# MarlinKinfit — Kinematic Fitting in MarlinReco

Jenny List

DESY

ILD Meeting Zeuthen, January 15 2008

# Why a kinematic fit?

**We plan to optimize the final physics performance!**

„improve resolution by exploiting a priori knowledge"

- „resolution ": (jet)energy, invariant masses, jet pairing, ...
- „a priori knowledge ": energy and momentum conservation, particle masses, vertices ....

**How does it work?**

- Vary measured quantities within errors (sensible error parametrisation important!)
- enforce hard constraints (e.g. $\Sigma E_j = \sqrt{s}$) by Lagrange multipliers
- „soft constraints ": additional $\chi^2$ term

# Goals of MarlinKinfit

**Would like**

- not to reinvent the wheel!
- $\Rightarrow$ use well tested minimisation backend ($\rightarrow$ LEP)
- to provide flexible machinery adjustable to many different configurations
- easy to use within ILC software $\Rightarrow$ C++, Marlin

**Beyond this scope**

- one processor which can do everything ($WW$, $ZZ$, $ZH$, $ZHH$, $t\bar{t}$, SUSY, ...)
- error parametrisation for particle momenta

# BaseFitObject: „reconstructed particles“

- four-momenta: measured / unmeasured parameters
- covariance matrix
- first partial derivatives of $p_x$, $p_y$, $p_z$, $E$ w.r.t. to parameters
- contribution to global $\chi^2$
- currently implemented:
  - `JetFitObject`: measured $E$, $\theta$, $\phi$
  - `NeutrinoFitObject`: unmeasured, $p_{x,y,z}$, $E$ from constraints
- new classes with different parametrisations welcome! (write them or ask me...)

# BaseConstraint: „a priori knowledge“

- Value: $= 0$ if constraint fulfilled
- list of FitObjects which contribute
- first partial derivatives w.r.t. $p_x$, $p_y$, $p_z$, $E$
- currently implemented: Px/y/zConstraint, EnergyConstraint, MassConstraint
- again, further contributions are welcome!

# `BaseFitter`: the fit engine itself

- list of `FitObjects`
- list of `Constraints`
- assembles global $\chi^2$ and covariance matrix
- actual $\chi^2$ minimisation
- currently implemented: `WWFitter` one-to-one $F \longrightarrow C$ translation of OPAL kinematic fit engine `WWFGO`. Results have to be checked to be identical down to machine precision.
- probably soon to come: `cernlib` free version (matrix inversion)

# Usage Example: $W^+W^- \longrightarrow$ 4 jets, equal mass constraint

- Energy and momentum conservation:
  - $\Sigma E - \sqrt{s} = 0$
  - $\Sigma p_{x/y/z} = 0$
- equal mass constraint: $M_{12} = M_{34}$
- try all three possible jet pairings!
- plots to come (just proof of principle):
  - 200 full hadronic $WW$ events
  - LDC00Sc, 4T, Mokka 5.4
  - TrackCheater, TrackwiseParticleFlow (O.Wendt)

# Usage Example: $W^+W^- \longrightarrow 4$ jets, equal mass constraint

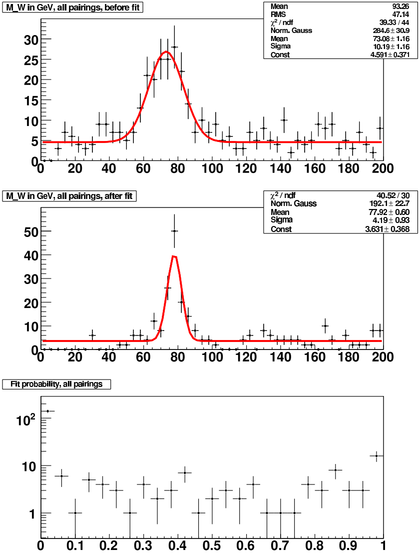
```
JetFitObject fitjets[NJETS];
for (int i = 0; i < 4; i++) {

   ReconstructedParticle* j = jetcol->getElementAt( i );
   lvec = HepLorentzVector ((j->getMomentum())[0], (j->getMomentum())[1],
(j->getMomentum())[2], j->getEnergy());

  fitjets[i] = new JetFitObject (lvec.e(), lvec.theta(), lvec.phi(), erre, errtheta, errphi);
}

FourJetPairing pairing (fitjets);
JetFitObject *permutedjets[NJETS];

for (int iperm = 0; iperm < pairing.getNPerm(); iperm++) {

   pairing.nextPermutation (permutedjets);
   PxConstraint pxc;
   for (int i = 0; i < NJETS; ++i) pxc.addToFOList (*(permutedjets[i]));

   PyConstraint pyc;
   for (int i = 0; i < NJETS; ++i) pyc.addToFOList (*(permutedjets[i]));

   PzConstraint pzc;
   for (int i = 0; i < NJETS; ++i) pzc.addToFOList (*(permutedjets[i]));

   EConstraint ec(500.);
   for (int i = 0; i < NJETS; ++i) ec.addToFOList (*(permutedjets[i]));
```

