# 1　算法概述

## 1.1 最速下降法

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。

## 1.2 阻尼牛顿法

步 0　给定终止误差值 $0 \le \varepsilon \ll 1$, $\delta \in (0,1)$, $\sigma \in (0, 0.5)$. 初始点 $x_0 \in \mathbb{R}^n$. 令 $k : 0$.

步 1　计算 $g_k = \nabla f(x_k)$. 若 $\|g_k\| \le \varepsilon$, 停算, 输出 $x^* \approx x_k$.

步 2　计算 $G_k = \nabla^2 f(x_k)$, 并求解线性方程组得解 $d_k$:

$$G_k d = -g_k \tag{3.6}$$

步 3　记 $m_k$ 是满足下列不等式的最小非负整数 $m$:

$$f(x_k + \delta^m d_k) \le f(x_k) + \sigma \delta^m g_k^T d_k. \tag{3.7}$$

步 4　令 $\alpha_k = \delta^{m_k}$, $x_{k+1} = x_k + \alpha_k d_k$, $k := k+1$, 转步 1.



## 1.3 BFGS 方法

（1）给定$x^1$，$\varepsilon > 0$。

（2）$H_1 = I_n$，计算$g_1 = \nabla f(x^1)$，$k = 1$。

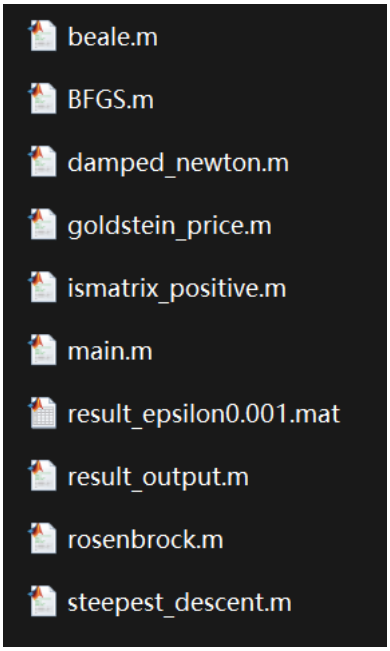
（3）$d^k = -H_k g_k$。

（4）$f(x^k + \alpha_k d^k) = \min_{\alpha \in [0,\infty)} f(x^k + \alpha d^k)$，则$x^{k+1} = x^k + \alpha_k d^k$。

（5）若$||\nabla f(x^{k+1})|| < \varepsilon$，停止，得$\bar{x} = x^{k+1}$否则执行（6）。

（6）$g_{k+1} = \nabla f(x^{k+1})$，$p^k = x^{k+1} - x^k$，$q^k = g_{k+1} - g_k$，计算$H_{k+1}$，令$k = k + 1$，返回（3）。

## 2 程序编写




程序架构如下：

（1）rosenbrock、beale、goldstein_price 分别为对应函数的求值函数

```
function [value, gradient_q, hessian_q] = goldstein_price(x)
% input: [x; y] output: value gradient hessian
function [value, gradient_q, hessian_q] = rosenbrock(x)
% input: [x; y] output: value gradient hessian
function [value, gradient_q, hessian_q] = beale(x)
% input: [x; y] output: value gradient hessian
```

（2）damped_newton、steepest_descent、BFGS 分别为对应算法的迭代函数

```
function [x_bar, fmin, data] = BFGS(func, x0, epsilon)
    % output: x_bar——最优点 fmin——最优值
    % input: func——目标函数 x0——初始点 epsilon——允许误差
function [x_bar, fmin, data] = damped_newton(func, x0, epsilon)
    % output: x_bar——最优点 fmin——最优值
    % input: func——目标函数 x0——初始点 epsilon——允许误差
function [x_bar, fmin, data] = steepest_descent(func, x0, epsilon)
    % output: x_bar——最优点 fmin——最优值
    % input: func——目标函数 x0——初始点 epsilon——允许误差
```

（3）result_output 为数据可视化输出函数

```
function [rosenbrock_t, beale_t, goldstein_t] =
result_output(iterative_method, epsilon, x0_rosenbrock, x0_beale,
x0_goldstein)
    % input: iterative_method——迭代方法
    %        epsilon——允许误差 default: 0.001
    %        x0——迭代初始点 defalut: (0, 0)
```

（4）main 函数调用 result_output 函数输出计算结果

```
[rosenbrock_SteepestDescent_result, beale_SteepestDescent_result,
goldstein_SteepestDescent_result] = result_output(@steepest_descent);

[rosenbrock_DampedNewton_result, beale_DampedNewton_result,
goldstein_DampedNewton_result] = result_output(@damped_newton);

[rosenbrock_BFGS_result, beale_BFGS_result, goldstein_BFGS_result] =
result_output(@BFGS);
```

## 3 优化结果

计算的结果均保存在对应的结构体中


>> disp(beale_BFGS_result)

x_bar: [2×1 double]

fmin: 3.2917e-09

data: [1×1 struct]

iteration_process: [10×4 table]

