1. 引言

我们在中学的时候学过一元二次函数，求解时引入一个求根公式，代入公式就可以得到不同的根，假如想计算一个高次方程的解，我们还能推导出求根公式吗？

伽罗瓦在群论中证实，五次及以上多项式方程没有显示表达的求根公式，但是科学研究（例如行星轨道计算）中还是有很多求解高次方程的真实需求。既然得不到明确的求根公式，我们可以用迭代的办法来不断逼近真实的解。

多项式方程求解的问题实际上可以看成是函数求极值，假如我们对$f'(x)=0$的求解得到驻点，将驻点代入f(x)就可以计算出函数的极值。显然，低次方程的求解可以用求根公式精确计算的 x 值，但是高次方程的求解就要用逼近的思路比如梯度下降法、最速下降法、牛顿法等。本文主要讨论牛顿法和使用 python 对一元函数和多元函数求极值。

2. 一元函数牛顿法

假如函数f(x)是一个一元函数，使用泰勒公式二阶近似可以得到

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2$$

一元函数取极值的条件是$f'(x)=0$，两端分别求导得到

$$f'(x) = f(x_0) + f''(x_0)(x - x_0) \approx 0$$

整理出x的迭代公式

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

进一步写出迭代关系表达式，得到

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

因为牛顿法的迭代关系是根据要求解极值函数的导数图像在该点的斜率进行迭代，因此没有步长的概念，在哪一点得到极值仅仅依靠迭代的步数。相应的，迭代的次数越多越接近极值点。

3. 多元函数牛顿法

多元函数用牛顿法求极值的思路依然采用泰勒公式近似展开，只不过这里的 X 为向量形式，假设 f(X)为二元函数:

$$f(X) \approx f\left(X^{(0)}\right) + \nabla f\left(X^{(0)}\right)^{T}\left(X - X^{(0)}\right) + \frac{1}{2}\left(X - X^{(0)}\right)^{T}\nabla^2 f\left(X^{(0)}\right)\left(X - X^{(0)}\right)$$

对上述公式两侧求梯度，令其等于零，之后再进行整理，可得迭代公式：

$$X = X^{(0)} - H^{-1}f(X^{(0)})\nabla f(X^{(0)})$$

上述第一个公式中$\nabla f(X^{(0)})$表示在$X^{(0)}$点处的梯度值，$\nabla^2 f(X^{(0)})$表示该点的二阶梯度也可以写成 Hessian 矩阵，迭代公式使用了 Hessian 矩阵的逆和梯度值来确定。

二元或多元函数的极小值问题常常可以转化为如下两步：

1.通过求偏导数等于 0 来得到一个非线性方程组；

2.通过求解这个非线性方程组的解，得到极值点。

对于多元函数$F(x_1, x_2, \dots, x_n)$求极值，可先对$x_1, x_2, \dots, x_n$求偏导数并令其为 0，得到形如下图所示的非线性方程组，这里我们用牛顿法进行迭代求解。

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ \dots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

若向量记号为$X = (x_1, x_2, \dots, x_n)^T$，$F = (f_1, f_2, \dots, f_n)^T$，方程组就可以写成$F(X) = 0$的形式。

根据上述推导可以得到多元函数求根公式为：

$$X^{(k+1)} = X^{(k)} - F'\left(X^k\right)^{-1} F(X^{(k)})$$

其中X是向量，$F'\left(X^k\right)^{-1}$是非线性方程组对应的雅可比矩阵。

具体求解的时候，我们可以先通过绘图命令绘制图形，看交点。然后将交点附近的值带入迭代矩阵。最后求出小于误差的收敛解。

4. 程序设计

```
from sympy import *
import numpy as np

x=symbols('x')
y=symbols('y')
f=sin(x*x+y*y)*exp(-0.1*(x*x+y*y+x*y+2*x))

# 求一阶导
f1=diff(f,x)
f2=diff(f,y)
# 求二阶导
f11=diff(f,x,2)
f12=diff(f1,y)
f21=diff(f2,x)
f22=diff(f,y,2)

# Hessian 矩阵的逆中的各个元素
```

```
    a,b,c,d=[x/(f11*f22-f12*f21) for x in [f22,-f12,-f21,f11]]

    # 初值
    F0 = np.array([[-0.5],[0]])
    er=1
    k=0
    while er>1e-16:
        # 将值代入
        a1,b1,c1,d1=a.evalf(subs={'x':F0[0][0],'y':F0[1][0]}),b.evalf(subs={'x':F0[0][0],'y':F0[1][0]}),c.evalf(subs={'x':F0[0][0],'y':F0[1][0]}),d.evalf(subs={'x':F0[0][0],'y':F0[1][0]})
        ff1,ff2=f1.evalf(subs={'x':F0[0][0],'y':F0[1][0]}),f2.evalf(subs={'x':F0[0][0],'y':F0[1][0]})
        # F 一阶导数列向量,F_Hessian 矩阵的逆
        F=np.array([[ff1],[ff2]],dtype = 'float')
        F_=np.array([[a1,b1],[c1,d1]],dtype = 'float')
        # 牛顿迭代法
        F=F0-np.dot(F_,F)
        k=k+1
        er=np.linalg.norm(F0-F,1)
        F0=F
        print('迭代次数',"{0:.0f}".format(k),', 方程根的近似值为 x=',F0[0][0],', y=',F0[1][0])
        print('误差 er=',er)
```

5. 结果

我们选取 x=-0.5,y=0 作为初值进行牛顿迭代法。运行结果如下所示。从中我们可以看到迭代到第 6 次的时候就已经满足设置的误差阈值。

迭代次数 1，方程根的近似值为 x= 0.041451618859459805，y= 0.014783922332102708

误差 er= 0.5562355411915625

迭代次数 2，方程根的近似值为 x= -0.0005867136025848874，y= -0.0001460162859276068

误差 er= 0.056968271080075

迭代次数 3，方程根的近似值为 x= -1.0527984802726539e-07，y= -1.7085582162799538e-08

误差 er= 0.0007326075230823041

迭代次数 4，方程根的近似值为 x= -3.3543449814750972e-15，y= -3.5975328257827963e-16

误差 er= 1.2236542647596667e-07

迭代次数 5，方程根的近似值为 x= -3.1554436208840472e-30，y= -2.465190328815662e-31

误差 er= 3.714098264053374e-15
迭代次数 6 , 方程根的近似值为 x= 0.0 , y= 0.0
误差 er= 3.4019626537656134e-30