Weekly Report #2
June 13 - June 20

Christina

Some Clarifications on the Data Generation Process

1. sim.make_event(x0,y0,z0,E0,Azi0,Pol0)
    - Creates charge vs. time information for each pads (10240) through “time buckets”
    - The peak charge corresponds to the time when the pad receives signal
2. sim.convert_event() converts each event to single event objects with x, y, z, a, and pad
    - The position of activated pad is converted into xy coordinate
    - Z coordinate is transformed from time bucket to distance
    - a is the charge signal at its peak
    - Pad is the pas number

Monte-Carlo Example (run_0102_clean)

#1. Real Data Fitting

Fitting: https://github.com/chchen123/Monte-Carlo/blob/master/Run_0102_clean_real_vs_sim.ipynb

1. Read files from config_e15503a data & input configs into MCFitter class
2. Read real event information (in this case run_0102_clean.h5)
3. Get a single event object named xyzs (with x, y, z, a, and pad) for a self-defined event ID
4. Use cleaning.hough_circle to find the center of curvature of the track
5. Use MCFitter.preprocess() to rotate, calibrate, and transform the coordinates from xyz (detector coordinate system) to uvw coordinates (beam coordinate system).
    a. Returns a pd file with xyz coordinates, uvw coordinates (used for plotting), a, and pad - all are used for subsequent fitting function process_event
    b. Returns calibrated center of curvature cu and cv

Fitting:

6. Use mcfitter.process_event to get the six track parameters (x0,y0,z0,E0,Azi0,Pol0), the minimum total chi2 value for each iteration, the parameters from all generated tracks, and the row numbers in ``all_params`` corresponding to the best points from each iteration

Analysis:

1. Plotted each parameter's value vs. the number of iteration. x and y values always approach to zero while the others do not have convergence.

#2. Data Simulation & Simulated Data Fitting

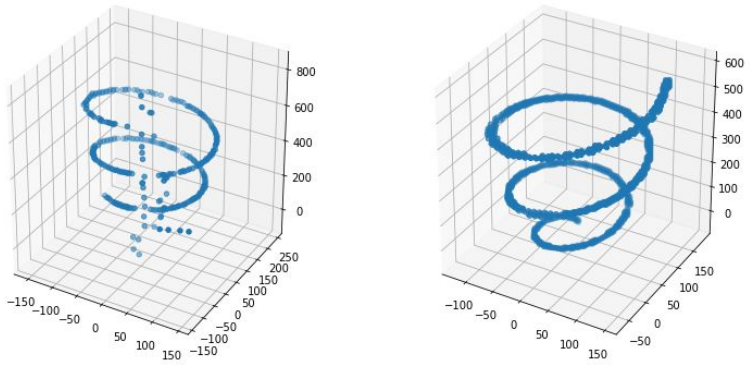
Simulation:

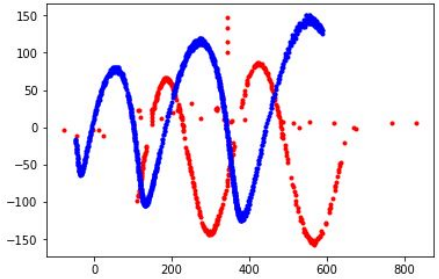
1. Input configs into EventSimulator class as sim
2. Use sim.make_event to generate individual event, using the results from Monte-Carlo fitting as input parameters
3. Use sim.convert_event to get single event objects pyevtClean; pyevtClean.xyzs to get an array with the simulated data's position in xyz coordinates
4. Use MCFitter.preprocess() to rotate, calibrate, and transform the coordinates from xyz (detector coordinate system) to uvw coordinates (beam coordinate system).
    a. Returns a pd file with xyz coordinates, uvw coordinates (used for plotting), a, and pad - all are used for subsequent fitting function process_event
    b. Returns calibrated center of curvature cu and cv

Plotting preprocessed real and simulated data together:

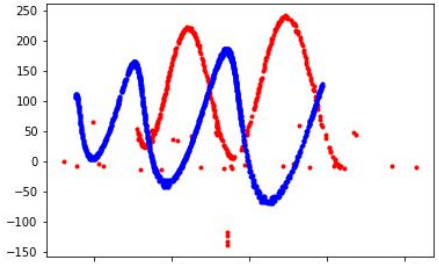
Event ID = 305

Red: real data; blue: simulated data

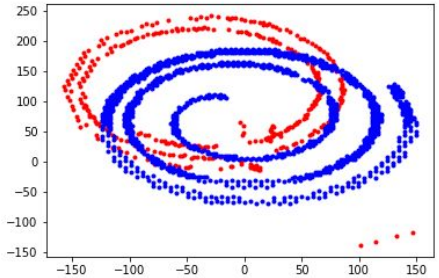




x vs. z



y vs. z



x vs. y

Fitting:

1. Use mcfitter.process_event to get the desired initial conditions: used same center of curvature as real data

2. Compare:

```
xyz, (cu, cv) = mcfitter.preprocess(xyzs[:,0:5], center=(cx, cy)) # get calibrated set of data
xyz_values = xyz.values #transform pd file to arrays
print(cu,cv)
mcres, minChis, all_params, good_param_idx = mcfitter.process_event(xyz, cu, cv, return_details=True)

#print(mcres, minChis, all_params, good_param_idx)
print(mcres)
#print(xyz)
```

```
-43.91625428724363 118.39379347393248
{'x0': 0.0026984530041676974, 'y0': -0.012081132331789162, 'z0': 0.6574585621792225, 'enu0': 2.821749548027407, 'azi0': -
2.9218747202038373, 'pol0': 1.925839177321652, 'posChi2': 3.8071334199211773, 'enChi2': 7.451859815475861, 'vertChi2': 3.
064708140678062, 'lin_scat_ang': 1.2375421957837878, 'lin_beam_int': 645.0546221844862, 'lin_chi2': 30.05353534736301, 'r
ad_curv': 118.44677427333164, 'brho': 0.21057586294734618, 'curv_en': 2.1237324082771036, 'curv_ctr_x': -43.9162542872436
3, 'curv_ctr_y': 118.39379347393248}
```

```
new_mcres, new_minChis, new_all_params, new_good_param_idx = mcfitter.process_event(new_xyz, new_cu, new_cv, return_detail
print(new_mcres)
```

```
-43.91625428724363 118.39379347393248
{'x0': -0.0004261155034616931, 'y0': 0.0007696311846247996, 'z0': 0.5009119965013761, 'enu0': 2.841774077919338, 'azi0':
-3.0908379300858875, 'pol0': 2.2547213705799973, 'posChi2': 85.17346437303671, 'enChi2': 1.4915776293723737, 'vertChi2':
0.01547813165274769, 'lin_scat_ang': 1.0307051436719932, 'lin_beam_int': 451.0632202089078, 'lin_chi2': 56.73590113286050
5, 'rad_curv': 150.7402416354336, 'brho': 0.29527210696892203, 'curv_en': 4.175682431306552, 'curv_ctr_x': -43.9162542872
4363, 'curv_ctr_y': 118.39379347393248}
```

Other Methods - Gradient Descent

Finding the smallest chi-square values - taking small steps in the direction of the steepest gradient until reaching the smallest value

The objective function = chi-square-position + chi-square-energy + chi-square-vertex

http://scikit-learn.org/stable/modules/sgd.html#

https://www.scipy-lectures.org/advanced/mathematical_optimization/