HW5

白博臣，何骐多，夏营

1 section1

牛顿迭代法是一种经典的求解函数根的方法，它是一种迭代方法。其思想为沿着当地的切线方向不断移动计算点，使其不断逼近真实根值。假设初始值为 x_k ，函数为 $f(x)$ ，则

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

其下一步预测的根就是 x_{k+1}。然后逐次迭代，x值会越来越接近真实的根植。

本题目中 $f = z^3 - 1$，其牛顿迭代求根的迭代方程为

$$x_{k+1} = x_k - \frac{x_k^3 - 1}{3x_k^2}$$

该方程有三个根，其实部与虚部都在 $[-2, 2]$ 区间内：

我们使用 Matlab 进行相关运算处理：

使用 parfor 循环来实现并行计算。parfor 循环是 MATLAB 中用于并行计算的一种机制，它会自动将循环迭代分配到多个处理器上并行执行，从而加速计算过程。

在循环内部，每次迭代都是独立的，不涉及对前一次迭代结果的依赖，这样可以确保并行计算的正确性。

GPU 加速运算：

首先，需要将数据移动到 GPU 上进行计算。在代码中，使用 gpuArray 函数将复数网格 X 的一行数据（X(i, :)）移动到 GPU 上，以便后续在 GPU 上进行计算。

然后，利用 GPU 提供的并行计算能力，通过调用 arrayfun 函数对每个网格点进行计算。arrayfun 函数可以将一个函数应用到数组中的每个元素，并在 GPU 上并行执行这些函数调用，从而加速计算过程。

在这里，solve 函数被应用到 GPU 上的每个网格点上，以计算方程的解。

计算完成后，使用 gather 函数将计算结果从 GPU 移回 CPU，以便后续处理和可视化。

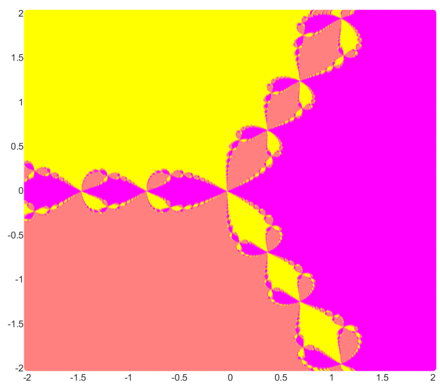
通过以上步骤，实现了在 GPU 上并行计算复平面上方程的根，大大加速了计算过程。

使用 Matlab 代码实现如下

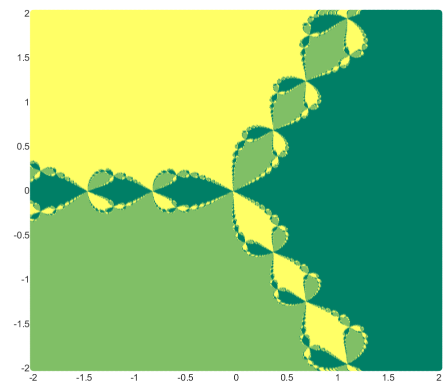
```
clc; clear; close all;
N = 20000;
epsilon = 1e-3;
root1 = 1;
root2 = -1 / 2 + sqrt(3)/2i;
root3 = -1 / 2 - sqrt(3)/2i;
x = linspace(-2, 2, N);
y = linspace(-2, 2, N);
```

```
[xgrid, ygrid] = meshgrid(x, y);
X = xgrid + 1i*ygrid;

% 使用parfor循环并在循环内部利用GPU加速的函数
A = zeros(N);
tic; % 开始计时
parfor i = 1:N
    % 将数据移动到GPU
    X_gpu = gpuArray(X(i, :));

    % 在每次迭代中，利用GPU加速计算结果
    A_gpu = arrayfun(@solve, X_gpu);

    % 将计算结果从GPU移回CPU
    A(i, :) = gather(A_gpu);

end

% 对计算结果进行处理
A(abs(A - root1) < epsilon) = 0;
A(abs(A - root2) < epsilon) = 1;
A(abs(A - root3) < epsilon) = 2;


% 绘制图像
% fig = figure('color',[1 1 1],'position',[400,100,500*1.5,416*1.5], 'Visible','off');
% contourf(x, y, A, 'LineStyle', 'none');
% clim([0, 2])
% colormap("cool")
% colorbar;
% saveas(fig, 'plot.png');

toc;

% writematrix(A, "result.txt")

