



**《数值逼近》实验报告**

**学 院**：

**班 级**：

**学 号**：

**姓 名**： 李

**指导教师**： 于、章、

聂、赵

教 务 处

2021 年 5 月

**实验一、插值法**

1. 实验目的

根据数据构造 Newton 插值多项式，并计算当年 10 月 8 日北京市的日照时长。

二、实验内容（题目、代码、结果）

2、天安门广场升旗的时间是日出的时刻，而降旗的时间是日落时分。根据天安门广场 管理委员会的公告，某年 10 月份升降旗的时间如下：

|  |  |
| --- | --- |
| 日期 1 15 22 | |
| 升旗  降旗 | 6:09 6:23 6:31 |
| 17:58 17:36 17:26 |

请根据上述数据构造 Newton 插值多项式，并计算当年 10 月 8 日北京市的日照时长。

代码如下：

*//显示中文*

#define **\_CRT\_SECURE\_NO\_WARNINGS**

#include <windows.h>*//用于函数SetConsoleOutputCP(65001);更改cmd编码为utf8*

#include<iostream>

#include<stdio.h>

#include<math.h>

#define **n** 3*//节点个数*

#define **f**(x) 4.0/(1+(x)\*(x))

using namespace **std**;

*//Newton插值多项式*

void **test01**()

{

    int i,j,xx,x[3] = {1,15,22};

    float y[3] = {17-6+(58-9)/60.0,17-6+(36-23)/60.0,17-6+(26-31)/60.0};

    float N[3][3],yy,temp;

    for(i=0;i<n;i++)

    {

        N[i][0] = y[i];*//0阶差商*

    }

    for(j=1;j<n;j++)

    {

        for(i=j;i<n;i++)

        {

            N[i][j] = (N[i][j-1]-N[i-1][j-1])/(x[i]-x[i-j]);*//构造差商*

        }

    }

    xx = 8;

    yy = 0.0;

    temp = 1.0;

    for(i=0;i<n;i++)

    {

        yy = yy + N[i][i]\*temp;

        temp = temp\*(xx-x[i]);

    }

**printf**("\n这年10月%d日北京日照时长为%7.4f小时.\n",xx,yy);

**printf**("即%d小时%3.0f分.\n",(int)**floor**(yy),60\***fmod**(yy,1.0));

}

int **main**()

{

*//显示中文*

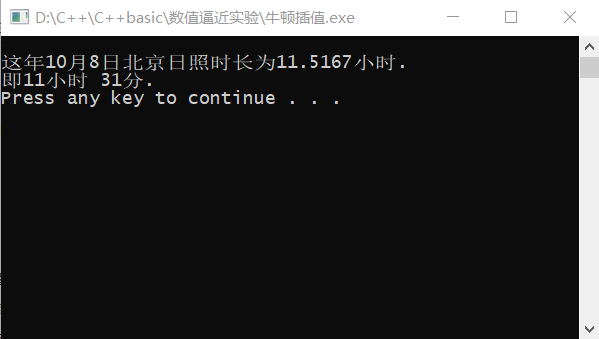
**SetConsoleOutputCP**(65001);

**test01**();

**system**("pause");

    return 0;

}

结果图示：

**实验二、数值积分**

一、实验目的

用复化梯形公式和复化 Simpson 公式计算定积分的近似值。

1. 实验内容（题目、代码、结果）

题目：

3、将区间四等分，分别用复化梯形公式和复化 Simpson 公式计算定积分的近似值。

代码：

#include<iostream>

using namespace **std**;

#include<stdio.h>

*//显示中文*

#define **\_CRT\_SECURE\_NO\_WARNINGS**

#include <windows.h>*//用于函数SetConsoleOutputCP(65001);更改cmd编码为utf8*

#define  **f**(x)  4.0 / (1 + (x) \* (x))

void **test**()

{

    float h = (1.0 - 0) / 4, temp, xk, yk, xkh, ykh, xk1, yk1;

    int i;

    temp = **f**(0); xk = 0;

    for (i = 1; i < 4; i++)

    {

        xk = xk + h;

        temp = temp + 2 \* **f**(xk);

