|  |
| --- |
| 东南大学 电子科学与工程学院 |
| **计算方法 第二次上机报告** |
| 06220143 顾豪阳 |

|  |
| --- |
| 2022-4-24 |

# 一、题目简介

**P111数值实验3.3** 请编写割线法的程序，并用它计算例16——

取初值为，利用割线法求方程的根-2（误差限取），并求算法的数值收敛阶。

# 二、理论分析

对于Newton法而言，需要知道函数的导数值，这对于很多函数来说求解并不容易，因为很有可能在迭代的过程中并不存在改点的导数值，因此可以使用割线法改善Newton法求导困难的情况。

对于Newton法来说，其迭代公式为：

且当结果为单根的时候，Newton法2阶局部收敛，并且是一种不动点迭代的方法，并没有记忆性；

而对于割线法，则使用两个点的割线来替代牛顿法中一个点的导数，这也避免了直接求解导数的困难。其通常写成：

不难看出，割线法并非不动点迭代法，而是一种有记忆性的迭代法。因此在上机过程中，需要另外定义两个变量存储上一次的数据。

# 三、数值实验过程

以书本P110例16为例：

取初值为，利用割线法求方程的根-2（误差限取）

**解：**通过化简可以看出方程的解为-2（单根）1（重根），将方程代入迭代公式中可得：

选取作为终止判据，得出最终的计算结果

# 四、程序代码与结果

使用C++编写计算程序，将割线法与牛顿法进行对比：

/\*以函数指针传入待解函数相比直接在函数里调用

可以同时实现几个函数名不同的不同函数\*/

#include<iostream>

#include<cmath>

#include <iomanip>

using namespace std;

double SecantMethod(double a,double b,double accuracy,double(\*fun)(double x));//函数指针 accuracy 精度

double Derivative(double (\*fun)(double x),double a,double accuracy=0.000001);//求解某点导数

double NewtonMethod(double a ,double accuracy,double (\*fun)(double x));//牛顿法求解

double f1(double x){

return x\*x\*x-3\*x+2;

}

int main(){

cout<<"用割线法求得的结果为："<<SecantMethod(-2.6,-2.4,1e-10,f1)<<endl;//终止判据

cout<<"用牛顿法求得的结果为："<<NewtonMethod(-2.6,1e-10,f1)<<endl;

system("pause");

return 0;

}

double SecantMethod(double a,double b,double accuracy,double(\*fun)(double x)){

cout<<"使用割线法进行迭代："<<endl;

const int max=30;

int j;

double x1,x2,temp,f1,f2,dx;

f1=fun(a);

f2=fun(b);

if(fabs(f1)<fabs(f2))

{

x1=a;

x2=b;

}

else

{

x1=b;

x2=a;

temp=f1;

f1=f2;

f2=temp;

}//f1是绝对值较小的那个

for(j=1;j<=max;j++)

{

dx=(x1-x2)\*f2/(f2-f1);

//cout<<dx;//每一步的结果输出

temp=x2;

x2+=dx;

x1=temp;

f1=f2;

f2=fun(x2);

//cout<<'\t'<<x2<<endl;

if(dx<0)dx=-dx;

cout<<"dx= "<<dx<<'\t';

cout<<"第"<<j<<"次迭代："<<fixed<<setprecision(12)<<x2<<'\n';//12位有效数字

if((dx<accuracy)||(f2==0))

{

return x2;

}

}

}

double NewtonMethod(double a ,double accuracy,double (\*fun)(double x)){

double x;double y;

x=y=a;

double dx=1;

int n=1;

cout<<"使用牛顿法进行迭代："<<endl;

while(fabs(dx)>accuracy){

dx=fun(x)/Derivative(fun,x);

cout<<"dx= "<<dx<<'\t';

y=x-dx;

x=y;

cout<<"第"<<n<<"次迭代："<<fixed<<setprecision(12)<<x<<endl;//12位有效数字

n++;

}

return x;

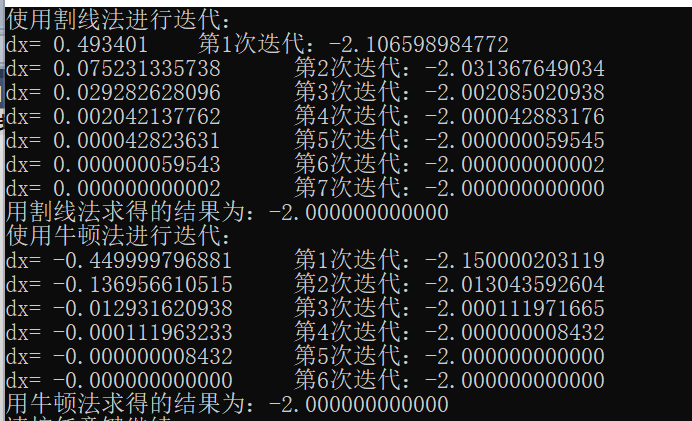
}

double Derivative(double (\*fun)(double x),double a,double accuracy){

return (fun(a)-fun(a-accuracy))/accuracy;

