第 3 章 无约束优化的梯度方法

求解无约束优化问题的方法大致可以分为两类：一类方法利用目标函数的梯度信息来求取搜索方向，这类方法称为梯度方法，将在本章介绍；另一类方法只计算目标函数值，而不计算梯度，这类方法称为直接方法，将在第 4 章介绍。

从 2.1 节中列出的几个基本研究点来看，本章主要介绍如何构造搜索方向，这是无约束最优化的核心问题，不同的搜索方向构造方案，形成不同的无约束最优化方法。


47min

3.1 无约束优化的最优性条件

本节介绍无约束优化问题的最优性条件，这些条件将为各种算法的推导和分析提供必不可少的理论基础，其实，一维情况下的无约束优化问题最优性条件已经在 2.3 节中介绍过，本节不过是将一维情况下的最优性条件推广到高维情况而已。

3.1.1 下降方向

考虑优化问题

$$\begin{cases}\min & f(\boldsymbol{x}) \\ \text{s.t.} & \boldsymbol{x}\in\mathbf{R}^n\end{cases} \tag{3-1}$$

其中，$f:\mathbf{R}^n\to\mathbf{R}$ 是 $\mathbf{R}^n$ 上的非线性实值函数。由于问题(3-1)对决策变量 $\boldsymbol{x}$ 没有任何限制，所以称其为无约束非线性优化问题。

首先介绍下降方向的一个充分条件。

定理 3-1 设函数 $f(\boldsymbol{x})$一阶连续可微，在点 $\bar{\boldsymbol{x}}$，如果存在方向 $\boldsymbol{d}$，使$\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}<0$，则存在 $\alpha_{\max}>0$，使对任意 $\alpha\in(0,\alpha_{\max})$有 $f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})<f(\bar{\boldsymbol{x}})$。

证明 设 $\varphi(\alpha)=f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})$，由 $\varphi(\alpha)$在 $\alpha=0$ 处的一阶泰勒展开式

$$\begin{aligned}\varphi(\alpha)&=f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})\\&=f(\bar{\boldsymbol{x}})+\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}\cdot\alpha+o(|\alpha|)\\&=f(\bar{\boldsymbol{x}})+\frac{1}{\alpha}\left[\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}+\frac{o(|\alpha|)}{\alpha}\right]\end{aligned}$$

注意到$\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}<0$和$o(|\alpha|)$是关于$\alpha$的高阶无穷小，故当$|\alpha|$充分小时，有

$$\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}+\frac{o(|\alpha|)}{\alpha}<0$$

故$f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})<f(\bar{\boldsymbol{x}})$。

参考定义2-1可知，定理3-1的后半部分表明$\boldsymbol{d}$是函数$f(\boldsymbol{x})$在$\bar{\boldsymbol{x}}$处的一个下降方向，也就是说在$\bar{\boldsymbol{x}}$点处满足条件$\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}<0$的方向$\boldsymbol{d}$一定是在$\bar{\boldsymbol{x}}$处的一个下降方向。这一条件为后续下降算法中构造搜索方向提供了理论依据。

定理3-1的几何解释如图3-1所示，图中同心圆为函数$f(\boldsymbol{x})=x_1^2+x_2^2$的等高线，在点$\bar{\boldsymbol{x}}$处的下降方向为和梯度$\nabla f(\bar{\boldsymbol{x}})$成钝角的方向组成的锥（图中阴影部分），例如方向$\boldsymbol{d}$在$\alpha\in(0,\alpha_{\max})$时的确可以使$f(\boldsymbol{x})$的函数值下降，但当$\alpha>\alpha_{\max}$时函数值就不降反升了。

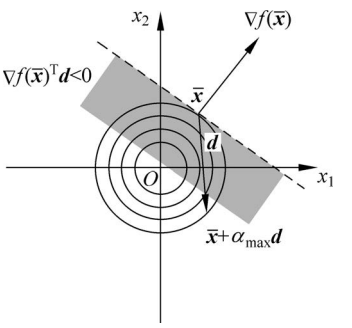

图3-1　下降方向的充分条件示意图

3.1.2　一阶必要条件

定理3-2　设函数$f(\boldsymbol{x})$一阶连续可微，若$\bar{\boldsymbol{x}}$是局部极小点，则梯度$\nabla f(\bar{\boldsymbol{x}})=0$。

证明　用反证法，设$\nabla f(\bar{\boldsymbol{x}})\neq 0$，令方向$\boldsymbol{d}=-\nabla f(\bar{\boldsymbol{x}})$，则有

$$\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}=-\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\nabla f(\bar{\boldsymbol{x}})=-\|\nabla f(\bar{\boldsymbol{x}})\|^2<0$$

于是由定理3-1，必存在$\alpha_{\max}>0$，使当$\alpha\in(0,\alpha_{\max})$时，成立

$$f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})<f(\bar{\boldsymbol{x}})$$

这与$\bar{\boldsymbol{x}}$是局部极小点矛盾，故定理成立。

其实满足$\nabla f(\boldsymbol{x})=0$的点称为稳定点，包括极大点、极小点和鞍点，也正是因为如此，定理3-2只是必要而非充分条件。

3.1.3　二阶必要条件

定理3-3　设函数$f(\boldsymbol{x})$二阶连续可微，若$\bar{\boldsymbol{x}}$是局部极小点，则梯度$\nabla f(\bar{\boldsymbol{x}})=0$，并且海森（Hessian）矩阵$\nabla^2 f(\bar{\boldsymbol{x}})$半正定。

证明　$\nabla f(\bar{\boldsymbol{x}})=0$已经在定理3-2中证明，以下证明$\nabla^2 f(\bar{\boldsymbol{x}})$半正定。对任一$\boldsymbol{d}\in\mathbf{R}^n$

$$f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})=f(\bar{\boldsymbol{x}})+\alpha\cdot\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+\frac{1}{2}\alpha^2\boldsymbol{d}^{\mathrm{T}}\nabla^2 f(\bar{\boldsymbol{x}})\boldsymbol{d}+o(\|\alpha\boldsymbol{d}\|^2)$$

$$=f(\bar{\boldsymbol{x}})+\frac{1}{2}\alpha^2\boldsymbol{d}^{\mathrm{T}}\nabla^2 f(\bar{\boldsymbol{x}})\boldsymbol{d}+o(\|\alpha\boldsymbol{d}\|^2)$$

于是

$$\frac{f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})-f(\bar{\boldsymbol{x}})}{\alpha^2}=\frac{1}{2}\boldsymbol{d}^{\mathrm{T}}\nabla^2 f(\bar{\boldsymbol{x}})\boldsymbol{d}+\frac{o(\|\alpha\boldsymbol{d}\|^2)}{\alpha^2}$$

由于 $\bar{\boldsymbol{x}}$ 是局部极小点,故当 $|\alpha|$ 充分小时,必有

$$f(\bar{\boldsymbol{x}}+\alpha\boldsymbol{d})-f(\bar{\boldsymbol{x}})\geqslant 0$$

故 $\boldsymbol{d}^{\mathrm{T}}\nabla^2 f(\bar{\boldsymbol{x}})\boldsymbol{d}\geqslant 0$,即 $\nabla^2 f(\bar{\boldsymbol{x}})$ 是半正定的。

3.1.4 二阶充分条件

定理 3-4 设函数 $f(\boldsymbol{x})$ 二阶连续可微,若梯度 $\nabla f(\bar{\boldsymbol{x}})=0$,并且海森矩阵 $\nabla^2 f(\bar{\boldsymbol{x}})$ 正定,则 $\bar{\boldsymbol{x}}$ 是局部极小点。

证明 由 $f(\boldsymbol{x})$ 在点 $\bar{\boldsymbol{x}}$ 处的二阶泰勒展开式的

$$f(\boldsymbol{x})=f(\bar{\boldsymbol{x}})+\nabla^{\mathrm{T}} f(\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})+\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})\nabla^2 f(\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})+o(\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2)$$

$$=f(\bar{\boldsymbol{x}})+\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^{\mathrm{T}}\nabla^2 f(\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})+o(\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2)$$

设 $\nabla^2 f(\bar{\boldsymbol{x}})$ 的最小特征值为 $\lambda_{\min}>0$,由于 $\nabla^2 f(\bar{\boldsymbol{x}})$ 正定,必有

$$(\boldsymbol{x}-\bar{\boldsymbol{x}})^{\mathrm{T}}\nabla^2 f(\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})\geqslant\lambda_{\min}\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2$$

从而

$$f(\boldsymbol{x})\geqslant f(\bar{\boldsymbol{x}})+\left[\frac{1}{2}\lambda_{\min}+\frac{o(\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2)}{\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2}\right]\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|$$

当 $\boldsymbol{x}\to\bar{\boldsymbol{x}}$ 时,$(o(\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2)/\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2)\to 0$,因此存在 $\bar{\boldsymbol{x}}$ 的 ε 邻域 $N(\bar{\boldsymbol{x}},\varepsilon)$,当 $\boldsymbol{x}\in N(\bar{\boldsymbol{x}},\varepsilon)$ 时有

$$f(\boldsymbol{x})\geqslant f(\bar{\boldsymbol{x}})$$

即 $\bar{\boldsymbol{x}}$ 是 $f(\bar{\boldsymbol{x}})$ 的局部极小点。

3.1.5 充要条件

前面几个定理给出的条件都不是充分必要的,而且都只是局部极小点的最优性条件。事实上,到目前为止,尚未发现一般无约束优化问题的充分必要最优性条件。在目标函数凸性的设定下,可以给出全局最优解的充分必要条件。

定理 3-5 设 $f(\boldsymbol{x})$ 是定义在 $\mathbf{R}^n$ 上的可微凸函数,则 $\bar{\boldsymbol{x}}\in\mathbf{R}^n$ 是全局极小点的充分必要条件是梯度 $\nabla f(\bar{\boldsymbol{x}})=0$。

证明 必要性由定理 3-2 得到,以下证明充分性。设 $\nabla f(\bar{\boldsymbol{x}})=0$,则对任一 $\boldsymbol{x}\in\mathbf{R}^n$,有

$$\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})=0$$

由于 $f(\boldsymbol{x})$ 是可微凸函数,则

$$f(\boldsymbol{x})\geqslant f(\bar{\boldsymbol{x}})+\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})=f(\bar{\boldsymbol{x}})$$

即 $\bar{\boldsymbol{x}}$ 是全局极小点。

在定理 3-5 中,若 $f(\boldsymbol{x})$ 还满足在点 $\bar{\boldsymbol{x}}$ 严格凸,即 $\nabla f(\bar{\boldsymbol{x}})=0$ 且 $\nabla^2 f(\bar{\boldsymbol{x}})$ 正定,则全局极小点是唯一的。

3.1.6　最优性条件应用案例

对于比较简单的优化问题，可以直接通过最优性条件来求解，以下给出两个案例。

【例 3-1】　利用最优性条件求解优化问题

$$\begin{cases}\min & f(\boldsymbol{x})=(x_1^2-1)^2+x_1^2+x_2^2-2x_1\\ \text{s.t.} & \boldsymbol{x}\in\mathbf{R}^2\end{cases}$$

解　由一阶必要条件

$$\nabla f(\boldsymbol{x})=\begin{bmatrix}\dfrac{\partial f}{\partial x_1}\\ \dfrac{\partial f}{\partial x_2}\end{bmatrix}=\begin{bmatrix}4x_1^3-2x_1-2\\ 2x_2\end{bmatrix}=0$$

得 $f(\boldsymbol{x})$的稳定点为 $\bar{\boldsymbol{x}}=(1,0)^{\mathrm{T}}$。函数 $f(\boldsymbol{x})$在 $\bar{\boldsymbol{x}}=(1,0)^{\mathrm{T}}$ 的海森矩阵为

$$\nabla^2 f(\bar{\boldsymbol{x}})=\begin{bmatrix}12x_1^2-2 & 0\\ 0 & 2\end{bmatrix}_{\boldsymbol{x}=(1,0)^{\mathrm{T}}}=\begin{bmatrix}10 & 0\\ 0 & 2\end{bmatrix}$$

显然，$\nabla^2 f(\bar{\boldsymbol{x}})$为正定矩阵，由二阶充分条件知 $\bar{\boldsymbol{x}}=(1,0)^{\mathrm{T}}$ 是局部极小点。

【例 3-2】　利用最优性条件求解优化问题

$$\begin{cases}\min & f(\boldsymbol{x})=\dfrac{1}{3}x_1^3+\dfrac{1}{3}x_2^3-x_2^2-x_1\\ \text{s.t.} & \boldsymbol{x}\in\mathbf{R}^2\end{cases}$$

解　由一阶必要条件

$$\nabla f(\boldsymbol{x})=\begin{bmatrix}\dfrac{\partial f}{\partial x_1}\\ \dfrac{\partial f}{\partial x_2}\end{bmatrix}=\begin{bmatrix}x_1^2-1\\ x_2^2-2x_2\end{bmatrix}=0$$

得 $f(\boldsymbol{x})$的稳定点为 $\boldsymbol{x}_1=(1,0)^{\mathrm{T}}$，$\boldsymbol{x}_2=(1,2)^{\mathrm{T}}$，$\boldsymbol{x}_3=(-1,0)^{\mathrm{T}}$ 和 $\boldsymbol{x}_4=(-1,2)^{\mathrm{T}}$。再由海森矩阵

$$\nabla^2 f(\boldsymbol{x})=\begin{bmatrix}2x_1 & 0\\ 0 & 2x_2-2\end{bmatrix}$$

得

$$\nabla^2 f(\boldsymbol{x}_1)=\begin{bmatrix}2 & 0\\ 0 & -2\end{bmatrix},\quad \nabla^2 f(\boldsymbol{x}_2)=\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}$$

$$\nabla^2 f(\boldsymbol{x}_3)=\begin{bmatrix}-2 & 0\\ 0 & -2\end{bmatrix},\quad \nabla^2 f(\boldsymbol{x}_4)=\begin{bmatrix}-2 & 0\\ 0 & 2\end{bmatrix}$$

可见，$\nabla^2 f(\boldsymbol{x}_1)$和$\nabla^2 f(\boldsymbol{x}_4)$是不定矩阵，$\nabla^2 f(\boldsymbol{x}_3)$是负定矩阵，$\nabla^2 f(\boldsymbol{x}_2)$是正定矩阵。于是由二阶充分条件知 $\boldsymbol{x}_2$ 是局部极小点。


