**北京信息科技大学实验报告**

**课程名称**  数值分析

**实验项目**  题目二

**实验仪器**  计算机

**学 院** 理学院

**专 业** 信息与计算科学

**小组成员(姓名+学号)**李子睿+2023012581

贺鸣阳+2023012588

苗含稷+2023012617

**实验日期**  2024.3.28

**成 绩**

**指导教师**  路康亚

**实验报告内容**

注：实验报告要写有实验题目、实验原理、代码、结果和分析。所有数学符号或公式都用公式编辑器或者Mathtype编辑(不能截图)。

**一、实验目的与要求（实验题目）**

1. 计算积分

(1)计算，取其较精确的近似值，利用递推公式



计算到的近似值；

(2)给出的上下界，取上下界的平均值作为的近似值，利用递推公式



计算到的近似值；

(3)分析两种计算方法的稳定性．

2. 用牛顿迭代法求解的根，使得计算精确到4位有效数字．取．

**二、实验原理（主要计算方法及公式）**

向前迭代：

输入题目给定被积函数并利用Python的算法库计算定积分的结果，之后根据题目给出的递推公式计算到的近似值并输出。

向后迭代：

输入题目给定被积函数并使用题目给出的递推公式进行反向递推，最终得出到的近似值。其中I30值我认为近似于0.006

稳定性分析：

我们分别让向前迭代和向后迭代得到的30个值与scipy为每个函数积分的值做差并画图发现向前迭代时在地13次以后偏差就会越来越大。反而如果初始值选的好像后迭代的方法可以保证I0算出的值与scipy的值差别很小，效果好于向前迭代。

牛顿法求零点：

通过线性近似逐步逼近函数的根。具体步骤为：

1. 在当前猜测点处用切线近似函数曲线
2. 找到切线与x轴的交点作为新的猜测点
3. 重复迭代直到满足精度要求

数学原理

对于方程 f(x) = 0，迭代公式为：

程序逻辑

程序每个循环迭代一次。计算当前x与f(x)和f(x)倒数的比值并更新x知道到达迭代次数或者上一次迭代与下一次的插值的绝对值小于阈值推出循环。

程序特点

不需要定义倒数，程序可以自动使用pytorch的自动微分功能计算导数值

三、**实验过程、步骤（程序代码）**

|  |
| --- |
| 1.(1)向前迭代算法 |
| import scipy  import numpy as np  import math  def func(x,n=0):      # 定义被积函数      return x\*\*(2\*n+1)\*math.exp(-x\*\*2)  def calculate\_Integral(f=func,offline=0,online=1):      # 计算积分      result, error = scipy.integrate.quad(f, offline, online)      return result,error  def forward\_iteration(k=15,I=calculate\_Integral()[0]):      # 使用递推公式计算 I\_{2n}      results = []      results.append(I)      for n in range(1, k+1):          I = -0.5 \* (math.exp(-1) - 2 \* n \* I)          results.append(I)      return results  if \_\_name\_\_ == "\_\_main\_\_":      I30=calculate\_Integral()      print(I30)      print(forward\_iteration(I30[0])) |

|  |
| --- |
| 1.(2)先后迭代 |
| def func(x,n=30):      # 定义被积函数      return x\*\*(2\*n+1)\*math.exp(-x\*\*2)  def calculate\_Integral(f=func,offline=0,online=1):      # 计算积分      result, error = scipy.integrate.quad(f, offline, online)      return result,error  def backward\_iteration(k=15,I15=0.06):      # 使用递推公式计算 I\_{2n}      results = []      results.append(I15)      I\_prev = I15      # 反向递推      for n in range(k, 0, -1):          I\_prev = math.exp(-1)/(2\*n) + I\_prev/n          results.append(I\_prev)      results.reverse()      return results  if \_\_name\_\_ == "\_\_main\_\_":      I15=calculate\_Integral()      print(I15)      print(backward\_iteration()) |

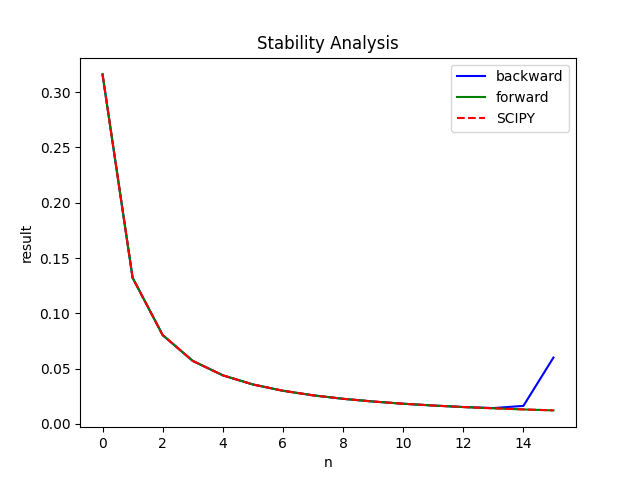
|  |
| --- |
| **稳定性分析** |
| import backward\_iteration as bi  import forward\_iteration as fi  import scipy\_calculate as sc  import numpy as np  import matplotlib.pyplot as plt  if \_\_name\_\_ == "\_\_main\_\_":      resultB=bi.backward\_iteration()      resultF=fi.forward\_iteration()      resultB=np.array(resultB)      resultF=np.array(resultF)      resultR=np.array(sc.calculate\_Integral())      stabilityB=resultB-resultR      stabilityF=resultF-resultR      x=np.arange(0,16)      plt.figure()      plt.title("stability")      plt.xlabel("n")      plt.ylabel("stability")      plt.plot(x,stabilityB,linestyle='-',color='blue',label="backward")      plt.plot(x,stabilityF,linestyle='--',color='green',label="forward")      plt.legend()      plt.show()        plt.figure()      plt.title("plotresult")      plt.xlabel("n")      plt.ylabel("result")      plt.plot(x,resultB,linestyle='-',color='blue',label="backward")      plt.plot(x,resultF,linestyle='-',color='green',label="forward")      plt.plot(x,resultR,linestyle='--',color='red',label="SCIPY")      plt.legend()      plt.show() |

