数值实验报告 I

实验名称	计算方法上机实践				实验时间		2025年 3月 1日
姓名	秦浩政 郭凯平 刘桂凡 刘佳鑫	班级	数据 2301	学号	2306030214 2306020510 2309050116 2309050117	成绩	

一、实验目的，内容

实验 1.3 算法稳定性

观察误差传播对计算结果的影响，体会选择稳定性好的算法的重要性。

实验内容:

由定积分定义序列 In =∫ x^n*e ^(x-1)dx(从 0到 1上的积分),n=1,2,... 易知:

用数学软件求出精确值(实际是精度很高的近似值)，用课本上的两种方法计算

实验 1.4 相近数相减问题

观察相近数相减导致的数值现象，研究减少误差的对策

实验内容:

(1)计算表达式√(x+a)-√ x，x>>|al，选取不同的测试用例计算，观察计算结果的精度变化，改写表达式的形式提高计算精度.

(2)计算表达式(arctanx-x)/x³,|xl ≤ 1，选取不同的测试用例计算，观察计算结果的精度变化，如何计算可以提高计算精度.

(3)用求根公式分别求解下面两个方程:

x^2+3x-2=0，x^2-1020x+1020=0.

观察计算结果的精度.若计算结果与真解相差很大，分析其原因，提出合适的算法确保计算结果有较高的精度.

实验 2.1 二分法

编程实现如下算法:用二分法求方程 f(x)=0 在区间[a,b]内的根的近似值，要求绝对误差限为 8.应注

意算法的通用性和效率

实验内容：.

测试用例参考:

(1)求方程 x3-7x+3=0 在区间[0,1]内的根;(2)求方程 lnx=0.5+cosx 在区间[1,2]内的根

二、算法描述

实验 1.3:

方法一：直接递推法

（1） **定义函数，该函数就是递推公式**

（2） **设置计算个数为 8 个，通过循环计算结果。计算各项的值**

（3） **通过绘图包。将结果以散点图的形式呈现**

方法二：反向地递推法

（1） **也是定义一个函数，该函数就是反向的递推公式，反向可以有效的减小 n 对该递推公式的影响**

（2） **设置计算个数为 8 个，通过循环计算结果。计算各项的值**

（3） **通过绘图包。将结果以散点图的形式呈现**

实验 1.4

（1） **由于 x>>|a|，直接计算可能会导致数值精度损失，因为两个相近的数相减会放大相对误差。为了提高计算精度，可以采用有理化方法将√(x+a)-√ x 转化为 a/ (√(x+a)+√ x) ，这种形式避免了两个相近数的直接相减，减小了数值误差**

（2） **由于 x 的绝对值小于 1，arctanx 和 x 的值相近，会有误差，因此采用 arctanx 的泰勒展开，避免了两个相近数相减可能带来的误差提升算法精度**

（3） **第一个直接运用求根公式计算即可，由于第二个在计算-b+（-）b^2-4ac 的时候，由于两个数相近，可能带来误差，因此我们计算的时候先计算相加的结果，当-b 是正数的时候计算加法，-b 是减法的时候采用减法，最后运用两根之积是 a/c 得到另一个根的结果**

实验 2.1

- **function: 目标函数 f(x) 。**
- **a_begin: 区间的左端点 a 。**
- **b_end: 区间的右端点 b 。**
- **e: 区间的绝对误差限，当 |b - a| < e 时结束算法。**
- **e2: 函数值的绝对误差限，当函数值 |f({a + b}{2})| < e2 时结束算法。**

算法过程:1.根据给出的区间确定初始的左右端点，并计算函数值 2.循环条件：使用 and，当两端点足够接近，即区间足够小，或是区间中点的函数值足够小的时候结束循环 3.循环体内部：计算中点处函数值与端点处函数值符号的异同：若与右边端点处函数值同号则将中点处定义为新的右端点，与左边同号则定义为新的左端点；若中点处函数值为零则刚好找到函数零点，直接输出结果 4.结束条件：当 |b - a| < e 或 |f(m)| < e2 时，退出循环，返回区间中点作为根的近似值

注释:1.精度控制：通过两层条件判断（区间长度和函数值误差）确保根的近似值满足给定的精度要求。2.区间更新：通过每次根据中点计算函数值并判断函数值符号，来逐渐缩小区间，最终找到函数的根。

