cosmogrb Documentation

J. Michael Burgess

Oct 14, 2020

Contents

cosmogrb is a package built upon popsynth to simulate GRBs from luminosity functions and various other distributions. Each GRB can be passed through an instrument's response resulting in data when can be later analyzed (preferably with 3ML. Thus, one can generate catalogs of data from theoretical assumptions an test what these assumptions lead to in terms of observation.

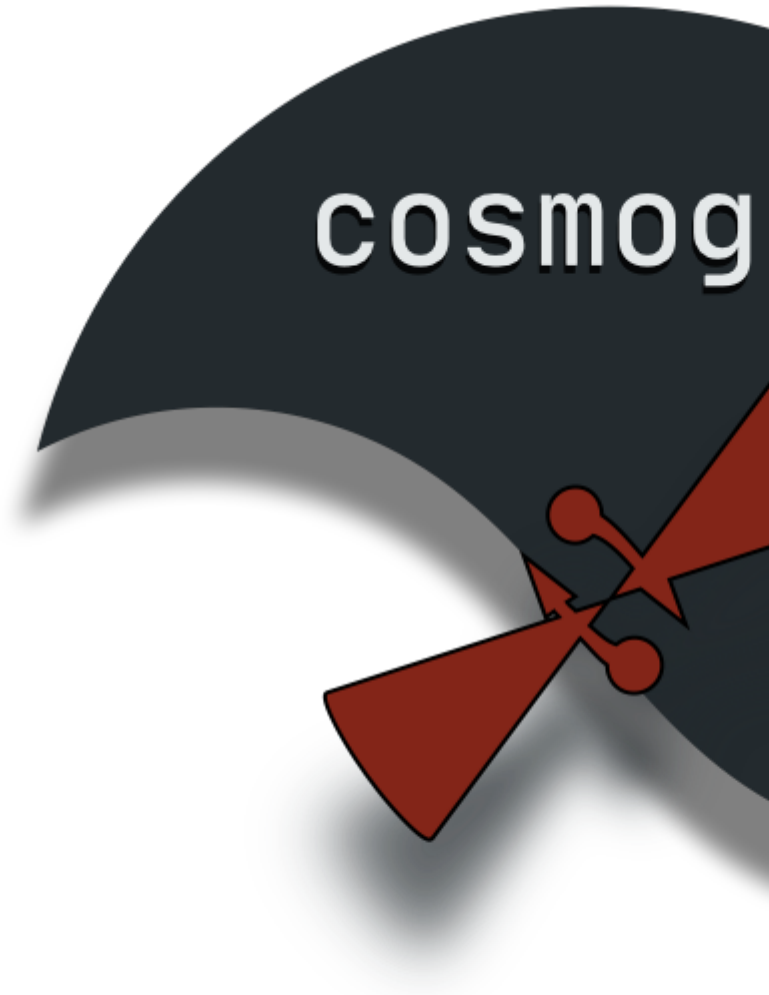
The code is currently in *alpha* so do not expect too much use out of it.

CHAPTER 1

Introduction

cosmogrb is a package built upon popsynth to simulate GRBs from luminosity functions and various other distributions. Each GRB can be passed through an instrument's response resulting in data when can be later analyzed (preferably with 3ML). Thus, one can generate catalogs of data from theoretical assumptions an test what these assumptions lead to in terms of observation.

The code is currently in *alpha* so do not expect too much use out of it.

