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 $\boldsymbol{x^3 - 5x + 3 = 0}$

(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® 画出函数草图如下：

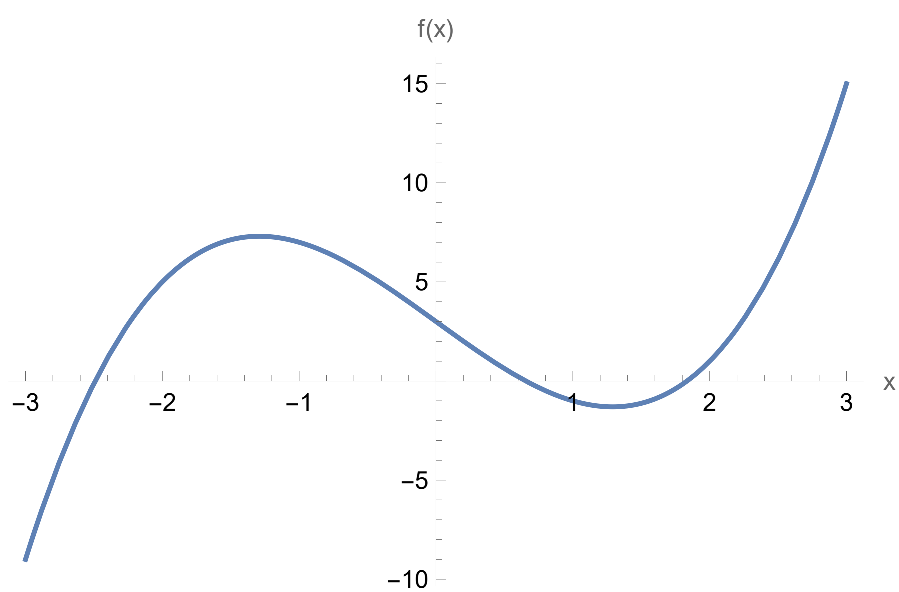


图 1: Plot of $x^3 - 5x + 3 = 0$.

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 $h(x)=f(x)\mathrm{e}^{ax}$，用一个指数因子来模拟潜在的非线性行为

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

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
for i ← 1 to max_iter do
    c ← (a + b) / 2                              // Compute midpoint
    if |b − a| < tol then
        break                                    // Convergence achieved
    end
    if f(a) · f(c) < 0 then
        b ← c                                    // Update interval
    end
    else
        a ← c                                    // Update interval
    end
end
return c
```

Algorithm 2: Newton-Raphson Method

```
Input: x0 (long double), tol (long double), max_iter (int)
Output: x (long double)                          // Approximate root
for i ← 1 to max_iter do
    if f'(x0) = 0 then
        raise error "Derivative zero"
    end
    x ← x0 − f(x0) / f'(x0)                      // Compute next approximation
    if |x − x0| < tol then
        break                                    // Convergence achieved
    end
    x0 ← x                                       // Update current value
end
return x
```

Algorithm 3: Hybrid Method

Input: a, b (long double), `tol` (long double), `max_iter` (int)

Output: c (long double) // Approximate root

