Chapter 3: PIECEWISE POLYNOMIAL INTERPOLATION

FuSen Lin

Department of Computer Science and Engineering
National Taiwan Ocean University

Scientific Computing, Fall 2007
3 Piecewise Polynomial Interpolation

- 3.0 Introduction to Piecewise Polynomial Interpolation
- 3.1 Piecewise Linear Interpolation
- 3.2 Piecewise Cubic Hermite Interpolation
- 3.3 Cubic Splines
For the data-fitting problem, if we attempt to produce an accurate polynomial interpolant, then the *high-degree* polynomial at equally spaced points should be avoided.

For the given data \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\), suppose that

\[
\alpha = x_1 < x_2 < \cdots < x_n = \beta.
\]

Another scheme to obtain an accurate interpolant across the interval \([\alpha, \beta]\) is to interpolate each subinterval \([x_i, x_{i+1}]\) with a low-degree polynomial.
This means the interpolating function can be expressed as

\[ S(x) = \begin{cases} 
S_0(x), & x \in [x_0, x_1] \\
S_1(x), & x \in [x_1, x_2] \\
\vdots \\
S_{n-1}(x), & x \in [x_{n-1}, x_n] 
\end{cases} \]

where \( S_i(x) \) are low-degree polynomials.

This idea is called piecewise polynomial interpolation and the \( x_i \) are called breakpoints or knots.

The simplest case is the piecewise linear interpolation, in which the functions \( S_i(x) \) are linear polynomials.

The piecewise linear functions do not have a continuous first derivative, and this creates problems in certain applications.
The piecewise cubic Hermite interpolants overcome this issue by forcing the continuity of the first derivative at each knot.

Second derivative continuity can be achieved by carefully choosing the first derivative values at the knots. This leads to the topic of splines, a very important idea in the area of approximation and interpolation.

The cubic splines produce the smoothest solution to the interpolation problem. We shall address this in Section 3.3.
Assume that \( x(1 : n) \) and \( y(1 : n) \) are given where 
\[ \alpha = x_1 < x_2 < \cdots < x_n = \beta \] 
and \( y_i = f(x_i), \ i = 1 : n. \) 
Connecting the points \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) with straight lines, we obtain a piecewise linear function, called piecewise linear interpolant or a spline of degree 1.

The piecewise linear interpolant is built upon the local linear interpolants

\[ S_i(z) := L_i(z) = a_i + b_i(z - x_i), \quad z \in [x_i, x_{i+1}], \]

for \( i = 1 : n - 1 \), where the coefficients are defined by

\[ a_i = y_i \quad \text{and} \quad b_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}. \]
Introduction to Piecewise Polynomial Interpolation

3.1 Piecewise Linear Interpolation

3.2 P.W. Cubic Hermite Interpolation

3.3 Cubic Splines

---

**Piecewise Linear Interpolation (Continuing)**

- Note that $L_i(z)$ is just the linear interpolant of $f$ at the two points $(x_i, y_i), (x_{i+1}, y_{i+1})$ and then define

$$S(z) := L(z) = \begin{cases} 
L_1(z), & \text{if } z \in [x_1, x_2] \\
L_2(z), & \text{if } z \in [x_2, x_3] \\
\vdots & \\
L_n(z), & \text{if } z \in [x_{n-1}, x_n]
\end{cases}$$

- To get the coefficients of $L_i(z)$, we only need to compute the $n - 1$ divided differences by a loop or by using pointwise division or by using the built-in function `diff`:

$$b = \text{diff}(y)/\text{diff}(x).$$
Next we want to evaluate $L(x)$ at any point $z \in [\alpha, \beta]$, it is necessary to first determine the subinterval that contains $z$.

That is to determine the index $i$ so that $x_i \leq z \leq x_{i+1}$. You may use the command `sum(x <= z)` or the better approach `binary search`.

The `binary search` roughly does $\log_2(n)$ comparisons to locate the appropriate subinterval. If $n$ is large, then this is much more efficient than the `sum(x <= z)` method, which requires $n$ comparisons.
If evaluating $L(x)$ at an ordered succession of points, then we can improve the subinterval location process by an initial guess.

