ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

“Fluxo de potência em corrente contínua e o método QR para resolução de sistemas lineares”

Turma 01

MAP3121 – Métodos Numéricos e Aplicações para Engenharia

Larissa Kimie Takayama - 8943365

Thomas Palmeira Ferraz - 9348985

SÃO PAULO, MAIO DE 2017

Sumário

1. Introdução

Para esse primeiro exercício computacional de Métodos Numéricos será analisado o problema de fluxo de potência em corrente contínua. O estudo de fluxo de potência em corrente contínua permite explorar a sensibilidade dos fluxos de potência a variações nos ângulos das tensões, ou seja, as variáveis deste problema serão apenas as potências ativas injetadas nas barras e os ângulos das tensões (não importando, portanto, o módulo dessas tensões).

A formulação do fluxo de potência em corrente contínua é dada por:

$$[P] = -[B][\theta]$$

Em que:

$$[P] = \begin{bmatrix} P_1 \\ P_2 \\ \dots \\ P_n \end{bmatrix} \quad , \quad [\theta] = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix} \quad , \quad [B] = \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \dots & \dots & \dots & \dots \\ B_{n1} & B_{n2} & \dots & B_{nn} \end{bmatrix}$$

Sendo n o número total de barras da rede, P o vetor de potência ativa injetada em cada barra, teta o vetor de ângulos das tensões em cada barra e B a matriz de susceptância do sistema.

Para que se chegasse a essa formulação foi necessário fazer algumas hipóteses simplificativas:

- Vj = 1 pu, j = 1, 2, ..., n (podemos aplicar isso pelo fato de os fluxos de potência ativa estarem mais ligados às diferenças angulares do que às diferenças entre módulos das tensões nas barras);
- Gjk<<Bjk, para todo par (i, j) (essa simplificação vale para sistemas de transmissão, em que G/B possui módulo inferior a 0,1);
- Senθkj é aproximadamente θkj (rad) (essa simplificação leva em conta que a diferença angular entre as tensões de duas barras adjacentes raramente ultrapassa 20°, e em termos de seno isso produz um erro relativo de apenas 2%).

Para esta atividade será feita a solução deste sistema linear através de um método chamado fatoração QR de matrizes. Este método basicamente vai transformar

uma matriz A nxm, com n>=m, em uma matriz triangular superior denominada R, que também será de ordem nxm.

Para que essa transformação funcione é preciso aplicar sucessivas transformações de Householder nessa matriz. Assim, basta definir um wi tal que a transformação Hwi leve o vetor coluna da matriz A (ai) em um múltiplo de ei (o i-ésimo vetor da base canônica do Rn. Logo:

$$w_i = a_i + \delta\frac{||a_i||}{||e_i||}e_i = a_i + \delta||a_i||e_i \quad , \quad \delta = sgn(A_{i,i})$$

Desta forma, para cada coluna da matriz A calcula-se um wi e aplica-se a transformação em todas as colunas de A. Ao final a matriz A torna-se uma matriz triangular superior (R).

Pode-se utilizar a fatoração QR de modo a resolver sistemas lineares, que é aquilo que vamos fazer neste exercício programa. Para tal aplica-se sucessivas transformações de Householder na matriz do sistema linear, ao mesmo tempo que aplica-se essas mesmas transformações no vetor b (considerando o sistema linear Ax=b). Ao final teremos uma um sistema linear equivalente Rx=~b. Como R é triangular superior basta então resolver o sistema de baixo para cima achando os valores do vetor x.

Também é possível utilizar a fatoração QR para resolver sistemas sobredeterminados, umas vez que minimizar o erro quadrático é o mesmo que minimizar:

$$||Q^t Ax - Q^t b|| = ||Rx - \bar{b}||$$

Assim, basta resolver o sistema com as primeiras m linhas de R e as primeiras m linhas de ~b, pois essa solução já minimiza o valor de $||Rx - \bar{b}||$ e, consequentemente, do erro quadrático, sendo essa solução a que melhor se aproxima de uma solução exata do problema sobredeterminado.

Depois de utilizar o método de fatoração QR e suas especificidades para resolver o sistema linear e encontrar o vetor solução de ângulos da formulação do fluxo de potência em corrente contínua, é possível também fazer o cálculo de fluxo de potência nas ligações.

Para isso, aplicando as mesmas hipóteses simplificativas citadas anteriormente, temos que esse valor de fluxo de potência será dada por:

$$P_{jk} = -b_{jk}\,(\theta_j - \theta_k)$$

Em resumo, neste exercício computacional será feito a resolução de um sistema linear pelo método de fatoração QR de forma a encontrar o vetor solução de ângulos de tensões. Depois com esses valores de ângulos e a matriz de susceptância (B) será encontrado o fluxo de potência das ligações da rede dada.