43min

3.2 最速下降法

最速下降法是最基础的基于梯度的非线性优化算法,本节首先介绍最速下降方向,然后介绍最速下降法的算法步骤、算法流程和收敛性,最后给出一个最速下降法案例。

3.2.1 最速下降方向

最速下降法的基本思想是:首先,找到在当前迭代点使函数值下降最快的方向,然后在该方向上进行精确线搜索以使目标函数值产生最大的下降量。最速下降方向即为在某一点处使目标函数值下降最快的方向,以下讨论如何求取最速下降方向。

首先,由微积分知识可知,函数 $f(\boldsymbol{x})$在点 $\boldsymbol{x}$ 处沿方向 $\boldsymbol{d}$ 的变化率可用方向导数来表达,而对于可微函数,方向导数等于梯度与方向的内积,即

$$\mathrm{D}f(\boldsymbol{x};\boldsymbol{d})=\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d} \tag{3-2}$$

其中,$\mathrm{D}f(\boldsymbol{x};\boldsymbol{d})$为函数 $f(\boldsymbol{x})$沿方向 $\boldsymbol{d}$ 的方向导数。

接着,$f(\boldsymbol{x})$在点 $\boldsymbol{x}$ 处下降最快的方向即为使 $\mathrm{D}f(\boldsymbol{x};\boldsymbol{d})$最小的方向,即优化问题

$$\min_{\|\boldsymbol{d}\|=1}\ \mathrm{D}f(\boldsymbol{x};\boldsymbol{d}) \tag{3-3}$$

的解。注意到式(3-2),故问题(3-3)等价于

$$\min_{\|\boldsymbol{d}\|=1}\ \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d} \tag{3-4}$$

由 Schwartz 不等式,有

$$|\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}|\leqslant\|\nabla f(\boldsymbol{x})\|\cdot\|\boldsymbol{d}\|=\|\nabla f(\boldsymbol{x})\|$$

去掉绝对值符号,可以得到

$$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}\geqslant-\|\nabla f(\boldsymbol{x})\|$$

且当

$$\boldsymbol{d}=-\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|} \tag{3-5}$$

时等号成立,因此,在点 $\boldsymbol{x}$ 处,由式(3-5)得到的方向使方向导数 $\mathrm{D}f(\boldsymbol{x};\boldsymbol{d})$最小,即为最速下降方向。

综上所述,负梯度方向即为最速下降方向(the Steepest Descent Direction),记为

$$\boldsymbol{d}_{\mathrm{S}}=-\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|} \tag{3-6}$$

3.2.2 最速下降法

最速下降法的迭代公式是

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k \tag{3-7}$$

其中,搜索方向 $\boldsymbol{d}_k$ 取最速下降方向,即 $\boldsymbol{d}_k=\boldsymbol{d}_{\mathrm{S}}$,或直接取

$$\boldsymbol{d}_k=-\nabla f(\boldsymbol{x}_k) \tag{3-8}$$

步长 λ_k 是从 $\boldsymbol{x}_k$ 出发沿 $\boldsymbol{d}_k$ 方向进行一维搜索的精确步长，即

$$\lambda_k = \underset{\lambda>0}{\operatorname{argmin}} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k) \tag{3-9}$$

最速下降法的算法步骤如下。

算法 3-1　最速下降法

第 1 步：输入，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$。

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$k=1$。

第 3 步：计算搜索方向，$\boldsymbol{d}_k=-\nabla f(\boldsymbol{x}_k)$。

第 4 步：终止准则，若 $\|\boldsymbol{d}_k\|\leqslant\varepsilon$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_k$，$f^*=f(\boldsymbol{x}_k)$，否则求解一维搜索子问题(3-9)。

第 5 步：置 $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k$，$k:=k+1$，转到第 3 步。

值得注意的是，第 4 步中求解一维搜索子问题时使用的是精确线搜索方法，而不是非精确线搜索。可以使用第 2 章中介绍的黄金分割法、斐波那契法、牛顿法、割线法或抛物线法。

最速下降法的算法流程如图 3-2 所示。

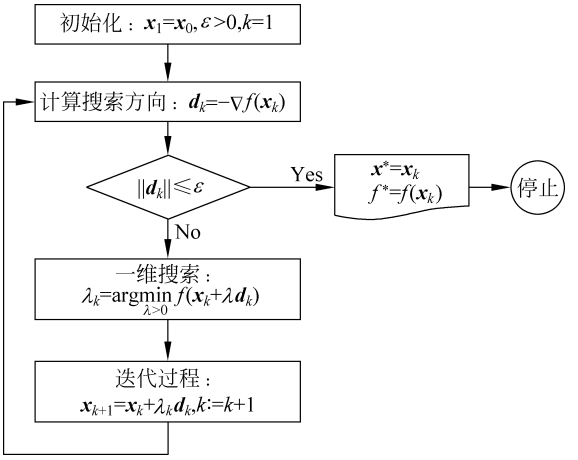

图 3-2　最速下降法的算法流程

3.2.3　最速下降法案例

以下通过一个具体案例给出最速下降法的 Python 代码。

【例 3-3】　用最速下降法求解优化问题

$$\begin{cases}\min & f(\boldsymbol{x})=4(x_1-2)^2+9(x_2+3)^2\\ \text{s.t.} & \boldsymbol{x}\in\mathbf{R}^2\end{cases}$$

显然，函数 $f(\boldsymbol{x})$ 表示的是一个二维空间中以 $\boldsymbol{x}^*=(2,-3)^{\mathrm{T}}$ 为中心的椭圆，故该问题的理论最优解为 $\boldsymbol{x}^*=(2,-3)^{\mathrm{T}}$，最优值为 $f^*=0$。目标函数的梯度为

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 8(x_1 - 2) \\ 18(x_2 + 3) \end{bmatrix}$$

以下为用最速下降法求解例 3-3 的 Python 代码。

```
"""
#代码 3-1 最速下降法
"""

import numpy as np

#原问题目标函数
def objfun(x):
    y = 4*(x[0]-2)**2+9*(x[1]+3)**2
    return y

#原问题目标函数的梯度函数
def gradfun(x):
    y =np.array([8*(x[0]-2),18*(x[1]+3)])
    return y

#一维搜索子问题目标函数
def lineobjfun(xk,dk,alpha):
    y = objfun(xk+alpha *dk)
    return y

#黄金分割法
def Golden(xk,dk,ak,bk,epsilon):
    lambdak = ak+0.382*(bk-ak)
    muk = ak+0.618*(bk-ak)
    flambdak = lineobjfun(xk,dk,lambdak)
    fmuk = lineobjfun(xk,dk,muk)
    while True:
        if bk-ak <= epsilon:
            xstar = (ak+bk)/2
            return xstar
        else:
            if flambdak > fmuk:
                ak = lambdak
                lambdak = muk
                flambdak = fmuk
                muk = ak+0.618*(bk-ak)
                fmuk = lineobjfun(xk,dk,muk)
            else:
                bk = muk
                muk = lambdak
                fmuk = flambdak
                lambdak = ak+0.382*(bk-ak)
                flambdak = lineobjfun(xk,dk,lambdak)
```

```
#最速下降法
def SteDesDir(x0,epsilon):
    xk = x0.copy()
    k = 0
    while True:
        k += 1
        gk = gradfun(xk)
        if np.linalg.norm(gk) <= epsilon:
            xstar = xk
            fstar = objfun(xk)
            return xstar,fstar,k
        else:
            dk = -gk
            lambdak = Golden(xk,dk,0.,3.,0.001)
            xk += lambdak*dk

#主程序
if __name__ == '__main__':
    epsilon = 0.001

    #第 1 组
    x0 = np.array([1.,1.])
    xstar,fstar,k = SteDesDir(x0,epsilon)
    print("第 1 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 2 组
    x0 = np.array([-2.,3.])
    xstar,fstar,k = SteDesDir(x0,epsilon)
    print("第 2 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 3 组
    x0 = np.array([10.,-10.])
    xstar,fstar,k = SteDesDir(x0,epsilon)
    print("第 3 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))
```

运行结果如下：

```
第 1 组:
Input:x0 = [ 1.99997618 -3.00000187]
Output: (xstar,fstar,k) = (array([ 1.99997618, -3.00000187]),
2.3005343989825326e-09, 6)
第 2 组:
Input:x0 = [ 1.99998104 -2.99997138]
```

```
Output: (xstar,fstar,k) = (array([ 1.99998104, -2.99997138]),
8.81133832674739e-09, 9)
第 3 组:
Input:x0 = [ 2.00004392 -2.9999901 ]
Output: (xstar,fstar,k) = (array([ 2.00004392, -2.9999901 ]),
8.597280705350136e-09, 12)
```

从运行结果可以看出,从不同的初始点出发均成功地求解了原问题,但循环次数不同。

3.2.4 最速下降法的收敛性

以下简要讨论最速下降法的收敛性。

定理 3-6 设 $f(\boldsymbol{x})$是一阶连续可微的实值函数,集合 S 是方程$\nabla f(\boldsymbol{x})=0$ 的解集合,即$S=\{\boldsymbol{x}\mid\nabla f(\boldsymbol{x})=0\}$,最速下降法产生的序列$\{\boldsymbol{x}_k\}$,若$\{\boldsymbol{x}_k\}$存在聚点,则其任一聚点 $\boldsymbol{x}^*\in S$。

定理 3-6 的详细证明参见文献[1]。简单地讲,定理 3-6 表达的意思是最速下降法产生的序列$\{\boldsymbol{x}_k\}$的每个聚点都是驻点。

最速下降法使用的是下降最快的方向,精确步长也使每个迭代步中目标函数的下降量达到了最大,所以直观上来看,最速下降法收敛速度应该非常快,但事实并非如此,对于某些目标函数,最速下降法会出现锯齿(Zig-Zag)现象,以下对此进行讨论。

首先,使用最速下降法搜索时,相继两个搜索方向是正交的。

对一维搜索子问题,由一阶必要最优性条件可知,精确步长 α_k 满足

$$\varphi'(\alpha_k)=\nabla f(\boldsymbol{x}_k+\alpha_k\boldsymbol{d}_k)^{\mathrm{T}}\boldsymbol{d}_k=0 \tag{3-10}$$

注意到 $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\alpha_k\boldsymbol{d}_k$,$\boldsymbol{d}_{k+1}=-\nabla f(\boldsymbol{x}_{k+1})^{\mathrm{T}}$ 和 $\boldsymbol{d}_k=-\nabla f(\boldsymbol{x}_k)$,故

$$\boldsymbol{d}_{k+1}^{\mathrm{T}}\boldsymbol{d}_k=0 \tag{3-11}$$

即相继两个搜索方向正交。

接着,如图 3-3 所示,若函数的等高线是比较扁狭的,而初始点恰好取在等高线的两极,则相继搜索方向正交的精确一维搜索就容易出现锯齿现象。

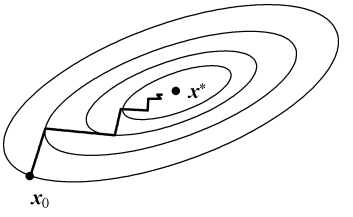

图 3-3 最速下降法锯齿现象示意图

锯齿现象会导致最速下降法在前段收敛较快,而越接近最优解时收敛速度越慢,所以最速下降法只有线性收敛速度,是收敛速度很慢的算法。为了克服最速下降法的锯齿现象,人们想出了很多改进方案,例如使用非精确线搜索,将收敛较慢的后段用其他算法替换,扰动搜索方向等,这些算法将在下文逐一介绍。


51min

3.3 牛顿法

2.6.1 节介绍了一维搜索中的牛顿法,本节将其推广到多维情形。牛顿法的基本思想是在迭代点 $\boldsymbol{x}_k$ 将目标函数用二次函数近似,用二次函数的极小点近似目标函数的极小点。

3.3.1　基本原理

设 $f:\mathbf{R}^n\to\mathbf{R}$ 是二次可微实值函数，又设 $\boldsymbol{x}_k$ 是 $f(\boldsymbol{x})$ 的极小点的一个估计，则函数 $f(\boldsymbol{x})$ 在点 $\boldsymbol{x}_k$ 的二次泰勒展开式为

$$f(\boldsymbol{x})=f(\boldsymbol{x}_k)+\nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}_k)+\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_k)^{\mathrm{T}}\nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k)+o(\|\boldsymbol{x}-\boldsymbol{x}_k\|^2) \tag{3-12}$$

其中，$\nabla^2 f(\boldsymbol{x})$ 是 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}_k$ 处的海森矩阵，$o(\|\boldsymbol{x}-\boldsymbol{x}_k\|^2)$ 为皮亚诺余项。在式(3-12)中忽略余项部分得到 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}_k$ 处的一个二次近似函数

$$f(\boldsymbol{x})\approx q(\boldsymbol{x})=f(\boldsymbol{x}_k)+\nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}_k)+\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_k)^{\mathrm{T}}\nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k) \tag{3-13}$$

为求 $q(\boldsymbol{x})$ 的稳定点，令

$$\nabla q(\boldsymbol{x})=0 \tag{3-14}$$

即

$$\nabla f(\boldsymbol{x}_k)+\nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k)=0 \tag{3-15}$$

设 $\nabla^2 f(\boldsymbol{x}_k)$ 可逆(非奇异)，则由式(3-15)可得牛顿法的迭代公式为

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}) \tag{3-16}$$

其中，$\nabla^2 f(\boldsymbol{x}_k)^{-1}$ 为海森矩阵 $\nabla^2 f(\boldsymbol{x})$ 的逆矩阵。一般将搜索方向

$$\boldsymbol{d}^N=-\nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k) \tag{3-17}$$

称作牛顿方向，于是式(3-16)可以解释为步长为 1 且搜索方向为牛顿方向的迭代。

在一维情况下，牛顿法的几何解释如图 3-4 所示，在点 $\boldsymbol{x}_1$ 处对函数 $f(\boldsymbol{x})$ 进行二次近似，得到二次函数 $q(\boldsymbol{x})$，其图像为一抛物线，最小化 $q(\boldsymbol{x})$ 得到下一个迭代点 $\boldsymbol{x}_2$，以此类推。由此可知，在二维情形下，牛顿法实际上是用一系列抛物面来近似曲线 $f(\boldsymbol{x})$，用抛物面的极小值点来逐渐逼近 $f(\boldsymbol{x})$ 的极小值点。