That is to evaluate the values $L(z_1), ..., L(z_m)$ in a vector $z = [z_1, z_2, ..., z_m]$ where $m$ is a typically large integer and

$$\alpha = z_1 < z_2 < \cdots < z_m = \beta.$$ 

In this case, rather than locate each $z_i$ via binary search, it is more efficient to exploit the systematic ”migration” of the evaluation point as it moves left to right across the subintervals.
Chances are that if \( i \) is the subinterval index associated with the current \( z \)-value, then \( i \) will be the correct index for the next \( z \)-value.

This "guess" at the correct subinterval can be checked before we launch the binary search process.

Example: to produce a sequence of piecewise linear approximations to the function

\[
humps(x) = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6.
\]
A Priori Determination of the Knots

- Consider how many knots we need to obtain a satisfactory P.W. linear interpolant. If \( z \in [x_i, x_{i+1}] \), then by Theorem 2,

\[
    f(z) = L(z) + \frac{f^{(2)}(\eta)}{2}(z - x_i)(z - x_{i+1}),
\]

where \( x_i \leq \eta \leq x_{i+1} \).

- If \( |f^{(2)}(x)| \leq M_2 \) for all \( x \in [\alpha, \beta] \) and \( \bar{h} \) is the length of the longest subinterval in the partition, then

\[
    |f(z) - L(z)| \leq \frac{M_2 \bar{h}^2}{8}
\]

for all \( z \in [\alpha, \beta] \).

- Assume that \( L(x) \) is based on the uniform partition

\[
    x_i = \alpha + \frac{i - 1}{n - 1}(\beta - \alpha), \quad i = 1 : n.
\]
To ensure that the error is less than or equal to a given tolerance $\delta$, we insist that

$$|f(z) - L(z)| \leq \frac{M_2 \bar{h}^2}{8} = \frac{M_2}{8} \left( \frac{\beta - \alpha}{n - 1} \right)^2 \leq \delta.$$ 

From this we conclude that $n$ must satisfy

$$n \geq 1 + (\beta - \alpha) \sqrt{M_2/8\delta}.$$ 

and we certainly choose the smallest integer $n$ for efficiency.
Adaptive P.W. Linear Interpolation

- The partition produced by `pwLStatic.m` does not take into account the sampled values of $f$. As a result, the uniform partition produced may be *much too refined* in the regions where $f''$ is much smaller than the upper bound $M_2$.

- This leads that *lots of subintervals and (perhaps costly)* $f$-evaluations will be required. Over regions where $f$ is smooth, the partition will be *overly refined*.

- To overcome this problem, we develop a recursive partitioning algorithm that *discovers* where $f$ is *extra nonlinear* and clusters the breakpoints accordingly.
Adaptive P.W. Linear Interpolation (Continuing)

- The subinterval is *acceptable* if
  \[
  \left| f \left( \frac{x_L + x_R}{2} \right) - \frac{f(x_L) + f(x_R)}{2} \right| \leq \delta
  \]
  or if \(x_R - x_L \leq h_{\text{min}}\), where \(\delta > 0\) and \(h_{\text{min}} > 0\) are refinement parameters.

- A partition \(x_1 < \cdots < x_n\) is *acceptable* if each subinterval is acceptable. Note that if
  \[
  x_L = x_L^1 < \cdots < x_L^n = m
  \]
  is an acceptable partition of \([x_L, m]\) and if
  \[
  m = x_R^1 < \cdots < x_R^n = x_R
  \]
  is an acceptable partition of \([m, x_R]\), then
  \[
  x_L = x_L^1 < \cdots < x_L^n = m = x_1^R < x_2^R < \cdots < x_n^R = x_R
  \]
  is an acceptable partition of \([x_L, x_R]\).
3.2 P.W. Cubic Hermite Interpolation

- For a given \( n \) points, we interpolate both function \( S_i(x) \) and its derivative with a cubic polynomial on each subinterval. This approach is called **piecewise cubic Hermite interpolation**.

- Consider the example: the interpolation of function \( f(z) = \cos(z) \) at the points \( x_1 = 0, \ x_2 = \delta, \ x_3 = 3\pi/2 - \delta \), and \( x_4 = 3\pi/2 \) by a cubic \( p_3(z) \).

