AlQalam
Alq J Med App Sci
https://journal.utripoli.edu.ly/index.php/Alqalam/index eISSN 2707-7179

Review article

Review New Applications using Interpolation and Polynomial Approximation

Abdalftah Elbori[1]*, Wafaa Abou zenad[1], Ali Albarki[2]

[1]*Department of Mathematics, Faculty of Science, Azzaytuna University, Tarhuna, Libya*
[2]*Department of Mathematics, Faculty of Education, Azzaytuna University, Tarhuna-Libya*

ARTICLE INFO

Corresponding Email. *Abdalftah81@yahoo.com*

Received*: 11-07-2024*
Accepted*: 01-10-2024*
Published*: 24-11-2024*

Keywords*. Lagrange Polynomials, Cubic Spline Interpolation, Divided Differences, Parametric Curves.*

ABSTRACT

This study analyses the mathematical structure of the interpolation to guarantee that the resulting function passes through all given data points. At the same time, polynomial approximation aims to find a polynomial function that closely matches the data but may not pass through all points and is typically determined using the least squares. Both methods have their advantages and are used in various fields, depending on the specific requirements of the problem at hand. The paper also explores the Lagrange Interpolation Error Theorem, providing insights into the accuracy and limitations of interpolation techniques. Understanding these methods and their error 1318ata1318cteristiccs helps in selecting the appropriate approach for various practical applications.

Cite this article. *Elbori A, Abou zenad W, Albarki A. Review New Applications using Interpolation and Polynomial Approximation. Alq J Med App Sci. 2024;7(4):1318-1333. https://doi.org/10.54361/ajmas.247459*

INTRODUCTION

Interpolations are important in many practical areas, including complex dynamics models for both mathematical calculations and systems research, boundary value solutions of various types of ordinary and partial differential equations, and simulation of various forms of offline temporary process data. Recursive and predictor-corrector forms of polynomial interpolation have been used in applications or in models devised to assist with practical problems. Examples include the construction of parallel algorithms for boundary value solution of linear ordinary differential equations with constant coefficients, one-time data compression, and harmful taint modeling [1,2].

Interpolation is a mathematical process of constructing new data points yielding the polynomial which will satisfy a given set of conditions. Usually, these conditions are a finite number of values of the polynomial itself, its derivatives at points, or function values. Polynomial approximation is concerned with polynomial interpolation of points that are not pre-specified. For the latter case, expansion models are often used to find the interpolating polynomial. Such models include the more popular least-squares polynomial approximations and the numerical approximations used to generate the Chebyshev polynomials. Optimization of Shape Parameters in Kernel Density Estimation using Asymptotic Theory for Nearest Neighbor Regression [3].

Definition and basic concepts

Given two integers, not both equal to 0, the division operation returns the unique integers q and r such that $a = bq + r$ and $0 \leq r < |b|$. In a finite Galois field, one can design an interpolation with linear complexity that is optimal in terms of both computational complexity and space using auxiliary polynomial and rational polynomial representations of the remainders in the Euclidean divisions involved. Such representations can also be constructed space-optimally. The running times, for any arbitrary selection of the field and two integers, are then linear and singly logarithmic in the related field size, respectively. At a cost with single logarithmic complexity in the field size, faster space-optimal

representations can be achieved with the following up. These representations allow one to manipulate the remainders by calls to polynomial addition, subtraction, and multiplication. High-speed modulas are instrumental in evaluating Reed-Solomon type error-correcting codes or interleaving with size- and position-adjusted dimensions at only a cost linear in the Galois field order [4].

The concept of a polynomial approximation of continuous maps of measure zero has been known for a long time in constructive analysis. For any continuous bounded function defined on a compact interval [a, b], there exists a sequence of polynomials with integer coefficients such that, in the uniform norm, the distance from one of the partial sums to the given function of measure zero is at most ε. A classical theorem that is based on Bezout's theorem states that, given a set of n points (each defined by a coordinate and a frequency) in the (two-dimensional version of the) plane, there is a rational function (a ratio of two polynomials) of degree at most d such that a zero of each of its partial derivatives can be found at any of the points. This leads to a speeding-up of the computation of the coefficients that represent wavelet functions on the fly. "Whichever direction you choose; the notion of interpolation is central in the field of polynomial approximation." In particular, division with remainder, a.k.a. Euclidean division, is essential in the process of constructing this function. Our primary fields of interest are mathematics and information and computer sciences, but we are open to discussing and developing any kind of potential windows of opportunity for applying these methods. Our investigations revealed that some of the functions and characteristics accepted as a basis are not actually effectively used. We believe that meaningful usage of this kind of mathematical knowledge remains an open question. We are interested in finding and researching the potentials and advantages of these activities. This list can also be effectively used as an extended appendix to this work and a manual for developers in the mentioned fields who intend to create and use polynomial approximations. This work is aimed at developing new applications in various fields using the mentioned mathematical methods. Of course, we aim to take advantage.

Interpolation and Lagrange Polynomials

One of the most useful classes of functions is polynomials

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

where n is a nonnegative integer and $a_0, a_1, \ldots, a_n$ are real constants? One reason is that any continuous function can be approximated by a polynomial arbitrarily close. By this we mean that given any continuous function, there exists a polynomial that is as “close” to the given function as desired.

Theorem 1:- (Weierstrass Approximation Theorem)

Suppose $f \in C[a,b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$ such that

$$|f(x) - P(x)| < \epsilon, \qquad \forall x \in [a,b]$$

A typical example for polynomial approximations is the Taylor polynomial

$$P_n(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \frac{(x-x_0)^3}{3!}f'''(x_0) + \cdots + \frac{(x-x_0)^n}{n!}f^n(x_0)$$

For example, using Taylor polynomials at $x_0 = 0$ to approximate $f(x) = e^x$, we obtain

$$P_0(x_0) = 1, P_1(x_0) = 1 + x$$

$$P_2(x_0) = 1 + x + \frac{x^2}{2!}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$P_n(x_0) = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} = \sum_{k=0}^{n} \frac{x^k}{k!}$$

Note that even for higher-degree polynomials, error becomes progressively worse as we move away from the point $x_0 = 0$.

$$f(3) = 20.0855$$

$$P_0(3) = 1, P_1(3) = 4, \ldots., P_4(3) = 16.375$$

Remark: For Taylor polynomials, all information used in the approximation is concentrated at the single number x_0, so these polynomials will generally give inaccurate approximations as we move away from x_0. This limits Taylor polynomial approximation to the situation in which approximations are needed only at numbers close to x_0.

