《计算方法》实验报告

**目录（按住ctrl鼠标左键即可跳转）**

[《计算方法》实验报告 1](#_Toc54727825)

[实验一 插值与拟合 1](#_Toc54727826)

[实验目的： 1](#_Toc54727827)

[实验内容及要求 2](#_Toc54727828)

[实验原理及算法描述 2](#_Toc54727829)

[程序代码及实验结果 2](#_Toc54727830)

[实验总结 9](#_Toc54727831)

[实验二 数值积分 9](#_Toc54727832)

[实验目的： 9](#_Toc54727833)

[实验内容及要求 10](#_Toc54727834)

[实验原理及算法描述 11](#_Toc54727835)

[程序代码及实验结果 12](#_Toc54727836)

[实验总结 17](#_Toc54727837)

[实验三 数值微分 18](#_Toc54727838)

[实验目的： 18](#_Toc54727839)

[实验内容及要求： 18](#_Toc54727840)

[实验原理及算法描述： 18](#_Toc54727841)

[程序代码及实验结果 20](#_Toc54727842)

[实验总结 22](#_Toc54727843)

[实验四 非线性方程求根迭代法 23](#_Toc54727844)

[实验目的： 23](#_Toc54727845)

[实验内容及要求： 23](#_Toc54727846)

[实验原理及算法描述： 23](#_Toc54727847)

