A COMMUNITY-FRIENDLY PYTHON TOOL TO ANALYZE EINSTEIN TOOLKIT SIMULATIONS

Introducing `kuibit`

February 4, 2021

Gabriele Bozzola

Department of Astronomy and Steward Observatory,
University of Arizona

Part 1: Overview and motivation

KUIBIT IS A PYTHON LIBRARY FOR POST-PROCESSING SIMULATIONS

- At first order, reimplementation of Kastaun's `PostCactus`
- Support for
  - 1D, 2D, 3D, HDF5 and ASCII grid data
  - timeseries, frequency series (`CarpetIOASCII`)
  - gravitational waves with `WeylScal4` (energy, angular momenta, mismatch, extrapolation to infinity, ...)
  - detector sensitivity curves
  - unit conversion
  - apparent horizons and quasi-local measures
- Take care of all the low-level details

TEASER: COMPUTE EQUATION VIOLATION

Problem: you output V^i in 3D HDF5 files from MPI run, compute the maximum violation of $\nabla^2 V^x + x\,\partial_i V^i = 0$ as a function of the iteration

TEASER: COMPUTE EQUATION VIOLATION

Problem: you output V^i in 3D HDF5 files from MPI run, compute the maximum violation of $\nabla^2 V^x + x\,\partial_i V^i = 0$ as a function of the iteration

```
def violation(path_sim_data, it):
    gfs = SimDir(path_sim_data).gridfunctions.xyz

    V = gfs['Vx'][it], gfs['Vy'][it], gfs['Vx'][it]

    laplacian_Vx = sum(V[0].gradient(order=2))
    div_V = sum(V[i].partial_derived(i) for i in range(3))

    eq = laplacian_Vx + dx_div_V * V[0].coordinates[0]
    return eq.abs_max()
```

KUIBIT, A CODE FOR THE COMMUNITY. MY GOALS:

`kuibit` is built form the ground-up to be used and extended by others

KUIBIT, A CODE FOR THE COMMUNITY. MY GOALS:

`kuibit` is built form the ground-up to be used and extended by others

Users

Newcomer-friendly
Workflow-agnostic
Hiding technical details
Lower entry barrier
Reduce friction to do science

KUIBIT, A CODE FOR THE COMMUNITY. MY GOALS:

`kuibit` is built form the ground-up to be used and extended by others

Users

Newcomer-friendly
Workflow-agnostic
Hiding technical details
Lower entry barrier
Reduce friction to do science

Developers

Easy to extend
Well-commented code
Openly developed

KUIBIT, A CODE FOR THE COMMUNITY. MY GOALS:

`kuibit` is built form the ground-up to be used and extended by others

Users

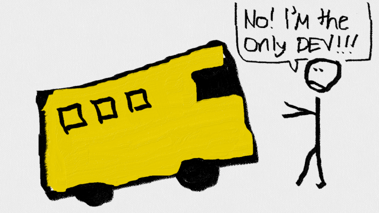
Newcomer-friendly
Workflow-agnostic
Hiding technical details
Lower entry barrier
Reduce friction to do science

Developers

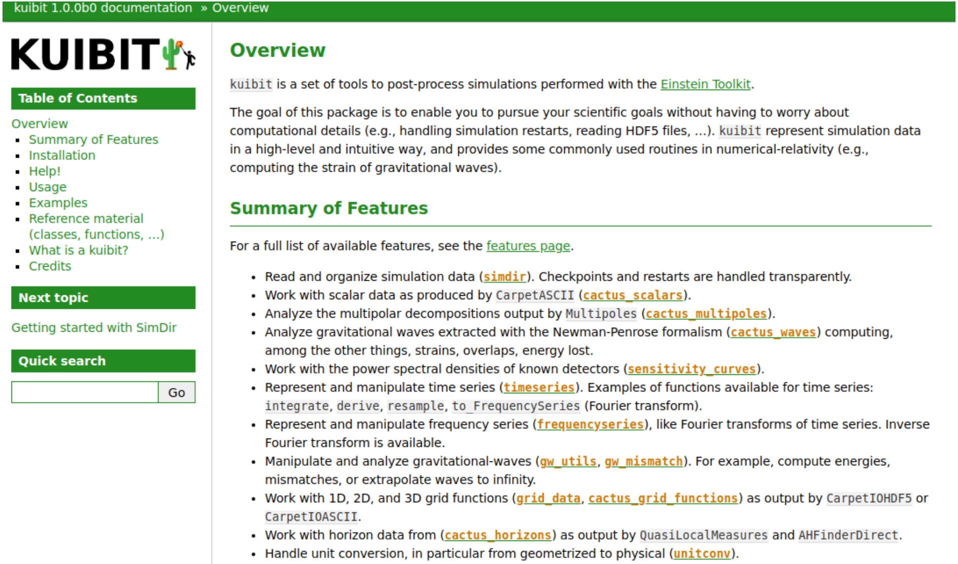
Easy to extend
Well-commented code
Openly developed

