IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 5, Issue 6, Ver. II (Nov -Dec. 2015), PP 98-107
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Real-Time Reconstruction of Discrete-Time Signals in High-Performance Digital Control Systems: Analytical Evaluation of Common Interpolation Techniques

A. Mumuni[1], F. Mumuni[2]
[1] *(Electrical/Electronics Engineering Department, Cape Coast Polytechnic, Cape Coast, Ghana)*
[2] *(Electrical Engineering Department, University of Mines and Technology, Tarkwa, Ghana)*

Abstract: *Complex computational problems arising in control systems that employ digital signal processing for control tasks such as precision motion control, real-time processing of sensor data and machine vision are tedious, requiring significant amounts of calculation of mathematical functions. In addition, such problems need to be solved in a strictly limited time, so the mathematical computations must be optimized for speed. One of the most important operations performed by such systems is the reconstruction of digital signals. Traditionally, in high-end applications, this is done using advanced interpolation techniques. The computation-intensive requirements of such applications require the formulation of the interpolation problem in a way that is efficient for computer execution. However, the choice of interpolation technique is not always straight-forward, as it depends on the nature and size of the problem. The present paper undertakes analytical study of the common interpolation techniques and synthesizes corresponding algorithms for computing them. A comparative analysis of the various interpolation methods provides an accessible means for selecting the most suitable algorithm for a given problem.*

Keywords: *Computational Complexity, Digital Signal Reconstruction, DSP, Interpolation*

I. Introduction

In digital control systems, very often, there is a need to reconstruct digital signals after processing to obtain original analog signals from the discrete-time samples. In practice, this can be achieved by a number of techniques. These include simple sample-and-hold, fractional delay filtering, classical interpolation techniques, as well as frequency domain methods [1-2]. In low-end applications, reconstruction of digital signals is carried out using sample-and-hold circuits to achieve a zero-order hold digital to analog conversion [3-4]. In applications where high precision is required, polynomial interpolation techniques may be used. The simplest form of interpolation is linear interpolation or first-order hold, which approximates values between successive samples by straight lines [5]. In situations where linear interpolation does not meet the required accuracy, higher order interpolation formulas are used [2, 6].

Reconstructing analog signals from digitized signals using higher order polynomial interpolation methods requires a significant amount of computational resources. In practical applications, it is often necessary to develop algorithms which reduce all calculations to a sequence of arithmetic and logical operations necessary for computer execution. At the same time, it is also important to pay close attention to the amount of computational time required to solve these problems. In this regard, the quality of algorithms is assessed on the basis of the order of growth of time needed to solve the problem. This figure of merit, in technical literature, is known as the computational complexity of an algorithm. Thus, computational complexity, in this context, is the time spent in computing solutions as a function of the problem size (i.e. input data size and information content). It is determined largely by the number of elementary operations needed to solve a particular problem. Despite the increasing speed of modern computers, the complexity of advanced interpolation algorithms is still a key issue in many engineering applications [7]. Practical implementation of many mathematical solutions has been impeded by the computational complexity of the mathematical algorithms [8].

The interpolation problem consists of converting discrete data, defined only at discrete points such as (x_0, y_0), (x_1, y_1), ... (x_n, y_n), to a continuous output (Fig. 1). In essence, this amounts to finding a smooth curve that passes through all the discrete points (x_i, y_i). Thus, there is the need to determine the values of the function at points which do not coincide with the given points (i.e. for $x \neq x_i$). This means finding a function $f(x)$ that satisfies the interpolation condition

$$f(x_i) = f_i = y_i \tag{1}$$

The points x_i are called nodes of interpolation.

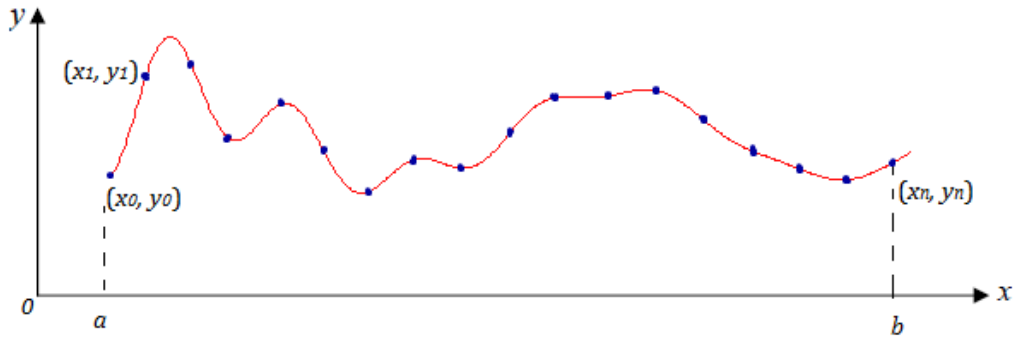

Fig. 1. Interpolation of discrete data.

