非线性方程的数值计算实验

姓名：王旋宇

学号：2012049010022

班级：数理基科班

老师：赖生建

时间：2014年9月28日星期日

实验目的

1. 了解迭代法、不动点。掌握不动点迭代的适用条件。能使用不动点迭代编程计算。
2. 了解二分法、试值法，能使用二分法、试值法进行编程计算。能对二分法进行精度预估计算。
3. 了解牛顿-拉夫森法。掌握牛顿-拉夫森法的适用条件。明白其缺陷。能使用其进行编程计算，并能在有缺陷的地方看出来。
4. 了解收敛速度、收敛阶，了解牛顿-拉夫森法的收敛速度。能使用加速牛顿-拉夫森法进行编程计算。

实验原理

迭代法

迭代指重复执行一个计算过程，直到找到答案。迭代是计算机科学中的一个基本要素，用来求解方程的根、线性和非线性方程组的解。

如果要进行迭代首先要有一个用于逐项计算的规则或者函数$g(x)$，并且有一个起点p_0。然后通过利用迭代规则$p_{k+1}=g(p_k)$，可得到序列值$\{p_k\}$。此序列有如下模式：

$$\begin{gathered} p_0(\text{初始值}) \\ p_1=g(p_0) \\ p_2=g(p_1) \\ \vdots \\ p_k=g(p_{k-1}) \\ p_{k+1}=g(p_k) \\ \vdots \end{gathered}$$

容易看出此序列是一个无限序列，如果序列趋于一个极限，那么我们可能就达到了求解目的。

不动点

定义 1（不动点）：函数$g(x)$的一个不动点是指一个实数P，满足$P=g(P)$。从图形角度分析，函数$g(x)$的不动点是$y=g(x)$和$y=x$的交点。迭代$p_{n+1}=g(p_n)$，其中$n=0,1,\cdots$,称为不动点迭代。

定理 1：设g是一个连续函数，且$\{p_n\}_{n=0}^{\infty}$是由不动点迭代生成的序列。如果$\lim_{n\to\infty} p_n=P$，则P是$g(x)$的不动点。

定理 2：设函数$g\in C[a,b]$。

如果对于所有$x\in[a,b]$，映射$y=g(x)$的范围满足$y\in[a,b]$，则函数g在$[a,b]$内有一个不动点。

此外，设$g'(x)$定义在(a,b)内，且对于所有$x\in[a,b]$，存在正常数$K<1$，使得

$|g'(x)| \le K < 1$，则函数g在$[a,b]$内有唯一的不动点P。

定理 3（不动点定理）：设有(i)$g, g' \in C[a,b]$,(ii)K是一个正常数，(iii) $p_0 \in [a,b]$，(iv)对于所有$x \in [a,b]$，有$g(x) \in [a,b]$。

如果对于所有$x \in [a,b]$，有$|g'(x)| \le K < 1$，则迭代$p_n = g(p_{n-1})$将收敛到唯一的不动点$P \in [a,b]$。在这种情况下，P称为吸引不动点。

如果对于所有$x \in [a,b]$，有$|g'(x)| > 1$，则迭代$p_n = g(p_{n-1})$将不会收敛到P。在这种情况下，P称为排斥不动点，而且迭代显示出局部发散性。

注 1：定理中假设$P \neq p_0$

注 2：因为函数g在包含P的一段间隔中是连续的，所以定理 2 中可以用更简单的判别条件$|g'(P)| \le K < 1$和$|g'(P)| > 1$。

不动点迭代的终止判别：虽然连续项的相近不能保证精度要求，但是通常我们还是用连续项的差异比较作为迭代终止的惟一判别标准。

波尔查诺二分法

起始区间$[a,b]$满足$f(a)$与$f(b)$符号相反的条件时，若函数在区间$[a,b]$上为连续函数，那么由于连续函数的图形无间断，所以他会在零点r跨过x轴，且r在区间内。通过二分法可将区间内的端点逐步逼近零点，直到得到一个任意小的包含零点的间隔。

由于二分法的过程包括嵌套区间间间隔和它们的中点，所以采用如下表示方法：

1) $[a_0,b_0]$是起始区间，$c_0 = (a_0 + b_0)/2$是中点。
2) $[a_1,b_1]$是第二个区间，它包含零点r，同时c_1是中点，区间的宽度范围是$[a_0,b_0]$的一半。
3) 得到第n个区间$[a_n,b_n]$（包含r，并有中点c_n）后，可构造出$[a_{n+1},b_{n+1}]$，它也包括r，宽度范围是$[a_n,b_n]$的一半。

