最优化方法

一、实验内容

使用 matlab 编写 3 个算法程序，分别为

1.负梯度方法；

2.非线性共轭梯度法；

3.DFP 方法；

其中 3 个方法使用的线搜索均将精确线搜索替换为非精确线搜索，并使用这三个程序解决无约束非线性最优化问题，问题和要求如下：

编写以下程序:

- 一个非精确线搜索的程序.
- 负梯度方法和非线性共轭梯度法的程序.
- DFP方法的程序.

对优化问题

$$\min \sum_{i=1}^{m} r_i^2(x), \tag{1}$$

利用编好的程序计算(1), $r_i(x)$为Watson函数:

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

其中$t_i = \frac{i}{29}, 1 \le i \le 29, r_{30}(x) = x_1, r_{31} = x_2 - x_1^2 - 1, n \in \{2, 6, 9, 12, 20, 31\}, m = 31$. 初始点可选为$x^{(0)} = (0, \ldots, 0)^\top$.

通过计算, 输出算法的迭代次数、CPU时间、迭代结束时的目标函数值, 比较不同方法的有效性.

二、算法实现

2.1.非精确线搜索(输入行向量)

该部分的算法有两大准则，一个是 Armijo-Goldstein 准则，另一个是 Wolfe-Powell 准则。由于需要使得算法收敛，所以各自有两个规则。

Armijo-Goldstein 准则，其中$\alpha_k \in (0,1)$，$\rho \in (0,0.5)$：

$$f(x_k + \alpha_k d_k) \le f(x_k) + \alpha_k \rho g_k^T d_k \cdots\cdots\cdots (1)$$

$$f(x_k + \alpha_k d_k) \ge f(x_k) + \alpha_k (1-\rho) g_k^T d_k \cdots\cdots\cdots (2)$$

第一个条件是为了确保得到的 x_{k+1} 的函数值下降(包括ρ的限制)， 第二个条件是为了确保不能

α_k 太小。在搜索的时候可能会将极小值区间排除在外，而 Wolfe 准则能解决这个问题，但是在实践的时候 Wolfe 的效率远不如 Armijo，可能调的参数不太理想，所以本实验还是基于 Armijo 准则实现非精确线搜索算法。

由于条件(2)只是限制不能α_k太小，条件(1)才是下降的条件，所以将该条件用作判断 x_{k+1} 是否符合。主要思想如下：

先做 20 次迭代，令α初值为 0.5，ρ初值为 1e-6，迭代过程中如果符合条件（1）则直接返回此时的α，如果不符合条件则α自乘初值(0.5,0.25,0.125…)。迭代过程的 20 次都不符合条件，则直接返回α=0.5^20。经过该算法得到的α用于计算 x_{k+1}, $x_{k+1}=x_k+\alpha d_k$，就能得到新的自变量。

2.2.负梯度方法(所有方法 ε 均设为 10^{-5}，输入都是行向量)

（1）给点初始点 $x^{(1)}$，允许误差 ε> 0，置 k = 1;

（2）若 $\left\|\nabla f(x^{(k)})\right\|<\varepsilon,$ 则停，$x^*=x^{(k)}$，否则，令 $d^{(k)}=-\nabla f(x^{(k)}),$

（3）在 $x^{(k)}$ 处沿方向 $d^{(k)}$ 作线搜索得 $x^{(k+1)}=x^{(k)}+\lambda_k d^{(k)},$ $k=k+1$, 回（2）.其中线搜索方法为 2.1 提到的方法。

2.3 非线性共轭梯度法

（1）给点初始点 $x^{(1)}$，令 $p_1=-\nabla f(x^{(1)}),$ k = 1;

（2）若 $\nabla f(x^{(k)})=0,$ 则停，$x^*=x^{(k)}$；否则，令

$$x^{(k+1)}=x^{(k)}+\alpha_k d^{(k)},$$ 其中α_k=Armijo(x_k,d_k);

$$d^{(k+1)}=-\nabla f(x^{(k+1)})+\frac{\left\|\nabla f(x^{(k+1)})\right\|^2}{\left\|\nabla f(x^{(k)})\right\|^2}d^{(k)};$$

2.4 DFP 方法

算法 2.1 (DFP 算法)

1. 给定初值 $\mathbf{x}_0$ 和精度阀值 ϵ, 并令 $D_0 = I$, $k := 0$.

2. 确定搜索方向 $\mathbf{d}_k = -D_k \cdot \mathbf{g}_k$.

3. 利用 *(1.13)* 得到步长 λ_k, 令 $\mathbf{s}_k = \lambda_k \mathbf{d}_k$, $\mathbf{x}_{k+1} := \mathbf{x}_k + \mathbf{s}_k$.

4. 若 $\|\mathbf{g}_{k+1}\| < \epsilon$, 则算法结束.

