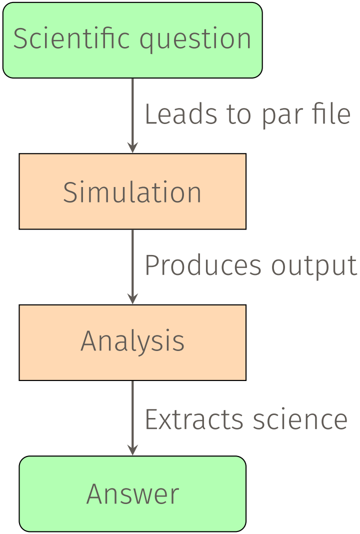
ANALYZING EINSTEIN TOOLKIT SIMULATIONS WITH PYTHON

July 27, 2021

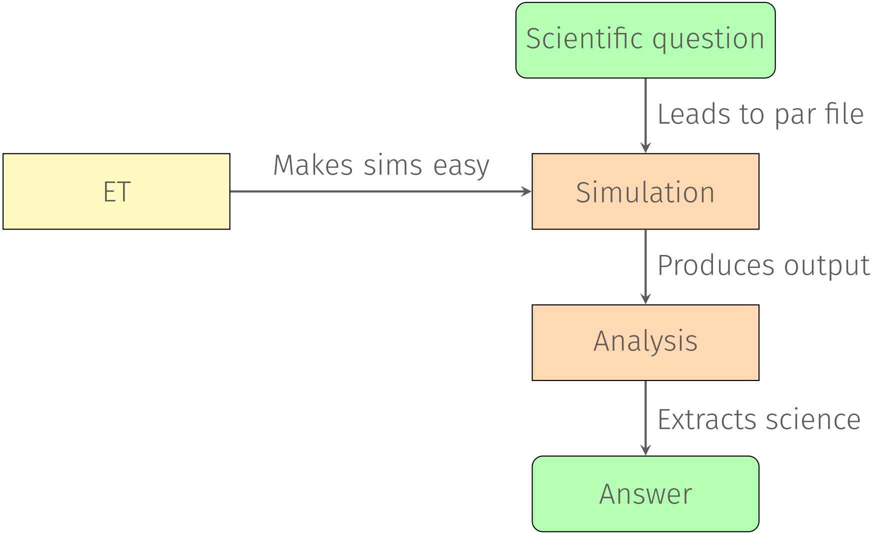
Gabriele Bozzola

Department of Astronomy and Steward Observatory,
University of Arizona

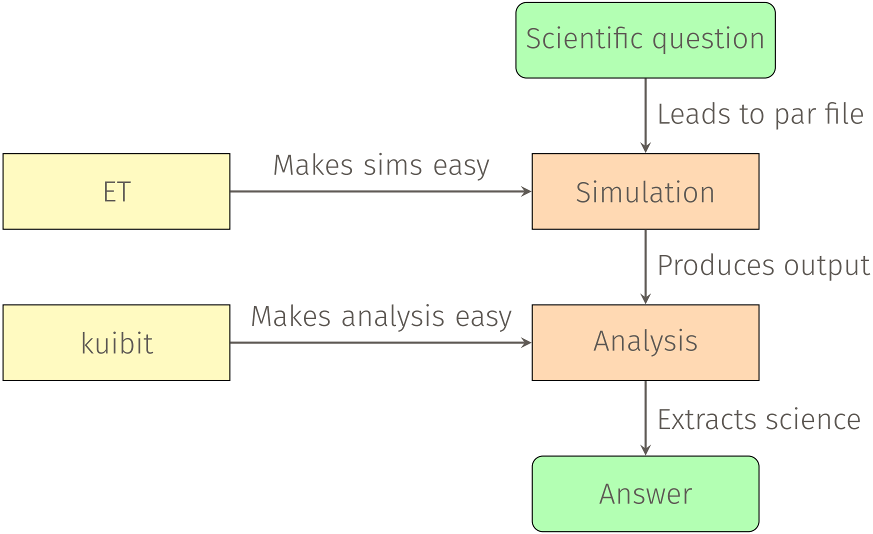
RUNNING A SIMULATION $\neq$ DOING SCIENCE



RUNNING A SIMULATION $\neq$ DOING SCIENCE



RUNNING A SIMULATION $\neq$ DOING SCIENCE



ET MAKES SIMS EASY BY TAKING CARE OF INFRASTRUCTURE AND (MOST) SCIENCE

When you use `ET`, you don't have to worry about:

- → Handling parallelization/memory management
- → Implementing checkpointing
- → Retrieving codes from different repos/sources
- → Compiling external libraries
- → Parallel output in efficient HDF5
- → Implementing science (spacetime and GRMHD solvers, wave extraction, horizon finder, ...)
- → ...