```
for i ← 1 to max_iter do
    c ← (a + b) / 2                              // Compute midpoint
    if (b − a) / 2 < tol then
        break                                    // Convergence achieved
    end
    if f′(c) ≠ 0 and d = c − f(c)/f′(c) is in (a, b) then
        if |d − c| < tol then
            return d                             // Convergence achieved
        end
        if f(a) · f(d) < 0 then
            b ← d                                // Update interval
        end
        else
            a ← d                                // Update interval
        end
        continue                                 // Proceed to next iteration
    end
    if f(a) · f(c) < 0 then
        b ← c                                    // Update interval (Bisection)
    end
    else
        a ← c                                    // Update interval (Bisection)
    end
end
return c
```

Algorithm 4: Brent's Method

Input: a, b, `tol` (long double), `max_iter` (int), `decimal_places` (int)

Output: *root* (long double) // Approximate root

if $f(a) \cdot f(b) \geq 0$ **then**
  raise error *"Brent's method fails. f(a) and f(b) should have opposite signs."*
end
if $|f(a)| < |f(b)|$ **then**
  Swap $a \leftrightarrow b$ // Ensure a corresponds to the larger $|f(x)|$
end
$c \leftarrow a$, $s \leftarrow b$, `mflag` $\leftarrow$ True // Initialize c, s, and set bisection flag
for $i \leftarrow 1$ ***to*** `max_iter` **do**
  if $f(b) \neq f(c)$ ***and*** $f(a) \neq f(c)$ **then**

$$s \leftarrow \frac{a \cdot f(b) \cdot f(c)}{(f(a) - f(b)) \cdot (f(a) - f(c))} + \frac{b \cdot f(a) \cdot f(c)}{(f(b) - f(a)) \cdot (f(b) - f(c))} + \frac{c \cdot f(a) \cdot f(b)}{(f(c) - f(a)) \cdot (f(c) - f(b))}$$

    // Inverse quadratic interpolation
  end
  else

$$s \leftarrow b - f(b) \cdot \frac{b - a}{f(b) - f(a)}$$

    // Secant method step
  end
  if $s < \frac{3a + b}{4}$ ***or*** $s > b$ ***or*** *(*`mflag` ***and*** $|s - b| \geq |b - c|/2$*)* ***or*** *(**not*** `mflag` ***and*** $|s - b| \geq |c - d|/2$*)* ***or*** *(*`mflag` ***and*** $|b - c| <$ `tol`*)* ***or*** *(**not*** `mflag` ***and*** $|c - d| <$ `tol`*)* **then**
    $s \leftarrow \frac{a + b}{2}$, `mflag` $\leftarrow$ True // Bisection step for stability
  end
  else
    `mflag` $\leftarrow$ False // Use interpolation or secant step
  end
  $c \leftarrow b$, $f(c) \leftarrow f(b)$ // Update c and prepare next iteration
  if $f(a) \cdot f(s) < 0$ **then**
    $b \leftarrow s$ // Set new upper bound
  end
  else
    $a \leftarrow s$ // Set new lower bound
  end
  if $|f(a)| < |f(b)|$ **then**
    Swap $a \leftrightarrow b$ // Ensure $|f(a)| \geq |f(b)|$ for stability
  end
  if $|b - a| <$ `tol` **then**
    break // Convergence achieved within tolerance
  end
end
return b

Algorithm 5: Ridder's Method

Input: a, b (long double), `tol` (long double), `max_iter` (int)

Output: x (long double) // Approximate root

for $i \leftarrow 1$ ***to*** *max_iter* **do**

  $c \leftarrow \frac{a+b}{2}$ // Compute midpoint

  $s \leftarrow \sqrt{f(c)^2 - f(a)f(b)}$

  if $s = 0$ **then**

    return c // Convergence achieved

  end

  $sign \leftarrow \operatorname{sgn}(f(a) - f(b))$ // Determine sign

  $x \leftarrow c + (c-a)\frac{f(c)}{s} \cdot sign$ // Compute new approximation

  if $|f(x)| < tol$ **then**

    return x // Convergence achieved

  end

  if $f(c) \cdot f(x) < 0$ **then**

    $a \leftarrow c$, $b \leftarrow x$ // Update interval

  end

  else if $f(a) \cdot f(x) < 0$ **then**

    $b \leftarrow x$ // Update interval

  end

  else

    $a \leftarrow x$ // Update interval

  end

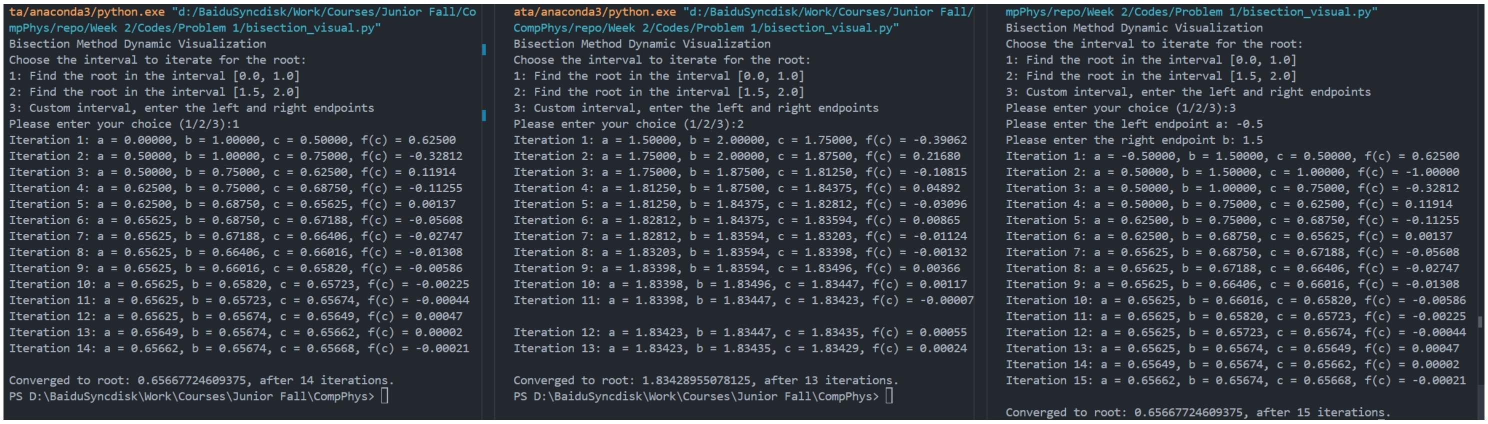
  if $|b - a| < tol$ **then**

    break // Convergence achieved

  end

end

return $\frac{a+b}{2}$



