

计算方法实验报告

姓名：郑淇晟

学号：200110614

院系：计算机科学与技术学院

专业：计算机科学与技术

班级：六班

实验报告一：拉格朗日插值

第一部分：问题分析 *（描述并总结出实验题目）*

本题考察利用拉格朗日插值法求解插值函数，并利用插值函数估计插值点的值。问题一、二采用等分插值，探究插值多项式系数大小，插值区间大小对估计值准确性的影响；问题四选取三个插值点，探究内插和外推的效果差别。

第二部分：数学原理

拉格朗日插值法的目标是利用插值多项式和已知的函数上的离散点去逼近真实的函数曲线，即

$$f(x) \approx P_n(x)$$

其中 P_n 满足以下条件：

$$P_n(x_i) = f(x_i) \quad , \quad i = 0, 1, 2, \dots, n$$

$$P_n = \sum_{i=0}^n f(x_i) * l_n(x_i)$$

$$l_n(x_i) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}$$

若 $x_k \in [a, b]$ ，且 $f(x)$ 充分光滑，则当 $x \in [a, b]$ 时有误差估计式

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!} \prod_{j=0}^n (x - x_j) \quad , \quad \varepsilon \in [a, b]$$

第三部分：程序设计流程

```
from sympy import *

# 获取方程
def get_func(Q):
    x = symbols('x')
    if Q == 1:
        expr = 1 / (1+x**2)
    elif Q == 2:
        expr = exp(x)
    elif Q == 4:
        expr = x**(0.5)
    f = lambdify(x, expr, "numpy")
    return f

# 返回插值节点的迭代器
def dataloader(a, b, N, f):
    for n in N:
        x_list = list()
        y_list = list()
        h = (b - a) / n
        for i in range(n+1):
            x = a + h*i
            y = f(x)
            x_list.append(x)
            y_list.append(y)
        yield x_list, y_list

# 拉格朗日插值，返回估计值
def lagrange_interpolation(x_list, y_list, X):
    V = list()
    for x in X:
        y = 0
        n = len(x_list)
        for i in range(n):
            l = 1
            for j in range(n):
                if j == i:
                    continue
                else:
                    l *= (x - x_list[j]) / (x_list[i] - x_list[j])
            y += l*y_list[i]
        V.append(y)
    return V

def main():
    print("*****问题1.1*****")
    f = get_func(Q=1)
    X = [0.75, 1.75, 2.75, 3.75, 4.75]
    N = [5, 10, 20]
    V = [f(x) for x in X]
    print(f"真实值 {V}")
    for x_list, y_list in dataloader(a=-5, b=5, N=N, f=f):
        V_hat = lagrange_interpolation(x_list, y_list, X)
        print(f"等分次数 {len(x_list) - 1}, 估计值 {V_hat}")

    print("*****问题1.2*****")
    f = get_func(Q=2)
    X = [-0.95, -0.05, 0.05, 0.95]
    N = [5, 10, 20]
    V = [f(x) for x in X]
    print(f"真实值 {V}")
    for x_list, y_list in dataloader(a=-1, b=1, N=N, f=f):
        V_hat = lagrange_interpolation(x_list, y_list, X)
        print(f"等分次数 {len(x_list) - 1}, 估计值 {V_hat}")

    print("*****问题2.1*****")
    f = get_func(Q=1)
    X = [-0.95, -0.05, 0.05, 0.95]
    N = [5, 10, 20]
    V = [f(x) for x in X]
    print(f"真实值 {V}")
    for x_list, y_list in dataloader(a=-1, b=1, N=N, f=f):
        V_hat = lagrange_interpolation(x_list, y_list, X)
        print(f"等分次数 {len(x_list) - 1}, 估计值 {V_hat}")

    print("*****问题2.2*****")
    f = get_func(Q=2)
    X = [-4.75, -0.25, 0.25, 4.75]
    N = [5, 10, 20]
    V = [f(x) for x in X]
    print(f"真实值 {V}")
    for x_list, y_list in dataloader(a=-5, b=5, N=N, f=f):
        V_hat = lagrange_interpolation(x_list, y_list, X)
        print(f"等分次数 {len(x_list) - 1}, 估计值 {V_hat}")

    print("*****问题4.1*****")
    f = get_func(Q=4)
    X = [5, 50, 115, 185]
    V = [f(x) for x in X]
    x_list = [1, 4, 9]
    y_list = [f(x) for x in x_list]
    print(f"真实值 {V}")
    V_hat = lagrange_interpolation(x_list, y_list, X)
    print(f"估计值 {V_hat}")

    print("*****问题4.2*****")
    f = get_func(Q=4)
    X = [5, 50, 115, 185]
    V = [f(x) for x in X]
    x_list = [36, 49, 64]
    y_list = [f(x) for x in x_list]
    print(f"真实值 {V}")
    V_hat = lagrange_interpolation(x_list, y_list, X)
    print(f"估计值 {V_hat}")

    print("*****问题4.3*****")
    f = get_func(Q=4)
    X = [5, 50, 115, 185]
    V = [f(x) for x in X]
    x_list = [100, 121, 144]
    y_list = [f(x) for x in x_list]
    print(f"真实值 {V}")
    V_hat = lagrange_interpolation(x_list, y_list, X)
    print(f"估计值 {V_hat}")

    print("*****问题4.4*****")
    f = get_func(Q=4)
    X = [5, 50, 115, 185]
    V = [f(x) for x in X]
    x_list = [169, 196, 225]
    y_list = [f(x) for x in x_list]
    print(f"真实值 {V}")
    V_hat = lagrange_interpolation(x_list, y_list, X)
    print(f"估计值 {V_hat}")

if __name__ == "__main__":
    main()
```

