2 非线性方程

数值计算要解决的核心问题就是方程求解，包括非线性方程、非线性方程组和线性方程组。迭代是方程求解的基本方法。

如果一个函数 $f(x)$ 满足（1）可加性，$f(x+y)=f(x)+f(y)$，（2）齐次性，$f(\lambda x)=\lambda f(x)$，则称 $f(x)$ 是**线性函数**，否则就是**非线性函数**。相应地，如果 $f(x)$ 是线性函数，$f(x)=c$ 就是**线性方程**，其中 c 是常数，称 $f(x)=0$ 为齐次方程；如果 $f(x)$ 是非线性函数，$f(x)=c$ 就是**非线性方程**。如果 $f(x)$ 是多项式，则称 $f(x)=c$ 为**代数方程**或多项式方程。非线性方程可能非常复杂，见下图：

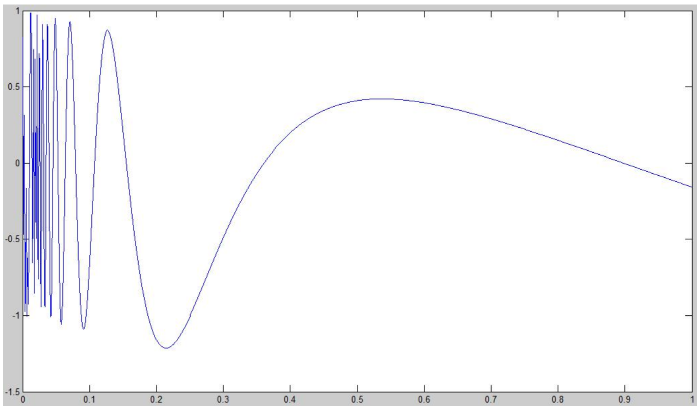


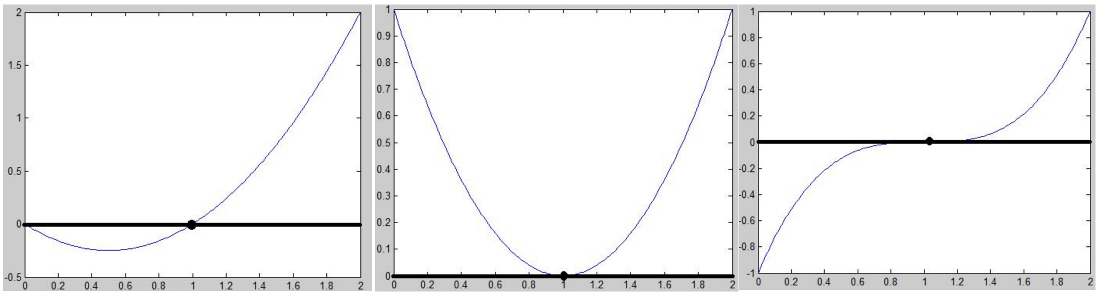
图 2.1 $f(x)=\sin\left(\dfrac{1}{x}\right)-x$ 在 $[0,1]$ 区间内的图像

一般记非线性方程为 $f(x)=0$，$f(x)$ 是超越函数或高次多项式。**非线性方程组**的一般形式是：$\begin{cases} f_1(x_1,x_2,\cdots,x_n) & = & 0 \\ f_2(x_1,x_2,\cdots,x_n) & = & 0 \\ & \cdots & \\ f_n(x_1,x_2,\cdots,x_n) & = & 0 \end{cases}$。方程的**解**亦称方程的**根**或函数的**零点**。根可能是实数或复数。若 $f(\alpha)=0, f'(\alpha)\neq 0,$ 则称 α

为**单根**；若 $f(\alpha)=f^{'}(\alpha)=\cdots=f^{(k-1)}(\alpha)=0$ 而 $f^{(k)}(\alpha)\neq 0$，则称 α 为 ***k* 重根**，见图 2.2。常见的求解问题有两类：

(1) 求解在给定范围内的**某个解，**

(2) 求解在给定范围内的**全部解**。



(a)单根　　　　(b)2 重根　　　　(c)3 重根

图 2.2 单根与重根

2.1 代数方程求根

塔塔利亚（Niccolò Tartaglia）三次方程求解方法。一元三次方程的一般形式是 $ax^3+bx^2+cx+d=0$，作一个线性变换（例如令 $x=y+h$，并适当选取 h）就可以把方程的二次项消去，所以只考虑形如 $x^3=px+q$ 的三次方程。假设方程的解 x 可以写成 $a-b$ 的形式，其中 a 和 b 待定，带入方程有 a^3-$3a^2b$+$3ab^2$-b^3=$p(a$-$b)$+q，得到 a^3-b^3=$(a$-$b)(p$+$3ab)$+q。根据待定系数法的思路，要找到这样的 a 和 b，满足 a^3-b^3=q 且 $3ab$+p=0。$a^3-b^3=q$ 的两边各乘以 $27a^3$，就得到 $27a^6-27a^3b^3$=$27qa^3$。由 p=$-3ab$ 可知 $27a^6$+p^3=$27qa^3$。这是一个关于 a^3 的二次方程，所以可以解得 a。进而可解出 b 和根 x。这个求解过程相当于在原始方程基础上，再增加两个变量 a、b 和两个方程 $x=a-b$ 及

$3ab+p=0$，构成一个非线性方程组，然后用一元二次方程求根公式直接求出a，再解出b和x。

直接用待定系数法也可以求解三次方程。对于三次方程的标准型$x^3=px+q$，可以假设其解形如：$x=\sqrt[3]{a}+\sqrt[3]{b}$，用待定系数法：

$$\left(\sqrt[3]{a}+\sqrt[3]{b}\right)^3=p\left(\sqrt[3]{a}+\sqrt[3]{b}\right)+q$$

$$a+b+3\sqrt[3]{a}\sqrt[3]{b}\left(\sqrt[3]{a}+\sqrt[3]{b}\right)=p\left(\sqrt[3]{a}+\sqrt[3]{b}\right)+q$$

令$a+b=q$，$3\sqrt[3]{a}\sqrt[3]{b}=p$即$ab=p^3/27$，求解二次方程：

$$y^2-qy+p^3/27=0$$

得到三次方程的解如下：

$$x=\sqrt[3]{\frac{q}{2}-\sqrt{\left(\frac{q}{2}\right)^2-\left(\frac{p}{3}\right)^3}}+\sqrt[3]{\frac{q}{2}+\sqrt{\left(\frac{q}{2}\right)^2-\left(\frac{p}{3}\right)^3}} \cdots\cdots\cdots\cdots\cdots\cdots(2.1)$$

注意此根式解存在的条件是$\left(\frac{q}{2}\right)^2-\left(\frac{p}{3}\right)^3\geq 0$。

费拉里（Ferrari L）四次方程解法。与三次方程一样，可以用一个线性变换消去四次方程中的三次项，故只考虑下面形式的一元四次方程：$x^4=px^2+qx+r$。下一步把等式的两边配成完全平方式。对于参数a，有$\left(x^2+a\right)^2=\left(p+2a\right)x^2+qx+r+a^2$，等式右边是完全平方式当且仅当它的判别式为0，即

$$q^2=4\left(p+2a\right)\left(r+a^2\right) \qquad (2.2)$$

此为a的三次方程。利用一元三次方程的解法，可以解出参数a。这样原方程两边都是完全平方式，开方后就是一个关于x的一元二次方程，于是可以求解原方程。

下面讨论直接用求根公式求解存在的问题。直接用一元二次方程求根公式求解存在的问题，前面已经讨论过了。对于一元三次方程，直接用公式(1)求解是有问题的，见下面的测试程序。误差较大的原因是当p很小时，$\frac{q}{2}$与$\sqrt{\left(\frac{q}{2}\right)^2-\left(\frac{p}{3}\right)^3}$接近相等，出现了“相近的浮点数相减”现象，损失了精度。修正方法是先引入两个变量，令：$u=\sqrt[3]{\frac{q}{2}-\sqrt{\left(\frac{q}{2}\right)^2-\left(\frac{p}{3}\right)^3}}$，$v=\sqrt[3]{\frac{q}{2}+\sqrt{\left(\frac{q}{2}\right)^2-\left(\frac{p}{3}\right)^3}}$，于是$u=\frac{uv}{v}=\frac{p}{3v}$，公式(2.1)变为$x=u+v=\frac{p}{3v}+v$，这是$q$为正时的情况，$q$为负时处理方法类似。

程序示例 2.1 直接求解三次方程的测试程序（误差 1.e-6）

```
#include "stdafx.h"
#include "math.h"

// f(x) = x^3-px-q
//  方程: f(x) = 0
int main(int argc, char* argv[])
//  直接用公式求解
{
	double p, q, x, d, s, u, v, f ;

	p = 1.e-6 ;
	q = 1. ;

	d = q*q/4-p*p*p/27 ;
	s = sqrt(fabs(d)) ;
	u = pow(fabs(q/2-s), 1./3) ;
	v = pow(fabs(q/2+s), 1./3) ;
	x = u+v ; //  根

	f = x*x*x-p*x-q ; // f(x),  修正前

	return 0;
}
```

程序示例 2.1 的运行结果是：

f	-1.0000000000287557e-006
p	9.9999999999999995e-007
q	1.0000000000000000
x	1.0000000000000000

程序示例 2.2 用修正的公式求解三次方程的测试程序（误差为 0）

```
#include "stdafx.h"
#include "math.h"
```

```
// f(x) = x^3-px-q
// 方程: f(x) = 0
int main(int argc, char* argv[])
// 用修正的公式求解
{
      double p, q, x, d, s, u, v, f ;

      p = 1.e-6 ;
      q = 1. ;

      d = q*q/4-p*p*p/27 ;
      s = sqrt(fabs(d)) ;
      v = pow(fabs(q/2+s), 1./3) ;
      u = p/3/v ;
      x = u+v ; //  根

      f = x*x*x-p*x-q ; // f(x)

      return 0;
}
```