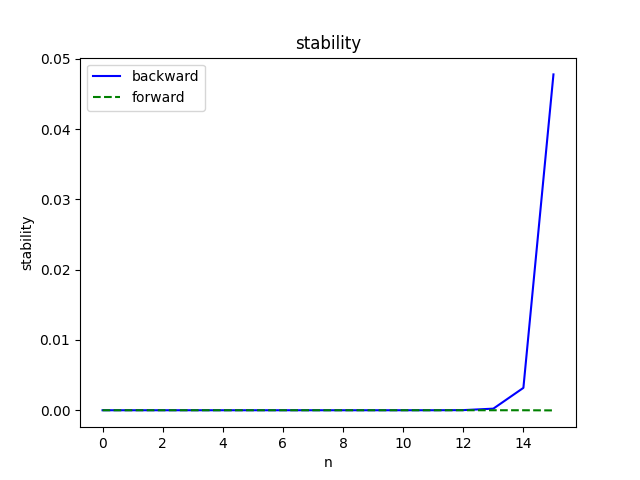
|  |
| --- |
| 牛顿法 |
| import torch  def newton\_method(f, x0=3.14, tol=1e-4, max\_iter=100):      x = torch.tensor(float(x0), requires\_grad=True)  # 确保转换为浮点数        for i in range(max\_iter):          # 清零之前的梯度          if x.grad is not None:              x.grad.zero\_()            # 计算函数值          fx = f(x)            # 反向传播计算梯度          fx.backward()            # 获取梯度          dfx = x.grad            # 检查梯度          if torch.abs(dfx) < 1e-8:  # 更安全的零值检查              raise ValueError("导数为零，无法继续迭代")            # 更新x值（需要脱离计算图）          with torch.no\_grad():              x\_new = x - fx / dfx              delta = torch.abs(x\_new - x)                # 更新x值并保持requires\_grad=True              x.copy\_(x\_new)            # 打印迭代信息          print(f"Iter {i+1}: x = {x.item():.6f}, f(x) = {fx.item():.6f}, f'(x) = {dfx.item():.6f}")            if delta < tol:              print("\n达到精度要求，迭代终止")              return x.item()  # 返回Python浮点数        print("\n达到最大迭代次数，可能未收敛")      return x.item()  # 定义函数  def f(x):      return 12 - 3\*x + 2\*torch.cos(x)  if \_\_name\_\_ == "\_\_main\_\_":      root = newton\_method(f, 2.0)  # 使用更好的初始值2.0      print(f"\n最终结果: {root:.8f}") |

1. **实验结果**

|  |  |
| --- | --- |
| 1.(1)向前迭代 | |
| n=0 | 0.31606027941427883 |
| n=1 | 0.13212055882855767 |
| n=2 | 0.08030139707139416 |
| n=3 | 0.05696447062846133 |
| n=4 | 0.043918161928124144 |
| n=5 | 0.035651089054899554 |
| n=6 | 0.029966813743676157 |
| n=7 | 0.025827975620011934 |
| n=8 | 0.022684084374374303 |
| n=9 | 0.020217038783647556 |
| n=10 | 0.018230667250754395 |
| n=11 | 0.016597619172577183 |
| n=12 | 0.015231709485205025 |
| n=13 | 0.014072502721944158 |
| n=14 | 0.013075317521497043 |
| n=15 | 0.012190042236734477 |

|  |  |
| --- | --- |
| 1.(2)向后迭代 | |
| n=1 | 0.31606027941431536 |
| n=2 | 0.13212055882859422 |
| n=3 | 0.08030139707146729 |
| n=4 | 0.056964470628680694 |
| n=5 | 0.04391816192900161 |
| n=6 | 0.035651089059286885 |
| n=7 | 0.029966813770000118 |
| n=8 | 0.025827975804279663 |
| n=9 | 0.02268408584851614 |
| n=10 | 0.020217052050924104 |
| n=11 | 0.01823079992351986 |
| n=12 | 0.016599078572997306 |
| n=13 | 0.015249222290246514 |
| n=14 | 0.014300169187483518 |
| n=15 | 0.01626264803904808 |
| n=16 | 0.06 |

****



**五、结果分析**

**六、小组成员分工**

李子睿：代码部分

贺鸣阳：向前迭代，向后迭代的报告

苗含稷：牛顿法部分的报告

分工比例1：1：1

（写清楚每位成员负责哪一部分内容，并给出工作比例，例如三人小组给定比例为1：0.9：0.95，若教师给定得分为98，则最终成员的成绩为98，98\*0.9，98\*0.95）

注：实验报告电子版提交至学习通