# Usage Example: $W^+W^- \to$ 4 jets, equal mass constraint

```
    MassConstraint w(0.);
    w.addToFOList (*(permutedjets[0]), 1);
    w.addToFOList (*(permutedjets[1]), 1);
    w.addToFOList (*(permutedjets[2]), 2);
    w.addToFOList (*(permutedjets[3]), 2);

    WWFitter fitter;
    for (int i = 0; i < NJETS; ++i) fitter.addFitObject (*(permutedjets[i]));
    fitter.addConstraint (pxc);
    fitter.addConstraint (pyc);
    fitter.addConstraint (pzc);
    fitter.addConstraint (ec);
    fitter.addConstraint (w);

    double prob = fitter.fit();
    cout << '' fit probability = '' << prob << endl;
    cout << '' error code: '' << fitter.getError() << endl;
    for (int i = 0; i < NJETS; ++i) cout << ''final four-vector of jet '' << i << ''='' <<
*(permutedjets[i]) << endl;

    cout << ''final mass of W 1:'' << w.getMass(1) << endl;
    cout << ''final mass of W 2:'' << w.getMass(2) << endl;
}
```

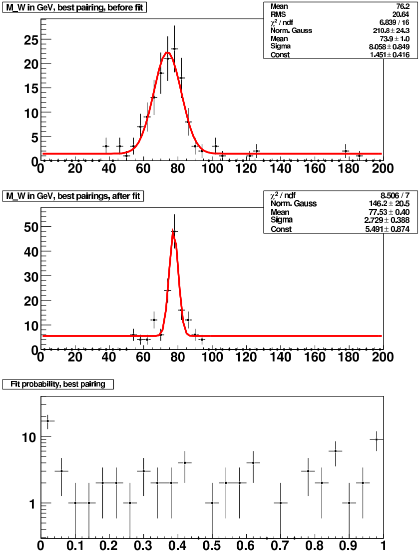
# Results: All jet pairings

- $M_W$ before fit:
  mean = (73.1±1.2) GeV,
  $\sigma$ =(10.2±1.2) GeV
- $M_W$ after fit:
  mean = (77.9±0.6) GeV,
  $\sigma$ =(4.2±0.9) GeV
- Fit probability: resonably flat
  (caveat: jet errors put „by hand" to $100\%/\sqrt{E}$ and 10 mrad for $\theta$ and $\phi$)



# Results: Best jet pairing

- $M_W$ before fit:
  mean = (73.9±1.0) GeV,
  $\sigma$ =(8.1±0.9) GeV
- $M_W$ after fit:
  mean = (77.5±0.4) GeV,
  $\sigma$ =(2.7±0.4) GeV
- Fit probability: flat
  (caveat: jet errors put
  „by hand“ to $100\%/\sqrt{E}$
  and 10 mrad for $\theta$ and $\phi$)



# Summary

- MarlinKinfit provides a flexible interface to an established fitter
- it is waiting to be used!
- check out from MarlinReco in `MarlinReco/[version]/Analysis/MarlinKinfit` including
  - `README`
  - doxygen documentation
  - note on the minimisation mathematics behind
  - and the *WW* example
- or look at: `http://www.desy.de/~blist/kinfit`
- feed-back, questions, complaints, more examples, fit objects welcome: `jenny.list@desy.de`

**... and happy fitting!**