数值代数　大作业

李佳 2100010793

1　问题描述

考虑 Stokes 方程

$$\begin{cases} -\Delta\vec{u}+\nabla p=\vec{F},\ (x,y)\in(0,1)\times(0,1), \\ \qquad div\ \vec{u}=0,\ (x,y)\in(0,1)\times(0,1), \end{cases}$$

条件为

$$\begin{aligned} \frac{\partial u}{\partial\vec{n}}=b,\ y=0, \quad & \frac{\partial u}{\partial\vec{n}}=t,\ y=1, \\ \frac{\partial v}{\partial\vec{n}}=l,\ x=0, \quad & \frac{\partial v}{\partial\vec{n}}=r,\ x=1, \\ u=0,\ x=0,1, \quad & v=0,\ y=0,1, \end{aligned}$$

其中 $\vec{u}=(u,v)$ 为速度, p 为压力, $\vec{F}=(f,g)$ 为外力, $\vec{n}$ 为外法向方向.

利用交错网格上的 MAC 格式离散 Stokes 方程, 可得到如下线性方程组

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}\begin{pmatrix} U \\ P \end{pmatrix}=\begin{pmatrix} F \\ 0 \end{pmatrix}. \tag{$*$}$$

1. 分别取 $N=64,128,256,512,1024,2048$, 以 DGS 为磨光子, 用基于 V-cycle 的多重网格方法求解离散问题 $(*)$, 停机标准为 $\|r_h\|_2/\|r_0\|_2\le 10^{-8}$, 对不同的 ν_1,ν_2,L, 比较 V-cycle 的次数和 CPU 时间, 并计算误差

$$e_N=h(\sum_{j=1}^{N}\sum_{i=1}^{N-1}|u_{i,j-\frac{1}{2}}-u(x_i,y_{j-\frac{1}{2}})|^2+\sum_{j=1}^{N-1}\sum_{i=1}^{N}|v_{i-\frac{1}{2},j}-v(x_{i-\frac{1}{2}},y_j)|^2)^{\frac{1}{2}}.$$

2. 分别取 $N=64,128,256,512$, 以 Uzawa Iteration Method 求解离散问题 $(*)$, 停机标准为 $\|r_h\|_2/\|r_0\|_2\le 10^{-8}$, 并计算误差

$$e_N=h(\sum_{j=1}^{N}\sum_{i=1}^{N-1}|u_{i,j-\frac{1}{2}}-u(x_i,y_{j-\frac{1}{2}})|^2+\sum_{j=1}^{N-1}\sum_{i=1}^{N}|v_{i-\frac{1}{2},j}-v(x_{i-\frac{1}{2}},y_j)|^2)^{\frac{1}{2}}.$$

3. 分别取 $N=64,128,256,512,1024,2048$, 以 Inexact Uzawa Iteration Method 为迭代法求解离散问题 $(*)$, 停机标准为 $\|r_h\|_2/\|r_0\|_2\le 10^{-8}$, 其中以 V-cycle 多重网格方法为预条件子求解每一步的子问题 $AU_{k+1}=F-BP_k$, 对不同的 $\alpha,\tau,\nu_1,\nu_2,L$, 比较外循环

的迭代次数和 CPU 时间, 并计算误差

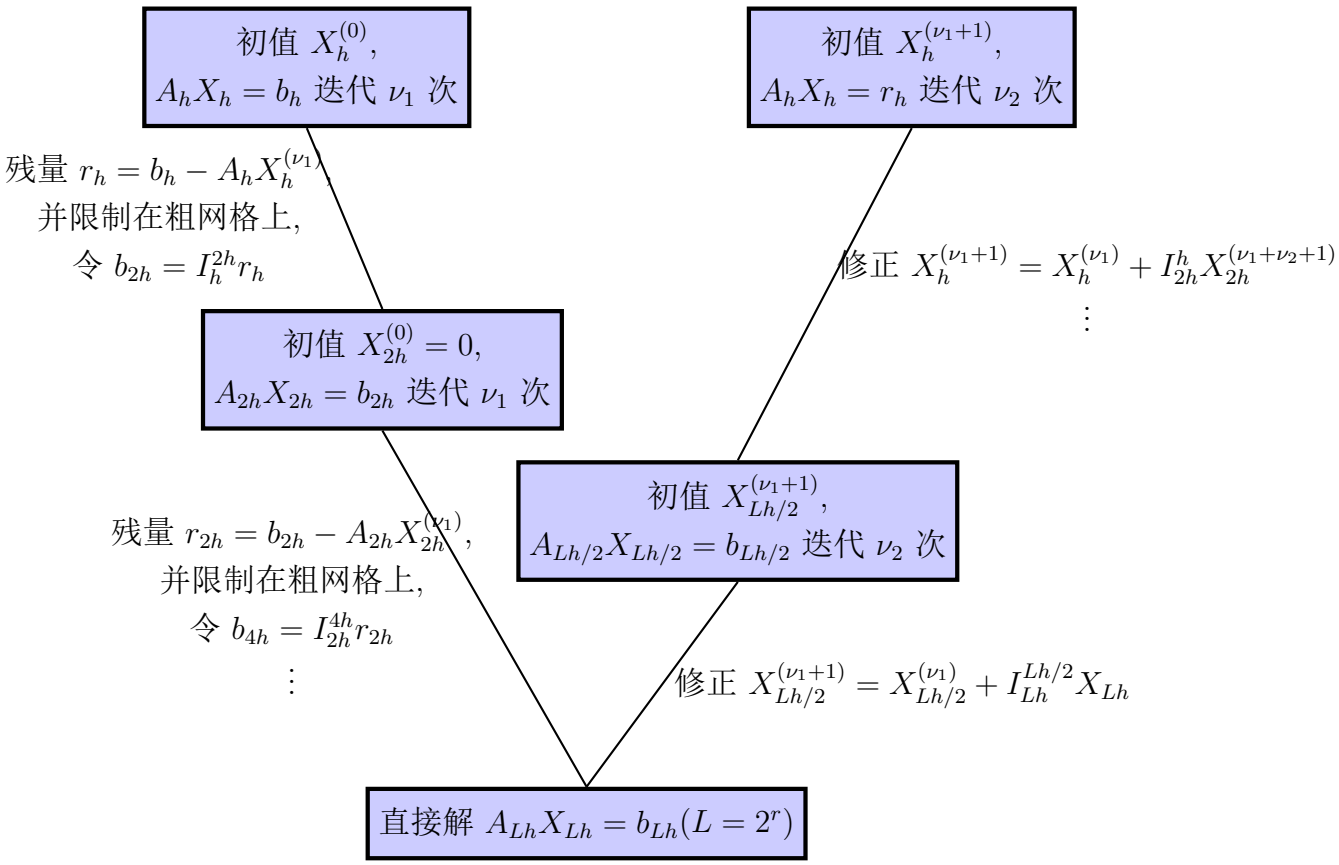
$$e_N = h(\sum_{j=1}^{N}\sum_{i=1}^{N-1}|u_{i,j-\frac{1}{2}} - u(x_i, y_{j-\frac{1}{2}})|^2 + \sum_{j=1}^{N-1}\sum_{i=1}^{N}|v_{i-\frac{1}{2},j} - v(x_{i-\frac{1}{2}}, y_j)|^2)^{\frac{1}{2}}.$$

2 数值方法

2.1 V-cycle 多重网格方法

V-cycle 多重网格方法是求解椭圆偏微分方程基于网格的离散方程的高效方法, 其思想来源于 Guass-Seidel 迭代法对此类问题的磨光性. 迭代法的磨光性使得误差中高频震荡的分量快速衰减, 而低频分量衰减较慢. 而只要将低频分量限制在粗网格上, 低频分量也成为相对高频, 从而容易在粗网格上消去.

