# 数值计算实验大报告

## 目录

# 实验一 拉格朗日插值法（2 课时）

## 一、实验目的

1. 了解拉格朗日插值法的基本原理和方法。
2. 掌握拉格朗日插值多项式的用法。

## 二、实验要求

1. 用 C 语言编程实现拉格朗日插值。
2. 进一步加深对拉格朗日插值法的理解。

## 三、实验原理

### （一）拉格朗日插值公式

经过 $(n+1)$ 个点，$(x_0,y_0),(x_1,y_1)\cdots,(x_n,y_n)$，构造一个 n 次多项式，形如：

$$p_n(x)=\sum_{k=0}^{n} y_k l_k(x)$$

使得 $p_n(x_k)=y_k \quad (k=0,1,2\cdots,n)$ 成立。

其中 $l_k(x)=\dfrac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}=\begin{cases}0 & x=x_k\\ 1 & x\neq x_k\end{cases}$

为插值基函数。

## （二）例子

已知 $f(x)$ 满足 $f(144)=12, f(169)=13, f(225)=15$ 作 $f(x)$ 的二次拉格朗日插值多项式，并求 $f(175)$ 的近似值。

解：设 $x_0=144, x_1=169, x_2=225$，$y_0=12, y_1=13, y_2=15$，则 $f(x)$ 的二次拉格朗日插值基函数为：

$$l_0(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}=\frac{(x-169)(x-225)}{2025}$$

$$l_1(x)=\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}=\frac{(x-144)(x-225)}{-1400}$$

$$l_2(x)=\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}=\frac{(x-144)(x-169)}{4536}$$

因此 $f(x)$ 的二次拉格朗日插值多项式为 $L_2(x)=y_0l_0(x)+y_1l_1(x)+y_2l_2(x)$；

且 $f(175)\approx L_2(175)=12l_0(175)+13l_1(175)+15l_2(175)=13.230\,158\,73$。

# 四、实验内容

## （一）算法流程图

如果程序的数据输入项(函数参数)为:插值节点及函数值，及待求点 x 的值;输出为待求点 x 对应的函数值，则程序流程图如图 1-1。

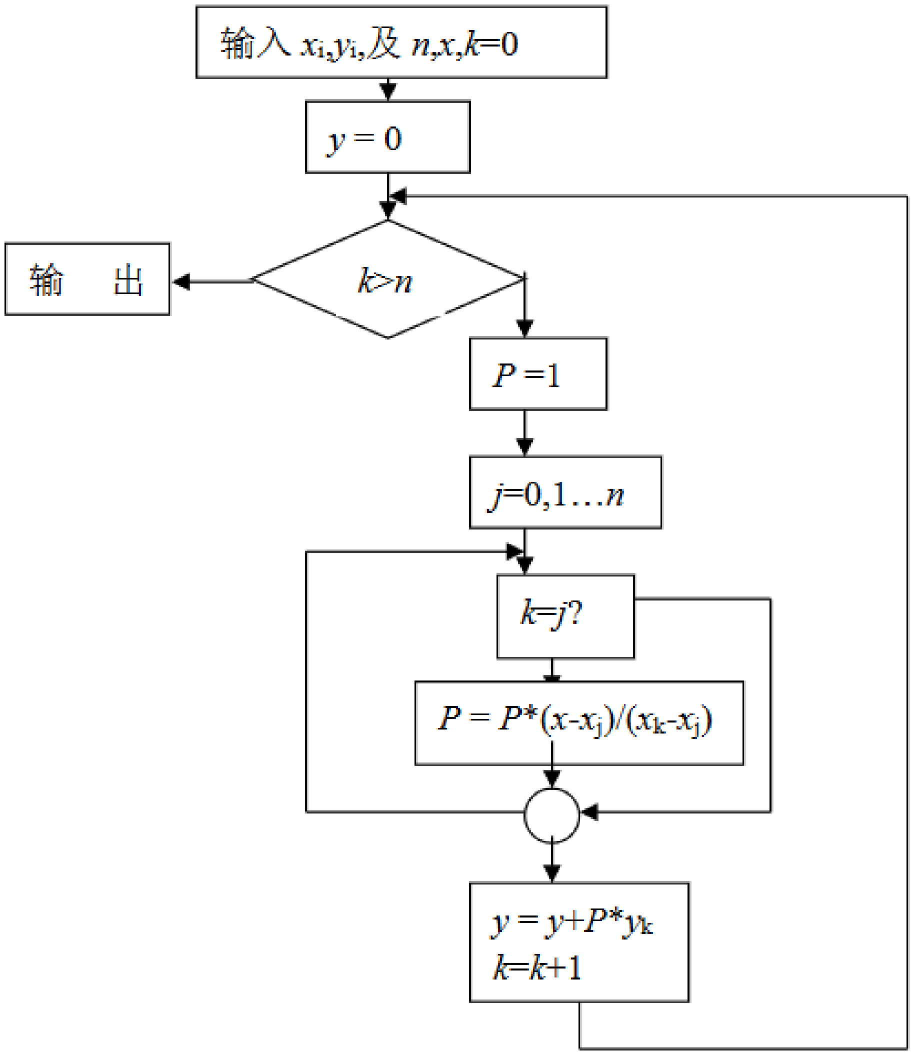



图 1-1 拉格朗日插值法算法流程图

## （二）编程作业

编写拉格朗日插值法通用子程序，并用以下函数表来上机求 f(0.15),f(0.31)。

| x | 0.0 | 0.1 | 0.195 | 0.3 | 0.401 | 0.5 |
|---|---|---|---|---|---|---|
| f(x) | 0.39894 | 0.39695 | 0.39142 | 0.38138 | 0.36812 | 0.35206 |

代码如下:

```
/ * C 语言源代码 */
#include <stdio.h>
main()
{
    static float Lx[10],Ly[10];
```

```
        int n,i,j;
        float x,y,p;
        printf("enter n=");
        scanf("%d",&n);                  //______________
        printf("enter xi\n");
        for(i=1;i<=n;i++)
            scanf("%f",&Lx[i]);
        printf("enter yi\n");
        for(i=1;i<=n;i++)
            scanf("%f",&Ly[i]);
        printf("enter x=");
        scanf("%f",&x);
            for( ______________)
            { __________;
                for( ____________)
                { if(i!=j)
                    __________________; //______________
                }
                ____________________;
            }
        printf("y=%f\n",y);
    }
```

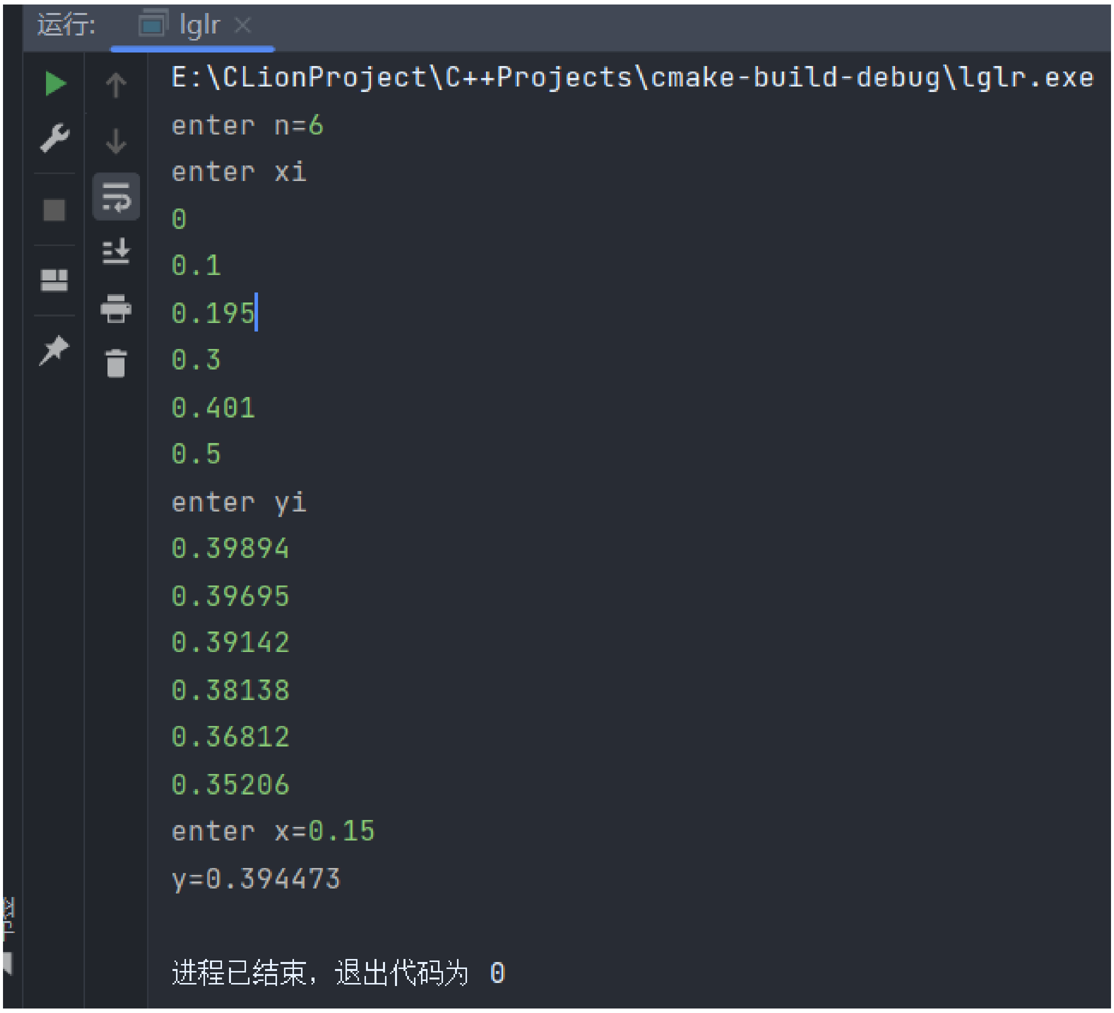
请完成这个程序，并在两处注释处写上正确的注释。在执行程序时，如果求 )15.0(f 的值，那么屏幕上应出现如下内容：

```
6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
```

0.36812
0.35206
enter x=0.15
y=0.

```c
#include <stdio.h>

int main() {
     // 输入
    static float Lx[10], Ly[10];
    int n, i, j;
    float x, y = 0, p;
    printf("enter n=");
    fflush(stdout);
    scanf("%d", &n); //数据点数量
     printf("enter xi\n");
    fflush(stdout);
    for (i = 1; i <= n; i++) scanf("%f", &Lx[i]);
    printf("enter yi\n");
    fflush(stdout);
    for (i = 1; i <= n; i++) scanf("%f", &Ly[i]);
     // 拟合
    printf("enter x=");
    fflush(stdout);
    scanf("%f", &x);
    for (i = 1; i <= n; i++) {
        p = 1;
        for (j = 1; j <= n; j++) {
            if (i != j) {
                // l_k(x) = (x-x[1])...(x-x[k-1])(x-x[k+1])..(x-x[n]) / (x[k]-
x[1])...(x[k]-x[k-1])(x[k]-x[k+1])...(x[k]-x[n])
                p *= (x - Lx[j]) / (Lx[i] - Lx[j]);
            }
        }
        y += p * Ly[i];// f(x) = SUM{ l_k(x) * y_k}
    }
    printf("y=%f\n", y);
}
```

运行效果:


运行:
lglr
E:\CLionProject\C++Projects\cmake-build-debug\lglr.exe
enter n=6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
0.36812
0.35206
enter x=0.15
y=0.394473
进程已结束，退出代码为 0

```
运行:    lglr ×
E:\CLionProject\C++Projects\cmake-build-debug\lglr.exe
enter n=6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
0.36812
0.35206
enter x=0.31
y=0.380219

进程已结束，退出代码为 0
```

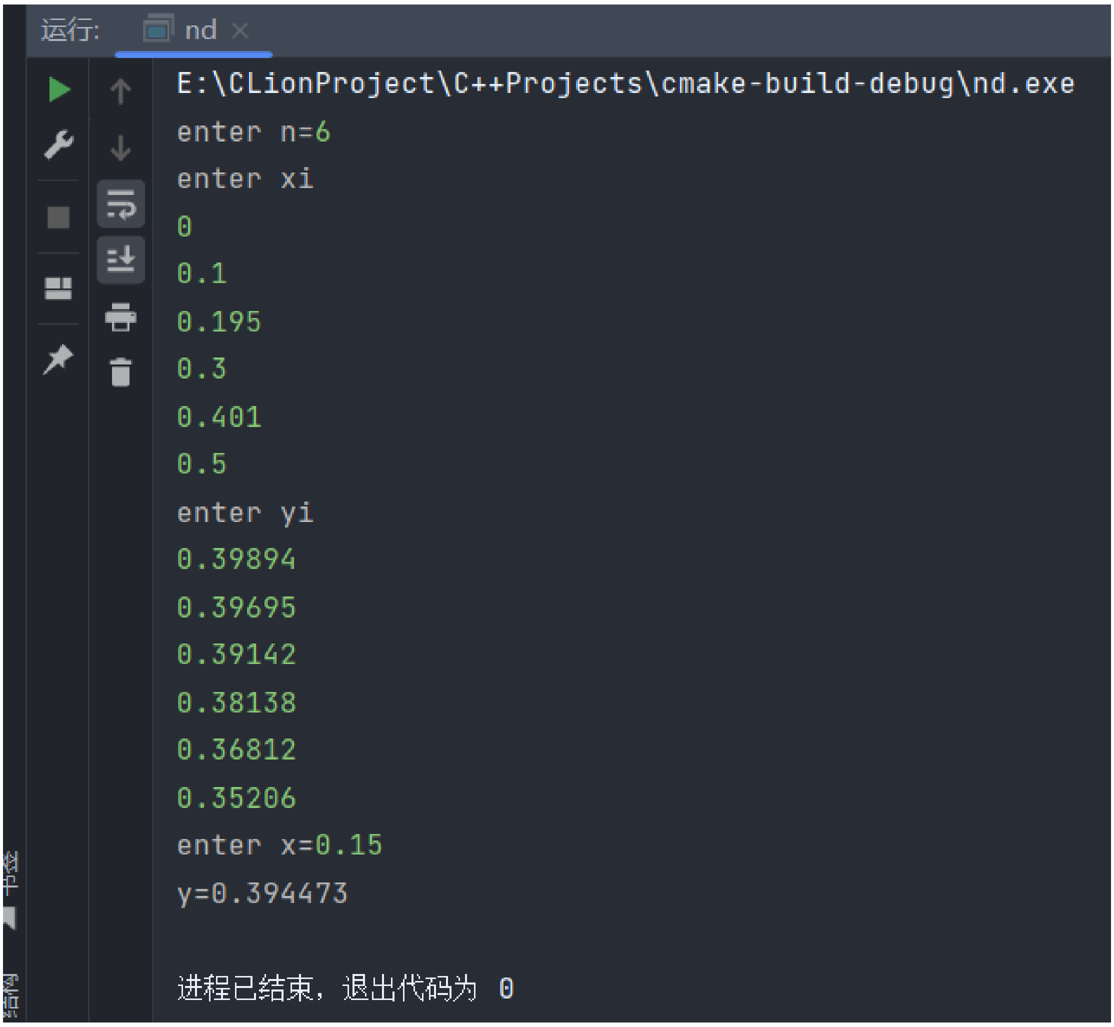
拟合 x=0. 15 结果为 y=0. 394473
拟合 x=0. 31 结果为 y=0. 380219
对比给出的数据点，结果较为可信

