Numerical Analysis

Curve Fitting Technique: Matlab Programming Applications

목 차

Data and Approximation Function

Data $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_j, y_j), \cdots, (x_K, y_K)\} = \{(x_j, y_j)\}_{j=1}^{j=K}$

Polynomial Approximation Function

$$y = f(x; \mathbf{a}) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$= \begin{pmatrix} 1 & x & x^2 & x^3 & x^4 & \cdots & x^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \boldsymbol{\varphi}^T(x)\mathbf{a}, \quad \boldsymbol{\varphi}(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^n \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

$$\boxed{y = f(x; \mathbf{a}) = \boldsymbol{\varphi}^T(x)\mathbf{a}}$$

$\boldsymbol{\varphi}(x)$ **: Regressor functions or basis functions**

$\mathbf{a}$ **: Regression coefficients of Interpolation coefficients**

Curve-fitting Technique General

Error vector and Curve-Fitting Techniques

Errors in the Approximation Function $\boxed{y = f(x;\mathbf{a}) = \boldsymbol{\varphi}^T(x)\mathbf{a}}$

$$
\begin{aligned}
e_1 &= y_1 - f(x_1;\mathbf{a}) = y_1 - \boldsymbol{\varphi}^T(x_1)\mathbf{a} \\
e_2 &= y_2 - f(x_2;\mathbf{a}) = y_2 - \boldsymbol{\varphi}^T(x_2)\mathbf{a} \\
&\vdots \\
e_K &= y_K - f(x_K;\mathbf{a}) = y_K - \boldsymbol{\varphi}^T(x_K)\mathbf{a}
\end{aligned}
$$

$$
\boxed{\mathbf{e} = \mathbf{y} - \begin{pmatrix} \boldsymbol{\varphi}^T(x_1) \\ \boldsymbol{\varphi}^T(x_2) \\ \vdots \\ \boldsymbol{\varphi}^T(x_K) \end{pmatrix} \mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{a}} \quad \boldsymbol{\varphi}(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^n \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}
$$

Least-square Regression $K >> n$

$$
\min_{\mathbf{a}} E = \mathbf{e}^T\mathbf{e} \rightarrow \quad \boxed{\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \mathbf{e})} \quad \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_K \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix}
$$

Polynomial Interpolation $K = n$

$$
\mathbf{e} = 0 \rightarrow \quad \boxed{\mathbf{a} = \mathbf{X}^{-1}\mathbf{y}}
$$

Curve-fitting Technique General

Classification of Data

Total Prepared Data = Training Data + Validation Data

$$D = D_{training} \quad \cup \quad D_{validation}$$

$D_{training}$ **: used to determine the regression or interpolation coefficients**

$D_{validation}$ **: used to validate accuracy of the approximation function**

Random number generator in Matlab: *r* = rand $(0 \le r = rand \le 1)$

```
>> rand      →  ans =  0.6324
>> rand(1)   →  ans =  0.0975

>> rand(2)   →  ans =  0.2785  0.9575
                       0.5469  0.9649

>> rand(3)   →  ans =  0.1576  0.4854  0.4218
                       0.9706  0.8003  0.9157
                       0.9572  0.1419  0.7922

>> rand(1,3) →  ans =  0.8491  0.9340  0.6787
```

Data Generation for Curve-fitting Test (I) : Generating Function

Generating function

$$y = g(x) = x + \sin(2\pi x) - 0.5\cos(4\pi x) + 1, \quad (0 \le x \le 2)$$

Computing noisy data

$$y_n = g(x) = x + \sin(2\pi x) - 0.5\cos(4\pi x) + 1 + n(x), \quad (0 \le x \le 2)$$

Pseudo-code to Generate Data with the random noise (the maximum amplitude= alpa)

```
(1) r1= rand                   ;
(2) x  = 2.0*r1                ;
(3) r2= rand                   ;
(4) n1= alpa*(2.0*r2-1.0)      ;
(3) y  =g(x) + n1              ;
```

Data Generation for Curve-fitting Test (I) : Generating Function

Generating function

$$y = g(x) = x + \sin(2\pi x) - 0.5\cos(4\pi x) + 1, \quad (0 \le x \le 2)$$

Computing noisy data

$$y_n = g(x) = x + \sin(2\pi x) - 0.5\cos(4\pi x) + 1 + n(x), \quad (0 \le x \le 2)$$

Pseudo-code to Generate Data with the random noise (the maximum amplitude= alpa)

```
(1) r1= rand                ;
(2) x  = 2.0*r1             ;
(3) r2= rand                ;
(4) n1= alpa*(2.0*r2-1.0)   ;
(3) y  =g(x) + n1           ;
```

❑ Data Generation for Curve-fitting Test (II) : Data_Generation.m

```
%----------------------------------------
%  Data Generation
%    Input:
%          N = number of total data
%          alpa = niose amplitude
%    Output
%          xe(N,1) : independent variable x  (sequential order)
%          ye(N,1) : exact dependent function values
%          xn(N,1) : independent variable x  (random order)
%          yn(N,1) : noisy dependent function values
%----------------------------------------
%  (1) Initialize
%----------------------------------------
xe(1:N,1)  = 0.0;
ye(1:N,1)  = 0.0;
xn(1:N,1)  = 0.0;
yn(1:N,1)  = 0.0;
%----------------------------------------
%  (2) Exact Data (sequential order)
%----------------------------------------
dx = 2.0/(N-1)          ;
for j=1:N
    x1=dx*(j-1)         ;
%
    xe(j,1) = x1          ;
    ye(j,1) = x1 + sin(2*pi*x1) - 0.5*cos(4*pi*x1) + 1.0      ;
end
%----------------------------------------
```

Data Generation for Curve-fitting Test (II) : Data_Generation.m

```
%----------------------------------------
%  (3) Noisy Data  (random sequence)
%----------------------------------------
for j=1:N
    r1=rand              ;
    x1=2.0*r1            ;
    r2=rand              ;
    n1=alpa*(2.0*r2-1.0) ;
%
    xn(j,1) = x1          ;
    yn(j,1) = x1 + sin(2*pi*x1) - 0.5*cos(4*pi*x1) + 1.0  + n1;
end
%----------------------------------------
```

Data Generation for Curve-fitting Test (III) : Data_Generation_Main.m

```
%----------------------------------------
% (1) Number of data
%----------------------------------------
%    N=100  ; alpa = 0.0 ;
%----------------------------------------
% (2) Generation of Total Data
%----------------------------------------
    Data_Generation    ;
%----------------------------------------
% (3) Classification of Training Data (80 %) and Validation Data (20%)
%----------------------------------------
% (3-1) Training Data (80 %)
%----------------------------------------
    Nt = int16(N*0.8)    ;
    xt(1:Nt,1)=xn(1:Nt,1);
    yt(1:Nt,1)=yn(1:Nt,1);
%----------------------------------------
% (3-2) Validation Data (20%)
%----------------------------------------
    Nv= N - Nt            ;
    xv(1:Nv,1)=xn(Nt+1:N,1);
    yv(1:Nv,1)=yn(Nt+1:N,1);
%----------------------------------------
```

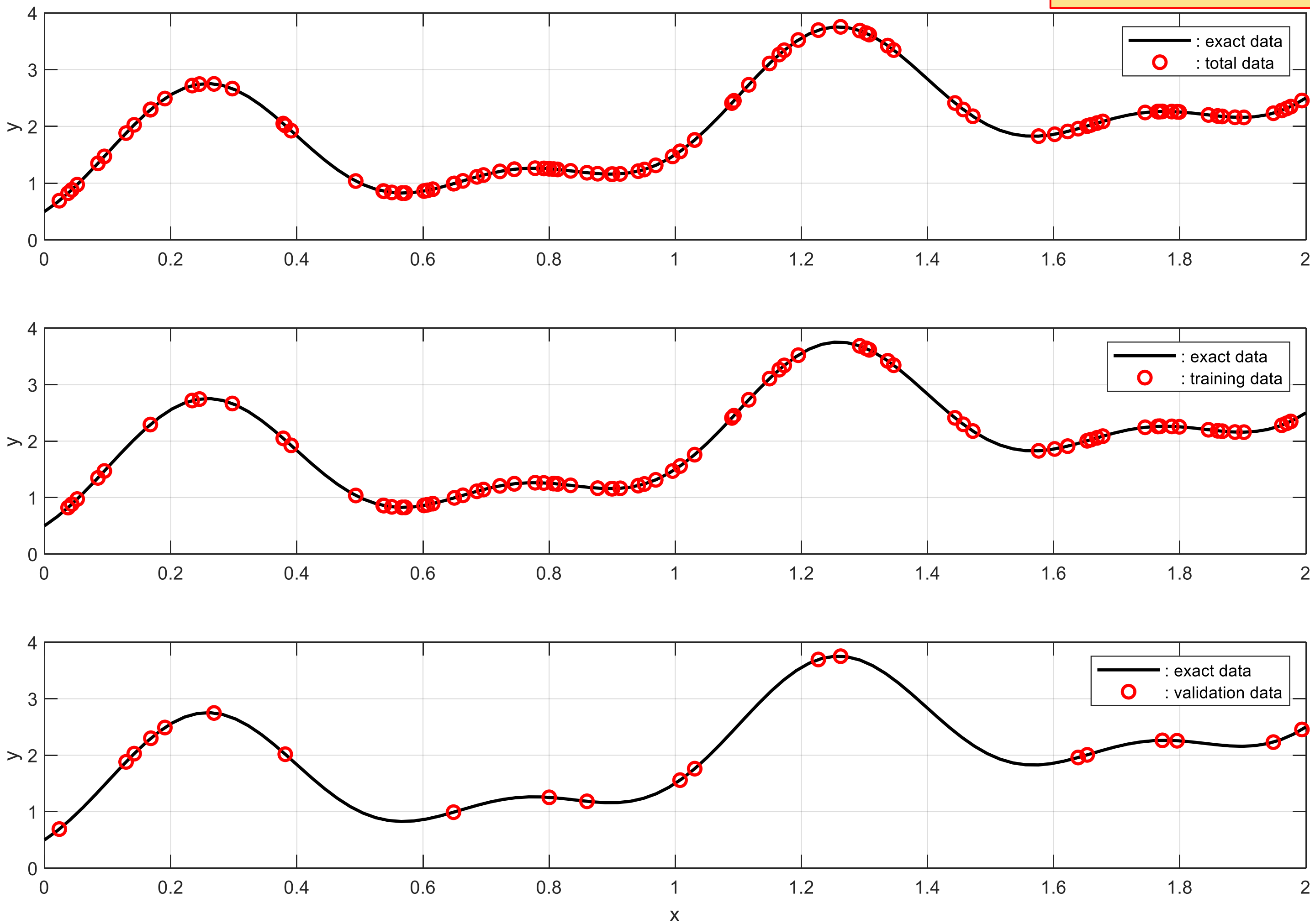
Data Generation for Curve-fitting Test (III) : Data_Generation_Main.m

```
%--------------------------------------------
% (4) Plot Training Data (80 %) and Validation Data (20%)
%--------------------------------------------
LW=1.5; L1 = 1.5 ;
%--------------------------------------------
% (4-1) Total Data
%--------------------------------------------
figure(1);set(gcf,'DefaultLineLineWidth',LW);set(gca,'DefaultLineLineWidth',LW)
    p1= plot(xe(:,1),ye(:,1),'-k')  ;grid on; hold on ;
    p2= plot(xn(:,1),yn(:,1),'ro') ;
    legend([p1,p2],': exact data',': total data')
%--------------------------------------------
% (4-2) Training Data (80 %)
%--------------------------------------------
figure(2);set(gcf,'DefaultLineLineWidth',LW);set(gca,'DefaultLineLineWidth',LW)
    p1= plot(xe(:,1),ye(:,1),'-k')  ;grid on; hold on ;
    p2= plot(xt(:,1),yt(:,1),'ro') ;
    legend([p1,p2],': exact data',': training data')
%--------------------------------------------
% (4-3) Validation Data (20%)
%--------------------------------------------
figure(3);set(gcf,'DefaultLineLineWidth',LW);set(gca,'DefaultLineLineWidth',LW)
    p1= plot(xe(:,1),ye(:,1),'-k')  ;grid on; hold on ;
    p2= plot(xv(:,1),yv(:,1),'ro') ;
    legend([p1,p2],': exact data',': validation data')
%--------------------------------------------
```

❑ Data Generation for Curve-fitting Test (IV) : Generated Data

N= 100 ; alpa = 0.0 without the measurement noise

```
N=100  ; alpa = 0.0 ;
Data_Generation_Main;
```

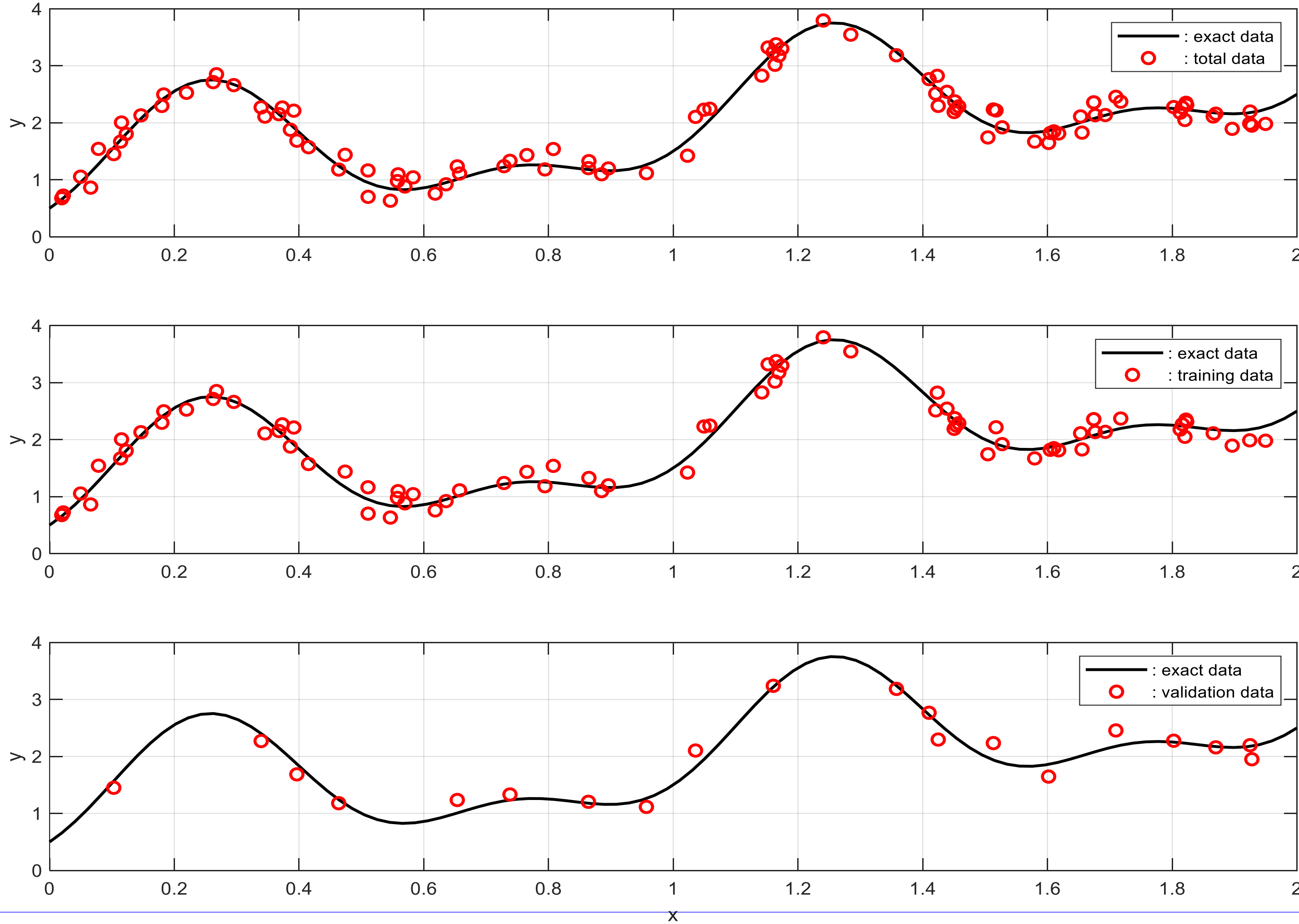


Curve-fitting Technique General

Data Generation for Curve-fitting Test (IV) : Generated Data

N= 100 ; alpa = 0.3 with the measurement noise (amplitude=0.3)

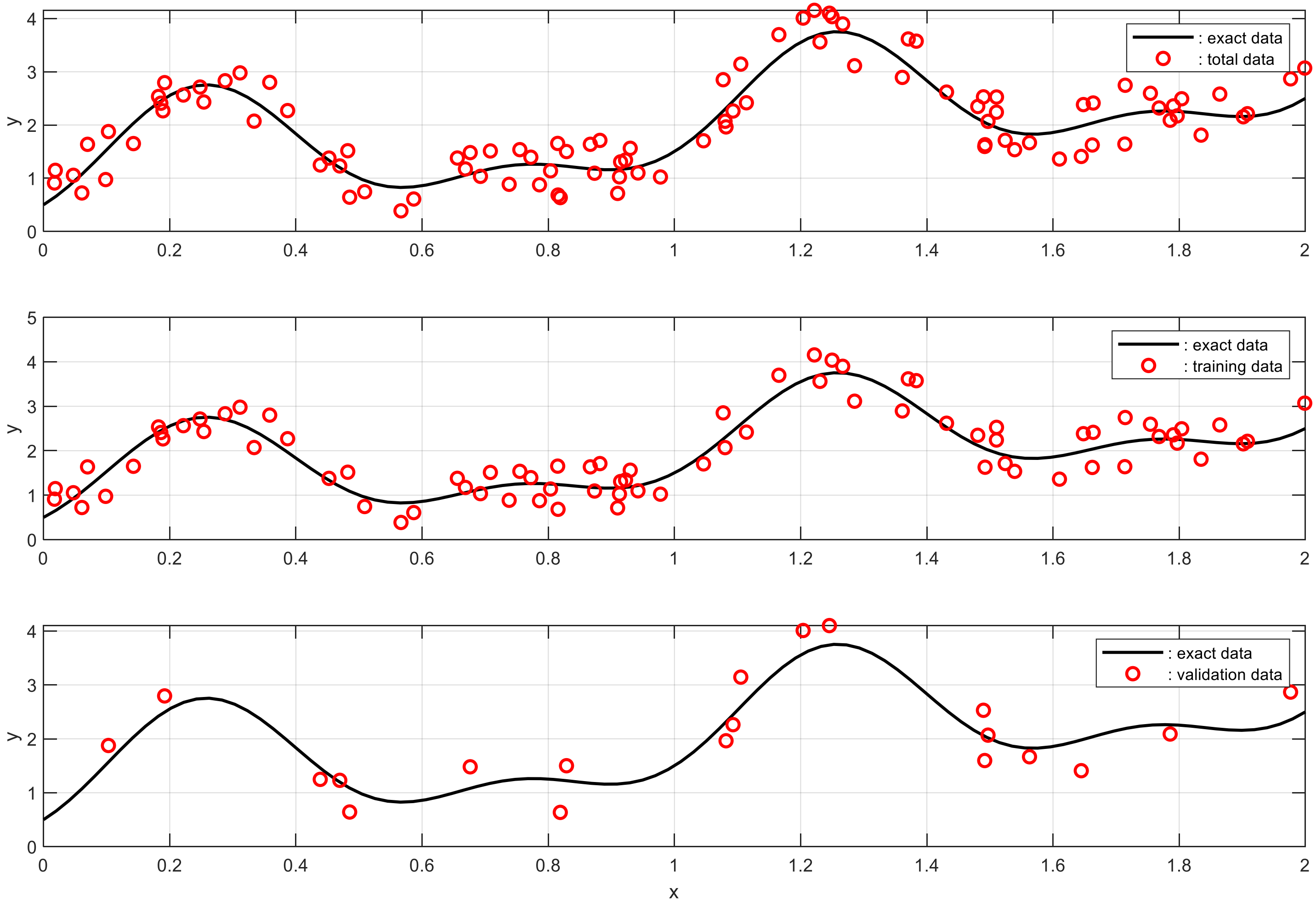
```
N=100  ; alpa = 0.3 ;
Data_Generation_Main;
```



