Engenharia de Sistemas e Computação
PESC/Coppe

COPPE UFRJ
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

UFRJ

# Relatório atividade 4

Aluno: Lyncoln Sousa de Oliveira

# Objetivos

- Buscar uma matriz esparsa simétrica positiva definida
- Aplicar os métodos Steepest Descent e Conjugate Gradients com e sem pré condicionamento pela diagonal
- Comparar tempo de processamento e iterações

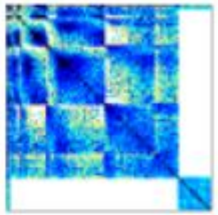
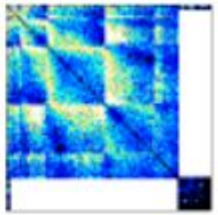
# Ambiente de execução

- Intel Core I5 8400
- 32 Gbs de memória ram

Engenharia de Sistemas e Computação PESC/Coppe

# Buscar matriz no ssgetpy

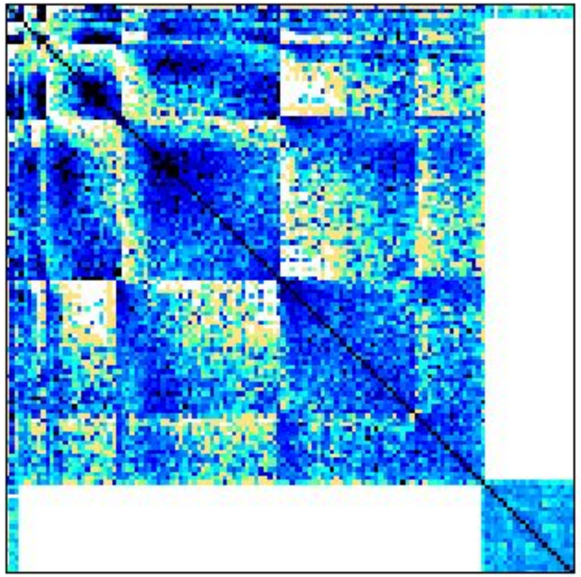
```
matrix = ssgetpy.search(rowbounds=(100_000,150_000),
                        colbounds=(100_000,150_000),
                        nzbounds = (0,1_000_000),
                        isspd = True)
matrix
```

| Id | Group | Name | Rows | Cols | NNZ | DType | 2D/3D Discretization? | SPD? | Pattern Symmetry | Numerical Symmetry | Kind | Spy Pl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2257 | Botonakis | thermomech_TC | 102158 | 102158 | 711558 | real | Yes | Yes | 1.0 | 1.0 | thermal problem |  |
| 2258 | Botonakis | thermomech_TK | 102158 | 102158 | 711558 | real | Yes | Yes | 1.0 | 1.0 | thermal problem |  |

# Matriz escolhida

Id: 2257 Grupo: Botonakis Nome: thermomech_TC

- Matriz com 102.158 linhas e colunas
- Tipo Real, com 711.558 números diferentes de 0
- Origem : Problema térmico



# Condicionamento

```python
max_eig = np.abs(eigsh(A, k = 1, which='LM',return_eigenvectors=False)[0])
min_eig = np.abs(eigsh(A, k = 1, which='SM',return_eigenvectors=False)[0])
```

- Normal: $\mathbf{A}\mathbf{x} = \mathbf{b}$ $\quad cond(\mathbf{A}) = \dfrac{|\lambda_{\max}(\mathbf{A})|}{|\lambda_{\min}(\mathbf{A})|} = 67.29$

- Precondicionado (diagonal): $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ $\quad cond(\mathbf{M}^{-1}\mathbf{A}) = \dfrac{|\lambda_{\max}(\mathbf{M}^{-1}\mathbf{A})|}{|\lambda_{\min}(\mathbf{M}^{-1}\mathbf{A})|} = 3.98$

Engenharia de Sistemas e Computação PESC/Coppe

# Algoritmos

**Steepest Descent**

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$\delta \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta$
While $i < i_{max}$ and $\delta > \varepsilon^2 \delta_0$ do
  $q \Leftarrow Ar$
  $\alpha \Leftarrow \frac{\delta}{r^T q}$
  $x \Leftarrow x + \alpha r$
  If $i$ is divisible by 50
    $r \Leftarrow b - Ax$
  else
    $r \Leftarrow r - \alpha q$
  $\delta \Leftarrow r^T r$
  $i \Leftarrow i + 1$

