Universidade de São Paulo
SME0205 Turma 01

Trabalho 1

12676615 João Pedro Pereira Balieiro

18/05/2023

Sumário

1 INTRODUÇÃO e OBJETIVOS

Nesse trabalho foram fornecidos os arquivos manh.el e manh.xy, que fornecem as arestas e as coordenadas dos vértices do grafo de ruas da ilha de Manhattan, NY. A partir desses arquivos foi construída uma matriz Laplaciana L do grafo de ruas, uma matriz de penalidades P e um vetor b, e foi possível chegar no sistema (L + P)x = Pb. O objetivo do trabalho é resolver o sistema usando quatro métodos diferentes e comparar o tempo de solução: Decomposição LU, Jacobi, Gaus-Seidel e Gradientes Conjugados.

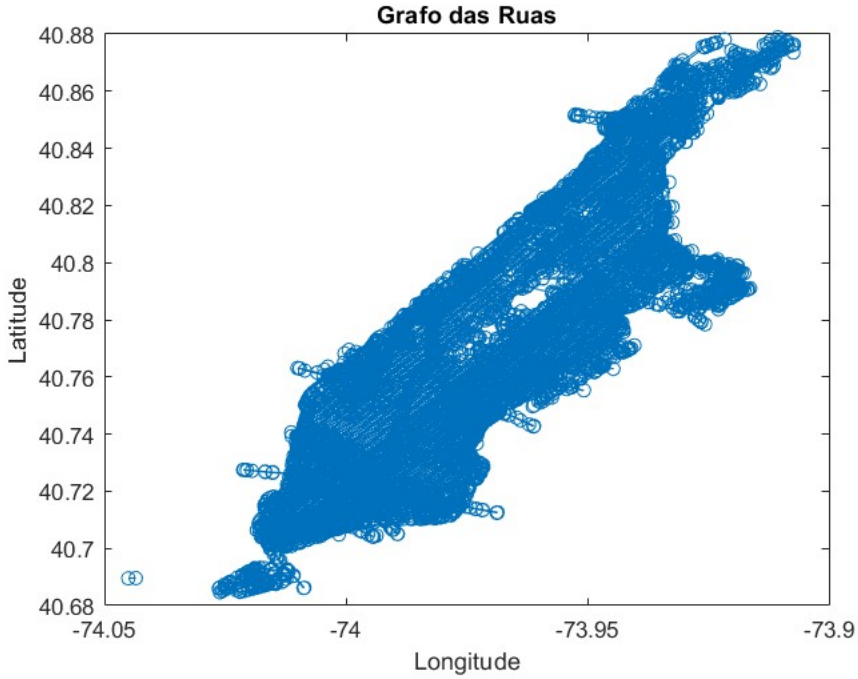

Figura 1: Grafo das Ruas

2 Códigos e métodos

2.1 Grafo das Ruas

Nessa primeira parte foi desenvolvido o código para receber os arquivos manh.el e manh.xy e plotar um grafo a partir da matriz de adjacencia.

21/05/23 14:06 C:\Users\fonso\OneDriv...\trabalho_1.m 1 of 1

```
% Item 0: Código para receber os arquivos manh.el e manh.xy e plotar o grafo de ↙
ruas da ilha de Manhattan, NY.
edgeFile = 'manh.el';
edges = dlmread(edgeFile);
coordFile = 'manh.xy';
coords = dlmread(coordFile);
numVertices = max(max(edges));
adjMatrix = zeros(numVertices+1, numVertices+1);
for i = 1:size(edges, 1)
 vertex1 = edges(i, 1) + 1;
 vertex2 = edges(i, 2) + 1;
 adjMatrix(vertex1, vertex2) = 1;
 adjMatrix(vertex2, vertex1) = 1;
end
figure;
gplot(adjMatrix, coords, '-o');
title('Grafo das Ruas');
xlabel('Longitude');
ylabel('Latitude');
```

Figura 2: Código do Grafo das Ruas

Nesse trecho de código, os arquivos 'manh.el' e 'manh.xy' são lidos para obter informações sobre as arestas e coordenadas dos vértices de um grafo. Em seguida, uma matriz de adjacência é construída com base nas informações das arestas, indicando as conexões entre os vértices. Por fim, o grafo é plotado utilizando as coordenadas dos vértices.

2.2 Selecionar vértices e atribuir valores

Nessa parte foi desenvolvido o código para selecionar alguns vértices, V_{i1}, V_{i2}, ...,V_{ik}, k « n (n é o número de vértices na maior componente do grafo e k é um número bem menor que n, k = 800) do grafo e atribuir valores a cada vértice selecionado c_{i1}, c_{i2}, ...,c_{ik}.

