Universidade do Algarve

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informática

Análise Numérica I

Trabalho 3 – Sistemas de Equações Lineares

Método da Jacobi e Método de Gauss-Seidel

Realizado por: João Andréz nº61066, Carlos Ferreira nº 71319, Diogo Freitas nº 90147, Diogo Carvalho nº 90247

Docente: Hermenegildo Borges de Oliveira

Ano Letivo: 2025/2026

Introdução

O objetivo deste trabalho é sumariamente a utilização de dois métodos distintos, método de Jacobi e de Gauss-Seidel, para a resolução de sistemas de equações lineares, utilizando uma implementação em Python. Neste trabalho, como requerido, ambos os métodos serão capazes de trabalhar com até 10 iterações e com tolerâncias relativas menores do que 10^{-4}.

Os métodos iterativos utilizados permitem obter soluções aproximadas através de sucessivas atualizações do vetor solução, partindo de uma aproximação inicial.

Iremos também apresentar testes do código que comprovam o seu funcionamento como esperado. Esses testes servem também para realizar a comparação das soluções aproximadas obtidas em cada método.

Código

Main.py

```
# TRABALHO 3 - Sistemas de Equações Lineares

# João Andréz - 61066

# Carlos Ferreira - 71319

# Diogo Freitas - 90147

# Diogo Carvalho - 90247


import sympy as sp  # type: ignore

from sympy import Matrix, Abs  # type: ignore


TOLERANCE_J = 1e-4

MAX_ITERATIONS_J = 10

TOLERANCE_GS = 1e-4

MAX_ITERATIONS_GS = 10


def parse_num(s):
```

```
    """
    Converte uma string numérica para float usando sympy.
    Aceita frações (ex: "1/3") e expressões simples (ex: "sqrt(2)",
"2*sqrt(3)").
    """
    s = s.strip().replace(",", ".")
    try:
        return float(sp.sympify(s))
    except Exception:
        raise ValueError(f"Valor inválido: {s}")


def is_diagonally_dominant_strict(A):
    """Dominância diagonal estrita por linhas."""
    n = A.rows
    for i in range(n):
        diagonal = Abs(A[i, i])
        row_sum = sum(Abs(A[i, j]) for j in range(n) if j != i)
        if diagonal <= row_sum:
            return False
    return True


def is_diagonally_dominant(A):
    """Dominância diagonal (não estrita) por linhas."""
    n = A.rows
    for i in range(n):
        diagonal = Abs(A[i, i])
        row_sum = sum(Abs(A[i, j]) for j in range(n) if j != i)
        if diagonal < row_sum:
```

```
            return False
    return True


def ensure_nonzero_diagonal(A, b):
    """
    Tenta reordenar linhas para evitar zeros na diagonal.
    Retorna (A2, b2, mudou?)
    """
    n = A.rows
    A2 = Matrix(A)
    b2 = Matrix(b)
    changed = False

    for i in range(n):
        if A2[i, i] != 0:
            continue

        swap_row = None
        for r in range(i + 1, n):
            if A2[r, i] != 0:
                swap_row = r
                break

        if swap_row is None:
            return A2, b2, changed

        A2.row_swap(i, swap_row)
```

```
            b2.row_swap(i, swap_row)
            changed = True

    return A2, b2, changed

def jacobi_method(A, b, x0):
    n = A.rows
    x = Matrix(x0)

    for k in range(MAX_ITERATIONS_J):
        x_new = Matrix.zeros(n, 1)

        for i in range(n):
            sum_val = sum(A[i, j] * x[j] for j in range(n) if j !=
i)
            x_new[i] = (b[i] - sum_val) / A[i, i]

        if x.norm() != 0:
            relative_error = (x_new - x).norm() / x.norm()
        else:
            relative_error = (x_new - x).norm()

        if relative_error < TOLERANCE_J:
            return x_new, k + 1

        x = x_new
```

```
    print(f"AVISO: Método não convergiu em {MAX_ITERATIONS_J}
iterações")
    return x, MAX_ITERATIONS_J


def gauss_seidel_method(A, b, x0):
    n = A.rows
    x = Matrix(x0)

    for k in range(MAX_ITERATIONS_GS):
        x_old = Matrix(x)

        for i in range(n):
            sum1 = sum(A[i, j] * x[j] for j in range(i))
            sum2 = sum(A[i, j] * x_old[j] for j in range(i + 1, n))
            x[i] = (b[i] - sum1 - sum2) / A[i, i]

        if x_old.norm() != 0:
            relative_error = (x - x_old).norm() / x_old.norm()
        else:
            relative_error = (x - x_old).norm()

        if relative_error < TOLERANCE_GS:
            return x, k + 1

    print(f"AVISO: Método não convergiu em {MAX_ITERATIONS_GS}