The most straightforward method of doing this is polynomial interpolation, where an n[th] order polynomial that passes through $n+1$ data points is solved:

$$\varphi(x) = P_n(x_i) = C_0 + C_1 x + C_2 x^2 + ... + C_n x^n \tag{2}$$

This approach is based on Weierstrass approximation theorem [9-11], which stipulates that any segment $[a,b]$ of a function $P(x)$ can be sufficiently well approximated by an analytical expression which is an algebraic polynomial. To construct the polynomial $P_n(x),$ it is necessary to find the coefficients $C_0, C_1, C_2, ...C_n.$ The coefficients are determined from the interpolation condition $P_n(x_i) = y_i,\ i = 1, 2, ..., n$. In practice, however, the number of nodes may be very large; leading to extreme difficulty in solving (2) by classical methods. The complexity of solving (2) by Gaussian method, for example, is of the order $O(n^3)$ [12-14]. In practical situations, the desired interpolation polynomial is constructed without solving this system. A number of techniques have been developed for doing this [15-18].

This work investigates common interpolation techniques from the point of view of realizing efficient algorithms for reconstructing discrete-time signals. The main objective of this analytical study is to provide accessible means of comparing the computational complexity of common interpolation techniques with the view of finding the most appropriate choice for a particular problem.

II. Classical Interpolation Techniques

Notwithstanding the proliferation of many interpolation algorithms, classical interpolation methods are still very popular. One reason for their popularity is that they have been developed long ago and are well-studied, they are efficient, and have proven reliable for practical applications. Also, many of the newer techniques, despite their computational efficiency, from a practical point of view of software implementation simplicity, have intricate formulations, making them less attractive than conventional interpolation techniques.

2.1. Lagrange technique

A more efficient alternative to reconstructing sampled data by solving the interpolation problem in canonical form is Lagrange polynomial formula [19]. The Lagrange interpolation method may also be used to design FIR, IIR, as well as adaptive Filters [20-21]. To construct the Lagrange polynomial for an arbitrary part of a segment $[a,b]$ containing $k+1$ points $x_i,\ x_{i+1}, ...,\ x_{i+k}$, an algebraic polynomial $Q_{km}(x_j)$ is chosen such that $Q_{km}(x_j) = \begin{cases} 1, & j = m \\ 0, & j \neq m \end{cases}$. The polynomial is zero for every x except $x = x_m$ (i.e. $x_i,\ x_{i+1}, ..., x_{i+k}$ are the roots of this polynomial). So the polynomial $Q_{km}(x_j)$ can be written in the form

$$Q_{km}(x) = c(x - x_i)(x - x_{i+1})...(x - x_{m-1})(x - x_{m+1})...(x - x_k) \tag{3}$$

The constant c is determined from the condition $Q_{km}(x_j) = 1$ by substituting $x = x_m$ in (3)

$$c = \frac{1}{(x_m - x_i)(x_m - x_{i+1})...(x_m - x_{m-1})(x_m - x_{m+1})...(x_m - x_k)} \tag{4}$$

and

$$Q_{km}(x) = \frac{(x - x_i)(x - x_{i+1})...(x - x_{m-1})(x - x_{m+1})...(x - x_k)}{(x_m - x_i)(x_m - x_{i+1})...(x_m - x_{m-1})(x_m - x_{m+1})...(x_m - x_k)} \tag{5}$$

$Q_{km}(x)$ are known as Lagrange basis polynomial.

The Lagrange polynomial for the chosen segment is, thus, written as

$$P_k(x) = \sum_{m=i}^{i+k} \frac{(x-x_i)(x-x_{i+1})...(x-x_{m-1})(x-x_{m+1})...(x-x_k)}{(x_m-x_i)(x_m-x_{i+1})...(x_m-x_{m-1})(x_m-x_{m+1})...(x_m-x_k)} f_m \tag{6}$$

If it is required to interpolate the whole interval $[a,b]$, then $i = 0$, $k = n$, and the Lagrange polynomial has the general form

$$P_n(x) = \sum_{i=0}^{n} \frac{(x-x_0)(x-x_1)...(x-x_{n-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_0)(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)} f_i = \sum_{i=0}^{n} f_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)} \tag{7}$$

Thus, the degree of the polynomial is n and for $x \neq x_i$ all terms in the summation operator reduce to zeros except the $i = j^{th}$ terms, which are f_i, thereby satisfying the interpolation condition (1).

As the terms of the factor $Q_{ni}(x)$ in relation (7) depend only on the choice of nodes x_i and the points x, and not on the function $f(x)$, the factors f_i can be computed only once and used in the construction of interpolation polynomials.

