北京化工大学

Beijing University of Chemical Technology

计算方法讲义 (四)

非线性方程求根

Cheng Yong

目录

创建日期：2019 年 7 月 5 日
更新日期：2020 年 2 月 13 日

第 1 章　非线性方程求根

1.1　迭代法

绪论中已经介绍过算法设计的校正思想。迭代法是求解方程近似根的一种方法，这种方法的关键是确定迭代函数 $\varphi(x)$(将方程 $f(x)=0$ 变换为 $x=\varphi(x)$)，建立迭代格式 $x_{k+1}=\varphi(x_k)$，然后从给定的初值 x_0 出发迭代出一系列的近似值 $x_1,x_2,\cdots,x_k,\cdots$，直到逼近方程的根 x^*，直到满足精度要求 $|x_{k+1}-x_k|<\epsilon$ 为止。

简单迭代法又称逐次迭代法，基本思想是构造不动点方程，以求得近似根。即由方程 $f(x)=0$ 变换为 $x=\varphi(x)$, 然后建立迭代格式：

$$x_{k+1}=\varphi(x_k) \tag{1.1}$$

当给定初始值 x_0 后，由迭代格式可求得数列 $\{x_k\}$。如果 $\{x_k\}$ 收敛于 x^*，则它就是方程的根。因为

$$x^*=\lim_{k\to\infty}x_{k+1}=\lim_{k\to\infty}x_k=\varphi(\lim_{k\to\infty}x_k)=\varphi(x^*) \tag{1.2}$$

例 1. 求方程

$$f(x)=x-10^x+2=0 \tag{1.3}$$

的一个根，取 $x_0=0$。

解：由于

$$10^x=x+2,\quad x=\frac{\log(x+2)}{\log 10},\quad \varphi(x)=\frac{\log(x+2)}{\log 10} \tag{1.4}$$

得到迭代格式为 $x_{k+1}=\varphi(x_k)$

$x_1=0.30102999566398114$
$x_2=0.36192228006214167$
$\cdots$
$x_6=0.3757965228864938$
$x_7=0.3758092423816728$

1.2　开方法

对于给定的 $a>0$，求开方值 $\sqrt{a}$ 就是要求解二次方程

$$x^2-a=0 \tag{1.5}$$

为此可使用校正技术从预报值生成校正值来逐步逼近方程的解。

设给定某个预报值 x_k，希望借助于某种简单方法确定校正量 Δx，使校正值

$$x_{k+1} = x_k + \Delta x \tag{1.6}$$

更好的满足方程，即：

$$x_k^2 + 2x_k\Delta x + (\Delta x)^2 \approx a \tag{1.7}$$

成立。设校正值 Δx 是个小量，舍去高阶小量 $(\Delta x)^2$ 后令：

$$x_k^2 + 2x_k\Delta x = a \tag{1.8}$$

从中解出 Δx，得：

$$\Delta x = \frac{a - x_k^2}{2x_k} \tag{1.9}$$

代入上式，即可求出开方公式：

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \quad k = 1, 2, 3, \cdots \tag{1.10}$$

上述演绎过程表明开方法的设计思想是逐步线性化，即将二次方程的求解化归为一次方程求解过程的重复。

定理 1. *开方算法* 任给初值 $x_0 > 0$，反复利用迭代公式即可获得满足精度要求的开方值：

$$\begin{cases} x_0 > 0 \\ x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \quad k = 1, 2, 3, \cdots \end{cases} \tag{1.11}$$

直到 $|x_{k+1} - x_k| < \varepsilon$($\varepsilon$ 为给定的精度) 为止，x_{k+1} 即为所求。

例 2. 用开方算法求 $\sqrt{2}$，取 $x_0 = 1$，$\varepsilon = 10^{-6}$

解：求解过程如下：

$$x_1 = \frac{1}{2}(x_0 + \frac{a}{x_0}) = \frac{1}{2}(1 + \frac{2}{1}) = 1.5, \quad x_2 = 1.41666667, \quad \cdots \tag{1.12}$$

k	x_k
1	1.500000
2	1.416667
3	1.414216
4	1.414214
5	1.414214

表 1.1: 计算结果

因此，$\sqrt{2} \approx 1.414214$，精确值为 1.41421356。

定理 2 (开方公式收敛性定理). 开方公式

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \quad k = 1, 2, 3, \cdots \tag{1.13}$$

对于任意给定的初值 $x_0 > 0$ 均收敛，即

$$\lim_{k\to\infty} x_k \to \sqrt{a} \tag{1.14}$$

或表示为迭代误差

$$\lim_{k\to\infty} e_k = \lim_{k\to\infty} |x_k - \sqrt{a}| \to 0 \tag{1.15}$$

证明. 由

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \tag{1.16}$$

可得

$$x_{k+1} - \sqrt{a} = \frac{1}{2}(x_k + \frac{a}{x_k}) - \sqrt{a} \tag{1.17}$$

$$= \frac{1}{2x_k}(x_k^2 + a - 2x_k\sqrt{a}) \tag{1.18}$$

$$= \frac{1}{2x_k}(x_k - \sqrt{a})^2 \tag{1.19}$$

同理

$$x_{k+1} + \sqrt{a} = \frac{1}{2}(x_k + \frac{a}{x_k}) + \sqrt{a} \tag{1.20}$$

$$= \frac{1}{2x_k}(x_k^2 + a + 2x_k\sqrt{a}) \tag{1.21}$$

$$= \frac{1}{2x_k}(x_k + \sqrt{a})^2 \tag{1.22}$$

两式相除

$$\frac{x_{k+1} - \sqrt{a}}{x_{k+1} + \sqrt{a}} = \frac{(x_k - \sqrt{a})^2}{(x_k + \sqrt{a})^2} = \left(\frac{x_k - \sqrt{a}}{x_k + \sqrt{a}}\right)^2 \tag{1.23}$$