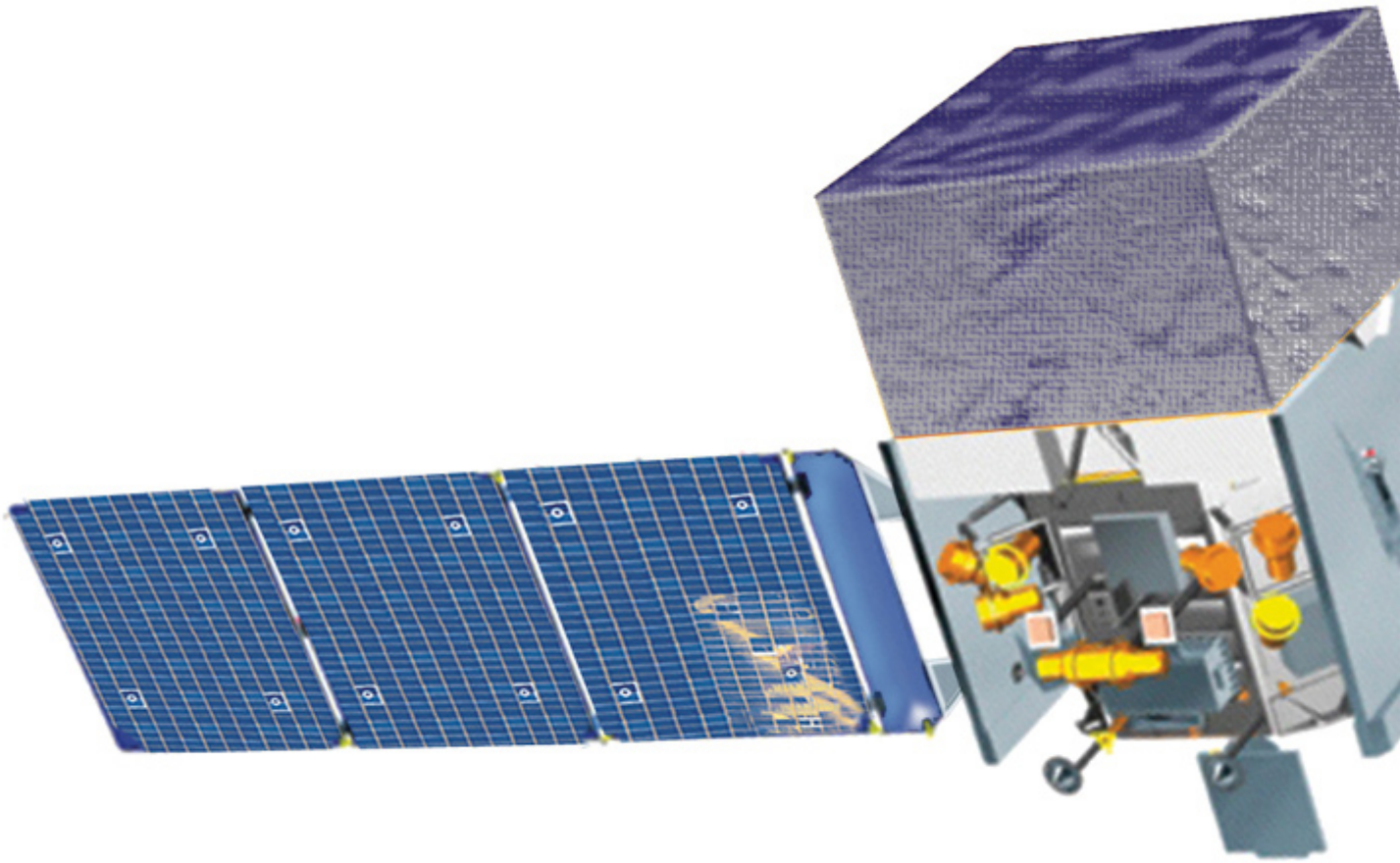


[4]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
from jupyterthemes import jtplot
plt.style.use('mike')
jtplot.style(context='talk', fscale=1, grid=False)

import cosmogrb
```

CHAPTER 2

GRBs

This section describes how to handle the low level simulation of GRBs. AS the code currently is built for simulating GRBs as observed by Fermi-GBM, we will focus our attention there. As the code expands, I will update the docs.



2.1 Instantiate the GRB with its parameters

For this example, we will create a GRB that has its flux coming from a single pulse shape that is described by a cutoff power law evolving in time.

$$F_{h\nu}(t) = K(t) \left(\frac{\nu}{\nu_0(t)} \right)^{-\alpha} \cdot \exp\left(-\frac{\nu}{\nu_0(t)} \right)$$

[2]:
```
grb = cosmogrb.gbm.GBMGRB_CPL(
    ra=312.0,
    dec=-62.0,
    z=1.0,
    peak_flux=5e-7,
    alpha=-0.66,
    ep=500.0,
    tau=2.0,
    trise=1.0,
    tdecay=1.0,
```

(continues on next page)

(continued from previous page)

```
    duration=80.0,
    T0=0.1,
)

grb.info()
```

```
                 0
name       SynthGRB
z                1
ra             312
dec            -62
duration        80
T0             0.1
```

```
                        0
peak_flux  5.000000e-07
alpha     -6.600000e-01
trise      1.000000e+00
tdecay     1.000000e+00
```

```
Empty DataFrame
Columns: [0]
Index: []
```

2.2 Examine the latent lightcurve

[3]:
```
time = np.linspace(0, 20, 500)

grb.display_energy_integrated_light_curve(time, color="#A363DE");
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-3-0737622692d6> in <module>
      1 time = np.linspace(0, 20, 500)
      2
----> 3 grb.display_energy_integrated_light_curve(time, color="#A363DE");
      4

