A Recurrence Equation-Based Solution for the Cubic Spline Interpolation Problem

Peter Z. Revesz

***Abstract*—This paper presents a simple and fast recurrence equation-based method for solving the cubic spline interpolation problem. The computational complexity of the method is $O(n)$, where n is the number of measurements. The recurrence equation-based method is illustrated by an example that estimates the movement of a moving object. The paper also describes a MATLAB implementation of the new method and the use of cubic spline interpolation within the MLPQ database system.**

***Index Terms*—cubic spline, interpolation, recurrence equation, tridiagonal matrix.**

I. Introduction

Cubic spline interpolation is a widely-used polynomial interpolation method for functions of one variable [3]. Cubic splines can be described as follows. Let f be a function from $\mathcal{R}$ to $\mathcal{R}$. Suppose we know about f only its value at locations $x_0 < \ldots < x_n$. Let $f(x_i) = a_i$. Piecewise cubic spline interpolation of f is the problem of finding the b_i, c_i and d_i coefficients of the cubic polynomials S_i for $0 \leq i \leq n-1$ written in the form:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad (1)$$

where each piece S_i interpolates the interval $[x_i, x_{i+1}]$ and fits the adjacent pieces by satisfying certain smoothness conditions. Taking once and twice the derivative of Equation (1) yields, respectively the equations:

$$S_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2 \quad (2)$$

$$S_i''(x) = 2c_i + 6d_i(x - x_i) \quad (3)$$

Equations (1-3) imply that $S_i(x_i) = a_i$, $S_i'(x_i) = b_i$ and $S_i''(x_i) = 2c_i$. For a smooth fit between the adjacent pieces the cubic spline interpolation requires that the following conditions hold for $0 \leq i \leq n-2$:

$$S_i(x_{i+1}) = S_{i+1}(x_{i+1}) = a_{i+1}, \quad (4)$$

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) = b_{i+1} \quad (5)$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) = 2c_{i+1} \quad (6)$$

This paper is organized as follows. Section II review the usual solution for cubic splines by solving a tridiagonal matrix. Section III presents a new recurrence equation-based solution. Section IV illustrates the use of the recurrence equation by giving an example of interpolating the motion of a moving object. Section V gives another recurrence equation that eliminates the use of the h_i variables. Section VI presents the implementation of the cubic spline interpolation in the MATLAB system and the MLPQ constraint database system. Finally, Section VII presents some conclusions.

Peter Z. Revesz is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska 68588-0115. E-mail: revesz@cse.unl.edu, Phone: (1+) 402 472–3488

II. The Tridiagonal Matrix-Based Solution

In this section we review the usual tridiagonal matrix-based solution for cubic splines. Let $h_i = x_{i+1} - x_i$. Substituting Equations (1-3) into Equations (4-6), respectively, yields:

$$a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = a_{i+1} \quad (7)$$

$$b_i + 2c_i h_i + 3d_i h_i^2 = b_{i+1} \quad (8)$$

$$c_i + 3d_i h_i = c_{i+1} \quad (9)$$

Equation (9) yields a value for d_i, which we can substitute into Equations (7-8). Equation (9) is equivalent to:

$$d_i = \frac{1}{3h_i}(c_{i+1} - c_i). \quad (10)$$

Substituting the above value for d_i into Equations (8-9) we get:

$$a_i + b_i h_i + \frac{2c_i + c_{i+1}}{3} h_i^2 = a_{i+1} \quad (11)$$

$$b_i + (c_i + c_{i+1})h_i = b_{i+1} \quad (12)$$

The latter two equations can be simplified as:

$$a_{i+1} - a_i = b_i h_i + \frac{2c_i + c_{i+1}}{3} h_i^2 \quad (13)$$

$$b_{i+1} - b_i = (c_i + c_{i+1})h_i \quad (14)$$

Solving Equation (13) for b_i yields:

$$b_i = (a_{i+1} - a_i)\frac{1}{h_i} - \frac{2c_i + c_{i+1}}{3} h_i \quad (15)$$

which implies for $j \leq n-3$ the condition:

$$b_{i+1} = (a_{i+2} - a_{i+1})\frac{1}{h_{i+1}} - \frac{2c_{i+1} + c_{i+2}}{3} h_{i+1} \quad (16)$$

