**成 绩**

**《数值分析》课程**

**设计报告**

**专业 数学与应用数学**

**学号 2021213521**

**姓名 周伟杰**

**2023 年 12 月 19 日**

**实验1 求线性方程组的解**

**题目1**

**1 问题重述**

用高斯消去法求下列线性方程组的解：



**2 算法介绍**

定义了一个名为 gaussian\_elimination 的函数，它接受两个参数：A（系数矩阵）和b（常数项向量）。然后将A，b转换为浮点数类型，再将A，合并为增广矩阵，统计行数和列数。

接着用循环将当前行的对角元素缩放为1。然后，使用这一行将下面所有行的对应列元素消为0。循环最后可以得到一个变量的解，最后在用这个解回代到上面得到的矩阵，依次得到每一个变量的解。

**3 实验结果及分析**

解为: [-1. 2. 0. 1.]

**4 附录**

import numpy as np

def gaussian\_elimination(A, b):

# 将输入转换为浮点数类型

A = A.astype(float)

b = b.astype(float)

# 合并增广矩阵

augmented\_matrix = np.column\_stack((A, b))

# 行数和列数

rows, cols = augmented\_matrix.shape

# 消去过程

for i in range(rows):

# 将当前列的对角元素缩放为1

current\_row = augmented\_matrix[i, :]

augmented\_matrix[i, :] = current\_row / current\_row[i]

# 将其他行的当前列元素消为0

for j in range(i + 1, rows):

factor = augmented\_matrix[j, i] / augmented\_matrix[i, i]

augmented\_matrix[j, :] -= factor \* augmented\_matrix[i, :]

# 回代过程

x = np.zeros(rows)

for i in range(rows - 1, -1, -1):

x[i] = augmented\_matrix[i, -1] - np.dot(augmented\_matrix[i, i+1:cols-1], x[i+1:cols-1])

return x

A = np.array([[2, 1, -1],

[-3, -1, 2],

[-2, 1, 2]])

b = np.array([8, -11, -3])

result = gaussian\_elimination(A, b)

print("解为:", result)

**题目2**

**1 问题重述**

分别用列主元高斯消去法和全主元高斯消去法求下列方程组的解：



**2 算法介绍**

列主元高斯消去法通过在每一步选择最大的主元，来提高算法对于数值计算的稳定性。其余均同高斯消去法相同。

而全主元高斯消去法在每一步消元过程中，不仅在当前列，而且在整个剩余子矩阵中找到绝对值最大的元素，并通过行列交换将其置于当前对角线位置。从而提高精度。

**3 实验结果及分析**

列主元高斯消去法：[1.1875 1.8125 0.875 ]

全主元高斯消去法：[0.875 1.8125 1.1875]

**4 附录**

**列主元高斯消去法：**

import numpy as np  
  
def gaussian\_elimination\_with\_pivoting(A, b):  
 *# 将输入转换为浮点数类型* A = A.astype(float)  
 b = b.astype(float)  
  
 *# 合并增广矩阵* augmented\_matrix = np.column\_stack((A, b))  
  
 *# 行数和列数* rows, cols = augmented\_matrix.shape  
  
 *# 消去过程* for i in range(rows):  
 *# 找到当前列绝对值最大的行* max\_row\_index = np.argmax(np.abs(augmented\_matrix[i:rows, i])) + i  
 *# 将最大行交换到当前行* augmented\_matrix[[i, max\_row\_index]] = augmented\_matrix[[max\_row\_index, i]]  
  
 *# 将当前列的对角元素缩放为1* current\_row = augmented\_matrix[i, :]  
 augmented\_matrix[i, :] = current\_row / current\_row[i]  
  
 *# 将其他行的当前列元素消为0* for j in range(i + 1, rows):  
 factor = augmented\_matrix[j, i] / augmented\_matrix[i, i]  
 augmented\_matrix[j, :] -= factor \* augmented\_matrix[i, :]  
  
 *# 回代过程* x = np.zeros(rows)  
 for i in range(rows - 1, -1, -1):  
 x[i] = augmented\_matrix[i, -1] - np.dot(augmented\_matrix[i, i+1:cols-1], x[i+1:cols-1])  
  
 return xA = np.array([[1, -1, 3],  
 [3, -3, 1],  
 [1, 1, 0]])  
  
b = np.array([2, -1, 3])  
  
result = gaussian\_elimination\_with\_pivoting(A, b)  
print(result)