这一思想同样适用于一部分其它的问题, 其一般的方法是选择合适的磨光子、限制算子、提升算子, 在细网格上进行若干次磨光, 将残量方程限制到粗网格上, 递归使用该方法求近似解, 再将误差提升到细网格上, 对解进行修正, 再做若干次磨光, 即完成了一次 V-cycle(如图所示).



2.2 DGS 迭代法

由于求解的线性方程系数矩阵是对称不定的, 此时难以保证 Guass-Seidel 迭代法的磨光性, 因此使用为 Stokes 方程的多重网格方法求解量身定做的 Distributive Guass Seidel 迭代法 (A. Brandt & N. Dinar 1979):

1. 用 Guass-Seidel 迭代法更新速度分量 (对应于线性方程 $AU = F - BP$.);

2. 依次对网格内 N^2 个单元格计算散度方程的残量, 更新与该单元相关的速度、压力 (如图所示).

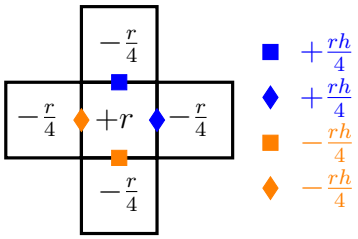


图 1. 内部单元

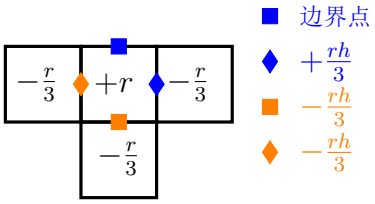


图 2. 边界单元

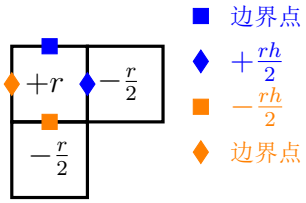


图 3. 顶点单元

2.3 Uzawa 迭代法

Uzawa 迭代法 (Hirofumi Uzawa 1958) 是求解鞍点问题的一类有效方法. 它的思想是将原线性方程转化为两组方程

$$\begin{cases} AU = F - BP \\ B^T U = O \end{cases}$$

第一组方程系数矩阵对称正定, 相对容易获得精确解 (如平方根法、共轭梯度法), 之后再对压力进行一次更新. 一次 Uzawa 迭代过程如下:

1. 求解 $AU_{k+1} = F - BP_k$(可以是精确解, 可以是近似解);

2. 更新压力 $P_{k+1} = P_k + \alpha(B^T U_{k+1})$.

2.4 共轭梯度法

共轭梯度法是求解对称正定系数矩阵的线性方程的有效方法. 其思想是将线性方程求解转化为一个函数的最值问题. 要找“下山”方向, 共轭梯度法不仅希望在当前下降最快的方向 (即梯度的反方向) 寻找, 还希望在前一步的下降方向上寻找, 也就是在两个方向张成的平面上寻找下山落点. 如果对梯度方向进行优化, 即将残量方程的近似解作为梯度方向的一个优化方向, 则相当于对问题做了预优, 可以减少迭代次数, 提高求解速度.

3 理论分析结果

本上机报告中称 ν_1 为" 前磨光次数", ν_2 为" 后磨光次数"; $r+1$ 为" 网格层数"($L=2^r$, 底层网格单位长度为 Lh); τ 为" 近似解精度要求".

1. (1) 随着前后磨光次数增加, 每次 V-cycle 的残量理论上应该更小, 故次数减少; CPU 时间同时受磨光次数和 V-cycle 次数影响, 但随着磨光次数增加, 磨光本身的效果理论上减弱, 因此在磨光次数较大时, CPU 时间有可能增大.

(2) 随着 L 减小, 等价于网格层数减小, 每次 V-cycle 后的残量理论上应当增加, 因此 V-cycle 次数应增大; 网格层数的减少虽然减少了每次循环的时间, 但也导致低频分量难以快速消去, 因此总的 CPU 时间可能随之增加.

(3) 随着网格规模 N 增大, 理论上得到的计算解与真解的误差减小.

2. 在相同的网格规模 N、相同的初值的情况下, Exact Uzawa 迭代法和 V-cycle 多重网格方法在停机标准下对残量有相同的要求, 理论上误差应当差别不大; 并且随着网格规模 N 增大, 理论上得到的计算解与真解的误差减小.

3. (1) 注意到 Uzawa 迭代法实际上是迭代格式

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

计算可知

$$B^TA^{-1}B=I-\frac{1}{N^2}\mathbf{1}\cdot\mathbf{1}^T,$$

$$\mathbf{1}=(1,1,...,1)^T_{1\times N^2}.$$

由于 $\mathbf{1}\cdot\mathbf{1}^T$ 与 $\mathbf{1}^T\cdot\mathbf{1}=N^2$ 有共同非零特征值, 可知 $B^TA^{-1}B$ 的全体特征值为 $1(N^2-1$ 重),$0(1$ 重), 从而 $I-\alpha B^TA^{-1}B$ 的特征值为 $1-\alpha$ (N^2-1 重), $1(1$ 重).

由此, 可知理论上 $0\le\alpha\le 2$ 时才能保证 $\rho(I-\alpha B^TA^{-1}B)\le 1$. 事实上此时等于 1. 尽管谱半径并不严格小于 1, 但当 $0<\alpha<2$ 时, 可以证明迭代法仍是收敛的. 设 $B^TA^{-1}B$ 特征值 0 对应的特征子空间为 $V_0=span\{\mathbf{1}\}$, 特征值 1 对应特征子空间 V_1, $V_0\oplus V_1=\mathbb{R}^{N^2}$ 且 $V_0\perp V_1$. 于是迭代矩阵的特征值 1 对应特征子空间 V_0, 特征值 $1-\alpha\neq 1$ 对应特征子空间 V_1. 考虑真解 $P=P^0+P^1, P^i\in V_i$, 满足方程

$$B^TA^{-1}(F-BP)=O,$$

可知 $B^TA^{-1}F=B^TA^{-1}BP=P^1\in V_1$. 设迭代得到的序列为 $\{P_k\}_{k\in\mathbb{N}}$, 同理可知 $B^TA^{-1}BP_k\in V_1$. 设 $\delta_k=P_k-P_{k-1}, k\in\mathbb{N}$, 则

$$\delta_1=-\alpha(B^TA^{-1}BP_0-B^TA^{-1}F)\in V_1,$$

$$\Rightarrow\delta_k=(I-\alpha B^TA^{-1}B)\delta_{k-1}=...=(I-\alpha B^TA^{-1}B)^{k-1}\delta_1=(1-\alpha)^{k-1}\delta_1,$$

由 $|1-\alpha|<1$, 可知 $\sum_{k=1}^{\infty}\delta_k$ 收敛, 故迭代法收敛.

由上述分析也可看出, 迭代法真正的" 谱半径" 应当是 $|1-\alpha|$. 因此理论上 α 越靠近 1, 迭代法收敛更快, 外循环次数越少, CPU 时间越少.

