**实验八**

**一、实验要求**

用数值积分方法近似计算



及圆周率



(a). 用复合Simpson求积公式计算，要求绝对误差限小于，试根据积分余项估计步长ℎ的取值范围.按要求选择一个步长进行计算，观察数值结果与误差要求是否相符. （提示：可以利用Matlab的符号运算工具箱求函数的高阶导数表达式，详见命令diff, syms, ezplot的帮助文档）

(b). 用Romberg外推方法求积分近似值(误差要求与(a)同).

(c). 用下面的复合Gauss 公式计算近似积分：

其中ℎ=(𝑏−𝑎)/𝑛,𝑥𝑖+12=𝑥𝑖+ℎ2. 复合Gauss 积分的思想是将[𝑎,𝑏]作等距划分：𝑥𝑖=𝑎+𝑖ℎ(𝑖=0,…,𝑛), 然后在每个子区间内应用两点Gauss公式.试对步长ℎ作先验估计，（误差要求与(a)同），然后计算近似积分.

1. **解题思路**

1.Simpson积分。

Set Sum=f(a)+f(b)+4\*f(a+h/2);//a,b为区间两端点

For k=1,2……,n-1

Sum+=4\*f(a+k\*h+h/2)+2\*f(a+k\*h);

OUTPUT Sum\*h/6

2.Romberg 积分

用两个double数组t1,t2分别存储第k和第k+1次计算得到的T值。a,b分别为区间两端点，h为步长。每次由上一层计算得到下一层，然后比较两层最后两个元素差的绝对值看是否满足要求，不满足则令t1=t2，再由t1继续计算下一层得到t2，如此反复，知道满足要求。伪代码如下：

Step1 Set t1[0]= (f(a)+f(b))\*h/2;

Step2 Set t2[0]=t1[0]/2+f(a,b,func,1),

t2[1]=4/3\*t2[0]-1.0/3\*t1[0];

Step3 set i=2,j=0

Step4 While (|t2[i-1]-t1[i-2]|> )

Step5 for j=1,2….,i-1

T2[j]=f2(t1[j-1],t2[j-1])//外推

Step6 t1=t2//更新t1

Step7 t2[0]=f3(t1[0])//由Tn计算T2n

Step8 i=i+1

Output t2[i-1]//得到结果

3.Gauss

Set Sum=0;//a,b为区间两端点

For k=0,2……,n-1

Sum+= + ;

OUTPUT Sum\*h/2

**三、预习计算：**

1.Simpson积分时：

(1).对于计算ln2时的积分函数











对任意，积分误差

，解得



(2).对于计算pi时的积分函数













可知在[0,1]上先减小后增加，可计算得在x=0处，绝对值最大，为24，可知同1，

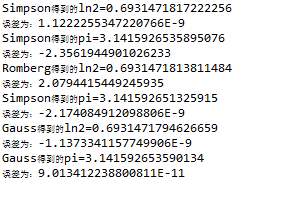
2.Gauss积分时:

对于积分函数和积分函数，都有，在对应的区间。故积分误差，解得

**四、算法设计**

|  |
| --- |
| **public** **class** Exp7  {  //f(x)=1/x  **public** **static** **double** fx1(**double** x)  {  **return** 1.0/x;  }  //f(x)=1/(x\*x)  **public** **static** **double** fx2(**double** x)  {  **return** 1.0/(1+x\*x);  }    //Romberg公式计算积分时的辅助函数  **public** **static** **double** fx3\_1(**double** a,**double** b,**int** n)  {  **int** i;  **double** h=(b-a)/(**double**)(n);  **double** sum=0.0;  **for**(i=0;i<n;i++)  {  sum+=*fx1*(a+i\*h+h/2);  }  **return** sum\*h/2;  }    **public** **static** **double** fx3\_2(**double** a,**double** b,**int** n)  {  **int** i;  **double** h=(b-a)/(**double**)(n);  **double** sum=0.0;  **for**(i=0;i<n;i++)  {  sum+=*fx2*(a+i\*h+h/2);  }  **return** sum\*h/2;  }    **public** **static** **double** Simpson1(**double** a , **double** b , **int** n)  {  **double** h = (b-a)/(**double**)(n);  **double** sim = *fx1*(a)+*fx1*(b)+4\**fx1*(a+h/2);  **for** (**int** i=1;i<n;i++)  {  sim+=4\**fx1*(a+i\*h+h/2)+2\**fx1*(a+i\*h);  }  **return** sim\*h/6;  }    **public** **static** **double** Simpson2(**double** a , **double** b , **int** n)  {  **double** h = (b-a)/(**double**)(n);  **double** sim = *fx2*(a)+*fx2*(b)+4\**fx2*(a+h/2);  **for** (**int** i=1;i<n;i++)  {  sim+=4\**fx2*(a+i\*h+h/2)+2\**fx2*(a+i\*h);  }  **return** sim\*h/6;  }    //Romberg积分方法  **public** **static** **double** Romberg1(**double** a,**double** b)  {  **double** h=b-a;  **double** t1[]=**new** **double**[1000];  **double** t2[]=**new** **double**[1000];  **int** i=2,j=1;  t1[0]=(*fx1*(a)+*fx1*(b))\*h/2;  t2[0]=t1[0]/2+*fx3\_1*(a,b,1);  t2[1]=Math.*pow*(4.0,1)/(Math.*pow*(4.0,1)-1)\*t2[0]-1.0/(Math.*pow*(4.0,1)-1)\*t1[0];  **while**(Math.*abs*(t2[i-1]-t1[i-2])>Math.*pow*(10.0,-9))  {  **for** (j=0;j<i;j++)  t1[j]=t2[j];  t2[0]=t1[0]/2+*fx3\_1*(a,b,(**int**)Math.*pow*(2.0,i));  **for** (j=1;j<=i+1;j++)  {  t2[j]=Math.*pow*(4.0,j)/(Math.*pow*(4.0,j)-1)\*t2[j-1]-1.0/(Math.*pow*(4.0,j)-1)\*t1[j-1];  }  i++;  }  **return** t2[i-1];  }    **public** **static** **double** Romberg2(**double** a,**double** b)  {  **double** h=b-a;  **double** t1[]=**new** **double**[1000];  **double** t2[]=**new** **double**[1000];  **int** i=2,j=1;  t1[0]=(*fx2*(a)+*fx2*(b))\*h/2;  t2[0]=t1[0]/2+*fx3\_2*(a,b,1);  t2[1]=Math.*pow*(4.0,1)/(Math.*pow*(4.0,1)-1)\*t2[0]-1.0/(Math.*pow*(4.0,1)-1)\*t1[0];  **while**(Math.*abs*(t2[i-1]-t1[i-2])>Math.*pow*(10.0,-9))  {  **for** (j=0;j<i;j++)  t1[j]=t2[j];  t2[0]=t1[0]/2+*fx3\_2*(a,b,(**int**)Math.*pow*(2.0,i));  **for** (j=1;j<=i+1;j++)  {  t2[j]=Math.*pow*(4.0,j)/(Math.*pow*(4.0,j)-1)\*t2[j-1]-1.0/(Math.*pow*(4.0,j)-1)\*t1[j-1];  }  i++;  }  **return** t2[i-1];  }    **public** **static** **double** Gauss1(**double** a,**double** b,**int** n)  {  **double** h=(b-a)/(**double**)(n);  **double** gauss=0.0;  **int** i;  **for** (i=0;i<n;i++)  gauss+=*fx1*(a+i\*h+h/2-h/2/Math.*pow*(3.0,0.5))+*fx1*(a+i\*h+h/2+h/2/Math.*pow*(3.0,0.5));  **return** gauss\*h/2;  }    **public** **static** **double** Gauss2(**double** a,**double** b,**int** n)  {  **double** h=(b-a)/(**double**)(n);  **double** gauss=0.0;  **int** i;  **for** (i=0;i<n;i++)  gauss+=*fx2*(a+i\*h+h/2-h/2/Math.*pow*(3.0,0.5))+*fx2*(a+i\*h+h/2+h/2/Math.*pow*(3.0,0.5));  **return** gauss\*h/2;  }    **public** **static** **void** main(String[] args)  {  **double** ln2=0.6931471806,pi=3.1415926535;  System.*out*.println("Simpson得到的ln2="+*Simpson1*(1 , 2 , 36));  System.*out*.println("误差为："+(*Simpson1*(1,2,36)-ln2));  System.*out*.println("Simpson得到的pi="+(4\**Simpson2*(0 , 1 , 36)));  System.*out*.println("误差为："+(*Simpson2*(0,1,36)-pi));  System.*out*.println("Romberg得到的ln2="+*Romberg1*(1 , 2));  System.*out*.println("误差为："+(4\**Romberg1*(1,2)-ln2));  System.*out*.println("Simpson得到的pi="+(4\**Romberg2*(0 , 1)));  System.*out*.println("误差为："+(4\**Romberg2*(0,1)-pi));  System.*out*.println("Gauss得到的ln2="+*Gauss1*(1 , 2 , 33));  System.*out*.println("误差为："+(*Gauss1*(1,2 , 33)-ln2));  System.*out*.println("Gauss得到的pi="+(4\**Gauss2*(0 , 1 , 33)));  System.*out*.println("误差为："+(4\**Gauss2*(0,1 ,33)-pi));    }  } |

**五、实验结果和结论**



实际值取保留十位小数：ln2=0.6931471806，pi=3.1415926535。

由结果可知，预习计算所得步长正确，满足精度要求。

数值积分时区间划分越精细，计算越精确，但是耗费计算量越大。故在满足精度要求的情况下，选择合适的步长h，可以减小计算量。