无约束最优化问题

李翔
华中科技大学物理学院应用物理学2011级01班
学号：U201117605，邮箱：563624348@qq.com
联系电话：13026306754

摘要：本文基于无约束最优化问题，从一维直接法，黄金搜索法，牛顿法，到二维最优梯度法，牛顿法，最后实现了 BFGS 算法，通过对算法的学习和比较，从而可以看出不同算法的有效性和优缺点，并且发现了一些存在的问题，为下一步更深入的研究打下基础。

关键词：最优化问题 无约束 算法 BFGS

一 引言

方程求根和最优化问题在很大程度上是相关的，因为它们都涉及到一个函数关于某个点的猜测和搜索，方程求根是搜索一个或多个函数的零点，而最优化问题则是搜索函数的最大值或最小值点。从数学上来说，曲线在最优值处是平直的，对应于该点值的导数 f'(x)=0. 同时，二阶导数 f''(x)可以判断这个最优值是最大值还是最小值。

由上，我们明白了根和最优质的关系，同时这给我们提出一个寻找最优值的可行方案。即，先对函数微分，求新函数的零点。（这种方法较复杂）

最优化问题可以简单的分为有约束和无约束两个，我主要依次学习了一维无约束优化问题，有三种寻找单变量函数最优值的方法：1st 黄金分割搜索法 2nd 二次插值法 3rd 牛顿法。 这些方法在后续解决多维最优化问题也是可行的。 对于多为多维无约束最优化问题，有直接法：如随机搜索法，单变量搜索法和模式搜索法，不需要计算函数的导数。另一方面，有剃度法利用一阶导数，有时是二阶导数来寻找最优值。针对不同算法的优缺点，有有一个改进后效率更高的算法： 共轭梯度法，牛顿法，马夸特法和准牛顿法。

其中准牛顿法也称为变度量法，是用一种类似牛顿法的方式试图估计到达最优值的直线路径。

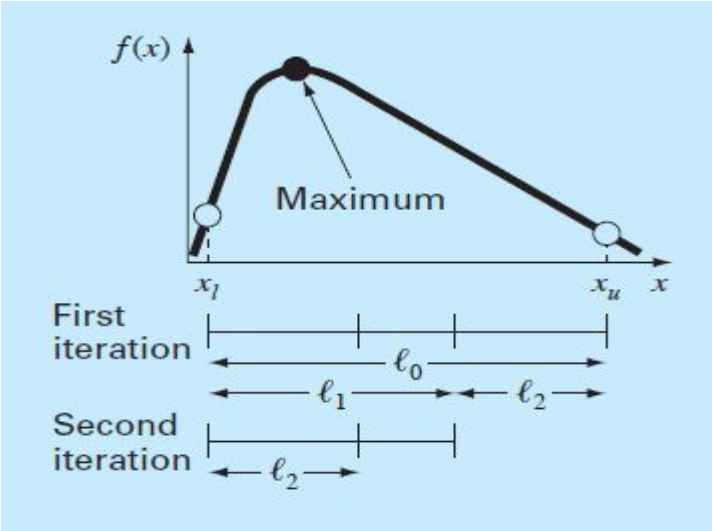
二．算法

PART I
一维无约束最优化问题

1.黄金分割法

原理：黄金分割搜索是一个简单的，通用的单变量搜索方法，在本质上和方程求根中的二分法相似。首先，定义一个区间，使其仅包含一个答案（单峰 unimodal）。用 a,b 表示该定义区间的上界和下界。为再次区间中找到最大值，需要三个函数值来确定是否存在最大值，该方法有效的关键是中间点的选取，可取如下条件：

$$l_0 = l_1 + l_2 \quad ① \qquad \frac{l_1}{l_0} = \frac{l_2}{l_1} \quad ②$$ 如下图(a=Xl , b=Xu)



联立①，②得

$$\frac{l_1}{l_1 + l_2} = \frac{l_2}{l_1} \qquad ③$$

去倒数，化简为

$$1 + R = \frac{1}{R}$$

或者

$$R^2 + R - 1 = 0 \qquad ④$$

解得：R = 0.61803...

算法步骤：

1st 选取黄金比 R=0.618，确定两个内点

d = 0.618*(b-a)
x1 = a + d
x2 = b - d

2nd 计算两个内殿的函数值：f(x1),f(x2)

If f(x1)>f(x2), 所以最大值在 x2,x1 和 b 所定义的区间内。这样对新区间而言，上界仍是 b, x2 成为新的下界。 此外 x1 成为新的 x2. 然后重复迭代即可。（反之亦然）

代码实现：

Problem:

求出函数 f(x)=2sin(x)-x^2/10 在[0,4]内的最大值。

```
1  module dat
2   implicit none
3   real*8::x1,x2,xu,xl,d,dc,fx1,fx2
4   endmodule
5
6   program main
7       use dat
8   implicit none
9   integer::i
10  print *,"Welcome !"
11  print *,"Input the range of the domain!"
12  print*,"Left"
13  read *,xl
14  print*,"Right"
15  read*,xu
16      dc=(5**0.5-1)/2
17          d=dc*(xu-xl)
18          x1=xl+d
19          x2=xu-d
20          call cal(x1,fx1)
21          call cal(x2,fx2)
22  do i=1,8,1
23        print *,"The x1 , x2 is ",x1,x2
24      x1=xl+d
25          x2=xu-d
26          call cal(x1,fx1)
27          call cal(x2,fx2)
28       if (fx1>fx2) then
29          xl=x2
```

```
30          x2=x1
31          d=dc*(xu-xl)
32          x1=xu-d
33      else
34          xu=x1
35          x1=x2
36          d=dc*(xu-xl)
37          x2=xl+d
38          endif
39
40  enddo
41  print *,"The f(x1),f(x2) is ",fx1,fx2
42        print *,"x1,x2",x1,x2
43  endprogram
44
45  subroutine cal(x,fx)
46      use dat
47      implicit none
48      real*8::x,fx
49      fx=2*sin(x)-(x**2)/10
50      return
51      endsubroutine
```

运行结果：

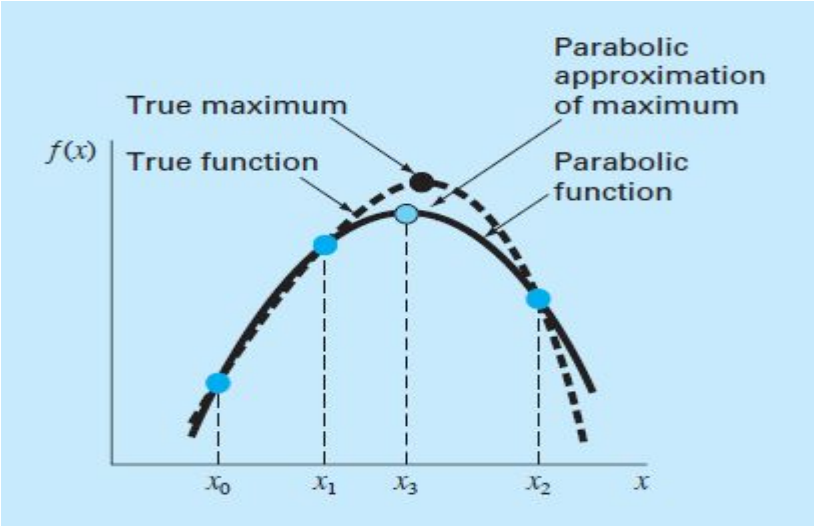
```
C:\Users\Lx\Documents\computational physics works\opt>a
 Welcome !
 Input the range of the domain!
 Left
0
 Right
4
 The x1, x2 is     2.4721360206604004          1.5278639793395996
 The x1, x2 is     1.5278639793395996          1.5278641261616599
 The x1, x2 is     1.5278640354206341          1.5278641261616599
 The x1, x2 is     1.5278640354206341          1.5278640915016735
 The x1, x2 is     1.5278640568416839          1.5278640915016735
 The x1, x2 is     1.5278640568416839          1.5278640782627362
 The x1, x2 is     1.4427191101052343          1.4427191233441730
 The x1, x2 is     1.4427191151620586          1.4427191233441730
 The x1, x2 is     1.4427191151620586          1.4427191202188834
 The x1, x2 is     1.4226191181900683          1.4226191213153578
 The f(x1), f(x2) is     1.7756991438852696          1.7753976732056735
 x1,x2    1.4226191193838225          1.4226191213153578
```