(2) 近似解精度要求 τ 越大, 近似解与真解偏差越大, 可能导致外循环迭代次数增大; CPU 时间同时与 τ 和外循环次数有关, 降低近似解精度可以减少每一步共轭梯度法的时间, 有利于降低 CPU 时间, 而外循环次数随之的增加会提高 CPU 时间, 故 CPU 时间随 τ 的变化不便预测.

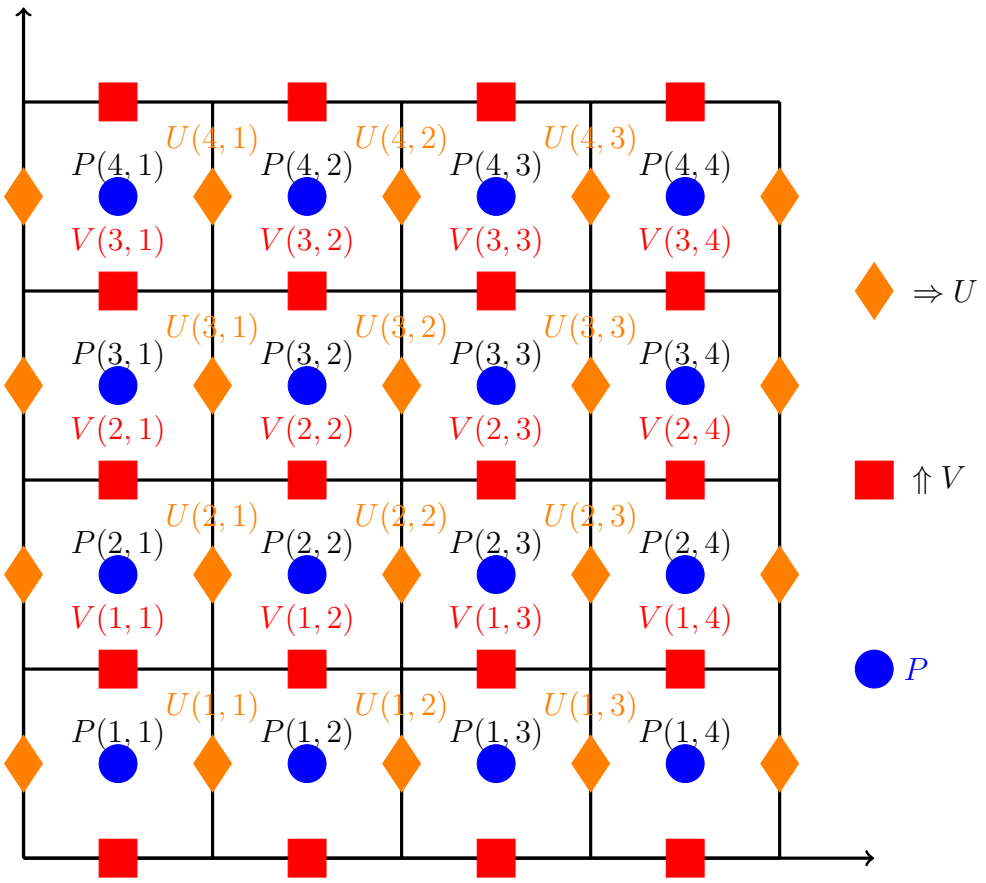
(3) 前后磨光次数越多, 一次 V-cycle 残量减小, 但由第 1 问的分析可知达到精度要求的效率可能降低, 因此 CPU 时间可能增加; 外循环迭代次数理论上只与 α, τ 有关, 因此外循环次数理论上与磨光次数关系不大.

(4) 随着 L 减小, 等价于网格层数减小, 一次 V-cycle 残量增大, 达到精度要求的效率可能降低, CPU 时间理论上应该增加; 外循环迭代次数理论上只与 α, τ 有关, 因此理论上与网格层数关系不大.

(5) 在相同的网格规模 N、相同的初值的情况下, Inexact Uzawa 迭代法和 Exact Uzawa 迭代法、V-cycle 多重网格方法在停机标准下对残量有相同的要求, 理论上误差应当差别不大; 并且随着网格规模 N 增大, 理论上得到的计算解与真解的误差减小.

4 具体算法实现

4.1 MAC 交错网格下实现迭代法



对问题中的未知量 U, V, P 进行如图所示的编号. 具体实现中, 由于矩阵是稀疏的, 因此各种矩阵 -向量乘法、DGS、GS 迭代法均采取分量形式, 迭代法每一步更新的顺序按编号顺序进行 (例如上图中, 对 U 的更新按照 $U(1,1) - U(1,2) - U(1,3) - U(2,1) -$

... $-U(4,3)$ 进行).

DGS 迭代法按照如下方式实现:

Algorithm 1: DGS 迭代法

Input: 初值 U,V,P, 常数项 F,G,D

Output: 迭代 1 次后的 [U,V,P]=DGS(U,V,P,F,G,D)

G-S 迭代分量形式更新速度: $[U,V]^T=(D_A-L_A)^{-1}(L_A^T[U,V]^T+[F,G]^T-BP)$;

for *i=1:N* **do**

  for *j=1:N* **do**

    if (i,j) 是内部单元 **then**

      $r=D(i,j)-N(U(i,j)-U(i,j-1)+V(i,j)-V(i-1,j))$;

      $d=r/(4N)$;

      $U(i,j) = U(i,j)+d$; $V(i,j) = V(i,j)+d$;

      $U(i,j-1)=U(i,j-1)-d$; $V(i-1,j)=V(i-1,j)-d$;

      $P(i,j) = P(i,j)+r$;

      $P(i,j-1)=P(i,j-1)-r/4$; $P(i,j+1)=P(i,j+1)-r/4$;

      $P(i+1,j)=P(i+1,j)-r/4$; $P(i-1,j)=P(i-1,j)-r/4$;

    end

    if (i,j) 是边界单元 *(以 2.2 节图 2 所示为例)* **then**

      $r=D(i,j)-N(U(i,j)-U(i,j-1)-V(i-1,j))$;

      $d=r/(3N)$;

      $U(i,j) = U(i,j)+d$;

      $U(i,j-1)=U(i,j-1)-d$; $V(i-1,j)=V(i-1,j)-d$;

      $P(i,j) = P(i,j)+r$;

      $P(i,j-1)=P(i,j-1)-r/3$; $P(i,j+1)=P(i,j+1)-r/3$;

      $P(i-1,j)=P(i-1,j)-r/3$;

    end

    if (i,j) 是顶点单元 *(以 2.2 节图 3 所示为例)* **then**

      $r=D(i,j)-N(U(i,j)-V(i-1,j))$;

      $d=r/(2N)$;

      $U(i,j) = U(i,j)+d$; $V(i-1,j)=V(i-1,j)-d$;

      $P(i,j) = P(i,j)+r$;

