

# 第二章作业

Anonymity 3220100000\*

\* College of Control Science and Engineering, Robotics Engineering

July 8, 2024

## 1 问题描述

工程师经常需要计算抛物的轨迹。假设坐标系如图所示，左边的球向右抛出后的轨迹可由如下公式计算：

$$y = (\tan \theta_0) x - \frac{g}{2v_0^2 \cos^2 \theta_0} x^2 + y_0$$

假设：

球抛出的初始速度为  $v_0 = 30$  米/秒；

抛出点的高度为  $y_0 = 1.8$  米；

重力加速度为  $g = 9.81$  米/秒<sup>2</sup>；

若球到达离抛出点水平距离  $x = 90$  米处时，球的高度为  $y = 1.0$  米，

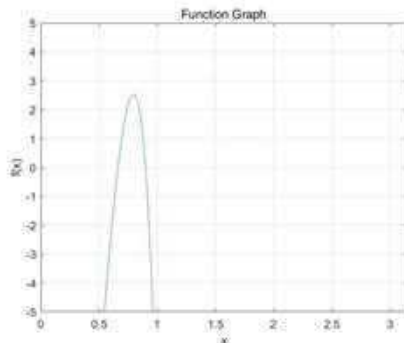
请：计算球的抛出角  $\theta_0$ 。要求分别用二分法、试位法、不动点迭代、*Newton - Raphson* 法和割线法求解并对各种方法进行比较。（提示：存在两个根）

## 2 思路分析

将所给数据代入公式得方程：

$$f(x) = 90 \tan \theta_0 - \frac{9.81 \times 9}{2 \cos^2 \theta_0} + 0.8 = 0$$

由于该方程周期为  $\pi$ ，因此在 matlab 中仅绘制其在  $0$  到  $\pi$  的函数图像（图中未显示的部分过小，已判断无根）



## 2.1 二分法

首先根据绘制的函数图像草图确定有根区间 (0.6,0.7) 和 (0.8,1.0) 将区间二等分, 通过判断  $f(x)$  的符号, 逐步将有根区间缩小, 直至有根区间满足计算精度, 便可求出满足精度要求的近似根。

## 2.2 试位法

对于原方程  $f(x) = 90 \tan \theta_0 - \frac{9.81 \times 9}{2 \cos^2 \theta_0} + 0.8 = 0$  直接使用试位法迭代次数较多, 因此采取通分后的函数  $f(x) = \frac{180 \tan \theta_0 \cos^2 \theta_0 - 88.29}{2 \cos^2 \theta_0} - 0.8$  进行计算可以极大地减少迭代次数。

与二分法类似, 分别将根的预估范围 (0.6,0.7) 和 (0.8,1.0) 代入求根, 用连接而成的直线作为新的上下界。

## 2.3 不动点迭代法

通过重组方程, 构建新函数  $g(x)$ , 从而通过不动点迭代法, 求解方程的根。由于原方程中含有  $\theta_0$  的项有两个, 分别是  $\tan \theta_0$  和  $\cos \theta_0$ , 因此可以构建两个迭代方程  $g(x)$ , 分别是  $g(x) = \arctan\left(\frac{9.81 \times 9}{180 \cos^2(x)} - \frac{0.8}{90}\right)$  和  $g(x) = \arccos\left(\sqrt{\frac{9.81 \times 9}{90} \cdot \tan(x) - \frac{0.8}{90}}\right)$  并通过选取多个初始点进行方程根的估算。

## 2.4 Newton - Raphson 法

本方法通过在给定初始值  $x_1$  处对原函数进行泰勒展开, 用切线近似代替曲线, 求切线与  $x$  轴的交点。由于近似知道根的值, 因此给定初始值为 0.65 和 0.9。

## 2.5 割线法

割线法与 Newton - Raphson 法类似, 只不过用割线代替了切线。又类似于试位法, 但试位法是按照符号取代, 割线法按照迭代顺序取代。同样的, 由于近似知道根的值, 初始给定两根的猜测范围是 (0.6,0.7) 和 (0.8,1.0)。

# 3 运行结果与分析

先采用 matlab 中自带的函数得到准确根, 认定位实际值:

## 命令行窗口

% 显示结果

```
fprintf('第一个根: %.6f\n', root1);
```

```
fprintf('第二个根: %.6f\n', root2);
```

```
第一个根: 0.662509
```

```
第二个根: 0.899398
```

