Chapter 4

Polynomial Interpolation and Splines Fitting

4.1 Definition of Interpolation

Consider a set D_n of $n+1$ data points in the (x, y) plane:

(4.1) $$D_n = \{(x_i, y_i)|\ i = 0, 1 ..., n;\ n \in \mathbb{N} \text{ with } x_i \neq x_j \text{ for } i \neq j\}.$$

We assume that D_n represents partially the values of a function $y = f(x)$, i.e.:

(4.2) $$f(x_i) = y_i\ \forall\ i = 0, 1 ..., n$$

Our basic objective in this chapter is to construct a continuous function $r(x)$ that "represents" $f(x)$ (or the empirical law $f(x)$ behind the set of data D_n). Thus $r(x)$ would represent $f(x)$ for all x, in particular for $x \notin$ the set of **nodes** $\{x_0, x_1, ..., x_n\}$. Such a function $r(x)$ is said to **interpolate** the set of data D_n if it satisfies the **interpolation conditions**:

(4.3) $$r(x_i) = y_i\ \forall\ i = 0, 1 ..., n$$

i.e. fitting the function $f(x)$ at the nodes $\{x_i\}$, .
Several kinds of interpolation may be considered by choosing $r(x)$ to be a polynomial function, a rational function or even a trigonometric one. The most natural is to consider polynomial or piecewise polynomial interpolation (spline functions) , as polynomial functions are the simplest in reproducing the basic arithmetic (floating-point) operations of addition, subtraction,

multiplication and division $\{+, -, \times, \div\}$. Consistently, we only analyze in this chapter **polynomial** and **spline interpolations**. Such type of interpolation is referred to as **Lagrange interpolation**.

4.2 General Lagrange Polynomial Interpolation

For simplicity, we assume that the set of data D_n is given as a natural increasing sequence of x-values, i.e.

$$x_0 < x_1 < ... < x_n.$$

Let also $h_i = x_i - x_{i-1}$, $\forall i \geq 1$. We state now the basic Lagrange Interpolation theorem.

Theorem 4.1. *There exists a unique polynomial of degree less than or equal to n:*

$$p_n(x) = p_{01...n}(x)$$

interpolating D_n, i.e such that $p_n(x_i) = y_i$, $\forall i = 0, 1, ..., n$.

Proof. The proof of this theorem is based on the **Lagrangian cardinal basis** associated with D_n that is given by:

$$L_n = \{l_i(x) : 0 \leq i \leq n\}$$

where the **cardinal functions** l_i are special polynomials of degree exactly n in $\mathbb{P}_n$ ($\mathbb{P}_n$ being the set of all polynomials of degree less than or equal to n). They are defined as follows, $\forall i = 0, ..., n$:
(4.4)

$$l_i(x) = \frac{\prod_{0 \leq j \neq i \leq n}(x - x_j)}{\prod_{0 \leq j \neq i \leq n}(x_i - x_j)} = \frac{(x - x_0)(x - x_1)...(x - x_{i-1})(x - x_{i+1})...(x - x_n)}{(x_i - x_0)(x_i - x_1)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_n)}$$

Once the cardinal functions (4.4) are available, we can interpolate any function f using **Lagrange form of the interpolation polynomial**:

$$p_{01...n}(x) = \sum_{i=0}^{n} l_i(x) f(x_i). \tag{4.5}$$

Obviously, the following properties are satisfied by a Lagrangian basis function, $\forall i, j = 0, 1, \ldots, n$:

- $l_i(x_j) = \delta_{ij} = \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases}$

- $p_{01\ldots n}(x_i) = y_i$

The definition of the Lagrange polynomial above is enough to establish the existence part of Theorem 4.1.
As for obtaining uniqueness of such a polynomial $p_{01\ldots n}$, we proceed by contradiction by supposing the existence of another polynomial $q(x) \in \mathbb{P}_n$, claiming to accomplish what $p(x)$ does; that is $q(x)$ satisfies as well the interpolation conditions $q(x_i) = y_i$ for $0 \le i \le n$. The polynomial:

$$(p_{01\ldots n}(x) - q(x))$$

is then of degree at most n, and takes on the value 0 at all nodes $x_0,\ x_1,\ \ldots,\ x_n$.
Recall however that a non-zero polynomial of degree n can have at most n roots, implying that $(p_{01\ldots n}(x) - q(x)) = 0$. One concludes therefore that $p_{01\ldots n}(x) = q(x)\,\forall x$, which establishes the uniqueness of $p_{01\ldots n}(x)$. ■

Remark 4.1. *It is obvious from equation (4.5) that:*

$$p_{01\ldots n}(x) = p_{i_0 i_1 \ldots i_n}(x)$$

for any permutation $\{i_0, i_1, ..., i_n\}$ *of the set of indices* $\{0, 1, ..., n\}$.

Example 4.1. *Write out the cardinal functions and the corresponding Lagrange interpolating polynomial based on the following data:*

$$D_2 = \{\ (1/4, -1),\ (1/3, 2),\ (1, 7)\}$$

Using equation (4.4), we have:

$$l_0(x) = \frac{(x-\frac{1}{3})(x-1)}{(\frac{1}{4}-\frac{1}{3})(\frac{1}{4}-1)} = 16(x-\frac{1}{3})(x-1)$$

$$l_1(x) = \frac{(x-\frac{1}{4})(x-1)}{(\frac{1}{3}-\frac{1}{4})(\frac{1}{3}-1)} = -18(x-\frac{1}{4})(x-1)$$

$$l_2(x) = \frac{(x-\frac{1}{3})(x-\frac{1}{4})}{(1-\frac{1}{3})(1-\frac{1}{4})} = 2(x-\frac{1}{3})(x-\frac{1}{4})$$

The interpolating polynomial in Lagrange's form is therefore given by:

$$p_{012}(x) = -36(x-\frac{1}{4})(x-1) - 16(x-\frac{1}{3})(x-1) + 14(x-\frac{1}{3})(x-\frac{1}{4}) = -38x^2 + \frac{349}{6}x - \frac{79}{6}$$

This form of the polynomial might be useful in computing $f(x)$ in the vicinity of the nodes 1/3, 1/4, 1.

Example 4.2. *Consider the following table of data associated with the function $f(x) = \ln(x)$.*

i	$\mathbf{x_i}$	$\mathbf{y_i}$
0	1.0	0
1	1.5	0.17609
2	2.0	0.30103
3	3.0	0.47712
4	3.5	0.54407
5	4.0	0.60206

Use Lagrange polynomials of orders 1 then 2 to approximate $f(1.2)$, noting that the exact value is $\ln(1.2) = 0.0791812460476480$.

- $\underline{\text{Linear Interpolation}}$ based on the points $\{x_0,\ x_1\}= \{1.0,\ 1.5\}$, where $l_0(.)$ and $l_1(.) \in \mathbb{P}_1$. Using (4.5), one has:

$$p_{01}(x) = y_0l_0(x) + y_1l_1(x) = 0\frac{x-1.5}{1.0-1.5} + 0.17609\frac{x-1.0}{1.5-1.0}$$

  Thus $p_{01}(1.2) = 0.070436$, and the relative error iin this approximation is 6.136716×10^{-1}

- $\underline{\text{Quadratic interpolation}}$ based on the points $\{1.0, 1.5,\ 2.0\}$, where $l_0(.)$, $l_1(.)$ and $l_2(.) \in \mathbb{P}_2$.

$$p_{012}(x) = y_0l_0(x) + y_1l_1(x) + y_2l_2(x) = 0\frac{(x-1.5)(x-2)}{(1.0-1.5)(1.0-2)}+$$

$$+0.17609\frac{(x-1.0)(x-2)}{(1.5-1.0)(1.5-2)} + 0.30103\frac{(x-1.0)(x-1.5)}{(2-1.0)(2-1.5)}$$

  Thus $p_{012}(1.2) = 0.076574$, and the relative error is now 3.292757×10^{-2}.

Remark 4.2. *Note that Lagrange's formula is not computationally practical in the sense that computing $p_{01...k}(x)$ with $k < n$, cannot be obtained from $p_{01...k-1}(x)$. The cardinal functions of the latter are polynomials of degree exactly $k-1$ in $\mathbb{P}_{k-1}$, while those of the former are polynomials of degree exactly k in $\mathbb{P}_k$. Thus, after computing the Lagrange cardinal functions for $p_{01...k-1}(x)$, one has to compute a totally distinct set of cardinal functions for $p_{01...k}(x)$.*

This motivates looking for recurrence formulae to the Lagrange interpolating polynomial.

4.3 Recurrence Formulae

These recurrence formulae are obtained through relations between two consecutive-order interpolation polynomials, specifically and for $k \geq 1$:

- Consider first $p_{012..k} \in \mathbb{P}_k$ and $p_{012..k-1} \in \mathbb{P}_{k-1}$. As

  $$p_{012..k}(x_i) - p_{012..k-1}(x_i) = 0, \, \forall\, i = 0, 1, ..., k-1$$

  This implies that:

  $$p_{012..k}(x) - p_{012..k-1}(x) = C(x-x_0)(x-x_1)...(x-x_{k-1}) \tag{4.6}$$

  Note that C is a constant as the right hand side polynomial is exactly of degree k.

- In a similar way considering now $p_{012..k} \in \mathbb{P}_k$ and $p_{12..k} \in \mathbb{P}_{k-1}$, one obtains:

  $$p_{012..k}(x) - p_{12..k}(x) = C^{'}(x-x_1)...(x-x_{k-1})(x-x_k). \tag{4.7}$$

  It is clear that $C = C^{'}$ as both constants are the coefficient of x^k in $p_{012..k}$.

(4.6) and (4.7) constitute the basis for Neville's and Newton's recurrence formulae, as shown hereafter.

4.3.1 Neville's Formula

Given that $C = C^{'}$, the algebraic operation:

$$(x - x_k)\times(4.6) \text{ - } (x - x_0)\times(4.7)$$

yields:

$$(x_0 - x_k)p_{01...k-1\,k}(x) = (x - x_k)p_{01...k-1}(x) - (x - x_0)p_{12...k-1\,k}(x).$$

Hence one reaches Neville's formula (also called Aitken-Neville's), given by:

$$p_{01...k-1\,k}(x) = \frac{(x-x_0)p_{12...k-1\,k}(x) - (x-x_k)p_{012...k-1}(x)}{x_k - x_0}, \; k \geq 1. \tag{4.8}$$

A more general **Neville's recurrence formulae** can be concluded. Specifically, for any $i \in \{0, 1, ..., n\}$. :

- **Base statement**: $p_i(x) = y_i$, $i = 0, 1, ..., n$
- **Recurrence statement**:
  (4.9)
  $$p_{i\,i+1\,...\,i+k}(x) = \frac{(x - x_i)p_{i+1\,i+2...\,i+k}(x) - (x - x_{i+k})p_{i\,i+1\,...\,i+k-1}(x)}{x_{i+k} - x_i},$$
  with
  $$0 \leq i < i + k \leq n.$$

Based on the set of data D_n in (5.1) and using the formulas above repeatedly, we can create an array of interpolating polynomials $\in \mathbb{P}_n$, where each successive polynomial can be determined from 2 adjacent polynomials in the previous column:

i	x_i	$p_i(x)$	$p_{i,i+1}(x)$	$p_{i,i+1,i+2}(x)$	$\cdots$	$p_{i,i+1,...,i+n}(x)$
0	x_0	$p_0(x)$				
1	x_1	$p_1(x)$	$p_{0,1}(x)$			
2	x_2	$p_2(x)$	$p_{1,2}(x)$	$p_{0,1,2}(x)$		
3	x_3	$p_3(x)$	$p_{2,3}(x)$	$p_{1,2,3}(x)$		
4	x_4	$p_4(x)$	$p_{3,4}(x)$	$p_{2,3,4}(x)$	...	
$\vdots$	...	...	...	...	...	...
n	x_n	$p_n(x)$	$p_{n-1,n}(x)$	$p_{n-2,n-1,n}(x)$	...	$p_{0,1,...,n}(x)$

For example,

$$p_{01}(x) = \frac{(x - x_0)p_1(x) - (x - x_1)p_0(x)}{x_1 - x_0}$$
$$p_{123} = \frac{(x - x_1)p_{23}(x) - (x - x_3)p_{12}(x)}{x_3 - x_1}$$

Remark 4.3. *Neville's recurrence expressions of the interpolating polynomial can be easily programmed. The consequent algorithms can be written either in a recursive or iterative form.*

In what follows, we write a recursive algorithm for Neville's formula leaving it as an exercise to transform it into an iterative one.

