ANÁLISE NUMÉRICA I -
Sistemas de Equações Lineares

UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Curso:
Licenciatura em Engenharia Informática

Unidade Curricular:
Análise Numérica I

Docente:
Hermenegildo Borges de Oliveira

Realizado por:
Daniel Maryna (64611)
Miguel Silva (80072)
Francisco Nunes (80061)
Brandon Mejia (79261)

11/2024

CONTEÚDO

INTRODUÇÃO

A **resolução de sistemas de equações lineares** é uma das bases da matemática aplicada e desempenha um papel essencial em várias áreas, como a engenharia, a física, a computação e a economia. Estes sistemas surgem frequentemente na modelagem de problemas do mundo real, como simulações de circuitos elétricos, análise estrutural de edifícios, otimização de recursos e processamento de imagens. Existem diferentes abordagens para resolver sistemas de equações lineares, que podem ser classificadas em métodos diretos e iterativos. **Os métodos iterativos, como o Método de Jacobi e o Método de Gauss-Seidel, são particularmente úteis para sistemas de grande dimensão** ou com matrizes esparsas, pois permitem obter soluções aproximadas com eficiência e precisão ajustável. Além disso, oferecem a vantagem de reduzirem a necessidade de operações aritméticas intensivas em comparação com métodos diretos, como a eliminação de Gauss.
O presente trabalho tem como objetivo **desenvolver e analisar dois algoritmos numéricos baseados no Método de Jacobi e no Método de Gauss-Seidel, implementados em Python**, de acordo com os critérios estabelecidos. Esses algoritmos são configurados para lidar com tolerâncias relativas inferiores a $10^{(-4)}$ e para realizar até 10 iterações, permitindo um equilíbrio entre precisão e desempenho.

A escolha dos métodos de Jacobi e Gauss-Seidel não é apenas técnica, mas também prática, dado o seu uso consolidado em aplicações industriais e científicas. Este trabalho pretende consolidar o entendimento teórico e prático sobre a utilização de métodos iterativos para sistemas lineares, evidenciando sua relevância na Análise Numérica e na computação científica.

FUNDAMENTAÇÃO TEÓRICA

Os **sistemas de equações lineares** são expressões matemáticas compostas por múltiplas equações com múltiplas incógnitas, frequentemente representados na forma matricial $Ax = b$, onde:

- A é uma matriz de coeficientes;
- x é o vetor de incógnitas;
- b é o vetor dos termos independentes.

Resolver um sistema de equações lineares significa determinar o vetor x que satisfaz a equação. Para isso, existem métodos diretos e iterativos. Neste trabalho, destacamos dois métodos iterativos: **Jacobi** e **Gauss-Seidel**.

Método de Jacobi

O Método de Jacobi é uma técnica iterativa que utiliza uma forma de substituição sucessiva para aproximar a solução de um sistema linear. A partir de uma estimativa inicial para o vetor x, calcula-se uma nova aproximação, considerando que cada incógnita depende apenas dos valores antigos das outras incógnitas. A fórmula geral para o cálculo de $x_i^{(k+1)}$ em cada iteração k é dada por:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\right)$$

onde a_{ii} são os elementos diagonais da matriz A, e a_{ij} são os elementos fora da diagonal.

Condições de Convergência: O Método de Jacobi converge se a matriz A for estritamente diagonal dominante ou se for simétrica e definida positiva. Em geral, a convergência é sensível à escolha dos valores iniciais.

Método de Gauss-Seidel

O Método de Gauss-Seidel é uma variação do Método de Jacobi que utiliza imediatamente os valores mais recentes calculados para as incógnitas, sempre que disponíveis. A fórmula geral para $x_i^{(k+1)}$ é:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_i^{(k+1)} - \sum_{j>i} a_{ij}x_i^{(k)}\right)$$

Este método tende a convergir mais rapidamente do que o de Jacobi, dado que incorpora as atualizações logo que são calculadas.

