Vertexing Hands-on Session

Belle II Physics Week

Francesco Tenchini
October 31st, 2019

Before starting

- This tutorial assumes basic familiarity with basf2 and access to KEKCC.
  - As long as you're able to run a steering file, you'll be fine.

- If you plan to follow along with the exercises, please **`ssh login.cc.kek.jp`** and set up basf2
  - `source /cvmfs/belle.cern.ch/tools/b2setup release-04-00-01`

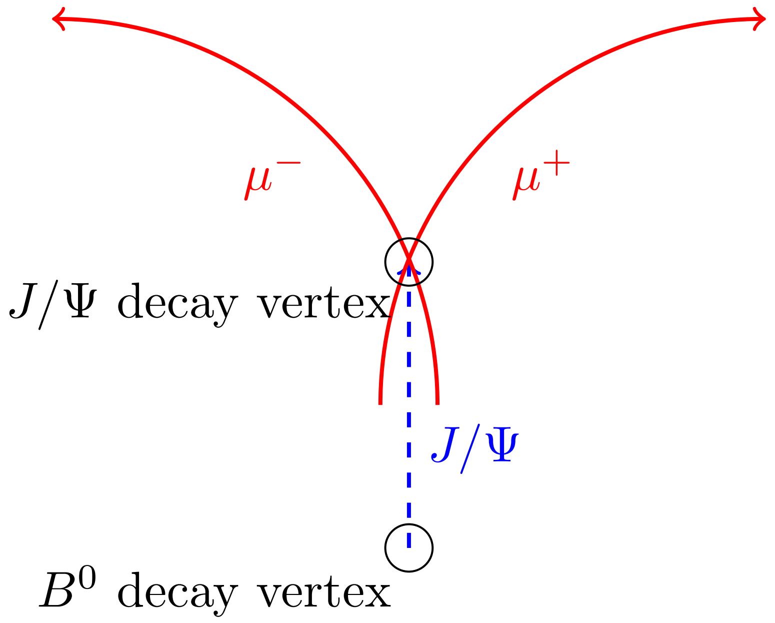
- Material for the tutorial is located in `/group/belle2/users/tenchini`
  - MC samples to run on are in `mdst/`
  - You can use `steering.py` as an example skeleton file (or use your own)

What is Vertexing

- Combine particle measurements under the assumption that they originate from a common point (or a set of points)
- Inputs: track helix, energy deposits, associated measurement covariances
- Outputs: vertex position, 4-momentum, covariance matrix

Why?

- Vertex position measurement
- Mass measurement
- Lifetime measurement
- (Combinatorial) Background rejection
- etc.



J.-F. Krohn

Fitting theory

For a number of parameters we are interested in, we have to find a minimal representation.

Find the best parameters describing the measurement

$$\vec{x}_{params} = \begin{pmatrix} \dots \\ x_i \\ y_i \\ z_i \\ px_i \\ py_i \\ pz_i \\ E_i \\ \dots \end{pmatrix} \quad \mathbb{R}^n \leftrightarrow \mathbb{R}^m \quad \vec{y} = \begin{pmatrix} \dots \\ d_0 \\ z_0 \\ \phi_0 \\ \omega \\ \tan\lambda \\ \dots \end{pmatrix} \qquad \vec{r} := \vec{m} - \vec{h} = 0 = \begin{pmatrix} d_{0\ measured} - d_{0\ hypothesis} \\ \phi_{0\ measured} - \phi_{0\ hypothesis} \\ \omega_{measured} - \omega_{hypothesis} \\ z_{0\ measured} - z_{0\ hypothesis} \\ \tan\lambda_{measured} - \tan\lambda_{hypothesis} \\ \dots \\ p_{x\ mother} - p_{x\ daughter1} - p_{x\ daughter2} \\ p_{y\ mother} - p_{y\ daughter1} - p_{y\ daughter2} \\ p_{z\ mother} - p_{z\ daughter1} - p_{z\ daughter2} \\ E_{mother} - E_{daughter1} - E_{daughter2} \\ \dots \\ m^2_{\pi^0} - E^2_{\pi^0} + |\vec{p}_{\pi^0}|^2 \\ \dots \end{pmatrix}$$

There is no exact analytical solution. So we define chi squared as the distance to the ideal solution.

$$\chi^2_{global} = (\vec{m} - \vec{h})\ \underset{\uparrow}{V^{-1}}(\vec{m} - \vec{h})^T$$

measurement covariance

In other words:

$$\boxed{\frac{\partial \chi^2_{global}}{\partial \vec{x}_{params}} = 0}$$

Least Square Estimator

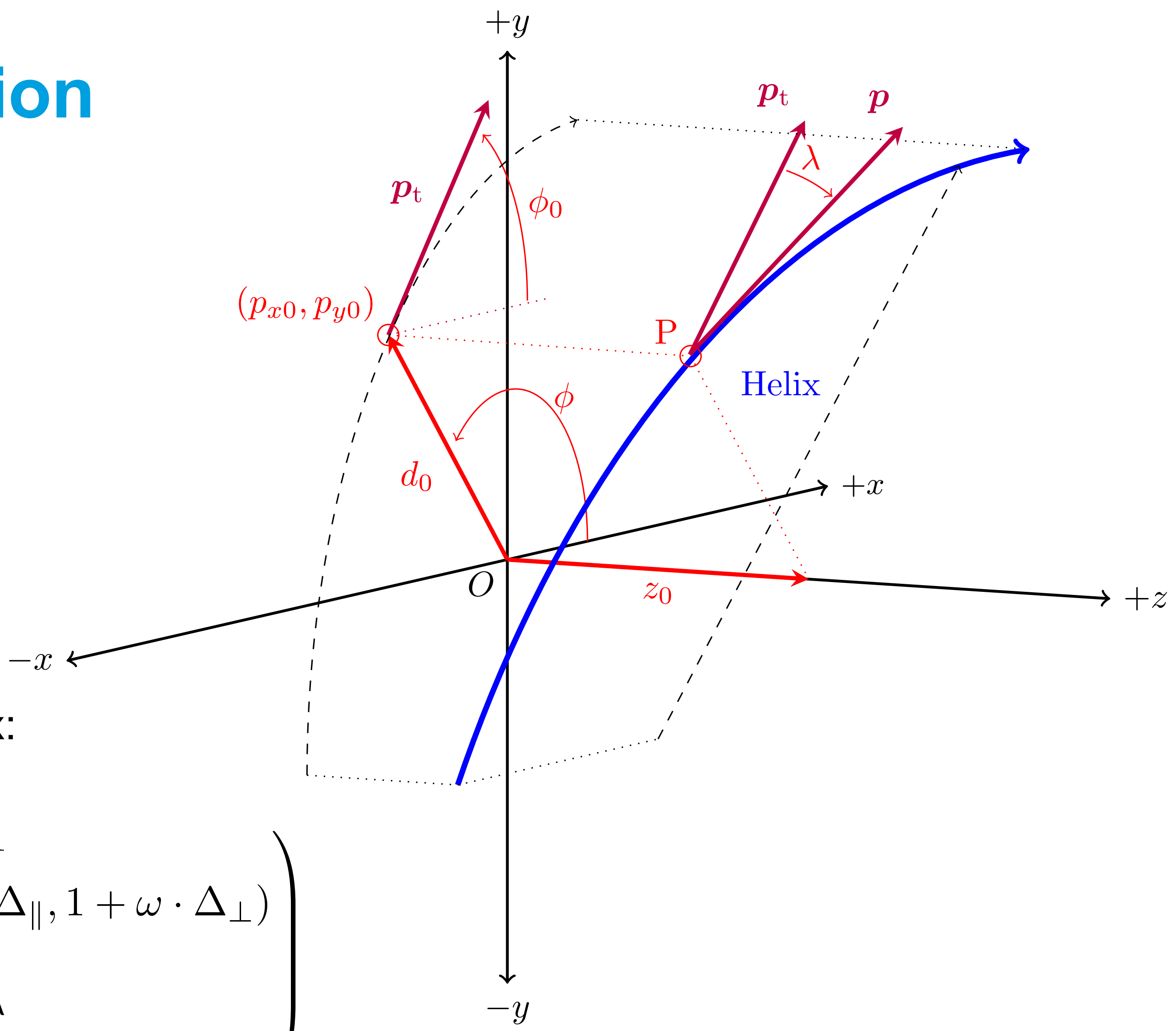
Example: Track Parametrisation

- Would be really easy (and linear) for a free body:

$$\boldsymbol{h}_{track}(\boldsymbol{x}) = \begin{pmatrix} x_0 \\ y_0 \\ t_x \\ t_y \\ p \end{pmatrix} = \begin{pmatrix} x - zp_x/p_z \\ y - zp_y/p_z \\ p_x/p_z \\ p_y/p_z \\ p \end{pmatrix}$$

- More complicated with a magnetic field - 5D helix:

$$\boldsymbol{h}_{\text{track}}(\boldsymbol{x}) = \begin{pmatrix} d_0 \\ \phi_0 \\ \omega \\ z_0 \\ \tan\lambda \end{pmatrix} = \begin{pmatrix} A(1+U)^{-1} \\ \operatorname{atan2}(p_y, p_x) - \operatorname{atan2}(\omega \cdot \Delta_{\parallel}, 1 + \omega \cdot \Delta_{\perp}) \\ a \cdot q/p_t \\ z + l \cdot \tan\lambda \\ p_z/p_t \end{pmatrix}$$



Linear Estimator

- In many cases the measurement **h** is "almost" linear - dh/dx varies slowly with respect to x.

- We can linearise it as $h(x) = h_0 + Hx$ with $H = \dfrac{dh}{dx} = constant$

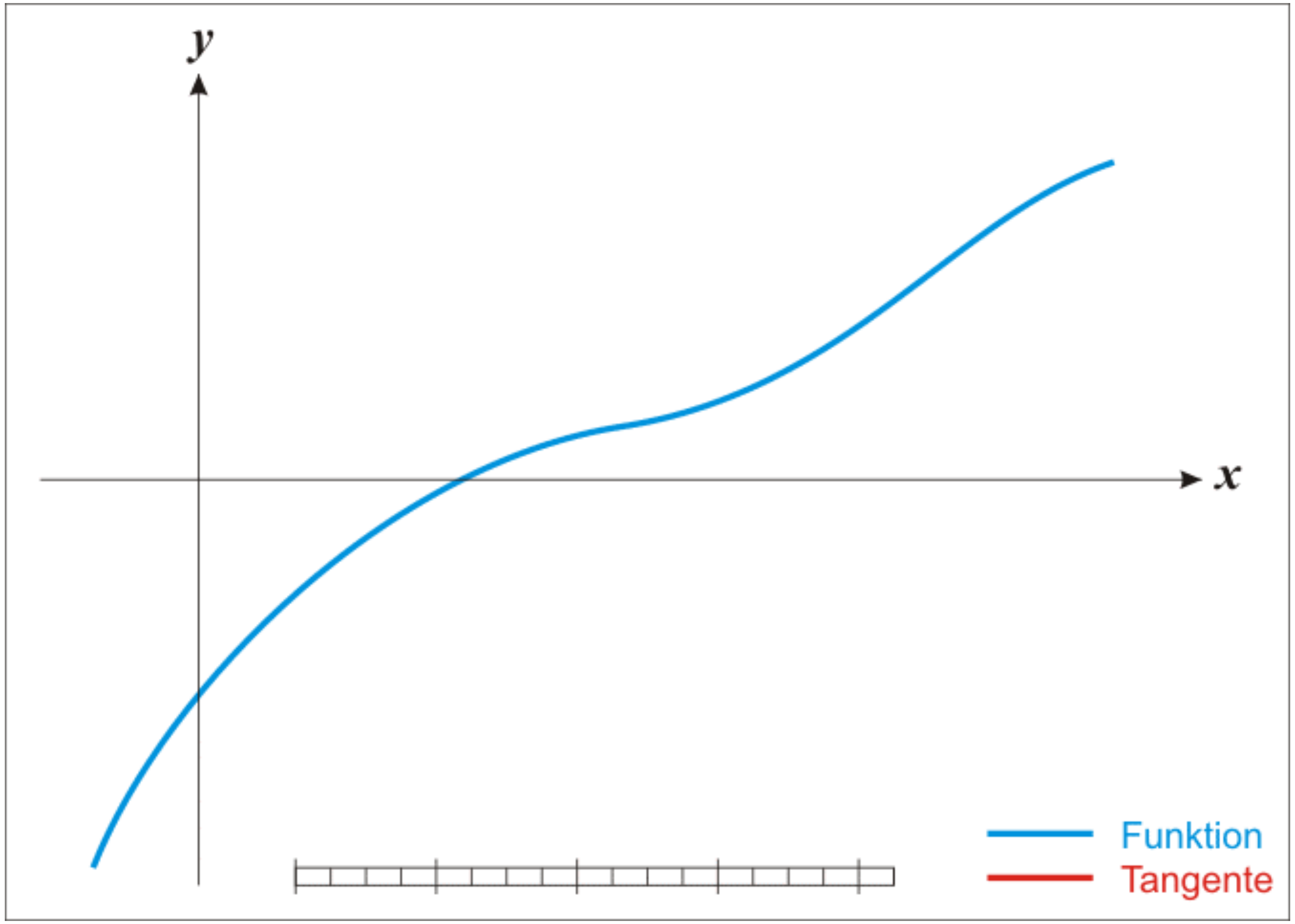
- Then the vanishing condition becomes $\dfrac{d\chi^2}{dx} = -2H^T V^{-1}(m - h_0 - Hx) = 0$

- And the solution is analytical: $\hat{x} = \underbrace{\boxed{(H^T V^{-1} H)^{-1}}}_{var(\hat{x})} H^T V^{-1}(m - h_0)$

- But what if it's non linear? Then we have $\dfrac{d\chi^2}{dx} = 2\,\dfrac{dh(x)}{dx}^T V^{-1}(h(x) - m) = 0$

Newton-Raphson Method

GIF taken from Wikipedia... this won't work in the PDF version



Newton-Raphson Method

- Iteratively solve $x_{n+1} = x_n - f(x_n)\left(\frac{df(x)}{dx}\right)^{-1}$ where $x_n = x_0 - \frac{d\chi^2}{dx}\left(\frac{d^2\chi^2}{dx^2}\right)^{-1}$
- The second derivative is

$$\frac{d^2\chi^2}{dx^2} = \underbrace{2\,\frac{dh(x)}{dx}^T V^{-1}\frac{dh(x)}{dx}}_{\text{also in the linear case}} + \underbrace{2\,\frac{d^2h(x)}{dx^2}^T V^{-1}(h(x)-m)}_{\text{new term}}$$

- Where the second term is nearly always dropped because:
    - It could make the second derivative negative, depending on the starting point.
    - If h(x) is only weakly nonlinear, this term is much smaller than the first.
- This is the same as saying that we linearise the model.