其中$c_n = (a_n + b_n)/2$，且如果$f(a_{n+1})f(b_{n+1}) < 0$，则对于所有的n，

$$[a_{n+1},b_{n+1}] = [a_n,c_n]\text{或}[a_{n+1},b_{n+1}] = [c_n,b_n] \tag{1}$$

终止判断时，可以在区间宽度小于误差容忍限度的时候终止。

定理 4（二分法定理）：设$f \in C[a,b]$，且存在数$r \in [a,b]$满足$f(r) = 0$。如果$f(a)$和$f(b)$的符号相反，且$\{c_n\}_{n=0}^{\infty}$表示式(1)中的二分法生成的中点序列，则：

$$|r - c_n| \le \frac{b-1}{2^{n+1}} \text{ 其中} n = 0,1,\cdots \tag{2}$$

这样，序列$\{c_n\}_{n=0}^{\infty}$收敛到零点$x = r$即可表示为

$$\lim_{n\to\infty} c_n = r \tag{3}$$

在定理四中给出了一个对计算结果精度的预先估计。假设第N个中点c_n是零点的近似值，且误差不小于δ，则有对于给定的初始区间$[a,b]$

$$N = int\left(\frac{\ln(b-a)-\ln(\delta)}{\ln(2)}\right) \tag{4}$$

试值法

又称试位法。由于二分法的收敛速度相对较慢，因为进行了改进。同样假设$f(a)$和$f(b)$符号相反。不过这里找到的c为：

$$c = b - \frac{f(b)(b-a)}{f(b)-f(a)} \tag{5}$$

按照和二分法中同样的区间构造方法，得到一个零点r的近似值序列：

$$c_n = b_n - \frac{f(b_n)(b_n-a_n)}{f(b_n)-f(a_n)} \tag{6}$$

可证明序列$\{c_n\}_{n=0}^{\infty}$将收敛到r。不能使用二分法的终止判别条件，要使用迭代的封闭性和$|f(c_n)|$的值同时用来作为程序的终止判别条件。

牛顿-拉夫森法

如果$f(x)$，$f'(x)$和$f''(x)$在根p附近连续，则可将它作为$f(x)$的特性，用于开发产生收敛到根p的序列$\{p_k\}$的算法。而且这种算法比二分法和试值法产生序列$\{p_k\}$的速度快。牛顿-拉夫森法依赖$f'(x)$和$f''(x)$的连续性，简称牛顿法，是此类方法中最有用和最好的方法之一。

定理 5（牛顿-拉夫森定理）：设$f \in C^2[a,b]$，且存在数$p \in [a,b]$，满足$f(p)=0$。如果$f'(p) \neq 0$，则存在一个数$\delta > 0$，对任意初始近似值$p_0 \in [p-\delta, p+\delta]$，使得由如下迭代定义的序列$\{p_n\}_{n=0}^{\infty}$收敛到$p$：

$$p_k = g(p_{k-1}) = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})} \quad \text{其中} k = 1,2,\cdots \tag{7}$$

注：函数 由如下公式定义：

$$g(x) = x - \frac{f(x)}{f'(x)} \tag{8}$$

并被称为牛顿-拉夫森迭代函数。终止判别使用连续项的差异作为判别条件。

定义 2：设$f(x)$和它的导数$f'(x)$在包含$x=p$的某区间内有定义且连续，则称$f(x)=0$在$x=p$处根的阶为M，当且仅当

$$f(p)=0, f'(p)=0, \cdots, f^{(M-1)}(p)=0, f^{(M)}(p) \neq 0 \tag{9}$$

如果一个根的阶$M=1$，则称此根为单根；如果一个根的阶$M>1$，则称此根为重根，并成为M重根。

定义 3（收敛阶）：设序列$\{p_n\}_{n=0}^{\infty}$收敛到p，并令$E_n = p - p_n, n \geq 0$， 。如果两个常量$A \neq 0$和$R > 0$存在，并且

$$\lim_{n\to\infty} \frac{|p-p_{n+1}|}{|p-p_n|^R} = \lim_{n\to\infty} \frac{|E_{n+1}|}{|E_n|^R} = A \tag{10}$$

