计算物理第三次上机作业

白博臣 2022141220036

## Problem 1

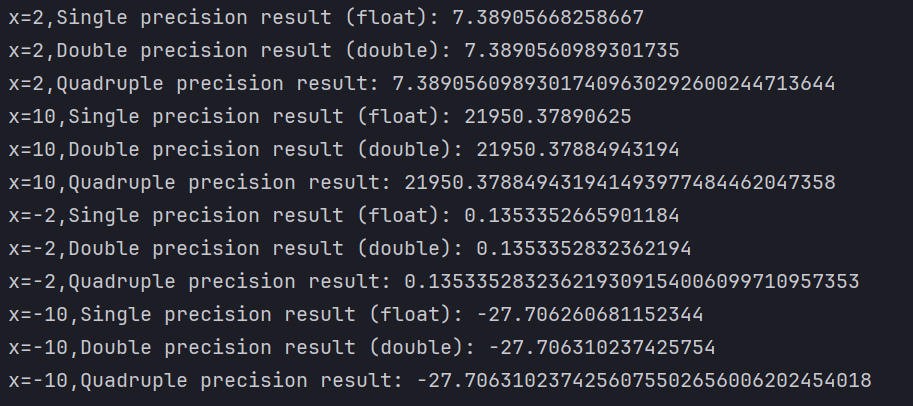
### (a)

在单精度（float）和双精度（double）下，可以使用 Python 中的 NumPy 库来进行计算。使用四倍精度（quadruple precision）进行计算可能需要使用第三方库，因为 Python 的内置浮点数类型（float、double）不支持四倍精度。一个常用的第三方库是 mpmath，它提供了多精度计算功能。

具体代码如下：

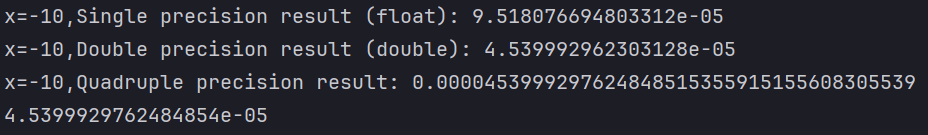
1. import numpy as np
2. from mpmath import mp
3. def taylor\_exp\_single(x, n\_terms=10):
4. result\_single = np.float32(1.0)
5. term\_single = np.float32(1.0)
6. for i in range(1, n\_terms):
7. term\_single = term\_single \* x / i
8. result\_single = np.float32(result\_single + term\_single)
9. return result\_single
10. def taylor\_exp\_double(x, n\_terms=10):
11. result\_double = np.float64(1.0)
12. term\_double = np.float64(1.0)
13. for i in range(1, n\_terms):
14. term\_double = term\_double \* x / i
15. result\_double = np.float64(result\_double + term\_double)
16. return result\_double
17. def taylor\_exp\_quad(x, n\_terms=10):
18. mp.dps = 36
19. result\_quad = mp.mpf('1.0')
20. term\_quad = mp.mpf('1.0')
21. for i in range(1, n\_terms):
22. term\_quad = term\_quad \* x / i
23. result\_quad = result\_quad + term\_quad
24. return result\_quad
25. x = 2.0
26. n\_terms = 20
27. result\_single = taylor\_exp\_single(x, n\_terms)
28. result\_double = taylor\_exp\_double(x, n\_terms)
29. result\_quad = taylor\_exp\_quad(mp.mpf(str(x)), n\_terms)
30. print(f"x=2,Single precision result (float): {result\_single}")
31. print(f"x=2,Double precision result (double): {result\_double}")
32. print(f"x=2,Quadruple precision result: {result\_quad}")
33. x = 10.0
34. n\_terms = 20
35. result\_single = taylor\_exp\_single(x, n\_terms)
36. result\_double = taylor\_exp\_double(x, n\_terms)
37. result\_quad = taylor\_exp\_quad(mp.mpf(str(x)), n\_terms)
38. print(f"x=10,Single precision result (float): {result\_single}")
39. print(f"x=10,Double precision result (double): {result\_double}")
40. print(f"x=10,Quadruple precision result: {result\_quad}")
41. x = -2.0
42. n\_terms = 20
43. result\_single = taylor\_exp\_single(x, n\_terms)
44. result\_double = taylor\_exp\_double(x, n\_terms)
45. result\_quad = taylor\_exp\_quad(mp.mpf(str(x)), n\_terms)
46. print(f"x=-2,Single precision result (float): {result\_single}")
47. print(f"x=-2,Double precision result (double): {result\_double}")
48. print(f"x=-2,Quadruple precision result: {result\_quad}")
49. x = -10.0
50. n\_terms = 20
51. result\_single = taylor\_exp\_single(x, n\_terms)
52. result\_double = taylor\_exp\_double(x, n\_terms)
53. result\_quad = taylor\_exp\_quad(mp.mpf(str(x)), n\_terms)
54. print(f"x=-10,Single precision result (float): {result\_single}")
55. print(f"x=-10,Double precision result (double): {result\_double}")
56. print(f"x=-10,Quadruple precision result: {result\_quad}")

