Lecture 10: Basic Optimization

Unconstrained Optimization (Derivative Methods)

The methods of the previous section do not utilize any derivative information about the objective function of interest. However, in many cases **the explicit functional form to be considered for minimization is known**, thus suggesting that derivatives may help in finding optimization solutions.

Indeed, in simple one-dimensional problems for finding the minimum of $f(x) = 0$, it is well known that a minimum is found when $f'(x) = 0 \text{ and } f''(x) > 0$. A maximum can be found when $f'(x) = 0 \text{ and } f''(x) < 0$. Such ideas are easily integrated into an optimization algorithm.

To begin, we generalize the concept of a minimum or maximum, i.e. an extremum(极值) for a **multidimensional function(多维函数)** *f(x)*. At an extremum, the gradient must be zero so that

$$\nabla f(\mathbf{x}) = 0$$

Unlike the one-dimensional case, there is no simple second derivative test to apply to determine if the extremum point is a minimum or maximum. The idea behind gradient descent, or steepest descent(最速下降), is **to use the derivative information as the basis of an iterative algorithm that progressively moves closer and closer to the minimum point** *f(x) = 0*.

To illustrate how to proceed in practice, consider the simple example two-dimensional surface (二维平面)

$$f(x, y) = x^2 + 3y^2$$

which has the minimum located at the origin *(x, y) = 0*.

The gradient for this function can be easily computed

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial x}\hat{\mathbf{x}} + \frac{\partial f}{\partial y}\hat{\mathbf{y}} = 2x\hat{\mathbf{x}} + 6y\hat{\mathbf{y}}$$

where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are unit vectors (单位向量) in the *x*- and *y*-directions, respectively.

Figure 5.3 illustrates the gradient (steepest) descent algorithm. At the initial guess point, the gradient $\nabla f(x)$ can be computed. This gives the gradient descent towards the minimum point of $f(x)$, i.e. the minimum is located in the direction given by $-\nabla f(x)$. Note that the gradient does not point at the minimum, but rather gives the steepest path for minimizing $f(x)$. **The geometry (几何) of the steepest descent suggests the construction of an algorithm whereby the next point of iteration is picked by following the steepest descent so that**

$$\xi(\tau) = \mathbf{x} - \tau \nabla f(\mathbf{x})$$

where the parameter τ dictates how far to move along the gradient descent curve.

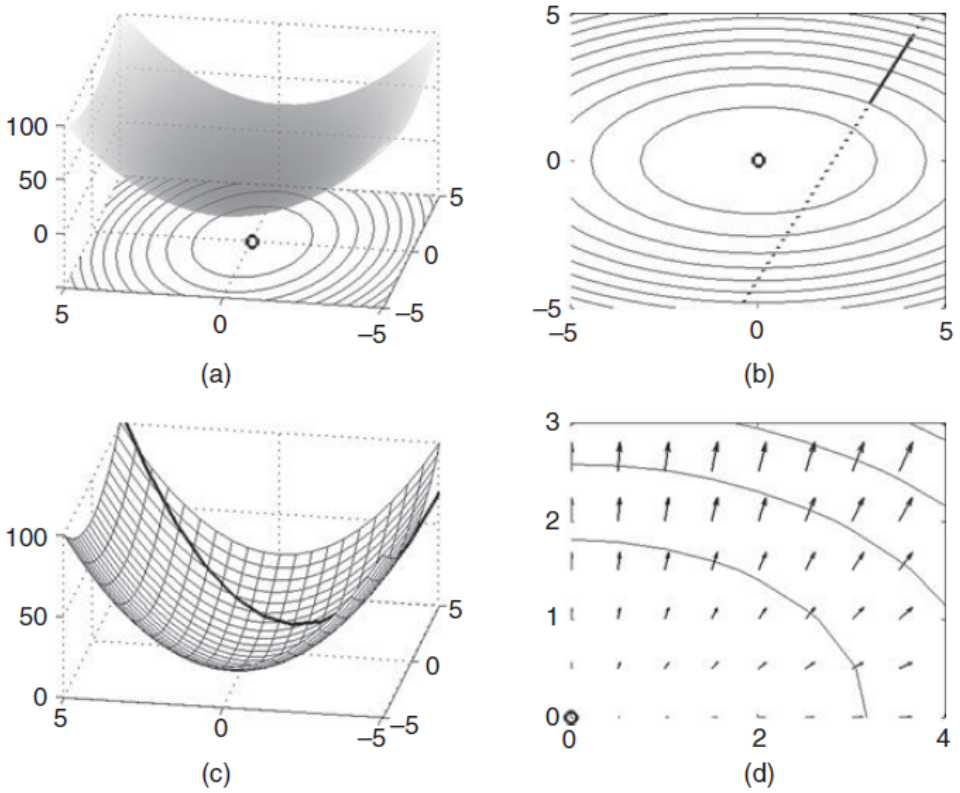


Figure 5.3: Graphical depiction of the gradient descent algorithm. The surface $f(x, y) = x^2 + 3y^2$ is plotted along with its contour lines. In (a), the surface is plotted along with the contour plot beneath. In (b), the gradient is calculated at the point $(x, y) = (3, 2)$. The gradient, which points away from the mininum, is plotted with the dark bolded line while the gradient line through $(x, y) = (3, 2)$ is plotted with a dotted line. The gradient descent moves along the steepest line of descent as shown in panel (c). Once the bottom of the descent curve is reached, a new descent path is picked. Panel (d) shows the overall gradient of the surface in the upper right quadrant.

Figure 5.3(c) shows that the gradient descent curves gives a descent path that eventually reaches bottom and starts to go back up again. In gradient descent, it is crucial to determine when this bottom is reached so that the algorithm is always going downhill in an optimal way. This requires the determination of the correct value of τ in the algorithm.

To compute the value of τ , consider the construction of a new function

$$F(\tau) = f(\xi(\tau))$$

which must be minimized now as a function of τ . This is accomplished by computing $dF/d\tau = 0$. Thus one finds

$$\frac{\partial F}{\partial \tau} = -\nabla f(\xi)\nabla f(\mathbf{x}) = 0$$

The geometrical interpretation (几何解释) of this result is the following: $\nabla f(x)$ is the gradient direction of the current iteration point and $\nabla f(\xi)$ is the gradient direction of the future point, thus τ is **chosen so that the two gradient directions are orthogonal (互相垂直)**.

For the example given above with $f(x, y) = x^2 + 3y^2$, we can easily compute this conditions as follows:

$$\xi = \mathbf{x} - \tau \nabla f(\mathbf{x}) = (1 - 2\tau)x\ \hat{\mathbf{x}} + (1 - 6\tau)y\ \hat{\mathbf{y}}$$

This is then used to compute

$$F(\tau) = f(\xi(\tau)) = (1 - 2\tau)^2 x^2 + 3(1 - 6\tau)^2 y^2$$

whereby its derivative with respect to τ gives

$$F'(\tau) = -4(1 - 2\tau)x^2 - 36(1 - 6\tau)y^2$$

Setting $F'(\tau) = 0$ then gives

$$\tau = \frac{x^2 + 9y^2}{2x^2 + 54y^2}$$

as the optimal descent step length.

In what follows, we develop a MATLAB code to perform the gradient descent search for the function $f(x, y) = x^2 + 3y^2$.

```
x(1)=3; y(1)=2; %initial guess
f(1)=x(1)2+3*y(1)2; %initial function value
forj=1:100
   tau=(8(j)^2+9*y(j)^2)/(2*x(j)^2+54*y(j)^2):
   x(j+1)=(1-2*tau)*x(j);%update values
   y(j+1)=(1-6*tau)*y(j);
   f(j+1)=x(j+1)^2+3*y(j+1)^2;

   if abs(f(j+1)-f(j))<10^(-6) %check convergence
      break
   end
end
```

The above algorithm converges in only 11 iteration steps to the minimal solution (see Fig. 5.4). Interestingly enough, if a simple radially symmetric function(径向对称函数) is considered, then the gradient descent converges in a single iteration since the gradient descent would point directly at the minimum.

