数值最优化作业笔记

snowyjone mo

2017 年 10 月 26 日

警示语：这不是一份正式的实验报告，仅仅是用于编写程序的随手笔记。

经过一天艰苦卓绝的斗争，我终于掌握了如何使用 MATLAB 求解解析梯度，但是我研究出来的这个东西对于不定维度的定义域实在是太不友好了，于是决定抛弃他，改回手解梯度。有的时候机器真的不能代替人类。不过我更想尝试一下能不能使用 Python 的 Mathematica 解决。有空再说。

1 对 Waston 函数的处理与计算

要求函数 $f(x)$ 的最小值，其中

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

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

显然可以发现，这个货长得太不优美了，那么我们不妨把它改写成

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

这个函数的形式事实上比较简单，用手拿起笔和纸计算它的偏导数（你要是想要口算或者动动脚趾头算你强）

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

进一步带入

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

这个时候我们就要考虑题目中给出的参数等等奇奇怪怪的条件

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

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

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

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

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

这样，梯度更容易写成清晰地矩阵运算的形式。

但是这么写虽然不太容易出错，计算的时候是无法处理一阶梯度、二阶 Hesse 矩阵和原函数值计算时间复杂度不同阶的问题。需要一切奇怪的方法避免掉。注意，函数值每次计算的时间复杂度是 $O(mn)$，直觉上而梯度、Hesse 矩阵的复杂度在一般情况下会依次提高一个 n。

然而真的么?

导数每增加一阶，每个位置上的自变量维度就减一，所以总的计算复杂度还是不变的。那我就可以放心地一次性计算出它的梯度和 Hesse 矩阵，而不必担心时间复杂度的增加。

核心问题是计算 r

构造两个辅助矩阵 T_1、T_2

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

再令

$$x_2 = \begin{bmatrix} 0 & x_2 & 2x_3 & 3x_4 & \cdots & (n-1)x_n \end{bmatrix}^T$$

那么写成 MATLAB 表达式的形式

```
r = T2 * x2 - power(T1 * x, 2) - 1
```

特殊处理一下后面两位

该算偏导数了，比较简单，这里略去。

Hesse 矩阵应该更简单了。但是我决定放弃优美的写法。

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

于是修了修代码。最难的部分结束。

```
01 function [f, r, grad, H, counter] = waston(x, n, counter_init)
02     m = 29;
03     t = (1:29) ./ 29;
04     T0 = repmat(t', 1, n - 1);
05     T0 = T0 .^ (0:n-2);
```

```
06      T1 = [T0, power(t, n - 1)'];
07      T2 = [zeros(m, 1), T0];
08
09      r = T2 * (x .* (0:n-1)') - power(T1 * x, 2) - 1;
10      r = [r; x(1); x(2) - x(1) ^ 2 - 1];
11
12      f = r' * r;
13
14      C = T1 * x;
15      prx = (0:28)' .* T2 - 2 * T1 .* C;
16      prz = [prx; [1, zeros(1, n-1)]; [-2 * x(1), 1, zeros(1, n-2)]];
17      size(prz)
18      size(r)
19      grad = 2 * prz' * r;
20
21      H = zeros(n, n);
22      for a = 1:n
23          for b = 1:n
24          H(a, b) = 2 * prz(:, b)' * prz(:, a) - 4 * 29 * sum(power(t, a + b - 2));
25              if (a == 1) && (b == 1)
26                  H(a, b) = H(a, b) - 4;
27              end
28          end
29      end
30      counter = counter_init + 1;
31  end
```

2 线搜索算法

Numerical Optimization 一书第 3.4 节写到，“For Netwon and quasi-Newton methods the step α_0 should always be used as the initial trial step length. This choice ensures that unit step lengths are taken whenever they satisfy the termination conditions and allows the rapid rate-of-convergence properties of these methods to take effect.”

