摘要

本文首先介绍前人求解形如$AX - XB = C$的 Sylvester 矩阵方程的一些成果，给出 Sylvester 方程解存在和唯一性定理，将 Arnoldi 算法推广到数域上的有限维内积空间中，并给出了部分数值求解的算法。本文利用 Kronecker 积将方程转化成$Sx = c$的形式进行理论分析，并通过对 Sylvester 方程隐式调用基于 Krylov 子空间求解$Sx = c$的 GMRES 方法，得到求解大规模 Sylvester 方程的全局投影方法。最后，编写对应的 MATLAB 程序，对该方法进行数值测试并与其它方法进行各方面比较，总结该方法的优缺点和适用范围。

关键词：Sylvester 矩阵方程，Krylov 子空间，GMRES，MATLAB

Abstract

This paper first introduce several methods of solving Sylvester matix equation in the form of $AX - XB = C$, then gives the existence and uniqueness theorems of Sylvester matix equation, extends Arnoldi method to the case of finite-dimensional inner product space and do some numerical tests of well-known algorithm. This paper translates Sylvester matix equation into form of $Sx = c$ using Kronecker product. We get global projection methods of solving large Sylvester matrix equation By implicit invocating GMRES method based on Krylov subspace to solve $Sx = c$. Finally, we write corresponding MATLAB code compares with other methods, summary the advantages and disadvantages of the method,and its scope of application.

Keywords：Sylvester matrix equation, Krylov subspace, GMRES, MATLAB

目录

1 绪论

1.1 研究背景和意义

Sylvester 矩阵方程（以下简称 Sylvester 方程）是由 Sylvester 等人首先提出并进行研究，在控制和通信理论，模型降解问题、广义特征值扰动理论、微分方程数值解、线性系统等应用数学相关领域中有重要应用[3,4]。在应用力学、流体力学等应用领域，经常遇到二阶系统的解耦问题，其中一种方法就是将其转化求齐次 Sylvester 方程。微分数值解方面也会出现 Sylvester 方程，如椭圆边值问题离散后可以写成 Sylvester 方程的形式。因此研究 Sylvester 方程具有重大理论和实践意义。

1.2 Sylvester 方程介绍

Sylvester 方程的定义：

$$AX - XB = C \tag{1-1}$$

其中$A \in R^{N\times N}$，$B \in R^{s\times s}$，$C \in R^{N\times s}$。

当$B = A^T$时，(1-1)称为 Lyapunov 方程。

如果我们取$D = -B$，那么(1-1)可以写成 $AX + XD = C$，因此在很多文献中 Sylvester 方程会以$AX + XB = C$的形式呈现。我们一般可以认为矩阵$N \geq s$，这是因为否则的话，我们可以对方程(1-1)两边进行转置得到$B^T X^T - X^T A^T = -C^T$转化成我们需要的情况。

Sylvester 方程的求解分为精确解和数值解。精确解可以提供数值解的理论基础，但是由于工程中遇到的 Sylvester 方程都是大规模的，因此数值解有其更重大的意义，计算机的高速发展给大规模 Sylvester 方程的数值求解提供了条件。

1.3 Sylvester 方程的文献综述

系数矩阵A，B很小时，使用 Kronecker 积以及向量运算符vec可以将 Sylvester 方程转化成线性矩阵方程。并且通过这种变换，很容易证明，当且仅当矩阵A和B没有相同的特征值时，Sylvester 矩阵方程有唯一解。但是由于这种变化会将方程的阶数平方级别的提升，使得这种方法只适合求解小规模 Sylvester 方程，详见[1,2,5]。

系数矩阵A，B中小型矩阵时，目前标准解法是 Bartels-Stewart 算法，该方法的核心是将系数矩阵A，B进行 Schur 分解，通过直接法求解简单形式的 Sylvester 方程得到最终解详见[1,2]。文章[5]中提出 Bartels-Stewart 算法的修改形式，Golub,Nash,Loan 在[1]中详细的给出了 Bartels-Stewart 算法修改形式，成为目前求解中小型 Sylvester 方程最优方法，我们将在后续章节对此详细讨论。

系数矩阵A，B是大规模矩阵时，通常这类情况，我们使用一些基于 Krylov 子空间的投影法。El Guennouni 等人在论文[6]中介绍了一种新的基于 block-Arnoldi 和非对称

block-Lanczos 算法的 Krylov 方法来近似求解，并给出了近似误差的范式界限，该方法会在本文中详细介绍。

当系数矩阵满足$B = A^T$ 时，方程(1-1)变成 Lyapunov 方程，若此时有$C = uu^T$，其中$u \in R^N$，Saad [7]提出一个 Galerkin 型 Krylov 子空间的方法来计算 Lyapunov 方程较小秩的解。Jaimoukha 和 Kasenally[20]将 Saad [7]的方法推广到$C = UU^T$其中$U \in R^{N\times p}$的情形，研究 Lyapunov 方程的论文也有很多，例如[8,17,18]，这里就不过多涉及了。

系数矩阵A，B是大规模矩阵，但是矩阵C秩很小时，有很多论文研究形式$AX + XB = CD^T = 0$形式的解，并给出了求解这种形式大规模 Sylvester 方程较好的解，详见[7,14,15]。

1.4 本文结构

第一章，说明 Sylvester 方程的研究背景和意义，介绍 Sylvester 方程的具体内容，最后，简介针对不同情况的 Sylvester 方程前人提出的各种解法。

第二章，给出方程解的存在和唯一性定理。并介绍 Kronecker 积，vec运算，上 Hessenberg 矩阵，Schur 分解的概念和性质等线性代数和矩阵论知识。

第三章，介绍求解 Sylvester 方程的直接法，如高斯消去法，Bartels-Stewart 算法以及 Hessenberg-Schur 算法等。

第四章，介绍 Krylov 子空间基本概念和性质，给出有限维内积空间的 Arnoldi 算法产生 Krylov 子空间一组标准正交基，介绍相应的 GMRES 算法和 MINRES 算法。

第五章，介绍求 Sylvester 方程的块方法。

第六章，具体给出求解 Sylvester 方程的全局投影方法。

第七章，针对前面提及的算法进行数值实现，对试验结果进行比较总结。

2 Sylvester 方程解的存在和唯一性定理

2.1 Sylvester 方程解的存在性

定理 2.1(Roth [8]) 方程(1-1)有解当且仅当

$$\begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \text{ 和 } \begin{pmatrix} A & C \\ 0 & B \end{pmatrix} \tag{2-1}$$

相似。

证明(Flanders-Wimmer[21])：若方程有解X，则

$$\begin{pmatrix} I & X \\ 0 & I \end{pmatrix}^{-1} \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \begin{pmatrix} I & X \\ 0 & I \end{pmatrix} = \begin{pmatrix} A & C \\ 0 & B \end{pmatrix}$$

反过来，设(2-1)中的两个矩阵相似，定义线性变换$f_i: M_{m+n} \to M_{m+n}, i = 1,2$

$$f_1(G) = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} G - G \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$$

$$f_2(G) = \begin{pmatrix} A & C \\ 0 & B \end{pmatrix} G - G \begin{pmatrix} A & C \\ 0 & B \end{pmatrix}$$

由于相似性假设，$dim\ ker\ f_1 = dim\ ker\ f_2$。直接计算可知

$$ker\ f_1 = \left\{ \begin{pmatrix} P & Q \\ R & S \end{pmatrix} \middle| \begin{matrix} AP = PA, AQ = QB \\ BR = RA, BS = SB \end{matrix} \right\},$$

$$ker\ f_2 = \left\{ \begin{pmatrix} P & Q \\ R & S \end{pmatrix} \middle| \begin{matrix} AP + CR = PA, AQ + CS = QB \\ BR = RA, \quad BS = SB \end{matrix} \right\}.$$

为了证明方程(1-1)有解，我们只要说明$Ker\ f_2$中有一个形如

$$\begin{pmatrix} P & Q \\ 0 & -I \end{pmatrix}$$

的矩阵，此时$AQ - C = QB$。记

$$V = \{(R,S) | BR = RA, BS = SB\}.$$

这是$M_{n,m+n}$的一个子空间。定义$g_i: ker\ f_i \to V, i = 1,2,$

$$g_i\left(\begin{pmatrix} P & Q \\ R & S \end{pmatrix} \right) = (R,S).$$

则

$$ker\ g_1 = ker\ g_2 = \left\{ \begin{pmatrix} P & Q \\ 0 & 0 \end{pmatrix} | AP = PA, AQ = QB \right\}.$$

我们将证明$Im\ g_1 = Im\ g_2$。显然$Im\ g_1 = V$，因为若$BR = RA, BS = SB$ 则

$$\begin{pmatrix} 0 & 0 \\ R & S \end{pmatrix} \in ker\ f_1, \quad g_1\left(\begin{pmatrix} P & Q \\ R & S \end{pmatrix} \right) = (R,S).$$

所以$Im\ g_2 \subseteq Im\ g_1$。但由维数定理知

$$dim\ ker\ g_i + dim\ Im\ g_i = dim\ ker\ f_i$$

因此$dim\ Im\ g_1 = dim\ Im\ g_2$。再由$Im\ g_2 \subseteq Im\ g_1$知$Im\ g_2 = Im\ g_1$。显然，

$$(0, -I) \in V = Im\ g_1 = Im\ g_2.$$

所以，存在

$$\begin{pmatrix} P & Q \\ R & S \end{pmatrix} \in ker\ f_2 \text{满足} g_i\left(\begin{pmatrix} P & Q \\ R & S \end{pmatrix}\right) = (R,S) = (0,-I),$$

即$\begin{pmatrix} P & Q \\ 0 & -I \end{pmatrix} \in ker\ f_2$，证毕。

2.2 Sylvester 方程解的唯一性

为了更清楚的说明解的唯一性定理，我们先做如下预备工作：

2.2.1 Kronecker 积的定义及其性质[11]

设 $A = (a_{ij}) \in M_{m,n}$，$B \in M_{s,t}$。A和B的 Kronecker 积记作$A \otimes B$，定义为下面的分块矩阵：

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix} \in M_{ms,nt}$$

Kronecker 积又称为张量积。Kronecker 积有很多良好且结论显然的性质。下面我们列出如下重要性质

(1) $(A \otimes B) \otimes C = A \otimes (B \otimes C), A \in M_{m,n}, B \in M_{s,t}, C \in M_{p,q}$

(2) $(A + B) \otimes C = A \otimes C + B \otimes C, A, B \in M_{m,n}, C \in M_{s,t}$

(3) $A \otimes (B + C) = A \otimes B + A \otimes C, A \in M_{m,n}, B, C \in M_{s,t}$

(4) $A \otimes B = 0$ 当且仅当$A = 0$ 或$B = 0$

(5) $(A \otimes B)(C \otimes D) = (\mathrm{AC}) \otimes (BD), A \in M_{m,n}, B \in M_{s,t}, C \in M_{n,k}, D \in M_{t,r}$

(6) 若A, B可逆，则$A \otimes B$也可逆，且$(A \otimes B)^{-1} = A^{-1}B^{-1}$

(7) 若$\lambda \in \sigma(A)$，x是对应的特征向量，$u \in \sigma(B)$，y是对应的特征向量，则

$$\lambda u \in \sigma(A \otimes B), x \otimes y \text{是对应的特征向量}$$

(8) 若$\sigma(A) = \{\lambda_1, \cdots, \lambda_n\}$，$\sigma(B) = \{u_1, \cdots, u_m\}$，则

$$\sigma(A \otimes B) = \{\lambda_i u_j | i = 1, \cdots, n, j = 1, \cdots, m\}$$

(9) $det(A \otimes B) = (det\ A)^m (det\ B)^n$

性质(5-9)中$A \in M_n, B \in M_m$。其中$\sigma(A)$表示A的全部特征值$\lambda_1, \cdots, \lambda_n$构成的集合$\sigma(A) = \{\lambda_1, \cdots, \lambda_n\}$。

上面所列举的有关 Kronecker 积的重要性质，这些结论的证明是十分显然的，这里就不给出了。

2.2.2 vec运算符及其和 Kronecker 积的关系

给定A，我们用$vecA$表示A的各列依次连接起来所得的向量：

设$A = (a_1, a_2, \cdots, a_n) \in M_{m,n}$，则

$$vecA = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

下面重要定理给出了 Kronecker 积和vec运算符的密切关系，是后面关于 Sylvester 方程解的唯一性讨论提供了理论基础。

定理 2.2 设$A \in M_{m,n}$，$B \in M_{n,k}$，$C \in M_{k,t}$，则

$$vec(ABC) = (C^T \otimes A)vecB$$

证明：对于矩阵G，用G_p表示G的第p列。设$C = (C_{ij})$，则

$$(ABC)_p = ABC_p = A\left(\sum_{i=1}^{k} c_{ip} B_i\right) = (c_{1p}A, c_{2p}A, \cdots, c_{kp}A)vecB = \left[(C_p)^T \otimes A\right] vecB$$