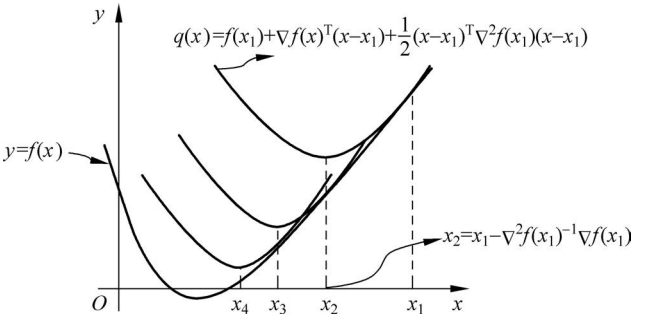

图 3-4　牛顿法示意图

3.3.2 牛顿法的算法步骤和流程

牛顿法的算法步骤如下。

算法 3-2 牛顿法

第 1 步：输入，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$。

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$k=1$。

第 3 步：计算梯度，$\nabla f(\boldsymbol{x}_k)$。

第 4 步：终止准则，若 $\|\nabla f(\boldsymbol{x}_k)\|\leqslant\varepsilon$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_k$，$f^*=f(\boldsymbol{x}_k)$，否则继续第 5 步。

第 5 步：计算海森矩阵 $\nabla^2 f(\boldsymbol{x}_k)$ 及其逆矩阵 $\nabla^2 f(\boldsymbol{x}_k)^{-1}$，计算牛顿方向

$$\boldsymbol{d}_k^N=-\nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$$

第 6 步：牛顿迭代，

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\boldsymbol{d}_k^N,k:=k+1$$

转到第 3 步。

牛顿法的算法流程如图 3-5 所示。

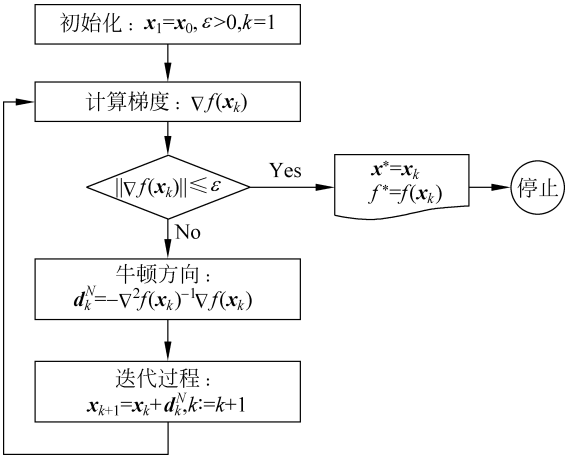

图 3-5 牛顿法的算法流程

对比牛顿法和最速下降法的流程图可以看出，它们的流程基本一致，不同在于搜索方向和搜索步长的选择。牛顿法因为要计算二阶梯度，即海森矩阵，所以也称为二阶算法。

牛顿法是收敛速度很快的算法。实际上，当牛顿法收敛时，前后两个迭代点满足关系

$$\|\boldsymbol{x}_{k+1}-\boldsymbol{x}^*\|\leqslant c\|\boldsymbol{x}_k-\boldsymbol{x}^*\|^2 \tag{3-18}$$

其中，c 是某常数，也就是说，牛顿法至少二阶收敛。

特别地，对于二次凸函数，用牛顿法求解，经过 1 次迭代即可达到极小值点。

设有二次凸函数

$$f(\boldsymbol{x})=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}+c \tag{3-19}$$

其中，$\boldsymbol{A}$ 是对称正定矩阵。

先用一阶必要条件求解极小值点。令

$$\nabla f(\boldsymbol{x})=\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}=\boldsymbol{0} \tag{3-20}$$

求解方程(3-20)得极小值点

$$\boldsymbol{x}^{*}=-\boldsymbol{A}^{-1}\boldsymbol{b} \tag{3-21}$$

再用牛顿法迭代求解。任取初始点 $\boldsymbol{x}_1$，根据牛顿法迭代公式(3-16)得

$$\boldsymbol{x}_2=\boldsymbol{x}_1-\nabla^2 f(\boldsymbol{x}_1)^{-1}\nabla f(\boldsymbol{x}_1)=\boldsymbol{x}_1-\boldsymbol{A}^{-1}(\boldsymbol{A}\boldsymbol{x}_1+\boldsymbol{b})=-\boldsymbol{A}^{-1}\boldsymbol{b} \tag{3-22}$$

显然，$\boldsymbol{x}^{*}=\boldsymbol{x}_2$，即经过 1 次迭代已经达到极小值点。

图 3-6 给出了最速下降法和牛顿法的迭代示意图。从图中可以看出，对于二次凸函数，每点的牛顿方向都是直接指向极小值点的，这就解释了为什么对于二次函数，牛顿法只需一次迭代便可到达极小值点。

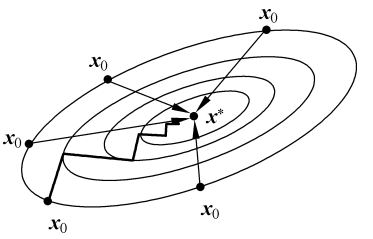

图 3-6　牛顿法和最速下降法比较示意图

以后还会遇到一些算法，当把它们用于二次凸函数时，类似于牛顿法，经有限次迭代必达到极小点，这种性质称为二次终止性。

牛顿法是局部算法。这意味着，当初始点远离极小值点时，牛顿方向可能并不是一个下降方向，因此牛顿法也就不收敛，即使牛顿方向是一个下降方向，使用牛顿迭代得到的点也不一定是一个沿牛顿方向的最好的点，这是因为牛顿法迭代的步长始终为 1，未进行任何优化。针对牛顿法局部收敛性和步长未进行优化的缺点，人们对牛顿法进行了各种改进，这些改进算法将在 3.3.3 节介绍。

3.3.3　牛顿法的改进

针对牛顿法局部收敛性和步长为 1 的缺点，人们提出了各种修正的牛顿法。本节介绍 3 种典型的修正牛顿法。

1. 阻尼牛顿法

阻尼牛顿法是指在牛顿法的基础上增加一维搜索，其迭代公式为

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k \tag{3-23}$$

其中，$\boldsymbol{d}_k=-\nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$ 为牛顿方向，λ_k 是由精确一维搜索得到的步长，即

$$\lambda_k=\underset{\lambda>0}{\operatorname{argmin}} f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k) \tag{3-24}$$

阻尼牛顿法的算法步骤如下。

算法 3-3　阻尼牛顿法

第 1 步：输入，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$。

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$k=1$。

第 3 步：计算梯度，$\nabla f(\boldsymbol{x}_k)$。

第 4 步：终止准则，若 $\|\nabla f(\boldsymbol{x}_k)\|\leqslant\varepsilon$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_k$，$f^*=f(\boldsymbol{x}_k)$，否则继续第 5 步。

第 5 步：计算海森矩阵 $\nabla^2 f(\boldsymbol{x}_k)$ 及其逆矩阵 $\nabla^2 f(\boldsymbol{x}_k)^{-1}$，计算牛顿方向

$$\boldsymbol{d}_k^N=-\nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$$

第 6 步：一维搜索

$$\lambda_k=\underset{\lambda>0}{\operatorname{argmin}} f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k^N)$$

第 7 步：迭代过程

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k^N,\quad k:=k+1$$

转到第 3 步。

由于阻尼牛顿法含有一维搜索，因此在牛顿方向是下降方向的前提下，每次迭代都可以使目标函数值有所下降，而且可以证明，阻尼牛顿法在适当的条件下具有全局收敛性，并且具有二阶收敛速度，但是，当迭代点远离极小值点时，牛顿方向不一定是下降方向，此时一维搜索得不到有效步长，阻尼牛顿法就失效了。

【例 3-4】　用阻尼牛顿法求解下列问题

$$\min\ f(\boldsymbol{x})=x_1^4+x_1x_2+(1+x_2)^2$$

该目标函数的梯度函数和海森矩阵分别为

$$\nabla f(\boldsymbol{x})=\begin{bmatrix}4x_1^3+x_2\\ x_1+2(1+x_2)\end{bmatrix},\quad \nabla^2 f(\boldsymbol{x})=\begin{bmatrix}12x_1^2 & 1\\ 1 & 2\end{bmatrix}$$

取初始点 $\boldsymbol{x}_1=(0,0)^{\mathrm{T}}$，在点 $\boldsymbol{x}_1$ 处，函数的梯度和海森矩阵分别为

$$\nabla f(\boldsymbol{x}_1)=\begin{bmatrix}0\\ 2\end{bmatrix},\quad \nabla^2 f(\boldsymbol{x}_1)=\begin{bmatrix}0 & 1\\ 1 & 2\end{bmatrix}$$

牛顿方向为

$$\boldsymbol{d}_1^N=-\nabla^2 f(\boldsymbol{x}_1)^{-1}\nabla f(\boldsymbol{x}_1)=-\begin{bmatrix}0 & 1\\ 1 & 2\end{bmatrix}^{-1}\begin{bmatrix}0\\ 2\end{bmatrix}$$

从 $\boldsymbol{x}_1$ 出发，沿 $\boldsymbol{d}_1^N$ 作一维搜索，令

$$\varphi(\lambda)=f(\boldsymbol{x}_1+\lambda\boldsymbol{d}_1^N)=16\lambda^4+1$$

求 $\varphi'(\lambda)$，并令其等于 0，得

$$\varphi'(\lambda)=64\lambda^3=0$$

则 $\lambda_1=0$。

显然，此时得不到有效步长，用阻尼牛顿法不能产生新的迭代点，但点 $\boldsymbol{x}_1=(0,0)^{\mathrm{T}}$ 尚不是问题的极小值点。产生这一困境的原因是海森矩阵 $\nabla^2 f(\boldsymbol{x}_1)$ 非正定，牛顿方向不是下降方向。Goldstein-Price 给出了一个解决这一困境的方案。

2. Goldstein-Price 修正牛顿法

Glodstein-Price 修正牛顿法的主要思想是将牛顿法和最速下降法结合起来使用。当牛顿方向不是下降方向或不是很好的下降方向时，使用最速下降方向进行过渡；当牛顿方向是下降方向时，使用牛顿方向进行一维搜索。

Goldstein-Price 修正牛顿法的搜索方向定义为

$$\boldsymbol{d}_k=\begin{cases}\boldsymbol{d}_k^N & \text{如果 } \cos(\boldsymbol{d}_k^N,-\nabla f(\boldsymbol{x}_k))\geqslant\eta\\ -\nabla f(\boldsymbol{x}_k) & \text{其他}\end{cases}\tag{3-25}$$

其中，$\eta\in(0,1)$，在式(3-25)中，如果 $\cos(\boldsymbol{d}_k^N,-\nabla f(\boldsymbol{x}_k))\geqslant\eta$，则认为牛顿方向是一个满足要求的下降方向，此时使用牛顿方向进行搜索，否则使用最速下降方向进行搜索。

Goldstein-Price 修正牛顿法的算法步骤如下。

算法 3-4　Goldstein-Price 修正牛顿法

第 1 步：输入，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$，参数 $\eta\in(0,1)$。

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$k=1$。

第 3 步：计算梯度，$\nabla f(\boldsymbol{x}_k)$。

第 4 步：终止准则，若 $\|\nabla f(\boldsymbol{x}_k)\|\leqslant\varepsilon$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_k$，$f^*=f(\boldsymbol{x}_k)$，否则继续第 5 步。

第 5 步：计算海森矩阵 $\nabla^2 f(\boldsymbol{x}_k)$ 及其逆矩阵 $\nabla^2 f(\boldsymbol{x}_k)^{-1}$，计算牛顿方向

$$\boldsymbol{d}_k^N=-\nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$$

如果 $\cos(\boldsymbol{d}_k^N,-\nabla f(\boldsymbol{x}_k))\geqslant\eta$，则 $\boldsymbol{d}_k=\boldsymbol{d}_k^N$，否则 $\boldsymbol{d}_k=-\nabla f(\boldsymbol{x}_k)$。

第 6 步：一维搜索

$$\lambda_k=\underset{\lambda>0}{\operatorname{argmin}} f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k)$$

第 7 步：迭代过程

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k,\quad k:=k+1$$

转到第 3 步。

Goldstein-Price 解决了阻尼牛顿法中牛顿方向不下降的问题，保证了算法的收敛性。

3. Goldfeld 修正牛顿法

Goldfeld 修正牛顿法由 Goldfeld 等于 1966 年提出，其基本思想是在海森矩阵非正定时(此时牛顿方向不一定是下降方向)，对海森矩阵进行修正，使其成为正定矩阵，从而得到一个下降方向。

具体地，令

$$\bar{\boldsymbol{G}}_k=\begin{cases}\nabla^2 f(\boldsymbol{x}_k) & \text{如果}\nabla^2 f(\boldsymbol{x}_k)\text{ 正定}\\ \nabla^2 f(\boldsymbol{x}_k)+v_k\boldsymbol{E} & \text{如果}\nabla^2 f(\boldsymbol{x}_k)\text{ 不正定}\end{cases}\tag{3-26}$$

其中，v_k 是将 $\nabla^2 f(\boldsymbol{x}_k)$ 修正为正定矩阵的较小正常数，$\boldsymbol{E}$ 是单位矩阵，则搜索方向

$$\boldsymbol{d}_k=-\bar{\boldsymbol{G}}_k^{-1}\nabla f(\boldsymbol{x}_k)\tag{3-27}$$

