用优化方法计算大规模 Sylvester 方程与矩阵分离度

刘 歆
北京大学（100871）
数学科学学院
科学与工程计算系
电子邮件：meteor_lx@etang.com
2004 年 5 月

Optimization Methods for Large-scale Sylvester Equation and the Separation of two Matrices

Xin Liu
Department of Scientific and Engineering Computing
School of Mathematical Sciences
Peking University (100871)
Email: meteor_lx@etang.com
May 2004

摘 要

Sylvester 方程是矩阵理论研究中的一种非常重要的方程；在控制论、系统理论等工程领域和数学领域有着极其广泛应用的 Lyapunov 方程就可以看成是一种特殊的 Sylvester 方程。矩阵分离度在 Sylvester 方程的敏度分析和不变子空间的扰动理论中有着非常重要的应用。因此选择高效的算法求解 Sylvester 方程和计算矩阵分离度是非常必要的。传统的数值代数方法效率不高，基本只能处理经典的小型问题或者做一些估计；而一般的优化方法虽然对小规模问题非常有效，但也面临对大规模问题的存储困难。本文给出的用于 Sylvester 方程求解和计算矩阵分离度的优化算法，能处理的问题规模比以往大了不少。它们不但解决了存储问题，而且运算量也小了，此外对有特殊形式的问题算法还有更好的效果。

关键词：Sylvester 方程，Lyapunov 方程，矩阵分离度，梯度型算法，投影梯度算法，投影 BB 方法，Poisson 方程，有限差分法。

全文通用记号

x_k	第 k 个迭代点
$g_k = \nabla f(x_k)$	目标函数 $f(x)$ 在 x_k 点的梯度向量
$\langle v_1, v_2 \rangle$	两个向量 v_1 与 v_2 的内积
$\dot{P}(t)$	矩阵 P 的每个元素都对 t 求一阶导
P^{T}	矩阵 P 或向量 P 的转置
$f'(x)$	函数 $f(x)$ 的一阶导函数

另外，如不作特殊声明，文中所有范数$\|\cdot\|$均指欧氏范数$\|\cdot\|_2$.

目 录

第一章 引 言

第一节 问题背景

1. Lyapunov 方程：

1892 年，俄罗斯数学家 Alexander Mikhailovitch Lyapunov，在他的博士论文中提出其著名的线性与非线性系统的稳定性理论时，引入了一类新的矩阵方程。这类方程能很好地描述系统的状态。后来人们把这类方程称为 Lyapunov 方程。1992 年 3 月，Lyapunov 当年的博士论文的完整的英译稿首次被发表在 *International Journal of Control* 上。

Lyapunov 方程以及具有 Lyapunov 型的方程应用极广，除了控制论、系统论之外，还在很多其他的工程领域和数学领域有着相当广泛的应用。例如：信号处理、线性代数、微分方程、边枝定界问题、通讯工程、建筑……。

Lyapunov 理论在十九世纪末被提出，它的应用从二十世纪六十年代开始被人们逐渐了解和接受，并且很快就成为控制论中最主要的研究课题之一。正因为有着广泛的应用，在过去四十年中，Lyapunov 方程也是数学家们的一个非常活跃的研究课题。

下面我们将介绍 Lyapunov 方程的具体形式。

一般称如下方程为微分 Lyapunov 矩阵方程，它能描述随时间连续变化的系统：

$$\begin{cases} A^{\mathrm{T}}(t)P(t) + P(t)A(t) + Q(t) = \dot{P}(t) \\ P(t_0) = P_0 \end{cases}. \qquad (1.1.1)$$

其中 $P(t)$ 是 $n\times n$ 的未知解矩阵， $A(t)$ 是 $n\times n$ 的系统矩阵， $Q(t)$ 是 $n\times n$ 对称矩阵（一般是正定或半正定的）。

如果系统随时间不变化，则 $\dot{P}=0$（固定状态），这时矩阵（1.1.1）变为代数 Lyapunov 矩阵方程：

$$A^{\mathrm{T}}P + PA + \mathrm{Q} = 0. \qquad (1.1.2)$$

如果系统随时间离散变化，那么对应的 Lyapunov 矩阵差分方程为：

$$\begin{cases} A^{T}(k)P(k)A(k) + Q(k) = P(k+1) \\ P(k_0) = P_0 \end{cases}. \qquad (1.1.3)$$

对于随时间不变的离散系统，它始终在固定状态： $P(k+1)=P(k)=P_0$，我们有离散形式的代数 Lyapunov 矩阵方程：

$$A^{\mathrm{T}}PA + \mathrm{Q} = P\,. \qquad (1.1.4)$$

以上就是最一般的四种 Lyapunov 方程，其中方程（1.1.2）和（1.1.4）又是研究和解决所有 Lyapunov 方程的基础。

2. Sylvester 方程：

我们称形如：

$$AX - XB = C\,. \qquad (1.1.5)$$

其中 $A \in \mathrm{R}^{n\times n}$， $B \in \mathrm{R}^{m\times m}$， $C \in \mathrm{R}^{n\times m}$， $X \in \mathrm{R}^{n\times m}$ 的矩阵方程为 Sylvester 方程。

在代数中，尤其是矩阵理论的研究中，Sylvester 是一类非常重要的矩阵方程。它的应用也非常广泛。我们刚才介绍的 Lyapunov 方程的代数形式就是一类特殊的 Sylvester 方程；在偏微分方程数值解理论中，椭圆型方程的有限差分格式也可以化为 Sylvester 方程。

将 Sylvester 方程的解矩阵化成列向量 $x \in \mathrm{R}^{nm}$，相应地，Sylvester 方程就化为：

$Dx = \tilde{c}$，其中 $D \in \mathrm{R}^{nm\times nm}$，$\tilde{c} \in \mathrm{R}^{nm}$ 都是由 A、B、C 决定的常量。这样我们就把 Sylvester 方程化成了和它等价的最简单的线性方程组。然而这样处理以后，问题的规模陡增，大大增加了存储量和计算量（这在本文后面会详细讨论）。于是寻求 Sylvester 方程的直接的快速的解法成为了研究 Sylvester 的重要课题。

3. 矩阵分离度的提出和应用：

我们记：

$$\mathrm{Sep}(A,\ B) = \min\{\ \|AX - XB\|_{\mathrm{F}}\ ,\ X \in \mathrm{R}^{n\times m}\ ,\ \|X\|_{\mathrm{F}} = 1\ \}\,. \qquad (1.1.6)$$

其中， $A \in \mathrm{R}^{n\times m}$ ，$B \in \mathrm{R}^{n\times m}$。我们称如上定义的 $\mathrm{Sep}(A,\ B)$ 为矩阵 A 和矩阵 B 的分离度。由于有限维线性赋范空间的范数是等价的，本文只考虑取 Frobenius 范数的矩阵分离度计算。

我们观察线性算子 L： $LX = AX - XB$，根据（1.1.6）容易验证它的逆 L^{-1} 的范数是：$\left\|L^{-1}\right\| = \max\limits_{\|x\|=1}\left\|L^{-1}x\right\| = 1 / \mathrm{Sep}(A,\ B)$，正是基于这样的性质，矩阵分离度的定义才会被提出。为了保证 L^{-1} 范数有意义，这里要求 $\lambda(A) \cap \lambda(B) = \varnothing$。否则，$\mathrm{Sep}(A,\ B) = 0$，这在[9]中有详细的证明。

我们刚才已经指出了 $\left\|L^{-1}\right\| = 1 / \mathrm{Sep}(A,\ B)$ 这一事实；而根据范数定义，我们又容易知

道：$\|L\| = \max_{\|X\|=1} \|AX - XB\| \le \|A\| + \|B\|$。如果我们和一般线性方程组 $Ax = b$ 的条件数那样类似地定义 Sylvester 方程 $AX - XB = C$ 的条件数 $\kappa = \|L\| \cdot \|L^{-1}\|$，就可以得到 $\kappa = (\|A\| + \|B\|) / \mathrm{Sep}(A, B)$。这样，为了估计扰动对 Sylvester 方程解的影响，估算矩阵分离度 $\mathrm{Sep}(A, B)$ 是十分重要的。事实上， [6]中已经给出了如下定理 1 的证明：

定理 1：设 $\tilde{X}$ 是 Sylvester 方程（1.1.5）的计算解，X 是精确解。记 $R =: A\tilde{X} - \tilde{X}B - C$，则有如下结论成立：

$$\frac{\|\tilde{X} - X\|}{\|X\|} \le \frac{\|A\| + \|B\|}{\mathrm{Sep}(A, B)} \cdot \frac{\|R\|}{\|C\|}$$

这个定理揭示出了矩阵分离度在 Sylvester 方程敏度分析中的重要价值。

此外，矩阵分离度在不变子空间的扰动理论中也有着重要的作用。这在[9]中有较为详细的讨论，这里不再赘述。

第二节　方法简介

上一节中，我们简单介绍了 Sylvester 方程和矩阵分离度问题的重要性。对于这两个线性代数问题的求解方法，在数值代数界已经被研究了几十年了，但仍然找不到高效率的算法，特别是针对大规模问题。于是，我们寻求用优化方法来解决这两个问题。

优化方法就是通过建模把一个问题化为如下优化形式：

$$\min f(x), \tag{1.2.1}$$

$$\text{s.t. } c_i(x) = 0,\ i \in E, \tag{1.2.2}$$

$$c_i(x) \ge 0,\ \ i \in I. \tag{1.2.3}$$

这里 E 和 I 分别是等式约束的指标集和不等式约束的指标集。(1.2.1)称为目标函数，(1.2.2)称为等式约束，（1.2.3）称为是不等式约束。当 E 和 I 都是空集时，我们称这样的优化问题为无约束优化问题；否则，称为约束优化问题。

对于所有的优化问题，我们通常有两大类方法：线搜索方法和信赖域方法。其中线搜索法是最为常用，且被研究和应用最广的优化方法。本文考虑到大规模问题的主要困难在于存储和运算量，所以只选择采用线搜索方法。

1. 无约束优化问题和梯度型方法：

对于无约束优化问题：

$$\min f(x). \tag{1.2.4}$$

线搜索方法的基本思想是对于给定的初始点 x_0 和某种终止条件，寻找一种合适的迭代方案：

$$x_{k+1} = x_k + \alpha_k d_k. \tag{1.2.5}$$

其中 x_k 为第 k 次迭代点， d_k 是第 k 次搜索方向（也称下降方向）， α_k 是第 k 次步长因子。一般而言，我们选择的迭代方案总是使得目标函数值有某种意义的下降：$-\nabla f(x_k)^{\mathrm{T}} d_k < 0$ 或者 $f(x_k + \alpha_k d_k) < f(x_k)$。

梯度型方法就是一种最基本的线搜索方法。顾名思义，梯度型算法是以负梯度方向作为极小化算法的下降方向，我国已故的著名数学家华罗庚先生形象地称之为“盲人上山法”。

作为线搜索方法的一种，除了下降方向之外，步长因子是另一个非常重要的因素。在梯度型方法中，不同的步长因子取法就是划分不同的梯度型算法的依据。我们熟知的最速下降法就是一种取步长：

$$\alpha_k = \arg\min_{\alpha} f(x_k + \alpha d_k). \tag{1.2.6}$$

的梯度型方法。最速下降法可以说是整个非线性最优化理论中最简单的算法了，而且它有着一些非常好的性质，例如计算量小，从任何初始点出发都能收敛等。然而，它的收敛速度非常的慢（线性收敛），而且对函数依赖程度高，于是，它反而成为最不好用的方法了。

随着研究的深入，人们在处理问题的规模越来越大，而且一般数值问题（无论是来自方程还是来自代数）都会归结于解最简单的线性方程组：

$$Ax = b. \tag{1.2.7}$$

其中 $A \in \mathrm{R}^{n\times n}$， $b \in \mathrm{R}^n$， $x \in \mathrm{R}^n$。据统计，全世界现今所有超级计算机所作的工作的 80% 是在解（1.2.7）式，所以寻求大规模线性方程组的快速解法是当今比较重要的研究课题。在优化中我们可以很简单地将（1.6）式转化成无约束优化形式：

$$\min f(x) = \frac{1}{2} x^{\mathrm{T}} A x - b^{\mathrm{T}} x. \tag{1.2.8}$$

在研究 x 维数非常大时候，（1.2.8）的既节省空间代价又节省时间代价的算法，优化学家们又把眼光放到了梯度型方法上来，因为它的计算量和存储量是最适合大规模二次问题（1.2.8）的。

然而传统的最速下降方法的收敛速度实在让人不敢恭维，因为它在二次型较扁时会出现明显的 zigzag 效应：即下一次迭代的下降方向 d_{k+1} 和当次迭代的下降方向 d_k 垂直，于是

当迭代点越接近极小点时下降的速度越慢，下降的轨迹成齿轮状。

经过简单推算我们可以发现最速下降法产生 zigzag 效应的症结所在：

$x_{k+1} = x_k - \alpha_k g_k$， 根据（1.2.6），$\alpha_k = \arg\min_{\alpha} f(x_k - \alpha g_k)$

于是有：$-f'(x_k - \alpha_k g_k)^{\mathrm{T}} g_k = 0$，也即：$-g_{k+1}^{\mathrm{T}} g_k = 0$

