(Marlin)Kinfit: Kinematic Fitting for the ILC

Benno List, Jenny List

ILD Detector Optimization WG Phone Meeting
20.2.2008

- Introduction: Kinematic fitting and the method of Lagrange multipliers
- The OPALFitter
- NewtonFitter: A new fitting engine
- Toy Monte Carlo Studies

Introduction

- Kinematic fitting: minimize a χ^2 under constraints
  => method of Lagrange multipliers (MINUIT not applicable)

Our work:

- Develop an object oriented software framework for kinematic fits:
  Fitter engine - Constraints - Fit objects
- Develop a new fitter engine: NewtonFitter
  => Solves kinematic fit problems with
  - Unmeasured quantities (Neutr(al)ino)
  - Hard constraints ($\Sigma p_x = 0$)
  - Additional “soft” constraints, i.e. additional χ^2 terms: $\chi^2 = (m-m_0)/\sigma^2$
    => needed if natural width of particles starts to be resolved by detector

The Method of Lagrange Multipliers

N measured parameters $\vec{\eta}$ Measured values $\vec{y}$, covariance matrix V

J unmeasured quantities $\vec{\xi}$

K constraint funtions $\vec{f}(\vec{\eta}, \vec{\xi})$

The usual χ^2 The constraints

The total χ^2_T: $\chi^2_T(\vec{\eta}, \vec{\xi}, \vec{\lambda}) = (\vec{y} - \vec{\eta})^T \cdot V^{-1} \cdot (\vec{y} - \vec{\eta}) + 2\vec{\lambda}^T \cdot \vec{f}(\vec{\eta}, \vec{\xi}).$

For minimum: Seek values where all derivatives vanish:

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

$$(F_\eta)_{kn} = \frac{\partial f_k}{\partial \eta_n} \quad (K \times N\text{matrix}) \qquad (F_\xi)_{kj} = \frac{\partial f_k}{\partial \xi_j} \quad (J \times N\text{matrix})$$

Solve this nonlinear set of equations:

$$\begin{aligned}
\vec{0} &= V^{-1} \cdot (\vec{\eta} - \vec{y}) + \vec{F}_\eta^T \cdot \vec{\lambda} \\
\vec{0} &= \vec{F}_\xi^T \cdot \vec{\lambda} \\
\vec{0} &= \vec{f}(\vec{\eta}, \vec{\xi})
\end{aligned}$$

The OPAL Fitter Method

The equations to solve:

$$\begin{aligned} \vec{0} &= V^{-1} \cdot (\vec{\eta} - \vec{y}) + \vec{F}_\eta^T \cdot \vec{\lambda} \\ \vec{0} &= \vec{F}_\xi^T \cdot \vec{\lambda} \\ \vec{0} &= \vec{f}(\vec{\eta}, \vec{\xi}) \end{aligned}$$

$$(F_\eta)_{kn} = \frac{\partial f_k}{\partial \eta_n} \quad (K \times N \text{matrix})$$

$$(F_\xi)_{kj} = \frac{\partial f_k}{\partial \xi_j} \quad (J \times N \text{matrix})$$

For iterative solution: Taylor-expansion of the constraints:

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

For each iteration, solve this linear system

$$\begin{aligned} \vec{0} &= V^{-1} \cdot (\vec{\eta}^{\nu+1} - \vec{y}) + (F_\eta^\nu)^T \cdot \vec{\lambda}^{\nu+1}, \\ \vec{0} &= (F_\xi^\nu)^T \cdot \vec{\lambda}^{\nu+1}, \\ \vec{0} &= \vec{f}^\nu + F_\eta^\nu \cdot (\vec{\eta}^{\nu+1} - \vec{\eta}^{\nu}) + F_\xi^\nu \cdot (\vec{\xi}^{\nu+1} - \vec{\xi}^{\nu}). \end{aligned}$$

In matrix form:

$$\begin{pmatrix} V^{-1} \cdot \vec{y} \\ \vec{0} \\ -\vec{f}^\nu + F_\eta^\nu \vec{\eta}^\nu + F_\xi^\nu \cdot \vec{\xi}^\nu \end{pmatrix} = \begin{pmatrix} V^{-1} & 0 & (F_\eta^\nu)^T \\ 0 & 0 & (F_\xi^\nu)^T \\ F_\eta^\nu & F_\xi^\nu & 0 \end{pmatrix} \cdot \begin{pmatrix} \eta^{\nu+1} \\ \vec{\xi}^{\nu+1} \\ \vec{\lambda}^{\nu+1} \end{pmatrix}$$

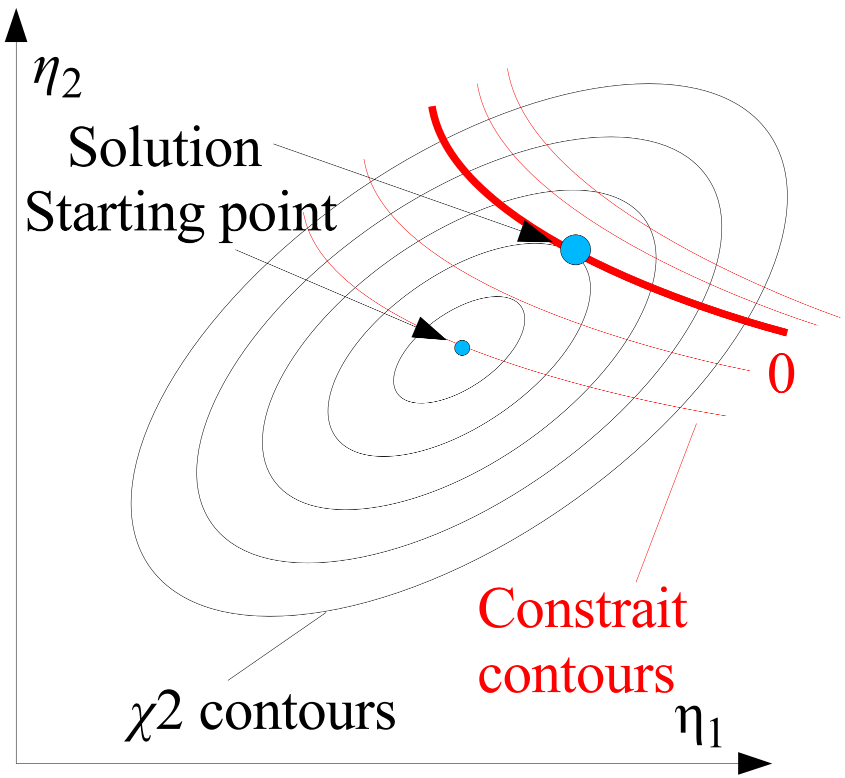
See L. Lyons: *Statistics for nuclear and particle physics,* Cambridge Univ. Press 1986.