## 3.1 问题最优解

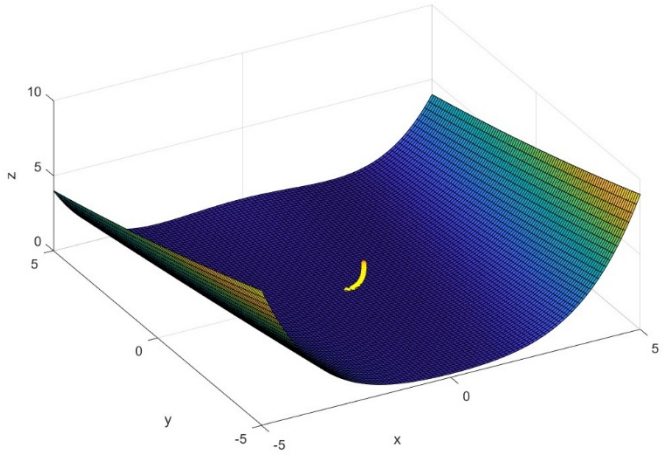
迭代初始点与允许误差设置如下：

```
%       epsilon——允许误差 default: 0.001
%       x0——迭代初始点 defalut: (0, 0)
% 为参数赋默认值
if nargin < 2 || isempty(epsilon)
    epsilon = 0.001;
end
if nargin < 3 || isempty(x0_rosenbrock)
    x0_rosenbrock = [0; 0];
end
if nargin < 4 || isempty(x0_beale)
    x0_beale = [4; 0.6];
end
if nargin < 5 || isempty(x0_goldstein)
    x0_goldstein = [0.5; -0.5];
    % goldstein price 函数具有多个局部极小点，需要选取合适初始点才可迭代到全局最优点
end
```

### 3.1.1 rosenbrock 函数优化结果

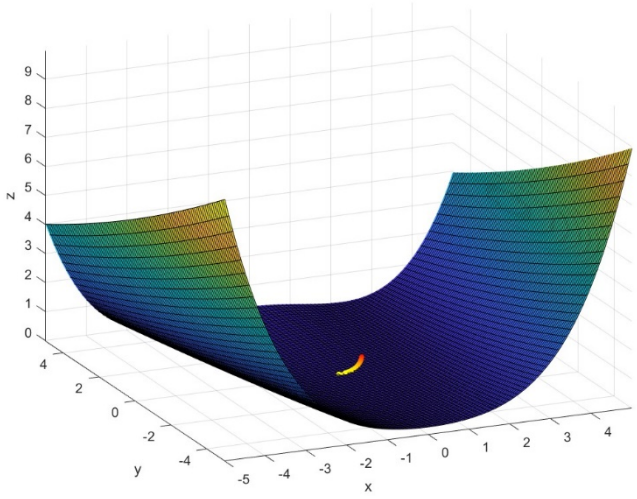
（1）最速下降算法迭代结果

| | |
|---|---|
| x_bar | [0.9991;0.9981] |
| fmin | 8.8341e-07 |
| data | *1x1 struct* |
| iteration_process | *5843x4 table* |



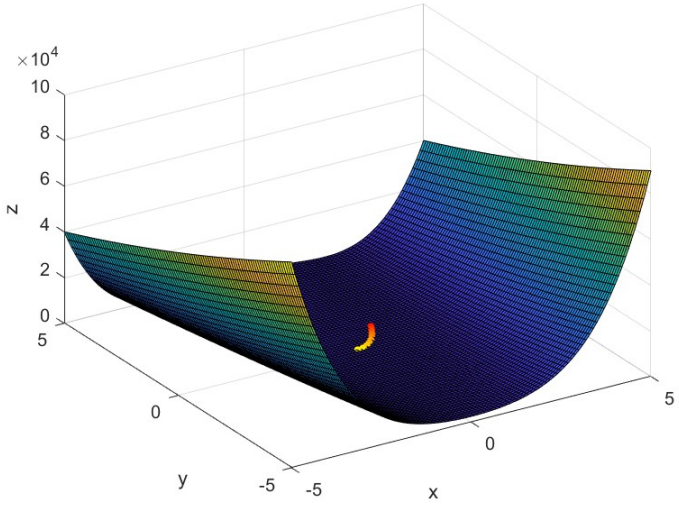

（2）阻尼牛顿法算法迭代结果

| | |
|---|---|
| x_bar | [1.0000;1.0000] |
| fmin | 1.2882e-11 |
| data | *1x1 struct* |
| iteration_process | *141x4 table* |




（3）BFGS 算法迭代结果

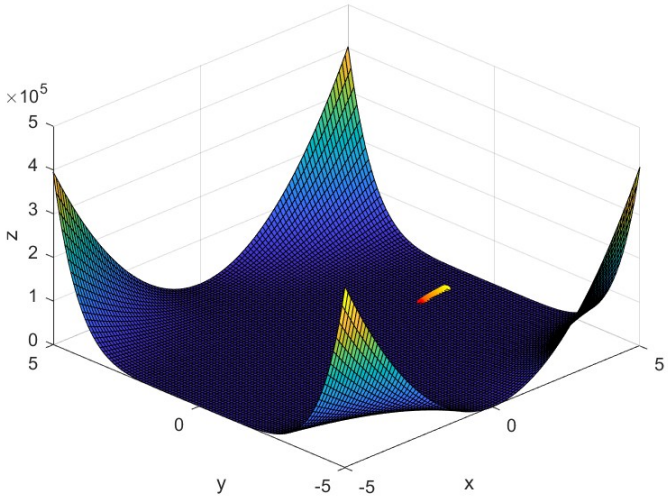
| | |
|---|---|
| x_bar | [0.9995;0.9990] |
| fmin | 2.4675e-07 |
| data | *1x1 struct* |
| iteration_process | *21x4 table* |




