# 第2章　非线性方程求根

**陈路**　　CHENLU.SCIEN@GMAIL.COM
201800301206

本文首先简要总结非线性方程求根的相关知识，包括非线性方程求根的基本概念和常见算法及其收敛速率。进而对一些常用算法进行编程实现，并应用到实际问题体会不同算法的优劣性。

## 1. 非线性方程求根知识总结

### 1.1 不动点迭代法

#### 1.1.1 迭代

迭代是指重复执行一个计算过程，直到找到答案. 首先需要一个函数$g(x)$和起始点$p_0$，然后依据迭代规则$p_{k+1} = g(p_k)$可得到迭代序列$\{p_k\}$. 如果这一序列中的数趋向于某个极限，那么就达到了我们的求解目的.

#### 1.1.2 不动点迭代

**定义 1**
函数$g(x)$的一个**不动点**是指一个实数$P$，满足$P = g(P)$.

**定义 2**
迭代$p_{n+1} = g(p_n)$，其中$n = 0, 1, \cdots$，称为**不动点迭代**.

**定理 1**
设$g$是一个连续函数，且$\{p_n\}_{n=0}^{\infty}$是由不动点迭代生成的序列.
如果$\lim_{n\to\infty} p_n = P$，则$P$是$g(x)$的不动点.

**不动点存在性定理**

**定理 2**
设函数$g \in C[a, b]$.
如果对于所有$x \in [a, b]$，映射$y = g(x)$的范围满足$y \in [a, b]$，那么函数$g$在$[a, b]$内至少有一个不动点.

此外，设$g'(x)$定义在$(a,b)$内，且对于所有$x \in (a,b)$，存在常数$0 < K < 1$使得$|g'(x)| \leqslant K < 1$，则函数$g$在$[a,b]$内有唯一的不动点.

**不动点迭代收敛条件**

**定理 3**
设有（i）$g, g' \in C[a,b]$，（ii）$K$是一个大于0的常数，（iii）$p_0 \in (a,b)$，（iv）对于所有$x \in [a,b]$，有$g(x) \in [a,b]$.
如果对于所有$x \in [a,b]$，有$|g'(x)| \leqslant K < 1$，那么对于任意初始值$p_0$，迭代$p_n = g(p_{n-1})$将收敛到唯一的不动点$P \in [a,b]$.

设函数$g$满足定理3中的所有条件，当用$p_n$去近似$P$时，引入的**误差边界**为：

$$|P - p_n| \leqslant K^n |P - p_0| \text{ 对于所有 } n \geqslant 1 \tag{1}$$

且

$$|P - p_n| \leqslant \frac{K^n |p_1 - p_0|}{1 - K} \text{ 对于所有 } n \geqslant 1 \tag{2}$$

### 1.2 全局收敛法

对于任意函数$f(x) = 0$，全局收敛法依赖于初始区间$[a,b]$，满足输入函数值$f(a), f(b)$的符号相反. 一旦给定了这个区间，无论区间多大，通过迭代总能找到输入函数一个根. 然而，如果函数$f(x) = 0$在区间$[a,b]$上有多个根，则必须使用不同的起始区间寻找每个根.

#### 1.2.1 二分法

假设函数$f(x)$在区间$[a,b]$上连续且两端点处函数值$f(a), f(b)$符号相反. 二分法判定过程的第一步是选择中点$c = (a+b)/2$，然后分析可能存在的3种情况：

1. 如果$f(a)$和$f(c)$符号相反，则在区间$[a,c]$内存在零点;
2. 如果$f(c)$和$f(b)$符号相反，则在区间$[c,b]$内存在零点;
3. 如果$f(c) = 0$，则$c$是零点

**定理 1**
设$f \in C(a,b)$，且存在数$r \in [a,b]$满足$f(r) = 0$.
如果$f(a)$和$f(b)$的符号相反，且$\{c_n\}_{n=0}^{\infty}$表示二分法生成的中点序列，则

$$|r - c_n| \leqslant \frac{b-a}{2^{n+1}} \quad \text{其中 } n = 0, 1, \cdots \tag{3}$$

#### 1.2.2 试位法

由于二分法的收敛速度较慢，因此试位法对它进行了改进. 假设函数$f(x)$在区间$[a,b]$上连续且两端点处函数值$f(a), f(b)$符号相反. 二分法使用区间中点进行迭代，如果找到经过点$(a, f(a))$和$(b, f(b))$的割线$L$与$x$轴的交点$(c, 0)$，就可以得到一个更好的近似值，即：

$$c = b - \frac{f(b)(b-a)}{f(b) - f(a)} \tag{4}$$

类似地，会出现3中可能性：

1. 如果$f(a)$和$f(c)$符号相反，则在区间$[a,c]$内存在零点;
2. 如果$f(c)$和$f(b)$符号相反，则在区间$[c,b]$内存在零点;
3. 如果$f(c) = 0$，则$c$是零点

### 1.3 局部收敛法

对于任意函数$f(x) = 0$，局部收敛法要求给定一个接近根的近似值以保证收敛性. 局部收敛法的速度远大于全局收敛法的速度. 一些混合方法首先采用全局收敛法，当迭代逼近根后切换到局部收敛法.

