

Joseph-Louis Lagrange (1736-1813)



Charles Hermite (1822-1901)

Summary of Lagrange/Hermite Interpolation

1) Find a “good” polynomial basis

➤ Lagrange Basis: $L_{n,k}(x)$

➤ (Newton) Divided-difference basis:

$$(x - x_0) \dots (x - x_{k-1})$$

➤ Hermite Basis: $H_{n,k}(x), \hat{H}_{n,k}(x)$

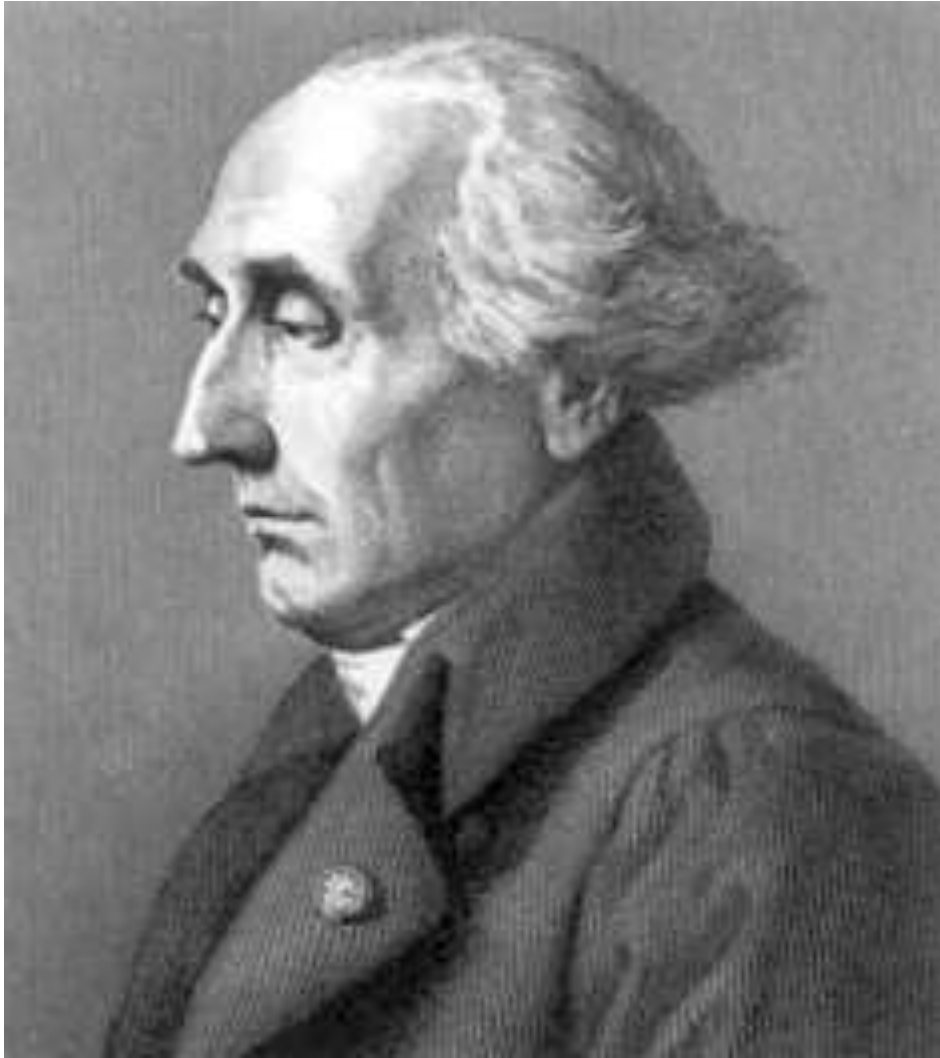
➤ (Newton) Divided-difference basis for Hermite:

$$(x - z_0) \dots (x - z_{k-1})$$

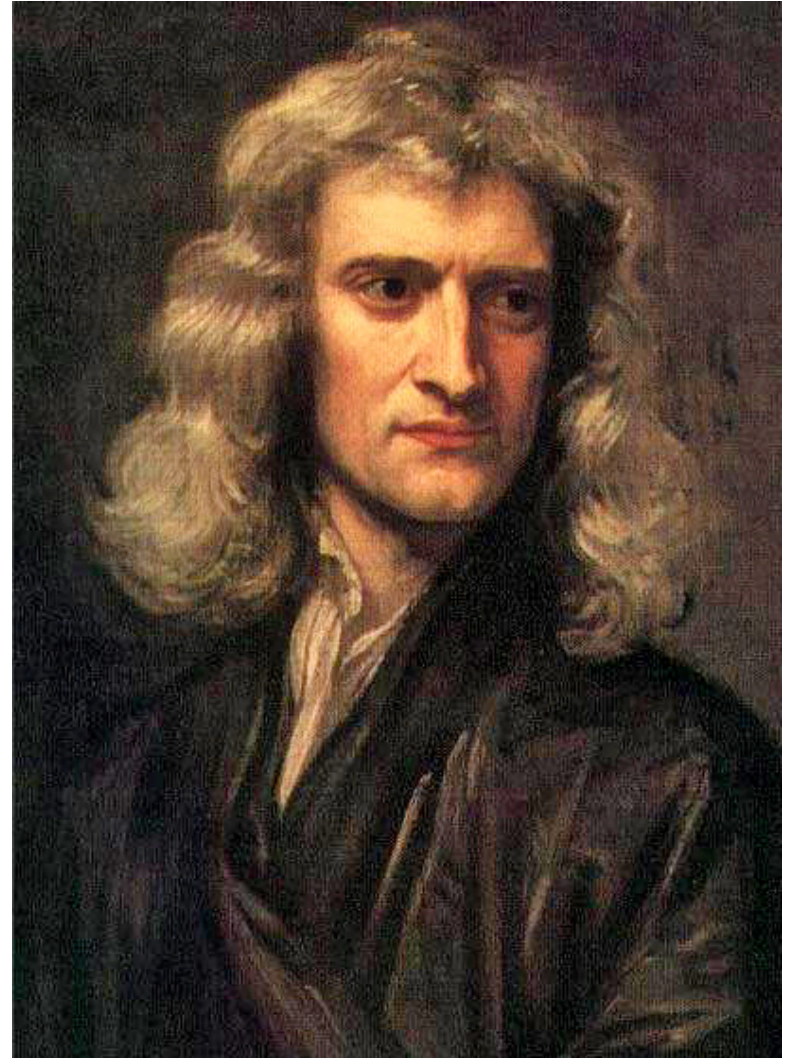
2) Interpolation formula

$$P_n(x) = \sum_{k=0}^n f(x_k) L_{n,k}(x)$$

$$H_n(x) = \sum_{k=0}^n f(x_k) H_{n,k}(x) + \sum_{k=0}^n f'(x_k) \hat{H}_{n,k}(x)$$



Joseph-Louis Lagrange (1736-1813)



Sir Isaac Newton (1642-1726)

Lagrange form V.S. Newton form

https://en.wikipedia.org/wiki/Newton_polynomial

Quoted from wiki:

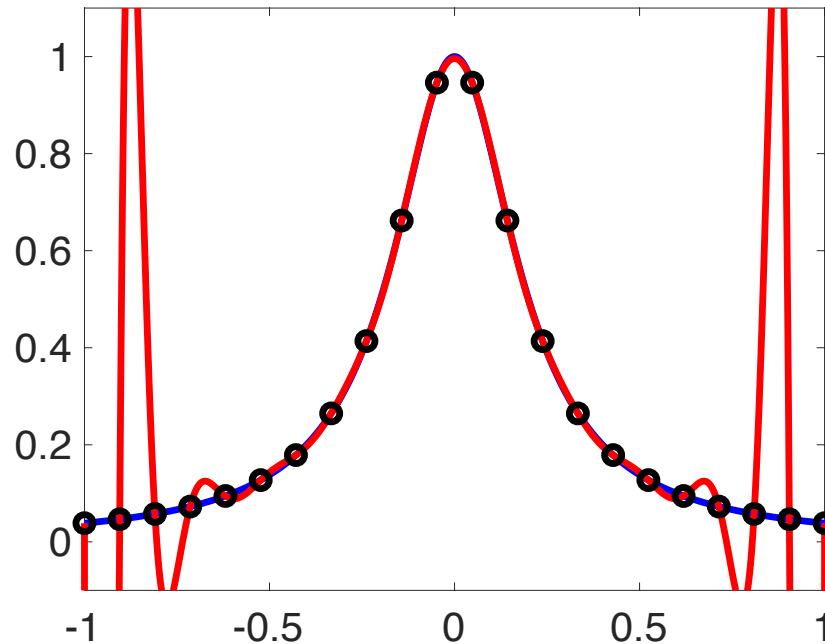
Lagrange is sometimes said to require less work, and is sometimes recommended for problems in which it's known, in advance, from previous experience, how many terms are needed for sufficient accuracy.

The divided difference methods have the advantage that more data points can be added, for improved accuracy, without re-doing the whole problem. The terms based on the previous data points can continue to be used. With the ordinary Lagrange formula, to do the problem with more data points would require re-doing the whole problem.

.....