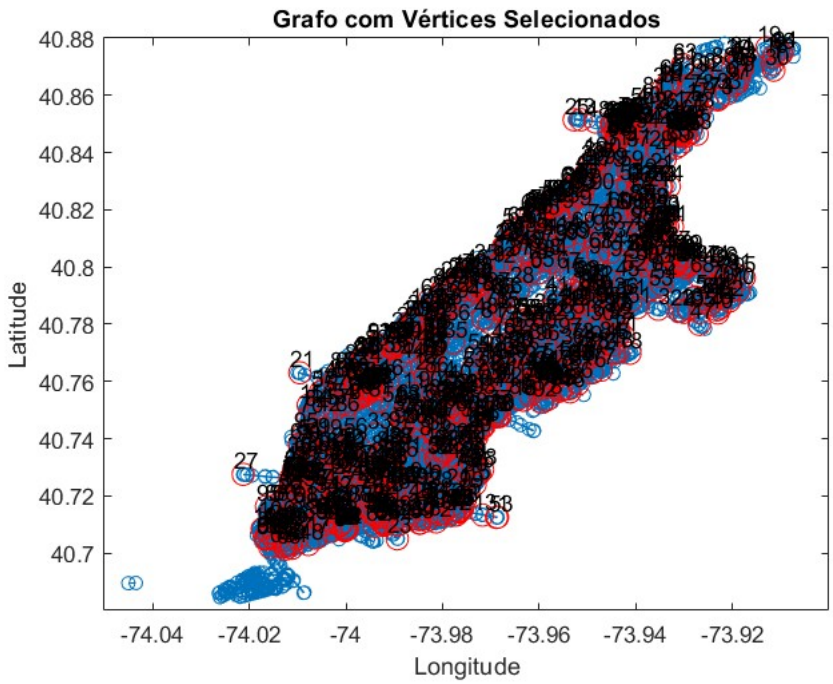

Figura 3: Grafo com vértices selecionados

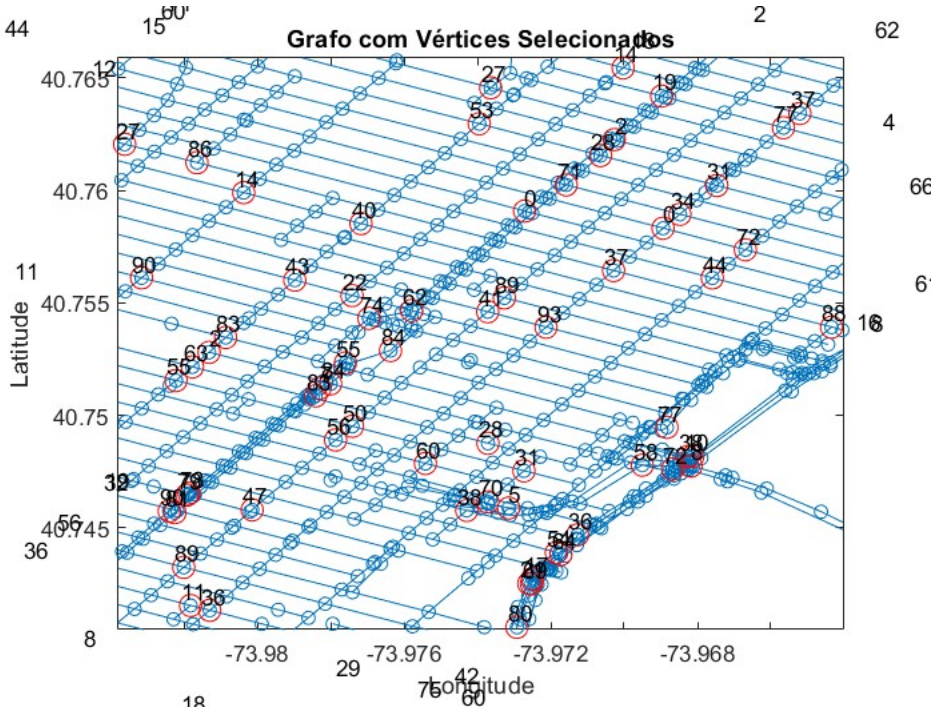

Figura 4: Grafo com vértices selecionados - zoom

21/05/23 15:44 C:\Users\fonso\OneDriv...\trabalho_1.m 1 of 1

```
% Item 1: Código para selecionar os vértices aleatoriamente e atribuir valores ↙
neles
graph = digraph(adjMatrix);
connectedComponents = conncomp(graph);
largestComponentIndex = mode(connectedComponents);
largestComponentIndices = find(connectedComponents == largestComponentIndex);
k = 800;
selectedVertices = randsample(largestComponentIndices, k);
selectedValues = randi([0, 800], k, 1);
figure;
gplot(adjMatrix, coords, '-o');
hold on;
plot(coords(selectedVertices, 1), coords(selectedVertices, 2), 'ro', 'MarkerSize',↙
10);
text(coords(selectedVertices, 1), coords(selectedVertices, 2), num2str ↙
(selectedValues), 'HorizontalAlignment', 'center', 'VerticalAlignment', 'bottom');
hold off;
title('Grafo com Vértices Selecionados');
xlabel('Longitude');
ylabel('Latitude');
```