❑ Data Generation for Curve-fitting Test (IV) : Generated Data

N= 100 ; alpa = 0.6 with the measurement noise (amplitude=0.6)

```
N=100  ; alpa = 0.6 ;
Data_Generation_Main;
```



목　차

Polynomial Regression

Regression function

$$y = f(x;\mathbf{a}) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \boldsymbol{\varphi}^T(x)\mathbf{a}, \quad \boldsymbol{\varphi}(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^n \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

Regression Coefficients

$$\boxed{\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

$$\boxed{\mathbf{e} = \mathbf{y} - \begin{pmatrix} \boldsymbol{\varphi}^T(x_1) \\ \boldsymbol{\varphi}^T(x_2) \\ \vdots \\ \boldsymbol{\varphi}^T(x_K) \end{pmatrix}\mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{a}} \quad \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_K \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix}$$

Least-Square Regression Analysis with Polynomials

❑ Program Structures

- A1_Polynomial_Regression_Main — **Main Program**
- Data_Generation
- Data_Generation_Main — **Data Generation**
- LU_Backward_substitution
- LU_decomposition — **LU-decomposition**
- LU_Forward_substitution

Input Data in Main Program

```
Norder = 3 ; N=100  ; alpa = 0.0 ;
```

Norder: Order of Polynomial
N : Number of total data (Training/Validation Data)
alpa : Noise amplitude

❑ Program Structures: A1_Polynomial_Regression_Main.m

```
%-----------------------------------------
%  Data Generation
%    Input:
%          Norder = order of Regression Polynomial
%          N      = number of total data
%          alpa   = niose amplitude
%    Output
%          xe(N,1) : independent variable x  (sequential order)
%          ye(N,1) : exact dependent function values
%          xn(N,1) : independent variable x  (random order)
%          yn(N,1): noisy dependent function values
%-----------------------------------------
%   Polynomial Regression
%-----------------------------------------
% (1) Number of data
%-----------------------------------------
    clear all ; close all;
%
    Norder = 3 ; N=100  ; alpa = 0.0 ;
%-----------------------------------------
% (2) Generation of Total Data
%-----------------------------------------
    Data_Generation_Main    ;
%-----------------------------------------
```

Least-Square Regression Analysis with Polynomials

Program Structures: A1_Polynomial_Regression_Main.m

```
%-----------------------------------------
% (3) Build Matrix A consisting of Regression Function Using Training Data
%-----------------------------------------
% (3-1) Regressor function
%-----------------------------------------
    NN = Norder + 1     ;
    A(1:Nt,1:NN) = 0.0 ;
    for j=1:Nt
        xx = xn(j,1)    ;   A(j,1) = 1.0    ;   x1 = 1.0        ;
        for k=2:NN
            x1 = x1*xx ;   A(j,k) = x1;
        end
    end
%-----------------------------------------
% (3-2) Output function
%-----------------------------------------
    B_vec(1:NN,1) = A'*yn(1:Nt,1) ; % =Transpose(A)*y
%-----------------------------------------
% (3-3) Leading Matrix
%-----------------------------------------
    AA = A'*A                  ; % =Transpose(A)*A
%-----------------------------------------
% (3-4) Regression Coefficients using LU-decomposition
%-----------------------------------------
    [AL_mat, AU_mat] = LU_decomposition(AA);
    [Y_vec] = LU_Forward_substitution (AL_mat, B_vec) ; % Forward Substitution    (Ly=b)
    [A_vec] = LU_Backward_substitution(AU_mat, Y_vec) ; % Regression Coefficients (Ua=y)
%-----------------------------------------
%   Please, don't use  A_vec = inv(AA)*B_vec;
%-----------------------------------------
```

❑ Program Structures: A1_Polynomial_Regression_Main.m

```
%----------------------------------------
% (4) Validation Using Exact Data
%----------------------------------------
    yer(1:N,1) = 0.0 ;
    for j=1:N
        xx = xe(j,1)    ;
        yer(j,1) = A_vec(1) ;
        x1 = 1.0        ;
        for k=2:NN
            x1 = x1*xx ;
            yer(j,1) = yer(j,1) + A_vec(k)*x1 ;
        end
    end
%----------------------------------------
% (5) Validation Using Validation Data
%----------------------------------------
    yvr(1:Nv,1) = 0.0  ;
    for j=1:Nv
        xx = xv(j,1)    ;
        yvr(j,1) = A_vec(1) ;
        x1 = 1.0        ;
        for k=2:NN
            x1 = x1*xx ;
            yvr(j,1) = yvr(j,1) + A_vec(k)*x1 ;
        end
    end
%----------------------------------------
```

Least-Square Regression Analysis with Polynomials

❑ Program Structures: A1_Polynomial_Regression_Main.m

```
%-----------------------------------------
% (6) Plot Results
%-----------------------------------------
LW=1.5; L1 = 1.5 ;
%-----------------------------------------
figure(5);set(gcf,'DefaultLineLineWidth',LW);set(gca,'DefaultLineLineWidth'
,LW)
    subplot(2,1,1), p1= plot(xe(:,1),ye(:,1),'-k')  ;grid on; hold on ;
ylabel('y') ;
                    p2= plot(xe(:,1),yer(:,1),'ro') ;
                   legend([p1,p2],': exact data',': regression polynomial')
    subplot(2,1,2), p1= plot(xe(:,1),ye(:,1),'-k')  ;grid on; hold on ;
xlabel('x') ; ylabel('y') ;
                    p2= plot(xv(:,1),yvr(:,1),'ro') ;
                    p3= plot(xt(:,1),yt(:,1),'bs') ;
                    legend([p1,p2,p3],': exact data',': Validation
point',': traing data')
%-----------------------------------------
```

Program Structures: LU_decomposition.m

```
function [L_mat, U_mat] = LU_decomposition(A_mat)
    Ndim = length(A_mat(:,1));
%
    U_mat(1,1:Ndim) = A_mat(1,1:Ndim)              ;
    L_mat(1,1)      = 1.0                          ;
    L_mat(2:Ndim,1) = A_mat(2:Ndim,1)/U_mat(1,1)   ;
 %
     for j=2:Ndim
%
        for k=j:Ndim
            sum=0;
            for m=1:j-1  ;    sum = sum + L_mat(j,m)*U_mat(m,k) ;     end
            U_mat(j,k)=A_mat(j,k)-sum;
        end
 %
        L_mat(j,j)      = 1.0              ;
        for k=j+1:Ndim
            sum=0;
            for m=1:j-1  ;    sum = sum + L_mat(k,m)*U_mat(m,j) ;     end
            L_mat(k,j)=(A_mat(k,j)-sum)/U_mat(j,j);
        end
    end

end
```

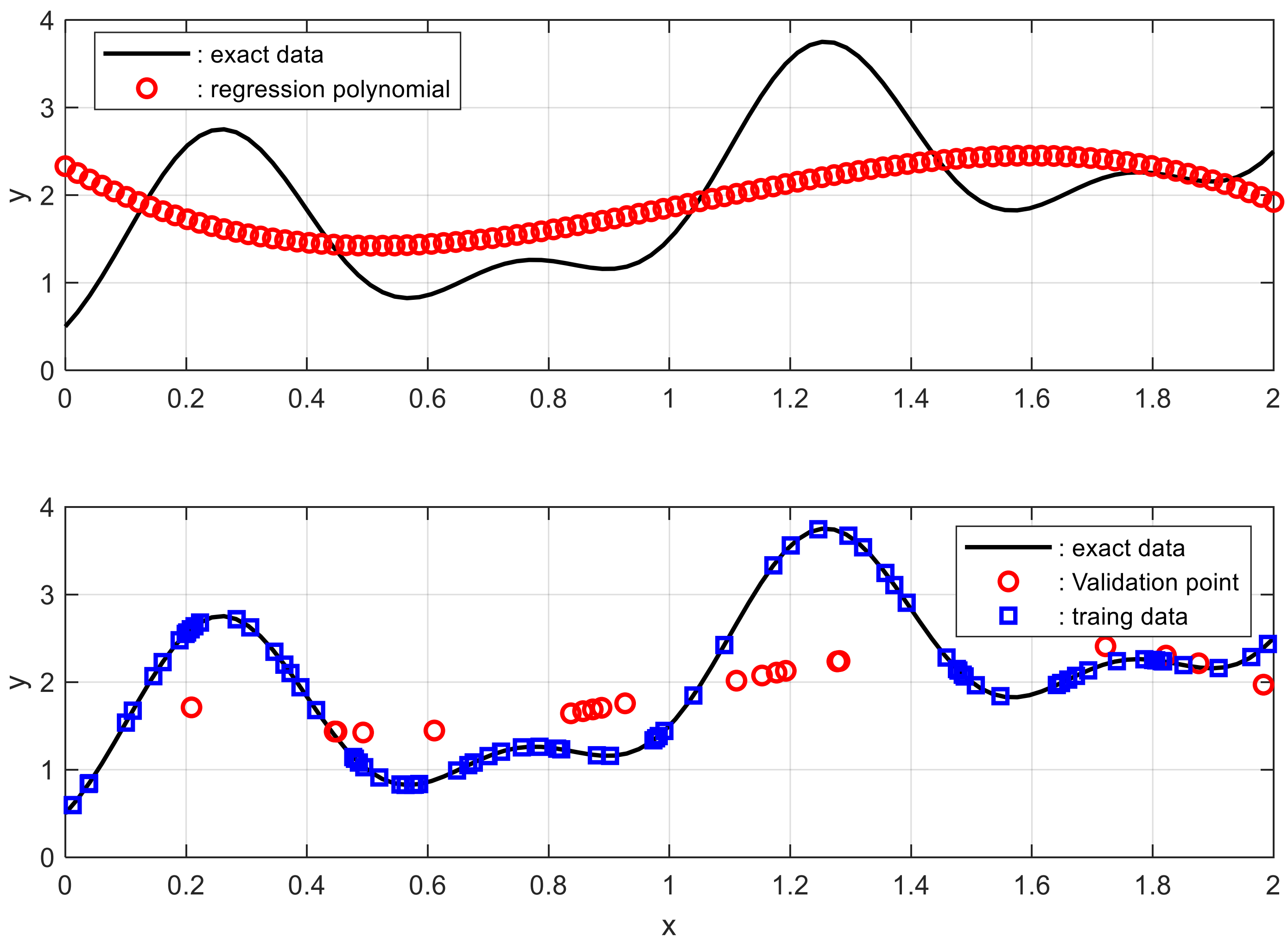
❑ Program Structures: LU_Forward_substitution.m /LU_Backward_substitution.m

```
function [Y_vec] = LU_Forward_substitution(L_mat, E_vec)
    Ndim  = length(L_mat(:,1));

    Y_vec(1,1)=E_vec(1,1);
    for j=2:Ndim
        sum =0.0 ;
        for k=1:j-1  ;          sum =sum + L_mat(j,k)*Y_vec(k,1);         end
        Y_vec(j,1)= E_vec(j,1) - sum ;
    end
End

function [X_vec] = LU_Backward_substitution(U_mat, Y_vec)
     Ndim  = length(U_mat(:,1));

    X_vec(Ndim,1)=Y_vec(Ndim,1)/U_mat(Ndim,Ndim);
%
    for j=Ndim-1:-1:1
        sum = 0.0 ;
        for k=j+1:Ndim ;      sum = sum + U_mat(j,k)*X_vec(k,1);      end
        X_vec(j,1)=(Y_vec(j,1) - sum)/U_mat(j,j);
    end
%
end
```