iterações")
    return x, MAX_ITERATIONS_GS
```

```
def read_int(prompt, min_value=None):
    while True:
        s = input(prompt).strip()
        try:
            v = int(s)
            if min_value is not None and v < min_value:
                print(f"Valor inválido (mínimo {min_value}).")
                continue
            return v
        except ValueError:
            print("Introduza um inteiro válido.")


def get_matrix_input():
    """Lê matriz A, vetor b e x0 da consola com validações."""
    print("Dimensão da matriz (n x n):")
    n = read_int("n = ", min_value=1)


    print(f"\nIntroduza os coeficientes da matriz A ({n}x{n}):")
    A_data = []
    for i in range(n):
        while True:
            row = input(f"Linha {i + 1} (separados por espaço):
").strip().split()
            if len(row) != n:
                print(f"Essa linha tem {len(row)} elementos;
esperava {n}. Tenta de novo.")
                continue
            try:
```

```
                A_data.append([parse_num(x) for x in row])
                break
            except ValueError:
                print("Linha inválida. Usa números, frações (1/3) ou
sqrt(...).")
    A = Matrix(A_data)

    while True:
        b_row = input("\nIntroduza o vetor b (n valores separados
por espaço): ").strip().split()
        if len(b_row) != n:
            print(f"b tem {len(b_row)} valores; esperava {n}.")
            continue
        try:
            b = Matrix([parse_num(x) for x in b_row])
            break
        except ValueError:
            print("b inválido. Usa números, frações (1/3) ou
sqrt(...).")

    while True:
        x0_row = input(
            "\nIntroduza a aproximação inicial x0 (Carregue no Enter
para zeros):\n"
            "Valores (space-separated, or empty): "
        ).strip().split()

        if len(x0_row) == 0:
            x0 = Matrix.zeros(n, 1)
```

```
            break

        if len(x0_row) != n:
            print(f"x0 tem {len(x0_row)} valores; esperava {n}.")
            continue

        try:
            x0 = Matrix([parse_num(x) for x in x0_row])
            break
        except ValueError:
            print("x0 inválido. Usa números, frações (1/3) ou
sqrt(...).")

    return A, b, x0

def main():
    print("=" * 50)
    print("Sistemas de Equações Lineares — Métodos Iterativos")
    print("=" * 50)

    while True:
        print("\nSelecione um método:")
        print("1. Método de Jacobi")
        print("2. Método de Gauss-Seidel")
        print("0. Sair")

        method_choice = input("Escolha (0/1/2): ").strip()
        while method_choice not in ("0", "1", "2"):
```

```
            method_choice = input("Escolha (0/1/2): ").strip()


        if method_choice == "0":

            print("Xau :)")

            break


        A, b, x0 = get_matrix_input()


        A, b, changed = ensure_nonzero_diagonal(A, b)

        if changed:

            print("\nNota: reordenei linhas para evitar zeros na
diagonal.")


        if any(A[i, i] == 0 for i in range(A.rows)):

            print("\nERRO: A tem zero(s) na diagonal; não posso
aplicar Jacobi/GS.")

            retry = input("Resolver outro sistema? (s/n):
").strip().lower()

            if retry not in ("s", "sim", "y", "yes"):

                print("Xau :)")

                break

            continue


        if is_diagonally_dominant_strict(A):

            print("\nMatriz A é diagonalmente dominante estrita
(condição suficiente).")

        elif is_diagonally_dominant(A):

            print("\nMatriz A é diagonalmente dominante (não
estrita). Pode convergir, mas não é garantido.")
```

```
        else:
            print("\nAVISO: A não é diagonalmente dominante; o
método pode não convergir.")

        if method_choice == "1":
            print("\nMétodo de Jacobi:")
            x_sol, it = jacobi_method(A, b, x0)
        else:
            print("\nMétodo de Gauss-Seidel:")
            x_sol, it = gauss_seidel_method(A, b, x0)

        print(f"Solução aproximada: {x_sol}")
        print(f"Iterações: {it}")
        print("=" * 50)

        retry = input("\nResolver outro sistema? (s/n):
").strip().lower()
        if retry not in ("s", "sim", "y", "yes"):
            print("Xau :)")
            break

if __name__ == "__main__":
    main()
```

