Sensor Data Processing Using Genetic Algorithms

James W. Hauser
University of Cincinnati, ECECS
Cincinnati, Ohio 45221
hauserjw@email.uc.edu

Carla N. Purdy
University of Cincinnati, ECECS
Cincinnati, Ohio 445221
carla.purdy@uc.edu

***Abstract*-An off-the-shelf package is commonly used to generate an approximating polynomial from partial sensor data. The floating-point coefficients are rounded to the target architecture's size. Rounding errors can actually be due to this solution space translation. A genetic algorithm can be used to find the optimal coefficient set in the restricted target space.**

I. INTRODUCTION

There are two uses for approximating functions. The first is to replace complicated functions by simpler functions. The second is for recovering a function from partial information [1]. These two purposes are not mutually exclusive. The commonly used classes of approximating functions are algebraic polynomials, trigonometric polynomials, and piecewise polynomial functions.

Engineers usually use an off-the-shelf package such as MATLAB [2] to generate approximating polynomials. The common fallacy is to attempt to approximate a function by using a single polynomial of high degree. But, a more efficient set of approximating functions can be found, in a piecewise fashion, by selecting breakpoints such that the error within each subinterval is less than some predetermined constraint and by limiting the degree of the approximating polynomials.

If the target for executing the approximation is an infinite precision floating-point processor, then the previously described method is more than adequate. But, if the target processor is a fixed-point processor or an ASIC, the floating-point coefficients will be rounded to the number of bits determined by the architecture. Errors are attributed to rounding.

The error is not really due to rounding but to the translation from one solution space to another. The floating-point coefficient set and the optimum coefficient set in the restricted solution space are often unrelated. A genetic algorithm can be used to search the restricted solution space to find the optimal coefficient set.

Other methods exist for performing function approximation. These methods include, for example, neural nets and cubic splines. The drawback of a neural net is the evaluation of the exponential function as well as the fact that the weights are again real [3]. Splines are also composed of real coefficients and have the additional drawback of requiring a "large" number of multiplications per output computation [4]. We are interested in absolute accuracy and efficiency of implementation in an ASIC design [5] rather than smoothness.

Research supported by the State of Ohio – DAGSI Program 2000.

II. A COMMON ENGINEERING PROBLEM

In our sample system (Fig. 1) the sensor being used is a thermistor and the analog preconditioning circuit is a voltage divider network. The A/D converter has 16 bits and the processor computing the actual temperature is an FPGA. The required accuracy is .01°C. Therefore, simplicity of implementation and memory storage are important design considerations.

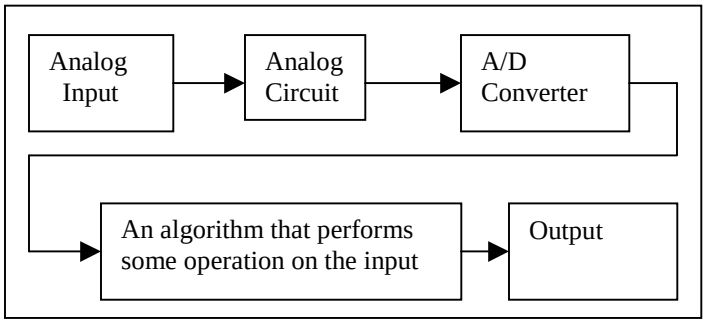

Fig. 1. Stages in sensor data processing.

A thermistor is a thermally sensitive resistor whose resistance changes about three orders of magnitude in a 100°C range. A model for the resistance-temperature relationship of a thermistor is the Steinhart and Hart equation,

$$1/\ (\ a + b\ (\ \mathrm{Ln}\ R\ ) + c\ (\ \mathrm{Ln}\ R)^3\ )\ =\ T. \qquad (1)$$

T is temperature in Kelvin units and the coefficients a, b and c are experimentally determined [6].

But the sample system above does not measure resistance. The input to the A/D is a "processed" voltage and additional non-linearity is added by the circuitry. In addition, the natural log function would be expensive to implement in an ASIC or fixed-point processor design.

