数值最优化实验报告：多种线搜索算法的实现和比较

15336134 莫凡

2017 年 10 月 30 日

摘要

本文是中山大学 2015 级信息与计算科学专业的运筹学与最优化课程作业 [1]。内容为使用各种基于线搜索的牛顿方法实现对 Waston 函数 [2] 的优化。本文主要使用 MATLAB R2016b 实现并对比了非精确线搜索的 (Strong) Wolfe 准则、Wolfe 准则、Goldstein 准则与 armijo 准则，应用了最速下降法、阻尼牛顿法、LM 方法、拟牛顿法（SR1 公式、DFP 公式、BFGS 公式）测量了 n=6,9,12 的情形。

为了保证算法的正确性，本文同时根据 [2] 的建议，实现了两个小函数用来检验算法的正确性。通过实验，发现阻尼牛顿法、LM 方法、SR1 公式、BFGS 公式均能达到预期的收敛效果，而 DFP 公式和最速下降法没有在指定迭代次数内收敛。对比多种线搜索步长方法，使用强 Wolfe 准则是相对效果较为出色的方案。

目录

1 问题的分析与重述

1.1 环境与假设

本次试验主要在 Windows8.1 Home Basic 操作系统下，使用 MATLAB R2016b 完成。为了对于函数值与梯度进行检验，也同时用到了以下软件或程序包

- R 3.3.4
- Wolfram Mathematica 10

本文所涉及的所有代码与参考文献（如果能搜集到电子版）全部托管在 `https://github.com/w007878/watson`，以 MIT 协议发布

本文所完成的任务，是通过基于线搜索的迭代方法，求得一个函数在定义域上的数值极小值。由于函数的复杂性，我们无法验证此极小值是否为全局最小值，只能借助文献 [2] 中的参考值来检验结果的正确性。

为了评估一项算法，我们采用一下几个指标

1. 算法是否在规定迭代次数内收敛，如果是，需要迭代多少步
2. 算法停止后（包括满足梯度条件或者迭代次数条件终止）是否得到正确的极小值
3. 函数调用了多少次
4. MATLAB 的 cputime

1.2 Watson 函数的定义与计算

考虑以下定义在 n 维空间上的实值函数

$$f(x) = \sum_{i=1}^{m} r_i^2(x)$$

其中 n, m, r_i 可以是各种各样的取值。现要实现一个优化算法，求解这个函数的极小值。

本次试验的主要任务是实现

$$r_i(x) = \sum_{j=2}^{n} (j-1)\, x_j\, t_i^{j-2} - \left(\sum_{j=1}^{n} x_j\, t_i^{j-1} \right)^2 - 1$$

的情况。除此之外，还要有

$$m = 31 \quad 2 \le n \le 31$$

$$t_i = \frac{i}{29} \quad 1 \le i \le 29$$

$$r_{30}(x) = x_1 \quad r_{31}(x) = x_2 - x_1^2 - 1$$

这个函数被称为 Watson 函数 [3]，是我们要优化的目标。首先我们要得到的就是它的梯度与 Hessian 矩阵的表达式。事实上，我们没有必要写出 ∇f 关于自变量 x 的表达式，而是用 $r(x)$ 和它的偏导数进行代替

$$\frac{\partial f}{\partial x_k} = 2\sum_{i=1}^{m} r_i(x)\frac{\partial r_i}{\partial x_k}$$

刨除最后两项不优美的式子，我们有

$$\frac{\partial f}{\partial x_k} = \sum_{i=1}^{m} \frac{\partial f}{\partial r_i}\frac{\partial r_i}{\partial x_k}$$

其中

$$\frac{\partial r_i}{\partial x_k} = (k-1)\, t_i^{k-2} - 2\left(\sum_{j=1}^{n} x_j t_i^{j-1}\right) t_i^{k-1}$$

