

# 计算方法大作业

李子龙

123033910195

2023 年 11 月 18 日

## 目 录

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 总结	9

## 1 问题

解方程组  $Ax = b$ ，其中

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

显然精确解为  $x^* = (1, 1, \dots, 1)^\top$ 。



## 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.float64)。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():

---

```
class Solver:
    """
    求解线性方程 Ax=b
    """
    def __init__(self, A: np.array, b: np.array):
        """
        A: n x m 矩阵
        b: n x 1 矩阵
        """
        self.A = A
        self.b = b
        assert len(self.A) == len(self.b)
        self.n = len(self.A)
        self.m = len(self.A[0])

    def solve(self):
        raise NotImplementedError
```

---

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

---

```
def construct_input(n):
    """
    构造输入
    """
    A = np.diag([6]*n, 0)+np.diag([8]*(n-1), -1)+np.diag([1]*(n-1), 1)
    b = np.array([7]+[15]*(n-2)+[14])
    return np.asarray(A, np.float64), np.asarray(b, np.float64)

def compare_output(x, target):
    return np.linalg.norm(x - target, np.inf)
```

---

对于迭代法:Jacobi 迭代法(第 6 节)和 Gauss-Seidel 迭代法(第 7 节),均派生于 IterativeSolver 类。设定迭代初值为  $\mathbf{x}_0 = (0, 0, \dots, 0)^\top$ , 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):
    """
    迭代法公共类
    """
    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.maxiter = maxiter
        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
    """
    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较

[illegible]

阶数 $n$	$\infty$ -范数 $\ \mathbf{x} - \mathbf{x}^*\ _\infty$
10	$2.84 \times 10^{-14}$
30	$2.98 \times 10^{-8}$
100	$3.52 \times 10^{13}$

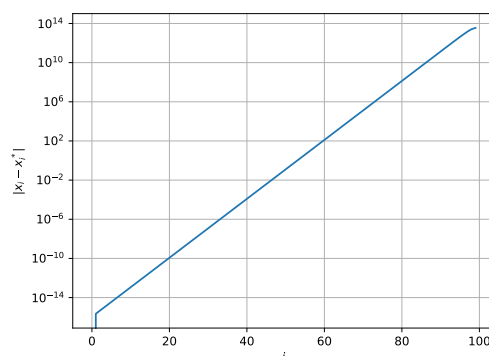


图 1 顺序 Guass 消元法结果

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
```

[illegible]

图 3 列主元 Guass 消元法结果

阶数 $n$	$\infty$ -范数 $\ \mathbf{x} - \mathbf{x}^*\ _\infty$
10	0.000
30	0.000
100	0.183

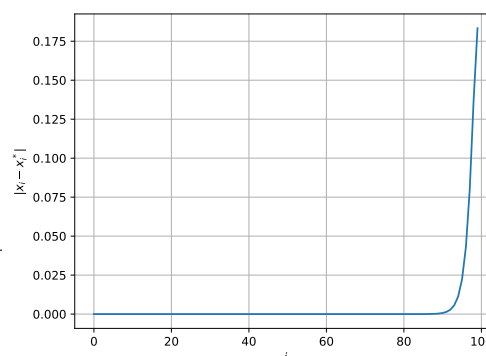


图 4 列主元 Gauss 消元法  $n = 100$  时结果每个坐标与精确值的差值绝对值

## 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
    """
    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

```

[illegible]

表 3 追赶法结果与精确值的比较

阶数 $n$	$\infty$ -范数 $\ \mathbf{x} - \mathbf{x}^*\ _\infty$
10	$2.84 \times 10^{-14}$
30	$2.98 \times 10^{-8}$
100	$3.52 \times 10^{13}$