运算结果如下：



图表 1 Problem1的运算结果

当时，由于级数项数太少（），造成了较大的误差，我们可以适当提高的取值，我们发现得到的结果与理论值更加接近。



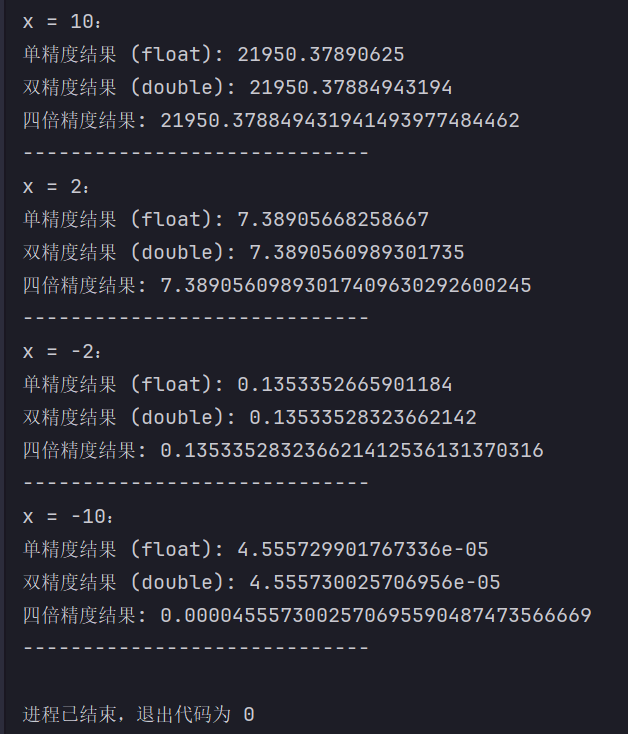
图表 2 增加的取值后的结果

### （b）

我们使用if条件语句进行条件判断，针对不同的正负数应用不同的计算方法，代码如下：

1. import numpy as np
2. from mpmath import mp
3. def taylor\_exp\_single(x, n\_terms=10):
4. if x >= 0:
5. result = np.float32(1.0)
6. term = np.float32(1.0)
7. for i in range(1, n\_terms):
8. term = term \* x / i
9. result = np.float32(result + term)
10. else:
11. result = np.float32(1.0) / taylor\_exp\_single(-x, n\_terms)
12. return result
13. def taylor\_exp\_double(x, n\_terms=10):
14. if x >= 0:
15. result = np.float64(1.0)
16. term = np.float64(1.0)
17. for i in range(1, n\_terms):
18. term = term \* x / i
19. result = np.float64(result + term)
20. else:
21. result = np.float64(1.0) / taylor\_exp\_double(-x, n\_terms)
22. return result
23. def taylor\_exp\_quad(x, n\_terms=10):
24. if x >= 0:
25. mp.dps = 30
26. result = mp.mpf('1.0')
27. term = mp.mpf('1.0')
28. for i in range(1, n\_terms):
29. term = term \* x / i
30. result = result + term
31. else:
32. result = 1 / taylor\_exp\_quad(mp.mpf(str(-x)), n\_terms)
33. return result
34. test\_values = [10, 2, -2, -10]
35. n\_terms = 20
36. for x in test\_values:
37. result\_single = taylor\_exp\_single(x, n\_terms)
38. result\_double = taylor\_exp\_double(x, n\_terms)
39. result\_quad = taylor\_exp\_quad(mp.mpf(str(x)), n\_terms)
40. print(f"x = {x}：")
41. print(f"单精度结果 (float): {result\_single}")
42. print(f"双精度结果 (double): {result\_double}")
43. print(f"四倍精度结果: {result\_quad}")
44. print("-----------------------------")