```
ta/anaconda3/python.exe "d:/BaiduSyncdisk/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: 0.65667724609375, after 14 iterations.
PS D:\BaiduSyncdisk\Work\Courses\Junior Fall\CompPhys>
```

```
ata/anaconda3/python.exe "d:/BaiduSyncdisk/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
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 = 1.83496, 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:\BaiduSyncdisk\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 = 0.65625, 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: 0.65667724609375, after 15 iterations.
```

图 2: `bisection_visual.py` 三种选项对比

1.4 结果示例

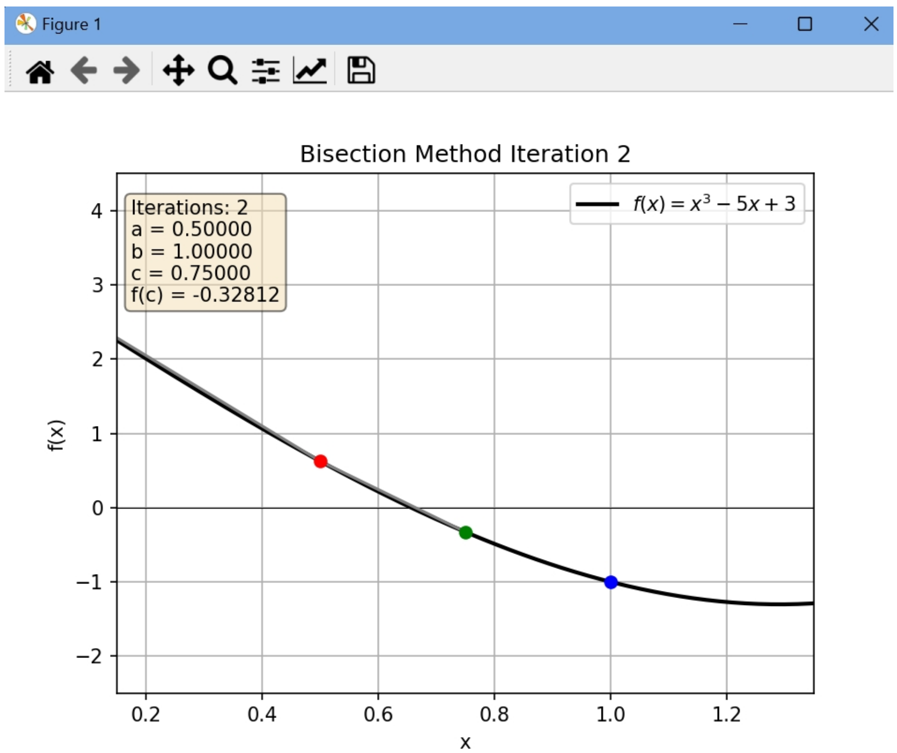


图 3: `bisection_visual.py` 动画示意

