desacoplamento possibilita que os subproblemas $P\theta$ e QV sejam resolvidos alternadamente, ou seja, dentro de uma mesma iteração são sempre utilizados os valores mais atualizados de θ e V.

2.5.1 Newton-Raphson desacoplado

Neste primeiro caso, as submatrizes jacobianas J_2 e J_3, já apresentadas na seção 2.3, são consideradas nulas. Todo o restante do processo é semelhante ao método de Newton-Raphson, com a diferença que as variáveis θ e V são atualizadas a cada meia iteração. As equações a serem usadas no processo passam a ser

$$\Delta P = J_1 \Delta \theta \tag{21}$$

$$\Delta Q = J_4 \Delta V \tag{22}$$

Portanto, a resolução do problema passa a ter os seguintes passos:

i) Estimar um valor para as variáveis de estado θ_i e V_i ;

ii) Calcular ΔP_i e comparar com a tolerância ε especificada. Caso esteja fora da tolerância, o processo para cálculo de $P\theta$ deverá prosseguir;

iii) Calcular J_1 utilizando as equações (13) e (14), substituindo os valores estimados atualizados das variáveis de estado. Inverter a matriz obtida;

iv) Utilizar os valores obtidos de ΔP_i para calcular $\Delta\theta_i$ através da equação (21). A nova estimativa da variável de estado θ_i passa a ser

$$\theta_i^{(\nu+1)} = \theta_i^{(\nu)} + \Delta\theta_i^{(\nu)}$$

v) Calcular ΔQ_i utilizando o valor atualizado de θ_i e comparar com a tolerância ε especificada. Caso esteja fora da tolerância, o processo para calcular QV deverá prosseguir;

vi) Calcular J_4 utilizando as equações (19) e (20), substituindo os valores estimados atualizados das variáveis de estado. Inverter a matriz obtida;

vii) Utilizar os valores obtidos de ΔQ_i para calcular ΔV_i através da equação (22). A nova estimativa da variável de estado V_i passa a ser

$$V_i^{(\nu+1)} = V_i^{(\nu)} + \Delta V_i^{(\nu)}$$

viii) Voltar ao passo ii para realizar uma nova iteração.

Como a velocidade de convergência dos subproblemas *Pθ* e *QV* podem ser diferentes, obtém-se algumas vantagens computacionais iterando-se apenas a parte não resolvida, mantendo-se, porém, o teste de convergência da parte convergida.

2.5.2 Desacoplado rápido

Neste método, as submatrizes jacobianas J_2 e J_3 também são consideradas nulas. Porém, algumas mudanças simplificadoras são feitas.

Primeiramente tomamos as equações (21) e (22) e fazemos as alterações abaixo:

$$\Delta P = J_1(\Delta\theta) = H(\Delta\theta) \tag{23}$$

$$\Delta Q = J_4'\left(\frac{\Delta V}{V}\right) = L\left(\frac{\Delta V}{V}\right) \tag{24}$$

sendo *H* igual à submatriz jacobiana J_1 e *L* igual à submatriz jacobiana J_4 multiplicada pela tensão *V*.

As equações referentes a *H* e *L* são apresentadas abaixo, considerando as equações (13), (14), (19) e (20):

$$H = \begin{cases} H_{ik} = V_i V_k (G_{ik}\, sen\,\theta_{ik} - B_{ik}\cos\theta_{ik}) & (25) \\ H_{ii} = -V_i \sum_{k\neq i} V_k (G_{ik}\, sen\,\theta_{ik} - B_{ik}\cos\theta_{ik}) & (26) \end{cases}$$

$$L = \begin{cases} L_{ik} = V_i V_k (G_{ik}\, sen\,\theta_{ik} - B_{ik}\cos\theta_{ik}) & (27) \\ L_{ii} = -2\,B_{ii} V_i^2 + V_i \sum_{k\neq i} V_k (G_{ik}\, sen\,\theta_{ik} - B_{ik}\cos\theta_{ik}) & (28) \end{cases}$$

O método considera as seguintes hipóteses simplificadoras:

$$\begin{cases} \cos\theta_{ik} \approx 1 \\ G_{ik}\, sen\,\theta_{ik} \ll B_{ik} \\ Q_i \ll B_{ii} V_i^2 \end{cases}$$

Após as simplificações, as equações referentes a *H* e a *L* passam a ser:

$$H = \begin{cases} H_{ik} = -B_{ik} V_i V_k & (29) \\ H_{ii} = -B_{ii} V_i^2 & (30) \end{cases}$$

$$L = \begin{cases} L_{ik} = -B_{ik} V_i V_k & (31) \\ L_{ii} = -B_{ii} V_i^2 & (32) \end{cases}$$

Substituindo *H* e *L* nas equações (23) e (24) temos:

$$\Delta P = V B' V \Delta\theta \tag{33}$$

$$\Delta Q = V B'' V \left(\frac{\Delta V}{V} \right) \tag{34}$$

Considerando V_k igual a 1 pu na equação (33), e simplificando as tensões na equação (34), obtemos as equações utilizadas neste método:

$$\frac{\Delta P}{V} = B' \Delta\theta \tag{35}$$

$$\frac{\Delta Q}{V} = B'' \Delta V \tag{36}$$

Considerações finais do método:

i) No cálculo de *B*', os elementos que influem na tensão não são considerados, tais como elementos *shunt* e *taps* de transformadores. As resistências série dos ramos também não são consideradas;

ii) No cálculo de *B*'' não são representados os transformadores defasadores.

A resolução do problema é similar à do método de Newton-Raphson desacoplado, com exceção dos passos (iii) e (vi). As matrizes *B*' e *B*'' são mantidas constantes em todo o processo iterativo.

