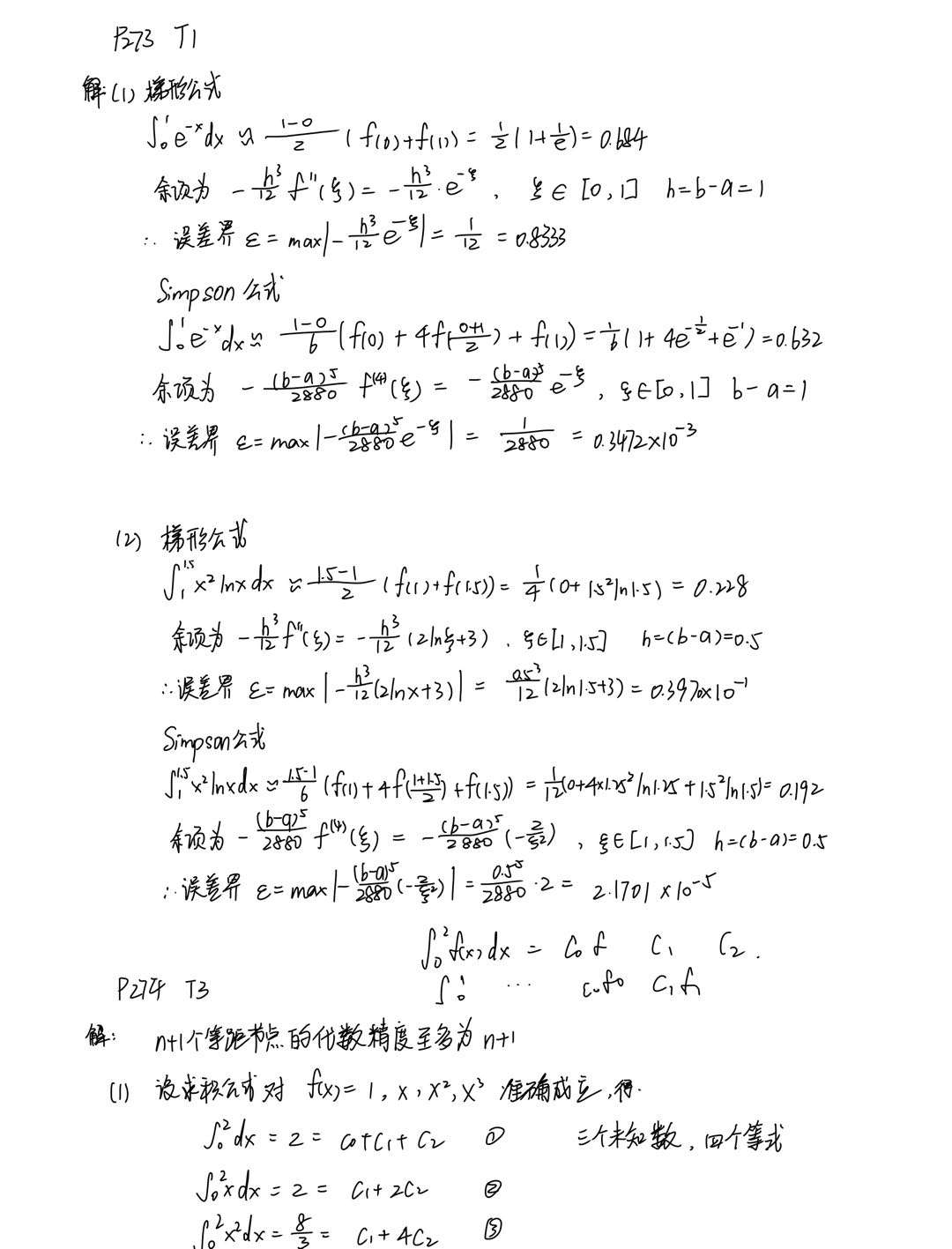
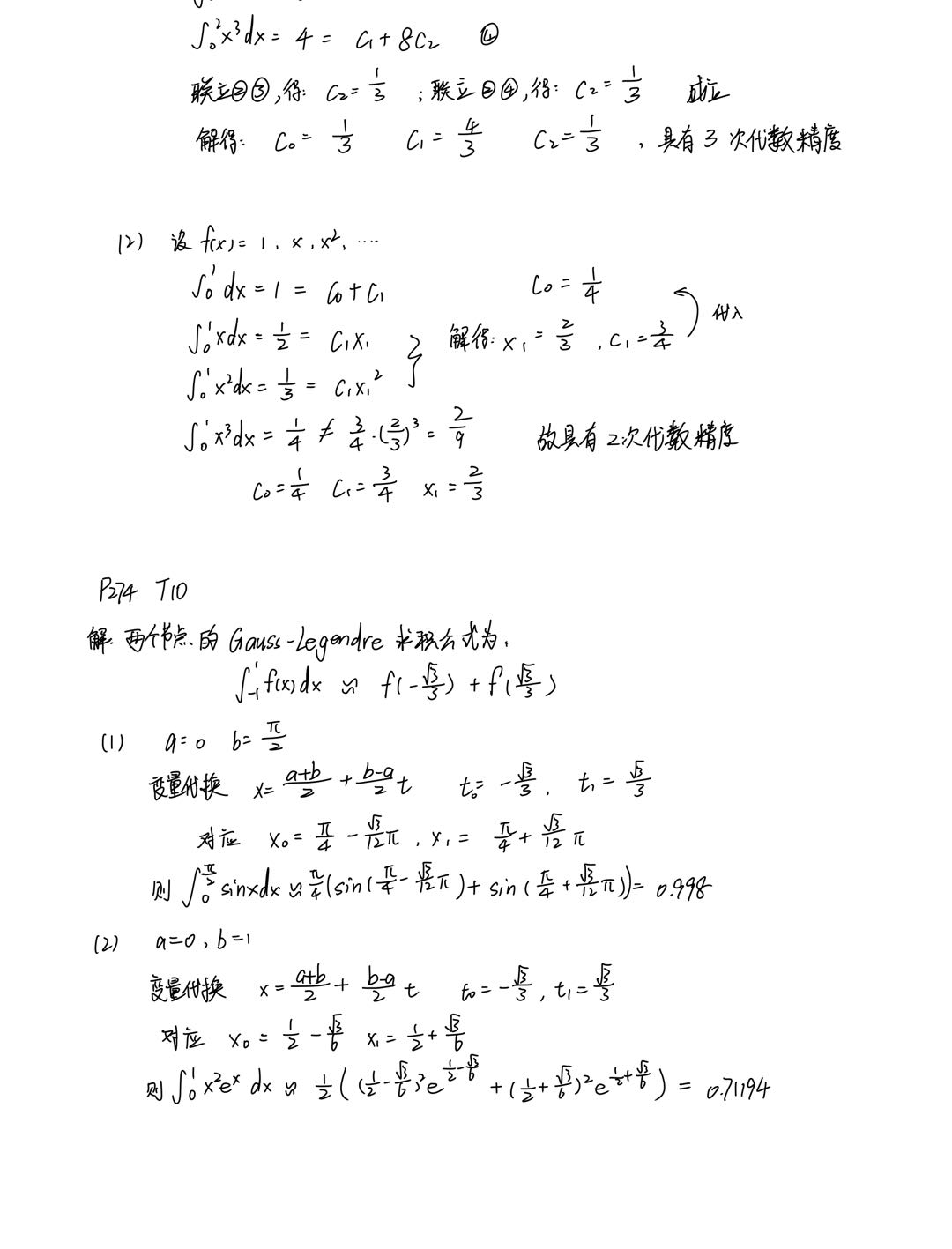
第八次作业

