DEVELOPMENTS OF MATHEMATICAL SOFTWARE LIBRARIES FOR THE LHC EXPERIMENTS

M. Hatlo, F. James, P. Mato, L. Moneta, M. Winkler, A. Zsenei, CERN, Geneva, Switzerland

Abstract

This paper describes the activities performed by a working group formed with the objective to support and develop mathematical software for the LHC experiments. The goal is to share common mathematical libraries between the experiments and the LCG application area projects. We present the various library components and their design, with particular emphasis on fitting and minimization, describing the new object oriented implementation in C++ of the MINUIT minimization package.

INTRODUCTION

A work package to support and develop mathematical software libraries for the LHC experiments has been initiated following the recommendations of the Requirements and Technology Assessment Group (RTAG) of the LHC Computing Grid project (LCG). This work package is part of the LCG SEAL project [1], which mandate is to provide the software infrastructure, basic frameworks, libraries and tools that are common among the LHC experiments. Developments of mathematical and statistical libraries used in physics analysis, event reconstruction and simulation are managed in this project with the goal to provide a coherent system to end-users and avoid maintenance and support of various mathematical libraries providing similar functionality. The activities are coordinated with existing projects in the high energy physics (HEP) community providing as well mathematical software libraries, such as ROOT [2], bringing the needs of experiments and other software projects together.

C++ MATHEMATICAL LIBRARIES

Inventory of Mathematical Libraries

To address the question of what the needed mathematical functionality for the HEP community, we have started looking at what is provided by the Fortran CERNLIB library [3]. We have also investigated the mathematical functionality present in ROOT [2] and in CLHEP [4] and compared with what is provided by general purpose mathematical libraries like the open source GNU Scientific Library (GSL) [5], or the commercial Nag C library [6]. We have then compiled an inventory of needed mathematical functions and algorithms and made it available online at the MathLib Web site [7]. For each entry, we reference the software packages (mainly CERNLIB, GSL and ROOT) implementing the function or algorithm and we add links to their documentation. As we will advance implementing these functions, we will also include links to our implementation. We plan to add as well links to the results of the comparison and validation studies we are performing.

Mathematical Library Components

The basic components that constitute the C++ mathematical libraries, shown in the diagram of Figure 1, are the following:

- **Mathematical functions**: library of special functions and statistical functions needed for HEP.
- **C++ Function classes**: library of classes describing generic funcitons, parametric functions or probability density functions.
- **Numerical algorithms**: library containing the methods for numerical integration, differentiation, function minimization, root finders, interpolators, etc..
- **Linear algebra**: library with vector and matrix classes and their operations.
- **Random number generator**: library with collections of generators and methods for generating random numbers according to distributions.

Mathematical Functions

This library includes special functions and other mathematical functions used mainly in statistical problems and needed by the HEP community. The software provides the means of evaluating a function at a point, satisfying one of the most common use case. It forms the functional layer of the system and it is composed of a set of free functions, with a coherent name schema based on the function names and grouped together in a common namespace. This design approach gives the possibility for users of extending the library by adding or overloading new functions in the same namespace. Moreover, the underlying implementation is hided behind and it can be replaced easily when needed. At the moment, we are planning to implement a large majority of these functions as wrapper to GSL [5] functions, which are written in C. If we want to replace them with a different implementation, this will be transparent for the user and without modifications on the interfaces. The name schema used to describe the functions will be the same of the existing proposal [8] for adding special functions to the C++ Standard Library.

