计算方法实验报告

胡皓然
210110404
计算机科学与技术学院
计算机类
4 班

实验报告一：拉格朗日插值

第一部分：问题分析

在本实验中，需要使用拉格朗日插值法对给定数据进行插值，并通过编程实现该算法。具体来说，给定 n 个数据点 $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$，要求使用拉格朗日插值法计算在一个新的输入点 x0 处的插值 y0。

第二部分：数学原理

给定平面上 $n+1$ 个不同的数据点 $(x_k, f(x_k)), k = 0, 1, \cdots, n, x_i \neq x_j, i \neq j$ 则满足条件

$$P_n(x_k) = f(x_k), \quad k = 0, 1, \cdots, n$$

的 n 次拉格朗日插值多项式

$$P_n(x) = \sum_{k=0}^{n} f(x_k) l_k(x)$$

是存在唯一的。若 $x_k \in [a, b], k = 0, 1, \cdots, n$, 且函数 $f(x)$ 充分光滑, 则当 $x \in [a, b]$ 时, 有误差估计式

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n), \quad \xi \in [a, b]$$

第三部分：程序设计流程

```
from math import *
from sympy import *

print("----------拉格朗日插值法----------")
t = Symbol('x')
FX = sympify(input("输入f(x)表达式\n"))
print("选择数据点的输入方式")
print("1. 区间n等分（自动生成）")
print("2. 手动输入")
choice = int(input())
if choice == 1:
    n = int(input("n="))  # n等分
    aa, bb = input("区间\n").split(" ")
    aa = float(aa)
    bb = float(bb)
    a = [[0.0] * 2 for i in range(n + 1)]
```

```
    h = (bb - aa) / n
    print("\n自动生成数据点")
    print("x\t\t f(x)")
    for k in range(n + 1):
        a[k][0] = aa + k * h
        a[k][1] = float(FX.subs(t, a[k][0]))
    for k in range(n + 1):
        print(format(a[k][0], '.6f'), '\t', format(a[k][1], '.6f'))
elif choice == 2:
    n = int(input("输入个数:"))
    n = n - 1
    a = [[0.0] * 2 for i in range(n + 1)]
    print("输入节点x值")
    inputx = input().split(" ")
    for k in range(n + 1):
        a[k][0] = int(inputx[k])
        a[k][1] = float(FX.subs(t, a[k][0]))
    print("\n数据点")
    print("x\t\t f(x)")
    for k in range(n + 1):
        print(format(a[k][0], '.6f'), '\t', format(a[k][1], '.6f'))

print("\n求解")
m = int(input("插值点个数:"))
print("插值点")
for i in range(m):
    x = float(input())
    y = 0.0
    k = 0
    while k <= n:
        l = 1.0
        for j in range(n + 1):
            if j != k:
                l = l * (x - a[j][0]) / (a[k][0] - a[j][0])
        y = y + l * a[k][1]
        k += 1
    print(
        "x:", format(x, '.6f'),
        "\t近似值:", format(y, '.8f'),
        "\t真值:", format(float(FX.subs(t, x)), '.8f'),
    )
```

第四部分：实验结果、结论与讨论

实验结果

问题 1（1）

n=5

```
x: 0.750000     近似值: 0.52897386      真值: 0.64000000
x: 1.750000     近似值: 0.37332482      真值: 0.24615385
x: 2.750000     近似值: 0.15373347      真值: 0.11678832
x: 3.750000     近似值: -0.02595403     真值: 0.06639004
x: 4.750000     近似值: -0.01573768     真值: 0.04244032
```

n=10

```
x: 0.750000     近似值: 0.67898958      真值: 0.64000000
x: 1.750000     近似值: 0.19058047      真值: 0.24615385
x: 2.750000     近似值: 0.21559188      真值: 0.11678832
x: 3.750000     近似值: -0.23146175     真值: 0.06639004
x: 4.750000     近似值: 1.92363115      真值: 0.04244032
```

n=20

```
x: 0.750000     近似值: 0.63675534      真值: 0.64000000
x: 1.750000     近似值: 0.23844593      真值: 0.24615385
x: 2.750000     近似值: 0.08065999      真值: 0.11678832
x: 3.750000     近似值: -0.44705196     真值: 0.06639004
x: 4.750000     近似值: -39.95244903    真值: 0.04244032
```

问题 1（2）

n=5

```
x: -0.950000    近似值: 0.38679816      真值: 0.38674102
x: -0.050000    近似值: 0.95124833      真值: 0.95122942
x: 0.050000     近似值: 1.05129028      真值: 1.05127110
x: 0.950000     近似值: 2.58578455      真值: 2.58570966
```

n=10

```
x: -0.950000    近似值: 0.38674102      真值: 0.38674102
x: -0.050000    近似值: 0.95122942      真值: 0.95122942
x: 0.050000     近似值: 1.05127110      真值: 1.05127110
x: 0.950000     近似值: 2.58570966      真值: 2.58570966
```

n=20

```
x: -0.950000    近似值: 0.38674102      真值: 0.38674102
x: -0.050000    近似值: 0.95122942      真值: 0.95122942
x: 0.050000     近似值: 1.05127110      真值: 1.05127110
x: 0.950000     近似值: 2.58570966      真值: 2.58570966
```