How the OPALFitter works

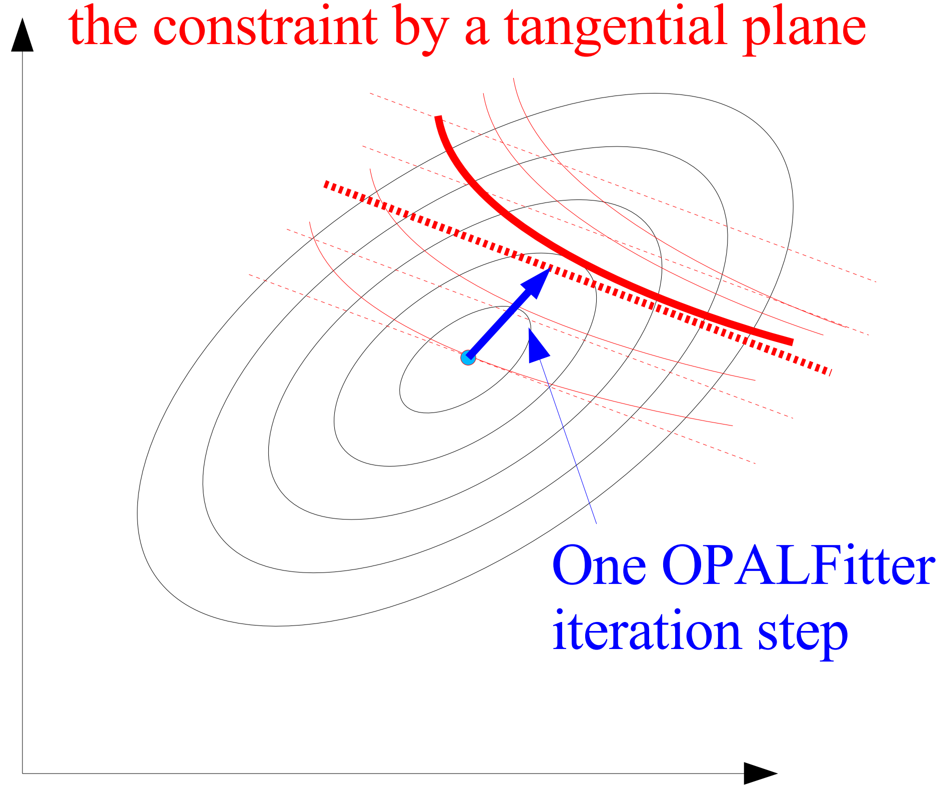
$$\begin{aligned}\vec{0} &= V^{-1}\cdot(\vec{\eta}-\vec{y}) + \vec{F}_{\eta}^{T}\cdot\vec{\lambda}\\ \vec{0} &= \vec{F}_{\xi}^{T}\cdot\vec{\lambda}\\ \vec{0} &= \vec{f}\,(\vec{\eta},\vec{\xi})\end{aligned}$$

The constraint line must be parallel to the χ^2 contours at the solution

The solution must lie on the 0-contour of the constraint



The OPALFitter approximates the constraint by a tangential plane



The Software

Three basic concepts:

- The Fitter Engine:
  - Sets up the system of equations and solves it
  - Administrates lists of constraints and fit objects
- The Constraint:
  - Takes 4-vectors of fit objects to calculate its own value
  - Can calculate its own derivatives w.r.t. the 4-vector components of the fit objects
- The Fit Object:
  - Encapsulates all details of the parametrization (number of parameters, parametrization)
  - Can calculate its own contribution to the global χ^2 and its derivatives
  - Can calculate the derivatives of 4-vector components w.r.t. all parameters

What Do We Need?

- Parameters (measured and unmeasured), measured values and covariances
  => stored locally in FitObjects
- (inverse) global covariance matrix: can be built from local covariance matrices (stored in FitObjects)
- Values of constraint functions
  => ConstraintObjects
- Constraints typically expressed in terms of 4-vector-components
  => get them from FitObjects
- Derivatives of constraints w.r.t. all parameters:
  Use chain-rule:
  - Constraint provides derivatives w.r.t. 4-vector components
  - FitObject provides derivatives of 4-vector components w.r.t. parameters

Sketch of the Fit Procedure

- Fitter has a list of FitObjects;
  each FitObject knows its own nuber of parameters and whether they are measured
  => Fitter assigns global parameter numbers to all parameters of FitObjects

- Fitter has a list of ConstraintObjects
  => assigns global numbers to them

- Fitter builds up system of equations:
  - resets vector and matrix to 0
  - asks FitObjects to add their parts
  - asks ConstraintObjects to add their parts

From FitObjects

$$\begin{pmatrix} V^{-1}\cdot\vec{y} \\ \vec{0} \\ -\vec{f}^{\nu} + F^{\nu}_{\eta}\vec{\eta}^{\nu} + F^{\nu}_{\xi}\cdot\vec{\xi}^{\nu} \end{pmatrix} = \begin{pmatrix} V^{-1} & 0 & (F^{\nu}_{\eta})^T \\ 0 & 0 & (F^{\nu}_{\xi})^T \\ F^{\nu}_{\eta} & F^{\nu}_{\xi} & 0 \end{pmatrix} \cdot \begin{pmatrix} \eta^{\nu+1} \\ \vec{\xi}^{\nu+1} \\ \vec{\lambda}^{\nu+1} \end{pmatrix}$$

From ConstraintObjects

- Fitter solves system of equations and updates parameters of FitObjects

- Fitter checks for convergence (Parameter changes small, constraints fulfilled), iterates if necessary

The User Code

Create FitObjects (2 jets)

Create Constraints:
$\Sigma p_x = 0$,
$\Sigma p_y = 0$,
Invariant mass = 90GeV

Tell constraints over which FitObjects they should sum

Create the Fitter Engine

Tell the Fitter which Objects are to be fitted, and which Constraints are to be observed

