# GECAMTools instruction

## introduction

GECAMTools is an Application Programming Interface (API) for GECAM data. Implemented via python3, this tool is designed to allow the general users to incorporate analysis of GECAM data into their own scripts and workflows without having to sweat very many details. To this end, the tool has a high-level API layer that allows user to read, simplify and visualize GECAM data with just a few lines of code. For experts and users who wish to have fine-grained control over various aspects of their analysis, this tool also provides a lower-level API layer. This tool currently implements the basic functions of data analysis, including time conversion, viewing light curves, viewing energy spectrums, generating response files and generating energy spectrum files for fitting the spectrum.

In addition, the versatility of GECAMTools is also considered. The data interface of this tool is currently applicable to events data, many functions could be extended to data from other instruments through inherited structures. Even if the data file definition is different from the GECAM file, redefining the loading interface of the file will allow subsequent analysis using this tool.

### Github: link

### developer：

Peng Zhang(IHEP)、Wangchen Xue(IHEP)、Yanqiu Zhang(IHEP)、Chao Zheng(IHEP) and Shaolin Xiong*(IHEP)

### Email:

Shaolin Xiong*(xiongsl@ihep.ac.cn)
Peng Zhang(zhangp97@ihep.ac.cn),Questions about the installation and use of GECAMTools can be directed to this e-mail address for feedback.

### GECAM Introduction and Data Publishing Site： link

## Installation Instructions

1. Required environment
   1.1 System environment: windows, linux, mac
   1.2 python environment: python version >= 3.6 (not recommended to install python 3.10 and above)

   windows 10: python=3.8.5 successfully installed.
   linux (Centos): python=3.9.5 installed successfully.
   mac (M1 chip): test python=3.9.16, first use conda install astropy==4.3.1, then use pip install gecamTools.zip as normal to install successfully.

2. Installation process
   2.1 Download the source program
   gecamTools-v**.zip

2.2 Installation

2.2.1 Use only the basic functions (i.e., no need to generate response matrix).

Use pip to install the source code, which automatically installs GECAMTools and related dependencies

`pip install gecamTools-master.zip`

```
 It is recommended to use Anaconda to create a separate python environment
to prevent incompatible versions of different software dependencies.
```

Miniconda instructions

```
2.2.2 Using full functionality (including response matrix generation)
    Calibration library CALDB download
    (1) To download CALDB, refer to the published link
```

GECAM CALDB

```
    (2) Download the latest version of CALDB from the GECAM cluster.(As of
2022-03-18, the path to the latest version of CALDB is: /gecamfs/soft/CALDB)
```

---

```
        Calibration Library CALDB Installation: Linux or Mac
        (1) After downloading the calibration library CALDB, put the source
/CALDB path/software/tools/caldbinit.sh in the environment file (~/.bashrc).
        (2) source environment file, and check whether the calibration library is
installed successfully by `from RSP_Generator import gen_rsp_fits`.
```

---

```
        Calibration library CALDB installation: Windows (tested on Win10).
        (1) Add a system variable: variable name: CALDB, address is the root
directory of CALDB, for example: E:\gecam\CALDB
        (2) Create a new CALDB.pth file in the site-packages of your python
environment.
            The corresponding directory for python or conda can be found by:
            >>> import os
            >>> os.path.dirname(os.__file__)
            >>> 'C:\\ Users\\ username\\.conda\\envs\\gecamTools\\lib'
            So create a new file:
C:\Users\Username\.conda\envs\gecamTools\Lib\site-packages\CALDB.pth
            The content of CALDB.pth is the path to the software folder in CALDB:
E:\gecam\CALDB\software
        (3) After finishing, you can enter python and check whether the
calibration library is installed successfully by `import RSP_Generator`.

2.3 Test
`import gecam
```

3. Uninstallation process

`pip uninstall GECAMTools`

## Get the test data used for the example

Download link for test data

1. Evt file（gbg_evt_tn210511_112749_fb_v00.fits）： download link
2. Attitude orbit file（gb_posatt_tn210511_112749_v00.fits）： download link

## function list

1. Basic functions
   1.1 Time conversion
2. Multi-detector batch analysis
   2.1 Display optical transition
   2.2 Display energy spectrum
   2.3 Generate response file (depends on GECAM CALDB)
   2.4 Generate energy spectrum file
3. Detailed single detector analysis
   3.1 Show optical transitions (view and correct background fit results for individual energy bands)
   3.2 Display energy spectrum
   3.3 Generate response file (depends on GECAM CALDB)
   3.4 Generate energy spectrum files
4. Analysis of eruption phenomena
   4.1 Flare duration estimation (T90, T50), customized data can be used.

# 1. basic function

## 1.1 get the version of GECAMTools

```
import gecam
gecam.get_software_name(),gecam.get_software_version()
```

```
('GECAMTools', '20230701')
```

## 1.2 time conversion

```
from gecam.time import GecamMet
```

```
trig_met=74431600.6
```

```
# met to time string
trig_time_str=GecamMet(trig_met).iso
# met to datetime
trig_datetime=GecamMet(trig_met).datetime
# met to mjd
trig_mjd=GecamMet(trig_met).mjd

trig_time_str,trig_datetime,trig_mjd
```

```
('2021-05-11T11:26:40.600000',
 datetime.datetime(2021, 5, 11, 11, 26, 40, 600000,
tzinfo=datetime.timezone.utc),
 59345.4768587963)
```

```
# time string to met
met1=GecamMet.from_iso(trig_time_str)
# datetime to met
met2=GecamMet.from_datetime(trig_datetime)
# MJD to met
met3=GecamMet.from_mjd(trig_mjd)

met1,met2,met3
```

```
(<GecamMet seconds = 74431600.600000>,
 <GecamMet seconds = 74431600.600000>,
 <GecamMet seconds = 74431600.600000>)
```

# 2. Batch analysis of multiple detectors

## 2.1 Read evt data (level 1 daily or trigger evt data)

```
from gecam.data.evt import Evt
from gecam.data.spec import SpecFile
from gecam.data.detector import GRD, CPD
from gecam.plot.light_curve import LightCurveFigure
from gecam.plot.spectrum import SpectrumFigure

import matplotlib.pyplot as plt
```

```
from gecam.data.evt import Evt

evt_path=r"test_gecam_data/gbg_evt_tn210511_112749_fb_v00.fits"
evt = Evt.open(evt_path)
```

```
from gecam.time import GecamMet

evt_info=evt.info
print("satellite:",evt_info.satellite)
print("satellite full name:",evt_info.satellite_full_name)
print("instrument:",evt_info.instrument)
print("trigger id:",evt_info.trig_id)
print("trigger met:",evt_info.trig_met, GecamMet(evt_info.trig_met).iso)
print("observe met range:",evt_info.obs_met_range)
print("location (ra, dec, error):",evt_info.loc)

# ebounds
# print(evt_info.ebounds)
# primary header
# print(evt_info.primary_header)
```

```
satellite: b
satellite full name: GECAM-B
instrument: GRD
trigger id: tn210511_112749_fb
trigger met: 74431600.6 2021-05-11T11:26:40.600000
observe met range: (74431501.0, 74431899.0)
location (ra, dec, error): (317.99, 59.53, 3.19)
```

### 2.1.2 Data croping and saving

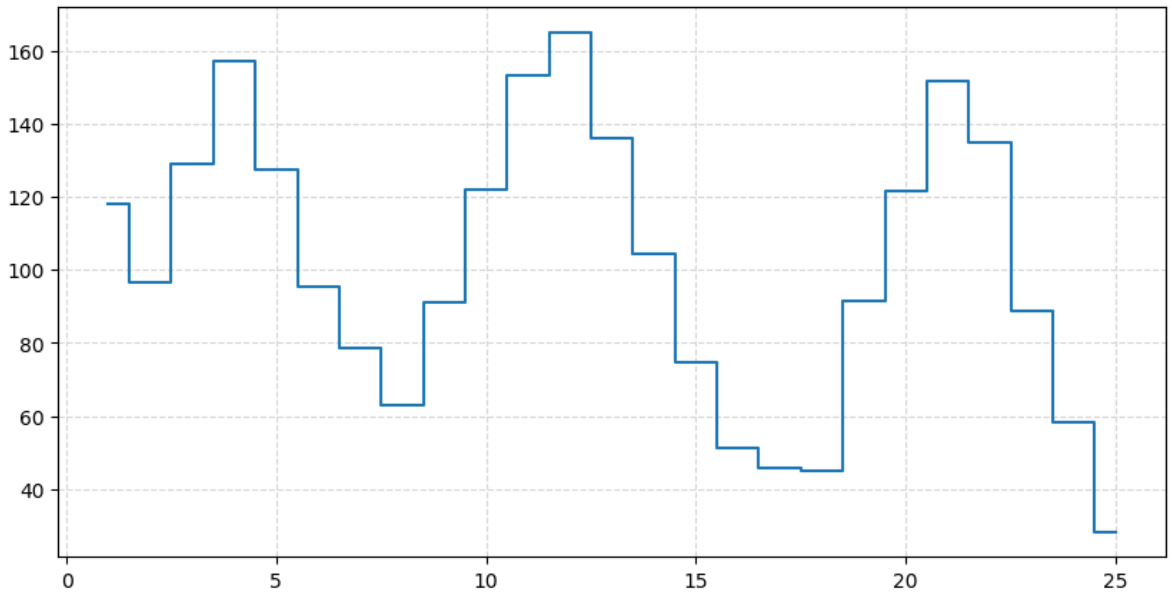
```
trig_met = evt.info.trig_met

# Each detector only saves data for the crop_time_range time period, leaving all other information unchanged
crop_time_range=[trig_met-10,trig_met+20]
crop_out_dir="out_gecam/"
croped_evt_path=evt.crop(crop_time_range,crop_out_dir)
croped_evt_path
```

```
'out_gecam/gbg_evt_tn210511_112749_fb_v00_74431590.6+30.0.fits'
```