所以根据（1.2.6）式取函数值下降最小的步长是造成最速下降法失败的根本。也就是说最好的下降方向和最好的步长并不合适用在一起。

在探索其它梯度型方法的过程中，两位加拿大优化学家：Jonathan Barzilai 和 Jonathan M. Borwein 做出了开创性的一步，1988 年他们在[1]中提出了“两点步长梯度法”，后来被命名为“BB 方法”。

BB 方法的思想非常简单：考虑迭代：$x_{k+1} = x_k - S_k g_k$，其中 $S_k = \alpha_k I$。且 α_k 满足：

$$\alpha_k = \arg\min_{\alpha} \left\| \Delta x - S_k \Delta g \right\|^2, \tag{1.2.9}$$

或者：

$$\alpha_k = \arg\min_{\alpha} \left\| S_k^{-1} \Delta x - \Delta g \right\|^2. \tag{1.2.10}$$

其中 $\Delta x = x_k - x_{k-1}$， $\Delta g = g_k - g_{k-1}$。我们注意到（1.2.9）、（1.2.10）式的定义形式是从拟牛顿条件变化而来的，目的是让由此产生的步长 α_k 具有某些拟牛顿属性。由于每次迭代都需要前两个迭代点的信息，所以 Barzilai 和 Borwein 最初把这种方法叫做“两点步长梯度法”。解（1.2.9），可得：

$$\alpha_k = \langle \Delta x, \Delta g \rangle / \langle \Delta g, \Delta g \rangle, \tag{1.2.11}$$

解（1.10），可得：

$$\alpha_k = \langle \Delta x, \Delta x \rangle / \langle \Delta x, \Delta g \rangle. \tag{1.2.12}$$

对于二次型问题（1.2.8），（1.2.11）和（1.2.12）分别等价于：

$$\alpha_k = \mathrm{g}_{k-1}^{\mathrm{T}} A \mathrm{g}_{k-1} / \mathrm{g}_{k-1}^{\mathrm{T}} A^2 g_{k-1}, \tag{1.2.13}$$

和：

$$\alpha_k = \mathrm{g}_{k-1}^{\mathrm{T}} \mathrm{g}_{k-1} / \mathrm{g}_{k-1}^{\mathrm{T}} A g_{k-1}. \tag{1.2.14}$$

这时步长 α_k 具有了实际含义，它分别等于上一次迭代的最大梯度值下降 $\arg\min_{\alpha} g(x_{k-1} + \alpha d_{k-1})$ 和上一次迭代的最大函数值下降 $\arg\min_{\alpha} f(x_{k-1} + \alpha d_{k-1})$。从这个实际含义可以直接看出最速下降法的 zigzag 效应在 BB 方法中将会得到有效的避免。

BB 方法虽然不能保证在每一步迭代目标函数值都有下降，但对于二次型问题（1.2.8），它确实能够收敛至最优解，而且 Marcos Raydan 在[7]中给出了算法 R-线性收敛的证明，经

过试验，对于问题（1.2.8），BB 方法的收敛速度大大快于最速下降法。

BB 方法可以说是打开了梯度型方法研究之门。

后来中科院计算数学所的戴彧虹老师给出了 AS 算法，参见[2]；他和袁亚湘老师一起给出了 AM 算法，参见[3]。关于这些方法的详细介绍和一些试验在本文的后面会有提及，这里不再赘述。

2. 等式约束优化问题和广义消去法：

对于只有等式约束的非线性优化问题：

$$\min f(x), \tag{1.2.15}$$

$$\text{s.t. } c(x)=0. \tag{1.2.16}$$

其中 $c(x)=(c_1(x),\ \dots\ , c_m(x))^{\mathrm{T}}$ 。

解决等式约束优化的最基本的思想是将约束条件化去，也即将等式约束优化问题转化为无约束优化问题。通常的手段有罚函数法、逐步二次规划法、可行点法等等。详见[12]、[13]。

广义消去法也是解等式约束优化问题的一类非常行之有效的方法。下面简单介绍一下广义消去法的基本原理。

考虑任何非奇异矩阵 $S\in \mathrm{R}^{n\times n}$ 以及变量替换： $x=Sw$ 。对 w 进行变量分离

$$w=\begin{bmatrix} w_B \\ w_N \end{bmatrix}, \tag{1.2.17}$$

其中 $w_B\in R^m$， $w_N\in R^{n-m}$ 。利用约束条件

$$c(S_B w_B + S_N w_N)=0, \tag{1.2.18}$$

进行变量消去得到

$$w_B=\overline{\varphi}(w_N). \tag{1.2.19}$$

于是优化问题（1.2.15）—（1.2.16）等价于

$$\min_{w_N\in R^{n-m}} f(S_B w_B + S_N w_N)=\overline{f}(w_N). \tag{1.2.20}$$

只要 $S_B^{\mathrm{T}}\nabla c(x)^{\mathrm{T}}$ 非奇异，则利用直接计算可得到既约梯度

$$\nabla_{w_N}\overline{f}(w_N)=\overline{g}(w_N)=S_N^{\mathrm{T}}[\nabla f(x)-\nabla c(x)^{\mathrm{T}}\lambda], \tag{1.2.21}$$

其中 λ 满足于

$$S_B^{\mathrm{T}}[\nabla f(x)-\nabla c(x)^{\mathrm{T}}\lambda]=0. \tag{1.2.22}$$

对于无约束优化问题（1.2.20），我们采取线搜索方法。在每一步迭代中，按某种方式产生下降方向 $\overline{d}_k$，并用某种线搜索方法给出试探步长 $\alpha_k > 0$；于是 $(w_{k+1})_N = (w_k)_N + \alpha_k \overline{d}_k$。我们需要计算 $w_B = \overline{\varphi}((w_k)_N + \alpha_k \overline{d}_k)$，这时应采用近似的拟牛顿法求解

$$c((S_k)_B w_B + (S_k)_N[(w_k)_N + \alpha_k \overline{d}_k]) = 0, \tag{1.2.23}$$

得到迭代公式

$$w_B^{(i+1)} = w_B^{(i)} - [(\nabla c(x_k)^{\mathrm{T}})(S_k)_B^{\mathrm{T}}]^{-1} c((S_k)_B w_B^{(i)} + (S_k)_N[(w_k)_N + \alpha_k \overline{d}_k]), \qquad i = 1, 2, \cdots. \tag{1.2.24}$$

不难看出，广义消去法每次迭代的变量增量 $x_{k+1} - x_k$ 实际上是两部分之和：

$$x_{k+1} = x_k + d_k^{(1)} + d_k^{(2)}, \tag{1.2.25}$$

其中，

$$d_k^{(1)} = \alpha_k (S_k)_N \overline{d}_k, \tag{1.2.26}$$

$$d_k^{(2)} = (S_k)_B[\overline{\varphi}((w_k)_N + \alpha_k \overline{d}_k) - (w_k)_B]. \tag{1.2.27}$$

广义消去法的迭代过程简单地说就是首先通过约束等式消去部分变量，同时将等式约束问题化为无约束问题；然后给出一种下降方向 $d_k^{(1)}$，并在 $d_k^{(1)}$ 上搜索试探步长；由于新的迭代点 $x_k + d_k^{(1)}$ 可能不在可行域内，所以在 $d_k^{(2)}$ 方向上利用近似的拟牛顿法得到正确步长，这一步好比把新的迭代点校正回可行域内。除非是特殊的约束条件，否则这种离开可行域再回来的技巧是不可避免的。为了使“离开程度”尽量的小，从而使得从 $x_k + d_k^{(1)}$ 用近似的拟牛顿法计算 x_{k+1} 收敛地更快，我们一般选取 $d_k^{(1)}$ 为一线性化可行方向，即：

$$(S_k)_N^{\mathrm{T}} \nabla c(x)^{\mathrm{T}} = 0. \tag{1.2.28}$$

3. 投影梯度法和投影 BB 法：

投影梯度法是一种特殊的广义消去法：遵循上面的讨论结果，选取 S_k 使得（1.2.28）成立。并且选取 $\overline{d}_k = -\overline{g}_k$，即在广义消去法中用梯度型方法。则由（1.2.26）可知：

$$d_k^{(1)} = -\alpha_k (S_k)_N (S_k)_N^{\mathrm{T}} \nabla f(x_k). \tag{1.2.29}$$

显然，$(S_k)_N(S_k)_N^{\mathrm{T}}$ 是从 R^n 到由 $(S_k)_N$ 的列向量所张成的子空间的一个线性映射。如果 $\nabla c(x_k)^{\mathrm{T}}$ 是列满秩的，则由 $(S_k)_N$ 的列向量所张成的子空间就是 $\nabla c(x_k)$ 的零空间。所以由（1.2.29）所定义的方向实际上就是目标函数的负梯度方向在约束函数的 Jacobi 矩阵的零空间的映照。如果 S_k 满足 $(S_k)_N^{\mathrm{T}}(S_k)_N = I$，则 $(S_k)_N(S_k)_N^{\mathrm{T}}$ 是一个投影算子，在 $\nabla c(x_k)^{\mathrm{T}}$ 列满秩的假设下有：$P_k = (S_k)_N(S_k)_N^{\mathrm{T}} = I - \nabla c(x_k)^{\mathrm{T}}(\nabla c(x_k)\nabla c(x_k)^{\mathrm{T}})^{-1}\nabla c(x_k)$。这时 $P_k\nabla f(x_k)$ 是 $\nabla f(x_k)$ 在 $\nabla c(x_k)$ 零空间上的投影。在实际计算中，我们可以利用 $\nabla c(x_k)^{\mathrm{T}}$ 的 QR 分解：

$\nabla c(x_k)^{\mathrm{T}} = \begin{bmatrix} Y_k & Z_k \end{bmatrix}\begin{bmatrix} R_k \\ 0 \end{bmatrix}$，选取 $(S_k)_N = Z_k$。下面给出一般投影梯度法的算法：

算法 1.1：投影梯度法

步 1：给出可行点 x_1，$\varepsilon \geq 0$，$\bar{\varepsilon}>0$，N 正整数，$k := 1$。

步 2：计算 QR 分解：

$$\nabla c(x_k)^{\mathrm{T}} = \begin{bmatrix} Y_k & Z_k \end{bmatrix}\begin{bmatrix} R_k \\ 0 \end{bmatrix};$$

$\bar{g}_k = Z_k^{\mathrm{T}}\nabla f(x_k)$；

如果 $\|\bar{g}_k\| \leq \varepsilon$ 则停；

$d_k = -Z_k\bar{g}_k$；取 $\alpha = \alpha_k^{(0)} > 0$。

步 3：$y := x_k + \alpha d_k$

$i := 0$

步 4：$y := y - Y_k R_k^{-1} c(y)$；

如果 $\|c(y)\| \leq \bar{\varepsilon}$ 和 $f(y) < f(x_k)$ 则转步 5；$i := i + 1$；

如果 $i < N$ 则转步 4；

$\alpha = \dfrac{\alpha}{2}$；转步 3。

步 5：$x_{k+1} = y$，$k := k+1$；转步 2。

为了保证投影梯度算法的收敛性，一般采用修正搜索条件或者限制初始步长来获取步长，参见[13]。然而这样势必会因搜索合适的步长因子而反复调用目标函数，大大增加了算法的时间代价。因此本文尝试将投影梯度法的试探步长选取为修正的 BB 步长，并用实验来

检验这样的选取是否能加快收敛，节省时间。我们一般把这样改进的投影梯度法称为投影 BB 法。

本文将在第二章中详细讨论矩阵分离度问题的优化算法：主要将讨论如何改进投影梯度法和投影 BB 法来适合大规模矩阵分离度问题，并且将改进的算法和 Lagrange—Newton 法、乘子罚函数法作了比较。第三章将讨论 Sylvester 方程的优化算法：将介绍两种优化转化方法，并且比较两种方法的在不同规模问题中的优劣；将比较梯度型各算法在计算大规模 Sylvester 方程中的优劣。第四章将给出建立在前两章基础之上的一些有趣的结果，其中第一节的讨论将得到一个揭示矩阵分离度和 Sylvester 方程之间关系的非常有趣的结论；第二节将本文介绍的计算 Sylvester 方程和矩阵分离度的优化算法用在解椭圆型方程的五点差分格式中，并将得到的结果和用有限元方法的结果加以可视化的比较。第五章对本文作了深入的总结，并且对今后的工作提出了展望。

第二章 用优化方法求矩阵分离度

第一节 算法设计

1. 问题转化

根据矩阵分离度的定义（1.1.6）我们可以非常容易地写出它的优化等价形式：

$$\min \ \|AX - XB\|_{\mathrm{F}}, \tag{2.1.1}$$

$$\text{s.t.} \ \ \|X\|_{\mathrm{F}} - 1 = 0. \tag{2.1.2}$$

我们记：$X = (X_{ij})$，$A = (a_{ij})$，$B = (b_{ij})$；并且做线性变换：T：$\mathrm{R}^{n\times m} \to \mathrm{R}^{nm}$，设 $TX = \tilde{x} \in \mathrm{R}^{nm}$，满足： $X_{ij} = \tilde{x}_{(j-1)n+i}$ ，或者 $\tilde{x}_p = X_{i_p j_p}$，其中 $i_p = \mathrm{mod}\,(p-1,\ n)+1$，$j_p = (p - i_p)/n + 1$。可将上面关于矩阵的优化问题（2.1.1）—（2.1.2）转化成下面一般的优化问题：

$$\min \ f(\tilde{x}) := \|AX - XB\|_{\mathrm{F}}^2 = \sum_{i=1}^{n}\sum_{j=1}^{m}\Big(\sum_{k=1}^{n} a_{ik}X_{kj} - \sum_{l=1}^{m} X_{il}b_{lj}\Big)^2, \tag{2.1.3}$$

$$\text{s.t.} \ \ c(\tilde{x}) := \|X\|_{\mathrm{F}}^2 - 1 = \sum_{p=1}^{nm}\tilde{x}_p^2 - 1 = 0. \tag{2.1.4}$$