**全主元高斯消去法：**

import numpy as np  
def full\_pivot\_gaussian\_elimination(A, b):  
 A = A.astype(float)  
 b = b.astype(float)  
 n = len(b)  
 x = np.zeros(n)  
 pivots = np.arange(n)  
  
 *# 将A和b合并为增广矩阵* aug\_matrix = np.column\_stack((A, b))  
  
 *# 消去过程* for k in range(n):  
 *# 查找当前子矩阵中绝对值最大的元素* i\_max, j\_max = np.unravel\_index(np.abs(aug\_matrix[k:n, k:n]).argmax(), aug\_matrix[k:n, k:n].shape)  
 i\_max += k  
 j\_max += k  
  
 *# 交换行和列* aug\_matrix[[k, i\_max]] = aug\_matrix[[i\_max, k]]  
 aug\_matrix[:, [k, j\_max]] = aug\_matrix[:, [j\_max, k]]  
  
 *# 记录列交换* pivots[[k, j\_max]] = pivots[[j\_max, k]]  
  
 *# 对角线元素不能为0* if aug\_matrix[k, k] == 0:  
 raise ValueError("Matrix is singular.")  
  
 *# 消去* for i in range(k + 1, n):  
 factor = aug\_matrix[i, k] / aug\_matrix[k, k]  
 aug\_matrix[i, k:] -= factor \* aug\_matrix[k, k:]  
  
 *# 回代过程* for i in range(n - 1, -1, -1):  
 x[i] = aug\_matrix[i, n] / aug\_matrix[i, i]  
 for k in range(i - 1, -1, -1):  
 aug\_matrix[k, n] -= aug\_matrix[k, i] \* x[i]  
  
 *# 重排解向量* for i in range(n):  
 if pivots[i] != i:  
 x[i], x[pivots[i]] = x[pivots[i]], x[i]  
  
 return x  
  
A = np.array([[1, -1, 3],  
 [3, -3, 1],  
 [1, 1, 0]])  
  
b = np.array([2, -1, 3])  
  
  
result = full\_pivot\_gaussian\_elimination(A, b)  
print(result)

**实验2 迭代法求线性方程组的解**

**题目1**

**1 问题重述**

The linear system  given by



has the unique solution . Use Jacobi’s iterative and Gauss-Seidel iterative technique to find approximation  to  starting with until

.

**2 算法介绍**

**Jacobi’s iterative：**

首先定义一个convert\_arrays函数，把转换成的形式，即输入

增广矩阵matrix，会将矩阵改为，然后定义函数multiply\_matrices，此函数接受两个矩阵的输入，即和(i为迭代次数)，在函数内部，首先定义一个行数同，1列的0矩阵temp，然后用一个嵌套循环为temp赋值，使temp等于，再修改使得，即下一次迭代的解。最后定义main函数，首先输入要求的线性方程组的增广矩阵，然后调用convert\_arrays函数，将矩阵修改后，创建一个0矩阵用来储存，然后调用while循环重复调用multiply\_matrices函数，不断为赋值，最后zai2精度满足要求时停止循环并输出结果。

**Gauss-Seidel iterative：**

过程与Jacobi’s iterative相同，只需要修改函数multiply\_matrices，在修改使得时，将这一步提前到上面嵌套循环的第一重循环里即可。

**3 实验结果及分析**

Jacobi’s iterative：[ 0.99967415 2.00044767 -1.00036916 1.00061919]

Gauss-Seidel iterative ：[ 1.00009128 2.00002134 -1.00003115 0.9999881 ]

**4 附录**

**Jacobi’s iterative：**

import numpy as np  
  
def convert\_arrays(matrix):  
 for i in range(len(matrix)):  
 for j in range(len(matrix[0])):  
 if i != j and j != len(matrix[0]) - 1:  
 matrix[i][j] = matrix[i][j] / (- matrix[i][i])  
 elif j == len(matrix[0]) - 1:  
 matrix[i][j] = matrix[i][j] / matrix[i][i]  
 matrix[i][i] = 0  
def multiply\_matrices(matrix1, matrix2):  
 temp = np.zeros((len(matrix2), 1))  
 for i in range(len(matrix2)):  
 for j in range(len(matrix2[0]) - 1):  
 temp[i] += matrix1[j] \* matrix2[i][j]  
 for i in range(len(matrix2)):  
 matrix1[i] = temp[i] + matrix2[i][len(matrix2[0]) - 1]  
  