2. Resolução do problema

2.1. Justificativa do uso do C++ como linguagem

A escolha do uso do C++ se deve ao fato de ser uma linguagem muito mais simples em muitos aspectos utilizados neste trabalho. Destaca-se a maior facilidade com alocação dinâmica de espaços de memória e de operações de leitura e escrita de arquivos no formato .txt. O uso de programação defensiva também é facilitado com a existência das classes de exceção nesta linguagem.

Além disso, é uma linguagem que possui muitas bibliotecas prontas, com algoritmos e estruturas de dados já implementados. O principal motivo que levou o grupo a optar pela por ela foi a possibilidade de otimização de código que se vislumbrava. Como será descrito à seguir, as otimizações pensadas envolveram de alguma forma bibliotecas não disponíveis no antigo C.

2.2. Leitura do arquivo txt e alocação dinâmica

Para ler e escrever em texto foram usados os objetos da classe padrão **fstream** que facilitam a interface com os objetos. Ela permite definir maneiras de leitura e escrita. Por exemplo, é possível definir o número de casas ou a notação que se deseja ler, mas de maneira completamente intuitiva, usando classes e métodos da biblioteca **fstream**. Para leitura usa-se o objeto **ifstream** e para escrita o **ofstream**.

Para alocação dinâmica de memória foi utilizado o operador de alocação padrão do C++ **new**, que facilmente pode ser usado para construção de vetores ou matrizes.

2.3. Algoritmo de transformação de Householder

A fim de aplicar sucessivas transformações de Householder em uma matriz (A) e em um vetor (b), criou-se uma função chamada houseHolder. Para fazer as operações

necessárias dentro dessa função criou-se algumas outras funções básicas de manipulação de vetores. Assim, para iniciar o entendimento do algoritmo de Householder, primeiro é preciso entender algumas outras funções que seguem abaixo.

A primeira função criada foi a função que calcula norma de um vetor, ou seja, faz a soma dos quadrados de cada termo do vetor e depois tira raiz quadrática dessa soma.

Para evitar contas com termos nulos no programa, visando sua otimização, acrescentou-se mais uma entrada nessa função que faz com que as contas sejam feitas apenas a partir de um certo ponto do vetor. Isso diminui o número de operações que devem ser feitas durante a transformação de Householder, uma vez que calcula-se a norma apenas de vetores que tem alguns de seus primeiros termos zerados. Segue abaixo o algoritmo feito.

```
double norma(double* v, int m, int inicio)
{
   int i;
   double soma=0, norma=0;
   for (i=inicio; i<m; i++)
   {
      soma = soma + (v[i]*v[i]);
   }
   norma = sqrt(soma);
   return norma;
}
```

Outra função necessária foi a função que multiplica um vetor por um escalar, ou seja, pega termo a termo de um vetor e o multiplica por um escalar.

```
void multEscalar(double* v, int m, double escalar, double* resultado)
{
   int i;
   for (i=0; i<m; i++)
   {
```

```
        resultado[i] = v[i] * escalar;
    }
}
```

A função que calcula um produto escalar de dois vetores também é necessária para a aplicação de transformação de Householder. Basicamente ela multiplica cada termo de um vetor pelo termo de mesmo índice de outro vetor e soma todos esses produtos. Novamente nota-se que para esta função foi utilizado mais um input que é o ponto de início que a função começa a fazer contas, evitando multiplicações com zero.

```
double prodEscalar(double* v1, int m, double* v2, int inicio)
{
    int i;
    double produto=0;
    for(i=inicio; i<m; i++)
    {
        produto = produto + (v1[i] * v2[i]);
    }
    return produto;
}
```

A última função utilizada foi a que subtrai dois vetores, ou seja, subtrai um termo de um vetor pelo termo de mesmo índice de outro vetor. Aqui também fez-se uso de mais um input na função indicando o ponto de início para se fazer as contas, evitando subtrações de zeros.

```
void subVetor(double* v1, int m, double* v2, double* subtracao, int inicio)
{
    int i;
    for(i=inicio; i<m; i++)
    {
        subtracao[i]=v1[i]-v2[i];
```

```
    }
}
```

Com o uso dessas funções anteriormente descritas fez-se o algoritmo para transformações de Householder que já faz a transformação simultaneamente em uma matriz (A) e um vetor (b). Basicamente, para cada coluna de A ele calcula o delta e o vetor w dado por:

$$w_i = a_i + \delta \frac{\|a_i\|}{\|e_i\|} e_i = a_i + \delta \|a_i\| e_i \quad , \quad \delta = sgn(A_{i,i})$$

Mas como w é quase igual a i-ésima coluna de A mudado apenas de um termo (pois o canônico tem apenas um termo não nulo), faz-se uso da própria i-ésima coluna de A mudada apenas de um termo. Isso evita muitas operações, uma vez que evita muitas contas com termos nulos.

