

牛顿法解RosenBrock函数 最小值问题

quandy2020@126.com

1 Rosenbrock函数

- Rosenbrock function

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

其中全局最小值 $(x, y) = (a, a^2)$, 当 $a = 1, b = 100$

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

优化问题：

$$\operatorname{argmin} f(x)$$

- Gradient

$$\nabla f = \begin{bmatrix} -400xy + 400x^3 + 2x - 2 \\ 200y - 200x^2 \end{bmatrix}$$

- Hessian

$$H = \begin{bmatrix} -400y + 1200x^2 + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

2 Newton's method

梯度下降（或称最陡下降）是一种用于寻找函数最小值的一阶迭代优化算法。为了通过梯度下降找到函数的局部最小值，沿着当前点处函数梯度（或近似梯度）的负方向迈出一步。如果沿着梯度的正方向迈出一步，那么就是在逼近该函数的局部最大值；这个过程则被称为梯度上升。

这一方法基于这样的观察：如果一个多变量函数在某点的邻域内被定义并且可微，那么该函数在负梯度方向下降最快。由此可得，如果：

$$x_{n+1} = x_n - \alpha \nabla F(x)$$

- 梯度处处垂直于等高线。
- 在每次线搜索后，新的梯度始终与先前的步进方向正交（对于任何线搜索都成立）。
- 因此，迭代往往以非常低效的方式在山谷中蜿蜒前行。

- 牛顿法如下：

Algorithm 1 Newton's Method

Initialize at x^0 , and set $k \leftarrow 0$.
 At iteration k :
 1. $d^k := -H(x^k)^{-1} \nabla f(x^k)$. If $d^k = 0$, then stop.
 2. Choose step-size $\alpha^k = 1$.
 3. Set $x^{k+1} \leftarrow x^k + \alpha^k d^k$, $k \leftarrow k + 1$.

- 梯度下降法如下：

Algorithm 1 Gradient descent

Require: A starting point x_0 , a number of iterations T , a sequence of stepsizes $(\alpha_t)_{0 \leq t \leq T-1}$
for $t = 0, \dots, T - 1$ **do**
 Define $x_{t+1} = x_t - \alpha_t \nabla f(x_t)$.
end for
return x_T

- 步长搜索

Algorithm 2 Backtracking line search

Require: Parameters $c, \tau \in]0; 1[$, maximal stepsize value a_{max}
 Define $\alpha_t = a_{max}$.
while $f(x_t - \alpha_t \nabla f(x_t)) > f(x_t) - c \alpha_t \|\nabla f(x_t)\|^2$ **do**
 Set $\alpha_t = \tau \alpha_t$.
end while
return α_t

3 Python code

- Rosenbrock function

```
1 def Rosenbrock(x,y):
2     return (1 + x)**2 + 100*(y - x**2)**2
```

- Rosenbrock gradient

```
1 def Grad_Rosenbrock(x,y):
2     g1 = -400*x*y + 400*x**3 + 2*x - 2
3     g2 = 200*y - 200*x**2
4     return np.array([g1,g2])
```

- Rosenbrock Hessian

```
1 def Hessian_Rosenbrock(x,y):
2     h11 = -400*y + 1200*x**2 + 2
3     h12 = -400 * x
4     h21 = -400 * x
5     h22 = 200
6     return np.array([[h11,h12],[h21,h22]])
```

