CONVERSION FROM ENGINEERING UNITS TO TELEMETRY COUNTS ON DRYDEN FLIGHT SIMULATORS[1]

Jay A. Fantini
Woodside Summit Group Incorporated
Edwards, California

ABSTRACT

Dryden real-time flight simulators encompass the simulation of pulse code modulation (PCM) telemetry signals. This paper presents a new method whereby the calibration polynomial (from first to sixth order), representing the conversion from counts to engineering units (EU), is numerically inverted in real time. The result is less than one-count error for valid EU inputs. The Newton-Raphson method is used to numerically invert the polynomial. A reverse linear interpolation between the EU limits is used to obtain an initial value for the desired telemetry count. The method presented here is not new. What is new is how classical numerical techniques are optimized to take advantage of modern computer power to perform the desired calculations in real time. This technique makes the method simple to understand and implement. There are no interpolation tables to store in memory as in traditional methods. The NASA F-15 simulation converts and transmits over 1000 parameters at 80 times/sec. This paper presents algorithm development, FORTRAN code, and performance results.

KEYWORDS

PCM Telemetry, Polynomials, Roots of Equations, Numerical Analysis, Newton-Raphson Method, Flight Simulators, FORTRAN, Real-Time Operation

---

This document describes work sponsored by the National Aeronautics and Space Administration, Dryden Flight Research Center, Edwards, California, under contract DRC-098-68.

NOMENCLATURE

$[a,b]$	Closed interval bounded by a,b
CPU	Central processing unit
DFRC	NASA Dryden Flight Research Center
EU	Engineering units
MFLOPS	Millions of floating point operations per second
PCM	Pulse code modulation
$P_n(x)$	polynomial with real coefficients of order n
$\forall$	For all
ϵ	Element of
$\Re$	Real number
$\surd$	Square root

INTRODUCTION

NASA Dryden Flight Research Center (DFRC), Edwards, California, flight simulators encompass the simulation of pulse code modulation (PCM) telemetry signals. These signal streams contain data relating to accelerations, velocities, and other vehicle states. The streams, along with voice, video, and simulated radar data, are used to rehearse for actual flight missions and to train control room personnel, so they can become familiar with expected values and learn emergency procedures.

A PCM system is an electronic device mounted in a vehicle that converts analog measurements into digital *counts*, or scaled integers that represent state values of the vehicle. These counts are transmitted through a radio signal to a ground station for real-time tracking and data archival. Control room personnel use this information to determine if the vehicle is operating correctly. In a flight simulator, the vehicle states are calculated and stored as floating point numbers. The process of converting the analog measurements into counts is programmed into the flight simulation in order to mimic the behavior of a PCM system.

Before a PCM system can be used, it must first be calibrated. Calibration consists of matching count values to their corresponding real-world analog values. These values are called engineering units (EU). Ideally, the PCM system should cover the entire range of the expected values of the physical parameter being measured. The PCM systems measure their transmission resolution in number of bits. That means the device can count from 0 to $2^n - 1$ where n is the number of bits. The total counts is thus 2^n. The majority of PCM systems range from 10 to 20 bits of resolution. These counts correspond to ranging from 1 part in 1,024 to 1 part in 1,048,576 because $2^{10} = 10^{24} \approx 10^3$ and $2^{20} = 1{,}048{,}576 \approx 10^6$.

The zero value would correspond to the one extreme value of the physical parameter, and the $2^n - 1$ value would correspond with the other extreme value. In practice, the entire domain is not used. For example for a 10-bit PCM system, the count domain of interest might be from 50 to 800 instead of 0 to 1023. The majority of the time, the relationship between the counts and the EU is linear, In cases where the relationship is not linear, a polynomial curve fit is performed on the counts and EU data pairs. This paper addresses the question of conversion from EU to counts when the relationship is a polynomial. The polynomial is valid for all count values that are bounded by the lower count limit and the upper count limit. The returned value is the corresponding EU value. The form of the polynomial is

$$\mathrm{EU} = P_n(x) \forall x \in [L,U] \tag{1}$$

where x is counts, and L,U are count bounds.