THE OUTPUT OF ET SIMULATIONS IS TYPICALLY AN INTRICATE MESS

Often, you have:

- → Several directories, one per simulation restart
- → `asc` files for reductions and scalars (max, min, ...)
- → `asc` and/or `h5` files for grid functions (`_file_XXX.h5` for 3D)
- → `gp` and `BH_Diagnostics` files for horizons
- → `mp` (ASCII or HDF5) files for waves
- → ASCII or HDF5 output from other thorns (e.g., `Outflow`, `VolumeIntegrals`)
- → ...

Formats and conventions are inconsistent and depend on the par file

KUIBIT MAKES ANALYSIS EASY BY TAKING CARE OF INFRASTRUCTURE AND (SOME) SCIENCE

When you use `kuibit`, you don't have to worry about:

- → Reading and parsing the data (ASCII, and HDF5)
- → Handling simulation restarts
- → Writing visualization routines
- → Implementing science (computing strains, ...)
- → Writing meaningful interfaces (e.g., `TimeSeries`)
- → ...

kuibit takes care of the low-level details and lets you focus on science

KUIBIT IS A PYTHON LIBRARY FOR POST-PROCESSING SIMULATIONS

- At first order, reimplementation of Wolfgang Kastaun's `PostCactus`
- Support for
  - 1D, 2D, 3D, HDF5 and ASCII grid data
  - timeseries, frequency series (`CarpetIOASCII`)
  - apparent horizons and quasi-local measures
  - gravitational waves with `WeylScal4` (energy, angular momenta, mismatch, extrapolation to infinity, ...)
  - detector sensitivity curves
  - unit conversion
  - visualization
  - writing command-line scripts
  - full list of features: **sbozzolo.github.io/kuibit/features.html**

KUIBIT HAS EXCELLENT DOCUMENTATION (SBOZZOLO.GITHUB.IO/KUIBIT)

Getting started with SimDir

The `SimDir` class provide easy access to simulation data. Most data analysis start by using this object. `SimDir` takes as input the top level directory containing simulation data, and read and organizes the content. `SimDir` contains the index of all the information that is possible to extract from the ASCII and HDF5 files. If there are restarts, `SimDir` will handle them transparently.

Defining a SimDir object

Assuming `gw150914` is the folder where a simulation was run. `gw150914` can possibly contain multiple checkpoints and restarts.

```
import kuibit.simdir as sd

sim = sd.SimDir("gw150914")
```

USAGE

kuibit.visualize_matplotlib.**plot_color**(*data*, ***kwargs*) [source]

Plot the given data.

You can pass (everything is processed by `preprocess_plot_grid()` so that at the end we have a 2D NumPy array): - A 2D NumPy array, - A `UniformGridData`, - A `HierarchicalGridData`, - A `BaseOneGridFunction`.

Depending on what you pass, you might need additional arguments.

If you pass a `BaseOneGridFunction`, you need also to pass `iteration`, and `shape`. If you pass `HierarchicalGridData`, you also need to pass `shape`. In all cases you can also pass `x0` and `x1` to define origin and corner of the grid. You can pass the option `resample=True` if you want to do bilinear resampling at the grid data level, otherwise, nearest neighbor resampling is done. When you pass the NumPy array, passing `coordinates` will argument will make sure that those coordinates are used.

All the unknown arguments are passed to `imshow`.

Parameters: • **data** (*2D NumPy array, or object that can be cast to 2D NumPy array.*) – Data that has to be plotted. The function expects a 2D NumPy array, but the decorator `preprocess_plot_grid()` allows it to take different kind of data.

APIs

kuibit 1.0.0b0 documentation » Working with time...

KUIBIT

Previous topic

Working with Simulation Directories

Next topic

Working with grid data

Quick search

Go

Working with time series, frequency series, and unit conversion

In this notebook, we show some of the most useful features of the timeseries module. To do so, we will analyze a fake gravitational-wave signal. We will also show the frequencyseries module and the unitconv modules.

First, let's generate this signal.

(This notebook is meant to be converted in Sphinx documentation and not used directly.)

```
[1]: import matplotlib.pyplot as plt
     import numpy as np
     from kuibit import timeseries as ts
     from kuibit import series
     from kuibit import unitconv as uc
     from kuibit.gw_utils import luminosity_distance_to_redshift

     %matplotlib inline
```

```
[2]: t = np.linspace(0, 20, 5000)
     y = np.sin(t)

     # Generate a TimeSeries by providing the times and the values of the series
     gw = ts.TimeSeries(t, y)
```

To access the times and the values, use `gw.t` and `gw.y`.