程序示例 2.2 的运行结果是：

f	0.00000000000000000
p	9.9999999999999995e-007
q	1.00000000000000000
x	1.0000003333333334

一般情况下用求根公式得到的解都是有误差的，有必要对解进行修正。以一元三次方程为例，令 $f_0 = x_0^3 - px_0 - q$ 接近为 0，即 x_0 是近似解，给 x_0 一个修正量 ε，代入原方程有 $(x_0+\varepsilon)^3 = p(x_0+\varepsilon)+q$，略去 ε 的高次项得到

$$\varepsilon \approx -\frac{f_0}{3x_0^2 - p} \tag{2.3}$$

将 x_0 修正为 $x_0+\varepsilon$ 可提高精度。在后面的章节中将看到这就是 Newton 法的特例，该方法的关键是**迭代**，即反复修正。迭代是整个数值方法的核心。

程序示例 2.3 对解进行一次修正有效提高了精度

```
#include "stdafx.h"
#include "math.h"

// f(x) = x^3-px-q
// 方程: f(x) = 0
```

```
int main(int argc, char* argv[])
// 直接用公式求解
{
	double p, q, x, d, s, u, v, f ;

	p = 1.e-6 ;
	q = 1. ;

	d = q*q/4-p*p*p/27 ;
	s = sqrt(fabs(d)) ;
	u = pow(fabs(q/2-s), 1./3) ;
	v = pow(fabs(q/2+s), 1./3) ;
	x = u+v ; // 根

	f = x*x*x-p*x-q ; // f(x), 修正前

	x =x-f/(3*x*x-p) ; // 修正
	f = x*x*x-p*x-q ; // f(x), 修正后

	return 0;
}
```

程序示例 2.3 的运行结果是：

f	3.3306690738754696e-013
p	9.9999999999999995e-007
q	1.0000000000000000
x	1.0000003333334444

利用求根公式进行求解，由于要进行多次平方根、立方根的计算，可造成一定的误差，所以求根公式不能直接使用。非线性问题，除少数情况外，一般不能用公式求解。一元五次及以上的方程没有一般的通解公式。通常采用迭代解法，即构造出一近似值序列逼近真解。解非线性方程和方程组有很大区别。解方程组要难得多，主要区别在于一维情形可以找到一个根的范围，然后逐步缩小根的范围，最终找到符合精度要求的根。而多维情况则很难确定根的存在。在后面的各节中，将针对一般非线性方程，介绍确定初始解、迭代求精、迭代加速以及步长调整等基本方法，并给出一些具体应用实例。

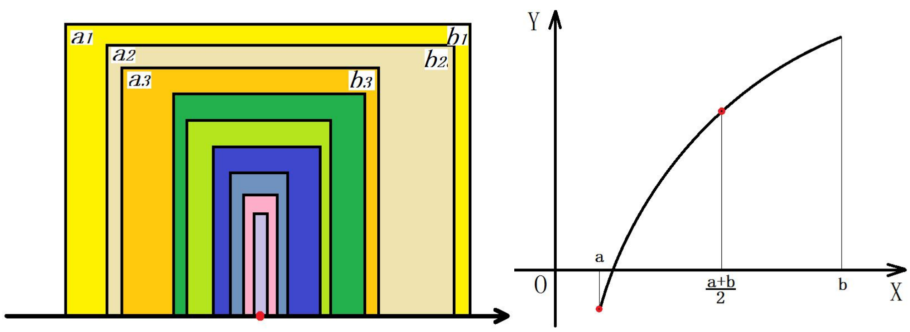
2.2 二分法

定理 2.1： 对于连续函数 $f(x)$，如果该函数在 a 和 b 处异号，即 $f(a)\cdot f(b)<0$，则 $f(x)$ 在 $[a,b]$ 内至少有一个根。

二分法的基本思想是将区间 $[a,b]$ 逐步二等分，使得每次缩小的区间中始终包含 $f(x)$ 的根 x^*，区间套 $[a_k,b_k]$ 满足：$[a_0,b_0]\supset[a_1,b_1]\supset[a_2,b_2]\cdots$，其中 $b_k-a_k=(b-a)/2^k$，x^* 在区间 $[a_k,b_k]$ 内。令

$$x_k=(a_k+b_k)/2 \tag{2.4}$$

于是 $\left|x^*-x_k\right|<(b_k-a_k)/2=(b-a)/2^{k+1}$。所以当 k 充分大后，x_k 收敛到 x^*。令 $(b-a)/2^{k+1}<\varepsilon$，其中 ε 是收敛精度，可以推出 k。二分法收敛的依据是闭区间套定理：闭区间 $[a_n,b_n]$，$n=0,1,2\cdots$，$\forall n$ 满足 $[a_n,b_n]\supset[a_{n+1},b_{n+1}]$，且 $\lim\limits_{n\to\infty}(b_n-a_n)=0$，则 $\exists!\xi$ 满足 $\xi=\bigcap[a_n,b_n]$，见图 2.3。



（a）区间套定理　　（b）二分法原理

图 2.3 区间套与二分法

算法 2.1 二分法算法

```
Input：函数 f(x), 区间[a, b],  f(a)·f(b)<0, 容差 e>0, 最大迭代次数 max
Output：根 x
Begin
    For i←0 to max, do
        x←a+(b−a)/2
        If |f(x)|<e, then
            Return Success
        End If
        If f(a)*f(x)>0, then
            a←x ;
        Else
            b←x ;
        End If
    End For
    Return Error
End
```

【例】用二分法求解方程 $f(x)=x^3-x-1=0$ 在 $[1,1.5]$ 内的解，精度为 10^{-2} 。

解：取初始区间 $[1,1.5]$，区间对分过程如下：

表 2.1 区间对分过程

序号	区间	区间端点函数值	中点及函数值
1	[1,1.5]	−1,0.875	1.25,−0.2969
2	[1.25,1.5]	−0.296875,0.875	1.375,0.2246
3	[1.25,1.375]	−0.296875,0.2246	1.3125,−0.05151
4	[1.3125,1.375]	−0.05151,0.2246	1.34375,0.08261
5	[1.3125,1.34375]	−0.05151,0.08261	1.328125,0.01458
6	[1.3125,1.328125]	−0.05151,0.01458	1.3203125,−0.01871
7	[1.3203125,1.328125]	−0.01871,0.01458	1.32421875,−0.002128

满足精度要求的解为 1.32421875 。

二分法的特点：

1. 对函数要求比较低，只要函数**连续即可，**
2. 收敛速度**慢**，收敛速度与以 1/2 为公比的等比级数相同，没有充分利用函数值，
3. 一次对分过程只能求一个解，不能求复根，
4. 一般用于为其它非线性方程快速求解方法**提供初值**。

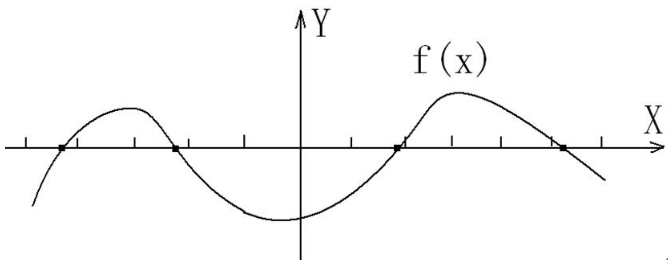


图 2.4 根所在区间的确定

区间的确定。可以用等分区间法估计区间$[a,b]$。在$f(x)$的连续区间$[a,b]$内，选择一系列的x值$x_1,x_2,x_3\cdots x_n$（一般取等间距），观察$f(x)$在这些点处的函数值的符号变化情况，当出现两个相邻点上的函数值异号时，根据根的存在定理，此小区间上至少有一个实根。

【例】确定$f(x)=x^2-2x-1=0$的有根区间。

解：设从$x=0$出发，取$h=0.25$为步长向右进行根的扫描，列表记录各个结点上函数值的符号，发现在区间$(2.25,2.5)$内必有实根，因此可取$x_0=2.25$或$x_0=2.5$作为根的初始近似值。在具体运用上述方法时，步长的选择是个关键。若步长h足够小，就可以求得任意精度的根的近似值；但h过小，在区间长度大时，会使计算量增大，h过大，又可能出现漏根的现象。因此，这种根的隔离法，只适用于求根的初始值。

表 2.2 用等分区间法确定根所在的区间

x	0	0.25	0.5	0.75	1.	1.25	1.5	1.75	2	2.25	2.5	2.75	3
$f(x)$	—	—	—	—	—	—	—	—	—	—	+	+	+

迭代次数固定，
0.5分割区间长度最短

为什么按等分的方式进行区间的分割？假设初始区间为$[0,1]$，取定一个固定的比例值t，满足$0<t<1$，在每一步迭代过程中，按$1-t$和t分割区间。对于一个子区间，零点出现在该子区间的概率与其长度成正比，所以在总计n次迭代过程中，按$1-t$比例保留子区间的迭代

二项分布

次数约为$(1-t)n$，按t比例保留子区间的迭代次数约为tn。故区间最终长度为$(1-t)^{(1-t)n}t^{tn}=\left[(1-t)^{(1-t)}t^t\right]^n$，该长度越小表明精度越高。下图为函数$\alpha(t)=(1-t)^{(1-t)}t^t$的图像，其最小值在$t=0.5$达到。证明如下：令$f(t)=(1-t)^{(1-t)}$，则$\ln f=(1-t)\ln(1-t)$，此式两边求导得$\frac{f'}{f}=-\ln(1-t)-1$，于是$f'=-[\ln(1-t)+1]f$；同样令$g(t)=t^t$，可得$g'=(\ln t+1)g$。由于$\alpha(t)=f(t)g(t)$，所以$\alpha'=f'g+fg'$，$\alpha(t)$取极值的条件是$\alpha'=0$，即$\ln t=\ln(1-t)$，求解得$t=0.5$。