Perform the Fit

```
//                       E    theta phi    dE dtheta dphi mass
JetFitObject jet1 (44., 1.2, 0.087, 5.0, 0.2,  0.1, 0.);
JetFitObject jet2 (46., 1.8, 3.120, 5.0, 0.2,  0.1, 0.);

// Constraint 0*sum(E) + 1*sum(px) + 0*sum(py) + 0*sum(pz) = 0
MomentumConstraint pxconstraint (0, 1, 0, 0, 0);
pxconstraint.addToFOList (jet1);
pxconstraint.addToFOList (jet2);

// Constraint 0*sum(E) + 0*sum(px) + 1*sum(py) + 0*sum(pz) = 0
MomentumConstraint pyconstraint (0, 0, 1, 0, 0);
pyconstraint.addToFOList (jet1);
pyconstraint.addToFOList (jet2);

// Constraint total mass = 90
MassConstraint mconstraint (90);
mconstraint.addToFOList (jet1);
mconstraint.addToFOList (jet2);

OPALFitter fitter;

fitter.addFitObject (jet1);
fitter.addFitObject (jet2);

fitter.addConstraint (pxconstraint);
fitter.addConstraint (pyconstraint);
fitter.addConstraint (mconstraint);

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

Advantages of the Software

- Fitter Engine decoupled from the rest
  => can try different algorithms
  (2 are implemented: OPALFitter and NewtonFitter)
- Constraints are decoupled from inner workings of FitObjects
- FitObject parametrization encapsulated:
  New Objects with different parametrization can be added easily
- Scheme can be extended for other problems: decay chains
  (constraints on 4-momenta and vertex positions)

A New Fitter Engine: NewtonFitter

- OPALFitter: Reference implementation, literal translation of FORTRAN code used in OPAL (WWFIT)
- Shortcomings of OPALFitter:
  - Does not use 2nd derivatives of constraints => could improve convergence
  - Difficult to extend to “soft constraints” (additional χ^2 terms)
- New approach: NewtonFitter

The Mathematics of the NewtonFitter

N parameters a_i, $i = 1 \ldots N$ Measured values $\vec{y}$, covarianve matrix V

K constraint funtions $\vec{f}\,(\vec{a})$

The total χ^2: $\chi^2_T\,(\vec{a}, \vec{\lambda}) = \chi^2\,(\vec{a}, \vec{y}) + \vec{\lambda}^T \cdot \vec{f}\,(\vec{a}).$

Seek stationary point, where all derivatives vanish:

$$
\begin{aligned}
\nabla_a \chi^2_T &= \nabla_a \chi^2 + \vec{\lambda}^T \cdot \nabla_a \vec{f}\,(\vec{a}) = \vec{0}, && (N \text{ equations}) \\
