Status of the GRAND simulation software

V. Niess[1], on behalf of the GRAND software team

GRAND Workshop

26th April 2019

Dunhuang, Gansu, China

(pdf) (html)

[1] Université Clermont Auvergne, CNRS/IN2P3, LPC, F-63000 Clermont-Ferrand, France

Innovative developments for the White Paper

- **Accelerator for radio signal computations from showers:** Radio Morphing, (Astropart. Phys. -*under review*-, arXiv:1811.01750)

  > Bottleneck of the simulation chain. Greatly improved with the Radio Morphing technique.

  Source: `Python3 + numpy`, **Interface**: `Python` package

- **Coupled ν_τ-τ transport by Backward Monte-Carlo**: DANTON (arXiv:1810.01978)

  > Recycling of tools developed within TREND and for muography. Detailed yet fast MC simulations.

  Source: `C99`, **Interface**: library (`C` API) *and* executable (`JSON` API)

  Beta version, *validated*, but the geometry & API will change

Dedicated wrappers developed for the WP

- **Topography utilities**: GRAND-TOUR

  > Wrapper of TURTLE (Submitted to CPC, arXiv:1904.0345) a topography library dedicated to MC. Extends TURTLE with local frames (ENU) and ray tracing helpers.

  Source: `C99`, **Interface**: library (`C` API) *and* `Python` module (`C` extension)

  Will be migrated to TURTLE (`C` API) and to the new Python3 tools

- **Generator of decaying τ for GRAND**: RETRO

  > Wrapper of DANTON. Generates decaying τ with a bias towards those emerging from rocks & pre-select candidate antennas for the Radio Morphing.

  Source: `C99`, **Interface**: executable (`JSON` API)

Scripts developed for the WP

- **Monte-Carlo production**

  > Two sites: CC-IN2P3 for the ν_τ to τ chain (using **TREND** resources) and ForHLR1 (**KIT**) for the Radio Morphing. Data stored as amendable `JSON` objects at CC-IN2P3, using iRODS.

- **Antenna response & digitization**

  > Convolution with pre-computed antenna response (**NEC**). Done at CC-IN2P3.

- **Analysis scripts**

  > Selection of candidate MC events and computation of the GRAND sensitivity. Done locally from reduced data.

  Source: `Python2`, **Interface**: `Python` scripts with CLI

Scripts are shared over GitHub, but the committed version might differ from the one used for the MC production or analysis.

Some related projects (alpha versions, unstable)

- **Monte-Carlo event displays**

  > Two successive projects have been developed:
  >
  > - `retro-player`: 3D with navigation, based on panda3d. *Fancy but ressource heavy and not convenient for quantitative analyses.*
  > - `retro-display`: 2D projections, based on `matplotlib`.

  Source: `Python2`, **Interface**: `Python` scripts with CLI and `JSON` API

- **Shell like navigation of iRODS data**: `ishell`

  > Allows to navigate and manage iRODS data as if using an `ssh` connection to a remote host. Available from PyPi.