III. SAMPLE SYSTEM DATA

Since the thermistor alone does not determine the input read at the A/D converter, the system must be calibrated by setting the temperature and recording corresponding A/D counts. The inherent non-linearity of the system is evident in Fig. 2.

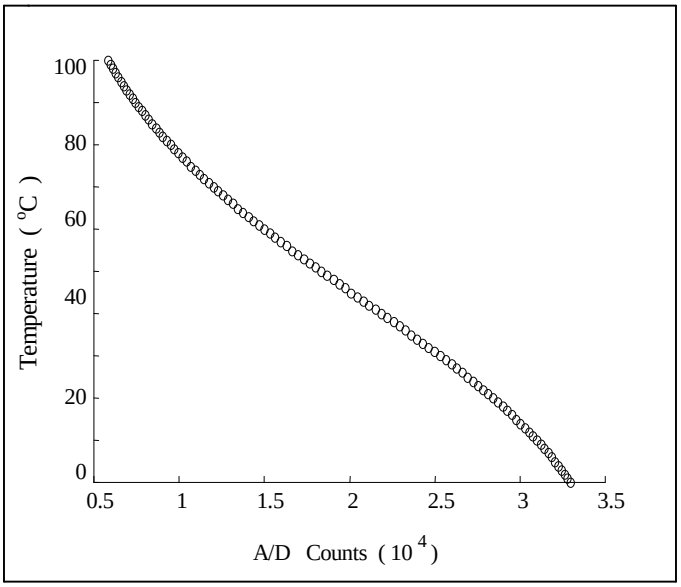

Fig. 2. Temperature as a function of A/D input.

IV. DESIGN OPTIONS

The simplest design would incorporate a lookup table with linear interpolation. The amount of input processing is minimized, but to obtain the required accuracy, more sample points are required when the curve is more nonlinear. A large amount of storage would also be required.

Another option would be to build up the interpolating polynomial in a piecewise fashion using a polynomial of small degree and a least squares fit. This could be accomplished using the MATLAB *polyfit* function. Table I is the result of this process using the sample data.

TABLE I
PIECEWISE INTERPOLATING POLYNOMIAL COEFFICIENTS USING MATLAB

	Breakpoint		a	b	c	d
1	5819	8200	-1888.2	642.40	-229.35	99.9
2	8200	9718	1838.5	109.48	-160.18	86.0
3	9718	13743	-365.97	197.22	-140.83	78.9
4	13743	21508	-119.70	74.211	-107.93	64.0
5	21508	27834	-157.97	1.4033	-92.844	41.0
6	27834	28698	-15555	459.49	-115.07	22.0
7	28698	32043	-803.20	-114.41	-117.08	19.0
8	32043	32948	-1876.9	-445.89	-167.28	4.99

The corresponding template for the interpolating polynomial is:

$$a\,x^3 + b\,x^2 + c\,x + d. \qquad (2)$$

"x" is the normalized A/D input (Q15) minus the lower breakpoint bracket value (Q15). Q15 describes a fixed-point integer representation of a floating-point number that has a sign and 15 decimal places.

Generalization testing over the entire A/D range yields acceptable results. That is, the absolute error is always less than .01°C. But the target processor does not have a floating-point unit and, as a result, the coefficients must be rounded to integers (16 bits).

Of course there are ways to maximize the precision. For example, divide each polynomial by its smallest coefficient and then shift left until one of the coefficients reaches the maximum size chosen, such as 16 bits. This works well as long as the range from maximum to minimum coefficient is not large. Storage needed is also increased. Fig. 3 shows the potential error resulting from rounding for breakpoint bracket 5.

