

流形优化期末报告

张晨

2024 年 9 月 23 日

1 Question 1

2 Question 2

编程实现教材上的 Algorithm 3 和 Algorithm 4, 随机生成 Stiefel 流形的一个二次函数, 用 Algorithm 4 极小化这个二次函数, 比较两个算法的计算效果, 探索非单调的作用, 或者交替使用 BB 步长两个公式的作用.

2.1 随机二次函数的生成

首先考虑实现二次函数的随机生成问题. 对于整数 $n \geq p > 0$, 欧氏空间 $\mathbb{R}^{n \times p}$ 上的 *Stiefel* 流形为

$$\text{St}(n, p) = \{X \in \mathbb{R}^{n \times p} | X^T X = I_p\}.$$

对于矩阵 $B \in \mathbb{R}^{n \times p}$ 和对称正定矩阵 $A \in \mathbb{R}^{n \times n}$, $\text{St}(n, p)$ 上的二次函数定义为

$$f(X; A, B) = \text{tr}(X^T A X + 2X^T B).$$

故二次函数 f 的生成问题等价于矩阵 A, B 的生成问题. 在 MATLAB 中, 可以使用 **randn** 函数随机生成矩阵, 因此, 可以取

$$B \leftarrow \text{randn}(n, p).$$

但 MATLAB 中并没有直接生成对称正定矩阵的函数. 为了随机生成 A , 先考虑一个简单的引理:

引理 2.1 记集合 \mathcal{A} 为对称正定矩阵全体, 定义集合

$$\mathcal{B} = \{R^T R + D | R \in \mathbb{R}^{n \times n}, D = \text{diag}(d_1, \dots, d_n), 0 < d_1, \dots, d_n < 1\},$$

则 $\mathcal{A} = \mathcal{B}$.

证明: 首先证明 $\mathcal{B} \subseteq \mathcal{A}$. 对于任意的 $B = R^T R + D \in \mathcal{B}$, 有

$$B^T = (R^T R + D)^T = R^T R + D^T = R^T R + D = B,$$

故 B 是对称的. 对于任意的 $x \in \mathbb{R}_*^n$, 有

$$x^T Bx = (Rx)^T (Rx) + x^T Dx \geq x^T Dx \geq d_1 \|x\|_2^2 > 0.$$

因此, $B \in \mathcal{A}$.

再证明 $\mathcal{A} \subseteq \mathcal{B}$. 对于任意的 $A \in \mathcal{A}$, 记 $\lambda > 0$ 为 A 的最小特征值, 取 $d = \min(\lambda/2, 1/2)$, 则矩阵 $A - dI_n$ 仍为对称正定矩阵. 注意到, 根据对称正定矩阵的 Cholesky 分解, 存在矩阵 $R \in \mathbb{R}^{n \times n}$, 使得

$$A = (A - dI_n) + dI_n = R^T R + dI_n \in \mathcal{B}.$$

综上, $\mathcal{A} = \mathcal{B}$. ■

根据这个引理, 结合 MATLAB 中取值在 $(0, 1)$ 上的随机函数 **rand**, 可以得到对称正定矩阵的随机生成方法:

$$\begin{aligned} R &\leftarrow \text{randn}(n, n), \\ A &\leftarrow R^T R + \text{diag}(\text{rand}(n)). \end{aligned}$$

详细的代码实现可以参考附录. 在下文中, 我们考虑 $n = 500$, $p = 50$ 时各算法的表现, 取 MATLAB 中随机函数的种子为 99, 生成的矩阵对 (A, B) 同样可见附录.

2.2 梯度下降算法与 BB 算法的比较

本小节中, 我们使用基于单调线搜索的梯度下降算法与基于非单调线搜索的 BB 算法对二次函数 $f(X) = \text{tr}(X^T A X + 2X^T B)$ 进行优化. 在参数选择上, 我们使用表 1 中的参数值.