3.5 Cubic Spline Interpolation

Problems with High Order Polynomial Interpolation



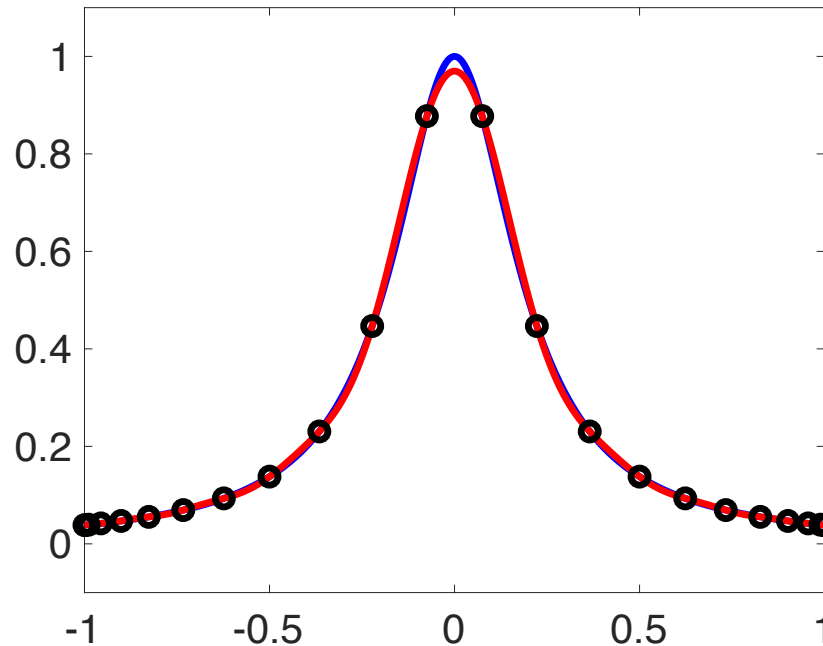
- 20th degree Lagrange interpolant for

$$f(x) = \frac{1}{1+25x^2}$$

on the interval $[-1,1]$ using 21 equal-spaced nodes.

- The Lagrange interpolating polynomial **oscillates** between interpolation points. [Runge's phenomenon]

Remedy 1: use non-uniform nodes



- 20th degree Lagrange interpolant for

$$f(x) = \frac{1}{1+25x^2}$$

on the interval $[-1,1]$ using $n=21$ **Chebyshev nodes**:

$$x_k = -\cos \frac{k\pi}{n}, k = 0, \dots, n$$

script .m file for Lagrange interp. with Cheb. nodes

lagrange_interp_cheby.m

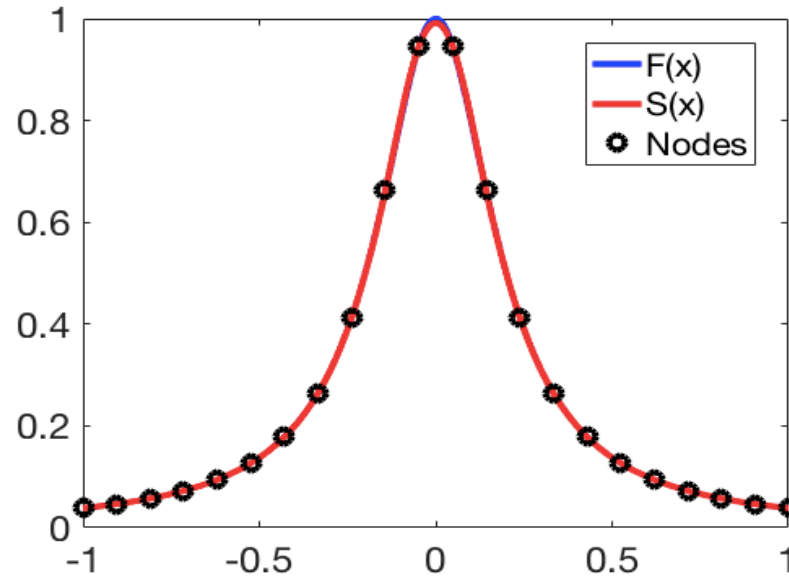
```
% function and interpolation nodes
f = @(x) 1./(1+25*x.*x);
n = 21; % degree 20 interpolation
%xNodes = linspace(-1,1,n+1);
xNodes = -cos([0:n]/n*pi); % chebyshev nodes
yNodes = f(xNodes);

% evaluate function at 1001 uniform points
m = 1001;
xGrid = linspace(-1,1,m);
pGrid = zeros(size(xGrid));

for k = 0:n
    yk = yNodes(k+1);
    phi_k = lagrange_basis(xGrid, xNodes, k); % k-th basis eval @
    xGrid
    pGrid = pGrid + yk*phi_k;
end

plot(xGrid, f(xGrid), 'b', xGrid, pGrid, 'r', xNodes, yNodes, 'ko', ...
     'LineWidth', 4, 'MarkerSize', 10)
set(gca, 'FontSize', 24)
axis([-1,1,-0.1,1.1])
shg
saveas(gcf, 'cheb.eps', 'epsc')
%saveas(gcf, 'uniform.eps', 'epsc')
```