三程序代码

实验 1.3

(1)

```python
import matplotlib.pyplot as plt


def ditui(n):
    # 创建一个列表来存储计算结果
    fib_series = [0.6321]
    for i in range(1, n + 1):
        next_value = 1 - i * fib_series[i - 1]  # 递推公式
        fib_series.append(next_value)

    return fib_series[n]


# 计算 I(n) 的值
n = 8
results = []
for i in range(n + 1):
    result = ditui(i)
    results.append(result)
    print(f"I({i}) = {result}")

# 绘制散点图
plt.scatter(range(n + 1), results, color='blue')

# 添加标题和标签
plt.title('Scatter Plot of I(n)')
plt.xlabel('n')
plt.ylabel('I(n)')

# 显示网格
plt.grid()

# 显示图形
plt.show()
```

```python
import matplotlib.pyplot as plt


def dituI(n):
    # 创建一个列表来存储计算结果
    fib_series = [0.6321]
    x = 0.3679
    for i in range(0, n + 1):
        fib_series[i] = 1 / (i + 1) - x / (i + 1)  # 递推
        fib_series.append(x)

    return fib_series[n]


# 计算 I(n) 的值
n = 8
results = []
for i in range(n + 1):
    result = dituI(i)
    results.append(result)
    print(f"I({i}) = {result}")

# 绘制散点图
plt.scatter(range(n + 1), results, color='blue')

# 添加标题和标签
plt.title('Scatter Plot of I(n)')
plt.xlabel('n')
plt.ylabel('I(n)')

# 显示网格
plt.grid()


# 显示图形
plt.show()
```

实验 1.4

```python
import math

# 测试用例
test = [(100, 1), (1000, 1), (5000, 1), (200, 2), (4000, 2)]


# 直接计算
def result_1(x, a):
    result1 = math.sqrt(x + a) - math.sqrt(x)
    return result1


# 有理化算法
def result_2(x, a):
    result2 = a / (math.sqrt(x + a) + math.sqrt(x))
    return result2


# 计算并比较结果
for x, a in test:
    result1 = result_1(x, a)
    result2 = result_2(x, a)
    print(f"x = {x}, a = {a}")
    print(f"直接计算结果: {result1}")
    print(f"有理化计算结果: {result2}")
```

```python
import math

# 测试用例
test = [0.01, 0.001, 0.0001, -0.001, -0.0001]


# 直接计算
def result_1(x):
    result1 = (math.atan(x) - x) / (x ** 3)
    return result1


# 泰勒展开算法
def result_2(x):
    result2 = (-1 / 3) + ((x ** 2) / 5) - ((x ** 4) / 7)
    return result2


# 计算并比较结果
for x in test:
    result1 = result_1(x)
    result2 = result_2(x)
    print(f"x = {x}")
    print(f"直接计算结果: {result1}")
    print(f"有理化计算结果: {result2}")
```