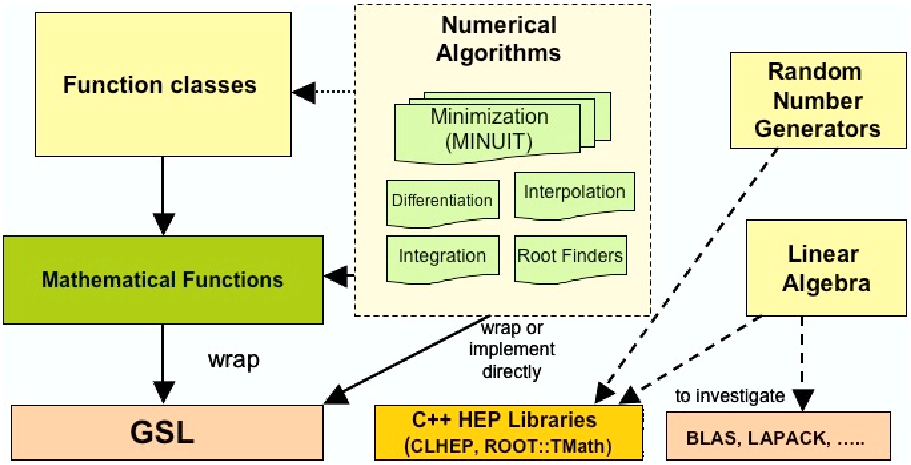

Figure 1: Schema showing the Math C++ component libraries and their relationships

C++ Function classes

When using numerical algorithms, it is convenient to have a common way of dealing with the various types of functions. This library provides the C++ classes and interfaces describing the function types needed by the algorithms. Evaluating a function at a point is a common use case but not the only one. Use cases exist which require additional operation on functions, such as controlling their shape to model the data in fitting problems or re-normalizing them in a particular range. Furthermore, we can exploit, for user convenience, the functionality provided by C++ to overload operators, to implement arithmetic operations, function composition and convolution. We define here a set of abstract interfaces and base classes to describe the various functions. The abstract function interfaces describe generic functions for the one and multi-dimensional case. Base classes are parametric functions which contain as attributes the function parameters and provide methods to set and retrieve them. They are used for example in fitting problems to model the data (model functions). Concrete classes are predefined functions such as Gaussian, Exponential, Polynomial, etc.., which can be used, through the implemented function operations, to compose more complicated functions. This library will be based on the Mathematical Functions layer.

Numerical Algorithms

This software component contains numerical methods for solving mathematical problems needed by the HEP community. These methods include numerical integration and differentiation, function minimization, function approximations, root finders, interpolation, differential equation solvers and Fast Fourier transforms. The linear algebra algorithms are not considered here since they are included in the linear algebra library. Some of these numerical methods are rather complex and they constitute a library per se, like the MINUIT C++ library for function minimization. The various algorithms make use of the abstract function interface. For maximising efficiencies in CPU performances and flexibility of usage, generic functional interfaces to the algorithm exist and they are based on template methods or on function pointers. Algorithms hide using well defined interfaces their implementations and they can be loaded at run time using the plug-in manager. This capability is provided for example in the fitting library (FML) which will be described in the following section. Some of the algorithms are being implemented as C++ wrappers (or adaptors) to the GSL [5] C algorithms, while others like MINUIT are re-implemented from scratch in C++.

Linear Algebra

In the case of linear algebra, it is convenient to have C++ matrix and vector classes and implement, through operator overloading, the arithmetic operations. It is essential to have maximum performances, since linear algebra algorithms are used extensively in event reconstruction where CPU performances play a crucial role. Our first goal here, before developing a new linear algebra library, is to evaluate the existing ones. We study their functionality and performances in realistic HEP test cases, such as track reconstruction. For the packages written in C (GSL [5]) or Fortran (BLAS, LAPACK [9]), we have developed a thin C++ wrapper layer with matrices and vector classes based on expression templates. We measure the CPU time spent in the combination of matrix operations, which are similar to those applied in the Kalman filter. We have compared, varying the matrix sizes, the following packages: uBLAS from Boost numerics [10], BLAS/LAPACK [9], CLHEP [4], GSL [5] and the new linear algebra package from ROOT [11]. For the specific case of track fitting, matrix with sizes up to 5x5, the package performing best is CLHEP, while for all larger sizes ROOT wins.

Random Number Generators

Random numbers are used in many HEP applications, like Monte Carlo simulations. Various libraries already ex-

ist which provide a large variety of generators and means to generate numbers according to defined distributions. The current studies focus on evaluating the quality of the existing random number generators existing of the present software packages.

VALIDATION STUDIES OF GSL

The performance and the numerical accuracy of some parts of GNU Scientific Library (GSL) [5] have been tested and validated comparing with other numerical software libraries. Two new tests for randomness have been proposed and applied to the main generators of GSL. We have also done tests for the numerical results and the performance of some special functions and distributions, the numerical integration algorithms and the GSL linear algebra package. The tests and their results are described in this report [12].

