数值实验报告 I

实验名称	计算方法上机实践				实验时间		2025 年 4 月 11 日
姓名	秦浩政 郭凯平 刘佳鑫 刘桂凡	班级	数据 2301	学号	2306030214 2306020510 2309050116 2309050117	成绩	

一、实验目的，内容

实验 7.1：Jacobi 迭代法与 G-S 迭代法收敛性及收敛速度比较

实验目的：

本次实验旨在加强对于 Jacobi 法以及 G-S 法的理解与运用。具体目标如下：

1.学习如何使用 Jacobi 迭代法以及 G-S 法迭代法求解线性方程组，结合 numpy 数组熟练掌握翻译题目的能力.

2.学会计算迭代矩阵的谱半径来判断收敛性，熟练掌握不同迭代法的向量形式以简化计算。

4.通过实验结果的分析，评估两种迭代法的性能差异，包括收敛速度、计算精度等，为实际应用中选择合适的数值方法提供依据。

实验内容：

1. 算法实现

• 编写 Python 代码实现 Jacobi 迭代法求解线性方程组。具体步骤包括初始化迭代向量、将已知矩阵化为对角矩阵,上三角矩阵以及下三角矩阵,组合成不同迭代公式所需要的迭代矩阵,计算并保存不同迭代次数时的结果。

• 对每个线性方程组给出三种情况：迭代不收敛，迭代收敛但在给定迭代次数中无法达到误差容忍的要求，迭代收敛并成功计算出符合要求的解。

2. 测试用例验证

• 对给定的四个线性方程组分别使用 Jacobi 法与 G-S 法并记录迭代次数，收敛性并比较。

• 对成功收敛的方程组给出近似解并比较精度。

3. 结果分析

• 对同时符合两种迭代方式的线性方程组，G-S 法的熟练速度明显快于 Jacobi 法。

二、算法描述

实验 7.1：Jacobi 迭代法与 G-S 迭代法

输入

• 系数矩阵 A（n×n 矩阵）
• 右端向量 b（长度为 n 的向量）
• 系数矩阵维度
• 最大迭代次数 max_iter（默认值为 30）
• 误差容忍度 Tol（默认值为 10^{-8}）

输出

• 收敛性判断(print 输出)
• 线性方程组的解向量 x
• 迭代次数
• 返回值表示是否成功计算出方程的近似解(True,False)

步骤

• 初始化迭代向量 x_before 为零向量，初始化确定对角矩阵,上三角矩阵以及下三角矩阵
• 对系数矩阵进行遍历,确定对角矩阵,上三角矩阵以及下三角矩阵
• 分别计算 Jacobi 迭代法以及 G-S 迭代法所需的迭代矩阵
• 计算两迭代矩阵的特征值，取其绝对值的最大值作为谱半径,与 1 比较判断收敛性
• 若收敛则开始迭代求解，进行 max_iter 次迭代,根据迭代公式计算 x 的新值
• 计算相邻两次迭代结果的误差$\|x_after - x_before\|_\infty$，若误差小于容差 Tol，则返回近似解
• 将x_1的值赋给 x，继续下一次迭代。
• 若不收敛则返回 False 并输出相应信息

三、程序代码

实验 7.1：Jacobi 迭代法与 G-S 迭代法收敛性及收敛速度比较

```python
def Jacobi(A, b, n, max_iter, Tol):
    D = np.zeros((n, n))
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for i in range(n):  # 计算对角线,下三角以及上三角矩阵
        for j in range(n):
            if i == j:
                D[i, j] = A[i, j]
            elif i > j:
                L[i, j] = A[i, j]
            else:
                U[i, j] = A[i, j]

    D_inv = np.linalg.inv(D)
    M = np.dot(-D_inv, L + U)  # 迭代矩阵
    g = np.dot(D_inv, b)

    eigvals = np.linalg.eigvals(M)
    eigvals = max(abs(eigvals))
    if eigvals<1:
        print("此方程组在Jacobi法下收敛")
        x_before = np.zeros((n, 1))
        for _ in range(max_iter):
            x_after = np.dot(M, x_before) + g
            if np.linalg.norm(x_after - x_before, ord=np.inf) < Tol:
                print(f"迭代{_}次后收敛, 方程的解为")
                print(f"{x_after}")
                return True
            x_before = x_after
        print(f"方程未在{max_iter}次内收敛, 需要提高迭代次数")
        return False
    else:
        print("方程组在Jacobi下不收敛")
        return False
```

```python
def GaussSeidel(A, b, n, max_iter, Tol):
    D = np.zeros((n, n))
    L = np.zeros((n, n))
    U = np.zeros((n, n))
    for i in range(n):  # 计算对角线, 下三角以及上三角矩阵
        for j in range(n):
            if i == j:
                D[i, j] = A[i, j]
            elif i > j:
                L[i, j] = A[i, j]
            else:
                U[i, j] = A[i, j]
    LD_inv = np.linalg.inv(L + D)
    M = -np.dot(LD_inv, U)
    g = np.dot(LD_inv, b)
    eigvals = np.linalg.eigvals(M)
    eigvals = max(abs(eigvals))
    if eigvals < 1:
        print("此方程组在高斯-赛德尔法下收敛")
        x_before = np.zeros((n, 1))  # 初始化解为零向量
        for _ in range(max_iter):
            for i in range(n):
                x_after = np.dot(M, x_before) + g
            if np.linalg.norm(x_after-x_before, ord=np.inf) < Tol:
                print(f"迭代 {_} 次后收敛, 方程的解为:")
                print(x_after)
                return True
            x_before = x_after  # 更新 x 为新解

        print(f"方程未在 {max_iter} 次内收敛, 需要提高迭代次数")
        return False
    else:
        print("方程组在高斯-赛德尔法下不收敛")
        return False
```