至于这是为什么。作者：实践是检验真理的唯一标准。

最 naive 的方法就是设定一个 $\rho < 1$，然后设定 α 的初始值然后逐步乘 ρ 直至它满足我们的几条准则。

书上还有一个基于两点三次诶尔米特插值的满足强 Wolfe 条件的算法，最后再搞

Aroijo 准则

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

Wolfe 准则

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

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

强 Wolfe 准则

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

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

Goldstein 准则

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

接下来我们考虑精确线搜索问题。这就是一个最优化一个标量的算法

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

就是……就是……就……
你妹的
优化函数

$$A_i = \sum_{j=i}^{n} x_j t_i^{j-1} \quad , B_i = \sum_{j=1}^{n} p_j t_i^{j-1} \quad , C_i = \sum_{j=1}^{n} (j-1) p_j t_i^{j-2}$$

$$\sum_{i=1}^{29} \left(r_i(x) + (2A_i B_i + C_i)\,\alpha - B_i^2 \alpha^2 \right)^2 + (x_1 + \alpha p_1)^2 + (x_2 + \alpha p_2 - (x_1 + \alpha p_1)^2 - 1)^2$$

3 参考值

使用 Tensorflow 得到一些近似的局部最优解以作为参考。为了简单，我们只考虑 n 为 10,15,20 的情况（再多我就死了）

$n = 10$, 使用 learning_rate=0.0001, 迭代 1000000 次的梯度下降得到近似收敛值 2×10^{-5} $(2.17652988556e-05)$

$x = [-6.106e{-}05\ 9.981e{-}01\ 3.041e{-}02\ 2.088e{-}01\ 1.767e{-}01\ 8.034e{-}02\ -1.076e{-}04\ -3.122e{-}02\ -5.505e{-}04$

警示语：以下内容在 2017 年 10 月 20 日的更新中被移除。

然后喜闻乐见得到了梯度

$$\nabla f = \begin{bmatrix} 2\sum_{i=1}^{29} r_i(x) \left[-2 \left(\sum_{j=1}^{n} x_j \, t_i^{j-1} \right) \right] + 2r_{30}(x) - 4x_1 \, r_{31}(x) \\ 2\sum_{i=1}^{29} r_i(x) \left[1 - 2 \left(\sum_{j=1}^{n} x_j t_i^{j-1} \right) \, t_i \right] + 2r_{31}(x) \\ \cdots \\ 2\sum_{i=1}^{29} r_i(x) \left[(k-1) \, t_i^{k-2} - 2 \left(\sum_{j=1}^{n} x_j t_i^{j-1} \right) \, t_i^{k-1} \right] \\ \cdots \\ 2\sum_{i=1}^{29} r_i(x) \left[(n-1) \, t_i^{n-2} - 2 \left(\sum_{j=1}^{n} x_j t_i^{j-1} \right) \, t_i^{n-1} \right] \end{bmatrix} \tag{1}$$

如此科学优美的梯度表达式，大家快给自己鼓鼓掌！（前两项实在是反人类，不过还好，特判一下就好了）

接下来的问题就是怎么计算了。讲道理的话，计算梯度比计算原函数的时间复杂度高出一个维度是很正常的，所以我也就不谋求什么优美的写法了。硬来呗。当然尽可能的向量化可以大幅提升计算效率。

写着写着发现这个梯度并不是那么容易算，于是还需要变形一下

$$2(k-1)\sum_{i=1}^{29} r_i(x) \, t_i^{k-2} - 4\sum_{j=1}^{n} x_j \sum_{i=1}^{29} r_i(x) \, t_i^{j+k-2}$$

写一下午加一晚上就这么几行代码

```
01 function [f, r, counter] = watson(x, n, cnt)
02     r = zeros(31, 1);
03     t = rep((1:29) / 29, 1, n);
04     tw1 = t .^ (0:n-1);
05     tw2 = (t .^ (-1:n-2)) .* (0:n-1);
06
07     r(1:29) = tw2 * x - (tw1 * x) .^ 2 - 1;
08     r(30) = x(1);
09     r(31) = x(2) - x(1) ^ 2 - 1;
10
11     f = r' * r;
12     counter = cnt + 1;
13 end
```

```
01 function [g, counter] = watson_grad(x, n, r, cnt)
02     g = zeros(31, 1);
03     t = (1:29) / 29;
04     T = repmat(t, n, 1);
05     for k = 1:31
06         Tw = T .^ (k-1:n+k-1)';
07         g = 2 * (k - 1) * r' * (t .^ (k - 2)) - 4 * x' * Tw * r;
08     end
09     g(1) = g(1) + 2 * r(30) - 4 * x(1) * r(31);
10     g(2) = g(2) + 2 * r(31);
11     counter = cnt + 1;
12 end
```

写到这里就会发现剩下的几种代码包括最速下降阻尼牛顿修正牛顿之类的都太好写了。过两天再写。。。