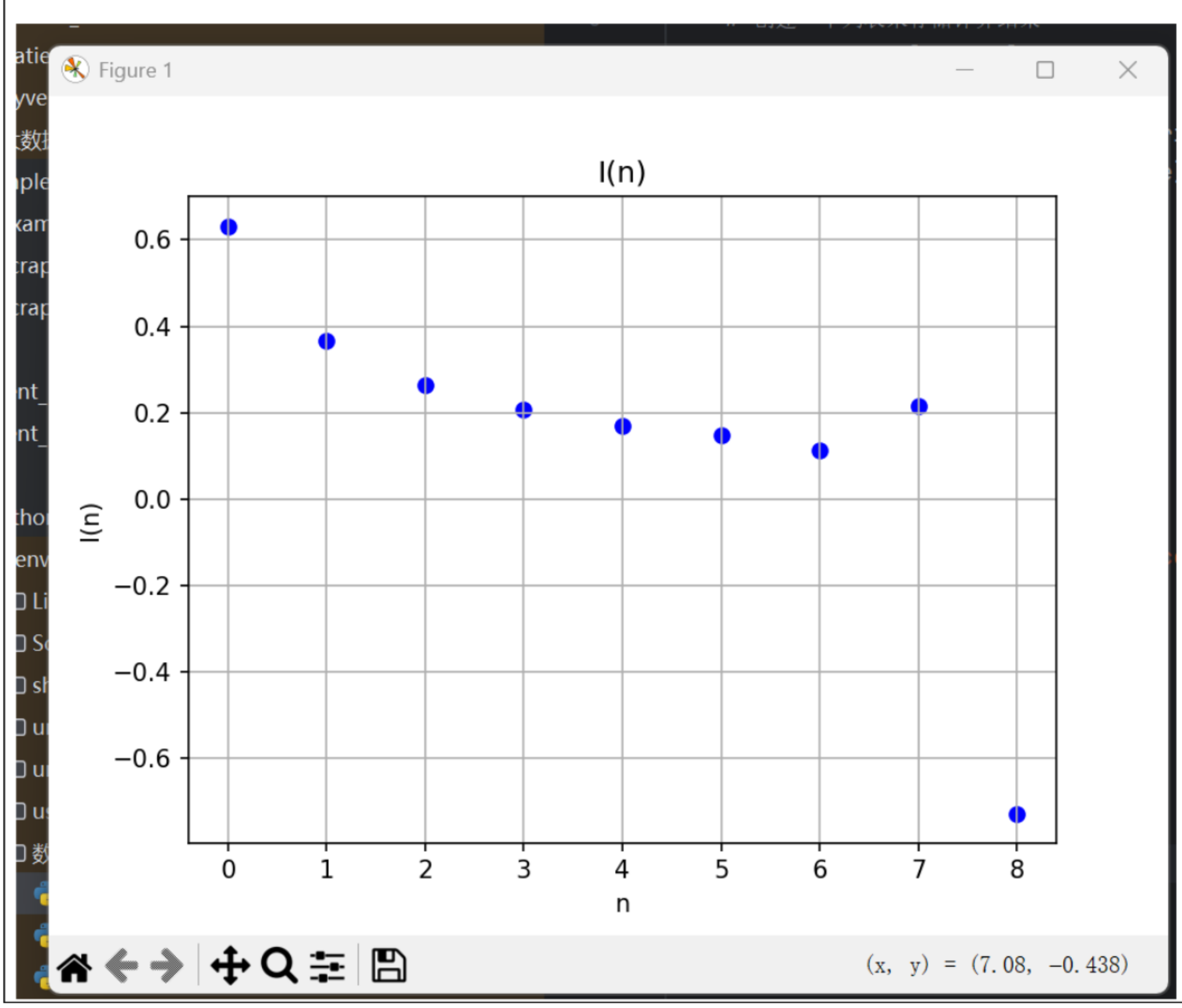
```python
import math


# 求根公式计算
def result_1(a, b, c):
    D = b ** 2 - 4 * a * c
    if D >= 0:
        x1 = (-b + math.sqrt(D)) / (2 * a)
        x2 = (-b - math.sqrt(D)) / (2 * a)
        return x1, x2
    else:
        breakpoint()


# 改进算法
def result_2(a, b, c):
    D = b ** 2 - 4 * a * c
    if D >= 0:
        if b > 0:
            x1 = (-b - math.sqrt(D)) / (2 * a)
            x2 = (c / a) / x1
        else:
            x1 = (-b + math.sqrt(D)) / (2 * a)
            x2 = (c / a) / x1
        return x1, x2
    else:
        breakpoint()


# 方程1
a = 1
b = 3
c = -2
x1, x2 = result_1(a, b, c)
x3, x4 = result_2(a, b, c)
print(f"求根公式计算结果: x1 = {x1}, x2 = {x2}")
print(f"改进算法后计算结果: x1 = {x3}, x2 = {x4}")
# 方程2
a = 1
b = -(10 ** 20)
c = 10 ** 20
x1, x2 = result_1(a, b, c)
x3, x4 = result_2(a, b, c)
print(f"求根公式计算结果: x1 = {x1}, x2 = {x2}")
print(f"改进算法后计算结果: x1 = {x3}, x2 = {x4}")
```

