LC-TOOL-2009-001

MarlinKinfit: An Object–Oriented Kinematic Fitting Package

Benno List, Jenny List

25.3.2009

Abstract

This note describes the core functionality of the MarlinKinfit package for performing constrained kinematic fits. It provides a modular framework in which particles and constraints among them can be combined in a flexible way to describe any desired event hypothesis. The actual minimisation is done by a fit engine which can be exchanged independently from the event hypothsis. The mathematics behind the basic fit engine in the package is described in detail. In the end, an example application is given.

1 Introduction

In a kinematic fit, a number of four–vectors, parametrized by parameters such as ϕ, θ and E, are fitted such that they fulfill constraints such as transverse momentum conservation or an invariant mass constraint, and minimize a χ^2 that describes the deviations between measured and fitted quantities.

In general, we have a number V of four–vectors $p_{\mathrm{v}} = (E_{\mathrm{v}}, p_{\mathrm{x,v}}, p_{\mathrm{y,v}}, p_{\mathrm{z,v}})$ that depend on N parameters a_{n} (for instance, p_1 depends on $a_1 = \phi_1$, $a_2 = \theta_1$, and $a_3 = E_1$). In a more general case, we could also have parameters such as an energy scale that affect more than one four-vector, so that in full generality we should assume that each four-vector depends on any number of parameters.

If these parameters have been measured with a certain accuracy, as for jets or charged leptons, deviations between the measured values y_{n} and the fitted values η_{n} lead to a χ^2.

In addition, there may be J unmeasured parameters ξ_{j}, for example the momentum components of a neutrino that escapes detection. So, in general the fit problem involves N measured parameters and J unmeasured parameters.

In addition, there will be a number of $K \geq 0$ constraints, which are expressed by functions f_{k} that depend on the $N + J$ parameters:

$$f_{\mathrm{k}} = f_{\mathrm{k}}\left(\eta_1 ... \eta_{\mathrm{N}}, \xi_1 ... \xi_{\mathrm{J}}\right),$$

where constraint k is given by the condition $f_{\mathrm{k}} = 0$. In practive, the constraints will in most cases be expressed in terms of the four-vector components, such as energy-momentum constraints or mass constraints.

Moreover, there may be additional χ^2 contributions from "soft" constraints, e. g. when one demands that the invariant mass of several four–vectors be compatible with a particle's mass (e. g. Z^0) within the particle's width. However the minimisation method decribed in this note cannot handle soft constraints in the presence of unmeasured quantities. Therefore we will discuss soft constraints in detail in a later publication describing the second fitting engine in the package, the `NewtonFitter`, based on the Newton–Raphson method, which can treat them also in the presence of unmeasured parameters.

One seeks now a (global) minimum of the total $\chi^2\,(\eta_1...\eta_\mathrm{N}, y_1...y_\mathrm{N})$, under the K conditions $f_\mathrm{k}\,(\eta_1...\eta_\mathrm{N}, \xi_1...\xi_\mathrm{J}) = 0$.

This minimum is characterized by N conditions

$$\frac{\partial\chi^2}{\partial\eta_\mathrm{n}} = 0.$$

Using the method of Lagrange multipliers, one defines a new quantity

$$\chi^2_\mathrm{T} = \chi^2\,(\eta_1...\eta_\mathrm{N}, y_1...y_\mathrm{N}) + 2\cdot\sum_{k=1}^{K}\lambda_\mathrm{k}\cdot f_\mathrm{k}\,(\eta_1...\eta_\mathrm{N}, \xi_1...\xi_\mathrm{J}),$$

where K additional unknown parameters are introduced, the Lagrange multipliers λ_k.

This χ^2_T depends on a total of $N+J+K$ unknowns, plus N measured parameters:

$$\chi^2_\mathrm{T} = \chi^2_\mathrm{T}\,(\eta_1...\eta_\mathrm{N}, \xi_1...\xi_\mathrm{N}, \lambda_1...\lambda_\mathrm{K}, y_1...y_\mathrm{N}),$$

and one can write the minimum conditions as

$$\begin{aligned}
\frac{\partial\chi^2_\mathrm{T}}{\partial\eta_\mathrm{n}} &= \frac{\partial\chi^2}{\partial\eta_\mathrm{n}} + 2\cdot\sum_{k=1}^{K}\lambda_\mathrm{k}\cdot\frac{\partial f_\mathrm{k}}{\partial\eta_\mathrm{n}} = C_{\eta_\mathrm{n}} + 2\cdot\sum_{k=1}^{K}\lambda_\mathrm{k}\cdot\frac{\partial f_\mathrm{k}}{\partial\eta_\mathrm{n}} = 0\\
\frac{\partial\chi^2_\mathrm{T}}{\partial\lambda_\mathrm{k}} &= f_\mathrm{k} = 0,
\end{aligned}$$

where we have introduced N functions

$$C_{\eta_\mathrm{n}}\,(\eta_1...\eta_\mathrm{N}, y_1...y_\mathrm{N}) = \frac{\partial\chi^2}{\partial\eta_\mathrm{n}}$$

Note that while one seeks a global minumum of χ^2 within the subspace spanned by the constraints, one will *not* have a global minimum of χ^2_T, which is not bounded from below, even if one would express the constraints in a way that the f_k are always nonnegative. This excludes the use of minimization packages like MINUIT [1] to solve the problem.