To summarise

$$\frac{1}{2}\frac{d\chi^2}{dx}\bigg|_{x_0} = -H^T V^{-1}(m - h(x_0)) \qquad \frac{1}{2}\frac{d^2\chi^2}{dx^2}\bigg|_{x_0} = H^T V^{-1} H$$

$$\hat{x} = x_0 - \frac{d\chi^2}{dx}\left(\frac{d^2\chi^2}{dx^2}\right)^{-1} \qquad H = \frac{dh(x)}{dx}\bigg|_{x_0}$$

- This is not linear, it's *linearised*: you need a good starting point and a few iterations.
- However, note the matrix inversion. This is a GIANT matrix: NxN where N is the total amount of fit parameters.
  - Execution time: ~O($n^{2.37}$)
  - Is this a problem? Maybe!
- If the determinant is zero, the solution is not unique: extra degrees of freedom.
  - Factor them out if possible, or add more equations.

J.-F. Krohn

Kalman Filter

- Cut the chi squared into smaller ones and solve them one by one, such that the matrix to invert becomes smaller

$$\chi^2_{J/\Psi} = \chi^2(\chi^2_{track}(\chi^2_{track})) = \chi^2(\chi^2_{track}, \chi^2_{track})$$

- So:

$$\chi^2 = (\boldsymbol{m}_1 - \boldsymbol{h}_1(\boldsymbol{x}_0))^T \boldsymbol{V}_1^{-1} (\boldsymbol{m}_1 - \boldsymbol{h}_1(\boldsymbol{x}_0)) + \ \dots + $$
$$(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{x}_{k-1}))^T \boldsymbol{V}_k^{-1} (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{x}_{k-1}))$$

- For k constraints. Each particle represents constraint!

$$\frac{\partial \chi^2_{global}}{\partial \vec{x}_{params}} = 0$$

Minimising this yields for the kth constraint - no proof, ("the reader can easily convince herself/himself that…")

$$\boldsymbol{C}_{k-1}^{-1}(\boldsymbol{x} - \boldsymbol{x}_{k-1}) + \boldsymbol{G}_k^T \boldsymbol{V}_k^{-1} (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{x}_{k-1})) = 0$$

Kalman Filter

G is the derivative of the constraint hypothesis with respect to all params

1 $$G_k := \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{x}} = -\frac{\partial \boldsymbol{r}_k}{\partial \boldsymbol{x}}.$$

R is the covariance of the residual system

2 $$\boldsymbol{R}_k = \boldsymbol{V}_k + \boldsymbol{G}_k \boldsymbol{C}_{k-1} \boldsymbol{G}_k^T$$

V: covariance of measurement

The nice thing here is that R is of the dimension of the constraint. So for a track this is 5 (helix).

3 $$\boldsymbol{K}_k = \boldsymbol{C}_{k-1} \boldsymbol{G}_k^T \boldsymbol{R}_k^{-1}$$

k times a small matrix inversion so ~O(k*5³) as the largest constraint dimension is a track with 5.

4 $$\boxed{\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{K}_k \boldsymbol{r}_k}$$

k: contraints
View each particle is a constraint.
A track is a helix,
a composite is a kinematic constraint,
etc…

C is the covariance of the entire system:

5 $$\begin{aligned} \boldsymbol{C}_k &= (1 - \boldsymbol{K}_k \boldsymbol{G}_k) \boldsymbol{C}_{k-1} (1 - \boldsymbol{K}_k \boldsymbol{G}_k)^T \\ &= \boldsymbol{C}_{k-1} - \boldsymbol{C}_{k-1} \boldsymbol{G}_k \boldsymbol{R}_k^{-1} \boldsymbol{G}_k \end{aligned}$$

6 $$\chi_k^2 = \boldsymbol{r}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{r}_k$$

7 $$\boxed{\chi^2 = \sum_{i=0}^{K} \chi_k^2}$$

You then loop through this until it converges or diverges…

Exact Constraints

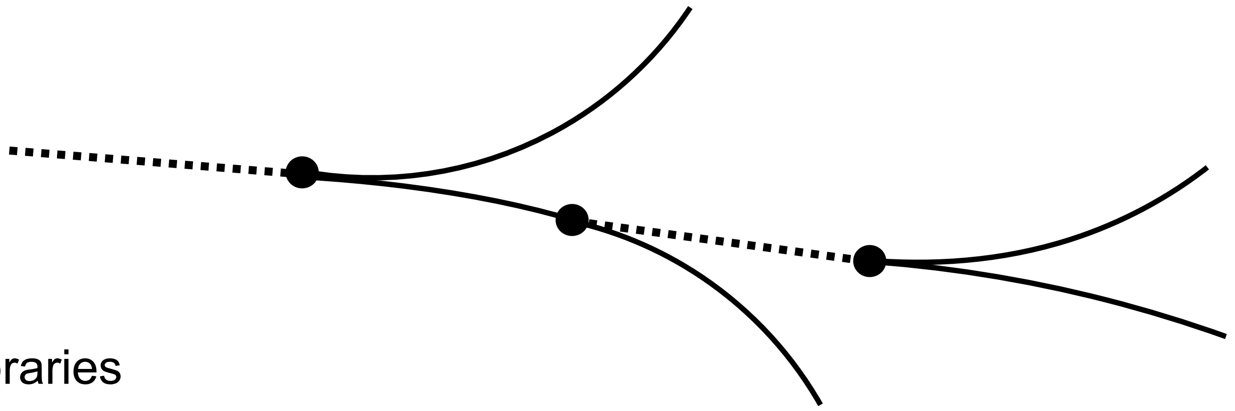
- So far, we've been handling measurement terms, contributing like $\Delta\chi^2_{measurement} = g_k(x)^T V_k^{-1} g_k(x)$
-
- What if I want an expression to be exact? Example: mass conservation.
- Then I define a term such as: $\Delta\chi^2 = \lambda_j g_j(x)$ (Lagrange multiplier) were g(x) is the expression.
- And the term contributes to the LSE as $0 = \dfrac{d\chi^2}{d\lambda_j} = g_j(x)$
- In the mass case, we have:

$$\chi^2_+ = \lambda \left[\sum (p_i^2 + m_i^2) - \left(\sum \vec{p}_i\right)^2 - m^2_{\text{pdg}} \right]$$

still valid even in Kalman applications...

