# 计 算 方 法

实验指导与实验报告

姓名 高 昊 达

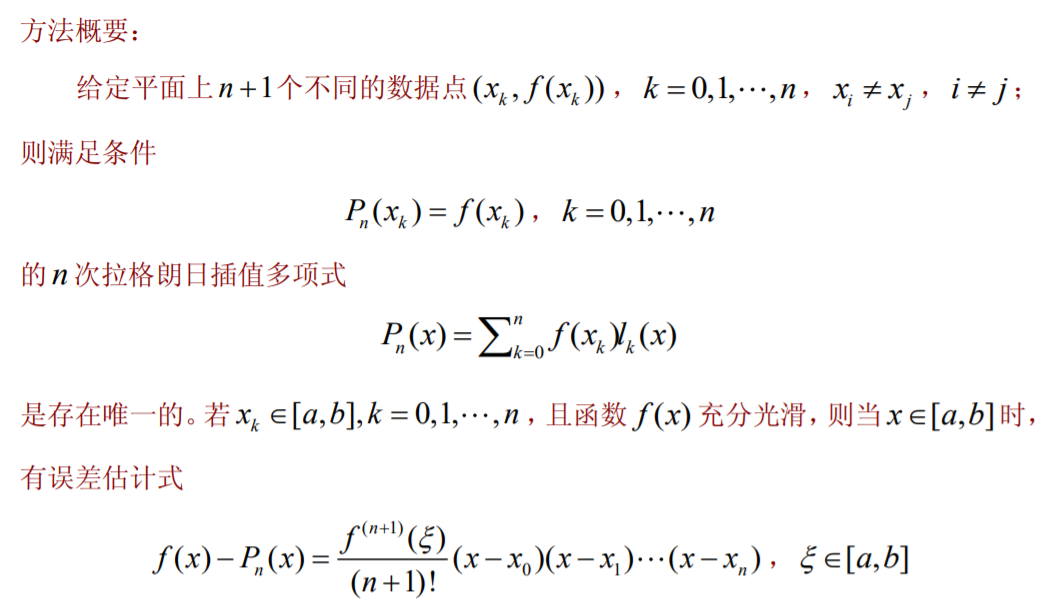
学号 1173710206

院系 计算机科学与技术学院

专业 软 件 工 程

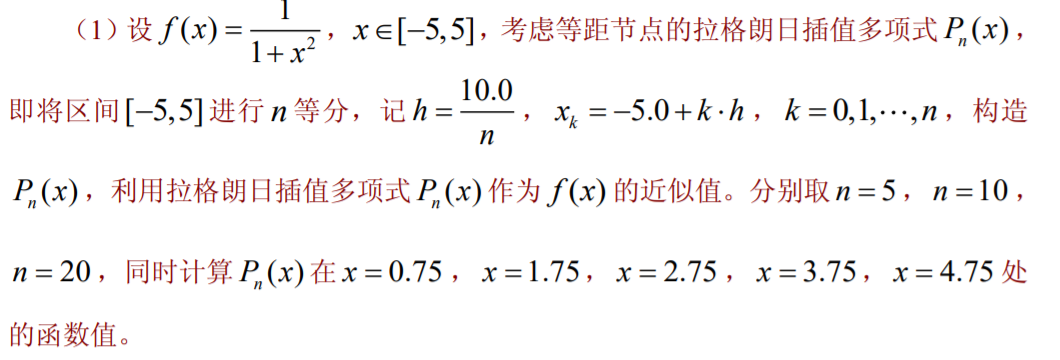
哈尔滨工业大学

## 拉格朗日插值



程序实现（Python）：

n = int(file.readline())  
x1 = []  
y1 = []  
for i in range(0, n + 1):  
 x1.append(0)  
 y1.append(0)  
 x1S, y1S = file.readline().split()  
 x1[i] = float(x1S)  
 y1[i] = float(y1S)  
  
xS = file.read().split()  
for tS in xS:  
 x = float(tS)  
 y = 0.0  
 k = 0  
 while k <= n:  
 L = 1.0  
 for j in range(0, k):  
 L \*= (x - x1[j]) / (x1[k] - x1[j])  
 for j in range(k + 1, n + 1):  
 L \*= (x - x1[j]) / (x1[k] - x1[j])  
 y += L \* y1[k]  
 k += 1  
 print(str(x) + " " + str(y))

