# 最优化方法课程作业实验报告

17341125　蒙亚愿　信息安全

## 1 实验要求

编写以下程序：

一个非精确搜索的程序，负梯度方法和非线性共轭梯度法的程序，$DFP$方法的程序。

利用编好的程序，计算优化问题$min \sum_{i-1}^{m} r_i^2(x)$，

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

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$x^{(0)} = (0, \ldots\ldots, 0)^T$。

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。

## 2 实验介绍

### 2.1 实验综述

规定初始点$x_0$，容许误差$eps$，以及$n$（$x_0$的维数）的取值。

根据$Wolfe$准则编写非精确线搜索程序，求出迭代步长$step$。再根据负梯度法、非线性共轭梯度法和DFP法计算迭代的$x_k$，判断终止条件，从而得到最小值$f_i$，最小值点$x_i$，迭代次数$k_i$以及计算时间$time_i$（$i \in 1,2,3$）。

其中，对函数$\sum_{i-1}^{m} r_i^2(x)$，计其函数$f$和梯度公式$G$，之后就可直接代入$x$，得到迭代中的函数值和梯度值。

同时，利用优化工具箱计算得到最小值，作为参照。

### 2.2 开发环境

运行环境：Windows  MATLAB R2019a

## 3 实验算法

### 3.1 非精确线搜索

$Wolfe$准则：

给定$\rho \in (0, 0.5), \sigma \in (\rho, 1)$，求$\alpha_k$使得以下两个不等式同时成立：

$$f(x_k + \alpha_k d_k) \le f(x_k) + \rho \alpha_k g_k^T d_k \ldots\ldots\ldots （1）$$
$$\nabla f(x_k + \alpha_k d_k)^T d_k \ge \sigma g_k^T d_k \ldots\ldots\ldots\ldots\ldots （2）$$

其中，$g_k = g(x_k) = \nabla f(x_k)$。

由于$g_k^T d_k < 0$，可以证明$Wolfe$准则的有限终止性，即$\alpha_k$的存在性。

计算步骤

1. 给定$\rho = 0.25$，$\sigma = 0.75$，$\alpha = 1$，$a = 0$，$b = inf$，k=0
2. 若满足式（1）转3，否则

$$b = \alpha_k$$
$$\alpha_{k+1} = \frac{\alpha_k + a}{2}$$

   继续计算2。

3. 若满足式（2），有

$$a = \alpha_k$$
$$\alpha_{k+1} = min\{2\alpha_k, \frac{b + \alpha_k}{2}\}$$

   返回$\alpha_k$。否则，转2。

---

## 3.2 负梯度方法

基本思想：设$f(x) \in C^1$。由于函数$f(x)$沿负梯度方向$-\nabla f(x)$下降最快，所以选取它作为每次迭代的搜索方向，即

$$d_k = -\nabla f(x_k)$$

计算步骤：

任给一个初始点$x_0$，利用上述的非精确线性搜索计算步长$step_0 = \alpha$，再计算$d_0 = -\nabla f(x_0)$，从$x_0$出发沿着$d_0$方向在半直线$x_0 + step_0\ d_0$上求得$f(x)$的第一个近似极小值点$x_1 = x_0 + step\, d_0$，再以$x_1$为起点，重复此过程，求得第二个近似极小值点$x_2 = x_1 + step_1\ d_1$，直到梯度值在允许的误差内为止。

迭代过程：

1. 给定初始点$x_0$，允许误差$eps$，令$k = 0$，$x_k = x_0$。
2. 计算$||\nabla f(x_k)||$
3. 若$||\nabla f(x_k)|| < eps$，则迭代结束，返回$k$，$x = x_k$，$f = f(x)$。否则，转4.
4. 令$d_k = -||\nabla f(x_k)||$，利用非线性搜索求步长$step$，更新$x_k$：

$$x_{k+1} = x_k + step * dk$$

   $k = k + 1$，转2。

---

## 3.3 非线性共轭梯度法

基本思想：找到n个两两共轭的方向，每次沿着一个方向优化得到该方向上的极小值后面再沿着其他方向极小值的时候，不会影响前面的得到的沿着那些方向的极小值，对n个方向都求出极小值就得到了n维问题的最小值。