Vertex Fitting Tools at Belle II

- KFit:
  - Basic kinematic fitter
  - Inherited from Belle
- RAVE:
  - Progressive single vertex fit
  - External package from CMS vertexing libraries
  - Deprecated for analysis use
- TreeFitter:
  - Progressive fit of the decay chain



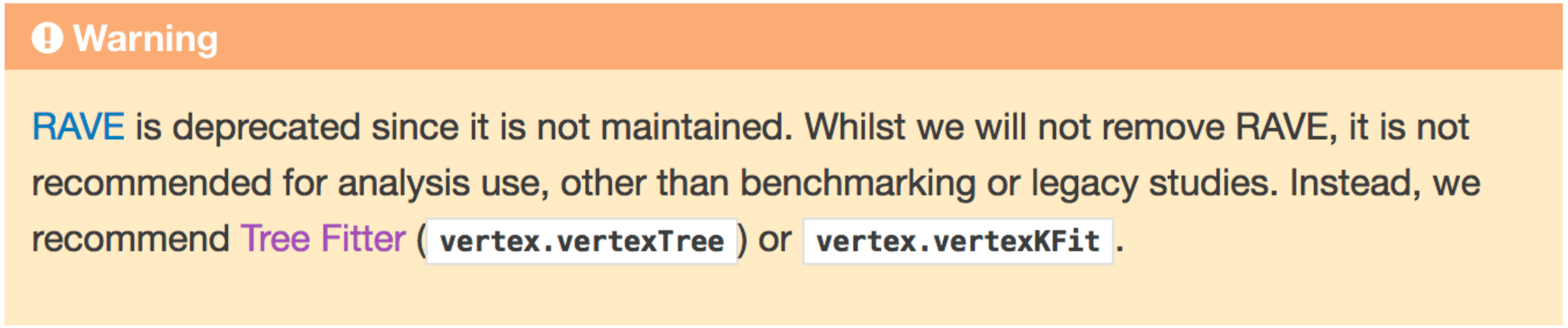
KFit

- Single vertex fit inherited from Belle.
- Non-iterative: pure matrix inversion.
- Called through various wrapper functions:

```
vertex.vertexKFit(list_name, conf_level, decay_string='', constraint='', path=None)
vertex.vertexKFitDaughtersUpdate(list_name, conf_level, constraint='', path=None)
vertex.massKFit(list_name, conf_level, decay_string='', path=None)
vertex.massKFitDaughtersUpdate(list_name, conf_level, decay_string='', path=None)
vertex.massVertexKFit(list_name, conf_level, decay_string='', path=None)
vertex.massVertexKFitDaughtersUpdate(list_name, conf_level, decay_string='', path=None)
```

- Can perform mass constrained fits
- Can perform IP constrained fits (more on this later).
- Can vertex fit one π^0 (again, more on this later).
- Need to fit a single vertex? KFit is (usually) fine.

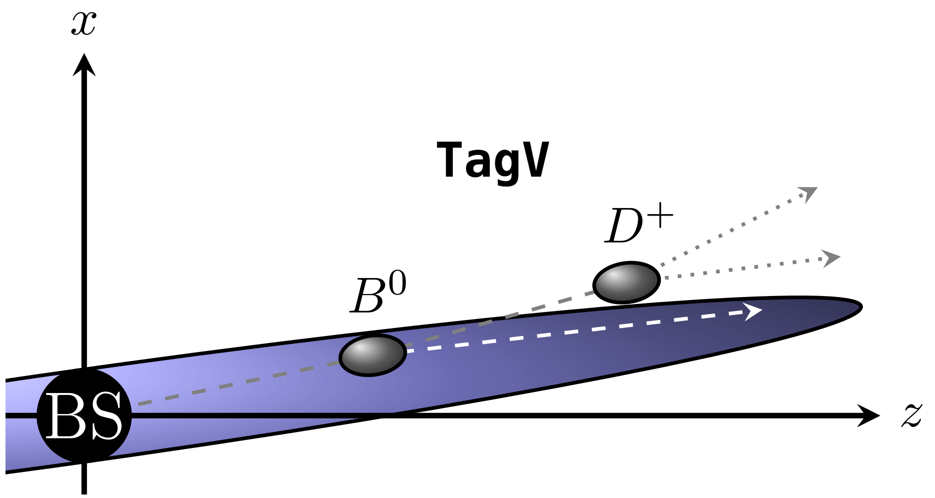
RAVE



Warning

RAVE is deprecated since it is not maintained. Whilst we will not remove RAVE, it is not recommended for analysis use, other than benchmarking or legacy studies. Instead, we recommend Tree Fitter (`vertex.vertexTree`) or `vertex.vertexKFit`.

- External package from CMS vertexing libraries
- Progressive single vertex fit (Kalman)
  - Actually *slower* than KFit, due to API overhead.
- Still required for some key applications:
  - V0Finder through Genfit integration in tracking.
  - TagV using adaptive vertex fitting.
    - If you want to do TDCVP you probably need this...
      - ... but I won't cover it here.



TreeFitter

- Kalman based filter for a whole particle decay chain ("tree").
- Automatically builds the tree structure based on provided logic (hypothesis-based).
  - Can even be a sum of different decay channels - the fitter knows.

TreeFitter

- Kalman based filter for a whole particle decay chain ("tree").
- Automatically builds the tree structure based on provided logic (hypothesis-based).
  - Can even be a sum of different decay channels - the fitter knows.

TreeFitter

- Kalman based filter for a whole particle decay chain ("tree").
- Automatically builds the tree structure based on provided logic (hypothesis-based).
    - Can even be a sum of different decay channels - the fitter knows.

```
import basf2 as b2
import modularAnalysis as ma

main = b2.create_path()

ma.fillParticleList('K-:all', 'some cut', path=main)
ma.fillParticleList('pi-:all', 'some cut', path=main)
ma.reconstructDecay('D0:kpi -> K-:all pi+:all', 'some cut', path=main)
ma.reconstructDecay('D*+:D0pi -> D0:kpi pi+:all', 'some cut', path=main)

ma.vertexTree(
    list_name='D*+:D0pi',
    conf_level=-1,
    massConstraint=['D0'],
    ipConstraint=True,
    updateAllDaughters=True,
    path=main,
)
```