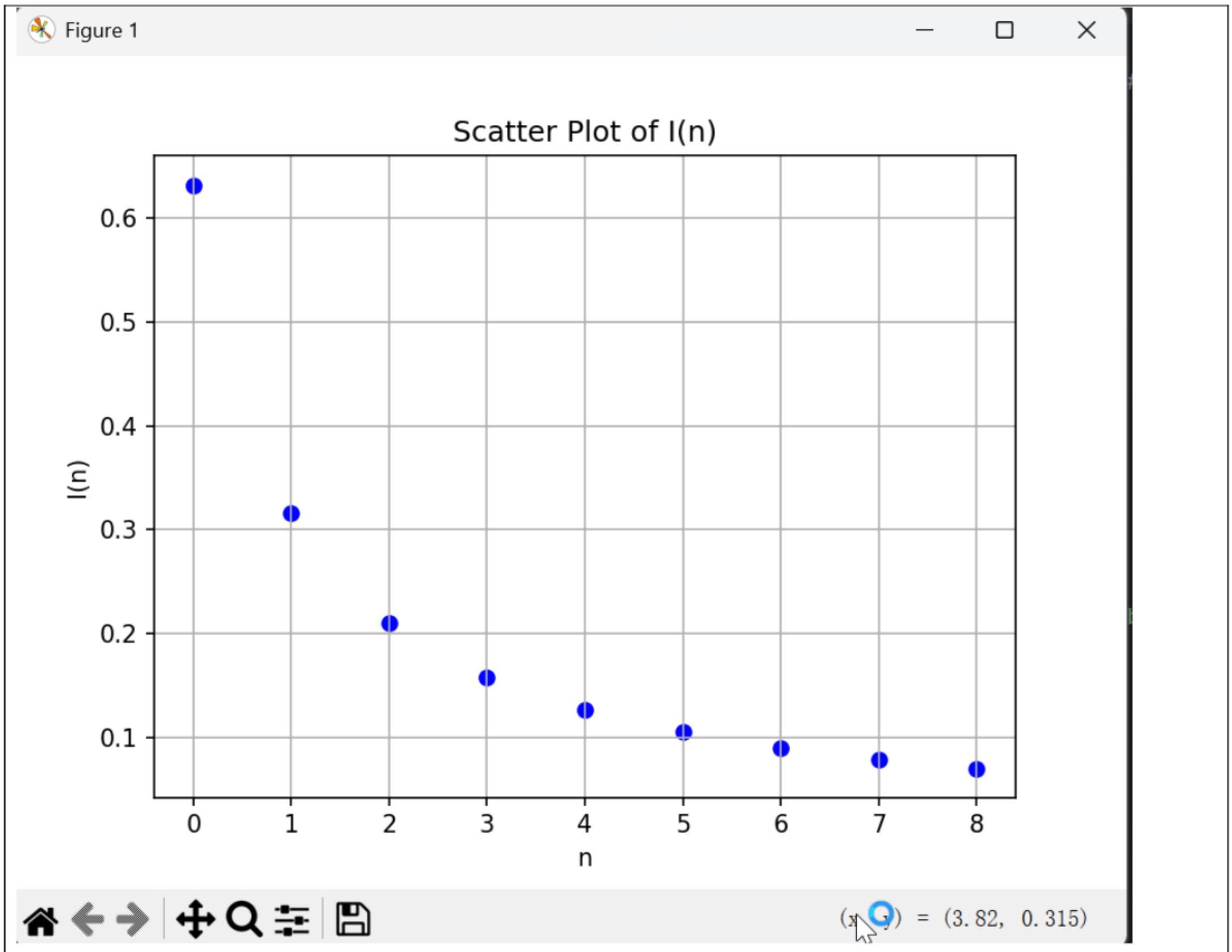
实验 2.1

```python
import math

2 usages
def dichotomy(function, a_begin, b_end, e, e2, max_itter): #输入函数，两个初始端点，绝对误差限要求，函数值要求，最大循环次数
    i = 1

    while abs(b_end-a_begin) > e and abs(function((a_begin + b_end)/2)) > e2 and i < max_itter:

        if function((a_begin + b_end)/2)*function(a_begin) > 0:
            a_begin = (a_begin + b_end)/2

        elif function((a_begin + b_end)/2)*function(a_begin) < 0:
            b_end = (a_begin + b_end)/2

        else:
            return (a_begin + b_end)/2

        i +=1

    return (a_begin + b_end)/2

#test
1 usage
def functest1(x):
    return x**3 - 7*x + 3
print("the approximately solution of function1 is" , dichotomy(functest1, a_begin: 0, b_end: 1, e: 1e-8, e2: 1e-8, max_itter: 100))

1 usage
def functest2(x):
    return 0.5 + math.cos(x) - math.log(x)
print("the approximately solution of function2 is" , dichotomy(functest2, a_begin: 1, b_end: 2, e: 1e-8, e2: 1e-8, max_itter: 100))
```

四数值结果

实验 1.3





实验 1.4（1）

```
x = 100, a = 1
直接计算结果：0.04987562112089946
有理化计算结果：0.04987562112089027
x = 1000, a = 1
直接计算结果：0.01580743742895762 7
有理化计算结果：0.01580743742895582 3
x = 5000, a = 1
直接计算结果：0.00707071429381755 9
有理化计算结果：0.00707071429382580 2
x = 200, a = 2
直接计算结果：0.07053477982094414
有理化计算结果：0.070534779820945
x = 4000, a = 2
直接计算结果：0.01580941237125444
有理化计算结果：0.01580941237125582 3
```