Computational complexity Lagrange interpolation algorithm

The constructive validation of Lagrange interpolation technique provides a method to construct the interpolating polynomial in the form expressed in (7). The method is stated in the following steps:

1. Computation of the Lagrange basis $Q_{ni}(x) = \prod_{i=0}^{n} \frac{(x-x_j)}{(x_i-x_j)}$
2. Multiplication of each i^{th} basis by the corresponding functional value f_i
3. Summation of each product term $Q_{ni}(x) f_i$

Recalling that the Lagrange coefficients $Q_{ni}(x)$ of a function $y_i = f(x_i)$ for any given segment $[a,b]$ are determined only by the nodes x_i and the points x, where it is required to compute the value of the polynomial. So if the calculations are carried out according to (5) above, the calculation of each coefficient (consisting of $n+1$ nodes) of the Lagrange basis requires $n-1$ elementary multiplications, n divisions, and $2n$ additions. Obviously, it is necessary to make $n+1$ iterations to compute all the polynomials (i.e. $P_0(x)$, $P_1(x)$, …, $P_n(x)$). Consequently, the computation of the Lagrange basis involves:

- $(n-1)(n+1) = n^2 - 1$ multiplications
- $2n(n+1) = 2n^2 + 2n$ additions
- $n(n+1) = n^2 + n$ divisions

The number of operations in step two is $(n+1)$ multiplications, and in step three n additions.

So there are, in total, $2n^2 + 3n$ additions, $n^2 + n$ multiplications, and $n^2 + n$ divisions.

2.2. Newton Interpolation Method

The Lagrange interpolation polynomial is inconvenient in cases where the nodes may need to be updated. If the task is formulated such that the addition of an extra point would require only the addition of a single polynomial of degree $(n+1)$ to the n^{th} degree Lagrange polynomial, then this addend can be found by taking the difference of the two Lagrange polynomials of degree $(n+1)$ and n. Simple transformations of the Lagrange interpolation polynomial will lead to the following expression:

$$P_n(x) = P_0(x) + \sum_{i=1}^{n} [(P_i(x) - P_{i-1}(x))] \tag{8}$$

where $P_i(x)$ is Lagrange polynomial of degree i.

If we let $Q_i(x) = P_i(x) - P_{i-1}(x)$, then $Q_i(x)$ is a polynomial of degree i whose value is zero at $x = x_0$, $x = x_1$, $x = x_2$, ... , $x = x_{i-1}$. Therefore, the polynomial $Q_i(x)$ can be written as $Q_i(x) = C_i(x - x_0)(x - x_1)...(x - x_{i-1})$, where C_i are coefficients at $x = x_0$, $x = x_1$, $x = x_{i-1}$. Therefore, the modified Lagrange polynomial (8) can be rewritten as $L_n(x) = L_{n-1}(x) + C_n(x - x_0)(x - x_1)...(x - x_{n-1})$.

Recursively expressing all the polynomials L_1, L_2, etc. in a similar way leads to another form of representing the Lagrange interpolation polynomial, which is called Newton interpolation polynomial [22]:

$$N_n(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + ... + C_n(x - x_0)(x - x_1)...(x - x_n) = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1}(x - x_j) \qquad (9)$$

$C_1, C_2, ..., C_n$ are known as Newton coefficients.

The Newtonian bases are $1, (x - x_0), (x - x_0)(x - x_1), ...(x - x_0)(x - x_0)...(x - x_{n-1})$.

In the general formulation of the Newton interpolation polynomial (9) the Newton coefficients C_i are computed from the interpolation condition (1):

$$C_i = \left[f_i - C_0 - \sum_{j=0}^{i-1} C_j \prod_{k=0}^{j-1}(x_i - x_k) \Big/ \prod_{k=0}^{j-1}(x_i - x_k) \right] \qquad (10)$$

Calculating the coefficients by this approach is computationally demanding. In practice, it is often appropriate to compute the coefficients using Newton's divided differences [23]. If $x_0 \neq x_1 \neq x_2 \neq ... \neq x_k$, then, the divided differences are defined as follows [24]:

Zero[th] order divided difference $f[x_i] = f(x_i) = f_i$, $i = 0, 1, 2, ...n$. The first order divided difference is

$f[x_i, x_{i+1}] = \dfrac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$, and the second order divided difference is

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

It can be seen that higher order divided differences are defined recursively using lower-order ones. Thus, k^{th} order divided difference for the section $[x_i, x_{i+k}]$ can be found through $(k-1)^{th}$ order divided difference by the recursive formula $f[x_i, x_{i+1}, x_{i+2}, ..., x_{i+k}] = \dfrac{f[x_{i+1}, x_{i+2}, ..., x_{i+k}] - f[x_i, x_{i+1}, ...x_{i+k-1}]}{x_{i+k} - x_i}$,

where $k = 1, 2, ..., n$; $i = 0, 1, ..., n - k$; $n -$ degree of the polynomial.

It is often reasonable to calculate the divided differences only for neighboring values of the discrete data. In this case, a higher order divided difference is calculated from two preceding ones of lower orders [25]. Hence the Newton coefficients $C_0, C_1, C_2, ...C_n$ can be expressed through divided differences as follows:

$C_0 = f_0 = f[x_0]$,

$C_1 = \dfrac{f_1 - f_0}{x_1 - x_0} = \dfrac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1]$,

$C_2 = \dfrac{\left(\dfrac{f_2 - f_1}{x_2 - x_1}\right) - \left(\dfrac{f_1 - f_0}{x_1 - x_0}\right)}{x_2 - x_0} = \dfrac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f[x_0, x_1, x_2]$,

and, $C_k = f[x_0, x_1, ..., x_k]$.