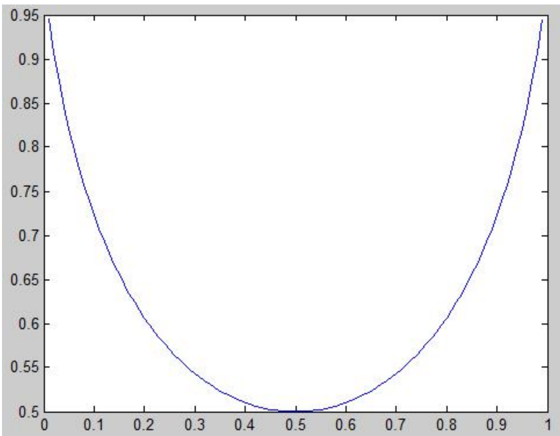


图 2.5 函数$\alpha(t)=(1-t)^{(1-t)}t^t$的图像

下面介绍一种改进的二分法——混合法（参考 False Position Method）。在混合法迭代过程中，区间的分断点轮流取区间中点和割线交点，割线交点是函数两个端点的连线与x轴的交点（图 2.6），这样可期待更快的收敛速度。下面求解方程$x^3-x-1=0$在$[1,1.5]$内的解，精度为10^{-6}，对比二分法和混合法，测试结果为：二分法迭代 19 次，混合法仅迭代 9 次。

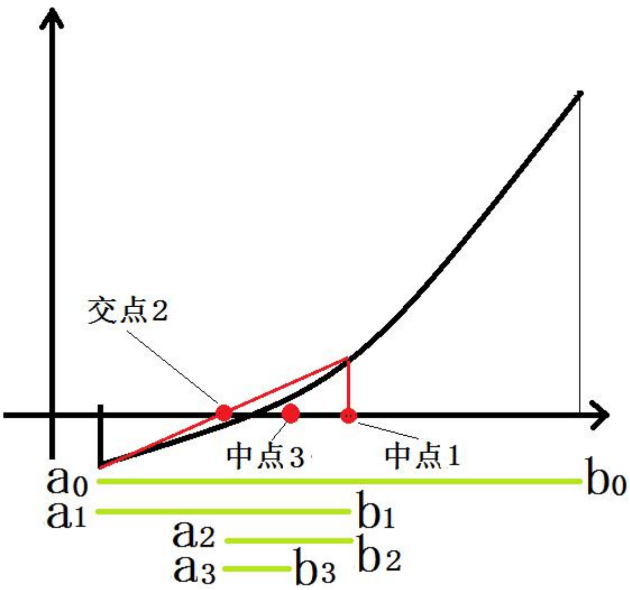


图 2.6 改进的二分法——混合法

程序示例 2.4 改进的二分法程序

```
#include "stdafx.h"
#include "math.h"
#include "stdio.h"

double f(double x)
{
	//return x*x*x-x-1. ;
	return exp(0.693147180559945+x)-2. ;
	//return (x-1.e-8)*(x-1.5) ;
}

/*	return -1:error, 0:no root, 1:one root, 3:迭代超过 max 次
 */
int sub2(double a,
	   double b,
	   double(*f)(double x),
	   double e,
	   int max,
	   double& x)	C语言中没有引用
{
	int i ;
	double x0, x1, y0, y1, y ;

	y0 = f(a) ;
	y1 = f(b) ;
	if( fabs(y0) < e )
	{
		x = a ;
		return 1 ;
	}
	else
	if( fabs(y1) < e )
	{
		x = b ;
		return 1 ;
	}
	if( y0*y1 > 0. )
		return 0 ; // no root
	x0 = a ;
```

```
        x1 = b ;
        for( i = 0 ; i < max ; i++ )
        {
                x = 0.5*(x0+x1) ;
                y = f(x) ;
                if( fabs(y) < e )
                        return 1 ;
                if( y0*y < 0. ) // [x0,x]
                {
                        x1 = x ;
                        y1 = y ;
                }
                else // [x,x1]
                {
                        x0 = x ;
                        y0 = y ;
                }
        }

        return 3 ;
}

int main(int argc, char* argv[])
{
        int rt ;
        double x ;

        rt = sub2(0., 1., f, 1.e-20, 100, x) ;

        return 0;
}
```

2.3 定点法

迭代法是求解非线性方程的基本方法。定点迭代法的思想是：将方程$f(x)=0$转化为方程$x=\varphi(x)$，构造

$$x_{k+1}=\varphi(x_k) \tag{2.5}$$

其中$k=0,1,2,\cdots$。给定初值x_0，可以计算得到$x_1=\varphi(x_0)$，$x_2=\varphi(x_1)$，…。称$\{x_k\}$为**迭代序列**，称$\varphi(x)$为**迭代函数**。如果$\{x_k\}$收敛于x^*，则x^*就是方程的解。迭代函数不唯一，也不一定收敛。

【例】求方程$f(x)=x-2^x+1.5=0$的一个根。

解：因为$f(0)>0$，$f(2)<0$，在[0, 2]中必有根。将原方程改为$2^x=x+1.5$，$x=\log_2(x+1.5)$。由此得迭代格式$x_{k+1}=\log_2(x_k+1.5)$，取初始值$x_0=0$，可逐次算得x_1 = 0.5850，x_2 = 1.0600，x_3 =1.3562，x_4 =1.5141，…，x_{14}

$= 1.6598$， x_{15} $=1.6598$，所以取 $x = 1.6598$ 为根。

如改写成 $x = 2^x - 1.5$ 且取 $x_0 = 2$，则迭代序列不收敛。定点法的收敛条件是什么？

定理 2.2 如果 $\varphi(x)$ 满足下列条件（1）当 $x\in$[a, b]时，$\varphi(x)\in$[a, b]，（2）对任意 $x\in$[a, b]，存在 0< L< 1，使 $|\varphi'(x)| \le L < 1$。则方程 $x = \varphi(x)$ 在[a, b]上有唯一的根 x^*,且对任意初值 $x_0 \in$[a, b]时，迭代序列 $x_{k+1} = \varphi(x_k)$ ，$k = 0, 1, 2, \cdots$，收敛于 x^*，见图 2.7。

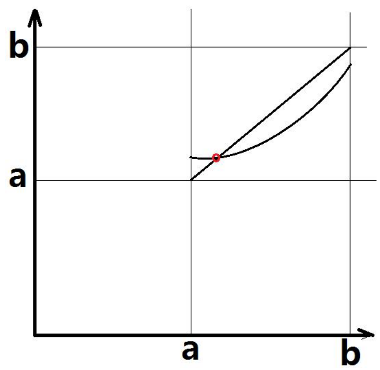


图 2.7 定点法的收敛条件

1912 年荷兰数学家布劳威尔（L.Brouwer）证明了不动点定理：**假设 D 是某个圆盘中的点集，f 是一个从 D 到它自身的连续函数，则存在点 x 满足** $f(x) = x$。有这样一个推论：把一张当地的地图平铺在地上，则总能在地图上找到一点，这个点下面的地上的点正好就是它在地图上所表示的位置，见图 2.8 和图 2.9。

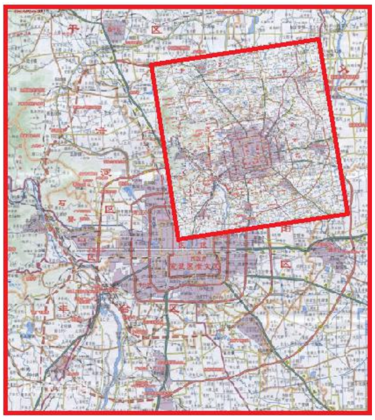


图 2.8 两张幅面不同的地图重叠——存在“不动点”

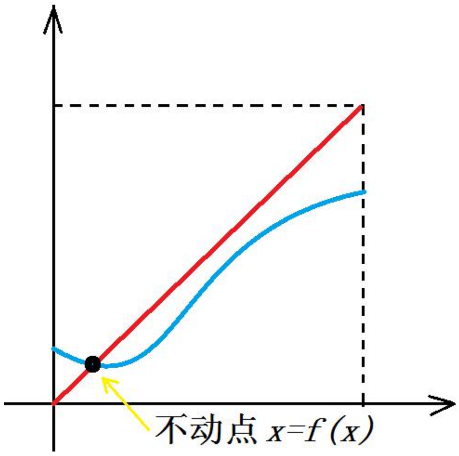


图 2.9 不动点定理的一维情况

有根性：

考虑到 $f(x)=x-\varphi(x), f(a)\le 0, f(b)\ge 0, f'(x)=1-\varphi'(x)>0$，所以 $x=\varphi(x)$ 有根并且是唯一的，参考图 2.10。

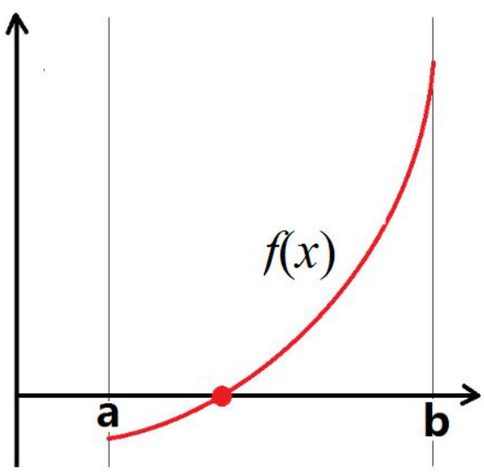


图 2.10 定义 $f(x)=x-\varphi(x)$

收敛性：

在下面的讨论中用到了：(1)等比级数求和公式 $1+q+q^2++q^n=\dfrac{1-q^{n+1}}{1-q}$；(2)柯西定理(Cauchy)：如果数列 $\{a_n\}$ 满足对 $\forall\ \varepsilon>0$，$\exists N$，使对 $\forall\, n>N$ 和 $m>N$，有 $|a_n-a_m|<\varepsilon$，则该数列收敛。(注意：可以证明柯西数列有界，有界就有收敛子列，可证柯西数列随子列收敛。)；(3)拉格朗日中值定理(Lagrange)：如果函数 $f(x)$ 在 $[a,b]$ 上连续、在 (a,b) 可导，则 $\exists\ \xi\in(a,b)$ 满足 $f(b)-f(a)=f'(\xi)(b-a)$。