这样，矩阵分离度问题就转化成了一个目标函数为二次函数，带一个二次等式约束的非线性有约束优化问题（2.1.3）—（2.1.4），问题的规模是 nm 维。

我们继续分析问题（2.1.1）—（2.1.2）中的目标函数和约束函数。目标函数（2.1.3）的梯度向量 $\nabla f(\tilde{x})$ 的第 p 个元素是

$$\begin{aligned}(\nabla f(\tilde{x}))_p = &\sum_{i=1}^{\mathrm{n}} 2a_{ii_p}\Big(\sum_{k=1}^{\mathrm{n}} a_{ik}X_{kj_p} - \sum_{l=1}^{\mathrm{m}} X_{il}b_{lj_p}\Big)\\ &+ \sum_{j=1}^{\mathrm{m}}\Big(\sum_{k=1}^{\mathrm{n}} a_{i_p k}X_{kj} - \sum_{l=1}^{\mathrm{m}} X_{i_p l}b_{lj}\Big),\end{aligned} \tag{2.1.5}$$

Hesse 矩阵 $\nabla^2 f(\tilde{x})$ 的 $(p,\ q)$ 元素为

$$(\nabla^2 f(\tilde{x}))_{p,q} = \begin{cases} \sum_{i=1}^{n} 2a_{ii_p}^2 + \sum_{j=1}^{m} 2b_{j_p j}^2 - 4a_{i_p i_p} b_{j_q j_q} & (i_p = i_q, j_p = j_q) \\ \sum_{j=1}^{m} 2b_{j_p j} b_{j_q j} - 2a_{i_p i_p}(b_{j_p j_q} + b_{j_q j_p}) & (i_p = i_q, j_p \neq j_q) \\ \sum_{j=1}^{m} 2a_{ii_p} a_{ii_q} - 2b_{j_q j_q}(a_{i_p i_q} + a_{i_q i_p}) & (i_p \neq i_q, j_p = j_q) \\ -2(a_{i_p i_q} b_{j_p j_q} + a_{i_q i_p} b_{j_q j_p}) & (i_p \neq i_q, j_p \neq j_q) \end{cases}, \quad (2.1.6)$$

约束函数（2.1.4）的 Jacobi 矩阵为

$$\nabla c(\tilde{x}) = (2\tilde{x}_1, 2\tilde{x}_2, \cdots\cdots, 2\tilde{x}_{nm}), \quad (2.1.7)$$

Hesse 矩阵为：

$$\nabla^2 c(\tilde{x}) = \mathrm{diag}(2, 2, \cdots\cdots, 2). \quad (2.1.8)$$

2. 传统方法回顾

求解矩阵分离度问题最传统的方法是利用数值代数的方法估计其下界。但这些方法一般效率不高，而且下界估计与其真实值之间可能存在很大的误差。参见[8]、[10]。因此，我们考虑设计优化的方法来求解矩阵分离度问题

[6]中给出了几种优化方法来求解矩阵分离度：Lagrange — Newton 法和乘子罚函数法。数值结果表明，Lagrange — Newton 法和乘子罚函数法对于小规模的计算矩阵分离度问题（$n \times m < 100$）是非常有效的，速度快，而且精度高。然而对于规模较大的问题，Lagrange — Newton 法要生成 $(nm+1)\times(nm+1)$ 维的矩阵，乘子罚函数法也要用到 $nm \times nm$ 维的 Hesse 矩阵，当 nm 较大时，一般 PC 机的内存是无法满足需要的。所以我们有必要针对大规模问题设计占用内存较小的优化算法。其次对于 Lagrange — Newton 法和乘子罚函数法，运算量都比较大 $(nm)^2$，而且算法收敛速度不快，对于大规模的问题，它们在一般 PC 机上的时间代价就会让人无法接受了。所以还有必要针对大规模问题设计运算量较小的算法。

3. 单约束投影梯度法

在第一章中，我们已经简单介绍了投影梯度法的基本思想和一般算法。我们已经可以看到投影梯度算法中，没有出现目标函数的 Hesse 矩阵 $\nabla^2 f(\tilde{x})$，而出现的 $f(x)$ 和 $\nabla f(x)$ 均

可由公式（2.1.3）和（2.1.5）求得，因而算法只需存储原始矩阵 A 和 B 即可。这明显优于传统的 Lagrange — Newton 法和乘子罚函数法。

仅凭上面的分析还不能推断投影梯度法就是解决大规模矩阵分离度问题的好办法。因为投影梯度法的第一步就要对约束函数的 Jacobi 矩阵的转置 $\nabla c(x_k)^{\mathrm{T}}$ 作 QR 分解，并且要分别记录 Q、R 两矩阵的信息以备后用，那么这一步所造成的巨大的运算量和存储空间消耗和我们尝试投影梯度法的初衷是南辕北辙的。

然而通过给出下面几个引理的证明，作者找到了适合计算矩阵分离度问题，乃至所有单约束非线性优化问题的投影梯度法改进算法，在本文中将称为单约束投影梯度法。

对于单约束优化问题

$$\min\ f(x), \tag{2.1.9}$$

$$\text{s.t.}\quad c(x) = 0\,. \tag{2.1.10}$$

对约束函数的 Jacobi 矩阵的转置 $\nabla c(x)^{\mathrm{T}}$ 作 QR 分解，也就等于对 $\nabla c(x)^{\mathrm{T}}$ 作 Householder 变换：

$$\omega=\frac{\nabla c(x)^{\mathrm{T}} - \alpha e_1}{\left\|\nabla c(x)^{\mathrm{T}} - \alpha e_1\right\|_2}, \tag{2.1.11}$$

其中 $e_1 = (1, \underbrace{0, 0, \cdots\cdots, 0}_{(nm\text{-}1)\text{个}})^{\mathrm{T}}$，$\alpha=\left\|\nabla c(x_k)^{\mathrm{T}}\right\|_2$。则 Householder 变换阵 $H = I - 2\omega\omega^{\mathrm{T}}$，容易证明 $\nabla c(x)^{\mathrm{T}} = H\cdot(\alpha e_1)$，于是我们令 $Q = H$，$R = \alpha e_1$，则 $\nabla c(x)^{\mathrm{T}} = QR$，即 Q、R 为 $\nabla c(x)^{\mathrm{T}}$ 的 QR 分解。

引理 1：问题（2.1.9）—（2.1.10）任何一次迭代的既约梯度 $\overline{g}_k = Z_k^{\mathrm{T}}\nabla f(x_k)$，可以用目标函数梯度 $\nabla f(x_k)$ 和（2.1.11）式定义的 ω 来表示。

[证明] 即约梯度 $\overline{g}_k = Z_k^{\mathrm{T}}\nabla f(x_k)$，其中 Z_k 满足：$\begin{bmatrix} Y_k & Z_k \end{bmatrix} = \begin{bmatrix} R_k \\ 0 \end{bmatrix}$ 是 $\nabla c(x_k)^{\mathrm{T}}$ 的 QR 分解；注意到 $\overline{g}_k$ 是 $\tilde{g}_k = \begin{bmatrix} Y_k^{\mathrm{T}} \\ Z_k^{\mathrm{T}} \end{bmatrix}\nabla f(x_k)$ 的第 2 个至第 nm 个元素所组成的子列矩阵。

观察 $\tilde{g}_k$ 可知：$\tilde{g}_k = \begin{bmatrix} Y_k^{\mathrm{T}} \\ Z_k^{\mathrm{T}} \end{bmatrix}\nabla f(x_k) = Q_k^{\mathrm{T}}\nabla f(x_k) = H^{T}\nabla f(x_k) = H\nabla f(x_k)$

$$= (I - 2\omega\omega^{\mathrm{T}})\,\nabla f(x_k) = \nabla f(x_k) - 2\omega\underbrace{\omega^{\mathrm{T}}\nabla f(x_k)}_{\text{一个数}}$$

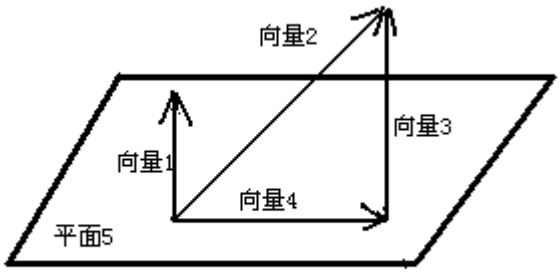
也即 $\overline{g}_k$ 是 $\nabla f(x_k) - 2(\omega^{\mathrm{T}}\nabla f(x_k))\omega$ 的第 2 个至第 nm 个元素所组成的子列矩阵。

（注：从代数上讲，也即：$\overline{g}_k = [0, I_{nm-1}]\cdot(\nabla f(x_k) - 2(\omega^{\mathrm{T}}\nabla f(x_k))\omega)$；

然而从计算的角度反而取子列矩阵的说法易于快速实现。）证毕。

引理 2：问题（2.1.9）—（2.1.10）任何一次迭代的下降方向 $d_k = -Z_k\overline{g}_k$，可以用目标函数梯度 $\nabla f(x_k)$ 和约束函数的 Jacobi 矩阵的转置 $\nabla c(x_k)^{\mathrm{T}}$ 表示。

[证明] 根据投影梯度法的几何意义，d_k 是负梯度在 $\nabla c(x_k)$ 的零空间的投影，由于 $\nabla c(x_k)$ 的零空间是 R^{nm} 空间中的一个 $nm-1$ 维超平面，所以我们可根据几何定义，直接用 $\nabla f(x_k)$ 与 $\nabla c(x_k)^{\mathrm{T}}$ 来表示 d_k。如下图所示：



附图 1

其中，向量 1 表示 $\nabla c(x_k)^{\mathrm{T}}$，向量 2 表示 $-\nabla f(x_k)$，平面 5 表示 $\nabla c(x_k)$ 的零空间；我们知道向量 1 垂直平面 5，过向量 2 的顶端点作平面 5 的垂线，可以得到向量 3（设为 h），连接向量 2 和向量 3 尾端点可以得到向量 4，显然，向量 4 也就是我们要求的投影 d_k。我们容易知道向量 3 和向量 1 是平行的，向量 2 和向量 4 的夹角 θ 就是向量 2 和平面 5 的夹角，利用以上两条结论我们就可以计算出：

$$\cos\theta = \frac{\left\langle -\nabla f(x_k),\ \nabla c(x_k)^{\mathrm{T}} \right\rangle}{\left\| -\nabla f(x_k) \right\|_2 \left\| \nabla c(x_k)^{\mathrm{T}} \right\|_2},$$

$$h=(\left\| -\nabla f(x_k) \right\|_2 \cos\theta)\frac{\nabla c(x_k)^{\mathrm{T}}}{\left\| \nabla c(x_k)^{\mathrm{T}} \right\|_2},$$

$$d_k = -\nabla f(x_k) - h$$

也即 $d_k = -\nabla f(x_k) + 2\dfrac{\left\langle \nabla f(x_k),\ \nabla c(x_k)^{\mathrm{T}} \right\rangle}{\left\| \nabla c(x_k)^{\mathrm{T}} \right\|_2^2}\nabla c(x_k)^{\mathrm{T}}$。证毕。

引理 3：问题(2.1.9)—(2.1.10)任何一次迭代最后的近似拟牛顿法公式：$y := y - Y_k R_k^{-1} c(y)$

可以写成与 QR 分解矩阵无关的形式。

[证明] $y := y - Y_k R_k^{-1} c(y)$，其中 Y_k、R_k 满足 $\begin{bmatrix} Y_k & Z_k \end{bmatrix} \begin{bmatrix} R_k \\ 0 \end{bmatrix}$ 是 $\nabla c(y)^{\mathrm{T}}$ 的 QR 分解。注意到 R_k 是一个数，即 $\alpha = \|c(y)\|_2$，故而 $y := y - Y_k R_k R_k^{-2} c(y)$，而 $Y_k R_k$ 即为 $\nabla c(y)^{\mathrm{T}}$，所以 $y := y - \dfrac{c(y)}{\|c(y)\|_2^2} \nabla c(y)^{\mathrm{T}}$。证毕。

根据上面三个引理的证明，我们可以知道，对于单约束的非线性优化问题（2.1.9）—（2.1.10），我们可以给出改进的投影梯度法，不但不需要进行一般化的 QR 分解，而且不用存储 QR 分解矩阵，甚至连 Householder 变换阵都不用存储。综合前文提到的投影梯度法不用存储目标函数 Hesse 矩阵信息，从存储的角度来说，单约束投影梯度法无疑是适用于大规模矩阵分离度计算的好算法。而且我们不难计算，单约束投影梯度法的运算量是 $nm(n+m)$，这也比传统的 Lagrange — Newton 法和乘子罚函数法要好。下面给出具体算法：

