数值分析与科学计算课程报告

赵天钧

学号：23210180134

2023 年 12 月 19 日

1 一阶方法

在处理优化问题时，如果只有一阶信息 (一阶导数)，也即梯度算法时，优化方向和步长的选取就变得较为关键。进一步，如果添加流形约束条件，如 $X \in \mathcal{M}$ 为 $Stiefel$ 流形，满足 $X^TX = I$. 则我们可以将它视为在欧式空间中的嵌入，函数的梯度不能直接返回。因此在迭代过程中，在沿梯度方向搜索后还必须添加拉回操作，如投影算子等，以保证每次选取的可行解落在流形中。在计算实例中，拉回算法往往是昂贵的，因此催生了不少替代方法简化复杂度。另一方面，诸如 $Armijo$ 条件和 $Wolfe$ 条件等提法都是对步长选择的进一步优化，使得算法有更好的收敛性。又如 BB 步长算法，往往搭配非单调下降搜索，交替选取最大与最小允许步长，进行优化搜索。

1.1 文再文老师的相关算法分析

文再文老师在一篇 2012 年发表的文章中，提供了一种基于 $Cayley$ 变换的替代拉回方法。首先介绍一下 $Cayley$ 变换：

$$Y = (I + \frac{a}{2}A)^{-1}(I - \frac{a}{2}A)X.$$

这里 A 是某反对称矩阵即 $A^T + A = 0$. a 是实数。易见，当 $X \in \mathcal{M}$ 有 $Y \in \mathcal{M}$. 从而在此变换下，可行解始终在 $Stiefel$ 流形上，可以理解为是替代拉回。进一步计算可知：

$$Y'(a) = -(I + \frac{a}{2}A)^{-1}A(\frac{X + Y(a)}{2}), Y'(0) = -AX.$$

于是在 X 处由 $Cayley$ 变换诱导的流形梯度方向就是 $-A$. 在文再文老师的代码中，他选取了 $GX^T - XG^T$ 作为乘子 A(一种常见且自然的选择)，这导致实际搜索方向并不是欧式梯度的投影 (那样的话我们应该选择计算更复杂的 $A = (PG)X^T - X(PG)^T$) 这里 P 是投影算子，G 是欧式梯度向量。在其他采用 $Cayley$ 变换的优化算法中，也有采取其他反对称乘子的。下一个问题是，如何快速地进行 $Cayley$ 变换。注意到 Y 的直接表达式中包含求逆运算，这贡献了主要运算量且使算法更不稳定。因此文再文老师在他的引理 4 中证明了，可以基于 SMW 方法降低计算量 (尤其是低秩情况) 并进行了复杂度估计，但仍需要进行求逆运算。在其他文献中，也有使用替代式 $Y_{n+1}(a) = X - \frac{a}{2}A(X + Y_n(a))$ 进行迭代逼近真实 $Y(a)$ 而绕开求逆运算的，这里不多做展开。

文再文老师使用 $Cayley$ 变换方法和带更新的 BB 步长算法，对一系列计算实例进行了一阶优化，呈现如下。

1.2 算法实现与比较

第一个问题是 $maxcutSDP$ 问题，优化目标是最大化 $trace(C*X)$. 这里 C 是 $n*n$ 对称正定矩阵，X 是低秩的 $Stiefel$ 流形上的点，可以表示为 $V'*V$，这里 V 是 $p*n$ 矩阵，$p<n, ||V_i|| = 1$. 因此可视为对 V 进行最优化搜索。测试数据中 $n = 512, p = 16$. 此代码的运行时间为 $1s$ 量级，能够将可行解处的梯度 (流形梯度) 优化至 $1e-5$ 量级。

第二个问题是 $eigenrand$ 问题，优化目标是最小化 $-0.5*trace(X'*A*X)$.，可以证明它会收敛到 A 的主特征值上去 (或者理解为谱半径)。这里 A 是对称正定矩阵，X 是 $n*p$ 矩阵。将文再文老师的代码与 $matlab$ 自带的求特征值函数 $ARPACK$ 算法进行比较，呈现如下。可以看到，随着矩阵规模的扩大，文再文老师的算法计算时间优势就越发明显。