经过 10 次迭代可以看到结果收敛于 x =1.4226 处。

2.二次插值法

原理： 二次插值法利用了二次多项式可以在最优点附近较好的逼近函数 f(x)的形状这一性质。（如下图）

三点插值决定了一个二次多项式或抛物线。因此，弱连接三点能界定一个最优值点，便可以利用这三点拟合一条抛物线，然后对其进行微分计算，将其结果置 0，解方程，便得到最优点 x 的一个估计值。



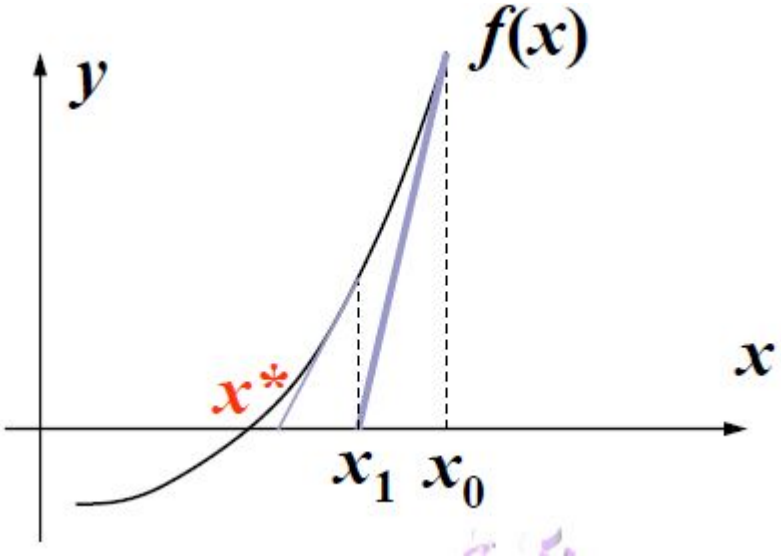
通过计算可得：

$$x_3 = \frac{f(x_0)(x_1^2 - x_2^2) + f(x_1)(x_2^2 - x_0^2) + f(x_2)(x_0^2 - x_1^2)}{2f(x_0)(x_1 - x_2) + 2f(x_1)(x_2 - x_0) + 2f(x_2)(x_0 - x_1)}$$

其中，x0,x1,x2 为初始估计值，而 x3 是由前三点构造的抛物线达到的最大值 x. 生成新点后,不断迭代即可。

3.**牛顿法**

原理：类似方程求根中的牛顿迭代法，在函数某一点绘制一条与函数相切的切线，该切线与 x 轴的交点通常表示根的新估计值。



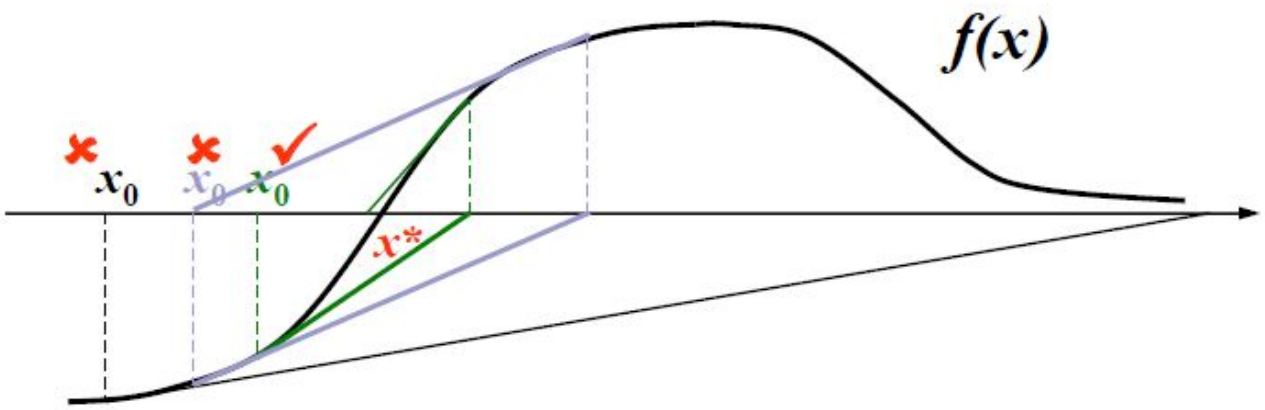
可以利用一种类似的开方法来寻找 f(x)的最优值：定义一个新的函数 g(x)=f ' (x), 由于最优值 x*也满足：

$$f'(x^*) = g(x^*) = 0$$

可以得出：

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

可以作为寻找 f(x)最小值或最大值的方法。但是该方法和牛顿迭代法同样有一个缺点，它可能不收敛。



算法步骤：

由 $$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$ 直接进行迭代。

Problem:

用牛顿法求函数 f(x)=2sin(x)-x^2/10 的最大值。 初始值估计为 2.5

代码实现：

```
module dat
   implicit none
   real*8::x1,x2,fx1,fx2,fx
   endmodule

program main
  use dat
  implicit none
  integer ::i
  x1=2.5
  do i=1,7,1
    call cal1(x1,fx1)
    call cal2(x1,fx2)
     call  cal(x1,fx)
     print *,"************"
    print*,"x,fx",x1,fx
    x2=x1-fx1/fx2
    x1=x2
    !call  cal(x1,fx)
    !print *,"************"
    !print*,x1,fx,fx1,fx2
    enddo

  end

  subroutine cal1(x,y)
    use dat
    implicit none
    real*8::x,y
    y=2*cos(x)-x/5
    endsubroutine
  subroutine cal2(x,y)
    use dat
    implicit none
    real*8::x,y
```

```
36      y=-2*sin(x)-0.2! 之前减1/5时有误差，数据类型!
37      endsubroutine
38    subroutine cal(x,y)
39      use dat
40      implicit none
41      real*8::x,y
42      y=2*sin(x)-(x**2)/10
43      return
44      endsubroutine
```

运行结果：

```
C:\Users\Lx\Documents\computational physics works\opt>a
 ************
x       2.5000000000000000      fx       0.57194428820791310
 ************
x      0.99508155453657687      fx        1.5785880101038243
 ************
x       1.4690107511249952      fx        1.7738493794622938
 ************
x       1.4276423210688058      fx        1.7757256442136429
 ************
x       1.4275517793014381      fx        1.7757256531474153
 ************
x       1.4275517787645942      fx        1.7757256531474153
 ************
x       1.4275517787645942      fx        1.7757256531474153
 ************
```

经过五次迭代，结果很快收敛到真值.