### 2.1.3 Selection of the detectors according to the angle of incidence of the source

```
from gecam.data.posatt import PosAtt
from gecam.coord import radec_to_thetaphi
from gecam.utils import rsp_utils
from gecam.data.detector import GRD
from gecam.coord import cal_det_incident_angle

# read posatt
```

```
posatt_path=r"test_gecam_data/gb_posatt_tn210511_112749_v00.fits"
posatt_obj = PosAtt.open(posatt_path)


choose_met = evt.info.trig_met
# satallite: b/c
choose_satellite="b"

# set source coordinate（J2000）
ra,dec,err_radius=evt.info.loc


# Obtain the satellite attitude (quaternion) corresponding to the point in time
# Tips:
# For GECAMC(HEBS)， MET(59537900) The pose quaternions around this time are
# formatted incorrectly, unfixed at this time
# pre_quat = ['Q2', 'Q3', 'Q4', 'Q1'] # until 59537900
# quat = ['Q1', 'Q2', 'Q3', 'Q4'] # from 59538144
quat = posatt_obj.get_quat(choose_met)
if choose_satellite=="c" and choose_met<=59537900:
    quat=[quat[3],quat[0],quat[1],quat[2]]

det_num_list=[]
det_incident_list=[]
for i in range(1,26):
    choose_det = GRD(i)
    # The satellite to which the detector belongs must be set. a/b/c
    choose_det.set_satellite(choose_satellite)

    ra,dec,err_radius=evt.info.loc

    det_incident=cal_det_incident_angle(choose_det, ra, dec, quat)

    det_num_list.append(i)
    det_incident_list.append(det_incident)
    # print(i,det_incident)

plt.figure(figsize=(10,5),dpi=100)
plt.step(det_num_list,det_incident_list,where="mid")
plt.grid(ls='--', c='#d7d7d7')
```

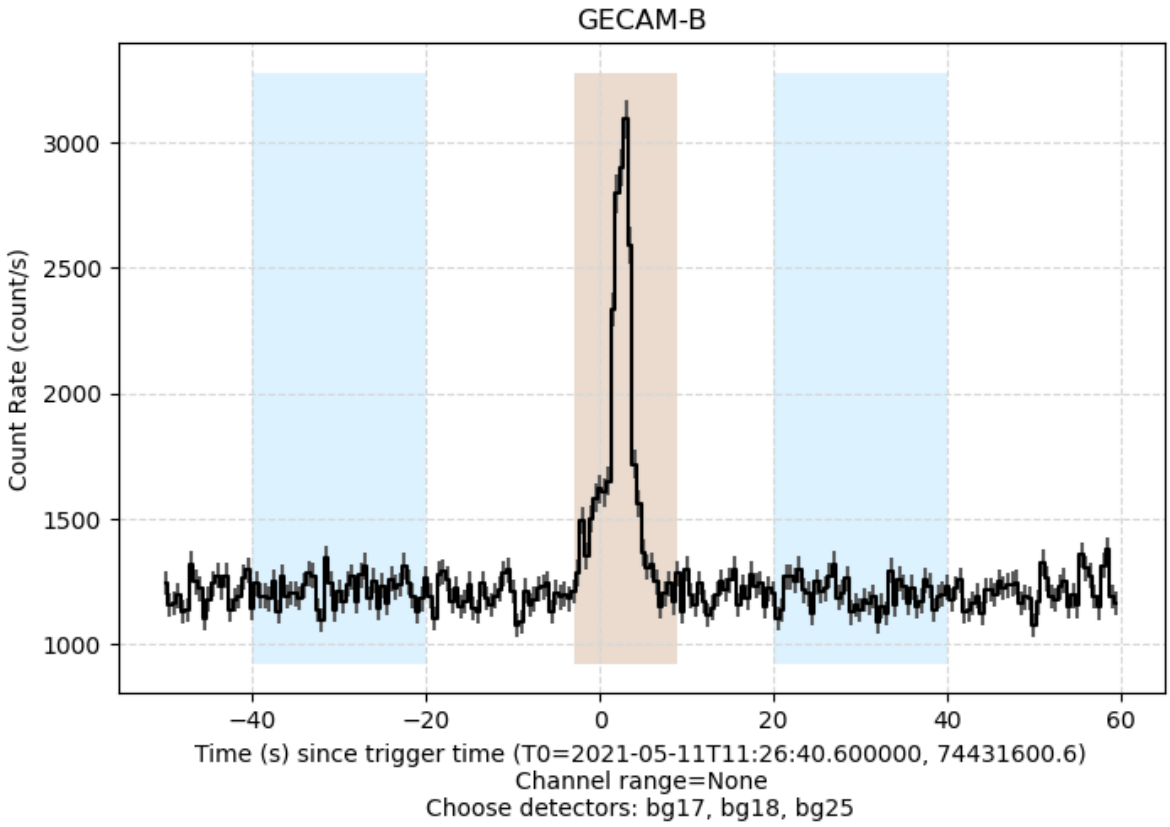



### Selection of a few detectors with the smallest source incidence angles:[17,18,25]

## 2.2 View superimposed light curves from multiple detectors

```
trig_met = evt.info.trig_met

chooose_det =
[GRD(17,gain_type="both"),GRD(18,gain_type="both"),GRD(25,gain_type="both")]
slice_kwargs_dic = {
    "time_range": [trig_met - 50, trig_met + 60],
    "only_recommend": True,
}

lc_kwargs_dic = {
    "time_bin": 0.5,
    "energy_bin": [40, 100, 300, 1000, 3000, 6000],
    # "channel_bin":1  # Customize the light curve by channel, when both
energy_bin and channel_bin exist, channel_bin has higher priority.
}

fig_kwargs_dic = {
    "bg_range": [[trig_met - 40, trig_met - 20],
                 [trig_met + 20, trig_met + 40]],
    "src_range": [trig_met - 3, trig_met + 9]
}

dets_lc_list, lc_data, lc_fig = evt.plot_light_curve_with_detectors(chooose_det,
slice_kwargs_dic,

 lc_kwargs_dic, fig_kwargs_dic)
```

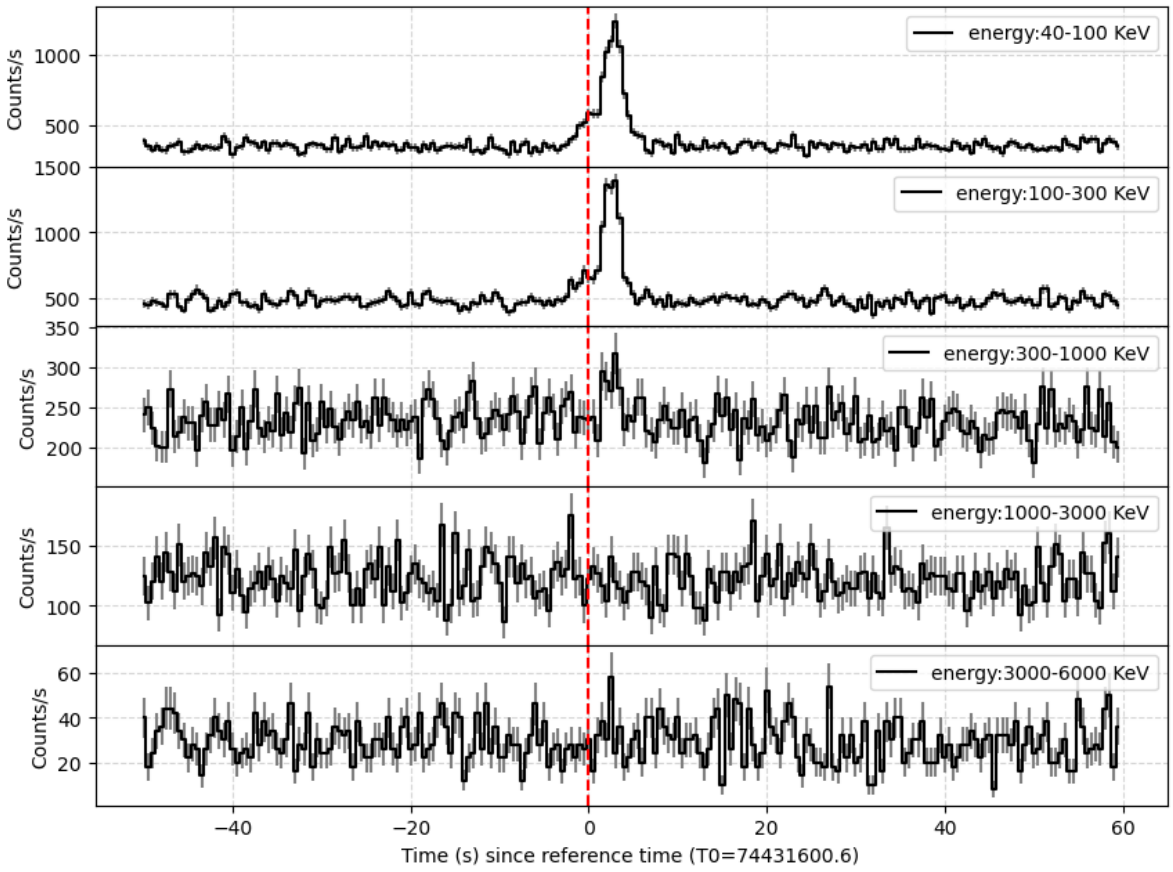
```
bg17H
bg17L
bg18H
bg18L
bg25H
bg25L
```