*fx* >>

### 3.1 二分法

运行结果如下，可以较为准确的给出两个标准根，与实际值完全相同：

## 命令行窗口

```
end
```

```
end
```

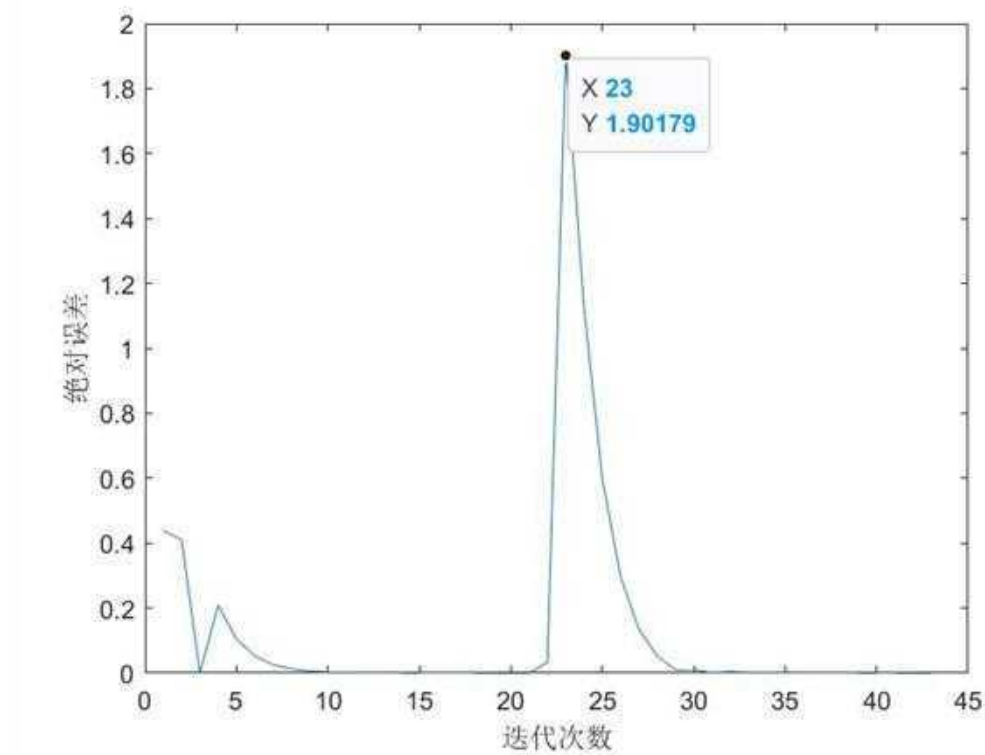
% 显示结果

```
fprintf('The roots are: %0.6f and %0.6f\n', root1, root2);
```

```
The roots are: 0.662509 and 0.899398
```

*fx* >> |

可视化误差随迭代次数的变化曲线为：



可以看到二分法计算存在一定的不稳定性,尤其前提是要清楚的知道根的分布。比如本题,如果将初始区间设置在  $(0, \pi)$ , 就很难得到最终正确的结果。

### 3.2 试位法

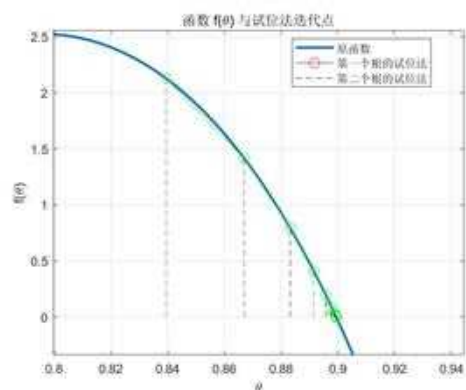
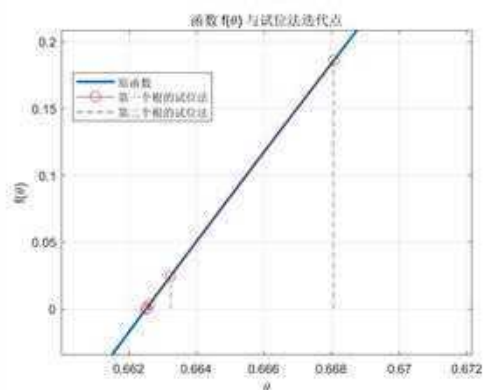
运行结果如下,可以较为准确的给出两个标准根,与实际值完全相同:

```

命令行窗口
fprintf(' 第二个根的值: %.6f\n', root2);
fprintf(' 第二个根的迭代次数: %d\n', iter2);
第一个根的值: 0.662509
第一个根的迭代次数: 7
第二个根的值: 0.899398
第二个根的迭代次数: 21
fx >>

```

可视化误差随迭代次数的变化曲线为:



试位法与二分法极其相似，只是通过连成的直线减小了计算的次数，但从复杂度角度考虑并没有变化。

对于本题的目标函数来说，在根的附近导数的绝对值过大，因此试位法与二分法的区别不大，甚至在不通分的情况下试位法会更加麻烦。

### 3.3 不动点迭代法

对于函数  $g(x) = \arctan\left(\frac{9.81 \times 9}{180 \cos^2(x)} - \frac{0.8}{90}\right)$ ，可以得到一个根 0.662509，与实际值完全相同。

```

命令行窗口
else
    disp('没有找到满足条件的根。');
end
满足条件的根为:
0.662509
0.662509
fx >>

```

对于函数  $g(x) = \arccos\left(\sqrt{\frac{9.81 \times 9}{90} \cdot \tan(x)} - \frac{0.8}{90}\right)$ ，无法得到任何一个根，即对于所有初始值选取都会发散。

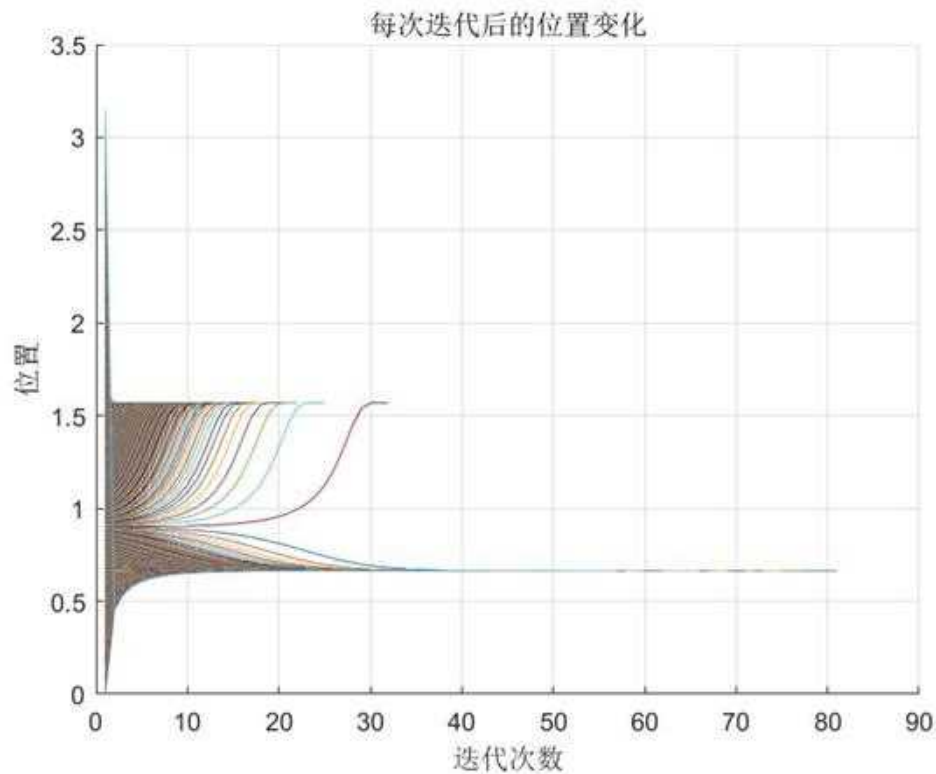
## 命令行窗口

```
disp(num2str(roots(1), '%.6f'));  
disp(num2str(roots(2), '%.6f'));  
else  
    disp('没有找到满足条件的根。');  
end
```

没有找到满足条件的根。

*fx* >>

因此对于不动点迭代法求解原方程只能得到一个根，存在一定的缺陷。对于能求解的一个根，进行迭代过程的可视化分析：



可以发现迭代次数非常多，运算较为复杂。

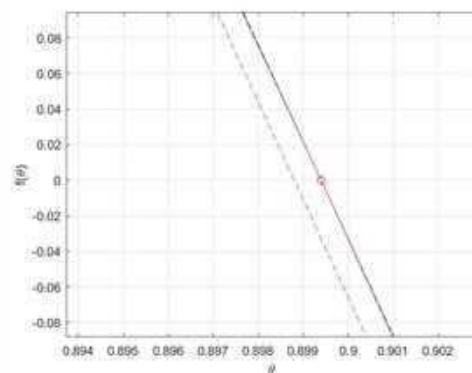
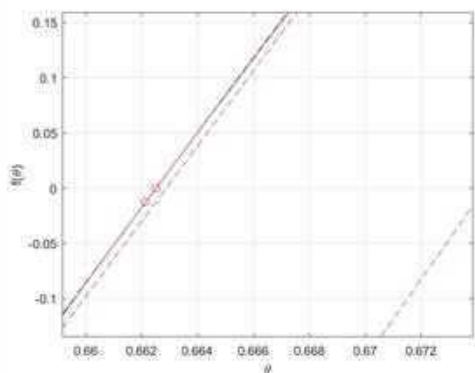
结合图像分析可知原函数的导数斜率过大，难以收敛，因此不适合使用不动点迭代法。

