Escola Politécnica da USP

Departamento de Matemática Aplicada (IME-USP)

Departamento de Energia e Automação (POLI-USP)

MAP3121 – Métodos Numéricos e Aplicações

PEA3301 – Introdução aos Sistemas de Potência

Exercício Programa nº 1

Engenharia Elétrica

Isabela Comegna	nº	Turma 02
Ricardo Lemos	nº 8993729	

Prof. Antoine Laurain

Prof.

1. Introdução

Esse relatório tem a finalidade de apresentar os métodos utilizados e os resultados obtidos na solução do enunciado do Exercício Programa 1 de Métodos Numéricos e Aplicações e Introdução aos Sistemas de Potência.

O principal problema apresentado pelo exercício programa é o de fluxo de potência em redes elétricas. Tal problema está ligado ao desenvolvimento dos computadores digitais, a partir da década de 1960. Nessa época já existia a necessidade de análise das redes elétricas de grande porte, porém as ferramentas disponíveis não supriam a necessidade de apresentar os aspectos relevantes em um sistema de potência real.

A aplicação dos métodos de Newton à resolução do problema de fluxo de potência surgiu na década de 1970 e foi importantíssima, devido à robustez e rápida convergência apresentadas pelo método.

Neste trabalho, então, o método de Newton, aliado a decomposição LU para a solução de sistemas lineares, foram aplicados para a resolução dos problemas, por meio de métodos implementados na linguagem de programação C.

2. Objetivos

O exercício programa tinha como objetivo o de resolução do problema de fluxo de potência em redes elétricas. Desta maneira, deveriam ser desenvolvidos métodos, na linguagem de programação C, que determinam a corrente que circula em todos os trechos das redes apresentadas, para a verificação da existência de sobrecarga neles. Deveria ser determinada a tensão em todos os nós de carga e, também, as perdas totais nas redes especificadas.

Por meio de técnicas de resolução de circuitos elétricos, aliadas a métodos numéricos de resolução de sistemas lineares, então, deveriam ser determinadas as incógnitas solicitadas.

3. Solução de Sistemas Lineares

Para implantar a solução de sistemas lineares utilizamos o método de decomposição LU. No qual uma matriz qualquer A é decomposta em uma multiplicação de duas matrizes L e U.

$$A = L * U$$

Utilizando o método de Eliminação de Gauss com condensação pivotal. Adquirimos as matrizes L e U tal que L é uma matriz triangular unitária inferior e U uma matriz triangular superior. Por exemplo:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Dada a equação matricial apresentada a seguir:

$$Ax = b$$

Pode-se reescrevê-la como:

$$LUx = b$$

Assim, é possível resolver o sistema mais facilmente em dois passos sendo eles:

$Ly = b$, para y.

$Ux = y$, para x.

A vantagem deste método é que ele apresenta uma eficiência computacional maior, uma vez que o vetor b pode ser alterado e mesmo assim a eliminação de Gauss não precisa ser refeita para todas as alterações.

No exercício programa em questão, foi utilizada a decomposição PLU, em que a matriz A tem a seguinte forma:

$A = PLU$, sendo P a matriz de permutações.

A resolução do sistema é feita de forma similar, com a diferença de que se calcula:

$Ly = Pb$, para y.

$Ux = y$, para x.

Para a decomposição PLU, foi criado um método no exercício apresentado em anexo a este relatório, chamado decomposição_LU. Tal método recebe a matriz a ser decomposta (A), sua dimensão n (em que A é uma matriz *n x n*) e um vetor p, o qual será alterado pelo método para refletir o vetor de permutações (P) após a chamada desta função. O método utilizado para a implementação da fatoração foi o método apresentado no enunciado do exercício programa em questão.

Aliado a este método, foram criados mais dois métodos, um para obtenção da matriz L a partir da matriz LU, e um para a obtenção da matriz U também a partir da matriz LU. Estes métodos não teriam a necessidade de terem sido criados devido à forma que a solução dos sistemas lineares foi implementada no programa. Eles, no entanto, foram criados para que o programa tivesse maior clareza e mais fácil interpretação e depuração de possíveis erros. Estes métodos, no programa, chamam-se obter_matriz_L e obter_matriz_U, respectivamente. Ambos recebem uma matriz LU já fatorada e sua dimensão n (em que LU é *n x n*) e retornam a matriz a qual deseja-se receber.

Para a solução de um sistema linear qualquer foi criado um método chamado resolucao_sistema_linear. Este método recebe uma matriz A, sua dimensão n (em que A é *n x n*), e um vetor b. Tal que seja possível resolver a seguinte equação:

$$Ax = b$$

O método primeiramente obtém a matriz LU a partir da fatoração de A, chamando o método de decomposição LU. Após isso, ele chama os métodos descritos acima e obtém as matrizes L e U separadamente (para maior clareza do código).

São criados os vetores c, y e x, que serão utilizados para a resolução do problema. Sendo c o vetor b alterado de acordo com a matriz de permutações do método de fatoração LU.

O vetor y é obtido de:

$Ly = c$, para y.

