A Library of Function Classes

J. Boudreau[†], M. Fischler[‡], P. Maksimović[§]

[†] *Department of Physics and Astronomy, University of Pittsburgh, Pittsburgh PA 15260*
[‡] *Fermi National Accelerator Laboratory, Batavia Il 60510*
[§] *Department of Physics, Harvard University, Cambridge MA 02138*

Abstract. We have written a library of function classes in C++, which is distributed as part of the CLHEP foundation class libraries for High Energy Physics. The main goals of the library are to provide objects having all the mathematical behavior of functions, and to provide a mechanism for parameterizing these functions. Our functions support all the normal function operations such as addition, subtraction, multiplication, division, composition, et. cetera. The library allows programmers to compose complicated one- or multi-dimensional functions with great economy and natural semantics.

Why Function Objects?

A common computing task is to compute a function, the actual choice of which is delayed until runtime. A built-in mechanism, pointers-to-functions, exists in C, C++, and FORTRAN[1]. A function pointer can refer to a function of one or more variables, and procedures can be written with a function-pointer interface, allowing the same procedure to work with multiple functions.

However this approach is severely limited, because function pointers lack most of the mathematical behavior expected of functions of real arguments. They cannot be added, subtracted, multiplied and divided. There is no law of composition. And they cannot have state, specifically, parameter values that govern the shape of the function. In FORTRAN or C there is little that one can do to remedy this. However in C++ it is possible to create function objects which behave properly under arithmetic operations, where the arithmetic is expressed in overloaded operators. This is useful as a computing tool in that it allows clients to build a complicated function out of simpler functions without writing new subroutines or classes.

Imagine, for example, fitting data Gaussian function plus an exponentially falling background. With function objects one could construct a fitting function by writing

```
f = exp + gauss;
```

passing the result to the fitting procedure. While this may look like floating point arithmetic, the actual type of each object is "`AbsFunction`", the name of our data type representing function-objects. The fitting procedure itself evaluates the function using the function call operator

```
double y = f(x)
```

as many times as required in order to minimize χ^2 with respect to the *state* of the Gaussian and exponential functions). In this simple example the desired function can be produced simply without introducing new subroutines and without inheriting any base classes. Working with a class library of function objects is like embedding a symbolic manipulation program in C++. The technique is similar to one described in reference (3).

Our Implementation

Our implementation of this idea is a package called GenericFunctions(1). The package consists of:

- A small number of abstract base classes representing functions, their arguments (in case these are multidimensional), and their parameters.

- Classes which represent unary and binary operations on functions, between functions and parameters, and between parameters.

- A small set of concrete function-objects.

Both the second and the third category of classes inherit from the first. The second category provides all the machinery which makes the functions behave as described above. Functions in the third category do not have to each be taught about algebraic rules: they inherit them

[1] For people literate in the body of software used in particle physics, the best known example is probably the function pointer FCN used in MINUIT. For others perhaps the best example is the qsort library routine.

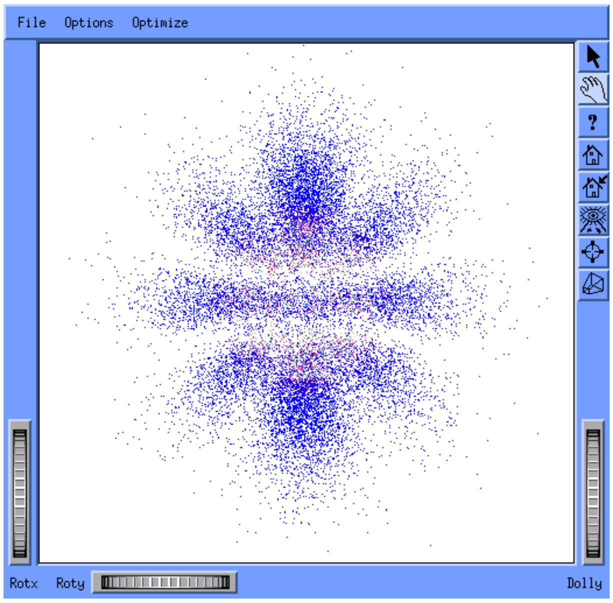

FIGURE 1. Multidimensional functions can be expressed as a direct product of lower dimensional functions. This hydrogen wavefunction was described as a direct product of lower dimensional functions.

automatically from the base classes. This category is designed to incorporate concrete function-object contributions from third parties. The recipe for extending the abstract base classes is documented in the form of a short checklist.

The operations we have defined are `+, -, *, /`, as well as `()` (function composition, allowing one to write constructs such as: `ln(tang(-x/2))`, with `ln`, `tang`, and `x` being function-objects).

Functions of more than one variable are supported, and derive from the same abstract base class as one-dimensional functions. We use the operator `%` to represent direct product of two functions. In some cases, n-dimensional functions may be expressed as direct products of lower dimensional functions: in two dimensions this happens whenever we may write $h(x,y) = f(x) * g(y)$. The direct product operator allows one to produce such functions easily when the lower-dimensional functions are available. Fig. 1 shows an example, a plot of a hydrogen $|\Psi|^2$ in an 5P state, a direct product of an angular wavefunction involving Legendre Polynomials and a radial wavefunction involving exponentials, power functions, and LaGuerre polynomials.