Depois calcula-se a transformação para cada coluna de A e também para b através da seguinte lógica:

$$H_w x = x - 2\frac{w \cdot x}{w \cdot w} w.$$

É nesta parte que faz-se uso das funções de norma, produto escalar, multiplicação de um vetor por escalar e subtração de vetores. Além disso, antes de fazer a divisão por (w . w) ele verifica se w não é um vetor nulo, ou seja, se ele não tem norma nula. Isso foi feito, pois em um caso testado o w se anulava e a resposta final dava resultados que não eram números justamente por causa da divisão por zero.

```
void houseHolder(double** A, double* b, int linhas, int colunas)
{
    int i, j, k;
    int delta=-1;
    double *w = new double [linhas];
    double *resParcial = new double [linhas];
    double resParcial2, resParcial3;
    for(i=0; i<colunas; i++)
    {
        if(A[i][i]>=0)
        {
```

```
        delta = 1;
    }

    for(k=0; k<i; k++)
    {
        w[k] = 0;
    }
    for (k=i; k < linhas; k++){
        w[k] = A[i][k];
    }

    resParcial3 = delta*norma(w, linhas, i);
    w[i] = w[i] + resParcial3;

    for(j=i; j<colunas; j++)
    {
        resParcial2 = 0;
        if (norma(w, linhas, i)!=0)
        {
            resParcial2 = 2*prodEscalar(w, linhas, A[j], i)/prodEscalar(w, linhas, w, i);
        }
        multEscalar(w, linhas, resParcial2, resParcial);
        subVetor(A[j], linhas, resParcial, A[j], i);
    }
    resParcial2 = 0;
    if (norma(w, linhas, i) != 0)
    {
        resParcial2 = 2*prodEscalar(w, linhas, b, i)/prodEscalar(w, linhas, w, i);
    }
    multEscalar(w, linhas, resParcial2, resParcial);
    subVetor(b, linhas, resParcial, b, i);
}
delete (w);
```

```
    delete (resParcial);
}
```

2.4. Algoritmo de solução de sistema linear

Depois de aplicado a transformação de Householder para a matriz e vetor fornecidos pelo problema, basta agora resolver um sistema linear em que a matriz é triangular superior. Para tal fez-se uma função chamada resolveSistema. Nela aplica-se a lógica de resolver o sistema de baixo para cima achando os últimos termos do vetor x (ou no nosso caso, o vetor teta) e ir aplicando o valor encontrado abaixo nas equações de cima.

```
void resolveSistema(mat A, vet b, double* x)
{
    x[A.ncolunas-1] = b.v[A.ncolunas-1]/A.A[A.ncolunas-1][A.ncolunas-1];
    for (int i = A.ncolunas-2; i >= 0; i--){
        x [i] = 0.0;
        for (int j = i+1; j < A.ncolunas; j++){
            x[i] -= A.A[j][i]*x[j];
        }
        x[i] = (x[i]+b.v[i])/A.A[i][i];
    }
}
```

Vale notar aqui também que essa mesma função irá resolver sistema sobredeterminado, já que no algoritmo se usa o número de colunas do sistema e não o número de linhas. Como explicado na Introdução, pelo método QR pode-se afirmar que a solução das primeiras m linhas do sistema, que torna R quadrada e triangular superior, é a solução de menor erro quadrático desse sistema sobredeterminado.

2.5. Algoritmo para Matrizes de Banda e outras otimizações

Além das otimizações colocadas no algoritmo houseHolder, foi pensado no problema das matrizes de banda. A ideia que tentamos implementar seria de tentar minimizar as semilarguras de banda de cada linha re-ordenando as linhas.

Primeiramente descobre-se qual deve ser a nova ordem das linhas, que são colocadas no vetor *posicoes*. O código que segue os passos adiante:

- São produzidos vetores *indices_inicio* e *indices_fim* que contém os valores dos índices do primeiro (início) e do último (fim) elementos não nulos de cada linha i;
- A partir deles cria-se um novo vetor que armazena pares. São armazenados os pares {*indices_inicio[i]*, *i*};
- Este vetor é ordenado pelo primeiro elemento do par, de modo a conter os indices de início de cada linha de ordem crescente;
- Para cada valor da primeira chave do vetor (ou seja, para cada valor de inicio(i) possível), recolhe-se todas as segundas chaves que são armazenadas em um novo vetor *inicios_atual* e que são todas as linhas que começam a partir deste índice.
- Enquanto o vetor *inicios_atual* não estiver vazio, executa-se os seguintes passos:
    - Busca-se qual dessas linha tem o maior fim (i);
    - Coloca-se essa linha no vetor *posicoes* e retira ela do vetor *inicios_atual.*

Após isso, cria-se uma nova matriz e vai copiando as linhas na ordem correta presente no vetor *posicoes*.

Adiante deve-se calcular as semilarguras de banda da esquerda e da direita para cada **coluna**. Agora deve-se calcular o índice dos primeiros e últimos elementos não nulos das colunas. A semilargura usada no código será a máxima entre as da esquerda e da direita.