Figura 5: Código para selecionar vértices e atribuir valores

Nessa parte do código, um conjunto de vértices é selecionado aleatoriamente a partir do grafo. Primeiro, o grafo é convertido em um objeto do tipo digraph (grafo direcionado) para realizar algumas operações. Em seguida, é identificada a maior componente conexa do grafo, ou seja, o subgrafo que contém o maior número de vértices conectados entre si. Foram selecionados 800 vértices. Os vértices são selecionados aleatoriamente utilizando a função randsample a partir dos índices da maior componente conexa. Além disso, valores numéricos aleatórios entre 0 e 800 são atribuídos a esses vértices selecionados. Em seguida, o grafo inicial é plotado, sendo os vértices selecionados destacados como pontos vermelhos com rótulos numéricos correspondentes aos valores atribuídos. O título e os rótulos dos eixos são definidos para o gráfico resultante.

2.3 Matriz Laplaciana

Nessa parte foi desenvolvido o código para construir a matriz Laplaciana L do grafo das ruas.

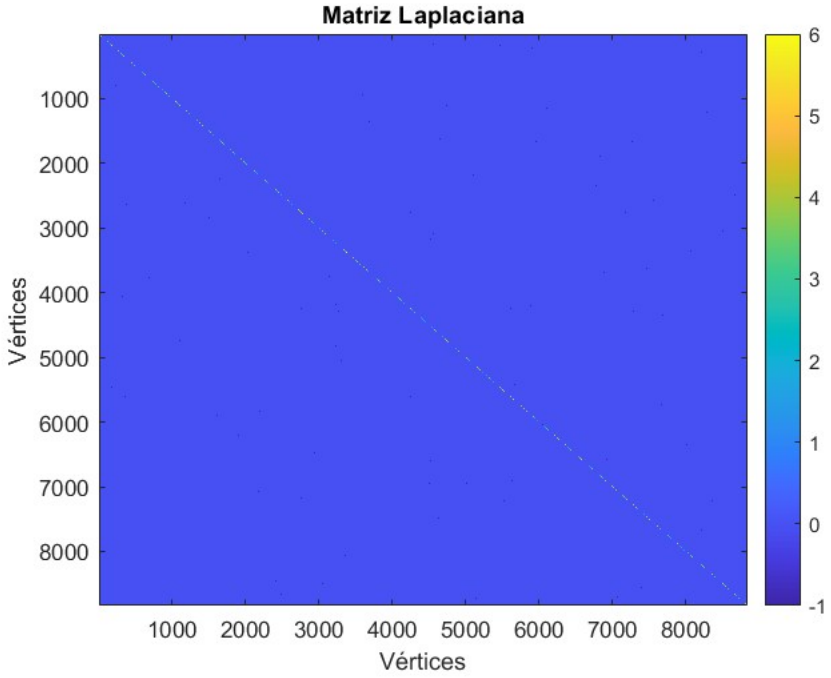

Figura 6: Matriz Laplaciana

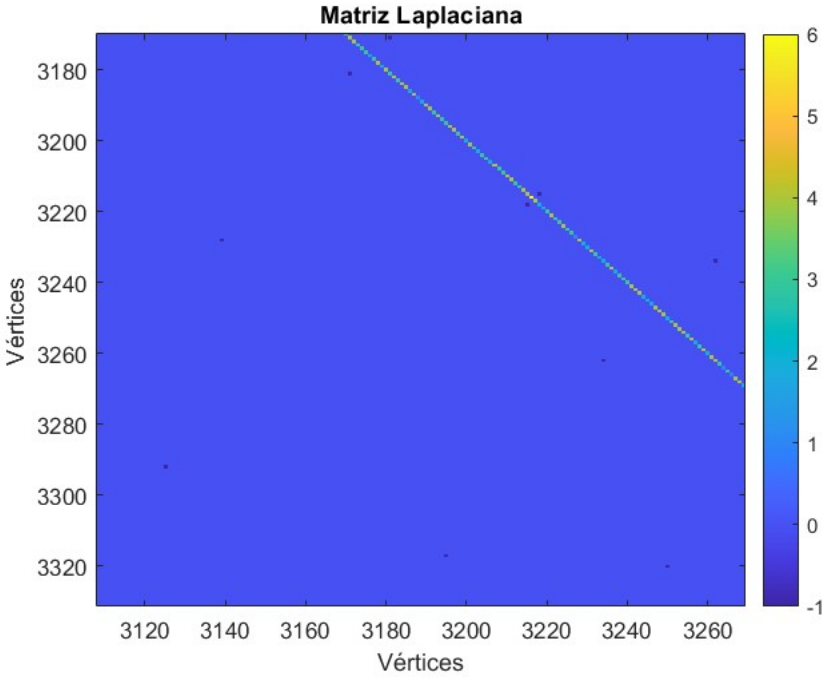

Figura 7: Matriz Laplaciana - zoom

21/05/23 14:06 C:\Users\fonso\OneDriv...\trabalho_1.m 1 of 1

```
% Item 2: Código para construir a matriz Laplaciana L do grafo das ruas
degreeMatrix = diag(sum(adjMatrix, 2));
L = degreeMatrix - adjMatrix;
figure;
imagesc(L);
colorbar;
title('Matriz Laplaciana');
xlabel('Vértices');
ylabel('Vértices');
```