则称该序列为以收敛阶R收敛到p，数A称为渐进误差常数。$R=1, R=2$时分别称序列$\{p_n\}_{n=0}^{\infty}$为线性收敛与二次收敛。

定理 6（牛顿-拉夫森迭代的收敛速度）： 设牛顿-拉夫森迭代产生序列$\{p_n\}_{n=0}^{\infty}$，收敛到函数$f(x)$的根p。如果p是单根，则是二次收敛，而且对于足够大的n有

$$|E_{n+1}| \approx \frac{|f''(p)|}{2|f'(p)|}|E_n|^2 \tag{11}$$

如果p是M阶多重根，则是线性收敛的，而且对于足够大的n有

$$|E_{n+1}| \approx \frac{M-1}{M}|E_n| \tag{12}$$

加速收敛

定理 7（牛顿-拉夫森迭代的加速收敛）：设牛顿-拉夫森算法产生的序列线性收敛到M阶根p，其中$M>1$，则牛顿-拉夫森迭代公式

$$p_k = p_{k-1} - \frac{Mf(p_{k-1})}{f'(p_{k-1})} \tag{13}$$

将产生一个收敛序列$\{p_n\}_{n=0}^{\infty}$二次收敛到p。也使用连续项的差异作为判别条件。

实验内容

1. (p40 1(d)) 使用程序 2.1 求解下面每个函数的不动点（尽可能多）近似值，答案精确到小数点后 12 位。同时，头早每个函数的图和直线$y=x$来显示所有不动点。
   (d)$g(x) = x^{x-\cos(x)}$

2. (p49 1) 如果在 240 个月内每月付款 300 美元，求解满足全部年金A为 500000 美元的利率I的近似值（精确到小数点后 10 位）。
3. (p69 5) 利用定理 2.7 中的加速牛顿-拉夫森算法修改程序 2.5，并用其求下列函数的M阶根p的近似值。
   1) $f(x)=(x-2)^5, M=5, p=2$, 初始值$p_0=1$。
   2) $f(x)=\sin(x^3), M=3, p=0$, 初始值$p_0=1$。
   3) $f(x)=(x-1)\ln(x), M=2, p=1$, 初始值$p_0=2$。
4. (p69 7) 设投射运动方程为

$$y=f(t)=9600\left(1-e^{-\frac{t}{15}}\right)-480t$$

$$x=r(t)=2400(1-e^{-\frac{t}{15}})$$

   a) 求当撞击地面时经过的时间，精确到小数点后 10 位。
   b) 求水平飞行行程，精确到小数点后 10 位。

实验分析

1. (p40 1(d)) 这个题目直接使用不动点迭代求不动点。但是由定理 3 我们知道使用不动点迭代有几个条件：(i)$g, g' \in C[a,b]$,(ii)K是一个正常数，(iii) $p_0 \in [a,b]$，(iv)对于所有$x \in [a,b]$，有$g(x) \in [a,b]$。直接看无法得出此函数是否符合这些条件，所以需要首先画图，确定可以使用不动点迭代的范围以及不动点的范围。如图所示：

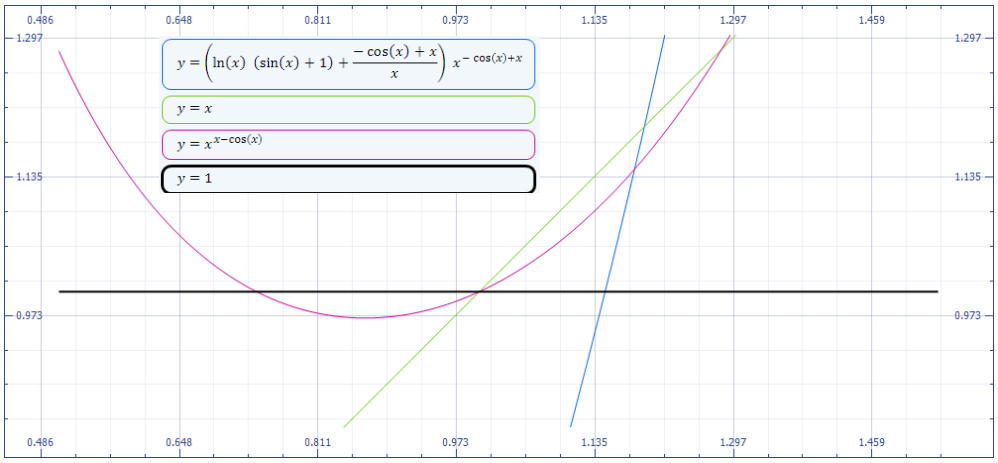

图 1：实验一分析图

可以看出，该函数有两个不动点，但是只有在$x=0.9$附近的不动点才满足不动点定理的要求，而在另一个不动点处导函数大于 1 无法应用导函数定理。同时我们可以大概估计不动点的初始值可取 0.9。