Algorithm 4.1. Algorithm for Neville's formula(Recursive version)

```
function [ z ]= Neville(x, y, s)
% Input data vectors x=[x1,x2,...,xk]  and y=[y1,y2,...,yk]
% s : value (or vector) at which we seek the interpolation
% Output z=p_{12...k}(s)=p(s)
 k = length(x);
if k=1
    z=y;
% z1=p_{12...(k-1)}(s) ;  z2=p_{2...k}(s)
% z= [(s-x1)*z2 - (s-xk)*z1] / (xk-x1)
else
   z1= Neville(x(1:k-1), y(1:k-1) , s);
   z2= Neville(x(2:k), y(2:k) , s);
   z= ((s-x(1))*z2 - (s-x(k))*z1)/(x(k)-x(1));
end
```

4.3.2 Newton's form for the interpolation polynomial

As for Neville's formula, we proceed with (4.6) by rewriting it in a more general recurrence form as follows:

$$p_{i\,i+1...,i+k}(x) = p_{i\,i+1...i+k-1}(x) + C(x - x_i)...(x - x_{i+k-1}), \tag{4.10}$$

with

$$0 \le i < i + k \le n.$$

Newton's formula is obtained by determining a proper expression for the constant C as a function of the data $D_k = \{(x_i, y_i)|i = 0, 1, ..., k\}$. Note that such constant can be computed by setting $x = x_{i+k}$ in (4.10), so that:

$$y_{i+k} = p_{i\,i+1...i+k-1}(x_{i+k}) + C(x_{i+k} - x_i)...(x_{i+k} - x_{i+k-1}),$$

and therefore:

$$C = C(x_i, x_{i+1}, ..., x_{i+k}; y_i, y_{i+1}, ..., y_{i+k}) = \frac{y_{i+k} - p_{i\,i+1...i+k-1}(x_{i+k})}{(x_{i+k} - x_i)...(x_{i+k} - x_{i+k-1})}.$$

For $k = 1$, this gives:

$$C = C(x_i, x_{i+1}; y_i, y_{i+1}) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}. \tag{4.11}$$

Define then

$$[x_i, x_{i+1}] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

as the first order divided difference associated with $\{x_i, x_{i+1}\}$, so that (4.10) is expressed as follows:

$$p_{i,i+1}(x) = p_i(x) + [x_i, x_{i+1}](x - x_i) \tag{4.12}$$

which is Newton's formula of order 1. More generally, we may define divided differences of any order $k \geq 1$, through a recurrence process as follows:

Definition 4.1. *Given the set of data*

$$D_n = \{(x_i, y_i) | i = 0, 1, ..., n\},\ x_i \neq x_j \text{ for } i \neq j.$$

Let $[x_i] = y_i$, $i = 0, 1, ..., n$. Then, for $0 \leq i < i + k \leq n$, the k^{th} order divided difference is given through the recurrence formula:

$$[x_i, x_{i+1}, ..., x_{i+k}] = \frac{[x_{i+1}, ..., x_{i+k}] - [x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+k} - x_i}. \tag{4.13}$$

Consequently, we prove that the constant C in (4.10) is a k^{th} order divided difference. This is done in the following proposition.

Theorem 4.2. *Let $0 \leq i < i + k \leq n$. Let*

$$p_{i\,i+1 ... i+k}(x) = p_{i\,i+1 ... i+k-1}(x) + C(x - x_i)...(x - x_{i+k-1}),$$

is the interpolating polynomial based on the nodes $\{x_i, ..., x_{i+k}\}$, as defined in (4.6). Then, the constant C is the k^{th} order divided difference

$$C = [x_i, x_{i+1}, ..., x_{i+k}] = \frac{[x_{i+1}, ..., x_{i+k}] - [x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+k} - x_i}$$

Proof. To obtain this result we use a mathematical induction process on k. Clearly, (4.12) indicates that the result is true for $k = 1$.
Assuming now that the proposition is correct for all $j \leq k-1$ with $i+j < n$, then, one writes on the basis of the induction hypothesis for $j = k - 1$, successively:

$$p_{i...i+k-1}(x) = p_{i...i+k-2}(x) + [x_i, x_{i+1}, ...x_{i+k-1}](x - x_i)...(x - x_{i+k-2})$$

where $[x_i, x_{i+1}, ...x_{i+k-1}]$ is the coefficient of x^{k-1} in the polynomial $p_{i...i+k-1}(x)$

and

$$p_{i+1\ldots i+k}(x) = p_{i+1,\ldots,i+k-1}(x) + [x_{i+1},\ldots,x_{i+k}](x-x_{i+1})\ldots(x-x_{i+k-1})$$

where $[x_{i+1},\ldots,x_{i+k}]$ is the coefficient of x^{k-1} in the polynomial $p_{i+1\ldots i+k}(x)$. Using now Neville's formula, one has:

$$p_{i\ldots i+k}(x) = \frac{(x-x_i)p_{i+1\ldots i+k}(x) - (x-x_{i+k})p_{i\ldots i+k-1}(x)}{x_{i+k}-x_i}.$$

By equating the coefficients of x^k on both sides of this identity one has:

$$C = \frac{[x_{i+1},\ldots,x_{i+k-1},x_{i+k}] - [x_i,x_{i+1},\ldots,,x_{i+k-1}]}{x_{i+k}-x_i},$$

which is the targeted result of the theorem. ■

As a consequence of this theorem, we may write now Newton's formula for Lagrange interpolating polynomial as follows: for $i < i+k \le n$:
(4.14)

$$p_{i\,i+1\ldots i+k-1\,i+k}(x) = y_i + [x_i,x_{i+1}](x-x_i) + \ldots + [x_i,x_{i+1},\ldots,x_{i+k}](x-x_i)\ldots(x-x_{i+k-1}).$$

or equivalently as:
(4.15)

$$p_{i\,i+1\ldots i+k-1\,i+k}(x) = y_i + \sum_{j=1}^{k}[x_i,\ldots,x_{i+j}](x-x_i)\ldots(x-x_{i+j-1}) = \sum_{j=0}^{k}[x_i,\ldots,x_{i+j}]\prod_{j=0}^{i-1}(x-x_{i+j})$$

More specifically:

$$p_{01\ldots n}(x) = y_0 + [x_0,x_1](x-x_0) + [x_0,x_1,x_2](x-x_0)(x-x_1) + \ldots$$
$$\ldots + [x_0,x_1,x_2,\ldots,x_n](x-x_0)(x-x_1)\ldots(x-x_{n-1})$$

Remark 4.4. *Note that, as expressed in (4.10), Newton's formula of the interpolating polynomial is built up in steps, in the sense that once $p_{i\,i+1\ldots i+k-1}(x)$ is found reproducing part of the data, determining $p_{i\,i+1\ldots i+k}(x)$, necessitates the computation of one new divide difference coefficient only.*

4.3.3 Construction of Divided Differences and Implementation of Newton's formula

Let $\{i_0,\, i_1, \ldots, i_k\}$ of be any permutation of the set of integers $\{i,\, i+1, \ldots, i+k\}$. Based on the uniqueness property of the interpolating polynomials:

$$p_{i\,i+1 \ldots i+k-1\, i+k}(x) = p_{i_0\, i_1 \ldots i_{k-1}\, i_k}(x)$$

and consequently the k^{th} order divided differences $[x_i, x_{i+1}, ..., x_{i+k}]$ and $[x_{i_0}, x_{i_1} ..., x_{i_{k-1}}, x_{i_k}]$ representing respectively the (same) coefficient of x^k in the two polynomials above, are equal. This leads to the following invariance property satisfied by divided differences:

Theorem 4.3. *Let* $\{i_0,\, i_1, \ldots, i_k\}$ *of be any permutation of the set of integers* $\{i,\, i+1, \ldots, i+k\}$. *Then:*

$$[x_i,\, x_{i+1}, \ldots, x_{i+k}] = [x_{i_0},\, x_{i_1}, \ldots, x_{i_k}]$$

Obviously, use of Newton's formula necessitates the computation of divided differences. As such, constructing divided differences tables associated with a set of data $D_n = \{(x_i, y_i)|i = 0, 1, ..., n\}$ is a preliminary step to any implementation of Newton's formula. The figure below displays a divided difference table for the case $n = 5$.

i	$\mathbf{x_i}$	$\mathbf{y_i}$	$[\cdot,\cdot]$	$[\cdot,\cdot,\cdot]$	$[\cdot,\cdot,\cdot,\cdot]$	$[\cdot,\cdot,\cdot,\cdot,\cdot]$	$[\cdot,\cdot,\cdot,\cdot,\cdot,\cdot]$
0	x_0	y_0					
			$[x_0, x_1]$				
1	x_1	y_1		$[x_0, x_1, x_2]$			
			$[x_1, x_2]$		$[x_0, x_1, x_2, x_3]$		
2	x_2	y_2		$[x_1, x_2, x_3]$		$[x_0, x_1, x_2, x_3, x_4]$	
			$[x_2, x_3]$		$[x_1, x_2, x_3, x_4]$		$[x_0, x_1, x_2, x_3, x_4, x_5]$
3	x_3	y_3		$[x_2, x_3, x_4]$		$[x_1, x_2, x_3, x_4, x_5]$	
			$[x_3, x_4]$		$[x_2, x_3, x_4, x_5]$		
4	x_4	y_4		$[x_3, x_4, x_5]$			
			$[x_4, x_5]$				
5	x_5	y_5					

The following `MATLAB` code takes as input 2 vectors x and y of equal length and returns the Divided Difference table of the first (n-1)-order divided differences, as a lower triangular matrix, using the `MATLAB diff` operator.

Algorithm 4.2. Constructing a divided difference table

```
function D = DivDiffTable(x,y)
% D is a lower Triangular matrix
% If   x=[x(1),x(2),...,x(n)]  is a vector of length n, then :
%  diff(x)=[(x(2)-x(1)), (x(3)-x(2)),....,(x(n)-x(n-1))] is a vector of length (n-1)
n=length(x) ;
m=length(y) ;
if m==n
   D=zeros(n,n) ;
   D(1:n, 1) = y(1:n) ;
   Y= D(1:n, 1)  ;
     for j=2: n
        V1=x(1:n-j+1) ; V2=x(j:n) ;
        D(j:n, j)= (diff(Y) ./ (V2-V1)' ) ;
       Y=D(j:n, j) ;
      end
end
```

Example 4.3. *Create the Divided Difference Table based on the set of data of Example 4.2 representing the function $f(x) = \ln(x)$:*

$$D_5 = \{(1,0)\,,\,(1.5,0.17609)\,,\,(2.0,0.30103)\,,\,(3,0.47712)\,,\,(3.5,0.54407)\,,\,(4,0.60206)\}$$

i	$\mathbf{x_i}$	$\mathbf{y_i}$	$[.,.]$	$[.,.,.]$	$[.,.,.,.]$	$[.,.,.,.,.]$	$[.,.,.,.,.,.]$
0	1.0	0					
			0.35218				
1	1.5	0.17609		−0.1023			
			0.24988		0.02655		
2	2.0	0.30103		−0.0492		−0.006404	
			0.17609		0.01054		0.001411
3	3.0	0.47712		−0.02813		−0.002172	
			0.13390		0.00511		
4	3.5	0.54407		−0.01792			
			0.11598				
5	4.0	0.60206					

Let us consider now approximations of $f(x)$ for values of x first at the top of the table, for example $x = 1.2$, then at the middle of the table, as $x = 2.5$.

