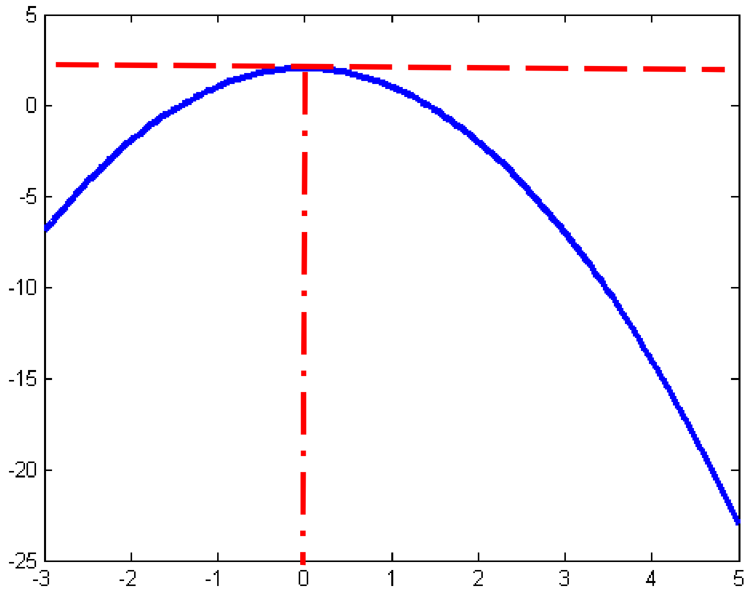
第2章 非线性方程的数值解法

2.1 背景

许多问题可以转化为方程求解

$$f(x)=0$$

例1： 求函数的最大值 $F(x)$



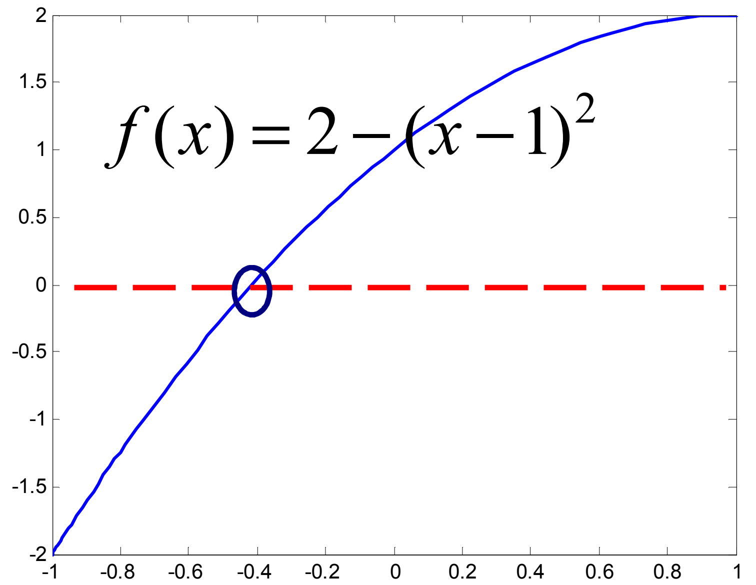
$$\Rightarrow \quad f(x)=F'(x)=0$$

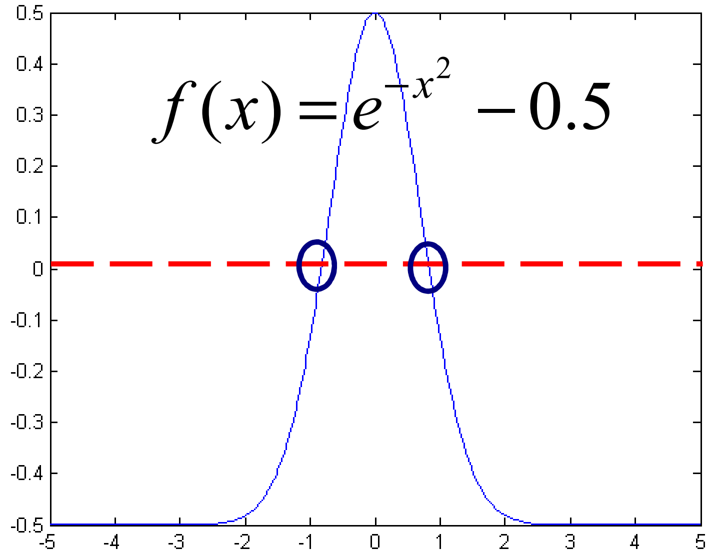
工程优化

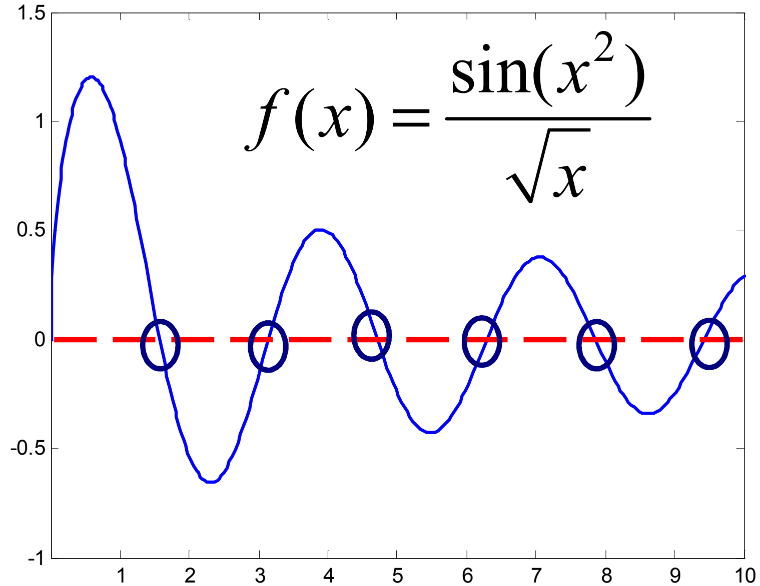
$$f(x) = 0$$

一般都是非线性的，

甚至无法写出表达式

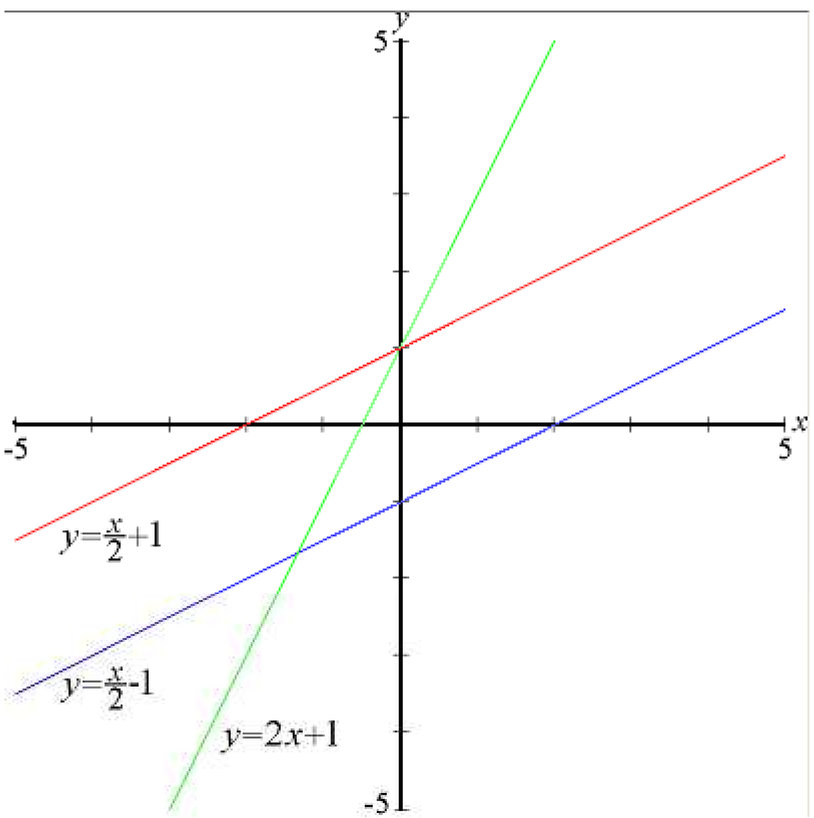






线性与非线性

$$f(x) = ax + b$$



性质：

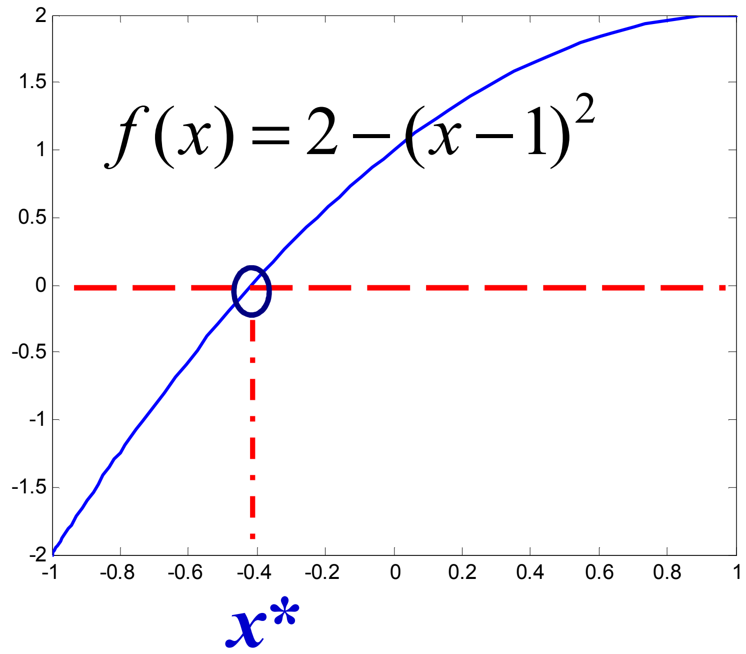