测试编号	两解相对误差	流形误差	自带用时	测试用时
1	6.46e-08	5.66e-16	0.455561	0.219965
2	1.66e-07	7.31e-16	0.373647	0.243345
3	9.44e-07	6.21e-16	0.416554	0.324157
4	1.91e-07	5.48e-16	0.366978	0.257050
5	5.82e-07	7.50e-16	0.366513	0.175239

矩阵规模为 2000 时测试数据

测试编号	两解相对误差	流形误差	自带用时	测试用时
1	1.51e-07	6.18e-16	0.935093	0.464216
2	1.95e-06	8.11e-16	0.952295	0.548911
3	1.85e-07	5.70e-16	0.922943	0.555478
4	6.33e-08	8.43e-16	0.870333	0.387242
5	1.83e-06	8.30e-16	0.966355	0.522415

矩阵规模为 3000 时测试数据

测试编号	两解相对误差	流形误差	自带用时	测试用时
1	9.05e-08	9.21e-16	2.002959	0.607977
2	5.88e-07	7.17e-16	1.818578	0.627859
3	7.07e-07	5.60e-16	1.767710	0.893577
4	5.85e-06	6.16e-16	1.848207	0.830183
5	1.66e-06	9.96e-16	1.725045	0.543994

矩阵规模为 4000 时测试数据

我自己写了一个使用 BB 步的流形优化算法，下面简称算法 2，将文再文老师的对应算法称为算法 1。我使用了直接投影拉回而没有使用 $Cayley$ 变换，从理论分析来说，这会导致更大的计算量。我对 $maxcut$ 问题进行了测试，并与文再文老师的代码进行比较，呈现下面表格中。

测试编号	算法 1 优化目标值	算法 2 优化目标值
1	-4.573582e+02	-4.573577e+02
2	-4.573582e+02	-4.573581e+02
3	-4.573582e+02	-4.573567e+02
4	-4.573582e+02	-4.573574e+02
5	-4.573582e+02	-4.573569e+02
6	-4.573582e+02	-4.573580e+02
7	-4.573582e+02	-4.563574e+02
8	-4.573582e+02	-4.573580e+02
9	-4.573582e+02	-4.573490e+02
10	-4.573582e+02	-4.573462e+02

两种算法的优化结果比较

测试编号	算法 1 流形梯度	算法 2 流形梯度
1	9.9e-06	7.9e-03
2	9.7e-06	4.1e-02
3	6.3e-06	2.6e-02
4	9.5e-06	3.4e-02
5	7.2e-06	1.1e-03
6	6.1e-06	5.1e-03
7	9.7e-06	1.9e+00
8	9.7e-06	9.3e-02
9	8.4e-06	9.6e-02
10	9.5e-06	1.8e-03

两种算法解处的流形梯度比较

测试编号	1 用时	2 用时	1 循环数	2 循环数
1	0.95	2.14	587	1008
2	1.04	1.76	702	832
3	0.72	1.81	497	866
4	0.83	0.88	551	432
5	0.99	1.48	674	698
6	0.94	1.67	673	778
7	0.96	0.16	740	66
8	1.00	2.98	717	1450
9	0.86	0.88	618	386
10	0.80	1.82	572	880

两种算法求解时间与循环数比较

从比较结果中不难发现，从优化效果上来说，文再文老师的代码更精确且更稳定。我的算法只能得到一个近似结果，相对误差约在 $1e-6$ 量级，尚在可接受范围内。从可行解的流形梯度来看，我的算法也明显更不稳定，求解效果较差，并且还存在提前停止求解的情况 (7 号测试结果明显是错误的)。在正确求解的数据点上，文再文老师的算法速度也始终更优，验证了我们理论上分析的算法复杂度差距。不过，当输入的正定矩阵 C 是随机生成，且维数 n 较大时 (测试中我选择了 500，1000，2000)，两种算法的表现都不太理想，部分结果呈现如下。

