06/10/2021

Tools for `grandlib`

This is the list proposed for classes and functions for grandlib, based on previous works & expertise playing with GRAND sims.
Authors: O. Martineau & B. Lago & S. Chiche & M. Tueros.

`grandlib` classes

Classes in `grandlib` would actually simply be (elaborate) structures containing the various infos associated to one entity (eg antenna). Yet most (if not all) operations would be external to the classes (see section "`grandlib` functions" below).

Antenna

A generic class to handle antenna info. Source: antenna class @ https://github.com/grand-mother/grand/blob/master/grand/simulation/antenna/generic.py

Attributes:

position *in GP300 referential (x = SN, y=EW, 0 = center of array + sea level) by default*
tilt *vector info on antenna orientation (in the GP300 frame to avoid issue with geodesy?). This may change with time (monitoing info)*
antenna_model *= Leff vector = f(theta,phi, nu) (will be corrected from calib info, hence unique for each antenna). Source: class AntennaModel in https://github.com/grand-mother/grand/blob/master/grand/simulation/antenna/generic.py*
Note: antenna model is a big vector,and we will probably have several antenna models. Hence memory/size issues should probably be taken into account here (pointer to a vector defined somewhere on the library)

To be added later:
elec_settings *Details on FE elec settings (amplification, filtering, triggering)*
ground_info *Info from calib on ground reflection*
atm_info *Info from atm calib*

3DTimetrace

A base class to handle a vectorial timetrace (voltage or Efield) @ the antenna level

Attributes

n_points *= timetrace length (should be read from electronics settings)*
step [ns] *n_points x step = total timetrace length*
traces [3,n_points] *numpy array with time traces for antenna arms X,Y,Z*
htraces [3, n_points] numpy array with Hilbert envelope of *time traces for axis X,Y,Z (in GP300 referential)*
anatrig_time [ns] absolute *trigger time determined from the combined time traces infosthrough post processing treatment. Several values (corresponding to different algorithms) should probably be considered.*

Are the following necessary? They are in principle straightforward to compute from basic numpy functions. Yet they might be saved to save CPU time.

mean *[3,1] array in µV or µV/m*
sigma *[3,1] array in µV or µV/m*
p2p *[3,1] array in µV or µV/m Peak to peak amplitude of signal*
hmax *Max of the Hilbert envelope (in µV or µV/m)*

Additional attributes:
Computed from analysis, they allow to determine if the timetrace can be associated to a possible EAS or to background. To be implemented later.
quality*: figure of merit qualifying the trace through a weighted score involving several parameters (nb of peaks, positions, signal/noise, calibration of the channel, etc). Only above a certain score should the trace be considered for further analysis. We could even consider two categories, with a (lower) threshold above which traces are valid for recons, and another one where showers are valid for EAS.*
n_peaks = nb of peaks in time trace… Needs a definition *(Nb of consecutive points above threshold?)*
peak_area [array of length n_peaks]
peak_duration [array of length n_peaks]
peak_amplitude [array of length n_peaks]
peak_time[array of length n_peaks]
+ many others, based for example on FFT quality, monitoring info (?), etc.

Voltage

A generic class to handle voltage @ the antenna level. **Inherits from 3DTimetrace class**

Attributes

GPS_trigtime [ns] *Experimental trigger time from GPS data*
is_trigged *boolean to determine if this antenna would have triggered (= True by default for experimental data)*
Additional attributes are probably needed to aggregate analysis of individual time traces and eventually determine if Voltage info is compatible with the EAS hypothesis.

Efield

A generic class to handle Efield @ the antenna level. **Inherits from 3DTimetrace class**

Attributes

eta *[deg] Polarization angle of the reconstructed Efield in the shower plane.*
a-ratio *[float] ratio of the geomagnetic to charge excess contributions.*

Shower

Speaks for itself, right?

Attributes:

energy [eV]
Xmax [g/cm2]

Xmaxpos *position in GP300 ref*
k *direction of origin*
corepos *position on the ground in GP300 ref*
… and also true value for sims and data)
+ many more simulation infos that will be either available for sims or reconstructed fluence, radio source, etc)

Event

A combination of 3DTimeTraces from different antennas in causal coincidence. Can be associated to an elm wave and a shower.

Attributes

ants *Antennas (see class above) participating in the event*
L *= event multiplicity*
voltage *objects of class Voltage corresponding to voltage along x,y z (either experimental or simulated). Length = L*
rec_efields *objects of class Efield corresponding to efield (length = L) reconstructed from voltage. Length = L*
sim_efields *objects of class Efield corresponding to simulated efield (length = L) Empty for experimental data.*

is_wave *boolean to decide if this group of timetraces is indeed associated to a single wave (based on quality of plane & sph recons)*
if is_wave == 1 then:
Plane recons paras:
k_p *vector of origin for plane wave fit*
chi2_p *Chi2 of plane wave fit*
Sph recons paras:
x_source *position of source*
chi2_p *Chi2 of plane wave fit*

is_shower *Boolean to assess EAS nature built on quality of recons*
trueShower *(if simulation) object of Class Shower with true shower parameters*
if is_shower == 1 then:
  recoShower *reconstructed shower parameters*

Projections in shower plane + modulus are straightforward with dedicated functions, so no need to save them.