$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(x_2) - f(x_0)}{x_2 - x_0}$$

线性函数之和也是线性的（叠加性）

$$f_1(x) + f_2(x)$$

傅里叶关系式 $q(T) = k\nabla T$

$$f(x)=0$$



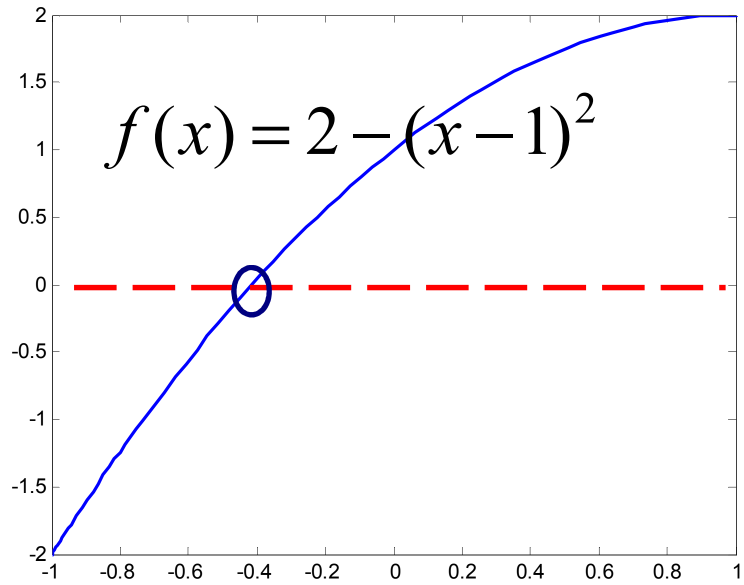
满足方程的 x 值通常叫做 方程的 根 或 解，也叫函数 $f(x)$ 的零点。

方程求根的三个步骤：

1. 根的存在性

2. 根的隔离

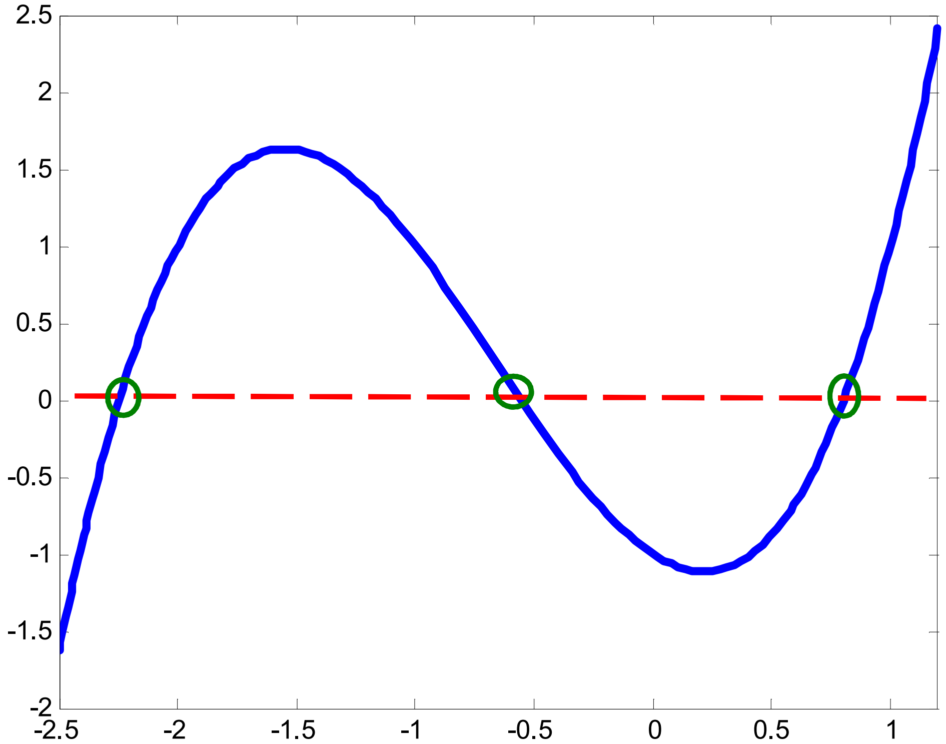
3. 根的精确化



根的存在定理(零点定理)：

$f(x)$为$[a, b]$上的连续函数，若 $f(a)\cdot f(b)<0$，则$[a, b]$中至少有一个实根。如果$f(x)$在$[a, b]$上还是单调递增或递减的，则$f(x)=0$仅有一个实根。