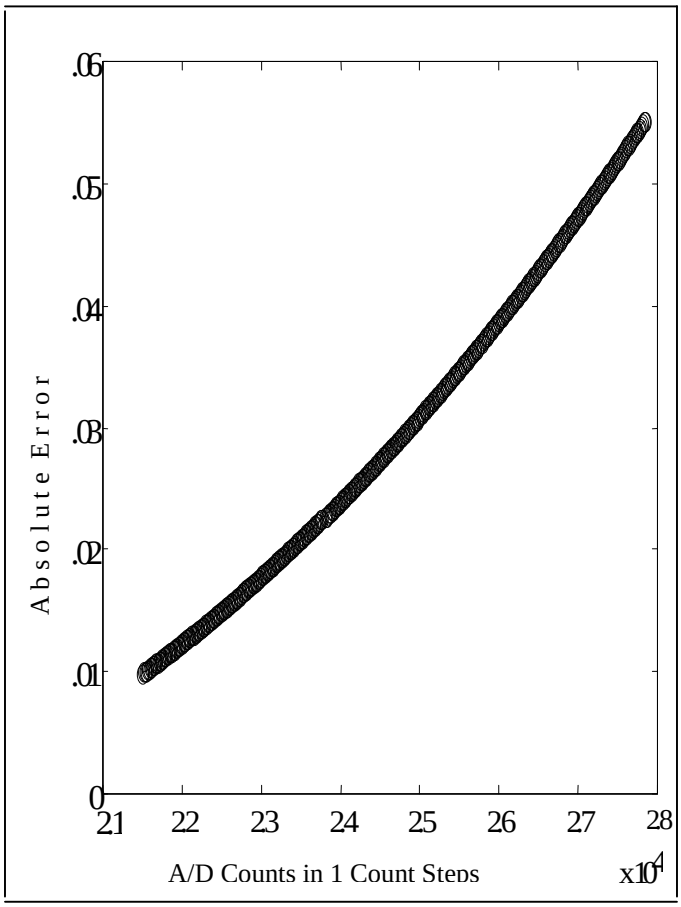

Fig. 3. Induced floating-point error caused by rounding.

V. AN ALTERNATE APPROACH

We used an infinite precision algorithm to find the piecewise interpolating polynomials and then tried creative tricks to minimize the damage caused by truncation. But, what if we could find the best set of integer coefficients from the start? The coefficients a, b, c and d would take values only from the integer set {-32768…32767}. Calculus based algorithms exist for finding the coefficients when they are real. Examples are least squares and gradient descent.

There are two ways to find the optimum integer coefficients:

- Enumeration: Try every possibility. Given the bracket, it takes 5 days on a 450 MHz Pentium to find the optimum integer coefficients.
- Heuristic methods such as a genetic algorithm.

VI. WHAT IS A GENETIC ALGORITHM?

First an initial population is randomly chosen. Two individuals are selected from the population for mating with higher probability given to more fit individuals (where fitness is problem dependent). A crossover point is chosen at random and the two pieces are swapped. During crossover a mutation infrequently occurs [7].

Table II contains the results of using this genetic algorithm (GA) paradigm on our sample data set. It is interesting to note that although the GA found the same number of breakpoint brackets, they are not the same as those found using MATLAB.

TABLE II
PIECEWISE INTERPOLATING POLYNOMIAL
COEFFICIENTS USING A GENETIC ALGORITHM

	Breakpoint		a	b	c	d
1	5819	8200	-2065	663	-230	100
2	8200	9041	9217	-151	-158	86
3	9041	10684	-2314	362	-152	82
4	10684	17302	-187	137	-129	75
5	17302	24641	-102	33	-96	53
6	24641	28420	-77	-61	-96	32
7	28420	32043	-855	-83	-116	20
8	32043	32948	-966	-482	-167	5

Fig. 4 compares the generalization test results of MATLAB infinite precision floating point versus the GA. To span MATLAB bracket 5, GA brackets 5 and 6 are plotted in Fig. 4.

Recall that the absolute error requirement was .01° C. The original floating point polynomial set met this requirement. When the coefficients were rounded the maximum error over bracket 5, whose breakpoint bracket range is 21508 to 27834, became .057 ° C (Fig. 3). Fig. 4 shows that the integer valued GA generalization results over the same range give errors less than .008°C.

The amount of processing time required for the GA to find the 32 optimum integer coefficients and breakpoint brackets was 25 minutes versus about 500 days using enumeration. Currently algorithm refinements are being implemented to further reduce this processing time. Fig. 5 contains the result of performing generalization testing over the defined input range. In Fig. 5 there is one point where the GA differs from the floating-point results by more than .01°C. In fact, the GA result is closer to the Steinhart and Hart (1) theoretical value then the MATLAB *polyfit* value.

