2008년 2학기 HW#3

수치해석기초

HW#3 : Cubic Spline

원자핵공학과
2003-12491
이 원 재

4. Cubic spline의 계수 bi, ci, di를 구하는 함수의 구현

In file cspline.m

```
function [BN, CN, DN] = cspline(XN, YN, YPL, YPR)
%%function [BN, CN, DN] = cspline(XN, YN, YPL, YPR)
%Suppose that the x and y points are specified by the two vectors, XN and YN
%slopes at the two end points are given as YPL and YPR.
%Use these two vectors and two values as the input parameters
%return the 3N coefficients in terms of three vectors BN, CN, DN as the output.
N=length(XN)-1;     %0부터 시작해서 N으로 끝나길 원함 전체 데이터 포인트의 개수는 N+1개

A=zeros(3*N,3*N);  % N-1개 데이터 포인트에 대해 각각 3개의 셑식을 쓰고,
                   % 나머지 3개는 index가 N인 마지막 데이터 포인트 그리고 양끝점에서의 미분값.
b=zeros(3*N,1);     % 3*(N)의 칼럼 영 벡터 선언.

%%%%%
A(1,1)=1; % 계수행렬의 첫번째 행을 먼저 채움
b(1)=YPL;% 첫번째 행에 해당하는 b 벡터의 값을 먼저 채움

for (i=1:N-1)                %계수행렬의 가운데 부분을 i가 1~N-1일때에 대해 채움.
   rowfori=3*i-1;            %/ROW FOR I///assign row number for i-th iteration.
        colfori=3*i-2;       %/COL FOR I///assign col number for i-th iteration.
        hi=XN(i+1)-XN(i);                    %to minimize the number of function call times.

        A(rowfori,colfori)=hi;          %
        A(rowfori,colfori+1)=hi*hi;     %%% 함수값 연속에 의한 term들.
        A(rowfori,colfori+2)=hi*hi*hi;  %
        b(rowfori)=YN(i+1)-YN(i);       %index를 하나 줄였으므로 실제 접근하려면 하나씩 더해줘야함
                                        %(i번째에서 i-1번째 y값 뺀것)

        A(rowfori+1,colfori)=1;              %%%
        A(rowfori+1,colfori+1)=2*hi;         %%%
        A(rowfori+1,colfori+2)=3*hi*hi;      %%%일차미분값 연속조건에 해당하는 식들
        A(rowfori+1,colfori+3)=-1;           %%%

        %A(rowfori+2,colfori)=0;             % 애초에 영 행렬로 계수벡터 선언하였으므로..
        A(rowfori+2,colfori+1)=1;            %%%
        A(rowfori+2,colfori+2)=3*hi;         %%%이차미분값 연속조건에 해당하는 식들
        %A(rowfori+2,colfori+3)=0;           %%%
        A(rowfori+2,colfori+4)=-1;           %%%
end

rowfori=3*N-1;                      %%N-1까지 채우고 나서 나머지 두행을 마무리
colfori=3*N-2;
hN=XN(N+1)-XN(N);                   %%데이터포인트 간격 맨끝에서부터 그 앞에 것

A(rowfori,colfori)=hN;
A(rowfori,colfori+1)=hN*hN;
A(rowfori,colfori+2)=hN*hN*hN;
b(rowfori)=YN(N+1)-YN(N);

A(rowfori+1,colfori)=1;
```

```
A(rowfori+ 1,colfori+ 1)=2*hi;
A(rowfori+ 1,colfori+ 2)=3*hi*hi;
b(rowfori+ 1)=YPR;

%%%%%%%%%%%%%%%%%%%%%%%%%
x=A₩b;                  %%% 일차 연립방정식의 풀이

for (i=1:N)             %%%%계수 벡터 취하여 리턴
            BN(i)=x(3*i-2);
            CN(i)=x(3*i-1);
            DN(i)=x(3*i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5. 세 개의 데이터 포인트를 이용하여 양 끝에서 함수의 기울기를 결정하는 함수의 구현 (Lagrange polynomial을 사용해 유도하기)

In file findslope.m

```
function [ypl, ypr] = findslope(XN,YN)
%%function [ypl, ypr] = findslope(XN,YN)
%determine the slope at the left end points using the first three data points
%and then to determine the slope at the right end points using the last three data points.
%using the second order Lagrange polynomial.
N=length(XN)-1;                 %0부터 시작해서 N으로 끝나게 하려고...

