# Analytical Track Fitting in Marlin

A.F.Żarnecki
Warsaw University

## Status report

- Track fitting method
- Algorithm development
- Algorithm parameter
- Conclusions and Plans

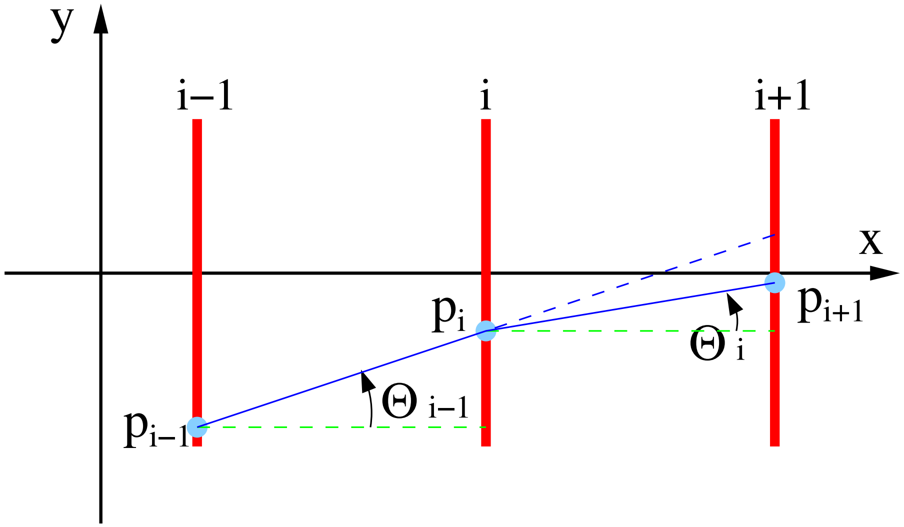
# Track fitting method

## Analytical approach

We determine track positions in each plane (including DUT), i.e. $N$ parameters $(p_i,\, i = 1 \ldots N)$, from $N-1$ measured positions in telescope planes $(y_i,\, i \neq i_{DUT})$.

We use constraints on multiple scattering!

Contribution of plane $i$ to $\chi^2$ of the fit



position measurement multiple scattering

$$\Delta\chi^2_i = \left(\frac{y_i - p_i}{\sigma_i}\right)^2 + \left(\frac{\Theta_i - \Theta_{i-1}}{\Delta\Theta_i}\right)^2$$

where: $\Theta_i = \dfrac{p_{i+1} - p_i}{x_{i+1} - x_i}$

Both terms present for planes $i \neq 1, i_{DUT}, N$,
first term missing for DUT, second for first and last plane

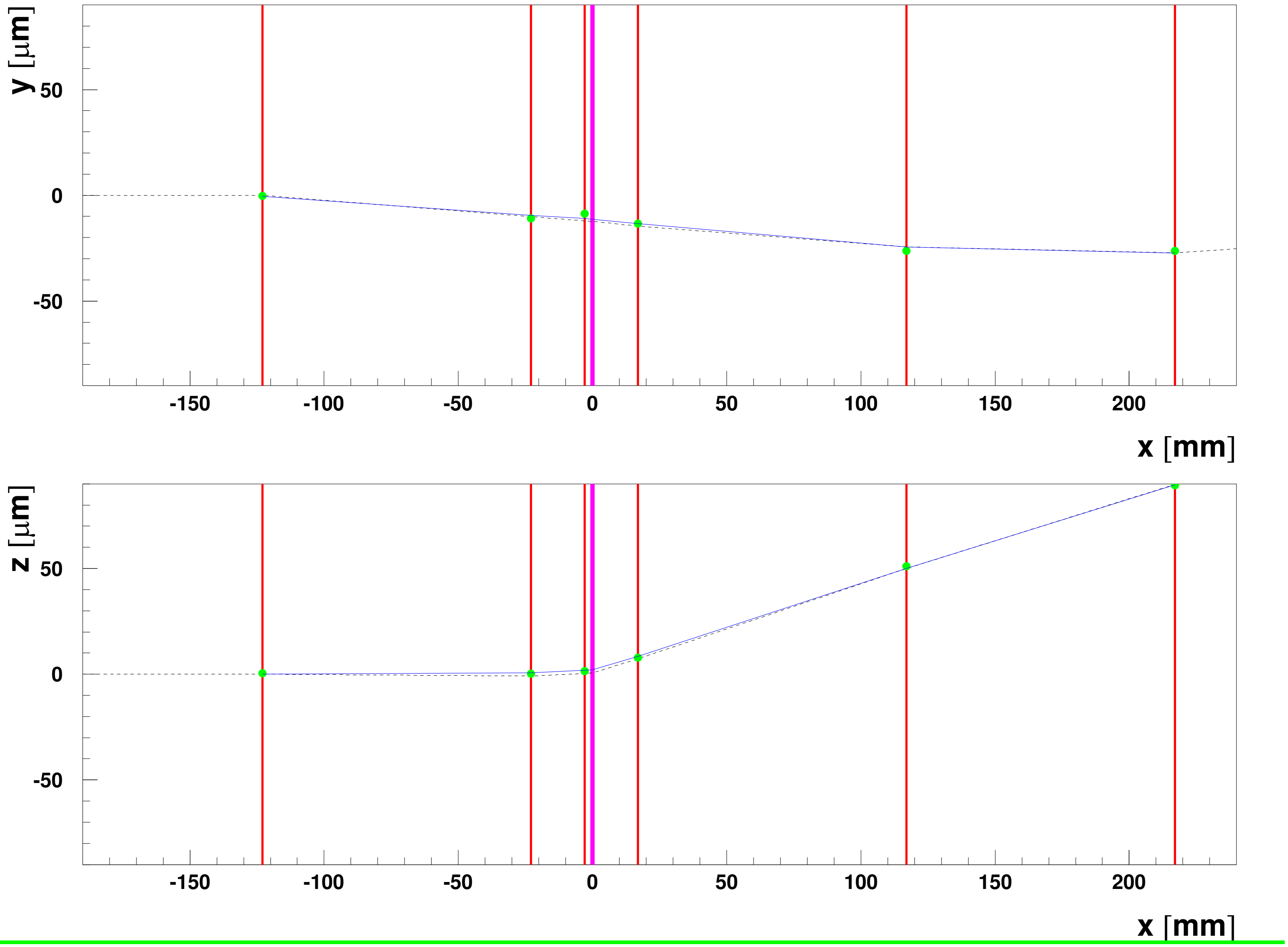
$\chi^2$ minimum can be found by solving the matrix equation.

