

计算方法大作业

Log Creative

2023年11月24日

目 录

目	录	1
1	问题	1
2	约定	2
3	顺序 Guass 消元法	3
4	列主元 Guass 消元法	4
5	追赶法	5
6	Jacobi 迭代法	7
7	Guass-Seidel 迭代法	8
8	结果比较与总结	9

1 问题

解方程组 $\mathbf{A}\mathbf{x} = \mathbf{b}$, 其中

$$A = \begin{pmatrix} 6 & 1 & & & \\ 8 & 6 & 1 & & & \\ & \ddots & \ddots & \ddots & \\ & & 8 & 6 & 1 \\ & & & 8 & 6 \end{pmatrix} \qquad b = \begin{pmatrix} 7 \\ 15 \\ \vdots \\ 15 \\ 14 \end{pmatrix}$$

显然精确解为 $\mathbf{x}^* = (1, 1, \cdots, 1)^{\mathsf{T}}$ 。



2 约定

基于 Python 3 以及 numpy 库(下为 np)编写,所有矩阵的数据类型均为 64 位浮点数(np.float 64)。 下述解法均派生于一个公共类 Solver,每个子类将会实现方法 solve():

考虑到输出每行均为 1,输出将会使用 ∞-范数与精确值 $\|x - x^*\|_{\infty}$ 进行比较。

对于迭代法:Jacobi 迭代法(第 6 节)和 Guass–Seidel 迭代法(第 7 节),均派生于 IterativeSolver 类。设定迭代初值为 $\mathbf{x}_0 = (0,0,\cdots,0)^{\mathsf{T}}$,最大迭代次数为 1000。迭代终止条件为 $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_{\infty} < 10^{-4}$ 或达到了最大迭代次数,实现于 self.iter_manager()。

```
class IterativeSolver(Solver):

###

####

####

def __init__(self, A: np.array, b: np.array, x0 = None, maxiter = 1000, xnormthres = 1e-4):

### super().__init__(A, b)

### assert self.m == self.n

### if x0 is None:

### x0 = np.zeros(len(A))

### self.x0 = x0

### self.xnormlist = []

### self.xnormthres = xnormthres

### def iter_manager(self, xnorm, iternum):

### if iternum > len(self.xnormlist) + 1:

### self.xnormlist.clear()

### self.xnormlist.append(xnorm)

### return xnorm >= self.xnormthres and iternum < self.maxiter
```



3 顺序 Guass 消元法

```
class SeqGuassSolver(Solver):
    顺序 Guass 消元法
    H/H/H
   def solve(self):
       A = np.copy(self.A)
       b = np.copy(self.b)
        # 消元过程
       for k in range(self.m):
           for i in range(k + 1, self.n):
               coeff = A[i][k] / A[k][k]
               for j in range(self.m):
                   A[i][j] -= A[k][j] * coeff
               b[i] -= b[k] * coeff
        # 回代过程
       x = np.empty(self.m)
       for i in reversed(range(self.n)):
           s = b[i]
           for j in range(i + 1, self.m):
              s -= A[i][j] * x[j]
           x[i] = s / A[i][i]
       return x
```

