2008 년 2 학기 HW#8

수치해석기초

HW#8: **Roots of Legendre Polynomial**

원자핵공학과

2003-12491

이 원 재

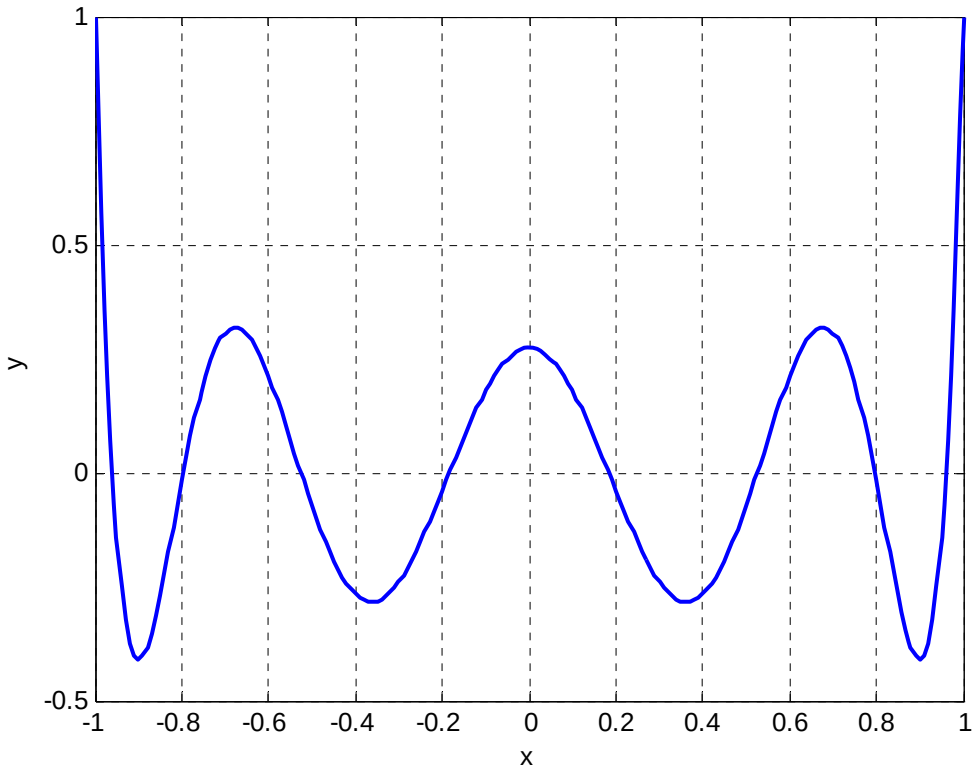
1. Define the polynomial for the x range of [-1,1] and make estimates of the roots by eyes. Also divide the positive domain [0,1] into four intervals such that each interval include one root.

주어진 다항함수를 다음과 같은 MATLAB 함수로 정의 한다.

In file p8.m

```
function [p8x] = p8(x)
n=length(x);

for (i=1:n)
        p8x(i)=35/128-315/32*x(i)^2+ 3465/64*x(i)^4-
3003/32*x(i)^6+ 6435/128*x(i)^8;
end
```

```
>> x=-1:0.01:1;
>> plot(x,p8(x),'LineWidth',2);
>> xlabel('x'); ylabel('y');
>> grid on
```



그래프를 보면 구간 [0,1] 사이에 4개의 근이 대략 0.2, 0.5, 0.8, 0.95 근처에 위치하고 있음을 알 수 있다.

따라서 다음과 같이 구간 [0,1]이 각 근을 포함하도록 4개의 구간으로 나누어 줄 수 있다.

```
>> xbracket=[0, 0.3 ; 0.3, 0.6 ; 0.6, 0.9 ; 0.9, 1;]

xbracket =

         0    0.3000
    0.3000    0.6000
    0.6000    0.9000
    0.9000    1.0000

>>
```

각 열은 1개의 근을 포함하는 subinterval로 볼 수 있다.