Figura 8: Código para construir a matriz Laplaciana

Nessa parte do código, a matriz Laplaciana do grafo é construída. Inicialmente, é calculada a matriz de graus, onde cada elemento da diagonal principal representa a soma dos pesos das arestas incidentes em um vértice específico. Isso é feito somando-se os elementos de cada linha da matriz de adjacência adjMatrix (que representa as conexões entre os vértices) usando a função sum com o argumento 2, indicando a soma ao longo das linhas. Em seguida, a matriz Laplaciana L é obtida subtraindo a matriz de adjacência adjMatrix da matriz de graus degreeMatrix. Essa operação é realizada elemento por elemento. Após construir a matriz Laplaciana, um gráfico visual da matriz é gerado utilizando a função imagesc. A escala de cores é exibida por meio da função colorbar.

2.4 Matriz de Penalidades

Nessa parte foi desenvolvido o código para construir a matriz de penalidades P, sendo P uma matriz diagonal onde a entrada $P_{jj} = \alpha$ se j corresponde ao índice de algum dos vértices escolhidos e $P_{ii} = \alpha$ caso contrário.

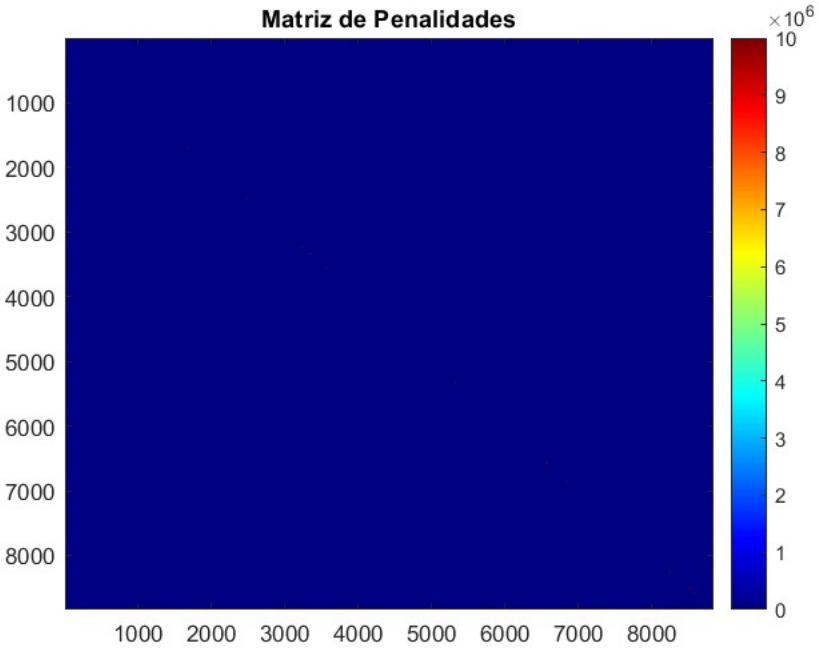

Figura 9: Matriz de penalidades

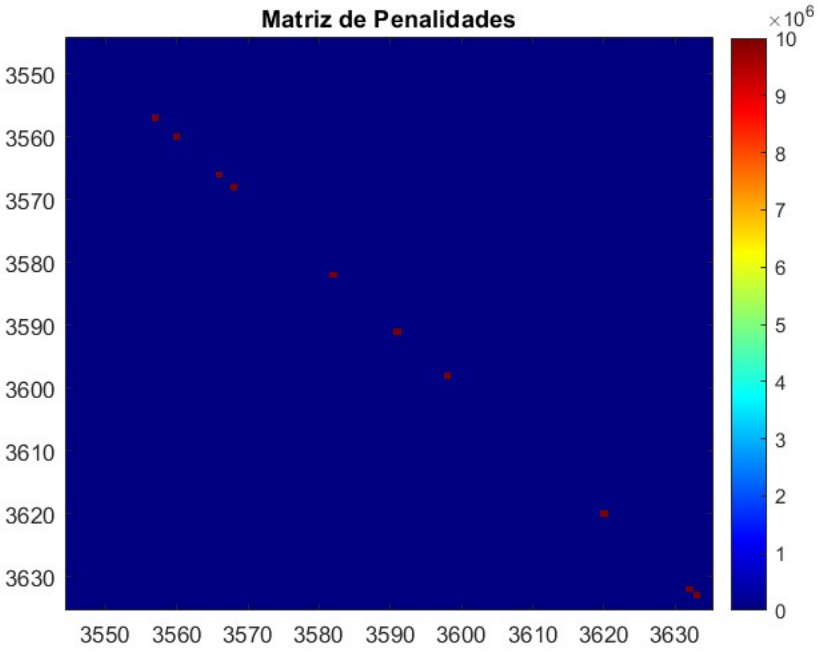

Figura 10: Matriz de penalidades - zoom

21/05/23 14:06 C:\Users\fonso\OneDriv...\trabalho_1.m 1 of 1

```
% Item 3: Código para construir a matriz de penalidades P
alpha = 1.0e7;
P = zeros(numVertices+1, numVertices+1);
P(selectedVertices, selectedVertices) = alpha * eye(k);
figure;
imagesc(P);
colormap('jet');
colorbar;
title('Matriz de Penalidades');
```

Figura 11: Código para construir a matriz de penalidades