Condições de Convergência: Assim como o Método de Jacobi, o Método de Gauss-Seidel requer que a matriz A seja estritamente diagonal dominante ou simétrica e definida positiva.

Tolerância e Critérios de Paragem

Ambos os métodos utilizam critérios de paragem baseados em tolerâncias para determinar a precisão da solução aproximada. A **tolerância relativa** ε é definida por:

$$\varepsilon = \frac{\left\|x^{(k+1)} - x^{(k)}\right\|}{\|x^{(k+1)}\|}$$

e o processo iterativo é interrompido quando ε é menor que um valor previamente estabelecido, como $10^{(-4)}$.

Comparação entre os Métodos

Método de Jacobi:

- Utiliza apenas os valores da iteração anterior para calcular a nova aproximação, o que facilita a paralelização, tornando-o mais adequado para sistemas distribuídos.
- Contudo, a convergência é geralmente mais lenta, pois os ajustes não incorporam os valores mais atualizados durante a iteração.

Método de Gauss-Seidel:

- Aproveita imediatamente os valores mais recentes calculados, permitindo que cada iteração se aproxime mais rapidamente da solução. Essa abordagem resulta numa convergência mais eficiente em muitos casos.
- Apesar disso, não é tão simples de implementar em sistemas paralelos, devido à dependência dos cálculos sequenciais.

No geral, o Método de Jacobi é indicado para sistemas onde a execução paralela é importante, enquanto o Método de Gauss-Seidel é mais eficiente em termos de número de iterações para sistemas bem condicionados.

Ambos os métodos são amplamente utilizados na engenharia e ciência computacional, especialmente em problemas envolvendo grandes sistemas esparsos, onde os métodos diretos, como a eliminação de Gauss, podem ser computacionalmente inviáveis.

IMPLEMENTAÇÃO DOS PROGRAMAS

Os programas implementados utilizam os métodos iterativos de **Jacobi** e **Gauss-Seidel** para resolver sistemas de equações lineares. Desenvolvidos em ***Python***, estes programas apresentam uma estrutura modular e clara, onde cada função desempenha um papel específico no processo de cálculo das soluções aproximadas.

O **método de Jacobi** segue uma abordagem organizada em três etapas principais:

1. **Verificação da Dominância Diagonal**:
   - Feita através da função `isDDM`, que assegura que a matriz dos coeficientes (A) é diagonalmente dominante, uma condição essencial para garantir a convergência.
2. **Cálculo da Primeira Iteração (X_0)**:
   - Executada pela função `get_X0_primeira_iteracao`, que utiliza a diagonal da matriz (A) e o vetor de termos independentes (B) para obter valores iniciais aproximados de X_0[1].
   - Complementada pela função `primeira_iteracao`, que decompõe a matriz A nos componentes D[2], L[3] e U[4], calculando $T = -D^{-1}(L + U)$ e $C = D^{-1}B$, elementos fundamentais para as iterações subsequentes.
3. **Execução do Processo Iterativo**:
   - Realizada pela função `metodo_Jacobi`, que utiliza T[5], C[6] e X_0 para calcular iterativamente X_k, verificando a convergência em cada passo com base nos critérios de paragem.

O **método de Gauss-Seidel** distingue-se por utilizar imediatamente os valores mais recentes calculados, proporcionando uma convergência geralmente mais rápida. Este método é composto por:

1. **Cálculo das Próximas Iterações**:
   - A função `proxima_iteracao` atualiza as aproximações das incógnitas em X, utilizando os valores mais recentes e as relações definidas pela matriz A e pelo vetor B.
2. **Execução do Método Completo**:
   - Conduzida pela função `metodo_Gauss_Seidel`, que inicia o processo com os valores de X_0 (calculados pelo método de Jacobi) e realiza as iterações necessárias para aproximar a solução, interrompendo o cálculo quando os critérios de paragem são atingidos.