```
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

```
PS D:\BaiduSyncdisk\Work\Courses\Junior Fall\CompPhys\repo\Week 2\Codes\Problem 1> ./main.exe

Function Plot:
                              |
                              |                         *
                              |                        *
             **********       |
           **          **     |                       *
         **              ***  |                      *
        *                   **|                     *
       *                      **                   *
      *                       | **                *
                              |   **            **
-----*------------------------+-----****-----***-----------
    *                         |         *****
   *                          |
                              |
  *                           |
                              |
 *                            |
                              |
*                             |
                              |
-3       -2        -1         0        1         2         3
x range: [-3, 3]
y range: [-10, 10]


Available methods:
1. Bisection Method
2. Hybrid Method
3. Brent Method
4. Ridder Method
5. Newton-Raphson Method
6. Problem Steps Mode
7. Compare All Methods
Select a method (1-7): 7

--- Comparison of All Methods (Precision: 1e-14) ---

Method                   Root 1 (-3,-2)              Iterations     Root 2 (0,1)                Iterations     Root 3 (1,3)                Iterations
-----------------------------------------------------------------------------------------------------------------------------------
Bisection Method         -2.490863615361029          47             0.656620431047109           47             1.834243184313927           48
Newton-Raphson Method    -2.490863615361032          4              0.656620431047110           5              1.834243184313922           6
Hybrid Method            -2.490863615361032          87             0.656620431047110           104            1.834243184313922           110
Brent's Method           -2.490863615361032          43             0.656620431047110           21             1.834243184313922           36
Ridder's Method          -2.490863615361032          3              0.656620431047111           5              1.834243184313922           5

Note: Precision is set to 1e-14, output displays 15 decimal places.
```

图 5: `main.cpp` 模式 7，对比五种方法

2 题目 2：二维函数的极小值搜索

2.1 题目描述

Search for the minimum of the function $g(x, y) = \sin(x + y) + \cos(x + 2y)$ in the whole space.

2.2 程序描述

原本这道题是我更感兴趣的，但第一题已经耗费我大量的时间，只能简单实现了四种有名的优化算法：最速下降法，共轭梯度下降法，模拟退火，遗传算法。尤其是共轭梯度下降法，在 *Numerical Recipe* 的 §10.6 中有详细的介绍，不得不承认这书写得还行。以及之前一直没搞懂的命名错误，最速下降法的最速[1]原来不代表 *Learning Rate* α 是有调节器的，而是指古老的固定步长。而共轭梯度法则是在新的“共轭方向”上进行搜索，这样可以避免最速下降法的 *zig zag* 现象，当然施法范围有限，主要对二次型和稀疏矩阵有效。模拟退火算法使我想起了大一和伙伴们电磁学荣誉课

