Geane Track Fitting

Nicholas Kinnaird

May 16, 2017

Abstract

In this document I will detail the concepts and mathematics of GEANE track fitting, as well as it's implementation in the gm2 simulation framework for the Muon g-2 Experiment at Fermilab. I will also detail the pecularities and intricacies of the code. This is done both for documentation purposes as well as a precursor to sections of my future thesis.

1 Introduction

The Muon g-2 Experiment at Fermilab will use tracking detectors in order to measure positron trajectories for the purpose of determining the beam distribution and its characteristics, both for the final ω_a result and for general beam diagnostics. A fraction of positrons originating from decaying muons will pass through the trackers. By fitting these tracks and extrapolating back to the average decay point, the beam can be characerized in a non-destructive fashion. The tracking can be done independently from the calorimeters in order to cross-check, or combined with them in order to provide the muon distribution that the calorimeters see for the direct ω_a calculation. It is arguable that this is the most important duty of the trackers. (Note that the integration of the two detector systems in the code has not been initiated.) It is also worth noting that there is a large percentage of tracks that hit the calorimeters hit zero or a small number of tracking modules, which this fitting code is not capable of handling.

Each tracker module consists of 4 layers of 32 straws with a stereo angle of 7.5 degrees, the first two "U" layers oriented with the tops of the straws at a greater radial position, and the second two "V" layers oriented with the bottoms of the straws at a greater radial position. A tracking module is shown in Figure 1. There are 3 tracker stations located at the 0, 12th, and 18th sections of the ring, counting clockwise from the top most point of the ring where the inflector resides. (Station 18 has been installed at time of writing, with station 0 planned for the fall. Station 12 is slated for the future sometime.) Each station consists of 8 tracking modules arranged in a staircase pattern that follows the curvature of the ring as seen in Figure 2. Further hardware and electronics information regarding the trackers will be omitted in this document.

Because of the proximity of the trackers to the muon beam, they will lie within a region of varying field. The radial field of the trackers rises from 0 Tesla at the outer ends to roughly .3 Tesla at the inner top and bottom ends, and the vertical field drops approximately 50% from the storage dipole field of 1.451 Tesla. Shown in Figures 3 and 4 is the location of the tracker with respect to the horizontal and vertical fields respectively. If one thing can be said to be special to g-2 when it comes to tracking it is these large field gradients over the tracking detector region and several meter long extrapolation distance back to the muon decay point. This is one of the main motivations for using the Geane (Geometry and Error Propagation) fitting algorithm and routines, which has direct access to the field.

The Geane fitting routines originated in Fortran with the EMC collaboration, and was used in the precursor E821 experiment as well as the PANDA experiment with some success. (cite stuff here?) (There might be some other instances of its use as well.) The core error propagation routines were at some point added to Geant4 under the error_propagation directory which is included in all default installs. The tracking code strengths lie with its direct implementation and access to the Geant4 geometry and field, and its ability to handle the field inhomogeneties. The Geane algorithm code which makes use of the Geant4 error_propagation routines follows the structure of [1] and is detailed in the Formalism section in this paper. It is a relatively straight forward least squares global χ^2 minimization algorithm.

Figure 1: Shown is a picture of one of the many tracking modules used in the Muon g-2 experiment. The first layer of straws with a stereo angle of 7.5 degrees can be seen, with the other 3 straw layers hiding behind it. The beam direction is roughly into the page in this picture, and this view is what the decay positrons will see. Picture provided by James.

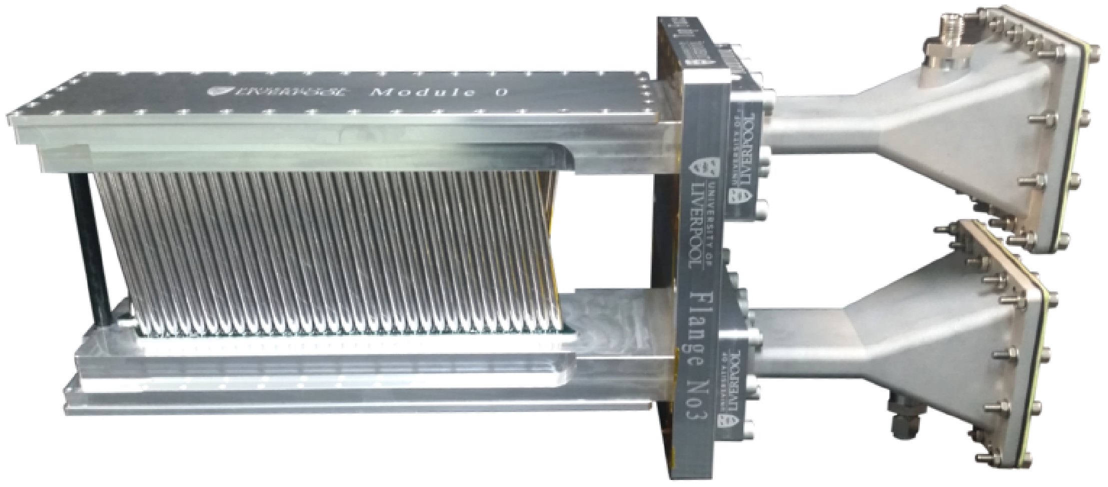


Figure 2: Tracker modules are arranged in the shown staircase pattern. In green and dark blue is the edge of the vacuum chamber (where the dark blue identifies the modification that was made to the old vacuum chambers), and it can be seen that vacuum chamber walls lie at the ends of the outside tracking modules. The position of a calorimeter can be seen in teal at the right. The dark red spots are the locations of the pole tips.

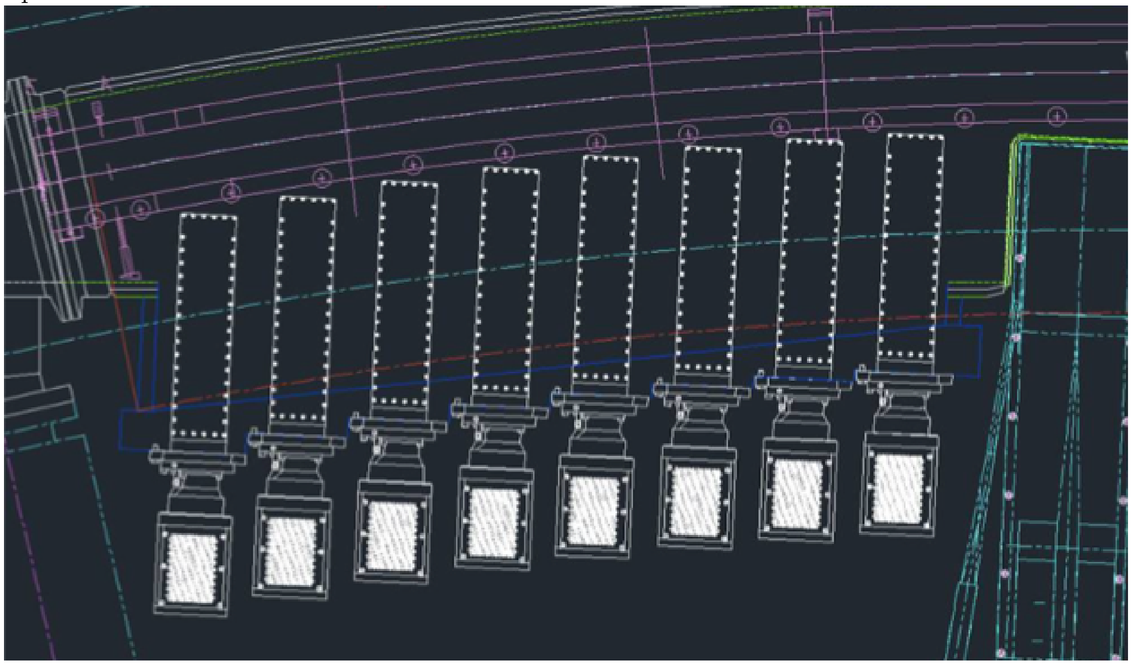


Figure 3: Shown is the vertical field of the g-2 magnet in and around the storage region as calculated in Opera 2D. The center of the storage region lies at 7.112 m along the x axis. The black box shows the rough location of the tracker with respect to the field (size exaggerated slightly). It can be seen that there is a large inhomogeneity within the tracker space, goring from left to right.