~/coding/projects/cosmogrb/cosmogrb/grb/grb.py in display_energy_integrated_light_
↪curve(self, time, ax, **kwargs)
    143         """
    144
--> 145         list(self._lightcurves.values())[0].display_energy_integrated_light_
↪curve(
    146             time=time, ax=ax, **kwargs
    147         )

IndexError: list index out of range
```

[ ]:
```
energy = np.logspace(1, 3, 1000)

grb.display_energy_dependent_light_curve(time, energy, cmap='PRGn', lw=.25, alpha=.5)
```

2.3 Simulate the GRB

Now we can create all the light curves from the GRB. Since are not currently running a Dask server, we tell the GRB to process serially, i.e., computing each light curve one at a time.

[ ]:
```
grb.go(serial=True)
```

2.4 Save the GRB to an HDF5 file

As this is a time-consuming operation, we want to be able to save the GRB to disk. This is done by serializing all the light curves and information about the GRB into an HDF5 file.

[ ]:
```
grb.save('test_grb.h5')
```

2.5 Reload the GRB

What if want to reload the GRB? We need to create and instance of **GRBSave** from the file we just created. Notice all the information about the GRB is recovered.

[5]:
```
grb_reload = cosmogrb.GRBSave.from_file('test_grb.h5')
grb_reload.info()
```

```
                0
name      SynthGRB
z                1
ra             312
dec            -62
duration        80
T0             0.1
```

```
                        0
alpha     -6.600000e-01
peak_flux  5.000000e-07
tdecay     1.000000e+00
trise      1.000000e+00
```

2.5.1 The GRBSave contents

The stores all the information about the light curves and the instrument responses used to generate the data. Each light curve/ response pair can be accessed as keys of the **GRBSave**. Then one can easily, examine/plot/process the contents of each light curve.

[6]:
```
for key in grb_reload.keys():

    lightcurve = grb_reload[key]['lightcurve']
    lightcurve.info()
```

```
                               0
name                          b0
instrument                   GBM
tstart                      -100
```

(continues on next page)

(continued from previous page)

```
tstop                       300
time adjustment      5.76246e+08
T0                           0.1
n_counts                  190656
n_counts_source               36
n_counts_background       190660
```

```
                0
angle  106.678116
```

```
                             0
name                        b1
instrument                 GBM
tstart                    -100
tstop                      300
time adjustment      5.76219e+08
T0                           0.1
n_counts                  206738
n_counts_source               70
n_counts_background       206695
```

```
               0
angle  64.599896
```

```
                             0
name                        n0
instrument                 GBM
tstart                    -100
tstop                      300
time adjustment      5.76253e+08
T0                           0.1
n_counts                  194080
n_counts_source              136
n_counts_background       194008
```

```
                0
angle  140.423466
```

```
                             0
name                        n1
instrument                 GBM
tstart                    -100
tstop                      300
time adjustment      5.76276e+08
T0                           0.1
n_counts                  198594
n_counts_source              127
n_counts_background       198530
```

```
                0
angle  123.192564
```

```
                             0
name                        n2
instrument                 GBM
tstart                    -100
tstop                      300
time adjustment      5.76203e+08
T0                           0.1
n_counts                  203221
```

(continues on next page)

(continued from previous page)

```
n_counts_source               87
n_counts_background       203189
```

```
               0
angle  87.337115
```

```
                              0
name                          n3
instrument                   GBM
tstart                      -100
tstop                        300
time adjustment      5.76206e+08
T0                           0.1
n_counts                  200535
n_counts_source              108
n_counts_background       200460
```

```
                0
angle  120.135933
```

```
                              0
name                          n4
instrument                   GBM
tstart                      -100
tstop                        300
time adjustment      5.76229e+08
T0                           0.1
n_counts                  204014
n_counts_source               89
n_counts_background       203953
```

```
                0
angle  115.423396
```

```
                              0
name                          n5
instrument                   GBM
tstart                      -100
tstop                        300
time adjustment      5.76212e+08
T0                           0.1
n_counts                  201485
n_counts_source               47
n_counts_background       201489
```

```
               0
angle  91.866668
```

```
                              0
name                          n6
instrument                   GBM
tstart                      -100
tstop                        300
time adjustment      5.76284e+08
T0                           0.1
n_counts                  203790
n_counts_source              121
n_counts_background       203697
```