In the following, we will first introduce the basic abstractions which are used in the MarlinKinfit package in order to allow to flexible configurations of the constraints, the measured and unmeasured quantities independent of the actual fitting engine. In section 3 we will describe the mathematics and the implementation of the reference fit engine in the package, the so-called `OPALFitter`, which is derived from the kinematic fitting software used by the OPAL experiment. Sections 4 and 5 contain information about the currently implemented constraints and particle parametrisations. Finally in section 6 an example application to four jet events is presented.

2 MarlinKinfit

The kinematic fitting package presented here is incorporated into the Marlin framework [2]. The source code is available within the MarlinReco package from [3].

The purpose of this package is to provide an object-oriented framework for kinematic fits. It provides a set of general base classes that are a priori not dependent on the experiment or the event hypothesis under consideration. Although a growing set of concrete cases is implemented, the actual parametrisation of the particles' four-momenta and the formulation of different constraints can easily be adapted by the user according to the desired physics process.

Also the actual fitting algorithm is exchangable. At the moment, two fitters are available, the `OPALFitter`, which is described in detail below, and the NewtonFitter, which is still under development and which will be described in a later publication.

2.1 Basic Concepts

This fitting package is based on three concepts:

- A fitting engine,
- constraints, and
- fit objects.

These concepts are reprensented by different classes, with abstract base classes that define the interfaces and concrete subclasses for the implementation.

Fit objects, which correspond to entities like jets, muons, or neutrinos, store information on their parameters and encapsulate the details of the parametrisation. If parameters have been measured, two parameter sets are stored, the measured and the fitted ones, together with their respective covariance matrices. The fit objects provide information on the number of their internal parameters, whether they are measured or unmeasured, and information on their physical quantities, in particular four vector components, and values of derivatives of the four vector components with respect to the various parameters.

Constraints between fit objects are formulated in terms of the physical quantities, in particular four vector components, of the fit objects. A constraint is a function of the physical quantities of several fit objects, which has to be zero if the constraint is fulfilled. A constraint object is associated with several fit objects, and communicates with them so that it can evaluate the value of the constraint function and its derivatives with respect to the internal parameters of the fit objects. These derivatives are evaluated according to the chain rule, so that the constraint object contains only derivatives with respect to the physical quantities, which are then multiplied by the derivatives with respect to the parameters that are provided by the fit objects.

The fit engine sets up and solves the system of equations of the fitting problem. It maintains lists of constraints and fitting objects.

2.2 How to Perform a Fit

To set up a fitting problem, first the fit objects have to be created:

```
JetFitObject j1 (47., 0.84 , 0.64,  5.0, 0.1, 0.1, 0.);
JetFitObject j2 (45., 2.30 , 2.50,  5.0, 0.1, 0.1, 0.);
JetFitObject j3 (46., 0.996, 3.83,  5.0, 0.1, 0.1, 0.);
JetFitObject j4 (42., 2.21 , 5.82,  5.0, 0.1, 0.1, 0.);
```

Here, the first three arguments are energy, polar and azimutal angle of the jet, the next three arguments are their errors, and the last argument is the jet mass.

Then, constraint objects that express constraints such as

- sum of $p_x = 0$
- sum of $p_y = 0$
- invariant mass of two jet pairs is equal

are created

```
PxConstraint pxc;     // sum(px)=0
PyConstraint pyc;     // sum(py)=0
MassConstraint mc(0.); // mass(particle set 1)-mass (particle set 2)=0
```

and the fit objects are associated with them:

```
pxc.addToFOList (j1);
pxc.addToFOList (j2);
pxc.addToFOList (j3);
pxc.addToFOList (j4);

pyc.addToFOList (j1);
pyc.addToFOList (j2);
pyc.addToFOList (j3);
```

```
pyc.addToFOList (j4);

mc.addToFOList (j1, 1);
mc.addToFOList (j2, 1);
mc.addToFOList (j3, 2);
mc.addToFOList (j4, 2);
```

Finally, the fit engine is created, and the fit and constraint objects are passed to the engine, which finally performs the fit:

```
OPALFitter fitter;

fitter.addFitObject (j1);
fitter.addFitObject (j2);
fitter.addFitObject (j3);
fitter.addFitObject (j4);

fitter.addConstraint (pxc);
fitter.addConstraint (pyc);
fitter.addConstraint (mc);

double prob = fitter.fit();
```

After the fit, the fit objects have their fitted parameters set to the fit result.

The fit engine has a list of fit objects. First it sets up a global list of measured and unmeasured quantities, and each of the internal parameters of the fit objects gets, in addition to its internal or local number, a global number. These global parameter numbers are stored, for efficiency reason, within the fit objects. Similarly, the constraints are numbered by the fit engine, and these global constraint numbers are also stored in the constraint objects. Remember that to each constraint there belongs a new parameter, its Lagrange multiplier. Measured parameters, unmeasured parameters, and Lagrange multipliers are globally numbered in consequitive order, starting at 0.

3 OPALFitter

The `OPALFitter` class reimplements the code originally implemented in the package WWFIT of the OPAL collaboration. This package uses the algorithm detailed in [4, 5], which is repeated here for convenience.

The notation here are those used by [4] and are the same in the code.