Maintainers

Reduce burden



KUIBIT IS DOCUMENTED



KUIBIT

Table of Contents

Overview

- Summary of Features
- Installation
- Help!
- Usage
- Examples
- Reference material (classes, functions, …)
- What is a kuibit?
- Credits

Next topic

Getting started with SimDir

Quick search

Overview

`kuibit` is a set of tools to post-process simulations performed with the Einstein Toolkit.

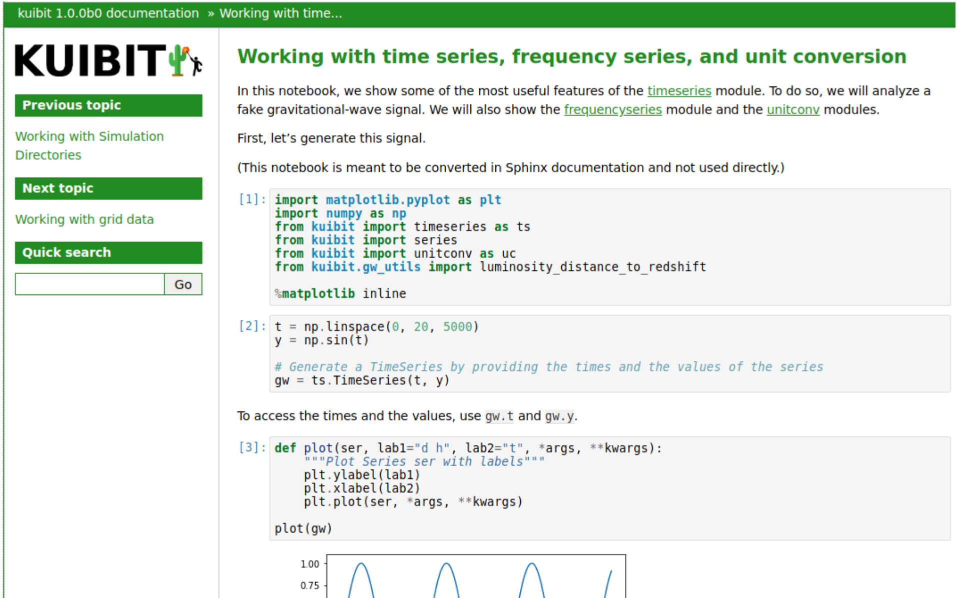
The goal of this package is to enable you to pursue your scientific goals without having to worry about computational details (e.g., handling simulation restarts, reading HDF5 files, …). `kuibit` represent simulation data in a high-level and intuitive way, and provides some commonly used routines in numerical-relativity (e.g., computing the strain of gravitational waves).

Summary of Features

For a full list of available features, see the features page.

- Read and organize simulation data (`simdir`). Checkpoints and restarts are handled transparently.
- Work with scalar data as produced by `CarpetASCII` (`cactus_scalars`).
- Analyze the multipolar decompositions output by `Multipoles` (`cactus_multipoles`).
- Analyze gravitational waves extracted with the Newman-Penrose formalism (`cactus_waves`) computing, among the other things, strains, overlaps, energy lost.
- Work with the power spectral densities of known detectors (`sensitivity_curves`).
- Represent and manipulate time series (`timeseries`). Examples of functions available for time series: `integrate`, `derive`, `resample`, `to_FrequencySeries` (Fourier transform).
- Represent and manipulate frequency series (`frequencyseries`), like Fourier transforms of time series. Inverse Fourier transform is available.
- Manipulate and analyze gravitational-waves (`gw_utils`, `gw_mismatch`). For example, compute energies, mismatches, or extrapolate waves to infinity.
- Work with 1D, 2D, and 3D grid functions (`grid_data`, `cactus_grid_functions`) as output by `CarpetIOHDF5` or `CarpetIOASCII`.
- Work with horizon data from (`cactus_horizons`) as output by `QuasiLocalMeasures` and `AHFinderDirect`.
- Handle unit conversion, in particular from geometrized to physical (`unitconv`).