[1] Vetor Inicial
[2] Matriz Diagonal de A
[3] Matriz triangular inferior de A
[4] Matriz triangular superior de A
[5] Matriz de transformação calculada como $T = -D^{-1}(L + U)$
[6] Vetor constante calculado como $C = D^{-1}B$

Método de Jacobi

1. isDDM(A):
   - **O que faz:** Verifica se a matriz A é diagonalmente dominante. Esta propriedade é fundamental para garantir a convergência do método.

```
def isDDM(m:Matrix):

    n = m.rows # number of rows
    # for each row
    for i in range(0, n):
        # for each column, finding sum of each row.
        sum = 0
        for j in range(0, n):
            sum += abs(m[i,j])
        # removing the diagonal element.
        sum -= abs(m[i,i])

        # checking if diagonal element is less than sum of non-
diagonal elements.
        if abs(m[i,i]) < sum:
            return False
    return True
```

Código 1 - Implementação da função isDDM.

2. get_X0_primeira_iteracao(D, B):
   - **O que faz:** Calcula o vetor inicial X_0 dividindo os elementos do vetor B pelos valores correspondentes da diagonal de D. Este vetor serve como ponto de partida para o processo iterativo.

```
def get_X0_primeira_iteracao(D: Matrix, B: Matrix):
    x_0 = []
    for i in range(0, B.shape[0]):
        x_0.append(B[i, 0] / D[i, i])

    return Matrix(x_0).reshape(B.shape[0], 1) # Retorna uma matriz
coluna
```

Código 2 - Implementação da função get_X0_primeira_iteracao.

3. primeira_iteracao(A, B):
   - **O que faz:** Decompõe a matriz A em suas componentes D, L e U, calculando os elementos T e C, usados nas iterações subsequentes. Também se obtém X_0 para iniciar as iterações.

```python
def primeira_iteracao(A: Matrix, B: Matrix):
    D = diag(*A.diagonal())  # STEP 1, DIVIDE A EM "D", "L" e 'U'
    L = A.lower_triangular() - D
    U = A.upper_triangular() - D
    D_inv = D.inv()  # STEP 2, CALCULA D⁻¹
    T = -D_inv * (L + U)  # STEP 3, CALCULA T
    C = D_inv * B  # STEP 4, CALCULA C
    X_0 = get_X0_primeira_iteracao(D, B)  # STEP 5, CALCULA X₀
    return T, C, X_0
```

Código 3 - Implementação da função primeira_iteracao.

4. metodo_Jacobi(A, B):
   - **O que faz:** Executa o método de Jacobi completo, verificando as condições de convergência e iterando até atingir o critério de paragem.

```python
def metodo_Jacobi(A: Matrix, B: Matrix):
    if is_determinante_zero(A):
        raise Exception("Determinante da matriz é zero, sistema impossível.")
    if not isDDM(A):
        print("A matriz não é diagonalmente dominante, pode não convergir.")
        if prompt_is_not_diagonalmente_dominante() == 0:
            return
iteracoes, erro = ITERACOES_ERRO
T, C, X_anterior = primeira_iteracao(A, B)
X_proximo = None
aproximacoes_iteracoes_jacobi.append([float(x) for x in X_anterior])
    for i in range(1, iteracoes + 1):
        X_proximo = T * X_anterior + C
        aproximacoes_iteracoes_jacobi.append([float(x) for x in X_proximo])
        if max((X_proximo - X_anterior).applyfunc(abs)) < erro:
            print(f"Convergiu na iteração {i}")
            return X_proximo
        X_anterior = X_proximo
    print("Número máximo de iterações excedido.")
    return X_proximo
```

Código 4 - Implementação da função metodo_Jacobi.