Substituting into Equation (14) the values for b_i and b_{i+1} from Equations (15-16) yields:

$$(a_{i+1}-a_i)\frac{1}{h_i}-(2c_i+c_{i+1})\frac{h_i}{3}+(c_i+c_{i+1})h_i = (a_{i+2}-a_{i+1})\frac{1}{h_{i+1}}-(2c_{i+1}+c_{i+2})\frac{h_{i+1}}{3}$$

The above can be rewritten as:

$$h_ic_i+2(h_i+h_{i+1})c_{i+1}+h_{i+1}c_{i+2} = \frac{3}{h_i}a_i-\left(\frac{3}{h_i}+\frac{3}{h_{i+1}}\right)a_{i+1}+\frac{3}{h_{i+1}}a_{i+2}$$

The above holds for $0 \le i \le n-3$. However, changing the index downward by one the following holds for $1 \le j \le n-2$:

$$h_{i-1}c_{i-1}+2(h_{i-1}+h_i)c_i+h_ic_{i+1} = \frac{3}{h_i}(a_{i+1}-a_i)-\frac{3}{h_{i-1}}(a_i-a_{i-1}) \tag{17}$$

which can be simplified as:

$$h_{i-1}c_{i-1}+2(h_{i-1}+h_i)c_i+h_ic_{i+1} = \frac{3}{h_{i-1}}a_{i-1}-\left(\frac{3}{h_{i-1}}+\frac{3}{h_i}\right)a_i+\frac{3}{h_i}a_{i+1} \tag{18}$$

The above is a system of $n-1$ linear equations for the unknowns c_i for $0 \le i \le n$. By Equation (3) $S_0''(x_0)=2c_0$ and by extending Equation (6) to $i=n-1$, $S_{n-1}''(x_n)=2c_n$.

The cubic spline interpolation allows us to specify several possible boundary conditions regarding the values of c_0, c_n. A commonly used boundary condition called a natural cubic spline assumes that $c_0 = c_n = 0$, which is equivalent to setting the second derivative of the splines at the ends to zero. Alternatively, in the clamped cubic spline interpolation, the assumed boundary condition is $b_0 = f'(x_0)$ and $b_n = f'(x_n)$ where the derivatives of the f at x_0 and x_n are known constants.

In addition, in solving a cubic spline a uniform sampling is also commonly assumed and available, that is, each h_i has the same constant value h. Then dividing Equation (18) by h yields:

$$c_{i-1}+4c_i+c_{i+1}=\frac{3}{h^2}(a_{i-1}-2a_i+a_{i+1}) \tag{19}$$

Since the values of a_i are known, the values of c_i can be found by solving the tridiagonal matrix-vector equation $Ax = B$. Under the natural cubic spline interpolation, we have:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \end{bmatrix}$$

the vector of unknowns is:

$$x = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

and the vector of constants is:

$$B = \begin{bmatrix} 0 \\ \frac{3}{h^2}(a_0-2a_1+a_2) \\ \vdots \\ \frac{3}{h^2}(a_{n-2}-2a_{n-1}+a_n) \\ 0 \end{bmatrix}.$$

Similarly, under the clamped spline interpolation we have:

$$A = \begin{bmatrix} 2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 2 \end{bmatrix}$$

the same vector of unknowns:

$$x = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

and the following vector of constants:

$$B = \begin{bmatrix} \frac{3}{h^2}(a_1-a_0)-\frac{3}{h}f'(x_0) \\ \frac{3}{h^2}(a_0-2a_1+a_2) \\ \vdots \\ \frac{3}{h^2}(a_{n-2}-2a_{n-1}+a_n) \\ \frac{3}{h}f'(x_n)-\frac{3}{h^2}(a_n-a_{n-1}) \end{bmatrix}.$$

Both the natural cubic spline and the clamped cubic spline boundary conditions yield a system of $n+1$ linear equations with only $n+1$ unknowns. Such a system normally yields a unique solution except in some special cases. Moreover, either system is a tridiagonal matrix system that can be solved in $O(n)$ time. Once the c_i values are found, the d_i and the b_i values also can be found by Equations (10) and (15), respectively. Computing the b_i and d_i coefficients can be done also within $O(n)$ time.