The ideal solution is to solve equation (1) in closed form when given the coefficients. For polynomials of degree 5 or higher, a fast numerical method that yields the root that lies within the minimum and maximum counts value would be desirable. Modern computers, used for real-time flight simulation, compute at speeds over 20 MFLOPS/CPU (scalar) with 15 decimal digits of precision. In a typical flight simulation, multiple CPU are used to distribute the workload. One CPU simulates the engine; another, the aerodynamic forces on the aircraft; and still another, the navigation. A dedicated CPU is used for simulating the PCM stream.

A typical NASA DFRC simulation has 100 to 1500 parameters that are converted every frame. The frame rate ranges from 80 to 200 Hz. The conversion polynomials range from first to sixth order with the majority of nonlinear polynomials being second and third order. The simulation is hosted on a computer equipped with 128 MB of memory and four to eight 64-bit CPU running at 195 MHz. The computing power of one CPU is sufficient to give the simulation engineer the ability to numerically solve these polynomials in real time through a quadratic convergent or more efficient numerical method provided the polynomials fulfill certain conditions.

STATEMENT OF PROBLEM

The objective of this research is to develop a method of solving equation (1) for x numerically in real time, for each parameter, such that the absolute error of the solution is less than 1 count. The count domain, EU range, polynomial, and EU value to be converted into a count are known. The EU value of interest must he within the EU range. The main condition on the polynomial is that it is monotonic, increasing or decreasing for all counts within the count range. Here monotonic means that for counts within the counts

domain, the slope of the polynomial does not change sign, and the slope is never equal to zero over a non-zero interval. If this condition were not met, then there would be two different counts that correspond to the same EU value. In formal notation, the conditions are as follows:

EU_MIN, the minimum expected value of parameter being measured;

EU_MAX, the maximum expected value of parameter being measured

count value L such that $P_n(L) = \text{EU\_MIN}$;

count value U such that $P_n(U) = \text{EU\_MAX}$;

monotonic polynomial $P_n(x)\ \forall x \in [L, U]$, $n \geq 1$; and

$\text{EU} \in \Re$ such that $P_n(L) \leq \text{EU} \leq P_n(U)$.

Linear interpolation is a common technique currently used on flight simulators to perform the conversion. The basis of this technique is as follows:

1. During initialization, generate a table containing $(P_n(x), x)$ for $x = L$ to $x = U$ with m evenly spaced entries. The simulation engineer chooses the value for m. This value is based on accuracy needs and computer memory usage. Note that the table entries are reversed.

2. During run time, perform a linear interpolation using the EU value and the generated table to obtain the required counts value.

This technique has the advantage of being very simple to implement because it is the same technique used to obtain coefficient values from aerodynamic tables. The main disadvantages are as follows:

1. Depending on size of m for each parameter, the method consumes large amounts of computer memory. For example in cases with 1000 parameters, each parameter generates an interpolation table of 64 entries. Storage for 128 numbers is required for each parameter. Assuming each number is a floating point type (e.g. FORTRAN's REAL*4) and consumes 4 bytes of computer memory, 512,000 bytes (0.5 MB) is used for the entire simulation. Note that the coefficients used to generate the interpolation tables can be erased from computer memory once the tables have been generated.

2. Further complexity is introduced by using a different value for *m* for each parameter. This value would be another number for the simulation program to track.

3. Careful analysis must be made to ensure that *m* is large enough to return a result to within the desired error. To minimize the error for a given *m*, uneven spacing in the interval is needed.