```

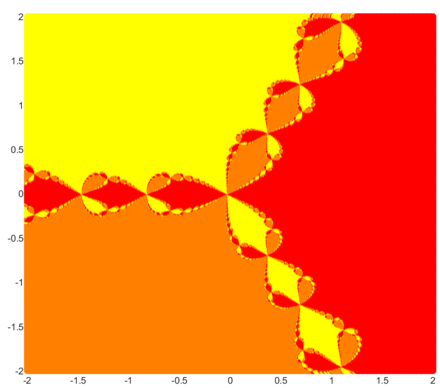
运行结果如下



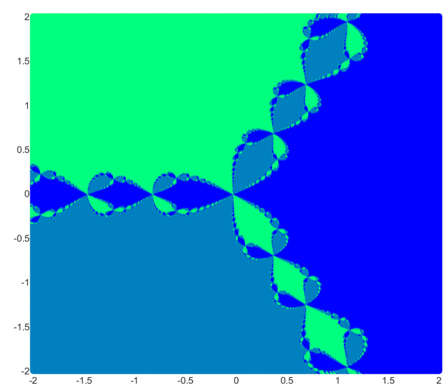
(a) spring theme



(b) summer theme



(c) autumn theme



(d) winter theme

图 1: NB 运行结果

2 Problem2

利用 Desmos 作图 利用 NR-Bisection 求解方程 $4cos(x) - e^x = 0$ ，使用 Java 代码实现如下

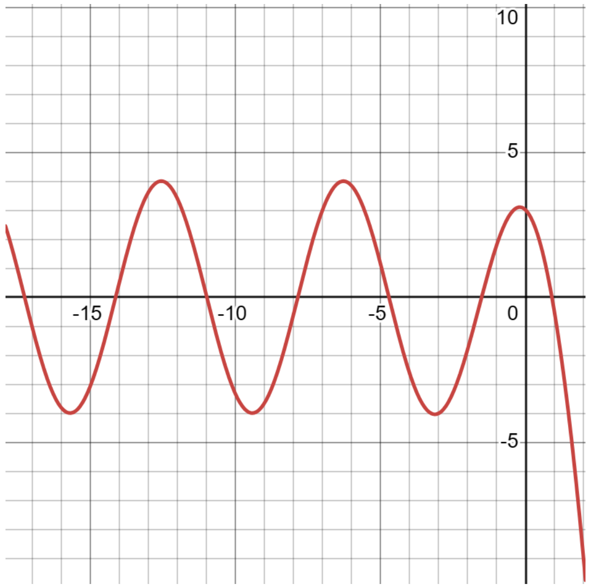


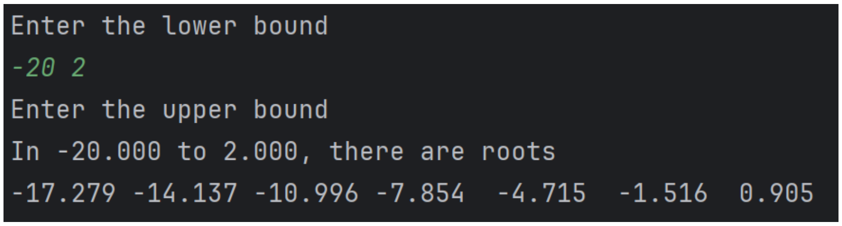
图 2: 函数图像绘制

```java
/**
 * This class demonstrates the numerical methods of finding roots using a combination of NR and Bisection methods.
 */

import java.util.Scanner;

import static java.lang.StrictMath.*;

public class NRBisection {
    static double epsilon = 1e-10;

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the lower bound");
        double xl = scanner.nextDouble();
        System.out.println("Enter the upper bound");
        double xu = scanner.nextDouble();
        int num = 100;
        int n = (int) ((xu - xl) * num);

        System.out.printf("In %.3f to %.3f, there are roots\n", xl, xu);
        for (int i = 0; i <= n; i++) {
            double x1 = xl + (double) i / num;
            double x2 = xl + (double) (i + 1) / num;
```

```
            if (function(x1) * function(x2) < 0) {
                findRoot(x1, x2);
            }
        }
    }

    /**
     * Finds the root of the function within the given range.
     *
     * @param x1 The lower bound of the range
     * @param x2 The upper bound of the range
     */
    private static void findRoot(double x1, double x2) {
        double x0;
        if (abs(function(x1)) < abs(function(x2))) {
            x0 = x1;
        } else {
            x0 = x2;
        }
        double root = solve(x1, x2, x0);
        System.out.printf("%.3f\t", root);
    }

    /**
     * Defines the function for which the root is to be found.
     *
     * @param x The input value
     * @return The value of the function at x
     */
    private static double function(double x) {
        return 4 * cos(x) - exp(x);
    }

    /**
     * Computes the derivative of the function at a given point.
     *
     * @param x The point at which the derivative is to be computed
     * @return The value of the derivative at x
     */
    private static double derivative(double x) {
        return -4 * sin(x) - exp(x);
    }

    /**
     * Implements the NR method to find a new approximation for the root.
     *
     * @param x The current approximation for the root
     * @return The new approximation using the NR method
```

```java
     */
    private static double xNR(double x) {
        return x - function(x) / derivative(x);
    }

    /**
     * Recursively applies the combined NR and Bisection methods to find the root.
     *
     * @param x1 The lower bound of the range
     * @param x2 The upper bound of the range
     * @param x  The current approximation for the root
     * @return The calculated root
     */
    private static double solve(double x1, double x2, double x) {
        if (abs(function(x)) < epsilon) {
            return x;
        } else {
            double xN = xNR(x);

            if (x1 < xN && x2 > xN) {
                return solve(x1, x2, xN);
            } else {
                xN = (x1 + x2) / 2;
                if (function(x1) * function(xN) <= 0) {
                    // Update parameter
                    if (abs(function(x1)) < abs(function(xN))) {
                        return solve(x1, xN, x1);
                    } else {
                        return solve(x1, xN, xN);
                    }
                } else {
                    if (abs(function(x2)) < abs(function(xN))) {
                        return solve(x2, xN, x2);
                    } else {
                        return solve(x2, xN, xN);
                    }
                }
            }
        }
    }
}
```