Curve-fitting Technique General

Results of Polynomial Regression

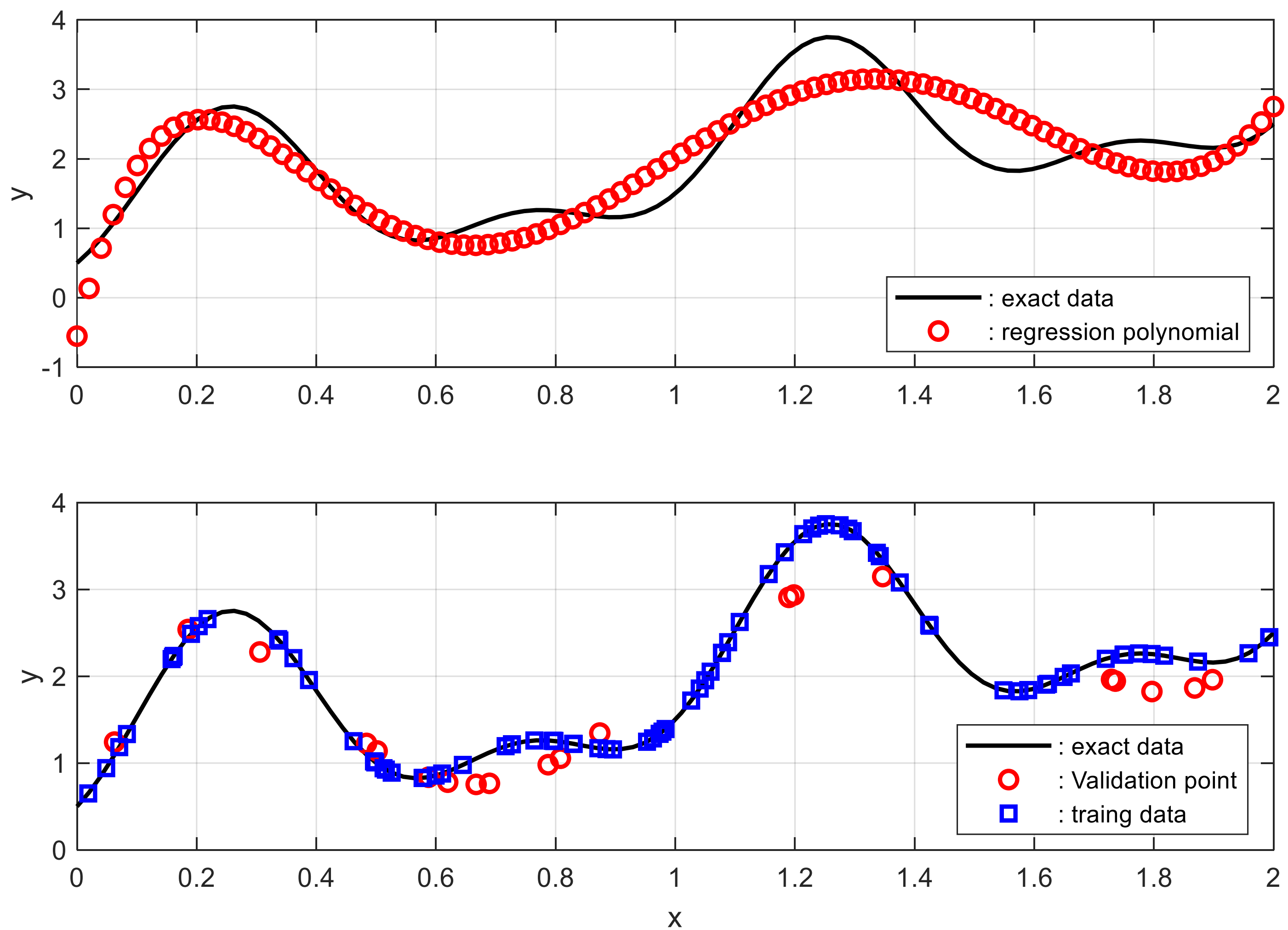
Norder = 3 ; N= 100 ; alpa = 0.0



Curve-fitting Technique General

Results of Polynomial Regression

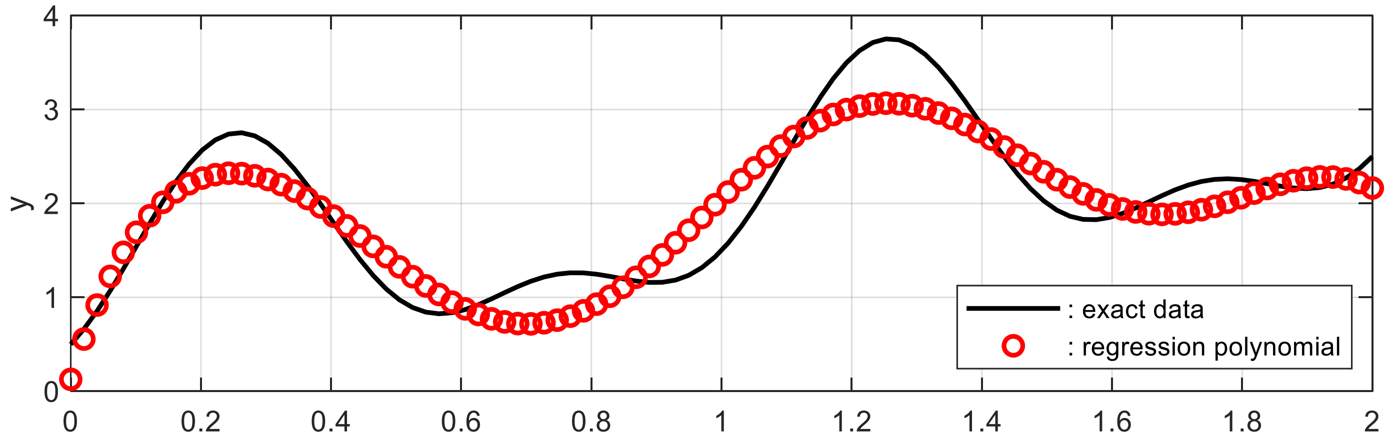
Norder = 6 ; N= 100 ; alpa = 0.0

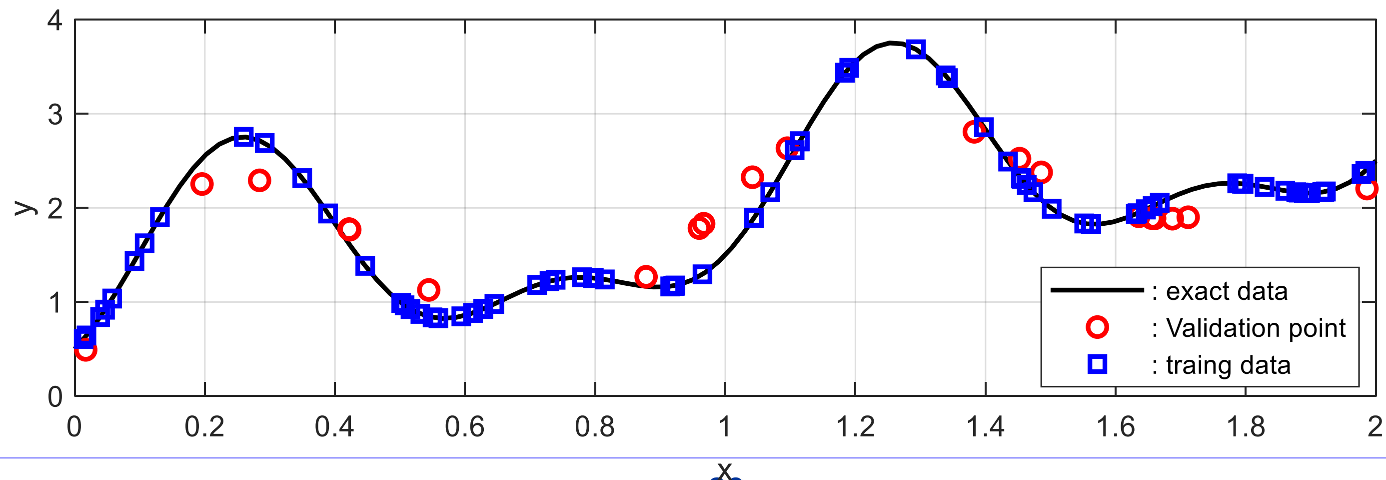


Curve-fitting Technique General

Results of Polynomial Regression

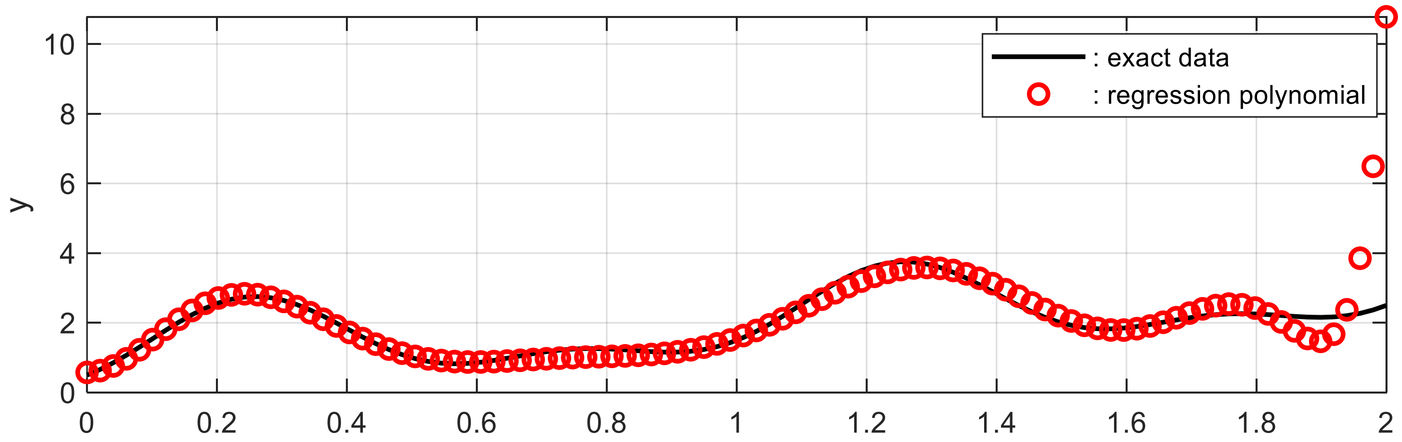
Norder = 9 ; N= 100 ; alpa = 0.0

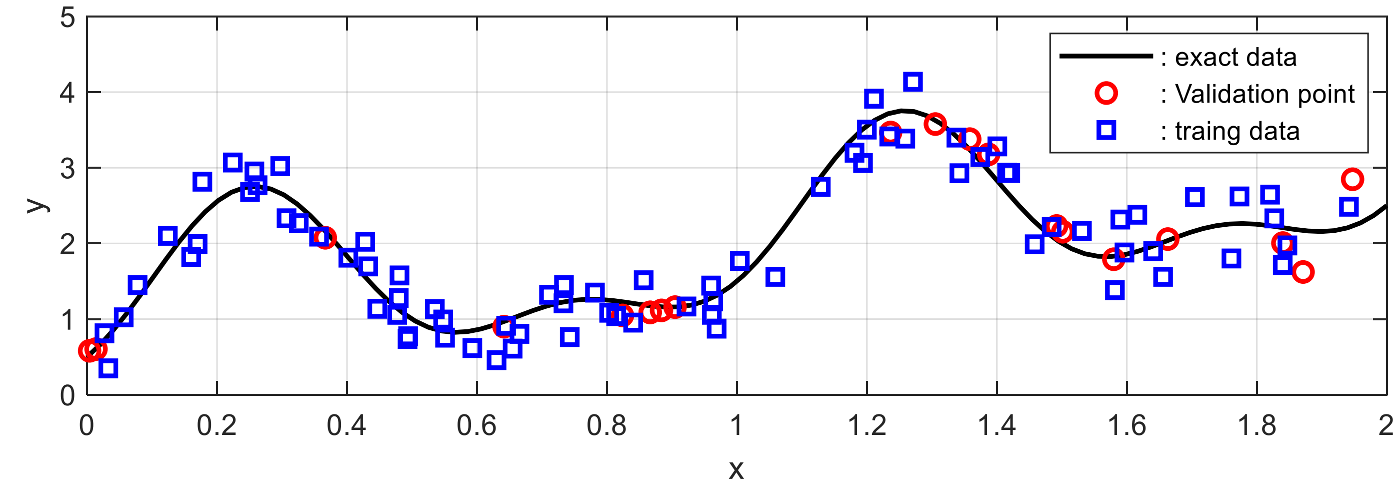




Results of Polynomial Regression

Norder = 12 ; N= 100 ; alpa = 0.5





목 차

Least-Square Regression Analysis with General Function

Regression using General Function

Regression function

$$
\begin{aligned}
y &= f(x;\mathbf{a}) \\
&= a_0 + a_1 x + a_2 x^2 + a_3 \cos(2\pi x) + a_4 \sin(2\pi x) + a_5 \cos(4\pi x) + a_6 \sin(4\pi x) \\
&= \boldsymbol{\varphi}^T(x)\mathbf{a}
\end{aligned}
$$

Regression Coefficients

$$\boxed{\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

$$
\boldsymbol{\varphi}(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \cos(2\pi x) \\ \sin(2\pi x) \\ \cos(4\pi x) \\ \sin(4\pi x) \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}
$$

Program Structures

A1_LSR_General_Function_Main	Main Program
A1_Polynomial_Regression_Main	
Data_Generation	
Data_Generation_Main	Data Generation
LU_Backward_substitution	
LU_decomposition	LU-decomposition
LU_Forward_substitution	

Input Data in Main Program

```
N=100  ; alpa = 0.0 ;
```

N : Number of total data (Training/Validation Data)
alpa : Noise amplitude

❑ Program Structures: A1_LSR_General_Function_Main.m

```
%-------------------------------------------
% (1) Number of data
%-------------------------------------------
   clear all ; close all;
%
   N=100  ; alpa = 0.5 ;
%-------------------------------------------
% (3-1) Regressor function
%-------------------------------------------
   NN = 7   ;
   A(1:Nt,1:NN) = 0.0 ;
   for j=1:Nt
      x1 = xn(j,1)    ;
      x2= 2.0*pi*x1   ;
      x4= 4.0*pi*x1   ;

      A(j,1) = 1.0      ;
      A(j,2) = x1       ;
      A(j,3) = x1*x1    ;
      A(j,4) = cos(x2) ;
      A(j,5) = sin(x2) ;
      A(j,6) = cos(x4) ;
      A(j,7) = sin(x4) ;
   end
%-------------------------------------------
```

Program Structures: A1_LSR_General_Function_Main.m

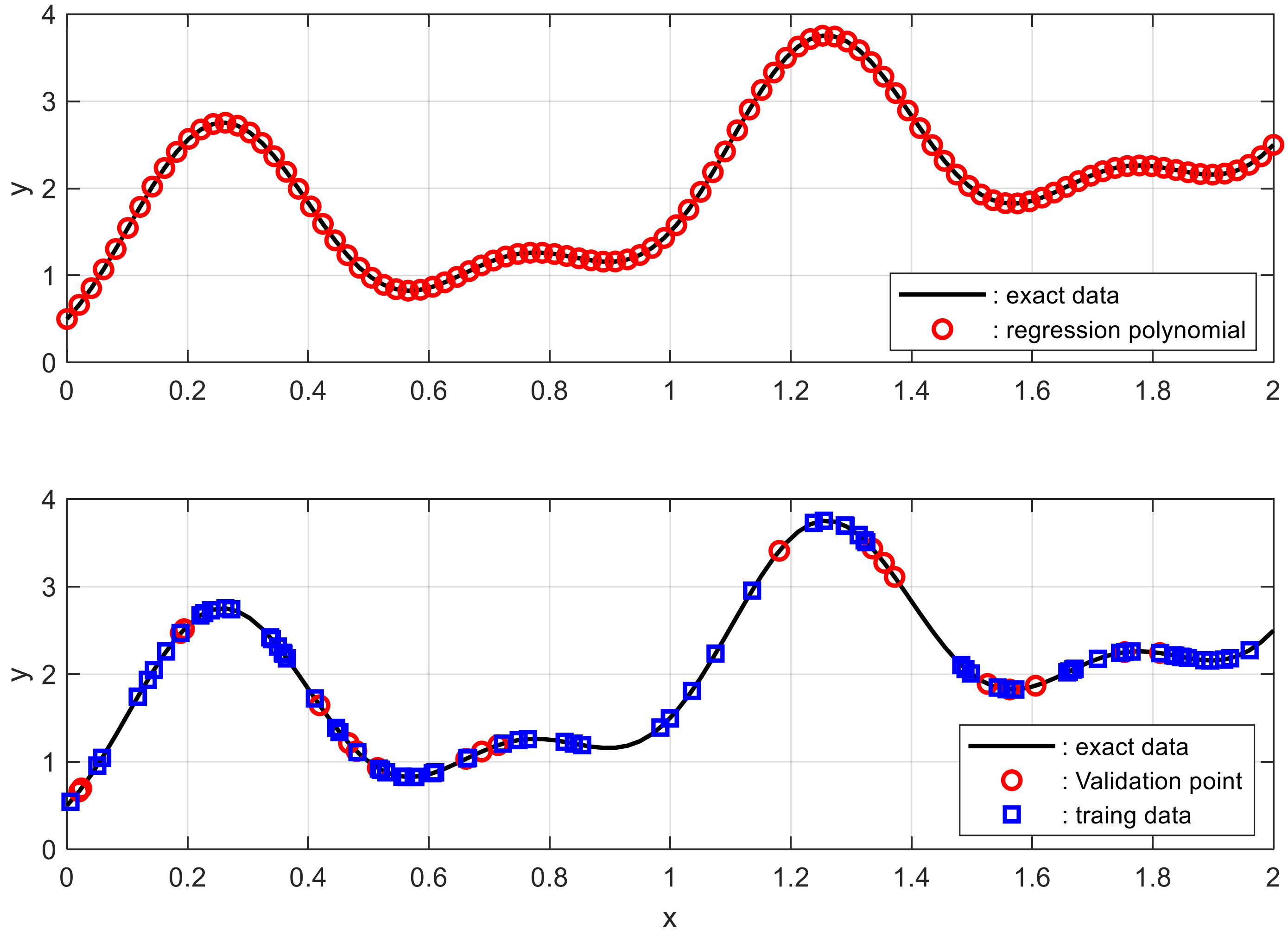
```
%---------------------------------------------------
% (4) Validation Using Exact Data
%---------------------------------------------------
   yer(1:N,1) = 0.0 ;
   for j=1:N
      x1 = xe(j,1)        ;
      x2= 2.0*pi*x1   ;
      x4= 4.0*pi*x1   ;

      Vec(1,1) = 1.0      ;
      Vec(2,1) = x1       ;
      Vec(3,1) = x1*x1    ;
      Vec(4,1) = cos(x2) ;
      Vec(5,1) = sin(x2) ;
      Vec(6,1) = cos(x4) ;
      Vec(7,1) = sin(x4) ;

      yer(j,1) = A_vec'*Vec(1:7,1) ;
   end
```

❑ Program Structures: A1_LSR_General_Function_Main.m

```
%-----------------------------------------------------------
% (5) Validation Using Validation Data
%-----------------------------------------------------------
   yvr(1:Nv,1) = 0.0  ;
   for j=1:Nv
      x1 = xv(j,1)        ;
      x2= 2.0*pi*x1    ;
      x4= 4.0*pi*x1    ;

      Vec(1,1) = 1.0      ;
      Vec(2,1) = x1       ;
      Vec(3,1) = x1*x1    ;
      Vec(4,1) = cos(x2) ;
      Vec(5,1) = sin(x2) ;
      Vec(6,1) = cos(x4) ;
      Vec(7,1) = sin(x4) ;

      yvr(j,1) = A_vec'*Vec(1:7,1) ;
   end
%-----------------------------------------------------------
```

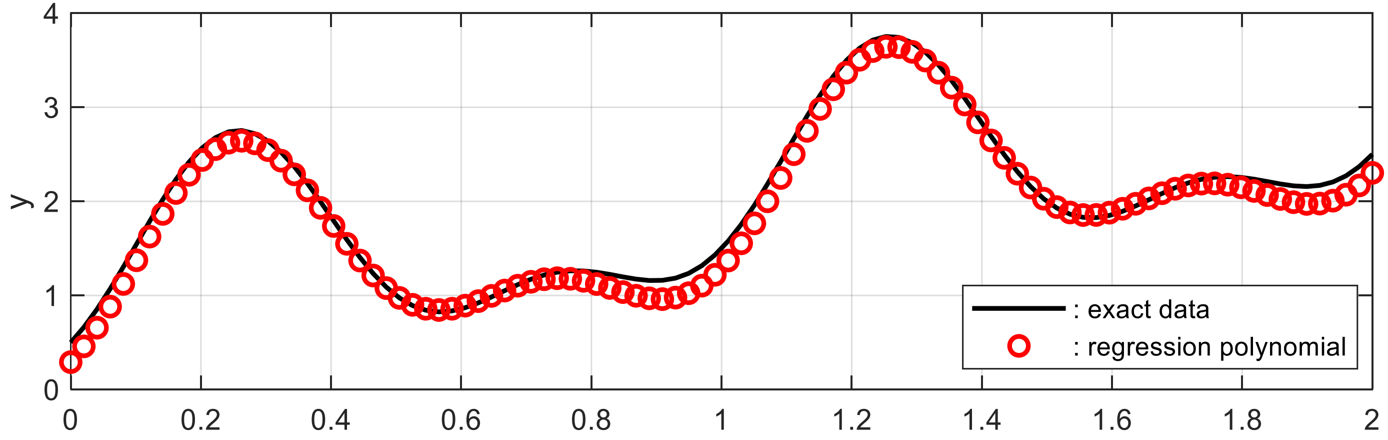
Results of Polynomial Regression

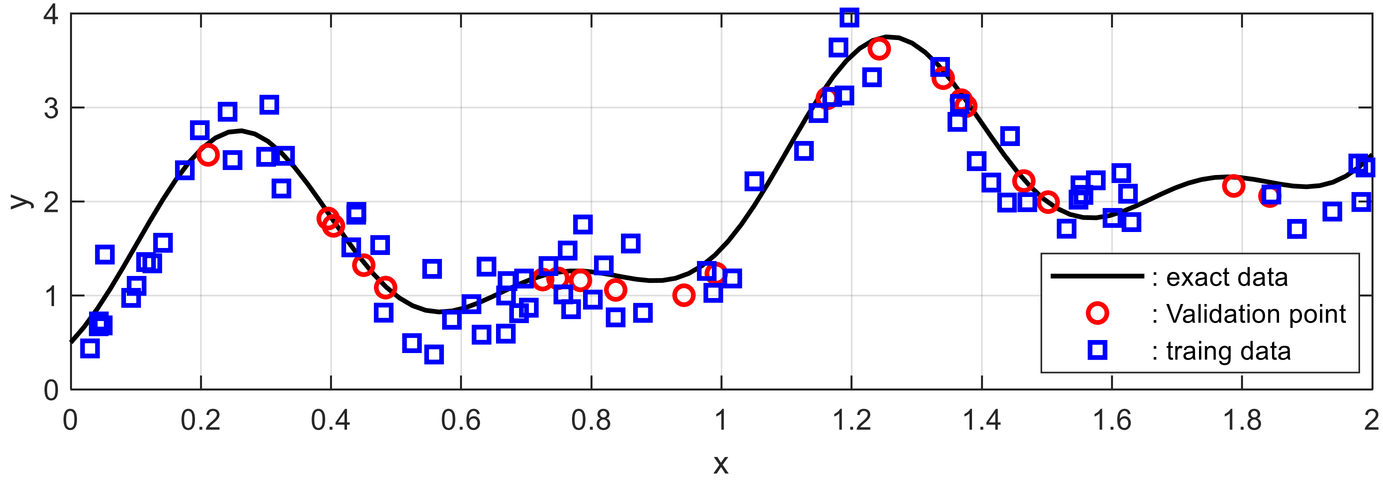
N= 100 ; alpa = 0.0



Results of Polynomial Regression

N= 100 ; alpa = 0.5





목 차

1. Curve-fitting Technique General
2. Least-Square Regression Analysis with Polynomials
3. Least-Square Regression Analysis with General Function
4. Polynomial Interpolation: Naïve Approach
5. Newton's Polynomial Interpolation
6. Lagrange Polynomial Interpolation
7. Cubic Spline Interpolation

Polynomial Interpolation: Naïve Approach

❑ Data and Approximation Function

Data $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_j, y_j), \cdots, (x_n, y_n)\} = \{(x_j, y_j)\}_{j=1}^{j=n}$