## 2.2.1 Extracting and mapping light curve data in sub-bands

```
import numpy as np

det_y_list = []
for det_lc_obj in dets_lc_list:
    # Get the light curve of the energy band, correct_by_dead_time=True, means
correct the dead time.
    det_time_x, det_y, det_y_err =
det_lc_obj.get_data(correct_by_dead_time=True)
    det_y_list.append(det_y)

det_y_list = np.array(det_y_list)
time_bin = lc_kwargs_dic["time_bin"]
time_bins = det_time_x
energy_bins = lc_kwargs_dic["energy_bin"]

choose_lc_y = det_y_list.sum(axis=0)

sub_fig_num = choose_lc_y.shape[0]
fig, axes = plt.subplots(sub_fig_num, 1, figsize=(10, sub_fig_num*1.5))
```

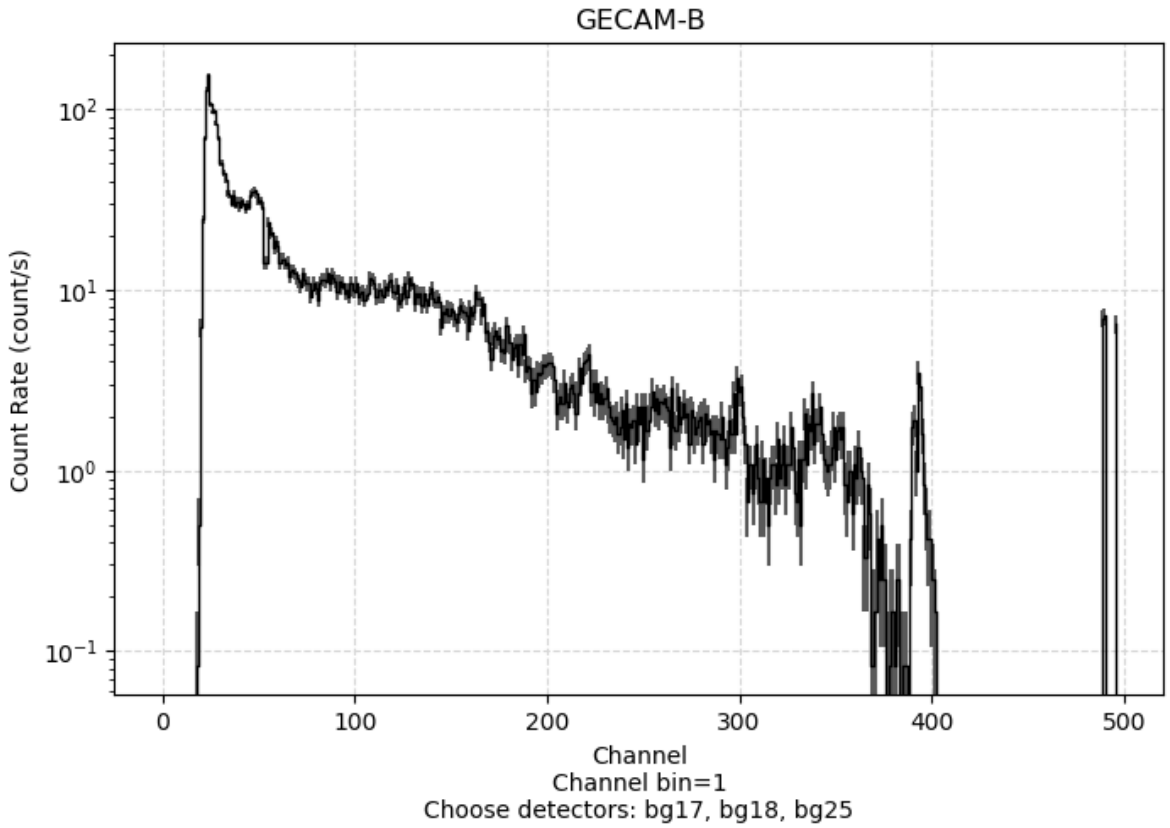
```
for index in range(sub_fig_num):
    ax = axes[index]

    channel_lc_x=time_bins[:-1] - trig_met
    channel_lc_y=choose_lc_y[index]/time_bin
    channel_lc_y_err=np.sqrt(choose_lc_y[index])/time_bin

    ax.step(channel_lc_x, channel_lc_y,
            label=f"energy:{energy_bins[index]}-{energy_bins[index + 1]}
KeV",color="black",where="mid")
    ax.errorbar(channel_lc_x, channel_lc_y, yerr=channel_lc_y_err, ls='',
color="gray")
    ax.set_ylabel("Counts/s")
    ax.legend(loc="upper right")
    ax.grid(ls='--', c='#d7d7d7')
    ax.axvline(0,c="red",ls='--')

axes[-1].set_xlabel(f"Time (s) since reference time (T0={trig_met})")
# plt.tight_layout()
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None,
hspace=0)
```




## 2.3 View the stacked energy spectrum of multiple detectors

```
chooose_det =
[GRD(17,gain_type="both"),GRD(18,gain_type="both"),GRD(25,gain_type="both")]
slice_kwargs_dic = {
    "time_range": [trig_met - 3, trig_met + 9],
    "only_recommend": True
}

spec_kwargs_dic = {
    "channel_bin": 1
}
fig_kwargs_dic = {}

dets_spec_list, plot_spec_data, spec_fig =
evt.plot_spectrum_with_detectors(chooose_det,

            slice_kwargs_dic,

            spec_kwargs_dic,

            fig_kwargs_dic)
plt.yscale("log")
plt.show()
```

```
bg17H
bg17L
bg18H
bg18L
bg25H
bg25L
```




## 2.4 Generate response files for multiple detectors

```
from gecam.data.posatt import PosAtt
from gecam.coord import radec_to_thetaphi
from gecam.utils import rsp_utils
from gecam.data.detector import GRD
# read posatt
posatt_path=r"test_gecam_data/gb_posatt_tn210511_112749_v00.fits"
posatt_obj = PosAtt.open(posatt_path)


choose_met = evt.info.trig_met
# satallite: b/c
choose_satellite="b"


choose_det = GRD(18, "high")
choose_det.set_satellite(choose_satellite)


ra,dec,err_radius=evt.info.loc

quat = posatt_obj.get_quat(choose_met)

if choose_satellite=="c" and choose_met<=59537900:
    quat=[quat[3],quat[0],quat[1],quat[2]]


# Calculation of the angle of incidence (deg) from the coordinates of the source
and the satellite attitude
```

```
theta, phi = radec_to_thetaphi(ra, dec, quat,satellite=choose_satellite)

# evt/bspec/btime
event_type="evt"

out_dir = "./test_rsp/"

rsp_out_path = rsp_utils.generate_rsp_fits(choose_det.full_name, theta, phi,
choose_met,event_type, out_dir)
rsp_out_path
```

```
read CALDB in env: /tmp/download/CALDB_v1.0
read index /tmp/download/CALDB_v1.0/data/gecam-b/grd/caldb.indx
Query_And_Read_Eff::read_data Error, no index record pass filter: detname=bg18H
{'MET': 744316OO.6}
Query_And_Read_Eff::read_eff_area_corr Error, Fail to read_data
gen_rsp Warning, fail to read_effarea_corr bg18H Do NOT correct effective area
```

```
'./test_rsp/gbg_18H_x_evt_v20230604.rsp'
```

```
# Generate response files for multiple detectors (must distinguish between high
and low gain)
import os
out_dir="./out_rsp/GECAMB/"

det_rsp_list = []

choose_dets=[]
choose_dets.extend([GRD(number=i) for i in range(1,3)])


for det in choose_dets:
    temp_rsp_list = []
    for gain_type in ["high", "low"]:
        det.set_gain_type(gain_type)
        det.set_satellite("b")
        if det.satellite=="c" and det.number in [6,12] and gain_type=="low":
            continue

        event_type="evt"

        temp_rsp_path = rsp_utils.generate_rsp_fits(det.full_name, theta, phi,
choose_met,event_type, out_dir)
        temp_rsp_list.append(os.path.basename(temp_rsp_path))
        print(temp_rsp_path)

    det_rsp_list.append(temp_rsp_list)
```

```
det_rsp_list
```

```
./out_rsp/GECAMB/gbg_01H_x_evt_v00.rsp
./out_rsp/GECAMB/gbg_01L_x_evt_v00.rsp
./out_rsp/GECAMB/gbg_02H_x_evt_v00.rsp
./out_rsp/GECAMB/gbg_02L_x_evt_v00.rsp
```

```
[['gbg_01H_x_evt_v00.rsp', 'gbg_01L_x_evt_v00.rsp'],
 ['gbg_02H_x_evt_v00.rsp', 'gbg_02L_x_evt_v00.rsp']]
```