(Note that in general, one can prove that the approximation-error is smaller when x is centered with respect to the nodes).

1. The 1st interpolating polynomials are successively as follows:
   • $p_{01}(x) = 0.35218(x-1)$,
   • $p_{012}(x) = p_{01}(x) - 0.1023(x-1)(x-1.5)$,
   • $p_{0123}(x) = p_{012}(x) + 0.02655(x-1)(x-1.5)(x-2)$,
   • $p_{01234}(x) = p_{0123}(x) - 0.006404(x-1)(x-1.5)(x-2)(x-3)$,
   • $p_{012345}(x) = p_{01234}(x) + 0.001411(x-1)(x-1.5)(x-2)(x-3)(x-3.5)$.

   As a result, approximations to $\ln(1.2) = 0.0791812460476248$ come as follows:

$\mathbf{p_{\ldots}(1.2)}$	Value	Relative error
$p_{01}(x)$	0.070436	1.10446×10^{-1}
$p_{012}(x)$	0.076574	3.2928×10^{-2}
$p_{0123}(x)$	0.0778484	1.6833×10^{-2}
$p_{01234}(x$	0.07840171	9.845×10^{-3}
$p_{012345}(x)$	0.0786821	6.30384×10^{-3}

2. To get approximations to $f(2.5)$,(using Theorem 4.3), we obtain successively linear, quadratic and cubic polynomials as follows:
   • $p_{23}(x) = y_2 + [x_2, x_3](x - x_2) = 0.30103 + 0.17609(x-2)$
   • $p_{231}(x) = p_{23}(x) + [x_2, x_3, x_1](x-x_2)(x-x_3)$
   $= p_{23}(x) + [x_1, x_2, x_3](x-x_2)(x-x_3) = p_{23}(x) - 0.0492(x-2)(x-3)$
   • $p_{2314}(x) = p_{231}(x) + [x_2, x_3, x_1, x_4](x-x_2)(x-x_3)(x-x_1)$
   $= p_{231}(x) + [x_1, x_2, x_3, x_4](x-x_2)(x-x_3)(x-x_1) = p_{231}(x) +$
   $0.01054(x-2)(x-3)(x-1.5)$
   • $p_{2310}(x) = p_{231}(x) + [x_2, x_3, x_1, x_0](x-x_2)(x-x_3)(x-x_1)$
   $= p_{231}(x) + [x_0, x_1, x_2, x_3](x-x_2)(x-x_3)(x-x_1) = p_{231}(x) +$
   $0.02655(x-2)(x-3)(x-1.5)$

   Another alternative starting with $p_{23}(x)$ would be:
   • $p_{234}(x) = p_{23}(x) + [x_2, x_3, x_4](x-x_2)(x-x_3) = p_{23}(x) -$
   $0.02813(x-2)(x-3)$
   • $p_{2345}(x) = p_{234}(x) + [x_2, x_3, x_4, x_5](x-x_2)(x-x_3)(x-x_4)p_{234}(x) +$
   $0.00511(x-2)(x-3)(x-3.5)$
   • $p_{2341}(x) = p_{234}(x) + [x_2, x_3, x_4, x_1](x-x_2)(x-x_3)(x-x_4)$
   $= p_{234}(x) + [x_1, x_2, x_3, x_4](x-x_2)(x-x_3)(x-x_4) = p_{234}(x) - 0.002172(x-$
   $2)(x-3)(x-3.5)$

This process can be carried through to obtain higher order interpolation polynomials. The following results are obtained for the approximation of $\ln(2.5) = 0.39794001$:

$\mathbf{p}_{\dots}(\mathbf{2.5})$	Value	Relative error
$p_{23}(x)$	0.389075	2.227725×10^{-2}
$p_{234}(x)$	0.3961067	4.6070814×10^{-3}
$p_{231}(x)$	0.4013733	$8.62774435 \times 10^{-3}$
$p_{2345}(x)$	0.3973825	1.4009867×10^{-3}
$p_{2341}(x)$	0.39874	2.0103315×10^{-3}

Hence, it appears clear that increasing the degree of the interpolation polynomial does not improve much the approximation of the exact value of $f(x)$.

We may now write an algorithm that implements Newton's formula using Algorithm 4.2: `DivDiffTable(x,y)`.

Algorithm 4.3. Program for Newton's formula

```
function p=NewtonForm(x,y,X)
%Input: two equally sized vectors x and y of length k
%          One vector X of length n
%Output: p(X) based on Newton interpolation formula on the data (x,y)
D=DivDiffTable(x,y);
k=length(x);%(equal to length of y)
n=length(X);X=X(:);
term=ones(n,1);
p=zeros(n,1);
for i=1:k
   p=p+D(i,i)*term;
   term=term.*(X-x(i));
end
```

To conclude on recurrence formulae for the Lagrange interpolation polynomial, a rule of thumb would be to use Neville's formula in case of computer implementation as it takes only one algorithm to program (Algorithm 4.1). On the other hand, Newton's formula requires writing 2 programs: one for divided differences (Algorithm 4.2) before developing Algorithm 4.3 for a straightforward evaluation of the interpolation polynomial.

4.4 Equally spaced Data: Difference Operators

Consider now the set of data D_n with equidistant $x-$nodes, i.e.

$$x_{i+1} - x_i = h, \; \forall i = 0, 1, ..., n-1.$$

In this case, we can compute divided differences associated with D_n by using the **"difference functions"** or **"difference operators"** , based on the $y-$data only. Specifically, we make the following definitions:

Definition 4.1. *Let $Y = [y_0, y_1, ..., y_n]$, then:*

1. $\Delta^1 Y = [\Delta y_0, \Delta y_1, ..., \Delta y_{n-1}]$ *is the vector of* n *first order differences associated with* Y*, where* $\Delta y_i = y_{i+1} - y_i$ *for* $i = 0, 1..., n-1$.
2. *By recurrence, for* $k = 2, 3, ...n$*, we may then define the vector of* k^{th} *order differences* $\Delta^k Y = [\Delta^k y_0, \Delta^k y_1, ..., \Delta^k y_{n-k}]$*, where* $\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i$ *for* $i = 0, 1, ..., n-k$.

Difference operators are linear in the sense that:

$$\Delta^k (Y + Z) = \Delta^k Y + \Delta^k Z \text{ and } \Delta^k (aY) = a\Delta^k Y,\; a \in \mathbb{R},\; k = 2, 3, ...n.$$

Besides, one easily obtains a relation between divided differences and differences of all orders as shown below.

Theorem 4.4. *Let D_n be a set of data as defined in (4.1), where the x-nodes are equally spaced with $x_{i+1} - x_i = h, \forall i = 0, 1, ..., n-1$. Then for all k where $1 \leq k \leq$ with $i + k \leq n$:*

$$[x_i, x_{i+1}, ..., x_{i+k}] = \frac{\Delta^k y_i}{h^k k!} \tag{4.16}$$

Proof. The proof is done by induction on k. After verifying the result for $k = 1$, assume that it is true for $1, ..., k-1$, i.e.:

$$[x_i, x_{i+1}, ..., x_{i+k-1}] = \frac{\Delta^{k-1} y_i}{h^{k-1}(k-1)!}.$$

Since,

$$[x_i, x_{i+1}, ..., x_{i+k}] = \frac{[x_{i+1}, ..., x_{i+k}] - [x_i, x_{i+1}, ..., x_{i+k-1}]}{(x_{i+k} - x_i)},$$

then:

$$[x_i, x_{i+1}, ..., x_{i+k}] = \frac{\Delta^{k-1}y_{i+1} - \Delta^{k-1}y_i}{h^{k-1}(k-1)!(x_{i+k} - x_i)} = \frac{\Delta^{k-1}y_{i+1} - \Delta^{k-1}y_i}{h^{k-1}(k-1)!\,kh}.$$

that reaches the required result. ■

Based on the theorem above and in case of equally spaced data, Newton's interpolating polynomial is expressed as follows::
(4.17)

$$p(x) = y_0 + \frac{\Delta y_0}{1!h}(x-x_0) + \frac{\Delta^2 y_0}{2!h^2}(x-x_0)(x-x_1) + \cdots + \frac{\Delta^n y_0}{n!h^n}(x-x_0)(x-x_1)\ldots(x-x_{n-1})$$

where it is understood that $p(x) = p_{012...n}(x)$.

Remark 4.5. *Note the resemblance of this formula with that of Taylor's formula for a function $f(x)$ where the n^{th} degree polynomial representing $f(x)$ is given by:*

$$q(x) = f(x_0) + f^{'}(x_0)(x - x_0) + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

This remark will be exploited in Chapter 5 when approximating derivatives such as $f^{(k)}(x_0)$ by k^{th} order differences $\frac{\Delta^k f(x_0)}{h^n}$.
The result of the above theorem allows us therefore to compute divided difference tables by simply first computing differences as displayed in the following table:

i	$\mathbf{x_i}$	$\mathbf{y_i}$	Δ	Δ^2	Δ^3
0	x_0	y_0			
			$\Delta y_0 = y_1 - y_0$		
1	x_1	y_1		$\Delta^2 y_0 = \Delta y_1 - \Delta y_0$	
			$\Delta y_1 = y_2 - y_1$		$\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0$
2	x_2	y_2		$\Delta^2 y_1 = \Delta y_2 - \Delta y_1$	
			$\Delta y_2 = y_3 - y_2$		$\vdots$
3	x_3	y_3		$\vdots$	$\vdots$
$\vdots$			$\vdots$	$\vdots$	$\vdots$
$n-2$	x_{n-2}	y_{n-2}			$\vdots$
			$\Delta y_{n-2} = y_{n-1} - y_{n-2}$		
$n-1$	x_{n-1}	y_{n-1}		$\Delta^2 y_{n-2} = \Delta y_{n-1} - \Delta y_{n-2}$	
			$\Delta y_{n-1} = y_n - y_{n-1}$		
n	x_n	y_n			

Algorithm 4.4. Constructing a difference table

Example 4.4. *The following set of data D_4 is associated with* 0*-th order Bessel's function of the first kind.*

$\mathbf{i}$	$\mathbf{x_i}$	$\mathbf{y_i}$
0	1.0	0.7651977
1	1.3	0.6200860
2	1.6	0.4554022
3	1.9	0.2818186
4	2.2	0.1103623

Since the x-data are equally space with $h = 0.3$, the differences table can be easily constructed out of this data:

$\mathbf{i}$	$\mathbf{x_i}$	$\mathbf{y_i}$	Δ	Δ^2	Δ^3	Δ^4
0	1.0	0.7651977				
			$\Delta y_0 = -0.1451117$			
1	1.3	0.6200860		$\Delta^2 y_0 = -0.0195721$		
			$\Delta y_1 = -0.1646838$		$\Delta^3 y_0 = 0.0106723$	
2	1.6	0.4554022		$\Delta^2 y_1 = -0.0088998$		$\Delta^4 y_0 = 0.0003548$
			$\Delta y_2 = -0.1735836$		$\Delta^3 y_1 = 0.0110271$	
3	1.9	0.2818186		$\Delta^2 y_2 = 0.0021273$		
			$\Delta y_3 = -0.1714563$			
4	2.2	0.1103623				

