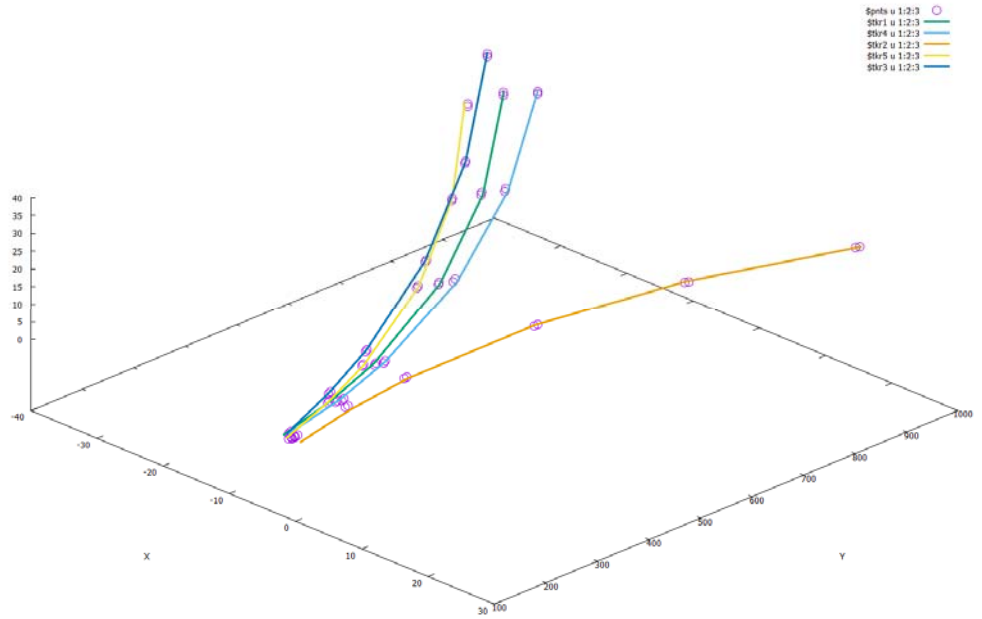
Kalman Filter Pattern Recognition

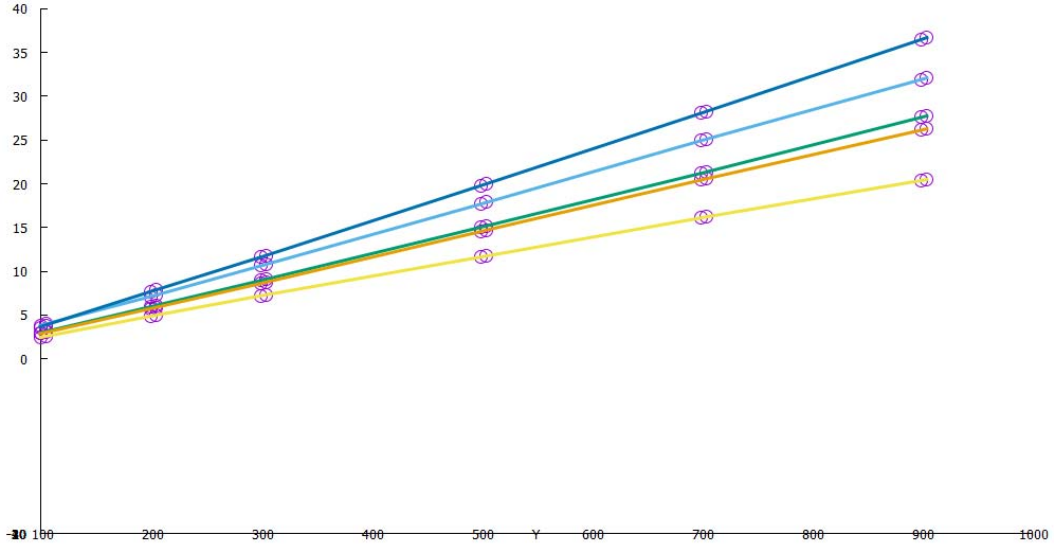
- I implemented a combinatorial pattern recognition based on the Kalman Filter track following and fitting.
- Seeds are generated from hits in 2 horizontal layers plus 3 stereo layers by a linear fit (line and parabola).
  - Multiple different sets of 5 such layers are tried.
  - All hit combinations in the 5 layers are tried and accepted or rejected according mainly to how closely they extrapolate to the origin. (No chi-square cut as this is a 0 d.o.f. fit.)
- The seeds are sorted by quality and then followed one-by-one with the Kalman filter, working toward the origin.
  - Tracks are allowed to share only a limited number of hits.
  - Shared hits are reviewed at the end and dropped or reassigned if appropriate.
- Final accepted tracks and covariance matrices are extrapolated by Runge-Kutta integration through the non-uniform field to the origin.