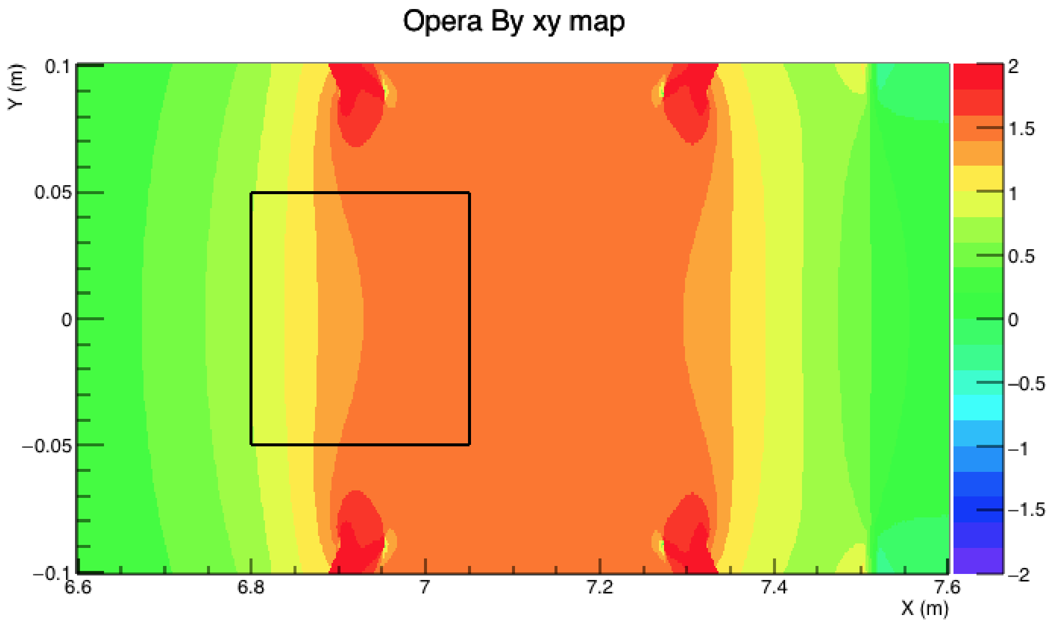
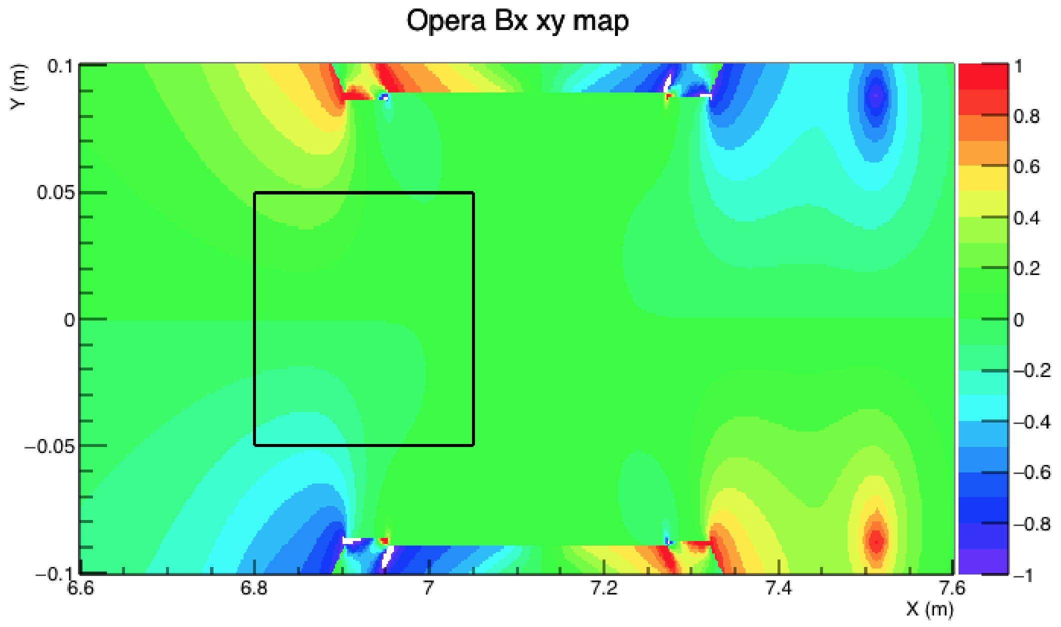


Figure 4: Shown is the radial field of the g-2 magnet in and around the storage region as calculated in Opera 2D. The center of the storage region lies at 7.112 m along the x axis. The black box shows the rough location of the tracker with respect to the field (size exaggerated slightly). It can be seen that there is a large homogeneity at the inner upper and lower ends compared to the right center. The shape of the pole pieces and tips can readily be seen.



2 Formalism

The author recommends reading the references [1], [2], and [3] in order to best understand the fitting algorithm. However, due to the at times confusing notation, ommitted equations or concepts, and differences between papers, I have attempted to summarize here the different sources and present the material in a more understandable and readable format, while at the same time drawing connections to the implemented code.

One can define a χ^2 for a track in the normal way by dividing the residuals of measured and predicted track parameters by their errors,

$$\chi^2 = (\vec{p} - \vec{x})^T (\sigma)^{-1} (\vec{p} - \vec{x}), \tag{1}$$

where $\vec{p}$ are predicted track parameters from a fit to the measured track parameters $\vec{x}$, and σ is a matrix of errors on the fitted parameters.

The Geant4 error_propagation routines can be used to determine these predicted parameters and error matrices by propagating track parameters from some initial guesses. By minimizing this χ^2 with respect to the track parameters one can then fit and improve the track. The Geant4 routines propagate particles along their average trajectories neglecting the effects of discrete processes, using a helix equation along small enough steps where the change in the magnetic field is small. The parameters are then a function of path length,

$$p_l = F_{l,l_0}(p_0), \tag{2}$$

where the coordinate system transforms described above can be used to determine parameters at specific X values, or detector planes. (From here on the dependence on path length or X position will be neglected, in favor of using plane indices.) In tandem, error matrices describing the expected distribution in true parameters about those predicted parameters due to said discrete process are also calculated:

$$\sigma^{ij} =< p^i p^j > - < p^i > \cdot < p^j >, \tag{3}$$

where i and j are track parameter indices.

The propagation of these parameters and error matrices are done using transport matrices, which express the infinitesimal changes in parameters at some plane (or path length) with respect to the parameters at some previous plane (or previous path length):

$$\delta p_N = T_{N,N-1} \delta p_{N-1}, \tag{4}$$

$$\sigma_N = T_{N,N-1} \sigma_{N-1} T^T_{N,N-1}. \tag{5}$$

Said transport and error matrices are 5x5 objects since the parameter vectors are 5x1 objects as described above. With parameters defined on such planes, one can define the χ^2 as:

$$\chi^2 = \sum_{i=1}^{N} [(p_i(p) - x_i)^T (\sigma_i)^{-1} (p_i(p) - x_i)], \tag{6}$$

where p_i are the average predicted parameters from some general starting parameters p. At first order one can only include the measurement error on parameters, which fill in the diagonals of σ_i, if random processes can be neglected. It is important to note that unmeasured parameters should have measurement errors of ininity (or some large value) along the diagonals in the code, which save the fact that residuals for unmeasured parameters do not exist. When the error matrix is inverted all rows and columns of the matrix with these large numbers will fall to 0 in the χ^2.

