数值实验：方程组的求解

班级：M0x　　序号：xx

xx　　019xxxxxxxxx

一、 题目

对方程组$\boldsymbol{Ax}=\boldsymbol{b}$，

其中，系数矩阵$\boldsymbol{A}=\begin{bmatrix}6 & 1 & 0 & \cdots & 0\\ 8 & 6 & 1 & \cdots & 0\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ 0 & \cdots & 8 & 6 & 1\\ 0 & \cdots & 0 & 8 & 6\end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix}7\\ 15\\ \vdots\\ 15\\ 14\end{bmatrix}$，显然方程组的解为$\boldsymbol{x}=\begin{bmatrix}1\\ 1\\ \vdots\\ 1\\ 1\end{bmatrix}$。

1．取阶数 $n=10,30,100$，用顺序 Gauss 消元法求解此方程组的解，计算结果与精确解作比较；

2．取阶数 $n=10,30,100$，用追赶法求解此方程组的解，计算结果与精确解作比较；

3．取阶数 $n=10,30,100$，用列主元 Gauss 消元法求解此方程组的解，计算结果与精确解作比较；

4．取阶数 $n=10,30,100$，用 Jacobi 迭代法求解此方程组的解，计算结果与精确解作比较；

5．取阶数 $n=10,30,100$，用 Gauss_Seidel 迭代法求解此方程组的解，计算结果与精确解作比较；

6．比较这些计算结果后，你有什么体会？

二、 实验及结果

1．顺序 Gauss 消元法

1.1 相关理论

用顺序高斯消元法求解线性方程组的基本思想是用逐次消去未知数的方法把原线性方程组$\boldsymbol{Ax}=\boldsymbol{b}$化为与其等价的三角形线性方程组，而求解三角形线性方程组可用回代的方法求解。所以顺序高斯消元法的过程就是用行的初等变换将原线性方程组系数矩阵化为简单的形式（上三角矩阵），从而把原来求解线性方程组的问题转化为求解简单方程组的问题。

顺序高斯消元法主要包含两个过程：消元和回代。如果$a_{kk}{}^{(k)}\neq 0\quad(k=$

$1,2,\cdots,n)$, 则可通过高斯顺序消元法将$\boldsymbol{Ax}=\boldsymbol{b}$约化为等价的三角形线性方程组。在消元过程中，主要的计算公式可以表示为：

$$m_{ik}=\frac{{a_{ik}}^{(k)}}{{a_{kk}}^{(k)}},\quad i>k$$

$${a_{ij}}^{(k+1)}={a_{ij}}^{(k)}-m_{ik}{a_{kj}}^{(k)},\quad i>k,j\geq k$$

$${b_i}^{(k+1)}={b_i}^{(k)}-m_{ik}{b_k}^{(k)},\quad i>k$$

在回代过程中计算公式可以表示为：

$$x_n=\frac{{b_n}^{(n)}}{{a_{nn}}^{(n)}}$$

$$x_k=\frac{1}{{a_{kk}}^{(k)}}\left({b_k}^{(k)}-\sum_{j=k+1}^{n}{a_{kj}}^{(k)}x_j\right)\qquad k=n-1,n-2,\cdots,1$$

1.2 结果

我用 Python 编写了顺序高斯消元法，见附件 homework.py 文件中的函数 Sequential_Gaussian(n=30)。阶数分别取 10，30，100，得到结果如下：（方程组的结果太长，不方便截图，具体结果见文末附录）

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取10时，顺序高斯消元法的求解结果：
方程组的解为：[1.0, 1.0000000000000002, 0.9999999999999996, 1.0000000000000009, 0.9999999999999982,
与精确解误差的1范数为：7.571721027943568e-14
与精确解误差的2范数为：3.8444508186971805e-14
与精确解误差的无穷范数为：2.842170943040401e-14
```

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取30时，顺序高斯消元法的求解结果：
方程组的解为：[1.0, 1.0000000000000002, 0.9999999999999996, 1.0000000000000009, 0.9999999999999982,
与精确解误差的1范数为：7.946800906211138e-08
与精确解误差的2范数为：4.0297635582746016e-08
与精确解误差的无穷范数为：2.9800503398291767e-08
```