Nessa parte do código, a matriz de penalidades P é construída. Primeiro, um valor numérico alpha é definido como 1.0e7 (10 milhões). Em seguida, uma matriz quadrada de zeros P é inicializada com dimensões (numVertices+1) x (numVertices+1), onde numVertices é o número total de vértices no grafo. Em seguida, os elementos da matriz P correspondentes aos vértices selecionados são atualizados. Isso é feito atribuindo-se o valor alpha multiplicado pela matriz identidade de tamanho k (onde k é o número de vértices selecionados) aos elementos de P nas posições indicadas pelos índices dos vértices selecionados (selectedVertices). A matriz identidade é uma matriz quadrada com uns na diagonal principal e zeros em todas as outras posições.

2.5 Vetor b

Nessa parte foi desenvolvido o código para construir o vetor b da seguinte forma: se $b_j = c_{is}$ se j = i_s e 0 caso contrário.

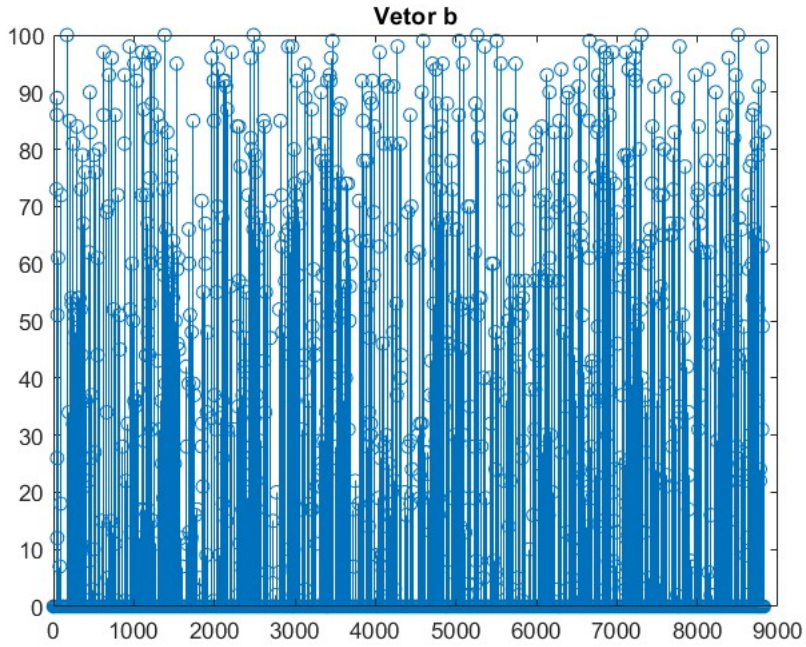

Figura 12: Vetor b

```
% Item 4: Código para construir o vetor b
b = zeros(numVertices+1, 1);
for i = 1:k
 vertexIndex = selectedVertices(i);
 b(vertexIndex) = selectedValues(i);
end
figure;
stem(b);
title('Vetor b');
```

Figura 13: Código para construir o vetor b

Nessa parte do código, o vetor b é construído. Primeiro, um vetor coluna de zeros b é inicializado com tamanho (numVertices+1) x 1, onde numVertices é o número total de vértices no grafo. Em seguida, é realizado um loop sobre os vértices selecionados, representados pelos índices i variando de 1 a k. Para cada vértice selecionado, o valor correspondente é atribuído ao elemento de b na posição indicada pelo índice do vértice (vertexIndex). O valor atribuído é obtido a partir do vetor selectedValues na posição i.

2.6 Decomposição LU

Nesse parte foi utilizado o método de Decomposição LU para resolver o sistema.

