非线性方程求解

一、历史背景

代数方程的求根问题是一个古老的数学问题。理论上，n 次代数方程在复数域内一定有 n 个根(考虑重数)。早在 16 世纪就找到了三次、四次方程的求根公式，但直到 19 世纪才证明大于等于 5 次的一般代数方程式不能用代数公式求解，而对于超越方程就复杂的多，如果有解，其解可能是一个或几个，也可能是无穷多个。一般也不存在根的解析表达式。因此需要研究数值方法求得满足一定精度要求的根的近似解。

二、算法及原理

本文设计的有三种算法如下

$$\begin{cases} \text{二分法} \\ \text{一般迭代法} \\ \text{牛顿迭代法} \end{cases}$$

1、二分法

原理：若 $f\in C[a,\ b]$，且 $f(a)\cdot f(b)<0$，则 f 在 $(a,\ b)$ 上至少有一实根。

基本思想：逐步将区间分半，通过判别区间端点函数值的符号，进一步搜索有根区间，将有根区间缩小到充分小，从而求出满足给定精度的根的近似值。

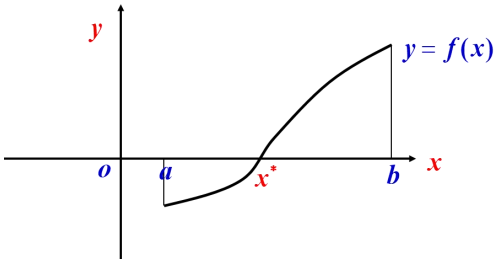

图 1 二分法几何意义

二分法的算法描述：

给定区间$[a, b]$ ，求 $f(x)=0$ 在该区间上的根 x.

输入: a 和 b; 容许误差 TOL; 最大对分次数 N_{max}.

输出: 近似根 x.

Step 1 Set $k = 1$;

Step 2 Compute x=f((a + b)/2);

Step 3 While ($k \le N_{max}$) do steps 4-6

Step 4 If |x| < TOL , STOP; Output the solution x.

Step 5 If x*f(a)<0 , Set b=x;

Else　Set $a=x$;

Step 6　Set　k=k+1; Compute　x=f((a + b)/2);Go To Step 3 ;

Step 7　Output the solution of equation:　x; STOP.

2、一般迭代法

原理：根据原函数$f(x)$，构造迭代形式$x=g(x)$，若迭代函数$g(x)$收敛，则通过若干次迭代可得到$f(x)=0$的近似解。

基本思想：迭代法是一种逐步逼近的方法，首先给出一个粗糙的初始值点，利用同一个迭代公式，反复计算，最终能得到一个较为准确的近似解。

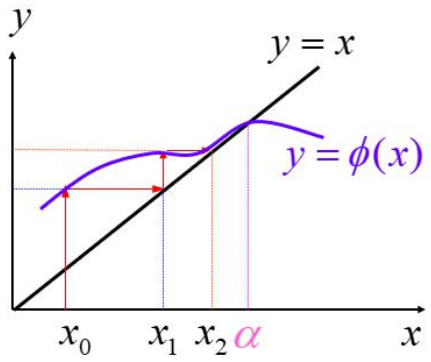

图 2　一般迭代法几何意义

算法描述：

给定初始近似值 x_0 ，求 $x=g(x)$ 的解.

输入: 初始近似值 x_0; 容许误差 *TOL*;

最大迭代次数 N_{max}.

输出: 近似解 x 或失败信息.

Step 1　Set　$i=1$;

Step 2　While ($i \le N_{max}$)　do　steps 3-6

Step 3　Set　$x=g(x_0)$;　/*　计算 x_i */

Step 4　If　$|x-x_0|<TOL$　then　Output (x);　/*成功*/

STOP;

Step 5　Set　i ++;

Step 6　Set　$x_0=x$;　/*　更新 x_0 */

Step 7　Output (The method failed after N_{max} iterations);　/*不成功 */

STOP.

3、牛顿迭代法

原理：若令一般迭代法中的$g(x)=x-\dfrac{f(x)}{f'(x)}$，则为牛顿迭代法，通过若干次迭代可得到

$f(x)=0$的近似解。

基本思想：基于一般迭代法，对迭代函数进行优化，使迭代过程加速。

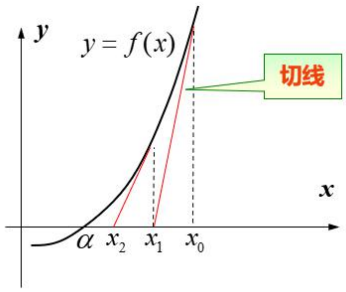

图 3 牛顿迭代发几何意义

算法描述：同一般迭代法，这里不再赘述。

三、代码实现

1、参数类 Options

```
/********Options参数详解************
    a               左区间
    b               右区间
    x0              x的初始值
    error_limit     误差限
    cycle_limit     最大循环次数
    h               求导时的步长
    show()          参数显示
 ****************************/
class Options {
public:
    double a = 0, b = 1;
    double x0 = 0;
    double error_limit = 0.000000001;
    double h = 0.0000001;
    int cycle_limit = 1000;

    void show();
};
```

2、二分法实现

```
double my_dichotomy(Options &opt, func_p func){
    double a = opt.a;
    double b = opt.b;
    double error_limit = opt.error_limit;
    int cycle_limit = opt.cycle_limit;
    double middle;
    double last_middle;
    int i = 0;

    if (func(a) * func(b) < 0) {
        middle = (a + b) / 2;
        last_middle = a;
        while (fabs(middle - last_middle) > error_limit) {
            func(middle)* func(a) > 0 ? a = middle : b = middle;
            last_middle = middle;
            middle = (a + b) / 2;
            i++;
            if (i > cycle_limit) {
                return -1.0;
            }
        }
        return middle;
    }
}
```

3、一般迭代法实现

```
double my_iteration(Options &opt, func_iteration_p func_interation) {
    double x0 = opt.x0;
    double error_limit =opt.error_limit;
    int cycle_limit = opt.cycle_limit;
    int i = 0;
    double x;

    while (i < cycle_limit) {
        x = func_interation(x0);
        if (fabs(x - x0) < error_limit) {
            return x;
        }
        i++;
        x0 = x;
    }
    return -1.0;
}
```

4、牛顿迭代法实现

```
double my_newton(Options &opt, func_p func){
    double error_limit = opt.error_limit;
    int cycle_limit = opt.cycle_limit;
    double x0 = opt.x0;
    double h = opt.h;
    int i = 0;
    double x;

    while (i < cycle_limit) {
        x = x0 - func(x0) / (func(x0 + h) - func(x0 - h)) / 2*h;
        if (fabs(x - x0) < error_limit) {
            return x;
        }
        i++;
        x0 = x;
    }
    return -1.0;
}
```