## 2.5 Generate spectrum files for multiple detectors

```
chooose_det =
[GRD(17,gain_type="both"),GRD(18,gain_type="both"),GRD(25,gain_type="high")]
lc_kwargs_dic = {
    "time_bin": 1,
    "channel_bin": [4, 4],  # [channel_bin_high_gain,channel_bin_low_gain]
}
lc_bg_fit_kwargs_dic = {
    "bg_time_range": [[trig_met - 40, trig_met - 10],
                      [trig_met + 25, trig_met + 60]],
    "fit_order": 1,
}
spec_file_kwargs_dic = {
    "src_range_list": [
        [trig_met - 1, trig_met + 9],
        [trig_met - 1, trig_met + 5],
        [trig_met + 5, trig_met + 9],
    ],
#     rsp_list: response file corresponding to the selected detector,
distinguishing between high and low gain
    "rsp_list": [["gbg_17H_x_evt_v00.rsp", "gbg_17L_x_evt_v00.rsp"],
                 ["gbg_18H_x_evt_v00.rsp", "gbg_18L_x_evt_v00.rsp"],
                 ["gbg_25H_x_evt_v00.rsp", "gbg_25L_x_evt_v00.rsp"]],
    "out_dir": "./out_gecam/spec/"
}

spec_data = evt.generate_spec_file_with_detecotrs(chooose_det, slice_kwargs_dic,
lc_kwargs_dic,
                                                  lc_bg_fit_kwargs_dic,
spec_file_kwargs_dic)
```

```
bg17H
```

```
/root/anaconda3/envs/gecamTools/lib/python3.9/site-
packages/gecam/fitting/polynomial_fitter.py:199: RuntimeWarning: background model
has negative value in following channel(s) (starting from 0): 116

This error maybe eliminated by reducing the order of polynomial (the current is
1).
  warnings.warn(
/root/anaconda3/envs/gecamTools/lib/python3.9/site-
packages/gecam/data/spec.py:135: UserWarning: Mask channel Quality of GECAM-A/B/C
(greater than 448) to 1.
  warnings.warn("Mask channel Quality of GECAM-A/B/C (greater than 448) to 1.")
```

```
bg17L
bg18H
bg18L
bg25H
```

```
spec_data.keys()
```

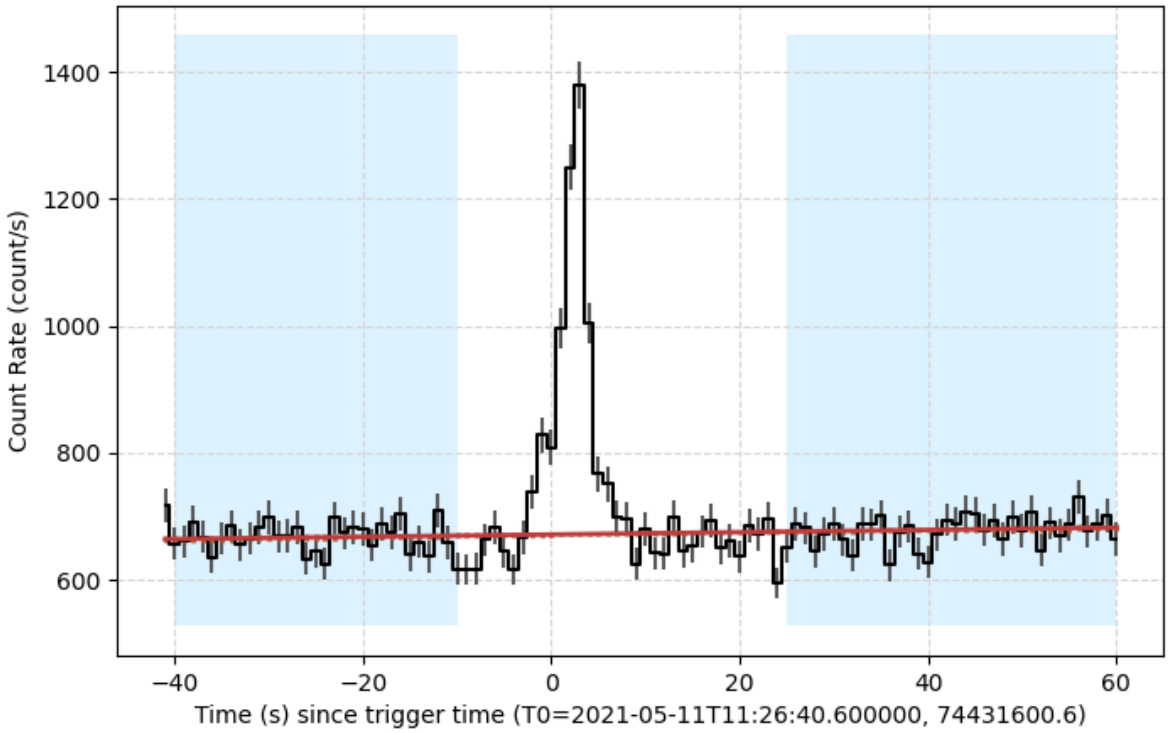
```
dict_keys(['bg18H', 'bg18L', 'bg25H'])
```

```
spec_data_bg18H = spec_data.get("bg18H")

bg18H_lc=spec_data_bg18H.get("lc")
bg18H_bg_lc=spec_data_bg18H.get("bg_lc")

spec_list_bg18H=spec_data_bg18H.get("src_spec_list")

spec_list_bg18H_src1 = spec_list_bg18H[0]

src_range1, spec_bg18H, bg_spec_bg18H, net_spec_bg18H = spec_list_bg18H_src1
```

```
spec_list_bg18H_src1
```

```
[[74431599.6, 74431609.6],
 <gecam.data.curve.Spectrum at 0x7f027681a310>,
 <gecam.data.curve.Spectrum at 0x7f027681a3a0>,
 <gecam.data.curve.Spectrum at 0x7f027681a460>]
```

```
det_sliced_c_lc_fig = LightCurveFigure(bg18H_lc.get_plot_data(),
trig_time=trig_met)

det_sliced_c_lc_fig.add_background(bg18H_bg_lc.get_plot_data(),
                                   bg_time_range=bg18H_bg_lc.bg_time_range,
label="bg")

# View the evaluation of the background fit of bg18H
# channel_range=[10,200]
# bg18H_bg_lc.show_fitting_quality(channel_range=channel_range)
# plt.show()
```




```
channel_bins = spec_bg18H.channel_bins
energy_bins = spec_bg18H.energy_bins
counts = spec_bg18H.counts
count_err = spec_bg18H.counts_err
```

# 3 Detailed analysis of individual detectors

## 3.1 read evt file

```
evt_path=r"test_gecam_data/gbg_evt_tn210511_112749_fb_v00.fits"
evt = Evt.open(evt_path)
```

## 3.2 filter data of single detector

```
det_events = evt.select_detector(18)
```

## 3.3 For evt of single detector, further filtering

### 3.3.1 If the data will be used later to generate a spectrum file, the gain must be filtered and full gain cannot be selected.

```
# Further data filtering for detector data, gain, time range, energy range,
channel range, whether to use only recommended examples.

# Filter by gain (if used to generate a spectrum, the gain must be set), both:
full gain, high: high gain, low: low gain
# gain_type=None, # gain_type=None, # gain_type=None, # gain_type=None

# Filter by time
# time_range=None, # filter by time

# filter energy_range or channel_range, not at the same time, if channel_range is
not None, then filter channel only
#energy_range=None, # filter only channels if channel_range is not None.
#channel_range=None.

# only_recommended instances
#only_recommend=True
det_sliced_events = det_events.slice(gain_type="high", only_recommend=True)
```