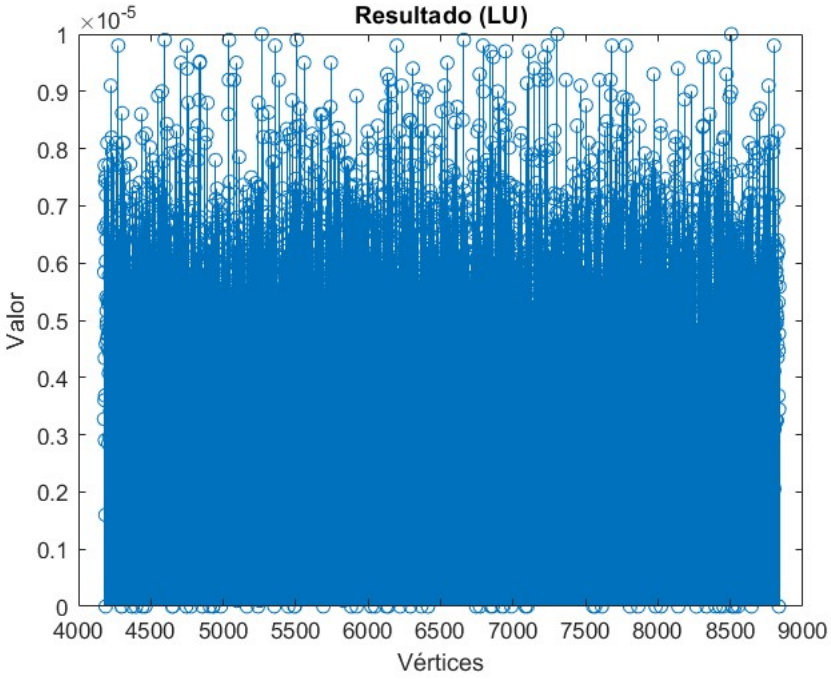

Figura 14: Grafo resultado - Decomposição LU

21/05/23 14:05 C:\Users\fonso\OneDriv...\trabalho_1.m 1 of 1

```
% Método 1: Decomposição LU
tic;
A_lu = L + P;
x_lu = A_lu\b;
time_lu = toc;
```

Figura 15: Decomposição LU

Nessa parte do código, o método de decomposição LU é utilizado para resolver o sistema de equações representado pela expressão (matriz Laplaciana + matriz de penalidades) * x = matriz de penalidades * vetor b. Primeiramente, a matriz A_{lu} é construída somando a matriz Laplaciana L com a matriz de penalidades P. Em seguida, a função \ é utilizada para calcular a solução x_{lu} do sistema, dado o vetor b. O tempo de execução do método é medido utilizando as funções tic e toc, e o resultado é armazenado na variável $time_{lu}$.

2.7 Jacobi

Nesse parte foi utilizado o método de Jacobi para resolver o sistema.

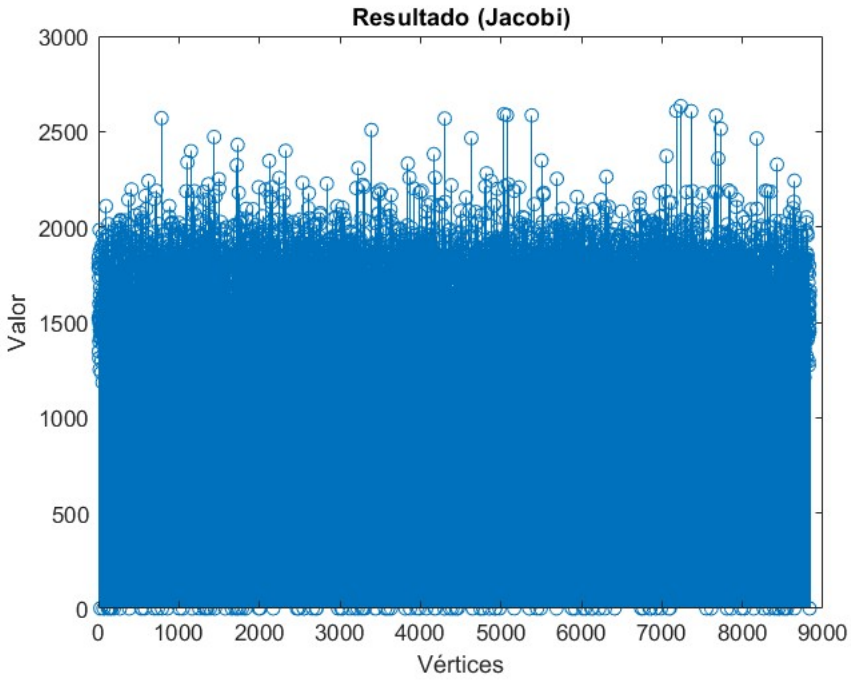

Figura 16: Grafo resultado - Jacobi

```
% Método 2: Jacobi
tic;
D = diag(diag(L));
R = L - D;
M_jacobi = -D\R;
N_jacobi = D\b;
x_jacobi = zeros(numVertices+1, 1);
maxIterations = 1000;
tolerance = 1e-6;
for iter = 1:maxIterations
   x_jacobi_new = M_jacobi * x_jacobi + N_jacobi;
   if norm(x_jacobi_new - x_jacobi) < tolerance
       break;
   end
   x_jacobi = x_jacobi_new;
end
time_jacobi = toc;
```