O código fica otimizado na medida que é possível calcular a norma ou o produto escalar dos vetores usando apenas uma parte deles. Ao invés de se calcular o produto

de 0 a n (onde n é o tamanho da coluna i), calcula-se de **i - lb** a **i + lb**, evitando assim somar e fazer produtos com zero. A mesma otimização pode ser usada para eventuais somas e subtrações.

Uma outra otimização possível a complementar esta seria reordenar as colunas da mesma forma. Assim é possível reduzir ainda mais a semilargura de banda das colunas. No entanto, seria necessário reordenar as incógnas. Isso é possível fazer armazenando em um vetor para cada coluna sua origem. Ao final do algoritmo, bastaria reordenador os vetores de solução seguindo tal ordem.

Além disso é possível implementar tantas outras melhorias. Uma delas é o uso de thread do C++ que permite execução de funções em paralelo no software. Com ela você consegue chamar várias vezes tal função ao mesmo tempo acelerando o processamento. O ideal é usá-las em processos que não dependem entre si.

2.6. Algoritmo de fluxo de potência de ligações

Por fim para calcular o fluxo de potência de ligações fez-se uma função fluxoPot. Nela os inputs são a matriz de susceptância B e o vetor de teta, a partir disso ele calcula os fluxos de potência de ligações colocando-as em uma matriz denominada P. Assim, por exemplo, o elemento P[1][1] representa o fluxo de ligação P1 e o elemento P[1][2] representa o fluxo de potência P12.

```
void fluxoPot(double** B, int tamanho, double* teta, double** P)
{
   int i, j;
   double soma;
   for(i=0; i<tamanho; i++)
   {
      soma = 0;
      for(j=0; j<tamanho; j++)
      {
         if(i!=j)
         {
            P[i][j] = B[j][i]*(teta[i]-teta[j]);
```

```
            soma = soma + P[i][j];
        }
    }
    P[i][i] = soma;
  }
}
```

Esse algoritmo foi testado com o exemplo dado no enunciado do EP e gerou os resultados de acordo com o esperado.

2.7. Cálculo do Erro Quadrático Médio

Ainda foi possível implementar uma função que calcula o EQM de soluções de sistemas lineares sobredeterminados. Como visto no enunciado basta calcular a raiz da soma dos quadrados dos elementos do vetor b entre os índices m+1 e n, onde m é o número de colunas e n o número de linhas da matriz A.

3. Análise de resultados

3.1. Testes iniciais

Primeiro foram feitos alguns testes iniciais sugeridos no enunciado do exercício programa, sendo eles:

a. Para o caso: n = m = 64, Ai,i = 2, i = 1, n, Ai,j = −1, se|i − j| = 1 e Ai,j = 0, se|i − j| > 1. Usando b(i) = 1, i = 1, n.

Com esses dados pode-se encontrar x aplicando o algoritmo de Householder e de resolver sistema linear. A saída resultou em:

3.199999999997666e+001	*1.769999999998687e+002*	*2.969999999997772e+002*
6.299999999995346e+001	*2.029999999998496e+002*	*3.179999999997596e+002*
9.299999999993094e+001	*2.279999999998315e+002*	*3.379999999997419e+002*
1.219999999999096e+002	*2.519999999998130e+002*	*3.569999999997249e+002*
1.499999999998886e+002	*2.749999999997948e+002*	*3.749999999997067e+002*

3.9199999999996885e+002	*5.2799999999996601e+002*	*3.7499999999998545e+002*
4.0799999999996703e+002	*5.2699999999996726e+002*	*3.5699999999998602e+002*
4.2299999999996533e+002	*5.2499999999996874e+002*	*3.3799999999998676e+002*
4.3699999999996379e+002	*5.2199999999997010e+002*	*3.1799999999998744e+002*
4.4999999999996260e+002	*5.1799999999997135e+002*	*2.9699999999998829e+002*
4.6199999999996146e+002	*5.1299999999997260e+002*	*2.7499999999998909e+002*
4.7299999999996061e+002	*5.0699999999997380e+002*	*2.5199999999999002e+002*
4.8299999999995998e+002	*4.9999999999997516e+002*	*2.2799999999999108e+002*
4.9199999999995958e+002	*4.9199999999997669e+002*	*2.0299999999999216e+002*
4.9999999999995953e+002	*4.8299999999997800e+002*	*1.7699999999999326e+002*
5.0699999999995970e+002	*4.7299999999997931e+002*	*1.4999999999999437e+002*
5.1299999999996021e+002	*4.6199999999998050e+002*	*1.2199999999999548e+002*
5.1799999999996089e+002	*4.4999999999998153e+002*	*9.2999999999996589e+001*
5.2199999999996169e+002	*4.3699999999998249e+002*	*6.2999999999997719e+001*
5.2499999999996271e+002	*4.2299999999998334e+002*	*3.1999999999998856e+001*
5.2699999999996362e+002	*4.0799999999998420e+002*	
5.2799999999996476e+002	*3.9199999999998488e+002*	

b. Para o caso: n = 20, m = 17, Ai,j = 1/(i + j − 1), se|i − j| ≤ 4 e Ai,j = 0, se|i − j| > 4. Usando b(i) = 1, i = 1, n.