A good interpolation polynomial needs to provide a relatively accurate approximation over an entire interval, and Taylor polynomials do not generally do this. It is usually more efficient to develop methods that use information spread at various points.

Lagrange Interpolation Polynomials

Suppose that a function $f(x)$ passes through two points (x_0, y_0) and (x_1, y_1). Define the following **linear Lagrange polynomials**

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \qquad L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

It is easy to verify that

$$L_0(x_0) = 1, \quad L_0(x_1) = 0,$$
$$L_1(x_0) = 0, \quad L_1(x_1) = 1.$$

The **linear Lagrange interpolating polynomial** through (x_0, y_0) and (x_1, y_1) is

$$P(x) = y_0 L_0(x) + y_1 L_1(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1$$

It can be verified that $P(x_0) = y_0$ and $P(x_1) = y_1$

Example

Determine the linear Lagrange interpolating polynomial that passes through the points $(2, 4)$ and $(5, 1)$.

Then, the linear Lagrange interpolating polynomial is

$$P(x) = y_0 L_0(x) + y_1 L_1(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 = -\frac{1}{3}(x - 5).4 + \frac{1}{3}(x - 2).1 = -x + 6$$

To generalize the concept of linear interpolation, consider the construction of a polynomial of degree at most n that passes through the following $n + 1$ points:

Definition: (Lagrange Interpolating Polynomials). The n-th Lagrange interpolating polynomials are defined by

$$L_{n,k}(x) = \frac{(x - x_0) \dots (x - x_{k-1})(x - x_{k+1}) \dots (x - x_n)}{(x_k - x_0) \dots (x_k - x_{k-1})(x_k - x_{k+1}) \dots (x_k - x_n)} = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}, k = 0,1,2, \dots, n$$

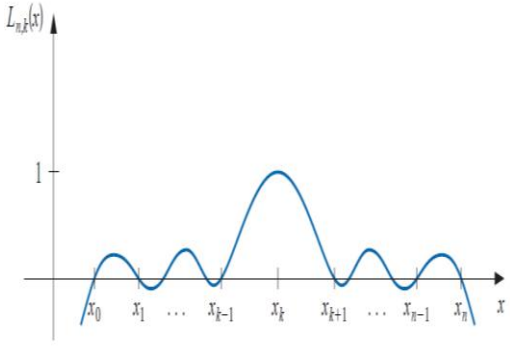

Figure 1: The n-th Lagrange interpolating polynomials

We may write $L_{n,k}(x)$ simply as $L_k(x)$ when there is no confusion as to its degree.

Theorem 2: If $x_0, x_1, \dots, x_n$ are $n + 1$ distinct numbers and f is a function whose values are given at these numbers, then a unique polynomial $P(x)$ of degree at most n exists with

$$f(x_k) = P(x_k), \text{ for each } k = 0,1, \dots, n.$$

This polynomial is given by

$$P(x) = f(x_0) L_{n,0}(x) + \cdots + f(x_n) L_{n,n}(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x), k = 0,1, \dots, n$$

and

$$L_{n,k}(x) = \frac{(x - x_0) \dots (x - x_{k-1})(x - x_{k+1}) \dots (x - x_n)}{(x_k - x_0) \dots (x_k - x_{k-1})(x_k - x_{k+1}) \dots (x_k - x_n)} = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}, k = 0,1,2, \dots, n$$

For the error bound of the Lagrange interpolation, we have the following result

Theorem 3 (Lagrange Interpolation Error Theorem).

Suppose $x_0, x_1, \dots, x_n$ are distinct numbers in the interval $[a, b]$ and $f \in C^{n+1}[a, b]$. Then, for each $x \in [a, b]$, there exists a number $\xi(x)$ between $\min\{x_0, x_1, \dots, x_n\}$ and $max\{x_0, x_1, \dots, x_n\}$, and hence in $[a, b]$, such that

$$f(x) = P(x) + \frac{f^{n+1}(\xi)}{(n+1)!}(x - x_0)(x - x_1) \dots (x - x_n)$$

where $P(x)$ is the interpolating polynomial of $f(x)$.

Comparison of Lagrange polynomials and Taylor polynomials

The nth-degree Taylor polynomial around x_0 concentrates all the known information at x_0, and has an error term of the form

$$\frac{f^{n+1}(\xi)}{(n+1)!}(x - x_0)^{n+1}$$

The nth-degree Lagrange polynomial uses information at distinct numbers $x_0, x_1, \dots, x_n$, and, in place of $(x - x_0)^n$, its error formula uses a product of the $n + 1$ terms

$$\frac{f^{n+1}(\xi)}{(n+1)!}(x - x_0)(x - x_1) \dots (x - x_n).$$

Example: we found the 2nd-degree Lagrange polynomial $P_2(x)$ for $f(x) = \frac{1}{x}$, we have on [2,4] using the nodes

$$x_0 = 2, \quad x_1 = 2.75, \quad x_2 = 4$$

Determine the error form of this polynomial, and the maximum error when the polynomial is used to approximate $f(x)$ for $x \in [2\ 4]$.

Since $f(x) = x^{-1}$, we have

$$f'(x) = -x^{-2}, f''(x) = 2x^{-3}, \quad f'''(x) = -6x^{-4}.$$

By Theorem 3, we have on the interval [2,4],

$$|f(x) - P(x)| = \left|\frac{f^3(\xi)}{3!}g(x)\right| = \frac{6\xi^{-4}}{3!}|g(x)| = \frac{1}{\xi^4}|g(x)| \le \frac{1}{16}|g(x)|$$

Where

$$g(x) = (x - 2)(x - 2.75)(x - 4) = x^3 - \frac{35}{7}x^2 + \frac{49}{2}x - 22.$$

We now need to determine the maximum value of $g(x)$ on $x \in [2\ 4]$. $g'(x) = \frac{1}{2}(3x - 7)(2x - 7)$

The critical points are $x_1 = \frac{7}{3}$ and $x_2 = \frac{7}{2}$. The global extrema are among the critical points and endpoints $x = 2$ and $x = 4$.

$$g(0) = 0, \quad g\left(\frac{7}{3}\right) = \frac{25}{108}, \quad g\left(\frac{7}{2}\right) = -\frac{9}{16}, \quad g(4) = 0$$

Hence, the maximum error is

$$\max_{x\in[2\ 4]} |f(x) - P(x)| \le \frac{1}{16}\left|-\frac{9}{16}\right| = \frac{9}{256} \approx 0.0351$$

we found the error at $x = 3$ is

$$|f(3) - P(3)| = 0.03418 < 0.0351 = \max_{x\in[2\ 4]} |f(x) - P(x)|$$

Divided Differences

Recall the n-th Lagrange Interpolation $P(x)$ of the function $f(x)$ at $n + 1$ distinct points $x_0, x_1, \dots, x_n$.