h1=XN(1+1)-XN(0+1);             %괄호안에서 덧셈 앞이 인덱스, 뒤에 더하기 1은 벡터의 원소접근 위한것.
h2=XN(2+1)-XN(1+1);
gl=h2/h1;                       %Gamma Left

ypl=(-YN(0+1)*(2+gl)+YN(1+1)*(2+gl+1/gl)-YN(2+1)/gl)/(h1+h2);


%%%% for the right end
h1=XN(N-1+1)-XN(N-2+1);
h2=XN(N+1)-XN(N-1+1);
gr=h1/h2;                       %Gamma Right

ypr=(1/gr*YN(N-2+1)-(2+gr+1/gr)*YN(N-1+1)+(2+gr)*YN(N+1))/(h1+h2);
```

6. 작성한 함수를 사용하여

$$f(x) = e^{-2x}\cos(4\pi\ x)$$

x:	0.0	0.1	0.2	0.3	0.4	0.5
y:	1.0000e+0	2.5300e-1	-5.4230e-1	-4.4400e-1	1.3885e-1	3.6788e-1
x:	0.6	0.7	0.8	0.9	1.0	
y:	9.3074e-2	-1.9950e-1	-1.6334e-1	5.1080e-2	1.3534e-1	

fitting 하기

다음과 같은 matlab 함수를 작성한다.

In file cubicsf.m

```
function [Yout] = cubicsf(XN,YN,Xin)
%%%function [Yout] = cubicsf(XN,YN,Xin)
%%cubic spline fit using given data points vetor XN, YN.
%%returning value for the desired point Xin.
N=length(XN)-1;  %주어진 데이터 포인트의 개수 -1 -> 구간의 개수

[ypl,ypr]=findslope(XN,YN);
[bn,cn,dn]=cspline(XN,YN,ypl,ypr);
%determine polynomial index or interval index for Xin.
i=1;
if (Xin==XN(N+1))
   i=N;
else
   for (i=1:N)
          if ( (Xin>=XN(i-1+1)) && (Xin<XN(i+1)) )
             break;
          else
             i=i+1;
          end
   end
end

Xshift=Xin-XN(i-1+1);
Yout=YN(i-1+1)+bn(i)*Xshift+cn(i)*Xshift*Xshift+dn(i)*Xshift*Xshift*Xshift;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

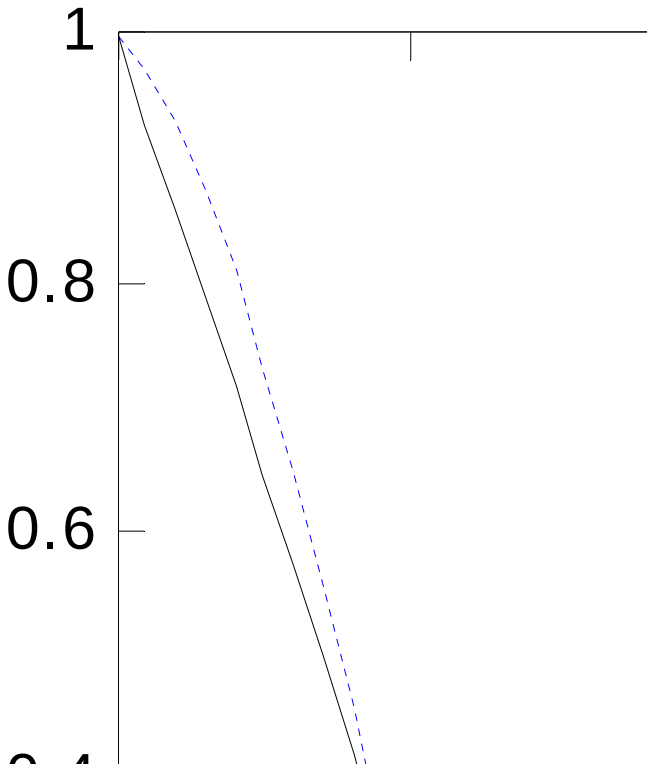
이 함수는 데이터 포인트를 나타내는 두 개의 벡터와 원하는 x 포인트를 입력 파라메터로 받아서 먼저 구간 양끝에서의 기울기를 구하는 루틴을 호출하고, 받은 기울기를 추가적인 입력 파라메터로 하여 cubic spline 계수를 구하는 루틴을 호출한다. 이렇게 얻은 bn, cn, dn벡터들을 얻게 되는데, 그 다음에는 임의의 점 Xin이 어떤 구간, 즉 몇 번째 구간에 속하는지 평가하여 그 구간에 맞는 bn(i), cn(i), dn(i)에 접근하여 값을 리턴한다.