As a by-product we get also an expected error on the position reconstructed at DUT.

# Track fitting method

## Simulation results

Example of the Geant 4 event



Color codes:

sensor planes

DUT

true particle path

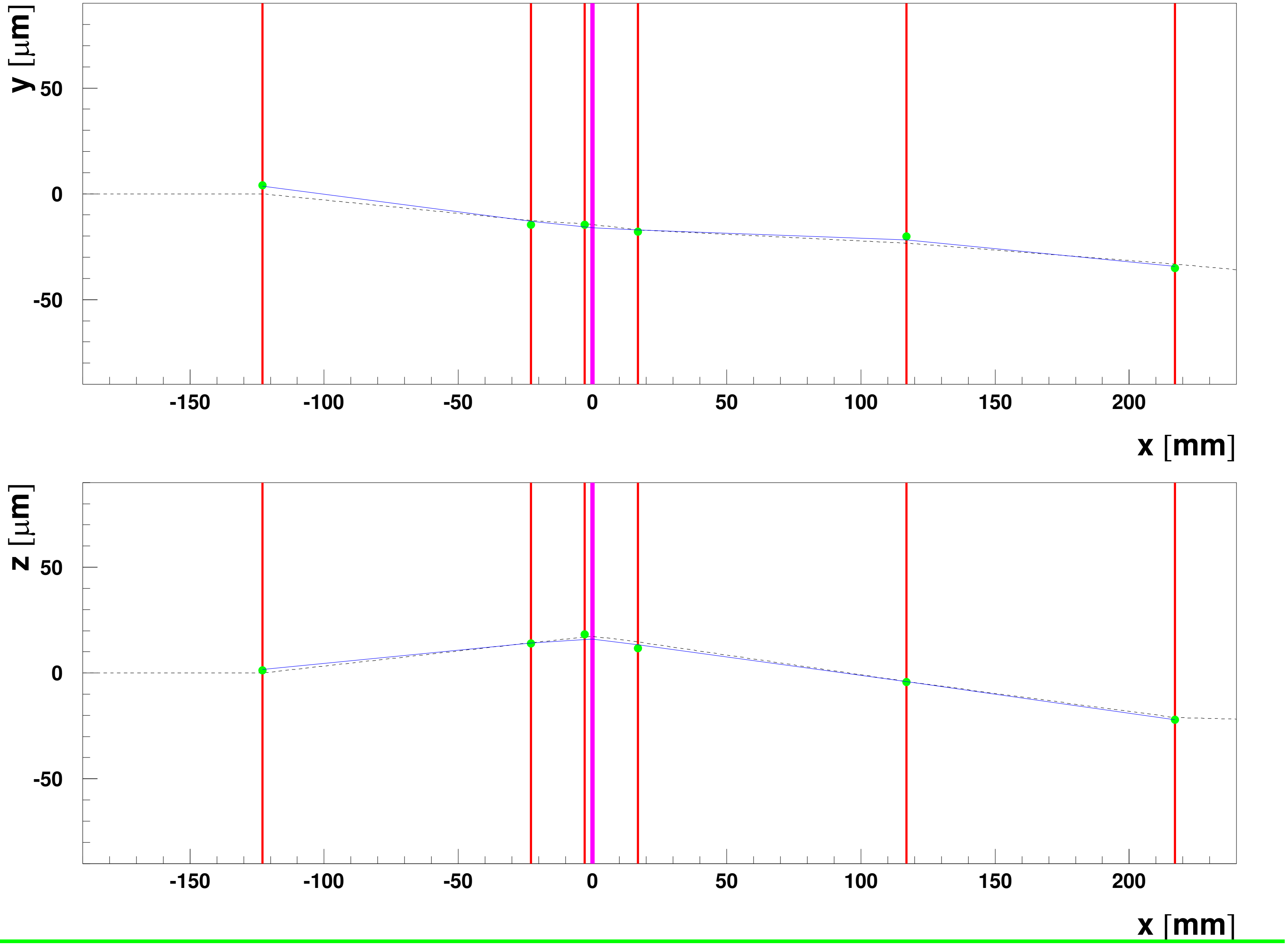
measurements

fitted track

# Track fitting method

## Simulation results

Example of the GEANT 4 event



Color codes:

sensor planes

DUT

true particle path

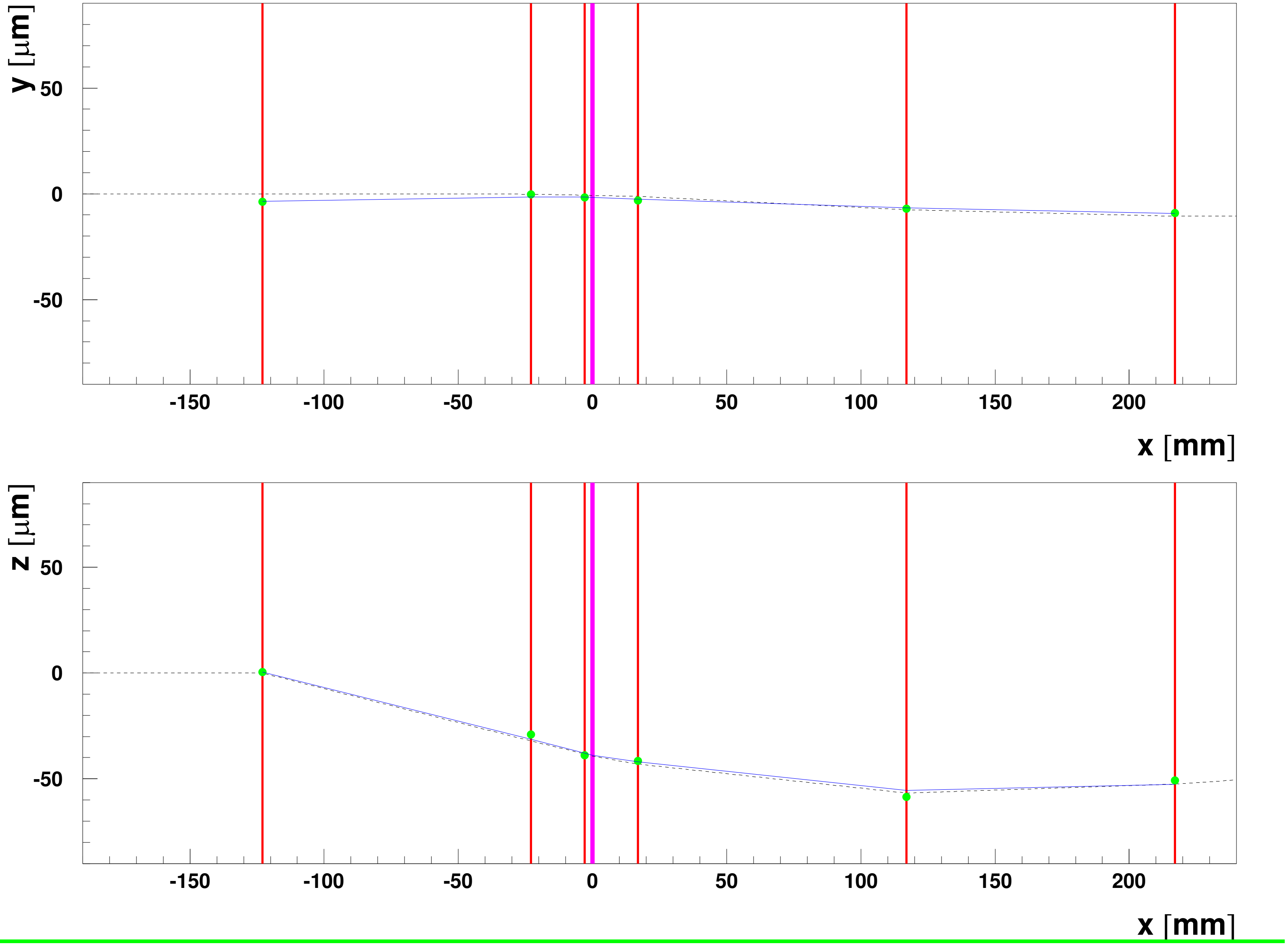
measurements

fitted track

# Track fitting method

## Simulation results

Example of the GEANT 4 event



Color codes:

sensor planes

DUT

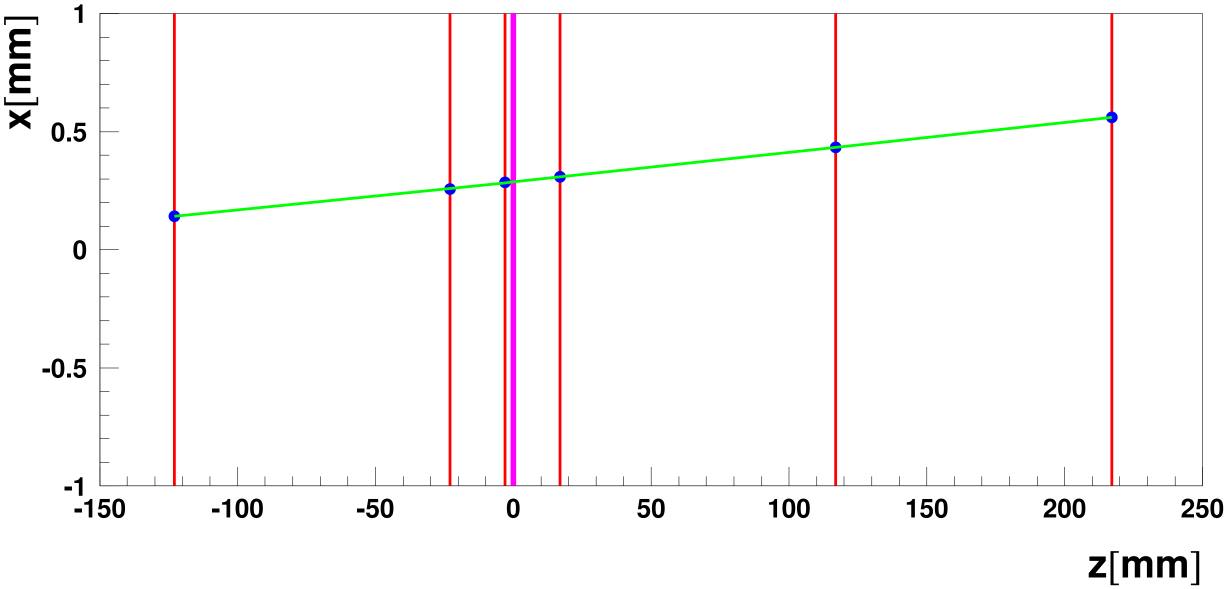
true particle path

measurements

fitted track

# Track fitting algorithm

## New development



The algorithm written in stand-alone FORTRAN was moved to LCIO/MARLIN environment