3.1 Mathematical Method

We have N measured quantities η_n, where the measured values are given by y_n with an $N \times N$ covariance matrix V, and J unmeasured quantities ξ_j, subject to K constraints expressed as

$$f_k\left(\eta_1, ..., \eta_N, \xi_1 ... \xi_J\right) = 0, \qquad k = 1...K. \tag{1}$$

For simplicity, we arrange these values and constrains functions in vectors $\vec{\eta}$, $\vec{y}$, $\vec{\xi}$, and $\vec{f}$.

We introduce K additional unknowns λ_k, the Lagrange multipliers, that form a vector $\vec{\lambda}$.

The total χ_T^2 that should be minimized is given by

$$\chi_T^2(\vec{\eta}, \vec{\xi}, \vec{\lambda}) = (\vec{y} - \vec{\eta})^T \cdot V^{-1} \cdot (\vec{y} - \vec{\eta}) + 2\vec{\lambda}^T \cdot \vec{f}(\vec{\eta}, \vec{\xi}). \tag{2}$$

Taking the various derivatives leads to the set of equations

$$\begin{array}{rcll} \nabla_\eta \chi_T^2 & = & -2V^{-1} \cdot (\vec{y} - \vec{\eta}) + 2\vec{F}_\eta^T \cdot \vec{\lambda} = \vec{0}, & (N \text{ equations}) \\ \nabla_\xi \chi_T^2 & = & \vec{F}_\xi^T \cdot \vec{\lambda} = \vec{0}, & (J \text{ equations}) \\ \nabla_\lambda \chi_T^2 & = & 2\vec{f}(\vec{\eta}, \vec{\xi}) = \vec{0}, & (K \text{ equations}) \end{array} \tag{3}$$

where F_η and F_ξ are matrices of dimension $K \times N$ and $K \times J$, respectively, defined as

$$(F_\eta)_{kn} = \frac{\partial f_k}{\partial \eta_n}, \tag{4}$$

$$(F_\xi)_{kj} = \frac{\partial f_k}{\partial \xi_j}. \tag{5}$$

$$\tag{6}$$

Therefore, the equations to be solved are (after dropping the factors of 2):

$$\vec{0} = V^{-1} \cdot (\vec{\eta} - \vec{y}) + \vec{F}_\eta^T \cdot \vec{\lambda}, \tag{7}$$

$$\vec{0} = \vec{F}_\xi^T \cdot \vec{\lambda}, \tag{8}$$

$$\vec{0} = \vec{f}(\vec{\eta}, \vec{\xi}). \tag{9}$$

Since the constraints $\vec{f}(\vec{\eta}, \vec{\xi})$ and their derivatives F_η and F_ξ are in general nonlinear functions, this system of equations has to be solved iteratively.

Let $\vec{\eta}^\nu$ and $\vec{\xi}^\nu$ denote the values at iteration ν. Then we can make a Taylor expansion around this point, and write (neglecting terms of 2nd and higher order)

$$\vec{f}(\vec{\eta}^{\nu+1}, \vec{\xi}^{\nu+1}) = f(\vec{\eta}^\nu, \vec{\xi}^\nu) + F_\eta^\nu \cdot (\vec{\eta}^{\nu+1} - \vec{\eta}^\nu) + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu). \tag{10}$$

Now Eqs. (7) to (9) read

$$\vec{0} = V^{-1} \cdot (\vec{\eta}^{\nu+1} - \vec{y}) + (F_\eta^\nu)^T \cdot \vec{\lambda}^{\nu+1}, \tag{11}$$

$$\vec{0} = (F_\xi^\nu)^T \cdot \vec{\lambda}^{\nu+1}, \tag{12}$$

$$\vec{0} = \vec{f}^\nu + F_\eta^\nu \cdot (\vec{\eta}^{\nu+1} - \vec{\eta}^\nu) + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu). \tag{13}$$

This system of equations is now solved.

One can solve Eq. (11) for $\vec{\eta}^{\nu+1}$:

$$\vec{\eta}^{\nu+1} = \vec{y} - V \cdot (F_\eta^\nu)^T \cdot \vec{\lambda}^{\nu+1} \tag{14}$$

and insert that into Eq. (13) to get

$$\vec{0} = \vec{f}^\nu + F_\eta^\nu \cdot \left(\vec{y} - V \cdot (F_\eta^\nu)^T \cdot \vec{\lambda}^{\nu+1} - \vec{\eta}^\nu \right) + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu), \tag{15}$$

which can be rewritten as

$$\vec{r} + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu) = S \cdot \vec{\lambda}^{\nu+1}, \tag{16}$$

where we have introduced

$$\vec{r} = \vec{f}^\nu + F_\eta^\nu \cdot (\vec{y} - \vec{\eta}^\nu), \tag{17}$$

$$S = F_\eta^\nu \cdot V \cdot (F_\eta^\nu)^T. \tag{18}$$

S is a symmetric $K \times K$ matrix that can be inverted, *provided that each constraint depends on at least one measured parameter.* Since V is symmetric and positive definite, also S is symmetric and positive definite.

However, the definition of S in Eq. (18) leads to a singular matrix if some constraints do not depend on any measured parameter. This is the case because for such a constraint k we have $(F_\eta)_{kn} = 0$ for all $n = 1...N$, and hence $S_{kk'} = \sum_{n,n'} (F_\eta)_{kn} V_{n,n'} (F_\eta)_{k'n'} = 0$ for this particular value of k.

The reason for this problem is that in our case Eqs. (11) and (13) alone are not sufficient to determine $\vec{\lambda}^{\nu+1}$.