4. The larger *m* is, the longer the interpolation takes. The majority of linear interpolation methods perform a linear search within the interval to find the correct subdivision before the actual interpolation takes place. The run time for this would be on the order of *m* ($O(m)$). For advanced search schemes, such as bisection, the run time would be $O(\log(m))$. Programming techniques can reduce these times; however, increased complexity and memory usage will result.

The method given in this paper overcomes these limitations. Memory usage is limited to storing the coefficients of $P_n(x)$, the order of the polynomial, the values of *L*, *U* and EU. Run time is a function of the order of the polynomial only for a given computer platform.

SOLUTION

The solution is to numerically solve equation (1) directly either by closed-form solution or with an iteration scheme. For polynomials of first and second order, the well-known linear and quadratic formulae are used. For third order and higher, the Newton-Raphson method is used.

In the linear case, the equation

$$y = ax+b \tag{2}$$

is solved through

$$x = (y - b)/a \tag{3}$$

which involves two floating point calculations.

In the quadratic case, only one of the roots of the quadratic formula is computed. For the equation,

$$y = ax^2 + bx + c \tag{4}$$

the general solution is

$$x = [-b \pm \sqrt{(b^2 - 4a(c - y))}] /(2a) \quad (5)$$

For the polynomial $P_2(x)$, only one of the roots lie in the interval bounded by L and U. One of the conditions cited by equation (1) is that $P_n(x)$ be strictly monotonic, increasing or decreasing for all x in $[L,U]$. This requirement prevents having the slope be zero within a subinterval of the stated interval. Thus, $P_n(x)$ will have a unique inverse for all EU in [EU_MIN, EU_MAX].

If $L \leq U$, then the desired root to $P_2(x)$ will be

$$x = [-b + \sqrt{(b^2 - 4a(c) - y))}] / (2a) \quad (6)$$

Otherwise, $P_2(x)$ will be

$$x = [-b - \sqrt{(b^2 - 4a(c - y))}] / (2a) \quad (7)$$

The most time-consuming part of this calculation for real-time computation is the square root.

For polynomials of third degree or higher, numerical methods are used. It is possible to solve third- and fourth-order polynomials in closed form, but the formulae involve radicals and trigonometric functions. This approach is too time consuming for real-time use and would not lend itself to being generalized for higher order polynomials. Because only the root that lies within $[L,U]$ is desired, a numerical method, such as Newton-Raphson or Halley's, can be used.

Two main concerns exist when using numerical techniques in real-time calculations: (1) convergence to root within precision and (2) time constraints and robustness of algorithm. Halley's method requires the calculation of not only $P_n(x)$ but also $\partial P_n(x) / \partial x$ and $\partial^2 P_n(x) / \partial x^2$. Halley's method enjoys cubic convergence; that is, the number of correct digits triple after every iteration for x near x_{ans} where x_{ans} is the root. In general, the quicker the method converges, the more critical it is that the initial approximation to the root be as close to the actual root as possible. This brittleness can be somewhat overcome by using the Newton-Raphson method[1] which is quadratic convergent. Newton's method is as follows:

$$\text{For } y = f(x) = 0,\ x = x - f(x) / s(x) \quad (8)$$

where $s(x) = \partial f(x) / \partial x$ for x_0 near x and

where x_0 is the initial approximation of the root.

For $P_n(x)$, the following two questions arise:

- How is x_0 selected when given $P_n(x)$, L, U?
- How are $P_n(x)$ and $\partial P_n(x) / \partial x$ evaluated quickly?

One of the conditions on $P_n(x)$ is that it be monotonic within $[L,U]$. From this condition, the existence of a unique inverse in the interval [EU_MIN,EU_MAX] is known. Assuming that

$$\text{EU\_MIN} \le \text{EU} \le \text{EU\_MAX} \tag{9}$$

where EU is the engineering units value whose corresponding count is sought, construct the equation of a fine using the points {(EU_MIN,L),(EU_MAX,U) giving

$$m = (U - L)/(\text{EU\_MAX-EU\_MIN})$$

$$b = \text{L} - \text{EU\_MIN}*m$$

$$x_0 = m * z + b \tag{10}$$

Evaluate equation (10) at z = EU to obtain the initial root to $P_n(x)$. Once x_0 has been derived from equation (10), iterate using equation (8), where $f(x) = P_n(x) - \text{EU}$ until convergent.