迭代过程：

1. 给定初始点$x_1$，令方向$k = 1$，$x_k = x_1$，$d_k = -\nabla f(x^{(1)})$
2. 若$\nabla f(x) < eps$，则迭代结束，返回$k$，$x = x_k$，$f = f(x)$。否则，转3.
3. 利用非精确性线搜索，计算步长$step = Wolfe(x_k, d_k)$，有

$$x_{k+1} = x_k + step * d_k$$
$$d_{k+1} = -\nabla f(x_{k+1}) + \frac{||\nabla f(x_{k+1})||^2}{||\nabla f(x_k)||^2} d_k$$

$k = k + 1$，转2.

---

### 3.4 DFP法

基本思想：用对称正定矩阵 $H_k$近似$(\nabla^2 f(x_k))^{-1}$，而$H_k$的产生从给定的$H_0$开始逐步修正得到的。

迭代过程：

1. 给定$H_0 = I(\text{单位矩阵})$，$k = 0$，$x_k = x_0$，利用非精确性线搜索计算步长$step$，然后计算
$$d_k = -H_0 * \nabla f(x_k)$$
$$x_{k+1} = x_k + step * d_k$$
2. 若$||\nabla f(x_{k+1})|| < eps$，迭代结束，返回返回$k$，$x = x_k$，$f = f(x)$。否则转3.
3. 令
$$\sigma_{k+1} = x_{k+1} - x_k$$
$$\gamma_{k+1} = \nabla f(x_{k+1}) - \nabla f(x_k)$$
$$E_{k+1} = \frac{\sigma_{k+1}\sigma_{k+1}^T}{\sigma_{k+1}\gamma_{k+1}} - \frac{H_k\gamma_{k+1}\gamma_{k+1}^T H_k}{\gamma_{k+1}^T H_k \gamma_{k+1}}$$
$$H_{k+1} = H_k + E_{k+1}$$
$$d_{k+1} = -H_{k+1} * \nabla f(x_{k+1})$$

非精确行线搜索计算步长$step_{k+1} = Wolfe(x_k, d_k)$

$k = k + 1$，$x_{k+1} = x_k + step_k d_k$，转2.

---

## 4 项目实现

### 4.1 $main.m$部分

功能：在main部分给定初始点$x_0$，同时标明$x_0$的长度$n$，容许误差$eps$。计算函数和梯度的表达式，然后用负梯度法、非线性共轭梯度法和DFP法计算函数的最小值。

由于在计算函数值和梯度值时，会有多次的循环，且每次代入不同的点$x_k$时，循环的内容都一样，为了省去不必要的时间开销，在程序的一开始，就先计算$f(x)$和$\nabla f(x)$的公式，在之后各个算法的迭代计算时，就直接带入已经计算好的公式就可以。