III. A New Recurrence Equation-Based Solution

In our solution to the cubic spline interpolation problem, we chose a boundary condition that requires solving the following tridiagonal system where x_i are rational variables, e_i are rational constants and $r \neq 0$ is a rational constant, and A is:

$$A = \begin{bmatrix} r & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Furthermore,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{n-1} \\ e_n \end{bmatrix}.$$

A. Relationship to Clamped and Natural Cubic Splines

Our new matrix is closely related to clamped cubic splines. Consider the first equation for the clamped cubic spline, which can be written as:

$$2c_0 + c_1 = \frac{3}{h}\left(\frac{(a_1 - a_0)}{h} - f'(x_0)\right)$$

The above equation becomes the following after multiplying by $r/2$:

$$rc_0 + \frac{r}{2}c_1 = \frac{3r}{2h}\left(\frac{(a_1 - a_0)}{h} - f'(x_0)\right)$$

Adding $(1 - r/2)c_1$ yields:

$$rc_0 + c_1 = \frac{3r}{2h}\left(\frac{(a_1 - a_0)}{h} - f'(x_0)\right) + \left(1 - \frac{r}{2}\right)c_1$$

Hence the first row of our new matrix A is equivalent to first row of the clamped cubic spline for any $r \neq 0$ if e_1 is:

$$e_1 = \frac{3r}{2h}\left(\frac{(a_1 - a_0)}{h} - f'(x_0)\right) + \left(1 - \frac{r}{2}\right)\tilde{c_1}.$$

where $\tilde{c_1}$ is an estimate for the value of c_1.

The last row of the new matrix allows fixing the value of c_n. This is a generalization of natural cubic spline which fixes the value to be 0.

B. A Recurrence Equation-Based Solution

In this section, we solve the new system using the value $r = 2 + \sqrt{3} \approx 3.732$. This value of r has several interesting properties that can be summarized in the following theorems.

Theorem 1: Let $r = 2 + \sqrt{3}$. Then the following properties hold:

1)
$$\frac{4r - 1}{r} = r$$

2)
$$\frac{1}{r^k} = s_k - s_{k-1}r$$

where $s_0 = 1, s_1 = 4, s_n = 4s_{n-1} - s_{n-2}$

Proof: To show the first identity, simply substitute the value $r = 2 + \sqrt{3}$ to get:

$$\begin{aligned} \frac{4r-1}{r} &= \frac{4(2+\sqrt{3}) - 1}{2+\sqrt{3}} = \frac{4\sqrt{3}+7}{2+\sqrt{3}} = \\ &= \frac{(2+\sqrt{3})(2+\sqrt{3})}{2+\sqrt{3}} = 2 + \sqrt{3} = r \end{aligned}$$

From Condition (1) follows by multiplying by r that:

$$4r - 1 = r^2$$

which is equivalent to:

$$1 = 4r - r^2$$

Dividing the above by r gives:

$$\frac{1}{r} = 4 - r$$

which is exactly the base case of Condition (2) when $k = 1$. Suppose now that Condition (2) holds for $k - 1$ prove for k. Then we have that:

$$\begin{aligned} \frac{1}{r^k} &= \frac{1}{r}\frac{1}{r^{k-1}} = \frac{1}{r}(s_{k-1} - s_{k-2}r) = \frac{1}{r}s_{k-1} - s_{k-2} \\ &= (4 - r)s_{k-1} - s_{k-2} = (4s_{k-1} - s_{k-2}) - s_{k-1}r \\ &= s_k - s_{k-1}r \end{aligned}$$

Therefore, Condition (2) holds for all $k \geq 1$.

