VtxKalman: a decay chain fit for Babar

Wouter H.

last update: June 18th 2003

Outline:

- Part I: decay chain fit
- Part II: $\triangle z$-fit

For up-to-date slides: www.slac.stanford.edu/$\sim$hulsberg/talks/vtxkalman.pdf

Part I: A decay chain fit

- Fits a complete decay chain ’in one go’
- Input:
  - reconstructed particles (+pid hypothesis)
  - decay tree hypothesis
  - evt: mass constraints, beamspot, beamenergy, lifetimes
- Output:
  - momenta, vertex positions
  - full covariance matrix

Why ?

- 'GeoKin' fit only updates parameter of daughters, not of 'granddaughters'

$\Longrightarrow$ parameters not optimal if

  - the decay tree is more than one layer deep (*e.g.* $B \to DD$), *and*
  - the fit is strongly non-linear

- 'GeoKin' fit does not compute the correlation between daughters
  - only technical problem for 'one-layer'
  - not possible for granddaughters'

- a global decay chain fit
  - propagates constraints through complete tree
    $\Longrightarrow$ could improve resolutions in a non-linear fit
  - has all correlations
    $\Longrightarrow$ could be interesting for $\mathrm{B} \to \mathrm{DD}$ analyses

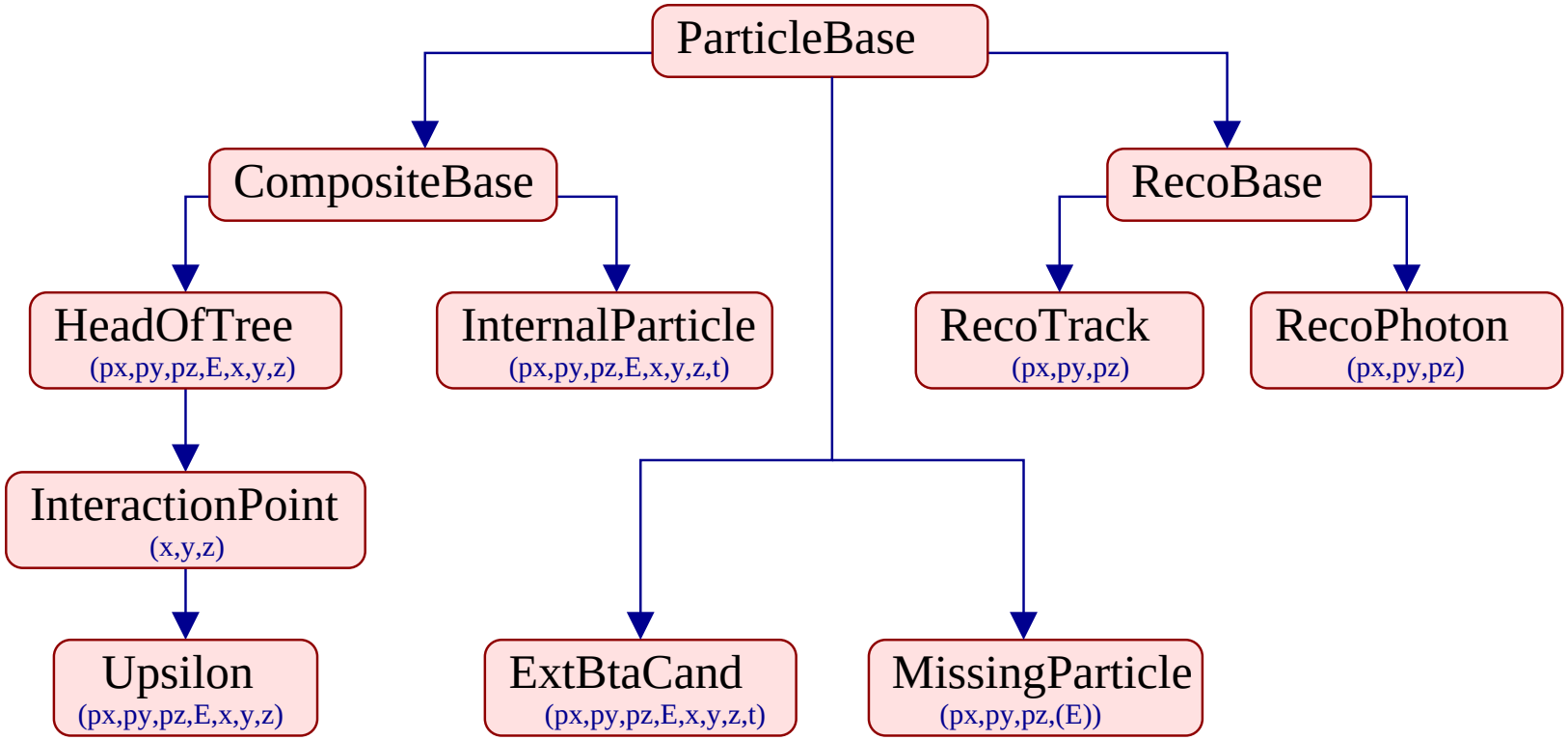
Modeling the decay chain

- decay chain broken down into 'particles'
  - 3-momentum or 4-momentum
  - *decay* vertex
  - lifetime

type	parameters	constraints
composite	$(x, y, z, \tau, p_x, p_y, p_z, E)$	geo+kin (+mass) (+lifetime)
resonance	(px, p_y, p_z, E)	kin (+mass)
photon	(p_x, p_y, p_z)	$(\vec{x}_{\rm cl}, E_{\rm cl})$
track	(p_x, p_y, p_z)	$(d_0, \phi_0, z_0, \omega, \tan(\lambda)$

- mass constraints of external particles are 'built in'
- internal mass, geometric and kinematic constraints enforced via Lagrange multipliers in the fit

Enter C++


ParticleBase
CompositeBase
RecoBase
HeadOfTree
(px,py,pz,E,x,y,z)
InternalParticle
(px,py,pz,E,x,y,z,t)
RecoTrack
(px,py,pz)
RecoPhoton
(px,py,pz)
InteractionPoint
(x,y,z)
Upsilon
(px,py,pz,E,x,y,z)
ExtBtaCand
(px,py,pz,E,x,y,z,t)
MissingParticle
(px,py,pz,(E))

Fitting

- decay chain model leads to large number of parameters with many correlations
- example: the decay tree of $\mathrm{B}^0 \to \mathrm{D}^+\mathrm{D}^-$ with $\mathrm{D} \to K\pi\pi$ has 40 parameters
- hard to solve with a global least χ^2 fit based on Newton-Rapson iteration since it requires inversion of the full covariance matrix
- therefore, use (extended) Kalman fit
  - add constraints one-by-one, updating pars and cov in each step
  - iterate non-linear constraints (*i.e.* mass)
  - iterate complete fit until failure or convergence
- Kalman filtering
  - not new for vertexing (e.g. FastVtx)
  - new for complete decay chain (I think)
  - new for exact ('Lagrange') constraints (I think)

