机器学习

第 2 章迭代优化方法

欧阳毅

浙江工商大学
管理工程与电子商务学院

2023

目录

梯度下降法

- 导数的概念:

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

- 由公式可见，对点 x0 的导数反映了函数在点 x0 处的瞬时变化速率，或者叫在点 x0 处的斜度。推广到多维函数中，就有了梯度的概念，梯度是一个向量组合，反映了多维图形中变化速率最快的方向。

$$\bigtriangledown f(x,y) = \{\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\}$$

梯度下降法

- 导数的概念:

$$\mathrm{f}'(\mathrm{x}_0)=\lim_{\Delta\mathrm{x}\to0}\frac{\Delta\mathrm{f}}{\Delta\mathrm{x}}=\lim_{\Delta\mathrm{x}\to0}\frac{\mathrm{f}(\mathrm{x}_0+\Delta\mathrm{x})-\mathrm{f}(\mathrm{x}_0)}{\Delta\mathrm{x}}$$

- 由公式可见，对点 x0 的导数反映了函数在点 x0 处的瞬时变化速率，或者叫在点 x0 处的斜度。推广到多维函数中，就有了梯度的概念，梯度是一个向量组合，反映了多维图形中变化速率最快的方向。

$$\bigtriangledown\mathrm{f}(\mathrm{x},\mathrm{y})=\{\frac{\partial\mathrm{f}}{\partial\mathrm{x}},\frac{\partial\mathrm{f}}{\partial\mathrm{y}}\}$$

梯度下降法

举一个非常简单的例子，如求函数 $f(x) = x^2$ 的最小值。利用梯度下降的方法解题步骤如下：

- 1、求梯度，$\bigtriangledown f = 2x$
- 2、向梯度相反的方向移动，如下

  $$x \leftarrow x - \eta \cdot \bigtriangledown f$$

  ，其中 η 为步长。如果步长足够小，则可以保证每一次迭代都在减小，但可能导致收敛太慢，如果步长太大，则不能保证每一次迭代都减少，也不能保证收敛。
- 3、循环迭代步骤 2，直到 x 的值变化到使得 f(x) 在两次迭代之间的差值足够小，比如 0.0001，也就是说，直到两次迭代计算出来的 f(x) 基本没有变化，则说明此时 f(x) 已经达到局部最小值了。
- 4、此时，输出 x ，这个 x 就是使得函数 f(x) 最小时的 x 的取值

Python 代码解析

```
def gradientDescent(X, theta, y, iterations, alpha):
    Cost = []
    Cost.append(computeCost(X, theta, y))
    for i in range(iterations):
        grad0 = np.mean(h(X, theta) - y)
        grad1 = np.mean((h(X, theta) - y) * (X[:,1].
                                    reshape([len(X), 1])
                                    ))
        theta[0] = theta[0] - alpha * grad0
        theta[1] = theta[1] - alpha * grad1
        Cost.append(computeCost(X, theta, y))
    return theta, Cost

theta = np.zeros((2,1))
iterations = 100
alpha = 0.01
```

为什么是负梯度方向?

- 一阶泰勒展开式的表达式

$$f(x) \approx f(x_0) + (x - x_0) \cdot \bigtriangledown f(x_0)$$

- 其中，$x - x_0$ 是微小矢量，它的大小就是我们之前讲的步进长度 η，$x - x_0$ 可表示为

$$x - x_0 = \eta$$

- $f(x)$ 的表达式为:

$$f(x) \approx f(x_0) + \eta \bigtriangledown f(x_0)$$

- 我们希望 $f(x) < f(x_0)$, 则有:

$$f(x) - f(x_0) \approx \eta \bigtriangledown f(x_0) < 0$$

- 也就是说，梯度 $\bigtriangledown f(x_0)$ 与 $x - x_0$ 异号，变化方向相反。

为什么是负梯度方向?

