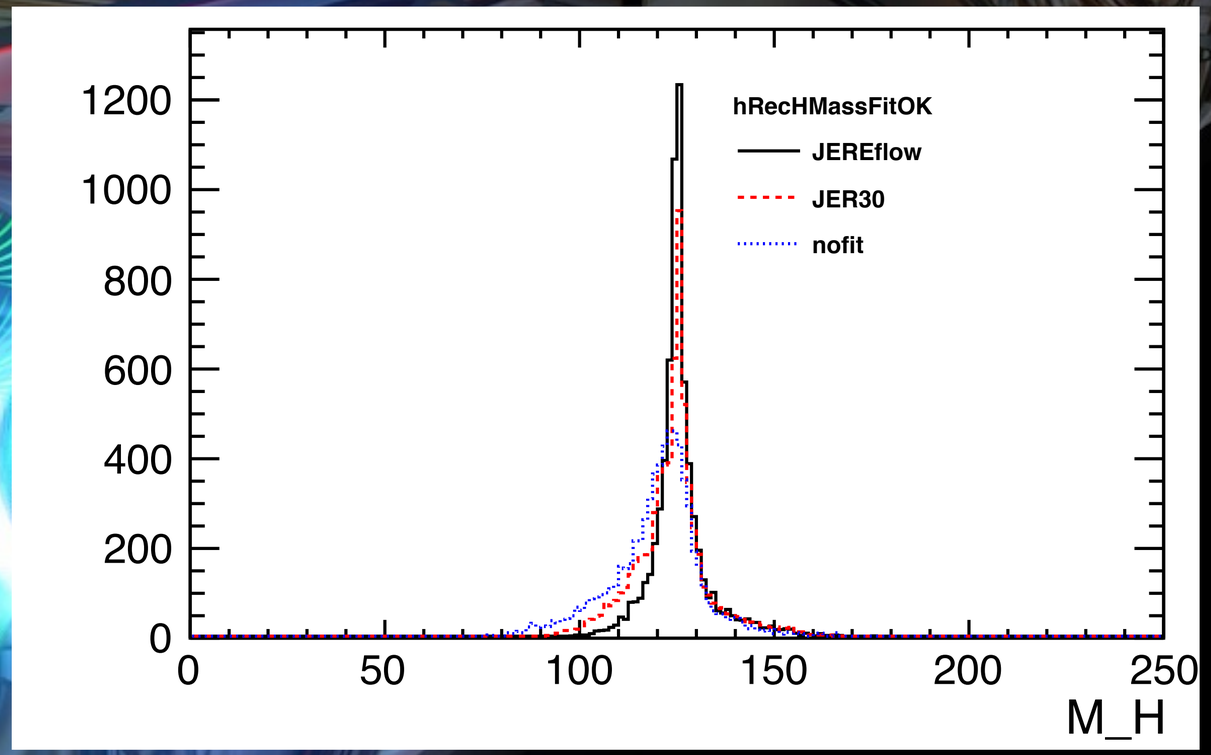


(Marlin)Kinfit: Kinematic fitting for Future e+e- Colliders

J. List (DESY) on behalf of the IDT-WG3 Software & Computing Group

MarlinKinfit Tutorial May 18, 2022

Overview

- Part I: “Lean-back & Listen” ;-)
  - Introduction: Kinematic fitting and the method of Lagrange multipliers
  - The basic minimisation algorithm: the OPALFitter
  - NewtonFitter: A new fitting engine
  - ISR handling
  - Error parametrisations
  - Ongoing developments
- Part II: “Hands-On”
  - Try it yourself!

References:

- constrained fitting: Chapter 7 of “Data analysis in high energy physics: A practical guide to statistical methods”
- MarlinKinfit: LC-TOOL-2009-001
- ISRPhoton:
  - LC-TOOL-2009-004
  - Nucl.Instrum.Meth.A 624 (2010) 184-191

Part I

Kinematic fitting

- lot's of knowledge about e+e- events beyond "raw" measured quantities:
  - known four-momentum of the initial state, e.g. $\Sigma p_Y = 0$ => hard constraint
  - event-by-event deviations from ISR+BS: known "on average" => pseudo-measured quantities
  - masses of intermediate particles, e.g. $M(j,j) = M_H$ or M_Z => hard or soft constraint
  - also know which quantities are very well measured and which less so => error parametrisation
- formulate an hypothesis under which to interpret the event
  - minimize a χ^2 under constraints by adjusting particle momenta
  - technique to force constraints: Lagrange multipliers (MINUIT not applicable)
- exploit this to
  - improve precision on observables, eg invariant masses
  - determine unmeasured quantities (e.g. neutrino momentum)
  - find best jet pairing
  - select / reject events which match / don't hypothesis

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

For minimum: Seek values where all derivatives vanish:

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

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

Solve this nonlinear set of equations:

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

The OPALFitter Method

The equations to solve:

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

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

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

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:

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

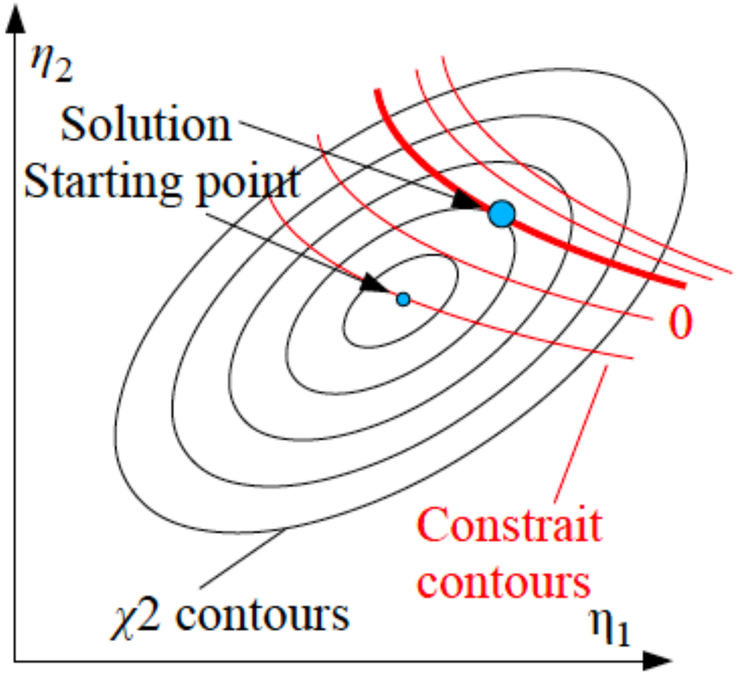
See L. Lyons: *Statistics for nuclear and particle physics,* Cambridge Univ. Press 1986.

How the OPALFitter works

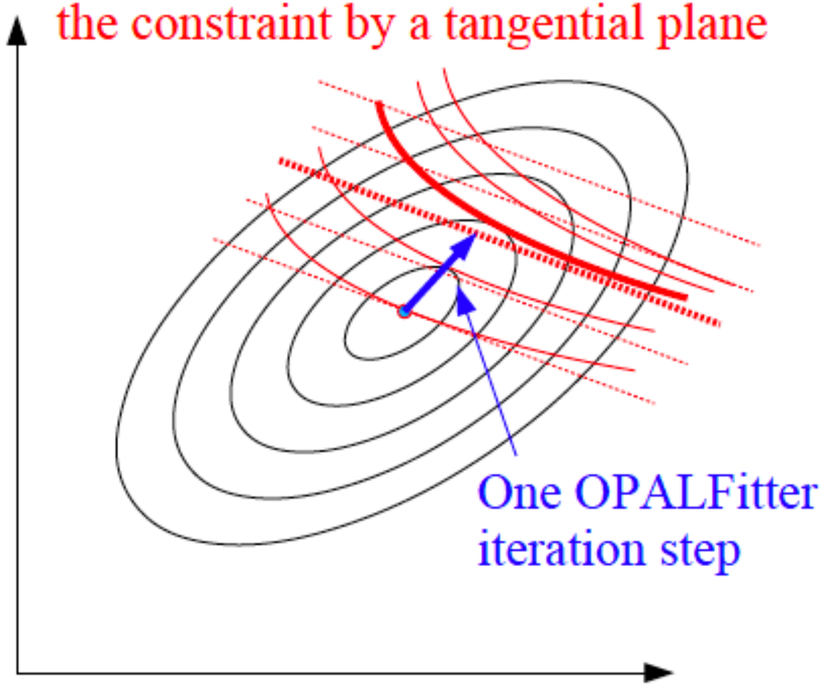
$$\vec{0} = V^{-1} \cdot (\vec{\eta} - \vec{y}) + \vec{F}_\eta^T \cdot \vec{\lambda}$$
$$\vec{0} = \vec{F}_\xi^T \cdot \vec{\lambda}$$
$$\vec{0} = \vec{f}(\vec{\eta}, \vec{\xi})$$

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



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

Especially true for our particle flow detectors!