得到如下的运算结果：



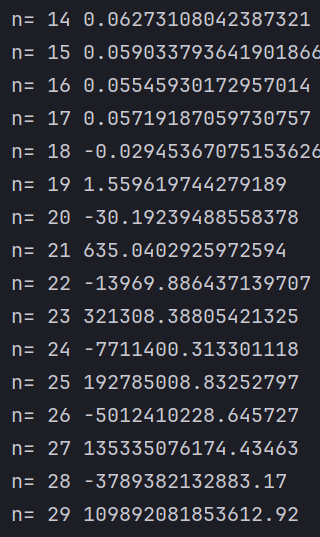
图表 3 改进后的输出结果

针对单精度、双精度、四倍精度的运算，我们发现越高精度的计算最后计算出的结果越接近理论值，原因是在进行级数累加的时候，越高精度意味这更多的正确位数，也就意味着更精确的数值。

## Problem 2

首先使用python编写这个循环语句：

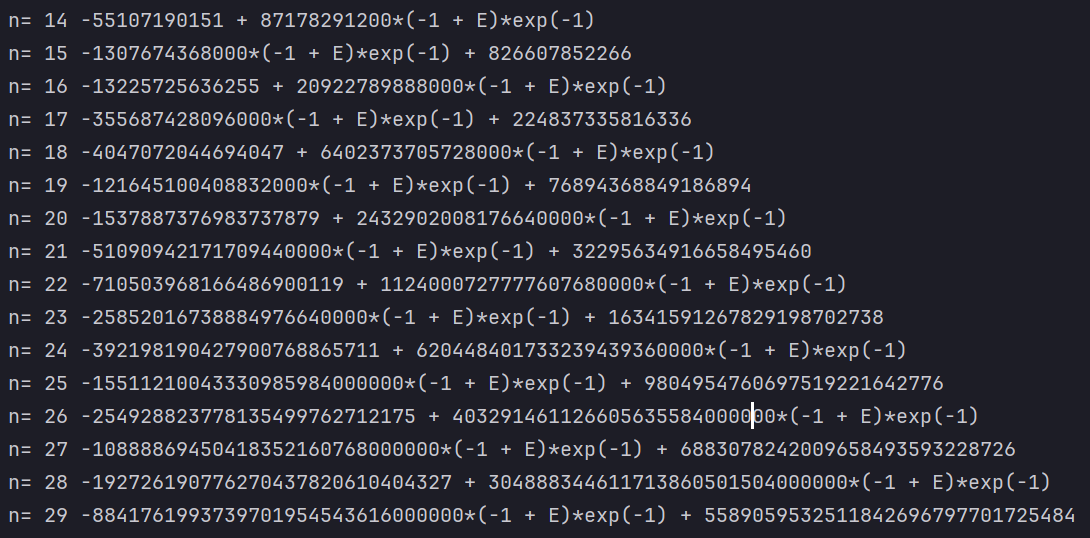
1. import numpy as np
2. def cal(n):
3. a\_0 = 1 / np.exp(1) \* (np.exp(1) - 1)
4. for i in range(1, n+1):
5. a\_n = 1 - (i) \* a\_0
6. a\_0 = a\_n
7. return a\_n
8. *# n = eval(input('n='))*
9. *# value = cal(n)*
10. *# print(value)*
11. for i in range(2, 30):
12. print('n=', i, cal(i))



图表 4 部分输出结果

我们发现在的时候，其算数误差越来越大，甚至当时，其结果逐渐发散，而不是趋近于0.

