**数值计算实验大报告**

目录

[实验一 拉格朗日插值法（2 课时） 3](#_Toc168592416)

[一、实验目的 3](#_Toc168592417)

[二、实验要求 3](#_Toc168592418)

[三、实验原理 3](#_Toc168592419)

[四、实验内容 4](#_Toc168592420)

[实验二 最小二乘法（4 课时） 18](#_Toc168592421)

[一、实验目的 18](#_Toc168592422)

[二、实验要求 18](#_Toc168592423)

[三、实验原理 18](#_Toc168592424)

[四、实验内容 20](#_Toc168592425)

[实验三 数值积分(2 课时) 32](#_Toc168592426)

[一、实验目的 32](#_Toc168592427)

[二、实验要求 32](#_Toc168592428)

[三、实验原理 32](#_Toc168592429)

[四、实验内容 34](#_Toc168592430)

[实验四 高斯消元法(2 课时) 43](#_Toc168592431)

[一、实验目的 43](#_Toc168592432)

[二、实验要求 43](#_Toc168592433)

[三、实验原理 43](#_Toc168592434)

[四、实验内容 46](#_Toc168592435)

[实验五 非线性方程求解(2 课时) 52](#_Toc168592436)

[一、实验目的 52](#_Toc168592437)

[二、实验要求 52](#_Toc168592438)

[三、实验原理 52](#_Toc168592439)

[四、实验内容 53](#_Toc168592440)

[实验六 常微分方程初值问题数值解法(4 课时) 60](#_Toc168592441)

[一、实验目的 60](#_Toc168592442)

[二、实验要求 60](#_Toc168592443)

[三、实验原理 60](#_Toc168592444)

[四、实验内容 62](#_Toc168592445)

## 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。

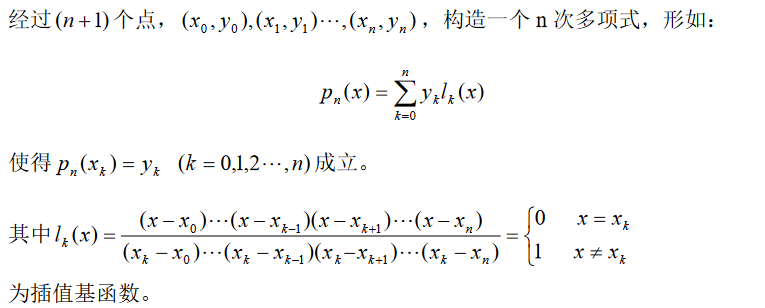
### 二、实验要求

1.用 C语言编程实现拉格朗日插值。

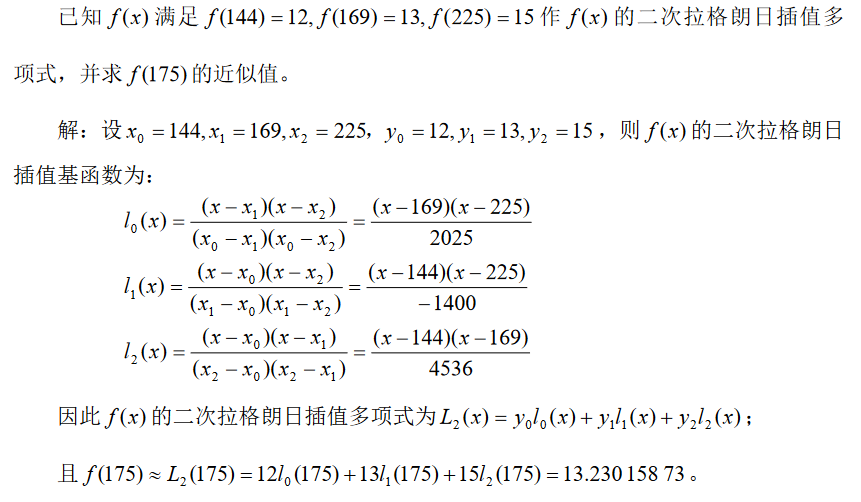
2.进一步加深对拉格朗日插值法的理解。

### 三、实验原理

#### (一) 拉格朗日插值公式



#### (二) 例子

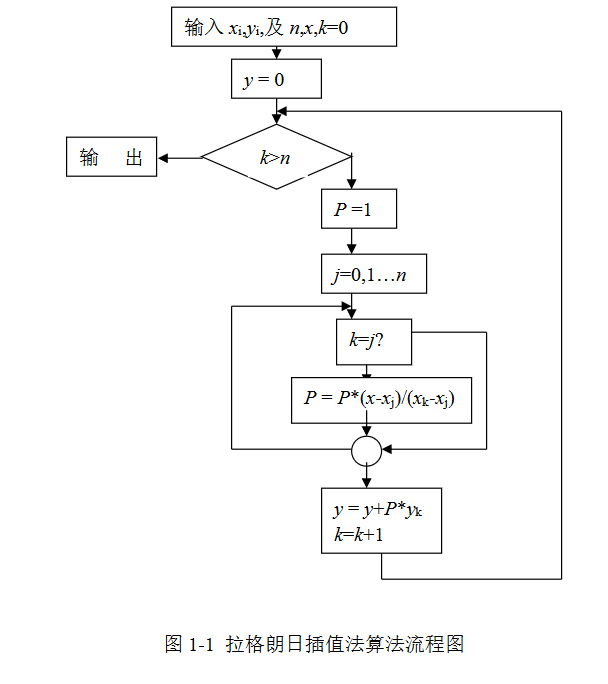


### 四、实验内容

#### (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。

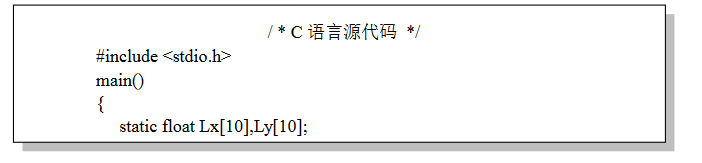


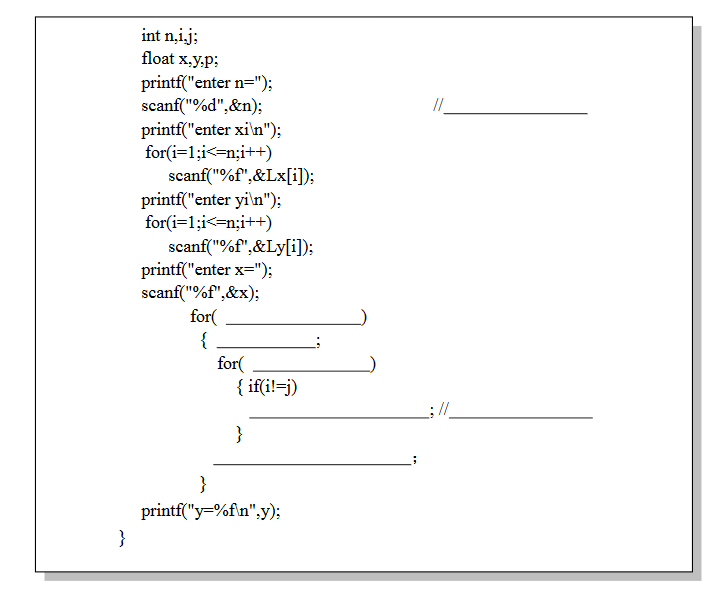
#### (二) 编程作业

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:





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 f(0.15)

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.?

