UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CIÊNCIA DA COMPUTAÇÃO

Leonardo Nascimento dos Santos
Vinícius Berger

COMPARAÇÃO DAS SOLUÇÕES E DO ESFORÇO COMPUTACIONAL NA RESOLUÇÃO DE UM SISTEMA DE EQUAÇÕES LINEARES VIA MÉTODO DE ELIMINAÇÃO DE GAUSS E VIA MÉTODO ITERATIVO DE GAUSS-SEIDEL PARA O CASO DE UM PROBLEMA "PENTADIAGONAL"

1. Introdução

Em Matemática, um sistema de equações lineares é um conjunto finito de equações de grau 1 aplicadas num mesmo conjunto, igualmente finito, de variáveis. Uma solução para um sistema linear é uma atribuição de valores às incógnitas que satisfazem simultaneamente todas as equações do sistema.

Existem inúmeros métodos para resolução de sistemas lineares. Dois deles serão tratados neste trabalho: o método de "Eliminação de Gauss" e o método iterativo de "Gauss-Seidel". O objetivo principal é comparar as soluções e o esforço computacional de cada método.

Um tipo particular de sistema linear será usado: sistemas cuja matriz de coeficientes é pentadiagonal. Matrizes pentadiagonais são aquelas cujos elementos se concentram em uma faixa central da matriz, tendo elementos não nulos nas cinco diagonais centrais e todo o restante nulo.

Levando em consideração essa particularidade, serão apresentadas as implementações dos dois métodos já mencionados, usando a linguagem python. O programa está dividido em três partes: o módulo principal (main.py), responsável por realizar a leitura dos dados do problema, invocar os métodos de resolução e calcular os erros; o módulo referente ao método "Eliminação de Gauss" (gauss.py), responsável pela resolução do sistema por meio do método de mesmo nome; e o módulo referente ao método iterativo de “Gauss-Seidel” (seidel.py), responsável pela resolução do sistema via método de Gauss-Seidel.

2. Método numérico 1: Eliminação de Gauss

A implementação deste método foi fortemente baseada na implementação fornecida nas aulas de laboratório (elimGauss_Com_pivot.c) e está dividida basicamente em duas partes: a triangularização e a resolução.

A parte de triangularização consiste em transformar o sistema fornecido em um sistema que possui uma matriz de coeficientes triangular superior. A maneira de se conseguir isso é percorrendo cada elemento da diagonal principal e zerando todos os elementos que estão abaixo deste na coluna via aplicação de operações matriciais elementares nas linhas correspondentes. Antes, porém, é definido o elemento pivô, ou seja, o elemento que permanecerá na diagonal principal. Este elemento deve ser o maior em valor absoluto dentre o que já está na diagonal mais os que serão zerados. Considerando o caso particular de um sistema pentadiagonal, os loops aninhados para acessar somente os elementos não nulos ficam assim:

```
# Para cada elemento na diagonal principal
for k in range(0, n):
   [...]

   # Define o intervalo que será percorrido abaixo do elemento.
   # Se for o último ou o penúltimo elemento, o intervalo será
   # [k+1, n) para evitar "List index out of range".
   # Se for qualquer outro elemento da diagonal principal o
   # intervalo será [k+1, k+3) para não acessar elementos nulos
   intervalo = range(k+1, n) if (k == n-1 or k == n-2) else
range(k+1, k+3)
```

Exemplo: considerando um sistema com n=20, os elementos acessados abaixo de A[2][2] (k=2) serão A[**2+1**][2] e A[**2+2**][2], pois a função range(k+1, k+3) retorna os valores: **k+1** e **k+2**. Considerando o mesmo sistema, o elemento percorrido abaixo de A[18][18] (k=18) é A[**18+1**][18], pois a função range(k+1, n) retorna o valor **k+1**. Isso acontece porque 18 == 20-2, uma das condições que define qual range será aplicado.