## （三）选做题

参考教材牛顿插值公式，编程实现用牛顿插值公式求上述条件下对应节点的函数值。

```c
#include <stdio.h>

int main() {
    static float Lx[10], Ly[10];
    int n, i, j;
    float x, y, p = 1;
    printf("enter n=");
    fflush(stdout);
    scanf("%d", &n); //数据点数量
    printf("enter xi\n");
```

```
    fflush(stdout);
    for (i = 1; i <= n; i++) scanf("%f", &Lx[i]);
    printf("enter yi\n");
    fflush(stdout);
    for (i = 1; i <= n; i++) scanf("%f", &Ly[i]);
    printf("enter x=");
    fflush(stdout);
    scanf("%f", &x);
    y = Ly[1];
    float Favg[10][10] = {0};//均差表
    for (i = 1; i <= n; i++) {
        Favg[0][i] = Ly[i];
    }
    for (i = 1; i < n; i++) {
        p *= (x - Lx[i]);// p[i] = (x-x[0])(x-x[1])...(x-x[i])
        for (j = i + 1; j <= n; j++) {//i 阶均差
            Favg[i][j] = (Favg[i - 1][j] - Favg[i - 1][j - 1]) / (Lx[j] - Lx[j -
i]);
        }
        y += Favg[i][i + 1] * p; // f(x) = SUM{ f[x[0],x[k]]*(x-x[0])…(x-x[k]) |
1<=k<=n }
    }
    printf("y=%f\n", y);
}
```
运行结果:


运行:
nd
E:\CLionProject\C++Projects\cmake-build-debug\nd.exe
enter n=6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
0.36812
0.35206
enter x=0.15
y=0.394473
进程已结束，退出代码为 0

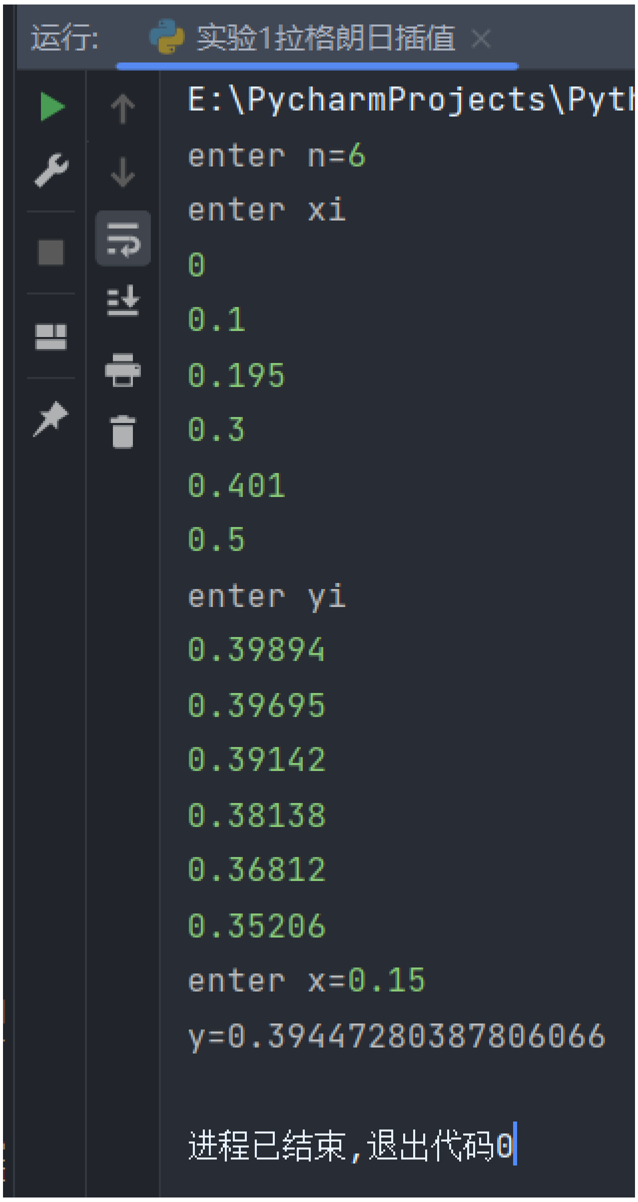
```
运行:    nd ×
E:\CLionProject\C++Projects\cmake-build-debug\nd.exe
enter n=6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
0.36812
0.35206
enter x=0.31
y=0.380219

进程已结束，退出代码为 0
```

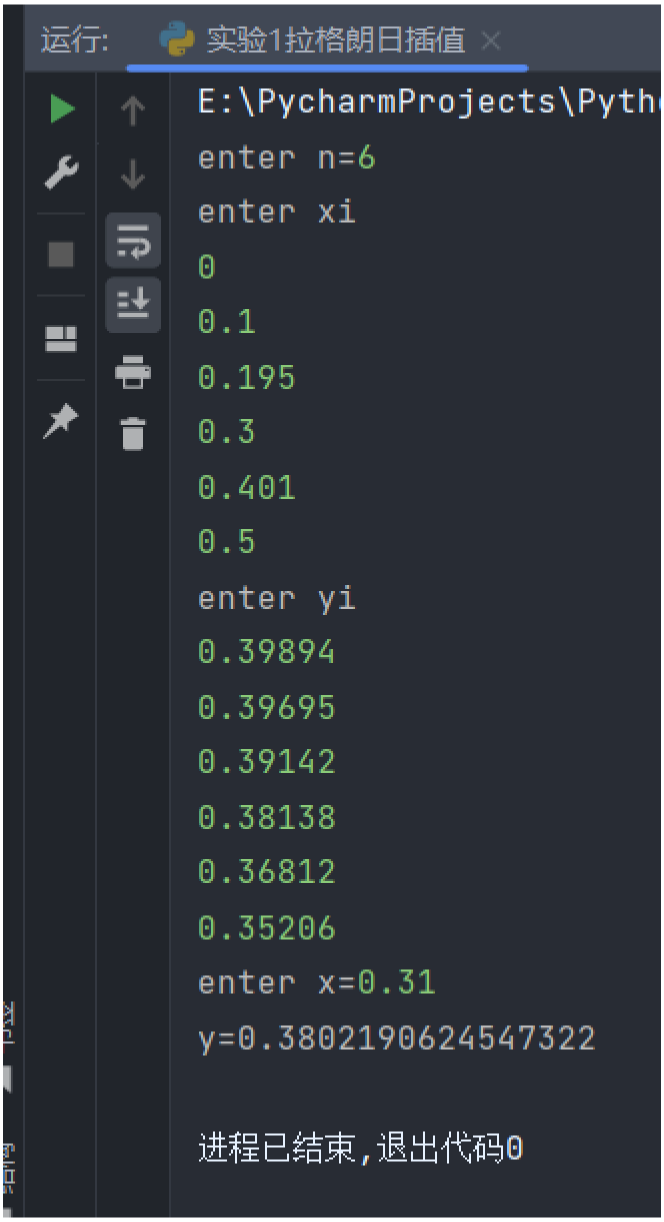
(四)使用其他编程语言(Python、Java)

**python:**

```python
def lagrange_interpolation(n, x_values, y_values, x):
    y = 0
    for i in range(n):
        p = 1
        for j in range(n):
            if i != j:
                p *= (x - x_values[j]) / (x_values[i] - x_values[j])
        y += p * y_values[i]
    return y
```

```python
n = int(input("enter n="))
x_values = []
y_values = []
print("enter xi")
for i in range(n):
    x_values.append(float(input()))
print("enter yi")
for i in range(n):
    y_values.append(float(input()))
x = float(input("enter x="))
y = lagrange_interpolation(n, x_values, y_values, x)
print("y={}".format(y))
```

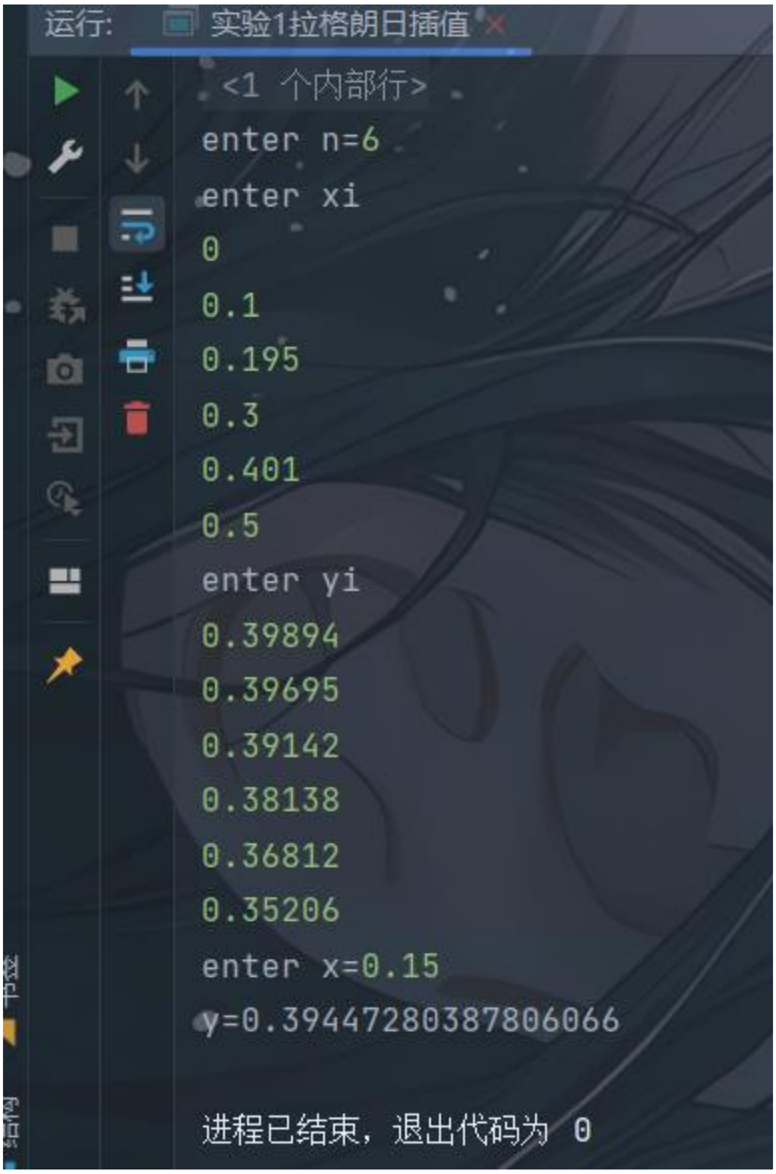


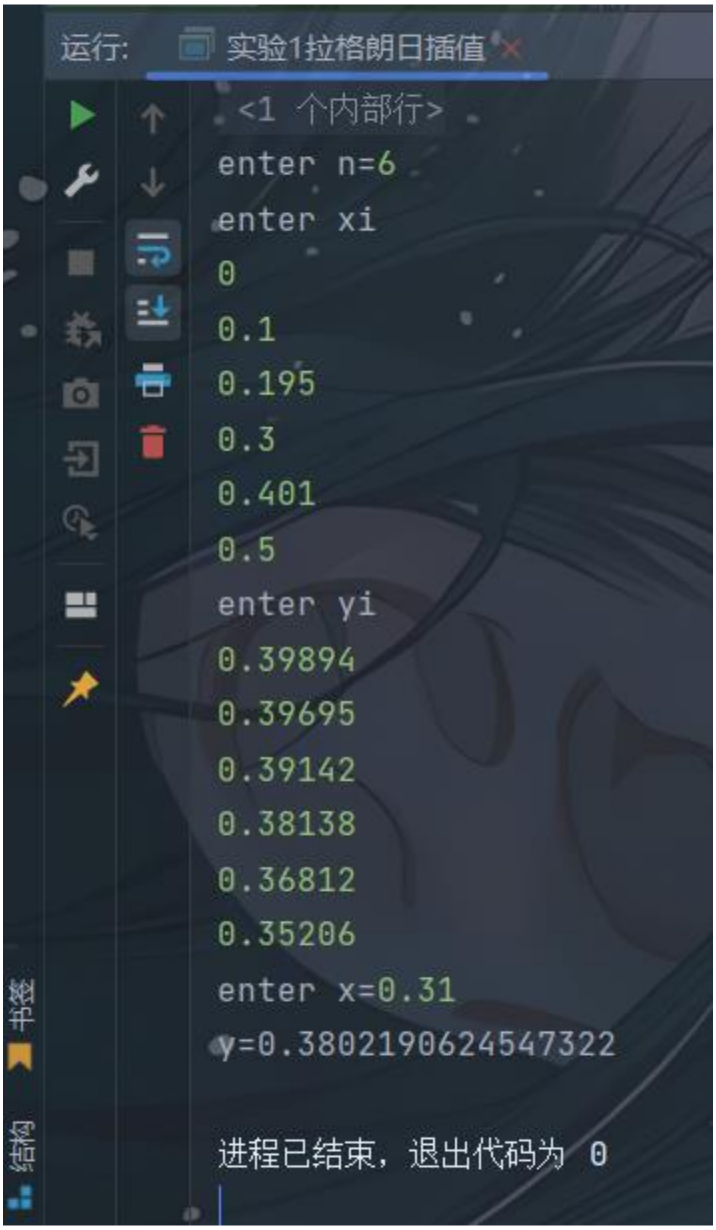



## java:

```java
package 数值计算实验;

import java.util.*;

public class 实验 1 拉格朗日插值 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("enter n=");
        int n = sc.nextInt();
```

```
        double[] xValues = new double[n], yValues = new double[n];
        System.out.println("enter xi");
        for (int i = 0; i < n; i++) xValues[i] = sc.nextDouble();
        System.out.println("enter yi");
        for (int i = 0; i < n; i++) yValues[i] = sc.nextDouble();

        System.out.print("enter x=");
        double x = sc.nextDouble();
        double y = lagrangeInterpolation(n, xValues, yValues, x);
        System.out.println("y=" + y);
    }

    public static double lagrangeInterpolation(int n, double[]
xValues, double[] yValues, double x) {
        double y = 0;
        for (int i = 0; i < n; i++) {
            double p = 1;
            for (int j = 0; j < n; j++) {
                if (i != j) {
                    p *= (x - xValues[j]) / (xValues[i] -
xValues[j]);
                }
            }
            y += p * yValues[i];
        }
        return y;
    }
}
```


运行:
实验1拉格朗日插值
<1 个内部行>
enter n=6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
0.36812
0.35206
enter x=0.15
y=0.39447280387806066
进程已结束，退出代码为 0


运行:
实验1拉格朗日插值
<1 个内部行>
enter n=6
enter xi
0
0.1
0.195
0.3
0.401
0.5
enter yi
0.39894
0.39695
0.39142
0.38138
0.36812
0.35206
enter x=0.31
y=0.3802190624547322
进程已结束，退出代码为 0
书签
结构

# 实验二 最小二乘法（4 课时）

## 一、实验目的

1. 了解最小二乘拟合的基本原理和方法，注意与插值方法的区别。
2. 掌握最小二乘法。

## 二、实验要求

1. 掌握用 C 语言作最小二乘多项式拟合的方法。
2. 进一步加深对最小二乘法的理解。

## 三、实验原理

### （一）最小二乘多项式拟合

已知数据对 $(x_{j,}y_j)(j=1,2,\cdots,n)$，求多项式

$$P(x)=\sum_{i=0}^{m}a_i x^i (m<n)$$

使得 $\Phi(a_0,a_1,\cdots,a_n)=\sum_{j=1}^{n}\left(\sum_{i=0}^{m}a_i x_j{}^i-y_j\right)^2$ 为最小，这就是一个最小二乘问题。