令 $x_{k+1}=\varphi(x_k)$，$\varphi(x)$ 满足定理中的收敛条件。为应用柯西定理证明 $\{x_n\}$ 收敛，需推出几个不等式公式：

先推出两个需要用到的不等式。使用 $x=\varphi(x)$ 和拉格朗日中值定理可以推出，$|x_{k+1}-x_k|=|\varphi(x_k)-\varphi(x_{k-1})|=|\varphi'(\xi)||x_k-x_{k-1}|\le L|x_k-x_{k-1}|$，其中 $L<1$，所以有 $\boxed{|x_{k+1}-x_k|\le L^k|x_1-x_0|}$。同样方法可以证明：对于整数 $r>0$ 有 $|x_{k+r}-x_{k+r-1}|\le L^r|x_k-x_{k-1}|$。

再推出一个需要用到的不等式：对于任意正整数 p，有 $|x_{k+p}-x_k|=|x_{k+p}-x_{k+p-1}+x_{k+p-1}-x_{k+p-2}+x_{k+p-2}-\cdots-x_k|\le(L^p+\cdots+L)|x_k-x_{k-1}|$，所以有 $|x_{k+p}-x_k|<\dfrac{L}{1-L}|x_k-x_{k-1}|$，即只要 $|x_k-x_{k-1}|$ 充分小，就可以保证 $|x_{k+p}-x_k|$ 足够小。

对于 $\forall\ \varepsilon>0$，令 $N=\left[\dfrac{\ln\dfrac{2|x_1-x_0|}{\varepsilon(1-L)}}{\ln\dfrac{1}{L}}\right]+1$，其中 $[x]$ 表示不超过实数 x 的最大整数，于是 $\dfrac{2L^N}{1-L}|x_1-x_0|<\varepsilon$。对 $\forall\, n>N$ 和 $m>N$，根据前面推导出的两个不等式公式有：

$$
\begin{aligned}
|x_n - x_m| &< |x_n - x_N| + |x_m - x_N| \\
&< \frac{2L}{1-L}|x_N - x_{N-1}| \\
&\le \frac{2L^N}{1-L}|x_1 - x_0| \\
&< \varepsilon
\end{aligned}
$$

所以根据柯西定理知$\{x_n\}$收敛，故存在**极限**x^*。

在求根过程中x^*是不知道的，不可能用$|x^* - x_k| < \varepsilon$来作为迭代结束条件，可用$|x_k - x_{k-1}|$**控制迭代次数。**下面证明这个结论。由于

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

$$
\le L\left(|x^* - x_{k+1}| + |x_{k+1} - x_k|\right)
$$

可以推导出：

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

$$
\boxed{|x^* - x_{k+1}| \le \frac{L}{(1-L)}|x_{k+1} - x_k|}
$$

所以可用$|x_k - x_{k-1}|$控制迭代次数。

【例】求方程$x^3 - 5x + 1 = 0$在[0, 1]内的根，精确到10^{-4}。

解：将方程变形为$x = \frac{1}{5}(x^3 + 1) = \varphi(x)$。$\varphi'(x) = 0.6x^2 > 0$，$\varphi(x)$在[0, 1]内为增函数，所以$L = \max|\varphi'(x)| < 1$满足收敛条件，取$x_0 = 0.5$，则

$$x_1 = \varphi(x_0) = 0.225，$$

$$x_2 = \varphi(x_1) = 0.2023，$$

$$x_3 = \varphi(x_2) = 0.2017，$$

$$x_4 = \varphi(x_3) = 0.2016，$$

$$x_5 = \varphi(x_4) = 0.2016，$$

近似根为$x^* = 0.2016$。

【例】求 25 的立方根，精确到 10^{-6}。

解：求解方程 $x^3=25$，由于 2.9^3=24.389，显然解在区间[2.9,3]之内。令

$$\varphi(x)=x-\frac{x^3-25}{3x^2}$$

在区间[2.9,3]内可验证

$$|\varphi'(x)|=\left|1-\frac{3x^2\cdot 3x^2-(x^3-25)\cdot 6x}{6x^4}\right|=\left|\frac{1}{2}-\frac{25}{x^3}\right|<1$$

所以取 $x_0=2.9$， $x_{n+1}=\varphi(x_n)$ 迭代收敛。迭代过程如下：

$$x_1=\varphi(x_0)\approx 2.924217$$

$$x_2=\varphi(x_1)\approx 2.924018$$

$$x_3=\varphi(x_2)\approx 2.924018$$

25 的立方根近似解为 x^*=2.924018。

收敛速度：

定义：设序列 $\{x_k\}$ 收敛于方程的根 x^*，如果存在正实数 p，使得 $\lim\limits_{k\to\infty}\frac{|x^*-x_{k+1}|}{|x^*-x_k|^p}=C$（$C$ 为非零常数），则称序列 $\{x_k\}$ 收敛于 x^* 的**收敛速度**是 p 阶的。

当 $p=1$ 时，称为线性收敛；当 $p=2$ 时，称为二次收敛，或平方收敛。若 $\varphi'(x)$ 连续，且 $\varphi'(x^*)\neq 0$，则迭代格式 $x_{k+1}=\varphi(x_k)$ 必为线性收敛。因为由 $|x^*-x_{k+1}|=|\varphi(x^*)-\varphi(x_k)|=|\varphi'(\xi)||x^*-x_k|$，可推出 $\lim\limits_{k\leftarrow\infty}\frac{|x^*-x_{k+1}|}{|x^*-x_k|}=|\varphi'(x^*)|\neq 0$。

迭代加速：

下面讨论迭代法的**埃特金**（Aitken）**加速**。假设$x_{k+1}=\phi(x_k)$是收敛的，因此有$\lim\limits_{k\to\infty}\dfrac{x_{k+1}-x^*}{x_k-x^*}=\phi'(x^*)$。当$k$充分大以后

$$\frac{x_{k+2}-x^*}{x_{k+1}-x^*}\approx\frac{x_{k+1}-x^*}{x_k-x^*} \tag{2.6}$$

从（2.6）式中解出

$$x^*\approx\frac{x_k x_{k+2}-x_{k+1}^2}{x_{k+2}-2x_{k+1}+x_k} \tag{2.7}$$

（2.7）式作为x_k则可能精度更高。埃特金加速法的几何意义见下图：

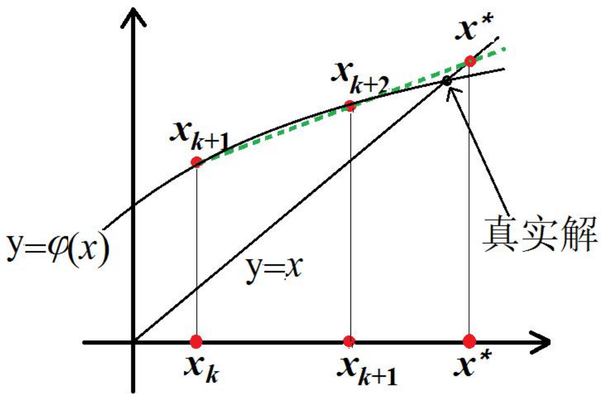


图 2.11 埃特金加速

程序示例 2.5 定点法程序实例

```
#include "stdafx.h"
#include "math.h"

// 用定点法解方程：x = 0.25*exp(x)
double fai(double x)
{
	return 0.25*exp(x) ;
}

// 迭代 34 次的解为 0.35740295618138967, 精度 1.e-15
int main()
{
	int i ;
	double x0 = 1., x1, e = 1.e-15 ;

	for( i = 0 ; i < 64 ; i++ )
	{
		x1 = fai(x0) ;
		if( fabs(x1-x0) < e )
			return 1 ;
		x0 = x1 ;
```

```
	}

	return 0;
}
```

程序示例 2.6 带埃特金加速的定点法程序实例

```
int main()
{
	int i ;
	double x0 = 1., x1, x2, x, y, d, e = 1.e-15 ;

	x1 = fai(x0) ;
	for( i = 0 ; i < 64 ; i++ )
	{
		x2 = fai(x1) ;
		if( fabs(x2-x1) < e )
			return 1 ;
		d = x2-2*x1+x0 ;
		if( fabs(d) > 1.e-20 )
		{
			x = (x0*x2-x1*x1)/d ;
			y = fai(x) ;
			if( fabs(x-y) < e )
				return 1 ;
		}
		x0 = x1 ;
		x1 = x2 ;
	}

	return 0;
}
```

表 2.3 埃特金加速程序实例测试结果

i	$x_2 - x_1$	$y - x$	d
0	−0.1863	0.0816	0.1341
1	−0.0838	0.0107	0.1024
2	−0.0329	0.0014	0.0509
3	−0.0122	0.0002	0.0207
4	−0.0044	2.385e−5	0.0077
5	−0.0016	3.055e−6	0.0028

从表 2.1 埃特金加速程序实例测试结果可以看出，埃特金加速在迭代的开始阶段是非常有效的。值得注意的是随着不断迭代，d 值趋于 0，导致埃特金加速加速失效。此现象揭示在数值计算中要避免相近的数相减。