表 1 顺序 Guass 消元法结果与精确值的比 一较

顺序 Guass 消元法:	n=10 inf	-norm=2.842170	943040401e-14
[1. 1. 1. 1. 1.]	1. 1. 1. 1. 1.]		
顺序 Guass 消元法:	n=30 inf	-norm=2.980050	3398291767e-08
[1. 1.	1.	1. 1	. 1.
1. 1.	1.	1. 1	. 1.
1. 1.	1.	1. 1	. 1.
1. 1.	1.	1. 1	. 1.
1. 1.	0.9999999	9 1.00000001 0	.99999998 1.00000003
顺序 Guass 消元法:	n=100 in	f-norm=3518222	4605183.875
[1.00000000e+00	1.00000000e+00	1.00000000e+	00 1.00000000e+00
1.00000000e+00	1.00000000e+00	1.00000000e+	00 1.00000000e+00
1.00000000e+00	1.00000000e+00		
1.00000000e+00	1.00000000e+00	1.00000000e+	00 1.00000000e+00
1.00000000e+00	1.00000000e+00	1.00000000e+	00 1.00000000e+00
1.00000000e+00	1.00000000e+00	1.00000000e+	00 1.0000000e+00
9.99999998e-01	1.00000000e+00	9.99999993e-	
9.99999970e-01	1.00000006e+00	9.99999881e-	01 1.00000024e+00
9.99999523e-01	1.00000095e+00	9.99998093e-	01 1.00000381e+00
9.99992371e-01	1.00001526e+00	9.99969484e-	
9.99877937e-01	1.00024413e+00	9.99511749e-	01 1.00097650e+00
9.98046994e-01	1.00390601e+00	9.92187977e-	
9.68751907e-01	1.06249619e+00	8.75007629e-	
5.00030518e-01	1.99993896e+00		
-6.99951172e+00	1.69990234e+01		
-1.26992187e+02	2.56984375e+02		
-2.04687500e+03	4.09675000e+03		
-3.27650000e+04	6.55330000e+04		
-5.24254998e+05	1.04851299e+06		
-8.38809450e+06	1.67761910e+07		
-1.34209407e+08	2.68418561e+08		
-2.14731981e+09	4.29457409e+09		
-3.43492531e+10	6.86817300e+10		
-5.47574907e+11	1.09085511e+12		
-8.24583389e+12	1.53922233e+13	-2.63866685e+	13 3.51822246e+13]

阶数 n	∞-范数 x - x * ∞
10	2.84×10^{-14}
30	2.98×10^{-8}
100	3.52×10^{13}

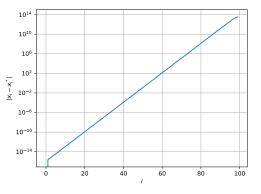


图 1 顺序 Guass 消元法结果

图 2 顺序 Guass 消元法 n = 100 时结果每个 坐标与精确值的差值绝对值



4 列主元 Guass 消元法

蓝色部分为列主元 Guass 消元法相较于顺序 Guass 消元法增加的代码部分。

```
class PivotGuassSolver(Solver):
    选主元 Guass 消元法
    def solve(self):
       A = np.copy(self.A)
        b = np.copy(self.b)
        # 消元过程
        for k in range(self.m):
           # 选主元
           max\_pivot = abs(A[k][k])
           max\_line = k
            for p in range(k + 1, self.n):
                if abs(A[p][k]) > max_pivot:
                    max_pivot = abs(A[p][k])
                   max_line = p
            # 交换行元素
            for j in range(self.m):
                tmp = A[k][j]
                A[k][j] = A[max\_line][j]
               A[max\_line][j] = tmp
            tmp = b[k]
            b[k] = b[max\_line]
            b[max\_line] = tmp
            for i in range(k + 1, self.n):
                coeff = A[i][k] / A[k][k]
                for j in range(self.m):
                   A[i][j] -= A[k][j] * coeff
                b[i] -= b[k] * coeff
        # 回代过程
        x = np.empty(self.m)
        for i in reversed(range(self.n)):
            s = b[i]
            for j in range(i + 1, self.m):
               s -= A[i][j] * x[j]
            x[i] = s / A[i][i]
        return x
```