Therefore, we have to use Eq. (12) as well. We can take Eq. (12), which is a set of J equations, and multiply it by F_ξ^ν (which is a $K \times J$ matrix) to arrive at this set of K equations:

$$\vec{0} = F_\xi^\nu \cdot (F_\xi^\nu)^T \cdot \vec{\lambda}^{\nu+1}. \tag{19}$$

We subtract this from Eq. (15) to get

$$\vec{0} = \vec{f}^\nu + F_\eta^\nu \cdot \left(\vec{y} - V \cdot (F_\eta^\nu)^T \cdot \vec{\lambda}^{\nu+1} - \vec{\eta}^\nu \right) + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu) - F_\xi^\nu \cdot (F_\xi^\nu)^T \cdot \vec{\lambda}^{\nu+1}. \tag{20}$$

As before, we can rewrite this in the form given by Eq. 16, only now S is given by

$$S = F_\eta^\nu \cdot V \cdot (F_\eta^\nu)^T + F_\xi^\nu \cdot (F_\xi^\nu)^T. \tag{21}$$

The new term $F_\xi^\nu \cdot (F_\xi^\nu)^T$ is evidently symmetric and, because it is the product of a matrix with its transpose, it is positive semidefinite, which means it has only nonnegative eigenvalues. Therefore, S is still symmetric and positive semidefinite. If the matrix S is still singular, the problem is ill defined.

For the same reason, also $W_1 = (F_\xi^\nu)^T \cdot S^{-1} \cdot F_\xi^\nu$ is positive semidefinite.

Therefore we can multiply Eq. (16) with S^{-1} to get

$$S^{-1} \cdot \left(\vec{r} + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu) \right) = \vec{\lambda}^{\nu+1}, \tag{22}$$

which inserted in Eq. (12) yields

$$\vec{0} = (F_\xi^\nu)^T \cdot S^{-1} \cdot \left(\vec{r} + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^\nu) \right). \tag{23}$$

This we can solve for $\vec{\xi}^{\nu+1} - \vec{\xi}^{\nu}$ and insert the result back into Eq. (16). Finally, we arrive at this set of equations:

$$\vec{\xi}^{\nu+1} = \vec{\xi}^{\nu} - \left((F_\xi^\nu)^T \cdot S^{-1} \cdot F_\xi^\nu\right)^{-1} \cdot (F_\xi^\nu)^T \cdot S^{-1} \cdot \vec{r}, \tag{24}$$

$$\lambda^{\nu+1} = S^{-1} \cdot \left(\vec{r} + F_\xi^\nu \cdot (\xi^{\nu+1} - \xi^\nu)\right), \tag{25}$$

$$\eta^{\nu+1} = \vec{y} - V \cdot (F_\eta^\nu)^T \cdot \vec{\lambda}^{\nu+1}. \tag{26}$$

Since S is symmetric and positive definite for well-posed problems, so is S^{-1}, and therefore also $W_1 = (F_\xi^\nu)^T \cdot S^{-1} \cdot F_\xi^\nu$ is symmetric and positive definite.

3.2 Description of the Code

Here, we describe the implementation in `OPALFitter::fit()`.

Before the first iteration, in `OPALFitter::initialize()` some initializations are performed. All fit objects are asked for their number of measured and unmeasured parameters, and are told the global parameter numbers. Also the constraints are counted. At the end of `OPALFitter::initialize()`, the variables `npar`, `nmea`, `nunm` and `ncon` correspond to correspond to $N+J$, N, J, and K, respectively.

Before the iterations start, the derivatives $\partial\vec{f}/\partial\vec{\eta}$ and $\partial\vec{f}/\partial\vec{\xi}$ are evaluated and stored in `dfeta`. `dfeta[k][n]` then corresponds to $\partial f_k/\partial\eta_n = (F_\eta)_{kn}$ for $k = 0...K-1$, $n = 0...N-1$, and $\partial f_k/\partial\xi_j = (F_\eta)_{kN+j}$ for $j = 0...J-1$. Also, vector `y` is filled with the measured values y_n ($n = 0...N-1$). Additionally, the variable `eta` is initialized. It holds in `eta[n]` the current fit values of the measured parameters η_n, and in `eta[nmea+j]` the values of the unmeasured parameters ξ_j.

During each iteration, the global covariance matrix V is collected from the fit objects in matrix `V`. The fact that this is repeated in each iteration makes it possible to use a generalized χ^2 that cannot be expressed through a fixed covariance matrix. Then the inverse V^{-1} is calculated as `VINV`.

Now the vector $\vec{r}$ is collected in variable `R`. Its calculation according to Eq. (17) is straightforward. At the same time, the matrix S is built (in variable `S`) according to Eq. (18). It is then inverted, so that hence `S` corresponds to S^{-1}.

Then, a matrix `W1` is formed as $(F_\xi^\nu)^T \cdot S^{-1} \cdot F_\xi^\nu$ and inverted, so that it corresponds to $\left((F_\xi^\nu)^T \cdot S^{-1} \cdot F_\xi^\nu\right)^{-1}$.