3 Fluxo de Potência Ativa Linear (Modelo CC)

3.1 Considerações iniciais

O Modelo CC é baseado no acoplamento entre as variáveis *P* (potência ativa) e θ (ângulo da tensão). O método é assim chamado pelo fato da relação entre essas variáveis ser do mesmo tipo da relação existente entre fluxo de corrente e queda de tensão em um circuito de corrente contínua. Neste método, são ignoradas as equações relativas à parte reativa do problema, e não são levadas em consideração as magnitudes das tensões nodais e os taps dos transformadores. Por isso, o método não pode substituir por completo os outros métodos não-lineares, mas tem grande aplicação em fases preliminares de estudos onde um grande número de casos deve ser analisado.

Este modelo linearizado permite estimar, com baixo custo computacional e precisão aceitável para muitas aplicações, a distribuição dos fluxos de potência ativa em linhas de transmissão e transformadores. No entanto, não é aplicável para sistemas de distribuição em baixa tensão, pois os fluxos neste sistema dependem muito das quedas de tensão.

Uma característica importante do modelo linearizado é o fato dele sempre fornecer uma solução, mesmo para os problemas que não podem ser resolvidos pelos métodos convencionais de cálculo do fluxo de carga [1,5]. Embora o resultado não seja correto, possui a vantagem de dar uma ideia aproximada sobre o quanto está sendo excedida a capacidade de transmissão da linha, enquanto os outros modelos não-lineares simplesmente informam que não há solução viável.

3.2 Resolução do problema [1,4]

Inicialmente tomamos a equação abaixo, já deduzida no método de Newton-Raphson (equação 11):

$$P_i = V_i \sum_{k=1}^{n} V_k (G_{ik} \cos\theta_{ik} + B_{ik} \operatorname{sen}\theta_{ik})$$

Consideraremos então as seguintes hipótese simplificadoras:

$$\begin{cases} G_{ik} \ll B_{ik} \\ \cos\theta_{ik} \approx 1 \\ \operatorname{sen}\theta_{ik} \approx \theta_{ik} = \theta_i - \theta_k \\ V_i \approx V_k \approx 1\,pu \end{cases}$$

Logo,

$$P_i = \sum_{k=1}^{n} B_{ik}\theta_{ik} \quad , \quad i = 1, n \tag{37}$$

Desenvolvendo, temos:

$$P_i = B_{i1}(\theta_i - \theta_1) + B_{i2}(\theta_i - \theta_2) + \ \ldots \ + B_{ii}(\theta_i - \theta_i) + \ \ldots \ + B_{in}(\theta_i - \theta_n)$$

$$P_i = -B_{i1}\theta_i - B_{i2}\theta_2 - \ \ldots \ - B_{in}\theta_n + (B_{i1} + B_{i2} + \ \ldots \ + B_{in})\theta_i$$

Sendo $(B_{i1} + B_{i2} + \ldots + B_{in})$ igual à $-B_{ii}$, temos:

$$P_i = \sum_{k=1}^{n} -B_{ik}\,\theta_k \quad ou \quad P_i = \sum_{k=1}^{n} B'_{ik}\,\theta_k \quad , \quad B'_{ik} = -B_{ik} \tag{38}$$

sendo

$$B'_{ik} = -B_{ik} = b_{ik} = -\frac{1}{x_{ik}} \quad , i \neq k$$

$$B'_{ii} = -B_{ii} = \sum_{k=1 \neq i}^{n} B_{ik} = \sum_{k=1 \neq i}^{n} -b_{ik} = \sum_{k=1 \neq i}^{n} \frac{1}{x_{ik}}$$

$$z_{ik} = j\,x_{ik} \quad ; \quad y_{ik} = \frac{1}{z_{ik}} = jb_{ik}$$

Matricialmente, temos:

$$P = B'\theta \tag{39}$$

Inicialmente, as perdas do sistema são desprezadas. Logo, a equação relativa à barra *swing* não é considerada. Retira-se, então, a linha e a coluna referentes à barra *swing* e calculamos

$$X = [\,B'\,]^{-1}$$

A equação utilizada no cálculo do fluxo de potência pelo Modelo CC é, então, dada por

$$\theta = X\,P \tag{40}$$

Quando ocorrerem variações na geração ou na carga, pode-se atualizar os ângulos das tensões considerando-se um novo vetor *P*. A matriz *X* também será modificada se ocorrer uma retirada (ou inclusão) de linha ou transformador no sistema.

Após a determinação dos ângulos, o fluxo de potência ativa no sistema pode ser determinado por

$$P_{ik} = \frac{\theta_i - \theta_k}{x_{ik}} \tag{41}$$

3.3 Cálculo das perdas de transmissão em uma linha *i-k* [1,4]

Seja a expressão do fluxo de potência ativa apresentada abaixo:

$$P_{ik} = -t_{ik}G_{ik}V_i^2 + V_iV_k(G_{ik}cos\theta_{ik} + B_{ik}sen\theta_{ik}) \tag{42}$$

Analogamente, temos:

$$P_{ki} = -t_{ki}G_{ki}V_k^2 + V_iV_k(G_{ki}cos\theta_{ki} + B_{ki}sen\theta_{ki}) \tag{43}$$

Sendo

$$Perdas = P_{ik} + P_{ki}$$

temos

$$Perdas = -G_{ik}(t_{ik}V_i^2+\frac{1}{t_{ik}}V_k^2) + V_i V_k(2G_{ik}cos\theta_{ik}) \quad (44)$$

Sabendo-se que $G_{ik} = -g_{ik}$ e $t_{ik} = 1$ *(LT),* resulta:

$$Perdas = g_{ik}(V_i^2+V_k^2-2V_i V_k cos\theta_{ik}) \quad (45)$$

Considerando as aproximações

$$\begin{cases} V_i=V_k=1 \\ cos\,\theta_{ik}\approx 1-\frac{\theta_{ik}^2}{2} \end{cases}$$

resulta:

$$Perdas = g_{ik}\left[2-2\left(1-\frac{\theta_{ik}^2}{2}\right)\right] = g_{ik}\theta_{ik}^2 \quad (46)$$

3.4 Representação das perdas no modelo CC [1,4]

Como foi apresentado anteriormente, a equação (39) utilizada no cálculo do fluxo de potência não leva em consideração as perdas que, na prática, ocorrem na transmissão. Como conseqüência disso, os fluxos de potência nas linhas conectadas à barra *swing* podem apresentar grandes erros. Portanto, faremos uma modificação na referida equação, para que as perdas passem a ser consideradas.

