第四章 方程求根

4.7　fzerotx，feval

4.8　fzerogui

4.9　寻求函数为某个值的解和反向插值

4.10　最优化和fmintx

4.7 fzerotx，feval

在MATLAB中函数fzero可实现zeroin算法

fzero函数除了基本算法外，还包括一下四项功能：

1、在它开始的部分，使用一个输入的初始估计值，并寻找使函数正负号发生变化的一个区间；

2、由函数f（x）返回的值将被检验，是否是无穷大、NaN（Not a Number的缩写，NaN是一个预定义的常量，表示“不明确的数值结果”）、或者复数；

3、可以改变默认的收敛阈值；

4、也可以要求得到更多的输出，例如调用函数求值的次数。

随本书一起的zeroin算法的版本是fzerotx，它由fzero简化而来，去掉了大多数附带的功能，而保留了zeroin主要的用途。

第一类的零阶贝塞尔函数J_0（x）

- 第一类贝塞尔函数（Bessel function of the first kind），又称贝塞尔函数（Bessel function），简称为J函数，记作J_α。
- 第一类α阶贝塞尔函数$J_\alpha(x)$是贝塞尔方程当α为整数或α非负时的解，须满足在x = 0 时有限。另一种定义方法是通过它在x = 0 点的泰勒级数展开（或者更一般地通过幂级数展开，这适用于α为非整数）：

$$J_\alpha(x)=\sum_{m=0}^{\infty}\frac{(-1)^m}{m!\Gamma(m+\alpha+1)}(\frac{x}{2})^{2m+\alpha}$$

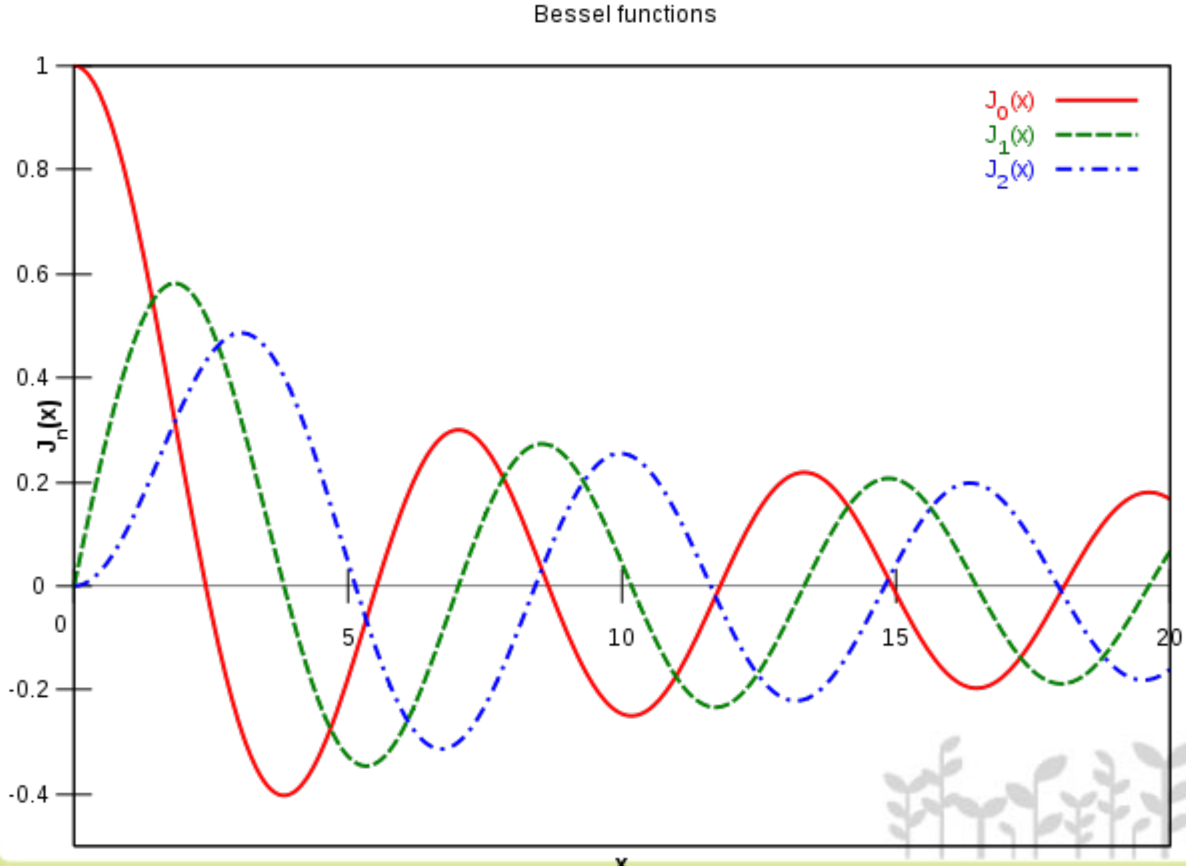
α=0时，

$$J_0(x)=\sum_{m=0}^{\infty}\frac{(-1)^m}{m!\Gamma(m+1)}(\frac{x}{2})^{2m}$$

上式中Γ(z)为Γ函数（它可视为阶乘函数向非整型自变量的推广）。

下图是0阶、1阶和2阶的贝塞尔函数$J_\alpha(x)$的图像$\alpha=(0,1,2)$

第一类贝塞尔函数的形状大致与按速率 $1/\sqrt{x}$ 衰减的正弦或余弦函数类似，但它们的零点并不是周期性的，另外随着x的增加，零点的间隔会越来越接近周期性。