这样有：

$$\frac{x_k - \sqrt{a}}{x_k + \sqrt{a}} = \left(\frac{x_{k-1} - \sqrt{a}}{x_{k-1} + \sqrt{a}}\right)^2 \tag{1.24}$$

$$= \left(\frac{x_{k-2} - \sqrt{a}}{x_{k-2} + \sqrt{a}}\right)^{2^2} = \cdots = \left(\frac{x_0 - \sqrt{a}}{x_0 + \sqrt{a}}\right)^{2^k} \tag{1.25}$$

令 $q = |\frac{x_0 - \sqrt{a}}{x_0 + \sqrt{a}}|$，显然当 $x_0 > 0$ 时，有 $0 < q < 1$，
则有 $\frac{x_k - \sqrt{a}}{x_k + \sqrt{a}} = q^{2^k}$，因此：

$$\lim_{k\to\infty} x_k = \frac{1 + q^{2^k}}{1 - q^{2^k}}\sqrt{a} = \sqrt{a} \tag{1.26}$$

□

1.3 Newton 法

Newton 牛顿迭代公式构建的基本思想是通过 Taylor 展开将非线性方程线性化。设 x_k 是 $f(x)=0$ 的一个近似根，把 $f(x)$ 在 x_k 处作泰勒展开

$$f(x)=f(x_k)+f'(x_k)(x-x_k)+\frac{f''(x_k)}{2!}(x-x_k)^2+\cdots+ \tag{1.27}$$

若取前两项来近似代替 $f(x)$(称为 $f(x)$ 的线性化)，则得近似的线性方程：

$$f(x)\approx f(x_k)+f'(x_k)(x-x_k)=0 \tag{1.28}$$

设 $f'(x_k)\neq 0$，令其解为 x_{k+1}，得

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)} \tag{1.29}$$

这称为 $f(x)=0$ 的牛顿迭代公式。它的迭代函数为：

$$x=x-\frac{f(x)}{f'(x)} \tag{1.30}$$

显然是 $f(x)=0$ 的同解方程，故其迭代函数为：

$$\varphi(x)=x-\frac{f(x)}{f'(x)}\quad (f'(x)\neq 0) \tag{1.31}$$

由

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)} \tag{1.32}$$

知 x_{k+1} 是点 $(x_k,f(x_k))$ 处 $y=f(x)$ 的切线：

$$\frac{y-f(x_k)}{x-x_k}=f'(x_k) \tag{1.33}$$

与 x 轴的交点的横坐标，如下图所示。

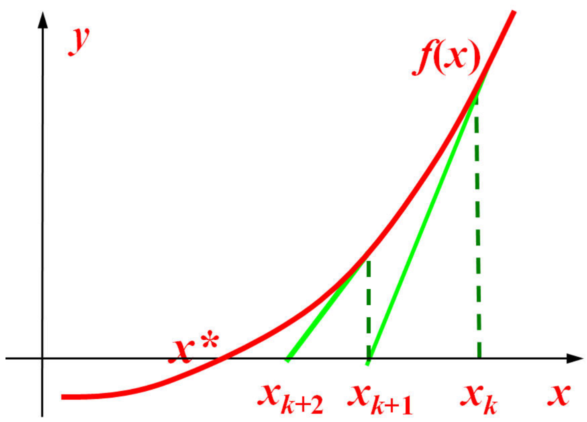

图 1.1: Newton 法的几何解释

也就是说，新的近似值 x_{k+1} 是用曲线 $y=f(x)$ 在 $(x_k, f(x_k))$ 的切线与 x 轴相交得到的。

继续取点 $(x_{k+1}, f(x_{k+1}))$，再做切线与 x 轴相交，又可得 $x_{k+2}, \cdots$。由图可见，只要初值取的充分靠近根 x^* ，这个序列就会很快收敛于根 x^*。Newton 迭代法又称切线法。

定理 3. 定理 设 $f(x)$ 在 $[a,b]$ 上满足下列条件：

- $f(a)\cdot f(b)<0$；
- $f'(x)\neq 0$；
- $f''(x)$ 存在且不变号；
- 取 $x_0\in[a,b]$，使得 $f''(x)\cdot f(x_0)>0$；