Some differences wrt GeoKin

- GeoKin for photons:
  - take cluster position and estimated origin
  - translate into four-vector (p_x, p_y, p_z, E)
  - use four vector in vertex fit

  clearly doesn't work for $K_s \to \pi^0\pi^0$

$\Longrightarrow$ use cluster position and energy instead of 'constructed' four-vector

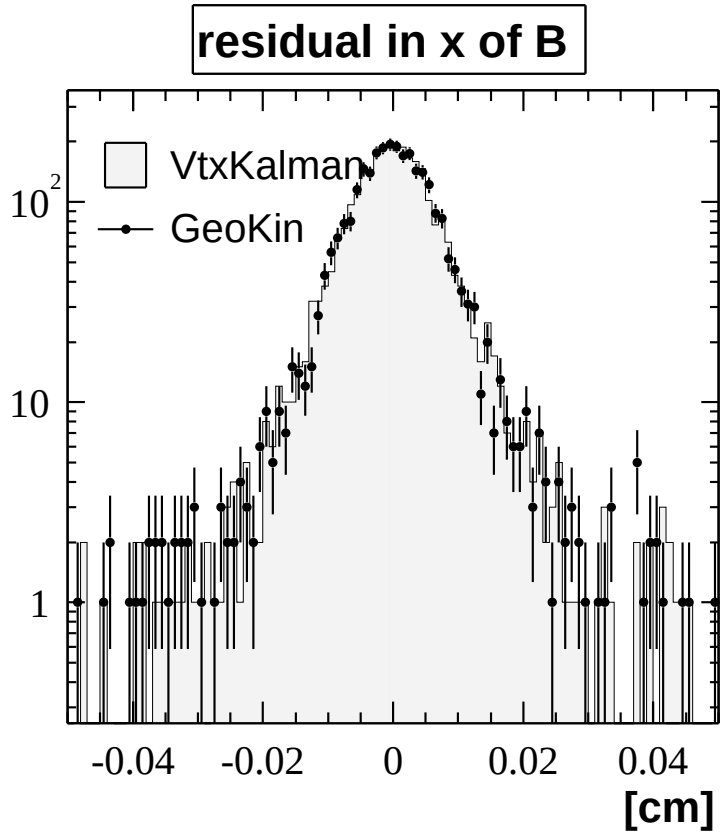
- same for charged tracks: helix parameters instead of four-vector

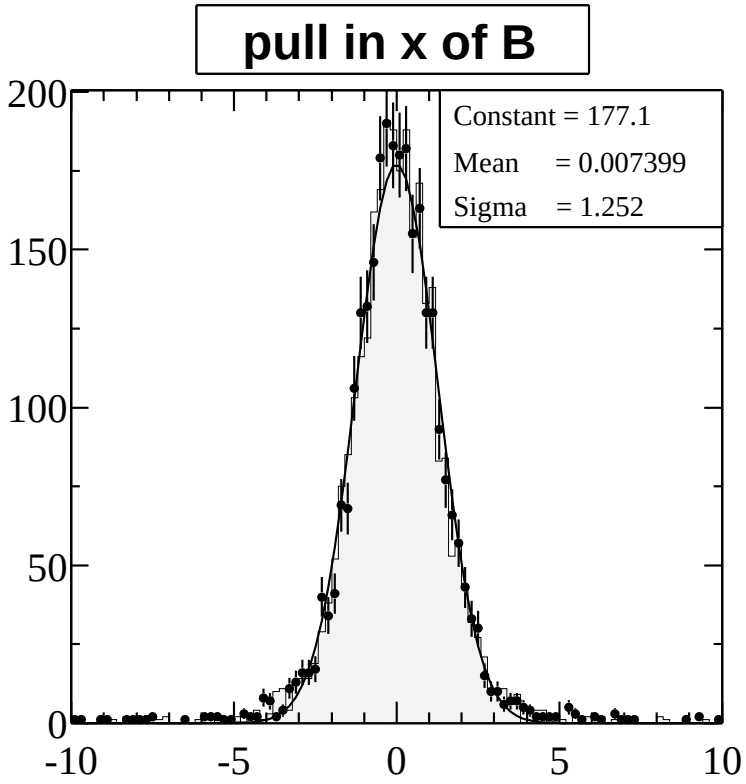
- in GeoKin the 'beamspot' is an xy-constraint on *decay* vertex
  - doesn't makes sense for B, or B decay products
    - GeoKin uses special 'beamspot-B-flight' assigns uncertainty in y of 30 μm
    - very non-gaussian

$\Longrightarrow$ define as constraint on *production* vertex ...

Comparison of position resolution

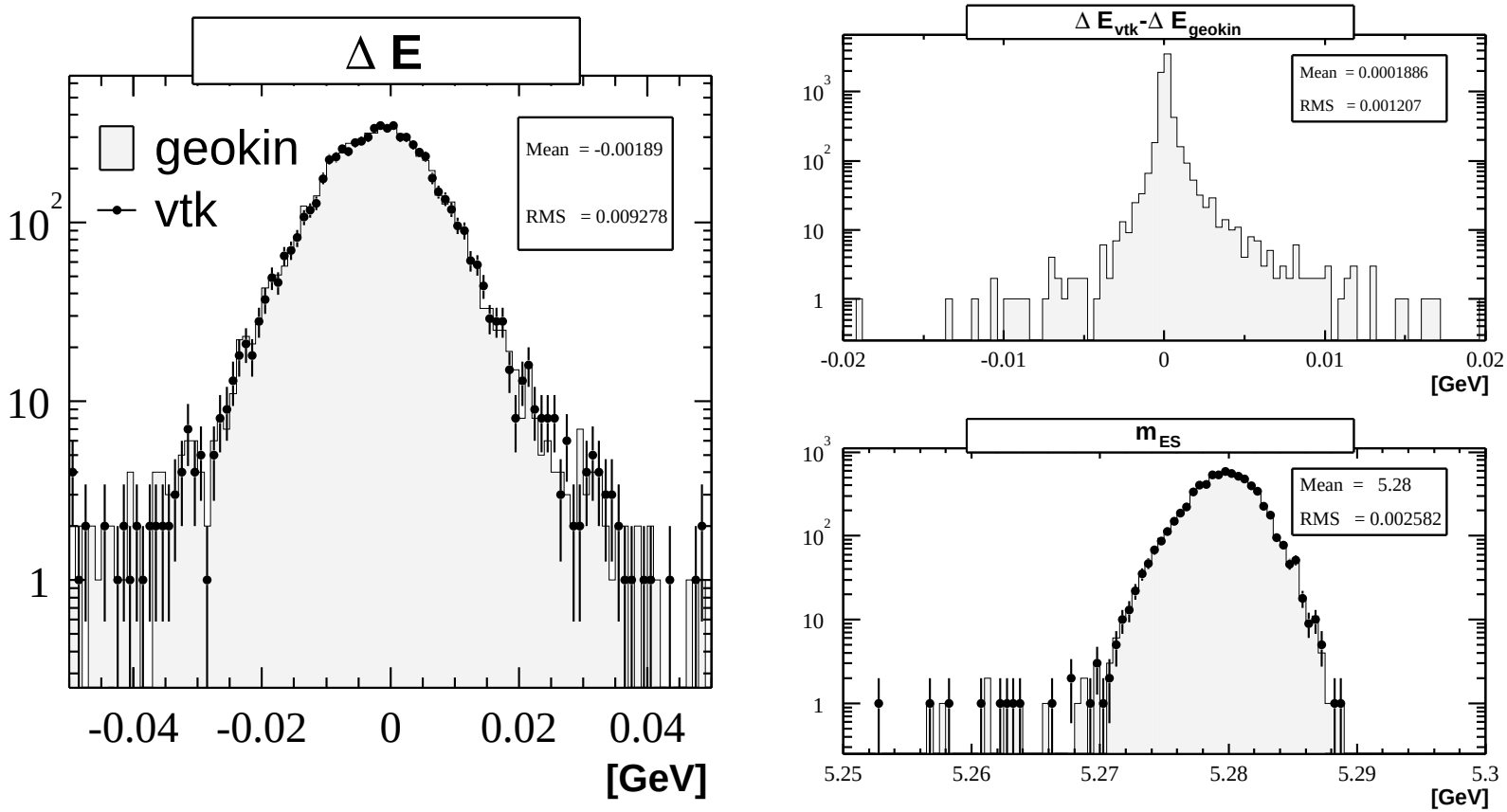
- $x - x_{mc}$ of B^0 in $B^0 \to D^+D^-$