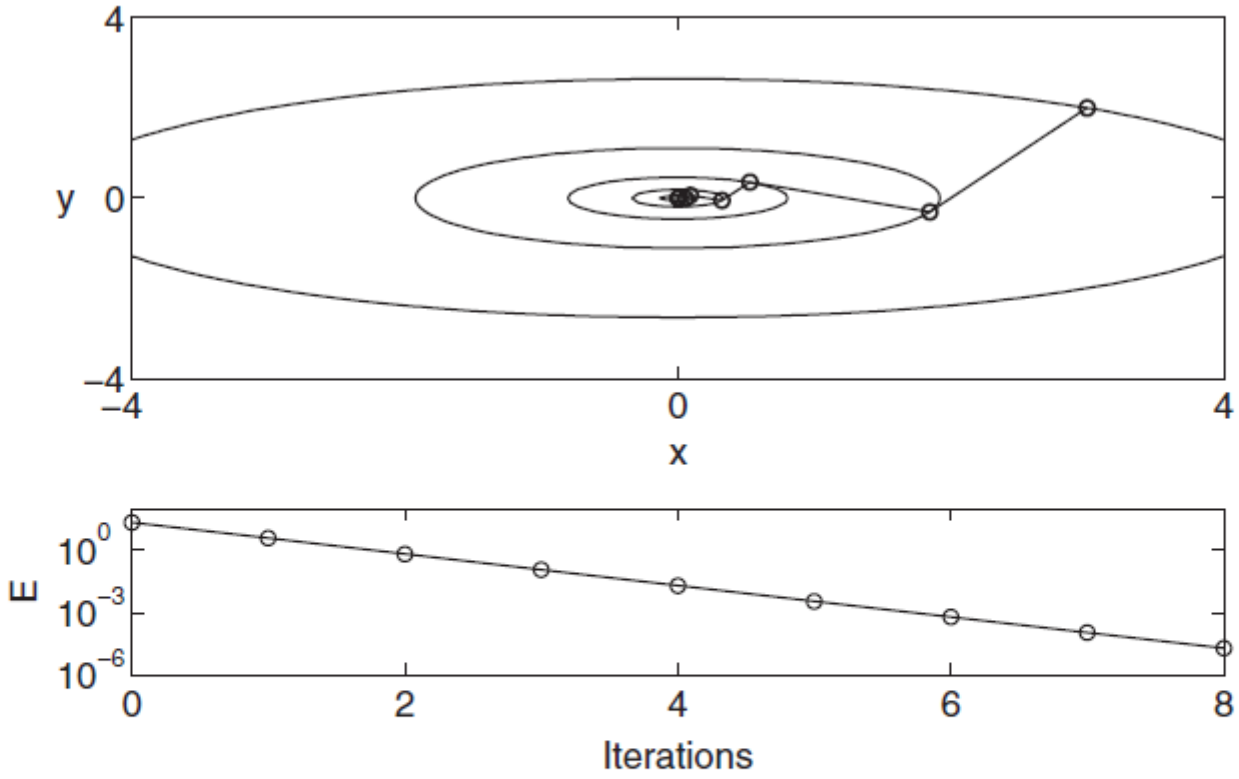


Figure 5.4: Gradient descent algorithm applied to the function $f(x, y) = x^2 + 3y^2$. In the top panel, the contours are plotted for each successive value (x, y) in the iteration algorithm given the initial guess $(x, y) = (3, 2)$. Note the orthogonality of each successive gradient in the steepest descent algorithm. The bottom panel demonstrates the rapid convergence and error (E) to the minimum (optimal) solution.

The above algorithm assumes a line search algorithm (直线查找法) to find an optimal value of τ . In particular, the value of τ picked here is optimal in the sense that a given line search is conducted so that the minimum of the gradient direction is picked as the next iteration point.

However, this is not a requirement. In fact, one can simply choose a fixed value of τ for stepping forward along the gradient direction. Figure 5.5 demonstrates this case for $\tau = 0.1$. This method also converges to the solution, however at a much slower rate. Such a method may be favorable in a case where the steepest descent algorithm *zig-zags* a large amount in trying to make the projective steps orthogonal. This can happen in cases where long-valley type structures exist in the function we are trying to minimize.

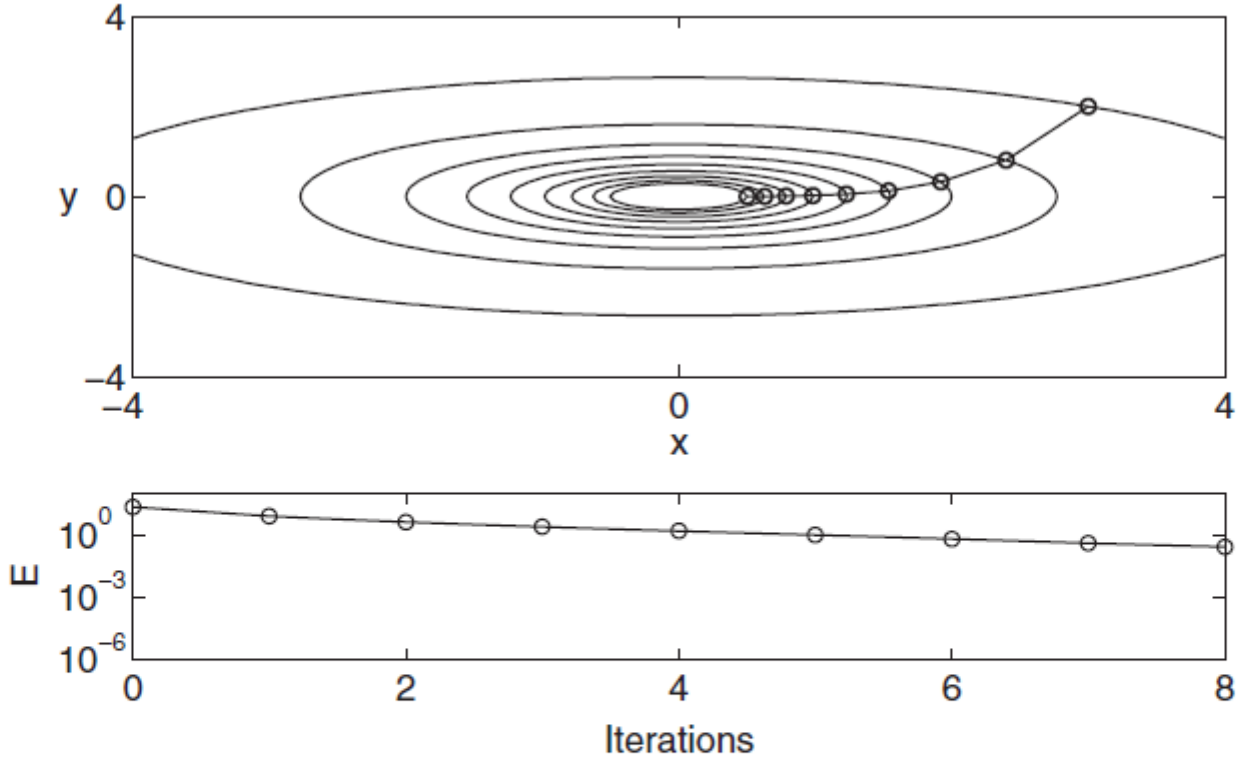


Figure 5.5: Gradient descent algorithm applied to the function $f(x, y) = x^2 + 3y^2$ with a fixed $\tau = 0.1$. In the top panel, the contours are plotted for each successive value (x, y) in the iteration algorithm given the initial guess $(x, y) = (3, 2)$. In this case, successive gradients are no longer orthogonal. The convergence and error (E) to the minimum (optimal) solution is slower with this line search method of a fixe value of τ.

fminsearch

Although not based upon gradient descent algorithms, the *fminsearch* algorithm in MATLAB is a generic, nonlinear unconstrained optimization method based upon the *Nelder–Mead* simplex method. We have already used this method as a means of doing nonlinear curve fitting. In that case, the objective function was the E_2 error which was to be minimized.

As a second example of this technique, consider once again a set of data that we wish to fit with the function $f(x) = A\cos(Bx) + C$ where A, B and C are the variables to be chosen for minimizing the error.

Our objective function in this case is the least-square error $E_2 = \sqrt{(1/N)\sum |f(x_j) - y_j|^2}$.

Thus we only need to consider minimizing $\sum |f(x_j) - y_j|^2$ with respect to A, B and C to achieve our goal.

The following code performs the optimization process with initial guesses given by (A, B, C) = (12, π/12, 63).

```
c=fminsearch('datafit',[12 pi/12 63]); %optimization
```

The function temp fit is given by datafit.m

```
function e2=tempfit(c)

x=1:24;
y=[75 77 76 73 69 68 63 59 57 55 54 52 50 ...
    50 49 49 49 50 54 56 59 63 67 72];

e2=sqrt(sum((c(1)*cos(c(2)*x)+c(3)-y).^2)/24);
```

The algorithm will rapidly converge to new values of the vector *c* which contains the updated and optimal value of A, B and C. To plot the results and compare the fit (see Fig. 5.6), the following code is used:

```
t=1:24; %raw data
tem=[75 77 76 73 69 68 63 59 57 55 54 52 ...
     50 50 49 49 49 50 54 56 59 63 67 72];
tt=1:0.01:24;
yfit=(c(1)*cos(c(2)*tt)+c(3)).';

plot (t,tem,'ko',tt,yfit,'k-')
```

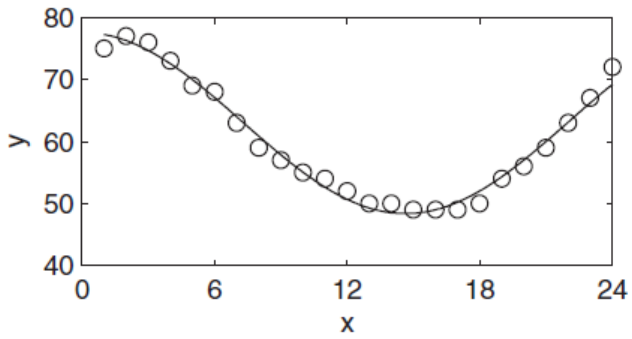


Figure 5.6: Minimization algorithm **fminsearch** used for curve fitting to a nonlinear function. The dots are the original data points and the solid line is the least-square fit. In this case, the least-square error E_2 is the objective function.