- 用第一类的零阶贝塞尔函数$J_0(x)$说明fzerotx是怎样工作的。$J_0(x)$可通过MATLAB命令besselj(0，x) 得到。

 运行如下程序，就能在MATLAB中得到第一类的零阶贝塞尔函数$J_0(x)$的图像。

```
x=0:pi/50:10*pi;
y=besselj(0,x);
plot(x,y,'-')
```

- 下面的程序能求出$J_0(x)$的前10个零解，并画出图4-2（除了图中红色的‘x’，后面将加上）。

```
bessj0=inline('besselj(0,x)');
for n = 1:10
z(n) = fzerotx(bessj0,[(n-1) n]*pi);
end
```

```
x = 0:pi/50:10*pi;
y = bessj0(x);
plot(z,zeros(1,10),'o',x,y,'-')
line([0 10*pi],[0 0],'color','black')
axis([0 10*pi -0.5 1.0])
```

从图中可以看出，J_0（x）的图形很像是cos（x）的幅值和频率经过调制后的版本，相邻两个零解的距离近似等于π。

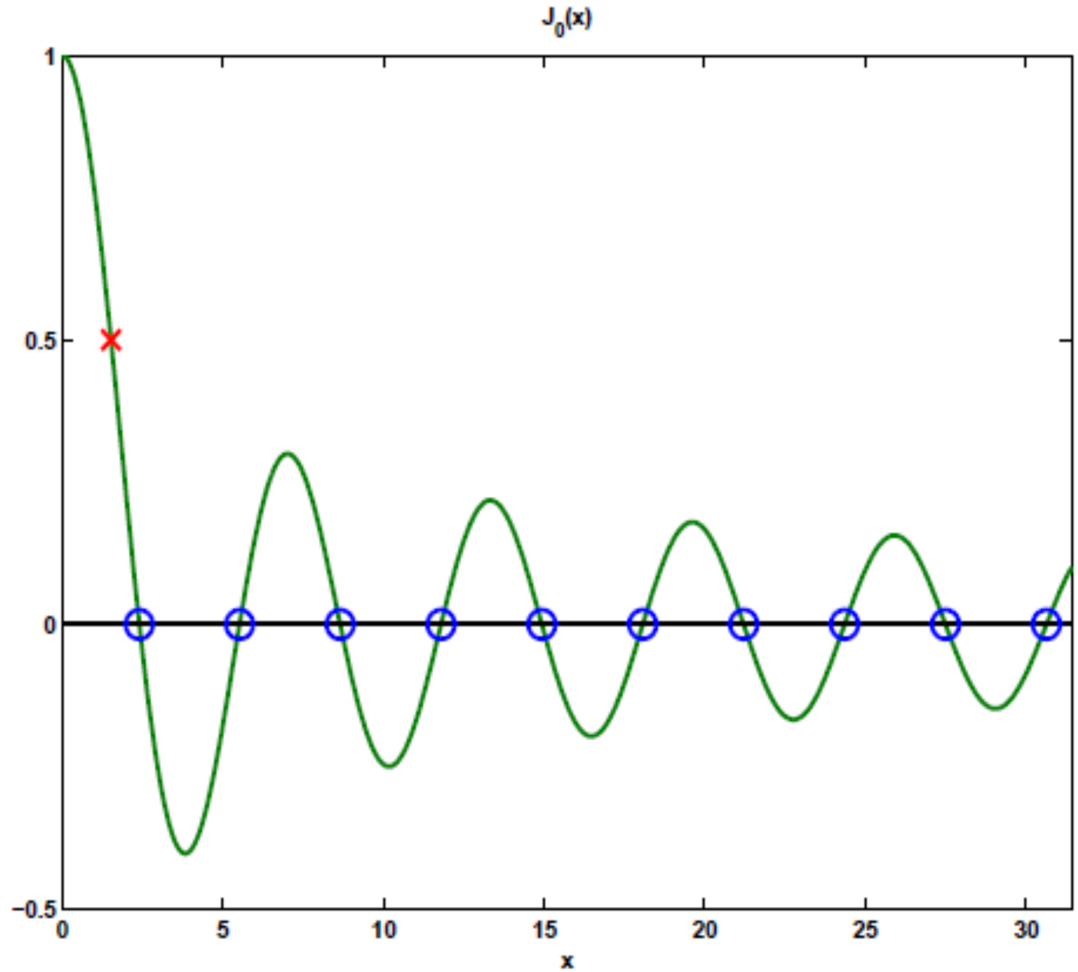


图4-2

- 函数fzerotx有两个输入参数：

  第一个参数指定要计算零解的函数F(x)，

  第二个参数指定初始的搜索区间[a,b]。

  以另一个函数为参数的函数，fzerotx也是MATLAB函数的函数的例子，ezpot是另外一个例子。本书的其他章—第6章，数值积分；第7章，常微分方程；甚至第9章，随机数—中介绍的名字含“tx”和“gui”的M文件也是函数的函数。