2. Write a MATLAB program to implement

1) Bisection Method

2) Modified Linear Interpolation Method (Secant Method)

3) Newton-Raphson Method

각각의 프로그램을 수록해보면

1) Bisection Method

In file bsm.m

```
function [xm,cnt,id] = bsm(fhs,a,b)
%BiSection Method
%첫번째 argument는 string으로된 임의의 함수명을 받는다.
%두번째, 세번째 argument는 x축 구간의 왼쪽과 오른쪽 경계이다.
%첫번째 리턴 인자는 구간내에서 구한 해이고, 두번째 리턴 인자는 반복횟수
%세번째는 수렴조건의 확인용 변수이다.
fh=str2func(fhs); %function handle
epsm=1.e-10; %수렴 조건
epsf=1.e-10; %함수값 수렴조건
if (a>b) %a가 왼쪽 경계가 되도록 함.
        temp=a;a=b;b=temp;
end
xl=a;fl=fh(xl);
xr=b;fr=fh(xr);

id=0;cnt=0;
while (id==0)
        xm=0.5*(xl+ xr);
        fm=fh(xm);
        if (fl*fm<0) %근이 왼쪽에 있을 때
                xr=xm;
                fr=fm;
        else %근이 오른쪽에 있을 때
                xl=xm;
                fl=fm;
        end
        delx=xr-xl;delf=fr-fl;
        if (abs(delx/xm)<epsm)
```

```
            id=1;
        end
        if (abs(delf)<epsf)
            id=id+ 2;
        end
        cnt=cnt+ 1;
end
```

2) Modified Linear Interpolation Method (Secant Method)

In file mlim.m

```
function [xm,cnt,id] = mlim(fhs,a,b)
%Modified Linear Interpolation Method
%첫번째 argument는 string으로된 임의의 함수명을 받는다.
%두번째, 세번째 argument는 x축 구간의 왼쪽과 오른쪽 경계이다.
%첫번째 리턴 인자는 구간내에서 구한 해이고,
%두번째 리턴 인자는 반복횟수
%세번째는 수렴조건의 확인용 변수이다.
fh=str2func(fhs);%function handle
epsm=1.e-10;
epsf=1.e-10;

if (a>b)%a가 왼쪽 경계, b가 오른쪽 경계가 되도록 함.
        temp=a;
        a=b;
        b=temp;
end

xl=a;fl=fh(xl);
xr=b;fr=fh(xr);

id=0;
fp=0;
cnt=0;
while (id==0)
        xm=xl-(xr-xl)/(fr-fl)*fl;
```

```
        fm=fh(xm);

        if (fl*fp>=0)%해를 지나치지 않았으면 반대편을 1/2로 해줌
                halving=1;
                fp=fm;
        else
                halving=0;
                fp=0;
        end

        if (fl*fm<0) %해가 xl과 xm 사이에 있으면
                xr=xm; %xm을 새로운 xr로 해줌
                fr=fm;
                if(halving==1) %해를 지나치지 않았으면
                        fl=0.5*fl; %반대쪽을 반으로 나눠줌
                end
        else  %해가 xm과 xr사이에 있게 되는 경우
                xl=xm;%새로 구한 점 xm이 xl이 되는 것
                fl=fm;
                if(halving==1) %그리고 해를 지나치지 않았다면 반으로 나눠줌
                        fr=0.5*fr;
                end
        end

        delx=xr-xl;delf=fr-fl;
        if (abs(delx/xm)<epsm)
                id=1;
        end
        if (abs(delf)<epsf)
                id=id+ 2;
        end
        cnt=cnt+ 1;
end
```

