

# 计算物理作业 2

杨远青 22300190015

2024 年 9 月 21 日

今天原本应该是鏖战 *Griffiths* 的猫与蔡先生的日子，可惜平板没电。

## 1 题目 1: 三次方程根的求解与精确化

### 1.1 题目描述

Sketch the function  $x^3 - 5x + 3 = 0$

(i) Determine the two positive roots to 4 decimal places using the bisection method.

**Note:** You first need to bracket each of the roots.

(ii) Take the two roots that you found in the previous question (accurate to 4 decimal places) and “polish them up” to 14 decimal places using the Newton-Raphson method.

(iii) Determine the two positive roots to 14 decimal places using the hybrid method.

### 1.2 程序描述

标准的求根问题，顺便练习一下 C++。先使用 Mathematica<sup>®</sup> 画出函数草图如下：

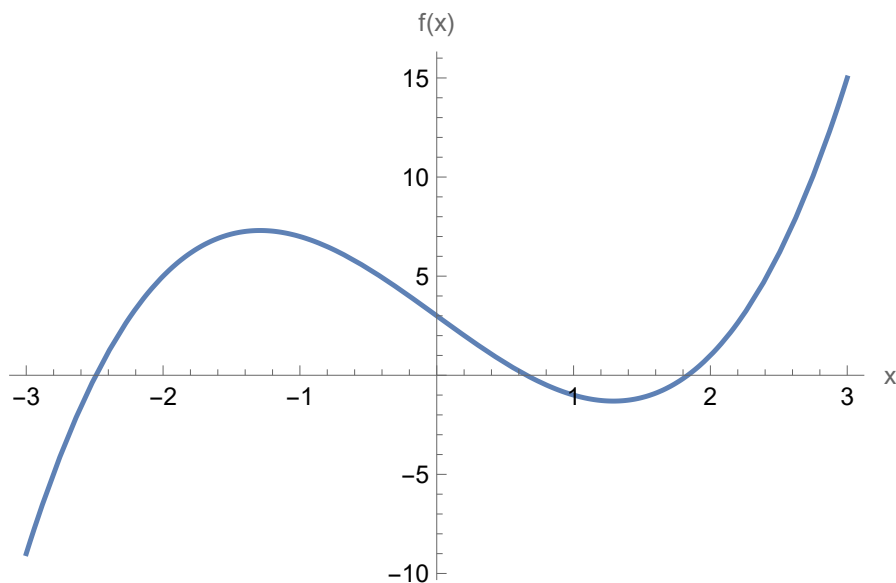


图 1: Plot of  $x^3 - 5x + 3 = 0$ .

发现有两个正根，分别在  $[0, 1]$  和  $[1, 2]$  之间，有点好奇二分法的迭代，便先用 Python 写一下二分法的动态过程，见 `./Codes/Problem 1/bisection_visual.py`，使用 `python -u bisection_visual.py` 可以交互运行（需要安装 `matplotlib, numpy` 库）。运行后有三个选项，分别是前两个正根的查找与自定义区间查找，可以自行选择。

`./Codes/Problem 1/`中还有 C++ 实现的二分法、牛顿法、混合法、Brent 法与 Ridder 法，算法实现集成在了 `methods.cpp` 中，`main.cpp` 负责封装与交互，`plotting.cpp` 用 ASCII 绘制了草图，`functions.cpp` 里面存储了本题函数与导数，拎出来是为了便于灵活测试其它函数，`utils.cpp` 里面有两个工具函数，分别负责按照题目三小问的顺序输出结果与对比测试各类方法在三个根上的表现。使用 `g++ *.cpp -o main` 编译，`./main` 运行（也有已经编译好的 `main.exe`），按照提示可以选择各类方法或者一起比较，也可以自定义查找区间与容差等参数。

原本对求根问题的兴趣不太大，至少没有 24 点那么有趣。在翻阅徐老师给的那本 *Numerical Recipe* 的时候，发现虽然出了第三版，但出版商给每个章节都加弹窗广告。是可忍孰不可忍，我对金力发誓，绝不向校图书馆荐购它。但里面提到的当下最常用的 Brent 法与 Ridder 法还是挺有意思的，在 `scipy` 里面也有它们的 C 语言实现，不过感觉写得没有 GNU Scientific Library 里面的清晰易懂，照猫画虎，在 GPT 的大力支持下，我用 C++ 实现了一遍，最繁琐的处理用户输入输出任务主要是它干的，想到了很多我没想到的细节，出色的前端工程师 (bushi)!

Ridder 法源自割线法和虚位法一脉。这一派的祖制是假设局部近似线性，因而在处理导数变化较大的地方表现不佳，尤其是割线法，虽号称有着黄金比率 1.618 的超线性收敛阶数，但在许多场合还不如二分法。新生代 Ridder 法就是变换一下  $h(x) = f(x)e^{ax}$ ，用一个指数因子来模拟潜在的非线性行为

$$e^{a(x_1-x_0)} = \frac{f(x_1) - \text{sign}[f(x_0)]\sqrt{f(x_1)^2 - f(x_0)f(x_2)}}{f(x_2)}.$$

在此基础上再使用虚位法，考虑到这个指数函数的查找提供的二阶加速，以及本身多一次的计算，其理论收敛指数为  $\sqrt{2} \approx 1.414 < 1.618$ ，but who cares about theory? 实际效果还是不错的，详情参见 5。

Brent 法的核心思想继承自 Dekker 法，后者与我们题目第三问提示的 Hybrid 法有些类似，但在局部加速收敛的过程中，用的不是基于导数的 Newton 法，而是用割线法，也即在小区间内倾向于使用单点迭代，但在迭代点越过区间范围时使用二分法补救，非常自然且优美的思路。Brent 主要是在 Dekker 法的基础上加入了逆二次插值，顾名思义，就是在局部二次拟合反函数  $x(y)$ ，用的方法也很暴力，高数书上的拉格朗日插值：

$$x = \frac{[y - f(x_1)][y - f(x_2)]x_3}{[f(x_3) - f(x_1)][f(x_3) - f(x_2)]} + \frac{[y - f(x_2)][y - f(x_3)]x_1}{[f(x_1) - f(x_2)][f(x_1) - f(x_3)]} + \frac{[y - f(x_3)][y - f(x_1)]x_2}{[f(x_2) - f(x_3)][f(x_2) - f(x_1)]}.$$