이 함수를 시험해 보기위해서 다음과 같은 매트랩 커맨드를 실행한다.

hw#2에서 작성한 루틴을 사용했기 때문에 그 함수정의가 들어있는 m-file이 필요하다.

In matlab command window

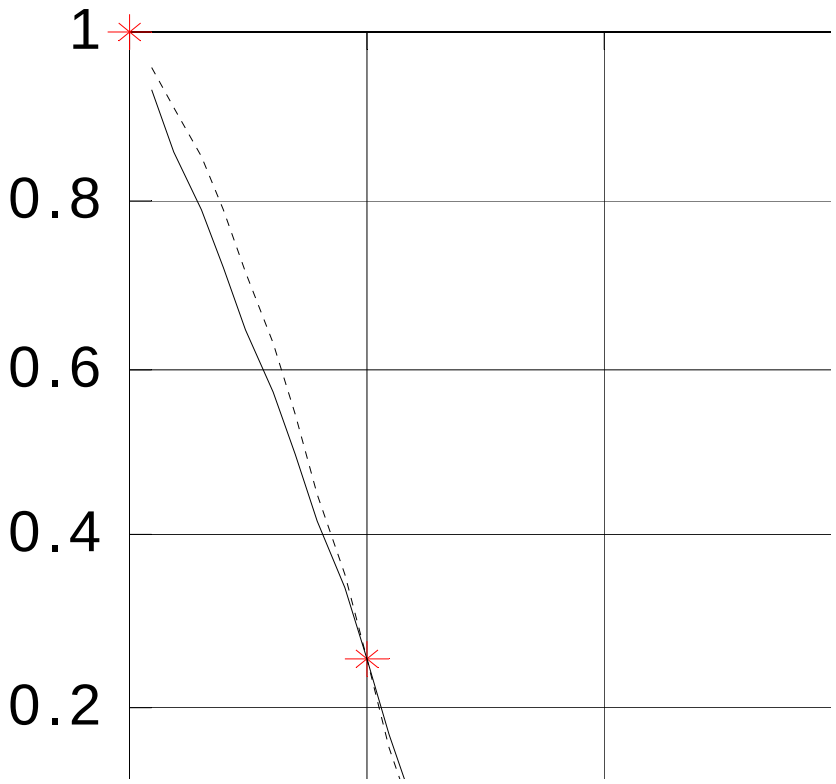
```
>>clear
>>x=0:0.1:1
>>y=ftest(x)
>>xs=0:0.01:1;
>>fx=ftest(xs);
>>for (i=1:length(xs)) cx(i)=cubicsf(x,y,xs(i));end
>>plot(x,y,'r*',xs,fx,'b:',xs,cx,'k-');
```



7. cubic spline fit 과 hw#2의 polynomial fit을 비교하기

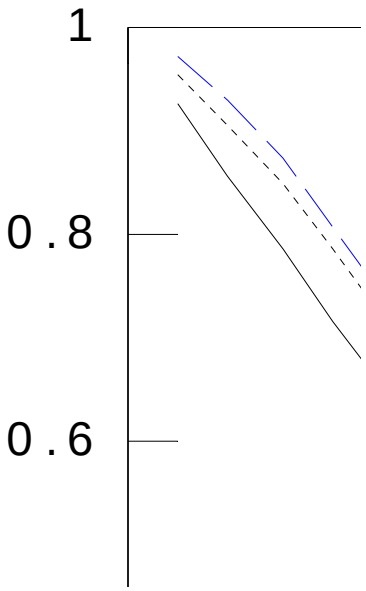
In matlab command window

```
>>clear
>>x=0:0.1:1
>>y=ftest(x)
>>xs=0.01:0.01:0.99;
>>fx=ftest(xs);
>>cf=cifi(x,y);
>>for (i=1:length(xs)) px(i)=plintp(xs(i),x,cf);end
>>for (i=1:length(xs)) cx(i)=cubicsf(x,y,xs(i));end
>>plot(x,y,'r*',xs,px,'k:',xs,cx,'k-');
>>grid on
```