Após a triangularização, a resolução do sistema é feita de forma retroativa. Enfatizando os intervalos percorridos, temos:

```
# Resolve as demais linhas do sistema
for i in range(n-2, -1, -1):

  [...]

   # Define o intervalo que será percorrido à direita da diagonal.
   # Se for o penúltimo, antepenúltimo ou um antes deste, o
   # intervalo será [i+1, n), caso contrário será [i+1, i+5)
   intervalo2 = range(i+1, n) if (i == n-2 or i == n-3 or i == n-4)
else range(i+1, i+5)
```

O intervalo percorrido à direita do elemento da diagonal foi definido como sendo de tamanho máximo igual a quatro elementos. Essa estratégia foi utilizada pois verificou-se que no pior caso, após o pivoteamento, a linha pivô irá abrigar 4 elementos após o elemento da diagonal, como exemplificado a seguir.

d_A	d_B	d_P	d_C	d_D	$\mathbf{d_C}$	$\mathbf{d_D}$
	d_A	d_B	d_P	d_C	d_D	
		d_A	d_B	d_P	$\mathbf{d_C}$	$\mathbf{d_D}$

As últimas linhas receberão tratamento especial: por não conter 4 posições a partir da diagonal, o limite superior do intervalo é definido como o fim da matriz, ou seja, o intervalo é dado por range(i+1, n). Isso evita o erro “List index out of range”. Para as demais linhas, o intervalo considera 4 elementos: range(i+1, i+5), que retornará os índices **i+1**, **i+2**, **i+3** e **i+4**.

3. Método numérico 2: Gauss-Seidel

Para realizar a implementação do método de Gauss-Seidel foi necessário analisar a matriz pentadiagonal. A estratégia adotada foi realizar a solução do sistema em três partes. Analisando a matriz, foi observado que existe um padrão nas duas primeiras linhas, nas linhas do "meio" e nas duas últimas linhas.

Na implementação desse método numérico, três funções foram criadas:

- solucionarDuasPrimeirasLinhas
- solucionarCentro
- solucionarDuasUltimasLinhas

As três funções realizam um comportamento semelhante, sofrendo diferenças apenas nos parâmetros que foram fornecidos e na manipulação de listas internas. Basicamente, a matriz é percorrida armazenando os elementos não nulos em uma lista auxiliar sem o elemento da diagonal.

A lista auxiliar é percorrida realizando as operações necessárias e logo após o vetor X é atualizado com a solução encontrada.

Para obter o número de iterações necessárias para encontrar a solução, foi utilizado uma estrutura de repetição que foi executada até a diferença relativa ser menor do que a tolerância fornecida.

4. Resultados

4.1 Execução 1

d_A	d_B	d_P	d_C	d_D
2.0	1.0	2.2	0.1	0.2

n	Operações		Erro máximo		Erro médio	
	Eliminação de Gauss	Gauss-Seidel	Eliminação de Gauss	Gauss-Seidel	Eliminação de Gauss	Gauss-Seidel
20	549	3948	4.44e-16	2.09e-11	2.66e-16	5.76e-12

Vale ressaltar que o método de Gauss-Seidel precisou de 21 iterações para ser concluído.

Matriz A original:

2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20

Vetor b original:

2.50	3.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.50	5.30	5.20

Matriz A após a aplicação do método 1 (Eliminação de Gauss):

2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	2.15	0.01	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	2.01	0.02	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	2.01	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	2.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.06	0.63	0.21	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10	0.20	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	0.61	0.20	0.06	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20	0.10
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	2.20
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.45	-0.97
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.67

Vetor b após a aplicação do método 1 (Eliminação de Gauss):

2.50	2.36	2.23	2.21	2.20	5.50	5.50	5.50	5.50	2.96	5.50	5.50	5.50	5.50	5.50	2.87	5.30	5.20	0.48	-2.67