PART II 多维无约束最优化问题

1.引言：

单变量函数的直观图形像是过山车，而二维变量函数的图形则变成了山脉和河谷，对于更多变量的函数，得到合适的图形是不太可能的。面对多维无约束最优化问题，我们可以用是否需要计算导数来分类，不需要计算导数的方法称为 非梯度法或直接法。需要计算导数的方法称为梯度法或上升（下降法）。

2.1 直接法

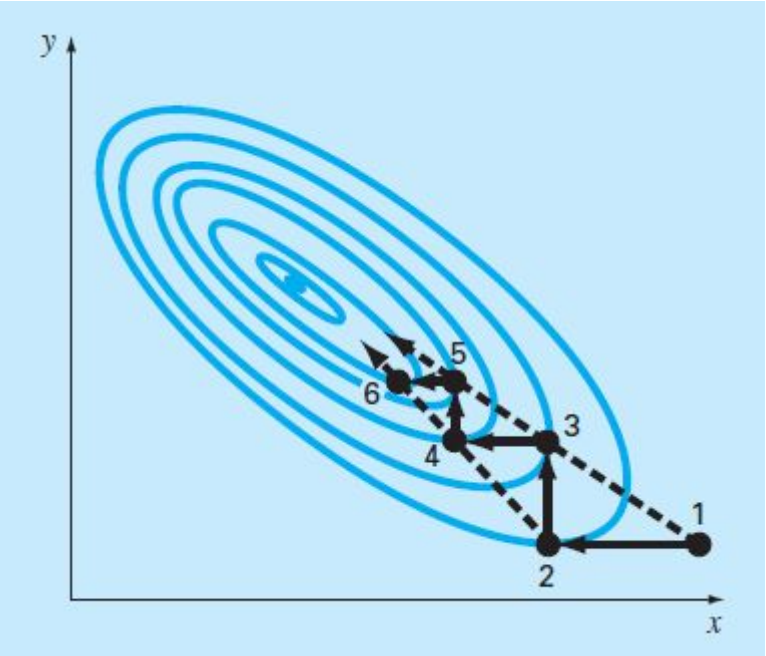
随机搜索法：重复计算函数在随机挑选的自变量处的值，通过一定计算追踪到真的最大值，这种简单强制的方法对不连续和不可微的函数同样有效。而且，通常可以找到全局最优而不是局部最优。但是主要的缺点就是随着自变量数目增多，所需要计算时间代价很大，效率不高。

2.2 单变量和模式检索

原理：

单变量的检索方法的基本思想时每次只改变一个变量的值来改进近似值，而其他的变量保持不变，由于只有一个变量的值被改变，所以问题便简化成一维情况，由此可以通过多种方法来解决。

如图：



用图形来描述单变量搜索方法，如图，从点 1 开始，保持 y 不变，沿着 x 轴搜索到一个最大值 2。保持 x 不变，沿着 y 轴移动到点 3，然后是 4,5,6。按照这样的步骤重复下去，最终是可以逼近最大值的，但是这种搜索方法的效率较低。

但是，有一个性质，即方向相同但起始点不同的一维检索获得的最大值点的连线直接指向全局最大值，这条线称为（共轭方向 Conjugate Direction）。

3.梯度法

梯度：向量，当某一函数在某点处沿着该方向的方向导数取得该点处的最大值，即函数在该点处沿方向变化最快，变化率最大（为该梯度的模）。

$$\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j}$$

表示函数 f(x,y)在某一点处的方向倒数.

多维情况：

$$\nabla f(\mathbf{x}) = \begin{Bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f}{\partial x_2}(\mathbf{x}) \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{Bmatrix}$$

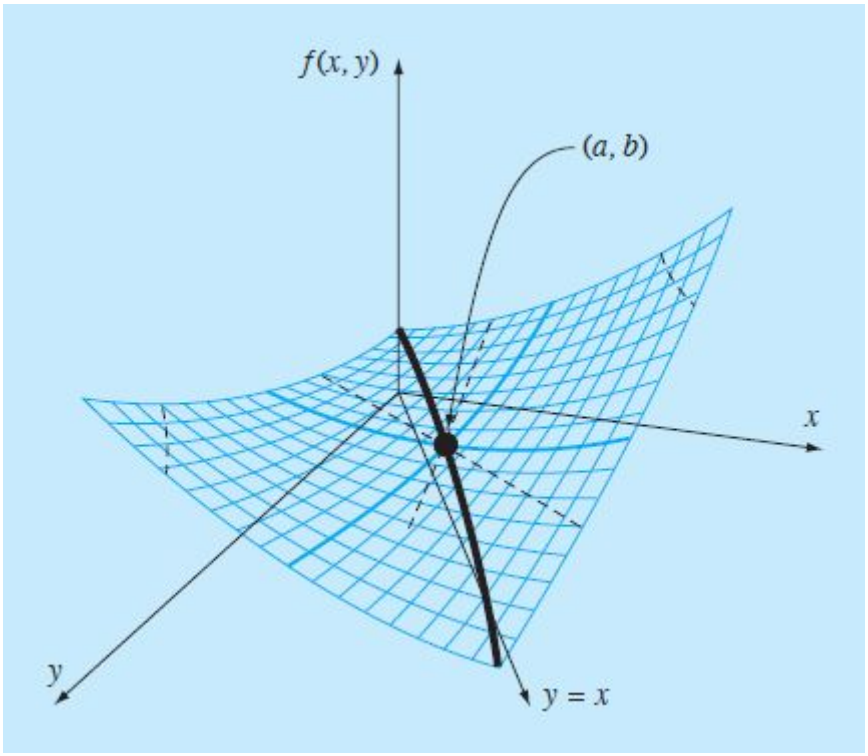
为探究函数最速上升（下降）路线，我们可以选取起始点，求出其偏导数，得到梯度。沿着梯度方向的路径即最速上升路径，每前进一单位距离上升的高度（变化率）为该处梯度的膜。即可以，没行进一步，更新一次梯度，沿着新的方向前进，不断反复即可到达最大值点。除了用来定义最速上升路径外，一阶导数也可以用来辨别是否达到了一个最优值，和一维函数相似，若关于 x 和 y 的偏导数都为 0，便达到了二维最优值。

最优值判据：

1）一阶导数提供了函数耳朵最陡路径，且可以表明是否达到了一个最优值

2）在一维情况下，二阶导数会表明这是一个最大值（f ’’(x)<0）,还是一个最小值（f ’’(x)>0）。但在二维乃至多维的情况下，这并不成立。

如图：



图中鞍点（a,b）看起来是一个最小值，在两个方向上，二阶偏导数都为正。然而，沿着直线 y=x 观察时，该点为最大值点。实际该点既不是最大值也不是最小值。

因此，必须引入新的判据：

定义 Hessian 矩阵：（二维函数）

$$H = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

判据：

$$|H| = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2$$

Three cases can occur

- If $|H| > 0$ and $\partial^2 f/\partial x^2 > 0$, then $f(x, y)$ has a local minimum.
- If $|H| > 0$ and $\partial^2 f/\partial x^2 < 0$, then $f(x, y)$ has a local maximum.
- If $|H| < 0$, then $f(x, y)$ has a saddle point.

3.1 最优梯度法

最优梯度法是应用目标函数的负梯度方向作为每一步迭代的搜索方向。因为每一步都取梯度方向的最优步长，故称为最优梯度法。该方法在一百五十年前由柯西提出。又因为该古老的数值解法在头几步下降最快，又称之为最速下降（上升）法。

步骤：

1st 设置初始点 X0,和迭代精度。

2nd 通过目标函数计算$\nabla f(X^0)$,$S_0=-\frac{\nabla f(X^0)}{\left\|\nabla f(X^0)\right\|}$ 和 $a_k=\frac{-\nabla^T f(X)S_0}{S_0^T H S_0}$.

3rd 判断梯度的膜是否小于迭代精度，如果小于则结束，解为 X0

4th 如果不小于，则 $X^1=X^0+\alpha_0 S_0$

5th 迭代执行步骤 1.