Thus, for the purpose of interpolation function on the interval $[x_i, x_{i+k}]$ the Newton polynomial, based on divided differences, can be written in the form

$$N_k(x) = f_i + f[x_i, x_{i+1}](x - x_i) + f[x_i, x_{i+1}, x_{i+2}](x - x_i)(x - x_{i+1}) + ...$$
$$+ f[x_i, x_{i+1}, ..., x_{i+k}](x - x_i)(x - x_{i+1})...(x - x_{i+k-1}) \qquad (11)$$

Computational complexity of Newtonian interpolation algorithm

Computing the value of the interpolation polynomial in the Newton form can be carried out in the following steps:

1. Finding the Newton coefficients C_i through divided differences

2. Computing the value of the polynomials (9) using the Newton coefficients $C_0, C_1, C_2, \ldots C_n$

To perform the first operation, one needs to carry out $n+1$ additions and $n+1$ divisions for each node x_i. This requires n iterations each for each of the arithmetic operations. Hence, the number of additions would be $\sum_{i=1}^{n}(n-i+1)=\sum_{i=1}^{n} i=\frac{n(n+1)}{2}$, and the number of division operations would also be same.

Calculation of the polynomial P_n (step 2) requires $n+1$ additions through n iterations, plus additional n summations for the terms $C_i\prod\left(x-x_j\right)$. Thus, for interpolation of $n+1$ nodes, the number of addition operations:

$$n+\sum_{i=1}^{n}(n-i+1)=n+\sum_{i=1}^{n} i=n+\frac{n(n+1)}{2}=\frac{n(n+3)}{2}.$$

Similarly, there would be $n+1$ multiplications through n iterations:

$$\sum_{i=1}^{n}(n-i+1)=\sum_{i=1}^{n} i=\frac{n(n+1)}{2}.$$

Thus, total number of each operation is n^2+2n additions, $\frac{1}{2}\left(n^2+n\right)$ multiplications, and $\frac{1}{2}\left(n^2+n\right)$ divisions.

2.3. Spline Interpolation

The Spline is a form of piecewise polynomial method used for interpolating discrete points. In a piecewise polynomial interpolation the segment $[a,b]$ is divided into sub-segments having a smaller number of interpolation nodes. Lower degree interpolation polynomial can then be used on each of these segments. The most widely used piecewise interpolation method is cubic spline interpolation. Cubic spline has a feature that each partial interval is represented by a third degree polynomial, and on the whole interval of interpolation, it is continuous together with its first and second derivatives [26-28].

On each interval $[x_{i-1}, x_i]$ the cubic spline has the form [29]

$$S_i(x)=a_i+b_i(x-x_{i-1})+c_i(x-x_{i-1})^2+d_i(x-x_{i-1})^3=f_i,\ x\in[x_{i-1}, x_i] \tag{12}$$

Thus, the spline satisfies the Lagrange interpolation condition

$$S_i(x_i)=f_i,\ \ i=0,1,2,\ldots n \tag{13}$$

The spline (12) on each of the segments $[x_{i-1}, x_i]$, $i=1,2,\ldots n$ is defined by four coefficients, and, therefore, for its construction on the whole interval $[a,b]$ 4n equations are needed to determine the coefficients. Condition (13) yields 2n equations. As noted, the spline $S_i(x)$, together with its first and second differentials, is continuous in all internal nodes. The condition of continuity of the differentials $S'_i(x)$ and $S''(x)$ gives $2(n-1)$ additional equations. Thus, there are $2n+2(n-1)=4n-2$ equations. The two missing conditions are determined in different ways. They may be obtained from the so-called boundary conditions [30-31], which are assigned on the basis of physical properties or other considerations related to the interpretation of the features of the discrete data. These include reference to first or second derivatives of the extremes of the intervals, or periodicity conditions. To obtain the remaining two unknowns, here we use the Free Boundary Conditions:

$$S''(a)=S''(b)=0 \tag{14}$$

The result is a system of $4n$ linear equations for determining the unknown coefficients a_i, b_i, c_i, d_i, $i=1,2,\ldots,n$. Preliminary analysis of these equations and a number of simple transformations lead to a fairly simple sequence of operations for finding the values of these coefficients as follows.

On the interval $[x_{i-1}, x_i]$ the second derivative of the spline $S''_i(x)$ is a straight line with slope m_i, and can be written in the form

$S''_i(x) = \gamma_i + m_i(x - x_i)$, where γ_i is a constant. Since $S''(x)$ is a linear function on $[x_{i-1}, x_i]$, its form is completely determined by its two extreme values γ_{i-1} and γ_i at the ends of the interval $[x_{i-1}, x_i]$.

$m_i = \dfrac{\gamma_i - \gamma_{i-1}}{h_i}$, So

$S''(x) = \gamma_{i-1}\dfrac{x_i - x}{h_i} + \gamma_i \dfrac{x - x_{i-1}}{h_i}$ for $x \in [x_{i-1}, x_i]$, where $h_i = x_i - x_{i-1}$, $\gamma_i = S''(x_i)$.

