Fixed Point Performance of Interpolation/Extrapolation Algorithms for Resource Constrained Wireless Sensors

Babak D. Beheshti
School of Engineering & Computing Sciences
New York Institute of Technology
Old Westbury, NY, USA

Abstract—**Date collection in wireless networks is performed by sampling the phenomenon to be measured at recurring intervals in time. In situations where the sensors must operate over a very long time without the possibility of battery replacement, the sleep cycle for these sensors must be as long as possible (and the longest the design allows). Longer sleep cycles, while extending battery life may be too long for the sampling rate required to collect data – i.e. more frequent wake ups would be needed from the sensor.**

On the other hand, the sensors can collect samples at a lower than required rate, but interpolate data-points in between the actual measured sampled data. Interpolation implemented in floating point can be demanding in terms of energy cost to the sensors. Therefore as it is generally the case, fixed point implementations have to be introduced to reduce the energy demands due to excessive computations/instruction count due to floating point arithmetic.

In this paper we present a fixed point implementation of the Neville's interpolation scheme, and present performance results as compared to the floating point (ideal) results.

Keywords- fixed point; interpolation; WSN; DSP.

I. Introduction to Interpolation

There is sufficient coverage of interpolation schemes in the literature. [1] Most practical techniques begin from a given point lose to the data points, say y_i, then then subsequently update the initial value with corrections that are decreasing in magnitude. If these corrections are not decreasing the solution will not converge. In addition the correction values information from other points is also incorporated in the calculation. The computational complexity of this technique is in the order of O(N2) operations. Additionally, to avoid divergent solutions, it is required that an interpolation algorithm also provide an estimate of its error.

We call the number of points less one, that are used in an interpolation algorithm the order of the interpolation. As will be seen in this paper, increasing the order does not necessarily increase the accuracy. This is particularly true in polynomial interpolation cases. The range for the order, as demonstrated in this paper is 4 to 6.

A recursive and practical interpolation algorithm is Neville's algorithm. In the following we briefly describe this algorithm as described in [2].

Let P_1 be the value at x of the unique polynomial of degree zero passing through the point (x_1, y_1); so $P_1 = y_1$. Now define P_2, P_3. . . P_N. Next we define intermediate points. Let P_{12} be the value at x of the unique polynomial of degree one passing through both (x_1, y_1) and (x_2, y_2). Similarly P_{23}, P_{34}. . . $P_{(N-1)N}$ are defined. We continue this trend of definitions for higher-order polynomials, up to $P_{123...N}$, which is the value of the unique interpolating polynomial through all N points. This final value constitutes the output of the interpolation function over the given N points.

Fig.1 illustrates the relation between these intermediate points and the final interpolated output.

x_1	$y_1 = p_1$			
x_2	$y_2 = p_2$	P_{12}	P_{123}	
x_3	$y_3 = p_3$	P_{23}	P_{234}	P_{1234}
x_4	$y_4 = p_4$	P_{34}		

Fig. 1 – Interpolation in Neville's Algorithm

At each level m, the C's and D's are the corrections that make the interpolation one order higher. The final answer $P_{1...N}$ is equal to the sum of any yi plus a set of C's and/or D's that form a path through the family tree to the rightmost child.

One can think of Neville's algorithm as a recursive technique to fill in the numbers in Fig. 1 one column at a time, from left to right. The following are the governing equations in this algorithm.

$$P_{i(i+1)\dots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\dots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\dots(i+m)}}{x_i - x_{i+m}}$$

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i}}{x_i - x_{i+m+1}}$$

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i}}{x_i - x_{i+m+1}}$$

II. Fixed Point Arithmetic

In general, the incentive to use fixed point arithmetic over floating point is that they are more efficient in terms of instruction count, once they are translated from a high level language to a processors assembly language. Also fixed point arithmetic comes for free on computers, as it requires no special purpose hardware – the existing integer arithmetic functionality can also be used to implement fixed point arithmetic. The smaller instruction count of fixed point arithmetic inherently implies a faster execution speed. Therefore in embedded applications, communications systems and many real time systems where the high power consumption of dedicated floating point processors is not practical, fixed point arithmetic is used instead.

The disadvantage of fixed point arithmetic on the other hand is the loss of range and precision as compared to floating point arithmetic. As an example, in a fixed < 8, 1 > representation (where the total number of bits in the word is nine, 8 bits constitute the whole part, and one bit constitutes the fractional part), the fractional part is only precise to a quantum of 0.5. That is, one cannot represent a number such as 0.75 in this format. We can only represent 0.75 with fixed<8,2> or higher number of bits in the fractional part. The tradeoff however is that by giving bits to the fractional part to increase precision, we take bits away from the whole part, thereby reducing range of the numbers that can be represented in this format. Reducing the range results in an increase in probability of overflow. In fixed point arithmetic, the sum of two N-bit numbers will need N+1 bits. The product of two N-bit numbers requires a 2N-bit word size.