则由迭代公式确定的牛顿迭代法序列 $\{x_k\}$ 收敛于 $f(x)$ 在 $[a,b]$ 上的唯一根 x^*。

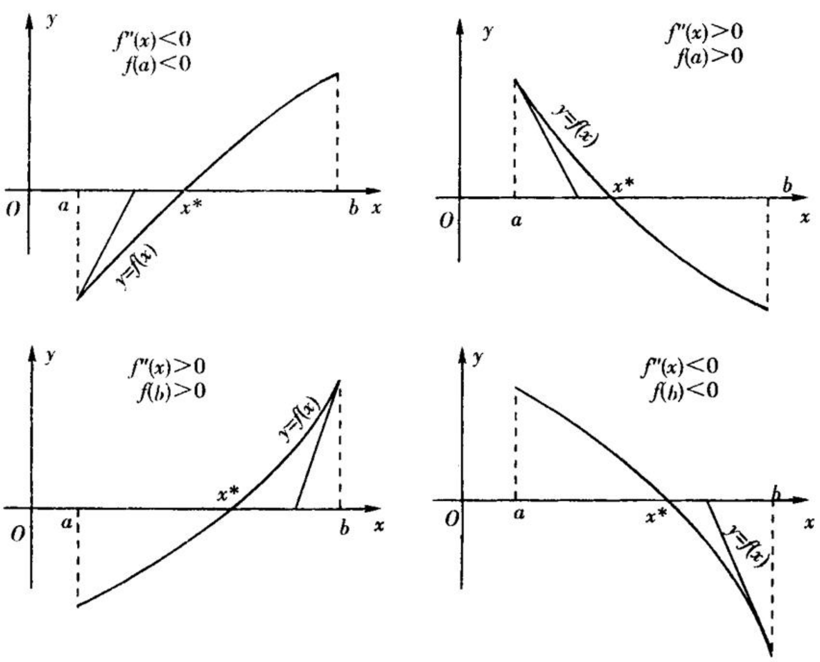

图 1.2: 牛顿迭代法条件

Newton 迭代法的收敛性依赖于 x_0 的选取。

例 3. 用牛顿法求下面方程的根：

$$f(x)=x^3+2x^2+10x-20=0 \tag{1.34}$$

解：因

$$f'(x) = 3x^2 + 4x + 10 \tag{1.35}$$

所以迭代公式为

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1.36}$$

$$= x_k - \frac{x_k^3 + 2x_k^2 + 10x_k - 20}{3x_k^2 + 4x_k + 10} \tag{1.37}$$

取 $x_0 = 1$，

$$x_1 = x_0 - \frac{x_0^3 + 2x_0^2 + 10x_0 - 20}{3x_0^2 + 4x_0 + 10} \tag{1.38}$$

$$x_1 = 1 - \frac{1^3 + 2 \cdot 1^2 + 10 \cdot 1 - 20}{3 \cdot 1^2 + 4 \cdot 1 + 10} = 1.411764706 \tag{1.39}$$