## （二）法方程组

$$\begin{bmatrix} m & \sum_{i=1}^{m} x_i & \cdots & \sum_{i=1}^{m} x_i^n \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 & \cdots & \sum_{i=1}^{m} x_i^{n+1} \\ \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{m} x_i^n & \sum_{i=1}^{m} x_i^{n+1} & \cdots & \sum_{i=1}^{m} x_i^{2n} \end{bmatrix} \begin{bmatrix} \mathrm{a}_0 \\ \mathrm{a}_1 \\ \cdots \\ \mathrm{a}_\mathrm{n} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \\ \cdots \\ \sum_{i=1}^{m} x_i^n y_i \end{bmatrix}$$

## （三）最小二乘法计算步骤

用线性函数 $P(x) = a + bx$ 为例，拟合给定数据 $(x_{i,} y_i)(i = 1,2,\cdots,m)$ 。

算法描述：

步骤 1：输入 $m$ 值，及 $(x_{i,} y_i)(i = 1,2,\cdots,m)$ 。

步骤 2：建立法方程组 $A^T AX = AY$ 。

步骤 3：解法方程组。

步骤 4：输出 $P(x) = a + bx$ 。

# 四、实验内容

## （一）算法流程图

1. 算法整体流程图

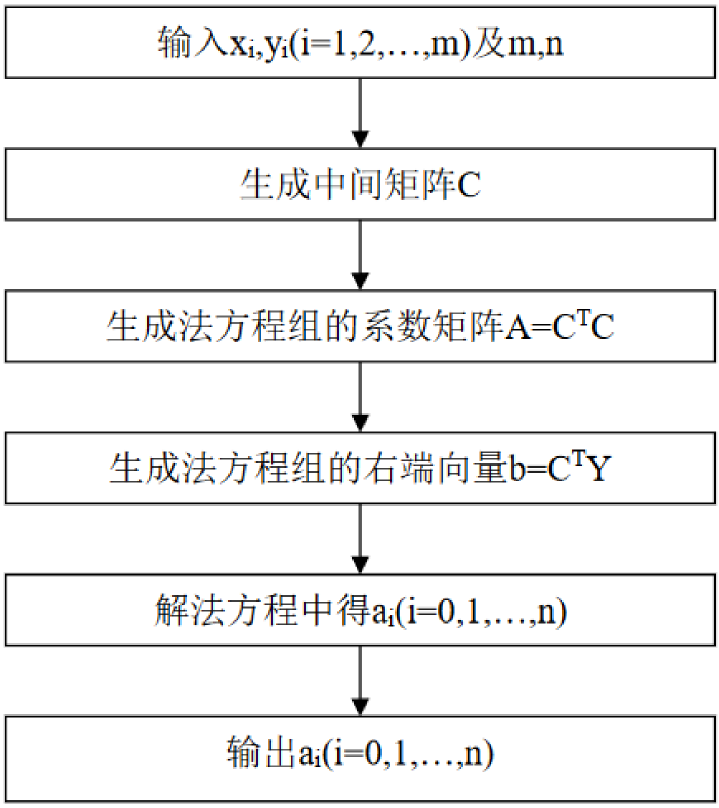



图 2-1　最小二乘法算法整体流程图

2. “生成中间矩阵 C” 算法流程图

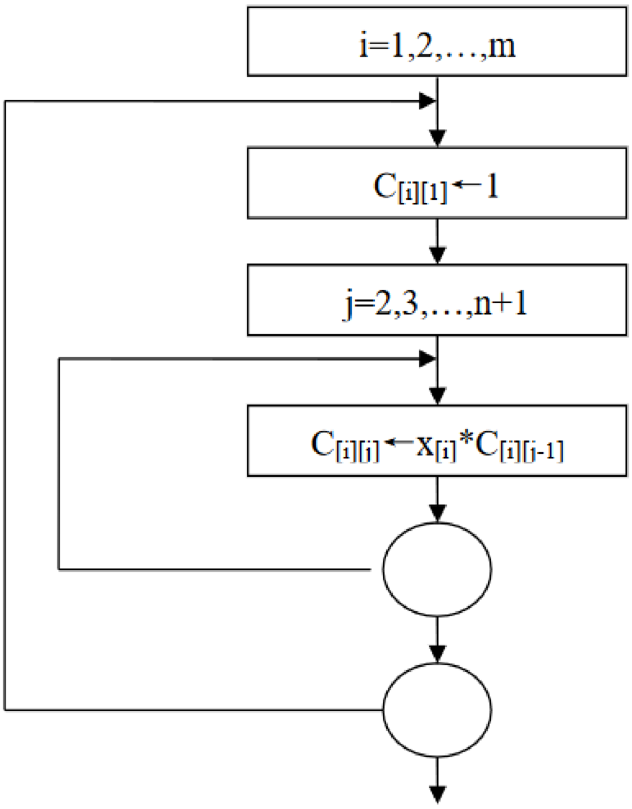



图 2-2 最小二乘法中“生成中间矩阵 C”算法流程图

## 3. 中间矩阵 C 的重要作用

$$
C=\begin{bmatrix} 1 & x_1 & x_1^{\ 2} & \dots & x_1^{\ n} \\ 1 & x_2 & x_2^{\ 2} & \dots & x_2^{\ n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_m & x_m^{\ 2} & \dots & x_m^{\ n} \end{bmatrix},\quad Y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}
$$

$$
\text{则 } C^T=\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_m \\ x_1^{\ 2} & x_2^{\ 2} & \dots & x_2^{\ 2} \\ \vdots & \vdots & \dots & \vdots \\ x_1^{\ n} & x_2^{\ n} & \dots & x_m^{\ n} \end{bmatrix},
$$

$$
A=C^TC=\begin{bmatrix} m & \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^{\ 2} & \dots & \sum_{i=1}^{m} x_i^{\ n} \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i^3 & \dots & \sum_{i=1}^{m} x_i^{n+1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \sum_{i=1}^{m} x_i^n & \sum_{i=1}^{m} x_i^{n+1} & \sum_{i=1}^{m} x_i^{n+2} & \dots & \sum_{i=1}^{m} x_i^{2n} \end{bmatrix},
$$

$$
b=C^TY=\begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \\ \sum_{i=1}^{m} x_i^2 y_i \\ \vdots \\ \sum_{i=1}^{m} x_i^n y_i \end{bmatrix}
$$

## （二）编程作业

测得铜导线在温度 Ti （℃)时的电阻 Ri 如下表，求电阻 R 与温度 T 的近似函数关系

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *Ti*(℃) | 19.1 | 25.0 | 30.1 | 36.0 | 40.0 | 45.1 | 50.0 |
| *Ri*(Ω) | 76.30 | 77.80 | 79.25 | 80.80 | 82.35 | 83.90 | 85.10 |

【提示】在进行程序实现时，务必注意中间矩阵的作用，以及非奇次线性方程组求解问题！

为了实验的顺利完成，此处给出解非奇次线性方程组的高斯消元法的函数。请认真阅读并理解。

```
float gs(float a[20][20],float b[20],int n )
{int i,j,k,l;
 float s;
  k=1;
     while(k!=n+1)
     {
         if(a[k][k]!=0)
        {
             for(i=k+1;i<=n+1;i++)
            {
                 a[i][k]=a[i][k]/a[k][k];
                 b[i]=b[i]-a[i][k]*b[k];
                 for(j=k+1;j<=n+1;j++)
                    a[i][j]=a[i][j]-a[i][k]*a[k][j];
            }
        }
         k=k+1;
     }
     for(k=n+1;k>=1;k--)
     {
         s=0;
         for(l=k+1;l<=n+1;l++)
             s=s+a[k][l]*b[l];
         b[k]=(b[k]-s)/a[k][k];
      }
 return 0;
}
```

实验主程序如下（请加上必要的注释）。

```
#include <bits/stdc++.h>

using namespace std;
int n, m;
double x[10], y[10];

/*
7 1
19.1 76.30
25.0 77.80
30.1 79.25
```

```
36.0 80.80
40.0 82.35
45.1 83.90
50.0 85.10
 */
int main() {
    // 输入
    cin >> m >> n;
    for (int i = 1; i <= m; i++) {
        cin >> x[i] >> y[i];
    }
    // 生成中间矩阵 C
    double C[m + 1][n + 2];
    for (int i = 1; i <= m; i++) {
        C[i][1] = 1;
        for (int j = 2; j <= n + 1; j++) {
            C[i][j] = x[i] * C[i][j - 1];
        }
    }
    // 生成法方程组系数矩阵 A = C^T * C
    double A[n + 2][n + 2];
    for (int i = 1; i <= n + 1; i++) {
        for (int j = 1; j <= n + 1; j++) {
            for (int k = 1; k <= m; k++) {
                A[i][j] += C[k][i] * C[k][j];//C^T * C
            }
        }
    }
    // 生成法方程组右端向量 b = C^T * Y
    double b[n + 2];
    for (int i = 1; i <= n + 1; i++) {
        for (int k = 1; k <= m; k++) {
            b[i] += C[k][i] * y[k];//C^T * Y
        }
    }
    // 使用高斯消元法解非齐次线性方程组 AX=b
    for (int k = 1; k <= n + 1; k++) {
        if (A[k][k] == 0) continue;
        for (int i = k + 1; i <= n + 1; i++) {
            A[i][k] = A[i][k] / A[k][k];
            b[i] = b[i] - A[i][k] * b[k];
            for (int j = k + 1; j <= n + 1; j++)
                A[i][j] = A[i][j] - A[i][k] * A[k][j];
        }
```

```
    }
    for (int k = n + 1; k >= 1; k--) {
        double s = 0;
        for (int l = k + 1; l <= n + 1; l++)
            s = s + A[k][l] * b[l];
        b[k] = (b[k] - s) / A[k][k];
    }
    // 输出结果
    cout << "a = ";
    for (int i = 1; i <= n + 1; i++) {
        cout << b[i] << ", ";
    }
    // 对结果进行验算,计算误差
    cout << "\nR(t) = " << b[1] << " + " << b[2] << "t\n";
    double diff = 0;
    for (int i = 1; i <= m; i++) {
        double r = b[1] + b[2] * x[i];
        double d = abs(r - y[i]) * 100 / y[i];
        cout << "R(" << x[i] << ") = " << r << " ;\t 相对误差为" << d << "%\n";
        diff += d;
    }
    cout << "平均误差:" << diff / m << "%\n";
}
```
拟合结果:

```
7 1
19.1 76.30
25.0 77.80
30.1 79.25
36.0 80.80
40.0 82.35
45.1 83.90
50.0 85.10
a = 70.5723, 0.291456,
R(t) = 70.5723 + 0.291456t
R(19.1) = 76.1391 ;  相对误差为0.210905%
R(25) = 77.8587 ;    相对误差为0.075408%
R(30.1) = 79.3451 ;  相对误差为0.119989%
R(36) = 81.0647 ;    相对误差为0.327573%
R(40) = 82.2305 ;    相对误差为0.145111%
R(45.1) = 83.7169 ;  相对误差为0.218206%
R(50) = 85.1451 ;    相对误差为0.0529462%
平均误差:0.164305%

进程已结束，退出代码为 0
```

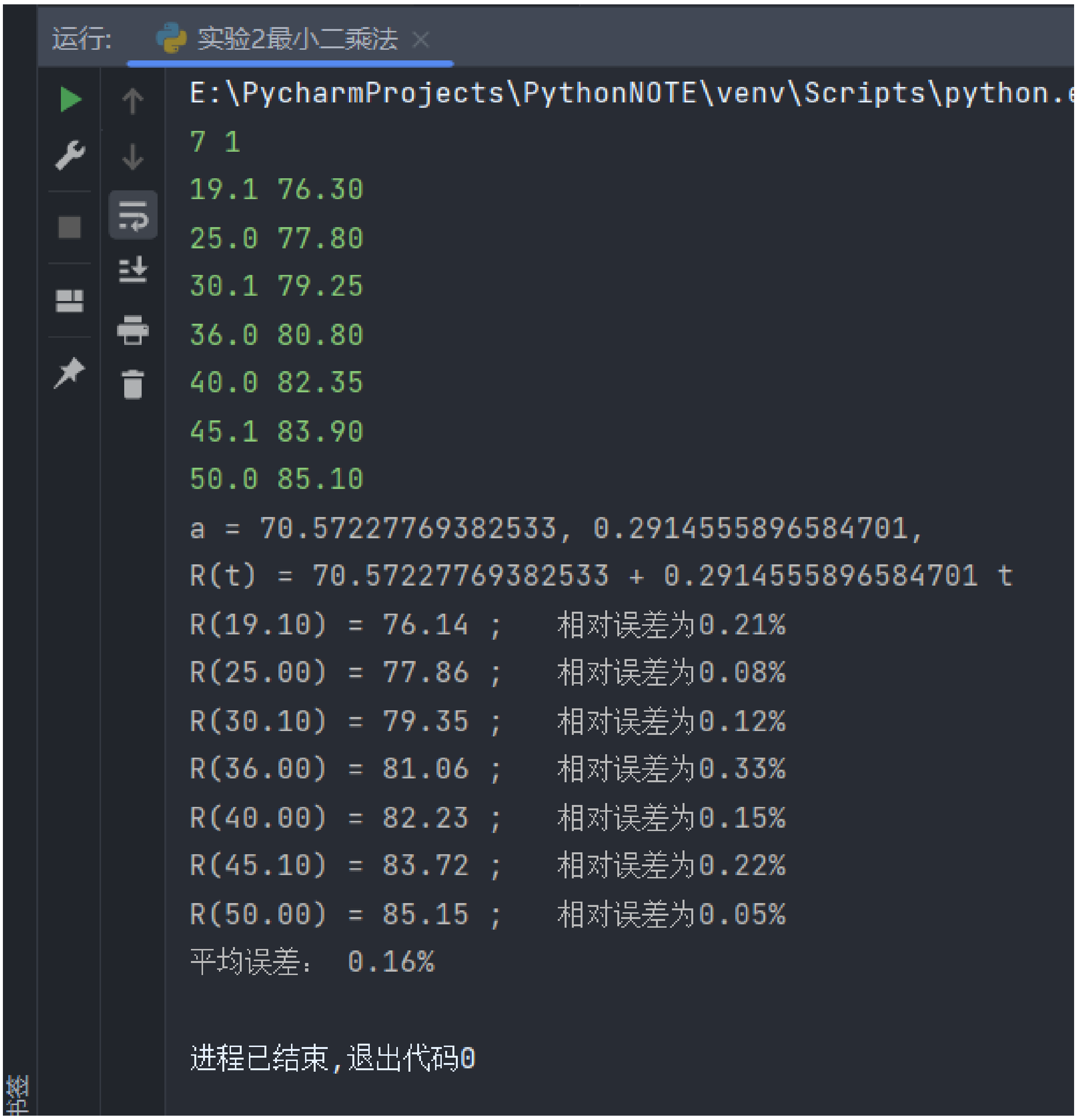
(三) 使用使用其他编程语言(Python、Java)

