SEAL-ROOT Math Plans for 2005

Math work package

Andras Zsenei, Anna Kreshuk, Lorenzo Moneta, Eddy Offermann

ROOT Core Software Meeting, 18 March, 2005

Math Work package

- **Responsability for this work package:**
  - Basic Mathematical functions
    - TMath and SEAL MathCore
  - Functions and Fitting
    - Parameteric function classes (TF1,...)
    - Minuit, Fumili, Linear and Robust fitters, quadratic prog.
  - Random Numbers
  - Linear Algebra
  - Physics Vector
- **Also, but not considered now:**
  - Histograms
  - Statistics (confidence level)
  - Neural Net, multivariate analysis, etc..

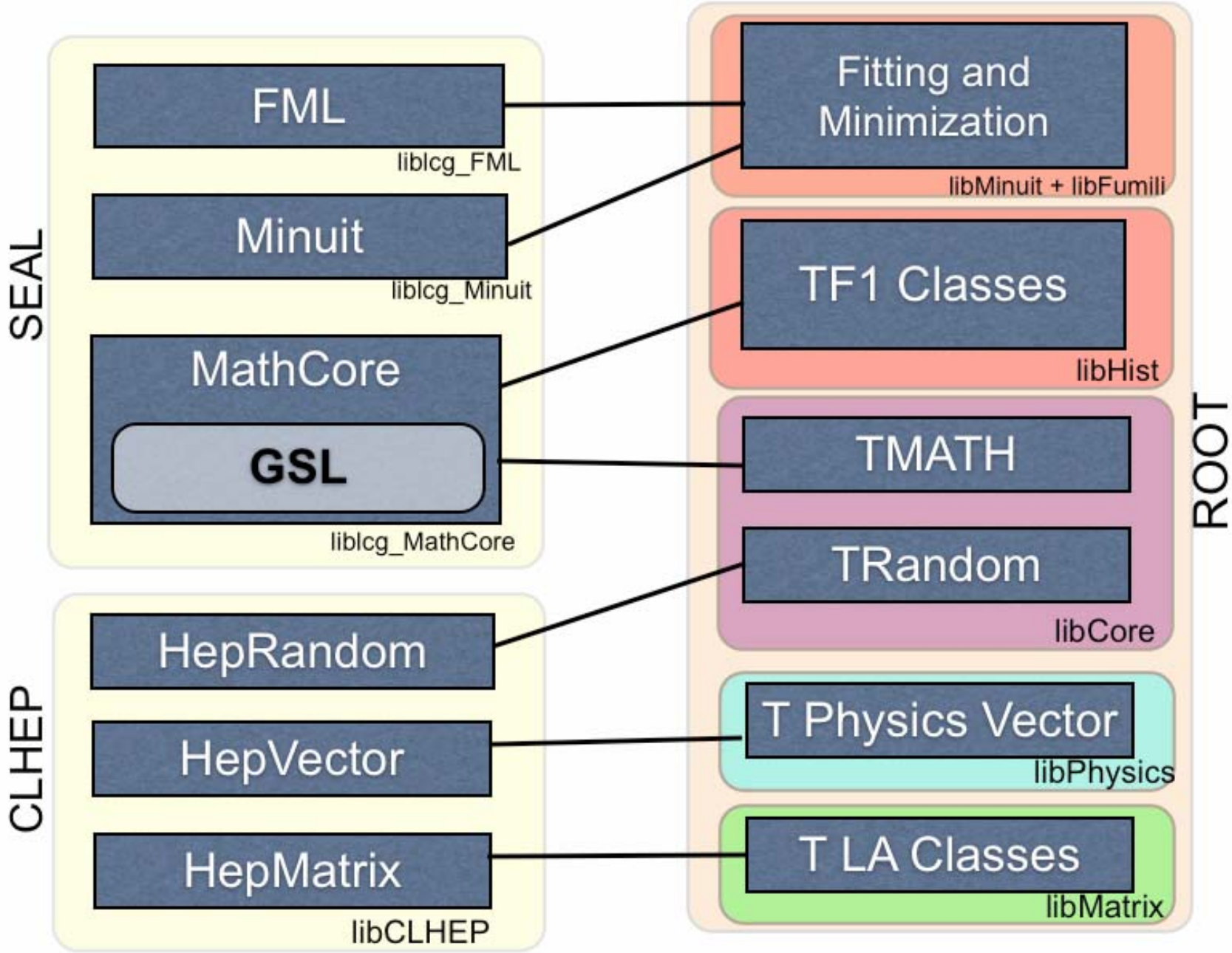
Outline

- Compare the contents of SEAL and ROOT
- Preliminary proposal for SEAL - ROOT integration and evolution for short-medium term
  - MathCore vs TMath
  - Improvements for Function classes (TF1)
    - integration of MathCore numerical algorithms
  - Fitting and Minimization
    - integration of SEAL Minuit and SEAL Fitting framework
- Review CLHEP and compare with ROOT
  - Random numbers
  - Physics Vectors
  - Linear Algebra
- Possible proposal on how to proceed with CLHEP

SEAL Math Lib Contents

- **MathCore**
  - library with the basic Math functionality
  - used GSL in implementation
  - interfaces could be re-implemented using another library
  - design reviewed by CMS
- **Minuit**
  - re-implementation of Minuit in C++
  - stand-alone package (no ext. dependencies)
- **FML (Fitting and Minimization Library)**
  - defines some generic interfaces for fitting and minimization
  - use Minuit

SEAL, CLHEP and ROOT Math Libraries



MathCore

- **MathCore Contents:**
  - (see http://seal.web.cern.ch/seal/MathLibs/MathCore/html/index.html)
    - **Mathematical functions (implemented as free function in *mathlib* namespace) :**
      - special functions like Bessel, Beta, Gamma, ... (~ 20 func.)
      - statistics functions: pdf, cumulative probability distributions and their inverse (~ 50 func.)
    - **numerical algorithms implemented using GSL**
      - integration (6 alg.)
      - differentiation (3 alg.)
      - root finders (6 alg.)
      - 1D minimization (2 alg.)
      - interpolation (4 alg.)
    - **Generic function classes and interfaces**
      - Generic function interface
      - Parametric function interface
      - Concrete classes (Polynomial function, ..)

TMath Contents

- **TMath namespace:**