Using this table, we may subsequently write any of the interpolation polynomials reproducing data in D_4. For example:
$p_{234}(x) = y_2 + \frac{\Delta y_2}{0.3}(x - x_2) + \frac{\Delta^2 y_2}{(0.3)^2 2!}(x - x_2)(x - x_3)$
$= 0.4554022 - \frac{0.1735836}{0.3}(x - 1.6) + \frac{0.0021273}{(0.3)^2}(x - 1.6)(x - 1.9)$
$p_{231}(x) = p_{23}(x) + [x_2, x_3, x_1](x - x_2)(x - x_3) = p_{23}(x) + [x_1, x_2, x_3](x - x_2)(x - x_3)$
$= p_{23}(x) + \frac{\Delta^2 y_1}{(0.3)^2 2!}(x - x_2)(x - x_3) = p_{23}(x) - \frac{0.0088998}{(0.3)^2 2!}(x - 1.6)(1 - 1.9)$

4.5 Errors in Polynomial Interpolation

When a function f is approximated on an interval $[a, b] = [x_0, x_n]$ by means of an interpolating polynomial p_n, it is naturally expected that the function be well approximated at all intermediate points between the nodes, and that as the number of nodes increases, this agreement will become more and more

accurate. Nevertheless, this expectation is $\underline{\text{ wrong and incorrect}}$.
A theoretical estimate of the error is derived in ([20], page 189) and leads to the following result:

Theorem 4.5. *Let f be a function in $C^{n+1}[a,b]$, and p_n the Lagrange polynomial of degree at most n, that interpolates f based on the set of data D_n. There exists some point $c \in (a,b)$ such that the error function:*

$$E_n(f(x)) = f(x) - p_n(x) = w_n(x)\frac{f^{(n+1)}(c)}{(n+1)!},$$

where $w_n(x) = (x-x_0)(x-x_1)....(x-x_n)$, and $x \in (a,b)$.

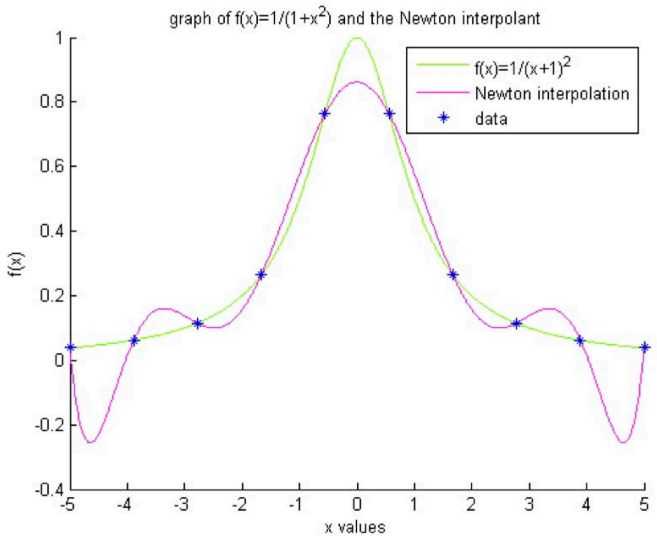
However, such result does **not** lead to a convergence result in the sense of:

$$\lim_{n\to\infty} |f(x) - p_n(x)| = 0, \, \forall x \in (a,b),$$

even if the function f possesses continuous derivatives of all orders in that interval.

Example 4.5. *A well-known counter example of this phenomenon is provided by the* **Runge function***:*

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



Let $p_{01\ldots n}(x)$ be the polynomial that interpolates this function at $n+1$ equally spaced nodes on the interval $[-5,+5]$ for example, including the endpoints. It is easy to verify the following contradictory results:

1. The curve representing $p_{01\ldots n}(x)$ assumes negative values, which obviously $f(x)$ does not have.

2. Adding more equally spaced nodes, leading to higher degree polynomials worsens the situation. The graphs of the resulting polynomials have wilder oscillations, especially near the endpoints of the interval, and the error increases beyond all bounds as confirmed in the graph .

Thus, in this case it can be shown that:

$$\lim_{n\to\infty} \max_{-5\leq x\leq +5} |f(x)-p_n(x)| = \infty.$$

This behaviour is called the "Runge' s phenomenon".
In a more advanced study of this topic [24], it is proved that the divergence of the polynomials is often due to the fact that the nodes of interpolation are equally spaced, which contrary to intuition, is usually a very poor and inappropriate choice . Specifically, one can show that:

$$|w_n(x)| \leq n!\frac{h^{n+1}}{4}$$

and therefore

$$\max_x |f(x)-p_n(x)| \leq \frac{\max_x |f^{(n+1)}(x)|}{4(n+1)} h^{n+1}$$

If $n\to\infty$, the order of magnitude of $\max_x |f^{(n+1)}(x)|$ could outweigh the nearly-zero order of $h^{n+1}/4(n+1)$.
In [24], numerical results are conducted in the case of the Runge function confirming this hypothesis. More specifically, it is verified that

$$\max_{-5\leq x\leq +5} |f^{(22)}(x)| = O(10^{19})$$

while the corresponding value of max $\frac{w_n(x)}{(n+1)!} = O(10^{-10})$
A much better and more adequate choice of nodes leading to more accurate results that help minimizing Runge' s phenomenon is obtained for example

with the set of Chebyshev nodes defined over the unit interval $[-1,+1]$ by:

$$x_i = \cos[\frac{2i-1}{2n}\pi]\,,\ 1 \leq i \leq n$$

(Note that these values are graphically obtained by projecting equally spaced points on the unit circle, down on the unit interval $[-1,+1]$). More generally over arbitrary interval $[a,b]$ the coordinates of Chebyshev nodes are:

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos[\frac{2i-1}{2n}\pi]$$

It is possible then to prove that

$$\lim_{n\to\infty} |f(x) - p_n(x)| = 0$$

This problem motivates the use of local piecewise polynomial interpolation.

4.6 Local Interpolation: Spline functions

As the **global** approach of interpolating polynomials does not provide in general a systematic and efficient way to approximate a function $f(x)$ on the basis of the data

$$D_n = \{(x_i, y_i)|i = 0, 1, ...n,\ ,\ x_0 = a, < x_1 < ... < x_{n-1} < x_n = b\},$$

we consider hereafter a **local** approach that considers approximating a function $f(x)$ by **spline functions**. Such functions are piecewise polynomials joined together with certain **imposed "smoothness conditions"**.
In the theory of splines, the interior points $\{x_i\}_{i=0}^n$ at which the function changes its expression are called the "nodes" or "knots" of the partition.
In this chapter, we analyze successively linear, quadratic and cubic spline interpolation rather than the global one.

4.6.1 Linear Splines

The simplest connection between two points is a line segment. A **spline of degree one** or **linear spline**, is therefore a function that consists of **linear** polynomial pieces joined together to achieve continuity of the polygonal curve representing it. Its formal definition is given as follows:

Definition 4.2. *A linear spline interpolating the data D_n is a function $s(x)$ such that:*

1. *$s_i(x) = \{s(x)|_{x\in[x_i,x_{i+1}]}\}$ is a polynomial of degree at most 1, i.e.*
$$s_i \in \mathbb{P}^1,\ \forall i = 0, 1, ..., n-1.$$

2. *$s(x)$ is* **globally continuous** *on $[a, b]$, (i.e. $s \in C([a, b]$, the set of all continuous functions on $[a, b]$).*

3. *$s(x_i) = y_i$, $\forall i = 0, 1, ..., n$. (***Interpolation conditions***)*

Note that there exists a unique function $s(x)$ verifying these three criteria:
- To determine $s(x)$, a total of $2n$ unknowns have to be evaluated as each of the linear polynomials $s_i(x)$ defined by the first criterion over the subinterval $[x_i, x_{i+1}]$ is determined by 2 parameters.
- The second and third criteria impose respectively continuity at the $n-1$ interior nodes in addition to the $n+1$ interpolation conditions, that add up to a total of $n - 1 + n + 1 = 2n$ restrictions.
The number of unknown parameters being equal to the number of imposed conditions, the equations of the linear spline are uniquely determined. We proceed directly to write them using Newton's linear interpolating polynomial form on each subinterval $[x_i, x_{i+1}]\,|\,i = 0, 1, ..., n-1$. This gives in a straightforward way:
(4.18)
$$s_i(x) = [x_i] + [x_i, x_{i+1}](x - x_i) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i),\ \forall i = 0, 1, ..., n-1.$$

Clearly, by joining the linear pieces $\{s_i(x)\,|i = 0, 1, ..., -1n\}$, one obtains the unique linear spline satisfying the definition above.

The linear spline algorithm is as follows:

Algorithm 4.5. Linear splines

```
function l = LinearSpline(x, y, r)
% Input: 2 vectors x and y of equal length, and a real number r
% Output: l= s(r): s=linear spline
n=length(x);
% seek i : x(i) < r < x(i+1)
i=max(find(x<r));
% compute l=s(r)=s_i(r)
l=y(i) + (y(i+1)-y(i)) / (x(i+1)-x(i)) * (r-x(i));
```

Example 4.6. *Consider the set of data $D_4 = \{(x_i, y_i = f(x_i))\ i = 0, 1, 2, 3, 4\}$*

where

i	x_i	y_i
0	1.0	7.6
1	1.3	2.0
2	1.6	4.5
3	1.9	2.8
4	2.2	11

Determine the linear spline function interpolating D_4, then interpolate $f(1.4)$

1. Given that $s_i(x) = y_i + \frac{y_{i+1}-y_i}{x_{i+1}-x_i}(x - x_i)$:
$s_0(x) = y_0 + \frac{y_1-y_0}{x_1-x_0}(x - x_0) = 7.6 - 18.6(x - 1)\ \ ; 1.0 \leq x \leq 1.3$
$s_1(x) = y_1 + \frac{y_2-y_1}{x_2-x_1}(x - x_1) = 2 + 8.3(x - 1.3)\ \ ; 1.3 \leq x \leq 1.6$
$s_2(x) = y_2 + \frac{y_3-y_2}{x_3-x_2}(x - x_2) = 4.5 - 5.6(x - 1.6)\ \ ; 1.6 \leq x \leq 1.9$
$s_3(x) = y_3 + \frac{y_4-y_3}{x_4-x_3}(x - x_3) = 2.8 + 27.3(x - 1.9)\ \ ; , 1.9 \leq x \leq 2.2$

2. Since $x_1 < x = 1.4 < x_2 \Rightarrow f(1.4) \approx s_1(1.4) = 2 + 8.3(1.4 - 1.3) = 2.83$

As indicated also by the graph, first order splines are not smooth functions, the first derivative being discontinuous at each interior node. This deficiency is overcome by looking at higher order degree splines.

4.6.2 Quadratic Spline Interpolation

We start by providing a definition for interpolating quadratic splines based on the data D_n.

Definition 4.3. *A quadratic spline interpolating the data D_n is a function $s(x)$ such that:*

1. *$s_i(x) = \{s(x)|_{x\in[x_i,x_{i+1}]}\}$, is a polynomial of degree at most 2 , i.e.*

$$s_i \in \mathbb{P}^2, \forall i = 0, 1, ..., n-1.$$

2. *$s(x)$ is globally of class C^1, that is:*
   - *(a)* $s \in C([a,b])$,
   - *(b)* $s' \in C([a,b])$.