**python:**

```python
import numpy as np

# 输入
m, n = map(int, input().split())
x = [0] * (m + 1)
y = [0] * (m + 1)
for i in range(1, m + 1):
    x[i], y[i] = map(float, input().split())
```

```python
# 生成中间矩阵 C
C = np.zeros((m + 1, n + 2))
for i in range(1, m + 1):
    C[i][1] = 1
    for j in range(2, n + 2):
        C[i][j] = x[i] * C[i][j - 1]

# 生成法方程组系数矩阵 A = C^T * C
A = np.zeros((n + 2, n + 2))
for i in range(1, n + 2):
    for j in range(1, n + 2):
        for k in range(1, m + 1):
            A[i][j] += C[k][i] * C[k][j]

# 生成法方程组右端向量 b = C^T * Y
b = np.zeros(n + 2)
for i in range(1, n + 2):
    for k in range(1, m + 1):
        b[i] += C[k][i] * y[k]

# 使用高斯消元法解非齐次线性方程组 AX=b
for k in range(1, n + 2):
    if A[k][k] == 0:
        continue
    for i in range(k + 1, n + 2):
        A[i][k] = A[i][k] / A[k][k]
        b[i] = b[i] - A[i][k] * b[k]
        for j in range(k + 1, n + 2):
            A[i][j] = A[i][j] - A[i][k] * A[k][j]

for k in range(n + 1, 0, -1):
    s = sum(A[k][l] * b[l] for l in range(k + 1, n + 2))
    b[k] = (b[k] - s) / A[k][k]

# 输出结果
print("a =", end=" ")
for i in range(1, n + 2):
    print(b[i], end=", ")
print("\nR(t) =", b[1], "+", b[2], "t")

# 对结果进行验算,计算误差
diff = 0
for i in range(1, m + 1):
```

```
    r = b[1] + b[2] * x[i]
    d = abs(r - y[i]) * 100 / y[i]
    print("R({:.2f}) = {:.2f} ;\t 相对误差为{:.2f}%".format(x[i], r, d))
    diff += d
print("平均误差: {:.2f}%".format(diff / m))
```

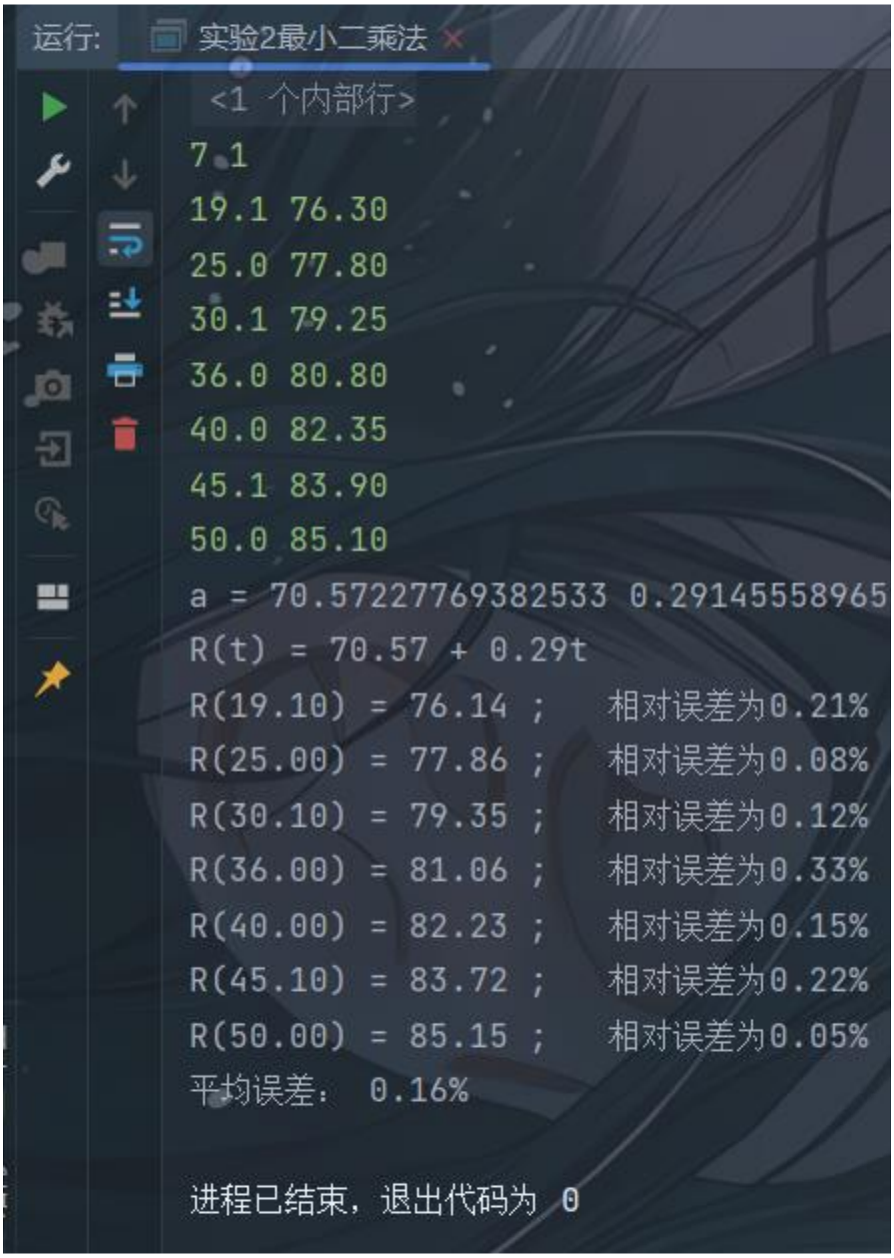



## java:

```
package 数值计算实验;

import java.util.Scanner;

public class 实验 2 最小二乘法 {
    public static void main(String[] args) {
```

```java
    // 输入
    Scanner sc = new Scanner(System.in);
    int m = sc.nextInt();
    int n = sc.nextInt();
    double[] x = new double[m + 1];
    double[] y = new double[m + 1];

    for (int i = 1; i <= m; i++) {
        x[i] = sc.nextDouble();
        y[i] = sc.nextDouble();
    }
    // 生成中间矩阵 C
    double[][] C = new double[m + 1][n + 2];

    for (int i = 1; i <= m; i++) {
        C[i][1] = 1;
        for (int j = 2; j <= n + 1; j++) {
            C[i][j] = x[i] * C[i][j - 1];
        }
    }
    // 生成法方程组系数矩阵 A = C^T * C
    double[][] A = new double[n + 2][n + 2];

    for (int i = 1; i <= n + 1; i++) {
        for (int j = 1; j <= n + 1; j++) {
            for (int k = 1; k <= m; k++) {
                A[i][j] += C[k][i] * C[k][j];
            }
        }
    }
    // 生成法方程组右端向量 b = C^T * Y
    double[] b = new double[n + 2];

    for (int i = 1; i <= n + 1; i++) {
        for (int k = 1; k <= m; k++) {
            b[i] += C[k][i] * y[k];
        }
    }
    // 使用高斯消元法解非齐次线性方程组 AX=b
    for (int k = 1; k <= n + 1; k++) {
        if (A[k][k] == 0) continue;
        for (int i = k + 1; i <= n + 1; i++) {
            A[i][k] = A[i][k] / A[k][k];
            b[i] = b[i] - A[i][k] * b[k];
```

```
                for (int j = k + 1; j <= n + 1; j++) {
                    A[i][j] = A[i][j] - A[i][k] * A[k][j];
                }
            }
        }

        for (int k = n + 1; k >= 1; k--) {
            double s = 0;
            for (int l = k + 1; l <= n + 1; l++) {
                s += A[k][l] * b[l];
            }
            b[k] = (b[k] - s) / A[k][k];
        }
        // 输出结果
       System.out.print("a =");
        for (int i = 1; i <= n + 1; i++) {
            System.out.print(" " + b[i]);
        }
        System.out.println();
        System.out.printf("R(t) = %.2f + %.2ft\n", b[1], b[2]);

        // 对结果进行验算,计算误差
       double diff = 0;
        for (int i = 1; i <= m; i++) {
            double r = b[1] + b[2] * x[i];
            double d = Math.abs(r - y[i]) * 100 / y[i];
            System.out.printf("R(%.2f) = %.2f ;\t 相对误差为%.2f%%\n",
x[i], r, d);
            diff += d;
        }
        System.out.printf("平均误差: %.2f%%\n", diff / m);
    }
}
```


运行:
实验2最小二乘法
<1 个内部行>
7.1
19.1 76.30
25.0 77.80
30.1 79.25
36.0 80.80
40.0 82.35
45.1 83.90
50.0 85.10
a = 70.57227769382533 0.29145558965
R(t) = 70.57 + 0.29t
R(19.10) = 76.14 ;   相对误差为0.21%
R(25.00) = 77.86 ;   相对误差为0.08%
R(30.10) = 79.35 ;   相对误差为0.12%
R(36.00) = 81.06 ;   相对误差为0.33%
R(40.00) = 82.23 ;   相对误差为0.15%
R(45.10) = 83.72 ;   相对误差为0.22%
R(50.00) = 85.15 ;   相对误差为0.05%
平均误差:  0.16%
进程已结束，退出代码为 0

# 实验三 数值积分(2 课时)

## 一、实验目的

1. 了解数值积分的基本原理和方法。
2. 掌握复合梯形公式。
3. 了解求积公式外推思想、Romberg 公式及 Romberg 积分法。

## 二、实验要求

1. 编写定步长复合梯形公式
2. 编写变步长复合梯形公式。
3. 进一步加深对数值积分的理解。

## 三、实验原理

### (一)定步长复合梯形公式

### 1. 公式

将积分区间$[a,b]$ $n$等分，分点为$x_i = a + ih$，$i = 0,1,\cdots,n$，其中$h = \dfrac{b-a}{n}$称为积分步长。

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

### 2. 例子

用复合梯形公式求积分 $\pi = \int_0^1 \frac{4}{1+x^2}dx$ 的近似值。(取 8 位小数，精确解为 3.14159265 )

$$\pi \approx T_8 = \frac{1}{16}\{f(0) + 2[f(\frac{1}{8}) + f(\frac{1}{4}) + f(\frac{3}{8}) + f(\frac{1}{2})$$
$$+ f(\frac{5}{8}) + f(\frac{3}{4}) + f(\frac{7}{8})] + f(1)\} = 3.138988$$

## （二）变步长复合梯形公式

### 1. 公式

$$T_{2n}=\frac{1}{2}T_n+\frac{b-a}{2n}\sum_{k=1}^{n}f\left[a+(2k-1)\frac{b-a}{2n}\right]$$

递推公式：

$$\begin{cases} T_1=\dfrac{b-a}{2}[f(a)+f(b)] \\ T_{2^k}=\dfrac{1}{2}T_{2^{k-1}}+\dfrac{b-a}{2^k}\displaystyle\sum_{i=1}^{2^{k-1}}f\left[a+(2i-1)\dfrac{b-a}{2^k}\right] \end{cases}\qquad (k=1,2,3,\cdots)$$

### 2. 例子

用递推公式求积分 $\pi=\int_0^1\frac{4}{1+x^2}dx$ 的近似值，使误差不超过 $10^{-6}$。

$$T_1=\frac{1}{2}[f(0)+f(1)]=\frac{1}{2}(4+2)=3$$

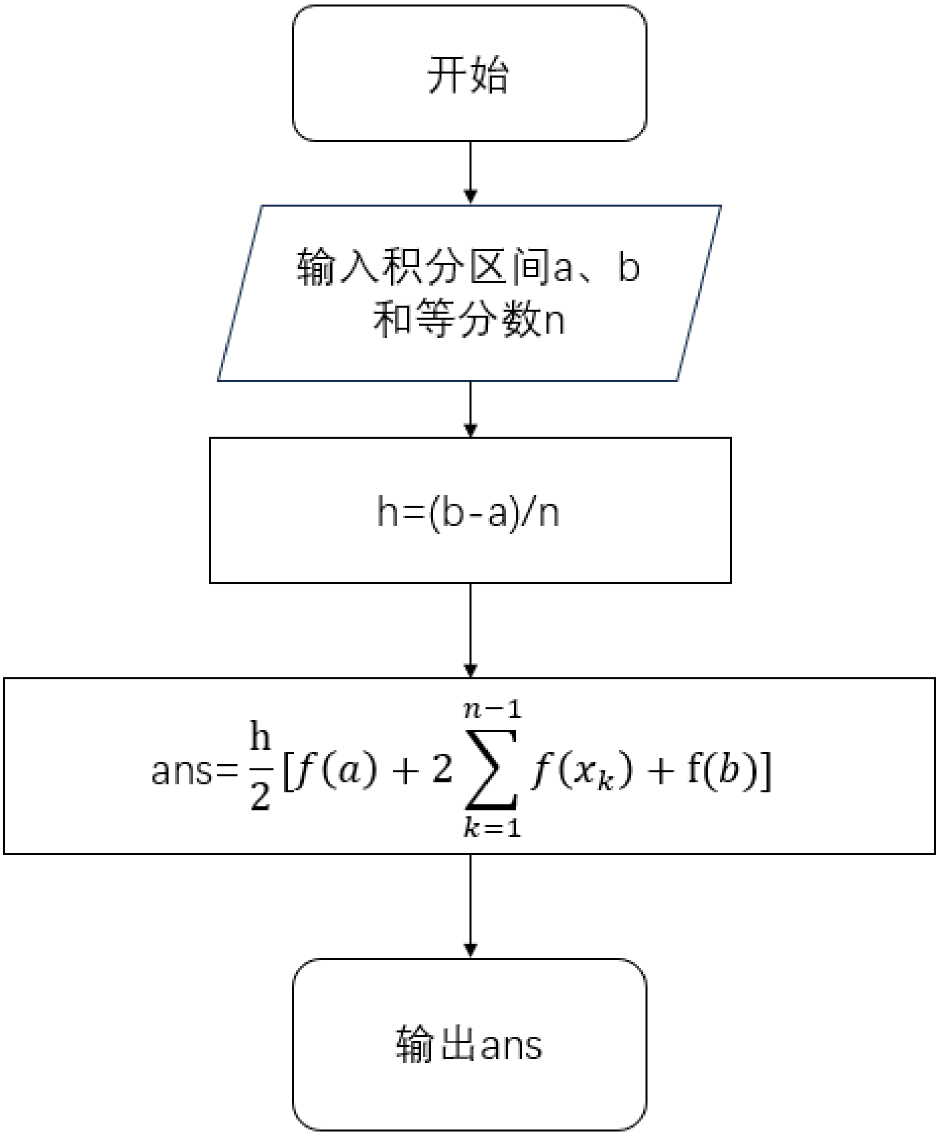
$$T_2=\frac{1}{2}T_1+\frac{1}{2}f(\frac{1}{2})=3.1$$

$$T_4=\frac{1}{2}T_2+\frac{1}{4}\left[f(\frac{1}{4})+f(\frac{3}{4})\right]=3.13117647$$

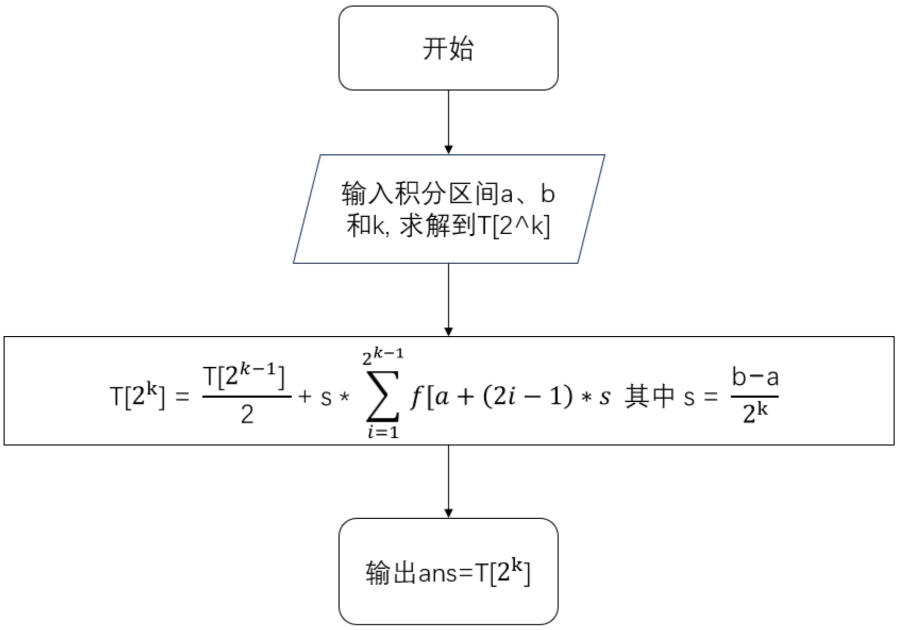
# 四、实验内容

## （一）算法流程图

### 1. 定步长复合梯形算法流程图



## 2. 变步长复合梯形算法流程图




## （二）编程作业

求 $\Pi = \int_0^1 \frac{4}{1+x^2}$ 的近似值。

(1)编写定步长复合梯形程序求解上式;
(2)编写变步长复合梯形程序求解上式，使误差不超过 10-6.
【提示】请根据前面的算法流程图进行编写程序。

## 1.定步长复合梯形程序求解

```
#include <bits/stdc++.h>

using namespace std;

double f(double x) {
    return 4 / (1 + x * x);
}

int main() {
    cout << "输入积分区间:";
```

```cpp
    double a, b;
    int n;
    cin >> a >> b;
    cout << "输入等分数:";
    cin >> n;
    double h = (b - a) / n;
    double ans = 0;
    ans += f(a) + f(b);
    for (int k = 1; k <= n - 1; k++) {
        ans += 2 * f(a + k * h);
    }
    ans *= h / 2;
    cout << "积分结果为:" << ans << endl;
    double pi = 3.14159265;
    cout << "与精确值(" << pi << ")的误差为:" << ans - pi << endl;
}
```

```
输入积分区间:0 1
输入等分数:100
积分结果为:3.14158
与精确值(3.14159)的误差为:-1.66631e-05

进程已结束，退出代码为 0
```