## 3.4 Extraction of light curves from filtered data

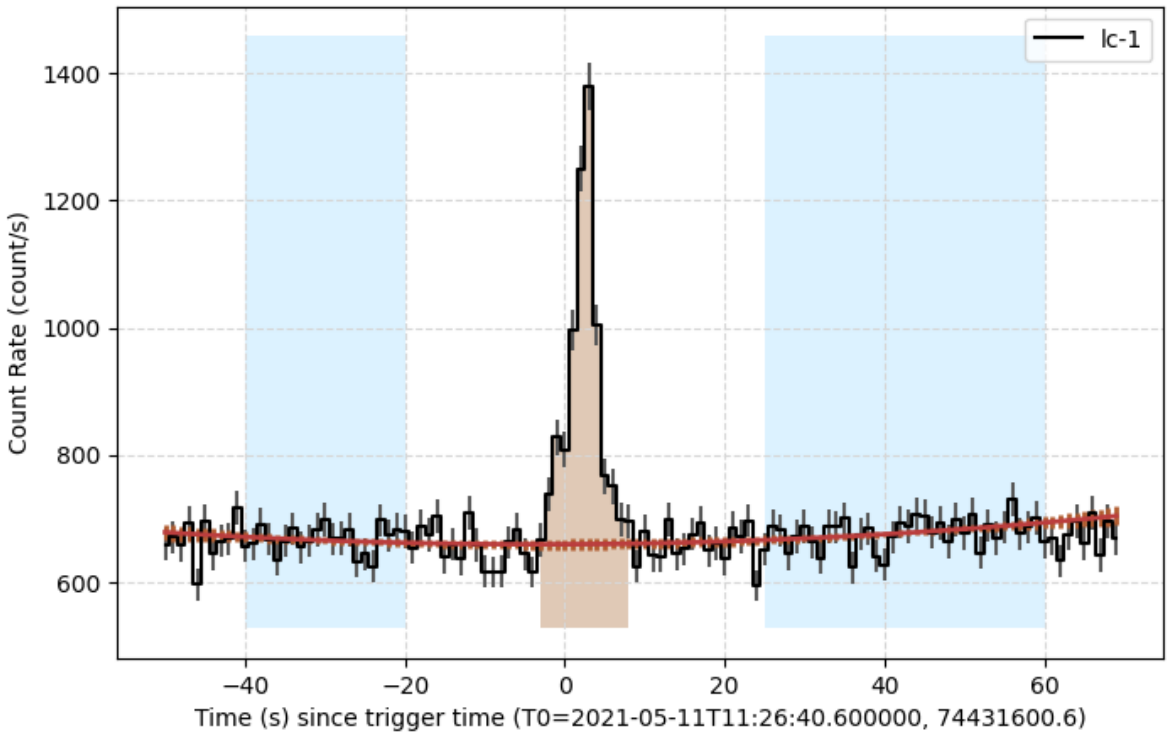
```
trig_met = evt.info.trig_met

lc_time_range = (trig_met - 50, trig_met + 70)

# define soource time range
src_time_range = [trig_met - 3, trig_met+9]

bg_time_range_list = [[trig_met - 40, trig_met - 20],
                      [trig_met + 25, trig_met + 60]]


time_bin = 1

channel_bin = 1

det_sliced_lc = det_sliced_events.to_light_curve(lc_time_range, time_bin,
channel_bin)

det_sliced_bg_lc = det_sliced_lc.fit_background(bg_time_range_list, fit_order=2)
```

```
det_src_lc = det_sliced_events.to_light_curve(src_time_range, time_bin,
channel_bin)
```

```
/root/anaconda3/envs/gecamTools/lib/python3.9/site-
packages/gecam/fitting/polynomial_fitter.py:271: RankWarning: The fit may be
poorly conditioned
  self._coeffs[i] = self._weighted_leastsq(X, y[i], w[i], False)
/root/anaconda3/envs/gecamTools/lib/python3.9/site-
packages/gecam/fitting/polynomial_fitter.py:199: RuntimeWarning: background model
has negative value in following channel(s) (starting from 0): 20, 170, 171

This error maybe eliminated by reducing the order of polynomial (the current is
2).
  warnings.warn(
```

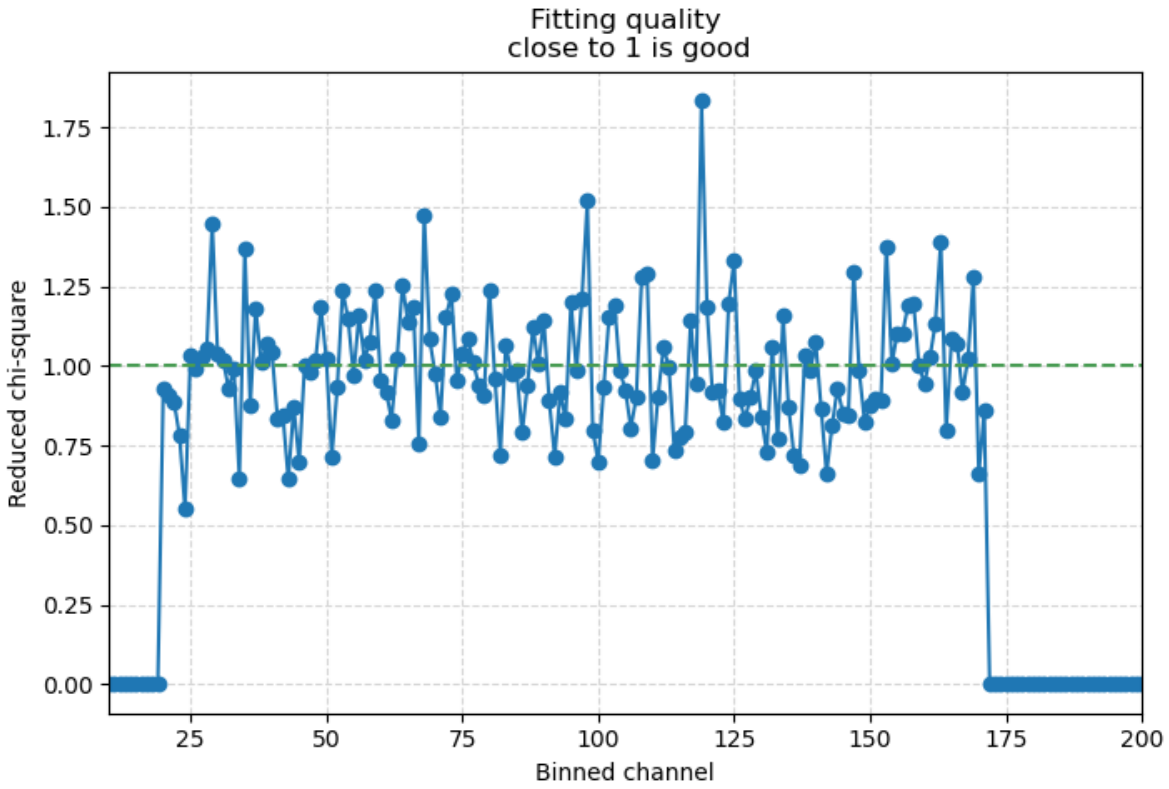
## 3.5 Drawing light curves that combine all energy segments

```
det_sliced_lc_fig = LightCurveFigure(det_sliced_lc.get_plot_data(),
trig_time=trig_met, dpi=100)

det_sliced_lc_fig.add_background(det_sliced_bg_lc.get_plot_data(),
                                 bg_time_range=det_sliced_bg_lc.bg_time_range)

det_sliced_lc_fig.add_selection(det_src_lc.get_plot_data())
det_sliced_lc_fig.show_legend()
```



## 3.6 View the evaluation of the fitting results for each energy band

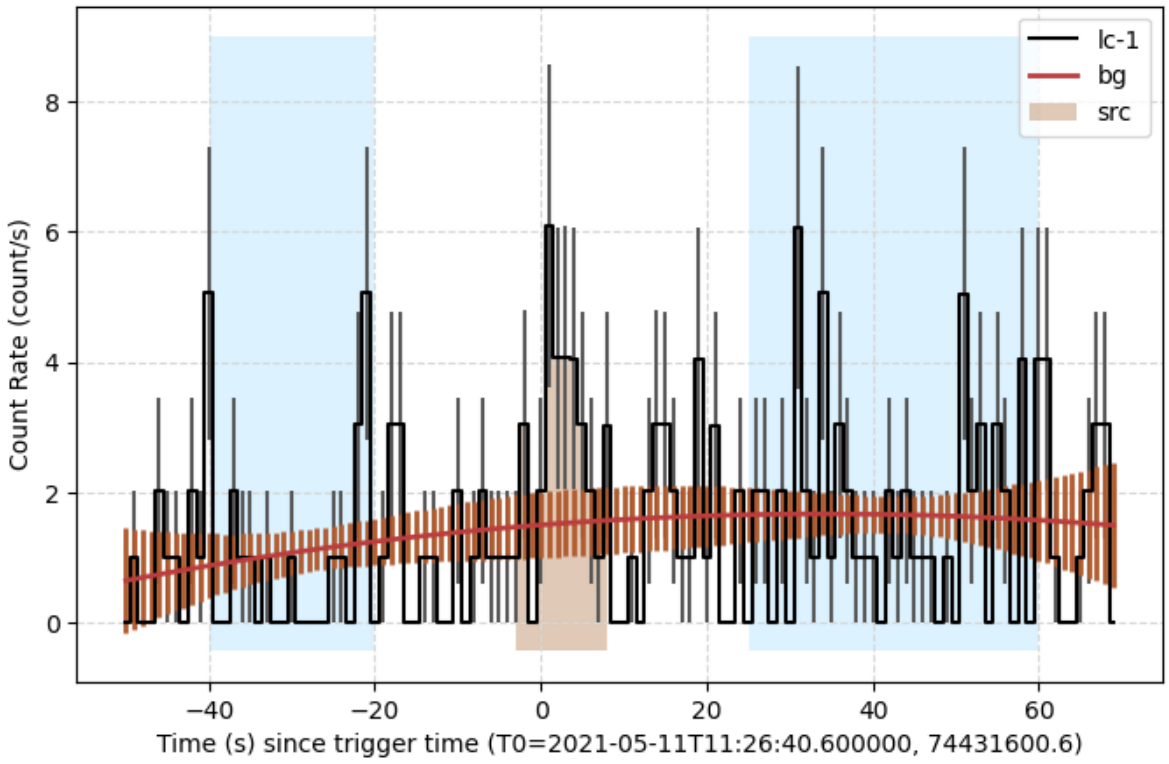
```
# Select to view the evaluation index of the fitting results of the 10th to 200th channel after merging, None: to view all channels.
# The closer the current metric is to 1, the better
channel_range=[10,200]
quality_data=det_sliced_bg_lc.show_fitting_quality(channel_range=channel_range)

# For a particular energy channel that was not fitted well.
# Then follow up with a program to view the optical transitions in that energy band, or re-fit the background for that energy band individually
```