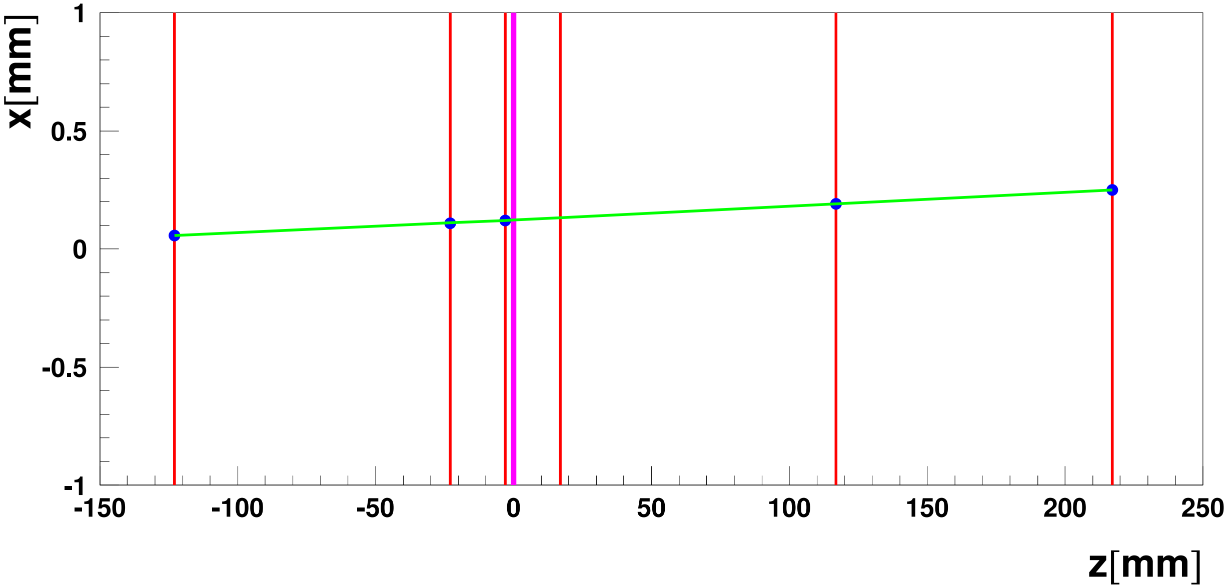
relatively easy

Hard part: algorithm works only for “ideal” events i.e. exactly one hit in each sensor

⇒ most of work invested to make it much more flexible...

# Track fitting algorithm

## New development



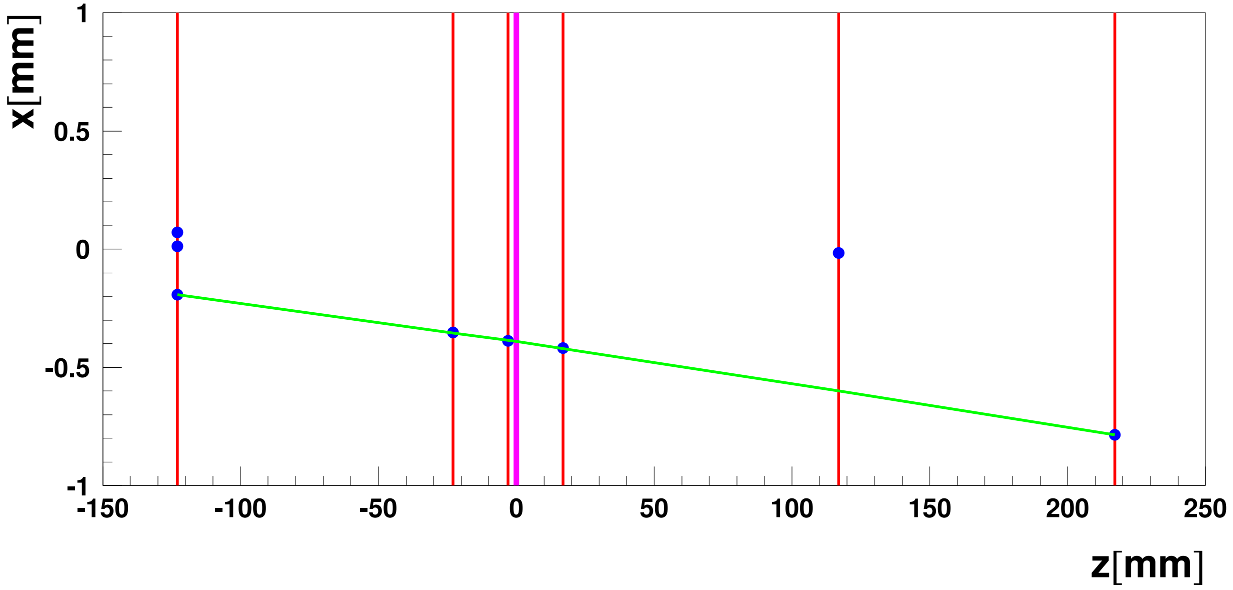
Sensor plane with missing hit ⇒ treat it as a nonactive layer

Full fitting procedure has to be repeated (fitting matrix changes)

Significant improvement of track finding efficiency !

# Track fitting algorithm

## New development



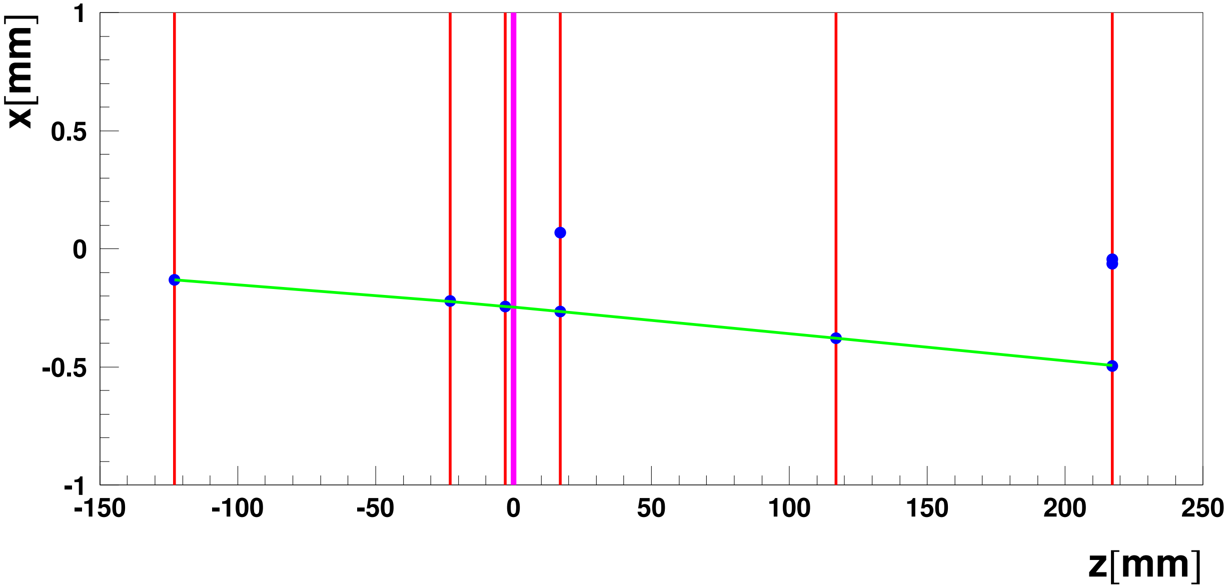
Missing hit + noise ⇒ consider possibility to “skip” one (or more) planes