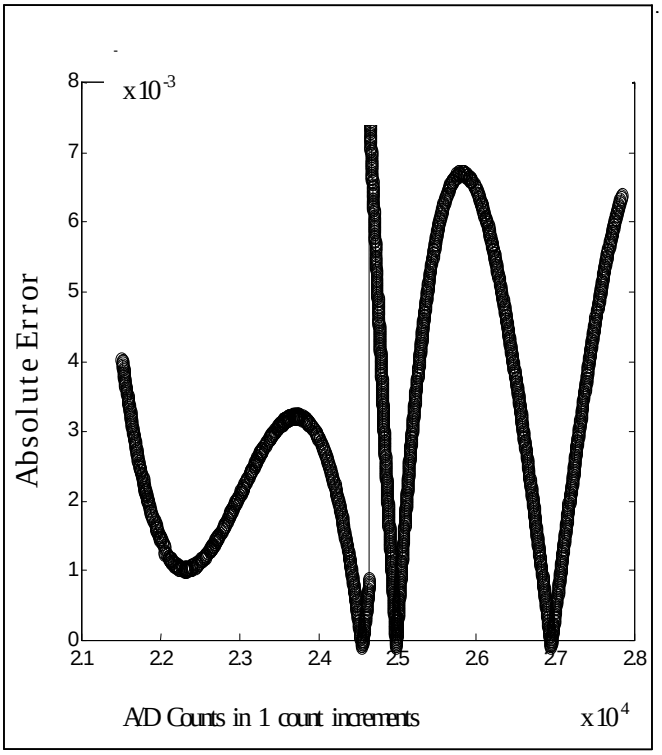

Fig. 4. MATLAB *polyfit* versus genetic algorithm.

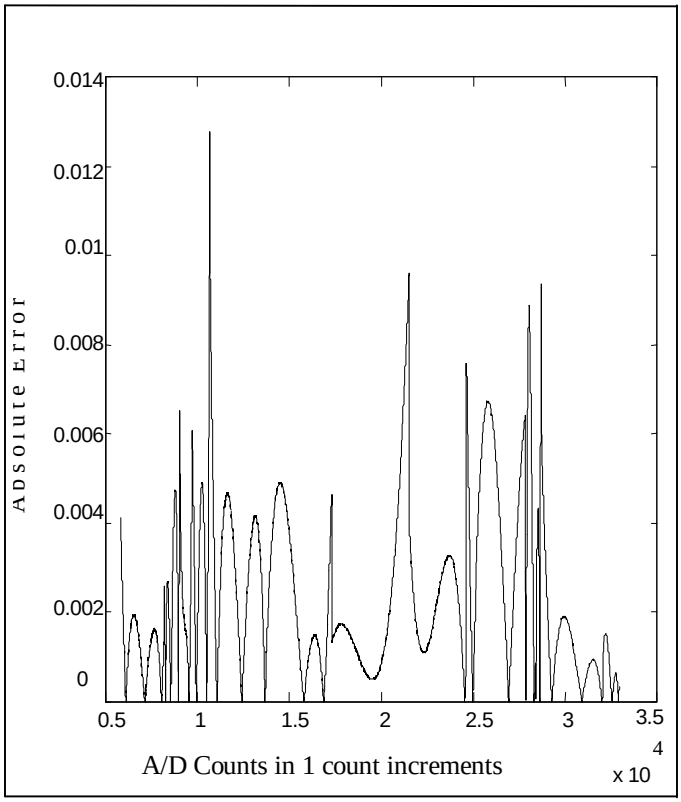

Fig. 5. GA generalization testing over the input range.

VII. SUCCESSFULLY IMPLEMENTING A GA

The genetic algorithm paradigm described above is just a crude outline of the hidden complexity of the actual implementation. Successful implementation is dependent on [8,9,10]:

- How the initial population is chosen.
- Numerical representation of the individual.
- Determination of the fitness function.
- Heuristics used to avoid premature convergence due to a "super" individual.
- The specific implementation of crossover.
- The specific implementation of mutation.
- Tuning and selection of the probabilities associated with crossover and mutation.
- Complexity analysis using statistical methods.