Figura 17: Jacobi

Nessa parte do código, o método de Jacobi é utilizado para resolver o sistema de equações representado pela expressão (matriz Laplaciana + matriz de penalidades) * x = matriz de penalidades * vetor b. O processo inicia com a construção da matriz diagonal D contendo os elementos diagonais da matriz Laplaciana L. A matriz R é obtida subtraindo a matriz diagonal D de L. Em seguida, é calculada a matriz M_{jacobi} como o negativo da matriz D inversa multiplicada por R. O vetor N_{jacobi} é calculado como a matriz D inversa multiplicada pelo vetor b. Um loop é executado com um máximo de iterações definido por maxIterations. A cada iteração, é calculada uma nova estimativa $x_{jacobinew}$ usando a fórmula do método de Jacobi. A condição norm($x_{jacobinew}$ - x_{jacobi}) < tolerance é verificada para determinar se a solução convergiu. Caso a condição seja satisfeita, o loop é interrompido.

2.8 Gauss-Seidel

Nesse parte foi utilizado o método de Gauss-Seidel para resolver o sistema.

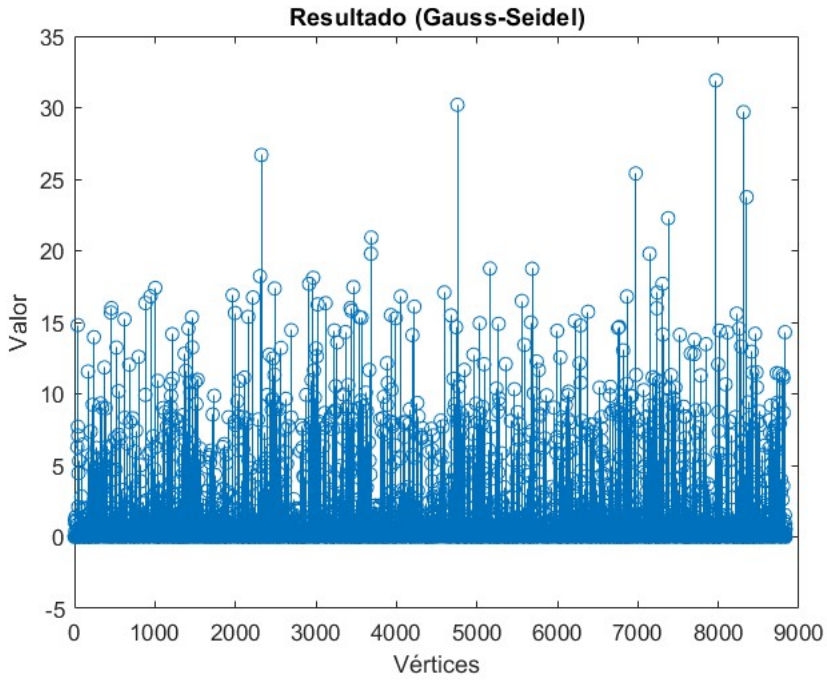

Figura 18: Grafo resultado - Gaus-Seidel

```
% Método 3: Gaus-Seidel
tic;
D = diag(diag(L));
L_gs = tril(L);
U_gs = triu(L);
M_gs = -(D + L_gs)\U_gs;
N_gs = (D + L_gs)\b;
x_gs = zeros(numVertices+1, 1);
for iter = 1:maxIterations
   x_gs_new = M_gs * x_gs + N_gs;
   if norm(x_gs_new - x_gs) < tolerance
       break;
   end
   x_gs = x_gs_new;
end
time_gs = toc;
```

Figura 19: Gauss-Seidel

Nessa parte do código, o método de Gauss-Seidel é utilizado para resolver o sistema de equações representado pela expressão (matriz Laplaciana + matriz de penalidades) * x = matriz de penalidades * vetor b. O processo inicia com a construção da matriz diagonal D contendo os elementos diagonais da matriz Laplaciana L. As matrizes L_{gs} e U_{gs} são obtidas a partir da matriz L, em que L_{gs} é a matriz triangular inferior de L e U_{gs} é a matriz triangular superior de L. Em seguida, é calculada a matriz M_{gs} como o negativo da inversa da soma de D com L_{gs}, multiplicada por U_{gs}. O vetor N_{gs} é calculado como a inversa da soma de D com L_{gs}, multiplicada por b. Um loop é executado com um máximo de iterações definido por maxIterations. A cada iteração, é calculada uma nova estimativa x_{gsnew} usando a fórmula do método de Gauss-Seidel. A condição norm(x_{gsnew} - x_{gs}) < tolerance é verificada para determinar se a solução convergiu. Caso a condição seja satisfeita, o loop é interrompido.