Note: tails of Breit-Wigner are too wide to provide meaningful constraint => approximate core by Gaussian, some freedom in choice of σ

The Basic Abstractions

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

A 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
- Fitter solves system of equations and updates parameters of FitObjects
- Fitter checks for convergence (Parameter changes small, constraints fulfilled), iterates if necessary

From FitObjects

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

From ConstraintObjects

The User Code - Formulation of Fit Hypothesis!

For head-on, else E_{CM} x crossing angle!

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();
```

Some design considerations…

- Fitter Engine decoupled from the rest
  => can try different algorithms
  (2 are implemented: OPALFitter and New~~ton~~Fitter)
- 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 Fitting Engine: The New(ton)Fitter

- 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: New~~ton~~Fitter

The Mathematics of the NewFitter

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

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

The total χ²: $\chi_T^2(\vec{a}, \vec{\lambda}) = \chi^2(\vec{a}, \vec{y}) + \vec{\lambda}^T \cdot \vec{f}(\vec{a}).$

Seek stationary point, where all derivatives vanish:

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

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 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

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

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

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

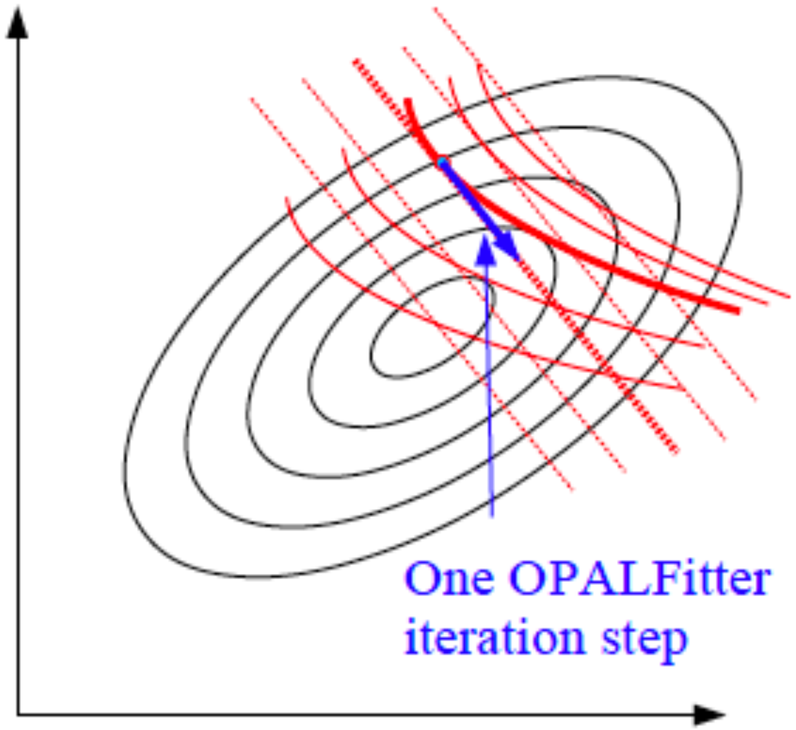
f_1

We need only:

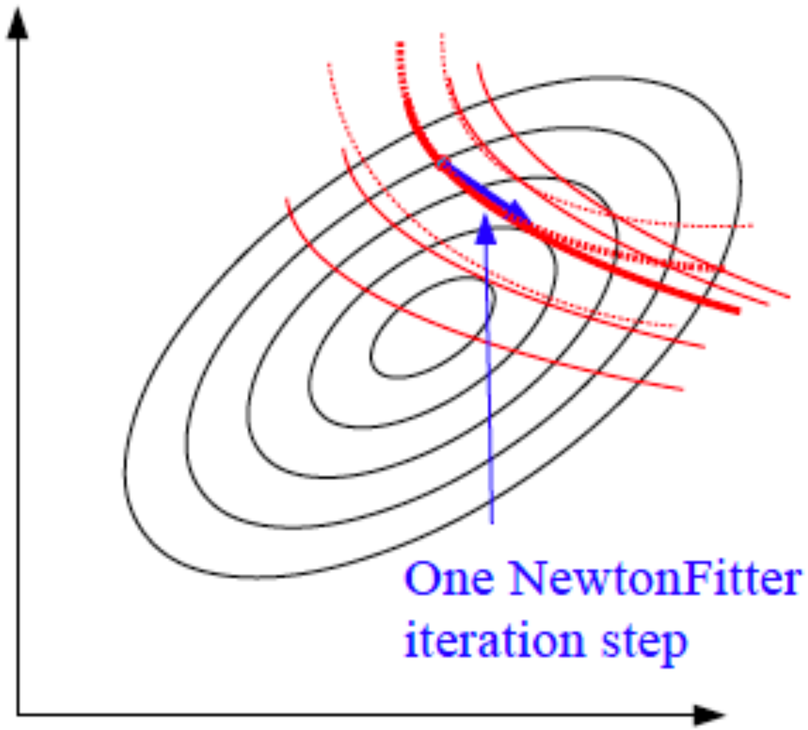
- Derivatives of 4-vectors w.r.t. parameters
- Dertivatives of constraints w.r.t. 4-vectors

OpalFitter vs NewFitter - I

OPALFitter:
Approximates constraint
by tangential plane



NewtonFitter:
Approximates constraint
by tangential paraboloid



OpalFitter vs NewFitter - II

- 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
- New~~ton~~Fitter:
  - Does not distinguish between measured and unmeasured quantities
  - Has already framework to add 2nd derivatives of constraint functions
  - => Soft constraints are easily added in New~~ton~~Fitter

Treatment of ISR & Beamstrahlung

PhotonFitObject, „unmeasured“

- treat photon parameters as unmeasured
- this „costs“ constraints
- usually not applicable for final states with true E_{miss} (ν, LSP)

PhotonFitObject, measured

“Pseudo-measurement”

- treat photon parameters as measured
- choose error parametrisation such that fitted E_γ spectrum reproduces ISR spectrum
- advantage: no loss of constraints!

Treatment of ISR & Beamstrahlung

PhotonFitObject, „unmeasured“

- treat photon parameters as unmeasured
- this „costs“ constraints
- usually not applicable for final states with true E_{miss} (ν, LSP)

PhotonFitObject, measured

“Pseudo-measurement”

- treat photon parameters as measured
- choose error parametrisation such that fitted E_γ spectrum reproduces ISR spectrum
- advantage: no loss of constraints!

How the pseudo-measurement works

Assume

- Photon px = py = 0, E = |pz|
- measured E = 0

Approximate ISR spectrum by

Take into account emission from either beam

$$\chi^2 = -\ln \mathcal{P}\left(\eta_{\mathrm{i,meas}} | \eta_{\mathrm{i}}\right) + const.$$

$$\mathcal{P}(y) = \beta\, y^{\beta-1} \qquad \beta = \frac{2\alpha}{\pi}\left(\ln\frac{s}{m_{\mathrm{e}}^2} - 1\right)$$

$$\mathcal{P}\left(p_{\mathrm{z},\gamma}\right) = \frac{\beta}{2E_{\mathrm{max}}} \cdot \left|\frac{p_{\mathrm{z},\gamma}}{E_{\mathrm{max}}}\right|^{\beta-1}$$

Find quantity flat from -1 to 1

$$z = \mathrm{sign}(p_{\mathrm{z},\gamma})\left(\frac{|p_{\mathrm{z},\gamma}|}{E_{\mathrm{max}}}\right)^{\beta}$$

And turn into Gaussian

$$\eta = \sqrt{2} \cdot \mathrm{erf}^{-1}(z)$$

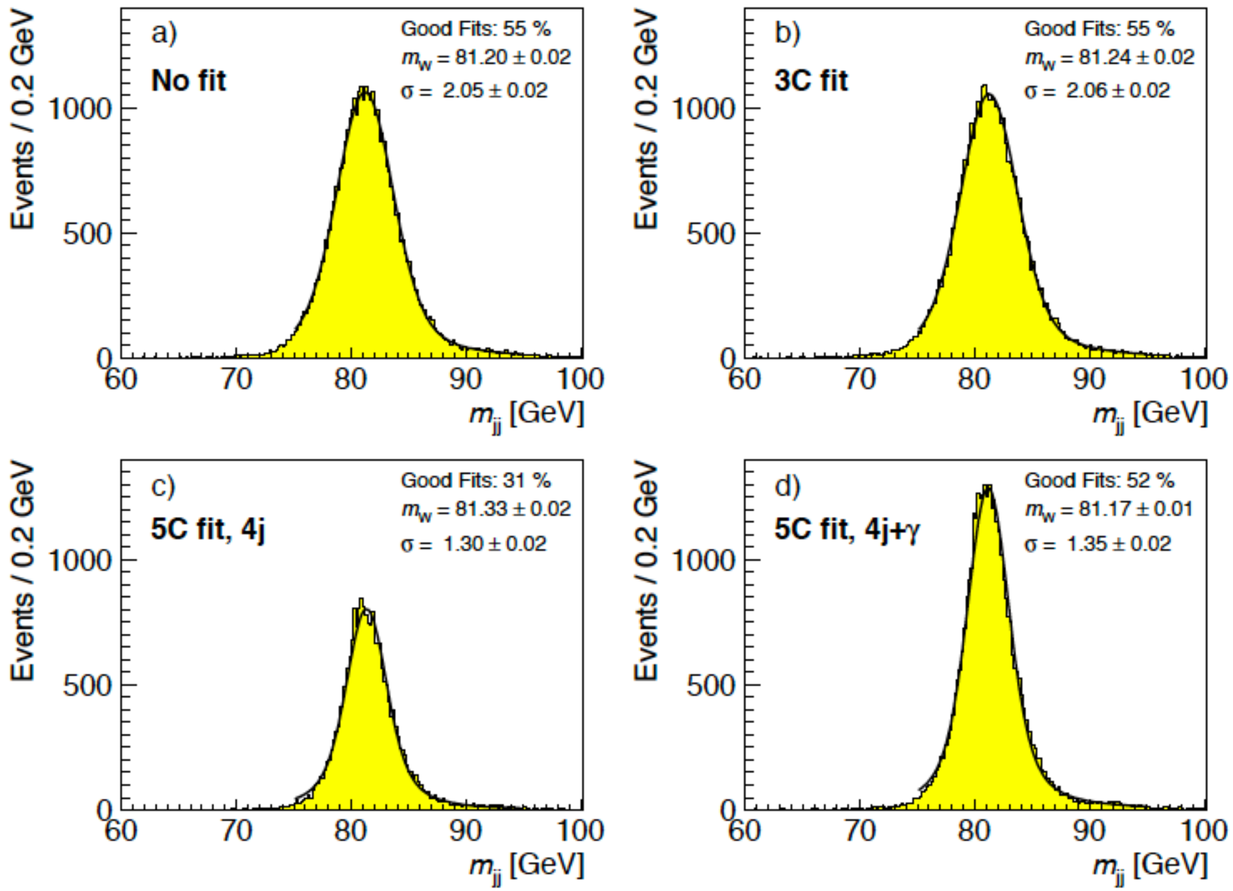
Note: β depends on E_{CM},

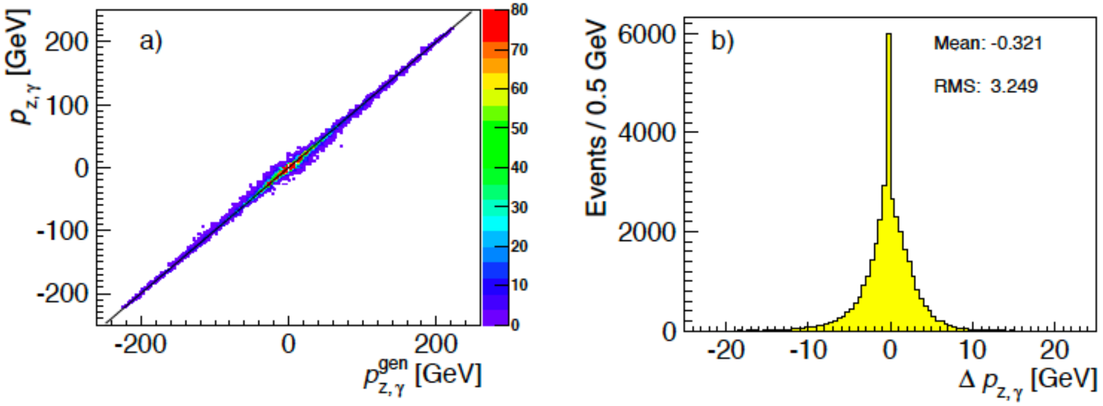
E_{max} depends on E_{CM} and targetted physics process!

Test on 4-jet events at 500 GeV

5C fit with 4j + γ:

- as many good fits as for 3C fit
- as narrow mass peak as 5C fit
- photon p_Z well reconstructed!





The Software

- MarlinKinfit: provides all classes for the inner working of the kinematic fit