}

实验结果算得：



**使用割线法进行迭代：**

dx= 0.493401000000 第1次迭代：-2.106598984772

dx= 0.075231335738 第2次迭代：-2.031367649034

dx= 0.029282628096 第3次迭代：-2.002085020938

dx= 0.002042137762 第4次迭代：-2.000042883176

dx= 0.000042823631 第5次迭代：-2.000000059545

dx= 0.000000059543 第6次迭代：-2.000000000002

dx= 0.000000000002 第7次迭代：-2.000000000000

用割线法求得的结果为：-2.000000000000

**使用牛顿法进行迭代：**

dx= -0.449999796881 第1次迭代：-2.150000203119

dx= -0.136956610515 第2次迭代：-2.013043592604

dx= -0.012931620938 第3次迭代：-2.000111971665

dx= -0.000111963233 第4次迭代：-2.000000008432

dx= -0.000000008432 第5次迭代：-2.000000000000

dx= -0.000000000000 第6次迭代：-2.000000000000

用牛顿法求得的结果为：-2.000000000000

和最终结论吻合。

计算阶数：

lis={-2.6,-2.4,-2.106598984772,-2.031367649034,-2.002085020938,-2.000042883176,-2.000000059545,-2.000000000002,-2.};

err=Abs[lis+2];

data=Table[{Log[err[[i]]],Log[err[[i+1]]]},{i,5}];

**Fit[data,{1,t},t]**

# 五、对实验的分析

通过每次迭代得出的数据来看，运用割线法所得的误差数据的收敛性并没有牛顿法来得快，再通过和书上的数据比较有细微的差别，经过分析应该是浮点数精度的问题，但是可以忽略，收敛阶数也大约在1.6左右，和书上得出的结论相吻合。

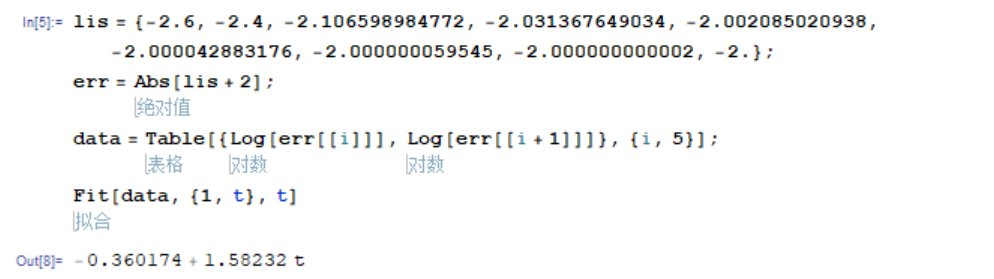


图1 计算割线法的收敛阶

而对于牛顿法来说，可以通过观察得到，收敛速度更快，但是因为C++中没有封装求导的函数库，自己写的求导函数精度不高，因此也可能有细微的差别。

# 六、结论和感想

对题目进行一定的拓展，我们刚刚求得x=1为方程的重根，那么我们来测试一下割线法和牛顿法对于重根的收敛速度



使用割线法进行迭代：

dx= 3.124320000000 第1次迭代：0.724324324324

dx= 0.133897083720 第2次迭代：0.590427240604

dx= 0.255751013798 第3次迭代：0.846178254402

…………

dx= 0.000001546900 第26次迭代：0.999997497049

dx= 0.000000956066 第27次迭代：0.999998453115

用割线法求得的结果为：0.999998453115

使用牛顿法进行迭代：

dx= -0.103703473213 第1次迭代：0.903703473213

dx= -0.048959862522 第2次迭代：0.952663335735

dx= -0.023859342948 第3次迭代：0.976522678683

…………

dx= -0.000001441592 第17次迭代：0.999998123839

dx= -0.000000740710 第18次迭代：0.999998864549

用牛顿法求得的结果为：0.999998864549

这里已经将精度调整为，因为重根的收敛速度比单根慢得多，这里也迭代了好几十次，可以看出在迭代效率方面还是牛顿法“略胜一筹”。

感想：牛顿法能以极快的收敛速度求得方程的解，但是在一些无法计算导数的特殊情况，割线法还是最合适的选择；并且，加上求导的时间，Newton法每次需要计算两个函数值，计算效率没有看起来那么高；总之，从求根总成本上来看，割线法比Newton法的效率更高。