测试编号	算法 1 流形梯度	算法 2 流形梯度
1	5.0e-03	5.4e-03
2	5.8e-02	1.2e-01
3	2.9e-02	4.2e-02
4	4.2e-03	1.5e-02

n=500 两种算法解处的流形梯度比较

测试编号	1 用时	2 用时	1 循环数	2 循环数
1	1.87	2.99	1369	1422
2	2.42	4.08	2000	2000
3	2.44	4.03	2000	2000
4	1.27	4.00	1007	2000

n=500 两种算法求解时间与循环数比较

测试编号	算法 1 流形梯度	算法 2 流形梯度
1	1.3e-02	5.0e-01
2	1.8e-02	1.1e-02
3	9.5e-03	3.7e-01
4	9.3e-03	6.1e-02

n=1000 两种算法解处的流形梯度比较

测试编号	1 用时	2 用时	1 循环数	2 循环数
1	9.18	15.05	3164	4000
2	11.54	13.86	4000	3620
3	6.06	11.60	1842	4000
4	7.08	17.97	2234	4000

n=1000 两种算法求解时间与循环数比较

可以看出对随机生成的矩阵 C，两个算法都需要较长的收敛时间，停机时间和 n 呈现平方正相关。并且解的准确性也都大幅下降。这主要是因为一阶算法使用的梯度信息过于粗糙，而拉回算子在迭代过程中持续扩大每次的误差，使得结果不理想。接下来我们讨论使用二阶信息的算法。

2 二阶方法

顾名思义，二阶算法不仅使用优化目标函数的梯度信息，还使用二阶导数信息 (高维时也即海森矩阵)。因此，如果一阶方法是把目标函数局部泰勒展开到一阶的话，二阶方法就添加了二阶信息，用二次函数模拟优化目标的局部表现。在这种近似下，我们能得到经典牛顿方程和牛顿方向，并且能够证明，在目标函数足够光滑和满足李氏条件时，算法有二次收敛性质。

$$X_{k+1} = X_k - \alpha Hess_{x_k}^{-1} grad(f(X_k))$$

在理论上只要海森矩阵始终正定，迭代就能继续进行。然而在处理实际问题时，它的某些特征值可能非常接近 0，这导致更加不稳定的运算和过大的迭代差距，导致可行解序列并不收敛。因此在海森矩阵正定性较差时，相应步长应该选择得足够小。然而，另一方面经典牛顿法只有局部收敛性，较小的步长同样导致算法收敛时间过长。此外，储存海森矩阵的代价是巨大的，尤其是 n 较大的情况下，更不要说取逆运算了。于是，一种优化想法是，在海森矩阵正定性较差时，考虑以 $Hess_{(X_k)} + E$ 为相应矩阵的更新方向进行搜索。直观地说，我们对海森矩阵进行了一个小扰动，使得其正定性提升，计算更稳定。这里自然要求：1、扰动不能太大，否则新方向与实际下降方向偏差较大。2、新矩阵计算代价更小，才有优化产生。3、新矩阵的条件数也较低，否则无法保证算法收敛性。这种方法称为修正牛顿法。经典牛顿法和修正牛顿法都涉及矩阵求逆，因此也都经常结合优化的求逆算法，如矩阵的 SVD 分解或 $Cholesky$ 分解等加速运算。然而，它们都需要处处计算并储存海森矩阵，这比起一阶情况下储存梯度

信息昂贵了 n 倍，计算量也相应增长，因此成为算法的主要复杂度贡献项。因此另一种思路是，尝试每次仅储存海森矩阵和搜索方向的乘积向量，而非矩阵。这里简要介绍一下需要用到的共轭梯度方法。

回顾一阶方法，优化目标函数为正定的二次函数，且其表示矩阵是纯对角阵。则此时其几何解释是一个以原点为最低点的，开口向上的多元二次函数。按之前讨论的一阶方法，可以从任意点沿梯度方向下降到原点，但由于矩阵的特征值不唯一，梯度方向并不处处指向原点，因此在下降过程中存在折返的情况。对于一般的对称正定矩阵，我们可以对其做 SVD 分解，得到 $A = UZV$ 其中 U, V 都是酉矩阵，Z 是纯对角阵，也可以在差一个坐标变换的意义下做类似分析。总之，沿梯度方向线搜索虽然是局部最优的，但在全局上可能面临下降不够快速的问题。