E o vetor x, que é a solução do sistema linear em questão, é obtido de:

$Ux = y$, para x.

4. Método de Newton na Forma Matricial

A fórmula comum do Método de Newton é:

$$x_{i+1} = x_i - \frac{f_{(x_i)}}{f'_{(x_i)}}$$

Na forma matricial temos:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{i+1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_i - \begin{bmatrix} f_{1(x_i)} \\ f_{2(x_i)} \\ \vdots \\ f_{n(x_i)} \end{bmatrix}_i * \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_i) & \frac{\partial f_1}{\partial x_2}(x_i) & \cdots & \frac{\partial f_1}{\partial x_n}(x_i) \\ \frac{\partial f_2}{\partial x_1}(x_i) & \frac{\partial f_2}{\partial x_2}(x_i) & \cdots & \frac{\partial f_2}{\partial x_n}(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x_i) & \frac{\partial f_n}{\partial x_2}(x_i) & \cdots & \frac{\partial f_1}{\partial x_1}(x_i) \end{bmatrix}^{-1}$$

Simplificando temos:

$$[x]_{i+1} = [x]_i - \left[F_{(x_i)}\right] * [J_{(x_i)}]^{-1}$$

Passando $[x]_i$ para o outro lado:

$$[x]_{i+1} - [x]_i = -\left[F_{(x_i)}\right] * [J_{(x_i)}]^{-1}$$

Para não lidarmos com a inversa de uma matriz, multiplicamos os dois lados pela Jacobiana:

$$\left[J_{(x_i)}\right] * ([x]_{i+1} - [x]_i) = -\left[F_{(x_i)}\right]$$

Substituindo $[x]_{i+1} - [x]_i$ por $[c]_{i+1}$:

$$\left[J_{(x_i)}\right] * [c]_{i+1} = -\left[F_{(x_i)}\right]$$

Agora temos um sistema linear do tipo $Ax = b$. Após resolver o sistema temos nossa solução da primeira iteração. O método de Newton é um método iterativo, ou seja, devem ser realizadas várias iterações até atingir um resultado aceitável.

Definimos aceitável como aquele resultado cujo erro é menor que um valor previamente decidido ε. Podemos estimar o erro com a diferença entre os resultados de duas iterações. Dado que estamos lidando com matrizes, vamos obter o maior valor dessa diferença de matrizes. Sendo:

$$Erro = \text{máx}\ ([x]_{i+1} - [x]_i) = \text{máx}([c]_{i+1})$$

E então admitimos uma solução adequada quando essa diferença for menor que o valor estipulado previamente, ou seja:

$$Erro = \ máx\ [c]_{i+1} < \varepsilon$$

Em algoritmos é o que chamamos de critério de parada. Em outras palavras, as iterações serão repetidas até a solução estar correta e com um erro pequeno.

5. Teste 1

O Teste 1 do enunciado do Exercício Programa é:

- Use seu código para determinar o ponto de mínimo da função $F_{(x,y)} = (x-2)^2 + (y-3)^2$, calculando para tanto o ponto onde seu gradiente se anula. (Quantas iterações do método de Newton são necessárias para convergência?)

Chegamos que o gradiente $\nabla F_{(x,y)} = (2x-4, 2y-6)$. Com isso temos duas equações e duas variáveis. Temos o seguinte sistema linear do método de Newton:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{i+1} = \begin{bmatrix} x \\ y \end{bmatrix}_i - \begin{bmatrix} f_{1(x_i,y_i)} \\ f_{2(x_i,y_i)} \end{bmatrix} * \begin{bmatrix} \frac{\partial f_1}{\partial x}(x_i,y_i) & \frac{\partial f_1}{\partial x}(x_i,y_i) \\ \frac{\partial f_1}{\partial y}(x_i,y_i) & \frac{\partial f_1}{\partial y}(x_i,y_i) \end{bmatrix}^{-1}$$

Sendo:

$$x_0 = (1,5)$$

$$f_{1(x,y)} = 2x - 4$$

$$f_{2(x,y)} = 2y - 6$$

$$J = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Então temos:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{i+1} = \begin{bmatrix} x \\ y \end{bmatrix}_i - \begin{bmatrix} 2x-4 \\ 2y-6 \end{bmatrix}_i * \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1}$$

E tivemos os seguintes resultados:

ITERAÇÃO	X	Y	ERRO
0	1.0	5.0	-
1	2.0	3.0	2.0
2	2.0	3.0	0.0

Em uma iteração já encontramos a solução exata, mas apenas validamos quando tivemos o erro de 0.0 na segunda iteração.

6. Teste 2

- Dada a função $F(x_1, x_2, x_3, x_4) = (4x_1 - x_2 + x_3 - x_1x_4, -x_1 + 3x_2 - 2x_3 - x_2x_4, x_1 - 2x_2 + 3x_3 - x_3x_4, {x_1}^2 + {x_2}^2 + {x_3}^2 - 1)$, determine a raiz que se obtém pelo método de Newton tomando $x = (1,1,1,1)$ como valor inicial.