Da equação (11), separando-se o termo referente à barra local *i*, temos:

$$P_i = G_{ii}V_i^2+V_i\sum_{k=1\neq i}^{n} V_k(G_{ik}cos\theta_{ik}+B_{ik}sen\theta_{ik}) \quad (47)$$

Considerando-se:

$$\begin{cases} V_i=V_k=1 \quad \forall i,k \\ G_{ik}=-g_{ik} \\ G_{ii}=\sum_{k=1\neq i}^{n} g_{ik} \\ B_{ik}\approx x_{ik}^{-1} \end{cases}$$

resulta

$$P_i = \sum_{k=1\neq i}^{n} g_{ik}(1-cos\theta_{ik})+\sum_{k=1\neq i}^{n} x_{ik}^{-1}sen\theta_{ik} \quad (48)$$

Fazendo as aproximações

$$\begin{cases} cos\,\theta_{ik} \approx 1 - \dfrac{\theta_{ik}^2}{2} \\ sen\,\theta_{ik} \approx \theta_{ik} \end{cases}$$

temos

$$P_i = \sum_{k=1 \neq i}^{n} g_{ik} \frac{\theta_{ik}^2}{2} + \sum_{k=1 \neq i}^{n} x_{ik}^{-1} \theta_{ik} \tag{49}$$

Assim,

$$P_i - \frac{1}{2} \sum_{k=1 \neq i}^{n} g_{ik}\, \theta_{ik}^2 = \sum_{k=1 \neq i}^{n} x_{ik}^{-1}\, \theta_{ik} \tag{50}$$

Verifica-se que o lado esquerdo desta equação é dado pela injeção líquida de potência ativa na barra *i* menos a metade das perdas ativas de todas as linhas adjacentes a esta barra.

Portanto, para incluir as perdas no cálculo dos ângulos, obtemos as perdas em cada linha do sistema e as dividimos entre suas barras terminais. Assim, para uma linha entre as barras *i* e *k*, metade das perdas passam a ser consideradas como uma carga adicional na barra *i* e a outra metade como uma carga adicional na barra *k*. O Modelo CC passa então a ser:

$$\boldsymbol{P} + \boldsymbol{P}_{perdas} = \boldsymbol{B}'\boldsymbol{\theta} \tag{51}$$

O sistema (51) pode ser resolvido com o seguinte procedimento:

i) Calcular uma solução temporária $\tilde{\theta}$ resolvendo o sistema

$$\boldsymbol{P} = \boldsymbol{B}'\tilde{\boldsymbol{\theta}} \tag{52}$$

ii) Calcular as perdas aproximadas considerando-se $\tilde{\theta}$ e distribuí-las por todo o sistema como cargas adicionais;

iii) Resolver o sistema (51) e determinar θ, observando que a matriz $\boldsymbol{B}'$ é a mesma do item i.

4 Fluxo de Potência Ativa Linear – Resultados

Foram realizadas simulações para o cálculo dos fluxos de potência em dois sistemas: o sistema 'Stevenson'[2] e o sistema 'Cigré'[3]. As simulações foram feitas no programa descrito no Apêndice C e no programa ANAREDE [6]. Também foi calculado o erro percentual entre os valores obtidos nos dois programas, como indicado abaixo:

$$Erro\ (\%) = \frac{|Fluxo(CC) - Fluxo(NR)|}{Fluxo(NR)} x\ 100$$

onde

Fluxo (CC): fluxo de potência ativa obtido com o modelo CC

Fluxo (NR): fluxo de potência ativa obtido com o método de Newton-Raphson (ANAREDE)

4.1 Sistema 'Stevenson'

A figura 4.1 apresenta um sistema com cinco barras, dois geradores, uma barra infinita, quatro linhas de transmissão e dois transformadores [2]. Os dados deste sistema são apresentados no Apêndice A. A tabela 4.1 contém o erro percentual entre os valores obtidos no ANAREDE e os obtidos no programa do Apêndice C sem considerar as perdas. A tabela 4.2 também contém o erro percentual entre os resultados dos dois programas, porém considerando as perdas.

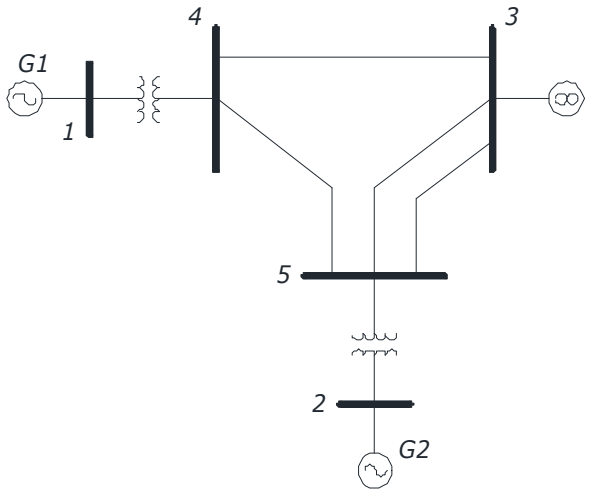

Figura 4.1 – Diagrama unifilar do sistema 'Stevenson'

Tabela 4.1 - Fluxos de potência simulados e erros percentuais (sem considerar as perdas)