[实验总结 27](#_Toc54727848)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学号 | |  | 姓名 |  | 班级 |  | |
| 实验项目名称 | | 求利用插值算法函数值 | | | | | |
| 一、实验名称 实验一 插值与拟合 | | | | | | | |
| 实验目的：  (1) 明确插值多项式和分段插值多项式各自的优缺点；  (2) 编程实现拉格朗日插值算法，分析实验结果体会高次插值产生的龙格现象；  (3) 理解最小二乘拟合，并编程实现线性拟合，掌握非线性拟合转化为线性拟合的方法  (4) 运用常用的插值和拟合方法解决实际问题。 | | | | | | | |
| 实验内容及要求  (1) 对于f(x)=1/(1+x2),-5<=x<=5  要求选取 11 个等距插值节点，分别采用拉格朗日插值和分段线性插值，计算 x 为 0.5, 4.5  处的函数值并将结果与精确值进行比较。  输入：区间长度，n(即 n+1 个节点)，预测点  输出：预测点的近似函数值，精确值，及误差  （2）已知 =1，= 2，= 3，用牛顿插值公式求的近似值。  输入：数据点集，预测点。  输出：预测点的近似函数值 | | | | | | | |
| 实验原理及算法描述   1. **Lagrange插值法的基本原理：**   拉格朗日插值公式：   1. **Lagrange插值算法描述：(也可以是算法流程图)**   **步骤1：** 构造处的插值基函数，其中，插值节点处的插值基函数为**；**  **步骤2：**以作为的系数，使得通过插值点；  **步骤3：把所有的**线性叠加，得到通过所有插值点的插值函数。  **Lagrange插值伪代码：**  **给定个插值点的情况下，求插值函数在点处的函数值。**  /\*输入参数  \***x**=(x0,x1,….,xn), 插值节点  \***y**=(y0,y1,…,yn); 被插函数f(x)在插值节点处的函数值  \***t**  求插值函数Ln (x)在t处的函数值  \*返回值 插值函数Ln (x)在t处的函数值  \*/ | | | | | | | |
| 程序代码及实验结果   1. **主程序**   int main(){  int n;  cout<<"请输入插入结点个数 ";  cin>>n;  vector<double>xi(n);  vector<double>yi(n);  cout<<"请输入xi以及对应的yi "<<endl;  for(int i=0;i<n;i++){  cout<<"第"<<i+1<<"个 ";  cin>>xi[i]>>yi[i];  }  int nn;  cout<<"请输入求解个数nn";  cin>>nn;  vector<double>xx(nn);  vector<double>result(nn);  vector<double>result\_p(nn);  cout<<"输入变量值";  for(int j=0;j<nn;j++){  cout<<"第"<<j+1<<"个 ";  cin>>xx[j];  }  cout<<endl;  lagrange(n,xi,yi,nn,xx,result);  partinter(n,xi,yi,nn,xx,result\_p);  vector<double>yy(nn);  yy[0]=0.8;  yy[1]=0.047059;    cout<<setw(12)<<"X"<<setw(12)<<"y(精确）"<<setw(12)<<"y(拉格朗日)"<<setw(12)<<"y(分段线性)"<<setw(12)<<"误差(拉)"<<setw(12)<<"误差(分)"<<endl;  for(int k=0;k<nn;k++){  cout<<setiosflags(ios::fixed)<<setprecision(6)<<setw(12)<<xx[k]<<setw(12)<<yy[k]<<setw(12)<<result[k]<<setw(12)<<result\_p[k]<<setw(12)<<yy[k]-result[k]<<setw(12)<<yy[k]-result\_p[k]<<endl;  }    cout<<endl<<"下面是牛顿插值求根号5";  // 下面是牛顿  int new\_n;  cout<<"输入插入解点个数 ";  cin>>new\_n;  vector<double>new\_xi(new\_n);  vector<double>new\_yi(new\_n);  cout<<"输入xi以及对应的yi"<<endl;  for(int i=0;i<new\_n;i++){  cout<<"第"<<i+1<<"个 ";  cin>>new\_xi[i]>>new\_yi[i];  }  double new\_number;  cout<<"请输入变量x的值 ";  cin>>new\_number;  cout<<"牛顿插值所求结果为="<<newton(new\_n,new\_xi,new\_yi,new\_number);  return 0;  }   1. **Lagrange插值子程序:**   void lagrange(int n,vector<double>&xi,vector<double>&yi,int nn,vector<double>&xx,vector<double>&result){  double temp\_result=0;  for(int i=0;i<nn;i++){    for(int j=0;j<n;j++){  double temp\_l=yi[j];  for(int k=0;k<n;k++){  if(k!