- 一阶泰勒展开式的表达式

$$f(x) \approx f(x_0) + (x - x_0) \cdot \bigtriangledown f(x_0)$$

- 其中，$x - x_0$ 是微小矢量，它的大小就是我们之前讲的步进长度 η，$x - x_0$ 可表示为

$$x - x_0 = \eta$$

- $f(x)$ 的表达式为:

$$f(x) \approx f(x_0) + \eta \bigtriangledown f(x_0)$$

- 我们希望 $f(x) < f(x_0)$, 则有:

$$f(x) - f(x_0) \approx \eta \bigtriangledown f(x_0) < 0$$

- 也就是说，梯度 $\bigtriangledown f(x_0)$ 与 $x - x_0$ 异号，变化方向相反。

为什么是负梯度方向?

- 一阶泰勒展开式的表达式

$$f(x) \approx f(x_0) + (x - x_0) \cdot \bigtriangledown f(x_0)$$

- 其中，$x - x_0$ 是微小矢量，它的大小就是我们之前讲的步进长度 η，$x - x_0$ 可表示为

$$x - x_0 = \eta$$

- $f(x)$ 的表达式为:

$$f(x) \approx f(x_0) + \eta \bigtriangledown f(x_0)$$

- 我们希望 $f(x) < f(x_0)$, 则有:

$$f(x) - f(x_0) \approx \eta \bigtriangledown f(x_0) < 0$$

- 也就是说，梯度 $\bigtriangledown f(x_0)$ 与 $x - x_0$ 异号，变化方向相反。

为什么是负梯度方向?

- 一阶泰勒展开式的表达式

$$\mathrm{f(x)} \approx \mathrm{f(x_0)} + (\mathrm{x} - \mathrm{x_0}) \cdot \bigtriangledown \mathrm{f(x_0)}$$

- 其中，$\mathrm{x} - \mathrm{x_0}$ 是微小矢量，它的大小就是我们之前讲的步进长度 η，$\mathrm{x} - \mathrm{x_0}$ 可表示为

$$\mathrm{x} - \mathrm{x_0} = \eta$$

- $\mathrm{f(x)}$ 的表达式为:

$$\mathrm{f(x)} \approx \mathrm{f(x_0)} + \eta \bigtriangledown \mathrm{f(x_0)}$$

- 我们希望 $\mathrm{f(x)} < \mathrm{f(x_0)}$, 则有:

$$\mathrm{f(x)} - \mathrm{f(x_0)} \approx \eta \bigtriangledown \mathrm{f(x_0)} < 0$$

- 也就是说，梯度 $\bigtriangledown \mathrm{f(x_0)}$ 与 $\mathrm{x} - \mathrm{x_0}$ 异号，变化方向相反。

梯度下降法

- 梯度下降法还可以分成：
  批量梯度下降法（Batch Gradient Descent）
  在更新参数时使用所有的样本来进行更新。

$$\mathrm{x_{t+1}} = \mathrm{x_t} - \eta \bigtriangledown_{\mathrm{x_t}} \mathrm{f}(\mathrm{x_t}) \tag{1}$$

- 随机梯度下降法（Stochastic Gradient Descent）
  仅选取一个样本来求梯度

$$\mathrm{x_{t+1}} = \mathrm{x_t} - \eta \bigtriangledown_{\mathrm{x_t}} \mathrm{f}(\mathrm{x}^{(\mathrm{i})}) \tag{2}$$

- 小批量梯度下降法（Mini-batch Gradient Descent））
  用一部分样本进行梯度计算

$$\mathrm{x_{t+1}} = \mathrm{x_t} - \eta \bigtriangledown_{\mathrm{x_t}} \mathrm{f}(\mathrm{x}^{(\mathrm{i:n})}) \tag{3}$$

作业

例

梯度下降法求解 f(u,v) 的最小值

$$f(u,v)=\begin{bmatrix}u & v\end{bmatrix}\begin{bmatrix}2 & 2\\ 2 & 5\end{bmatrix}\begin{bmatrix}u\\ v\end{bmatrix}$$