def main():  
 my\_matrix = np.array([[10, -1, 2, 0, 6], [-1, 11, -1, 3, 25], [2, -1, 10, -1, -11], [0, 3, -1, 8, 15]], dtype = float)  
 convert\_arrays(my\_matrix)  
 x = np.zeros((15, len(my\_matrix)))  
 i = 0  
 while True:  
 if i == 0:  
 multiply\_matrices(x[i], my\_matrix)  
 else:  
 x[i] = x[i - 1]  
 multiply\_matrices(x[i], my\_matrix)  
 if (max(np.abs(x[i] - x[i - 1]))) / (np.sqrt(np.sum(x[i]\*\*2))) <= 10\*\*(-3):  
 break  
 i = i + 1  
 print(x[i])main()

**Gauss-Seidel iterative：**

import numpy as np  
  
def convert\_arrays(matrix):  
 for i in range(len(matrix)):  
 for j in range(len(matrix[0])):  
 if i != j and j != len(matrix[0]) - 1:  
 matrix[i][j] = matrix[i][j] / (- matrix[i][i])  
 elif j == len(matrix[0]) - 1:  
 matrix[i][j] = matrix[i][j] / matrix[i][i]  
 matrix[i][i] = 0  
def multiply\_matrices(matrix1, matrix2):  
 temp = np.zeros((len(matrix2), 1))  
 for i in range(len(matrix2)):  
 for j in range(len(matrix2[0]) - 1):  
 temp[i] += matrix1[j] \* matrix2[i][j]  
 matrix1[i] = temp[i] + matrix2[i][len(matrix2[0]) - 1]  
  
def main():  
 my\_matrix = np.array([[10, -1, 2, 0, 6], [-1, 11, -1, 3, 25], [2, -1, 10, -1, -11], [0, 3, -1, 8, 15]], dtype = float)  
 convert\_arrays(my\_matrix)  
 x = np.zeros((15, len(my\_matrix)))  
 i = 0  
 while True:  
 if i == 0:  
 multiply\_matrices(x[i], my\_matrix)  
 else:  
 x[i] = x[i - 1]  
 multiply\_matrices(x[i], my\_matrix)  
 if (max(np.abs(x[i] - x[i - 1]))) / (np.sqrt(np.sum(x[i]\*\*2))) <= 10\*\*(-3):  
 break  
 i = i + 1  
 print(x[i])main()

**题目2**

**1 问题重述**

The linear system  given by



has the solution .Compare the iterations from the Gauss-Seidel method and the SOR method with  using  for both methods.

**2 算法介绍**

[包括:算法介绍（可以有流程图），收敛性、误差分析、参考文献等]

宋体，五号，单倍行距

**3 实验结果及分析**

[包括:必要的结果截图，对结果的说明、分析、讨论等]

宋体，五号，单倍行距

**4 附录**

[包括: 算法实现的源代码、全部结果等]

**实验1 例如,用二分法求非线性方程的根**

**1 问题重述**

[包括问题背景、题目要求等]

宋体，五号，单倍行距

**2 算法介绍**

[包括:算法介绍（可以有流程图），收敛性、误差分析、参考文献等]

宋体，五号，单倍行距

**3 实验结果及分析**

[包括:必要的结果截图，对结果的说明、分析、讨论等]

宋体，五号，单倍行距

**4 附录**

[包括: 算法实现的源代码、全部结果等]

**实验1 例如,用二分法求非线性方程的根**

**1 问题重述**

[包括问题背景、题目要求等]

宋体，五号，单倍行距

**2 算法介绍**

[包括:算法介绍（可以有流程图），收敛性、误差分析、参考文献等]

宋体，五号，单倍行距

**3 实验结果及分析**

[包括:必要的结果截图，对结果的说明、分析、讨论等]

宋体，五号，单倍行距

**4 附录**

[包括: 算法实现的源代码、全部结果等]

**实验1 例如,用二分法求非线性方程的根**

**1 问题重述**

[包括问题背景、题目要求等]

宋体，五号，单倍行距

**2 算法介绍**

[包括:算法介绍（可以有流程图），收敛性、误差分析、参考文献等]

宋体，五号，单倍行距

**3 实验结果及分析**

[包括:必要的结果截图，对结果的说明、分析、讨论等]

宋体，五号，单倍行距

**4 附录**

[包括: 算法实现的源代码、全部结果等]