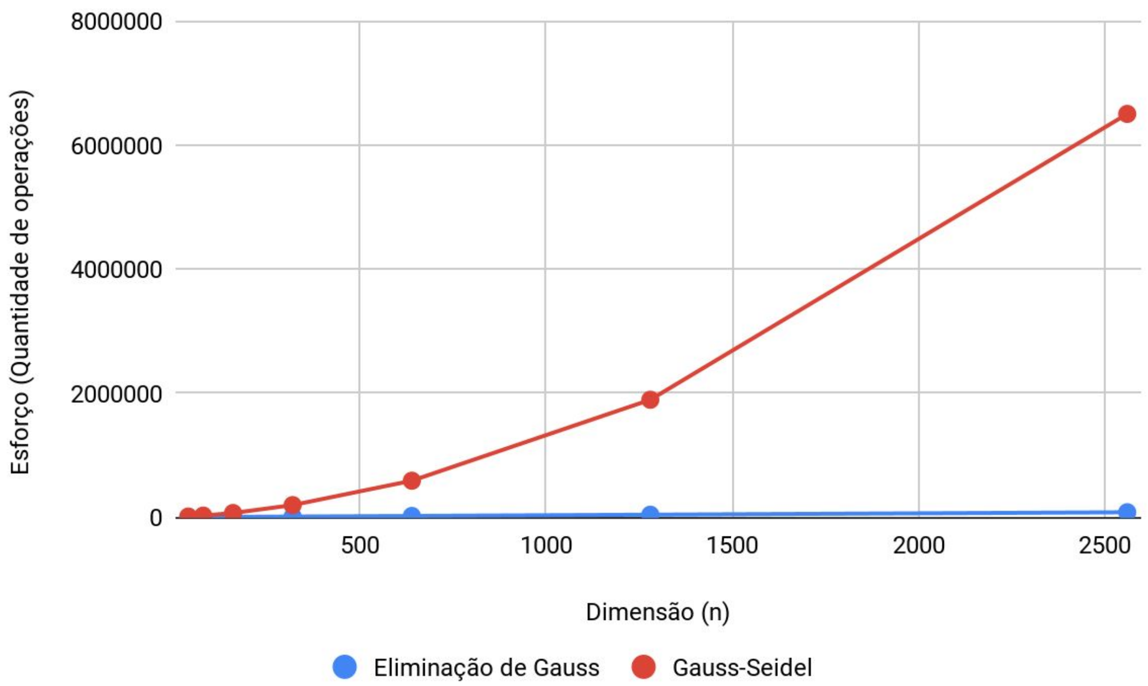
Os elementos pintados na cor cinza são aqueles cujo valor garantidamente é nulo e portanto não são acessados durante a execução do algoritmo do método de “Eliminação de Gauss”.

4.2 Execução 2

d_A	d_B	d_P	d_C	d_D
2.0	1.0	2.2	0.1	0.2

n	**Operações**		**Erro máximo**		**Erro médio**	
	Eliminação de Gauss	**Gauss-Seidel**	**Eliminação de Gauss**	**Gauss-Seidel**	**Eliminação de Gauss**	**Gauss-Seidel**
40	1169	9700	2.44e-15	3.66e-11	9.68e-16	6.13e-12
80	2409	25216	2.44e-15	3.11e-11	9.50e-16	2.91e-12
160	4889	66696	2.99e-15	7.34e-11	1.26e-15	3.99e-12
320	9849	194468	7.77e-15	4.31e-11	3.14e-15	1.50e-12
640	19769	587696	1.37e-14	7.66e-11	6.37e-15	1.77e-12
1280	39609	1892624	1.22e-14	1.35e-10	5.98e-15	2.10e-12
2560	79289	6499352	2.30e-14	1.51e-10	8.63e-15	1.42e-12