This is a silly example, with 5 particles of similar momentum starting from the origin point. Three of the five tracks have all 12 points and the other two are missing just the innermost hit.

This was the first try with such a large number of tracks, but usually it doesn't work this well. Things tend to get confused in the first layer or two.

Nevertheless, this shows that the logic is there, although not really tuned.

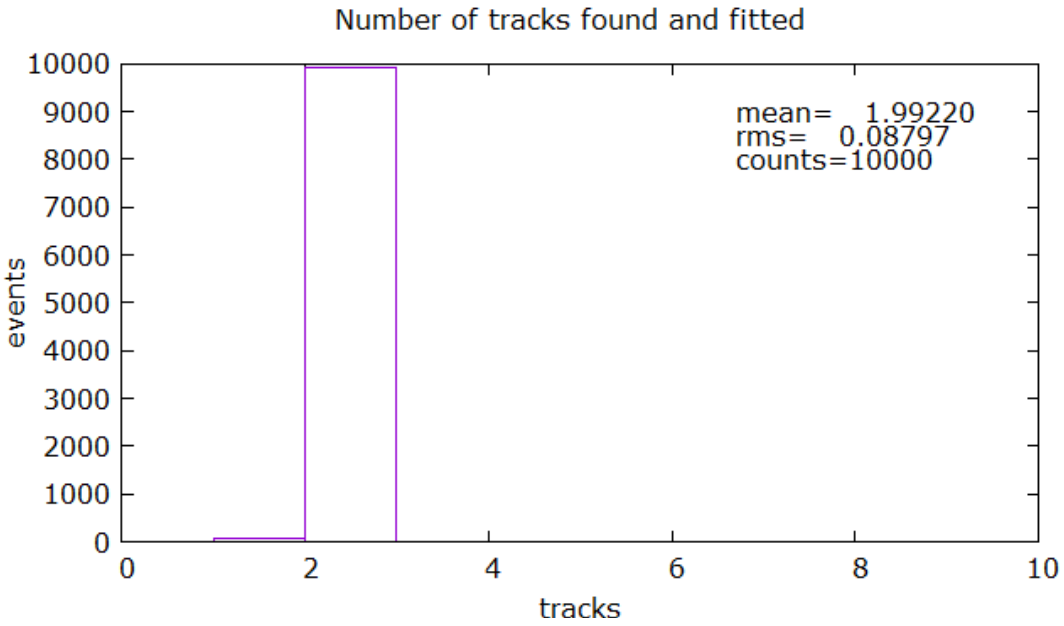