your decay reconstruction

minimum confidence level, -1 = not converged

takes names or pdg codes

beam constraint

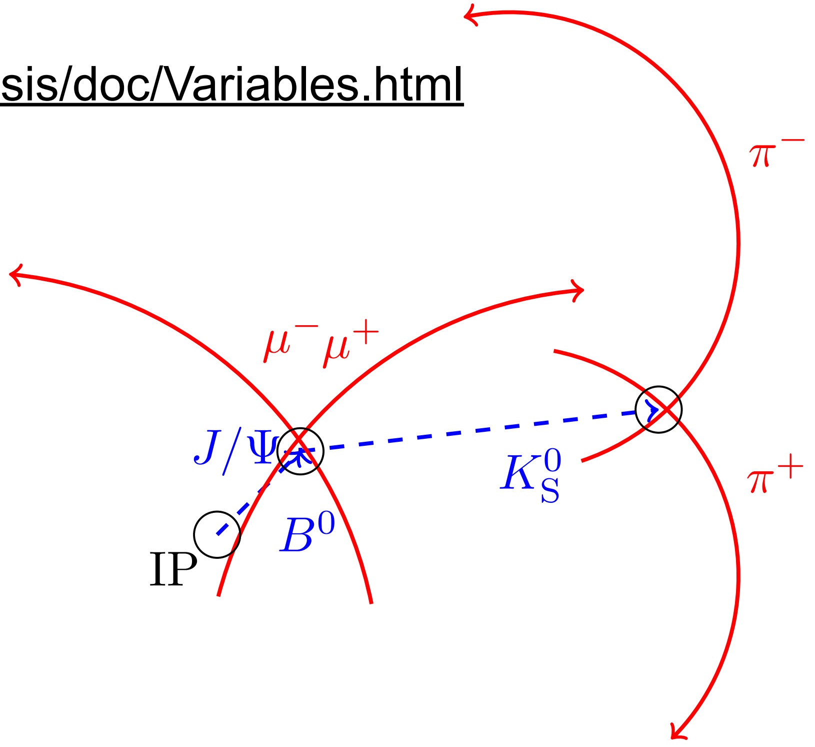
update daughter kinematics

Exercise 1: Basic Vertexing

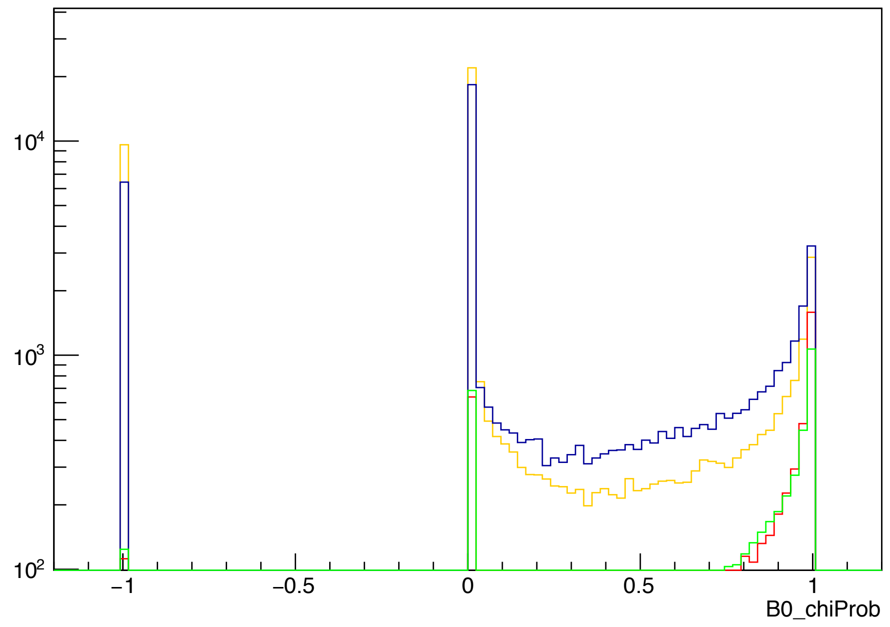
- Run the TreeFitter, and show that we can actually resolve the vertices and mass peaks.
- To speed this up, all lists in the example steering file are MC matched.
  - If you are concerned about particle mis-identification, the vertex fit may help!
- Broad cuts around masses are recommended.
- You can find variables names to plot on Sphinx:
  - https://b2-master.belle2.org/software/development/sphinx/analysis/doc/Variables.html

- The p-value of the fit is saved as `chiProb`.

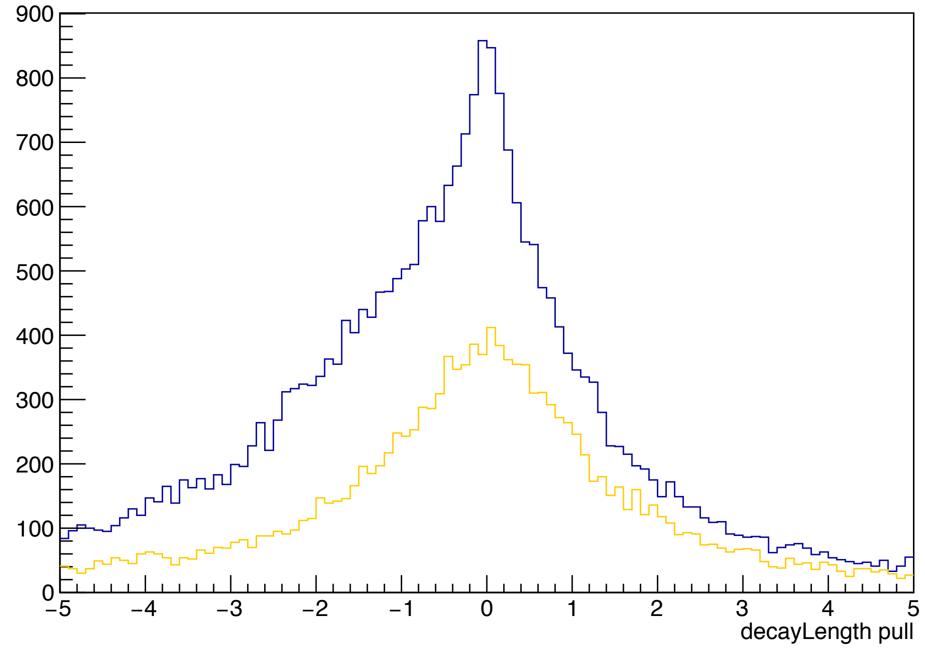
- Do we need a mass constraint on the Ks?

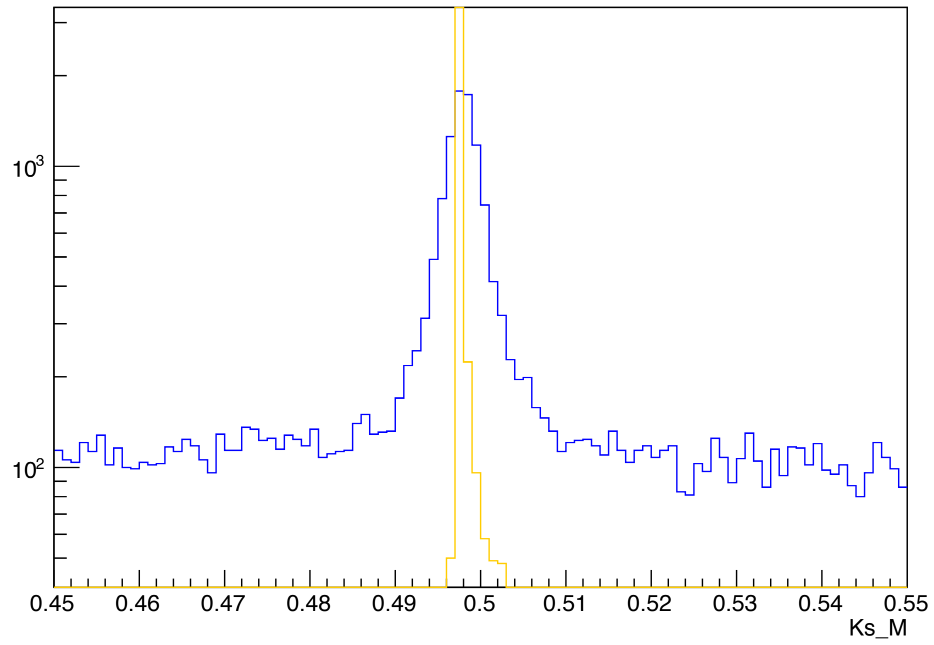


Some plots from Exercise 1...