```
               0
angle  126.89814
```

```
                              0
name                         n7
instrument                  GBM
tstart                     -100
tstop                       300
time adjustment      5.76264e+08
T0                           0.1
n_counts                  205853
n_counts_source              142
n_counts_background       205745
```

```
                0
angle  150.685788
```

```
                              0
name                         n8
instrument                  GBM
tstart                     -100
tstop                       300
time adjustment      5.76264e+08
T0                           0.1
n_counts                  205156
n_counts_source              127
n_counts_background       205069
```

```
                0
angle  121.106339
```

```
                              0
name                         n9
instrument                  GBM
tstart                     -100
tstop                       300
time adjustment      5.76252e+08
T0                           0.1
n_counts                  198283
n_counts_source               85
n_counts_background       198226
```

```
                0
angle  108.143856
```

```
                              0
name                         na
instrument                  GBM
tstart                     -100
tstop                       300
time adjustment      5.76287e+08
T0                           0.1
n_counts                  198458
n_counts_source              582
n_counts_background       197918
```

```
               0
angle  57.909913
```

```
                              0
name                         nb
instrument                  GBM
tstart                     -100
tstop                       300
```

(continues on next page)

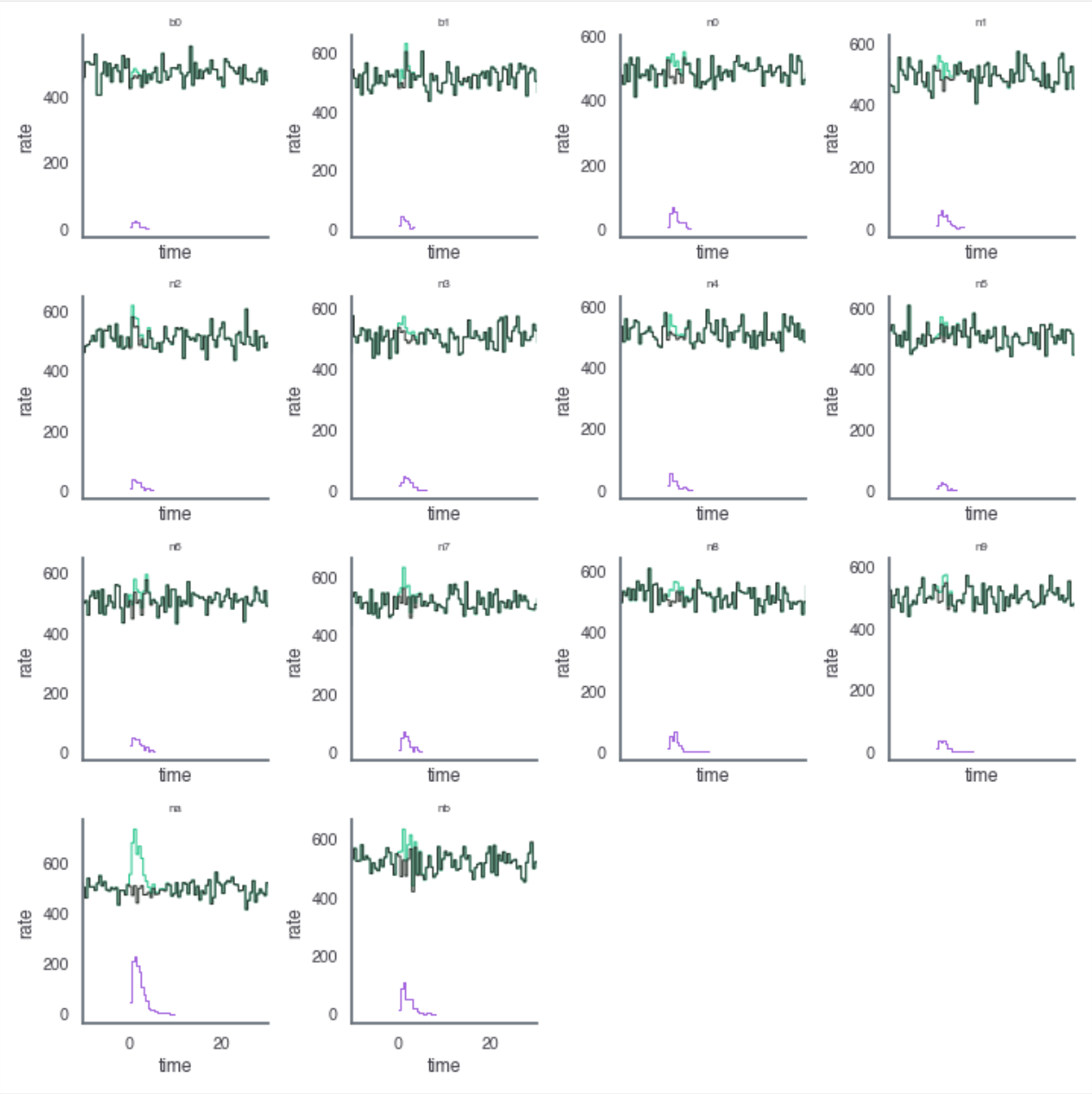
(continued from previous page)

