GEMINI AND HEPFITTING COMPONENTS OF LHC++

Zbigniew Szatkik (*)
Department of Applied Mathematics, University of Mining and Metallurgy, Cracow, Poland

Abstract
Design concepts and usage of Gemini and HEPFitting components of LHC++ environment are presented. Gemini's approach to error analysis and the relation of Minos-type errors and Hessian-based errors are briefly discussed.

1. INTRODUCTION

Gemini [1] is a GEneral MINImization and error analysis package implemented as a C++ class library. It provides a unified object oriented Application Programming Interface to various minimizers. The currently supported set of minimizers (Minuit [2] and NAG C [3]) can be extended. For the common subset of functionality, it is up to the user which minimization engine does the work. The user can easily switch between various minimizers without essential changes in the application code. Gemini finds a minimum of an objective function, possibly subject to general constraints, and performs an error analysis. While being a part of LHC++, Gemini only depends on the actual minimizer and may thus be used without the other LHC++ components.

HEPFitting [4] is a collection of C++ fitting classes, based on Gemini. It allows for loading data, defining a model and a fitting criterion, performing a fit and obtaining fit results, including error analysis. Basic HEPFitting classes are derived from Gemini classes and thus inherit Gemini's basic minimization and error analysis features. Additional, special features of HEPFitting include a simple way of setting parameters for the error analysis and a mechanism for building a model out of predefined components as well as for defining an arbitrary model. A suitable objective function to be minimized is automatically created, according to the fitting method chosen. Models can be fitted both to HTL histograms [5] and to arbitrary data points which can be loaded via user's arrays.

The intended primary area of application of both packages is batch data processing. Thus, apart from a simple text-mode printout of Minos- and Hessian based confidence regions, the packages do not provide any other visualization tools.

2. THE CONCEPT AND IMPLEMENTATION OF GEMINI

2.1 Prerequisites

Minuit, a FORTRAN minimization package, has been in successful use in HEP for more than 30 years. Especially, its error analysis has become a de facto standard in physics data analysis. An analysis of users' requirements at the early stage of the Minuit-replacement project [6] showed that a Minuit's successor should provide Minuit's functionality in a Minuit-style and, additionally, that some new features like, e.g. general constraints, should be provided. It has also been stressed on many occasions that Minos-type error analysis must be possible with the new package, in order that it can be accepted by the HEP community.

No single commercially available product could fulfill all the requirements, although we identified the family of NAG C minimizers as the most prospective candidate, both with respect to functionality and to performance. We thus decided to write a small (currently about 3200 lines of code) C++ open-ended package which could internally use various working engines and which would

(*) The author was working for LHC++ while being a Scientific Associate at CERN in 1997 and 1998.
provide all the requested functionality in the requested style. Switching between various minimizers should not require any essential changes in the user's code in order to ease the cross-checks and the transition period for the Minuit-users. This is how Gemini came to life.

Special effort proved to be necessary in order to implement Minos-type errors. Being standard within the HEP community, this type of errors is by no means standard in the non-HEP world. To the best of our knowledge, Minuit was the only package which implemented this type of errors. We thus decided to write a special Minos analysis module for Gemini, so that any type of minimizer could be plugged-in, regardless on whether it is able to perform the Minos analysis or not.

2.2 Functions and objects

The basic types of objects used within Gemini are: the objective function object, the minimization object and the contour object.

The objective function is the function to be minimized. It is defined by the user as a C/C++ function, which computes the objective function value and, optionally, its gradient, for a given vector of function arguments. The function is then captured into an objective function object, which is of the type \texttt{OBJfun}.

The minimization object is the main object, which contains the complete problem definition. It also provides methods for assigning an objective function object, defining the objective function arguments and their admissible regions, setting minimization options, running a minimizer, obtaining the current status of the minimization process, obtaining results and error analysis. If the minimization object is declared as being of type \texttt{CMinuit}, then Minuit is used as the minimization engine. Similarly, if it is of type \texttt{NAGmin}, then NAG C minimizers are used. A generic pointer of the type (\texttt{GEmini *}) can point both to \texttt{CMinuit} and \texttt{NAGmin} type objects and can be used, if minimization objects are dynamically allocated on the heap. This allows the user to select the minimizer at run time rather than at compilation time.