The mechanism by which functions are given state is the `Parameter` class. `Parameters` are like double precision constants, except that 1) they have upper and lower limits, and 2) they can be "connected" to other parameters; in other words, told to take their value from somewhere else. Parameters can be used in arithmetic expressions together with functions and together with built-in floating-point types. And, they can be private member data of function objects. If a function parameters are connected and the function is subsequently used in an expression, the expression is now also controlled by the parameter. More details on how to use the library are given in the package documentation(1).

We have implemented a small set of functions for the first release of the library. This now includes the functions in Table 1. Improvements and extensions are in progress.

How it works

Any expression involving functions and/or parameters is stored as an expression tree which is used later in evaluating the function. The tree for the simple expression `f=sine(x)/x` is shown in Fig. 2. The tree is built in parts, and each time a new operation is applied sub-trees are copied. Thus it is slow to build an expression tree. In evaluating the function, the expression tree is navigated and the return value for the entire function tree is computed recursively. This step incurs overhead from virtual function calls, but one has to compare this to the time spent in evaluating the functions. In case one is evaluating trigonometric or other higher functions the overhead is negligible. Good numerical algorithms are the key to efficiency.

Availability and Distribution

The GenericFunctions is now part of the CLHEP class library(2), organized and edited by an international organizing committee and editorial board, distributed through CERN. The GenericFunctions package lives in a namespace.

Future directions

We envision extending this library by requiring function objects to specify derivatives which allows automatic differentiation of *all* functions, simple or composite. This also permits function approximation, and allows high-speed local approximation of computationally intensive functions, using a few terms in a Taylor series. We can foresee extending the library by including operators, endowed with their own algebra. It is possible to add symbolic manipulation rules to the package, so that expressions such as $sin^2(x) + cos^2(x)$ are simplified and the ex-

Table 1. List of functions presently implemented.

Class	Implementing
Variable	the variable itself
Const	fixed constants
Power	power functions
Sqrt	square root
AnalyticConvolution	Moser-Roussarie functions
Exponential	Positive Exponentials (zero at negative lifetime)
ReverseExponential	Negative Exponentials (zero at positive lifetime)
Exp	standard exponential
Sin	sine function
Cos	cosine function
SphericalBessel	spherical Bessel functions
SphericalNeumann	spherical Neumann functions
Rectangular	rectangular functions
Periodic Rectangular	periodic rectangular function
LogGamma	natural log of Gamma function
IncompleteGamma	incomplete Gamma function
Gaussian	Gaussian or normal function
Erf	Error function
CumulativeChiSquare	Cumulative χ^2 distribution function
AssociatedLaguerre	Associated Laguerre Polynomials
AssociatedLegendre	Associated Legendre Polynomials

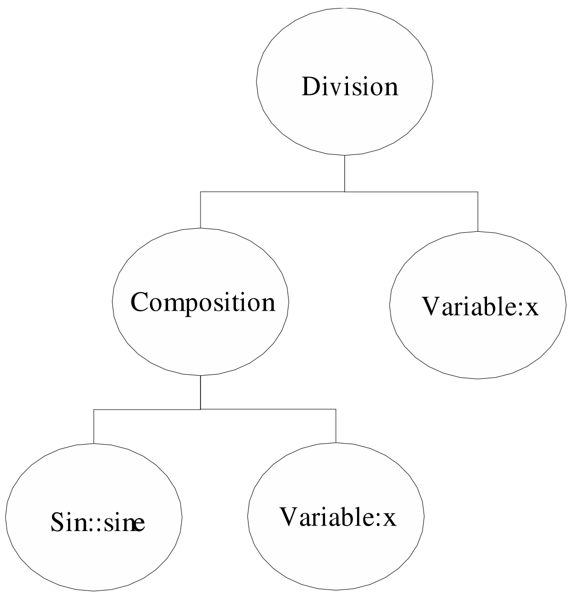

FIGURE 2. The expression tree representing the function sine(x)/x.

pression tree pruned accordingly. These goals will be addressed according to our own needs and public demand.

Acknowledgments

We wish to thank Andreas Pfeiffer and Evgueni Tcherniaev for their collaboration, and Walter Brown and Bob Jacobsen for illuminating discussions.

REFERENCES

1. *www.fnal.gov/tf/zoom/Pkg/CLHEP/docs/GenericFunctions/0Generic.html*.
2. *wwwinfo.cern.ch/asd/lhc++/clhep/, or from FNAL at www.fnal.gov/tf/zoom/Pkg/CLHEP/doc/html/0CLHEP.html.*
3. Barton, J., and Nackman, L., *Scientific and Engineering C++*, Reading, Massachusetts: Addison-Wesley, 1994.