1

```
初始值
 x=[[2]
 [2]]

 g=[[  4]
 [100]]

 f=[[104]]
----------------------------------------------
第1次迭代
 x=[[ 1.91987713]
 [-0.0030718 ]]

 g=[[ 3.83975426]
 [-0.15359017]]

 f=[[3.68616409]]
----------------------------------------------
第2次迭代
 x=[[0.07088777]
 [0.07088777]]

 g=[[0.14177554]
 [3.54438854]]

 f=[[0.13065198]]
----------------------------------------------
第3次迭代
 x=[[ 0.0680479 ]
 [-0.00010888]]

 g=[[ 0.13609581]
 [-0.00544383]]

 f=[[0.00463081]]
----------------------------------------------
第4次迭代
 x=[[0.00251254]
 [0.00251254]]

 g=[[0.00502508]
 [0.1256269 ]]

 f=[[0.00016413]]
----------------------------------------------
第5次迭代
 x=[[ 2.41188215e-03]
 [-3.85901144e-06]]

 g=[[ 0.00482376]
 [-0.00019295]]
 f=[[5.8175478e-06]]
----------------------------------------------
```

2

```
初始值
 x=[[0]
 [0]]

 g=[[-10.]
 [ -4.]]

 f=[[60]]
----------------------------------------------
第1次迭代
 x=[[8.]
 [6.]]

 g=[[-1.77635684e-15]
 [ 8.88178420e-16]]

 f=[[8.]]
----------------------------------------------
```

3

```
初始值
 x=[[0]
 [0]]

 g=[[ 9.]
 [-3.]]

 f=[[16]]
----------------------------------------------
第1次迭代
 x=[[-1.125]
 [ 0.75 ]]

 g=[[0.]
 [0.]]

 f=[[9.8125]]
----------------------------------------------
```

5

```
初始值
```

```
 x=[[1]
 [1]]

 g=[[3.]
 [1.]]

 f=[[2]]
------------------------------------------------
第1次迭代
 x=[[0.0625]
 [0.6875]]

 g=[[-0.4375]
 [ 1.3125]]

 f=[[0.4375]]
------------------------------------------------
第2次迭代
 x=[[9.02056208e-17]
 [0.00000000e+00]]

 g=[[ 3.60822483e-16]
 [-9.02056208e-17]]

 f=[[1.6274108e-32]]
------------------------------------------------
```

6

```
初始值
 x=[[8]
 [9]]

 g=[[24.]
 [ 6.]]

 H=[[1. 0.]
 [0. 1.]]

 f=[[45]]
------------------------------------------------
第1次迭代
 x=[[4.86153846]
 [8.21538462]]

 g=[[-1.10769231]
 [ 4.43076923]]

 H=[[ 0.12696797 -0.03148758]
 [-0.03148758  1.00380126]]

 f=[[4.98461538]]
------------------------------------------------
第2次迭代
```

```
 x=[[5.]
 [6.]]

 g=[[0.]
 [0.]]

 H=[[ 1.25000000e-01 -1.38777878e-17]
 [-6.93889390e-18  5.00000000e-01]]

 f=[[0.]]
----------------------------------------------
```

7

```
初始值
 x=[[0]
 [0]]

 g=[[-10.]
 [ -4.]]

 H=[[1. 0.]
 [0. 1.]]

 f=[[60]]
----------------------------------------------
第1次迭代
 x=[[7.25  ]
 [5.4375]]

 g=[[-0.9375]
 [-0.375 ]]

 f=[[8.45703125]]
----------------------------------------------
第2次迭代
 x=[[7.9296875 ]
 [5.94726562]]

 g=[[-0.08789062]
 [-0.03515625]]

 f=[[8.00401688]]
----------------------------------------------
第3次迭代
 x=[[7.9934082 ]
 [5.99505615]]

 g=[[-0.00823975]
 [-0.0032959 ]]

 f=[[8.0000353]]
----------------------------------------------
```

8

```
初始值
 x=[matrix([[0.259],
        [0.965]]), matrix([[0.965],
        [0.259]]), matrix([[0],
        [0]])]

 f=[[-1.826469]]
----------------------------------------------
第1次迭代
 x=[matrix([[1.2852],
        [1.2852]]), matrix([[0.259],
        [0.965]]), matrix([[0.965],
        [0.259]])]

 f=[[-5.46718288]]
----------------------------------------------
第2次迭代
 x=[matrix([[1.2852],
        [1.2852]]), matrix([[0.5792],
        [1.9912]]), matrix([[0.259],
        [0.965]])]

 f=[[-5.46718288]]
----------------------------------------------
第3次迭代
 x=[matrix([[1.6054],
        [2.3114]]), matrix([[1.2852],
        [1.2852]]), matrix([[0.5792],
        [1.9912]])]

 f=[[-6.65035092]]
----------------------------------------------
第4次迭代
 x=[matrix([[1.6054],
        [2.3114]]), matrix([[2.3114],
        [1.6054]]), matrix([[1.2852],
        [1.2852]])]

 f=[[-6.65035092]]
----------------------------------------------
第5次迭代
 x=[matrix([[2.295],
        [2.295]]), matrix([[1.6054],
        [2.3114]]), matrix([[2.3114],
        [1.6054]])]

 f=[[-6.738925]]
----------------------------------------------
```