The first three equations can be written as:

$$\begin{aligned} rx_1 + \ x_2 \qquad\qquad &= e_1 \\ x_1 + 4x_2 + \ x_3 \qquad &= e_2 \\ x_2 + 4x_3 + x_4 &= e_3 \end{aligned}$$

Multiplying the second row by r, subtracting from it the first row, and then dividing it by r and using Condition (1) of Theorem 1 gives:

$$\begin{aligned} rx_1 + \ x_2 \qquad\qquad &= e_1 \\ rx_2 + \ x_3 \qquad &= e_2 - \frac{e_1}{r} \\ x_2 + 4x_3 + x_4 \qquad &= e_3 \end{aligned}$$

Multiplying now the third row by r, subtracting from it the second row, and then dividing it by r and again using Condition (1) of Theorem 1 gives:

$$\begin{aligned} rx_1 + \ x_2 \qquad\qquad &= e_1 \\ rx_2 + \ x_3 \qquad &= e_2 - \frac{e_1}{r} \\ rx_3 + x_4 \qquad &= e_3 - \frac{e_2}{r} + \frac{e_1}{r^2} \end{aligned}$$

Continuing this process until the last row, we get:

$$\begin{aligned} rx_{n-3} + x_{n-2} &= e_{n-3} - \tfrac{e_{n-4}}{r} + \tfrac{e_{n-5}}{r^2} - \ldots + (-1)^{n-4}\tfrac{e_1}{r^{n-4}} \\ rx_{n-2} + x_{n-1} &= e_{n-2} - \tfrac{e_{n-3}}{r} + \tfrac{e_{n-4}}{r^2} - \ldots + (-1)^{n-3}\tfrac{e_1}{r^{n-3}} \\ rx_{n-1} + \ x_n \ &= e_{n-1} - \tfrac{e_{n-2}}{r} + \tfrac{e_{n-3}}{r^2} - \ldots + (-1)^{n-2}\tfrac{e_1}{r^{n-2}} \\ x_n &= e_n \end{aligned}$$

Dividing each row except the last one by r yields:

$$\begin{aligned} x_{n-3} + \tfrac{x_{n-2}}{r} &= \tfrac{e_{n-3}}{r} - \tfrac{e_{n-4}}{r^2} + \ldots + (-1)^{n-4}\tfrac{e_1}{r^{n-3}} \\ x_{n-2} + \tfrac{x_{n-1}}{r} &= \tfrac{e_{n-2}}{r} - \tfrac{e_{n-3}}{r^2} + \tfrac{e_{n-4}}{r^3} - \ldots + (-1)^{n-3}\tfrac{e_1}{r^{n-2}} \\ x_{n-1} + \tfrac{x_n}{r} &= \tfrac{e_{n-1}}{r} - \tfrac{e_{n-2}}{r^2} + \tfrac{e_{n-3}}{r^3} - \ldots + (-1)^{n-2}\tfrac{e_1}{r^{n-1}} \\ x_n &= e_n \end{aligned}$$

Note that each row $1 \le i \le n-1$ will be the following:

$$x_i + \frac{x_{i-1}}{r} = \sum_{0 \le k \le (i-1)} (-1)^k \frac{e_{i-k}}{r^{k+1}}$$

We define the values for α_0, α_i for $1 < i \le n-1$, and α_n, respectively, as follows:

$$\begin{aligned} \alpha_0 &= 0 \\ \alpha_i &= \frac{e_i - \alpha_{i-1}}{r} = \sum_{0 \le k \le (i-1)} (-1)^k \frac{e_{i-k}}{r^{k+1}} \\ \alpha_n &= e_n \end{aligned} \tag{20}$$

The solution to the linear equation system can be described in terms of the α constants as follows:

$$\vdots$$

$$\begin{aligned} x_{n-3} &= \alpha_{n-3} - \frac{\alpha_{n-2}}{r} + \frac{\alpha_{n-1}}{r^2} - \frac{\alpha_n}{r^3} \\ x_{n-2} &= \alpha_{n-2} - \frac{\alpha_{n-1}}{r} + \frac{\alpha_n}{r^2} \\ x_{n-1} &= \alpha_{n-1} - \frac{\alpha_n}{r} \\ x_n &= \alpha_n \end{aligned}$$