Using this, the vector `dxi` corresponding to $\xi^{\nu+1} - \xi^\nu$ is calculated as

$$\delta\xi = \xi^{\nu+1} - \xi^\nu = -\alpha\left((F_\xi^\nu)^T \cdot S^{-1} \cdot F_\xi^\nu\right)^{-1} \cdot (F_\xi^\nu)^T \cdot S^{-1} \cdot \vec{r} \tag{27}$$

$$= -\alpha W_1 \cdot (F_\xi^\nu)^T \cdot S^{-1} \cdot \vec{r} \tag{28}$$

The quantity α (variable `alph`) is initially set to 1 and can be used to decrease the step size if bad convergence is observed.

Then, new values of the unmeasured quantities ξ_j are calculated as $\xi_j^{\nu+1} = \xi_j^{\nu} + \delta\xi_j$.

Then the Lagrange multipliers are calculated in vector `alam` according to

$$\lambda^{\nu+1} = S^{-1} \cdot \left(\vec{r} + F_{\xi}^{\nu} \cdot \delta\xi\right) .$$

Now, new values for the fitted parameters are calculated according to Eq. (26).

Next, the derivatives $\partial\vec{f}/\partial\vec{\eta}$ and $\partial\vec{f}/\partial\vec{\xi}$ are reevaluated and stored again in `dfeta`.

3.3 Convergence criteria and step size reduction

In order to establish convergence, the `OPALFitter` evaluates two quantities:

$$\begin{aligned} \chi^2 &= (\vec{y} - \vec{\eta})^T V^{-1} (\vec{y} - \vec{\eta}) && (29) \\ \chi^2_{\mathrm{K}} &= 2 \cdot \sum_k |\lambda_{\mathrm{k}} f_{\mathrm{k}}| && (30) \end{aligned}$$

The `OPALFitter` assumes convergence if either of the following two conditions is fulfilled:

- χ_{K} changes by less than 10^{-3} within one iteration, and
  χ^2 changes by less than $10^{-4} \cdot \chi^2$ within one iteration, and
  $\chi_{\mathrm{K}} < 10^{-2} \cdot \chi^2$, or
- all constraint functions have values $f_{\mathrm{K}} < \epsilon$, and
  all parameters have changed by less than $\epsilon = 10^{-6}$ within the last iteration.

A bad step, which worsens the result instead of improving it, is detected by the following condition:

- χ_{K} increases by more than 5 % relative and by more than 10^{-10} absolute within the step, and
- $\chi_{\mathrm{K}} > 10^{-2} \cdot \chi^2$.

Each step is scaled by a factor α, which is initially set to $\alpha = 1$. If a bad step is detected, the step is repeated with a reduced value of α, which is set to half its old value. This value of α is kept for subsequent steps, but is increased by 0.1 after each successful step, until the value of $\alpha = 1$ is reached again.

4 Constraints

All classes implementing constraints are derived from the `BaseConstraint` class. Since the minimisation method of the `OPALFitter` is based on a first order Taylor expansion, their main task is to determine the value of the constraint function and its first derivatives with respect to the parameters.

Let us consider the two main classes of constraints in turn:

- energy and momentum constraints, and
- mass constraints.

4.1 Energy and Momentum Constraints

Energy and momentum constraints are used to express energy and momentum conservation in collisions.

In e^+e^- collisions, we'll have typically four constraints:

$$\begin{aligned}\sum_i p_{\mathrm{x,i}} &= 0 \\ \sum_i p_{\mathrm{y,i}} &= 0 \\ \sum_i p_{\mathrm{z,i}} &= 0 \\ \sum_i E_{\mathrm{i}} &= \sqrt{s},\end{aligned}$$

where $\sqrt{s}$ is the e^+e^- center-of-mass energy, and the sum will run over all final-state particles (including jets and neutrinos, and possibly (unmeasured) ISR photons).

All these constraints can be written as

$$f = \sum_i \left(c_{\mathrm{x}} p_{\mathrm{x,i}} + c_{\mathrm{y}} p_{\mathrm{y,i}} + c_{\mathrm{z}} p_{\mathrm{z,i}} + c_{\mathrm{e}} E_{\mathrm{i}} + d \right),$$

with suitable coefficients c_{x}, c_{y}, c_{z}, c_{e}, and d; the sum extends over those four-vecors that are subjected to the constraint.

Then we have (using $\frac{\partial f}{\partial p_{\mathrm{x,i}}} = c_{\mathrm{x}}$):

$$\frac{\partial f}{\partial \eta_{\mathrm{n}}} = \sum_i \left(c_{\mathrm{x}} \frac{\partial p_{\mathrm{x,i}}}{\partial \eta_{\mathrm{n}}} + c_{\mathrm{y}} \frac{\partial p_{\mathrm{y,i}}}{\partial \eta_{\mathrm{n}}} + c_{\mathrm{z}} \frac{\partial p_{\mathrm{z,i}}}{\partial \eta_{\mathrm{n}}} + c_{\mathrm{e}} \frac{\partial E_{\mathrm{i}}}{\partial \eta_{\mathrm{n}}} \right),$$

or, in case of unmeasured parameters:

$$\frac{\partial f}{\partial \xi_{\mathrm{j}}} = \sum_i \left(c_{\mathrm{x}} \frac{\partial p_{\mathrm{x,i}}}{\partial \xi_{\mathrm{j}}} + c_{\mathrm{y}} \frac{\partial p_{\mathrm{y,i}}}{\partial \xi_{\mathrm{j}}} + c_{\mathrm{z}} \frac{\partial p_{\mathrm{z,i}}}{\partial \xi_{\mathrm{j}}} + c_{\mathrm{e}} \frac{\partial E_{\mathrm{i}}}{\partial \xi_{\mathrm{j}}} \right).$$

The `EnergyMomentumConstraint` class can be used to implement a constraint on any linear combination of E, p_x, p_y and p_z by specifying the appropriate c factors in the constructor.

