2008년 2학기 HW#2

수치해석기초

HW#2 : Polynomial Interpolation

원자핵공학과
2003-12491
이 원 재

2. a) Lagrange polynomial의 계수 구하는 함수의 구현

In file cifi.m

```
function [arrayCi] = cifi(argX,argF)
% x좌표 벡터와 각 좌표에 대한 함수값 f벡터를 각각 argX 와 argF로 받은 다음에
% i가 0부터 n까지 각각에 대해 Ci값을 계산하고 그에fi를 곱하여 arrayCi의 i번째 성분으로 하여 리턴한다
n=length(argX);
n=n-1;              %0부터 인덱스 시작해서 n으로 끝나게 함
                    %벡터값 접근할때는 인덱스에 +1해서 접근
i=0;k=0;

for (i=0:n)                   %C0 부터 Cn까지 계수를 구함
   if (i==0)                  %k가 i가 아닐때를 제하기 위해 먼저 i가 0일때 k를 1부터 n까지..
      tempc=1;
      for (k=1:n)
          tempc=tempc*(argX(i+1)-argX(k+1));
      end
      tempc=argF(i+1)/tempc;     %i번째 함수값을 곱해주고 i번째 x에서
                                        %다른 모든점까지의 거리의 곱의 역수를 곱함
      arrayCi(i+1)=tempc;
   elseif (i~=n)              %i가 0과 n의 사이에 있을때
      tempc=1;
      for (k=0:i-1)
          tempc=tempc*(argX(i+1)-argX(k+1));
      end
                                        %i를 제외하고 k에 대해 0부터 n까지 계산
      for (k=i+1:n)
          tempc=tempc*(argX(i+1)-argX(k+1));
      end
      tempc=argF(i+1)/tempc;
      arrayCi(i+1)=tempc;
   else                       %i가 n과 같을때로서 k는 0부터 n-1까지
      tempc=1;
      for (k=0:n-1)
          tempc=tempc*(argX(i+1)-argX(k+1));
      end
      tempc=argF(i+1)/tempc;
      arrayCi(i+1)=tempc;
   end
end
```