### 3.1.2 beale 函数优化结果

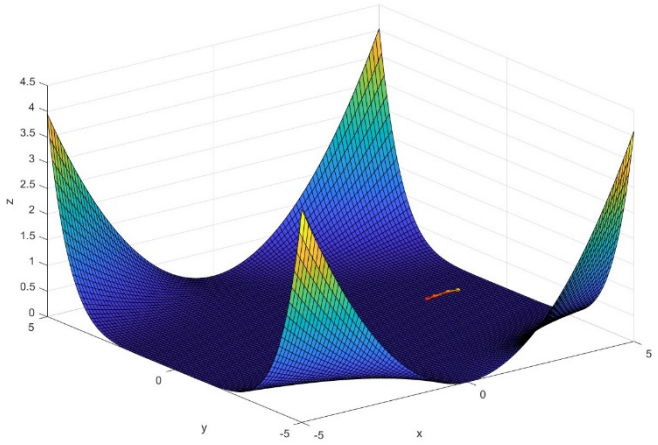
（1）最速下降算法迭代结果

| | |
|---|---|
| x_bar | [3.0024;0.5006] |
| fmin | 9.3271e-07 |
| data | *1x1 struct* |
| iteration_process | *844x4 table* |



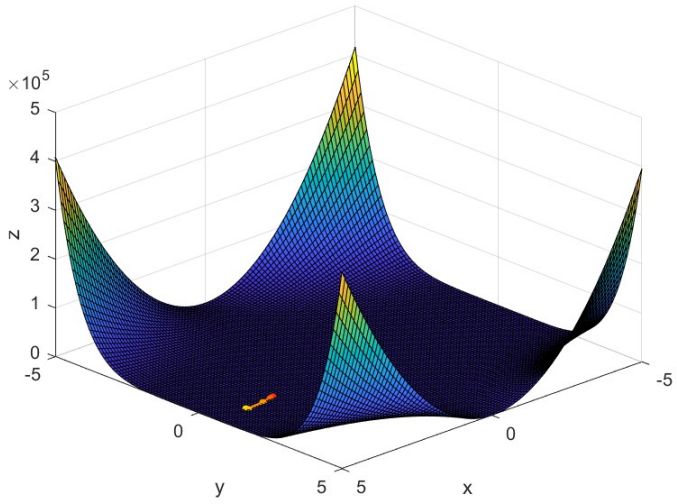

（2）阻尼牛顿法算法迭代结果

| | |
|---|---|
| x_bar | [2.9999;0.5000] |
| fmin | 1.2486e-09 |
| data | *1x1 struct* |
| iteration_process | *5x4 table* |




（3）BFGS 算法迭代结果

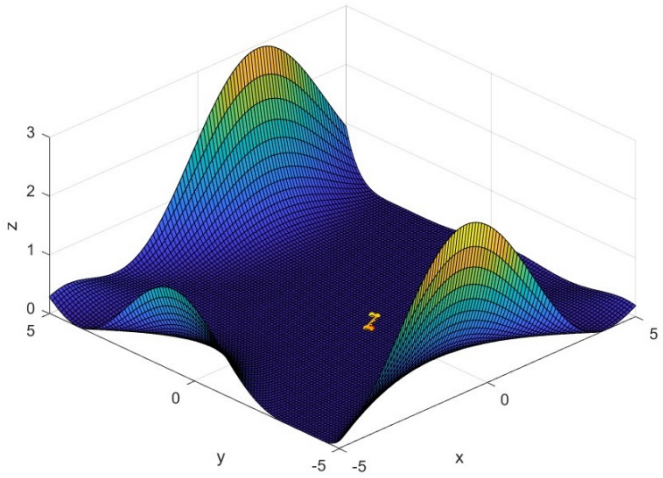
| | |
|---|---|
| x_bar | [3.0000;0.5000] |
| fmin | 3.2917e-09 |
| data | *1x1 struct* |
| iteration_process | *10x4 table* |




### 3.1.3　goldstein_price 函数优化结果

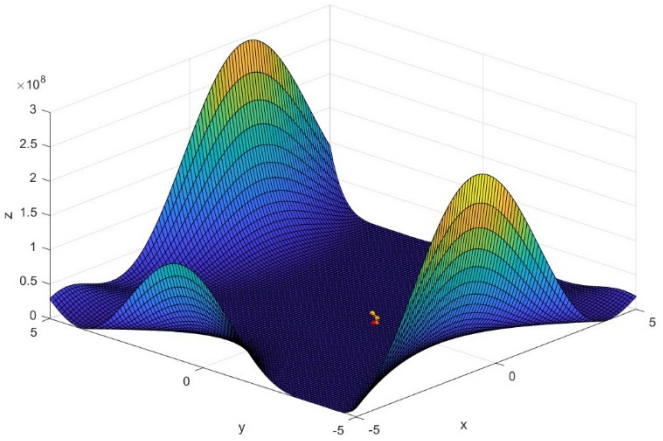
（1）最速下降算法迭代结果

| | |
|---|---|
| x_bar | [9.4229e-07;-1.0000] |
| fmin | 3.0000 |
| data | *1x1 struct* |
| iteration_process | *11x4 table* |