THERE ARE SIMPLE TUTORIALS



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]:

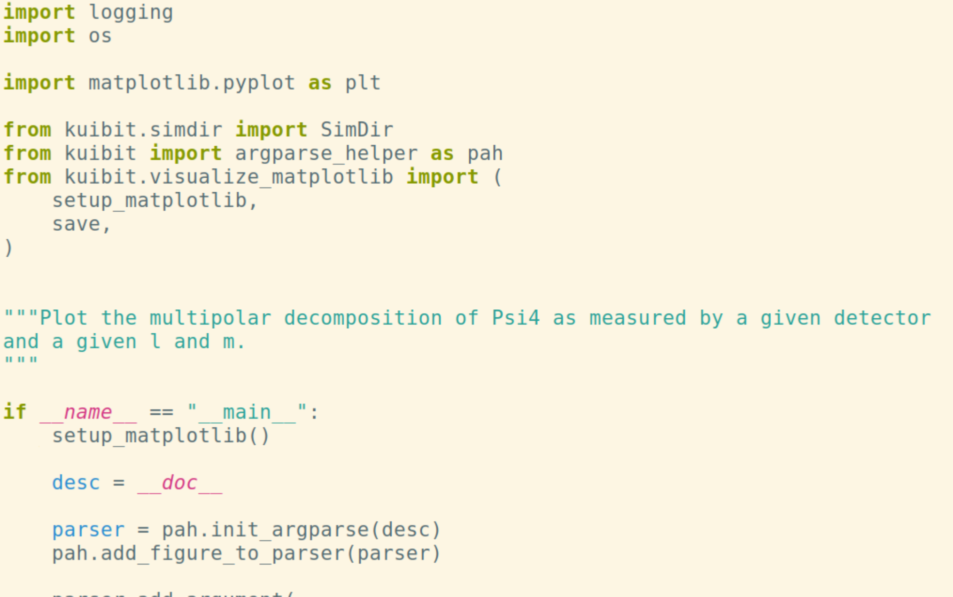
```
t = np.linspace(0, 20, 5000)
y = np.sin(t)

# 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`.

[3]:

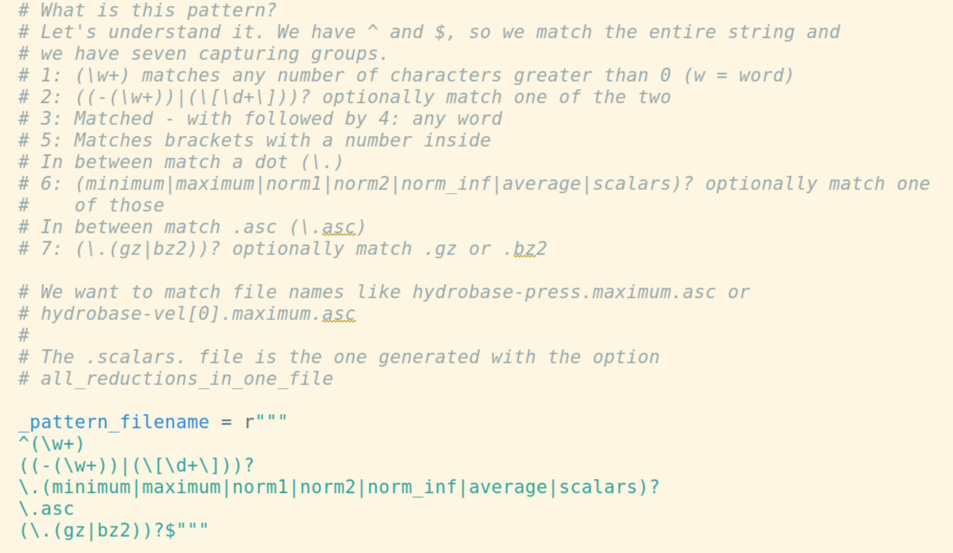
```
def plot(ser, lab1="d h", lab2="t", *args, **kwargs):
    """Plot Series ser with labels"""
    plt.ylabel(lab1)
    plt.xlabel(lab2)
    plt.plot(ser, *args, **kwargs)