Therefore, x_i for each row $1 \le i \le n$ will be:

$$\begin{aligned} x_n &= \alpha_n \\ x_i &= \alpha_{i-1} - \frac{x_{i+1}}{r} \end{aligned} \tag{21}$$

The above can be solved in closed form as follows:

$$x_i = \sum_{0 \le k \le (n-i)} \left(\frac{-1}{r}\right)^k \alpha_{i+k} \tag{22}$$

Note that no matter what exactly are the initial values for e, we have pre-solved the system. This can lead to a faster evaluation of the cubic spline than solving the tridiagonal system each time. We need only $O(n)$ multiplications and subtractions to compute the values of all the x_i. Moreover, when any new measurement is made, the conventional tridiagonal matrix-based algorithm requires a complete redo of the entire computation in $O(n)$ time. In contrast, Equation (21) leads to a faster update because to each x_i for $i \le n$ we need to add only the term:

$$\left(\frac{-1}{r}\right)^{n+1-i} \alpha_{n+1}.$$

We also need to make $x_{n+1} = \alpha_{n+1}$.. Afterward updating the other α_i constants can be done also similarly efficiently by adding a single term that contains e_{n+1}.

IV. A Moving Object Example

Suppose that an object is released from a height of 400 feet with zero initial velocity. Suppose also that we measure the object's position to be 384, 336 and 256 feet from earth at one, two and three seconds after release. We also suspect that the object is in free fall with a gravitational acceleration of $32 ft/sec^2$ at one second after release and at three seconds after release. Find a cubic spline approximation for the object's position at all times from the release to three seconds after.

We will measure the distance traveled from the release point. The cubic polynomials we need to find for the intervals $[0, 1], [1, 2]$ and $[2, 3]$ can be expressed as follows:

$$\begin{aligned} S_0(x) &= a_0 + b_0x + c_0x^2 + d_0x^3 \\ S_1(x) &= a_1 + b_1(x-1) + c_1(x-1)^2 + d_1(x-1)^3 \\ S_2(x) &= a_2 + b_2(x-2) + c_2(x-2)^2 + d_2(x-2)^3 \end{aligned}$$

We have $n = 4$, $a_0 = 400$, $a_1 = 384$, $a_2 = 336$, $a_3 = 256$ and the uniform step size is $h = 1$. By our assumptions of zero initial velocity $f'(0) = 0$ and free fall at one second $c_1 = -16$ and free fall at four seconds $c_3 = -16$, which implies $e_4 = -16$. The matrix A and the vectors x and B are:

$$A = \begin{bmatrix} r & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad x = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

and

$$B = \begin{bmatrix} -16r - 16 \\ -96 \\ -96 \\ -16 \end{bmatrix}$$

because

$$B = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} \frac{3r}{2}(-16) + \left(1 - \frac{r}{2}\right)(-16) \\ 3(400 - (2 \times 384) + 336) \\ 3(384 - (2 \times 336) + 256) \\ -16 \end{bmatrix}$$

By Equation (20), we have:

$$\alpha_1 = \frac{e_1}{r} = -16 - \frac{16}{r}$$

$$\alpha_2 = \frac{e_2 - \alpha_1}{r} = -16 - \frac{16}{r}$$

$$\alpha_3 = \frac{e_3 - \alpha_2}{r} = -16 - \frac{16}{r}$$

$$\alpha_4 = e_4 = -16$$

By Equation (21) we also have when calculating in reverse order:

$$c_3 = \alpha_4 = -16$$

$$c_2 = \alpha_3 - \frac{c_3}{r} = -16$$

$$c_1 = \alpha_2 - \frac{c_2}{r} = -16$$

$$c_0 = \alpha_1 - \frac{c_1}{r} = -16$$

Solving for the b_i coefficients by Equation (15) gives:

$$b_0 = \tfrac{1}{1}(384 - 400) - \tfrac{1}{3}(-16 - 32) = 0$$
$$b_1 = \tfrac{1}{1}(336 - 384) - \tfrac{1}{3}(-16 - 32) = -32$$
$$b_2 = \tfrac{1}{1}(256 - 336) - \tfrac{1}{3}(-16 - 32) = -64$$