- 一个函数作为参数传递给另一个函数，可采用的方式有以下几种：

  函数句柄，

  内嵌对象，

  匿名函数。

函数句柄

定义：在一个内部函数或定义于M文件的函数的名字前面加一个‘@’符号，下面是几个例子：

@cos

@humps

@bessj0

其中bessj0.m是一个含两行代码的M文件。

```
function y=bessj0（x）
y=besselj（0，x）
```

这样，这些句柄就可以用作函数的函数输入参数

z=fzerotx（@bessj0，[0，pi]）

注意@besselj也是一个合法的函数句柄，只是它对应一个带两个输入参数的函数。

内嵌对象

定义：内嵌对象是一种定义简单函数的方法，它不需要再生成新的文件。下面是几个例子：

```
F=inline('cos(pi*t)')
F=inline('z^3-2*z-5')
F=inline('besselj(0,x)')
```

内嵌对象可以用作函数的函数的输入参数,就像

```
z=fzerotx(F,[0,pi])
```

这么使用。内嵌对象也可用来直接计算函数的值，下面是一个例子。

```
residual=F(z)
```

匿名函数

定义：类似@（arguments） expression的结构定义了函数句柄，但没有给它一个名字，所以称为匿名函数。

从MATLAB第7版开始，内嵌对象将被匿名函数这个更强大的结构代替。在MATLAB第7版中，仍允许使用内嵌对象，但推荐使用匿名函数，因为它能生成更高效率的程序代码。上面的一些例子变为

```
F=@(t) cos(pi*t)
F=@(z) z^3-2*z-5
F=@(x) besselj(0,x)
```

M文件、内嵌对象和匿名函数，可以定义超过一个输入参数的函数。在本节讨论的问题中，这些附加参数的值可以通过fzerotx传递给目标函数。这些值在函数求根的迭代过程中保持不变，因此我们可以寻找函数值为特定的y时x的解，而不仅仅是求函数值为零的解。例如，考虑方程

$$J_0(\xi)=0.5$$

在MATLAB第6版中，定义一个带两个或三个参数的内嵌对象。

F=inline('besselj(0,x)-y','x','y')

或 B=inline('besselj(n,x)-y','x','n','y')

在MATLAB第7版中，定义一个带两个或三个参数的匿名函数。

F=@(x,y)besselj(0,x)-y

或 B=@(x,n,y)besselj(n,x)-y

然后，执行

xi=fzerotx(F,[0,2],.5)或xi=fzerotx(B,[0,2],0,.5)

得到结果为 xi=

1.5211

varargin可以看做“Variable length input argument list”的缩写。在matlab中, varargin提供了一种函数可变参数列表机制。就是说， 使用了“可变参数列表机制”的函数允许调用者调用该函数时根据需要来改变输入参数的个数。

在图4-2中，用‘x’标记了上述解对应的点（ξ，J_0（ξ））。

在MATLAB第6版中，可以使用feval对函数参数求值。表达式feval(F,x,...) 等价于F(x,...)

它们的区别在于，使用feval时，允许F作为一个被传递来的参数。在MATLAB第7版中，feval就不再需要了。

fzerotx程序开始的一段注释内容如下:

```
function b = fzerotx(F,ab,varargin);
%FZEROTX  Textbook version of FZERO.
%  x = fzerotx(F,[a,b]) tries to find a zero of F(x) between
%  a and b.  F(a) and F(b) must have opposite signs.
%  fzerotx returns one endpoint of a small subinterval of
%  [a,b] where F changes sign.
%  Additional arguments, fzerotx(F,[a,b],p1,p2,...),
%  are passed on, F(x,p1,p2,...).
```

- 第一段代码对定义搜索区间的变量a、b和c初始化，在初始区间的端点处对函数F求值。

```
a = ab(1);
b = ab(2);
fa = F(a,varargin{:});
fb = F(b,varargin{:});
if sign(fa) == sign(fb)
   error('Function must change sign on the interval')
end

c = a;
fc = fa;
d = b - c;
e = d;
```

- 下面是主循环的开始。在每次迭代步的开始，先对a、b和c重新排列，使它们满足zeroin算法中描述的条件。

- 这部分是收敛条件判断，并可能从循环中退出。

```
m = 0.5*(a - b);
tol = 2.0*eps*max(abs(b),1.0);
if (abs(m) <= tol) | (fb == 0.0),
   break
end
```

- 下部分代码是在二分法和两种基于插值的方法间进行选择。

```
% Choose bisection or interpolation
if (abs(e) < tol) | (abs(fc) <= abs(fb))
   % Bisection
   d = m;
   e = m;
else
   % Interpolation
   s = fb/fc;
   if (a == c)
      % Linear interpolation (secant)
      p = 2.0*m*s;
      q = 1.0 - s;
```

```
while fb ~= 0

   % The three current points, a, b, and c, satisfy:
   %    f(x) changes sign between a and b.
   %    abs(f(b)) <= abs(f(a)).
   %    c = previous b, so c might = a.
   % The next point is chosen from
   %    Bisection point, (a+b)/2.
   %    Secant point determined by b and c.
   %    Inverse quadratic interpolation point determined
   %    by a, b, and c if they are distinct.

   if sign(fa) == sign(fb)
      a = c;  fa = fc;
      d = b - c;  e = d;
   end
   if abs(fa) < abs(fb)
      c = b;    b = a;    a = c;
      fc = fb;  fb = fa;  fa = fc;
   end
```

```
    else
       % Inverse quadratic interpolation
       q = fc/fa;
       r = fb/fa;
       p = s*(2.0*m*q*(q - r) - (b - c)*(r - 1.0));
       q = (q - 1.0)*(r - 1.0)*(s - 1.0);
    end;
    if p > 0, q = -q; else p = -p; end;
    % Is interpolated point acceptable
    if (2.0*p < 3.0*m*q - abs(tol*q)) & (p < abs(0.5*e*q)
       e = d;
       d = p/q;
    else
       d = m;
       e = m;
    end;
 end
```