2.9 Gradientes Conjugados

Nesse parte foi utilizado o método de Gradientes Conjugados para resolver o sistema.

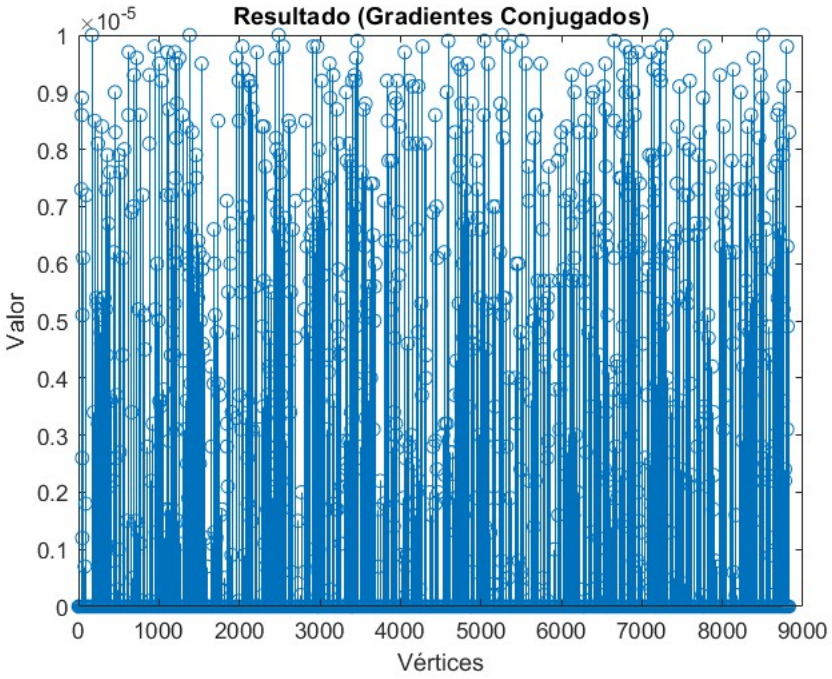

Figura 20: Grafo resultado - Gradientes Conjugados

21/05/23 14:05 C:\Users\fonso\OneDriv...\trabalho_1.m 1 of 1

```
% Método 4: Gradientes Conjugados
tic;
A_gc = L + P;
x_gc = pcg(A_gc, b, tolerance, maxIterations);
time_gc = toc;
```

Figura 21: Gradientes Conjugados

Nessa parte do código, o método dos Gradientes Conjugados é utilizado para resolver o sistema de equações representado pela expressão (matriz Laplaciana + matriz de penalidades) * x = matriz de penalidades * vetor b. A matriz A_gc é construída somando a matriz Laplaciana L com a matriz de penalidades P. Em seguida, a função pcg é chamada para realizar a resolução do sistema utilizando o método dos Gradientes Conjugados. Os argumentos passados para a função são a matriz A_gc, o vetor b, a tolerância tolerance e o número máximo de iterações maxIterations. O resultado é armazenado na variável x_gc, que representa a solução do sistema.

3 RESULTADO E CONCLUSÕES

Método	Tempo de Solução(segundos)
Decomposição LU	13,302
Jacobi	52,538
Gauss-Seidel	17,993
Gradientes Conjugados	0,47599

Tabela 1: Tempo de solução para cada método

Com base nos resultados obtidos, chegamos nas seguintes conclusões:

Decomposição LU: O método de decomposição LU apresentou um tempo de execução de 13,302 segundos. Ele é um método direto que envolve a decomposição da matriz em fatores LU, o que permite resolver o sistema de equações de forma mais eficiente. No entanto, esse método pode exigir maior tempo de execução em comparação com os métodos iterativos, especialmente para matrizes grandes.

Jacobi: O método de Jacobi apresentou um tempo de execução de 52,538 segundos. Esse método é iterativo e envolve a iteração entre os elementos da solução, atualizando-os de acordo com uma fórmula específica. No entanto, o tempo de execução mais longo pode ser devido à convergência mais lenta do método de Jacobi, especialmente para sistemas de equações mal-condicionados.

Gauss-Seidel: O método de Gauss-Seidel obteve um tempo de execução de 17,993 segundos. Similar ao método de Jacobi, o Gauss-Seidel também é um método iterativo, mas ele atualiza os elementos da solução de forma mais direta do que o método de Jacobi. Isso geralmente resulta em uma convergência mais rápida em comparação com o método de Jacobi.

Gradientes Conjugados: O método dos Gradientes Conjugados demonstrou um tempo de execução de 0,47599 segundos, o que é significativamente menor em comparação com os outros métodos. O método dos Gradientes Conjugados é um método iterativo projetado para sistemas de equações lineares simétricas e positivas-definidas. Ele utiliza informações do resíduo e dos gradientes para convergir rapidamente para uma solução precisa.