The term "Fixed Point" comes from the fact that every word has the same number of digits and the virtual point separating the fractional and whole parts is always fixed at the same position.

b_{n+m-1}	…	b_{n+1}	b_n	b_{n-1} …	b_1 b_0
m-bits of whole part				n-bits of fractional part	

The following rules to implement aarithmetic operations in fixed point have to be observed:

Addition: two numbers must be represented in the same <m,n> format to be added, it means that the binary point must be aligned. Adding two N-bit numbers can require a N+1 bit number, therefore scaling (truncating after shifting right, or rounding after shifting) must be incorporated.

Multiplication:

<m1,n1> * <m2,n2> = <(m1+m2+1),(n1+n2)>. This implied that multiplying two N-bit numbers requires a 2N bit storage. Typically, to maintain the word size at N-bits, the product must be "scaled" (truncating after shifting right, or rounding after shifting)

The source code for fixed point multiplication is as follows:

```
void fixed_mul(fx_word_t *a, fx_word_t *b, fx_word_t *c)
{
        *c = (*a) * (*b);
        *c = round_fx_word_t(*c, SCALE);
   }
```

The rounding is done by the code below:

```
fx_word_t round_fx_word_t(fx_word_t a, int scale)
{
        fx_word_t half_lsb_fixed, retval;
        //printf("scale = %d\n", scale);
        half_lsb_fixed = (fx_word_t)(1 << (scale - 1));

        retval = (a + half_lsb_fixed);   //this is
actually the float equivalent!
        retval = retval >> scale;        //shift right to
get rid of extra fraction part
        //printf("retval = %f\n", retval);
        return (retval);
   }
```

Where "fx_word_t" can be defined as 16, 32 or 64 bit word size. Also "scale" is the number of bits in the fractional part.

Resolution: is the smallest non-zero magnitude representable (2^{-n})

Accuracy: is the magnitude of the maximum difference between a real value and its representation.

III. Our Fixed Point Implementation of Interpolation

We began with a floating point version of the Neville's algorithm. Once the performance of the floating point implementation was verified against the ideal test cases, the algorithm was ported to the fixed point as follows:

- All float data types were converted to integer type.
- All arithmetic operations were converted to their fixed point versions as discussed above. In case of multiplication we used rounding to t0 ½ LSB (Least Significant Bit), as opposed to truncation, as the former provides better numerical accuracy.
- The fixed point results were converted to floating point for the purposes of analysis and illustration.

The word size was selected to be 32 bits, however 16-bit and 64 bit word sizes were also exercised. The results were identical and therefore not presented in this paper. It is noteworthy that if we normalized the input data to the interpolation algorithm, then word size will become of importance, as overflow considerations come into the picture. We address the issue of overflow later in this paper.

IV. Performance Analysis

To test the performance of the fixed point implementation, a polynomial of order 4 was selected to be the curve to test the interpolation data against. N data points were selected from the curve to be supplied as the input ordered-pairs of (xi, yi) to the interpolation routine. The polynomial is of the form:

$$y_i = a_0 \times x^0 + a_1 \times x^1 + a_2 \times x^2 + a_3 \times x^3 + a_4 \times x^4$$

The selected polynomial for the range of data points under test has a shape is illustrated in Fig. 2.

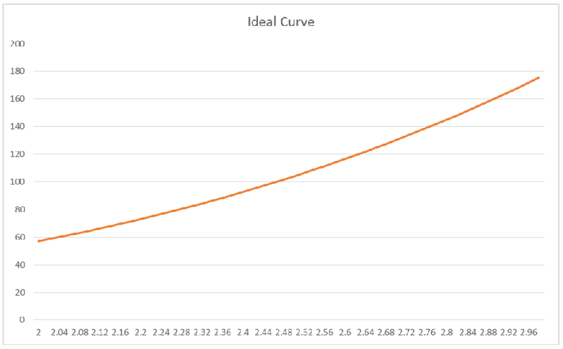

Fig. 2 – The Polynomial Used for the Test Cases

The order N was varied from 3 to 20. The results of the % max error for each experiment is shown in Table 1. As can be seen, increasing the order past 4 does not have any impact on the performance of the fixed point interpolation

TABLE I. MAX ERROR VS. ORDER

Order	Max Error
N = 5	82.57
N = 6	82.57
N = 7	82.57
N = 8	82.57
N = 20	82.57

In order to have a sense of performance of the algorithm over a range of data in the polynomial, we submitted an x in the range of 2.0 to 3.0 to the interpolation algorithm and plotted the % error between the actual polynomial value at each point and the interpolated value. The graphs are shown for several orders of the interpolation in Fig. 3. It is important to note that the family of graphs shown in Fig. 3 are identical. We have introduced an offset in the chart to ensure that the identical graphs do not overlap and hide one another.