- 最后一部分代码为下一次迭代步计算F。

```
    % Next point
    c = b;
    fc = fb;
    if abs(d) > tol
       b = b + d;
    else
       b = b - sign(b-a)*tol;
    end
    fb = F(b,varargin{:});
end
```

4.8 fzerogui

M文件fzerogui用图形界面显示了zero算法和fzerotx程序的执行过程。在迭代的每一步，用户有机会用鼠标选择下一个点的位置，其中，一直包括屏幕上显示为红色的区间二分点。

如果当前a、b和c三个点的位置互不相同时会显示一个蓝色的点，它是由IQI算法得到的下一个近似解。当a=c，也就是只有两个互不相同的点时，屏幕上会显示由割线法得到的一个绿色的点。同时，屏幕上也会用一条虚线画出

f（x），但是算法并不“知道”虚线上的其他函数值。你可以选择任何一点，作为下一个近似解，而不一定遵循zeroin算法选择二分点或插值点。你甚至可以直接去选择虚线与坐标轴的交点。

通过贝塞尔函数寻找第一个零解来演示fzerogui是如何工作 的。

可知$J_0(x)$的第一个局部极小值在x=3.83附近，执行

```
bessj0=inline('besselj(0,x)');
z=fzerogui(bessj0,[0 3.83])
```