Com os dados fornecidos no enunciado foi possível montar a matriz A e o vetor b e aplicar Householder sucessivas vezes de modo a chegar no vetor solução x, que segue abaixo.

2.8815508535046472e+000	*3.4219279534192784e+000*	*-1.2038993662416275e+000*
-1.8337657572058907e+000	*3.6565617345778292e+000*	*1.2841813968786314e-001*
-1.5139885278205583e+000	*1.2036830022516796e+000*	*6.5015898717970186e+000*
-1.5219047149651244e+000	*6.1253408312630828e+000*	*1.1491052601308391e+001*
-4.5379229122874637e-001	*-2.4797174783544125e+000*	*1.4580525739921184e+001*
5.8566991990602828e+000	*-1.4779302618478478e+000*	

c. Para o arquivo “2_Completa_D” fornecido no site da disciplina.

Neste teste foi pego a matriz e o vetor, já em txt, e aplicou-se o algoritmo desenvolvido em grupo, resultando no vetor de ângulos abaixo:

1.3157894891632968e-001
-7.8947367389113551e-002

Nesse sistema, podemos ver que o maior ângulo obtido na resposta foi de 0.13157°, sendo que o fator de potência máximo então seria aproximadamente 1, ou seja, praticamente resistivo e este seria um sistema sem problema de fator de potência baixo.

3.2. Exemplo reduzido com 5 barras

Também foi testado o arquivo “4_Completa_D” fornecido no site da disciplina, que nada mais é que um sistema de 5 barras dispostas como na figura abaixo.

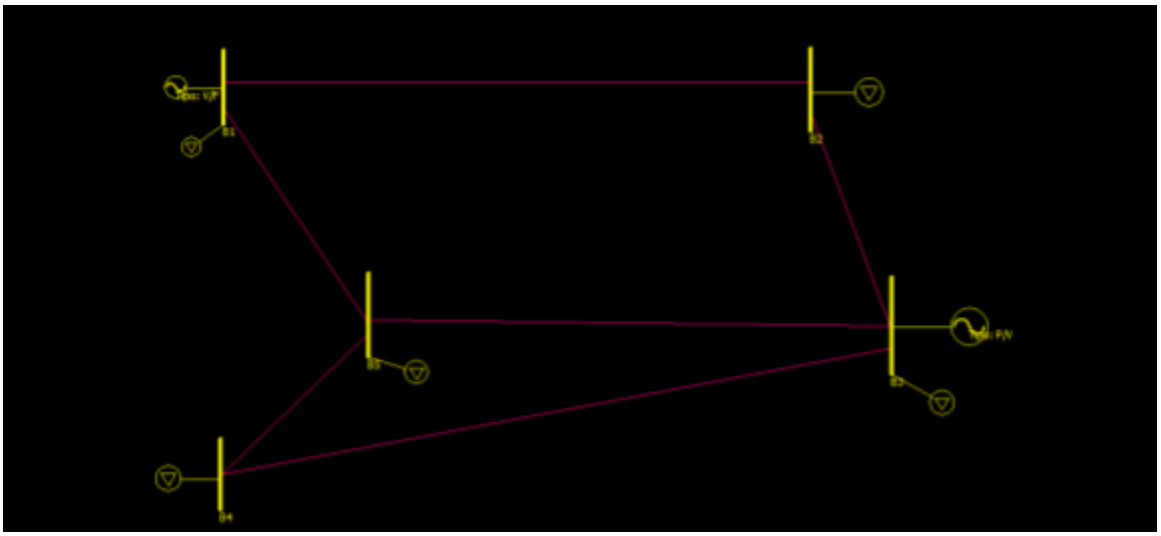

Figura 1 - Exemplo reduzido com 5 barras.

Com a matriz B e o vetor P fornecidos foi possível encontrar o vetor de ângulos do sistema. O resultado foi o seguinte:

-2.8129930002669085e-001

-3.9133259749361859e-001
-2.4374812100315352e-001
-3.3821189703212973e-001

Neste sistema temos que o maior ângulo (em módulo) foi de 0,391332°, o que nos dá um fator de potência de aproximadamente 1. Isso significa que esse sistema de transmissão não teria problema de fator de potência baixo.

3.3. Rede completa com 77 barras

A seguir foi testado o arquivo “76A_Completa_D” fornecido no site da disciplina, que é um exemplo de rede com 77 barras e 80 ligações contendo todos os segmentos de um sistema elétrico (geração, transmissão, distribuição primária e distribuição secundária).