[1] 但武思怡前辈分享的最速下降法好像是另一个意思，疑惑 & 学习 *ing*。

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), α (float), maxIter (int), tol (float)
Output: x, y (float)                              // Approximate minimum
x ← x_0, y ← y_0
for i ← 1 to maxIter do
    compute gradient ∇f(x, y)
    if ‖∇f(x, y)‖ < tol then
        break                                     // Convergence achieved
    end
    (x, y) ← (x, y) − α · ∇f(x, y)                // Update variables
end
return x, y
```

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

Input: x_0, y_0 (float), `maxIter` (int), `tol` (float)

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

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

$\nabla f \leftarrow \nabla f(x, y)$

$\mathbf{d} \leftarrow -\nabla f$ // Initial search direction

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

 // Backtracking Line Search (Armijo Condition)

 // Find step size α such that

$$f(x + \alpha d_x, y + \alpha d_y) \leq f(x, y) + c \cdot \alpha \cdot (\nabla f \cdot \mathbf{d})$$

 // Determine step size α

 $(x, y) \leftarrow (x, y) + \alpha \cdot \mathbf{d}$

 $\nabla f_{\text{new}} \leftarrow \nabla f(x, y)$

 if $\|\nabla f_{new}\| <$ *tol* **then**

  break // Convergence achieved

 end

 $\beta \leftarrow \dfrac{\|\nabla f_{\text{new}}\|^2}{\|\nabla f\|^2}$ // Fletcher-Reeves coefficient

 $\mathbf{d} \leftarrow -\nabla f_{\text{new}} + \beta \cdot \mathbf{d}$

 $\nabla f \leftarrow \nabla f_{\text{new}}$

end

return x, y

Algorithm 8: Simulated Annealing

Input: x_0, y_0 (float), T_0 (float), `Tmin` (float), α (float), `maxIter` (int)

Output: x, y (float) `// Approximate minimum`

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

$T \leftarrow T_0$ `// Initial temperature`

$f_{\text{current}} \leftarrow f(x, y)$ `// Current function value`

for $i \leftarrow 1$ ***to*** *`maxIter`* ***and*** $T >$ *`Tmin`* **do**