Remedy 2: use smooth piecewise polynomials



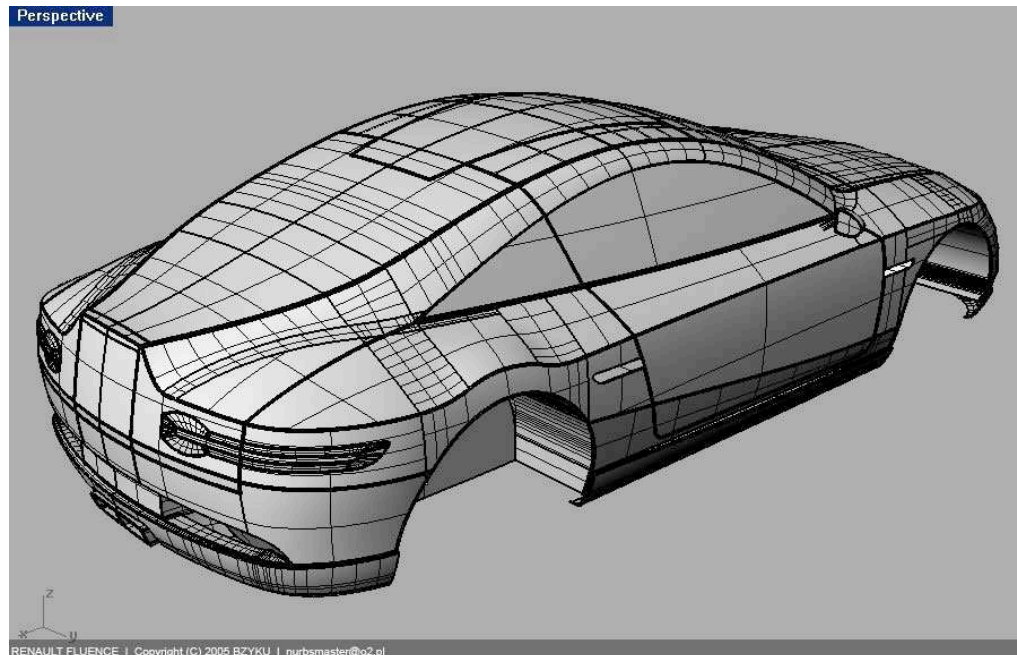
- $n=21$ equal-spaced interpolation nodes

$$x_k = -1 + \frac{2k}{n}, k = 0, \dots, n$$

- On each small interval $[x_j, x_{j+1}]$, $S(x)$ is a polynomial of degree 3
- $S(x)$ has continuous first and second derivatives at internal nodes x_j for $j = 1, \dots, n - 1$

(smooth) piecewise polynomial interpolation [splines]

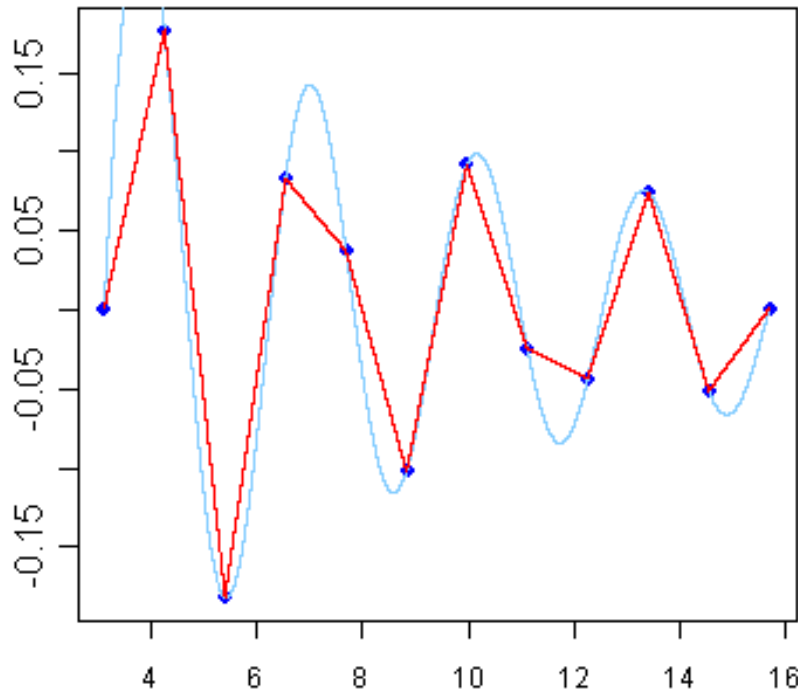
- **Spline:** A spline consists of a long strip of wood (a lath) fixed in position at a number of points. The lath will take the shape which minimizes the energy required for bending it between the fixed points, and thus adopt the smoothest possible shape.