Resultados dos Testes

Método de Jacobi

```
Dimensão da matriz (n x n):
n = 2

Introduza os coeficientes da matriz A (2x2):
Linha 1 (separados por espaço): 4 1
Linha 2 (separados por espaço): 2 3

Introduza o vetor b (n valores separados por espaço): 1 2

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Jacobi:
AVISO: Método não convergiu em 10 iterações
Solução aproximada: Matrix([[0.0999871399176955], [0.599922839506173]])
Iterações: 10
```

```
Dimensão da matriz (n x n):
n = 3

Introduza os coeficientes da matriz A (3x3):
Linha 1 (separados por espaço): 10 -1 2
Linha 2 (separados por espaço): -1 11 -1
Linha 3 (separados por espaço): 2 -1 10

Introduza o vetor b (n valores separados por espaço): 6 25 -11

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0
x0 tem 2 valores; esperava 3.

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Jacobi:
Solução aproximada: Matrix([[1.04328483625982], [2.26921832989550], [-1.08170972374018]])
Iterações: 8
```

```
Dimensão da matriz (n x n):
n = 3

Introduza os coeficientes da matriz A (3x3):
Linha 1 (separados por espaço): 120 -5 10
Linha 2 (separados por espaço): -4 150 -6
Linha 3 (separados por espaço): 12 -8 130

Introduza o vetor b (n valores separados por espaço): 125 140 134

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Jacobi:
Solução aproximada: Matrix([[1.00001011338788], [0.999994133080089], [1.00000948805386]])
Iterações: 5
```

```
Dimensão da matriz (n x n):
n = 2

Introduza os coeficientes da matriz A (2x2):
Linha 1 (separados por espaço): 5 200
Linha 2 (separados por espaço): 150 4

Introduza o vetor b (n valores separados por espaço): 10 12

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0

AVISO: A não é diagonalmente dominante; o método pode não convergir.

Método de Jacobi:
AVISO: Método não convergiu em 10 iterações
Solução aproximada: Matrix([[-597773515677118.], [-364743162108072.]])
Iterações: 10
```

```
Dimensão da matriz (n x n):
n = 3

Introduza os coeficientes da matriz A (3x3):
Linha 1 (separados por espaço): 3/2 -1/4 0
Linha 2 (separados por espaço): -1/5 2 -1/3
Linha 3 (separados por espaço): 0 -1/2 5/2

Introduza o vetor b (n valores separados por espaço): 1 -2 3/2

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Jacobi:
Solução aproximada: Matrix([[0.520465277777778], [-0.877187500000000], [0.424558333333333]])
Iterações: 8
```

Método de Gauss-Seidel

```
Dimensão da matriz (n x n):
n = 2

Introduza os coeficientes da matriz A (2x2):
Linha 1 (separados por espaço): 4 1
Linha 2 (separados por espaço): 2 3

Introduza o vetor b (n valores separados por espaço): 1 2

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Gauss-Seidel:
Solução aproximada: Matrix([[0.100003215020576], [0.599997856652949]])
Iterações: 7
```

```
Dimensão da matriz (n x n):
n = 3

Introduza os coeficientes da matriz A (3x3):
Linha 1 (separados por espaço): 10 -1 2
Linha 2 (separados por espaço): -1 11 -1
Linha 3 (separados por espaço): 2 -1 10

Introduza o vetor b (n valores separados por espaço): 6 25 -11

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Gauss-Seidel:
Solução aproximada: Matrix([[1.04327066474257], [2.26923043426254], [-1.08173108952226]])
Iterações: 5
```

```
Dimensão da matriz (n x n):
n = 3

Introduza os coeficientes da matriz A (3x3):
Linha 1 (separados por espaço): 120 -5 10
Linha 2 (separados por espaço): -4 150 -6
Linha 3 (separados por espaço): 12 -8 130

Introduza o vetor b (n valores separados por espaço): 125 140 134

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Gauss-Seidel:
Solução aproximada: Matrix([[0.999999961688301], [1.00000007494909], [1.00000000814872]])
Iterações: 4
```

```
Dimensão da matriz (n x n):
n = 2

Introduza os coeficientes da matriz A (2x2):
Linha 1 (separados por espaço): 5 200
Linha 2 (separados por espaço): 150 4

Introduza o vetor b (n valores separados por espaço): 10 12

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0

AVISO: A não é diagonalmente dominante; o método pode não convergir.

Método de Gauss-Seidel:
AVISO: Método não convergiu em 10 iterações
Solução aproximada: Matrix([[7.38604903268846e+28], [-2.76976838725817e+30]])
Iterações: 10
```

```
Dimensão da matriz (n x n):
n = 3

Introduza os coeficientes da matriz A (3x3):
Linha 1 (separados por espaço): 3/2 -1/4 0
Linha 2 (separados por espaço): -1/5 2 -1/3
Linha 3 (separados por espaço): 0 -1/2 5/2

Introduza o vetor b (n valores separados por espaço): 1 -2 3/2

Introduza a aproximação inicial x0 (Carregue no Enter para zeros):
Valores (space-separated, or empty): 0 0 0

Matriz A é diagonalmente dominante estrita (condição suficiente).

Método de Gauss-Seidel:
Solução aproximada: Matrix([[0.520466666666667], [-0.877193333333333], [0.424561333333333]])
Iterações: 5
```