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

where

$$L_{n,k}(x) = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}, k = 0,1,2, \dots, n$$

If we have one more data point available $(x_{n+1}, f(x_{n+1}))$, then how to construct a new $n + 1$-th degree interpolation $P_{n+1}(x)$. To have to abandon all Lagrange polynomial $L_{n,k}(x)$), and reconstruct new Lagrange polynomials $L_{n+1,k}(x)$.

Is there a more efficient way for adding more data points? Although the interpolation polynomial $P_n(x)$ is unique, there are alternative representations that are useful in certain situations.
In fact, we can write $P_n(x)$ in the following form

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0) \dots (x - x_{n-1})$$

for appropriate constants $a_0, a_1, \dots, a_n$. Note that $P_n(x_0) = f(x_0) \Rightarrow a_0 = f(x_0)$

$$P_n(x_1) = f(x_1) \Rightarrow f(x_1) = f(x_0) + a_1(x_1 - x_0) \Rightarrow a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

By same way we can get

$$a_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

We now introduce the divided-difference notation, which will be very useful in determine the values of a_i
Divided Difference: The **zeroth divided difference** of the function f at x_i is $f[x_i] = f(x_i)$.
The **first divided difference** of f at x_i and x_{i+1} is

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

The **second divided difference** of f at x_i, $x_{i+1}$and x_{i+2} is

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}.$$

In general, the k-**th divided difference** of f at $x_i, x_{i+1}, \dots, x_{i+k}$ is

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

It can be seen that

$$a_0 = f(x_0) = f[x_0], \qquad a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1] \;\&\; a_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} = f[x_0, x_1, x_2]$$

In general, we have $a_2 = [x_0, x_1, \dots, x_{i+k}]$ for all $k = 0,1, \dots, n$.

Interpolation with Newton's Divided Difference
The Lagrange interpolation $P_n(x)$ of $f(x)$ at $x_0, x_1, \dots, x_n$ can be written as

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \dots, x_k](x - x_0) \dots (x - x_{k-1})$$

Divided Difference Table

x	$f(x)$	First divided differences	Second divided differences	Third divided differences
x_0	$f[x_0]$	$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$		
x_1	$f[x_1]$	$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$	$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$	
x_2	$f[x_2]$	$f[x_2, x_3] = \frac{f[x_3] - f[x_2]}{x_3 - x_2}$	$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1}$	$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0}$
x_3	$f[x_3]$	$f[x_3, x_4] = \frac{f[x_4] - f[x_3]}{x_4 - x_3}$	$f[x_2, x_3, x_4] = \frac{f[x_3, x_4] - f[x_2, x_3]}{x_4 - x_2}$	$f[x_1, x_2, x_3, x_4] = \frac{f[x_2, x_3, x_4] - f[x_1, x_2 x_3]}{x_4 - x_1}$
x_4	$f[x_4]$	$f[x_4, x_5] = \frac{f[x_5] - f[x_4]}{x_5 - x_4}$	$f[x_3, x_4, x_5] = \frac{f[x_4, x_5] - f[x_3, x_4]}{x_5 - x_3}$	$f[x_2, x_3, x_4, x_5] = \frac{f[x_3, x_4, x_5] - f[x_2, x_3, x_4]}{x_5 - x_2}$
x_5	$f[x_5]$			

Then determine the Newton interpolation polynomial
Example: Compute a divided difference table for these function values:

x	3	1	5	6

	$f(x)$	1	-3	2	4

Arrange the table vertically to have

i	x	$f[x]$	$f[x_{i-1},x_i]$	$f[x_{i-2},x_{i-1},x_i]$	$f[x_{i-3},x_{i-2},x_{i-1},x_i]$
0	3	1			
			2		
1	1	-3			
			$\frac{5}{4}$		
2	5	2		$\frac{-3}{8}$	
			2		
3	6	4		$\frac{3}{20}$	$\frac{7}{4}$

The Newton interpolation polynomial is

$$P(x) = f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1) + f[x_0,x_1,x_2,x_3](x-x_0)(x-x_1)(x-x_2)$$
$$= 1 + 2(x-3) - \frac{3}{8}(x-3)(x-1) + \frac{7}{4}(x-3)(x-1)(x-5)$$
$$= \frac{1}{40}(7x^3 - 78x^2 + 301x - 250)$$

Cubic Spline Interpolation

We here introduced the approximation of arbitrary functions on closed intervals using a single polynomial. However, high-degree polynomials can oscillate erratically, that is, a minor fluctuation over a small portion of the interval can induce large fluctuations over the entire range. The following is a 20th degree Lagrange interpolation approximating the back of a duck. Clearly this does not reflect the profile of the back.

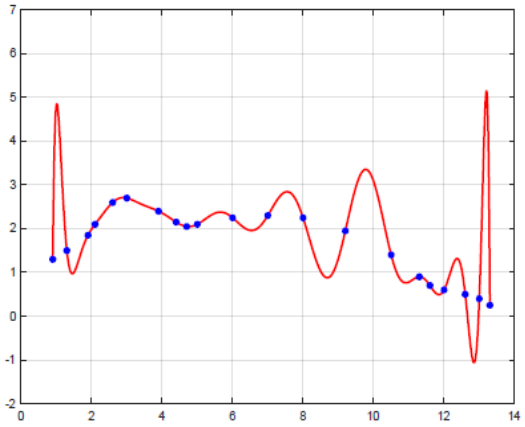

Figure 2: Cubic Spline Interpolation

We introduce interpolation using piecewise polynomials. This will effectively prevent oscillation. The simplest piecewise-polynomial approximation is **piecewise-linear interpolation**. Consider a set of data points:

x_0	x_1	x_2	...	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	...	$f(x_n)$

The following is a piecewise linear polynomial interpolation of a smooth curve
it is continuously differentiable over the entire domain (to ensure the smoothness). **It requires no specific derivative** information of the original function, except perhaps at the two endpoints of the interval (minimum information from original function). The most common piecewise-polynomial approximation is called **cubic spline interpolation.** The interpolation uses piecewise cubic polynomials, and globally second-order differentiable $\left(S \in C^2([x_0,x_n])\right)$.