3. *$s(x_i) = y_i$, $\forall i = 0, 1, ..., n$. (Interpolation conditions)*

In order to determine the equations of the interpolating quadratic spline, we start by counting the number of unknown parameters and imposed conditions from the definition.
- From the first criterion, each of the $s_i(x)$ is determined by 3 parameters. Hence, full obtention of $s(x)$ requires $3n$ unknowns.
- The second and third criteria impose respectively continuity of s and $s^{'}$ at the $n-1$ interior nodes in addition to the $n+1$ interpolation conditions, that add up to a total of $2(n-1)+n+1 = 3n-1$ restrictions.
Obviously, to obtain a <u>unique</u> determination of the interpolating quadratic spline, there appears to be a deficit of one further constraint!
There is a variety of ways of providing an additional condition on top of the three above . For example, one may impose specific values on $s'(x_0)$, such as:

(4.19) $$s'(x_0) = 0, \text{ \textbf{("natural quadratic spline")}}$$

or use the **forward difference approximation** formula to the derivative

(4.20) $$s'(x_0) = [x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$$

Instead of deriving the quadratic spline through a system of $3n$ equations in $3n$ unknowns, a shorter way to proceed is by noting first that $s^{'}(x)$ is a linear interpolating spline on the set of data $D_n^{'} = \{(x_i, s'(x_i) \,|i = 0, 1, ..., n\}$. In this view, introduce first the set of unknowns:

$$\{z_i = s'(x_i), \text{ for } i = 0, 1, ...n\}$$

Obviously, it is enough to start first by writing the equation of $s_i^{'}(x)$ followed by an integration process.

- On the subinterval $[x_i, x_{i+1}]$:

$$s_i'(t) = z_i + (\frac{z_{i+1} - z_i}{x_{i+1} - x_i})(t - x_i), \, \forall t \in [x_i, x_{i+1}], \, \forall i = 0, 1, ..n-1.$$

- Integration of this last equation from x_i to $x : x_i \le x \le x_{i+1}$, yields:

(4.21) $$s_i(x) = y_i + z_i(x - x_i) + \frac{1}{2}(\frac{z_{i+1} - z_i}{x_{i+1} - x_i})(x - x_i)^2.$$

- Imposing then the interpolation conditions $s_i(x_{i+1}) = y_{i+1}$, $i = 0, 1, ...n-$

1, provides a new set of n equations:

$$y_{i+1} = y_i + z_i(x_{i+1} - x_i) + (\frac{z_{i+1} - z_i}{x_{i+1} - x_i})\ \frac{(x_{i+1} - x_i)^2}{2}$$

- Through algebraic simplification and letting $h_{i+1} = x_{i+1} - x_i$, one obtains:
$$\frac{y_{i+1} - y_i}{h_{i+1}} = \frac{z_{i+1} + z_i}{2},\ i = 0, 1, ..., n-1.$$

Obviously, to determine $s_i(x)$, the values of the sequence $\{z_i\}$ should be computed first. Given an arbitrary z_0 chosen as suggested in equations (4.28) or (4.20), the sequence $\{z_i\}_{i=1}^n$ can be found from the recurrence relation:

$$z_{i+1} = -z_i + 2[x_i, x_{i+1}],\ i = 0, 1, ..., n-1, \tag{4.22}$$

where $[x_i, x_{i+1}]$, is the set of first order divided differences associated with the data D_n. It suffices then to determine the equations of the quadratic spline over the interval $[a, b]$, from equations (4.21)

Algorithm 4.6. Algorithm for Quadratic Spline

```
function q = QuadraticSpline(x, y, r)
% Input: 2 vectors x and y of equal length, and a real number r
% Output: q= s(r): s= quadratic spline based on data x and y
n=length(x); z=zeros(1, n);
% compute z(1) (or set z(1)=0) then compute z(i), i=2,...,n
z(1)=(y(2)-y(1)) / (x(2)-x(1));
for i=1:n-1
  z(i+1) = -z(i) + 2* (y(i+1)-y(i)) / (x(i+1)-x(i));
end
% seek i : x(i) < r < x(i+1) and compute q=s(r)=s_i(r)
i=max(find(x<r));
q=y(i) + z(i)*(r - x(i)) + (z(i+1)-z(i)) / (x(i+1)-x(i)) *  ((r -x(i))^2 /2) );
```

Example 4.7. *Find the natural quadratic spline interpolant for the following data:*

$$D_5 = \{(-1, 2); (0, 1); (0.5, 0); (1, 1); (2, 2); (2.5, 3)\}$$

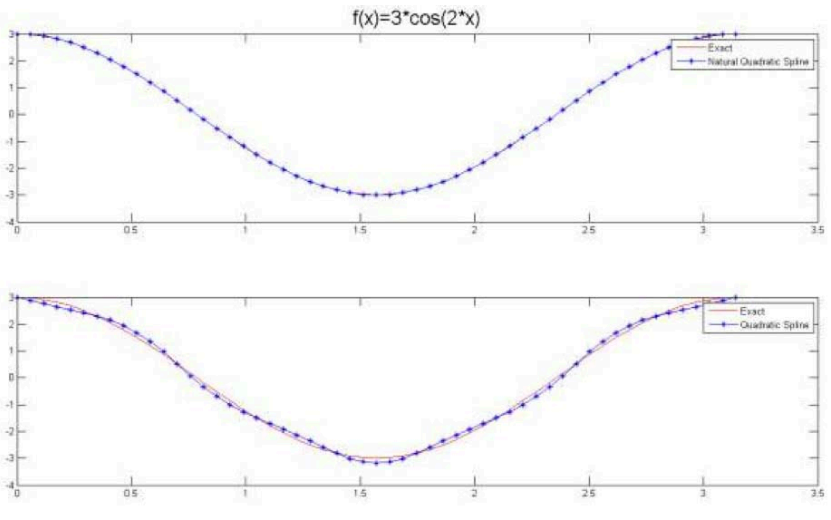
Let $z_0 = 0$. Using equation (4.21) recursively:

$$\{z_i\}_{i=0}^{5} = \{0, -2, -2, 6, -4, 8\}$$

From equation (4.22) the natural quadratic spline is given by:
$s_0(x) = -(x+1)^2 + 2$; $-1.0 \leq x \leq 0$
$s_1(x) = -2x + 1$; $0 \leq x \leq 0.5$.
$s_2(x) = 8(x - \frac{1}{2})^2 - 2(x - \frac{1}{2})$; $0.5 \leq x \leq 1.0$
$s_3(x) = -5(x-1)^2 + 6(x-1) + 1$; $1.0 \leq x \leq 2.0$
$s_4(x) = 12(x-2)^2 - 4(x-2) + 2$; $2.0 \leq x \leq 2.5$

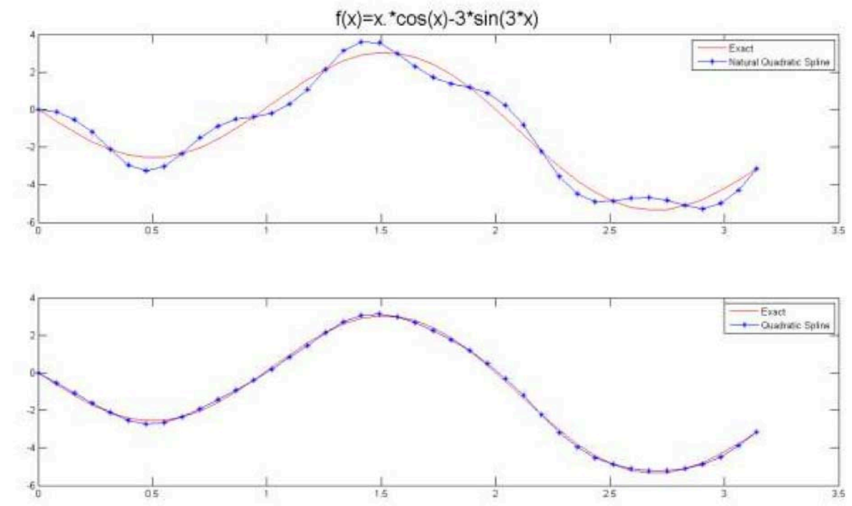
Remark 4.6. *On the choice of the additional condition on z_0.*

When conducting numerical tests regarding the use of the natural spline condition $z_0 = 0$, it was found that such constraint provides a good quadratic spline approximation results **only** in the case where the data $\{x_i, y_i\}$ correspond to a function $f(x)$ for which $f'(x_0) = 0$. This is shown in the following figure for $f(x) = 3\cos(2x)$.

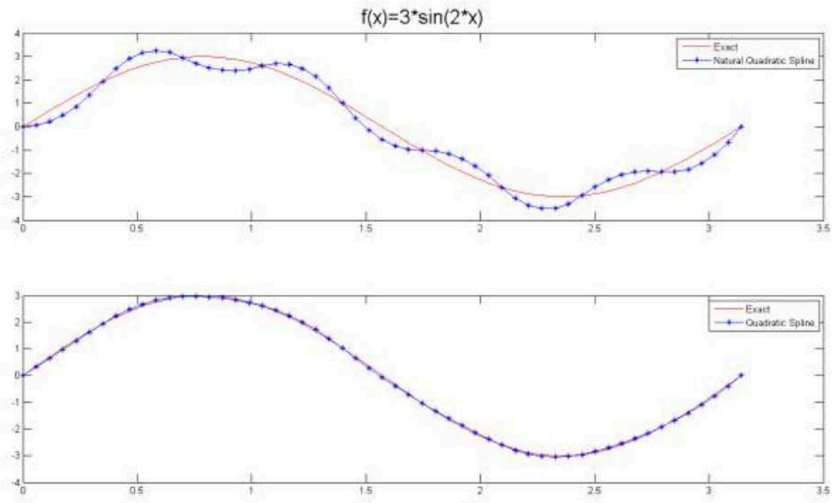


Otherwise, the additional condition $z_0 = [x_0, x_1]$ that generally approximates $f'(x_0)$, provides more accurate results. The next 2 figures attest for such facts for the functions:

- $f(x) = x\cos(x) - 3\sin(3x)$


f(x)=x.*cos(x)-3*sin(3*x)
Exact
Natural Quadratic Spline
Exact
Quadratic Spline

- $f(x) = 3\sin(2x)$


f(x)=3*sin(2*x)
Exact
Natural Quadratic Spline
Exact
Quadratic Spline

4.6.3 Cubic Spline Interpolation

In the previous two cases one notes the following:
- The polygonal lines representing linear splines lack of smoothness as the slope of the spline may change abruptly at each node.
- As for quadratic splines, the discontinuity is in the 2^{nd} derivative, and is therefore not so evident; nevertheless, the curvature of the spline changes abruptly at each node, and the curve may not be visually pleasing.
Cubic splines allow for smoother data fitting and they are most frequently used in applications. It can be proved that cubic spline functions are among the best interpolation functions that are available at an acceptable computational cost. In this case, we join cubic polynomials together in such a way that the resulting spline function has its first and second derivatives continuous everywhere. At each interior node, 3 conditions will be imposed, so that the graph of the function will look smoother than in the case of linear and quadratic splines. Discontinuities, of course, may occur in the 3^{rd} derivative, but these cannot be easily detected visually. Cubic splines are formally defined as follows.

Definition 4.4. *A cubic spline that interpolates the data D_n, is a function $s(x)$ such that:*

1. $s_i(x) = \{s(x)|_{x\in[x_i,x_{i+1}]}\}$ *is a polynomial of degree at most 3 , i.e.*

$$s_i \in \mathbb{P}^3 \, \forall i = 0, 1, ..., n-1.$$

2. $s(x) \in C^3([a, b])$*, i.e.:*
   (a) $s \in C([a, b])$.
   (b) $s' \in C([a, b])$.
   (c) $s'' \in C([a, b])$.