    }

    temp = temp + **f**(1);

    temp = temp \* h / 2;

    cout << endl;

    cout << "复化梯形公式计算的结果：" << temp << endl;

    temp = 0;

    h = (1.0 - 0) / 2;

    xk = 0;

    yk = **f**(0);

    for (i = 0; i < 2; i++)

    {

        xkh = xk + h / 2;

        ykh = **f**(xkh);

        xk1 = xk + h;

        yk1 = **f**(xk1);

        temp = temp + h \* (yk + 4 \* ykh + yk1) / 6;

        xk = xk1;

        yk = yk1;

    }

    cout << endl;

    cout << "复化 Simpson 公式计算的结果：" << temp << endl;

    cout << endl;

}

int **main**()

{

*//显示中文*

**SetConsoleOutputCP**(65001);

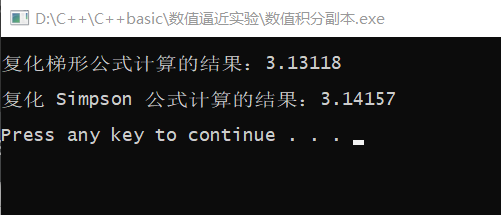
**test**();

**system**("pause");

    return 0;

}

结果图示：



**实验三、常微分方程数值解法**

一、实验目的

分别用 Euler 方法、改进 Euler 方法和经典四阶 Runge-Kutta 方法求解初值问题。

1. 实验内容（题目、代码、结果）

题目：

4、取步长 h = 0.1 ，分别用 Euler 方法、改进 Euler 方法和经典四阶 Runge-Kutta 方法求解初值问题



并将计算结果与精确解相比较。

代码：

#include<stdio.h>

#include<math.h>

#include<iostream>

#define **\_CRT\_SECURE\_NO\_WARNINGS**

#include <windows.h>*//用于函数SetConsoleOutputCP(65001);更改cmd编码为utf8*

using namespace **std**;

#define **f**(x,y)  ((y)-2\*(x)/(y))

#define **y**(x)    **sqrt**(1+2\*(x))

int **main**()

{

*//显示中文*

**SetConsoleOutputCP**(65001);

    float a = 0,b = 1.0f,h=0.1f,y0=1.0f,x,ye,yp,ym,k1,k2,k3,k4,yr,yx;

**printf**("\n    精确解            Euler方法");

**printf**("          改进Euler方法     4阶Runge-Kutta方法");

**printf**("\n x       y      ye[k]    |ye[k]-y|   ");

**printf**("ym[k]   |ym[k]-y|   yr[k]   |yr[k]-y|\n");

**printf**("%3.1f  %8.6f  %8.6f  %8.6f  ",a,y0,y0,0);

**printf**("%8.6f  %8.6f  %8.6f  %8.6f  \n",y0,0,y0,0.0);

    x=a;

    ye=y0;*//Euler方法的初值*

    ym=y0;*//改进的Euler方法的初值*

    yr=y0;*//4阶Runge-Kutta方法的初值*

    while(x<b)

    {

        ye = ye + h\***f**(x,ye);

        yp = ym + h\***f**(x,ym);

        ym = ym + h/2\*(**f**(x,ym)+**f**(x+h,yp));

        k1 = **f**(x,yr);

        k2 = **f**(x+h/2,yr+h/2\*k1);

        k3 = **f**(x+h/2,yr+h/2\*k2);

        k4 = **f**(x+h,yr+h\*k3);

        yr = yr + h/6\*(k1+2\*k2+2\*k3+k4);

        x = x + h;

        yx = **y**(x);

**printf**("%3.1f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f\n",x,yx,ye,**fabs**(ye-yx),ym,**fabs**(ym-yx),yr,**fabs**(yr-yx));