因此，引入共轭方向的概念。我们在线搜索时，希望按各正交方向分别进行，防止出现折返情况。其次，由于前述的坐标变换的存在，此时的正交不再是欧式意义的正交，而是共轭，即 $d_{k+1}^T Q d_k = 0$，Q 是坐标变换矩阵，当它是恒等矩阵时共轭即正交，而海森矩阵也可以视为此处的坐标变换。通过进一步分析可知，对于一般的对称正定矩阵 $A = Z^2, Z = P^{-1}\Lambda P.$ 这里 P 为正交矩阵，Λ 元素为 A 各特征值的平方根。则 $U = P^{-1}\frac{1}{\Lambda}P$ 的各行 (或列) 向量就是彼此共轭的一组基，我们也即可以分别沿其进行标准的线搜索，而不会产生折返。

理论分析虽然给我们提供了一组共轭方向，但并不适合实际计算。共轭梯度法是算法中典型的寻找共轭方向进行搜索的办法。设搜索方向满足 $d_{k+1} = -g_k + \beta_k d_k$，将海森矩阵视为此处坐标变换，即可得到 $Hestenes - Stiefel$ 公式：

$$\beta_k = \frac{grad_{k+1}^T Hess_{x_k} d_k}{d_k^T H d_k}.$$

类似的也有不同的等价计算 β_k 的公式，如 $Fletcher - Reeves$ 公式：

$$\beta_k = \frac{grad_{k+1}^T grad_{k+1}}{grad_k^T grad_k}.$$

和 $Polk - Ribiere - Polyak$ 公式：

$$\beta_k = \frac{grad_{k+1}^T (grad_{k+1} - grad_k)}{grad_k^T grad_k}.$$

回到之前的优化牛顿方法。为了避免储存和计算海森矩阵，我们自然应该选取不带 $Hess$ 的计算公式，在算法中我们采用了 FR 公式版本的共轭梯度方法。注意到上述公式的等价性在二次函数时成立，因此在优化实际问题中，其表现也会不同。

进一步，我们也需要处理海森矩阵正定性较差的问题，此时引入带正则化项的子问题：

$$m_k(x) = grad_{x_k}^T(x - x') + \frac{1}{2}(x - x')Hess_{x_k}(x - x') + \sigma||x - x'||^2.$$

直观解释是，m_k 表示我们使用展开到二阶项的目标函数近似后，加入了正则项作为惩罚 (因此 σ 非负)，从而在优化的同时，控制步长不会太大。另一方面，对 m_k 求导后易见，其海森矩阵在原函数的对应海森矩阵基础上，添加了一个正对角阵，按修正牛顿法的解释，也就避免了海森矩阵正定性不足的问题，保证算法收敛性。

2.1 文再文老师的相关算法分析

文再文老师在上述分析的基础上，改进了正则化项的选取。注意到，σ 值越大，优化时对大步长的惩罚就越大，使得迭代更加保守。反之，迭代则更加激进。文再文老师的自适应正则化即使用了这一性质，当预测下降量 m_k 和实际下降量 $f(x') - f(x)$ 相当贴近时，表明近似效果较好，可以进行激进的迭代，缩小 σ；反之则说明近似较差，需要精细搜索，增大 σ。此外文再文老师的代码中还对共轭梯度进行了截断操作确保沿下降方向迭代，并且在使用二阶方法前先用一阶方法 (BB 步长线搜索等) 得到近似解。

2.2 算法实现与比较

第一个运算实例是低秩矩阵近似问题，即最小化 $H.*(X - G)$ 的 Frobenius 范数，这里 X 是低秩的，同样可以写成 $V' * V$. 部分运行结果如下 可以看到一

矩阵规模	一阶方法结果	二阶方法结果	一阶流形梯度	二阶流形梯度
500	1.23352890e+02	1.23352890e+02	1.52e-05	9.64e-07
1000	1.46689496e+03	1.46689496e+03	8.12e-05	6.60e-06
2000	1.21908918e+04	1.21908918e+04	1.09e-04	1.05e-05

低秩逼近结果

阶方法虽然已经能得到足够近似的优化目标，但是配合二阶方法可以得到更准确的解的位置，拥有更接近零的流形梯度。

第二个运算实例是矩阵信息补全，即最小化 $P.*X-A$ 的 Frobenius 范数，这里的 X 同样低秩，和前一个问题较为类似。第三个测试问题是 DFT 背景下的求谱半径问题的变体，优化函数形式较复杂，不做展开，只呈现结果

