数值代数大作业上机报告

王泽昊 1700010718

注: 本报告所有记号沿用作业里的记号.

1 算法要点

1.1 MAC格式

在实际运算中, 需要给出MAC格式

$$\begin{bmatrix} A & B \\ B^t & 0 \end{bmatrix} \begin{bmatrix} U \\ P \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}$$

中的矩阵 A, B 的具体形式.

我们按自然顺序将 u, v, p 写成 U, P, 其对应的 A, B 如下:

首先令

$$R = \begin{bmatrix} 3 & -1 & & & \\ -1 & 4 & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 4 & -1 \\ & & & -1 & 3 \end{bmatrix}_{N\times N},$$

$$S = \begin{bmatrix} 0 & -1 & & & \\ -1 & & \ddots & & \\ & \ddots & & -1 & \\ & & -1 & 0 & -1 \\ & & & -1 & 0 \end{bmatrix}_{(N-1)\times(N-1)},$$

$$T = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix}_{(N-1)\times N},$$

我们给出 A, B 的具体形式如下:

$$A = \frac{1}{h^2} \begin{bmatrix} I_{N-1} \otimes T + S \otimes I_N & 0 \\ 0 & R \otimes I_{N-1} + I_N \otimes S \end{bmatrix},$$

$$B = \frac{1}{h} \begin{bmatrix} T \otimes I_N \\ I_N \otimes T \end{bmatrix}.$$

1.2 限制、提升算子

在MAC格式下, 我们根据不同问题, 采用两种不同的限制、提升算子, 一种为作业里提到的方法, 另一种叙述如下:

首先令

$$J = \begin{bmatrix} 4 & 3 & 1 & & & & & & \\ & 1 & 3 & 3 & 1 & & & & \\ & & & \ddots & \ddots & & & & \\ & & & & & 1 & 3 & 3 & 1 \\ & & & & & & & 1 & 3 & 4 \end{bmatrix}_{N\times 2N}$$

$$K = \begin{bmatrix} 1 & 2 & 1 & & & & & \\ & & 1 & 2 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & & 1 & 2 & 1 \end{bmatrix}_{(N-1)\times(2N-1)}$$

令 $h=\frac{1}{2N}$, 将步长为 $2h$ 的粗网格 Ω^{2h} 的 u,v,p 提升至步长为 h 的细网格 Ω^h 上, 为以下形式:

$$\begin{bmatrix} U^h \\ P^h \end{bmatrix} = I_{2h}^h \begin{bmatrix} U^{2h} \\ P^{2h} \end{bmatrix},$$

其中 I_{2h}^h 是提升算子, 表达式如下:

$$I_{2h}^h = diag(\frac{1}{8}K^t \otimes J^t, \frac{1}{8}J^t \otimes K^t, \frac{1}{16}J^t \otimes J^t);$$

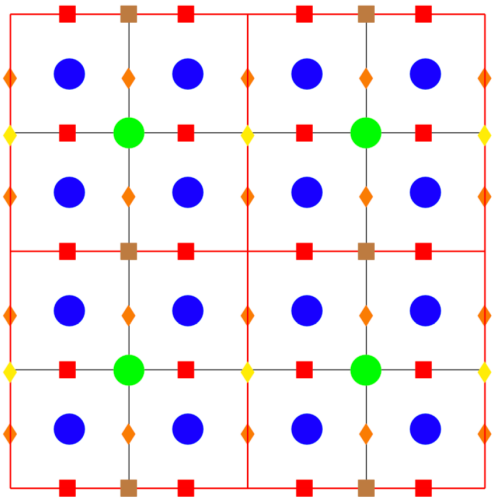
将步长为 h 的细网格 Ω^h 的 u,v,p 限制至步长为 $2h$ 的粗网格 Ω^{2h} 上, 为以下形式:

$$\begin{bmatrix} U^{2h} \\ P^{2h} \end{bmatrix} = I_h^{2h} \begin{bmatrix} U^h \\ P^h \end{bmatrix},$$

其中 I_h^{2h} 是限制算子, 表达式如下:

$$I_h^{2h} = \frac{1}{4}(I_{2h}^h)^t = diag(\frac{1}{32}K \otimes J, \frac{1}{32}J \otimes K, \frac{1}{64}J \otimes J).$$

由于大规模矩阵运算困难, 所以我们在实际运算中考虑无矩阵形式如下:



- 对于压力(p), 限制算子的作用为限制后的位置(绿色的圆)其左上、左下、右上和右下四个方向的加权平均值之平均值, 每个方向上可能有 1、2、4 个限制前的压力, 分别取权重为 1、$(3,1)$、$(9,3,3,1)$(近处权重大). 提升算子的作用为距离提升后的位置最近的四个提升前的压力之加权平均值, 权重为 $(9,3,3,1)$(近处权重大), 边界按类似比例处理.
- 对于速度, 限制算子的作用为限制后的位置其上下(u)或左右(v)方向上的加权平均值之平均值, 每个方向上可能有 3 或 6 个限制前的速度, 分别取权重为 $(1,2,1)$、$(3,6,3,1,2,1)$(近处权重大), 对于左右(u)或上下(v)边界上的速度, 不做限制全部取 0. 提升算子的作用为距离提升后的位置最近的四个或两个提升前的位置的速度之加权平均值, 权重为 $(3,3,1,1)$ 或 $(3,1)$(近处权重大), 对于左右(u)或上下(v)边界上的速度, 不做限制全部取 0.

对于作业中给出的限制提升算子, 笔者采用了显式写法, 具体形式如下:

令 $h=\frac{1}{2N}$, 将步长为 h 的细网格 Ω^h 的 u,v,p 限制至步长为 $2h$ 的粗网格 Ω^{2h} 上, 为以下形式:

$$\begin{bmatrix} U^{2h} \\ P^{2h} \end{bmatrix} = I_h^{2h} \begin{bmatrix} U^h \\ P^h \end{bmatrix},$$

其中 I_h^{2h} 是限制算子, 表达式如下:

首先令

$$I_1 = \begin{bmatrix} 1 & 0 & 1 & & & & & & \\ & & 1 & 0 & 1 & & & & \\ & & & & \cdots & \cdots & \cdots & & \\ & & & & & & 1 & 0 & 1 \end{bmatrix}_{(N-1)\times(2N-1)},$$

$$I_2 = \begin{bmatrix} 0 & 1 & 0 & & & & & \\ & & 0 & 1 & 0 & & & \\ & & & & \cdots & \cdots & \cdots & \\ & & & & 0 & 1 & 0 & \\ & & & & & & 0 & 1 & 0 \end{bmatrix}_{(N-1)\times(2N-1)},$$

$$J = \begin{bmatrix} 1 & 1 & & & & & & \\ & & 1 & 1 & & & & \\ & & & & \cdots & \cdots & & \\ & & & & & & 1 & 1 \end{bmatrix}_{N\times 2N},$$

则

$$I_h^{2h} = diag(\frac{1}{4}I_2 \otimes J + \frac{1}{8}I_1 \otimes J, \frac{1}{4}J \otimes I_2 + \frac{1}{8}J \otimes I_1, \frac{1}{4}J \otimes J);$$

将步长为 $2h$ 的粗网格 Ω^{2h} 的 u, v, p 提升至步长为 h 的细网格 Ω^h 上, 为以下形式:

$$\begin{bmatrix} U^h \\ P^h \end{bmatrix} = I_{2h}^h \begin{bmatrix} U^{2h} \\ P^{2h} \end{bmatrix},$$

其中 I_{2h}^h 是提升算子, 表达式如下:

首先令

$$I = \begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ 1 & 3 & & & \\ & 3 & \cdots & & \\ & 1 & \cdots & & \\ & & \cdots & 1 & \\ & & \cdots & 3 & \\ & & & 3 & 1 \\ & & & 2 & 3 \\ & & & & 2 \end{bmatrix}_{2N\times N},$$

$$J = \begin{bmatrix} 1 & & & \\ 2 & & & \\ 1 & 1 & & \\ & 2 & & \\ & 1 & \cdots & \\ & & \cdots & \\ & & \cdots & 1 \\ & & & 2 \\ & & & 1 \end{bmatrix}_{(2N-1)\times(N-1)},$$

$$K = \begin{bmatrix} 1 & & & \\ 1 & & & \\ & 1 & & \\ & 1 & \cdots & \\ & & \cdots & \\ & & \cdots & \\ & & & 1 \\ & & & 1 \end{bmatrix}_{2N\times N},$$

则

$$I_{2h}^h = diag(\frac{1}{4}J \otimes I, \frac{1}{4}I \otimes J, K \otimes K).$$

1.3 DGS迭代法

首先我们使用作业中给出的DGS迭代法按自然顺序逐步更新各分量, 速度较慢, 在较大规模矩阵时耗时过大(详见下一节的数值结果), 甚至有较多不收敛的情况, 笔者猜测是作业中给出的算法有一些小问题, 于是我们又参考了文献[1]中的方法给出算法如下:

Algorithm (DGS) $[u^{k+1}, p^{k+1}] \leftarrow \mathtt{DGS}(u^k, p^k, f, g)$

(1) Relax the momentum equation

$$u^{k+\frac{1}{2}} = u^k + \hat{A}^{-1}(f - Au^k - B^\intercal p^k),$$

(2) Relax the transformed mass equation

$$\delta q = \hat{A}_p^{-1}(g - Bu^{k+\frac{1}{2}}).$$

(3) Distribute the correction to the original variables

$$u^{k+1} = u^{k+\frac{1}{2}} + B^\intercal \delta q,$$
$$p^{k+1} = p^k - B(B^\intercal \delta q).$$

注: 在最初进行数值试验时, 由于作业中给出的DGS迭代法是无矩阵(Matrix Free)的, 故笔者采用隐式写法, 而此时用作业或ppt中给出的提升限制算子较难达到收敛且Cputime巨大, 故笔者参考了相关文献的方法(如上), 其他部分均采用作业中给出的方法, 较快地得到了令人较为满意的结果, 在下面的数值试验部分会给出相关数据.

1.4 Uzawa迭代法

给定 P_0, 令 $k = 0$.

1. 求解 $AU_{k+1} = F - BP_k$;
2. 更新压力 $P_{k+1} = P_k + \alpha(B^t U_{k+1})$;
3. 判断误差是否小于允许的值: 如果小于, 则停止迭代, 否则回到第一步.

关于参数 α 的选取:

因为

$$P_{k+1} = P_k + \alpha(B^t A^{-1}(F - BP_k)) = (I - \alpha B^t A^{-1} B)P_k + \alpha B^t A^{-1} F,$$

因此, 关于压力的更新方法, 相当于下面的线性方程组

$$B^t A^{-1} B P = B^t A^{-1} F$$

的迭代格式. 下面证明最优参数(即使谱半径 $\rho(I - \alpha B^t A^{-1} B)$ 最小的 α)为:

$$\alpha_* = \frac{2}{\lambda_{min}(B^t A^{-1} B) + \lambda_{max}(B^t A^{-1} B)}.$$

Proof. 记 $M = B^t A^{-1} B$.

假设 λ, v 是 M 的一组特征值和特征向量, 则有

$$Mv = \lambda v,$$

故有

$$(I - \alpha M)v = (1 - \alpha\lambda)v.$$

由上述过程可逆, 故

$$\sigma(I - \alpha M) = \{1 - \alpha\lambda | \lambda \in \sigma(M)\},$$

其中 $\sigma(A)$ 为矩阵 A 的谱集.

记 $\lambda_1 = min\, \sigma(M), \lambda_2 = max\, \sigma(M)$, 故

$$\begin{aligned} & min \quad \rho(I - \alpha M) \\ \Leftrightarrow\ & min \ \ max\{|1 - \alpha\lambda_1|, |1 - \alpha\lambda_2|\} \end{aligned} \tag{1}$$

而

$$\begin{aligned} & max\{|1-\alpha\lambda_1|, |1-\alpha\lambda_2|\} \\ \geq & \frac{|1-\alpha\lambda_1|+|1-\alpha\lambda_2|}{2} \\ \geq & |\alpha(\lambda_1-\lambda_2)| \end{aligned}$$

等号成立条件为

$$\alpha = \frac{2}{\lambda_1+\lambda_2},$$

故易知使得(1)式成立的 α 应满足

$$\alpha = \frac{2}{\lambda_1+\lambda_2}.$$

□

注: 在笔者进行最初的数值试验时, 由于采用隐式写法, 故在求解精确解步骤时改为利用一次G-S迭代法求解近似解, 即为简化版Inexact Uzawa, 也可以达到收敛, 而且收敛速度是可以接受的, Cputime大概为正版Uzawa迭代法的十倍, 在下面的数值试验部分会提到本次试验结果.

1.5 Inexact Uzawa迭代法

给定 P_0, 令 $k=0$.

1. 近似求解 $AU_{k+1} = F - BP_k$, 得到近似解 $\hat{U}_{k+1}$;
2. 更新压力 $P_{k+1} = P_k + \alpha(B^t\hat{U}_{k+1})$;
3. 判断误差是否小于允许的值: 如果小于, 则停止迭代, 否则回到第一步.