=j){  temp\_l=temp\_l\*(xx[i]-xi[k]);  temp\_l=temp\_l/(xi[j]-xi[k]);  }  }  temp\_result+=temp\_l;  }  result[i]=temp\_result;  temp\_result=0;  }  }   1. **分段插值子程序**   void partinter(int n,vector<double>&xi,vector<double>&yi,int nn,vector<double>&xx,vector<double>&result\_p){  double temp\_result;  for(int i=0;i<nn;i++){  int temp=0;  for(int j=0;j<n;j++){  if(xi[j]>xx[i]){  temp=j;  break;  }  }  temp\_result=yi[temp-1]\*(1-(xx[i]-xi[temp-1])/(xi[temp]-xi[temp-1]))+yi[temp]\*(xx[i]-xi[temp-1])/(xi[temp]-xi[temp-1]);  result\_p[i]=temp\_result;  }  }   1. **牛顿插值子程序**   double newton(int n,vector<double>&xi,vector<double>&yi,double number){  vector<vector<double> >diff(n);  for(int i=0;i<n;i++){  diff[i].resize(n);  }  for(int j=0;j<n;j++) diff[0][j]=yi[j];  for(int k=1;k<n;k++){  for(int i=k;i<n;i++){  diff[k][i]=(diff[k-1][i]-diff[k-1][i-1])/(xi[i]-xi[i-1]);  }  }    double result\_n=0;  for(int i=0;i<=n-1;i++){  double temp=diff[i][i];  for(int j=0;j<i;j++){  temp=temp\*(number-xi[j]);  }  result\_n+=temp;  }    return result\_n;  }  **完整源代码**：  #include<iostream>  #include<iomanip>  #include<string>  #include<vector>  using namespace std;  void lagrange(int n,vector<double>&xi,vector<double>&yi,int nn,vector<double>&xx,vector<double>&result); //变量解释 n代表插值点数量 xi yi分别为x y 对应值 nn求解个数 xx为存储求解x的数组 result为拉格朗日插值结果  void partinter(int n,vector<double>&xi,vector<double>&yi,int nn,vector<double>&xx,vector<double>&result\_p); // result\_p为分段插值结果  double newton(int n,vector<double>&xi,vector<double>&yi,double number);  int main(){  int n;  cout<<"请输入插入结点个数 ";  cin>>n;  vector<double>xi(n);  vector<double>yi(n);  cout<<"请输入xi以及对应的yi "<<endl;  for(int i=0;i<n;i++){  cout<<"第"<<i+1<<"个 ";  cin>>xi[i]>>yi[i];  }  int nn;  cout<<"请输入求解个数nn";  cin>>nn;  vector<double>xx(nn);  vector<double>result(nn);  vector<double>result\_p(nn);  cout<<"输入变量值";  for(int j=0;j<nn;j++){  cout<<"第"<<j+1<<"个 ";  cin>>xx[j];  }  cout<<endl;  lagrange(n,xi,yi,nn,xx,result);  partinter(n,xi,yi,nn,xx,result\_p);  vector<double>yy(nn);  yy[0]=0.8;  yy[1]=0.047059;    cout<<setw(12)<<"X"<<setw(12)<<"y(精确）"<<setw(12)<<"y(拉格朗日)"<<setw(12)<<"y(分段线性)"<<setw(12)<<"误差(拉)"<<setw(12)<<"误差(分)"<<endl;  for(int k=0;k<nn;k++){  cout<<setiosflags(ios::fixed)<<setprecision(6)<<setw(12)<<xx[k]<<setw(12)<<yy[k]<<setw(12)<<result[k]<<setw(12)<<result\_p[k]<<setw(12)<<yy[k]-result[k]<<setw(12)<<yy[k]-result\_p[k]<<endl;  }    cout<<endl<<"下面是牛顿插值求根号5";  // 下面是牛顿  int new\_n;  cout<<"输入插入解点个数 ";  cin>>new\_n;  vector<double>new\_xi(new\_n);  vector<double>new\_yi(new\_n);  cout<<"输入xi以及对应的yi"<<endl;  for(int i=0;i<new\_n;i++){  cout<<"第"<<i+1<<"个 ";  cin>>new\_xi[i]>>new\_yi[i];  }  double new\_number;  cout<<"请输入变量x的值 ";  cin>>new\_number;  cout<<"牛顿插值所求结果为="<<newton(new\_n,new\_xi,new\_yi,new\_number);  return 0;  }  void lagrange(int n,vector<double>&xi,vector<double>&yi,int nn,vector<double>&xx,vector<double>&result){  double temp\_result=0;  for(int i=0;i<nn;i++){    for(int j=0;j<n;j++){  double temp\_l=yi[j];  for(int k=0;k<n;k++){  if(k!