Ramo entre as barras	Modelo CC sem considerar as perdas (MW)	Newton-Raphson - ANAREDE (MW)	Erro (%)
1 - 4	350,00	350,00	0,00
2 - 5	185,00	185,00	0,00
3 - 4	-210,65	-207,60	1,38
3 - 5	-174,35	-173,00	0,77
4 - 5	39,35	39,40	0,13

Tabela 4.2 - Fluxos de potência simulados e erros percentuais (considerando as perdas)

Ramo entre as barras	Modelo CC considerando as perdas (MW)	Newton-Raphson - ANAREDE (MW)	Erro (%)
1 - 4	350,00	350,00	0,00
2 - 5	185,00	185,00	0,00
3 - 4	-209,29	-207,60	0,81
3 - 5	-173,34	-173,00	0,20
4 - 5	39,07	39,40	0,84

4.2 Sistema 'Cigré'

A figura 4.2 apresenta um sistema com dez barras, sete geradores e treze linhas de transmissão [3]. Os dados deste sistema são apresentados no Apêndice B. A tabela 4.3 contém o erro percentual entre os valores obtidos no ANAREDE e os obtidos no programa do Apêndice 3 sem considerar as perdas. A tabela 4.4 também contém o erro percentual entre os resultados dos dois programas, porém considerando as perdas.

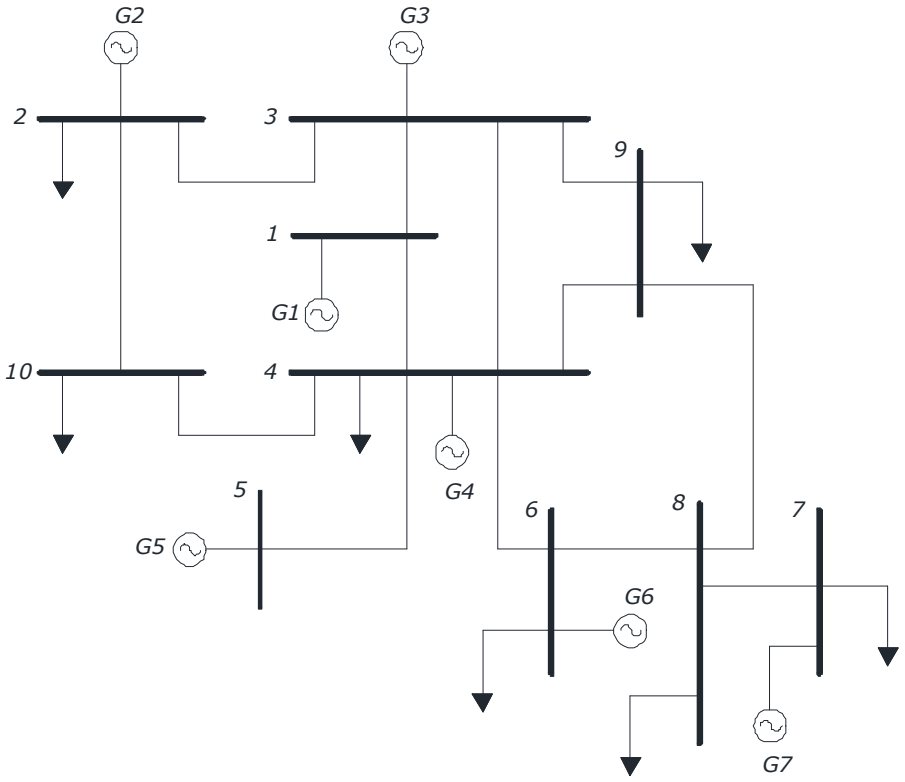

Figura 4.2 – Diagrama unifilar do sistema 'Cigré'

Tabela 4.3 - Fluxos de potência simulados e erros percentuais (sem considerar as perdas)

Ramo entre as barras	Modelo CC sem considerar as perdas (MW)	Newton-Raphson / ANAREDE (MW)	Erro (%)
1 - 3	62,63	64,00	2,14
1 - 4	154,37	153,00	0,89
2 - 3	-81,64	-79,80	2,25
2 - 10	1,64	-0,20	(*)
3 - 4	56,90	54,50	4,22
3 - 9	180,10	182,40	1,26
4 - 5	-230,00	-227,80	0,96
4 - 6	-42,92	-40,50	5,64
4 - 9	28,82	31,00	7,03
4 - 10	88,36	91,50	3,43
6 - 8	37,08	39,40	5,89
7 - 8	84,00	84,00	0,00
8 - 9	21,08	22,20	5,04

Tabela 4.4 - Fluxos de potência simulados e erros percentuais (considerando as perdas)

Ramo entre as barras	Modelo CC considerando as perdas (MW)	Newton-Raphson / ANAREDE (MW)	Erro (%)
1 - 3	64,03	64,00	0,004
1 - 4	151,65	153,00	0,88
2 - 3	-81,68	-79,80	2,36
2 - 10	0,36	-0,20	(*)
3 - 4	54,35	54,50	0,27
3 - 9	180,52	182,40	1,03
4 - 5	-229,00	-227,80	0,53
4 - 6	-41,21	-40,50	1,75
4 - 9	29,98	31,00	3,29
4 - 10	90,24	91,50	1,38
6 - 8	38,61	39,40	2,01
7 - 8	83,59	84,00	0,49
8 - 9	21,57	22,20	2,84

(*) Apesar do erro percentual calculado para esta linha ser bastante alto, a diferença entre os fluxos, em valor absoluto, é compatível com os valores obtidos nas demais linhas.

Observando os resultados nos dois sistemas, verifica-se que o erro entre os valores de fluxo obtidos no Modelo CC e os obtidos com o método de Newton-Raphson é pequeno (menor que oito por cento). Uma vantagem disso é que com baixo custo computacional é possível realizar alguns estudos de planejamento com uma precisão aceitável quando o comportamento reativo do sistema ainda não está sendo considerado.