One of the most critical aspects of implementing an efficient GA is choice of the initial population. By virtue of (2) where the lower breakpoint value is subtracted from the input value, the "d" coefficient is already fixed. It is set to the value of the output corresponding to the input's lower breakpoint value. Therefore, the search space is reduced from 2^{64} to 2^{48}, which is equal to 65536^3, which corresponds to the number of combinatorial possibilities for three coefficients of sixteen bits each.

What is needed is a small yet representative set of candidate 3rd degree polynomials to be used as input to the GA. One possibility is to generate the initial population randomly using a Bernoulli random number generator with the P[1] = .5 and P[0] = .5. But, the number of polynomials generated will be small and there is no guarantee of generating a good uniform distribution.

To obtain a uniform sampling of the solution space, a three coefficient by sixteen-bit solution space can be visualized as a cube, 65536 units on a side. Within this large cube there are 512 cubes, 8192 units on a side. Our initial population was chosen by randomly permuting in three dimensions a maximum of 4096 units from the center of each of these sub-cubes. The resulting set of polynomials was then randomly shuffled.

Heuristic algorithms like the GA are characterized differently than their deterministic counterparts. The population represents 1024 possible solutions. Although the heuristics used do not necessarily always guide the search in the correct direction, they quite often do. In our implementation, when the GA converges to a clearly sub-optimal solution, the search is restarted with a new population. The probability that the new population chosen will lead to another sub-optimal solution is small but not zero [11].

The problem of finding a set of piecewise continuous interpolating third degree polynomials with integer coefficients is akin to finding multiple needles in multiple haystacks. That is, there are no clearly defined valleys or ridges in the solution space to follow to the optimum solution.

In general, a heuristic algorithm should be used for a great variety of important problems. If optimal is unattainable, then sacrifice optimality and settle for a feasible solution that can be computed efficiently.

VIII. CONCLUSION

It is clear that caution must be exercised when results derived from an infinite precision algorithm are transformed to a fixed-point implementation.

Through the example presented here, it has been shown that a genetic algorithm can be used to search the restricted integer solution space to find the optimal integer polynomial coefficients directly. With this method, the system designer can achieve near floating accuracy when converting an analog sensor input to its digital representation, while using the available integer architecture efficiently.

ACKNOWLEDGMENT

The authors wish to thank Brian Code and Jack Shidemantle from YSI Inc., Beavercreek, Ohio, and Professors Rob Ewing, Gary Lamont, and Mark Oxley from the Air Force Institute of Technology, Wright-Patterson AFB, Ohio, for their encouragement and technical support.

REFERENCES

[1] Conte and de Boor, *Elementary Numerical Analysis*, McGraw-Hill, 1980.

[2] The MATH WORKS Inc, *MATLAB User's Guide*, Prentice Hall, 1999.

[3] Kothari, University of Cincinnati, "Intelligent Systems ECECS 636," Unpublished.

[4] Rogers, "G/SPLINES: A Hybrid of Friedman's multivariate adaptive regression splines (MARS) algorithm with Holland's genetic algorithm." Research Institute for Advanced Computer Science, 1991.

[5] Waser and Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart and Winston, 1982.

[6] YSI Precision Temperature Group, *YSI Precision Thermistors & Probes*, 1999.

[7] Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[8] Back, "Self-adaption in genetic algorithms." European Conference on Artificial Life, pages 263-271, MIT Press, 1992.

[9] Juels and Wattenberg, "Stochastic hillclimbing as a baseline method for evaluating genetic algorithms." *Advances in Neural Information Processing Systems 8*, pages 430-436, MIT Press, 1995.

[10] Khuri, Back and Heitkotter, "An evolutionary approach to combinatorial optimization problems." Proceedings of the 22nd Annual ACM Computer Science Conference, pages 66-73, Phoenix, 1994.

[11] Pearl, *Heuristics*, Addison-Wesley Publishing, 1985.