**Conjugate Gradients**

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow r$
$\delta_{new} \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta_{new}$
While $i < i_{max}$ and $\delta_{new} > \varepsilon^2 \delta_0$ do
  $q \Leftarrow Ad$
  $\alpha \Leftarrow \frac{\delta_{new}}{d^T q}$
  $x \Leftarrow x + \alpha d$
  If $i$ is divisible by 50
    $r \Leftarrow b - Ax$
  else
    $r \Leftarrow r - \alpha q$
  $\delta_{old} \Leftarrow \delta_{new}$
  $\delta_{new} \Leftarrow r^T r$
  $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
  $d \Leftarrow r + \beta d$
  $i \Leftarrow i + 1$

- Retirados do material de apoio
- Modificação na verificação de convergência
- Tolerância $10^{-8}$

$$Erro_k = ||\mathbf{x}_k - \mathbf{x}^*||_2 = ||\mathbf{x}_k - \mathbf{0}||_2 = ||\mathbf{x}_k||_2$$

# Implementação

Engenharia de Sistemas e Computação PESC/Coppe

### Steepest Descent

```
def steepest_descent(A,b,qtd,tol):
    n = A.shape[0]
    x = np.random.rand(n,1)

    itters = [0]
    results = [np.linalg.norm(x,2)]

    res = np.subtract(b, A.dot(x))
    alfa_num = np.matrix.transpose(res).dot(res)

    for i in range(qtd):
        q = A.dot(res)
        alfa_deno = np.matrix.transpose(res).dot(q)
        alfa = np.divide(alfa_num, alfa_deno)

        x = np.add(x, alfa*res)

        norm_x = np.linalg.norm(x,2)

        itters.append(i+1)
        results.append(norm_x)

        if(i % 50 == 0):
            res = np.subtract(b, A.dot(x))
        else:
            res = np.subtract(res, alfa*q)

        alfa_num = np.matrix.transpose(res).dot(res)

        if norm_x <= tol:
            return (itters,results)

    return False
```

### Steepest Descent Diagonal Preconditioned

```
def steepest_descent_diagonal(A,b,qtd,tol):
    n = A.shape[0]
    x = np.random.rand(n,1)

    itters = [0]
    results = [np.linalg.norm(x,2)]

    res = np.subtract(b, A.dot(x))
    M = A.diagonal()
    M_inv = spdiags(np.divide(eye(n).data, M), diags= 0,  m = n, n = n)
    z = M_inv.dot(res)
    alfa_num = np.matrix.transpose(z).dot(res)

    for i in range(qtd):

        q = A.dot(z)
        alfa_deno = np.matrix.transpose(z).dot(q)
        alfa = np.divide(alfa_num,alfa_deno)

        x = np.add(x, alfa*z)

        norm_x = np.linalg.norm(x,2)

        itters.append(i+1)
        results.append(norm_x)

        if(i % 50 == 0):
            res = np.subtract(b, A.dot(x))
        else:
            res = np.subtract(res, alfa*q)

        z = M_inv.dot(res)

        alfa_num = np.matrix.transpose(z).dot(res)

        if  norm_x  <= tol:
            return (itters,results)

    return False
```