二、最速梯度下降法

下面先给出最速梯度下降法的计算步骤：

第 1 步　选取初始点 x^0，给定终止误差 $\varepsilon > 0$，令 $k := 0$；

第 2 步　计算 $\nabla f(x^k)$，若 $\|\nabla f(x^k)\| \le \varepsilon$，停止迭代.输出 x^k.否则进行第三步；

第 3 步　取 $p^k = -\nabla f(x^k)$；

第 4 步　进行一维搜索，求 t_k，使得

$$f(x^k + t_k p^k) = \min_{t \ge 0} f(x^k + t p^k)$$

令 $x^{k+1} = x^k + t_k p^k$，$k := k+1$，转第 **2** 步。

由以上计算步骤可知，最速下降法迭代终止时，求得的是目标函数**驻点的一个近似点。**

其中确定最优步长 t_k 的方法如下：

方法一：采用任一种一维寻优法

此时的 $f(x^k - t\nabla f(x^k))$ 已成为步长 t 的一元函数，故可用任何一种一维寻优法求出 t_k，即

$$f(x^{k+1}) = f(x^k - t_k \nabla f(x^k)) = \min_{t} f(x^k - t\nabla f(x^k))$$

方法二：微分法

因为

$$tf(x^k - t\nabla f(x^k)) = \varphi(t)$$

所以，一些简单情况下，可令

$$\varphi'(t) = 0$$

以解出近似最优步长 t_k 的值。

三、最速梯度下降法直观理解

在上面给出了最速梯度下降法的计算步骤，这里给出它的一些直观理解。

第一步：

第 1 步　选取初始点 x^0，给定终止误差 $\varepsilon > 0$，令 $k := 0$；

第 2 步　计算 $\nabla f(x^k)$，若 $\|\nabla f(x^k)\| \le \varepsilon$，停止迭代，输出 x^k，否则进行第三步；

第 3 步　取 $p^k = -\nabla f(x^k)$；

第 4 步　进行一维搜索，求 t_k，使得

$$f(x^k + t_k p^k) = \min_{t \ge 0} f(x^k + t p^k)$$

令 $x^{k+1} = x^k + t_k p^k$，$k := k+1$，转第 **2** 步。

第一步就是迭代法的初始点选择。

第二步：

第 1 步　选取初始点 x^0，给定终止误差 $\varepsilon > 0$，令 $k := 0$；

第 2 步　计算 $\nabla f(x^k)$，若 $\|\nabla f(x^k)\| \le \varepsilon$，停止迭代，输出 x^k，否则进行第三步；

第 3 步　取 $p^k = -\nabla f(x^k)$；

第 4 步　进行一维搜索，求 t_k，使得

$$f(x^k + t_k p^k) = \min_{t \ge 0} f(x^k + t p^k)$$

令 $x^{k+1} = x^k + t_k p^k$，$k := k+1$，转第 **2** 步。

可能有童鞋问这里的第二步的迭代终止条件为什么是 $||df(x^k) \leq \varepsilon||$?

这是因为根据下面这个定理：

定理 2　设 $f: R^n \to R^1$ 在点 $x^* \in R^n$ 处可微。若 x^* 是无约束问题的局部最优解，则

$$\nabla f(x^*) = 0$$

由数学分析中我们已经知道，使 $\nabla f(x) = 0$ 的点 x 为函数 f 的驻点或平稳点。函数 f 的一个驻点可以是极小点；也可以是极大点；甚至也可能既不是极小点也不是极大点，此时称它为函数 f 的鞍点。以上定理告诉我们，x^* 是无约束问题的的局部最优解的必要条件是：x^* 是其目标函数 f 的驻点。

也就是说，我们最终如果到达了局部最优解的话，求出来的梯度值是为0的，也就是说该点梯度为0是该点是局部最优解的必要条件。

所以我们的终止条件就是到达某处的梯度为0，在一些条件不是太苛刻的情况下，我们也可以不让它严格为0，只是逼近于0即可。这就是第二步的解释。

第三步：

第 1 步　选取初始点 x^0，给定终止误差 $\varepsilon > 0$，令 $k := 0$；

第 2 步　计算 $\nabla f(x^k)$，若 $\|\nabla f(x^k)\| \leq \varepsilon$，停止迭代，输出 x^k，否则进行第三步；

第 3 步　取 $p^k = -\nabla f(x^k)$；

第 4 步　进行一维搜索，求 t_k，使得

$$f(x^k + t_k p^k) = \min_{t \geq 0} f(x^k + t p^k)$$

令 $x^{k+1} = x^k + t_k p^k$，$k := k+1$，转第 2 步。

这步在是在选取迭代方向，也就是从当前点迭代的方向。这里选取当前点的梯度负方向，为什么选择这个方向，是因为梯度的负方向是局部下降最快的方向，这里不详细证明，可以参考我以前的一个回答：为什么梯度反方向是函数值局部下降最快的方向？

第四步：

第 1 步　选取初始点 x^0，给定终止误差 $\varepsilon > 0$，令 $k := 0$；

第 2 步　计算 $\nabla f(x^k)$，若 $\|\nabla f(x^k)\| \le \varepsilon$，停止迭代，输出 x^k，否则进行第三步；

第 3 步　取 $p^k = -\nabla f(x^k)$；

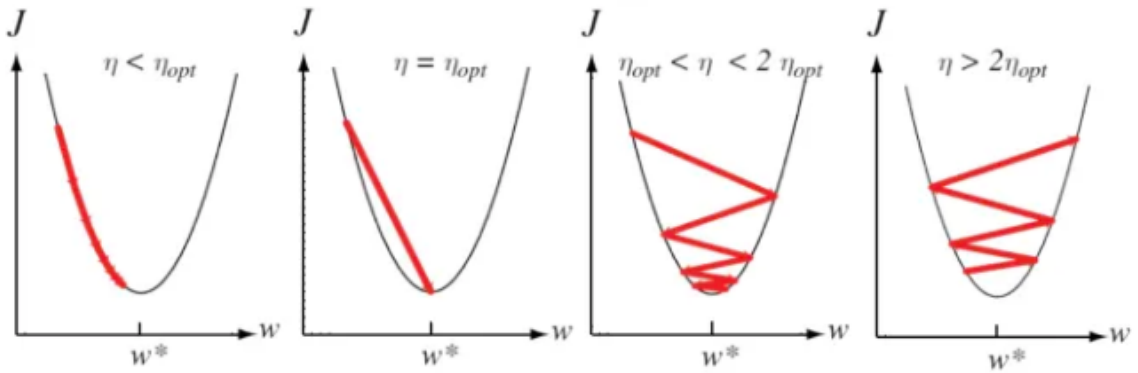
第 4 步　进行一维搜索，求 t_k，使得

$$f(x^k + t_k p^k) = \min_{t \ge 0} f(x^k + t p^k)$$

令 $x^{k+1} = x^k + t_k p^k$，$k := k+1$，转第 **2** 步。

第四步也是非常重要的，因为在第三步我们虽然确定了迭代方向，并且知道这个方向是局部函数值下降最快的方向，但是还没有确定走的步长，如果选取的步长不合适，也是非常不可取的，下面会给出一个例子图，***那么第四步的作用就是在确定迭代方向的前提上，确定在该方向上使得函数值最小的迭代步长。***

下面给出迭代步长过大过小都不好的例子图：



四、最速梯度下降法实例

例 1 $\min f(x)=x_1-x_2+2x_1^2+2x_1x_2+x_2^2$ 给定初始点 $X^{(1)}=(0,0)^T$

解：目标函数 $f(x)$ 的梯度 $\nabla f(x)=\begin{bmatrix}\frac{\partial f(x)}{\partial(x_1)}\\ \frac{\partial f(x)}{\partial(x_2)}\end{bmatrix}=\begin{bmatrix}1+4x_1+2x_2\\ -1+2x_1+2x_2\end{bmatrix}$

$\nabla f(X^{(1)})=\begin{bmatrix}1\\-1\end{bmatrix}$ 令搜索方向 $d^{(1)}=-\nabla f(X^{(1)})=\begin{bmatrix}-1\\1\end{bmatrix}$ 再从 $X^{(1)}$ 出发，沿 $d^{(1)}$ 方向作一维寻优，令步长变量为 λ，最优步长为 λ_1，则有 $X^{(1)}+\lambda d^{(1)}=\begin{bmatrix}0\\0\end{bmatrix}+\lambda\begin{bmatrix}-1\\1\end{bmatrix}=\begin{bmatrix}-\lambda\\\lambda\end{bmatrix}$