所以需要3个全局变量：

```
global f G flag
```

代表的意义如下：

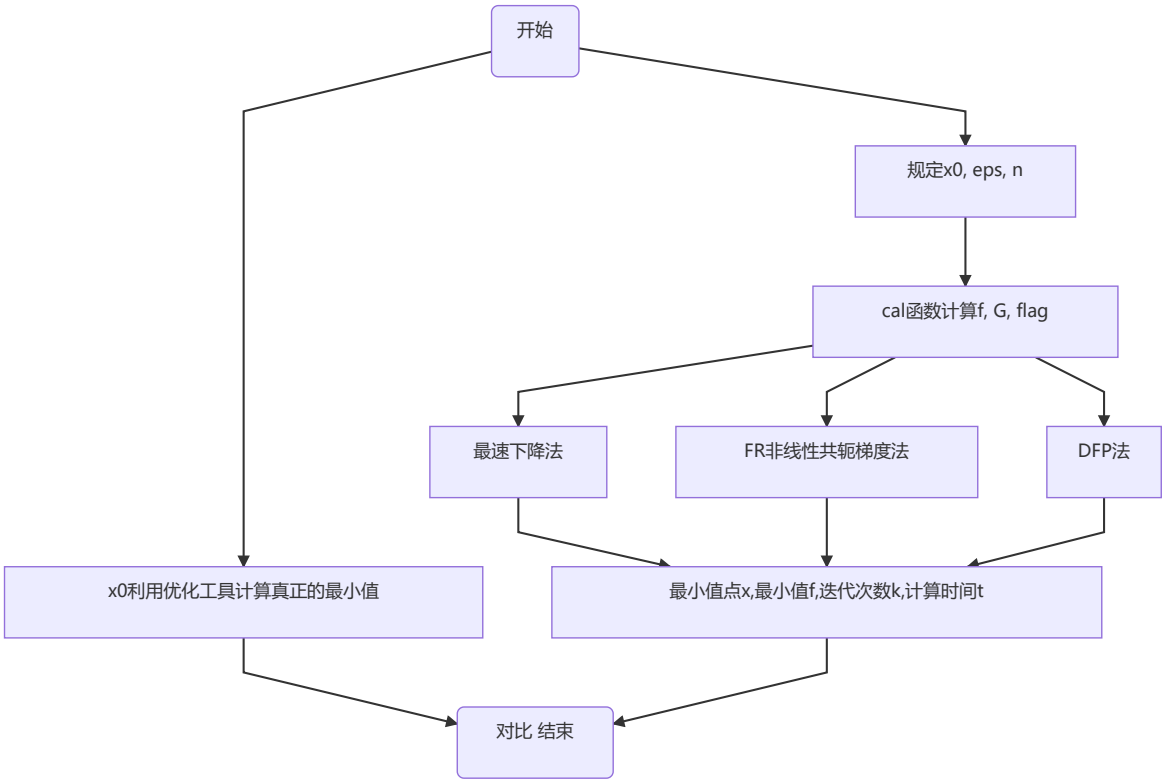
| 变量 | 含义 |
| --- | --- |
| $flag$ | double类型，若$flag = 1$，则已经计算好了$f$和$G$，否则就没有 |
| $f$ | sym类型，是一个关于$x$的函数$f(x)$，$x$是一个n×1的矢量，$f$是一个标量 |
| $G$ | sym类型，是一个关于$x$的函数$\nabla f(x)$，$x$是一个n×1的矢量，$G$是一个1×n的矢量 |

在函数cal(n)内计算以上三个全局变量，并统计计算时间。

然后，就可以通过负梯度法、非线性共轭梯度法、DFP法计算函数的最小值、得到迭代次数和计算时间。

同时，用matlab中的优化工具计算真正的最小值。

流程图：




实现如下：

```
clear all ;
clc ;
%%
x0 = [0;0;] ;
eps = 1.0e-3 ;
n = 2 ;

%%
tic
global f G flag
cal(n) ;
toc

%%
options = optimoptions('fmincon','MaxFunctionEvaluations',1000000) ;
[x_min, f_min] = fminunc(@cal_min, x0)

%% 负梯度法
[x1,f1,k1] = opt_steepest(eps,x0)

%% FR法
[x2, f2, k2] = Conjugate_grad(eps, x0)

%% DFP法
[x3, f3, k3] = DFP(eps, x0)
```

## 4.2 $cal.m$函数

输入：变量$x$的长度$n$

功能：计算全局变量$flag, f, G$

计算函数$f$和$G$的公式：

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

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

$$其中，\ t_i = \frac{i}{29}, 1 \le i \le 29, r_{30}(x) = x_1, r_{31} = x_2 - x_1^2 - 1$$

$$n \in \{2,6,9,12,20,31\}, m = 31。x = [x_1, x_2, \ldots\ldots, x_n]^T$$

$$G = \nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots\ldots \frac{\partial f}{\partial x_n}]$$

首先创建符号变量$x_1$，$x_2$，$\ldots\ldots x_n$，然后放在$X$里，$X = [x_1, x_, \ldots\ldots, x_n]$。

```
for i = 1:n
    syms(['x', num2str(i)]) ;
end
syms X
X = sym('x', [1,n]) ;
```

再创建符号$r, f1, f2$，含义如下：

| 符号变量 | 含义 |
| --- | --- |
| $r$ | 函数$\sum_{i-1}^{m} r_i^2(x)$ |
| $f1$ | 每次i的循环中的$\sum_{j=2}^{n}(j-1)x_j t_i^{j-2}$ |
| $f2$ | 每次i的循环中的$\sum_{j-1}^{n} x_j t_i^{j-1}$ |