源码

1

```
# 最速下降法

import numpy as np

epsilon = 0.01

f = lambda x: np.matrix([1,25]) * np.power(x, 2)

H = np.matrix([[2,0],
                [0,50]])

g = lambda x: np.multiply(np.matrix([2,50]).T, x)

x = np.matrix([2,2]).T

print("初始值\n x={}\n\n g={}\n\n f={}".format(x, g(x), f(x)))
print("----------------------------------------------")

i = 1
while np.linalg.norm(g(x)) > epsilon:
    grad = g(x)
    step = (grad.T * grad) / (grad.T * H * grad)
    x = x - grad * step
    print("第{}次迭代\n x={}\n\n g={}\n\n f={}".format(i, x, g(x), f(x)))
    print("----------------------------------------------")
    i += 1
```

2

```
# 牛顿法

import numpy as np

epsilon = 0.01

def f(x):
    x1 = x[0][0]
    x2 = x[1][0]
    return 60 - 10 * x1 - 4 * x2 + x1 ** 2 + x2 ** 2 - x1 * x2

def g(x):
    grad = np.zeros((2,1))
    x1 = x[0][0]
    x2 = x[1][0]
    grad[0, 0] = -10 + 2 * x1 - x2
    grad[1, 0] = -4 + 2 * x2 - x1
    return grad

H = np.matrix([[2,-1],
                [-1,2]])

x = np.matrix([0,0]).T

print("初始值\n x={}\n\n g={}\n\n f={}".format(x, g(x), f(x)))
```

```python
print("----------------------------------------------")

i = 1
while np.linalg.norm(g(x)) > epsilon:
    grad = g(x)
    x = x - H.I * grad
    print("第{}次迭代\n x={}\n\n g={}\n\n f={}".format(i, x, g(x), f(x)))
    print("----------------------------------------------")
    i += 1
```

3

```python
# 修正牛顿法

import numpy as np

epsilon = 0.01

def f(x):
    x1 = x[0][0]
    x2 = x[1][0]
    return 4 * (x1 + 1) ** 2 + 2 * (x2 - 1) ** 2 + x1 + x2 + 10

def g(x):
    grad = np.zeros((2,1))
    x1 = x[0][0]
    x2 = x[1][0]
    grad[0, 0] = 8 * x1 + 9
    grad[1, 0] = 4 * x2 - 3
    return grad

H = np.matrix([[8,0],
               [0,4]])

x = np.matrix([0,0]).T

print("初始值\n x={}\n\n g={}\n\n f={}".format(x, g(x), f(x)))
print("----------------------------------------------")

i = 1
while np.linalg.norm(g(x)) > epsilon:
    grad = g(x)
    G = H.I
    p = -G * grad
    step = -(grad.T * p) / (p.T * H * p)
    x = x + p * step
    print("第{}次迭代\n x={}\n\n g={}\n\n f={}".format(i, x, g(x), f(x)))
    print("----------------------------------------------")
    i += 1
```

5

```python
# 共轭梯度法

import numpy as np
```

```
epsilon = 0.01

def f(x):
    x1 = x[0][0]
    x2 = x[1][0]
    return 2 * x1 ** 2 + x2 ** 2 - x1 * x2

def g(x):
    grad = np.zeros((2,1))
    x1 = x[0][0]
    x2 = x[1][0]
    grad[0, 0] = 4 * x1 - x2
    grad[1, 0] = 2 * x2 - x1
    return grad

H = np.matrix([[4,-1],
               [-1,2]])

x = np.matrix([1,1]).T

print("初始值\n x={}\n\n g={}\n\n f={}".format(x, g(x), f(x)))
print("----------------------------------------------")

i = 1
grad = g(x)
p = -grad
while np.linalg.norm(g(x)) > epsilon:
    grad = g(x)
    step = -(grad.T * p) / (p.T * H * p)
    x = x + step * p
    beta = (np.linalg.norm(g(x)) / np.linalg.norm(grad)) ** 2
    p = -g(x) + p * beta
    print("第{}次迭代\n x={}\n\n g={}\n\n f={}".format(i, x, g(x), f(x)))
    print("----------------------------------------------")
    i += 1
```