参数	具体数值	参数描述
t_0	0.02	回退法的初始步长
ρ	0.5	回退幅度
c	0.001	Armijo 条件的系数
M	5	非单调线搜索的前项个数
α_{\max}	100	BB 步长的上界
α_{\min}	0.01	BB 步长的上界
method	QR 分解	收缩映射算法

表 1: 算法参数

我们先分析一下两个算法的收敛速度, 随机选取初始点 X_0 , 观察函数值 $f(X_k)$ 与点列 X_k 的误差变化情况, 结果见图 1, 其中左图为函数值 $f(X_k)$ 的相对误差随迭代次数的变化情况, 右图为自变量点列 X_k 关于极小值点 X^* 的相对误差, 其度量为两点作为 $\mathbb{R}^{n \times p}$ 矩阵时, 差值 $X_k - X^*$ 的 Frobenius 范数.

从图 1 中可以看出: 首先, 两个算法都在 50 步内迭代到了最小值点, 误差达到了机器精度; 其次, 这两个算法的收敛速度都是线性收敛, 且 BB 算法的速度要显著快于梯度下降算法; 最后, 函数值的收敛速度比自变量点列的收敛速度更快, 这一点是由二次函数的性质产生的.

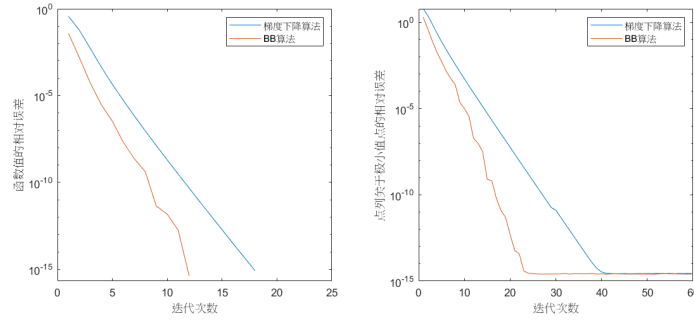


图 1: 相对误差的变化情况

接下来, 对于不同的收缩映射算法, 我们对比一下算法的收敛速度, 结果见图 2, 其中左图均为函数值的相对误差变化情况, 右图均为自变量点列的相对误差变化情况, 第一行为使用梯度下降算法得到的误差变化图, 第二行为使用 BB 算法得到的误差变化图.

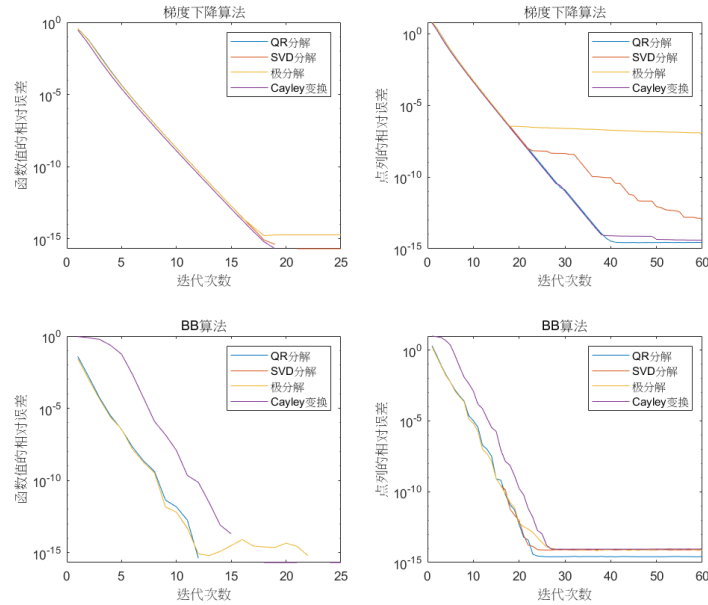


图 2: 不同收缩映射算法的表现

从图 2 中可以看到, 不同收缩映射算法在函数值上的表现是相同的, 以同样的速度线性收敛. 值得一提的是, 基于 Cayley 变换的收缩映射算法会导致 BB 算法在前几次迭代的效果略差, 但在之后的迭代中仍然获得了和其他收缩映射算法相同的收敛速度. 当考察自变量 X 的收敛情况时, 不同的收缩映射算法产生了不同的表现. 在 BB 算法中, 基于 Cayley 变换的收缩映射算法仍然在前几次迭代中表现略差, 不过在长期迭代中, 这些收缩映射的表现是类似的; 但在梯度下降算法中, 基于 QR 分解和 Cayley 变换的收缩映射算法很快到达了机器精度, 基于 SVD 分解的收缩映