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取100时，顺序高斯消元法的求解结果：
方程组的解为：[1.0, 1.0000000000000002, 0.9999999999999996, 1.0000000000000009, 0.9999999999999982,
与精确解误差的1范数为：93819265613823.66
与精确解误差的2范数为：47575050905416.73
与精确解误差的无穷范数为：35182224605183.875
```

图 1 阶数 n 取 10，30，100 时，用顺序高斯消元法求解结果

2．追赶法

2.1 相关理论

求解$\boldsymbol{Ax}=\boldsymbol{f}$等价于求解两个三角形方程组：

① $\boldsymbol{Ly=f}$ 求$\boldsymbol{y}$；　　② $\boldsymbol{Ux=y}$ 求$\boldsymbol{x}$

从而得到求解三对角线方程组的追赶法公式：

（1） 计算$\{\beta_i\}$的递推公式：

$$\beta_1 = c_1/b_1,$$
$$\beta_i = c_i/(b_i - a_i\beta_{i-1}), \quad i = 2,3,\cdots,n-1$$

（2） 解$\boldsymbol{Ly=f}$

$$y = f_1/b_1,$$
$$y_i = (f - a_i y_{i-1})/(b_i - a_i\beta_{i-1}), \quad i = 2,3,\cdots,n$$

（3） 解$\boldsymbol{Ux=y}$

$$x_n = y_n,$$
$$x_i = y_i - \beta_i x_{i+1}, \quad i = n-1, n-2, \cdots, 1$$

计算系数$\beta_1 \to \beta_2 \to \cdots \to \beta_{n-1}$及$y_1 \to y_2 \to \cdots \to y_n$的过程就称为追的过程，计算方程组的解$x_1 \to x_2 \to \cdots \to x_n$的过程就是赶的过程。

追赶法公式实际上就是把高斯消去法用到求解三对角线方程组上去的结果。此时的计算公式非常简单，计算量仅为 *5n-4* 次乘除法，所以追赶法的计算量是较小的。在计算机实现时只需用三个一维数组分别存储 $\boldsymbol{A}$ 的三条线元素$\{a_i\}$，$\{b_i\}$，$\{c_i\}$，此外还需用两组工作单元保存$\{\beta_i\}$，$\{y_i\}$和$\{x_i\}$

2.2 结果

我用 Python 编写了追赶法，见附件 homework.py 文件中的函数 Chasing(n=30)。阶数分别取 10，30，100，得到结果如下：（方程组的结果太长，不方便截图，具体结果见文末附录）

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取10时，追赶法的求解结果：
方程组的解为：[1.0, 0.9999999999999998, 1.0000000000000004, 0.9999999999999991, 1.0000000000000018,
与精确解误差的1范数为：7.560618797697316e-14
与精确解误差的2范数为：3.843440743376389e-14
与精确解误差的无穷范数为：2.842170943040401e-14
```

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取30时，追赶法的求解结果：
方程组的解为：[1.0, 0.9999999999999998, 1.0000000000000004, 0.9999999999999991, 1.0000000000000018,
与精确解误差的1范数为：7.946800884006677e-08
与精确解误差的2范数为：4.029763558273537e-08
与精确解误差的无穷范数为：2.9800503398291767e-08
```

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取100时，追赶法的求解结果：
方程组的解为：[1.0, 0.9999999999999998, 1.0000000000000004, 0.9999999999999991, 1.0000000000000018,
与精确解误差的1范数为：93819265613823.66
与精确解误差的2范数为：47575050905416.73
与精确解误差的无穷范数为：35182224605183.875
```

图 2 阶数 n 取 10，30，100 时，用追赶法求解结果

3．列主元 Gauss 消元法

3.1 相关理论

用顺序高斯消元法求解线性方程组时，在消元过程中可能出现$a_{kk}^{(k)}=0$的情况，这时消去法将无法进行，即使主元素$a_{kk}^{(k)}\neq 0$，但是值很小时，用其作为除数，会导致其他元素数量级的严重增长和舍入误差的扩散，最后使得计算不可靠。对一般矩阵来说，最好每一步选取系数矩阵中当前列绝对值最大的元素作为主元素，避免主元过小，这样可使高斯消去法有较好的稳定性。

在进行到第 k 步时，选取右下角方阵$\boldsymbol{A}^{(k)}$的第一列中最大的元素，使满足

$$\left|a_{i_k,k}\right|=\max_{k\le i\le n}|a_{ik}|\neq 0$$

交换增广矩阵的第 k 行与第i_k行的元素，再用顺序高斯消元法中提到的计算公式进行消元计算，消元结束后，同样使用顺序高斯消元法的计算公式进行回代，最终可以得到求解结果。

3.2 结果

我用 Python 编写了列主元高斯消元法，见附件 homework.py 文件中的函数 Col_Gaussian(n=30)。阶数分别取 10，30，100，得到结果如下：（方程组的结果太长，不方便截图，具体结果见文末附录）