6

```
# DFP

import numpy as np

epsilon = 0.01

def f(x):
    x1 = x[0][0]
    x2 = x[1][0]
    return 4 * (x1 - 5) ** 2 + (x2 - 6) ** 2

def g(x):
    grad = np.zeros((2,1))
    x1 = x[0][0]
    x2 = x[1][0]
    grad[0, 0] = 8 * x1 - 40
    grad[1, 0] = 2 * x2 - 12
    return grad
```

```
G = np.matrix([[8,0],
               [0,2]])

x = np.matrix([8,9]).T

i = 1
H = np.matrix(np.eye(2))

print("初始值\n x={}\n\n g={}\n\n H={}\n\n f={}".format(x, g(x), H, f(x)))
print("----------------------------------------------")


while np.linalg.norm(g(x)) > epsilon:
    grad = g(x)
    p = -H * grad
    alpha = -(grad.T * p) / (p.T * G * p)
    new_x = x + p * alpha
    new_grad = g(new_x)
    s = new_x - x
    y = new_grad - grad
    H = H + (s * s.T) / (s.T * y) - (H * y * y.T *H) / (y.T * H * y)
    x = new_x
    print("第{}次迭代\n x={}\n\n g={}\n\n H={}\n\n f={}".format(i, x, g(x), H, f(x)))
    print("----------------------------------------------")
    i += 1
```

7

```
# 坐标轮换法

import numpy as np

epsilon = 0.1

def f(x):
    x1 = x[0][0]
    x2 = x[1][0]
    return x1 ** 2 + x2 ** 2 - x1 * x2 - 10 * x1 - 4 * x2 + 60

def g(x):
    grad = np.zeros((2,1))
    x1 = x[0][0]
    x2 = x[1][0]
    grad[0, 0] = 2 * x1 - x2 - 10
    grad[1, 0] = 2 * x2 - x1 - 4
    return np.matrix(grad)

G = np.matrix([[2,-1],
               [-1,2]])

x = np.matrix([0,0]).T

k = 1
H = np.matrix(np.eye(2))

print("初始值\n x={}\n\n g={}\n\n H={}\n\n f={}".format(x, g(x), H, f(x)))
print("-----------------------------------------------")



while True:
    old_x = x
    for i in range(2):
        grad = g(x)
        p = H[i].T
        alpha = -(grad.T * grad) / (grad.T * G * p)
        x = x + p * alpha

    print("第{}次迭代\n x={}\n\n g={}\n\n f={}".format(k, x, g(x), f(x)))
    print("-----------------------------------------------")
    k += 1
    if np.linalg.norm(x - old_x) < epsilon:
        break
```

8

```
# 单纯形法

import numpy as np

def f(x):
    x1 = x[0][0]
    x2 = x[1][0]
    return x1 ** 2 + 2 * x2 ** 2 - 4 * x1 - 8 * x2 + 5

epsilon = 0.1

alpha = 1.1

beta = 0.5

x1 = np.matrix([0,0]).T

x2 = np.matrix([0.965,0.259]).T

x3 = np.matrix([0.259,0.965]).T

i = 1

x = [x1,x2,x3]

x.sort(key= lambda x: f(x))
print("初始值\n x={}\n\n f={}".format(x, f(x[0])))
print("-----------------------------------------------")

while True:
    x.sort(key= lambda x: f(x))
    old = f(x[0])
    middle_point = (x[0] + x[1]) / 2
    reflect_point = middle_point + (middle_point - x[2])
    if f(reflect_point) < f(x[0]):
        # 小于最优点，扩张
        extern_point = middle_point + alpha * (middle_point - x[2])
        if f(extern_point) < f(reflect_point):
            x[2] = extern_point
        else:
            x[2] = reflect_point
    elif f(reflect_point) > f(x[0]) and f(reflect_point) < f(x[1]):
        # 位于最优点和次优点之间，直接替代
        x[2] = reflect_point
    elif f(reflect_point) > f(x[1]) and f(reflect_point) < f(x[2]):
        # 位于次优点和最差点之间，收缩
        shrink_point = middle_point + beta * (reflect_point - middle_point)
        x[2] = shrink_point
    else:
        # 比最差点还要差，压缩
        compress_point = middle_point - beta * (middle_point - x[2])
        if f(compress_point) < f(x[2]):
            # 压缩点小于最差点
            x[2] = compress_point
        else:
            # 压缩点依旧大于最差点，压缩原三角形
            x[1], x[2] = (x[0] + x[1]) / 2, (x[0] + x[2]) / 2

    x.sort(key= lambda x: f(x))
    print("第{}次迭代\n x={}\n\n f={}".format(i, x, f(x[0])))
    print("-----------------------------------------------")
    i += 1
    if abs((f(x[2]) - f(x[0])) / f(x[0])) <= epsilon:
        break
```