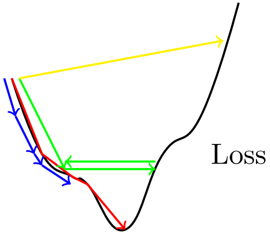
其中 $\begin{bmatrix}u_0\\ v_0\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}$ 给出计算过程和迭代次数

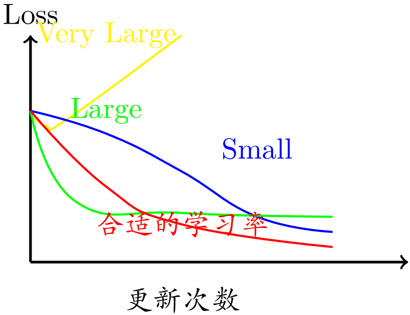
目录

如何设置学习率

$$x_{t+1} = x_t - \eta \bigtriangledown_{x_t} f(x_t) \tag{4}$$

- 如何设置学习率 η?





如何设置学习率

$$\mathrm{x_{t+1}} = \mathrm{x_t} - \eta \bigtriangledown_{\mathrm{x_t}} \mathrm{f}(\mathrm{x_t}) \tag{5}$$

- 初始迭代时，使用较大的学习率加速下降
- 迭代几次后，减小学习率防止振荡和越过

SGD+Momentum 算法

- 保持一个不随时间变化的速度，并将梯度估计添加到这个速度上，在这个速度方向上前进，而不是随梯度变化方向，给一个摩擦系数作为这个速度的衰减项。

SGD

$$x_{t+1} = x_t - \eta \bigtriangledown_{x_t} f(x_t)$$

动量法 Momentum：

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \eta v_{t+1}$$

- 更新是先估计当前梯度向量，取梯度和速度向量的和方向作为参数更新的方向

目录

AdaGrad 算法

$$x_{t+1} = x_t - \eta \bigtriangledown_{x_t} f(x_t) \tag{6}$$

- 如何设置学习率 η?
- 初始迭代时，使用较大的学习率加速下降
- 迭代几次后，减小学习率防止振荡和越过

AdaGrad 算法

$$x_{t+1} = x_t - \frac{\eta^t}{\sigma^t} g^t$$

$$\eta^t = \frac{\eta}{\sqrt{t+1}}, g^t = \bigtriangledown_{x_t} f(x_t)$$

其中 σ^t 为是每个参数的所有偏微分的均方根

AdaGrad 算法

$$\mathrm{x_{t+1}} \leftarrow \mathrm{x_t} - \frac{\eta}{\sqrt{\sum_{\mathrm{i}=0}^{\mathrm{t}}(\mathrm{g^i})^2}}\mathrm{g^t}$$

$$\eta^{\mathrm{t}} = \frac{\eta}{\sqrt{\mathrm{t}+1}}, \mathrm{g^t} = \triangledown_{\mathrm{x_t}}\mathrm{f}(\mathrm{x_t})$$

Adagrad：如果一个参数的梯度一直都较大，那么其对应的学习率就变小一点，防止震荡，

- 而一个参数的梯度一直都较小，那么这个参数的学习率就变大一点，使得其能够更快地更新

AdaGrad 算法

```
Require: : 全局学习率 $\eta$, 初始化参数 $\theta$，小常数 $\epsilon$
Ensure: : $\theta$
初始化梯度累积变量 r=0
while 没有达到停止条件 do
    从训练集中采集包含 m 个样本 $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ 的小批量，对应目标为 $y^{(i)}$
    计算梯度：$G \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i),\theta}), y^{(i)})$
    累积平方梯度：$r \leftarrow r + G \odot G$(逐元素求平方和，求内积)
    计算增量：$\triangle\theta \leftarrow -\frac{\eta}{\sqrt{r+\epsilon}} \odot G$
    更新：$\theta \leftarrow \theta + \triangle\theta$
end while
```