A contour is a set of points from the boundary of a (bounded) set in a two-dimensional subspace. In Gemini, it typically represents either an elliptical boundary of a Hessian-based confidence region for a selected pair of parameters, or a Minos contour, i.e. the curve on which the minimum of the objective function, with respect to all the remaining parameters, equals the current minimum plus a user-defined value. Contours are implemented in Gemini as an abstract data type with overloaded assignment and addition operators. Addition means, in this case, merging and can be used for overlaying the contours. A public method \texttt{plot()} produces a text-mode printout of, possibly overlayed, contours.

2.3 Example

The following example is a complete C++ program. It minimizes the Powell's quartic function of four arguments defined as

\[ f(x, y, z, u) = (x + 10y)^2 + 5(z - u)^2 + (y - 2z)^4 + 10(x - u)^4 \]

subject to one nonlinear constraint

\[ x^2 + y^2 + z^2 + u^2 = 1 \]

and one linear constraint

\[ x + y + z + u = 0. \]

Note that, for the unrestricted problem with the Powell's objective function, the Hessian becomes singular at the origin and the minimum point is not uniquely defined. With added constraints, the Hessian singularity point is excluded from the admissible region. Also note that this
nonlinear optimization problem cannot be solved with Minuit, which only allows for simple bound constraints to be imposed.

```c
#include "gemini.h"

inline double square(double x) { return(x*x); }
const int nop=4; // number of parameters

// objective function (Powell's quartic function of 4 vars)
void myfun(int n, double g[], double *objf, const double parms[], int code)
{
    // define aliases for convenience
    const double &x=parms[0], &y=parms[1], &z=parms[2], &u=parms[3];

    *objf = square(x+10*y) + 5*square(z-u) + square(square(y-2*z)) +
            10*square(square(x-u));
    if(code == 2){
        // gradient components
        g[0] = 2*(x+10*y) + 40*(x-u)*square(x-u);
        g[1] = 20*(x+10*y) + 4*(y-2*z)*square(y-2*z);
        g[2] = 10*(z-u) - 8*(y-2*z)*square(y-2*z);
        g[3] = -10*(z-u) - 40*(x-u)*square(x-u);
    }
}

// non-linear constraint function (sum of squares equals 1)
void nlf(int nop, double g[], double *val, const double parms[], int code)
{
    const double &x=parms[0], &y=parms[1], &z=parms[2], &u=parms[3];

    *val = square(x) + square(y) + square(z) + square(u);
    if(code == 2){
        // gradient components
        g[0]=2*x; g[1]=2*y; g[2]=2*z; g[3]=2*u;
    }
}

int main()
{
    // capture objective function into objective function object
    OBJfun fcn(myfun);

    // create main minimization object
    NAGmin nlp("Non-linear optimization example", nop, &fcn);

    // impose non-linear constraint
    if( nlp.setNlinConstraint(11, nlf, 1.0, 1.0) ) exit(1);

    // impose linear constraint
    double lincoef[nop];
    if( nlp.setLinConstraint(1, lincoef, 0, 0) ) exit(1);

    nlp.printSetup();
    if( nlp.minimize() ) exit(1);
    nlp.printResults();
    return(0);
}
```
3. ERROR ANALYSIS IN GEMINI

3.1 General concept of errors

The general concept of ‘errors’ or ‘uncertainties’ in Gemini is the same as in Minuit. For a given objective function \( F(\theta) \) to be minimized, with \( \theta = (\theta_1, ..., \theta_p) \), and for a given error parameter UP, the ‘uncertainty set' US of the solution \( \tilde{\theta} = (\tilde{\theta}_1, ..., \tilde{\theta}_p) \) is defined as

\[
\text{US} = \{ \theta : F(\theta) - F(\tilde{\theta}) \leq \text{UP} \}
\]  

(1)

For any sub-vector of \( \tilde{\theta} \), the uncertainty set is constructed as the orthogonal projection of US onto the corresponding plane spanned by the selected components.