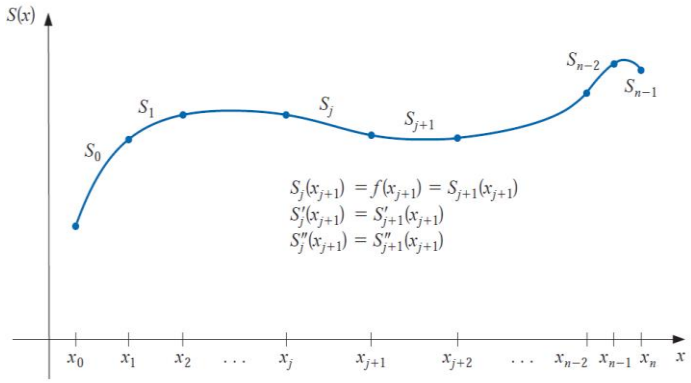

Figure 3: Cubic spline interpolation piecewise-polynomial approximation

Cubic Spline Interpolation (Natural Spline)

Consider a set of data points

$$\begin{array}{cccccc} x_0 & x_1 & x_2 & \dots & x_n \\ f(x_0) & f(x_1) & f(x_2) & \dots & f(x_n) \end{array}$$

We construct a cubic spline interpolant $S(x)$ for f satisfies. On each subinterval $[x_j, x_{j+1}]$, $S(x)$ is a cubic polynomial, denoted by $S_j(x)$ for $j = 0,1, \dots, n-1$.

$$S_j(x_j) = f(x_j) \text{ and } S_j(x_{j+1}) = f(x_{j+1}) \text{ for } j = 0,1, \dots, n-1.$$
$$S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \text{ for } j = 0,1, \dots, n-2$$
$$S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \text{ for } j = 0,1, \dots, n-2$$

The **natural boundary conditions**

$$S(x_0) = S(x_n) = 0$$

Example: Construct a natural spline that passes through the points (1,2), (2,3), and (3,5).

This spline consists of two cubics. The two subintervals are [1,2] and [2,3]. We write the piecewise cubic polynomial as follows

$$S(x) = \begin{cases} S_0(x) = a_0 + b_0(x-1) + c_0(x-1)^2 + d_0(x-1)^3 & \text{0n } [1,2] \\ S_1(x) = a_1 + b_1(x-2) + c_1(x-2)^2 + d_1(x-2)^3 & \text{0n } [2,3] \end{cases}$$

There are eight constants to be determined, which requires 8 conditions. Four conditions come from nodal values

$$S_0(1) = 2 \Rightarrow a_0 = 2,$$
$$S_0(2) = 3 \Rightarrow b_0 + c_0 + d_0 = 1,$$
$$S_1(2) = 3 \Rightarrow a_1 = 3,$$
$$S_1(3) = 5 \Rightarrow b_1 + c_1 + d_1 = 2,$$

Two conditions come from derivatives at interior nodes $x_1 = 2$.

$$S'_0(2) = S'_1(2) \Rightarrow b_0 + 2c_0 + 3d_0 = b_1$$
$$S''_0(2) = S''_1(2) \Rightarrow 2c_0 + 6d_0 = 2c_1$$

The last two conditions are from the natural boundary conditions

$$S''_0(1) = 0 \Rightarrow 2c_0 = 0$$
$$S''_1(3) = 0 \Rightarrow 2c_1 + 6d_1 = 0$$

Solve this system of the eight equations gives the spline

$$S(x) = \begin{cases} 2 + \frac{3}{4}(x-1) + \frac{1}{4}(x-1)^3 & \text{0n } [1,2] \\ 3 + \frac{3}{2}(x-2) + \frac{3}{4}(x-2)^2 - \frac{1}{4}(x-2)^3 & \text{0n } [2,3] \end{cases}$$

Construction of a natural cubic spline: Assume the following $n+1$ distinct points

x_0	x_1	x_2	...	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	...	$f(x_n)$

subdivide the interval $[x_0, x_n]$ into n subintervals: $I_j = [x_j, x_{j+1}], j = 0,1, \dots, n-1$. The cubic spline $S(x)$ restricted on the interval I_j is a cubic polynomial $S_j(x)$ for each $0 \le j \le n-1$:

$$S(x)|_{I_j} = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

There are 4n constants a_j , b_j , c_j , and d_j to be determined

On the interval $[x_j, x_{j+1}]$, denote the length $h_j = x_{j+1} - x_j$. We note that

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

It is easy to see that

$$a_j = f(x_j),\ j = 0,1, \dots, n-1. \qquad (3.1)$$

If we also let $a_n = f(x_n)$, then by some calculations we can represent b_j and d_j in terms of c_j for 0 $0 \le j \le n-1$.