The evaluation of $P_n(x)$ and $\partial P_n(x) / \partial x$ can be sped up by use of Hörner's notation. For a polynomial,

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ...a_0 \tag{11}$$

The polynomial can be rewritten as follows:

$$P_n(x) = (((a_n x + a_{n-1})x + a_{n-2})\text{x} + a_{n-3}...) + a_0 \tag{12}$$

Equation (12) enables computation of $P_n(x)$, using n multiplications and n additions. To compute $\partial P_n(x) / \partial x$ through Hörner's notation, equation (13) is used.

$$\partial x^n / \partial x = n x^{n-1} \tag{13}$$

This computation gives

$$\partial P_n(x) / \partial x = [[[a_n n \cdot x + a_{n-1}(n - 1)]x + a_{n!2}(n - 2)]...a_1] \quad (14)$$

Equations (12) and (14) enable one to evaluate a polynomial and its derivative without direct calculation of integral powers. This evaluation results in increased performance as multiplication takes fewer CPU cycles than evaluation of powers.

Numerical experimentation has shown that four iterations are sufficient to obtain a result that differs from the actual root by less then 0.5 counts. This result is then rounded to the nearest integer before being returned to the calling program.

IMPLEMENTATION IN FORTRAN

The appendix contains the FORTRAN listing of the method as used at DFRC. This method consists of three routines: PLYSLV which solves the polynomial, PLYEVL which evaluates the polynomial at the selected point through Hörner's notation, and PLYSLP which evaluates both $P_n(x)$ and $\partial P_n(x) / \partial x$ at the same time to reduce subroutine calling overhead.

The inputs for PLYSLV are as follows:

CTMIN — The value of CT_MIN expressed as a double precision number.

CTMAX — The value of CT_MAX expressed as a double precision number.

COEF — The double precision array of coefficients of the polynomial where COEF(0) is the constant term.

N — The order of the polynomial expressed as an integer.

X — The EU value input.

The returned value is the solution of equation (15)

$$P_n(x) - \mathrm{EU}=0 \quad (15)$$

expressed as a double precision number where x is bounded by CTMIN and CTMAX

As an example to solve the polynomial

$$P_3(x) = 8.9617387\text{E}{-8}x^3 - 2.5674509\text{E}{-4}x^2 + 0.80511892x - 106.4193 \quad (16)$$

where CTMIN = 0, CTMAX = 1023

P_3 (CTMIN) = −106.4193, and P_3 (CTMAX) = 544.470472738 are generated. Solving for EU 0.0 is required. The root that lies in [−106.4193, 544.470472738] is x = 137.955 which when rounded is 138 counts.

PERFORMANCE

This method was tested on a computer with a 64-bit microprocessor running at 195 MHz. Compilation and optimizations flags were used to obtain maximum performance. A small main program was written that takes the following input:

- The polynomial coefficients and order of polynomial
- The values for EU MIN and EU MAX
- The number of steps, M

The program varies the EU value from EU_MIN to EU_MAX along M evenly divided steps. This variance prevents the optimizer from optimizing away any calculations and serves to simulate a slowly varying EU value. For all cases, M = 10^7.

Order of Polynomial	Solutions/Sec	Remarks
6	588,000	Highest order tested
5	714,000	
4	769,000	
3	909,000	Lowest order using iteration
2	3,030,000	Quadratic formula used
1	9,090,000	Linear solution

CONCLUSION

The method presented here to convert from engineering units values to counts have the advantages of (1) greatly reduced memory usage as linear interpolation tables are not used and (2) removal of errors caused by linear interpolation. These attributes allow the simulated pulse code modulated stream to be compared with actual aircraft-generated

streams to determine the difference in behavior between flight simulation models and actual aircraft. The main disadvantage of the method is that a great deal of computer power is required to solve the higher order polynomials in real time. As computer performance improves because of technical advances, the power demand of the method will be less of a concern.