故 $f(x)=f(X^{(1)}+\lambda d^{(1)})=(-\lambda)-\lambda+2(-\lambda)^2+2(-\lambda)\lambda+\lambda^2=\lambda^2-2\lambda=\varphi_1(\lambda)$

令 $\varphi_1'(\lambda)=2\lambda-2=0$ 可得 $\lambda_1=1$ $X^{(2)}=X^{(1)}+\lambda_1 d^{(1)}=\begin{bmatrix}0\\0\end{bmatrix}+\begin{bmatrix}-1\\1\end{bmatrix}=\begin{bmatrix}-1\\1\end{bmatrix}$ 求出 $X^{(2)}$ 点之后，与上类似地，进行第二次迭代：$\nabla f(X^{(2)})=\begin{bmatrix}-1\\-1\end{bmatrix}$ 令 $d^{(2)}=-\nabla f(X^{(2)})=\begin{bmatrix}1\\1\end{bmatrix}$

令步长变量为 λ，最优步长为 λ_2，则有

$X^{(2)}+\lambda d^{(2)}=\begin{bmatrix}-1\\1\end{bmatrix}+\lambda\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}\lambda-1\\\lambda+1\end{bmatrix}$

故

$f(x)=f(X^{(2)}+\lambda d^{(2)})=(\lambda-1)-(\lambda+1)+2(\lambda-1)^2+2(\lambda-1)(\lambda+1)+(\lambda+1)^2=5\lambda^2-2\lambda-1=\varphi_2(\lambda)$

令 $\varphi_2'(\lambda)=10\lambda-2=0$ 可得 $\lambda_2=\frac{1}{5}$ $X^{(3)}=X^{(2)}+\lambda_2 d^{(2)}=\begin{bmatrix}-1\\1\end{bmatrix}+\frac{1}{5}\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}-0.8\\1.2\end{bmatrix}$

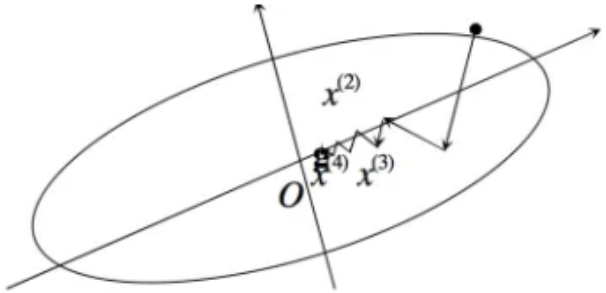
$\nabla f(X^{(3)})=\begin{bmatrix}0.2\\-0.2\end{bmatrix}$ 此时所达到的精度 $\|\nabla f(X^{(3)})\|\approx 0.2828$

因此我们找到了近似最优解： X^3 ，然后将 X^3 带入 $f(x)$ 中，即可得到要求的最小值。

五、最速下降法的缺点

需要指出的是，某点的负梯度方向，通常只是在该点附近才具有这种最速下降的性质。

在一般情况下，当用最速下降法寻找极小点时，其搜索路径呈直角锯齿状（如下图），***在开头几步，目标函数下降较快；但在接近极小点时，收敛速度长久不理想了。特别适当目标函数的等值线为比较扁平的椭圆时，收敛就更慢了。***



因此，在实用中常用最速下降法和其他方法联合应用，在前期使用最速下降法，而在接近极小值点时，可改用收敛较快的其他方法。

六、最速下降法与梯度下降法的区别

来源：最速下降法与梯度下降法

准确来说，它们并不是完全等价。

对于梯度下降法，我们需要预先设定步长α。

$$x_{i+1} = x_i - \alpha \nabla f_{x_i}$$

而最速下降法的这个步长α_k是通过一个优化函数计算得到的。

$$\alpha_k = \mathrm{argmin}_{\alpha_k} f(x_i - \alpha_k \nabla f_{x_i})$$

1： 梯度下降法

1.1 梯度

导数我们都非常熟悉，既可以表示某点的切线斜率，也可以表示某点变化率，公式如下表示：

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

当函数是多元时，倒数就变成了偏导数： $f_x(x,y)$ 表示当 y 不变时， $f(x,y)$ 沿着 x 轴的变化变化率； $f_y(x,y)$ 表示当 x 不变时， $f(x,y)$ 沿着 y 轴的变化变化率。

但是多元函数是一个平面，方向有很多， x 轴 y 轴只是其中两个方向而已，假如我们需要其他方向的变化率怎么办呢？因此方向导数就有用了，顾名思义，方向导数可以表示任意方向的倒数。

假如二次函数 $f(x,y)$ ，方向 $u = cos\theta\, i + sin\theta\, j$ （为单位向量）的方向导数公式如下： $\lim_{t \to 0} \frac{f(x+t\ cos\theta, y+t\ sin\theta) - f(x)}{t}$ ，记为 $D_u f$ 。其中

$D_u f = f_x(x,y)cos\theta + f_y(x,y)sin\theta = [\, f_x(x,y) \quad f_y(x,y)\,] \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix}$ 。我们记为：

$D_u f = \mathbf{A} \times \mathbf{I} = |\mathbf{A}||\mathbf{I}|cos\alpha$ ，其中 α 是两向量的夹角。我们可以知道，当 α 为0时，方向导数 $D_u f$ 达到最大，此时的方向导数即为梯度。

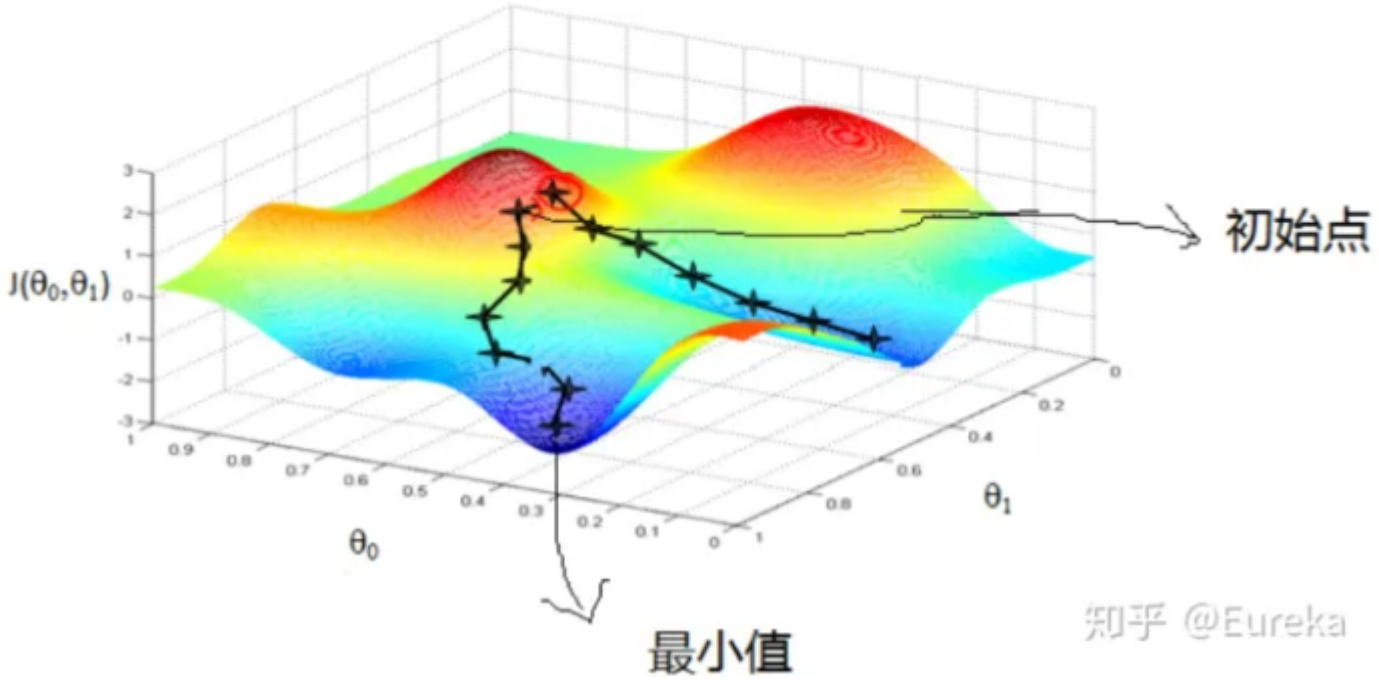
更多的关于梯度的知识点可以看里面的回答：如何直观形象的理解方向导数与梯度以及它们之间的关系？

从几何意义上来说，梯度向量就是函数变化增加最快的地方。具体来说，对于函数 $f(x,y)$,在点 (x_0, y_0) ，沿着梯度向量的方向就是 $\begin{bmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial y_0} \end{bmatrix}$ 的方向是 $f(x,y)$ 增加最快的地方（还记得梯度怎么来的吗？方向导数的最大值，粗暴点，就是导数最大值）。或者说，沿着梯度向量的方向，更加容易找到函数的最大值。反过来说，沿着梯度向量相反的方向（去负号），则就是更加容易找到函数的最小值。