同理，可求 $x_2, x_3, \cdots$，计算结果列于下表。

k	x_k
1	1.411764706
2	1.369336471
3	1.368808189
4	1.368808108

表 1.2: 计算结果

从计算结果可以看出，牛顿法的收敛速度是很快的，进行了四次迭代就得到了较满意的结果，精度为 10^{-7}。

定义 1. 定义：p 阶收敛 如果迭代误差 $e_k = x^* - x_k$ 当 $k \to \infty$ 时成立：

$$\frac{e_{k+1}}{e_k^p} \to c \quad (c \neq 0\text{的常数}) \tag{1.40}$$

则称迭代过程是 p 阶收敛的。当 $p = 1$ 时称线性收敛，当 $p = 2$ 时称平方收敛，当 $1 < p < 2$ 时称方法为超线性收敛。

定理 4. 定理 *Newton* 迭代法

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1.41}$$

在 $f(x) = 0$ 的单根 x^* 临近为平方收敛。

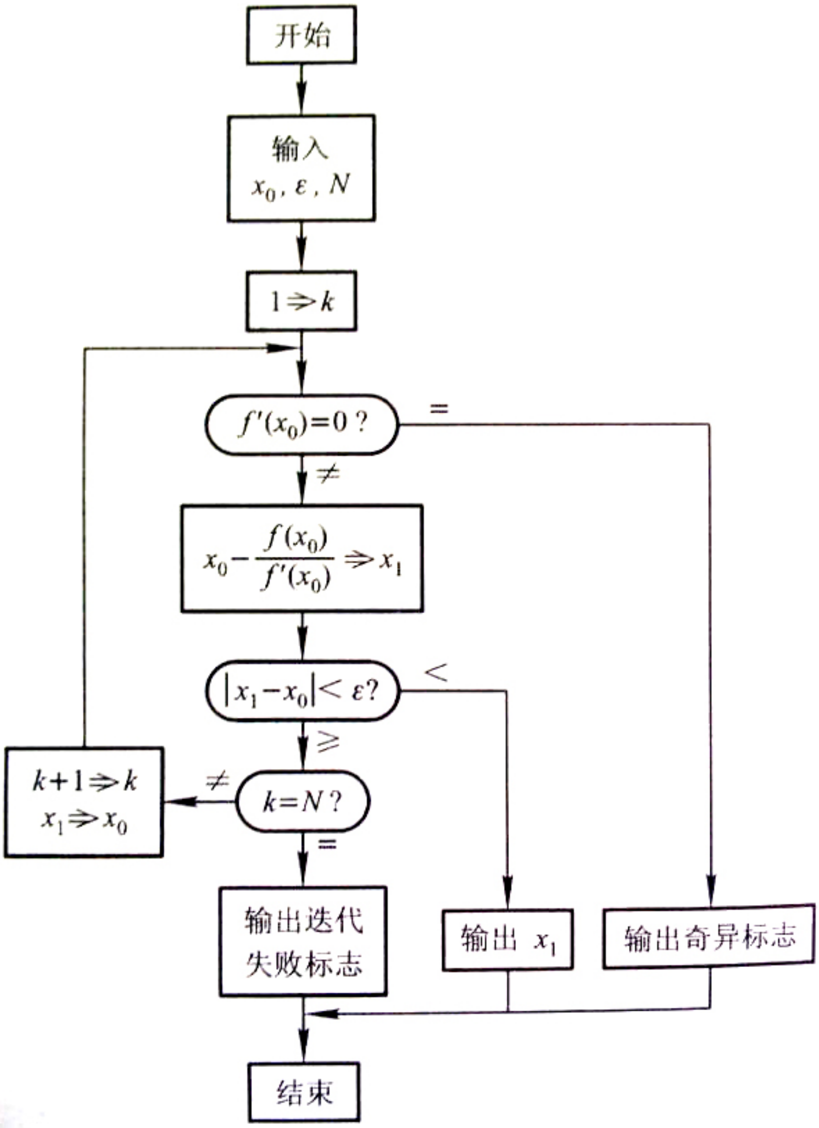

图 1.3: 牛顿法流程图

```
#include <stdio.h>
#include <math.h>
double f(double x);
double df(double x);


void main(void) {
    double epsilon,x0,x1,fx0,dfx0;
    long i,maxi;
    printf("\n请输入x的精度要求：");
    scanf("%lf",&epsilon);
    printf("\n请输入迭代初值：");
    scanf("%lf",&x1);
    printf("\n请输入最大迭代次数：");
```

```c
    scanf("%ld",&maxi);

    for(i=0; i<maxi; i++) {
        x0=x1;
        fx0=f(x0);
        dfx0=df(x0);
        x1=x0-fx0/dfx0;
        if(fabs(x1-x0)<=epsilon)
            break;
    }

    if(i<maxi)
        printf("\n方程f(x)=0的根x=%lf。",x1);
    else
        printf("\n迭代次数已超过上限。");
}


double f(double x) {
    return x*exp(x) - 1; /*计算并返回函数值f(x)*/
}
double df(double x) {
    return exp(x) + x*exp(x); /*计算并返回函数值f'(x)*/
}
```

图 1.4: Newton 迭代法结果

Newton 迭代法优缺点：

Newton 迭代法逻辑结构简单、收敛速度很快 (平方收敛)，但它通常依赖初值 x_0 的选取，如果初值 x_0 选择不当，将导致迭代发散或产生无限循环；此外，每一步迭代都需要计算导数值 $f'(x)$，有时计算 $f'(x)$ 是不方便的。

基于这两点，产生了几种 Newton 迭代法的变形形式。

1. 牛顿下山法；

2. 弦截法；
3. 快速弦截法；

1.4 改进的牛顿法

一般地说，Newton 法的收敛性依赖于初值 x_0 的选取，如果 x_0 偏离解 x^* 较远，则 Newton 法可能发散或产生无限循环。

例 4. 用 *Newton* 求方程

$$x^3 - x - 1 = 0 \tag{1.42}$$

在 $x = 1.5$ 附近的一个根。

解：因

$$f'(x) = 3x^2 - 1 \tag{1.43}$$

可得牛顿迭代公式：

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1.44}$$

$$= x_k - \frac{x_k^3 - x_k - 1}{3x_k^2 - 1} \tag{1.45}$$

分别取 $x_0 = 1.5$ 和 $x_0 = 0.6$，计算结果如下表。

k	x_k	x_k
0	1.5	0.6
1	1.34783	17.90000
2	1.32520	11.94680
3	1.32472	7.985519

表 1.3: 计算结果

由上表可知道，当 $x_0 = 0.6$ 时结果偏离所求的根，不收敛 (发散) 或收敛较慢。

1.4.1 Newton 下山法

为了防止迭代发散，通常对迭代过程再附加一项要求，即保证函数值单调下降：

$$|f(x_{k+1})| < |f(x_k)| \tag{1.46}$$

满足这项要求的算法称下山法。将 Newton 法与下山法结合使用，即在下山法保证迭代函数值稳定下降的前提下，用 Newton 法加快速度，即可得到如下 Newton 下山法：

$$x_{k+1} = x_k - \lambda \frac{f(x_k)}{f'(x_k)} \tag{1.47}$$

其中 $0 < \lambda < 1$，称下山因子，在迭代过程中通过适当地选取 λ 以使下山条件 $|f(x_{k+1})| < |f(x_k)|$ 满足。下山因子的选择是个逐步探索的过程，从 $\lambda = 1$ 开始反复将因子 λ 的值减半进行试算，一旦单调条件 $|f(x_{k+1})| < |f(x_k)|$ 满足，则称为"下山成功"。反之，如果在上述过程中找不到使下山条件 $|f(x_{k+1})| < |f(x_k)|$ 成立的下山因子 λ，则称"下山失败"，这时需另选初值 x_0 重算。