成为点 $\boldsymbol{x}_k$ 处的一个下降方向。

式(3-26)的关键是 v_k 的取值。首先使 $\bar{\boldsymbol{G}}_k$ 正定的 v_k 肯定是存在的,这是因为 $\nabla^2 f(\boldsymbol{x}_k)+v_k\boldsymbol{E}$ 的特征值就等于 $\nabla^2 f(\boldsymbol{x}_k)$ 的特征值与 v_k 之和,所以只要 v_k 足够大,总能使 $\nabla^2 f(\boldsymbol{x}_k)+v_k\boldsymbol{E}$ 的特征值为正,即使 $\nabla^2 f(\boldsymbol{x}_k)+v_k\boldsymbol{E}$ 正定,但 v_k 也不能太大,否则就覆盖了 $\nabla^2 f(\boldsymbol{x}_k)$ 原有的信息,所以 v_k 应取将 $\nabla^2 f(\boldsymbol{x}_k)$ 修正为正定矩阵的较小的常数,具体可取为稍大于 $\nabla^2 f(\boldsymbol{x}_k)$ 的最负特征值。

Goldfeld-Price 修正牛顿法的具体实现策略很多,也都比较复杂,超出了本书的范围,此处不再详细讨论。感兴趣的读者可以参考文献[1]。

3.3.4 牛顿法案例

以下通过一些具体案例给出牛顿法及修正牛顿法的 Python 代码。

【例 3-5】 用牛顿法求解优化问题

$$\min\ x_1^2+(x_2-1)^4$$

显然,函数的唯一极小值点是 $\boldsymbol{x}^*=(0,1)^{\mathrm{T}}$,其梯度函数为

$$\nabla f(\boldsymbol{x})=\begin{bmatrix}2x_1\\4(x_2-1)^3\end{bmatrix}$$

海森矩阵函数为

$$\nabla^2 f(\boldsymbol{x})=\begin{bmatrix}2 & 0\\0 & 12(x_2-1)^2\end{bmatrix}$$

以下为用牛顿法求解例 3-5 的 Python 代码。

```
"""
#代码 3-2 牛顿法
"""

import numpy as np

#原问题目标函数
def objfun(x):
    y = x[0]**2+(x[1]-1.)**4
    return y

#原问题目标函数的梯度函数
def gradfun(x):
    y =np.array([2*x[0],4.*(x[1]-1)**3])
    return y

#原问题目标函数的海森矩阵函数
def hessianfun(x):
    y = np.array([[2.,0.],[0.,12*(x[1]-1.)**2]])
    return y
```

```
#牛顿法
def Newton(x0,epsilon):
    xk = x0.copy()
    k = 0
    while True:
        k += 1
        gk = gradfun(xk)
        if np.linalg.norm(gk) <= epsilon:
            xstar = xk
            fstar = objfun(xk)
            return xstar,fstar,k
        else:
            Gk = hessianfun(xk)
            Gk_inv = np.linalg.inv(Gk)
            dk = -np.dot(Gk_inv,gk);
            xk += dk

#主程序
if __name__ == '__main__':
    epsilon = 0.001

    #第 1 组
    x0 = np.array([1.,2.])
    xstar,fstar,k = Newton(x0,epsilon)
    print("第 1 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 2 组
    x0 = np.array([-2.,3.])
    xstar,fstar,k = Newton(x0,epsilon)
    print("第 2 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 3 组
    x0 = np.array([10.,-10.])
    xstar,fstar,k = Newton(x0,epsilon)
    print("第 3 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))
```

运行结果如下：

```
第 1 组:
Input:x0 = [1. 2.]
Output: (xstar,fstar,k) = (array([0.    , 1.05852766]), 1.1733963864404772e-05, 8)
第 2 组:
Input:x0 = [-2. 3.]
```

```
Output: (xstar,fstar,k) = (array([0.    , 1.05202459]), 7.325455873891444e-06, 10)
第 3 组:
Input:x0 = [ 10. -10.]
Output: (xstar,fstar,k) = (array([0.    , 0.94347946]), 1.0205288104191327e-05, 14)
```

从运行结果可以看出,从不同的初始点出发均成功求解了原问题,但循环次数不同,当初始点离极小值点较远时,循环次数更多一些。

【例 3-6】 用 Goldstein-Price 修正牛顿法求解例 3-4,以下为其 Python 代码。

```
"""
#代码 3-3 Goldstein-Price 修正牛顿法
"""

import numpy as np

#原问题目标函数
def objfun(x):
    y = x[0]**4+x[0]*x[1]+(1+x[1])**2
    return y

#原问题目标函数的梯度函数
def gradfun(x):
    y =np.array([4.*x[0]**3+x[1],x[0]+2.*(1.+x[1])])
    return y

#原问题目标函数的海森矩阵函数
def hessianfun(x):
    y = np.array([[12.*x[0]**2,1.],[1.,2.]])
    return y

#一维搜索子问题目标函数
def lineobjfun(xk,dk,alpha):
    y = objfun(xk+alpha *dk)
    return y

#黄金分割法
def Golden(xk,dk,ak,bk,epsilon):
    lambdak = ak+0.382*(bk-ak)
    muk = ak+0.618*(bk-ak)
    flambdak = lineobjfun(xk,dk,lambdak)
    fmuk = lineobjfun(xk,dk,muk)
    while True:
        if bk-ak <= epsilon:
            xstar = (ak+bk)/2
            return xstar
        else:
            if flambdak > fmuk:
                ak = lambdak
```

```
                lambdak = muk
                flambdak = fmuk
                muk = ak+0.618*(bk-ak)
                fmuk = lineobjfun(xk,dk,muk)
            else:
                bk = muk
                muk = lambdak
                fmuk = flambdak
                lambdak = ak+0.382*(bk-ak)
                flambdak = lineobjfun(xk,dk,lambdak)

#Goldstein-Price 修正牛顿法
def Goldstein_Price_Newton(x0,epsilon):
    eta = 0.3
    xk = x0.copy()
    k = 0
    while True:
        k += 1
        gk = gradfun(xk)
        if np.linalg.norm(gk) <= epsilon:
            xstar = xk
            fstar = objfun(xk)
            return xstar,fstar,k
        else:
            Gk = hessianfun(xk)                          #海森矩阵
            Gk_inv = np.linalg.inv(Gk)                   #海森矩阵的逆
            dkN = -np.dot(Gk_inv,gk)                     #牛顿方向

            coss = np.dot(dkN,-gk)/(np.linalg.norm(dkN)*np.linalg.norm(-gk))
            if coss >= eta:
                dk = dkN                                 #使用牛顿方向搜索
            else:
                dk = -gk                                 #使用最速下降方向搜索

            lambdak = Golden(xk,dk,0.,3.,0.001)          #精确一维搜索
            xk += lambdak*dk

#主程序
if __name__ == '__main__':
    epsilon = 0.001

    #第 1 组
    x0 = np.array([0.,0.])
    xstar,fstar,k = Goldstein_Price_Newton(x0,epsilon)
    print("第 1 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 2 组
```

```
    x0 = np.array([-2.,3.])
    xstar,fstar,k = Goldstein_Price_Newton(x0,epsilon)
    print("第 2 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 3 组
    x0 = np.array([10.,-10.])
    xstar,fstar,k = Goldstein_Price_Newton(x0,epsilon)
    print("第 3 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))
```

运行结果如下：

```
第 1 组:
Input:x0 = [0. 0.]
Output: (xstar,fstar,k) = (array([ 0.69589498, -1.34798772]),
-0.5824451725273642, 5)
第 2 组:
Input:x0 = [-2. 3.]
Output: (xstar,fstar,k) = (array([ 0.69588586, -1.34794462]),
-0.5824451744350345, 7)
第 3 组:
Input:x0 = [ 10. -10.]
Output: (xstar,fstar,k) = (array([ 0.69588436, -1.3479421 ]),
-0.5824451744436266, 6)
```

可以看出，Goldstein-Price 修正牛顿法避免了阻尼牛顿法失效的困境，当取不同初始点时，均得到了极小值点。


80min

3.4 共轭梯度法

最速下降法和牛顿法在确定搜索方向时都只考虑了目标函数在当前迭代点的信息，并未考虑历史搜索方向，而历史搜索方向是可能给当前搜索方向的确定提供有用信息的。本节将介绍共轭梯度法，它是一种基于共轭方向的算法，在确定当前搜索方向时，引入了上一次搜索方向信息。以下首先介绍共轭方向的概念。

3.4.1 共轭方向

定义 3-1 设 $\boldsymbol{A}\in\mathbf{R}^{n\times n}$ 是对称正定方阵，若两个方向 $\boldsymbol{d}_1,\boldsymbol{d}_2\in\mathbf{R}^n$ 满足

$$\boldsymbol{d}_1^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_2=0 \tag{3-28}$$

则称这两个方向关于 $\boldsymbol{A}$ 共轭，或称它们关于 $\boldsymbol{A}$ 正交。

进一步地，若 $\boldsymbol{d}_1,\boldsymbol{d}_2,\cdots,\boldsymbol{d}_k\in\mathbf{R}^n$ 是 k 个方向，它们两两关于 $\boldsymbol{A}$ 共轭，即满足

$$\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_j=0,\quad i\neq j,i,j=1,2,\cdots,k \tag{3-29}$$

则称这组方向是 $\boldsymbol{A}$ 共轭的，或称它们为 $\boldsymbol{A}$ 的 k 个共轭方向。

在定义 3-1 中，如果 $\boldsymbol{A}$ 退化为单位矩阵，则两个方向关于 $\boldsymbol{A}$ 共轭退化为两个方向正交，因此可以认为共轭是正交的推广。

实际上，如果 $\boldsymbol{A}$ 是一般对称正定矩阵，$\boldsymbol{d}_i$ 和 $\boldsymbol{d}_j$ 关于 $\boldsymbol{A}$ 共轭，即 $(\boldsymbol{d}_i)^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{d}_j)=0$，也就是说方向 $\boldsymbol{d}_i$ 与方向 $\boldsymbol{A}\boldsymbol{d}_j$ 正交。同时有

$$(\boldsymbol{d}_i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_j=[(\boldsymbol{d}_i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_j]^{\mathrm{T}}=(\boldsymbol{d}_j)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{d}_i=(\boldsymbol{d}_j)^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{d}_i)=0 \tag{3-30}$$

也就是说，方向 $\boldsymbol{d}_j$ 与方向 $\boldsymbol{A}\boldsymbol{d}_i$ 也正交。

进一步还可以证明，若 $\boldsymbol{A}$ 为对称正定矩阵，$\boldsymbol{d}_1,\boldsymbol{d}_2,\cdots,\boldsymbol{d}_k$ 为 k 个 $\boldsymbol{A}$ 共轭的非零向量，则它们线性无关。

共轭方向具有明确的几何意义。

设二次函数

$$f(\boldsymbol{x})=\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{x}-\bar{\boldsymbol{x}}) \tag{3-31}$$

其中，$\boldsymbol{A}\in\mathbf{R}^{n\times n}$ 为对称正定矩阵，$\bar{\boldsymbol{x}}$ 是一个定点。

求 $f(\boldsymbol{x})$ 的一阶梯度，并令其等于 0，即

$$\nabla f(\boldsymbol{x})=\boldsymbol{A}(\boldsymbol{x}-\bar{\boldsymbol{x}})=0 \tag{3-32}$$

得到稳定点 $\boldsymbol{x}=\bar{\boldsymbol{x}}$，又由于点 $\bar{\boldsymbol{x}}$ 处的海森矩阵

$$\nabla^2 f(\boldsymbol{x})=\boldsymbol{A} \tag{3-33}$$

对称正定，所以由二阶充分条件可知，$\bar{\boldsymbol{x}}$ 为函数 $f(\boldsymbol{x})$ 的极小值点。

函数 $f(\boldsymbol{x})$ 的等值面

$$C=\left\{\boldsymbol{x}\in\mathbf{R}^n\mid\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{x}-\bar{\boldsymbol{x}})=c\right\} \tag{3-34}$$

是以 $\bar{\boldsymbol{x}}$ 为中心的椭球面，如图 3-7 所示。设 $\boldsymbol{x}_1$ 是在该等值面上的一点，则该等值面在点 $\boldsymbol{x}_1$ 处的法向量为

$$\nabla f(\boldsymbol{x})=\boldsymbol{A}(\boldsymbol{x}_1-\bar{\boldsymbol{x}}) \tag{3-35}$$

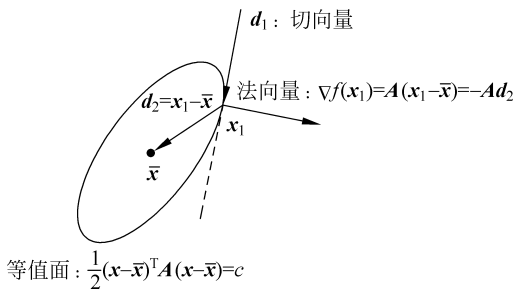

图 3-7　共轭方向示意图

又设 $\boldsymbol{d}_1$ 为点 $\boldsymbol{x}_1$ 处的切向量，则有 $\boldsymbol{d}_1$ 和 $\nabla f(\boldsymbol{x})$ 正交，即

$$0=\boldsymbol{d}_1^{\mathrm{T}}\nabla f(\boldsymbol{x}_1)=\boldsymbol{d}_1^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{x}_1-\bar{\boldsymbol{x}}) \tag{3-36}$$

令 $\boldsymbol{d}_2=\boldsymbol{x}_1-\bar{\boldsymbol{x}}$，即 $\boldsymbol{d}_2$ 是从 $\boldsymbol{x}_1$ 出发指向 $\bar{\boldsymbol{x}}$ 的方向，则式(3-36)成为

$$\boldsymbol{d}_1^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_2=0 \tag{3-37}$$

即 $\boldsymbol{d}_1$ 和 $\boldsymbol{d}_2$ 关于 $\boldsymbol{A}$ 共轭。

式(3-37)表明等值面上一点的切向量与从这一点指向极小值点的向量关于 $\boldsymbol{A}$ 共轭。由此可知，在极小化二次函数(3-31)时，若沿 $\boldsymbol{d}_1$ 和 $\boldsymbol{d}_2$ 这两个关于 $\boldsymbol{A}$ 共轭的方向进行搜索，则只需经过两次迭代就能达到极小值点 $\bar{\boldsymbol{x}}$。事实上，对于二次函数，若沿一组共轭方向(非零向量)搜索，经有限步迭代必达到极小值点。这是一种极好的性质，它保证了根据共轭方向构造的算法(例如共轭梯度法)具有二次终止性。