因此

$$vec(ABC) = \begin{pmatrix} (C_1)^T \otimes A \\ \vdots \\ (C_t)^T \otimes A \end{pmatrix} vecB = (C^T \otimes A)vecB$$

证毕。

2.2.3 Sylvester 方程的转化

Sylvester 方程$AX - XB = C$,其中$A \in R^{N\times N}$，$B \in R^{s\times s}$，$C \in R^{N\times s}$。对 Sylvester 方程两边做vec操作可得$vec(AXI_s - I_N XB) = vec(C)$。由定理 2.2 知：

$$(I_s \otimes A - B^T \otimes I_N)vec(X) = vec(C) \tag{2-2}$$

我们设$S = I_s \otimes A - B^T \otimes I_N, x = vec(X), c = vec(C)$，则

$$Sx = c \tag{2-3}$$

其中$S \in R^{Ns\times Ns}, x \in R^{Ns}, c \in R^{Ns}$。

2.2.4 Sylvester 方程解的唯一性定理

定理 2.3 方程(1-1)解存在且唯一当且仅当A和B没有公共特征值。

证明：设$\sigma(A) = \{\lambda_1, \cdots, \lambda_N\}$，$\sigma(B) = \{u_1, \cdots, u_s\}$，则由 2.2.1 节中的结论(8)知

$$\sigma(S) = \sigma(I_s \otimes A - B^T \otimes I_N) = \{\lambda_i - \mu_j | i = 1, \cdots, N, j = 1, \cdots, s\}$$

因此，$\sigma(S)$没有 0 特征值当且仅当A和B没有公共特征值，又(1-1) 解存在且唯一等价于(2-2) 解存在且唯一等价于(2-3) 解存在且唯一等价于$\sigma(S)$没有 0 特征值。因此方程(1-1)解存在且唯一当且仅当A和B没有公共特征值，证毕。

3 直接法求解 Sylvester 方程

由定理 2.3 知 Sylvester 方程(1-1)解存在且唯一当且仅当A和B没有公共特征值。在本文的后续内容中除非特殊声明，都默认该条件成立，即本文后续只讨论解唯一的情况，实际上工程中矩阵A和B一般是没有公共特征值的。另外，本论文中算法讲解都与 MATLAB 相结合，伪代码风格也是类 MATLAB 的。

3.1 高斯消元法求解小规模 Sylvester 方程

根据 2.2.3 节知求解(1-1)等价于求解(2-2)和(2-3)。而(2-3)的求解有很多现成的方法，例如高斯消元法，Jacobi 迭代法或 Gauss-Seidel 迭代法，小规模线性方程最常见的是用高斯消元法的直接方法求解。下面给出详细的求解步骤：

算法 3.1(记作 M1)

计算 $S = I_s \otimes A - B^T \otimes I_N$

计算 $c = vec(C)$

利用高斯消元法求解 $Sx = c$

重组解向量x 得到解矩阵X

算法 M1 结束

由于 M1 中计算所得的$S \in R^{Ns \times Ns}$，因此使用高斯消去法求解$Sx = c$的时间复杂度是$O(Ns)^3$，空间复杂度$O(Ns)^2$。如果$N = 100, s = 10$，算法 M1 的运算量也高达 1e9，但由于计算的S是一个稀疏矩阵 MATLAB 对这种矩阵做了内部优化导致计算量相对较低，能在 0.06 秒内完成操作，运算内存消耗大约为 8M。如果$N = 300, s = 20$，算法 M1 的运算时间为 4.5 秒，内存约为 288M 一次计算勉强可以接受，但是如果$N = 1000, s = 100$算法 M1 的运算量也高达 10^15.大约需要一天的时间才能完成运算，并且内存 40G，但是普通电脑的内存一般为 2-16G，是无法计算满足需要的，不过当前 MATLAB 可以通过虚拟内存完成该操作，不过是以时间作为代价的，是不可取的。另外，高斯消元法解的精度依赖于S的条件数，而实践表明S的条件数随着阶数增大而急剧增大。无论是时间复杂度还是空间复杂度亦或是精度三方面综合考虑算法 M1 只适合求解小规模 Sylvester 方程，就目前普通计算机而言，$N \times s < 1000$时，可以考虑使用简单易懂的算法 M1。

3.2 Bartels-Stewart 和 Hessenberg-Schur 算法求解中小规模 Sylvester 方程

由于 Bartels-Stewart 算法[2]和 Hessenberg-Schur 算法[1]有很多共性，因此这里我们先介绍它们的共性，在具体的讲解的细节，这两个算法都是理论和实践相对较好的算法，并且是解决中小型规模 Sylvester 方程的标准方法。

Bartels-Stewart 和 Hessenberg-Schur 都试图化简原方程(1-1)中的系数矩阵成简单形式，然后再对简单形式进行求解。具体步骤如下：

(1) 应用相似变换将矩阵A,B转化成简单形式$A_1 = U^{-1}AU$和$B_1 = V^{-1}BV$。

(2) 求解方程$UF = CV$得到矩阵F。

(3) 求解转换的简单方程$A_1Y - YB_1 = F$。

(4) 求解$XV = UY$得到解矩阵X。

采用标准 Wilkinson 逆元误差分析[11]，我们知道上述做法的相对误差是$u[\kappa_2(U) + \kappa_2(V)]$，其中$u$是计算机的机器精度，$\kappa_2(W)$是矩阵$W$二范式对应的条件数。定义如下：

$$\kappa_2(W) = \|W\|_2\|W^{-1}\|_2 = \max_{x\neq 0}\sqrt{\frac{(Wx)^T(Wx)}{x^Tx}} \times \max_{y\neq 0}\sqrt{\frac{y^Ty}{(Wy)^T(Wy)}}$$

当$\kappa_2(W)$相对u较大时，我们说W是病态的。

然而不幸的是，在第一步产生的U和V很有可能是病态的。即使在矩阵A,B阶数很小时，这种情况仍有可能发生。这导致无论是 Bartels-Stewart 算法还是 Hessenberg-Schur 算法都在步骤 1 中采用正交阵U和V将矩阵A,B转化成简单形式（正交阵满足$U^TU = I$,因此易知$\kappa_2(U) = 1$）。Bartels-Stewart 和 Hessenberg-Schur 算法的唯一区别在于步骤(1)中 Bartels-Stewart 算法将矩阵A进行 Shcur 分解，而 Hessenberg-Schur 算法矩阵A转化成上 Hessenberg 矩阵，这也就导致在步骤(3)的求解也有一些小小的差异。我们下面详细介绍这两种算法。

3.2.1 Bartels-Stewart 算法求解 Sylvester 方程

Bartels-Stewart 算法的核心在于使用 QR 方法计算两个实 Schur 分解完成步骤 1：

$$U^TAU = R \qquad V^TB^TV = S$$

其中U和V是正交阵，R和S 是拟上三角的(拟上三角阵是说，至多出现对角块为2×2的块上三角矩阵)。

我们将方程(1-1)转化成

$$RY - YS^T = F \quad (F = U^TCV, Y = U^TXV)$$

如果$s_{k,k-1} = 0$，那么我们有：

$$(R - s_{kk}I)y_k = f_k + \sum_{j=k+1}^{n} s_{kj}y_j$$

这里$Y = [y_1|\cdots|y_s]$和$F = [f_1|\cdots|f_s]$。即y_k可以由$y_{k+1},\cdots,y_s$通过解拟上三角线性方程组得到，只需简单的$\frac{N^2}{2}$运算即可求出。如果$s_{k,k-1} \neq 0$，那么我们使用类似上面的方法将y_{k-1},y_k放在一起同时求解，这是的计算量是$2N^2$。

我们设进行一次 QR 分解需要的迭代步数为k，那么整个 Bartels-Stewart 算法的计算量大约为$W_{BS}(N,s) = kN^3 + ks^3 + \frac{5}{2}(N^2s + s^2N)$。一般我们可以认为这里$k = 10$。

下面给出 Bartels-Stewart 算法的伪代码：

算法 3.2.1(记作 M2)

(1) 用 Schur 分解$U^TAU = R$ 和 $V^TB^TV = S$。

(2) 计算 $F = U^T CV$ 。

(3) 求解$RY - YS^T = F$转化成用高斯消去法求解 s 个线性方程组得到Y。

(4) 计算 $X = UYV^T$ 得到 Sylvester 方程的解。

算法 M2 结束。

3.2.2 Hessenberg-Schur 算法求解 Sylvester 方程

下面我们开始详细介绍 Hessenberg-Schur 算法，我们对原方程中的矩阵A,B做正交变换得到。

$$H = U^T AU,\ S = V^T B^T V$$

这里H是上 Hessenberg 矩阵，S是拟上三角阵。U,V是正交阵。

一个矩阵H被称为上 Hessenberg 矩阵当且仅当对于所有$i > j + 1$成立$h_{i,j} = 0$。对矩阵A应用$N - 1$次 Householder 变换，大约需要$\frac{5}{3}N^3$步运算就能完成步骤$H = U^T AU$，详见[12,p.347]。利用 QR 方法对B^T做实 Schur 分解完成步骤$S = V^T B^T V$，运算量约为 kN^3。

我们将方程(1-1)转化成

$$HY - YS^T = F\ \ (F = U^T CV, Y = U^T XV)$$

类似于 Bartels-Stewart 算法，我们对$s_{k,k-1}$分情况讨论

如果$s_{k,k-1} = 0$，那么我们有：

$$(H - s_{kk} I)y_k = f_k + \sum_{j=k+1}^{n} s_{kj} y_j$$

这里$Y = [y_1|\cdots|y_s]$和$F = [f_1|\cdots|f_s]$。即y_k可以由$y_{k+1},\cdots,y_s$通过解拟上三角线性方程组得到，只需简单的N^2运算即可求出。

如果$s_{k,k-1} \neq 0$，那么考虑第$k-1$和第k列我们得到

$$H[y_{k-1}|y_k] + [y_{k-1}|y_k]\begin{bmatrix} s_{k-1,k-1} & s_{k,k-1} \\ s_{k-1,k} & s_{k,k} \end{bmatrix} = [f_{k-1}|f_k] - \sum_{j=k+1}^{n} \left[s_{k-1,j} y_j \middle| s_{kj} y_j\right] \equiv [g|w]$$

上式是一个关于y_{k-1}, y_k的$2N \times 2N$线性系统，通过调整可以用高斯消去法在$6n^2$步计算出y_{k-1}, y_k。整个步骤(4)的计算量大约为$3N^2 s + \frac{1}{2}sN^2$详见[2] 。

最后我们计算$X = UYV^T$得到 Sylvester 方程的解。

这里需要补充说明的是， QR 方法是一种迭代方法，虽然没有理论证明 QR 方法一定收敛，但是实际情况是，QR 方法总是收敛的，设 Schur 分解的迭代次数为 k，可以认为$k = 10$， 即$10s^3$步运算就能完成矩阵B的 Schur 分解，详见[13,p1-27]。由于U是正交阵，所以$F = U^T CV$，直接使用矩阵乘法计算$N^2 s + s^2 N$步运算可以完成。计算$X = UYV^T$得到 Sylvester 方程的解，这一步 计算量大约为$N^2 s + s^2 N$。因此整个 Hessenberg-Schur 算法求解的复杂度大约为$W_{HS}(N,s) = \frac{5}{3}N^3 + 10s^3 + 5N^2 s + \frac{5}{2}s^2 N$。就时间复杂度而言，Hessenberg-Schur 算法相对于 Bartels-Stewart 算法更优，特别当

$N \gg s$时更为明显，不过 Bartels-Stewart 算法更容易理解和掌握，总之它们都是当前求解 Sylvester 方程很优秀的算法。但是这类方法有一个不可避免的缺点就是解不是同步求得的，当前方程的解依赖之前的解，会导致误差会累积。

下面给出 Hessenberg-Schur 算法的伪代码

算法 2.3.3(记作 M3)