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}) \qquad (3.2)$$

$$c_j = \frac{1}{3h_j}(c_{j+1} - c_j) \qquad (3.3)$$

Here, we also use the definition

$$c_n = S_n''(x_n)/2$$

We obtain a linear system for c_j, 0 $0 \le j \le n-1$

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

Write it as a linear system $Ax = b$ where

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & & \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & \\ \vdots & & \ddots & \ddots & 0 \\ & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

The matrix A is strictly diagonally dominant, that is,

$$|a_{ii}| > \sum_{j=1}^{n} |a_{ij}|, \forall i = 1,2, \dots, n.$$

The matrix A is nonsingular and the linear system $Ax = b$ has a unique solution. To solve the linear system $Ax = b$ with Matlab, we can use the command "backslash" x = A\ b. Let us construct Matlab File For Natural Spline coefficient as the following**:**

```
function [a1, b1, c1, d1] = natural_spline_coef(dataX, dataY) % Number of intervals n = length(dataX) - 1; h = zeros(1, n); % Calculate intervals h(j) for j = 1:n h(j) = dataX(j + 1) - dataX(j); end % Initialize coefficient matrix A and vector bb A = zeros(n + 1, n + 1); A(1, 1) = 1; % Natural spline condition at the start A(n + 1, n + 1) = 1; % Natural spline condition at the end for j = 2:n A(j, j - 1) = h(j - 1); A(j, j + 1) = h(j); A(j, j) = 2 * (h(j - 1) + h(j)); end bb = zeros(n + 1, 1); for j = 2:n bb(j) = 3 / h(j) * (dataY(j + 1) - dataY(j)) - 3 / h(j - 1) * (dataY(j) - dataY(j - 1)); end % Solve for c1 coefficients c1 = A \ bb; % Initialize a1, b1, and d1 a1 = dataY(:); % Ensure a1 is a column vector b1 = zeros(n, 1); d1 = zeros(n, 1); % Calculate b1 and d1 coefficients for j = 1:n b1(j) = 1 / h(j) * (a1(j + 1) - a1(j)) - h(j) / 3 * (2 * c1(j) + c1(j +
```

1)); d1(j) = (c1(j + 1) - c1(j)) / (3 * h(j)); end % Adjust sizes of a1 and c1 for output a1 = a1(1:end-1); % Remove the last element for segment consistency c1 = c1(1:end-1); % Remove the last element for segment consistency end

```
function [a1,b1,c1,d1] = natural_spline_coef(dataX,dataY)
n = length(dataX) – 1;h = zeros(1,n);for j = 1:n
h(j) = dataX(j+1) – dataX(j);
end
A = zeros(n+1,n+1);  A(1,1) = 1; A(n+1,n+1) = 1;
for j = 2:n
A(j,j-1) = h(j-1); A(j,j+1) = h(j);
A(j,j) = 2*(h(j-1)+h(j));
end
bb = zeros(n+1,1);
for j = 2:n
bb(j) = 3/h(j)*(dataY(j+1)-dataY(j)) – 3/h(j-1)*(dataY(j)-dataY(j-1));
end
c1= A\bb;
a1= reshape(dataY(1:n+1),n+1,1);
b1= zeros(n,1);
d1= zeros(n,1);
for j = 1:n
b1(j) = 1/h(j)*(a1(j+1)-a1(j)) – h(j)/3*(2*c1(j)+c1(j+1));
d1(j) = (c1(j+1) – c1(j))/(3*h(j));
end
a1(n+1) = [];
c1(n+1) = [];
```

Matlab File for Natural Spline Interpolation

```
function y1 = natural_spline(dataX,dataY,x);  [a1,b1,c1,d1] = natural_spline_coef(dataX,dataY);
y1 = zeros(size(x));
for n = 1:length(x)
for j = 1:length(dataX)-1
if dataX(j) <= x(n) && dataX(j+1) >= x(n)
k = j;
break
end
end
xk = dataX(k);
y1(n) = a1(k) + b1(k)*(x(n)-xk) + c1(k)*(x(n)-xk)^2 + d1(k)*(x(n)-xk)^3;
end
```

Example: By using Taylor polynomial to approximate the exponential function $f(x) = e^x$. Use the following data points

x	0	1	2	3
$f(x)$	1	e	e^2	e^3

To form a natural cubic spline $S(x)$ that approximates $f(x) = e^x$ By using Matlab

```
dataX = [0,1,2,3]; dataY = [exp(0),exp(1),exp(2),exp(3)];
[a,b,c,d] = natural_spline_coef(dataX,dataY);
disp(' a1 b1 c1 d1')
disp('------------------------')
disp([a,b,c,d])
x = 0:0.01:3;
y = natural_spline(dataX,dataY,x);
figure(1)
plot(dataX,dataY,'r*','linewidth',3)
hold on
plot(x,exp(x),'b-','linewidth',3)
plot(x,y,'k-.','linewidth',3)
lgd = legend('data points','y = e^x','y = S(x)');
```

```
hold off
grid on
lgd.FontSize = 14;
lgd.Location = 'NorthWest';
```

$$S(x) = \begin{cases} 1.000 + 1.4660x + 0.2523x^3 & 0\text{n } [0,1] \\ 2.7183 + 2.2229(x-1) + 0.7569(x-1)^2 + 1.6911(x-1)^3 & 0\text{n } [1,2] \\ 7.3891 + 8.8098(x-2) + 5.8301(x-2)^2 - 1.9434(x-2)^3 & 0\text{n } [2,3] \end{cases}$$

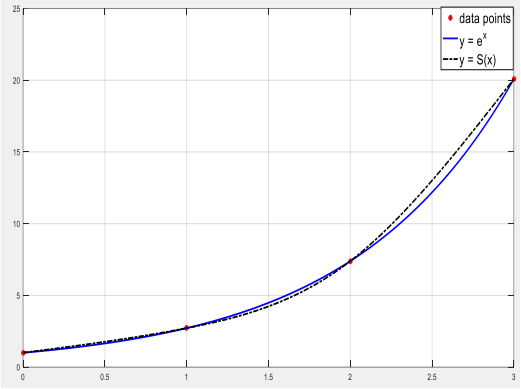

Figure 4. Natural cubic spline to approximate the exponential function $f(x) = e^x$

Clamped Splines

The clamped spline is another type of cubic splines that use different boundary conditions. Comparing with the free boundary condition in the natural spline,

$$S''(a) = 0, \qquad S''(b) = 0,$$

the clamped spline specifies the slope at the endpoints, i.e.,

$$S'(a) = f'(a), \qquad S'(b) = f'(b),$$

So, the clamped spline requires additional information

x_0	x_1	x_2	...	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	...	$f(x_n)$
$f'(x_0)$	$f'(x_1)$	$f'(x_2)$	...	$f'(x_n)$

The rest of the conditions are exactly the same as the natural splines.