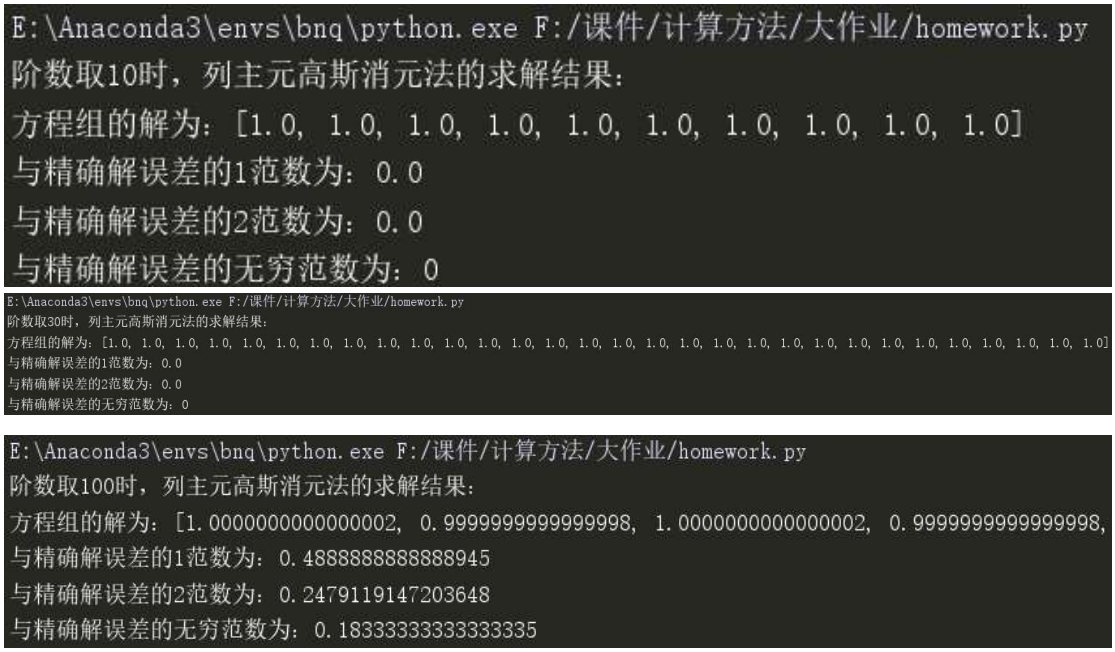

图 3 阶数 n 取 10，30，100 时，用列主元高斯消元法求解结果

4．Jacobi 迭代法

4.1 相关理论

将线性方程组的系数矩阵$\boldsymbol{A}$分成三部分$\boldsymbol{A}=\boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U}$，其中$\boldsymbol{D}$为对角矩阵，$\boldsymbol{L}$为对角线为 0 的下三角矩阵，$\boldsymbol{U}$为对角线为 0 的上三角矩阵。可以得到$\boldsymbol{Ax}=\boldsymbol{b}$ 的 Jacobi 迭代矩阵为：

$$\boldsymbol{x}^{k+1}=\boldsymbol{B}_J\boldsymbol{x}^k+\boldsymbol{f}_J$$

其中 $\boldsymbol{B}_J=\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$, $\boldsymbol{f}_J=\boldsymbol{D}^{-1}\boldsymbol{b}$。

进而可以得到的 Jacobi 迭代法的计算公式：

$$\boldsymbol{x}^{(0)}=\left({x_1}^{(0)},{x_2}^{(0)},\cdots,{x_n}^{(0)}\right)^T$$

$$x_i{}^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1,j\neq i}^{n}a_{ij}x_j{}^{(k)}\right)\quad i=1,2,\cdots$$

由上面的计算公式可知，Jacobi 迭代的计算非常简单，每迭代一次只需要计算一次矩阵和向量乘法且计算过程中原始矩阵$\boldsymbol{A}$始终不变。

4.2 结果

我用 Python 编写了 Jacobi 迭代法，见附件 homework.py 文件中的函数 Jacobi_iter (n=30, error=1e-5)。阶数分别取 10，30，100，得到结果如下：（方程组的结果太长，不方便截图，具体结果见文末附录）

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取10，终止条件为x_k和x_k+1误差的1范数小于1e-05时，Jacobi迭代法的求解结果：
迭代190次后
方程组的解为：[0.9999999999784233, 0.9999999992937134, 0.9999999953696296, 0.9999999904933435, 0.9999999514840336,
与精确解误差的1范数为：4.212020110028192e-06
与精确解误差的2范数为：2.3539414349821726e-06
与精确解误差的无穷范数为：1.6959640233293882e-06
```

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取30，终止条件为x_k和x_k+1误差的1范数小于1e-05时，Jacobi迭代法的求解结果：
迭代575次后
方程组的解为：[1.0, 1.0, 1.0, 1.0, 1.0, 1.0000000000000002, 1.000000000000001, 1.0000000000000024,
与精确解误差的1范数为：5.014817224369139e-06
与精确解误差的2范数为：2.8620296859497622e-06
与精确解误差的无穷范数为：2.4601896075893137e-06
```

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取100，终止条件为x_k和x_k+1误差的1范数小于1e-05时，Jacobi迭代法的求解结果：
迭代1777次后
方程组的解为：[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
与精确解误差的1范数为：4.898561587252104e-06
与精确解误差的2范数为：2.784239701618152e-06
与精确解误差的无穷范数为：2.3874642871568597e-06
```

图 4 阶数 n 取 10，30，100 时，用 Jacobi 迭代法求解结果

5．Gauss_Seidel 迭代法

5.1 相关理论

同样将线性方程组的系数矩阵$\boldsymbol{A}$分成三部分$\boldsymbol{A}=\boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U}$，其中$\boldsymbol{D}$为对角矩阵，$\boldsymbol{L}$为对角线为0的下三角矩阵，$\boldsymbol{U}$为对角线为0的上三角矩阵。可以得到$\boldsymbol{Ax}=\boldsymbol{b}$的 Gauss_Seidel 迭代矩阵为：

$$\boldsymbol{x}^{k+1}=\boldsymbol{B}_{G\_S}\boldsymbol{x}^{k}+\boldsymbol{f}_{G\_S}$$

其中 $\boldsymbol{B}_{G\_S}=(\boldsymbol{D}-\boldsymbol{L})^{-1}\boldsymbol{U}$，$\boldsymbol{f}_{G\_S}=(\boldsymbol{D}-\boldsymbol{L})^{-1}\boldsymbol{b}$。

进而可以得到的 Gauss_Seidel 迭代法的计算公式：

$$\boldsymbol{x}^{(0)}=\left({x_1}^{(0)},{x_2}^{(0)},\cdots,{x_n}^{(0)}\right)^T$$

$$x_i{}^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j{}^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}x_j{}^{(k)}\right)\quad i=1,2,\cdots$$

Jacobi 迭代法不使用变量的最新更新值$x_i{}^{(k+1)}$，而 Gauss_Seidel 迭代法却使用了第 i 个分量之前的最新更新值$x_j{}^{(k+1)},(j=1,2,\cdots,i-1)$。Gauss_Seidel 迭代法可以看作是 Jacobi 迭代法的一种改进，Gauss_Seidel 迭代法每迭代一次只需要计算一次矩阵和向量乘法。

5.2 结果

我用 Python 编写了 Gauss_Seidel 迭代法，见附件 homework.py 文件中的函数 Gauss_Seidel_iter (n=30, error=1e-5)。阶数分别取 10，30，100，得到结果如下：（方程组的结果太长，不方便截图，具体结果见文末附录）