### 3.4 Newton – Raphson 法

对于 Newton – Raphson 法运行结果可以看到，收敛速度非常快，迭代次数很少，可以快速的算出答案，且计算结果与真实值完全相同。

```
命令行窗口
end
end
第1个根: 0.662509
第1个根迭代次数: 4
第2个根: 0.899398
第2个根迭代次数: 3
fx >>
```

对运算过程进行可视化分析有：



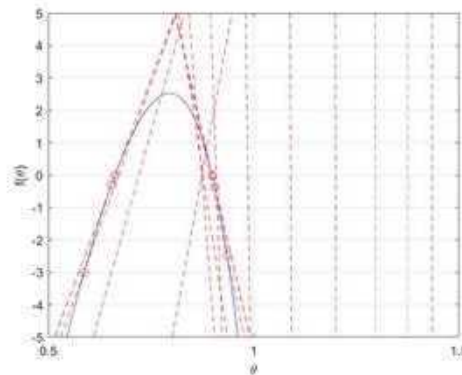
虽然在测试程序中初始值的选取与实际值较为接近，但是对于本题中的函数，其他初始值也同样可以用很少的迭代次数得到准确的结果。

比如我们将初始值修改为 0.3 和 1.5 后，仍可以快速的得到运行结果：

#### 命令行窗口

```
end  
end  
第1个根: 0.662509  
第1个根迭代次数: 6  
第2个根: 0.899398  
第2个根迭代次数: 13
```

*fx* >>



因此，对于计算目标函数的根，*Newton - Raphson* 法是一种较为适合的方法。

### 3.5 割线法

割线法的运行结果可以看到，迭代次数比 *Newton - Raphson* 法还要少，且尤其对于本题中的复杂函数，计算割线要比求导容易的多。将结果与真实值相比完全相同。

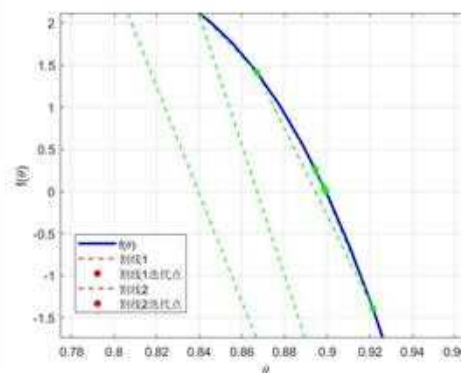
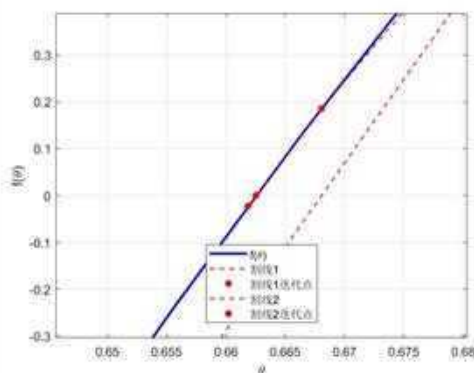
#### 命令行窗口

```
grid on;  
hold off;  
第一个根: 0.662509  
第一个根的迭代次数: 4  
第二个根: 0.899398  
第二个根的迭代次数: 7
```

*fx* >>

对运算过程进行可视化分析有：





可以发现，尤其对于本题函数这种趋近于直线的函数，割线法的求解相当快速，能够快速的收敛到根的范围。且与 *Newton - Raphson* 法相比不用计算复杂的导数，更为简便。

### 3.6 小结

综合分析上述五种求解非线性方程根的方法，二分法和试位法思路简单清晰，对于大多数函数较为适用，但本题的函数斜率过大，较难计算。而不动点迭代法可以快速得到结果，然而初始值的选取与  $g(x)$  函数的构造存在一定的不确定性，且对于本题是无法得到全部的两个根。而 *Newton - Raphson* 法与割线法的思路较复杂，但对于本题来说比较适合，可以以较少的迭代次数得到全部的两个根，其中割线法较 *Newton - Raphson* 法更为简单快速。

虽然五种方法各有优缺点，但是他们计算出的值与实际值在精度范围内均完全相同，可以验证这五种方法的准确性与可靠性。同时，在进行精确的计算时，我们都是在对于整体函数有了一个初步的印象后进行的，即估计出了根的所在范围后进行。