plot(gw)
```

THERE ARE REAL-WORLD EXAMPLES



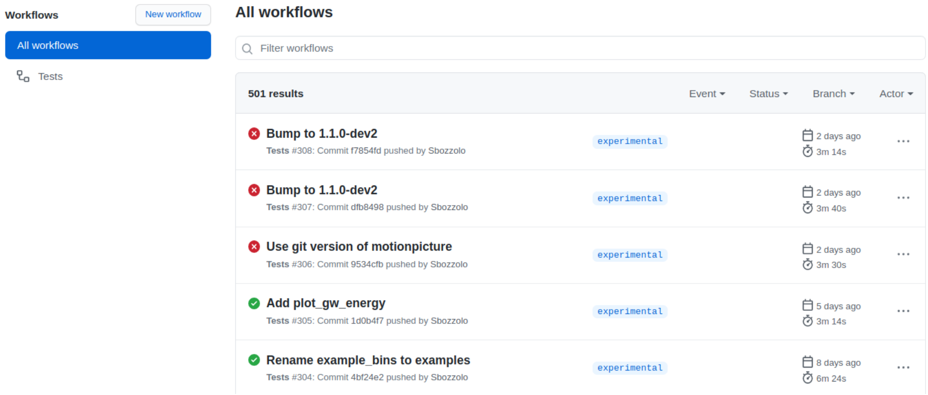
```python
import logging
import os

import matplotlib.pyplot as plt

from kuibit.simdir import SimDir
from kuibit import argparse_helper as pah
from kuibit.visualize_matplotlib import (
    setup_matplotlib,
    save,
)


"""Plot the multipolar decomposition of Psi4 as measured by a given detector
and a given l and m.
"""

if __name__ == "__main__":
    setup_matplotlib()

    desc = __doc__

    parser = pah.init_argparse(desc)
    pah.add_figure_to_parser(parser)

    parser.add_argument(
```

KUIBIT IS THOROUGHLY COMMENTED



```
# What is this pattern?
# Let's understand it. We have ^ and $, so we match the entire string and
# we have seven capturing groups.
# 1: (\w+) matches any number of characters greater than 0 (w = word)
# 2: ((-(\w+))|(\[\d+\]))? optionally match one of the two
# 3: Matched - with followed by 4: any word
# 5: Matches brackets with a number inside
# In between match a dot (\.)
# 6: (minimum|maximum|norm1|norm2|norm_inf|average|scalars)? optionally match one
#    of those
# In between match .asc (\.asc)
# 7: (\.(gz|bz2))? optionally match .gz or .bz2

# We want to match file names like hydrobase-press.maximum.asc or
# hydrobase-vel[0].maximum.asc
#
# The .scalars. file is the one generated with the option
# all_reductions_in_one_file