  `// Generate a new candidate solution`

  $x_{\text{new}} \leftarrow x + \mathcal{U}(-0.5, 0.5)$, $y_{\text{new}} \leftarrow y + \mathcal{U}(-0.5, 0.5)$ `// RandomUniform`

  $f_{\text{new}} \leftarrow f(x_{\text{new}}, y_{\text{new}})$

  $\Delta \leftarrow f_{\text{new}} - f_{\text{current}}$ `// Change in function value`

  `// Acceptance Criterion`

  if $\Delta < 0$ ***or*** $e^{-\Delta/T} > \mathcal{U}(0, 1)$ **then**

    $x \leftarrow x_{\text{new}}$, $y \leftarrow y_{\text{new}}$ `// Accept new solution`

    $f_{\text{current}} \leftarrow f_{\text{new}}$ `// Update current function value`

  end

  $T \leftarrow \alpha \cdot T$ `// Cool down`

end

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)

population $\leftarrow \{(x, y) \mid x, y \sim \mathcal{U}(-10, 10)\}$ `// Initialize population randomly`
evaluate_fitness(population) `// Evaluate initial fitness`
for *generation* 1 *to* `G` **do**
    for $i \leftarrow 1$ ***to*** `N` **do**
        $a, b \leftarrow$ random_selection(population)
        add min(a, b) to selected_population
    end
    for $i \leftarrow 1$ ***to*** `N` *step 2* **do**
        if $\mathcal{U}(0, 1) <$ `pc` **then**
            $\alpha \leftarrow \mathcal{U}(0, 1)$
            offspring_1.x $\leftarrow \alpha\cdot$ selected_population[i].x + (1 - α)$\cdot$ selected_population[$i+1$].x
            offspring_1.y $\leftarrow \alpha\cdot$ selected_population[i].y + (1 - α)$\cdot$ selected_population[$i+1$].y
            offspring_2.x $\leftarrow \alpha\cdot$ selected_population[$i+1$].x + (1 - α)$\cdot$ selected_population[i].x
            offspring_2.y $\leftarrow \alpha\cdot$ selected_population[$i+1$].y + (1 - α)$\cdot$ selected_population[i].y
            replace selected_population[i] and [$i+1$] with offspring_1, offspring_2
        end
    end
    for *each individual in selected_population* **do**
        if $\mathcal{U}(0, 1) <$ `pm` **then**