Solving for the d_i coefficients by Equation (10) gives:

$$d_0 = \frac{1}{3}(-16 - (-16)) = 0$$

$$d_1 = \frac{1}{3}(-16 - (-16)) = 0$$

$$d_2 = \frac{1}{3}(-16 - (-16)) = 0$$

The above values show that an object in free fall has an increasing velocity but its acceleration remains constant. Using the above values, the cubic spine interpolation can be described as:

$$\begin{aligned} S_0(x) &= 400 - 16x^2 \\ S_1(x) &= 384 - 32(x-1) - 16(x-1)^2 = 400 - 16x^2 \\ S_2(x) &= 336 - 64(x-2) - 16(x-2)^2 = 400 - 16x^2 \end{aligned}$$

Hence in each piece the cubic spline interpolation gives $400 - 16x^2$, which agrees with the expected physics equation for the position of a moving object that starts with zero velocity from an elevation of 400 feet and freely falls downward with an acceleration of $32ft/sec^2$.

V. An Alternative Recurrence Equation

In this section we give an alternative recurrence equation that can be calculated directly using x_is instead of indirectly using the h_is. First note that Equation (18) can be rewritten as:

$$\begin{aligned} & h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_ic_{i+1} = \\ & \frac{3}{h_{i-1}}a_{i-1} - \frac{3(h_{i-1} + h_i)}{h_{i-1}h_i}a_i + \frac{3}{h_i}a_{i+1} \end{aligned} \qquad (23)$$

Note also that $h_i = x_{i+1} - x_i$ and $h_{i-1} = x_i - x_{i-1}$ implies that:

$$h_{i-1} + h_i = x_{i+1} - x_{i-1}$$

```
function[X] = CubicSpline(B)
N = length(B);
r = sqrt(3) + 2;
alpha = zeros(1, N);
X = zeros(1, N);

% Calculate the alpha values.
alpha(1) = 0;
alpha(N) = B(N);
for i = 1:(N-1)
    for k = 1:i
        alpha(i) = alpha(i) +
            (-1)^(k-1) * B(i-k+1)/(r^k);
    end
end

% Solve for X.
for i = 1:N
    for k = 1:(N-i+1)
        X(i) = X(i) +
           (-1/r)^(k-1) * alpha(i+k-1);
    end
end
```

Fig. 1: The MATLAB implementation.

Substituting the above into Equation (23) yields:

$$\begin{aligned}(x_i - x_{i-1})c_{i-1} + 2(x_{i+1} - x_{i-1})c_i + (x_{i+1} - x_i)c_{i+1} = \\ \frac{3}{(x_i - x_{i-1})}a_{i-1} - \frac{3(x_{i+1} - x_{i-1})}{(x_i - x_{i-1})(x_{i+1} - x_i)}a_i + \\ \frac{3}{(x_{i+1} - x_i)}a_{i+1}\end{aligned}$$

Unlike Equation (19), the above does not assume a uniform spacing of the x_i values.

VI. Implementations of the Cubic Spline Algorithm

We implemented the cubic spline interpolation algorithm in both MATLAB and the MLPQ database system in order to provide ready-to-use versions to potential users. These implementations are described separately below in subsections A and B, respectively.

A. The MATLAB Implementation

The MATLAB is a high-level programming language and a interactive environment for numerical computation. MATLAB is a popular system that by 2004 already had over a million users. The user is expected to provide only the constant array B as input to the function *CubicSpline*, which was implemented in MATLAB as shown in Figure 1. The implementation is simple and runs fast in MATLAB.