2.4 牛顿法

牛顿法(Newton)利用非线性方程 $f(x)=0$ 的切线实现迭代过程，

是实用有效的求解非线性方程的方法之一。已知方程 $f(x)=0$ 的一个近似根 x_0，将 $f(x)$ 在 x_0 处作泰勒展开，

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\cdots \tag{2.8}$$

取（2.8）式中的前两项来近似代替 $f(x)$，则得近似的线性方程 $f(x_0)+f'(x_0)(x-x_0)=0$。设 $f'(x_0)\neq 0$，解得 $x=x_0-\dfrac{f(x_0)}{f'(x_0)}$ 作为近似根 x_1，于是

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)} \tag{2.9}$$

称(2.9)式为求解 $f(x)=0$ 的**牛顿迭代公式**，$\{x_k\}$ 称为**牛顿迭代序列**。

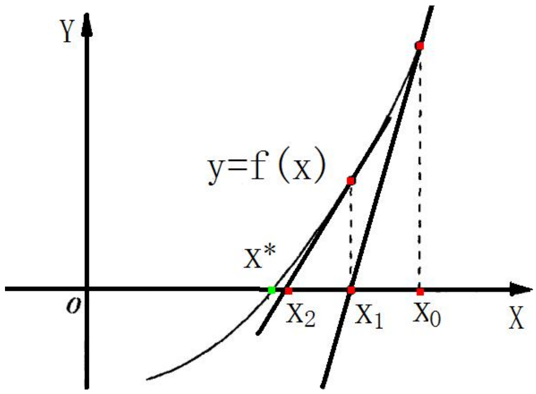


图 2.12 牛顿法的迭代过程

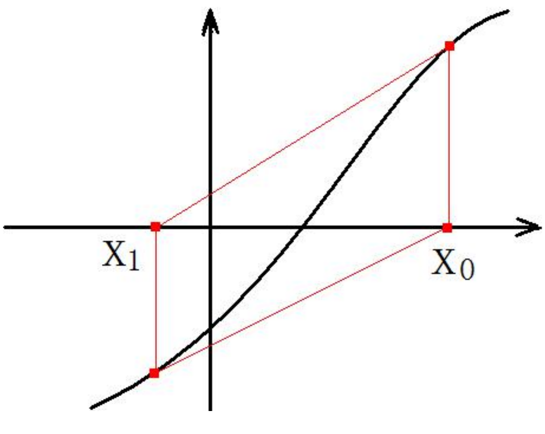


图 2.13 牛顿法迭代不收敛的情况

牛顿法的几何意义。求得 x_k 以后，过曲线 $y=f(x)$ 上对应点 $(x_k, f(x_k))$ 作切线，切线为 $y=f(x_k)+f'(x_k)(x-x_k)$。求切线和 x 轴交点，即得 x_{k+1}，因此牛顿法也称**切线法**，参考图 2.12。对于 $y=f(x)$ 有拐点且初始点不在零点附近，牛顿迭代可能不收敛，见图 2.13。

收敛性：

将牛顿法公式对应的定点法迭代格式为

$$\varphi(x)=x-\frac{f(x)}{f'(x)} \tag{2.10}$$

由于 $|\varphi'(x)|=\frac{|f''(x)|}{[f'(x)]^2}|f(x)|$，若在根 x^* 某个邻域 R：$|x^*-x|\le\delta$ 内，$f'(x)\ne 0$，$f''(x)$ 有界，只要 $|f(x)|$ 充分小，就能使 $|\varphi'(x)|\le L<1$，牛顿迭代法收敛于 x^*。也就是说牛顿法在零点局部一定收敛，那么牛顿法在某个区间上的收敛条件是什么？

定理 2.3 设 $f(x)$ 在 $[a,b]$ 上二阶导数连续，满足下列条件：

(1) $f(a)f(b)<0$；

(2) $f'(x)\ne 0$；

(3) $f''(x)$ 保号，即 $f''(x)$ 在 $[a,b]$ 内恒大于 0 或恒小于 0；

(4) $\left|\frac{f(a)}{f'(a)}\right|\le b-a, \left|\frac{f(b)}{f'(b)}\right|\le b-a$。

则对任意 $[a,b]$ 内的初始值 x_0 牛顿迭代序列 $\{x_k\}$ 收敛于 $f(x)$ 在 $[a,b]$ 上的唯一根 x^*。

条件(1)保证根的存在，条件(2)表示 $f(x)$ 单调，所以根唯一，条件(3)保证曲线的凸凹向不变，见图 2.14。

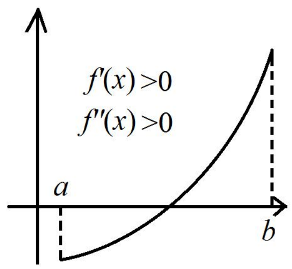


图 2.14 $f''(x)$ 保号保证了函数凸凹性不变

下面证明定理 2.3。

首先，假设 $f'(x)>0$，即函数单增。由条件(1)和(2)可推出 $\dfrac{f(a)}{f'(a)}\le 0$ 和 $\dfrac{f(b)}{f'(b)}\ge 0$ 。

其次，令 $\varphi(x)=x-\dfrac{f(x)}{f'(x)}$，于是 $\varphi'(x)=\dfrac{f''(x)}{\left[f'(x)\right]^2}f(x)$，令 x^* 满足 $f(x^*)=0$，则 $\varphi(x)$ 的极值只能是 $\varphi(a)$、$\varphi(b)$ 和 $\varphi(x^*)=x^*$，再根据条件(4)得到 $a\le\varphi(x)\le b$，这就证明了 $\{x_k\}$ **有界**。

然后，令 $x=x_k$，$\Delta x=-\dfrac{f(x_k)}{f'(x_k)}$，将此两个式子代入 Taylor 公式 $f(x+\Delta x)=f(x)+\Delta x f'(x)+\dfrac{\Delta x^2}{2}f''(\xi)$ 得 $f(x_{k+1})=\dfrac{\Delta x^2}{2}f''(\xi)$，由于 $f''(x)$ 保号，所以 $f(x_{k+1})$ 保号，$\Delta x=-\dfrac{f(x_k)}{f'(x_k)}$ 也保号，所以 $\{x_k\}$ **单调**($k>0$)。

最后，由 $\{x_k\}$ 单调有界知 $\{x_k\}$ 收敛。假设 $\{x_k\}$ 收敛于 $\bar{x}$，由于 $x_{k+1}=x_k-\dfrac{f(x_k)}{f'(x_k)}$，所以 $\dfrac{f(x_k)}{f'(x_k)}$ 收敛于 0，即 $f(\bar{x})$ 等于 0。

平方收敛：

牛顿法的优点是平方收敛的。将 $f(x)$ 在 x_k 处按泰勒展开，x^* 代替 x 得

$$f(x^*)=f(x_k)+f'(x_k)(x^*-x_k)+\frac{f''(\xi)}{2!}(x^*-x_k)^2=0 \tag{2.11}$$

所以有 $f(x_k)+f'(x_k)(x^*-x_k)=-\frac{1}{2}f''(\xi)(x^*-x_k)^2$ ，用导数 $f'(x_k)$ 除上式左右两端，整理后得到 $x^*-x_k+\frac{f(x_k)}{f'(x_k)}=-\frac{1}{2}\frac{f''(\xi)}{f'(x_k)}(x^*-x_k)^2$，即 $x^*-x_{k+1}=-\frac{1}{2}\frac{f''(\xi)}{f'(x_k)}(x^*-x_k)^2$。所以当 $k\to\infty$ 时有

$$\frac{|x^*-x_{k+1}|}{|x^*-x_k|^2}=\left|\frac{f''(\xi)}{2f'(x_k)}\right|\to\left|\frac{f''(x^*)}{f'(x^*)}\right| \tag{2.12}$$

故牛顿法是平方收敛的。

牛顿算法的流程图见图 2.15：

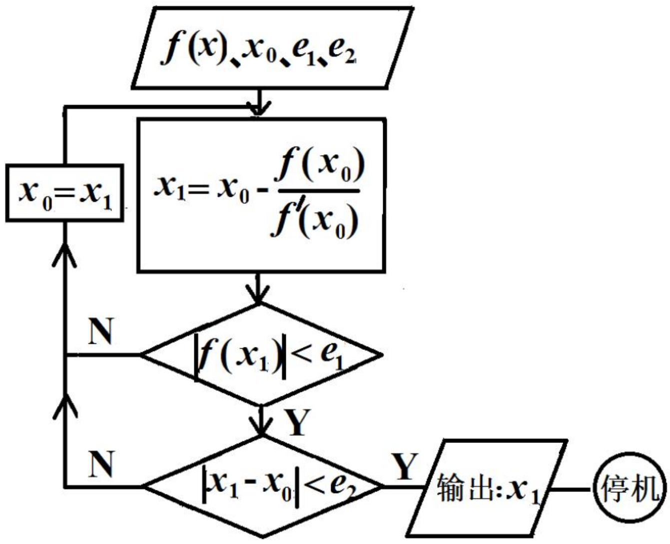


图 2.15 牛顿法算法流程图($f'(x)\neq 0, e_1>0, e_2>0$ 是容差)

【例】用牛顿法解方程 $f(x)=x(x+1)^2-1=0$ 在 0.4 附近的根。

解：由于 $f'(x)=(x+1)(3x+1)$，所以

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

迭代结果见下表，取 x^* 为 0.46557，见表 2.4。

表 2.4

k	0	1	2	3
x_k	0.4	0.47013	0.46559	0.46557

【例】构建牛顿迭代公式求解平方根$\sqrt{d}\left(d>0\right)$。

解： 设 $f(x)=x^2-d$,因为 $f'(x)=2x$，得迭代公式：

$$x_{k+1}=x_k-\frac{x_k^2-d}{2x_k}$$

化简得

$$x_{k+1}=\frac{1}{2}\left(x_k+\frac{d}{x_k}\right)$$

下面给出实现牛顿迭代过程的参考代码（$f(x)$和 $df(x)$用于计算函数值及其导数值，容差 $e1>0$ 用于判断迭代是否收敛，容差 $e2>0$ 用于判断导数是否为 0）：