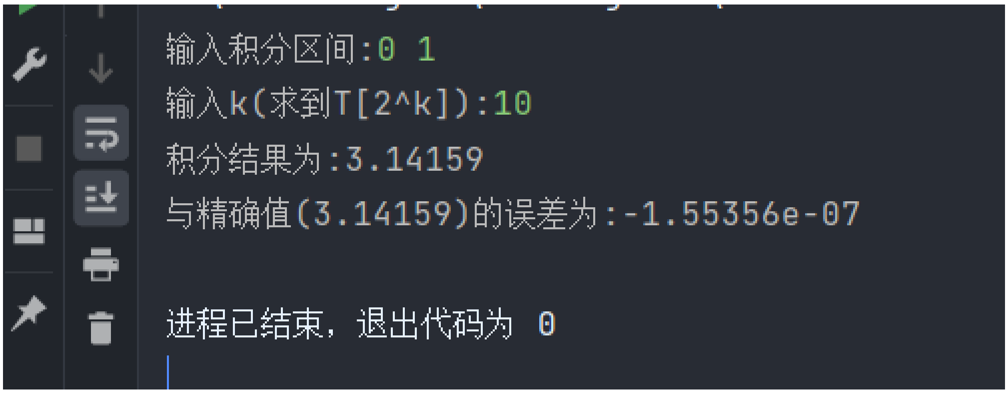
## 2.变步长复合梯形程序求解

```cpp
#include <bits/stdc++.h>

using namespace
        std;

double f(double x) {
    return 4 / (1 + x * x);
}

double getTk1(int k, double a, double b) {
    double T[(1 << k) + 1];
    T[1] = (f(a) + f(b)) / 2;
    for (int i = 2; i <= (1 << k); i <<= 1) {
        // T[2^k] = T[2^(k-1)]/2 + s * sum{ f[a+(2i-1)*s] | i=1->2^(k-1) } 其中 s =
```

```
(b-a)/2^k
        T[i] = T[i / 2] / 2;
        double s = (b - a) / i;
        double t = 0;// t = sum{ f[a+(2i-1)*s] | i=1->2^(k-1) }
        for (int j = 1; j <= i / 2; j++) {
            double x = a + (2 * j - 1) * s;
            t += f(x);
        }
        T[i] += s * t;
    }
    return T[1 << k];
}

double getTk2(int k, double a, double b) {
    // 压缩 T 数组, 2^k -> k
    double T[k + 1];// T[k] <--> T[2^k]
    T[0] = (f(a) + f(b)) / 2;
    for (int i = 1; i <= k; i++) {
        // T[2^k] = T[2^(k-1)]/2 + s * sum{ f[a+(2i-1)*s] | i=1->2^(k-1) } 其中 s =
(b-a)/2^k
        // T[i] = T[i-1]/2 + s * sum{ f[a+(2j-1)*s] | j=1->2^(i-1) } 其中 s = (b-
a)/2^i
        T[i] = T[i - 1] / 2;
        double s = (b - a) / (1 << i);
        double t = 0;// t = sum{ f[a+(2i-1)*s] | i=1->2^(k-1) }
        for (int j = 1; j <= (1 << (i - 1)); j++) {
            double x = a + (2 * j - 1) * s;
            t += f(x);
        }
        T[i] += s * t;
    }
    return T[k];
}

double getTk3(int k, double a, double b) {
    // T 的后一项只依靠前一项,所以仅使用一个变量存储即可
    double T = (f(a) + f(b)) / 2;
    for (int i = 1; i <= k; i++) {
        // T[2^k] = T[2^(k-1)]/2 + s * sum{ f[a+(2i-1)*s] | i=1->2^(k-1) } 其中 s =
(b-a)/2^k
        // T[i] = T[i-1]/2 + s * sum{ f[a+(2j-1)*s] | j=1->2^(i-1) } 其中 s = (b-
a)/2^i
        T /= 2;
        double s = (b - a) / (1 << i);
```

```
        double t = 0;// t = sum{ f[a+(2i-1)*s] | i=1->2^(k-1) }
        for (int j = 1; j <= (1 << (i - 1)); j++) {
            double x = a + (2 * j - 1) * s;
            t += f(x);
        }
        T += s * t;
    }
    return T;
}

int main() {
    cout << "输入积分区间:";
    double a, b;
    int k;
    cin >> a >> b;
    cout << "输入 k(求到 T[2^k]):";
    cin >> k;

    double ans = getTk3(k, a, b);
    cout << "积分结果为:" << ans << endl;
    double pi = 3.14159265;
    cout << "与精确值(" << pi << ")的误差为:" << ans - pi << endl;
}
```

运行结果:




## (三) 使用使用其他编程语言(Python、Java)

**python:**

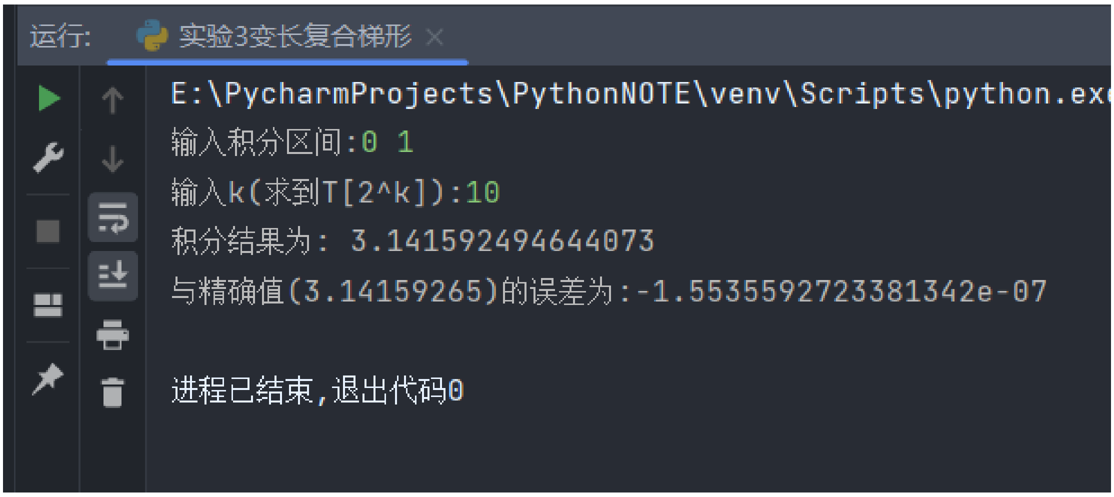
```python
def f(x):
    return 4 / (1 + x * x)
```

```python
a, b = map(float, input("输入积分区间:").split())
n = int(input("输入等分数:"))
h = (b - a) / n
ans = 0
ans += f(a) + f(b)
for k in range(1, n):
    ans += 2 * f(a + k * h)
ans *= h / 2
print("积分结果为:", ans)
pi = 3.14159265
print("与精确值(", pi, ")的误差为:", ans - pi)
```

```
运行:  实验3定长复合梯形
E:\PycharmProjects\PythonNOTE\venv\Scripts\python.exe E
输入积分区间:0 1
输入等分数:100
积分结果为: 3.141575986923129
与精确值( 3.14159265 )的误差为: -1.6663076871203940e-05

进程已结束,退出代码0
```

```python
def f(x):
    return 4 / (1 + x * x)


def getTk(k, a, b):
    T = (f(a) + f(b)) / 2
    for i in range(1, k + 1):
        T /= 2
        s = (b - a) / (1 << i)
        t = 0
        for j in range(1, (1 << (i - 1)) + 1):
            x = a + (2 * j - 1) * s
            t += f(x)
        T += s * t
    return T


a, b = map(float, input("输入积分区间:").split())
k = int(input("输入 k(求到 T[2^k]):"))

ans = getTk(k, a, b)
```

```python
print("积分结果为:", ans)
pi = 3.14159265
print("与精确值({0})的误差为:{1}".format(pi, ans - pi))
```



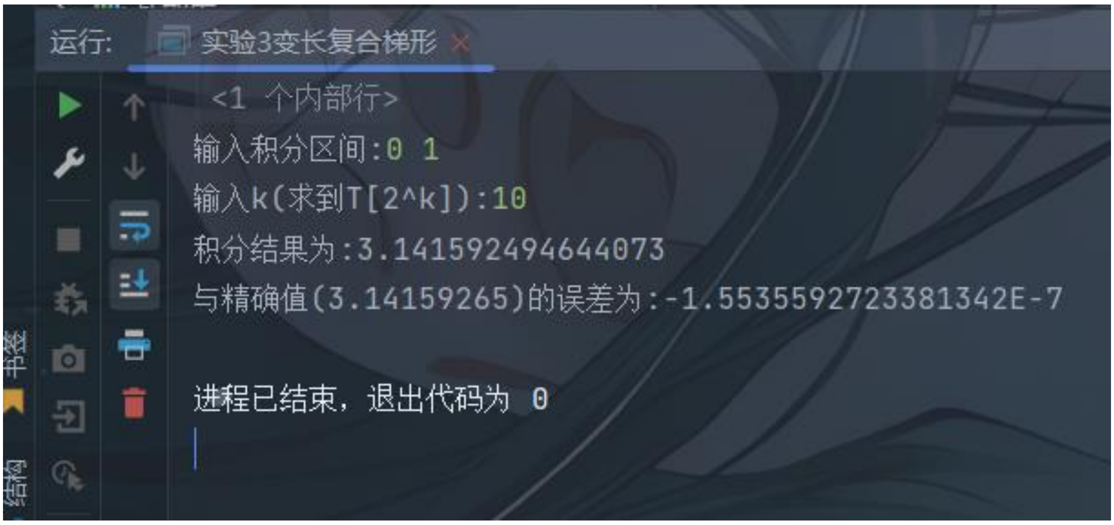

## java:

```java
package 数值计算实验;

import java.util.Scanner;

public class 实验 3 定长复合梯形 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("输入积分区间:");
        double a = sc.nextDouble();
        double b = sc.nextDouble();
        System.out.print("输入等分数:");
        int n = sc.nextInt();
        double h = (b - a) / n;
        double ans = 0;
        ans += f(a) + f(b);
        for (int k = 1; k <= n - 1; k++) {
            ans += 2 * f(a + k * h);
        }
        ans *= h / 2;
        System.out.println("积分结果为:" + ans);
        double pi = 3.14159265;
        System.out.println("与精确值(" + pi + ")的误差为:" + (ans -
pi));
    }
```

```java
    public static double f(double x) {
        return 4 / (1 + x * x);
    }
}
```


```java
package 数值计算实验;

import java.util.Scanner;

public class 实验 3 变长复合梯形 {
    public static double f(double x) {
        return 4 / (1 + x * x);
    }

    public static double getTk(int k, double a, double b) {
        double T = (f(a) + f(b)) / 2;
        for (int i = 1; i <= k; i++) {
            T /= 2;
            double s = (b - a) / (1 << i);
            double t = 0;
            for (int j = 1; j <= (1 << (i - 1)); j++) {
                double x = a + (2 * j - 1) * s;
                t += f(x);
            }
            T += s * t;
        }
        return T;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("输入积分区间:");
```

```java
        double a = sc.nextDouble(), b = sc.nextDouble();
        System.out.print("输入 k(求到 T[2^k]):");
        int k = sc.nextInt();

        double ans = getTk(k, a, b);
        System.out.println("积分结果为:" + ans);
        double pi = 3.14159265;
        System.out.println("与精确值(" + pi + ")的误差为:" + (ans -
pi));
    }
}
```



# 实验四 高斯消元法(2 课时)

# 一、实验目的

1.掌握高斯选主元消去法公式的用法，适用范围及精确度。
2.通过高斯选主元消去法求矩阵方程的解，验证高斯消去法。

# 二、实验要求

1.写出高斯选主元消去法解线性方程组算法，编写程序上机调试出结果
2.进一步加深对高斯消去法的理解。

# 三、实验原理

## (一)高斯消元法

Gauss 消去法就是将方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

通过(n-1)步消元，将上述方程组转化为上三角方程组

$$\begin{bmatrix} a_{11}^{(1)} & a_{(12)}^{(11)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

再回代求此方程组的解。

## (二)选列主元素高斯消去法

给定线性方程组 Ax=b，记 A(1)=A，b(1)=b，列主元 Gauss 消去法的具体过程如下:
首先在增广矩阵 B(1)=(A(1),b(1))的第一列元素中，取

$$\left|a_{k1}^{(1)}\right| = \max_{1\le i\le n}\left|a_{i1}^{(1)}\right| \text{ 为主元素，} r_k \leftrightarrow r_1.$$

然后进行第一步消元得增广矩阵 B(2)=(A(2),b(2))。 再在矩阵 B(2)=(A(2),b(2))的第二列元素中，取

$$\left|a_{k2}^{(2)}\right| = \max_{2\le i\le n}\left|a_{i2}^{(2)}\right| \text{ 为主元素，} r_k \leftrightarrow r_2.$$

然后进行第二步消元得增广矩阵 B(3)=(A(3),b(3))。按此方法继续进行下去，经过 n-1 步选主元和消元运算,得到增广矩阵 B(n)=(A(n),b(n)).则方程组 A(n)x=b(n)是与原方程组
等价的上三角形方程组,可进行回代求解.
易证，只要 A|≠0,列主元 Gauss 消去法就可顺利进行

## (三)例子

采用 4 位十进制浮点计算，分别用顺序 Gauss 消去法和列主元 Gauss 消去法求解线性方程组:

$$\begin{cases} 0.012x_1 + 0.01x_2 + 0.167x_3 = 0.6781 \\ x_1 + 0.8334x_2 + 5.91x_3 = 12.1 \\ 3200x_1 + 1200x_2 + 4.2x_3 = 981 \end{cases}$$