Cubic Splines

- Idea: Use **piecewise polynomial interpolation**, i.e., divide the interval into smaller sub-intervals, and construct different low degree polynomial approximations (with small oscillations) on the sub-intervals.

Example. *Piecewise-linear* interpolation



- In mathematics, a spline is a function that is piecewise-defined by polynomial functions, and which possesses a high degree of smoothness at the places where the polynomial pieces connect.
- **Example.** Irwin-Hall distribution. Nodes are -2, -1, 0, 1, 2.

$$f(x) = \begin{cases} \frac{1}{4}(x+2)^3 & -2 \leq x \leq -1 \\ \frac{1}{4}(3|x|^3 - 6x^2 + 4) & -1 \leq x \leq 1 \\ \frac{1}{4}(2-x)^3 & 1 \leq x \leq 2 \end{cases}$$


Notice: $f \in C^2(-2,2)$, i. e. f, f', f'' are all continuous:

$$f'(-1) = \frac{3}{4}, f'(1) = -\frac{3}{4}, f''(-1) = \frac{6}{4}, f''(1) = \frac{6}{4}.$$

cubic spline interpolation is a piecewise-polynomial approximation that

- a) use *cubic* polynomials between each successive pair of nodes.
- b) has both continuous first and second derivatives at the internal nodes

Definition 3.10. Given a function f on $[a, b]$ and nodes $a = x_0 < \dots < x_n = b$, a **cubic spline interpolant** S for f satisfies:

(a) $S(x)$ is a cubic polynomial $S_j(x)$ on $[x_j, x_{j+1}]$ with:

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \\ \forall j = 0, 1, \dots, n - 1.$$

(b) [**interpolation property**]

$$S_j(x_j) = f(x_j) \text{ and } S_j(x_{j+1}) = f(x_{j+1}), \forall j = 0, 1, \dots, n - 1.$$

(c) [**continuous first derivative**]

$$S'_j(x_{j+1}) = S'_{j+1}(x_{j+1}), \forall j = 0, 1, \dots, n - 2.$$

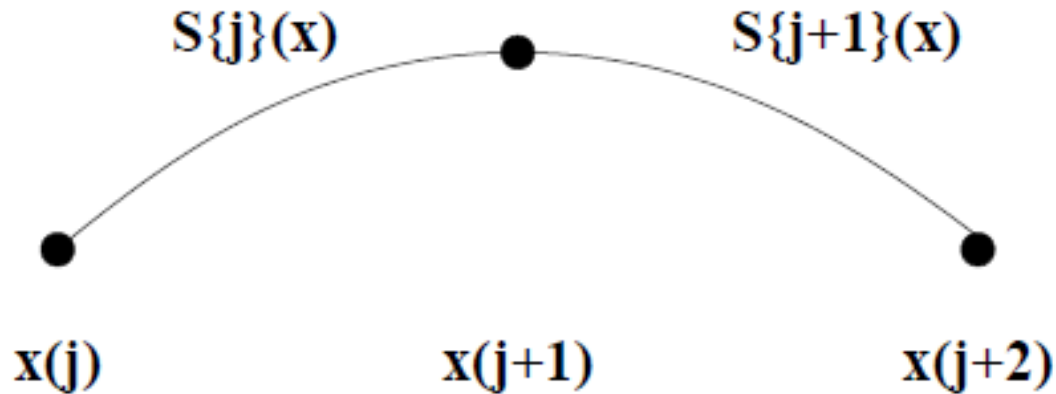
(d) [**continuous second derivative**]

$$S''_j(x_{j+1}) = S''_{j+1}(x_{j+1}), \forall j = 0, 1, \dots, n - 2.$$

(e) [One of the following **boundary conditions**]:

(i) $S''(x_0) = S''(x_n) = 0$ (called **free** or **natural boundary**)

(ii) $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (called **clamped boundary**)



The spline segment $S_j(x)$ is on $[x_j, x_{j+1}]$. The spline segment $S_{j+1}(x)$ is on $[x_{j+1}, x_{j+2}]$. Things to match at interior point x_{j+1} :

- Their function values: $S_j(x_{j+1}) = S_{j+1}(x_{j+1}) = f(x_{j+1})$
- First derivative values: $S'_j(x_{j+1}) = S'_{j+1}(x_{j+1})$
- Second derivative values: $S''_j(x_{j+1}) = S''_{j+1}(x_{j+1})$

The Cubic spline interpolation problem

Given data $(x_0, f(x_0)), \dots, (x_n, f(x_n))$, find the coefficients a_j, b_j, c_j, d_j for $j = 0, \dots, n - 1$ of the **Cubic spline interpolant** $S(x)$:

$$S(x) = \begin{cases} S_0(x) & x_0 \leq x \leq x_1 \\ \dots & \dots \\ S_j(x) & x_j \leq x \leq x_{j+1} \\ \dots & \dots \\ S_{n-1}(x) & x_{n-1} \leq x \leq x_n \end{cases}$$

Where $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$, such that conditions in (b)-(e) of **Definition 3.10** holds.