AdaGrad 算法

```
def sgd_adagrad(parameters, sqrs, lr):
 eps = 1e-10
 for param, sqr in zip(parameters, sqrs):
    sqr[:] = sqr + param.grad.data ** 2
    div = lr / torch.sqrt(sqr + eps) * param.grad.data
    param.data = param.data - div
```

目录

RMSProp 算法

- RMSProp 主要是为了解决 AdaGrad 方法中学习率过度衰减的问题
- Root Mean Square :RMS

```
Require: : 全局学习率 $\eta$, 初始化参数 $\theta$, 小常数 $\epsilon$, 衰减率 $\rho$
Ensure: : $\theta$
初始化梯度累积变量 r=0
while 没有达到停止条件 do
    从训练集中采集包含 m 个样本 $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ 的小批量，对应目标为 $y^{(i)}$
    计算梯度：$G \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i),\theta}), y^{(i)})$
    累积平方梯度：$r \leftarrow \rho r + (1-\rho) G \odot G$(逐元素求平方和，求内积)
    计算增量：$\triangle\theta \leftarrow -\frac{\eta}{\sqrt{r+\epsilon}} \odot G$
    更新：$\theta \leftarrow \theta + \triangle\theta$
end while
```

目录

1. 迭代优化方法
    - 梯度下降法
    - SGD+Momentum
    - AdaGrad 算法
    - RMSProp 算法
    - Adam 算法
    - 最速下降法
    - 牛顿法

Adam 算法

- 除了加入历史梯度平方的指数衰减平均（r）外，还保留了历史梯度的指数衰减平均（s），相当于动量。

```
Require: : 全局学习率 $\eta$, 初始化参数 $\theta$, 小常数 $\epsilon$, 衰减率 $\rho_1,\rho_2$
Ensure: : $\theta$
初始化梯度累积变量 r=0,s=0
while 没有达到停止条件 do
    从训练集中采集包含 m 个样本 $\{x^{(1)},x^{(2)},...,x^{(m)}\}$ 的小批量，对应目标为 $y^{(i)}$
    计算梯度: $G \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i),\theta}),y^{(i)})$
    $r \leftarrow \rho_2 r + (1-\rho_2) G \odot G$
    修正 $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
    修正 $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
    计算增量: $\triangle\theta \leftarrow -\frac{\eta}{\sqrt{\hat{r}}+\epsilon} \odot \hat{s}$
    更新: $\theta \leftarrow \theta + \triangle\theta$
end while
```

Adam 算法

```
def adam(params, states, hyperparams):
    p1, p2, eps = 0.9, 0.999, 1e-6
    for p, (v, s) in zip(params, states):
        v[:] = p1 * v + (1 - p1) * p.grad.data
        s[:] = p2 * s + (1 - beta2) * p.grad.data**2
        v_bias = v / (1 - p1 ** hyperparams['t'])
        s_bias = s / (1 - p2 ** hyperparams['t'])
        p.data -= hyperparams['lr'] * v_bias / (torch.
                                    sqrt(s_bias) + eps)
```

目录

最速下降法

- 一种迭代求解 $\mathrm{Ax} = \mathrm{b}$ 线性系统方法其中 x 是未知向量，b 和 A 是已知

定义

给定一个大小为 $\mathrm{n} \times \mathrm{n}$ 的实对称矩阵 A ，若对于任意长度为 n 的非零向量 x，有 $\mathrm{x}^{\mathrm{T}}\mathrm{Ax} > 0$ 恒成立，则矩阵 A 是一个正定矩阵