5. 计算 $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.

6. 计算 $D_{k+1} = D_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{D_k \mathbf{y}_k \mathbf{y}_k^T D_k}{\mathbf{y}_k^T D_k \mathbf{y}_k}$.

7. 令 $k := k + 1$, 转至步 *2*.

其中第三步是使用的是精确线搜索，按照要求将其替换为 2.1 的非精确线搜索。其中 gk 就是函数在 xk 的梯度。

2.5 fun 和 gfun(均处理行向量)

fun 就是最优化问题对应的函数，gfun 就是 fun 的梯度(程序中命名为 gradfunc)，而且这些函数的传入参数都有 x 的维度 dimen。为了求导方便，又写了一个 mix 函数，该函数对应 rix 的平方项里面的求和项，也就是

mx(i) = $\mathbf{t}_i^{j-1} * \mathbf{x}_j$，j 从 1 到 dimen 求和$mx(i) = \sum_{j=1}^{dimen} \mathbf{xj} * \mathbf{t}_i^{j-1}$，

rx(i) = $\sum_{j=2}^{dimen} (\mathbf{j}-\mathbf{1}) * \mathbf{x}_j * ti^{j-2}$-$mx(i)^2$-1，

fun= $\sum_{i=1}^{29} rx(i)^2$ + x_1^2 + (x2 –x_1^2-1)^2;

对 gfun，若 j∈[3,dimen]，ti = i/29：

$$\text{gfun(j)} = \sum_{i=1}^{29} 2\mathbf{rx(i)}((\mathbf{j}-\mathbf{1})\mathbf{t}_i^{j-2} - 2 * \mathbf{mx(i)} * \mathbf{t}_i^{j-1})$$

若 j=1，$\text{gfun(1)} = \sum_{i=1}^{29} 2\mathbf{rx(i)} * (-2\mathbf{mx(i)}) + 2x_1^2 + 4(x_1^3 + x_1 - x_1x_2)$

若 j=2，$\text{gfun(2)} = \sum_{i=1}^{29} 2\mathbf{rx(i)}(1 - 2 * \mathbf{mx(i)} * t_i) + 2(x_2 - x_1^2 - 1)$

通过上述过程就能实现 fun 和 gfun(gradfunc)。

三、实验结果

各项参数设置：Armijo 的参数：迭代次数最大值 mk=20，算法中的σ对应条件中的ρ，值为 10^{-6}，$\alpha = \rho^m$，其中ρ等于 0.5。三个算法的 ε 都是 10^{-5}。下面就是三种方法对不同维度的 x 运行程序得到的迭代次数、CPU 时间以及目标函数值的结果：

负梯度方法(20 和 31 时间太久不纳入表格)：

性能 \ n	2	6	9	12	20	31
迭代次数	40	271176	3285453	981314	-	-
CPU 时间/s	0.0191	122.3113	3880.8	1568.7	-	-
目标函数值	0.5466	0.0023	7.7383e-6	2.2080e-7	-	-

非线性共轭梯度法：

性能 \ n	2	6	9	12	20	31
迭代次数	35	3395	9520	5657	5908	17187
CPU 时间/s	0.0029	2.9079	16.1352	15.0735	32.0273	124.8396
目标函数值	0.5466	0.0023	1.4227e-6	9.3595e-8	8.3607e-9	1.2690e-8

DFP 方法：

性能 \ n	2	6	9	12	20	31
迭代次数	9	502	162832	1782	20470	3952121
CPU 时间/s	0.0246	0.1031	43.9075	0.8462	16.2989	4402.3
目标函数值	0.5466	0.0023	1.3998e-6	1.5723e-7	6.5963e-9	1.6763e-11

四、实验结论和体会

可以从上述三个表格看到，共轭梯度法和 DFP 方法都是比负梯度方法要好得多，而且根据 n 等于 2、6、12、20 这几个结果看来 DFP 性能优于共轭梯度法。但这都只是片面的，因为非精确线搜索的搜索时间还是比较看运气的，主要是看设置的参数值为多少，因为参数的设置会影响每一次迭代的

步长，进而影响后续的搜索方向和步长。实验过程中还尝试了线搜索的σ的值设置为 1e-4、0.1、0.2，也尝试更改ρ的值为 0.8、0.7，第三部分已经给出同一参数下较好的结果了。但是如果乱调参数可能会导致结果不收敛,因为 Armijo 有着确保步长不能过小的条件，一旦过小会导致条件(1)不成立，就会出现无穷大的情况。但是实验过程中使用 Wolfe 准则，内部用二分法实现，当 n 为 9 时 DFP 和共轭梯度法使用该线搜索算法收敛速度很慢，更不用说 n=31 了，所以舍弃了该方法。