Removing hit always results in better $\chi^2$ ⇒ introduce $\chi^2$ “penalty”

avoid plane skipping for good tracks

# Track fitting algorithm

## New development



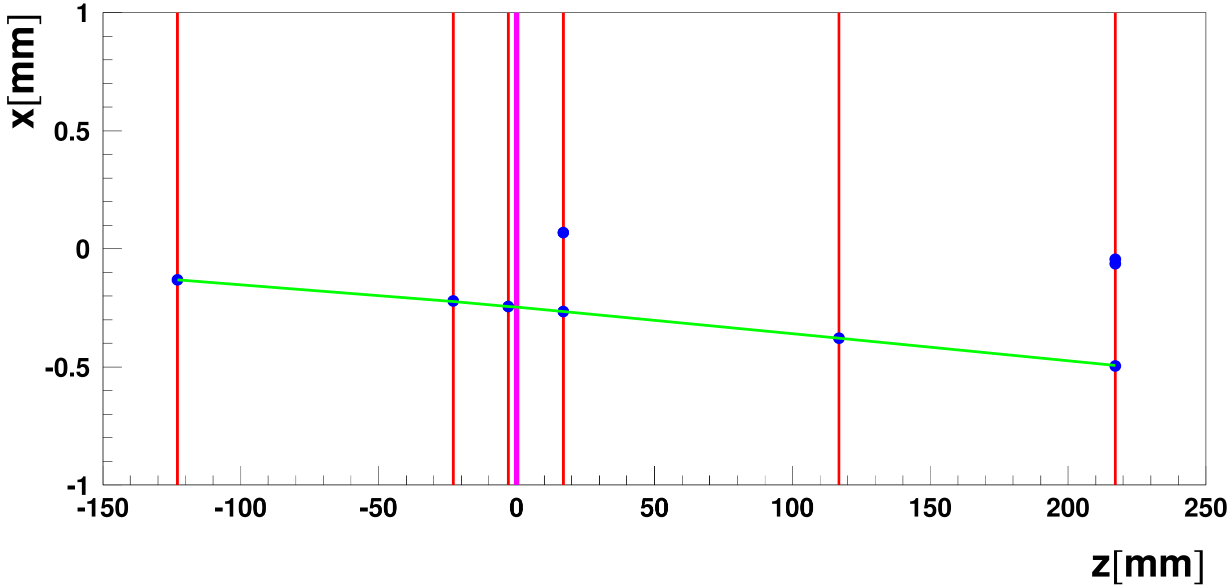
Additional hits (noise) ⇒ consider different hit selection hypothesis

Number of possible hit selections: no missing hits

$$N_{pos} = \prod_{i \in planes} n_i$$

# Track fitting algorithm

## New development



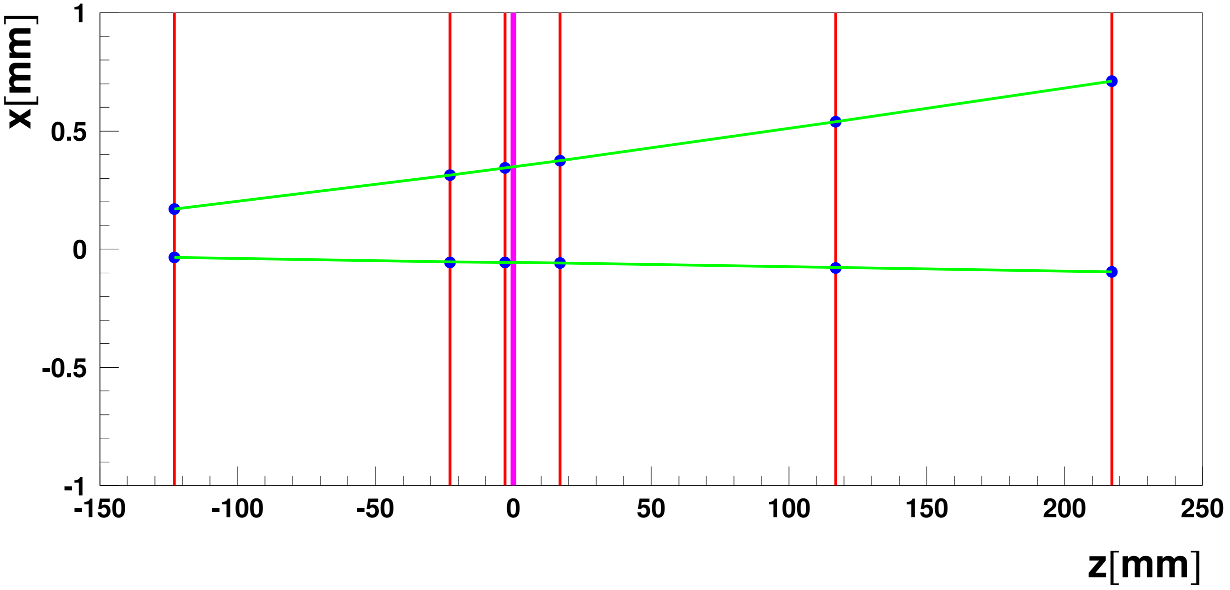
Additional hits (noise) ⇒ consider different hit selection hypothesis