1.2 梯度下降的直观解释

比如我们在一座大山上的某处位置，由于我们不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。这样一步步的走下去，一直走到觉得我们已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山峰低处（局部最小值）。

从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。



1.3 梯度下降算法法

梯度下降法(Gradient descent)或最速下降法(steepest descent)是求解无约束最优化问题的一种常用的、实现简单的方法。

假设 $f(x)$ 是 R^n 上具有一阶连续偏导数的函数。求解

$$\min_{x \in R^n} f(x)$$

无约束最优化问题。 f^* 表示目标函数 $f(x)$ 的极小点。

梯度下降法是一种迭代算法。选取适当的初值 $x^{(0)}$ ，不断迭代，更新 x 的值，进行目标函数的极小化，直到收敛。由于负梯度方向是使函数下降最快的的方向，在迭代的每一步，以负梯度方向更新 x 的值，从而达到减少函数值的目的。

第 $k+1$ 次迭代值

$$x^{(k+1)} \leftarrow x^{(k)} + \lambda_k p_k$$

其中， p_k 是搜索方向，取负梯度方向 $p_k = -\nabla f\left(x^{(k)}\right)$ ， λ_k 是步长，由一维搜索确定，即 λ_k 使得

$$f\left(x^{(k)} + \lambda_k p_k\right) = \min_{\lambda \geq 0} f\left(x^{(k)} + \lambda p_k\right)$$

梯度下降算法：

输入：目标函数 $f(x)$，梯度函数 $g\left(x^{(k)}\right)=\nabla f\left(x^{(k)}\right)$，计算精度 ε

输出： $f(x)$ 的极小点 x^*

1. 取初值 $x^{(0)} \in R^n$，置 $k=0$
2. 计算 $f\left(x^{(k)}\right)$
3. 计算梯度 $g_k=g\left(x^{(k)}\right)$，当 $\|g_k\|<\varepsilon$ 时，停止迭代，令 $x^*=x^{(k)}$；否则，令 $p_k=-g\left(x^{(k)}\right)$，求 λ_k，使

$$f\left(x^{(k)}+\lambda_k p_k\right)=\min_{\lambda \geq 0} f\left(x^{(k)}+\lambda p_k\right)$$

4. 置 $x^{(k+1)}=x^{(k)}+\lambda_k p_k$，计算 $f\left(x^{(k+1)}\right)$

当 $\|f\left(x^{(k+1)}\right)-f\left(x^{(k)}\right)\|<\varepsilon$ 或 $\|x^{(k+1)}-x^{(k)}\|<\varepsilon$ 时，停止迭代，令 $x^*=x^{k+1}$

5. 否则，置 $k=k+1$，转3.

1.4 梯度下降的算法调优

在使用梯度下降时，需要进行调优。哪些地方需要调优呢？

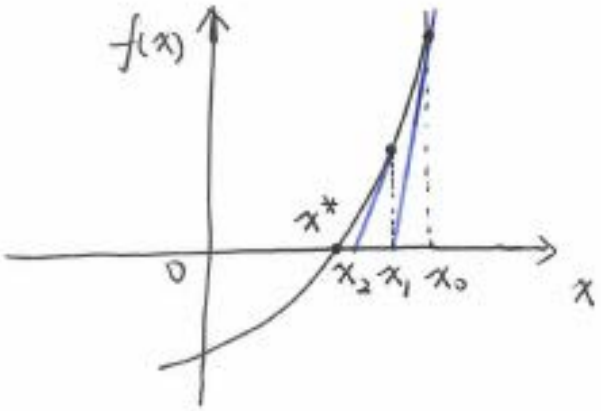
1. **算法的步长选择。**步长取值取决于数据样本，可以多取一些值，从大到小，分别运行算法，看看迭代效果，如果损失函数在变小，说明取值有效，否则要增大步长。步长太大，会导致迭代过快，甚至有可能错过最优解。步长太小，迭代速度太慢，很长时间算法都不能结束。所以算法的步长需要多次运行后才能得到一个较为优的值。
2. **算法参数的初始值选择。**初始值不同，获得的最小值也有可能不同，因此梯度下降求得的只是局部最小值；当然如果损失函数是凸函数则一定是最优解。由于有局部最优解的风险，需要多次用不同初始值运行算法，关键损失函数的最小值，选择损失函数最小化的初值。
3. **归一化。**由于样本不同特征的取值范围不一样，可能导致迭代很慢，为了减少特征取值的影响，可以对特征数据归一化。

§ Unconstrained Opt. (Newton's Method) <Lec10> Page 1

Problem: $\min_x f(x) \quad (x \in \mathbb{R}^n)$ (P)

where f is convex, twice differentiable, and $\operatorname{dom} f = \mathbb{R}^n$.

Prerequiste: Before solving (P), we first consider a simple root-finding problem $f(x) = 0 \quad (x \in \mathbb{R})$.



Tangent line at $(x_k, f(x_k))$:

$$f(x) = f(x_k) + f'(x_k)(x - x_k)$$

Set $f(x) = 0$, yielding

$$\boxed{x_{k+1} = x_k - f'(x_k)^{-1} f(x_k)}$$

"Newton(-Raphson) Method"

Page 2

For (P), consider the optimality condition:

$$\nabla f(x) = 0 \qquad (*)$$

Applying "Newton's Method" (root-finding) to (*),

let $F(x) = \nabla f(x)$,

$$x_{k+1} = x_k - \nabla F(x_k)^{-1} F(x_k)$$

$$= x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

called "Newton's Method" for $\min_x f(x)$.

(or "pure Newton's Method")

In practice, use "damped Newton's Method" (i.e. Newton's method)

$$x_{k+1} = x_k - t\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

Step size t is typically chosen by "backtracking line search", with $0 < \alpha < 1/2$, $0 < \beta < 1$.

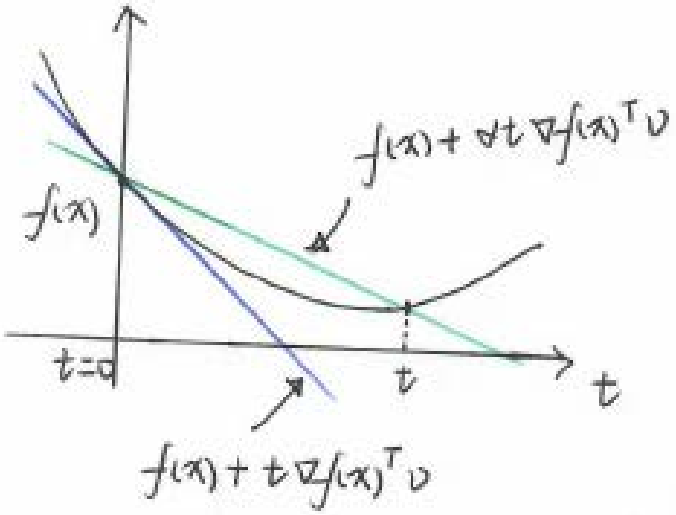
Recall:

Start with $t = 1$ and while

$$f(x + tv) > f(x) + \alpha t \nabla f(x)^T v$$

we shrink $t = \beta t$, else perform the Newton update.

($v = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$)



* Newton's Method

---

given a starting point $x \in \text{dom} f$, tolerance $\epsilon > 0$

repeat

1. Compute the Newton step and decrement

$$\Delta x = -\nabla^2 f(x)^{-1} \nabla f(x), \quad \lambda^2 = \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$$

2. Stopping criterion

quit if $\lambda^2/2 < \epsilon$

3. Line search. Choose step size t by backtracking line search.

4. Update $x = x + t\Delta x$

Now let's understand the stopping criteria $\lambda^2/2 < \epsilon$, and λ.

Alternative viewpoint on Newton's method.

Recall the motivation for G.D. step at x:

minimize the quadratic approximation

$$f(y) \approx f(x) + \nabla f(x)^T (y-x) + \frac{1}{2t}\|y-x\|^2$$

over y, yielding the update $x^+ = x - t\nabla f(x)$.

Newton's method use a better quadratic approximation:

$$f(y) \approx \underbrace{f(x) + \nabla f(x)^T (y-x) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x)}_{\text{记为 } g(y)}$$

minimize over y & yielding

$$x^+ = x - \left(\nabla^2 f(x)\right)^{-1} \nabla f(x).$$

/* $\nabla g(y) = 0 \Rightarrow \nabla f(x) + \nabla^2 f(x)(y-x) = 0$ */

At point x, the "Newton decrement" is defined as

$$\lambda(x) = \left(\nabla f(x)^T \left(\nabla^2 f(x)\right)^{-1} \nabla f(x)\right)^{1/2}$$

This relates to the difference between $f(x)$ and $\min_y h(y)$:

$$f(x) - \min_y h(y)$$

$$= f(x) - \left[f(x) + \nabla f(x)^T \left(-(\nabla^2 f(x))^{-1} \nabla f(x)\right) + \frac{1}{2} \nabla f(x)^T (\nabla^2 f(x))^{-1} \nabla^2 f(x) (\nabla^2 f(x))^{-1} \nabla f(x) \right]$$

$$= \frac{1}{2} \nabla f(x)^T [\nabla^2 f(x)]^{-1} \nabla f(x) = \frac{1}{2} \lambda(x)^2$$

Thus, we can think of $\lambda^2(x)/2$ as an approximated upper bound on the suboptimality gap $f(x) - f^*$ (where $f^* = \min_x f(x)$).