$[a,b]=[-1,1]$ $\qquad f(a)=-2$ $\qquad f(b)=2$



$$[a,b]=[-1.5,1.2] \qquad f(a)<0 \qquad f(b)>0$$

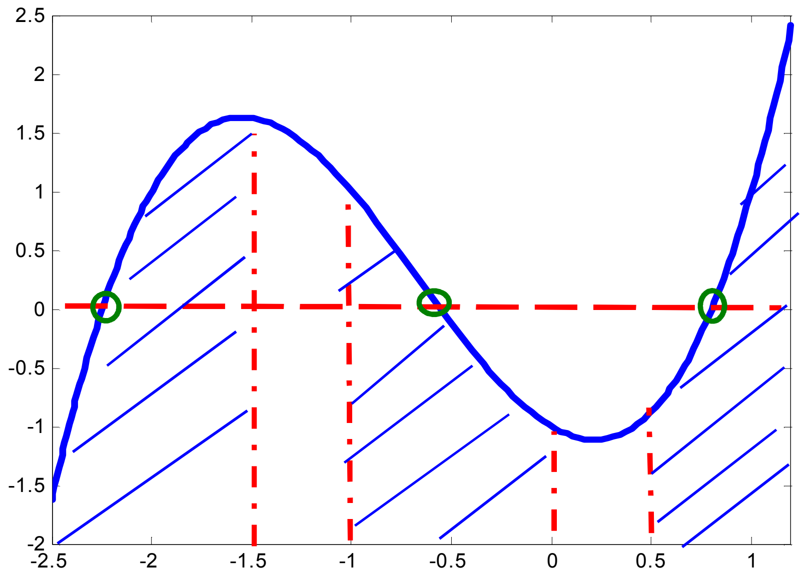
但有多个根

根的隔离

在用近似方法时，需要知道方程的根所在区间。若区间 $[a, b]$ 含有方程 $f(x)=0$ 的根，则称$[a, b]$为 $f(x)$ 的**有根区间**；若区间 $[a, b]$ 仅含方程 $f(x)=0$ 的一个根，则称 $[a, b]$ 为 $f(x)$ 的一个**隔根区间**。

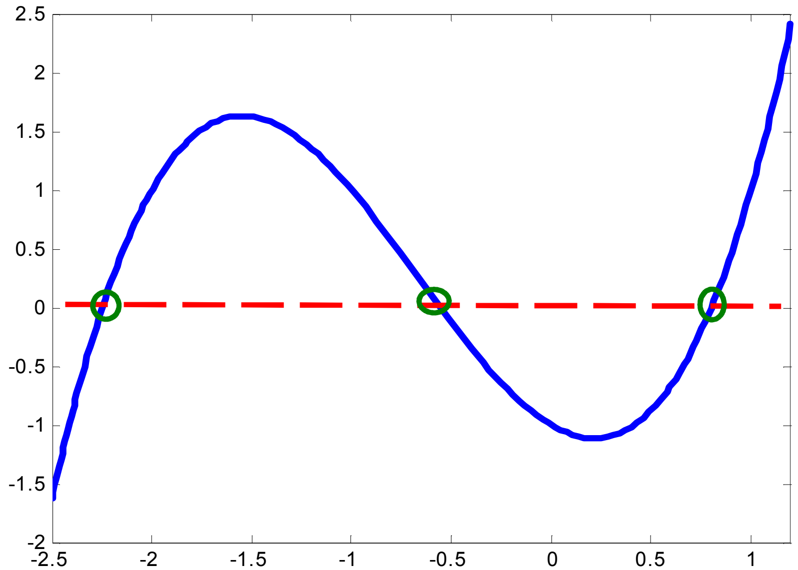
$[-2.5,\ 1.2]$　　有根区间

$$\left.\begin{array}{l}[-2.5,\ -1.5]\\ [-1,\ 0]\\ [0.5,\ 1.2]\end{array}\right\}\ \text{隔根区间}$$