Comparação do esforço computacional dos métodos



4.3 Execução 3

d_A	d_B	d_P	d_C	d_D
1.0	1.0	4.5	1.0	1.0

n	**Operações**		**Erro máximo**		**Erro médio**	
	Eliminação de Gauss	**Gauss-Seidel**	**Eliminação de Gauss**	**Gauss-Seidel**	**Eliminação de Gauss**	**Gauss-Seidel**
40	1169	8924	2.22e-16	3.32e-11	8.32e-17	8.54e-12
80	2409	18124	2.22e-16	3.32e-11	6.93e-17	5.10e-12
160	4889	36524	2.22e-16	3.32e-11	6.24e-17	3.39e-12
320	9849	73324	2.22e-16	3.32e-11	5.89e-17	2.53e-12
640	19769	146924	2.22e-16	3.32e-11	5.72e-17	2.11e-12
1280	39609	294124	2.22e-16	3.32e-11	5.63e-17	1.89e-12
2560	79289	588524	2.22e-16	3.32e-11	5.59e-17	1.79e-12

4.4 Execução 4

d_A	d_B	d_P	d_C	d_D
4.0	8.0	10	0.8	0.4

n	**Operações**		**Erro máximo**		**Erro médio**	
	Eliminação de Gauss	**Gauss-Seidel**	**Eliminação de Gauss**	**Gauss-Seidel**	**Eliminação de Gauss**	**Gauss-Seidel**
40	1169	4268	3.33e-16	6.38e-12	1.94e-16	2.14e-12
80	2409	8668	3.33e-16	6.38e-12	2.35e-16	1.09e-12
160	4889	17468	3.33e-16	6.38e-12	2.56e-16	5.53e-13
320	9849	35068	3.33e-16	6.38e-12	2.67e-16	2.83e-13
640	19769	70268	3.33e-16	6.38e-12	2.72e-16	1.47e-13
1280	39609	140668	3.33e-16	6.38e-12	2.74e-16	8.02e-14
2560	79289	281468	3.33e-16	6.38e-12	2.76e-16	4.64e-14

5 Código fonte

5.1 main.py

```
import math
import numpy as np
import gauss
import seidel


# Exibe o erro existente no vetor
# solução x, considerando que a so-
# lução correta é o vetor [1, 1, ... 1]
def calcularErro(x):
   e = np.zeros(len(x), float)
   emax = 0
   esoma = 0
   for i, y in enumerate(x):
       e[i] = math.fabs(y - 1)
       esoma += e[i]
       if e[i] > emax:
           emax = e[i]
   print("Erro máximo: ", emax)
   print("Erro médio: ", esoma/len(x))


# Função principal
def main():

   # Realiza as leituras
   n=int(input("Dimensão do sistema (n): "))
   da=float(input("Valor da diagonal a (da): "))
   db=float(input("Valor da diagonal b (db): "))
   dp=float(input("Valor da diagonal p (dp): "))
   dc=float(input("Valor da diagonal c (dc): "))
   dd=float(input("Valor da diagonal d (dd): "))

   # Cria uma matriz e vetores b e x zerados
   A = np.zeros((n,n), float)
   b = [0.]*n
   x = [0.]*n
```

```python
    # Preenche as cinco diagonais
    for i in range(0,n):
        for j in range(0,n):
            d = i - j
            if d == 2:
                A[i][j] = da
            elif d == 1:
                A[i][j] = db
            elif d == 0:
                A[i][j] = dp
            elif d == -1:
                A[i][j] = dc
            elif d == -2:
                A[i][j] = dd

    # Gera um vetor b tal que a solução seja x=[1.0, 1.0, ... 1.0]
    for i in range(0,n):
        for j in range(0,n):
            b[i] = b[i]+A[i][j]

    print("\n\n\nMÉTODO ELIMINAÇÃO DE GAUSS:")
    Ag = A.copy()
    bg = b.copy()
    xg = x.copy()
    gauss.resolver(Ag, n, bg, xg)
    calcularErro(xg)

    print("\n\n\nMÉTODO GAUSS-SEIDEL:")
    As = A.copy()
    bs = b.copy()
    xs = x.copy()
    seidel.resolver(As, n, bs, xs, dp)
    calcularErro(xs)


# Chama a função principal
main()
```