Levando-se em conta as perdas na transmissão, em geral há uma diminuição dos erros percentuais nas linhas conectadas à barra *swing*. Isto pode ser verificado nas linhas 3-4 e 3-5 do sistema 'Stevenson' e nas linhas 1-4, 3-4, 4-5, 4-6, 4-9 e 4-10 do sistema 'Cigré'.

Referências Bibliográficas

[1] MONTICELLI, A. J. **Fluxo de carga em redes de energia elétrica**. São Paulo: Ed. Edgard Blücher, 1983.

[2] GRAINGER, J. J. ; STEVENSON JR., W.D. **Power system analysis**. New York: McGraw-Hill, 1994.

[3] DAHL, J. M. **Cálculo de índices de segurança em sistemas de energia elétrica baseado em simulação no domínio do tempo**. 2006. 109f. Dissertação (Mestrado em Engenharia Elétrica), Pontifícia Universidade Católica do Rio de Janeiro – PUC Rio.

[4] Notas de aula do curso de Análise de Redes Elétricas, Pontifícia Universidade Católica do Rio de Janeiro – PUC Rio, 2010.

[5] Notas de aula do curso de Análise de Sistemas de Energia Elétrica II, Pontifícia Universidade Católica do Rio de Janeiro – PUC Rio, 2009.

[6] Cepel, Programa ANAREDE – Análise de Redes – Versão Universitária, 2001.

[7] ELGERD, O. I. **Introdução à teoria de sistemas de energia elétrica**. São Paulo: Ed. McGraw-Hill do Brasil Ltda, 1978.

[8] CELES, W.; CERQUEIRA, R.; RANGEL, J.L. **Introdução a estrutura de dados**: com técnicas de programação em C. Rio de Janeiro: Elsevier, 2004 - 5ª Reimpressão.

Apêndice A

Dados do Sistema 'Stevenson' [2]

Os dados das barras estão na tabela A.1 e os dados das linhas e transformadores estão na tabela A.2. A base utilizada é 100 MVA.

Tabela A.1 - Dados das barras

Barra	Tipo	Geração (MW)	Carga (MW)
01	PV	350,0	-
02	PV	185,0	-
03	SWING	-	-
04	PQ	-	100,0
05	PQ	-	50,0

Tabela A.2 - Dados das linhas e transformadores

Da barra	Para barra	Circuito (nº)	Resistência (%)	Reatância (%)	Susceptância (MVAr)
1	4	1	0,0000	2,2000	0,00
2	5	1	0,0000	4,0000	0,00
3	4	1	0,7000	4,0000	8,20
3	5	1	0,8000	4,7000	9,80
3	5	2	0,8000	4,7000	9,80
4	5	1	1,8000	11,000	22,60

Apêndice B

Dados do Sistema 'Cigré' [3]

Os dados das barras estão na tabela B.1 e os dados das linhas estão na tabela B.2. A base utilizada é 100 MVA.

Tabela B.1: Dados das barras

Barra	Tipo	Geração (MW)	Carga (MW)
01	PV	217,0	-
02	PV	120,0	200,0
03	PV	256,0	-
04	SWING	299,2	650,0
05	PV	230,0	-
06	PV	160,0	80,0
07	PV	174,0	90,0
08	PQ	-	100,0
09	PQ	-	230,0
10	PQ	-	90,0

Tabela B.2: Dados das linhas

Da barra	Para barra	Resistência (%)	Reatância (%)	Susceptância (MVAr)
1	3	0,9877	4,8395	20,25
1	4	0,9877	4,8395	10,125
2	3	4,5037	12,365	20,25
2	10	1,6395	6,3803	30,375
3	4	1,1852	7,8025	30,375
3	9	1,1457	5,5309	20,25
4	5	0,3951	1,9753	20,25
4	6	0,7506	1,9753	121,50
4	9	4,8790	19,161	20,25
4	10	1,6395	6,5185	30,375
6	8	1,8765	6,2815	20,25
7	8	1,1852	7,8025	30,375
8	9	4,8790	19,161	20,25

Apêndice C

Código do programa para cálculo do Fluxo de Potência através do Modelo CC [8]

Função principal:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "op_matrizes.h"
#include "arquivos.h"

int main (void)
{
	int nos, linhas, i, ref, m, n = 0, r, s, x, y, z, w, mm, nn = 0, rr, ss;
	float base = 100.0, fluxo_pot, perdas, fluxo_pot_com_perdas;
	float *pot_ger, *pot_dem, *pot_liq, *resist, *reat, *pot_liq_sem_ref, *calc_ang, *vetor_ang,
		*pot_mais_perdas, *pot_mais_perdas_sem_ref, *calc_ang1, *vetor_ang1;
	float **matriz_R, **matriz_X, **matriz_G, **matriz_B, **B_linha, **B_linha_inv, **matriz_perdas;
	int *de, *para;
	char arq_dados[15];
	FILE* result;

	result = fopen("Resultados.odt", "a");

	printf ("\nDADOS DO SISTEMA\n");
	printf ("----------------\n");

	printf ("Nome do arquivo de dados: ");
	scanf ("%s", arq_dados);

	while ((fopen(arq_dados, "r")) == NULL)
	{
		printf ("\nArquivo invalido. Tente novamente: ");
		scanf ("%s", arq_dados);
	}

	nos = conta_barras (arq_dados, "BARRA");

	pot_ger = vetor_dados_reais (arq_dados, nos, "( Pg)");
	per_unit_vet (pot_ger, base, nos);

	pot_dem = vetor_dados_reais (arq_dados, nos, "( Pl)");
	vetor_negativo (pot_dem, nos);
	per_unit_vet (pot_dem, base, nos);