3. $s(x_i) = y_i$, $\forall i = 0, 1, ..., n$. *(Interpolation conditions)*

Following the same pattern as preceedingly, and counting the number of unknown parameters and imposed conditions from the definition, we note the following:
- From the first criterion, each of the $s_i(x)$ is determined by 4 parameters. Hence, full obtention of $s(x)$ requires $4n$ unknowns.
- The second and third criteria impose now respectively $3(n-1)$ continuity conditions for s, $s^{'}$ and $s^{''}$ at the interior nodes, in addition to the $n+1$ interpolation conditions

Hence for a total of $4n$ unknowns, one has a total of $3(n-1)+n+1 = 4n-2$ constraints. Obviously, to allow unique determination of the interpolating cubic spline, there appears to be a deficit of two constraints!
These two supplementary conditions may be for example supplied as follows:

1. Letting $s''(x_0) = s''(x_n) = 0$, the spline is called a **natural spline** (or free boundary)

2. An alternative to the natural spline is to use:

$$s''(x_0) = \frac{1}{2}[x_0, x_1, x_2] \approx f"(x_0) \text{ and } s''(x_n) = \frac{1}{2}[x_{n-2}, x_{n-1}, x_n] \approx f"(x_n)$$

3. Letting $s'(x_0) = f_0$ and $s'(x_n) = f_n$, the spline is called a **clamped spline**. However, for this type of boundary conditions to hold, it is necessary to have the values of $f'(x_0)$ and $f'(x_n)$(or at least an accurate approximation).

We will restrict our analysis to **natural cubic splines** only.
Instead of determining the solution of the problem through a system of $4n$ equations in 4n unknowns, we note that $s^{'}(x)$ and $s^{''}(x)$ are quadratic and linear splines based respectively on the data sets $D_n^{'} = \{(x_i, s'(_i)\}$ and $D_n^{''} = \{(x_i, s"(_i)\}$, where the unknowns:

$$\{z_i = s'(x_i) | i = 0, 1, ..., n\}$$

and

$$\{w_i = s''(x_i) | i = 0, 1, ..., n\}$$

represent respectively the sets of **slopes** and **moments** at the nodes.
Obviously, we should proceed by first writing $s_i^{"}(x)$ on the interval $[x_i, x_{i+1}]$ followed by 2 successive integrations.

- On the subinterval $[x_i, x_{i+1}]$:
(4.23)
$$s_i^{''}(t) = w_i + \frac{w_{i+1} - w_i}{x_{i+1} - x_i}(t - x_i), \forall t \in [x_i, x_{i+1}], \forall i = 0, 1, ..., n-1.$$

- Integration of (4.23) from x_i to x, $x_i \leq x \leq x_{i+1}$ yields:
(4.24)
$$s_i'(x) - z_i = w_i(x - x_i) + \frac{w_{i+1} - w_i}{x_{i+1} - x_i}\frac{(x - x_i)^2}{2}, \forall x \in [x_i, x_{i+1}], \forall i = 0, 1, ..., n-1$$

- Imposing in (4.24) the conditions $s_i'(x_{i+1}) = z_{i+1}$ at internal nodes, provide a new set of $n-1$ equations. Specifically:

  (4.25) $$\frac{z_{i+1} - z_i}{h_{i+1}} = \frac{w_{i+1} + w_i}{2},\ i = 0, 1, ..., n-1.$$

  which is equivalent to:

  (4.26) $$z_{i+1} = z_i + h_{i+1}\frac{w_i + w_{i+1}}{2},\ i = 1, ..., n-1.$$

- A second integration of equation (4.24) from x_i to x yields the cubic polynomials $s_i(x)$:
  (4.27)
  $$s_i(x) = y_i + z_i(x-x_i) + w_i\frac{(x-x_i)^2}{2} + \frac{w_{i+1} - w_i}{6h_{i+1}}(x-x_i)^3,\ \forall x \in [x_i, x_{i+1}],\ \forall i = 0, 1, ..., n-1.$$

- Imposing then the interpolation conditions $s_i(x_{i+1}) = y_{i+1}$ provides a new set of $n-1$ equations given by:
  (4.28)
  $$y_{i+1} = y_i + z_i h_{i+1} + w_i\frac{h_{i+1}^2}{2} + \frac{(w_{i+1} - w_i)h_{i+1}^2}{6},\ \forall i = 0, 1, ..., n-1$$

- This last equation leads to 2 simultaneous equations verified at all internal node of the spline, i.e. for all $i = 1, .., n-1$:

  $$\begin{cases} \frac{y_{i+1}-y_i}{h_{i+1}} = z_i + (w_{i+1} + 2w_i)\frac{h_{i+1}}{6} \\ \frac{y_i - y_{i-1}}{h_i} = z_{i-1} + (w_i + 2w_{i-1})\frac{h_i}{6} \end{cases}$$

  Subtracting these last 2 equations and using (4.25) gives:

  $$[x_i, x_{i+1}] - [x_{i-1}, x_i] = h_i\frac{w_{i-1} + w_i}{2} + h_{i+1}\frac{(w_{i+1} + 2w_i)}{6} - h_i\frac{(w_i + 2w_{i-1})}{6}$$

  Equivalently:
  (4.29)
  $$\frac{h_i}{6}w_{i-1} + \frac{h_i + h_{i+1}}{3}w_i + \frac{h_{i+1}}{6}w_{i+1} = (h_i + h_{i+1})[x_{i-1}, x_i, x_{i+1}],\ i = 1, 2, ..., n-1$$

Since the sought spline is “natural” ($w_0 = w_n = 0$), equations (4.29) provide therefore a system of $n-1$ equations in $n-1$ unknowns given by:

(4.30) $$Aw = r,$$

where the coefficient matrix A is:
(4.31)

$$A = \begin{pmatrix} (h_1+h_2)/3 & h_2/6 & 0 & 0 & 0 \dots 0 & \dots & 0 \\ h_2/6 & (h_2+h_3)/3 & h_3/6 & 0 & 0 \dots 0 & \dots & 0 \\ 0 & h_3/6 & (h_3+h_4)/3 & h_4/6 & 0 \dots 0 & \dots & 0 \\ 0 & 0 & h_4/6 & \dots & \dots \dots \dots & \dots & h_{n-1}/6 \\ 0 & \dots & \dots & 0 & \dots \dots \dots & h_{n-1}/6 & (h_{n-1}+h_n)/3 \end{pmatrix}_{n-1\times}$$

and the vectors w and r are respectively:

$$w = (w_1, w_2, ..., w_{n-1})^T$$

and

$$r = (r_1, r_2, ..., r_{n-1})^T \text{ with } r_i = (h_i + h_{i+1})[x_{i-1}, x_i, x_{i+1}]_y \tag{4.32}$$

Note also that the matrix $A = \{a_{ij}\}$ has the following properties:

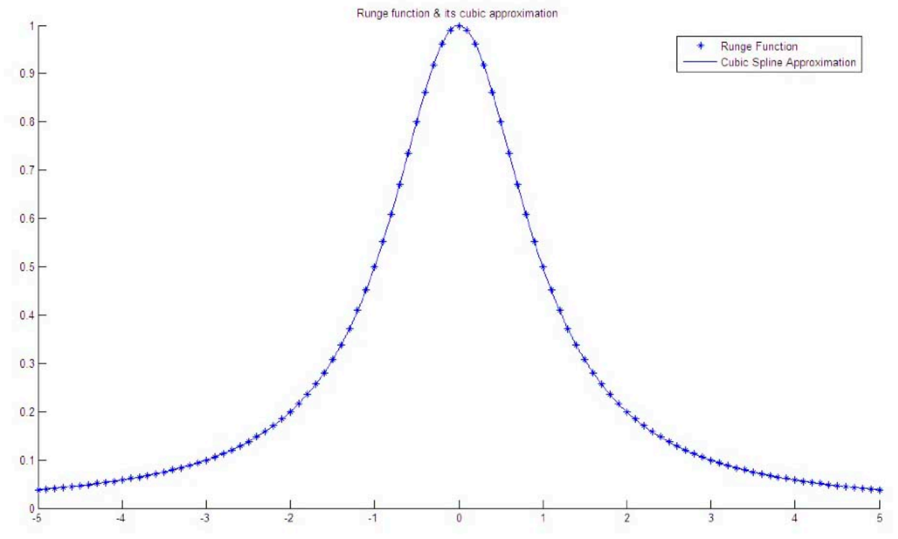
- A is symmetric, since $a_{ij} = a_{ji}$
- A is tridiagonal, since $a_{ij} = 0$ for all i, j with $|i - j| > 1$.
- A is strictly diagonally dominant, since $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$, $\forall i$.

Under these conditions, the system (4.30) has a unique solution that can be obtained through a straightforward Gauss reduction process that does not necessitate any pivoting strategy. We can now write a **pseudocode for the natural cubic spline.**

Algorithm 4.7. Cubic Spline

```
% Input the data  D_n
% Output: cubic spline  s(x)   interpolating on  D_n

% Obtain first w by solving   Aw=r by performing the following steps:

  1. Generate $r = [r_1, ..., r_{n-1}]^T$ with $r_i = (h_i + h_{i+1})[x_{i-1}, x_i, x_{i+1}], i = 1, ..., n-1$.

  2. Generate the matrix $A$.

  3. Perform Gauss reduction on $[A|r]$.
```

```
4. Perform back substitution on reduced system to get $w$ with $w_0 = w_n = 0$.
5. Compute $z_0 = [x_0, x_1] - (2w_0 + w_1)h_1/6$
6. Compute $z_{i+1} = z_i + h_{i+1}(w_{i+1} + w_i)/2, i = 0, 1, ..., n-1$.
7. Generate $s(x)$ through generating $s_i(x)$:
   $s_i(x) = y_i + z_i(x - x_i) + w_i(x - x_i)^2/2 + ((w_{i+1} - w_i/6h_{i+1})(x - x_i)^3$, $i = 0, 1, ..., n-1$.
```



Example 4.8. *Determine the natural cubic spline interpolating the following set of data:*

$$D_3 = \{(-1, 1); (1, 2); (2, -1); (2.5, 0)\}$$

- Since $w_0 = w_3 = 0$
  and $h_1 = x_1 - x_0 = 2$; $h_2 = x_2 - x_1 = 1$; $h_3 = x_3 - x_2 = 0.5$,
  the system (4.30) is:

$$\begin{pmatrix} 1 & 1/6 \\ 1/6 & 1/2 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -7/2 \\ 5 \end{pmatrix}$$

- Applying the Naive Gauss reduction on that system followed by back substitution:

$$w = [w_0 = 0, w_1 = -93/17, w_2 = 201/17, w_3 = 0]'$$

- Using (4.28) with $i = 0$ leads to

$$z_0 = [x_0, x_1] - (2w_0 + w_1)h_1/6 = -45/34$$

- Once the value of z_0 and (4.26), the vector of slopes is fully determined with:

$$z = [z_0 = -45/34, z_1 = --231/34/17, z_2 = -123/34, w_3 = -45/68]'$$

- Using (4.27) for successively $i = 0, 1, 2$, the equations of the cubic spline are then as follows:

$$S(x) = \left\{ \begin{array}{c} S_0(x) = 1 - \frac{9}{7}(x+1) - \frac{31}{68}(x+1)^3\,;\, -1 \leq x \leq 1 \\ S_1(x) = 2 - \frac{231}{34}(x-1) - \frac{93}{17}(x-1)^2 + \frac{49}{17}(x-1)^3\,;\, 1 \leq x \leq 2 \\ S_2(x) = -1 - -\frac{123}{34}(x-2) + \frac{201}{17}(x-2)^2 - \frac{67}{17}(x-2)^3\,; 2 \leq x \leq 2.5 \end{array} \right.$$