## 3.7 Draw the light curves of the energy segment numbered 100

```
# Set the energy band number to 100 (if a merge was performed during the generation of the
# light curve of the energy band, the energy band number after the merge will be selected)
choose_channel_index = 119


channel_lc = det_sliced_lc.get_channel_lc(choose_channel_index)
channel_src_lc = det_src_lc.get_channel_lc(choose_channel_index)
channel_bg_lc = det_sliced_bg_lc.get_channel_lc(choose_channel_index)
```

```
det_sliced_c_lc_fig = LightCurveFigure(channel_lc.get_plot_data(),
trig_time=trig_met)

det_sliced_c_lc_fig.add_background(channel_bg_lc.get_plot_data(),
                                   bg_time_range=channel_bg_lc.bg_time_range,
label="bg")

det_sliced_c_lc_fig.add_selection(channel_src_lc.get_plot_data(), label="src")
det_sliced_c_lc_fig.show_legend()
# plt.show()
```

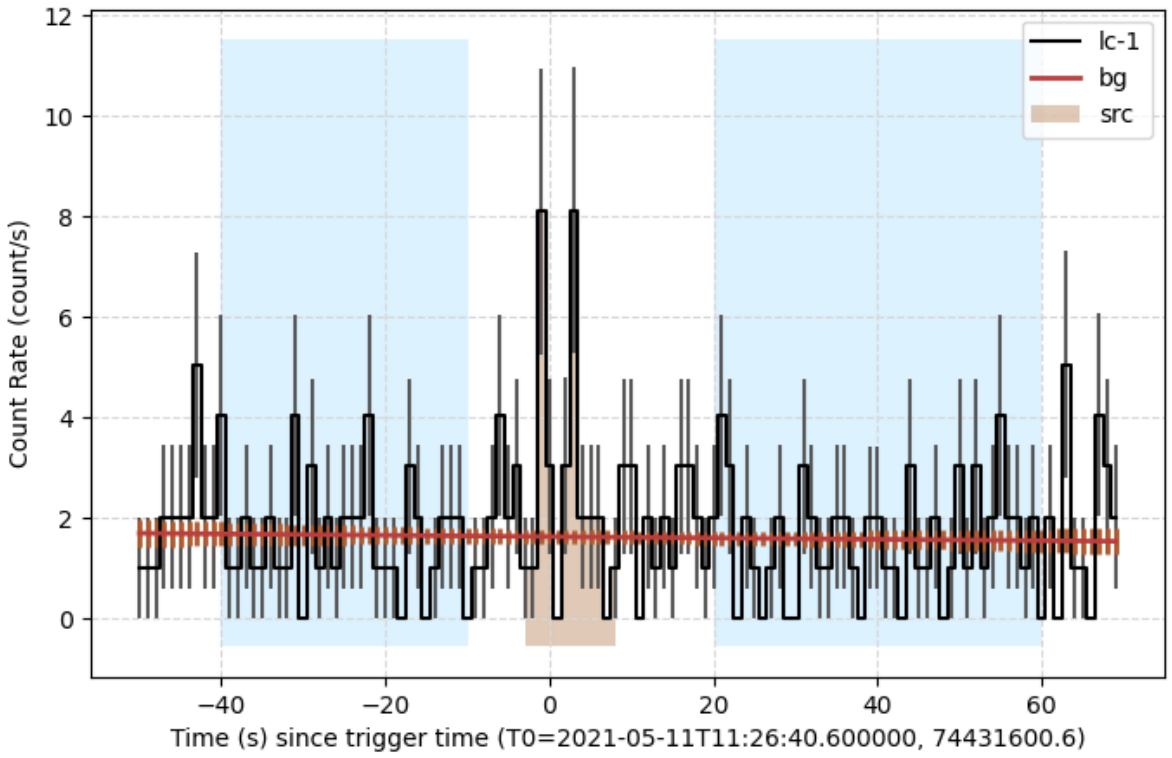



## 3.8 Adjusting the background fit for individual energy bands

```
# Selection of energy segment number 100, (Selection of energy segment number
after merging)
choose_channel_index = 100

bg_time_range_list2 = [[trig_met - 40, trig_met - 10],
                       [trig_met + 20, trig_met + 60]]
fit_order = 1

channel_lc = det_sliced_lc.get_channel_lc(choose_channel_index)
channel_bg_lc = channel_lc.fit_background(bg_time_range_list2,
fit_order=fit_order)
print(channel_bg_lc.fit_info)

channel_src_lc = det_src_lc.get_channel_lc(choose_channel_index)

det_c_lc_fig = LightCurveFigure(channel_lc.get_plot_data(), trig_time=trig_met)
```

```
det_c_lc_fig.add_background(channel_bg_lc.get_plot_data(),
                            bg_time_range=channel_bg_lc.bg_time_range,
label="bg")
det_c_lc_fig.add_selection(channel_src_lc.get_plot_data(), label="src")
det_c_lc_fig.show_legend()
```

```
['2pass', 1, 49.8133576613236, 68.0, 0.7325493773724059]
```



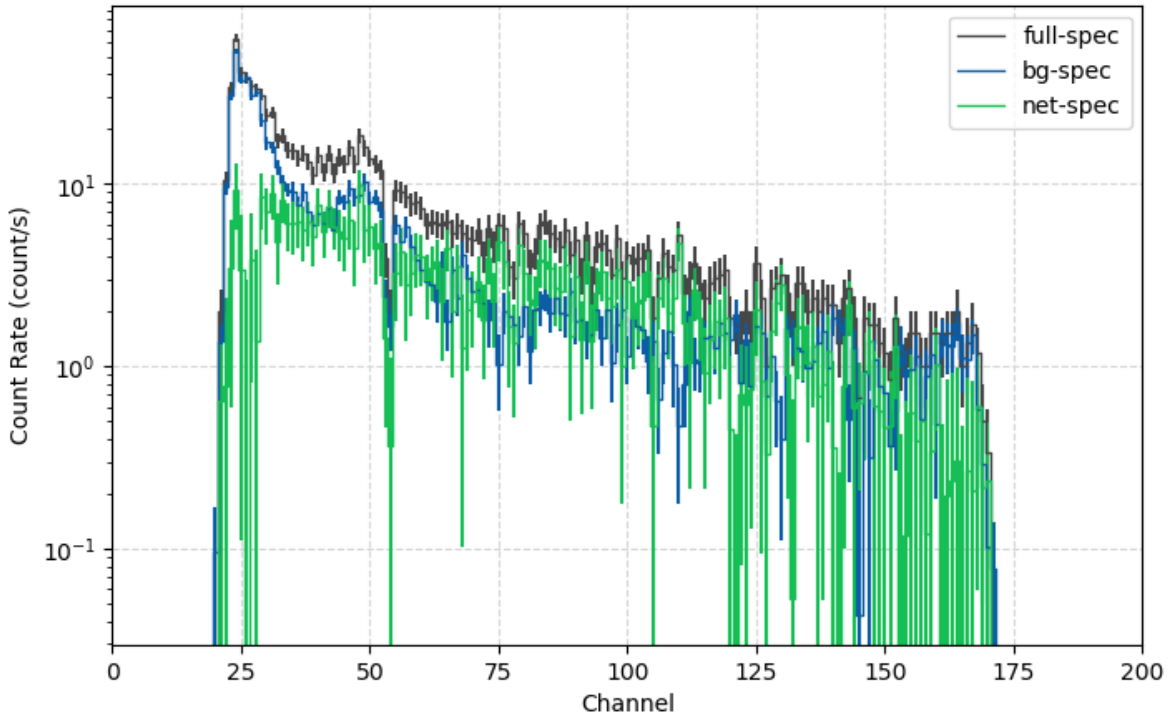

## 3.9 Generate the energy spectrum of the current detector

```
# Generation of spectral file data, time-discretized spectra

from gecam.data.spec import SpecFile

spec_file = SpecFile(det_sliced_lc, det_sliced_bg_lc)

# add first source range
src_time_range=(trig_met -1, trig_met + 5)
spec, bg_spec, net_spec = spec_file.add_src(src_time_range)

# add second source range
src_time_range2 = (trig_met + 5, trig_met + 9)
spec2, bg_spec2, net_spec2 = spec_file.add_src(src_time_range2)
```

## 3.10 View spectrum

```
from gecam.plot.spectrum import SpectrumFigure
```

```
spec_fig = SpectrumFigure()
spec_fig.add_data(spec.get_plot_data(), color="#474747", err_color="#474747",
label="full-spec",
                  linewidth=1)
spec_fig.add_data(bg_spec.get_plot_data(), color="#0c5da5", err_color="#0c5da5",
label="bg-spec",
                  linewidth=1)
spec_fig.add_data(net_spec.get_plot_data(), color="#16bf55",
err_color="#16bf55", label="net-spec",
                  linewidth=1)
spec_fig.set_xlim([0, 200])
# spec_fig.set_yscale("linear")
spec_fig.show_legend()
```




## 3.10 Exporting the energy spectrum to an energy spectrum file

```
# Response file corresponding to the spectrum
rsp_path = "test.rsp"
out_dir=r"./"
spec_file.write(out_dir, rsp_path=rsp_path)
```

# Burst phenomenon analysis

## Estimated duration of the burst (T90, T50)

```
from gecam.data.evt import Evt
from gecam.analysis.burst_duration import BurstDuration
from gecam.data.detector import Detector, GRD

evt_path = r"test_gecam_data/gbg_evt_tn210511_112749_fb_v00.fits"
evt = Evt.open(evt_path)
```