算法 2.1：单约束投影梯度法

步 1: 给出初始可行点 x_1，算法终止变量 $0 \le \varepsilon \ll 1$，

用 Newton 法使下一迭代点回归可行域的检验参数 $\bar{\varepsilon} > 0$，
用 Newton 法使下一迭代点回归可行域的最多迭代步数 N（正整数），

非精确线性搜索准则参数 $0 < b_1 \le b_2 < 1$，

迭代次数 $k := 1$

步 2: 对 $\nabla c(x_k)^{\mathrm{T}}$ 作 Householder 变换，如下：

令 $e_1 = (1, \underbrace{0, 0, \cdots\cdots, 0}_{(nm-1)\text{个}})^{\mathrm{T}}$ 是单位矩阵的首列

计算 $\omega = \dfrac{\nabla c(x_k)^{\mathrm{T}} - \alpha e_1}{\left\| \nabla c(x_k)^{\mathrm{T}} - \alpha e_1 \right\|_2}$，其中 $\alpha = \left\| \nabla c(x_k)^{\mathrm{T}} \right\|_2$

步 3: 计算 $\beta = \omega^{\mathrm{T}} \nabla f(x_k)$，计算 $\tilde{g}_k = \nabla f(x_k) - 2\beta\omega$，

取即约梯度 $\bar{g}_k$ 为 $\tilde{g}_k$ 的第 2 个至第 nm 个元素所组成的子列矩阵。

步 4: 如果 $\|\bar{g}_k\|_2 \le \varepsilon$，则算法终止；

否则，$d_k = -\nabla f(x_k) + 2\gamma \nabla c(x_k)^{\mathrm{T}}$，其中，$\gamma = \left\langle \nabla f(x_k), \nabla c(x_k)^{\mathrm{T}} \right\rangle / \alpha^2$。

步 5: 取初始步长 $\lambda_k = 0.2$。

步 6: 检验 λ_k 是否满足非精确线性搜索准则：

$$f(x) - f(x+\lambda_k d) \geq -\lambda_k b_1 d^{\mathrm{T}} \nabla f(x)$$

$$d^{\mathrm{T}} \nabla f(x+\lambda_k d) \geq b_2 d^{\mathrm{T}} \nabla f(x)$$

满足则转步 7；不满足，$\lambda_k = \frac{\lambda_k}{2}$，转步 6。

步 7: $y := x_k + \lambda_k d_k$； $i := 0$

步 8: $y := y - \frac{c(y)}{\alpha^2} \nabla c(y)^{\mathrm{T}}$，其中，$\alpha = \left\| \nabla c(y)^{\mathrm{T}} \right\|_2$。

如果 $\left\| c(y) \right\|_2 \leq \bar{\varepsilon}$，转步 9；否则，$i := i+1$，

如果 $i < N$，转步 8，否则，$\lambda = \frac{\lambda}{2}$，转步 7。

步 9: $x_{k+1} = y$，$k := k+1$，转步 2。

4. 单约束投影 BB 方法

为了提高投影梯度法的效率，作者尝试用 BB 步长取法来代替非精确线性搜索。由于投影梯度方法本来就不是真正的梯度型方法，为了易于理解改进的合理性，并且从节省空间消耗和提高程序的可读性方面考虑，我们将不采用 BB 方法的最原始定义（1.2.11）和（1.2.12）来计算 BB 步长，而是用 BB 法对于二次型问题特有的实际含义，上一次迭代的最大梯度值下降 $\arg\min_{\alpha} g(x_{k-1} + \alpha d_{k-1})$ 和上一次迭代的最大函数值下降 $\arg\min_{\alpha} f(x_{k-1} + \alpha d_{k-1})$ 来计算 BB 步长，即（1.2.13）和（1.2.14）来计算。

在投影梯度法中下降方向 d_k 就是负既约梯度方向，所以根据（1.2.13）或（1.2.14），$\lambda_k = d_{k-1}^{\mathrm{T}} A^* d_{k-1} / d_{k-1}^{\mathrm{T}} A^{*2} d_{k-1}$ 或 $\lambda_k = d_{k-1}^{\mathrm{T}} d_{k-1} / d_{k-1}^{\mathrm{T}} A^* d_{k-1}$ 就是相应的 BB 步长，其中 A^* 应该是目标函数的 Hesse 矩阵，这时我们不希望存储的。稍加分析后，我们发现 A^* 都是以 $A^* d_{k-1}$ 的形式出现在步长公式中的，而 $A^* d_{k-1} = \nabla f(d_{k-1})$，用 $\nabla f(d_{k-1})$ 来代替 $A^* d_{k-1}$，就完全可以避免存储目标函数的 Hesse 矩阵 A^*。

对于单约束问题，我们仍旧要用到前面提到的三个引理来改进算法以减小算法因 QR 分解而造成的空间消耗。为保证方法的收敛性，我们需要使用修正的 BB 方法，也就是当使用 BB 步长 4 次，迭代点还未下降，我们要退回 4 次以前的迭代点作线性搜索。下面给出算法：

算法 2.2：单约束投影 BB 法

步 1: 给出初始可行点 x_1，算法终止变量 $0 \le \varepsilon \ll 1$，

用 Newton 法使下一迭代点回归可行域的检验参数 $\bar{\varepsilon}>0$，
用 Newton 法使下一迭代点回归可行域的最多迭代步数 N （正整数），

非精确线性搜索准则参数 $0<b_1 \le b_2<1$，

$k:=1$， $index:=1$， $f_{\min}=f(x_1)$；

步 2: 对 $\nabla c(x_k)^{\mathrm{T}}$ 作 Householder 变换，如下：

（注：本题中，由于约束函数只有一个，因此， $\nabla c(x_k)^{\mathrm{T}}$ 是一列向量；两个 ∇ 算符含义并不完全相同，前者是求 Jacobi 矩阵，后者是求梯度。）

令 $e_1=(1,\ \underbrace{0,0,\cdots\cdots,0}_{(nm-1)\text{个}})^{\mathrm{T}}$ 是单位矩阵的首列

计算 $\omega=\dfrac{\nabla c(x_k)^{\mathrm{T}}-\alpha e_1}{\left\|\nabla c(x_k)^{\mathrm{T}}-\alpha e_1\right\|_2}$，其中 $\alpha=\left\|\nabla c(x_k)^{\mathrm{T}}\right\|_2$

步 3: 计算 $\beta=\omega^{\mathrm{T}}\nabla f(x_k)$ ，计算 $\tilde{g}_k=\nabla f(x_k)-2\beta\omega$，

取即约梯度 $\bar{g}_k$ 为 $\tilde{g}_k$ 的第 2 个至第 nm 个元素所组成的子列矩阵。

步 4: 如果 $\left\|\bar{g}_k\right\|_2 \le \varepsilon$，则算法终止；

否则， $d_k=-\nabla f(x_k)+2\gamma\nabla c(x_k)^{\mathrm{T}}$，其中， $\gamma=\left\langle\nabla f(x_k),\ \nabla c(x_k)^{\mathrm{T}}\right\rangle / \alpha^2$。

步 5: 如果 $k=1$ 或者 $k-index=4$（用 BB 方法 4 步未降函数值，则表示该点不适合 BB 步长），则转步 6；否则计算 $\lambda_k=d_{k-1}^{\mathrm{T}}d_{k-1}/d_{k-1}^{\mathrm{T}}\nabla f(d_{k-1})$ 如果， $\lambda_k \le 0$ 或者无定义，转步 6；否则转步 7。

步 6: $x_k:=x_{index}$； $index:=k$；利用非精确线性搜索准则：

$$f(x)-f(x+\lambda_k d) \ge -\lambda_k b_1 d^{\mathrm{T}}\nabla f(x)$$

$$d^{\mathrm{T}}\nabla f(x+\lambda_k d) \ge b_2 d^{\mathrm{T}}\nabla f(x) \qquad \text{求：}\ \lambda_k>0\text{。}$$

步 7: $y:=x_k+\lambda_k d_k$； $i:=0$

步 8: $y:=y-\dfrac{c(y)}{\alpha^2}\nabla c(y)^{\mathrm{T}}$，其中， $\alpha=\left\|\nabla c(y)^{\mathrm{T}}\right\|_2$。

如果 $\left\|c(y)\right\|_2 \le \bar{\varepsilon}$，转步 9；否则， $i:=i+1$，

如果$i < N$，转步 8，否则，$\lambda = \frac{\lambda}{2}$，转步 7。

步 9: $x_{k+1} = y$，$k := k+1$；如果 $f(x_k) \geq f_{\min}$，则转步 2；

否则$index := k$，$f_{\min} = f(x_k)$，转步 2。

第二节 数值结果

这一节中我们将给出 Lagrange — Newton 法、乘子罚函数法和投影梯度法、投影 BB 方法的数值结果。

数值实验在：Inter(R) Celeron(TM) CPU 1.2GHz，256MHz 的内存的计算机上用 Visual C++6.0 完成。

1. 投影方法与传统方法的比较

我们的系数矩阵 A 和 B 由 C++语言的伪随机机制生成，下面的结果都是对种子值分别取 0、1、10、99、527 生成的 A 和 B 分别运行后五次的平均值。初始点都取 $X = 2I_{n\times m}$。

由于经典算法和投影 BB 算法在结构上有较大区别，我们比较它们的迭代次数没有意义，所以我们比较它们各自的机器运算时间。

Lagrange — Newton 法：

Lagrange 乘子：$\lambda_1 = 1$；

算法终止变量：$\varepsilon = 10^{-4}$。

n	m	约束误差值	时间(s)
1	2	0.003272	0.024000
2	2	0.002520	0.112000
2	5	0.001331	0.436600
5	5	0.001439	3.198600
7	8	0.000598	11.690800
10	10	0.003301	35.891000
20	20	—	—

表 1：Lagrange—Newton 法数值结果

乘子罚函数法：

Lagrange：$\lambda_1 = 1$；

初始罚因子：$\sigma_1 = 1$；

算法终止变量：$\varepsilon = 10^{-5}$。

n	m	约束误差值	时间(s)
1	2	0.000057	0.284400
2	2	0.000057	0.578800
2	5	0.000061	1.047600
5	5	0.000062	4.220200
7	8	0.000062	13.740000
10	10	0.000063	47.328000
20	20	—	—

表 2：乘子罚函数法数值结果

投影梯度方法：

算法终止变量：$\varepsilon = 10^{-5}$；

约束终止变量：$\bar{\varepsilon} = 10^{-6}$， $N = 1000$；

其它参数：$b_1 = 0.2$， $b_2 = 0.4$。

n	m	约束误差值	时间(s)
1	2	0.000000	0.042600
2	2	0.000000	0.214400
2	5	0.000000	0.829200
5	5	0.000000	4.128000
7	8	0.000000	11.980200
10	10	0.000000	35.262200
20	20	0.000000	974.226000

表 3：投影梯度法数值结果

投影 BB 方法：

算法终止变量：$\varepsilon = 10^{-5}$；

约束终止变量：$\bar{\varepsilon} = 10^{-6}$， $N = 1000$；

其它参数：$b_1 = 0.2$， $b_2 = 0.4$。

n	m	约束误差值	时间
1	2	0.000000	0.005400
2	2	0.000000	0.032400

2	5	0.000000	0.110800
5	5	0.000000	0.494400
7	8	0.000000	2.422800
10	10	0.000000	4.937200
20	20	0.000000	311.748000

表 4：投影 BB 法数值结果

结论：我们发现投影梯度方法和投影 BB 方法比 Lagrange—Newton 法和乘子罚函数法要快很多；而且当 $nm>100$ 时，Lagrange—Newton 法和乘子罚函数法就面临存储困难了；此外，投影方法能够很好地保证约束条件而不增加时间负担。

2. 投影梯度法与投影 BB 方法的比较

我们的系数矩阵 A 和 B 由 C++语言的伪随机机制生成，下面的结果都是对种子值分别取 0、1、10、99、527 生成的 A 和 B 分别运行后五次的平均值。初始点都取 $X=2I_{n\times m}$。算法终止变量：$\varepsilon=10^{-3}$；约束终止变量：$\bar{\varepsilon}=10^{-6}$，$N=1000$；其它参数：$b_1=0.2$，$b_2=0.4$。

n	m	投影梯度方法		投影 BB 方法	
		时间(s)	迭代次数	时间(s)	迭代次数
5	5	0.892200	1801.2	0.0712000	149
7	8	8.010200	13604.6	0.923800	823.4
10	10	10.891400	12972.2	0.359400	50.8
10	20	372.296800	158920.6	4s	1487.2
20	20	113.928600	13728.8	11s	482.4
30	30	About 11m	41026	About 3m	1772
50	50	—	—	About 20m	1098.4
80	80	—	—	3.5h-4h	2455.8

表 5：投影梯度法与投影 BB 法在大规模问题中的比较

结论：投影 BB 方法比投影梯度方法快很多。

第三章 用优化方法求解 Sylvester 方程

第一节 算法设计

1. 传统方法的困难

Lyapunov 方程（1.1.2）的传统代数求解方法，效率不高，是因为它牵涉到要做系统矩阵 A 的特征值分析，所以一般只适合做小规模的定性分析。参见[4]。而对于一般 Sylvester 方程（1.1.5）的求解传统代数没有直接而有效的方法。