\nabla_\lambda \chi^2_T &= \vec{f}\,(\vec{a}) = \vec{0}, && (K \text{ equations})
\end{aligned}
\qquad
\begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{\partial \chi^2_T}{\partial \vec{a}} \\ \frac{\partial \chi^2_T}{\partial \vec{\lambda}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \chi^2}{\partial a_i} + \sum_k \lambda_k \cdot \frac{\partial f_k}{\partial a_i} \\ f_k \end{pmatrix}
$$

Newton-Raphson iterative method to solve *y*(*x*)=0:

$$
x^{\nu+1} = x^\nu - \frac{y\,(x^\nu)}{y'\,(x^\nu} \quad \text{=> solve} \quad y'\,(x^\nu) \cdot (x^\nu - x^{\nu+1}) = y\,(x^\nu)
$$

Here: Solve this system of equations in each step:

$$
\left(\begin{array}{ccc|ccc}
\frac{\partial^2 \chi^2}{\partial a_1 \partial a_1} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_1 \partial a_1} & \ldots & \frac{\partial^2 \chi^2}{\partial a_1 \partial a_N} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_1 \partial a_N} & \frac{\partial f_1}{\partial a_1} & \ldots & \frac{\partial f_K}{\partial a_1} \\
\ldots & & \ldots & \ldots & & \ldots \\
\frac{\partial \chi^2}{\partial a_N \partial a_1} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_N \partial a_1} & \ldots & \frac{\partial^2 \chi^2}{\partial a_N \partial a_N} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_N \partial a_N} & \frac{\partial f_1}{\partial a_N} & \ldots & \frac{\partial f_K}{\partial a_N} \\
\hline
\frac{\partial f_1}{\partial a_1} & \ldots & \frac{\partial f_1}{\partial a_N} & 0 & \ldots & 0 \\
\ldots & & \ldots & \ldots & & \ldots \\
\frac{\partial f_K}{\partial a_1} & \ldots & \frac{\partial f_K}{\partial a_N} & 0 & \ldots & 0
\end{array}\right)
\cdot
\left(\begin{array}{c}
a_1^\nu - a_1^{\nu+1} \\
\ldots \\
a_N^\nu - a_N^{\nu+1} \\
\hline
\lambda_1^\nu - \lambda_1^{\nu+1} \\
\ldots \\
\lambda_K^\nu - \lambda_K^{\nu+1}
\end{array}\right)
=
\left(\begin{array}{c}
\frac{\partial \chi^2}{\partial a_1} + \lambda_k^\nu \cdot \frac{\partial f_k}{\partial a_1} \\
\ldots \\
\frac{\partial \chi^2}{\partial a_N} + \lambda_k^\nu \cdot \frac{\partial f_k}{\partial a_N} \\
\hline
f_1 \\
\ldots \\
f_K
\end{array}\right)
$$

Application of the Chain Rule