4.6.4 Solving a Triadiagonal System

Note that in case in D_n, the x-data are equidistant, i.e.

$$h_{i+1} = x_{i+1} - x_i = h, \, \forall i = 0, ..., n-1$$

the matrix A in (4.31) becomes:

$$A = h \begin{pmatrix} 2/3 & 1/6 & 0 & 0... & 0 \\ 1/6 & 2/3 & 1/6 & 0.. & 0 \\ 0 & 1/6 & 2/3 & 1/6.. & 0 \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ 0 & ... & 0 & 1/6 & 2/3 \end{pmatrix}$$

Since also $[x_{i-1}, x_i, x_{i+1}] = \frac{y_{i+1}-2y_i+y_{i-1}}{2h^2}$, the right hand side r in (4.32) simplifies, and the system (4.30) becomes:

$$h\begin{pmatrix} 2/3 & 1/6 & 0 & 0... & 0 \\ 1/6 & 2/3 & 1/6 & 0.. & 0 \\ 0 & 1/6 & 2/3 & 1/6 & 0.. \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ 0 & ... & 0 & 1/6 & 2/3 \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ ... \\ ... \\ ... \\ w_{n-1} \end{pmatrix} = \frac{1}{h}\begin{pmatrix} y_2 - 2y_1 + y_0 \\ y_3 - 2y_2 + y_1 \\ y_4 - 2y_3 + y_2 \\ ...... \\ y_n - 2y_{n-1} + y_{n-2} \end{pmatrix}.$$

The elements of the matrix A can be made independent of h, through dividing each of the equations by h, thus yielding the following tridiagonal system:

$$\begin{pmatrix} 2/3 & 1/6 & 0 & 0... & 0 \\ 1/6 & 2/3 & 1/6 & 0.. & 0 \\ 0 & 1/6 & 2/3 & 1/6 & 0.. \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ 0 & ... & 0 & 1/6 & 2/3 \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ ... \\ ... \\ ... \\ w_{n-1} \end{pmatrix} = \frac{1}{h^2}\begin{pmatrix} y_2 - 2y_1 + y_0 \\ y_3 - 2y_2 + y_1 \\ y_4 - 2y_3 + y_2 \\ ... \\ ... \\ y_n - 2y_{n-1} + y_{n-2} \end{pmatrix}.$$

In what follows, we consider the general triadiagonal system of equations: $Aw = r$:

$$\begin{pmatrix} a_1 & b_1 & 0 & 0... & 0 \\ c_1 & a_2 & b_2 & 0.. & 0 \\ 0 & c_2 & a_3 & b_3 & 0.. \\ ... & ... & ... & ... & ... \\ ... & ... & c_{N-2} & a_{N-1} & b_{N-1} \\ 0 & ... & 0 & c_{N-1} & a_N \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ ... \\ ... \\ ... \\ w_N \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ ... \\ ... \\ ... \\ r_N \end{pmatrix},$$

where the “diagonal” entries of the matrix A are generated by :

$$\begin{cases} \text{the "main diagonal" vector } a = [a_i : 1 \leq i \leq N] \\ \text{the “upper diagonal” vector } b = [b_i : 1 \leq i \leq N-1] \\ \text{the ‘lower diagonal” vector } c = [c_i : 1 \leq i \leq N-1] \end{cases}$$

satisfying the following properties:

$$\begin{cases} |a_i| > |b_i| + |c_{i-1}| : 2 \leq i \leq N-1 \\ |a_1| > |b_1|,\ |a_N| > |c_{N-1}| \end{cases}$$

The following algorithm solves this given system:

Algorithm 4.8. Diagonally dominant Triangular systems

```
function w=SolveTridiag(a,b,c,r)
% N is the dimension of a and r; N-1 is the dimension of b and c
% Start with the Gauss reduction process then use back-substitution
for k=1:N-1
    m=c(k)/a(k);
    a(k+1)=a(k+1)-m*b(k);
    r(k+1)=r(k+1)-m*r(k);
end
for k=N:-1:1
    w(k)=r(k)/a(k);
    if k>1
        r(k-1)=r(k-1)-w(k)*b(k-1);
    end
end
```

This algorithm takes $2N-1$ divisions, $3(N-1)$ multiplications and as many algebraic additions, thus a total of $8N-7$ flops.

4.6.5 Errors in Spline Interpolation

From ([25], pages 14 and 61) , we can state the following convergence result:

Theorem 4.6. *Let f be a function in $C^{k+1}[a,b]$, and S_k the Spline Function , that interpolates f based on the set of data D_n, where $k=1,2,3$. Then,*

$$\max_{[a,b]} |f(x)-S_k(x)| \leq C_k h^{k+1} \max_{[a,b]} |f^{(k+1)}(x)|$$

where $h=\max|x_i-x_{i-1}|$, for $1 \leq i \leq n$.

For example:

- If $k=1$, then $\max_{[a,b]} |f(x)-S_1(x)| = O(h^2)$
- If $k=2$, then $\max_{[a,b]} |f(x)-S_2(x)| = O(h^3)$
- If $k=3$, then $\max_{[a,b]} |f(x)-S_3(x)| = O(h^4)$

Note also that in Spline Interpolation, increasing the number of nodes for a fixed value of k, will definitely lead to convergence. One can prove that:

$$\forall x \in [a,b],]\lim_{n\to\infty} S_k(x) = f(x).$$

This property is noticeably absent for global Lagrange interpolation (recall Runge example).

4.7 Concluding Remarks

1. Based on a set of data D_n, considering higher degree Lagrange interpolating polynomials does not guarantee reaching more accurate approximations of the unknown function f; this problem can be overcome by spline functions, particularly cubic splines. However, neither are suitable to extrapolate information from the available set of data D_n. To generate new values at points lying outside the interval $[x_0, x_n]$, one could use for example, regression analysis based least squares approximations.

2. Polynomial interpolation can also be used to approximate multi-dimensional functions. In particular, spline functions interpolation is well suited when the region is partitioned into polygons in 2D (triangles or quadrilaterals) and polyhedra in 3D (tetrahedra or prisms). See([24]).

4.8 Exercises

Polynomial Interpolation

1. Use the Lagrange interpolation process to obtain a polynomial of least degree that satisfies the following set of data: $D_3 = \{(0,7),(2,11),(3,28),(4,63)\}$.

2. For the four interpolation nodes $-1,1,3,4$, what are the l_i functions required in the Lagrange interpolation procedure ? Draw the graphs of these functions to show their essential properties. Use Lagrange interpolation form to obtain a polynomial of least degree that satisfies the following set of data: $D_3 = \{(-1,1),(1,0),(3,2),(4,-3)\}$.

3. Write the Lagrange form of the interpolating polynomial of degree ≤ 2 that interpolates $f(x)$ at x_0, x_1 and x_2, where $x_0 < x_1 < x_2$.

4. Given the data

$$D_4 = \{(1,-1),(2,-1/3),(2.5,3/32),(3,4/3),(4,25)\}$$

   (a) Construct the divided difference table.

   (b) Use the "best" quadratic then cubic Newton's interpolating polynomial, to find an approximation to $f(2.9)$.

5. Write Newton's interpolating polynomial for the data shown:

$$\{(0,7),(2,11),(3,28),(4,63)\}$$

6. Using a difference table, derive the polynomial of least degree that assumes the values $2, 14, 4, 2$ and 2 respectively for $x = -2, -1, 0, 1$ and 2. Use that result, to find a polynomial that takes the values shown and has at $x = 3$ the value 10.

7. The polynomial $p(x) = x^4 - x^3 + x^2 - x + 1$ satisfies the following set of data

i	$\mathbf{x_i}$	$\mathbf{y_i}$
0	-2	31
1	-1	5
2	0	1
3	1	1
4	2	11
5	3	61

Find a polynomial q that takes these values:

$\mathbf{i}$	$\mathbf{x_i}$	$\mathbf{y_i}$
0	-2	31
1	-1	5
2	0	1
3	1	1
4	2	11
5	3	30

8. Construct a divided difference (or difference) table based on the two given sets of data in the preceding exercise, then use Newton's polynomials of all orders (1,2, 3, 4) to approximate $f(2.5)$, in each case.

9. Create the table of all Neville's polynomials in P_4 satisfying the following set of data:

$\mathbf{i}$	$\mathbf{x_i}$	$\mathbf{y_i}$
0	1.0	-1.5
1	2.0	-0.5
2	2.5	0.0
3	3.0	0.5
4	4.0	1.5

10. Determine by two methods the polynomial of degree 2 or less whose graph passes through the points $(0, 1.1), (1, 2)$, and $(2, 4.2)$. Verify that they are the same.

11. Let $f(x) = x^3 + 2x^2 + x + 1$. Find the polynomial of degree 4 that interpolates the values of f at $x = -2, -1, 0, 1, 2$. Find the polynomial of degree 2 that interpolates the values of f at $x = -1, 0, 1$.

12. (a) - Consider the following set of data:

$$D_5 = \{(-2, 1); (-1, 4); (0, 11); (1, 16); (2, 13); (3, -4)\}$$

Show that the interpolating polynomial based on D_5 is cubic.
(b) - The set D_5 is altered as follows:

$$D_5' = \{(-2, 1); (-1, 4); (0, 11); (1, 16); (2, 10); (3, -4)\},$$

so that $y_4 = 10$. Based on D_5' and using the polynomial found in part (a), find $q_{01234}(x)$, without computing new divided differences.

13. The polynomial $p(x) = x^4 + 3x^3 - 2x^2 + x + 1$ interpolates the set of data

i	**0**	**1**	**2**	**3**	**4**
$\mathbf{x_i}$	-1	-2	0	1	2
$\mathbf{y_i}$	-4	-17	1	4	35

Without computing any Difference or Divided Difference, use Newton's form to determine the polynomial $q(x)$ interpolating the following set of data:

i	**0**	**1**	**2**	**3**	**4**
$\mathbf{x_i}$	-1	-2	0	1	2
$\mathbf{y_i}$	-4	0	1	4	35

Spline Interpolation

14. Determine whether each of the following functions is a first degree spline, and plot its graph.

(a)

$$S(x) = \begin{cases} x\,;\ -1 \le x \le 0 \\ 1 - x\,;\ 0 \le x < 1 \\ 2x - 2\,;\, 1 \le x \le 2 \end{cases}$$

(b)

$$S(x) = \begin{cases} x\,;\ -1 \le x \le 0.5 \\ 0.5x + 2(x - 0.5)\,;\ 0.5 \le x \le 2 \\ x + 1.5\,;\ 2 \le x \le 4 \end{cases}$$

15. Determine the linear spline function $s(x)$ interpolating the set of data D_3 and plot its graph. Interpolate $f(2.4)$.

$$D_3 = \{(0,1);(1.5,3);(2,5);(3,2)\}$$

16. Is $S(x) = |x|$ a first degree spline ? Why or why not ?

17. Find the natural quadratic spline interpolant for the following data

(a)

i	**x**	**y**
0	-1	2
1	0	1
2	$1/2$	0
3	1	1
4	2	2
5	$5/2$	3

(b)

i	**x**	**y**
0	1	2
1	2	1
2	$5/2$	0
3	3	1
4	4	3

18. Are these functions quadratic splines ? Explain why or why not ?Plot their graphs.

(a)

$$Q(x) = \begin{cases} -x^2\,;\, 0 \le x \le 1 \\ x\,;\, 0 \le x \le 100 \end{cases}$$

(b)

$$Q(x) = \begin{cases} x\,;\, -50 \le x \le 1 \\ x^2\,;\, 1 \le x \le 2 \\ 4\,;\, 2 \le x \le 50 \end{cases}$$

19. Determine a quadratic spline that interpolates the data $f(0) = 0, f(1) = 1, f(2) = 2$, and satisfiea $s'(0) = 2$.

20. Do there exist a,b, c and d so that the function

$$S(x) = \begin{cases} -x\,;\, -10 \le x \le -1 \\ ax^3 + bx^2 + cx + d\,;\, -1 \le x \le 1 \\ x\,;\, 1 \le x \le 10 \end{cases}$$

is a natural cubic spline function ?