  https://github.com/iLCSoft/MarlinKinfit/
  => only “experts” need to touch this

- MarlinKinfitProcessors:
  contains example Marlin processors for various fit hypotheses
  https://github.com/iLCSoft/MarlinKinfitProcessors
  => users can copy an example from here and adjust it to their needsP

- Available in Key4HEP world via MarlinWrapper and LCIO <-> EDM4HEP conversion

MarlinKinfitProcessors - a subset

- **TopTester**: toy MC for ttbar -> bbqqlnu - mostly for development work
- **TTbarExample**: tests ttbar -> 6 jets hypothesis with E,p conservation, M_W constraints and equal M_{top} constraint
- **WW5CFit**: tests WW -> 4 jets hypothesis with E,p conservation and equal mass constraint
- **ZH5CFit**: tests ZH -> 4 jets hypothesis with E,p conservation, hard m_H and soft m_Z constraint
- **ZHllqq5CFit: tests ZH -> ll jj hypothesis with E,p conservation, hard m_H constraint and soft m_Z constraint => will be used for hands-on part**
- various examples of mass constrained fitting of decays, eg π^0 -> $\gamma\gamma$ => not for today…

MarlinKinfit

- **Fit engines: base class `BaseFitter`:**
  - `OPALFitterGSL`
  - `NewFitterGSL` = improved version of `NewtonFitter`
- **Fit objects: base class `BaseFitObject` <- `ParticleFitObject`**
  - `JetFitObject` (E, θ, φ)
  - `LeptonFitObject` (ptinv, θ, φ) - can be initialized from LCIO Track
  - `NeutrinoFitObject` (E, θ, φ, m=0)
  - `ZinvisibleFitObject` (E, θ, φ, m=mZ)
  - `ISRPhotonFitObject` (pz)
- **Constraints: base class `BaseConstraint` <- `BaseHardConstraint`, `BaseSoftConstraint`**
  - `MomentumConstraint`
  - `MassConstraint`
  - `SoftGaussMassConstraint`
  - …

MarlinKinfit

- **Jet pairings: base class `BaseJetPairing`:**
  - `FourJetPairing:` 3 permutations, for equal mass bosons
  - `FourJetZHPairing:` 6 permutations, for 2 different bosons
  - `2B4JPairing`: 6 permutations for tt- > bWbW -> bjj bjj
- **What the hell is going on?**
  - `TextTracer`
  - `RootTracer`

=> User actions to be executed during fit iterations, eg print out or filling root tree

Recent and ongoing developments

- fit quality strongly depends on **input uncertainties** on measured quantities
  => error parametrisation!
  - leptons: “easy” since momentum from track fit => use track covariance matrix
  - jets: enhance ParticleFlow to “ErrorFlow” => combine uncertainties from indicidual PFOs and confusion parametrisation to estimate dE, dθ, dφ for each jet individually
    => Yasser Radkhorrami, arXiv :2110.13731 [hep-ex] & PhD in prep.
  - figures of merit:
    - fit probability flat between 0 and 1
    - pulls of measured quantities Gaussian with μ=0, σ=1

- ongoing developments
  - pass full jet covariance matrix to kinfit
  - allow for correlated uncertainties between jets => account for jet clustering errors…
  - neutrino correction for b/c jets with semi-leptonic decays

Further ideas / plans (lacking person power - any takers?! ;-))

- (re-)enable mass constrained decay-chain / vertex fitting
- fully optimise convergence behaviour of NewFitter
- try even smarter minimasation algorithms
- automatize usage of non-Gaussian pdfs (a la ISRPhoton)

Part II

- assumes you have a working iLCSoft environment
- and know how to write/run a MarlinProcessor
- have access to grid or /pnfs/desy.de/ilc/prod/ilc
  OR
  download example file from desycloud

Goals for today

- perform a 4.5C fit on ee->ZH->$\mu\mu$bb at 250 GeV
- using JetFitObject, LeptonFitObject, ISRPhotonFitObject, MomentumConstraint, SoftGaussianMassConstraint, OPALFitter in an existing example
- see impact of
  - including / not including ISR photon in fit hypothesis
  - using ErrorFlow vs simple x/sqrt(E) for JER
- switch on tracing for a specific event

Get the relevant processors

- download the stuff for this tutorial:
  `git clone https://github.com/ILDAnaSoft/MarlinKinfitTutorial.git`
- change to MarlinKinfitTutorial folder:
  `cd MarlinKinfitTutorial`
- download MarlinKinfitProcessors (some adjustments for this tutorial, thus need HEAD version):
  `git clone https://github.com/ILCSoft/MarlinKinfitProcessors.git`
- initialize iLCSoft, e.g.:
  `/cvmfs/ilc.desy.de/sw/x86_64_gcc82_centos7/v02-02-03/init_ilcsoft.sh`
- build MarlinKinfitProcessors, and AddNeutralPFOCovMat and HDecayMode:

```
cd MarlinKinfitProcessors
mkdir build
cd build
cmake -C $ILCSOFT/ILCSoft.cmake ..
make
make install
cd ../..
```

- set MARLIN_DLL (get default by echo $MARLIN_DLL, change path to MarlinKinfitProcessors to your local)

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/rv02-01-02.sv02-01-02.mILD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

-

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/ rv02-01-02.sv02-01-02.mlLD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

fish out H-> bb out of H -> all

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

-

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/ rv02-01-02.sv02-01-02.mlLD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

fish out H-> bb out of H -> all

Fix in neutral PFO error propagation, not needed from v02-02 onwards

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

-

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/ rv02-01-02.sv02-01-02.mlLD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

fish out H-> bb out of H -> all

Fix in neutral PFO error propagation, not needed from v02-02 onwards

Find isolated muon PFOs

-

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/ rv02-01-02.sv02-01-02.mlLD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

fish out H-> bb out of H -> all

Fix in neutral PFO error propagation, not needed from v02-02 onwards

Find isolated muon PFOs

Form jets

-

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/ rv02-01-02.sv02-01-02.mlLD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

fish out H-> bb out of H -> all

Fix in neutral PFO error propagation, not needed from v02-02 onwards

Find isolated muon PFOs

Form jets

Apply error flow

-

Steering file: ZHAnalysis.xml

- Data file:
  - at DESY: /pnfs/desy.de/ilc/prod/ilc/mc-opt/ild/dst-merged/250-SetA/test/ILD_l5_o1_v02_nobg/v02-01-02/ rv02-01-02.sv02-01-02.mlLD_l5_o1_v02_nobg.E250-SetA.I401010.Pe2e2h.eL.pR.n000.d_dstm_14763_0.slcio
  - for grid path, production details c.f. https://ild.ngt.ndu.ac.jp/elog/dbd-prod/318
- Workflow