- For small \( \delta \), the \( p_3(z) \) seems to interpolate both \( f \) and \( f' \) at \( z = 0 \) and \( z = 3\pi/2 \). This is called the **Hermite cubic interpolant**.
Suppose a subinterval \([x_L, x_R]\) is given and the function values \(y_L = f(x_L)\) and \(y_R = f(x_R)\) and also the derivative values \(s_L = f'(x_L)\) and \(s_R = f'(x_R)\) are known. We want to find a cubic polynomial

\[
q(z) = a + b(z - x_L) + c(z - x_L)^2 + d(z - x_L)^2(z - x_R)
\]
satisfying \(q(x_L) = y_L\), \(q(x_R) = y_R\), \(q'(x_L) = s_L\), and \(q'(x_R) = s_R\).

This means that we need to seek coefficients \(a\), \(b\), \(c\), and \(d\) from the above four equations. Noting that

\[
q'(z) = b + 2c(z - x_L) + d[2(z - x_L)(z - x_R) + (z - x_L)^2].
\]
We see that

\[ a = y_L \quad a + b\Delta x + c(\Delta x)^2 = y_R \]

\[ b = s_L \quad b + 2c\Delta x + d(\Delta x)^2 = s_R, \]

where \( \Delta x = x_R - x_L \).

Expressing this in matrix-vector form, we obtain

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & \Delta x & (\Delta x)^2 & 0 \\
0 & 1 & 2\Delta x & (\Delta x)^2 \\
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d \\
\end{bmatrix}
= 
\begin{bmatrix}
y_L \\
s_L \\
y_R \\
s_R \\
\end{bmatrix}
\]
The solution to this triangular system is straightforward:

\[ a = y_L, \quad b = s_L, \quad c = \frac{y'_L - s_L}{\Delta x}, \quad d = \frac{s_R + s_L - 2y'_L}{(\Delta x)^2} \]

where

\[ y'_L = \frac{y_R - y_L}{\Delta x} = \frac{y_R - y_L}{x_R - x_L}. \]
**Theorem 3**: Suppose $f(z)$ and its first four derivatives are continuous on $[x_L, x_R]$ and that there is a positive constant satisfies

$$|f^{(4)}(z)| \leq M_4 \quad \text{for all} \quad z \in [x_L, x_R].$$

If $q$ is the cubic Hermite Interpolant of $f$ at $x_L$ and $x_R$, then

$$|f(z) - q(z)| \leq \frac{M_4}{384} h^4, \quad \text{where} \quad h = x_R - x_L.$$
Proof of Theorem 3 (Continuing)

PROOF: If $q_{\delta}(z)$ is the cubic Hermite Interpolant of $f$ at $x_L$, $x_L + \delta$, $x_R - \delta$, and $x_R$, then from Theorem 2 we have

$$|f(z) - q_{\delta}(z)| \leq \frac{M_4}{24}(z - x_L)(z - x_L - \delta)(z - x_R + \delta)(z - x_R),$$

for all $z \in [x_L, x_R]$. We assume without proof that

$$\lim_{\delta \to 0} q_{\delta}(z) = q(z)$$

and so

$$|f(z) - q(z)| \leq \frac{M_4}{24}|(z - x_L)(z - x_L)(z - x_R)(z - x_R)|.$$

The maximum value of the quartic (degree 4) polynomial occurs right at the midpoint $z = x_L + h/2$ and so for all $z \in [x_L, x_R]$ we have

$$|f(z) - q(z)| \leq \frac{M_4}{24} \left(\frac{h}{2}\right)^4 = \frac{M_4}{384} h^4.$$
Set-Up the Cubic Hermite Interpolant

We now consider a whole interval \([a, b]\) where \(a = x_1 < \cdots < x_n = b\) and show how to combine a sequence of Hermite Cubic Interpolants together so that the resulting piecewise cubic polynomial \(C(z)\) interpolates the data \((x_1, y_1), \ldots, (x_n, y_n)\), with the prescribed slopes \(s_1, \ldots, s_n\).

