BrianHetrick.com Application Note AN-2.0

A Naïve Non-Linear Optimization Algorithm and Implementation

Goal

The goal of this Application Note is to describe the design and construction of a naïve brute force non-linear optimization program. The program is written in the C# language and supplied as open source under the GNU Public License, version 3. As an example, the program is used to determine the coefficients of a thermistor response model.

Introduction

Occasionally, the programmer is faced with a modeling challenge: given a set of data, determine a mathematical model that fits the data. The mathematical model is of the general form:

$$\hat{y}_i = f(p_1, p_2, \ldots; x_i)$$

where (x_i, y_i) are the data pairs of the observed independent and dependent variable, $\hat{y}_i$ is the dependent variable value computed by the model for a given value of x_i, and the set of parameters $p_1, p_2, \ldots$ is to be found. The overall form of the function f might be suggested by theory, or might be entirely heuristic.

In general, this problem requires significant expertise in both numerical analysis and in the field from which the data are derived, and significant effort. However, such problems also arise as part of, for example, sensor calibration, which is typically a problem left to engineers. A computational model which happens to produce results with a specified accuracy for a specified range of independent variable values may be "good enough" for a particular application.

One method of producing such a computational model is to minimize the error of the model as a function of the set of parameters. The error of the model is generally a function of the error of the individual estimates, $y_i - \hat{y}_i$. For example, the common "least squares" optimization minimizes the sum

$$\sum_{i=0}^{n} (y_i - \hat{y}_i)^2$$

over the set of observations $\{(x_i, y_i)\}$. This error value may be regarded as a function of the model parameters $p_1, p_2, \ldots$. The modeling challenge is then to adjust the values of the parameters so as to minimize the error value.

This is an optimization problem, and generally a non-linear one. This Application Note describes a program which attempts to minimize value of an arbitrary objective function of a set of parameters $p_1, p_2, \ldots$. The Application Note also shows as an example using this to compute model parameters for a particular set of data.

The Optimization Algorithm

Entire books have been written on optimization algorithms and the circumstances in which they are useful. One popular algorithm is known as steepest descent. This algorithm examines the behavior of the objective function in the neighborhood of a current point, then adjusts the current point such that the objective function decreases by the largest amount possible. This process is repeated until the objective function does not decrease anywhere in the neighborhood of the current point. This algorithm is simple to implement and generally quite robust. It is not, however, particularly speedy.

There are several choices which must be made when implementing this algorithm. One critical choice is the step size. At what distance from the current point are the neighborhood points? Generally the step size should be relative to the magnitude of the current point. However, the step size should not be only rela-

tive to the magnitude of the current point, for then zero-crossing becomes impossible. The algorithm implemented in the program described in this Application Note attempts to balance these concerns by using a step size depending on the order of magnitude of the value being stepped, with an imposed minimum step size.

A second critical choice is the size of the neighborhood investigated. Ideally the neighborhood would be large enough for the algorithm to find its way out of local minima and pursue a global minimum. However, the computational expense of investigating the neighborhood increases exponentially as the size of the neighborhood increases. The algorithm implemented in the program described in this Application Note uses defines neighborhood points as one or two steps away from the current point in any combination of directions. The number of steps is however easily changed.

The Implementation

The algorithm implementation models the parameters of the objective function as decimal floating point values with a restricted number of digits and a restricted minimum exponent value. The step size applied to a given parameter is one least significant digit. The values are normalized when possible. Values which if normalized would require an exponent value less than the minimum value are carried as unnormalized values.

For example, consider the default case of four significant digits and a minimum exponent of -8. The smallest positive normalized value is 1.000E-8, while the smallest non-zero positive value is 0.001E-8. Adjacent values to 1.000E0 are 1.001E0 and 9.999E-1; adjacent values to 0.001E-8 are 0.002E-8 and 0.000E-8 (zero).