---

grandlib functions

06/10/2021

Signal handling:

filter(trace, step, freq_min,freq_max,filter_model, filter_model_)

Inputs:

- trace is simply a numpy array of size = n_channels x n_points
- step is the time step

Returns: filtered time trace of same dim as trace

Maybe several filtering methods should be defined: Butterworth, experimental (i.e. determined from true filter, …)

path: /sps/trend/chiche/grandlib/functions/filter_traces.py

generate_gaussian_noise(sigma)

Inputs: sigma is a 1xn array with std dev of the noise for the n channel

Returns: noise realization of size = n_channels x n_points. Of course noise is channel-independent. This can then be added to the trace.

Other function should be defined as well, with different noise models.

path: /sps/trend/chiche/grandlib/functions/add_noise.py

digitize(trace, fSamp)

Digitize the signal @ the given sampling frequency.

there are several ways of doing this

path: /sps/trend/chiche/grandlib/functions/digitize.py

Hilbert(trace) *compute Hilbert envelope…*

fft(trace, fftparameters)

inversefft(fft, inversefft_parameters)

find_peak(trace, model, model_parameters)

(returns peak location (and if it exists a noticeable peak or not, or how many, etc)

find_peak_area(trace, model, model_parameters, window=auto)

(returns the peak area, and the integration limits if window= auto)

find_baseline(trace, model, model_parameters)

(finds the 0 level of the trace)

find_noise(trace, model, model_parameters)

(finds the noise in the trace. returns a trace to be substracted from the signal)

remove_noise(trace, noisetrace)

find_signal_over_noise(trace)

using all the above, to get an idea if the trace is usable for reconstruction or not.

06/10/2021

Geometry tools

project_to_shower_plane(k, X, Xpos)
Inputs:

- k: vector of propagation of the shower
- X: vector coordinates in GRAND ref. X can be a position (dim = (3,1)) a time trace (dim = (3,n_points)).
- Xpos (radio source position, true or reconstructed)

Returns: X coordinate in k,kB,k(kB) referential. Units = meters

path: function "GetInShowerPlane" in /sps/trend/chiche/grandlib/functions/shower.py
also a function "GetinGeographicFrame" to go from (v, vxb, vxvxb) to (x,y,z)

project_to_angular_plane(k, X, Xpos)
Inputs: same as previous
Returns: angular plane coordinates (omega, eta) for each position X. Units = deg

path: (to get omega angle) function "get_w" in */sps/trend/chiche/grandlib/functions/shower.py*
(to get eta angle) function "get_alpha" in */sps/trend/chiche/grandlib/functions/shower.py*

Also polarization computation functions (modulus, geomagnetic componant, charge excess component) path: */sps/trend/chiche/grandlib/functions/get_polarization.py*

compute_gramage(XposA, XposB, k, atm_model)
to compute the grammage between two points, and from there derive other functions.
Inputs:

- XposA, XPosB: stop and start positions in GRAND ref. Default value for XPosB could be entrance in the atmosphere (most common value)
- k: vector of propagation of the shower
- atm_model: atmospheric model to be used. Default should be provided (Linsley)

Returns:

- Grammage between XposB and XposA. Units = g/cm2.

Path: The function is almost written in "_dist_decay_Xmax" in */sps/trend/chiche/grandlib/functions/shower.py (just a few modifications are needed to give the expected output)*

early-late_correction()
Corrects amplitude for 1/distance dilution
path: */sps/trend/chiche/grandlib/functions/correct_early_late.py*

project_spherical(k, point_of_origin) project vector in spherical refential e_k, e_theta, e_phi. Usefull for antenna response computation. Can also be convenient because E_k or V_k (component *along direction of propagation) is null under plane wave hypothesis.*

Detector simulation

convolve(ant, trace, Xpos)

06/10/2021

Input:

- ant is the Antenna object associated with the time trace.
- trace is the Efield @ antenna position. Format = (3,n_points) numpy array.
- Xpos is the radio source position (true or reconstructed). It is needed to compute the wave direction of propagation and determine the wave direction of origin.

Returns: Voltage at antenna output. Format = (3,n_points) numpy array.
This function could use a Shower object as input (all needed infos are in it). Additionally, we could consider computing traces at different stages (antenna output/FE output/digitizer output) and return all of these, or implement different function at each stage.
Source: compute_voltage @
https://github.com/grand-mother/grand/blob/master/grand/simulation/antenna/generic.py

deconvolve(ant, trace, Xpos)
Inputs: same as above, except that trace is here the voltage @ antenna ouput.
Returns: Reconstructed Efield at antenna output. Format = (3,n_points) numpy array.
Inverse function from the above.

Reconstruction

These are
test_EAS(event) *Applies tests on the traces and additional attribute to determine if the event is an EAS candidate. Returns a is_EAS Boolean. Several implementations are probably needed.*
plane_recons(event) *Guess*
sph_recons(event) *Guess*
compute_ADF() *Valentin's method for recons based on Amplitude Distribution Function*
compute_energy() *Rio's method*