表 2 列主元 Guass 消元法结果与精确值的 比较

제소규	冰 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 -						
	列主元 Guass 消元法: n=10 inf-norm=0.0						
列主元 Guass							
[1. 1. 1. 1	. 1. 1. 1.	1. 1. 1. 1.	1. 1. 1. 1	. 1. 1. 1. :	1. 1. 1. 1.		
1. 1. 1. 1							
列主元 Guass		10 ir	f-norm=0.18	333333333333	3335		
[1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	1.	1.	1.	1.	1.		
1.	0.99999999	1.00000001	0.99999998	1.00000004	0.99999991		
1.00000017	0.99999965	1.0000007	0.9999986	1.0000028	0.99999441		
1.00001119	0.99997762	1.00004476	0.99991049	1.00017899	0.9996421		
1.00071545	0.99857051	1.00285339	0.99431559	1.0112793	0.97779948		
1.04296875	0.91979167	1.1375	0.81666667	1			
				-			

阶数 n	∞-范数 x - x * ∞
10	0.000
30	0.000
100	0.183

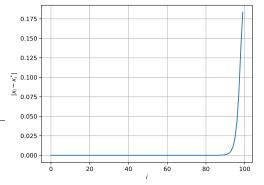


图 3 列主元 Guass 消元法结果

图 **4** 列主元 **Guass** 消元法 *n* = 100 时结果每个坐标与精确值的差值绝对值

5 追赶法

追赶法增加了对是否是三对角矩阵以及对角占优矩阵的检查。

```
class ThomasSolver(Solver):
   追赶法
    11 11 11
   def solve(self):
       A = np.copy(self.A)
       b = np.copy(self.b)
        # 检查是否是三对角矩阵以及对角占优
       is_square = self.m == self.n
       is_lower_zero = np.array_equal(np.tril(A, -2), np.zeros([self.n, self.m]))
       is_upper_zero = np.array_equal(np.triu(A, 2), np.zeros([self.n, self.m]))
       if not (is_square and is_lower_zero and is_upper_zero):
           print("不是三对角矩阵!")
       is\_dominant = (abs(A[0][0]) > abs(A[0][1]) > 0)
       for i in range(self.n - 1):
           is_dominant &= (abs(A[i][i]) >= abs(A[i][i-1]) + abs(A[i][i+1]))
       is\_dominant \&= (abs(A[self.n-1][self.n-1]) > abs(A[self.n-1][self.n-2]) >
        → 0)
```



```
if not is_dominant:
    print(" 不是对角占优! ")
# 计算 beta
beta = np.empty(self.n - 1)
beta[0] = A[0][1] / A[0][0]
for i in range(1, self.n - 1):
   beta[i] = A[i][i+1] / (A[i][i] - A[i][i-1] * beta[i-1])
# 解 Ly=f
y = np.empty(self.n)
y[0] = b[0] / A[0][0]
for i in range(1, self.n):
    y[i] = (b[i] - A[i][i-1] * y[i-1]) / (A[i][i] - A[i][i-1] * beta[i-1])
# 解 Ux=y
x = np.empty(self.n)
x[self.n - 1] = y[self.n - 1]
for i in reversed(range(self.n - 1)):
    x[i] = y[i] - beta[i] * x[i+1]
return x
```

```
追赶法: n=10 inf-norm=2.84
[1. 1. 1. 1. 1. 1. 1. 1. 1. ]
不是对角占优!
                       inf-norm=2.842170943040401e-14
追赶法: n=30
                       inf-norm=2.9800503398291767e-08
                            1.
                            1.00000001 0.9999999 1.00000002 0.99999997]
不是对角占优!
追赶法: n=100
[ 1.00000000e+00
                                                              1.00000000e+00
                      1.00000000e+00 1.0000000e+00
  1 000000000+00
                      1.00000000e+00
                                          1.00000000e+00