听起来好像没有很高大上，但 Brent 的神来之笔在于，考虑到了实际过程中的 `rounderror` 问题，譬如在三个点的拟合时，如果其中两个的函数值本身就接近，会导致拟合效果很差（脑补一下对直线进行水平轴抛物线拟合的糟糕后果），所以对是否返回二分法进行了详细的分类讨论，详情参见 4。原版 Brent 还引入了一些中间变量来减少舍入误差，两年后还提出来用双曲外推替代二次插值，我就没继续学习了。

### 1.3 伪代码

---

**Algorithm 1:** Bisection Method

---

**Input:**  $a, b$  (long double),  $tol$  (long double),  $max\_iter$  (int)  
**Output:**  $c$  (long double) // Approximate root

```
1 for  $i \leftarrow 1$  to  $max\_iter$  do
2    $c \leftarrow \frac{a+b}{2}$  // Compute midpoint
3   if  $|b-a| < tol$  then
4     break // Convergence achieved
5   end
6   if  $f(a) \cdot f(c) < 0$  then
7      $b \leftarrow c$  // Update interval
8   end
9   else
10     $a \leftarrow c$  // Update interval
11  end
12 end
13 return  $c$ 
```

---

---

**Algorithm 2:** Newton-Raphson Method

---

**Input:**  $x_0$  (long double),  $tol$  (long double),  $max\_iter$  (int)  
**Output:**  $x$  (long double) // Approximate root

```
1 for  $i \leftarrow 1$  to  $max\_iter$  do
2   if  $f'(x_0) = 0$  then
3     raise error "Derivative zero"
4   end
5    $x \leftarrow x_0 - \frac{f(x_0)}{f'(x_0)}$  // Compute next approximation
6   if  $|x - x_0| < tol$  then
7     break // Convergence achieved
8   end
9    $x_0 \leftarrow x$  // Update current value
10 end
11 return  $x$ 
```

---

---

**Algorithm 3:** Hybrid Method

---

**Input:**  $a, b$  (long double),  $tol$  (long double),  $max\_iter$  (int)  
**Output:**  $c$  (long double) // Approximate root

```
1 for  $i \leftarrow 1$  to  $max\_iter$  do
2    $c \leftarrow \frac{a+b}{2}$  // Compute midpoint
3   if  $\frac{b-a}{2} < tol$  then
4     break // Convergence achieved
5   end
6   if  $f'(c) \neq 0$  and  $d = c - \frac{f(c)}{f'(c)}$  is in  $(a, b)$  then
7     if  $|d - c| < tol$  then
8       return  $d$  // Convergence achieved
9     end
10    if  $f(a) \cdot f(d) < 0$  then
11       $b \leftarrow d$  // Update interval
12    end
13    else
14       $a \leftarrow d$  // Update interval
15    end
16    continue // Proceed to next iteration
17  end
18  if  $f(a) \cdot f(c) < 0$  then
19     $b \leftarrow c$  // Update interval (Bisection)
20  end
21  else
22     $a \leftarrow c$  // Update interval (Bisection)
23  end
24 end
25 return  $c$ 
```

---

---

**Algorithm 4: Brent's Method**

---

```
Input:  $a, b, \text{tol}$  (long double),  $\text{max\_iter}$  (int),  $\text{decimal\_places}$  (int)
Output:  $\text{root}$  (long double) // Approximate root
1 if  $f(a) \cdot f(b) \geq 0$  then
2 |   raise error "Brent's method fails.  $f(a)$  and  $f(b)$  should have opposite signs."
3 end
4 if  $|f(a)| < |f(b)|$  then
5 |   Swap  $a \leftrightarrow b$  // Ensure  $a$  corresponds to the larger  $|f(x)|$ 
6 end
7  $c \leftarrow a, s \leftarrow b, \text{mflag} \leftarrow \text{True}$  // Initialize  $c, s$ , and set bisection flag
8 for  $i \leftarrow 1$  to  $\text{max\_iter}$  do
9 |   if  $f(b) \neq f(c)$  and  $f(a) \neq f(c)$  then
10 | |    $s \leftarrow \frac{a \cdot f(b) \cdot f(c)}{(f(a) - f(b)) \cdot (f(a) - f(c))} + \frac{b \cdot f(a) \cdot f(c)}{(f(b) - f(a)) \cdot (f(b) - f(c))} + \frac{c \cdot f(a) \cdot f(b)}{(f(c) - f(a)) \cdot (f(c) - f(b))}$ 
10 | |   // Inverse quadratic interpolation
11 |   end
12 |   else
13 | |    $s \leftarrow b - f(b) \cdot \frac{b - a}{f(b) - f(a)}$  // Secant method step
14 |   end
15 |   if  $s < \frac{3a + b}{4}$  or  $s > b$  or ( $\text{mflag}$  and  $|s - b| \geq |b - c|/2$ ) or (not  $\text{mflag}$  and  $|s - b| \geq |c - d|/2$ ) or
15 |   ( $\text{mflag}$  and  $|b - c| < \text{tol}$ ) or (not  $\text{mflag}$  and  $|c - d| < \text{tol}$ ) then
16 | |    $s \leftarrow \frac{a + b}{2}, \text{mflag} \leftarrow \text{True}$  // Bisection step for stability
17 |   end
18 |   else
19 | |    $\text{mflag} \leftarrow \text{False}$  // Use interpolation or secant step
20 |   end
21 |    $c \leftarrow b, f(c) \leftarrow f(b)$  // Update  $c$  and prepare next iteration
22 |   if  $f(a) \cdot f(s) < 0$  then
23 | |    $b \leftarrow s$  // Set new upper bound
24 |   end
25 |   else
26 | |    $a \leftarrow s$  // Set new lower bound
27 |   end
28 |   if  $|f(a)| < |f(b)|$  then
29 | |   Swap  $a \leftrightarrow b$  // Ensure  $|f(a)| \geq |f(b)|$  for stability
30 |   end
31 |   if  $|b - a| < \text{tol}$  then
32 | |   break // Convergence achieved within tolerance
33 |   end
34 end
35 return  $b$ 
```