=j){  temp\_l=temp\_l\*(xx[i]-xi[k]);  temp\_l=temp\_l/(xi[j]-xi[k]);  }  }  temp\_result+=temp\_l;  }  result[i]=temp\_result;  temp\_result=0;  }  }  void partinter(int n,vector<double>&xi,vector<double>&yi,int nn,vector<double>&xx,vector<double>&result\_p){  double temp\_result;  for(int i=0;i<nn;i++){  int temp=0;  for(int j=0;j<n;j++){  if(xi[j]>xx[i]){  temp=j;  break;  }  }  temp\_result=yi[temp-1]\*(1-(xx[i]-xi[temp-1])/(xi[temp]-xi[temp-1]))+yi[temp]\*(xx[i]-xi[temp-1])/(xi[temp]-xi[temp-1]);  result\_p[i]=temp\_result;  }  }  double newton(int n,vector<double>&xi,vector<double>&yi,double number){  vector<vector<double> >diff(n);  for(int i=0;i<n;i++){  diff[i].resize(n);  }  for(int j=0;j<n;j++) diff[0][j]=yi[j];  for(int k=1;k<n;k++){  for(int i=k;i<n;i++){  diff[k][i]=(diff[k-1][i]-diff[k-1][i-1])/(xi[i]-xi[i-1]);  }  }    double result\_n=0;  for(int i=0;i<=n-1;i++){  double temp=diff[i][i];  for(int j=0;j<i;j++){  temp=temp\*(number-xi[j]);  }  result\_n+=temp;  }    return result\_n;  }  **实验截图** | | | | | | | |
| 实验总结 | | | | | | | |
| 五、教师评语（或成绩）  教师签字 ： | | | | | | | |
| 实验项目名称 | 数值积分 | | | | | | | |
| 一、实验名称 实验二 数值积分 | | | | | | | | |
| 实验目的：   * 1. 熟悉复化梯形方法、复化 Simpson 方法、梯形递推算法、龙贝格算法；   2. 能编程实现复化梯形方法、复化 Simpson 方法、梯形递推算法、龙贝格算法；   3. 理解并掌握自适应算法和收敛加速算法的基本思想；   4. 分析实验结果体会各种方法的精确度，建立计算机求解定积分问题的感性认识 | | | | | | | | |
| 实验内容及要求 | | | | | | | | |
| 实验原理及算法描述   1. **基本原理：** | | | | | | | | |
| 程序代码及实验结果   1. **主程序**   int main(){  double a,b;  cout<<"输入a，b"<<endl;;  cin>>a>>b;  double xxx=romberg(a,b);        cout<<endl;  //第二问  cout<<"下面是求瑞士面积";  int n=27;  double x[27]={7,10.5,13,17.5,34,40.5,44.5,48,56,61,68.5,76.5,80.5,91,96,101,104,106.5,111.5,118,123.5,136.5,142,146,150,157,158};  double y1[27]={44,45,47,50,50,38,30,30,34,36,34,41,45,46,43,37,33,28,32,65,55,54,52,50,66,66,68};  double y2[27]={44,59,70,72,93,100,110,110,110,117,118,116,118,118,121,124,121,121,121,122,116,83,81,82,86,85,68};  cout<<"瑞士国土面积估算为"<<switzerland(x,y1,y2,n)\*4.9<<"准确值为41288"<<endl;  return 0;  }   1. **Romberg插值子程序:**   double romberg(double a,double b){  double h=b-a;  vector<double>T(10);  vector<double>S(10);  vector<double>C(10);  vector<double>R(10);  T[1]=h/2\*(f(a)+f(b));  double temp=0;  double k=1;      for(k=1;k<=5;k++){  temp=ladder(h,T[k],a,b);  T[k+1]=temp;  S[k+1]=(T[k+1]\*4-T[k])/3;  if(k==1){  h=h/2;  continue;  }    C[k+1]=(S[k+1]\*16-S[k])/15;    if(k==2){  h=h/2;  continue;  }    R[k+1]=(C[k+1]\*64-C[k])/63;    if(k==3){  h=h/2;  continue;  }    }    cout<<setw(10)<<'k'<<setw(10)<<'T'<<setw(10)<<'S'<<setw(10)<<'C'<<setw(10)<<'R'<<endl;  for(int i=1;i<=5;i++){  cout<<setw(10)<<i-1<<setw(10)<<T[i];  if(i>1){  cout<<setw(10)<<S[i];  }  if(i>2){  cout<<setw(10)<<C[i];  }  if(i>3){  cout<<setw(10)<<R[i];  }  cout<<endl;  }  cout<<" the 4 result is:"<<R[5]<<endl;  return R[2];  }  **完整源代码（c++）**  #include<iostream>  #include<iomanip>  #include<vector>  using namespace std;  double f(double x){  if(x==0) return 1.0;  else if(x==0.125) return 0.9973978;  else if(x==0.25) return 0.9896158;  else if(x==0.375) return 0.9767267;  else if(x==0.5) return 0.9588510;  else if(x==0.625) return 0.9361556;  else if(x==0.75) return 0.9088516;  else if(x==0.875) return 0.8771925;  else if(x==1) return 0.8414709;  else if(x==0.0625) return 0.9993491;  else