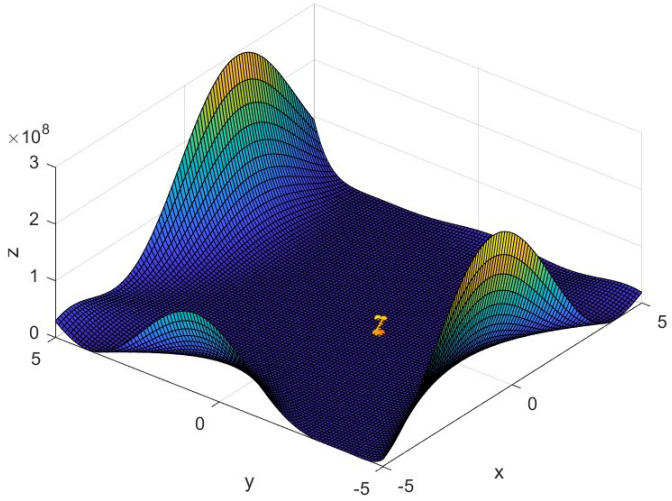

（2）阻尼牛顿法算法迭代结果

| | |
|---|---|
| x_bar | [-3.0923e-09;-1.0000] |
| fmin | 3.0000 |
| data | *1x1 struct* |
| iteration_process | *6x4 table* |




（3）BFGS 算法迭代结果

| | |
|---|---|
| x_bar | [-3.9749e-07;-1.0000] |
| fmin | 3.0000 |
| data | *1x1 struct* |
| iteration_process | *10x4 table* |