程序示例 2.7 牛顿迭代基本算法

```
double f(double x)
{
    // 计算并返回函数值 f(x)...
}

double df(double x)
{
    // 计算并返回函数导数值 f'(x)...
}

// 牛顿迭代过程，x0 是初值，
int newton(double(*f)(double x),
           double(*df)(double x),
           double x0,
           double e1,
           double e2,
           int max,
           double& x)
{
    int i ;
    double y, d ;

    x = x0 ;
    for( i = 0 ; i < max ; i++ )
    {
        y = f(x) ;
        d = df(x) ;
        if( fabs(d) < e2 )
            return 0 ;
        d = y/d ;
        x -= d ;
        if( fabs(d) < e1 ) // 收敛
            return 1 ;
```

```
    }
    return 0 ;
}
```

2.5 牛顿下山法

牛顿法对初始值x_0要求较高。**牛顿法下山**法可提高牛顿法的收敛性：将牛顿法迭代公式修改为

$$x_{k+1} = x_k - \lambda \frac{f(x_k)}{f'(x_k)} \tag{2.13}$$

其中$k = 0, 1, 2, \cdots$，λ>0 是一个参数，λ的选取应使$|f(x_{k+1})| < |f(x_k)|$成立。当$|f(x_{k+1})| < \varepsilon_1$且$|x_{k+1} - x_k| < \varepsilon_2$时（$\varepsilon_1$为$y$值精度，$\varepsilon_2$为$x$值精度），就停止迭代，取$x^* \approx x_{k+1}$，否则再减小λ继续迭代。按上述过程，得到单调序列$\{|f(x_k)|\}$。λ称为**下山因子**，要求满足$0 < \varepsilon_\lambda \le \lambda \le 1$，$\varepsilon_\lambda$为**下山因子下限**。一般可取$\lambda = 1, \frac{1}{2}, \frac{1}{2^2}, \cdots, \lambda \ge \varepsilon_\lambda$，且使$|f(x_{k+1})| < |f(x_k)|$。下山法放宽了初值$x_0$的选取条件，有时用牛顿法不收敛，但用下山法收敛。下面给出牛顿下山法的算法步骤。

算法 2.2 牛顿下山法

Input:函数$f(x)$满足$f'(x) \ne 0$，初始值x_0，容差$\varepsilon_1 > 0$，$\varepsilon_2 > 0$，最大循环次数max

Output:根x

Begin

　　$\lambda \leftarrow 1$ // 置下山因子λ 为 1

　　For $i \leftarrow 0$ to max, do

　　　　$x = x_0 - \lambda \frac{f(x_0)}{f'(x_0)}$

　　　　If $|f(x)| < \varepsilon_1$ **and** $|x - x_0| < \varepsilon_2$, then

　　　　　　Return Success

　　　　End If

　　　　If $|f(x)| < |f(x_0)|$, then

　　　　　　$x_0 \leftarrow x$

　　　　　　$\lambda \leftarrow 1$

　　　　Else

```
            λ←0.5×λ
        End If
    End For
    Return Error
End
```

表 2.5 牛顿下山法迭代过程

k	x_k	$f(x_k)$	λ
0	0.58	-1.3848	0.001
1	0.73053	-1.34066	0.05
2	0.84206	-1.24498	0.1
3	0.95251	-1.08832	0.5
4	1.26855	-0.22719	1.
5	1.32790	0.01362	1.
6	1.32473	4.01009e-05	1.
7	1.32472	3.51380e-10	1.

【例 1】方程 $f(x)=x^3-x-1=0$ 的一个根为 $x^*\approx 1.32472$，若取初值 $x_0=$ 0.58，用牛顿法计算 $x_1=x_0-\dfrac{f(x_0)}{f'(x_0)}$，则 x_1 比 x_0 误差更大，导致迭代不收敛。改用牛顿下山法 $x_{k+1}=x_k-\lambda\dfrac{f(x_k)}{f'(x_k)}$，$k=0, 1, 2\cdots$，λ的取值及迭代结果见表 2.5。

【例 2】为对比分析牛顿法和牛顿下山法，程序示例 2.8 分别用此两种方法迭代计算点到椭圆的最近距离。在一百万次计算中：牛顿法失败 1045 次，牛顿下山法失败 18 次，所以牛顿下山法显著提高了迭代成功率。

程序示例 2.8 点到椭圆最近点计算程序

```
// a,b 是椭圆(x^2/a^2+y^2/b^2=1)长短轴长，(x,y)是一点
static double a = 1., b = 0.5, x = 0., y = 0. ;
//  点 P=(x,y)到椭圆上的点 Q=(a*cos(t),b*sin(t))的距离为：
// d(t) = (a*cos(t)-x)^2+(b*sin(t)-y)^2，当 Q 是最近点时 d'(t) = 0，即：
// (a*cos(t)-x)*(-a*sin(t))+(b*sin(t)-y)*b*cos(t) = 0
```

```
double f(double t)
{
	double c = cos(t), s = sin(t) ;
	return (b*b-a*a)*c*s+x*a*s-y*b*c ;
}

double df(double t)
{
	double c = cos(t), s = sin(t) ;
	return (b*b-a*a)*(c*c-s*s)+x*a*c+y*b*s ;
}

// Newton method, return 0:no root, 1:one root
int newton(double(*f)(double t),
	      double(*df)(double t),
	      double t0,
	      double e,
	      int max,
	      double& t)
{
	int i ;
	double Y, d ;

	t = t0 ;
	for( i = 0 ; i < max ; i++ )
	{
		Y = f(t) ;
		d = df(t) ;
		if( fabs(d) < 1e-100 )
			return 0 ; // error
		d = Y/d ;
		t -= d ;
		if( fabs(Y) < e )
			return 1 ;
	}

	return 0 ;
}

// Newton down_hill method, return 0:no root, 1:one root
int newton2(double(*f)(double t),
	       double(*df)(double t),
	       double t0,
	       double e,
	       int max,
	       double& t)
{
	int i, j ;
	double Y, d, old, lemda ;

	t = t0 ;
	for( i = 0 ; i < max ; i++ )
	{
		old = f(t) ;
		d = df(t) ;
		if( fabs(d) < 1e-100 )
			return 0 ; // error
		d = old/d ;
		lemda = 1. ;
		for( j = 0 ; j < 8 ; j++ )
		{
			Y = f(t-lemda*d) ;
			if( fabs(old) > fabs(Y) )
				break ;
			lemda *= (-0.5) ;
		}
		if( j < 8 ) //  调整 lemda 因子成功
```

```
            t -= lemda*d ;
        else //  调整 lemda 因子失败
        {
            t -= d ;
            Y = f(t) ;
        }
        if( fabs(Y) < e )
            return 1 ;
    }

    return 0 ;
}

int main()
{
    int unsuc = 0 ;
    double t0, t ;

    for( int i = 0 ; i < 1000000 ; i++ ) // 1M 次
    {
        x = (double)rand()/RAND_MAX ;
        y = (double)rand()/RAND_MAX ;
        t0 = atan2(y, x) ; // [-PI,PI]
        //if( newton(f, df, t0, 1.e-6, 256, t) == 0 ) //  失败率:1045/1M
        if( newton2(f, df, t0, 1.e-6, 256, t) == 0 ) //  失败率:18/1M
            unsuc++ ;
    }

   return 0;
}
```

2.6 弦截法

牛顿法有较高的收敛速度，但要计算函数的导数，有一定的计算量。本节给出用近似值取代函数导数的方法，简化牛顿迭代算法。

(1) **单点弦截法**。用平均变化率 $\frac{f(x_k)-f(x_0)}{x_k-x_0}$ 来替牛顿迭代公式 $x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$ 中的导数 $f'(x_k)$，得到迭代公式

$$x_{k+1}=x_k-\frac{f(x_k)}{f(x_k)-f(x_0)}(x_k-x_0) \tag{2.14}$$

(2.14)式称**单点弦截法迭代公式**。单点弦截法的几何意义如图2.16。两点 $(x_k,f(x_k))$ 和 $(x_0,f(x_0))$ 连线交 x 轴得 x_{k+1}，连线均以 $(x_0,f(x_0))$ 作为一个端点，只有一个端点不断更换，故名为单点弦截法。不断重复这个过程得到 $\{x_n\}$，$\{x_n\}$ 逼近曲线 $y=f(x)$ 与 x 轴交点的横坐标 x^*。

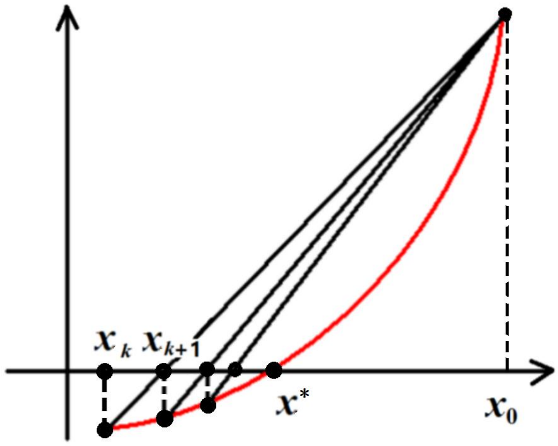


图 2.16 单点弦截法

(2) **双点弦截法。**改用$\dfrac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$代替导数$f'(x_k)$，就可以得到双点弦截法迭代公式

$$x_{k+1}=x_k-\frac{f(x_k)}{f(x_k)-f(x_{k-1})}(x_k-x_{k-1}) \tag{2.15}$$

双点弦截法的几何意义如图 2.17。两点$(x_{k\text{-}1},f(x_{k\text{-}1}))$和$(x_k,f(x_k))$连线与 x 轴交点的横坐标记为x_{k+1}，不断重复这个过程得到$\{x_n\}$，$\{x_n\}$逼近曲线$y=f(x)$与 x 轴交点的横坐标 x^*。双点弦截法具有超线性敛速。注意在双点弦截法中，从两个初始点 x_0 和 x_1 开始迭代。假设$f(x_0)f(x_1)<0$，当x_{k+1}确定后，如果$f(x_k)f(x_{k+1})>0$，则可以选择用x_{k-1}取代x_k后再计算x_{k+2}，从而保持$f(x_k)f(x_{k+1})<0$。