if(x==0.1875) return 0.9941509;  else if(x==0.3125) return 0.9838032;  else if(x==0.4375) return 0.9684029;  else if(x==0.5625) return 0.9480936;  else if(x==0.6875) return 0.9230648;  else if(x==0.8125) return 0.8935491;  else if(x==0.9375) return 0.8598198;  else return 0;  }  double ladder(double h,double T,double a,double b){  double s=0,x=a+h/2;    do{  s+=f(x);  x+=h;  }while(x<=b);    double T2=T/2+h/2\*s;  return T2;    }  double romberg(double a,double b){  double h=b-a;  vector<double>T(10);  vector<double>S(10);  vector<double>C(10);  vector<double>R(10);  T[1]=h/2\*(f(a)+f(b));  double temp=0;  double k=1;      for(k=1;k<=5;k++){  temp=ladder(h,T[k],a,b);  T[k+1]=temp;  S[k+1]=(T[k+1]\*4-T[k])/3;  if(k==1){  h=h/2;  continue;  }    C[k+1]=(S[k+1]\*16-S[k])/15;    if(k==2){  h=h/2;  continue;  }    R[k+1]=(C[k+1]\*64-C[k])/63;    if(k==3){  h=h/2;  continue;  }    }    cout<<setw(10)<<'k'<<setw(10)<<'T'<<setw(10)<<'S'<<setw(10)<<'C'<<setw(10)<<'R'<<endl;  for(int i=1;i<=5;i++){  cout<<setw(10)<<i-1<<setw(10)<<T[i];  if(i>1){  cout<<setw(10)<<S[i];  }  if(i>2){  cout<<setw(10)<<C[i];  }  if(i>3){  cout<<setw(10)<<R[i];  }  cout<<endl;  }  cout<<" the 4 result is:"<<R[5]<<endl;  return R[2];  }  double switzerland(double x[],double y1[],double y2[],int n){  double sum=0,temp1,temp2;  for(int i=1;i<n;i++){  temp1=(x[i]-x[i-1])\*(y1[i]+y1[i-1])/2;  temp2=(x[i]-x[i-1])\*(y2[i]+y2[i-1])/2;  sum=sum+temp2-temp1;  }  return sum;  }  int main(){  double a,b;  cout<<"输入a，b"<<endl;;  cin>>a>>b;  double xxx=romberg(a,b);        cout<<endl;  //第二问  cout<<"下面是求瑞士面积";  int n=27;  double x[27]={7,10.5,13,17.5,34,40.5,44.5,48,56,61,68.5,76.5,80.5,91,96,101,104,106.5,111.5,118,123.5,136.5,142,146,150,157,158};  double y1[27]={44,45,47,50,50,38,30,30,34,36,34,41,45,46,43,37,33,28,32,65,55,54,52,50,66,66,68};  double y2[27]={44,59,70,72,93,100,110,110,110,117,118,116,118,118,121,124,121,121,121,122,116,83,81,82,86,85,68};  cout<<"瑞士国土面积估算为"<<switzerland(x,y1,y2,n)\*4.9<<"准确值为41288"<<endl;  return 0;  }  **实验运行截图：** | | | | | | | | |
| 实验总结 | | | | | | | | |
| 五、教师评语（或成绩）  教师签字 ： | | | | | | | | |
|  | | | | | | | | |
| 实验项目名称 | | 利用欧拉算法等求出数值解 | | | | | |
| 一、实验名称： 实验三 数值微分 | | | | | | | |
| 实验目的：   * 1. 熟悉数值微分中Euler法，改进Euler法，Rung-Kutta方法；   2. 能编程实现Euler法，改进Euler法，Rung-Kutta方法；   3. 通过实验结果分析各个算法的优缺点;   4. 明确步长对算法的影响并理解变步长的Rung-Kutta方法 | | | | | | | |
| 实验内容及要求：   1. 0 < x<1   取h=0.1时用Euler法，改进Euler法，Rung-Kutta方法求其数值解并与精确解进行比较。  输入：求解区间，初值，数值解个数  输出：数值解 | | | | | | | |
| 实验原理及算法描述：   1. **原理：**   在许多科学技术问题中，建立的模型常常以常微分方程的形式表示。然而，除了少数特殊类型的常微分方程能用解析方法求其精确解外，要给出一般方程解析解的表达式是困难的。所以只能用近似方法求其数值解，在实际工作中常用计算机求常微分方程的数值解。所谓常微分方程的数值解即对于常微分方程初值问题计算出在一系列节点 a = x0< x1<…< xn= b 处的未知函数 y(x)近似值y0,y1,…yn，即找到一系列离散点（x0,y0）（x1,y1）（x2,y2）…（xn,yn）近似满足常微分方程。数值解法的基本思想用差商代替导数,实现连续问题离散化，选取不同的差商代替导数可以得到不同公式。  改进欧拉公式是常用方法之一，包括预测和校正两步。先用欧拉公式进行预报，再将预报值代入梯形公式进行校正，从而达到二阶精度。通过龙格-库塔法我们可以获得更高精度。经典龙格-库塔法即在区间[xn,xn+1]取四点，并对这四点的斜率进行加权平均作为平均斜率，通过泰勒公式寻找使局部截断误差为O(h5)（即4阶精度）的参数满足条件。  改进的欧拉公式： 预测  校正：  四阶（经典）龙格-库塔公式：     1. **描述：** 2. 改进的欧拉算法见流程图 3. 经典龙格-库塔算法见流程图   图5.1Euler法，改进Euler法，R-K方法参考输出 | | | | | | | |