```
time adjustment      5.76277e+08
T0                           0.1
n_counts                  206218
n_counts_source              206
n_counts_background       206091
```

```
                0
angle  77.225671
```

For example, let's look at the total, source, and background data light curves generated.

[9]:
```
fig, axes = plt.subplots(4,4,sharex=True,sharey=False,figsize=(10,10))
row=0
col = 0
for k,v  in grb_reload.items():
    ax = axes[row][col]

    lightcurve =v['lightcurve']

    lightcurve.display_lightcurve(dt=.5, ax=ax,lw=1,color='#25C68C')
    lightcurve.display_source(dt=.5,ax=ax,lw=1,color="#A363DE")
    lightcurve.display_background(dt=.5,ax=ax,lw=1, color="#2C342E")
    ax.set_xlim(-10, 30)
    ax.set_title(k,size=8)



    if col < 3:
        col+=1
    else:
        row+=1
        col=0

axes[3,2].set_visible(False)
axes[3,3].set_visible(False)
plt.tight_layout()
```



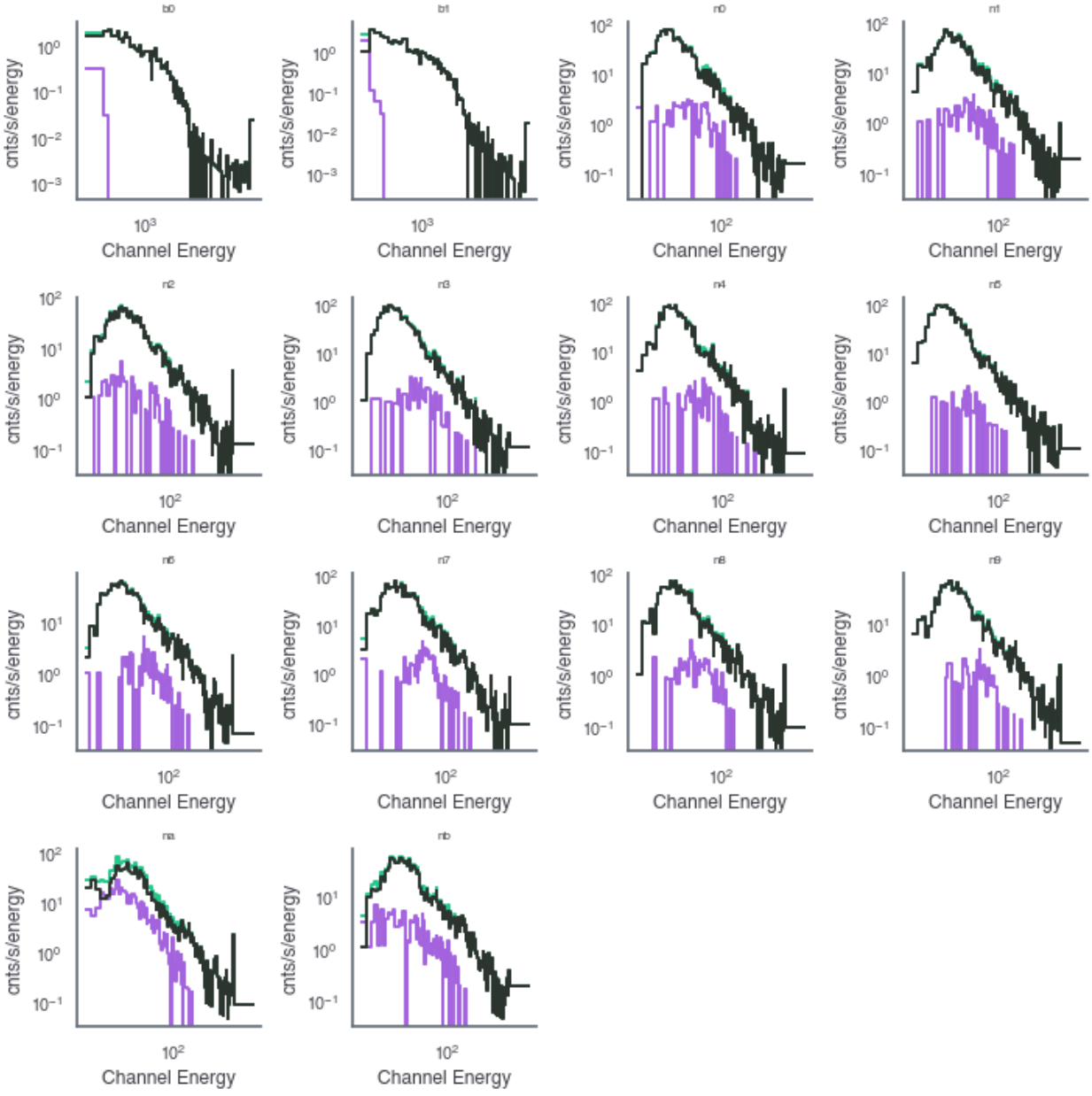
And we can look at the generated count spectra/

[10]:
```
fig, axes = plt.subplots(4,4,sharex=False,sharey=False,figsize=(10,10))
row=0
col = 0