TUTORIALS

Examples

Below you will find a list of examples to perform more or less common analysis. You can immediately start doing science without writing one line of code using these examples. The scripts provided can be used for plotting, extracting gravitational waves, or other useful information. To get the most out of these examples, check out the recommendations on how to use the examples page.

Note that all these examples contain a significant fraction of boilerplate that is needed to keep them general and immediately useful. When learning `kuibit`, you can ignore all of this.

You can download these examples as archive from the GitHub release page, which is automatically updated with each release.

Scripts

- plot_1d_vars.py
- plot_ah_coordinate_velocity.py
- plot_ah_found.py
- plot_ah_radius.py
- plot_ah_separation.py
- plot_constraints.py

EXAMPLES

THE EXAMPLES CAN IMMEDIATELY BE USED

- → Plot horizon trajectories → `plot_ah_trajectories.py`
- → Plot timeseries (e.g., maximum rest-mass density) → `plot_timeseries.py`
- → Compute GW strain → `plot_strain_lm.py`
- → Make a 2D video of a grid variable → `mopi grid_var`
- → ...

```
$ mopi grid_var --resolution 500 --plane xy --variable rho_b --colorbar
                --datadir simulation --interpolation-method bicubic
                --logscale --vmin -7 --vmax -1 --parallel --outdir movie
                -x0 -30 -30 -x1 30 30
```

THE EXAMPLES CAN IMMEDIATELY BE USED

- Plot horizon trajectories →`plot_ah_trajectories.py`
- Plot timeseries (e.g., maximum rest-mass density) →`plot_timeseries.py`
- Compute GW strain →`plot_strain_lm.py`
- Make a 2D video of a grid variable →`mopi grid_var`
- …

```
$ mopi grid_var --resolution 500 --plane xy --variable rho_b --colorbar
                --datadir simulation --interpolation-method bicubic
                --logscale --vmin -7 --vmax -1 --parallel --outdir movie
                -x0 -30 -30 -x1 30 30
```

`kuibit` improves reproducibility and code reuse

EXAMPLES SUPPORT TAB-COMPLETION

```
kuibit-aOKykpx3-py3.8 ❯ plot_grid_var.py --resolution
--absolute                                                                  (Whether to take the absolute value.)
--ah-alpha                                              (Alpha (transparency) for apparent horizons (default: %(default)s))
--ah-color                                                           (Color name for horizons (default is '%(default)s').)
--ah-edge-color                                             (Color name for horizons boundary (default is '%(default)s').)
--ah-show                                                                                   (Plot apparent horizons.)
--ah-time-tolerance          (Tolerance for matching horizon time [simulation units] (default is '%(default)s').)
--colorbar                                                                         (Whether to draw the color bar.)
--configfile                                                                                     (Config file path)
--corner
--datadir                                                                                         (Data directory)
--figname                                                     (Name of the output figure (not including the extension).)
--fig-extension                                                 (Extension of the output figure (default: %(default)s).)
--help                                                                            (show this help message and exit)
--ignore-symlinks                                                                  (Ignore symlinks in the data directory)
--interpolation-method         (Interpolation method for the plot. See docs of np.imshow. (default: %(default)s))
--iteration                                                                   (Iteration to plot. If -1, the latest.)
--logscale                                                                                (Whether to use log scale.)
--multilinear-interpolate  (Whether to interpolate to smooth data with multilinear interpolation before plotting.)
--origin
--outdir                                                                                        (Output directory)
--pickle-file                                                                       (Read/write SimDir to this file)
--plane                                                                         (Plane to plot (default: %(default)s))
--resolution                                            (Resolution of the grid in number of points (default: %(default)s))
--variable                                                                                      (Variable to plot.)
--verbose                                                                                   (Enable verbose output)
--vmax                                            (Maximum value of the variable. If logscale is True, this has to be the log.)
--vmin                                            (Minimum value of the variable. If logscale is True, this has to be the log.)
```

A (very) quick tour

EXAMPLE: PLOT CONTOURS B2/P RATIO WITH Z = 2 AT T = 0

```
from kuibit.simdir import SimDir
from kuibit.visualize_matplotlib import plot_contourf, save

vars_xyz = SimDir('simulation').gridfunctions.xyz

b, P = vars_xyz['b'][0], vars_xyz['P'][0] # 0 is the iteration

ratio = b*b/P
print(f'Maximum {ratio.abs_max()} is at {ratio.coordinates_at_max()}')

ratio_on_z2 = ratio.sliced([None, None, 2])

plot_contourf(ratio_on_z2, x0=[-30, -30], x1=[30,30], shape=[500,500],
              colorbar=True)
save('plot.pdf')
```

```python
from kuibit.simdir import SimDir
from matplotlib.pyplot import plot, savefig

rho = SimDir('simulation').timeseries.maximum['rho']

rho_fft = rho.to_FrequencySeries()

plot(abs(rho_fft))

savefig('plot.pdf')

print(f'Value at time t = 3 is {rho(3)}')
```