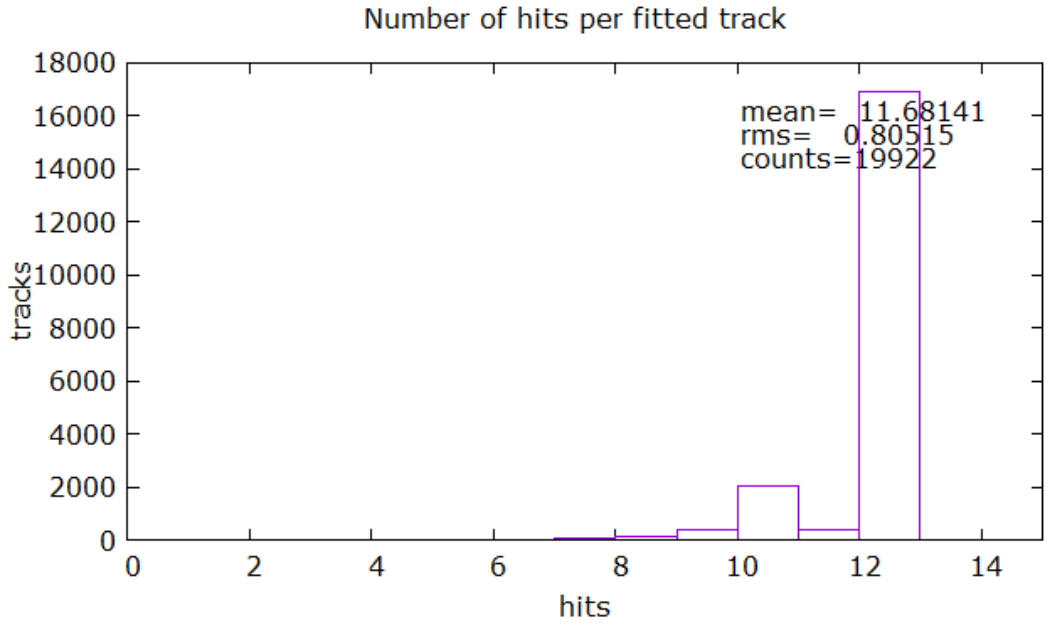


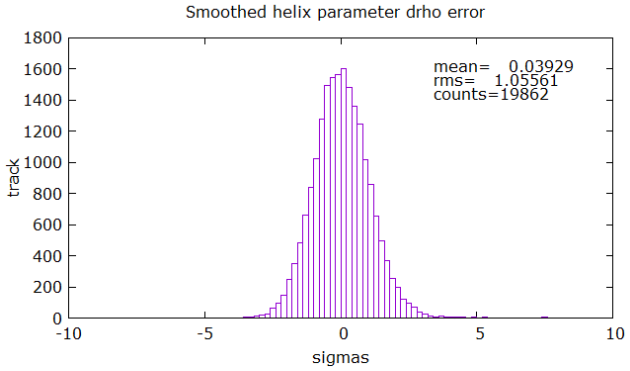
Testing with 10,000 events of two particle each, with momentum spread around 1 GeV and launched from the origin.

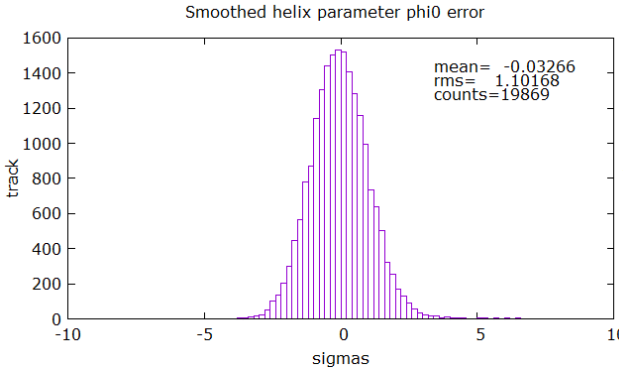
Almost all of the tracks are found, and most have all 12 hits.

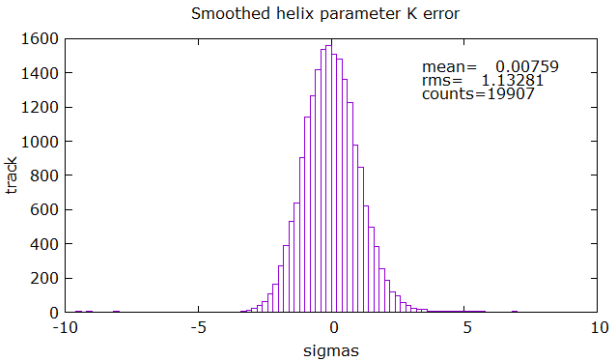
On the following page the reconstructed helix parameters, after swimming to the origin, are compared with the generated values at the origin.