$$\lambda_k^\nu \frac{\partial^2 f_k}{\partial a_i \partial a_j} = \lambda_k^\nu \frac{\partial^2 f_k}{\partial P_{i'} \partial P_{j'}} \cdot \frac{\partial P_{i'}}{\partial a_i} \cdot \frac{\partial P_{j'}}{\partial a_j} + \lambda_k^\nu \frac{\partial f_k}{\partial P_{i'}} \cdot \frac{\partial P_{i'}^2}{\partial a_i \partial a_j}$$

$$\frac{\partial f_k}{\partial a_i} = \frac{\partial f_k}{\partial P_{i'}} \cdot \frac{\partial P_{i'}}{\partial a_i}$$

$$\frac{\partial^2 \chi^2}{\partial a_i \partial a_j} = 2\left(V^{-1}\right)_{ij}$$

Measured parameters only

$$\frac{\partial \chi^2}{\partial a_i} = 2\left(V^{-1}\right)_{ij}(a_j - y_j)$$

$$\left(\begin{array}{ccc|ccc}
\frac{\partial^2\chi^2}{\partial a_1\partial a_1} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_1 \partial a_1} & \dots & \frac{\partial^2\chi^2}{\partial a_1\partial a_N} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_1 \partial a_N} & \frac{\partial f_1}{\partial a_1} & \dots & \frac{\partial f_K}{\partial a_1} \\
\dots & & \dots & \dots & & \dots \\
\frac{\partial^2\chi^2}{\partial a_N\partial a_1} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_N \partial a_1} & \dots & \frac{\partial^2\chi^2}{\partial a_N\partial a_N} + \lambda_k \cdot \frac{\partial^2 f_k}{\partial a_N \partial a_N} & \frac{\partial f_1}{\partial a_N} & \dots & \frac{\partial f_K}{\partial a_N} \\
\hline
\frac{\partial f_1}{\partial a_1} & \dots & \frac{\partial f_1}{\partial a_N} & 0 & \dots & 0 \\
\dots & & \dots & \dots & & \dots \\
\frac{\partial f_K}{\partial a_1} & \dots & \frac{\partial f_K}{\partial a_N} & 0 & \dots & 0
\end{array}\right) \cdot \left(\begin{array}{c}
a_1^\nu - a_1^{\nu+1} \\ \dots \\ a_N^\nu - a_N^{\nu+1} \\ \hline \lambda_1^\nu - \lambda_1^{\nu+1} \\ \dots \\ \lambda_K^\nu - \lambda_K^{\nu+1}
\end{array}\right) = \left(\begin{array}{c}
\frac{\partial \chi^2}{\partial a_1} + \lambda_k^\nu \cdot \frac{\partial f_k}{\partial a_1} \\ \dots \\ \frac{\partial \chi^2}{\partial a_N} + \lambda_k^\nu \cdot \frac{\partial f_k}{\partial a_N} \\ \hline f_1 \\ \dots \\ f_K
\end{array}\right)$$

$$f_1$$

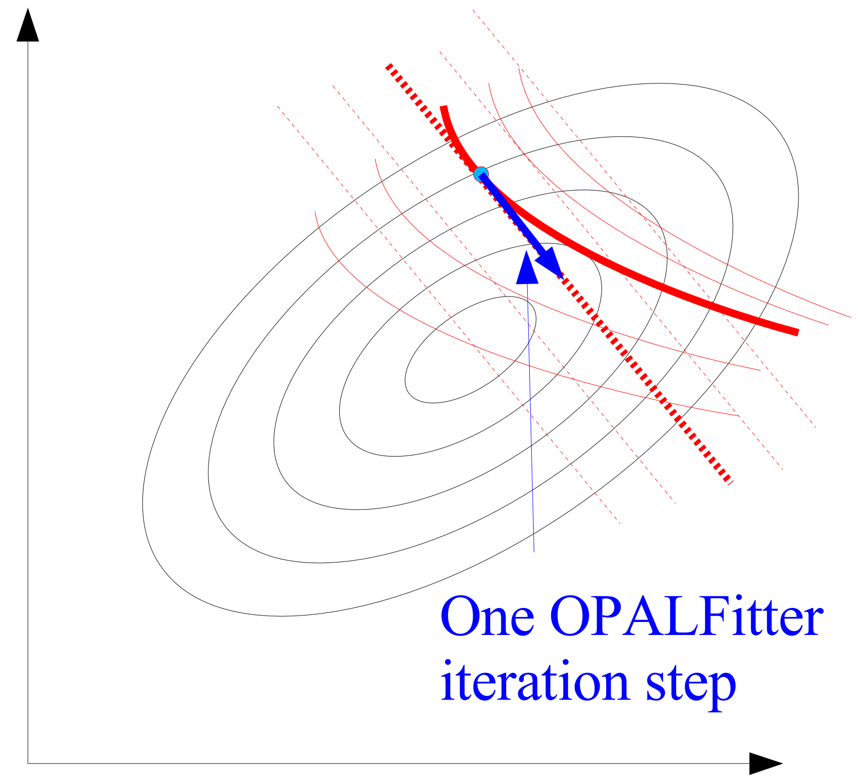
