Solving statistics problems with symbolic C# framework

Sergey L. Gladkiy

Introduction

Statistical analysis is an important part of many commercial applications: scientific, economical, spreadsheets, databases and other. Modern commercial applications are commonly built with using many specialized frameworks for implementing different modules, including statistical data processing.

This article describes the capabilities of the symbolic C# framework **ANALYTICS** for solving statistical problems. The framework is written entirely in C# and its uniqueness is that it contains as predefined methods for statistical data analysis as the symbolic evaluation methods and various additional numerical tools.

Background

The **ANALYTICS** framework is a general purpose symbolic library (http://sergey-l-gladkiy.narod.ru/index/analytics/0-13). The main functionality of the library is evaluating symbolic (analytical) math expressions. For an example, there is a simple code for dealing with math expressions:

```
Translator t = new Translator();
t.Add("A", 0.5);
t.Add("x", 12.4);

string f = "A*ln(Pi*x)";
double y1 = (double)t.Calculate(f);
t.Variables["x"].Value = 27.3;
double y2 = (double)t.Calculate(f);
```

Here the ***Translator*** is one of the main classes of the framework that realizes most of the symbolic capabilities. The code adds two variables to the ***Translator*** instance: '**A**' and '**x**'. Then it evaluates the symbolic expression using current variable values, changes the value of the '**x**' variable and evaluates the same expression again.

One of interesting features of the library is that it allows calculating **symbolic derivatives** of math expressions:

```
Translator t = new Translator();
string f = "A*e^(π*x)+B*sin(3*x^2)";
string df = t.Derivative(f, "x");
```

The result of the symbolic derivative calculation is '**e^(π*x)*π*A+cos(3*x^2)*6*x*B'**. Let us note here, that the derivative calculation algorithm makes some simplification for the result expression. For the example above, the simplification is the '**6**' multiplier which is got as the product of the coefficient '**3**' and the power '**2**'.

The **ANALYTICS** framework has other unique features. First, it allows evaluation of math expression not only for real values, but for other types, including complex values, vectors and matrices. Moreover, it allows creating extensions for making evaluations with any user defined data types (more information here http://sergey-l-gladkiy.narod.ru/ana_docs/ANALYTICS_C_manual.en.pdf). Also the library includes many ready-to-use numerical tools, totally integrated with the symbolic capabilities: linear and nonlinear least squares approximation, numerical integration, solving initial value problems for ordinary differential equation systems, solving systems of nonlinear equations, finding roots and extremums of functions.

The library includes specialized package for statistical data analysis: calculating common statistical characteristics, like mean value, mode, median, deviation and variance; creating sequences of special numbers and progressions; using various probability distributions, like Gauss, Laplace, Gumbel, logistic and other.
All these features of the framework allows solving a wide range of statistical problems. The problems can be solved as using predefined statistical functions, as using advanced symbolic evaluations.

Solving simple statistical problems

First, let us consider solution of simple statistical problems with using as symbolic capabilities as the specialized package. Let we have an array of real values and we need to calculate the sum of array's elements which are less than some value. We can use only symbolic evaluations for solving the problem:

```
double[] a = new double[16] { 0.25, 0.35, 0.04, 0.01, 0.15, 0.2, 0.2, 0.1, 0.1, 0.15, 0.25, 0.12,
0.1, 0.02, 0.4, 0.05 };
Translator t = new Translator();
t.Add("A", a);
t.Add("max", 0.1);
string f = "∑if{A<max}(A 0)";
double S = (double)t.Calculate(f);
```

The result of the evaluation is '**S = 0.12**'. The evaluation uses function '**if**' with condition '**A<max**' that compares all elements of the array '**A**' with the '**max**' value. The function returns array of the same length as '**A**' with elements from the array, if the condition satisfied, and 0 if not. Then the sum operator '**∑**' makes summation of all result elements getting the required problem solution.
Next problem seems to be very simple – calculate mean value of a real array. But statistical data analysis defines many 'meanings' of mean value - arithmetic mean, geometric mean, harmonic mean, quadratic mean (https://en.wikipedia.org/wiki/Mean#Power_mean). The special statistics package of the **ANALYTICS** library contains functions for evaluating any of the mentioned means. So, the problem solution can be done using one simple formula:

```
double[] a = new double[16] { 0.35, 0.15, 0.4, 0.02, 0.05, 0.25, 0.12, 0.1, 0.1, 0.25, 0.25, 0.1,
0.1, 0.2, 0.04, 0.5 };
Translator t = new Translator();
t.Add("A", a);
string f = "Mean{1}(A)";
double AM = (double)t.Calculate(f);
```

Here '***Mean{P}(A)***' is a parametric function with parameter '***P***' for 'power' of the mean as explained in the reference above. In our case '***P=1***' means evaluating simple arithmetic average.
But the problem of mean value calculation can be made more complicated using the concept of **generalized mean**. The generalized mean defined with the following formula (https://en.wikipedia.org/wiki/Mean#Generalized_means):

$$mean = f^{-1}\left(\frac{1}{N}\sum_{i=1}^{N} f\left(A_i\right)\right)$$

where **f** is some appropriate function and $\mathbf{f}^{-1}$ is the inverse of **f**.

Let us state the problem of calculation of 'exponential' mean. That is the function **f** is exponent and its inverse is natural logarithm. Unfortunately, the statistics package does not allow evaluation of the mean. But one of the advantages of using the symbolic framework is that you can write the formula for the evaluation.

```
double[] a = new double[16] { 0.35, 0.15, 0.4, 0.02, 0.05, 0.25, 0.12, 0.1, 0.1, 0.25, 0.25, 0.1,
0.1, 0.2, 0.04, 0.5 };
Translator t = new Translator();
t.Add("A", a);
string fe = "ln(∑(e^A)/#A)";
double EM = (double)t.Calculate(fe);
```

here prefix operator '**#**' returns the number of elements in '**A**' array ('**N**' value in the formula for generalized mean). As it is expected, we get different values for arithmetic mean and exponential mean: 0.18625 and 0.195535456077713 respectively.