在前1~29次$i$循环后，有

$$r = r + (f_1 - f_2^2 - 1)^2$$

$i = 30$时，有

$$r = r + x_1^2$$

$i = 31$时，有

$$r = r + (x_2 - x_1^2 - 1)^2$$

最后得到的$r$就是一个关于矢量$x$的函数，对$r$求雅可比矩阵就得到函数的偏导，此时函数和偏导都已经计算完成，再令$flag = 1$。

```
f = r ;
G = jacobian(r, sym('x', [1,n])) ;
flag = 1 ;
```

代码实现：

```
function [f, G, flag] = cal(n)
    global f  G  flag
    for i = 1:n
        syms(['x', num2str(i)]) ;
    end
    syms X
    X = sym('x', [1,n]) ;

    syms r f1 f2
    r = 0 ;
    for i = 1:29
       f1 = 0 ;  f2 = 0 ;
       for j = 1:n
          if(j~=1)
             f1 = f1 + (j-1) * X(j) * (i/29)^(j-2) ;
          end
          f2 = f2 + X(j) * (i/29)^(j-1) ;
       end
       r = r + (f1 - f2^2 -1)^2 ;
    end

    r = r + X(1)^2 ;
    r = r + (X(2) - X(1)^2 - 1)^2 ;

    f = r ;
    G = jacobian(r, sym('x', [1,n])) ;
    flag = 1 ;
end
```

---

## 4.3 $fun.m$函数

输入：变量$x$

返回：函数值$r = f(x)$

功能：把$x$代入已经计算好的函数$f$中，得到函数值$r$

在程序开始的时候就计算好了函数的表达式，$flag = 1$，所以每次计算函数值时，就直接用subs函数代入就可以，再规定用小数表示就可以。

```
function [r] = fun(X)
    global f
    global flag
    n = length(X) ;
    if(flag==1)
        r = vpa(subs(f, sym('x', [1,n]) , X')) ;
    end
end
```

---

## 4.4 $gfun.m$函数

输入：变量$x$

返回：函数梯度值$gf = \nabla f(x)$

功能：功能：把$x$代入已经计算好的梯度函数$G$中，得到梯度$gf$

同fun，在程序开始的时候就计算好了梯度的表达式，$flag = 1$，所以每次计算梯度值时，就直接用subs函数代入就可以，再规定用小数表示就可以。

```
function gf = gfun(x)
    n = length(x) ;
    global flag
    global G
    if(flag==1)
        gf = subs(G,sym('x',[1,n]), x')' ;
        gf = vpa(gf) ;
    end
end
```

## 4.5 $Opt_Wolfe.m$函数

输入：初始点$x_k$，方向$d_k$

输出：步长$\alpha$

功能：根据$Wolfe$准则进行非精确行线搜索，计算每次迭代的步长$stepp = \alpha_k$

根据算法3.1中迭代步骤，可以直接实现。第一个if条件判断（1）式，第二个if条件判断（2）式。

```
function [alpha] = Opt_Wolfe(xk, dk)
    rho = 0.25; sigma = 0.75;
    alpha = 1; a = 0; b = Inf;
    while (1)
        if ~(fun(xk+alpha*dk)<=fun(xk)+rho*alpha*gfun(xk)'*dk)
                b = alpha;
                alpha = (alpha+a)/2;
        continue;
        end
        if ~(gfun(xk+alpha*dk)'*dk >= sigma*gfun(xk)'*dk)
                a = alpha;
                alpha = min([2*alpha, (b+alpha)/2]);
        continue;
        end
        break;
    end

end
```