Example: We revisit previous Example, and this time we construct a clamped spline that passes through the points $(1,2), (2,3)$, and $(3,5)$ that has $S'(1) = 2$ and $S'(3) = 1$. There are two pieces in the spline $S(x)$:

$$S_0(x) = a_0 + b_0(x-1) + c_0(x-1)^2 + d_0(x-1)^3 \quad 0\text{n } [1,2]$$
$$S_1(x) = a_1 + b_1(x-2) + c_1(x-2)^2 + d_1(x-2)^3 \quad 0\text{n } [2,3]$$

Most conditions (6 out of 8) are the same as the natural spline

$$f(1) = 2 \Rightarrow a_0 = 2,\ S_0(2) = 3 \Rightarrow b_0 + c_0 + d_0 = 1.$$
$$f(2) = 3 \Rightarrow a_1 = 3,\ S_1(3) = 5 \Rightarrow b_1 + c_1 + d_1 = 2.$$
$$S_0'(2) = S_1'(2) \Rightarrow b_0 + 2c_0 + 3d_0 = b_1$$
$$S_0''(2) = S_1''(2) \Rightarrow 2c_0 + 6d_0 = 2c_1$$

The clamped boundary conditions yield

$$S_0'(1) = 2 \Rightarrow b_0 = 2,$$
$$S_1'(3) = 1 \Rightarrow b_1 + 2c_1 + 3d_1 = 1,$$

Solve the system for eight unknowns, we have

$$S(x) = \begin{cases} 2 + 2(x-1) - \frac{5}{2}(x-1)^2 + \frac{3}{2}(x-1)^3 & 0\text{n } [1,2] \\ 3 + \frac{3}{2}(x-2) + 2(x-2)^2 - \frac{3}{2}(x-2)^3 & 0\text{n } [2,3] \end{cases}$$

Construction of a clamped cubic spline

Define the cubic polynomial on each interval to be

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

The coefficients a_j , b_j , c_j , and d_j g are define as (3.1) and (3.2). For , $j = 1,2,\dots,n-1$, we have the following equations for c_j

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

In addition, we have the clamped boundary conditions:

$$2h_0 c_0 + h_0 c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a)$$

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

The clamped spline defined on $a = x_0 < x_1 < \cdots < x_n = b$ is unique. $\{c_j\}$ satisfies the linear system $Ax = b$ where

$$A = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \\ \vdots & & \ddots & \ddots & 0 \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix}, \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Matlab File For Clamped Spline Coefficient

```
function [a1,b1,c1,d1] = clamped_spline_coef(dataX,dataY,dFa,dFb)
n = length(dataX) – 1;
h = zeros(1,n);
for j = 1:n
h(j) = dataX(j+1) – dataX(j);
end
A = zeros(n+1,n+1);
A(1,1) = 2*h(1); A(1,2) = h(1);
A(n+1,n) = h(n); A(n+1,n+1) = 2*h(n);
for j = 2:n
A(j,j-1) = h(j-1);
A(j,j+1) = h(j);
A(j,j) = 2*(h(j-1)+h(j));
end
bb = zeros(n+1,1);
bb(1) = 3/h(j)*(dataY(2)-dataY(1)) – 3*dFa;
bb(n+1) = 3*dFb – 3/h(n)*(dataY(n+1)-dataY(n));
for j = 2:n
bb(j) = 3/h(j)*(dataY(j+1)-dataY(j)) – 3/h(j-1)*(dataY(j)-dataY(j-1));
end
c1 = A\bb;
a1 = reshape(dataY(1:n+1),n+1,1);
b1 = zeros(n,1);
d1 = zeros(n,1);
for j = 1:n
b1(j) = 1/h(j)*(a1(j+1)-a1(j)) – h(j)/3*(2*c1(j)+c1(j+1));
d1(j) = (c1(j+1) – c1(j))/(3*h(j));
end
```

```
a1(n+1) = [];
c1(n+1) = [];
```

Matlab File For Clamped Spline Interpolations

```
function y = clamped_spline(dataX,dataY,dFa,dFb,x)
[a1,b1,c1,d1] = clamped_spline_coef(dataX,dataY,dFa,dFb);
y1 = zeros(size(x));
for n = 1:length(x)
for j = 1:length(dataX)-1
if dataX(j) <= x(n) && dataX(j+1) >= x(n)
k = j;
break
end
end
xk = dataX(k);
y1(n) = a1(k) + b1(k)*(x(n)-xk) + c1(k)*(x(n)-xk)^2 + d1(k)*(x(n)-xk)^3;
end
```

Example: We revisit the spline interpolation of $f(x) = e^x$. Use the following data points

x	0	1	2	3
$f(x)$	1	e	e^2	e^3

This time we use the clamped spline with the additional information $f'(0) = 1$, and $f'(3) = e^3$. Then, compare the accuracy with the natural spline interpolation. We solve the problem using Matlab programing.

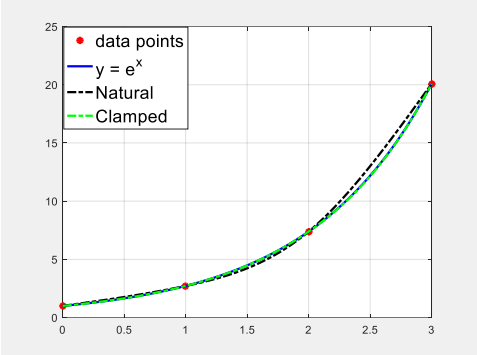

Figure 5: Clamped cubic spline to approximate the exponential function $f(x) = e^x$

The natural spline is

$$S(x) = \begin{cases} 1.000 + 1.4660x + 0.2523x^3 & \text{0n } [0,1] \\ 2.7183 + 2.2229(x-1) + 0.7569(x-1)^2 + 1.6911(x-1)^3 & \text{0n } [1,2] \\ 7.3891 + 8.8098(x-2) + 5.8301(x-2)^2 - 1.9434(x-2)^3 & \text{0n } [2,3] \end{cases}$$

The clamped spline is

$$S(x) = \begin{cases} 1.000 + 1.000x + 0.4447x^2 + 0.2736x^3 & \text{0n } [0,1] \\ 2.7183 + 2.7102(x-1) + 1.2655(x-1)^2 + 0.6951(x-1)^3 & \text{0n } [1,2] \\ 7.3891 + 7.3265(x-2) + 3.3509(x-2)^2 + 2.0191(x-2)^3 & \text{0n } [2,3] \end{cases}$$

From the plot, we can see that the clamped spline is more accurate than the natural spline. This is not surprise since the boundary conditions for the clamped spline are exact.

