

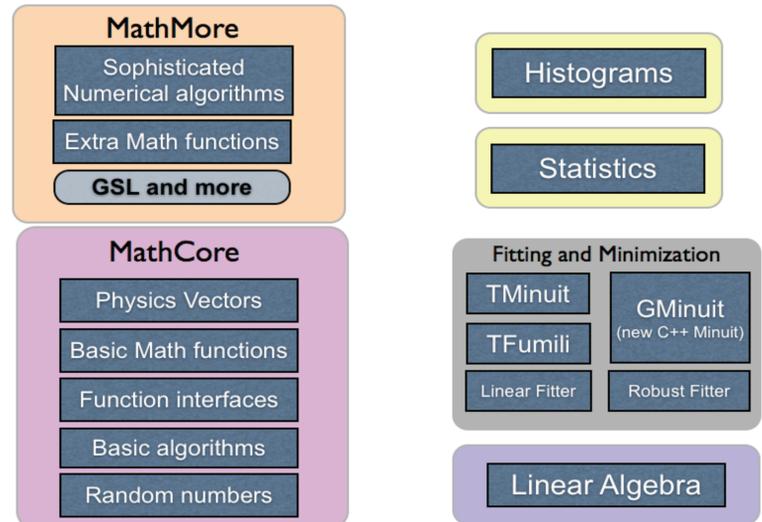


W. Brown¹, M. Fischler¹, L. Moneta², A. Zsenei²
¹ Fermi National Accelerator Laboratory, Batavia, Illinois, USA
² CERN – European Organization for Nuclear Research, Geneva, Switzerland

In the new organization for the Math Libraries in ROOT we have two new libraries: *MathCore* and *MathMore*. *MathCore* contains the basic Math functionality and has no dependency on any external libraries or any other ROOT libraries. In the current release 5.0.4 it contains the physics and geometry vector package plus some basic mathematical functions.

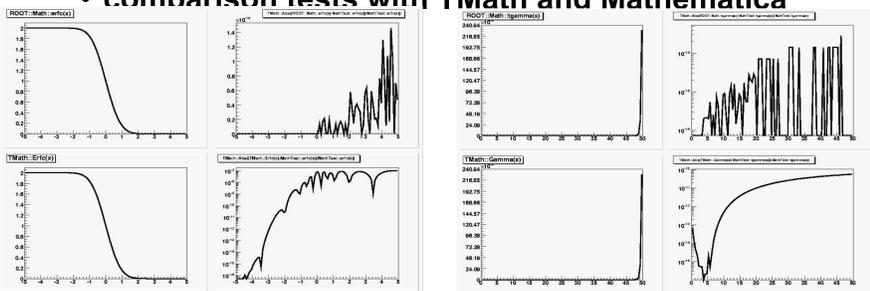
MathMore is a package with some extra functionality typically less used than those in *MathCore*. The current implementation in *MathMore* is based on GSL which is built inside the library to avoid having an external dependency.

Both *MathCore* and *MathMore* can be built and used as external package outside the ROOT framework.



Mathematical Functions in *MathCore* and *MathMore*

- **Special Functions:**
 - use interface proposed to C++ standard:
`double cyl_bessel_i (double nu, double x);`
- **Statistical Functions:**
 - Probability density functions (pdf)
 - Cumulative dist. (lower tail and upper tail)
 - Inverse of cumulative distributions
 - Coherent naming scheme. Example chi2:
`chisquared_pdf`
`chisquared_prob, chisquared_quant,`
`chisquared_prob_inv, chisquare_quant_inv`
- **New functions have better precision**
 - comparison tests with TMath and Mathematica



3D and 4D Vector packages

Generic Vector package properties :

- **Template on the scalar type**
- **Template on the Coordinate System type**
`LorentzVector<PxPyPzE4D < double > >`
`LorentzVector<PtEtaPhiE4D < double > >`
- **Point and Vector distinction in 3D:**
`Displacement3DVector < Cartesian4D <double > >`
`Position3DPoint < Cartesian4D <double > >`
- **3D Rotation classes**
`Rotation3D, AxisAngle, EulerAngles, Quaternion`
`RotationX, RotationY, RotationZ`
- **LorentzRotation**
 - generic LorentzRotation class + Boost classes
- **Example of usage:**

Constructors

```
XYZTVector v0; // create an empty vector (x=y=z=t=0)
XYZTVector v1(1,2,3,4); // create a vector (x=1, y=2, z=3, t=4)
PtEtaPhiEVector v2(1,2,M_PI,5); // create a vector (pt=1, eta=2, phi=PI, E=5)

PtEtaPhiEVector v3(v1); // create from a Cartesian4D LV
HepLorentzVector q(1,2,3,4); // CLHEP Lorentz Vector
XYZTVector v3(q); // create from a CLHEP LV
```

Accessors

```
double x = v1.X() = v1.Px(); // have both x() and px()
double t = v1.T() = v1.E(); // have both t() and e()
double eta = v1.Eta();
XYZTVector w = v1.Vec(); // return vector with spatial components
```

Operations

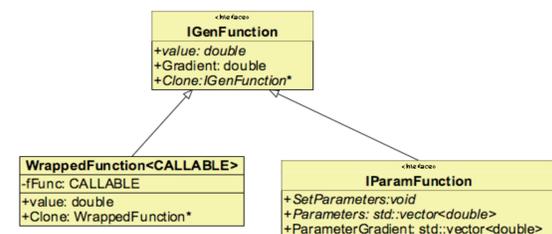
```
v1 += v2; v1 -= v2; // additions and subtractions
v3 = v1 + v2; v4 = v1 - v2;
double a; v1 *= a; v1 /= a; // multipl. and divisions with a scalar
double p = v1.Dot(v2); // prefer Dot (less ambiguous)
v3 = v1.Cross(v2); // Cross product
```

Numerical Algorithms in *MathMore*

- C++ interface to some GSL numerical algorithms
- Currently present are algorithms for 1D functions:
 - **Numerical Derivation**
 - central evaluation based on the 5 points rule and forward/backward with 4 points rule
 - **Adaptive Integration**
 - bracketing and polishing algorithms which use function derivatives finite and infinite intervals
 - **Root Finders**
 - bracketing and polishing algorithms which use function derivatives
 - **Interpolation**
 - type: linear, polynomial and Akima spline
 - **Chebyshev polynomials**
 - for function approximation

Function Interfaces

- Minimal interface used by all numerical algorithm:
 - abstract *IGenFunction* and *IParamFunction* classes
 - template *WrappedFunction* class to wrap any C++ callable object (functors, C free function, etc..)
 - set of pre-defined parametric functions, like Polynomial function
 - have *TF1* implementing these interfaces ?



Example of Usage

- Algorithms can be used with *IGenFunction* objects
 - Callable objects can be wrapped using the *WrappedFunction* class
 - Integration example (with a user free C function)

```
// with a user provided free C function f(x)
double f( double x) { .....}
// create WrappedFunction class
ROOT::Math::WrappedFunction wFunc(f);
// create integrator class
ROOT::Math::Integrator ig(wFunc);
// integrate between a and b
double result = ig.Integral(a, b);
double error = ig.Error();
```