Vamos definir:

$$F(x_1, x_2, x_3, x_4) = \begin{bmatrix} f_1(x_1, x_2, x_3, x_4) \\ f_2(x_1, x_2, x_3, x_4) \\ f_3(x_1, x_2, x_3, x_4) \\ f_4(x_1, x_2, x_3, x_4) \end{bmatrix} = \begin{bmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \\ f_4(x) \end{bmatrix}$$

$$f_1(x) = 4x_1 - x_2 + x_3 - x_1x_4$$

$$f_2(x) = -x_1 + 3x_2 - 2x_3 - x_2x_4$$

$$f_3(x) = x_1 - 2x_2 + 3x_3 - x_3x_4$$

$$f_4(x) = {x_1}^2 + {x_2}^2 + {x_3}^2 - 1$$

$$J = \begin{bmatrix} 4 - x_4 & -1 & 1 & -x_1 \\ -1 & 3 - x_4 & -2 & -x_2 \\ 1 & -2 & 3 - x_4 & -x_3 \\ 2x_1 & 2x_2 & 2x_3 & 0 \end{bmatrix}$$

Assim temos a seguinte equação matricial do método de Newton:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_{i+1} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_i - \begin{bmatrix} f_{1(x)} \\ f_{2(x)} \\ f_{3(x)} \\ f_{4(x)} \end{bmatrix}_i * \begin{bmatrix} 4 - x_4 & -1 & 1 & -x_1 \\ -1 & 3 - x_4 & -2 & -x_2 \\ 1 & -2 & 3 - x_4 & -x_3 \\ 2x_1 & 2x_2 & 2x_3 & 0 \end{bmatrix}_i^{-1}$$

Obtemos os seguintes resultados:

x_1	0.000000
x_2	0.7071068
x_3	0.7071068
x_4	1.000000

- Iterações: 6
- Erro: 10^{-6}

7. Teste 3

- Utilize o método de Newton para determinar solução do sistema *n − 1 × n − 1*, cujas equações são

$$-x_{i-1} + 2x_i - x_{i+1} = \frac{e^{x_i}}{n^2}, i = 1, \dots, n-1,$$

Com $x_0 = x_n = 0$, a partir da aproximação inicial nula. (Teste para n = 20, 40 e 80)

As equações geram as seguintes funções:

$$f_i(x) = -x_{i-1} + 2x_i - x_{i+1} - \frac{e^{x_i}}{n^2}$$

E obtemos a Jacobiana:

$$\begin{vmatrix} 2-\frac{e^{x_1}}{n^2} & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2-\frac{e^{x_2}}{n^2} & -1 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2-\frac{e^{x_3}}{n^2} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2-\frac{e^{x_{n-3}}}{n^2} & -1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 2-\frac{e^{x_{n-2}}}{n^2} & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 2-\frac{e^{x_{n-1}}}{n^2} \end{vmatrix}$$

Então temos:

$$[x]_{i+1} = [x]_i - [F(x)]_i * [J]_i^{-1}$$

Com o método de Newton já implementado, resolvemos esse sistema.

8. Rede 1: Stevenson

Nos arquivos disponibilizados para o projeto foram disponibilizados dois arquivos de texto (no formato .txt), os quais correspondiam a duas tabelas das redes. Um dos arquivos é relacionado a uma tabela das barras, em que estão os tipos das barras (PQ, PV ou Swing), a tensão nominal de fase e outras informações especificas para cada barra. O outro arquivo é uma tabela de admitâncias nodais, com a condutânica Gjk e susceptância Bjk de um elemento.

Para o tratamento destes arquivos, criou-se uma função para criar uma matriz a partir de cada tabela fornecida, podendo-se, assim, acessar os valores fornecidos facilmente. Esta função foi chamada no arquivo do código de lerMatrizDadosBarras.

Com posse dos dados das barras da rede de Stevenson, é possível perceber que esta rede tem uma barra do tipo swing, três barras PQ e uma barra PV. Para facilidade de resolução do sistema para as variáveis pedidas, as barras foram reorganizadas de forma que primeiro estivessem as barras PQ, depois a barra PV e por último a barra swing. Essa permutação foi realizada pela função criarMatrizG (para permutar os valores de G) e pela função criarMatrizB (para permutar os valores de B).

Para que seja possível determinar os valores das fases e dos módulos das tensões de cada barra, é necessário realizar o cálculo de F(x) e o cálculo da Jacobiana do sistema apresentado na apostila do exercício programa.

Foi criada uma função calculo_F, que calcula, a partir dos valores de V e teta de uma determinada iteração, o valor da matriz F, solução do método de Newton.

Criou-se, também, uma função calculo_J, que a partir dos valores de V e teta de uma determinada iteração a matriz do jacobiano utilizado para a solução do método de Newton.

Há, ainda, uma função de metodo_de_newton_stevenson, que faz as iterações do método de Newton para o sistema criado a partir do Jacobiano e a matriz F calculadas pelas funções anteriores e calcula o resultado dentro de uma margem de erro especificada pelos programadores.