Theorem (Error bound for Clamped Cubic Splines): Let $f \in C^4[a,b]$ with

$$\max_{a \le x \le b} |f^4(x)| = M$$

If $S(x)$ is the unique clamped cubic spline interpolation to f with respect to the nodes $a = x_0 < x_1 < \cdots < x_n = b$, then for all $x \in [a,b]$,

$$|f(x) - S(x)| \le \frac{5M}{384} \max_{0 \le j \le n-1} \left(x_{j+1} - x_j\right)^4$$

Remark: A fourth-order error bound also holds in the case of natural spline interpolation, but it is more difficult to express. There are other cubic spline interpolations that do not require the derivative of f. For example, the popular "not-a-knot spline" requires that the third-order derivative $S'''(x)$ is continuous at x_1 and x_{n-1}

Example: In this example, we approximate the top profile of the duck using cubic spline interpolation

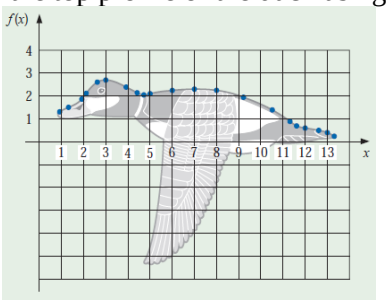

Figure 6: Interpolation cubic spline to approximate the top profile of duck

In general, the more points we use, the better approximation we can expect. We chose 21 data points as depicted above and shown below in the table. Note that more points are placed where the curve is changing more rapidly

x	0.9	1.3	1.9	2.1	2.6	3.0	3.9	4.4	4.7	5.0	6.0	7.0	8.0	9.2	10.5	11.3	11.6	12.0	12.6	13.0	13.3
$f(x)$	1.3	1.5	1.85	2.1	2.6	2.7	2.4	2.15	2.05	2.1	2.25	2.3	2.25	1.95	1.4	0.9	0.7	0.6	0.6	0.4	0.25

Since we don't have derivative information, we use natural spline interpolation. We write a MATLAB driver file for this example. Plotting the natural spline interpolation, we observe that the spline curve accurately recovers the top profile of the duck.

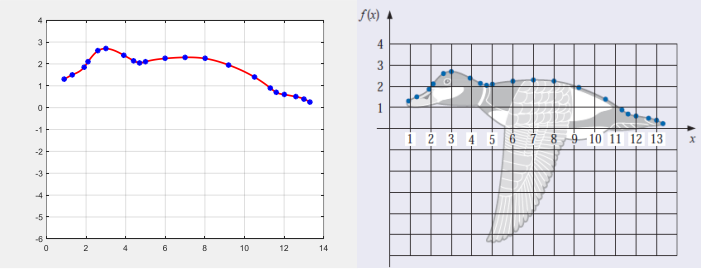

Figure 7. Natural and clamped cubic spline to approximate the top profile of duck

To use a clamped spline, we would need derivative approximations for the endpoints. Even if these approximations were available, we could expect little improvement because of the close agreement of the natural cubic spline to the curve of the top profile**.** For comparison, we also use Lagrange Interpolation

```
dataX = [0.9 1.3,1.9 2.1 2.6 3.0 3.9 4.4 4.7 5.0 6.0 7.0 8.0 ...
9.2 10.5 11.3 11.6 12.0 12.6 13.0 13.3];
dataY = [1.3 1.5 1.85 2.1 2.6 2.7 2.4 2.15 2.05 2.1 2.25 2.3 2.25 ...
1.95 1.4 0.9 0.7 0.6 0.5 0.4 0.25];
x = 0.9:0.01:13.3;
y = natural_spline(dataX,dataY,x);
yy = LagrangeInterpolation(dataX,dataY,x);
figure(1)
clf; plot(dataX,dataY,'b*','linewidth',2)
hold on
plot(x,y,'r-','linewidth',2)
plot(x,yy,'k-','linewidth',2)
axis([0,14,-2,7])
hold off
grid on
lgd = legend('Data Point','Natural Spline','Lagrange');  lgd.FontSize = 16;  lgd.Location = 'NorthWest';
```

Plotting the Lagrange interpolation, we observe that the 20th-degree polynomial oscillates wildly. It produces a very strange illustration of the back of a duck

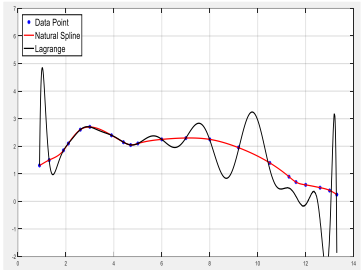

Figure 8. Natural and clamped cubic spline to approximate the top profile of duck

This example shows the superiority of the cubic spline interpolation, and we need the Lagrange Interpolation code

```
function [P,R,S] = LagrangeInterpolation(X,Y,XX)
if size(X,1) > 1;  X = X'; end
if size(Y,1) > 1;  Y = Y'; end
if size(X,1) > 1 || size(Y,1) > 1 || size(X,2) ~= size(Y,2)
  error('both inputs must be equal-length vectors')
end
N = length(X); pvals = zeros(N,N);
for I = 1:N
  pp = poly(X( (1:N) ~= i));
  pvals(I,😊 = pp ./ polyval(pp, X(i));
end
P = Y*pvals;  if nargin==3
  YY = polyval(P,XX);
  P = YY;
end
if nargout > 1
  R = roots( ((N-1):-1:1) .* P(1😢N-1)) );
  if nargout > 2
    S = polyval(P,R);
  end
end
```

Constructing a cubic spline to approximate the lower profile of the duck would be more difficult since the curve for this portion cannot be expressed as a function of x, and at certain points the curve does not appear to be smooth. These problems can be resolved by using separate splines to represent various portions of the curve, but a more effective approach to approximating curves of this type is considered in the next section.