后开始几步的情况。

1、一开始，b=c，所以有区间二分点和割线交点两种选择（如图4-3）。

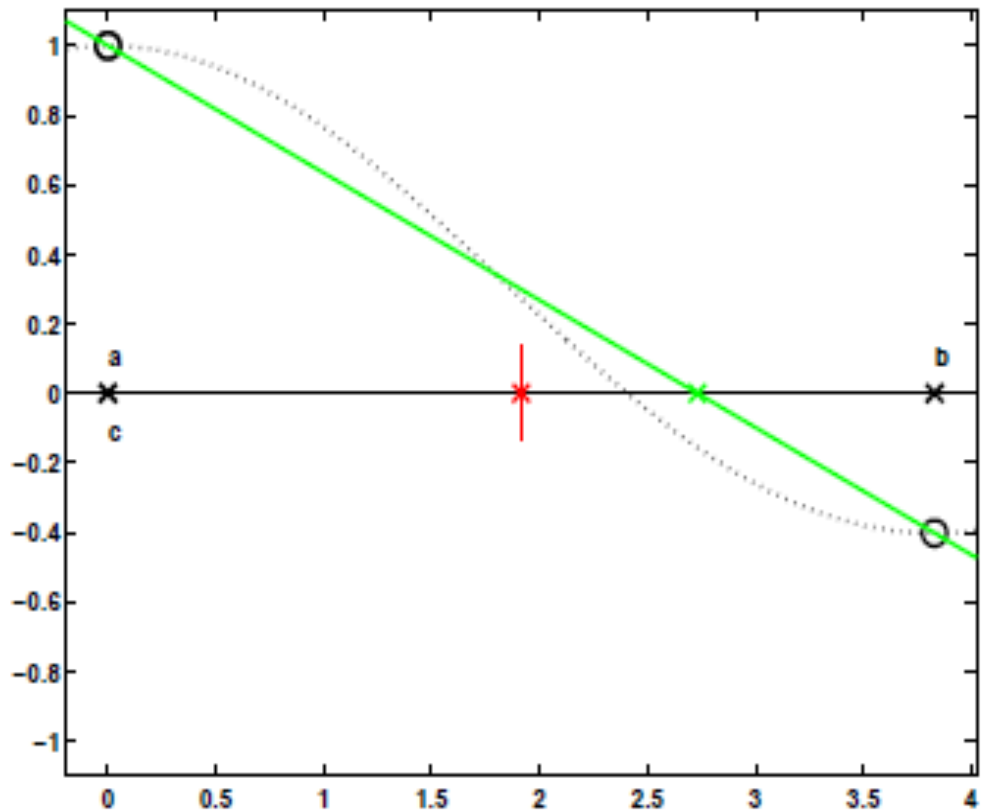


图4-3

2、如果选择割线点，那么b点移动到这里，并计算x=b时的J0(x).这时有三个不同的点，因此下一步在区间二分点和IQI算法得到的点之间选择一个（如图4-4）

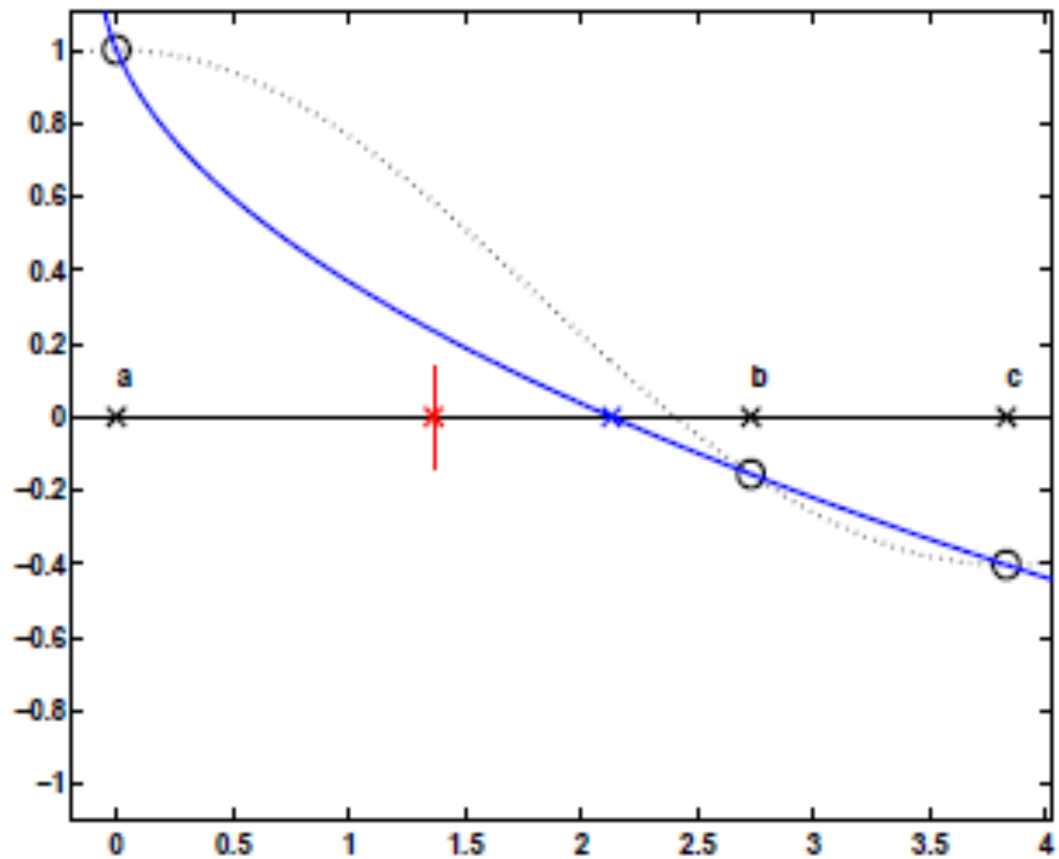


图4-4

3、如果选择IQI算法得到的点，搜索区间变小，图形用户界面也相应地放大，下一步仍须在区间二分点和割线交点之间选择一个，这时可以看到，这两点碰巧非常接近（如图4-5）