$$\frac{\partial f_k}{\partial a_i} = \frac{\partial f_k}{\partial P_{i'}} \cdot \frac{\partial P_{i'}}{\partial a_i}$$

We need only:

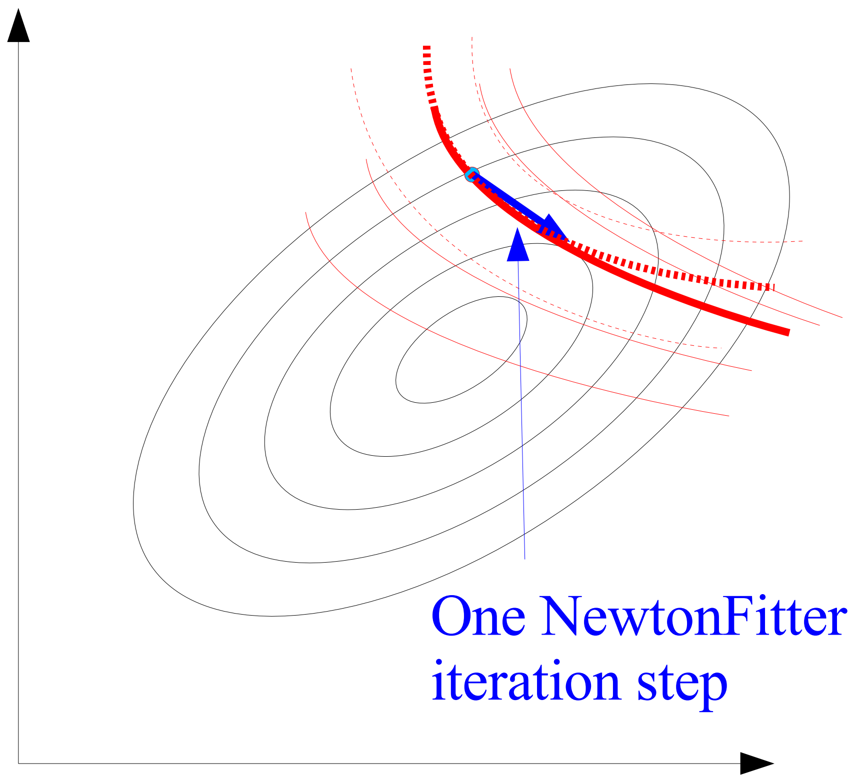
- Derivatives of 4-vectors w.r.t. parameters
- Dertivatives of constraints w.r.t. 4-vectors

OPALFitter vs. NewtonFitter

OPALFitter:
Approximates constraint
by tangential plane



NewtonFitter:
Approximates constraint
by tangential paraboloid

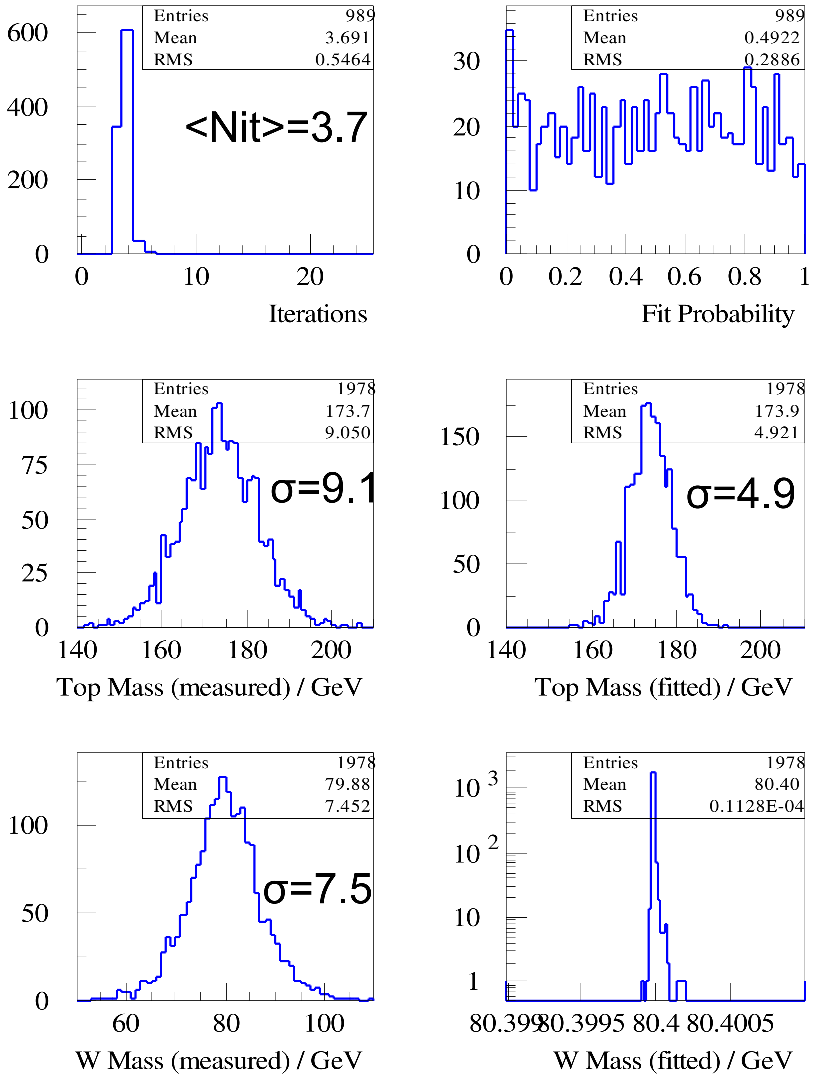


Toy Monte Carlo Studies

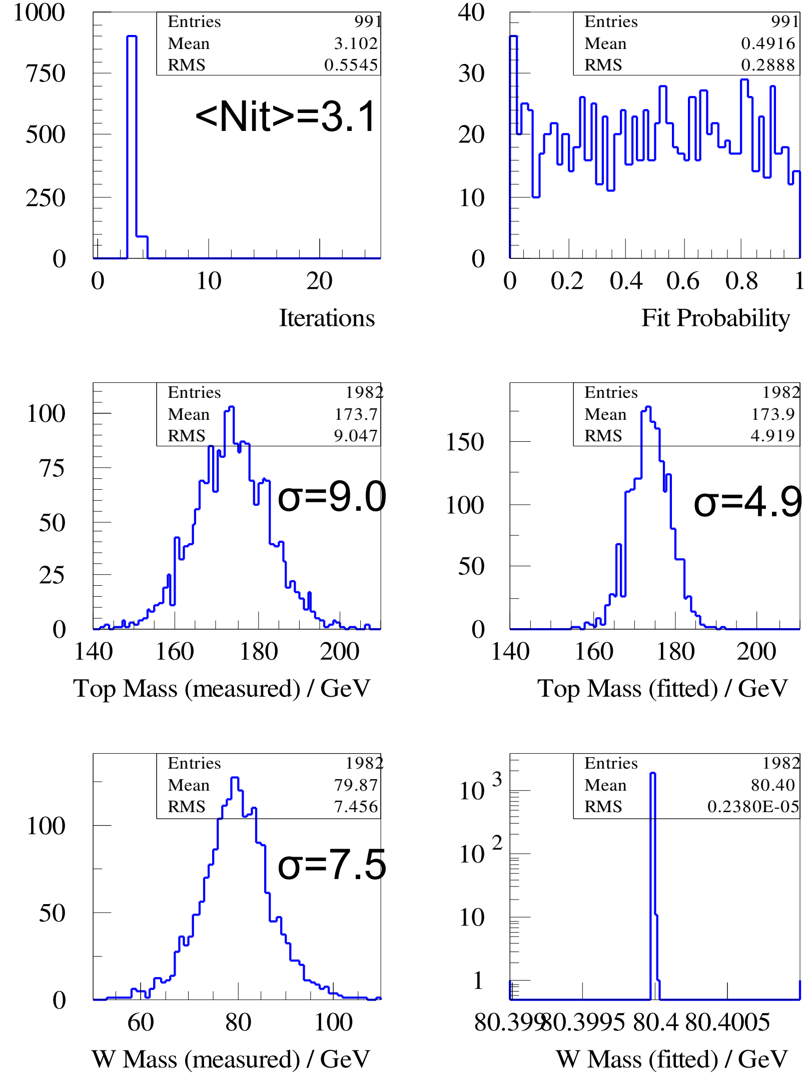
- $e^+ e^-$ -> t tbar, t -> bW, W->jj:
  6 jets in final state, $\sqrt{s}$=500GeV,
  no beamstrahlung, isotropic decays
- Mass of t and W: Nonrelativistic Breit-Wigner
- Smear jets with $\delta E/E = 35\%/\sqrt{E}$, $\delta\theta$=0.1rad, $\delta\varphi$=0.1rad
- Parametrize jets with E, θ, φ, treat them as massless
- Fit event (perfect jet-pairing) with 7 constraints:
  - $\Sigma p_x = 0$, $\Sigma p_y = 0$, $\Sigma p_z = 0$, $\Sigma E = 500$GeV
  - $m(\mathrm{W}_1) = 80.4$GeV, $m(\mathrm{W}_2) = 80.4$GeV
  - $m(\mathrm{t}_1) = m(\mathrm{t}_2)$
- 18 measured values, 7 constraints => 7dof

Toy MC: e+e- -> ttbar -> 6 jets

OPALFitter: 1.1% failed fits



NewtonFitter: 0.9% failed fits

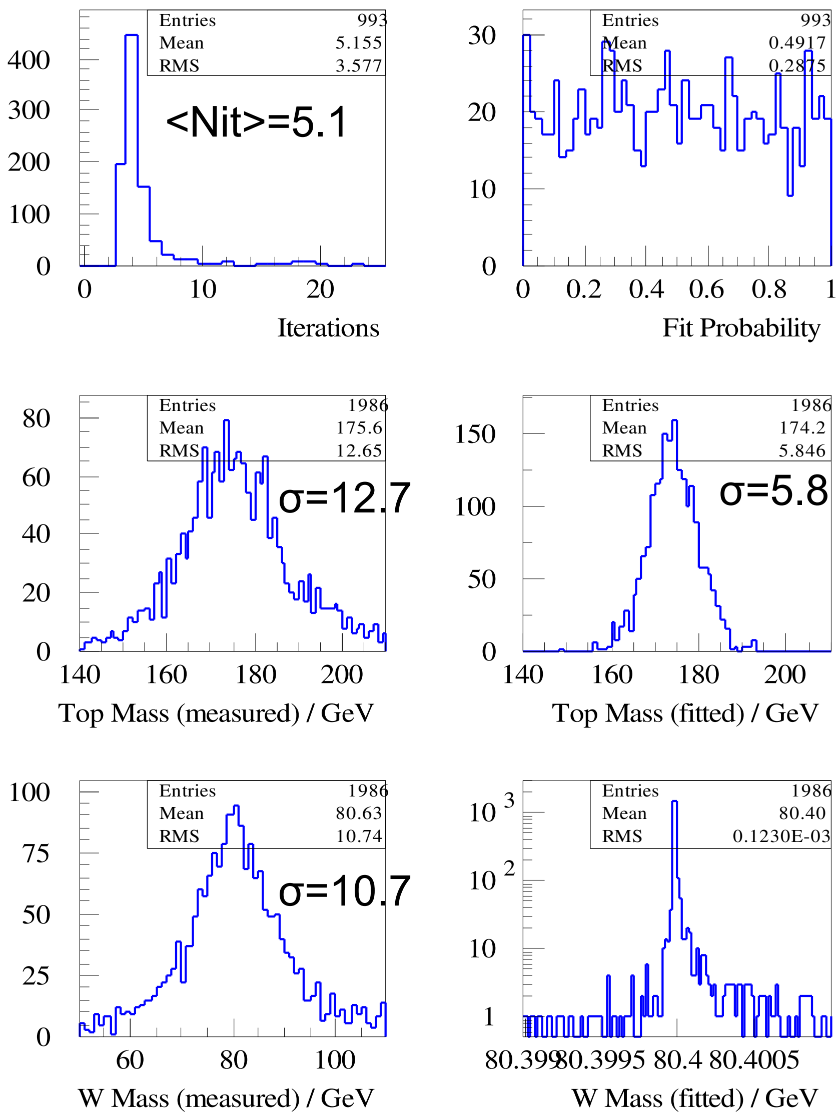


Semileptonic ttbar events

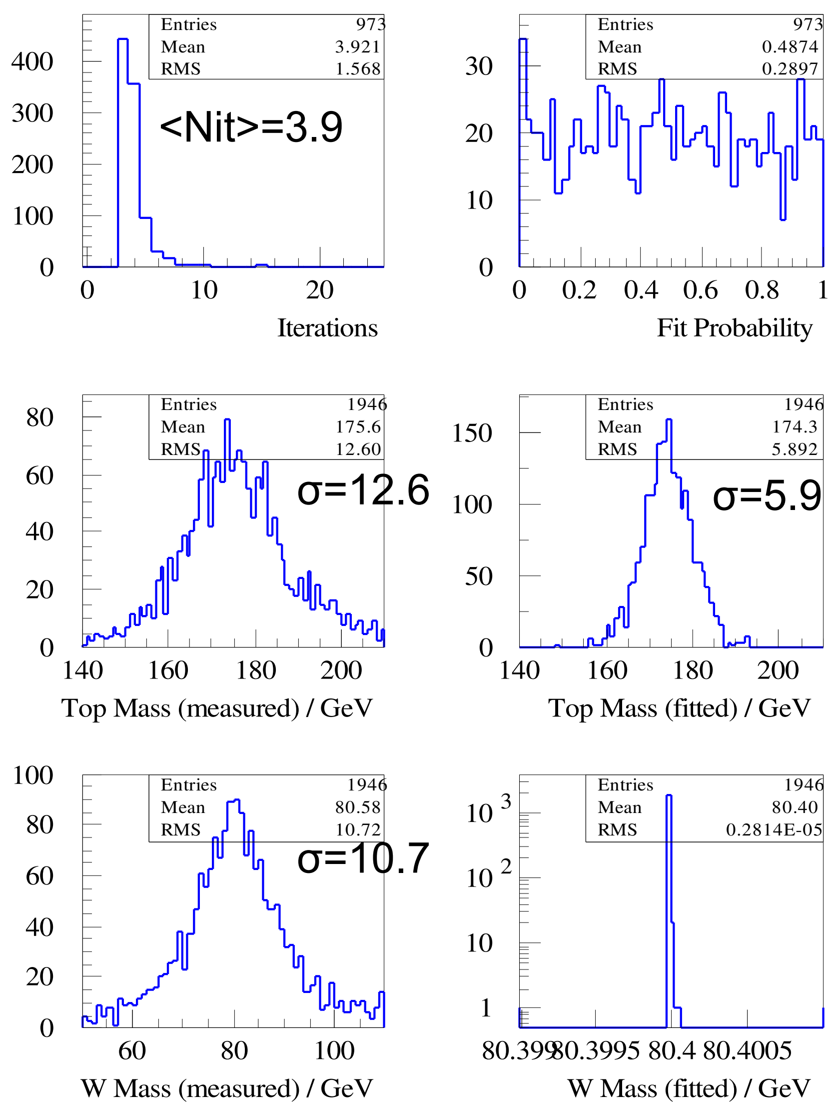
- Now generate $e^+ e^-$ -> t tbar, t -> bW, W_1->jj, W_2->ev
  4 jets + electron + neutrino in final state
- Smear electron with $\delta E/E = 10\%/\sqrt{E}$, $\delta\theta$=0.1rad, $\delta\varphi$=0.1rad
- Starting momentum of neutrino:
  given from px, py, pz of the event
- All constraints as in previous example
- 15 measured values, 3 unmeasured, 7 constraints -> 4dof

Toy MC: e+e- -> ttbar -> 4 jets e v

OPALFitter: 0.7% failed fits



NewtonFitter: 2.7% failed fits



Convergence

- NewtonFitter is not yet optimized for best convergence
- NewtonFitter generally needs less steps, but each step is more expensive. Overall NewtonFitter is ~2 times faster (may vary with the problem)
- Convergence criteria (so far) for NewtonFitter:
  - No parameter is changed by more than 1% of its sigma
  - All constraints are fulfilled within 1% of their resolution (resolution determined by error propagation from parameter errors)