Example 1. Construct a natural cubic spline $S(x)$ through $(1,2)$, $(2,3)$ and $(3,5)$.

Building Cubic Splines

- **Define:** $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$
and $h_j = x_{j+1} - x_j, \forall j = 0, 1, \dots, (n - 1)$.
- **Also define** $a_n = f(x_n); b_n = S'(x_n); c_n = S''(x_n)/2$.

From Definition 3.10:

- 1) $S_j(x_j) = a_j = f(x_j)$ for $j = 0, 1, \dots, (n - 1)$.
- 2) $S_j(x_{j+1}) = a_{j+1} = a_j + b_j h_j + c_j (h_j)^2 + d_j (h_j)^3$
for $j = 0, 1, \dots, (n - 1)$.
- 3) $S'_j(x_j) = S'_{j+1}(x_j): b_{j+1} = b_j + 2c_j h_j + 3d_j (h_j)^2$
for $j = 0, 1, \dots, (n - 1)$.
- 4) $S''_j(x_j) = S''_{j+1}(x_j): c_{j+1} = c_j + 3d_j h_j$
for $j = 0, 1, \dots, (n - 1)$.
- 5) Natural or clamped boundary conditions

Solve a_j, b_j, c_j, d_j by substitution:

1. Solve Eq. 4) for $d_j = \frac{c_{j+1} - c_j}{3h_j}$ (3.17), and substitute into Eqs. 2) and 3) to get:

2. $a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3} (2c_j + c_{j+1});$ (3.18)

$$b_{j+1} = b_j + h_j (c_j + c_{j+1}). \quad (3.19)$$

3. Solve for b_j in Eq. (3.18) to get:

$$b_j = \frac{1}{h_j} (a_{j+1} - a_j) - \frac{h_j}{3} (2c_j + c_{j+1}). \quad (3.20)$$

Reduce the index by 1 to get:

$$b_{j-1} = \frac{1}{h_{j-1}} (a_j - a_{j-1}) - \frac{h_{j-1}}{3} (2c_{j-1} + c_j).$$

4. Substitute b_j and b_{j-1} into Eq. (3.19) [reduce index by 1]:

$$h_{j-1} c_{j-1} + 2(h_{j-1} + h_j) c_j + h_j c_{j+1} = \frac{3}{h_j} (a_{j+1} - a_j) - \frac{3}{h_{j-1}} (a_j - a_{j-1}) \quad (3.21)$$

for $j = 1, 2, \dots, (n - 1)$.

Solving the Resulting Equations for c

$$\forall j = 1, 2, \dots, (n - 1)$$

$$\begin{aligned} & h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_jc_{j+1} \\ &= \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \end{aligned} \quad (3.21)$$

Remark: $(n-1)$ equations for $(n+1)$ unknowns $\{c_j\}_{j=0}^n$. Eq. (3.21) is solved with **boundary conditions**.

- Once compute c_j , we then compute:

$$b_j = \frac{(a_{j+1} - a_j)}{h_j} - \frac{h_j(2c_j + c_{j+1})}{3} \quad (3.20)$$

and

$$d_j = \frac{(c_{j+1} - c_j)}{3h_j} \quad (3.17) \text{ for } j = 0, 1, 2, \dots, (n - 1)$$

Building Natural Cubic Spline

- Natural boundary condition:

1. $0 = S''_0(x_0) = 2c_0 \rightarrow c_0 = 0$

2. $0 = S''_n(x_n) = 2c_n \rightarrow c_n = 0$

Step 1. Solve Eq. (3.21) together with $c_0 = 0$ and $c_n = 0$ to obtain $\{c_j\}_{j=0}^n$.

Step 2. Solve Eq. (3.20) to obtain $\{b_j\}_{j=0}^{n-1}$.

Step 3. Solve Eq. (3.17) to obtain $\{d_j\}_{j=0}^{n-1}$.

Building Natural Cubic Spline

- Natural boundary condition:

1. $0 = S''_0(x_0) = 2c_0 \rightarrow c_0 = 0$

2. $0 = S''_n(x_n) = 2c_n \rightarrow c_n = 0$

Step 1. Solve Eq. (3.21) together with $c_0 = 0$ and $c_n = 0$ to obtain $\{c_j\}_{j=0}^n$.

Step 2. Solve Eq. (3.20) to obtain $\{b_j\}_{j=0}^{n-1}$.

Step 3. Solve Eq. (3.17) to obtain $\{d_j\}_{j=0}^{n-1}$.