- no difference in overall parameter resolution
- no difference in pulls

Comparison of ΔE resolution



- event-by-event difference of the order of 0.2σ
- no difference in overall parameter resolution
- no difference in pulls

Why no improvement?

1. is the problem too linear to see a difference ?

2. is the Extended Kalman Filter non suitable for propagating the non-linear constraints all through the tree?
   - EKF only inherits the parameters of the previous iteration, not its covariance
   - I am thinking of an alternative . . .

CPU time consumption

The disclaimer

- calculating correlations takes time
- CPU consumption scales roughly with n^2

  $\Longrightarrow$ expect VtxKalman to be slow for large decay trees
- time consumption predominantly in matrix manipulation
- current limitation: accessing data in HepSymMatrix, HepMatrix

CPU (2)

Measured with 'clock()', on a desktop

- a single-level decay chain $\mathrm{D}^+ \to K\pi\pi)$
  - 16 parameters
  - factor ~ 1.5 *faster* than GeoKin (0.67 ms vs 1.07 ms)

- a 2-level decay chain $\mathrm{B}^0 \to \mathrm{D}^+(\to K\pi\pi)\mathrm{D}^-(\to K\pi\pi)$
  - 41 parameters
  - 2 mass constraints
  - equally fast as GeoKin

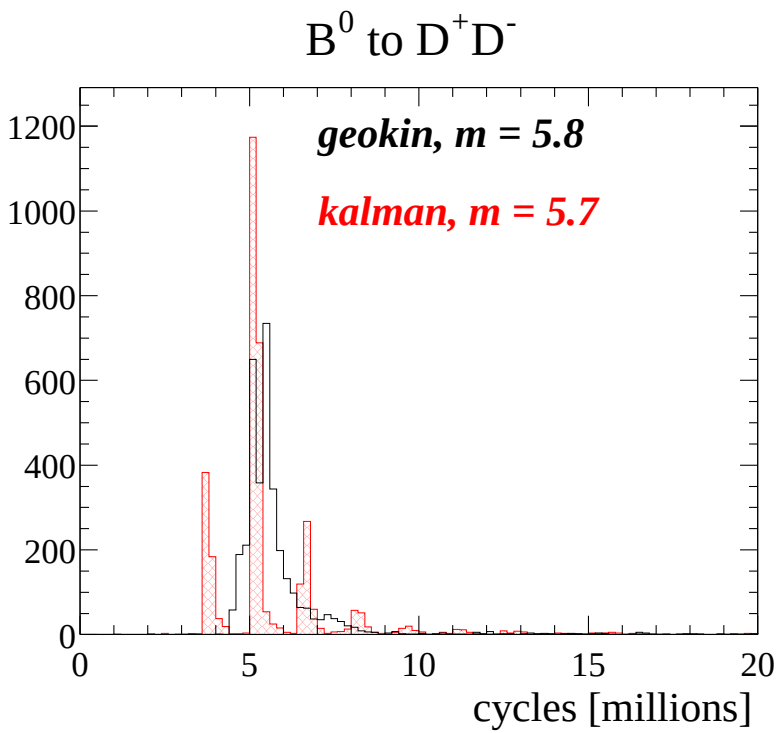
- a multi-level decay chain $\mathrm{B}^0 \to \mathrm{D}^{*0}\bar{\mathrm{D}}^{*0}$ with $\mathrm{D}^{*0} \to \pi^0(\gamma\gamma)\mathrm{D}^0(\mathrm{K}\pi)$
  - 63 parameters
  - 6 mass constraints
  - factor 2 *slower* than GeoKin

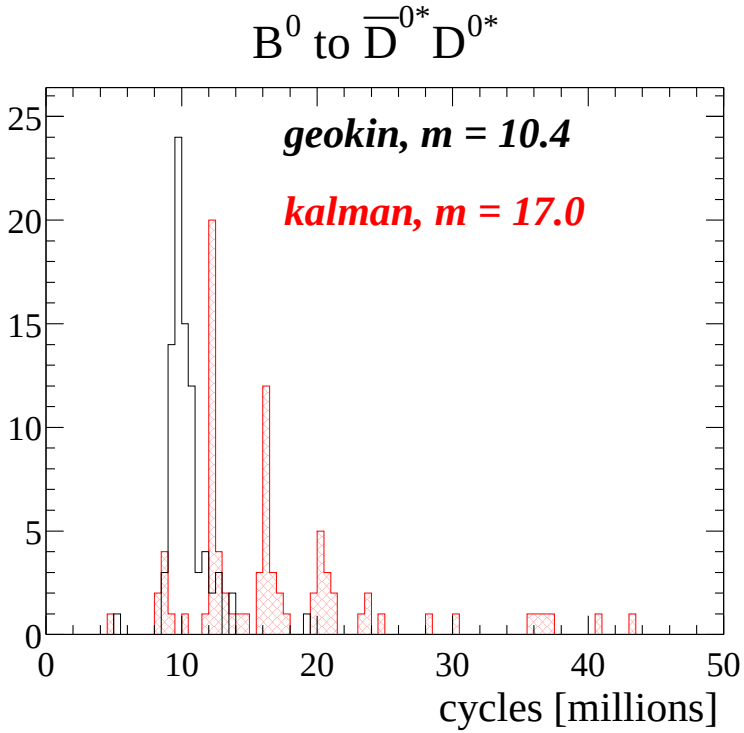
Note: VtxKalman *can* fit any tree in layers

$\Longrightarrow$ *for the same information, it is always faster*

CPU (3)