一般而言，数值代数中会采取下面方法对方程（1.1.5）进行变换：

若记 $X = (X_{ij})$，先把 X 的各元素按 j 由小到大排列，j 相同的再按 i 由小到大排列，这种排列方式我们称之为“自然顺序排列”。用自然顺序排列后能得到如下线性方程组：

$$D\tilde{x} = \tilde{c}, \qquad (3.1.1)$$

其中 $D = f(A,B) \in \mathrm{R}^{n\times m}$，$\tilde{x}$ 是 X 经过自然排序后得到的 nm 维列向量，$\tilde{c}$ 是 C 经过自然排序后得到的 nm 维列向量。这样，求解 Sylvester 方程（1.1.5）的问题就转化成求解经典线性方程组（3.1.1）的形式了。然而由于问题的规模较大，当 nm 比较大的时候，存储 $nm\times nm$ 阶系数矩阵 D 就成了很大的困难，而且随着问题规模的增大，即使 D 具有特殊的形式能够存储，算法的时间代价也不容忽视。所以当今数值代数界，对于 Sylvester 方程（1.1.5）更愿意寻求的直接的数值解法，而不通过变换（3.1.1）。

于是，为了解决传统代数方法的困难，我们希望设计一种不需要存储系数矩阵的优化算法，并且它的运算量要适合大规模问题，而且收敛速度快。

由于来源实际问题中的 Lyapunov 方程，或是其它形式的 Sylvester 方程，往往会有比较特殊的形式（比如说三对角、块对角等）。这时我们期望所设计的算法应该能够被稍加改动就从时间代价和空间代价上都更针对特殊形式。

将线性方程组（3.1.1）转化为等价的二次函数求极小的方法：

$$\min f(\tilde{x}) = \frac{1}{2}\tilde{x}D\tilde{x} - \tilde{c}\tilde{x}$$

就是一种将 Sylvester 方程化为优化问题的方法。但是在算梯度 $\nabla f(x)$ 时需要系数矩阵 D 的信息，和传统数值代数方法一样面临着存储问题，不能够计算大规模问题。我们称这种转化为**传统优化转化**。

下面作者将给出另一个将求解 Sylvester 方程转化成优化问题的方法，建立在该转化基础上的一系列优化算法都适合大规模问题。但是在小规模问题上，它们的时间代价比对应的建立在传统优化转化上的算法高。

2. 问题的转化

根据 Sylvester 方程的定义（1.1.5）我们容易地对它进行如下变形：

$$AX - XB - C = 0. \tag{3.1.2}$$

我们观察如下优化问题：

$$\min \left\| AX - XB - C \right\|_{\mathrm{F}}. \tag{3.1.3}$$

我们知道（3.1.2）有解，等价于（3.1.3）的最优值是 0，并且此时（3.1.3）的最优解和（3.1.2）的解是互相等价的；如果（3.1.2）无解，则等价于（3.1.3）的最优值大于 0。因此，我们将试图通过求解优化问题（3.1.3）来求解（3.1.2）。

特别值得注意的是：我们用优化方法求解（3.1.3）时，并不能求得它所有的最优解，这和所取的初始点有关。作者在此声明，本章所讨论的求 Sylvester 方程的解，只限于（3.1.2）有解时，求它的一个解。至于（3.1.2）是否有解，以及它的解的个数等问题，我们将在第四章中详细讨论。

我们记：$X=(X_{ij})$，$A=(a_{ij})$，$B=(b_{ij})$，$C=(c_{ij})$。做线性变换 T：$\mathrm{R}^{n\times m}\to\mathrm{R}^{nm}$，设 $TX=\tilde{x}\in\mathrm{R}^{nm}$，满足：$X_{ij}=\tilde{x}_{(j-1)n+i}$，或者 $\tilde{x}_p=X_{i_p j_p}$，其中 $i_p=\mathrm{mod}(p-1,\ n)+1$，$j_p=(p-i_p)/n+1$。于是上面关于矩阵的优化问题（3.1.3）可以转化成下面一般的无约束优化问题：

$$\begin{aligned}\min f(\tilde{x}) &:= \left\| AX - XB - C \right\|_{\mathrm{F}}^2 \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m}\Big(\sum_{k=1}^{n}a_{ik}X_{kj} - \sum_{l=1}^{m}X_{il}b_{lj} - c_{ij}\Big)^2.\end{aligned} \tag{3.1.4}$$

这样，我们 Sylvester 方程求解问题转化成了一个无约束二次函数问题，问题的规模是 nm 维。我们称这种转化方法为**新优化转化**。

3. 解决方案

我们可以给出（3.1.4）式的目标函数的梯度向量为

$$(\nabla f(\tilde{x}))_p = \sum_{i=1}^{\mathrm{n}} 2a_{ii_p}\Big(\sum_{k=1}^{\mathrm{n}}a_{ik}X_{kj_p} - \sum_{l=1}^{\mathrm{m}}X_{il}b_{lj_p} - c_{ij_p}\Big)$$

$$+\sum_{j=1}^{m}\Big(\sum_{k=1}^{n}a_{i_pk}X_{kj}-\sum_{l=1}^{m}X_{i_pl}b_{lj}-c_{i_pj}\Big),\qquad(3.1.5)$$

Hesse 矩阵为

$$(\nabla^2 f(\tilde{x}))_{p,q}=\begin{cases}\sum_{i=1}^{n}2a_{ii_p}^2+\sum_{j=1}^{m}2b_{j_pj}^2-4a_{i_pi_p}b_{j_qj_q} & (i_p=i_q,j_p=j_q)\\ \sum_{j=1}^{m}2b_{j_pj}b_{j_qj}-2a_{i_pi_p}(b_{j_pj_q}+b_{j_qj_p}) & (i_p=i_q,j_p\neq j_q)\\ \sum_{j=1}^{m}2a_{ii_p}a_{ii_q}-2b_{j_qj_q}(a_{i_pi_q}+a_{i_qi_p}) & (i_p\neq i_q,j_p=j_q)\\ -2(a_{i_pi_q}b_{j_pj_q}+a_{i_qi_p}b_{j_qj_p}) & (i_p\neq i_q,j_p\neq j_q)\end{cases}.\qquad(3.1.6)$$

对于大规模无约束二次型求极小问题，第一章中我们已经介绍过用梯度型方法比较省时间，所以我们的解决方案就是对（3.1.4）用各种梯度型方法，并将在下一节中对各种梯度型方法求解这个问题的结果进行比较。

对于大规模问题我们要求所设计的算法不能需要存储 Hesse 矩阵的信息，可是在用梯度型算法求步长时不可避免地会出现（1.2.13）和（1.2.14）的形式：

$$\alpha_k=\mathrm{g}_{k-1}^{\mathrm{T}}A^*\mathrm{g}_{k-1}/\mathrm{g}_{k-1}^{\mathrm{T}}A^{*2}g_{k-1}$$

和

$$\alpha_k=\mathrm{g}_{k-1}^{\mathrm{T}}\mathrm{g}_{k-1}/\mathrm{g}_{k-1}^{\mathrm{T}}A^*g_{k-1}$$

其中：A^* 就是 Hesse 矩阵。为了不使用 Hesse 矩阵，我们就要用到前一章介绍的技巧，用已有的函数和存储信息来代替 Hesse 矩阵：我们发现 A^* 都是以 A^*d_{k-1} 的形式出现在步长公式中的，而 $A^*d_{k-1}=\nabla\overline{f}(d_{k-1})$，用 $\nabla\overline{f}(d_{k-1})$ 来代替 A^*d_{k-1}，就完全可以避免存储目标函数的 Hesse 矩阵 A^*。这里的 $\overline{f}(x)$ 是前一章（2.1.3）—（2.1.4）定义的目标函数。我们能这么做是因为由（2.1.3）—（2.1.4）和（3.1.4）定义的目标函数的 Hesse 矩阵是相同的，而在前一章中我们已经讲过 A^*g 就是 $\nabla\overline{f}(g)$。而 $\nabla\overline{f}(x)$ 的计算公式也完全按照（2.1.5）：

$$(\nabla\overline{f}(\tilde{x}))_p=\sum_{i=1}^{n}2a_{ii_p}\Big(\sum_{k=1}^{n}a_{ik}X_{kj_p}-\sum_{l=1}^{m}X_{il}b_{lj_p}\Big)+\sum_{j=1}^{m}\Big(\sum_{k=1}^{n}a_{i_pk}X_{kj}-\sum_{l=1}^{m}X_{i_pl}b_{lj}\Big).\qquad(3.1.7)$$

此外，对于特殊的 Sylvester 方程，比如系数矩阵 A 和 B 是三对角矩阵，我们可以在算法的程序里将求函数值、梯度值的（3.1.4）、（3.1.5）和（3.1.7）各函数计算方法化简，可以把三对角元素的值定义为常量，这样连存储系数矩阵的空间也可省去了。我们的程序能够很容易实现对特殊问题的改进，是因为在算法中所有需要用到系数矩阵 A 和 B 信息的地方全部通过和（3.1.4）、（3.1.5）及（3.1.7）挂钩而不是和求目标函数 Hesse 矩阵的函数（3.1.6）挂钩。

下面简要列出本文中将应用的所有梯度型算法：（它们的差别在步 3）

算法 3.1：最速下降法（SD）

步 1：给定初始点 $x_0 \in \mathrm{R}^{nm}$，算法终止变量 $0 \le \varepsilon \ll 1$，迭代步数 $k := 0$。

步 2：如果 $\|g_k\| \le \varepsilon$ 则停；否则取下降方向 $d_k = -g_k$，转步 3。

步 3：取步长 $\alpha_k = g_k^{\mathrm{T}} g_k / g_k^{\mathrm{T}} \nabla f'(g_k)$。

步 4：$x_{k+1} = x_k + \alpha_k d_k$，$k := k+1$，转步 2。

算法 3.2：最小梯度法（MG）

步 1：给定初始点 $x_0 \in \mathrm{R}^{nm}$，算法终止变量 $0 \le \varepsilon \ll 1$，迭代步数 $k := 0$。

步 2：如果 $\|g_k\| \le \varepsilon$ 则停；否则取下降方向 $d_k = -g_k$，转步 3。

步 3：取步长 $\alpha_k = g_k^{\mathrm{T}} \nabla f'(g_k) / \nabla f'(g_k)^{\mathrm{T}} \nabla f'(g_k)$。

步 4：$x_{k+1} = x_k + \alpha_k d_k$，$k := k+1$，转步 2。

算法 3.3：由（1.10）决定的 BB 方法（BB1）

步 1：给定初始点 $x_0 \in \mathrm{R}^{nm}$，算法终止变量 $0 \le \varepsilon \ll 1$，迭代步数 $k := 0$。

步 2：如果 $\|g_k\| \le \varepsilon$ 则停；否则取下降方向 $d_k = -g_k$，转步 3。

步 3：如果 $k = 0$，取步长 $\alpha_k = g_k^{\mathrm{T}} g_k / g_k{}^{\mathrm{T}} \nabla f'(g_k)$

否则，取步长 $\alpha_k = g_{k-1}^{\mathrm{T}} g_{k-1} / g_{k-1}^{\mathrm{T}} \nabla f'(g_{k-1})$。

步 4：$x_{k+1} = x_k + \alpha_k d_k$，$k := k+1$，转步 2。

算法 3.4：由（1.9）决定的 BB 方法（BB2）

步 1：给定初始点 $x_0 \in \mathrm{R}^{nm}$，算法终止变量 $0 \le \varepsilon \ll 1$，迭代步数 $k := 0$。

步 2：如果 $\|g_k\| \le \varepsilon$ 则停；否则取下降方向 $d_k = -g_k$，转步 3。

步 3：如果 $k = 0$，取步长 $\alpha_k = g_k^{\mathrm{T}} \nabla f'(g_k) / \nabla f'(g_k)^{\mathrm{T}} \nabla f'(g_k)$

否则，取步长 $\alpha_k = g_{k-1}^{\mathrm{T}} \nabla f'(g_{k-1}) / \nabla f'(g_{k-1})^{\mathrm{T}} \nabla f'(g_{k-1})$。

步 4：$x_{k+1} = x_k + \alpha_k d_k$，$k := k+1$，转步 2。

算法 3.5：交替步长方法（AS）

步 1：给定初始点 $x_0 \in \mathrm{R}^{nm}$，算法终止变量 $0 \le \varepsilon \ll 1$，迭代步数 $k := 0$。

步 2：如果 $\|g_k\| \le \varepsilon$ 则停；否则取下降方向 $d_k = -g_k$，转步 3。

步 3：如果 k 是偶数，取步长 $\alpha_k = g_k^{\mathrm{T}} g_k / g_k{}^{\mathrm{T}} \nabla f'(g_k)$

否则，取步长 $\alpha_k = g_{k-1}^{\mathrm{T}} g_{k-1} / g_{k-1}^{\mathrm{T}} \nabla f'(g_{k-1})$。

步 4：$x_{k+1} = x_k + \alpha_k d_k$，$k := k+1$，转步 2。

算法 3.6：交替最小方法（AM）

步 1：给定初始点 $x_0 \in \mathrm{R}^{nm}$，算法终止变量 $0 \le \varepsilon \ll 1$，迭代步数 $k := 0$。