矩阵规模	一阶方法结果	二阶方法结果	一阶流形梯度	二阶流形梯度
1000	1.68950883e-10	3.06856858e-11	2.68e-06	9.40e-07
2000	1.94460033e-09	2.62081542e-11	5.99e-06	6.03e-07
4000	2.25345292e-09	1.41705605e-10	3.20e-06	8.03e-07

低秩补全结果

矩阵规模	一阶方法结果	二阶方法结果	一阶流形梯度	二阶流形梯度
2000	2.75967660e+03	2.75967660e+03	3.71e-05	1.09e-06
3000	2.75967660e+03	2.75967660e+03	1.11e-04	5.83e-06
5000	2.75967660e+03	2.75967660e+03	3.08e-05	9.07e-07

低秩补全结果

在此基础上我们来看自适应方法对求解速度的影响。针对第一个测试问题，我们对比有自适应调节系数的正则化项，不调节系数的正则化项和无正则化项的算法，依次命名为算法 1、2、3。将矩阵规模和秩大小固定为 $n=1000, p=100$.

可以看到在此算例中正则化和自适应的正则化对算法有优化但程度都不显著，这可能是因为上述算例都已经能够被二次函数较好地逼近，且相关的海森矩阵性质较好，因此正则化项带来的优化不大。

1 用时	2 用时	3 用时	1 梯度	2 梯度	3 梯度
17.43	17.71	17.87	9.5e-07	8.7e-07	8.7e-07
17.30	17.75	18.21	9.5e-07	8.7e-07	8.7e-07
17.47	17.74	17.69	9.5e-07	8.7e-07	8.7e-07

自适应项比较

最后，我按照课程 ppt 的流程写了一个自适应正则化的牛顿算法，测试问题就取为最小化 $\frac{1}{2}X' * A * X + B' * X$ 多元的二次函数优化问题。测试结果如下 这里目标函数中的矩阵是随机生成的对称正定矩阵。由于我没有先用一阶方

测试编号	算法用时	求解处流形梯度	循环数
1	0.71	9.9e-06	1206
2	0.99	9.7e-06	1235
3	0.93	9.2e-06	1194
4	1.11	1.0e-05	1202

