数值分析作业: 一元方程求解

Due on March, 2022

葛雨辰 201800150053

1. 序言

本章节研究一元方程求根问题，依据 2.2 节的第三段话，我认为求根问题的算法分为**直接求根的算法**（二分法，牛顿法，割线法与试错法等）与**通过转化为不动点迭代间接求根的算法**（不动点迭代法，Aitken 加速法，Steffensen 迭代法等）。最后有关多项式有特殊的算法算法，涵盖 Horner 算法，及采取**嵌套方式**去计算多项式在固定点的值与导数值求多项式及导数的值，同时 Muller 求根算法可以很好地逼近复根。

首先声明以下所有程序都**按照方法名称命明**，变量含义自明。

2. 二分法，牛顿法，割线法与试位法对比

2.1 求解三阶 Legendre 方程在 1.2 附近的根

考虑三阶的 Legendre 方程 (作平移变换，方便误差估计)

$$P_3(x) = \frac{1}{2}\big[5(x-1)^3 - 3(x-1)\big] = 0$$

在 1.2 附近的根。观察在 $[0,2]$ 上的方程曲线图像如下所示，我们很容易发现这个根的精确值 $p=1$。

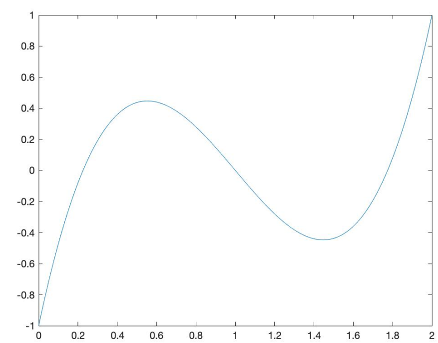


图 1: 三阶的 Legendre 方程图像

下面分别运行二分法，牛顿法，割线法与试位法的程序，如下所示：

```
>> Erfen(f,0.6,1.2,1.0e-5,100)
The solution of the equation is 1.00000305175781.
The time of iteration is 16.
历时 0.176147 秒。
>> Newton(f,1.2,1.0e-5,100)
The solution of the equation is 1.
The time of iteration is 4.
历时 0.080788 秒。
>> Secant(f,0.6,1.2,1.0e-5,100)
The solution of the equation is 1.00000000002203.
The time of iteration is 5.
历时 0.046289 秒。
>> Falseposition(f,0.6,1.2,1.0e-5,100)
The solution of the equation is 1.00000000002203.
The time of iteration is 5.
历时 0.042587 秒。
```

发现除二分法正常运行 (效率较低，迭代次数为 16 次)，牛顿法，割线法均以极少次数与 CPU 时间迭代到 $p=1$。因此下面再看一个例子主要观察下面三者的区别。

2.2 求解含正弦函数的方程在 0.48 周围的根

下面考虑另一个方程（2.3 节习题 14）

$$f(x) = tan(\pi x) - 6 = 0$$

在 0.48 周围 (注意: 0.5 点处趋向于无穷取不到) 的一个零点取值为 $arctan(6)/\pi \approx 0.447431543288747$。下图是该方程在 $[0, 0.5)$ 之间的图像:

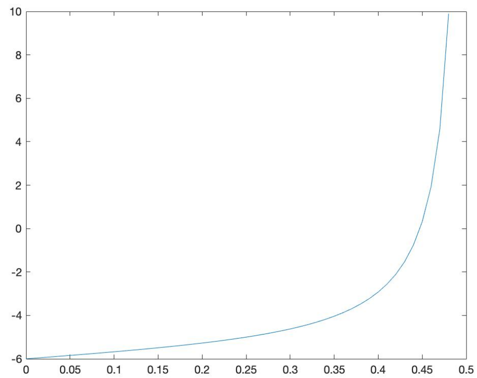


图 2: 含正弦函数的方程图像

再下面是分别运行二分法，牛顿法，割线法与试位法的程序，如下所示:

```
>> Erfen(f,0.33,0.48,1.0e-5,100)
The solution of the equation is 0.447434692382813.
The time of iteration is 14.
历时 0.143779 秒。
>> Newton(f,0.48,1.0e-5,100)
The solution of the equation is 0.447431543288749.
The time of iteration is 6.
历时 0.132283 秒。
>> Secant(f,0.33,0.48,1.0e-5,100)
The solution of the equation is 0.447431545613683.
The time of iteration is 19.
历时 0.142881 秒。
>> Falseposition(f,0.33,0.48,1.0e-5,100)
The solution of the equation is 0.447419477613368.
The time of iteration is 20.
历时 0.149907 秒。
```

结果发现，所有程序都照常运行迭代到所求根的附近。重点观察牛顿法与割线法：牛顿法迭代次数（6 次）相对于割线法迭代次数（19 次）较少，意即割线法比牛顿法收敛速度慢一点；但牛顿法 CPU 运行时间（0.132283 秒）相对于割线法 CPU 运行时间（0.142881 秒）所差无几，这似乎令人费解。细想可知牛顿法每次都要计算导数值，而割线法每次仅需计算函数值, 故每次迭代的计算量较小。因此这两种方法可以说各有所长：**牛顿法迭代次数较少而割线法迭代计算量较小。因此分析具体函数时应考察函数导数的复杂度**。最后，割线法与试位法所有数据近乎一致，但**试位法需要初始值不同号的前提**，否则算法运行结果几乎是错误的。

然后对比误差精度，所有方程都达到要求的误差，关键对比牛顿法与割线法发现，牛顿法的精度（10^{-14} 级）远超割线法的精度（10^{-8} 级）。因此，**牛顿法在同样误差精度的要求下，迭代次数不仅少同时精度同时远大于割线法的精度**。

2.3 求解含正弦函数方程的迭代过程分析

最后，我们利用 Matlab 生成牛顿法，割线法迭代过程（表格），如下所示:

```
>> Newton=A(1:19);
>> Secant=B(1:19);
>> Falseposition=C(1:19);
>> Name={'1';'2';'3';'4';'5';'6';'7';'8';'9';'10';'11';'12';'13';'14';'15';'16';'17';'18';'19'};
>> table(Newton,Secant,Falseposition,'RowNames',Name)

ans =

  19×3 table
```

	Newton	Secant	Falseposition
1	0.467582501925891	0.375506924956847	0.375506924956847
2	0.455129191517774	0.403240063383015	0.403240063383015
3	0.448551233938483	0.505512621820948	0.420228557549826
4	0.447455184250706	0.398519129859361	0.430667054666278
5	0.447431553823758	0.393289091355395	0.437092814309592
6	0.447431543288749	0.49602306948007	0.441052873954212
7	0	0.397455821329561	0.443495067834315
8	0	0.401299293620553	0.445001826963951
9	0	0.48983477958541	0.445931695765643
10	0	0.410343931517097	0.446505640280715
11	0	0.41760604420838	0.446859932460667
12	0	0.467963351544295	0.447078648191422
13	0	0.435884078771263	0.44721367344854
14	0	0.442951990831066	0.447297033930201
15	0	0.44840369002789	0.44734849889977
16	0	0.447349506650761	0.44738027255042
17	0	0.447430039771623	0.447399889203615
18	0	0.447431545613683	0.44741200032049
19	0	0	0.447419477613368