  Source: `Python2`, **Interface**: `Python` package and executable with CLI

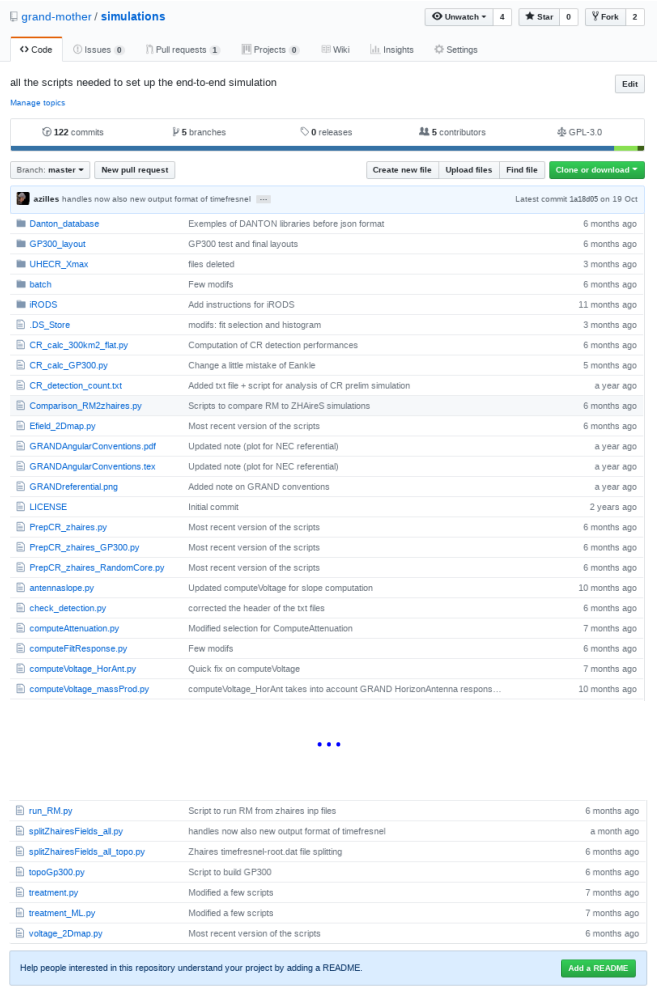
But ...

the code became messy

We had to crush lines for the White Paper. We had little time to gather and discuss the organization of the software.

> E.g. grand-mother/simulations has 55 files (43 Python scripts) from 7 contributors. They span 9k lines of code (half of which are commented out). This repo has 4 out of sync branches as well.

> Following the White Paper submission, we decided to clean up things before getting farther ...



Code statistics (GitHub, pre-White Paper)

Young developers (Physicists) contributing to the GRAND simulation use `Python`. This seems to be a common trend among *nowdays* Scientists.

- With `Python` one can easily be *productive*. But, the product can also easily be *slow* ...
- To be efficient, one needs to defer *CPU intensive* tasks to a lower level language and/or a specialized hardware (e.g. GPU).

Actually `FORTRAN`, and `C` are extensively used under the hood, e.g. when importing `numpy`, or when calling external Monte-Carlo libraries (e.g. DANTON).

Language	Files	lines
Python	**128**	**18,803**
C	11	1,992
Bourne Shell	12	462
Markdown	15	260
JSON	9	124
make	7	103
CSS	1	73
TeX	1	59
HTML	1	59

Table 1: statistics, excluding the `reconstruction` package.

Actions for the software upgrade

- We discussed by phone, on Skype, slack and by email. Following, a poll was organised in December 2018.

  > The detailed list and results are available on the software wiki.

- A preliminary design was proposed and discussed at the December GRAND call.

  > Building upon existing tools rather than developing brand new ones was approved.

- We met in Paris in January 2019.

  > A **mini workshop** was organized for the **software upgrade**. We designed a tentative new layout for the code, and exercised with a preliminary wrapper for **GRAND packages**.

Below are the conclusions of this process

Decisions for the software update

- **Python** is adopted as high level interface, i.e. a glue for the low level simulation components.

  > Wrappers need to be developed for abstracting generic objects, E.g. coordinates, geo-magnetic field, particle showers, antenna arrays, ...

- The interface will be built on `astropy` (and so `numpy`). It provides *validated* utilities:

  - **data structures**, e.g. coordinates systems, tables, ...
  - **files I/O**, e.g. HDF5, YAML, ...
  - **algorithms**, e.g. convolution and filtering.

  > It has weak dependencies and a reasonable memory footprint (35 MB + 70 MB for `numpy`).

- Standard utilities will be used in order to control and improve the code quality, as it develops.

  > There are new tools to learn. Those are however not specific to GRAND. These skills will be useful for any (Python) project.

Decisions for the software update

- **Python 3.7** will be used (the current head).

  > `Python 2` will be deprecated in 2020.

- The GRAND software should run on **Linux** (batch systems) and **OSX**, i.e. it must be UNIX / POSIX compliant.

  > Half of us work on a Mac. Nobody seems to develop on `Windows`, according to the poll.

- The software is distributed under **GNU LGPL-3.0** license, by default, using `The GRAND collaboration` as copyright owner. Individual authorship can however be recognised in a separate `AUTHORS` file.