| 程序代码及实验结果   1. **主程序**   int main() {  double a, b;  int n;  n = 10;  vector<double>eu(n + 1);  vector<double>im\_eu(n + 1);  vector<double>run(n + 1);  a = 0, b = 1;  eu[0] = 1;  im\_eu[0] = 1;  run[0] = 1;  double aim[12] = { 1.095445,1.183216,1.264911,1.341641,1.414214,1.483240,1.549193,1.612452,1.673320,1.732051 };  euler(a, b, n, eu);  improve\_euler(a, b, n, im\_eu);  runge(a, b, n, run);  cout << setw(12) << "x" << setw(12) << "y(4阶龙格)" << setw(12) << "y(改进)" << setw(12) << "y(欧拉)" << setw(12) << "y(精确)" << setw(12) << endl;  for (int i = 0; i < n; i++) {  cout << setiosflags(ios::fixed) << setprecision(6) << setw(12) << (i + 1) \* 0.1 << setw(12) << run[i + 1] << setw(12) << im\_eu[i + 1] << setw(12) << eu[i + 1] << setw(12) << aim[i] << endl;  }  return 0;  }   1. **完整源代码:**   #include<iostream>  #include<iomanip>  #include<vector>  using namespace std;  double f(double x, double y) {  double temp;  temp = y - (x \* 2) / y;  return temp;  }  void euler(double a, double b, int n, vector<double>& eu) {  double x = (double)n;  double h = (b - a) / x;  for (int i = 0; i < n; i++) {  eu[i + 1] = eu[i] + h \* f(a + h \* (i + 1), eu[i]);  }  }  void improve\_euler(double a, double b, int n, vector<double>& im\_eu) {  double x = (double)n;  double h = (b - a) / x;  double yp, yc;  for (int i = 1; i <= n; i++) {  yp = im\_eu[i - 1] + h \* f(a + h \* (i - 1), im\_eu[i - 1]);  yc = im\_eu[i - 1] + h \* f(a + h \* i, yp);  im\_eu[i] = (yc + yp) / 2;  }  }  void runge(double a, double b, int n, vector<double>& run) {  double x = (double)n;  double h = (b - a) / x;  double k1, k2, k3, k4;  for (int i = 0; i < n; i++) {  k1 = f(a + h \* i, run[i]);  k2 = f(a + h \* i + h / 2, run[i] + h \* k1 / 2);  k3 = f(a + h \* i + h / 2, run[i] + h \* k2 / 2);  k4 = f(a + h \* (i + 1), run[i] + h \* k3);  run[i + 1] = run[i] + h \* (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;  }  }  int main() {  double a, b;  int n;  n = 10;  vector<double>eu(n + 1);  vector<double>im\_eu(n + 1);  vector<double>run(n + 1);  a = 0, b = 1;  eu[0] = 1;  im\_eu[0] = 1;  run[0] = 1;  double aim[12] = { 1.095445,1.183216,1.264911,1.341641,1.414214,1.483240,1.549193,1.612452,1.673320,1.732051 };  euler(a, b, n, eu);  improve\_euler(a, b, n, im\_eu);  runge(a, b, n, run);  cout << setw(12) << "x" << setw(12) << "y(4阶龙格)" << setw(12) << "y(改进)" << setw(12) << "y(欧拉)" << setw(12) << "y(精确)" << setw(12) << endl;  for (int i = 0; i < n; i++) {  cout << setiosflags(ios::fixed) << setprecision(6) << setw(12) << (i + 1) \* 0.1 << setw(12) << run[i + 1] << setw(12) << im\_eu[i + 1] << setw(12) << eu[i + 1] << setw(12) << aim[i] << endl;  }  return 0;  }  **实验截图** | | | | | | | |
| 实验总结 | | | | | | | |
| 五、教师评语（或成绩）  教师签字 ： | | | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 |  | 姓名 |  | 班级 |  |