四. 数值结果

测试用例

```
matrixA = np.array([[10, 3, 1], [2, -11, 3], [1, 3, 12]])
matrixB = np.array([[1,0.8,0.8], [0.8,1,0.8], [0.8,0.8,1]])
matrixC = np.array([[1,2,-2], [1,1,1], [2,2,1]])
matrixD = np.array([[2,1,1], [1,2,-1], [-1,-1,2]])
Jacobi(matrixA, np.array([[2], [-5], [4]]), n: 3, max_iter: 30, Tol: 1e-8)
GaussSeidel(matrixA, np.array([[2], [-5], [4]]), n: 3, max_iter: 30, Tol: 1e-8)
Jacobi(matrixB, np.array([[1], [2], [3]]), n: 3, max_iter: 30, Tol: 1e-8)
GaussSeidel(matrixB, np.array([[1], [2], [3]]), n: 3, max_iter: 30, Tol: 1e-8)
Jacobi(matrixC, np.array([[4], [-2], [1]]), n: 3, max_iter: 30, Tol: 1e-8)
GaussSeidel(matrixC, np.array([[4], [-2], [1]]), n: 3, max_iter: 30, Tol: 1e-8)
Jacobi(matrixD, np.array([[-6], [7], [2]]), n: 3, max_iter: 30, Tol: 1e-8)
GaussSeidel(matrixD, np.array([[-6], [7], [2]]), n: 3, max_iter: 30, Tol: 1e-8)
```

```
此方程组在Jacobi法下收敛
迭代18次后收敛，方程的解为
[[0.02541209]
 [0.51442308]
 [0.20260989]]
此方程组在高斯-赛德尔法下收敛
迭代 8 次后收敛，方程的解为:
[[0.02541209]
 [0.51442308]
 [0.20260989]]
方程组在Jacobi下不收敛
此方程组在高斯-赛德尔法下收敛
方程未在 30 次内收敛，需要提高迭代次数
此方程组在Jacobi法下收敛
迭代3次后收敛，方程的解为
[[12.]
 [-9.]
 [-5.]]
方程组在高斯-赛德尔法下不收敛
此方程组在Jacobi法下收敛
迭代29次后收敛，方程的解为
[[-7.49999999]
 [ 7.83333332]
 [ 1.16666667]]
此方程组在高斯-赛德尔法下收敛
迭代 21 次后收敛，方程的解为:
[[-7.5       ]
 [ 7.83333333]
 [ 1.16666667]]
```

五. 计算结果分析

Jacobi 迭代法与 G-S 迭代法收敛性及收敛速度比较

测试用例结果

对于给定的测试用例 1,

此方程组在 Jacobi 法下收敛
迭代 18 次后收敛，方程的解为
[[0.02541209]
 [0.51442308]
 [0.20260989]]

此方程组在高斯-赛德尔法下收敛
迭代 8 次后收敛，方程的解为:
[[0.02541209]
 [0.51442308]
 [0.20260989]]
 运行代码后可以明显观察到二者都收敛,G-S 法所需迭代次数小于 Jacobi 法所需迭代次数 ；
对于测试用例 2,
方程组在 Jacobi 下不收敛