2. (p49 1) 此题目使用二分法。初始区间由估算可得为[0.14,0.16]。收敛判断条件为区间长度小于误差允许范围。
3. (p69 5) 此题包含三小题。用加速牛顿-拉夫森算法。分析待求解的三个函数，它们的导函数分别是$f'(x)=5(x-2)^4, f'(x)=3x^2\cdot\cos x^3, f'(x)=\ln(x)+\frac{x-1}{x}$。易知三个函数以及它们的导函数在根附近的区间内是二阶连续的，满足牛顿-拉夫森定理条件。由题目中给出的阶数可以使用加速收敛。
4. (p69 7) 此题用牛顿-拉夫森算法。分析待求解函数，它的导数为$f'(t)=640\cdot e^{-\frac{t}{15}}-$

480。易知$f(t)$在根附近的区间时二阶连续的，满足牛顿-拉夫森定理条件。直接用牛顿-拉夫森迭代即可。初始值带入计算试值取$t = 5$。求出时间之后把时间带入$r(t)$计算水平行程。

实验代码

1. (p40 1(d))

```
/*王旋宇  9.28.14  改*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//函数 g（）为待求不动点的方程
double g(double x);

int main()
{
    //freopen("out.txt","w",stdout);
    //p 为待求不动点，pnext 用来储存每次迭代产生新的序列值
    double p,pnext;
    //err 为连续项之间的差异，tol 为允许误差范围
    double err,tol;
    //记录迭代次数
    int n = 0;
    //初始值取 0.8
    p = 0.9;
    //精度要求为小数点后 12 位
    tol = 1E-12;

    do{
        //输出当前序列值得函数值，调试时使用
        printf("g(%.12f) = %.12f\n",p,g(p));
        //生成下一个序列值
        pnext = g(p);
        //迭代次数加 1
        n++;
        //计算连续项之间的差异
        err = pnext - p;
        //调试的时候使用，输出误差查看
        printf("error = %.12f-%.12f=%.15f\n",pnext,p,err);
        p = pnext;
        //调试时防止序列不收敛可以每一次循环观察结果，便于调试，及时
kill 程序
        //getchar();
    }while(fabs(err) >= tol);//当不满足精度要求的时候再次循环
```

```
    //输出结果
    printf("迭代了%d 次\n",n);
    printf("ans:fixed_point = %10.12f\tg(q) = %10.12f",pnext,g(pnext));
    return 0;
}

//待求不动点函数
double g(double p)
{
    //函数为 g(x)=x^(x-cos(x))
    return (pow(p,p-cos(p)));
}
```

2. (p49 1)

```
/*王旋宇  9.28.14  改*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//函数 f()来计算年金
double f(double rate);
//函数 bisevtion()用来进入二分法
void bisection(double a0,double b0);

int main()
{
    //freopen("out.txt","w",stdout);
    //a,b 分别为区间左右端点值
    double a,b;
    //初始区间赋值
    a = 0.14;
    b = 0.16;
    //进入二分法开始计算
    bisection(a,b);

    return 0;
}

void bisection(double a,double b)
{
    //err 为误差
    double err;
    //tol 为误差允许范围
    double tol = 1E-10;
    //c 为区间中点
```

```
    double c;
    //记录迭代次数
    int n = 0;

    //使用 do...while 循环避免第一次循环需要判断
    do{
        //计算区间中点
        c = (a+b)/2;
        //调试使用，观察区间及其中点取值，检查区间是否构造正确
        printf("a = %.10f\tb = %.10f\tc = %.10f\n",a,b,c);
        //构造新区间
        if(f(c) != 0){//判断中点是不是零点，不是的话构造新的区间继续迭代
            //调试使用。输出 c 处的函数值看是否迭代收敛
            printf("f(c) = %.10f\n",f(c));
            //判断零点在[a,c]或者[b,c]中，进而构造新的区间
            if(f(c) * f(a) > 0)a = c;
            else b = c;
            n++;
        }
        else break;//如果 c 是零点，跳出循环
        //计算误差
        err = fabs(b - a);
    }while(err > tol);//误差大于允许范围的时候继续迭代
    //输出结果
    printf("迭代了%d 次\n",n);
    printf("ans:rate = %.10f",c);
}

double f(double rate)
{
    //monthly、month、total 分别是月金、月数、和年金
    double monthly;
    double nmonth;
    double total;
    //r12 用来代替公式中的 I/12，以便使公式更简洁不容易出错
    double r12;
    r12 = rate/12;
    //初始化赋值
    monthly = 300;
    nmonth = 240;
    total = 500000;
    //计算年金并返回
    return (12*monthly/(rate))*( pow(1+r12,nmonth) -1)-total;
}
```

