ACD sensor response simulation

For technical reasons it is easiest to operate in terms of MIP equivalent units

For technical reasons it is easiest to operate in terms of MIP equivalent units as much as possible.

NOTE and WARNING:
Because of non-linearities, this is only self-consistent if you take a very specific meaning of MIP equivalent units. (meq)
Namely, the ratio between the light yeild observed and the most probable light yeild from a single MIP at normal incidence.
So, saying that a signal is 2 mips means that the light yield is twice the light yeild expected from a single MIP at normal incidence.

Some subtleties of this defintion:
1) Because of light output saturation heavy ions signals scale like
signal(meq) = Z^2 * 0.608 + 0.393 exp(-0.00483 * Z^2),
not
signal(meq) = Z^2.
Even though the energy deposited does go like Z^2.
2) Since large signal saturate the electronics, having twice as much light yeild doesn't imply that the measure PHA will
be twice as large
3) The number is defined against normal incidence tracks. Since we don't know the track angle apriori for any given ACD hit, we
can only correct the track incidenc angle only in certain cases when we can associate tracks with ACD hits.
4) Geometrical effects on the light attenuation are accounted for in the Monte Carlo _after_ the conversion to MIP equivalent units
and are not accounted for in the data at all. This is especially important for ribbons. The signal expressed in MIP equivalent units
for ribbons doesn't really mean much, especially if we don't know where along the ribbon the incident track struck.
5) For tiles it is easy to extract the PHA value corresponding to the most probable light yeild from a single MIP at normal incidence,
for ribbons this is a more complicated process because of the light attenuation along the ribbon and the large fraction of tracks that don't
traverse the entire ribbon but only graze the edges.

Furthermore, certain calibrations are very easy to preform/ or have already been done.
Therefore we should operate in terms of the output of those calibrations. Specifically:
1) pedestals (low and high range)
2a) MIP peaks for tiles
2b) PHA value for MIP at normal incidence at end of ribbon for ribbons
3) Crossover values in PHA between low and high range readout
4) veto threshold expressed in terms of PHA
5) cno threshold expressed in terms of PHA
6) slope(pha/meq) and saturation(pha) values for high range readout

Desciprtion of Monte Carlo signal simulation process.

1) We start with the energy deposited in the tile.
E_dep(MeV) from GEANT.
This includes Z^2 and pathlength effects, but does not include effects of light yeild

2) Convert from E_dep to photo-electrons(pe):
pe_nom(pe) = E_dep(MeV) * pe_per_MeV(pe/MeV)

To get pe_per_MeV we for tiles we need 3 pieces of information:
2a) pe_per_MeV for tiles comes from:

1.9 (MeV/meq)	# mev_per_mip = 1.9 (tiles)
pe_per_mip(pe/meq)	# calibration by PMT
saturation factor S(Z) for heavy ions	# voltz: S(Z) = 0.608 + 0.393 exp(-0.00483 * Z^2)
attenuation factor from geometry A(x)	# only applied close to tile edge

2b) pe_per_MeV for ribbons comes from:

0.45 (MeV/meq)	# mev_per_mip = 0.45 (ribbons)
pe_per_mip(pe/meq)	# calibration by PMT
saturation factor S(Z) for heavy ions	# voltz: S(Z) = 0.608 + 0.393 exp(-0.00483 * Z^2)
attenuation factor from geometry A(x)	# applied as a function of length along ribbon/ segement

Combine these data to get pe_per_MeV for the hit in question
2c) pe_per_MeV(pe/MeV) = A(x) * S(Z) * pe_per_mip / mev_per_mip

Calculate the nominal # of photo-electorns
2d) pe_nom(pe) = E_dep(MeV) * pe_per_MeV(pe/MeV)

3) Throw Poissons to simulate dynode chain
pe_obs(pe) = PoissonAndGain(pe_nom,gain=4,iter=5)
This means we those a Poisson about pe_nom, then scale that number by the gain=4,
and throw a Poisson about new number, repeat iter=5 times.

4) Convert the signal into MIP equivalent
signal(meq) = pe_obs(pe) / pe_per_mip(pe/meq)
Where pe_per_mip(pe/meq) is the same number as used in step 2a or 2b above.

5) Convert signal into PHA bins
5a) First we determine if this is read out in the low or high range. To do this we express the crossover point in terms of mips.
xover(meq) = (low_max(pha) - ped(pha)) / (peak(pha) - ped(pha))   # low_max(pha) is the highest PHA read in the low range

5b) if signal(meq) < xover(meq) we use the low range to read out. It is linear.
pha_low   = signal(meq) * peak(pha/meq) + ped_low(pha)
pha_low   += Gauss(ped_low_width(pha))

5c) if signal(meq) > xover(meq) we use the high range to read out. Must account for saturation.
pha_high = signal(meq) * slope(pha/meq) * saturate(pha) / (saturate(pha) + slope(pha/meq) * signal(meq)) + ped_high(pha)
pha_high += Gauss(ped_high_width(pha))

6) determine if the PHA signal is about above zero suppression thershold
if (pha_low > pha_thresh) keep pha   # pha_thresh comes from a calibration

7) determine if the signal is above veto threshold
if (veto_signal(meq) > veto_threshold(meq)) assert veto

get the veto threshold expressed in mips:
7a) veto_threshold = use_veto_pha_calib ?
(veto_thresh_pha(pha) - ped(pha)) / (peak(pha/meq) - ped(pha)) :   # get the number from a calibration
veto_mip(meq)   # get the number from job options

add noise to the signal level before comparing to the threshold
7b) veto_signal(meq) = signal(meq) + Gauss(veto_noise(meq))   # veto_noise comes from job options

8) determine if the signal is above cno threshold
if (cno_signal(meq) > cno_threshold(meq)) assert cno

get the cno threshold expressed in mips:
8a) cno_threshold(meq) = use_cno_pha_calib ?
(saturate(pha) * (cno_thresh_pha(pha) - ped(pha))) / (slope(pha/meq) * (saturate(pha) - cno_thresh_pha(pha) + ped(pha))) :
ped(pha), slope(pha/meq) and saturate(pha) from calibrations
cno_mip(meq)   # get the number from job options

add noise to the signal level before comparing to the threshold
8b) cno_signal(meq) = signal(meq) + Gauss(cno_noise(meq))

Reconstruction

1) start with pha
1a) if low range use linear scale
signal(meq) = (PHA - ped(pha)) / (peak(pha) - ped(pha))   # peak(pha) and ped(pha) from calibrations

1b) if high range use saturation eq.
signal(meq) = saturate(pha) * (PHA - ped_high(pha))) / (slope(pha/meq) * (saturate(pha) - PHA + ped_high(pha)))
# ped(pha), slope(pha/meq) and saturate(pha) from calibrations

2) express signal in terms of Z.
To do this correctly requires knowing the length_factor of the track in the tile/ribbon

signal(meq) = Z^2 * S(Z) * length_factor   where S(Z) = 0.608 + 0.393 exp(-0.00483 * Z^2)

length_factor = tile ? length / 10mm : length / ribbon_width;

I am not certain that there is an analytic solution for Z, but it can certainly be solved very quickly by iterating.

3) for merit tuple express signal in MeV.
signal(MeV) = signal(meq) * mev_per_mip(MeV/meq)   # mev_per_mip = 1.9 MeV/mip for tiles, 0.45 MeV/mip for ribbons

Note that this last step is only accurate in the linear regime. But since it is going in the merit tuple that means it is being used from background rejection, in which case the linear regime is the important one. For GCR calibration (ie, estimating if