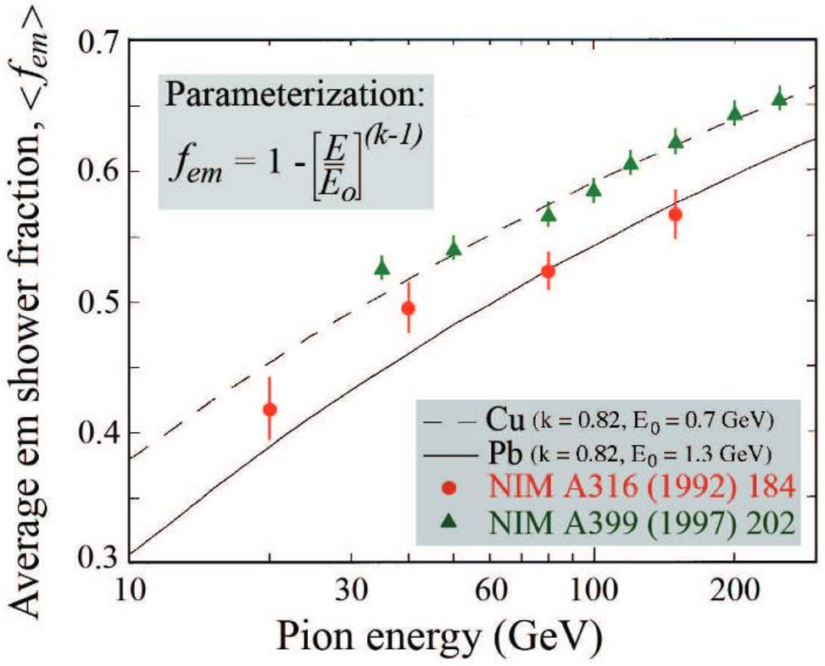
JShowerReconstruction

(from a newcomer's perspective)

Why not one all-purpose shower fitter?

- Low-energy (1-20 GeV) showers are difficult to detect!
  - Few hit coincidences (**low L1 hit probability**)

- Relatively large hadronic components
  - Vast majority (~90%) of hadronic shower particles are pions
  - Neutral pions decay into 2γ's
    --> EM showers
  - At lower shower energies, smaller EM component
    (from ~30% at 10 GeV to ~50% at 100 GeV)
  - The higher the hadronic component,
    the higher the fraction of invisible energy
    (due to nuclear interactions)



R. Wigmans – Lecture series on Calorimetry (2008)

The reconstruction chain

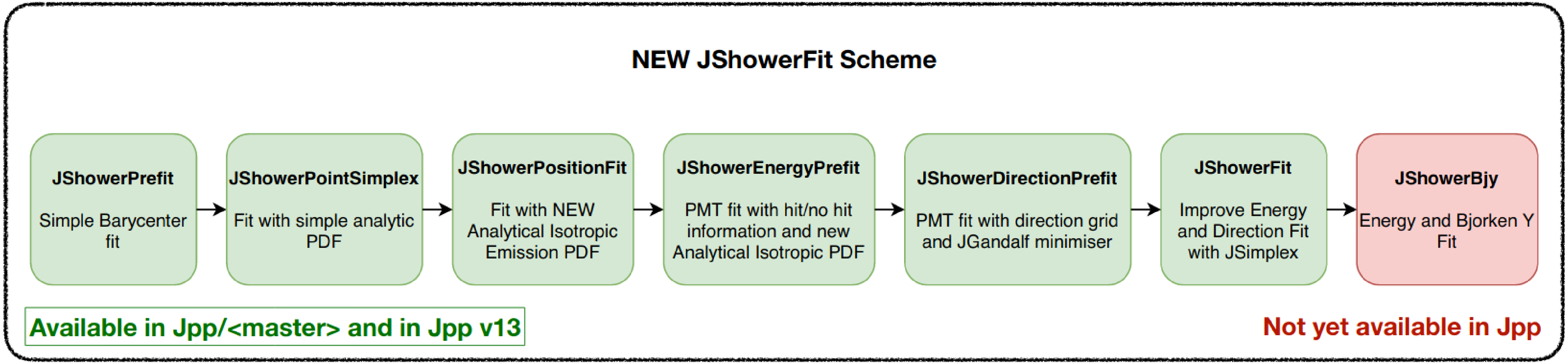
- At present, six parts:

  I. Vertex + direction prefit
  II. 1st vertex likelihood minimization (Powell's method)
  III. 2nd vertex likelihood minimization (Levenberg-Marquardt)

  Vertex

  IV. Energy prefit
  V. 2nd direction prefit
  VI. Energy and direction likelihood minimization (Powell's method)

  Energy + direction



NEW JShowerFit Scheme

JShowerPrefit
Simple Barycenter fit

JShowerPointSimplex
Fit with simple analytic PDF

JShowerPositionFit
Fit with NEW Analytical Isotropic Emission PDF

JShowerEnergyPrefit
PMT fit with hit/no hit information and new Analytical Isotropic PDF

JShowerDirectionPrefit
PMT fit with direction grid and JGandalf minimiser

JShowerFit
Improve Energy and Direction Fit with JSimplex

JShowerBjy
Energy and Bjorken Y Fit

Available in Jpp/<master> and in Jpp v13

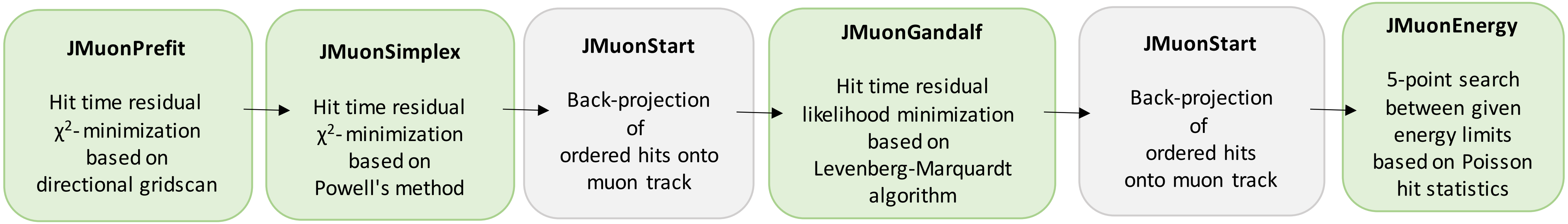
Not yet available in Jpp

From Alba's June 2020 collaboration meeting presentation

The reconstruction chain

- At present, six parts

- C.f. the JMuonReconstruction chain:



JMuonPrefit
Hit time residual χ^2- minimization based on directional gridscan

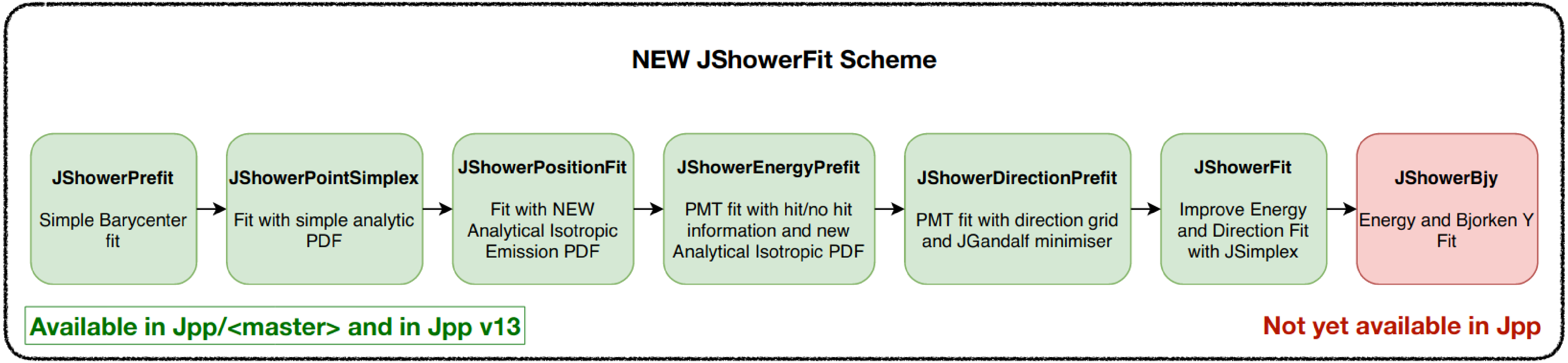
JMuonSimplex
Hit time residual χ^2- minimization based on Powell's method

JMuonStart
Back-projection of ordered hits onto muon track

JMuonGandalf
Hit time residual likelihood minimization based on Levenberg-Marquardt algorithm

JMuonStart
Back-projection of ordered hits onto muon track

JMuonEnergy
5-point search between given energy limits based on Poisson hit statistics



NEW JShowerFit Scheme

JShowerPrefit
Simple Barycenter fit

JShowerPointSimplex
Fit with simple analytic PDF

JShowerPositionFit
Fit with NEW Analytical Isotropic Emission PDF

JShowerEnergyPrefit
PMT fit with hit/no hit information and new Analytical Isotropic PDF

JShowerDirectionPrefit
PMT fit with direction grid and JGandalf minimiser

JShowerFit
Improve Energy and Direction Fit with JSimplex

JShowerBjy
Energy and Bjorken Y Fit

Available in Jpp/<master> and in Jpp v13

Not yet available in Jpp

From Alba's June 2020 collaboration meeting presentation

Vertex prefit

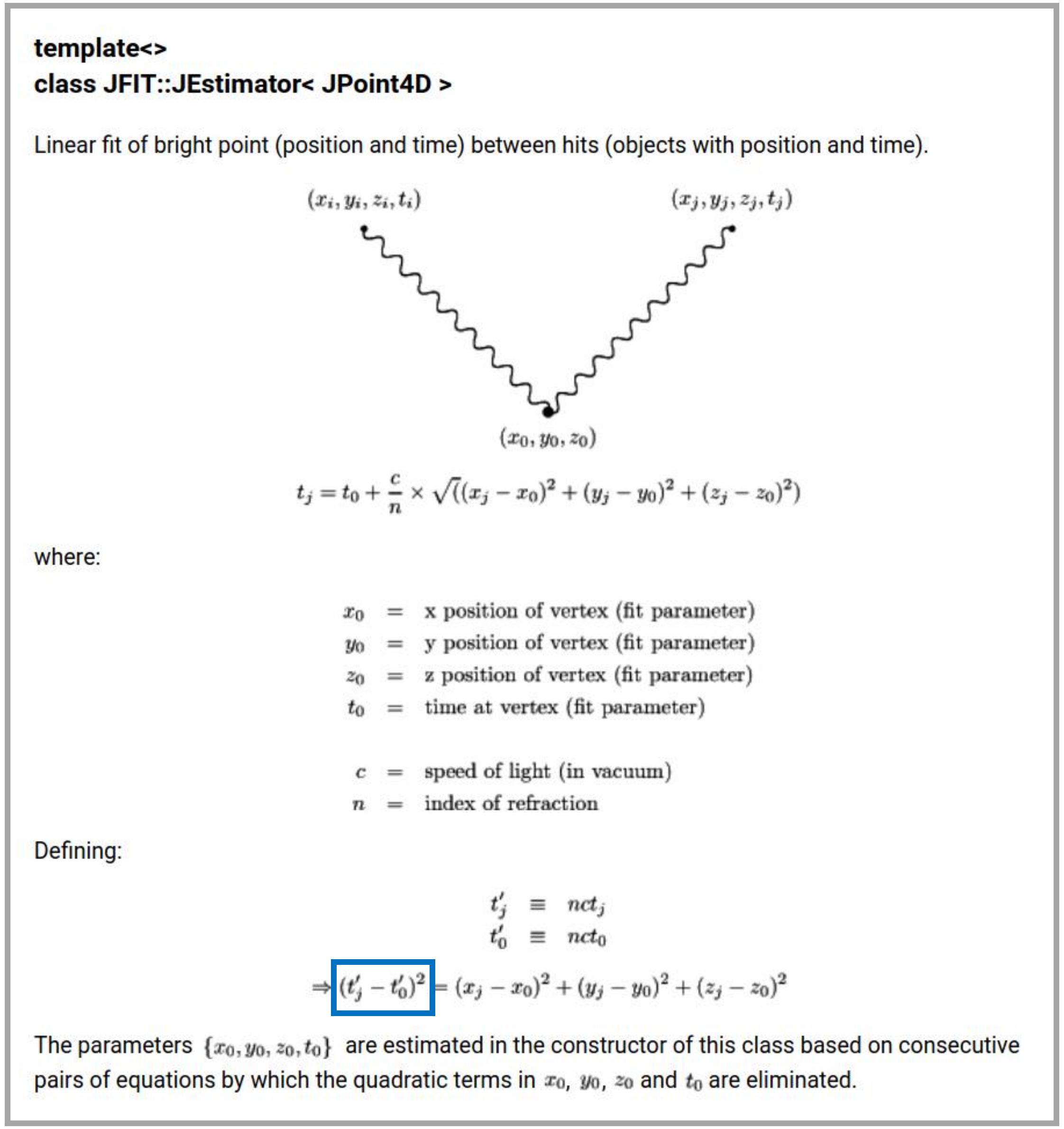
- The vertex prefit consists of roughly 2 steps:

  I. Hit selection

    - Find a cluster of causally related hits originating from a single emission point (vertex)
    - Minimize background

  II. Least squares vertex (barycenter) fit based on **time of flight**

    - Find the neutrino vertex which minimizes squared hit time residuals



template<>
class JFIT::JEstimator< JPoint4D >

Linear fit of bright point (position and time) between hits (objects with position and time).

$$t_j = t_0 + \frac{c}{n} \times \sqrt{((x_j - x_0)^2 + (y_j - y_0)^2 + (z_j - z_0)^2)}$$

where:

x_0 = x position of vertex (fit parameter)
y_0 = y position of vertex (fit parameter)
z_0 = z position of vertex (fit parameter)
t_0 = time at vertex (fit parameter)

c = speed of light (in vacuum)
n = index of refraction

Defining:

$$t'_j \equiv nct_j$$
$$t'_0 \equiv nct_0$$

$$\Rightarrow \boxed{(t'_j - t'_0)^2} = (x_j - x_0)^2 + (y_j - y_0)^2 + (z_j - z_0)^2$$

The parameters $\{x_0, y_0, z_0, t_0\}$ are estimated in the constructor of this class based on consecutive pairs of equations by which the quadratic terms in x_0, y_0, z_0 and t_0 are eliminated.