To this end we define the \(i\)th local cubic by

\[
q_i(z) = a_i + b_i(z - x_i) + c_i(z - x_i)^2 + d_i(z - x_i)^2(z - x_{i+1})
\]

and the piecewise cubic polynomial by

\[
C(x) = \begin{cases} 
q_1(x), & \text{if } x \in [x_1, x_2] \\
q_2(x), & \text{if } x \in [x_2, x_3] \\
\vdots & \\
q_{n-1}(x), & \text{if } x \in [x_{n-1}, x_n]
\end{cases}
\]
Our goal is to determine coefficients \( a(1 : n), b(1 : n), c(1 : n), \) and \( d(1 : n) \) so that

\[
C(x_i) = y_i, \quad C'(x_i) = s_i, \quad i = 1 : n.
\]

This will be the case if we solve the following \( n - 1 \) cubic Hermite problems:

\[
q_i(x_i) = y_i, \quad q'_i(x_i) = s_i, \quad q_i(x_{i+1}) = y_{i+1}, \quad q'_i(x_{i+1}) = y_{i+1}.
\]

The results are

\[
a_i = y_i, \quad b_i = s_i, \quad c_i = \frac{y'_i - s_i}{\Delta x_i}, \quad d_i = \frac{s_{i+1} + s_i - 2y'_i}{(\Delta x_i)^2}
\]

where

\[
\Delta x_i = x_{i+1} - x_i \quad \text{and} \quad y'_i = \frac{y_{i+1} - y_i}{\Delta x_i} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.
\]
We could use `HCubic.m` with a for-loop to resolve the coefficients:

```
for i = 1 : n - 1,
    [a(i), b(i), c(i), d(i)] = HCubic(x(i), y(i), s(i), x(i + 1), y(i + 1), s(i + 1))
end
```

But a better solution is to vectorize the computation, and this gives `pwCH.m`

In Theorem 3, if $M_4$ bounds $|f^{(4)}(x)|$ on the interval $[x_1, x_n]$ then the error bound should be modified as

$$|f(z) - C(z)| \leq \frac{M_4}{384} \tilde{h}^4$$

for all $z \in [x_1, x_n]$, where $\tilde{h}$ is the length of the longest subinterval (i.e., $\max |x_{i+1} - x_i|$).
The evaluation of $C(z)$ is similar to that of any piecewise interpolation having two parts. First, the position of $z$ (in which subinterval) has to be determined and then the relevant local cubic must be evaluated.

The MATLAB function `pwCEval.m` can be used to evaluate $C$ at a vector of $z$ values.
3.3 Cubic Spline Interpolation

- In the **piecewise cubic Hermite interpolation** problem, we are given \( n \) triplets

\[
(x_1, y_1, s_1), \ldots, (x_n, y_n, s_n)
\]

and determine a function \( C(x) \) that is piecewise cubic with the property that \( C(x_i) = y_i \) and \( C'(x_i) = s_i \) for \( i = 1 : n \). The **disadvantage** of this method is that: The function \( C(x) \) does not have a continuous second derivative.

- This prompts us to pose the **cubic spline interpolation** problem: Given \((x_1, y_1), \ldots, (x_n, y_n)\) with \( \alpha = x_1 < \cdots < x_n = \beta \), a piecewise cubic spline function \( S(z) \) with the property that \( S, S', \) and \( S'' \) are continuous.

- By choosing the appropriate slope values \( s_1, \ldots, s_n \), the function \( S(z) \) that solves this problem is a **cubic spline interpolant**.
Assume that $S(z)$ is the Cubic Hermite Interpolant of the data $(x_i, y_i, s_i)$ for $i = 1 : n$. Is it possible to choose $s_1, \ldots, s_n$ so that $S''$ is continuous?

Let us look at what happens to $S''$ at each of the interior knots $x_2, \ldots, x_{n-1}$:

To the left of $x_{i+1}$, $S(z)$ is defined by the local cubic

$$q_i(z) = y_i + s_i(z - x_i) + \frac{y'_i - s_i}{\Delta x_i} (z - x_i)^2 + \frac{s_i + s_{i+1} - 2y'_i}{(\Delta x_i)^2} (z - x_i)^2 (z - x_{i+1})$$

where $y'_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$ and $\Delta x_i = x_{i+1} - x_i$. 
The 2nd derivative of this local cubic is given by

\[ q_i'' = 2 \frac{y_i' - s_i}{\Delta x_i} + \frac{s_i + s_{i+1} - 2y_i'}{(\Delta x_i)^2} [4(z - x_i) + 2(z - x_{i+1})]. \]  

