Proposal for upgrading the optimization routines in ROOT: adding parameter constraints

Eddy Offermann

February 7, 2004

1 Constrained Optimization

Through MINUIT, ROOT has a powerful non-linear optimizer which can handle simple boundary-constraints on the individual parameter. In principle, one could, through the use of *slack* parameters and *Lagrangian* multipliers, extend this to more general boundary-constraints (including inequalities). However, for linear (and quadratic) optimization problems, better approaches are available. Below, we discuss the different cases.

Time estimate: 5 months

1.1 Linear Problem

In this case we have to solve for source vector $\mathbf{x}$ the equation

$$\mathcal{A}\mathbf{x} - \mathbf{b} \;=\; 0 \quad \text{, where} \quad \begin{array}{ll} \mathbf{x} & (n)\text{- source vector,} \\ \mathcal{A} & (m \times n) \text{ matrix with Rank } (\mathcal{A}) \;=\; k, \\ \mathbf{b} & (m)\text{- target vector} \end{array}$$

If $k = \text{Rank}(\mathcal{A}) \neq m$, we will not find the solution because it is over determined and we replace the equation by

$$r^2 = (\mathcal{A}\mathbf{x} - \mathbf{b})^2 \;=\; \text{minimum}$$

If $k < \min(m, n)$, the system is under-determined and no unique solution can be found.

In the table below, all cases are listed. In the next section we will discuss the changes/additions to the linear algebra package of ROOT in order to move smoothly through the table.

	Rank $(\mathcal{A})$	residual	uniqe solution
$m = n$	$k = n$	$r = 0$	yes
	$k < n$	$r \geq 0$	no
$m > n$	$k = n$	$r \geq 0$	yes
	$k < n$	$r \geq 0$	no
$m < n$	$k = m$	$r = 0$	yes
	$k < m$	$r \geq 0$	no

Things become more challenging when parameter constraints are applied on $\mathbf{x}$:

$$\mathcal{E}\mathbf{x} \le \mathbf{d} \quad \text{, where} \quad \begin{array}{ll} \mathcal{E} & (l \times n) \text{ matrix,} \\ \mathbf{b} & (l)\text{- target vector} \end{array}$$

We will (of course) only consider constraint matrices $\mathcal{E}$ for which

$$\text{Rank}(\mathcal{E}) = l < n$$

In case of an equality constraint, solutions can be found by performing SVD on both $\mathcal{A}$ and $\mathcal{E}$, see Lawson and Hanson [1] and a clear algorithmic implementation by Siegmund Brandt [2] . However, in case of an inequality constraint the new concept of *Linear Programming* has to be introduced in ROOT.

The web page by Harvey Greenberg [3] is a good starting point to investigate software availibility. You will find there a link to a software survey by Rober Fourer [4] . Unfortunately, most packages are commercial, even those originating from academics .

I found two packages in the public domain :

GLPK This is GNU's implementation of linear programming. It is written in C. Unfortunately, it does not use the GSL functionality. For instance, it has its own implementation of the necessary linear algebra. Also no extensions are availble for quadratic programming .

OOQP It originates from Argonne National Laboratory and is written in C++ . The package has all the necessary tools to solve linear and quadratic convex problems of the following type:

$$\text{minimize } (\frac{1}{2}\mathbf{x}^T\mathcal{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x}) \quad \text{, subject to} \quad \begin{array}{lcl} \mathcal{A}\mathbf{x} & = & \mathbf{b} \\ \mathcal{C}\mathbf{x} & \ge & \mathbf{d} \end{array}$$

My plan is to start with the OOQP package and replace the internals like vector and matrix definitions and operators by the ROOT equivalents. In fact, this requires some upgrades to the ROOT package, in particular sparse matrix support will become necessary. This will be discussed in the next section.

1.2 non-Linear Case

Here, I will be brief. My plan is to extend the *existing* MINUIT implementation in ROOT. For equalty constraints, using Lagrangian multipliers, terms will have to be added to the Hessian matrix and gradient vector. For completeness, I have spelled out in detail the exact formulas in appendix A .

The user interface will probably be the same as for the objective function : a user-supplied function. A more user-friendly approach with for instance `TFormula` will need some additional work because this class supports only expression up to four independent parameters.