```
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取10，终止条件为x_k和x_k+1误差的1范数小于1e-05时，Gauss_Seidel迭代法的求解结果：
迭代68次后
方程组的解为：[1.0000000037098156, 0.999999981784771, 1.0000000651499727, 0.9999997993622828,
与精确解误差的1范数为：4.3293877529770874e-05
与精确解误差的2范数为：2.323273597941408e-05
与精确解误差的无穷范数为：1.7436007729187963e-05
E:\Anaconda3\envs\bnq\python.exe F:/课件/计算方法/大作业/homework.py
阶数取30，终止条件为x_k和x_k+1误差的1范数小于1e-05时，Gauss_Seidel迭代法的求解结果：
迭代233次后
方程组的解为：[1.0000000000000002, 0.9999999999999996, 1.000000000000001, 0.9999999999999973,
与精确解误差的1范数为：7.180809051299253e-05
与精确解误差的2范数为：3.7713128423297146e-05
与精确解误差的无穷范数为：2.815263976041482e-05
```

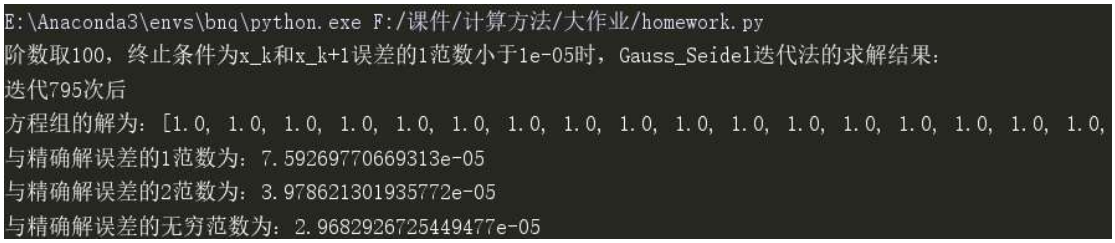

图 5 阶数 n 取 10，30，100 时，用顺序高斯消元法求解结果

6．思考

在本次的实验中，我共采用了 5 种方法求解给定的线性方程组$\boldsymbol{Ax}=\boldsymbol{b}$，包括 3 种直接求解的方法：顺序高斯消元法、追赶法和列主元高斯消元法，还有两种迭代求解的方法：Jacobi 迭代法和 Gauss_Seidel 迭代法。对于这 5 种求解线性方程组的方法，我使用了 Python 语言进行编程，得到了题目中线性方程组在阶数为 10、30 和 100 时的解，并使用 1 范数、2 范数和无穷范数来表示该解与精确解的误差。

从求解结果来看，3 种直接求法当中，列主元高斯消元法求得的解与精确解的误差最小，在阶数为 10 和 30 时，列主元高斯消元法求得的解与精确解误差的 1 范数、2 范数和无穷范数都为 0.0，而在 100 阶时误差也控制在可接受的数量级。然而顺序高斯消元法和追赶法虽然在阶数为 10 和 30 时计算的误差很小，但在阶数达到 100 时，误差却达到了 10^{14} 的数量级，远大于列主元高斯消元算得的误差。究其原因，并结合上述对这几种方法基本思想的介绍，我认为列主元高斯消元法避免了小数当主元的情况，从而减少了小数做分母产生的误差，在阶数很高时效果更加明显。相比于顺序高斯消元法，列主元高斯消元法还避免了主元为 0 的情况，虽然在本题中未出现这种情况，但也是列主元高斯消元法的一个优点。此外，我还计算了这 3 种方法计算所需的时间，在 100 阶时，顺序高斯消元法、追赶法和列主元高斯消元法运行的时间分别为 65.976ms，0.098ms，46.973ms，虽然运行时间都很短，但是还是可以看出追赶法的运行时间相对于其他两种直接求法要短很多，因为对于这种三对角方程，追赶法的计算量仅为 O(n)数量级，而其余两种方法都是 $O(n^3)$的数量级，这也是追赶法的一个优势。

在 Jacobi 迭代法和 Gauss_Seidel 迭代法的编程中，我都令迭代停止的条件为 $\| x^{k+1}-x^{k} \|_1 < 10^{-5}$，从结果来看，Jacobi 迭代法算得的结果与精确解的误差在 10^{-6} 数量级，而 Gauss_Seidel 迭代法算得的结果与精确解的误差在 10^{-5} 数量级，两者相差不大，Jacobi 迭代法的结果略微优于 Gauss_Seidel 迭代法。而从收敛速度方面来看，Jacobi 迭代法在阶数为 10、30 和 100 时分别用了 150、975 和 1777 次迭代完成了收敛，而 Gauss_Seidel 迭代法在阶数为 10、30 和 100 时分别用了 68、233 和 795 次迭代完成了收敛，可见在本题中 Gauss_Seidel 迭代法的收敛速度要快于 Jacobi 迭代法，通过计算$\boldsymbol{B}_J$和$\boldsymbol{B}_{G_S}$的谱半径，有$\rho(\boldsymbol{B}_{G_S}) < \rho(\boldsymbol{B}_J)$，同样可

以得出在本题中 Gauss_Seidel 迭代法的收敛速度更快的结论。

本次实验中，我通过对线性方程组采用 5 种不同的方法进行求解，进一步了解了每种方法的基本思想和他们各自的优缺点，对线性方程组的求解方法有了更加清晰的认识。

三、 附录

1、 顺序高斯消元法

n=10

[1.0, 1.0000000000000002, 0.9999999999999996, 1.0000000000000009, 0.9999999999999982, 1.0000000000000036, 0.9999999999999933, 1.0000000000000124, 0.9999999999999787, 1.0000000000000284]

n=30

[1.0, 1.0000000000000002, 0.9999999999999996, 1.0000000000000009, 0.9999999999999982, 1.0000000000000036, 0.9999999999999929, 1.0000000000000142, 0.9999999999999716, 1.0000000000000568, 0.9999999999998863, 1.0000000000002274, 0.9999999999995453, 1.0000000000009095, 0.999999999998181, 1.0000000000036378, 0.9999999999927249, 1.0000000000145493, 0.999999999970905, 1.0000000000581757, 0.9999999998837055, 1.0000000002323617, 0.9999999995361861, 1.0000000009239902, 0.9999999981665706, 1.0000000036086547, 0.999999993015507, 1.0000000130377202, 0.9999999776496225, 1.0000000298005034]

n=100

[1.0, 1.0000000000000002, 0.9999999999999996, 1.0000000000000009, 0.9999999999999982, 1.0000000000000036, 0.9999999999999929, 1.0000000000000142, 0.9999999999999716, 1.0000000000000568, 0.9999999999998863, 1.0000000000002274, 0.9999999999995453, 1.0000000000009095, 0.999999999998181, 1.000000000003638, 0.9999999999927245, 1.000000000014551, 0.999999999970898, 1.000000000058204, 0.9999999998835918, 1.0000000002328164, 0.9999999995343671, 1.0000000009312657, 0.9999999981374685, 1.000000003725063, 0.9999999925498742, 1.0000000149002517, 0.9999999701994966, 1.0000000596010068, 0.9999998807979864, 1.0000002384040272, 0.9999995231919456, 1.0000009536161087, 0.9999980927677825, 1.000003814464435, 0.99999237107113, 1.00001525785774, 0.9999694842845201, 1.0000610314309597, 0.9998779371380806, 1.0002441257238388, 0.9995117485523224, 1.0009765028953552, 0.9980469942092896, 1.003906011581421, 0.9921879768371582, 1.0156240463256836, 0.9687519073486328, 1.0624961853027342, 0.8750076293945318, 