Likewise, to the right of \( x_{i+1} \), the piecewise cubic \( S(z) \) is defined by

\[ q_{i+1}(z) = y_{i+1} + s_{i+1}(z - x_{i+1}) + \frac{y_{i+1}' - s_{i+1}}{\Delta x_{i+1}} (z - x_{i+1})^2 + \frac{s_{i+1} + s_{i+2} - 2y_{i+1}'}{(\Delta x_{i+1})^2} (z - x_{i+1})^2 (z - x_{i+2}) \]

The 2nd derivative of this local cubic is given by

\[ q_{i+1}'' = 2 \frac{y_{i+1}' - s_{i+1}}{\Delta x_{i+1}} + \frac{s_{i+1} + s_{i+2} - 2y_{i+1}'}{(\Delta x_{i+1})^2} [4(z - x_{i+1}) + 2(z - x_{i+2})]. \]  

(2)
To force the 2nd derivative continuity at \( x_{i+1} \), we insist

\[
q_i''(x_{i+1}) = \frac{2}{\Delta x_i}(2s_{i+1} + s_i - 3y_i')
\]

and

\[
q_{i+1}''(x_{i+1}) = \frac{2}{\Delta x_{i+1}}(3y_{i+1}' - 2s_{i+1} - s_{i+2})
\]

be equal.

That is,

\[
\Delta x_{i+1}s_i + 2(\Delta x_i + \Delta x_{i+1})s_{i+1} + \Delta x_is_{i+2} = 3(\Delta x_{i+1}y'_i + \Delta x_iy'_{i+1})
\]

for \( n = 1 : n - 2 \). If we choose \( s_1, ..., s_n \) to satisfy these equations, then \( S''(z) \) is continuous.
For the example of the $n = 7$ case, the equations designated by formula (3) are as follows:

\begin{align*}
i = 1 \quad &\Rightarrow \quad \Delta x_2 s_1 + 2(\Delta x_1 + \Delta x_2)s_2 + \Delta x_1 s_3 = 3(\Delta x_2 y'_1 + \Delta x_1 y'_2) \\
i = 2 \quad &\Rightarrow \quad \Delta x_3 s_2 + 2(\Delta x_2 + \Delta x_3)s_3 + \Delta x_2 s_4 = 3(\Delta x_3 y'_2 + \Delta x_2 y'_3) \\
&\vdots \quad \Rightarrow \quad \vdots \\
i = 5 \quad &\Rightarrow \quad \Delta x_6 s_5 + 2(\Delta x_5 + \Delta x_6)s_6 + \Delta x_5 s_7 = 3(\Delta x_6 y'_5 + \Delta x_5 y'_6)
\end{align*}

Notice that we have 5 constraints and 7 parameters and therefore two degrees of freedom.
If we move the two parameters ($s_1$ and $s_7$) to the right hand side and assemble the results in matrix-vector form, then we obtain a 5-by-5 linear system

$$Ts(2:6) = T\begin{bmatrix} s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \begin{bmatrix} 3(\Delta x_2y_1' + \Delta x_1y_2') - \Delta x_2s_1 \\ 3(\Delta x_3y_2' + \Delta x_2y_3') \\ 3(\Delta x_4y_3' + \Delta x_3y_4') \\ 3(\Delta x_5y_4' + \Delta x_4y_5') \\ 3(\Delta x_6y_5' + \Delta x_5y_6') - \Delta x_5s_7 \end{bmatrix} = r,$$

where

$$T = \begin{bmatrix} 2(\Delta x_1 + \Delta x_2) & \Delta x_1 & 0 & 0 & 0 \\ \Delta x_3 & 2(\Delta x_2 + \Delta x_3) & \Delta x_2 & 0 & 0 \\ 0 & \Delta x_4 & 2(\Delta x_3 + \Delta x_4) & \Delta x_3 & 0 \\ 0 & 0 & \Delta x_5 & 2(\Delta x_4 + \Delta x_5) & \Delta x_4 \\ 0 & 0 & 0 & \Delta x_6 & 2(\Delta x_5 + \Delta x_6) \end{bmatrix}$$
Matrices like this that are zero everywhere except on the diagonal, subdiagonal, and superdiagonal are said to be tridiagonal.