No mass fit
No mass fit, true
Mass fit
Mass fit, true





Mass Constraint

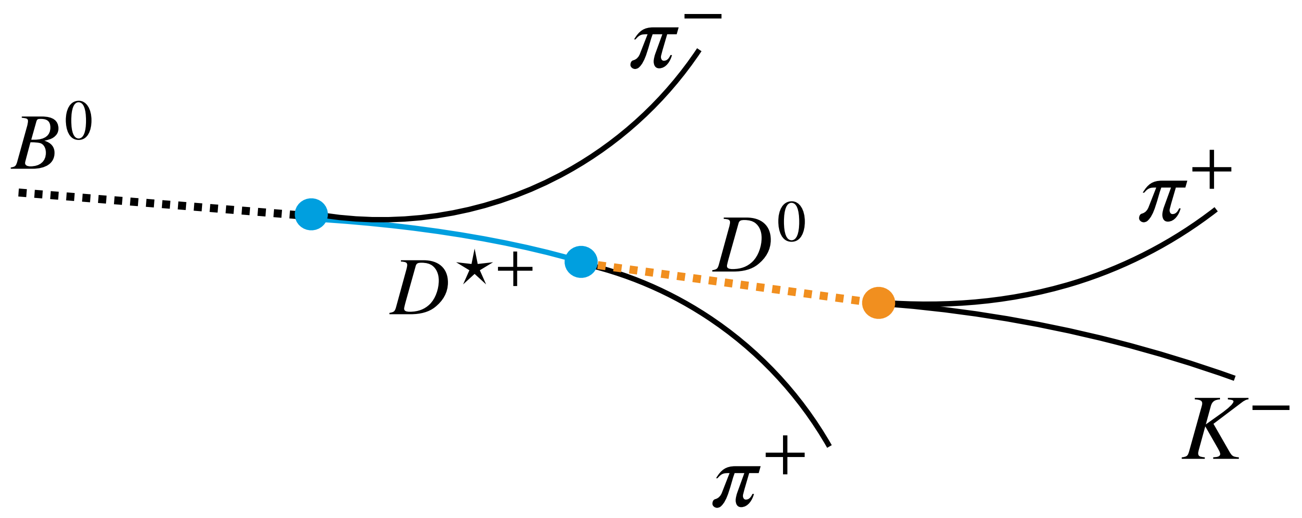
- We already partially covered this before: $r^{\alpha}(\boldsymbol{x}) = m^2_{\mathrm{PDG}} - E^2 + |\boldsymbol{p}|^2 - \boldsymbol{H}^{\alpha-1} \cdot (\boldsymbol{x}^{\alpha}_{k-1} - \boldsymbol{x}^{\alpha-1})$

- Some things to note:
  - Fixing the mass to the nominal value only makes sense if the particle has a narrow mass peak. If the width is measurable, then a mass constraint will cause a bias.
  - A lot of the combinatorial rejection comes from pure momentum conservation and vertex fit, but a mass constraint will still help.
  - If you fit on the mass, your mass will be a delta, you can't use it anymore...

- Remember that **garbage in is garbage out**, and may slow down the execution and/or cause fit failures.
  - Do some preselection before fitting.

Parametrising the Decay Tree

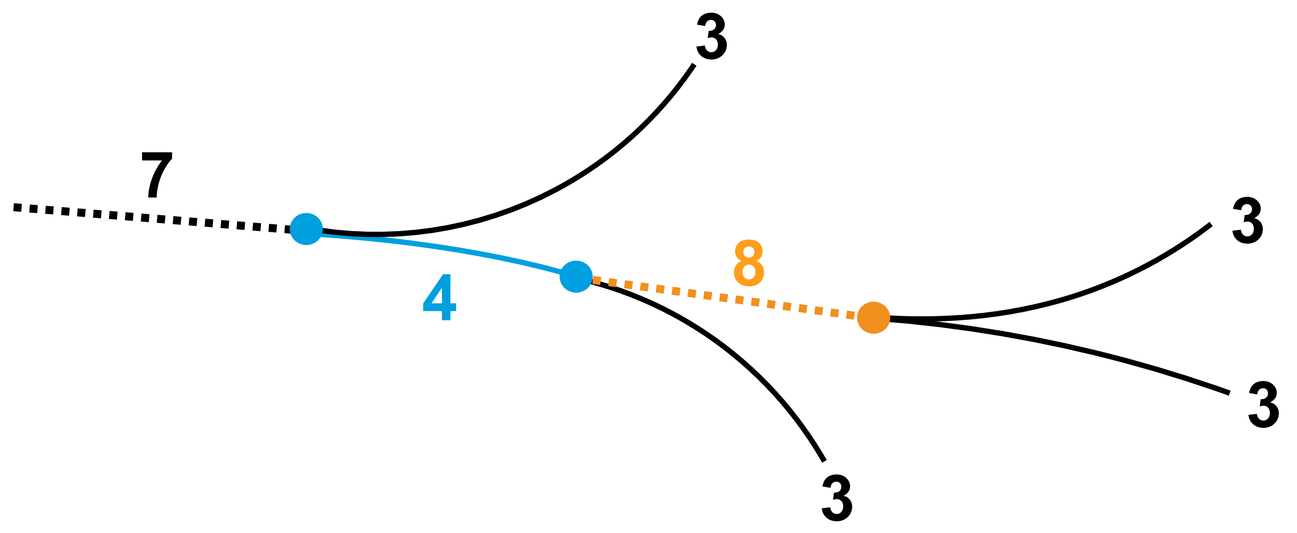
- There are several ways. This is one of them.
- For each particle, assign a 3-momentum {px,py,pz}
  - If the particle is a final state, use the nominal mass. Otherwise, assign an energy {E}
- If the particle is long lived ("composite") or the head of the decay, assign a decay vertex {x,y,z}
  - If it's not the head of the decay, assign a flight length {θ}.
- If it's short lived ("resonance") nothing more is required. The decay vertex of the mother is used.



Mini-Exercise: Can you count the parameters of this decay?

Parametrising the Decay Tree

- There are several ways. This is one of them.
- For each particle, assign a 3-momentum {px,py,pz}
  - If the particle is a final state, use the nominal mass. Otherwise, also assign an energy {E}.
- If the particle is long lived ("composite") or the head of the decay, assign a decay vertex {x,y,z}
  - If it's not the head of the decay, assign a flight length {θ}.
- If it's short lived ("resonance") add nothing beyond 4-momentum. The decay vertex of the mother is used.



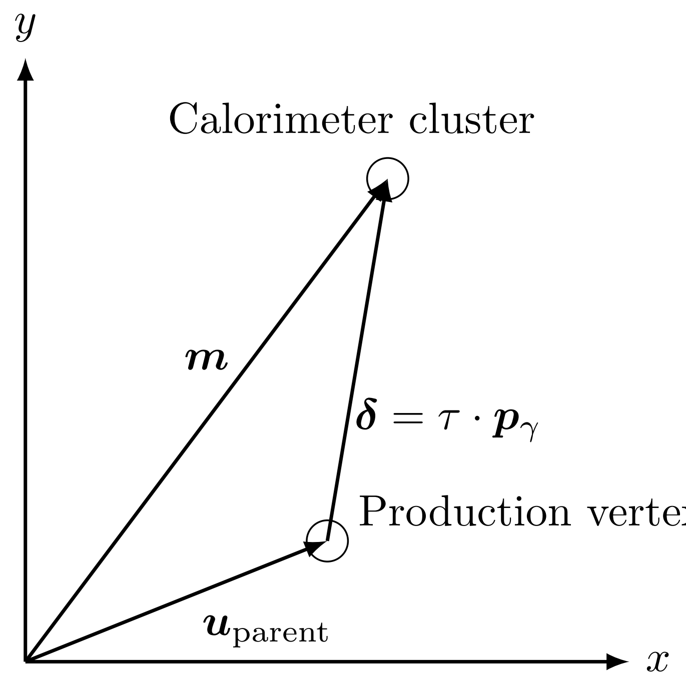
$\boxed{= 31}$

Neutral Final States

- We already discussed the charged track:

$$\boldsymbol{h}_{\text{track}}(\boldsymbol{x}) = \begin{pmatrix} d_0 \\ \phi_0 \\ \omega \\ z_0 \\ \tan\lambda \end{pmatrix} = \begin{pmatrix} A(1+U)^{-1} \\ \operatorname{atan2}(p_y, p_x) - \operatorname{atan2}(\omega \cdot \Delta_\parallel, 1 + \omega \cdot \Delta_\perp) \\ a \cdot q/p_t \\ z + l \cdot \tan\lambda \\ p_z/p_t \end{pmatrix}$$