Integrating $S''(x)$ twice gives another form of the cubic spline representation

$$S(x) = \gamma_{i-1}\frac{(x_i - x)^3}{6h_i} + \gamma_i \frac{(x - x_{i-1})^3}{6h_i} + A_i(x_i - x) + B_i(x - x_{i-1}) \tag{15}$$

Where A_i and B_i are the constants of integration. According to the interpolation condition $S_i(x_i) = f_i$, consequently, letting $x = x_i$, yields $S_i(x_i) = f_i = \gamma_i \dfrac{h_i^2}{6} + B_i h_i$ or $B_i = \dfrac{f_i}{h_i} - \dfrac{h_i}{6}\gamma_i$.

Also if we let $x = x_{i-1}$ and noting that $S_i(x_{i-1}) = S_{i-1}(x_{i-1})$, we obtain $S_{i-1}(x_{i-1}) = f_{i-1} = \gamma_{i-1}\dfrac{(h_i)^3}{6h_i} + A_i h_i$. Thus,

$A_i = \dfrac{f_{i-1}}{h_i} - \dfrac{h_i}{6}\gamma_{i-1}$.

Substituting A_i and B_i in equation (15) gives

$$S_3(x) = \frac{x_i - x}{h_i} f_{i-1} + \frac{x - x_{i-1}}{h_i} f_i + \frac{(x_i - x)^3 - h_i^2(x_i - x)}{6h_i}\gamma_{i-1} + \frac{(x - x_{i-1})^3 - h_i^2(x - x_{i-1})}{6h_i}\gamma_i \tag{16}$$

From (16) the one-sided derivatives for $x_1, x_2, \ldots, x_{n-1}$ using property that the spline is continuous over $[x_{i-1}, x_i]$ can be obtained:

$$S'(x_i - 0) = \frac{h_i}{6}\gamma_{i-1} + \frac{h_i}{3}\gamma_i + \frac{f_i - f_{i-1}}{h_i} \tag{17}$$

$$S'(x_i + 0) = -\frac{h_{i+1}}{3}\gamma_i - \frac{h_{i+1}}{6}\gamma_{i+1} + \frac{f_{i+1} - f_i}{h_{i+1}} \tag{18}$$

Equating (17) and (18) for $i = 1, 2, ..., n-1$, we obtain $n-1$ equations of the form

$$\frac{h_i}{6}\gamma_{i-1} + \frac{h_i + h_{i+1}}{3}\gamma_i + \frac{h_{i+1}}{6}\gamma_{i+1} = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \tag{19}$$

This equation can be written in the matrix form [32] $A\gamma = q$ with $n-1$ unknowns $\gamma_i\ (i = 1, 2, ..., n-1)$, where

$$A = \begin{pmatrix} \frac{h_1 + h_2}{3} & \frac{h_2}{6} & 0 & & 0 & \cdots & 0 & 0 & 0 \\ \frac{h_2}{6} & \frac{h_2 + h_3}{3} & \frac{h_3}{6} & & 0 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 \cdots & \frac{h_{n-2}}{6} & \frac{h_{n-2} + h_{n-1}}{3} & \frac{h_{n-2}}{3} \\ 0 & 0 & 0 & 0 \cdots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1} + h_n}{3} \end{pmatrix},$$

$\gamma = (\gamma_1, \gamma_2, ..., \gamma_{n-1})^T$, and $q = \left(\dfrac{f_2 - f_1}{h_2} - \dfrac{f_1 - f_0}{h_1}, ..., \dfrac{f_n - f_{n-1}}{h_n} - \dfrac{f_{n-1} - f_{n-2}}{h_{n-1}}\right)^T$.

Thus, in matrix form the equation is

$$\begin{pmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ c_2 & a_2 & b_2 & 0 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & c_{n-2} & a_{n-2} & b_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & c_{n-1} & a_{n-1} \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{n-2} \\ \gamma_{n-1} \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_{n-2} \\ q_{n-1} \end{pmatrix} \tag{20}$$

$c_i = \frac{h_i}{6}, \quad a_i = \frac{h_i + h_{i+1}}{3}, \quad b_i = \frac{h_{i+1}}{6}, \quad q_i = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i}.$

Solving the spline interpolation requires solving the tridiagonal matrix A [33]. The solution of such systems can be organized in a way that does not include the zero elements of the matrix, thereby saving memory and decreasing the required amount of computation. This method in mathematics literature is called Thomas algorithm [34]. Thomas algorithm consists of two stages - forward substitution and backward substitution. Forward substitution consists in calculating the auxiliary coefficients α_i and β_i, whereby each unknown γ_i is expressed in terms of γ_{i+1}:

$$\gamma_i = (\gamma_{i+1})\alpha_i + \beta_i, \; i = 1, 2, ..., n-1 \tag{21}$$

From the first equation of (20), $\gamma_1 = -\frac{b_1}{a_1}\gamma_2 + \frac{q_1}{a_1}$. But from (21) $\gamma_1 = -\alpha_1\gamma_2 + \beta_1$.