第四部分：实验结果、结论与讨论

实验结果：

```
*****问题1.1*****
待估计点 [0.75, 1.75, 2.75, 3.75, 4.75]
真实值 [0.64, 0.24615384615384617, 0.11678832116788321, 0.06639004149377593, 0.042440318302387266]
等分次数 5, 估计值 [0.528973858173077, 0.3733248197115384, 0.15373347355769232, -0.025954026442307696, -0.015737680288461536]
等分次数 10, 估计值 [0.6789895772933959, 0.1905804667537568, 0.21559187891256765, -0.23146174989674448, 1.9236311497192042]
等分次数 20, 估计值 [0.6367553359164334, 0.23844593373813291, 0.08065999342165547, -0.4470519607088335, -39.95244903304179]
*****问题1.2*****
待估计点 [-0.95, -0.05, 0.05, 0.95]
真实值 [0.38674102345450123, 0.951229424500714, 1.0512710963760241, 2.585709659315846]
等分次数 5, 估计值 [0.38679815885799373, 0.9512483333804466, 1.0512902758084777, 2.585784550984614]
等分次数 10, 估计值 [0.3867410232556754, 0.9512294244990157, 1.0512710963777372, 2.58570965954876]
等分次数 20, 估计值 [0.386741023454367, 0.9512294245007143, 1.0512710963760237, 2.5857096593164632]
*****问题2.1*****
待估计点 [-0.95, -0.05, 0.05, 0.95]
真实值 [0.5256241787122208, 0.9975062344139651, 0.9975062344139651, 0.5256241787122208]
等分次数 5, 估计值 [0.5171472886029412, 0.9927906709558822, 0.9927906709558822, 0.5171472886029412]
等分次数 10, 估计值 [0.5264079831033781, 0.9975068566175548, 0.9975068566175548, 0.5264079831033801]
等分次数 20, 估计值 [0.5256203657475115, 0.997506234424685, 0.9975062344246846, 0.5256203657482508]
*****问题2.2*****
待估计点 [-4.75, -0.25, 0.25, 4.75]
真实值 [0.008651695203120634, 0.7788007830714049, 1.2840254166877414, 115.58428452718766]
等分次数 5, 估计值 [1.147034731494406, 1.3021524636392101, 1.841210410969898, 119.62100705608145]
等分次数 10, 估计值 [-0.0019565504594416527, 0.7786863439040118, 1.2841444870024037, 115.60736006305402]
等分次数 20, 估计值 [0.00865169378436137, 0.7788007830714531, 1.2840254166876928, 115.58428452936269]
*****问题4.1*****
待估计点 [5, 50, 115, 185]
真实值 [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.601470508735444]
估计值 [2.2666666666666666, -20.23333333333329, -171.9000000000001, -492.7333333333331]
*****问题4.2*****
待估计点 [5, 50, 115, 185]
真实值 [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.601470508735444]
估计值 [3.1157509157509224, 7.0717948717948715, 10.167032967033009, 10.038827838827785]
*****问题4.3*****
待估计点 [5, 50, 115, 185]
真实值 [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.601470508735444]
估计值 [4.439111613024664, 7.284961415396175, 10.722755505364201, 13.535667231319408]
*****问题4.4*****
待估计点 [5, 50, 115, 185]
真实值 [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.601470508735444]
估计值 [5.497172048896061, 7.800127713920801, 10.800492610837424, 13.600620324758257]
```