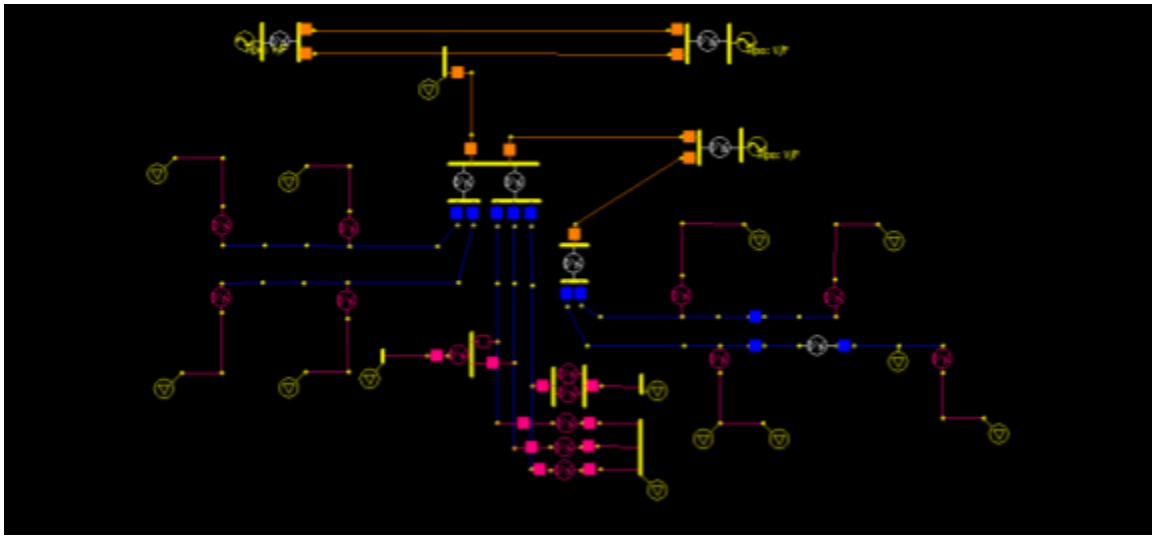

Figura 2 - Rede completa com 77 barras.

Com a matriz de susceptância (B) e o vetor de potência ativa (P) fornecidos foi possível encontrar o vetor de ângulos do sistema. O resultado foi o seguinte:

-1.0030034311554453e-003	*-7.1484686950992488e-003*	*-9.2494299266202302e-003*
-1.1395730624276521e-003	*-7.1484686950938252e-003*	*-1.4897130598580254e-003*
-1.2630362034473266e-003	*-1.4058293293313375e-003*	*-1.4951021127240164e-003*
-1.3864993444670014e-003	*-1.4219855980757898e-003*	*-1.5058675136273042e-003*
-7.0407602328075155e-003	*-1.4327564439049595e-003*	*-8.7555009212645968e-004*
-1.3896730605868846e-003	*-1.4435272897341301e-003*	*-1.5058675136277498e-003*
-7.0946144639627137e-003	*-1.4542981355633007e-003*	*-1.5004911655900079e-003*
-7.1484686951179188e-003	*-1.4004439064166827e-003*	*-1.5220219673965842e-003*
-7.1484686951172180e-003	*-1.4112147522464789e-003*	*-1.5220219673974741e-003*
-7.1484686951158206e-003	*-1.4166001751609964e-003*	*-1.5058802184560011e-003*
-7.1484686951151172e-003	*-1.4219855980755144e-003*	*-1.5274055753361625e-003*
-7.5956742219518798e-003	*-1.4273710209900326e-003*	*-1.5274055753370518e-003*
-8.1505882110962484e-003	*-9.3927676164095154e-004*	*-2.2810352901925623e-002*
-8.7055022002406231e-003	*-6.4132146405773806e-003*	*-4.4102266858492452e-002*
-9.7933576529998425e-003	*-6.4293709093210822e-003*	*-4.4002226859235972e-002*
-2.0095150551436406e-002	*-1.7008255863727836e-002*	*-1.9897397800844714e-002*
-1.8909581986398402e-002	*-2.7603297086873396e-002*	*-2.0093211799022643e-002*
-2.5097150439762399e-002	*-1.7024412132471824e-002*	*-1.6059876479404198e-003*
-4.0523466111211229e-002	*-2.7619453355617565e-002*	*-1.6274118594799757e-003*
-6.1713548557495772e-002	*-6.4239854864066000e-003*	*-1.6274118594808581e-003*
-3.5692191662906415e-002	*-1.7019026709556899e-002*	*-1.7221019659111997e-003*
-4.6287232886047219e-002	*-2.7614067932702376e-002*	*-1.6220619666549216e-003*
-5.7168469441779383e-002	*-1.1458297912225236e-002*	*-9.3927676164095154e-004*
-1.6311888562018440e-001	*-3.2648380358527920e-002*	*-1.2630362034473320e-003*
-2.6906930179856037e-001	*-5.3838462804822226e-002*	
-7.1484686951085157e-003	*-7.1484686951165163e-003*	

Novamente, nesse sistema o fator de potência máximo é praticamente 1, ou seja, não há problemas com fator de potência baixo.