In order to get the best fit track, the χ^2 should be minimized with respect to the initial track parameters p, and evaluated at some chosen parameters:

$$\frac{\partial \chi^2}{\partial p}|_{p=p_0'} = 0, \tag{7}$$

resulting in

$$0 = \sum_{i=1}^{N} [(\frac{\partial p_i(p)}{\partial p}|_{p=p_0'})^T (\sigma_i)^{-1} (p_i(p_0') - x_i) \tag{8}$$

$$+(p_i(p_0') - x_i)^T \frac{\partial (\sigma_i)^{-1}}{\partial p}|_{p=p_0'} (p_i(p_0') - x_i) \tag{9}$$

$$+(p_i(p_0') - x_i)^T (\sigma_i)^{-1} (\frac{\partial p_i(p)}{\partial p}|_{p=p_0'})] \tag{10}$$

where the 1st and 3rd terms are identical, and the 2nd term is small if one assumes that error matrix doesn't change much with respect to the starting parameters. (Fair since most of the error comes from measurement, and as long as the initial guess is decent enough such that the path length through material doesn't change much from one iteration to the next.)

This simplifies to:

$$\sum_{i=1}^{N} T_{i0}^T (\sigma_i)^{-1} (p_i(p_0') - x_i) = 0. \tag{11}$$

To solve this simply make the substitution

$$p_i(p_0') = p_i(p_0) + \frac{\partial p_i(p_0)}{\partial p} \Delta p_0 = p_i(p_0) + T_{i0} \Delta p_0, \tag{12}$$

where p_0' are the improved starting parameters for the next iteration calculated from the previous starting parameters p_0, and Δp_0 are the changes in the starting parameters to improve the track. This equation can be plugged into the above if one makes the assumption that $\frac{\partial p_i(p)}{\partial p} = T_{i0}$ does not change much from one iteration the next, which follows from the inherent nature of making small adjustments to the track in order to improve it.

Thus one arrives at

$$\Delta p_0 = \sigma_{p_0} \sum_{i=1}^{N} T_{i0}^T (\sigma_i)^{-1} (x_i - p_i(p_0)), \tag{13}$$

where

$$\sigma_{p_0} = [\sum_{i=1}^{N} T_{i0}^T (\sigma_i)^{-1}) T_{i0}]^{-1}, \tag{14}$$

is the 5x5 covariance matrix of fitted parameters on the starting plane, whose diagonals describe the errors in the 5 track parameters on that plane and in the region close to it. (The fit does not directly return fit errors for track parameters on other planes.)

However, since random processes should not be neglected for optimal tracking results, it makes more sense to return to the original χ^2 in equation 1, only now the included matrix and vector objects are combined into one large linear algebra equation. Instead of a sum over N 5x1 objects multiplying 5x5 error matrices, the vectors are combined into a single 5Nx1 vector multiplying a single 5Nx5N matrix. The 5x5 diagonals of this large error matrix should now include the effects due to material processes as calculated in Geant from equation 3 as well as the measurement errors.

Because now parameters at one plane are no longer independent of the parameters at other planes, due to correlations from these random processes, it's necessary to add off-diagonal elements into the large error matrix. These 5x5 blocks come from

$$\sigma_{MN} = T_{MN}\sigma_N, \tag{15}$$

for the top diagonals, and the transpose for the bottom diagaonals, where M and N are two separate planes within the detector. (σ_N is the error matrix on plane N calculated from the starting plane.) This follows from equation 3 evaluated at plane M with respect to a path length from plane N, which is equivalent to 15.

You can then minimize the χ^2 in the same way, only again with the matrix objects being aggregates of the per plane objects:

$$\Delta\vec{p_0} = \sigma_{p_0}\tau^T(\sigma)^{-1}(\vec{p_0} - \vec{x}), \tag{16}$$

$$\sigma_{p_0} = [\tau^T(\sigma)^{-1})\tau]^{-1}, \tag{17}$$

where τ is the combined transport matrices from the individual 5x5 matrices, a 5Nx5 object.

The unmeasured parameter errors of infinity still come into play in the final calculation in the same was as before. Because however these matrix objects are very large, and the tracking must have a certain amount of speed in order to keep up with data, it is useful to reduce the size of these matrices. (It also makes things easier programming wise. Note that there are other some other ways to speed things up, specifically the banded inversion method as detailed in reference blah. This method was not used in favor of getting the code working in the simpler form in the first place, but it is a possiblity in the future to use this technique to speed things up even more.) It suffices to simply remove all rows and columns where said infinity values exist in the error matrix. This is mathematically equivalent to inverting the error matrix with the infinities included, which make all rows and columns where they exist go to zero. The associated unmeasured parameter rows in the residual vector and transport matrices must similarly be removed. This results in an Nx1 residual vector, NxN error matrix, 5xN combined transport matrix transpose, which multiply against the 5x5 covariance out front to still result in a 5x1 fix to the starting parameters, and a scalar χ^2 value. (Note that these element removals should be done just before the final calculation, and not higher up in the algrebra, otherwise plane correlations are not properly calculated.)

By calculating the last two equations one can fit the track, acquire a χ^2 describing the degree of the fit, determine how the track parameters can be improved at the starting point, and calculate errors on those starting parameters. This algorithm can be iterated a number of times to get a best fit track until successive iterations produce no improvement, where usually 3 or 4 iterations is enough. Note that there is remarkable robustness with respect to the initial starting parameters in fitting the track. Of course if the initial starting paramaters are too poor, then the fit will not converge.

3 In the gm2 Framework

3.1 Event Generation, Geometry, and Material

While not a direct part of the reconstruction and fitting code, there is relevant information with respect to event generation in the simulation upon which the reconstruction acts. Event generation before fitting is done using the mdc fcl files in gm2ringsim using the main simulation, where one should also include the tracker dummy plane geometry for truth comparison. These are built in the TrackerDummyPlane_service and associated geometry files. At time of writing there are some modified mdc#-geane fcl files which can readily be used. Only the necessary straw geometry and associated mother volumes, as well as the parallel world dummy planes are necessary to fit tracks, though including other geometry can of course change particle behaviour. All 3 main trackers are included in their original positions, except possibly for the upcoming June commissioning run where only the 18th tracker should be included.

Fcl parameters exist for the StrawTrackerCadMesh_service and Straws_service in order to turn material on or off at will, "materialTracker" and "strawMaterial" respectively. The rest of the geometry has to be manually changed and rebuilt in order to remove material in case one wants to turn off more material. These include VacuumChamberCadMesh and the World, as well as the "buildSupportPost" option in strawtracker.fcl and strawtracker_standAlone-staggered.fcl, "buildTrolly" in vac.fcl (where the associated material is hardcoded in), and "trolleySupportMaterial" also in vac.fcl. One should make sure to perform reconstruction with the same fcl parameters as were used in the event generation for proper results. (These options are purely for debugging the tracking. Close to perfect results have been shown for tracking within a vacuum world, DocDB 4876 and DocDB 4894.)

3.2 Reconstruction Flow

The overall tracking infrastructure and reconstruction flow can be seen in Figure 5. Data comes from simulation or the real experiment, are turned into art objects upon which the gm2 framework acts, and are grouped based on their positions in space and time spread. The RunGeane.fcl file (detailed more below) performs the entire chain within the blue box in Figure 5, excepting the track extrapolation stage. Within the reconstruction flow objects called Track Candidates are produced, which are the input to the Geane fitting code. The Geane fitting specific flow can be seen in Figure 6.