为了在代码中计算出这个东西，我们构造两个辅助矩阵。（为简单起见，用 m 代替 29）

$$T_1 = \begin{bmatrix} t_1^0 & t_1^1 & \cdots & t_1^{n-1} \\ t_2^0 & & \cdots & t_2^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ t_m^0 & \cdots & \cdots & t_m^{n-1} \end{bmatrix} \qquad T_2 = \begin{bmatrix} 0 & t_1^0 & t_1^1 & \cdots & t_1^{n-2} \\ 0 & t_2^0 & & \cdots & t_2^{n-2} \\ & \cdots & \cdots & \cdots & \cdots \\ 0 & t_m^0 & \cdots & \cdots & t_m^{n-2} \end{bmatrix}$$

得到 MATLAB 代码

```
tmp1 = 4 * T1' * (r(1:29) .* (T1 * x));
tmp2 = 2 * (0:n-1)' .* (T2' * r(1:29));
grad = tmp2 - tmp1;
```

然后根据 r_{30} 与 r_{31} 两项，单独处理一下 $\frac{\partial f}{\partial x_1}$ 与 $\frac{\partial f}{\partial x_2}$，梯度就计算完成了

接下来是 Hessian 矩阵。有表达式

$$\frac{\partial^2 f}{\partial x_b \partial x_a} = 2\sum_{i=1}^{m}\left(\frac{\partial r_i}{\partial x_b}\frac{\partial r_i}{\partial x_a} + \frac{\partial^2 r_i}{\partial x_b\, \partial x_a}\, r_i(x)\right)$$

这时我们采用最简单暴力的方法单独计算每一项偏导数以得到矩阵

```
prz = (0:n-1) .* T2 - 2 * (T1 * x) .* T1;
for a = 1:n
    for b = 1:n
        H(a, b)=2*prz(:,a)'*prz(:,b)+2*(-2*power(t,a+b-2))*r(1:29);
    end
end
```

代码中的 prz 表示的是每个 r 对每个 x 的偏导数。

通过对计算复杂度的分析，我们发现计算函数值、梯度和 Hessian 矩阵并没有很大的区别。同时，梯度和 Hessian 矩阵都需要用到相同的中间变量。而由于 n 很小，所以这点复杂度的差别基本忽略不计，于是使用一个函数同时计算出他的函数值、梯度与 Hessian 矩阵。

2 测试环节

2.1 三个测试函数

为了验证优化算法的正确性，在使用它优化 Watson 函数之前使用几个简单的函数进行测试。

这几个函数都具有 $f(x) = \sum_{i=1}^{m} r_i^2(x)$ 的形式。

2.1.1 Rosenbrock function

Rosenbrock function[4] 满足

$$n = 2 \quad m = 2$$
$$r_1(x) = 10(x_2 - x_1^2)$$
$$r_2(x) = 1 - x_1$$

若以点 $(-1.2, 1)$ 作为迭代起点，这个函数应当收敛到点 $(1, 1)$, $\quad f = 0$

2.1.2 Freudenstein-and-Roth function

Freudenstein-and-Roth function[5] 满足

$$n = 2 \quad m = 2$$
$$r_1(x) = -13 + x_1 + ((5 - x_2)x_2 - 2)x_2$$
$$r_2(x) = -29 + x_1 + ((x_2 + 1)x_2 - 14)x_2$$

若以点 $(0.5, -2)$ 作为迭代起点，可能会得到 1. 局部极小值 $(11.41\cdots, -0.8968\cdots)$, $\quad f = 48.9842 \quad$ 2. 全局最小值 $(5, 4)$, $\quad f = 0$

2.1.3 Powell-badly-scaled function

Powell-badly-scaled function[6] 满足

$$n = 2 \quad m = 2$$
$$r_1(x) = 10^4 x_1 x_2 - 1$$
$$r_2(x) = e^{-x_1} + e^{-x_2} - 1.0001$$