Tests of mathematical functions

A comparison of numerical results and timing performance of mathematical functions typically used in HEP applications is done. The special functions being tested are the Bessel functions, Gamma, logarithmic Gamma and incomplete Gamma functions and the Error function. We test in addition some statistical functions, such as the Landau distributions, the Chi2 distribution and its probability. We compare the numerical values obtained using the algorithms from GSL [5], ROOT/TMath [2] or NagC [6]. Varying the input values, we study the relative and absolute difference obtained with respect to the GSL values and compare it with the estimated error from GSL, which is provided by the algorithm. Figure 2 shows for example the results obtained for the cylindrical Bessel function. In addition, we perform timing tests, measuring the amount of CPU time to evaluate the functions for the three different libraries, averaging on the input values. In general, we have observed that the results obtained by GSL are in agreement with those obtained by NagC and often at the level of the expected numerical accuracy. On the other end GSL is found to be the worst in time performances compared to both NagC and ROOT, taking in same cases up to three times more to call the algorithms.

Tests of Random number generators

Two new tests have been designed and applied to random number generators. The first test, Frequency test, fill a d dimensional space with points formed from a sequence of random numbers. The frequency is found in a small volume of the space. This frequency should not be correlated with the frequency in other volumes of the space. The second, Orbit test, arranges a sequence of random numbers into multidimensional points. We look for points that are close to the first point. For these points we calculate the distance between the next point and the second point. There should be no correlations between the two distances. We apply the tests to eight of the most used generator of GSL, including

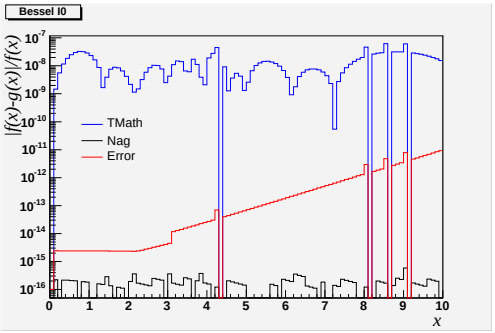

Figure 2: Relative difference from GSL - NagC and GSL - ROOT TMath and estimated error for the modified cylindrical Bessel function of first order, $I_0(x)$

Mersenne Twister and RanLux. All these generators pass the two tests. The only generator failing is the Park and Miller Minimal Standard algorithm gsl_rng_minstd which is known to generate points distributed in hyper-planes and is used to verify the test procedure.

FITTING AND MINIMIZATION

A major project activity is related to fitting and minimization, which is heavily used in physics analysis. Multi parameters fit to large set of data are commonly used to estimate the parameter values and their errors. Minimization algorithms are therefore needed to find the numerical solution to the fitting problems. A very popular minimization package used by high energy physicist is MINUIT [13], which in addition to minimization algorithms, contains methods for analyzing the solutions and estimate the parameter error correlation matrix. These combined capabilities are very difficult to find in other minimizers existing outside HEP. There is therefore a strong requirement to have the MINUIT algorithms available in the new C++ mathematical libraries. In addition to MINUIT, we provide also a generic fitting library, FML, which facilitates the usage of minimizers in standard fitting problems.

C++ MINUIT

MINUIT has been rewritten in C++ providing and enhancing all the functionality already existing in the original Fortran package. The effort has been directed and supervised by the original author of MINUIT. The new package has been redesigned in an object oriented style and implemented using standard C++. It works on all the major platforms and it is independent of any external packages. The profits from basing on an object oriented design are an increased flexibility, easy maintainability in the long term and opening to extensions (new algorithms, new functionality, changes in user interfaces). The current released version of C++ MINUIT provides already all the functionality present in the Fortran version, containing the MIGRAD, SIMPLEX and MINIMIZE minimizers, HESSE, MINOS and CONTOURS for analysing the solutions and

methods to control the input parameters (fix, release and set limits). In addition, the new C++ version has the functionality to set one side parameter limits. The most important goals of MINUIT C++ are to achieve a numerical accuracy equivalent to the Fortran version and to have equivalent computational performances. Extensive minimization and comparison tests have demonstrated that these goals have been reached. There is a small performance penalty only when minimizing functions trivial to compute. For the typical cases of computational expensive functions, where good performances are required, the differences observed in CPU time with respect to the previous version are negligible.