但对比效果不明显，因此画出折线图对比。如下所示为图像。
分析显示：牛顿法可以平稳逼近零点，割线法波动较大，试错法较平稳但收敛速度与割线法持平。

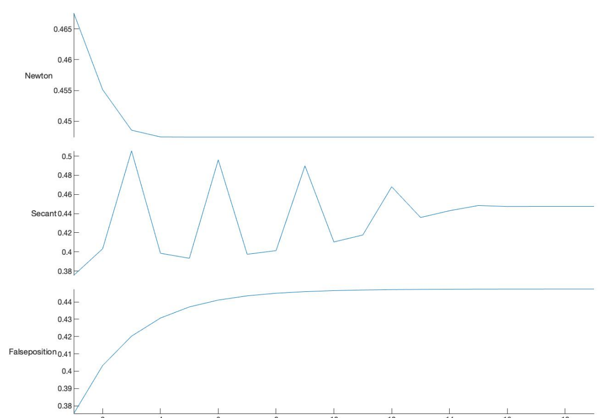


图 3: 求解第二个方程中牛顿法，割线法，试错法的折线图

3. 不动点迭代与 Steffensen 迭代对比

3.1 不动点迭代与 Steffensen 迭代求解三阶 Legendre 方程

依上面所示，牛顿法，割线法与试位法对 $P_3(x)=0$ 在 1.2 附近的根均有较好的效果，现在转用不动点法求解。通过令 $g(x):=P_3(x)+x$ 将方程求根转化为不动点迭代 $g(x)=x$。则下面分别运行不动点迭代与 Steffensen 迭代的程序，并将初始点（宽松地）设置为 1.4。如下所示：

```
>> Fixedpoint(f,1.4,1.0e-5,100)
The solution of the equation is 0.999997584470432.
The time of iteration is 15.
历时 0.108872 秒。
>> Steffensen(f,1.4,1.0e-5,100)
The solution of the equation is 1.
The time of iteration is 3.
历时 0.052457 秒。
```

发现上述计算结果都满足在误差范围内，其中不动点迭代的误差计算如下

$$\left|\frac{p_{15}-p}{p}\right| = 2.415529568\times 10^{-6}.$$

发现不动点迭代的关键位数（Significant Digit）为 6 位而 Steffensen 迭代计算结果完全精准！同时不动点迭代需要的迭代次数为 15 次，而 Steffensen 迭代的迭代次数仅为 3 次。在计算机 CPU 时间上，不动点迭代为 0.060890 秒，而 Steffensen 迭代的迭代次数仅为 0.223619 秒。经过其他方程的验证，二者差别同样明显。**这说明在一定条件下，由于 Steffensen 迭代为二阶收敛，加速后的 Steffensen 迭代相比于不动点迭代更优。**

3.2 求解三阶 Legendre 方程的迭代过程分析

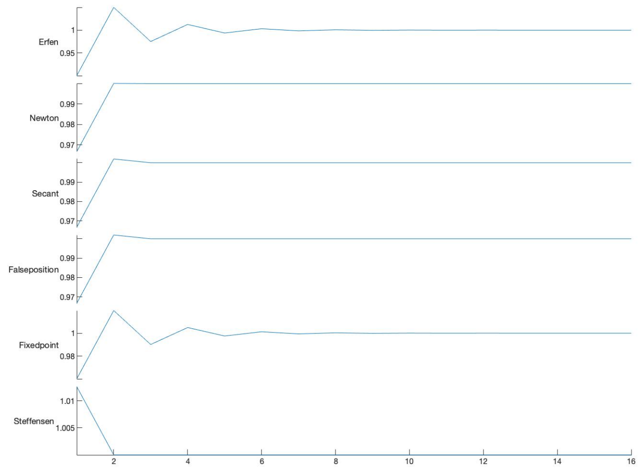
最后我们分析二分法，牛顿法，割线法，试位法，不动点迭代与 Steffensen 迭代在求解三阶 Legendre 方程的迭代过程。同 2.3 节，实现如下：

```
>> table(Erfen,Newton,Secant,Falseposition,Fixedpoint,Steffensen,'RowNames',Name)

ans =

  19×6 table
```