- 二次型

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

- 若 A 是对称且正定的，则最小化 f(x) 等于求解 $\mathrm{Ax} = \mathrm{b}$

若 A 是对称且正定的, 则是严格的凸二次规划问题，具有极小值

最速下降法 I

定义

给定一个函数 $f : R^n \to R$ 在 x_0 可导，最速方向为 $-\bigtriangledown f(x^{(0)})$.
定义函数

$$\phi(\alpha) = f(x^{(0)} + \alpha R)$$

其中 R 为单位向量 $||R|| = 1$

- 向 R 方向走 α 距离看看，利用求导链式规程:

$$\begin{aligned}
\phi'(\alpha) &= \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial \alpha} + ... + \frac{\partial f}{\partial x_n}\frac{\partial x_n}{\partial \alpha} && (7)\\
&= \frac{\partial f}{\partial x_1}R_1 + ... + \frac{\partial f}{\partial x_n}R_n && (8)\\
&= < \bigtriangledown f(x^{(0)} + \alpha R), R > && (9)\\
&= 0 && (10)
\end{aligned}$$

最速下降法 II

- $\mathrm{R}^{(k)} = -\bigtriangledown f(x^{(k)})$

关键如何确定步长 α_k?
一般情况:

$$\alpha_k = \arg\min_{\alpha_k} f(x_i - \alpha_k \bigtriangledown f(x_i)) = \arg\min_{\alpha_k} f(x_i + \alpha_k \mathrm{R}^k)$$

- 通过最小化 $\phi_k(\alpha)$(走 α 距离就到极值点，导数为 0)，即：$\phi_k(\alpha)' = 0$ 找出最小化时的 α，来更新 $x^{(k+1)}$
- 并迭代此过程, 求函数 $\phi_k(t)$ 的最小值

最速下降法 III

- 每一步，我们选择使 f 下降最快的方向。即

$$\mathrm{R}^{(\mathrm{i})} = -\bigtriangledown \mathrm{f}(\mathrm{x}^{(\mathrm{i})})$$

$$\mathrm{x}^{(\mathrm{k}+1)} = \mathrm{x}^{(\mathrm{k})} + \alpha_{\mathrm{k}} \mathrm{R}^{(\mathrm{k})}$$

$$\mathrm{x}^{(\mathrm{k}+1)} = \mathrm{x}^{(\mathrm{k})} - \alpha_{\mathrm{k}} \bigtriangledown \mathrm{f}(\mathrm{x}^{(\mathrm{k})})$$

最速下降法-第 1 种方法通过 ϕ 求 α I

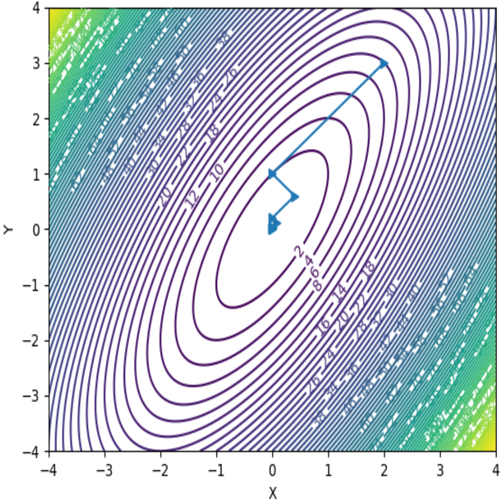
例

请运用 Steepest descent 算法求解函数,$x_0 = [2, 3]$

$$f(x, y) = 4x^2 - 4xy + 2y^2$$

.

最速下降法-凸函数优化问题



最速下降法-凸函数优化问题

```python
from sympy import *
x1, x2 = symbols('x1, x2')
fun = 4*x1**2 -4*x1*x2 +2*x2**2
g1,g2 = diff(fun, x1), diff(fun, x2)
def steepest_descent(f, x0, iterations):
    x1_val,x2_val = x0[0],x0[1]
    grad_x1 = g1.subs({x1:x1_val, x2: x2_val}).evalf()
    grad_x2 = g2.subs({x1:x1_val, x2: x2_val}).evalf()
    for i in range(iterations):
      alpha = symbols('alpha')
      x1_exp = x1_val + alpha * grad_x1
      x2_exp = x2_val + alpha * grad_x2
      f_new = f.subs({x1:x1_exp,x2:x2_exp})
      g_new=diff(f_new,alpha)
      alpha = solve(g_new,alpha)[0]

      grad_x1 = g1.subs({x1:x1_val, x2:x2_val}).evalf()
      grad_x2 = g2.subs({x1:x1_val, x2:x2_val}).evalf()
      x1_value=x1_value+alpha*grad_x1
      x2_value=x2_value+alpha*grad_x2
```

最速下降法的性质 I

- 最速下降法相邻两次搜索方向正交:$(\mathrm{R}^{(k+1)}).\mathrm{R}^{(k)} = 0$

证明.