---

---

**Algorithm 5:** Ridder's Method

---

**Input:**  $a, b$  (long double),  $tol$  (long double),  $max\_iter$  (int)  
**Output:**  $x$  (long double) // Approximate root

```
1 for  $i \leftarrow 1$  to  $max\_iter$  do
2    $c \leftarrow \frac{a+b}{2}$  // Compute midpoint
3    $s \leftarrow \sqrt{f(c)^2 - f(a)f(b)}$ 
4   if  $s = 0$  then
5     return  $c$  // Convergence achieved
6   end
7    $sign \leftarrow \text{sgn}(f(a) - f(b))$  // Determine sign
8    $x \leftarrow c + (c - a) \frac{f(c)}{s} \cdot sign$  // Compute new approximation
9   if  $|f(x)| < tol$  then
10    return  $x$  // Convergence achieved
11  end
12  if  $f(c) \cdot f(x) < 0$  then
13     $a \leftarrow c, b \leftarrow x$  // Update interval
14  end
15  else if  $f(a) \cdot f(x) < 0$  then
16     $b \leftarrow x$  // Update interval
17  end
18  else
19     $a \leftarrow x$  // Update interval
20  end
21  if  $|b - a| < tol$  then
22    break // Convergence achieved
23  end
24 end
25 return  $\frac{a+b}{2}$ 
```

---

```

ta/anaconda3/python.exe "d:\BaiduSyndisk\Work\Courses\Junior Fall\Co
mpPhys/repo/Week 2/Codes/Problem 1/bisection_visual.py"
Bisection Method Dynamic Visualization
Choose the interval to iterate for the root:
1: Find the root in the interval [0.0, 1.0]
2: Find the root in the interval [1.5, 2.0]
3: Custom interval, enter the left and right endpoints
Please enter your choice (1/2/3):1
Iteration 1: a = 0.00000, b = 1.00000, c = 0.50000, f(c) = 0.62500
Iteration 2: a = 0.50000, b = 1.00000, c = 0.75000, f(c) = -0.32812
Iteration 3: a = 0.50000, b = 0.75000, c = 0.62500, f(c) = 0.11914
Iteration 4: a = 0.62500, b = 0.75000, c = 0.68750, f(c) = -0.11255
Iteration 5: a = 0.62500, b = 0.68750, c = 0.65625, f(c) = 0.00137
Iteration 6: a = 0.65625, b = 0.68750, c = 0.67188, f(c) = -0.05608
Iteration 7: a = 0.65625, b = 0.67188, c = 0.66406, f(c) = -0.02747
Iteration 8: a = 0.65625, b = 0.66406, c = 0.66016, f(c) = -0.01308
Iteration 9: a = 0.65625, b = 0.66016, c = 0.65820, f(c) = -0.00586
Iteration 10: a = 0.65625, b = 0.65820, c = 0.65723, f(c) = -0.00225
Iteration 11: a = 0.65625, b = 0.65723, c = 0.65674, f(c) = -0.00044
Iteration 12: a = 0.65625, b = 0.65674, c = 0.65649, f(c) = 0.00047
Iteration 13: a = 0.65649, b = 0.65674, c = 0.65662, f(c) = 0.00002
Iteration 14: a = 0.65662, b = 0.65674, c = 0.65668, f(c) = -0.00021

Converged to root: 0.65667724689375, after 14 iterations.
PS D:\BaiduSyndisk\Work\Courses\Junior Fall\CompPhys>

ata/anaconda3/python.exe "d:\BaiduSyndisk\Work\Courses\Junior Fall\
CompPhys/repo/Week 2/Codes/Problem 1/bisection_visual.py"
Bisection Method Dynamic Visualization
Choose the interval to iterate for the root:
1: Find the root in the interval [0.0, 1.0]
2: Find the root in the interval [1.5, 2.0]
3: Custom interval, enter the left and right endpoints
Please enter your choice (1/2/3):2
Please enter your choice (1/2/3):2
Iteration 1: a = 1.50000, b = 2.00000, c = 1.75000, f(c) = -0.39062
Iteration 2: a = 1.75000, b = 2.00000, c = 1.87500, f(c) = 0.21680
Iteration 3: a = 1.75000, b = 1.87500, c = 1.81250, f(c) = -0.10815
Iteration 4: a = 1.81250, b = 1.87500, c = 1.84375, f(c) = 0.04892
Iteration 5: a = 1.81250, b = 1.84375, c = 1.82812, f(c) = -0.03096
Iteration 6: a = 1.82812, b = 1.84375, c = 1.83594, f(c) = 0.00865
Iteration 7: a = 1.82812, b = 1.83594, c = 1.83203, f(c) = -0.01124
Iteration 8: a = 1.83203, b = 1.83594, c = 1.83398, f(c) = -0.00132
Iteration 9: a = 1.83398, b = 1.83594, c = 1.83496, f(c) = 0.00366
Iteration 10: a = 1.83398, b = 1.83447, c = 1.83447, f(c) = 0.00117
Iteration 11: a = 1.83398, b = 1.83447, c = 1.83423, f(c) = -0.00007

Iteration 12: a = 1.83423, b = 1.83447, c = 1.83435, f(c) = 0.00055
Iteration 13: a = 1.83423, b = 1.83435, c = 1.83429, f(c) = 0.00024

Converged to root: 1.83428955078125, after 13 iterations.
PS D:\BaiduSyndisk\Work\Courses\Junior Fall\CompPhys>

mpPhys/repo/Week 2/Codes/Problem 1/bisection_visual.py"
Bisection Method Dynamic Visualization
Choose the interval to iterate for the root:
1: Find the root in the interval [0.0, 1.0]
2: Find the root in the interval [1.5, 2.0]
3: Custom interval, enter the left and right endpoints
Please enter your choice (1/2/3):3
Please enter the left endpoint a: -0.5
Please enter the right endpoint b: 1.5
Iteration 1: a = -0.50000, b = 1.50000, c = 0.50000, f(c) = 0.62500
Iteration 2: a = 0.50000, b = 1.50000, c = 1.00000, f(c) = -1.00000
Iteration 3: a = 0.50000, b = 1.00000, c = 0.75000, f(c) = -0.32812
Iteration 4: a = 0.50000, b = 0.75000, c = 0.62500, f(c) = 0.11914
Iteration 5: a = 0.62500, b = 0.75000, c = 0.68750, f(c) = -0.11255
Iteration 6: a = 0.62500, b = 0.68750, c = 0.65625, f(c) = 0.00137
Iteration 7: a = 0.62500, b = 0.68750, c = 0.67188, f(c) = -0.05608
Iteration 8: a = 0.65625, b = 0.67188, c = 0.66406, f(c) = -0.02747
Iteration 9: a = 0.65625, b = 0.66406, c = 0.66016, f(c) = -0.01308
Iteration 10: a = 0.65625, b = 0.66016, c = 0.65820, f(c) = -0.00586
Iteration 11: a = 0.65625, b = 0.65820, c = 0.65723, f(c) = -0.00225
Iteration 12: a = 0.65625, b = 0.65723, c = 0.65674, f(c) = -0.00044
Iteration 13: a = 0.65625, b = 0.65674, c = 0.65649, f(c) = 0.00047
Iteration 14: a = 0.65649, b = 0.65674, c = 0.65662, f(c) = 0.00002
Iteration 15: a = 0.65662, b = 0.65674, c = 0.65668, f(c) = -0.00021

Converged to root: 0.65667724689375, after 15 iterations.