3.4.2 二次函数的共轭梯度法

共轭梯度法是一种特殊的共轭方向法，其基本思想是把共轭性与最速下降法相结合，在每次迭代中，基于迭代点处的梯度方向构造一个和前续搜索方向共轭的方向，并沿这个方向进行精确一维搜索。根据共轭方向的基本性质，这种方法具有二次终止性。

首先讨论对于二次凸函数的共轭梯度法，然后把这种方法推广到极小化一般函数的情形。

考虑问题

$$\min f(\boldsymbol{x})\triangleq\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}+c \tag{3-38}$$

其中，$\boldsymbol{x}\in\mathbf{R}^n$，$\boldsymbol{A}$ 是对称正定矩阵，$\boldsymbol{b}\in\mathbf{R}^n$ 是任意 n 维向量，c 是常数。$\boldsymbol{A}$ 的对称正定性保证了 $f(\boldsymbol{x})$是二次凸函数。

图 3-8 给出了一个共轭梯度法极小化二次凸函数的示意图，设第 k 次迭代的迭代点为 $\boldsymbol{x}_k$，搜索方向为 $\boldsymbol{d}_k$，经精确线搜索后得到下一迭代点 $\boldsymbol{x}_{k+1}$，即

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k \tag{3-39}$$

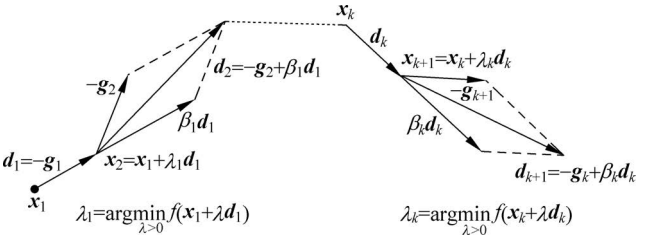

图 3-8 共轭梯度法示意图

其中

$$\lambda_k=\underset{\lambda>0}{\operatorname{argmin}} f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k) \tag{3-40}$$

下一迭代点 $\boldsymbol{x}_{k+1}$ 处的搜索方向 $\boldsymbol{d}_{k+1}$ 由 $\boldsymbol{x}_{k+1}$ 点处的负梯度方向 $-\boldsymbol{g}_{k+1}=-\nabla f(\boldsymbol{x}_{k+1})$和上一次搜索方向 $\boldsymbol{d}_k$ 经线性组合

$$\boldsymbol{d}_{k+1}=-\boldsymbol{g}_{k+1}+\beta_k\boldsymbol{d}_k \tag{3-41}$$

得到，其中 β_k 是待定参数，接着进入下一轮迭代。

在上述过程中，有两个参数需要计算，一是 $\boldsymbol{x}_k$ 在 $\boldsymbol{d}_k$ 方向的精确步长 λ_k，二是计算下一个搜索方向 $\boldsymbol{d}_{k+1}$ 的待定参数 β_k，以下分别讨论。

1. 求 λ_k

为了求解一维搜索问题

$$\min_{\lambda>0}\varphi(\lambda)\triangleq f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k) \tag{3-42}$$

只需求 $\varphi'(\lambda)$，并令其等于 0，得

$$\varphi'(\lambda)=\nabla f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k)^{\mathrm{T}}\boldsymbol{d}_k=0 \tag{3-43}$$

由于 $\nabla f(\boldsymbol{x})=\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}$，故式(3-43)为

$$\begin{aligned}[\boldsymbol{A}(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k)+\boldsymbol{b}]^{\mathrm{T}}\boldsymbol{d}_k&=[\boldsymbol{A}\boldsymbol{x}_k+\boldsymbol{b}+\lambda\boldsymbol{A}\boldsymbol{d}_k)]^{\mathrm{T}}\boldsymbol{d}_k\\&=[\boldsymbol{g}_k+\lambda\boldsymbol{A}\boldsymbol{d}_k]^{\mathrm{T}}\boldsymbol{d}_k=0\end{aligned} \tag{3-44}$$

故

$$\lambda_k=-\frac{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{d}_k}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_k} \tag{3-45}$$

2. 求 β_k

按照共轭梯度法的基本思想，$\boldsymbol{d}_{k+1}$ 和 $\boldsymbol{d}_k$ 应为 $\boldsymbol{A}$ 共轭的，即

$$\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{k+1}=0 \tag{3-46}$$

将式(3-41)代入公式(3-46)的

$$\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}(-\boldsymbol{g}_{k+1}+\beta_k\boldsymbol{d}_k)=-\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{g}_{k+1}+\beta_k\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_k=0 \tag{3-47}$$

故

$$\beta_k=\frac{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{g}_{k+1}}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_k} \tag{3-48}$$

式(3-48)是由 Fletcher 和 Reeves 首先提出的，所以该共轭方向法称为 Fletcher-Reeves 共轭梯度法，简称 FR 法，公式(3-48)称为 FR 公式。

事实上，对于正定二次函数，不仅 $\boldsymbol{d}_k$ 和 $\boldsymbol{d}_{k+1}$ 是 $\boldsymbol{A}$ 共轭的，根据 FR 法产生的搜索方向 $\boldsymbol{d}_1,\boldsymbol{d}_2,\cdots,\boldsymbol{d}_m$ 是两两 $\boldsymbol{A}$ 共轭的。我们有以下定理。

定理 3-7　对于正定二次函数(3-38)，具有精确一维线搜索的 FR 法在 $m\leqslant n$ 次一维搜索后即终止，并且对所有 $i(1\leqslant i\leqslant m)$，下列关系成立

(1) $\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_j=0,i\neq j,i,j=1,2,\cdots,m$；

(2) $\boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{g}_j=0,(j=1,2,\cdots,i-1)$；

(3) $\boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{d}_i=-\boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{g}_i$(蕴含 $\boldsymbol{d}_i\neq0$)。

我们略去定理 3-1 的证明，感兴趣的读者可以参考文献[4]。

对于正定二次函数，可以推导出一个 FR 公式的等价公式，使不做矩阵运算就能求出系数 β_k。考虑到 $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k$，则

$$\begin{aligned}\beta_k &= \frac{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{g}_{k+1}}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_k} = \frac{\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k)/\lambda_k}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k)/\lambda_k} \\ &= \frac{\boldsymbol{g}_{k+1}^{\mathrm{T}}(\boldsymbol{g}_{k+1}-\boldsymbol{g}_k)}{\boldsymbol{d}_k^{\mathrm{T}}(\boldsymbol{g}_{k+1}-\boldsymbol{g}_k)} = \frac{\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{g}_{k+1}-\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{g}_k}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{g}_{k+1}-\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{g}_k}\end{aligned} \tag{3-49}$$

根据定理 3-1 可知 $\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{g}_k=0$，$\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{g}_k=-\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{g}_k$，并且由精确线搜索得 $\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{g}_{k+1}=0$，则式(3-49)得到

$$\beta_k = \frac{\|\boldsymbol{g}_{k+1}\|^2}{\|\boldsymbol{g}_k\|^2} \tag{3-50}$$

需要着重指出的是，初始搜索方向选择最速下降方向($\boldsymbol{d}_1=-\nabla f(\boldsymbol{x}_1)$)十分重要，如果选择别的方向作为初始方向，其余方向均按公式(3-41)构造，则构造出的一组搜索方向不一定是 $\boldsymbol{A}$ 共轭的。

针对二次凸目标函数的最速下降法、牛顿法、共轭梯度法搜索方向比较如图 3-9 所示。从图中可以看出，牛顿方向直接指向最优解，所以牛顿法只需一次迭代就可以达到最优解。共轭梯度方向经过一次过渡后指向最优解，所以共轭梯度法经过两次迭代就可以达到最优解，具有二次终止性。最速下降方向呈 Z 字形收敛到最优解，在要到达到最优解时出现了严重的锯齿现象，收敛速度变慢。

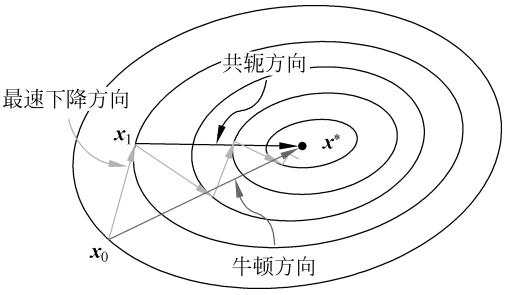

图 3-9　最速下降法、牛顿法、共轭梯度法搜索方向比较示意图

针对二次凸函数(3-38)的共轭梯度法的算法步骤如下。

算法 3-5　共轭梯度法

第 1 步：输入，目标函数信息 $\boldsymbol{A}$，$\boldsymbol{b}$，c，n(n 为问题的维度)，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$。

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$k=0$。

第 3 步：第 1 次迭代

$$\boldsymbol{g}_k = \boldsymbol{A}\boldsymbol{x}_k+\boldsymbol{b}, \boldsymbol{d}_k = -\boldsymbol{g}_k$$

$$\lambda_k = -\frac{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{d}_k}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_k}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k, k:=k+1$$

第 4 步：终止准则，若 $\|\boldsymbol{g}_k\|\leqslant\varepsilon$ 或 $k\geqslant n$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_k$，$f^*=f(\boldsymbol{x}_k)$，否则继续第 5 步。

续表

第 5 步：计算共轭方向：

$$\boldsymbol{d}_k = -\boldsymbol{g}_k + \beta_{k-1}\boldsymbol{d}_{k-1}$$

其中，

$$\beta_{k-1} = \frac{\boldsymbol{d}_{k-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{g}_k}{\boldsymbol{d}_{k-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{k-1}}$$

第 6 步：迭代过程：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k\boldsymbol{d}_k$$

其中，

$$\lambda_k = -\frac{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{d}_k}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_k}$$

令 $k:=k+1$，返回第 4 步。

针对二次凸函数(3-38)的共轭梯度法的算法流程如图 3-10 所示。

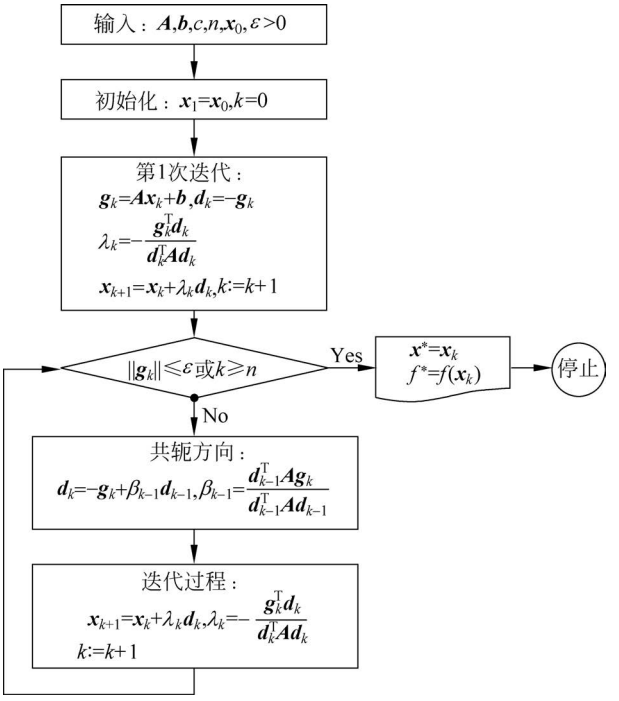

图 3-10　二次函数共轭梯度法的算法流程

这里还应该指出，在共轭梯度法中，可以采用不同的公式计算因子 β_k，除了式(3-50)之外，还有以下几种常见的形式：

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^{\mathrm{T}}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k)}{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{g}_k} \tag{3-51}$$

$$\beta_k=\frac{\boldsymbol{g}_{k+1}^{\mathrm{T}}(\boldsymbol{g}_{k+1}-\boldsymbol{g}_k)}{\boldsymbol{d}_k^{\mathrm{T}}(\boldsymbol{g}_{k+1}-\boldsymbol{g}_k)} \tag{3-52}$$

$$\beta_k=\frac{\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{g}_{k+1}}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{g}_k} \tag{3-53}$$

其中,式(3-51)是由 Polak、Ribiere 和 Polyak 提出,使用这个公式的共轭梯度法称为 PRP 共轭梯度法;式(3-52)是由 Crowder 和 Wolfe 提出的;式(3-53)是由 Dixon 提出的。当目标函数是二次凸函数且初始方向取负梯度方向时,从式(3-50)到式(3-53)4 个公式是等价的,但用于一般函数时,得到的搜索方向一般是不同的,这一点将在 3.4.3 节讨论。

3.4.3 一般函数的共轭梯度法

3.4.2 节介绍的共轭梯度法是基于目标函数为二次凸函数这一基本假设的,本节放开这一假设,讨论目标函数为一般函数 $f(\boldsymbol{x})$的共轭梯度法。

相较于目标函数为二次凸函数的共轭梯度法,目标函数为一般函数的共轭梯度法主要有以下几点不同:

(1) 步长 λ_k 不能再用式(3-45)计算,因为式(3-45)的推导正是基于目标函数为二次凸函数这一事实的。可以用通常的一维搜索方法来确定步长。

(2) 对于一般函数 $f(\boldsymbol{x})$,类似于二次凸函数中的对称正定方阵 $\boldsymbol{A}$ 是不存在的,必须用迭代点处的海森矩阵$\nabla^2 f(\boldsymbol{x}_k)$进行代替,但这会带来新的问题,因为对于一般函数 $f(\boldsymbol{x})$,$\nabla^2 f(\boldsymbol{x}_k)$并不一定正定,所以共轭方向并不一定保证下降,从而导致共轭梯度法成为非下降算法。

(3) 一般来讲,对于一般函数 $f(\boldsymbol{x})$,共轭梯度法不具有二次终止性,即不能在 n 步达到最优。

针对一般函数共轭梯度法不具有二次终止性的缺陷,一般可以有以下两种解决方案:

(1) 直接延续共轭梯度迭代,即总是使用式(3-41)构造搜索方向。

(2) 重新开始共轭梯度迭代,即以 n 次迭代作为一轮,每 n 次迭代后就重新取一次梯度方向,重新开始共轭梯度法。

其实,解决方案(1)和(2)都仅仅是根据求共轭方向的公式来构造搜索方向而已,并不能保证构造出的方向是共轭的,因为此时并不存在一个固定的对称正定矩阵 $\boldsymbol{A}$。