- For neutral clusters, we have:



y
Calorimeter cluster
$\boldsymbol{m}$
$\boldsymbol{\delta} = \tau \cdot \boldsymbol{p}_\gamma$
Production vertex
$\boldsymbol{u}_{\text{parent}}$
x

$$0 = u_{parent} + \delta - m$$

$$\boldsymbol{h}_{\text{photon}}(\boldsymbol{x}) = \begin{pmatrix} u_x + \tau \cdot p_x \\ u_y + \tau \cdot p_y \\ u_z + \tau \cdot p_z \\ \sqrt{p_x^2 + p_y^2 + p_z^2} \end{pmatrix} \quad \text{and} \quad \boldsymbol{m}_{\text{photon}}(\boldsymbol{x}) = \begin{pmatrix} m_x \\ m_y \\ m_z \\ E_m \end{pmatrix}$$

$\downarrow$

Or KLong, or... neutron?

$\nearrow$ **3 equations**

$$\boldsymbol{r}'^{\alpha}_{\text{photon}}(\boldsymbol{x}) = \begin{pmatrix} (m_i - u_i) - (m_k - u_k)\frac{p_i}{p_k} \\ (m_j - u_j) - (m_k - u_k)\frac{p_j}{p_k} \\ E_m - \sqrt{p_i^2 + p_j^2 + p_k^2} \end{pmatrix} + \boldsymbol{H}^{\alpha-1} \cdot (\boldsymbol{x}^{\alpha}_{k-1} - \boldsymbol{x}^{\alpha-1})$$

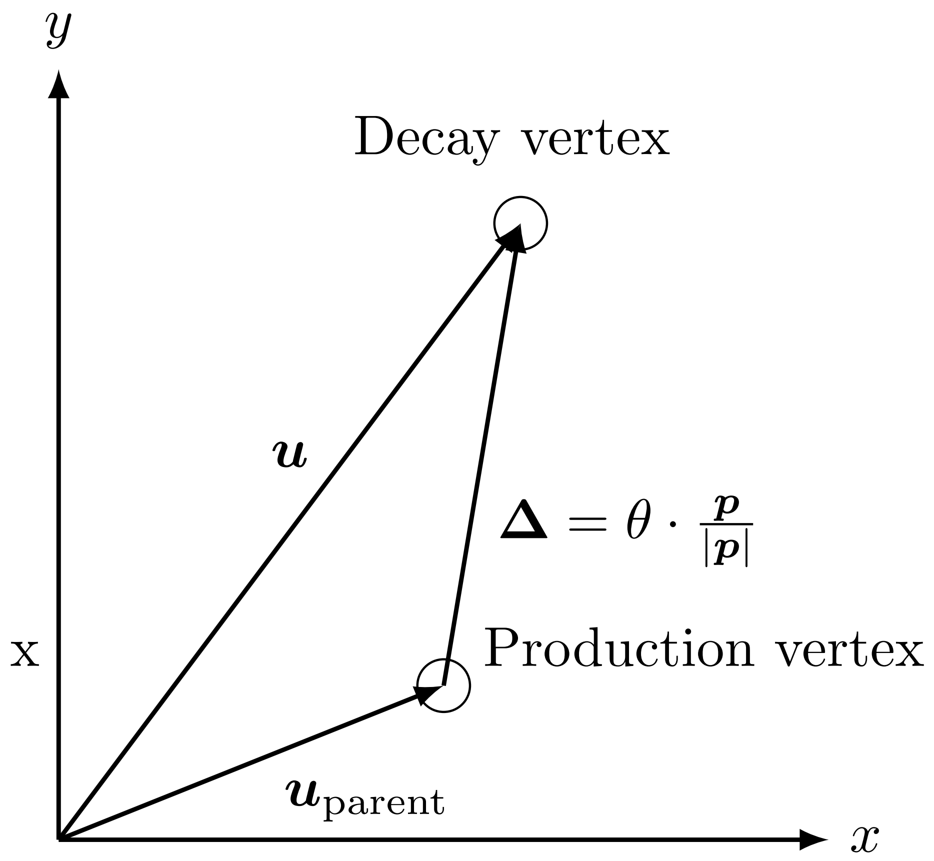
Obtained by factoring out the highest momentum component.

Flight Length: Geometric Constraint

- Not really a constraint, but rather a way the TreeFitter accounts for correlations between flight parameters.
- Applied automatically: $0 = u_{parent} + \Delta - u$
- We have a very good handle on momentum uncertainties:
  - Measure on the momentum projection

$$r(x) = u_{parent} + \theta \cdot \frac{p}{|p|} - u$$

- 1 new parameter {θ} but 3 equations: **-2 degrees of freedom!**
- After the fit is completed, the fitter fills `ExtraInfo("decayLength")` and `ExtraInfo("decayLength")`
  - ... or you can use the variables `flightDistance` and `flightDistanceErr`



y
Decay vertex
$\boldsymbol{u}$
$\boldsymbol{\Delta} = \theta \cdot \frac{\boldsymbol{p}}{|\boldsymbol{p}|}$
x
Production vertex
$\boldsymbol{u}_{\text{parent}}$
x

Flight Length: Geometric Constraint

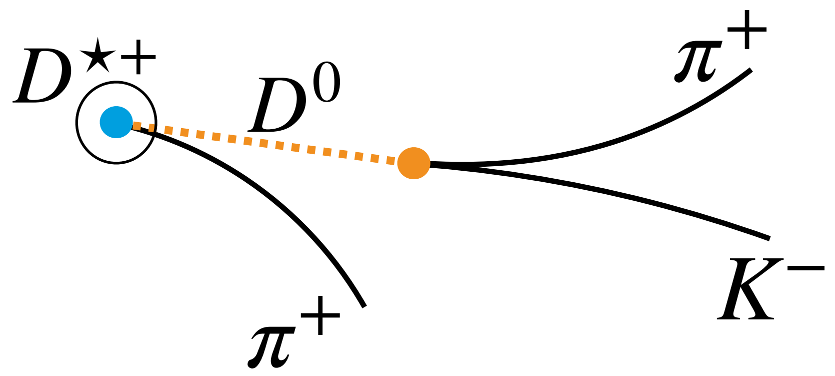
- But what about charged, long lived particles?
- Particles such as Σ+ can travel for several cm and are affected by the magnetic field.
- Unfortunately at the moment **all composites are assumed to fly straight**. (Jira ticket BII-3893)
- If you're interested in working on channels with long lived charged composites, the feature will have to be developed. Please comment on the ticket or send me an e-mail.