  - Numerical constants (Pi, e, h, etc...)
  - Trigonometric functions
  - Elementary functions
  - Other basic functions (abs, min, max, range, sign)
  - Min and max of arrays
  - Statistics: mean/rms of sequences (arrays)
  - algorithm (binary search, hashing, sorting)
  - vector operations (cross, normalize)
  - Special functions (Bessel, Erf, Beta, Gamma, etc...)
  - Statistical functions (Poisson, Prob, Student, F dist, Gauss, BreitWigner, Landau etc....)
  - Kolmogorov probability

Differences TMath-MathCore

- **Special Functions:**
  - Both have most used ones : Beta, Gamma, Erf, ...
  - MathCore has more complete set functions :
    - all Bessel types and for any order nu
    - Legendre polynomials, elliptic integrals, hypergeometric, exponential integral and Riemann
- **Statistical Functions:**
  - Both have most used functions
    - normal, Chi2, Binomial, Poisson, Gamma, Cauchy, t, F dist.)
  - MathCore has for each one pdf, cdf and their inverse
  - TMath has in addition
    - Incomplete Beta, Kolmogorov prob., Struve and Voigt functions

Proposal for TMath

- **Changes to TMath in the short term:**
  - **Have union of all functions TMath - MathCore**
  - **Separate mathematical function in a different namespace, CVS repository and library**
    - **Have a Math library which can be built independently**
    - **for ROOT distributions, we might include still in the ROOT libCore**
    - **Math functions need to be in the libCore ?**
      - **some math functions (i.e. *Prob*) are used only by Hist and Physics libraries,**
      - **others (i.e. *Bessel*) are not used at all within ROOT**
  - **Change interface to the one proposed to the C++ standard (used in SEAL MathCore)**
    - **Define consistent names for statistical functions**
- **Keep current TMath for backward compatibility**

Example for Special Functions

- **Have a separate namespace:**
  - **easy transition in the future when they will be in *std* namespace**
- **Example of new interfaces and how to keep backward compatibility**

```
namespace specfunc {

  double cyl_bessel_i(double nu, double x);

  double cyl_bessel_k(double nu, double x);

...

}


namespace TMath {

  double BesselI(double x, int n) {
    return cyl_bessel_i(static_cast<double>(n), x);
  }

  double BesselK(double x, int n) {
    return cyl_bessel_k(static_cast<double>(n), x);
  }
.....
```