因此，尤其是后三种方法，更适用于准确估计方程根的情况，二分法与试位法则可以用于粗略估计根的范围。

## 4 实验心得

本次实验我采用了五种方法计算了一个非线性方程的根，且都得到了较为精确的结果。通过本次实验，我熟悉了求方程根各种方法的具体思路，也提高了自己的 matlab 编程能力。

对于五种方法，我均对结果进行了可视化分析，以直观的图像展现出迭代的过程，也可以从中看出不同方法之所以迭代次数相差较大的原因。这有助于我以后对于任意的非线性方程求根的精确值时，可以根据其图像快速的判断出其适合的求解方式。数形结合始终是学习数学重要的一环。

## 5 源代码

### 5.1 绘制原函数图像

```
1 % 定义函数
2 f = @(x) 90*tan(x) - (9.81*9)/(2*cos(x)^2) + 0.8;
3
```

```

4 % 绘制函数图像
5 fplot(f, [-pi/2, pi/2]);
6 grid on;
7 xlabel('x');
8 ylabel('f(x)');
9 title('Function Graph');
10
11 % 设置横坐标范围为 -6 到 6
12 xlim([0, pi]);
13 ylim([-5, 5]);

```

## 5.2 得到准确的根

```

1 % 定义函数
2 f = @(theta) 90 * tan(theta) - (9.81 * 9) / (2 * cos(theta)^2) + 0.8;
3
4 % 求解方程
5 root1 = fzero(f, 0.65);
6 root2 = fzero(f, 0.9);
7
8 % 显示结果
9 fprintf('第一个根: %.6f\n', root1);
10 fprintf('第二个根: %.6f\n', root2);

```

## 5.3 二分法

```

1 % 定义方程
2 f = @(theta) 90 * tan(theta) - (9.81 * 9) / (2 * cos(theta)^2) + 0.8;
3
4 % 初始化根的计数器
5 root_count = 0;
6
7 % 存储根的数组
8 roots = zeros(2, 1);
9
10 % 设置精度
11 tolerance = 1e-6;
12
13 % 开始二分法 - 寻找第一个根在 [0.6, 0.7] 之间
14 lower_bound = 0.6;
15 upper_bound = 0.7;

```

```

16 errors = []; % 存储绝对误差
17 iterations = 0; % 迭代次数
18 while root_count < 1
19     iterations = iterations + 1;
20     mid_point = (lower_bound + upper_bound) / 2;
21     error = abs(f(mid_point));
22     errors(iterations) = error;
23     if error < tolerance
24         % 找到一个根
25         root_count = root_count + 1;
26         roots(root_count) = mid_point;
27         % 重新设置搜索范围以寻找下一个根
28         lower_bound = 0.8;
29         upper_bound = 1.0;
30     elseif f(mid_point) * f(lower_bound) < 0
31         % 根在左半边
32         upper_bound = mid_point;
33     elseif f(mid_point) * f(upper_bound) < 0
34         % 根在右半边
35         lower_bound = mid_point;
36     end
37 end
38
39 % 开始二分法 - 寻找第二个根在[0.8, 1.0]之间
40 while root_count < 2
41     iterations = iterations + 1;
42     mid_point = (lower_bound + upper_bound) / 2;
43     error = abs(f(mid_point));
44     errors(iterations) = error;
45     if error < tolerance
46         % 找到第二个根
47         root_count = root_count + 1;
48         roots(root_count) = mid_point;
49     elseif f(mid_point) * f(lower_bound) < 0
50         % 根在左半边
51         upper_bound = mid_point;
52     elseif f(mid_point) * f(upper_bound) < 0
53         % 根在右半边
54         lower_bound = mid_point;
55     end
56 end

```

```

57
58 % 显示结果
59 fprintf('The roots are: %0.6f and %0.6f\n', roots(1), roots(2));
60
61 % 绘制绝对误差随迭代次数的变化
62 figure;
63 plot(1:iterations, errors);
64 xlabel('迭代次数');
65 ylabel('绝对误差');