The primary code for the Geane fitting are the GeaneReco_module.cc and GeaneFitter.cc files. Detailed information regarding all of these files is given below, but here is provided a shorter summary. Track candidates drop into the GeaneReco file which initially sets up both the Geant4 world and the error_propagation routines (once), creates a GEANEArtRecord, and fills relevant track fitting objects. GeaneReco then tracks particles along their average trajectories in the Geant4 world, producing the matrix objects necessary as described in the Formalism section. All of these objects are stored within the GEANEArtRecord, which is then passed to the GeaneFitter class, which contains all the methods and math needed for fitting the track again described in the Formalism section, after which a fix to the track is provided along with higher level track objects. GeaneReco and GeaneFitter then loop until the track fitting either succeeds or fails, after which GeaneReco produces TrackArtRecords and TrackStateArtRecords, as well as a final updated GEANEArtRecord for the track. These TrackArtRecords can then pass to the extrapolation stage or go into some refinement stage (which has not been developed yet). The produced GEANEArtRecords can then be passed through filters and analyzers depending on what the user is investigating or looking at (again detailed below).

3.3 fcl File Specifics

Again, reconstruction is performed using RunGeane.fcl in gm2tracker on the events generated from somewhere higher in the chain. It is important to explain some fcl parameters necessary for the reconstruction. First, there is a fcl parameter useSD to turn off the straw sensitive detectors so that in the reconstruction phase hits are not regenerated. If this is not included, it will default to true and cause crashes in the reconstruction. Secondly, the RunGeane.fcl file loads all 3 trackers necessary for symbol and name definitions, but such that tracker 0 is rotated such that the tracking planes are parallel to the global geant X axis (using the rotateArcTracker fcl parameter for the ArcService). This is done to avoid the issue of error propagation instability close to the Z axis, as detailed in DocDB 4567 while at the same time still observing the correct azimuthally symmetric 2D field for the tracks. Track hits are rotated from their separate trackers to this one tracker frame with the geaneWorld0, geaneWorld12, and geaneWorld18 transforms defined at the bottom of StrawTrackerCadMesh_service.cc, as detailed in the Coordinate Systems section. (In the future, if fields or geometries are not identical between trackers, the details here will have to be improved.) As a reminder there are material fcl parameters for the straws and straw trackers which should be the same as were used in the event generation. The RunGeane.fcl file also includes some analyzer and filter modules by default, which are detailed below.

go into digitalizer/smearing fcl parameters (use true t0 and stuff) here to really describe this RunGeane.fcl

Figure 5: Shown is the infrastructure flow for the entire track reconstruction chain. In the green blocks are the sources of track data to fit, either from simulation from Geant, or real data from Midas. In blue is the offline reconstruction block. Straw digits are formed in the digitalization step, those are then calibrated and grouped into time islands, clusters, and seeds which then combine to form track candidates. It is these track candidates which are the input into the Geane fitting code (and other future fitting code). The fitting code then outputs tracks which the track extrapolation stage will run over. This document focuses on the Track Fitting stage and will not go into detail on the other parts of the chain. This picture is taken from one of Tammy's talks. Note that there is some iteration here that is not shown.

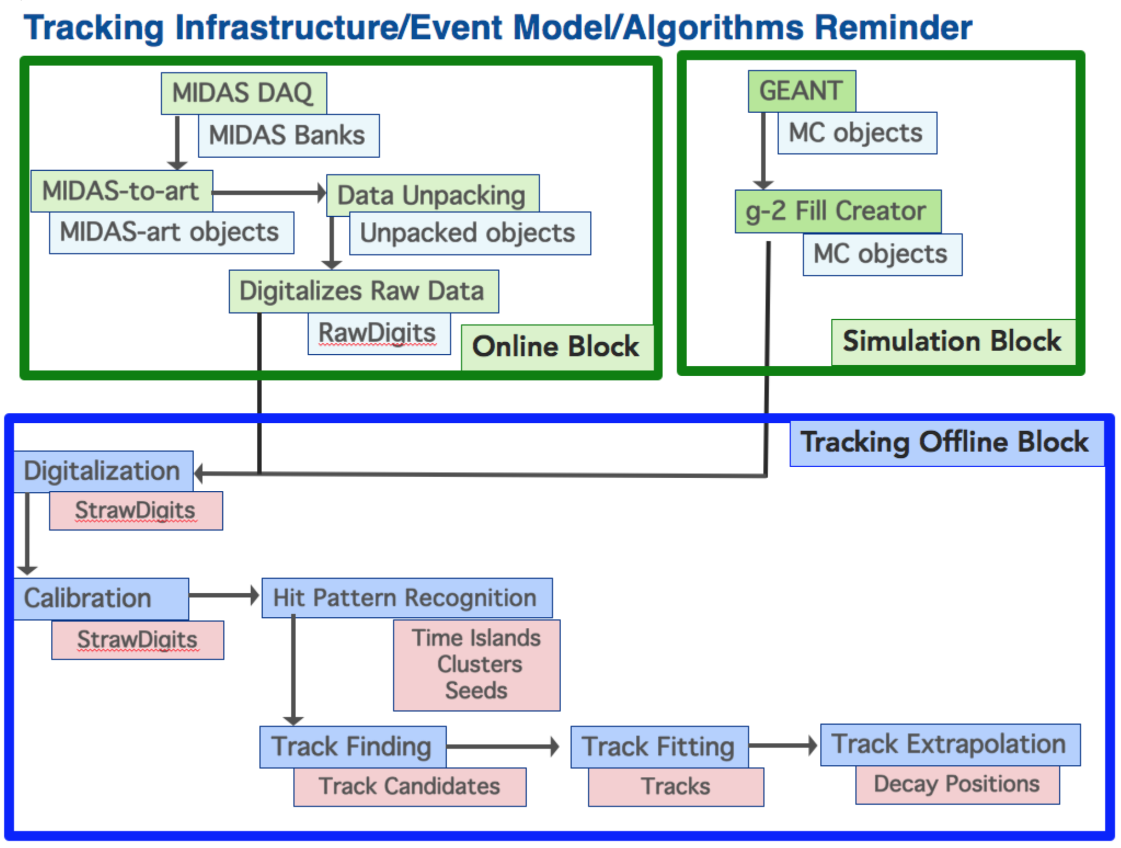
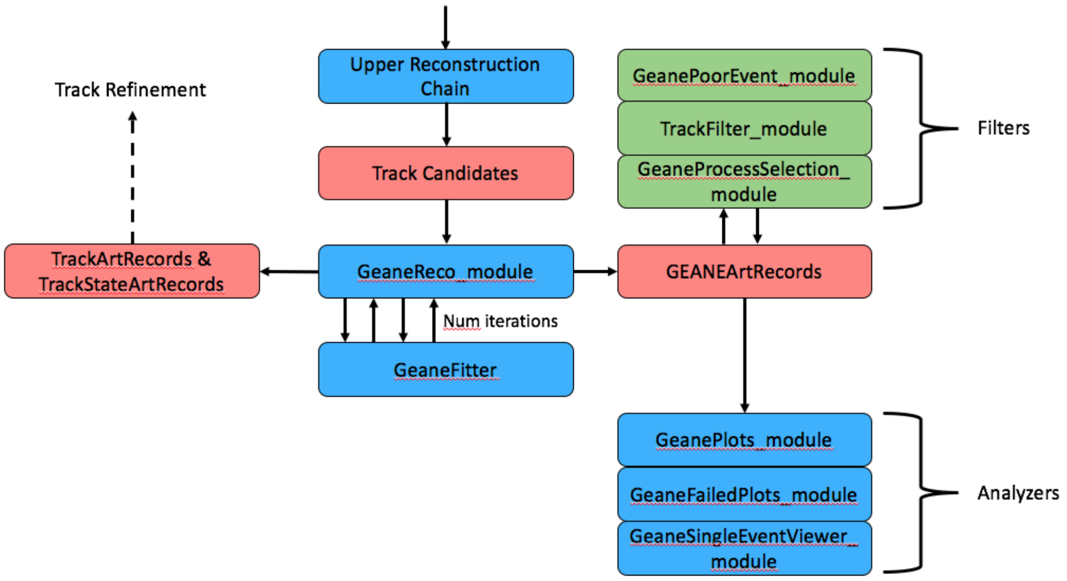


Figure 6: Shown is the Geane fitting code flow. See the text for a thorough explanation of this flow, and the specifics of each box.



file, try and not leave anything out that's important - track candidates, circle fitter, etc. - temp aside before Joe's stuff is done

then detail the geaneFitParams.fcl file for the geane fitting specific parameters - create an itemized list or table possibly about these parameters - while referencing the fact that some of these parameters might disappear or change - The reconstruction code links to geaneFitParams.fcl in gm2tracker which has many variables for changing momentum spreads, straw measurement errors, random seeds, toggling truth info, etc. Things get more complicated here so ask me before trying to play with this stuff.