## Generate cumulative count curves for net light curve

```
trig_met=evt.info.trig_met
det_list = [GRD(17, gain_type="both"),GRD(18, gain_type="both"), GRD(25,
gain_type="both")]


slice_kwargs_dic = {
    "time_range": [trig_met - 20, trig_met + 25],
    "only_recommend": True
}
lc_kwargs_dic = {
    "time_bin": 0.05,
    "energy_bin":[5,5000]
}
lc_bg_fit_kwargs_dic = {
    "bg_time_range": [[trig_met - 15, trig_met - 3], [trig_met + 10, trig_met +
20]],
    "fit_order": 0
}
duration_obj = BurstDuration()
lc_dic,
cumsum_net_lc_data=duration_obj.generate_net_light_curve_with_detectors(evt,
det_list, slice_kwargs_dic,

 lc_kwargs_dic,lc_bg_fit_kwargs_dic)
```

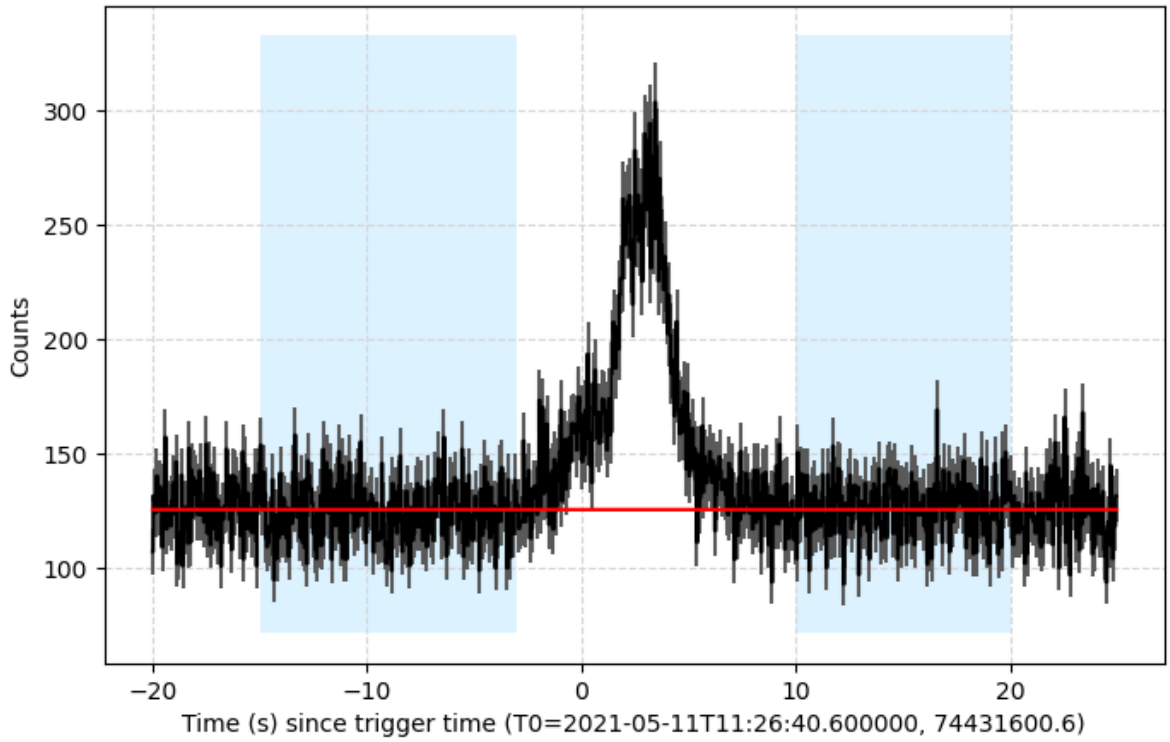
```
bg17H
bg17L
bg18H
bg18L
bg25H
bg25L
```

```
lc_dic.keys()
```

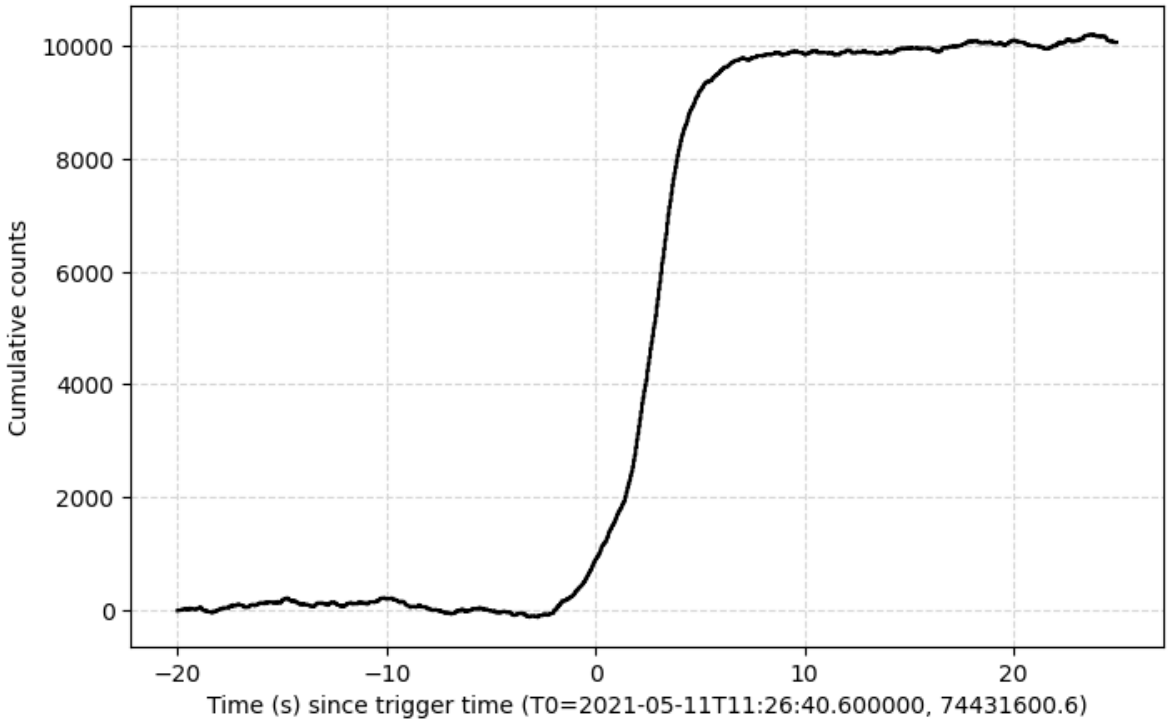
```
dict_keys(['bg18H', 'bg18L', 'bg25H', 'bg25L'])
```

```
# lc_dic["bg18H"].total_lc.dead_time_on_bins
```

```
from gecam.plot.light_curve import LightCurveFigure

total_lc_x, total_lc_y, total_lc_y_err = duration_obj.total_lc_1D_data
bg_lc_x, bg_lc_y, bg_lc_y_err = duration_obj.bg_lc_1D_data
# net_lc_x, net_lc_y, net_lc_y_err = duration_obj.net_lc_1D_data

lc_bg_range = lc_bg_fit_kwargs_dic.get("bg_time_range")

lc_fig = LightCurveFigure((total_lc_x[:-1], total_lc_y, total_lc_y_err),
                          trig_time=duration_obj.evt_info.trig_met, dpi=100)
lc_fig.add_data((bg_lc_x[:-1], bg_lc_y, bg_lc_y_err),
color="red",err_color="#FF5959")
lc_fig.add_background(bg_time_range=lc_bg_range)
lc_fig.set_ylabel("Counts")
```




```
# View cumulative count curves for net light curve
set_time_range=None
cumsum_lc_fig =
duration_obj.plot_light_curve_cumsum(set_time_range=set_time_range)
```