1.2499847412109362, 0.5000305175781286, 1.9999389648437393, -0.9998779296874645, 4.999755859374872, -6.999511718749517, 16.999023437498124, -30.99804687499261, 64.99609374997067, -126.99218749988313, 256.98437499953343, -510.96874999813565, 1024.9374999925462, -2046.8749999701922, 4096.749999880783, -8190.499999523163, 16383.99999809271, -32764.999992370955, 65532.99996948405, -131062.99987793667, 262128.99951174762, -524254.99804699235, 1048512.9921879731, -2097022.9687519, 4194048.8750076145, -8388094.500030488, 16776191.00012201, -33552375.000488162, 67104737.00195289, -134209407.00781202, 268418561.03124905, -536836095.1249981, 1073668097.4999962, -2147319808.9999924,

4294574088.9999847, -8588886046.9999695, 17176723584.999939, -34349253118.999878, 68681730048.999756, -137296355326.99951, 274324291584.99902, -547574906878.99805, 1090855108608.9961, -2164531396606.9922, 4260347510784.985, -8245833891838.971, 15392223264768.945, -26386668453886.906, 35182224605184.875]

2、 追赶法

n=10

[1.0, 0.9999999999999998, 1.0000000000000004, 0.9999999999999991, 1.0000000000000018, 0.9999999999999966, 1.0000000000000067, 0.9999999999999876, 1.0000000000000213, 0.9999999999999716]

n=30

[1.0, 0.9999999999999998, 1.0000000000000004, 0.9999999999999991, 1.0000000000000018, 0.9999999999999964, 1.000000000000007, 0.9999999999999858, 1.0000000000000284, 0.9999999999999432, 1.0000000000001137, 0.9999999999997727, 1.0000000000004547, 0.9999999999990905, 1.000000000001819, 0.9999999999963624, 1.000000000007275, 0.9999999999854507, 1.000000000029095, 0.9999999999418243, 1.0000000001162945, 0.9999999997676383, 1.0000000004638139, 0.9999999990760098, 1.0000000018334294, 0.9999999963913453, 1.000000006984493, 0.9999999869622798, 1.0000000223503775, 0.9999999701994966]

n=100

[1.0, 0.9999999999999998, 1.0000000000000004, 0.9999999999999991, 1.0000000000000018, 0.9999999999999964, 1.000000000000007, 0.9999999999999858, 1.0000000000000284, 0.9999999999999432, 1.0000000000001137, 0.9999999999997727, 1.0000000000004547, 0.9999999999990905, 1.000000000001819, 0.9999999999963622, 1.0000000000072755, 0.999999999985449, 1.000000000029102, 0.9999999999417959, 1.0000000001164082, 0.9999999997671836, 1.0000000004656329, 0.9999999990687343, 1.0000000018625315, 0.9999999962749371, 1.0000000074501258, 0.9999999850997483, 1.0000000298005034, 0.9999999403989932, 1.0000001192020136, 0.9999997615959728, 1.0000004768080544, 0.9999990463838913, 1.0000019072322175, 0.999996185535565, 1.00000762892887, 0.9999847421422601, 1.0000305157154799, 0.9999389685690403, 1.0001220628619194, 0.9997558742761612, 1.0004882514476776, 0.9990234971046448, 1.0019530057907104, 0.9960939884185791, 1.0078120231628418, 0.9843759536743164, 1.0312480926513672, 0.9375038146972658, 1.1249923706054683, 0.7500152587890638, 1.4999694824218714, 6.103515626065814e-05, 2.9998779296874645, -2.999755859374872, 8.999511718749517, -14.999023437498124, 32.99804687499261, -62.99609374997067, 128.99218749988313, -254.98437499953346, 512.9687499981356, -1022.9374999925462, 2048.874999970192, -4094.7499998807834, 8192.499999523163, -16381.99999809271, 32766.999992370955, -65530.99996948405, 131064.99987793667, -262126.99951174762, 524256.99804699235, -1048510.9921879731, 2097024.9687519, -4194046.8750076145, 8388096.500030488, -16776189.00012201, 33552377.000488162, -67104735.00195289, 134209409.00781202, -268418559.03124905, 536836097.1249981, -1073668095.4999962, 2147319810.9999924, -4294574086.9999847, 8588886048.9999695, -17176723582.999939, 34349253120.999878, -68681730046.999756, 137296355328.99951, -274324291582.99902, 547574906880.99805, -1090855108606.9961, 2164531396608.9922, -4260347510782.985, 8245833891840.971, -15392223264766.945, 26386668453888.906, -35182224605182.875]

3、 列主元高斯消元法

n=10

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

n=30

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