测试算法求解二次函数优化

法近似到解附近，因此有时算法停解时的流形梯度并没有得到优化，换言之算法不收敛，这和理论相符。最后附 matlab 代码，分别是我自己写的 BB 步长求解 maxcut 问题，修改后的文再文老师的 BB 步长解同一问题，和我自己写的用正则化牛顿法求解二次函数问题。

```
clear;
src = [fileparts(mfilename('fullpath')) '/data/Maxcut/'];
file = strcat(src,'torusg3-82','.mat');
load(file,'n','m','C');
tic;

%n = , , 50010001500;
p = max(min(round(sqrt(2*n)/2), 20),1);
%C = randn(n, n);
%C = C' * C;
x0 = randn(p,n);
m0 = sqrt(dot(x0,x0,1));
x0 = bsxfun(@rdivide, x0, m0);
x = x0;
g = 2*(x*C);
f = sum(dot(g,x))/2;
%    xtg = dot(x,g,1);    gg = dot(g,g,1);
%    xx = dot(x,x,1);     xxgg = xx.*gg;
%  dtX = bsxfun(@times, xtg, x) - bsxfun(@times, xx, g);
% nrmG = norm(dtX, 'fro');
nrmG = norm(g, 'fro');
nrmx = dot(x,x,1);
Q = 1; Cval = f; tau = 1e-3;
gamma = 0.85;
eps = 1e-14;
for iter = 1 : 2000
xp = x;      fp = f;      gp = g;    %dtXP =  dtX;
xx = dot(x,x,1);
xtg = dot(x,g,1);
pull = bsxfun(@times, xtg, x) - bsxfun(@times, xx, g);
% g1 = gp - 0.5*xp*(xp'*gp+gp'*xp);
nls = 1; deriv = (1e-4)*nrmG^2;
while 1
x = xp + tau*pull;
g = 2*(x*C);
```

```
f = sum(dot(g,x))/2;

if f < Cval - tau*deriv || nls >= 5
break
end
tau = 0.1*tau;
nls = nls+1;
end
xx = dot(x,x,1);
feasi = compute_feasi(xx);

if feasi > eps
nrmx = dot(x,x,1);
x = bsxfun(@rdivide,x,sqrt(nrmx));

g = 2*(x*C);
f = sum(dot(g,x))/2;

end
%      xtg = dot(x,g,1);    gg = dot(g,g,1);
%     xx = dot(x,x,1);      xxgg = xx.*gg;
%   dtX = bsxfun(@times, xtg, x) - bsxfun(@times, xx, g);
xx = dot(x,x,1);
xtg = dot(x,g,1);
pull = bsxfun(@times, xtg, x) - bsxfun(@times, xx, g);
nrmG = norm(pull, 'fro');

s = x - xp;
ns = norm(s);
y = g - gp;
ny = norm(y);
if nrmG < 1e-5 || ns < 1e-6
break;
end
sy = abs(iprod(s,y));      tau = 1e-3;
```

```
if sy > 0
if mod(iter,2)==0
tau = (norm(s(:),'fro')^2)/sy;
else
tau = sy/(norm(y(:),'fro')^2);
end
tau = max(min(tau, 1e20), 1e-20);
end

Qp = Q; Q = gamma*Qp + 1; Cval = (gamma*Qp*Cval + f)/Q;
end
tsolve = toc;
fprintf(' f: %8.6e, nrmG: %2.1e, cpu: %4.2f, OutIter: %3d', ...
f, nrmG, tsolve, iter);
function feasi = compute_feasi(xx)
nrmx = sqrt(xx);
feasi = norm(nrmx -1);
end
function a = iprod(x,y)
a = real(sum(sum(conj(x).*y)));
end
```

```
clear;
src = [fileparts(mfilename('fullpath')) '/data/Maxcut/'];
file = strcat(src,'torusg3-82','.mat');
load(file,'n','m','C');
%n = , , 50010001500;
p = max(min(round(sqrt(2*n)/2), 20),1);
%C = randn(n, n);
%C = C' * C;

x0 = randn(p,n);
m0 = sqrt(dot(x0,x0,1));
x = bsxfun(@rdivide, x0, m0);
g = 2*(x*C);
f = sum(dot(g,x))/2;

xtg = dot(x,g,1);    gg = dot(g,g,1);
xx = dot(x,x,1);     xxgg = xx.*gg;
dtX = bsxfun(@times, xtg, x) - bsxfun(@times, xx, g);
nrmG = norm(dtX, 'fro');

Q = 1; Cval = f; tau = 1e-3;
gamma = 0.85;
eps = 1e-14;
tic;
for iter = 1 : 2000
xp = x;      fp = f;      gp = g;    dtXP = dtX;

nls = 1; deriv = (1e-4)*nrmG^2;
while 1
tau2 = tau/2;      beta = (1+(tau2)^2*(-xtg.^2+xxgg));
a1 = ((1+tau2*xtg).^2 -(tau2)^2*xxgg)./beta;
a2 = -tau*xx./beta;
x = bsxfun(@times, a1, xp) + bsxfun(@times, a2, gp);
```

```
g = 2*(x*C);
f = sum(dot(g,x))/2;
if f < Cval - tau*deriv || nls >= 5
break
end
tau = 0.1*tau;
nls = nls+1;
end
xx = dot(x,x,1);
feasi = compute_feasi(xx);