Polynomial Interpolating Function

$$y = f(x; \mathbf{a}) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$= \begin{pmatrix} 1 & x & x^2 & x^3 & x^4 & \cdots & x^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \boldsymbol{\varphi}^T(x)\mathbf{a}, \quad \boldsymbol{\varphi}(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^n \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

$$\boxed{y = f(x; \mathbf{a}) = \boldsymbol{\varphi}^T(x)\mathbf{a}}$$

$\boldsymbol{\varphi}(x)$ **: Basis functions for Interpolation**

$\mathbf{a}$ **: Interpolating coefficients**

Solution of Coefficients

$$\boxed{\mathbf{e} = \mathbf{y} - \begin{pmatrix} \boldsymbol{\varphi}^T(x_1) \\ \boldsymbol{\varphi}^T(x_2) \\ \vdots \\ \boldsymbol{\varphi}^T(x_K) \end{pmatrix} \mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{a}}$$

$$\mathbf{e} = 0 \rightarrow \quad \boxed{\mathbf{a} = \mathbf{X}^{-1}\mathbf{y}}$$

Polynomial Interpolation: Naïve Approach

❑ Program Structures

A1_Polynomial_Interpolation_Naive_Main	**: Main program**
Data_Generation_for_Interpolation	**: Data Generator**
LU_Backward_substitution	**: Back Substitution**
LU_decomposition	**: LU-Decomposition**
LU_Forward_substitution	**: Forward Substitution**

Polynomial Interpolation: Naïve Approach

❑ Program Structures: A1_Polynomial_Interpolation_Naive_Main.m

```
%----------------------------------------
%  Data Generation
%    Input:
%          N      = number of total data
%    Output
%          Xin(N,1) : independent variable x  (sequential order)
%          Yin(N,1) : Interpolated function
%----------------------------------------
%   Polynomial Regression
%----------------------------------------
% (1) Number of data
%----------------------------------------
   clear all ; close all;
%
   Nd = 5  ; Nv= 60 ; % Number of Data for Interpolation and Validation
%----------------------------------------
% (2) Generation of Total Data
%----------------------------------------
   N=Nd ; Data_Generation_for_Interpolation    ;
   Xdata(1:N,1) = xe(1:N,1) ;
   Ydata(1:N,1) = ye(1:N,1) ;
%----------------------------------------
```

❑ Program Structures: A1_Polynomial_Interpolation_Naive_Main.m

```
%----------------------------------------------------------
% (3) Build Matrix A consisting of Regression Function  Using Training Data
%----------------------------------------------------------
% (3-1) Interpolating function
%----------------------------------------------------------
   AA(1:N,1:N) = 0.0 ;
   for j=1:N
      xx = Xdata(j,1)    ;
      AA(j,1) = 1.0      ;
      x1 = 1.0           ;
      for k=2:N ;      x1 = x1*xx ;          AA(j,k) = x1;        end
   end
%----------------------------------------------------------
% (3-2) Output function
%----------------------------------------------------------
   B_vec(1:N,1) = Ydata(1:N,1) ; %
%----------------------------------------------------------
% (3-3) Interpolation Coefficients using LU-decomposition
%----------------------------------------------------------
   [AL_mat, AU_mat] = LU_decomposition(AA);
   [Y_vec] = LU_Forward_substitution (AL_mat, B_vec) ; % Forward Substitution    (Ly=b)
   [A_vec] = LU_Backward_substitution(AU_mat, Y_vec) ; % Regression Coefficients (Ua=y)
%----------------------------------------------------------
%   Please, don't use  A_vec = inv(AA)*B_vec;
%----------------------------------------------------------
```

Polynomial Interpolation: Naïve Approach

❑ Program Structures: A1_Polynomial_Interpolation_Naive_Main.m

```
%-----------------------------------------
% (4) Validation Using Exact Data
%-----------------------------------------
   xe=[];
   ye=[];
%
   N=Nv ; Data_Generation_for_Interpolation    ;
   Xv(1:N,1) = xe(1:N,1) ;
   Yv(1:N,1) = ye(1:N,1) ;
%
   Xin(1:Nv,1) = Xv(1:Nv,1) ;
   Yin(1:Nv,1) = 0.0 ;
   for j=1:Nv
      xx = Xv(j,1)   ;
      Yin(j,1) = A_vec(1) ;
      x1 = 1.0       ;
      for k=2:Nd
         x1 = x1*xx ;
         Yin(j,1) = Yin(j,1) + A_vec(k)*x1 ;
      end
   end
%-----------------------------------------
```

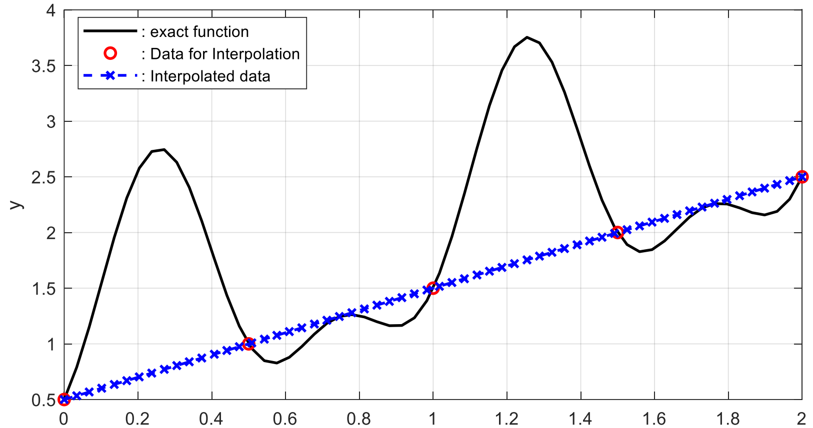
❑ Program Structures: A1_Polynomial_Interpolation_Naive_Main.m

```
%-----------------------------------------
% (5) Plot Results
%-----------------------------------------
LW=1.5; L1 = 1.5 ;
%-----------------------------------------
figure(6);set(gcf,'DefaultLineLineWidth',LW);set(gca,'DefaultLineLineWidth',LW)
                  p1= plot(Xv(:,1),Yv(:,1),'-k')  ;grid on; hold on ; ylabel('y') ;
                  p2= plot(Xdata(:,1),Ydata(:,1),'ro') ;
                  p3= plot(Xin(:,1),Yin(:,1),'--xb') ;
                 legend([p1,p2,p3],': exact function',': Data for Interpolation',': Interpolated data')
%-----------------------------------------
```

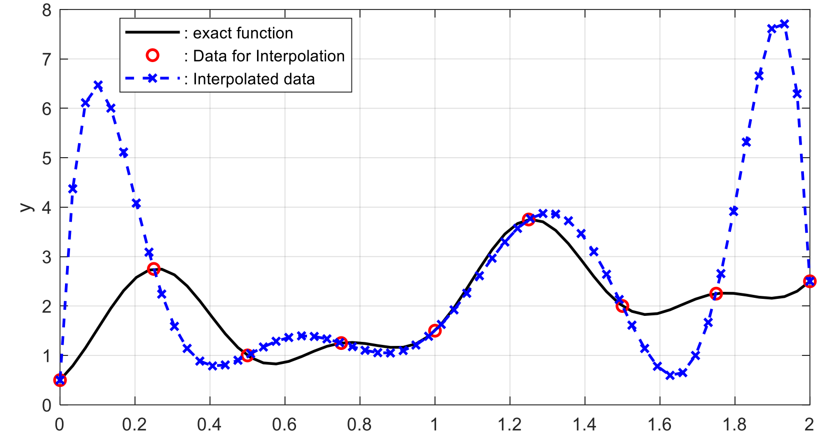
Polynomial Interpolation: Naïve Approach

❑ Results of Naïve Polynomial Interpolation

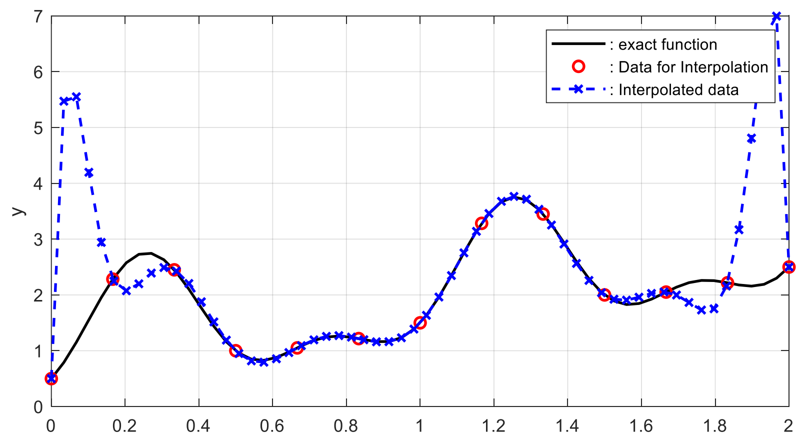
Nd= 5 ; Nv = 60



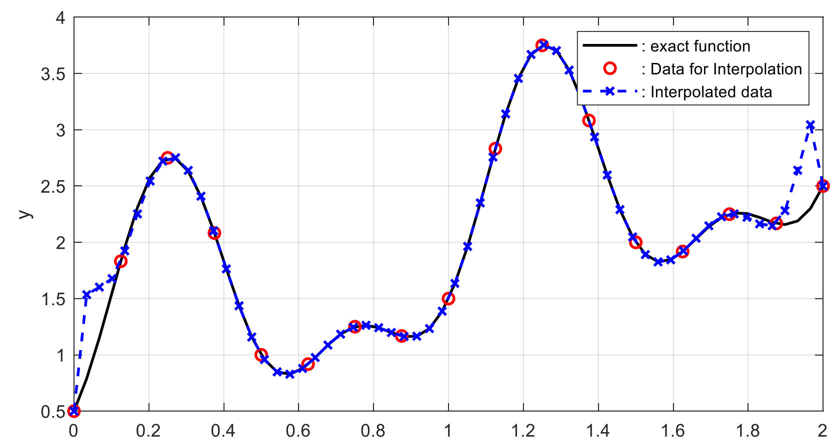
Nd= 9 ; Nv = 60



Nd= 13 ; Nv = 60



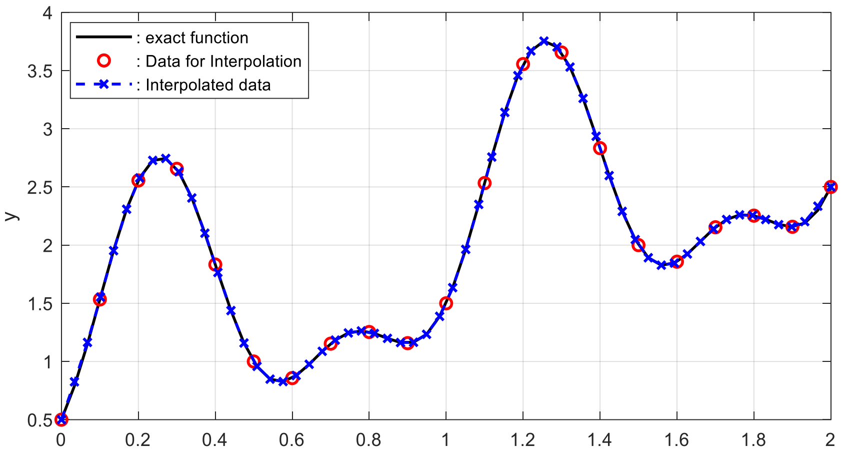
Nd= 17 ; Nv = 60



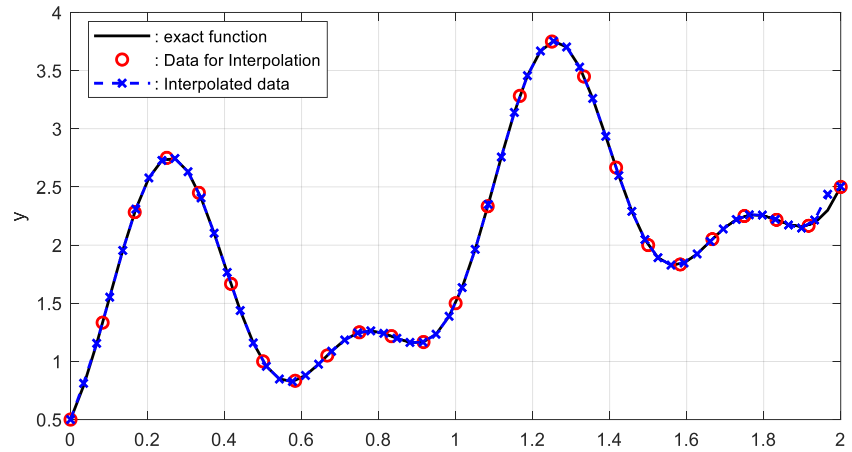
Polynomial Interpolation: Naïve Approach

❑ Results of Naïve Polynomial Interpolation

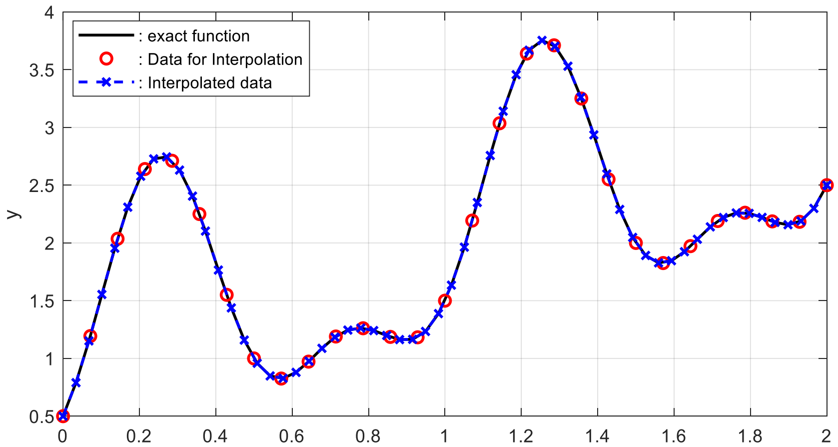
Nd= 21 ; Nv = 60



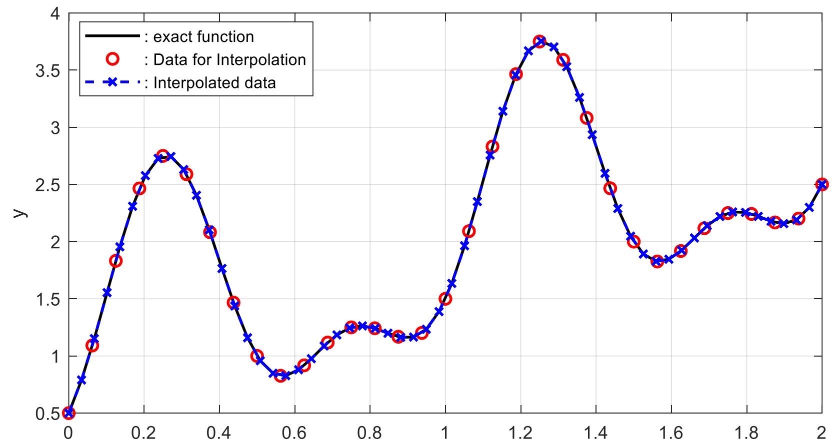
Nd= 25 ; Nv = 60



Nd= 29 ; Nv = 60



Nd= 33 ; Nv = 60



목 차

Polynomial Interpolation: Newton's Polynomial Interpolation

❑ Data and Approximation Function

Data $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_j, y_j), \cdots, (x_n, y_n)\} = \{(x_j, y_j)\}_{j=1}^{j=n}$

Polynomial Interpolating Function

$$
\begin{aligned}
y &= f(x; \mathbf{a}) \\
&= a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2) + a_3(x - x_1)(x - x_2)(x - x_3) \\
&\quad + \cdots + a_n(x - x_1)(x - x_2)(x - x_3)\cdots(x - x_{n-1})(x - x_n)
\end{aligned}
$$

Solution

$$
\begin{aligned}
(x_1, y_1) &\rightarrow y_1 = a_0 \\
(x_2, y_2) &\rightarrow y_2 = a_0 + a_1(x_2 - x_1) \\
(x_3, y_3) &\rightarrow y_3 = a_0 + a_1(x_3 - x_1) + a_2(x_3 - x_1)(x_3 - x_2) \\
&\vdots \\
(x_j, y_j) &\rightarrow y_j = a_0 + a_1(x_j - x_1) + \cdots + a_{j-1}(x_j - x_1)(x_j - x_2)(x_j - x_3)\cdots(x_j - x_{j-1}) \\
(x_{j+1}, y_{j+1}) &\rightarrow y_{j+1} = a_0 + a_1(x_{j+1} - x_1) + \cdots + a_j(x_{j+1} - x_1)(x_{j+1} - x_2)(x_{j+1} - x_3)\cdots(x_{j+1} - x_j) \\
&\vdots
\end{aligned}
$$

Data and Approximation Function

Derivation of the Coefficient Solution in a Compact form

$$a_j = \frac{y_{j+1} - \left\{a_0 + a_1(x_{j+1}-x_1) + \cdots + a_{j-1}(x_{j+1}-x_1)(x_{j+1}-x_2)(x_{j+1}-x_3)\cdots(x_{j+1}-x_{j-1})\right\}}{(x_{j+1}-x_1)(x_{j+1}-x_2)(x_{j+1}-x_3)\cdots(x_{j+1}-x_j)}$$

$$= \frac{y_{j+1} - a_0 - \sum_{k=1}^{k=j-1} a_k \prod_{l=1}^{l=k}(x_{j+1}-x_l)}{\prod_{l=1}^{l=j}(x_{j+1}-x_l)}$$

$$\boxed{a_j = \frac{y_{j+1} - a_0 - \sum_{k=1}^{k=j-1} a_k P(x_{j+1},k)}{P(x_{j+1},j)} \qquad P(x,k) = \prod_{l=1}^{l=k}(x-x_l)}$$

Program Structures

- A1_Polynomial_Interpolation_Newton_Main
- Data_Generation_for_Interpolation
- Product_funtion

: **Computing** $P(x,k)=\prod_{l=1}^{l=k}(x-x_l)$

```
function [Pfun_vec] = Product_funtion(x,xvec,j)
%
    Pfun_vec(1,1) = x - xvec(1,1)          ;
    for k=2:j
        km = k-1                           ;
        dx   = x - xvec(k,1)               ;
        Pfun_vec(k,1) = Pfun_vec(km,1)*dx  ;
    end
%
end
```

❑ Program Structures: A1_Polynomial_Interpolation_Newton_Main.m

```
%---------------------------------------
%  Data Generation
%    Input:
%          N       = number of total data
%    Output
%          Xin(N,1) : independent variable x  (sequential order)
%          Yin(N,1) : Interpolated function
%---------------------------------------
%   Polynomial Regression
%---------------------------------------
% (1) Number of data
%---------------------------------------
   clear all ; close all;
%
   Nd = 21  ; Nv= 60 ; % Number of Data for Interpolation and Validation
%---------------------------------------
% (2) Generation of Data
%---------------------------------------
   N=Nd ; Data_Generation_for_Interpolation    ;
   Xdata(1:N,1) = xe(1:N,1) ;
   Ydata(1:N,1) = ye(1:N,1) ;
%---------------------------------------
```