# Implementação

Engenharia de Sistemas e Computação PESC/Coppe

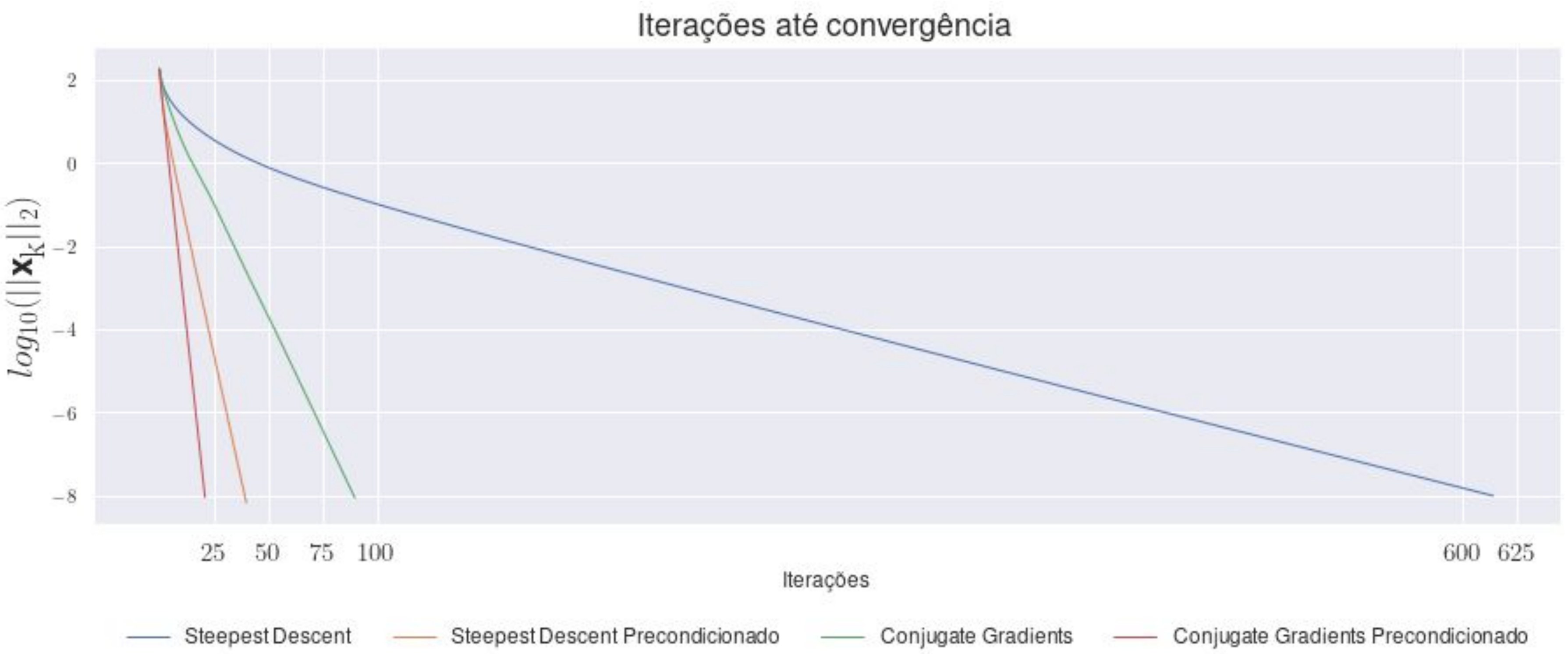
**Conjugate Gradients**

```
def conjugate_gradient(A,b,qtd,tol):
    x = np.random.rand(A.shape[0],1)

    itters = [0]
    results = [np.linalg.norm(x,2)]

    res = np.subtract(b, A.dot(x))
    res_t = np.matrix.transpose(res)
    d = res
    delta_new = res_t.dot(res)

    for i in range(qtd):
        q = A.dot(d)
        alpha = np.divide(delta_new, (np.matrix.transpose(d).dot(q)))

        x = np.add(x, alpha*d)

        norm_x = np.linalg.norm(x,2)

        if(i % 50 == 0):
            res = np.subtract(b, (A.dot(x)))
        else:
            res = np.subtract(res, alpha*q)

        delta_old = delta_new
        delta_new = np.matrix.transpose(res).dot(res)
        beta = np.divide(delta_new, delta_old)
        d = np.add(res, beta*d)

        itters.append(i+1)
        results.append(np.linalg.norm(x,2))

        if norm_x <= tol:
            return (itters,results)

    return False
```

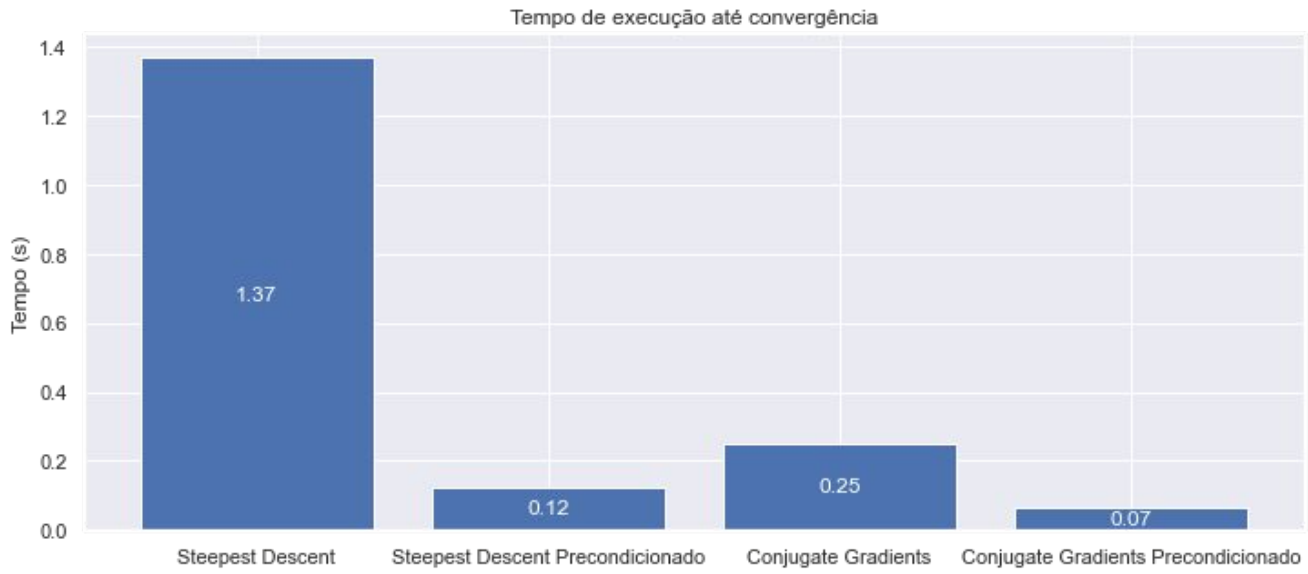
**Conjugate Gradients Diagonal Preconditioned**