Newton step

$$\Delta x_{\mathrm{nt}} = -\nabla^2 f(x)^{-1} \nabla f(x)$$

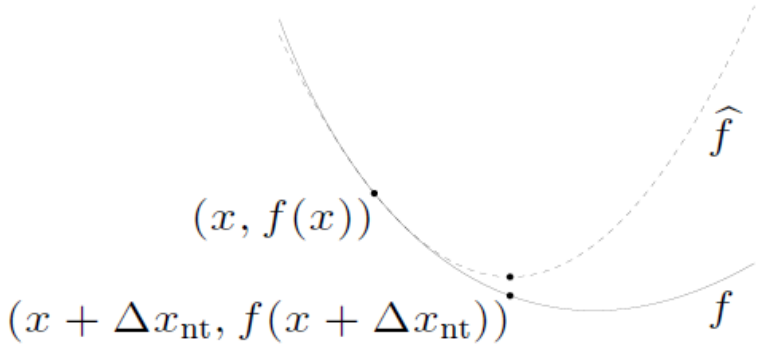
interpretations

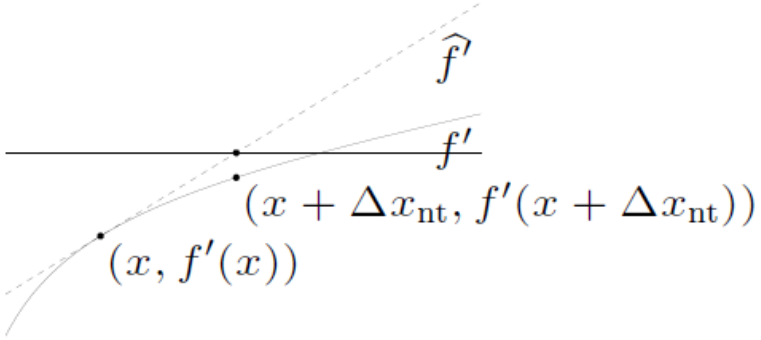
- $x + \Delta x_{\mathrm{nt}}$ minimizes second order approximation

$$\widehat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v$$

- $x + \Delta x_{\mathrm{nt}}$ solves linearized optimality condition

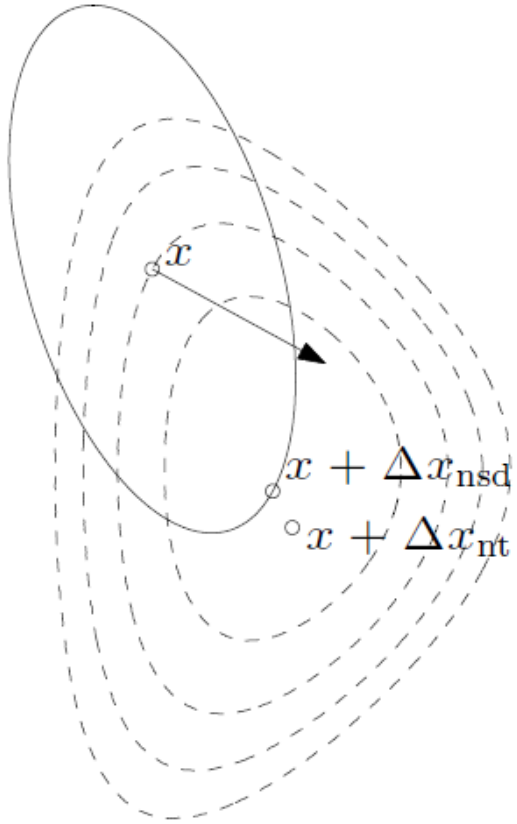
$$\nabla f(x+v) \approx \nabla \widehat{f}(x+v) = \nabla f(x) + \nabla^2 f(x) v = 0$$





- Δx_{nt} is steepest descent direction at x in local Hessian norm

$$\|u\|_{\nabla^2 f(x)} = \left(u^T \nabla^2 f(x) u\right)^{1/2}$$



dashed lines are contour lines of f; ellipse is $\{x + v \mid v^T \nabla^2 f(x) v = 1\}$

arrow shows $-\nabla f(x)$

Newton decrement

$$\lambda(x) = \left(\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)\right)^{1/2}$$

a measure of the proximity of x to $x^\star$

properties

- gives an estimate of $f(x) - p^\star$, using quadratic approximation $\widehat{f}$:

$$f(x) - \inf_y \widehat{f}(y) = \frac{1}{2}\lambda(x)^2$$

- equal to the norm of the Newton step in the quadratic Hessian norm

$$\lambda(x) = \left(\Delta x_{\rm nt}^T \nabla^2 f(x) \Delta x_{\rm nt}\right)^{1/2}$$

- directional derivative in the Newton direction: $\nabla f(x)^T \Delta x_{\rm nt} = -\lambda(x)^2$
- affine invariant (unlike $\|\nabla f(x)\|_2$)

Newton's method

---

given a starting point $x \in \mathbf{dom}\, f$, tolerance $\epsilon > 0$.

repeat

1. *Compute the Newton step and decrement.*
   $$\Delta x_{\mathrm{nt}} := -\nabla^2 f(x)^{-1} \nabla f(x); \quad \lambda^2 := \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x).$$
2. *Stopping criterion.* **quit** if $\lambda^2/2 \leq \epsilon$.
3. *Line search.* Choose step size t by backtracking line search.
4. *Update.* $x := x + t\Delta x_{\mathrm{nt}}$.

---

affine invariant, $i.e.$, independent of linear changes of coordinates:

Newton iterates for $\tilde{f}(y) = f(Ty)$ with starting point $y^{(0)} = T^{-1}x^{(0)}$ are

$$y^{(k)} = T^{-1}x^{(k)}$$

Classical convergence analysis

assumptions

- f strongly convex on S with constant m
- $\nabla^2 f$ is Lipschitz continuous on S, with constant $L > 0$:

$$\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq L\|x - y\|_2$$

(L measures how well f can be approximated by a quadratic function)

outline: there exist constants $\eta \in (0, m^2/L)$, $\gamma > 0$ such that

- if $\|\nabla f(x)\|_2 \geq \eta$, then $f(x^{(k+1)}) - f(x^{(k)}) \leq -\gamma$
- if $\|\nabla f(x)\|_2 < \eta$, then

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2$$

damped Newton phase ($\|\nabla f(x)\|_2 \geq \eta$)

- most iterations require backtracking steps
- function value decreases by at least γ
- if $p^\star > -\infty$, this phase ends after at most $(f(x^{(0)}) - p^\star)/\gamma$ iterations

quadratically convergent phase ($\|\nabla f(x)\|_2 < \eta$)

- all iterations use step size $t = 1$
- $\|\nabla f(x)\|_2$ converges to zero quadratically: if $\|\nabla f(x^{(k)})\|_2 < \eta$, then

$$\frac{L}{2m^2}\|\nabla f(x^l)\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^k)\|_2\right)^{2^{l-k}} \leq \left(\frac{1}{2}\right)^{2^{l-k}}, \qquad l \geq k$$

conclusion: number of iterations until $f(x) - p^\star \leq \epsilon$ is bounded above by

$$\frac{f(x^{(0)}) - p^\star}{\gamma} + \log_2 \log_2(\epsilon_0/\epsilon)$$

- γ, ϵ_0 are constants that depend on m, L, $x^{(0)}$
- second term is small (of the order of 6) and almost constant for practical purposes
- in practice, constants m, L (hence γ, ϵ_0) are usually unknown
- provides qualitative insight in convergence properties ($i.e.$, explains two algorithm phases)

二、牛顿法思想

用目标函数的二阶泰勒展开近似该目标函数，通过求解这个二次函数的极小值来求解凸优化的搜索方向。

三、牛顿法推导

目标函数 $f(x)$ 在 x 处的二阶 Taylor 展开为

$$f(x+v) \approx f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v ,$$

当 x 固定时，v 取多少可以使 $f(x+v)$ 最小呢？可以考虑上式的右边，其为 v 的二次函数。

使上式两边对 v 求偏导

$$\frac{\partial}{\partial v} f(x+v) = \nabla f(x) + \nabla^2 f(x) v ,$$

当 $f(x+v)$ 最小时，$f(x+v)$ 对 v 的偏导为零，即

$$\nabla f(x) + \nabla^2 f(x) v = 0$$

则有

$$v = -\nabla^2 f(x)^{-1} \nabla f(x)$$

将这个方向作为搜索方向，这个方向被称作 Newton 步径，即

$$\Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x) ,$$

$\nabla^2 f(x)$ 即为函数 $f(x)$ 的 Hessian 矩阵，由凸函数的 Hessian 矩阵的正定性可知，除 $f(x)$ 到达极值点 $\nabla f(x)=0$ 外，有

$$\nabla f(x)^T \Delta x_{nt} = -\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) < 0$$