问题 2（1）

n=5

```
x: -0.950000    近似值: 0.51714729      真值: 0.52562418
x: -0.050000    近似值: 0.99279067      真值: 0.99750623
x: 0.050000     近似值: 0.99279067      真值: 0.99750623
x: 0.950000     近似值: 0.51714729      真值: 0.52562418
```

n=10

```
x: -0.950000    近似值: 0.52640798      真值: 0.52562418
x: -0.050000    近似值: 0.99750686      真值: 0.99750623
x: 0.050000     近似值: 0.99750686      真值: 0.99750623
x: 0.950000     近似值: 0.52640798      真值: 0.52562418
```

n=20

```
x: -0.950000    近似值: 0.52562037      真值: 0.52562418
x: -0.050000    近似值: 0.99750623      真值: 0.99750623
x: 0.050000     近似值: 0.99750623      真值: 0.99750623
x: 0.950000     近似值: 0.52562037      真值: 0.52562418
```

问题 2（2）

n=5

```
x: -4.750000    近似值: 1.14703473      真值: 0.00865170
x: -0.250000    近似值: 1.30215246      真值: 0.77880078
x: 0.250000     近似值: 1.84121041      真值: 1.28402542
x: 4.750000     近似值: 119.62100706    真值: 115.58428453
```

n=10

```
x: -4.750000    近似值: -0.00195655     真值: 0.00865170
x: -0.250000    近似值: 0.77868634      真值: 0.77880078
x: 0.250000     近似值: 1.28414449      真值: 1.28402542
x: 4.750000     近似值: 115.60736006    真值: 115.58428453
```

n=20

```
x: -4.750000    近似值: 0.00865169      真值: 0.00865170
```

```
x: -0.250000    近似值: 0.77880078      真值: 0.77880078
x: 0.250000     近似值: 1.28402542      真值: 1.28402542
x: 4.750000     近似值: 115.58428453    真值: 115.58428453
```

问题 4（1）

```
x: 5.000000     近似值: 2.26666667      真值: 2.23606798
x: 50.000000    近似值: -20.23333333    真值: 7.07106781
x: 115.000000   近似值: -171.90000000   真值: 10.72380529
x: 185.000000   近似值: -492.73333333   真值: 13.60147051
```

问题 4（2）

```
x: 5.000000     近似值: 3.11575092      真值: 2.23606798
x: 50.000000    近似值: 7.07179487      真值: 7.07106781
x: 115.000000   近似值: 10.16703297     真值: 10.72380529
x: 185.000000   近似值: 10.03882784     真值: 13.60147051
```

问题 4（3）

```
x: 5.000000     近似值: 4.43911161      真值: 2.23606798
x: 50.000000    近似值: 7.28496142      真值: 7.07106781
x: 115.000000   近似值: 10.72275551     真值: 10.72380529
x: 185.000000   近似值: 13.53566723     真值: 13.60147051
```

问题 4（4）

```
x: 5.000000     近似值: 5.49717205      真值: 2.23606798
x: 50.000000    近似值: 7.80012771      真值: 7.07106781
x: 115.000000   近似值: 10.80049261     真值: 10.72380529
x: 185.000000   近似值: 13.60062032     真值: 13.60147051
```

思考题

思考题 1 存在的问题：次数高时，插值多项式的值反而更不准确。解决方法：可以将等距节点替换为切比雪夫零点。

思考题 2 不是。对比问题 1(1) 与问题 2(1) 的结果，对比问题 1(2) 与问题 2(2) 的结果，可知插值区间不是越小越好。

思考题 3 内插是指在已知数据点之间进行插值，插值结果只在已知数据点之间有效。外推则是指在已知数据点之外进行插值，插值结果在已知数据点之外也可能有效。

实验报告二 龙贝格积分法

第一部分：问题分析

龙贝格积分法是一种数值积分方法，通过递归分割区间和近似计算函数积分值。相对于传统的数值积分方法，如梯形法、辛普森法等，龙贝格积分法具有更高的精度和更快的收敛速度。本实验可以让我们深入了解龙贝格积分法的原理和实现过程，锻炼数值计算和编程能力，同时也能对数值积分方法的优缺点有更加深入的认识。