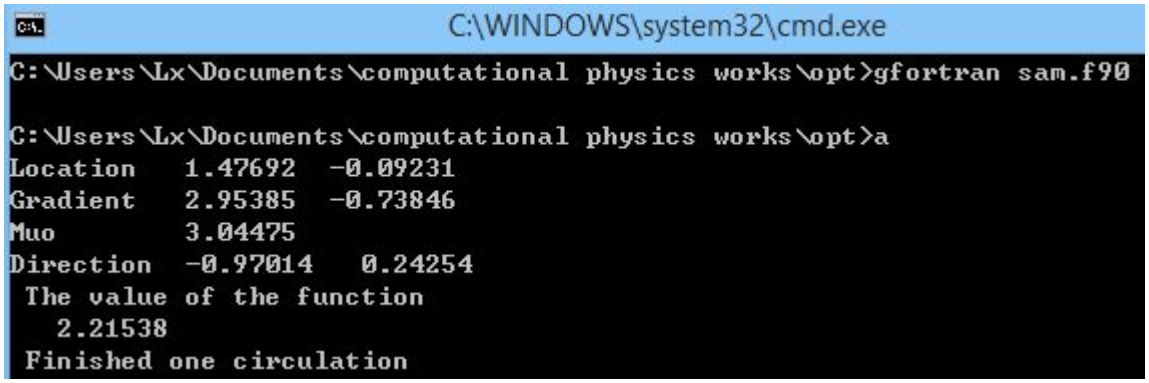
代码：

```
module dat1
    implicit none
    real*8,dimension(:),allocatable,save::Gra,Sk,X1,X0
    real*8,dimension(:,:),allocatable,save::Hess
    real*8::Alpha,mGra,fx
    endmodule

program main
    use dat1
    implicit none
    integer::i,pau
    allocate (Hess(1:2,1:2))
    allocate(Gra(1:2),Sk(1:2),X1(1:2),X0(1:2))!Gradient,
    X0=[2,2]
    call G(X0,Gra,mGra,Sk,Hess,Alpha)
    do i=1,10,1
         if (mGra<0.01) exit
         X1=X0+Alpha*Sk
         call G(X1,Gra,mGra,Sk,Hess,Alpha)
         X0=X1
         fx=x0(1)**2+4*x0(2)**2
         10 format(5f10.5)
         write (*,'("Location",$)');write(*,10)(X0);
         write(*,'("Gradient",$)');write(*,10)(gra);
         write(*,'("Muo     ",$)');write(*,10)(mgra);
         write(*,'("Direction",$)');write(*,10)(sk)
         print*,"The value of the function";write(*,10)(fx)
         print*,"Finished one circulation "
         write(*,*)
     enddo
endprogram

subroutine G(x,y,c,s,He,A)
    use dat1
    implicit none
    real*8::C,A
    real*8,dimension (1:2)::x,y,s
    real*8,dimension(1:2,1:2)::He
    y(1)=2*x(1)
    y(2)=8*x(2)
    c=(y(1)**2+y(2)**2)**(0.5)
    S=-1.0/c*(y)
    He(1,1)=2
    He(1,2)=0
    He(2,1)=0
    He(2,2)=8
    A=-1.0*(dot_product(y,S)/(dot_product(Matmul(S,He),S)))
    return
endsubroutine

```

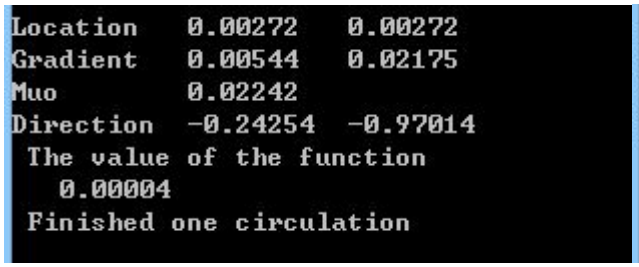
运行结果：

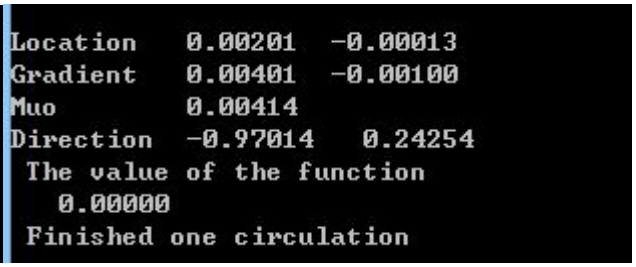
起始点：[2,2]

第二点



.....





最后两步，收敛在[0,0]点，一共用了七步。

总结最优梯度法：

负梯度方向的意义是值函数在这一点周围的最速下降方向，是一个局部的性质，不是全局。因此，延负梯度方向寻优的最优梯度法，其搜索路径实际上是呈直角的锯齿形前进，开始步长较大，越接近极小点，步长越小，收敛速度越慢。所以最优梯度法的最要特点为能较快的接近于最优点的邻近区域范围。

3.2Fletcher－Reeves（共轭梯度法）P383

3.3 牛顿法

牛顿法，又称二阶梯度法。它不带利用了目标函数在搜索点的梯度，而且还利用了目标函数的二阶导数，考虑了梯度变化的趋势。因此，比最优梯度法及共轭梯度法都有所改进，能更快的搜索出最优点。

迭代公式：

$$X^{k+1} = X^k - H_k^{-1}\nabla f(X^k)$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

步骤：

1st 设置初始点 X0,和迭代精度。

2nd 通过目标函数计算 $\nabla f(X^0)$，$\left\|\nabla f(X^0)\right\|$

3rd 判断梯度的膜是否小于迭代精度，如果小于则结束，解为 X0

4th 如果不小于，则计算 Hessian Matrix.和 The inverse matrix of Hessian Matrix (缺公式)

5th 通过 $X^{k+1} = X^k - H_k^{-1}\nabla f(X^k)$，生成新的坐标

6th 迭代执行步骤 1.

问题：

解函数，$Z = x^2 + y^2 + xy^2 + yx^2$ 的最优点。（同时还测试了 $Z = x^2 + 4y^2$ ）

代码：

```
!牛顿法对我选取的该函数存3个驻点！！有时会收敛到（0，0）
!Location   -1.2361    3.2361
!Location    3.2361   -1.2361
!Location   -0.6667   -0.6667
module dat1
    implicit none
    real,dimension(:,:),allocatable,save::Hess,ivHess
    real,dimension(:),allocatable,save::Gra
    real::fx,mgra
    real,dimension(1:2)::X1,X0
endmodule

program main
    use dat1
    implicit none
    integer::i,j,k
    allocate (Hess(1:2,1:2))
    allocate(Gra(1:2))!Gradient,
    X0=[10,10]!start
    print*," The location of start point :    (",X0(1),",",X0(2),")"
    do i=1,10,1
        call G(X0,Gra,mGra,Hess)
        write(*,'("Hess :",1x)');write(*,10)(Hess(j,:),j=1,2)
        call nizhen(2)
        print*,"Ok"
         if (mGra<0.00001) exit
         X1=X0-0.5*matmul(ivHess,Gra)
         X0=X1
         fx=x0(1)**2+x0(2)**2+x0(1)*x0(2)**2+x0(2)*x0(1)**2
         fx=-fx
         10 format(2f10.4)
```

```
        write (*,'("Location",$)');write(*,10)(X0);
        write(*,'("Gradient",$)');write(*,10)(gra);
        !write(*,'("Muo     ",$)');write(*,10)(mgra);
        print*,"The value of the function";write(*,10)(fx)
        print*,"Finished one circulation "
        k=i
    enddo
    print*,k," step"
endprogram

subroutine G(x,Gx,mg,he)
!璇
   use dat1
   implicit none
   real::mg
   real,dimension (1:2)::x,Gx
   real,dimension(1:2,1:2)::He
   He(1,1)=1+x(2)! Input the Hessian metrix
   He(1,2)=x(1)+x(2)
   He(2,1)=He(1,2)
   He(2,2)=1+x(1)
   He=-He!transfer the minus
   Gx(1)=2*x(1)+2*x(2)*x(1)+x(2)**2
   Gx(2)=2*x(2)+2*x(2)*x(1)+x(1)**2
   Gx=-Gx
   mg=(Gx(1)**2+Gx(2)**2)**(0.5)
endsubroutine

! aa涓哄師鐭╅樀锛宎涓哄瓙鍒綍a鐨勫€嗙煩闃碉紝n涓虹煩闃靛a鐨勮淮鏁?
subroutine nizhen(n)
  use dat1
  implicit none
  integer::n,i,j,k
  real,dimension(1:2,1:2)::b,a
  a=Hess
  b=0
  do i=1,n,1
    b(i,i)=1
  enddo
  do i=1,n,1
    b(i,1)=b(i,1)/a(i,i)
    b(i,2)=b(i,2)/a(i,i)
    a(i,i:n)=a(i,i:n)/a(i,i)
   do j=i+1,n,1
     do k=1,n,1
   b(j,k)=b(j,k)-b(i,k)*a(j,i)
  enddo
  a(j,i:n)=a(j,i:n)-a(i,i:n)*a(j,i)
 enddo
  enddo
  do i=n,1,-1
    do j=i-1,1,-1
   do k=1,n
     b(j,k)=b(j,k)-b(i,k)*a(j,i)
    enddo
  enddo
   enddo
   10 format (2f10.5)
   ivhess=b
   write(*,'("The inversemetrix is :",1x)')
   write(*,10)(ivhess(i,:),i=1,2)
end
```