*#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]);

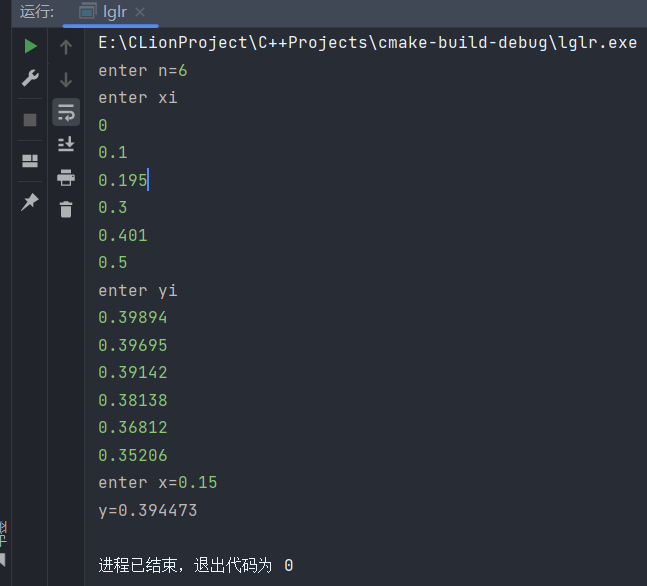
}  
 printf("enter yi\n");  
 fflush(stdout);  
 *for* (i = 1; i <= n; i++){

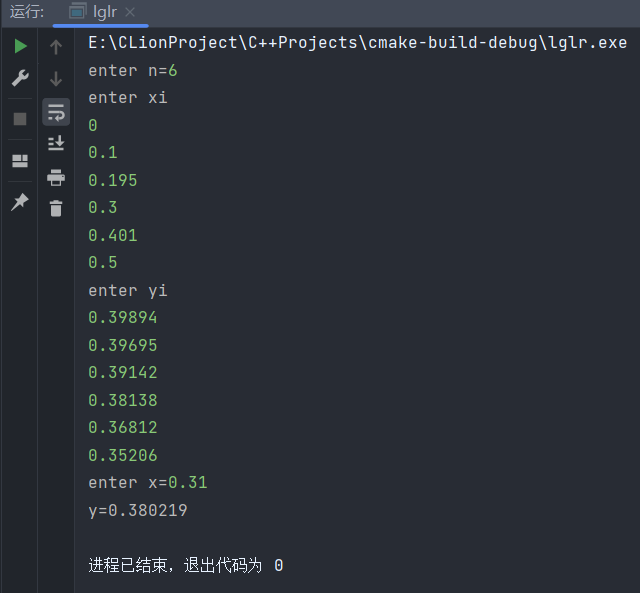
scanf("%f", &Ly[i]);

}  
 *// 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}* }

*// 拟合结果*  
 printf("y=%f\n", y);  
}

运行效果:





拟合x=0.15结果为y=0.394473

拟合x=0.31结果为y=0.380219

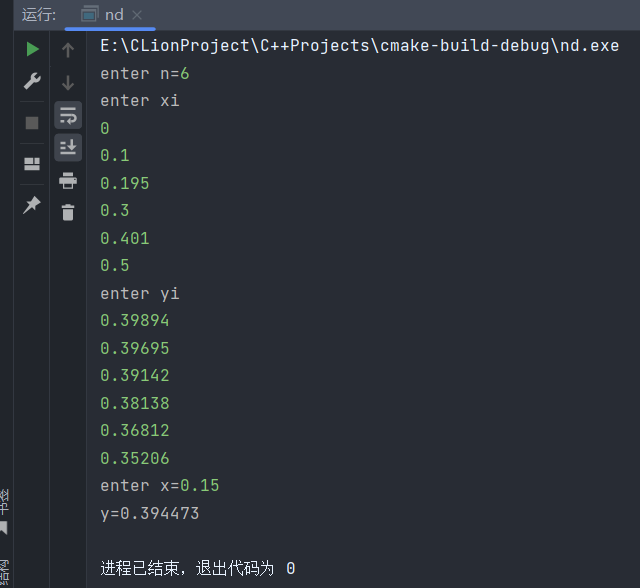
对比给出的数据点, 结果较为可信

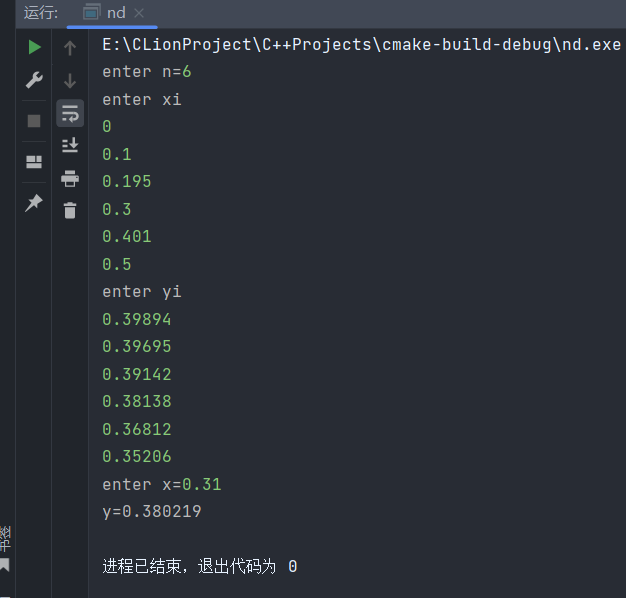
#### (三) 选做题

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。

*#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:

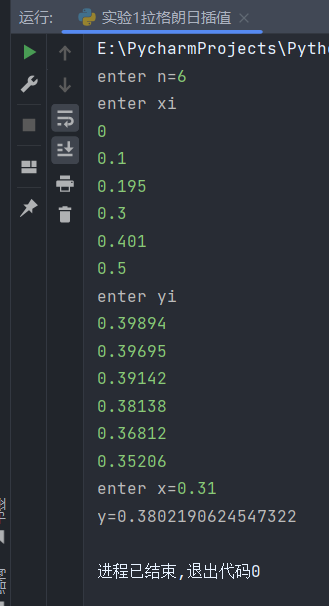
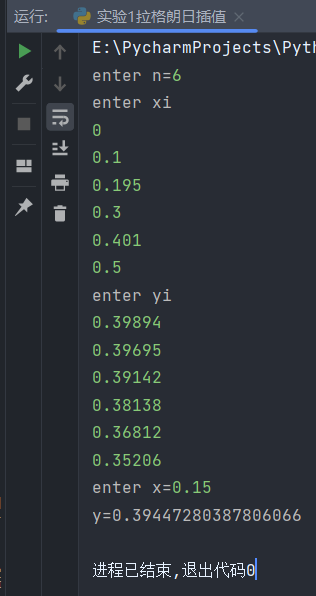




#### (四)使用其他编程语言(Python、Java)

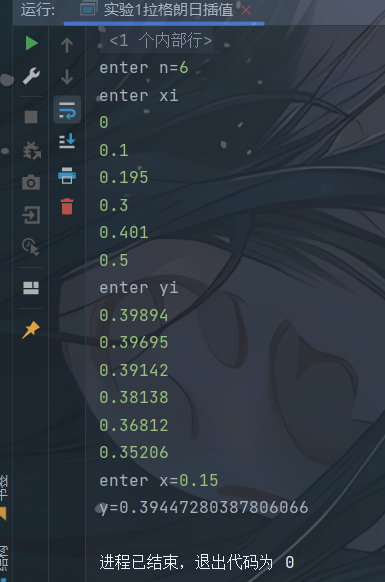
##### python:

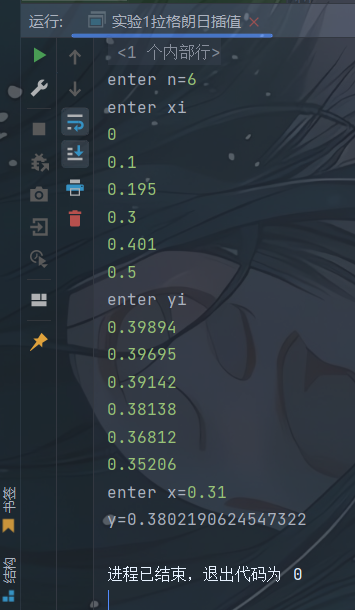
*def* lagrange\_interpolation(n, x\_values, y\_values, x):  
 y = 0  
 *for* i *in* range(n):  
 p = 1  
 *for* j *in* range(n):  
 *if* i != j:  
 p \*= (x - x\_values[j]) / (x\_values[i] - x\_values[j])  
 y += p \* y\_values[i]  
 *return* y  
  
  
n = int(input("enter n="))  
x\_values = []  
y\_values = []  
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:

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;  
 }  
}





## 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。

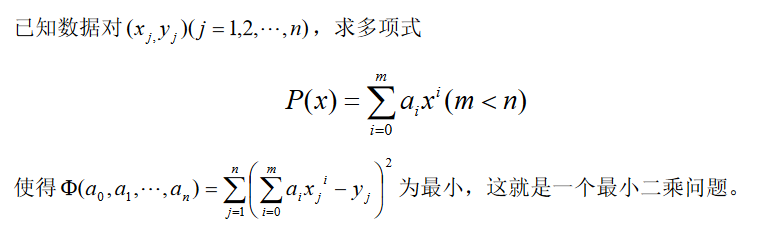
### 二、实验要求

1.掌握用 C 语言作最小二乘多项式拟合的方法。

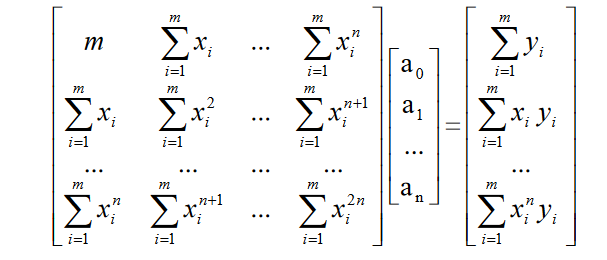
2.进一步加深对最小二乘法的理解。

### 三、实验原理

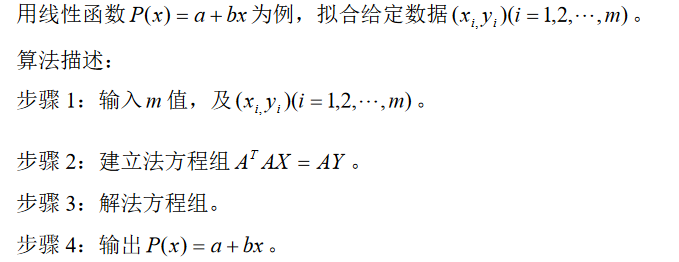
#### （一）最小二乘多项式拟合



#### （二）法方程组



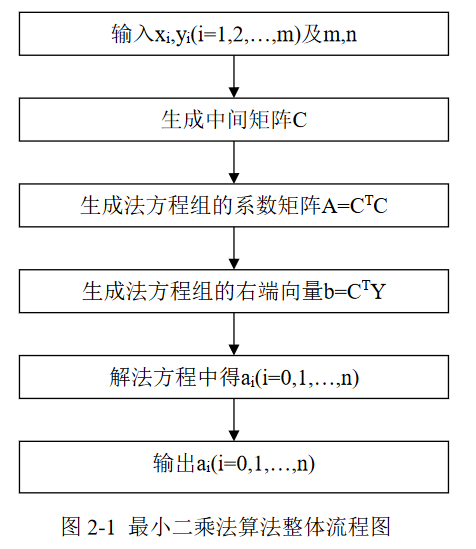
#### （三）最小二乘法计算步骤



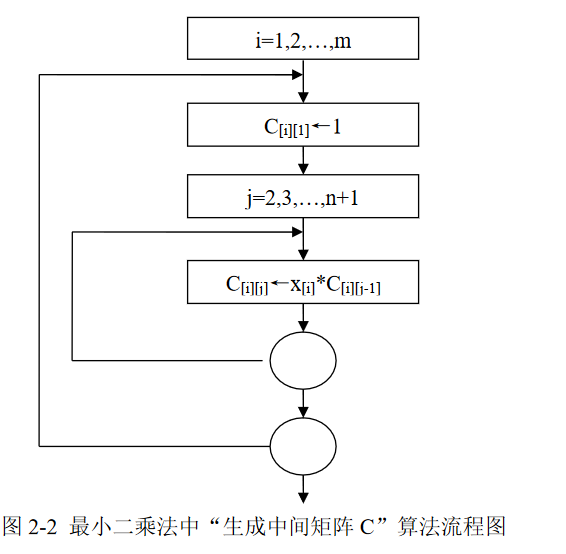
### 四、实验内容

#### （一）算法流程图

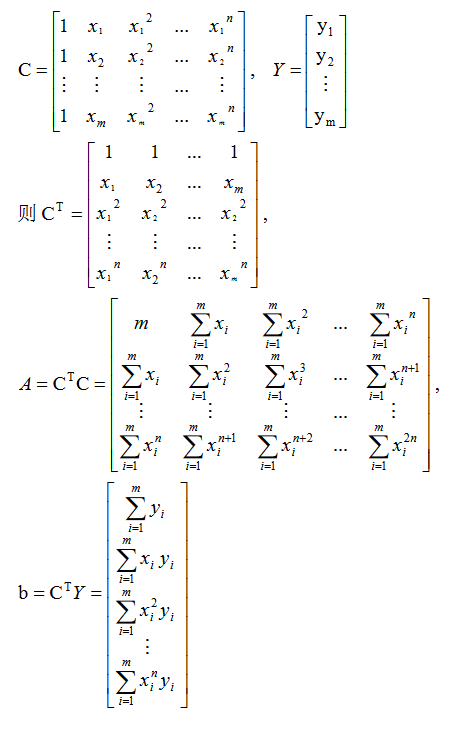
##### 1.算法整体流程图



##### 2.“生成中间矩阵 C”算法流程图

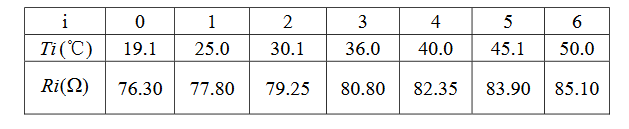


##### 3.中间矩阵 C 的重要作用



#### （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

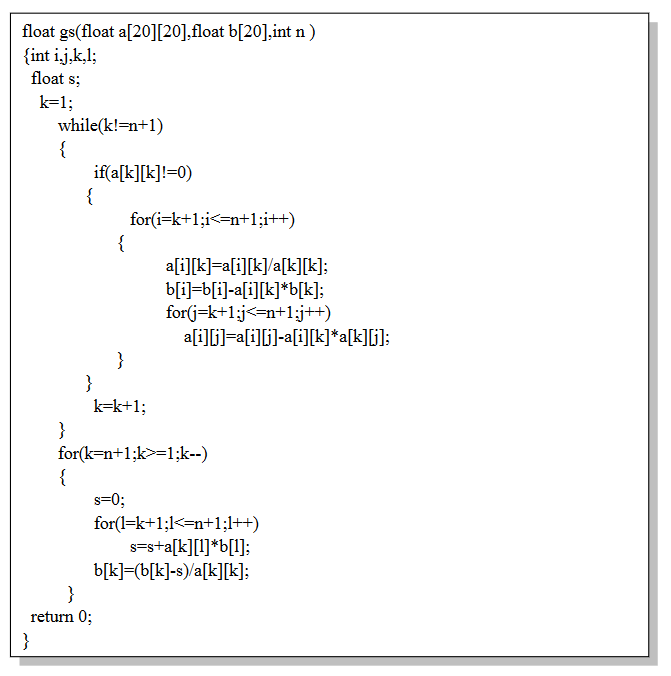


【提示】在进行程序实现时，务必注意中间矩阵的作用，以及非奇次线性方程组求

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。



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";  
}

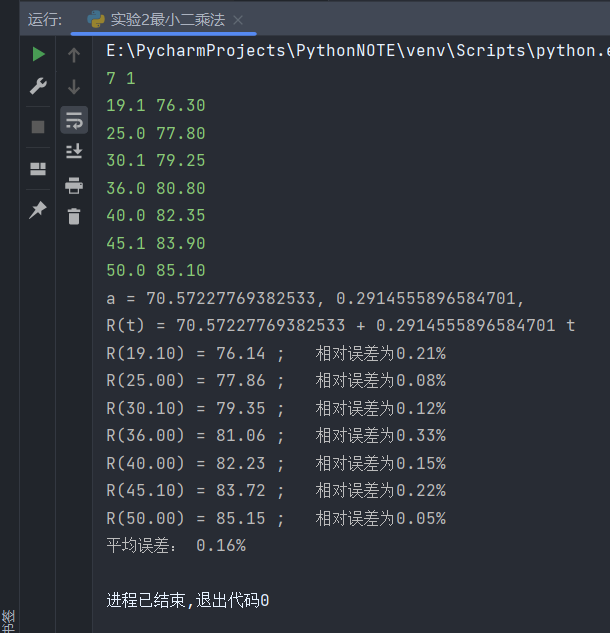
拟合结果:



#### (三)使用其他编程语言(Python、Java)

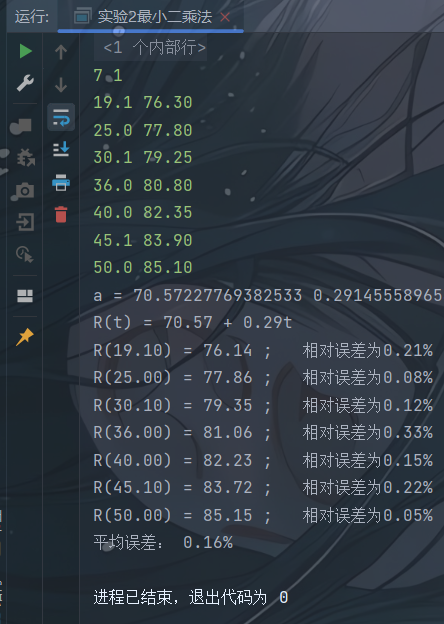
##### python:

*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())  
  
*# 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) {  
 // 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);  
 }  
}