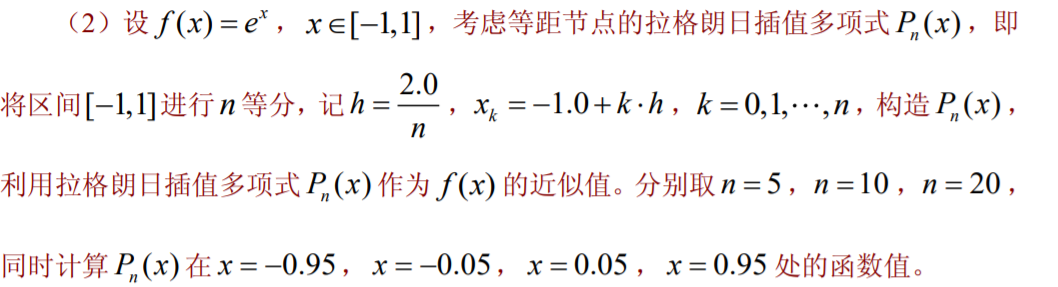


运行结果：

when n = 5  
  
0.75 0.528973858173077  
1.75 0.3733248197115384  
2.75 0.15373347355769232  
3.75 -0.025954026442307696  
4.75 -0.015737680288461536

when n = 10  
  
0.75 0.6789895772933959  
1.75 0.1905804667537568  
2.75 0.21559187891256765  
3.75 -0.23146174989674448  
4.75 1.9236311497192042

when n = 20  
  
0.75 0.6367553359164334  
1.75 0.23844593373813291  
2.75 0.08065999342165547  
3.75 -0.4470519607088335  
4.75 -39.95244903304179

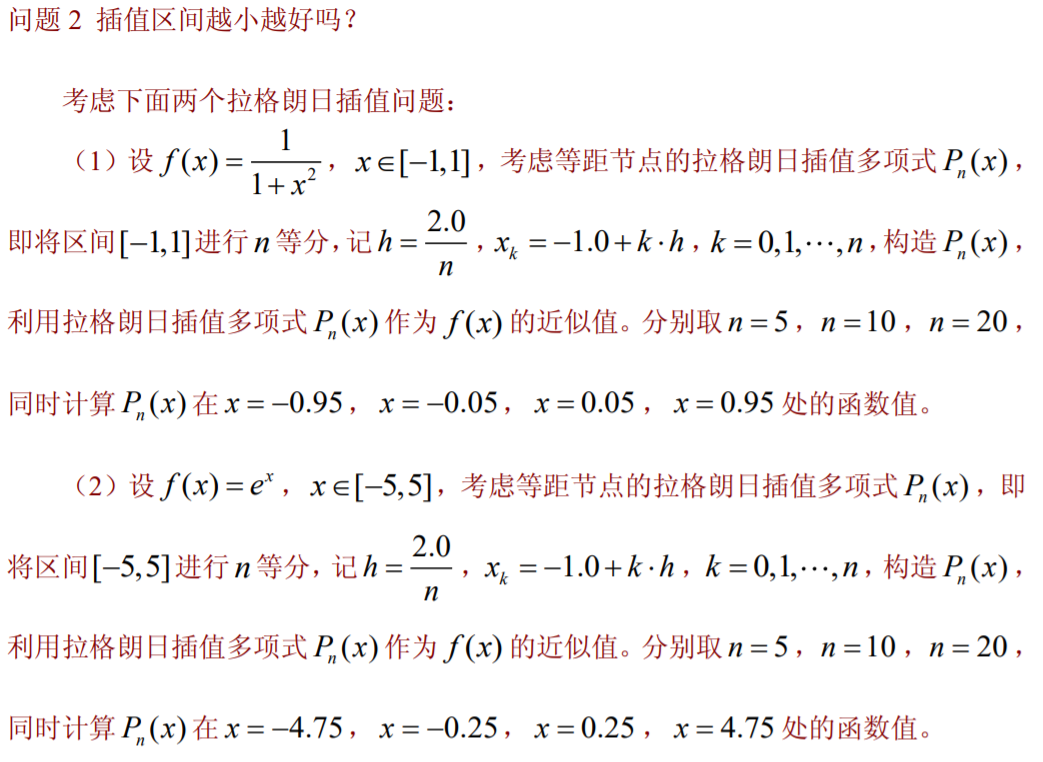


when n = 5  
  
-0.95 0.38679815885799373  
-0.05 0.9512483333804466  
0.05 1.0512902758084777  
0.95 2.585784550984614

when n = 10  
  
-0.95 0.3867410232556754  
-0.05 0.9512294244990157  
0.05 1.0512710963777372  
0.95 2.58570965954876