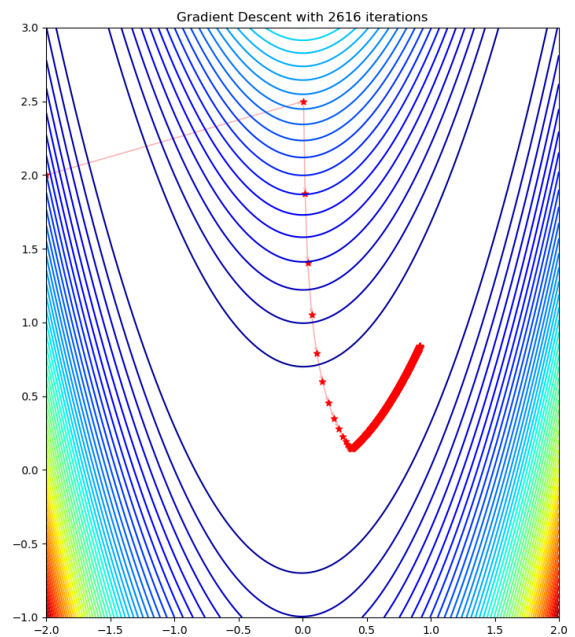
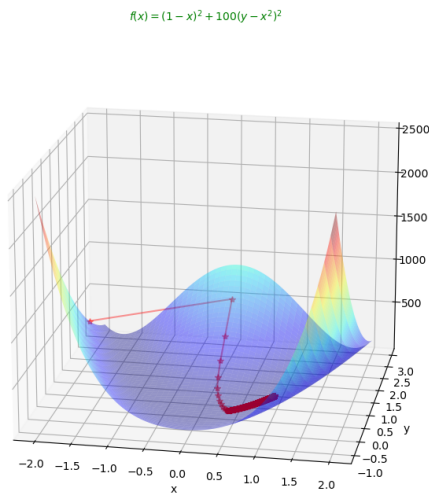
- Gradient Descent implementation

```

1  def GradientDescent(Grad,x,y, gamma = 0.00125, epsilon=0.0001, nMax =
    10000 ):
2      #Initialization
3      i = 0
4      iter_x, iter_y, iter_count = np.empty(0),np.empty(0), np.empty(0)
5      error = 10
6      X = np.array([x,y])
7
8      #Looping as long as error is greater than epsilon
9      while np.linalg.norm(error) > epsilon and i < nMax:
10         i +=1
11         iter_x = np.append(iter_x,x)
12         iter_y = np.append(iter_y,y)
13         iter_count = np.append(iter_count ,i)
14         #print(X)
15
16         X_prev = X
17         X = X - gamma * Grad(x,y)
18         error = X - X_prev
19         x,y = X[0], X[1]
20
21     print(X)
22     return X, iter_x,iter_y, iter_count

```

最优解：[0.91654302 0.83970004]



- Python完整代码

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  from mpl_toolkits import mplot3d
4
5  # https://xavierbourretsicotte.github.io/Intro_optimization.html
6
7  def Rosenbrock(x,y):
8      return (1 + x)**2 + 100*(y - x**2)**2

```

```

9
10 def GradRosenbrock(x,y):
11     g1 = -400*x*y + 400*x**3 + 2*x -2
12     g2 = 200*y -200*x**2
13     return np.array([g1,g2])
14
15 def HessianRosenbrock(x,y):
16     h11 = -400*y + 1200*x**2 + 2
17     h12 = -400 * x
18     h21 = -400 * x
19     h22 = 200
20     return np.array([[h11,h12],[h21,h22]])
21
22 def GradientDescent(Grad,x,y, gamma = 0.00125, epsilon=0.0001, nMax =
23     10000 ):
24     #Initialization
25     i = 0
26     iter_x, iter_y, iter_count = np.empty(0),np.empty(0), np.empty(0)
27     error = 10
28     X = np.array([x,y])
29     #Looping as long as error is greater than epsilon
30     while np.linalg.norm(error) > epsilon and i < nMax:
31         i +=1
32         iter_x = np.append(iter_x,x)
33         iter_y = np.append(iter_y,y)
34         iter_count = np.append(iter_count ,i)
35
36         X_prev = X
37         X = X - gamma * Grad(x,y)
38         error = X - X_prev
39         x,y = X[0], X[1]
40     print(X)
41     return X, iter_x,iter_y, iter_count
42
43 def NewtonMethod():
44     ## 1 Newton's Method
45     root,iter_x,iter_y, iter_count =
46     GradientDescent(GradRosenbrock,-2,2)
47     x = np.linspace(-2,2,250)
48     y = np.linspace(-1,3,250)
49     X, Y = np.meshgrid(x, y)
50     Z = Rosenbrock(X, Y)
51
52     #Angles needed for quiver plot
53     anglesx = iter_x[1:] - iter_x[:-1]
54     anglesy = iter_y[1:] - iter_y[:-1]
55
56     ## 2 Surface plot
57     fig = plt.figure(figsize = (16,8))
58     ax = fig.add_subplot(1, 2, 1, projection='3d')
59     ax.plot_surface(X,Y,Z,rstride = 5, cstride = 5, cmap = 'jet', alpha
60     = .4, edgecolor = 'none' )
61     ax.plot(iter_x,iter_y, Rosenbrock(iter_x,iter_y),color = 'r', marker
62     = '*', alpha = .4)
63
64     ax.view_init(45, 280)
65     ax.set_xlabel('x')
66     ax.set_ylabel('y')

```