The inequality constraints will be handled by introducing *slack* parameters. For instance the x constraint :

$$f(x) \leq 0$$

can be reformulated by introducing slack parameter y as

$$f(x) - y = 0 \;\; ; \;\; y \geq 0$$

where the first part is just an equality constraint and $y \geq 0$ can be handled by MINUIT.

2 Extension of the Matrix/Vector Class

Currently, a matrix/vector package is installed in ROOT that was derived from an earlier version of the linear algebra package by Oleg Kiselyov, see [12] . The package is fairly complete and contains most implementations of level 1 and 2 BLAS (element-wise operations and various multiplications), and higher-level operations like matrix inverse and eigen-value analysis. But of course, still some are missing, in particular functionality to solve a set of linear equations. Therefore, this is a good opportunity to have another look at other C++ linear-algebra packages, either replacing the current `TMatrix/TVector` classes or taking parts to enhance the current set.

The issue of replacing or upgrading might not seem relevant because nothing prohibits the ROOT user of linking to an outside Linear Algebra package. However, without careful incorporation it into the ROOT environment, objects created with the additional classes can not use the powerful I/O and the C++ interpreter CINT.

Time estimate: 2 months

2.1 Overview

The Object-Oriented Numerics Page [7] and Jack Dongarra's list [8] give a current overview of linear-algebra packages in C++ . I will only consider packages that besides being "free" and the ability to handle dense matrices, fulfill the following criteria:

- no dependence on other external (wrt ROOT) libraries, in particular packages that are just a wrapper around FORTRAN or C implementations of LAPACK. It is interesting to see that efforts like LAPACK++ have been stopped and is succeeded by a complete C++-native package, called TNT.
- not only matrix/vector storage and element-wise operations should be available but also higher-level functionality.

That leaves more or less the following packages: GSL [9], TNT [10], CLHEP [11] and LINALG [12]. Here some observations on each of the packages and a list of features that is unique to them and should be implemented in the ROOT matrix package.

GSL The linear-algebra part of the GNU scientific library is very complete. In particular full functionality is available for complex numbers. However, the number of methods is overwhelming and not encapsulated, leaving it up to the user to make the right choices. For example, why would you want a right-side Householder transformation publicly accessible. One could spend the effort of writing a new C++ interface, however, giving the fact that several of the other packages cover the same ground in the real-dense-matrix arena and, even more to the point, all of them seem to rehash the same LAPACK [14] algorithms, it is more efficient to provide a few algorithm additions to the other packages.

TNT This toolkit provides an integrated collection of generic matrix/vector classes based on components of STL but is not as complete (yet) as the other contenders. The project seemed to be abandoned but recently Roldan Pozo added some more functionality by using some code of the JAVA- project JAMA [13] project. Another issue might be the fact that it is heavily templated and introducing templated classes into ROOT would be a first (and who likes to be first). Despite its limited matrix-tool palette, it is the only package that tackles the eigenvalue problem for a non symmetric matrix and anticipates sparse matrix support.

CLHEP The look and feel of these classes is quite similar to those in LINALG, in particular have a look at the matrix-view classes and the ability to create functors that define operations to every matrix element. I have only two complains:

- The authors choose to have matrices and function input parameters start at (1,1) instead of the more C-like (0,0) .
- No functionality is available for eigenvalue analysis.

On the other hand there are also interesting ideas in this package that should be available in ROOT:

- Create a general matrix class from which all different matrix classes inherit, see Fig. 1. (As the pictures shows, I dropped the CLHEP diagonal class and instead have one for sparse matrices.) This scheme has several advantages. First of all, speed can be increased because many matrix operations can take advantage of the layout classification. Secondly, many matrix operations make only sense on certain matrices. In the current `TMatrix` class of ROOT both a general invert operation exist and a specialized one for positive definite matrices; this can now be separated.
- Another speedup is accomplished by avoiding memory allocation on the heap for small matrices ($<$ 25 elements in CLHEP) through a data[25] member of the matrix class.