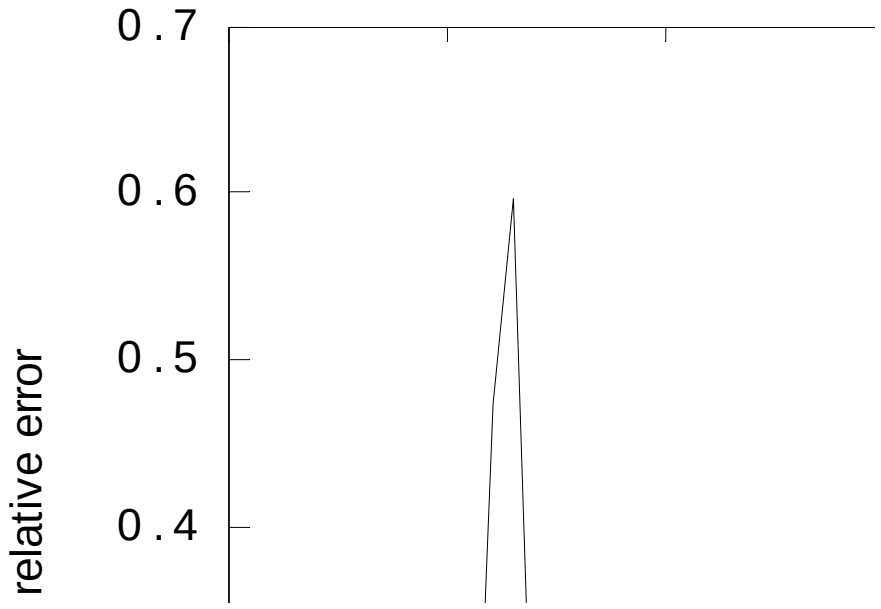
주어진 11개 데이터 포인트 들에 대해 polynomial fit 과 cubic spline fit을 겹쳐서 그려놓은 그림이 위에 있다. 두 곡선은 구간의 중간 지점 0.2에서 0.8 사이에서는 비교적 일치하나 구간의 끝쪽으로 갈수록 조금씩 달라지는 것을 볼 수 있다. 이것을 원 함수 f(x)와 비교해 보기 위해 다음을 입력하면

```
>>plot(x,y,'r*',xs,px,'k:',xs,cx,'k-',xs, fx,'b--');
```



두 그래프 모두 구간의 양끝 부분에서는 원 함수와 차이를 보이고 있음을 확인 할 수 있다. 좀더 정량적으로 보기 위해 각 fitting의 상대오차의 절대값을 비교해 보기 위해 다음의 명령을 입력하면

```
>>errorpx=abs((fx-px)./fx);
>>errorcx=abs((fx-cx)./fx);
>>plot(xs,errorpx,'k:',xs,errorcx,'k-');
```



구간 [0,0.2]에서는 cubic spline fit의 오차가 크고, 구간 [0.8,1]에서는 cubic spline fit의

오차가 작은 것을 관찰 할 수 있다.

이제 실제 함수의 양 끝값에서의 실제 미분값인 y'(0)=-2, y'(1)=-2*e^-2를 cubicsf.m file의 ypl, ypr 대신에 대입하고 cubicsft.m file로 이름을 바꿔 저장한 다음 cubic spline을 구해보면

In file cubicsf.m

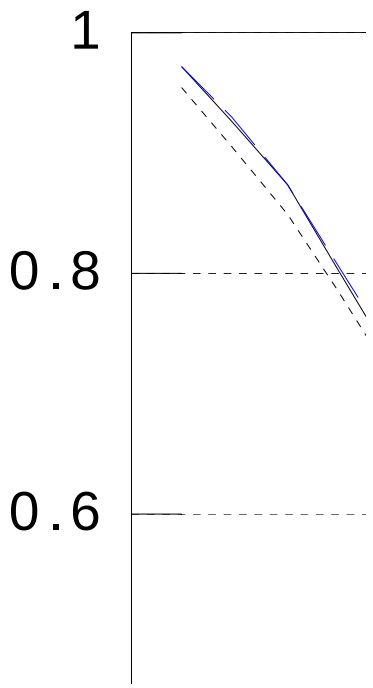
....

```
[bn,cn,dn]=cspline(XN,YN,-2,-2*exp(-2));
```

...