运行结果如下



```
Enter the lower bound
-20 2
Enter the upper bound
In -20.000 to 2.000, there are roots
-17.279 -14.137 -10.996 -7.854  -4.715  -1.516  0.905
```

图 3: NRBisection 运行结果

3 Problem3

利用 Desmos 绘制函数图像，观察 h 的变化对根的影响.

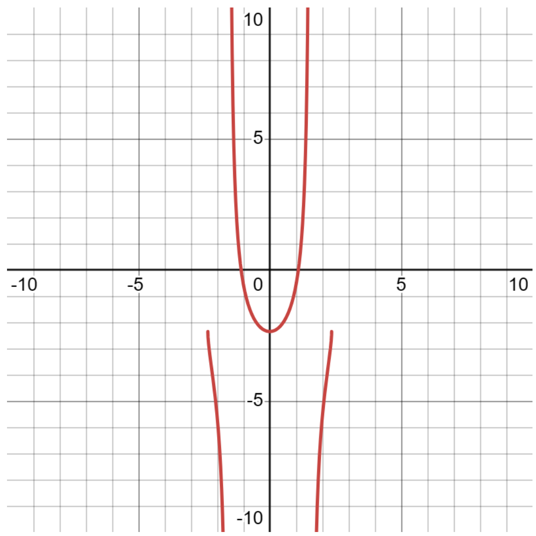


图 4: $y = f(x)$ 的图像

观察可得，对 $f(x)$ 而言在 $n\pi < h < (n+1)\pi(n = 0, 1, ...)$ 时，有 (n+1) 个根。根的范围

$$\begin{aligned} k\pi &< x_k < \left(k + \frac{1}{x}\right)\pi \\ -\left(k + \frac{1}{x}\right)\pi &< x_k < -k\pi \quad (k = 1, 2, ..., n) \end{aligned} \tag{1}$$

而对于 $g(x)$，在 $\left(n + \frac{1}{x}\right)\pi < h < \left(n + \frac{3}{2}\right)\pi(n = 0, 1, ...)$ 时，有 (n+1 个根。根的范围

$$\begin{aligned} \left(k + \frac{1}{x}\right)\pi &< x_k < (k+1)\pi \\ -(k+1)\pi &< x_k < -\left(k + \frac{1}{x}\right)\pi \quad (k = 1, 2, ..., n) \end{aligned} \tag{2}$$

利用 full Muller-Brent 求解根，使用 Java 代码实现如下

```java
import java.util.Scanner;
import java.util.function.Function;

import static java.lang.StrictMath.*;
```