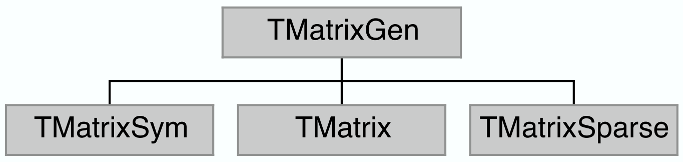

Figure 1: Inheritance scheme for matrix classes

LINALG This package has by far the best C++ feel to it (if abstraction is a measure for that) . It has more or less the same functionality as CLHEP, meaning that also here the eigenvalue analysis is missing and indexing starts at 1 . It misses a few matrix decompositions like QR, LU and Cholesky. LINALG has paid special attention to the hidden calls of constructors and deconstructors through the consistent use of the *lazy-matrix* concept. The lazy-matrix is a recipe describing the matrix properties, it can be passed on without instantiating the complete matrix.

A few years ago, LINALG was inserted into the ROOT package. Issues like indexing, too abstracted matrix-element access, lack of eigenvalue routines, missing matrix decompositions were alleviated. However, the author has added in the mean time some interesting functionality like an SVD class and linear equation solving. As discussed next, this should be added to the ROOT implementation.

Before discussing further, let us remind ourselves that **none** of the above packages seems to have a clear path/concept how to support issues like sparse matrices (ok maybe TNT), distributed computation of subspaces of matrices etc. There seems to be no public domain C++ package that covers all ground. On the other hand, do we really need such a package for ROOT ? Surely, if one needs very specialized matrix algorithms, one can link the appropriate library . For these applications, interactive access through CINT, is most likely not a high priority.

By now the reader will not be surprised that I favor keeping LINALG as the ROOT matrix/vector package. I see no point in exchanging one C++ interface to the original LINPACK and EISPACK routines for another ! However, I do want to make a series of extensions/modifications to the current package:

1. **Float_t:** Remove the `Float_t` version of the matrix and vector classes: `TMatrix`, `TVector`. I see no need to keep these around while all internal calculations are anyhow performed in double precision. Of course, constructors will be added to `TMatrixD` to support `Float_t` input.

```
TMatrixD(Int_t nrows,Int_t ncols,const Float_t* elements,Option_t* option);
TMatrixD(Int_t row_lwb,Int_t row_upb,Int_t col_lwb,Int_t col_upb,
         const Float_t* elements,Option_t* option);
```

2. **const:** The functionality of the matrix view classes `TMatrixDRow`, `TMatrixDColumn`, `TMatrixDDiag` is fairly complete , however `const`-ness is not always correctly enforced, see for instance the following (currently) allowed operation:

```
const TMatrixD a;
TMatrixDDiag(a) = 1;
```

Just like LINALG and CLHEP we will introduce their `const` counterparts: `TMatrixDRow_const`, `TMatrixDColumn_const`, `TMatrixDDiag_const`.

3. **sub matrix:** Add the procedures to get and set a part of a matrix:

```
TMatrixD GetSub(Int_t min_row,Int_t max_row,Int_t min_col,Int_t max_col) const;
void     SetSub(Int_t row,Int_t col,const TMatrixD &msub);
```

where `GetSub` returns a sub matrix and `SetSub` inserts the sub matrix `msub` starting at `(row,col)`.

4. **decomp:** Along the lines of MATCLASS [16] and LINALG, it seems an intelligent idea to separate decomposition from the matrix definitions through decomposition classes,

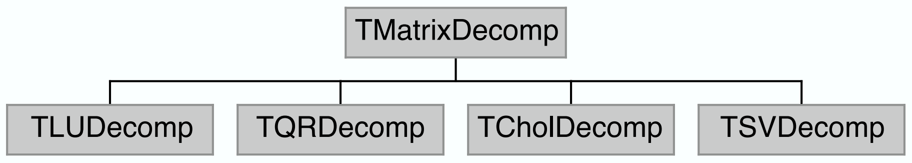

Figure 2: Inheritance for matrix decomposition classes

This setup has several advantages:

- Hand in hand with decomposition goes solving of linear equations, matrix inversion, matrix condition etc. . Each decomposition method has its own unique algorithms to arrive at the answers, moreover some bagage has to be carried along, which one would not like to store in the matrix classes. For instance, it is not as efficient and precise to explicitly calculate the inverted matrix followed by a vector multiplication but better to solve directly with the decomposed matrix pieces. This necessitates keeping decomposed pieces around.
- This separation gives a good opportunity to guide the user to the correct/optimal separation technique. For instance,

```
TSVDecomp(const TMatrixGen &m);
```

  will be available indicating that it is suitable for all matrix types, however, for the CHOLESKY decomposition only one type is allowed:

```
TCholDecomp(const TMatrixSym &m);
```

Many of the pieces are already available in ROOT, the other packages can fill in the blanks with their different LAPACK implementations.