3.4 GEANE TrackFitting - change this title

update this - The GEANE track fitting routines consists of a couple of main files within gm2tracker. Track candidates from upstream in the reconstruction chain are the input to this code. A single GEANEArtRecord is produced for each event or track, and is passed from one method to another, constantly being updated as the track fitting iterates along. See below for a quick summary of the method flow, and a table for the data objects within GEANEArtRecord.

1. **GeaneReco_module.cc**

   This file consists of the main code which sets up the GEANE track fitting, pulls in upstream data products, interfaces with both Geant4 and the gm2 art framework, and outputs TrackStateArtRecords and TrackArtRecords for use downstream.

   The methods errorPropagation, angleCorrection, and TrackCorrelation (in GEANEFitter) are called iteratively within a loop until track fitting succeeds or fails ... update this

   It's methods consists of:

   - produce
     General overriden produce method. Calls main track fitting code on an event by event basis. Produces data products for downstream use.
   - InitializeGEANE
     Geant4 GEANE initialization method called by the constructor, once per run.

- trackFitting
  Sets up variables and objects for track fitting after reading in upstream data products (track candidates and dummy planes). Contains main track fitting iteration loops. Also contains complex L/R sequence checking routines which can be ignored for now (and probably need to be changed at some point).
- modifyMeasuredParams
  Short method for L/R sequence checking stuff, can be ignored for now. Just modifies measured parameters based on particular sequence
- errorPropagation
  This method tracks particles through the detector using Geant4 error propagation routines with the correct geometry and field. It builds transport matrices, error matrices, and predicted parameters which are the objects used for fitting the track. It tracks on a plane by plane and step by step basis. These routines can be used to track particles forwards and backwards, where the forwards tracking is used in the code. Changing to backwards tracking would be very non-trivial. Add more detail here later..
- angleCorrection
  Method to iteratively correct measured parameters from a radial DCA value to a U or V value based on the momentum of the track and approximating a constant field within the straw. Also corrects the errors using a simple straight line approximation which is good enough.

2. **GeaneFitter.cc**
   This file consists of the main track fitting chi2 algorithm. It multiplies measured parameters, predicted parameters, error matrices, and transport matrices together to produce a chi2 for the track and an improvement to the starting paramters. It also reads in the measured hits errors in order to properly fit the track. There is a fcl parameter matrixDebug which can be used to turn on or off the many large matrix cout debugging statements.
   - TrackCorrelation
     The main matrix multiplication routines.
   - preSequenceChecking
     A method in order to create a hybrid wire / U or V error matrix for L/R sequence checking. Only called once per event.
   - sequenceChecking
     The main L/R sequence checking method for an individual event. This method is called many thousands of times as each U or V sequence is checked. Ignore for now.
   - convertTo
     Some methods for converting Eigen 5 vectors from GeV cm to MeV mm and vice versa.
3. **GeanePlots_module.cc**
   The main analyzer/plotting module which runs on the output GEANEArtRecords produced upstream. This creates many plots including chi2 distributions, p value distributions, number of iterations, track parameter, residuals, pulls, etc. Truth information for these plots is currently necessary as written, but can be take out in the future for real data. There are some fcl paramters to make cuts on different parameters. There are also some other plotter modules which are similar but much reduced in scope.
4. **GEANESingleEventViewer_module.cc**
5. **GEANEProcessSelection_module.cc**
6. **GeaneFailedPlots_module.cc**

7. **GeanePoorEvent_module.cc**

8. **refit geane stuff?**

Note: Ignore copious LR ambiguity solving code for now. When last checked it was working pretty well but not perfectly, and it needs to be studied more. Many changes have been made to other code which will probably break the interfaces to the LR code as well. Also all of this LR stuff might be pulled out and placed into other tracking code modules before and after the GEANE code, but that is unclear.

Table 1: GEANEArtRecord.hh active variables. This is subject to change. GEANEArtRecord contains vectors of variables on planes, as well as larger objects containing information about the whole track. Note that Eigen matrix objects cannot be stored into art data products. For this reason, and for minimal code changes, it was decided to add a data object for each Eigen member, made up of vectors with the word "Data" tagged at the end. The data objects are saved when creating GEANEArtRecords using a utils file. In analyzers accessing the GEANEArtRecords, these data objects are then swapped into the Eigen objects using the same utils file.

art::Ptr < gm2strawtracker::TrackCandidateArtRecord > candidate
One track corresponding to one candidate corresponding to one GEANEArtRecord for the whole track.
std::vector< art::Ptr< gm2truth::GhostDetectorArtRecord > > dummyPlaneHits
Associated dummy plane hits on planes aligned with straw wires, vector consists of hit dummy planes corresponding to hit wire planes (if a straw plane was skipped but that dummy plane was hit, it is not included in this vector. Vector has size N = num hits in straws that form the track +1 for the 0 plane.)
int failureMode
Different failure modes for failed track reconstruction, 0 means it passed.
double chi2
Chi2 for whole track.
std::vector<double> chi2Iterations
Chi2s for whole track for different iterations.
std::vector<double> chi2Planes
Individual chi2s on each plane which add up to total chi2, vector consists of hit planes with size N.
int numIterations
Number of iterations to converge.
unsigned int dof
DoF of track = number of hit planes - 5 track parameters.
double chi2DoF
chi2/dof
double pValue
Fit pValue for whole track.
double energyDiff
energy loss between first and last hit in track - from truth, for material characterizing
std::vector<double> startingTrackParameters
Starting parameters for track: size 6, 3 position then 3 momentum, x y z px py pz, best starting parameters updated after each iteration, starting parameters x position defined before first hit.
std::vector<double> startingTrackGuessOffsets
Size 10, x y z px py pz p 1/p pu/px pv/px offsets in different starting track parameters for plotting purposes.
int trackNumPlanesHit
Total number of planes hit.