赖显松 2021214726

1 书面作业题目





2 编程题目

2.1 1203课堂任务1

**生成任意阶次Cotes系数表的函数：**

按照Cotes系数公式，得到对应的Cotes系数的函数（输入：n阶次， 第k个系数；输出：对应的Cotes系数C(n)k）。

求解步骤为：根据公式，其中较难求的是多项式连乘加上积分，这个其实也很好处理，用上一次作业中提到的，多项式展开其实就是一维卷积；另外，展开后的多项式积分也很容易，n次方积分就是n+1次方除以n。

代码如下：

1. **def** cotesInteg(n, k):
2. ori\_papa = np.zeros((n+3,)) # 经过padding
3. temp\_para = ori\_papa.copy()
4. inte\_para = np.zeros((n+2,))
5. ori\_papa[1] = 1
6. **for** j **in** range(0, n+1):
7. **if** j != k:
8. kenel = np.array([1, -j])
9. **for** i **in** range(n+2):
10. temp\_para[i+1] = np.sum(ori\_papa[i:i+2] \* kenel)
11. ori\_papa = temp\_para.copy()
12. inte\_para[0] = 0
13. **for** i **in** range(1, n+2):
14. inte\_para[i] = ori\_papa[i]/i
15. **return** s(n, inte\_para)