5.2 gauss.py

```
import math


# Eliminação de Gauss com Pivoteamento
def resolver(A, n, b, x):

    # Guarda a quantidade de operações realizadas
    operacoes = 0

    # Triangularização com pivoteamento
    # Para cada elemento na diagonal principal
    for k in range(0, n):

        # Guarda o maior elemento em módulo e seu índice
        maior = math.fabs(A[k][k])
        imaior = k

        # Define o intervalo que será percorrido abaixo do elemento
        # da diagonal.
        # Se for o último ou o penúltimo elemento, o intervalo será
        # [k+1, n) para evitar "List index out of range".
        # Se for qualquer outro elemento da diagonal principal o
        # intervalo será [k+1, k+3) para não acessar elementos nulos
        intervalo = range(k+1, n) if (k == n-1 or k == n-2) else
range(k+1, k+3)

        # Encontra o maior elemento em valor absoluto e seu indice
        for i in intervalo:
            if (math.fabs(A[i][k]) > maior):
                maior = math.fabs(A[i][k])
                imaior = i

        # Se o elemento da diagonal não for o maior em valor
        # absoluto
        if (imaior != k):

            # Troca a linha k com a linha imaior na matriz A
            for j in range(0, n):
                aux = A[k][j]
```

```
                A[k][j] = A[imaior][j]
                A[imaior][j] = aux

            # Troca o elemento k pelo imaior no vetor b
            baux = b[k]
            b[k] = b[imaior]
            b[imaior] = baux

        # Zera os elementos abaixo do pivo, aplicando operações
        # elementares
        for i in intervalo:

            # Define o valor de m
            m = A[i][k] / A[k][k]
            operacoes += 1

            # Zera o elemento
            A[i][k] = 0

            # Define o intervalo que será percorrido à direita do
            # elemento da diagonal.
            # Se for o penúltimo, antepenúltimo ou um antes deste, o
            # intervalo
            # será [k+1, n) para evitar "List index out of range".
            # Se for qualquer outro elemento da diagonal principal o
            # intervalo
            # será [k+1, k+5) para acessar o mínimo possível de
            # elementos nulos
            intervalo2 = range(k+1, n) if (k == n-2 or k == n-3 or k == n-4) else range(k+1, k+5)

            # Percorre o restante da linha, a partir da diagonal
            for j in intervalo2:

                # Operação elementar na linha i da matriz A
                A[i][j] = A[i][j] - m * A[k][j]
                operacoes += 2

            # Operação elementar no elemento i do vetor b
            b[i] = b[i] - m * b[k]
            operacoes += 2
```

```
    # Resolve a última linha do sistema
    x[n - 1] = b[n - 1] / A[n - 1][n - 1]
    operacoes += 1

    # Resolve as demais linhas do sistema
    for i in range(n - 2, -1, -1):

        soma = b[i]

        # Define o intervalo que será percorrido à direita do
        # elemento da diagonal.
        # Se for o penúltimo, antepenúltimo ou um antes deste, o
        # intervalo
        # será [i+1, n) para evitar "List index out of range".
        # Se for qualquer outro elemento da diagonal principal o
        # intervalo
        # será [i+1, i+5) para acessar o mínimo possível de
        # elementos nulos
        intervalo2 = range(i+1, n) if (i == n-2 or i == n-3 or i ==
n-4) else range(i+1, i+5)

        # Percorre o restante da linha a partir da diagonal
        for j in intervalo2:

            # Acumula o elemento na variavel soma
            soma = soma - A[i][j] * x[j]
            operacoes += 2

        # Divide a soma pelo coeficiente da diagonal
        x[i] = soma / A[i][i]
        operacoes += 1

    # Exibe a quantidade de operações realizadas
    print("Operações realizadas: ", operacoes)
```