#### 1.3.1 牛顿法

**定理 1**

设$f \in C^2[a,b]$，且存在数$p \in [a,b]$，满足$f(p) = 0$. 如果$f'(p) \neq 0$，则存在一个数$\delta > 0$，对任意初始近似值$p_0 \in [p-\delta, p+\delta]$，使得由如下迭代定义的序列$\{p_k\}_{k=0}^{\infty}$收敛到$p$：

$$p_k = g\left(p_{k-1}\right) = p_{k-1} - \frac{f\left(p_{k-1}\right)}{f'\left(p_{k-1}\right)} \quad \text{其中 } k = 1, 2, \cdots \tag{5}$$

其中，函数$g(x)$定义如下：

$$g(x) = x - \frac{f(x)}{f'(x)} \tag{6}$$

并被称为**牛顿-拉夫森迭代函数**.

#### 1.3.2 割线法

牛顿法中每次迭代需要计算两个函数：$f(p_{k-1})$和$f'(p_{k-1})$，当函数形式非常复杂时，求其导数往往较为费时，因此需要一种和牛顿法收敛速度一样快且无需计算导数的方法，这就是割线法. 割线法包含的公式与试位法公式一致，只是在关于如何定义每个后续项的逻辑判定上不一样. 割线法需要两个靠近零点$(p, 0)$的初始点$(p_0, f(p_0))$和$(p_1, f(p_1))$，定义$p_2$为经过两个初始点

的直线与x轴的交点的横坐标，即

$$p_2 = g(p_1, p_0) = p_1 - \frac{f(p_1)(p_1 - p_0)}{f(p_1) - f(p_0)} \tag{7}$$

根据两点迭代公式可以得到一般项为：

$$p_{k+1} = g(p_k, p_{k-1}) = p_k - \frac{f(p_k)(p_k - p_{k-1})}{f(p_k) - f(p_{k-1})} \tag{8}$$

## 2. 分析讨论题

**问题 1**

求方程$2x^2 + x - 15 = 0$的正根$x* = 2.5$近似值，分别利用如下三种格式编程计算：

1. $x_{k+1} = 15 - 2x_k^2, k = 0, 1, 2, \cdots$，取初始值$x_0 = 2$.
2. $x_{k+1} = \frac{15}{2x_k+1}, k = 0, 1, 2, \cdots$，取初始值$x_0 = 2$.
3. $x_{k+1} = x_k - \frac{2x_k^2+x_k-15}{4x_k+1}, k = 0, 1, 2, \cdots$，取初始值$x_0 = 2$.

依次计算$x_1, x_2, \cdots, x_k, \cdots$，并作图观察解的稳定性、收敛性，并分析原因.

**解**:

不动点迭代算法的代码实现：

```
function [k, p, err, P] = fixpt(g, p0, TOL, N)
% Fixed Point Interation
% Input - g    the iteration function as a string
%       - p0   initial guess for the fixed point
%       - TOL  tolerance
%       - N    the maximum number of iterations
% Output - k   the number of iterations carried out
%        - p   the approximation of the fixed point
%        - err the error in the approximation
%        - P   sequence {p_n}

% Initialization
P(1) = p0; % initial guess
if nargin < 3
  TOL = 1e-3; % tolerance default
end
if nargin < 4
  N = 200; % maximum interation default
end

% Compute fixed point
for k = 2:N
  P(k) = feval(g, P(k-1));
  err = abs(P(k) - P(k-1));
  relerr = err/(abs(P(k)) + eps);
```

Table 1: 问题1.1迭代过程

| 迭代次数 | 根的估计值 | 误差 | 相对误差 |
|---|---|---|---|
| 2 | 7.000000 | 5.000000 | 0.714286 |
| 3 | -83.000000 | 90.000000 | 1.084337 |
| 4 | -13763.000000 | 13680.000000 | 0.993969 |
| 5 | -378840323.000000 | 378826560.000000 | 0.999964 |
| 6 | -2870399806614886 72.000000 | 28703998028264 8352.000000 | 1.000000 |

```
26   p = P(k);
27   fprintf('Interation NO.%d,\t approximation p=%f,\t absolute error=%f,\t relative
       error=%f\n',...
28     k, p, err, relerr);
29   if (err < TOL) || (relerr < TOL)
30     break, end
31 end
32
33 if k == N
34   disp('Maximum number of iterations exceeded');
35 end
36 P = P'; % transpose the vector
37 end
```