因此 Newton 步径是下降方向。

这种方法就是牛顿法(Newton 法)。

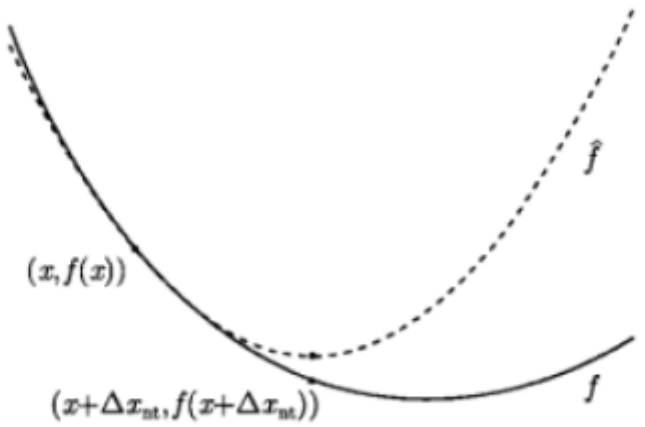


图1

图 1 大概说明了牛顿法的原理，实线是目标函数 $f(x)$，虚线是 $f(x)$ 在 x 处的 Taylor 二阶展开。

四、牛顿法和最速梯度下降法的区别

我们可以看出，牛顿法和最速梯度的不同就是在于最速梯度下降法的迭代方向是梯度的负方向，迭代步长根据一维搜索得到。

而牛顿法的迭代方向为上述推导的牛顿步径，迭代步长可以看为定值1。如下：

则有 $x^{(k+1)} = x^{(k)} - H_k^{-1}\nabla f(x^{(k)})$，而 $-H_k^{-1}\nabla f(x^{(k)})$ 则为点 $x^{(k)}$ 处的牛顿方向，也就是该点的下降方向。通过该公式进行迭代直至该点的梯度收敛，即其梯度的二阶范数$||\nabla f(x^{(k)})|| < \epsilon$。

五、牛顿法的优缺点

1. ***优点***是：对于二次正定函数，迭代一次即可以得到最优解，对于非二次函数，若函数二次性较强或迭代点已经进入最优点的较小邻域，则收敛速度也很快。
2. ***缺点***：

- 保证不了迭代方向是下降方向，***这就是致命的！***先看一个定理：

定理 **1** 设 $f: R^n \to R^1$ 在点 $\bar{x} \in R^n$ 处可微。若存在 $p \in R^n$，使

$$\nabla f(\bar{x})^T p < 0$$

则向量 p 是 f 在点 $\bar{x}$ 处的下降方向。

我们判断牛顿法的迭代方向是否朝着函数值下降的方向移动，也就是判断一下迭代方向和 当前点的梯度值做内积，如果小于0就说明是下降方向，我们来计算一下：

迭代方向和当前点的梯度做内积为:

$$\nabla f(x)^T \Delta x_{nt} = -\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$$

想让它小于0，也就是要求

$$-\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$$

大于0，这恰好是海塞矩阵正定的条件，也就是需要满足当前点的海塞矩阵是正定的，这个要求是很强的，因此并不能保证牛顿法的迭代方向是一定沿着函数值下降的方向。

换句话说就是不一定迭代能够收敛，后面的阻尼牛顿法会解决这个问题，牛顿法就到此为止了。

- 计算量相当复杂，除需计算梯度除外，还需要计算二阶偏导数矩阵和它的逆矩阵，计算量，存储量都很大，并且都以维数N的平方比增加，当N很大的时候，计算量的问题就更加突出。

线搜索算法（line search method）的每一次迭代都会计算出一个搜索方向 p_k，并决定在这个方向上移动的步长是多少。迭代公式如下：

$$x_{k+1} = x_k + \alpha_k p_k$$

式中，a_k 表示第 k 次迭代所移动的步长（step length），p_k 表示搜索方向或下降方向（descent direction）。而一个线搜索算法成功的关键便是同时在搜索方向 p_k 和步长 a_k 上的有效选择。

一般情况下，大多数的线搜索算法（line search method）都需要 p_k 是一个梯度下降方向，这样才能保证目标函数 f 沿着搜索方向是下降的。 线搜索方向 p_k 经常使用如下形式：

$$p_k^T \nabla f_k = p_k^T B_k^{-1} \nabla f_k < 0$$

根据B_k的不同形式，可以得到常用的最速下降法、牛顿法和拟牛顿法等三种不同的线性搜索算法：

- 最速下降法（Steepest Descent method, when B_k is identity matrix I）
- 牛顿法 （Newton's method, when B_k is Hessian $\nabla^2 f(x_k)$ ）
- 拟牛顿法 （quasi-Newton method, when B_k is an approximation to the Hessian）

3.1 步长（STEP LENGTH）

在计算步长（step length）α_k 的时候， 会面临一个性能与质量的平衡问题。α_k 最理想的选择是单变量函数（univariate function） $\phi(\alpha)$ 全局最小时的取值。单变量函数 $\phi(\alpha)$ 定义如下：

$$\phi(\alpha) = f(x_k + \alpha p_k), \alpha > 0$$

但是一般情况下，需要很长的时间才能确定这个理想的 α_k 取值。为了减少计算理想步长而带来的时间消耗，更实用的策略是使用不精确的线搜索（**inexact line search**）来确定一个步长，用最小的成本达到目标函数 f 充分减少（adequate reductions）的目的。

完成线搜索主要分两个步骤：

- 括号阶段，寻找步长 α_k 最可能的区间（A bracketing phase finds an interval containing desirable step lengths）
- 二分阶段，在可能的区间计算出好的 α_k （bisection or interpolation phase computes a good step length within this interval）

那如何计算出高质量的 α_k 来满足优化算法快速收敛的需求呢？可以通过WOLFE CONDITIONS 和 GOLDSTEIN CONDITIONS 两个约束条件来实现。

THE WOLFE CONDITIONS

Wolfe conditions 约束条件由sufficient decrease（3.6a）和 curvature conditions（3.6b）两个约束条件组成，具体如下：

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, (3.6a)$$

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k, (3.6b)$$

式中，$0 < c_1 < c_2 < 1$

THE GOLDSTEIN CONDITIONS

与 Wolfe conditions 约束一样，Goldstein conditions 可以保证步长 α 达到充分下降，同时也会避免步长不会太小的情况。约束条件如下：

$$f(x_k)+(1-c)\alpha_k\nabla f_k^T p_k \le f(x_k+\alpha_k p_k) \le f(x_k)+c\alpha_k\nabla f_k^T p_k, (3.11)$$

式中，$0<c<1/2$ 。第二个不等式是充分下降条件，而第一个不等式则用来约束步长。

Goldstein conditions 与Wolfe conditions相比的一个缺点是公式（3.11）中的第一个不等式可能排除 ϕ 的所有最小值。然而，Goldstein conditions 与Wolfe conditions有很多相似之处，它们的收敛理论也很相似。Goldstein conditions 常用于牛顿法，但不太适合需要保持正定 Hessian 近似的拟牛顿方法。（The Goldstein conditions are often used in Newton-type methods but are not well suited for quasi-Newton methods that maintain a positive definite Hessian approximation.）

SUFFICIENT DECREASE AND BACKTRACKING

从Wolfe conditions可以看出，仅用充分下降条件（3.6a）来确保算法在给定的搜索方向上取得合理进展是不够的。但是，如果线搜索算法适当地选择其候选步长，通过使用所谓的回溯方法（backtracking approach），便可以省去额外的条件(3.6b)并使用充分下降条件终止行搜索过程。基于其最基本的形式，回溯过程如下：

Algorithm 3.1 (Backtracking Line Search).