Método de Gauss-Seidel

1. proxima_iteracao(A, B, X):
   - **O que faz:** Calcula os valores atualizados do vetor X para a próxima iteração, utilizando os coeficientes da matriz A e os valores mais recentes das incógnitas.

```python
def proxima_iteracao(A: Matrix, B: Matrix, X: Matrix):
    n = A.shape[0]
    x_novo = 0.0
    for i in range(0, n):
        for j in range(0, n):
            if i == j:
                x_novo += (B[j, 0])
            elif i != j:
                x_novo += A[i, j] * X[j, 0] * -1
        X[i, 0] = x_novo * (1 / A[i, i])
        x_novo = 0.0
    return X
```

Código 5 - Implementação da função proxima_iteracao.

2. metodo_Gauss_Seidel(A, B):
   - **O que faz:** Implementa o método completo de Gauss-Seidel, verificando as condições de convergência e iterando até atingir o critério de paragem.

```
def metodo_Gauss_Seidel(A: Matrix, B: Matrix):
    if is_determinante_zero(A):
        raise Exception("Determinante da é zero, sistema impossível.")
    if not isDDM(A):
        print("Não é diagonalmente dominante, o método pode não
convergir.")
        if prompt_is_not_diagonalmente_dominante() == 0:
            return
    # Se a matriz for diagonalmente dominante ou o utilizador decidir
continuar
    iteracoes, erro = ITERACOES_ERRO
    X_0 = get_X0_primeira_iteracao(A, B)
    X_anterior = proxima_iteracao(A, B, X_0)
    X_proximo = None
    # Armazenar a primeira aproximação
    aproximacoes_iteracoes_gauss_seidel.append([float(x) for x in X_0])
    aproximacoes_iteracoes_gauss_seidel.append([float(x) for x in
X_anterior])
    for i in range(2, iteracoes):
        X_proximo = proxima_iteracao(A, B, X_anterior)  # Procura o próximo
valor de X^(k+1)
        aproximacoes_iteracoes_gauss_seidel.append([float(x) for x in
X_proximo])
        # Critério de parada
        if max((X_proximo - X_anterior).applyfunc(abs)) < erro:
            print(f"Convergiu na iteração {i}")
            return X_proximo
        X_anterior = X_proximo
    # Caso não tenha convergido, imprimir a última aproximação encontrada
    print("Número máximo de iterações excedido. Procedimento concluído sem
sucesso.")
    return X_proximo
```

Código 6 - Implementação da função metodo_Gauss_Seidel.

FLUXO DE EXECUÇÃO

O fluxo de execução do programa consiste nos seguintes passos principais, comuns aos dois métodos (Jacobi e Gauss-Seidel):

1. **Entrada de Dados**:
    - O programa solicita ao utilizador o número de linhas e colunas da matriz A, que deve ser quadrada.
    - Em seguida, o utilizador insere os valores das linhas da matriz A e da matriz B, que representa os termos independentes.
2. **Seleção do Método**:
    - O utilizador escolhe o método que deseja aplicar:
        - **1:** Método de Gauss-Jordan (não relevante para esta análise).
        - **2:** Método de Jacobi.
        - **3:** Método de Gauss-Seidel.
3. **Execução do Método**:
    - O programa realiza as operações necessárias para calcular uma solução aproximada do sistema de equações, utilizando o método selecionado.
    - Verifica condições de convergência, como a dominância diagonal da matriz A, e avisa o utilizador caso estas não sejam satisfeitas, permitindo que ele decida continuar ou não.
4. **Resultados e Interações**:
    - O programa informa se o método convergiu ou se o número máximo de iterações foi excedido.
    - Oferece ao utilizador a opção de visualizar todas as iterações realizadas, caso tenha interesse.