- Problems with all iterative approaches:
  - Need a good start value
  - One iteration may send parameters far off
    => in NewtonFitter: scale step size such that no parameter is changed by more than 4sigma in a single step (can be optimized)

Soft Constraints

- Problem:
  Constraints may not be fulfilled exactly by physical situation
- Examples:
  - Mass of a W/Z has Breit-Wigner-shape,
    deviation may be bigger than detector resolution
  - Beamstrahlung leads to nonzero pz and reduction of $\sqrt{s}$
  - Proton remnant may carry nonzero px, py
- Possible solution:
  - Instead of imposing $f(a_i) = 0$ (hard constraint), add term to χ^2:
    $\chi^2_C = (f(a_i) / \sigma)^2$
  - Other penalty functions could be more appropriate (beamstrahlung!)
- Should improve fit probability distribution

Soft Constraints, Technicalities

- OPALFitter:
  - Distinguishes between measured and unmeasured quantities
  - assumes that $\partial^2\chi^2/\partial\xi_i\,\partial\xi_j = 0$ for unmeasured quantities
  - => Additional χ^2 terms that involve unmeasured quantities are not possible
- NewtonFitter:
  - Does not distinguish between measured and unmeasured quantities
  - Has already framework to add 2nd derivatives of constraint functions
  - => Soft constraints are easily added in NewtonFitter

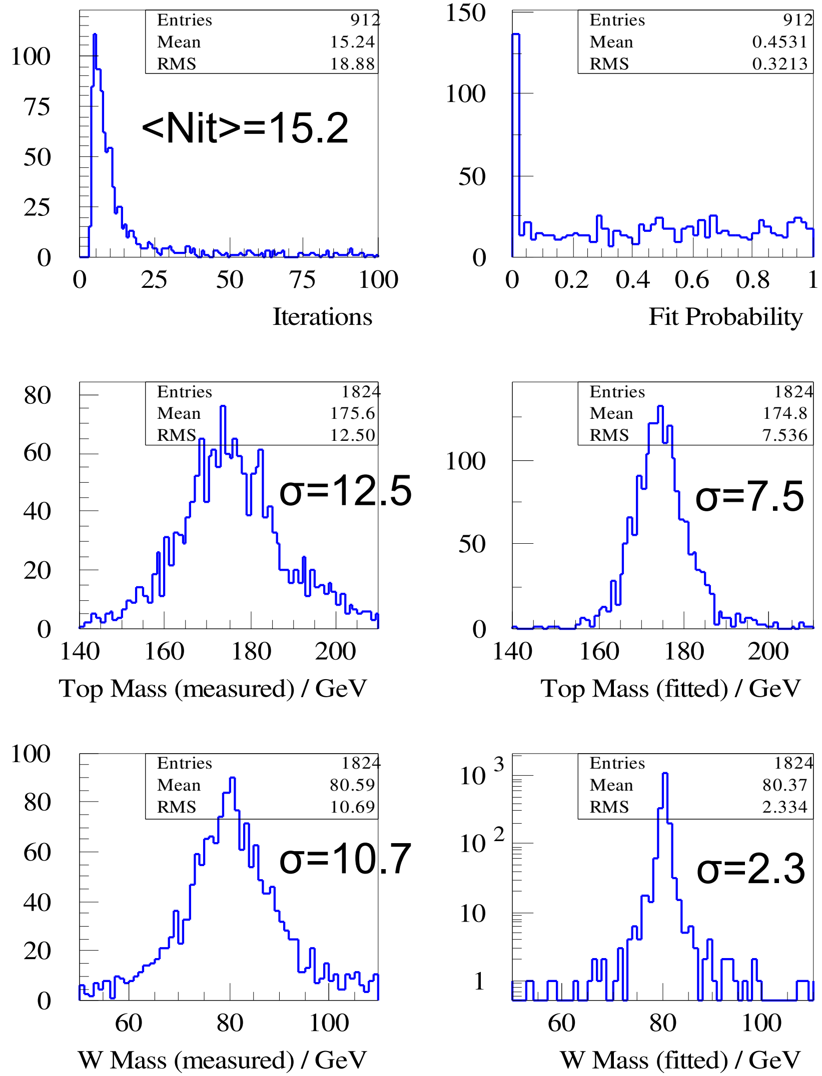
Toy MC Study:

- Use ttbar -> 4j e v Monte Carlo as before
- Replace hard mass constraints by soft ones:
  - $\chi 2$ += $(m(W_i) - m_0)^2/\sigma^2$ with m_0=80.4GeV, σ = 2.1GeV = Γ_W
  - $\chi 2$ += $(m(t_1) - m(t_2))^2/\sigma^2$ with $\sigma = \sqrt{2}$ 1.4GeV = $\sqrt{2}\,\Gamma_t$
- Remark:
  A Breit-Wigner is much broader than a Gaussian;
  for a correct fit probability distribution, one needs a different penalty function. However, experience shows that this makes the constraint effectively useless.
  (Remark 2: The correct penalty function is *not* simply -2ln(L))

Toy MC: e+e- -> ttbar -> 4 jets e ν, soft c.

- Soft constraints are more difficult to handle for the fitter:
  - More iterations needed
  - Rate of failed fits higher than for hard constraints