4.2 Mass Constraints

Mass constraints fall into two categories:

- The sum of several four-vecors has to have a definite invariant mass m:

$$f = \left(\sum_i E_\mathrm{i}\right)^2 - \left(\sum_i p_\mathrm{x,i}\right)^2 - \left(\sum_i p_\mathrm{y,i}\right)^2 - \left(\sum_i p_\mathrm{z,i}\right)^2 - m^2$$

- Equal mass constraint: Two sets of four-vectors have the same invariant mass:

$$\begin{aligned} f &= \left(\sum_i E_\mathrm{i}\right)^2 - \left(\sum_i p_\mathrm{x,i}\right)^2 - \left(\sum_i p_\mathrm{y,i}\right)^2 - \left(\sum_i p_\mathrm{z,i}\right)^2 \\ &\quad - \left(\sum_j E_\mathrm{j}\right)^2 + \left(\sum_j p_\mathrm{x,j}\right)^2 + \left(\sum_j p_\mathrm{y,j}\right)^2 + \left(\sum_j p_\mathrm{z,j}\right)^2 \end{aligned}$$

These two types can be summarized in one type of constraint:

$$\begin{aligned} f &= \left(\sum_i E_\mathrm{i}\right)^2 - \left(\sum_i p_\mathrm{x,i}\right)^2 - \left(\sum_i p_\mathrm{y,i}\right)^2 - \left(\sum_i p_\mathrm{z,i}\right)^2 \\ &\quad - \left(\sum_j E_\mathrm{j}\right)^2 + \left(\sum_j p_\mathrm{x,j}\right)^2 + \left(\sum_j p_\mathrm{y,j}\right)^2 + \left(\sum_j p_\mathrm{z,j}\right)^2 \\ &\quad -m^2. \end{aligned}$$

Then we get[1]

$$\begin{aligned} \frac{\partial f}{\partial \eta_\mathrm{n}} &= 2\Sigma E_\mathrm{i} \cdot \left(\sum_i \frac{\partial E_\mathrm{i}}{\partial \eta_\mathrm{n}}\right) - 2\Sigma p_\mathrm{x,i} \cdot \left(\sum_i \frac{\partial p_\mathrm{x,i}}{\partial \eta_\mathrm{n}}\right) - 2\Sigma p_\mathrm{y,i} \cdot \left(\sum_i \frac{\partial p_\mathrm{y,i}}{\partial \eta_\mathrm{n}}\right) - 2\Sigma p_\mathrm{z,i} \cdot \left(\sum_i \frac{\partial p_\mathrm{z,i}}{\partial \eta_\mathrm{n}}\right) \\ &\quad -2\Sigma E_\mathrm{j} \cdot \left(\sum_j \frac{\partial E_\mathrm{j}}{\partial \eta_\mathrm{n}}\right) + 2\Sigma p_\mathrm{x,j} \cdot \left(\sum_j \frac{\partial p_\mathrm{x,j}}{\partial \eta_\mathrm{n}}\right) + 2\Sigma p_\mathrm{y,j} \cdot \left(\sum_j \frac{\partial p_\mathrm{y,j}}{\partial \eta_\mathrm{n}}\right) + 2\Sigma p_\mathrm{z,j} \cdot \left(\sum_j \frac{\partial p_\mathrm{z,j}}{\partial \eta_\mathrm{n}}\right) \end{aligned}$$

4.3 Global Derivative Matrix of the Constraints

For reasons of efficiency, each constraint adds its own contribution to a global matrix $F_{\xi\eta}$ of dimension $K \times (N + J)$, defined as:

[1] of course there are the analoguos expression for the unmeasured quantities, which are not explicitely listed here, since the partial derivatives do not depend on the parameters being measured or unmeasured.

$$F_{\xi\eta} = \begin{pmatrix} \cdots & \frac{\partial f_\mathrm{k}}{\partial \eta_1} & \cdots \\ \cdots & & \cdots \\ \cdots & \frac{\partial f_\mathrm{k}}{\partial \eta_\mathrm{N}} & \cdots \\ \cdots & \frac{\partial f_\mathrm{k}}{\partial \xi} & \cdots \\ \cdots & & \cdots \\ \cdots & \frac{\partial f_\mathrm{k}}{\partial \xi_\mathrm{J}} & \cdots \end{pmatrix} .$$

This is implemented in the method `AddToGlobalDerivativeMatrix`.

5 Fit Objects

All classes implementing fit objects for kinematic fitting are derived from the `ParticleFitObject` class, which in turn is derived from the `BaseFitObject` class.

These base classes define the minimal functionality any fit object must provide. The main task of a fit object, as define by `BaseFitObject`, is to keep parameters (and errors) that define the physical quantities and encapsulate the actually chosen parametrisation from the rest of the fitting machinery. The `ParticleFitObject` provides as physical quantities the four-vector of a particle. For other applications it is also possible to define more general interfaces, which e.g. provide also vertex positions or trajectory information, as needed for decay chain fits.