## 4.6 $opt_steepest.m$函数

输入：允许误差$eps$，初始点$x_0$

返回：最小值点$x$，最小值$f$，迭代次数$k$

功能：通过负梯度法计算函数的最小值。

根据算法3.2，按照其迭代步骤，可以直接实现，其中计算步长时，调用$Opt_Wolfe$函数。最后的结果用小数表示。

```
function [x,f,k] = opt_steepest(eps,x0)
    digits(10) ;
    [gf] = gfun(x0) ;
    xk = x0 ;  k = 0;
    judge = abs(norm(gf))   ;
    while( judge > eps)
        dk = -gf ;
        step = Opt_Wolfe(xk, dk)  ;
        xk = xk + step * dk    ;
        [gf] = gfun(xk)   ;
        k = k+1 ;
        judge = abs(norm(gf))  ;
    end
    x = vpa(xk) ;
    f = vpa(fun(xk)) ;
end
```

## 4.7 $Conjugate_grad.m$函数

输入：允许误差$eps$，初始点$x_0$

返回：最小值点$x$，最小值$f$，迭代次数$k$

功能：通过负梯度法计算函数的最小值。

根据算法3.3，按照其迭代步骤，可以直接实现，其中计算步长时，调用$Opt_Wolfe$函数。最后的结果用小数表示。

```
function [x,f,k] = Conjugate_grad(eps, x0)
    digits(10) ;
    xk = x0 ;  k = 1 ;
    gf = gfun(xk) ;
    judge = norm(gf) ;
    dk = -gf ;

    while(abs(judge) > eps)
        step = Opt_Wolfe(xk, dk)  ;

        xk_pre = xk ;
        gf_pre = gf ;

        xk = xk + step * dk  ;
        gf = gfun(xk)  ;

        dk = -gf + ( norm(gf)^2 / norm(gf_pre)^2 ) * dk   ;

        judge = norm(gf)   ;

        k = k+1 ;
    end
    x = xk ;
    f = fun(xk) ;
end
```

## 4.7 $DFP.m$函数

输入：允许误差$eps$，初始点$x_0$

返回：最小值点$x$，最小值$f$，迭代次数$k$

功能：通过负梯度法计算函数的最小值。

根据算法3.4，按照其迭代步骤，可以直接实现，其中计算步长时，调用$Opt_Wolfe$函数。最后的结果用小数表示。

```
function [x,f,k] = DFP(eps, x0)
    digits(10) ;
    H= eye(length(x0))   ;
    xk_pre = x0 ;
    gf_pre = gfun(x0)  ;
    dk = -H * gf_pre   ;
    step = Opt_Wolfe(x0, dk)   ;
    xk = xk_pre + step * dk   ;
    gf = gfun(xk)   ;
    k = 0;

    judge = abs(norm(gf))   ;
    while(judge > eps)
        delta = xk - xk_pre   ;
        gamma = gf - gf_pre   ;
        H = H + (delta * delta')/(delta' * gamma) - (H * gamma * gamma' *
H)/(gamma' * H * gamma)   ;
        dk = -H * gf   ;
        step = Opt_Wolfe(xk, dk)   ;
        xk_pre = xk   ;
        xk = xk_pre + step * dk   ;

        gf_pre = gfun(xk_pre)  ;
        gf = gfun(xk)   ;

        judge = abs(norm(gf))     ;
        k = k+1 ;

    end
    x = xk   ;
    f = fun(xk)   ;

end
```

## 4.8 $cal_min.m$函数

输入：初始点$X$

返回：函数值$y$

功能：严格按照公式循环计算，代入点值$X$，得到确切的函数值$y$。用于优化工具的函数句柄，计算出真正的最小值，作为以上三个算法得到的最小值参考。

```
function y = cal_min(X)
    r = 0 ;
    n = length(X) ;
    for i = 1:29
       f1 = 0 ;  f2 = 0 ;
```

```
        for j = 1:n
            if(j~=1)
                f1 = f1 + (j-1) * x(j) * (i/29)^(j-2) ;
            end
            f2 = f2 + x(j) * (i/29)^(j-1) ;
        end
        r = r + (f1 - f2^2 -1)^2 ;
    end

    r = r + x(1)^2 ;
    r = r + (x(2) - x(1)^2 - 1)^2 ;

    y = r ;

end
```