```

## 5.4 试位法

```

1 % 定义函数
2 f = @(theta) 90 * tan(theta) - (9.81 * 9) ./ (2 * cos(theta).^2) +
    0.8;
3
4 % 绘制原函数的曲线
5 theta = linspace(0.1, 1.4, 1000); % 生成一些 theta 值
6 figure;
7 plot(theta, f(theta), 'LineWidth', 2);
8 hold on;
9
10 % 第一个根的求解
11 a1 = 0.6; % 第一个根的初始区间左端点
12 b1 = 0.7; % 第一个根的初始区间右端点
13 tolerance = 1e-6; % 容许误差
14 max_iterations = 1000; % 最大迭代次数
15
16 % 初始化迭代变量
17 iter1 = 0;
18 root1 = 0;
19 roots1 = []; % 存储每次迭代后的根
20
21 % 使用试位法求解第一个根
22 while iter1 < max_iterations
23     iter1 = iter1 + 1;
24     root1 = (a1 * f(b1) - b1 * f(a1)) / (f(b1) - f(a1));
25     roots1 = [roots1, root1]; % 将根添加到列表中
26
27     if abs(f(root1)) < tolerance
28         break;

```

```

29     elseif f(a1) * f(root1) < 0
30         b1 = root1;
31     else
32         a1 = root1;
33     end
34 end
35
36 % 绘制试位法的曲线
37 plot(roots1, f(roots1), 'ro-', 'MarkerSize', 8);
38
39 % 在 x 轴上标记每次迭代的点
40 for i = 1:length(roots1)
41     plot([roots1(i), roots1(i)], [0, f(roots1(i))], 'k--');
42 end
43
44 % 第二个根的求解
45 a2 = 0.8; % 第二个根的初始区间左端点
46 b2 = 1.0; % 第二个根的初始区间右端点
47
48 % 初始化迭代变量
49 iter2 = 0;
50 root2 = 0;
51 roots2 = []; % 存储每次迭代后的根
52
53 % 使用试位法求解第二个根
54 while iter2 < max_iterations
55     iter2 = iter2 + 1;
56     root2 = (a2 * f(b2) - b2 * f(a2)) / (f(b2) - f(a2));
57     roots2 = [roots2, root2]; % 将根添加到列表中
58
59     if abs(f(root2)) < tolerance
60         break;
61     elseif f(a2) * f(root2) < 0
62         b2 = root2;
63     else
64         a2 = root2;
65     end
66 end
67
68 % 绘制试位法的曲线
69 plot(roots2, f(roots2), 'go-', 'MarkerSize', 8);

```

```

70
71 % 在 x 轴上标记每次迭代的点
72 for i = 1:length(roots2)
73     plot([roots2(i), roots2(i)], [0, f(roots2(i))], 'k--');
74 end
75
76 % 显示结果
77 fprintf('第一个根的价值为: %.6f\n', root1);
78 fprintf('第一个根的迭代次数: %d\n', iter1);
79 fprintf('第二个根的价值为: %.6f\n', root2);
80 fprintf('第二个根的迭代次数: %d\n', iter2);
81
82 % 设置图例
83 legend('原函数', '第一个根的试位法', '第二个根的试位法', 'Location', 'best');
84 xlabel('\theta');
85 ylabel('f(\theta)');
86 title('函数 f(\theta) 与试位法迭代点');
87
88
89 % 显示图形
90 grid on;
91 ylim([-5, 5]);
92 hold off;

```

## 5.5 不动点迭代 tan 函数

```

1 % 定义函数 f(x)
2 f = @(x) 90*tan(x) - (9.81*9)/(2*cos(x)^2) + 0.8;
3
4 % 定义不动点迭代函数 g(x)
5 g = @(x) atan((9.81*9)/(180*cos(x)^2) - 0.8/90);
6
7 % 选择多组初始近似值
8 x0_values = [linspace(0, pi/2, 500), linspace(pi/2, pi, 500)]; % 在区
    间 [0, pi/2] 和 [pi/2, pi] 上均匀选择 1000 个初始值
9
10 % 设定迭代次数上限和误差容限
11 max_iter = 1000;
12 tolerance = 1e-8;
13

```

```

14 % 存储满足条件的根
15 roots = [];
16
17 % 迭代并比较结果
18 for i = 1:length(x0_values)
19     x0 = x0_values(i);
20     x = x0;
21     for iter = 1:max_iter
22         x_next = g(x);
23         if abs(x_next - x) < tolerance
24             % 满足收敛条件
25             root = x_next;
26             if root > 0 && root < pi
27                 % 检查是否在 [0, pi] 区间内
28                 roots = [roots, root];
29             end
30             break;
31         end
32         x = x_next;
33     end
34 end
35
36 % 输出满足条件的根
37 if length(roots) >= 2
38     disp('满足条件的根为: ');
39     disp(num2str(roots(1), '%.6f'));
40     disp(num2str(roots(2), '%.6f'));
41 else
42     disp('没有找到满足条件的根。');
43 end