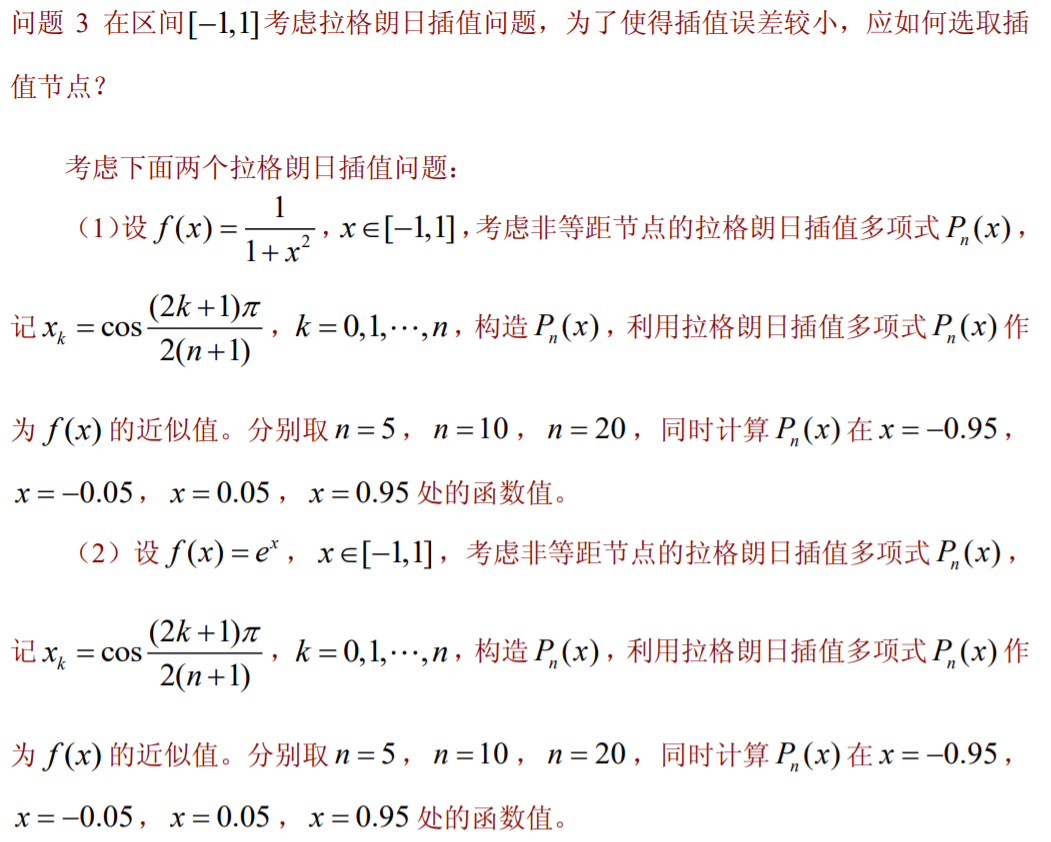
when n = 20  
  
-0.95 0.386741023454367  
-0.05 0.9512294245007143  
0.05 1.0512710963760237  
0.95 2.5857096593164632

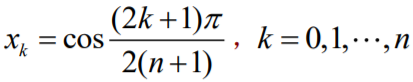
可见误差是偶然出现的，这个方法不能保证函数完全拟合。边缘数据可能会出现误差特别大的情况。