Function Implementation

- **Which implementation to use ?**
  - **MathCore is a wrapper to GSL function**

```
double cyl_bessel_i(double nu, double x) {

  return gsl_sf_bessel_Inu(nu, x);

}
```

  - **GSL is good but we need to distribute with GPL license**
  - **We should consider also alternative implementations**
  - ***cephes*** **(available at netlib: http://www.netlib.org/cephes/)**
    - **C library with single, long and long double implementations for special functions**
    - **open source free license**

Additional TMath Changes

- **Longer term modifications :**
  - **use STL for sorting algorithm, min/max etc.**
    - **evaluate the performances,**
    - **and in case drop old ones**
  - **use in addition std::vector in interface instead of C arrays**
    - **performances are the same**
  - **Separate also statistic functions acting on containers:**
    - **mean, RMS, median, skewness**
    - **have template functions for all of them**

GSL

- What to do with GSL ?
  - Good library but license problem
    - GPL requires us to distribute with GPL license
    - no problem for our academic use, but problem comes if the software is then used in commercial applications
- Have a C++ Wrapper (like in MathCore) hiding the implementation
- A possible solution could be to have alternative implementations
  - many already exist (i.e. *cephes*), it should not be so much more work
  - have a compilation flag to build with or without GSL ?
  - HEP users build with GSL, while others use alternative ?
    - it would be difficult to manage for the core libs
- To avoid external dependency we could distributed what we use of GSL together with our Software

Function Classes in ROOT

- TF1, TF2, TF3, TF12
  - parametric 1, 2 and 3D functions and projection
    - various mathematical functionality (Eval, Derivative, Integral, getMin/Max, getX, etc..)
    - Fitting functionality (GetChi2(), Fix/Release Parameters)
    - Generate random numbers
    - Plotting functionality (Paint())
- TFormula
  - to evaluate simple expression and pre-defined functions (Gauss, Poly, Expo, etc...)

Proposal for Functions

- **Short term:**
  - separate plotting functionality
    - make independent of plotting (use VirtualHistPainter)
- **Medium Term:**
  - Separate implementation of the numerical algorithms from the function
    - have integration, derivation in separate classes
      - interfaces with different algorithm implementations
    - combine with what is provided in MathCore based on GSL
  - Redesign Mathematical function classes
    - some work started in MathCore
    - defines new function classes and interfaces:
      - GeneralFunction , ParametrizedFunction, Pdf
      - support arithmetic operations, composition, convolution
  - improve TFormula (longer term)

Fitting and Minimization

- Fitting in ROOT goes through the TVirtualFitter
  - abstract class designed for Minuit
    - mixes fitting and minimization
  - Have implementations based on TFitter (TMinuit), TLinearFitter and TFumili
    - lots of repetition in the code (for example in fixing/setting parameters)
  - Missing a real framework for Fitting and Minimization
    - people implemented additional packages (i.e. RooFit)
- In SEAL MathLibs we have a fitting framework
  - FML complemented with new Minuit
  - we should start from this and combine with the proposed re-design of the Function classes

SEAL Fitting packages

- MINUIT
  - re-implementation of Minuit in C++
  - added new functionality
    - Fumili, single side bounds
  - stand-alone package (no external dependency)
- FML (Fitting and Minimization Library)
  - provides general way of fitting data using various fitting methods and minimization engines
  - defines generic interfaces for fitting and minimization
    - have minimizer implementations based on new Minuit
    - for testing we have implementations based on F77 Minuit and NagC
  - support for standard fitting methods (Chi2, M.L., etc..)
  - very efficient in term of performances

Proposal for Fitting

- **Short term :**
  - have Fitter class using SEAL Minuit implementing the TVirtualFitter interface
  - basically already done in the past by Matthias Winkler
  - make it default engine ? (need more evaluation)
- **Medium/Long term:**
  - integrate FML in ROOT, redesigning and adapting to the new Function classes
  - have clear separation for Fitting and Minimization
    - separate interfaces as in SEAL FML
  - make best solution to integrate all existing implementations:
    - Linear and Robust Fitters, Fumili, Minuit and quadratic optimizers