此方程组在高斯-赛德尔法下收敛
方程未在 30 次内收敛，需要提高迭代次数
可知方程组仅满足 G-S 法的迭代条件,然而迭代次数超过三十次,说明对此方程组而言 G-S 具有更强的收敛性

- **收敛性分析**：总体而言 Jacobi 法与 G-S 法均存在近自己收敛而对方不收敛的情况，然而结合总体观察,G-S 法的收敛性稍强于 Jacobi 法
- 收敛速度分析：在二者均收敛的情况下，G-S 法收敛速度大于 Jacobi 迭代法，且有明显优势
- **解的精度**：G-S 法在同等迭代次数的前提下一定高于 Jacobi 法。

总结:
G-S 法与 Jacobi 法相比具有更稳定的收敛性，更快的收敛速度，同时也对数据具有更高的利用率
六. 计算中出现的问题，解决方法及体会

问题：
1. 进行 G-S 法的编写时一开始根据课件迭代矩阵写的是(L-D)^-1*U，然而计算结果总是发生错误,
2. 解决方法：
查阅资料下更改为(L+D)^-1*(-U)后解决问题.
体会：
通过本次实验，我们小组加深了对于 Jacobi 法以及 G-S 法的掌握程度以及其收敛性,收敛速度的认识，熟练掌握了运用 numpy 数组进行矩阵运算的能力，并熟悉掌握了不同迭代方法的向量形式以简化程序.

教师评语	老师不好意思,一开始做了一半后来那一半的doc弄丢了,所以7.2的部分还在下面 指导教师：　　年　月　日

数值实验报告 I

实验名称	计算方法上机实践				实验时间		2025 年 4 月 11 日
姓名	秦浩政 郭凯平 刘佳鑫 刘桂凡	班级	数据 2301	学号	2306030214 2306020510 2309050116 2309050117	成绩	

一、实验目的，内容

实验 7.2：SOR 迭代法

实验目的：

本次实验的主要目标是深入理解和掌握 SOR 迭代法在求解线性方程组中的原理和实现过程。具体目标如下：

学习如何应用 **SOR 迭代法** 求解线性方程组，并结合理论知识，进一步理解迭代法的**收敛性和松弛因子**在收敛过程中的作用。

掌握通过计算 **Jacobi 迭代法** 的迭代矩阵的 **谱半径**，来确定 **SOR 迭代法** 的最佳松弛因子，理解谱半径与迭代法收敛性之间的关系。

通过实验结果的分析，评估 **SOR 迭代法** 的性能，重点分析 **收敛速度**、**计算精度** 等指标，为实际应用中选择合适的数值方法提供依据。

实验内容：

算法实现

编写 Python 代码实现 **SOR 迭代法** 来求解线性方程组。主要步骤包括：初始化迭代向量，计算 **Jacobi 迭代法** 的迭代矩阵及其谱半径，确定 **最佳松弛因子**（如果 **Jacobi 迭代法** 收敛），并根据 **SOR 迭代公式** 进行迭代求解。

实现迭代的终止条件：当相邻两次迭代结果的误差小于给定容差（Tol）时，认为迭代收敛，并输出最终解。

测试用例验证

选择一个具体的线性方程组作为测试用例，给定系数矩阵 A 和右端项向量 b，调用实现的 **SOR 迭代法** 函数进行求解。

记录并输出 **最佳松弛因子**（如果存在）、迭代是否收敛的标志（flag）以及最终的解向量。

结果分析

分析不同 **松弛因子** 对迭代收敛速度的影响。通过调整松弛因子的值（在合理范围内），观察迭代次数的变化，探讨**最佳松弛因子** 在加速收敛过程中的作用。

比较 **SOR 迭代法** 与其他迭代法（如 **Jacobi 迭代法** 和 G-S 迭代法）的收敛性能，分析 **SOR 迭代法** 相对于其他方法的优势和局限性。

二、算法描述

实验 7.2：SOR 迭代法

输入

- 系数矩阵 A
- 右端向量 b
- 系数矩阵维度 n
- 最大迭代次数 max_iter（默认值为 30）

- 误差容忍度 Tol（默认值为 10^{-8}）

输出

- 最佳松弛因子 w（若 Jacobi 迭代法收敛）
- 线性方程组的解向量 x
- 迭代是否收敛的标志 flag

步骤

- 初始化迭代向量x和x_1为零向量，令 flag 为 False
- 迭代系数矩阵 A，计算对角线矩阵，上三角矩阵以及下三角矩阵
- 计算 Jacobi 迭代矩阵的特征值，取其绝对值的最大值作为谱半径。
- 若$\rho < 1$，则根据公式$w = \frac{2}{1+\sqrt{1-\rho^2}}$计算最佳松弛因子 w；否则 print 相应信息，并将 w 设为 None。
- 进行 max_iter 次迭代：
- 对于每个未知数 I，根据 SOR 迭代公式更新x_1中的元素。
- 计算相邻两次迭代结果的误差$\|x_1 - x\|_\infty$，若误差小于容差 Tol，则将 flag 设为 True，并返回x_1和 flag。
- 将x_1的值赋给 x，继续下一次迭代。