Parametric Curves

The techniques we developed so far in this chapter cannot be used to generate curves of the form shown below because this curve cannot be expressed as a function $y = f(x)$. We will see how to represent general curves (even some hand-drawn curves) using parametric forms. This technique can be extended to represent general curves/surfaces in computer graphics. Given a set of data points

$$\begin{array}{ccccccc} x_0 & & x_1 & & x_2 & \dots & x_n \\ y_0 & & y_1 & & y_2 & \dots & y_n \end{array}$$

we can use a parameter t, and construct polynomial or piecewise polynomial approximation for

$$x = x(t), \text{ and } y = y(t),$$

To do this, we specify an interval $[t_0, t_n]$, with $t_0 < t_1 < \cdots < t_n$, and construct two approximation functions with $x_i = x(t_i)$, and $y_i = y(t_i),\ i = 0,1, \dots, n.$

Example: Construct a pair of Lagrange polynomials to approximate the curve show below

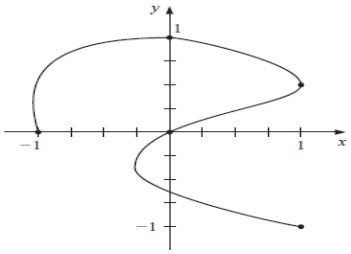

Figure 9. The curve parametric represents some points

There are five points, so we choose the points $\{t_i\}_{i=0}^{4}$ equally spaced in [0,1]:

i	0	1	2	3	4
t_i	0	0.25	0.5	0.75	1
x_i	1	0	1	0	-1
y_i	-1	0	0.5	1	0

Matlab code is

```
1332ata = [0,0.25,0.5,0.75,1];
dataX = [-1,0,1,0,1];
dataY = [0,1,0.5,0,-1];
qt = 0:0.001:1;
qx = LagrangeInterpolation(1332ata,dataX,qt);
qy = LagrangeInterpolation(1332ata,dataY,qt);
figure(1); clf
plot(dataX,dataY,'r*','linewidth',2)
hold
plot(qx,qy,'b-.','linewidth',2)
grid on
lgd = legend('Data Point','Lagrange');
lgd.FontSize = 16;
lgd.Location = 'NorthEast';
```

The Interpolation polynomials are

$$x(t) = 64t^4 - \frac{352}{3}t^3 + 60t^2 - \frac{14}{3}t - 1,$$

$$y(t) = -\frac{64}{3}t^4 + 48t^3 - \frac{116}{3}t^2 + 11t.$$

Plotting the parametric system produced the graph below:

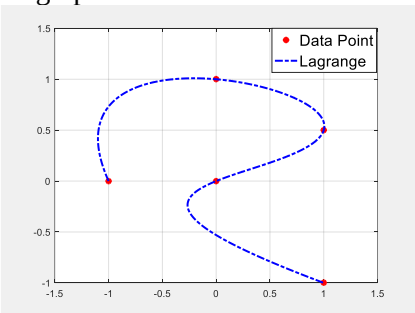

Figure 10. Plotting the parametric using Lagrange interpolation.

For this example, we can also use the natural cubic Spline interpolation for the parametric system

```
1332ata = [0,0.25,0.5,0.75,1];
dataX = [-1,0,1,0,1];
dataY = [0,1,0.5,0,-1];
qt = 0:0.001:1;
qx = LagrangeInterpolation(1332ata,dataX,qt);
```

```
qy = LagrangeInterpolation(1333ata,dataY,qt);
qxS = natural_spline(1333ata,dataX,qt);
qyS = natural_spline(1333ata,dataY,qt);
figure(1); clf
plot(dataX,dataY,'r*','linewidth',2)
hold
plot(qx,qy,'b-.','linewidth',2)
plot(qxS,qyS,'k-.','linewidth',2)
grid on
lgd = legend('Data Point','Lagrange','Natural Spline');
lgd.FontSize = 16;
lgd.Location = 'NorthEast';
```

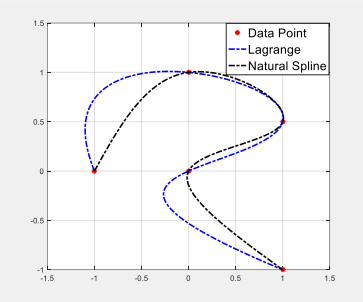

Figure 10. Plotting the data points using both Lagrange interpolation and Natural spline.

CONCLUSION

In this paper, there are many concepts are used to explain Interpolation and Lagrange Polynomials and its approximation theorem, also it is discussed the langrage interpolation Error and Cubic Spline Interpolation and its error, also we added and modified some algorithms related to them by using Matlab, final we discuss also Error bound for Clamped Cubic Splines and parametric cubic

REFERENCES

1. Harris L. Lagrange polynomials, reproducing kernels and cubature in two dimensions. Journal of approximation theory. 2015;195:43-56.
2. McKinley S. Levine, Cubic spline interpolation. College of the Redwoods, 1998;45(1):1049-1060.
3. Trosset M. Optimal shapes for kernel density estimation. Communications in statistics-theory and methods, 1993;22(2):375-391.
4. Milne J. Fields and Galois theory (v4. 60). order, 2018;3:138.

مراجعة التطبيقات الجديدة باستخدام الاستيفاء والتقريب متعدد الحدود

عبدالفتاح البركي[1]، وفاء ابوزنيد[2]، علي البركي[3]

[1]قسم الرياضيات، كلية العلوم، جامعة الزيتونة، ترهونة، ليبيا
[2]قسم الرياضيات، كلية التربية، جامعة الزيتونة، ترهونة، ليبيا

المستخلص

تحلل هذه الدراسة البنية الرياضية للتدخل لضمان مرور الدالة الناتجة عبر جميع نقاط البيانات المعطاة. وفي الوقت نفسه، يهدف التقريب متعدد الحدود إلى إيجاد دالة متعددة الحدود تتطابق بشكل وثيق مع البيانات، ولكنها قد لا تمر عبر جميع النقاط ويتم تحديدها عادةً باستخدام المربعات الصغرى. تتمتع كلتا الطريقتين بمزاياهما ويتم استخدامهما في مجالات مختلفة، اعتمادًا على المتطلبات المحددة للمشكلة المطروحة. كما يستكشف البحث نظرية خطأ التدخل لاغرانج، مما يوفر رؤى حول دقة وحدود تقنيات التدخل. يساعد فهم هذه الطرق وخصائص خطأها في اختيار النهج المناسب لمختلف التطبيقات العملية.

الكلمات الدالة. حدود لاجرانج، استيفاء المنحنيات المكعبة، الفروق المقسمة، المنحنيات البارامترية.