❑ Program Structures: A1_Polynomial_Interpolation_Newton_Main.m

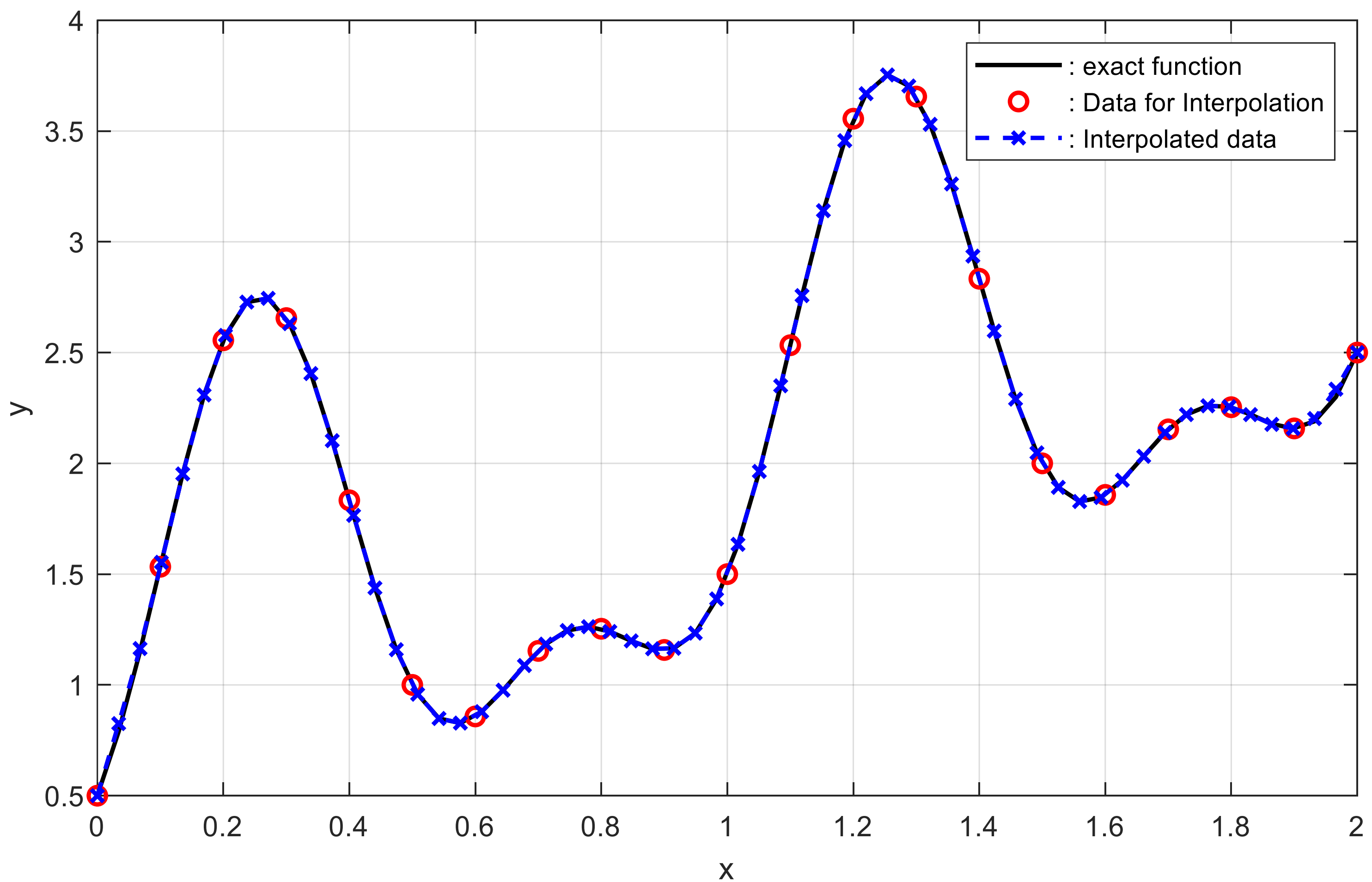
```
%-----------------------------------------
% (3) Coefficients for Newton Polynomial Interpolation
%-----------------------------------------
   A_vec(1,1) = Ydata(1,1)                                    ; % (x1,y1,a1)
   A_vec(2,1) =(Ydata(2,1) - A_vec(1,1))/(Xdata(2,1)-Xdata(1,1)) ; % (x2,y2,a2)
   for j=3:N
      jm= j-1   ;
%
      xx = Xdata(j,1)                        ; % (xj,yj,aj)
      [Pfun_vec] = Product_funtion(xx,Xdata,jm) ; % (xj,yj,aj)
%
      sum = A_vec(1,1)  ;
      for k = 2:jm
         km=k-1;
         sum = sum + A_vec(k,1)*Pfun_vec(km)    ;
      end
%
      A_vec(j,1) =(Ydata(j,1) - sum)/Pfun_vec(jm)  ;
   end
%-----------------------------------------
```

❑ Program Structures: A1_Polynomial_Interpolation_Newton_Main.m

```
%----------------------------------------
% (4) Validation Using Exact Data
%----------------------------------------
   xe=[];
   ye=[];
%
   N=Nv ; Data_Generation_for_Interpolation    ;
   Xv(1:N,1) = xe(1:N,1) ;
   Yv(1:N,1) = ye(1:N,1) ;
%
   Xin(1:Nv,1) = Xv(1:Nv,1) ;
   Yin(1:Nv,1) = 0.0 ;
   for j=1:Nv
      xx = Xv(j,1)          ;
      [Pfun_vec] = Product_funtion(xx,Xdata,jm)               ;
      Yin(j,1)    = A_vec(1,1) + A_vec(2,1)*(xx - Xdata(1,1))     ;
      for k=3:Nd
         km = k - 1                                     ;
         Yin(j,1) = Yin(j,1) + A_vec(k,1)*Pfun_vec(km,1)         ;
      end
   end
%----------------------------------------
% (5) Plot Results: The same as before
%----------------------------------------
```

❑ Results of Newton’s Polynomial Interpolation

Nd= 21 ; Nv = 60



목 차

Lagrange Polynomial Interpolation

❑ Polynomial Interpolation (PI) is Unique: All PI are same.

Data $D=\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\cdots,(x_j,y_j),\cdots,(x_n,y_n)\}=\{(x_j,y_j)\}_{j=1}^{j=n}$

Newton Interpolation

$$\begin{aligned} y &= f(x;\mathbf{a}) \\ &= a_0+a_1(x-x_1)+a_2(x-x_1)(x-x_2)+a_3(x-x_1)(x-x_2)(x-x_3) \\ &\quad +\cdots+a_n(x-x_1)(x-x_2)(x-x_3)\cdots(x-x_{n-1})(x-x_n) \\ &= a_0+a_1g_1(x)+a_2g_2(x)+a_3g_3(x)+\cdots+a_ng_n(x) \end{aligned}$$

$$\boxed{\begin{aligned} y &= f(x;\mathbf{a}) \\ &= \mathbf{g}^T(x)\mathbf{a} \\ &= \mathbf{a}^T\mathbf{g}(x) \end{aligned}} \qquad \boxed{\mathbf{g}(x)=\begin{pmatrix} 1 \\ g_1(x) \\ g_2(x) \\ \vdots \\ g_n(x) \end{pmatrix}}$$

Solution

$$\begin{aligned} (x_1,y_1) &\to y_1=a_0+a_1g_1(x_1)+a_2g_2(x_1)+a_3g_3(x_1)+\cdots+a_ng_n(x_1) \\ (x_2,y_2) &\to y_2=a_0+a_1g_1(x_2)+a_2g_2(x_2)+a_3g_3(x_2)+\cdots+a_ng_n(x_2) \\ &\vdots \\ (x_n,y_n) &\to y_n=a_0+a_1g_1(x_n)+a_2g_2(x_n)+a_3g_3(x_n)+\cdots+a_ng_n(x_n) \end{aligned}$$

$$\boxed{\to \begin{pmatrix} 1 & g_1(x_1) & g_2(x_1) & \cdots & g_n(x_1) \\ 1 & g_1(x_2) & g_2(x_2) & \cdots & g_n(x_2) \\ 1 & g_1(x_3) & g_2(x_3) & \cdots & g_n(x_3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & g_1(x_n) & g_2(x_n) & \cdots & g_n(x_n) \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}=\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}}$$

$$\boxed{\begin{pmatrix} \mathbf{g}^T(x_1) \\ \mathbf{g}^T(x_2) \\ \mathbf{g}^T(x_3) \\ \vdots \\ \mathbf{g}^T(x_n) \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}=\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} \to \left(\begin{aligned} \mathbf{G}^T\mathbf{a} &= \mathbf{y} \\ \mathbf{a} &= \mathbf{G}^{-T}\mathbf{y} \\ y &= \mathbf{a}^T\mathbf{g}(x) \\ &= \mathbf{y}^T\mathbf{G}^{-1}\mathbf{g}(x) \\ &= \mathbf{y}^T\mathbf{L}(x) \\ &= \mathbf{L}^T(x)\mathbf{y} \end{aligned}\right)}$$

$$\boxed{y=\mathbf{L}^T(x)\mathbf{y}=L_1(x)y_1+L_2(x)y_2+L_3(x)y_3+\cdots+L_4(x)y_4=\sum_{j=1}^{j=n}L_j(x)y_j}$$

The result represent the same form as the Lagrange Interpolation with

$$\boxed{\begin{aligned} & L_j(x_k)=\delta_{j,k} \qquad (j,k=1,2,\cdots,n) \\ & L_j(x)=\prod_{\substack{k=1 \\ k\neq j}}^{k=n}\left(\frac{x-x_k}{x_j-x_k}\right) \end{aligned}}$$

❑ Data Generation for Curve-fitting Test (I) : Generating Function

Generating function

$$y = g(x) = x + \sin(2\pi x) - 0.5\cos(4\pi x) + 1, \quad (0 \le x \le 2)$$

$$\frac{dy}{dx} = g'(x) = 1 + 2\pi\cos(2\pi x) + 2\pi\sin(4\pi x)$$

$$\frac{d^2y}{dx^2} = g''(x) = -4\pi^2\sin(2\pi x) + 8\pi^2\cos(4\pi x)$$

Lagrange Interpolation

$$y = y_1L_1(x) + y_2L_2(x) + y_3L_3(x) + \cdots + y_nL_n(x) = \sum_{j=1}^{j=n} y_jL_j(x)$$

$$L_j(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_{n-1})(x-x_n)}{(x_j-x_1)(x_j-x_2)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_{n-1})(x_j-x_n)} = \prod_{\substack{k=1 \\ k\neq j}}^{k=n}\left(\frac{x-x_k}{x_j-x_k}\right)$$

$$L_j(x_k) = \delta_{j,k} \quad (j,k = 1,2,\cdots,n)$$

Data Generation for Curve-fitting Test (I) : Generating Function

Lagrange approximation of function derivatives

$$y = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x) + \cdots + y_n L_n(x) = \sum_{j=1}^{j=n} y_j L_j(x)$$

$$\frac{dy}{dx} = \sum_{j=1}^{j=n} y_j \frac{dL_j(x)}{dx} = \sum_{j=1}^{j=n} y_j L'_j(x)$$

$$\frac{d^2 y}{dx^2} = \sum_{j=1}^{j=n} y_j \frac{d^2 L_j(x)}{dx^2} = \sum_{j=1}^{j=n} y_j L''_j(x)$$

$$\text{or} \quad \frac{d^2 y}{dx^2} = \sum_{j=1}^{j=n} y'_j \frac{dL_j(x)}{dx} = \sum_{j=1}^{j=n} y'_j L'_j(x) \leftarrow y'_j = \sum_{k=1}^{k=n} y_k L'_k(x_j)$$

$$L_j(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_{n-1})(x-x_n)}{(x_j-x_1)(x_j-x_2)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_{n-1})(x_j-x_n)}$$

$$L'_j(x) = \frac{L_j(x)}{(x-x_1)} + \frac{L_j(x)}{(x-x_2)} + \cdots + \frac{L_j(x)}{(x-x_{j-1})} + \frac{L_j(x)}{(x-x_{j+1})} + \cdots + \frac{L_j(x)}{(x-x_n)} = \sum_{\substack{k=1 \\ k \neq j}}^{k=n} \frac{L_j(x)}{(x-x_k)}$$

Lagrange Polynomial Interpolation

❑ Program Structures

- A1_Polynomial_Interpolation_Lagrange_Main : Program Main for Interpolation
- A2_Polynomial_Interpolation_Lagrange_Derivative_Main : Program Main for Derivative Estimation
- Data_Generation_for_Interpolation
- Lagrange_funtion : Lagrange Polynomial
- Lagrange_funtion_derivative : Derivative of Lagrange Polynomial

$$L_j(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_{n-1})(x-x_n)}{(x_j-x_1)(x_j-x_2)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_{n-1})(x_j-x_n)} = \frac{P(x)}{P(x_j)}$$

$$L_j'(x) = \frac{(x-x_2)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_{n-1})(x-x_n)}{P(x_j)} + \frac{(x-x_1)(x-x_3)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_{n-1})(x-x_n)}{P(x_j)} + \cdots$$

$$L_j'(x) = \frac{L_j(x)}{(x-x_1)} + \frac{L_j(x)}{(x-x_2)} + \cdots + \frac{L_j(x)}{(x-x_{j-1})} + \frac{L_j(x)}{(x-x_{j+1})} + \cdots + \frac{L_j(x)}{(x-x_n)} = \sum_{\substack{k=1 \\ k\neq j}}^{k=n} \frac{L_j(x)}{(x-x_k)}$$

Lagrange Polynomial Interpolation

Program Structures: Lagrange_funtion.m

```
%---------------------------------------
function [FunLag] = Lagrange_funtion(x,N,XN)
%---------------------------------------
% Input
%    x        : evaluation point     %    N        : Number of nodes  %    XN(1:N)  : Nodes
% Output
%    FunLag(1:N)  : Lagrange function values at x
%---------------------------------------
% (1) Initialize
%---------------------------------------
   FunLag(1:N,1) = 1.0    ;
%---------------------------------------
% (2) Loop
%---------------------------------------
   for j = 1: N
      Fun = 1.0 ;
      for k=1:j-1
         Fun = Fun*(x - XN(k) )/( XN(j) - XN(k) ) ;
      end
      for k=j+1:N
         Fun = Fun*(x - XN(k) )/( XN(j) - XN(k) ) ;
      end
      FunLag(j,1) = Fun ;
   end
%---------------------------------------
```

❑ Program Structures: Lagrange_funtion_derivative.m

```
%-----------------------------------------
function [dot_FunLag] = Lagrange_funtion_derivative(x,N,XN)
%-----------------------------------------
% (1) Initialize
%-----------------------------------------
  [FunLag] = Lagrange_funtion(x,N,XN) ;
  dot_FunLag(N,1) = 0.0             ;
%-----------------------------------------
% (1) Finding the zero Node
%-----------------------------------------
   K0 = 0 ;
   for k = 1:N
      DX = abs( x - XN(k) ) ;
      if ( abs( DX ) < 1.0E-15 )
         K0 = k ;            break  ;
      end
   end
%-----------------------------------------
% (2) Case when K0=0
%-----------------------------------------
   if ( K0 == 0 )
%-----------------------------------------
      for j = 1: N
         FL             = FunLag(j,1) ;
         for k=1:j-1 ;        dot_FunLag(j,1) = dot_FunLag(j,1) + FL/( x - XN(k) ) ;         end
         for k=j+1:N ;        dot_FunLag(j,1) = dot_FunLag(j,1) + FL/( x - XN(k) ) ;         end
      end
%-----------------------------------------
```

❑ Program Structures: Lagrange_funtion_derivative.m

```
    else
%-------------------------------------
      for j = 1: N
%-------------------------------------
         FL             = FunLag(j,1) ;
         for k=1:j-1
            if( k ~= K0 )
               dot_FunLag(j,1) = dot_FunLag(j,1) + FL/( x - XN(k) ) ;
            end
         end
%
         for k=j+1:N
            if( k ~= K0 )
               dot_FunLag(j,1) = dot_FunLag(j,1) + FL/( x - XN(k) ) ;
            end
         end
%-------------------------------
          Xnew(1:N-2) = 0.0 ;
          m= 0          ;
          for k=1:N
             if( k~= j && k~=K0 )
                m=m+1   ;
                Xnew(m) = XN(k)     ;
             end
          end
%-------------------------------
```

```
%-------------------------------------
          FL = 1.0 ;
          for k=1:m
             FL = FL*( x - Xnew(k) )/( XN(j) - Xnew(k) ) ;
          end
%
         if (j~= K0)
            dot_FunLag(j,1) = dot_FunLag(j,1) + FL/( XN(j) - XN(K0) ) ;
         end
%-------------------------------------
      end
%-------------------------------------
   end
%-------------------------------------
```