%%%

b) 임의의 점 x에 대한 polynomial value를 리턴하는 함수의 구현

In file plintp.m (Polynomial Lagrange INTPolation)

```
function [approxF] = plintp(x,argX,cf)
% 계산을 원하는 점 x와 알려진 데이터 포인트의 x좌표 벡터,
% 그리고 앞에서 계산한 라그랑지 다항식 계수벡터를 입력인자로 받아서
% 원하는 점에서의 polynomial 값을 리턴한다.
n=length(argX);
n=n-1;
```

```
i=0;
poli=1;
tempf=0;

for (i=0:n)
   if (i==0)
      poli=1;
      for (k=1:n)
          poli=poli*(x-argX(k+1));
      end
      poli=cf(i+1)*poli;          %i번째 라그랑지다항식 계수와 (x-xk)의 곱을 구한것을 곱해서
      tempf=tempf+poli;                    %임시저장변수에 더함
   elseif (i~=n)
      poli=1;
      for (k=0:i-1)
          poli=poli*(x-argX(k+1));
      end                         %k==i일때는 skip!
      for (k=i+1:n)
          poli=poli*(x-argX(k+1));
      end
      poli=cf(i+1)*poli;          %i번째 라그랑지다항식 계수와 (x-xk)의 곱을 구한것을 곱해서
      tempf=tempf+poli;                    %임시저장변수에 더함
   else
      poli=1;
      for (k=0:n-1)
          poli=poli*(x-argX(k+1));
      end
      poli=cf(i+1)*poli;          %i번째 라그랑지다항식 계수와 (x-xk)의 곱을 구한것을 곱해서
      tempf=tempf+poli;                    %임시저장변수에 더함
   end
end
approxF=tempf;                    %값 리턴
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3. 주어진 damped oscillation 함수에 대한 11개 데이터 포인트를 이용하여 101개의 점에서 값 구하기

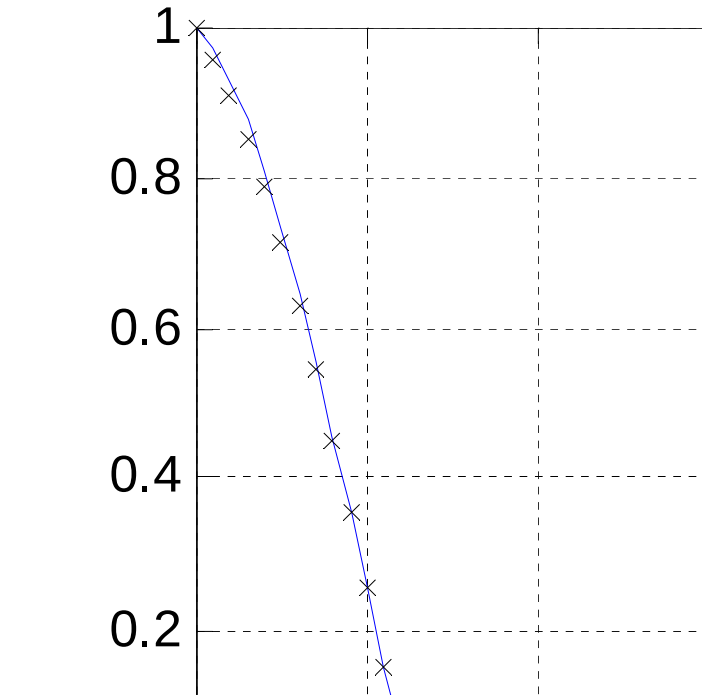
주어진 damped oscillation 함수를 매트랩 function file로 다음과 같이 구현한다

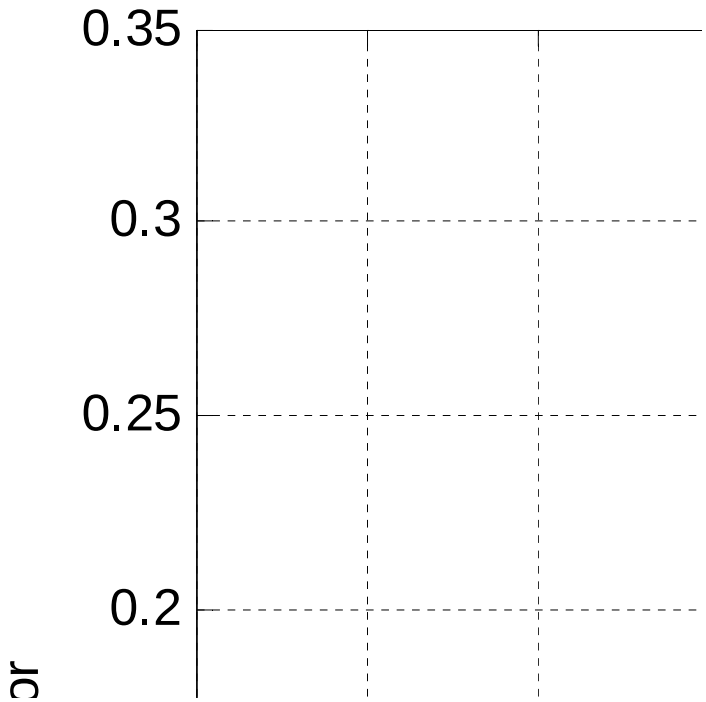
```
%%%%%%%%%%%%%%%%%%%%%%
function [y] = ftest(x)
n=length(x);
for (i=1:n)
        y(i)=exp(-2*x(i))*cos(4*pi*x(i));
end
%%%%%%%%%%%%%%%%%%%%%%
```

다음과 같은 명령어를 매트랩 커맨드 윈도우에서 사용하면

```
>> x=0:0.1:1
x =
  Columns 1 through 8
 0    1.0000e-001    2.0000e-001    3.0000e-001    4.0000e-001    5.0000e-001    6.0000e-001
7.0000e-001
  Columns 9 through 11
 8.0000e-001   9.0000e-001   1.0000e+000