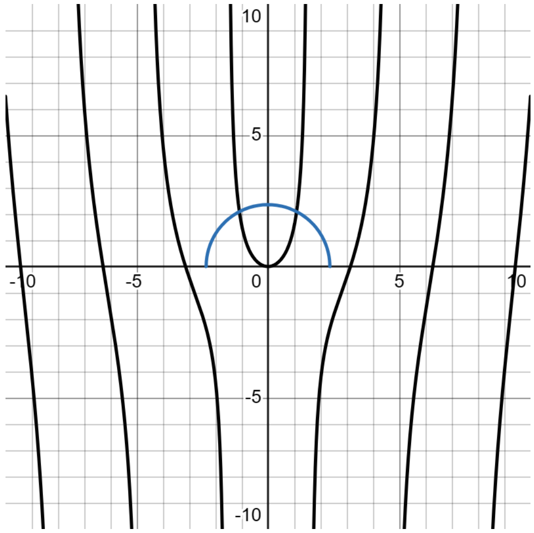


图 5: $y = x\tan x$ 和 $y = sqrth^2 - x^2$ 的图像

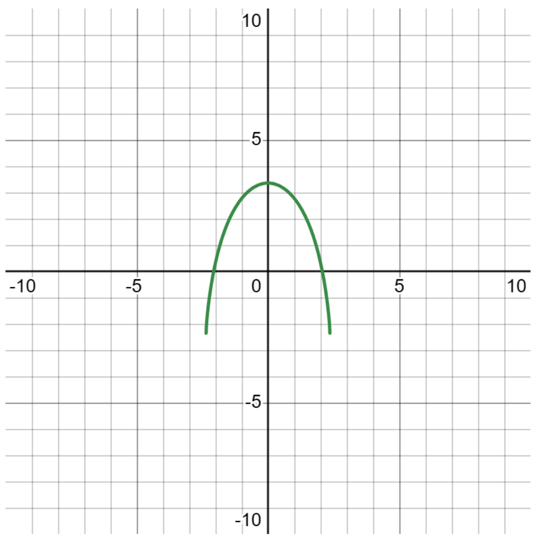


图 6: $y = g(x)$ 的图像

```java

/**
 * This class implements the Muller's Bisection method to find roots of transcendental equations.
 */
public class FullMB {

    static int N = 100;
    static double epsilon = 1e-10;

    public static void main(String[] args) {

        System.out.println("Enter the h");
        Scanner scanner = new Scanner(System.in);
        double h = scanner.nextDouble();
```

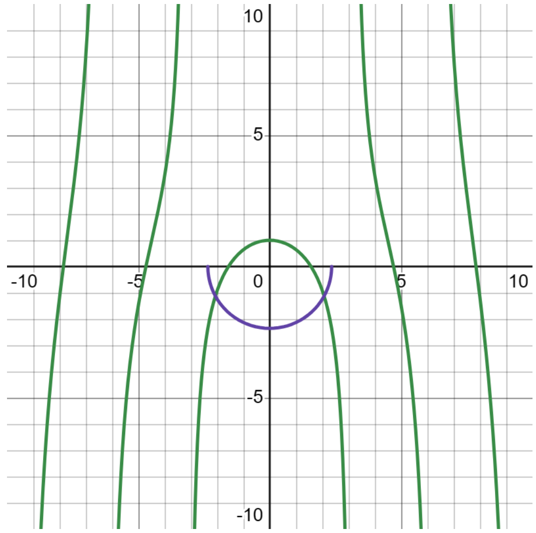


图 7: $y = x\cot x$ 和 $y = -sqrt h^2 - x^2$ 的图像

```

        Function<Double, Double> f = (x) -> x * tan(x) - sqrt(h * h - x * x);
        Function<Double, Double> g = (x) -> x / tan(x) + sqrt(h * h - x * x);

        System.out.println("Root of x tan(x) - sqrt(h^2 - x^2) = 0");
        int n = (int) (abs(h / PI));
        for (int i = n; i >= 0; i--) {
            double xl = -(i + 1.0 / 2) * PI;
            double xu = -i * PI;
            rank(xl, xu, f);
        }
        for (int i = 0; i <= n; i++) {
            double xl = i * PI;
            double xu = (i + 1.0 / 2) * PI;
            rank(xl, xu, f);
        }
        System.out.println();
        System.out.println("Root of x cot(x) + sqrt(h^2 - x^2) = 0");
        n = (int) (abs(h / PI) - 1.0 / 2);
        for (int i = n; i >= 0; i--) {
            double xl = -(i + 1) * PI;
            double xu = -(i + 1.0 / 2) * PI;
            rank(xl, xu, g);
        }
        for (int i = 0; i <= n; i++) {
            double xl = (i + 1.0 / 2) * PI;
            double xu = (i + 1) * PI;
            rank(xl, xu, g);
        }
    }