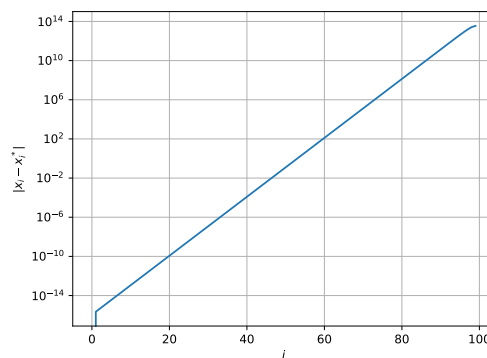


图 6 追赶法  $n = 100$  时结果每个坐标与精确值的差值绝对值

图 5 追赶法结果



## 6 Jacobi 迭代法

```
class JacobiSolver(IterativeSolver):
    """
    Jacobi 迭代法
    """
    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(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
[1.      1.      1.      1.      1.      1.
 1.      1.      1.      1.      1.      1.
 1.      1.      1.      1.      1.      1.
 0.99999999 0.99999999 0.99999996 0.99999993 0.99999972 0.99999956
 0.99999824 0.99999736 0.99999028 0.99998695 0.99996032 0.99996473]
迭代次数: 1000
Jacobi 迭代法: n=100      inf-norm=1.64270699135404e-05
[ 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  9.99999998e-01
 1.00000000e+00  9.99999990e-01  1.00000000e+00  9.99999960e-01
 1.00000000e+00  9.99999841e-01  1.00000000e+00  9.99999364e-01
 1.00000000e+00  9.99997457e-01  1.00000000e+00  9.99989827e-01
 1.00000000e+00  9.99959310e-01  1.00000000e+00  9.99837240e-01
 1.00000000e+00  9.99348958e-01  1.00000000e+00  9.97395833e-01
 9.99999998e-01  9.89583330e-01  9.99999984e-01  9.58333307e-01
 9.99999878e-01  8.33333129e-01  9.99999058e-01  3.33331753e-01
 9.99992800e-01 -1.66667876e+00  9.9945490e-01 -9.66675830e+00
 9.99590932e-01 -4.16667354e+01  9.96954343e-01 -1.69671787e+02
 9.77488166e-01 -6.81704497e+02  8.34728890e-01 -2.72994420e+03
 -2.05590268e-01 -1.09236892e+04 -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 -4.97743327e+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]
```

图 7 Jacobi 迭代法结果

表 4 Jacobi 迭代法结果与精确值的比较

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

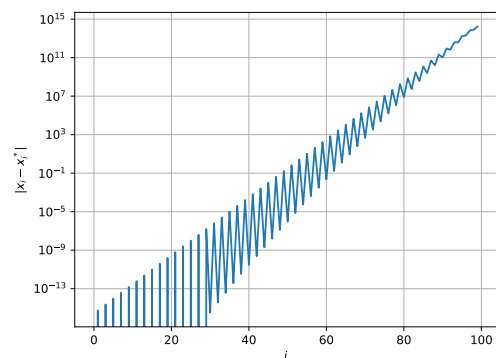


图 8 Jacobi 迭代法  $n = 100$  时结果每个坐标与精确值的差值绝对值



## 7 Guass-Seidel 迭代法

```
class GuassSeidelSolver(IterativeSolver):
    """
    Guass-Seidel 迭代法
    """
    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
```

```
迭代次数: 52
Guass-Seidel 迭代法: n=10 inf-norm=0.0004310713408720579
[1.00000009 0.99999955 1.00000161 0.99999504 1.00001381 0.99999646
1.000008309 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. 1. 1. 1.
1. 1. 1. 1. 1. 1.
1. 1. 1. 0.99999999 1.00000002 0.99999995
1.00000014 0.99999964 1.00000089 0.9999978 1.00000535 0.9998725
1.00002966 0.99993317 1.00014405 0.99970992 1.00051746 0.99931006]
迭代次数: 722
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
1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
1.00000000e+00 9.99999999e-01 1.00000000e+00 9.9999998e-01
1.00000000e+00 9.99999990e-01 1.00000002e+00 9.9999990e-01
1.00000008e+00 9.9999984e-01 1.00000032e+00 9.9999936e-01
1.00000127e+00 9.99997457e-01 1.00000509e+00 9.9998872e-01
1.00020235e+00 9.9995310e-01 1.00008138e+00 9.99837240e-01
1.00032552e+00 9.99348958e-01 1.00130208e+00 9.97395833e-01
1.00502833e+00 9.89583333e-01 1.02083333e+00 9.5833333e-01
1.08333333e+00 8.33333333e-01 1.33333333e+00 3.3333333e-01
2.33333333e+00 -1.66666667e+00 6.33333333e+00 -9.6666667e+02
2.23333333e+00 -4.16666667e+01 8.63333333e+01 -1.6966667e+02
3.42333333e+02 -6.81666667e+02 1.36633333e+03 -2.7266667e+03
5.46233333e+03 -1.09216667e+04 2.18463333e+04 -4.36896667e+04
8.73823333e+04 -1.74761667e+05 3.49526333e+05 -6.99049665e+05
1.39810233e+06 -2.79620156e+06 5.59240625e+06 -1.1848093e+07
2.23696210e+07 -4.74392363e+07 8.94784650e+07 -1.78956884e+08
3.57913601e+08 -7.1582651e+08 8.14365031e+09 -2.8632898e+09
5.7265368e+09 -1.8452896e+10 2.29059441e+10 -7.30873921e+10
9.16035994e+10 -1.8316245e+11 3.66145962e+11 -7.3157096e+11
1.46028888e+12 -2.9012452e+12 5.77243605e+12 -9.13616202e+13
2.19902326e+13 -4.10484341e+13 7.03687442e+13 -1.3244992e+13]
```