针对一般目标函数使用重启机制的共轭梯度法的算法步骤如下。

算法 3-6 共轭梯度法(针对一般目标函数,使用重启机制)

第 1 步:输入,初始点 $\boldsymbol{x}_0$,容忍误差 $\varepsilon>0$。

第 2 步:初始化,置 $\boldsymbol{y}_1=\boldsymbol{x}_1=\boldsymbol{x}_0$,$k=j=1$。

第 3 步:计算梯度,$\boldsymbol{d}_1=-\nabla f(\boldsymbol{y}_1)$。

第 4 步:终止准则,若 $\|\nabla f(\boldsymbol{y}_j)\|\leqslant\varepsilon$,则停止迭代,输出 $\boldsymbol{x}^*=\boldsymbol{y}_j$,$f^*=f(\boldsymbol{x}^*)$,否则继续第 5 步。

第 5 步:步长和迭代

续表

$$\lambda_j = \underset{\lambda>0}{\arg\min} f(\boldsymbol{y}_j + \lambda \boldsymbol{d}_j)$$

$$\boldsymbol{y}_{j+1} = \boldsymbol{y}_j + \lambda_j \boldsymbol{d}_j$$

第 6 步：重启判断，若 $j<n$，则转到第 7 步，否则转到第 8 步。

第 7 步：计算共轭方向：计算

$$\boldsymbol{d}_{j+1} = -\nabla f(\boldsymbol{y}_{j+1}) + \beta_j \boldsymbol{d}_j$$

其中，

$$\beta_j = \frac{\| \nabla f(\boldsymbol{y}_{j+1}) \|^2}{\| \nabla f(\boldsymbol{y}_j) \|^2}$$

令 $j:=j+1$，返回第 4 步。

第 8 步：重启共轭方向，令 $\boldsymbol{x}_{k+1}=\boldsymbol{y}_{n+1}$，$\boldsymbol{y}_1=\boldsymbol{x}_{k+1}$，$j=1$，$k:=k+1$，计算

$$\boldsymbol{d}_1 = -\nabla f(\boldsymbol{y}_1)$$

转到第 4 步。

针对一般目标函数使用重启机制的共轭梯度法的算法流程如图 3-11 所示。

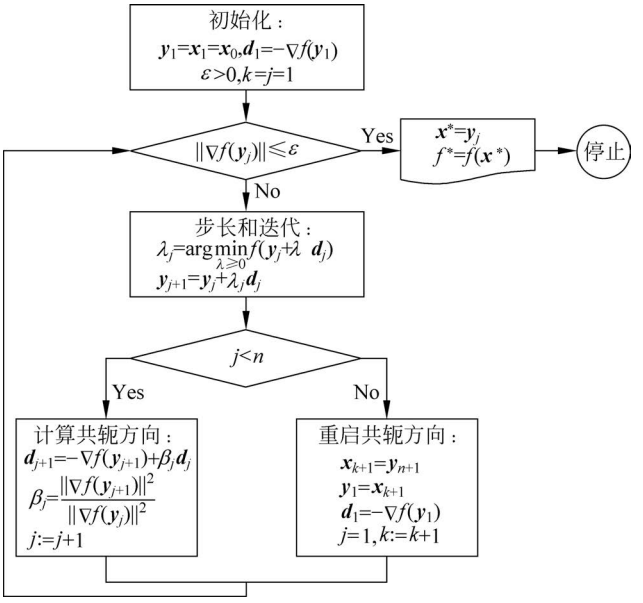

图 3-11　一般函数共轭梯度法（带重启机制）的算法流程

3.4.4　共轭梯度法案例

本节通过两个具体的案例分别给出针对二次凸目标函数和一般目标函数的共轭梯度法。

【例 3-7】 用 FR 法求二次凸优化问题

$$\min\ (x_1-2)^2+2(x_2-1)^2$$

显然，该问题的目标函数为二次凸函数，其唯一极小值点为 $\boldsymbol{x}^*=(2,1)^{\mathrm{T}}$，可以将目标

函数等价地写成矩阵形式

$$f(\boldsymbol{x})=\frac{1}{2}\begin{bmatrix}x_1 & x_2\end{bmatrix}\begin{bmatrix}2 & 0\\ 0 & 4\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}-4 & -4\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+6$$

故该问题的输入为

$$\boldsymbol{A}=\begin{bmatrix}2 & 0\\ 0 & 4\end{bmatrix},\quad \boldsymbol{b}=\begin{bmatrix}-4\\ -4\end{bmatrix},\quad c=6,\quad n=2$$

以下为用共轭梯度法求解例 3-7 的 Python 代码。

```
"""
#代码 3-4 共轭梯度法
"""

import numpy as np

#原问题目标函数
def objfun(A,b,c,x):
    y = np.dot(x,np.dot(A,x))+np.dot(b,x)+c
    return y

#原问题目标函数的梯度函数
def gradfun(A,b,c,x):
    y =np.dot(A,x)+b
    return y

#原问题目标函数的海森矩阵函数
def hessianfun(A,b,c,x):
    return A

#共轭梯度法
def Conjugate_Gradient_Method(A,b,c,x0,epsilon):
    xk = x0.copy()
    k = 0
    dim = len(b)

    #第 1 次迭代
    gk = gradfun(A,b,c,xk)              #计算梯度
    A = hessianfun(A,b,c,xk)            #计算海森矩阵
    dk = -gk                            #第 1 次使用负梯度方向
                                        #搜索步长
    lambdak = -np.dot(gk,dk)/np.dot(dk,np.dot(A,dk))
    xk += lambdak*dk                    #迭代
    dk_old = dk
    k += 1

    #后续迭代
    while True:
```

```
        gk = gradfun(A,b,c,xk)              #计算梯度
        A = hessianfun(A,b,c,xk)            #计算海森矩阵
        if k >= dim or np.linalg.norm(gk) <= epsilon:
            xstar = xk
            fstar = objfun(A,b,c,xk)
            return xstar,fstar,k
        else:
                                            #计算系数 beta
            betak = np.dot(dk_old,np.dot(A,gk))/np.dot(dk,np.dot(A,dk_old))
            dk = -gk+betak*dk_old           #搜索方向
            lambdak = -np.dot(gk,dk)/np.dot(dk,np.dot(A,dk))
            xk += lambdak*dk                #迭代
            dk_old = dk
            k += 1

#主程序
if __name__ == '__main__':
    #输入目标函数参数
    A = np.array([[2.,0.],[0.,4.]])
    b = np.array([-4.,-4.])
    c = 6

    epsilon = 0.001

    #第 1 组
    x0 = np.array([0.,0.])
    xstar,fstar,k = Conjugate_Gradient_Method(A,b,c,x0,epsilon)
    print("第 1 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 2 组
    x0 = np.array([-2.,3.])
    xstar,fstar,k = Conjugate_Gradient_Method(A,b,c,x0,epsilon)
    print("第 2 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 3 组
    x0 = np.array([10.,-10.])
    xstar,fstar,k = Conjugate_Gradient_Method(A,b,c,x0,epsilon)
    print("第 3 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))
```

运行结果如下：

```
第 1 组:
Input:x0 = [0. 0.]
```

```
Output: (xstar,fstar,k) = (array([2., 1.]), 6.0, 2)
第 2 组:
Input:x0 = [-2. 3.]
Output: (xstar,fstar,k) = (array([2., 1.]), 5.999999999999998, 2)
第 3 组:
Input:x0 = [ 10. -10.]
Output: (xstar,fstar,k) = (array([2., 1.]), 6.000000000000005, 2)
```

从运行结果可以看出,使用不同的初始点,共轭梯度法均在 2 步达到问题的极小值点,符合二次终止性特点。

【例 3-8】 用求一般函数的共轭梯度法求解问题

$$\min((x_1-3)(x_1+4))^2+((x_2-3)(x_2+4))^2$$

显然,该目标函数是一个非凸函数,其所有局部极小点为

$$\boldsymbol{x}^*=(3,-4)^{\mathrm{T}},(3,3)^{\mathrm{T}},(-4,3)^{\mathrm{T}},(-4,-4)^{\mathrm{T}}$$

其梯度函数为

$$\nabla f(\boldsymbol{x})=\begin{bmatrix}2(x_1^2+x_1-12)(2x_1+1)\\2(x_2^2+x_2-12)(2x_2+1)\end{bmatrix}$$

海森矩阵函数为

$$\nabla^2 f(\boldsymbol{x})=\begin{bmatrix}4(x_1^2+x_1-12)+2(2x_1+1)^2 & 0\\0 & 4(x_2^2+x_2-12)+2(2x_2+1)^2\end{bmatrix}$$

以下为用带重启机制的求一般函数的共轭梯度法求解例 3-8 的代码。

```
"""
#代码 3-5 一般函数共轭梯度法(带重启机制)
"""

import numpy as np

#原问题目标函数
def objfun(x):
    y = ((x[0]-3.)*(x[0]+4.))**2+((x[1]-3.)*(x[1]+4.))**2
    return y

#原问题目标函数的梯度函数
def gradfun(x):
    y = np.array([2.*(x[0]**2+x[0]-12.)*(2.*x[0]+1.),
                  2.*(x[1]**2+x[1]-12.)*(2.*x[1]+1.)])
    return y

#原问题目标函数的海森矩阵函数
def hessianfun(x):
    y = np.array([[4.*(x[0]**2+x[0]-12.)+2.*(2.*x[0]+1.)**2,0.],
                  [0.,4.*(x[1]**2+x[1]-12.)+2.*(2.*x[1]+1.)**2]])
```

```
    return y

#共轭梯度法(对一般函数带重启机制)
def Conjugate_Gradient_Method(x0,epsilon):
    xk = x0.copy()
    dim = len(xk)
    k = 0

    while True:
        gk = gradfun(xk)                          #计算梯度
        A = hessianfun(xk)                        #计算海森矩阵
                                                  #终止条件判断
        if np.linalg.norm(gk) <= epsilon:
                xstar = xk
                fstar = objfun(xk)
                return xstar,fstar,k

        #第 1 次迭代
        dk = -gk                                  #第 1 次使用负梯度方向
                                                  #搜索步长
        lambdak = -np.dot(gk,dk)/np.dot(dk,np.dot(A,dk))
        xk += lambdak*dk                          #迭代
        dk_old = dk
        k += 1

        #后续迭代
        while True:
            gk = gradfun(xk)                      #计算梯度
            A = hessianfun(xk)                    #计算海森矩阵
            if np.linalg.norm(gk) <= epsilon:
                xstar = xk
                fstar = objfun(xk)
                return xstar,fstar,k
            else:
                                                  #计算系数 beta
                betak = np.dot(dk_old,np.dot(A,gk))/np.dot(dk,np.dot(A,dk_old))
                                                  #搜索方向
                dk = -gk+betak*dk_old
                lambdak = -np.dot(gk,dk)/np.dot(dk,np.dot(A,dk))
                xk += lambdak*dk                  #迭代
                dk_old = dk
                k += 1

                if np.mod(k,dim)==0:              #重启共轭梯度法
                    break

#主程序
if __name__ == '__main__':
    epsilon = 0.001
```

```
#第 1 组
x0 = np.array([2.,4.])
xstar,fstar,k = Conjugate_Gradient_Method(x0,epsilon)
print("第 1 组:")
print("Input:x0 = {}".format(x0))
print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

#第 2 组
x0 = np.array([-3.,5.])
xstar,fstar,k = Conjugate_Gradient_Method(x0,epsilon)
print("第 2 组:")
print("Input:x0 = {}".format(x0))
print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

#第 3 组
x0 = np.array([10.,-10.])
xstar,fstar,k = Conjugate_Gradient_Method(x0,epsilon)
print("第 3 组:")
print("Input:x0 = {}".format(x0))
print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

#第 4 组
x0 = np.array([1.,-1.])
xstar,fstar,k = Conjugate_Gradient_Method(x0,epsilon)
print("第 4 组:")
print("Input:x0 = {}".format(x0))
print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))
```

运行结果如下：

```
第 1 组:
Input:x0 = [2. 4.]
Output: (xstar,fstar,k) = (array([3.00000004, 3.00000004]),
1.5063572340024025e-13, 7)
第 2 组:
Input:x0 = [-3. 5.]
Output: (xstar,fstar,k) = (array([-4., 3.]), 8.67308323667337e-18, 9)
第 3 组:
Input:x0 = [ 10. -10.]
Output: (xstar,fstar,k) = (array([ 3.00000706, -4.00000682]),
4.721974485918222e-09, 11)
第 4 组:
Input:x0 = [ 1. -1.]
Output: (xstar,fstar,k) = (array([-0.5, -0.5]), 300.125, 7)
```

从结果可以看出，带重启机制的共轭梯度法的确可以解出非凸优化问题的局部极小值点，但是最优解的得出也和初始点的选择有关，例如第 4 组就没有求得极小值点，而是得到了一个鞍点。

3.5　拟牛顿法

3.3 节介绍了牛顿法，其特点是收敛速度快，但海森矩阵及其逆矩阵的计算量大。牛顿法收敛速度快的本质原因是在搜索方向中使用了目标函数的二阶梯度信息，但也正是二阶梯度的计算大大增加了牛顿法的计算量。能否设计一种方法，只计算一阶梯度，但又能通过一阶梯度来近似二阶梯度或直接近似二阶梯度的逆，这样既降低了计算量，又保持了收敛速度。拟牛顿法正是基于这一想法而设计的。

3.5.1　对称秩 1(SR1)校正法

下面先分析在牛顿法中，海森矩阵的逆矩阵应该满足的条件，然后根据这一条件来构造一个矩阵，使该矩阵只需一阶梯度信息便可计算，但又满足该条件。

设在经过 k 次迭代后，得到迭代点 $\boldsymbol{x}_{k+1}$，将目标函数 $f(\boldsymbol{x})$在 $\boldsymbol{x}_{k+1}$ 处进行泰勒展开，并取二阶近似，得到

$$f(\boldsymbol{x}_k)\approx f(\boldsymbol{x}_{k+1})+\nabla f(\boldsymbol{x}_{k+1})^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}_{k+1})+\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_{k+1})^{\mathrm{T}}\nabla^2 f(\boldsymbol{x}_{k+1})(\boldsymbol{x}-\boldsymbol{x}_{k+1}) \tag{3-54}$$