n=100

[1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000002, 0.9999999999999998, 1.0000000000000004, 0.9999999999999989, 1.0000000000000024, 0.9999999999999949, 1.0000000000000102, 0.9999999999999793, 1.0000000000000415, 0.9999999999999167, 1.0000000000001665, 0.9999999999996667, 1.0000000000006668, 0.9999999999986662, 1.0000000000026676, 0.9999999999946645, 1.0000000000106712, 0.9999999999786573, 1.0000000000426854, 0.999999999914629, 1.0000000001707423, 0.9999999996585152, 1.0000000006829697, 0.9999999986340604, 1.0000000027318794, 0.999999994536241, 1.000000010927518, 0.9999999781449642, 1.0000000437100702, 0.9999999125798649, 1.0000001748402496, 0.999999650319584, 1.0000006993604984, 0.9999986012803372, 1.00000279743399, 0.9999944051533627, 1.0000111896079034, 0.9999776211256783, 1.0000447563827037, 0.9999104926983515, 1.0001789927482605, 0.9996421019236248, 1.000715446472168, 0.9985705057779948, 1.0028533935546875, 0.9943155924479168, 1.011279296875, 0.9777994791666667, 1.04296875, 0.9197916666666668, 1.1375, 0.8166666666666667]

4、 Jacobi 迭代法

n=10

[0.9999999999784233, 0.9999999992937134, 0.9999999953696296, 0.9999999904933435, 0.9999999514840336, 0.9999999172422059, 0.9999996433160673, 0.9999994945014165, 0.9999983040359767, 0.9999984924592704]

n=30

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0000000000000002, 1.000000000000001, 1.0000000000000024, 1.0000000000000067, 1.0000000000000238, 1.0000000000000486, 1.000000000000201, 1.000000000000369, 1.0000000000016014, 1.000000000002874, 1.0000000000123637, 1.000000000022008, 1.0000000000932825, 1.0000000001631786, 1.0000000006812961, 1.0000000011610728, 1.0000000047553075, 1.0000000078336397, 1.0000000311498434,

1.000000048968846, 1.0000001849760713, 1.0000002689497451, 1.0000009108735723, 1.0000010949822615, 1.0000024601896076]

n=100

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0000000000000002, 1.0000000000000004, 1.0000000000000013, 1.000000000000005, 1.0000000000000082, 1.000000000000039, 1.0000000000000624, 1.00000000000028, 1.0000000000004778, 1.0000000000020632, 1.0000000000035874, 1.0000000000151128, 1.0000000000260896, 1.000000000108423, 1.0000000001838592, 1.0000000007544634, 1.0000000012517567, 1.000000005044637, 1.0000000081455158, 1.0000000318977822, 1.0000000495018269, 1.0000001844080821, 1.0000002664124505, 1.000000891927203, 1.0000010714135728, 1.0000023874642872]

5、 Gauss_Seidel 迭代法

n=10

[1.0000000037098156, 0.999999981784771, 1.0000000651499727, 0.9999997993622828, 1.0000005586171075, 0.9999985706955125, 1.0000033608115002, 0.999992855581826, 1.000013077005797, 0.9999825639922708]

n=30

[1.0000000000000002, 0.9999999999999996, 1.000000000000001, 0.9999999999999973, 1.0000000000000067, 0.9999999999999828, 1.0000000000000453, 0.9999999999998789, 1.000000000000327, 0.999999999999113, 1.0000000000024065, 0.9999999999934869, 1.000000000017547, 0.9999999999530109, 1.0000000001249412, 0.9999999996704161, 1.0000000008619128, 0.9999999977670132, 1.0000000057264984, 0.9999999854770842, 1.000000036378392, 0.9999999101449238, 1.0000002183565428, 0.9999994796292905, 1.0000012103369469, 0.999997272896887, 1.0000058780599095, 0.9999881633634611, 1.0000211144798203, 0.9999718473602396]

n=100

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0000000000000002, 0.9999999999999994, 1.0000000000000016, 0.9999999999999959, 1.0000000000000113, 0.9999999999999689, 1.0000000000000842, 0.999999999999775, 1.0000000000005957, 0.9999999999984364, 1.0000000000040752, 0.9999999999894449, 1.0000000000271827, 0.9999999999303779, 1.000000000177356, 0.9999999995507124, 1.0000000011314707, 0.9999999971687051, 1.0000000070347101, 0.9999999826612466, 1.000000042340328, 0.9999998977365906, 1.0000002437288091, 0.9999994286701003, 1.0000013108687968, 0.9999970781677833, 1.0000062472551225, 0.9999874868401101, 1.000022262195044, 0.9999703170732746]

代码