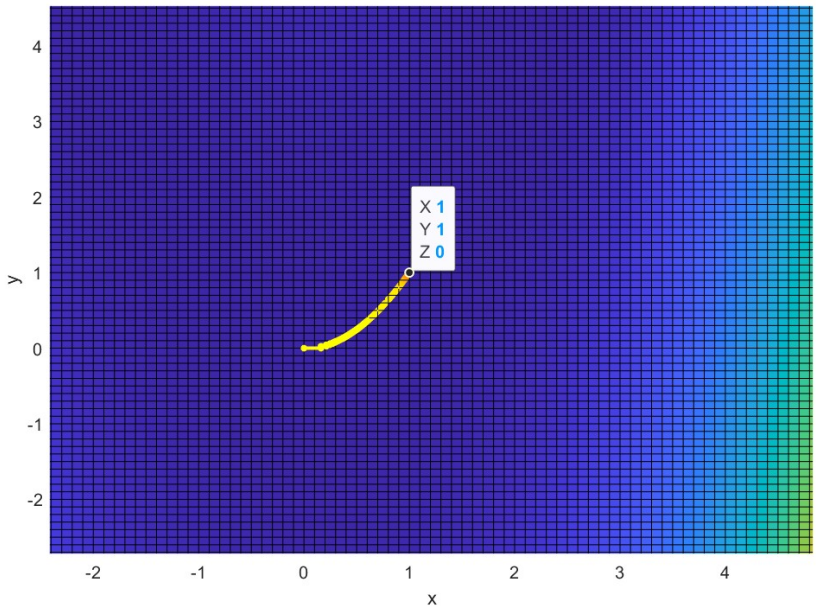



## 3.2 收敛路径与收敛速率

收敛路径由黄色变为红色，代表迭代由初始点到达最终点，越靠近迭代重点红色越突出。

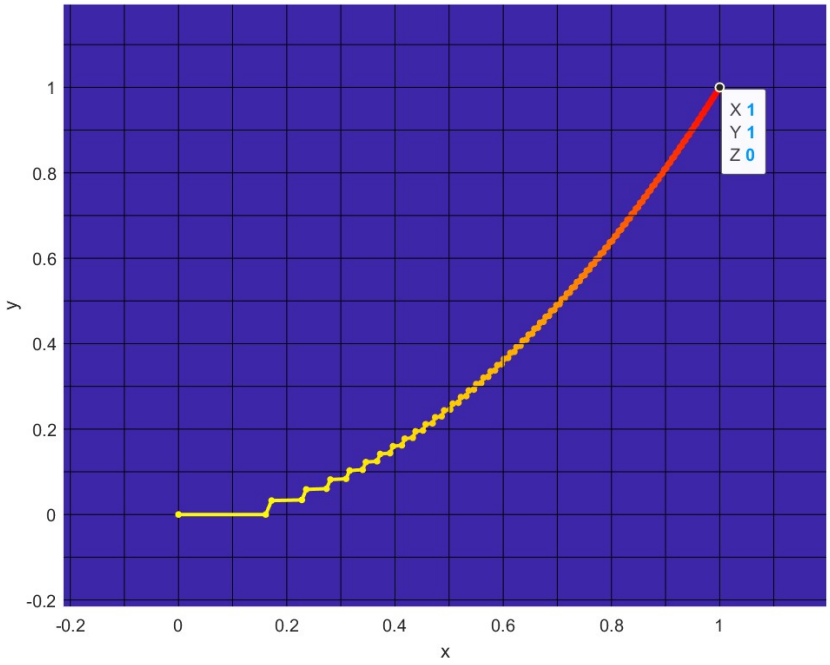
### 3.2.1 Rosenbrock 函数收敛路径

（1）最速下降法收敛路径



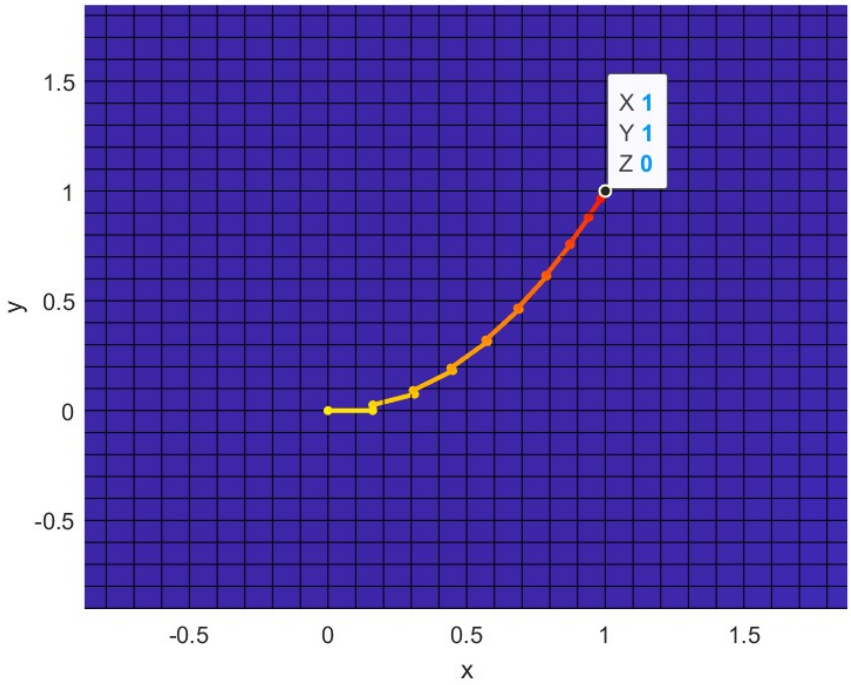

共迭代 5483 次

（2）阻尼牛顿法收敛路径




共迭代 141 次

（3）BFGS 法收敛路径




共迭代 21 次

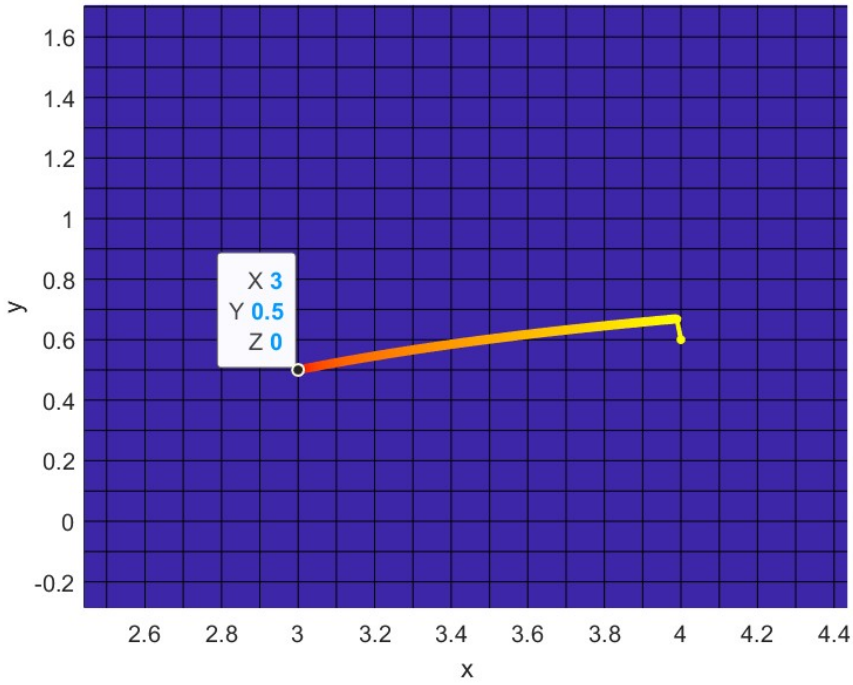
收敛速率分析：

从迭代过程图中可看出 BFGS 的收敛速率最快，最速下降法的收敛速率最慢。从表格（完整表格存于对应的结构体数据）中也可看出，由于最速下降法存在锯齿现象，在极小点附近步长极小。

| 1<br>Index | 2<br>Value | 3<br>x | 4<br>y |
|---|---|---|---|
| 5824 | 9.1405e-07 | 0.9990 | 0.9981 |
| 5825 | 9.1244e-07 | 0.9990 | 0.9981 |
| 5826 | 9.1088e-07 | 0.9990 | 0.9981 |
| 5827 | 9.0933e-07 | 0.9990 | 0.9981 |
| 5828 | 9.0773e-07 | 0.9990 | 0.9981 |
| 5829 | 9.0612e-07 | 0.9990 | 0.9981 |
| 5830 | 9.0446e-07 | 0.9990 | 0.9981 |
| 5831 | 9.0280e-07 | 0.9991 | 0.9981 |
| 5832 | 9.0105e-07 | 0.9991 | 0.9981 |
| 5833 | 8.9930e-07 | 0.9991 | 0.9981 |
| 5834 | 8.9765e-07 | 0.9991 | 0.9981 |
| 5835 | 8.9600e-07 | 0.9991 | 0.9981 |
| 5836 | 8.9443e-07 | 0.9991 | 0.9981 |
| 5837 | 8.9285e-07 | 0.9991 | 0.9981 |
| 5838 | 8.9133e-07 | 0.9991 | 0.9981 |
| 5839 | 8.8980e-07 | 0.9991 | 0.9981 |
| 5840 | 8.8824e-07 | 0.9991 | 0.9981 |
| 5841 | 8.8667e-07 | 0.9991 | 0.9981 |