This purely geometrical concept is meaningful, in qualitative sense, for arbitrary objective functions. ‘Errors’ or ‘uncertainties’ are related to the shape of the objective function in a neighborhood of the minimum.

Well-defined quantitative meaning, in probabilistic terms, can be assigned to such defined ‘errors' or 'uncertainties' in statistical problems, when the objective function is a fit criterion, for example a chi-squared, log-likelihood or least squares loss function.

Error analysis based on the plain difference \( F(\theta) - F(\tilde{\theta}) \) is called Minos analysis, as in Minuit. In this context, we also use terms like Minos error and Minos confidence region. Minos analysis can be computationally very costly, however, as it requires multiple function minimization to find points on the boundary of US or of its projection. It will be seen below, how Minos analysis can formally be justified in statistical terms. For maximum likelihood estimators and standard minimum chi-squared estimators, for example, it can be done via the asymptotic chi-squared distribution of a suitably transformed likelihood ratio.

A standard way to overcome the computational difficulty of Minos analysis is to approximate \( F(\theta) - F(\tilde{\theta}) \) with \( 0.5 \cdot (\theta - \tilde{\theta})^T H(\theta - \tilde{\theta}) \), with \( H \) being the Hessian of \( F \) at \( \tilde{\theta} \). One obtains this approximation via the standard Taylor expansion of \( F \) around \( \tilde{\theta} \) and using the fact that the gradient of \( F \) at the minimum is zero. With this approximation, approximate versions of both US and its projections can be found analytically, so that multiple function minimization can be avoided. This leads to the standard Hessian-based error analysis and is related to asymptotically normal distributions of estimators.

In the following sections, those two approaches are described in more detail and their links to standard statistics exposed. Only unconstrained minimization problems are discussed here. For a discussion of error analysis for problems with constraints, see e.g. [1]. It is always assumed that the problem is regular enough for the underlying mathematical theory to be applicable. The aim of this description is to expose the main ideas rather than to present technical details. Relevant mathematical results can be found, for example, in the books [7-9].

Ref. [10] is a standard statistical reference for HEP-physicists. It contains, in particular, a discussion of the Minos idea in less formal terms of an 'implicit transformation to linearity and back', which provides further insight into the idea of Minos.

3.2 Minos error analysis

The Minos uncertainty set US for the whole vector \( \theta \) is defined above in (1). In order to obtain an uncertainty set for two components only, say \( \theta_1 \) and \( \theta_2 \), we have to project US onto the plane spanned by those components. This projection is a set of points \( (\theta_1, \theta_2) \) such that \( F(\theta) \leq F(\tilde{\theta}) + \text{UP} \), for some \( \theta_3, ..., \theta_p \). Equivalently, it is the set of points \( (\theta_1, \theta_2) \) such that the
minimum of $F(\theta)$ with respect to $\theta_3,\ldots,\theta_p$ and with $\theta_1$ and $\theta_2$ fixed is not greater than $F(\tilde{\theta}) + \text{UP}$. The boundary of this set is thus the contour of the function

$$\tilde{F}(\theta_1,\theta_2) = \min_{\theta_3,\ldots,\theta_p} F(\theta)$$

which corresponds to $\tilde{F}(\theta_1,\theta_2) = F(\tilde{\theta}) + \text{UP}$. For a single parameter, say $\theta_1$, we define a function

$$\hat{F}(\theta_1) = \min_{\theta_2,\ldots,\theta_p} F(\theta)$$

and construct the uncertainty set, or the projection of US, as $\{\theta_1 : \hat{F}(\theta_1) \leq F(\tilde{\theta}) + \text{UP}\}$. For a regular function $F$, genuine local minimum $\tilde{\theta}$ and 'small' UP, this will be an interval $[\theta_1, \tilde{\theta}_1]$, say. The positive and negative Minos errors are then defined as, correspondingly, $\tilde{\theta}_1 - \theta_1$ and $\theta_1 - \tilde{\theta}_1$.