  > Note that GNU GPL are contaminant licences. Note also that in principle some of us (French contracts) do not own the copyright on code developed at work. The employer does.

- The source code will continue to be hosted on **GitHub**. New versions and binary releases will be distributed on PyPi.

Status of the upgrade

- Package manager: `grand-pkg` status beta

  > Allows to create, update & configure GRAND packages. Provides stats on your code and utilities for unit testing, documenting and distributing.

- New wrappers :
  - `grand_libs` status WIP

    > Install and manage libraries. Implemented: `GULL` (geomagnetic field) & `TURTLE` (topography). Missing: `DANTON`.

  - `grand_tools` status WIP

    > Implemented: utilities for coordinates transform, geomagnetic field and *preliminary* topography. Missing: antenna arrays and particle showers.

  - `grand_store` status WIP

    > Manage & distribute large data sets, e.g. topography tiles.

Status of the upgrade

- Code refactoring:
  - Radio Morphing status done

    > Preliminary package version and tools. Could benefit from using `grand_tools` once stable.

  - `radio_simus` status beta

    > Digitization, background injection & signal filtering.

  - τ generator (`retro`) status to do

    > Rewrite (or wrap) in Python using `grand_tools` and generalize, e.g. for **cosmic rays**.

- Website & docs status beta (here)

  > Summary statistics for GRAND packages and unified documentation.

`grand-pkg` : a package manager for GRAND

More on this in the hands-on session on Saturday.

- Based on `git` (via hooks), web services (via GitHub) and `pip`. Requires `Python 3`. Install via `pip3` as:

```
pip3 install --user grand-pkg
```

- Provides 3 utilities, mirroring `git` commands:
  - `grand-pkg-init`

    > Create or initilialise a GRAND package.

  - `grand-pkg-update`

    > Update the package manager and the related GRAND package data.

  - `grand-pkg-configure`

    > List or modify the meta-data of a GRAND package.

`grand-pkg` : a package manager for GRAND

More on this in the hands-on session on Saturday.

- Provides a pre-configured test environments via vanilla `Python` modules: `unittest` and `doctest`. Run the test suite as:

```
python3 -m tests
```

- Provides helpers for `setuptools` and `distutils` for building and distributing the package.

> Note that because of this, installing a GRAND package from the source currently requires `grand_pkg` to be already installed. Binary distributions, e.g. available on PyPi, do not.

Extra utilities are foreseen: `grand-pkg-clone` and `grand-pkg-upload` for easier management and distribution of GRAND packages.

`grand_libs` : wrapping of C libraries

Problematic: distribute and wrap versioned `C` librairies for the end user.

- GRAND specific libraries are built and installed as package data.

  > This is done automatically the first time that a library wrapper is imported or if the `C` library version is updated.

- The Python wrapping is done with `numpy.ctypes`.

  > vectorization is provided by simple C wrappers appended to the library source at compile time.

Pros: requires **little extra development**, e.g. compared to writing a `C` extension module for `Python`. In addition, the distributed binary is **portable** since it is pure `Python` byte-code.

Cons: generates a **slow-down** the first time the wrapper is imported (library built). The compiled library might differ from its tested version depending on the **host environment**.

Note that this approach is opposite to what is usually done, e.g. by `conda`.

`grand_tools.coordinates`: extension of `astropy.coordinates`

Problematic: correctly handle coordinates between simulation components that use different systems.

`astropy` provides a model for handling coordinates systems. The base objects are:

- `BaseRepresentation`, coordinates parameterisation within a frame, e.g. Cartesian or spherical.
- `BaseCoordinateFrame`, a container with frame attributes (e.g. origin, basis) and optionally coordinates data as `numpy.array` in a default representation.

> `astropy.coordinates` does not have a vector algebra. Coordinates transform as points, even when using a `UnitVectorRepresentation`.

> Low level frames, e.g. ITRS, do not forward the observation time, but use a default one instead. This is error prone, see e.g. #8390.

`grand_tools.coordinates`: extension of `astropy.coordinates`

Implemented solution: wrap TURTLE functions as extra Representations and frames for astropy.coordinates.

- `GeodeticRepresentation` for to geodetic (latitude, longitude, height) transforms.