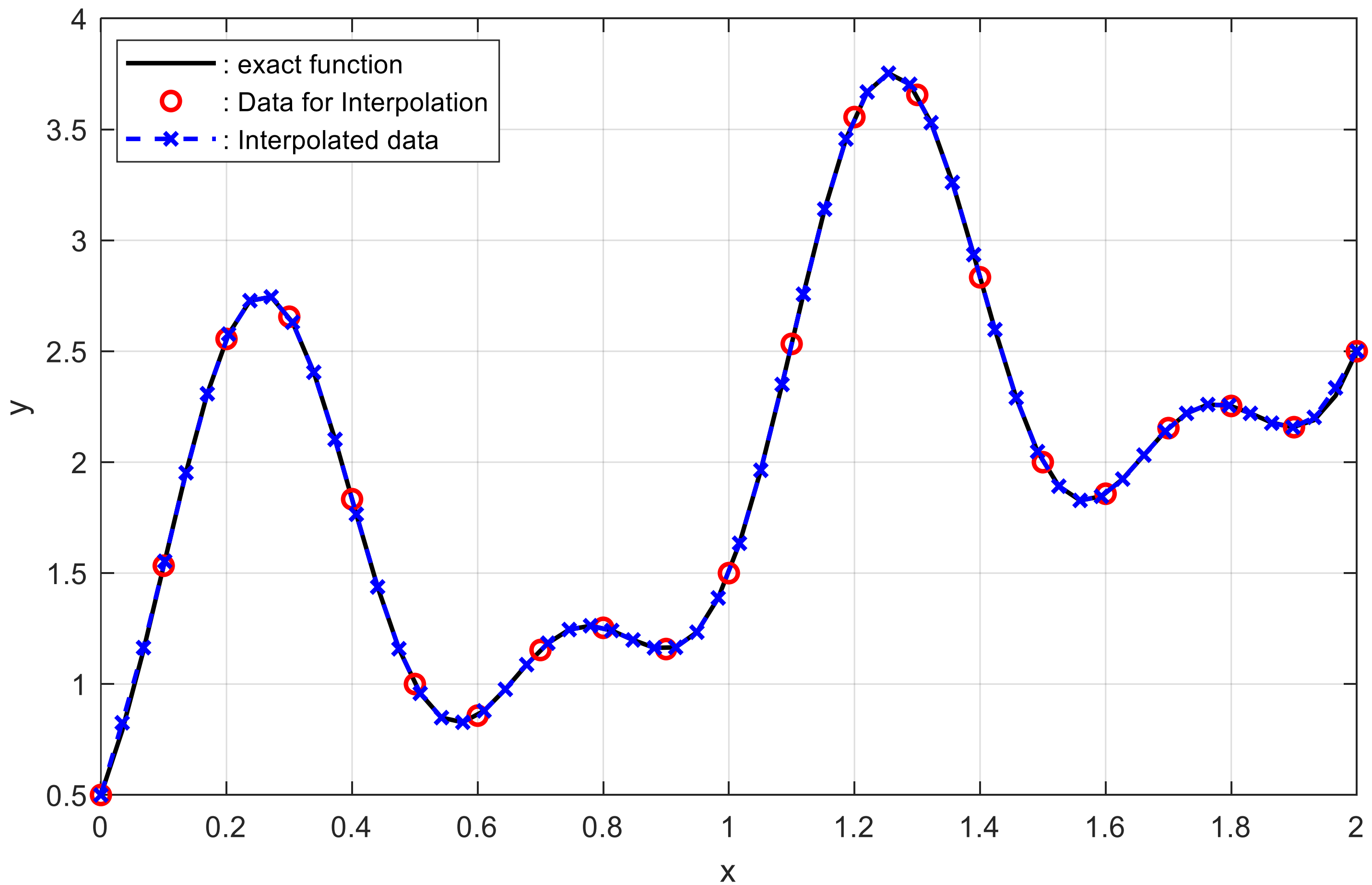
Lagrange Polynomial Interpolation

Program Structures: Lagrange_funtion_derivative.m

```
%----------------------------------------
%  (1) Initialize
%----------------------------------------
xe(1:N,1)  = 0.0;        ye(1:N,1)  = 0.0;
dye(1:N,1)  = 0.0;      ddye(1:N,1)  = 0.0;
xn(1:N,1)  = 0.0;       yn(1:N,1)  = 0.0;
%----------------------------------------
%  (2) Exact Data (sequential order)
%----------------------------------------
dx = 2.0/(N-1)           ;
for j=1:N
    x1=dx*(j-1)          ;
%
    xe(j,1) = x1          ;
    ye(j,1) = x1 + sin(2*pi*x1) - 0.5*cos(4*pi*x1) + 1.0      ;
end
%----------------------------------------
%  (3) Exact derivative (sequential order)
%----------------------------------------
dx = 2.0/(N-1)           ;
for j=1:N
    x1=dx*(j-1)          ;
%
    xe(j,1)  = x1          ;
    dye(j,1) = 1.0 + 2.0*pi*cos(2*pi*x1) + 2.0*pi*sin(4*pi*x1)      ;
    ddye(j,1) = -4.0*pi*pi*sin(2*pi*x1)  + 8.0*pi*pi*cos(4*pi*x1)   ;
end
%----------------------------------------
```

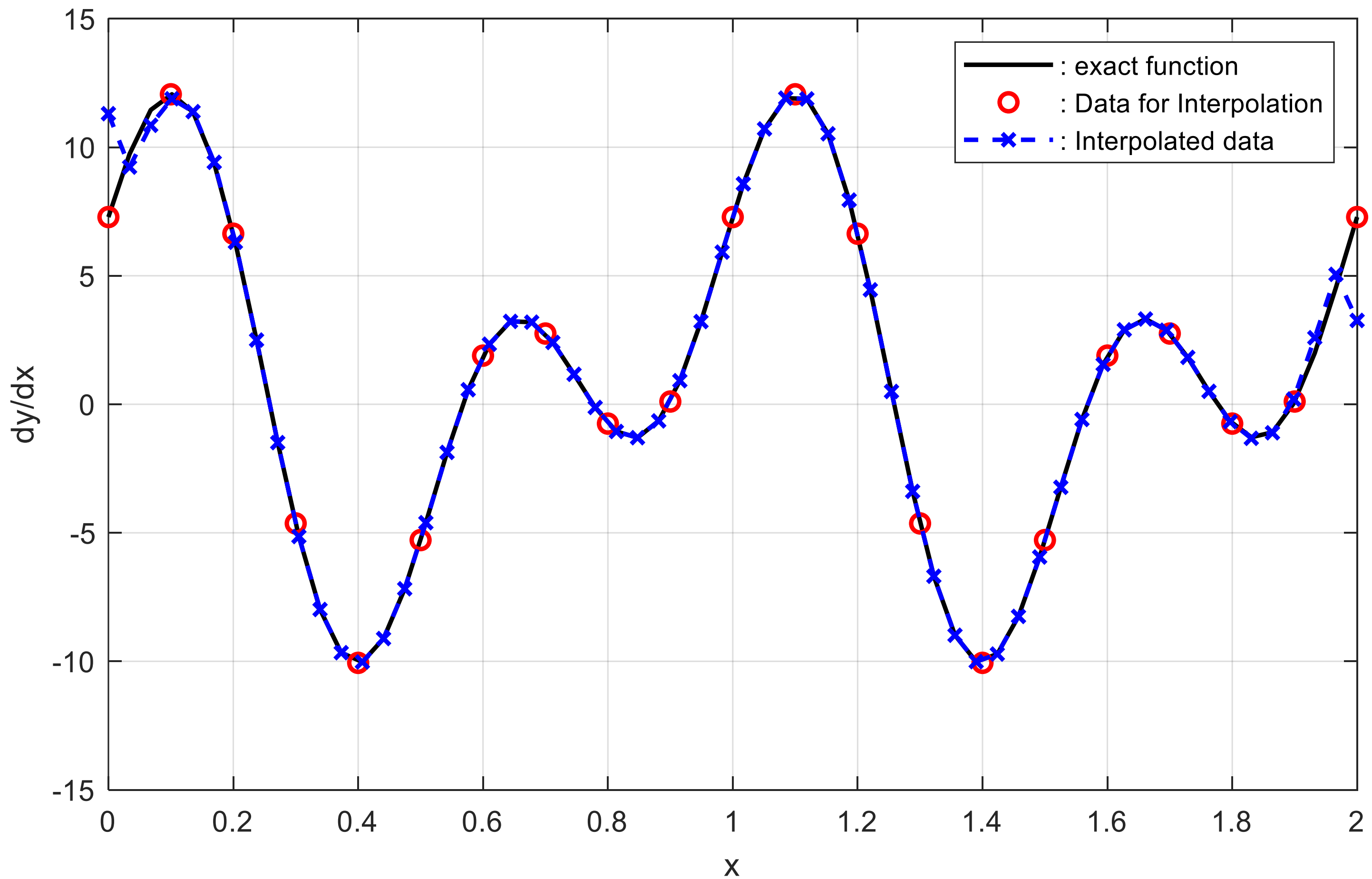
Results of Lagrange Polynomial Interpolation: Function Values

Nd= 21 ; Nv = 60



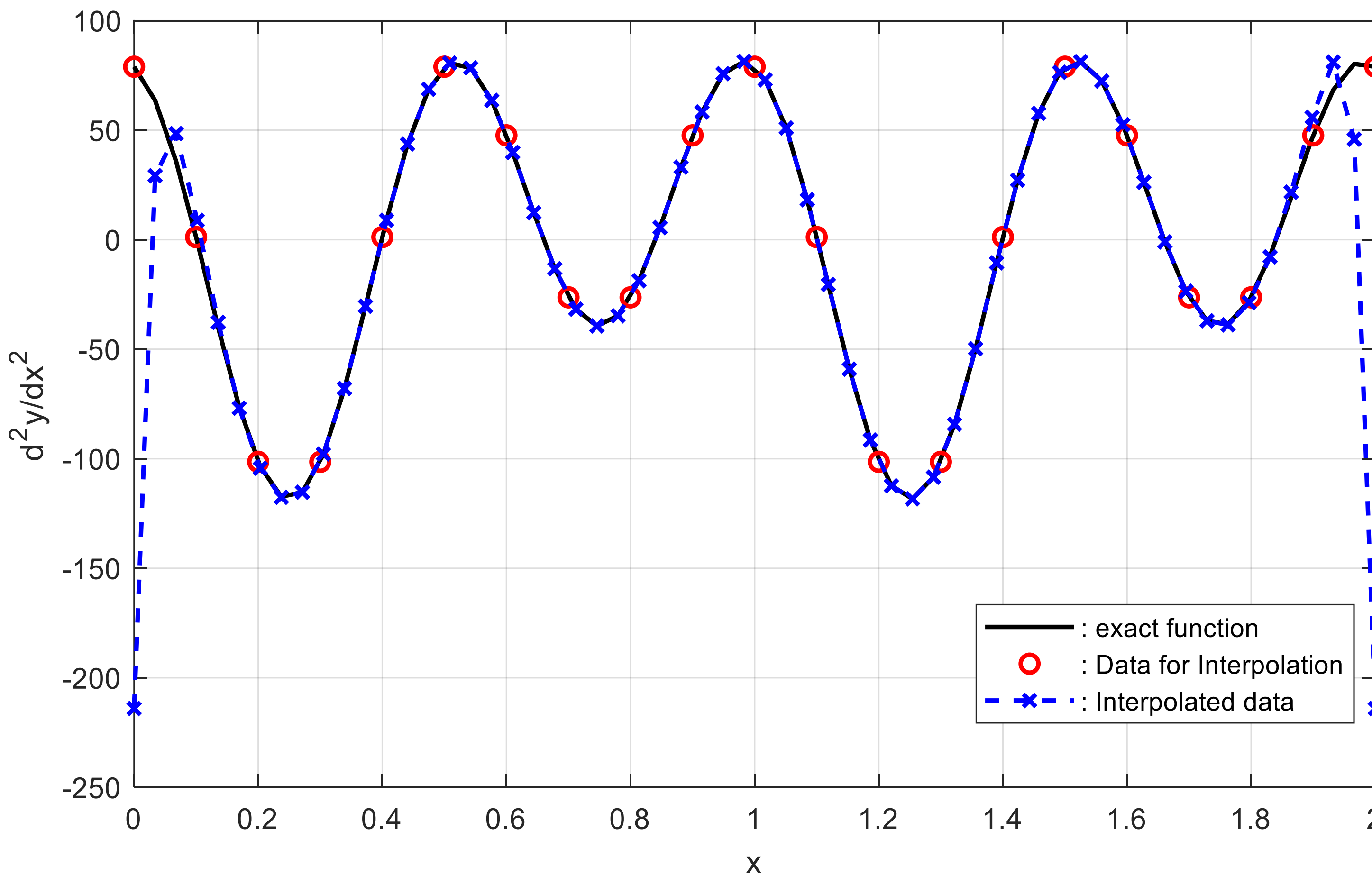
Results of Lagrange Polynomial Interpolation: First Derivative

Nd= 21 ; Nv = 60



Results of Lagrange Polynomial Interpolation: Second Derivative

Nd= 21 ; Nv = 60



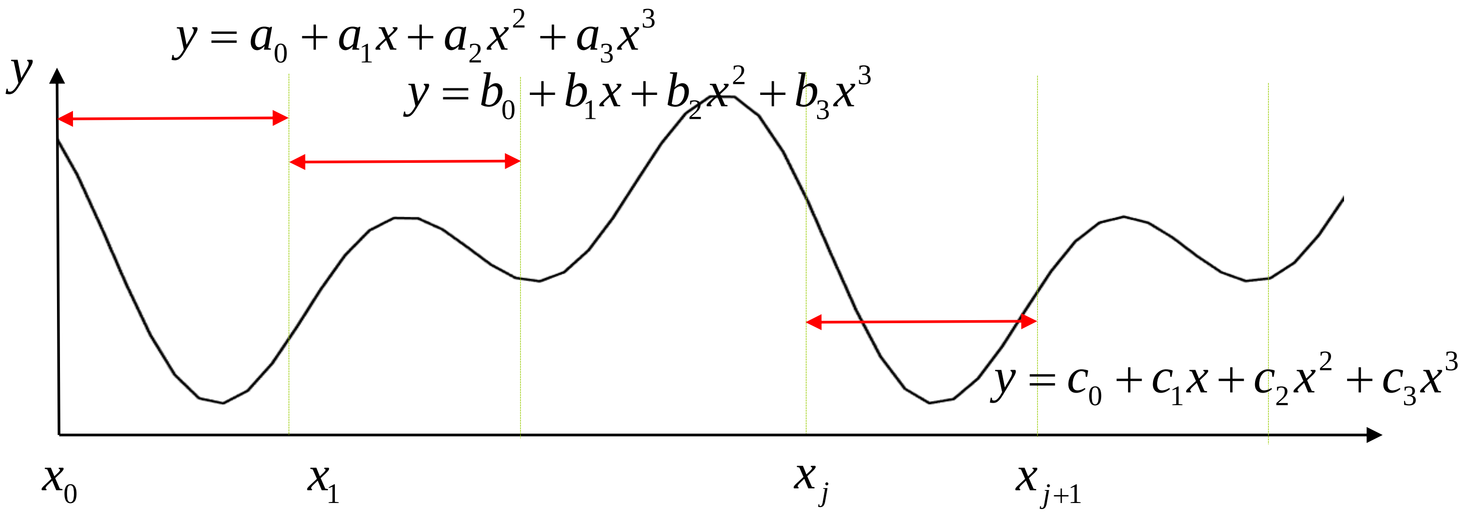
목 차

Cubic Spline Interpolation

❑ Formulation using Non-dimensional Local Variable

Data $D=\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\cdots,(x_j,y_j),\cdots,(x_n,y_n)\}=\{(x_j,y_j)\}_{j=1}^{j=n}$

Local Variable



$$\tau=\frac{x-x_j}{x_{j+1}-x_j}=\frac{x-x_j}{\Delta x_j}\in[0,1] \leftarrow \begin{pmatrix} x_j\le x\le x_{j+1} \\ \Delta x_j = x_{j+1}-x_j \end{pmatrix}$$

$$y=\alpha_{j0}+\alpha_{j1}\tau+\alpha_{j2}\tau^2+\alpha_{j3}\tau^3$$

$$\frac{dy}{dx}=\frac{d\tau}{dx}\frac{dy}{d\tau}=\frac{1}{\Delta x_j}\left(\alpha_{j1}+2\alpha_{j2}\tau+3\alpha_{j3}\tau^2\right)$$

$$\frac{d^2y}{dx^2}=\frac{1}{\Delta x_j^2}\left(2\alpha_{j2}+6\alpha_{j3}\tau\right)$$

$$\frac{d^3y}{dx^3}=\frac{6\alpha_{j3}}{\Delta x_j^3}$$

Cubic Spline Interpolation

❑ Formulation using Non-dimensional Local Variable

Build Linear Algebraic Equations $y = \alpha_{j0} + \alpha_{j1}\tau + \alpha_{j2}\tau^2 + \alpha_{j3}\tau^3$ $\boxed{4(n-1)}$

(a) Function Values

$$x = x_j, \quad (j = 1 \sim n-1) \qquad y_j = \alpha_{j0} \leftarrow \tau = 0$$

$$x = x_n \qquad y_n = \alpha_{n-1,0} + \alpha_{n-1,1} + \alpha_{n-1,2} + \alpha_{n-1,3} \leftarrow \tau = 1$$

$\boxed{n}$

(b) Continuity in Function Values

$$x = x_{j+1}, \quad (j = 1 \sim n-2) \qquad y_{j+1} = \alpha_{j0} + \alpha_{j1} + \alpha_{j2} + \alpha_{j3} \leftarrow \tau = 1$$

$\boxed{n-2}$

(c) Continuity in the first derivatives $\boxed{\frac{dy}{dx} = \frac{d\tau}{dx}\frac{dy}{d\tau} = \frac{1}{\Delta x_j}\left(\alpha_{j1} + 2\alpha_{j2}\tau + 3\alpha_{j3}\tau^2\right)}$

$$x = x_{j+1}, \quad (j = 1 \sim n-2) \qquad \frac{1}{\Delta x_j}\left(\alpha_{j1} + 2\alpha_{j2} + 3\alpha_{j3}\right) = \frac{1}{\Delta x_{j+1}}\alpha_{j+1,1}$$

$\boxed{n-2}$

(d) Continuity in the second derivatives $\boxed{\frac{d^2y}{dx^2} = \frac{1}{\Delta x_j^2}\left(2\alpha_{j2} + 6\alpha_{j3}\tau\right)}$

$$x = x_{j+1}, \quad (j = 1 \sim n-2) \qquad \frac{1}{\Delta x_j^2}\left(2\alpha_{j2} + 6\alpha_{j3}\right) = \frac{2}{\Delta x_{j+1}^2}\alpha_{j+1,2}$$

$\boxed{n-2}$

(d) Zero in the third derivative at two end points and the second point

$$\alpha_{13} = \alpha_{n-1,3} = 0$$

$\boxed{2}$

Cubic Spline Interpolation

❑ Formulation using Non-dimensional Local Variable

$$
\left(\begin{array}{cccccccccccccccccccccccccccccccccccc}
1&0&0&0&0&0&0&0&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
1&1&1&1&0&0&0&0&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
0&1&2&3&0&-\beta_1&0&0&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
0&0&2&6&0&0&-2\beta_1^2&0&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
0&0&0&1&0&0&0&0&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&1&0&0&0&0&0&0&0&&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&1&1&1&1&0&0&0&0&&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&0&1&2&3&0&-\beta_2&0&0&&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&0&0&2&6&0&0&-2\beta_2^2&0&&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&\ddots&&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&&\ddots&&&&&&&&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&&1&0&0&0&0&0&0&0&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&&1&1&1&1&1&0&0&0&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&&0&1&2&3&0&-\beta_j&0&0&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&&0&0&2&6&0&0&-2\beta_j^2&0&&&&&&&&&&&&&&&&\\
&&&&&&&&&&&&&&&&1&0&0&0&0&0&0&0&&&&&&&&&&&&\\
&&&&&&&&&&&&&&&&1&1&1&1&0&0&0&0&&&&&&&&&&&&\\
&&&&&&&&&&&&&&&&0&1&2&3&0&-\beta_{j+1}&0&0&&&&&&&&&&&&\\
&&&&&&&&&&&&&&&&&0&2&6&0&0&-2\beta_{j+1}^2&0&&&&&&&&&&&&\\
&&&&&&&&&&&&&&&&&&&&&&&\ddots&&&&&&&&&&&&\\
&&&&&&&&&&&&&&&&&&&&&&&&1&0&0&0&0&0&0&0&&&&\\
&&&&&&&&&&&&&&&&&&&&&&&&1&1&1&1&0&0&0&0&&&&\\
&&&&&&&&&&&&&&&&&&&&&&&&0&1&2&3&0&-\beta_{n-1}&0&0&&&&\\
&&&&&&&&&&&&&&&&&&&&&&&&0&0&2&6&0&0&-2\beta_{n-2}^2&0&&&&\\
&&&&&&&&&&&&&&&&&&&&&&&&&&&&1&0&0&0&0&0&0&0\\
&&&&&&&&&&&&&&&&&&&&&&&&&&&&1&1&1&1&0&0&0&0\\
&&&&&&&&&&&&&&&&&&&&&&&&&&&&0&1&2&3&0&-\beta_{n-2}&0&0\\
&&&&&&&&&&&&&&&&&&&&&&&&&&&&0&0&2&6&0&0&-2\beta_{n-2}^2&0\\
&&&&&&&&&&&&&&&&&&&&&&&&&&&&0&0&0&0&1&1&1&1\\
&&&&&&&&&&&&&&&&&&&&&&&&&&&&0&0&0&0&0&0&0&1
\end{array}\right)
\begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \vdots \\ \boldsymbol{\alpha}_j \\ \boldsymbol{\alpha}_{j+1} \\ \vdots \\ \boldsymbol{\alpha}_{n-1} \end{pmatrix}
=
\begin{pmatrix} y_1 \\ y_2 \\ 0 \\ 0 \\ 0 \\ y_2 \\ y_3 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ y_j \\ y_{j+1} \\ 0 \\ 0 \\ y_{j+1} \\ y_{j+2} \\ 0 \\ 0 \\ \vdots \\ y_{n-3} \\ y_{n-2} \\ 0 \\ 0 \\ y_{n-2} \\ y_{n-1} \\ 0 \\ 0 \\ y_n \\ 0 \end{pmatrix}
\quad \leftarrow \beta_j = \frac{\Delta x}{\Delta x_j}
$$