运行结果：

一共测试了两个函数，对于一般的二次函数（例：$Z=x^2+4y^2$），牛顿法可以一步达到极值点。对于非二次函数，$Z=x^2+y^2+xy^2+yx^2$，遇到了类似一维求根牛顿下山法中，左右折叠，两点之间反复，发散，等多种情况。有些情况可以通过改变步长，如本例，我将步长缩为之前的 0.5 倍(8 steps 达到迭代精度)，0.1(81 steps),0.2 (37 steps),0.3(22 steps), 0.4(14 steps),0.6(10 steps), 0.7(13 steps), 0.8(22 steps),0.9 倍(51 steps...), 0.99(557 steps)。由上可见，步长的控制对牛顿法的影响是显著的。

```
C:\Users\Lx\Desktop\实习需要用到的文件>a
  The location of start point :    (    10.0000000     ,    10.0000000     )
Hess :
  -11.0000  -20.0000
  -20.0000  -11.0000
The inversemetrix is :
   0.03943  -0.07168
  -0.07168   0.03943
 Ok
Location    4.8387    4.8387
Gradient -320.0000 -320.0000
 The value of the function
 -273.4047
 Finished one circulation
```

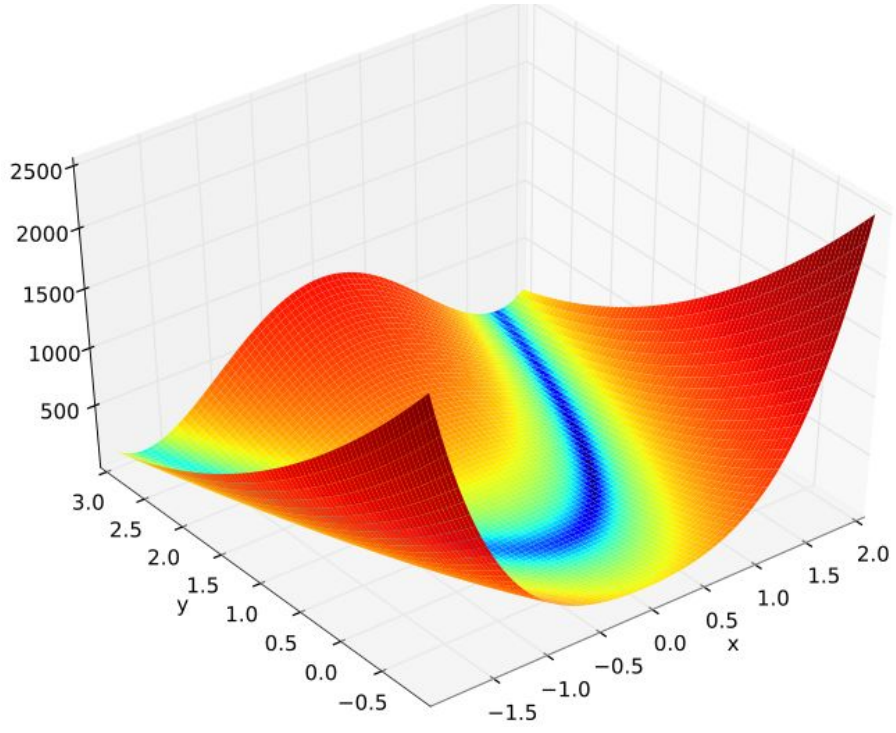
.....

```
Location    0.0000    0.0000
Gradient   -0.0003   -0.0003
 The value of the function
   -0.0000
 Finished one circulation
Hess :
   -1.0000   -0.0000
   -0.0000   -1.0000
The inversemetrix is :
  -1.00000   0.00000
   0.00000  -1.00000
 Ok
          8  step
```

Rosenbrock 函数:

翻阅文献，看到在数学优化中，一个用来测试优化算法性能的非凸函数，由 Howard Harry Rosenbrock 在1960年提出。也称为 Rosenbrock 山谷或 Rosenbrock 香蕉函数，也简称为香蕉函数。

Rosenbrock 函数的定义如下：$f(x,y) = (1-x)^2 + 100(y-x^2)^2.$



Rosenbrock 函数的每个等高线大致呈抛物线形，其全局最小值也位在抛物线形的山谷中（香蕉型山谷）。很容易找到这个山谷，但由于山谷内的值变化不大，要找到全局的最小值相当困难。

于是将 Rosenbrock 函数，代入算法测试，设置当梯度模$\varepsilon<10^{-4}$时，迭代结束，起始点[-1.2,1.0]。

（只是将前面代码里面的函数部分进行替换即可）

举例：

```
1
2  module dat1
3      implicit none
4      real,dimension(:,:),allocatable,save::Hess,ivHess
5      real,dimension(:),allocatable,save::Gra
6      real::fx,mgra
7      real,dimension(1:2)::X1,X0
8  endmodule
9
10 program main
11     use dat1
12     implicit none
13     integer::i,j,k
14     allocate (Hess(1:2,1:2))
15     allocate(Gra(1:2))!Gradient,
16     X0=[-1.2,1.0]!start
17     fx=100*(x0(2)-x0(1)**2)**2+(1-x0(1))**2
18     print*," The location of start point :    (",X0(1),",",X0(2),")"
19     print*,"The original value of fx",fx
20     do i=1,10000,1
21         call G(X0,Gra,mGra,Hess)
22         write(*,'("Hess :",1x)');write(*,10)(Hess(j,:),j=1,2)
23         call nizhen(2)
24          if (mGra<0.0001) exit
25          X1=X0-matmul(ivHess,Gra)
26          X0=X1
27          fx=100*(x0(2)-x0(1)**2)**2+(1-x0(1))**2
28          !fx=-fx
29          10 format(2f10.4)
30          write (*,'("Location",$)');write(*,10)(X0);
31          write(*,'("Gradient",$)');write(*,10)(gra);
32          !write(*,'("Muo     ",$)');write(*,10)(mgra);
33          print*,"The value of the function";write(*,10)(fx)
34          print*,"Finished one circulation "
35          k=i
36      enddo
37      print*,"The value of the function",fx
38      write(*,'("Gradient",$)');write(*,10)(gra);
39      print*,k," step"
40 endprogram
41
42 subroutine G(x,Gx,mg,he)
43 !该步通过输入坐标X，计算梯度Gx，模mg，Hessian metrix (He)
44     use dat1
45     implicit none
46     real::mg
47     real,dimension (1:2)::x,Gx
48     real,dimension(1:2,1:2)::He
49     He(1,1)=-400*x(1)+1200*x(1)**2+2! Input the Hessian metrix
50     He(1,2)=-400*x(1)
51     He(2,1)=He(1,2)
52     He(2,2)=200
53     !He=-He!transfer the minus
54     Gx(1)=-400*x(1)*(x(2)-x(1)**2)-2+2*x(1)
55     Gx(2)=200*(x(2)-x(1)**2)
56     !Gx=-Gx
57     mg=(Gx(1)**2+Gx(2)**2)**(0.5)
58 endsubroutine
59
60 ! aa为原矩阵，b为存放aa的逆矩阵，n为矩阵aa的维数
61 subroutine nizhen(n)
62   use dat1
63   implicit none
64   integer n,i,j,k
65   real,dimension(1:2,1:2)::b,a
66   a=Hess
```

```
67    b=0
68    do i=1,n
69      b(i,i)=1
70    enddo
71    do i=1,n
72      b(i,:)=b(i,:)/a(i,i)
73      a(i,i:n)=a(i,i:n)/a(i,i)
74     do j=i+1,n
75       do k=1,n
76     b(j,k)=b(j,k)-b(i,k)*a(j,i)
77    enddo
78    a(j,i:n)=a(j,i:n)-a(i,i:n)*a(j,i)
79   enddo
80    enddo
81    do i=n,1,-1
82      do j=i-1,1,-1
83     do k=1,n
84       b(j,k)=b(j,k)-b(i,k)*a(j,i)
85      enddo
86    enddo
87     enddo
88     10 format (2f10.5)
89     ivhess=b
90     write(*,'("The inversemetrix is :",1x)')
91     write(*,10)(ivhess(i,:),i=1,2)
92   end
```

最优梯度法：

```
  The location of point :    (  0.99989396017572862      ,  0.99978757725265632
     )
Location    0.99989    0.99979
Gradient   -0.00007   -0.00007
Muo         0.00010
Direction    0.70454    0.70966
 The value of the function
   0.00000
 Finished one circulation
        3359 steps
```

需要 3359 steps.