>> y=ftest(x)
y =
  Columns 1 through 8
1.0000e+000     2.5300e-001   -5.4230e-001   -4.4400e-001     1.3885e-001     3.6788e-001
9.3074e-002  -1.9950e-001
  Columns 9 through 11
 -1.6334e-001   5.1080e-002   1.3534e-001

>> cf=cifi(x,y)
cf =
 Columns 1 through 8
   2.7557e+003   -6.9720e+003   -6.7250e+004    1.4682e+005    8.0353e+004   -2.5547e+005
5.3862e+004   6.5973e+004

 Columns 9 through 11
 -2.0255e+004  -1.4076e+003   3.7295e+002
>> xs=0:0.01:1;
>> for(i=1:length(xs)) px(i)=plintp(xs(i),x,cf); end
>> fx=ftest(xs);
>> plot(xs,fx,xs,px,'cx')
>> grid on
>> error=abs((fx-px)./fx);
>> plot(xs,error)
>> grid on
```

이렇게 얻은 그래프를 수록하면





원래의 주어진 데이터 포인트들에서는 정확하게 들어맞아서 오차가 거의 0이나 그 외의 위치에서는 정확성이 보장되지 않음을 확인 할 수 있다.

4. 추가된 데이터 포인트를 사용하여 임의의 점에서의 오차 구하기

다음과 같이 추가된 데이터 포인트를 사용해서 x점에서 오차를 구하는 함수를 작성한다.

In file ex.m

```
function [err] = ex(x, argX, argY, xadd, yadd)
%첫번째 인자는 오차를 구하고자 하는 점, 두번째와 세번째 인자는 주어진 데이터 포인트들,
%네번째와 다섯번째 인자는 추가적인 데이터 포인트이다.
%추가적인 데이터 포인트를 포함하여 차분상을 구하면
%이 차분상이 곧 원함수 f(x)를 n+1번 미분한 것을 (n+1)!로 미분한것이 되어 x를 위한 오차를
%추가적인 데이터 포인트를 사용하여 근사적으로 구할수 있다.

n=length(argX);      %원래주어진 데이터 포인트 좌표의 개수를 취한다
argX(n+1)=xadd;      %데이터 포인트 x좌표벡터에 추가적인 데이터 포인트 좌표를 추가한다
argY(n+1)=yadd;      %데이터 포인트 값 벡터에 추가적인 데이터 포인트에 해당되는 값을 추가한다.
                     %위에 주어진 벡터들을 사용하여 n+1차 차분상을 제공된 함수를 사용하여 구한다
df=divdif(argX,argY);
tempd=1;
for (i=0:n-1)
   tempd=tempd*(x-argX(i+1)); %i=1부터 n까지 접근하여 본래 주어진 데이터 포인트 모두에 대해 계산한다.
end
err=df*tempd;        %주어진 E(x)식의 구현
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ddn,dd]=divdif(x,y)
%
% obtain divided difference
n=length(x);
dda=y; %intermediate array
dd=zeros(1,n); %divided difference of various order
dd(1)=dda(1);
for i=1:n-1
    for j=1:n-i
        dda(j)=(dda(j+1)-dda(j))/(x(j+i)-x(j));
    end
    dd(i+1)=dda(1); %i-th order divided difference
end
ddn=dd(n);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% divdif 제공된 함수 이용
```

이제 추가된 데이터 포인트로부터 오차를 구하면

```
>> x1=0.85;
>> y1=ftest(x1)
y1 =
 -5.6452e-002
>> ex(0.95,x,y,x1,y1)
ans =
 -1.9987e-002

>> yp=plintp(x1,x,cf)
yp =
 -5.9608e-002
```

```
>> yf=ftest(x1)
yf =
 -5.6452e-002
>> yf-yp
ans =
  3.1559e-003
>>
```

따라서 n+1차 차분상을 사용하여 오차를 구하면 -1.9987e-002 가 나오고, n개 데이터 포인트를 사용한 라그랑지 다항식에서 구한 x=0.95에서의 오차는 3.1559e-003 가 나와서 절대값으로 보면 n+1차 차분상을 사용하여 구한 오차가 더 크게 평가되었음을 알 수 있다.