            individual.x $\leftarrow$ clip(individual.x + $\mathcal{U}(-0.5, 0.5)$, -10, 10)
            individual.y $\leftarrow$ clip(individual.y + $\mathcal{U}(-0.5, 0.5)$, -10, 10)
        end
        individual.fitness $\leftarrow f(individual.x, individual.y)$
    end
    population $\leftarrow$ selected_population
end
best $\leftarrow$ argmin$\{f(x, y) \mid (x, y) \in population\}$
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, ($V_0 = 10\,\mathrm{eV}$, $a = 0.2\,\mathrm{nm}$)

$$V(x) = \begin{cases} V_0 & x \le -a \quad \text{Region I} \\ 0 & -a < x < a \quad \text{Region II} \\ V_0 & x \ge a \quad \text{Region III} \end{cases}$$

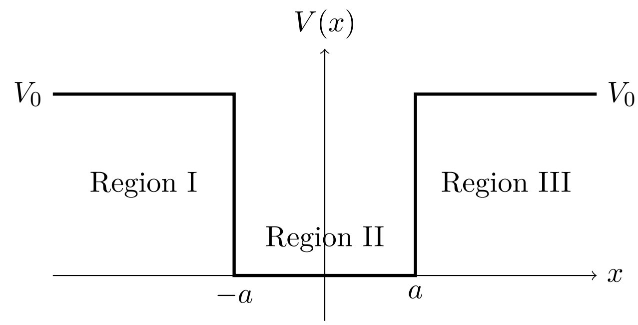


图 7: Finite square-well potential and its corresponding mathematical expression.

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：

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

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：

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

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 ψ。

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 $f_3(z)$

$z_0 \leftarrow 3.240175521$; `// Initialize parameter` z_0

$f_1(z) \leftarrow \tan(z)$; $\quad f_2(z) \leftarrow -\cot(z)$; $\quad f_3(z) \leftarrow \sqrt{\left(\frac{z_0}{z}\right)^2-1}$; `// Define functions`

$z \in [0.1,\, z_0]$, $z \neq n \cdot \frac{\pi}{2},\ n \in \mathbb{Z}$; `// Define valid range for` z

$z \leftarrow \mathrm{Solve}(f_1(z) = f_3(z))$; `// Find intersections of` f_1 `and` f_3

$z \leftarrow \mathrm{Solve}(f_2(z) = f_3(z))$; `// Find intersections of` f_2 `and` f_3

$z \leftarrow \mathrm{Exclude}(z \approx n \cdot \frac{\pi}{2})$; `// Exclude invalid solutions`

$\mathrm{Plot}(f_1(z), f_2(z), f_3(z))$; `// Plot all functions`

$\mathrm{Mark}(f_1(z) \cap f_3(z), \mathrm{red})$; `// Mark valid intersections of` f_1 `and` f_3

$\mathrm{Mark}(f_2(z) \cap f_3(z), \mathrm{green})$; `// Mark valid intersections of` f_2 `and` f_3