      $P(i,j+1)=P(i,j+1)-r/2$; $P(i-1,j)=P(i-1,j)-r/2$;

    end

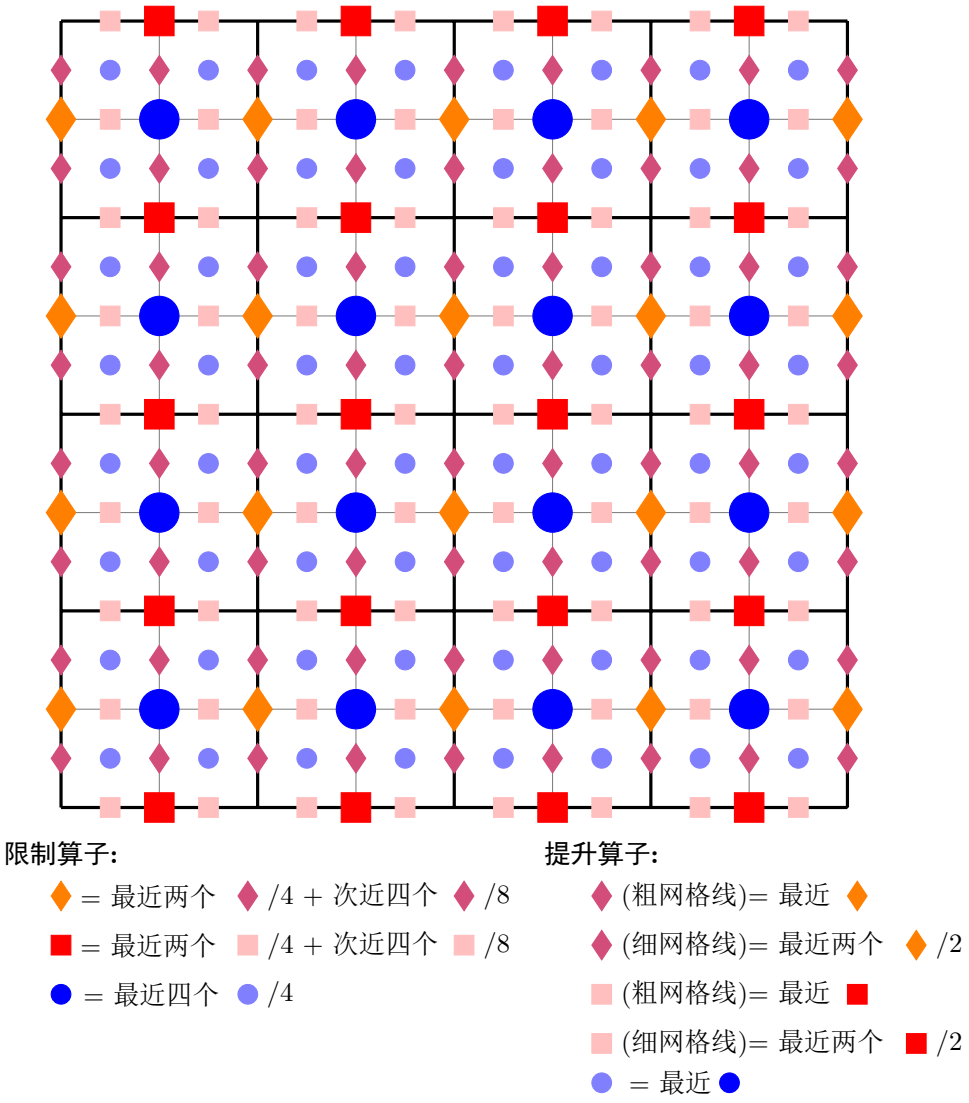
  end

end

4.2 多重网格方法

交错网格形式下的多重网格方法采用的提升、限制算子如图所示：

Figure 1: 提升算子与限制算子的选取



记提升算子、限制算子分别为函数 prolongation, restriction. 交错网格下的 V-cycle 多重网格方法按如下方式实现 (磨光子 smoother 在第一问中取为 DGS, 在第三问中取为 GS):

Algorithm 2: V-cycle 多重网格方法

Input: V-cycle 层数 r, 前磨光次数 ν_1, 后磨光次数 ν_2, 初值 X, 常数项 b, 磨光子 smoother

Output: 迭代一次后的解 X=Vcycle$(r, \nu_1, \nu_2, X, b$,smoother)

if $r == 0$ **then**
  $X =$ 真解;
  (由于题中矩阵实际上是奇异的, 因此实际实现时求的是近似解, 即用磨光子磨光 $\nu_1 + \nu_2$ 次)
else
  for $i = 1 : \nu_1$ **do**
    $X = \text{smoother}(X, b)$;
  end
  $res = b - AX$; $r_0 = \text{restriction}(res)$;
  $e_0 = \text{Vcycle}(r-1, \nu_1, \nu_2, O, r_0, smoother)$; $e = \text{prolongation}(e_0)$;
  $X = X + e$;
  for $i = 1 : \nu_2$ **do**
    $X = \text{smoother}(X, b)$;
  end
end

4.3 Uzawa 迭代法

Uzawa 迭代法的实现在数值方法 (2.3 节) 已经写明. 下面给出 Inexact Uzawa 迭代法中, 以 V-cycle 为预条件子, 用预优共轭梯度法 (PCG) 求解子问题 $AU_{k+1} = F - BP_k$ 的具体实现:

Algorithm 3: 以 V-cycle 为预条件子的共轭梯度法

Input: 近似解精度要求 τ, 前后磨光次数 ν_1, ν_2,V-cycle 层数 r, 上一步散度残量 r_0, 初值 X, 常数项 b

Output: 近似解 X=PCG$(\tau, \nu_1, \nu_2, r, r_0, X, b)$

$k = 0$;
$res = b - AX$;
while $\|res\|_2 > \tau\|r_0\|_2$ **do**
  $z = \text{Vcycle}(r, \nu_1, \nu_2, O, res$,GS);
  $k = k + 1$;
  if $k == 1$ **then**
    $p = z$; $\rho = res^T z$;
  else
    $\tilde{\rho} = \rho$; $\rho = res^T z$;
    $\beta = \rho/\tilde{\rho}$; $p = z + \beta p$;
  end
  $w = Ap$; $a = \rho/p^T w$;
  $X = X + ap$; $res = res - aw$;
end

5 数值结果及相应分析

取 PPT 上的算例, 迭代法均以初值为 0 进行数值试验.

5.1 以 DGS 为磨光子的 V-cycle 多重网格方法

对 $N = 64, 128, 256, 512, 1024, 2048$, 以 $\nu_1 = \nu_2 = 3$, $L = N/2$(网格层数 $r + 1 = \log_2 N$) 为初始状态, 分别调整 ν_1, ν_2, r 计算达到精度要求所需的 CPU 时间与 V-cycle 次数, 结果如下:

Table 1: $\nu_2 = 3, L = N/2$ 时随 ν_1 变化的 CPU 时间 (s)

N \ ν_1	1	2	3	4	5	6	7	8	9	10
64	0.0249	0.020	0.020	0.021	0.024	0.025	0.030	0.033	0.046	0.034
128	0.0730	0.0970	0.092	0.083	0.092	0.099	0.107	0.117	0.125	0.126
256	0.327	0.328	0.376	0.428	0.399	0.507	0.525	0.597	0.647	0.646
512	2.352	2.286	2.480	2.884	3.186	2.962	3.247	3.463	3.728	4.059
1024	12.700	13.031	15.170	16.752	18.552	17.547	19.279	20.917	22.405	23.926
2048	62.438	64.268	72.889	82.596	91.682	86.038	93.524	101.535	108.989	116.972

Table 2: $\nu_2 = 3, L = N/2$ 时随 ν_1 变化的 V-cycle 次数

N \ ν_1	1	2	3	4	5	6	7	8	9	10
64	7	7	6	6	6	6	6	6	6	5
128	8	7	7	6	6	6	6	6	6	6
256	8	7	7	7	6	6	6	6	6	6
512	8	7	7	7	7	6	6	6	6	6
1024	8	7	7	7	7	6	6	6	6	6
2048	8	7	7	7	7	6	6	6	6	6