## 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

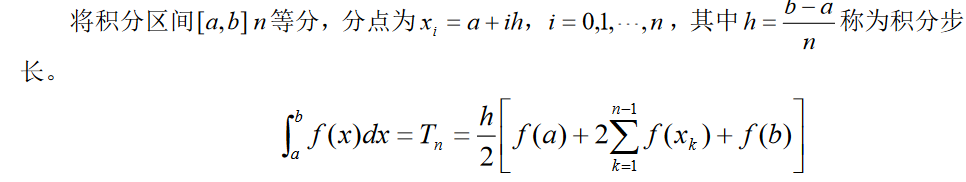
2.编写变步长复合梯形公式。

3.进一步加深对数值积分的理解。

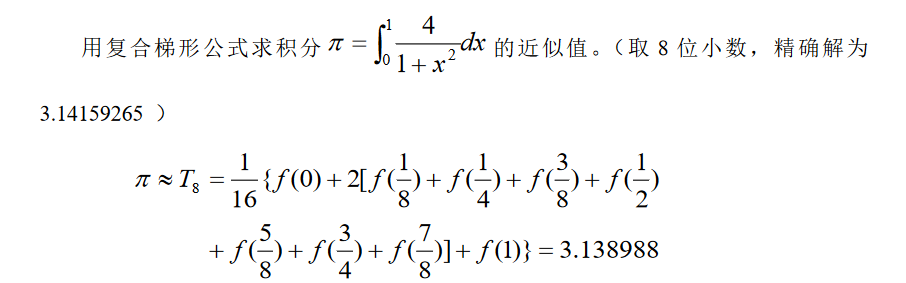
### 三、实验原理

#### (一)定步长复合梯形公式

##### 1.公式

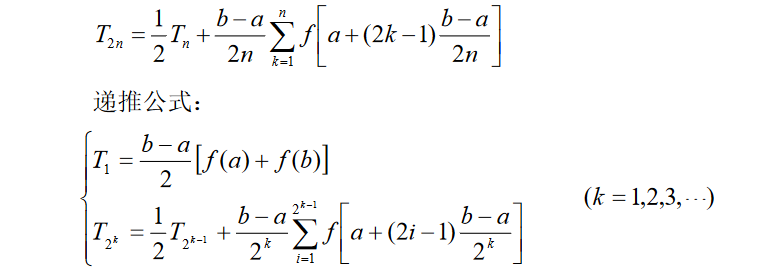


##### 2. 例子

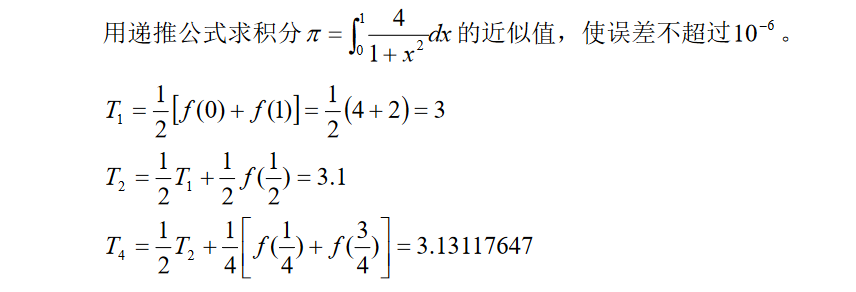


#### (二) 变步长复合梯形公式

##### 1. 公式



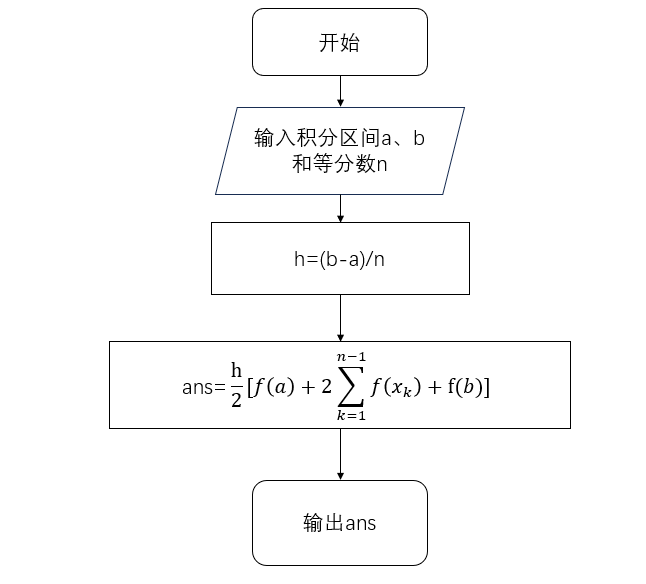
##### 2. 例子



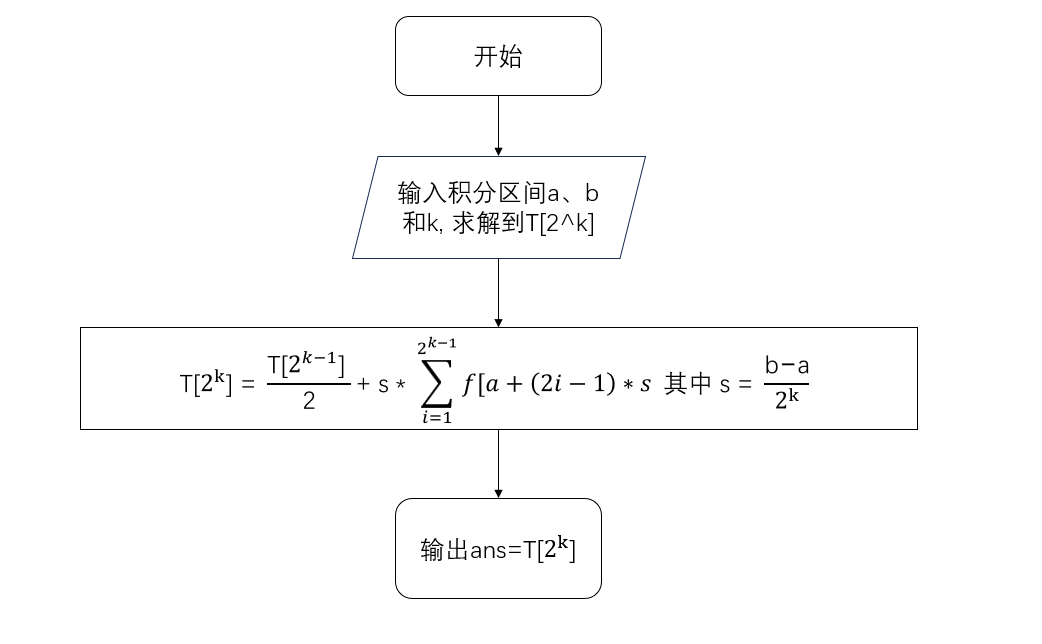
### 四、实验内容

#### (一) 算法流程图

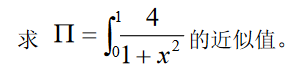
##### 1. 定步长复合梯形算法流程图



##### 2. 变步长复合梯形算法流程图



#### (二) 编程作业



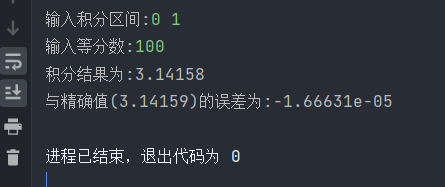
(1)编写定步长复合梯形程序求解上式;

(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

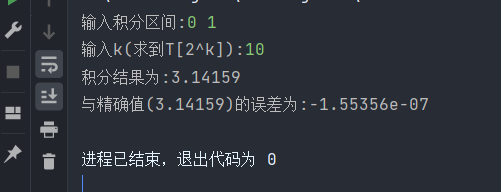
*#include* <bits/stdc++.h>  
  
*using namespace* std;  
  
*double* f(*double* x) {  
 *return* 4 / (1 + x \* x);  
}  
  
*int* main() {  
 cout << "输入积分区间:";  
 *double* a, b;  
 *int* n;  
 cin >> a >> b;  
 cout << "输入等分数:";  
 cin >> n;  
 *double* h = (b - a) / n;  
 *double* ans = 0;  
 ans += 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;  
}



##### 2.变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);  
}  
  
*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;  
}

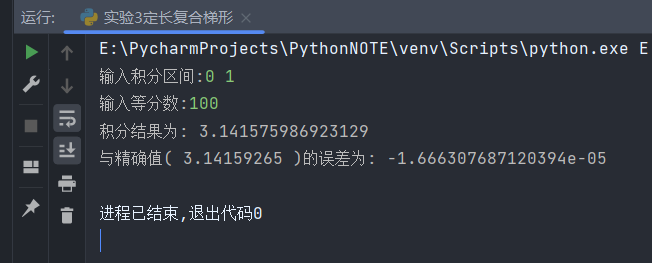
运行结果:



#### (三)使用使用其他编程语言(Python、Java)

##### python:

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

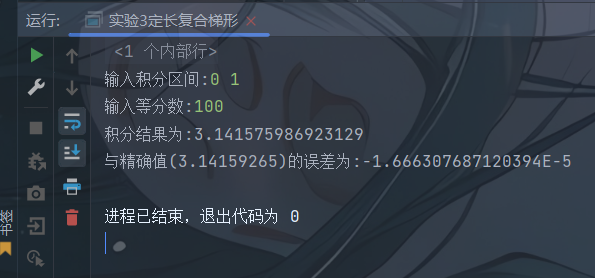


*def* f(x):  
 *return* 4 / (1 + x \* x)  
  
  
*def* getTk(k, a, b):  
 T = (f(a) + f(b)) / 2  
 *for* i *in* range(1, k + 1):  
 T /= 2  
 s = (b - a) / (1 << i)  
 t = 0  
 *for* j *in* range(1, (1 << (i - 1)) + 1):  
 x = a + (2 \* j - 1) \* s  
 t += f(x)  
 T += s \* t  
 *return* T  
  
  
a, 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)  
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))

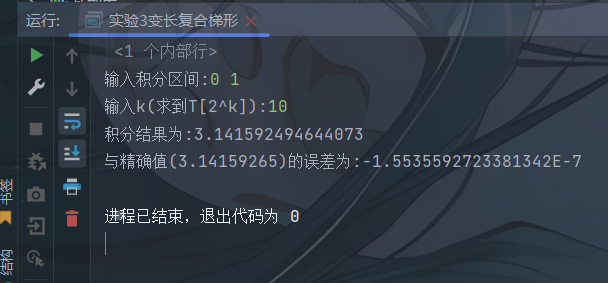


##### java:

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



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:");  
 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

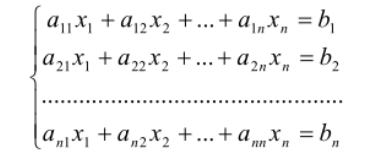
1.写出高斯选主元消去法解线性方程组算法，编写程序上机调试出结果

2.进一步加深对高斯消去法的理解。

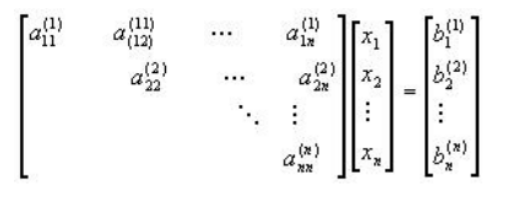
### 三、实验原理

#### (一)高斯消元法

Gauss消去法就是将方程组



通过(n-1)步消元，将上述方程组转化为上三角方程组



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



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



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

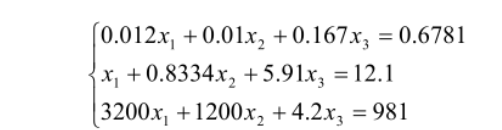
步选主元和消元运算,得到增广矩阵 B(n)=(A(n),b(n)).则方程组 A(n)x=b(n)是与原方程组

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

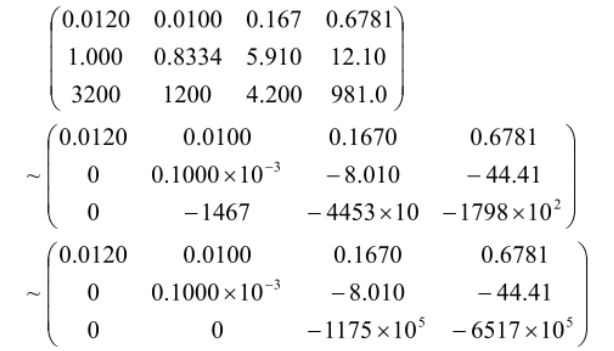
#### (三)例子

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:



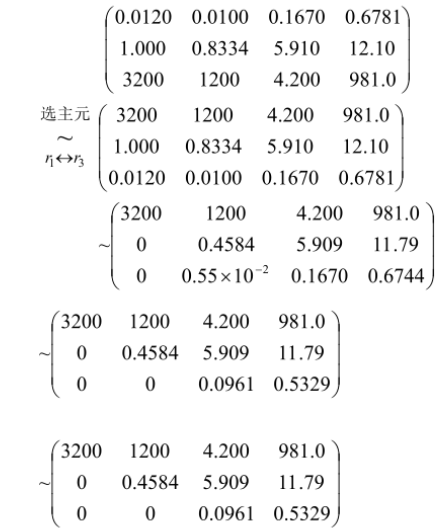
方程组具有四位有效数字的精确解为x1\*=17.46，x2\*=-45.76，x3\*=5.546

解(1)用顺序 Gauss 消去法求解，消元过程为



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

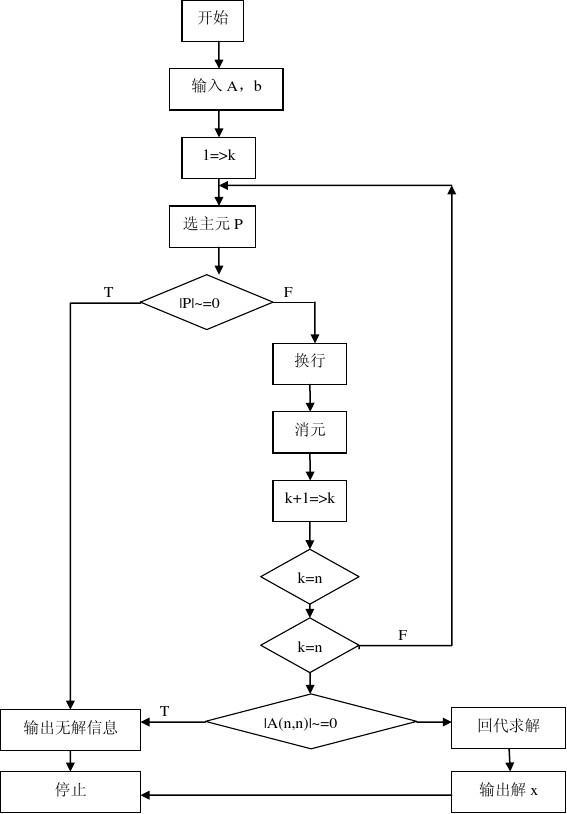


回代得:x3=5.545，x2=-45.77，x1=17.46

### 四、实验内容

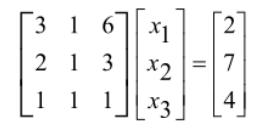
#### (一)算法流程图

高斯列主元消去法N-S图



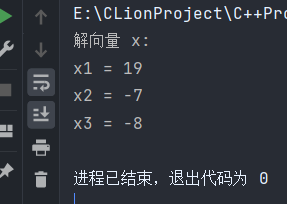
#### (二)编程作业

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。



*#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;  
}

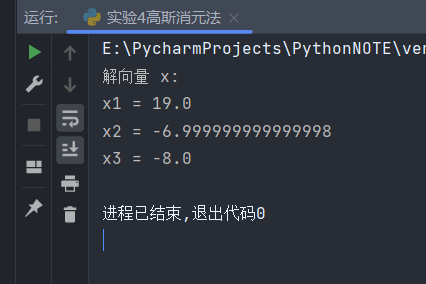
运行结果:



#### (三)使用使用其他编程语言(Python、Java)

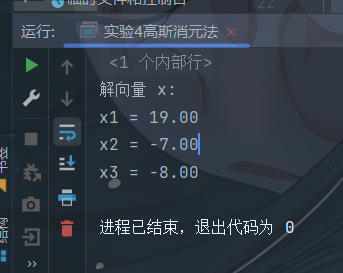
##### python:

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):  
 *# 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]}")



##### 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

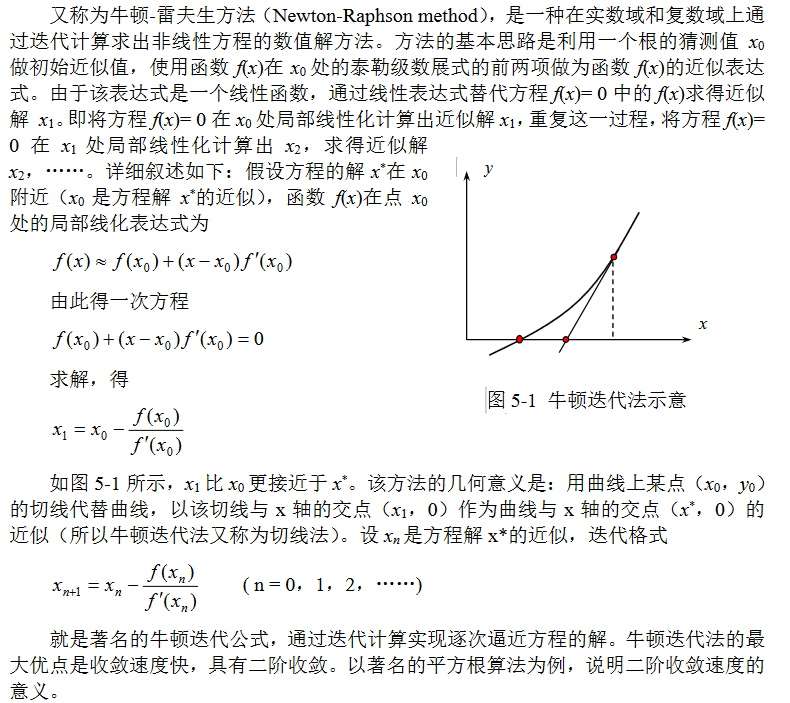
### 二、实验要求

1.设计牛顿迭代法算法，编写程序上机调试。

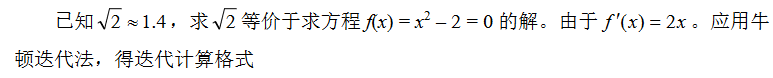
2.进一步加深对迭代法的理解。

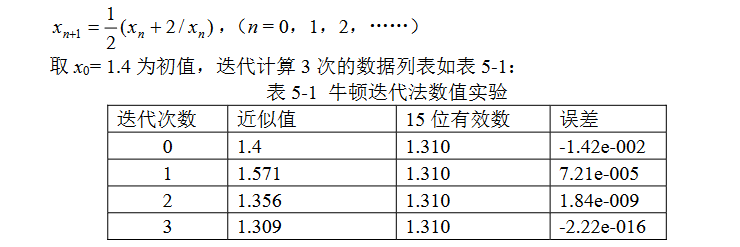
### 三、实验原理

#### (一)牛顿迭代法



#### (二) 例子

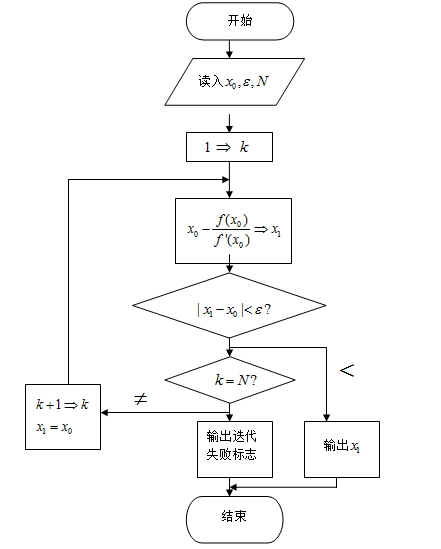
****

****

### 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

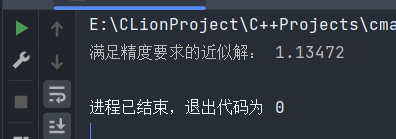
要求求解过程中用一个变量I控制三种状态，其中:

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  
 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;  
}