3.4. Rede reticulada

Depois o próximo teste solicitado no enunciado era o do arquivo “76B_Completa_D” fornecido no site da disciplina. Esse exemplo é uma rede reticulada

com 77 barras e 117 ligações e foi inspirada no sistema reticulado usado pela Eletropaulo na região central de SP.

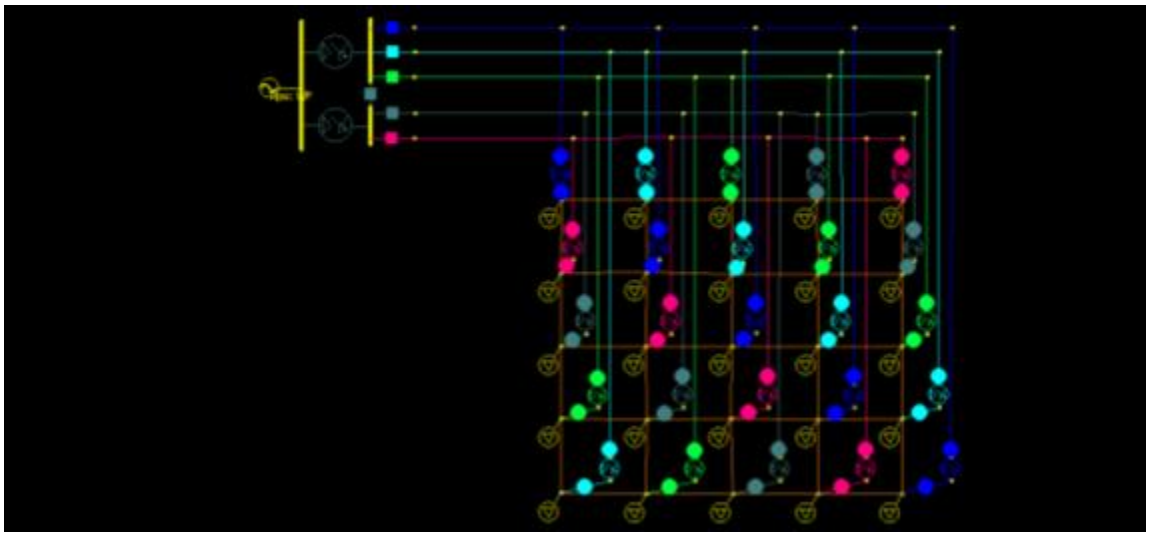

Figura 3 - Rede reticulada.

Com a matriz de susceptância (B) e o vetor de potência ativa (P) fornecidos foi possível encontrar o vetor de ângulos do sistema. O resultado foi o seguinte:

-6.1104283136887722e-003	*-1.8652870447163562e-002*	*-6.2402450983316321e-003*
-1.7402497633867125e-002	*-1.7431147395546957e-002*	*-6.2782003961948654e-003*
-1.8695507417682510e-002	*-5.9071676056560310e-003*	*-5.9630856487079822e-003*
-5.9071394147014577e-003	*-6.0505262244424131e-003*	*-6.1010329437265693e-003*
-6.0545718313734726e-003	*-6.1559211664592665e-003*	*-6.2168974774479014e-003*
-6.1640431765894290e-003	*-6.2274539915533004e-003*	*-1.8345238473424556e-002*
-6.2355577590790029e-003	*-6.2613171472781885e-003*	*-1.7220655422563465e-002*
-6.2732179484826121e-003	*-1.7089258513503531e-002*	*-6.2887222056676219e-003*
-5.9635264031553279e-003	*-6.2965334528607246e-003*	*-6.3165037031527541e-003*
-6.1068272667312478e-003	*-1.8478676054751550e-002*	*-1.8682395455526912e-002*
-6.2061394972191810e-003	*-1.7331501265093349e-002*	*-1.8754326025077137e-002*
-1.8442408824875429e-002	*-6.3344890123620563e-003*	*-1.7435756664504559e-002*
-6.2145425744542819e-003	*-1.8759818495270298e-002*	*-5.9071637143495751e-003*
-1.8572921614320791e-002	*-1.8797831986298918e-002*	*-6.0508130177449691e-003*
-6.2833190138228238e-003	*-5.9071379804640492e-003*	*-6.1605991285628142e-003*
-6.3115370186035673e-003	*-6.0545691793286284e-003*	*-6.2324233787677645e-003*
-1.7325408764410931e-002	*-6.1643346790291667e-003*	*-6.2662854460354637e-003*

