**第1次上机实验报告**

**学号：3123007606 班级：计算机科学与技术(2)班 姓名：黄炳城**

目录

[实验1：减少运算次数的实验结果分析 1](#_Toc128511668)

[实验2：求解非线性方程的二分法实现 2](#_Toc128511669)

# 实验1：减少运算次数的实验结果分析

【实验目的】比较不同算法求多项式的运算次数与用时。

设计2种不同的算法计算以下函数的值，分别测试*x* = 0.1，1，2，10。

 (1)

【实验条件】

计算机配置：

CPU：Intel(R)Core(TM)i7-8550U CPU@1.80GHz

内存大小：16.0GB

操作系统：Windows11专业版

【算法介绍】

算法1：直接法

暴力计算，通过逐项计算多项式的每一项，最后将所有项相加。

算法2：

将多项式转换为嵌套形式，避免重复计算高次幂。

【实验结果及分析】

表1 算法比较结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *x* | 算法 | 函数结果*f* | 乘法次数 | 加法次数 | 用时(秒) |
| 0.1 | 算法1 | 1.034567901234568 | 100000 | 200000 | 0.0632939338684082 |
| 算法2 | 1.234567901234568 | 100000 | 100000 | 0.04228377342224121 |
| 1 | 算法1 | 5000250001 | 100000 | 200000 | 0.06794452667236328 |
| 算法2 | 5000150001 | 100000 | 100000 | 0.03687119483947754 |
| 2 | 算法1 | Len(30109) | 100000 | 200000 | 13.160678386688232 |
| 算法2 | Len(30109) | 100000 | 100000 | 0.3622548580169678 |
| 10 | 算法1 | Len(100006) | 100000 | 200000 | 225.44058465957642 |
| 算法2 | Len(100006) | 100000 | 100000 | 1.0834879875183105 |

结果讨论：

秦九韶算法通过减少运算步骤和避免重复计算，显著提升了效率。即使乘法次数相同，秦九韶算法的加法次数减半且操作更高效，使其运行时间更短。

【算法1源代码】

|  |
| --- |
| 把源代码贴于此处  import sys  sys.set\_int\_max\_str\_digits(0) # 解除整数转换的位数限制  from time import \*  startT = time()  x = 10  f = 1  countMul = 0  countAdd = 0  for i in range(1, 100001):  f += (i + 2) \* x \*\* (i + 1)  countMul += 1  countAdd += 2  endT = time()  print("time:", endT - startT)  print("result:", f)  print("乘法次数", countMul)  print("加法次数", countAdd) |

【算法2源代码】

|  |
| --- |
| 把源代码贴于此处  #算法二：秦九韶算法  from time import \* #引入时间库  startT = time() #记录起始时间  x = 10  powN = 100000  aN = powN + 1  countMul = 0  countAdd = 0  S = aN  for i in range(powN, 0, -1):  S = i + x \* S  countMul += 1  countAdd += 1  endT = time()  print("time:",endT-startT)  print("result:",S)  print("乘法次数",countMul)  print("加法次数",countAdd |

# 实验2：求解非线性方程的二分法实现

【实验目的】

验证二分法的可靠性和稳定性，观察浮点数精度对数值计算的影响。

【实验结果及分析】

表2 二分法求解过程数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | 下限*x*low | 上限*x*up | (*x*up + *x*low)/2 | *f*((*x*up + *x*low)/2)  的正负性 |
| 0 | 1.3 | 1.5 | 1.4 | < 0 |
| 1 | 1.4 | 1.5 | 1.4 | <0 |
| 2 | 1.4 | 1.45 | 1.45 | >0 |
| 3 | 1.4 | 1.425 | 1.425 | >0 |
| 4 | 1.4125 | 1.425 | 1.4125 | <0 |
| 5 | 1.4125 | 1.41875 | 1.41875 | >0 |
| 6 | 1.4125 | 1.415625 | 1.415625 | >0 |
| 7 | 1.4140625 | 1.415625 | 1.4140625 | <0 |
| 8 | 1.4140625 | 1.41484375 | 1.41484375 | >0 |
| 9 | 1.4140625 | 1.414453125 | 1.414453125 | >0 |
| 10 | 1.4140625 | 1.4142578125 | 1.4142578125 | >0 |
| 11 | 1.41416015625 | 1.4142578125 | 1.41416015625 | <0 |
| 12 | 1.414208984375 | 1.4142578125 | 1.414208984375 | <0 |
| 13 | 1.414208984375 | 1.4142333984375 | 1.4142333984375 | >0 |
| 14 | 1.414208984375 | 1.41422119140625 | 1.41422119140625 | >0 |
| 15 | 1.414208984375 | 1.41421508789062 | 1.41421508789062 | >0 |
| 16 | 1.41421203613281 | 1.41421508789062 | 1.41421203613281 | <0 |
| 17 | 1.41421356201172 | 1.41421508789062 | 1.41421356201172 | <0 |

结果讨论：

通过31次迭代成功将误差控制在10-11量级，验证了二分法的可靠性和稳定性，同时揭示了浮点数精度对数值计算的影响。优化后的代码可进一步提升效率和鲁棒性。

【算法源代码】

|  |
| --- |
| 把源代码贴于此处  #二分法解方程程序  a = 2 #f(x) = x\*x -a  LIMIT = 1e-20 #终止条件  #方程函数f(x)的定义  def f(x):  return x\*x - a  #f()函数结束  #二分法求解方程  #初始设置  xlow = float(input("请输入x值下限"))  xup = float(input("请输入x值上限"))  #循环处理  iter = 0 #迭代计算  while (xup - xlow)\*(xup - xlow) > LIMIT:  xmid = (xlow + xup) / 2  iter += 1  if f(xmid) == 0:  break  elif f(xmid) > 0:  xup = xmid  else:  xlow = xmid  print("{:.15g} {:.15g} {:.15g} {:.15g} {:.15g}".format(iter,xlow,xup,xmid,f(xmid)/2)) |

附加题：对于大量的输出数据，有什么便捷的方法把列表数据输出，方便贴到word文档呢？或者直接把数据矩阵输出到.txt文件中呢？