  - Fit probability has peak at low values
- => Needs more tuning
- 2 possible strategies for better convergence:
  - Start with large sigma values, then decrease (sort of simulated annealing)
  - Start with hard constraints, then relax
- But: it works

failed fits: 8.8%

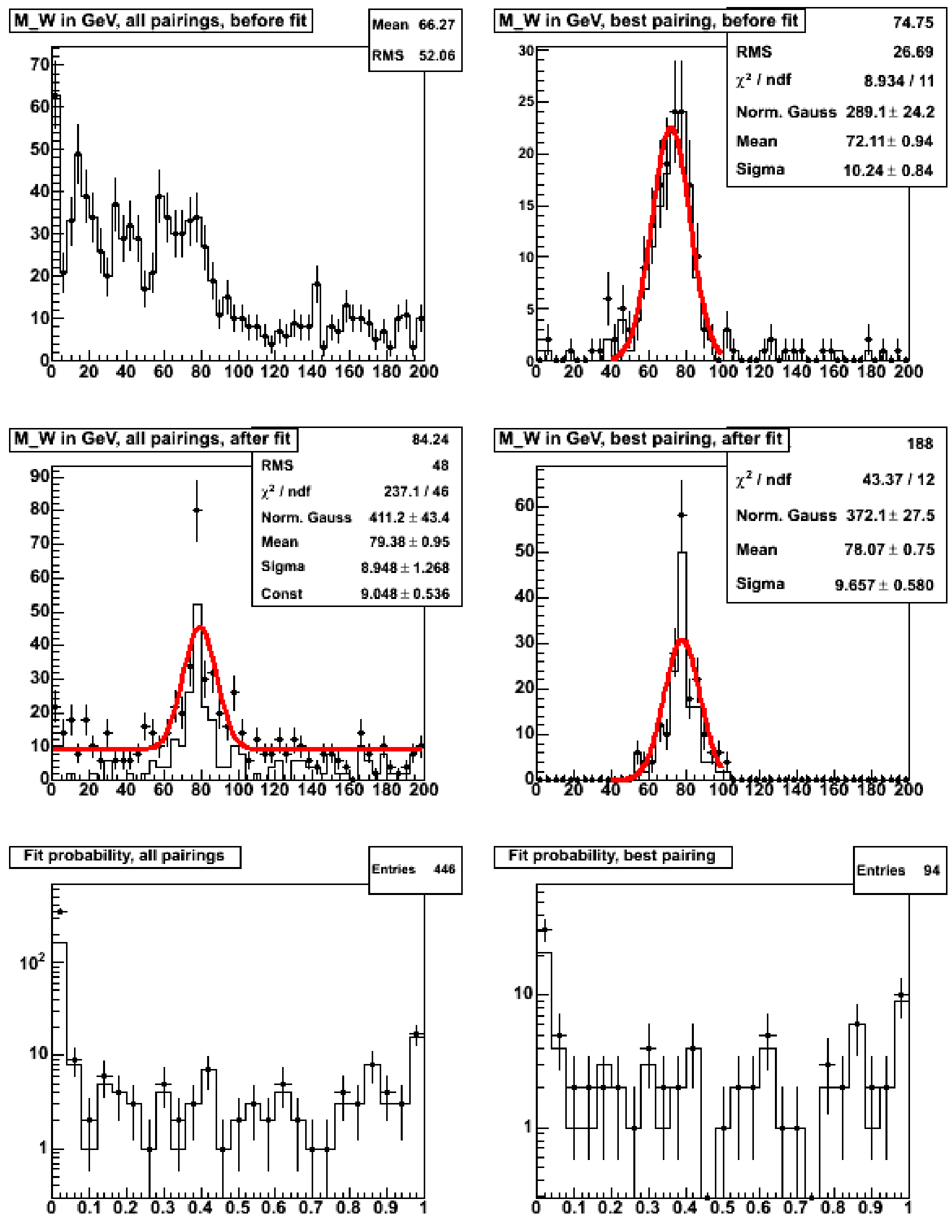


Availability

- The Kinfit code has been provided in Marlin by Jenny: MarlinKinfit => check it out
  (See also Jenny's talk in Zeuthen:
  http://ilcagenda.linearcollider.org/contributionDisplay.py?contribId=52&sessionId=4&confId=2389)
- Development is still ongoing, so use the cvs HEAD version if possible
- An example processor to fit WW events has been written by Jenny and is included in MarlinKinfit

Result from Marlin Processor (Jenny List)

- 200 full WW events
- LDC00Sc, 4T, Mokka 5.4
- Track cheater, TrackwiseParticleFlow (O. Wendt)
- Energy scale not tuned
- Just a proof of principle



Available Classes

- Fit engines implemented so far:
  - OPALFitter:
  - NewtonFitter
- FitObjects implemented so far:
  - JetFitObject: Jet with E, θ, φ parametrization, mass can be set
  - NeutrinoFitObject: Neutrino with E, θ, φ parametrization
- Hard constraints implemented so far:
  - MomentumConstraint: $a \cdot \Sigma E + b \cdot \Sigma p_x + c \cdot \Sigma p_y + d \cdot \Sigma p_z - e = 0$
  - MassConstraint: m(object list 1) - m(object list 2) - $m_0 = 0$
- Soft constraints implemented so far:
  - SoftGaussMomentumConstraint: $(a \cdot \Sigma E + b \cdot \Sigma p_x + c \cdot \Sigma p_y + d \cdot \Sigma p_z - e)^2/\sigma^2 = \chi^2$
  - MassConstraint: $(m(\text{object list 1}) - m(\text{object list 2}) - m_0)^2/\sigma^2 = \chi^2$

Summary and Conclusions

- Kinfit provides a flexible framework for kinematic fitting:
  Fit engine, constraints and fitted objects are separated and can be combined in a flexible way
- A new fit engine NewtonFitter is provided in addition to the well-tested OPALFitter
- NewtonFitter can handle soft constraints that involve unmeasured quantities
- Some (example) FitObject classes have been implemented,
  plus hard and soft momentum and mass constraints
- Work continues
- Your feedback is welcome!