根据算法条件（沿着 $\mathrm{R}^{k)}$ 方向走 $\alpha^{(k)}$ 步到达处有 $\phi'(\alpha^{(k)}) = 0$),
所以 $\frac{\mathrm{df}(\mathrm{x}^{(k)}+\alpha^{(k)}\mathrm{R}^{(k)})}{\mathrm{d}\alpha^{(k)}} = 0$, 并且 $\mathrm{R}^{(k+1)} = -\bigtriangledown \mathrm{f}(\mathrm{x}^{(k+1)})$

$$\begin{align}
\frac{\mathrm{df}(\mathrm{x}^{(k)}+\alpha^{(k)}\mathrm{R}^{(k)})}{\mathrm{d}\alpha^{(k)}} &= \frac{\mathrm{df}(\mathrm{x}^{(k)}+\alpha^{(k)}\mathrm{R}^{(k)})}{\mathrm{d}(\mathrm{x}^{(k)}+\mathrm{a}^{k}\mathrm{R}^{(k)})}\frac{\mathrm{d}(\mathrm{x}^{(k)}+\mathrm{a}^{(k)}\mathrm{R}^{(k)})}{\mathrm{da}^{k}} \tag{11}\\
&= \bigtriangledown \mathrm{f}(\mathrm{x}^{(k)}+\mathrm{a}^{k}\mathrm{R}^{(k)})\mathrm{R}^{(k)} \tag{12}\\
&= \bigtriangledown \mathrm{f}(\mathrm{x}^{(k+1)})\mathrm{R}^{(k)} \tag{13}\\
&= (-\mathrm{R}^{(k+1)}).\mathrm{R}^{(k)} \tag{14}\\
&= 0 \tag{15}
\end{align}$$

□

最速下降法-第 2 种实现方法 I

- 考虑二次函数 f, 其表达式为

$$f(x) = \frac{1}{2}x^T Ax + bx + r$$

由于 $\nabla^2 f(x) = A$, 函数 f 是严格凸的，当且仅当 A 对称且正定

- 若 A 是正定的，则最小化 f(x) 等于求解 $Ax = b$

$$\bigtriangledown f(x^{(i)}) = Ax^{(i)} - b = -R^{(i)}$$

$$R^{(i)} = b - Ax^{(i)}$$

所以 $R^{(i)}$ 也可以看着残差: 表示离正确 b 值有多远

最速下降法-α 如何确定?

- 根据上两式和求导链式规则:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\mathrm{f}(\mathrm{x}^{(\mathrm{i}+1)}) = \frac{\mathrm{df}(\mathrm{x}^{(\mathrm{i}+1)})}{\mathrm{dx}^{(\mathrm{i}+1)}}\frac{\mathrm{dx}^{(\mathrm{i}+1)}}{\mathrm{d}\alpha} = \bigtriangledown \mathrm{f}(\mathrm{x}^{(\mathrm{i}+1)})\mathrm{R}^{(\mathrm{i})} = -(\mathrm{R}^{(\mathrm{i}+1)})^{\mathrm{T}}\mathrm{R}^{(}$$

$$\begin{aligned}
(\mathrm{R}^{(1)})^{\mathrm{T}}\mathrm{R}^{(0)} &= 0 \qquad (16)\\
(\mathrm{b}-\mathrm{Ax}^{(1)})^{\mathrm{T}}\mathrm{R}^{(0)} &= 0 \qquad (17)\\
(\mathrm{b}-\mathrm{A}(\mathrm{x}^{(0)}+\alpha\mathrm{R}^{(0)}))^{\mathrm{T}}\mathrm{R}^{(0)} &= 0 \qquad (18)\\
\alpha = \frac{(\mathrm{R}^{(0)})^{\mathrm{T}}\mathrm{R}^{(0)}}{\mathrm{R}^{(0)})^{\mathrm{T}}\mathrm{A}\mathrm{R}^{(0)}} & \qquad (19)
\end{aligned}$$