In order to give Minos errors a quantitative, statistical meaning, let us assume first that $F$ equals -$2\log$-likelihood for a regular statistical model. The unrestricted minimum $\tilde{\theta}$ of $F$ is then a maximum likelihood estimator of $\theta$. Let further $\hat{\theta}$ be the minimum of $F$, subject to $r$ independent restrictions on $\theta$. It is well-known that, for any true $\theta$ which satisfies the restrictions, $F(\hat{\theta}) - F(\tilde{\theta})$ is asymptotically chi-squared distributed with $r$ degrees of freedom - a fact used for the construction of the so-called asymptotic likelihood ratio test ($\lambda$-test).

It follows immediately that, for any true $\theta$ with the given values of the first two components, $\hat{F}(\theta_1,\theta_2) - F(\tilde{\theta})$ is asymptotically chi-squared distributed with two degrees of freedom (we impose two constraints by fixing the values of $\theta_1$ and $\theta_2$) and, for any true $\theta$ with the given value of the first component, $\hat{F}(\theta_1) - F(\tilde{\theta})$ is asymptotically chi-squared distributed with one degree of freedom (we fix the value of $\theta_1$ only).

A standard Neyman asymptotic (1-$\alpha$)-confidence region for $(\theta_1,\theta_2)$ can then be constructed as

$$\{ (\theta_1,\theta_2) : \hat{F}(\theta_1,\theta_2) - F(\tilde{\theta}) \leq c_\alpha \}$$

with $c_\alpha$ being the (1-$\alpha$)-quantile of the chi-squared distribution with two degrees of freedom. This is exactly the projection of US, with $\text{UP} = c_\alpha$, onto the plane spanned by the first two components.

Similarly, an asymptotic Neyman (1-$\alpha$)-confidence region for $\theta_1$ is

$$\{ \theta_1 : \hat{F}(\theta_1) - F(\tilde{\theta}) \leq c_\alpha \}$$

with $c_\alpha$ being the (1-$\alpha$)-quantile of the chi-squared distribution with one degree of freedom. Again, this is the projection of US with $\text{UP} = c_\alpha$ onto the first axis.

With obvious modifications, similar argument applies, of course, to any subset of the components of $\theta$, which leads to the following conclusion:

If $F$ equals -$2\log$-likelihood, then Minos confidence regions for $r$ components of $\theta$ have the asymptotic coverage probability 1-$\alpha$, if UP is the (1-$\alpha$)-quantile of the chi-squared
distribution with \( r \) degrees of freedom. With \( r = 1 \) and \( \text{UP} = 1 \), the coverage probability corresponds to that of a '\pm one-sigma error bar' for a single parameter.

The scale factor of \( F \) is essential. Additive terms, which do not depend on \( \theta \) can be dropped, however.

In Gaussian models, -2*\text{log-likelihood} equals, up to a constant, additive term, the chi-squared fit criterion and the whole analysis applies. In many other cases, the equality holds asymptotically, thus validating the Minos analysis with \( F \) being the chi-squared fit criterion. In particular, this is true for the Poisson histogram cells counts model (see e.g. [1]).