Fitting and Functions design ideas

- Have various Fitting classes:
  - Fitter, FitData, FitParameters, FitResult, etc..
    - fitter gets the input data
      - option to copy in (often more efficient) or able to iterate on original container
    - builds from ModelFunction an objective function according to method:
      - chi2, binned and unbinned maximum likelihood, etc..
    - Use facilities from Function classes
      - combining functions with addition, convolution etc..
      - use pre-defined pdf (like in RooFit)
    - have class containing all fit result (parameters, error matrix, chi2, ndf)
  - Minimization (optimization) classes:
    - find minimum of a multidim. function:
      - Minuit, quadratic programming

CLHEP

- **CLHEP packages:**
  - **Random (and Random Objects)**
  - **Matrix**
  - **Vector**
  - **Geometry**
  - **HepPDT and HepMC (keep separate)**
  - **Evaluator (keep separate, not needed)**
  - **Generic Functions**
  - **Units**

CLHEP Exp. Feedback

- **Feedback received in 2003 from CLHEP workshop**
  - (see http://proj-clhep.web.cern.ch/proj-clhep/Workshop-2003/CLHEP_LHCfeedback.pdf)
  - **CLHEP scope is not clear defined**
    - seems more an heterogeneous collection
  - **Random :**
    - missing the saving of generator seeds
    - statics variable (engines) give problems on Windows
  - **Matrix**
    - poor performances for symmetric matrices (not clear which operations)
    - some numerical instabilities
    - dependent on Random
    - missing constructor taking Hep3Vector

CLHEP Exp. Feedback (2)

- Vector (Physics Vector)
  - most used package
  - Comments received:
    - too bloated interface
    - desire to have it template on scalar type
    - not seed for public setter methods
    - confusion between classes in Vector and Geometry packages
  - Alternative proposal is a package used by CMS tracking and muon
    - 3D, 2D Vector classes Points classes
    - separation points-vector as distinct types

Random (CLHEP vs ROOT)

- CLHEP:
  - Nice separation engine - distributions
    - abstract class for Engine and various implementations
    - singleton class, HepRandom for default engine (HepJames)
    - classes for each distribution (RandFlat, RandGauss, etc..)
- ROOT
  - TRandom base class with default engine
    - rndm() from Cernlib
      - fast generator but with small period (10**8) and obsolete in Cernlib
    - base class defines functionality for random distributions
    - possibility to store in a file (TRandom.Write())
  - TRandom2 (based on rdm2()) and TRandom3 (based on Mersenne Twister (623 dim.)
    - both inherit from TRandom

CLHEP-ROOT Random Engines

Random Number generators	CLHEP	ROOT
Rand	**x**	
Drand48	**x**	
DualRand	**x**	
Hard160 and Hard288	**x**	
HepJamesRandom (RANMAR)	**x**	
Mersenne Twister	**x**	**x**
RanLux and RanLux64	**x**	
Ranshi	**x**	
TripleRand (DualRand + Hard288)	**x**	
rndm		**x**
rndm2		**x**