RESULTADOS

Método de Gauss-Seidel

```
Digite o numero de linhas/colunas da matriz (Quadrada):
2
Digite as linhas da matriz A (Quadrada):
4 -1
-2 4
Digite a linha da matriz B (Vertical):
2 4
1) Metodo de Gauss Jordan
2) Metodo de Jacobi
3) Metodo de Gauss Seidel
Qual o método que deseja aplicar?
3
Convergiu na iteração 2
Deseja ver todas as iterações do método? (Y/N)
N
Ordem da aproximação: | X₀ | X₁ | X₂ | ... | Xₙ |
Solução aproximada:  Matrix([[0.843750000000000], [1.42187500000000]])
```

Ilustração 1 - Execução do método Gauss-Seidel.

Matrizes Fornecidas

- Matriz dos Coeficientes (A):

$$A = \begin{bmatrix} 4 & -1 \\ -2 & 4 \end{bmatrix}$$

- Vetor dos Termos Independentes (B):

$$B = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Processo e Resultados

1. O utilizador selecionou o **método de Gauss-Seidel** (opção 3).
2. O programa informa que **convergiu na iteração 2**. Isso significa que, após duas iterações, a diferença entre as aproximações consecutivas para X ficou abaixo da tolerância definida $10^{(-4)}$.
3. A solução aproximada obtida foi:

$$X = \begin{bmatrix} 0.84375 \\ 1.421875 \end{bmatrix}$$

Interpretação com Fundamentação Teórica

A matriz A é **diagonalmente dominante**, pois:

- Para a primeira linha: $|4| > |-1|$
- Para a segunda linha: $|4| > |-2|$

Isso garante que o método de Gauss-Seidel converge para a solução correta.

O sistema de equações lineares representado é:

$$4x_1 - x_2 = 2$$
$$-2x_1 + 4x_2 = 4$$

A solução encontrada é uma aproximação precisa da solução exata, validando a eficiência do método.

Método de Jacobi

```
Digite o numero de linhas/colunas da matriz (Quadrada):
3
Digite as linhas da matriz A (Quadrada):
2 3 1
4 2 5
1 5 3
Digite a linha da matriz B (Vertical):
5 10 8
1) Metodo de Gauss Jordan
2) Metodo de Jacobi
3) Metodo de Gauss Seidel
Qual o método que deseja aplicar?
2
A matriz não é diagonalmente dominante, o método de Jacobi pode não convergir.
Deseja continuar? (Y/N)y
Número máximo de iterações excedido. Procedimento concluído sem sucesso.
Deseja ver todas as iterações do método? (Y/N)
y
--------------------------------------------------------------------------------

Todas as aproximações (em valores decimais):
--->| X₀ | X₁ | X₂ | ... | Xₙ |

Iteração 0: [2.5, 5.0, 2.6666666666666665]
Iteração 1: [-6.333333333333334, -6.666666666666666, -6.5]
Iteração 2: [15.75, 33.91666666666667, 15.888888888888886]
Iteração 3: [-56.31944444444445, -66.22222222222221, -59.111111111111114]
Iteração 4: [131.38888888888886, 265.4166666666667, 131.81018518518516]
Iteração 5: [-461.5300925925926, -587.3032407407406, -483.4907407407407]
Iteração 6: [1125.2002314814813, 2136.787037037037, 1135.3487654320986]
Iteração 7: [-3770.354938271605, -5083.772376543209, -3933.711805555555]
Iteração 8: [9595.014467592591, 17379.9893904321, 9732.405606995882]
Iteração 9: [-30933.68688914609, -43516.04295267489, -32162.320473251024]
Iteração 10: [81357.72466563786, 142278.17496141975, 82840.63388417351]

Solução aproximada:  Matrix([[81357.7246656379], [142278.174961420], [82840.6338841735]])
```

Ilustração 2 - Execução do método Jacobi.