```

图 2: bisection\_visual.py 三种选项对比

## 1.4 结果示例

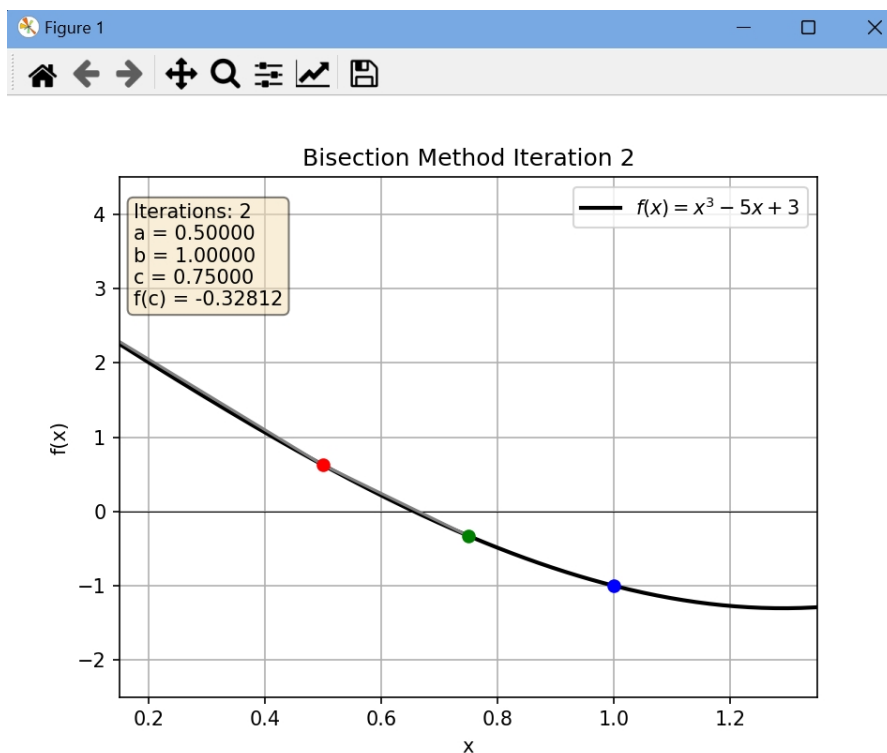


图 3: bisection\_visual.py 动画示意

```
Select a method (1-7): 6

--- Problem Steps Execution ---

Part (i): Bisection Method to find roots to 4 decimal places
Root in [-3.00, -2.00]: -2.4909
Iterations: 14
Root in [0.00, 1.00]: 0.6567
Iterations: 14
Root in [1.00, 3.00]: 1.8343
Iterations: 15

Part (ii): Newton-Raphson Method to refine roots to 14 decimal places
Refined root starting from -2.4909: -2.49086361536103
Iterations: 3
Refined root starting from 0.6567: 0.65662043104711
Iterations: 3
Refined root starting from 1.8343: 1.83424318431392
Iterations: 3

Part (iii): Hybrid Method to find roots to 14 decimal places
Root in [-3.00, -2.00] (Hybrid): -2.49086361536103
Iterations: 87
Root in [0.00, 1.00] (Hybrid): 0.65662043104711
Iterations: 104
Root in [1.00, 3.00] (Hybrid): 1.83424318431392
Iterations: 110

--- Summary of Problem Steps Results ---

Method: Bisection Method
  Root 1: -2.4909 | Iterations: 14
  Root 2: 0.6567 | Iterations: 14
  Root 3: 1.8343 | Iterations: 15

Method: Hybrid Method
  Root 1: -2.49086361536103 | Iterations: 87
  Root 2: 0.65662043104711 | Iterations: 104
  Root 3: 1.83424318431392 | Iterations: 110

Method: Newton-Raphson Method
  Root 1: -2.49086361536103 | Iterations: 3
  Root 2: 0.65662043104711 | Iterations: 3
  Root 3: 1.83424318431392 | Iterations: 3

Do you want to run the program again? (y/n): █
```

图 4: main.cpp 模式 6, 按题目顺序尝试三种方法





的 *Thompson* 问题优化，强大的谢院士当时写的就是这个，龚老师还回信了 hhh。遗传算法感觉没有名字那么玄妙，反倒有一种抽奖的感觉，不过可能也因此它的适用范围更广吧。四种算法中前两者是需要梯度的，后面的更普适一些，就像不要导数的 Brent 求根法。在 *Numerical Recipe* 的 §10.4 还学到了一种不同于它们的 *Downhill Simplex Method* 方法，它不依赖于分量的一维优化，而是更直接的缩放、反射操作，但没时间实现了...