Thus, equating the two relations equations gives

$$\begin{cases} \alpha_1 = -\frac{b_1}{a_1} \\ \beta_1 = \frac{q_1}{a_1} \end{cases} \tag{22}$$

Now putting $\gamma_1 = -\alpha_1\gamma_2 + \beta_1$ in the second equation of (20):

$c_2(\alpha_1\gamma_2 + \beta_1) + a_2\gamma_2 + b_2\gamma_3 = q_2$. Thus, γ_2 can be expressed through γ_3:

$\gamma_2 = \frac{q_2 - b_2\gamma_3 - c_2\beta_1}{c_2\alpha_1 + a_2}$ or $\gamma_2 = \alpha_2\gamma_3 + \beta_2,$

and so $\alpha_2 = -\frac{b_2}{c_2\alpha_1 + a_2}$ and $\beta_2 = \frac{q_2 - c_2\beta_1}{c_2\alpha_1 + a_2}.$

Similarly, all auxiliary coefficients α_i and β_i can be calculated by the recursive formulas

$$\begin{cases} \alpha_i = -\frac{b_i}{c_i\alpha_{i-1} + a_i} \\ \beta_i = \frac{q_i - c_i\beta_{i-1}}{a_i + c_i\alpha_{i-1}} \end{cases} \tag{23}$$

If we let $e_i = a_i + c_i\alpha_{i-1}$, then $\alpha_i = -\frac{b_i}{e_i}$ and $\beta_i = \frac{q_i - c_i\beta_{i-1}}{e_i}.$

Backward substitution consists of recursively calculating γ_i. This involves, first, finding γ_{n-1} using equation (21) and the last equation of the tridiagonal matrix (20).

$$\gamma_{N-1} = \alpha_{N-1}\gamma_N + \beta_{N-1} \tag{24}$$

$$q_N = a_N\gamma_N + c_N\gamma_{N-1} \tag{25}$$

Eliminating γ_{N-1} from the two equations gives

$$\gamma_N = \frac{q_N - \alpha_N\beta_{N-1}}{a_N + c_N\alpha_{N-1}} \tag{26}$$

Then, using the formula (21), and the previously calculated coefficients α_i and β_i according to formulas (22) and (23), we can successively calculate all unknowns $\gamma_{n-1}, \gamma_{n-2}, \ldots, \gamma_1$.

Computational Complexity Of Cubic Spline Interpolation Algorithm

From the above analysis, the solution of the spline S(x) requires only finding h_i, A_i, B_i, q_i and γ_i. The steps involved in calculating can be summarized as follows:

1. Computation of h_i requires $n-1$ additions
2. Computation of A_i and B_i both require $2(n-1)$ divisions, $n-1$ multiplications and $n-1$ additions each
3. Computation of q_i requires $2(n-1)$ additions and $n-1$ divisions
4. Computation of γ_i involves a number of steps:
   - Computation of α_1 and β_1 as per (22). This requires 2 division operations
   - Computation of α_i and β_i ($i = 2, 3..., n-1$) as per (23). This requires $2(n-2)$ additions, $2(n-2)$ multiplication, and $2(n-2)$ divisions
   - Computation of γ_n as per (26). This requires 2 additions, 2 multiplications, and 1 division operation
   - Computation of γ_i ($i = n-1, n-2, ..., 1$) as per (21). This requires $n-1$ additions, and $n-1$ multiplication

Hence, the total number of operations is $7n-6$ additions, $5n-3$ multiplications, and $7n-4$ divisions. This gives a total operation count of $19n-13$.

Noting that the coefficients α_i and e_i are defined solely by the choice of nodes and do not depend on the functional values y_i, computation can be reduced if there is the need to solve a series of problems involving different functional values at predetermined nodes. In this situation, α_i and e_i are calculated only once. Only β_i are computed for subsequent problems. This reduces the number of operations by $3(n-1)$, resulting in $16n-10$ total number operations. Thus, the total number of operations involved in computing the spline $S(x)$ is $18n-17$. If we are carrying out a series of analysis involving the same nodes, subsequent computation are reduced to $15n-14$.

III. Results And Discussion

Table 1 summarizes the results of the foregoing analysis. The last column is the total count of all operations, N. This is plotted against input data size, n in Fig. 2 with a logarithmic scale on both axes. It is worth noting that in most practical cases, addition, multiplication and division operations do not cost the same. But since the durations of these basic operations are known, it is usually straight-forward to compute the total cost in terms of the number of machine cycles.

Table 1. Summary of computational complexity of classical interpolation methods

	Additions	**Multiplications**	**Divisions**	**Total**
Lagrange	$2n^2+3n$	n^2+n	n^2+n	$2n^2+3n$
Newton	n^2+2n	$0.5\left(n^2+n\right)$	$0.5\left(n^2+n\right)$	$4n^2+5n$
Cubic Spline	$7n-6$	$5n-3$	$7n-4$	$19n-13$

In high performance digital control systems with real-time requirements, the interpolation problem usually needs to be solved within a strictly short duration, say a few milliseconds. The choice of a good interpolation algorithm allows modern DSP systems not only to provide high accuracy and performance, but also extend the magnitude of tasks that can be performed. For instance, is if a task required 100 milliseconds and a digital signal processor with processing speed of 1 million floating point operations per second (MFLOPS) is employed, the maximum number of discrete inputs that can be interpolated would be 10000, 400, and 500 for the spline, Lagrange and Newton, respectively (Fig. 2). It is obvious that for the spline algorithm, increasing processor speed by a factor of 100 results in an increase problem size in the order of 100. For the Lagrange and Newton interpolators, a speed increase by 100 times results in only a 10 times increase in problem size.