Number of possible hit selections: with missing hits/plane skipping

$$N_{pos} = \prod_{i \in planes} (n_i + 1)$$

# Track fitting algorithm

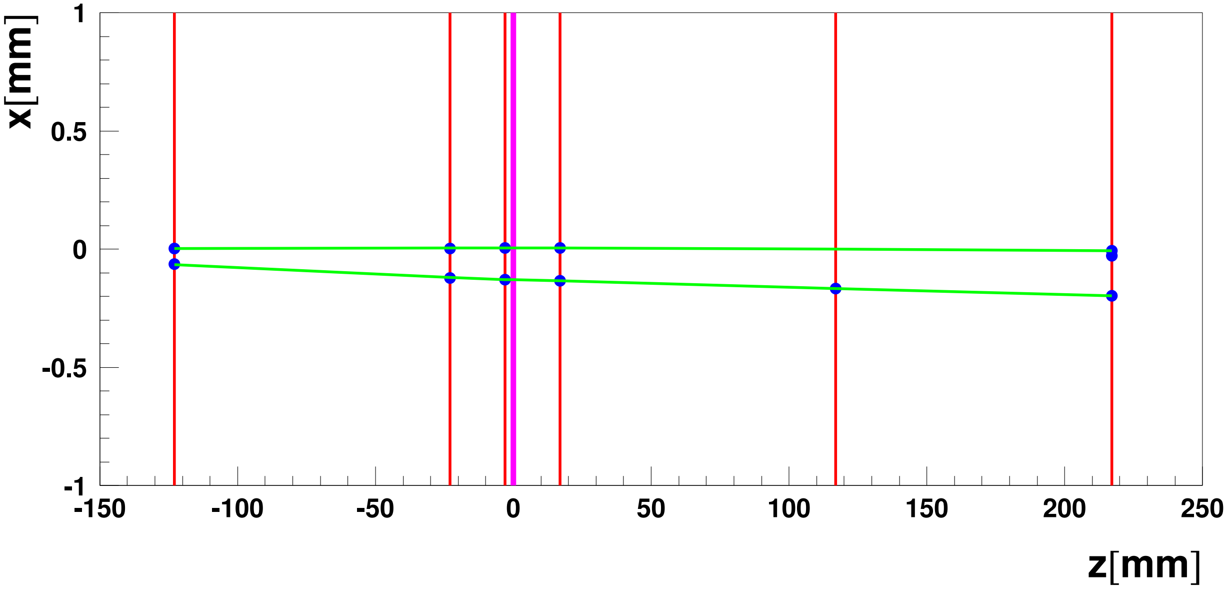
## New development



In general case more than one track can be found.

# Track fitting algorithm

## New development

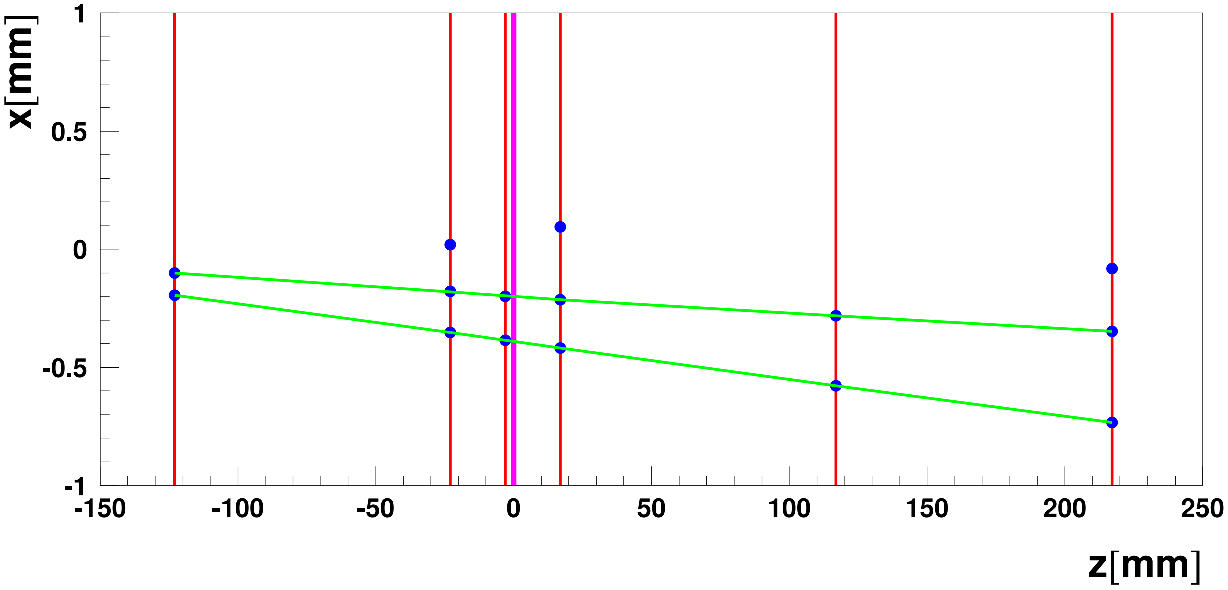


In general case more than one track can be found.