Different choices for the end slopes $s_1$ and $s_n$ yield different cubic spline interpolants.

Having defined the end slopes, the interior slopes $s(2 : n - 1)$ are determined by solving an $(n - 2) \times (n - 2)$ linear system.
In each case we consider here, the matrix of coefficients looks like

\[
T = \begin{bmatrix}
  t_1 & t_2 & 0 & \cdots & 0 \\
\Delta x_3 & 2(\Delta x_2 + \Delta x_3) & \Delta x_2 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \Delta x_{n-2} & 2(\Delta x_{n-3} + \Delta x_{n-2}) & \Delta x_{n-3} & t_{n-2,n-3} \\
0 & \cdots & 0 & t_{n-2,n-2} & t_{n-2,n-2}
\end{bmatrix},
\]

while the right hand side has the form

\[
\begin{bmatrix}
  r_1 \\
  3(\Delta x_3 y'_2 + \Delta x_2 y'_3) \\
  \vdots \\
  3(\Delta x_{n-2} y'_{n-3} + \Delta x_{n-3} y'_{n-2}) \\
r_{n-2}
\end{bmatrix}.
\]
In the above matrices, the values $t_{11}$, $t_{12}$, and $r_1$ depend on how $s_1$ is chosen, and the values $t_{n-2,n-3}$, $t_{n-2,n-2}$, and $r_{n-2}$ depend on how $s_n$ is defined. Moreover, the matrix $T$ can be shown to be *nonsingular*.

Now we shall set up the first and last rows of $T$ and the first and last components of $r$, which depend on the end conditions $s_1$ and $s_n$.

The Complete Spline is obtained by setting $s_1 = \mu_L$ and $s_n = \mu_R$, where $\mu_L$ and $\mu_R$ are real.
With these constraints, setting $i = 1$ and $i = n - 2$ in (3) gives

$$\Delta x_2 \mu_L + 2(\Delta x_1 + \Delta x_2)s_2 + \Delta x_1 s_3 = 3(\Delta x_2 y'_1 + \Delta x_1 y'_2)$$

$$\Delta x_{n-1}s_{n-2} + 2(\Delta x_{n-2} + \Delta x_{n-1})s_{n-1} + \Delta x_{n-2} \mu_R = 3(\Delta x_{n-1} y'_{n-2} + \Delta x_{n-2} y'_{n-1})$$

And so the first and last equations becomes

$$2(\Delta x_1 + \Delta x_2)s_2 + \Delta x_1 s_3 = 3(\Delta x_2 y'_1 + \Delta x_1 y'_2) - \Delta x_2 \mu_L$$

$$\Delta x_{n-1}s_{n-2} + 2(\Delta x_{n-2} + \Delta x_{n-1})s_{n-1} = 3(\Delta x_{n-1} y'_{n-2} + \Delta x_{n-2} y'_{n-1}) - \Delta x_{n-2} \mu_R$$
Instead of prescribing the slope of the spline at the endpoints, we can give the value of its second derivative. In particular, if we insist that $\mu_L = q''_1(x_1)$, then from Eq.(2) we obtain

$$\mu_L = 2 \frac{y'_1 - s_1}{\Delta x_1} - 2 \frac{s_1 + s_2 - 2y'_1}{\Delta x_1},$$

from which we conclude that

$$s_1 = \frac{1}{2} \left( 3y'_1 - s_2 - \frac{\mu_L}{2} \Delta x_1 \right).$$

Substituting this results into the $i = 1$ case of Eq.(1) and rearranging, we obtain

$$(2\Delta x_1 + 1.5\Delta x_2)s_2 + \Delta x_1 s_3 = 1.5\Delta x_2 y'_1 + 3\Delta x_1 y'_2 + \frac{\mu_L}{4} \Delta x_1 \Delta x_2.$$
The Natural Spline (Continuing)

Likewise, by setting $\mu_R = q''_{n-1}(x_n)$ then Eq.(2) implies

$$\mu_R = 2 \frac{y'_{n-1} - s_{n-1}}{\Delta x_{n-1}} + 4 \frac{s_{n-1} + s_n - 2y'_{n-1}}{\Delta x_{n-1}},$$

from which we conclude that

$$s_n = \frac{1}{2} \left( 3y'_{n-1} - s_{n-1} - \frac{\mu_R}{2} \Delta x_{n-1} \right)$$