Building Clamped Cubic Spline

- Clamped boundary condition:

$$a) S'_0(x_0) = b_0 = f'(x_0)$$

$$b) S'_{n-1}(x_n) = b_{n-1} + h_{n-1}(c_{n-1} + c_n) = f'(x_n)$$

Remark: a) and b) gives additional equations:

$$2h_0c_0 + h_0c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(x_0) \quad (a)$$

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = -\frac{3}{h_{n-1}}(a_n - a_{n-1}) + 3f'(x_n) \quad (b)$$

Step 1. Solve Eq. (3.21) together with Eqs. (a) and (b) to obtain $\{c_j\}_{j=0}^n$.

Step 2. Solve Eq. (3.20) to obtain $\{b_j\}_{j=0}^{n-1}$.

Step 3. Solve Eq. (3.17) to obtain $\{d_j\}_{j=0}^{n-1}$.

Building Cubic Splines: Summary

Step 0: Obtain a -vector and h -vector:

$$a = [a_0, a_1, \dots, a_n]', \quad h = [h_0, h_1, \dots, h_{n-1}]'$$

$$a_j = f(x_j), \quad h_j = x_{j+1} - x_j$$

Step 1: solve tri-diagonal linear system $Ac = F$ to obtain c -vector $c = [c_0, c_1, \dots, c_n]'$, where

Natural:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & & & \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & \cdots & & \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \cdots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} & \\ 0 & \cdots & 0 & 0 & 0 & 1 & \end{bmatrix} \quad F = \begin{bmatrix} 0 & & & & & & \\ & 3(a_2 - a_1)/h_1 - 3(a_1 - a_0)/h_0 & & & & & \\ & 3(a_3 - a_2)/h_2 - 3(a_2 - a_1)/h_1 & & & & & \\ & & \vdots & & & & \\ & & & & & & 3(a_n - a_{n-1})/h_{n-1} - 3(a_{n-1} - a_{n-2})/h_{n-2} \\ & & & & & & 0 \end{bmatrix}$$

Clamped:

$$A = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & & & \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & \cdots & & \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \cdots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} & \\ 0 & \cdots & 0 & 0 & h_{n-1} & 2h_{n-1} & \end{bmatrix} \quad F = \begin{bmatrix} & & & & & & 3(a_1 - a_0)/h_0 - 3f'(x_0) \\ & 3(a_2 - a_1)/h_1 - 3(a_1 - a_0)/h_0 & & & & & \\ & 3(a_3 - a_2)/h_2 - 3(a_2 - a_1)/h_1 & & & & & \\ & & \vdots & & & & \\ & & & & & & 3(a_n - a_{n-1})/h_{n-1} - 3(a_{n-1} - a_{n-2})/h_{n-2} \\ & & & & & & -3(a_n - a_{n-1})/h_{n-1} + 3f'(x_n) \end{bmatrix}$$

Building Cubic Splines: Summary

Step 2: Recover b -vector $b = [b_0, b_1, \dots, b_{n-1}]'$:

$$b_j = \frac{(a_{j+1} - a_j)}{h_j} - \frac{h_j(2c_j + c_{j+1})}{3} \quad (3.20)$$

Step 3: Recover d -vector $d = [d_0, d_1, \dots, d_{n-1}]'$:

$$d_j = \frac{c_{j+1} - c_j}{3h_j} \quad (3.17)$$

Remark: Main computational cost is the linear system solve in Step 1. In MATLAB, we simply use the “backslash” `\` command to solve the system:

$$c = A \setminus F;$$

Example 2. (MATLAB) Use data points

$(0,1), (1, e), (2, e^2), (3, e^3)$ to form

a) a natural cubic spline $S(x)$ that approximates

$$f(x) = e^x.$$

b) a clamped cubic spline $S(x)$ that approximates

$$f(x) = e^x \text{ with derivative information } f'(0) = 1, f'(3) = e^3.$$

natural cubic spline (script .m file)