第二部分：数学原理

利用复化梯形求积公式、复化辛普生求积公式、复化柯特斯求积公式的误差估计式计算积分 $\int_a^b f(x)dx$ 。记 $h=\frac{b-a}{n}, x_k=a+k\cdot h, k=0,1,\cdots,n$, 其计算公式:

$$T_n=\frac{1}{2}h\sum_{k=1}^{n}\left[f\left(x_{k-1}\right)+f\left(x_k\right)\right]$$

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

$$S_n=\frac{1}{3}\left(4T_{2n}-T_n\right)$$

$$C_n=\frac{1}{15}\left(16S_{2n}-S_n\right)$$

$$R_n=\frac{1}{63}\left(64C_{2n}-C_n\right)$$

一般地, 利用龙贝格算法计算积分, 要输出所谓的 T-数表

$$\begin{array}{cccccc} T_1 & & & & \\ T_2 & S_1 & & & \\ T_4 & S_2 & C_1 & & \\ T_8 & S_4 & C_2 & R_1 & \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

第三部分：程序设计流程

```
from sympy import *
from math import *
import sys

x = Symbol("x")
a, b, e, f = input("input:\na,b,epsilon,f\n").split(" ")
a = float(a)
b = float(b)
e = float(e)
f = sympify(f)

n = 10

t = [[[] for i in range(n)] for i in range(n)]
h = b - a
t[0][0] = h * (f.subs(x, a) + f.subs(x, b)) / 2

for i in range(1, n):
    ii = 2 ** (i - 1)
    sum = 0
    for k in range(1, ii + 1):
        sum += f.subs(x, a + (k - 0.5) * h)
    t[0][i] = t[0][i - 1] / 2 + h / 2 * sum

    for m in range(1, i + 1):
        k = i - m
        t[m][k] = (4**m * t[m - 1][k + 1] - t[m - 1][k]) / (4**m - 1)

    if fabs(t[i][0] - t[i - 1][0]) < e:
        for j in range(i + 1):
            for k in range(i - j + 1):
                print(format(t[j][k], '.8f'), end="\t")
            print()
        print("近似值: ", format(t[i][0], '.8f'))
        sys.exit()

    h = h / 2
```

第四部分：实验结果、结论与讨论

实验结果

问题 1（1）

```
1.35914091      0.88566062      0.76059633      0.72889018      0.72093578
0.72783385      0.71890824      0.71832146      0.71828431
0.71831320      0.71828234      0.71828184
0.71828185      0.71828183
0.71828183
近似值： 0.71828183
```

问题 1（2）

```
5.12182642      9.27976291      10.52055428     10.84204347     10.92309389     10.94339842
10.66574174     10.93415141     10.94920653     10.95011070     10.95016660
10.95204539     10.95021020     10.95017097     10.95017033
10.95018107     10.95017035     10.95017031
10.95017031     10.95017031
10.95017031
近似值： 10.95017031
```

问题 1（3）

```
3.00000000      3.10000000      3.13117647      3.13898849      3.14094161      3.14142989
3.13333333      3.14156863      3.14159250      3.14159265      3.14159265
3.14211765      3.14159409      3.14159266      3.14159265
3.14158578      3.14159264      3.14159265
3.14159267      3.14159265
3.14159265
近似值： 3.14159265
```

问题 1（4）

```
0.75000000      0.70833333      0.69702381      0.69412185      0.69339120
0.69444444      0.69325397      0.69315453      0.69314765
0.69317460      0.69314790      0.69314719
0.69314748      0.69314718
0.69314718
近似值： 0.69314718
```

思考题

二分次数越多，精度越高，关系是近似线性的。

实验报告三 牛顿迭代法

第一部分：问题分析

该实验的目的是通过编写程序使用牛顿迭代法来求解非线性方程的根，牛顿迭代法是一种常用的数值计算方法，可以高效地求解非线性方程的根。

通过这个实验，我们可以了解牛顿迭代法的原理和实现方法，同时也可以掌握编写程序求解非线性方程根的能力。

第二部分：数学原理

求非线性方程 $f(x)=0$ 的根 x^*, 牛顿迭代法计算公式

$$
\begin{aligned}
&x_0=\alpha\\
&x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}\\
&n=0,1,\cdots
\end{aligned}
$$

一般地, 牛顿迭代法具有局部收玫性, 为保证迭代收玫, 要求, 对充分小的 $\delta>0, \alpha \in O\left(x^*, \delta\right)$ 。如果 $f(x) \in C^2[a, b], f\left(x^*\right)=0, f'\left(x^*\right) \neq 0$, 那么, 对充分小的 $\delta>0$, 当 $\alpha \in O\left(x^*, \delta\right)$ 时, 由牛顿迭代法计算出的 $\{x_n\}$ 收玫于 x^*, 且收玫速度是 2 阶的; 如果 $f(x) \in C^m[a, b], f\left(x^*\right)=f'\left(x^*\right)=\cdots=f^{(m-1)}\left(x^*\right)=0, f^{(m)}\left(x^*\right) \neq 0(m>1)$, 那么, 对充分小的 $\delta>0$, 当 $\alpha \in O\left(x^*, \delta\right)$ 时, 由牛顿迭代法计算出的 $\{x_n\}$ 收玫于 x^* , 且收玫速度是 1 阶的;