最速下降法

Step 1: 计算

$$R^{(i)} = b - Ax^{(i)}$$

Step 2:

$$\alpha^{i} = \frac{(R^{(i)})^{T} R^{(i)}}{(R^{(i)})^{T} A R^{(i)}}$$

Step 3:

$$x^{(i+1)} = x^{(i)} + \alpha^{(i)} R^{(i)}$$

优化 由于 Step1，Step2 中有两次矩阵与向量乘，计算开销较大，若 Step3 两边同乘-A

$$b - Ax^{(i+1)} = b - Ax^{(i)} - \alpha^{(i)} A R^{(i)}$$

$$R^{(i+1)} = R^{(i)} - \alpha^{(i)} A R^{(i)}$$

$AR^{(i)}$ 只需要做一次做一次矩阵向量乘

最速下降法-第 2 种方法通过 R 求 α I

例

请运用 Steepest descent 算法求解函数,$x_0 = [2, 3]$

$$f(x, y) = 4x^2 - 4xy + 2y^2$$

最速下降法-凸函数优化问题 I

```
import numpy as np
count= 0
A = np.array([[8, -4], [-4, 4]])
b = np.array([[0],[0]])
x0 = np.array([[2],[3]])
Xi=np.array([[-1000],[-1000]])
Xnew=x0
while( (np.linalg.norm( Xnew- Xi))**2 >= 0.001):
    Xi=Xnew
    R= b - np.dot(A,Xi)
    Rt=R.T
    alpha=(np.dot(Rt,R))/( np.dot(np.dot(Rt, A),R))
    Xnew = Xi + (alpha)*R
    count+=1
```

目录

判断一个矩阵 A 是否为正定矩阵

- 求出 A 的所有特征值。若 A 的特征值均为正数，则 A 是正定的；若 A 的特征值均为负数，则 A 为负定的
- 计算 A 的各阶顺序主子式。若 A 的各阶顺序主子式均大于零，则 A 是正定的；若 A 的各阶主子式中，奇数阶主子式为负，偶数阶为正，则 A 为负定的。

$\mathrm{D} = \begin{bmatrix} 3 & 2 & 0 \\ 2 & 4 & -2 \\ 0 & -2 & 5 \end{bmatrix}$,D 的各阶顺序主子式: $\mathrm{D}_1 = 3 > 0$,

$\mathrm{D}_2 = \begin{vmatrix} 3 & 2 \\ 2 & 4 \end{vmatrix} = 8 > 0$, $\mathrm{D}_3 = \begin{vmatrix} 3 & 2 & 0 \\ 2 & 4 & -2 \\ 0 & -2 & 5 \end{vmatrix} = 28 > 0$

判断一个矩阵 A 是否可逆

定义

矩阵 A 是 n 阶可逆矩阵的等价条件:

- 1. A 的行列式不等于 0
- 2. A 的秩等于 n，即 A 为满秩矩阵
- 3. A 的行（列）向量组线性无关
- 4. 齐次方程组 Ax=0 只有零解
- 5. 正定矩阵是可逆的

牛顿法

- 设 $f(x)$ 是二次可微实函数，又设 $x^{(k)}$ 是 $f(x)$ 一个极小点的估计，我们把 $f(x)$ 在 $x^{(k)}$ 处展开成 Taylor 级数，并取二阶近似

$$f(x) \sim f(x^{(k)}) + \triangledown f(x^{(k)})^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T \triangledown^2 f(x^{(k)})(x -$$