**1**. 使用格式“$x_{k+1} = 15 - 2x_k^2, k = 0, 1, 2, \cdots$，初始值$x_0 = 2$，精确度=$10^{-4}$”时，迭代过程及误差如表1. 在此情况下，算法无法收敛.

**2**. 使用格式“$x_{k+1} = \frac{15}{2x_k+1}, k = 0, 1, 2, \cdots$，初始值$x_0 = 2$”时，迭代过程及误差如表2.
在此情况下，算法经过48次迭代后达到精度$10^{-4}$，根的估计值为2.500104.
迭代过程可以通过图1可视化呈现.

Table 2: 问题1.2迭代过程(部分省略)

| 迭代次数 | 根的估计值 | 误差 | 相对误差 |
|---|---|---|---|
| 2 | 3.000000 | 1.000000 | 0.333333 |
| 3 | 2.142857 | 0.857143 | 0.400000 |
| 4 | 2.837838 | 0.694981 | 0.244898 |
| 5 | 2.246964 | 0.590874 | 0.262966 |
| 6 | 2.730287 | 0.483324 | 0.177023 |
| 7 | 2.321775 | 0.408513 | 0.175948 |
| 8 | 2.657902 | 0.336127 | 0.126463 |
| 9 | 2.374995 | 0.282907 | 0.119119 |
| 10 | 2.608700 | 0.233706 | 0.089587 |
| ... | ... | ... | ... |
| 20 | 2.517270 | 0.037851 | 0.015036 |
| 21 | 2.485691 | 0.031578 | 0.012704 |
| 22 | 2.511981 | 0.026290 | 0.010466 |
| 23 | 2.490055 | 0.021926 | 0.008805 |
| 24 | 2.508315 | 0.018259 | 0.007280 |
| 25 | 2.493090 | 0.015225 | 0.006107 |
| 26 | 2.505771 | 0.012681 | 0.005061 |
| 27 | 2.495200 | 0.010572 | 0.004237 |
| 28 | 2.504007 | 0.008807 | 0.003517 |
| 29 | 2.496666 | 0.007341 | 0.002940 |
| 30 | 2.502782 | 0.006116 | 0.002444 |
| ... | ... | ... | ... |
| 40 | 2.500449 | 0.000988 | 0.000395 |
| 41 | 2.499626 | 0.000823 | 0.000329 |
| 42 | 2.500312 | 0.000686 | 0.000274 |
| 43 | 2.499740 | 0.000572 | 0.000229 |
| 44 | 2.500217 | 0.000476 | 0.000191 |
| 45 | 2.499820 | 0.000397 | 0.000159 |
| 46 | 2.500150 | 0.000331 | 0.000132 |
| 47 | 2.499875 | 0.000276 | 0.000110 |
| 48 | 2.500104 | 0.000230 | 0.000092 |

Table 3: 问题1.3迭代过程

| 迭代次数 | 根的估计值 | 误差 | 相对误差 |
|---|---|---|---|
| 2 | 2.555556 | 0.555556 | 0.217391 |
| 3 | 2.500550 | 0.055006 | 0.021997 |
| 4 | 2.500000 | 0.000550 | 0.000220 |
| 5 | 2.500000 | 0.000000 | 0.000000 |

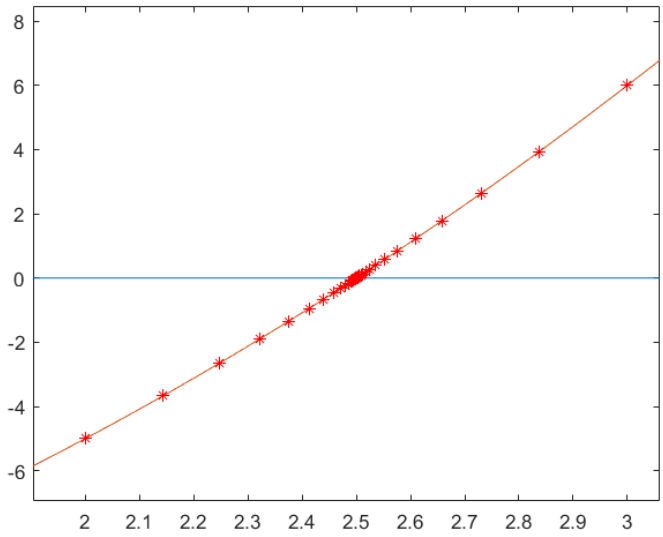


Figure 1: 问题1.2的迭代过程图示

**3**. 使用格式“$x_{k+1} = x_k - \frac{2x_k^2+x_k-15}{4x_k+1}, k = 0, 1, 2, \cdots$，初始值$x_0 = 2$”时，迭代过程及误差如表3 在此情况下，算法经过5次迭代后达到精度$10^{-4}$，根的估计值为2.500000.
迭代过程可以通过图2可视化呈现.