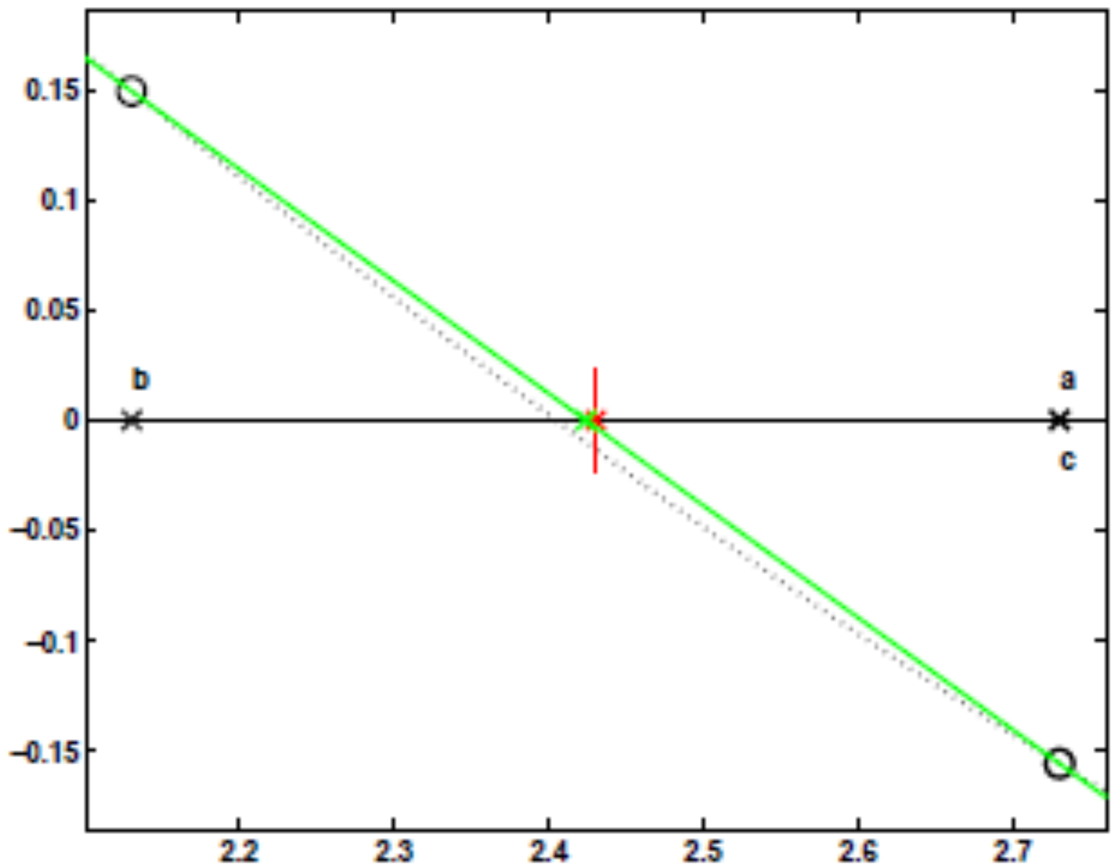


图4-5

4、现在可以选择两个点中的任何一个，或者也可以选靠近它们的其他点。再这样执行两步，区间不断缩小并达到图4-6所示的情形。这是算法快接近收敛时出现的典型情况，函数的图形看上去非常像一条直线，而割线交点或IQI算法得到的点远比区间二分点快得多的收敛过程。再经过几步操作，使函数值改变正负号的区间 长度变得非常小（相对于原始的长度）同时算法停止，将最后得到的b作为结果返回。

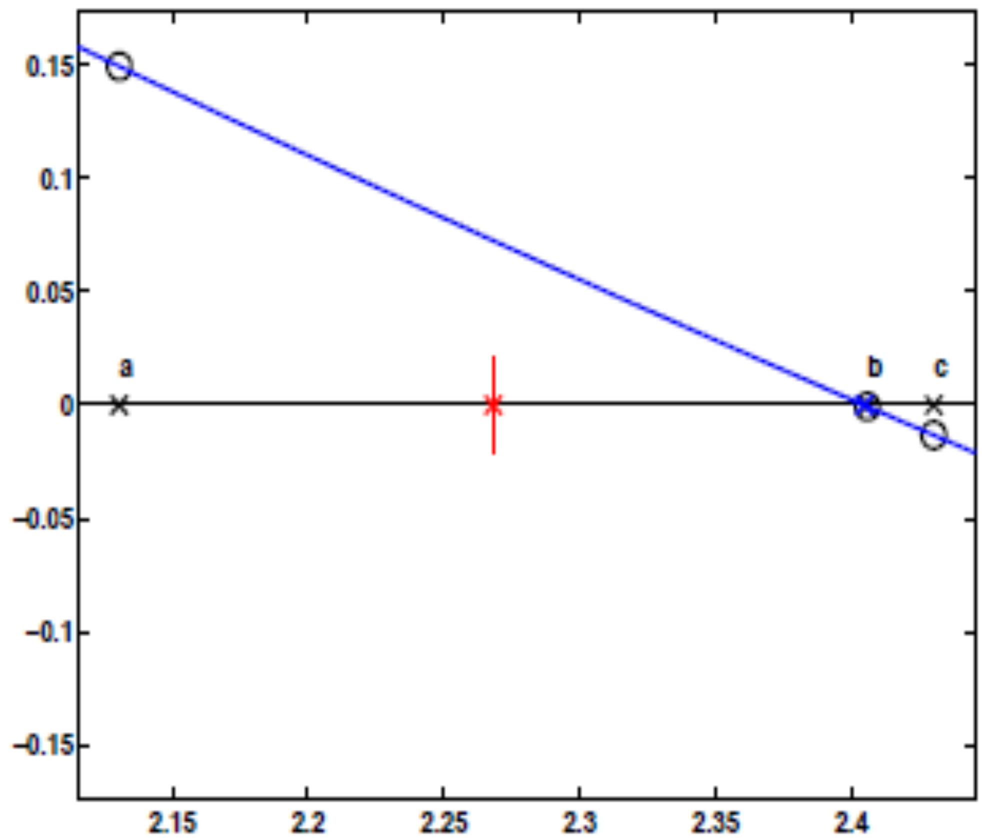


图4-6

4.9 寻找函数为某个值的解和反向插值

下面两个问题非常相似：

- 给定一个函数F(x)和值η，求ξ使得F(ξ)=η；
- 给定对未知函数F(x)采样得到的一些数据点（x_k，y_k），以及一个值η，求ξ使得F(ξ)=η。

对于第一个问题，我们可以对任意的x计算F(x)，因此可以使用一个函数零值求解器，求解转后后的函数f(x)=F(x)-η。这将得到ξ，使得f(ξ)=0,因此F(ξ)=η。

对于第二个问题，我们需要做某种插值函数，比如根据pchiptx（x_k，y_k，x）或splinetx（x_k，y_k，x）得到的。这种方法通常能够较好地完成目标，但计算代价较大，因为零值求解器要反复地计算插值函数的值。利用本书所带的程序，这将导致多次计算插值函数的系数和反复确定合适的区间位置。

对有些情况，更好的一个方法是反向插值，它使用pchip和spline算法时，将x_k和y_k的角色进行颠倒。

这个方法要求：给定的y_k具有单调性，或者至少y_k的某个包含目标值η的子集单调。按这个方法生成出另一个分段多项式，记为Q(y)，使得$Q(y_k)=x_k$.这样就没有必要使用函数零值求解器了，简单地在$y=\eta$处计算$\xi=Q(y)$即可。