A Using Lagrangian Multipliers in Equality-Constrained Minimization

A.1 No Constraints

Suppose the objective function χ^2 has to be minimized for the parameters a_i, $i = 1, n$.

A χ^2-extreme is given by

$$\frac{\partial \chi^2}{\partial a_i} = 0 \quad i = 1, n \tag{1}$$

Expanding this derivative around the parameter point $\mathbf{a}_0$

$$\left.\frac{\partial \chi^2}{\partial a_i}\right|_0 + \sum_{k=1}^{n} \left.\frac{\partial^2 \chi^2}{\partial a_k \partial a_i}\right|_0 \Delta a_k = 0 \quad i = 1, n \tag{2}$$

With this equation the new parameters can be calculated where the χ^2 is smaller than at the previous point. How close this point is to an extreme depends on how good the approximation of neglecting higher-order terms in the Taylor expansion of Eq. (1) was .

A.2 With Constraints

Suppose we add the following constraints to the parameters:

$$g_j(a_i) = c_j \quad j = 1, m \tag{3}$$

As a consequence the n parameters can not be moved anymore independent through the coordinate space. To be more specific, $n - m$ parameters can be chosen freely while the remaining m parameters are determined by the boundary conditions given by Eq. (3). Therefore, an extreme can not be determined anymore by solving Eq. (1). The minimization problem can, however, be rephrased and written in a form like Eq. (1) by realizing that if the χ^2 has an extreme, taking into account the constraints of Eq. (3) the following condition is fulfilled:

$$\frac{\partial \chi^2}{\partial a_i} + \sum_{j=1}^{m} \lambda_j \frac{\partial g_j}{\partial a_i} = 0 \quad i = 1, n \tag{4}$$

This means that the function $\chi^2 + \sum_{j=1}^{m} \lambda_j g_j$ is actually minimized. The constants λ_j are called *Lagrangian Multipliers*. There are now a total of $n + m$ parameters and $n + m$ equations. By Taylor expanding Eq. (4) around $(a_i)_0$ one gets :

$$\begin{aligned} &\left.\frac{\partial \chi^2}{\partial a_i}\right|_0 + \sum_{k=1}^{n} \left.\frac{\partial^2 \chi^2}{\partial a_k \partial a_i}\right|_0 \Delta a_k + \\ &\quad + \sum_{j=1}^{m} \lambda_j \left(\left.\frac{\partial g_j}{\partial a_i}\right|_0 + \sum_{i=1}^{n} \left.\frac{\partial^2 g_j}{\partial a_k \partial a_i}\right|_0 \Delta a_k \right) = 0 \quad i = 1, n \end{aligned} \tag{5}$$

Assuming that the step size of the parameters is not too large the boundary condition can also be expanded up to first order around $(a_i)_0$:

$$g_j|_0 + \sum_{k=1}^{n} \left.\frac{\partial g_j}{\partial a_k}\right|_0 \Delta a_k = c_j \quad j = 1, m \tag{6}$$

Equations (5) and (6) can be rewritten in the following form:

$$\begin{aligned}\sum_{k=1}^{n}\alpha_{ik}\Delta a_k+\sum_{k=1}^{m}\gamma_{ik}\lambda_k &= \beta_i \quad i=1,n\\ \sum_{k=1}^{n}\delta_{jk}\Delta a_k &= \epsilon_j \quad j=1,m\end{aligned}\tag{7}$$

