Proposed structure for tools packages [GRAND code reformatting]

Here we present a detailed list of classes that could compose the **tools** package in order to give detailed guidelines when it comes to implementation. Please take time to read it, think about it and improve it! This is an *alive* document that will be key in defining the best GRAND code. Note that using existing solutions (astropy or other standard libraries) is very welcome.

# Class Position

Gives position of an object in space.

### Attributes

* **[x, y, z]**

Position in the [**ECEF referential**](https://en.wikipedia.org/wiki/ECEF) (defined for instance as 0: center of Earth, x towards Greenwich meridian, y in Equator plane and z towards geographic pole)

### Methods

* [x,y,z] = get\_cartesian(posA, refA=’ecef’)

returns Cartesian coordinates of object *posA* in referential *refA.* Default is ECEF referential. This applies throughout this doc (and the code).

* [rho,theta,phi] = get\_spherical (posA, refA=’ecef’)

returns spherical coordinates of object *posA* in referential *refA*. Note: angles measured according to GRAND convention: *theta* wrt **z** axis and *phi* wrt **x** axis, sign follows trigonometric convention.

* [lat, long, z] = get\_geographic\_coordinates(pos)

returns geographic coordinates of *posA*. *z* is WGS84 (GPS) value at this *(lat, long)***.**

All these “get” methods should be associated with equivalent “create” methods:

* pos = create\_position(x, y, z, refA)
* pos = create\_position(rho, theta, phi, refA)
* pos = create\_position(lat, long, z)

In addition:

* h = get\_height(posA)

gets height above ground at the corresponding *(lat, long)* location.

* h = get\_altitude(posA)

gets altitude above sea level at the corresponding *(lat, long)* location. Can differ from *z*.

* u = get\_Bfield(posA)

returns the B field vector at location *posA*.

# Class Vector

Defines a vector and proposes associated operations. In practice a vector can be defined as a position in one referential (and therefore class vector probably useless?) but this could make things clearer for end user.

### Attributes:

* (posA, posB)

couple of positions defining the vector

### Methods

* [x y z] = get\_ cartesian(u, refA)

returns Cartesian coordinates of vector *u* in referential *refA*.

* [rho theta phi] = get\_ spherical(u, refA)

returns spherical coordinates of vector *u* in referential *refA*. Note: angles measured according to GRAND convention: theta wrt **z** axis and phi wrt **x** axis, and sign follows trigonometric convention.

* u = create\_vector(posA, refA)

creates vector linking origin of *refA* to position *posA*.

* u = create\_vector(posA, posB)

creates vector linking position *posA* to position *posB*.

* a = dot(u,v)

scalar product

# Class Referential

### Attributes

* Origin (type = position)
* x, y and z base vectors (type = vector)

### Methods

Define usual referentials:

* ref = get\_ecef()

returns ECEF referential

* *ref* = get\_grand\_ref(posA)

returns [ENU referential](https://en.wikipedia.org/wiki/Axes_conventions) with origin=*posA* and following GRAND conventions (x=geographic North, y=West, z= vertical at location *posA*)

* *ref* = get\_zhaires\_ref(posA)

returns ENU referential with origin=*posA* and following ZHaireS conventions (x=geomagnetic North, y=West, z= vertical at location *posA*)

* *ref* = get\_coreas\_ref(posA)

returns ENU referential with origin=*posA* and following CoREAS conventions (x=East, y=North, z= vertical at location *posA*)

* pos = get\_origin(refA)

returns origin of referential *refA*.

* [vec, vec, vec] = get\_base(refA)

returns base vectors of referential *refA*.

# Class Detector

### Attributes

* {…, posX, …}

List of antenna position at ground

* {…, vecX, ….}

Associated list of vectors normal to ground at antenna location. **Here we need to specify area over which slope is computed: 30m probably not enough. 200m?**

**Warning: detector could in principle be infinite (or at list very large) for RETRO simulation… This may require specific handling.**

* Iterator to access the position & slope lists.
* Antenna type (type=string), could prove useful in the long run (ie different antenna type 🡺 different antenna response).
* Antenna height (above ground). Identical for all antennas. Necessary for shadowing/signal attenuation computation.

### Methods:

* det = create\_from\_file(“xxx.txt”, refA)

generates a detector object *det* from a file giving antenna positions in referential *refA*. Altitude and slopes may be computed on the fly from TURTLE library functions if not present in file.

* det = create\_from\_parametrisation(step=1000, pattern=”square”,boundingbox={posA, posB, posC, posD})

generates a parametric array inside *boundingbox* (infinite over the full Earth ground surface if not specified), following the specific *pattern* (square, hexagon, etc) and *step* size. TURTLE computes heights & slopes on the fly.

* {…, posX, …} = get\_positions(det)

returns antenna positions

* {…, posX, …} = get\_slopes(det)
* det = select(parameters)

returns a subdetector *det* composed of the antennas passing the *parameters* selection cut. These still have to be defined, but could typically be of the type “distance to shower axis < 2000m”.

# Class Shower

### Attributes:

* list of particle IDs (following usual conventions)
* list of associated impulsions (type = vector)
* injection height (type = position): point of first interaction

### Methods

* E = get\_energy(shower)

computes (from impulsions) energies of particles contributing to the shower (ie all except muons & neutrinos) & sums them up.

* u = get\_direction(shower, refA)

get direction **of origin** (propagation\*-1) of *shower* in referential *refA*. Note: here we follow the (new) GRAND convention: receiver point of view, looking for the direction of origin of the shower.

* posA, grammage = get\_Xmax (shower, refA)

computes (average/expected) value of Xmax (in g/cm²) and actual position *posA* in *refA*. This assumes a specific atmospheric model (to be added in attributes?).