根的精确化

对根的近似值逐步提高精度，使之满足一定的要求。



数值方法求根的目标！

根的性质 $f(x)=0$

即使[a, b]只含有一个根 x^*，这个根也可能分为：

单根： $f(x)=(x-x^*)\varphi(x) \qquad \varphi(x^*)\neq 0$

重根： $f(x)=(x-x^*)^n\varphi(x) \qquad \varphi(x^*)\neq 0$

判断方法：

$$f(x^*)=f'(x^*)=f''(x^*)=\ldots=f^{(n-1)}(x^*)=0,$$

$$f^{(n)}(x^*)\neq 0$$

⇨ **x^*是 n 重根**

本章只考虑单根情况

主要考虑三种方法：

1. 二分法

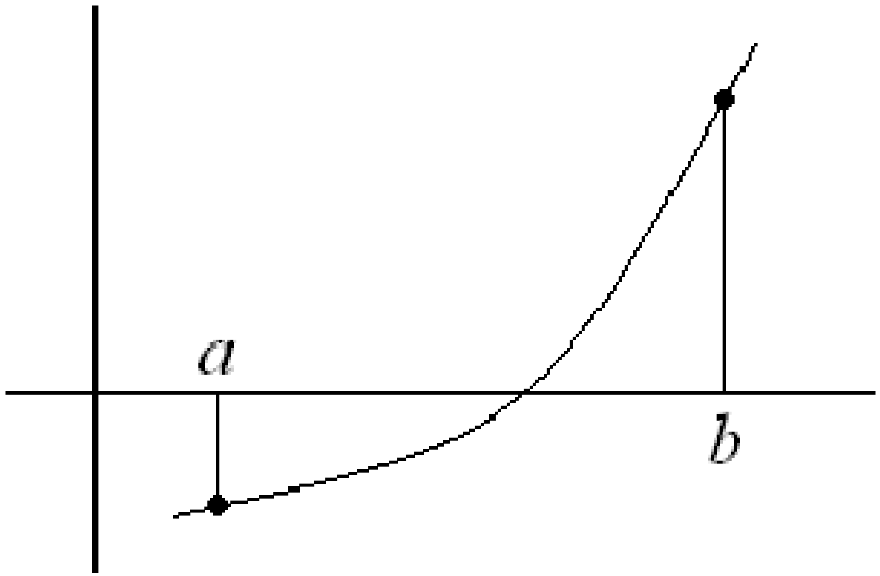
2. 简单迭代法及加速迭代法

3. Newton法

2.2 二分法

原理：

根的存在定理： $f(x)$为$[a, b]$上的连续函数，若 $f(a)\cdot f(b)<0$，则$[a, b]$中必有一个实根。

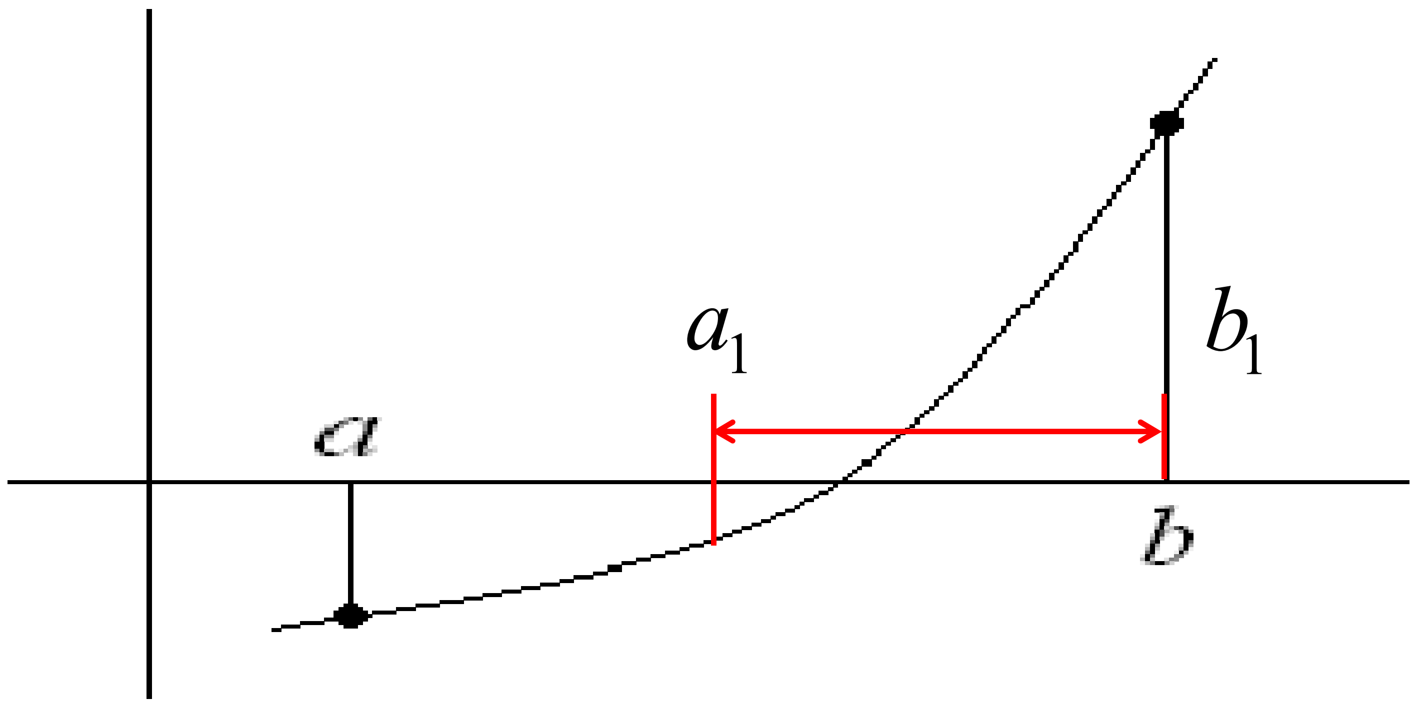


- 二分法步骤

1. 取区间中点 $x_0=(a+b)/2$。

若 $f(a)\cdot f(x_0)<0$, 则根在 $[a_1, b_1]=[a, x_0]$内；

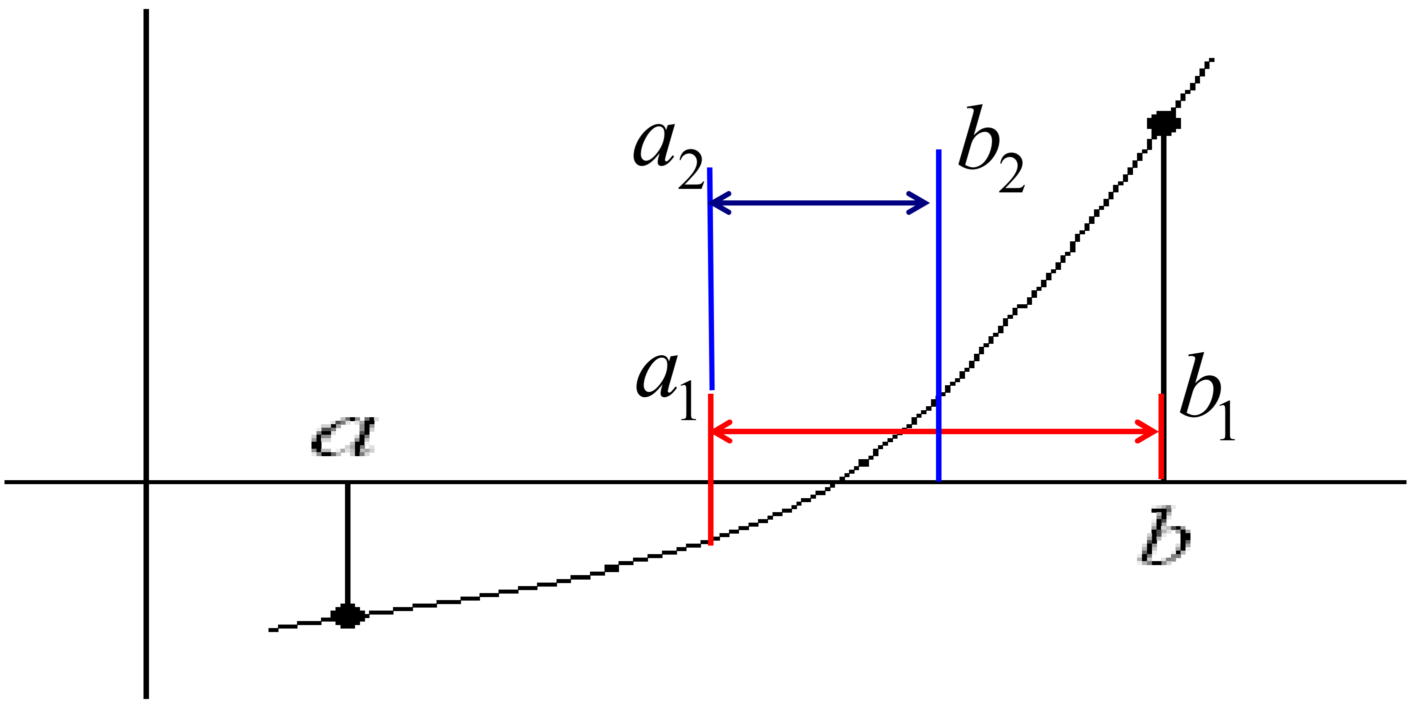
否则，根在$[a_1, b_1]=[x_0，b]$内。



2. 取 $[a_1, b_1]$区间中点 $x_1=(a_1+b_1)/2$。

若 $f(a_1)\cdot f(x_1) < 0$, 则根在 $[a_2, b_2] = [a_1, x_1]$内；

否则，根在$[a_2, b_2] = [x_1，b_1]$内。

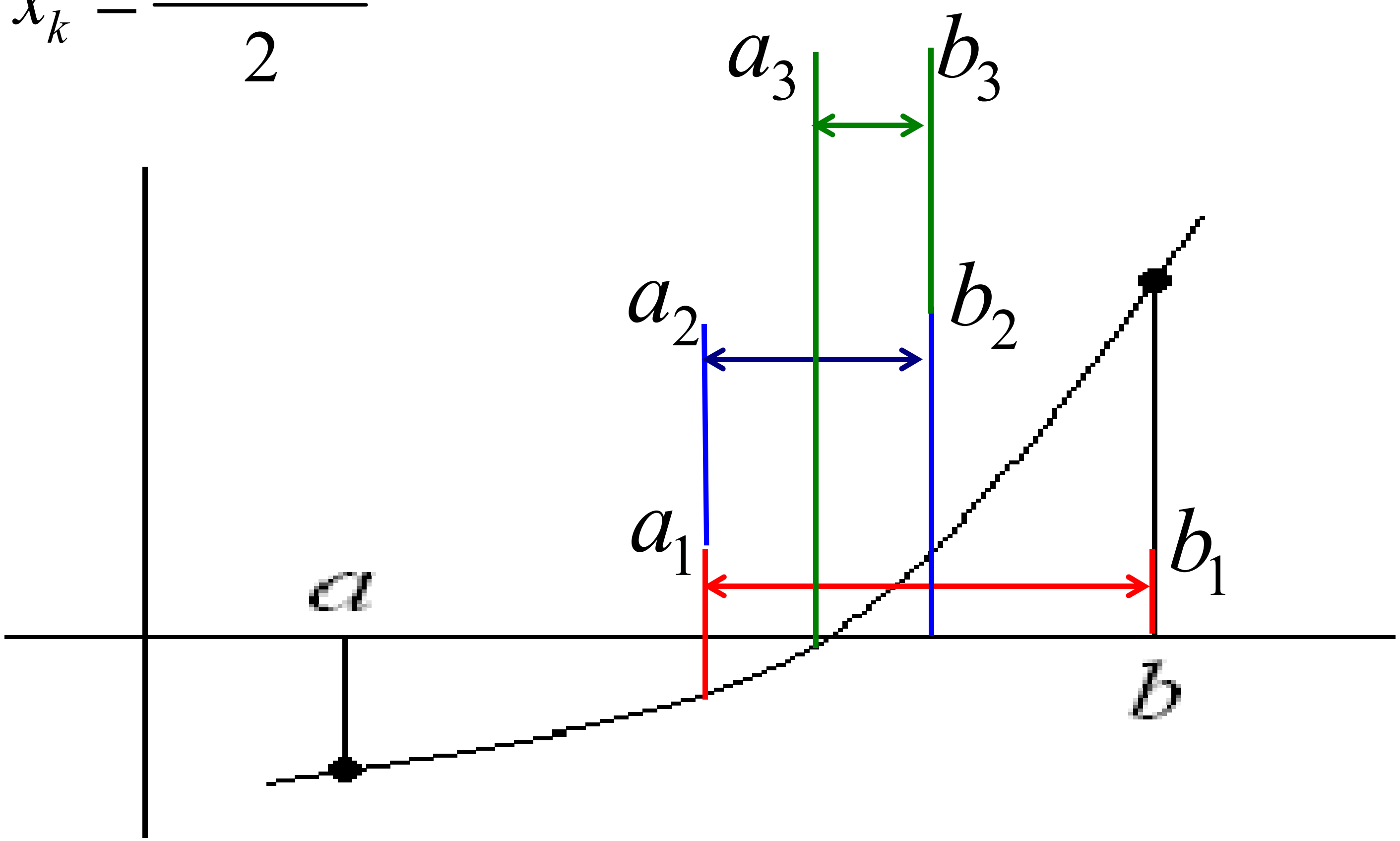


3. 重复以上步骤，得到一些列有根区间

$$[a_k, b_k] \subset [a_{k-1}, b_{k-1}] \subset \cdots \subset [a_2, b_2] \subset [a_1, b_1] \subset [a, b]$$

当 $b_k - a_k < \varepsilon$ 时，取中点为解的近似值

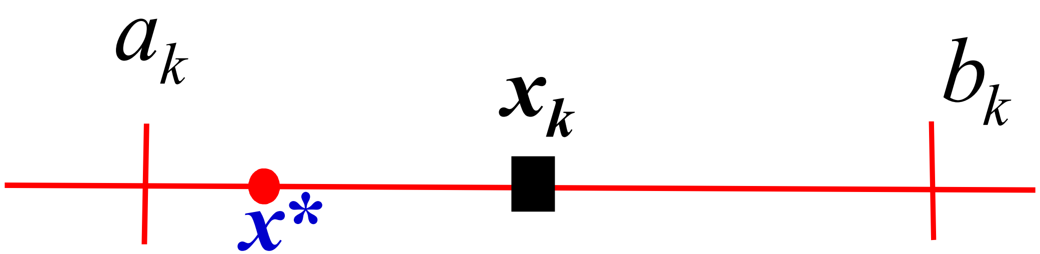
$$x^* \approx x_k = \frac{a_k + b_k}{2}$$



- 二分法分析

$$b_1 - a_1 = \frac{b-a}{2} \qquad b_2 - a_2 = \frac{b_1 - a_1}{2} = \frac{b-a}{2^2}$$

$$b_k - a_k = \frac{b-a}{2^k}$$



$$|x^* - x_k| \le \frac{b_k - a_k}{2} = \frac{b-a}{2^{k+1}}$$

- 二分次数

$$|x^* - x_k| \le \frac{b-a}{2^{k+1}} < \varepsilon \quad \Rightarrow \quad \ln\left(\frac{b-a}{2^{k+1}}\right) < \ln\varepsilon$$