3. (p69 5)

```
/*王旋宇  9.28.14 改*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//f()为待求解函数。此为求其值
double f(double x);
//函数 f_d()为待求解函数的导数。此为求导数值
double f_d(double x);
//函数 ac_nt_rps()为加速牛顿-拉夫森算法程序
void ac_nt_rps();

int main()
{
    //freopen("out.txt","w",stdout);
    //进入加速牛顿-拉夫森算法程序
    ac_nt_rps();
    return 0;
}

void ac_nt_rps()
{
    //M 与 P 分别为函数根的阶与对应的根
    double M;
    double p;
    //输入待求根的阶
    printf("order M = ");
    scanf("%lf",&M);
    //输入待求函数的初始值
    printf("start with p0 = ");
    scanf("%lf",&p);

    //迭代
    while(f(p) != 0){//当所得序列值不是所求根的时候
        //调试使用，输出当前序列值以及函数值
        printf("f(%f) = %.10f\n",p,f(p));
        //加速牛顿-拉夫森迭代公式计算新序列值
        p = p - (M*f(p))/f_d(p);
        //调试使用，检验每次生成项的函数值
        //getchar();
    }
    //输出答案
    printf("ans:p = %f",p);
```

```
}

double f(double x)
{
    //分别为计算第一题、第二题、第三题的函数值
    //return ( pow((x - 2),5) );
    //return ( sin(pow(x,3)) );
    return ( (x-1)*log(x) );
}

double f_d(double x)
{
    //分别为计算第一题、第二题、第三题的导函数值
    //return (5 * pow(x - 2,4));
    //return (3 * pow(x,2) * cos(pow(x,3)));
    return (log(x) + (x - 1)/x);
}
```

4. (p69 7)

```
/*王旋宇  9.28.14  改*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//函数 nt_rps()为牛顿-拉夫森算法程序
void nt_rps();
//f(),f_d(),r()分别为计算竖直方向高速、竖直方向导函数数值、水平方向行程
double f(double t);
double f_d(double t);
double r(double t);

int main()
{
    //freopen("out.txt","w",stdout);
    //进入牛顿-拉夫森算法
    nt_rps();
    return 0;
}

void nt_rps()
{
    //t 为时间，预估初始值为 5
    double t = 5;
    //tol 为误差容忍范围
    double tol;
```

```
    //printf("f(5) = %.10f\n",f(5));
    tol = 1E-10;

    //迭代
    while(fabs(f(t)) > tol){//如果超过了误差容忍范围继续迭代
        //牛顿-拉夫森迭代公式
        t = t - f(t)/f_d(t);
        //调试使用，观察生成项的值以及函数值
        printf("f(%.10f) = %.10f\n",t,f(t));
    }
    //迭代结束，输出答案
    printf("\nans: f(%.10f) = %.10f\nr(t) = %.10f",t,f(t),r(t));
}

double f(double t)
{
    //返回 y 方向高度
    return ( 9600 * (1 - exp((-t) / 15)) - 480*t );
}

double f_d(double t)
{
    //返回 y 方向运动方程的导函数值
    return ( 640 * (exp((-t) / 15)) - 480 );
}

double r(double t)
{
    //返回 x 方向运动行程
    return ( 2400 * (1 - exp((-t) / 15) ) );
}
```