REFERENCES

1. Hamming, R. W., *Numerical Methods for Scientists and Engineers*, second edition, McGraw-Hill: New York, 1973.

APPENDIX

The FORTRAN-77 listing for converting from engineering units to telemetry counts is given in this appendix.

```
      DOUBLE PRECISION FUNCTION PLYSLV(CTMIN,CTMAXCOEF,N,X)
      IMPLICIT NONE
      DOUBLE PRECISION CTMIN,CTMAX,X
      DOUBLE PRECISION ANS,A,B,C,T,D,E,Y,S
      DOUBLE PRECISION PMIN,PMAX,XM,XB
      INTEGER N,I
      DOUBLE PRECISION COEF(0:N)
      DOUBLE PRECISION PLYEVL

C     CTMIN and CTMAX are arranged such that
C     Pn(CTMIN) < Pn(CTMAX)

      ANS=0.0D0
      IF (N .EQ. 1) THEN
            A=COEF(l)
            B=COEF(0)
            ANS=(X-B)/A
            GOTO 900
            END IF

      IF (N .EQ. 2) THEN
            A=COEF(2)
            B=COEF(l)
            C=COEF(0)-X
            T=2.0D0*A
```

```
                D=-B
                E=DSQRT(B*B-4.0D0*A*C)
                IF (CTMIN .LE. CTMAX) THEN
                        ANS=(D+E)/T
                ELSE
                        ANS=(D-E)/T
                        END IF
                GOTO 900
                END IF

C       Perform a reverse linear interpolation
C       between the points [CTMIN,Pn(CTMIN)]
C       and [CTMAX,Pn(CTMAX)] using the value
C       of X (number to be solved for) to generate
C       X0 (initial value of counts value).

        PMIN=PLYEVL(COEF,N,CTMIN)
        PMAX=PLYEVL(COEF,N,CTMAX)
        XM=(CTMAX-CTMIN)/(PMAX-PMIN
        XB=CTMIN-PMIN*XM
        ANS=XM*X+XB
        DO 100 I=1,4
        CALL PLYSLP(COEF,N,ANS,Y,S)
        ANS=ANS-(Y-X)/S
100     CONTINUE

900     PLYSLV=ANS
        RETURN
        END

C       ************************************************************

        SUBROUTINE PLYSLP(C,N,X,S,T)
        IMPLICIT NONE
        DOUBLE PRECISION X,S,T
        INTEGER N,I
        DOUBLE PRECISION C(0:N)

C       This subroutine will compute the value of
C       Pn(x),dPn(x)/dx when given:
C       Coefficients with C(0) being the constant
```

```
C       term, N is the Order of the Polynomial
C       and X is the value to evaluate at.
C       Outputs: S is Pn(x), T is dPn(x)/dx
C       This subroutine is for solving polynomials
C       of order 4-6 for PCM conversion.
C       N must be > 1.
C       23-MAY-1997

        S=C(N)
        T=S*DBLE(N)
        DO I 00 I=N-1, 1,-I
        S=S*X+C(I)
        T=T*X+C(I)*DBLE(l)
100     CONTINUE
        S=S*X+C(0)
        RETURN
        END

C       ****************************************

        DOUBLE PRECISION FUNCTION PLYEVL(C,N,X)
        IMPLICIT NONE
        DOUBLE PRECISION S,X
        INTEGER N,I
        DOUBLE PRECISION C(0:N)

        S=C(N)
        DO 100 I=N-1,0,-l
        S=S*X+C(I)
100     CONTINUE
        PLYEVL=S
        RETURN
        END
```