同理设计实验，由实验数据可得出小的区间不一定能更加精确。

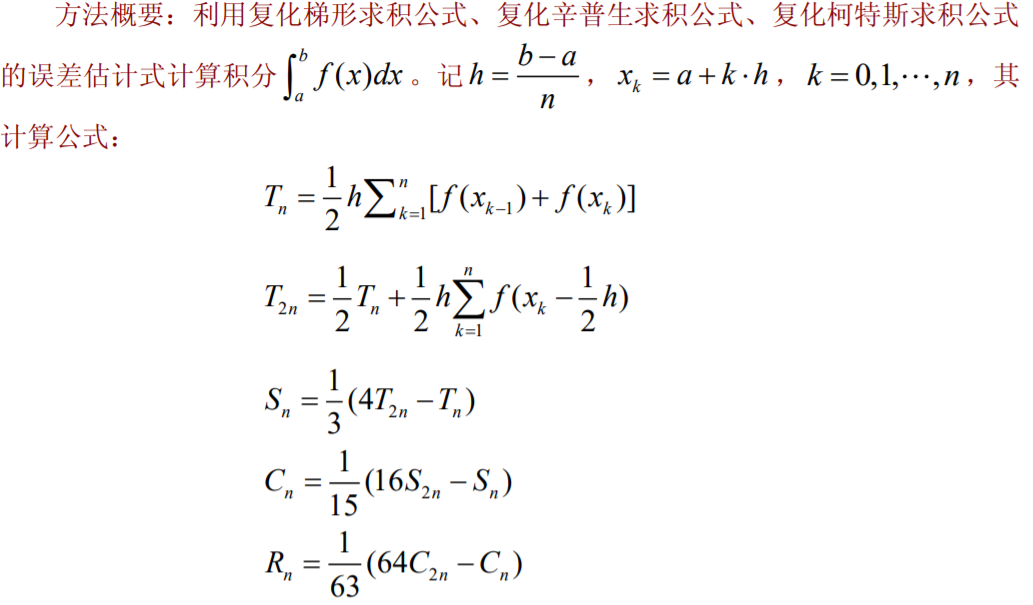
在（1）中，差值区间变小以后误差会变小，但是在（2）中，区间变小以后误差就变大了可以得出曲线小的也不一定准确。



由数据可知，非等距选择节点的优点就是得出结果的误差都在同一个数量级上，而等距选择节点，误差的数量级不同，而且有的误差是相当的大。所以按照题目中非等距的选择节点效果更好一些。

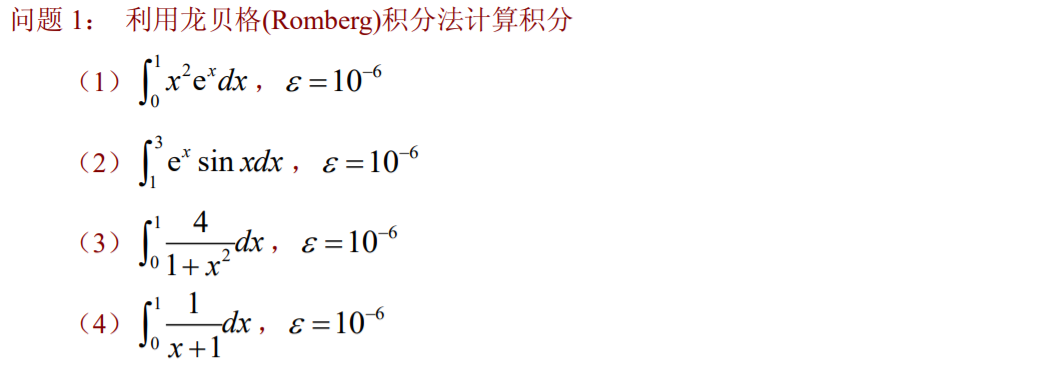
内插即所求点在样本范围之内，而外推则是在范围之外。由实验数据可知，内插是优于外推的，所以选择数据尽量选择内推的节点。

## 龙贝格积分法



程序实现（Python）：

import numpy as np  
  
  
def f(x):  
 return x \* x \* np.exp(x)  
  
  
# a, b, N, eps = input().split()  
a = 0.0  
b = 1.0  
N = 10  
eps = 1e-6  
  
h = (b - a) / N  
m = 1.0  
T = [0, 0, 0]  
S = [0, 0, 0]  
C = [0, 0, 0]  
R = [0, 0, 0]  
T[1] = h / 2.0 \* (f(a) + f(b))  
print(T[1])  
  