(2)

```
x = 0.01
直接计算结果：-0.3333133476258046
有理化计算结果：-0.3333133347619047
x = 0.001
直接计算结果：-0.3333331332310008
有理化计算结果：-0.33333313333347614
x = 0.0001
直接计算结果：-0.3333333651890806
有理化计算结果：-0.3333333313333333
x = -0.001
直接计算结果：-0.3333331332310008
有理化计算结果：-0.33333313333347614
x = -0.0001
直接计算结果：-0.3333333651890806
有理化计算结果：-0.3333333313333333
```

(3)

```
求根公式计算结果：x1 = 0.5615528128088303, x2 = -3.5615528128088303
改进算法后计算结果：x1 = -3.5615528128088303, x2 = 0.5615528128088303
求根公式计算结果：x1 = 1e+20, x2 = 0.0
改进算法后计算结果：x1 = 1e+20, x2 = 1.0
```

实验 2.1

```
/Users/horatius/Desktop/experiment/PycharmProjects/pythonProjec
the approximately solution of function1 is 0.44080771133330364
the approximately solution of function2 is 1.6004905253648758


Process finished with exit code 0
```

五．计算结果分析

实验 1.3

注意到在算法 1.1 中，I8 的值变化太大，成了负值，其误差|e(n)|=n!|e(0)|随 n 次数的增大，其误差会越来越大，不太稳定。而在 1.2 计算中，我们要确保数值的误差不会对最终结果产生显著影响。在上面的代码中，由于我们使用了直观的迭代计算方法，相对来说数值更稳定。通过绘制散点图，我们可以观察到 I(n) 随着 n 增加的变化趋势。因为 I(n) 使用的是逐步递减的形式，每个后续的值都依赖于前一个值，因此随着 n 增大，确实会观察到这些值在逐渐减小并趋于稳定接近 0。这里也反映了相应的收敛性和稳定性。

实验 1.4

（1）从结果可以看出，有理化后的表达式与直接计算的结果在数值上是相同的，但在实际计算中，

有理化后的表达式能够更好地保持数值精度，特别是在 a 相对于 x 很小时。

（2）通过比较直接计算和泰勒级数展开方法的结果，我们可以观察到泰勒级数展开方法在大多数情况下能够提供与直接计算相当或更精确的结果，尤其是在 x 接近 0 时。

（3）对于二次函数运用求根公式解题，如果不是整数根，对于计算机有一个精度，因此对于第二种这种，在运用求根公式的时候也有相近数相减的问题，我们可以运用韦达定理来解决，通用结果可以看出，这种方式的结果更加合理

实验 2.1

通过观察结果，计算结果与搜索得到的零点值一致，算法效果较好

六. 计算中出现的问题，解决方法及体会

实验 1.3

问题：在反向递推的过程中，N 的取值对真实值有影响

解决办法：N 取足够大的值，如 500，1000 等

体会：在看一个办法是否可行的时候，要从原理上去证明该方法的可行性以及有什么限制条件

实验 1.4

问题：求根公式直接计算第二个方程求解与真实值相差较大

解决办法：第一种是运用韦达定理，先求一个较准确的根，进一步得到另一个根。另一种方法是增大计算机的计算精度，从而使计算的结果和真实值接近，可以利用 Decimal, getcontext，设置精度，但在代码中我们采用了第一种更加合理的方法

体会：在运用求根公式的时候应该判断该方程是否适用

实验 2.1

问题和解决办法：1.写循环条件的时候一开始写的是 b_end-a_begin > e，因为在算法中不存在二者大小关系互换的情况；随后的过程中想要进一步加入中点处函数值的要求以提高循环结束的速度，于是加入了 function((a_begin + b_end)/2) > e2，然而输出却从 00.4408077113330364 变成了 0.5，结果精确度极速下降.随后通过打印每轮的函数值结果发现在第一次迭代时中点处的函数值为-0.375，所以需要给第二个循环条件加上 abs().2.随后数值结果准确，但小组讨论后认为不一定满足对大部分函数的普适要求，可能有些函数会使程序进入死循环．于是加入新参数 max_itter 设定最大循环次数.

体会：要满足对大部分函数的普适性比较困难，比如此算法必须要求函数内部只有一个零点，面对多零点的情况就有可能使区间内没有零点，程序进入无用循环

教师评语	指导教师：　　年　　月　　日