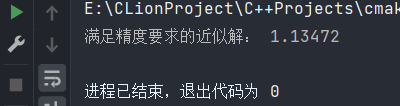


#### (三)选做题

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。

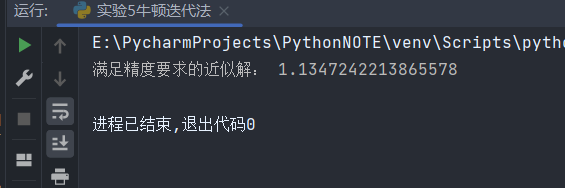
*#include* <iostream>  
*#include* <cmath>  
  
*double* f(*double* x) {  
 *return* pow(x, 6) - x - 1;  
}  
  
*double* df(*double* x) {  
 *return* 6 \* pow(x, 5) - 1;  
}  
  
*double* newton\_iteration(*double* x0, *double* acc, *int* max\_iter, *int* &i) {  
 *double* x = x0;  
 *for* (*int* iter = 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;  
 }  
 *// 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;  
}



#### (四)使用使用其他编程语言(Python、Java)

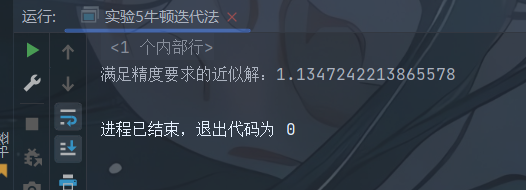
##### python:

*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)