CPU time (’Gerhard’s cycles’) on MC





- VtxKalman timing proportional to number of iterations

Fitting $\mathrm{K_S} \to \pi^0\pi^0$

- GeoKin uses separate fitter for $\mathrm{K_S} \to \pi^0\pi^0$:
  - uses clusters (like VtxKalman)
  - needs direction constraint for Ks (not optimal)
    $\Longrightarrow$ due to missing implementation for production vertex constraint
  - not completely finished (and doesn't work now)

- default algorithm is 'VtxKsToPi0Pi0WalkFit'
  - uses production vertex constraint
  - *very* slow

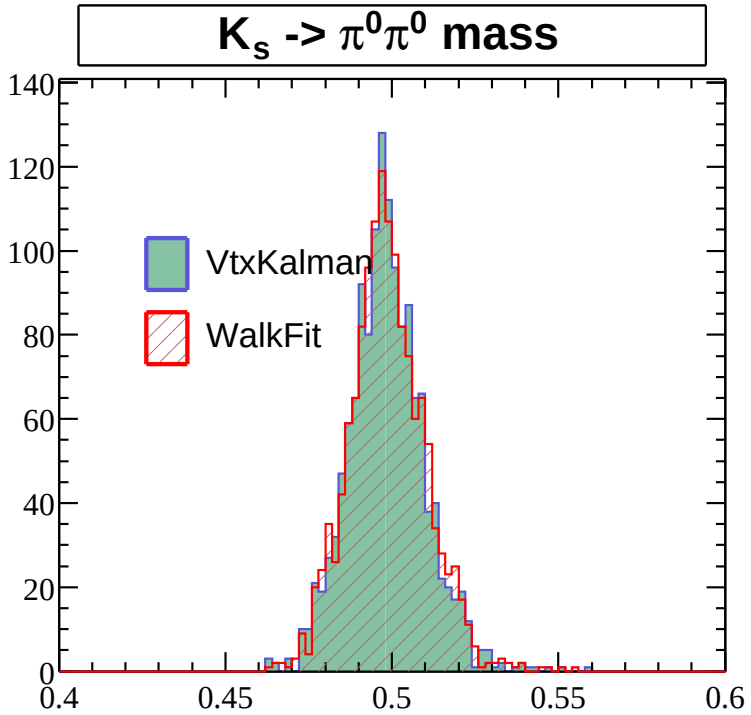
- VtxKalman can fit this decay, given
  - mass constraints on the π^0s
  - production vertex

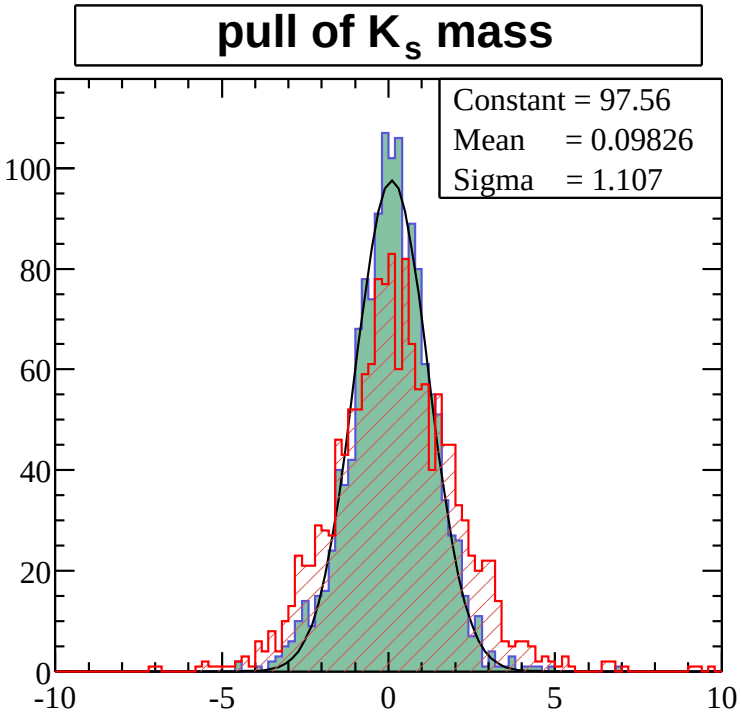
$\mathrm{K_S} \to \pi^0\pi^0$: *VtxKalman versus 'WalkFit'*

- $\mathrm{K_S} \to \pi^0\pi^0$ from $\mathrm{B}^0 \to \mathrm{J}/\psi\mathrm{K_S}$
  $\Longrightarrow$ relatively hard
- $\mathrm{K_S}$ tree build using MC truth
  $\Longrightarrow$ no composition sequence involved
- photon energy larger than 20 MeV and within 4 σ from MC truth
- use beamspot for production vertex
  $\Longrightarrow$ B decaylength irrelevant here
- use π^0 mass constraints
- fit has 2 degrees of freedom

$K_S \to \pi^0\pi^0$: *Comparing mass resolution*

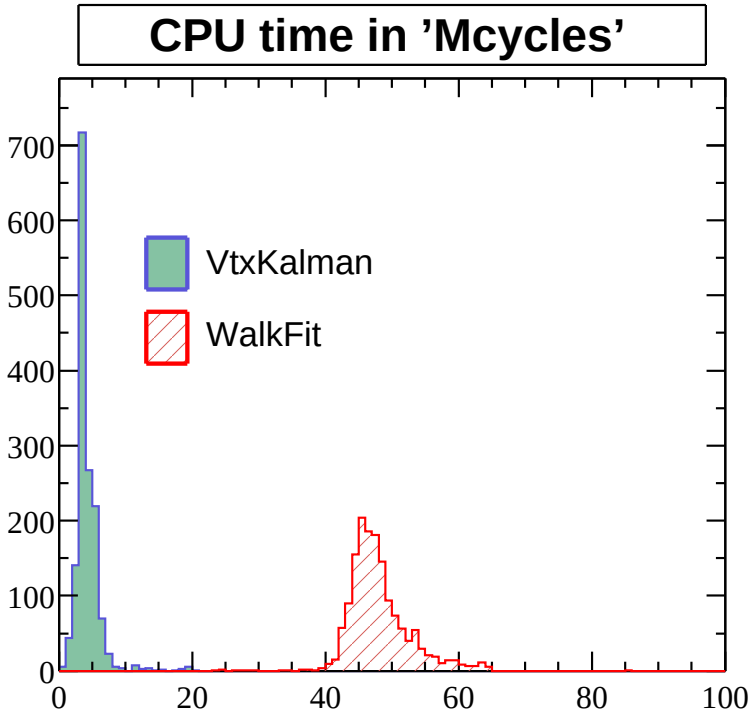
For *good* fits:

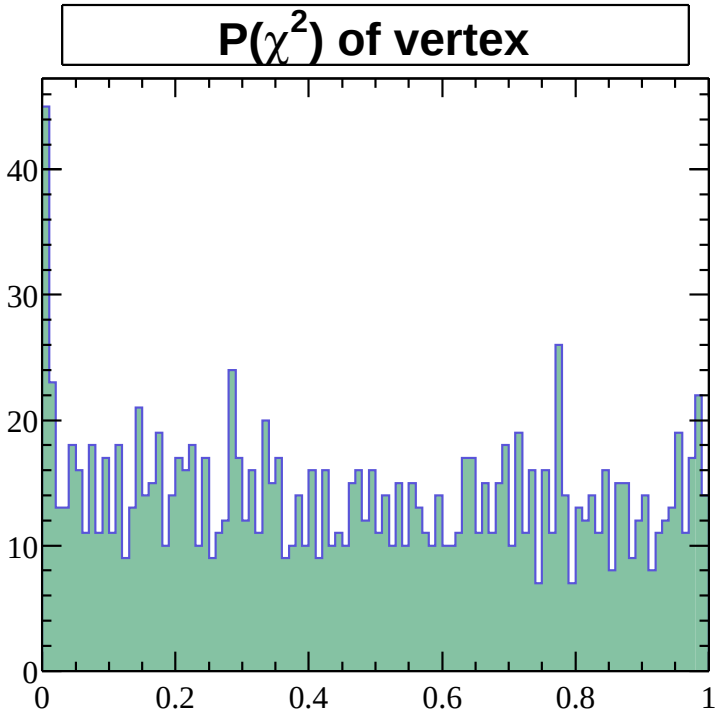




- comparable resolution
- VtxKalman has much better pull

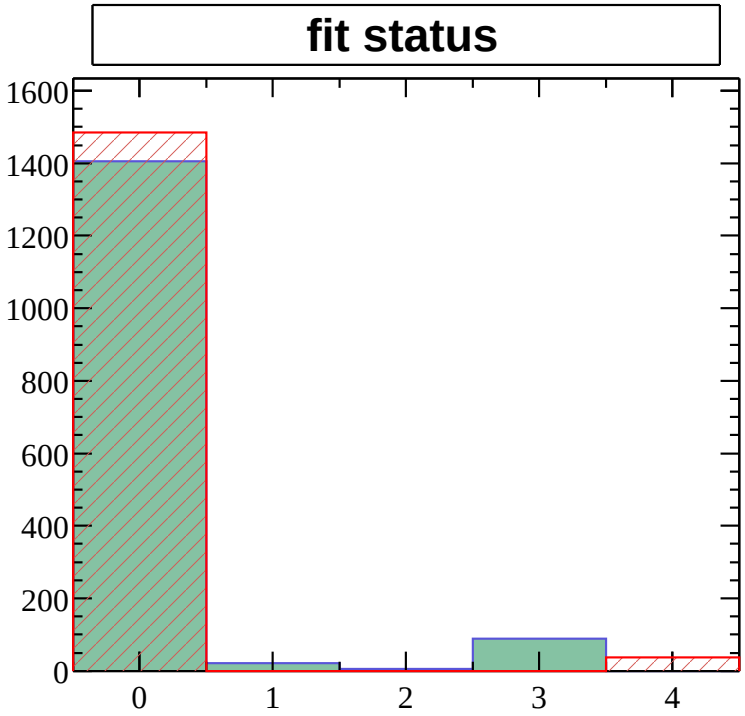
$K_S \to \pi^0\pi^0$: *CPU time,* χ^2

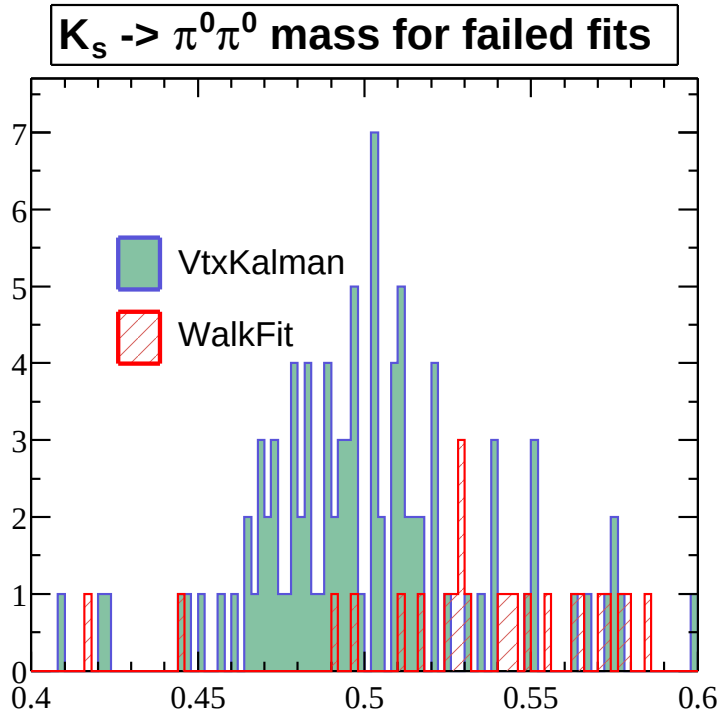




- both fits are slow
- VtxKalman approximately 10 times faster (~ 2 ms vs ~ 20 ms)
- VtxKalman χ^2 distribution looks 'perfect' (no χ^2 from the walk fit)
- note: since (almost) all particles from beam spot:,
  can interprete χ^2 as due to π^0 mass constraints
  $\Longrightarrow$ bad χ^2 *means* ≥ 1 bad π^0