```
import math    # 仅用于调用 math.sqrt 求解平方根
import time    # 仅用于计算运行时间


# 生成增广矩阵[A|b]
def GenerateMatrix(n = 30):
    A = [[] for _ in range(n)]
    A[0] =[6, 1] + (n - 2) * [0] + [7]
    A[-1] = (n - 2) * [0] + [8, 6] + [14]
    for i in range(1, n-1):
        A[i] = (i - 1) * [0] + [8, 6, 1] + (n - i - 2) * [0] + [15]
    return A

# 1 范数
def Norm_1(x, y):
    norm = 0
    for i in range(len(x)):
        norm += abs(y[i] - x[i])
    return norm

# 2 范数
def Norm_2(x, y):
    norm = 0
    for i in range(len(x)):
        norm += (y[i] - x[i]) ** 2
    return math.sqrt(norm)

# 无穷范数
def Norm_infinity(x, y):
    max_x = 0
    for i in range(len(x)):
        if max_x < abs(x[i] - y[i]):
            max_x = abs(x[i] - y[i])
    return max_x
```

```
# 顺序高斯消元法
def Sequential_Gaussian(n = 30):
    A = GenerateMatrix(n)   # 生成 n 阶系数矩阵
    x = [0] * n   # 初始化解为 n 维 0 向量
    # 消元过程
    for k in range(n):
        for i in range(k + 1, n):
            m = A[i][k] / A[k][k]     # 计算比值 m
            for j in range(k, n):
                A[i][j] -= m * A[k][j]     # 更新 a
            A[i][-1] -= m * A[k][-1]     # 更新 b
    # 回代过程
    # print(A)
    x[-1] = (A[n - 1][-1] / A[n - 1][n - 1])
    for k in range(n-2, -1 ,-1):
        s = 0
        for j in range(k + 1, n):
            s += A[k][j] * x[j]     # 求和
        x[k] = (A[k][-1] - s) / A[k][k]
    return x


# 追赶法
def Chasing(n = 30):
    A = GenerateMatrix(n)   # 生成 n 阶系数矩阵
    x = [0] * n   # 初始化解为 n 维 0 向量
    a = [0] + [8] * (n - 1); b = n * [6]; c = [1] * (n - 1) + [0]
    alpha = [0] * n; beta = [0] * n; y = [0] * n   # 定义 alpha, y, beta 为 n 维向量
    # alpha, y, beta 的初值
    alpha[0] = b[0]
    y[0] = A[0][-1] / alpha[0]
    beta[0] = c[0] / alpha[0]
    # 追 过程
    for i in range(1, n):
        alpha[i] = b[i] - a[i] * beta[i - 1]   # 更新 alpha
        y[i] = (A[i][-1] - a[i] * y[i - 1]) / alpha[i]   # 更新 y
        beta[i] = c[i] / alpha[i]     # 更新 beta
    # 赶 过程
```