Vertex prefit (hit selection)

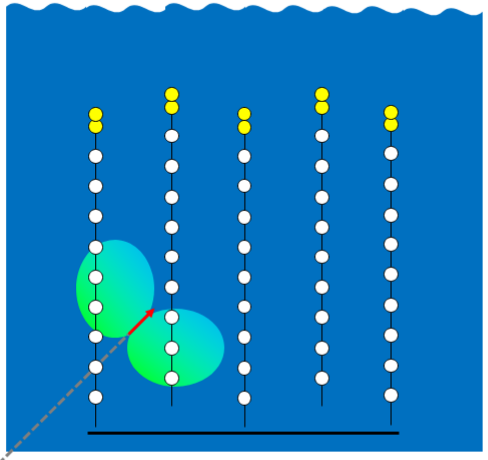
- Based on the MX (mix) trigger
  - Uses a mix of L0 and L1 hits

Hit rates:

- RL0 ~ 115 x 18 x 31 x 5 kHz = 320 MHz
- RL1 ~ 115 x 18 x 1 kHz = 2 MHz

In typical time window of ~ 100 ns, expect ~ **O(1) L1 hit** and ~ **O(10-100) L0 hits**

- Which L0's are causally related to a L1 hit?
  - Low energy showers are well-contained --> **use distance** as selection criterion



size of shower much smaller than size of detector

⇩

limit distance between L1/L0 hits

See [Maarten's presentation on JTriggerMXShower]

Vertex prefit (hit selection)

- Start with a cluster of causally connected L2s ($\Delta t < 20$ ns and $\alpha < 135$ deg) that satisfy:

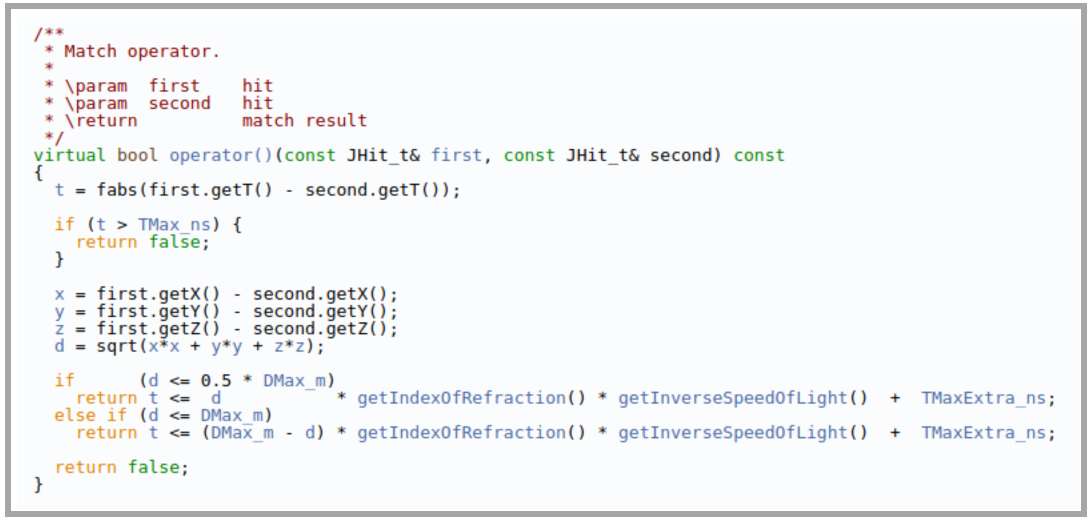
$$|t_i - t_j| \leq \boxed{|\vec{x}_i - \vec{x}_j|/c_{water}} + \boxed{T_{extra}} \quad \text{with } |\vec{x}_i - \vec{x}_j| = 50 \text{ m and } T_{extra} = 20 \text{ ns.}$$

i.e.: hit time difference smaller than **TOF between the two hit DOMs**
+ some extra time (to account for scattering and hit time resolution)

c_{water} ~ 0.29979 / 1.34 ~ 0.22 m/ns $\longrightarrow$ $|t_i - t_j| \leq$ 220 ns

- Subsequently check if any surrounding L0s are causally connected with either of the L2 hits (using same condition) and add them to the time-sorted L2 cluster

- Matching operator defined in **JMatch3G**



```
/**
 * Match operator.
 *
 * \param  first    hit
 * \param  second   hit
 * \return          match result
 */
virtual bool operator()(const JHit_t& first, const JHit_t& second) const
{
  t = fabs(first.getT() - second.getT());

  if (t > TMax_ns) {
    return false;
  }

  x = first.getX() - second.getX();
  y = first.getY() - second.getY();
  z = first.getZ() - second.getZ();
  d = sqrt(x*x + y*y + z*z);

  if      (d <= 0.5 * DMax_m)
    return t <=  d          * getIndexOfRefraction() * getInverseSpeedOfLight()  +  TMaxExtra_ns;
  else if (d <= DMax_m)
    return t <= (DMax_m - d) * getIndexOfRefraction() * getInverseSpeedOfLight()  +  TMaxExtra_ns;

  return false;
}
```

Vertex prefit (fitting algorithm)

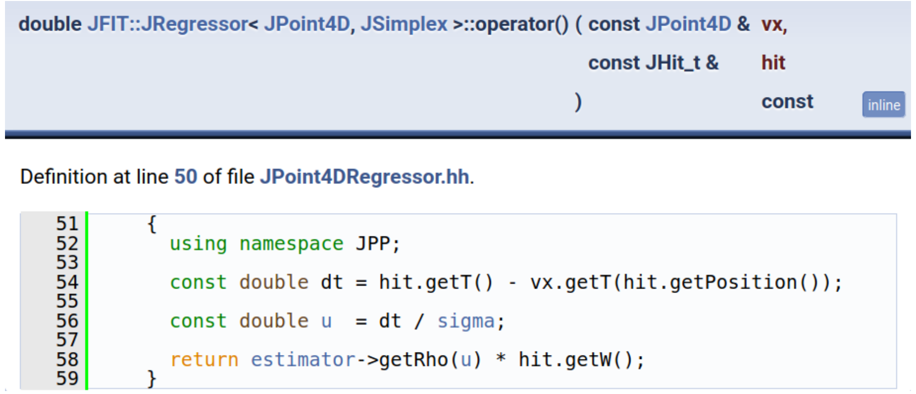
- The least squares fit for the arrival time residuals is largely analogous to what is done in JMuonPrefit
    - See also the appendix of Brían's track reconstruction document
- But it involves additional iterative steps, meant to enhance the precision (whilst limiting CPU time!)