natural_cubic_spline.m

```
% input: data points
f = @(x) exp(x);
xNodes = [0:3]'; % column vector
yNodes = f(xNodes);

%--- STEP 0: obtain the a-vector
a = yNodes;

%--- STEP 1: form linear system, solve for c-vector
n = length(xNodes)-1;
% mesh size, vector of size n
h = xNodes(2:n+1)-xNodes(1:n);
% form the matrix
A = zeros(n+1,n+1);
A(1,1) =1;
for j = 2:n
    A(j,j-1) = h(j-1);
    A(j,j) = 2*(h(j-1)+h(j));
    A(j,j+1) = h(j);
end
A(n+1,n+1)=1;
% form the right hand side
F = zeros(n+1,1);
for j = 2:n
    F(j) = 3/h(j)*(a(j+1)-a(j))...
        - 3/h(j-1)*(a(j)-a(j-1));
end

% solve for c [backslash]
c = A\F;
```

```
%--- STEP 2: solve for b-vector
b = zeros(n,1);
for j = 1:n
    b(j) = 1/h(j)*(a(j+1)-a(j))...
        -h(j)/3*(2*c(j)+c(j+1));
end

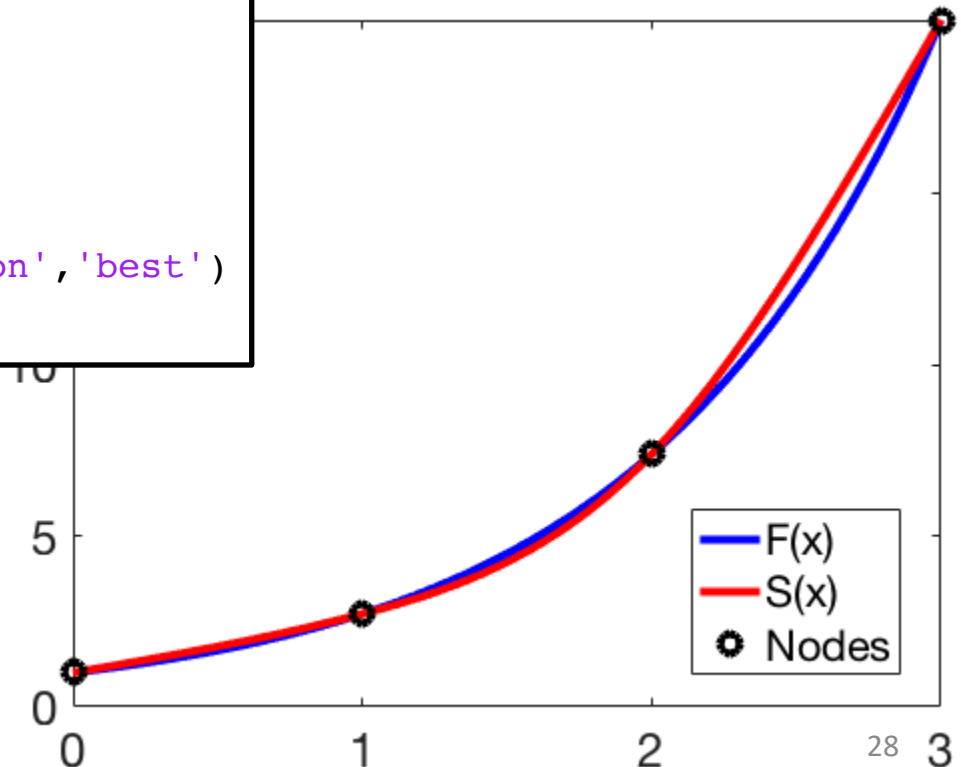
%--- STEP 3: solve for d-vector
d = zeros(n,1);
for j=1:n
    d(j) = (c(j+1)-c(j))/3/h(j);
end
```

natural cubic spline (script .m file continued)

natural_cubic_spline.m

```
%--- STEP 4: plot solution
% loop over subintervals
xGrid = []; sGrid = [];
for j = 1:n
    x0 = xNodes(j); x1 = xNodes(j+1);
    xTemp = linspace(x0,x1,20);
    sTemp = a(j)+b(j)*(xTemp-x0)...
            +c(j)*(xTemp-x0).^2+d(j)*(xTemp-x0).^3;
    xGrid = [xGrid, xTemp];
    sGrid = [sGrid, sTemp];
end
plot(xGrid, f(xGrid), 'b', ...
     xGrid, sGrid, 'r', ...
     xNodes, yNodes, 'ko', ...
     'LineWidth', 4, 'MarkerSize', 10)
legend('F(x)', 'S(x)', 'Nodes', 'Location', 'best')
set(gca, 'FontSize', 24)
```

*****clamped cubic spline code is given
in course webpage**



Existence and Uniqueness

Theorem 3.11 If f is defined at the nodes: $a = x_0 < \dots < x_n = b$, then f has a unique natural spline interpolant S on the nodes; that is a spline interpolant that satisfied the natural boundary conditions $S''(a) = 0, S''(b) = 0$.

Theorem 3.12 If f is defined at the nodes: $a = x_0 < \dots < x_n = b$ and differentiable at a and b , then f has a unique clamped spline interpolant S on the nodes; that is a spline interpolant that satisfied the clamped boundary conditions $S'(a) = f'(a), S'(b) = f'(b)$.

Error Bound

Theorem 3.13 If $f \in C^4[a, b]$, let $M = \max_{a \leq x \leq b} |f^{(4)}(x)|$. If S is the unique clamped cubic spline interpolant to f with respect to the nodes: $a = x_0 < \dots < x_n = b$, then with

$$h = \max_{0 \leq j \leq n-1} (x_{j+1} - x_j)$$

$$\max_{a \leq x \leq b} |f(x) - S(x)| \leq \frac{5Mh^4}{384}.$$