5.2 seidel.py

```
from math import fabs

#Armazena a dimensão da matriz
n=0

#Cria uma matriz zerada e vetores zerados
A = []
b = [0.]*n
X = [0.]*n
xAnterior = [0.]*n

#Variáveis para realizar o cálculo da diferença relativa
numeroOperacoesSeidel = 0
tol = 0.0000000001
difRel = 1
difMax = 0
xMax = 0

#Função que soluciona as duas primeiras linhas da matriz
def solucionarDuasPrimeirasLinhas(linhaInicial, linhaFinal,
elementoInicial, elementoFinal, addElemInicial, addElemFinal,
diagonalPrincipal):

    #Variáveis
    limiteDiag = 0 #Variável utilizada antes e depois do elemento da
                   #diagonal
    soma = 0
    posicao = 0
    naoNulos = []
    global numeroOperacoesSeidel
    global difMax
    global xMax

    #Percorre a matriz A utilizando somente elementos não nulos
    for linha in range(linhaInicial, linhaFinal):
        for elemento in range(elementoInicial, elementoFinal):
            #Armazena os elementos em uma lista
            naoNulos.append(A[linha][elemento])
```

```
        #Remove o elemento da diagonal
        linhaSemElemDiag = naoNulos[0:limiteDiag] +
naoNulos[limiteDiag+1:]
        XsemElemDiag = X[0:limiteDiag] + X[limiteDiag+1:]

        #Percorre a linha sem o elemento da diagonal e realiza a
        #soma dos elementos
        for elemento in linhaSemElemDiag:
            soma += elemento*XsemElemDiag[posicao]
            posicao += 1
            numeroOperacoesSeidel = numeroOperacoesSeidel + 2

        #Armazena o resultado no vetor X
        X[limiteDiag] = (b[limiteDiag] - soma) / diagonalPrincipal
        numeroOperacoesSeidel = numeroOperacoesSeidel + 2

        #Calcula a diferença relativa
        if(fabs(X[limiteDiag] - xAnterior[limiteDiag]) > difMax):
            difMax = fabs(X[limiteDiag] - xAnterior[limiteDiag])

        if(fabs(X[limiteDiag]) > xMax):
            xMax = fabs(X[limiteDiag])

        #Atualiza as variáveis
        xAnterior[limiteDiag] = X[limiteDiag]
        limiteDiag += 1
        posicao = 0
        soma = 0
        naoNulos = []
        elementoInicial += addElemInicial
        elementoFinal += addElemFinal

#Função que soluciona a parte central da matriz
def solucionarCentro(linhaInicial, linhaFinal, elementoInicial,
elementoFinal, addElemInicial, addElemFinal, diagonalPrincipal):

    #Variáveis
    xIni = 0
    xFim = 2
    soma = 0
    posicao = 0
```

```python
naoNulos = []
global numeroOperacoesSeidel
global difMax
global xMax

#Percorre a matriz A utilizando somente elementos não nulos
for linha in range(linhaInicial, linhaFinal):
    for elemento in range(elementoInicial, elementoFinal):
        #Armazena os elementos em uma lista
        naoNulos.append(A[linha][elemento])

    #Remove o elemento da diagonal
    linhaSemElemDiag = naoNulos[0:2] + naoNulos[3:]
    XsemElemDiag = X[xIni:xFim] + X[xFim+1:xFim+3]

    #Percorre a linha sem o elemento da diagonal e realiza a
    #soma dos elementos
    for elemento in linhaSemElemDiag:
        soma += elemento*XsemElemDiag[posicao]
        posicao += 1
        numeroOperacoesSeidel = numeroOperacoesSeidel + 2

    #Armazena o resultado no vetor X
    X[xFim] = (b[xFim] - soma) / diagonalPrincipal
    numeroOperacoesSeidel = numeroOperacoesSeidel + 2