```

## 5.6 不动点迭代 cos 函数

```

1 % 定义函数 f(x)
2 f = @(x) 90*tan(x) - (9.81*9)/(2*cos(x)^2) + 0.8;
3
4 % 定义不动点迭代函数 g(x)
5 g = @(x) acos(sqrt((9.81*9)/90 * tan(x) - 0.8/90));
6
7 % 选择多组初始近似值
8 x0_values = [linspace(0, pi/2, 500), linspace(pi/2, pi, 500)]; % 在区

```

```

    间  $[0, \pi/2]$  和  $[\pi/2, \pi]$  上均匀选择 1000 个初始值
9
10 % 设定迭代次数上限和误差容限
11 max_iter = 1000;
12 tolerance = 1e-8;
13
14 % 存储满足条件的根
15 roots = [];
16
17 % 迭代并比较结果
18 for i = 1:length(x0_values)
19     x0 = x0_values(i);
20     x = x0;
21     for iter = 1:max_iter
22         x_next = g(x);
23         if abs(x_next - x) < tolerance
24             % 满足收敛条件
25             root = x_next;
26             if root > 0 && root < pi
27                 % 检查是否在  $[0, \pi]$  区间内
28                 roots = [roots, root];
29             end
30             break;
31         end
32         x = x_next;
33     end
34 end
35
36 % 输出满足条件的根
37 if length(roots) >= 2
38     disp('满足条件的根为: ');
39     disp(num2str(roots(1), '%.6f'));
40     disp(num2str(roots(2), '%.6f'));
41 else
42     disp('没有找到满足条件的根。');
43 end

```

## 5.7 不动点迭代可视化结果

```

1 % 定义函数 f(x)
2 f = @(x) 90*tan(x) - (9.81*9)/(2*cos(x)^2) + 0.8;

```



```

3
4 % 定义不动点迭代函数 g(x)
5 g = @(x) atan((9.81*9)/(180*cos(x)^2) - 0.8/90);
6
7 % 选择多组初始近似值
8 x0_values = [linspace(0, pi/2, 500), linspace(pi/2, pi, 500)]; % 在区
    间 [0, pi/2] 和 [pi/2, pi] 上均匀选择 1000 个初始值
9
10 % 设定迭代次数上限和误差容限
11 max_iter = 1000;
12 tolerance = 1e-8;
13
14 % 存储满足条件的根
15 roots = [];
16
17 % 存储每次迭代后的位置变化
18 positions = cell(1, length(x0_values));
19
20 % 迭代并比较结果
21 for i = 1:length(x0_values)
22     x0 = x0_values(i);
23     x = x0;
24     position_history = [x0]; % 存储每次迭代后的位置
25     for iter = 1:max_iter
26         x_next = g(x);
27         if abs(x_next - x) < tolerance
28             % 满足收敛条件
29             root = x_next;
30             if root > 0 && root < pi
31                 % 检查是否在 [0, pi] 区间内
32                 roots = [roots, root];
33             end
34             break;
35         end
36         x = x_next;
37         position_history = [position_history, x]; % 更新位置历史记录
38     end
39     positions{i} = position_history; % 存储位置变化历史记录
40 end
41
42 % 输出满足条件的根

```

```

43 if length(roots) >= 2
44     disp('满足条件的根为: ');
45     disp(num2str(roots(1), '%.6f'));
46     disp(num2str(roots(2), '%.6f'));
47 else
48     disp('没有找到满足条件的根。');
49 end
50
51 % 绘制每次迭代后的位置变化
52 figure;
53 hold on;
54 for i = 1:length(positions)
55     plot(positions{i});
56 end
57 xlabel('迭代次数');
58 ylabel('位置');
59 title('每次迭代后的位置变化');
60 grid on;
61 hold off;

```

## 5.8 Newton – Raphson 法

```

1 % 定义函数
2 f = @(theta) 90 * tan(theta) - (9.81 * 9) ./ (2 * cos(theta).^2) +
    0.8;
3
4 % 初值和容许误差
5 theta0_1 = 0.65;
6 theta0_2 = 0.9;
7 tolerance = 1e-8;
8 maxIterations = 1000;
9
10 % 创建画布
11 figure;
12
13 % 定义 theta 范围
14 theta_range = linspace(0, pi/2, 1000);
15
16 % 绘制原函数图像
17 f_values = f(theta_range);
18 plot(theta_range, f_values, 'b');