使用 `g++ *.cpp -o main` 编译，`./main` 运行（也有已经编译好的 `main.exe`），按照提示可以选择各类方法或者一起比较，也可以自定义各种算法初始值。

## 2.3 伪代码

---

**Algorithm 6:** Steepest Descent Method

---

**Input:**  $x_0, y_0$  (float),  $\alpha$  (float), `maxIter` (int), `tol` (float)

**Output:**  $x, y$  (float) // Approximate minimum

1  $x \leftarrow x_0, y \leftarrow y_0$

2 **for**  $i \leftarrow 1$  **to** `maxIter` **do**

3     **compute** gradient  $\nabla f(x, y)$

4     **if**  $\|\nabla f(x, y)\| < \text{tol}$  **then**

5         **break** // Convergence achieved

6     **end**

7      $(x, y) \leftarrow (x, y) - \alpha \cdot \nabla f(x, y)$  // Update variables

8 **end**

9 **return**  $x, y$

---

---

**Algorithm 7:** Nonlinear Conjugate Gradient Method (Fletcher-Reeves)

---

**Input:**  $x_0, y_0$  (float),  $\text{maxIter}$  (int),  $\text{tol}$  (float)  
**Output:**  $x, y$  (float) // Approximate minimum

```
1  $x \leftarrow x_0, y \leftarrow y_0$ 
2  $\nabla f \leftarrow \nabla f(x, y)$ 
3  $\mathbf{d} \leftarrow -\nabla f$  // Initial search direction
4 for  $i \leftarrow 1$  to  $\text{maxIter}$  do
    // Backtracking Line Search (Armijo Condition)
    // Find step size  $\alpha$  such that
5
     $f(x + \alpha d_x, y + \alpha d_y) \leq f(x, y) + c \cdot \alpha \cdot (\nabla f \cdot \mathbf{d})$ 
    // Determine step size  $\alpha$ 
6
     $(x, y) \leftarrow (x, y) + \alpha \cdot \mathbf{d}$ 
7
     $\nabla f_{\text{new}} \leftarrow \nabla f(x, y)$ 
8
    if  $\|\nabla f_{\text{new}}\| < \text{tol}$  then
9
        break // Convergence achieved
10
    end
11
     $\beta \leftarrow \frac{\|\nabla f_{\text{new}}\|^2}{\|\nabla f\|^2}$  // Fletcher-Reeves coefficient
12
     $\mathbf{d} \leftarrow -\nabla f_{\text{new}} + \beta \cdot \mathbf{d}$ 
13
     $\nabla f \leftarrow \nabla f_{\text{new}}$ 
14 end
15 return  $x, y$ 
```

---

---

**Algorithm 8:** Simulated Annealing

---

**Input:**  $x_0, y_0$  (float),  $T_0$  (float),  $T_{min}$  (float),  $\alpha$  (float),  $maxIter$  (int)  
**Output:**  $x, y$  (float) // Approximate minimum

```
1  $x \leftarrow x_0, y \leftarrow y_0$ 
2  $T \leftarrow T_0$  // Initial temperature
3  $f_{current} \leftarrow f(x, y)$  // Current function value
4 for  $i \leftarrow 1$  to  $maxIter$  and  $T > T_{min}$  do
    // Generate a new candidate solution
5  $x_{new} \leftarrow x + \mathcal{U}(-0.5, 0.5), y_{new} \leftarrow y + \mathcal{U}(-0.5, 0.5)$  // RandomUniform
6  $f_{new} \leftarrow f(x_{new}, y_{new})$ 
7  $\Delta \leftarrow f_{new} - f_{current}$  // Change in function value
    // Acceptance Criterion
8 if  $\Delta < 0$  or  $e^{-\Delta/T} > \mathcal{U}(0, 1)$  then
9      $x \leftarrow x_{new}, y \leftarrow y_{new}$  // Accept new solution
10     $f_{current} \leftarrow f_{new}$  // Update current function value
11 end
12  $T \leftarrow \alpha \cdot T$  // Cool down
13 end
14 return  $x, y$ 
```

---

---

**Algorithm 9:** Genetic Algorithm

---

**Input:** Population size  $N$ , Generations  $G$ , Mutation rate  $pm$ , Crossover rate  $pc$

**Output:** Approximate minimum solution  $(x, y)$

```
1 population  $\leftarrow \{(x, y) \mid x, y \sim \mathcal{U}(-10, 10)\}$  // Initialize population randomly
2 evaluate_fitness(population) // Evaluate initial fitness
3 for generation 1 to  $G$  do
4   for  $i \leftarrow 1$  to  $N$  do
5      $a, b \leftarrow$  random_selection(population)
6     add min( $a, b$ ) to selected_population
7   end
8   for  $i \leftarrow 1$  to  $N$  step 2 do
9     if  $\mathcal{U}(0, 1) < pc$  then
10       $\alpha \leftarrow \mathcal{U}(0, 1)$ 
11      offspring_1.x  $\leftarrow \alpha \cdot$  selected_population[ $i$ ].x +  $(1 - \alpha) \cdot$  selected_population[ $i + 1$ ].x
12      offspring_1.y  $\leftarrow \alpha \cdot$  selected_population[ $i$ ].y +  $(1 - \alpha) \cdot$  selected_population[ $i + 1$ ].y
13      offspring_2.x  $\leftarrow \alpha \cdot$  selected_population[ $i + 1$ ].x +  $(1 - \alpha) \cdot$  selected_population[ $i$ ].x
14      offspring_2.y  $\leftarrow \alpha \cdot$  selected_population[ $i + 1$ ].y +  $(1 - \alpha) \cdot$  selected_population[ $i$ ].y
15      replace selected_population[ $i$ ] and [ $i + 1$ ] with offspring_1, offspring_2
16    end
17  end
18  for each individual in selected_population do
19    if  $\mathcal{U}(0, 1) < pm$  then
20      individual.x  $\leftarrow$  clip(individual.x +  $\mathcal{U}(-0.5, 0.5)$ , -10, 10)
21      individual.y  $\leftarrow$  clip(individual.y +  $\mathcal{U}(-0.5, 0.5)$ , -10, 10)
22    end
23    individual.fitness  $\leftarrow f(\text{individual.x}, \text{individual.y})$ 
24  end
25  population  $\leftarrow$  selected_population
26 end
27 best  $\leftarrow$  argmin $\{f(x, y) \mid (x, y) \in \text{population}\}$ 
28 return best.x, best.y
```