1. An extra distinction based on L2 cluster size:
    - If total L2+L0 hits, $N_{tot} < 41$:
        1. Repeat the chi2-minimization $N_{tot} - 4$ times, removing 1 L0 hit in each iteration
        2. Choose the fit which yields the least chi2
    - If total L2+L0 hits, Ntot > 40:
        1. Perform the chi2-minimization just once, using all hits within the cluster

JSimplex

- JSimplex performs a first, simple likelihood minimization, using **Powell's method:**
    - Iteratively finds a set of N mutually **conjugate directions**, with which to minimize the likelihood landscape
    - Does not require any gradients!
- For each step, the likelihood is re-evaluated:

$\prod$ hits

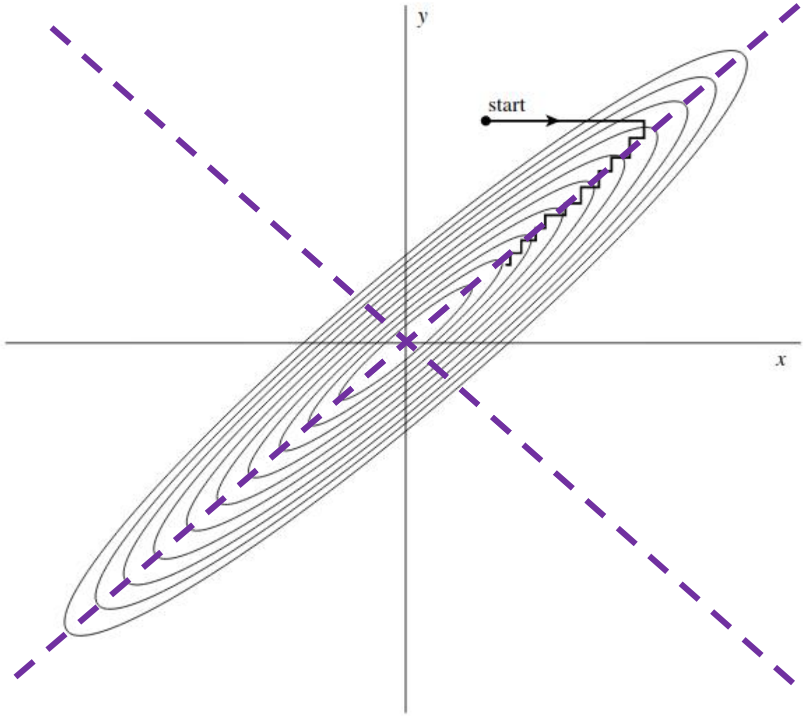


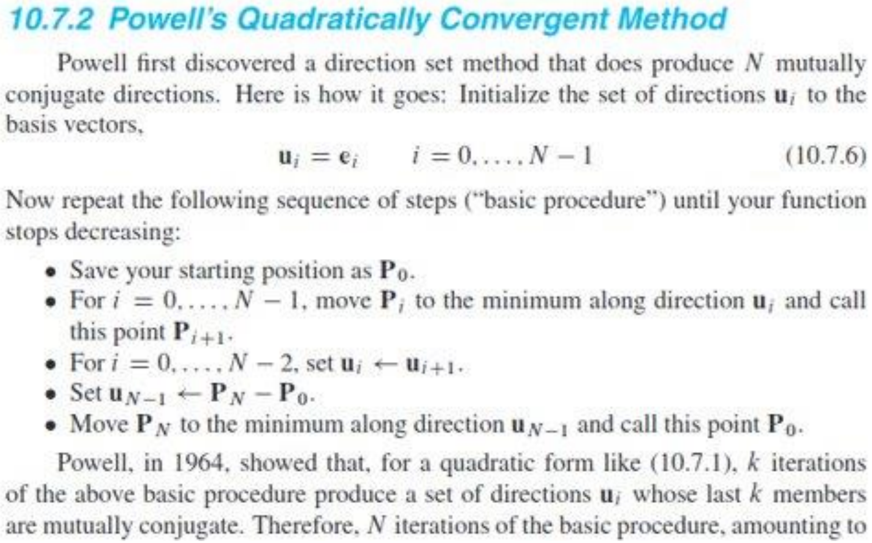
double JFIT::JRegressor< JPoint4D, JSimplex >::operator() (const JPoint4D & vx,
const JHit_t & hit
) const inline

Definition at line 50 of file JPoint4DRegressor.hh.

```
{
  using namespace JPP;
  const double dt = hit.getT() - vx.getT(hit.getPosition());
  const double u  = dt / sigma;
  return estimator->getRho(u) * hit.getW();
}
```

- Terminate when fractional improvement becomes negligible (default 1e-4)





10.7.2 Powell's Quadratically Convergent Method

Powell first discovered a direction set method that does produce N mutually conjugate directions. Here is how it goes: Initialize the set of directions $\mathbf{u}_i$ to the basis vectors,

$$\mathbf{u}_i = \mathbf{e}_i \qquad i = 0, \ldots, N-1 \tag{10.7.6}$$

Now repeat the following sequence of steps ("basic procedure") until your function stops decreasing:

- Save your starting position as $\mathbf{P}_0$.
- For $i = 0, \ldots, N-1$, move $\mathbf{P}_i$ to the minimum along direction $\mathbf{u}_i$ and call this point $\mathbf{P}_{i+1}$.
- For $i = 0, \ldots, N-2$, set $\mathbf{u}_i \leftarrow \mathbf{u}_{i+1}$.
- Set $\mathbf{u}_{N-1} \leftarrow \mathbf{P}_N - \mathbf{P}_0$.
- Move $\mathbf{P}_N$ to the minimum along direction $\mathbf{u}_{N-1}$ and call this point $\mathbf{P}_0$.

Powell, in 1964, showed that, for a quadratic form like (10.7.1), k iterations of the above basic procedure produce a set of directions $\mathbf{u}_i$ whose last k members are mutually conjugate. Therefore, N iterations of the basic procedure, amounting to

Press, WH et al. (2007); Numerical Recipes: The Art of Scientific Computing (3rd ed.)

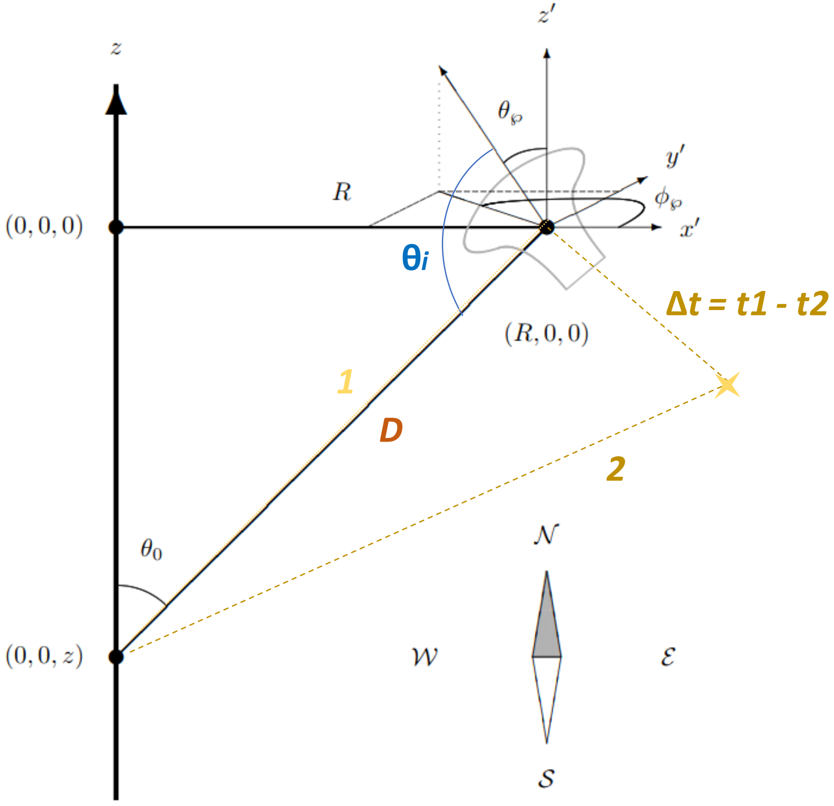
JShowerPositionFit

- The fits from JPrefit and JSimplex should now be sufficiently close to the true optimum, to allow for a final, full MLE, including non-linear effects such as scattering