**求解不同阶次的Cotes求积：**

首先初始化，生成Cotes系数表，通过循环采用不同阶次（1至8）的Cotes求积公式计算对应的积分。

代码如下：

1. # 生成cotes系数表
2. m, n = 9, 9
3. table = np.zeros((m, n))
4. **for** i **in** range(1, m):
5. **for** j **in** range(0, i+1):
6. table[i, j] = integ.cotes(i, j)
8. a, b = 1, 2
10. **def** func(x):
11. f = np.log(x)
12. # f = (10/x)\*\*2 \* np.sin(10/x)
13. **return** f
15. **for** n **in** range(1, 9):
16. x = np.linspace(a, b, n+1, endpoint=True)
17. fx = x.copy()
18. **for** i **in** range(len(x)):
19. fx[i] = func(x[i])
20. inteCotes = 0
21. **for** k **in** range(n+1):
22. inteCotes += table[n, k] \* fx[k]
23. **print**(n,"阶积分结果：",inteCotes)

**结果：**

第一个积分公式的积分结果（图2. 1）：

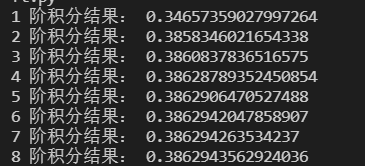


图2. 1 第一个积分式不同阶次的Cotes求积公式积分结果

积分的精确结果为0.38629，可以看出，积分的结果精度随阶数增加升高，并且5阶时已实现小数点后5位相同。

同理对第二个积分式子改动，得到结果（图2. 2）：

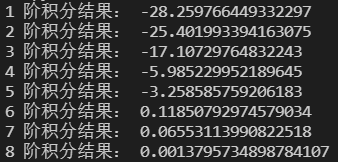


图2. 2 第二个积分式不同阶次的Cotes求积公式积分结果

积分的精确结果为1.426，Cotes积分结果很不理想，没有接近精确解的结果。是因为积分原函数的特殊性，以及高阶的cotes系数表出现了负值的系数，导致数值不稳定的原因。

2.2 1203课堂任务2

复合梯形求积公式：复合梯形公式通过在相邻节点处使用梯形积分公式，随着节点数的增多，精度提高。当节点趋于无穷，误差理论为0。

1. **for** n **in** range(2, 5000):
2. x = np.linspace(a, b, n+1, endpoint=True)
3. fx = x.copy()
4. **for** i **in** range(len(x)):
5. fx[i] = func(x[i])
7. h = (b - a) / n
8. # 复合梯形求积公式
9. cs = func(a) + func(b)
10. **for** k **in** range(1, n):
11. cs += 2 \* func(x[k])
12. cs \*= h/2
13. **print**(n,"节点梯形积分结果：", cs)
14. **if** np.abs(cs + 1.42602475634) < 1e-4:
15. **print**("到达精度的步长为：", h)
16. **break**

第二个积分式复合梯形公式积分结果如图2. 3所示。

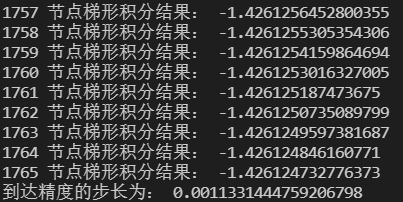


图2. 3 第二个积分式复合梯形公式积分结果

复合梯形公式到达10-4精度时，节点的个数为1765，步长h为0.001133144…。

同理采用复合Simpson公式：

1. sinp = func(a) + func(b)
2. **for** k **in** range(1, n):
3. sinp += 4 \* func((x[k-1] + x[k])/2) + 2 \* func(x[k])
4. sinp += 4 \* func((x[k-1] + x[k])/2)
5. sinp \*= h/6
6. **print**(n,"节点复合Simpson积分结果：", sinp)
7. **if** np.abs(sinp + 1.42602475634) < 1e-4:
8. **print**("到达精度的步长为：", h)
9. **break**

结果（图2. 4）：

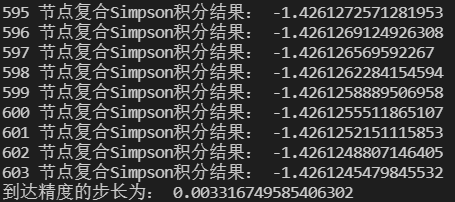


图2. 4 第二个积分式复合Simpson公式积分结果

复合Simpson公式在603个节点的时候就达到了10-4精度，步长h为0.331675…。显然在相同节点个数的情况下，复合Simpson公式的精度要比复合梯形公式高。

2.3 计算实习题——Gauss复合求积：

**Gauss-Chebyshev求积函数：**

（输入：n阶次，区间左a，区间右b，积分原函数func；输出：积分结果）

根据公式，没有特别的地方，代码如下：

1. **def** gaussCheb(n, a, b, func):
2. t = np.zeros((n+1, ))
3. A = np.ones((n+1,)) \* np.pi/(n+1)
4. **for** k **in** range(n+1):
5. t[k] = np.cos((2\*k+1)\*np.pi/2/(n+1))
6. I = 0
7. **for** k **in** range(n+1):
8. I += np.sqrt(1-t[k]\*\*2) \* func((a+b)/2 + (b-a)/2\*t[k])
9. **return** I \* (b-a)/2\*np.pi/(n+1)