牛顿法：

起始点（-1.2,1.0)

```
Hess :
  802.0026 -400.0005
 -400.0005  200.0000
The inversemetrix is :
   0.49986    0.99971
   0.99971    2.00443
 The value of the function   1.71951342E-12
Gradient     0.0000    -0.0000
         667  step
```

起始点 (-1,1)

```
Hess :
  802.0005 -400.0001
 -400.0001  200.0000
The inversemetrix is :
   0.49997    0.99993
   0.99993    2.00487
 The value of the function   5.68434189E-14
Gradient     0.0000    -0.0000
         285  step
```

起始点（10,1)

```
Hess :
  802.0253 -400.0051
 -400.0051  200.0000
The inversemetrix is :
   0.49875   0.99751
   0.99751   2.00005
 The value of the function   1.61098065E-10
Gradient    0.0001   -0.0000
        9911  step

C:\Users\Lx\Desktop\实习需要用到的文件>
```

当初始点选取为（-100.2，1) 随便取的，在(-1.2,-1.0) 基础上加了 2 个 0.
运行结果：

```
The inversemetrix is :
   0.00000  -0.00005
  -0.00005   0.01490
Location  -99.6660 9933.3105
Gradient -179.9889    0.1071
 The value of the function
10133.6426
 Finished one circulation
Hess :
**********39866.3984
39866.3984  200.0000
The inversemetrix is :
   0.00000  -0.00005
  -0.00005   0.01490
Location  -99.6659 9933.2998
Gradient -183.8442    0.0877
 The value of the function
10133.6318
 Finished one circulation
 The value of the function   10133.6318
Gradient -183.8442    0.0877
       10000  step
```

从左图可以看出，在 10000 步左右的时候，位置坐标已经从（-100.2,-1.0) 变成，（-99.6660,9933.3105) 并且有些数值早已经溢出，这是一个发散的情况。
（循环控制在 10000 步内是为了节省时间， 足以说明问题即可。）

总结：

相对来说牛顿法速度快于最优梯度法,因为香蕉函数的性质（其全局最小值也位在抛物线形的山谷中（香蕉型山谷）。很容易找到这个山谷，但由于山谷内的值变化不大，要找到全局的最小值相当困难。），其搜索路径实际上是呈直角的锯齿形前进，开始步长较大，越接近极小点，步长越小，收敛速度越慢。所以最优梯度法的最要特点为能较快的接近于最优点的邻近区域范围，但是之后收敛速度明显下降。

对于牛顿法单变量推广到多变量的情况， 因为牛顿法相对来说不但利用了目标函数在搜索点的梯度，而且还利用了目标函数的二阶导数，考虑了梯度变化的趋势。因此，比最优梯度法及共轭梯度法都有所改进，能更快的搜索出最优点。

但是，由于每次迭代时都需要计算二阶导数和矩阵求逆，因此这种方法不适用于变量很多的函数。而且，牛顿法正如一维中的方程求根一样，初始点的不同会导致算法的稳定性的变化，若起点不再最优值附近，牛顿法有可能发散。(如上图例子）

最优梯度法：主要特点为能较快的接近于最优点的邻近区域范围

牛顿法：利用了目标函数在搜索点的梯度和二阶导数信息，考虑了梯度变化的趋势。因此，比最优梯度法及共轭梯度法都有所改进，能更快的搜索出最优点。

结语：结合最优梯度法和牛顿法各自的优缺点，我们就可以构造出更为优秀的算法，从而可以较好的解决最优化问题。

PART III
准牛顿法

由于直接最优梯度法和牛顿法各自的优缺点，提出了拟牛顿法，也称为变度量法（Variable-Metric Method). 是类似牛顿法的方式试图估计到达最优值的直线路径。然而，由于何塞矩阵的 f 的二阶导数在每步都发生变化，所以拟牛顿法试图利用仅由 f 的一阶偏导数组成矩阵 A 去逼近 H。这种类型主要有两种方法： Davidon-Fletcher-Powell (DFP) 和 Broyden-Fletcher-Goldfarb-Shanno (BFGS)。

下来，我们重点讨论下 BFGS 算法，BFGS 算法 是以其发明者 Broyden, Fletcher, Goldfarb 和 Shanno 的姓氏首字母命名。

由上文结合（3.1）知：

最优梯度法的搜索方向为 $\mathrm{P}^{\mathrm{k}}=-\nabla f(X^k)$.，迭代公式为 $\mathrm{X}^{k+1}=X^k+\alpha_k P^k$，缺点是收敛慢；

牛顿法搜索方向为 $P_k=-H_k^{-1}\nabla f(X^k)$,迭代公式 $X^{k+1}=X^k-H_k^{-1}\nabla f(X^k)$，缺点是计算麻烦

于是提出一个 $n\times n$ 的对称矩阵 H' 去逼近 H_k^{-1},其搜索方向与牛顿法相似，搜索方向为 $\mathrm{P}^{\mathrm{k}}=-H'\nabla f(X^k)$，迭代公式为 $\mathrm{X}^{k+1}=X^k+\lambda_k P_k$。

原理：尺度矩阵 H'的构造

变尺度法是改变尺度矩阵 H'，将其从一开始的单位矩阵过渡到 Hessian 矩阵的逆 H^{-1}, 这个过程不断改变搜索方向，所以又称 H' 为该方法下的方向矩阵。

该矩阵的构造要求是：1.逼近 H^{-1}; 2.由上一步的 $H^{'}$ 转化而来 ; 3. $n\times n$ 的对称矩阵 H'

1)因为梯度 g 的增量 = 梯度的导数*自变量增量，表达式为 $\Delta g=H*\Delta X$,同乘矩阵的逆的得

$$H^{-1}\Delta g=\Delta X \qquad ①$$

2)为使 $H^{'}\approx H^{-1}$,令 $H^{'}\Delta g^k=\Delta X^k$, 其中（$\Delta X^k=X^{k+1}-X^k,\Delta g^k=\nabla f(X^{k+1})-\nabla f(X^k)$ ）

3)为使 H'^{k+1} 由 H^k 转化而来，取 $H'^{k+1}=H'^k+\Delta H'$ ②

将②代入①，有 $(H'^k+\Delta H'^k)\Delta g^k=\Delta X^k$

即：$H'^k\Delta g^k+\Delta H'^k\Delta g^k=\Delta X^k$,同时令 $Z^k=H^k\Delta g^k$

则有： $\Delta H'^k\Delta g^k=\Delta X^k-Z^k$ ③

4） 最后一条为了保证构造矩阵为 $n\times n$ 的对称矩阵 H'，将③式 $\Delta H'^k=\Delta X^k\Delta g^{k-1}-Z^k\Delta g^{k-1}$，可以看成是以 ΔX和Z 为元素构造的两个对称n×n矩阵基 可以写成如下形式：

$$\Delta H'^k = B^k \Delta X^k (\Delta X^k)^T - C^k Z^k (Z^k)^T \qquad ④$$

式中 B，C 为待定参量。也是我们构造矩阵中需要求得的参数。将④带入到③

有 $$\Delta H'^k \Delta g^k = B^k \Delta X^k (\Delta X^k)^T \Delta g^k - C^k Z^k (Z^k)^T \Delta g^k = \Delta X^k - Z^k \qquad ⑤$$