	Erfen	Newton	Secant	Falseposition	Fixedpoint	Steffensen
1	0.9	0.966666666666667	0.966666666666667	0.966666666666667	0.96	1.01267605633803
2	1.05	1.00012414649286	1.00196463654224	1.00196463654224	1.01984	1.00000084884462
3	0.975	0.999999999993622	0.99999657021817	0.99999657021817	0.99009952382976	1
4	1.0125	1	1.00000000002203	1.00000000002203	1.00494781198758	0
5	0.99375	0	0	0	0.997526396822736	0
6	1.003125	0	0	0	1.00123676375046	0
7	0.9984375	0	0	0	0.999381622854105	0
8	1.00078125	0	0	0	1.00030918798179	0
9	0.999609375	0	0	0	0.999845406082997	0
10	1.0001953125	0	0	0	1.00007729694926	0
11	0.99990234375	0	0	0	0.999961351526522	0
12	1.000048828125	0	0	0	1.00001932423659	0
13	0.9999755859375	0	0	0	0.999990337881721	0
14	1.00001220703125	0	0	0	1.00000483105914	0
15	0.999993896484375	0	0	0	0.999997584470432	0
16	1.00000305175781	0	0	0	0	0
17	0	0	0	0	0	0
18	0	0	0	0	0	0
19	0	0	0	0	0	0

同样，绘制折线图对比。如下所示为图像。

分析显示：二分法表现平稳，在大幅度波动 8 次后便趋向于平稳；牛顿法在 3 次迭代后直接平稳达到精确值；割线法与试位法迭代次数稍多于牛顿法且精确度稍逊于牛顿法；不动点迭代表现与二分法一致，表现平稳；加速后的 Steffensen 迭代表现与牛顿法一样好在 3 次迭代后平稳达到精确值。



4. 多项式中的 Horner 与 Muller 算法

首先是 Horner 算法，即**采用嵌套方式**去计算多项式在固定点的值与导数值。例如 $P(x) = x^2 + x$ 在 x=1 处的函数值与导数值分别为 2,3。如下所示：

```
>> Horner(2,[0,1,1],1)

ans =

     2     3
```

其次是 Muller 算法的一个简单应用：求解 $x^2 + 1 = 0$ 的复根。如下所示：

```
>>  Muller(x^2+1,-1,0,1,1.0e-5,100);
The solution of the equation is 0-1i.
The time of iteration is 4.
历时 0.069975 秒。
>> Muller(f,0.6,1,1.2,1.0e-5,100);
Method failed after N0 iterations, N0=100
历时 2.270759 秒。
>> Muller(f,0.6,1.1,1.2,1.0e-5,100);
The solution of the equation is 1.
The time of iteration is 6.
历时 0.119300 秒。
```

之后还是回归到求解三阶 Legendre 方程在 1.2 附近的根，观察与之前的算法在迭代次数及 CPU 运行时间上的对比。利用 Muller 算法求解三阶 Legendre 方程，代码在上方实现。

发现，如初始点不在零点取值，则算法在这个多项式方程上表现很优：首先迭代次数仅为 6 次就达到精度要求，同时**在同一个方程求根问题中历时在所有算法中几乎最短**，且最终结果的关键位数为 12，计算如下：

$$\left|\frac{p_6 - p}{p}\right| = 1.511 \times 10^{-12} < 5 \times 10^{-12}$$

5. 代码附录

5.1 二分法

```
% Bisection method for a solution of f(x)=0
function output=Erfen(f,a,b,TOL,N0)
% Calculate runtime of the program
tic;
% If TOL is missing, error is assumed to be the standard one 1E-3.
if(nargin==4)
    TOL=1.0e-3;
end
% Initialize variable for iteration ordinal number
i=1;
FA=subs(f,a);
k=1;
J=zeros(1,100);
while (i<=N0)
    p=a+(b-a)/2;
    FP=subs(f,p);
    J(k)=p; % show the process of iteration
    k=k+1;
    if ((FP==0)||((b-a)/2<TOL))
        disp(['The solution of the equation is ',num2str(p,15),'.']);
        disp(['The time of iteration is ',num2str(i),'.']);
        output=J;
        toc
        return;
    end
    i=i+1;
    if(FA*FP>0)
        a=p;
        FA=FP;
    else
        b=p;
    end
end
% the procedure was unsuccessful
disp(['Method failed after N0 iterations, N0=',num2str(N0)])
toc
```