结论：

问题一：插值次数 n 并非总是越大越好，由问题 1.2 实验结果可见，等分次数越多结果确实越准确；但问题 1.1 某些值随着等分次数增加而偏离精确值

问题二：插值区间并非越小越好，问题 1.2 和问题 2.2 是同一函数在两个大小不同的区间的积分，但二者的精确度差异不大

问题四：由实验结果可知，内插比外推更可靠

讨论：

1. 随着插值次数的增加，在插值函数在区间两端产生龙格现象，可采用分段低次插值等方法。

2. 分别对比 1.1 与 2.1 和 1.2 与 2.2 的结果与准确值的误差可知插值区间不是越小越好。

4. 如果 x 的值在插值节点最大值和最小值之间，则成为内插，否则成为外推，内插的结果相较于外推更加准确。

实验报告二：龙贝格积分法

第一部分：问题分析 *(描述并总结出实验题目)*

本题考察利用龙贝格积分法求定积分的近似解。

给定被积函数，积分区间和目标精度，运用龙贝格积分法迭代计算数表 T，若当前结果满足目标精度，则结束迭代过程，输出当前结果。

第二部分：数学原理

待求解问题:求解下列方程，使精度达到给定的 epsilon

$$\int_a^b f(x) dx$$

梯形公式:

$$\int_a^b f(x) dx \approx \frac{b-a}{2} (f(a) + f(b))$$

龙贝格积分法的思想即为，通过迭代增加二分次数和外推次数来逼近定积分的精确解。实际做法为计算数表 T（横轴表示二分次数，纵轴表示外推次数），若当前结果满足目标精度，则结束迭代过程，输出当前结果。

二分:

$$T_{2n} = \frac{1}{2} (T_n + h \sum_{k=1}^n f(a + (k - \frac{1}{2})h))$$

外推:

$$T_m^{(k+1)} = \frac{4^m T_{m-1}^{(k+1)} - T_{m-1}^{(k)}}{4^m - 1}$$

第三部分：程序设计流程

```
1  import numpy as np
2  from sympy import *
3
4  # 判断哪个问题
5  def get_func(Q):
6      x = symbols('x')
7      if Q == 1:
8          expr = x**2 * exp(x)
9      elif Q == 2:
10         expr = sin(x) * exp(x)
11     elif Q == 3:
12         expr = 4/(1+x**2)
13     elif Q == 4:
14         expr = 1/(x+1)
15     else:
16         return
17     return lambdify(x, expr, "numpy")
18
19 # 返回T[0][0]的值
20 def T_0_0(f, a, b):
21     return 0.5*(b-a)*(f(a)+f(b))
22
23 # 初始化T[0][i]
24 def init(T, f, h, a, i):
25     T[0][i] = 0.5*T[0][i-1]
26     for k in range(1, 2**(i-1)+1):
27         T[0][i] += 0.5 * h * f(a+(k-1/2)*h)
28     return T
29
30 # 龙贝格积分
31 def Romberg(Q, a, b, epsilon=1e-6):
32     i = 0
33     h = b-a
34     f = get_func(Q)
35     T = np.zeros(shape=(10,10))
36     T[0][0] = T_0_0(f, a, b)
37     loss = 100
38     while(loss > epsilon):
39         i += 1
40         T = init(T, f, h, a, i)
41         for m in range(1, i+1):
42             T[m][i-m] = (4**m * T[m-1][i-m+1] - T[m-1][i-m]) / (4**m-1)
43             loss = np.abs(T[m][0] - T[m-1][0])
44             h = (b-a) / 2**i
45     return T, i
46
47 # 打印数表T
48 def print_res(T, i):
49     for j in range(i+1):
50         for k in range(i-j+1):
51             print("%.16f"%T[j][k], end=' ')
52         print("")
53
54 def main():
55     print("***** 问题1.1*****")
56     print_res(*Romberg(Q=1, a=0, b=1))
57     print("***** 问题1.2*****")
58     print_res(*Romberg(Q=2, a=1, b=3))
59     print("***** 问题1.3*****")
60     print_res(*Romberg(Q=3, a=0, b=1))
61     print("***** 问题1.4*****")
62     print_res(*Romberg(Q=4, a=0, b=1))
63
64 if __name__ == "__main__":
65     main()
```