合并同类项，可得：$\begin{matrix} B^k \Delta X^k (\Delta X^k)^T \Delta g^k = \Delta X^k \\ C^k Z^k (Z^k)^T \Delta g^k = Z^k \end{matrix}$，以此成矩阵的逆最终得：

$$\begin{aligned} B^k &= \frac{1}{(\Delta X^k)^T \Delta g^k} \\ C^k &= \frac{1}{(Z^k)^T \Delta g^k} \end{aligned} \qquad ⑥$$

综上，我们可以得到递推公式：

$$H'^{k+1} = H'^k + \Delta H' = H'^k + B^k \Delta X^k (\Delta X^k)^T - C^k Z^k (Z^k)^T$$

步骤：

1st 设置初始点 X0,和迭代精度。

2nd 第一步通过最优梯度法计算出迭代方向 $\mathrm{P}^k = -\nabla f(X^k)$，$H^0 = I$，然后在该方向上进行一维搜索，$\mathrm{X}^{k+1} = X^k + \lambda_k P_k$ 求出最优参数 λ_k 对应一维搜索极小点。

3rd 上一步最优梯度法搜索结束后，由牛顿法，生成下一步搜索方向 $P_k = -H_k^{-1} \nabla f(X^k)$ 然后进行一维搜索，$\mathrm{X}^{k+1} = X^k + \lambda_k P_k$，求出最优参数 λ_k 对应一维搜索极小点。

4th 判断新的位置的梯度的膜是否小于迭代精度，如果小于则结束，解为 X^{k+1}

5th 如果不小于迭代精度，递推继续，由类牛顿法的原理部分阐述，结合⑥和递推公式

$H'^{k+1} = H'^k + \Delta H' = H'^k + B^k \Delta X^k (\Delta X^k)^T - C^k Z^k (Z^k)^T$，更新生成 H'^{k+1}。

6th 用更新的 H'^{k+1} 代入类牛顿法（第 3rd 步骤）生成新的搜索方向：$P_k = -H_k^{-1} \nabla f(X^k)$。返回到步骤 3rd.

7th 最终迭代到收敛精度内，然后结束，解为 X^{k+1}。

代码部分：

第一部分为简单的一个函数 $f(x,y) = x^2 + 2xy + 2y^2$,二次型函数的实现。

```
!一般函数的算法 函数形式： y= x1^2+2*X1*X2+2*X2^2
module dat1
    implicit none
    real,dimension(:,:),allocatable,save::Hess0,Hess1,BHess,Chess
    real,dimension(:),allocatable,save::Gra0,Gra1,deltaG,sk
    real::fx,mgra
    real,dimension(1:2)::pX1,pX0,deltaX
endmodule

program main
```

```
    use dat1
    implicit none
    real::alpha
    integer::i,j,k
    allocate (Hess0(1:2,1:2),Hess1(1:2,1:2),BHess(1:2,1:2),CHess(1:2,1:2))
    allocate(Gra0(1:2),Gra1(1:2),Sk(1:2))!Gradient
    pX0=[1.0,-2.0]!start
    fx=px0(1)**2+2*px0(1)*px0(2)+2*px0(2)**2
    print*," The location of start point :    (",pX0(1),",",pX0(2),")"
    print*,"The original value of fx",fx
    call setup(px0,gra0,Hess0,alpha,Sk)
    px1=px0+alpha*Sk

do i=1,10,1
    call dfx(Px1,Gra1,mgra)
    if (mGra<0.0001) exit
    call G(pX0,pX1,gra0,Gra1,hess0,hess1)
    call sam1(px1,gra1,hess1,alpha,sk,mgra)
    px0=px1
    gra0=gra1
    hess0=hess1
    px1=px0+alpha*sk
    k=i
    10 format(2f10.5)
    write (*,'("Location",$)');write(*,10)(pX1);
    write(*,'("Muo     ",$)');write(*,10)(mgra);
    print*,"The value of the function";write(*,10)(fx)
    print*,"Finished one circulation "
enddo
    print*,k," step"
endprogram

subroutine sam1(x,Gx,he,A,S,mg)
    !该步为最优梯度法部分，求步长和方向向量
    use dat1
    implicit none
    real::mg,A
    real,dimension (1:2)::x,S,Gx
    real,dimension(1:2,1:2)::He,he1
    call dfx(x,Gx,mg)
    S=-(Matmul(He,Gx))
    He1(1,:)=(/2.0,2.0/)
    He1(2,:)=(/2.0,4.0/)
    A=-1.0*(dot_product(Gx,S)/(dot_product(Matmul(S,He1),S)))
endsubroutine

subroutine G(x0,x1,Gx0,gx1,he0,he1)
!该步通过输入坐标X，计算梯度Gx，模mg，更新Hessian（即用前一个H逼近生成新的H）
    use dat1
    implicit none
    real::A,B,C,mg
    integer ::j
    real,dimension (1:2)::x0,x1,Gx0,Gx1,dx,dG,Z
    real,dimension(1:2,1:2)::He0,He1,HeB,HeC
    call dfx(x1,gx1,mg)
    Dx=x1-x0
    Dg=gx1-gx0
    Z=Matmul(He0,dG)
    B=1.0/(dot_product(dx,dG))
    C=1.0/(dot_product(z,dG))
    call Xcheng(Dx,HeB)
    HeB=HeB*B
    call Xcheng(Z,HeC)
    HeC=HeC*C
    He1=He0+HeB-HeC
endsubroutine

subroutine Xcheng(x,XX)
    !实现一阶二元张量叉乘为二阶张量
    implicit none
    real,dimension(1:2)::x
    real,dimension(1:2,1:2)::XX
    XX(1,1)=x(1)**2
    XX(1,2)=x(1)*x(2)
    xx(2,1)=xx(1,2)
    XX(2,2)=x(2)**2
endsubroutine

subroutine dfx(x,df,m)
```

```
    !求测试函数导数，偏x 为 df(1),偏y为df(2))
    implicit none
    real::m
    real,dimension(1:2)::x,df
    df(1)=2.0*x(1)+2.0*x(2)
    df(2)=2.0*x(1)+4.0*x(2)
    m=(df(1)**2+df(2)**2)**(0.5)
endsubroutine

subroutine setup(x,Gx,he,A,S)
    !起始的第一步采用最优梯度法计算
    use dat1
    implicit none
    real::mg,A
    real,dimension (1:2)::x,S,Gx
    real,dimension(1:2,1:2)::He,He1
    He(1,1)=1! Input the Hessian metrix
    He(1,2)=0
    He(2,1)=He(1,2)
    He(2,2)=1
    call dfx(x,Gx,mg)
    print*,Gx
    S=-(Matmul(He,Gx))
    He1(1,:)=(/2.0,2.0/)
    He1(2,:)=(/2.0,4.0/)
    A=-1.0*(dot_product(Gx,S)/(dot_product(Matmul(S,He1),S)))
endsubroutine
```

运行结果：

```
C:\Users\Lx\Desktop\（无主题）>a
  The location of start point :     (    1.00000000     ,   -2.00000000     )
 The original value of fx    5.00000000
  -2.00000000       -6.00000000
Location   -0.00000   -0.00000
Muo          1.26491
 The value of the function
   5.00000
 Finished one circulation
            1  step
```