_pattern_filename = r"""
^(\w+)
((-(\w+))|(\[\d+\]))?
\.(minimum|maximum|norm1|norm2|norm_inf|average|scalars)?
\.asc
(\.(gz|bz2))?$"""
```

THERE ARE UNIT-TESTS AND CI



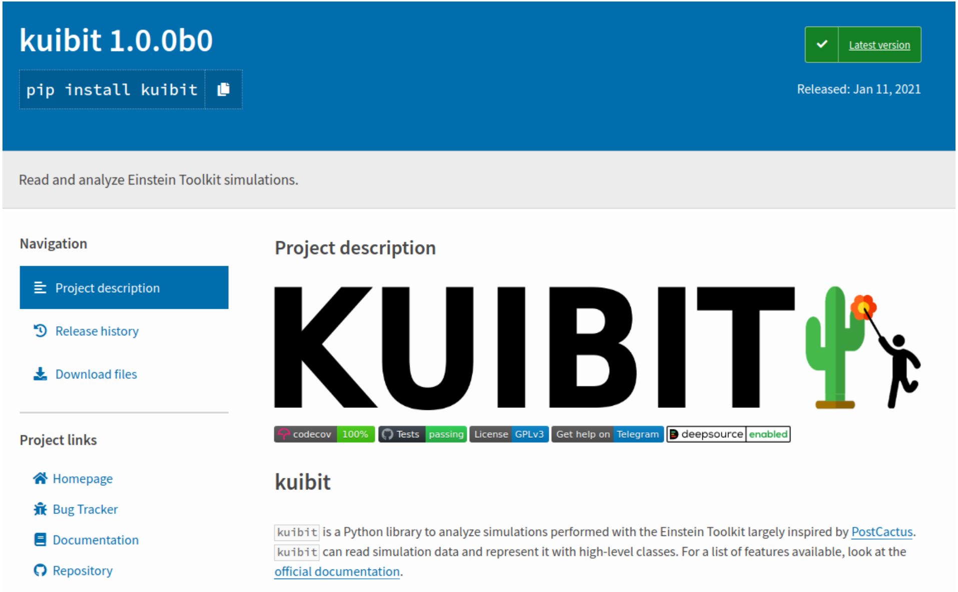
All workflows

Filter workflows

501 results

	Run	Branch	When	Duration
✗	**Bump to 1.1.0-dev2** **Tests** #308: Commit f7854fd pushed by Sbozzolo	`experimental`	2 days ago	3m 14s
✗	**Bump to 1.1.0-dev2** **Tests** #307: Commit dfb8498 pushed by Sbozzolo	`experimental`	2 days ago	3m 40s
✗	**Use git version of motionpicture** **Tests** #306: Commit 9534cfb pushed by Sbozzolo	`experimental`	2 days ago	3m 30s
✓	**Add plot_gw_energy** **Tests** #305: Commit 1d0b4f7 pushed by Sbozzolo	`experimental`	5 days ago	3m 14s
✓	**Rename example_bins to examples** **Tests** #304: Commit 4bf24e2 pushed by Sbozzolo	`experimental`	8 days ago	6m 24s

KUIBIT IS BUILT WITH MODERN TOOLS



kuibit 1.0.0b0

```
pip install kuibit
```

✓ Latest version

Released: Jan 11, 2021

Read and analyze Einstein Toolkit simulations.

Navigation

- Project description
- Release history
- Download files

Project links

- Homepage
- Bug Tracker
- Documentation
- Repository

Project description

codecov 100% | Tests passing | License GPLv3 | Get help on Telegram | deepsource enabled

kuibit

`kuibit` is a Python library to analyze simulations performed with the Einstein Toolkit largely inspired by PostCactus. `kuibit` can read simulation data and represent it with high-level classes. For a list of features available, look at the official documentation.

Part 2: (Some) capabilities and examples

KUIBIT HAS THREE GROUPS OF MODULES

Objects

`TimeSeries`

`FrequencySeries`

`UniformGridData`

`HierarchicalGridData`

...

KUIBIT HAS THREE GROUPS OF MODULES

Objects

TimeSeries
FrequencySeries
UniformGridData
HierarchicalGridData
...

Readers

SimDir
HorizonsDir
MultipolesDir
GravitationalWavesDir
ScalarsDir
GridFunctionsDir
...

KUIBIT HAS THREE GROUPS OF MODULES

Objects

TimeSeries
FrequencySeries
UniformGridData
HierarchicalGridData
…

Readers

SimDir
HorizonsDir
MultipolesDir
GravitationalWavesDir
ScalarsDir
GridFunctionsDir
…

Utilities

gw_mismatch
sYlm
sensitivity_curves
…

UTILITIES (MOSTLY GRAVITATIONAL-WAVE STUFF AT THE MOMENT)

Convenience functions and useful routines:

- → `gw_utils` (e.g., `luminosity_distance_to_redshift`, `antenna_pattern`)
- → `unitconv` (e.g., from geometrized to physical and viceversa)
- → `gw_mismatch`
- → `sensitivity_curves` (LISA, aLIGO, CE, ET, ...)

Under development (`experimental` branch):

- → `argparse_helper`
- → `visualize_matplotlib`
- → `visualize_mayavi`

OBJECTS (TIME AND FREQUENCY SERIES AND GRID DATA)

- → Support natively all mathematical operations (e.g. `ts1 + np.sin(ts2)**3` (if it makes sense)
- → Complex or real
- → Are callable `ts(10)` (internally using configurable splines)
- → Have several useful methods (e.g., cropping, Fourier transform, resampling, integrate, derive, ...)

OBJECTS (TIME AND FREQUENCY SERIES AND GRID DATA)

- Support natively all mathematical operations (e.g. `ts1 + np.sin(ts2)**3` (if it makes sense)
- Complex or real
- Are callable `ts(10)` (internally using configurable splines)
- Have several useful methods (e.g., cropping, Fourier transform, resampling, integrate, derive, …)
- `*Series` support native plotting with matplotlib (`plt.plot(ts)`)

OBJECTS (TIME AND FREQUENCY SERIES AND GRID DATA)

- → Support natively all mathematical operations (e.g. `ts1 + np.sin(ts2)**3` (if it makes sense)
- → Complex or real
- → Are callable `ts(10)` (internally using configurable splines)
- → Have several useful methods (e.g., cropping, Fourier transform, resampling, integrate, derive, ...)
- → `*Series` support native plotting with matplotlib (`plt.plot(ts)`)
- → `HierarchicalGridData` is essentially a collection of `UniformGridData`
- → Retain information from simulation (e.g., refinement level number, iteration number)
- → `HierarchicalGridData` cannot be visualized directly and have to be resampled to `UniformGridData`

READERS DEAL WITH THE OUTPUT MESS AND PRESENT US WITH AN OBJECT

Readers:

- Find the files associated to what you asked
- Deal with reading (e.g., HDF5 files, compressed files, reading correct column)
- Clean up the data (e.g., simulation restarts)
- Are nested with usually three “levels”

`SimDir`	Main point of entry (find all the files)
`*Dir (e.g., GridFunctionsDir)`	Process files from `SimDir`
`All* (e.g., AllGridFunctions)`	Organizes in the various variables
`One* (e.g., OneGridFunction)`	Has one variable (usually indexed by iterations)

All are dictionary-like that you can print, or get keys, or access with attributes.

EXAMPLE: PLOT FOURIER AMPLITUDE OF MAX(RHO) 1