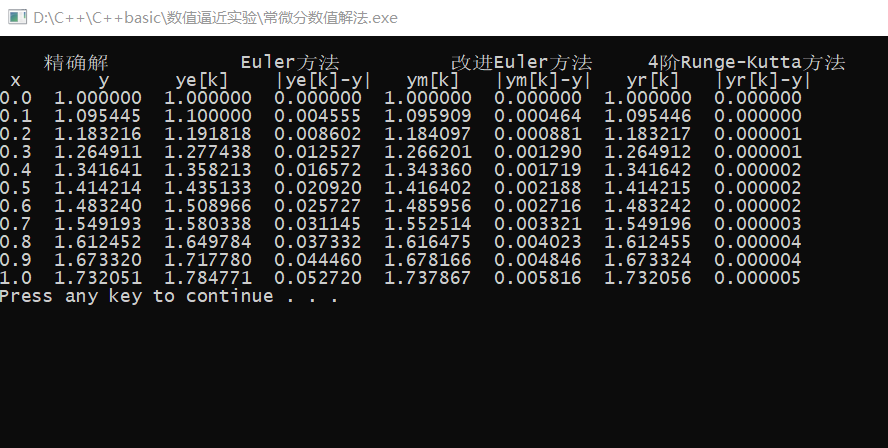
    }

**system**("pause");

    return 0;

}

结果图示：



**实验四、方程求根常用方法**

一、实验目的

用迭代法求非线性方程的根。

二、实验内容（题目、代码、结果）

题目：

6、用迭代法求非线性方程的根。

取迭代函数，精度要求为，最多迭代 100 次。

代码：

#include<stdio.h>

#include<math.h>

#define **phi**(x)   0.5\*(3+**sin**((x))-1-(x)\*(x))*//迭代函数*

*//显示中文*

#define **\_CRT\_SECURE\_NO\_WARNINGS**

#include <windows.h>*//用于函数SetConsoleOutputCP(65001);更改cmd编码为utf8*

#include<iostream>

using namespace **std**;

int **main**()

{

*//显示中文*

**SetConsoleOutputCP**(65001);

    int n=1,N;

    float x,x0,del;

**printf**("x0 = ");

**scanf**("%f",&x0);

**printf**("\ndel = ");

**scanf**("%f",&del);

**printf**("\nN = ");

**scanf**("%d",&N);

**printf**("\n k     x(k)");

**printf**("\n  %2d  %f",0,x0);

    while (n<N)

    {

        x = **phi**(x0);

        if(**fabs**(x-x0)<del)

        {

**printf**("\n\n 近似解 = %f \n",x);

            return 0;

        }

**printf**("\n %2d   %f",n,x0);

        n = n+1;

        x0 = x;

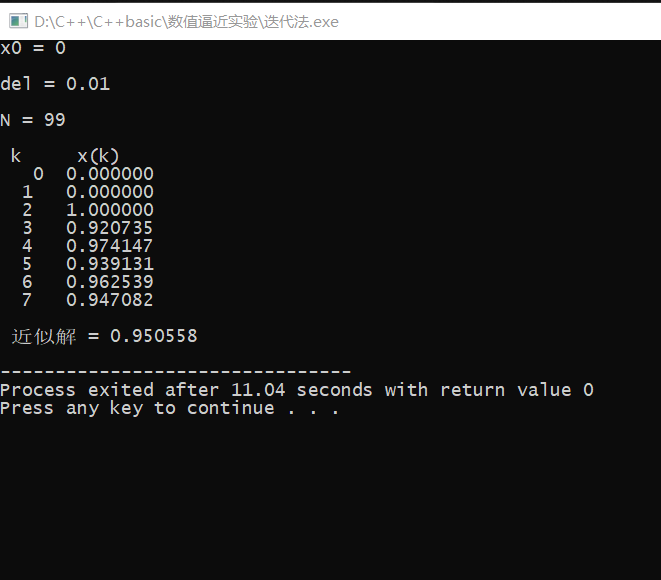
    }

**printf**("\n\n%d次迭代后未达到精度要求.\n",N);

**system**("pause");

}

结果图示：



**实验五、算法设计与实现综合**

一、实验目的

编写Hermite 插值法通用程序解决问题。

二、实验内容（题目、代码、结果）

题目：

8、已知函数在 10 个节点上的函数值和导数值，如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | 0.10 | 0.15 | 0.30 | 0.45 | 0.55 |
|  | 0.904837 | 0.860708 | 0.740818 | 0.637628 | 0.576950 |
|  | - 0.904837 | 0.860708 | -0.740818 | -0.637628 | -0.576950 |
| x | 0.60 | 0.70 | 0.85 | 0.90 | 1.00 |
|  | 0.548812 | 0.496585 | 0.427415 | 0.406570 | 0.367879 |
|  | -0.548812 | -0.496585 | -0.427415 | -0.406570 | -0.367879 |