Choose $\bar{\alpha} > 0, \rho \in (0,1), c \in (0,1)$;

Set $\alpha \leftarrow \bar{\alpha}$;

repeat unti
$f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
$\alpha \leftarrow \rho\alpha$

end(repeat)

Terminate with $\alpha_k = \alpha$

上述流程在牛顿法和拟牛顿法中会选取 $\bar{\alpha} = 1$ 为初始步长，但在其他算法中可能有不同的值，如最速下降或则共轭梯度法。

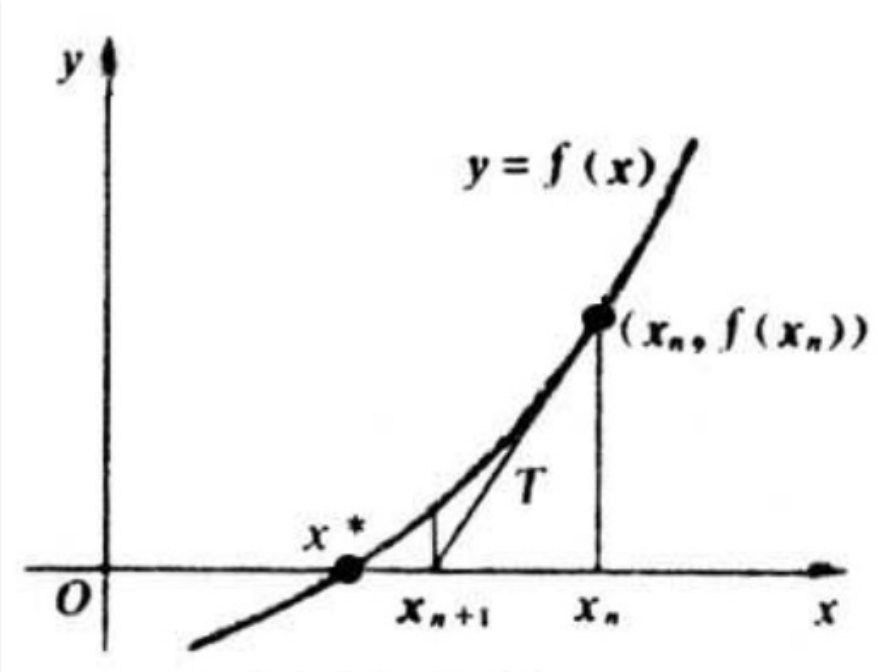
2： 牛顿法

一般来说，牛顿法主要应用在两个方面，1：求方程的根；2：最优化。

2.1 求解方程

并不是所有的方程都有求根公式，或者求根公式很复杂，导致求解困难。利用牛顿法，可以迭代求解。

原理是利用泰勒公式，在 x_0 处展开，且展开到一阶，即 $f(x)=f(x_0)+f'(x_0)(x-x_0)$ 。求解方程 $f(x)=0$ ，等价于 $f(x_0)+f'(x_0)(x-x_0)=0$ ，求解 $x=x_1=x_0-f(x_0)/f'(x_0)$ 。因为这是利用泰勒公式的一阶展开，$f(x)=f(x_0)+f'(x_0)(x-x_0)$ 处并不是完全相等，而是近似相等，这里求得的 x_1 并不能让 $f(x)=0$ ，只能说 $f(x_1)$ 的值比 $f(x_0)$ 的值更接近于0，于是迭代的想法就很自然了，可以进而推出 $x_{n+1}=x_n-f(x_n)/f'(x_n)$ ，通过迭代，这个式子必然在 $f(x^*)=0$ 的时候收敛。整个过程如下：



2.2 最优化

无约束最优化问题

$$\min_{x \in R^n} f(x)$$

其中 x^* 为目标函数的极小点。

设 $f(x)$ 具有二阶连续偏导数，若第 k 次迭代值为 $x^{(k)}$ ，则可将 $f(x)$ 在 $x^{(k)}$ 附近进行二阶泰勒展开

$$f(x) = f\left(x^{(k)}\right) + g_k^T\left(x - x^{(k)}\right) + \frac{1}{2}\left(x - x^{(k)}\right)^T H\left(x^{(k)}\right)\left(x - x^{(x)}\right)$$

其中， $g_k = g\left(x^{(k)}\right) = \nabla f\left(x^{(k)}\right)$ 是 $f(x)$ 的梯度向量在点 $x^{(k)}$ 的值， $H\left(x^{(k)}\right)$ 是 $f(x)$ 的海赛矩阵

$$H(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{n \times n}$$

在点 $x^{(k)}$ 的值。

函数 $f(x)$ 有极值的必要条件是在极值点处一阶导数为0，即梯度向量为0。特别的当 $H\left(x^{(k)}\right)$ 是正定矩阵时，函数 $f(x)$ 的极值为极小值。

我们为了得到一阶导数为0的点，可以用到2.1里的求解方程方法。根据二阶泰勒展开，对 $\nabla f(x)$ 在 $x^{(k)}$ 进行展开得（也可以对上述泰勒公式再进行求导）

$$\nabla f(x) = g_k + H_k\left(x - x^{(k)}\right)$$

其中，$H_k = H\left(x^{(k)}\right)$，则

$$g_k + H_k\left(x^{(k+1)} - x^{(k)}\right) = 0$$
$$x^{(k+1)} = x^{(k)} - H_k^{-1} g_k$$

我们令

$$H_k p_k = -g_k$$

则得到迭代公式

$$x^{(k+1)} = x^{(k)} + p_k$$

最终可在 $\nabla f(x^*) = 0$ 收敛。

> **牛顿法**：
> 输入：目标函数 $f(x)$，梯度 $g(x) = \nabla f(x)$，海赛矩阵 $H(x)$，精度要求 ε
> 输出：$f(x)$ 的极小点 x^*
> 1. 取初始点 $x^{(0)}$，置 $k = 0$
> 2. 计算 $g_k = g\left(x^{(k)}\right)$
> 3. 若 $\|g_k\| < \varepsilon$ 则停止计算，得近似解 $x^* = x^{(k)}$
> 4. 计算 $H_k = H\left(x^{(k)}\right)$，并求 p_k
>
> $$H_k p_k = -g_k$$
>
> 5. 置 $x^{(k+1)} = x^{(k)} + p_k$
> 6. 置 $k = k + 1$，转2.

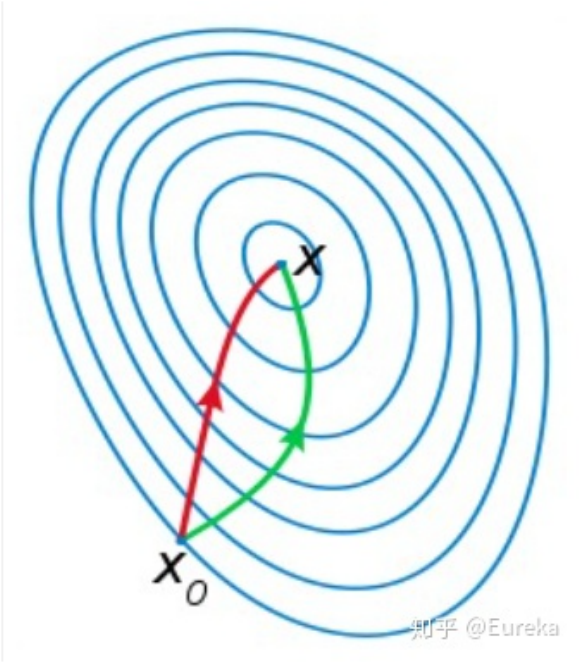
2.3： 牛顿法与梯度下降法

梯度下降法和牛顿法相比，两者都是迭代求解，不过梯度下降法是梯度求解，而牛顿法是用二阶的海森矩阵的逆矩阵求解。相对而言，使用牛顿法收敛更快（迭代更少次数）。但是每次迭代的时间比梯度下降法长。

梯度下降法： $x^{(k+1)} = x^{(k)} - \lambda \nabla f(x^k)$

牛顿法： $x^{(k+1)} = x^{(k)} - \lambda (H^{(k)})^{-1} \nabla f(x^k)$

如下图是一个最小化一个目标方程的例子，红色曲线是利用牛顿法迭代求解，绿色曲线是利用梯度下降法求解。



至于为什么牛顿法收敛更快，通俗来说梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。更多的可见：最优化问题中，牛顿法为什么比梯度下降法求解需要的迭代次数更少？

1.3 实例

下面是一个可以使用牛顿方法解决的优化问题示例:

$$\min f(x) = x^2 + \sin(x)$$

我们可以用牛顿法来解决这个问题:

1. 定义优化问题: 目标函数为 $f(x) = x^2 + \sin(x)$ 。
2. 求 $f(x)$ 一阶导数和二阶导数: $f(x)$ 的一阶导数是 $f'(x) = 2x + \cos(x)$ ，二阶导数是 $f\,''\,(x) = 2 - \sin(x)$ 。
3. 选择一个初始猜想:设 $x_0 = 0$ 是我们的初始值。
4. 应用牛顿法:我们可以使用牛顿方法的更新公式，迭代优化初始值，直到收敛。更新公式为: $x_{n+1} = x_n - [f''(x_n)]^{-1} f'(x_n)$ 代入函数和初始猜测，得到:

$$x_{n+1} = x_n - \frac{(2x_n + \cos(x_n))}{(2 - \sin(x_n))}$$

我们现在可以迭代应用这个公式直到收敛。让我们进行两次迭代：

$$x_1 = 0 - cos(0)/2 = -0.5$$
$$x_2 = -0.5 - [-2 * 0.5 + \cos(-0.5)]/[2 - \sin(-0.5)] = -0.4506$$

5. 检查收敛性: 我们可以通过比较连续近似值与公差水平之间的差异来检查收敛性。让我们将公差级别设置为0.000001。最后两个近似值之间的差值为 $|x_2 - x_1| = 0.0494$ ，大于公差水平。继续迭代优化然后进行第6步和第7步。