```

```java
    /**
     * Find roots within a given range using the Muller's Bisection method.
     *
     * @param xl Lower bound of the range
     * @param xu Upper bound of the range
     * @param f  The function for which roots are to be found
     */
    public static void rank(double xl, double xu, Function<Double, Double> f) {
        double step = 0.001;
        double n = (xu - xl) / step;
        for (int i = 0; i < n - 3; i++) {
            double x = xl + i * step;
            if (f.apply(x) * f.apply(x + step) < 0) {
                double root = fullMB(x, x + step, x + 2 * step, f);
                System.out.printf("%.3f\t", root);
            }
        }
    }

    /**
     * Muller's Bisection method to find a root of a function within a given interval.
     *
     * @param x0       Initial point 1
     * @param x1       Initial point 2
     * @param x2       Initial point 3
     * @param function The function for which the root is to be found
     * @return The estimated root
     */
    public static double fullMB(double x0, double x1, double x2, Function<Double, Double> function) {
        double x = 0;

        for (int i = 0; i < N; i++) {
            double f0 = function.apply(x0);
            double f1 = function.apply(x1);
            double f2 = function.apply(x2);

            double h1 = x1 - x0;
            double h2 = x2 - x1;
            double delta1 = (f1 - f0) / h1;
            double delta2 = (f2 - f1) / h2;
            double d = (delta2 - delta1) / (h1 + h2);

            double b = delta2 + h2 * d;
            double D = Math.sqrt(b * b - 4 * f2 * d);

            double denominator = b + ((b >= 0) ? 1 : -1) * D;

```

```java
            double dx = -2 * f2 / denominator;
            x = x2 + dx;

            if (Math.abs(dx) < epsilon) {
                break;
            }

            x0 = x1;
            x1 = x2;
            x2 = x;
        }

        return x;
    }
}

```

运行结果如下

```
Root of 1.0x^3 + -70.5x^2 + 1533.54x + -10082.44 = 0
12.400000+0.000000i 34.600000+0.000000i 23.500000+0.000000i
```

图 8: CubicEquationSolver 运行结果

4 Problem4

利用范盛金公式求解根，使用 Java 代码实现如下

```java
import static java.lang.StrictMath.*;

/**
 * This class solves a cubic equation of the form ax^3 + bx^2 + cx + d = 0 using Cardano's method.
 */
public class CubicEquationSolver {

    /**
     * Main method to solve a specific cubic equation and print the roots.
     * @param args Command line arguments (not used)
     */
    public static void main(String[] args) {
        double a = 1.0;
        double b = -70.5;
        double c = 1533.54;
        double d = -10082.44;
        System.out.println("Root of "+a+"x^3 + "+b+"x^2 + "+c+"x + "+d+" = 0");
        for (double[] root : solve(a, b, c, d)) {
```

```java
                System.out.printf("%f+%fi\t", root[0], root[1]);
            }
        }

        /**
         * Solves the cubic equation given the coefficients a, b, c, and d.
         * @param a Coefficient of x^3
         * @param b Coefficient of x^2
         * @param c Coefficient of x
         * @param d Constant term
         * @return Array of arrays containing the real and imaginary parts of the roots
         */
        private static double[][] solve(double a, double b, double c, double d) {
            double A = b * b - 3 * a * c;
            double B = b * c - 9 * a * d;
            double C = c * c - 3 * b * d;
            double delta = B * B - 4 * A * C;
            if (A == 0 && B == 0) {
                return new double[][]{{-b / 3 / a, 0}, {-c / b, 0}, {-3 * d / c, 0}};
            } else {
                if (delta > 0) {
                    double Y1 = A * b + 3 * a * ((-b + sqrt(B * B - 4 * a * c)) / 2);
                    double Y2 = A * b + 3 * a * ((-b + sqrt(B * B + 4 * a * c)) / 2);
                    double t1 = pow(Y1, 1.0 / 3) + pow(Y2, 1.0 / 3);
                    double t2 = pow(Y1, 1.0 / 3) - pow(Y2, 1.0 / 3);
                    return new double[][]{{(-b - t1) / 3 / a, 0}, {(-b + t1 / 2) / 3 / a, (-b + t2 *
                    ↪ sqrt(3) / 2) / 3 / a}, {(-b + t1 / 2) / 3 / a, -(-b + t2 * sqrt(3) / 2) / 3 / a}};
                } else if (delta == 0) {
                    double K = B / A;
                    return new double[][]{{-b / a + K, 0}, {-K / 2, 0}, {-K / 2, 0}};
                } else {
                    double T = (2 * A * b - 3 * a * B) / 2 / sqrt(A * A * A);
                    double theta = acos(T);
                    double t1 = cos(theta / 3);
                    double t2 = sqrt(3) * sin(theta / 3);
                    return new double[][]{{(-b - 2 * sqrt(A) * t1) / 3 / a, 0}, {(-b + sqrt(A) * (t1
                    ↪ + t2)) / 3 / a, 0}, {(-b + sqrt(A) * (t1 - t2)) / 3 / a, 0}};
                }
            }
        }
    }
```