  > In astropy this is done with a specific `EarthLocation` object that can be converted to an `ITRS` frame.

- `HorizontalRepresentation` for angular coordinates.

  > In astropy this is done with an `AltAzFrame`.

- `ENU` frame for local coordinates.

  > Origin and basis are configurable, e.g. x-axis along the magnetic North, or the geographic East.

- `ECEF` analog to `ITRS` but with extra attributes.

grand_tools.coordinates: extension of astropy.coordinates

- `ENU` and `ECEF` frames forward the observation time, if undefined. This allows simple transform as:

```
>>> ecef = enu.transform_to(ECEF)
```

  instead of:

```
>>> itrs = enu.transform_to(ITRS(obstime=enu.obstime))
```

- In addition, the type of the coordinates (vector or point) is automatically inferred from the data units.

> If equivalent to `meters` then it is assumed that the data transform as a point. Otherwise as a vector.

> An explicit `is_vector` flag will be added for overriding this behaviour, e.g. for the antenna equivalent length vector.

Example of usage of `grand_tools`

```
>>> import astropy.units as u
>>> from grand_tools.coordinates import ECEF
>>> from grand_tools import geomagnet

>>> coordinates = ECEF(representation_type="geodetic", latitude=45 * u.deg,
...                    longitude=3 * u.deg, obstime="2019-01-01")
>>> field = geomagnet.field(coordinates)
```

The TURTLE and GULL library are used transparently for the end user. The IGRF12 geomagnetic model is used by default, packaged with `grand_tools`.

```
>>> from grand_tools.coordinates import ENU
>>> from astropy.coordinates import EarthLocation

>>> x = np.array([0, 1, 2, 3]) * u.m
>>> y = np.array([1, 2, 3, 0]) * u.m
>>> z = np.array([2, 3, 0, 1]) * u.m

>>> coordinates = ENU(x, y, z, location=EarthLocation(latitude = 45 * u.deg,
...                                                   longitude = 3 * u.deg),
...                                                   obstime="2019-01-01"))
>>> field = geomagnet.field(coordinates)
```

Local coordinates and vectorized data can be used as well.

Outlooks

- **ν simulation** chain *functional*. Upgrade & packaging of the developped **software tools** *ongoing*.
- Adapt the simulation chain for **cosmic rays**, *ongoing*.
- Adapt / integrate the tools developed for the simulation chain for the **reconstruction**: to be done.
- ARA-GRAND meeting, *collaborative software tools*, Berlin, June 18-20th.

Get a deeper insight on the various components of the software in the hands-on sessions

Hands-on sessions - Morning

- Introduction to (*GRAND*) **Python** 09h00-10h30

> Setup a **proper work environment** and **1st steps** with `Python 3`. For more expert people, be a **beta tester**: experiment with the upgraded GRAND software, *and submit issues* when you find **bugs**.

- Simulation of **air showers** for GRAND 11h00-12h30

1. **ZHAires**

> Introduction to AIRES and it radio extension: ZHAires. Installation, run your first simulation and plot the results.

2. **Radio Morphing**

> Brief overview and discussion of the code. Then, produce a radio-signal distribution of an air-shower with Radio Morphing.

Hands-on sessions - Afternoon (1)

- GRAND **scripts** : playing with the generated radio signals 14h00-15h30

1. **Signal** processing

> Detailed computation of the antenna response to an electromagnetic transient signal, and/or get accounted with measured signals, e.g. how the filtering frequency band may affect time traces and amplitude pattern at ground on the GRANDProto300 detector.

2. **Angular reconstruction**

> Reconstruct the arrival direction of the shower with two methods : plane wave front and hyperbolic wave front. Play with the reconstruction parameters.

Hands-on sessions - Afternoon (2)

- **C tools** for the GRAND simulation and their interfaces 16h00-17h30

  1. Handling topography data with `TURTLE`.

     > Brief introduction to TURTLE. Experiment with its interface and/or with the `grand-tour` wrapper.

  2. Play with the ν simulation: `DANTON` and `RETRO`.

     > Basics of the backward ν_τ-τ coupled transport. Experiment with the C and/or JSON API.