- JShowerPositionFit constitutes this final minimization step
  - Exploits the analytical arrival time PDFs, dependent on:
    i. **Δt**: **the arrival time residual**
    ii. D: the **(nu-vertex, PMT)-distance**
    iii. Cos(**θi**): **cosine angle of incidence**

- The fit probability is defined as:
  - P = E * PDF(D, cos(**θi**), **Δ**t)

- Minimization is based on the **Levenberg-Marquard Algorithm**
  - Requires analytical calculation of the Hessian matrix



JPDF documentation – Maarten de Jong (see doxygen)

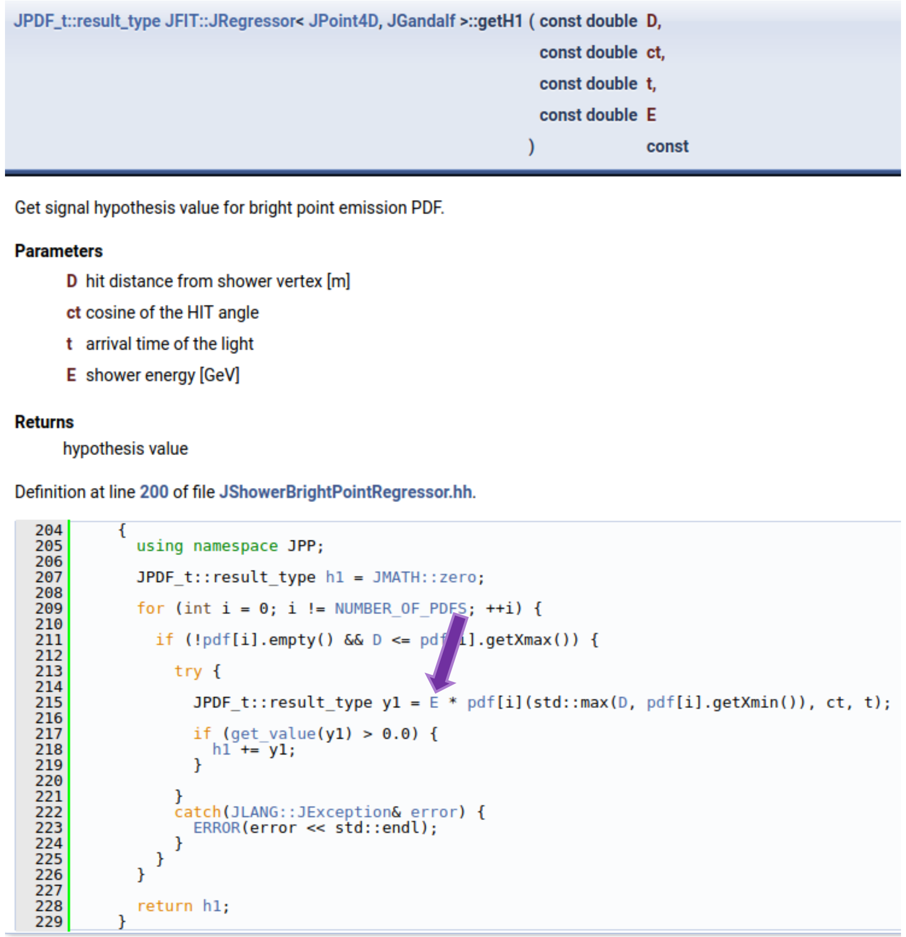
JShowerPositionFit

- The fits from JPrefit and JSimplex should now be sufficiently close to the true optimum, to allow for a final, full MLE, including non-linear effects such as scattering

- JShowerPositionFit constitutes this final minimization step
  - Exploits the analytical arrival time PDFs, dependent on:
    - **i.** **Δ**t: **the arrival time residual**
    - ii. D: the **(nu-vertex, PMT)-distance**
    - iii. Cos(**θi**): **cosine angle of incidence**

- The fit probability is defined as:
  - P = **E** * PDF(D, cos(**θi**), **Δ**t)

- Minimization is based on the **Levenberg-Marquard Algorithm**
  - Requires analytical calculation of the Hessian matrix



JPDF_t::result_type JFIT::JRegressor< JPoint4D, JGandalf >::getH1 (const double D,
const double ct,
const double t,
const double E
) const

Get signal hypothesis value for bright point emission PDF.

Parameters

D	hit distance from shower vertex [m]
ct	cosine of the HIT angle
t	arrival time of the light
E	shower energy [GeV]

Returns

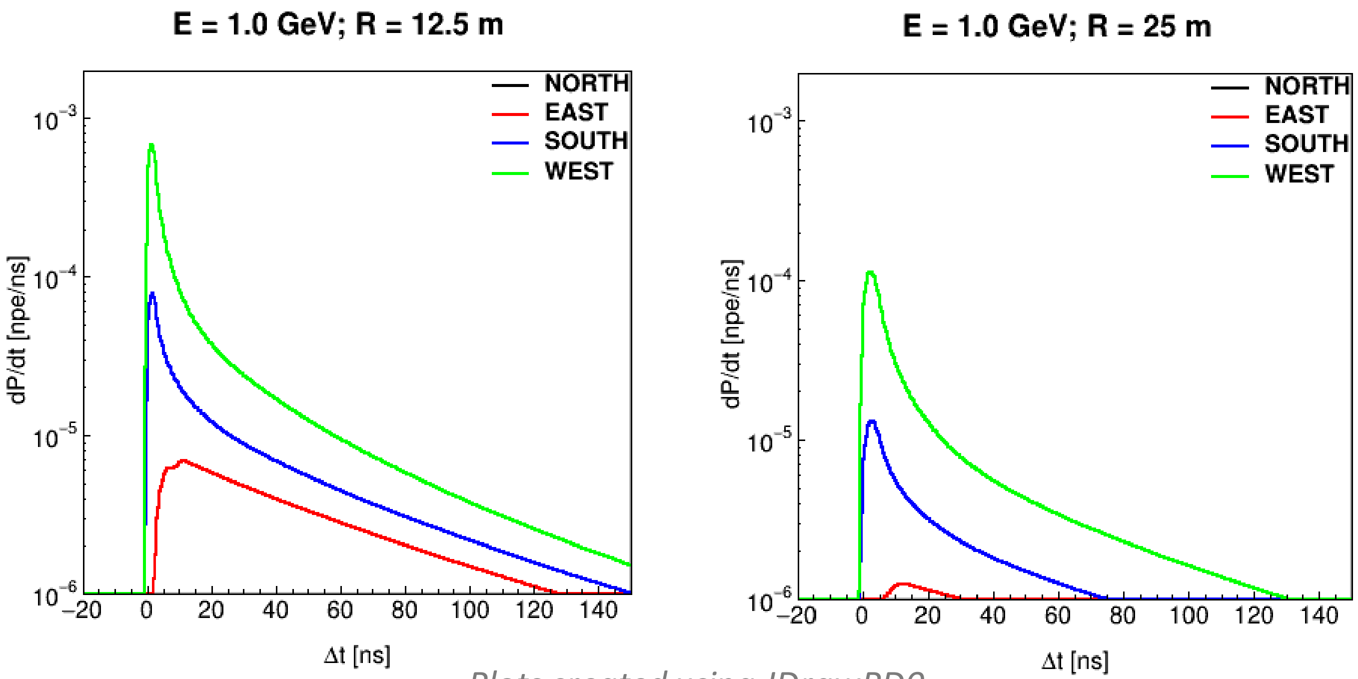
hypothesis value

Definition at line 200 of file JShowerBrightPointRegressor.hh.