##### 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);  
 }  
}



## 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。

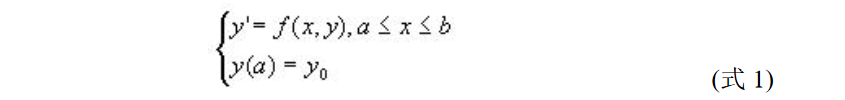
### 二、实验要求

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。

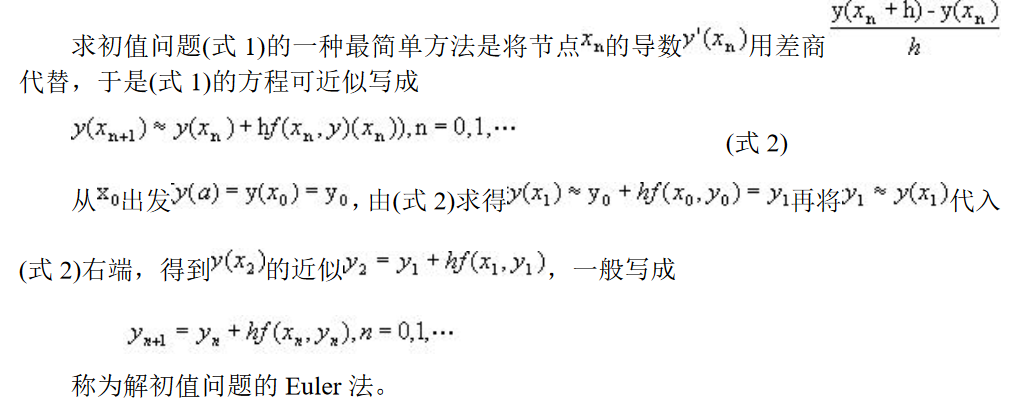
### 三、实验原理

#### （一）常微分方程初值问题

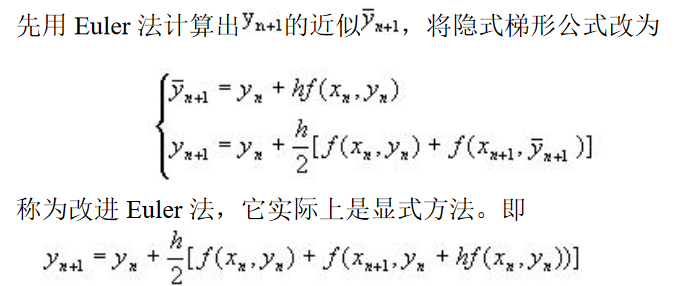


的数值解法，这也是科学与工程计算经常遇到的问题。

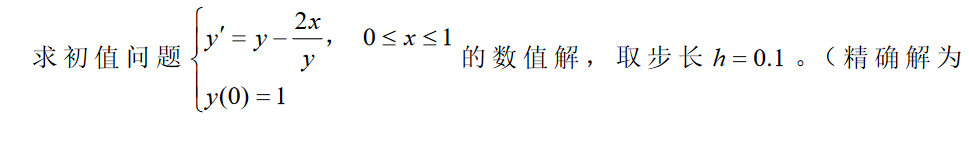
#### （二）欧拉法

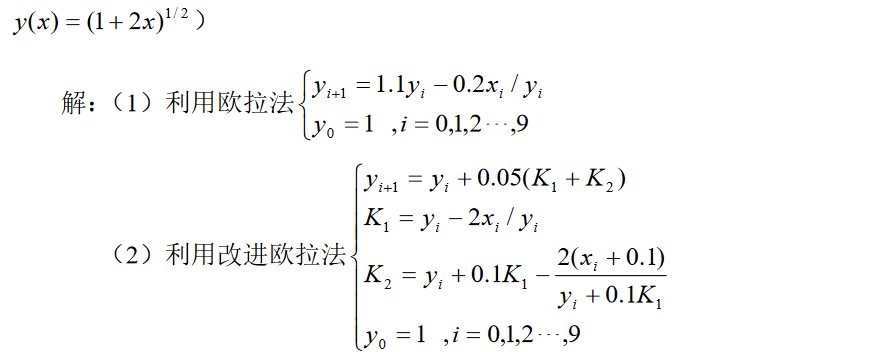


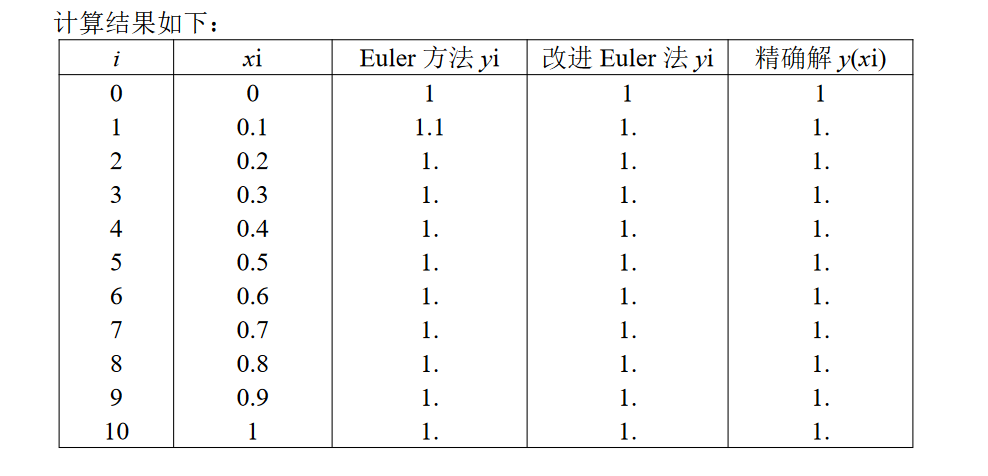
#### （三）改进欧拉法



#### （四）例子



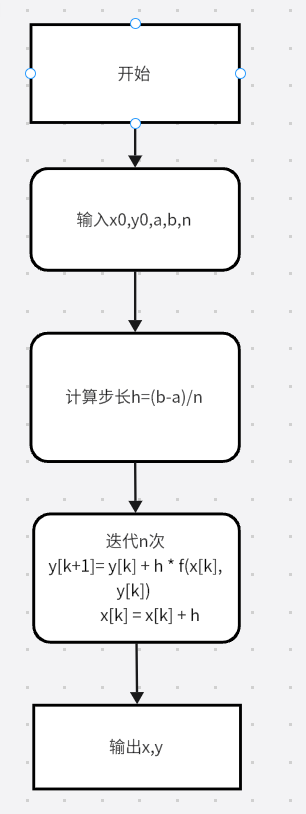
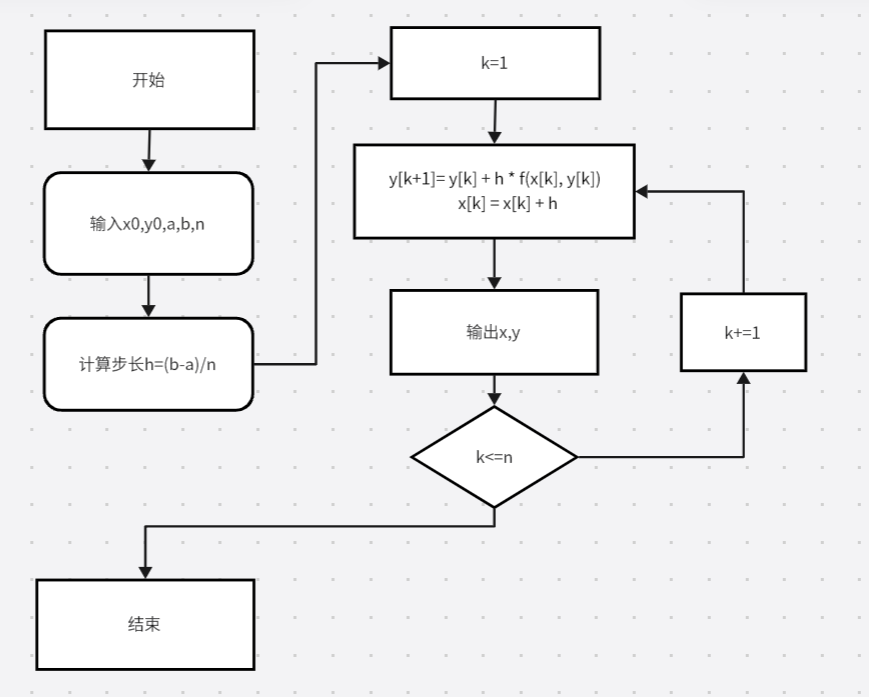


****

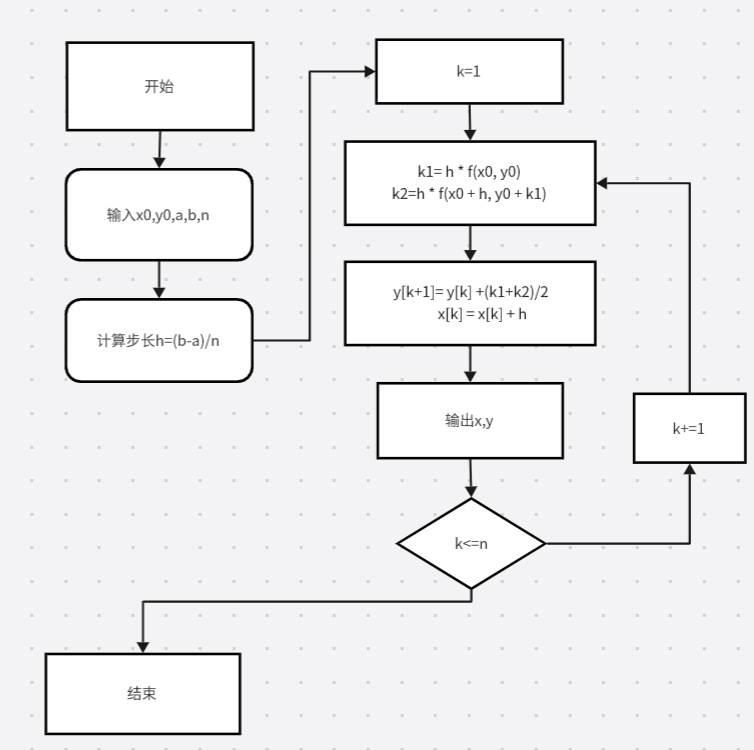
### 四、实验内容

#### （一）算法流程图

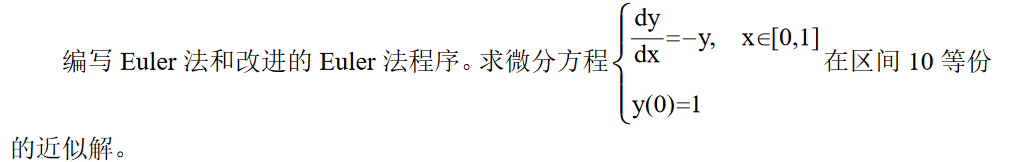
##### 1. 请根据欧拉公式，画出其算法流程图



##### 2. 请根据改进欧拉公式，画出其算法流程图。

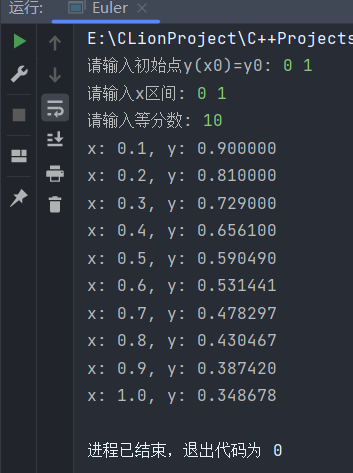