(1) 用 Householder 变换计算 $H = U^T AU$ 得到正交阵 U 和上 Hessenberg 矩阵H 。

(2) 用 Schur 分解计算$S = V^T B^T V$ 得到正交阵 V 和拟上三角阵 S。

(3) 计算 $F = U^T CV$。

(4) 求解$HY - YS^T = F$转化成用高斯消去法求解 s 个线性方程组得到Y。

(5) 计算 $X = UYV^T$ 得到 Sylvester 方程的解

算法 M3 结束。

3.2.3 Sylvester 方程的稳定性分析

我们在这一小节使用 2.2.3 节的记号。我们下面分析 Sylvester 方程的稳定性。为了对我们误差进行准确的分析，我们需要计算整个算法中每一步操作在有限的精度所带来的误差。

基于线性系统的敏感性知识我们知道，如果矩阵$S = I_s \otimes A - B^T \otimes I_N$是病态的，那么，矩阵$A, B$和$C$微小的改变将对解造成巨大的影响，为了判断矩阵$S$是否是病态的，我们需要定义线性变换空间$R^{N\times s}$到$R^{N\times s}$的范式。

$$f : R^{N\times s} \to R^{N\times s} \quad \|f\| = \max_{x\in R^{N\times s}} \frac{\|f(X)\|_F}{\|X\|_F}$$

$$\phi : R^{N\times s} \to R^{N\times s} \quad \phi(X) = AX - XB$$

这里 Frobenius 范式$\|\cdot\|_F$被定义成$\|W\|_F = \sum_{i,j}|w_{ij}|^2$。$\|\phi\| = \|S\|_2 \le \|A\|_2 + \|B\|_2$由于我们一直假定$S$是非奇异的。因此

$$\|\phi^{-1}\| = \left[\min_{x\in R^{N\times s}} \frac{\|\phi(X)\|_F}{\|X\|_F}\right]^{-1} = \|S^{-1}\|_2$$

现在我们考虑在计算机上解(1-1)会出现的机器精度u。一般保存矩阵A, B, C四舍五入误差上界分别为$u\|A\|_F, u\|B\|_F, u\|C\|_F$。即我们求得的最好解$\hat{X}$是满足

$$(A + E)\hat{X} - \hat{X}(B + F) = (C + G)$$

其中$\|E\|_F \le u\|A\|_F$，$\|F\|_F \le u\|B\|_F$，$\|G\|_F \le u\|C\|_F$。

定理 3.1 如果$AX - XB = \mathrm{C}$，$(A + E)\hat{X} - \hat{X}(B + F) = (C + G)$且$u[\|A\|_F + \|B\|_F]\|\phi^{-1}\| \le \frac{1}{2}$，那么我们有

$$\frac{\|X - \hat{X}\|_F}{\|X\|_F} \le 4u[\|A\|_F + \|B\|_F]\|\phi^{-1}\|$$

证明详见[11]。

由定理 3.1 知即使矩阵$A,$B是病态的，只要$\|\phi^{-1}\|$相对较小，也能使得解的相对精度在可接受范围内。

3.2.4 Hessenberg-Schur 算法误差分析

很多时候，我们并不过于看重解的相对误差，在很多实际应用中我们更加关注残差$\left\|\mathrm{A}\hat{X}-\hat{X}B-C\right\|_F$而非相对误差。我们现在着重考虑 Hessenberg-Schur 算法所产生的误差。我们用 hat 标识来代表计算中的实际值。我们有

$$\hat{U}=U_1+E_u \qquad U_1^TU_1=I,\|E_u\|_F\le\varepsilon \tag{3-1}$$

$$\hat{V}=V_1+E_v \qquad V_1^TV_1=I,\|E_v\|_F\le\varepsilon \tag{3-2}$$

$$\hat{H}=U_1^T(A+E_1)\,U_1 \qquad \|E_1\|_F\le\varepsilon \tag{3-3}$$

$$\hat{S}=V_1^T(B+E_2)^T\,V_1 \qquad \|E_2\|_F\le\varepsilon \tag{3-4}$$

$$\hat{F}=U_1^T(C+E_3)\,V_1 \qquad \|E_3\|_F\le\varepsilon \tag{3-5}$$

这里，ε是机器精度u很小的倍数，我们使用 2-范式，一个直接的误差分析可知，如果

$$\|\phi^{-1}\|\varepsilon(2+\varepsilon)[\|A\|_2+\|B\|_2]\le\frac{1}{2} \tag{3-6}$$

那么

$$\frac{\|X-\hat{X}\|_F}{\|X\|_F}\le(9\varepsilon+\varepsilon^2)\|\phi^{-1}\|[\|A\|_F+\|B\|_F] \tag{3-7}$$

不等式(3-7)表明计算$\hat{X}$的误差是 O($\|\phi^{-1}\|$)的，由于ε是机器精度u很小的倍数，根据 3.2.3 节的分析，我们知道 Hessenberg-Schur 算法和任何求解 Sylvester 方程的算法有着相同的舍入精度。通过类似的分析，如果条件(3-1)-(3-6)成立，那么我们有：

$$\left\|\mathrm{A}\hat{X}-\hat{X}B-C\right\|_F\le(10\varepsilon+3\varepsilon^2)(\|A\|_F+\|B\|_F)\|X\|_F$$

注意到$\left\|\mathrm{A}\hat{X}-\hat{X}B-C\right\|_F$的界与$\|\phi^{-1}\|$无关。

3.2.5 Bartels-Stewart 算法和 Hessenberg-Schur 算法的适用性分析

我们再次讨论算法 M2 和 M3 的适用范围，目前 Bartels-Stewart 和 Hessenberg-Schur 算法是目前求解中小型规模 Sylvester 方程的最好方法。就算法本身而言，它们的每一步计算都是有其理论依据，并且每一步计算都是稳定的，虽然求解矩阵 Schur 分解的 QR 方法没有理论保证收敛，但是实践说明 QR 方法是一个稳定高效的求解 Schur 分解的方法。这保证了算法 M2 和 M3 的稳定性。另外由于做相似变换的矩阵全是正交矩阵，因此保证了算法 M2 和 M3 的准确性，即保证了算法的精度。由上面的分析知，整个 Bartels-Stewart 方法求解的运算量大约为$W_{BS}(N,s)=10N^3+10s^3+\frac{5}{2}(N^2s+s^2N)$。整个 Hessenberg-Schur 方法求解的运算量大约为$W_{HS}(N,s)=\frac{5}{3}N^3+10s^3+5N^2s+\frac{5}{2}s^2N$，空间复杂度大约为$3N^2+9s^2+4Ns$。这是一个相对快的算法。当$N=1000,s=100$，整个计算量也不会超过 2*10^9。内存消耗大约 12M 左右。从各个方面看，算法 M2 和 M3 都优于算法 M1。当$N=3000,s=100$时，整个计算量也不会超过 2*10^12(大约需要 1 分钟)。内存消耗大约1.2G。是一般电脑可以接受的。虽然这时算法 M2 和 M3 是可接受

的，但是这时应用算法 M2 和 M3 就不太合适了，我们需要需求其他的方法。当然上面的时间分析是初略的，因为一个算法的运行时间受到很多因素影响，例如，计算机 CPU，高速缓存，是否支持多核，运行软件版本，以及测试数据等等。

3.2.6 GMRES 算法求 Sylvester 方程

这里所谓的大规模稀疏 Sylvester 方程式指矩阵A是大规模稀疏矩阵，利用 Krylov 子空间方法近似求解上述s个线性方程组，这里需要声明的是下面方法只适合矩阵B的阶数要远小于矩阵A的情况。[7,14]

类似于 Bartels-Stewart 算法和 Hessenberg-Schur 算法，该方法也将方程$AX - XB = C$转化成易于求解的形式。具体做法是步骤

(1) 考虑矩阵B的实 Schur 分解$B = UR_BU^T$，其中U是正交阵，R_B是块上三角的（块的尺寸是1×1或者2×2）那么方程(1-1)被转换成$AY - YR_B = D$，其中$Y = UX$，且$D = CU$。

假设矩阵 B 的所有特征值时实的，那么R_B就是一个上三角矩阵，求解$AY - YR_B = D$等价于求解s个线性系统

$$(A - r_{11}I)y_1 = d_1$$

和

$$(A - r_{kk}I)y_k = d_k + \sum_{i=1}^{k-1} r_{ik}y_i,\ k = 2, \cdots, s \tag{3-8}$$

其中$R_B = [r_{ij}]; Y = [y_1, \cdots, y_k]$且$D = [d_1, \cdots, d_k]$。

由于上述线性系统无法同时求得，当前方程依赖于前面方程的解，而我们在这里求解(5-1)是基于 Krylov 子空间的 GMRES 方法，该方法是一个近似解法，因此求解的误差会累积，这也导致我们的s（矩阵B的阶数）需要远远小于N（矩阵A的阶数）才能保证解在可接受范围内。

算法 2.3.4（记作 M4）

利用 Schur 分解计算 $B = UR_BU^T$ 得到正交阵U 和拟上三角阵R_B

计算 $D = CU$

用 GMRES 方法求解 $AY - YR_B = D$

计算 $X = U^TY$ 得到 Sylvester 方程的解

算法 M4 结束

另外如果我们事先知道矩阵A是对称矩阵，那么我们可以将 GMRES 方法换成 MINRES 从而大大优化算法 M4 时间复杂度和空间复杂度以及精度。

由于用 GMRES 方法计算 s 个线性方程的时间复杂度为kN^2s其中 k 是平均迭代次数，一般在 20 次左右当$\mathrm{N} = 10000, \mathrm{s} = 10$。运算量大约为 2.5*10^11(大约在 1 分钟能完成运算)。但是由于 GMRES 方法并不保证误差在可控制范围内，并且由于误差的累积，使得算法 M4 真实效果很差，这里就不再过多考虑算法 M4 了。

4 有限维内积空间中求解$\mathcal{A}x = b$

利用 Krylov 子空间方法求解 Sylvester 方程有两种常见形式，一种称为全局的，一种称为块的，但实际上，这两种方法本质上是一样的，只是存储形式和计算形式上有所区别，因此，本文在有限维内积空间中考虑 Krylov 子空间方法，将我们通常对向量或者矩阵[11,12,13,18]使用的 Krylov 子空间方法推广到有限维内积空间上。

4.1 有限维内积空间的一些记号

设数域 F 有限维内积空间V的维数为m。为了防止符号混淆，我们设V上的内积为$dot(\cdot,\cdot)$。设由内积诱导出的范式为$\|\cdot\|$。$\mathcal{A}$是V上的可逆线性变换，$b \in V$，求$x \in V$使得$\mathcal{A}x = b$。因为我们假设$\mathcal{A}$是V上的可逆线性变换，所以解$x = \mathcal{A}^{-1}b$存在且唯一。我们任取V中一组基$[V_1, \dots, V_m]$，设$\mathcal{A}$在这组基下的矩阵下A。

设$X = [x_1, \dots, x_k] \in V^k$,定义$\mathcal{A}X = [\mathcal{A}x_1, \dots, \mathcal{A}x_k] \in V^k$。对任意数域 F 上的矩阵$k \times n$矩阵$T$

$$XT = [x_1, \dots, x_k]T = [y_1, \dots, y_n] \in V^n$$

其中$T = (t_{ij})_{k\times n}$

$$y_i = \sum_{j=1}^{k} t_{ij}x_j$$

4.2 Krylov 子空间

给定有限维内积空间V中的线性变换$\mathcal{A}$和元素b可以得到以下序列

$$b_0 = b, b_1 = \mathcal{A}b_0, \cdots, b_{k+1} = \mathcal{A}b_k = \mathcal{A}^k b, \cdots$$

我们将

$$\varkappa_k(\mathcal{A}, b) = span\{b, \mathcal{A}b, \cdots, \mathcal{A}^{k-1}b\}$$

称为线性变换$\mathcal{A}$和元素b生成的 Krylov 子空间。显然我们有$\varkappa_k(\mathcal{A}, b) \sqsubseteq \varkappa_{k+1}(\mathcal{A}, b)$，我们称 Krylov 子空间在第$l$项终止，是指$l$为满足$\varkappa_l(\mathcal{A}, b) = \varkappa_{l+1}(\mathcal{A}, b)$的最小正整数，因为$\varkappa_k(\mathcal{A}, b) \sqsubseteq V$是有限维内积空间，因此上述$l$必然存在。

我们通常需要一组标准正交基来描述 Krylov 子空间。得到一组正交基最常见的方法是 Arnoldi 算法，当$\mathcal{A}$是对称线性变换时，相应的 Arnoldi 算法又称 Lanczos 算法。下面我们给出 Arnoldi 算法。