**复合Gauss-Chebyshev求积函数：**

（输入：m个区间，n阶次，区间左a，区间右b，积分原函数func；输出：积分结果）

复合Gauss-Chebyshev求积函数与原来相比并没有很明显的不同，思路只是将原来的积分区间分割为若干等分，对每一等分分别进行Gauss-Chebyshev求积，最后累加起来，代码如下：

1. **def** combGaussCheb(m, n, a, b, func):
2. x = np.linspace(a, b, m+1, endpoint=True)
3. I\_plus = 0
4. **for** k **in** range(m):
5. I\_plus += gaussCheb(n, x[k], x[k+1], func)
6. **return** I\_plus

选取求积公式的阶次为10，节点数增加的同时，积分结果的变化很小，1000个节点后仍未达到10-4的精度；选取更高的阶次：100，在18个节点的时候就达到了精度要求（图2. 5）。可见复合积分公式阶次和区间数选取恰当，可以提高求解的精度。换不同的阶次，找到对应的节点个数，画出节点个数随阶次的变化曲线（图2. 6）：

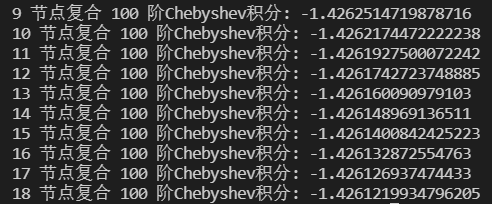


图2. 5 正交多项式对应的线性关多项式系数对比

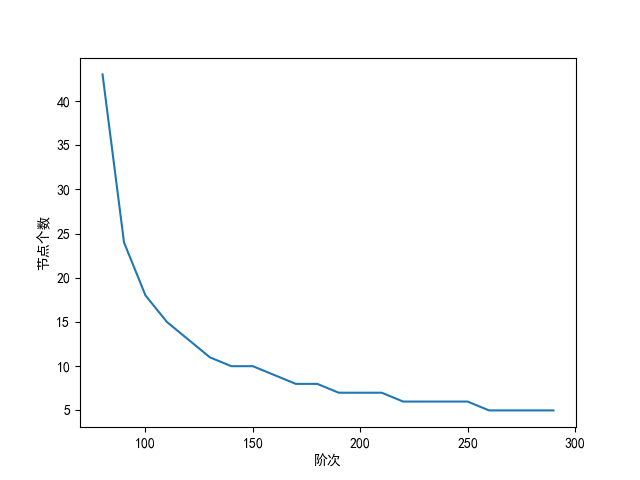


图2. 6 复合Gauss-Chepshev求积公式10-4精度节点随阶次变化曲线

可以发现，在阶次为140左右，节点个数的变化趋于平缓，因此，节点个数为10，阶次为140左右的复合Gauss-Chepshev求积是较合适的选择。

2.3 计算实习题——Romberg求积：

Romberg求积需要用到复合梯形公式，将上述复梯形公式的代码集成为函数（输入：n段，左区间a，右区间b，积分原函数func；输出：求积结果cs）：

1. **def** combTrap(n, a, b, func):
2. h = (b - a) / n
3. x = np.linspace(a, b, n+1, endpoint=True)
4. cs = func(a) + func(b)
5. **for** k **in** range(1, n):
6. cs += 2 \* func(x[k])
7. **return** cs \* h/2

Romberg求积函数（输入：p层，左区间a，右区间b，积分原函数func；输出：求积结果表格T（p, p））：

求解积分，得到在外推公式7层达到10-4精度（图2. 7）。

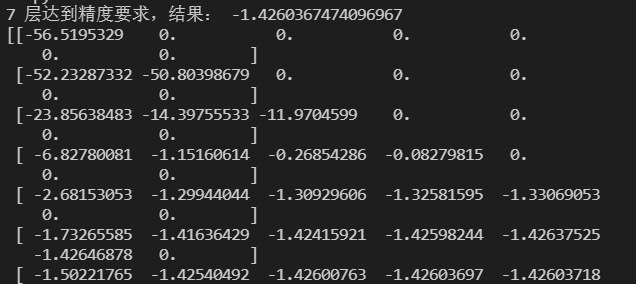


图2. 7 Romberg外推求积结果

参考文献

1. 关治．数值方法[M]．北京：清华大学出版社，2006：66-92．

附录

见附件python源代码。