---

## 5 实验结果分析及思考

首先用一个简单的二次型函数进行测试。

$$x_0 = [1, 1]^T$$
$$f(x) = x_1^2 + 4x_2^2$$

显然，该函数的最小值在点$(0; 0)$处，函数值为0。

各个算法的结果如下：

| 算法 | 最小值点$x_k$ | 最小值$f_k$ | 迭代次数 | 耗时(seconds) |
|---|---|---|---|---|
| **负梯度法** | (0； 0) | 0 | 2 | 2.053140 |
| **FR法** | 1.0e-04 *(-8622; 0.2669) | 1.0282e-08 | 11 | 0.030228 |
| **DFP法** | 1.0e-04 *( -0.0567; -0.4515) | 8.1863e-09 | 2 | 0.021890 |

可以看到，负梯度法可以准确地找到最小值点和最小值，非线性共轭梯度法和DFP法找到的相比之下存在一定的误差，但是也近似准确。

从迭代次数上，负梯度法和DFP法占优，FR法迭代次数是其他两个的5倍。

但是从时间上看，FR法和DFP法耗时约是负梯度法的1%，是非常快的。

总的来说，在不考虑时间成本上，负梯度法得到最好的结果；如果考虑时间成本，我认为DFP法相对来说比较好。FR法在这两个方面并不占优。

---

用$n = 2$，初始点$x_0 = (0; 0)$的Watson函数平方求和测试。

用优化工具和三种算法计算得：

| 算法 | 最小值点$x_k$ | 最小值$f_k$ | 迭代次数 | 耗时(s) |
| --- | --- | --- | --- | --- |
| **优化工具箱** | (-0.5014; 1.0737) | 0.5466 | 7 | 0.539359 |
| **负梯度法** | (-0.5013525704; 1.073647554) | 0.5466078588 | 11 | 6.979397 |
| **FR法** | (-0.5013844057; 1.073650374) | 0.5466078604 | 7 | 3.954554 |
| **DFP法** | (-0.5013686318; 1.07365019) | 0.5466078559 | 7 | 3.020269 |

由于优化工具箱中没有共轭梯度法来计算方向，所以就和其他三个算法对比，结果差不多。

可以看到三个优化算法计算得到的结果和优化工具箱计算的结果四舍五入之后得到的一样，所以三个的计算结果都可以算是精确的。

优化工具箱的耗时都非常少。三个算法中仍然是DFP法耗时最少，FR法比DFP法少一点，负梯度法耗时最多。

负梯度法的迭代次数比优化工具箱的多，而DFP法和FR法的迭代次数却比较少。

增大维度，$n=6$时，有

| 算法 | 最小值点$x_k$ | 最小值$f_k$ | 迭代次数 | 耗时(s) |
| --- | --- | --- | --- | --- |
| **优化工具箱** | -0.0157<br>1.0124<br>-0.2330<br>1.2603<br>-1.5136<br>0.9929 | 0.0023 | 52 | 0.535162 |
| **负梯度法** | -0.02082521792<br>1.00642359<br>-0.1708058551<br>1.052029529<br>-1.263841912<br>0.8840123683 | 0.002459873015 | 25631 | 33581.835293 |
| **FR法** | -0.01510272949<br>1.012228482<br>-0.2333156884<br>1.267820158<br>-1.526008409<br>0.999824659 | 0.002288881499 | 696 | 959.595869 |
| **DFP法** | -0.0157861574<br>1.012454076<br>-0.2331821907<br>1.260459287<br>-1.513298596<br>0.9925786623 | 0.002287699797 | 364 | 153.528753 |

此时负梯度法的计算时间约为9个半小时，而相比之下FR法和DFP法的时间就算是快的。所以在一定的误差允许下，维数越大，迭代次数越多，运行时间就越久。

此时，增加允许误差为0.01，维度仍然是6，看三个算法的计算结果和迭代时间。

| 算法 | 最小值点$x_k$ | 最小值$f_k$ | 迭代次数 | 耗时(s) |
| --- | --- | --- | --- | --- |
| **优化工具箱** | -0.0157<br>1.0124<br>-0.2330<br>1.2603<br>-1.5136<br>0.9929 | 0.0023 | 52 | 0.552250 |
| **负梯度法** | -0.0509441569<br>0.9810035749<br>0.128417882<br>-0.01341461204<br>0.04574220786<br>0.3037845132 | 0.00928584762 | 247 | 308.781145 |
| **FR法** | -0.003566191393<br>1.017671914<br>-0.2950412243<br>1.518983411<br>-1.850590236<br>1.156355439 | 0.002632709087 | 88 | 109.624375 |
| **DFP法** | -0.01340240716<br>1.012352698<br>-0.235845971<br>1.288085375<br>-1.557316792<br>1.017298301 | 0.002300155784 | 322 | 134.652351 |