B. The MLPQ Implementation

The MLPQ (Management of Linear Programming Queries) database system is built at the University of Nebraska-Lincoln under the author's direction [10]. The MLPQ database system allows querying of relational databases, geographic databases and constraint databases [7]. In the current version of MLPQ only linear constraints over the rational numbers are implemented, but there is a plan to extend the implementation to polynomial constraints over the real numbers.

The goal of the MLPQ implementation is to allow users to conveniently retrieve the interpolated value rather than to find the coefficients of Equation (1). Suppose that we already found the coefficients a_i, b_i, c_i, d_i of Equation (1). Note that Equation (1) can be written as:

$$\begin{aligned}S_i(x) &= a_i + b_i(x - x_i) + c_i(x^2 - 2xx_i + x_i^2) + \\ &\quad d_i(x^3 - 3x^2x_i + 3xx_i^2 - x_i^3)\end{aligned}$$

The above can be rewritten as follows:

$$\begin{aligned}S_i(x) &= (a_i - b_ix_i + c_ix_i^2 - d_ix_i^3) + \\ &\quad (b_i + 2c_ix_i + 3d_ix_i^2)x + \\ &\quad (c_i + 3d_ix_i)x^2 + \\ &\quad d_ix^3\end{aligned}$$

Define the constant coefficients $\hat{a}_i = a_i - b_ix_i + c_ix_i^2 - d_ix_i^3$, $\hat{b}_i = b_i + 2c_ix_i + 3d_ix_i^2$, $\hat{c}_i = c_i + 3d_ix_i$ and $\hat{d}_i = d_i$. Then S_i is simply a polynomial of x with these coefficients.

$$S_i(x) = \hat{a}_i + \hat{b}_ix + \hat{c}_ix^2 + \hat{d}_ix^3$$

In the MLPQ constraint representation, the interpolated cubic spline value at location x is a cubic polynomial constraint, which can be viewed as a linear constraint of x, x^2 and x^3. Therefore, we represent $S_i(x)$ over the interval $[x_i, x_{i+1}]$ by the linear constraint tuple:

S	X	Xto2	Xto3	
s	x	y	z	$x_i \le x,\ x \le x_{i+1},$ $s = \hat{a}_i + \hat{b}_ix + \hat{c}_iy + \hat{d}_iz$

To specify the entire cubic spline interpolation, we need to create a constraint relation that contains a separate constraint tuple for each interval. For example, the moving object cubic spline interpolation of Section IV can be represented by the following constraint relation with three constraint tuples:

Spline

S	X	Xto2	Xto3	
s	x	y	z	$0 \le x,\ x \le 1,\ s = 400 - 16y$
s	x	y	z	$1 \le x,\ x \le 2,\ s = 400 - 16y$
s	x	y	z	$2 \le x,\ x \le 3,\ s = 400 - 16y$

From the entire relation, MLPQ users can retrieve the interpolated value at any desired x by specifying the values of x, x^2 and x^3. For example, to retrieve the cubic spline interpolation value at location $x = 2$, the query would be the following:

```
SELECT   S
FROM     Spline
WHERE    X = 2 AND Xto2 = 4 AND Xto3 = 8
```

For the above query, the MLPQ system would return the answer of $S = 336$. The need to give also the values of x^2 and x^3 is only a present inconvenience to the users. This inconvenience can be eliminated in the future when polynomial constraints over the real numbers will be fully implemented within the MLPQ database system.

The ability to represent a cubic spline interpolation within a database system enables many applications because database queries are more flexible than just calculating single interpolation values. For example, suppose that in another application the *Internet* relation describes the monthly volume of Internet traffic through a network node. Then a user can find the increase in the Internet traffic from the first month to the fifth month by the following query:

```
SELECT   Traffic_Increase
FROM     Internet AS I1, Internet AS I2
WHERE    I1.X = 1 AND I2.X = 5 AND
         I1.Xto2 = 1 AND I2.Xto2 = 25 AND
         I1.Xto3 = 1 AND I2.Xto3 = 125 AND
         Traffic_Increase = I1.S - I2.S
```

VII. CONCLUSION

The general method described in this paper can be used in a wide variety of applications which require interpolation of a function of one variable, for example, in data mining, data classification and efficient data encryption and transmission [2], [9], [11], [12], [13], [8], [14]. Interpolation of measurement data can generate constraint databases that can be efficiently queried using constraint query languages [6], [7].