5

4

4

4

4

4

2

Formulation using Non-dimensional Local Variable

$$
\boxed{\begin{aligned} \alpha_{1,0} &= y_1 \\ \alpha_{2,0} &= y_2 \\ \alpha_{3,0} &= y_3 \\ &\vdots \\ \alpha_{n-1,0} &= y_{n-1} \end{aligned}}
\qquad
\boxed{\begin{aligned} \alpha_{1,3} &= 0 \\ \alpha_{n-1,3} &= 0 \end{aligned}}
$$

$$
\left(\begin{array}{ccccccccccccccccccccccccccc}
0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 2 & 3 & -\beta_1 & 0 & 0 \\
0 & 2 & 6 & 0 & -\gamma_1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 2 & 3 & -\beta_2 & 0 & 0 \\
 & & & 0 & 2 & 6 & 0 & -\gamma_2 & 0 \\
 & & & & & & 1 & 1 & 1 & 0 & 0 & 0 \\
 & & & & & & 1 & 2 & 3 & -\beta_3 & 0 & 0 \\
 & & & & & & 0 & 2 & 6 & 0 & -\gamma_3 & 0 \\
 & & & & & & & & & & \ddots \\
 & & & & & & & & & 1 & 1 & 1 & 0 & 0 & 0 \\
 & & & & & & & & & 1 & 2 & 3 & -\beta_j & 0 & 0 \\
 & & & & & & & & & 0 & 2 & 6 & 0 & -\gamma_j & 0 \\
 & & & & & & & & & & & & 1 & 1 & 1 & 0 & 0 & 0 \\
 & & & & & & & & & & & & 1 & 2 & 3 & -\beta_{j+1} & 0 & 0 \\
 & & & & & & & & & & & & 0 & 2 & 6 & 0 & -\gamma_{j+1} & 0 \\
 & & & & & & & & & & & & & & & & & & \ddots \\
 & & & & & & & & & & & & & & & & & \ddots \\
 & & & & & & & & & & & & & & & & & 1 & 1 & 1 & 0 & 0 & 0 \\
 & & & & & & & & & & & & & & & & & 1 & 2 & 3 & -\beta_{n-4} & 0 & 0 \\
 & & & & & & & & & & & & & & & & & 0 & 2 & 6 & 0 & -\gamma_{n-4} & 0 \\
 & & & & & & & & & & & & & & & & & & & & 1 & 1 & 1 & 0 & 0 & 0 \\
 & & & & & & & & & & & & & & & & & & & & 1 & 2 & 3 & -\beta_{n-3} & 0 & 0 \\
 & & & & & & & & & & & & & & & & & & & & & 2 & 6 & 0 & -\gamma_{n-3} & 0 \\
 & & & & & & & & & & & & & & & & & & & & & & & 1 & 1 & 1 & 0 \\
 & & & & & & & & & & & & & & & & & & & & & & & 1 & 2 & 3 & -\beta_{n-2} & 0 & 0 \\
 & & & & & & & & & & & & & & & & & & & & & & & 0 & 2 & 6 & 0 & -\gamma_{n-2} & 0 \\
 & & & & & & & & & & & & & & & & & & & & & & & 0 & 0 & 0 & 1 & 1 & 1 \\
 & & & & & & & & & & & & & & & & & & & & & & & & & & 0 & 0 & 1
\end{array}\right)
\begin{pmatrix}
\alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{2,1} \\ \alpha_{2,2} \\ \alpha_{2,3} \\ \alpha_{3,1} \\ \alpha_{3,2} \\ \alpha_{3,3} \\ \alpha_{4,1} \\ \alpha_{4,2} \\ \alpha_{4,3} \\ \vdots \\ \alpha_{j,1} \\ \alpha_{j,2} \\ \alpha_{j,3} \\ \alpha_{j+1,1} \\ \alpha_{j+1,2} \\ \alpha_{j+1,3} \\ \vdots \\ \\ \vdots \\ \alpha_{n-3,1} \\ \alpha_{n-3,2} \\ \alpha_{n-3,3} \\ \alpha_{n-2,1} \\ \alpha_{n-2,2} \\ \alpha_{n-2,3} \\ \alpha_{n-1,1} \\ \alpha_{n-1,2} \\ \alpha_{n-1,3}
\end{pmatrix}
=
\begin{pmatrix}
0 \\ y_2 - y_1 \\ 0 \\ 0 \\ y_3 - y_2 \\ 0 \\ 0 \\ y_4 - y_3 \\ 0 \\ 0 \\ \vdots \\ y_{j+1} - y_j \\ 0 \\ 0 \\ y_{j+2} - y_{j+1} \\ 0 \\ 0 \\ \vdots \\ \\ \vdots \\ y_{n-3} - y_{n-4} \\ 0 \\ 0 \\ y_{n-2} - y_{n-3} \\ 0 \\ 0 \\ y_{n-1} - y_{n-2} \\ 0 \\ 0 \\ y_n - y_{n-1} \\ 0
\end{pmatrix}
\leftarrow
\begin{pmatrix}
\beta_j = \dfrac{\Delta x_j}{\Delta x_{j+1}} \\
\gamma_j = 2\beta_j^2
\end{pmatrix}
$$

❑ Formulation using Non-dimensional Local Variable

$$
\begin{array}{l}
\alpha_{1,0} = y_1 \\
\alpha_{2,0} = y_2 \\
\alpha_{3,0} = y_3 \\
\vdots \\
\alpha_{n-1,0} = y_{n-1}
\end{array}
\qquad
\begin{array}{l}
\alpha_{1,3} = 0 \\
\alpha_{n-1,3} = 0
\end{array}
$$

$$
\left(\begin{array}{ccccccccccccc}
0\;0\;1 & 0\;0\;0 & & & & & & & & & & & \\
1\;1\;1 & 0\;0\;0 & & & & & & & & & & & \\
1\;2\;3 & -\beta_1\;0\;0 & & & & & & & & & & & \\
0\;2\;6 & 0\;-\gamma_1\;0 & & & & & & & & & & & \\
 & 1\;1\;1 & 0\;0\;0 & & & & & & & & & & \\
 & 1\;2\;3 & -\beta_2\;0\;0 & & & & & & & & & & \\
 & 0\;2\;6 & 0\;-\gamma_2\;0 & & & & & & & & & & \\
 & & 1\;1\;1 & 0\;0\;0 & & & & & & & & & \\
 & & 1\;2\;3 & -\beta_3\;0\;0 & & & & & & & & & \\
 & & 0\;2\;6 & 0\;-\gamma_3\;0 & & & & & & & & & \\
 & & & & \ddots & & & & & & & & \\
 & & & & & 1\;1\;1 & 0\;0\;0 & & & & & & \\
 & & & & & 1\;2\;3 & -\beta_j\;0\;0 & & & & & & \\
 & & & & & 0\;2\;6 & 0\;-\gamma_j\;0 & & & & & & \\
 & & & & & & 1\;1\;1 & 0\;0\;0 & & & & & \\
 & & & & & & 1\;2\;3 & -\beta_{j+1}\;0\;0 & & & & & \\
 & & & & & & 0\;2\;6 & 0\;-\gamma_{j+1}\;0 & & & & & \\
 & & & & & & & & \ddots & & & & \\
 & & & & & & & & & 1\;1\;1 & 0\;0\;0 & & \\
 & & & & & & & & & 1\;2\;3 & -\beta_{n-4}\;0\;0 & & \\
 & & & & & & & & & 0\;2\;6 & 0\;-\gamma_{n-4}\;0 & & \\
 & & & & & & & & & & 1\;1\;1 & 0\;0\;0 & \\
 & & & & & & & & & & 1\;2\;3 & -\beta_{n-3}\;0\;0 & \\
 & & & & & & & & & & \;\;2\;6 & 0\;-\gamma_{n-3}\;0 & \\
 & & & & & & & & & & & 1\;1\;1 & 0 \\
 & & & & & & & & & & & 1\;2\;3 & -\beta_{n-2}\;0\;0 \\
 & & & & & & & & & & & 0\;2\;6 & 0\;-\gamma_{n-2}\;0 \\
 & & & & & & & & & & & 0\;0\;0 & 1\;1\;1 \\
 & & & & & & & & & & & & 0\;0\;1
\end{array}\right)
\left(\begin{array}{c}
\alpha_{1,1}\\ \alpha_{1,2}\\ \alpha_{1,3}\\ \alpha_{2,1}\\ \alpha_{2,2}\\ \alpha_{2,3}\\ \alpha_{3,1}\\ \alpha_{3,2}\\ \alpha_{3,3}\\ \alpha_{4,1}\\ \alpha_{4,2}\\ \alpha_{4,3}\\ \vdots\\ \alpha_{j,1}\\ \alpha_{j,2}\\ \alpha_{j,3}\\ \alpha_{j+1,1}\\ \alpha_{j+1,2}\\ \alpha_{j+1,3}\\ \vdots\\ \vdots\\ \alpha_{n-3,1}\\ \alpha_{n-3,2}\\ \alpha_{n-3,3}\\ \alpha_{n-2,1}\\ \alpha_{n-2,2}\\ \alpha_{n-2,3}\\ \alpha_{n-1,1}\\ \alpha_{n-1,2}\\ \alpha_{n-1,3}
\end{array}\right)
=
\left(\begin{array}{c}
0\\ y_2-y_1\\ 0\\ 0\\ y_3-y_2\\ 0\\ 0\\ y_4-y_3\\ 0\\ 0\\ \vdots\\ y_{j+1}-y_j\\ 0\\ 0\\ y_{j+2}-y_{j+1}\\ 0\\ 0\\ \vdots\\ \vdots\\ y_{n-3}-y_{n-4}\\ 0\\ 0\\ y_{n-2}-y_{n-3}\\ 0\\ 0\\ y_{n-1}-y_{n-2}\\ 0\\ 0\\ y_n-y_{n-1}\\ 0
\end{array}\right)
\leftarrow
\left(\begin{array}{l}
\beta_j = \dfrac{\Delta x_j}{\Delta x_{j+1}} \\
\gamma_j = 2\beta_j^2
\end{array}\right)
$$

❑ Formulation using Non-dimensional Local Variable

$$\begin{pmatrix} \mathbf{B}_1 & \mathbf{C}_1 & 0 & \cdots & 0 & 0 \\ \mathbf{A}_2 & \mathbf{B}_2 & \mathbf{C}_1 & \cdots & 0 & 0 \\ 0 & \mathbf{A}_3 & \mathbf{B}_3 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \mathbf{B}_{n-2} & \mathbf{C}_{n-2} \\ 0 & 0 & 0 & \cdots & \mathbf{A}_{n-1} & \mathbf{B}_{n-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \boldsymbol{\alpha}_3 \\ \boldsymbol{\alpha}_4 \\ \vdots \\ \boldsymbol{\alpha}_{n-1} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_4 \\ \vdots \\ \mathbf{b}_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{B}_1 & \mathbf{C}_1 & 0 & \cdots & 0 & 0 \\ 0 & \bar{\mathbf{B}}_2 & \bar{\mathbf{C}}_1 & \cdots & 0 & 0 \\ 0 & 0 & \bar{\mathbf{B}}_3 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \bar{\mathbf{B}}_{n-2} & \bar{\mathbf{C}}_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & \bar{\mathbf{B}}_{n-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \boldsymbol{\alpha}_3 \\ \boldsymbol{\alpha}_4 \\ \vdots \\ \boldsymbol{\alpha}_{n-1} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \bar{\mathbf{b}}_2 \\ \bar{\mathbf{b}}_3 \\ \bar{\mathbf{b}}_4 \\ \vdots \\ \bar{\mathbf{b}}_{n-1} \end{pmatrix}$$

$$\begin{aligned} &\mathbf{B}_1\boldsymbol{\alpha}_1 + \mathbf{C}_1\boldsymbol{\alpha}_2 = \mathbf{b}_1 \\ &\mathbf{A}_2\boldsymbol{\alpha}_1 + \mathbf{B}_2\boldsymbol{\alpha}_2 + \mathbf{C}_2\boldsymbol{\alpha}_3 = \mathbf{b}_2 \end{aligned}$$

$$\begin{aligned} &\mathbf{B}_1\boldsymbol{\alpha}_1 + \mathbf{C}_1\boldsymbol{\alpha}_2 = \mathbf{b}_1 \\ &\boldsymbol{\alpha}_1 + \mathbf{B}_1^{-1}\mathbf{C}_1\boldsymbol{\alpha}_2 = \mathbf{B}_1^{-1}\mathbf{b}_1 \\ &\mathbf{A}_2\boldsymbol{\alpha}_1 + \mathbf{A}_2\mathbf{B}_1^{-1}\mathbf{C}_1\boldsymbol{\alpha}_2 = \mathbf{A}_2\mathbf{B}_1^{-1}\mathbf{b}_1 \\ \\ &\mathbf{A}_2\boldsymbol{\alpha}_1 + \mathbf{B}_2\boldsymbol{\alpha}_2 + \mathbf{C}_2\boldsymbol{\alpha}_3 = \mathbf{b}_2 \\ &---------------------- \\ &\left(\mathbf{B}_2 - \mathbf{A}_2\mathbf{B}_1^{-1}\mathbf{C}_1\right)\boldsymbol{\alpha}_2 + \mathbf{C}_2\boldsymbol{\alpha}_3 = \mathbf{b}_2 - \mathbf{A}_2\mathbf{B}_1^{-1}\mathbf{b}_1 \\ &\bar{\mathbf{B}}_2\boldsymbol{\alpha}_2 + \mathbf{C}_2\boldsymbol{\alpha}_3 = \bar{\mathbf{b}}_2 \leftarrow \begin{pmatrix} \bar{\mathbf{B}}_2 = \mathbf{B}_2 - \mathbf{A}_2\mathbf{B}_1^{-1}\mathbf{C}_1 \\ \bar{\mathbf{b}}_2 = \mathbf{b}_2 - \mathbf{A}_2\mathbf{B}_1^{-1}\mathbf{b}_1 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} &\bar{\mathbf{B}}_{n-1}\boldsymbol{\alpha}_{n-1} = \bar{\mathbf{b}}_{n-1} \\ &\bar{\mathbf{B}}_{n-2}\boldsymbol{\alpha}_{n-2} = \bar{\mathbf{b}}_{n-2} - \bar{\mathbf{C}}_{n-2}\boldsymbol{\alpha}_{n-1} \\ &\qquad\vdots \\ &\bar{\mathbf{B}}_2\boldsymbol{\alpha}_2 = \bar{\mathbf{b}}_2 - \bar{\mathbf{C}}_2\boldsymbol{\alpha}_2 \\ &\mathbf{B}_1\boldsymbol{\alpha}_1 = \mathbf{b}_1 - \mathbf{C}_1\boldsymbol{\alpha}_2 \end{aligned}$$

Solve using LU-decomposition

$$\begin{aligned} &\mathbf{B}^{-1}\mathbf{C} = \mathbf{F}, \qquad \mathbf{B}^{-1}\mathbf{b} = \mathbf{d} \\ &\mathbf{BF} = \mathbf{C} \\ &\mathbf{LUF} = \mathbf{C} \\ &\mathbf{LU}\left(\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_j, \cdots\right) = \left(\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_j, \cdots\right) \\ &\mathbf{LUf}_j = \mathbf{c}_j \quad (j = 1, 2, 3, \cdots) \\ \\ &\mathbf{LUd} = \mathbf{b} \end{aligned}$$

Or Use LUD for A^{-1}

Cubic Spline Interpolation

❑ Program Structures

A1_Cubic_Spine_Interpolation_Main	: **Program Main for Spline Interpolation**
CubicSpline_Build_BlockMatrix_ABC	: **Building LAE to Compute Spline Coefficients**
CubicSpline_Build_RHS_Vector	
CubicSpline_Evaluate_Derivatives	: **Estimate Derivatives using Spline Coefficients**
CubicSpline_Evaluate_Function	: **Estimate Function Values using Spline Coefficients**
Data_Generation_for_Interpolation	
Interval_Finding	: **Find Interval containing the input x**
LAE_Block_TriDiagonal_Solver	: **Block Tri-Diagonal Matrix Solver**
LU_Backward_substitution	
LU_decomposition	: **LU-decomposition Solvers** → **Use when all Diagonal elements are Non Zeroes**
LU_Forward_substitution	
LU_Matrix_Inverse	

❑ Program Structures: CubicSpline_Build_BlockMatrix_ABC.m

```
%----------------------------------------
% Data
%   Input:      N       : Number of total nodes
%               XN(N,1) : Nodes
%   Output
%        Am(3,3,N-1) : Matrix Aj in Block Tri-diagonal Matrix
%        Bm(3,3,N-1) : Matrix Bj in Block Tri-diagonal Matrix
%        Cm(3,3,N-1) : Matrix Cj in Block Tri-diagonal Matrix
%----------------------------------------
function [Am,Bm,Cm] = CubicSpline_Build_BlockMatrix_ABC(N,XN)
%----------------------------------------
% (1) Initialize Matrix A, B, C and Compute DX(j), Beta, Gamma.
%----------------------------------------
  Am(1:3,1:3,1:N-1) = 0.0 ;  Bm(1:3,1:3,1:N-1) = 0.0 ;   Cm(1:3,1:3,1:N-1) = 0.0 ;
%
  Beta(1:N-2) = 0.0 ;      Gama(1:N-2) = 0.0 ;
%
  for j = 1:N-1 ;    jp = j + 1 ;        DX(j) = XN(jp,1) - XN(j,1) ;     end
  for j = 1:N-2 ;    jp = j + 1 ;        Beta(j) = DX(j)/DX(jp)      ;    Gama(j) = 2.0*Beta(j)^2 ;    end
%----------------------------------------
% (2) Matrix Build
%----------------------------------------
% (2-1) k=1
%----------------------------------------
  Bm(1,1,1) = 0.0 ;  Bm(1,2,1) = 0.0 ;  Bm(1,3,1) = 1.0 ;
  Bm(2,1,1) = 1.0 ;  Bm(2,2,1) = 1.0 ;  Bm(2,3,1) = 1.0 ;
  Bm(3,1,1) = 1.0 ;  Bm(3,2,1) = 2.0 ;  Bm(3,3,1) = 3.0 ;
%
  Cm(3,1,1) = -Beta(1) ;
```