for i in range(2, N + 1):  
 ii = 2 \*\* (i - 1)  
 tmp = 0.0  
 for k in range(1, ii + 1):  
 tmp += f(a + (k - 0.5) \* h)  
 T[2] = T[1] / 2.0 + h / 2.0 \* tmp  
 print(T[2])  
 S[2] = (4 \* T[2] - T[1]) / 3.0  
 if m != 1.0:  
 C[2] = (16.0 \* S[2] - S[1]) / 15.0  
 print(C[2])  
 elif m != 2.0:  
 R[2] = (64.0 \* C[2] - C[1]) / 63.0  
 print(R[2])  
 elif m != 3.0:  
 tol = np.abs(R[2] - R[1])  
 if tol < eps:  
 break  
 R[1] = R[2]  
 C[1] = C[2]  
 S[1] = S[2]  
 T[1] = T[2]  
 h = h / 2.0  
 m = m + 1.0



计算结果：

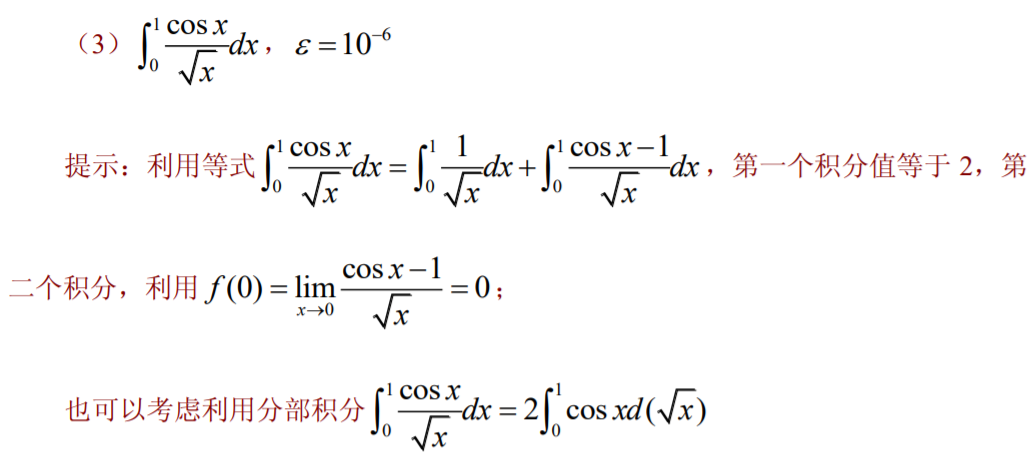
0.71834  
10.95023  
3.14165  
0.69317

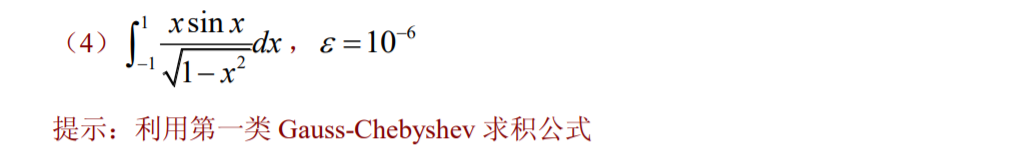
结果是0.94606



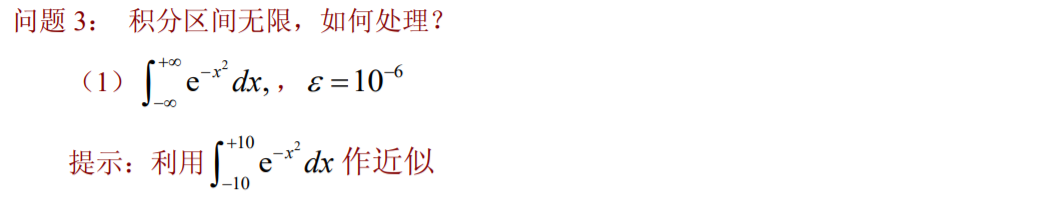
结果是1.49969



结果是1.80898

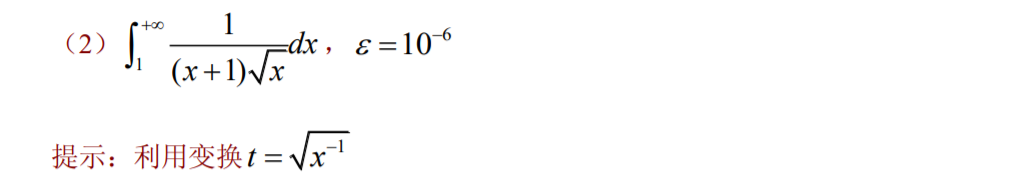


结果是1.3826

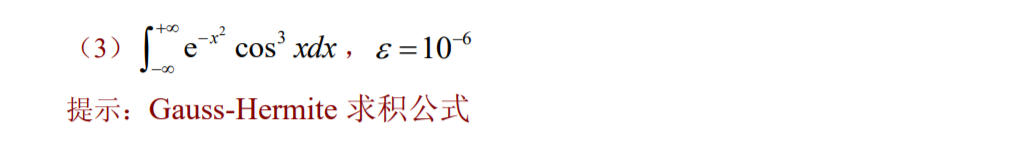


把正无穷的值取成一个足够大的值，误差就会变得很小。

结果是：1.7725



结果是1.5708

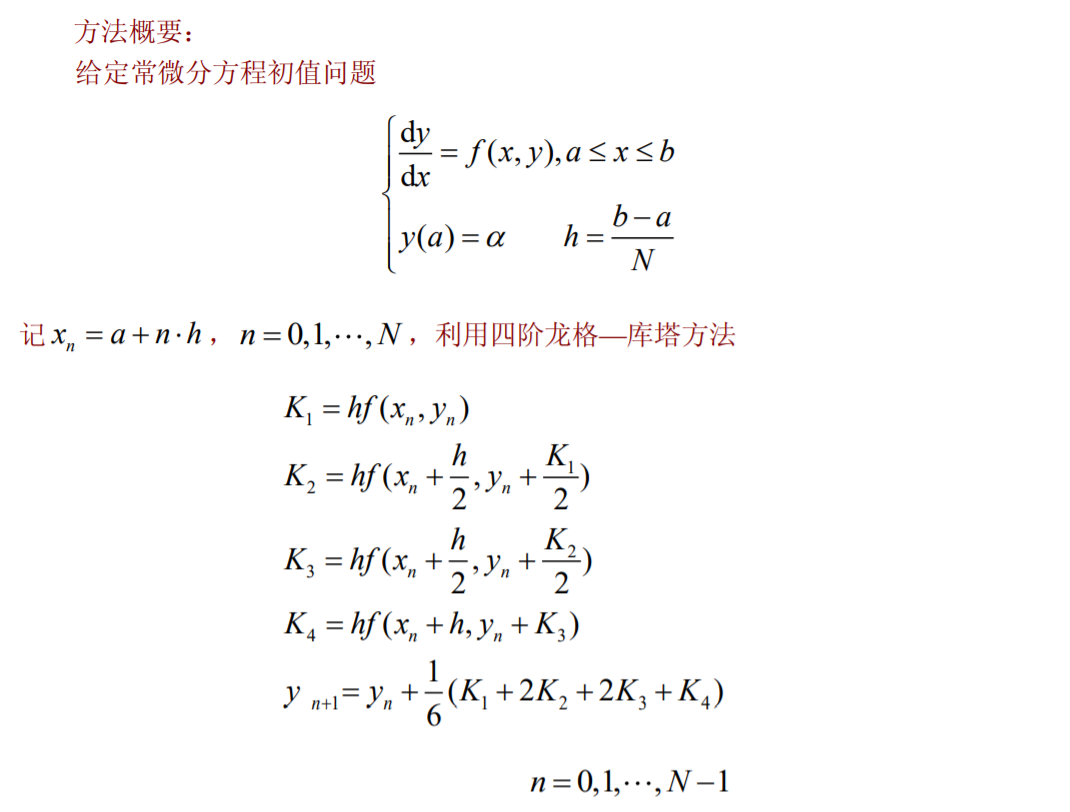


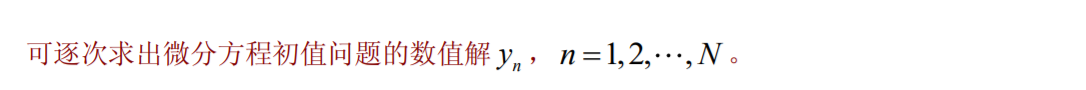