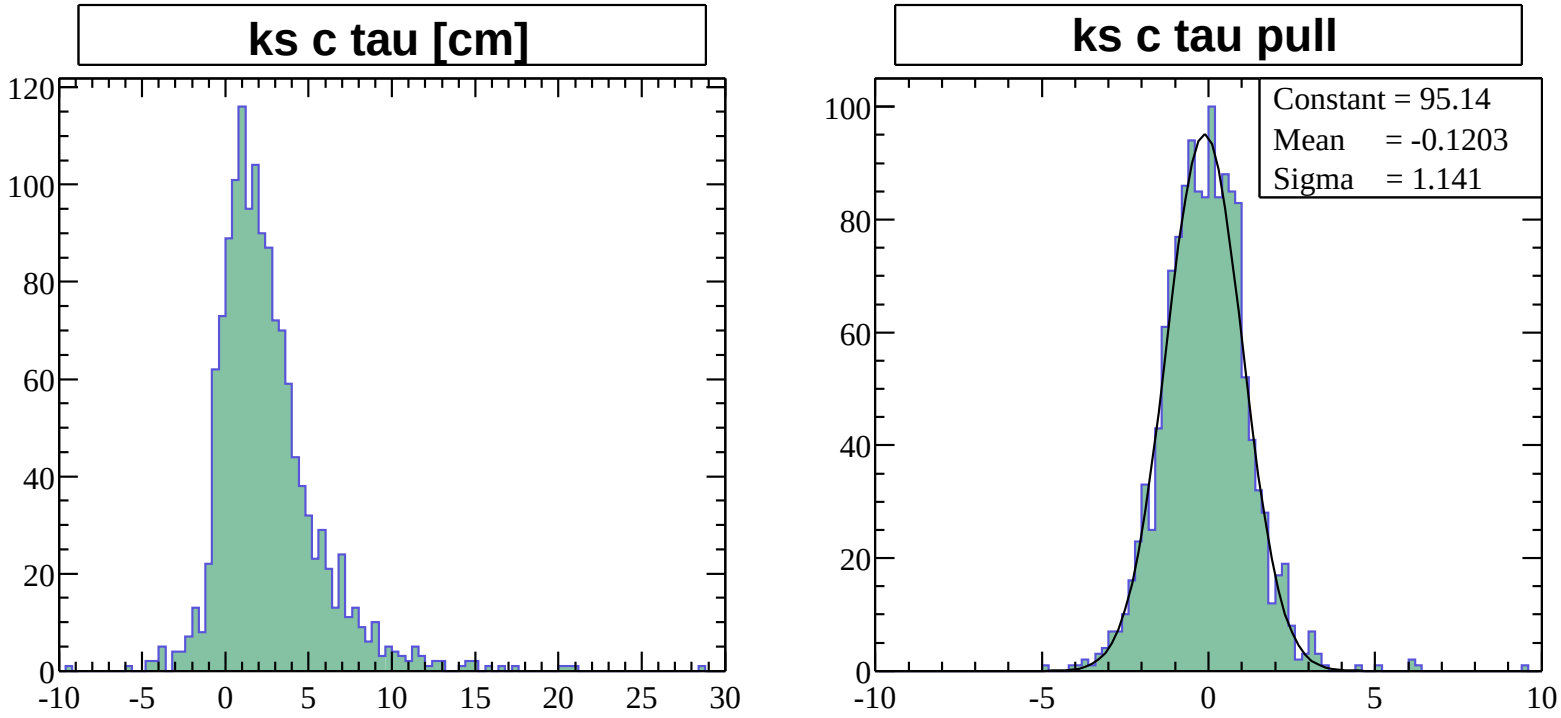
$K_S \to \pi^0\pi^0$: *Failed fits*





- VtxKalman fit 'fails' a bit more often
- much worse resolution for failed fits, but still signal
  $\Longrightarrow$ must find what that is
- (however, on the same events, WalkFit mass is *just as bad*)

$\mathrm{K_S} \to \pi^0\pi^0$: *decay length*



- $c\tau$ distribution looks good

$\Longrightarrow$ could use for background suppression

Problem 1: the order of constraints

- in a kalman fit constraints are applied in a specific order
- in non-linear fits the result is sensitive to this order
- example: $X \rightarrow Y + Z + \pi^0$ with a mass constraint π^0
  $\Longrightarrow$ geometric constraint on π^0 must be applied *before* mass constraint, otherwise the π^0 'walks away'
- order of constraints
  - all independent 'external' constraints (tracks, photons, beamspot, fitted composites)
  - kinematic constraints (four-vector addition)
  - geometric constraints (mother-to-daughter)
  - mass constraints

  scheme is not yet completely satisfactory

Problem 2: initialization

- kalman fits very sensitive to initialization (both pars and cov)
- parameters are initialized using TrkPoca, four vector addition, pdt lifetimes etc
  - it was hard to come up with a scheme that always works
    $\Longrightarrow$ the 'initialization' routine is the most complicated and fuzzy part of the package
- covariance matrix initialition is basically result of fine-tuning
- I keep hitting decays for which things do not work out well
  $\Longrightarrow$ will this converge?

Problem 3: machine accuracy

- the expression for the updated covariance matrix in the kalman fit is

$$C_k \;=\; (1 - K_k H_k)\, C_{k-1}\, (1 - K_k H_k)^T \;+\; K_k V_k {K_k}^T \qquad (1)$$

  identical to the simpler expression

$$C_k \;=\; (1 \;-\; H_k K_k)\, C_{k-1} \qquad (2)$$

  $\Longrightarrow$ much faster, but . . .

- the latter is known to be more sensitive to machine accuracy . . .

- *first time* I observe sensitivity to this effect

- managed to code expression (1) such that it is 'only' factor 2 slower than (2)

Problem 4: interfacing to Beta

Main problems:

1. some information cannot be stored in BtaFitParams
   - correlation between daughters
   - lifetimes

   $\Longrightarrow$ only accessible if explicitely calling 'vtxkalman::VtkVertex'
2. fitting not independent of the way the tree is built
   example: $\mathrm{B}^0 \to \mathrm{D}^{*+}\pi^-$
3. current behaviour of vertexers wro updating daughter parameters is confusing

Problem 4c: updated daughters

- by default, copying a BtaCandidate yields a *shallow* copy
  - the underlying data structure, the BtaCandBase, is not copied
  - decay trees *share* daughter parameters
- currently, vertexers make only shallow copies
  - only update the top layer of the decay tree
  - *never* update daughter parameters

  clearly, at odds with the idea of a decay tree fit
- to access 'daughter' parameters the daughter must be changed!
  - currently steered by 'fitAll' flag (but users do not understand this)
  - this daughter is changed *for ever*
- *it is a nightmare that the output could depend on the order in which the composition sequences are called*

Summary I: pros/cons of new fit

- \+ correlations in complete decay tree
- \+ consistent treatment of mother constraints (*e.g.* for $\mathrm{B}^0 \to \mathrm{D}^* X$)
- \+ photons treated as ecal clusters (*e.g.* for $\mathrm{K_S} \to \pi^0\pi^0$)
- \+ tracks treated as helices
- \+ beam spot constraint on production vertex
- \+ missing particles, lifetime constraints
- \+ applicable to practically all decay chains (can get rid of several dedicated fitters)
- — no kinematic fit (but do we really need that?)
- — kalman filter still 'fails' more often than global fit
- — decay chain fitting slightly at odds with 'fitting layers'

Summary II: status

- implementation roughly complete (5863 lines), first version in babar-cvs (VtxKalman)
- parameter pulls look fine, resolutions equivalent to GeoKin
- interfaced via VtxFitterOper (not in cvs)
- implemented 'Fast' mode and 'SingleTrack' mode; however, purely kinematic fitting not supported
- tested as replacement of GeoKin in the CompositionFactory
  - tested on different MC samples
  - tested on B-flavor sample (UpsilonQA)
    - small loss in yield ($\sim 2\%$) ...
    - both background and signal $\Longrightarrow$ probably failed fits
- a write-up of the Kalman-math is in progress: `www.slac.stanford.edu/~hulsberg/notes/vtxkalman.ps`

Summary III: plans

- try to recover remaining loss of signal events on B-flavor sample
- come up with a safe solution for 'updated daughter parameters'

$\Longrightarrow$ needs some redesign in VtxBase/VtxFitter

- ...