$$\Rightarrow \quad \ln(b-a) - (k+1)\ln 2 < \ln\varepsilon$$

$$k > \frac{\ln(b-a) - \ln(2\varepsilon)}{\ln 2}$$

$b-a=1$:

$$\varepsilon = 10^{-4}, \quad k > 12.2877$$

$$\varepsilon = 10^{-5}, \quad k > 15.6096$$

$$\varepsilon = 10^{-6}, \quad k > 18.9316$$

$$\varepsilon = 10^{-10}, \quad k > 32.2193$$

例 2. 用二分法求 $f(x)=3x^2+2x-10=0$ 在 [1,2] 间的一个根，精度1.0e-6

$$x^* = (\sqrt{31}-1)/3 \approx 1.522588120943341$$

```c
# include <stdio.h>
float f(float x)
{
  return (3*x*x+2*x-10);
}

float bisection(float a, float b, float eps)
{
  float xc;
  int k=0;
  while (b-a>eps)
   {
     xc=0.5*(a+b);
     printf("k=%d   x=%f  f(x)=%f  \n", k, xc, f(xc));
     if(f(a)*f(xc)<0)
          b=xc;
     else
          a=xc;
      k=k+1;
   }
 }
```

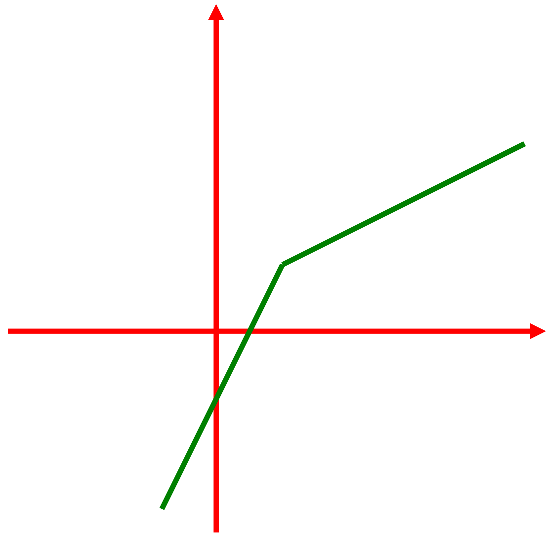
```c
void main()
{
  bisection(1, 2, 1.0e-6);
}
```

```
k=0    x=1.500000  f(x)=-0.250000
k=1    x=1.750000  f(x)=2.687500
k=2    x=1.625000  f(x)=1.171875
k=3    x=1.562500  f(x)=0.449219
k=4    x=1.531250  f(x)=0.096680
k=5    x=1.515625  f(x)=-0.077393
k=6    x=1.523438  f(x)=0.009460
k=7    x=1.519531  f(x)=-0.034012
k=8    x=1.521484  f(x)=-0.012287
k=9    x=1.522461  f(x)=-0.001416
k=10    x=1.522949  f(x)=0.004021
k=11    x=1.522705  f(x)=0.001302
k=12    x=1.522583  f(x)=-0.000057
k=13    x=1.522644  f(x)=0.000623
k=14    x=1.522614  f(x)=0.000283
k=15    x=1.522598  f(x)=0.000113
k=16    x=1.522591  f(x)=0.000028
k=17    x=1.522587  f(x)=-0.000014
k=18    x=1.522589  f(x)=0.000007
k=19    x=1.522588  f(x)=-0.000004
```