第四部分：实验结果、结论与讨论

实验结果：

```
*****问题1.1*****
1.3591409142295225 0.8856606159522773 0.7605963324480420 0.7288901770146928 0.7209357789376580
0.7278338498598623 0.7189082379466303 0.7183214585369098 0.7182843129119797
0.7183131971524148 0.7182823399095951 0.7182818365369844
0.7182818501120901 0.7182818285469430
0.7182818284623739
*****问题1.2*****
5.1218264196658465 9.2797629072611727 10.5205542838186421 10.8420434675574278 10.9230938896137779 10.9433984211867941
10.6657417364596139 10.9341514093377992 10.9492065288036908 10.9501106969658935 10.9501665983778000
10.9520453875296777 10.9502102034347502 10.9501709748433722 10.9501703251385951
10.9501810735284817 10.9501703521673193 10.9501703148258223
10.9501703101227665 10.9501703146793847
10.9501703146838381
*****问题1.3*****
3.0000000000000000 3.1000000000000001 3.1311764705882354 3.1389884944910889 3.1409416120413889 3.1414298931749749
3.1333333333333333 3.1415686274509809 3.1415925024587068 3.1415926512248222 3.1415926535528365
3.1421176470588241 3.1415940941258884 3.1415926611425631 3.1415926537080376
3.1415857837618737 3.1415926383967960 3.1415926535900294
3.141592652777171 3.1415926536496106
3.1415926536382441
*****问题1.4*****
0.7500000000000000 0.7083333333333333 0.6970238095238095 0.6941218503718503 0.6933912022075267
0.6944444444444443 0.6932539682539683 0.6931545306545305 0.6931476528194188
0.6931746031746032 0.6931479014812346 0.6931471942970781
0.6931474776448320 0.6931471830719328
0.6931471819167450
```

结论：

龙贝格积分法能够较为准确地估计定积分的值，且收敛速度较快，随着二分次数增加和外推次数增加，精度不断提升。

讨论：

思考题 2：实验中计算精度随着二分次数的增加而增加

实验报告三：牛顿迭代法

第一部分：问题分析 *（描述并总结出实验题目）*

本题考查利用牛顿迭代法计算非线性方程的根.

通过指定初值和迭代次数，以及损失的阈值，求解非线性方程组的近似解，并进一步探究初值的选取原则以及参数的选取和不同的方程对计算过程的影响

第二部分：数学原理

求解非线性方程 $f(x) = 0$ 的根 x^* 的牛顿计算公式：

$$x_0 = \alpha$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, \dots$$

通常，牛顿迭代法具有局部收敛性，要求对充分小的 $\delta > 0$

$$\alpha \in O(x^*, \delta) \quad \textcircled{1}$$

如果， $f(x) \in C^2[a, b]$ ， $f(x^*) = 0$ ， $f'(x^*) \neq 0$ ，那么在条件①下，由牛顿（Newton）迭代法计算出的 $\{x_n\}$ 收敛于 x^* ，且收敛阶为 2；

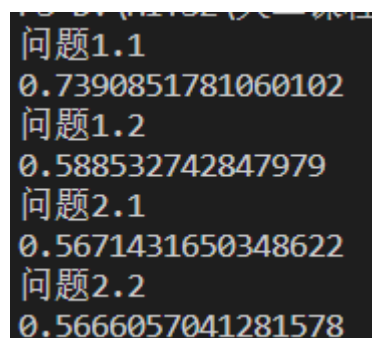
如果， $f(x) \in C^m[a, b]$ ， $f(x^*) = f'(x^*) = \dots = f^{(m-1)}(x^*) = 0$ ， $f^{(m)}(x^*) \neq 0$ ($m > 1$)，那么在条件①下，由牛顿（Newton）迭代法计算出的 $\{x_n\}$ 收敛于 x^* ，且收敛阶为 1。