                                                              1 000000000+00
                      1.00000000e+00
  1.00000000e+00
                      1.00000000e+00
                                           1.00000000e+00
                                                              1.00000000e+00
  1.00000000e+00
                      1.00000000e+00
1.00000000e+00
                                           1.00000000e+00
1.00000000e+00
                                                              1.00000000e+00
  1.00000000e+00
                      9.99999996e-01
                                           1.00000001e+00
                                                              9.99999985e-01
                      9.99999940e-01
9.99999046e-01
                                          1.00000012e+00
1.00000191e+00
                                                              9.99999762e-01
9.99996186e-01
  1.00000003e+00
  1.000000048e+00
                                                              9.99938969e-01
9.99023497e-01
9.84375954e-01
  1.00000763e+00
                      9.99984742e-01
                                           1.00003052e+00
                                          1.00048825e+00
1.00781202e+00
  1.00012206e+00
                      9.99755874e-01
  1.00195301e+00
                      9.96093988e-01
  1.03124809e+00
                      9.37503815e-01
                                           1.12499237e+00
                                                              7.50015259e-01
  1.49996948e+00
8.99951172e+00
                                           2.99987793e+00 -2.99975586e+00
3.29980469e+01 -6.29960937e+01
                     -1.49990234e+01
  1.28992187e+02 -2.54984375e+02
                                           5.12968750e+02 -1.02293750e+03
  2.04887500e+03
                    -4.09475000e+03
                                           8.19250000e+03
  3.27670000e+04 -6.55310000e+04
                                          1.31065000e+05 -2.62127000e+05
  5.24256998e+05 -1.04851099e+06
8.38809650e+06 -1.67761890e+07
                                          2.09702497e+06 -4.19404688e+06
3.35523770e+07 -6.71047350e+07
  1.34209409e+08 -2.68418559e+08
                                           5.36836097e+08 -1.07366810e+09
  2.14731981e+09 -4.29457409e+09
                                          8.58888605e+09 -1.71767236e+10
1.37296355e+11 -2.74324292e+11
  5.47574907e+11 -1.09085511e+12
                                           2.16453140e+12 -4.26034751e+12
  8.24583389e+12 -1.53922233e+13
                                          2.63866685e+13 -3.51822246e+13]
```