Table 3: $\nu_1 = 3, L = N/2$ 时随 ν_2 变化的 CPU 时间 (s)

N \ ν_2	1	2	3	4	5	6	7	8	9	10
64	0.329	0.048	0.048	0.042	0.036	0.044	0.031	0.031	0.033	0.035
128	0.107	0.137	0.139	0.103	0.109	0.116	0.100	0.132	0.122	0.130
256	0.438	0.452	0.440	0.429	0.475	0.523	0.563	0.514	0.539	0.572
512	2.413	2.144	2.443	2.327	3.438	3.103	5.082	3.256	3.562	5.652
1024	20.719	20.526	17.682	17.028	23.383	19.869	20.553	18.134	19.621	21.348
2048	74.769	77.584	77.553	87.454	82.842	91.873	99.822	90.993	97.085	104.249

Table 4: $\nu_1 = 3, L = N/2$ 时随 ν_2 变化的 V-cycle 次数

N \ ν_2	1	2	3	4	5	6	7	8	9	10
64	8	7	6	6	6	6	5	5	5	5
128	8	7	7	6	6	6	5	5	5	5
256	9	8	7	6	6	6	6	5	5	5
512	9	7	7	6	6	6	6	5	5	5
1024	9	8	7	6	6	6	6	5	5	5
2048	9	8	7	7	6	6	6	5	5	5

Table 5: $\nu_1 = \nu_2 = 3$ 时随 L 变化的 CPU 时间 (s)

N \ L	$N/2^1$	$N/2^2$	$N/2^3$	$N/2^4$
64	0.035	0.026	0.085	0.469
128	0.101	0.100	0.311	1.150
256	0.438	0.439	1.526	5.250
512	2.661	2.631	8.211	29.698
1024	15.151	15.136	47.926	168.882
2048	74.162	73.109	230.706	795.285

Table 6: $\nu_1 = \nu_2 = 3$ 时随 L 变化的 V-cycle 次数

N \ L	$N/2^1$	$N/2^2$	$N/2^3$	$N/2^4$
64	6	7	23	87
128	7	7	23	84
256	7	7	23	82
512	7	7	22	80
1024	7	7	22	78
2048	7	7	22	76

取 $N = 2048$ 时的计算结果可作下图:

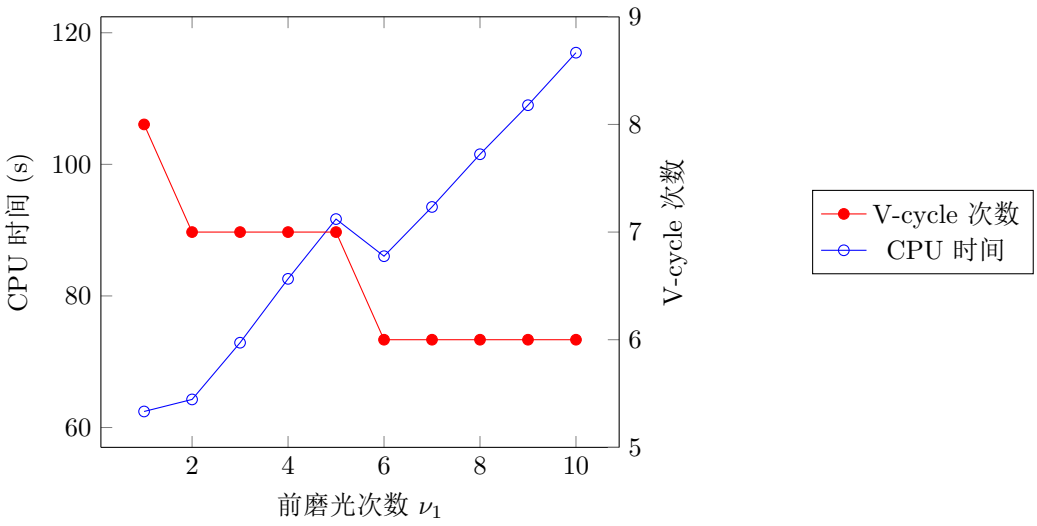


Figure 2: $N = 2^{11}, \nu_2 = 3, L = 2^{10}$ 时, V-cycle 次数与 CPU 时间随 ν_1 变化图

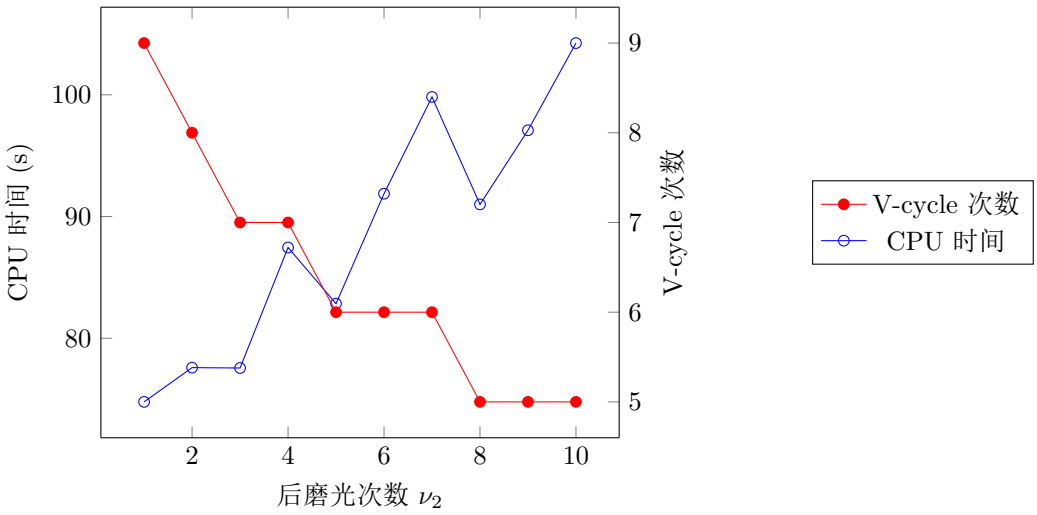


Figure 3: $N = 2^{11}, \nu_1 = 3, L = 2^{10}$ 时, V-cycle 次数与 CPU 时间随 ν_2 变化图

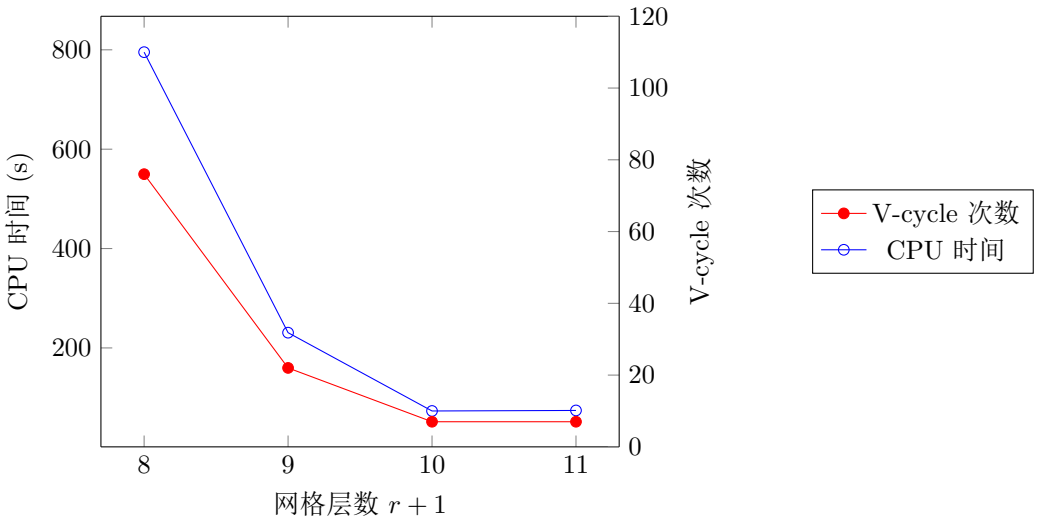


Figure 4: $N = 2^{11}, \nu_1 = \nu_2 = 3$ 时, V-cycle 次数与 CPU 时间随网格层数 $r+1$ 变化图

对每个网格规模 N 计算恰好满足精度要求时的误差, 如下图所示:

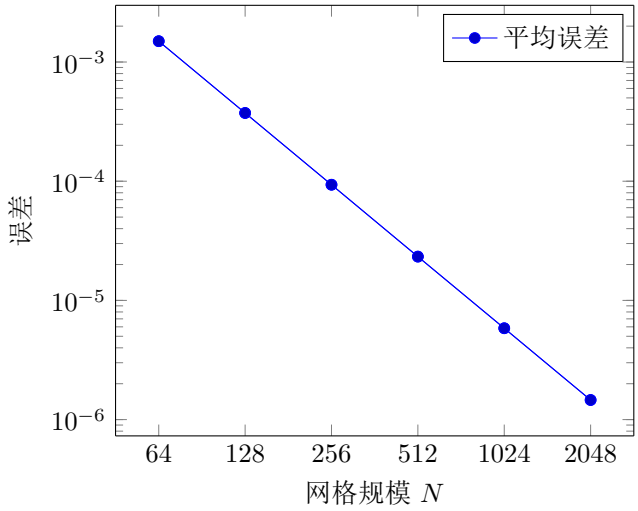


Figure 5: 以 DGS 为磨光子的 V-cycle 多重网格方法得到的误差随网格规模变化图

由图表结果可知:

- 随着前磨光次数、后磨光次数的增加, 所需的 V-cycle 次数减少, 但磨光增多的时间成本高于减少的循环次数, CPU 时间仍然呈上升趋势;
- 随着网格层数增加, 所需 CPU 时间和 V-cycle 次数显著降低;
- 与前磨光次数相比, 后磨光次数的变化对所需的 V-cycle 次数影响略大; 两者影响基本相当;
- 与前后磨光次数相比, 网格层数的变化对所需 CPU 时间和 V-cycle 次数影响更大. 这说明限制在粗网格上磨光对误差的削减效果比单纯在细网格上增加磨光次数带来的效果更显著;
- 随着网格规模 N 增大, 达到精度要求时数值解与真解的平均误差降低.

5.2 Exact Uzawa 迭代法

对 $N = 64, 128, 256, 512$, 使用 Exact Uzawa 迭代法得到数值解, 与真解的误差如图所示:

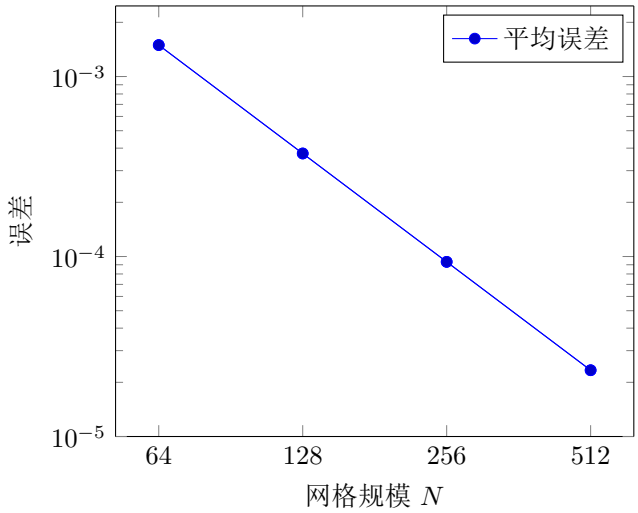


Figure 6: Exact Uzawa 迭代法得到的误差随网格规模变化图

由图像可知, 随着网格规模增加, 达精度要求的数值解与真解的误差降低; 并且同一网格规模、同一初值的情况下, Exact Uzawa 方法与以 DGS 为磨光子的 V-cycle 多重网格方法得到的误差几乎相等.

5.3 Inexact Uzawa 迭代法

对 $N = 64, 128, 256, 512, 1024, 2048$, 以 $\alpha = 1, \tau = 2, \nu_1 = \nu_2 = 3$, $L = N/2$(网格层数 $r + 1 = \log_2 N$) 为初始状态, 分别调整 $\alpha, \tau, \nu_1, \nu_2, r$(由于 V-cycle 中已经看出 $\nu_{1,2}$ 的影响相差不大, 因此此处取 $\nu_1 = \nu_2$ 一起调整.) 计算达到精度要求所需的 CPU 时间与外循环迭代次数, 结果如下:

Table 7: $\alpha = 1, \nu_1 = \nu_2 = 3, L = N/2$ 时随 τ 变化的 CPU 时间 (s)

N \ τ	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
64	0.039	0.028	0.027	0.030	0.038	0.037	0.031	0.030	0.036	0.054
128	0.157	0.119	0.110	0.134	0.110	0.118	0.112	0.126	0.127	0.143
256	0.591	0.515	0.533	0.517	0.542	0.525	0.521	0.505	0.527	0.538
512	3.818	3.882	3.648	3.356	3.096	3.068	3.152	3.107	3.337	3.389
1024	21.652	23.144	22.298	21.543	21.608	21.648	20.701	20.831	20.987	20.807
2048	132.544	120.557	115.602	116.179	111.502	111.208	111.161	111.567	112.556	111.439

N \ τ	2.2	2.4	2.6	2.8	3.0	3.2	3.4	3.6	3.8	4.0
64	0.037	0.028	0.040	0.039	0.033	0.032	0.032	0.033	0.032	0.031
128	0.124	0.121	0.126	0.130	0.133	0.117	0.113	0.109	0.114	0.129
256	0.516	0.525	0.531	0.500	0.519	0.530	0.510	0.531	0.521	0.504
512	3.200	3.246	3.197	3.427	3.222	2.962	2.990	3.017	3.034	3.017
1024	20.392	20.666	20.540	19.397	19.390	19.374	19.402	19.643	19.595	19.652
2048	111.314	106.644	106.548	102.031	101.896	97.760	102.300	102.200	101.918	103.574

Table 8: $\alpha = 1, \nu_1 = \nu_2 = 3, L = N/2$ 时随 τ 变化的外循环迭代次数

N \ τ	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
64	6	6	6	6	6	6	6	6	6	8
128	6	6	6	7	7	7	7	7	7	7
256	6	6	6	6	7	7	7	7	7	7
512	6	6	6	6	6	6	6	6	7	7
1024	5	6	6	6	6	6	6	6	6	6
2048	6	6	6	6	6	6	6	6	6	6
N \ τ	2.2	2.4	2.6	2.8	3.0	3.2	3.4	3.6	3.8	4.0
64	8	8	9	10	10	10	10	10	10	10
128	7	7	8	8	8	8	7	7	7	7
256	7	7	7	7	7	7	7	7	7	7
512	7	7	7	7	7	7	7	7	7	7
1024	6	6	6	7	7	7	7	7	7	7
2048	6	6	6	6	6	6	6	6	6	7