方程组具有四位有效数字的精确解为 x1*=17.46，x2*=-45.76，x3*=5.546
解(1)用顺序 Gauss 消去法求解，消元过程为

$$\begin{pmatrix} 0.0120 & 0.0100 & 0.167 & 0.6781 \\ 1.000 & 0.8334 & 5.910 & 12.10 \\ 3200 & 1200 & 4.200 & 981.0 \end{pmatrix}$$

$$\sim\begin{pmatrix} 0.0120 & 0.0100 & 0.1670 & 0.6781 \\ 0 & 0.1000\times10^{-3} & -8.010 & -44.41 \\ 0 & -1467 & -4453\times10 & -1798\times10^{2} \end{pmatrix}$$

$$\sim\begin{pmatrix} 0.0120 & 0.0100 & 0.1670 & 0.6781 \\ 0 & 0.1000\times10^{-3} & -8.010 & -44.41 \\ 0 & 0 & -1175\times10^{5} & -6517\times10^{5} \end{pmatrix}$$

回代得:x3=5.546，x2=100.0，x1=-104.0

(2)用列主元 Gauss 消去法求解，消元过程为

$$\begin{pmatrix} 0.0120 & 0.0100 & 0.1670 & 0.6781 \\ 1.000 & 0.8334 & 5.910 & 12.10 \\ 3200 & 1200 & 4.200 & 981.0 \end{pmatrix}$$

$$\underset{r_1\leftrightarrow r_3}{\overset{\text{选主元}}{\sim}}\begin{pmatrix} 3200 & 1200 & 4.200 & 981.0 \\ 1.000 & 0.8334 & 5.910 & 12.10 \\ 0.0120 & 0.0100 & 0.1670 & 0.6781 \end{pmatrix}$$

$$\sim\begin{pmatrix} 3200 & 1200 & 4.200 & 981.0 \\ 0 & 0.4584 & 5.909 & 11.79 \\ 0 & 0.55\times10^{-2} & 0.1670 & 0.6744 \end{pmatrix}$$

$$\sim\begin{pmatrix} 3200 & 1200 & 4.200 & 981.0 \\ 0 & 0.4584 & 5.909 & 11.79 \\ 0 & 0 & 0.0961 & 0.5329 \end{pmatrix}$$

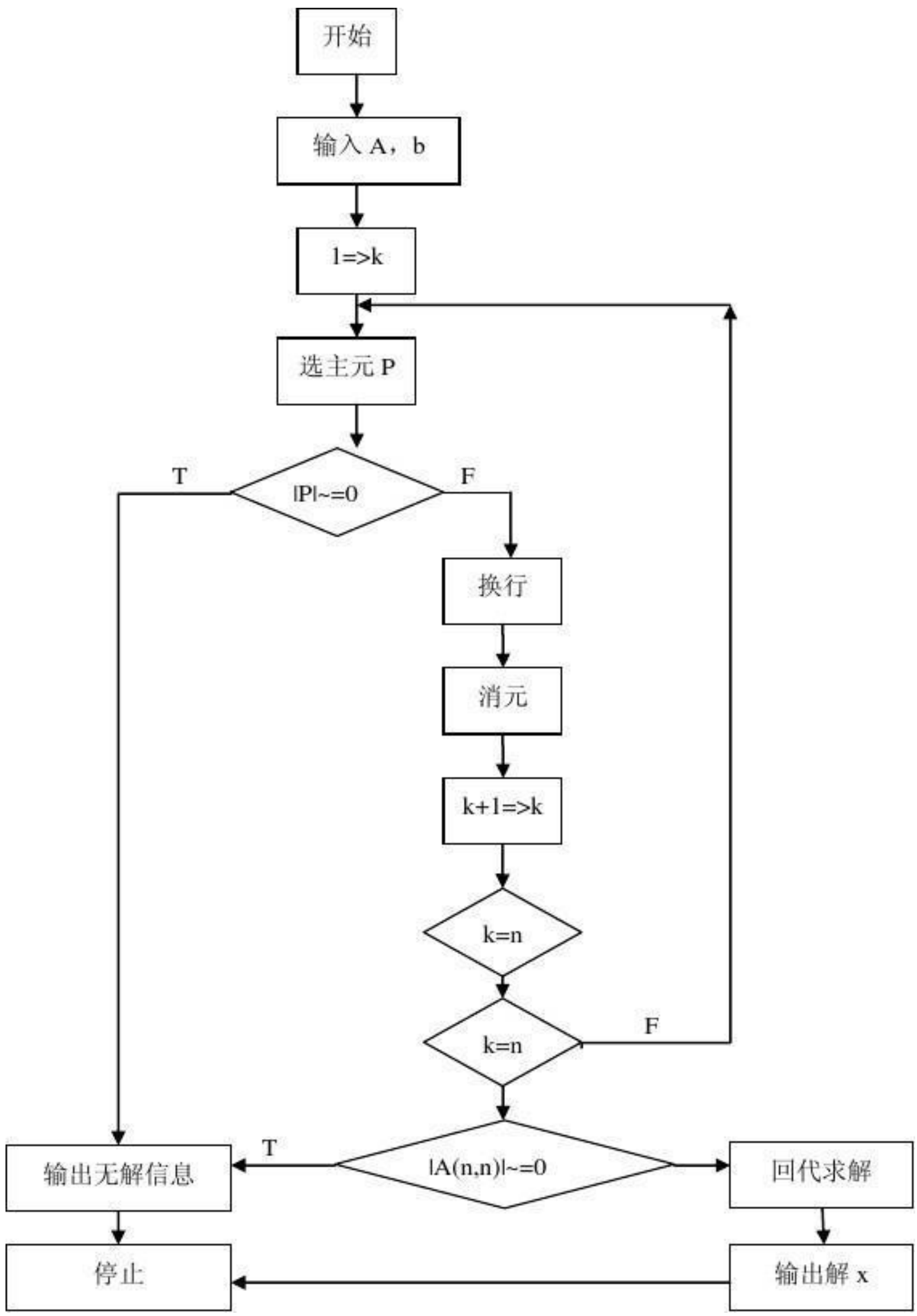
$$\sim\begin{pmatrix} 3200 & 1200 & 4.200 & 981.0 \\ 0 & 0.4584 & 5.909 & 11.79 \\ 0 & 0 & 0.0961 & 0.5329 \end{pmatrix}$$

回代得:x3=5.545，x2=-45.77，x1=17.46

# 四、实验内容

## (一)算法流程图

高斯列主元消去法 N-S 图



## (二)编程作业

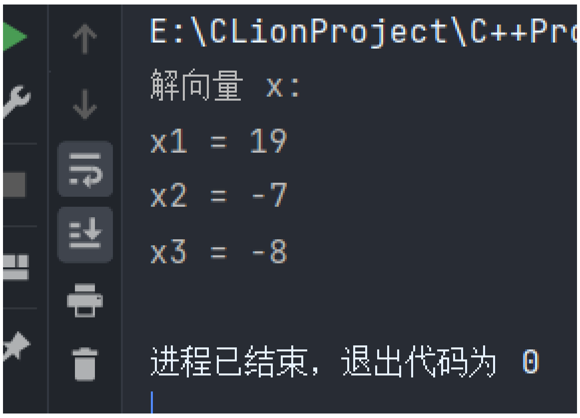
编写选列主元的高斯消去法。求出下列线性方程组 Ax=b 的解 x。

$$\begin{bmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix}$$

```
#include <iostream>
#include <cmath>

using namespace std;


int main() {
    int n = 3;
    double A[3][3] = {{3, 1, 6},
                      {2, 1, 3},
                      {1, 1, 1}};    // 系数矩阵 A
    double b[] = {2, 7, 4};  // 常数项向量 b
    // 高斯列主元消去法函数求解线性方程组
    for (int k = 0; k < n - 1; k++) {
        // 寻找主元所在的行
        int maxRow = k;
        double maxVal = abs(A[k][k]);
        for (int i = k + 1; i < n; i++) {
            if (abs(A[i][k]) > maxVal) {
                maxRow = i;
                maxVal = abs(A[i][k]);
            }
        }
        // 交换最大行和当前行
         swap(A[k], A[maxRow]);
       swap(b[k], b[maxRow]);
        // 消元
        for (int i = k + 1; i < n; i++) {
            double factor = A[i][k] / A[k][k];
            for (int j = k; j < n; j++) {
                A[i][j] -= factor * A[k][j];
            }
            b[i] -= factor * b[k];
```

```
        }
    }
    // 回代
    double x[n];
    for (int i = n - 1; i >= 0; i--) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
        x[i] /= A[i][i];
    }
    // 输出解向量
    cout << "解向量 x: " << endl;
    for (int i = 0; i < n; i++) {
        cout << "x" << i + 1 << " = " << x[i] << endl;
    }
    return 0;
}
```

运行结果:




## (三) 使用使用其他编程语言(Python、Java)

### python:

```
n = 3
A = [[3, 1, 6],
     [2, 1, 3],
     [1, 1, 1]]  # 系数矩阵 A
b = [2, 7, 4]  # 常数项向量 R

# 高斯列主元消去法函数求解线性方程组
for k in range(n - 1):
```

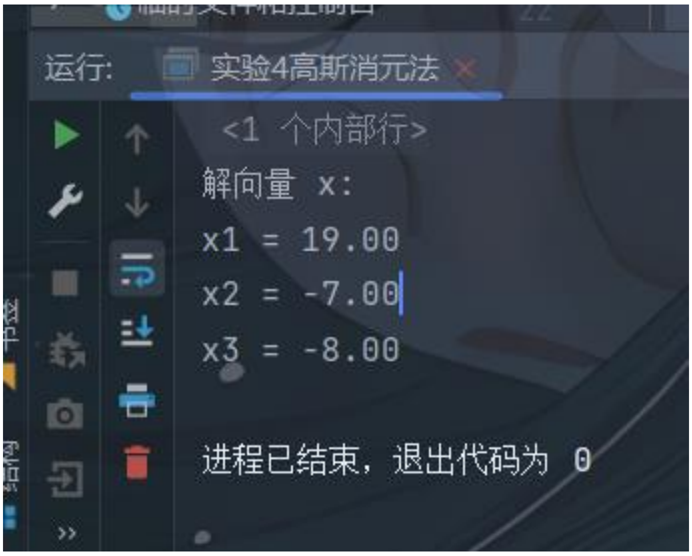
```python
    # 寻找主元所在的行
    maxRow = k
    maxVal = abs(A[k][k])
    for i in range(k + 1, n):
        if abs(A[i][k]) > maxVal:
            maxRow = i
            maxVal = abs(A[i][k])
    # 交换最大行和当前行
    A[k], A[maxRow] = A[maxRow], A[k]
    b[k], b[maxRow] = b[maxRow], b[k]
    # 消元
    for i in range(k + 1, n):
        factor = A[i][k] / A[k][k]
        for j in range(k, n):
            A[i][j] -= factor * A[k][j]
        b[i] -= factor * b[k]

# 回代
x = [0] * n
for i in range(n - 1, -1, -1):
    x[i] = b[i]
    for j in range(i + 1, n):
        x[i] -= A[i][j] * x[j]
    x[i] /= A[i][i]

# 输出解向量
print("解向量 x:")
for i in range(n):
    print(f"x{i + 1} = {x[i]}")
```

```
运行:  实验4高斯消元法
E:\PycharmProjects\PythonNOTE\ve
解向量 x:
x1 = 19.0
x2 = -6.999999999999998
x3 = -8.0

进程已结束,退出代码0
```

## java:

```
package 数值计算实验;

public class 实验4高斯消元法 {
    public static void main(String[] args) {
        int n = 3;
        double[][] A = {{3, 1, 6},
                {2, 1, 3},
                {1, 1, 1}};    // 系数矩阵 A
        double[] b = {2, 7, 4};   // 常数项向量 R

        // 高斯列主元消去法函数求解线性方程组
       for (int k = 0; k < n - 1; k++) {
            // 寻找主元所在的行
          int maxRow = k;
            double maxVal = Math.abs(A[k][k]);
            for (int i = k + 1; i < n; i++) {
                if (Math.abs(A[i][k]) > maxVal) {
                    maxRow = i;
                    maxVal = Math.abs(A[i][k]);
                }
            }
            // 交换最大行和当前行
          swapRows(A, k, maxRow);
            swapElements(b, k, maxRow);
            // 消元
          for (int i = k + 1; i < n; i++) {
                double factor = A[i][k] / A[k][k];
                for (int j = k; j < n; j++) {
                    A[i][j] -= factor * A[k][j];
                }
                b[i] -= factor * b[k];
            }
        }

        // 回代
       double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            x[i] = b[i];
            for (int j = i + 1; j < n; j++) {
                x[i] -= A[i][j] * x[j];
            }
```

```
            x[i] /= A[i][i];
        }

        // 输出解向量
        System.out.println("解向量 x:");
        for (int i = 0; i < n; i++) {
            System.out.printf("x%d = %.2f%n", i + 1, x[i]);
        }
    }

    private static void swapRows(double[][] A, int row1, int row2) {
        double[] temp = A[row1];
        A[row1] = A[row2];
        A[row2] = temp;
    }

    private static void swapElements(double[] b, int index1, int
index2) {
        double temp = b[index1];
        b[index1] = b[index2];
        b[index2] = temp;
    }
}
```



# 实验五 非线性方程求解(2 课时)

## 一、实验目的

1. 了解求解非线性方程的解的常见方法
2. 编写牛顿迭代法程序求解非线性方程

## 二、实验要求

1. 设计牛顿迭代法算法，编写程序上机调试。
2. 进一步加深对迭代法的理解。

## 三、实验原理

### (一)牛顿迭代法

又称为牛顿-雷夫生方法（Newton-Raphson method），是一种在实数域和复数域上通过迭代计算求出非线性方程的数值解方法。方法的基本思路是利用一个根的猜测值 $x_0$ 做初始近似值，使用函数 $f(x)$在 $x_0$ 处的泰勒级数展式的前两项做为函数 $f(x)$的近似表达式。由于该表达式是一个线性函数，通过线性表达式替代方程 $f(x)=0$ 中的 $f(x)$求得近似解 $x_1$。即将方程 $f(x)=0$ 在 $x_0$ 处局部线性化计算出近似解 $x_1$，重复这一过程，将方程 $f(x)=0$ 在 $x_1$ 处局部线性化计算出 $x_2$，求得近似解 $x_2$，……。详细叙述如下：假设方程的解 $x^*$在 $x_0$ 附近（$x_0$ 是方程解 $x^*$的近似），函数 $f(x)$在点 $x_0$ 处的局部线化表达式为

$$f(x) \approx f(x_0) + (x - x_0) f'(x_0)$$

由此得一次方程

$$f(x_0) + (x - x_0) f'(x_0) = 0$$

求解，得

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

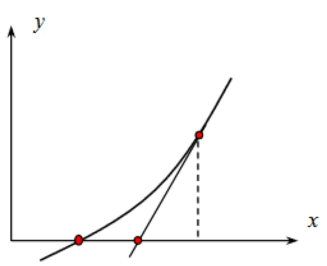


图 5-1 牛顿迭代法示意

如图 5-1 所示，$x_1$ 比 $x_0$ 更接近于 $x^*$。该方法的几何意义是：用曲线上某点（$x_0$，$y_0$）的切线代替曲线，以该切线与 x 轴的交点（$x_1$，0）作为曲线与 x 轴的交点（$x^*$，0）的近似（所以牛顿迭代法又称为切线法）。设 $x_n$ 是方程解 x*的近似，迭代格式