if feasi > eps
nrmx = dot(x,x,1);
x = bsxfun(@rdivide,x,sqrt(nrmx));
g = 2*(x*C);
f = sum(dot(g,x))/2;

end
xtg = dot(x,g,1);    gg = dot(g,g,1);
xx = dot(x,x,1);     xxgg = xx.*gg;
dtX = bsxfun(@times, xtg, x) - bsxfun(@times, xx, g);

nrmG = norm(dtX, 'fro');
s = x - xp;
if nrmG < 1e-5  || norm(s) < 1e-6
break;
end
y = dtX - dtXP;
sy = abs(iprod(s,y));     tau = 1e-3;
if sy > 0
if mod(iter,2)==0
tau = (norm(s(:),'fro')^2)/sy;
else
tau = sy/(norm(y(:),'fro')^2);
end
```

```
tau = max(min(tau, 1e20), 1e-20);
end

Qp = Q; Q = gamma*Qp + 1; Cval = (gamma*Qp*Cval + f)/Q;
end
tsolve = toc;
fprintf(' f: %8.6e, nrmG: %2.1e, cpu: %4.2f, OutIter: %3d', ...
f, nrmG, tsolve, iter);
function feasi = compute_feasi(xx)
nrmx = sqrt(xx);
feasi = norm(nrmx -1);
end
function a = iprod(x,y)
a = real(sum(sum(conj(x).*y)));
end
```

```
function [x, out] = newCG(x0, fun, HXP, A, B)
x = x0;
sigma = 10;
gamma0 = 0.2;
gamma1 = 2.5;
c1 = 0.1;
c2 = 0.9;
gtol = 1e-05;
iter = 0;
maxiter = 2000;
while 1
[fval, rg, eg] = fun(x, A, B);
[xfeasi, mx] = CG(x, rg, eg, HXP, sigma, A, B);
[f_test, ~] = fun(xfeasi, A, B);
rho = (f_test - fval) / mx;
if rho >= c1
x = xfeasi;
end
if      rho < c1
sigma = gamma1 * sigma;
elseif rho > c2
sigma =  max(gamma0 * sigma, 1e-10);
end
iter = iter +1;
if norm(rg) < gtol || iter > maxiter
break;
end
end

out.iter = iter;
out.fval = fval;
out.nrmG = norm(rg, 'fro');
end

function [x_test, mx] = CG(x, rg, eg, HXP, sigma, A, B)
```

```
ctol = 1e-01;
[l, c] = size(x);
M = stiefelfactory(l, c);
eta = zeros(l, c);d = eta;
r0 = rg;
r = r0;P = -r;
sitr = 1;
alpha0 = 1e-02;
epsilon = 1e-01;
c = 1e-01;
nrmr0 = trace(r0'*r0);
T = 1e-01;
while sitr < l
Hessp = HXP(x, P, A, B) + sigma * eye(l) * P;
pi = trace(P'*Hessp);
nopi = pi / trace(P'* P);
if nopi < ctol
if sitr == 1
s = -P;
else
s = eta;
end
if nopi < -ctol
d = P;
end
break
end
nr = trace(r'*r);
alpha = nr / pi;
eta = eta + alpha * P;

rfeasi = r + alpha * Hessp;
nrfr = trace(rfeasi'*rfeasi);
if nrfr < min(nrmr0, T)
s = eta;
```

```
break
end
beta = nrfr / nr;
P = -rfeasi + beta * P;
r = rfeasi;
sitr = sitr + 1;
end
if norm(d, 'fro') == 0
dir = s;
else
tau = trace(d'*rg) /trace(d'*Hessp);
dir = s + tau * d;
end
alpha = alpha0;
nls = 0;
while 1
[mxf, ~] = qr(x + alpha * dir, 0);
%mxf = M.proj(x,alpha * dir)+x;
dx = mxf - x;
mx = eg' * dx + dx' * A' * dx + 0.5 * sigma * (dx' * dx);
if mx < c * alpha * trace(rg'*dir)
break
end
alpha = alpha * epsilon;
nls = nls +1;
if nls > 25
break
end
end
x_test = mxf;
end
```

```
clear
n = 50;
p = 1;
A = randn(n, n);
A = A' * A;
B = randn(n, p);
x0 = randn(n, p);
m0 = sqrt(dot(x0,x0,1));
x0 = bsxfun(@rdivide, x0, m0);
tic; [~, out_MINE] = newCG(x0, @fun, @HXP, A, B); tsolve= toc;

fprintf('f: %8.6e, nrmG: %2.1e, cpu: %4.2f, OutIter: %3d', ...
out_MINE.fval, out_MINE.nrmG, tsolve, out_MINE.iter);


function [f, rg, eg] = fun(X, A, B)
f = trace(X' * A * X + 2 * X' * B);
[l, c] = size(X);
M = stiefelfactory(l, c);
eg = 2 * A * X + 2 * B;
rg = M.proj(X,eg);
end

function H = HXP(X, P, A, B)
[l, c] = size(X);
M = stiefelfactory(l, c);
eg = 2 * A * X + 2 * B;
ehess = A * P;
H = M.ehess2rhess(X, eg, ehess, P);
end
```