```
from kuibit.simdir import SimDir
s = SimDir('.')     # type(s) => kuibit.simdir.SimDir
ts = s.timeseries   # type(ts) => kuibit.cactus_scalars.ScalarsDir
maxx = ts['max']    # type(maxx) => kuibit.cactus_scalars.AllScalars
rho = maxx['rho']   # type(rho) => kuibit.timeseries.TimeSeries

# print(maxx) => Available maximum timeseries: ['rho_b', 'M1', 'H', 'M3',
```

What happened here? `kuibit` has

1. Scanned and organized all the available files in `.`
2. Identified what files contain scalar data
3. Identified what reductions are available
4. Identified what variables are available
5. Cleaned-up simulation restarts

EXAMPLE: PLOT FOURIER AMPLITUDE OF MAX(RHO) 2

```python
import matplotlib.pyplot as plt
from kuibit.simdir import SimDir
rho = SimDir('.').ts.max['rho']

# Preprocessing
rho.crop(0, 10)  # Edit in-place
rho_w = rho.tukey_windowed(0.1)  # Return a new object

plt.plot(abs(rho_w.to_FrequencySeries()))

# Other useful methods:
# derive, integrate, band_pass, crop, smooth, window,
# resample, redshift, and all the mathematical operations
```

Only four lines of code that work on any simulation!

EXAMPLE: SNR FOR LISA

```python
from kuibit.simdir import SimDir
from kuibit.sensitivity_curves import Sn_LISA

detectors = SimDir('.').gravitationalwaves
radius = 91.2

complex_strain = detectors[radius].get_strain_lm(2, 2, pcut=120)
strain_f = complex_strain.to_FrequencySeries()

SNRsq = strain_f.inner_product(strain_f, noises=Sn_LISA(strain_f.f),
                               fmin=20)
```

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

```
b = SimDir('.').gridfunctions.xyz['b'][0]
P = SimDir('.').gridfunctions.xyz['P'][0]
# type(P) => kuibit.grid_data.HierarchicalGridData

ratio = b*b/P

ratio_uniform = ratio.to_UniformGridData([1000, 1000],
                                         x0=[-10, 10], x1=[10,10],
                                         resample=True)
# type(ratio_uniform) => kuibit.grid_data.UniformGridData