- 二分法的优缺点

1. 计算过程简单

2. 对函数要求低（连续即可）



3. 收敛速度较慢

4. 函数值计算中只使用其正负号信息，未充分利用

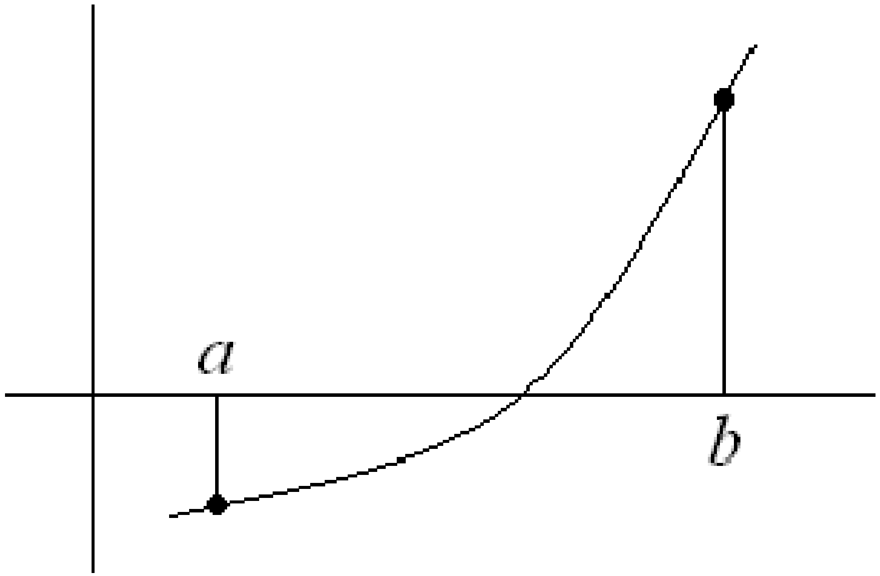
更高效率的方法？

2.3 迭代法

精髓

“简单”的重复

逐步逼近



$$x_0 \to x_1 \to x_2 \to x_3 \to \cdots\cdots \to x_k \to \cdots\cdots$$

精益求精

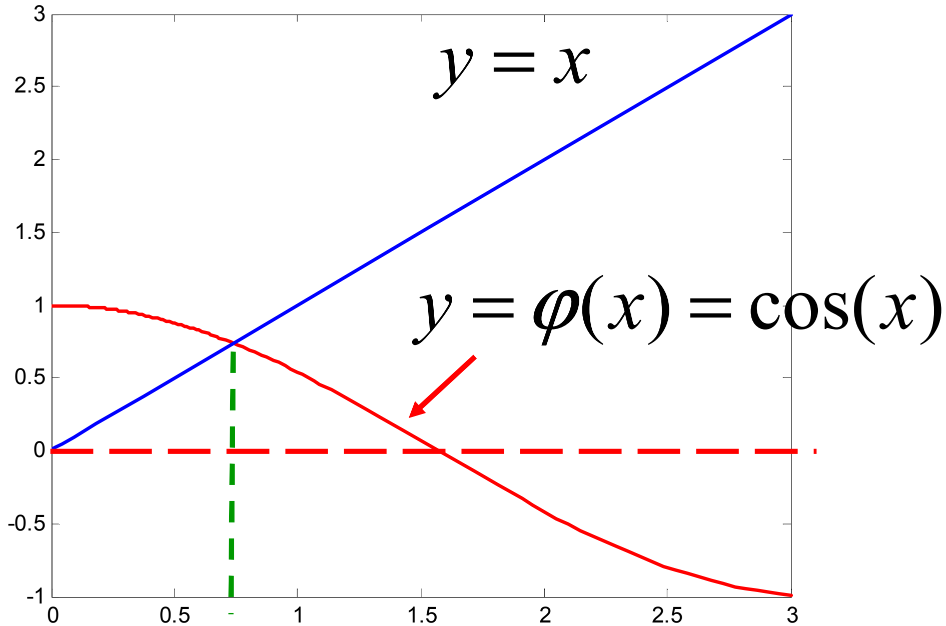
简单迭代法

基本思想： $f(x)=0 \Leftrightarrow \underline{x=\varphi(x)}$

不动点方程

迭代公式：

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



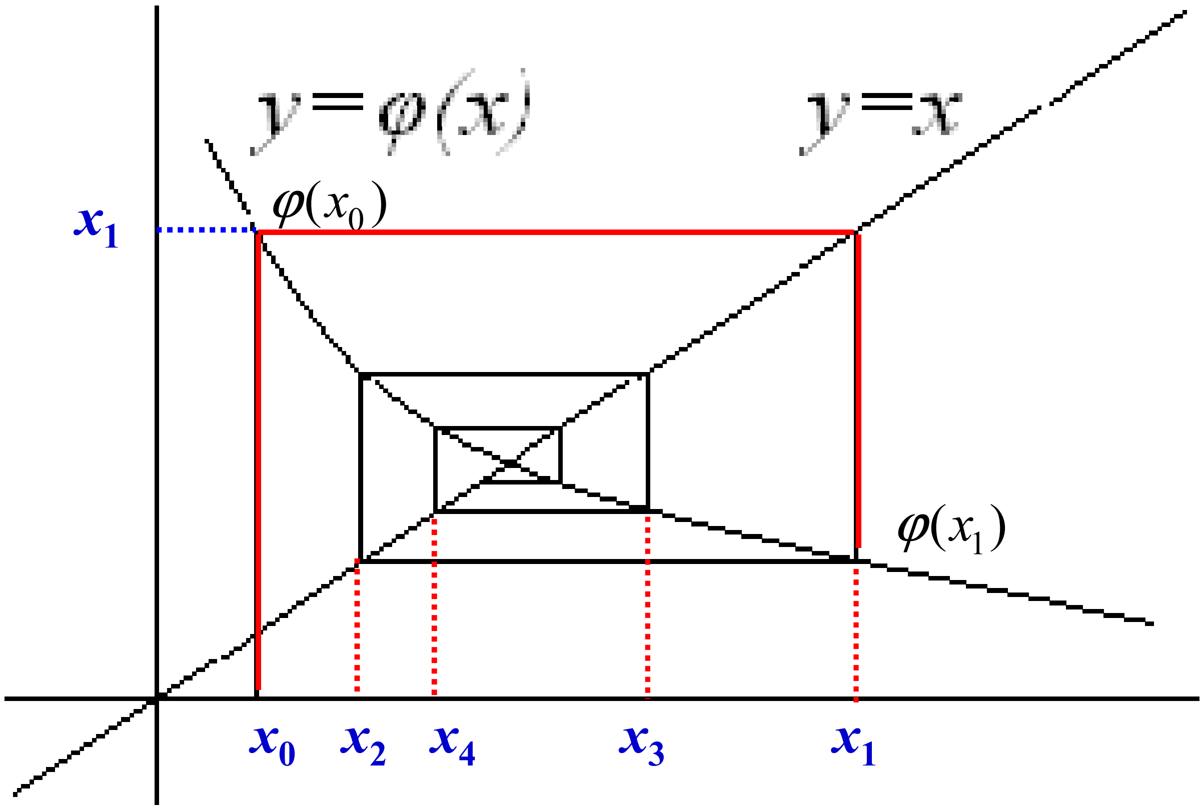