Program Structures: CubicSpline_Build_BlockMatrix_ABC.m

```
%---------------------------------------
%  (2-2) k=2 ~ (N-2)
%---------------------------------------
   for k=2: (N-2)
      km = k-1 ;
      Am(1,2,k)= 2.0 ; Am(1,3,k) = 6.0      ;
%
      Bm(1,1,k)= 0.0 ; Bm(1,2,k)= -Gama(km) ; Bm(1,3,k) = 0.0 ;
      Bm(2,1,k)= 1.0 ; Bm(2,2,k)=  1.0      ; Bm(2,3,k) = 1.0 ;
      Bm(3,1,k)= 1.0 ; Bm(3,2,k)=  2.0      ; Bm(3,3,k) = 3.0 ;
%
      Cm(3,1,k) = -Beta(k) ;
   end
%---------------------------------------
%  (2-3) k=N-1
%---------------------------------------
   k = N-1 ; km = k-1 ;
      Am(1,2,k)= 2.0 ; Am(1,3,k)= 6.0       ;
%
      Bm(1,1,k)= 0.0 ; Bm(1,2,k)= -Gama(km) ; Bm(1,3,k) = 0.0 ;
      Bm(2,1,k)= 1.0 ; Bm(2,2,k)=  1.0      ; Bm(2,3,k) = 1.0 ;
      Bm(3,1,k)= 0.0 ; Bm(3,2,k)=  0.0      ; Bm(3,3,k) = 1.0 ;
%---------------------------------------
end
%---------------------------------------
```

❑ Program Structures: Interval_Finding.m

```
%-----------------------------------------
function [K] = Interval_Finding(Xp,N,XN)
%-----------------------------------------
%  (1) Lower and Upper Limit
%-----------------------------------------
   NL = 1  ; NU = N  ;
%-----------------------------------------
%  (2) Finding Interval containing Xp
%-----------------------------------------
   for k=1:N
      Nm = int16( (NL + NU)/2 )   ;
      Xm = XN(Nm,1)               ;
      if( Xp <= Xm )
         NU = Nm  ;
      else
         NL = Nm  ;
      end
%
      if( NU-NL <=1 )
         break
      end
   end
   K = NL ;
%-----------------------------------------
end
%-----------------------------------------
```

❑ Program Structures: CubicSpline_Evaluate_Function.m

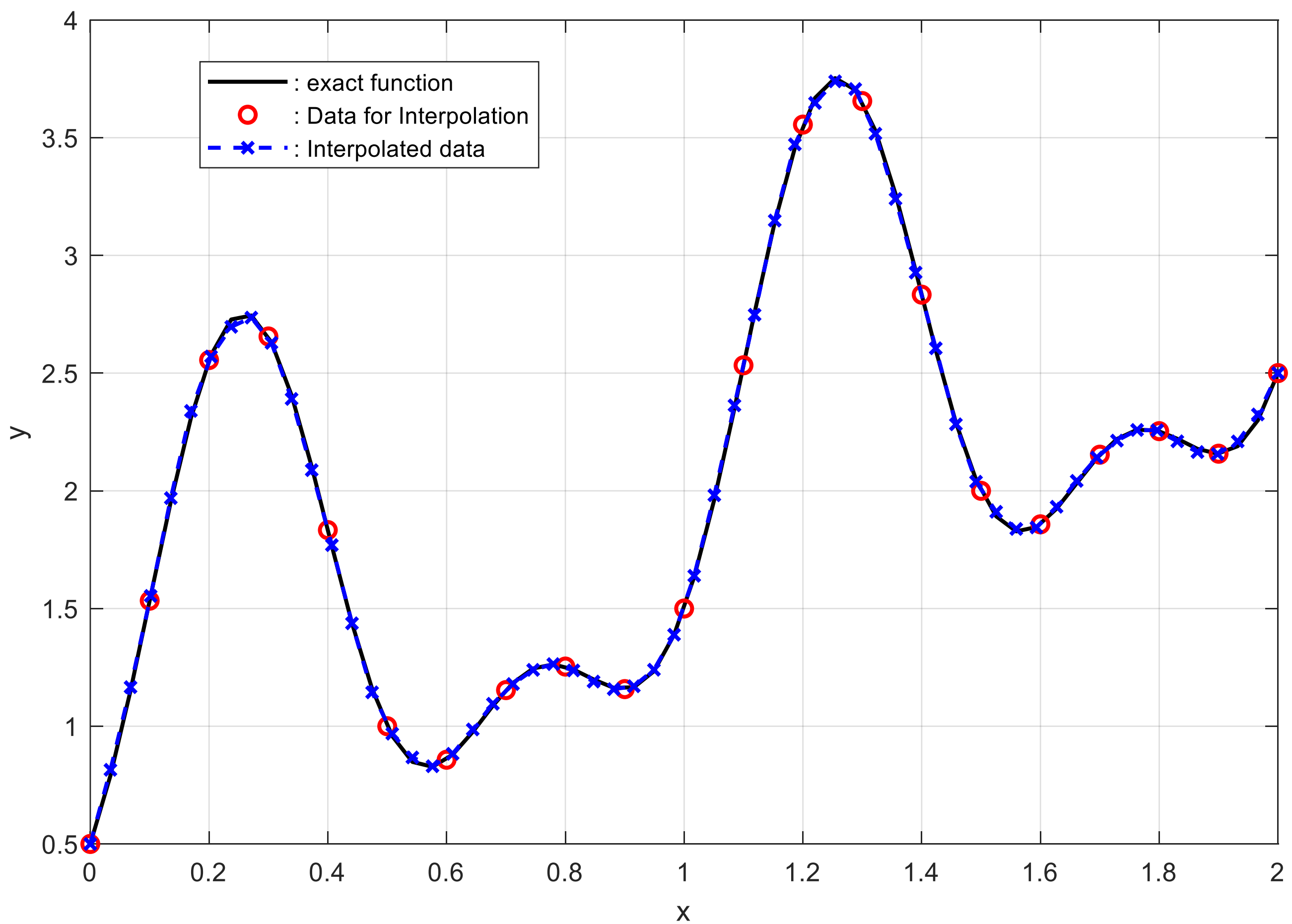
```
%-----------------------------------------
function [Yp] = CubicSpline_Evaluate_Function(Xp,N,XN,Sc)
%-----------------------------------------
%  (1) Finding Interval and Spline Coefficients
%-----------------------------------------
    [K] = Interval_Finding(Xp,N,XN) ;
%
    Coef(1:4) = Sc(1:4,K)           ;
%-----------------------------------------
%  (2) Non-dimensional Variable
%-----------------------------------------
    Tau = ( Xp - XN(K,1) ) /( XN(K+1,1) - XN(K,1) ) ;
%-----------------------------------------
%  (3) Function Value using Spline Polynomuials
%-----------------------------------------
    Yp = Coef(1) + ( Coef(2) + ( Coef(3) + Coef(4)*Tau )*Tau )*Tau ;
%-----------------------------------------
end
%-----------------------------------------
```

❑ Program Structures: CubicSpline_Evaluate_Derivatives.m

```
%-------------------------------------------
function [dYp,ddYp] = CubicSpline_Evaluate_Derivatives(Xp,N,XN,Sc)
%-------------------------------------------
%  (1) Finding Interval and Spline Coefficients
%-------------------------------------------
    [K] = Interval_Finding(Xp,N,XN) ;
%
    Coef(1:4) = Sc(1:4,K)           ;
%-------------------------------------------
%  (2) Non-dimensional Variable
%-------------------------------------------
    Dx  = XN(K+1,1) - XN(K,1) ;
    Tau = ( Xp - XN(K,1) )/Dx ;
%-------------------------------------------
%  (3) Function Value using Spline Polynomuials
%-------------------------------------------
    dYp  = Coef(2) + ( 2.0*Coef(3) + 3.0*Coef(4)*Tau )*Tau ;
    ddYp = 2.0*Coef(3) + 6.0*Coef(4)*Tau                  ;
%
    dYp  =  dYp/Dx    ;
    ddYp = ddYp/Dx^2  ;
%-------------------------------------------
end
%-------------------------------------------
```

Results of Cubic Spline Interpolation

Nd= 21 ; Nv = 60



Evaluation of Function Derivatives Using Spline Coefficients

Spline Polynomials

$$\tau = \frac{x - x_j}{x_{j+1} - x_j} = \frac{x - x_j}{\Delta x_j} \in [0,1] \quad \leftarrow \begin{pmatrix} x_j \le x \le x_{j+1} \\ \Delta x_j = x_{j+1} - x_j \end{pmatrix}$$

$$y = \alpha_{j0} + \alpha_{j1}\tau + \alpha_{j2}\tau^2 + \alpha_{j3}\tau^3$$

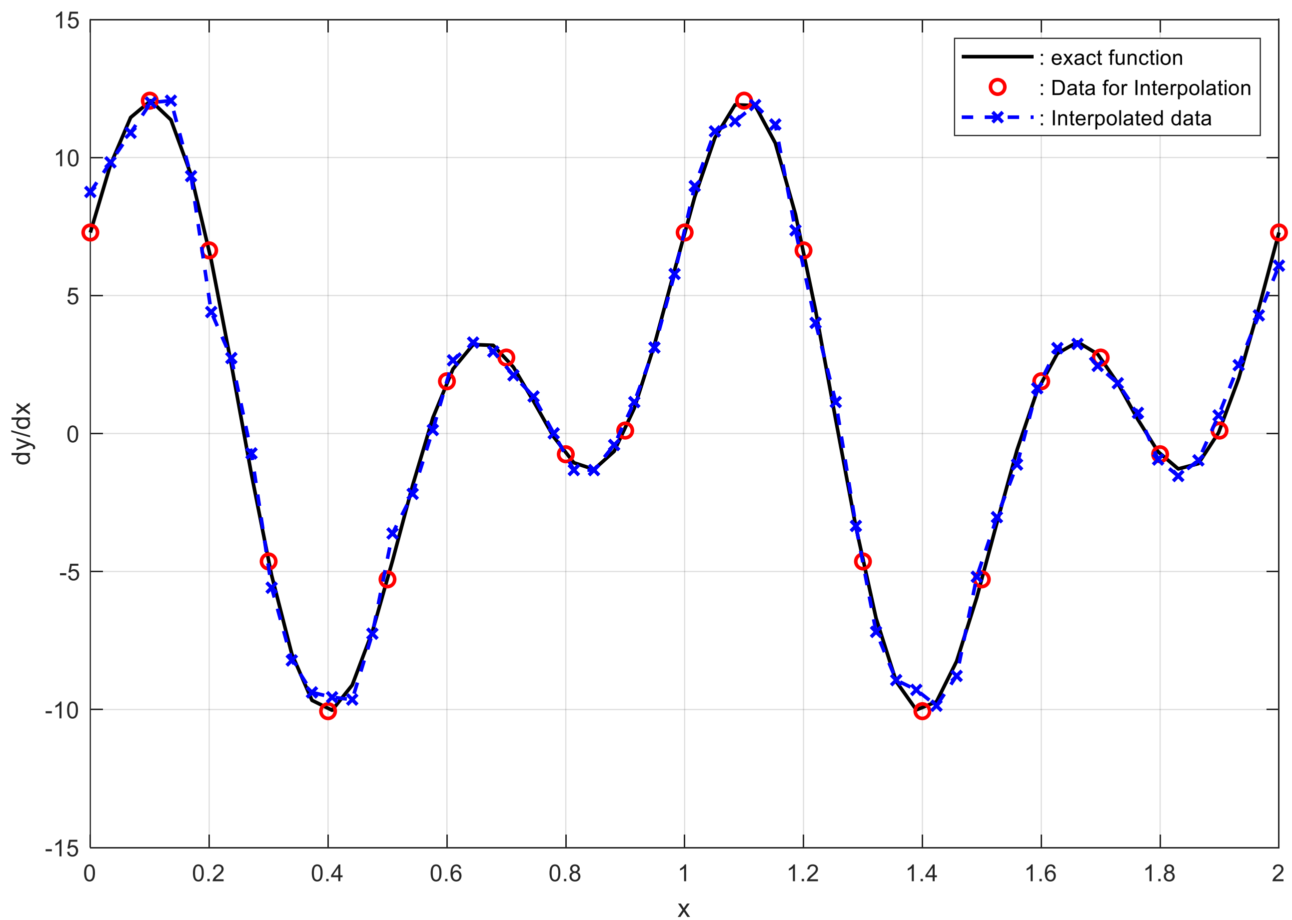
Derivatives

$$\frac{dy}{dx} = \frac{d\tau}{dx}\frac{dy}{d\tau} = \frac{1}{\Delta x_j}\left(\alpha_{j1} + 2\alpha_{j2}\tau + 3\alpha_{j3}\tau^2\right)$$

$$\frac{d^2 y}{dx^2} = \frac{1}{\Delta x_j^2}\left(2\alpha_{j2} + 6\alpha_{j3}\tau\right)$$

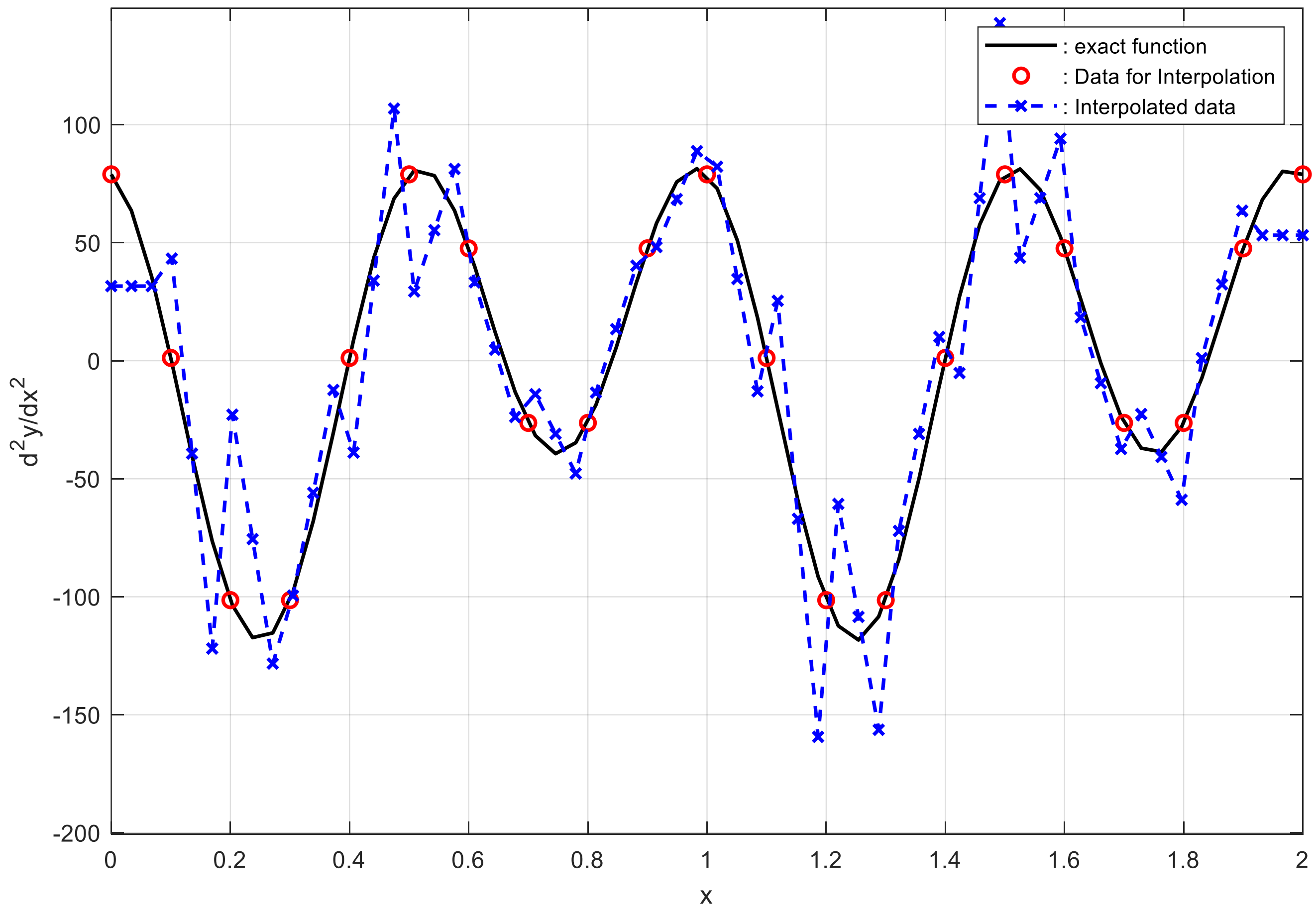
Results of Cubic Spline Interpolation: First Derivative

Nd= 21 ; Nv = 60



Results of Cubic Spline Interpolation: Second Derivative

Nd= 21 ; Nv = 60



Reference for Spline Integrator

Spline Polynomials

$$\tau = \frac{x - x_j}{x_{j+1} - x_j} = \frac{x - x_j}{\Delta x_j} \in [0,1] \leftarrow \begin{pmatrix} x_j \le x \le x_{j+1} \\ \Delta x_j = x_{j+1} - x_j \end{pmatrix}$$

$$y = \alpha_{j0} + \alpha_{j1}\tau + \alpha_{j2}\tau^2 + \alpha_{j3}\tau^3$$

$$\rightarrow dx = \Delta x_j d\tau$$

Integration Formula over $x_j \le x \le x_{j+1}$

$$I(x_j, x_{j+1}) = \int_{x_j}^{x_{j+1}} y(x)dx = \int_0^1 y\big(x(\tau)\big)\Delta x_j d\tau = \Delta x_j \int_0^1 y\big(x(\tau)\big)d\tau$$

$$= \Delta x_j \int_0^1 \left(\alpha_{j0} + \alpha_{j1}\tau + \alpha_{j2}\tau^2 + \alpha_{j3}\tau^3\right)d\tau$$

$$= \Delta x_j \left(\alpha_{j0}\tau + \alpha_{j1}\tau^2 + \frac{1}{2}\alpha_{j2}\tau^3 + \frac{1}{3}\alpha_{j3}\tau^4\right)\Bigg|_0^1$$

$$= \Delta x_j \left(\alpha_{j0} + \alpha_{j1} + \frac{1}{2}\alpha_{j2} + \frac{1}{3}\alpha_{j3}\right)$$

End of Lecture