### 3.3 Hessian-based error analysis

In the Hessian-based error analysis, \( F(\theta) - F(\tilde{\theta}) \) is approximated with \( 0.5 \cdot (\theta - \tilde{\theta})^T H(\theta - \tilde{\theta}) \), with \( H \) being the Hessian of \( F \) at \( \tilde{\theta} \). The approximate version \( \text{US'} \) of the uncertainty set \( \text{US} \), corresponding to a given value of the UP parameter takes then the form

\[
\text{US'} = \{ \theta : 0.5 \cdot (\theta - \tilde{\theta})^T H(\theta - \tilde{\theta}) \leq \text{UP} \}.
\]  

(2)

The orthogonal projection \( \text{US'}_i \) of \( \text{US'} \) onto the plane spanned by, say, the first \( r \) components of \( \theta \) consists of all points \( (\theta_1, ..., \theta_r) \) such that the minimum of \( (\theta - \tilde{\theta})^T H(\theta - \tilde{\theta}) \) with respect to \( \theta_{r+1}, ..., \theta_p \) is not greater than \( 2 \cdot \text{UP} \). Let us split \( \theta - \tilde{\theta} \) into two sub-vectors: \( \theta_i \) consisting of the first \( r \) components and \( \theta_{\mu} \) consisting of the remaining \( p-r \) components. Correspondingly, we can write

\[
H = \begin{bmatrix}
H_1 & H_{12} \\
H_{12}^T & H_2
\end{bmatrix}
\]

with \( H_1 \) of size \((r, r)\) and \( H_2 \) of size \((p-r, p-r)\). Looking for a minimum with respect to \( \theta_{\mu} \) and with \( \theta_i \) fixed, we have then

\[
(\theta - \tilde{\theta})^T H(\theta - \tilde{\theta}) = \theta_i^T H_i \theta_i + 2 \theta_i^T H_{12} \theta_{\mu} + \theta_{\mu}^T H_2 \theta_{\mu} = G(\theta_{\mu})
\]

and

\[
\text{grad} G(\theta_{\mu}) = 2 H_{12}^T \theta_i + 2 H_2 \theta_{\mu}.
\]

The equation \( \text{grad} G(\tilde{\theta}_{\mu}) = 0 \) gives the minimum point \( \tilde{\theta}_{\mu} = -H_2^{-1} H_{12}^T \theta_i \). The minimum value is

\[
G(\tilde{\theta}_{\mu}) = \theta_i^T (H_1 - H_{12} H_2^{-1} H_{12}^T) \theta_i
\]

which gives

\[
\text{US'}_i = \{ \theta_i : 0.5 \cdot \theta_i^T (H_i - H_{12} H_2^{-1} H_{12}^T) \theta_i \leq \text{UP} \}.
\]

On the other hand, using the symmetric, block matrix inversion formula, we have

\[
\begin{bmatrix}
H_1 & H_{12} \\
H_{12}^T & H_2
\end{bmatrix}^{-1} = \begin{bmatrix}
(H_1 - H_{12} H_2^{-1} H_{12}^T)^{-1} & X \\
X^T & H_2^{-1} (I - H_{12}^T X)
\end{bmatrix}
\]

(3)

with \( X^T = -H_2^{-1} H_{12}^T (H_1 - H_{12} H_2^{-1} H_{12}^T)^{-1} \). This means that, with \( S = (0.5 \cdot H)^{-1} \), we can write
US' = \{ \theta : (\theta - \tilde{\theta})^T S^{-1} (\theta - \tilde{\theta}) \leq UP \}

and, denoting by $S_r$ the upper left $(r, r)$ portion of $S$, the projection $US'_r$ takes the form

$$US'_r = \{ \theta_r : \theta_r^T S_r^{-1} \theta_r \leq UP \}.$$  \hfill (4)

In order to set this in relation with statistics, recall that if $F$ equals $-2 \times \text{log-likelihood}$, then the maximum likelihood estimator $\tilde{\theta}$ is, in regular cases, asymptotically normally distributed

$$\tilde{\theta} \sim \text{AN}(\theta, I_\theta^{-1})$$

where $I_\theta^{-1} = 0.5 \cdot E_\theta H$ is the Fisher information matrix for the whole data set. Again, in regular cases, one can reasonably assume that $E_\theta H \approx H(\tilde{\theta})$ and use the inverse of $0.5 \cdot H(\tilde{\theta})$ as an estimate of the asymptotic covariance matrix of $\tilde{\theta}$.