int trackFirstPlaneHit
int trackLastPlaneHit
std::vector<int> trackPlanesHitList *List of hit planes, with missed planes excluded from the vector. Ex. 1 2 4 5 8 9*
Sequence information excluded from this table for now.
std::vector<std::vector<double> > wireUVPositions *Wire center U and V postions, first vector is track param vector size 5 (0 1 2 unfilled, 3 is U, 4 is V), second vector is planenumber from 0 - 32 (formatted this way to align with other similar vectors - can probably be reduced.)*
std::vector<double> measuredDCAs *Vector of measured DCAs for hit planes, with size 33. Mainly to hold on to smearing values for now.*
std::vector<double> UVerrors *Vector of UV measurement errors for planes 0-32. Built in order to accomadate varying errors in the future.*
std::vector<double> planeXPositions *Vector of X postions of hit wire planes with size 33 (0 - 32), 0 plane being in front of the first module that was hit.*
std::vector<std::vector<double> > geaneMeasuredParameters *Measured GEANE parameters, first vector is param num 0 - 4, second vector is plane number 0 - 32, units are MeV mm. 1/P, Pu/Pz, Pv/Pz, U, V - only U or V is filled at the start of the GEANE fitting module.*
std::vector<std::vector<double> > geanePredictedParameters *Predicted GEANE parameters, first vector is param num 0 - 4, second vector is plane number 0 - 32, units are MeV mm. 1/P, Py/Px, Pz/Px, Y, Z - all params filled in tracing stage of GEANE fitting module - coord system has to be orthogonal - converted to UV locally in the fitting module.*
std::vector<Eigen::MatrixXd> geaneTransportMatrices std::vector<std::vector<double> > geaneTransportMatricesData *Units of GeV cm - transport matrices between planes tracked to in GEANE fitting module, 5x5 objects. Vector has size 33.*
std::vector<Eigen::MatrixXd> geaneErrorMatrices std::vector<std::vector<double> > geaneErrorMatricesData *Error matrices on tracked to planes, units GeV cm, size 33.*
Eigen::MatrixXd covarianceTotalInverse std::vector<double> covarianceTotalInverseData *5x5 inverse of total covariance matrix for track. Diagonals represent errors in 5 track paramaters on plane 0.*
std::vector<Eigen::VectorXd> paramPredictedInUVEigen std::vector<std::vector<double> > paramPredictedInUVEigenData *Predicted parameters in UV space as an eigen object (converted from geanePredictedParameters above) for calculation convenience and some LR information storage. Order of vectors is switched here, first is planenum, second is paramnum, units are MeV mm.*
Objects below here are full track objects with larger sizes, held on to for fast sequence checking. Units GeV cm.
std::vector<Eigen::MatrixXd> extendedTransportMatrixBegToEnd std::vector<std::vector<double> > extendedTransportMatrixBegToEndData *Accumulated/combined transport matrices from starting plane to all following planes.*
Eigen::MatrixXd extendedCombinedTransportMatricesTranspose std::vector<double> extendedCombinedTransportMatricesTransposeData *Transpose of larger eigen object composed of above begtoend transport matrices.*
Eigen::MatrixXd extendedReducedMatrix std::vector<double> extendedReducedMatrixData *Total error correlation matrix, reduced to size NxN (N = num planes hit).*
Eigen::MatrixXd extendedReducedMatrixInverse

std::vector<double> extendedReducedMatrixInverseData *Inverse of above saved once for LR checking.*
Eigen::MatrixXd extendedModifiedReducedMatrix std::vector<double> extendedModifiedReducedMatrixData *Reduced matrix from above that's going to be modified into a hybrid error matrix separately for U and V fits but that needs to be held onto for all sequences.*

3.5 Code details

Building and multipilying transport and SC2SD matrices Modifying measured parameters based on momentum angle/field and previous predicted Converting things to eigen objects

JacobianToUV stuff - or combine with other coordinate transform stuff Matrix accumulation/reduction Notes on loop indices, start points and such

3.6 Angular correction

Because our straws don't in fact measure U and V coordinates directly, but instead distance of closest approach radii deriving from measured hit times, in order to utilize the minimization procedure on measured track parameters these radii must first be converted to U and V parameters. (And similarly for the U and V errors.) (Note that the future Kalman filter will not be subject to this disadvantage.) This is done in the angularCorrection method at the bottom of the GeaneReco module. The wires that are hit as well as the initial raw measured errors are stored indefinitely, and it is upon these objects that corrections are made. These corrections will be dependent on the angle of the track, so it's imporant to note that during each successive iteration, the "measured" parameters are adjusted by the latest "predicted" momenta. It was found that for the error correction, a simple straight line correction was sufficient for ideal results. For the position correction, it was found that a constant field correction for curved tracks was sufficient.

To calculate these corrections, first the momentum perpendicular to the straw measurement axis can be ignored. Because the positron tracks curve in only one direction through the tracker, one needs to calculate the correction depending on whether the track went to the left or right side of the wire. See Figure 7 for a pictorial representation of the problem. The calculation of the right side correction follows, with the left side correction being calculated in a similar manner.

An equation with the single unknown parameter being of interest can by approximated by

$$(r+d)^2 = r^2 + u^2 - 2ru\cos(90+\theta),$$

where the 90 degrees is approximate for large curvature of tracks , and we want to solve for u. The angle θ can be determined from

$$\hat{z_{\|}} \cdot \hat{p_{\|}} = \cos\theta, \theta = \cos^{-1}\frac{p_{\|}}{p},$$

where p parallel is the momentum parallel to the z parallel axis at the wire plane and can be determined within the code. Using a trig identity and solving for u gives

$$u = -r\sqrt{1-(\frac{p_{\|}}{p})^2} + \sqrt{d^2 + 2dr + r^2(1-(\frac{p_{\|}}{p})^2)},$$

for the right side correction, and

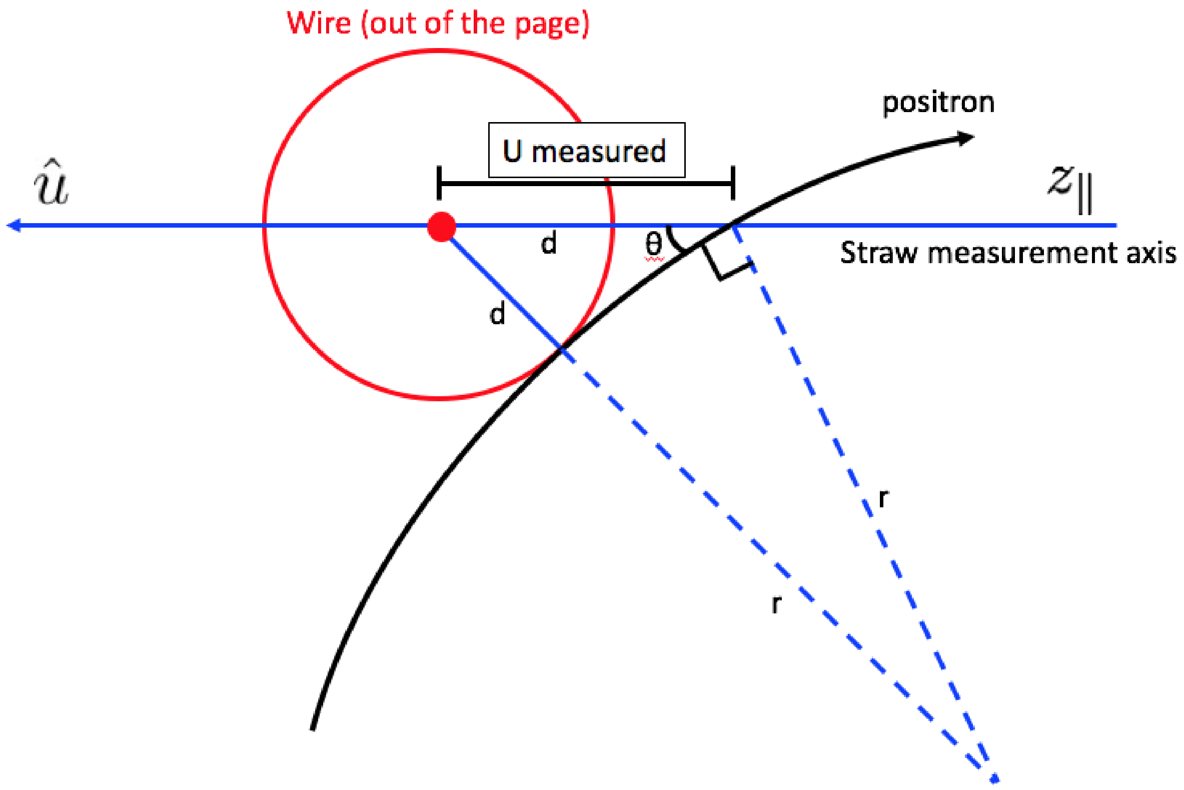
$$u = +r\sqrt{1-(\frac{p_{\|}}{p})^2} - \sqrt{d^2 + 2dr + r^2(1-(\frac{p_{\|}}{p})^2)},$$

for the left side correction. (Corrections to v are identical.) The radius of the particle circle can be calculated from the circular momentum and magnetic field at the predicted hit position. The straightline correction is done simply using the Pythagorean theorem in a simpler manner, with the correction to the errors then being

$$\sigma'_{uv} = \frac{\sigma_{uv}}{\sqrt{1-(\frac{p_{\|}}{p})^2}}$$

.