|  |  |
| --- | --- |
| 实验项目名称 | 利用牛顿下山算法求解方程 |
| 一、实验名称： 实验四 非线性方程求根迭代法 | |
| 实验目的：   1. 熟悉非线性方程求根简单迭代法，牛顿迭代及牛顿下山法 2. 能编程实现简单迭代法，牛顿迭代及牛顿下山法 3. 认识选择迭代格式的重要性    1. 对迭代速度建立感性的认识；分析实验结果体会初值对迭代的影响 | |
| 实验内容及要求：   1. 用牛顿下山法解方程（初值为0.6）   输入：初值，误差限，迭代最大次数，下山最大次数  输出：近似根各步下山因子 | |
| 实验原理及算法描述：   1. **原理：**   求非线性方程组的解是科学计算常遇到的问题，有很多实际背景．各种算法层出不穷，其中迭代是主流算法。只有建立有效的迭代格式，迭代数列才可以收敛于所求的根。因此设计算法之前，对于一般迭代进行收敛性的判断是至关重要的。牛顿法也叫切线法，是迭代算法中典型方法，只要初值选取适当，在单根附近，牛顿法收敛速度很快，初值对于牛顿迭代至关重要。当初值选取不当可以采用牛顿下山算法进行纠正。  一般迭代：  牛顿公式：  牛顿下山公式：  下山因子  下山条件   1. **算法描述：**   一般迭代算法见流程图；  牛顿下山算法见流程图。  图3.1一般迭代算法流程图  图3.2牛顿下山算法流程图 实验1输出参考书本p130结果。 | |
| int newton(vector<double>& x, double e, int n, int m) {  int j;  double v;  for (int i = 0; i < n; i++) {  if (f1(x[i]) == 0) {  cout << "奇异标志";  return 0;  }  v = 1;  for (j = 0; j < m;) {  x[i + 1] = v \* (x[i] - f(x[i]) / f1(x[i])) + (1 - v) \* (f(x[i]) / f1(x[i]));  if (fabs(f(x[i + 1])) > fabs(f(x[i]))) {  j++;  v = v \* 0.5;  }  else {  break;  }  }  if (fabs(x[i + 1] - x[i]) < e) {  cout << "迭代结果为：" << x[i + 1];  return 0;  }  }  cout << "迭代失败";  return 0;  }  **源代码**：  #include<iostream>  #include<math.h>  #include<vector>  using namespace std;  double f(double x) {  return x \* x \* x - x - 1;  }  double f1(double x) {  return 3 \* x \* x - 1;  }  int newton(vector<double>& x, double e, int n, int m) {  int j;  double v;  for (int i = 0; i < n; i++) {  if (f1(x[i]) == 0) {  cout << "奇异标志";  return 0;  }  v = 1;  for (j = 0; j < m;) {  x[i + 1] = v \* (x[i] - f(x[i]) / f1(x[i])) + (1 - v) \* (f(x[i]) / f1(x[i]));  if (fabs(f(x[i + 1])) > fabs(f(x[i]))) {  j++;  v = v \* 0.5;  }  else {  break;  }  }  if (fabs(x[i + 1] - x[i]) < e) {  cout << "迭代结果为：" << x[i + 1];  return 0;  }  }  cout << "迭代失败";  return 0;  }  int main() {  cout << "实验4" << endl;  vector<double>x(10, 1);  x[0] = 0.6;  double e = 0.0001;  int n = 20, m = 20;  int a;  a = newton(x, e, n, m);  return 0;  }  **运行截图**： | |
| 实验总结 | |
| 五、教师评语（或成绩）  教师签字 ： | |