This new package is starting to be used by physics users and it is already being integrated in existing analysis tools and in the reconstruction code for the CMS experiment. We are adding now a new minimization algorithm, FUMILI,inside the package. FUMILI is a specialized algorithms for least squares and likelihood minimization ad it requires different input values. Thanks to the object oriented design, the integration of FUMILI is going to be possible without requiring a large extra effort. The new classes, which need to be implemented, are easily identified and the amount of extra code necessary to support the enhancement is minimal.

FML

Since the majority of cases, where MINUIT is used in the HEP community, is to solve fitting problems, we provide in addition a package for performing fits based on the minimization (maximization) of standard Chi2 or Likelihood functions. The user does not need to write the objective function to be minimized, but needs to supply only the input data points and the function to model the data. The objective function to be minimized is then constructed automatically by the library with particular care on performances. The package supports binned and unbinned fits and provides a set of pre-defined model functions such as Gaussian, Polynomials or Exponentials, which can be added together. FML defines also interfaces for minimization, currently implemented using the MINUIT minimizers. Additional minimizers from external packages, such as GSL or the Nag C library can be added and, through a plug-in manager, loaded at run time. The current release is used to implement the AIDA fitting interfaces for the PI project [14]. The next steps are to provide Python (and ROOT CINT) binding to FML to be able to use it in interactive applications. The package will also be integrated and re-designed to be consistent with the new developments in the C++ function class library.

CONCLUSIONS

A work package, as part of the SEAL LCG project, is formed to provide support and develop mathematical software libraries for the LHC experiments. We have provided a Web site containing an inventory of the required mathematical functions and algorithms with references to the corresponding implementation and documentation. We have produced a design for new C++ mathematical libraries which implement the set of identified functions and algorithms. We are starting now implementing them using the GNU Scientific Library [5]. Extensive validation and test studies have confirmed the numerical quality of GSL and proofed so far, for the selected algorithms, that is suitable for computation required in the HEP community. We have also delivered a new version of the popular minimization package MINUIT in C++ with same functionality, performance and quality as the original version. A user convenient fitting package using MINUIT is also provided. The integration with the LHC experiment software is initiated and the libraries are started to be used by physicists working not only for the LHC community (experiments and LCG software projects). We collaborate not only with the LHC community, but also with the wider HEP community including other laboratories and institutes. We are also cooperating with the ROOT project with the aim to provide a common mathematical software.

ACKNOWLEDGEMENTS

We would like to thank W. Brown, R. Brun, V. Innocente, P. Kunz, E. Myklebust, J. Mosciki, E. Offermann, M. Paterno, T. Todorov for having contributed to the developments or having provided feedback on the mathematical software libraries presented in this paper.

REFERENCES

[1] The LCG SEAL project, http://www.cern.ch/seal

[2] The ROOT project, http://root.cern.ch

[3] The CERN Program Library, http://wwwasdoc.web.cern.ch/wwwasdoc/cernlib.html

[4] CLHEP, http://www.cern.ch/clhep

[5] The GNU Scientific Library, http://www.gnu.org/software/gsl

[6] The Numerical Algorithm Group (Nag) C Library, http://www.nag.co.uk/numeric/cl/CLdescription.asp

[7] MathLib Web site, http://www.cern.ch/mathlib

[8] W. Brown and M. Paterno, A proposal to Add Mathematical Special Functions to the C++ Standard Library, http://std.dkuug.dk/jtc1/sc22/wg21/docs/papers/2003/n1422.html

[9] BLAS, http://www/netlib.org/blas LAPACK, http://www.netlib.org/lapack

[10] uBLAS, http://www.boost.org/libs/numeric/ublas/doc/index.htm

[11] The ROOT Linear Algebra, contrib. 303 to this conference

[12] M. P. Hatlo, Validation studies of software libraries, http://seal.cern.ch/documents/mathlib/thesis_marte.pdf

[13] F. James, MINUIT, CERN Program Library Writeup D506, http://wwwasdoc.web.cern.ch/wwwasdoc/minuit/minmain.html

[14] PI, http:://cern.ch/pi and contrib. 204 to this conference