表 3 追赶法结果与精确值的比较

阶数 n	∞-范数 x - x * ∞
10	2.84×10^{-14}
30	2.98×10^{-8}
100	3.52×10^{13}

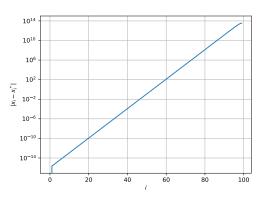


图 5 追赶法结果

图 6 追赶法 n = 100 时结果每个坐标与精确 值的差值绝对值



6 Jacobi 迭代法

```
class JacobiSolver(IterativeSolver):
    Jacobi 迭代法
    11 11 11
    def solve(self):
        x = np.copy(self.x0)
        A = np.copy(self.A)
        b = np.copy(self.b)
        xnorm = np.inf
        it.ernum = 0
        while self.iter_manager(xnorm, iternum):
            newx = np.empty(self.n)
            for i in range(self.n):
                 s = 0
                 for j in range(self.n):
                     if j == i:
                         continue
                 s += A[i][j] * x[j]
newx[i] = (b[i] - s) / A[i][i]
            xnorm = np.linalg.norm(newx - x, np.inf)
            x = newx
            iternum += 1
        print(" 迭代次数: {}".format(iternum))
        return x
```

```
迭代次数: 159
Jacobi 迭代法: n=10
                                   inf-norm=5.590596764126765e-05
[1. 1.00000003 1.00000006 1.00000035 1.00000065 1.00000307 1.00000481 1.00001875 1.00002288 1.00005591]
迭代次数: 526
Jacobi 迭代法: n=30
                                   inf-norm=3.96780056398649e-05
 0.9999999 0.9999999 0.99999996 0.99999993 0.99999972 0.99999956
 0.99999824 0.99999736 0.99999028 0.99998695 0.99996032 0.999964731
迭代次数: 1000
Jacobi 迭代法: n=100
                                     inf-norm=164270699135404.9
                                              1.00000000e+00 1.0000000e+00
1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00
                        1.00000000e+00
[ 1.00000000e+00
   1.00000000e+00
1.00000000e+00
                        1.00000000e+00
1.00000000e+00
   1.00000000e+00
                        1.00000000e+00
                                               1.00000000e+00
                                                                     1.00000000e+00
   1.00000000e+00
1.00000000e+00
                        1.00000000e+00
9.99999999e-01
                                               1.00000000e+00
1.00000000e+00
   1.00000000e+00
                         9.99999990e-01
                                               1.00000000e+00
                                                                     9.99999960e-01
   1.00000000e+00
1.00000000e+00
                        9.99999841e-01
9.99997457e-01
                                               1.00000000e+00
                                               1.00000000e+00
                                                                     9.99989827e-01
                        9.99959310e-01
9.99348958e-01
                                                                     9.99837240e-01
9.97395833e-01
   1.00000000e+00
                                               1.00000000e+00
   9.99999998e-01
                         9.89583330e-01
                                               9.99999984e-01
                                                                     9.58333307e-01
   9.99999878e-01
                       8.33333129e-01
-1.66667876e+00
                                               9.99999058e-01 3.33331753e-01
9.99945490e-01 -9.66675830e+00
   9.99992800e-01
   9.99590932e-01
                       -4.16673545e+01
                                               9.96954343e-01 -1.69671787e+02
  9.77488166e-01
-2.05590268e-01
                       -6.81704497e+02
-1.09236892e+04
                                              8.34728890e-01 -2.72994420e+03
-7.74050676e+00 -4.37043123e+04
  -6.19921240e+01 -1.74867068e+05 -4.50325166e+02 -6.99803623e+05
 -3.21374724e+03 -2.80156210e+06 -2.27610092e+04 -1.12226838e+07 -1.60165748e+05 -4.50050936e+07 -1.11954274e+06 -1.80809861e+08
  -7.76799062e+06 -7.28639006e+08 -5.34436267e+07 -2.95106032e+09
 -3.63968007e+08 -1.20473284e+10 -2.44716291e+09 -4.977433277e+10 -1.61761911e+10 -2.09109892e+11 -1.04402949e+11 -8.96432177e+11
  -6.50054534e+11 -3.90993802e+12 -3.81532093e+12 -1.69730829e+13
  -2.00233770e+13 -6.76384210e+13 -7.93116926e+13 -1.64270699e+14]
```

表 4 Jacobi 迭代法结果与精确值的比较

阶数 n	迭代次数 T	∞-范数 $\ x - x^*\ _{\infty}$
10	159	5.59×10^{-5}
30	526	3.97×10^{-5}
100	1000	1.64×10^{14}