Figure 7: Shown here is a positron passing through a straw. The desire is to convert the measured parameter d into a U or V position, which can be done by approximating the particle trajectory as a circle in a constant magnetic field over the course of the straw and using trigonometry. Sizes and angles are exaggerated.



3.7 Code peculiarities / notes

Where to structure this in the file?

Notes on coordinate systems I use - UV pointing outward vs XYZ, XUV x being forward, UVW in geante src, maybe IJK/TUV in one paper-prob not, GeaneTrackerWorld[0,12,18], rotateArcTracker (bit more specific), magnetic field access coordinates?, JacobianToUV

accessing the field in a specific way

4 gm2Geane package in artg4 - eloss, brems, ionizations, msc

See many docdbs by Nick Kinnaird on geane updates for more info if wanted. Summary of stuff here.

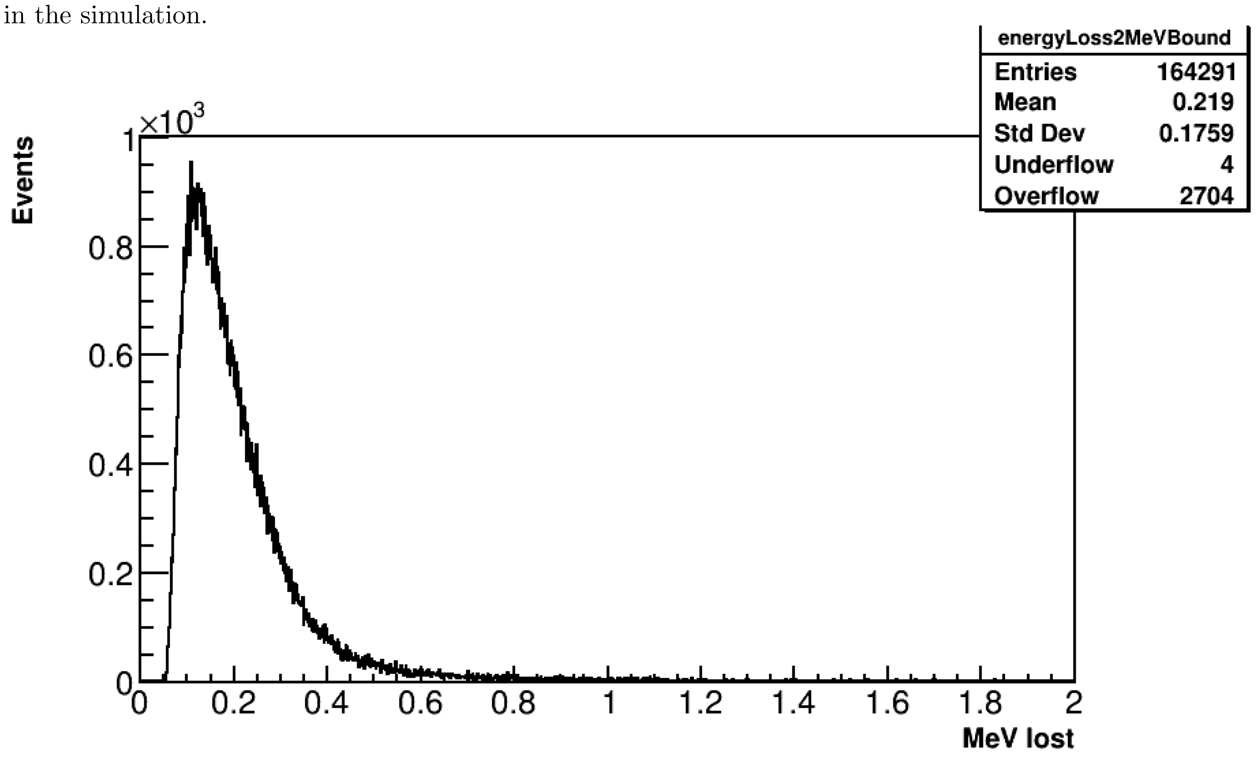
There is a folder under artg4/gm2Geane, where slightly (but importantly) modified geant/geane code that the geane track fitting uses is located. Most importantly is that the reconstruction was taking too much energy away from the particle during propagation due to bremsstrahlung, so that has been removed. (Thanks James for help with this.) (Probably explain flow of file - extrapolator tables etc.).

Besides those associated files, all files from the geant folder "error_propagation" have been copied into this directory as well, with the only non-name changes lying in gm2GeaneFreeTrajState.cc. (Many files had to be copied due to naming/linking - some could probably have been omitted but I decided to copy the whole folder.) Ionization and multiple scattering errors have been modified according to Lavezzi thesis (explain more) - with no improvement in results unfortunately. (Probably turn defaults back to original code (still gm2Geane) with the option to turn things on somehow if desired - at least leave them in for future tuning.)

Should I go more in depth about the error_propagation/gm2Geane source code? Probably - at least for certain main files.

Add many plots and pictures of things - pulls, single events, tracks, etc etc.

Figure 8: Shown here is the energy loss between the first and last hit in the tracker from simulation. (Momentum magnitude difference.) Sources of energy loss come from ionization and bremsstrahlung processes, which account for the long Landau tail running off to infinity. The distribution has a mean of approximately 220 keV and and peak centered at about 150 keV, which is reasonable for the material composition of our trackers and tracker gas. More than 50% of the energy loss comes from the mylar walls of the straws as seen in the simulation.



5 Coordinate Systems

The only material not covered here in full detail is the mathematical explanation of reference frames, for which one should see the reference papers. It suffices to summarize as follows: GEANE objects (matrices and parameter vectors) are defined and calculated in the Geant4 source code in the free particle system. Then there is an intermediate surface system defined in XYZ, as the Geant4 surface trajectory system must be defined in an orthogonal coordinate system, before which parameter objects are converted to the most natural detector system of XUV. (Important note - be very careful with coordinate system variables, letters are reused between different papers and code bases with different configurations and meanings constantly.)

$1/p, \lambda, \phi, y_\perp, z_\perp$, free system

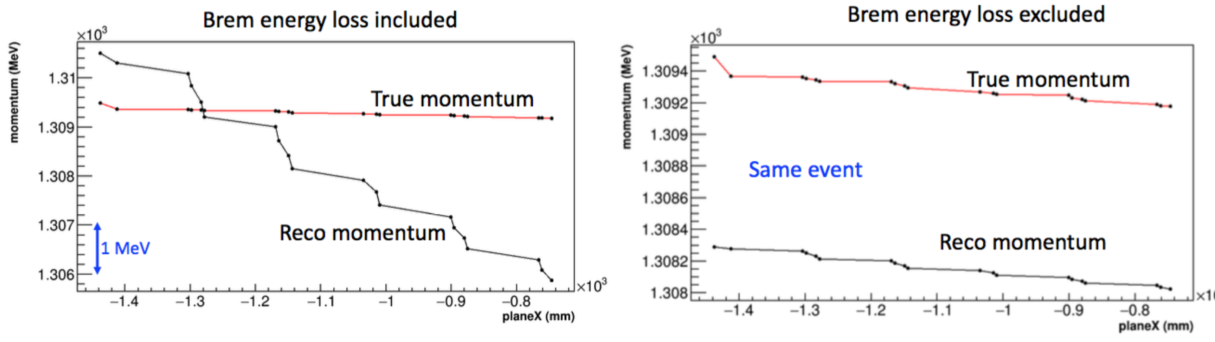
$1/p, py/px, pz/px, y, z$, surface system

$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sin\theta & -\cos\theta \\ \sin\theta & -\cos\theta \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix}$, yz to uv matrix, where θ is 7.5°, the 5x5 transformation is just a 1 in the top left corner, and then this matrix in the remaining 2 diagonal blocks

$1/p, pu/px, pv/px, u, v$, uv system

[4], [5]

Figure 9: It was noticed during the course of debugging that the Geant4 error_propagation routines were consistently removing too much energy on average from all tracks as they passed through the tracker during reconstruction. It was discovered that the Geant4 tables were taking out too much energy due to bremsstrahlung processes and it was decided to remove this effect. The left plot shows in a red the true particle momentum as a function of X distance through the tracker, and in black the reconstructed momentum before bremsstrahlung was taken out of the reconstruction for a single event. The differing slopes signify the problem. The right plot shows the same event but with bremsstrahlung taken out. Notice scale change, and that the reconstructed momentum aligns much more readily with the truth.