步 2：如果 $\|g_k\| \le \varepsilon$ 则停；否则取下降方向 $d_k = -g_k$，转步 3。

步 3：如果 k 是偶数，取步长 $\alpha_k = g_k^{\mathrm{T}} \nabla f'(g_k) / \nabla f'(g_k)^{\mathrm{T}} \nabla f'(g_k)$

否则，取步长 $\alpha_k = g_k^{\mathrm{T}} g_k / g_k{}^{\mathrm{T}} \nabla f'(g_k)$。

步 4：$x_{k+1} = x_k + \alpha_k d_k$，$k := k+1$，转步 2。

第二节　数值结果

1. 新优化转化与传统优化转化的比较：

我们的系数矩阵 A 和 B 由 C++语言的伪随机机制生成，下面的结果都是对种子值分别取 0、1、10、99、527 生成的 A 和 B 分别运行后五次的平均值。初始点都取 $X = 2I_{n\times m}$ 算法终止变量：$\varepsilon = 10^{-6}$。基于两种转化的优化算法都是 BB1 方法。

对于同样的 Sylvester 方程，通过两种转化所导致的优化问题不一样，所以比较迭代次数是没有意义的，我们比较基于两种转化的算法的机器运算时间。

时间(s)	n=2, m=2	n=2, m=5	n=5, m=5
传统优化转化	0.008000	0.061200	0.523400
新优化转化	0.061000	0.472800	3.871200

时间(s)	n=7, m=8	n=10, m=10	n=20, m=20
传统优化转化	2.244800	2.821600	无法存储
新优化转化	23.804800	53.645000	1618.020000

表 6：两种优化转化在解 Sylvester 方程中的比较

结论：对小规模问题：$nm < 360$，传统优化转化的速度快于新优化转化；但当 $nm \geq 360$ 时，传统优化转化就无法存储了。对于大规模的取随机矩阵的一般问题，新优化转化的速度非常缓慢。

2. 梯度形各方法的比较

为比较建立在新优化转化方法上的各梯度型方法的结果，我们选用大规模有特殊形式的问题：n=m，此时，问题的规模为 n^2 维。

系数矩阵：

$$A_{n\times n}=\begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix},\quad B_{n\times n}=\begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix},\quad C_{n\times n}=\frac{1}{n^2}\begin{bmatrix} -1 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & -1 \end{bmatrix}$$

初始点：$X = I_{n\times n}$。

迭代次数	算法终止变量：$\varepsilon = 10^{-6}$				10^{-3}	10^{-2}		10^{-1}
	n=5	n=10	n=20	n=30	n=40	n=50	n=80	n=100
SD	1468	15574	—	—	—	—	—	—
MG	1538	15436	—	—	—	—	—	—
BB1	126	351	1895	3537	4213	1043	1894	432
BB2	124	483	1672	3734	4866	1291	2354	446
AS	63	359	1854	3124	4145	912	1507	306
AM	146	790	4652	8010	9288	2374	4208	900
时间(s)	<1	前 2<10 其它<1	<10	<30	<30	<30	<60	<60

表 7：梯度型各方法在解大规模 Sylvester 方程中的比较

结论：SD 和 MG 方法从收敛速度上来讲不适合大规模 Sylvester 方程。而两类 BB 方法和 AS 方法对大规模 Sylvester 方程的计算差别不大；AM 方法就稍微差点。但总而言之，后四种方法都适合大规模具有特殊形式的 Sylvester 方程计算。

第四章 问题分析与应用

第一节 矩阵分离度与 Sylvester 方程的关系

为了探索 Sylvester 方程解的结构，我们来考察矩阵分离度和 Sylvester 方程的关系。为了使我们的分析更为明了，我们会把前面的一些定义再写一边。

我们再来看一遍矩阵分离度的定义：

$$\mathrm{Sep}(A,\ B) = \min\{\ \|AX\ -\ XB\|_{\mathrm{F}}\ ,\ X \in \mathrm{R}^{n\times m}\ ,\ \|X\|_{\mathrm{F}} = 1\ \}. \qquad (4.1.1)$$

把它写成明显的优化形式：

$$\begin{aligned} &\min\ \|AX\ -\ XB\|_{\mathrm{F}} \\ &\text{s.t.}\ \ \|X\|_{\mathrm{F}}\ -\ 1 = 0 \end{aligned}. \qquad (4.1.2)$$

对于 Sylvester 方程：

$$AX\ -\ XB = C. \qquad (4.1.3)$$

我们在上一章中已经提到了它和如下优化问题的一些关系：

$$\min\ \|AX\ -\ XB\ -\ C\|_{\mathrm{F}}, \qquad (4.1.4)$$

我们知道（4.1.3）有解，等价于（4.1.4）的最优值是 0，并且此时（4.1.4）的最优解和（4.1.3）的解是互相等价的；如果（4.1.3）无解，则等价于（4.1.4）的最优值大于 0。

为了弄清楚整个脉络，我们先观察特殊的 Sylvester 方程：

$$AX\ -\ XB = 0. \qquad (4.1.5)$$

和它的优化转换：

$$\min\ \|AX\ -\ XB\|_{\mathrm{F}}. \qquad (4.1.6)$$

的关系。由于 $X = 0$ 明显是（4.1.5）和（4.1.6）的共同解，所以我们考虑（4.1.5）和（4.1.6）时，总希望考虑它们的非零解。这时（4.1.6）式其实就等于（4.1.2）式了，所以我们可以不加证明的得到下面的定理 2：

定理 2： 矩阵分离度 $\mathrm{Sep}(A,\ B) = 0$ 等价于 Sylvester 方程 $AX\ -\ XB = 0$ 有非零解。

推论 1： 矩阵分离度 $\mathrm{Sep}(A,\ B) \neq 0$ 时，Sylvester 方程 $AX\ -\ XB = C$（$C \neq 0$），有唯一解。

[证明] 由定理 2 我们知道，矩阵分离度 $\mathrm{Sep}(A,\ B) \neq 0$ 时，Sylvester 方程 $AX\ -\ XB = 0$ 没有非零解。由第三章我们知道 Sylvester 方程 $AX\ -\ XB = 0$ 和 Sylvester 方程

$AX - XB = C$ 分别对 X 用自然顺序排列变换后能得到如下线性方程组：

$D\tilde{x} = 0$ 和 $D\tilde{x} = \tilde{c}(\tilde{c} \neq 0)$。由于 $AX - XB = 0$ 没有非零解，所以 $D\tilde{x} = 0$ 没有非零解，于是 D 满秩。而因为 D 满秩，我们可以推出 $D\tilde{x} = \tilde{c}(\tilde{c} \neq 0)$ 只有唯一解，因此 Sylvester 方程 $AX - XB = C$ 只有唯一解。

推论 1 和它的证明揭示出了矩阵分离度和 Sylvester 方程的关系好比行列式和一般线性方程组的关系：

行列式： $\|A\|$		**矩阵分离度：** Sep(A, B)	
为零	**不为零**	**为零**	**不为零**
线性方程组 $Ax = 0$ **有非零解**	**线性方程组** $Ax = 0$ **无非零解**	**Sylvester 方程** $AX - XB = 0$ **有非零解**	**Sylvester 方程** $AX - XB = 0$ **无非零解**
线性方程组 $Ax = b(b \neq 0)$ 无解或有无穷解 要看 $\text{rank}(A)$ 和 $\text{rank}(A,b)$ 的关系	**线性方程组** $Ax = b(b \neq 0)$ **有唯一解**	Sylvester 方程 $AX - XB = C$ 无解或有无穷解 要看 $\text{rank}(D)$ 和 $\text{rank}(D,\tilde{c})$ 的关系	**Sylvester 方程** $AX - XB = C$ $C \neq 0$ **有唯一解**

表 8：行列式与矩阵分离度之类比

推论 1 还保证了当矩阵分离度不为零的时候，（4.1.3）和（4.1.4）是完全等价的。也就是说当矩阵分离度不为零的时候，上一章所讨论的算法等价于完全求解了 Sylvester 方程。

第二节 在椭圆型方程中的应用

1. 椭圆型方程的有限差分方法

我们考虑求解单位正方形区域的 Poisson 方程第一边值问题：

$$\begin{cases} \Delta u = \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = -f(x, y), \quad 0 < x, y < 1 \\ u\big|_{\Gamma} = \phi \end{cases}. \tag{4.2.1}$$

其中Γ为正方形区域的边界。我们采用五点差分格式来解方程（4.1.7）。

我们取正方形网格均分单位正方形区域为n^2个小正方形，即网格步长取：

$$h_1 = h_2 = h = \frac{1}{n}: \begin{cases} x = ih, & i = 1, 2, \ldots, n-1 \\ y = jh, & j = 1, 2, \ldots, n-1 \end{cases}$$

用二阶差商：

$$\frac{\partial^2 u}{\partial x^2}\Big|_{(x_i, y_j)} = \frac{1}{h^2}[u(x_{i-1}, y_j) - 2u(x_i, y_j) + u(x_{i+1}, y_j)] + O(h^2), \tag{4.2.2}$$

$$\frac{\partial^2 u}{\partial y^2}\Big|_{(x_i, y_j)} = \frac{1}{h^2}[u(x_i, y_{j-1}) - 2u(x_i, y_j) + u(x_i, y_{j+1})] + O(h^2). \tag{4.2.3}$$

代入（4.2.1）后，再用$u_{i,j}$代替$u(x_i, y_j)$，并略去误差项得：

$$4u_{i,j} - (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) = h^2 f_{ij},$$

$$i, j = 1, 2, \ldots, n-1, \tag{4.2.4}$$

$$u_{i,0} = \phi_{i,0}, \quad u_{i,n} = \phi_{i,n}, \qquad i = 0, 1, 2, \ldots, n,$$

$$u_{0,j} = \phi_{0,j}, \quad u_{n,j} = \phi_{n,j}, \qquad j = 0, 1, 2, \ldots, n。$$

现假定 $\phi \equiv 0$，将（4.2.4）写成矩阵形式即为：

$$T_{n-1}U + UT_{n-1} = h^2 F, \tag{4.2.5}$$

其中 $U = [u_{ij}]$， $F = [f_{ij}]$， $T_{n-1} = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \in \mathrm{R}^{(n-1)\times(n-1)}$。

如果我们再取 $f_{ij} \equiv -1$，就会发现（4.2.5）是一个具有特殊形式的 Sylvester 方程：

$AX - XB = C$。其中：A、B和C都是属于$\mathrm{R}^{(n-1)\times(n-1)}$的矩阵。

A 是三对角矩阵 $\begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}$，$B=-A$，$C=h^2F=\frac{1}{(n-1)^2}\begin{bmatrix} -1 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & -1 \end{bmatrix}$。

我们取初始点：$X=2I_{(n-1)\times(n-1)}$，用投影 BB 方法计算（4.2.5）的矩阵分离度，并用基于新优化转化的 BB1 方法计算 Sylvester 方程（4.2.5）。

2. 五点差分格式的条件数

条件数：$\kappa = (\|A\| + \|B\|) / \mathrm{Sep}(A, B)$；

约束条件误差范围：$\varepsilon=10^{-6}$。

	n=10	n=20	n=50	n=100
算法终止条件 ε =	10^{-4}	10^{-4}	10^{-2}	10^{-1}
条件数	292.6225	2157.1229	7541.20	52363.1

表 9：投影 BB 方法计算五点差分格式条件数

结论：五点差分格式的条件数随着位数的增加，越来越大，因此问题也是越来越病态。所以可以预见用五点差分格式计算椭圆型方程不是好方法。

3. 五点差分格式的数值解

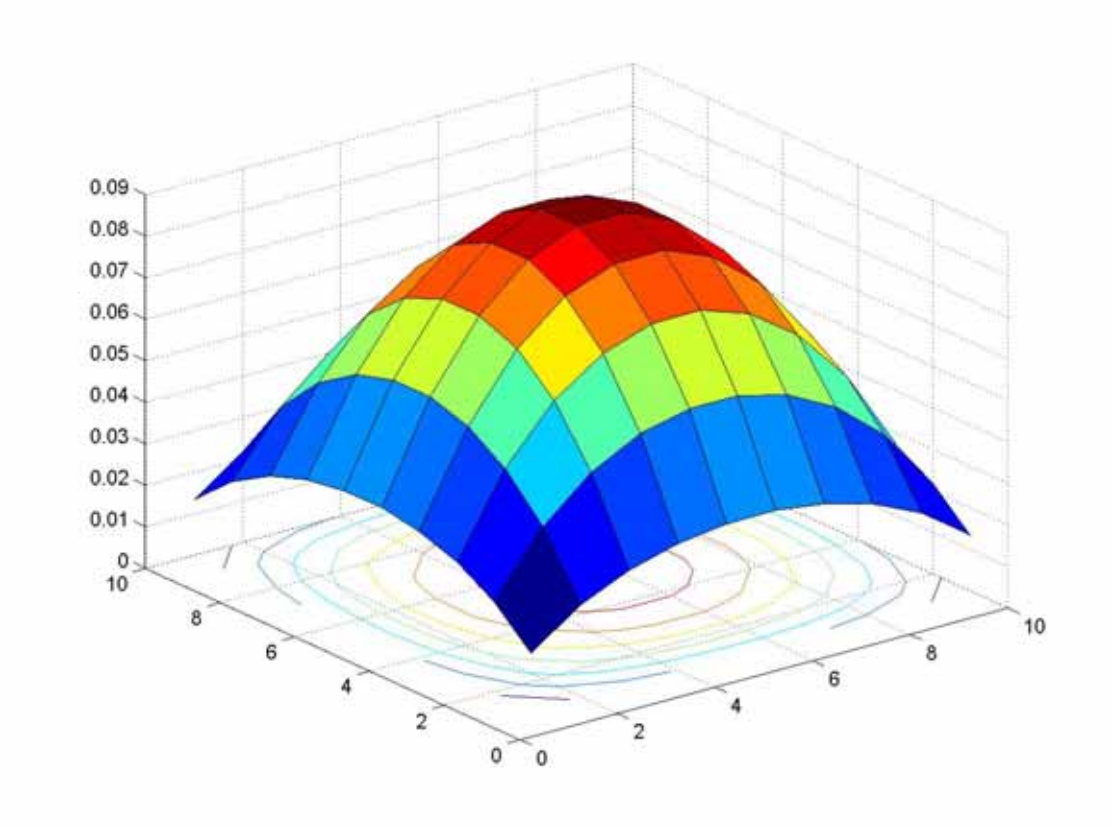
根据（4.2.2）和（4.2.3），我们知道五点差分格式的截断误差是 $O(h^2)$，也即 $O(\frac{1}{n^2})$。