对式(3-54)两边求梯度得到

$$\nabla f(\boldsymbol{x})\approx\nabla f(\boldsymbol{x}_{k+1})+\nabla^2 f(\boldsymbol{x}_{k+1})(\boldsymbol{x}-\boldsymbol{x}_{k+1}) \tag{3-55}$$

将$\nabla f(\boldsymbol{x}_{k+1})$移到左边，并令 $\boldsymbol{x}=\boldsymbol{x}_k$，$\nabla f(\boldsymbol{x}_k)=\boldsymbol{g}_k$，$\nabla f(\boldsymbol{x}_{k+1})=\boldsymbol{g}_{k+1}$，则

$$\boldsymbol{g}_k-\boldsymbol{g}_{k+1}\approx\nabla^2 f(\boldsymbol{x}_{k+1})(\boldsymbol{x}_k-\boldsymbol{x}_{k+1}) \tag{3-56}$$

又令 $\boldsymbol{q}_k=\boldsymbol{g}_k-\boldsymbol{g}_{k+1}$ 为梯度差(或速度差)，$\boldsymbol{p}_k=\boldsymbol{x}_k-\boldsymbol{x}_{k+1}$ 为位移，则式(3-56)为

$$\boldsymbol{q}_k\approx\nabla^2 f(\boldsymbol{x}_{k+1})\boldsymbol{p}_k \tag{3-57}$$

又设海森矩阵$\nabla^2 f(\boldsymbol{x}_{k+1})$可逆，则

$$\boldsymbol{p}_k\approx\nabla^2 f(\boldsymbol{x}_{k+1})^{-1}\boldsymbol{q}_k \tag{3-58}$$

式(3-58)给出了一个迭代点 $\boldsymbol{x}_{k+1}$ 处的海森矩阵的逆矩阵应该大致满足的条件。为了不计算 $\nabla f(\boldsymbol{x}_{k+1})^{-1}$ 本身，可以用一个不包含二阶梯度的矩阵 $\boldsymbol{H}_{k+1}$ 来替换 $\nabla^2 f(\boldsymbol{x}_{k+1})^{-1}$，即

$$\boldsymbol{p}_k=\boldsymbol{H}_{k+1}\boldsymbol{q}_k \tag{3-59}$$

称式(3-59)为拟牛顿条件，其中 $\boldsymbol{H}_{k+1}$ 可以通过 $\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$ 来计算，以下讨论两种常见的计算 $\boldsymbol{H}_{k+1}$ 的方法。

求 $\boldsymbol{H}_{k+1}$ 的基本思路是在 $\boldsymbol{H}_k$ 的基础上，通过加上一个校正矩阵 $\Delta\boldsymbol{H}_k$ 得到 $\boldsymbol{H}_{k+1}$，即

$$\boldsymbol{H}_{k+1}=\boldsymbol{H}_k+\Delta\boldsymbol{H}_k \tag{3-60}$$

初始的 $\boldsymbol{H}_1$ 可以取任意一个 n 阶对称正定矩阵,通常取单位矩阵 $\boldsymbol{E}$ 即可。

在 $\boldsymbol{H}_k$ 满足对称性的前提下,要使 $\boldsymbol{H}_{k+1}$ 也满足对称性,就必须保证 $\Delta\boldsymbol{H}_k$ 具有对称性,因此,可设

$$\Delta\boldsymbol{H}_k=\boldsymbol{v}_k\boldsymbol{v}_k^{\mathrm{T}} \tag{3-61}$$

这里$\boldsymbol{v}_k\neq 0$ 是一个由 $\boldsymbol{H}_k$、$\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$ 表示的向量。显然,$\Delta\boldsymbol{H}_k$ 是对称的,因为

$$(\Delta\boldsymbol{H}_k)^{\mathrm{T}}=(\boldsymbol{v}_k\,\boldsymbol{v}_k^{\mathrm{T}})^{\mathrm{T}}=\boldsymbol{v}_k\boldsymbol{v}_k^{\mathrm{T}}=\Delta\boldsymbol{H}_k$$

而且,$\Delta\boldsymbol{H}_k$ 的秩为 1,因为

$$1\leqslant R(\Delta\boldsymbol{H}_k)\leqslant\min\{R(\boldsymbol{v}_k),R(\boldsymbol{v}_k^{\mathrm{T}})\}=1$$

所以称这种校正方法为对称秩 1(Symmetric Rank 1,SR1)校正。

以下讨论如何用 $\boldsymbol{H}_k$、$\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$ 表示$\boldsymbol{v}_k$。

由拟牛顿条件式(3-59),校正式(3-60)和式(3-61)得

$$(\boldsymbol{H}_k+\boldsymbol{v}_k\boldsymbol{v}_k^{\mathrm{T}})\boldsymbol{q}_k=\boldsymbol{p}_k$$

即

$$\boldsymbol{H}_k\boldsymbol{q}_k+\boldsymbol{v}_k\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k=\boldsymbol{p}_k$$

假设$\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k\neq 0$,则

$$\boldsymbol{v}_k=\frac{1}{\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k}(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k) \tag{3-62}$$

两边同时和 $\boldsymbol{q}_k$ 作内积得

$$\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k=\frac{1}{\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k}(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k$$

进一步可得

$$(\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k)^2=(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k \tag{3-63}$$

另外,再由式(3-62)得

$$\boldsymbol{v}_k\boldsymbol{v}_k^{\mathrm{T}}=\frac{1}{(\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{q}_k)^2}(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}} \tag{3-64}$$

结合式(3-63)、(3-64)可得

$$\Delta\boldsymbol{H}_k=\boldsymbol{v}_k\boldsymbol{v}_k^{\mathrm{T}}=\frac{(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}}{(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k} \tag{3-65}$$

由此,推出了用 $\boldsymbol{H}_k$、$\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$ 表示校正矩阵 $\Delta\boldsymbol{H}_k$ 的公式。由式(3-60),对称秩 1 校正公式为

$$\boldsymbol{H}_{k+1}=\boldsymbol{H}_k+\frac{(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}}{(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k} \tag{3-66}$$

基于对称秩 1 校正的拟牛顿法算法步骤如下。

算法 3-7　拟牛顿法(基于 SR1 校正)

第 1 步：输入，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$，初始拟牛顿矩阵 $\boldsymbol{H}_0=\boldsymbol{E}$

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$\boldsymbol{H}_1=\boldsymbol{H}_0$，$k=1$

第 3 步：终止准则，若 $\|\nabla f(\boldsymbol{x}_k)\|\leqslant\varepsilon$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_k$，$f^*=f(\boldsymbol{x}_k)$，否则继续第 4 步。

第 4 步：计算拟牛顿方向

$$\boldsymbol{d}_k=-\boldsymbol{H}_k\nabla f(\boldsymbol{x}_k)$$

第 5 步：一维搜索和迭代

$$\lambda_k=\underset{\lambda\geqslant 0}{\operatorname{argmin}} f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k)$$

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k$$

第 6 步：对称秩 1 校正：

$$\boldsymbol{p}_k=\boldsymbol{x}_k-\boldsymbol{x}_{k+1},\boldsymbol{q}_k=\nabla f(\boldsymbol{x}_k)-\nabla f(\boldsymbol{x}_{k+1})$$

$$\boldsymbol{H}_{k+1}=\boldsymbol{H}_k+\frac{(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}}{(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k}$$

置 $k:=k+1$，转到第 3 步。

基于对称秩 1 校正的拟牛顿法的算法流程如图 3-12 所示。

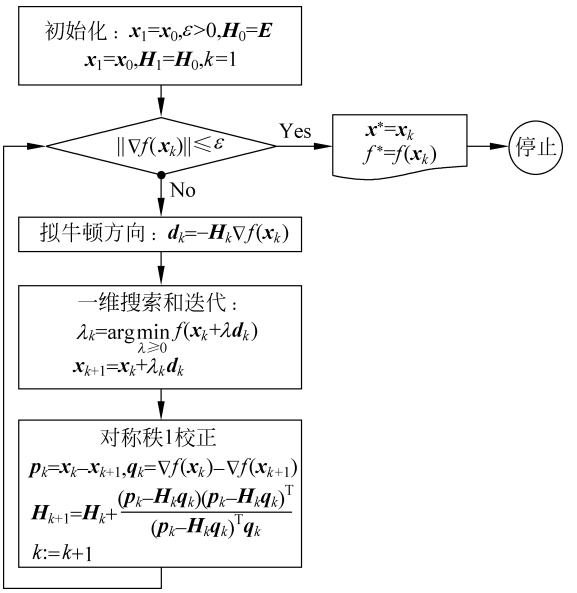

图 3-12　基于对称秩 1 校正的拟牛顿法的算法流程

可以证明，对称秩 1 校正在一定条件下是收敛的，并且具有二次终止性，具体证明此处不再阐述，有兴趣的读者可以参考文献[1]。

对称秩 1 校正也存在缺陷，首先，在 $\boldsymbol{H}_k$ 为正定的情况下，由式(3-66)可知，仅当

$$(\boldsymbol{p}_k-\boldsymbol{H}_k\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k>0 \tag{3-67}$$

时,才能确保 $\boldsymbol{H}_{k+1}$ 也正定,但式(3-67)是无法保证的,即使式(3-67)成立,也可能因为其值太小而导致 $\Delta\boldsymbol{H}_k$ 太大或无界,从而产生数值计算上的困难。

3.5.2 DFP 校正法

从对称秩 1 校正的校正公式(3-66)来看,校正矩阵 $\Delta\boldsymbol{H}_k$ 与 $\boldsymbol{p}_k$ 和 $\boldsymbol{H}_k\boldsymbol{q}_k$ 相关,所以可以考虑用 $\boldsymbol{p}_k$ 和 $\boldsymbol{H}_k\boldsymbol{q}_k$ 的某种组合来构造 $\Delta\boldsymbol{H}_k$,又考虑到对称秩 1 校正的构造方式,可设

$$\Delta\boldsymbol{H}_k = a\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}} + b(\boldsymbol{H}_k\boldsymbol{p}_k)(\boldsymbol{H}_k\boldsymbol{p}_k)^{\mathrm{T}}$$

即

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + a\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}} + b\boldsymbol{H}_k\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{H}_k \tag{3-68}$$

将式(3-68)代入拟牛顿条件式(3-59)得

$$\boldsymbol{H}_k\boldsymbol{q}_k + a\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k + b\boldsymbol{H}_k\boldsymbol{q}_k\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k = \boldsymbol{p}_k$$

经整理得

$$(1 + b(\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k))\boldsymbol{H}_k\boldsymbol{q}_k + (a(\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k) - 1)\boldsymbol{p}_k = 0$$

假设 $\boldsymbol{p}_k$ 和 $\boldsymbol{H}_k\boldsymbol{q}_k$ 线性无关,则上式系数为 0,即

$$\begin{cases} 1 + b(\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k) = 0 \\ a(\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k) - 1 = 0 \end{cases}$$

解得

$$\begin{cases} a = \dfrac{1}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k} \\ b = -\dfrac{1}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k} \end{cases} \tag{3-69}$$

于是得到校正公式

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}}}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k} - \frac{\boldsymbol{H}_k\boldsymbol{q}_k\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k} \tag{3-70}$$

校正公式(3-70)由 Davidon 首先提出,后来又被 Flecher 和 Powell 改进,所以称为 DFP 矫正法,又称变尺度法。

与对称秩 1 校正相比,DFP 校正的优势在于,在迭代尚未到达稳定点,即 $\nabla f(\boldsymbol{x}_k) \neq \boldsymbol{0}$ 时,DFP 校正构造的矩阵序列 $\boldsymbol{H}_i(i = 1, 2, \cdots, k)$ 均为正定矩阵,因此,基于 DFP 校正的拟牛顿法的每个搜索方向

$$\boldsymbol{d}_i = -\boldsymbol{H}_i\nabla f(\boldsymbol{x}_i), \quad i = 1, 2, \cdots, k$$

均为下降方向,所以基于 DFP 校正的拟牛顿法为下降算法。

还可以证明,若目标函数为正定二次函数

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x} + c$$

其中,$\boldsymbol{A}$ 为 n 阶对称正定矩阵,则用基于 DFP 校正的拟牛顿法优化该函数,经有限步迭代必

达到极小值点，即具有二次终止性。

关于矩阵序列 $\boldsymbol{H}_i(i=1,2,\cdots,k)$ 的正定性和基于 DFP 校正的拟牛顿法的二次终止性的证明，本书不作阐述，感兴趣的读者可以参考文献[4]。

针对二次凸优化问题，基于 DFP 矫正法的拟牛顿法的算法步骤和流程与基于对称秩 1 矫正法的拟牛顿法基本一样，只需用式(3-70)代替式(3-66)，此处不再赘述。

针对一般目标函数优化问题，与 3.4.3 节用共轭梯度法求一般目标函数的优化问题一样，可以使用重启机制来设计算法。

算法 3-8　拟牛顿法(针对一般目标函数，使用重启机制)

第 1 步：输入，初始点 $\boldsymbol{x}_0$，容忍误差 $\varepsilon>0$，初始拟牛顿矩阵 $\boldsymbol{H}_0=\boldsymbol{E}$。

第 2 步：初始化，置 $\boldsymbol{x}_1=\boldsymbol{x}_0$，$\boldsymbol{H}_1=\boldsymbol{H}_0$，$\boldsymbol{g}_1=\nabla f(\boldsymbol{x}_1)$，$k=1$。

第 3 步：计算拟牛顿方向

$$\boldsymbol{d}_k=-\boldsymbol{H}_k\boldsymbol{g}_k$$

第 4 步：一维搜索和迭代

$$\lambda_k=\underset{\lambda\geqslant 0}{\operatorname{argmin}} f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k)$$

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k$$

第 5 步：终止准则，若 $\|\nabla f(\boldsymbol{x}_{k+1})\|\leqslant\varepsilon$，则停止迭代，输出 $\boldsymbol{x}^*=\boldsymbol{x}_{k+1}$，$f^*=f(\boldsymbol{x}^*)$，否则继续第 6 步。