结果是1.0821



结果是0.4

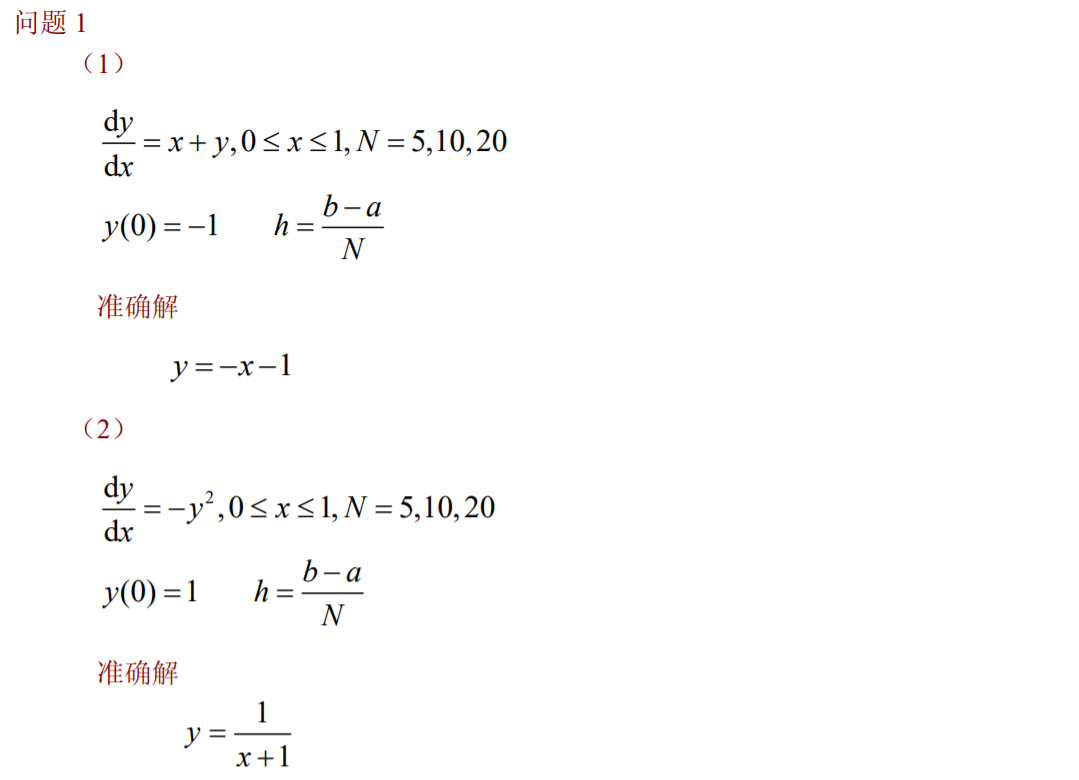
## 四阶龙格-库塔方法





代码实现：

K = []  
a, b, alpha, N = input.split()  
x0 = a, y0 = alpha, h = (b - a) / N  
for n in range(1, N + 1):  
 K[1] = h \* f(x0, y0)  
 K[2] = h \* f(x0 + h / 2, y0 + K[1] / 2)  
 K[3] = h \* f(x0 + h / 2, y0 + K[2] / 2)  
 K[4] = h \* f(x0 + h, y0 + K[3])  
 x1 = x0 + h  
 y1 = y0 + (K[1] + 2 \* K[2] + 2 \* K[3] + K[4]) / 6  
 print(str(x1) + " " + str(y1))  
 x0 = x1, y0 = y1