射算法在 20 次迭代后放缓了收敛速度, 基于极分解的收缩映射算法只能达到 10^{-6} 精度, 无法达到机器精度.

值得特别一提的是, 在理论上, 基于 SVD 分解的收缩映射和基于极分解的收缩映射应当是同一个, 但这两种不同的收缩映射在梯度下降算法下的表现是不同的, 这可能是计算误差导致的: 在实现基于 SVD 分解的收缩映射时, 我们直接使用了 MATLAB 内置的 `svd` 函数, 这一函数是使用商业库 MKL 构建的, 因此会进行专业的优化, 达到比极分解更好的表现.

2.3 非单调线搜索的作用

本小节中, 我们通过使用不同的前项个数 M 对二次函数 $f(X)$ 进行优化, 探索非单调线搜索的作用. 在正式讨论前, 先观察一个现象: 取初始步长 $t_0 = 0.02$ 和 0.08 , 前项个数 $M = 5$, 其他参数同表 1, 查看单调线搜索与非单调线搜索的表现. 其结果见图 3, 其中左图为初始步长 $t_0 = 0.02$ 时的表现, 右图为初始步长 $t_0 = 0.08$ 时的表现, 图像为函数值的相对误差随迭代次数的变化情况, Armijo 条件指单调线搜索, Grippo 条件指 Grippo 提出的非单调线搜索, 凸组合条件指 H. Zhang 与 W. W. Hager 提出的非单调线搜索, 其系数 $\rho = 0.5$.

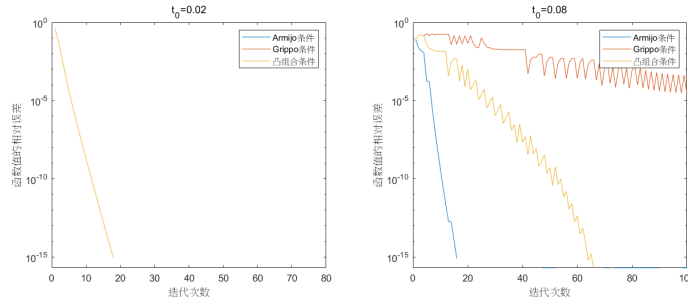


图 3: 步长对函数值误差的影响

从图 3 中可以看到, 当初始步长 $t_0 = 0.02$ 时, 单调线搜索与非单调线搜索都在很少的迭代步数中达到了非常低的误差, 但当初始步长 t_0 提高 4 倍, 变成 0.08 后, 单调线搜索的收敛速度没有降低, 但非单调线搜索的表现大大降低了, 凸组合线搜索在更多的迭代步数后才能达到同样的误差, 而 Grippo 线搜索的收敛速度变得非常缓慢. 一种可能的解释是, Grippo 条件比 Armijo 条件和凸组合条件更宽松, 这导致算法在接近最小值点、初始步长很大时容易越过最小值点而跳到更远的地方, 导致下降缓慢.

大步长造成的影响并非仅限于 Grippo 线搜索, 事实上, 大步长对每一个梯度下降算法都造成了影响. 为了说明这一点, 我们来查看不同初始步长下自变量 X 的收敛情况, 所有参数同上, 结果见图 4.

从图 4 中可以看到, 当初始步长 $t_0 = 0.02$ 时, 所有算法都在 40 步内迭代到了机器误差. 当初始步长变为 0.08 后, Grippo 线搜索不出所料没有收敛, 但单调线搜索与凸组合线搜索的误差也变大了, 凸组合线搜索的误差维持在了 10^{-10} , 而单调线搜索更是只能达到 10^{-8} . 为了更直观地体现步长对自变量收敛值的影响, 我们使用不同的步长进行实验, 结果见图 5, 其中 x 轴为初始步长, y 轴为迭代 1000 步后的自变量误差, 其余参数同上.