第三部分：程序设计流程

```
1  from sympy import *
2  import numpy as np
3
4  # 获取原方程和求导后的方程
5  def get_func(Q):
6
7      x = symbols('x')
8      if Q == 11:
9          expr = cos(x) - x
10         f = lambdify(x, expr, "numpy")
11         f_grad = lambdify(x, expr.diff(x), "numpy")
12     elif Q == 12:
13         expr = exp(-x) - sin(x)
14         f = lambdify(x, expr, "numpy")
15         f_grad = lambdify(x, expr.diff(x), "numpy")
16     elif Q == 21:
17         expr = x - exp(-x)
18         f = lambdify(x, expr, "numpy")
19         f_grad = lambdify(x, expr.diff(x), "numpy")
20     elif Q == 22:
21         expr = x**2 - 2*x*exp(-x) + exp(-2*x)
22         f = lambdify(x, expr, "numpy")
23         f_grad = lambdify(x, expr.diff(x), "numpy")
24     return f, f_grad
25
26 # 牛顿迭代
27 def Newton(x0, N, Q, epsilon1 = 1e-6, epsilon2 = 1e-4):
28
29     # 获得问题方程
30     f, f_grad = get_func(Q)
31
32     # 迭代过程
33     for i in range(N):
34         if np.abs(f(x0)) < epsilon1:
35             print(f"{x0}")
36             break
37         elif np.abs(f_grad(x0)) < epsilon2:
38             print("FAIL")
39             break
40         x1 = x0 - f(x0) / f_grad(x0)
41         loss = np.abs(x1 - x0)
42         if loss < epsilon1:
43             print(f"{x1}")
44             break
45         else:
46             x0 = x1
47
48     # 输出错误标志
49     if i>=N:
50         print("FAIL")
51
52 def main():
53     print("问题1.1")
54     Newton(x0=np.pi/4, N=10, Q=11)
55     print("问题1.2")
56     Newton(x0=0.6, N=10, Q=12)
57     print("问题2.1")
58     Newton(x0=0.5, N=10, Q=21)
59     print("问题2.2")
60     Newton(x0=0.5, N=20, Q=22)
61
62 if __name__ == "__main__":
63     main()
```

第四部分：实验结果、结论与讨论

实验结果：



```
问题1.1
0.7390851781060102
问题1.2
0.588532742847979
问题2.1
0.5671431650348622
问题2.2
0.5666057041281578
```

结论：

牛顿迭代法可以用于求解非线性方程组的解，且精确度高，具有局部收敛性；但是，当非线性函数的次数大于 1 的时候，牛顿迭代法的收敛速度明显变慢；同一方程的不同迭代格式收敛速度也不一样。