例 5. 使用下山法求方程 $f(x) = x^3 - x - 1 = 0$ 的根，取 $x_0 = 0.6$。

解：迭代公式如下：

$$x_{k+1} = x_k - \lambda \frac{f(x_k)}{f'(x_k)} = x_k - \lambda \frac{x_k^3 - x_k - 1}{3x_k^2 - 1} \tag{1.48}$$

牛顿下山法的计算结果：

k	λ	x_k
0	1	0.6
1	$\frac{1}{2^5}$	1.14063
2	1	1.36681
3	1	1.32628
4	1	1.32472

表 1.4: 计算结果

```cpp
#include <iostream>
#include <math.h>
#include <queue>
#include <iomanip>

using namespace std;

//定义队列保存近似根
queue <double> root;
//记录每步的下山次数
queue <int> count;
//记录每步的因子
queue <double> factor;
//定义原函数
```

```
double function(double x,double y);
//定义导函数
double derivative(double x);
//定义牛顿下山法
void Newton(double x,double e,int N,int M);
//输出函数
void print();

int main()
{
    //定义初值，误差限
    double x,e;
    //定义迭代最大次数，下山最大次数
    int N, M;
    char c = 'y';
    while(c != 'n'){
        cout<<"请依次输入初值、误差限、迭代最大次数、下山最大次数:";
        cin>>x>>e>>N>>M;
        Newton(x,e,N,M);
        print();
        cout<<"是否继续？按n退出：";
        cin>>c;
    }
    return 0;
}
//定义原函数
double function(double x)
{
    return pow(x,3) - x - 1;
}
//定义导函数
double derivative(double x)
{
    return 3*x*x - 1;
}
//定义牛顿下山法
void Newton(double x,double e,int N,int M)
{
    int k = 0,i;
    double lamda,x1;
    while(k<N)
    {
        if(derivative(x)==0){
            cout<<"奇异标志"<<endl;
            return;
        }
        else{
            i = 0;
```

```
            lamda = 1;
            factor.push(lamda);
            while(true)
            {
                x1 = x - lamda * function(x)/derivative(x);
                root.push(x1);
                if(fabs(function(x1) < fabs(function(x))))
                {
                    break;
                }
                else{
                    i +=1;
                    lamda *= 0.5;
                }
                if(i>=M)
                {
                    cout<<"迭代失败，请重新输入初值x"<<endl;
                    return;
                }
                factor.push(lamda);
                count.push(i);
                root.push(x1);
            }
            if(fabs(x1 - x) < e)
            {
                cout<<"结果为："<<x1<<endl;
                return;
            }
            else{
                k+=1;
                x = x1;
            }
        }
    }
    cout<<"迭代失败"<<endl;
    return;
}
//输出函数
void print()
{
    cout<<"
      ---------------------------------------------------------------------------
      "<<endl;
    cout<<" k "<<"近似根 "<<"下山因子 "<<"下山次数 "<<endl;
    int k =1,c;
    double f;
    while(!root.empty()){
        f = factor.front();
```

```
        c = count.front();
        cout<<setiosflags(ios::fixed);
        cout<<setprecision(6)<<setw(2)<<k<<" "<<root.front()<<" "<<f<<" "<<setw(2)<<c<<
            endl;
        if(factor.size() != 1)
        {
            factor.pop();
        }
        if(count.size() != 1){
            count.pop();
        }
        root.pop();
        k++;
    }
}
```