3) Newton-Raphson Method

In file nrm.m

```
function [xk,cnt,id] = nrm(fhs,a,b)
%Newton Rhapson Method
%첫번째 argument는 string으로된 임의의 함수명을 받는다.
%두번째, 세번째 argument는 x축 구간의 왼쪽과 오른쪽 경계이다.
%첫번째 리턴 인자는 구간내에서 구한 해이고,
%두번째 리턴 인자는 반복횟수
%세번째는 수렴조건의 확인용 변수이다.
fh=str2func(fhs);%function handle
epsm=1.e-10; %x의 convergence 조건
epsf=1.e-10; %y의 convergence 조건

if (a>b) %a가 왼쪽 경계, b가 오른쪽 경계가 되도록 함.
        temp=a;
        a=b;
        b=temp;
end

xk_1=(a+ b)/2; %구간의 중간점을 시작점으로 함.
id=0;cnt=0; %계산횟수를 세기위한 변수
while (id==0)
        xk=xk_1-fh(xk_1)/df(fhs,xk_1);%fixed point iteration의 g(x)함수를 알맞게 줌
        delx=(xk-xk_1)/xk; %x의 convergence 변수
        if (abs(delx)<epsm)
                id=1; %이렇게 해두면 자동으로 while loop에서 break가 됨
        end
        xk_1=xk;
        cnt=cnt+ 1;
end

function [result] = df(fhs,x)
fh=str2func(fhs);%function handle
epsilon=1.e-6;
result=(fh(x+ epsilon)-fh(x-epsilon))/2/epsilon;
```

3. Apply these three methods to find the four roots and compare the number of trials of these methods for each root. Use the absolute convergence criterion of 1.0×10^{-10} for both x and y. For the Newton-Raphson method choose the middle point of the interval as the starting point.

위에서 설정한 구간에 대해 근을 각각 구해줄 수 있도록 다음과 같은 함수를 작성한다.

```
function [grphmat] = froot(fhs,xbrk)
[n,c]=size(xbrk);%subinterval의 개수를 변수 n에 넣어줌
grphmat=zeros(n,6);
for (i=1:n)
        [grphmat(i,1),grphmat(i,2),grphmat(i,3)]=bsm(fhs,xbrk(i,1),xbrk(i,2));
        [grphmat(i,4),grphmat(i,5),grphmat(i,6)]=mlim(fhs,xbrk(i,1),xbrk(i,2));
        [grphmat(i,7),grphmat(i,8),grphmat(i,9)]=nrm(fhs,xbrk(i,1),xbrk(i,2));
end
%출력 스트림
fprintf('₩t Bisection method          |      Secant method          |      Newton-
Rhapson method₩n');
fprintf('₩t    root          iter.   id |    root          iter.   id |    root          iter.   id
₩n');
for(i=1:n)
        fprintf('₩t %8f %8d %4d   |   %8f %8d %4d | %8f %8d %4d ₩n', grphmat(i,:));
end
```

```
>> [grphmat] = froot('p8',xbracket);
      Bisection                |          Secant                |      Newton-Rhapson
     root          iter.     id  |       root          iter.    id |    root          iter.     id
0.183435          33        2   |   0.183435          8        3 | 0.183435          4         1
0.525532          33        1   |   0.525532          11       3 | 0.525532          4         1
0.796666          32        1   |   0.796666          11       3 | 0.796666          5         1
0.960290          30        1   |   0.960290          12       3 | 0.960290          5         1
>>
```

위에서 xbracket 변수에는 1번 문제에서 규정해 둔 4행 2열의 구간정보가 들어있다.

결과를 살펴보면 세 방법 모두 4개의 근을 성공적으로 구했으며 반복횟수는 bisection method의 경우 약 32회, modiefied linear interpolation method는 약 11회, Newton-Rhapson method 의 경우에는 4 또는 5회의 반복으로 근을 구해냈음을 알 수 있다. 세가지 방법 모두 각 근에 대해 같은 구간을 사용하였으므로 동등한 조건하에서 비교되었다고 볼 수 있고, 이를 통해 Newton-Rhapson method가 가장 효율적이며 Bisection method가 가장 비 효율적임을 확인 할 수 있다.

