Industrial Automation

Tech Note 27 Nonlinear Scaling

Abstract:

This document describes using Crimson® 3.0 programming to scale a nonlinear input signal.

Products:

G3 Series HMI / G3 Kadet HMI / Graphite® HMI / Graphite® Controllers / Modular Controller / Data Station Plus / ProducTVity Station

Use Case: Scaling a Nonlinear Input Signal

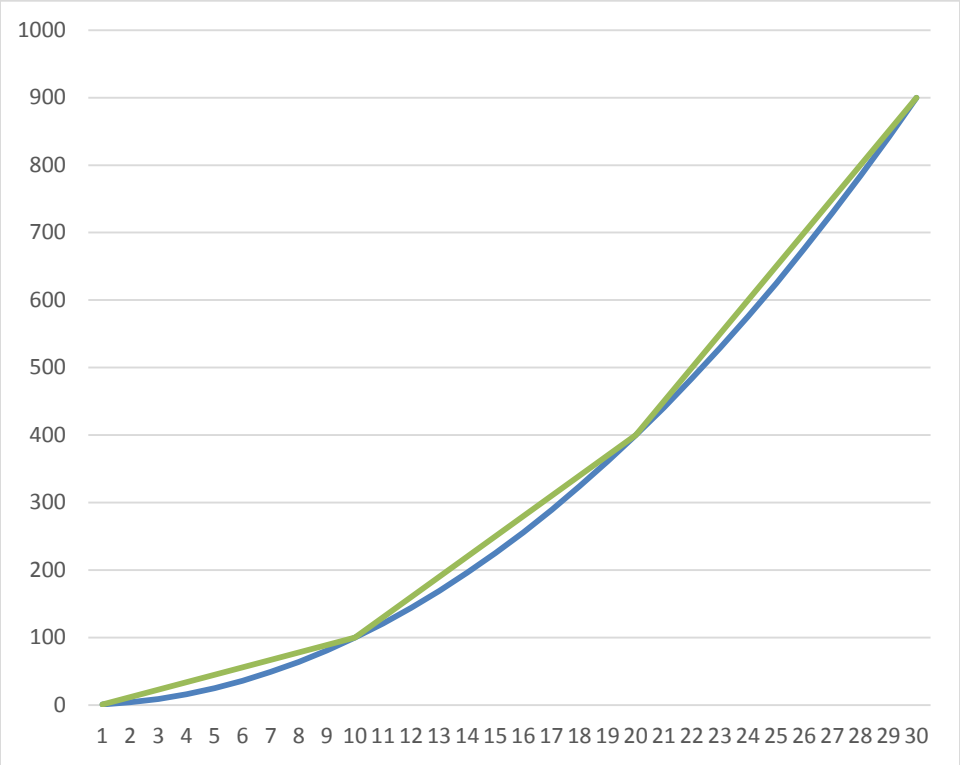
Many applications involve a nonlinear relationship between the input signal and the engineering units that it represents, such as the level of a conical or horizontally mounted cylindrical tank. This document explains how to scale the signal using Crimson programming, as well as replacing the functionality of the obsolete CSINI8L0 and CSINV8L0 modules.

Required Software:

Crimson 3.0

Theory

When the equation that relates the input signal to the engineering units is unknown the alternative is to divide the curve into multiple lines. The more lines that are used the closer the engineering units will be. In order to generate these lines, a series of X (input signal) and Y (engineering unit) coordinates are entered into arrays. The input signal is compared to the elements of the X array in order to find which line segment should be used to calculate the engineering units.



Blue Line: $y = x^2$ **Green Line** (4 points): 1,1; 10,100; 20,400; 30, 900

Application

1. Create and populate X and Y arrays.
   - The arrays can be exposed for entry from the user interface or populated by a program.
   - For a more accurate result, use more data points. The chart below shows the inaccuracy of using only 4 points compared to the true equation output:

```
// populate arrays with known data points

X[0] = 1;
Y[0] = 1;

X[1] = 10;
Y[1] = 100;

X[2] = 20;
Y[2] = 400;

X[3] = 30;
Y[3] = 900;
```

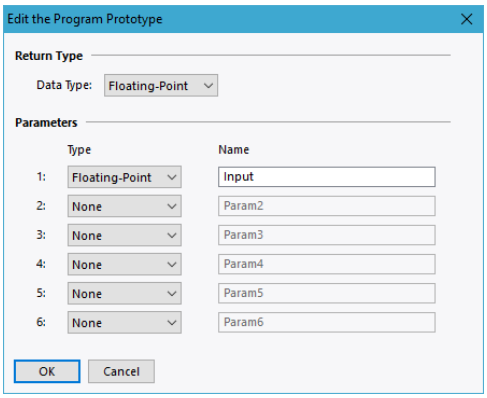
Input	Squared	4 Segment Calculation
1	1	1
2	4	12
3	9	23
4	16	34
5	25	45
6	36	56
7	49	67
8	64	78
9	81	89
10	100	100
11	121	130
12	144	160
13	169	190
14	196	220
15	225	250
16	256	280
17	289	310
18	324	340
19	361	370
20	400	400
21	441	450
22	484	500
23	529	550
24	576	600
25	625	650
26	676	700
27	729	750
28	784	800
29	841	850
30	900	900

2. Create the program (CalcVal).

a. Edit the program's prototype.

1) Return Type – Data Type: Floating-Point or Integer

2) Parameters – Floating-Point or Integer (Input)



b. Write the program code.

```
// declare locals
int i;
float PVSpan, InputSpan;

// loop until we get to the point in the arrays where the actual input value is
while((Input > X[i]) && (X[i] != 0)) i ++;

// if it is right on a point, return the display value with the offset
if(Input == X[i])
        return Y[i];

// if it is between points, calculate the display value
else {
        // calculate PV change in this segment
        PVSpan = Y[i] - Y[i-1];
        // calculate input change in this segment
        InputSpan = X[i] - X[i-1];
        // if there is an input change, calculate the PV (y=mx+b)
        if(InputSpan > 0)
                return  (((Input - X[i-1]) * PVSpan) / InputSpan) + Y[i-1];
        else
                return Y[i-1];
}

// if the input falls outside of the scaled range, return a high number
if((i != 0)&&(X[i] == 0))
        return 99999.1;
```

3. Use the program as the Source of a tag.

a. Create a new Numeric Tag.

b. Change its Source to General.

c. Type in the name of the program, with the name of the source value as its argument: CalcVal(Reading).

For more information: http://www.redlion.net/support/policies-statements/warranty-statement