In matlab command window

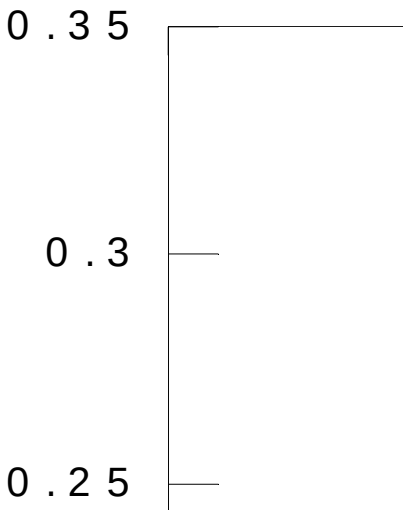
```
>>x=0:0.1:1;
>>y=ftest(x);
>>xs=0.01:0.01:0.99;
>>fx=ftest(xs);
>>cf=cifi(x,y);
>>for (i=1:length(xs)) px(i)=plintp(xs(i),x,cf);end
>>for (i=1:length(xs)) csx(i)=cubicsft(x,y,xs(i));end
>>plot(x,y,'r*',xs,px,'k:',xs,csx,'k-');
>>grid on
```



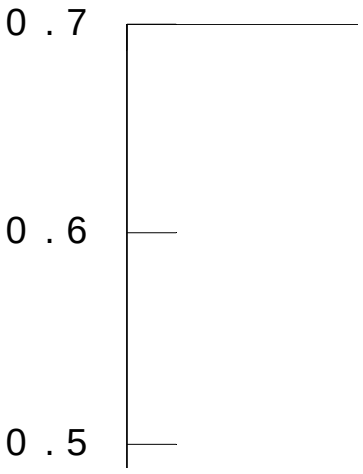
원 함수 f(x)와 구분이 되지 않을 정도로 cubic spline fitting이 잘 이루어 졌음을 볼 수 있다.

오차를 확인하기 위해

```
>>errorpx=abs((fx-px)./fx);
>>errorcsx=abs((fx-csx)./fx);
>>plot(xs,errorpx,'k:',xs,errorcsx,'k-');
```




```
>>plot(xs,errorpx,'k:',xs,errorcsx,'k-',xs,errorcx,'k-.');
```



dash dot 로 그려진 곡선은 3점을 통해 양끝 값의 기울기를 추정했을 때의 오차이고, 실선은 실제 미분값을 적용했을 때의 오차이다. 양끝 값에서 실제 미분값을 적용하면 구간의 끝 부분에서 cubic spline의 오차가 줄어들게 되어 polynomial fit의 오차보다 더 작아지는 것을 확인 할 수 있다.

3개의 포인트를 이용해서 구간끝의 미분값을 구해서 spline fit을 했을 때는 3개의 데이터 포인트가 비교적 멀리 떨어져 있어 정확하지 않은 (실제 함수의 미분값과는 다른) 미분값을 산출하게 되며, 이 정보를 사용하기 때문에 구간 양끝에서 오차가 크게 나오는 것이라고 생각할 수 있다. 따라서 구간 끝에서 실제 함수의 미분값을 사용하여 spline fit을 하면 오차가 현저히 줄어드는 것이라고 생각할 수 있다.

<부록1> 마지막 그래프를 얻기 위한 매트랩 커맨드 명령 스트림

```
>> clear
>> x=0:0.1:1;
y=ftest(x);
xs=0.01:0.01:0.99;
fx=ftest(xs);
cf=cifi(x,y);
for (i=1:length(xs)) px(i)=plintp(xs(i),x,cf);end
for (i=1:length(xs)) cx(i)=cubicsf(x,y,xs(i));end
for (i=1:length(xs)) csx(i)=cubicsft(x,y,xs(i));end
plot(x,y,'r*',xs,px,'k:',xs,cx,'k-');
errorpx=abs((fx-px)./fx);
errorcsx=abs((fx-csx)./fx);
errorcx=abs((fx-cx)./fx);
plot(xs,errorpx,'k:',xs,errorcsx,'k-',xs,errorcx,'k-.');
```

<부록2>

함수리스트

cspline.m
findslope.m
cubicsf.m
cubicsft.m

숙제 #2에 사용했던 함수의 재사용

ftest.m
cifi.m
plintp.m