给定线性变换$\mathcal{A}$和元素b和一个正整数k，下面算法计算一个长度为k的 Arnoldi 分解

$$\mathcal{A}Q_{k+1} = Q_k\widehat{H}_k$$

其中$\widehat{H}_k$是数域 F 上的$(k+1) \times k$阶不可约上 Hessenberg 矩阵。设H_k为$\widehat{H}_k$的前k列。

Arnoldi 算法

取$q_1 = \|b\|^{-1}b$
for $j = 1{:}\,k$
　　$w = \mathcal{A}q_j$
　　for $i = 1{:}\,j$
　　　　$h_{ij} = dot(w, q_i)$
　　　　$w = w - h_{ij}q_i$
　　end
　　$h_{j+1,j} = \|w\|$
　　if $h_{j+1,j} = 0$
　　　　stop
　　else
　　$q_j = h_{j+1,j}^{-1}w$
　　end
end
Arnoldi 算法结束

Arnoldi 算法实际上这是求$\mathcal{A}q_k$在$\Re(Q_k)^{\perp}$上正交投影的 Gram-Schmidt 正交化过程，从 Gram-Schmidt 正交化过程研究可知，为了保证数据稳定性，应该使用修正的 Gram-Schmidt 正交化过程，另外，根据 Gram-Schmidt 正交化过程的研究经验，使用重正交过程可以使得正交性更加稳定。在代码实现时可以加入重正交过程。

由 Arnoldi 算法的过程我们知道

$$h_{ij} = dot\left(\mathcal{A}q_j, q_i\right)$$

因此当$i > j + 1$时，$\mathcal{A}q_j \in \varkappa_{j+1}(\mathcal{A}, b) \sqsubseteq \varkappa_{i-1}(\mathcal{A}, b)$又$q_i \perp \varkappa_{i-1}(\mathcal{A}, b)$，因此$q_i \perp \mathcal{A}q_j$即$h_{ij} = 0$对任意$i > j + 1$成立。

当$\mathcal{A}$是对称线性变换时，$h_{ij} = dot\left(\mathcal{A}q_j, q_i\right) = dot\left(\mathcal{A}q_i, q_j\right) = h_{ji}$。即此时$H_k$是对称三角阵。此时我们可以得到类似于 Arnoldi 分解的一个长度为k的 Lanczos 分解。

Lanczos 算法
$\beta_0 = 0, q_0 = 0$
for $j = 1{:}\,k$
　　$w = \mathcal{A}q_j; \alpha_j = dot(w, q_j)$
　　$w = w - \alpha_j q_j - \beta_{j-1}q_{j-1}$
　　$\beta_j = \|w\|_2$
　　if $\beta_j = 0$
　　　　stop
　　else
　　　　$q_{j+1} = \beta_j^{-1}w$
　　end
end

Lanczos 算法结束。

由 Lanczos 算法和 Arnoldi 算法的伪代码（稍加修改就能在 MATLAB 上运行）可以看出，Lanczos 算法运算量是 Arnoldi 算法的$\frac{2}{k}$，可见，当$\mathcal{A}$是对称线性变换时，运算量可以得到很大的优化。但是，实践中为了数值稳定性，Arnoldi 算法和 Lanczos 算法都需要进行重正交操作，因此，实践中，Arnoldi 算法和 Lanczos 算法没有太大的区别，区别在于求解方程时候 Lanczos 算法可以产生递推公式来求解，大大优化了内存，使得算法不需要经常重启，在后来我们会讲到。

由上述伪代码可知，Arnoldi 和 Lanczos 算法不仅得到了 Krylov 子空间$\varkappa_k(\mathcal{A},b)$的一组标准正交基，并且得到了线性变换$\mathcal{A}$在$\varkappa_k(\mathcal{A},b)$的一组标准正交基下的投影$H_k$。

4.3 基于 Krylov 子空间求解$\mathcal{A}x=b$的 GMRES 和 MINRES 方法

所谓用 Krylov 子空间求解是指，构造一个 Krylov 子空间

$$\varkappa_k(\mathcal{A},b)=span\{b,\mathcal{A}b,\cdots,\mathcal{A}^{k-1}b\}$$

然后在 Krylov 子空间$\varkappa_k(\mathcal{A},b)$中找到一个向量$x_k$以某种意义最佳逼近$\mathcal{A}x=b$的真实解。这里我们介绍两种常用的标准：一类是剩余极小化标准；另一类是剩余正交化标准：

所谓的剩余极小化标准，是求$x_k\in\varkappa_k(\mathcal{A},b)$使得

$$\|r_k\|=min\{\|b-\mathcal{A}x\|:x\in\varkappa_k(\mathcal{A},b)\} \tag{4-1}$$

其中$r_k=b-Ax_k$称为x_k的剩余向量，也称残量。由此准则可以导出解对称线性方程组的极小剩余法（MINRES 法）和解非对称线性方程组的广义极小剩余法（GMRES 法）等方法。

所谓的剩余正交化标准，是求$x_k\in\varkappa_k(\mathcal{A},b)$使得

$$r_k\perp\mathfrak{X}_k \tag{4-2}$$

其中$\mathfrak{X}_k$另一个选定的k维子空间，条件(4-10)经常被称为 Galerkin 条件。由这一准则可以导出双共轭梯度法(BCG 方法)，共轭梯度平方法（CGS 方法）和稳定的双共轭梯度法（BICGSTAB 法）等方法，这类方法通常称为投影类方法。

需要补充说明的是，为何在 Krylov 子空间

$$\varkappa_k(\mathcal{A},b)=span\{b,\mathcal{A}b,\cdots,\mathcal{A}^{k-1}b\}$$

中寻找一个近似解。这是因为由 Caylay-Hamilton 定理[19]可以证明，线性变换$\mathcal{A}$在V的一组基下的矩阵下A对应的特征多项式

$$det(\lambda I-A)=a_m\lambda^m+\cdots+a_1\lambda+a_0$$

就是它的化零多项式。即$a_mA^m+\cdots+a_1A+a_0I=0$。即$a_m\mathcal{A}^m+\cdots+a_1\mathcal{A}+a_0\mathcal{E}=0$。由于我们假定矩阵$\mathcal{A}$可逆，因此$a_0\neq0$，所以$\frac{1}{a_0}(a_m\mathcal{A}^{m-1}+\cdots+a_1\mathcal{E})\mathcal{A}=I$，即

$\mathcal{A}^{-1}=\frac{1}{a_0}(a_m\mathcal{A}^{m-1}+\cdots+a_1\mathcal{E})$,所以真实解$x=\mathcal{A}^{-1}b=\frac{1}{a_0}(a_m\mathcal{A}^{m-1}b+\cdots+a_1b)\in\varkappa_m(\mathcal{A},b)$即真实解在$\varkappa_m(\mathcal{A},b)$。如果 Krylov 子空间在第$l$项终止，则$\varkappa_l(\mathcal{A},b)=$

$\varkappa_m(\mathcal{A},b)$，因此，我们有理由在其子空间$\varkappa_k(\mathcal{A},b)$寻找近似解，当$k$足够大时，求得的解即是真实解。下面我们介绍基于剩余极小化标准的 GMRES 方法。

4.3.1 GMRES 法

我们取$q_1=b/\|b\|_2$为初始向量，利用 Arnoldi 算法得到一个长度为k的 Arnoldi 分解

$$\mathcal{A}Q_k=Q_{k+1}\widetilde{H_k} \tag{4-3}$$

其中$Q_{k+1}=[Q_k,q_{k+1}]$是内积空间V的一组标准正交基。$\widetilde{H_k}$ 是不可约上 Hessenberg 矩阵。$\varkappa_k(\mathcal{A},b)=\Re(Q_k)$，我们要在$\varkappa_k(\mathcal{A},b)$中找$x_k$使其满足式(4-1)。

因为$x_k\in\varkappa_k(\mathcal{A},b)=\Re(Q_k)$ 即$x_k\in\varkappa_k(\mathcal{A},b)$等价于存在$y_k\in F^{k\times k}$使得$x_k=Q_ky_k$。因此，任意$x=Q_ky\in\varkappa_k(\mathcal{A},b)$有

$$\|b-\mathcal{A}x\|=\|b-\mathcal{A}Q_ky\|=\left\|\beta_0Q_{k+1}e_1-Q_{k+1}\widetilde{H_k}y\right\|=\left\|\beta_0e_1-\widetilde{H_k}y\right\| \tag{4-4}$$

其中$\beta_0=\|b\|$。由于$\widetilde{H_k}$是上 Hessenberg 矩阵，我们可以通过k个 Given 变换

$$G_j=diag\left(I_{j-1},\begin{bmatrix}c_j & s_j\\ -s_j & c_j\end{bmatrix},I_{k-j}\right),c_j^2+s_j^2=1$$

使得

$$G\widetilde{H_k}=\begin{bmatrix}R_k\\0\end{bmatrix} \tag{4-5}$$

其中$G=G_k\cdots G_1$，$\gamma_i\neq0,i=1,\cdots k$

$$R_k=\begin{pmatrix}\gamma_1 & \delta_1 & \varepsilon_1 & & \\ & \gamma_2 & \delta_2 & \ddots & \\ & & \ddots & \ddots & \varepsilon_{k-2}\\ & & & \ddots & \delta_{k-1}\\ & & & & \gamma_k\end{pmatrix}$$

因此

$$\left\|\beta_0e_1-\widetilde{H_k}y\right\|_2^2=\left\|G(\beta_0e_1-\widetilde{H_k}y)\right\|_2^2=\left\|\begin{bmatrix}R_k\\0\end{bmatrix}y-\begin{bmatrix}t_k\\\rho_k\end{bmatrix}\right\|_2^2=\|R_ky-t_k\|_2^2+\rho_k^2$$

其中，$\begin{bmatrix}t_k\\\rho_k\end{bmatrix}=G(\beta_0e_1)$，$\mathrm{t_k}=(\tau_1,\cdots,\tau_\mathrm{k})^\mathrm{T}$。直接计算可知：

$$\tau_1=\beta_0c_1,\tau_j=(-\mathbf{1})^{j-1}\beta_0s_1\cdots s_{j-1}c_j\ ,\mathrm{j}=2,\cdots,k,\rho_k=(-\mathbf{1})^{k}\beta_0s_1\cdots s_k \tag{4-6}$$

因此式(4-9)等价于

$$\|r_k\|_2^2=min\{\|R_ky-t_k\|_2^2+\rho_k^2:\ x=Q_ky\in\varkappa_k(A,b)\} \tag{4-7}$$

因为$\widetilde{H_k}$是不可约上三角矩阵，所以上三角矩阵R_k对角元全非 0，即R_k可逆，当$y_k=R_k^{-1}t_k$时，上式达到最小值。因此当$x_k=Q_ky_k$时，$\|r_k\|_2=min\{\|b-Ax\|_2:x\in\varkappa_k(A,b)\}=|\rho_k|$。故而在实际计算时可以将$|\rho_k|/\beta_0\leq\varepsilon$作为迭代收敛准则。至此，已经完成了 GMRES 方法的核心步骤。

然而 GMRES 最大的缺点是，我们需要保存Q_k所有的元素，存储量是$O(kN)$的，而计算机内存是有限的，随着k的增加，如果出现的近似解x_k无法满足我们的精度要求，

k就不能增加了，那么这是我们就需要使用循环的 GMRES 方法，它的基本做法是，先选取一个不太大的正整数m用来产生x_m作为初始向量重新开始，直到近似解x_m达到我们需要的精度循环中断，算法结束，该方法通常称循环 GMRES(m)方法。具体过程这里就不给出了，详见[13,p219]。

我们将求解$\mathcal{A}$x = b的 GMRES 方法总结如下：

1. 令$q_1 = b/\|b\|_2$。根据 Arnoldi 算法产生一个长度为为k的 Arnoldi 分解。
2. 利用 givens 变换求$\widetilde{H_k}$ 的 QR 分解(4-5)
3. $R_k y_k = t_k$得到y_k
4. 计算$x_k = Q_k y_k$。
5. 残量范数小于给定精度终止，否则继续增加k直到达到最大允许迭代步数。

当$\mathcal{A}$是对称线性变换时，Arnoldi 算法相应的变成了 Lanczos 算法，计算量显著下降，并且利用$\mathcal{A}$的对称性，可以得到一组x_k的递推公式，而不用存储V基Q_k，从而大大优化了内存。并且提高了算法，这就是下一小节介绍的 MINRES 法。