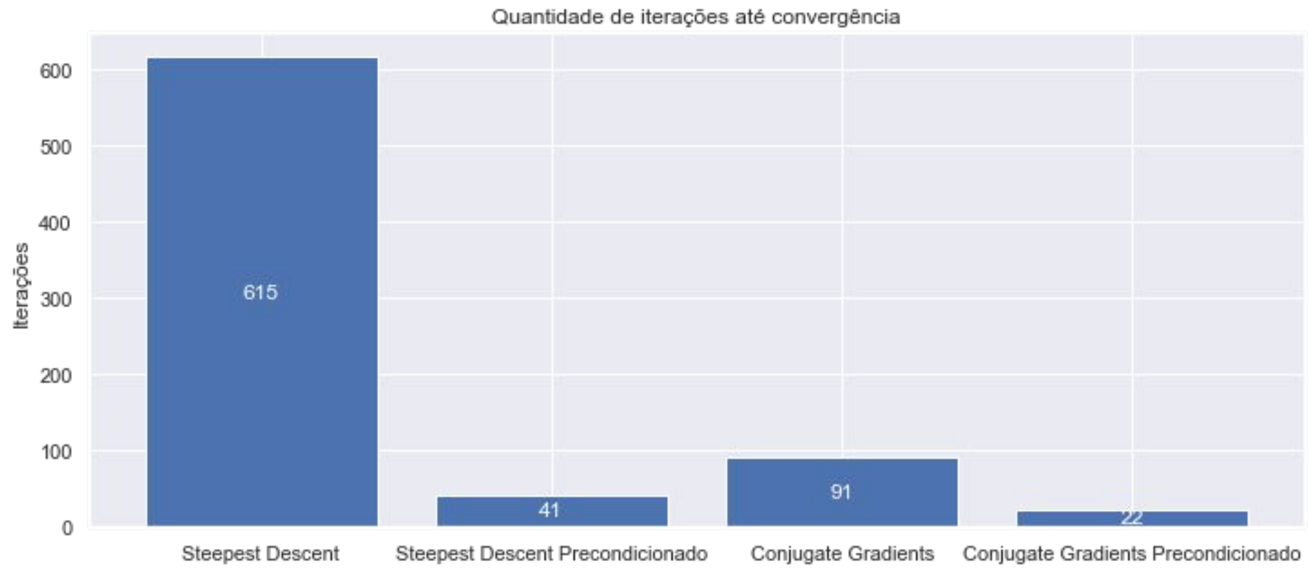
```
def conjugate_gradient_diagonal(A,b,qtd,tol):
    n = A.shape[0]
    x = np.random.rand(n,1)

    itters = [0]
    results = [np.linalg.norm(x,2)]

    res = np.subtract(b, A.dot(x))
    res_t = np.matrix.transpose(res)
    M = matrix.diagonal()
    M_inv = spdiags(np.divide(eye(n).data, M), diags= 0,  m = n, n = n)
    d = M_inv.dot(res)
    delta_new = res_t.dot(d)

    for i in range(qtd):
        q = A.dot(d)
        alpha = np.divide(delta_new, (np.matrix.transpose(d).dot(q)))

        x = np.add(x, alpha*d)
        norm_x = np.linalg.norm(x,2)

        if(i % 50 == 0):
            res = np.subtract(b, (A.dot(x)))
        else:
            res = np.subtract(res, alpha*q)

        s = M_inv.dot(res)
        delta_old = delta_new
        delta_new = np.matrix.transpose(res).dot(s)
        beta = np.divide(delta_new,delta_old)
        d = np.add(s, beta*d)

        itters.append(i+1)
        results.append(norm_x)

        if norm_x <= tol:
            return (itters,results)

    return False
```

Engenharia de Sistemas e Computação PESC/Coppe



- Executado 150 vezes
- Média do tempo das últimas 100



# Conclusões

- O método Conjugate Gradients converge mais rápido
- O precondicionamento pela diagonal ajudou para essa matriz
- Pode piorar em outras matrizes