This is clearly related to (2) and means that, since $(\theta - \tilde{\theta})^T S^{-1} (\theta - \tilde{\theta})$ is asymptotically chi-squared distributed with $p$ degrees of freedom, in order to have the asymptotic $1-\alpha$ coverage probability for $US'$, one should set $UP$ to the $(1-\alpha)$-quantile of the chi-squared distribution with $p$ degrees of freedom.

Further, $S_r$ in (4) can be interpreted as the covariance matrix of the marginal distribution of $(\theta_1, ..., \theta_r)$ and, in view of its asymptotic normality, setting $UP$ in (4) to the $(1-\alpha)$-quantile of the chi-squared distribution with $r$ degrees of freedom, we get the asymptotic coverage probability $1-\alpha$ for $US'_r$.

For the chi-squared fit criterion, the approximation argument from the previous section applies. More generally, the above argument can be extended to any M-estimator, in which case the asymptotic covariance matrix needs not to be the inverse of the Fisher information matrix, but continues to be the (properly normalized) inverted Hessian.

For practical recommendations on how to set the value of the error parameter $UP$, see [1]. Note, however, that if the HEPFitting package is used, then the value of $UP$ is automatically set, according to the fitted model and the fitting method.

4. THE CONCEPT AND USAGE OF HEPFITTING

4.1 Basic features

HEPFitting is a small C++ package (currently about 1200 lines of code) built on top of Gemini with the aim to ease the most common fitting applications of the minimizers. HEPFitting can internally use both Minuit and NAG C minimizers. No change in the application code is needed, the switch being done when the application code is being compiled. If _MINUIT_ is defined at compilation time, then Minuit is used. Otherwise, NAG C minimizers are used.

The main fitting object of the type HEPHistofit contains a complete definition of the fitting problem and provides methods for defining the problem, loading data, performing the fit and obtaining results. The fitting region can be restricted by imposing interval limits on space variables, as well as by including/excluding any single histogram bin or data point.

Currently implemented fitting criterions are chi-squared and Poisson maximum likelihood. Various options are provided for handling empty bins and zero-errors data points. The user only defines the model itself. A suitable objective function to be minimized is automatically defined by HEPFitting, according to the selected fitting criterion, and the error parameter is properly set so that
the computed errors are the standard one-sigma-errors. All available components of the objective function gradient are constructed from the model function gradient, as provided by the user in the model function. The package checks, whether the data set and the model assigned to the fitting object are compatible with respect to the space dimensionality.

Special methods are provided for obtaining both elliptical (or Hessian-based) and Minos confidence regions for a selected pair of parameters, which only require that the user specifies the requested confidence level.

4.2 Defining the model

The fitted model is encapsulated in an object of the type *MODELfun*, which can be used to capture an arbitrary user's model, defined as a C/C++ function, as well as to compose a model out of predefined elementary models, like Gaussian, polynomial and exponential ones. Any additional functionality needed in the user's application can be added through the inheritance mechanism, in which case the user derives his/her own class from *MODELfun* and overrides the virtual member function *modelfun* with the actual model function.

When a model is fitted to a histogram, the model function values in bin reference points are multiplied by bin volumes, before being compared to bin contents. The model function thus represents the intensity function of the underlying Poisson process or a density function and is independent of the particular histogram binning. The user can ignore bin volumes by selecting special options, as described in [4].

Various ways of defining a model are described in detail in [4]. Here, we only discuss the way a model can be constructed out of predefined components.