4.3.2 MINRES 法

这里特别需要强调的，MINRES 方法只适合于$\mathcal{A}$是对称线性变换的情形 MINRES 法与 GMRES 前面的操作完全类似，只是相应的将 Arnoldi 算法得到 Arnoldi 分解换成 Lanczos 算法得到 Lanczos 分解。其它步骤完全一致，但是由于 MINRES 的特殊性可以在不明确给出y_k的情况下，由y_k直接导出x_k的递推公式，对内存做了相当大的优化，从而避开了 GMRES 方法需要存储Q_k的列向量而需要重启的缺点。下面我们来导出递推公式，详见[13,p213-214]。

如上所述，MINRES 前面的操作与 GMRES 方法完全类似，只是相应的将 Arnoldi 算法得到 Arnoldi 分解换成 Lanczos 算法得到 Lanczos 分解，将上 Hessenberg 矩阵H_k换成对称三对角矩阵

$$\widetilde{T_k} = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \ddots & \alpha_k \\ & & & & \beta_k \end{pmatrix}$$

我们使用 4.3.1 中的记号。由(4-6)知道

$$\|r_k\|_2^2 = min\{\|R_k y - t_k\|_2^2 + \rho_k^2 : x = Q_k y \in \varkappa_k(A, b)\}$$

因此

$$y_k = R_k^{-1} t_k \tag{4-8}$$

所以

$$x_k = Q_k y_k = Q_k R_k^{-1} t_k = P_k t_k \tag{4-9}$$

其中$P_k = Q_k R_k^{-1}$。这样我们只需算出P_k即可。另$P_k = [p_1, \cdots, p_k]$。比较等式$P_k R_k = Q_k$两边各列可知

$$\begin{aligned} &\gamma_1 p_1 = q_1 \\ &\delta_1 p_1 + \gamma_2 p_2 = q_2 \\ &\varepsilon_{j-2} p_{j-2} + \delta_{j-1} p_{j-1} + \gamma_j p_j = q_j\,, \qquad j = 3, \cdots, k \end{aligned}$$

由此求出P_k的列向量为

$$\begin{aligned} &p_1 = q_1/\gamma_1 \\ &p_2 = (q_2 - \delta_1 p_1)/\gamma_2 \\ &p_j = (q_j - \varepsilon_{j-2} p_{j-2} - \delta_{j-1} p_{j-1})/\gamma_j\,, \qquad j = 3, \cdots, k \end{aligned} \tag{4-10}$$

当 Lanczos 分解的长度从k增加到$k+1$，我们有

$$\begin{aligned} &\widetilde{T_{k+1}} = \begin{bmatrix} \widetilde{T_k} & \widetilde{t_{k+1}} \\ 0 & \beta_{k+1} \end{bmatrix}, \qquad \widetilde{t_{k+1}} = (0, \cdots, 0, \beta_k, \alpha_{k+1})^T \\ &R_{k+1} = \begin{bmatrix} R_k & r_{k+1} \\ 0 & \gamma_{k+1} \end{bmatrix}, \qquad r_{k+1} = (0, \cdots, 0, \varepsilon_{k-1}, \delta_k)^T \end{aligned} \tag{4-11}$$

其中

$$\varepsilon_{k-1} = s_k \beta_k,\ \widetilde{\beta_k} = c_{k-1} \beta_k$$

$$\delta_k = c_k \widetilde{\beta_k} + s_k \alpha_{k+1},\ \widetilde{\alpha_{k+1}} = -s_k \widetilde{\beta_k} + c_k \alpha_{k+1}$$

γ_{k+1}是由第$k+1$个 Givens 变换确定的，即$\gamma_{k+1} = c_{k+1}\widetilde{\alpha_{k+1}} + s_{k+1}\beta_{k+1}$。根据(4-7)有

$$\tau_{k+1} = \rho_k c_k\,, \rho_{k+1} = \rho_k s_{k-1}$$

根据(4-10)和(4-11)我们有

$$P_{k+1} = Q_{k+1} R_{k+1}^{-1} = [Q_k, q_{k+1}] \begin{bmatrix} R_k^{-1} & * \\ 0 & \gamma_{k+1}^{-1} \end{bmatrix} = [P_k, p_{k+1}]$$

因此，最终我们得到

$$x_{k+1} = P_{k+1} t_{k+1} = [P_k, p_{k+1}] \begin{bmatrix} t_k \\ \tau_{k+1} \end{bmatrix} = x_k + \tau_{k+1} p_{k+1}$$

该递推公式求解方法被称为极小剩余法（MINRES）

5 块 Krylov 子空间方法求解 Sylvester 方程

由式(2-3)知 Sylvester 方程(1-1)可以化成$Sx = c$ 的形式，而由第 4 章知求解内积空间中方程$\mathcal{A}x = b$可以使用 GMRES 方法，因此，我们可以将(1-1)化成(2-3)再利用 GMRES 方法求解，但是，一方面将(1-1)转化成(2-3)会使得线性方程的阶数平方级增加，而计算机内存是有限的，另一方面没有充分利用$S = I_s \otimes A - B^T \otimes I_N$的特殊性。因此上述做法是不可取的，但是我们可以隐式的运行 GMRES 方法，对原方程(1-1)类似使用求解$\mathcal{A}x = b$的方式构造 Krylov 子空间，在构造的子空间寻找某种意义的最优近似解。这种方法也称为块 Krylov 子空间方法，下面我们将展示我们这种做法的细节。

5.1 Krylov-Sylvester 子空间[12, 22]

我们再次提及 Sylvester 方程

$$AX - XB = C$$

其中$A \in R^{N\times N}$，$B \in R^{s\times s}$，$C \in R^{N\times s}$。

我们定义线性变换S：$R^{N\times s} \to R^{N\times s}$

$$S(X) = AX - XB$$

由S 和C可以 生成 Krylov 子空间

$$\varkappa_k(S, C) = \{C, S(C), \cdots, S^{k-1}(\mathrm{C})\}$$

其中$S^i(C) = S(S^{i-1}(C))$以及$S^0(C) = C$。我们将由 Sylvester 方程产生的 Krylov 子空间称作 Krylov-Sylvester 子空间

下面定理说明 Krylov-Sylvester 子空间与$\varkappa_k(A, C)$是等价的。

定理 5.1 对任意$m > 1$，$\varkappa_m(S, C) = \varkappa_m(A, C)$。

证明：直接计算可知

$$\forall m \geq 0\ ,\ S^m(C) = \sum_{i=0}^{m}\binom{m}{i}A^{m-i}C(-B)^i\ ,\ \text{其中}\binom{m}{i} = \frac{m!}{i!(m-i)!}$$

因此$K_m(S, C) \subset K_m(A, C)$。

反方向的结论可以由归纳法证明。这个包含关系当$m = 1$时是显然的。假设对于$k = 1, \cdots, m$成立$K_m(A, C) \subset K_m(S, C)$那么

$$A^m C = \sum_{i=0}^{m}\binom{m}{i}A^{m-i}C(-B)^i - \sum_{i=1}^{m}\binom{m}{i}A^{m-i}C(-B)^i$$

$$= S^m(C) - \sum_{i=1}^{m}\binom{m}{i}A^{m-i}C(-B)^i \in K_{m+1}(S, C)$$

即$K_{m+1}(A, C) \subset K_{m+1}(S, C)$。证毕。

值得一提的是，在早期的很多文献中，往往需要条件C是满秩矩阵，这是因为早期 V. Simoncini[24]在该条件下证明了定理 5.1。然而实际上这个条件是多余的。

由定理 5.1 可知，我们只需考察由矩阵A, C生成$\varkappa_k(A, C)$就可以得到我们需要的

Krylov-Sylvester 子空间$\varkappa_k(S,C)$。因此我们把$\varkappa_k(A,C)$也称作 Krylov-Sylvester 子空间，下面我们只考察$\varkappa_k(A,C)$。

我们想在 Krylov-Sylvester 子空间$\varkappa_k(A,C)$中找到一个矩阵X_k以某种意义最佳逼近$AX-XB=C$的真实解。但是我们现在还没有给出一个度量矩阵大小的范式，下面我们先定义矩阵的一种常见内积以及它诱导出的 Frobenius 范式。

对于$X,Y\in R^{N\times s}$定义它们的内积为

$$\langle X,Y\rangle_F = trace(X^TY)$$

其中X^T是X的转置，$trace(T)$是方阵T的对角元素之和。

实际上$R^{N\times s}$关于内积$\langle X,Y\rangle_F$构成了一个有限维内积空间，因此根据第 4 章，我们可以构造一个 Arnoldi 分解得到 Krylov 子空间$\varkappa_k(S,C)$的一组基，再在这组基下求剩余最小化问题(4-1)下面我们就具体这样实施。

5.2 修正的全局 Arnoldi 算法[22]

我们类比 Arnoldi 算法构造Krylov子空间的 Arnoldi 算法，用修正的全局 Arnoldi 算法构造$\varkappa_k(A,C)$的一组 F-正交基$\{V_1,\cdots,V_k\}$。即$\mathrm{trace}(V_i,V_j)=0$且$\mathrm{trace}(V_i,V_i)=1$对任意$i\neq j,i,j=1,\cdots,k$成立。算法如下所示：

修正的全局 Arnoldi 算法（记作算法 A1）

选择一个$N\times s$的矩阵V_1使得$\|V_1\|_F=1$。

for $j=1,\cdots,k$

  $W=AV_j$

  for $i=1,\cdots,j$

    $h_{ij}=trace(V_i^TW)$

    $W=W-h_{ij}V_i$

  end

  $h_{j,j+1}=\|W\|_F$

  if $h_{j,j+1}=0$

    stop

  end

  $V_{j+1}=W/h_{j,j+1}$

end

算法 A1 结束

我们用v_k表示$N\times ks$矩阵$[V_1,\cdots,V_k]$。$\widehat{H}_k$表示$(k+1)\times k$不可约上 Hessenberg 矩阵。其非 0 元在算法 A1 中以及给出。H_k是$\widehat{H}_k$去掉最后一列得到的不可约上 Hessenberg 矩阵。我们用$H_{:,j}$表示H_k的第j列。下面我们具体介绍块方法的两种常见形式。

5.3 全局类 FOM-Sylvester(GFSL)算法[22]

我们应用 5.1 和 5.2 节的记号，给定初始猜测解X_0以及对应的残量$R_0 = C - S(X_0)$。GFSL 方法在第k步得到近似解X_k使得

$$X_k - X_0 \in \varkappa_k(S, C) \tag{5-1}$$

且残量R_0满足正交性

$$R_k = C - S(X_k) \perp_F \varkappa_k(S, C) \tag{5-2}$$

由定理 5.1 知$\varkappa_k(S, C) = \varkappa_k(A, C)$，因此关系式(5-1)和(5-2)可改写为

$$X_k - X_0 \in \varkappa_k(S, C) \tag{5-3}$$

$$R_k = C - S(X_k) \perp_F \varkappa_k(S, C) \tag{5-4}$$

式(5-8)存在$y_k \in R^k$表明

$$X_k = X_0 + v_k * y_k$$

再由(5-9)得到

$$< R_0 - A(v_k * y_k) + (v_k * y_k)B, V_i >_F = 0, i = 1, \cdots, k \tag{5-5}$$

$$\sum_{i=1}^{k} < AV_i - V_iB, V_i >_F y_k^{(i)} = < R_0, V_i >_F, i = 1, \cdots, k \tag{5-6}$$

其中$y_k = (y_k^{(1)}, \cdots, y_k^{(k)})^{\mathrm{T}}$。如果我们再算法 A1 中选择$V_1 = R_0/\|R_0\|_F$。那么式(5-5)等价于

$$(H_k - P_k)y_k = \|R_0\|_F e_1 \tag{5-7}$$

其中e_1是单位阵I_k的第一列，H_k是由算法 A1 得到的，$(P_k)_{i,j} =< V_jB, V_i >_{\mathrm{F}} = trace(V_i^T V_jB)$，对所有$i, j = 1, \cdots, k$成立。现在具体给出 GFSL 算法：

GFSL 算法（记作算法 M5）

(1) 选择一个初始矩阵X_0，计算$R_0 = C - AX_0 + X_0B$以及$V_1 = R_0/\|R_0\|_F$。

(2) 算法 A1 作用于块 Krylov 子空间$\varkappa_k(A, R_0)$得到一组 F-正交基$\{V_1, \cdots, V_k\}$。

(3) 定义矩阵$(P_k)_{i,j} =< V_jB, V_i >_{\mathrm{F}} = trace\left(V_i^T V_jB\right),\ i, j = 1, \cdots, k$