21. Do there exist coefficients for which the function

$$S(x) = \begin{cases} x+1\,;\ -2 \le x \le -1 \\ ax^3+bx^2+cx+d\,;\ -1 \le x \le 1 \\ x-1\,;\ 1 \le x \le 2 \end{cases}$$

is a natural cubic spline function ? Why or why not ?

22. Determine the natural cubic spline that interpolates the function $f(x) = x^6$ over the interval $[0,2]$ using nodes $0,1$ and 2 .

23. Find the natural cubic spline interpolant for this table

i	x	y
0	1	0
1	2	1
2	3	0
3	4	1
4	5	0

24. Find the natural cubic spline interpolant for this table

i	x	y
0	1	0
1	2	1
2	3	0
3	4	1
4	5	0
	6	1

25. Consider the following set of data generated using the function $f(x) = x cos\,x - 2x^2 + 3x - 1$

i	x	y
0	0.1	-0.620499
1	0.2	-0.283986
2	0.3	$+0.006600$
3	0.4	$+0.248424$

(a) Construct the natural cubic spline for the data above

(b) Use the cubic spline constructed above to approximate $f(0.25$ and $f'(0.25)$, and calculate the absolute error.

26. Give an example of a cubic spline with nodes $0, 1, 2$, and 3 that is quadratic in $[0, 1]$, cubic in $[1, 2]$, and quadratic in $[2, 3]$.

27. Give an example of a cubic spline with nodes $0, 1, 2$, and 3 that is linear in $[0, 1]$, but of degree 3 in the other two intervals.

28. List all the ways in which the following functions fail to be natural cubic splines:

    (a)
    $$S(x) = \begin{cases} x+1\,;\ -2 \le x \le -1 \\ x^3 - 2x + 1\,;\ -1 \le x \le 1 \\ x-1\,;\ 1 \le x \le 2 \end{cases}$$

    (b)
    $$S(x) = \begin{cases} x^3 + x - 1\,;\ -1 \le x \le 0 \\ x^3 - x - 1\,;\ 0 \le x \le 1 \end{cases}$$

29. Determine the coefficients so that the function
    $$S(x) = \begin{cases} x^2 + x^3\,;\ 0 \le x \le 1 \\ a + bx + cx^2 + dx^3\,;\ 1 \le x \le 2 \end{cases}$$
    is a cubic spline that has the property $S_1'''(x) = 12$

30. Use the data points $(0, 1), (1, e), (2, e^2), (3, e^3)$ to form a natural cubic spline that approximates $f(x) = e^x$.

31. Use the cubic spline obtained in the preceding exercise, to approximate $\int_0^3 e^x\, dx$. What is the relative error in this approximation ?

32. Construct a natural cubic spline to approximate $f(x) = e^{-x}$ based on the nodes $x = 0, 0.25, 0.75$ and 1. Integrate the spline over the interval $[0, 1]$ and compare the results to $\int_0^1 e^{-x}\, dx$. Use the derivatives of the spline to approximate $f'(0.5)$ and $f"(0.5)$. Compare the approximations to the actual values.

33. How many additional conditions are needed to specify uniquely a spline of degree 4 over n knots ?

34. Let S be a cubic spline that has knots $t_0 < t_1 < ... < t_n$. Suppose that on the 2 intervals $[t_0, t_1]$ and $[t_2, t_3]$, S reduces to linear polynomials. What can be said of S on $[t_1, t_2]$?

4.9 Computer Projects

Exercise 1: Polynomial Interpolation
Let $x = [x_1, x_2, ..., x_n]$ and $y = [y_1, y_2, ..., y_n]$ be $\underline{\underline{\text{2 vectors of equal length n}}}$, representing a set of n points in the plane:

$$D_n = \{(x_i, y_i) | x_1 < x_2 < ... < x_n \quad ; i = 1, 2, ..., n\}$$

where $y_i = f(x_i)$ for some real valued function f.
To solve Exercise 1, use the MATLAB **function p =NevillePolynomial(x, y, r)** given in the lecture notes. (DO NOT CHECK VALIDITY OF INPUTS)
This function takes as input the 2 vectors x and y and a real number r, with $x_1 < r < x_n$, and computes

$$p = p_{1,2,...,n}(r)$$

where $\underline{\underline{p_{1,2,...,n}(.) \text{ is Neville's form of the Interpolating Polynomial based on the set of data } D_n}}$.

1. Write a MATLAB **function v =VectorNevillePolynomial(x, y, w)** that takes as input the 2 vectors x and y, and a vector w of any length, and computes the values of Neville's polynomial at each component of w. The output of this function is a vector v whose components are:
$$v(i) = p_{1,2,...,n}(w(i)), \ \forall \ i = 1, ..., length(w)$$
(Assume that $x_1 < w(i) < x_n \ \ \forall i$).

2. Consider the $\underline{\text{Runge function}}$ $f(x) = \frac{1}{1+x^2}$ on the interval $[-5, +5]$.

   Write a MATLAB **function [x, fx, s, fs] =GenerateVectors(n, f)** that takes as input an integer n and the Runge function f. Your function:

   - First : generates a vector x of length n, whose components are n-equally spaced points in the interval $[-5, +5]$ including the end-points, evaluates f at these points, and saves these values in a vector fx.
     $\underline{\text{Hint: The MATLAB built-in function } \textbf{linspace(a,b,n)} \text{ generates a row vector of n}}$ $\underline{\text{equally spaced points between a and b, including the end-points.}}$

- Secondly : generates a vector s of length $(n-1)$ whose i^{th} component is the midpoint of the interval $[x_i, x_{i+1}]$, that is:

  $$s = [s_1 = \frac{x_1 + x_2}{2}, ..., s_i = \frac{x_i + x_{i+1}}{2}, ..., s_{n-1} = \frac{x_{n-1} + x_n}{2}]$$

  Your function then evaluates f at all components of s and saves these values in a vector fs.

3. Write a MATLAB **function PlotPolynomial(n, f)** that takes as input an integer n and the Runge function f and plots in the same figure window, the $\underline{\text{graphs of } f \text{ and } p_{1,2,...,n}}$ over the set of ordered points in

   $$X = x\ U\ s = \{x_i, s_i, x_{i+1}\,|\, i = 1, ..., n-1\}$$

   Note that $p_{1,2,...,n}$ is Neville's form of the Interpolating Polynomial based on the set of data represented by x and fx.

4. Write a MATLAB **function EP =ErrorPolynomial(n, f)** that takes as input an integer n and the Runge function f. Your function outputs a matrix EP of size $(n-1) \times 4$, whose 4 columns are successively the vectors:

   $$f(s) \qquad p_{1,2,...,n}(s) \qquad err = |p_{1,2,...,n}(s) - f(s)| \qquad relerr = \frac{|p_{1,2,...,n}(s) - f(s)|}{|f(s)|}$$

5. Test each of the functions of this exercise on 2 different test cases $n > 10$, (n : odd integer). Save your results and graphs in a word document.

Exercise 2: Spline Interpolation
All questions are as in Exercice 2, but applied to the QUADRATIC SPLINE instead of the interpolating polynomial.
Let $x = [x_1, x_2, ..., x_n]$ and $y = [y_1, y_2, ..., y_n]$ be $\underline{\underline{\text{2 vectors of equal length n}}}$, representing a set of n points in the plane:

$$D_n = \{(x_i, y_i) | x_1 < x_2 < ... < x_n \ \ ; i = 1, 2, ..., n\}$$

where $y_i = f(x_i)$ for some real valued function f.

To solve Exercise 2, use the MATLAB **function q =QuadraticSpline(x, y, r)** given in the lecture notes. (DO NOT CHECK VALIDITY OF INPUTS)
This function takes as input the 2 vectors x and y and a real number r, with $x_1 < r < x_n$, and computes

$$q = Q(r)$$

where $\underline{\underline{Q(.) \text{ is the Quadratic Spline Interpolating the set of data } D_n}}$.

1. Write a MATLAB **function v =VectorQuadraticSpline(x, y, w)** that takes as input the 2 vectors x and y, and a vector w of any length, and computes the values of the Quadratic Spline function at each component of w.
(Assume that $x_1 < w(i) < x_n \ \ \forall i = 1, ..., length(w)$).

2. Consider the Runge function $f(x) = \frac{1}{1+x^2}$ on the interval $[-5, +5]$.

   Write a MATLAB **function PlotSpline(n, f)** that takes as input an integer n and the Runge function f and plots in the same figure window, the $\underline{\text{graphs of } f \text{ and } Q}$ over the set of ordered points in

   $$X = x\ U\ s = \{x_i, s_i, x_{i+1}\,|\,i = 1, ..., n-1\}$$

   Note that Q is the Quadratic Spline interpolating the set of data represented by x and fx.
   <u>Hint</u>: Call for the function **GenerateVectors(n, f)** programed in Exercise 2.

3. Write a MATLAB **function ES =ErrorSpline(n, f)** that takes as input the integer n and the Runge function f. Your function outputs a matrix ES of size $(n+1) \times 4$ whose 4 columns are successively the vectors:

   $$f(s) \qquad Q(s) \qquad err = |Q(s) - f(s)| \qquad relerr = \frac{|Q(s) - f(s)|}{|f(s)|}$$

4. Test each of the functions of this exercise on 2 different test cases $n > 20$, (n : odd integer). Save your results and graphs in a word document.

Exercise 3: Quadratic Spline Interpolation
Let $D_n = \{(x_i, y_i)|i = 1, 2, ..., n\ ;\ x_1 < x_2 < ..... < x_n;\ y_i = f(x_i)\ ,\ \text{f : unknown}\}$

be a given set of n points in the plane. The objective of this exercise is to determine the Quadratic Spline Interpolant $S(x)$, based on D_n. For this purpose:

1. Write a Matlab **function z = QuadrSplineDerivatives(x,y)** which takes as input a set of 2 vectors $x = [x_1, x_2, ..., x_n]$ and $y = [y_1, y_2, ..., y_n]$ as given by D_n, and returns a vector z : whose components are the derivatives of the Quadratic Spline at all nodes of the interpolation. Select $z(1)$ arbitrarily.

2. Write a Matlab **function C = QuadrSplineCoefficients(x,y)** which takes as input a set of 2 vectors x and y, finds the derivatives of the corresponding Quadratic Spline at all the nodes of the interpolation, and returns a matrix C of size $3 \times (n-1)$ representing the coefficients $(y_i, z_i, \frac{z_{i+1}-z_i}{x_{i+1}-x_i})$ of the Quadratic Spline over each subinterval $[x_i, x_{i+1}]$.

3. Write a Matlab **function E = EvaluateQuadrSpline(x,y,u)** which computes the value of $S(u)$ by locating first u in the appropriate subinterval $[x_i, x_{i+1}]$. Your function should also display an error message if $u \notin [x_1, x_n]$. (For example"The value of S(2.5) cannot be evaluated")

4. Write a Matlab **function V = EvaluateQuadrSpline1(x,y,w)** which computes the value of the Quadratic Spline at each component of a given vector w of any length.

5. Write a Matlab **function PlotQuadrSpline(x,y)** which takes as input a set of 2 vectors x and y and plots the graph of $S(x)$ over each subinterval $[x_i, x_{i+1}]$.

6. Test each one of the functions above for 2 different test cases, and save the results in a word document.