---

## 2.4 结果示例

```
PS D:\BaiduSyncdisk\Work\Courses\Junior Fall\CompPhys\repo\Week 2\Codes\Problem 2> .\main.exe

Optimization Algorithms Menu:
1. Steepest Descent Method
2. Conjugate Gradient Method
3. Simulated Annealing
4. Genetic Algorithm
5. Compare All Methods
Enter your choice (1-5): 5

Comparing All Methods with Default Parameters...

Default Parameters:
Initial x: 0.0000, Initial y: 0.0000
Steepest Descent alpha: 0.0050, maxIter: 100000, tol: 1.000e-08
Conjugate Gradient maxIter: 100000, tol: 1.000e-08
Simulated Annealing T0: 2000.000, Tmin: 1.000e-08, alpha: 0.990, maxIter: 200000
Genetic Algorithm populationSize: 100, generations: 5000, mutationRate: 0.0200, crossoverRate: 0.8000

Results:
Steepest Descent Method:
Minimum at (-0.00000, -1.57080)
Minimum value: -2.00000
Total iterations: 22503
Execution Time: 9.970e-04 seconds

Conjugate Gradient Method:
Minimum at (0.00000, -1.57080)
Minimum value: -2.00000
Total iterations: 75
Execution Time: 0.000e+00 seconds

Simulated Annealing:
Minimum at (6.28167, -1.56745)
Minimum value: -1.99998
Total iterations: 2590
Execution Time: 0.000e+00 seconds

Genetic Algorithm:
Minimum at (0.00761, -1.57460)
Minimum value: -1.99999
Total iterations: 500000
Execution Time: 6.543e-02 seconds

Do you want to run the program again? (y/n):
```

图 6: main.cpp 模式 5, 对比四种方法

### 3 题目 3: 有限深方势阱中的电子能级与波函数

#### 3.1 题目描述

Electron in the finite square-well potential is, ( $V_0 = 10 \text{ eV}$ ,  $a = 0.2 \text{ nm}$ )

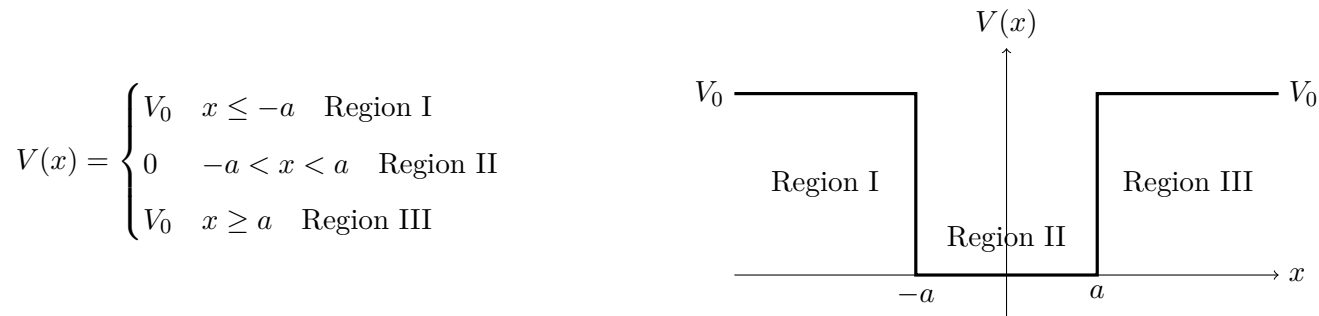


图 7: Finite square-well potential and its corresponding mathematical expression.

Find the three lowest eigen states (**both** energies and wavefunctions).

#### 3.2 程序描述

关于解析求解的方法, 详情参见 *Griffiths* 的量子力学教材, 这里不再赘述。主要是发现没时间了 (大哭)

两种算法, 一种针对本题的解析求解, 即超越方程图解法: `./Codes/Problem 3/potential.py`, 另一种是数值求解: `./Codes/Problem 3/schrodinger.py`, 差分表示哈密顿矩阵, 适用于更广泛的势能, 偷懒都用 Python 实现了, 不考虑效率还挺爽。

第二种方法是更普适的, 有时间再推广研究研究: 有限差分法用于离散化薛定谔方程中的二阶导数项, 便于在离散网格上构造哈密顿矩阵。在一维情况下, 薛定谔方程为:

在离散网格上, 每个点  $x_i$  的波函数值用  $\psi_i = \psi(x_i)$  表示, 网格步长为  $dx$ 。二阶导数  $\frac{d^2\psi(x)}{dx^2}$  在  $x_i$  处的离散近似为:

$$\left. \frac{d^2\psi(x)}{dx^2} \right|_{x_i} \approx \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{dx^2}$$

离散化后, 整个系统的哈密顿矩阵  $H$  可以用三对角矩阵的形式表示。哈密顿矩阵的对角元和非对角元由动能项和势能项共同构成。对于每个网格点  $x_i$ :

1. 对角元  $H[i, i]$  包含动能项和势能项:

$$H[i, i] = -\frac{\hbar^2}{2m} \left( \frac{-2}{dx^2} \right) + V(x_i)$$

2. 非对角元  $H[i, i+1]$  和  $H[i, i-1]$  只包含动能项:

$$H[i, i+1] = H[i, i-1] = -\frac{\hbar^2}{2m} \left( \frac{1}{dx^2} \right)$$

最终, 哈密顿矩阵  $H$  的形式为:

$$H = \begin{pmatrix} \ddots & & & & & & & 0 \\ \ddots & -\frac{2\hbar^2}{2mdx^2} + V(x_{i-1}) & & \frac{\hbar^2}{2mdx^2} & & & & \\ \ddots & & \frac{\hbar^2}{2mdx^2} & & -\frac{2\hbar^2}{2mdx^2} + V(x_i) & & \frac{\hbar^2}{2mdx^2} & \ddots \\ & & & \frac{\hbar^2}{2mdx^2} & & -\frac{2\hbar^2}{2mdx^2} + V(x_{i+1}) & & \ddots \\ 0 & & & & & & \ddots & \ddots \end{pmatrix}$$