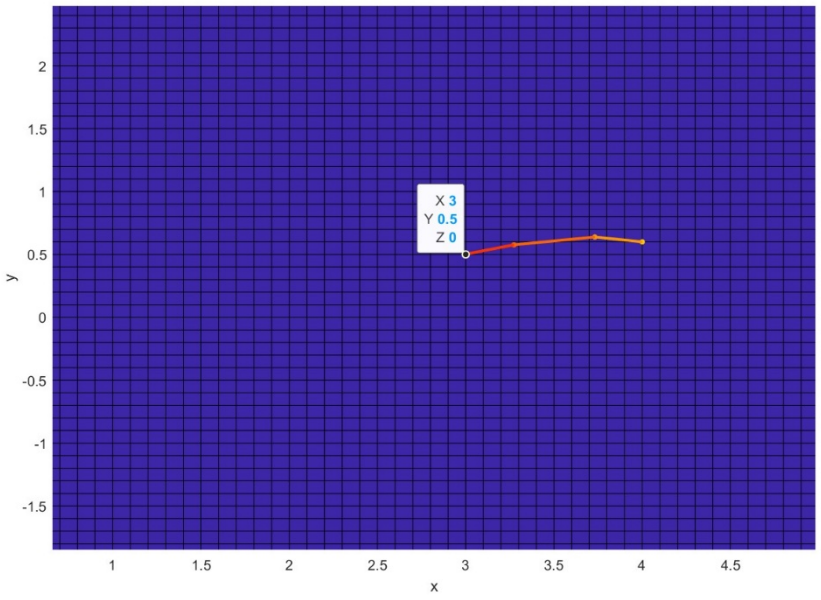
### 3.2.2 Beale 函数收敛路径

（1）最速下降法收敛路径



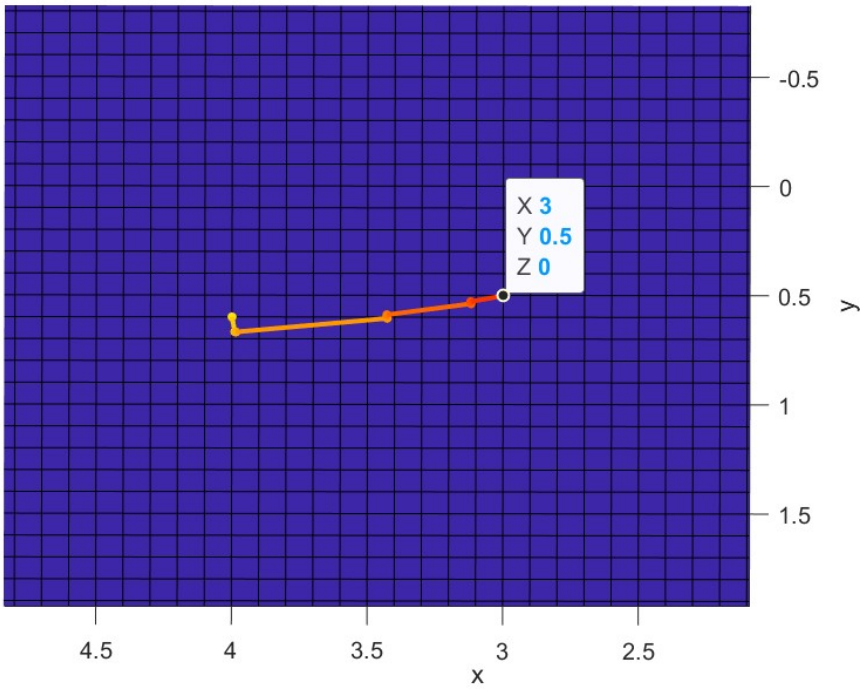

共迭代 844 次

（2）阻尼牛顿法收敛路径




共迭代 5 次

（3）BFGS 法收敛路径



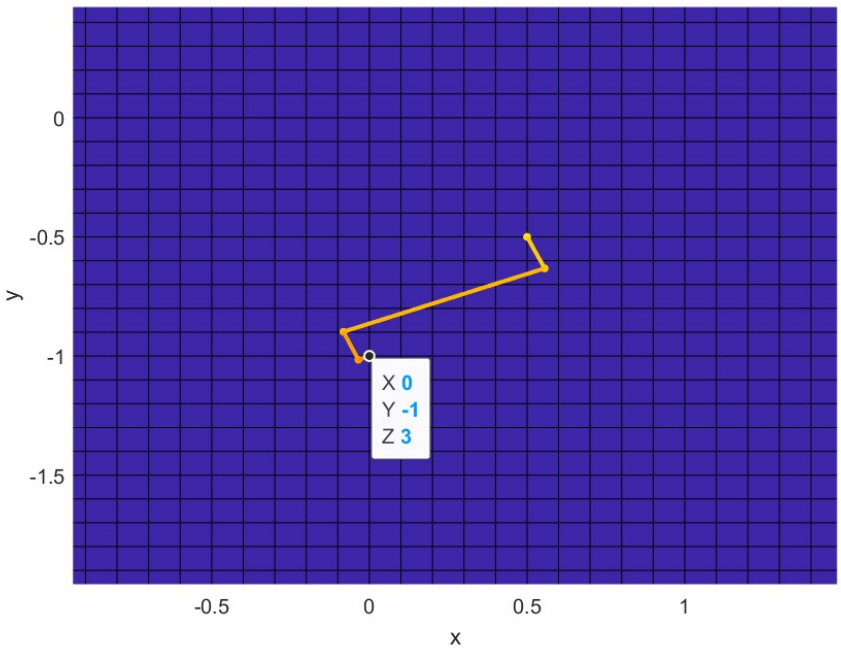

共迭代 10 次

从迭代过程图中可看出阻尼牛顿法的收敛速率最快，最速下降法的收敛速率最慢。从表格（完整表格存于对应的结构体数据）中也可看出，由于最速下降法存在锯齿现象，在极小点附近步长极小。