用熟悉的计算机语言编写 Hermite 插值法通用程序，计算 f (0.356) 的近似值，并与 f (0.356) 比较。

代码：

#include<iostream>

#include<stdio.h>

#include<math.h>

#include<cmath>

using namespace **std**;

float X[10] = {0.10,0.15,0.30,0.45,0.55,0.60,0.70,0.85,0.90,1.00};

float Y[10] = {0.904837,0.860708,0.740818,0.637628,0.576950,0.548812,0.496585,0.427415,0.406570,0.367879};

float YY[10] = {-0.904837,-0.860708,-0.740818,-0.637628,-0.576950,-0.548812,-0.496585,-0.427415,-0.406570,-0.367879};

*//#define f(x) e^(-x);*

float **f**(float x)

{

    return **exp**(-x);

}

float **Hermite**(float \*x, float  \*y, float  \*dy, int n, float t)

{

    float result[10];

    for(int i = 0;i<n;i++)

    {

        float h = 1.0;

        float a = 0.0;

        for(int j = 0;j<n;j++)

        {

            if(j!=i)

            {

                h = h\*(**pow**(t-x[j],2))/(**pow**(x[i]-x[j],2));

                a = a + 1/(x[i]-x[j]);

            }

        }

        result[i] = h\*((x[i]-t)\*(2\*a\*y[i]-dy[i])+y[i]);

    }

    float r = 0;

    for(int k = 0;k<n;k++)

    {

        r = r + result[k];

    }

    return r;

}

int **main**()

{

    float t = 0.356;

    cout<<"近似值为"<<**Hermite**(X, Y, YY, 10, t)<<endl;

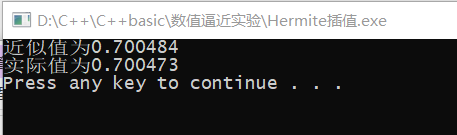
    cout<<"实际值为"<<**f**(t)<<endl;

**system**("pause");

    return 0;

}

结果图示：



**实验六、算法设计与实现综合**

一、实验目的

求方程的根（可取 x0 = 1 ），计算到 Leonardo 所得到的准确度。

二、实验内容（题目、代码、结果）

题目：

9、公元 1225 年，比萨的数学家 Leonardo（即 Fibonacci（斐波那契）），1170-1250）研 究了方程



得到一个根 x\* ≈ 1.368 808 107 ，没有人知道他用什么方法得到这个值。对于这个方程， 分别用下列方法：

（1）迭代法；

（2）迭代法；

（3）对（1）的 Aitken 加速方法；

（4）对（2）的 Aitken 加速方法；

（5）Newton 法 。

求方程的根（可取 x0 = 1 ），计算到 Leonardo 所得到的准确度。计算机语言不限。

代码：

#include<stdio.h>

#include<math.h>

*//显示中文*

#define **\_CRT\_SECURE\_NO\_WARNINGS**

#include <windows.h>*//用于函数SetConsoleOutputCP(65001);更改cmd编码为utf8*

#include<iostream>

using namespace **std**;

#define **phi01**(x)   20/((x)\*(x)+2\*(x)+10)*//迭代函数*

#define **phi02**(x) (20-2\*((x)\*(x))-((x)\*(x)\*(x)))/10

#define **f**(x)   ((x)\*(x)\*(x))+2\*((x)\*(x))+10\*(x)-20

#define **f1**(x)  3\*((x)\*(x))+4\*(x)+10

void **test01**()

{

    int n=1,N;

    double x,x0,del;

**printf**("x0 = 1");

*//scanf("%f",&x0);*

    x0 = 1;

**printf**("\ndel = 0.000000001");

*//scanf("%f",&del);*

    del = 0.000000001;

**printf**("\nN = 100");

*//scanf("%d",&N);*

    N = 100;

**printf**("\n k     x(k)");

**printf**("\n  %2d  %f",0,x0);

    while (n<N)

    {

        x = **phi01**(x0);

        if(**fabs**(x-x0)<del)

        {

**printf**("\n\n 迭代法式1的近似解 = %.9lf \n",x);

            return;

        }

**printf**("\n %2d   %.9lf",n,x0);

        n = n+1;

        x0 = x;

    }

**printf**("\n\n%d次迭代后未达到精度要求.\n",N);

}

void **test02**()