Beam Spot Constraint

- This means different things in different fitters.
- General concept: use the beam spot information to constrain the vertex. But how?
- KFit has 3 versions:
  - **ipconstraint -** constrain the vertex to the beam spot.
  - **iptube -** as above, but only on the x-y plane.
  - **pointing -** constrain the momentum vector to pass through the beam spot.
- TreeFitter handles it by creating a new "Origin" particle to act as the new head of the decay.
  - Reminder: resonances share the vertex with the mother.
  - If the old head was short lived, this is equivalent to **ipconstraint**.
  - If it was long lived, it's more akin to **pointing** (and removes 2 degrees of freedom because of the flight length constraint)

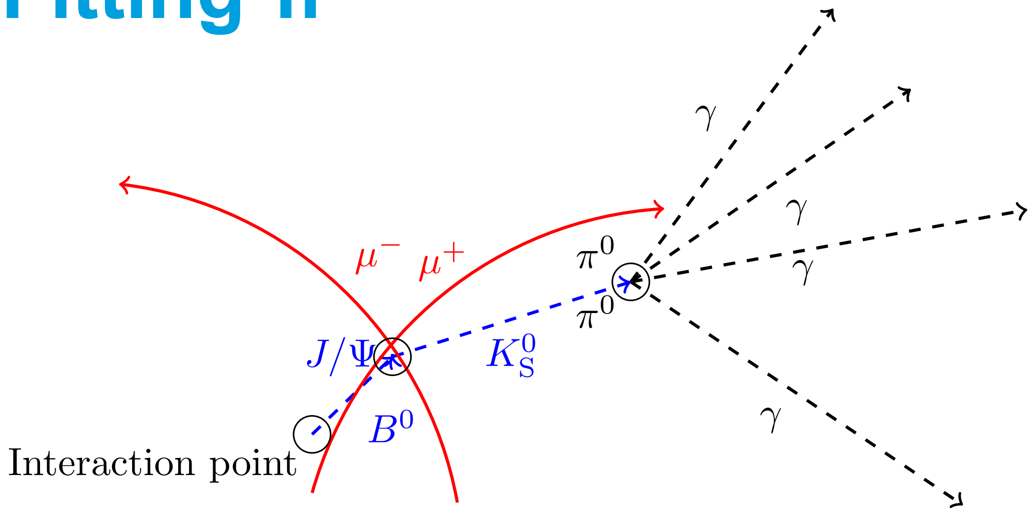
Exercise 2: Beam Constraint and Flight Length

- Consider the decay below.

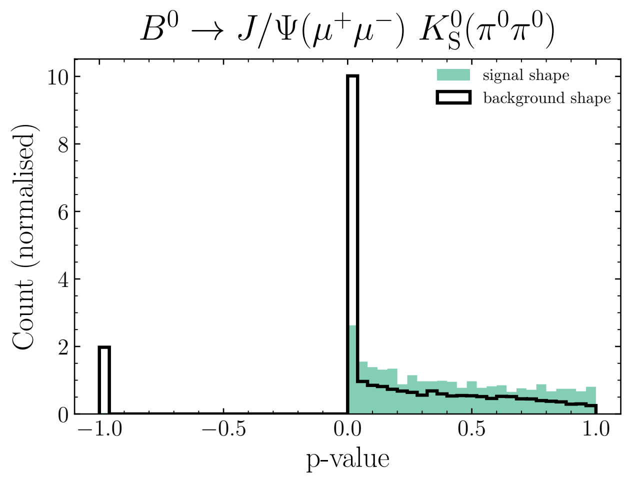


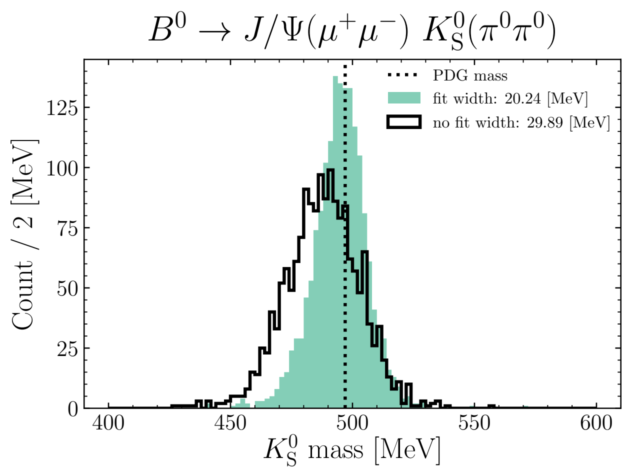
- The D* decays into a single charged track: here the beamspot plays an important role to improve resolution.
- Show what happens with and without beamspot. Extract the flight length of the D^0 and compare to MC truth.

Fitting π⁰



- Consider the decay $B^0 \to J/\psi\ K_s$, $K_s \to \pi^0\pi^0$
- I had an exercise for this, but it ran into technical issues.
  - So it's left to the reader.
- Since photons are assumed to come from (0,0,0), this will introduce a bias on the Ks mass.
- KFit can refit ONE neutral pion to the fitted vertex position, but no more, and only if tracks are present.
  - TreeFitter can handle the fit provided the π^0 is mass constrained.





Degrees of Freedom

- When in doubt, count the degrees of freedom: NDF = N(equations) - N(parameters)
- If NDF>1 you can fit, otherwise you need to add a constraint (mass, beamspot, ...)

Mini-Exercise: Can you fit these decays?

- $\pi^0(\gamma\gamma)$?
- $D^0 \to K\pi\pi^0(\gamma\gamma)$?
- $D^0 \to K_s(\pi\pi)\pi^0(\gamma\gamma)$?

	parameters	equations	net
track	{px, py, pz}	5 (helix)	2
neutral	{px, py, pz}	3	0
resonance	{E, px, py, pz}	4 (energy conservation)	0
long lived	{E, px, py, pz} +{x,y,z,θ}	4 (energy conservation) +3 (flight)	-1
head	{E, px, py, pz} +{x,y,z}	4 (energy conservation)	-3
mass		1	1
beamspot		0 or 2 (flight)	~

Degrees of Freedom

- When in doubt, count the degrees of freedom: NDF = N(equations) - N(parameters)
- If NDF>1 you can fit, otherwise you need to add a constraint (mass, beamspot, ...)

Mini-Exercise: Can you fit these decays?

- $\pi^0(\gamma\gamma)$?
  - [-3+0+0] = **-3** → **NO, not even with a mass constraint**
- $D^0 \to K\pi\pi^0(\gamma\gamma)$?
  - [-3+2*2+0+2*0] = **1** → **YES**
- $D^0 \to K_s(\pi\pi)\pi^0(\gamma\gamma)$?
  - [-3-1+2*2+0+2*0] = **0** → **YES, if mass constrained**

	parameters	equations	net
track	{px, py, pz}	5 (helix)	2
neutral	{px, py, pz}	3	0
resonance	{E, px, py, pz}	4 (energy conservation)	0
long lived	{E, px, py, pz} +{x,y,z,θ}	4 (energy conservation) +3 (flight)	-1
head	{E, px, py, pz} +{x,y,z}	4 (energy conservation)	-3
mass		1	1
beamspot		0 or 2 (flight)	~

Miscellaneous and Closing Note

- If you're using or plan to use TreeFitter, ensure you are working on **`release-04-00-01`** or later.
- There's some nasty bugs in **`release-03-02-04`** that will affect your efficiency, performance or both.
- Right now, TreeFitter is incompatible with Bremsstrahlung correction.
  - This is due to not consistently treating the corrected electron as a final state.
  - Fix is pending.

- I almost certainly forgot or omitted something.
- If you have questions, please ask.
- ... or visit questions.belle2.org .

Thank you for your attention!

Backup