4. 输出结果

- 若 w 不为 None，输出最佳松弛因子 w、收敛标志 flag 和最终解向量 x。

三、程序代码

实验 7.2：SOR 迭代法

```
def sor(A, b, n, max_iter=30, tol=1e-8):
    x = np.zeros(n)  # 初始解 x
    x_1 = np.zeros(n)  # 上一次的解向量
    flag = False
    D = np.zeros((n, n))
    L = np.zeros((n, n))
    U = np.zeros((n, n))
    for i in range(n):  # 计算对角线，下三角以及上三角矩阵
        for j in range(n):
            if i == j:
                D[i, j] = A[i, j]
            elif i > j:
                L[i, j] = A[i, j]
            else:
                U[i, j] = A[i, j]
    LD_inv = np.linalg.inv(L + D)
    M = -np.dot(LD_inv, U)
    g = np.dot(LD_inv, b)
    eigvals = np.linalg.eigvals(M)
    eigvals = max(abs(eigvals))
    if eigvals < 1:
        omega = 2 / (1 + np.sqrt(1 - eigvals ** 2))
    else:
        print("Jacobi 迭代法不收敛，无法计算最佳松弛因子。")
        omega = None
```

```
    if omega is not None:
        for k in range(max_iter):
            for i in range(n):
                s1 = np.dot(A[i, :i], x[:i])  # 左侧部分的和
                s2 = np.dot(A[i, i + 1:], x_1[i + 1:])  # 右侧部分的和
                x_1[i] = (1 - omega) * x[i] + (omega / A[i, i]) * (b[i] - s1 - s2)
            if np.linalg.norm(x_1 - x, ord=np.inf) < tol:
                flag = True
                break  # 收敛时退出迭代
            x = x_1.copy()
        return omega, x_1, flag
    return omega, x_1, flag
# Test
A = np.array( object: [[4, 3, 1],
              [3, 4, -1],
              [0, -1, 4]], dtype=float)
b = np.array( object: [1, -5, 3], dtype=float)
n = 3

omega, x, flag = sor(A, b, n)

if omega is not None:
    print(f"最佳松弛因子: {omega}")
    print(f"flag = {flag}")
    print(f"解为: {x}")
```

四. 数值结果

测试用例

```
# Test
A = np.array( object: [[4, 3, 1],
              [3, 4, -1],
              [0, -1, 4]], dtype=float)
b = np.array( object: [1, -5, 3], dtype=float)
n = 3

omega, x, flag = sor(A, b, n)
```

```
/Users/horatius/Desktop/experiment/PycharmProjec
最佳松弛因子: 1.1489864564740064
flag = False
解为: [ 3.17635015 -2.96078923 -0.23305232]

Process finished with exit code 0
```

五. 计算结果分析

实验 7.2：SOR 迭代法 实验

测试用例结果

对于给定的测试用例 1,

最佳松弛因子: 1.1489864564740064
flag = False
解为: [3.17635015 -2.96078923 -0.23305232]

- **收敛性分析**：有题可知成功收敛，在引入最佳松弛因子的情况下得到了良好的收敛性
- 收敛速度分析：在均收敛的情况下， SOR 法的收敛速度大于 G-S 法收敛速度大于 Jacobi 迭代法，且有明显优势

计算中出现的问题，解决方法及体会

问题：

1. 计算效率问题：在处理大规模线性方程组时，代码中的嵌套循环会导致计算量过大，迭代时间过长。

2. 解决方法：

充分利用 python 库多的优点，使用向量化计算(像在 7.1 中使用的一样)，简化程序

体会：

通过本次实验，我们小组加深了对于寻找最佳松弛变量，计算 Jacobi 迭代法的谱半径来帮助运算 SOR 迭代法等操作的熟练度，更深刻的理解了 SOR 法的思想内涵以及优缺点，为不同场景下灵活使用不同方法提供了帮助.

教师评语	指导教师： 年 月 日