(4) 解线性方程$(H_k - P_k)y_k = \|R_0\|_F e_1$得到$y_k$

(5) 得到近似解$X_k = X_0 + v_k * y_k$。

算法 M5 结束

由算法 M5 的步骤可知 GFSL 算法是一个简洁实用的高效块 Krylov 近似解法，下面我们来分析一下残量在第k步的界。

$$\begin{aligned}\|R_k\|_F &= \|C - AX_k + X_kB\|_F = \|R_0 - A(v_k * y_k) + (v_k * y_k)B\|_F \\ &= \|R_0 - (Av_k) * y_k + [V_1B, \cdots, V_kB] * y_k\|_F\end{aligned} \tag{5-8}$$

如果我们用$V_1, \cdots, V_k$表示V_iB，那么

$$V_iB = \sum_{j=1}^{k} < V_iB, V_j > V_j + \varepsilon_k^{(i)} \quad i = 1, \cdots, k$$

因此

$$[V_1B, \cdots, V_kB] = v_kP_k + \varepsilon_k$$

其中$\varepsilon_k = [\varepsilon_k^{(1)}, \cdots, \varepsilon_k^{(k)}]$。因此我们有

$$\|R_k\|_F = \|R_0 - (Av_k) * y_k + [V_1B, \cdots, V_kB] * y_k\|_F$$

$$= \|R_0 - (v_k H_k + Z_{k+1}) * y_k + (v_k P_k + \varepsilon_k) * y_k\|_F$$
$$= \|v_k * ((H_k - P_k)y_k - \|R_0\|_F e_1) + (-Z_{k+1} + \varepsilon_k) * y_k\|_F$$

如果我们通过$(H_k - P_k)y_k = \|R_0\|_F e_1$计算得到$y_k$

那么我们有

$$\|R_k\|_F = \|(Z_{k+1} - \varepsilon_k) * y_k\|_F$$

当$\varepsilon_k = 0$时，我们可以得到

$$\|R_k\|_F = \|Z_{k+1} * y_k\|_F = h_{k+1,k}|e_k^T y_k|$$

至此 GFSL 算法的步骤和残量的界都以给出。

5.4 全局类 GMRES-Sylvester(GGSL)算法[22]

由于全局类 GMRES-Sylvester(GGSL)方法和全局全局类 FOM-Sylvester(GFSL)方法几乎完全一致，因此这里只是简单的给出 GGSL 方法的运算步骤

GGSL 算法(记作算法 M6)

(1) 选择一个初始矩阵X_0，计算$R_0 = C - AX_0 + X_0 B$以及$V_1 = R_0/\|R_0\|_F$。

(2) 算法 A1 作用于块 Krylov 子空间$\varkappa_k(A, R_0)$得到一组 F-正交基$\{V_1, \cdots, V_k\}$。

(3) 定义$(k+1) \times k$矩阵$(\hat{P}_k)_{i,j} =< V_j B, V_i >_{\mathrm{F}},\ i = 1, \cdots, k+1, j = 1, \cdots, k$

(4) 解极小化问题$\|\|R_0\|_F e_1 - (H_k - P_k)y_k\|_F$得到$y_k$

(5) 得到近似解$X_k = X_0 + v_k * y_k$。

算法 M6 结束

本质上，GGSL 和 GFSL 方法的唯一区别在于步骤(4)。GGSL 通过求解最小化问题得到y_k而 GFSL 是通过解方程得到y_k。其它过程完全类似，这里就不在雷同介绍了。

6. 全局投影方法求解 Sylvester 方程

本章是全文的重点，本章将详细的给出求解 Sylvester 方程的全局投影方法，详细给出计算步骤，编写该方法的 MATLAB 代码，就时间复杂度，空间复杂度以及精度多方面与前面提到的直接法，块方法，以及 MATLAB2014 之后加入的 sylvester 内建函数进行数值比较。

首先给出整个求解思路，我们使用考虑 Krylov 子空间$\varkappa_k(S,c)$中找一个解x_k达到极小剩余标准，即本质上是套用第四章的有限维内积空间求解$\mathcal{A}x=b$的 GMRES 方法。但是我们利用了S的特殊性，并且隐式的利用第四章的结论，使得这种套用无论从内存还是时间上都有极大的优化。下面我们给出具体的操作。

由式(2-3)知，方程(1-1)可以转化成

$$Sx=c \tag{6-1}$$

其中$S=I_s\otimes A-B^T\otimes I_N, x=vec(X), c=vec(C)$。

对式(5-1)我们使用第四章的结论，这里我们给出 Arnoldi 分解，再求 QR 分解，最后得到近似解，下面我们针对方程 5.1 给出详细的操作。但是这里必须提出的是，我们并不明确的给出矩阵S，在求Sx时，我们利用

$$Sx=vec(AX-XB) \tag{6-2}$$

其中$vec(X)=x$。这样做既节约了内存，由提高了运算效率。

Arnoldi 算法

取$q_1=b/\|b\|$
for $j=1:k$
　$w=Sq_j$　　这一步计算参考(5-2)式
　for $i=1:j$
　　$h_{ij}=dot(w,q_i)$　这里就是向量的内积
　　$w=w-h_{ij}q_i$
　end
　$h_{j+1,j}=\|w\|$　　这里就是向量的模
　if $h_{j+1,j}=0$
　　stop
　else
　$q_j=w/h_{j+1,j}$　　单位化
　end
end

Arnoldi 算法结束

根据第 4 章，我们对$\widetilde{H_k}$进行 QR 分解，再求解$R_ky_k=t_k$得到y_k，再计算$x_k=Q_ky_k$。当残量范数小于给定精度终止，否则继续增加k直到达到最大允许迭代步数。由于$\widetilde{H_{k+1}}$的

前$(k+1)\times k$阶子式是$\widetilde{H_k}$，因此我们不用每次都求一遍$\widetilde{H_{k+1}}$。可以在进行每一步 Arnoldi 算法的过程中进行 QR 分解，求解$R_k y_k = t_k$得到y_k，计算$x_k = Q_k y_k$，直到达到给定精度或者达到最大迭代步数。

为了清晰给出每一步的详细操作，下面给出该方法的 MATLAB 求解代码如下：

```
function [X,res] = sylGmres(A,B,C,tol,maxit)
%---------------------------------------
% 求解方程 AX-XB=C
%---------------------------------------
[n,m]=size(C);
Cnorm = norm(C,'fro');
if Cnorm == 0          % 矩阵 C 为 0 矩阵，直接输出答案
   X = zeros(n,m);
   res = 0;
   return;
end
tol = tol*Cnorm;       % 相对误差转绝对误差
res = zeros(maxit,1);
V = zeros(n*m,maxit);  % 初始化
V(:,1)=C(:)/Cnorm;
H = zeros(maxit+1,maxit);
X=zeros(n,m);
for j=1:maxit
   MV = reshape(V(:,j),n,m);
   W = A*MV-MV*B;            % 转化成矩阵进行运算
   W = W(:);                 % 运算完成变回向量
   for i=1:j                 % Arnodi 方法正交化
      H(i,j) = dot(V(:,i),W);
      W = W - H(i,j)*V(:,i);
   end
   for i=1:j                 % 重正交化
      s = dot(V(:,i),W);
      H(i,j) = H(i,j)+s;
      W = W-s*V(:,i);
   end
   H(j+1,j) = norm(W);       % 为第 j＋1 步做准备
   if H(j+1,j) < tol         % 如果无法迭代只能退出了。
      break;
   end
```

```
    V(:,j+1) = W/H(j+1,j);
    [Q,R]=qr(H(1:j+1,1:j));      % 对 H 进行 qr 分解
    y=R(1:j,:)\(Cnorm*Q(1,1:j)');% 求解 y
    X = reshape(V(:,1:j)*y(1:j),n,m); % 计算本次最佳 X
    res(j) = norm(A*X-X*B-C,'fro');   % 求第 j 次的残量
    if res(j) < tol                   % 达到精度终止
       break;
    end
end
res = res(1:j)/Cnorm;   %纪录下迭代过程中的相对残量范式
end
```

由以上代码可以清晰的给出如果在每一步 Arnoldi 算法中进行求解，在真个过程中由于k相对较小，又由于$\widetilde{H_k}$是上 Hessenberg 的，所以一次 QR 分解复杂度为k^2，整个过程的 QR 分解复杂度为k^3由于k相对较小，几乎可以忽略。同样的每次计算 y 的复杂度也是k^2几乎可以忽略，计算量最大的部分其实是 Arnoldi 分解式求内积和计算W部分，复杂度为$\mathrm{N}^2 \times \mathrm{s} + \mathrm{N} \times \mathrm{s} \times k^2$。相对之前提到的直接方法时间复杂度有了很大的提升，而空间复杂度主要集中在向量V上。空间复杂度为$\mathrm{N} \times \mathrm{s} \times k$。

7 数值测试

我们现在对本文中提到的算法 M1-M3 以及全局投影法进行数值测试，从时间复杂度，空间复杂度和精度等多方面进行比较，给出这算法 M1-M3 各自的使用范围，本次数值测试的笔记本电脑的硬件配置如下：

处理器 CPU ： Inter Core i5 1.6 GHz 双核

二级缓存 L2： 256KB

三级缓存 L3：3MB

内存 Memory：4GB

软件使用 MATLAB2014b

7.1 算法 M1 测试

先给出算法 M1 源代码 M1.m 以及测试代码 testM1.m

```
function X=M1(A,B,C)
%---------------------------------------
% 求解 Sylvester 方程 AX-XB=C 记作算法 M1
%---------------------------------------
[N,s]=size(C);     %得到 A 和 B 的阶数 N 和 s
S=kron(eye(s),A)-kron(B',eye(N)); %将原方程转化成 Sx＝b
X=S\C(:);          %直接求解转换后的矩阵
X=reshape(X,N,s);  %将得到的向量转换成矩阵得到最终解
end

%test M1
clear
len = 20;
N = round(10.^linspace(1,2.4,len));       %取 10-251 中的 20 个点
s = round(10.^linspace(1,1.2,len)); %取 10-16 中的 20 个点
m1time = zeros(1,len);                    %各个点对应的运算时间
m1res = zeros(1,len);                     %各个点对应的残量范数
for i=1:len
   A=rand(N(i));
   B=rand(s(i));
   C=rand(N(i),s(i));                     %产生随机矩阵 A，B，C 用于测试
   t0 = cputime;
   X=M1(A,B,C);
   m1time(i)=cputime-t0;
```

```
    m1res(i)=norm(A*X-X*B-C);                    %计算每次计算的时间和残差范式
end
plot(N.*s,m1time);
title('Guass 直接法求 Sylvester 方程时间分析','FontSize',12);
xlabel('矩阵行列乘积 N*s','FontSize',12);
ylabel('运行时间 time','FontSize',12);
pause;

plot(N.*s,m1res);
title('Guass 直接法求 Sylvester 方程残量分析','FontSize',12);
xlabel('矩阵行列乘积 N*s','FontSize',12);
ylabel('残量 norm(AX-XB-C)','FontSize',12);
```