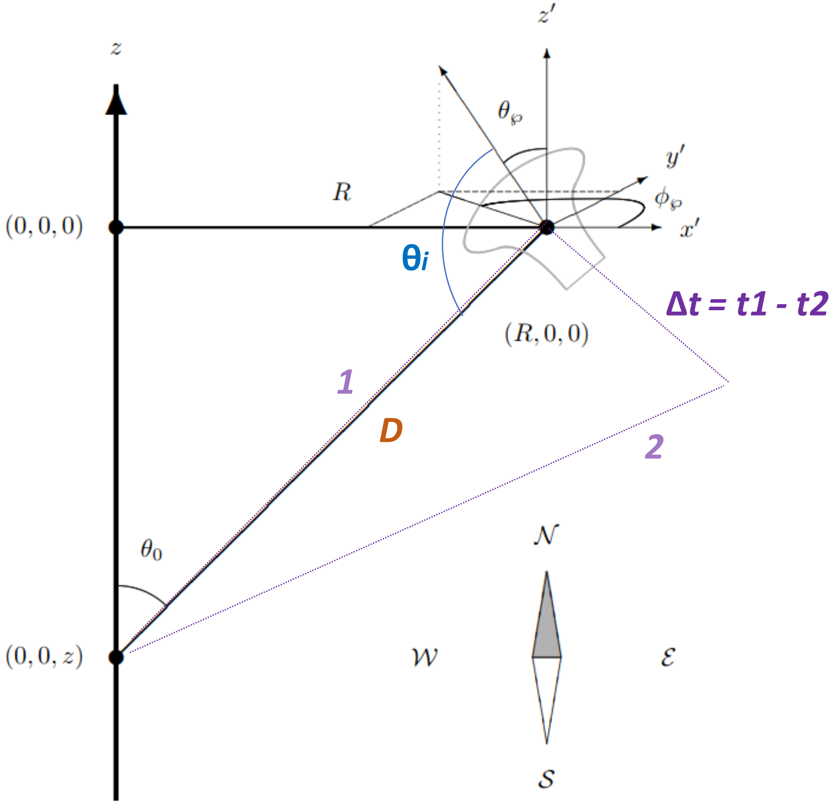
```
{
  using namespace JPP;

  JPDF_t::result_type h1 = JMATH::zero;

  for (int i = 0; i != NUMBER_OF_PDFS; ++i) {

    if (!pdf[i].empty() && D <= pdf[i].getXmax()) {

      try {

        JPDF_t::result_type y1 = E * pdf[i](std::max(D, pdf[i].getXmin()), ct, t);

        if (get_value(y1) > 0.0) {
          h1 += y1;
        }

      }
      catch(JLANG::JException& error) {
        ERROR(error << std::endl);
      }
    }
  }

  return h1;
}
```

JShowerPositionFit

- The fits from JPrefit and JSimplex should now be sufficiently close to the true optimum, to allow for a final, full MLE, including non-linear effects such as scattering
- JShowerPositionFit constitutes this final minimization step



Plots created using JDrawPD0



JPDF documentation – Maarten de Jong (see doxygen)

JShowerEnergyPrefit

- Similar in working to JMuonEnergy
  - Energy estimation based on a binomial hit/no-hit likelihood estimation

- Uses the Jpp isotropic emission PDFs to calculate the hit probabilities, given a PMT at distance D from the vertex, with photon incidence angle θi, originating from a shower with total energy E

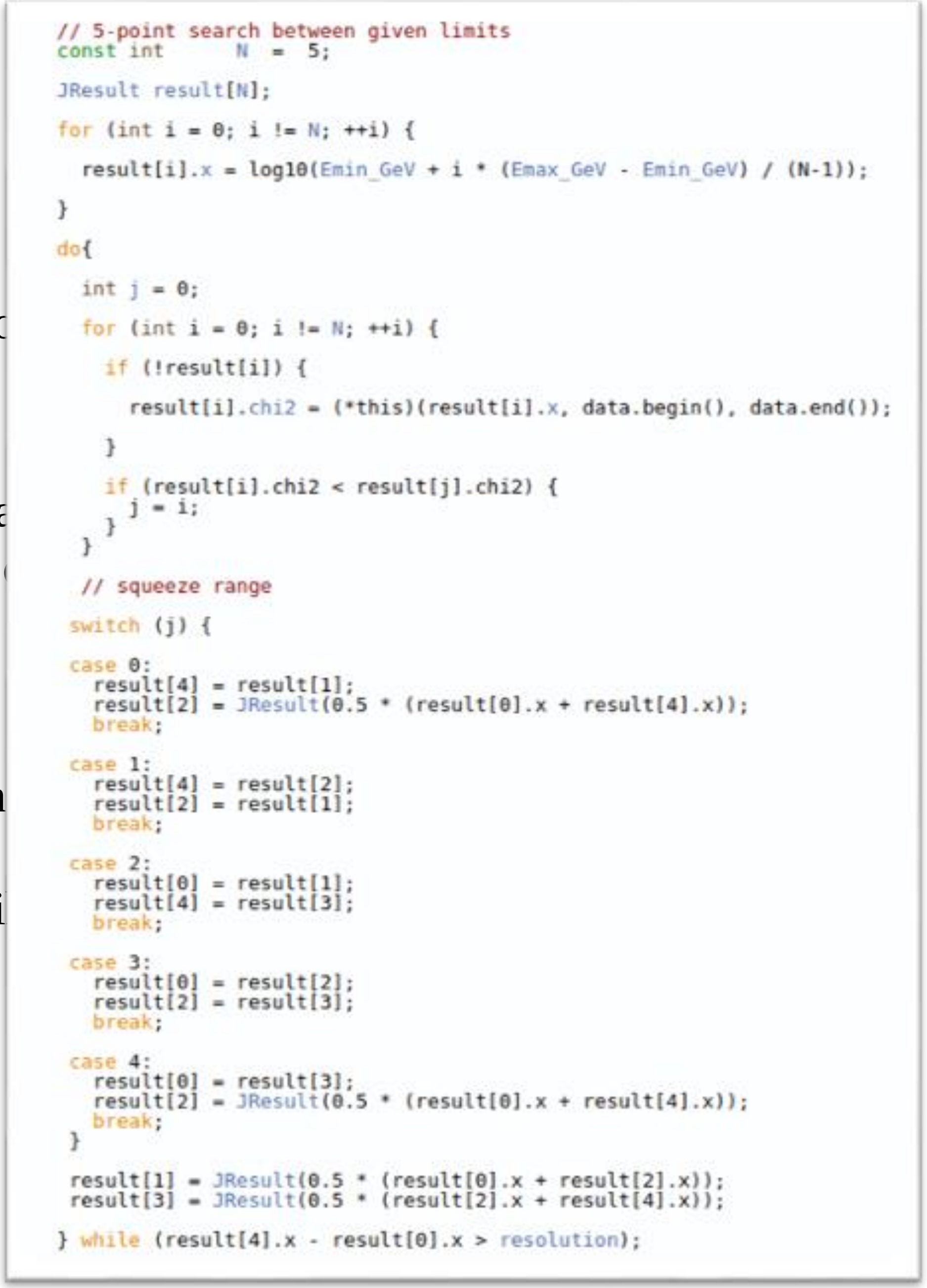
- An iterative five-point search is performed between a given energy range
  - In each iteration, pick the point with maximal hit/no-hit likelihood and refine the five points