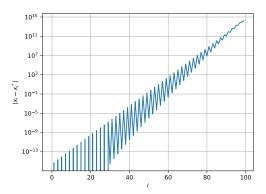


图 8 Jacobi 迭代法 n = 100 时结果每个坐标与精确值的差值绝对值

图 7 Jacobi 迭代法结果



7 Guass-Seidel 迭代法

```
class GuassSeidelSolver(IterativeSolver):
    Guass-Seidel 迭代法
    11 11 11
    def solve(self):
        x = np.copy(self.x0)
        A = np.copy(self.A)
        b = np.copy(self.b)
        xnorm = np.inf
        iternum = 0
        while self.iter_manager(xnorm, iternum):
            newx = np.empty(self.n)
            for i in range(self.n):
                s = 0
                for j in range(i):
                    s += A[i][j] * newx[j]
                for j in range(i + 1, self.n):
                    s += A[i][j] * x[j]
                newx[i] = (b[i] - s) / A[i][i]
            xnorm = np.linalg.norm(newx - x, np.inf)
            x = newx
            iternum += 1
        print(" 迭代次数: {}".format(iternum))
        return x
```

```
Guass-Seidel 迭代法: n=10
                                         inf-norm=0.0004310713408720579
[1.00000009 0.99999955 1.00000161 0.99999504 1.00001381 0.99996466 1.00008309 0.99982337 1.0003233 0.99956893]
迭代次数: 208
Guass-Seidel 迭代法: n=30
                                         inf-norm=0.000689943484630029
 1. 1. 1. 0.9999999 1.0000002 0.9999999
1.00000014 0.99999964 1.00000089 0.9999978 1.00000535 0.99998725
 1.00002966 0.99993317 1.00014405 0.99970992 1.00051746 0.99931006]
Guass-Seidel 迭代法: n=100
                                          inf-norm=93824992236886.06
1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00
                                            1.00000000e+00
1.00000000e+00
                                                                1.00000000e+00
1.00000000e+00
                                            1.00000000e+00
                                                                 1.00000000e+00
  1 000000000e+00
                       1.00000000e+00
                                            1.00000000e+00
                                                                 1.00000000e+00
                       1.00000000e+00
  1.00000000e+00
                       9.9999999e-01
                                            1.00000000e+00
                                                                 9.99999998e-01
  1 000000000+00
                       9.99999990e-01
                                            1 00000002e+00
                                                                 9 99999960e-01
  1.00000127e+00
                       9.99997457e-01
                                            1.00000509e+00
                                                                 9.99989827e-01
  1.00002035e+00
                       9.99959310e-01
                                            1.00008138e+00
                                                                 9.99837240e-01
  1.00032552e+00
                       9.99348958e-01
                                            1.00130208e+00
                                                                 9.97395833e-01
  1.00520833e+00
                       9.89583333e-01
                                            1.02083333e+00
                                                                 9.58333333e-01
                      8.33333333e-01
-1.66666667e+00
                                            1.33333333e+00
6.33333333e+00
  1.08333333e+00
                                                                 3.3333333e-01
  2.33333333e+00
                                                                -9.66666667e+00
  2.23333333e+01 -4.16666667e+01
                                            8.63333333e+01 -1.69666667e+02
  3.42333333e+02
5.46233333e+03
                     -6.81666667e+02
-1.09216667e+04
                                            1.36633333e+03 -2.72966667e+03
2.18463333e+04 -4.36896667e+04
  8.73823333e+04 -1.74761667e+05
                                            3.49526333e+05 -6.99049665e+05
  1.39810233e+06 -2.79620165e+06
2.23696210e+07 -4.47392363e+07
                                            5.59240625e+06 -1.11848093e+07
8.94784650e+07 -1.78956884e+08
  3.57913601e+08 -7.15826516e+08
                                            1.43165031e+09 -2.86328968e+09
                                            2.29050941e+10 -4.58073921e+10
3.66145962e+11 -7.31576096e+11
  9.16035994e+10 -1.83162459e+11
  1.46028888e+12 -2.90912452e+12
2.19902326e+13 -4.10484341e+13
                                            5.77243605e+12 -1.13616202e+13
7.03687442e+13 -9.38249922e+13]
```

表 5 Guass-Seidel 迭代法结果与精确值比较

阶数 n	迭代次数 T	∞-范数 x - x * ∞
10	52	0.000431
30	208	0.000690
100	722	9.38×10^{13}

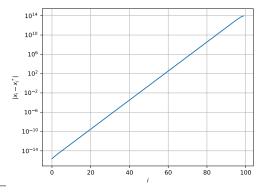


图 9 Guass-Seidel 迭代法结果

图 **10** Guass–Seidel 迭代法 n = 100 时结果 每个坐标与精确值的差值绝对值



蓝色部分为 Guass-Seidel 迭代法相较于 Jacobi 迭代法修改的部分。

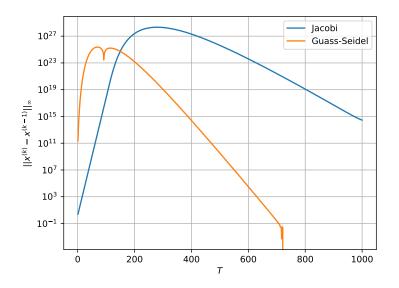


图 11 n = 100, Jacobi 迭代法和 Guass–Seidel 迭代法 $||x^{(k)} - x^{(k-1)}||_{\infty}$ 与迭代次数 T 的关系