```
    x[-1] = y[-1]
    for i in range(n - 2, -1, -1):
        x[i] = y[i] - beta[i] * x[i + 1]
    return x

# 列主元素消元法
def Col_Gaussian(n = 30):
    A = GenerateMatrix(n)  # 生成 n 阶系数矩阵
    x = [0] * n  # 初始化解为 n 维 0 向量
    # 消元过程
    for k in range(n - 1):
        max_x = 0; idx = 0
        for i in range(k, n):
            if max_x < abs(A[i][k]):
                max_x = abs(A[i][k]); idx = i  # 找出第 k 列元素最大的行
        if idx != k:
            for j in range(k, n):
                A[k][j], A[idx][j] = A[idx][j], A[k][j]  # 交换系数矩阵的行
            A[k][-1], A[idx][-1] = A[idx][-1], A[k][-1]  # 交换 b 的行
        for i in range(k + 1, n):
            m = A[i][k] / A[k][k]  # 计算比值 m
            for j in range(k + 1, n):
                A[i][j] -= m * A[k][j]  # 更新 a
            A[i][-1] -= m * A[k][-1]  # 更新 b
    # 回代过程
    x[-1] = A[-1][-1] / A[-1][-2]
    for i in range(n - 2, -1, -1):
        s = 0
        for j in range(i + 1, n):
            s += A[i][j] * x[j]  # 求和
        x[i] = (A[i][-1] - s) / A[i][i]
    return x

# Jacobi 迭代法
def Jacobi_iter(n = 30, error = 1e-5):
    A = GenerateMatrix(n)
    x = [0] * n; x_next = [0] * n
```

```
    iter_num = 0
    x_next = Jacobi_loop(n, A, x, x_next)
    while (Norm_1(x, x_next) > error):
    #while iter_num < 10000:
        x = [i for i in x_next]
        iter_num += 1
        x_next = Jacobi_loop(n, A, x, x_next)
    return x_next, iter_num + 1

# Jacobi 迭代法中的循环部分
def Jacobi_loop(n, A, x, x_next):
    for i in range(n):
        s1 = 0; s2 = 0
        for j in range(n):
            if j < i:
                s1 += A[i][j] * x[j]
            elif j > i:
                s2 += A[i][j] * x[j]
        x_next[i] = (A[i][-1] - s1 - s2) / A[i][i]
    return x_next

#  高斯-赛德尔迭代法
def Gauss_Seidel_iter(n = 30, error = 1e-5):
    A = GenerateMatrix(n)
    x = [0] * n; x_next = [0] * n
    iter_num = 0
    x_next = Gauss_Seidel_loop(n, A, x, x_next)
    while (Norm_1(x, x_next) > error):
    #while iter_num < 10000:
        x = [i for i in x_next]
        iter_num += 1
        x_next = Gauss_Seidel_loop(n, A, x, x_next)
    return x_next, iter_num + 1

#  高斯-赛德尔迭代法中的循环部分
def Gauss_Seidel_loop(n, A, x, x_next):
    for i in range(n):
```

```
            s0 = 0; s1 = 0
            for j in range(n):
                if j < i:
                    s0 += A[i][j] * x_next[j]
                elif j > i:
                    s1 += A[i][j] * x[j]
            x_next[i] = (A[i][-1] - s0 - s1) / A[i][i]
        return x_next


def PrintResult(n, method, error = 1e-5):
    methods = {Sequential_Gaussian: "顺序高斯消元法", Chasing: "追赶法", Col_Gaussian: "
列主元高斯消元法", Jacobi_iter: "Jacobi 迭代法", Gauss_Seidel_iter: "Gauss_Seidel 迭代法"}
    if method in methods.keys():
        x_real = [1] * n
        if method in [Sequential_Gaussian, Chasing, Col_Gaussian]:
            print('阶数取{}时，{}的求解结果：'.format(n, methods.get(method)))
            x= method(n)
        else:
            print('阶数取{}，终止条件为 x_k 和 x_k+1 误差的 1 范数小于{}时，{}的求解
结果：'.format(n, error, methods.get(method)))
            x, iter_num = method(n, error)
            print('迭代{}次后'.format(iter_num))
        print('方程组的解为：{}'.format(x))
        print('与精确解误差的 1 范数为：{}'.format(Norm_1(x, x_real)))
        print('与精确解误差的 2 范数为：{}'.format(Norm_2(x, x_real)))
        print('与精确解误差的无穷范数为：{}'.format(Norm_infinity(x, x_real)))
    else:
        print('请检查求解方法的名称！')




if __name__ == '__main__':
    start = time.time()
    #PrintResult(100, Sequential_Gaussian)
```

```
PrintResult(100, Chasing)
#PrintResult(100, Col_Gaussian)
#PrintResult(100, Jacobi_iter)
#PrintResult(100, Gauss_Seidel_iter)
end = time.time()
print((end - start) * 1000, 'ms')
```