for k, v in grb_reload.items():
    ax = axes[row][col]

    lightcurve = v['lightcurve']

    lightcurve.display_count_spectrum(tmin=0, tmax=5, ax=ax,color='#25C68C')
    lightcurve.display_count_spectrum_source(tmin=0, tmax=5, ax=ax,color="#A363DE")
    lightcurve.display_count_spectrum_background(tmin=0, tmax=5, ax=ax, color="#2C342E
↪")
```

(continues on next page)

(continued from previous page)

```
    ax.set_title(k,size=8)

    if col < 3:
        col+=1
    else:
        row+=1
        col=0

axes[3,2].set_visible(False)
axes[3,3].set_visible(False)
plt.tight_layout()
```



2.6 Convert HDF5 to standard FITS files

In the case of GBM, we can convert the saved HDF5 files into TTE files for analysis in 3ML.

[12]:
```
cosmogrb.grbsave_to_gbm_fits("test_grb.h5")
!ls SynthGRB_*
```

```
ls: SynthGRB_*: No such file or directory
```

[ ]:

CHAPTER 3

Simulating a Universe of GRBs

Now we move to the main purpose of the code which is simulating many GRBs from distribution. We will set up a toy model for demonstration. The first thing we need to do is create our population with popsynth.

[1]:
```
# Scientific libraries

import numpy as np

import matplotlib.pyplot as plt
%matplotlib notebook
from jupyterthemes import jtplot

jtplot.style(context='notebook', fscale=1, grid=False)
plt.style.use('mike')
```

3.1 Create a population of GRBs

Using popsynth, we construct a population. The source parameters to be simulated need to be generated in the population.

.. note:: In the future, a specific format for populations will be defined. This will create a much more user-friendly interface for creating populations.

[2]:
```
import popsynth
from popsynth.aux_samplers.normal_aux_sampler import NormalAuxSampler
from popsynth.aux_samplers.trunc_normal_aux_sampler import TruncatedNormalAuxSampler
from popsynth.aux_samplers.lognormal_aux_sampler import LogNormalAuxSampler
```

```
/Users/jburgess/.environs/cosmogrb/lib/python3.7/site-packages/tqdm/autonotebook.py:
↪17: TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use␣
↪`tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  " (e.g. in jupyter console)", TqdmExperimentalWarning)
```

cosmogrb requires certain parameters to be simulated in a population. We will create the auxiliary samplers to do this.

[3]:
```
class TDecaySampler(popsynth.AuxiliarySampler):
    def __init__(self):
        """
        samples the decay of the of the pulse
        """

        super(TDecaySampler, self).__init__(name="tdecay", sigma=None, observed=False)

    def true_sampler(self, size):

        t90 = 10 ** self._secondary_samplers["log_t90"].true_values
        trise = self._secondary_samplers["trise"].true_values

        self._true_values = (
            1.0 / 50.0 * (10 * t90 + trise + np.sqrt(trise) * np.sqrt(20 * t90 +␣
↪trise))
        )


class DurationSampler(popsynth.AuxiliarySampler):
    def __init__(self):
        "samples how long the pulse last"

        super(DurationSampler, self).__init__(
            name="duration", sigma=None, observed=False
        )

    def true_sampler(self, size):

        t90 = 10 ** self._secondary_samplers["log_t90"].true_values

        self._true_values = 1.5 * t90
```

Now that we have creates our extra distribution samplers, we can go ahead and create the population sampler. We will use a simple SFR like redshift distribution and a Pareto (power law) luminosity function

[4]:
```
# redshift distribution
r0_true = 3
rise_true = 1.
decay_true = 4.0
peak_true = 1.5