简单迭代法又称为不动点迭代法

如果迭代序列收敛，则

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

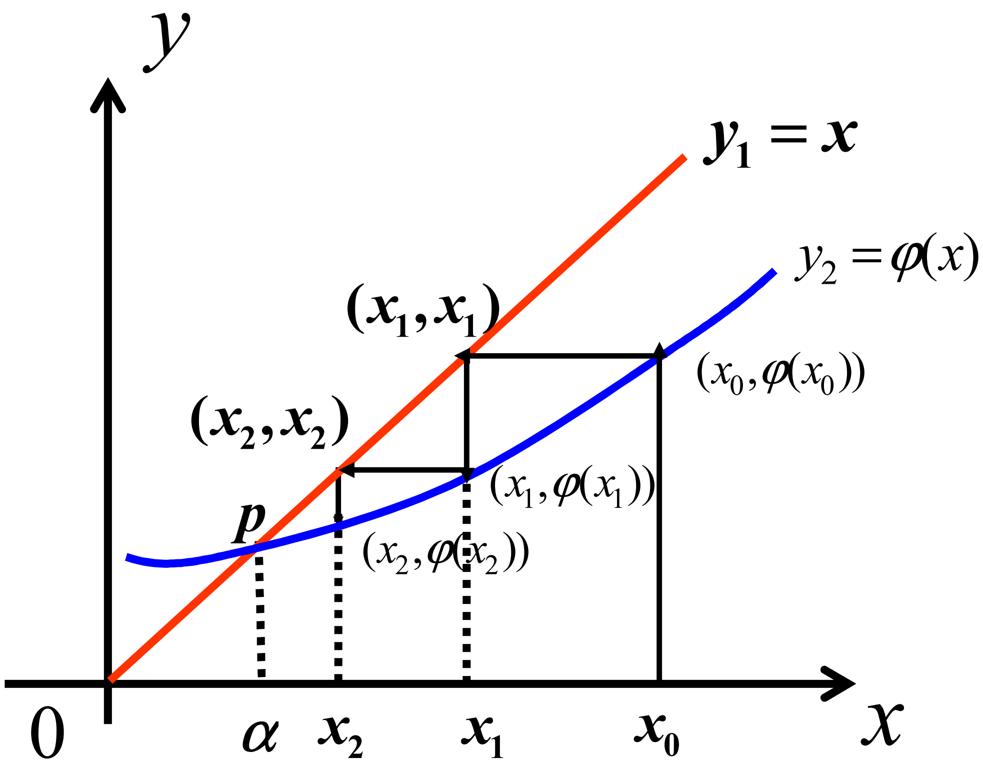
几何解释

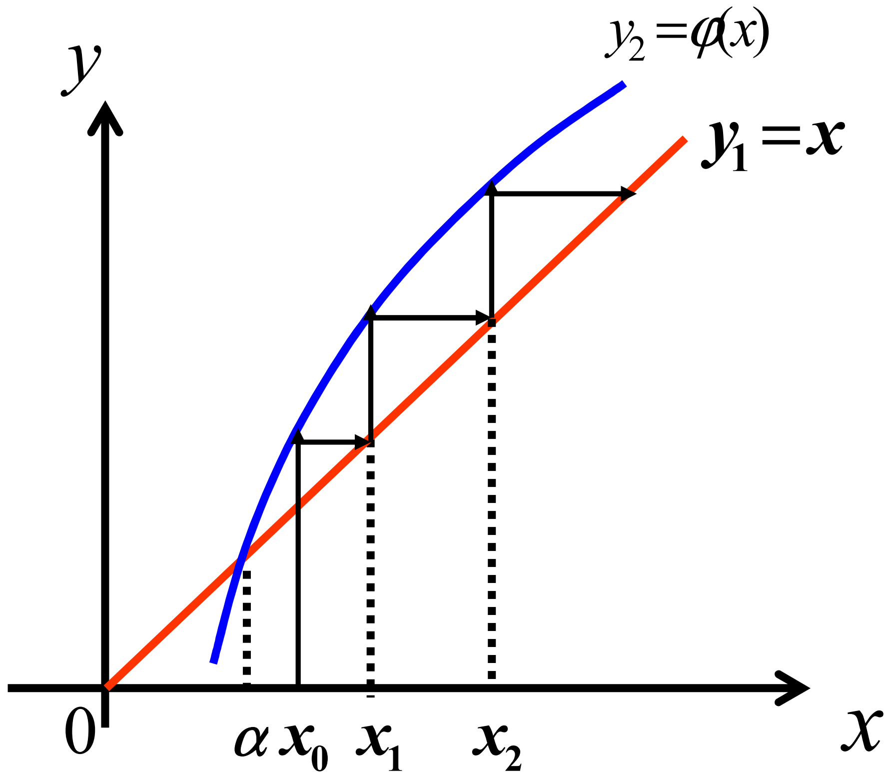
$$x_{k+1} = \varphi(x_k)$$

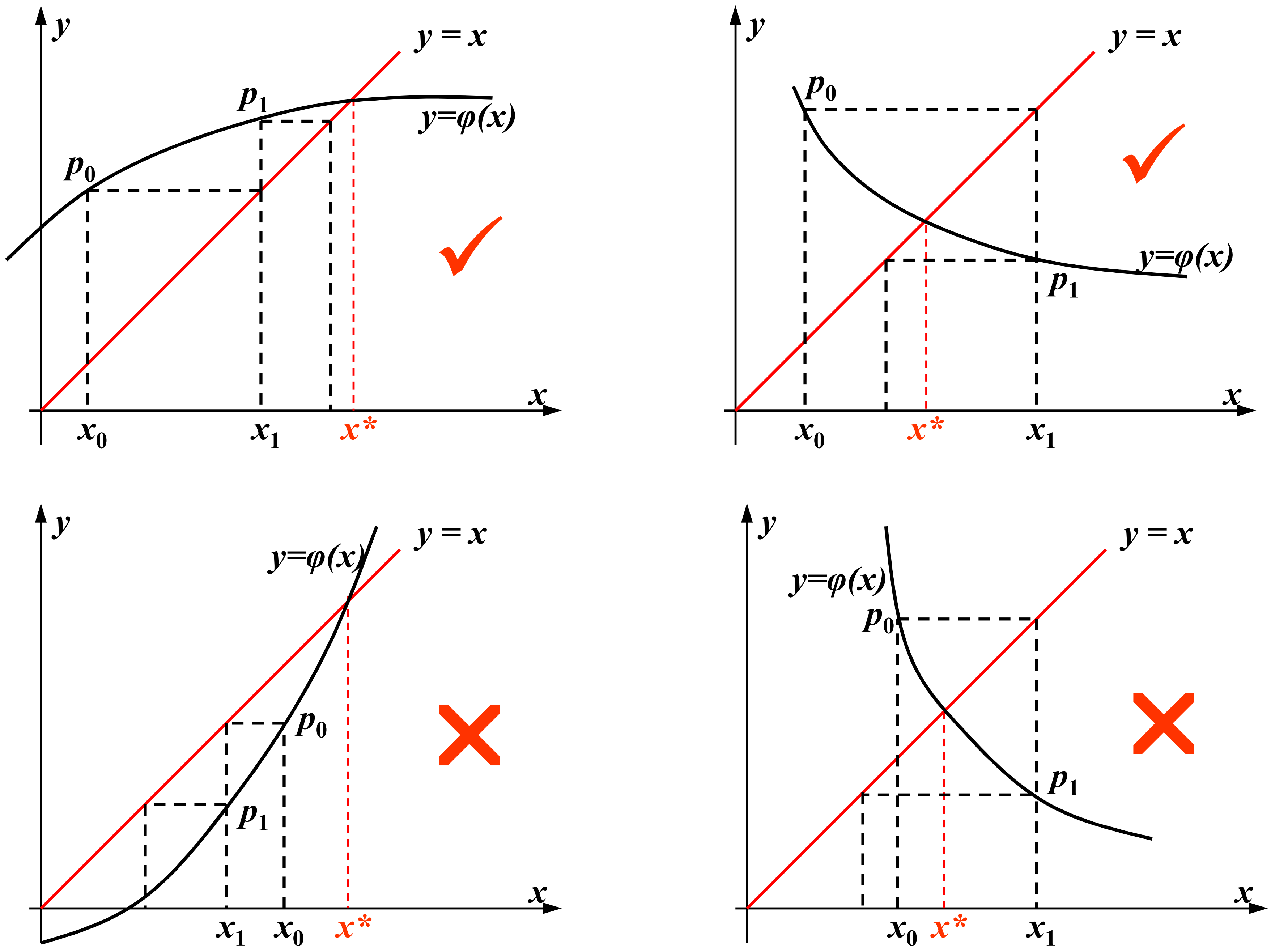


收敛及发散

$y_1 = x$, $y_2 = \varphi(x)$, 交点横坐标即为方程的根






y
y = x
p1
p0
y=φ(x)
x
x0
x1
x*
y
y = x
p0
y=φ(x)
p1
x
x0
x*
x1
y
y = x
y=φ(x)
p0
p1
x
x1
x0
x*
y
y = x
y=φ(x)
p0
p1
x
x0
x*
x1