(1) n=10，即网格为 10×10，此时差分格式的截断误差为 10^{-2}。

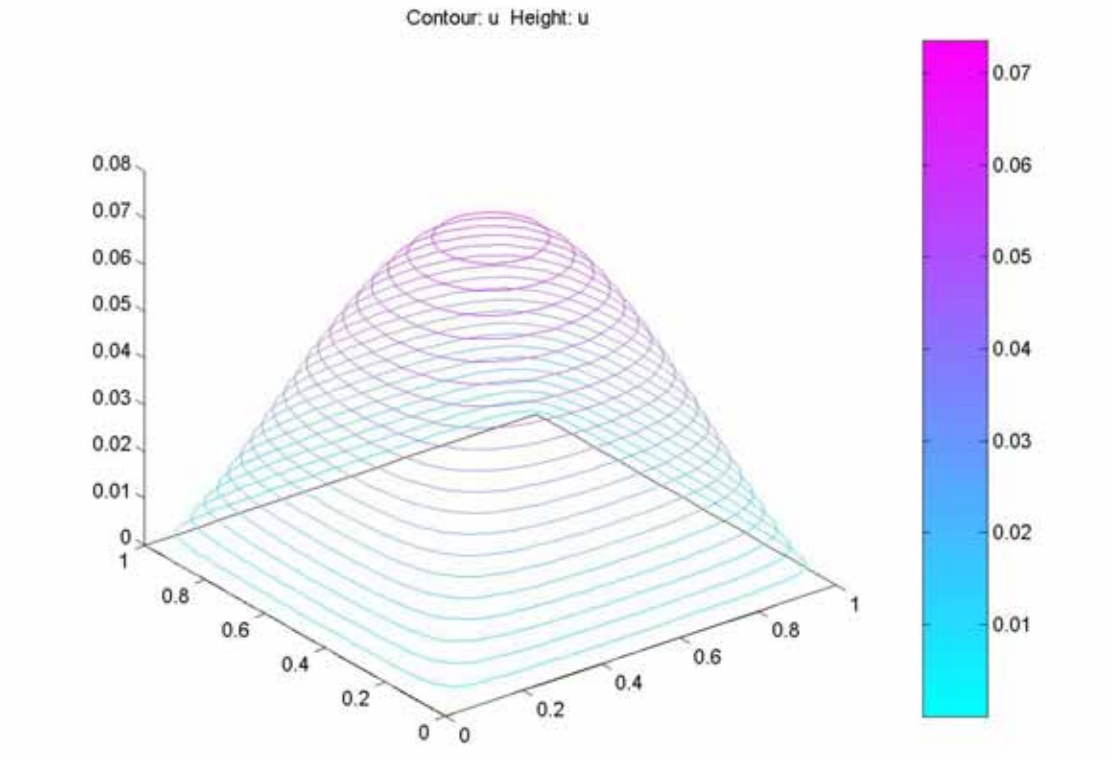
我们取算法的终止误差：$\varepsilon=10^{-6}$。

结果参见附图 2。

同时我们给出用取三角形网格的有限元方法计算椭圆方程的数值结果，参见附图 3。我可以把有限元方法的结果作为标准来检验差分方法。


0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
10
8
6
4
2
0
0
2
4
6
8
10

附图 2

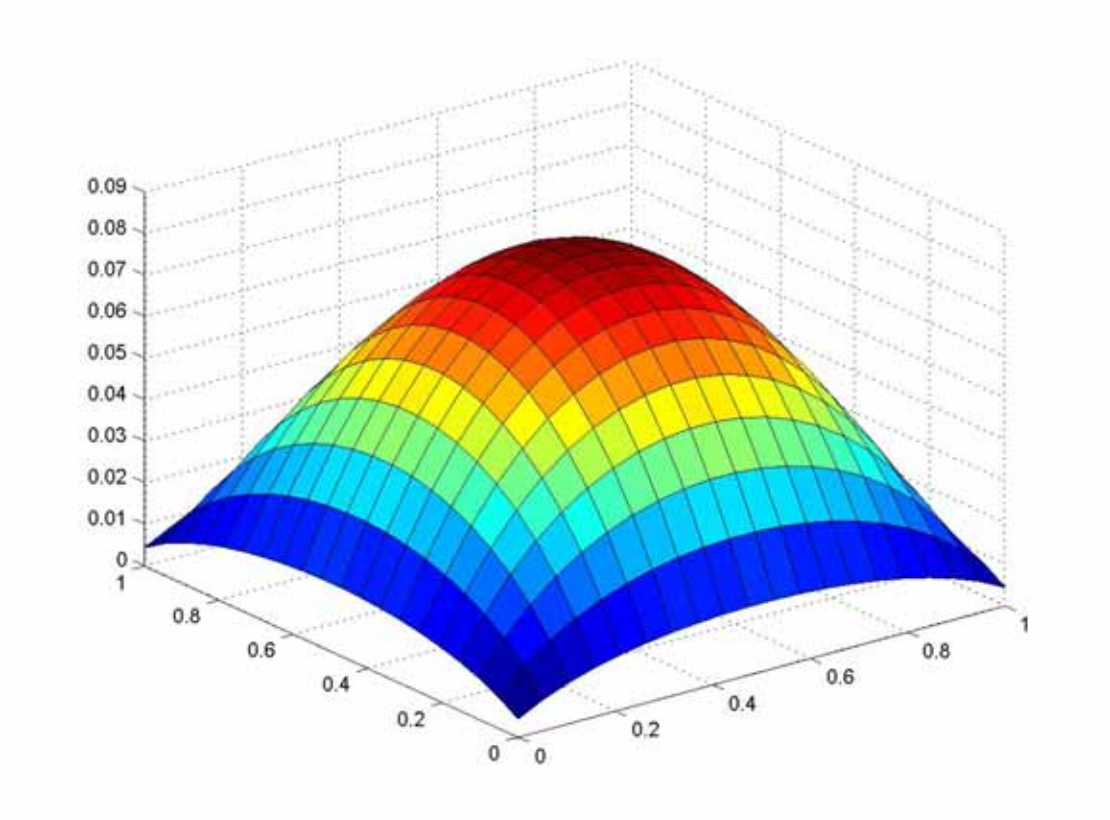

Contour: u Height: u
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
0.07
0.06
0.05
0.04
0.03
0.02
0.01

附图 3

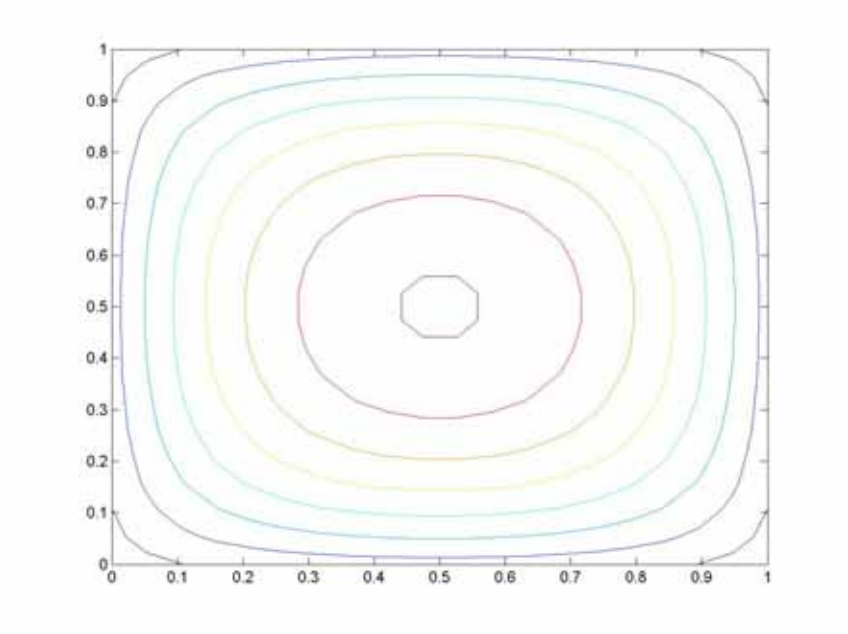
(2) n=20，即网格为 20×20，此时差分格式的截断误差为 10^{-3}。