A single-component model may easily be defined with a specialized constructor of *MODELfun* which takes the component name as an argument (G for Gaussian, E for exponential, Pn for polynomial of degree n). The model object created this way may immediately be assigned to a fitting object. For example, in order to fit a Gaussian and then a second-degree polynomial, one can proceed as follows

```
...  
MODELfun model1("G");  
MODELfun model2("P2");  
HEPHistoFit myFittingObject;  

myFittingObject.setModel( &model1 );  
// perform the fit  
...  
myFittingObject.setModel( &model2 );  
// perform the fit  
...  
```

In order to compose a multi-component model from the built-in standard models, the user has to derive his/her own class from *MODELfun* and override the virtual member function *modelfun*. The overridden function defines the expression. The components used in the expression may be added *en bloc* using the *MODELfun* public method *setComponents( char *string )*, where *string* is composed of blank- or comma-separated components symbols. For other techniques which may be used, see [4].

The order, in which the components are added or, equivalently, the order in which they are placed in *string* is significant. The global vector of parameters consists of the components’ parameters stored in the order, in which the components have been added. Similarly, the indices, by which the components are referenced in the expression, correspond to the order, in which the components have been added.
The component functions are referenced in the expression as \( f(i,x,p) \), where \( i \) stands for the component's index and takes values 0, 1, 2, ..., the vector of space coordinates is denoted by \( x \) and \( p \) is the global vector of parameters, as passed through the arguments of the \( \text{modelfun}(...) \) function.

If only the four basic operations (+ - * /) are used to build the expression, then there is a simple way to also provide the gradient with respect to parameters, which can significantly improve the performance of the minimizer. The rule can be formulated as follows: "Look at the components \( f(i,x,p) \) as if they were all functions of the same single variable \( p \), use \( df(i,x,p) \) to denote derivatives, apply well-known differentiation rules and assign to \( g \) the resulting expression". A suitable gradient vector will then automatically be created and used.

4.3 Example

In the following example, the model is defined as the product of a second-degree polynomial and a Gaussian. Since the parametrization of such a model is redundant, the Gaussian's 'mass' parameter has to be fixed, before performing a fit. Otherwise, the model would not be identifiable.

```cpp
class myModelObject : public MODELfun{
public:
    // define the expression by overriding modelfun()
    double modelfun(const double x[], const double p[], array_n<double>& g, int code){
        // compute gradient, if requested
        if(code==2)
            g = df(0,x,p)*f(1,x,p)+f(0,x,p)*df(1,x,p);
        // return model function value
        return f(0,x,p)*f(1,x,p);
    };

int main(){
    // create empty fitting object
    HEPHistoFit myFittingObject;

    // create model object and define model's components, then assign
    myModelObject P2Gmodel;
P2Gmodel.setComponents("P2,G");
    myFittingObject.setModel( &P2Gmodel );

    // load data, set initial parameters' values e.t.c.
    ...

    // fix the first parameter of the Gaussian at 1 (It's
    // the 4th parameter preceded by 3 parameters of P2
    myFittingObject.parmDef(4,"Gmass",1,1,1,1);

    // perform a fit
    myFittingObject.perform(PoissonMLfit);
    myFittingObject.printResults();

    // 90% confidence regions for the free parameters of the Gaussian
    GeminiContour c1, c2;
    myFittingObject.ellipticalConfidenceRegion(5,6,c1,0.90);
    myFittingObject.minosConfidenceRegion(5,6,c2,0.90);
    (c1+c2).plot(); // overlay and plot
}
```
5. CONCLUSIONS

With Gemini, we believe to have a flexible and open framework for function minimization. Numerous tests have proved that the family of NAG C minimizers can satisfy the requirements of the HEP community. However, with its own Minos analysis module, Gemini becomes independent of the minimizer actually used, so that NAG C may easily be replaced with another minimizer, without affecting the users' code.

Using HEPFitting helps the users to keep to a minimum the amount of code written in order to perform standard fits. The implementation of the build-in elementary models and the internal mechanism of computing all available derivatives of the objective function result in considerable improvements in the minimizer's performance, by keeping the number of the model function calls at the minimum.

ACKNOWLEDGEMENTS

The author kindly acknowledges CERN's hospitality while working on the CSC 99 lecture.

REFERENCES