由于算法 M1 效率相对较差，这里就不在图形上与 M2 或 M3 比较了，并且由于内存限制，算法 M1 只能求解小阶数的方程。根据测试数据，我们有下面两张表。

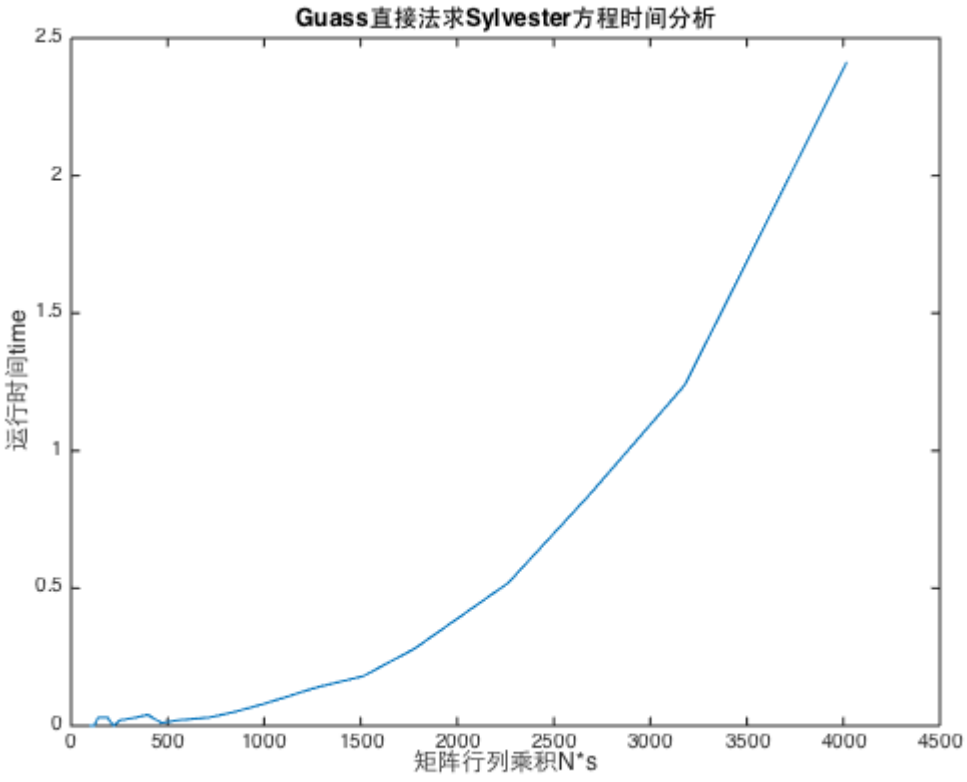


图 7-1

上图是有 testM1.m 文件生成的图，横轴是矩阵 C 的行列之积，纵轴是对于的运行时间，可以看出算法 M1 大致与三次函数类似，与我们理论分析结论相近，可以当$N \times s = 1000$是，运行时间只有不足 0.1 秒，但是，当$N \times s = 4000$时，运行一次算法 M1 所需的时间是 2.5s。这是很难让人接受的。由附录中代码可知，算法 M1 的代码复

杂度十分低，因此，如果我们求解的 Sylvester 方程阶数满足$N \times s = 1000$，并且我们使用频率并不高时，可以考虑使用算法 M1 计算小规模 Sylvester 方程。

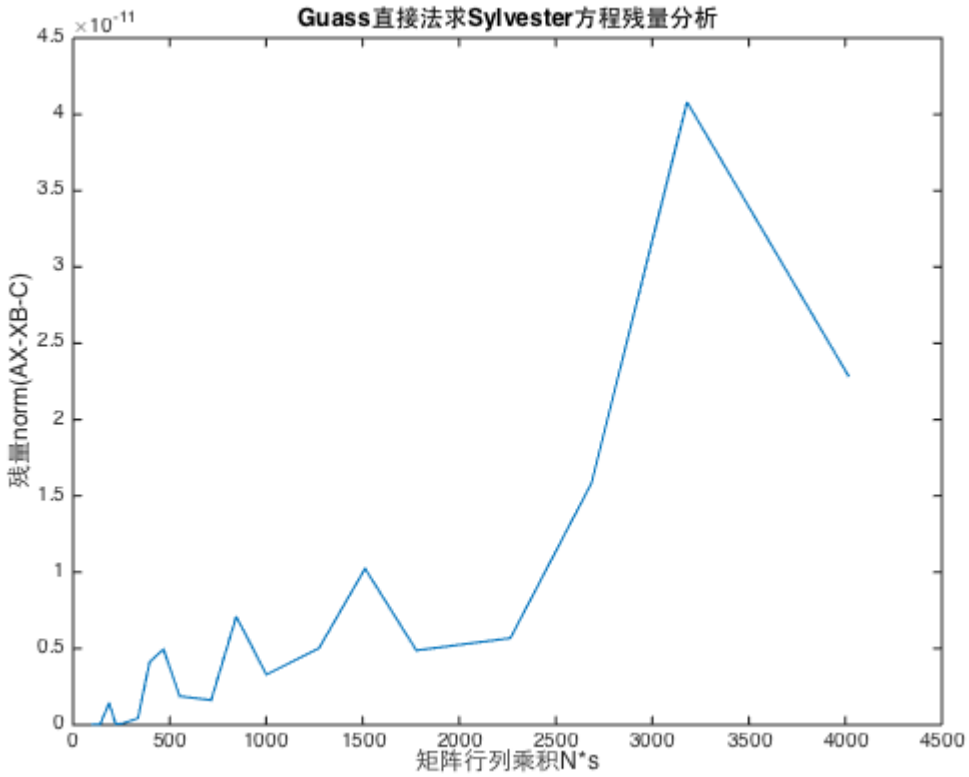


图 7-2

这里需要注意的是，图 7-2 虽然看上去十分凌乱，但是实际上纵轴是被乘以了 1e-10 次方的，也就是说，本次实验中，解产生的残量都不超过 1e-10。因此数值上看，小阶数算法 M1 还是相对稳定的算法。

7.2 算法 M2 和 M3 比较测试

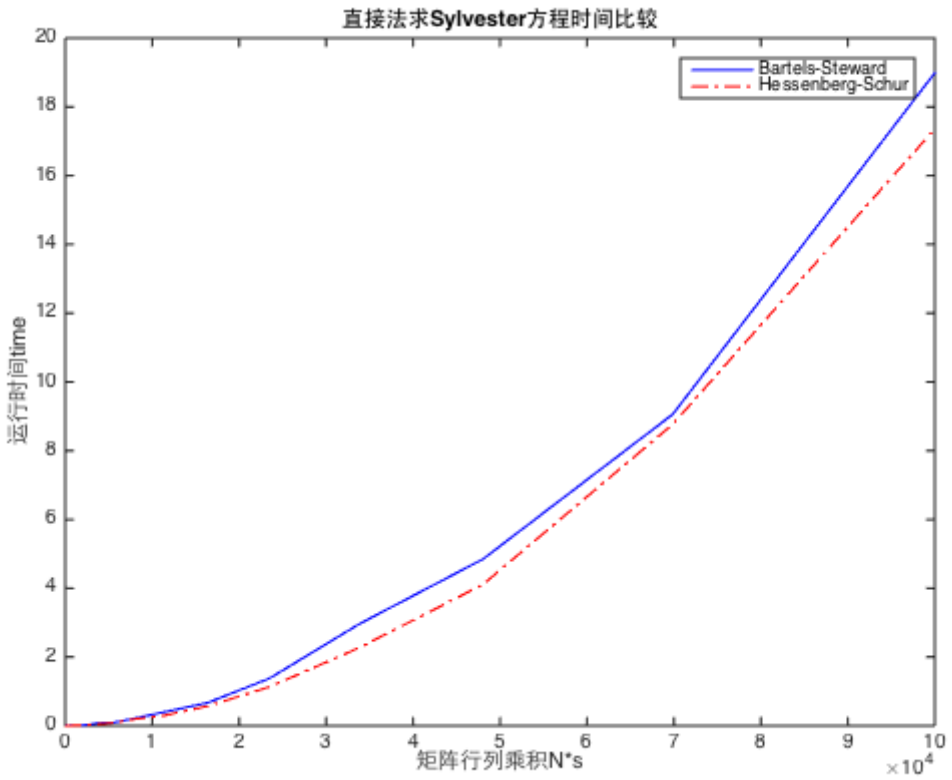
先给出算法 M2 和 M3 源代码 M2.m，M3.m 以及中间函数 mySyl.m 以及测试代码 testM2M3.m

```
function Y=mySyl(R,S,F)
%--------------------------------
% slove RY-YS=F S 是下 Hessenberg 矩阵
%--------------------------------
[m,n]=size(F);
Y=zeros(m,n);
i = n;
while i>0
   if i==1||S(i-1,i)==0                %如果 S(i-1,i)=0，那么直接求解
      R(1:m+1:m*m)=R(1:m+1:m*m)-S(i,i);
```

```
        Y(:,i)=R\F(:,i);
        R(1:m+1:m*m)=R(1:m+1:m*m)+S(i,i);
        F(:,1:i)=F(:,1:i)+Y(:,i)*S(i,1:i);
        i = i-1;
    else                                %如果 S(i-1,i)=0，转化成 2m*2m 阶求解。
        TY = M1(R,S(i-1:i,i-1:i),F(:,i-1:i));
        Y(:,i-1)=TY(1:m);
        Y(:,i)=TY(m+1:end);
        F=F+Y(:,i-1:i)*S(i-1:i,:);
        i = i-2;
    end
end
end

function X=M2(A,B,C)
%--------------------------------------
% 求解 Sylvester 方程 AX-XB=C 记作算法 M2
%--------------------------------------
[U,R]=schur(A);    %对 A 运用 Schur 分解得到正交阵 U 使得 U'AU=R
[V,S]=schur(B');   %与上面类似
F=U'*C*V;          %将原方程转化成 RY-YS‘=F
Y=mySyl(R,S',F);   %对简单形式直接求解
X=U*Y*V';          %由简单形式解得原方程解
end

function X=M3(A,B,C)
%--------------------------------------
% 求解 Sylvester 方程 AX-XB=C 记作算法 M3
%--------------------------------------
[U,R]=hess(A); %对 A 运用 householder 变换得到正交阵 U 使得 U'AU=R
[V,S]=schur(B');%对 B‘运用 Schur 分解得到正交阵 V 使得 V'B'V=S
F=U'*C*V;      %将原方程转化成 RY-YS‘=F
Y=mySyl(R,S',F);%对简单形式直接求解
X=U*Y*V';      %由简单形式解得原方程解
end

%test M2 and M3
clear
```

```
len = 20;
N = round(10.^linspace(1,3,len));          %取需要测试的 A 的阶数
s = round(10.^linspace(1,2,len));          %取需要测试的 B 的阶数
m2time = zeros(1,len);
m2res = zeros(1,len);
m3time = zeros(1,len);
m3res = zeros(1,len);
for i=1:len
   A=rand(N(i));
   B=rand(s(i));
   C=rand(N(i),s(i));                      %参数对应阶数的随机矩阵 A，B，C
   t0 = cputime;
   X2=M2(A,B,C);
   m2time(i)=cputime-t0;
   m2res(i)=norm(A*X2-X2*B-C);             %计算 BS 算法的运行时间和残量范数

   t0 = cputime;
   X3=M3(A,B,C);
   m3time(i)=cputime-t0;
   m2res(i)=norm(A*X3-X3*B-C);             %计算 HS 算法的运行时间和残量范数
end
plot(N.*s,m2time,'b-',N.*s,m3time,'r-.');  %绘制时间随阶数变化的图像
title('直接法求 Sylvester 方程时间比较');
legend('Bartels-Steward','Hessenberg-Schur');
xlabel('矩阵行列乘积 N*s');
ylabel('运行时间 time');
pause; %按任意键，绘制下一张图
plot(N.*s,m2res,'b-',N.*s,m3res,'r-.');    %绘制残量范数随阶数变化的图像
legend('Bartels-Steward','Hessenberg-Schur');
title('直接法求 Sylvester 方程残量比较');
ylabel('残量 norm(AX-XB-C)');
```

由于算法 M2 和 M3 十分相似，因此，我们将算法进行比较测试。



表

图 7-3

需要注意的是，这里算法 M2,M3 的运行时间虽然都差不多有 10 秒，但是这里横轴单位是 1e4 的，因此，算法相对较慢是由于这里测试的阶数很大造成的，M2 和 M3 毫无疑问是当前最好的求解 Sylvester 方程的直接方法，这里可以看出，算法 M3 确实是要优于算法 M2 的，这里由于实验中 N 和 s 比较相近，才导致我们看到的效果不是十分明显。