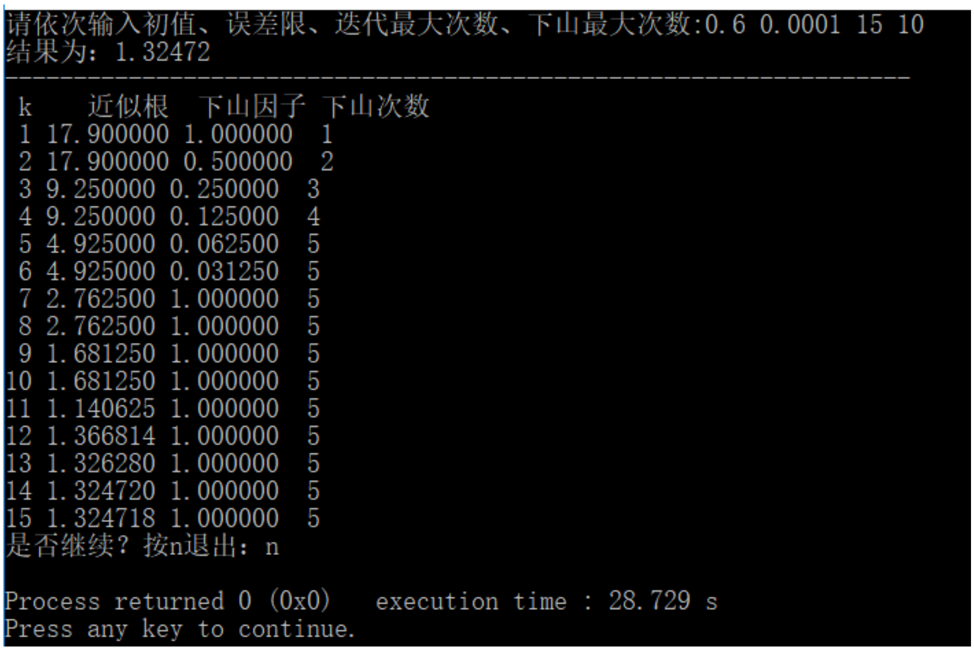

图 1.5: 牛顿下山法运行结果

1.4.2 弦截法

牛顿法要计算 $f'(x)$，现用 $f(x)$ 的值近似 $f'(x)$，为了避开导数的计算，认为切线斜率近似等于割线斜率。即改用差商 $\frac{f(x_k)-f(x_0)}{x_k-x_0}$ 来替换 Newton 公式中的导数 $f'(x_k)$，即得到如下离散化形式的牛顿公式：

$$f'(x_k) \approx \frac{f(x_k) - f(x_0)}{x_k - x_0} \tag{1.49}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_0)}(x_k - x_0) \tag{1.50}$$

迭代函数为：

$$\varphi(x) = x - \frac{f(x)}{f(x) - f(x_0)}(x - x_0) \tag{1.51}$$

单点弦截法为线性收敛。

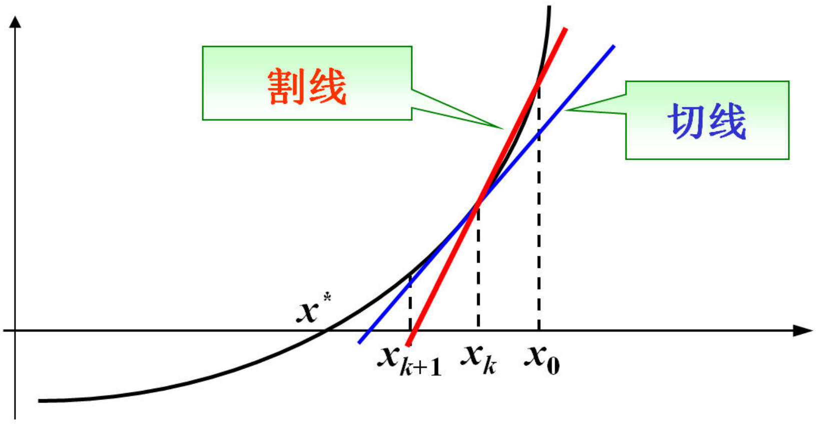

图 1.6: 弦截法几何意义

值得注意的是弦截法 (1.51) 仅为线性收敛。

1.4.3 快速弦截法

快速弦截法也称为两点弦截法，和弦截法类似，快速弦截法也认为切线斜率近似等于割线斜率。为了提高弦截法的收敛速度，改用差商 $\frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$ 来替代牛顿公式中的导数，得到如下的迭代公式：

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \tag{1.52}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})}(x_k - x_{k-1}) \tag{1.53}$$

这种迭代方法称为快速弦截法。快速弦截法虽然提高了收敛速度，但在计算 x_{k+1} 时需要用到前两步的信息 x_k 和 x_{k-1}，因此这种迭代方法称为两步法。

快速弦截法需要 2 个初值 x_0 和 x_1，其收敛阶 1.618。

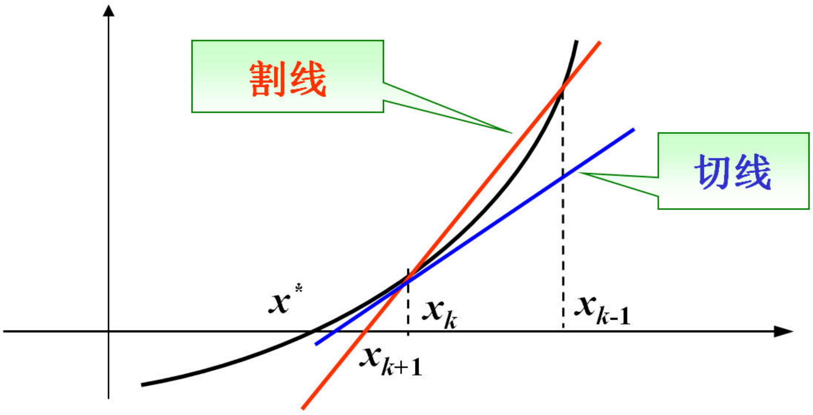

图 1.7: 快速弦截法

例 6. 使用 *Newton* 法和快速弦截法求方程 $xe^x-1=0$ 的根。

解：使用 Newton 法和快速弦截法，迭代公式分别如下：

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}=x_k-\frac{x_ke^{x_k}-1}{e^{x_k}+x_ke^{x_k}} \tag{1.54}$$

$$x_{k+1}=x_k-\frac{f(x_k)(x_k-x_{k-1})}{f(x_k)-f(x_{k-1})} \tag{1.55}$$

k	x_k	x_k
0	0.5	0.5
1	0.57102	0.6
2	0.56716	0.565315
3	0.56714	0.567094
4	0.56714	0.567143