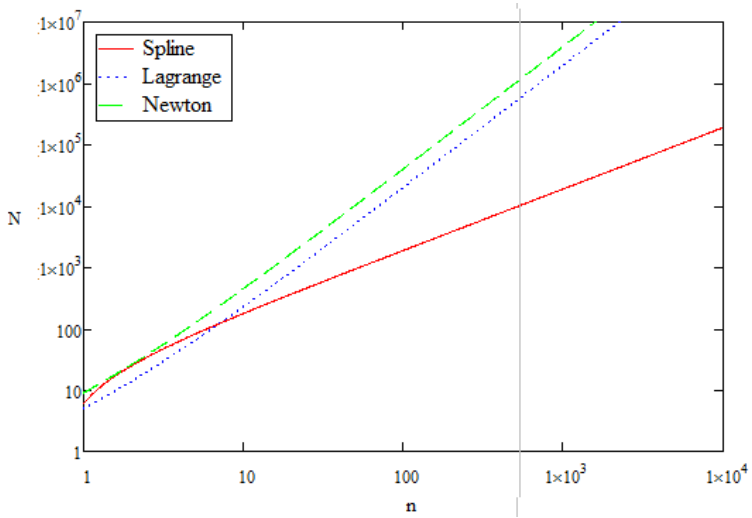

Fig. 2. Comparison of computational complexity of common interpolation methods.

On the basis of this analysis, it can be said that the Cubic Spine interpolation algorithm is the best choice (for data points greater than 8). The Lagrange formula also outperforms Newton's. However, as noted above, if there would be the need to update interpolating data, then Newton's would be a better choice.

In general, the complexity of an algorithm is evaluated in order p of magnitude of input data n, and is denoted $O(n^p)$, so called big-O notation [35-36]. In this representation, only the fastest growing (highest order) term is considered; all terms of lower order are ignored. For example, the time complexity of the Lagrange algorithm is equal to $4n^2 + 5n$, but its computational complexity is of order n^2 and is expressed as $O(n^2)$. Likewise, Newton interpolation formula is $O(n^2)$ and the cubic spline is $O(n)$ However, there are situations where it is useful to know the exact relationship of the dependence of complexity on input data. This is especially so when a small number of input data is to be interpolated, or when the interpolation formula contains a large constant which can have a significant effect on the quality of the algorithm. On the graph of Fig. 2, it can be observed that the Lagrange formula is the best for a task involving 2 to 8 data inputs. Other factors, such as available memory resources, required accuracy, and the nature of signals, may also influence the choice of algorithm. For instance, if the signal is to be sampled at a much higher rate than the analog signal frequency, then the Lagrange interpolation formula is preferred to the Spline [26]. Also, some classes of algorithms may be less stable for certain types of problems.

IV. Conclusion

The study shows that a number of interpolation methods can be used to construct an approximate analog output from digital signals. The required accuracy can be achieved by all the above methods of polynomial interpolation. However, the practical implementation of the different interpolation techniques requires different computational resources. So in addition to the accuracy of approximation of the digitized signals, one should take into account the CPU time required to construct these signals. The present study attempted to analyze and compare the most popular classical interpolation methods used in digital signal processing and digital control systems for the purpose of reconstructing discrete-time signals. The results of the study can be used as a basis for selecting the most appropriate algorithm for a particular interpolation task. It is also expected that this study will serve as a basis for further investigation into the issues of effective use of interpolation algorithms for reconstructing discrete data. For example, further theoretical and experimental study

can be conducted on the effectiveness of parallel implementations of interpolation methods on computer systems of different architectures.

References

[1] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time signal processing, vol. 2* (Englewood Cliffs, NJ: Prentice Hall, 1989).

[2] T. Saramäki, Polynomial-Based Interpolation for Digital Signal Processing (DSP) and Telecommunication Applications, *IEEE CAS Distinguished Lecture Program*, 2002.

[3] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and systems*, *2nd ed.,* 1997.

[4] J. T. Olkkonen, and H. Olkkonen, Sampling and Reconstruction of Zero-Order Hold Signals by Parallel RC Filters, *Wireless Engineering and Technology*, *2(3),* (2011): 153.

[5] V. Stojanović, J. G. Proakis, and D. G. Manolakis, *Digital signal processing: Principles, Algorithms, and Applications*, *4 ed.* (Upper Saddle River, NJ: Prentice Hall, 2006).