第三部分：程序设计流程

```
from math import *
from sympy import *
import sys

print("-----------牛顿迭代法-----------")

x = Symbol('x')

FX, x0, e1, e2, N = input("input:\nfunction,x0,epsilon1,epsilon2,N\n").split(",")
FX = sympify(FX)
x0 = float(x0)
e1 = float(e1)
e2 = float(e2)
N = int(N)

n = 1
```

```
while n <= N:
        F = float(FX.subs(x, x0))
        DF = float(diff(FX, x, 1).subs(x, x0))
        if fabs(F) < e1:
                print(x0)
                sys.exit()
        if fabs(DF) < e2:
                print("failed")
                sys.exit()
        x0 = x0 - F / DF
        Tol = fabs(F / DF)
        if Tol < e1:
                print(x0)
                sys.exit()
        n = n + 1
print("failed")
```

第四部分：实验结果、结论与讨论

实验结果

问题 1（1）

```
0.7390851781060086
```

问题 1（2）

```
0.588532742847979
```

问题 2（1）

```
0.5671431650348622
```

问题 2（2）

```
0.5666057041281513
```

思考题

思考题 1 牛顿迭代函数的导函数的绝对值必须小于 1，即必须保证其收敛。在实际运用中，只要选取的初值 x_0 能使 $f'(x) \cdot f''(x) > 0$ 即可。

思考题 2 对于同一方程有不同的迭代形式，次数越高，精度越低。

实验报告四 高斯列主元消元法

第一部分：问题分析

该实验的目的是通过编写程序使用高斯列主元消元法来求解线性方程组。高斯列主元消元法是一种常用的数值计算方法，可以高效地求解线性方程组。

通过这个实验，我们可以了解高斯列主元消元法的原理和实现方法，同时也可以掌握编写程序求解线性方程组的能力。

第二部分：数学原理

高斯 (Gauss) 列主元消去法：对给定的 n 阶线性方程组 $Ax = b$，首先进行列主元消元过程，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，在进行列主元消元过程前，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。这就是所谓的平衡技术。然后再进行列主元消元过程。

第三部分：程序设计流程

```
from numpy import *
import sys

print("---------高斯消元法---------")

n = int(input("input:\nn,a[i,j],b[i]\n"))

a = zeros((n + 1, n + 1))
for i in range(n):
    row = input().split()
    for j in range(n):
        a[i + 1][j + 1] = float(row[j])

b = [0.0] * (n + 1)
row = input().split()
for i in range(n):
    b[i + 1] = float(row[i])

for k in range(1, n):
    max_a = fabs(a[n, k])
    p = n
    for j in range(k, n):
```

```
        # 寻找待操作的行p
        if a[j, k] > max_a:
            max_a = a[j, k]
            p = j
    if max_a == 0:
        print("singular!")
        sys.exit()

    if p != k:
        # 交换p,k两行
        a[[p, k], :] = a[[k, p], :]
        b[p], b[k] = b[k], b[p]

    # 计算
    for i in range(k + 1, n + 1):
        m = a[i, k] / a[k, k]
        for j in range(k, n + 1):
            a[i, j] = a[i, j] - a[k, j] * m
        b[i] = b[i] - b[k] * m

if a[n, n] == 0:
    print("singular!")
    sys.exit()

# 求解x
x = [0.0] * (n + 1)
x[n] = b[n] / a[n, n]
for k in range(n - 1, 0, -1):
    sum = 0
    for j in range(k + 1, n + 1):
        sum += a[k, j] * x[j]
    x[k] = (b[k] - sum) / a[k, k]

# 输出x
for i in range(1, n + 1):
    print(x[i])
```

第四部分：实验结果、结论与讨论

实验结果

问题 1（1）

```
1.0000000000000027
1.000000000000002
0.9999999999999971
0.9999999999999992
```

问题 1（2）

```
1.000000000000133
0.9999999999998009
0.999999999999888
1.0000000000000293
```

问题 1（3）

```
0.9999999999999596
1.000000000000441
0.9999999999989548
1.0000000000006732
```

问题 1（4）

```
1.0
1.0
1.0
1.0
```

问题 2（1）

```
0.9536791069017717
0.3209568455211036
1.0787080757932384
-0.09010850953957895
```

问题 2（2）

```
0.5161772979585416
0.41521947283013527
0.10996610286788916
1.0365392233362005
```

问题 2（3）

```
1.0
0.9999999999999998
1.0000000000000002
```

问题 2（4）

```
1.0
1.0
1.0000000000000002
```