{

    int n=1,N;

    double x,x0,del;

**printf**("x0 = 1");

*//scanf("%f",&x0);*

    x0 = 1;

**printf**("\ndel = 0.000000001");

*//scanf("%f",&del);*

    del = 0.000000001;

**printf**("\nN = 20");

*//scanf("%d",&N);*

    N = 20;

**printf**("\n k     x(k)");

**printf**("\n  %2d  %f",0,x0);

    while (n<N)

    {

        x = **phi02**(x0);

        if(**fabs**(x-x0)<del)

        {

**printf**("\n\n 迭代法式2的近似解 = %.9lf \n",x);

            return;

        }

**printf**("\n %2d   %.9lf",n,x0);

        n = n+1;

        x0 = x;

    }

**printf**("\n\n%d次迭代后未达到精度要求.\n",N);

}

void **Aitken01**()

{

    int n=1,N;

    double x0,x1,x2,x3,del;

**printf**("x0 = 1");

*//scanf("%f",&x0);*

    x0 = 1;

**printf**("\ndel = 0.000000001");

*//scanf("%f",&del);*

    del = 0.000000001;

**printf**("\nN = 100");

*//scanf("%d",&N);*

    N = 100;

**printf**("\n k     x(k)");

**printf**("\n  %2d  %f",0,x0);

    while (n<N)

    {

        x1 = **phi01**(x0);

        x2 = **phi01**(x1);

        x3 = x2-(((x2-x1)\*(x2-x1))/(x2-2\*x1+x0));

        if(**fabs**(x3-x0)<del)

        {

**printf**("\n\n 迭代法式1的Aitken近似解 = %.9lf \n",x3);

            return;

        }

**printf**("\n n =   %2d  x1 =  %.9lf",n,x3);

        n = n+1;

        x0 = x3;

    }

**printf**("\n\n%d次迭代后未达到精度要求.\n",N);

}

void **Aitken02**()

{

    int n=1,N;

    double x0,x1,x2,x3,del;

**printf**("x0 = 1");

*//scanf("%f",&x0);*

    x0 = 1;

**printf**("\ndel = 0.000000001");

*//scanf("%f",&del);*

    del = 0.000000001;

**printf**("\nN = 100");

*//scanf("%d",&N);*

    N = 100;

**printf**("\n k     x(k)");

**printf**("\n  %2d  %f",0,x0);

    while (n<N)

    {

        x1 = **phi02**(x0);

        x2 = **phi02**(x1);

        x3 = x2-(((x2-x1)\*(x2-x1))/(x2-2\*x1+x0));

        if(**fabs**(x3-x0)<del)

        {

**printf**("\n\n 迭代法式2的Aitken近似解 = %.9lf \n",x3);

            return;

        }

**printf**("\n n =   %2d  x1 =  %.9lf",n,x3);

        n = n+1;

        x0 = x3;

    }

**printf**("\n\n%d次迭代后未达到精度要求.\n",N);

}

void **Newton**()

{

    int n=1,N;

    double x,x0,del;

**printf**("x0 = 1");

*//scanf("%f",&x0);*

    x0 = 1;

**printf**("\ndel = 0.000000001");

*//scanf("%f",&del);*

    del = 0.000000001;

**printf**("\nN = 100");

*//scanf("%d",&N);*

    N = 100;

**printf**("\n k     x(k)");

**printf**("\n  %2d  %f",0,x0);

    while (n<N)

    {

        float f\_1 = **f**(x0);

        float f\_2 = **f1**(x0);

        float d = f\_1/f\_2;

        x =x0 - d;

        if(**fabs**(x-x0)<del)

        {

**printf**("\n\n Newton迭代法的近似解 = %.9lf \n",x);

            return;

        }

**printf**("\n %2d   %.9lf",n,x);

        n = n+1;

        x0 = x;

    }

**printf**("\n\n%d次迭代后未达到精度要求.\n",N);

}

int **main**()

{

*//显示中文*

**SetConsoleOutputCP**(65001);

**test01**();

    cout**<<endl**;

**test02**();

    cout**<<endl**;

**Aitken01**();

    cout**<<endl**;

**Aitken02**();

    cout**<<endl**;

**Newton**();

    cout**<<endl**;

**system**("pause");

    return 0;

}

结果图示：