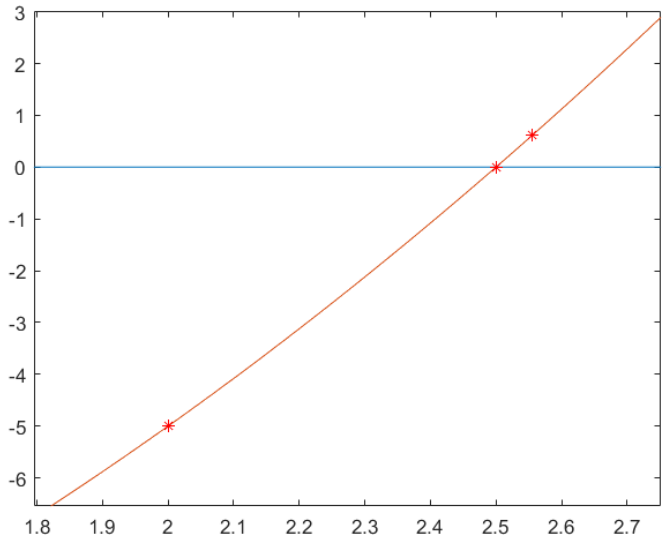


Figure 2: 问题1.3的迭代过程图示

**问题 2**

证明方程$2-3x-\sin(x)=0$在$(0,1)$内有且仅有一个实根，使用二分法求误差不大于0.0005的根及其需要迭代的次数.

**解**:

令$f(x)=2-3x-sin(x)$,则$f'(x)=-3-cos(x)<0$,所以$f(x)$在区间(0,1)内单调递减.

又因为$f(0)=2>0, f(1)=-1-sin1<0$，所以$f(x)$在$(0,1)$内有且仅有一个实根.

二分法的代码实现：

```
function [c, yc, err] = bisect(f, a, b, TOL)
% Bisection method
% Input - f   the input function as a string
%       - a   the left end point of the initial section
%       - b   the right end point of the initial section
%       - TOL tolerance
% Output - c   the zero point of input function
%        - yc  = f(c)
%        - err the error in the approximation

% Initialization
ya = feval(f, a);
yb = feval(f, b);
% Check condition
if ya*yb > 0
  disp('The end-points value of the initial section should not be of the same sign')
    ;
```

```
return
end

% Compute the number of needed iteration round
N = 1 + round((log(b-a)-log(TOL))/log(2));
% Main process
for k = 1:N
  c = (a+b)/2;
  yc = feval(f, c);

  % Print intermediate messages
  fprintf('Interation NO.%d,\t current section = [%f,%f],\t c=%f,\t f(c)=%f\n',...
    k, a, b, c, yc);

  if yc == 0
    a = c;
    b = c;
  elseif yb * yc > 0
    b = c;
    yb = yc;
  else
    a = c;
    ya = yc;
  end

  if (b - a) < TOL
    break, end
  end

  c = (a + b)/2;
  yc = feval(f, c);
  err = abs(b-a);
end
end
```

经过11次迭代后误差为$-2.454600314258926 \times 10^{-4}$.

此时根c= 0.505371093750000

**问题 3**

利用牛顿法求解方程

$$\frac{1}{2} + \frac{1}{4}x^2 - x\sin x - \frac{1}{2}\cos 2x = 0 \tag{9}$$

分别取$x_0 = \frac{\pi}{2}, 5\pi, 10\pi$，使得精度不超过$10^{-5}$. 比较初值对计算结果的影响.