实验结果与分析

1. 实验一结果如下表所示：

n	p_n	$\lvert p_n - p_{n-1} \rvert$
0	0.900000000000	
1	0.971094670700	0.071094670700199
2	0.988141866996	0.017047196295830
3	0.994789915954	0.006648048957911
4	0.997651516340	0.002861600385592
5	0.998929875619	0.001278359279118
6	0.999510032548	0.000580156929053
7	0.999775175326	0.000265142778717

8	0.999896735415	0.000121560088626
9	0.999952547821	0.000055812405455
10	0.999978190209	0.000025642388796
11	0.999989974906	0.000011784697059
12	0.999995391660	0.000005416753803
13	0.999997881593	0.000002489933111
14	0.999999026181	0.000001144587747
15	0.999999552339	0.000000526158270
16	0.999999794212	0.000000241872459
17	0.999999905400	0.000000111187940
18	0.999999956512	0.000000051112782
19	0.999999980009	0.000000023496416
20	0.999999990810	0.000000010801246
21	0.999999995775	0.000000004965307
22	0.999999998058	0.000000002282540
23	0.999999999107	0.000000001049278
24	0.999999999590	0.000000000482351
25	0.999999999811	0.000000000221736
26	0.999999999913	0.000000000101931
27	0.999999999960	0.000000000046858
28	0.999999999982	0.000000000021540
29	0.999999999992	0.000000000009902
30	0.999999999996	0.000000000004552
31	0.999999999998	0.000000000002093
32	0.999999999999	0.000000000000962

表 1：实验一实验结果

该表最左边一列代表迭代次数，中间代表迭代生成序列，右边代表相邻两项的差异。可以看出，最终迭代了 32 次，在最后如果用精度到小数点后 15 位可以看出最终结果是 0.999999999999182，保留到小数点后 12 位就是 0.999999999999。此时误差为 0.000000000000962（保留小数点后 15 位），小于设定的误差容忍限度 1×10^{-12}。

该函数和$y = x$的交点图像如下图：

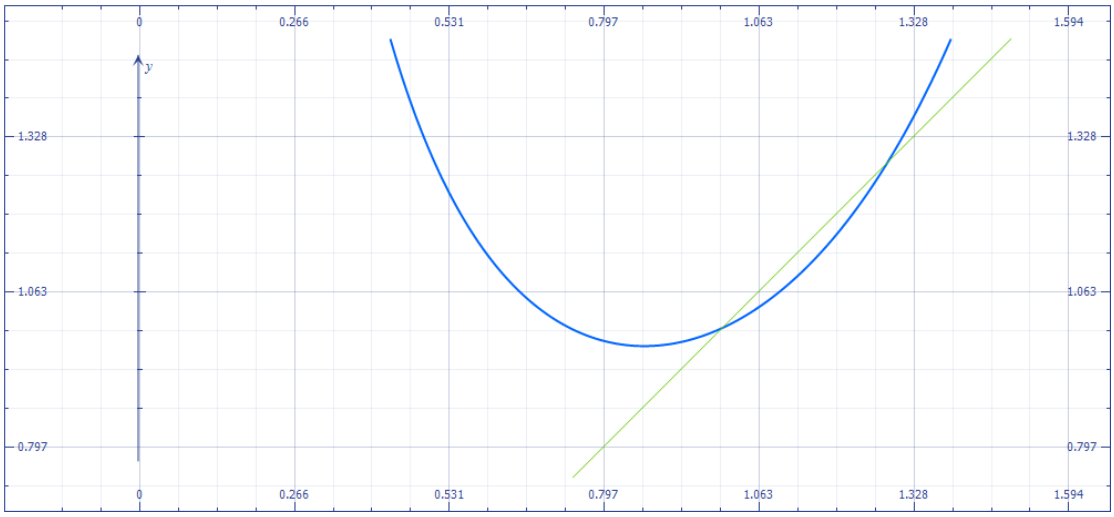

图 2：实验一函数的不动点

直接使用不动点迭代计算效果不好，有表 1 可以看出，显然生成项逐渐的逼近不动点$p = 1$但是逼近的不好。32 次迭代之后就达到了精度要求但是依然没有找到不动点。