Add axes, grid, title, legend; `// Customize the plot`

return $\mathrm{Intersections}(f_1(z), f_3(z), f_2(z), f_3(z))$; `// 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_{bound} (eV)

$N \leftarrow 1000, \quad L \leftarrow 10 \times 10^{-9}$; `// Number of points and length scale`

$dx \leftarrow \dfrac{2L}{N-1}$; `// Spatial step size`

$x_i \leftarrow -L + i \cdot dx, \quad \text{for } i = 0 \text{ to } N-1$; `// Generate grid points`

$V_i \leftarrow \begin{cases} -V_0, & \text{if } |x_i| \leq a \\ \texttt{// Potential inside the well} 0, & \text{otherwise } \texttt{// Potential outside the well} \end{cases}$

for $i \leftarrow 1$ ***to*** $N-2$ **do**

  $H_{i,i-1} \leftarrow 1, \quad H_{i,i} \leftarrow -2, \quad H_{i,i+1} \leftarrow 1$; `// Fill the Hamiltonian matrix`

end

$H \leftarrow H \cdot \dfrac{-(\hbar)^2}{2m\,dx^2}$ `// Construct Hamiltonian matrix` H

$H_{i,i} \leftarrow H_{i,i} + V_i, \quad \text{for } i = 0 \text{ to } N-1$; `// Add potential to Hamiltonian`

Find eigenvalues E_n and eigenvectors ψ_n such that $H\psi_n = E_n\psi_n$; `// Solve eigenvalue problem`

$E_{\text{bound}} \leftarrow \{E_n \mid E_n < 0,\ n > 0\}$; `// Select bound state energies`

$\psi_{\text{bound}} \leftarrow$ corresponding eigenvectors ψ_n; `// Extract corresponding wavefunctions`

$E_{\text{bound\_eV}} \leftarrow E_{\text{bound}}/e$; `// Convert energy to eV`

$\psi_n \leftarrow \psi_n / \sqrt{\sum_{i=0}^{N-1} |\psi_n(x_i)|^2\,dx}, \quad \text{for each } \psi_n \text{ in } \psi_{\text{bound}}$; `// Normalize each wavefunction`

Plot $\psi_n(x)$ shifted by E_n, for each E_n in $E_{\text{bound\_eV}}$; `// Plot normalized wavefunctions`

return $E_{\text{bound\_eV}}$; `// Return bound state energies`

3.4 结果示例

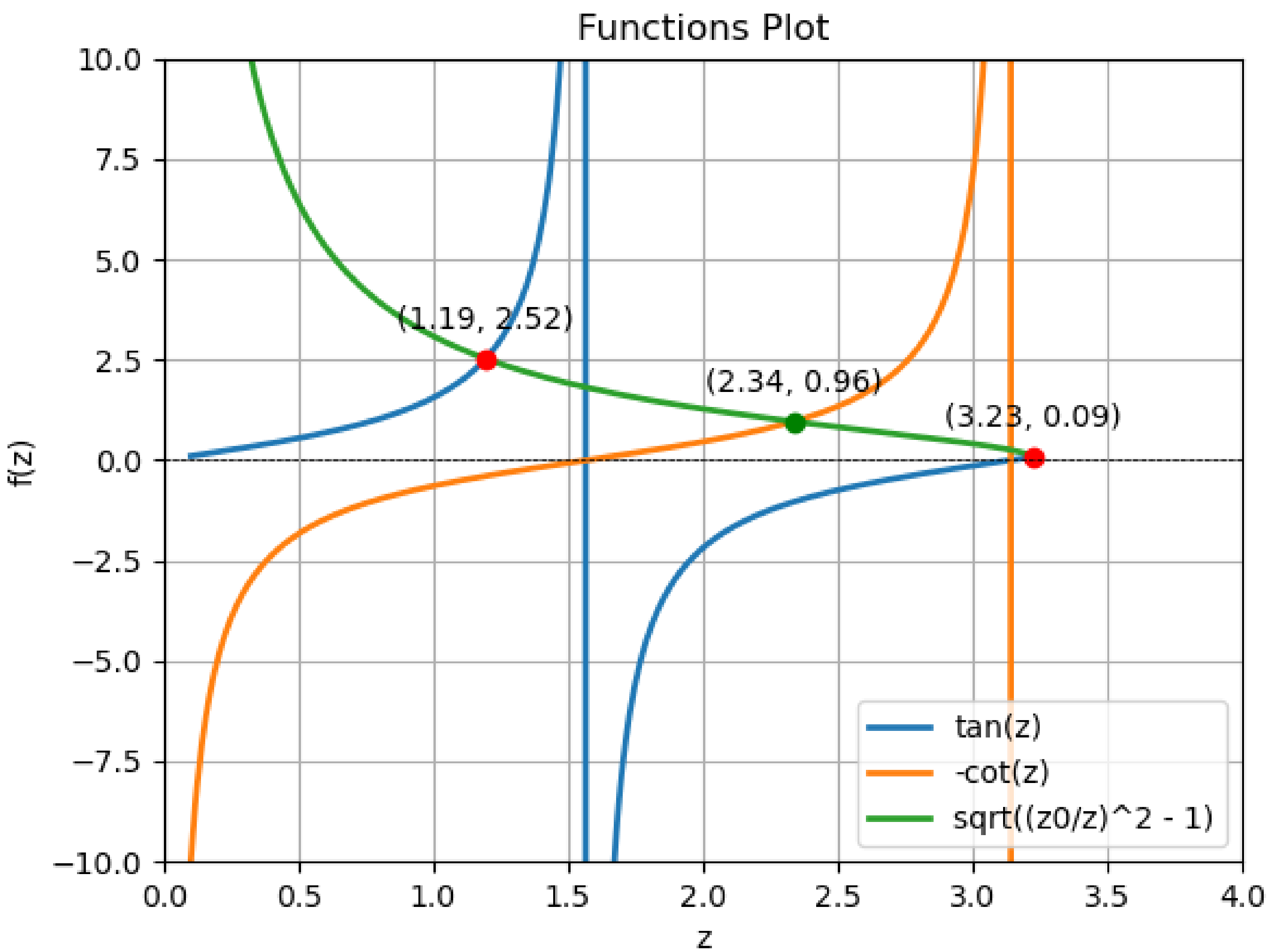


图 8: `potential.py` 解析图解法

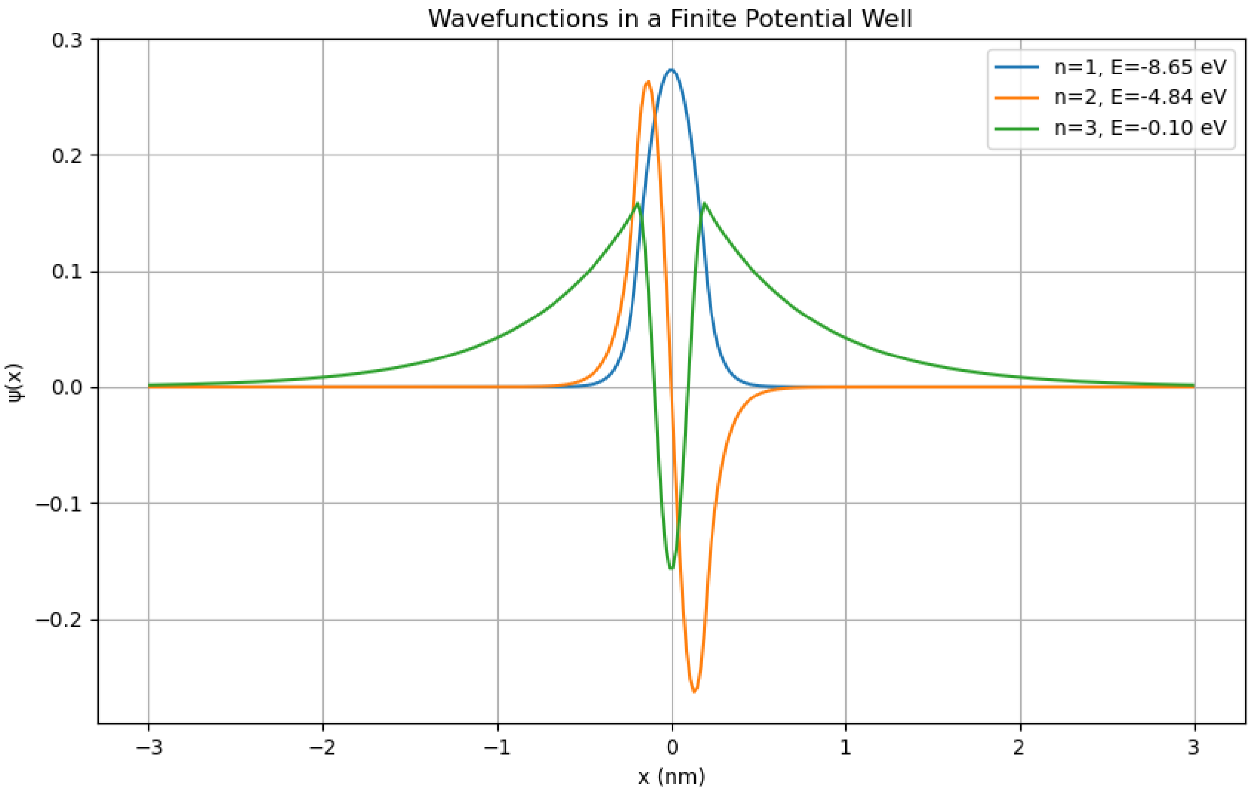


图 9: `schrodinger.py` 数值差分法