Subject: Re: Fitting scheme
Posted by jabernathy on Tue, 11 May 2010 18:44:26 GMT
View Forum Message <> Reply to Message

Hello all,

Here are my thoughts a fitting scheme:

I like the idea of a TrackFitterBase. From experience, trying to put all of the functionality in one class can be unwieldy.

However, after that it is more complicated.

As it was pointed out before, there are different methods (and meanings) of the residuals. One has to be careful that the correct routine is called for a fitted track.

One idea that I had previously was to let the fitting processor be responsible for calculating the residuals. For example, a reconstruction steering file could contain this processor:

```
	<processor name="MyTrackFitterLikelihood" type="TrackFitterLikelihoodProcessor">
  <parameter name="Sigma0" type="double">0.81013581</parameter>
  <parameter name="FitSigma0?" type="boolean">true</parameter>
  <parameter name="Noise" type="double">0.001</parameter>
  <parameter name="SigmaZ" type="double">0.5</parameter>
  <parameter name="ProduceTrackTuples?" type="bool">true</parameter>
  <parameter name="PathIntegralStepSize" type="double">25</parameter>
  <parameter name="OmegaTau" type="double">10</parameter>
  <parameter name="DriftVelocity" type="double">35</parameter>
  <parameter name="ReadoutElectronicsInnerZ" type="double">2245</parameter>
  <parameter name="TPCOuterRadius" type="double">1800</parameter>
  <parameter name="FitNonHomogenousFields" type="bool">true</parameter>
 </processor>
```

Then the analysis code could run the same processor except with a flag telling the fitter to only calculate the residuals:

```
	<processor name="MyTrackFitterLikelihood" type="TrackFitterLikelihoodProcessor">
  <parameter name="CalculateResiduals">true</parameter>
		...
 </processor>
```

One advantage to using this method is that the fitter could check the track to see if it was fit using this method, and quit gracefully, if it wasn't. It also allows most of the code to be kept in one place. Note, I am not suggesting that the processor _is_ the fitter, they could still use a sub-class to manage the fitting and residual calculations.

One disadvantage of this method is that I don't think analysis logic should be placed in the fitter. If the residual calculation is done with the fitting processor then it is probably going to be an "all-or-nothing" choice of which tracks to calculate residuals for. This may not be a bad thing - a _real_ analysis processor could just ignore any unneeded information.

Another method that I thought of was keeping the track fitter processors in pairs: one for fitting and one for residuals. So there would be a TrackFitterLikelihoodProcessor in the reconstruction chain and TrackResidualLikelihoodProcessor in the analysis chain. This may not be a bad idea. It allows logic to be located where it needs to be. Code could be shared by using the common object inherited from the TrackFitterBase.

I hesitate to suggest using a factory method for calculating the residuals because it seems to be moving in an orthogonal direction to the intent of Marlin - keeping the chain "modularized". Also, if a factory method/processor is used then it could be difficult to manage the processor parameters because each residual method is parameterized differently.

However, I must admit, I don't know a great deal about the analysis side of things.

Is it common for a objects to be reconstructed with one method and then fit with another?

When analysis is done is it usually done to all of the tracks or just ones which are interesting?