如何在这两个方法中进行选择，主要依赖于已知的数据是否能很好地用分段多项式插值所表示。也就是说，要看使用插值时把x当成独立自变量好，还是把y当成独立自变量好。

4.10 最优化和fmintx

寻找函数的最大值、最小值的工作，与求函数的零解紧密相关，本节介绍一个类似于zeroin的算法，它可找出一个单变量函数的局部极小值。

问题的定义中包括一个函数f（x）和它所在的区间[a，b]，目标是求一个x值，它使f（x）在给定区间上达到局部最小。如果这个函数是幺模的，即在这个区间上仅有一个局部极小，那么我们的算法就可以找到它。但如果有多个局部极小，这个算法只能找到其中一个，而这一个也未必是整个区间上的极小值。此外，区间两端点中的某一个也可能是极小点。

能不能使用区间二分法？

不能使用区间二分法，因为即使我们知道f(a)、f(b)和f((a+b)/2)的值，也无法确定该丢弃哪半个区间，而保证剩下的区间包括最小值。

能不能使用三等分的方法？有哪些缺点？

将区间三等分的方法是可行的，但效率不高。令h=（b-a）/3,

则u=a+h和v=b-h将区间分为三等份。假设我们发现f(u)<f(v),

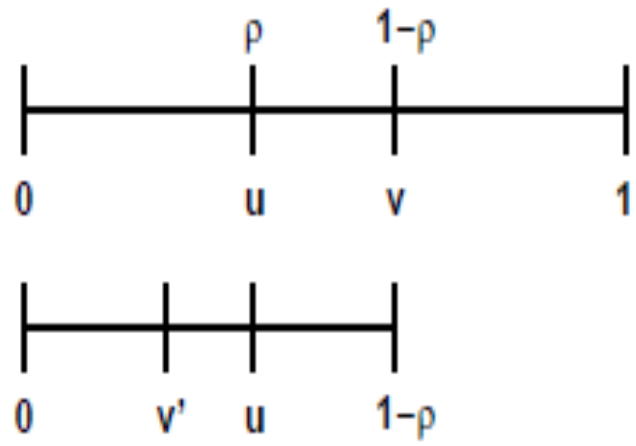
那么就用v的值代替b，从而将区间长度缩短为原来的三分之二，同时扔保证缩小后的区间包括极小值。

缺点：然而，由于u在新区间的中点处，它在下一步没有用，这样每一步都需要计算函数值两次。

黄金分割搜索法

黄金分割搜索法是类似于二分法，求最小值的自然算法。

它的主要思想如图4-7所示，其中a=0、b=1。令h=ρ(b-a),ρ为比1/3略大的量，我们将介绍如何确定它。然后u=a+h，v=b-h将区间分为三个不等长的部分。



下面第一步是计算f(u)和f(v)。假设我们发现f(u)<f(v),那么极小值点就应该在a和v之间，需要将b替换为v并重复上面的过程。如果选择了一个正确的ρ值，使点u的位置正好合适，可在下一步中使用。这样除第一步外，后继每步计算只需计算一次函数值。

定义这个ρ值的方程为 $$\frac{\rho}{1-\rho}=\frac{1-\rho}{1}$$

或者

$$\rho^2-3\rho+1=0$$

方程的解为

$$\rho=2-\phi=(3-\sqrt{5})/2\approx 0.382$$

这里ϕ是黄金分割比。
使用黄金分割搜索，区间的长度随每步计算，以 ϕ-1≈0.618 的比例缩小。经过

$$\frac{-52}{\log_2(\phi-1)}\approx 75$$

步后，区间的长度将大致减小为原始长度的eps倍，这个eps是IEEE双精度浮点数计算舍入误差的大小。

最优化搜索

经过开始的一些步后，通常有足够的历史信息，给出区间内三个不同的点，以及对应的函数值。如果由这三个点插值产生的抛物线的极小值点落在这个区间内，那么下一个点通常选择这个极小值点，而不是区间的黄金分割点。黄金分割搜索和抛物线插值相结合，提供了一个一维优化问题的可靠而有效的求解方法。

最优化搜索过程停止判断的设置是需要技巧的。在f（x）的极小值点，一阶导数f'(x)为零。因此在极小值点附近，f（x）近似一个没有一次项的二次函数：

$$f(x) \approx a + b(x-c)^2 + \ldots$$

函数极小值发生在x=c处，并且其值为f(c)=a。如果x靠近c，比如x≈c+δ而δ很小，那么

$$f(x) \approx a + b\delta^2$$

当计算函数值时，x上的微小改变会被平方。如果a和b都不等于零，且大小差不多，那么停止判断中应该包括sqrt（eps），因为x上任何更小的改变都不会影响f（x）的值。但如果a和b有不同的数量级，或者a和c中有一个近似于零，那么使用eps乘以区间长度比sqrt（eps）更为合适。

fmintx

MATLAB中两个关于求局部极小值的函数的函数

1、函数的函数fminbnd，它使用黄金分割搜索和抛物线插值相结合的方法求单实变量、单实值函数的局部极小值。这个函数基于Richard Brent写的一个Fortran子程序[12]。

2、函数的函数fminsearch，它使用一种称为Nelder-Meade单纯形的算法来求一个多实变量、单实值函数的局部极小值。

MATLAB中的最优化工具盒，收集了一些求解其他种类优化问题的程序，它们包括有约束优化、线性规划以及大规模稀疏优化问题。

本书配套的NCM程序包中有一个函数fmintx，它使fminbnd的简化版本。简化措施之一是有关停止判据的，

当区间长度小于指定参数tol时，就停止搜索过程。tol默认值为10^{-6}在完整的fminbnd程序中使用了复杂的停止判据，其中包含了对x和f（x）的相对的、和绝对的阈值没有确定。