Table 9: $\tau = 2, \nu_1 = \nu_2 = 3, L = N/2$ 时随 α 变化的 CPU 时间 (s)

N \ α	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5
64	0.067	0.043	0.050	0.059	0.041	0.057	0.046	0.055	0.062	0.076	0.096
128	0.219	0.255	0.242	0.174	0.141	0.121	0.159	0.184	0.251	0.288	0.379
256	1.279	1.014	0.927	0.694	0.626	0.514	0.640	0.822	1.049	1.277	1.669
512	7.408	6.746	5.645	4.707	3.821	3.342	4.227	5.561	6.948	9.222	12.167
1024	48.725	39.359	32.770	33.540	26.400	20.977	25.992	31.563	42.623	52.871	70.068
2048	248.198	230.756	189.224	160.474	129.700	110.324	130.148	178.472	215.554	264.801	337.976

Table 10: $\tau = 2, \nu_1 = \nu_2 = 3, L = N/2$ 时随 α 变化的外循环迭代次数

N \ α	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5
64	21	17	14	12	8	8	8	11	14	17	22
128	21	15	12	10	8	7	9	11	14	18	24
256	19	15	13	10	9	7	9	12	15	19	25
512	19	16	13	11	9	7	9	12	14	18	23
1024	19	16	13	11	8	6	7	9	12	15	20
2048	20	13	11	9	7	6	7	10	12	16	20

Table 11: $\tau = 2, \alpha = 1, L = N/2$ 时随 $\nu_1 = \nu_2$ 变化的 CPU 时间 (s)

N \ $\nu_{1,2}$	2	3	4	5	6	7	8	9	10
64	0.048	0.051	0.051	0.040	0.042	0.036	0.046	0.040	0.042
128	0.136	0.140	0.115	0.134	0.130	0.129	0.146	0.157	0.137
256	0.543	0.495	0.525	0.517	0.561	0.647	0.628	0.678	0.654
512	3.212	3.249	2.916	3.170	3.314	3.637	3.825	4.460	4.138
1024	19.981	20.418	19.644	19.944	20.752	22.949	25.080	25.629	27.570
2048	116.559	109.936	97.587	105.197	111.394	123.031	126.831	137.741	149.530

Table 12: $\tau = 2, \alpha = 1, L = N/2$ 时随 $\nu_1 = \nu_2$ 变化的外循环迭代次数

N \ $\nu_{1,2}$	2	3	4	5	6	7	8	9	10
64	8	8	10	9	9	8	8	8	7
128	8	7	6	8	8	8	8	8	7
256	7	7	6	6	8	8	8	8	7
512	6	7	6	6	6	6	8	8	7
1024	6	6	6	6	6	6	6	6	6
2048	6	6	6	6	6	6	6	6	6

Table 13: $\tau = 2, \alpha = 1, \nu_1 = \nu_2 = 3$ 时随 L 变化的 CPU 时间 (s)

N \ L	$N/2^1$	$N/2^2$	$N/2^3$	$N/2^4$
64	0.038	0.024	0.047	0.120
128	0.125	0.099	0.181	0.354
256	0.478	0.483	0.806	1.693
512	3.211	3.233	5.245	9.671
1024	20.456	20.572	29.067	52.666
2048	110.314	111.045	141.993	312.938

Table 14: $\tau = 2, \alpha = 1, \nu_1 = \nu_2 = 3$ 时随 L 变化的外循环迭代次数

N \ L	$N/2^1$	$N/2^2$	$N/2^3$	$N/2^4$
64	8	6	11	12
128	7	6	9	13
256	7	7	8	11
512	7	7	8	10
1024	6	6	7	8
2048	6	6	6	8

取 $N = 1024$ 时的计算结果可作下图:

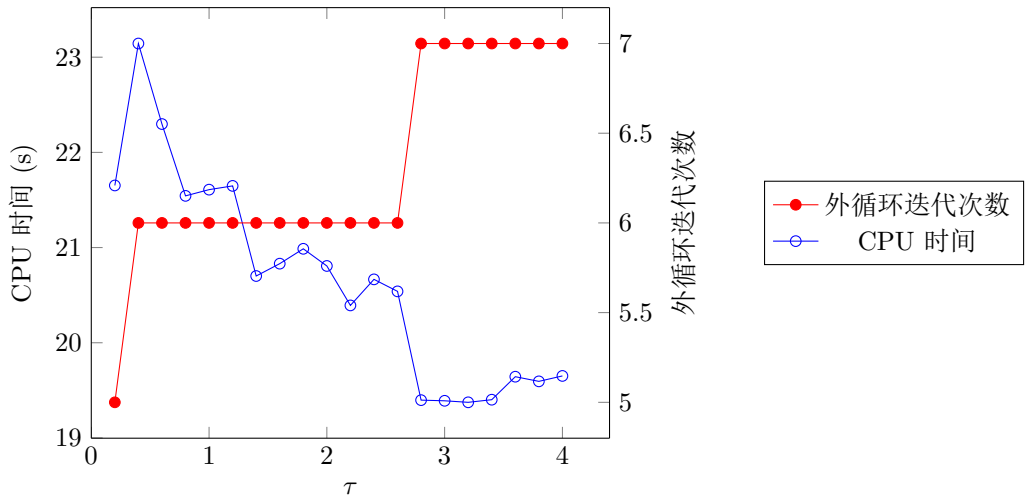


Figure 7: $N = 2^{10}, \alpha = 1, \nu_1 = \nu_2 = 3, L = 2^9$ 时, 外循环迭代次数与 CPU 时间随 τ 变化图

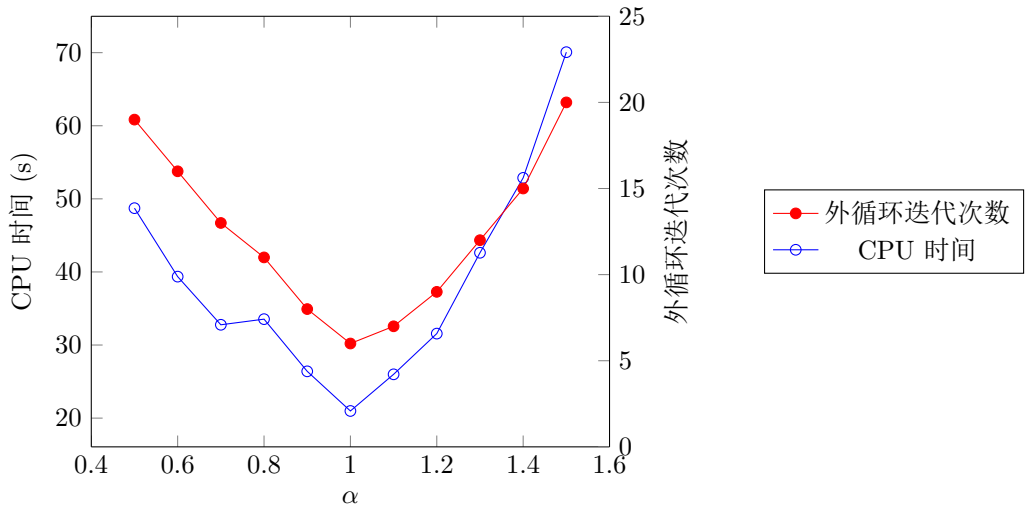


Figure 8: $N = 2^{10}, \tau = 2, \nu_1 = \nu_2 = 3, L = 2^9$ 时, 外循环迭代次数与 CPU 时间随 α 变化图