讨论：

思考题 1：

实验一确定初值的原则是在初值处，函数的一阶导绝对值小于 1，以保证迭代收敛。实际中只需初值处一阶导和二阶导同号即可。

思考题 2：

实验二第二问的计算速度明显比第一问慢得多。原因是，虽然两个问题求解的是同一个方程，但是迭代格式不同，所以迭代次数和收敛速度会有所不同。

实验报告四：高斯列主元消去法

第一部分：问题分析 *（描述并总结出实验题目）*

本题考查利用高斯列选主元消去法求解方程组 $Ax = b$ 。

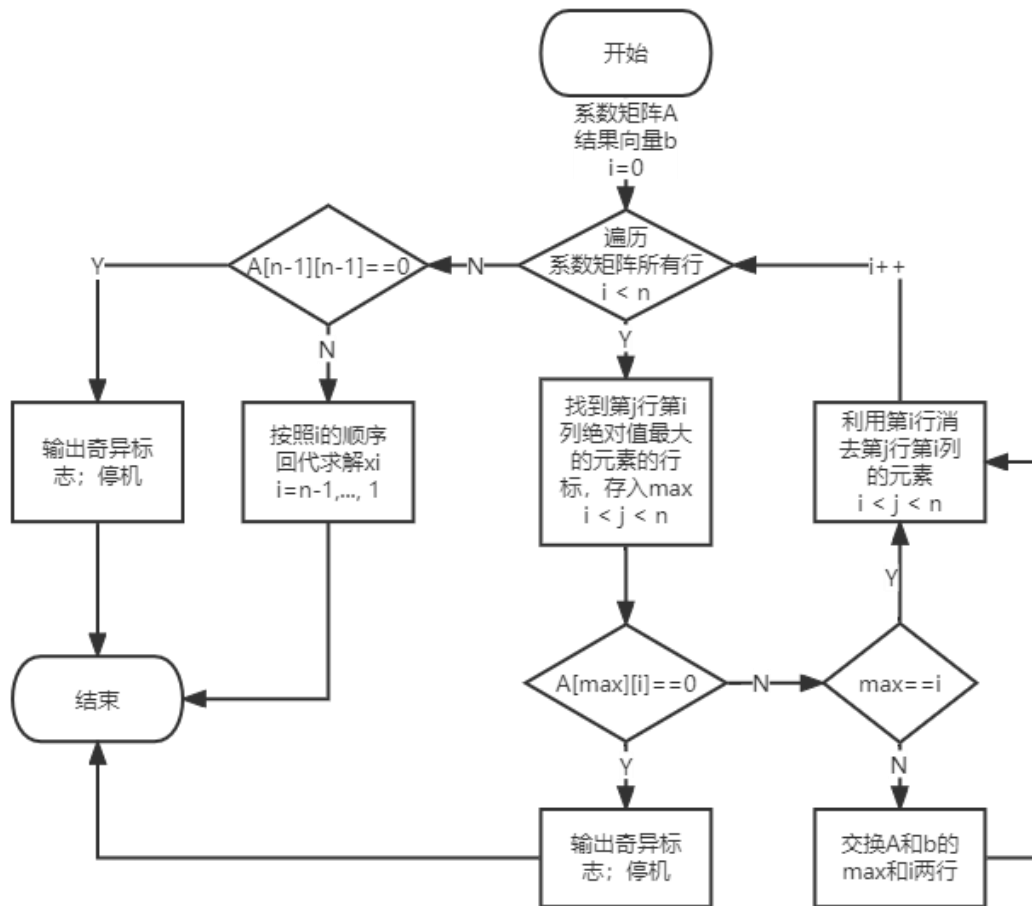
对给定的 n 阶线性方程组 $Ax = b$ ，首先进行列主元消去过程，然后进行回代过程，最后得到解或确定该线性方程组是奇异的。所谓列选主元的即选出当前列尚未更新的行绝对值最大的元素，是一种平衡技术。

第二部分：数学原理

对给定的 n 阶线性方程组 $Ax = b$ ，首先进行列主元消去过程，然后进行回代过程，最后得到解或确定该线性方程组是奇异的。

如果系数矩阵的元素按绝对值在数量级方面相差很大，那么，在进行列主元消去过程前，先把系数矩阵的元素进行行平衡：系数矩阵的每行元素和相应的右端向量元素同除以该行元素绝对值最大的元素，然后再进行列主元消去过程。

第三部分：程序设计流程



该题代码查看：\Gauss\Q1.py 和 \Gauss\Q2.py

第四部分：实验结果、结论与讨论

实验结果：

```
问题 1.1
[1.0000000000000027 1.0000000000000018 0.9999999999999971
0.9999999999999992]
问题 1.2
[1.00000000000001177 0.999999999999824 0.999999999999901
问题 1.3
[0.999999999999927 1.0000000000000693 0.9999999999998548
1.0000000000000085 ]
问题 1.4
[1. 1. 1. 1.]
```

```
问题 2.1
[ 0.9536791069017718 0.3209568455211036 1.0787080757932384
-0.0901085095395789]
问题 2.2
[0.5161772979585416 0.4152194728301353 0.1099661028678892
1.0365392233362005]
问题 2.3
[1. 0.9999999999999998 1.0000000000000002]
问题 2.4
[1. 1. 1.]
```

结论：实验中，浮点数的运算因为计算机表示位数有限而存在舍入误差，因此计算结果与真实值会有一定的差距，但对本实验中的数据而言，差距十分之微小，可以忽略不计。