Since for the fit a parametrisation where the distribution of the measured values is as close to a normal distribution as possible is most favorable, different kinds of particles (implying different kinds of measurements!) might require different parametrisations. For each desired parametrisation a concrete class should be derived from this abstract base class. It needs to be able to convert its parameters to E, p_x, p_y, p_z and to provide the derivatives of E, p_x, p_y, p_z with respect to the internal parameters.

Depending on the type of particle, some or all parameters might be unmeasured (neutrinos!). They are treated differently by the fit algorithm and are thus flagged accordingly.

In order to insert its derivatives into the global covariance matrix of all FitObjects in the event, each `FitObject` needs to know the position of its parameters in the overall parameter list.

Currently, a `JetFitObject` parametrising a (measured) jet via its energy, polar and azimutal angle as well as a `NeutrinoFitObject` parametrising missing momentum via p_x, p_y and p_z as unmeasured quantities are implemented [2]. The errors of the measured parameters have to be supplied by the user along with the measured values.

[2] In addition, a dedicated `PhotonFitObject` for treating ISR and Beamstrahlung photons escaping through the beam pipe is under development [6]

5.1 Implementation of new FitObjects

Depending on the experimental conditions, the measurement of a particle's or jet's fourvector is parametrized differently. Typically, one will choose a parametrization in terms of variables which are uncorrelated as far as possible, and which have approximately Gaussian errors.

If a parametrization in terms of a new set of variables is needed, the user has to create a new class derived from `ParticleFitObject`.

This new class will generally have its own implementation of the following methods:

- `getChi2` returns the χ^2 of the fit parameters w.r.t. the measured ones. For this method, a default implementation is provided in the base class `ParticleFitObject`; however, it may be necessary to provide a special implementation which takes care of cases where e.g. azimuthal angles are involved.
- `getE`, `getPx`, `getPy`, and `getPz` return the energy and the three-momentum components of the FitObject.
- `getDE`, `getDPx`, `getDPy`, and `getDPz` return the derivative of the energy and the three-momentum components of the FitObject w.r.t. one of the internal parameters.
- `addTo1stDerivatives` adds the first derivatives

  $$c_{\mathrm{e}} \frac{\partial E}{\partial \eta_{\mathrm{i}}} + c_{\mathrm{x}} \frac{\partial p_{\mathrm{x}}}{\partial \eta_{\mathrm{i}}} + c_{\mathrm{y}} \frac{\partial p_{\mathrm{y}}}{\partial \eta_{\mathrm{i}}} + c_{\mathrm{z}} \frac{\partial p_{\mathrm{z}}}{\partial \eta_{\mathrm{i}}}$$

  w.r.t. each of its parameters η_{i} to a global derivative matrix; the coefficients $c_{\mathrm{e}}, c_{\mathrm{x}}, c_{\mathrm{y}}$ and c_{z} are provided (in this order) in a single array as an argument. This method is in principle redundant, because `getDE`, `getDPx`, `getDPy`, and `getDPz` also have to be defined; however, for efficiency reasons this method has to be provided.
- `addToGlobalChi2DerVector` adds the λ times the first derivatives

  $$\lambda \cdot \left(c_{\mathrm{e}} \frac{\partial E}{\partial \eta_{\mathrm{i}}} + c_{\mathrm{x}} \frac{\partial p_{\mathrm{x}}}{\partial \eta_{\mathrm{i}}} + c_{\mathrm{y}} \frac{\partial p_{\mathrm{y}}}{\partial \eta_{\mathrm{i}}} + c_{\mathrm{z}} \frac{\partial p_{\mathrm{z}}}{\partial \eta_{\mathrm{i}}} \right)$$

  w.r.t. each of its parameters η_{i} to a global derivative matrix; the coefficients $c_{\mathrm{e}}, c_{\mathrm{x}}, c_{\mathrm{y}}$ and c_{z} are provided (in this order) in a single array as an argument, also λ is provided. Again, this method is provided for efficiency reasons.
- `getError2` returns the value of

  $$\left(\frac{\partial f_{\mathrm{k}}}{\partial E, p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}}} \right)^T \cdot \left(\frac{\partial E, p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}}}{\partial \vec{\eta}} \right)^T \cdot V^{-1} \cdot \left(\frac{\partial E, p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}}}{\partial \vec{\eta}} \right) \left(\frac{\partial f_{\mathrm{k}}}{\partial E, p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}}} \right),$$

where the vector $\left(\frac{\partial f_\mathrm{k}}{\partial E, p_\mathrm{x}, p_\mathrm{y}, p_\mathrm{z}}\right)$ is provided as an argument, and V^{-1} is the covariance matrix of the local parameters $\vec{\eta}$. This method is needed to perform an error propagation for the value of a constraint function.

For efficiency reasons, a method `invalidateCache` is called whenever a parameter is changed. This mechanism may be used to avoid repeated calculations of various quantities during an iteration step. The user may look at the implementation of class `JetFitObject` for an example how to use such a caching mechanism.

A number of additional methods are needed for the NewtonFitter fit engine, which also needs the second derivatives of the constraints and hence of the FitObject's fourvectors. These are called `addTo2ndDerivatives`. Again, the user may look at the implementation of class `JetFitObject` for an example how to implement this method, which is, however, not needed by the reference fit engine `OPALFitter`.