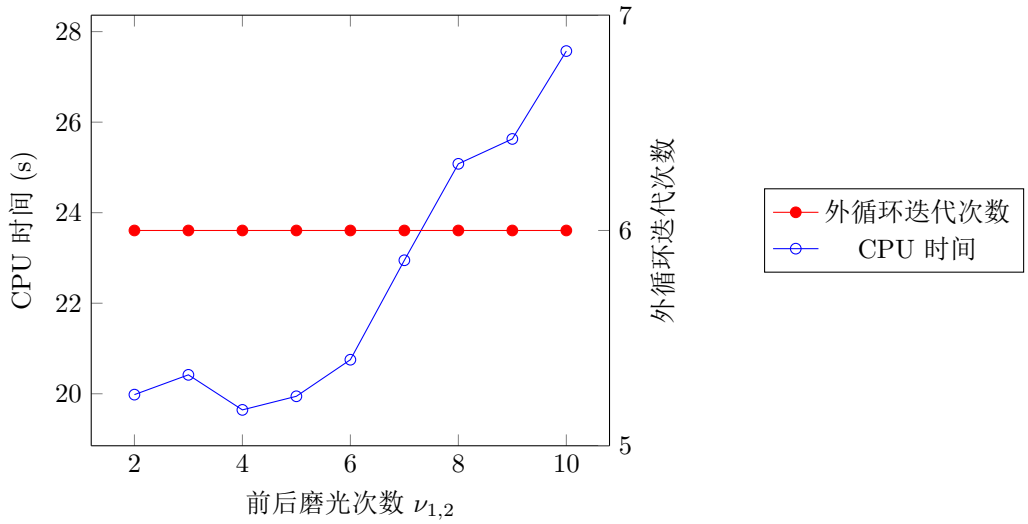


Figure 9: $N = 2^{10}, \alpha = 1, \tau = 2, L = 2^9$ 时, 外循环迭代次数与 CPU 时间随 $\nu_1 = \nu_2$ 变化图

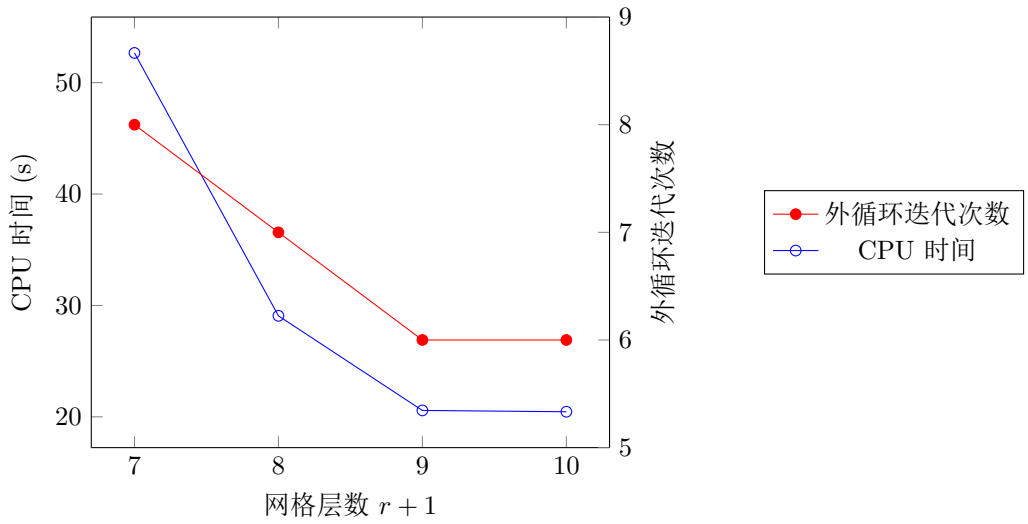


Figure 10: $N = 2^{10}, \alpha = 1, \tau = 2, \nu_1 = \nu_2 = 3$ 时, 外循环迭代次数与 CPU 时间随网格层数 $r+1$ 变化图

对每个网格规模 N 计算恰好满足精度要求时的误差, 如下图所示:

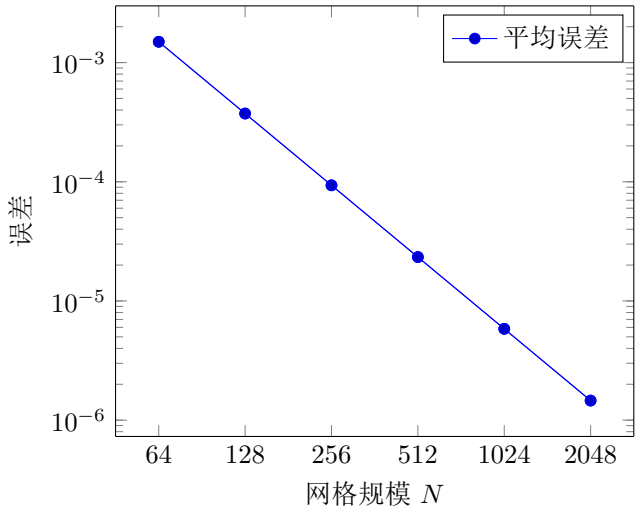


Figure 11: Inexact Uzawa 迭代法得到的误差随网格规模变化图

由图表结果可知:

- 其他条件不变, 在迭代法仍能收敛时, 随着 τ 增大, 外循环迭代次数呈上升趋势, CPU 时间呈下降趋势; 这说明在迭代法能收敛的情况下, τ 更大、近似解精度要求更低, 尽管会导致外循环次数增加, 但依然有利于加速收敛.
- 其他条件不变, 随着 α 向 1 靠近, 外循环迭代次数与 CPU 时间均呈下降趋势;
- 其他条件不变, 在磨光次数不小于 2 时, 随着前后磨光次数同时增加, CPU 时间增加, 但外循环迭代次数几乎不变. 这说明继续增加磨光次数, 对一次 Vcycle 近似求解 $Az_k = p_k$ 的精度影响不显著, 因此共轭梯度法求解子问题的近似解差异不大, 导致外循环次数变化不大, 用于磨光的时间成本几乎没有回报;
- 其他条件不变, 随着网格层数增加, 外循环迭代次数与 CPU 时间呈显著下降趋势;
- 与 τ、磨光次数 $\nu_{1,2}$ 相比, α、网格层数 $r+1$ 的改变对 CPU 时间的影响更显著; 这是因为: 迭代法的参数 α 基本决定外循环迭代次数, 而网格层数 $r+1$ 最能影响预条件子预优效率. 这两个参数对 Inexact Uzawa 方法的效率影响最显著.
- 随着网格规模增加, 达精度要求的数值解与真解的误差降低; 并且同一网格规模、同一初值的情况下, Inexact Uzawa 方法与 Exact Uzawa、以 DGS 为磨光子的 V-cycle 多重网格方法得到的误差几乎相等.