The example of generalized mean shows that the framework allows solving base statistical problems using predefined function with one formula, as more specific 'nonstandard' problems can be solved using additional symbolic capabilities of the library.

Solving advanced statistical problems

Now, let us consider some complicated statistical problems that can be hardly solved without using specialized statistical package. Some of such problems concern probability distributions.

Let we know that some bacteria species life-time described by normal distribution with parameters μ=4 hours and σ=1 hour (https://en.wikipedia.org/wiki/Normal_distribution). The problem is calculating the probability that the bacteria dies in the interval t_1=3 and t_2=3.5 hours. This probability can be calculated using the following formula:

$$Pr\left[t_1 \le X \le t_2\right] = P\left(t_2\right) - P\left(t_1\right)$$

where P_1 and P_2 are the probability that the bacteria lives up to the t_1 and t_2 respectively. These two probabilities can be calculated with the cumulative distribution function **CDF** (https://en.wikipedia.org/wiki/Cumulative_distribution_function). The statistics package of the framework contain many predefined CDF's for commonly used distributions, including the normal (Gaussian) distribution. So, we can solve the stated problem using the following code:

```
Translator t = new Translator();
t.Add("μ", 4.0);
t.Add("σ", 1.0);
t.Add("t1", 3.0);
t.Add("t2", 3.5);
string fP = "ΔGaussCDF{μ σ}(t1 t2 2)";
double P = ((double[])t.Calculate(fP))[0];
```

First, we added variables for the parameters of the problem. The function '***GaussCDF{μ σ}(t1 t2 2)***' produces array of Gaussian CDF on the interval **[t1, t2]** with 2 values. The delta '**Δ**' operator calculates difference between these 2 values getting the array with 1 item, which is the solution result. For our data values the probability that the bacteria dies in the specified time interval is about 15%.

The problem solution is very simple using the predefined function for normal CDF. But we should note here that the implementation of the function is not simple, because it requires evaluation of special function – so called **Error function** (https://en.wikipedia.org/wiki/Error_function). For other distributions the CDF can require other special functions.

So, let us consider alternative solution of the same problem. The required probability value can also be calculated with the following formula:

$$Pr\left[t_1 \le X \le t_2\right] = \int_{t_1}^{t_2} f(x)\,dx$$

where **f(x)** is the probability density function of the distribution (https://en.wikipedia.org/wiki/Probability_density_function). The difference from the previous formula is that the integral used. But in the case of normal distribution the PDF is very simple compared with CDF:

$$f(x) = \frac{1}{\sqrt{2\cdot\pi\cdot\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\cdot\sigma^2}\right)$$

In the last formula only standard functions and operations used. Thus, we can use alternative formula for solving the same problem about life-time probability of bacteria species:

```
Translator t = new Translator();
t.Add("μ", 4.0);
t.Add("σ", 1.0);
t.Add("t1", 3.0);
t.Add("t2", 3.5);

string fx = "Array{t1 t2}(100)";
double[] x = (double[])t.Calculate(fx);
t.Add("x", x);
string f = "1/√(2*π*σ^2)*exp(-1/(2*σ^2)*(x-μ)^2)";
double[] fv = (double[])t.Calculate(f);
t.Add("f", fv);
string fp = "∑f*(t2-t1)/#f";
double p = (double)t.Calculate(fp);
```

The first function '***Array{t1 t2}(100)***' created the array of 100 uniform values on the interval **[t1, t2]**. Then we calculated 100 values of the Gaussian PDF using the formula above. And finally, we integrated the function using appropriate summation instead the definite integral – '***∑f*(t2-t1)/#f***'. The value of the calculated probability is the same as with the previous method – about 15%. So, the problem solved using standard functions of the framework, but the solution is slightly longer than using specialized statistical package.

Another common functionality of statistical packages is random numbers generation. Pseudo-random numbers used, for an example, in various Monte-Carlo modeling methods and this requires generation of values for different distributions. The statistical package of **ANALYTICS** framework allows generating random numbers for 6 various distributions, including normal Gaussian. The following code calculates histogram of Gaussian distribution:

```
Translator t = new Translator();
t.Add("μ", 4.0);
t.Add("σ", 1.0);
string fr = "GaussRnd{μ σ}(0 10 1e6)";
double[] r = (double[])t.Calculate(fr);
t.Add("r", r);
string fh = "Histogram{10}(r)";
double[] h = (double[])t.Calculate(fh);
```

The first function '***GaussRnd{μ σ}(0 10 1e6)***' created the array of one million random values distributed with Gaussian function of specified parameters on interval **[0..10]**. Then histogram of the random samples calculated for 10 subintervals. Here is the histogram values:

(862 12213 77663 236536 342959 237951 78421 12407 950 38)

The source code of the examples (VS 2010 solution) can be found in the download for the article.

Conclusions

This article introduced the capabilities of the symbolic framework **ANALYTICS** for solving statistical problems. The framework is written entirely in C# (as Delphi and Java version also available) and allows statistical data processing with many predefined functions and with advanced symbolic evaluations. Special statistical package for the framework includes functions for calculating common statistical characteristics, generating sequences of special numbers and progressions and using probability distributions. The symbolic framework **ANALYTICS** can be found here http://sergey-l-gladkiy.narod.ru/index/analytics/0-13.