Matrizes Fornecidas

- Matriz dos Coeficientes (A):

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 4 & 2 & 5 \\ 1 & 5 & 3 \end{bmatrix}$$

- Vetor dos Termos Independentes (B):

$$B = \begin{bmatrix} 5 \\ 10 \\ 8 \end{bmatrix}$$

Processo e Resultados

1. O utilizador selecionou o **método de Jacobi** (opção 2).
2. O programa verificou que a matriz A **não é diagonalmente dominante**, e alertou o utilizador:
    - Para a primeira linha: $|2| < |3| + |1|$
    - Para a segunda linha: $|2| < |4| + |5|$
    - Para a terceira linha: $|3| < |1| + |5|$
3. Mesmo assim, o utilizador optou por continuar.
4. O programa não conseguiu atingir a convergência dentro do número máximo de iterações (10), resultando em aproximações que divergiram, como mostrado:
    - Última iteração:

$$X = \begin{bmatrix} 81357.72 \\ 142278.17 \\ 82840.63 \end{bmatrix}$$

Interpretação com Fundamentação Teórica

- A ausência de dominância diagonal implica que o método de Jacobi pode não convergir, como foi o caso.
- Os valores obtidos cresceram de forma descontrolada (divergência), o que confirma que o método depende fortemente das condições de convergência.

CONCLUSÃO

No presente estudo, desenvolvemos uma aplicação dos métodos iterativos de **Jacobi** e **Gauss-Seidel** para a resolução de **sistemas de equações lineares**. Através da implementação em **Python** e da análise prática dos resultados, foi possível verificar as **diferenças** entre os métodos, bem como os **fatores que influenciam o seu desempenho e convergência**.

Os testes realizados destacaram a importância das **condições de convergência**, como a **dominância diagonal** da matriz dos coeficientes, que foi determinante para o sucesso do método de Jacobi. Observámos que, na ausência dessa propriedade, este método **não conseguiu convergir**, gerando resultados **divergentes**. Este comportamento reforça a **sensibilidade do método de Jacobi** às características da matriz.

Por outro lado, o **método de Gauss-Seidel** demonstrou ser mais **eficiente e robusto**, convergindo rapidamente para a solução do sistema quando aplicado a uma matriz diagonalmente dominante. A capacidade deste método de utilizar imediatamente os **valores mais recentes** durante o processo iterativo contribuiu para a sua **eficiência**, como evidenciado no exemplo onde convergiu em apenas **2 iterações**.

Além disso, o trabalho reforçou a relevância de critérios como o **número máximo de iterações** e a **tolerância**, que asseguram a **precisão das soluções aproximadas** e evitam processos iterativos excessivamente longos. Também evidenciámos a importância da **escolha adequada do método** com base nas características do sistema, realçando as **vantagens e limitações de cada abordagem**.

Concluímos que os **métodos iterativos** são ferramentas poderosas para a resolução de sistemas lineares, especialmente para **matrizes de grandes dimensões ou esparsas**, desde que as condições de convergência sejam satisfeitas. Este trabalho proporcionou uma compreensão aprofundada dos **métodos de Jacobi e Gauss-Seidel**, consolidando conceitos fundamentais da **análise numérica** e da **computação científica**.

REFERÊNCIAS

Conteúdo Programático da Disciplina de Análise Numérica 1:

- Capítulo 4: Sistemas de Equações Lineares (PDF fornecido durante as aulas).

Material Complementar:

- Notas de aula fornecidas pelo professor Hermenegildo Borges de Oliveira, abordando os métodos iterativos de Jacobi e Gauss-Seidel, com ênfase em suas condições de convergência.

Livro Texto:

- **Burden, R. L., & Faires, J. D.** (2016). *Análise Numérica*. 10ª edição. Cengage Learning.

Biblioteca Python Utilizada:

- Documentação oficial da **SymPy**: https://www.sympy.org

Sites Consultados para Validação Teórica:

- GeeksforGeeks: Verificação de matrizes diagonalmente dominantes. Disponível em: https://www.geeksforgeeks.org
- Wolfram MathWorld: Descrição e propriedades de métodos iterativos. Disponível em: https://mathworld.wolfram.com