**解**:

牛顿法的代码实现：

```
function [p0, y, k, err] = newton(f, df, p0, delta, epsilon, N)
% Newton Raphson method
```

```
% Input - f        the input function as a string
%       - df       the derivative of f as a string
%       - p0       the initial approximation to zero point of f
%       - delta    tolerance for the value of zero point
%       - epsilon  tolerance for the value of function f at zero point
%       - N        maximum number of iterations
% Output - p0      the approximation to zero point of input function
%         - y      = f(p0)
%         - k      the number of iterations
%         - err    the error in the approximation

fprintf('Initial guess\t p0 = %f\n', p0);

for k = 1:N
  p1 = p0 - (feval(f, p0) / feval(df, p0));
  err = abs(p1 - p0);
  relerr = 2 * err / (abs(p1) + delta);
  p0 = p1;
  y = feval(f, p0);

  % Print intermediate messages
  fprintf('Interation NO.%d,\t p0=%f\t f(p0)=%f\n', k, p0, y);

  if (err < delta) || (relerr < delta) || (abs(y) < epsilon)
    break, end
  end
end
```

当$x_0 = \pi/2$时，经过14次迭代即可满足精度，此时解为$x = 0.000005611212547$
当$x_0 = 5\pi$时，经过40次迭代才可满足精度，此时解为$x = -0.00000051311029$
当$x_0 = 10\pi$时，需要经过273次迭代才可满足精度，此时解为$x = -0.00000543338169$

**问题 4**
已知$f(x) = 5x - e^x$在$(0, 1)$之间有一个实根，试分别利用二分法、牛顿法、割线法、试位法设计相应的计算格式，并编程求解（精确到4位小数）.

**解**:

(1)二分法
二分法的代码实现请见问题2 2.
选取初始点$p_0 = 0, p_1 = 1$，经过14次迭代后达到精度0.0001，此时x=0.2592
(2)牛顿法
牛顿法的代码实现请见问题3 2.
选取初始点$p_0 = 0, p_1 = 1$，只需4次迭代后即可达到精度0.0001，迭代过程为：
-1.0000　0.1588　0.2576　0.2592　0.2592
(3)割线法
代码实现:

```
function [p1, y, k, err] = secant(f, p0, p1, delta, epsilon, N)
% Secant method
% Input - f       the input function as a string
%       - p0      the initial approximation to zero point of f
%       - p1      the initial approximation to zero point of f
%       - delta   tolerance for the value of zero point
%       - epsilon tolerance for the value of function f at zero point
%       - N       maximum number of iterations
% Output - p1     the approximation to zero point of input function
%        - y      = f(p0)
%        - k      the number of iterations
%        - err    the error in the approximation

fprintf('Initial guess\t p0 = %f, p1 = %f\t f(p1)=%f\n', p0, p1, feval(f, p1));

for k = 1:N
  p2 = p1 - feval(f, p1) * (p1 - p0) / (feval(f, p1) - feval(f, p0));
  err = abs(p2 - p1);
  relerr = 2 * err / (abs(p2) + delta);
  p0 = p1;
  p1 = p2;
  y = feval(f, p1);

  % Print intermediate messages
  fprintf('Interation NO.%d,\t p0 = %f, p1 = %f\t f(p1)=%f\n', k, p0, p1, y);

  if (err < delta) || (relerr < delta) || (abs(y) < epsilon)
    break, end
end
end
```

使用割线法时选取初始点应慎重，有的选点会出现不正确的结果。

选取$p_0 = 0.3, p_1 = 0.5$后进行迭代，经过4次迭代产生精度误差不超过0.0001的解x=0.2592

(4)试位法

试位法代码实现:

```
function [c, yc, err] = regula(f, a, b, delta, epsilon, N)
% Regula falsi method
% Input - f       the input function as a string
%       - a       the left end point of the initial section
%       - b       the right end point of the initial section
%       - delta   tolerance for the value of zero point
%       - epsilon tolerance for the value of function f at zero point
%       - N       maximum number of iterations
% Output - c      the zero point of input function
%        - yc     = f(c)
%        - err    the error in the approximation

% Initialization
ya = feval(f, a);
yb = feval(f, b);
% Check condition
```

```
if ya*yb > 0
  disp('The end-points value of the initial section should not be of the same sign')
    ;
return
end

for k = 1:N
  dx = yb*(b - a) / (yb - ya);
  c = b - dx;
  ac = c - a;
  yc = feval(f, c);

  % Print intermediate messages
  fprintf('Interation NO.%d,\t current section = [%f,%f],\t c=%f,\t f(c)=%f\n',...
    k, a, b, c, yc);

  if yc == 0
    break
  elseif yb*yc > 0
    b = c;
    yb = yc;
  else
    a = c;
    ya = yc;
  end

  dx = min(abs(dx), ac);
  if abs(dx) < delta
    break, end
  if abs(yc) < epsilon
    break, end
end

err = abs(b-a)/2;
yc = feval(f, c);
end
```

错位法选取初始点$p_0 = 0, p_1 = 1$，经过13次迭代后达到精度0.0001，x=0.2592