| 1<br>Index | 2<br>Value | 3<br>x | 4<br>y |
|---|---|---|---|
| 825 | 1.4985e-06 | 3.0030 | 0.5008 |
| 826 | 1.4617e-06 | 3.0030 | 0.5007 |
| 827 | 1.4257e-06 | 3.0030 | 0.5008 |
| 828 | 1.3906e-06 | 3.0029 | 0.5007 |
| 829 | 1.3564e-06 | 3.0029 | 0.5007 |
| 830 | 1.3229e-06 | 3.0029 | 0.5007 |
| 831 | 1.2904e-06 | 3.0028 | 0.5007 |
| 832 | 1.2586e-06 | 3.0028 | 0.5007 |
| 833 | 1.2275e-06 | 3.0028 | 0.5007 |
| 834 | 1.1973e-06 | 3.0027 | 0.5007 |
| 835 | 1.1678e-06 | 3.0027 | 0.5007 |
| 836 | 1.1390e-06 | 3.0027 | 0.5007 |
| 837 | 1.1109e-06 | 3.0026 | 0.5007 |
| 838 | 1.0835e-06 | 3.0026 | 0.5006 |
| 839 | 1.0568e-06 | 3.0026 | 0.5007 |
| 840 | 1.0307e-06 | 3.0025 | 0.5006 |
| 841 | 1.0053e-06 | 3.0025 | 0.5006 |
| 842 | 9.8051e-07 | 3.0025 | 0.5006 |