	pot_liq = soma_vetor (pot_ger, pot_dem, nos);

	linhas = conta_linhas (arq_dados, "(De)", "9999");

	resist = vetor_dados_reais (arq_dados, linhas, "( R% )");
	per_unit_vet (resist, base, linhas);

	reat = vetor_dados_reais (arq_dados, linhas, "( X% )");
	per_unit_vet (reat, base, linhas);

	de = vetor_dados_int (arq_dados, linhas, "(De)");
	para = vetor_dados_int (arq_dados, linhas, "(Pa)");

	matriz_R = cria_matriz (nos);

	matriz_X = cria_matriz (nos);
```

```
for (i = 0; i < linhas; i++)
{
	if (matriz_R[de[i]-1][para[i]-1] != 0.00)
	{
		matriz_R[de[i]-1][para[i]-1] = paralelo (matriz_R[de[i]-1][para[i]-1], resist[i]);
		matriz_X[de[i]-1][para[i]-1] = paralelo (matriz_X[de[i]-1][para[i]-1], reat[i]);
	}
	else
	{
		matriz_R[de[i]-1][para[i]-1] = resist[i];
		matriz_X[de[i]-1][para[i]-1] = reat[i];
	}
	matriz_R[para[i]-1][de[i]-1] = matriz_R[de[i]-1][para[i]-1];
	matriz_X[para[i]-1][de[i]-1] = matriz_X[de[i]-1][para[i]-1];
}

matriz_G = calc_condutancia (matriz_R, matriz_X, nos);

matriz_B = inv_elem_imag_fora_diag (matriz_X, nos);

matriz_negativa (matriz_B, nos);

calc_elem_diagonal (matriz_B, nos);

matriz_negativa (matriz_B, nos);

printf ("\nBarra de referencia do sistema: ");
scanf ("%d", &ref);
if (ref > nos)
{
	printf("\nO sistema so possui %d barras. Digite outra: ", nos);
	scanf ("%d", &ref);
}

B_linha = retira_ref_mat (matriz_B, nos, ref);

B_linha_inv = cria_matriz (nos);
inverte_matriz (B_linha, nos-1, B_linha_inv);

pot_liq_sem_ref = retira_ref_vet (pot_liq, nos, ref);

calc_ang = multiplica_mat_vet (B_linha_inv, pot_liq_sem_ref, nos-1);

vetor_ang = cria_vetor (nos);

for (m = 0; m < nos; m++)
{
	if (m != (ref-1))
	{
		preenche_vetor (vetor_ang, m, calc_ang[n]);
		n++;
	}
}
printf ("\nFLUXOS DE POTENCIA NO SISTEMA SEM CONSIDERAR AS PERDAS:\n\n");
fprintf(result, "\n\nFLUXOS DE POTENCIA NO SISTEMA SEM CONSIDERAR AS PERDAS\n");

for (r = 0; r < nos ; r++)
    for (s = 0; s < nos ; s++)
	if (s > r)
	   if (matriz_X[r][s] != 0)
	   {
		fluxo_pot = ((vetor_ang[r] - vetor_ang[s])/matriz_X[r][s])*base;
		printf ("\n P%2.0d-%2.0d = %9.2f", r+1, s+1, fluxo_pot);
		fprintf(result, "\nP%2.0d-%2.0d = %9.2f", r+1, s+1, fluxo_pot);
	   }
```

```
        // CALCULO DAS PERDAS

        matriz_perdas = cria_matriz (nos);
        for (x = 0; x < nos ; x++)
        {
            for (y = 0; y < nos ; y++)
                if (y > x)
                {
                        perdas = (vetor_ang[x] - vetor_ang[y])*(vetor_ang[x] - vetor_ang[y])*matriz_G[x][y];
                        matriz_perdas[x][y] = perdas/2.0;
                        matriz_perdas[y][x] = perdas/2.0;
                }
        }

        pot_mais_perdas = cria_vetor (nos);

        for (z = 0; z < nos; z++)
        {
                pot_mais_perdas[z] = pot_liq[z];

                for (w = 0; w < nos; w++)
                        pot_mais_perdas[z] += -matriz_perdas[z][w];
        }

        pot_mais_perdas_sem_ref = retira_ref_vet (pot_mais_perdas, nos, ref);

        calc_ang1 = multiplica_mat_vet (B_linha_inv, pot_mais_perdas_sem_ref, nos-1);

        vetor_ang1 = cria_vetor (nos);
        for (mm = 0; mm < nos; mm++)
                if (mm != (ref-1))
                {
                        preenche_vetor (vetor_ang1, mm, calc_ang1[nn]);
                        nn++;
                }

        printf ("\nFLUXOS DE POTENCIA NO SISTEMA CONSIDERANDO AS PERDAS:\n\n");
        fprintf(result, "\n\nFLUXOS DE POTENCIA NO SISTEMA CONSIDERANDO AS PERDAS\n");

        for (rr = 0; rr < nos ; rr++)
            for (ss = 0; ss < nos ; ss++)
                if (ss > rr)
                   if (matriz_X[rr][ss] != 0)
                   {
                        fluxo_pot_com_perdas = ((vetor_ang1[rr] - vetor_ang1[ss])/matriz_X[rr][ss])*base;
                        printf ("\n P%2.0d-%2.0d = %9.2f", rr+1, ss+1, fluxo_pot_com_perdas);
                        fprintf(result, "\nP%2.0d-%2.0d = %9.2f", rr+1, ss+1, fluxo_pot_com_perdas);
                   }

        printf("\n\n");
        system("PAUSE");

        return 0;

}
```

Funções auxiliares:

Biblioteca para manipulação de arquivos - 'arquivos.h'

```
int conta_barras (char* arquivo, char* palavra_chave)
{
	int nlinhas = 0, n = 0;
	char s[76];