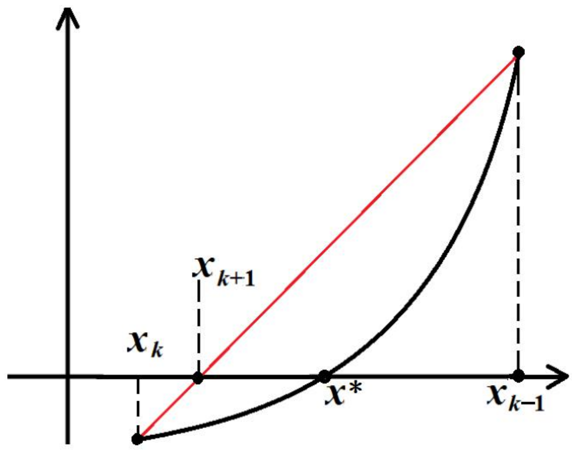


图 2.17 双点弦截法

穆勒(David E Muller)方法本质上也是弦截法。对于方程 $f(x)=0$，该方法输入为 x_0,x_1,x_2 及对应的函数值 f_0,f_1,f_2，构造一条二次函数 $g(x)=ax^2+bx+c$ 满足 $g(x_i)=f_i$， $i=0,1,2$，然后计算 $g(x)=0$ 的零点，取一个合适的零点作为 $f(x)=0$ 的近似根。若未达到收敛条件，则用该零点更新 x_0,x_1,x_2 和 f_0,f_1,f_2，继续迭代。

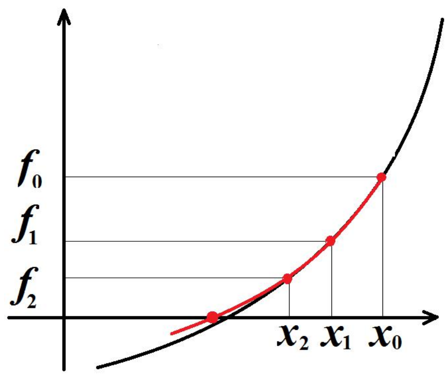


图 2.18 穆勒方法

2.7 哈雷法

对于非线性方程 $f(x)=0$，牛顿法在初始点 x_0 处构造切线，切线与 x 轴相交得到近似零点。按照此思路，可构造高阶收敛的迭代方法。比如构造二次函数 $g(x)$，该函数在初始点 x_0 处满足 $g(x_0)=f(x_0)$，

$g'(x_0)=f'(x_0)$， $g''(x_0)=f''(x_0)$，用$g(x)$的零点作为$f(x)=0$的近似零点，通过迭代得到$f(x)=0$满足精度要求的零点。或者构造圆弧，该圆弧在初始点x_0处与$f(x)=0$对应的曲线相切且具有相同的曲率中心。假设x_k是$f(x)=0$的近似零点，根据 Taylor 公式，

$$f(x)\approx f(x_k)+f'(x_k)(x-x_k)+\frac{f''(x_k)}{2}(x-x_k)^2 \tag{2.16}$$

如何选取下一个迭代点x_{k+1}？为使x_{k+1}进一步接近零点， x_{k+1}最好满足如下等式，

$$0=f(x_k)+f'(x_k)(x_{k+1}-x_k)+\frac{f''(x_k)}{2}(x_{k+1}-x_k)^2 \tag{2.17}$$

于是

$$0=f(x_k)+(x_{k+1}-x_k)\left[f'(x_k)+\frac{f''(x_k)}{2}(x_{k+1}-x_k)\right] \tag{2.18}$$

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)+\dfrac{f''(x_k)}{2}(x_{k+1}-x_k)} \tag{2.19}$$

从(2.18)式中直接解出x_{k+1}有一定的困难，这里使用一个技巧，即将(2.18)式改写成(2.19)式后，再用牛顿迭代公式$x_{k+1}-x_k=-\dfrac{f(x_k)}{f'(x_k)}$替换$x_{k+1}-x_k$， 于是得到哈雷(Edmond Halley)迭代公式

$$x_{k+1}=x_k-\frac{f(x_k)f'(x_k)}{f'(x_k)^2-f''(x_k)f(x_k)/2} \tag{2.20}$$

哈雷迭代公式收敛速度是 3 阶的。

2.8 非线性方程组

与求解单变量非线性方程相比，非线性方程组的求解要难得多。一维情形较易找到根的范围，而多维情况则很难确定解集的结构。单

变量非线性方程的解可能是孤立的“点”，双变量非线性方程组的解集可以构成“曲线”，而三个变量的非线性方程组的解集也许是“曲面”。总之，非线性方程组的解集（解空间）可能相当复杂。

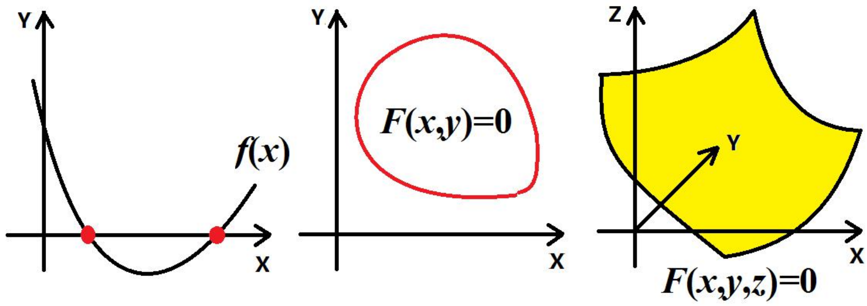


图 2.19 非线性方程与非线性方程组的解集结构对比

非线性方程组表示为：

$$\begin{cases} f_1(x_1,\cdots,x_n)=0 \\ \qquad\vdots \\ f_m(x_1,\cdots,x_n)=0 \end{cases} \tag{2.21}$$

(2.21)式简记为

$$F(X)=\begin{pmatrix} f_1(X) \\ \vdots \\ f_m(X) \end{pmatrix} \tag{2.22}$$

方程组的 Jacobi 矩阵记为：

$$J(X)=\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}_{m\times n} \tag{2.23}$$

根据多元函数的 Taylor 公式：

$$F(X+\Delta X)\approx F(X)+J(X)\Delta X \tag{2.24}$$

利用 Newton 迭代方法，在有初始值的情况下，可以通过数值方法解方程(2.24)，得到

$$\Delta X = -(J(X))^{+} F(X) \tag{2.25}$$

(2.25)式中上标“+”表示**广义逆矩阵**。下面以二元非线性方程组（2.26）为例推导牛顿迭代公式：

$$\begin{cases} f(x,y)=0 \\ g(x,y)=0 \end{cases} \tag{2.26}$$

根据多元函数的 Taylor 展开公式，可推出以下两个公式：

$$f(x+\Delta x, y+\Delta y) \approx f(x,y) + \Delta x f_x(x,y) + \Delta y f_y(x,y) \tag{2.27}$$

$$g(x+\Delta x, y+\Delta y) \approx g(x,y) + \Delta x g_x(x,y) + \Delta y g_y(x,y) \tag{2.28}$$

令

$$f(x,y) + \Delta x f_x(x,y) + \Delta y f_y(x,y) = 0 \tag{2.29}$$

$$g(x,y) + \Delta x g_x(x,y) + \Delta y g_y(x,y) = 0 \tag{2.30}$$

可以得到

$$\Delta x = \frac{\begin{vmatrix} -f & f_y \\ -g & g_y \end{vmatrix}}{\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix}} \quad \Delta y = \frac{\begin{vmatrix} f_x & -f \\ g_x & -g \end{vmatrix}}{\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix}} \tag{2.31}$$

写成迭代公式的形式：

$$x_{k+1} = x_k - \frac{\begin{vmatrix} f & f_y \\ g & g_y \end{vmatrix}}{\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix}} \quad y_{k+1} = y_k - \frac{\begin{vmatrix} f_x & f \\ g_x & g \end{vmatrix}}{\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix}} \tag{2.32}$$

注意（2.32）式中 f, f_x, f_y, g, g_x, g_y 略去了自变量 (x_k, y_k)。

为减少计算量和提高解的精度，一般不会直接利用上述方法求解。先需要对方程组进行**分解**，最大限度地降低数值求解过程中变量和方程的个数。最简单的分解是分块，使相互独立的方程组可以单独被求解，其次可以利用矩阵分块的方法，将方程分解成上三角的形式，

然后依次求解：

$$\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}\begin{pmatrix} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \end{pmatrix} \Rightarrow \begin{pmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \vdots & & \ddots & \\ A_{k1} & \cdots & \cdots & A_{kk} \end{pmatrix}\begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \\ \\ b_n \end{pmatrix} \tag{2.33}$$

下面以草图约束求解为例，进一步说明在实际应用中非线性方程组的求解过程。在三维特征造型系统中，草图求解非常重要。草图就是一个几何约束系统，将点、线段、圆转化为变量，几何约束（如平行、垂直等关系）和尺寸约束（如距离、半径等标注）转化为方程，可将草图对应为非线性方程组。图 2.20 是由 6 个点、9 个约束构成的草图，该草图对应一个 12 个变量（一个点对应两个变量）、9 个方程（一个约束对应一个方程）的非线性方程组。在求解过程中，草图通常被看作是一个图$\Omega=(V,E)$（图论中的图），顶点集合 V 就是点、线段、圆等几何元素，边集合 E 就是几何元素之间的尺寸约束或几何约束。通过特殊算法可以将图Ω分解为子图，再用数值方法单独求解。图 2.21 中三角形 ABC 和 DEF 构成刚体 1 和刚体 2，图中带 2 的圆圈表示一个点有两个自由度，可以先单独求解两个刚体，再将两个刚体作为独立的整体求解，满足约束 d7,d8,d9。该方法降低了方程求解规模，提高了解的精度。