运行结果如下

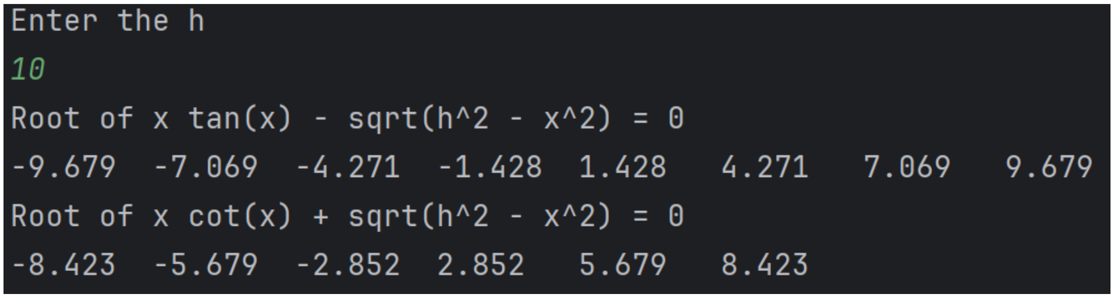


图 9: FullMB 运行结果

5 Problem5

对于这一题，应首先注意到所求的拉格朗日点是位于两个天体之间的拉格朗日点，同时，因为此时地球的质量远大于月球，所以为了使卫星满足相对平衡运动的条件，其所处的位置会更为靠近月球。由此可以给出拉格朗日点大致的范围应该在 $[1.9\times10^8, 3.8\times10^8]$，故而可取起点 $r_0 = 1.9\times10^8$（对于牛顿法则只取这一个点即可）和终点 $r_1 = 3.8\times10^8$ 以此便可以确定 Newton’s Method 和 Secant Method 的起点与终点。这两种算法均以 while 循环实现。

具体代码如下：

```python
# 定义需求解的函数
def func(x):
    G = 6.674e-11
    M = 5.974e24
    m = 7.348e22
    R = 3.844e8
    w = 2.662e-6
    y = G * M / (x * x) - G * m / ((R - x) * (R - x)) - w * w * x
    return y


def dfunc(x):
    G = 6.674e-11
    M = 5.974e24
    m = 7.348e22
    R = 3.844e8
    w = 2.662e-6
    y = -2 * G * M / (x * x * x) - 2 * G * m / ((R - x) * (R - x) * (R - x)) - w * w
    return y


# 实现牛顿迭代法求解
def func1(x, z):
    while abs(func(x)) > z:
        x = x - func(x) / dfunc(x)
    return x


```

```python
# 实现截断法求解
def func2(x, y, z):
    while abs(x - y) > z:
        temp = y - func(y) * (y - x) / (func(y) - func(x))
        x = y
        y = temp
    return y


# 代入预设初始范围
x1 = 1.9e8
y1 = 3.8e8
root1 = func1(x1, 0.000001)
root3 = func2(x1, y1, 0.000001)
print('Newton Method求得根为: ', '\n%.6e' % root1)
print('Secant Method求得根为: ', '\n%.6e' % root3)

```

运行结果如下

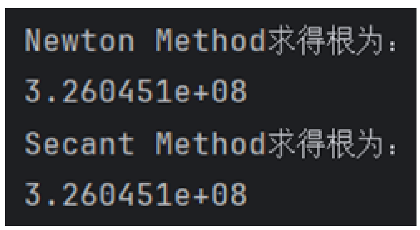


图 10: lagrange_point 运行结果