With missing hits

# Track fitting algorithm

## New development

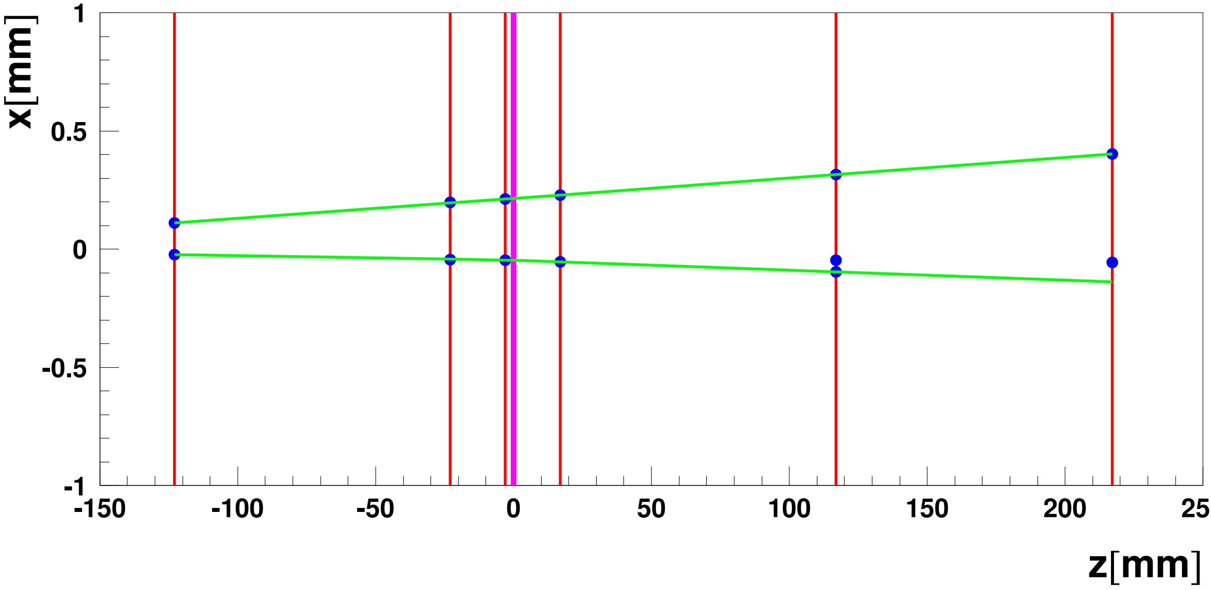


In general case more than one track can be found.

With missing hits or additional hits

# Track fitting algorithm

## New development



In general case more than one track can be found.

With missing hits or additional hits

No hit sharing between tracks allowed

# Track fitting processor

## TestFitter

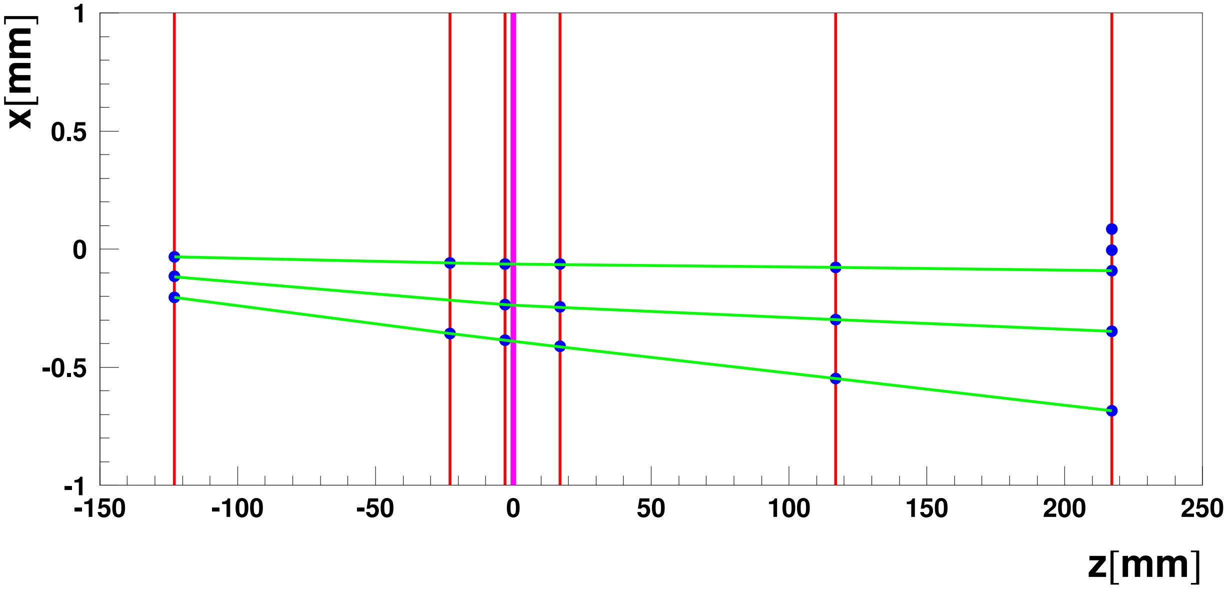
Summary of analytical track fitter processor algorithm:

1. Read measured track points from input TrackerHit collection and copy to local tables
2. Prepare lists of hits for each active sensor plane
3. Count hit numbers, return if not enough planes fired
4. Calculate number of fit hypothesis (including missing hit possibility)
5. Fit each hypotheses and calculate $\chi^2$ (including "penalties")
6. Select the best $\chi^2$ solution
7. Write fitted track parameters (positions and errors) to output TrackerHit collection
8. Remove best track hits from plane hit lists and goto 3