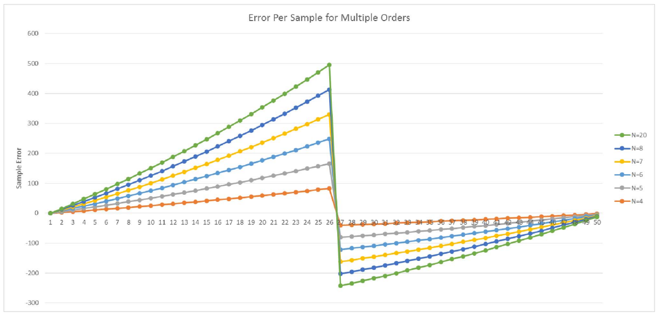

Fig. 3– Error per Sample for a Range of Data from the Polynomial between the Interpolated and Ideal Values

Next, we experimented with what effectively constitutes the quantization noise in our fixed point implementation. The fractional part of the fixed sized word contributes to the accuracy of the arithmetic operations. We varied the size of the fractional part from 4 bits to 12 bits and recorded the maximum error between the ideal results and the output of the interpolation algorithm. The results are shown in Fig. 4.

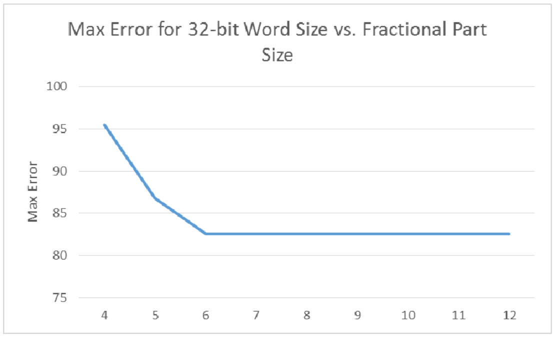

Fig. 4– Maximum Error vs. Fractional Part Size

As can be seen, the quantization noise for the 32-bit implementation of this test data reduces as we increase the fractional part. However, we observe no improvement beyond the fractional part equal to 6. This can be interpreted as follows: beyond fractional part equal 6, the interpolation errors overtake quantization error, and therefore the quantization error becomes negligible as compared to other error sources.

CONCLUSION

In this paper we have presented a fixed point implementation of the Neville's interpolation algorithm to be used in wireless sensor nodes, as well any other resource constrained processors. We have shown that the order of the interpolator does not provide any improved performance beyond N=5. We have also demonstrated that with a 32-bit word size, the interpolator can be used with reasonable accuracy.

REFERENCES

[1] R.E. Crochiere and L.R. Rabiner. (1983). Multirate Digital Signal Processing. Englewood Cliffs, NJ: Prentice–Hall.

[2] Numerical Recipes in C, The Art of Scientific Computing, Second Edition, William H. Press, Cambridge University Press 1988, 1992.

[3] Chapter 2. Acton, F.S. 1970, Numerical Methods That Work; 1990, corrected edition (Washington: Mathematical Association of America), Chapter 3.

[4] Kahaner, D., Moler, C., and Nash, S. 1989, Numerical Methods and Software (Englewood Cliffs, NJ: Prentice Hall), Chapter 4.

[5] Johnson, L.W., and Riess, R.D. 1982, Numerical Analysis, 2nd ed. (Reading, MA: Addison-Wesley), Chapter 5.

[6] Ralston, A., and Rabinowitz, P. 1978, A First Course in Numerical Analysis, 2nd ed. (New York: McGraw-Hill), Chapter 3.

[7] Wikipedia: http://en.wikipedia.org/wiki/Fixed-point_arithmetic

[8] http://www.digitalsignallabs.com/fp.pdf

[9] Wonyong Sung , Ki-Il Kum, Simulation-based word-length optimization method for fixed-point digital signal processing systems, IEEE Transactions on Signal Processing, v.43 n.12, p.3087-3090, December 1995 [doi>10.1109/78.476465]

[10] Dahlquist, G., Björck, Å. (1974) Numerical Methods. Prentice-Hall, Englewood Cliffs, N.J.

[11] Davis, P. J. (1975) Interpolation and Approximation. Dover, New York

[12] A. V. Oppenheim and C. J. Weinstein Proc. IEEE, vol. 60, pp.957 1972

[13] T. Thong and B. Liu IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-24, pp.563 1976

[14] J. H. Wilkinson Rounding Errors in Algebraic Processes, 1963 :Prentice-Hall

[15] M. Abramowitz and I. A. Stegun Handbook of Mathematical Functions, 1964