- 上式中最后一项的中间部分表示 $f(x)$ 在 $x^{(k)}$ 处的 Hessian 矩阵。对上式求导并令其等于 0，可以的到下式：

$$\triangledown f(x^{(k)}) + \triangledown^2 f(x^{(k)})(x - x^{(k)}) = 0$$

$$-\triangledown f(x^{(k)}) = \triangledown^2 f(x^{(k)})(x - x^{(k)})$$

- 若 Hessian 矩阵可逆，x 看着 $x^{(k+1)}$

牛顿法：

$$x^{(k+1)} = x^k + [-\triangledown^2 f(x^{(k)})^{-1} \triangledown f(x^{(k)})]$$

Newton 方法求解凸函数优化问题

```
$\mathrm{initalize}\ x_0, k = 1;$
$\mathrm{while}\ (||\bigtriangledown f(x^{(k)})|| \geq \epsilon\ \mathrm{and}\ (k < \mathrm{maxiter})\ \mathrm{do}$
    $x^{(k+1)} = x^{(k)} - \bigtriangledown f(x^{(k)}) H_f(x^{(k)})^{-1}$
    $k = k + 1$
$\mathrm{end\ while}$
$\mathrm{return}\ x^{(k)}$
```

牛顿法 I

例

请运用 Newton's method 算法求解函数最小值

$$f(x,y) = x^4 + 2x^2y^2 + y^4$$

牛顿法

```
import numdifftools as nd
def func(X):
    x,y=X[0],X[1]
    return x**4+2*x**2*y**2+y**4
def NewtonMethod(x):
    err = 100.0
    while err>0.0001:
        p_x = x
        prev = func(x)
        hessian = nd.Hessian(func)
        h2 = hessian(x)
        h_inv= np.linalg.inv(h2)
        gradient = nd.Gradient(func)(x)
        mul = h_inv@gradient
        x = x - mul
        err = abs(prev - func(x))
        if (p_x[0] == x[0] and p_x[1] == x[1]):
            break
    return x,func(x)
```

拟牛顿方法 (Quasi-Newton) I

- 拟牛顿法是构造与 Hessian 矩阵相似的正定矩阵.
- 把 $f(x)$ 在 $x^{(k+1)}$ 处展开成 Taylor 级数，并取二阶近似

$$f(x) \approx f(x^{(k+1)}) + \bigtriangledown f(x^{(k+1)})(x - x^{(k+1)}) + \frac{1}{2}(x - x^{(k+1)})^T \bigtriangledown^2 f(x^{($$

- 令 $g_k = g(x^{(k)}) = \bigtriangledown f(x^{(k)})$,
  对 $f(x)$ 两边同时作用梯度算子 $\bigtriangledown$:

$$\bigtriangledown f(x) \approx \bigtriangledown f(x^{(k+1)}) + H_{k+1}(x - x^{(k+1)})$$

  上式若取 $x = x^{(k)}$ 可得:

$$t_k = g_{k+1} - g_k \approx H_{k+1}(x^{(k+1)} - x^{(k)})$$

- 发现梯度和 x 的增量之前就差一个 Hession 矩阵

DFP 算法 I

- DFP 算法是以 Davidon、Fletcher、Powell 的首字母命名的
- 用 B_{k+1} 来近似 H_{k+1}^{-1}

拟牛顿条件

令 $s_k = x_{k+1} - x_k$, $t_k = H_{k+1} \cdot s_k$, 若
$t_k = g_{k+1} - g_k \approx H_{k+1}(x^{(k+1)} - x^{(k)})$

$$t_k \approx H_{k+1} \cdot s_k$$

$$s_k \approx H_{k+1}^{-1} \cdot t_k$$

牛顿法:

$$x^{(k+1)} = x^k + [- \bigtriangledown^2 f(x^{(k)})^{-1} \bigtriangledown f(x^{(k)})]$$