若以点 $(0, 1)$ 作为迭代起点，应当得到极小值 $(1.908\cdots 10^{-5}, 9.106\cdots)$, $\quad f = 0$

2.2 辅助软件检测

使用 R 语言检车 Watson 函数的数值，用它的 numDeriv 包检测 MATLAB 程序中计算的梯度和 Hessian 矩阵的数值。

使用 Mathematica 来检查以上三个测试函数的梯度与 Hessian 矩阵的解析形式是否正确。

3 线搜索程序

假设 d 是线搜索下降方向（Newton/quasi-Newton 方向），我们要求得正实数 α，使得 $f(x_k+\alpha d) \leq f(x_k)$，然后更新 $x_{k+1} = x_k + \alpha d$

线搜索的程序都比较简单。所以不需要进行额外的测验。实现之后可以直接用于后文中的优化算法。

3.1 定步长

基本牛顿方法直接设步长 $\alpha = 1$。如果采用这种方式，应当将下降方向向量正则化之后再应用。

3.2 精确线搜索

令

$$\alpha = \arg\min_{\alpha} f(x_k + \alpha d)$$

这里直接使用 MATLAB 内置的优化函数

3.3 非精确线搜索

我们使用最常规的“回溯法”求得非精确线搜索步长。伪代码如下

LINEAR-SEARCH(ρ, ϵ, α_0)

$\alpha = \alpha_0$
while not IS-SATISFY-CONDITION(α)
    $\alpha = \rho\alpha$
    if $\alpha < \epsilon$
        return α
return α

其中 ϵ 是用于限制步长不应小于某个数的阈值，以防算法收敛速度过慢。代码中使用了 10^{-5}。ρ 是用于更新 α 的值，在实践中取了 0.99 0.999 0.99999999，事实证明因为下降的速度是指数，所以没有太大关系

3.3.1 Arojio Condition

$$f(x_k + \alpha p_k) \leq f(x_k) + c\alpha p_k^T$$

3.3.2 Wolfe Condition

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1\alpha \nabla f_k^T p_k$$

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k$$

3.3.3 Strong Wolfe Condition

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1\alpha \nabla f_k^T p_k$$

$$|\nabla f(x_k + \alpha p_k)^T p_k| \leq c_2 |\nabla f_k^T p|$$

根据前人的经验 [7]，Arojio 准则中的 c、(Strong) Wolfe 准则中的 c_1 取值 10^{-4}，(Strong) Wolfe 准则中的 c_2 取值 0.9，可以得到更好的步长效果

3.3.4 Goldstein Condition

$$f(x_k) + (1-c)\nabla f_k^T p_k \alpha \leq f(x_k + \alpha p_k) \leq f(x_k) + c\nabla f_k^T p_k \alpha$$

根据 [8] 与 [9] 的建议，使用线搜索的 Newton 类方法较为常用的是满足 (Strong) Wolfe Condition 的非精确步长。

4 优化算法

4.1 阻尼牛顿法

牛顿方向是

$$d = -G^{-1}g$$

其中 G 是 Hessian 矩阵，g 是梯度

阻尼牛顿法是在牛顿方向上，通过先搜索得到合适的步长，然后进行迭代。迭代停止的条件为

5 实验与结果

6 后记

6.1 需要改进的实验内容

6.2 实验过程

参考文献

[1] 高立. 数值最优化方法, 2014.

[2] Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Softw.*, 7(1):17–41, March 1981.

[3] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

[4] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[5] Ferdinand Freudenstein and Bernhard Roth. Numerical solution of systems of nonlinear equations. *Journal of the ACM (JACM)*, 10(4):550–556, 1963.

[6] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*. Gordon and Breach, 1970.

[7] Jean Charles Gilbert and Claude Lemaréchal. Some numerical experiments with variable-storage quasi-newton algorithms. *Mathematical programming*, 45(1):407–435, 1989.

[8] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[9] Jianzhong Zhang and Chengxian Xu. Properties and numerical performance of quasi-newton methods with modified quasi-newton equations. *Journal of Computational and Applied Mathematics*, 137(2):269 – 278, 2001.