[6] R. W. Schafer, and L. R. Rabiner, A digital signal processing approach to interpolation, *Proceedings of the IEEE*, *61(6*), (1973): 692-702.

[7] S. K. Narang, A. Gadde, and A. Ortega, Signal processing techniques for interpolation in graph structured data, *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013.

[8] A. Dutt, M. Gu, and V. Rokhlin, Fast algorithms for polynomial interpolation, integration, and differentiation, *SIAM Journal on Numerical Analysis*, *33(5),* (1996): 1689-1711.

[9] A. Pinkus, Weierstrass and approximation theory, *Journal of Approximation Theory*, *107(1),* (2000): 1-66.

[10] M. H. Stone, The generalized Weierstrass approximation theorem, *Mathematics Maqazine*, *21(5),* (1948): 237-254.

[11] D. Pérez, and Q.Yamilet, A survey on the Weierstrass approximation theorem, *Divulgaciones Matemáticas*, *16(1),* (2008): 231-247.

[12] K. Chalupka, C. K. Williams, and I. Murray, A framework for evaluating approximation methods for Gaussian process regression, *The Journal of Machine Learning Research*, *14(1*), (2013): 333-350.

[13] X. Tang, and Y. Feng, A New Efficient Algorithm for Solving Systems of Multivariate Polynomial Equations, *IACR Cryptology ePrint Archive*, 2005, 312.

[14] D. Andrén, H. Lars, and M. Klas. On the complexity of matrix reduction over finite fields, *Advances in Applied Mathematics*, *39(4),* (2007): 428-452.

[15] E. Meijering, A chronology of interpolation: from ancient astronomy to modern signal and image processing, *Proceedings of the IEEE*, *90(3),* (2002): 319-342.

[16] T. M. Lehmann, C. Gönner, and K. Spitzer, Survey: Interpolation methods in medical image processing, *IEEE Transactions on Medical Imaging*, *18(11),* 1999.

[17] B. A. Eckstein, Evaluation of spline and weighted average interpolation algorithms, *Computers & Geosciences*, *15(1),* (1989): 79-94.

[18] O. Abramov, and A. S. McEwen, An evaluation of interpolation methods for Mars Orbiter Laser Altimeter (MOLA) data, *Int. J. Remote Sensing*, *25(3),* (2004): 669–676.

[19] R. Séroul*, Programming for mathematicians*, Springer Science & Business Media, 2012.

[20] A. Krukowski, and İ. Kale , *DSP system design: Complexity reduced IIR filter implementation for practical applications*, Springer Science & Business Media, 2003.

[21] V. V. Vučković, The Reconstruction of the Compressed Digital Signal using the Lagrange Polynomial Interpolation, *Electronic Engineering*, 2008.

[22] M. Zhang, and W. Xiao, Construct and Realization of Newton Interpolation Polynomial Based on Matlab7, *Procedia Engineering*, *15*, (2011): 3831-3835.

[23] R. L. Burden, and J. D. Faires, *Numerical Analysis* (Stamford, CT: Thomson Learning. Inc., 2001).

[24] M. Abramowitz, and I. A. Stegun, *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*, *No. 55*, Courier Corporation, 1964.

[25] V. S. Ryaben'kii, and S. V. Tsynkov, *A theoretical introduction to numerical analysis*, CRC Press, 2006.

[26] L. R. M. Silva, C. A. Duque, and P. F. Ribeiro, Smart signal processing for an evolving electric grid, *EURASIP Journal on Advances in Signal Processing* , *1,* (2015): 1-13.

[27] A. Ferrer, L. Jordi, et al, Efficient cubic spline interpolation implemented with FIR filter, *International Journal of Computer Information Systems and Industrial Management Applications*, *vol. 5*, (2013): 98-105.

[28] T. Michaeli, Y. C. Eldar, and V. Pohl, Causal signal recovery from U-invariant samples, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2011.

[29] P. Z Revesz, Cubic Spline Interpolation by Solving a Recurrence Equation Instead of a Tridiagonal Matrix, *Mathematical Methods in Science and Engineering*, 2014.

[30] J. H. Mathews, and K.D. Fink, *Numerical Methods Using MATLAB*, *vol. 31* (Upper Saddle River, NJ: Prentice hall, 1999).

[31] M I. Syam, Cubic spline interpolation predictors over implicitly defined curves, *Journal of Computational and Applied Mathematics*, *157(2),* (2003): 283-295.

[32] S. Gao, Z. Zhang, and C. Cao, On a generalization of cubic spline interpolation, *Journal of Software*, *6(9)*, (2011): 1632-1639.

[33] V. Malyshkin, Parallel Computing Technologies, *13th International Conference, PaCT 2015*, *Proceedings*, *vol. 9251*, Petrozavodsk, Russia, 2015.

[34] J. D. Hoffman, and S. Frankel, *Numerical methods for engineers and scientists*, CRC press, 2001.

[35] E. Bach, and J. O. Shallit, *Algorithmic Number Theory: Efficient Algorithms*, *vol. 1*, MIT press, 1996.

[36] P. H. Dave, *Design and analysis of algorithms*, Pearson Education India, 2009.