-5.7258063658807033e-003	*-1.7400120005297125e-002*	*-6.1108690099200037e-003*
-5.9635106022202138e-003	*-1.8790664572317792e-002*	*-6.2203339651866552e-003*
-6.1007150531692672e-003	*-5.9071528471810455e-003*	*-1.7089331337606626e-002*
-1.8442162203945608e-002	*-6.0504974104211504e-003*	*-1.8576107853345646e-002*
-6.2158386579715449e-003	*-6.1599803206633316e-003*	*-6.2857646352038237e-003*
-1.7219831981791407e-002	*-6.2317980691919779e-003*	*-6.3290689117331709e-003*
-6.2820111591034081e-003	*-5.9574131092558907e-003*	*-1.8684157955397181e-002*
-1.8583909916966655e-002	*-6.2697572346911605e-003*	
-6.3260515866551291e-003	*-5.9574148644641464e-003*	

Assim como nos exemplos testados anteriormente, o maior ângulo resultante desse vetor de ângulos nos dá um fator de potência bem alto, praticamente 1. Esse sistema não tem problema com fator de potência baixo.

3.5. Rede real

O teste mais desafiador foi o de uma rede real, que é um circuito primário real de uma concessionária elétrica contendo 6260 barras. Esse arquivo “6259_Completa_D” também foi fornecido no site da disciplina.

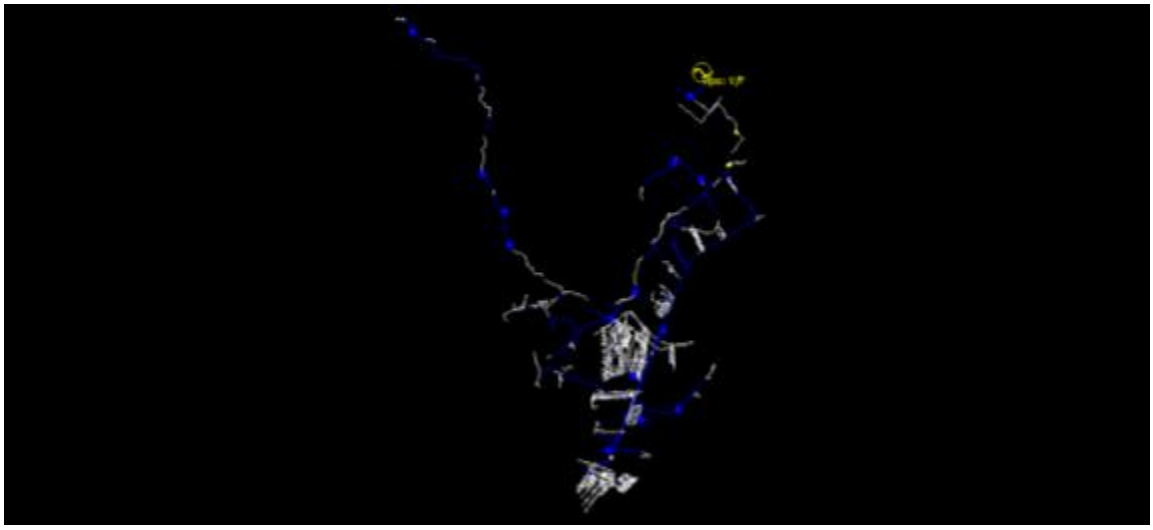

Figura 4 - Rede real.

Com os arquivos txt da matriz B e do vetor P, o grupo pôde encontrar o resultado do vetor de ângulos de tensões através do código desenvolvido. Por ser uma matriz relativamente grande o programa leva cerca de 50 minutos para processar todos os valores e dar as respostas dos elementos do vetor de ângulos. Porém conferindo com o resultado dado por Mathlab foi possível verificar que o resultado obtido estava correto.

Por motivos de tamanho o resultado gerado para esse caso não foi colocado neste relatório.

3.6. Outros

Além de testes com sistemas lineares simples que o grupo acabou por fazer durante o desenvolvimento do código, um teste interessante que foi feito e que seria interessante registrar aqui é de um caso de sistema sobredeterminado.

Escolheu-se resolver um sistema em que:

$$[B] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}, \quad [P] = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad [\theta] = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

Resolvendo esse sistema sem uso do programa sabemos que a solução dele deverá ser $[0 \ \frac{1}{3}]^t$ e que o erro quadrático é de aproximadamente 0,81.

Agora fazendo uso do programa desenvolvido pelo grupo o resultado dado foi:

2.6168207644729590e-017
3.3333333333333337e-001

O que é compatível com o resultado esperado.

4. Conclusão

Ao final dessa atividade pode-se desenvolver um programa em C++ que, dado uma matriz de susceptância e um vetor de potência ativa, resolve o sistema linear através do método de fatoração QR, resultando em um vetor de ângulos. O programa também calcula o fluxo de potência de ligações através do vetor de ângulos e da matriz de susceptância.

Com esses dados foi possível analisar, por exemplo, o fator de potência do sistema, que é um parâmetro importante para a concessionária.