CLHEP-ROOT Random Distributions

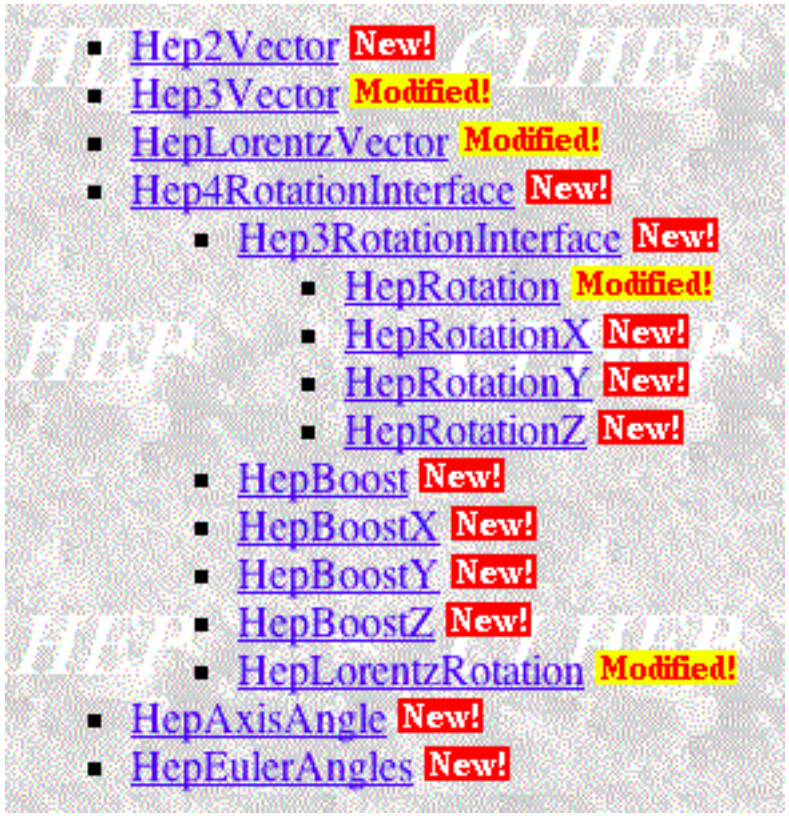
Random Distributions	CLHEP	ROOT
Flat	**x**	**x**
Exponential	**x**	**x**
Gaussian	**x**	**x**
Breit-Wigner	**x**	**x**
ChiSquare	**x**	
Gamma	**x**	
Landau	**x**	**x**
Poisson	**x**	**x**
Binomial	**x**	**x**
Student t	**x**	
Sphere		**x**
Histogram	**from array of numbers**	**from TH1**
General Function (*)	**from array of numbers**	**from TF1 (**)**

(*) Only for getting random numbers in a limited range

(**) possible to use all TMath functions

Physics Vectors

- CLHEP Vector package
  - not clear hierarchy for Vector, Rotation and Boost classes
    - multitude of constructors and member functions
- ROOT Physics Vectors
  - flat hierarchy:
    - TVector2
    - TVector3
    - TLorentzVector
    - TRotation
    - TLorentzRotation



CLHEP vs ROOT (Physics Vectors)

- Hep3Vector vs TVector3
  - Hep3Vector has more methods (too much repetition)
    - more setters, getX() and X()
    - define nearness and parallelism (need to be in base class ?)
    - can be a boost
  - TVector3
    - has Pt(), Px(), ...
- HepLorentzVector vs TLorentzVector
  - HepLorentzVector has more methods:
    - boost functionality, nearness and parallelism
    - ordering
    - restMass(), euclideanNorm()
  - TLorentzVector
    - couple of more setters

CLHEP vs ROOT (Physics Vectors)

- HepRotation vs TRotation
  - HepRotation
    - different classes for X,Y,Z rotations
    - additional copy constructors and transform()
    - awkward methods to get columns and rows
      - colX(), colY(),... and col1(), col2()
    - define nearness and ordering
- HepLorentzRotation vs TLorentzRotation
  - HepLorentzRotation
    - same as for HepRotation
    - decomposition as Boost + 3D Rotation

Linear Algebra

- CLHEP classes:
  - HepGenMatrix
    - HepMatrix
    - HepSymMatrix
    - HepDiagMatrix (should not derive from SymMatrix ?)
    - HepVector
- ROOT LA classes:
  - TMatrixDBase (TMatrixFBase)
    - TMatrixD (TMatrixF)
    - TMatrixDSym (TMatrixFSym)
    - TMatrixDSparse
  - TVectorD (TVectorF)
  - many utility classes and more specialized matrix classes
  - Decomposition classes
    - LU, QR, SVD, Choleski