设 $\hat{U}_{k+1}$ 是方程 $AU_{k+1} = F - BP_k$ 的近似解. 定义

$$\delta_k = A\hat{U}_{k+1} - F + BP_k,$$

若

$$||\delta_k|| \leq \tau||B^t\hat{U}_k||,$$

则(适当选取 α)当 τ 充分小是, 上述迭代方法是收敛的.
注: 在实际进行数值试验时, 笔者发现利用一次G-S迭代法仅需迭代很少次数便可很快收敛, 故将此算法改为手动控制迭代次数, 经试验只需迭代3至6次即可达到较好的效果.

1.6 V-cycle

1. 在Ω^h上，以$u_h^{(0)}$为初值，用G-S 迭代法，对$A_hU_h = F_h$, 迭代ν_1次（$\nu = 1,2,3,\ldots$），得到$u_h^{(\nu_1)}$，并计算残量

$$r_h = F_h - A_h u_h^{(\nu_1)}$$

2. 将残量限制到网格Ω^{2h}上，即

$$F_{2h} = I_h^{2h} r_h$$

3. 在Ω^{2h}上，以$u_{2h}^{(0)} = 0$为初值，用G-S迭代法，对$A_{2h}u_{2h} = F_{2h}$迭代ν_1次，得到$u_{2h}^{(\nu_1)}$，并计算残量

$$r_{2h} = F_{2h} - A_{2h} u_{2h}^{(\nu_1)}$$

我们把讲义中G-S迭代法改为DGS迭代法、Uzawa迭代法和Inexact Uzawa迭代法时, 对应需要将每一步的残量改为

$$r_h = \begin{bmatrix} F_h - A_h u_h^{(\nu_1)} \\ 0 - B_h^t U_h^{(\nu_1)} \end{bmatrix},$$

对应的方程一并改为MAC格式下完整的方程

$$\begin{bmatrix} A_h & B_h \\ B_h^t & 0 \end{bmatrix} \begin{bmatrix} u_h \\ p_h \end{bmatrix} = r_h$$

2 数值试验

2.1 第一题

在本题中, 我们利用之前提到的参考文献中的算法先给出一组结果, 其中 $L = log_2(N), v_1 = 1, v_2 = 0$:

N	CPUtime(s)	V-cycle	e_N
64	0.051717	42	2.070050e-03
128	0.171350	49	5.173892e-04
256	0.833522	56	1.293396e-04
512	5.313062	63	3.233441e-05
1024	29.907339	69	8.083570e-06
2048	133.666204	75	2.020889e-06

$L = log_2(N), v_1 = 1, v_2 = 0$ 时:

N	CPUtime(s)	V-cycle	e_N
64	0.104940	22	2.070051e-03
128	0.332443	25	5.173904e-04
256	1.739755	27	1.293412e-04
512	9.946418	30	3.233529e-05
1024	48.397206	33	8.084005e-06
2048	239.132092	35	2.021253e-06

笔者经过多次试验, 发现随着 v_1, v_2 逐渐增大或 L 的逐渐减小, V-cycle次数会变小, 但随之而来的是CPUtime的增大和误差的变大, 于是笔者猜测此时取 $v_1 = 1, v_2 = 0$ 可得到较好的结果. 而当其他条件给定时, e_N 的值会随着 N 的增大而变小, 这说明网格取的越细误差越小.

对于作业中给出的方法, 结果不是很理想, 当 v_1, v_2 较小时经常不会收敛, 收敛时耗时也较大, 数值结果见下表, 其中 $L = log_2(N), v_1 = 1, v_2 = 0, N = 1024$ 和 $N = 2048$ 由于经常不收敛且耗时巨大, 所以笔者并未得出结果:

N	CPUtime(s)	V-cycle	e_N
64	1.379635581	95	0.0246424643842949
128	9.044428927	150	0.0136953186218800
256	153.4092576	587	0.00885179692240105
512	3136.175237	1397	0.00819890143493493

2.2 第二题

由于大型矩阵特征值计算较为困难, 于是笔者在进行一些估计和调试之后取出了一些合适的 α 的值进行试验, 下面是一些试验结果.

首先 $L = log_2(N), v_1 = 2, v_2 = 0, \alpha = 1$ 时:

N	CPUtime(s)	V-cycle	e_N
64	0.024305	1	2.070050e-03
128	0.091823	1	5.173892e-04
256	0.503140	1	1.293396e-04
512	2.378279	1	3.233442e-05
1024	11.023531	1	8.083582e-06
2048	59.456071	1	2.020921e-06

此时V-cycle均只需进行一次即可得到相当好的结果, 这说明了Uzawa迭代法可以在较少的V-cycle内得到很好的结果, 我们观察到此时的误差 e_N 和第一问中用DGS迭代法得到的误差很相近, 这说明这个方法并不能使误差变得更小.