So both new estimates for the parameters $a_i, i=1,n$ and $\lambda_j, j=1,m$ can be calculated by inverting a matrix, similar to the unconstrained problem. This is even more clear by writing Eq. (7)) in matrix notation:

$$\left(\begin{array}{ccc|ccc} \alpha_{11} & \dots & \alpha_{1n} & \gamma_{11} & \dots & \gamma_{1m}\\ \vdots & & \vdots & \vdots & & \vdots\\ \alpha_{n1} & \dots & \alpha_{nn} & \gamma_{n1} & \dots & \gamma_{nm}\\ \hline \delta_{11} & \dots & \delta_{1n} & 0 & \dots & 0\\ \vdots & & \vdots & \vdots & & \vdots\\ \delta_{m1} & \dots & \delta_{mn} & 0 & \dots & 0\end{array}\right)\left(\begin{array}{c}\Delta a_1\\ \vdots\\ \Delta a_n\\ \hline \lambda_1\\ \vdots\\ \lambda_m\end{array}\right)=\left(\begin{array}{c}\beta_1\\ \vdots\\ \beta_n\\ \hline \epsilon_1\\ \vdots\\ \epsilon_m\end{array}\right)\tag{8}$$

Assuming N data points (y_{exp},σ) and fit function y_{fit} are used in the calculation of χ^2, then the expressions for the parameters in Eq. (8) are:

$$\begin{aligned}\alpha_{ik} &= \frac{1}{2}\frac{\partial^2\chi^2}{\partial a_k\partial a_i}\bigg|_0 = \sum_{l=1}^{N}\frac{1}{\sigma_l^2}\left(\frac{\partial y_{fit}}{\partial a_k}\bigg|_0\frac{\partial y_{fit}}{\partial a_i}\bigg|_0\right)\\ \beta_i &= -\frac{1}{2}\frac{\partial\chi^2}{\partial a_i}\bigg|_0 = \sum_{l=1}^{N}\frac{1}{\sigma_l^2}\left(\frac{\partial y_{fit}}{\partial a_i}\bigg|_0(y_{\text{exp}}-y_{\text{fit}}|_0)_l\right)\\ \gamma_{ik} &= \frac{1}{2}\left(\frac{\partial g_k}{\partial a_i}\bigg|_0+\sum_{j=1}^{n}\frac{\partial^2 g_k}{\partial a_j\partial a_i}\bigg|_0\right)\\ \delta_{ik} &= \frac{1}{2}\frac{\partial g_i}{\partial a_k}\bigg|_0\\ \epsilon_i &= \frac{1}{2}(c_i-g_i|_0)\end{aligned}\tag{9}$$

Notice that if one neglects the second derivative in γ_{ik}, one gets :

$$\gamma_{ki}=\delta_{ik}\tag{10}$$

The covariance matrix of the parameters $a_i, i=1,n$ is given by the $(n\times n)$ left upper part of the matrix obtained by inverting the matrix in Eq. (7) .

References

[1] C.L Lawson and R.J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs NJ 1974 .

[2] Siegmund Brandt, *Data Analysis: Statistical and Computational Methods for Scientists and Engineers*, Springer Verlag 1998 .

[3] Harvey Greenberg,*http://carbon.cudenver.edu/ hgreenbe/courseware/LPshort/LPsites.html* .

[4] Robert Fourer,*http://lionhrtpub.com/orms/surveys/LP/LP-survey.html* .

[5] GNU *Linear Programming Kit*, http://www.gnu.org/software/glpk/glpk.html .

[6] E. Michael Gertz and Stephen Wright, OOQP, *A package for solving convex quadratic programming problems*, http://www.cs.wisc.edu/ swright/ooqp .

[7] Todd Velthuizen,*http://www.oonumerics.org/oon/*, Last updated October, 2003.

[8] Jack Dongarra, *http://www.netlib.org/utk/people/JackDongarra/la-sw.html*, Last updated April, 2003.

[9] GNU *Scientific Library*, http://www.gnu.org/software/gsl/

[10] TNT *Template Numerical Toolkit*, http://math.nist.gov/tnt/ .

[11] CLHEP *A Class Library for High-Energy Physics*, http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/ .

[12] LINALG *Basic Linear Algebra classlib*, http://okmij.org/ftp .

[13] JAMA *A Java Matrix Package*, http://math.nist.gov/javanumerics/jama/ .

[14] J. H. Wilkinson and C. Reinsch, *Handbook for Automatic Computation, Linear Algebra, Vol. II*, Springer, New York, 1971

[15] Gene H. Golub and Charles F. Van Loan, *Matrix Computations (Johns Hopkins Series in the Mathematical Sciences)*, 3rd edition (November 1996) .

[16] C.R. Birchenhall, ftp://ftp.mcc.ac.uk/pub/matclass/ .