```
   <processor name="MyHdecayMode"/>
<if condition="MyHdecayMode.GoodEvent">
   <processor name="MyAddNeutralPFOCovMatLite" condition="" />
   <processor name="MyIsolatedLeptonTaggingProcessor" condition="" />
   <processor name="MyFastJetProcessor"/>
   <processor name="MyErrorFlow" condition="" />
   <processor name="MyZHllqqFit" condition="" />
</if>
```

fish out H-> bb out of H -> all

Fix in neutral PFO error propagation, not needed from v02-02 onwards

Find isolated muon PFOs

Form jets

Apply error flow

Do the fit!

-

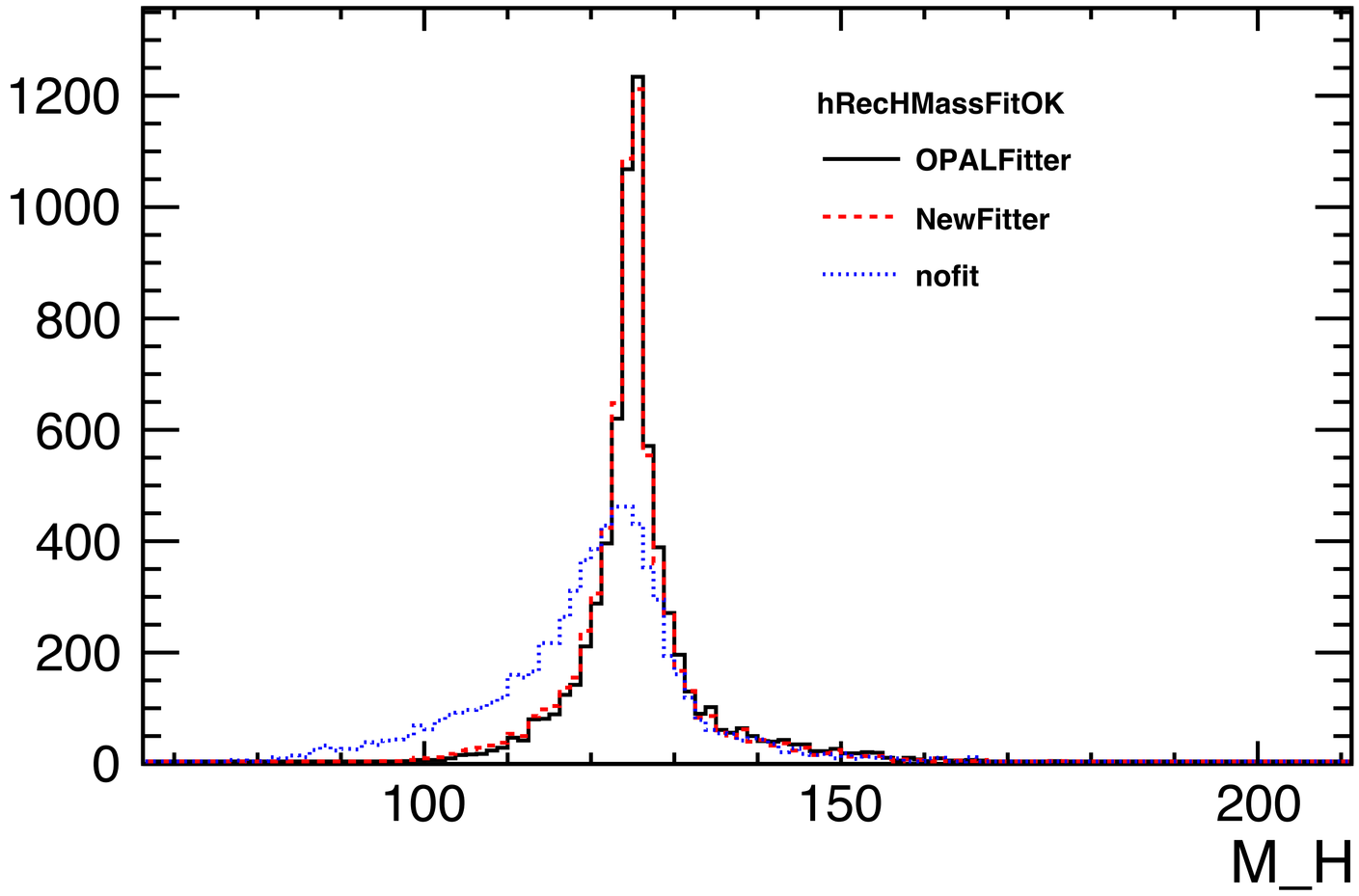
MarlinKinfitProcessors/ZHllqq5C.C

- walk-through
  - fit objects & errors
  - constraints
  - fitter
  - steering parameters
  - output to RAIDA histograms
- still work-in-progress, please ignore for today:
  - LCIO output: store fitted jets, bosons etc as ReconstructedParticles

NOW: try the first fit!
defaults:
- **ISR photon on**
- **ErrorFlow on**

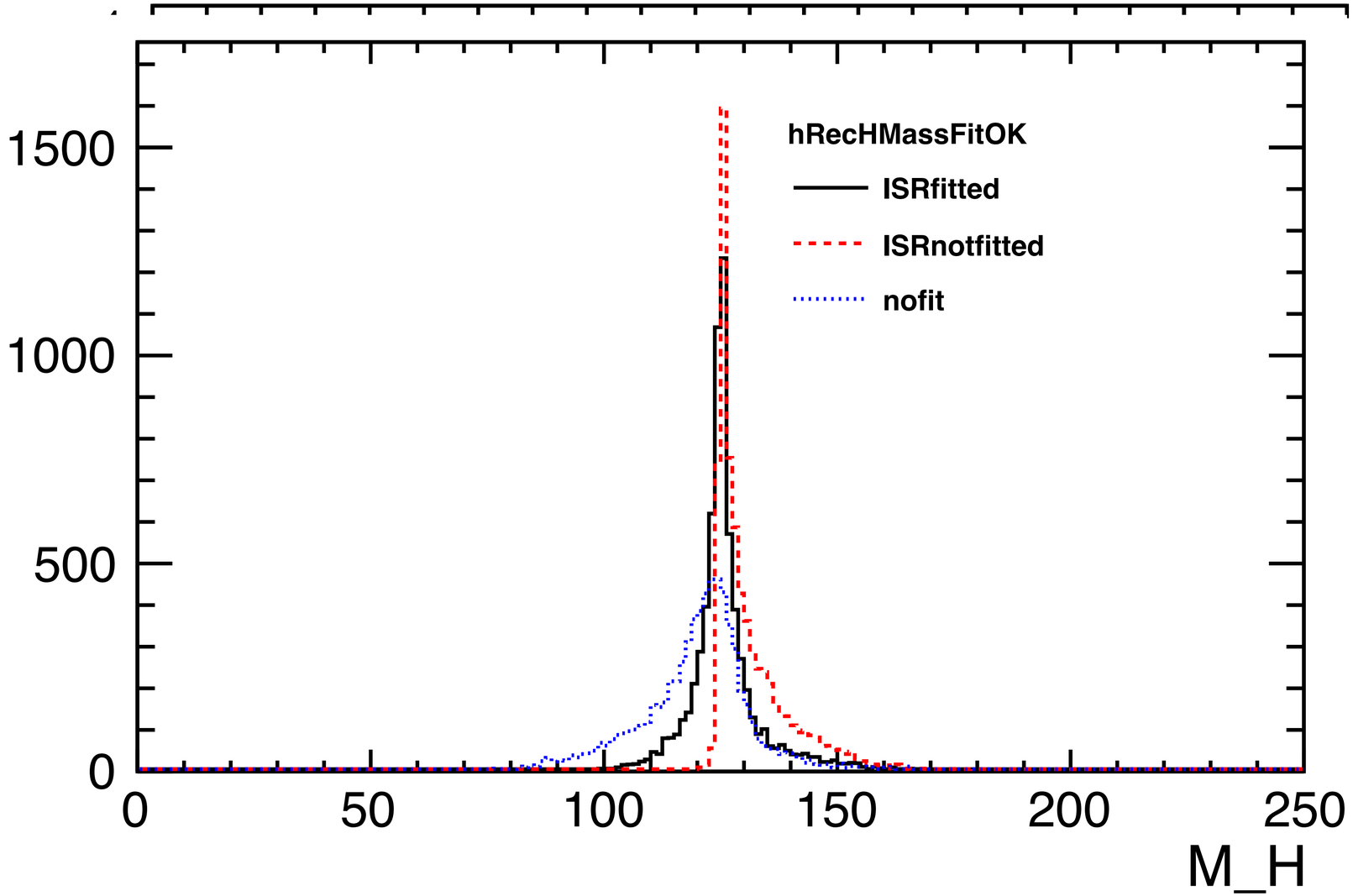
First plots

- M(bb) before and after fit, default config
- macro subdirectory has simple marcos to plot 2 or 3 histograms



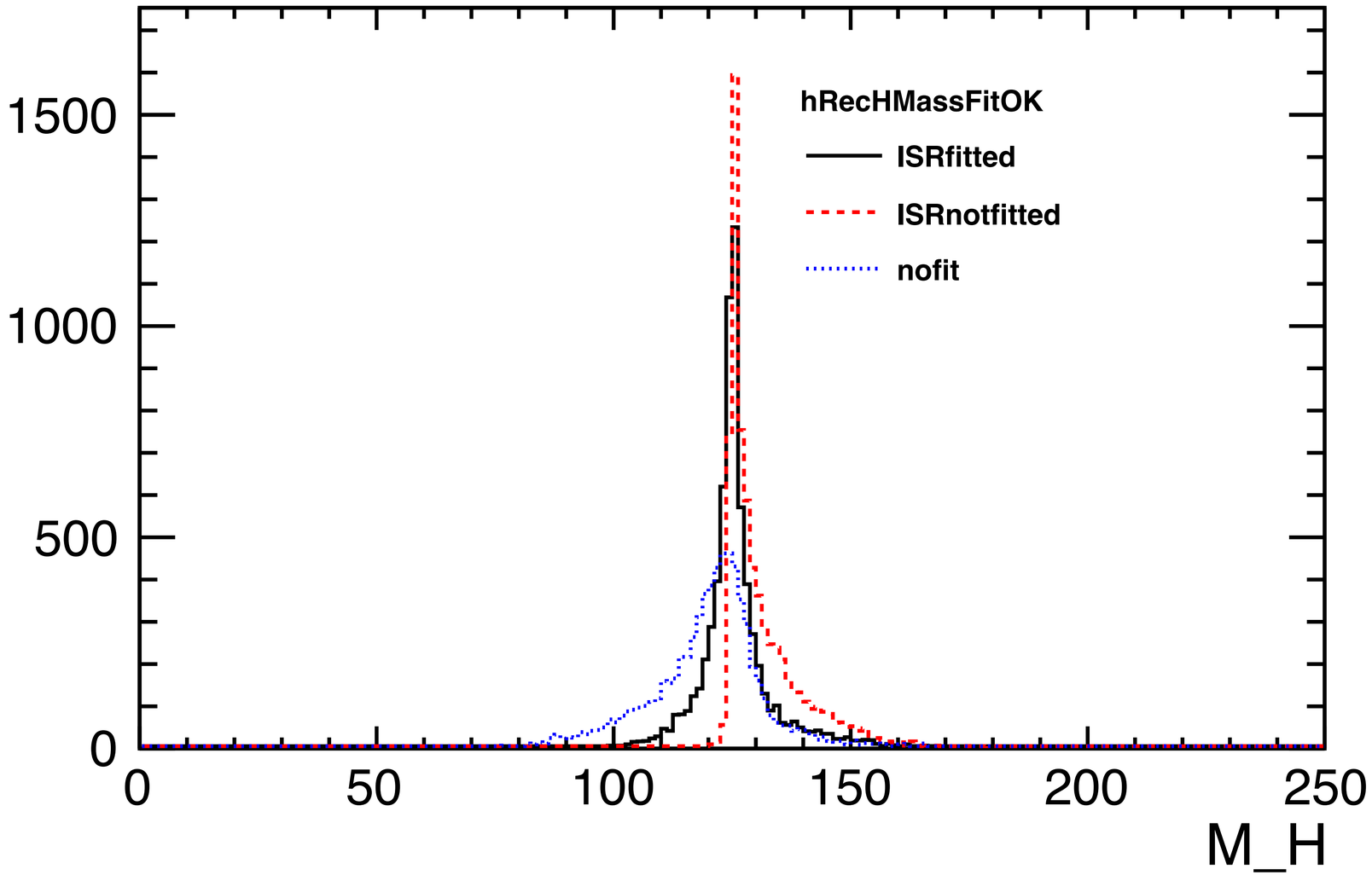
Now switch off ISR treatment and try again…

- plot eg:
  - Error code
  - Number of iteration
  - Fit probability
  - Jet pulls
  - M(bb)



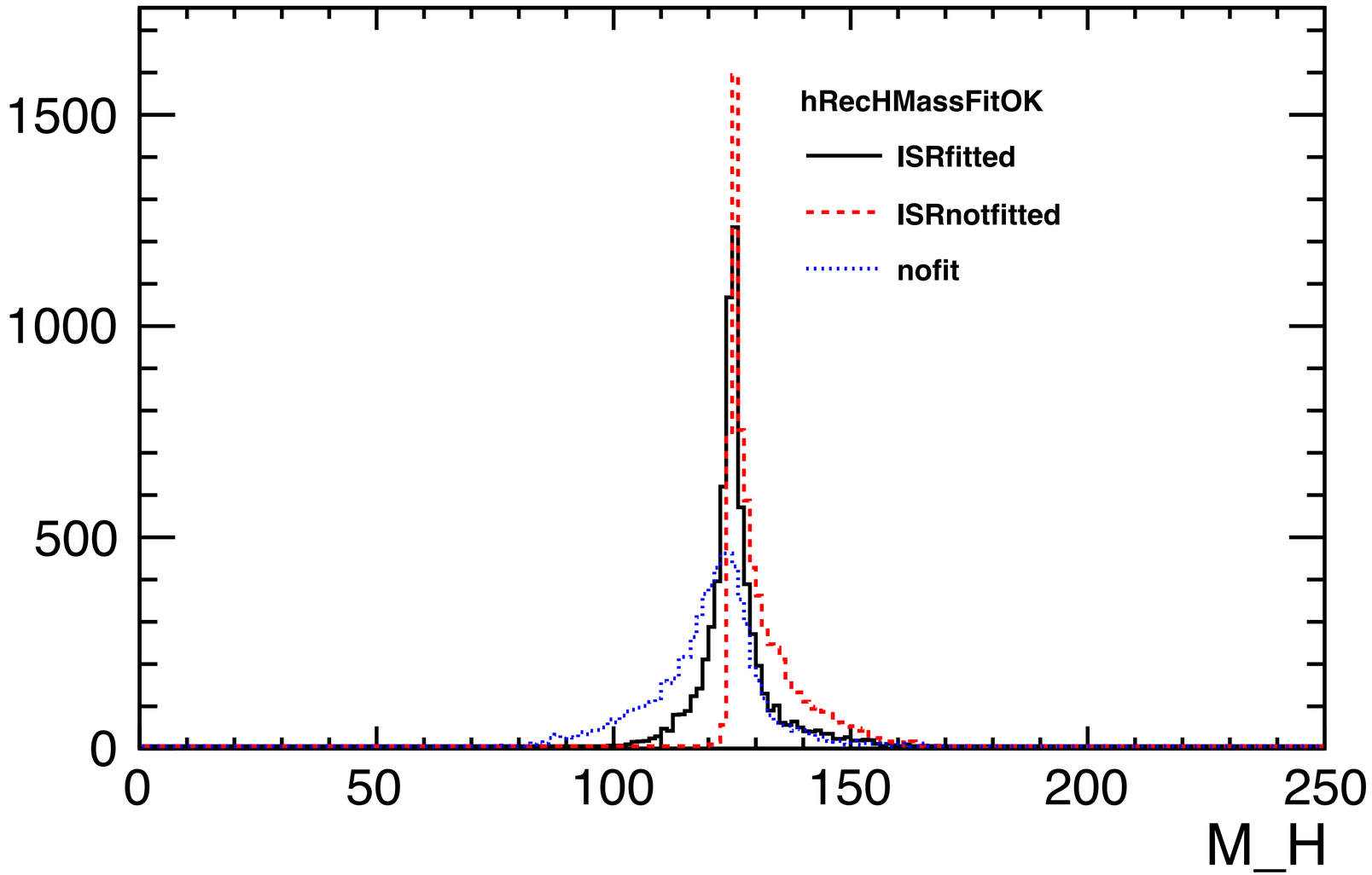
Now switch off ISR treatment and try again…

- plot eg:
  - Error code
  - Number of iteration
  - Fit probability
  - Jet pulls
  - M(bb)



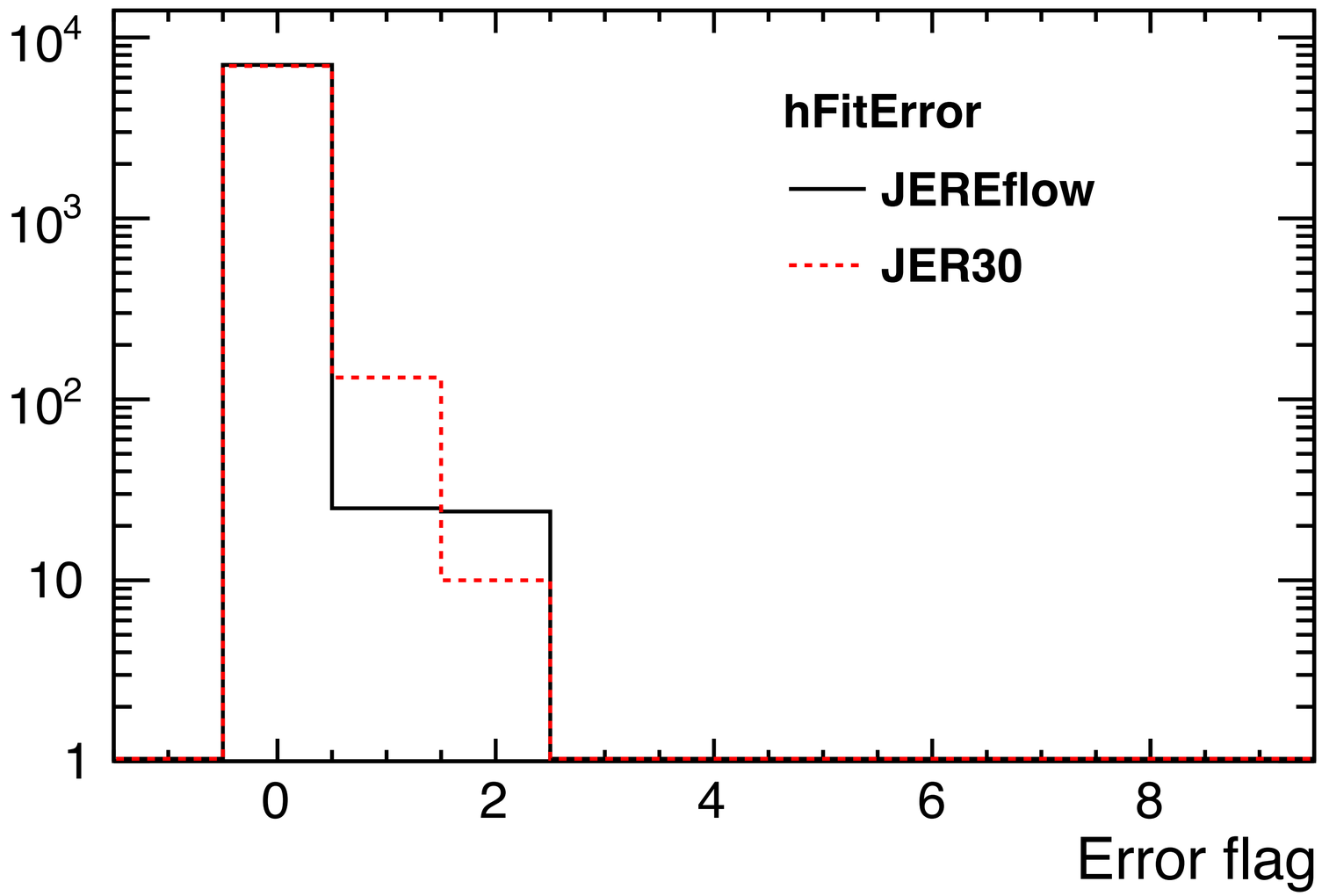
Now switch off ISR treatment and try again…

- plot eg:
  - Error code
  - Number of iteration
  - Fit probability
  - Jet pulls
  - M(bb)



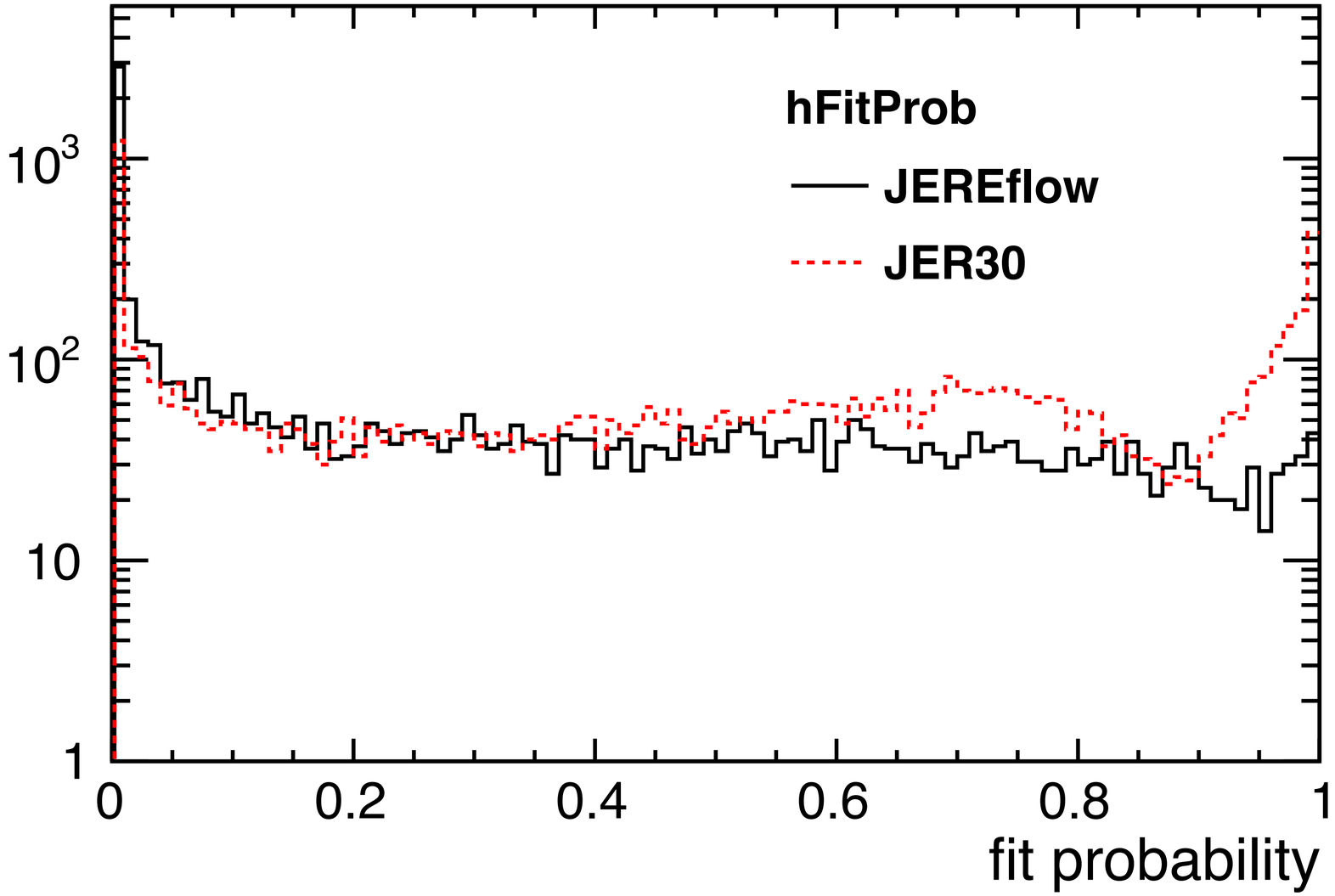
Now: ISR back on, but assume JER = 30%/sqrt(E)

- note: angular uncertainties still from ErrorFlow…