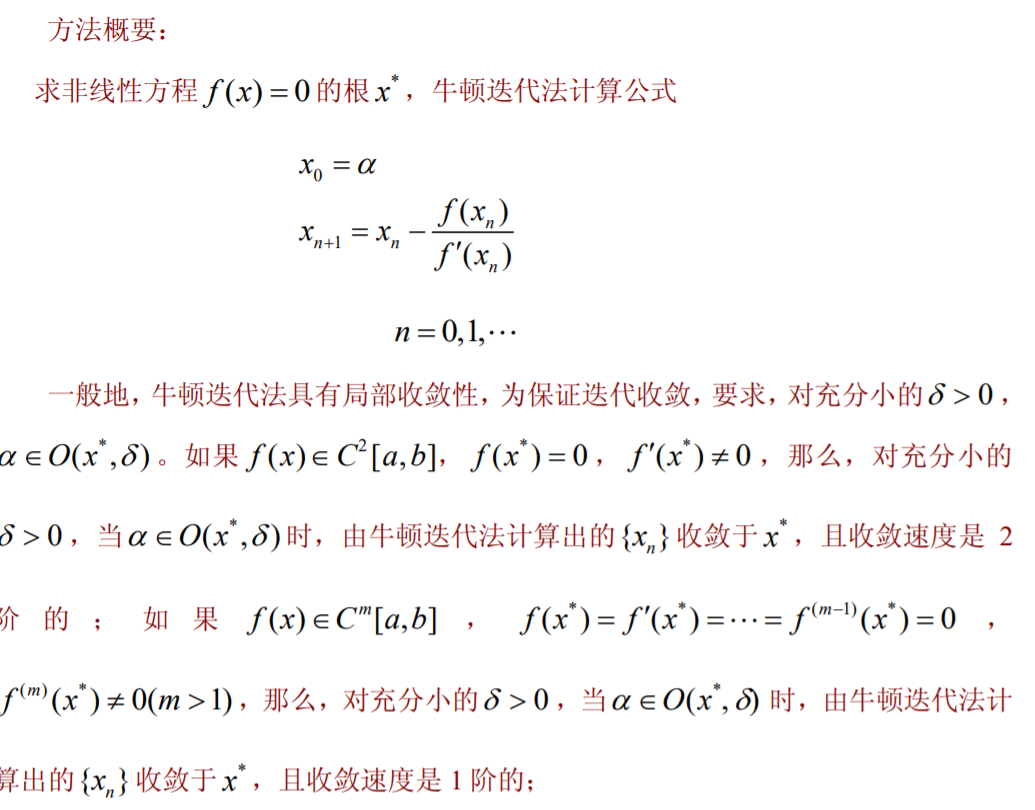


1.实验1中的数值解和解析解是相同的，但是实验二中的不同。四阶R-K方法的过程中用小段的线性算法来近似获得微分方程的数值解，（1）的解析解阶数为1，（2）的阶数无限，所以1中的是一样的，2中的不一样。

2.N取较大的值会使结果更精确。N越大，步长就越小，有利于R-K方法取到更精确的值。

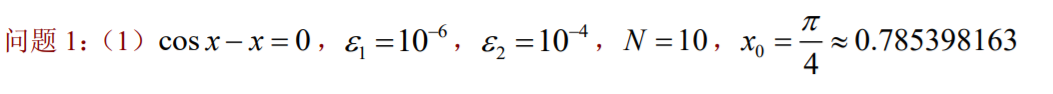
3.N过小会导致误差过大无法得到精确解。

## 牛顿迭代法

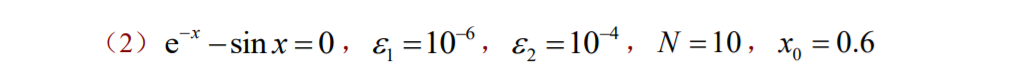


代码实现：

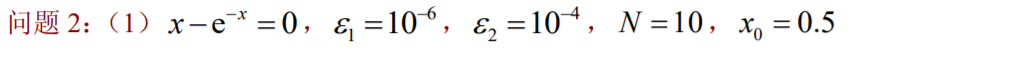
import numpy as np  
  
alpha = 0x5f3759df  
  
  
def f(x):  
 return np.sqrt(x)  
  
  
def df(x):  
 return np.sqrt(x) / x / 2  
  
  
def \_\_main\_\_():  
 eps1, eps2, cnt = input().split()  
 n = 1  
 x0 = alpha  
 while n < cnt:  
 f\_res = f(x0)  
 df\_res = df(x0)  
 if np.abs(f\_res) < eps1:  
 return x0  
 elif np.abs(df\_res) < eps2:  
 return -1  
 x1 = x0 - f\_res / df\_res  
 tol = np.abs(x1 - x0)  
 if np.abs(tol) < eps1:  
 return x1  
 n += 1  
 x0 = x1  
 return -1



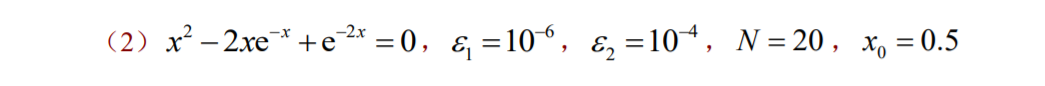
结果：0.7391



结果：0.5885



结果：0.5671



结果：0.5669