表 5 Gauss-Seidel 迭代法结果与精确值比较

阶数 $n$	迭代次数 $T$	$\infty$ -范数 $\ \mathbf{x} - \mathbf{x}^*\ _\infty$
10	52	0.000431
30	208	0.000690
100	722	$9.38 \times 10^{13}$

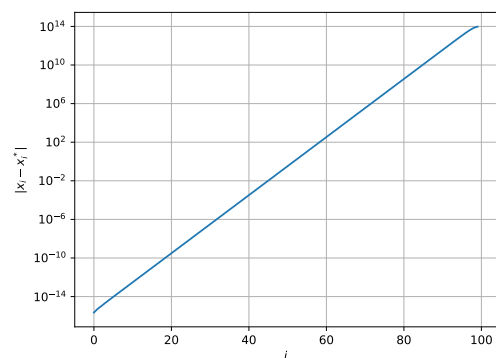


图 10 Guass-Seidel 迭代法  $n = 100$  时结果每个坐标与精确值的差值绝对值

图 9 Guass-Seidel 迭代法结果



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

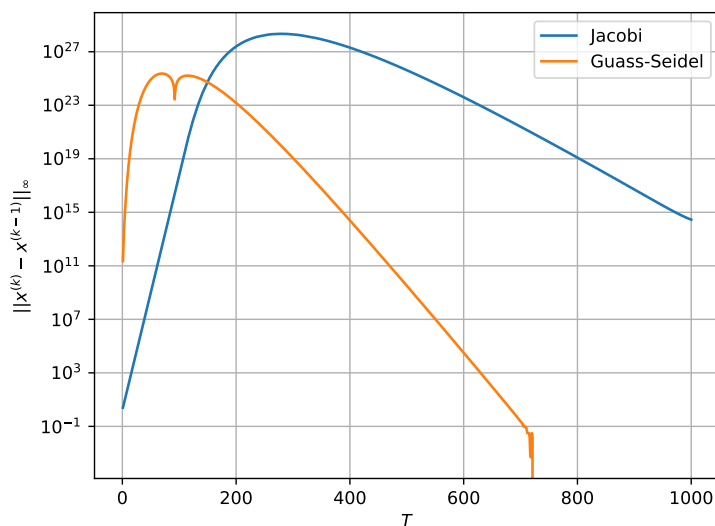


图 11  $n = 100$ , Jacobi 迭代法和 Guass-Seidel 迭代法  $\|x^{(k)} - x^{(k-1)}\|_{\infty}$  与迭代次数  $T$  的关系

## 8 总结

表 6 不同方法结果与准确值的  $\infty$ -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	$2.84 \times 10^{-14}$	0.000	$2.84 \times 10^{-14}$	$5.59 \times 10^{-5}$	0.000431
30	$2.98 \times 10^{-8}$	0.000	$2.98 \times 10^{-8}$	$3.97 \times 10^{-5}$	0.000690
100	$3.52 \times 10^{13}$	0.183	$3.52 \times 10^{13}$	$1.64 \times 10^{14}$	$9.38 \times 10^{13}$

表 6 和图 12 展示了不同方法结果与准确值的  $\infty$ -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

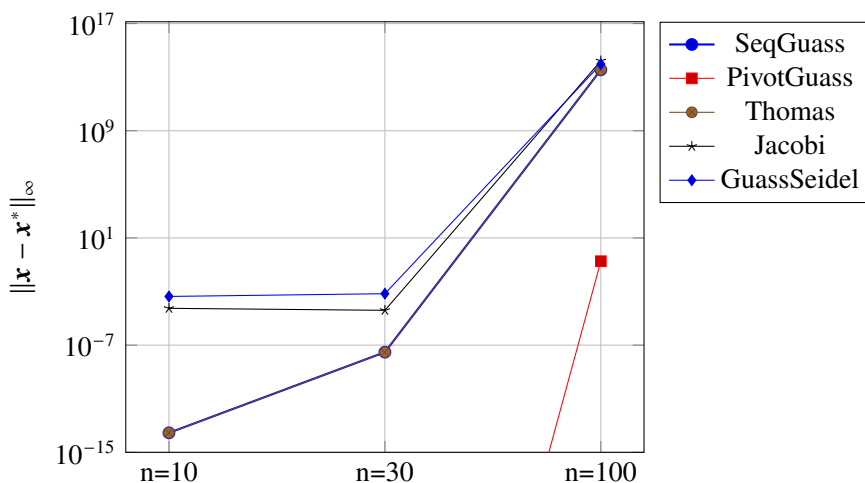


图 12 不同方法结果与准确值的  $\infty$ -范数  $\|x - x^*\|_\infty$  比较图

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。