表 1.5: 计算结果

1.5 埃特金加速算法

1.5.1 压缩影像原理

定理 5 (压缩影像原理). 设 $\varphi(x)$ 在 $[a,b]$ 上具有连续的一阶导数，且满足以下两项条件：

1. 封闭性条件　对于任意 $x\in[a,b]$ 总有 $\varphi(x)\in[a,b]$；
2. 压缩性条件　存在定数 $L:0\le L<1$，使对于任意 $x\in[a,b]$ 成立

$$|\varphi'(x)|\le L \tag{1.56}$$

则迭代过程 $x_{k+1}=\varphi(x_k)$ 对于任意初值 $x_0\in[a,b]$ 均收敛于方程 $x=\varphi(x)$ 的根 x^*，且有下列误差估计式

$$|x^*-x_k|\leq\frac{1}{1-L}|x_{k+1}-x_k| \tag{1.57}$$

$$|x^*-x_k|\leq\frac{L^k}{1-L}|x_1-x_0| \tag{1.58}$$

为了改进迭代法的有效性，提高其收敛速度，运用松弛技术，将每一步的新值和老值适当加权平均，以得到更高精度的改进值。

1.5.2 埃特金算法

设 x_k 是根 x^* 的某个近似值，用迭代公式校正一次得到

$$\overline{x}_{k+1}=\varphi(x_k) \tag{1.59}$$

假设 $\varphi'(x)$ 在所考察的范围内改变不大，其估计值为 L，则有

$$x^*-\overline{x}_{k+1}\approx L(x^*-x_k) \tag{1.60}$$

由此解出 x^* 为

$$x^*\approx\frac{1}{1-L}\overline{x}_{k+1}-\frac{L}{1-L}x_k \tag{1.61}$$

也就是说，如果将迭代值 $\overline{x}_{k+1}$ 与 x_k 加权平均，得到的新的值

$$x_{k+1}=\frac{1}{1-L}\overline{x}_{k+1}-\frac{L}{1-L}x_k \tag{1.62}$$

是个比 $\overline{x}_{k+1}$ 更好的近似值。由于 L 是导数的估计值，因此在计算上不方便。在实际应用中，为便于计算，将改进值再一次进行迭代，得到新的迭代值 $\widetilde{x}_{k+1}$，即有：

$$\widetilde{x}_{k+1}=\varphi(\overline{x}_{k+1}) \tag{1.63}$$

并有下式成立，

$$x^*-\widetilde{x}_{k+1}\approx L(x^*-\overline{x}_{k+1}) \tag{1.64}$$

将它与式 (1.60) 联立起来，消除 L，得到下式

$$\frac{x^*-\overline{x}_{k+1}}{x^*-\widetilde{x}_{k+1}}\approx\frac{x^*-x_k}{x^*-\overline{x}_{k+1}} \tag{1.65}$$

求解 x^*，得到最后解，如下：

$$x^*\approx\widetilde{x}_{k+1}-\frac{(\widetilde{x}_{k+1}-\overline{x}_{k+1})^2}{\widetilde{x}_{k+1}-2\overline{x}_{k+1}+x_k} \tag{1.66}$$

这样新得到的改进值，就不再含有导数的信息，但需要用两次迭代值 $\overline{x}_{k+1}$ 和 $\widetilde{x}_{k+1}$ 进行加工。

第一步：迭代

$$\overline{x}_{k+1} = \varphi(x_k) \tag{1.67}$$

第二步：再迭代

$$\widetilde{x}_{k+1} = \varphi(\overline{x}_{k+1}) \tag{1.68}$$

第三步：加速

$$x_{k+1} = \widetilde{x}_{k+1} - \frac{(\widetilde{x}_{k+1} - \overline{x}_{k+1})^2}{\widetilde{x}_{k+1} - 2\overline{x}_{k+1} + x_k} \tag{1.69}$$