JShowerEnergyPrefit

- Similar in working to JMuonEnergy
    - Energy estimation based on a binomial hit/no-hit likeliho

- Uses the Jpp isotropic emission PDFs to calculate the hit proba
  at distance D from the vertex, with photon incidence angle θi,

- An iterative five-point search is performed between a given en
    - In each iteration, pick the point with maximal hit/no-hit li
      and refine the five points



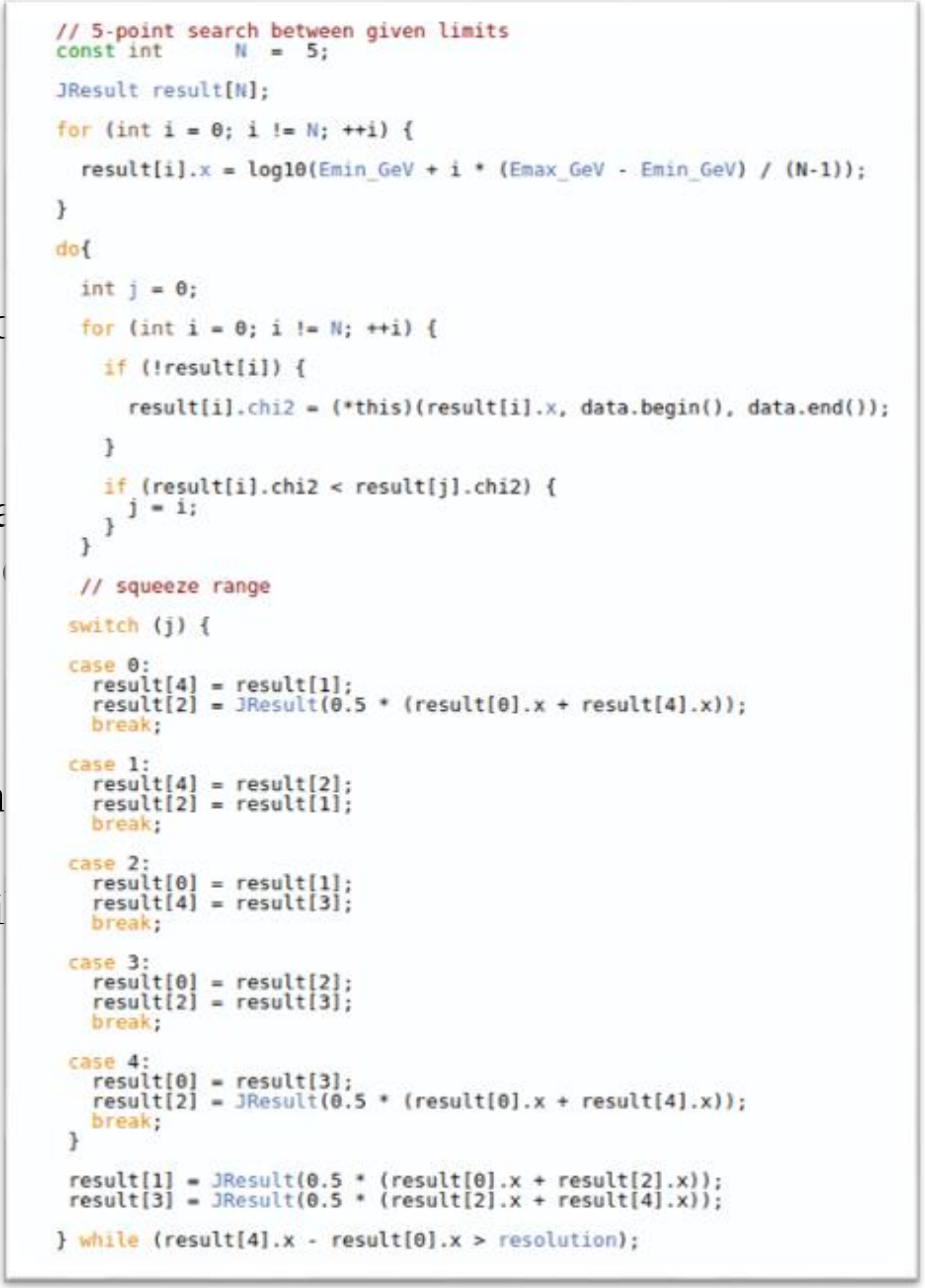
```
// 5-point search between given limits
const int      N  =  5;

JResult result[N];

for (int i = 0; i != N; ++i) {

  result[i].x = log10(Emin_GeV + i * (Emax_GeV - Emin_GeV) / (N-1));

}

do{

  int j = 0;

  for (int i = 0; i != N; ++i) {

    if (!result[i]) {

      result[i].chi2 = (*this)(result[i].x, data.begin(), data.end());

    }

    if (result[i].chi2 < result[j].chi2) {
      j = i;
    }
  }

  // squeeze range

 switch (j) {

 case 0:
   result[4] = result[1];
   result[2] = JResult(0.5 * (result[0].x + result[4].x));
   break;

 case 1:
   result[4] = result[2];
   result[2] = result[1];
   break;

 case 2:
   result[0] = result[1];
   result[4] = result[3];
   break;

 case 3:
   result[0] = result[2];
   result[2] = result[3];
   break;

 case 4:
   result[0] = result[3];
   result[2] = JResult(0.5 * (result[0].x + result[4].x));
   break;
 }

 result[1] = JResult(0.5 * (result[0].x + result[2].x));
 result[3] = JResult(0.5 * (result[2].x + result[4].x));

} while (result[4].x - result[0].x > resolution);
```

JShowerDirectionPrefit

- Similar in working to JMuonEnergy
  - Energy estimation based on a binomial hit/no-hit likeliho

- Uses the Jpp isotropic emission PDFs to calculate the hit proba
  at distance D from the vertex, with photon incidence angle θi,

- An iterative five-point search is performed between a given en
  - In each iteration, pick the point with maximal hit/no-hit li
    and refine the five points