EXAMPLE: COORDINATE VELOCITY V^X OF AN APPARENT HORIZON

```python
from kuibit.simdir import SimDir
from matplotlib.pyplot import plot, savefig

hor = SimDir('simulation').horizons.get_apparent_horizon(1)

cent_x = hor.ah.centroid_x

vel_x = cent_x.differentiated()

plot(vel_x)

savefig('plot.pdf')
```

EXAMPLE: EXTRACTING THE (2,2) GRAVITATIONAL-WAVE STRAIN

```python
from kuibit.simdir import SimDir

radius = 100
detector = SimDir('simulation').gravitationalwaves[radius]

# Fixed frequency integration, with window
strain = detector.get_strain_lm(2, 2, # multipole (2,2)
                                100,  # pcut
                                0.1,  # alpha in the Tukey window
                                window_function='tukey')

strain.save('strain.dat')
```

EXAMPLE: EXTRACTING THE GRAVITATIONAL-WAVE STRAIN OBSERVED BY VIRGO

```
from kuibit.simdir import SimDir

detector = SimDir('simulation').gravitationalwaves[100]

# Summing all (l, m) with spin-weighted spherical harmonics accounting
# for detector location on Earth and geometry
(detector.get_observed_strain('47', # Right ascension in degrees
                              '32', # Declination in degrees
                              '2015-09-14 09:50:45' # UTC time
                               100,   # pcut
                               0.1,   # alpha in the Tukey window
                               window_function='tukey').virgo)
                              .save('strain.dat')
```

EXAMPLE: 3D CONTOUR PLOT

```python
from kuibit.simdir import SimDir
from mayavi import mlab

res, xmax = 300, 100

rho = (SimDir(".").gf.xyz['rho_b'][0]
       .to_UniformGridData([res, res, res],
       [-xmax, -xmax, -xmax],
       [xmax, xmax, xmax])
       .log10())

mlab.contour3d(*rho.coordinates_from_grid(as_same_shape=True),
               rho.data,
               transparent=True)
```

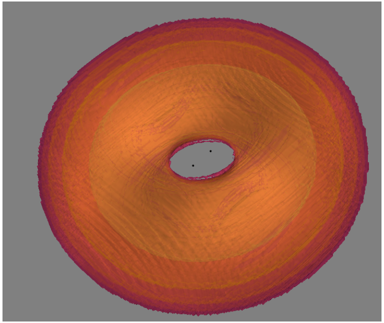
EXAMPLE: 3D CONTOUR PLOT

```
from kuibit.simdir import SimDir
from mayavi import mlab

res, xmax = 300, 100

rho = (SimDir(".").gf.xyz['rho_b'][0]
       .to_UniformGridData([res, res, res],
       [-xmax, -xmax, -xmax],
       [xmax, xmax, xmax])
       .log10())

mlab.contour3d(*rho.coordinates_from_grid(as_same_shape=True),
               rho.data,
               transparent=True)
```



MY SUGGESTION ON HOW TO GET STARTED (= EXAMPLES AS SOLVED PROBLEMS)

1. Consider one of your simulations
2. Install `kuibit`: `pip install kuibit`
3. Download examples: `curl -O github.com/sbozzolo/kuibit/releases/latest/download/examples.tar.gz`
4. Make some plots with the examples, and try to reproduce them from scratch
5. In this, read documentation and tutorials at **sbozzolo.github.io/kuibit**
6. Report problems on GitHub/via email/on Telegram (**t.me/kuibit**)

CALL FOR CONTRIBUTIONS

- `kuibit` is a great framework to make your codes available to the entire community
- Openly developed, accessible, well-commented, easy-to-extend
- Great learning opportunity!
- **github.com/Sbozzolo/kuibit**

- `kuibit` is published in the Journal of Open-Source Software
- Telegram user group/support/announcements at **t.me/kuibit**
- Feel free to reach me at `gabrielebozzola@email.arizona.edu`
- A *kuibit* is a Tohono O'odham stick to pluck Saguaro's fruit

Harvest the fruit of your `Cactus` simulations with `kuibit`!

Credits: Wolfgang Kastaun, NumPy, SciPy, h5py, TACC, NSF, NASA