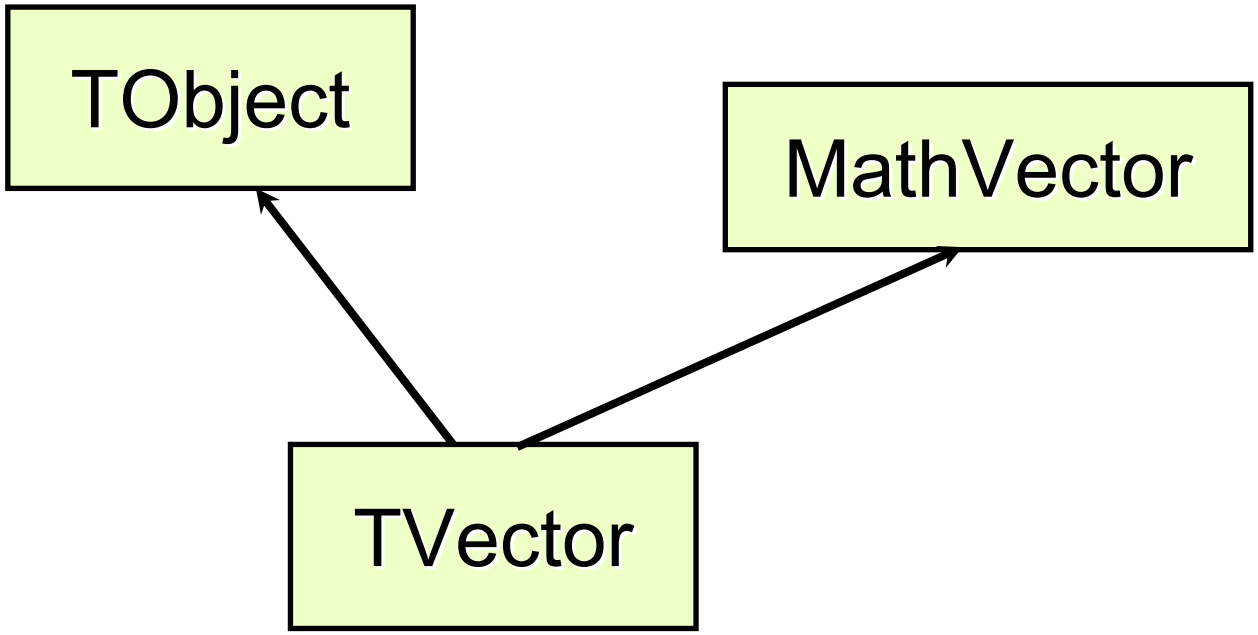
Linear Algebra (2)

- More functionality in ROOT
  - Decomposition for solving LA systems
  - Support for Sparse matrix
  - concept of LazyMatrix for minimizing copying
- Both CLHEP optimized support for small matrices
  - pre-allocation on the stack up to 6x6 matrix
  - optimized inversion algorithm for small matrix (except 2x2)
- Comments on ROOT :
  - No Decomposition for TMatrixF
  - missing support for complex matrices
  - duplication TMatrixF - TMatrixD
    - move to templates ?

CLHEP Proposal

- ROOT basically contains all CLHEP functionality and in some case even more
- Implementations are in same case identical
- Non sense for long term maintenance having the two separate packages
- Move in the direction of a single new library
- Expect that experiments will agree to use new library
  - If new classes do not have dependency on other ROOT library
  - need to remove TObject inheritance
    - several possible technical solutions

Possible Solution



- **Method signature in the new Math classes is ROOT-like or CLHEP-like ?**
  - **it is more a political choice**
- **If are CLHEP-like, implementation in T-classes is inlined using methods of the base class**

Proposal for Random

- **Short Term:**
  - make TRandom3 the default engine and rename it
  - remove TRandom2 ?
  - add some other engine from CLHEP (RanLux, HepJames)
  - add other distributions (taking from CLHEP and/or GSL):
    - Gamma, Chi2, LogNormal, F-dist, t-dist, geometric, etc..
- **medium/long term:**
  - Evaluate random number proposal to C++ standard
    - template classes on Engines and Distribution Type

```
template<class Engine, class Distribution>
class variate_generator {
    result_type operator() () ;      // for generating random numbers
    };
```

    - a similar implementation already exists in Boost
  - re-implement from CLHEP using the new interface ?
    - wrapper in ROOT based on that library ?

Proposal for Vector and LA

- **Remove from the ROOT Physics and Linear Algebra classes the TObject inheritance**
  - have an independent Physics and Matrix library
- **Physics Vector**
  - add missing functionality present in CLHEP
    - make nearness concept in a separte class ?
    - move to a new cleaner interface ?
      - problem of preserving backward compatibility
- **Linear Algebra**
  - based in the short term on ROOT implementation
  - make more detailed evaluation studies with other Linear Algebra packages
  - consider move to template classes for adding support for complex matrices

Proposed new Math Structure