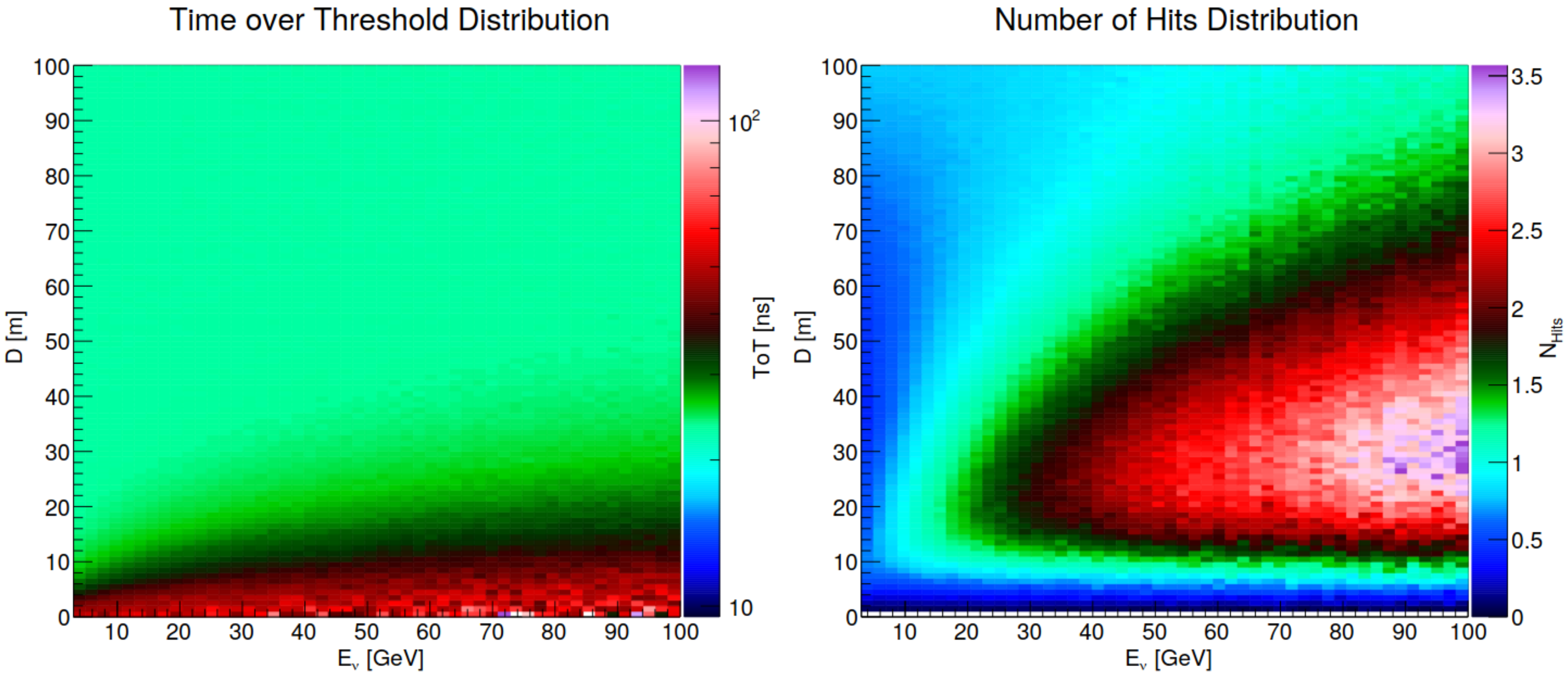


```
// 5-point search between given limits
const int      N  =  5;

JResult result[N];

for (int i = 0; i != N; ++i) {

  result[i].x = log10(Emin_GeV + i * (Emax_GeV - Emin_GeV) / (N-1));

}

do{

  int j = 0;

  for (int i = 0; i != N; ++i) {

    if (!result[i]) {

      result[i].chi2 = (*this)(result[i].x, data.begin(), data.end());

    }

    if (result[i].chi2 < result[j].chi2) {
      j = i;
    }
  }

  // squeeze range

 switch (j) {

 case 0:
   result[4] = result[1];
   result[2] = JResult(0.5 * (result[0].x + result[4].x));
   break;

 case 1:
   result[4] = result[2];
   result[2] = result[1];
   break;

 case 2:
   result[0] = result[1];
   result[4] = result[3];
   break;

 case 3:
   result[0] = result[2];
   result[2] = result[3];
   break;

 case 4:
   result[0] = result[3];
   result[2] = JResult(0.5 * (result[0].x + result[4].x));
   break;
 }

 result[1] = JResult(0.5 * (result[0].x + result[2].x));
 result[3] = JResult(0.5 * (result[2].x + result[4].x));

} while (result[4].x - result[0].x > resolution);
```

Some slides by Brían on direction fit

Bonus

ToT vs number of hits



(a) (b)

Figure 5.5: (a) ToT and (b) number of hits distributions in function of neutrino energy and distance from the neutrino interaction vertex.

Alba Domini – Ph.D. Thesis (2019)

JPrefit – Mathematical foundation

- Define the difference between two hit arrival times:

$$\begin{aligned} t_j'^2 - t_i'^2 - 2(t_j' - t_j')t_0' &= x_j^2 - x_i^2 - 2(x_j - x_j)x_0 \\ &+ y_j^2 - y_i^2 - 2(y_j - y_j)y_0 \,. \end{aligned}$$

- Casting this into matrix form, $H\,\Theta = Y$:

$$H = \begin{pmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) & -2(t_2' - t_1') \\ 2(x_3 - x_2) & 2(y_3 - y_2) & -2(t_3' - t_2') \\ . & . & . \\ . & . & . \\ 2(x_1 - x_n) & 2(y_1 - y_n) & -2(t_1' - t_n') \end{pmatrix}, \qquad \Theta = \begin{pmatrix} x_0 \\ y_0 \\ t_0' \end{pmatrix} \qquad \longrightarrow \qquad Y = \begin{pmatrix} x_2^2 - x_1^2 + y_2^2 - y_1^2 - t_2'^2 + t_1'^2 \\ x_3^2 - x_2^2 + y_3^2 - y_2^2 - t_3'^2 + t_2'^2 \\ . \\ . \\ x_1^2 - x_n^2 + y_1^2 - y_n^2 - t_1'^2 + t_n'^2 \end{pmatrix},$$

- To which the least squares solution is:

$$\boxed{\Theta = \left(H^T V^{-1} H\right)^{-1} \times H^T V^{-1} \times Y} \qquad V = J \times \begin{pmatrix} \sigma_1^2 & . & . & . & 0 \\ . & \sigma_2^2 & . & . & . \\ . & . & \sigma_3^2 & . & \\ . & . & . & . & . \\ 0 & . & . & . & \sigma_n^2 \end{pmatrix} \times J^T \,. \qquad J_{ij} = \delta Y_i \;/\; \delta t_j \,.$$

with V for the correlation matrix

Vertex prefit (fitting algorithm)

- Secondary grid scan of vertex hypotheses around the initial reconstructed vertex from step 1. (outcome of step 1. still very sensitive to inclusion of background hits)
  - From –20 m to +20 m in 3 steps for (x,y,z)
  - From –50 ns to +50 ns in 4 steps for (t)
- For each of these, make an L0-only hit cluster, with D < 80 m and Δt < 100 ns
- The resulting hit clusters are fitted using an analytical PDF (see Janník's thesis) which penalizes PMT hits not oriented towards the assumed vertex position:

$$P(t_{res}, \psi) = \frac{1}{\sqrt{a^2 + (t_{res}/ns)^2 + p(\psi)}}$$

$$p(\psi) = \begin{cases} 0 & \text{if } \cos\psi < 0 \\ p_{max} & \text{if } \cos\psi > \cos\psi_{th} \\ p_{max} \times \cos\psi / \cos\psi_{th} & \text{else.} \end{cases}$$

$$\cos\psi_{th} = \frac{d_{ref}}{\sqrt{d_{ref}^2 + d^2}}$$

Maximal penalty angle

Prevent singularity

Hit time residual

Penalty function