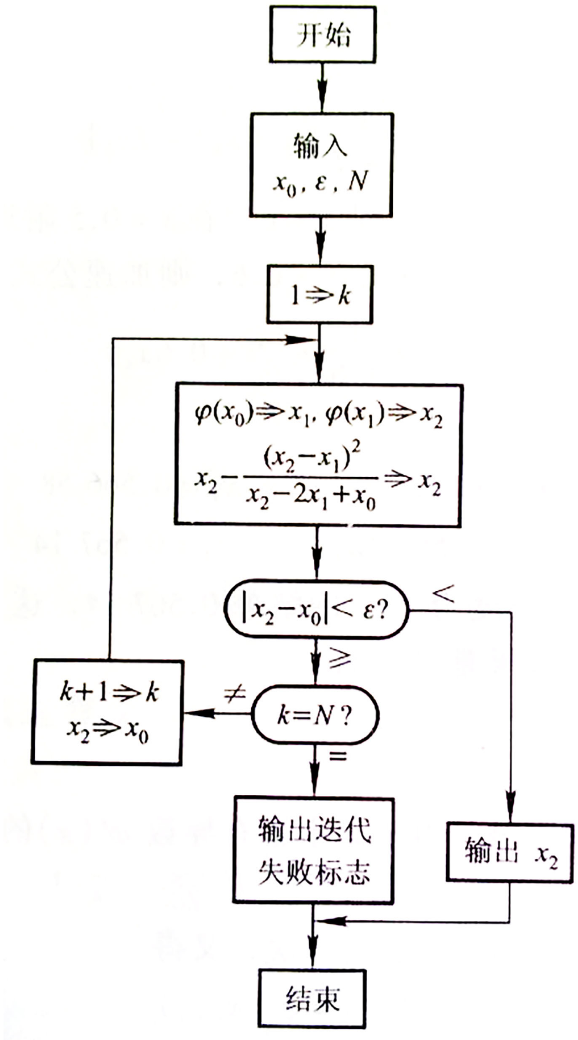

图 1.8: 埃特金算法流程图

```
#include<math.h>
#include <stdio.h>
double f(double x) {
    double y;
    y=pow(x-1,-1/2);
    return(y);
}

double point(double xo) {
    double x1,x2,xt,xs;
    x1=f(xo);
    printf("x1=%lf",x1);
    x2=f(x1);
    printf("\tx2=%lf",x2);
    xt=xo-(2*x1)+x2;
    printf("\txt=%lf",xt);
    xs=(xo*x2)-(x1*x1);
    printf("\txt=%lf",xs);
    xt=xs/xt;
    printf("\txt=%lf\n",xt);
    return xt;
}

double repass(double a,double eps) {
    double c1,c2;
    c1=point(a);
    printf("x1=%lf\n",c1);
    c2=point(c1);
    printf("x2=%lf\n",c2);
    while(fabs(c2-c1)>eps) {
        c1=point(c1);
        c2=point(c1);
        printf("x1=%lf\n",c1);
        printf(">>>>>>>>>x=%lf\n",c2);
    }
    return c2;
}

int main() {
    double a=1,x,eps=1e-8;
    double repass(double a,double eps);
    printf(" Please input x(0):\n");
    x=repass(a,eps);
    printf("\n x(*)=%lf\n",x);
    return 0;
}
```

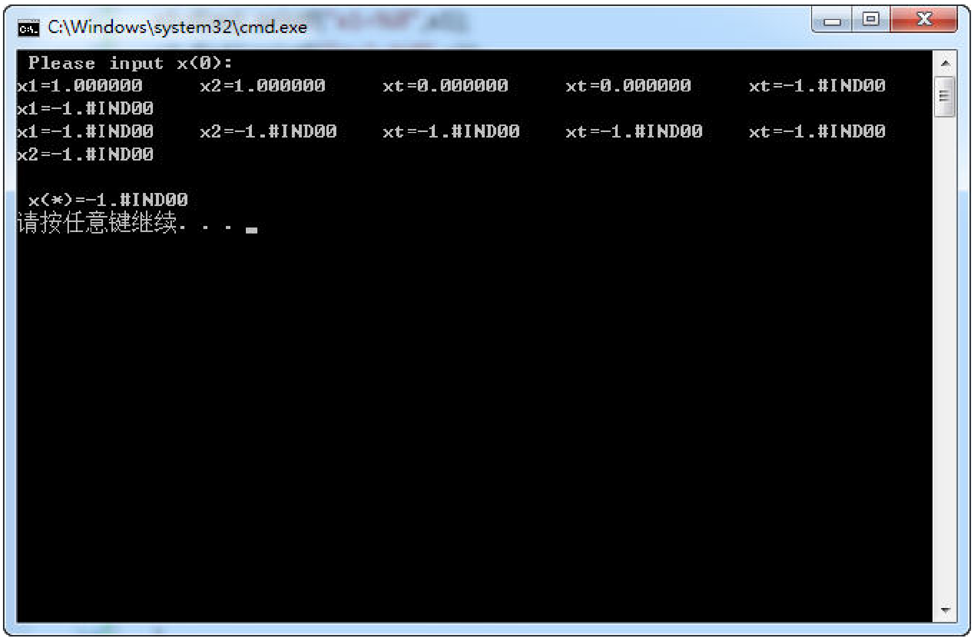

图 1.9: 埃特金算法运行结果

1.6 本章小结

本章主要介绍了方程求根的迭代法，重点讲解了开方法、牛顿法及其改进算法、压缩影像原理和 Aitken 加速方法等。

- 迭代法思想；
- 开方法；
- Newton 法；
- Newton 法的改进；
- 迭代过程的加速；

本章习题

P135-138 页，第 1，3，17，22 题。

参考文献

[1] 靳天飞，杜忠友等. *计算方法*. 北京：清华大学出版社, 2010.

[2] 李桂成. *计算方法 (第 3 版)*. 北京：电子工业出版社. 2019.8.

[3] 安妮 • 戈林鲍姆. 数值方法：设计、分析和算法实现. 北京：机械工业出版社, 2016.4.