```

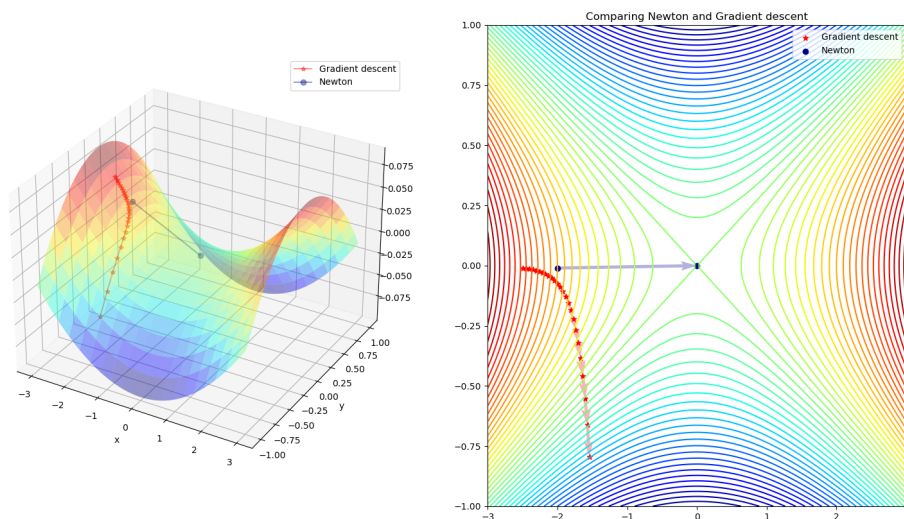
63
64     ax.set_title(r"$f(x) = (1 - x)^2 + 100(y - x^2)^2$", c='g',
horizontalalignment='center', fontsize=10)
65
66     #Contour plot
67     ax = fig.add_subplot(1, 2, 2)
68     ax.contour(X,Y,Z, 50, cmap = 'jet')
69     #Plotting the iterations and intermediate values
70     ax.scatter(iter_x,iter_y,color = 'r', marker = '*')
71     ax.quiver(iter_x[:-1], iter_y[:-1], anglesx, anglesy, scale_units =
'xy', angles = 'xy', scale = 1, color = 'r', alpha = .3)
72     ax.set_title('Gradient Descent with {}
iterations'.format(len(iter_count)))
73     plt.show()
74
75 def main():
76     NewtonMethod()
77
78 if __name__ == '__main__':
79     main()

```

4 鞍点函数测试

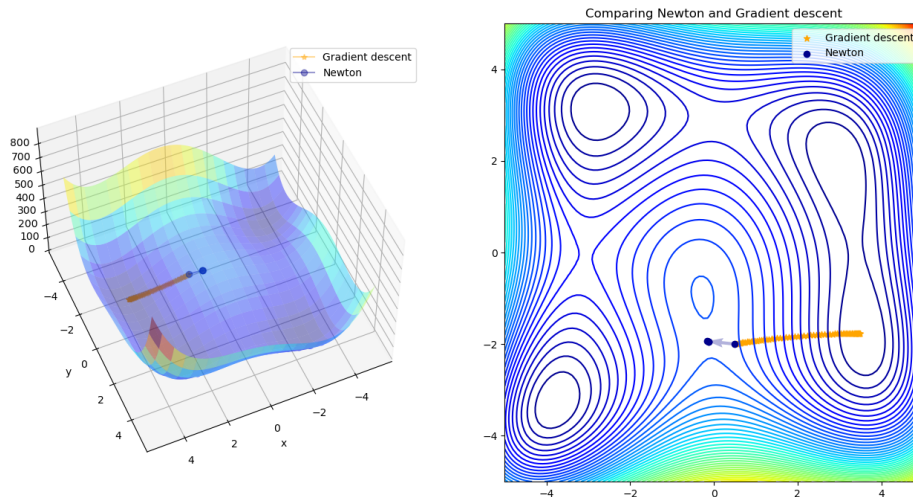
- a single saddle point function

$$f(x, y) = 0.01x^2 - 0.1y^2$$



- multiple saddle points function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$



5 Ceres code

- Ceres Solver求解Rosenbrock函数极值