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

plt.contourf(*ratio_on_z2.coordinates_from_grid(as_meshgrid=True),
             ratio_on_z2.data_xyz)
```

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

What happened here? `kuibit` has

1. Scanned, organized, identified all ASCII and HDF5 grid files
2. Read (preferably) 3D HDF5 at given iteration
3. Read metadata from HDF5 to learn about ghost zones
4. Tried to combine different components (MPI processes) to a single one
5. Combined different variables keeping track of their definition grid
6. Resampled with trilinear interpolation AMR to uniform grid[1]
7. Extracted only the plane with $z = 2$

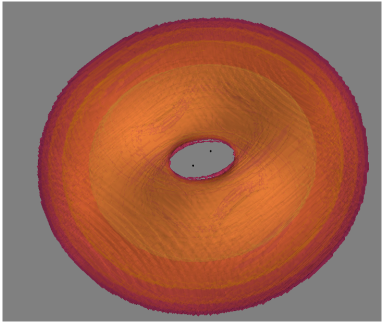
[1] Extremely high RAM consumption!

EXAMPLE: 3D CONTOUR PLOT

```
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)
```

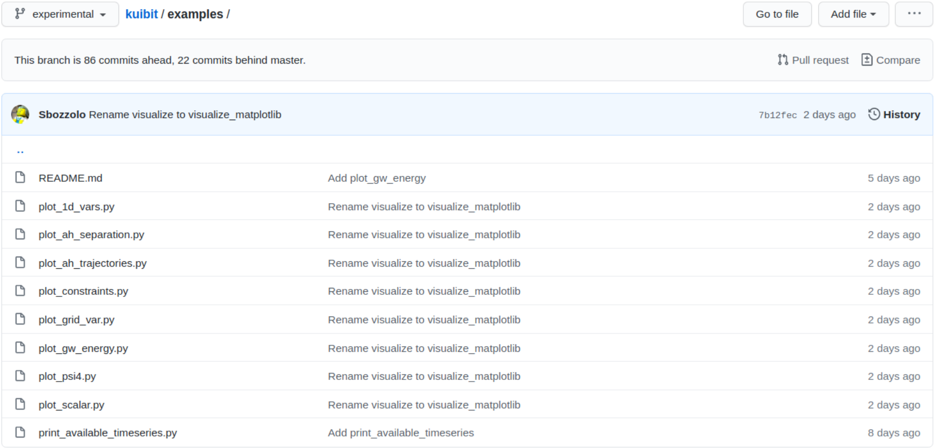
EXAMPLE: 3D CONTOUR PLOT

```
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)
```

EXPERIMENTAL BRANCH HAS MODULES FOR VISUALIZATION AND REAL EXAMPLES



kuibit / examples /

This branch is 86 commits ahead, 22 commits behind master.

Sbozzolo Rename visualize to visualize_matplotlib `7b12fec` 2 days ago

..

README.md	Add plot_gw_energy	5 days ago
plot_1d_vars.py	Rename visualize to visualize_matplotlib	2 days ago
plot_ah_separation.py	Rename visualize to visualize_matplotlib	2 days ago
plot_ah_trajectories.py	Rename visualize to visualize_matplotlib	2 days ago
plot_constraints.py	Rename visualize to visualize_matplotlib	2 days ago
plot_grid_var.py	Rename visualize to visualize_matplotlib	2 days ago
plot_gw_energy.py	Rename visualize to visualize_matplotlib	2 days ago
plot_psi4.py	Rename visualize to visualize_matplotlib	2 days ago
plot_scalar.py	Rename visualize to visualize_matplotlib	2 days ago
print_available_timeseries.py	Add print_available_timeseries	8 days ago

FINAL REMARKS

- → Code needs **a lot** of testing and real-world usage
- → I haven't touched upon horizon or multipole data, but hopefully you will be able to navigate the documentation
- → Telegram user group/support at `t.me/kuibit`
- → Feel free to reach me at `gabrielebozzola@email.arizona.edu`
- → I hope `kuibit` can become officially part of `Einstein Toolkit`
- → A *kuibit* is a Tohono O'odham stick to harvest Saguaro's fruit

If we have more time

WORKING WITH HORIZONS

(This module will likely improve in the future)

```
horizons = SimDir('.').horizons
# print(horizons)
# => Horizons found 2: 2 horizons from QLM, 2 horizons from AHFinderDirect

# Access horizon with both the AH and the QLM indices
qlm_index, ah_index = 1, 2

hor = horizons[qlm_index, ah_index]
# hor contains the QLM properties
type(hor.mass) # => kuibit.timeseries.TimeSeries
# hor.ah is a dictionary with all the AH properties
print(hor.ah.mass) # => kuibit.timeseries.TimeSeries

x, y, z = hor.shape_at_iteration(0)
```