Substituting this results into the $i = n - 2$ case of Eq.(1) and rearranging, we obtain

$$\Delta x_{n-1} s_{n-2} + (1.5\Delta x_{n-2} + 2\Delta x_{n-1}) s_{n-1} = 3\Delta x_{n-1} y'_{n-2} + 1.5\Delta x_{n-2} y'_{n-1} - \frac{\mu_R}{4} \Delta x_{n-2} \Delta x_{n-1}.$$  

Thus, the setting up of $T$ and $r$ and the resolution of $s$ are complete.

If $\mu_L = \mu_R = 0$, then the resulting spline is called the natural spline.
The Not-a-Knot Spline

- The method for prescribing the end conditions is appropriate if *no endpoint derivative information is available*. One idea is to ensure *third derivative continuity at both* $x_2$ and $x_{n-1}$, which is called the *not-a-knot spline*.

- Note from Eq.(1) that

$$q''''_i(x) = 6 \frac{s_i + s_{i+1} - 2y'_i}{(\Delta x_i)^2},$$

and so $q''''_1(x) = q''''_2(x)$ says that

$$\frac{s_1 + s_2 - 2y'_1}{(\Delta x_1)^2} = \frac{s_2 + s_3 - 2y'_2}{(\Delta x_2)^2}.$$
The Not-a-Knot Spline (Continuing)

- It follows that this will be the case if we set
  
  \[ s_1 = -s_2 + 2y'_1 + \left( \frac{\Delta x_1}{\Delta x_2} \right)^2 (s_2 + s_3 - 2y'_2). \]

  As a result of making the \textit{third derivative continuous} at \( x_2 \), the cubics \( q_1(x) \) and \( q_2(x) \) are identical.

- Likewise, if \( q''''_{n-2}(x_{n-1}) = q''''_{n-1}(x_{n-1}) \) then
  
  \[ \frac{s_{n-2} + s_{n-1} - 2y'_{n-2}}{(\Delta x_{n-2})^2} = \frac{s_{n-1} + s_n - 2y'_{n-1}}{(\Delta x_{n-1})^2}. \]

  It follows that this will be the case if we set
  
  \[ s_n = -s_{n-1} + 2y'_{n-1} + \left( \frac{\Delta x_{n-1}}{\Delta x_{n-2}} \right)^2 (s_{n-2} + s_{n-1} - 2y'_{n-2}). \]
Error bounds for the cubic spline interpolant are complicated to derive. The bounds are not good if the end conditions are *improperly chosen*. However, if the end values are *properly chosen* or if the *not-a-knot approach* is used, then the error bounds has the form $M_4 \bar{h}^4$ where

$$\bar{h} = \max_{1 \leq i \leq n} |x_{i+1} - x_i|, \quad |f^{(4)}(x)| \leq M_4, \quad x \in [\alpha, \beta].$$

The *SplineErr.m* confirms the error bounds for the case of an ’easy’ $f(x)$.
The Cubic Spline Interpolant

- The function `CubicSpline.m` can be used to construct the cubic spline interpolant with any of the three aforementioned types of end conditions.

- Notice that a two-argument call is all that is required to produce the **not-a-knot spline**. Other cases, we must give 3 more arguments—derivative, $\mu_L$, and $\mu_R$. We show this by the script `CubicSplineTest.m`. 
The MATLAB function `Spline.m` can be used to compute not-a-knot spline Interpolants. It can be called with either 2 or 3 arguments.

A two-argument call to `Spline.m` returns what is called *pp-representation* of the spline (piecewise polynomial).

Example: To interpolate the function $f(x) = \arctan(x)$ across the interval $[-5, 5]$ with an $n = 9$ not-a-knot spline.
The function `ppval.m` can be used to evaluate a piecewise polynomial at the given values.

The function `unmkpp.m` extracts the degree \((k - 1)\) of spline, the number of subintervals \(L\), and their coefficients of local polynomials.

The function `mkpp.m` Makes piecewise polynomial with input the breaks and coefficients (see the script `ShowSplineTools.m` and help).