收敛条件

$$x_0 = a$$
$$x_{k+1} = \varphi(x_k), \quad k = 1, 2, 3, \ldots\ldots$$

$$x_0 \to x_1 \to x_2 \to x_3 \to \cdots\cdots \to x_k \to \cdots\cdots$$ 收敛吗？

特例： $x_{k+1} = ax_k + b$ $\qquad \varphi(x) = ax + b$

|a| < 1：收敛；否则，发散

启示：收敛性与迭代函数的导数有关 $\varphi'(x) = a$

$$|\varphi'(x)| < 1$$ ？

定理1（收敛定理）

(I) 当 $x \in [a, b]$ 时， $\varphi(x) \in [a, b]$;

(II) 对 $\forall x \in [a, b]$, 有 $|\varphi'(x)| \le L < 1$ 成立。

精度方面

a. $$|x^* - x_k| \le \frac{L}{1-L} |x_k - x_{k-1}|$$

b. $$|x^* - x_k| \le \frac{L^k}{1-L} |x_1 - x_0| \qquad (k = 1, 2, \dots)$$

Proof:

$$|x^*-x_k|=|x^*-x_{k+1}+x_{k+1}-x_k|$$

$$\leq|x^*-x_{k+1}|+|x_{k+1}-x_k|$$

$$=|\varphi(x^*)-\varphi(x_k)|+|\varphi(x_k)-\varphi(x_{k-1})|$$

$$=|\varphi'(\xi)(x^*-x_k)|+|\varphi'(\eta)(x_k-x_{k-1})|$$

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

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

$$(1-L)|x^*-x_k| \leq L|x_k - x_{k-1}|$$

$$= L|\varphi(x_{k-1}) - \varphi(x_{k-2})| = L|\varphi'(\xi)(x_{k-1} - x_{k-2})| \leq L^2 |x_{k-1} - x_{k-2}|$$

$$\cdots$$

$$\leq L^k |x_1 - x_0|$$

定理2（局部收敛定理）

（I） 在根x^*附近， $\varphi'(x)$ 连续；

（II） $|\varphi'(x)|<1$

例： 迭代公式 $$x_{k+1}=\frac{x_k}{2}+\frac{1}{x_k}$$ 在$x^*=\sqrt{2}$ 附近局部收敛

收敛速度

求方程$f(x) = x^2 + x - 4 = 0$在$[1,2]$间的根。

构造如下三个格式：

（1）$x_{n+1} = 4 - x_n^2$

（2）$x_{n+1} = \dfrac{4}{1 + x_n}$

（3）$x_{n+1} = x_n - \dfrac{x_n^2 + x_n - 4}{1 + 2x_n}$

1.5	**1.5**	**1.5**
1.75	**1.6000**	**1.5625**
0.9375	**1.5385**	**1.5616**
3.1211	**1.5758**	**1.5616**
-5.7412	**1.5529**	**1.5616**
-28.9617	**1.5668**	**…**
…	**1.5583**	
	1.5635	
	1.5604	
	1.5623	
	1.5611	
	1.5614	
	1.5617	
	1.5616	
	1.5616	
	1.5616	
	1.5616	
	…	

迭代法收敛速度

定义 设数列$\{x_k\}$收敛于x^*,令误差$e_k = x^* - x_k$,如果存在某个实数 $p \geq 1$ 及常数 C ，使

$$\lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^p} = C$$

则称数列$\{x_k\}$为p 阶收敛

显然， p越大，数列收敛的越快。所以，迭代法的收敛阶是对迭代法收敛速度的一种度量。

判断收敛阶的简单方法

根据迭代函数的各阶导数判断

$$\varphi'(x^*) = \varphi''(x^*) = \ldots = \varphi^{(p-1)}(x^*) = 0$$

$$\varphi^{(p)}(x^*) \neq 0$$

***p* 阶收敛**

（1）$x_{n+1} = \dfrac{4}{1+x_n}$

（2）$x_{n+1} = x_n - \dfrac{x_n^2 + x_n - 4}{1+2x_n}$

$$\varphi(x) = \frac{4}{1+x} \qquad \varphi'(x) = -\frac{4}{(1+x)^2} \neq 0 \qquad p=1$$

$$\varphi(x) = x - \frac{x^2 + x - 4}{1+2x} \qquad \varphi'(x) = \frac{(x^2 + x - 4)(1+2x)}{(1+2x)^2}$$

$$\varphi'(x^*) = 0 \qquad \varphi''(x^*) = \frac{2}{1+2x^*} \neq 0 \qquad p=2$$

迭代加速方法

- 基本思想

$$x_0 = a$$

$$x_{k+1} = \varphi(x_k), \quad k = 1, 2, 3, ...$$

$$x_0 \to x_1 \to x_2 \to x_3 \to \cdots\cdots \to x_k \to \cdots\cdots$$

改造迭代函数 $\phi(x) = \varphi(x) + \lambda[\varphi(x) - x]$ 使 得 $\phi'(x^*) \approx 0$

$$\lambda = \frac{\varphi'(x^*)}{1 - \varphi'(x^*)} \approx \frac{\varphi'(x_k)}{1 - \varphi'(x_k)}$$

简化加速方法（1） $\phi'(x^*) \approx L \longrightarrow \lambda = \frac{L}{1-L}$

$$x_{k+1} = \frac{1}{1-L}\varphi(x_k) - \frac{L}{1-L}x_k$$

L的信息来源于迭代函数的导数

简化加速方法（2）：直接使用加权平均—松弛方法

$$x_{k+1} = \omega_k \varphi(x_k) + (1-\omega_k)x_k$$

复杂加速方法：**Aitken方法—预估校正方法**

$$\bar{x}_{k+1} = \varphi(x_k)$$

$$\bar{\bar{x}}_{k+1} = \varphi(\bar{x}_k)$$

$$x_{k+1} = \bar{\bar{x}}_{k+1} - \frac{(\bar{\bar{x}}_{k+1} - \bar{x}_{k+1})^2}{\bar{\bar{x}}_{k+1} - 2\bar{x}_{k+1} + x_k}$$

与前面的加速方法思路不同，不包含导数信息

```c
#include <stdio.h>
#include <math.h>
#define MaxDepth 100 /*最大迭代深度*/
#define epsilion 1e-5
typedef double (*calfun) (double);

double f1(double x)
{
	return x*x*x-1;
}

int aitken(calfun fun,double x0,double *ans)
{
/*
    x0初始值
    x1,x2存放迭代的中间结果
*/
	int i;
	double x1,x2,y,z;
	x1=x0;
	for(i=0;i<MaxDepth;i++)
	{
	   y=fun(x1);
	   z=fun(y);
	   x2=z-((z-y)*(z-y)/(z-2*y+x1));
	   if(fabs(x2-x1)<1e-5)
	   {
		*ans=x2;
		return 1;
	   }
	   x1=x2;
	}
	printf("After %d repeate,no solved.\n",MaxDepth);
	return    0;
}
```

```
int main()
{
   double ans;
    if(aitken(f1,1.5,&ans))
    {
          printf("%lf\n",ans);
    }
    getche();
    return 0;
}
```