Part II: Δz Fitting

- common recipe:
  - remove tracks from reco-B (or *B-CP*)
  - estimate trajectory of tag-B from reco-B and beamspot/energy constraint
  - reconstruct vertex for tag-B from remaining tracks and the seed

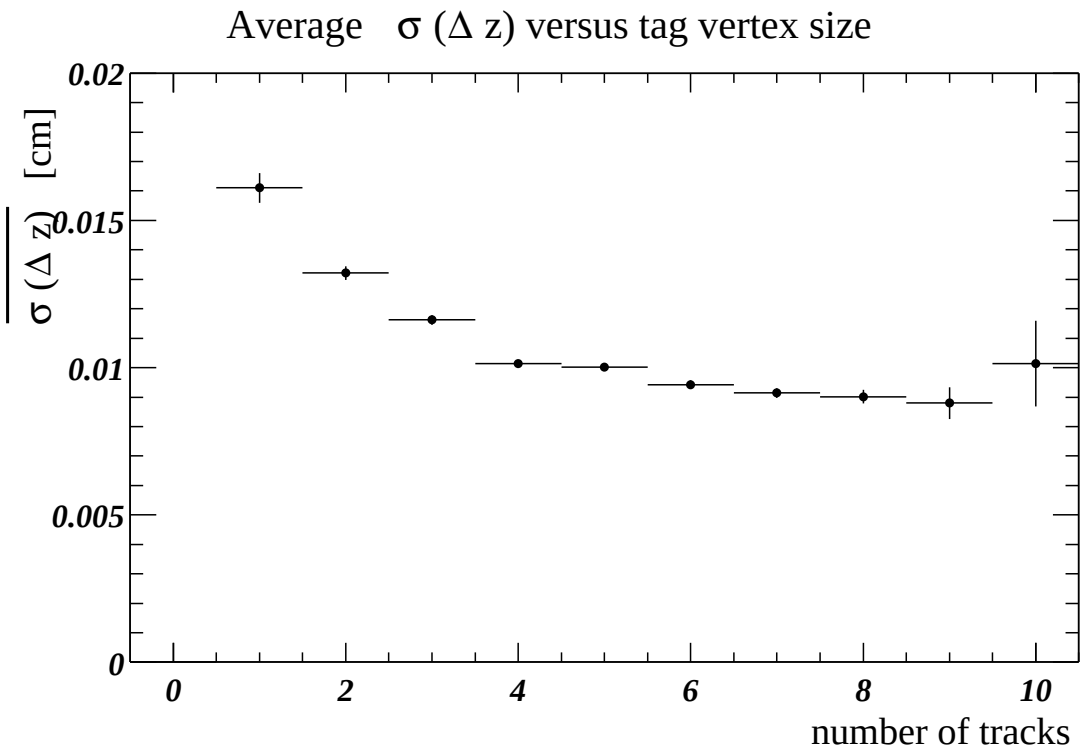
- in GeoKin:
  - combine all tracks into one vertex
    (after removing conversions and substituting $\mathrm{K_S}$tracks with mother)
  - *remove* incompatible tracks
    $\Longrightarrow$ tracks with χ^2 *contribution* $> \chi^2_{\mathrm{max}}$

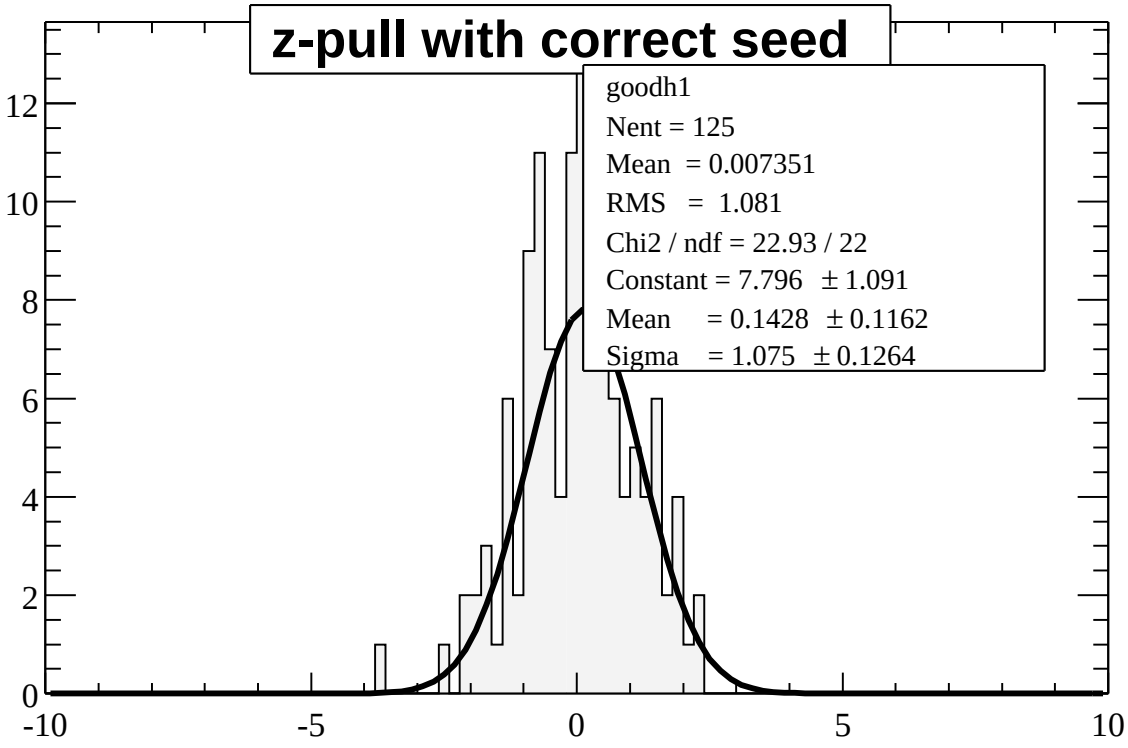
- alternative scheme:
  - start with a good seed
  - *add* compatible tracks
    $\Longrightarrow$ tracks with $\Delta\chi^2 < \chi^2_{\mathrm{max}}$