	FILE* fp = fopen (arquivo, "rt");

	if (fp == NULL)
		printf ("Erro na abertura do arquivo!\n");

	while (fgets (s, 16, fp) != NULL)
	{
		n++;
		if (strstr (s, palavra_chave) != NULL)
			nlinhas++;
	}
	fclose (fp);
	return nlinhas;
}

int conta_linhas (char* arquivo, char* palavra_chave1, char* palavra_chave2)
{
	int nlinha = 0, n = 0, c;
	char linha[80];
	FILE* fp = fopen (arquivo, "r");

	if (fp == NULL)
		printf ("Erro na abertura do arquivo!\n");

	while (fgets (linha, 80, fp) != NULL)
	{
		n++;
		if (strstr (linha, palavra_chave1) != NULL)
		{
			while ((c = fgetc(fp)) != '\n')
			{
				if (c == '\n')
					fgetc(fp);
			}

			while (strstr (linha, palavra_chave2) == NULL)
			{
				nlinha ++;
				fgets (linha, 80, fp);
			}
			break;
		}
	}
	fclose (fp);
	return nlinha;
}

float* vetor_dados_reais (char* arquivo, int nb, char* palavra_chave)
{
	float* dados = cria_vetor (nb);
	int n = 0, a, b = 0, j, col = 0, k, i, d, c, dig = 0;
	char linha[80];
	FILE* fp = fopen (arquivo, "r");

	if (fp == NULL)
		printf ("Erro na abertura do arquivo!\n");
```

```
	for (a = 0; palavra_chave[a] != '\0'; a++)
		dig++;

	while (fgets (linha, 80, fp) != NULL)
	{
		n++;
		if (strstr (linha, palavra_chave) != NULL)
		{
			for (j = 0; &linha[j] != strstr (linha, palavra_chave); j++)
				col++;

			for (k = 0; k < col; k++)
				fgetc(fp);
			if (fgets (linha, (dig+1), fp) != NULL)
				n++;

			for (i = 0; i < nb; i++)
			{
				sscanf (linha, "%f", &dados[i]);

				while ((c = fgetc(fp)) != '\n')
				{
					if (c == '\n')
						fgetc(fp);
				}

				for (k = 0; k < col; k++)
					fgetc(fp);

				if (fgets (linha, (dig+1), fp) != NULL)
					n++;
			}
			break;
		}
	}
	fclose (fp);
	return dados;
}

int* vetor_dados_int (char* arquivo, int nb, char* palavra_chave)
{
	int* dados = cria_vetor_int (nb);
	int n = 0, a, b = 0, j, col = 0, k, i, d, c, dig = 0;
	char linha[80];

	FILE* fp = fopen (arquivo, "r"), *aux;

	if (fp == NULL)
		printf ("Erro na abertura do arquivo!\n");

	for (a = 0; palavra_chave[a] != '\0'; a++)
		dig++;

	while (fgets (linha, 80, fp) != NULL)
	{
		n++;
		if (strstr (linha, palavra_chave) != NULL)
		{
			for (j = 0; &linha[j] != strstr (linha, palavra_chave); j++)
				col++;

			for (k = 0; k < col; k++)
				fgetc(fp);
```

```
                        if (fgets (linha, (dig+1), fp) != NULL)
                                n++;


                        for (i = 0; i < nb; i++)
                        {
                                sscanf (linha, "%d", &dados[i]);

                                while ((c = fgetc(fp)) != '\n')
                                {
                                        if (c == '\n')
                                                fgetc(fp);
                                }

                                for (k = 0; k < col; k++)
                                        fgetc(fp);

                                if (fgets (linha, (dig+1), fp) != NULL)
                                        n++;
                        }
                        break;
                }
        }
        fclose (fp);
        return dados;
}
```

Biblioteca para manipulação de matrizes - 'op_matrizes.h'

```
float** cria_matriz (int n)
{
  int i, j;
  float** m = (float**)malloc(sizeof(float*)*n);

  if (m == NULL)
  {
    printf ("Erro");

    return 0;
  }

  for(i = 0; i < n; i++)
  {
        m[i] = (float*)malloc(sizeof(float)*n);

        if (m[i] == NULL)
        {
                printf ("Erro");
                return 0;
        }

        for (j = 0; j < n ; j++)
                m[i][j] = 0.0;
  }

  return m;
}

float* cria_vetor (int n)
{
  int i;
  float* m = (float*)malloc(sizeof(float)*n);
```

```
  if (m == NULL)
  {
    printf ("Erro");
    return 0;
  }


  for(i = 0; i < n; i++)
  {
          for (i = 0; i < n ; i++)
                  m[i] = 0;
  }
  return m;
}

int* cria_vetor_int (int n)
{
  int i;
  int* m = (int*)malloc(sizeof(int)*n);

  if (m == NULL)
  {
    printf ("Erro");
    return 0;
  }

  for(i = 0; i < n; i++)
  {
          for (i = 0; i < n ; i++)
                  m[i] = 0;
  }
  return m;
}

void preenche_vetor (float* vetor, int col, float valor)
{
        vetor[col] = valor;
  return;
}

void imprime_matriz (float** matriz, int n)
{
  int i=0, j=0;

  for (i = 0; i < n ; i++)
  {
    for (j = 0; j < n ; j++)
        printf ("%9.4f ", matriz[i][j]);

    printf ("\n\n");
  }
  return;
}

void imprime_vetor (float* vetor, int n)
{
  int i = 0;

  for (i = 0; i < n ; i++)
      printf ("\n%9.4f\n", vetor[i]);
  return;
}

void imprime_vetor_int (int* vetor, int n)
{
  int i = 0;
```

```
  for (i = 0; i < n ; i++)
       printf ("\n%d\n", vetor[i]);
  return;
}

float* multiplica_mat_vet (float** matriz, float* vetor, int n)
{
        float *vet, *v;
        int i = 0, j = 0;