我们取算法的终止误差：$\varepsilon = 10^{-6}$。

结果参见附图 4，等高线参见附图 5。



附图 4

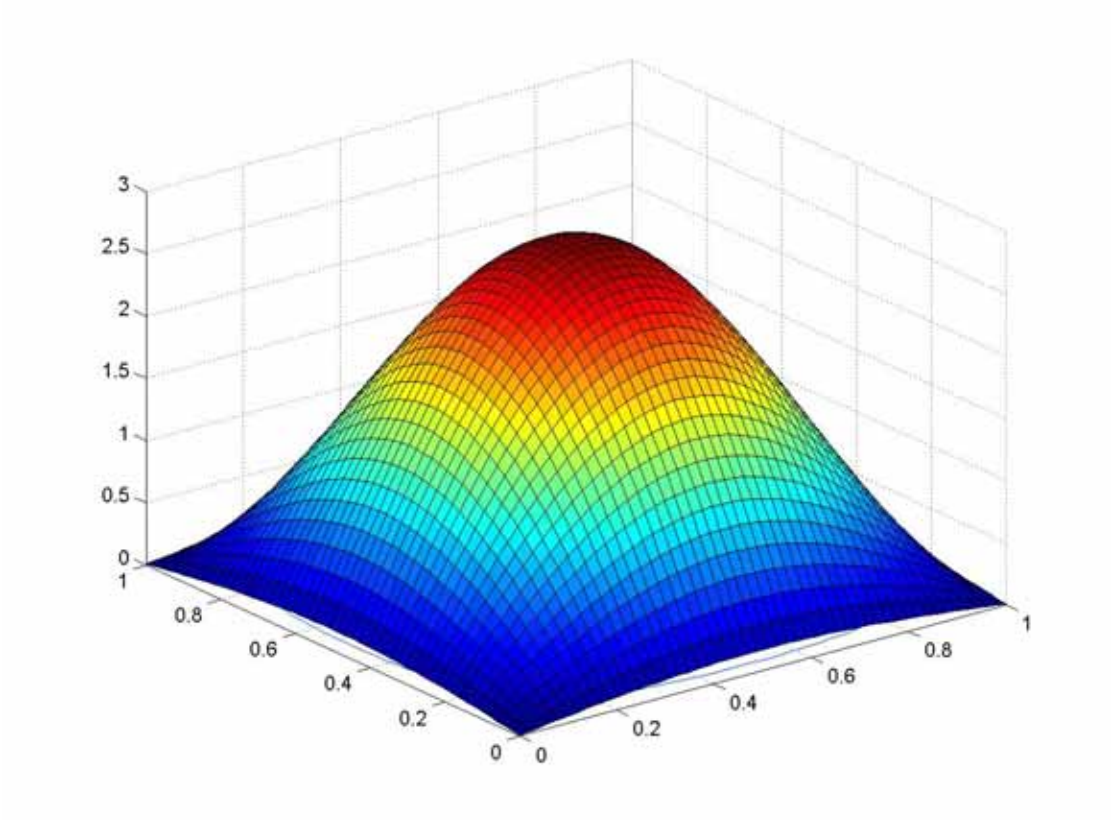


附图 5

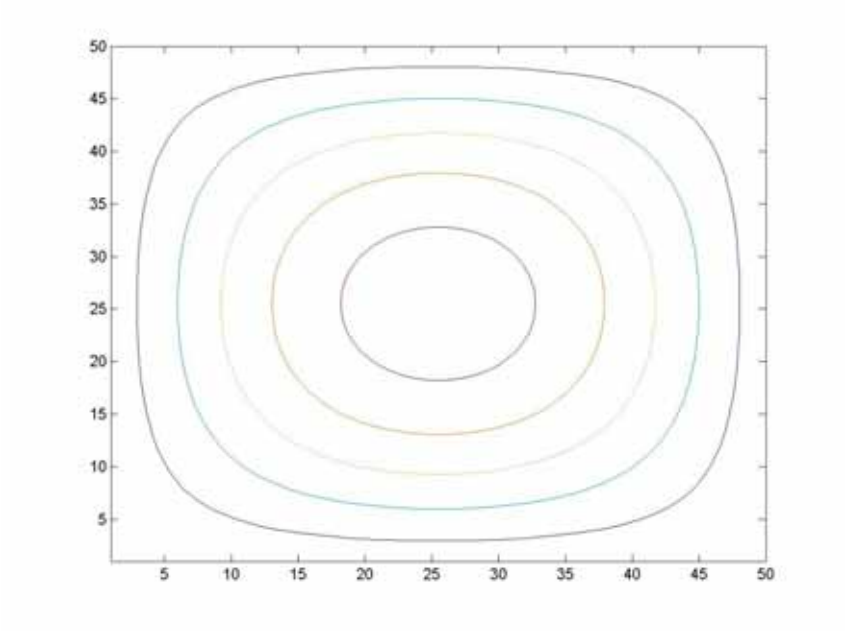
(3) n=50，即网格为50×50，此时差分格式的截断误差为10^{-4}。

我们取算法的终止误差：$\varepsilon = 10^{-2}$。

结果参见附图 6，等高线参见附图 7。



附图 6

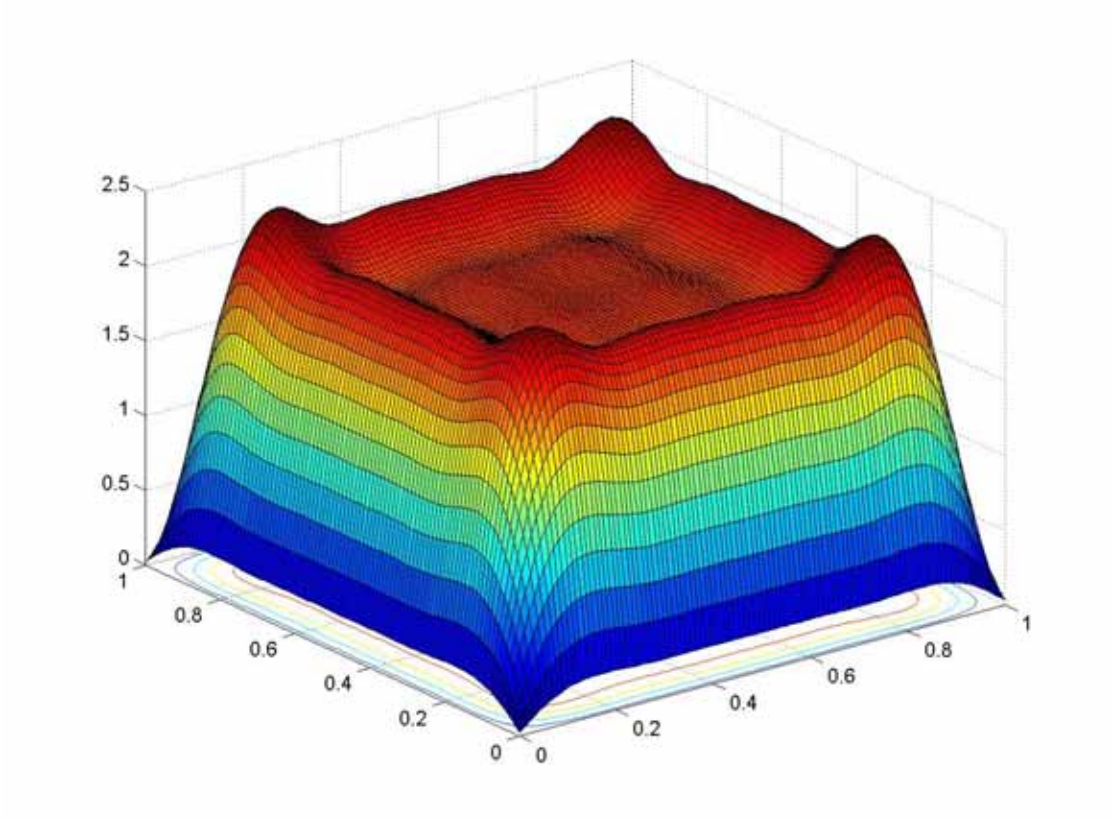


附图 7

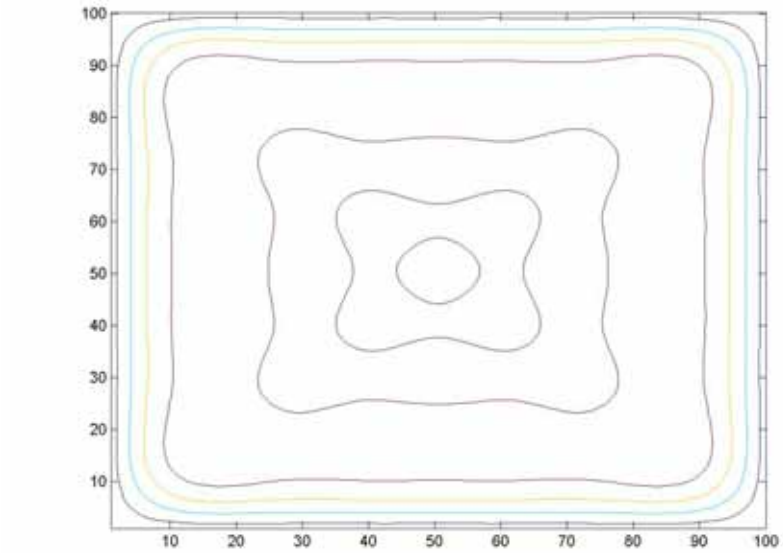
(4) n=100，即网格为100×100，此时差分格式的截断误差为10^{-4}。

我们取算法的终止误差：$\varepsilon=10^{-1}$。

结果参见附图 8，等高线参见附图 9。



附图 8



附图 9

结论：我们不难发现，由于条件数太差，五点差分格式是很病态的，导致解波动较大。对于 n=10、20，不难发现我们的解在边界处的结果不好；对于 n=50，由于作者机器性能不高，并且受不能作长时间科学计算的客观条件所限，因此只能取算法终止误差为 $\varepsilon = 10^{-2}$，比截断误差10^{-4}还大，因此得到的结果和标准（附图 3）相比，有一些差距；对于 n=100，和上面的原因一样，作者取算法终止误差为 $\varepsilon = 10^{-1}$，我们可以明显地从附图 8 中看出：迭代还没进行彻底，我们的算法已经结束了。这些结果和条件数给我们的信息完全相符，所以我们可以得到最后结论：用五点差分格式解椭圆型偏微分方程不好。

第五章 总结和展望

第一节 矩阵分离度问题

本文分析了罚函数法不适合处理矩阵分离度问题的瓶颈所在——大规模问题的目标函数的 Hesse 矩阵难以存储，于是提出用投影方法计算。我们从辩证法里知道：一样东西有它的优点，就肯定有它的缺点。而投影梯度方法的缺点在于它需要计算约束函数 Jacobi 矩阵逆的 QR 分解。而 QR 分解的运算量是很大的，同时还需要存储它的 Q、R 矩阵。

为解决这个缺点，本文分析了矩阵分离度问题的特殊性，修正了投影梯度方法，提出了单约束投影梯度方法，不用 QR 分解，也不用存分解矩阵。

实验结果表示，投影梯度方法明显快于罚函数法，而且还能处理罚函数法不能处理的大规模问题。

在此基础上，本文又试验了单约束投影 BB 方法，发现在处理大规模矩阵分离度问题中，它明显优于单约束投影梯度法。本文提出的单约束投影梯度法和单约束投影 BB 方法从理论上适用于一切单约束非线性优化问题。

下面从存储量和运算量的角度来分析各算法：

	存储量	运算量
Lagrange—Newton 法	$O((nm+1)^2)$	$O(n^2m^2)$
乘子罚函数法	$O((nm)^2)$	$O(n^2m^2)$
投影梯度法 投影 BB 方法	$O(\max\{nm, n^2, m^2\})$	$O(nm(n+m))$

表 9：矩阵分离度各优化算法的存储量、运算量比较

所以可以看出，本文算法的存储量和运算量，当 n 和 m 接近时，比较令人满意；但当 n 和 m 差得比较远时，它和传统方法比就没有什么优势了，也会出现存储和速度慢的问题。对于这类问题，在今后的工作中，我们还可以加以考虑。

此外我们也看出，对于 $n=m\geq 80$ 的问题，我们的算法也相当慢了，这也是我们值得继续探究的地方。

第二节 Sylvester 方程的求解

用优化方法求解代数方程，首先要找到合适的转换方法。本文分析了传统转化方法，并提出了新转化方法，并且从理论和实验的角度都说明了传统转化方法适合小规模 Sylvester

方程的求解，新转化方法适合大规模 Sylvester 方程的求解。

本文应用新转化方法，比较了在处理大规模无约束二次函数优化问题中，各梯度型方法：SD 方法、MG 方法、BB 方法、AS 方法和 AM 方法的优劣。同时也说明了我们的算法更适合于大规模具有特殊形式的 Sylvester 方程。

本文分析了矩阵分离度和 Sylvester 方程的关系，给出了一个定理，并证明了它的一个重要推论。该推论揭示了矩阵分离度和 Sylvester 方程的关系，好比行列式和一般线性方程组的关系。

最后给出将本文的主要算法应用在椭圆型方程求解上的实例。用投影 BB 算法计算了 Poisson 方程第一边值条件的五点差分格式的条件数，用基于新优化转化的 AS 方法求解了 Poisson 方程第一边值条件的五点差分格式。

本文介绍的优化转化方法求解 Sylvester 方程不足之处前文里已经有了介绍，本文提出的新优化转化其实是以牺牲运算量来节省存储空间的。于是我们今后可以继续探究的是，如何设计一种对大、小规模问题都合适的算法；如何设计一种对大规模的一般问题仍然有很高速度的算法。

参考文献

[1] Jonathan Barzilai、Jonathan M. Borwein，*Two-Point Step Size Gradient Methods*，IMA Journal of Numerical Analysis(1988)8,141-148。

[2] Y. H. Dai，*Alternate Stepsize gradient method*，Research report AMSS-2001-041, Academy of Mathematics and System Science, Chinese Academy of Science, 2001。

[3] Y. H. Dai、Y. X. Yuan，*Alternate Minimization Gradient Method*，International Workshop on "Numerical Linear Algebra, Numerical Method for PDE, and Mathematical Programming" (Curitiba, Brazili, August 20-23, 2001)。

[4] Zoran Gajic、Muhammad Tahir Javed Qureshi，*Lyapunov Matrix Equation in System Stability and Control*，（Mathematics in Science and Engineering Volume 195），Academic Press，Inc.，1995。

[5] 李荣华、冯果忱，*微分方程数值解法*，高等教育出版社，2002。

[6] 马世谦，*用优化方法计算矩阵分离度*，北京大学本科生毕业论文，2003。

[7] Marcos Raydan，*On the Barlizai and Borwein choice of steplength for the gradient method*，IMA Journal of numerical Analysis(1993)13,321-326。

[8] J. G. Sun，*Estimation of the seperaion of two matrices*，J. Comput. Math. 2:3(1984):189-200。

[9] 孙继广，*矩阵扰动分析*，科学出版社，1987。

[10] S. F. Xu，*Lower bound estimate for the separation of two matrices*，Lin. Alg. and Its Appl. 262,(1997):67-82。

[11] 徐树方、高立、张平文，*数值线性代数*，北京大学出版社，2000。

[12] 袁亚湘，*非线性规划数值方法*，上海科学技术出版社，1993。

[13] 袁亚湘、孙文瑜，*最优化理论与方法*，科学出版社，2001。

致 谢

谨在学位论文完成之际，把我最真诚的谢意献给指导我毕业设计的恩师高立教授，对高老师两年来给予我的关心、帮助和指导，致以最衷心的感激。两年来，无论是在课堂上还是在讨论班中，高老师给我的指导和帮助都是巨大的，她条理清晰的课堂讲解、严谨务实的治学态度和敏锐的洞察力给我留下了不可磨灭的印象，并将一直影响我今后的学习和工作；在课下，高老师既像一位和蔼的长者关心我们的学习、生活，又能像一个同龄的朋友和我们没有隔阂地交流，这也是我难以忘怀的。我想我能在最优化研究这条路上继续走下去，是和高老师指引和帮助分不开的，我在毕业设计研究中的每一点成绩都倾注着高老师的无私心血。

这里我还要特别感谢班主任李若老师、汤华中老师长期以来对我关心和指导。虽然很少听他们的课，但是在平时严谨务实的治学和工作态度，非常让我崇敬。他们的他们渊博的学识和高尚的品格，将永远都是我学习的榜样。

我要感谢应隆安老师、徐树方老师、李铁军老师以及学院其他教育指导帮助过我的老师们，感谢他们的谆谆教诲使我成长起来。感谢学院负责学生工作的田立青老师和刘雨龙老师四年来对我无微不至的关心和帮助。

我还要感谢王鑫、高勤、刘思等同学，平日里和他们的讨论使我受益匪浅，从他们那里我还得到了不少宝贵的建议和指点。

最后，我要把诚挚的谢意带给我的父母和朋友，是他们无私的爱和鼓励使我在自己选择的道路上不畏挫折，坚定地走下去。我会满怀热情地面对今后的学习、工作以回报他们的殷切希望。

作 者

2004 年 5 月