물론 초기에 각 근을 찾기 위한 구간을 좀더 좁혀서 주면 더 적은 반복횟수로도 근을 찾을 수 있을 것으로 생각되는데 이를 확인 해 보면

```
>> xbracket1=[0.1,0.2;0.5,0.6;0.7,0.8;0.9,1]
xbracket1 =

    0.1000    0.2000
    0.5000    0.6000
    0.7000    0.8000
    0.9000    1.0000
>> [grphmat] = froot('p8',xbracket1);
```

Bisection			Secant			Newton-Rhapson		
root	iter.	id	root	iter.	id	root	iter.	id
0.183435	32	2	0.183435	8	3	0.183435	4	1
0.525532	31	1	0.525532	7	3	0.525532	4	1
0.796666	31	1	0.796666	8	3	0.796666	5	1
0.960290	30	1	0.960290	12	3	0.960290	5	1

```
>>
```

구간을 상당히 좁혀서 넘겨 주었음에도 불구하고 반복횟수가 두드러지게 줄어들지는 않았음을 확인 할수 있다. 이러한 이유를 생각해 보면 먼저 각 알고리즘은 수렴조건이 1.e-10와 같이 매우 엄격한 조건으로 반복횟수가 결정되는데, 근이 존재할 구간을 좁혀서 넘겨준다고 하여도 원래 구간에서 반복횟수 1회 내지 2회를 거치면 얻을 수 있는 구간 간격이므로 큰 영향을 받지 않음을 알 수 있다. 특히 Newton-Rhapson method는 본래 방법이 효율적이어서 수렴을 빨리 하고, 또한 구간을 양쪽에서 줄이면 구간의 중간점을 초기값으로 취하는 알고리즘에 따라 별 도움을 받지 못하는 것을 볼 수 있다.

4. In this case, it is easy to guess the range where the roots are located. But in general the location of the roots is not known. In the general case, how would you determine the initial points? Discuss your logic for finding the first root. Then discuss how you would use the roots you already found to avoid finding those roots in the subsequent root finding. Apply your new method to determine all four roots with the Newton-Raphson method without the prior determination of the range of the roots already done in Prob 1.

일반적인 경우에는 근의 위치가 어디에 있을지는 알수 없다. 하지만 대부분의 경우 공학적인 문제에서 원하는 근이 음수인지 양수인지와 어느 정도 크기의 근을 찾고 있는지는 예상할수 있다. 따라서 근을 구할 때 원하는 근을 찾을 때, (-∞, ∞)가 아닌 [0, ∞) 또는 [a,b]와 같이 근이 존재할 비교적 큰 범위를 줄 수 있다. 이렇게라도 범위를 줄 수 없다면 할 수없이 0을 시작점으로 해서 singular인지 아닌지 잘 확인해 보면서 찾아봐야 할 것이다.

만약 근의 범위가 이번 문제에서처럼 [0,1]로 주어져 있다면 근이 중근이 아니라는 조건하에서 첫번째 근, 그리고 나머지 근의 범위를 다음과 같은 방법으로 찾아낼 수 있다.

우선 대구간 [a,b]의 왼쪽 경계에서 시작하여 그 위치에서 함수값과 dx 만큼 오른쪽으로 이동한 위치에서의 함수값을 곱해서 0보다 작거나 같은지를 확인한다. 만약 0보다 크다면 그 구간내에는 근이 존재하지 않는 것이므로 오른쪽 x값을 다음번 반복에서의 왼쪽 x 값으로 하고 그 다음번 오른쪽 x 값은 dx 만큼 더한 값으로 한다. 이와 같은 방법을 계속하면서 첫번째 근이 구간내에 가둬질 때까지 반복하면 된다. dx값을 되도록 작게 하면 근을 놓칠 가능성이 줄어들지만 너무 작게 하면 첫번째 근을 구간내에 가두는데 그만큼 시간이 많이 걸리게 된다. 이번 문제에서와 같은 경우 dx값을 적당히 주어도 첫번째 근을 놓치지 않고 찾아낼 수 있게 된다. 따라서 이런 문제와 같은 경우는 대구간[0,1] 내에 존재할 모든 근을 이러한 방법으로 모두 가둬낼 수가 있다. 그렇지 않은 경우 dx값을 작게하여 첫번째 근이 존재하는 구간을 을 찾아낸 후 Newton-Rhapson method를 이용하여 첫번째 근을 구한다. 이후에 두번째 근을 찾기 위한 Newton-Rhapson method의 시작점을 결정하는 과정이 필요하다. 이 시작점을 잘 결정하지 못 할경우 즉, 기존 근에 너무 근접한 점을 시점으로 잡으면 첫번째 근을 찾게 된다. 따라서 Newtorn rhapson 법을 쓰면서 접선을 그었을 때, 그 x 절편이 먼저 구해놓은 근보다 작으면 잘못된 점임을 바로 알 수 있다. 따라서 이 문제를 푸는 방법은 시작점을 먼저 구한 근에서 조금 움직이고 Newton-Rhapson method 도중 접선의 x절편이 먼저 구한 근에 가까워 지는 경우, 구간을 재조정해 주는 방법을 생각해야 하