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (n = 0，1，2，\cdots\cdots)$$

就是著名的牛顿迭代公式，通过迭代计算实现逐次逼近方程的解。牛顿迭代法的最大优点是收敛速度快，具有二阶收敛。以著名的平方根算法为例，说明二阶收敛速度的意义。

## （二）例子

已知 $\sqrt{2} \approx 1.4$，求 $\sqrt{2}$ 等价于求方程 $f(x) = x^2 - 2 = 0$ 的解。由于 $f'(x) = 2x$。应用牛顿迭代法，得迭代计算格式

$$x_{n+1} = \frac{1}{2}(x_n + 2/x_n)，（n = 0，1，2，\cdots\cdots）$$

取 $x_0$= 1.4 为初值，迭代计算 3 次的数据列表如表 5-1：

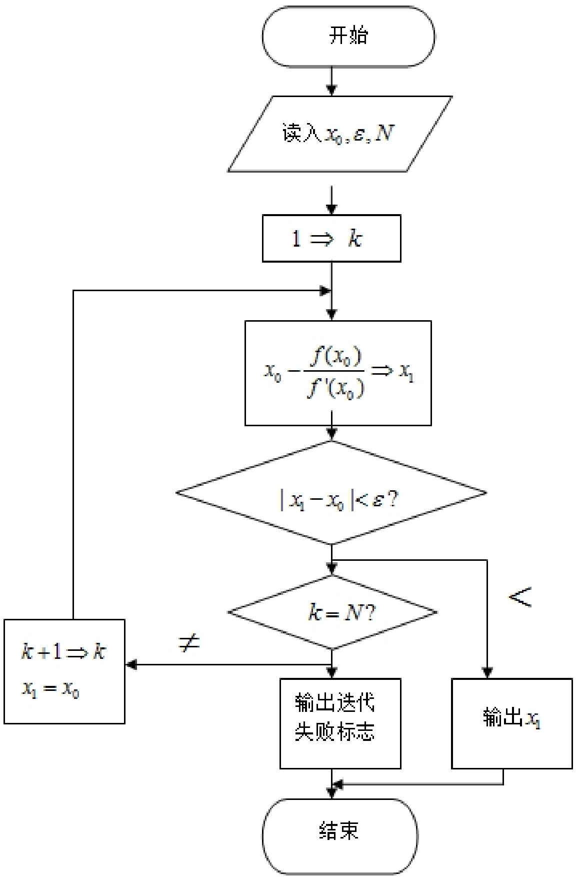
表 5-1 牛顿迭代法数值实验

| 迭代次数 | 近似值 | 15 位有效数 | 误差 |
|---|---|---|---|
| 0 | 1.4 | 1.310 | -1.42e-002 |
| 1 | 1.571 | 1.310 | 7.21e-005 |
| 2 | 1.356 | 1.310 | 1.84e-009 |
| 3 | 1.309 | 1.310 | -2.22e-016 |

# 四、实验内容

## (一)算法流程图

牛顿迭代法算法流程图

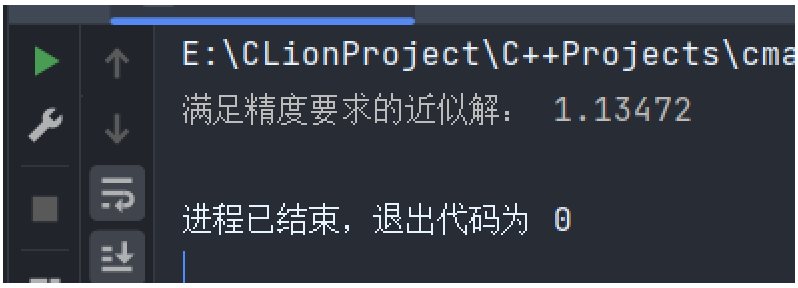


## (二)编程作业

编写 Newton 迭代法通用子程序。实现方程 f(x)=x^6-x-1-0 的满足精度要求的解
要求求解过程中用一个变量 I 控制三种状态，其中:
i=0 表示求解满足给定精度的近似解:
i=1 表示 f(x0)=0，计算中断;
i=2 表示迭代 n 次后精度要求仍不满足

```
#include <iostream>
#include <cmath>

double f(double x) {
    return pow(x, 6) - x - 1;
}

double df(double x) {
    return 6 * pow(x, 5) - 1;
}

double newton_iteration(double x0, double acc, int max_iter, int &i) {
    double x = x0;
    for (int iter = 0; iter < max_iter; ++iter) {
        double fx = f(x);
        if (fabs(fx) < acc) { // 找到满足精度的近似解
            i = 0;
            return x;
        }
        double dfx = df(x);
        if (dfx == 0) {// f`(x0)=0, 计算中断
            i = 1;
            return x;
        }
        x = x - fx / dfx; // 迭代
    }
    // 迭代 n 次后精度仍未满足
    i = 2;
    return x;
}

int main() {
    double x0 = 1.0;// 初始近似解
    double acc = 1e-6;// 精度
    int max_iter = 1000;// 最大迭代次数
```

```cpp
    int i;
    double root = newton_iteration(x0, acc, max_iter, i);
    if (i == 0) {
        std::cout << "满足精度要求的近似解： " << root << std::endl;
    } else if (i == 1) {
        std::cout << "f(x0)=0，计算中断" << std::endl;
    } else {
        std::cout << "迭代 n 次后精度要求仍不满足" << std::endl;
    }
    return 0;
}
```




## (三)选做题

1. 用简化牛顿法计算上述例题。
2. 用牛顿下山法计算上述例题。

```cpp
#include <iostream>
#include <cmath>

double f(double x) {
    return pow(x, 6) - x - 1;
}

double df(double x) {
    return 6 * pow(x, 5) - 1;
}

double newton_iteration(double x0, double acc, int max_iter, int &i) {
    double x = x0;
    for (int iter = 0; iter < max_iter; ++iter) {
        double fx = f(x);
        if (fabs(fx) < acc) { // 找到满足精度的近似解
            i = 0;
            return x;
        }
        double dfx = df(x);
```

```cpp
        if (dfx == 0) {// f`(x0)=0, 计算中断
            i = 1;
            return x;
        }
        // 牛顿下山法
        double step = fx / dfx;
        while (f(x - step) > f(x) && step > 1e-8) {
            step /= 2;
        }
        x = x - step;
    }
    // 迭代 n 次后精度仍未满足
    i = 2;
    return x;
}

int main() {
    double x0 = 1.0;// 初始近似解
    double acc = 1e-6;// 精度
    int max_iter = 1000;// 最大迭代次数
    int i;
    double root = newton_iteration(x0, acc, max_iter, i);
    if (i == 0) {
        std::cout << "满足精度要求的近似解： " << root << std::endl;
    } else if (i == 1) {
        std::cout << "f(x0)=0，计算中断" << std::endl;
    } else {
        std::cout << "迭代 n 次后精度要求仍不满足" << std::endl;
    }
    return 0;
}
```

```
E:\CLionProject\C++Projects\cmak
满足精度要求的近似解： 1.13472

进程已结束，退出代码为 0
```

(四) 使用使用其他编程语言(Python、Java)

**python:**

```python
def f(x):
    return x ** 6 - x - 1


def df(x):
    return 6 * x ** 5 - 1


def newton_iteration(x0, acc=1e-6, max_iter=1000):
    x = x0
    for it in range(max_iter):
        fx = f(x)
        if abs(fx) < acc:  # 找到满足精度的近似解
            return x
        dfx = df(x)
        if dfx == 0:  # f'(x0)=0, 计算中断
            return x
        x = x - fx / dfx  # 迭代
    # 迭代n次后精度仍未满足
    return x


x0 = 1.0  # 初始近似解
root = newton_iteration(x0)
print("满足精度要求的近似解：", root)
```

运行: 实验5牛顿迭代代法

```
E:\PycharmProjects\PythonNOTE\venv\Scripts\pyth
满足精度要求的近似解：  1.1347242213865578

进程已结束,退出代码0
```

## java:

```java
package 数值计算实验;

public class 实验5牛顿迭代法 {
    public static double f(double x) {
        return Math.pow(x, 6) - x - 1;
    }

    public static double df(double x) {
        return 6 * Math.pow(x, 5) - 1;
    }

    public static double newtonIteration(double x0, double acc, int
maxIter) {
        double x = x0;
        for (int it = 0; it < maxIter; it++) {
            double fx = f(x);
            if (Math.abs(fx) < acc) { // 找到满足精度的近似解
              return x;
            }
            double dfx = df(x);
            if (dfx == 0) { // f'(x0)=0, 计算中断
              return x;
            }
            x = x - fx / dfx; // 迭代
        }
        // 迭代n次后精度仍未满足
      return x;
    }

    public static void main(String[] args) {
        double x0 = 1.0; // 初始近似解
      double acc = 1e-6; // 精度
      int maxIter = 1000; // 最大迭代次数
      double root = newtonIteration(x0, acc, maxIter);
        System.out.println("满足精度要求的近似解: " + root);
    }
}
```


运行:
实验5牛顿迭代法
<1 个内部行>
满足精度要求的近似解：1.1347242213865578
进程已结束，退出代码为 0

# 实验六 常微分方程初值问题数值解法(4 课时)

## 一、实验目的

1.掌握 Euler 法和改进的 Euler 法公式的用法。
2.进一步加深对微分方程数值解的理解。

## 二、实验要求

1.编写欧拉法程序。
2.编写改进的欧拉法程序，学会用改进的欧拉公式来求解常微分方程初值问题。

## 三、实验原理

### （一）常微分方程初值问题

$$\begin{cases} y' = f(x,y), a \le x \le b \\ y(a) = y_0 \end{cases} \qquad \text{(式 1)}$$

的数值解法，这也是科学与工程计算经常遇到的问题。

### （二）欧拉法

求初值问题(式 1)的一种最简单方法是将节点$x_n$的导数$y'(x_n)$用差商$\dfrac{y(x_n+h)-y(x_n)}{h}$代替，于是(式 1)的方程可近似写成

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y)(x_n)), n = 0,1,\cdots \qquad \text{(式 2)}$$

从$x_0$出发$y(a) = y(x_0) = y_0$，由(式 2)求得$y(x_1) \approx y_0 + hf(x_0, y_0) = y_1$再将$y_1 \approx y(x_1)$代入(式 2)右端，得到$y(x_2)$的近似$y_2 = y_1 + hf(x_1, y_1)$，一般写成

$$y_{n+1} = y_n + hf(x_n, y_n), n = 0,1,\cdots$$

称为解初值问题的 Euler 法。

## （三）改进欧拉法

先用 Euler 法计算出$y_{n+1}$的近似$\bar{y}_{n+1}$，将隐式梯形公式改为

$$\begin{cases} \bar{y}_{n+1} = y_n + hf(x_n, y_n) \\ y_{n+1} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})] \end{cases}$$

称为改进 Euler 法，它实际上是显式方法。即

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_n + hf(x_n, y_n))]$$

## （四）例子

求初值问题 $\begin{cases} y' = y - \dfrac{2x}{y}, & 0 \le x \le 1 \\ y(0) = 1 \end{cases}$ 的数值解，取步长 $h = 0.1$。（精确解为 $y(x) = (1 + 2x)^{1/2}$）

解：（1）利用欧拉法$\begin{cases} y_{i+1} = 1.1y_i - 0.2x_i / y_i \\ y_0 = 1 \quad, i = 0,1,2\cdots,9 \end{cases}$

（2）利用改进欧拉法$\begin{cases} y_{i+1} = y_i + 0.05(K_1 + K_2) \\ K_1 = y_i - 2x_i / y_i \\ K_2 = y_i + 0.1K_1 - \dfrac{2(x_i + 0.1)}{y_i + 0.1K_1} \\ y_0 = 1 \quad, i = 0,1,2\cdots,9 \end{cases}$

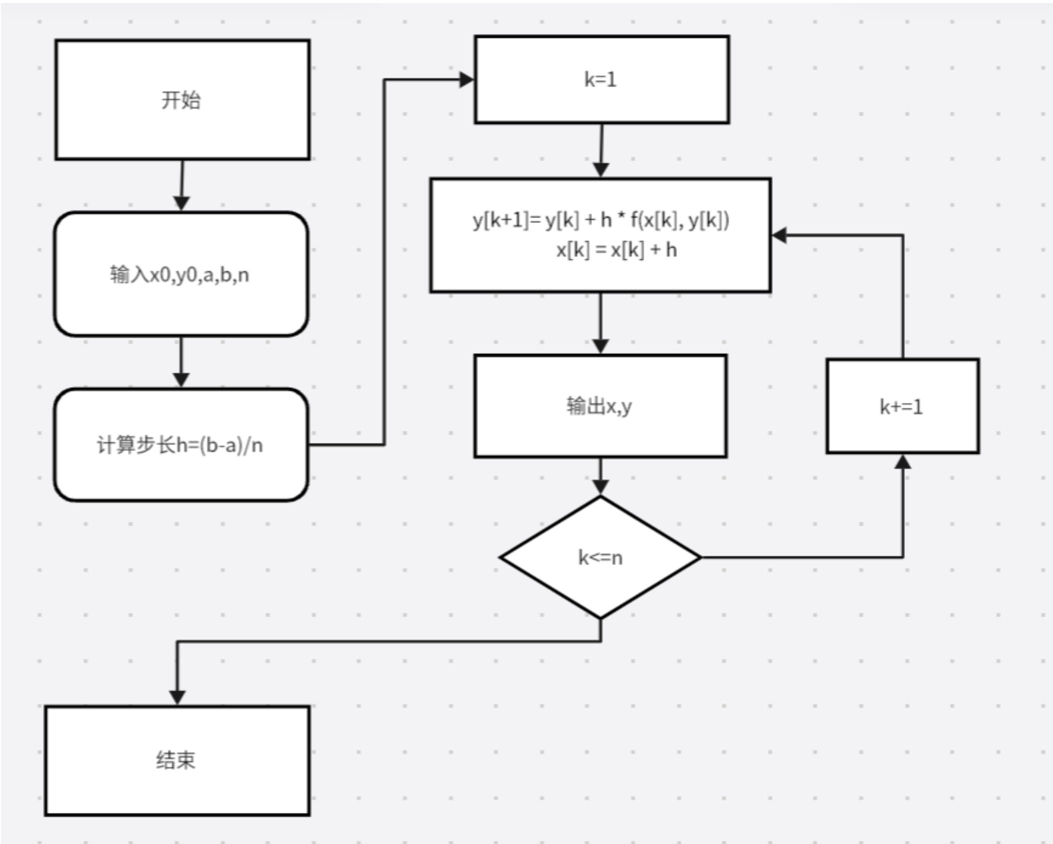
计算结果如下：

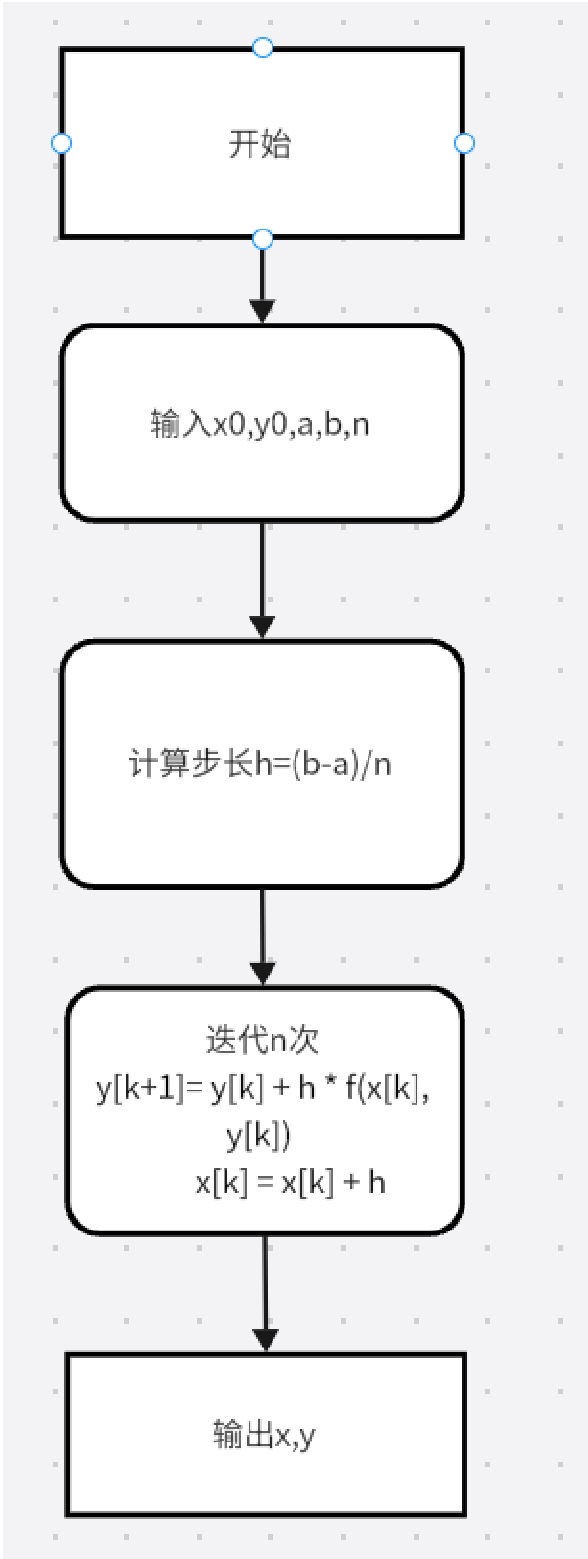
| *i* | *xi* | Euler 方法 *yi* | 改进 Euler 法 *yi* | 精确解 *y*(*xi*) |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0.1 | 1.1 | 1. | 1. |
| 2 | 0.2 | 1. | 1. | 1. |
| 3 | 0.3 | 1. | 1. | 1. |
| 4 | 0.4 | 1. | 1. | 1. |
| 5 | 0.5 | 1. | 1. | 1. |
| 6 | 0.6 | 1. | 1. | 1. |
| 7 | 0.7 | 1. | 1. | 1. |
| 8 | 0.8 | 1. | 1. | 1. |
| 9 | 0.9 | 1. | 1. | 1. |
| 10 | 1 | 1. | 1. | 1. |