```

```

19 xlabel( '\theta ');
20 ylabel( 'f(\theta) ');
21 ylim([-5 5]);
22 grid on;
23 hold on;
24
25 % 循环求解两个根
26 for root_num = 1:2
27     % 初值选择
28     if root_num == 1
29         theta = theta0_1;
30     else
31         theta = theta0_2;
32     end
33
34     iteration = 0;
35     error = Inf;
36
37     % 迭代求解
38     while abs(error) > tolerance && iteration < maxIterations
39         f_value = f(theta);
40         f_derivative = (f(theta + 1e-6) - f(theta)) / 1e-6; % 导数的近
                        似值
41         delta_theta = -f_value / f_derivative; % 根据牛顿-拉弗森方法计
                        算增量
42         theta = theta + delta_theta; % 更新 theta
43         error = delta_theta; % 误差定义为增量
44         iteration = iteration + 1;
45
46         % 计算切线斜率
47         tangent_slope = f_derivative;
48
49         % 计算切线截距
50         tangent_intercept = f_value - tangent_slope * theta;
51
52         % 绘制切线
53         plot(theta_range, tangent_slope * theta_range +
                tangent_intercept, 'r--');
54         plot(theta, f(theta), 'ro');
55         drawnow; % 实时更新图像
56     end

```

```

57
58 % 输出结果或提示未收敛
59 if iteration == maxIterations
60     disp(['第', num2str(root_num), '个根: 达到最大迭代次数而未收敛
        ']);
61 else
62     root = theta;
63     fprintf('第%d个根: %.6f\n', root_num, root);
64     fprintf('第%d个根迭代次数: %d\n', root_num, iteration);
65 end
66 end

```

## 5.9 割线法

```

1 % 定义方程函数
2 f = @(theta) 90 * tan(theta) - (9.81 * 9) ./ (2 * cos(theta).^2) +
    0.8;
3
4 % 第一个根的初始猜测值
5 x0_1 = 0.6;
6 x1_1 = 0.7;
7
8 % 第二个根的初始猜测值
9 x0_2 = 0.8;
10 x1_2 = 1.0;
11
12 % 容差和最大迭代次数
13 tol = 1e-6;
14 max_iter = 1000;
15
16 % 初始化变量
17 iter = 0;
18
19 % 绘制方程函数曲线
20 theta_range = linspace(0.1, 1.2, 100); % 定义绘图范围
21 f_values = f(theta_range);
22 plot(theta_range, f_values, 'b-', 'LineWidth', 2);
23 hold on;
24
25 % 设置纵坐标限制
26 ylim([-5, 5]);

```

```

27
28 % 主循环, 执行割线法来寻找第一个根
29 while iter < max_iter
30     % 绘制割线
31     plot([x0_1, x1_1], [f(x0_1), f(x1_1)], 'r--', 'LineWidth', 1);
32
33     % 使用割线法公式计算下一个近似值
34     x_next = x1_1 - (f(x1_1) * (x1_1 - x0_1)) / (f(x1_1) - f(x0_1));
35
36     % 绘制割线法迭代点
37     plot(x_next, f(x_next), 'ro', 'MarkerSize', 5, 'MarkerFaceColor',
          'r');
38
39     % 检查容差是否满足
40     if abs(x_next - x1_1) < tol
41         root_1 = x_next;
42         fprintf('第一个根: %f\n', root_1);
43         fprintf('第一个根的迭代次数: %d\n', iter);
44         break;
45     end
46
47     % 更新下一次迭代的值
48     x0_1 = x1_1;
49     x1_1 = x_next;
50     iter = iter + 1;
51 end
52
53 % 初始化迭代计数器
54 iter = 0;
55
56 % 主循环, 执行割线法来寻找第二个根
57 while iter < max_iter
58     % 绘制割线
59     plot([x0_2, x1_2], [f(x0_2), f(x1_2)], 'g--', 'LineWidth', 1);
60
61     % 使用割线法公式计算下一个近似值
62     x_next = x1_2 - (f(x1_2) * (x1_2 - x0_2)) / (f(x1_2) - f(x0_2));
63
64     % 绘制割线法迭代点
65     plot(x_next, f(x_next), 'go', 'MarkerSize', 5, 'MarkerFaceColor',
          'g');

```

```

66
67 % 检查容差是否满足
68 if abs(x_next - x1_2) < tol
69     root_2 = x_next;
70     fprintf('第二个根: %f\n', root_2);
71     fprintf('第二个根的迭代次数: %d\n', iter);
72     break;
73 end
74
75 % 更新下一次迭代的值
76 x0_2 = x1_2;
77 x1_2 = x_next;
78 iter = iter + 1;
79 end
80
81 % 添加标题和标签
82 xlabel('\theta');
83 ylabel('f(\theta)');
84 legend('f(\theta)', '割线1', '割线1迭代点', '割线2', '割线2迭代点', '
    Location', 'best');
85 grid on;
86 hold off;

```