我们利用符号计算库sympy进行计算：



图表 5 利用sympy计算出的结果

我们选取在之前计算中差距较大的项，再次计算：

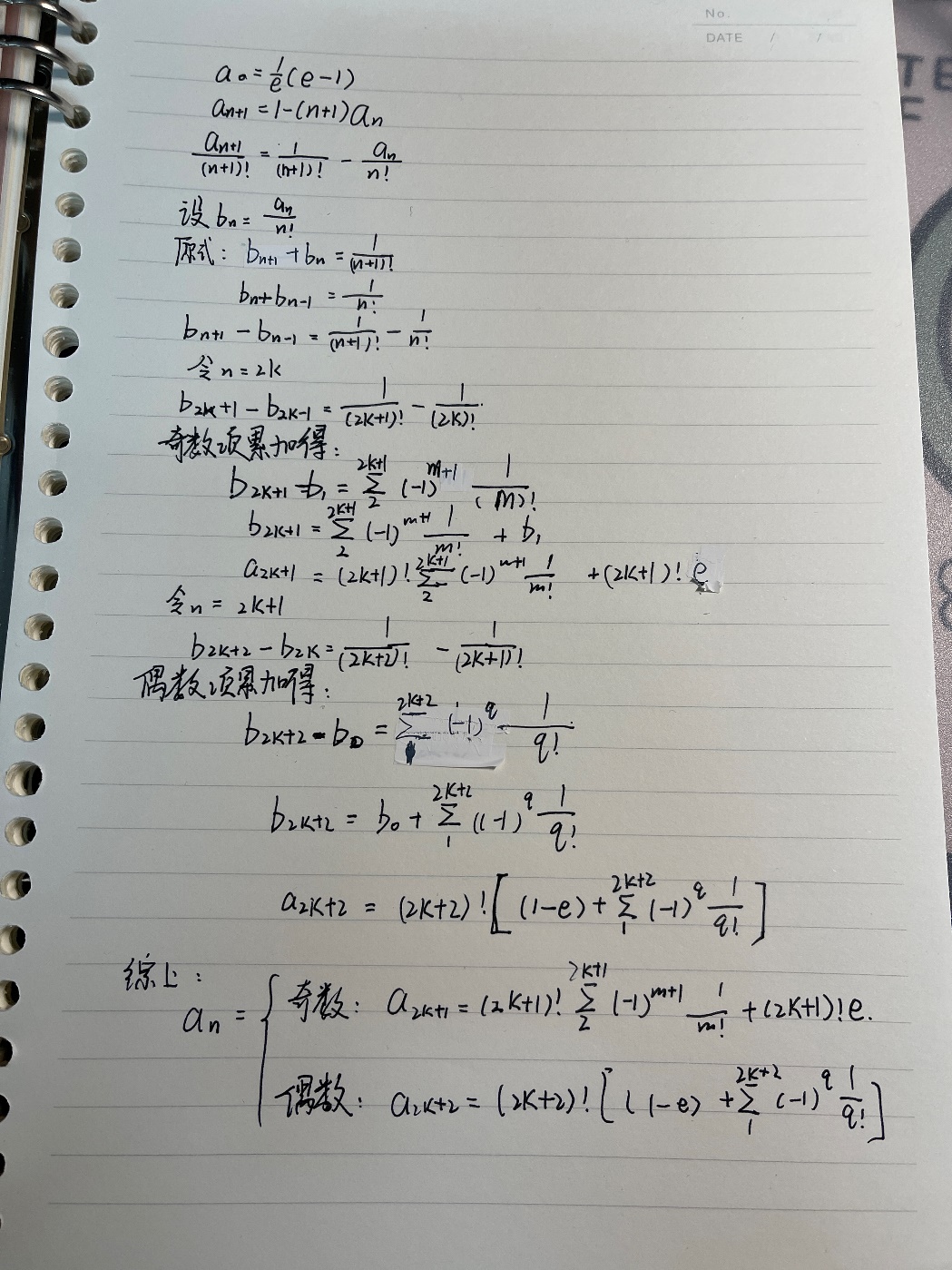
1. import numpy as np
2. a=-8841761993739701954543616000000\*(-1 + np.exp(1))\*np.exp(-1) + 5589059532511842696797701725484
3. b=-455 + 720\*(-1 + np.exp(1))\*np.exp(-1)
4. c=-4047072044694047 + 6402373705728000\*(-1 + np.exp(1))\*np.exp(-1)
5. print(a,b,c)



图表 6 再次计算某些项的数值

我们发现与我们通过循环语句写出来的结果不符合，所以可以推断出应该是在计算循环的过程中，随着某些项的位数逐渐增多，原本的精度已经不支持计算，从而导致了数值计算不准确。

同样我们也可以推导出该数列的通项公式：



得出来的通项我们发现后面恰好是泰勒展开式，所以可以得出，该数列在趋近于无穷的时候应该趋近于0.