6 Example

The performance of the fit has been tested on $e^+e^- \to u\bar{d}d\bar{u}$ events generated by the matrix element based event generator Whizard [7] and passed through the full detector simulation and reconstruction chain of the ILD detector [8] proposed for the International Linear Collider. After forcing the event into four jets using the Durham jet algorithm, applying a few basic reconstruction quality cuts and correcting the overall jet energy scale by 1%, a kinematic fit with the following five constraints is performed:

- $\sum_{i=1}^{4} E_i = 500$ GeV
- $\sum_{i=1}^{4} p_{x,i} = 0$ GeV, $\sum_{i=1}^{4} p_{y,i} = 0$ GeV, $\sum_{i=1}^{4} p_{z,i} = 0$ GeV
- $M(j_1, j_2) = M(j_3, j_4)$,

where $(E_i, p_{x,i}, p_{y,i}, p_{z,i})$ denotes the four-vector of jet i and $M(j_1, j_2)$ and $M(j_3, j_4)$ are the two invariant di-jet masses in each event. Since there are three possibilities to combine the four jets into two di-jet systems, the fit is repeated three times per event with a different jet permutation.

Figure 1 shows the di-jet masses for the permutation which yields the highest fit probability. The shaded histogram shows the mass distribution obtained using the measured quantities, while the hatched histogram is obtained from the fitted momenta. The histograms are fitted with the sum of two non-relativistic Breit-Wigner functions for the W^+W^- and the ZZ contribution which are folded with a Gaussian resolution. The Z mass and width as well as the W width are have been fixed to their literature values, while the W mass m_W ,

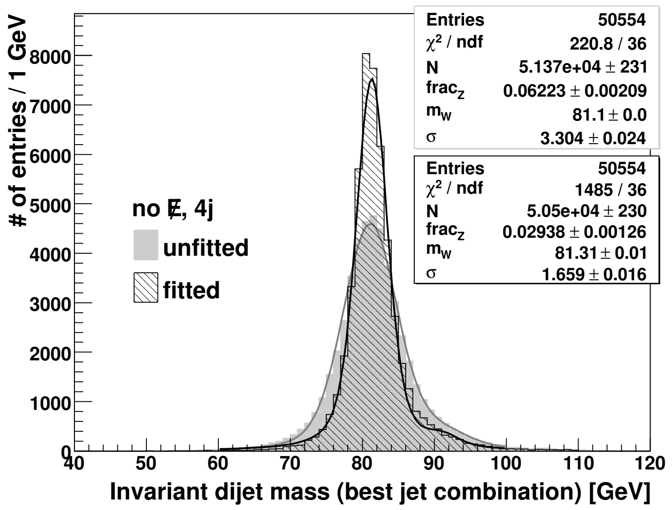

Figure 1: Invariant di-jet mass without and with kinematic fit (plot: courtesy Moritz Beckmann [9]).

the overall normalisation N, the fraction of ZZ events frac_Z and width σ of the Gaussian are free parameters. It can clearly be seen that after kinematic fitting the width of the Gaussian is significantly reduced by nearly a factor two from $\sigma = 3.3$ GeV to $\sigma = 1.7$ GeV.

Figure 1 contains only events where less than 5 GeV of the total energy is lost due to ISR and Beamstrahlung. Similar results can be obtained in the presence of significant photon radiation when the photon(s) are taken into account in the fit. Preliminary results of the fit including photon radiation can be found in [6], a detailed publication is in preparation.

7 Summary

The MarlinKinfit package provides an object–oriented framework for constrained kinematic fitting. It is based on the three main abstraction of fit objects, constraints and a fit engine. Thanks to generic interfaces, fit objects and constraints can be combined in a flexible way to describe any desired event hypothesis, allowing in addition an indepent exchange of the fit engine. The reference fit engine of the package has been described in detail in this note and an example of the performance on four-jet events without significant photon radiation is given, where a fit requiring energy and momentum conservation with respect to the initial state plus two equal di-jet masses improves the mass resolution by nearly a factor of two.

MarlinKinfit contains additional classes which are still under development, for instance a second fit engine, based on the Newton–Raphson method, which can also handle soft constraints in the presence of unmeasured parameters, as well as a photon fit object taking into account ISR and Beamstrahlung photons. These will be described in future notes.

References

[1] F. James: "MINUIT. Function Minimization and Error Analysis. Reference Manual, Version 94.1." *CERN Program Library Long Writeup D506,* *http://wwwasdoc.web.cern.ch/wwwasdoc/minuit/minmain.html.*

[2] F. Gaede, "Marlin and LCCD: Software tools for the ILC," Nucl. Instrum. Meth. A **559** (2006) 177.

[3] `http://www-zeuthen.desy.de/lc-cgi-bin/cvsweb.cgi/MarlinReco/?cvsroot=marlinreco`

[4] Frodesen, Skjeggestad, Tøfte: "Probability and Statistics in Particle Physics", Chap.10.8.

[5] Louis Lyons: "Statistics for nuclear and particle physics." *226 pp., Cambridge (Cambridge Univ. Press) 1986.*

[6] J. List, M. Beckmann and B. List, "Kinematic Fitting in the Presence of ISR at the ILC," arXiv:0901.4656 [hep-ex].

[7] W. Kilian, T. Ohl and J. Reuter, "WHIZARD: Simulating Multi-Particle Processes at LHC and ILC," arXiv:0708.4233 [hep-ph].

[8] ILD Detector Concept Study Group, "ILD Letter of Intent", to be submitted (2009).

[9] M. Beckmann, diploma thesis in preparation.