# the luminosity
Lmin_true = 1e51
alpha_true = 1.5
r_max = 7.0


pop_gen = popsynth.populations.ParetoSFRPopulation(
    r0=r0_true,
    rise=rise_true,
    decay=decay_true,
    peak=peak_true,
    Lmin=Lmin_true,
    alpha=alpha_true,
    r_max=r_max,
)
```

Now set up and add all the auxiliary samplers

[5]:
```
ep = LogNormalAuxSampler(mu=300.0, tau=0.5, name="log_ep", observed=False)
alpha = TruncatedNormalAuxSampler(
    lower=-1.5, upper=0.1, mu=-1, tau=0.25, name="alpha", observed=False
)
tau = TruncatedNormalAuxSampler(
    lower=1.5, upper=2.5, mu=2, tau=0.25, name="tau", observed=False
)
trise = TruncatedNormalAuxSampler(
    lower=0.01, upper=5.0, mu=1, tau=1.0, name="trise", observed=False
)
t90 = LogNormalAuxSampler(mu=10, tau=0.25, name="log_t90", observed=False)

tdecay = TDecaySampler()
duration = DurationSampler()
tdecay.set_secondary_sampler(t90)
tdecay.set_secondary_sampler(trise)

duration.set_secondary_sampler(t90)


pop_gen.add_observed_quantity(ep)
pop_gen.add_observed_quantity(tau)
pop_gen.add_observed_quantity(alpha)
pop_gen.add_observed_quantity(tdecay)
pop_gen.add_observed_quantity(duration)
```

```
registering auxilary sampler: log_ep
registering auxilary sampler: tau
registering auxilary sampler: alpha
registering auxilary sampler: tdecay
registering auxilary sampler: duration
```

We sample the population. It is important to specify that there is no selection as we will implement the full trigger later.

[6]:
```
pop = pop_gen.draw_survey(no_selection=True, boundary=1e-2)
```

```
The volume integral is 114.333557
```

```
HBox(children=(IntProgress(value=0, description='Drawing distances', max=104,␣
↪style=ProgressStyle(description_...
```

```

Expecting 104 total objects
Sampling: log_ep
Sampling: tau
Sampling: alpha
Sampling: tdecay
tdecay is sampling its secondary quantities
Sampling: log_t90
Sampling: trise
Sampling: duration
duration is sampling its secondary quantities
Applying soft boundary
```

```
HBox(children=(IntProgress(value=0, description='samping detection probability',␣
↪max=104, style=ProgressStyle(...
```

```
No Selection! Added back all objects
NO HIDDEN OBJECTS
Deteced 104 objects or to a distance of 4.29
```

[7]:
```
pop.display_obs_fluxes_sphere(size=1., cmap='cividis', background_color='black')
```

```
VBox(children=(Figure(camera=PerspectiveCamera(fov=45.0, position=(0.0, 0.0, 2.0),␣
↪quaternion=(0.0, 0.0, 0.0, ...
```

```
ToggleButton(value=False, description='Rotate')
```

We save the population to a file for reloading later

[8]:
```
pop.writeto("population.h5")
```

3.2 Simulation the population with cosmogrb

We will use dask to handle the parallel generation of all the GRBs in the universe. The code can be run serially as well, but it is possible that the time will be equaivalent to the actual age of the Universe.

[10]:
```
from dask.distributed import LocalCluster, Client
from cosmogrb.instruments.gbm import GBM_CPL_Universe
```

[11]:
```
cluster = LocalCluster(n_workers=24)
client = Client(cluster)
client
```

[11]:
```
<Client: 'tcp://127.0.0.1:45077' processes=24 threads=96, memory=201.45 GB>
```

Now we pass the population file to a specialized GBM observed universe. Here GRBs have simple FRED-like pulses and evolving peak νF_ν energies. We need to specify as path to save all the generated files.

[11]:
```
universe = GBM_CPL_Universe('population.h5', save_path="/data/jburgess/cosmogrb/")
```

Pass the client to the **go** function and wait while your GRBs go off and have their data recorded by GBM... or whatever instrument is included in the package next.

[12]:
```
universe.go(client)
```

When we are done, we will want to save the meta information (file locations, etc) to a file to recover later. We call this object a **Survey**.

[ ]:
```
.. note::
   In the future, there will be the option to place the entire simulation in one␣
↪large file to avoid having to keep track of file locations. For now, if once␣
↪changes the location of the files, further processing of the survey will not be␣
↪possible
```

[17]:
```
universe.save('survey.h5')
```

[12]:
```
client.close()
cluster.close()
```

We can now shut off our cluster.

3.3 Processing a Survey

Creating GRBs does not automatically run an instrument's detection algorithm on them as we want to store the raw data and possibly analyze *why* a GRB was not detected as a function of its latent parameters. This is typically an expensive process, so we will again use dask to farm out jobs.

[13]:
```
cluster = LocalCluster(n_workers=24)
client = Client(cluster)
client
```

[13]:
```
<Client: 'tcp://127.0.0.1:37589' processes=24 threads=96, memory=201.45 GB>
```

We must import the trigger analysis class specific to GBM for the survey. An error will occur if we use the wring class.

[3]:
```
from glob import glob
from cosmogrb.universe.survey import Survey
from cosmogrb.instruments.gbm.gbm_trigger import GBMTrigger
```