这个三对角矩阵通过求解本征值问题，可以得到系统的能量特征值  $E$  以及对应的波函数  $\psi$ 。

### 3.3 伪代码

---

**Algorithm 10:** Intersection Calculation and Plotting of Functions

---

**Input:**  $z_0$  (float),  $f_1(z)$ ,  $f_2(z)$ ,  $f_3(z)$

**Output:** Intersections between  $f_1(z)$  and  $f_3(z)$ ,  $f_2(z)$  and  $f_3(z)$

```

1  $z_0 \leftarrow 3.240175521;$  // Initialize parameter  $z_0$ 
2  $f_1(z) \leftarrow \tan(z);$   $f_2(z) \leftarrow -\cot(z);$   $f_3(z) \leftarrow \sqrt{(\frac{z_0}{z})^2 - 1};$  // Define functions
3  $z \in [0.1, z_0], z \neq n \cdot \frac{\pi}{2}, n \in \mathbb{Z};$  // Define valid range for  $z$ 
4  $z \leftarrow \text{Solve}(f_1(z) = f_3(z));$  // Find intersections of  $f_1$  and  $f_3$ 
5  $z \leftarrow \text{Solve}(f_2(z) = f_3(z));$  // Find intersections of  $f_2$  and  $f_3$ 
6  $z \leftarrow \text{Exclude}(z \approx n \cdot \frac{\pi}{2});$  // Exclude invalid solutions
7  $\text{Plot}(f_1(z), f_2(z), f_3(z));$  // Plot all functions
8  $\text{Mark}(f_1(z) \cap f_3(z), \text{red});$  // Mark valid intersections of  $f_1$  and  $f_3$ 
9  $\text{Mark}(f_2(z) \cap f_3(z), \text{green});$  // Mark valid intersections of  $f_2$  and  $f_3$ 
10 Add axes, grid, title, legend; // Customize the plot
11 return Intersections( $f_1(z)$ ,  $f_3(z)$ ,  $f_2(z)$ ,  $f_3(z)$ ); // Return valid intersections

```

---



---

**Algorithm 11:** Finite Difference Solution of the Schrödinger Equation for a Finite Potential Well

---

**Input:** Constants  $\hbar$ ,  $m$ ,  $V_0$ ,  $a$ ,  $N$ ,  $L$

**Output:** Bound state energy levels  $E_{\text{bound}}$  (eV)

```
1  $N \leftarrow 1000$ ,  $L \leftarrow 10 \times 10^{-9}$ ; // Number of points and length scale
2  $dx \leftarrow \frac{2L}{N-1}$ ; // Spatial step size
3  $x_i \leftarrow -L + i \cdot dx$ , for  $i = 0$  to  $N-1$ ; // Generate grid points
4  $V_i \leftarrow \begin{cases} -V_0, & \text{if } |x_i| \leq a \\ // \text{Potential inside the well} & // \text{Potential outside the well} \end{cases}$ 
5 for  $i \leftarrow 1$  to  $N-2$  do
6 |  $H_{i,i-1} \leftarrow 1$ ,  $H_{i,i} \leftarrow -2$ ,  $H_{i,i+1} \leftarrow 1$ ; // Fill the Hamiltonian matrix
7 end
8  $H \leftarrow H \cdot \frac{-(\hbar)^2}{2m dx^2}$  // Construct Hamiltonian matrix  $H$ 
9  $H_{i,i} \leftarrow H_{i,i} + V_i$ , for  $i = 0$  to  $N-1$ ; // Add potential to Hamiltonian
10 Find eigenvalues  $E_n$  and eigenvectors  $\psi_n$  such that  $H\psi_n = E_n\psi_n$ ; // Solve eigenvalue problem
11  $E_{\text{bound}} \leftarrow \{E_n \mid E_n < 0, n > 0\}$ ; // Select bound state energies
12  $\psi_{\text{bound}} \leftarrow$  corresponding eigenvectors  $\psi_n$ ; // Extract corresponding wavefunctions
13  $E_{\text{bound\_eV}} \leftarrow E_{\text{bound}}/e$ ; // Convert energy to eV
14  $\psi_n \leftarrow \psi_n / \sqrt{\sum_{i=0}^{N-1} |\psi_n(x_i)|^2 dx}$ , for each  $\psi_n$  in  $\psi_{\text{bound}}$ ; // Normalize each wavefunction
15 Plot  $\psi_n(x)$  shifted by  $E_n$ , for each  $E_n$  in  $E_{\text{bound\_eV}}$ ; // Plot normalized wavefunctions
16 return  $E_{\text{bound\_eV}}$ ; // Return bound state energies
```