2. 实验二结果如下表所示：

n	左端点a_k	中点，c_k	右端点b_k	年金$f(c_k)$-500000	$\|a-b\|$
1	0.1400000000000	0.1500000000000	0.1600000000000	-50828.1555572751000	0.0100000000000
2	0.1500000000000	0.1550000000000	0.1600000000000	-17793.5973994457000	0.0050000000000
3	0.1550000000000	0.1575000000000	0.1600000000000	-280.8918928686390	0.0025000000000
4	0.1575000000000	0.1587500000000	0.1600000000000	8736.4134648080300	0.0012500000000
5	0.1575000000000	0.1581250000000	0.1587500000000	4205.6423558383600	0.0006250000000
6	0.1575000000000	0.1578125000000	0.1581250000000	1956.8738824397400	0.0003125000000
7	0.1575000000000	0.1576562500000	0.1578125000000	836.6191736479580	0.0001562500000
8	0.1575000000000	0.1575781250000	0.1576562500000	277.5211236131370	0.0000781250000
9	0.1575000000000	0.1575390625000	0.1575781250000	-1.7709590712910	0.0000390625000
10	0.1575390625000	0.1575585937500	0.1575781250000	137.8536818051630	0.0000195312500
11	0.1575390625000	0.1575488281250	0.1575585937500	68.0360121092409	0.0000097656250
12	0.1575390625000	0.1575439453125	0.1575488281250	33.1311892976733	0.0000048828125
13	0.1575390625000	0.1575415039063	0.1575439453125	15.6797808383669	0.0000024414063
14	0.1575390625000	0.1575402832031	0.1575415039063	6.9543273131549	0.0000012207031
15	0.1575390625000	0.1575396728516	0.1575402832031	2.5916632148014	0.0000006103516
16	0.1575390625000	0.1575393676758	0.1575396728516	0.4103468487471	0.0000003051758
17	0.1575390625000	0.1575392150879	0.1575393676758	-0.6803074308102	0.0000001525879
18	0.1575392150879	0.1575392913818	0.1575393676758	-0.1349806037079	0.0000000762939
19	0.1575392913818	0.1575393295288	0.1575393676758	0.1376830270621	0.0000000381470
20	0.1575392913818	0.1575393104553	0.1575393295288	0.0013512050712	0.0000000190735
21	0.1575392913818	0.1575393009186	0.1575393104553	-0.0668147181686	0.0000000095367
22	0.1575393009186	0.1575393056870	0.1575393104553	-0.0327317578036	0.0000000047684
23	0.1575393056870	0.1575393080711	0.1575393104553	-0.0156902766900	0.0000000023842
24	0.1575393080711	0.1575393092632	0.1575393104553	-0.0071695496513	0.0000000011921

25	0.1575393092632	0.1575393098593	0.1575393104553	-0.0029091861334	0.0000000005960
26	0.1575393098593	0.1575393101573	0.1575393104553	-0.0007790042854	0.0000000002980
27	0.1575393101573	0.1575393103063	0.1575393104553	0.0002860866411	0.0000000001490
28	0.1575393101573	0.1575393102318	0.1575393103063	-0.0002464587428	0.0000000000745

表 2：实验二数据表

n 为迭代次数，在最后一次迭代中，区间宽度已经小于设定的误差容忍限度1×10^{-10}。此时可以看出收益与预期收益相比还是绝对误差不小，但是相对误差也为$-4.929174856 \times 10^{-10}$达到了小数点后十位。可以认为最后一次产生的中点值就是所求解。运用公式(4)求解计算次数：

$$N = int\left(\frac{\ln(b-a)-\ln(\delta)}{\ln(2)}\right) = int(27.5754247590989) = 27 \tag{14}$$

3. 实验三结果

1) 对于$f(x) = (x-2)^5, M = 5, p = 2$, 初始值$p_0 = 1$。实验结果如下图所示：

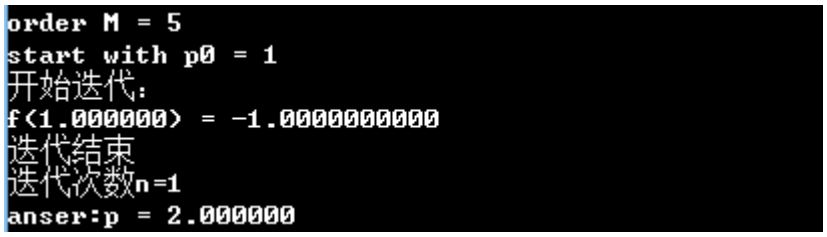

图 3：实验三第一小题实验结果

经过一次加速牛顿-拉夫森迭代之后就迭代到根的精确值。对比之下如果使用牛顿-拉夫森方法则会缓慢的逼近$p = 2$但是几百次循环之后还是无法达到。也需要添加精度判别条件才能终止循环。