函数humps

MATLAB的demos目录下，有一个名为humps的函数，它用于演示MATLAB中绘图、积分和方程求根的有关命令。这个函数的表达式为

$$h(x)=\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}$$

使用命令

```
F=inline('-humps(x)');
fmintx(F,-1,2,1.e-4)
```

按如下的输出一步步搜索humps函数的极小值，搜索点同时示于图4-8。可以看到，在二、三、七步使用的是黄金分割搜索，而当搜索靠近极小值点时，就完全使用抛物线插值方法了。

step	x	f(x)
init:	0.1458980337	-25.2748253202
gold:	0.8541019662	-20.9035150009
gold:	-0.2917960675	2.5391843579
para:	0.4492755129	-29.0885282699
para:	0.4333426114	-33.8762343193
para:	0.3033578448	-96.4127439649
gold:	0.2432135488	-71.7375588319
para:	0.3170404333	-93.8108500149
para:	0.2985083078	-96.4666018623
para:	0.3003583547	-96.5014055840
para:	0.3003763623	-96.5014085540
para:	0.3003756221	-96.5014085603

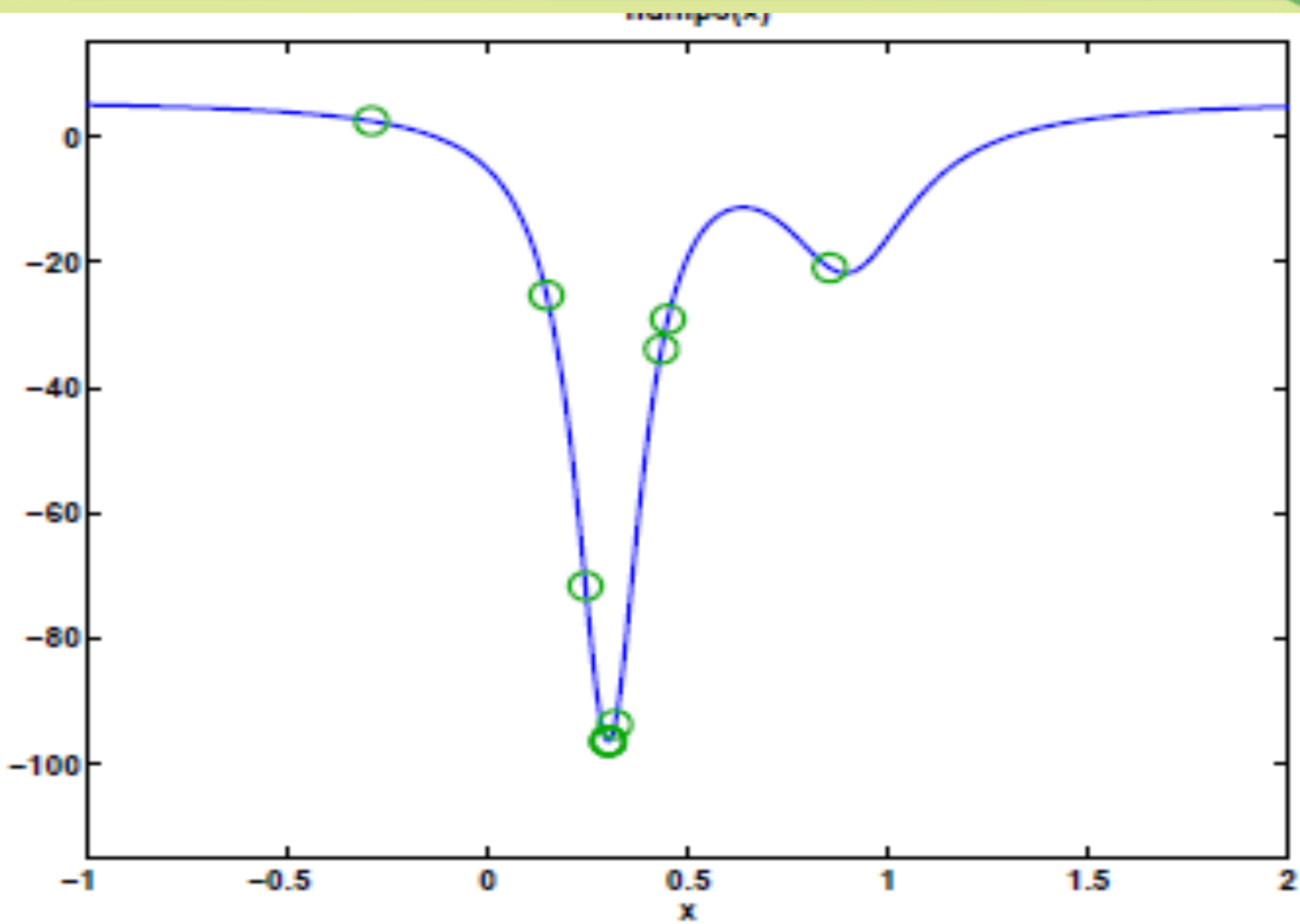


图4-8