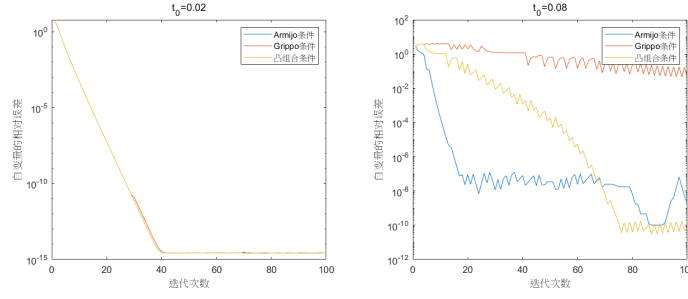


图 4: 不同步长下自变量误差的变化

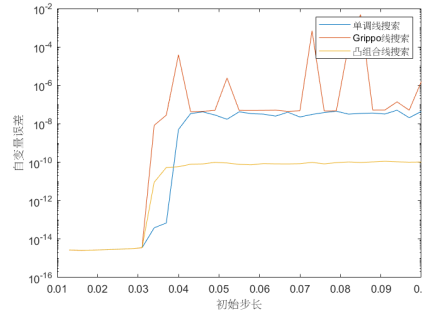


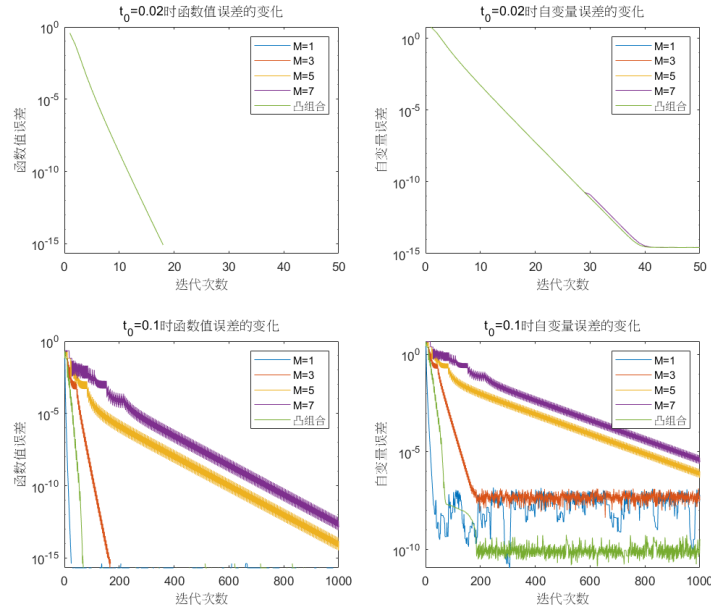
图 5: 步长对自变量误差的影响

从图 5 中可以看到, 当初始步长 $t_0 \leq 0.03$ 时, 三个算法都能达到机器精度, 但当初始步长 $t_0 > 0.03$ 时, 三个算法跳跃到了更大的误差上. 凸组合线搜索在大步长时, 自变量误差稳定在了 10^{-11} 上; 单调线搜索和 Grippo 线搜索只能达到 10^{-8} 量级的精度, 并且 Grippo 线搜索甚至在 1000 步后仍无法保证收敛, 误差有时会升到 10^{-3} . 产生这种情况的原因仍不清楚, 有待更详细的实验与分析.

现在我们正式考察不同参数 M 对梯度下降算法的影响, 取初始步长 $t_0 = 0.02, 0.1$, $M = 1, 3, 5, 7$, 其余参数同表 1, 结果见图 6.