8 结果比较与总结

首先,分析问题本身的情况。讨论在 n = 10, 30, 100 时矩阵 A 的条件数,有表 6 的结果。

表 6 矩阵条件数

n	cond(A)
10	1727.556025
30	1.83702×10^9
100	2.95670×10^{16}

可以看到这个问题本身就是病态的,在 n = 30,100 时已经非常病态,容易产生累积误差,也就越难得到方程组准确的解。

表 7 不同方法结果与准确值的 ∞ -范数 $||x - x^*||_{\infty}$ 比较表

n	顺序 Guass 迭代法	列主元 Guass 迭代法	追赶法	Jacobi 迭代法	Guass-Seidel 迭代法
10	2.84×10^{-14}	0.000	2.84×10^{-14}	5.59×10^{-5}	0.000431
30	2.98×10^{-8}	0.000	2.98×10^{-8}	3.97×10^{-5}	0.000690
100	3.52×10^{13}	0.183	3.52×10^{13}	1.64×10^{14}	9.38×10^{13}

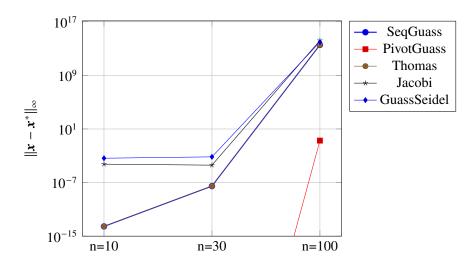


图 12 不同方法结果与准确值的 ∞-范数 ||x - x*||∞ 比较图

表 7 和图 12 展示了不同方法结果与准确值的 ∞ -范数 $\|x - x^*\|_{\infty}$ 的比较,在 n = 10, n = 30 时,这 些方法结果与准确值的误差都是可以接受的。

对于直接法(顺序 Guass 消元法、列主元 Guass 消元法、追赶法),列主元 Guass 消元法是最优的,尽可能避免了大数除以小数的现象;顺序 Guass 消元法和追赶法结果与准确解的误差一致,但是对于这种特殊的三对角矩阵,追赶法的复杂度为 O(n),比顺序 Guass 消元法、列主元 Guass 消元法的 $O(n^3)$ 下降了不少。

对于迭代法 (Jacobi 迭代法、Guass–Seidel 迭代法),图 11 显示了两种方法迭代过程,可见 Guass–Seidel 迭代法比 Jacobi 迭代法收敛更快,Jacobi 迭代法在 n=100 的 1000 轮迭代后仍未收敛。迭代法的时间复杂度为 O(Tn),T 为迭代次数。

从每种方法 n = 100 时每个坐标与精确值的差值与绝对值图来看,下标越大,累计的误差越大,呈指数增长。最开始的一些值误差仍在控制范围之内。这就意味着对于这种大型矩阵而言,应当把重要的、精度要求高的变量放在前面(比如主成分分析中的主要成分),以达到更理想的效果。

总之,对于本题,列主元 Guass 消元法的结果是最优的。

- 对于大规模矩阵, 尽可能地把重要的变量放在前面,
 - 对于三对角矩阵这种特殊情况,追赶法可能是运行速度最快的,如果是对角占优的,误差也会被控制;
 - 当不是三对角矩阵时, 为了求解大规模矩阵可以考虑迭代法更快求解;
 - 合理时间范围内更精确的精度应该超过了这几种方法的范畴, 需要进一步地根据实际问题 优化;
- 对于小规模矩阵,由于顺序 Guass 消元法误差较大且要求对角线元素不为 0、追赶法误差与顺序 Guass 消元法一致,所以一般使用列主元 Guass 消元法精确求解。