References

[1] E. Nagy V. Innocente M. Maire. *GEANE: Average Tracking and Error Propagation Package*. January 13, 1994.

[2] E. Nagy V. Innocente. "Trajectory fit in presence of dense materials". In: *Nuclear Instruments and Methods In Physics Research* A.324 (Aug. 1993), pp. 297–306.

[3] Lia Lavezzi. "The fit of nuclear tracks in high precision spectroscopy experiments". pp. 57-86. PhD thesis. University of Pavia, Nov. 2007.

[4] W. Wittek A. Strandlie. "Derivation of Jacobians for the propagation of covariance matrices of track parameters in homogeneous magnetic fields". In: *Nuclear Instruments and Methods In Physics Research* A.566 (July 2006), pp. 687–698.

[5] László Urbán Kati Lassila-Perini. "Energy loss in thin layers in GEANT". In: *Nuclear Instruments and Methods In Physics Research* A.362 (Dec. 1994), pp. 416–422.

6 notes and other

Due to the CBO of the beam, 1 tracker accounts for 90% of the physics, and a second tracker accounts for much of the remaining. (Find/learn sources for this to explain a bit better..)

starting error matrix

linking equations to code variables

initial errors into tracing/error propagation?

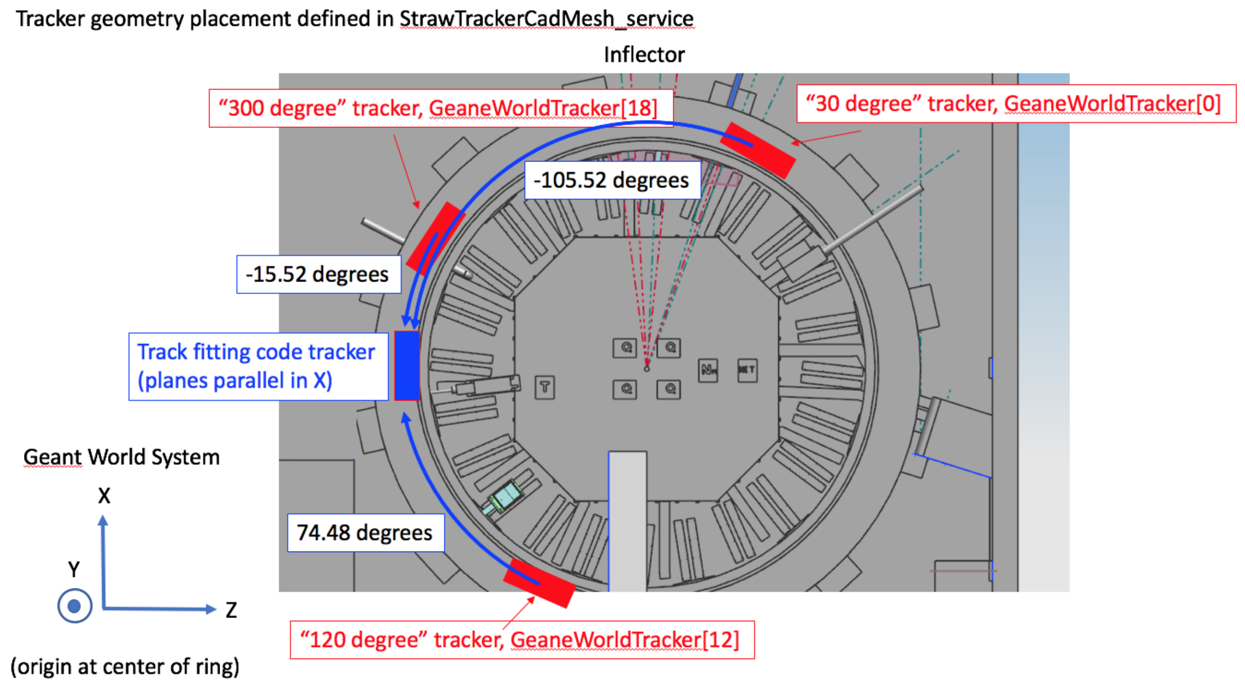
mention that it's possible to make a Kalman filter which uses the geane matrix objects? and that I didn't do so because we wanted two separate methods for the tracking originally and someone else was working on the kalman filter? (and the chi2 minimization was used in the last experiment)

make a subsection on the left-right ambiguity work

negative chi2 stuff

note on older branches for development at all?

Figure 10: Shown here is a picture of the 3 trackers in the world geometry in their approximate positions. First note the world coordinate system shown in the bottom left of the plot, where the origin lives at the center of the ring. Tracks are then generated and read out from the trackers in their three world positions in red. Due to the reconstruction bug where tracks improperly reconstruct if their momenta is too aligned with the globabl Z axis, the reconstruction rotates the entire arc including tracker 0 to the blue position, where planes are parallel in X, and this problem is avoided. Track parameters from the 3 positions are then rotated to this reference frame by the amounts shown on the plots for the reconstruction stage, and at the end are rotated back.


Tracker geometry placement defined in StrawTrackerCadMesh_service
Inflector
"300 degree" tracker, GeaneWorldTracker[18]
"30 degree" tracker, GeaneWorldTracker[0]
-105.52 degrees
-15.52 degrees
Track fitting code tracker (planes parallel in X)
Geant World System
X
Y
Z
(origin at center of ring)
74.48 degrees
"120 degree" tracker, GeaneWorldTracker[12]

6.1 Kalman Filter

There is some precedent for using the error propagation matrix objects within a Kalman filter as detailed in the Lavezzi thesis [3], but due to the circumstances following the creation of this code that technique was not followed. At the beginning of development, a Kalman filter was being started in tandem by others, which has since fallen by the wayside. In order to have two separate methods for fitting however, the least squares global minimization was chosen. The simpler method was also chosen as the author was unfamiliar with track fitting and coding in general when beginning development.

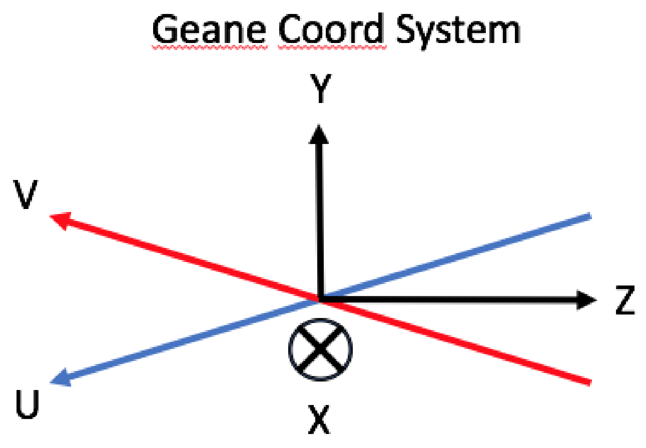
7 Acknowledgements

Should I put an acknowledments section in? (Rob, Rob, James, Joe) From a technical document perspective maybe not?

8 Plots

It'd probably be good to make one big set of plots to toss into the back for people to look at/reference. All sorts of pulls, parameter plots, residuals, etc. Sort of like Renee's verification pdf.

Figure 11: This picture shows the coordinate system in which the the Geane track reconstruction is performed, in relation to the world coordinate system. The origin remains at the center of the ring, with the tracking planes parallel in X in the reconstruction, going forward in number. Y is vertically up, and Z is horizontally to the right. U and V are defined such that they have greater values with higher radii and increasing straw number.



Geane Coord System
Y
V
Z
U
X