        v = (float*) malloc (n*sizeof(float));
        vet = (float*) malloc (n*sizeof(float));

        for (i = 0; i < n; i++)
        {
                v[i] = 0;
                vet[i] = 0;

                for (j = 0; j < n; j++)
                {
                        v[i] = matriz[i][j] * vetor[j];
                        vet[i] += v[i];
                }
        }
        return vet;
}

float** retira_ref_mat (float **matriz, int n, int ref)
{
        int i = 0, k = 0, j = 0, m, a = ref-1;
        float** nova_matriz = cria_matriz (n-1);

        for (i = 0; i < n; i++)
        {
                if (i != a)
                {
                        m = 0;
                        for (j = 0; j < n; j++)
                        {
                                if (j != a)
                                {
                                        nova_matriz[k][m] = matriz[i][j];
                                        if (m < (n-1))
                                                m++;
                                }
                        }
                        k++;
                }
        }
        return nova_matriz;
}

float* retira_ref_vet (float *vetor, int n, int ref)
{
        int i = 0, k = 0, a = ref-1;
        float* novo_vetor = cria_vetor (n-1);

        for (i = 0; i < n; i++)
        {
                if (i != a)
                {
                        novo_vetor[k] = vetor[i];
                        if (k < (n-1))
                                k++;
                }
        }
```

```
	return novo_vetor;
}

void matriz_negativa (float **matriz, int n)
{
	int i = 0, j = 0;
	for (i = 0; i < n; i++)
		for (j = 0; j < n; j++)
			if (matriz[i][j] != 0)
				matriz[i][j] = (-1) * matriz[i][j];
	return;
}

void vetor_negativo (float *vetor, int n)
{
	int i = 0;

	for (i = 0; i < n; i++)
		vetor[i] = (-1) * vetor[i];
	return;
}

float** inv_elem_imag_fora_diag (float **matriz, int n)
{
	int i = 0, j = 0;
	float** nova_matriz = cria_matriz (n);

	for (i = 0; i < n; i++)
		for (j = 0; j < n; j++)
		{
			if (matriz[i][j] != 0)
				nova_matriz[i][j] = 1.0000/matriz[i][j];
		}

	matriz_negativa (nova_matriz, n);
	return nova_matriz;
}

void calc_elem_diagonal (float **matriz, int n)
{
	int a, b, c;

	for (a = 0; a < n; a++)
		for (b = 0; b < n; b++)
			if (a == b)
				for (c = 0; c < n; c++)
				{
					if (c != b)
						matriz[a][b] -= matriz[a][c];
				}
	return;
}

float* soma_vetor (float *vetor1, float *vetor2, int n)
{
	int i = 0;
	float* soma = cria_vetor (n);

	for (i = 0; i < n; i++)
		soma[i] = vetor1[i] + vetor2[i];
	return soma;
}

int get_minor(float **src, float **dest, int row, int col, int ordem)
{
   int col_count = 0, row_count = 0;
```

```
    for(int i = 0; i < ordem; i++)
    {
        if( i != row )
        {
            col_count = 0;
            for(int j = 0; j < ordem; j++ )
            {
                if( j != col )
                {
                    dest[row_count][col_count] = src[i][j];
                    col_count++;
                }
            }
            row_count++;
        }
    }
    return 1;
}

float calc_determinante( float **mat, int ordem)
{
    if( ordem == 1 )
        return mat[0][0];

    float det = 0;

    float **minor;
    minor = new float*[ordem-1];
    for(int i=0;i<ordem-1;i++)
        minor[i] = new float[ordem-1];

    for(int i = 0; i < ordem; i++ )
    {
        get_minor( mat, minor, 0, i , ordem);
        det += pow( -1.0, i ) * mat[0][i] * calc_determinante( minor,ordem-1 );
    }

    for(int i=0;i<ordem-1;i++)
        delete [] minor[i];
    delete [] minor;

    return det;
}

void inverte_matriz (float **a, int ordem, float **y)
{
    double det = 1.0/calc_determinante(a,ordem);

    float *temp = new float[(ordem-1)*(ordem-1)];
    float **minor = new float*[ordem-1];
    for(int i = 0; i < ordem-1; i++)
        minor[i] = temp+(i*(ordem-1));

    for(int j = 0;j < ordem; j++)
    {
        for(int i = 0;i < ordem; i++)
        {
            get_minor(a,minor,j,i,ordem);
            y[i][j] = det*calc_determinante(minor,ordem-1);
            if( (i+j)%2 == 1)
                y[i][j] = -y[i][j];
        }
    }
    delete [] minor[0];
    delete [] minor;
```

```
}

float** calc_condutancia (float **matriz_r, float **matriz_x, int n)
{
	float **condut;
	int i = 0, j = 0;
	float mod, ang;

	condut = cria_matriz (n);

	for (i = 0; i < n ; i++)
	     for (j = 0; j < n ; j++)
	     {
		if ( j > i )
		{
			if ((matriz_r[i][j] != 0) || (matriz_x[i][j] != 0))
			{
				mod = 1.0 / hypot (matriz_r[i][j], matriz_x[i][j]);

				if (matriz_r[i][j] != 0)
					ang = (-1.0)*(atan2f (matriz_x[i][j], matriz_r[i][j]));
				else
					ang = -1.570796;

				condut[i][j] = mod * cos(ang);
			   condut[j][i] = mod * cos(ang);
			}

			else
			{
				condut[i][j] = 0;
				condut[j][i] = 0;
			}
		}
	      }
	return condut;
}

void per_unit_vet (float* vetor, float vbase, int n)
{
	int i;

	for (i = 0; i < n ; i++)
			vetor[i] = vetor[i] / vbase;
  return;
}

float paralelo (float num1, float num2)
{
	float i = (num1 * num2)/(num1 + num2);
  return i;
}
```