는데, 이 방법은 계산에 너무 많은 로드가 생기게 된다. 따라서 구간에 존재하는 해를 모두 소구간으로 가둬서 문제를 푸는 방법이 우선은 효율적인 방법이라고 할 수 있다.

한편 이렇게 Newton-Rhapson method를 이용하여 해를 구하는 도중에 운이 좋지 않은 경우 극대점이나 극소점을 만나서 x절편이 날아가 버리는 경우(구간을 벗어나는 경우)가 생길수 있는데 이런 경우가 생기는 경우에는 Bisection method를 사용하여 구간을 재조정하면 문제를 해결할 수 있게 된다.

따라서 구간을 자동으로 잡아주는 루틴과, 이렇게 잡은 소구간으로 Newton-Rhapson method를 사용하는 루틴을 작성해 보면

In file brk.m

```
function [xb] = brk(fhs,x1,x2,n,nroot)
%BRacKetting function p8(x) within large interval [x1,x2]
%discretization number n, and max interval number nroot
%fh takes string of function name

fh=str2func(fhs);%function handle
xb=zeros(1,2);
dx=(x2-x1)/n;
rownb=0;
x=x1;
fp=fh(x);%function value past
for (i=1:n)
        x=x+ dx;
        fc=fh(x);%function value current
        if (fc*fp<=0.0)
                rownb=rownb+ 1;
                xb(rownb,1)=x-dx;
                xb(rownb,2)=x;
        end
        if(rownb == nroot)
                break;
        end
        fp=fc;
end
```

In file nrmu.m

```
function [xk,cnt] = nrmu(fhs,a,b)
%Newton Rhapson Method Upgrade
fh=str2func(fhs);%function handle
epsm=1.e-10;
epsf=1.e-10;
if (a>b)
        temp=a;
        a=b;
        b=temp;
end
xk_1=(a+ b)/2;
id=0;
cnt=0;%Newton Rhapson 반복 횟수
cntb=0;%bisection으로 구간 재조정한 횟수
while (id==0)
        xk=xk_1-fh(xk_1)/df(fhs,xk_1);
        if (xk<a)%구간을 벗어난 경우 bisection method를 사용해서 구간 재조정
                xm=(a+ b)/2;
                fm=fh(xm);
                if(fm*fh(a)<0)
                        b=xm;
                        xk_1=(a+ b)/2;
                        cntb=cntb+ 1;
                        continue;
                else
                        a=xm;
                        xk_1=(a+ b)/2;
                        cntb=cntb+ 1;
                        continue;
                end
        end

        delx=(xk-xk_1)/xk;
        if (abs(delx)<epsm)
                id=1;
```

```
        end
        xk_1=xk;
        cnt=cnt+ 1;
end
%cntb;

function [result] = df(fhs,x)
fh=str2func(fhs);%function handle
epsilon=1.e-6;
result=(fh(x+ epsilon)-fh(x-epsilon))/2/epsilon;
```

In file nrmauto.m

```
function [xkroot,cnt] = nrmauto(fhs,a,b)
%Newton Rhapson Method AUTOmatic
m=100;
nroot=m;%구할 최대 근 개수
[xb] = brk(fhs,a,b,m,nroot);
[n,c]=size(xb);
for (i=1:n)
        [xkroot(i),cnt(i)] = nrmu(fhs,xb(i,1),xb(i,2));
end
```