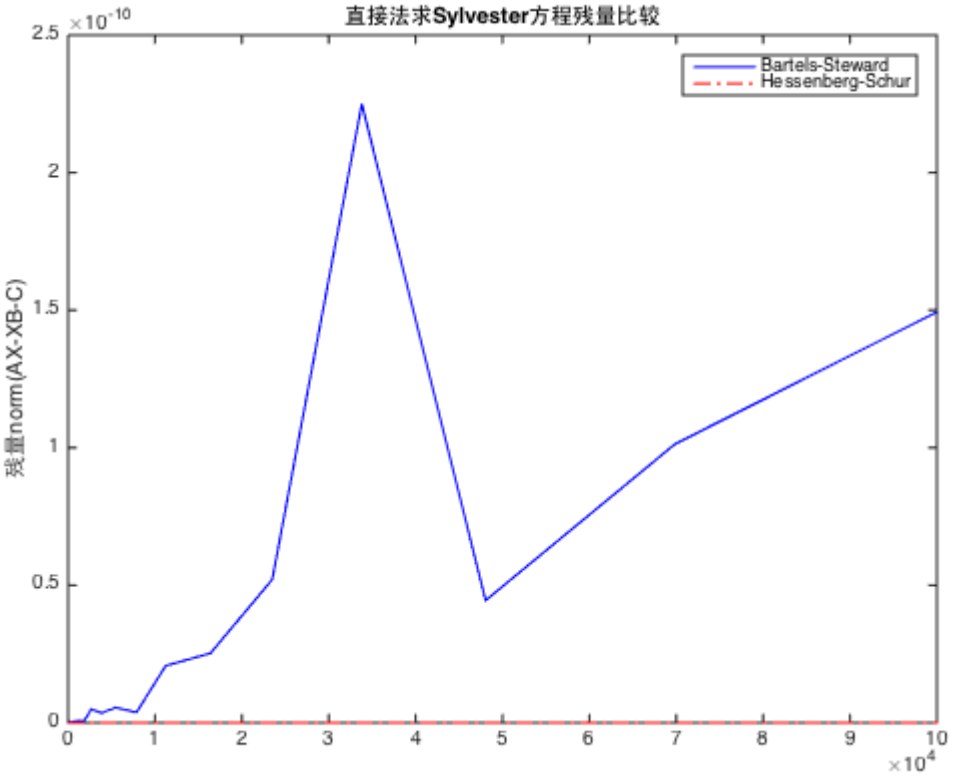


图 7-4

与从图上看残量的阶数与 M1 相差不大。这里我们看到代表算法 M3 的红色的点划线逼近与横轴，说明算法 M3 在精度上明显优于算法 M2，从各方面看，Hessenberg-Schur 算法(M3)各方面都略优于 Bartels-Stewart 算法(M2)。因此我们在实际应用中应该优先选择算法 Hessenberg-Schur 方法来解决中小规模 Sylvester 方程。当然 Hessenberg-Schur 算法是吸收了 Bartels-Stewart 算法的思路对其优化得出的，因此 Bartels-Stewart 算法是一个值得纪念并且有理论价值的算法。

7.3 全局投影方法数值测试

下面先给出测试代码

```
% test syl
clear;
n=10000;
m=100;
A = rand(n,n)+0.1*n*eye(n);  %这里的随机矩阵 A 几乎主对角线占优
%A = rand(n,n);
B = rand(m,m);
C = rand(n,m);
Cnorm = norm(C,'fro');
```

```
tol = 1e-15;                % 相对精度
maxit = 30;                 % 最大迭代步数
[X,res] = syl(A,B,C,tol,maxit); % 全局投影法求解
norm(A*X-X*B-C)/Cnorm

[dX,dres] = doublesyl(A,B,C,tol,maxit);  % 重正交全局投影法求解
norm(A*dX-dX*B-C)/Cnorm

[bX,bres] = block_syl(A,B,C,tol,maxit);  % 块全局投影方法
norm(A*bX-bX*B-C)/Cnorm

semilogy(res,'b-');
hold on
semilogy(dres,'g.');
semilogy(bres,'r-');      % 绘制残量范式的变化
legend('全局投影法','重正交全局投影法','块方法');
title('Krylov 子空间方法的迭代收敛速度分析');
xlabel('迭代步数 k');
ylabel('残量范式相对值');
```

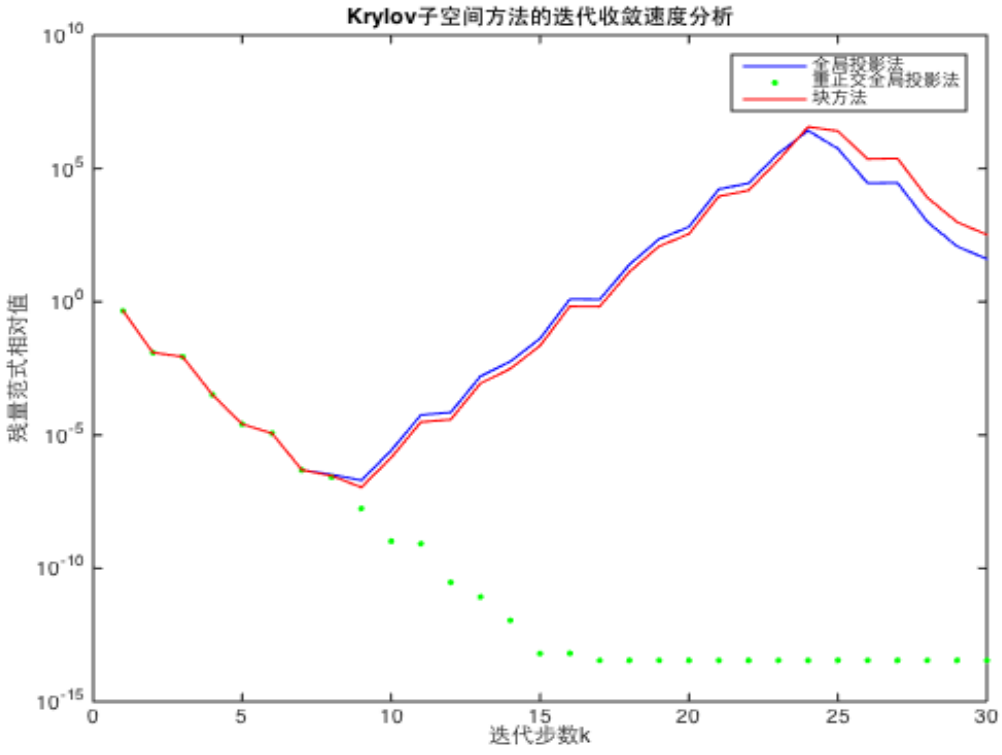


图 7-5

这里的块方法是指第 5 章提到的方法，重正交全局投影法是第 6 章给的具体代码，而投影法第 6 章中是去掉了重正交化的代码，由下表可以看出，加了正交化后，算法精度明显提高，而效率并没有收到太大的影响，因为主要计算量并不在正交化上。

7.4 各类算法时间比较

S=100	time	N=	500		800		1200	
BS 算法			0.74	4.71	2.29	10.65	4.35	29.41
HS 算法			0.49	5.85	1.12	15.67	2.61	39.51
重正交投影法			0.25	0.22	0.40	0.41	0.78	0.76
MATLAB 自带			0.44	0.44	1.15	1.11	3.07	3.01

表 7-1

S=300	time	N=	500		800		1200	
BS 算法			1.88	11.28	3.95	33.23	8.22	88.25
HS 算法			1.5	17.33	3.6	50.68	6.44	127.73
重正交投影法			1.03	0.83	1.62	1.43	2.62	2.59
MATLAB 自带			0.52	0.63	1.45	1.50	3.69	3.57

表 7-2

S=500	time	N=	500		800		1200	
BS 算法			3.58	18.88	7.01	54.83	12.93	147.79
HS 算法			3.16	29.15	6.14	85.22	11.76	210.41

重正交投影法	1.51	1.64	2.67	2.76	4.48	4.36
MATLAB 自带	0.71	1.02	2.09	2.01	4.45	4.47

表 7-3

表 7-1,7-2,7-3 分别是在矩阵B对称和对称情况下，阶数为 100,300,500 时，各个算法随N的变化而导致的时间变化，由表可知 BS 算法和 HS 算法在矩阵B为对称矩阵时的效率明显高于非对称情况，并且此时 HS 算法有时略优于 MATLAB 自带的 sylvester 函数。另外矩阵A的阶数对直接法运行时间影响更大，而对重正交投影法等迭代方法并影响不明显。但是精度方面直接法都是相当稳定的，而重正交投影法只对矩阵A稀疏和主对角占优，或者对称正定的情形收敛十分快，其它情况的收敛速度不太乐观。

7.5 数值测试总结

我们在实际计算过程中，如果需要求解中小规模($s < N < 3000$) Sylvester 方法，我们可以首选使用 Hessenberg-Schur 算法(M3)，而当$N > 3000$时我们可以考虑使用全局投影法等 Krylov 子空间方法进行求解。但是这时，我们并不能确定该方法的收敛速度是否能够达到我们的需要，因此我们还需要对 Krylov 子空间方法进行更深入的研究，给出一些预处理条件来加速迭代过程。从实验测试方面看，当我们的矩阵A近似主对角线占优时，我们的全局投影方法收敛速度很快，并且如果A是稀疏矩阵时，收敛速度也比较理想。就目前而言，基于 Krylov 子空间的方法仍然有很多值得深入研究的地方，还有很大的改进空间。

参考文献

[1] Golub G, Nash S, Van Loan C. A Hessenberg-Schur method for the problem AX+ XB= C[J]. Automatic Control, IEEE Transactions on, 1979, 24(6): 909-913.

[2] Bartels R H, Stewart G W. Solution of the matrix equation AX+ XB= C [F4][J]. Communications of the ACM, 1972, 15(9): 820-826.

[3] Datta B N. Linear and numerical linear algebra in control theory: Some research problems[J]. Linear Algebra and Its Applications, 1994, 197: 755-790.

[4] Baur U, Benner P. Factorized solution of Lyapunov equations based on hierarchical matrix arithmetic[J]. Computing, 2006, 78(3): 211-234.

[5] Enright W H. Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations[J]. ACM Transactions on Mathematical Software (TOMS), 1978, 4(2): 127-135.

[6] A. El Guennouni, K. Jbilou and J. Riquet, Block Krylov subspace methods for solving large Sylvesterequations, Preprint LMPA, No. 132 (2000), Universit édu Littoral, to appear in Numer. Algorithms.

[7] Saad Y. Numerical solution of large Lyapunov equations[M]. Research Institute for Advanced Computer Science, NASA Ames Research Center, 1989.

[8] Roth W E. The equations AX-YB= C and AX-XB= C in matrices[J]. Proceedings of the American Mathematical Society, 1952, 3(3): 392-395.

[9] Flanders H, Wimmer H K. On the Matrix Equations AX-XB= C and AX-YB= C[J]. SIAM Journal on Applied Mathematics, 1977: 707-710.

[10]詹兴致. 矩阵论[M]. 北京: 高等教育出版社, 2008. 13-18.

[11]Wilkinson J H. The algebraic eigenvalue problem[M]. Oxford: Clarendon Press, 1965.

[12]徐树方, 钱江. 矩阵计算六讲[M]. 北京: 高等教育出版社, 2011.1-27.

[13]Saad Y, Schultz M H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems[J]. SIAM Journal on scientific and statistical computing, 1986, 7(3): 856-869.

[14]Bao L, Lin Y, Wei Y. A new projection method for solving large Sylvester equations[J]. Applied numerical mathematics, 2007, 57(5): 521-532.

[15]Bao L, Lin Y, Wei Y. Krylov subspace methods for the generalized Sylvester equation[J]. Applied mathematics and computation, 2006, 175(1): 557-573.

[16]Li J R, White J. Low rank solution of Lyapunov equations[J]. SIAM Journal on Matrix Analysis and Applications, 2002, 24(1): 260-280.

[17]徐树方. 控制论中的矩阵计算[M]. 北京: 高等教育出版社, 2011.

[18]李炯生, 查建国, 王新茂. 线性代数[M]. 安徽: 中国科学技术大学出版社, 2010.

[19]Lancaster P, Tismenetsky M. The theory of matrices: with applications[M]. Academic press, 1985.

[20] Jaimoukha I M, Kasenally E M. Krylov subspace methods for solving large Lyapunov equations[J]. SIAM Journal on Numerical Analysis, 1994, 31(1): 227-251.
[21] Lieb E H. Proofs of some conjectures on permanents[M] Inequalities. Springer Berlin Heidelberg, 2002: 101-108.
[22] Salkuyeh D K, Toutounian F. New approaches for solving large Sylvester equations[J]. Applied mathematics and computation, 2006, 173(1): 9-18.
[23] Jbilou K, Messaoudi A, Sadok H. Global FOM and GMRES algorithms for matrix equations[J]. Applied Numerical Mathematics, 1999, 31(1): 49-63.
[24] Simoncini V. On the numerical solution of AX− XB= C[J]. BIT Numerical Mathematics, 1996, 36(4): 814-830.

致谢

经过一个多月的努力，终于了解并掌握了该论题的常见方法并写完本论文。在此感谢我的指导老师-鲍亮老师提供了优质的论文参考文献，耐心的查阅我的论文，并多次修正本论文中的错误，给我提供了很多宝贵的建议。感谢我的同学和朋友帮我检查论文格式问题。感谢论文中涉及的各位学者的研究成果给我的帮助和启发。

在此向帮助和指导我的各位老师以及帮助我的各位学长学姐和同学表示最衷心的感谢。

由于本人学术水平有限，论文难免有不足之处，恳请各位老师和学友批评指正。