#### （二）编程作业



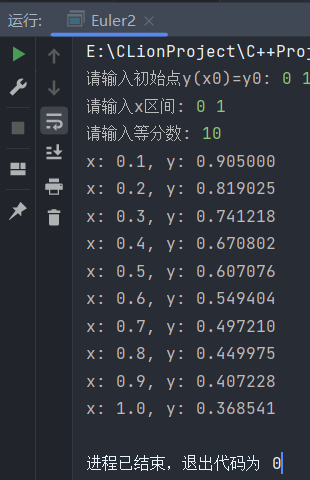
##### 1. Euler法程序

*#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.

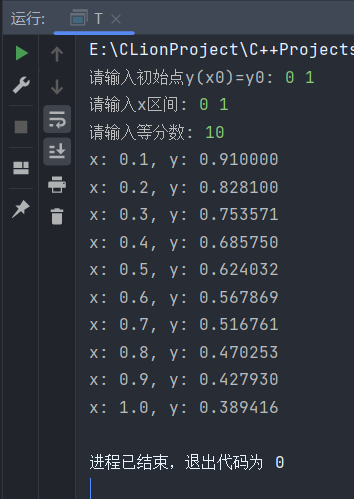
x=0., y=0.

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。

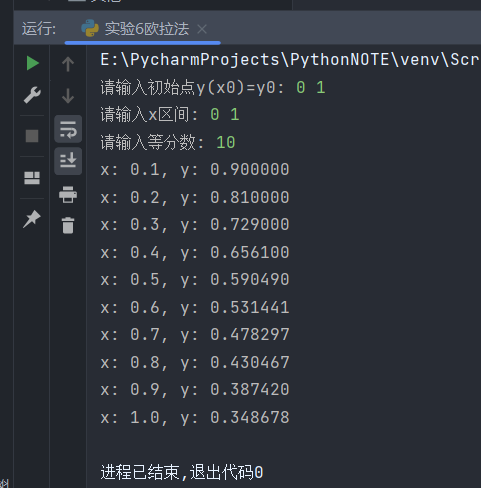
*#include* <bits/stdc++.h>  
  
*using namespace* std;  
  
*double* f(*double* x, *double* y) {  
 *return* -y;  
}  
  
*int* main() {  
 *double* h, x0, y0, a, b, n;  
 cout << "请输入初始点y(x0)=y0: ";  
 cin >> x0 >> y0;  
 cout << "请输入x区间: ";  
 cin >> a >> b;  
 cout << "请输入等分数: ";  
 cin >> n;  
 h = (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:

*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*



##### 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: ");  
 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  
 }  
 }  
}