    #Calcula a diferença relativa
    if(fabs(X[xFim] - xAnterior[xFim]) > difMax):
        difMax = fabs(X[xFim] - xAnterior[xFim])

    if(fabs(X[xFim]) > xMax):
        xMax = fabs(X[xFim])

    #Atualiza as variáveis
    xAnterior[xFim] = X[xFim]
    xFim += 1
    xIni += 1
    posicao = 0
    soma = 0
    naoNulos = []
    elementoInicial += addElemInicial
```

```python
        elementoFinal += addElemFinal

#Função que soluciona as duas últimas linhas da matriz
def solucionarDuasUltimasLinhas(linhaInicial, linhaFinal,
elementoInicial, elementoFinal, addElemInicial, addElemFinal,
diagonalPrincipal):

    #Variáveis
    xIni = n-4
    xFim = n-2
    soma = 0
    posicao = 0
    naoNulos = []
    global numeroOperacoesSeidel
    global difMax
    global xMax

    #Percorre a matriz A utilizando somente elementos não nulos
    for linha in range(linhaInicial, linhaFinal):
        for elemento in range(elementoInicial, elementoFinal):
            #Armazena os elementos em uma lista
            naoNulos.append(A[linha][elemento])

        #Remove o elemento da diagonal
        linhaSemElemDiag = naoNulos[0:2] + naoNulos[3:]
        XsemElemDiag = X[xIni:xFim] + X[xFim+1:]

        #Percorre a linha sem o elemento da diagonal e realiza a
        #soma dos elementos
        for elemento in linhaSemElemDiag:
            soma += elemento*XsemElemDiag[posicao]
            posicao += 1
            numeroOperacoesSeidel = numeroOperacoesSeidel + 2

        #Armazena o resultado no vetor X
        X[xFim] = (b[xFim] - soma) / diagonalPrincipal
        numeroOperacoesSeidel = numeroOperacoesSeidel + 2

        #Calcula a diferença relativa
        if(fabs(X[xFim] - xAnterior[xFim]) > difMax):
            difMax = fabs(X[xFim] - xAnterior[xFim])
```

```
        if(fabs(X[xFim]) > xMax):
            xMax = fabs(X[xFim])

        #Atualiza as variáveis
        xAnterior[xFim] = X[xFim]
        xFim += 1
        xIni += 1
        posicao = 0
        soma = 0
        naoNulos = []
        elementoInicial += addElemInicial
        elementoFinal += addElemFinal


#Função principal
def resolver(A2, n2, b2, x2, dp):

    global A
    global n
    global b
    global X
    global xAnterior

    A = A2
    n = n2
    b = b2
    X = x2

    # Preenche vetor inicial
    xAnterior = [0.] * n
    for i in range(0, n):
        xAnterior[i] = b[i]/A[i][i]

    #Variáveis
    global difRel
    global difMax
    global xMax
    numeroIteracoesSeidel = 0
```

```python
#Realiza o método de Gauss Seidel
while(difRel > tol):

    #Variáveis
    difMax = 0
    xMax = 0
    numeroIteracoesSeidel += 1

    #Realiza o método de GaussSeidel
    #Parâmetros: linhaInicial, linhaFinal, elementoInicial,
    #elementoFinal, addElemInicial, addElemFinal,
    #diagonalPrincipal
    solucionarDuasPrimeirasLinhas(0,2,0,3,0,1,dp)
    solucionarCentro(2,n-2,0,5,1,1,dp)
    solucionarDuasUltimasLinhas(n-2,n,n-4,n,1,0,dp)

    #Calcula a diferença relativa
    difRel = difMax / xMax

#Imprime o total de iterações
print('Total de iterações: {}'.format(numeroIteracoesSeidel))
#Imprime o total de operações realizadas
print('Operações realizadas: {}'.format(numeroOperacoesSeidel))
```