5.2 不动点迭代法

```
% Fixed point itteration for a solution to p=g(p)
function output=Fixedpoint(g,p0,TOL,N0)
% Calculate runtime of the program
tic;
% If TOL is missing, error is assumed to be the standard error 1E-3.
if(nargin==3)
    TOL=1.0e-3;
end
% Initialize variable for iteration ordinal number
i=1;
k=1;
J=zeros(1,100);
while (i<=N0)
    p=subs(g,p0);
    p=double(p);
    J(k)=p; % show the process of iteration
```

```
        k=k+1;
        if (abs(p-p0)<TOL)
            disp(['The solution of the equation is ',num2str(p,15),'.']);
            disp(['The time of iteration is ',num2str(i),'.']);
            output=J;
            toc
            return;
        end
        i=i+1;
        p0=p;
    end
    % the procedure was unsuccessful
    disp(['Method failed after N0 iterations, N0=',num2str(N0)])
    toc
```

5.3 Muller 法

```
        % Muller's method for a solution of f(x)=0
    function output=Muller(f,x0,x1,x2,TOL,N0)
    % Calculate runtime of the program
    tic;
    % If TOL is missing, error is assumed to be the standard one 1E-3.
    if(nargin==4)
        TOL=1.0e-3;
    end
    % Initialize variable for iteration ordinal number
    i=3;
    h1=x1-x0;
    h2=x2-x1;
    fx0=subs(f,x0);
    fx0=double(fx0);
    fx1=subs(f,x1);
    fx1=double(fx1);
    fx2=subs(f,x2);
    fx2=double(fx2);
    d1=(fx1-fx0)/h1;
    d2=(fx2-fx1)/h2;
    d=(d2-d1)/(h2+h1);
    k=1;
    J=zeros(1,100);
    while (i<=N0)
        b=d2+h2*d;
        D=(b^2-4*fx2*d)^(1/2);
        if (abs(b-D)<abs(b+D))
            E=b+D;
        else
            E=b-D;
        end
        h=-2*fx2/E;
        p=x2+h;
        J(k)=p; % show the process of iteration
        k=k+1;
        if (abs(h)<TOL)
            disp(['The solution of the equation is ',num2str(p),'.']);
            disp(['The time of iteration is ',num2str(i),'.']);
            output=J;
            toc
            return;
        end
        i=i+1;
        x0=x1;
```

```
        x1=x2;
        x2=p;
        h1=x1-x0;
        h2=x2-x1;
        fx0=subs(f,x0);
        fx0=double(fx0);
        fx1=subs(f,x1);
        fx1=double(fx1);
        fx2=subs(f,x2);
        fx2=double(fx2);
        d1=(fx1-fx0)/h1;
        d2=(fx2-fx1)/h2;
        d=(d2-d1)/(h2+h1);
    end
    % the procedure was unsuccessful
    disp(['Method failed after N0 iterations, N0=',num2str(N0)])
    toc
```

5.4 牛顿法

```
        % Newton's method for a solution to f(x)=0
    function output=Newton(f,p0,TOL,N0)
    % Calculate runtime of the program
    tic;
    % If TOL is missing, error is assumed to be the standard error 1E-3.
    if(nargin==3)
        TOL=1.0e-3;
    end
    % Initialize variable for iteration ordinal number
    i=1;
    k=1;
    J=zeros(1,100);
    while (i<=N0)
        fx=subs(f,p0);
        fx=double(fx);
        df=diff(f);
        df=subs(df,p0);
        df=double(df);
        p=p0-fx/df;
        J(k)=p; % show the process of iteration
        k=k+1;
        if (abs(p-p0)<TOL)
            disp(['The solution of the equation is ',num2str(p,15),'.']);
            disp(['The time of iteration is ',num2str(i),'.']);
            output=J;
            toc
            return;
        end
        i=i+1;
        p0=p;
    end
    % the procedure was unsuccessful
    disp(['Method failed after N0 iterations, N0=',num2str(N0)])
    toc
```