第 6 步：重启判断，若 $k<n$，则转到第 7 步，否则转到第 8 步。

第 7 步：DFP 校正：计算

$$\boldsymbol{g}_{k+1}=\nabla f(\boldsymbol{x}_{k+1})$$

$$\boldsymbol{p}_k=\boldsymbol{x}_k-\boldsymbol{x}_{k+1},\boldsymbol{q}_k=\boldsymbol{g}_k-\boldsymbol{g}_{k+1}$$

$$\boldsymbol{H}_{k+1}=\boldsymbol{H}_k+\frac{\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}}}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k}-\frac{\boldsymbol{H}_k\boldsymbol{q}_k\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k}$$

$$k:=k+1$$

返回第 3 步。

第 8 步：重启 DFP 校正，置 $\boldsymbol{x}_0=\boldsymbol{x}_{k+1}$，转到第 2 步。

针对一般目标函数的优化问题，基于 DFP 校正，并使用重启机制的拟牛顿法的算法流程如图 3-13 所示。

3.5.3　BFGS 校正法

3.5.2 节中推导 DFP 校正公式时使用的是拟牛顿条件式(3-59)，其中 $\boldsymbol{H}_{k+1}$ 起到的是 $\nabla^2 f(\boldsymbol{x}_{k+1})^{-1}$ 的作用，若不考虑海森矩阵的逆，而仅仅用一个矩阵 $\boldsymbol{B}_{k+1}$ 来取代海森矩阵 $\nabla^2 f(\boldsymbol{x}_{k+1})$，可以得到另一种形式的拟牛顿条件

$$\boldsymbol{q}_k=\boldsymbol{B}_{k+1}\boldsymbol{p}_k \tag{3-71}$$

拟牛顿条件式(3-59)和式(3-71)从形式上来看是一样的，只是把 $\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$ 交换了位置，将 $\boldsymbol{H}_{k+1}$ 换成了 $\boldsymbol{B}_{k+1}$(注意：$\boldsymbol{H}_{k+1}$ 起到的是 $\nabla^2 f(\boldsymbol{x}_{k+1})^{-1}$ 的作用，而 $\boldsymbol{B}_{k+1}$ 起到的是 $\nabla^2 f(\boldsymbol{x}_{k+1})$ 的作用)，所以使用和 DFP 校正公式完全一样的推导过程，将 $\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$ 互换，将 $\boldsymbol{H}_k$ 换成 $\boldsymbol{B}_k$ 即可得到 $\boldsymbol{B}_{k+1}$ 的校正公式

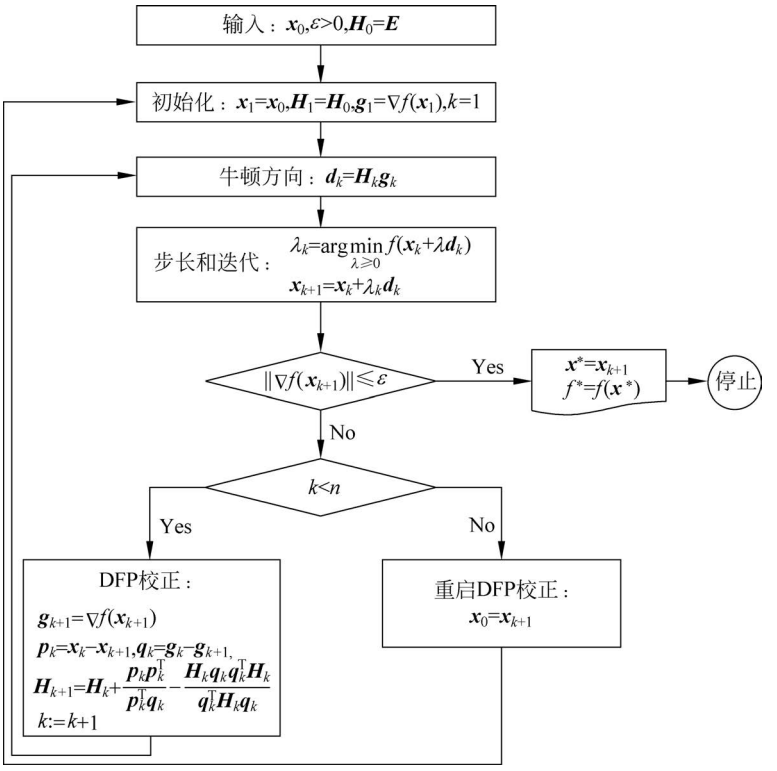

图 3-13　基于 DFP 校正的一般函数拟牛顿法(带重启机制)算法流程

$$\boldsymbol{B}_{k+1}=\boldsymbol{B}_k+\frac{\boldsymbol{q}_k\boldsymbol{q}_k^{\mathrm{T}}}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{p}_k}-\frac{\boldsymbol{B}_k\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{B}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{B}_k\boldsymbol{p}_k} \tag{3-72}$$

式(3-72)称为关于矩阵 $\boldsymbol{B}_k$ 的 BFGS 校正,有时也称为 DFP 的对偶公式。

设 $\boldsymbol{B}_{k+1}$ 可逆,则由式(3-71)可得,$\boldsymbol{p}_k=\boldsymbol{B}_{k+1}^{-1}\boldsymbol{q}_k$,这表明 $\boldsymbol{B}_{k+1}^{-1}$ 也满足拟牛顿条件(3-59),因此,只要能求出 $\boldsymbol{B}_{k+1}^{-1}$ 即可得到一个 $\boldsymbol{H}_{k+1}$ 的校正公式。根据式(3-72)求 $\boldsymbol{B}_{k+1}^{-1}$ 要用到求广义逆矩阵的 Sherman-Morrison 公式,这已经超出了本书的讨论范围,因此此处略去详细过程,感兴趣的读者可以参考文献[5]。求解后得

$$\boldsymbol{H}_{k+1}^{\mathrm{BFGS}}=\boldsymbol{H}_k+\left(1+\frac{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{q}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k}\right)\frac{\boldsymbol{p}_k\boldsymbol{p}_k^{\mathrm{T}}}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k}-\frac{\boldsymbol{p}_k\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{H}_k+\boldsymbol{H}_k\boldsymbol{q}_k\boldsymbol{p}_k^{\mathrm{T}}}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{q}_k} \tag{3-73}$$

这个重要的公式是由 Broyden、Fletcher、Goldfard 和 Shanno 于 1970 年提出的,它可以像 DFP 校正公式(3-70)一样使用,而且数值经验表明,它比 DFP 公式还好,因此目前得到广泛应用。

和式(3-72)的构造方法完全一样,式(3-73)也有与之对应的关于 $\boldsymbol{B}_{k+1}$ 的对偶校正公式

$$\boldsymbol{B}_{k+1}^{\mathrm{DFP}}=\boldsymbol{B}_k+\left(1+\frac{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{B}_k\boldsymbol{p}_k}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{p}_k}\right)\frac{\boldsymbol{q}_k\boldsymbol{q}_k^{\mathrm{T}}}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{p}_k}-\frac{\boldsymbol{q}_k\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{B}_k+\boldsymbol{B}_k\boldsymbol{p}_k\boldsymbol{q}_k^{\mathrm{T}}}{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{p}_k} \tag{3-74}$$

而式(3-74)正好和式(3-70)互逆。

综合式(3-70)、式(3-72)、式(3-73)、式(3-74),可以总结出它们的关系如图 3-14 所示。

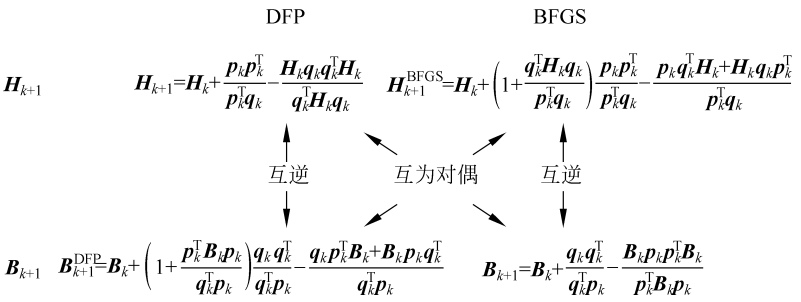

图 3-14　拟牛顿校正公式关系示意图

这里对偶指的是互换 $\boldsymbol{p}_k$ 和 $\boldsymbol{q}_k$,用 $\boldsymbol{B}_k$ 取代 $\boldsymbol{H}_k$ 或用 $\boldsymbol{H}_k$ 取代 $\boldsymbol{B}_k$。

基于 BFGS 校正公式的拟牛顿算法步骤和流程与算法 3.7 和 3.8 基本一致,此处不再赘述。

3.5.4　拟牛顿法案例

本节给出一个用拟牛顿法求解优化问题的具体案例,并比较不同方法的数值结果。

【例 3-9】　用拟牛顿法求优化问题

$$\min\ 4(1-x_1)^2+5(x_2-x_1^2)^2$$

显然,该问题有全局极小点 $\boldsymbol{x}^*=(1,1)^T$,其梯度函数为

$$\nabla f(\boldsymbol{x})=\begin{bmatrix}-8(1-x_1)-20x_1(x_2-x_1)^2\\ 10(x_2-x_1^2)\end{bmatrix}$$

用基于 DFP 校正公式和 BFGS 校正公式的拟牛顿法求解例 3-9 的 Python 代码如下:

```
"""
#代码 3-6 拟牛顿法
"""

import numpy as np

#原问题目标函数
def objfun(x):
    y = 4.*(1-x[0])**2+5.*(x[1]-x[0]**2)**2
    return y

#原问题目标函数的梯度函数
def gradfun(x):
    y =np.array([-8.*(1-x[0])-20.*x[0]*(x[1]-x[0]**2),
                 10.*(x[1]-x[0]**2)])
    return y
```

```
#一维搜索子问题目标函数
def lineobjfun(xk,dk,alpha):
    y = objfun(xk+alpha*dk)
    return y

#黄金分割法
def Golden(xk,dk,ak,bk,epsilon):
    lambdak = ak+0.382*(bk-ak)
    muk = ak+0.618*(bk-ak)
    flambdak = lineobjfun(xk,dk,lambdak)
    fmuk = lineobjfun(xk,dk,muk)
    while True:
        if bk-ak <= epsilon:
            xstar = (ak+bk)/2
            return xstar
        else:
            if flambdak > fmuk:
                ak = lambdak
                lambdak = muk
                flambdak = fmuk
                muk = ak+0.618*(bk-ak)
                fmuk = lineobjfun(xk,dk,muk)
            else:
                bk = muk
                muk = lambdak
                fmuk = flambdak
                lambdak = ak+0.382*(bk-ak)
                flambdak = lineobjfun(xk,dk,lambdak)

#拟牛顿法
def Quasi_Newton_Method(x0,epsilon):
    xk = x0.copy()
    Hk = np.eye(len(xk))
    k = 0

    #循环迭代
    gk = gradfun(xk)                                    #计算梯度
    while True:
        if np.linalg.norm(gk) <= epsilon:
            xstar = xk
            fstar = objfun(xk)
            return xstar,fstar,k
        else:
            xk_old = xk.copy()                          #上一迭代点
            dk = -np.dot(Hk,gk)                         #拟牛顿方向
            lambdak = Golden(xk,dk,0.,3.,0.001)         #一维搜索
            xk += lambdak*dk                            #迭代

            gk_old = gk.copy()                          #上一梯度
```

```
            gk = gradfun(xk)                                    #计算梯度

            pk = xk_old - xk                                    #位移
            qk = gk_old - gk                                    #梯度差
            pk = np.expand_dims(pk,axis=0)
            qk = np.expand_dims(qk,axis=0)

            #Hk_DFP
            Hk += (pk.T@pk)/(pk@qk.T)-(Hk@qk.T@qk@Hk)/(qk@Hk@qk.T)
            #Hk_BFGS
#Hk += ((1.+qk@Hk@qk.T)/(pk@qk.T))*((pk.T@pk)/(pk@qk.T))-(
#    (pk.T@qk@Hk+Hk@qk.T@pk)/(pk@qk.T))

            k += 1

#主程序
if __name__ == '__main__':

    epsilon = 0.001

    #第 1 组
    x0 = np.array([2.,1.])
    xstar,fstar,k = Quasi_Newton_Method(x0,epsilon)
    print("第 1 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 2 组
    x0 = np.array([-2.,3.])
    xstar,fstar,k = Quasi_Newton_Method(x0,epsilon)
    print("第 2 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))

    #第 3 组
    x0 = np.array([-3.,2.])
    xstar,fstar,k = Quasi_Newton_Method(x0,epsilon)
    print("第 3 组:")
    print("Input:x0 = {}".format(x0))
    print("Output: (xstar,fstar,k) = {}".format((xstar,fstar,k)))
```

代码 3-6 中分别使用了关于 $\boldsymbol{H}_k$ 的 DFP 校正公式和 BFGS 校正公式，其中 DFP 校正公式的运行结果如下：

```
第 1 组:
Input:x0 = [2. 1.]
Output: (xstar,fstar,k) = (array([1.00000087, 1.00000127]),
4.0727633986697135e-12, 5)
```

```
第 2 组：
Input:x0 = [-2. 3.]
Output: (xstar,fstar,k) = (array([1.00000509, 1.00002714]),
1.5412319736337878e-09, 8)
第 3 组：
Input:x0 = [-3. 2.]
Output: (xstar,fstar,k) = (array([1.00013199, 1.0003324 ]),
9.30781665498777e-08, 7)
```

BFGS 校正公式的运行结果如下：

```
第 1 组：
Input:x0 = [2. 1.]
Output: (xstar,fstar,k) = (array([1.00001475, 1.00006451]),
6.9997057519901e-09, 18)
第 2 组：
Input:x0 = [-2. 3.]
Output: (xstar,fstar,k) = (array([1.00002711, 1.00005245]),
2.9560477071388793e-09, 11)
第 3 组：
Input:x0 = [-3. 2.]
Output: (xstar,fstar,k) = (array([1.00003705, 1.00005663]),
7.016546377154817e-09, 42)
```

可见，两个校正公式均找到了全局极小值点，DFP 校正公式的搜索次数低于 BFGS 校正公式。