# Track fitting algorithm

## New development

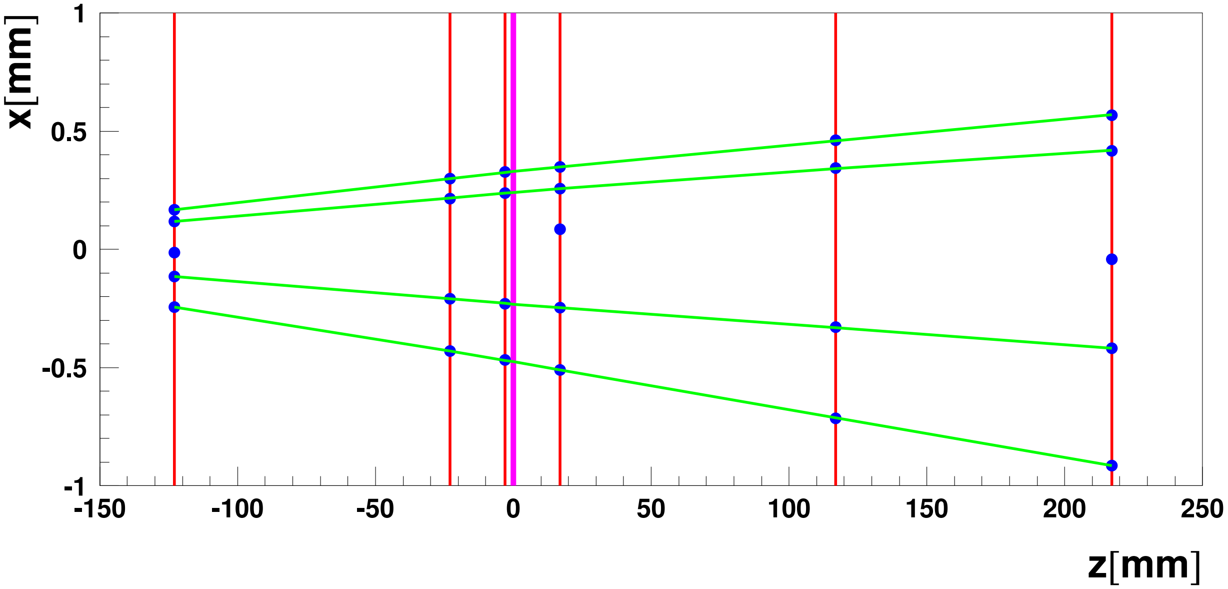
Example of multiple track fit from new algorithm



# Track fitting algorithm

## New development

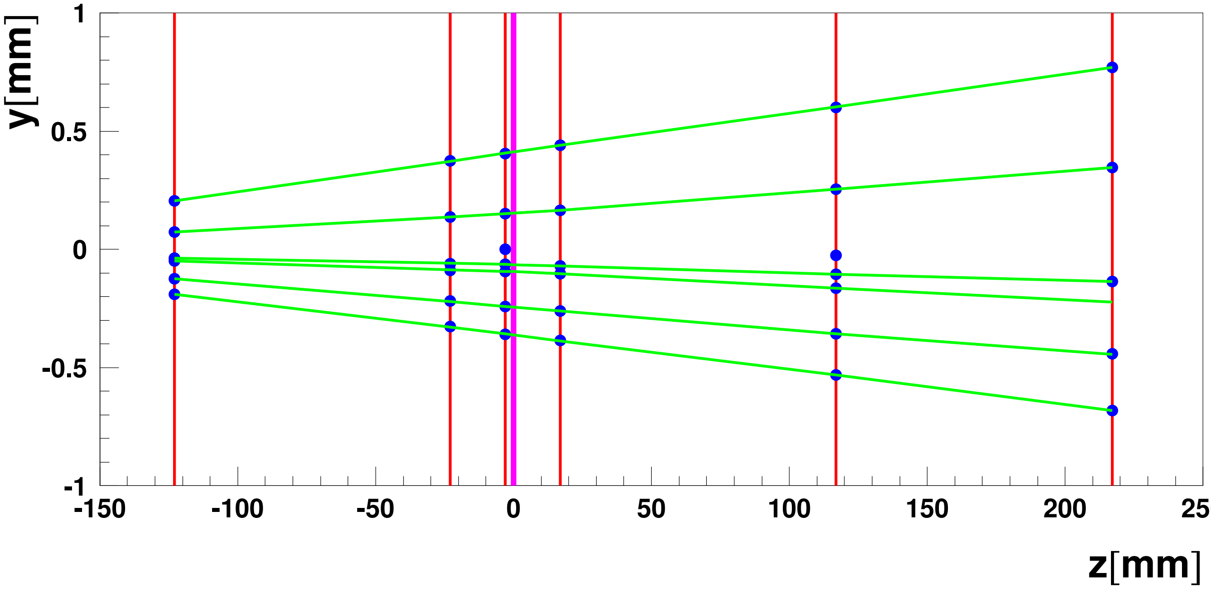
Example of multiple track fit from new algorithm



# Track fitting algorithm

## New development

Example of multiple track fit from new algorithm



# Track fitting processor

## TestFitter parameters

**AllowMissingHits** *default: 1*

Allowed number of missing hits in the track

**MissingHitPenalty** *default: 0*

Chi2 penalty for missing hit in the track

**AllowSkipHits** *default: 1*

Allowed number of hits removed from the track

**SkipHitPenalty** *default: 100*

Chi2 penalty for removing hit from the track

**Chi2Max** *default: 1000*

Maximum Chi2 for accepted track fit

# Track fitting processor

## TestFitter parameters

**SearchMultipleTracks** *default: true*

Flag for searching multiple tracks in events with multiple hits

**UseBeamConstraint** *default: false*

Flag for using beam direction constraint in the fit

*not tested yet in Marlin*

**UseDUT** *default: false*

Flag for including DUT measurement in the fit

**UseNominalResolution** *default: false*

Flag for using nominal resolution instead of position errors

*makes algorithm to run a little bit faster*

# Track fitting processor

## TestFitter parameters

**GeometryFileName** *default: geometry.dat*

Name of the geometry description file

*should be read from database !?*

**InputCollectionName** *default: meshit*

Name of the input TrackerHit collection

**OutputCollectionName** *default: testfit*

Collection name for fit output

**DebugEventCount** *default: 1*

Print out every DebugEnevtCount event

**Ebeam** *default: 6*

Beam energy [GeV]

*should be read from RunHeader !?*

# Track fitting processor

## Still missing

- Interface to geometry data base
- Additional checks for consistency of geometry description
- Fit statistics and histograms
  Is it possible to implement “on-line” graphical output?
- Output of full fit information (links to track hits, $\chi^2$)
  How to store multiple track fits ?
- Alignment corrections (from condition data base?)

Additional processors could use fit results to:

- Calculate alignment corrections
- Calculate plane efficiencies, efficiency maps, etc...
- Analyze DUT performance

# Summary

TestFitter processor ready and running in LCIO/Marlin environment

Not perfect C++ style, but working...

All designed functionality implemented

Needs geometry input and output data structure definition,
otherwise ready for public release