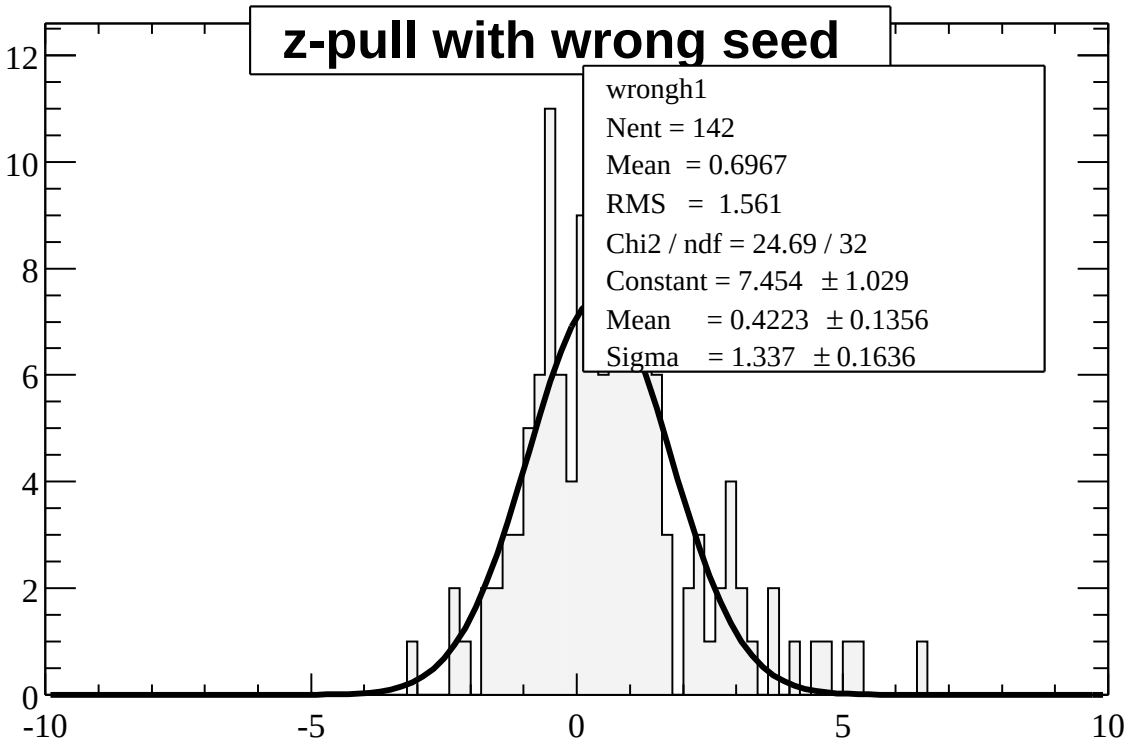
Tag vertex reconstruction

Obvious conflict:

- the more tracks, the smaller the statistical uncertainty
- the more tracks, the larger the bias due to D daughters







What can you gain with a good seed?

Seed how-to

It is important to choose a good seed, but how?

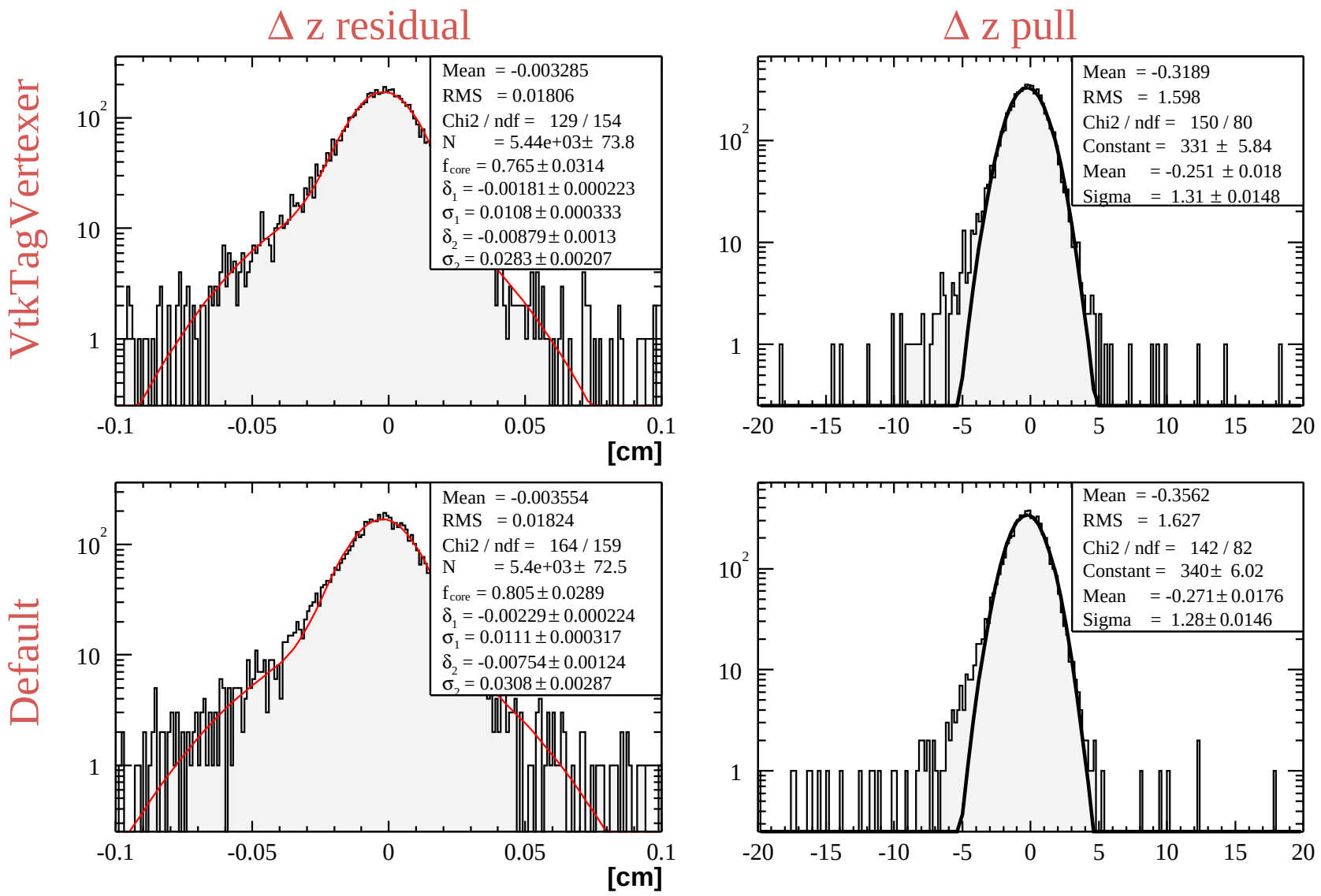
Looked at several observables

- $\chi^2 \Longrightarrow$ no good: slow tracks have large errors
- $p_\perp \Longrightarrow$ no good: *most* B do not yield 'one' high $p_\perp$ track
- best choice: likelihood 'density'

$$-2\log\mathcal{L} \;=\; \chi^2 \;+\; \log[\det(V)]$$

  where V is the covariance of the 'pseudo-B + seed-track' vertex
- (in fact, using σ_z^2 instead of full V makes little difference)

Results: Δz resolution



- no improvement in Δz resolution
- *maybe*, very small hint for reduced bias

Conclusions

- there exist an alternative vertex fit
  - full covariance matrix of a decay chain fit
  - better model of photons
  - no improvement in parameter resolution
  - small improvement in CPU time consumption
- there exists an alternative Δz fit
  - slightly more ‘advanced’ choice of tag vertex tracks
  - no improvement in Δz resolution
  - possibly, tiny reduction of Δz bias
- is it worth pursuing either of these projects?

Deep copies and refitted daughters (II)

Two solutions

1. only deep copies in decay trees
   - safe
   - not useful for vertexers that do not calculate updated daughters
   - severe penalty in memory consumption/CPU

   I would say that this is not a viable solution
2. shallow copies in decay trees, but access to daughter parameters only via deep copies
   - requires only small modifications in current design