# 四、实验内容

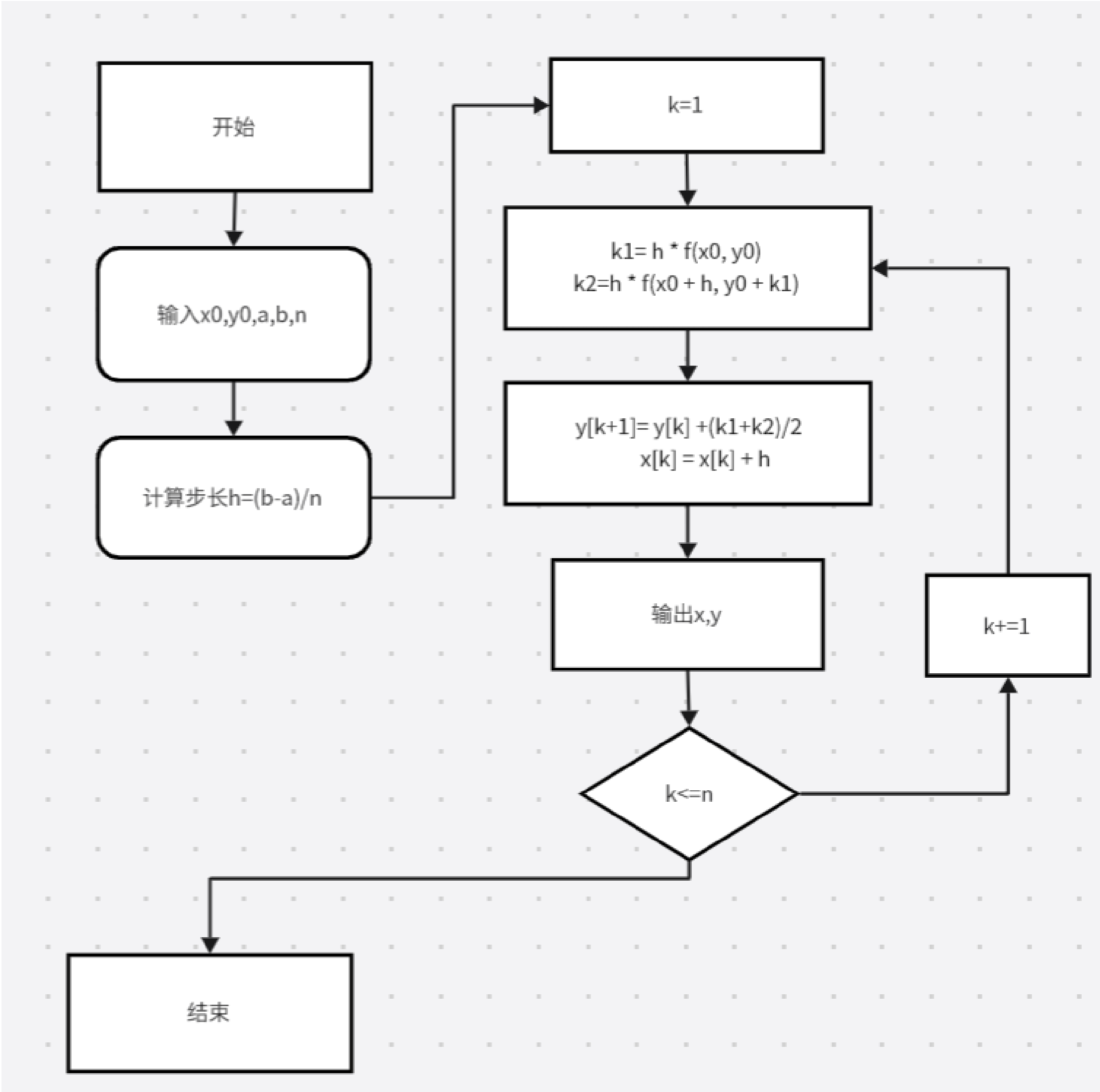
## （一）算法流程图

### 1. 请根据欧拉公式，画出其算法流程图




开始
输入x0,y0,a,b,n
计算步长h=(b-a)/n
迭代n次
y[k+1]= y[k] + h * f(x[k], y[k])
x[k] = x[k] + h
输出x,y

2. 请根据改进欧拉公式，画出其算法流程图。

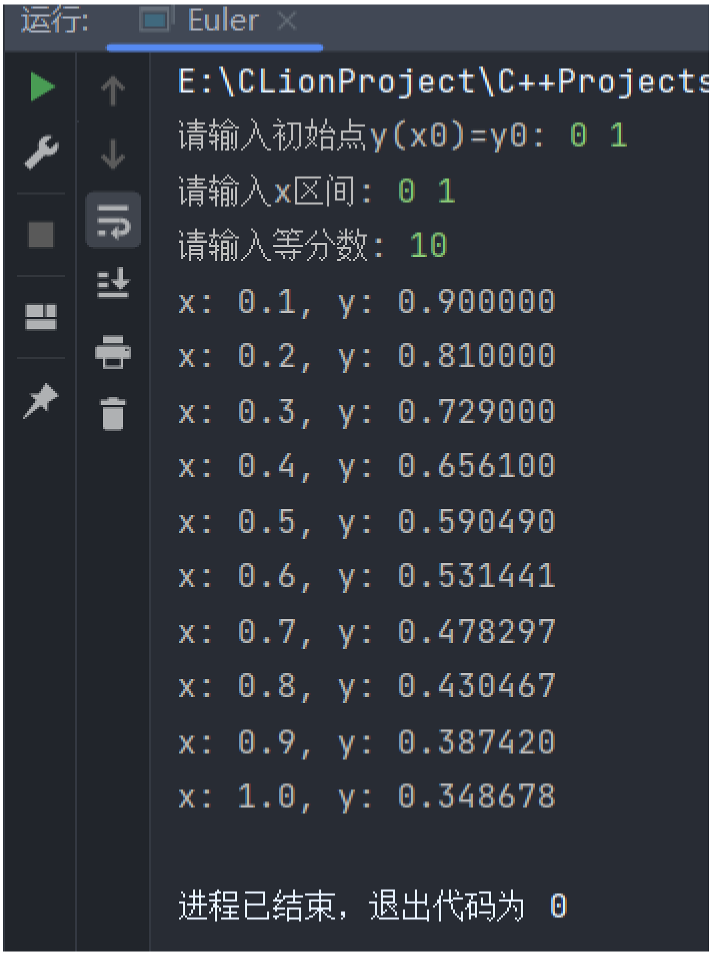



## （二）编程作业

编写 Euler 法和改进的 Euler 法程序。求微分方程 $\begin{cases} \dfrac{dy}{dx} = -y, & x \in [0,1] \\ y(0)=1 \end{cases}$ 在区间 10 等份的近似解。

## 1. Euler 法程序

```
#include <bits/stdc++.h>

using namespace std;

double f(double x, double y) {
    return -y;
}

int main() {
    double h, x0, y0, a, b, n;
    cout << "请输入初始点 y(x0)=y0: ";
    cin >> x0 >> y0;
    cout << "请输入 x 区间: ";
    cin >> a >> b;
    cout << "请输入等分数: ";
    cin >> n;
    h = (b - a) / n;
    for (int i = 0; i < n; i++) {
        y0 = y0 + h * f(x0, y0); // 迭代计算下一个点的 y 值
        x0 = x0 + h; // 更新 x 值
        printf("x: %.1f, y: %.6f\n", x0, y0); // 输出结果
    }
    return 0;
}
```




## 2. 改进的 Euler 法程序

```
#include <bits/stdc++.h>

using namespace std;

double f(double x, double y) {
    return -y;
}

int main() {
    double h, x0, y0, a, b, n;
    cout << "请输入初始点 y(x0)=y0: ";
    cin >> x0 >> y0;
    cout << "请输入 x 区间: ";
    cin >> a >> b;
    cout << "请输入等分数: ";
    cin >> n;
    h = (b - a) / n;
```

```
    for (int i = 0; i < n; i++) {
        double k1 = h * f(x0, y0); // 计算 k1
        double k2 = h * f(x0 + h, y0 + k1); // 计算 k2
        y0 = y0 + (k1 + k2) / 2; // 更新 y 值
        x0 = x0 + h; // 更新 x 值
        printf("x: %.1f, y: %.6f\n", x0, y0); // 输出结果
    }
    return 0;
}
```


在编写改进的 Euler 法程序时，有关输入输出部分，可参照以下屏幕上应出现内容。

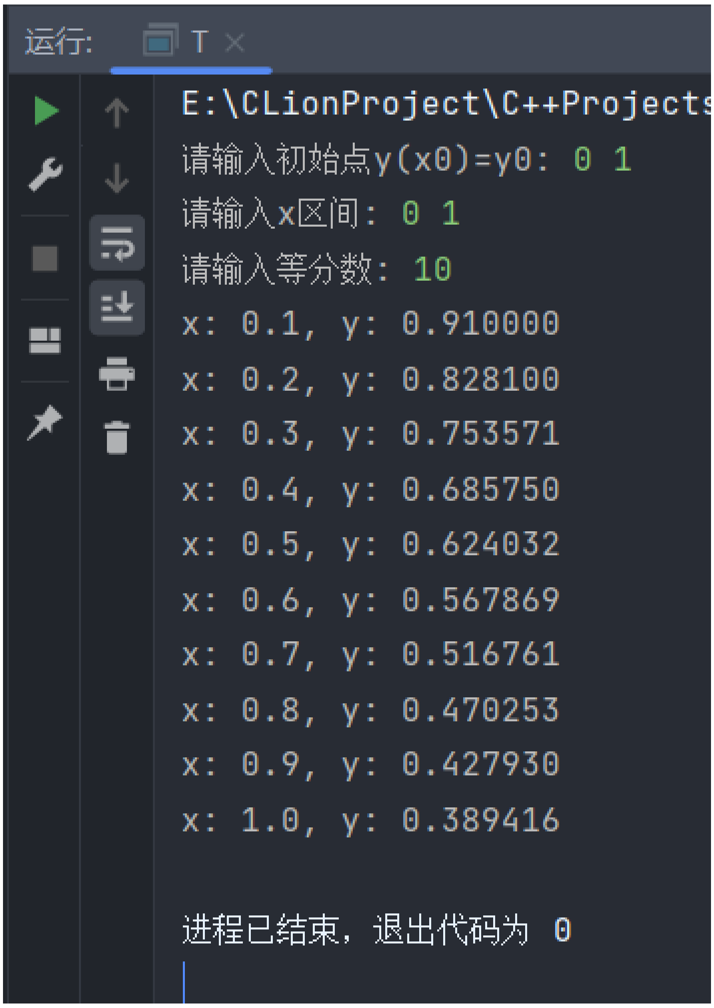
```
please input a,b,h and a0
0 1 0.1 1
x=0., y=1.
x=0., y=0.
x=0., y=0.
x=0., y=0.
x=0., y=0.
x=0., y=0.
x=0., y=0.
x=0., y=0.
```

x=0., y=0.
x=0., y=0.
x=1., y=0.

## （三）选做题

使用梯形公式编写程序，解决上述例子问题。

```
#include <bits/stdc++.h>

using namespace std;

double f(double x, double y) {
    return -y;
}

int main() {
    double h, x0, y0, a, b, n;
    cout << "请输入初始点 y(x0)=y0: ";
    cin >> x0 >> y0;
    cout << "请输入 x 区间: ";
    cin >> a >> b;
    cout << "请输入等分数: ";
    cin >> n;
    h = (b - a) / n;

    for (int i = 0; i < n; i++) {
        double k1 = h * f(x0 - h, y0);
        double k2 = h * f(x0, y0 + k1);
        y0 = y0 + k2;
        x0 = x0 + h;
        printf("x: %.1f, y: %.6f\n", x0, y0); // 输出结果
    }
    return 0;
}
```




## (四) 使用使用其他编程语言(Python、Java)

### python:

```python
def f(x, y):
    return -y


x0, y0 = map(float, input("请输入初始点 y(x0)=y0: ").split())
a, b = map(float, input("请输入 x 区间: ").split())
n = int(input("请输入等分数: "))
h = (b - a) / n

for i in range(n):
    y0 = y0 + h * f(x0, y0)  # 迭代计算下一个点的 y 值
   x0 = x0 + h  # 更新 x 值
   print(f"x: {x0:.1f}, y: {y0:.6f}")  # 输出结果
```

```
运行:  实验6欧拉法
E:\PycharmProjects\PythonNOTE\venv\Scr
请输入初始点y(x0)=y0: 0 1
请输入x区间: 0 1
请输入等分数: 10
x: 0.1, y: 0.900000
x: 0.2, y: 0.810000
x: 0.3, y: 0.729000
x: 0.4, y: 0.656100
x: 0.5, y: 0.590490
x: 0.6, y: 0.531441
x: 0.7, y: 0.478297
x: 0.8, y: 0.430467
x: 0.9, y: 0.387420
x: 1.0, y: 0.348678

进程已结束,退出代码0
```

## java:

```java
package 数值计算实验;

import java.util.Scanner;

public class 实验 6 欧拉法 {
    public static double f(double x, double y) {
        return -y;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double h, x0, y0, a, b, n;
        System.out.print("请输入初始点 y(x0)=y0: ");
        x0 = sc.nextDouble();
        y0 = sc.nextDouble();
        System.out.print("请输入 x 区间: ");
```

```java
        a = sc.nextDouble();
        b = sc.nextDouble();
        System.out.print("请输入等分数: ");
        n = sc.nextDouble();
        h = (b - a) / n;
        for (int i = 0; i < n; i++) {
            y0 = y0 + h * f(x0, y0); // 迭代计算下一个点的 y 值
          x0 = x0 + h; // 更新 x 值
          System.out.printf("x: %.1f, y: %.6f\n", x0, y0); // 输出结果
       }
    }
}
```