2) 对于$f(x) = \sin(x^3), M = 3, p = 0$, 初始值$p_0 = 1$。实验结果如下图所示：

图 4：实验三第二小题实验结果

可以看出对于此函数进行了四次迭代，就迭代到根的精确值。对比之下用牛顿-拉夫森方法是进行 143 次迭代生成项的小数点后 25 位都已经为零，但是仍然在缓慢的逼近。也需要添加精度判别条件才能终止循环。

3) 对于$f(x) = (x-1)\ln(x), M = 2, p = 1$, 初始值$p_0 = 2$。实验结果如下图所示：

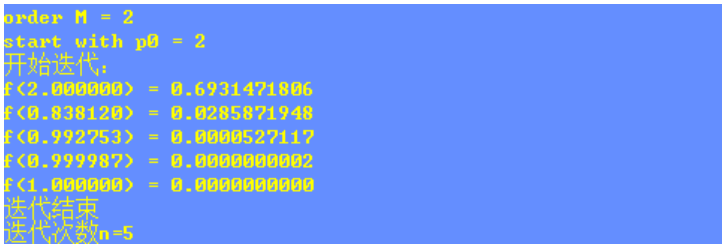

图 5：实验三第三小题实验结果

可以看出对于此函数进行了四次迭代，就迭代到根的精确值。对比之下用牛顿-拉夫森方法是进行了 53 次迭代之后才得出结果。

4. (p69 7) 实验结果如下表所示：

迭代次数 n	生成项t_n	$f(t)$的值	$\lvert t_n - t_{n-1}\rvert$
1	5.00000000000	321.299418491623	
2	20.0000000000	-2530.5325259110	15.000000000000
3	11.8710242444	-448.8992335576	8.128975755650
4	9.5077270186	-56.9513132048	2.363297225750
5	9.1022367641	-1.8778672040	0.405490254540
6	9.0879179005	-0.0023849390	0.014318863610
7	9.0878996688	-0.0000000039	0.000018231637
8	9.0878996688	-0.0000000000	0.000000000000

表 3：实验 4 数据表

可得撞击地面时经过的时间是 9.0878996688，水平飞行行程为 1090.5479602542。计算中，初始值设为 t = 5，经过 8 次迭代就可以得到符合精度要求的结果。

实验结论及总结

由实验内容一的结果及其分析可以看出，在使用一个算法的时候，一定要注意到算法的适用条件。在实验分析中可以看出有两个不动点，但是通过对函数的导函数值进行判断就可以知道只有一个适合使用不动点迭代，并且在进行迭代的时候初始值也需要慎重选择。此次实验中选择的初始值是 0.9，可以满足封闭性。在调试的时候可以选择一个其它的数值例如 0.55 或者 1.2，可以看出来在期间[0.55，0.64]以及[1.15，1.2]时并不满足对于区间之内所有点所有的$|g'(x)| \leq K < 1$但是依然可以收敛。或许定理 3 的注 2 可以解释这个问题。需要更进一步的探究这个问题。但是在选择初始值为 0.5 或者 1.3 是可以很明显的看出来生成项不收敛。

实验二可以看出二分法在满足二分法定理的时候可以很放心的使用不用担心出现其它问题。并且 体现了二分法的优点：可以对一个计算结果精度进行预先估计。在此实验中可以算出预估的结束迭代次数的确为 27 次（表二中 n 为 1 为初始项，未经过迭代）。与实验结果吻合。

实验三体现了牛顿-拉夫森算法的缺陷以及与加速牛顿-拉夫森之间的差别。当把题目中要求使用加速牛顿-拉夫森的几个函数使用牛顿-拉夫森的时候，发现有的可能陷入循环，有的则需要更多的（此实验中为 10 倍多的迭代次数）迭代次数才能得出同样的结果。

实验四中，终止迭代的判定条件本来是相邻项之间的差异，但是把判定条件修改为函数值与期望值得差的绝对值得时候（即$f(t) - 0$），发现经过的迭代次数完全相同。还没有研究之间是否有什么关系，有待继续深入探究。