增加允许的误差后，可以看到此时DFP法在精确度和运行时间上的性能还是比较好的，三种算法的精确度是DFP>FR法>负梯度法，同时不可否认的是DFP法和精确值仍有差距，但是相比于其他算法已经小很多了。

随着维数越来越大，每次迭代的步长也越来越小，导致迭代次数变多，消耗的时间越来越多。但是能够确定的是，在充分时间条件和良好的客观环境下，一定能够计算出最小值。

全部的测试结果可见excel表格。

纵观n在各个取值下，三个算法的性能。除了一些异常点，整体趋势仍是随着$x$的维度变大，计算时间复杂度和精确度下降，总体的性能评价为DFP法优于FR法，远远优于负梯度法。不过在特定情况（$n=9$）下，DFP法却体现不出优势，反而是FR法和负梯度法的效果比较好，具体是偶然还是事实，亟待考证。

总结，其实每种算法的特点就在于步长和搜索方向的计算上。在本次实验中，这三个算法的步长$step$都是通过$Wolfe$准则的非精确性线搜索得到的，所以区别只再与方向的计算上。负梯度法利用负梯度方向构造搜索方向，对二次型函数最有利，但是对高维问题的求解计算效率低。FR法可靠性较差。DFP法利用牛顿法的二次收敛的优点，通过构造变尺度矩阵逼近函数Hessian矩阵的逆矩阵，使得搜索方向渐渐地逼近牛顿方向，不需要计算函数的二阶导数及其逆矩阵；同时收敛速度很快；并且在一定的精度下，DFP得到的数据是精确度最高的，所以从时间、迭代次数和精确度三个方面来考虑，DFP是三个算法里的相对最优的算法。

---

全局变量$flag, f, G$的计算时间

| n | 时间(s) |
| --- | --- |
| 2 | 2.330282 |
| 6 | 3.506547 |
| 9 | 4.096319 |
| 12 | 4.547136 |
| 20 | 6.655753 |
| 31 | 11.711099 |

可以看到维数越大，计算函数的值也越大。如果不创建全局变量来记录函数的表达式，那么在每次迭代计算函数值和梯度时，都要再进行相同的循环计算，这样无谓的计算导致运行时间非常的久。我一开始没注意到这个问题的时候，$n=2$时，负梯度法耗时93.376045秒，改用全局变量后，就耗时6秒左右。所以需要把计算函数表达式的时间独立出来。

另外，由于我在fun函数和gfun函数中，使用subs函数，而subs函数效率很低，所以要尽量避免subs函数的使用，但是在计算函数值的时候，如果不用subs函数代入变量$x$，而是把$x$带入到循环里去计算，消耗的时间反而更多。在我没想到更好的方法之前，只好用subs函数，期待以后学到更多东西之后再回来优化一下算法。

---

## 6 实验感想

本次实验有点难度，尤其是在不是很熟悉matlab的情况下，之前都是用matlab来做图像处理，第一次拿来做数学题所以很多实现的细节需要学习。此外，算法老师上课已经讲的很清楚了，课后实现一下然后做大作业就和在一起就可以，这些感觉问题不大。主要是在函数那些怎么写感觉到头疼，因为平时课后测试的函数都很简单，它的梯度都能够直接写出来，而大作业的函数就比较复杂，直接写出梯度也很麻烦。然后我就查到用符号变量，也查了好久怎么用，虽然现在也不知道这样做对不对，但是应该也算实现了那个函数了吧，脑子不太灵光只能用笨笨的办法。但是这样的算法应该能实现，对于大维数只是时间问题。本学期的最优化学得比较快，一周只有一次课，但通过亲手做了一次大作业也确实学到了东西。老师教得很好，很有耐心、负责任，感谢一学期老师和TA的帮助，祝老师TA合家欢乐，万事如意~