The simple one-variable function interpolation can be also extended to higher dimensions yielding interpolations of higher-dimensional functions that describe surfaces [5] and three-dimensional spatio-temporal or moving objects [1], [4]. This extension remains an interesting future work.

REFERENCES

[1] A. Anderson, P. Z. Revesz, Efficient MaxCount and threshold operators of moving objects, *Geoinformatica*, 13 (4), 2009, pp. 355–396.

[2] Z. Brahimi, H. Bessalah, A. Tarabet, M. K. Kholladi, Selective encryption techniques of jpeg2000 code stream for medical images transmission, *WSEAS Transactions on Circuits and Systems*, 7, 2008.

[3] R. L. Burden, J. D. Faires, *Numerical Analysis*, 9 Edn., Springer, New York, USA, 2014.

[4] J. Chomicki, P. Z. Revesz, Constraint-based interoperability of spatiotemporal databases, *Geoinformatica*, 3 (3), 1999, pp. 211–243.

[5] L. Li, and P. Z. Revesz, Interpolation methods for spatio-temporal geographic data, *Computers, Environment and Urban Systems*, 28 (3), 2004, pp. 201–227.

[6] P. C. Kanellakis, G. M. Kuper, P. Z. Revesz, Constraint query languages, *Journal of Computer and System Sciences*, 51 (1), 1995, pp. 26–52.

[7] P. Z. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*, Springer, New York, USA, 2010.

[8] P. Z. Revesz, A method for predicting the citations to the scientific publications of individual researchers, *18th International Database Engineering and Applications Symposium*, ACM Press, 2014, pp. 9–18.

[9] P. Z. Revesz, C. Assi, Data mining the functional characterizations of proteins to predict their cancer-relatedness, *International Journal of Biology and Biomedical Engineering*, 7 (1), 2013, pp. 7–14.

[10] P. Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, Y. Wang, The MLPQ/GIS constraint database system, *ACM SIGMOD International Conference on Management of Data*, ACM Press, 2000.

[11] P. Z. Revesz, T. Triplet, Classification integration and reclassification using constraint databases, *Artificial Intelligence in Medicine*, 49 (2), 2010, pp. 79–91.

[12] P. Z. Revesz, T. Triplet, Temporal data classification using linear classifiers, *Information Systems*, 36 (1), 2011, pp. 30–41.

[13] P. Z. Revesz, R. Woodward, Variable bounds analysis of a climate model using software verification techniques, *in* J. Balicki et. al (Eds), *Applications of Information Systems in Engineering and Bioscience*, WSEAS Press, 2014, pp. 31–36.

[14] V. Skala, Fast interpolation and approximation of scattered multidimensional and dynamic data using radial basis functions, *WSEAS Transactions on Mathematics*, 12, 2013.

Peter Z. Revesz holds a Ph.D. degree in Computer Science from Brown University. He was a postdoctoral fellow at the University of Toronto before joining the University of Nebraska-Lincoln, where he is a professor in the Department of Computer Science and Engineering. Dr. Revesz is an expert in databases, data mining, big data analytics and bioinformatics. He is the author of Introduction to Databases: From Biological to Spatio-Temporal (Springer, 2010) and Introduction to Constraint Databases (Springer, 2002). Dr. Revesz held visiting appointments at the IBM T. J. Watson Research Center, INRIA, the Max Planck Institute for Computer Science, the University of Athens, the University of Hasselt, the U.S. Air Force Office of Scientific Research and the U.S. Department of State. He is a recipient of an AAAS Science and Technology Policy Fellowship, a J. William Fulbright Scholarship, an Alexander von Humboldt Research Fellowship, a Jefferson Science Fellowship, a National Science Foundation CAREER award, and a Faculty International Scholar of the Year award by Phi Beta Delta, the Honor Society for International Scholars.