이 함수는 근을 구할 함수명과, 구간을 넘겨받은 후, 근이 존재하는 소구간들을 찾은 후에 이 소구간들을 차례대로 modified 된 Newton-Rhapson method 에 넘겨주어 구간내의 근을 찾아낸다.

```
>> [xkroot,cnt] = nrmauto('p8',0,1)
xkroot =

    0.1834    0.5255    0.7967    0.9603

cnt =

     3     3     4     4
```

더 넓은 구간에서 찾아보면

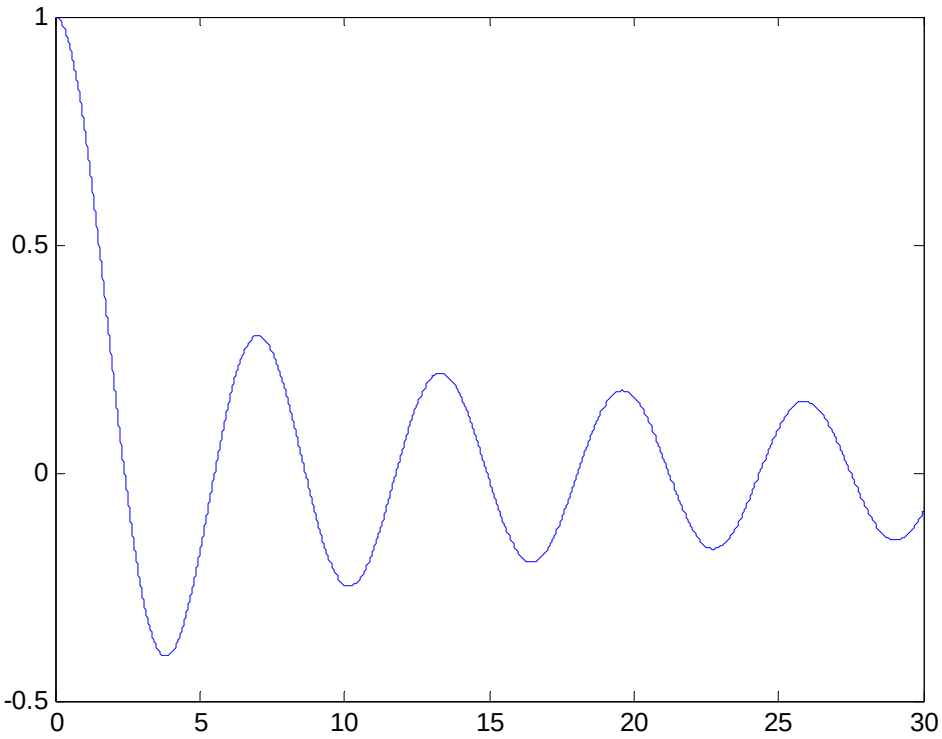
```
>> [xkroot,cnt] = nrmauto('p8',-1,1)
xkroot =

-0.9603    -0.7967    -0.5255    -0.1834     0.1834     0.5255     0.7967     0.9603

cnt =
     5     4     4     3     3     4     4     5

>>
```

매우 빠르게 근을 구할 수 있음을 알 수 있다.

한편, 다른 함수의 근도 이러한 루틴으로 구할 수 있는지 살펴보기 위해 다음과 같은 함수를 정의한 후 근을 구해보면

```
function [result]=j0(x)
result=besselj(0,x);
```

```
>> x=0:0.01:30;
>> plot(x, j0(x))
```



>> [xkroot,cnt] = nrmauto('j0',0,30)

xkroot =

2.4048 5.5201 8.6537 11.7915 14.9309 18.0711 21.2116
24.3525 27.4935

cnt =

4 4 4 4 3 4 3 4 3

구간내의 모든 근을 구할 수 있음을 알 수 있다.

<부록> matlab function list

p8.m

bsm.m

mlim.m

nrm.m

froot.m

brk.m

nrmu.m

nrmauto.m

j0.m