起始第一步为最优梯度法，之后的牛顿法只用一步即达优，即总共进行两步实现二次函数的最优化搜索。

代码：对香蕉函数测试的实现（对代码同时进行了模块化改进，增加以后使用的便捷性）

```
!香蕉函数的实现，为了解决一维搜索的发散问题，这个算法实现中，一维搜索求步长引入了Hessian 的信息
module dat1
    implicit none
    real,dimension(:,:),allocatable,save::Hess0,Hess1,BHess,Chess
    real,dimension(:),allocatable,save::Gra0,Gra1,sk
    real::fx,mgra
    real,dimension(1:2)::pX1,pX0,deltaX
endmodule

program main
    use dat1
    implicit none
    real::alpha
    integer::i,j,k
    allocate (Hess0(1:2,1:2),Hess1(1:2,1:2),BHess(1:2,1:2),CHess(1:2,1:2))
    allocate(Gra0(1:2),Gra1(1:2),Sk(1:2))!Gradient
    px0=[-1.2,1.0]!start
    print*," The location of start point :    (",pX0(1),",",pX0(2),")"
    print*,"The original value of fx",fx
    call setup(px0,gra0,Hess0,alpha,Sk)
    px1=px0+alpha*Sk
```

```
22
23  do i=1,1000,1
24      call dfx(Px1,Gra1,mgra)
25      if (mGra<0.0001) exit
26
27      call G(pX0,pX1,gra0,Gra1,hess0,hess1)
28      call sam1(px1,gra1,hess1,alpha,sk,mgra)
29      px0=px1
30      gra0=gra1
31      hess0=hess1
32      px1=px0+alpha*sk
33      k=i
34      10 format(2f10.5)
35      fx=100*(px1(2)-px1(1)**2)**2+(1-px1(1))**2
36      write (*,'("Location",$)');write(*,10)(pX1);
37      write(*,'("Muo     ",$)');write(*,10)(mgra);
38      print*,"The value of the function";write(*,10)(fx)
39      print*,"Finished one circulation "
40  enddo
41  print*,k," step"
42  endprogram
43
44  subroutine sam1(x,Gx,he,A,S,mg)
45      !该步为最优梯度法部分
46      use dat1
47      implicit none
48      real::mg,A
49      real,dimension (1:2)::x,S,Gx
50      real,dimension(1:2,1:2)::He,he0
51      call dfx(x,Gx,mg)
52      S=-(Matmul(He,Gx))
53      call Hessian(X,He0)! 用Hessian矩阵代替
54      A=-(dot_product(Gx,S)/(dot_product(Matmul(S,He0),S)))
55  endsubroutine
56
57  subroutine G(x0,x1,Gx0,gx1,he0,he1)
58      !该步通过输入坐标X，计算梯度Gx，模mg，更新Hessian（即用前一个H逼近生成新的H）
59      use dat1
60      implicit none
61      real::A,B,C,mg
62      integer ::j
63      real,dimension (1:2)::x0,x1,Gx0,Gx1,dx,dG,Z
64      real,dimension(1:2,1:2)::He0,He1,HeB,HeC
65      call dfx(x1,gx1,mg)
66      Dx=x1-x0
67      Dg=gx1-gx0
68      Z=Matmul(He0,dG)
69      B=1.0/(dot_product(dx,dG))
70      C=1.0/(dot_product(z,dG))
71      call Xcheng(Dx,HeB)
72      HeB=HeB*B
73      call Xcheng(Z,HeC)
74      HeC=HeC*C
75      He1=He0+HeB-HeC
76      !10 format(2f10.4)
77      !write(*,'("Hess:")');Write(*,10)(He1(j,:),j=1,2)
78  endsubroutine
79
80  subroutine Xcheng(x,XX)
81      implicit none
82      real,dimension(1:2)::x
83      real,dimension(1:2,1:2)::XX
84      XX(1,1)=x(1)**2
85      XX(1,2)=x(1)*x(2)
86      xx(2,1)=xx(1,2)
87      XX(2,2)=x(2)**2
88  endsubroutine
89
90  subroutine dfx(x,df,m)!求测试函数导数，偏x 为 df(1),偏y为df(2))
91      implicit none
92      real::m
93      real,dimension(1:2)::x,df
94      df(1)=-400*x(1)*(x(2)-x(1)**2)-2+2*x(1)
95      df(2)=200*(x(2)-x(1)**2)
96      m=(df(1)**2+df(2)**2)**(0.5)
97   endsubroutine
98
99  subroutine setup(x,Gx,he,A,S)
100 !该步为最优梯度法部分
```

```
    use dat1
    implicit none
    real::mg,A
    real,dimension (1:2)::x,S,Gx
    real,dimension(1:2,1:2)::He,He1
    He(1,1)=1! Input the Hessian metrix
    He(1,2)=0
    He(2,1)=0
    He(2,2)=1
    call dfx(x,Gx,mg)
    S=-(Matmul(He,Gx))
    call Hessian(x,He1)! 用Hessian矩阵代替
    A=-(dot_product(Gx,S)/(dot_product(Matmul(S,He1),S)))
endsubroutine

subroutine Hessian(x,he)
    implicit none
    real,dimension(1:2)::X
    real,dimension(1:2,1:2)::he
    He(1,1)=-400*x(1)+1200*x(1)**2+2! Input the Hessian metrix
    He(1,2)=-400*x(1)
    He(2,1)=He(1,2)
    He(2,2)=200
endsubroutine
```

起始点（-1.2，1.0）

```
 Finished one circulation
Location    1.00034    1.00067
Muo         0.03711
 The value of the function
   0.00000
 Finished one circulation
Location    1.00002    1.00003
Muo         0.00963
 The value of the function
   0.00000
 Finished one circulation
Location    1.00000    1.00000
Muo         0.00061
 The value of the function
   0.00000
 Finished one circulation
         614  step
```

对香蕉函数，在精度仍为10^{-4}的要求下，类牛顿法迭代了 614 步达优，在与牛顿法的 667 步相比具有一定优势。

起始点（10.0,1.0）

```
 Finished one circulation
Location    1.00001    1.00002
Muo         0.00309
 The value of the function
   0.00000
 Finished one circulation
Location    1.00000    1.00000
Muo         0.00044
 The value of the function
   0.00000
 Finished one circulation
        7704  step
```

当起始点更换为（10,1）时，相比较原先的 9911 步迭代，也有一定的优势。

因为观察迭代中每一个坐标的变化，发现会存在围绕最优点来回折叠行走的现象，于是我想通过改变迭代的步长看看会引入什么变化。以下为部分截图。

```
Location   1.00000   1.00000
Muo        0.00026
 The value of the function
   0.00000
 Finished one circulation
Location   1.00000   1.00000
Muo        0.00013
 The value of the function
   0.00000
 Finished one circulation
        7303  step
```

步长 1.4

```
 The value of the function
   0.00000
 Finished one circulation
Location   1.00000   1.00000
Muo        0.00011
 The value of the function
   0.00000
 Finished one circulation
        6286  step
```

步长 1.5

记录如下：

步长系数	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
迭代次数	7704	7594	7257	6836	7303	6286	6422	6788	6171	>10000

总结：

当步长系数小于 0.8 和大于 1.9 时，会用更多的步数收敛（大于 10000 步），故不作记录。从以上这个结果可以分析，第一步的最优梯度法之后，每一步进行类牛顿法的一维搜索，这并不能保证方向依然是向着最速下降的方向，所以，从全局来看，依然是通过走折线到达最优值，BFGS 方法在求解最优化问题时，需要有改进的地方。

存在的问题和需要改进的地方：

在 BFGS 算法中，一维寻优对象是一个单变量的二元四次函数，无约束，所以按照之前讨论到的一维寻优搜索方式中，只有牛顿法适合求解，但实际测试中，牛顿法发散，故目前没有很好的解决这个问题，实际的一维搜索依然是用到了精确的 Hessian 矩阵的信息，这个步长因为从原理上和一维寻优求出的步长应该是相同的，故先跳过了这个问题，测试了 BFGS 算法整体的效果。

参考资料：

《Numerical Optimization》—Chen Changjun

《Numberical Methods for Engineers》—Steven_Chapra,_Steven_C._Chapra,_Raymond_Canale

《Fortran 77和90/95编程入门》—中国科技大学天文与应用物理系，丁泽军编

《最优化方法应用基础》—卢险峰

《最优化计算原理与算法程序设计》—栗塔山 等编著