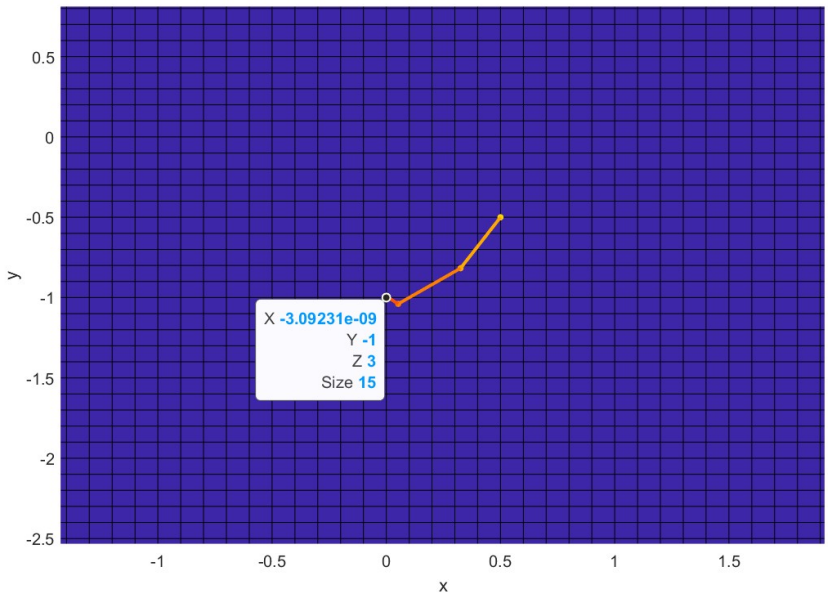
### 3.2.3　Goldstein_price 函数收敛路径如下

（1）最速下降法收敛路径



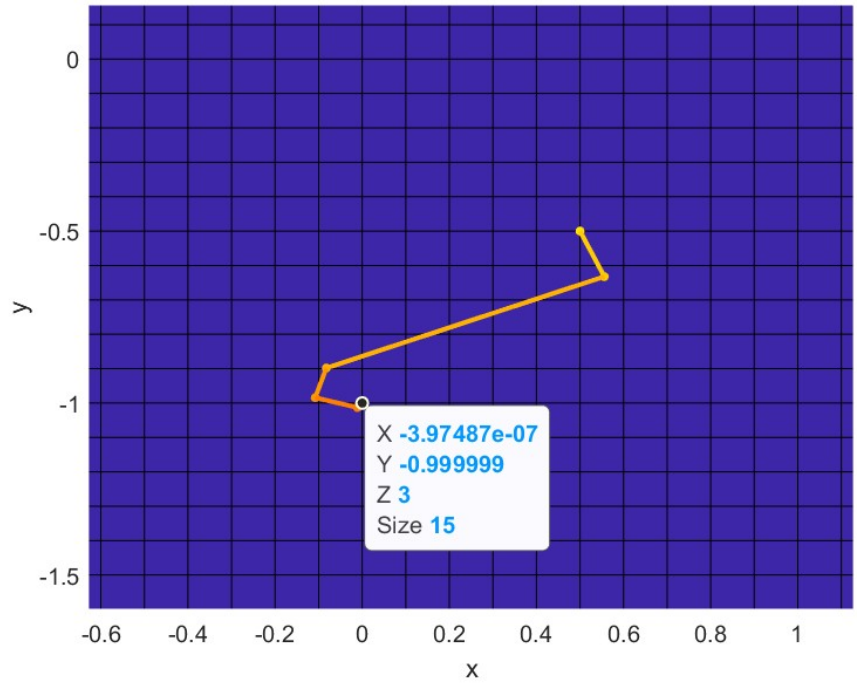

共迭代 11 次

（2）阻尼牛顿法收敛路径



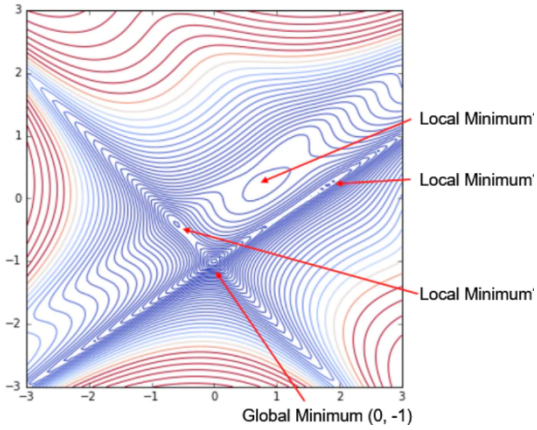

共迭代 6 次

（3）BFGS 法收敛路径

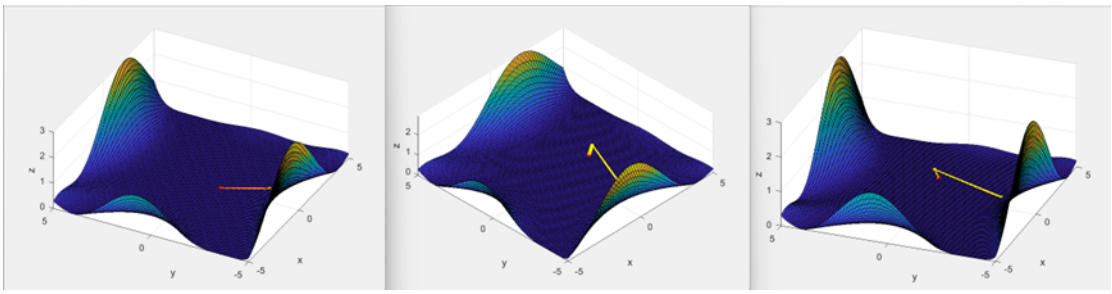



共迭代 10 次

从迭代过程图中可看出阻尼牛顿法与 BFGS 方法的收敛速度相近。由于 goldstein_price 函数具有多个局部极小点，所以迭代初始点的选择较为重要，否则会收敛至局部极小而不是全局极小。




本次迭代选择的初始点为[-0.5, 0.5]，三种算法均可以迭代到全局极小。选择初始点为[1, -3]则只有阻尼牛顿法会到达全局极小点。



# 4　算法优缺点

## 4.1 最速下降法

从迭代过程可以看出，最速下降法往往仅在迭代开始时具有较快的收敛速度，在三种目标函数的迭代过程中，最速下降法的迭代次数总是远远超过其他两种算法。

优点：

（1）程序简单，计算量小；

（2）对初始点没有特别的要求；

（3）最速下降法是整体收敛的，且为线性收敛

缺点：

（1）它只在局部范围内具有“最速”属性，对整体求解过程，它的下降速度是缓慢的；

（2）靠近极小值时速度减慢；

（3）存在锯齿现象

## 4.2 阻尼牛顿法

阻尼牛顿法因为牛顿方向不一定是下降方向，所以在远离极小点时很有可能不收敛，在迭代开始时需要选择合适的初始点。

优点：

（1）二次终止性：在满足某些条件下，阻尼牛顿法在收敛时至少二级收敛，它的收敛速度非常快。

（2）稳定性：通过引入步长，阻尼牛顿法可以避免牛顿法在某些情况下可能出现的发散问题。

（3）适用于非线性问题：阻尼牛顿法可以应用于寻找非线性函数的最小值或零点。

缺点：

（1）计算成本：阻尼牛顿法需要在每一步计算函数的黑塞矩阵（Hessian）及其逆，在处理大规模问题时算力的开支非常昂贵。

（2）黑塞矩阵的正定性：阻尼牛顿法需要假设黑塞矩阵是正定的，在实际问题中并不总是成立。

（3）初始点选择：和其他迭代优化方法一样，阻尼牛顿法的收敛性和收敛速度受到初始点的影响。

## 4.2 BFGS 方法

BFGS 方法是拟牛顿法的一种，它克服了牛顿法中黑塞矩阵可能非正定的问题，同样也具有二次终止性。在三种函数的迭代过程中可以看出，BFGS 方法具有良好的收敛性与迭代速度。

优点：

（1）避免直接计算黑塞矩阵：BFGS 方法只需要计算目标函数的一阶导数，而不是直接计算二阶导数（黑塞矩阵），减少了计算复杂性和计算成本。

（2）超线性收敛速度：BFGS 方法在良好条件下可以具有超线性的收敛速度，这比梯度下降法的线性收敛速度要快得多。

缺点：

（1）存储和计算成本：尽管 BFGS 方法避免了直接计算黑塞矩阵，但是需要存储和更新一个大小为 n*n 的矩阵，这在 n 非常大的情况下可能会导致存储和计算成本显著增加。

（2）线搜索的复杂性：BFGS 方法需要在每一步进行线搜索，以确定沿搜索方向的步长，增加了计算成本。

（3）对初始点和初始矩阵的依赖性：BFGS 方法的收敛性和收敛速度可能会受到初始点和初始矩阵选择的影响。