The algorithm evaluates the objective function in the immediate neighborhood of the current point. Points in the immediately neighborhood are derived from the current point by finding values one or two steps adjacent to the current parameter values. For example, the immediate neighborhood of (1.234E0, 1.000E0) is the set of points {(1.232E0, 9.998E-1), (1.232E0, 9.999E-1), (1.232E0, 1.000E0), (1.232E0, 1.001E0), (1.232E0, 1.002E0), (1.233E0, 9.998E-1), ... (1.236E0, 1.000E0), (1.236E0, 1.001E0), (1.236E0, 1.002E0)}. The number of points in the immediate neighborhood of the current point is $5^k - 1$, where *k* is the number of parameters. The algorithm enumerates these points with a length *k*, radix 5 counter.

The implementation depends on a user-supplied starting point and a user-supplied objective function. The user-supplied starting point and the arguments to the user-supplied function are normal float values. The implementation translates float values to and from the internal decimal floating point values as required.

The implementation source code is available as the companion ZIP file to this document. It is open source licensed under the GNU Public License, version 3.

An Example

Thermistors are temperature-dependent resistors: the resistance of the device depends upon its temperature. The resistance of common negative temperature coefficient (NTC) thermistors common follows the Steinhart-Hart equation[1]:

$$\frac{1}{T} = a + b\ln(R_T) + c(\ln(R_T))^3$$

where T is the temperature in degrees Kelvin, R_T is the device resistance in ohms at temperature T, and a, b, and c are device-dependent coefficients. The device-dependence of the coefficients frequently only corrects for a device-dependent multiplicative factor; the ratio R_T/R_0 of the resistance at a temperature T to the resistance at a specific temperature T_0 is formulation-dependent, not device-dependent. Therefore, the temperature can be computed from a measured resistance by the model

$$T = \frac{1}{a + b\ln(\frac{R_T}{R_0}) + c(\ln(\frac{R_T}{R_0}))^2 + d(\ln(\frac{R_T}{R_0}))^3}$$

where R_T is the measured resistance of the device, R_0 is the device-specific resistance at a specific temperature T_0, and a through d are coefficients that depend on the technology and the reference temperature T_0 but *not* on the particular device. Thus a particular part is characterized by a single quantity R_0 rather than by three coefficients. (Note this second equation has a squared term, unlike the Steinhart-Hart equation.)

The US Sensor catalog[2] provides resistance-to-temperature tables for their precision thermistors. This cata-

1 Wikepedia. "Thermistor." *Wikipedia*. 28 August 2008. 6 September 2008. <http://en.wikipedia.org/wiki/Thermistor>.

2 US Sensor Corp. "US Sensor Catalog." No publication date. 6 September 2008. <http://www.ussensor.com/US_SENSOR_CATALOG.pdf>.

log also provides resistance ratio-to-temperature tables for their standard thermistors. This data for a particular family of inexpensive parts was extracted for the temperature region of interest. The temperature region is that of temperatures commonly observed outside in the continental United States, -50°C (-58°F) to 55°C (131°F). The tabulated data are as follows:

°C	R_T/R_0
-50	67.0200
-45	47.2000
-40	33.6500
-35	24.2700
-30	17.7000
-25	13.0400
-20	9.7080
-15	7.2960
-10	5.5330
-5	4.2330
0	3.2650
5	2.5400
10	1.9900
15	1.5710
20	1.2490
25	1.0000
30	0.8057
35	0.6531
40	0.5326
45	0.4368
50	0.3602
55	0.2986

The non-linear optimization program was then used to minimize the sum of squared error of a model using the second equation above. The parameters computed for the model were:

Parameter	Value
a	3.354E-03
b	2.564E-04
c	2.394E-06
d	9.165E-08

The temperatures predicted by the model with the resulting coefficients was then computed for the appropriate resistance ratios, and compared with the tabulated temperature. The predicted temperature differed from the tabulated temperature by no more than 0.01°C over the range of interest.

Summary

This Application Notes describes an implementation of a naïve steepest descent optimization algorithm. The algorithm determined a set of coefficients which would be useful for using a thermistor to measure environmental temperature. The software implementing the optimization algorithm is available under the GPL version 3.

Further Information

This paper, source code for the program, and the OpenOffice.org spreadsheet comparing the actual data with the computed data are available at http://www.brianhetrick.com/tr/AN-2.zip.

Copyright and Grant of License