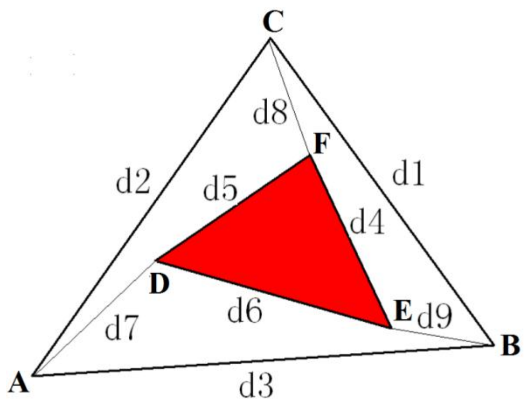


图 2.20 草图

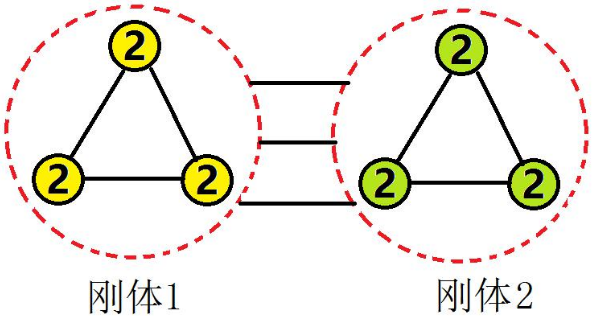


图 2.21 三角形 ABC 和 DEF 构成两个刚体

2.9 迭代法应用

迭代法结合具体的问题有广泛的应用。下面给出一个迭代法的应用实例：已知三条平面曲线及其上面三个初始切点，求解一个圆与三条曲线都相切。

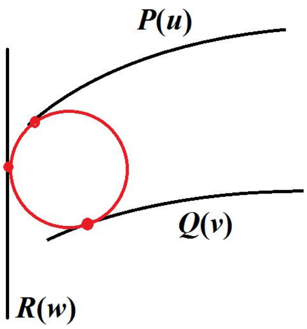


图 2.22 三边倒圆在叶片截面线建模中的应用

【例】 三边倒圆。已知平面曲线 $\boldsymbol{P}(u),\boldsymbol{Q}(v),\boldsymbol{R}(w)$ 及初始切点 $\boldsymbol{P}(u_0),\boldsymbol{Q}(v_0),\boldsymbol{R}(w_0)$，求圆心 $\boldsymbol{O}(x,y)$ 和半径 r，使对应的圆与三条曲线相切。这里曲线是用矢量的形式表示的，例如 $\boldsymbol{P}(u)=\begin{pmatrix}\cos t\\ \sin t\end{pmatrix}$。

解： 根据曲线和圆相切的几何条件可以列出方程组：

$$
\begin{cases}
\left(\boldsymbol{P}(u)-\boldsymbol{O},\boldsymbol{P}_u(u)\right)=0 \\
\left(\boldsymbol{Q}(v)-\boldsymbol{O},\boldsymbol{Q}_v(v)\right)=0 \\
\left(\boldsymbol{R}(w)-\boldsymbol{O},\boldsymbol{R}_w(w)\right)=0 \\
\left(\boldsymbol{P}(u)-\boldsymbol{O},\boldsymbol{P}(u)-\boldsymbol{O}\right)=r^2 \\
\left(\boldsymbol{Q}(v)-\boldsymbol{O},\boldsymbol{Q}(v)-\boldsymbol{O}\right)=r^2 \\
\left(\boldsymbol{R}(w)-\boldsymbol{O},\boldsymbol{R}(w)-\boldsymbol{O}\right)=r^2
\end{cases}
\tag{2.34}
$$

在此非线性方程组中，变量为u,v,w,x,y,r，前 3 个方程表示相切，后 3 个方程表示切点到圆心距离等于半径，$(,)$表示矢量的内积。令

$$a(u,v,w,x,y,r)=\left(\boldsymbol{P}(u)-\boldsymbol{O},\boldsymbol{P}_u(u)\right) \tag{2.35}$$

$$b(u,v,w,x,y,r)=\left(\boldsymbol{Q}(v)-\boldsymbol{O},\boldsymbol{Q}_v(v)\right) \tag{2.36}$$

$$c(u,v,w,x,y,r)=\left(\boldsymbol{R}(w)-\boldsymbol{O},\boldsymbol{R}_w(w)\right) \tag{2.37}$$

$$d(u,v,w,x,y,r)=\left(\boldsymbol{P}(u)-\boldsymbol{O},\boldsymbol{P}(u)-\boldsymbol{O}\right)-r^2 \tag{2.38}$$

$$e(u,v,w,x,y,r)=\left(\boldsymbol{Q}(v)-\boldsymbol{O},\boldsymbol{Q}(v)-\boldsymbol{O}\right)-r^2 \tag{2.39}$$

$$f(u,v,w,x,y,r)=\left(\boldsymbol{R}(w)-\boldsymbol{O},\boldsymbol{R}(w)-\boldsymbol{O}\right)-r^2 \tag{2.40}$$

构造迭代公式，

$$
\begin{bmatrix}
a_u & a_v & a_w & a_x & a_y & a_r \\
b_u & b_v & b_w & b_x & b_y & b_r \\
c_u & c_v & c_w & c_x & c_y & c_r \\
d_u & d_v & d_w & d_x & d_y & d_r \\
e_u & e_v & e_w & e_x & e_y & e_r \\
f_u & f_v & f_w & f_x & f_y & f_r
\end{bmatrix}
\begin{bmatrix}
\Delta u \\ \Delta v \\ \Delta w \\ \Delta x \\ \Delta y \\ \Delta r
\end{bmatrix}
=
\begin{bmatrix}
-a \\ -b \\ -c \\ -d \\ -e \\ -f
\end{bmatrix}
\tag{2.41}
$$

u,v,w的初始值是已知的，过$\boldsymbol{P}(u_0),\boldsymbol{Q}(v_0),\boldsymbol{R}(w_0)$三个点做圆得到圆心和半径的初始值$x_0,y_0,r_0$。通过上面的迭代公式得到满足精度要求的圆心和半径：

$$
\begin{cases}
u_{n+1} = u_n + \Delta u \\
v_{n+1} = v_n + \Delta v \\
w_{n+1} = w_n + \Delta w \\
x_{n+1} = x_n + \Delta x \\
y_{n+1} = y_n + \Delta y \\
r_{n+1} = r_n + \Delta r
\end{cases} \tag{2.42}
$$

2.10 总结

用数值方法求解非线性方程，优点是：(1)算法通用性强，(2)算法实现简单，缺点是：(1)精度低，(2)效率低，(3)有时给出的解不合理。关于非线性方程及非线性方程组，不存在通用的求解方法，确定解的结构、范围最困难，可根据初始值用牛顿法迭代得到单个解。在数值计算中，迭代是一种最常用的思路。

练习题：

1.定点法的收敛条件是什么？

2.输入函数 $f(x)$, 区间[a,b]，容差 e>0,最大迭代次数 MAXIT，试绘制二分法求解过程的算法流程图。

3.用二分法在区间 $[0,1]$ 内求解 $f(x)=e^x-2$（自定合理的收敛条件，要求至少迭代 10 次），（1）用 C 语言实现求解算法；（2）根据收敛速度的定义估算此迭代过程的收件速度；（3）给出用双精度浮点数求解此方程所能达到的最高精度的根。

4.在 2.3 节中给出的收敛速度定义是否适用于二分法？

5.输入 $x=\varphi(x)$、[a,b]区间及收敛容差 ε，绘制出定点法流程图、编程实现通用算法并做完整测试（使用埃特金加速法）。

6.牛顿迭代的收敛条件是什么？

7.编写一个求解一元二次方程的算法，要求用修正的求根公式得到根

之后，在需要的情况下用牛顿法迭代提高解的精度，参考下面产生随机数的代码，用随机数验证该算法。

程序示例 2.9 生成[0,1]随机数的程序

```
#include "stdafx.h"
#include "stdlib.h"
#include "time.h"

int main()
{
    srand((unsigned)time(NULL)); // 用当前时间设定种子
    double r = (double)rand()/RAND_MAX ; // 生成一个[0,1]内的随机数
    return 0;
}
```

8. 基于牛顿下山法用 C 语言实现求二维点 (x_0, y_0) 到椭圆（方程为：$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$）最近距离的算法，并用随机数验证算法的有效性。点到椭圆最近距离的相关几何关系见下图所示：

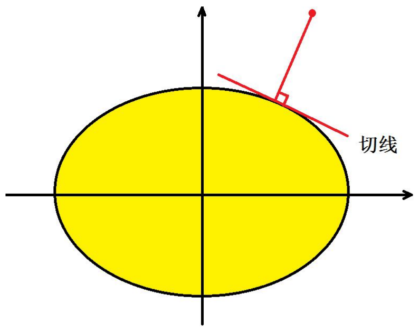


图 2.23 点到椭圆最近距离

9.（1）给定 f_0, f_1, f_2, f_3，求系数 a,b,c,d 使有理多项式 $\lambda(t)=\frac{t+d}{at^2+bt+c}$ 满足当 $t=0$ 时函数值及 1 至 3 阶导数为 f_0, f_1, f_2, f_3；（2）试以（1）的结果为依据构造出求解非线性方程的迭代公式，并分析收敛速度。