```

1  // f(x,y) = (1-x)^2 + 100(y - x^2)^2;
2  struct Rosenbrock {
3      bool operator()(const double* parameters, double* cost) const {
4          const double x = parameters[0];
5          const double y = parameters[1];
6          cost[0] = (1.0 - x) * (1.0 - x) + 100.0 * (y - x * x) * (y - x * x);
7          return true;
8      }
9
10     static ceres::FirstOrderFunction* Create() {
11         constexpr int kNumParameters = 2;
12         return new ceres::NumericDiffFirstOrderFunction<Rosenbrock,
13                                                         ceres::CENTRAL,
14                                                         kNumParameters>(
15             new Rosenbrock);
16     }
17 };

```

- C++完整代码

```

1  #include "ceres/ceres.h"
2  #include "glog/logging.h"
3
4  // f(x,y) = (1-x)^2 + 100(y - x^2)^2;
5  struct Rosenbrock {
6      bool operator()(const double* parameters, double* cost) const {
7          const double x = parameters[0];
8          const double y = parameters[1];
9          cost[0] = (1.0 - x) * (1.0 - x) + 100.0 * (y - x * x) * (y - x * x);
10         return true;
11     }
12
13     static ceres::FirstOrderFunction* Create() {
14         constexpr int kNumParameters = 2;
15         return new ceres::NumericDiffFirstOrderFunction<Rosenbrock,

```

```

16                                     ceres::CENTRAL,
17                                     kNumParameters>{
18         new Rosenbrock);
19     }
20 };
21
22 int main(int argc, char** argv) {
23     google::InitGoogleLogging(argv[0]);
24
25     double parameters[2] = {-1.2, 1.0};
26
27     ceres::GradientProblemSolver::Options options;
28     options.minimizer_progress_to_stdout = true;
29
30     ceres::GradientProblemSolver::Summary summary;
31     ceres::GradientProblem problem(Rosenbrock::Create());
32     ceres::Solve(options, problem, parameters, &summary);
33
34     std::cout << summary.FullReport() << "\n";
35     std::cout << "Initial x: " << -1.2 << " y: " << 1.0 << "\n";
36     std::cout << "Final   x: " << parameters[0] << " y: " << parameters[1]
37                 << "\n";
38     return 0;
39 }

```

```

Cost:
Initial          2.420000e+01
Final            3.997452e-20
Change           2.420000e+01

Minimizer iterations          36

Time (in seconds):

  Cost evaluation              0.000000 (0)
  Gradient & cost evaluation    0.000009 (44)
  Polynomial minimization       0.000345
  Total                        0.002054

Termination:          CONVERGENCE (Parameter tolerance reached. Relative step_norm: 1.897782e-11 <= 1.000000e-08.)

Initial x: -1.2 y: 1
Final   x: 1 y: 1

```