- plot eg:
  - Error code
  - Fit probability: significant part of jets measured better than 30%/sqrt(E)
  - M(bb)



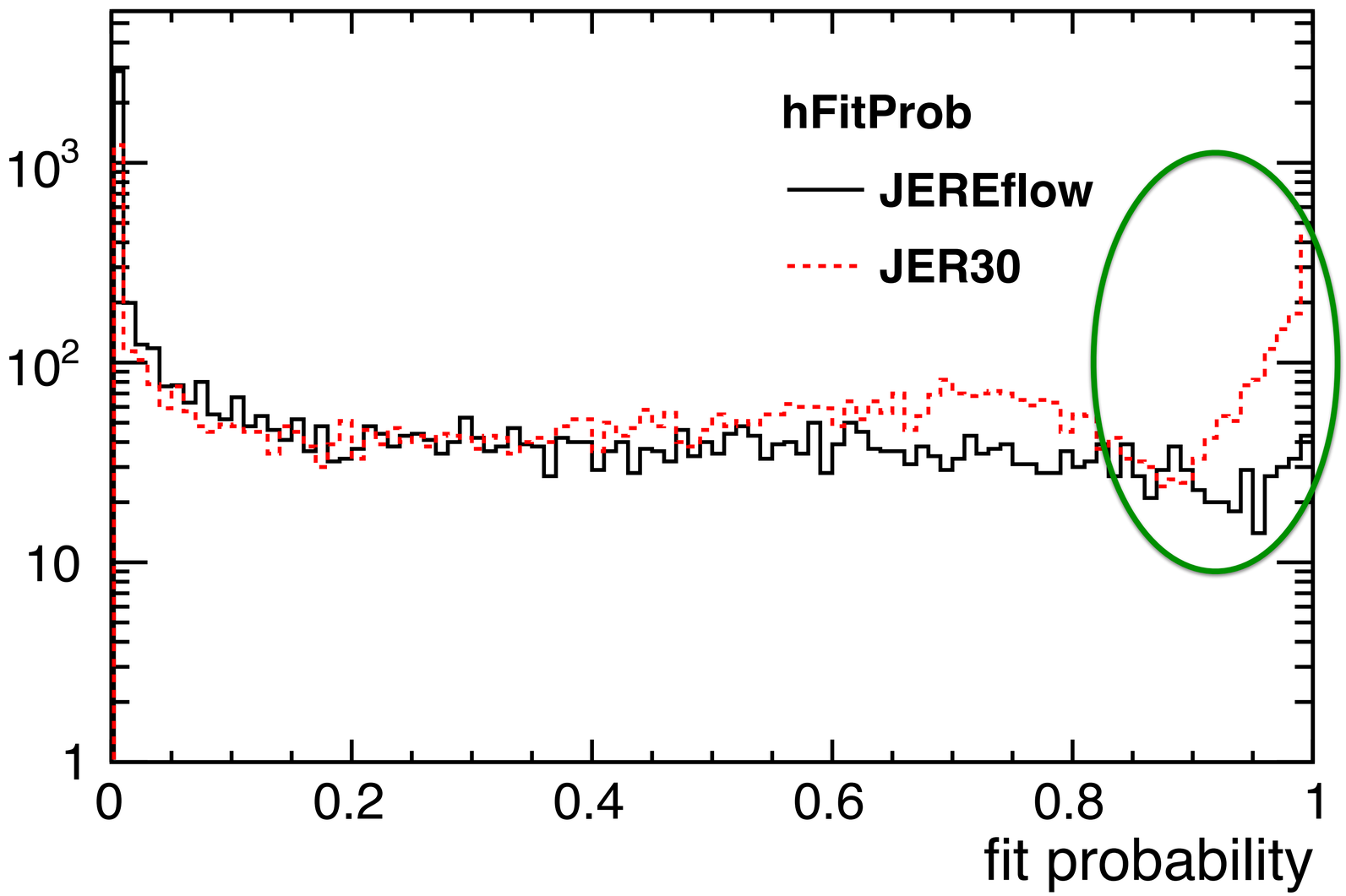
Now: ISR back on, but assume JER = 30%/sqrt(E)

- note: angular uncertainties still from ErrorFlow…
- plot eg:
  - Error code
  - Fit probability: significant part of jets measured better than 30%/sqrt(E)
  - M(bb)



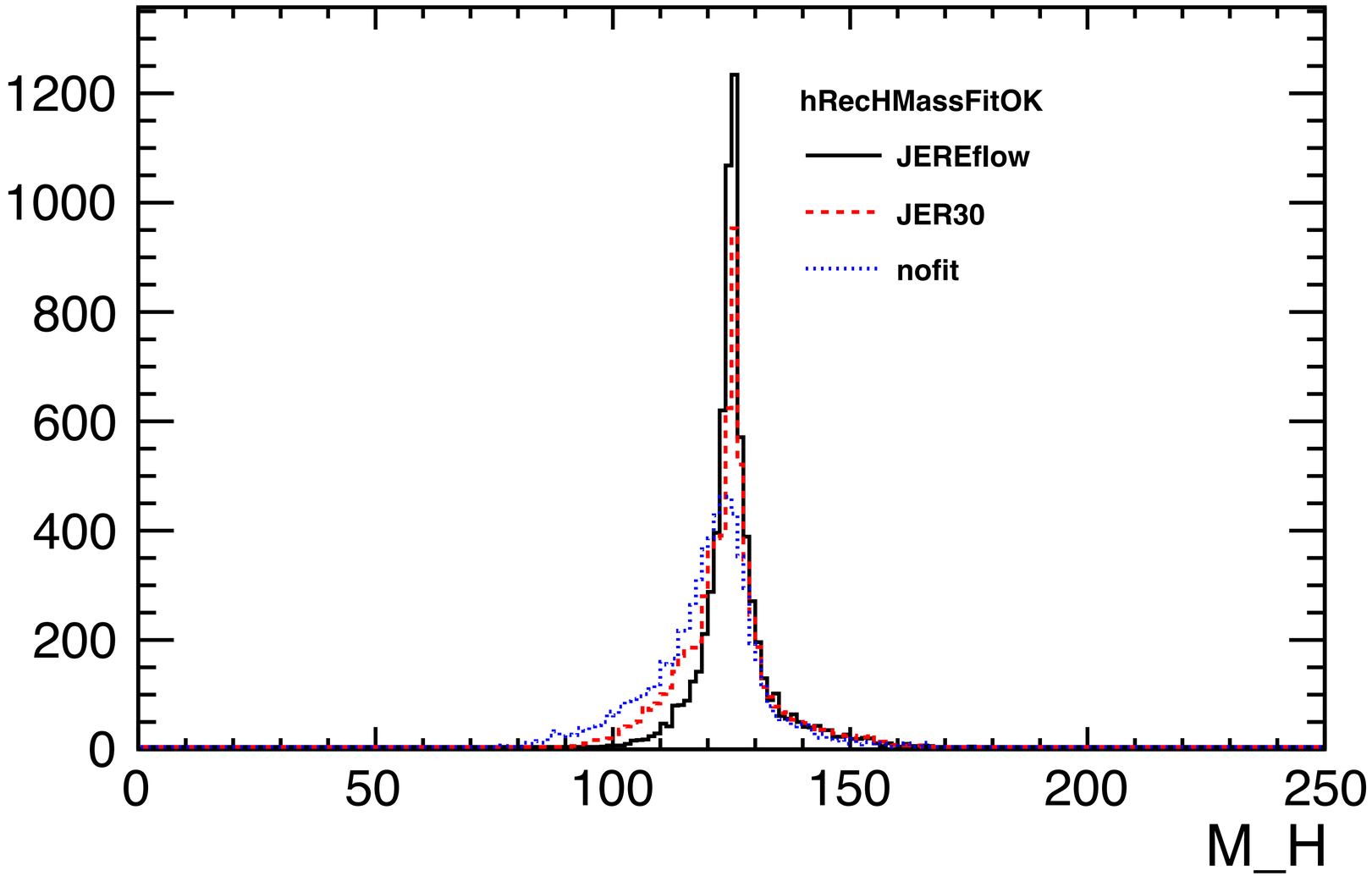
Now: ISR back on, but assume JER = 30%/sqrt(E)

- note: angular uncertainties still from ErrorFlow…
- plot eg:
  - Error code
  - Fit probability: significant part of jets measured better than 30%/sqrt(E)
  - M(bb)



Now: ISR back on, but assume JER = 30%/sqrt(E)

- note: angular uncertainties still from ErrorFlow…
- plot eg:
  - Error code
  - Fit probability: significant part of jets measured better than 30%/sqrt(E)
  - M(bb)



Problem chasing

- now switch on tracing for event 2:
  <!-- number of individual event to be traced (default: -1) -->
  <parameter name="ievttrace" type="int">-1</parameter>
- <parameter name="ievttrace" type="int"> 2 </parameter>
- look into log file!
- look into MarlinKinfit/src/TextTracer

Now it is your turn!

- try your own fit
  - with your event hypothesis
  - on your physics channel
- need help ?
  => jenny.list@desy.de

Contact information

https://linearcollider.org/team/wg3/

WG3 Subgroups

- Speakers Bureau
- Machine-Detector Interface Subgroup
- Detector and Technology R&D Subgroup
- **Software and Computing Subgroup**
- **Physics Potential and Opportunities Subgroup**

Links

- **IDT-WG3 Mandate and Workplan**
- **WG3 meeting pages**
- **Documents for Physics & Detectors**

Contact for IDT-WG3

- Chair: **Hitoshi Murayama**, UC Berkeley/U. Tokyo
- Deputy: **Jenny List**, DESY
- Deputy: **Claude Vallée**, CPPM-IN2P3