```
You do not have threeML installed
```

```
/home/jburgess/.venv/cosmogrb/lib/python3.6/site-packages/popsynth-0.3.3-py3.6.egg/
↪popsynth/distribution.py:7: TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm`␣
↪in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter␣
↪console)
  from tqdm.autonotebook import tqdm as progress_bar
```

The survey can be reloaded from the file we saved. All the information about the GRBs which were simulated is contained in the file. We then process the triggers. We selected a trigger threshold of 4.5 σ to mimic the true GBM trigger. Afterwards, we will save the survey back to a file, this time with the processed trigger information.

[6]:
```
survey = Survey.from_file('survey.h5')
survey.process(GBMTrigger, client=client, threshold=4.5)
survey.write('survey.h5')
```

Upon reloading the survey, we can verify that, indeed, GBM triggered on some of the event!

[4]:
```
survey = Survey.from_file('survey.h5')
survey.info()
```

```
                0
n_grbs        104
is_processed  True
n_detected      62
```

We can can examine one of the detected GRBs

[18]:
```
survey['SynthGRB_1'].detector_info.info()
```

```
                      0
name          SynthGRB_1
is_detected         True
```

```
                                                                    0
triggered_detectors                                    [b'n2', b'n5']
triggered_time_scales                                  [0.016, 0.016]
triggered_times        [3.339550858072471e-11, 3.339550858072471e-11]
```

[20]:
```
fig, axes = plt.subplots(4,4,sharex=True,sharey=False,figsize=(10,10))
row=0
```

(continues on next page)

(continued from previous page)

```
col = 0
for k,v  in survey['SynthGRB_1'].grb.items():
    ax = axes[row][col]

    lightcurve =v['lightcurve']

    lightcurve.display_lightcurve(dt=.5, ax=ax,lw=1,color='#25C68C')
    lightcurve.display_source(dt=.5,ax=ax,lw=1,color="#A363DE")
    lightcurve.display_background(dt=.5,ax=ax,lw=1, color="#2C342E")
    ax.set_xlim(-10, 30)
    ax.set_title(k,size=8)



    if col < 3:
        col+=1
    else:
        row+=1
        col=0

axes[3,2].set_visible(False)
axes[3,3].set_visible(False)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

as well as examining one that was not detected:

[21]:
```
survey['SynthGRB_0'].detector_info.info()
```

```
                      0
name          SynthGRB_0
is_detected         False
```

```
                                                                                0
triggered_detectors                                        [b'na', b'n7', b'n6']
triggered_time_scales                                      [0.016, 0.016, 0.016]
triggered_times          [164.20800000008825, 170.0800000000902, 3.4400...
```

[22]:
```
fig, axes = plt.subplots(4,4,sharex=True,sharey=False,figsize=(10,10))
row=0
col = 0
for k,v  in survey['SynthGRB_0'].grb.items():
    ax = axes[row][col]

    lightcurve =v['lightcurve']

    lightcurve.display_lightcurve(dt=.5, ax=ax,lw=1,color='#25C68C')
    lightcurve.display_source(dt=.5,ax=ax,lw=1,color="#A363DE")
    lightcurve.display_background(dt=.5,ax=ax,lw=1, color="#2C342E")
    ax.set_xlim(-10, 30)
    ax.set_title(k,size=8)



    if col < 3:
        col+=1
```

(continues on next page)

(continued from previous page)

```
    else:
        row+=1
        col=0

axes[3,2].set_visible(False)
axes[3,3].set_visible(False)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

CHAPTER 4

Indices and tables

- genindex
- modindex
- search