---

### 3.4 结果示例

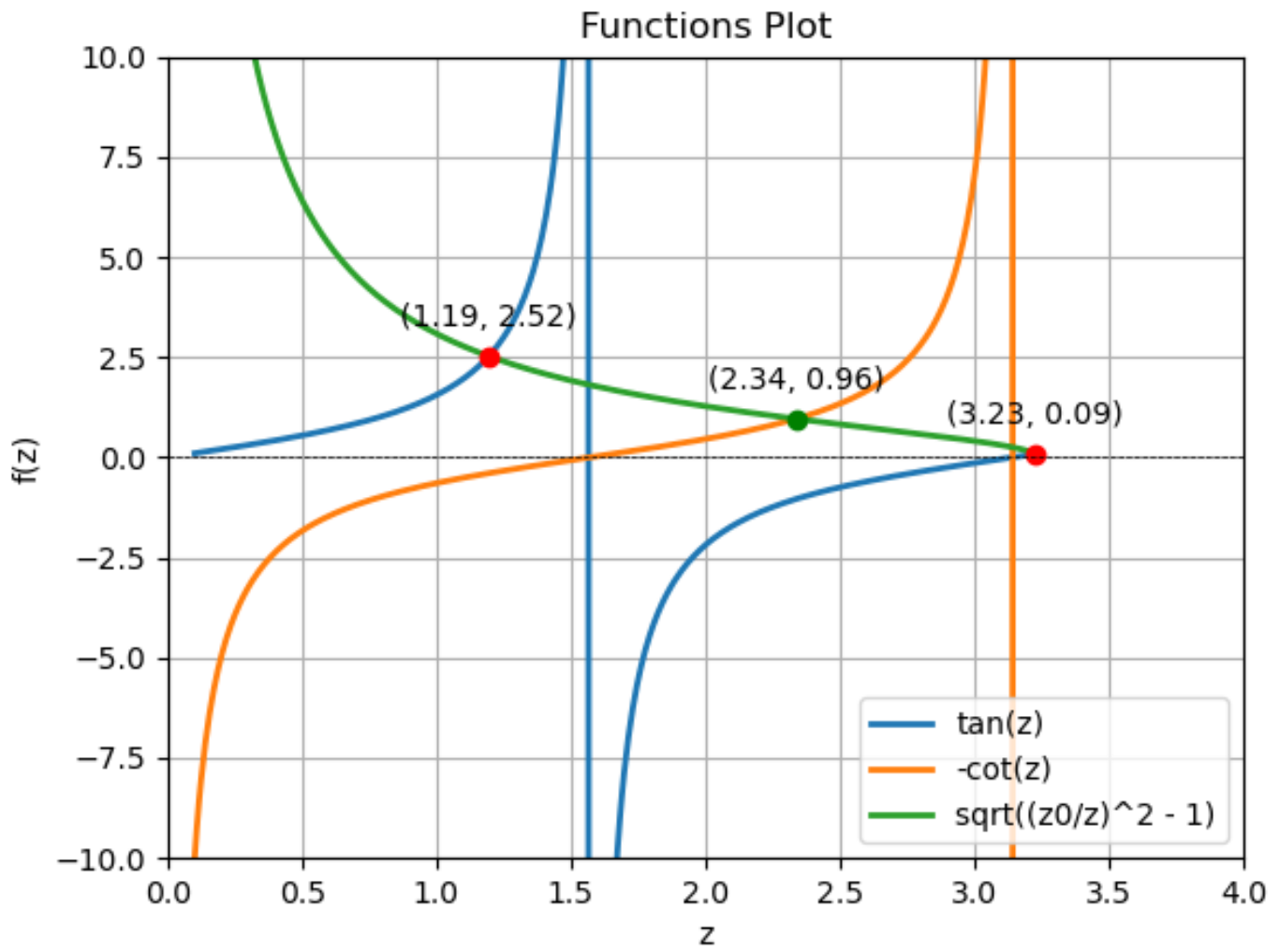


图 8: potential.py 解析图解法

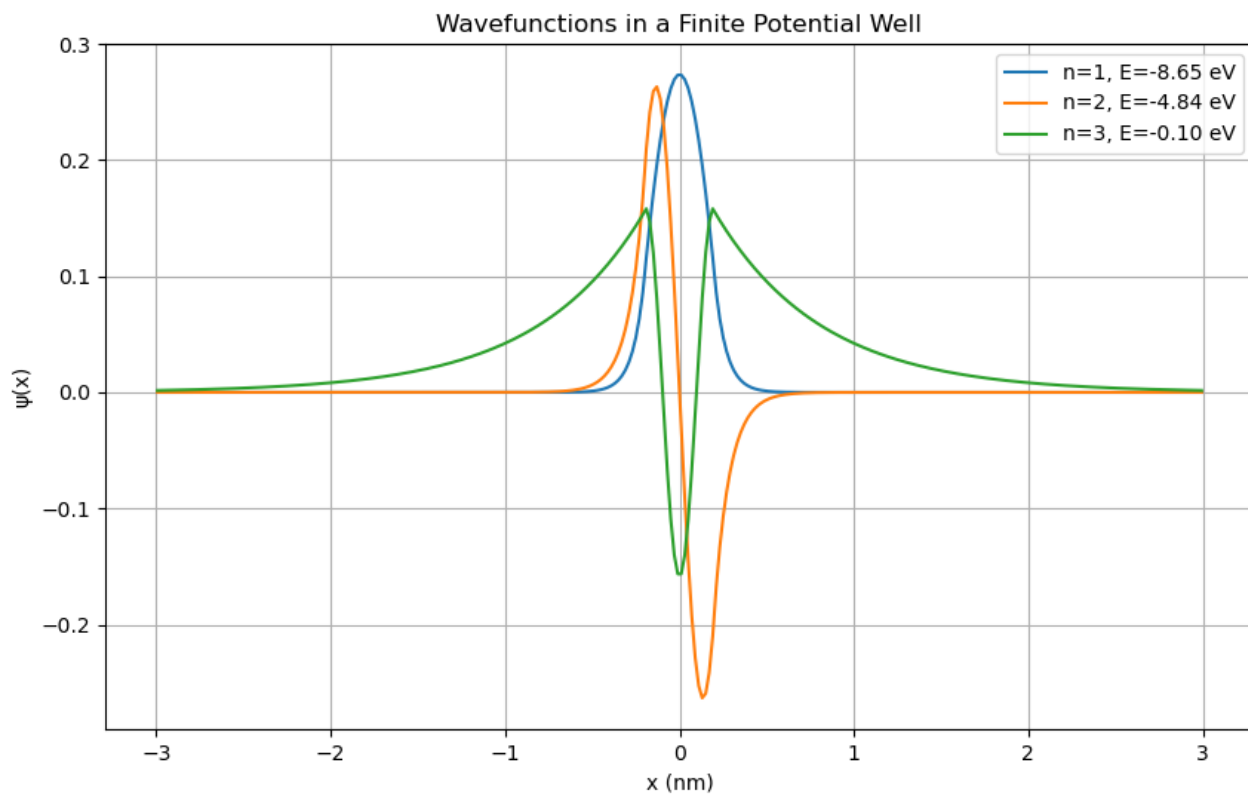


图 9: schrodinger.py 数值差分法