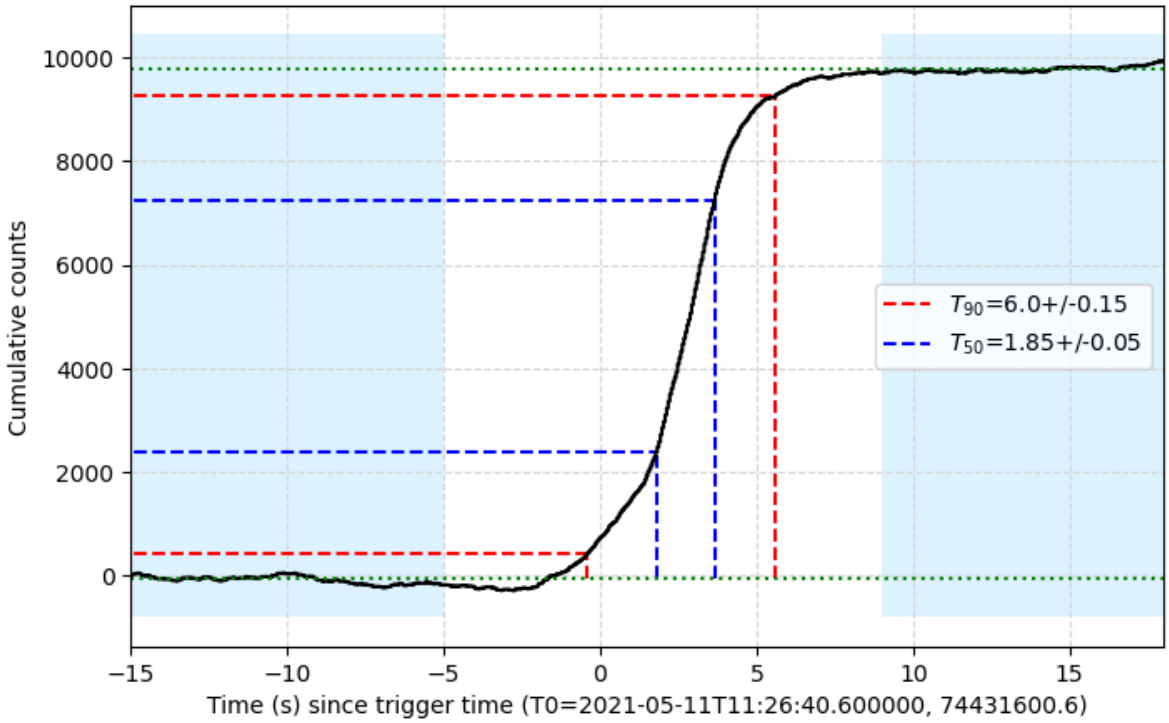

## calculate T90,T50

```
# Select the background range of the cumulative curve
cumsum_bg_range = [[trig_met - 15, trig_met - 5], [trig_met + 9, trig_met + 18]]

t90, t90_err,t50,
t50_err=duration_obj.cal_burst_duration_by_cumsum_counts(cumsum_bg_range)

print("T90:",round(t90,4),"T90 error:",round(t90_err,4),"T90 start met:",
duration_obj._T90_start)
print("T50:",round(t50,4),"T50 error:",round(t50_err,4),"T50 start met:",
duration_obj._T50_start)
```

```
T90: 6.0 T90 error: 0.15 T90 start met: 74431600.22500002
T50: 1.85 T50 error: 0.05 T50 start met: 74431602.4249999
```

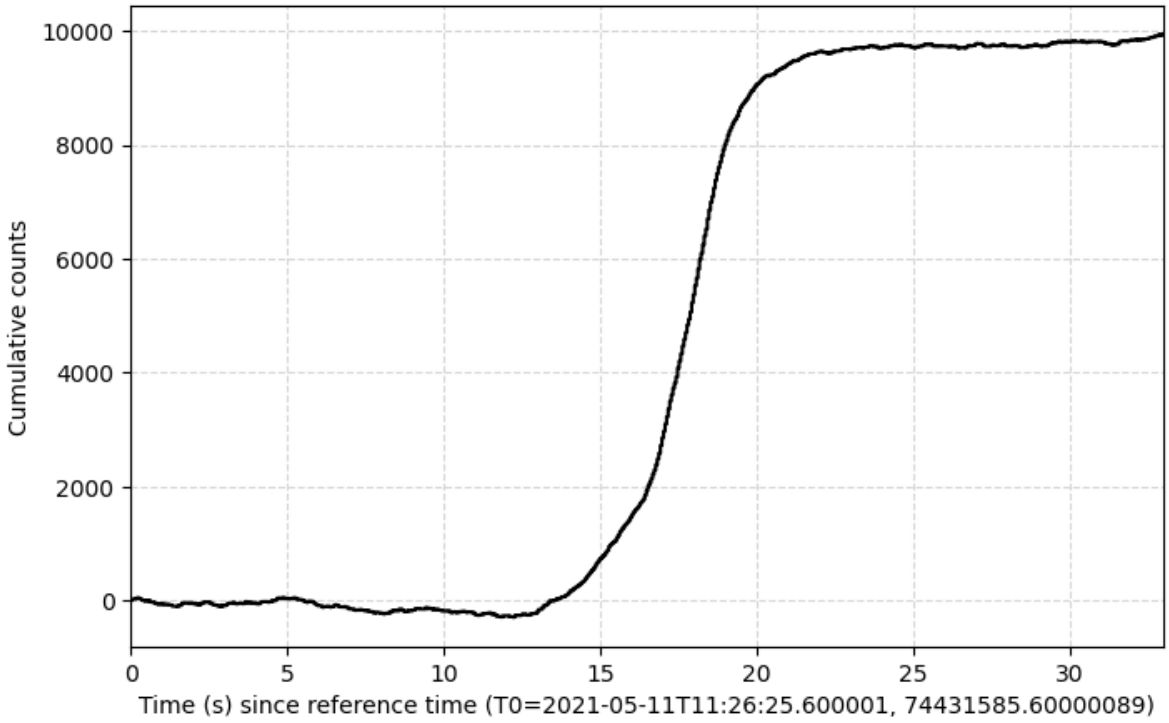
## draw T90, T50

```
set_time_range = [trig_met - 15, trig_met + 18]
# set_time_range=None
cumsum_lc_fig =
duration_obj.plot_light_curve_cumsum(set_time_range=set_time_range)
```




## Customize the data to estimate T90, T50

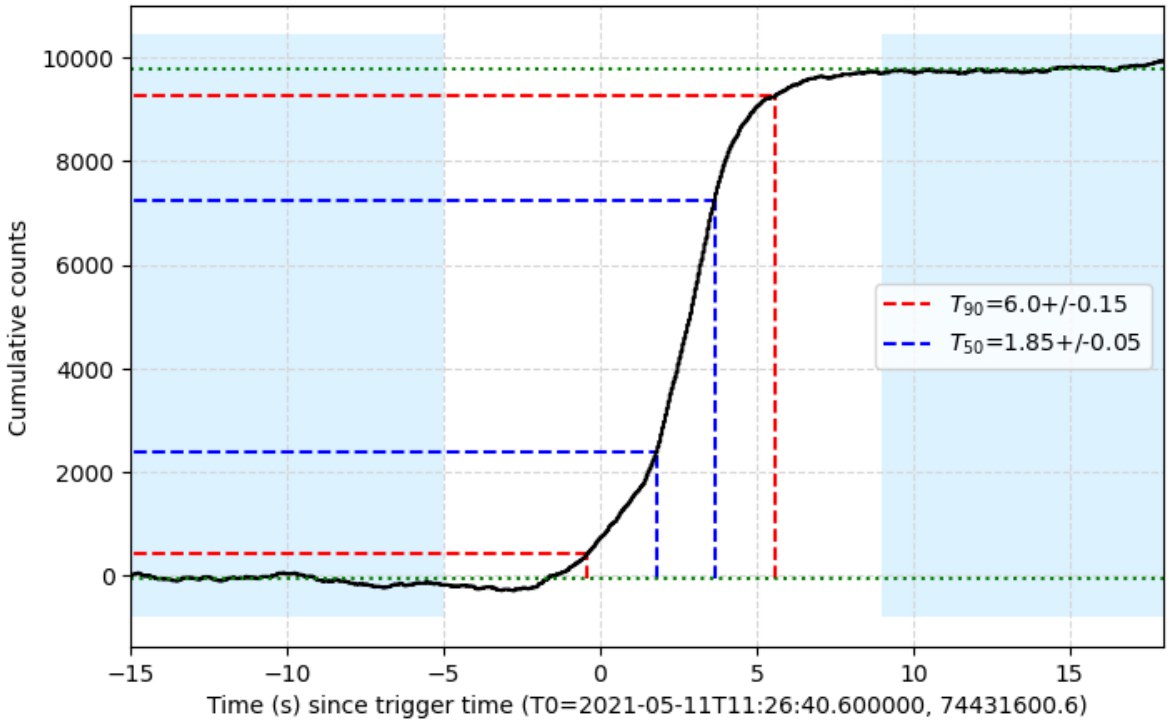
```
duration_obj3 = BurstDuration()

# Prepare the data: total_lc and net_lc
# (currently borrowing the light curves generated in the previous section as an
example).

# total_lc_data (list): total light curve (time bins, counts(one dimension),
counts error)
# net_lc_data (list): net light curve (time bins, counts(one dimension), counts
error)
total_lc_data, net_lc_data = duration_obj.total_lc_1D_data,
duration_obj.net_lc_1D_data

duration_obj3.update_custom_data(total_lc_data, net_lc_data)

cumsum_lc_fig3 =
duration_obj3.plot_light_curve_cumsum(set_time_range=set_time_range)
```




```
cumsum_bg_range = [[trig_met - 15, trig_met - 5], [trig_met + 9, trig_met + 18]]

t90, t90_err, t50, t50_err =
duration_obj3.cal_burst_duration_by_cumsum_counts(cumsum_bg_range)

print("T90:",round(t90,4),"T90 error:",round(t90_err,4),"T90 start met:",
duration_obj._T90_start)
print("T50:",round(t50,4),"T50 error:",round(t50_err,4),"T50 start met:",
duration_obj._T50_start)
```

```
T90: 6.0 T90 error: 0.15 T90 start met: 74431600.22500002
T50: 1.85 T50 error: 0.05 T50 start met: 74431602.4249999
```

```
from gecam.time import GecamMet

set_time_range = [trig_met - 15, trig_met + 18]
cumsum_lc_fig3_2 =
duration_obj3.plot_light_curve_cumsum(set_time_range=set_time_range,ref_time=tri
g_met)

# The time of the light change may not be the default GECAM met time, which can
be overridden xlabel
cumsum_lc_fig3_2.set_xlabel(f"Time (s) since trigger time (T0=
{GecamMet(trig_met).iso}, {trig_met})")
# cumsum_lc_fig3_2.fig.savefig("test.png",bbox_inches="tight")
```


Cumulative counts
10000
8000
6000
4000
2000
0
−15
−10
−5
0
5
10
15
Time (s) since trigger time (T0=2021-05-11T11:26:40.600000, 74431600.6)
$T_{90}$=6.0+/-0.15
$T_{50}$=1.85+/-0.05