接下来我们取 $L = log_2(N), v_1 = 4, v_2 = 1, \alpha = 0.8$:

N	CPUtime(s)	V-cycle	e_N
64	0.152414	3	2.070050e-03
128	0.640640	3	5.173911e-04
256	3.682733	3	1.293471e-04
512	20.261710	3	3.236448e-05
1024	97.596606	3	8.202979e-06
2048	508.400202	3	2.455340e-06

经比较可知, 之前的一组值不仅计算效率更高, 其误差 e_N 也要更小一些, 但两组误差仍然很接近, 差距大概在1e-4以内.

2.3 第三题

如之前所说, 在本题中经试验, 发现若利用G-S迭代法对Uzawa迭代法第一步求近似解时, 无需多次迭代即可达到很小的残量以达到收敛, 故笔者并未设置 τ 以作为停机标准, 而是手动选取了Uzawa第一步G-S迭代法解方程的迭代次数, 最终笔者认为迭代一次就够用了. 下面给出一些数值结果.

首先是 $v_1 = 4, v_2 = 0, \alpha = 0.2$:

N	CPUtime(s)	V-cycle	e_N
64	0.058470	27	2.070049e-03
128	0.200355	28	5.173889e-04
256	0.970071	29	1.293398e-04
512	6.201045	32	3.233487e-05
1024	29.585627	36	8.083776e-06
2048	138.963910	39	2.021030e-06

接下来是 $v_1 = 1, v_2 = 0, \alpha = 0.3$:

N	CPUtime(s)	V-cycle	e_N
64	0.071232	71	2.070050e-03
128	0.198714	73	5.173892e-04
256	0.801898	77	1.293396e-04
512	5.384371	81	3.233441e-05
1024	27.146270	86	8.083570e-06
2048	112.409590	90	2.020888e-06

观察发现此时V-cycle次数较之标准Uzawa均有些大, 而耗时更少, 误差很相近(有时会更少), 更加说明了Inexact Uzawa迭代法在计算效率中的优越性, Uzawa迭代法的效率主要取决于Uzawa迭代法中第一步的方程是否容易很快计算出精确解, 若对应的矩阵 A 是病态的, 那么Uzawa迭代法就不是很好用了.

2.4 第四题

本题在Inexact Uzawa第一步采用G-S迭代法求近似解, 利用V-cycle作为预条件子, 迭代 v 次, 下面将给出数值结果.

首先我们取 $\alpha = 1, v_1 = 1, v_2 = 0, v = 3$:

N	CPUtime(s)	Iteration time	e_N
64	0.036656	19	2.070050e-03
128	0.146046	20	5.173892e-04
256	0.692578	21	1.293396e-04
512	3.878609	22	3.233442e-05
1024	18.011342	23	8.083572e-06
2048	78.826963	24	2.020887e-06

接下来我们取 $\alpha = 1.1, v_1 = 1, v_2 = 0, v = 5$:

N	CPUtime(s)	Iteration time	e_N
64	0.043904	13	2.070050e-03
128	0.163506	14	5.173892e-04
256	0.614274	15	1.293396e-04
512	3.981875	15	3.233442e-05
1024	18.428674	15	8.083572e-06
2048	79.119895	15	2.020890e-06

在此时, 误差 e_N 和CPUtime已经很相近, 总的来讲比之前方法的效果要好一些.

3 总结

在所有(正确)方法中, 最终所达的误差 e_N 都很接近, 不同方法或不同参数所得到的 e_N 之间的差距基本都在1e-4以内, 主要差别在于循环(迭代)次数和CPUtime, 另外根据矩阵的病态程度, 应该选取不同方法, 若矩阵过于病态, 则Uzawa迭代法中的第一步就较难实现, 此时考虑近似解效率更高, 但在容易求解的时候, Uzawa迭代法的结果和效率都更好一些.

4 参考文献

[1]L Chen. Programming of MAC Scheme for Stokes Equations

[2]M Wang, L Chen. Multigrid Methods for the Stokes Equations using Distributive Gauss – Seidel Relaxations based on the Least Squares Commutator

注:本次作业与几位同学有进行相关讨论, 例如参考文献的查找以及错误更正, 和基本算法的改进.