5.5 试错法

```
    % Method of False position for a solution to f(x)=0
function output=Falseposition(f,p0,p1,TOL,N0)
% Calculate runtime of the program
tic;
% If TOL is missing, error is assumed to be the standard error 1E-3.
if(nargin==3)
    TOL=1.0e-3;
end

% Initialize variable for iteration ordinal number
i=2;
q0=subs(f,p0);
q0=double(q0);
q1=subs(f,p1);
q1=double(q1);
k=1;
J=zeros(1,100);
while (i<=N0)
    p=p1-q1*(p1-p0)/(q1-q0);
    J(k)=p; % show the process of iteration
    k=k+1;
    if (abs(p-p1)<TOL)
        disp(['The solution of the equation is ',num2str(p,15),'.']);
        disp(['The time of iteration is ',num2str(i),'.']);
        output=J;
        toc
        return;
    end
    i=i+1; %update p0,q0,p1,q1.
    q=subs(f,p);
    q=double(q);
    if (q*q1<0)
        p0=p1;
        q0=q1;
    end
    p1=p;  %update p1,q1.
    q1=q;
end
% the procedure was unsuccessful
disp(['Method failed after N0 iterations, N0=',num2str(N0)])
toc
```

5.6 Horner 法

```
% Horner's method to evaluate a polynomial and its derivative at x0
function output=Horner(n,a,x0)
% compute cofficient b(n) for P and b(n-1) for Q
y=a(n+1);
z=a(n+1);
for j=n-1:1
    y=x0*y+a(j+1); % compute b(j) for P
    z=x0*z+y;  % compute b(j-1) for Q
end
y=x0*y+a(1);% compute b(0) for P
output=[y,z];
```

5.7 割线法

```
% Secant method for a solution to f(x)=0
function output=Secant(f,p0,p1,TOL,N0)
% Calculate runtime of the program
tic;
% If TOL is missing, error is assumed to be the standard error 1E-3.
if(nargin==3)
    TOL=1.0e-3;
end
% Initialize variable for iteration ordinal number
i=2;
q0=subs(f,p0);
q0=double(q0);
q1=subs(f,p1);
q1=double(q1);
k=1;
J=zeros(1,100);
while (i<=N0)
    p=p1-q1*(p1-p0)/(q1-q0);
    J(k)=p; % show the process of iteration
    k=k+1;
    if (abs(p-p1)<TOL)
        disp(['The solution of the equation is ',num2str(p,15),'.']);
        disp(['The time of iteration is ',num2str(i),'.']);
        output=J;
        toc
        return;
    end
    i=i+1;
    p0=p1; % update p0,q0,p1,q1
    q0=q1;
    p1=p;
    q1=subs(f,p);
    q1=double(q1);
end
% the procedure was unsuccessful
disp(['Method failed after N0 iterations, N0=',num2str(N0)])
toc
```

5.8 Steffensen 法

```
% Steffensen's method for a solution to p=g(p)
function output=Steffensen(g,p0,TOL,N0)
% Calculate runtime of the program
tic;
% If TOL is missing, error is assumed to be the standard error 1E-3.
if(nargin==3)
    TOL=1.0e-3;
end
% Initialize variable for iteration ordinal number
i=1;
k=1;
J=zeros(1,100);
while (i<=N0)
    p1=subs(g,p0);
    p1=double(p1);
    p2=subs(g,p1);
    p2=double(p2);
    p=p0-(p1-p0)*(p1-p0)/(p2-2*p1+p0);
    J(k)=p; % show the process of iteration
    k=k+1;
    if (abs(p-p0)<TOL)
```

```
            disp(['The solution of the equation is ',num2str(p,15),'.']);
            disp(['The time of iteration is ',num2str(i),'.']);
            output=J;
            toc
            return;
        end
        i=i+1;
        p0=p;
    end
    % the procedure was unsuccessful
    disp(['Method failed after N0 iterations, N0=',num2str(N0)])
    toc
```