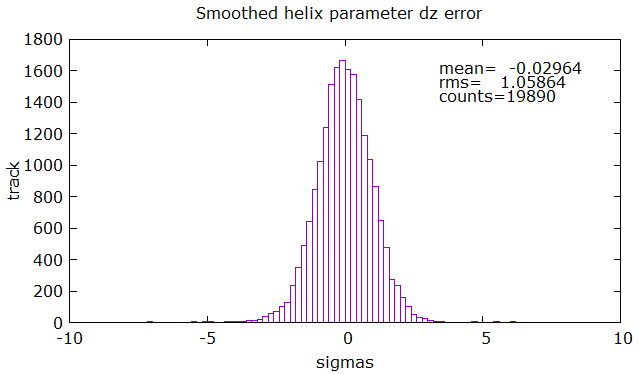


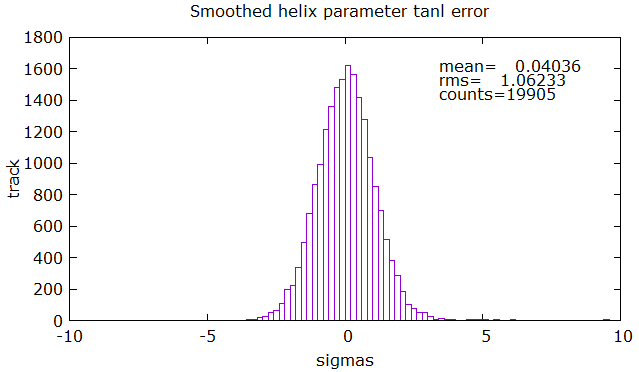




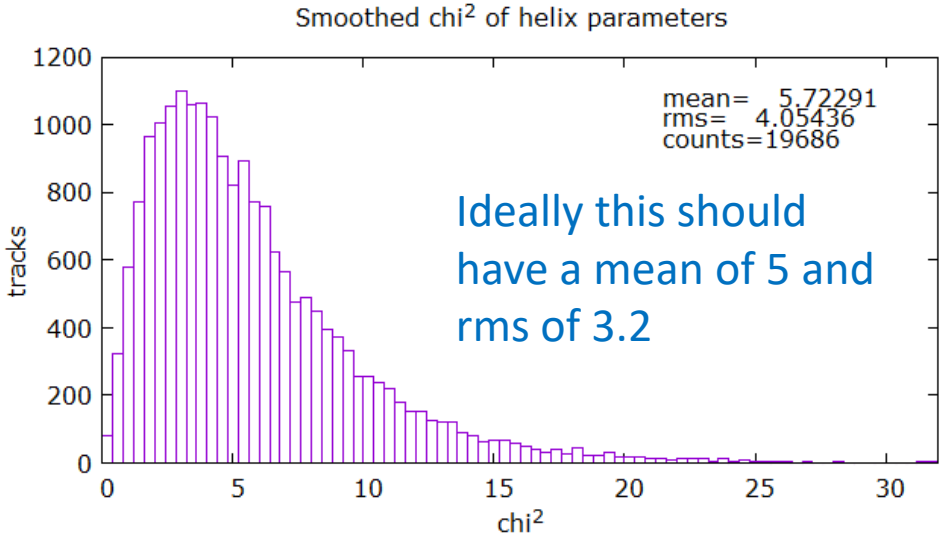








The Runge-Kutta extrapolation of the covariance matrix to the origin is not working perfectly, as the errors appear to be underestimated by several percent (despite being perfect at the first measuring plane).



Interfaces

- The HPS field map is read into the class FieldMap.java. At present the map is read from a binary disk file, but that can be changed.
- The positions and orientations (alignment) of the detector planes are loaded into the public class “SiModule.java” using the constructor, one object per detector plane (stereo or non-stereo).
- The hits (location of a hit strip in the detector frame) are loaded into “Measurement.java” using a public method in “SiModule.java”
- The public class KalmanPatRecHPS is instantiated and given an ArrayList of SiModule representing the set of hits in the upper or lower half of HPS. The constructor executes the pattern recognitions and fitting.
  - Thus far the cuts and parameters are hard-wired into “KalmanPatRecHPS.java” as local variables.
- The found and fitted tracks are put into an ArrayList of “KalTrack.java” objects.
- “KalTrack.java” has a number of public methods for refitting tracks, extrapolating them to the origin, etc.

Next Steps

- Work with Miriam to complete the interface with the HPS code.
- Test with MC events and tune the parameters and algorithms to optimize performance and execution speed.
- Tie up some loose ends. For example, “KalTrack.java” could easily include a method to extrapolate to the electromagnetic calorimeter, if needed.
- Test with some real data.