从图 6 中可以看到, 当初始步长 $t_0 = 0.02$ 时, 不同的参数 M 带来的函数值变化和自变量变化是几乎完全一致的. 这可能是由于本问题中选取的矩阵 A 很好, 使得梯度下降法在很少的迭代次数后就达到了最小值点附近, 而初始步长又很小, 导致回退法并没有产生回退步骤, 进而让回退法的梯度下降法退化成了固定步长的梯度下降法, 因此这些算法并没有明显的区分. 当初始步长 $t_0 = 0.1$ 时, 不同的参数 M 产生了不同的效果:

- 随着 M 的增大, 梯度下降法的收敛速度变得越来越慢.
- 随着 M 的增大, 函数值误差的波动也越来越大.
- $M = 5$ 和 $M = 7$ 的收敛速度是一样的, 但 $M = 7$ 在算法前期更频繁地短暂停留在了某个点, 导致其需要花费更多的迭代次数才能达到合适的误差.
- 相比于单调线搜索, 非单调线搜索的自变量误差要更稳定一些.

图 6: 不同参数 M 下梯度下降法的表现

- 凸组合线搜索的收敛速度要略慢于单调线搜索, 但收敛后凸组合线搜索的自变量误差要小于单调线搜索的自变量误差.

对于第一点, 这有可能是因为本问题中的矩阵 A 性质很好, 因此单调线搜索就可以产生很好的效果, 而非单调线搜索反而因为更宽松的条件导致容易跑离最小值点; 对于第二点, 这是符合直观的, 因为单调线搜索可以保证函数值单调下降, 而非单调线搜索有可能使得函数值短暂上升, 并且 M 越大, 函数值的波动就越剧烈, 不过其总体上是下降的; 对于第三点, 这有可能说明在本问题中, M 的不断增大并不会导致算法的收敛速度不断变慢, 而是有可能会使收敛速度达到一个下界, 不会继续降低; 第四点和第五点的产生原因仍有待更进一步的分析.

2.4 交替使用 BB 步长的作用

本小节中, 我们探索一下交替使用 BB 步长的作用. 选取参数如下: $\alpha_{\max} = 1000$, $\alpha_{\min} = 0.001$, 初始步长 $\alpha_0 = 0.02$, 其余参数同表 1, 我们直接考察使用交替步长、仅使用短 BB 步长、仅使用长 BB 步长这三种情况下 BB 算法的表现情况, 结果见图 7, 其中左图为函数值 $f(X)$ 随迭代次数的相对误差变化, 中间为自变量 X 随迭代次数的相对误差变化, 右图为 BB 步长随迭代次数的相对误差变化.

从图 7中可以看出, 在本问题中, 选择不同的 BB 步长并不会带来很大不同, 三种算法无论是在函数值 $f(X)$ 还是在自变量 X 的误差变化上都非常相似. 这三种算法不同的是 BB 步长的表现, 当全程使用长 BB 步长时, 每一步的 BB 步长会略大于使用交替 BB 步长时的长 BB 步长, 同样, 全程使用短 BB 步长时, 每一步的 BB 步长会略小于使用交替 BB 步长时的短 BB 步长.

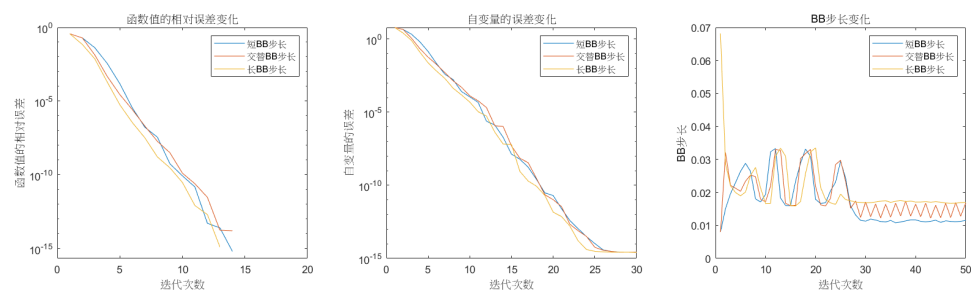


图 7: 不同 BB 步长的表现