GECAMTools 使用说明

简介

GECAMTools是GECAM数据的应用编程接口(API)。本工具基于 python3实现，目的是允许一般用户将GECAM数据分析整合到自己的脚本和工作流中，而不必考虑太多的细节。为此，本工具有高级的API层，允许用户仅用几行代码来读取、简化和可视化GECAM数据。对于专家用户以及希望对其分析的各个方面进行精细控制的用户，本工具提供了一个较低级别的API层。本工具目前已实现数据分析的基本功能，包括时间转换、查看光变、查看能谱、生成响应文件和生成能谱文件用于能谱拟合。

此外，GECAMTools的设计有考虑到通用性。本工具的数据接口目前使用的是数据是GECAM数据，可通过继承结构将许多功能推广到其他仪器的数据上。即使数据文件定义与GECAM的文件定义不同,一旦这些数据文件的读取接口被重定义，就可以使用本工具进行后续分析。

Github: 链接

开发人员:

张鹏(IHEP)、薛王陈(IHEP)、张艳秋(IHEP)、郑超(IHEP)和熊少林*(IHEP)

Email:

熊少林*(xiongsl@ihep.ac.cn)
张鹏(zhangp97@ihep.ac.cn),GECAMTools的安装和使用问题可联系此邮箱反馈。

GECAM介绍和数据发布网站：链接

安装说明

1. 所需环境
   1.1 系统环境：windows、linux、mac
   1.2 python环境：python版本>=3.6 (不建议安装python 3.10及以上版本)

   windows 10:实测python=3.8.5成功安装。
   linux (Centos): 实测python=3.9.5成功安装。
   mac (M1 芯片): 测试python=3.9.16, 首先用conda install astropy==4.3.1, 再正常使用pip install gecamTools.zip 即可成功安装

2. 安装流程
   2.1 下载源程序
   gecamTools-master.zip

   2.2 安装
   2.2.1 仅使用基本功能(即无需生成响应矩阵)
   使用pip进行源码安装，自动安装GECAMTools以及相关依赖库
   `pip install gecamTools-master.zip`

   > 建议使用Anaconda创建独立的python环境使用，防止出现不同软件的依赖库版本不兼容问题。

Miniconda使用说明

```
2.2.2 使用全部功能(包含生成响应矩阵)
    标定库CALDB 下载
    (1) 下载CALDB, 参考已公布的链接
```

GECAM CALDB

```
    (2) 从GECAM集群上下载最新版的CALDB.(截止2022-03-18，最新版本CALDB的路径为：/gecamfs/soft/CALDB)
```

```
        标定库CALDB 安装：Linux或Mac
        (1) 标定库CALDB下载之后, 将source /CALDB path/software/tools/caldbinit.sh 这句话放环境文件(~/.bashrc)里
        (2) source 环境文件，并通过 `from RSP_Generator import gen_rsp_fits` 来检验标定库是否安装成功
```

```
        标定库CALDB 安装: Windows (测试于win10):
        (1) 添加系统变量:  变量名为：CALDB,  地址为CALDB对应的根目录，例：E:\gecam\CALDB

        (2) 在python环境的site-packages中新建一个CALDB.pth文件
            可通过以下方法找到python或者conda对应的目录：
            >>> import os
            >>> os.path.dirname(os.__file__)
            >>> 'C:\\Users\\用户名\\.conda\\envs\\gecamTools\\lib'
            因此新建文件：C:\Users\用户名\\.conda\envs\gecamTools\Lib\site-packages\CALDB.pth
            CALDB.pth中内容为CALDB中software文件夹路径：E:\gecam\CALDB\software
        (3) 完成之后，可进入 python，通过 `import RSP_Generator` 来检验标定库是否安装成功

2.3 测试
`import gecam`
```

3. 卸载流程
   `pip uninstall GECAMTools`

获取示例所用的测试数据

测试数据的下载链接

1. 事例文件（gbg_evt_tn210511_112749_fb_v00.fits）： download link
2. 姿态轨道文件（gb_posatt_tn210511_112749_v00.fits）： download link

版本更新说明

--v20230701（当前版本）

修复问题:

1. 修复细节错误。

--v20230609

修复问题：

1. 修正计算T90时，净光变误差计算错误。

新增：

1. 增加gecam的事例数据根据时间段截取保存的功能(evt.crop)，见2.1.2节。

--v20230607

修复问题：

1. 修正画图错误。
2. 修正多探头叠加的光变只能单独修正死时间：
   evt.plot_light_curve_with_detectors和evt.plot_spectrum_with_detectors将不再返回叠加后的光变和能谱对象（total_lc_obj和total_spec_obj）。

--v20230605

修复问题：

1. 修正画图错误。
2. 细节问题修复。

--v20230531

修复问题：

1. 修正能谱错误（计数为整数）
2. 总光变自动修正死时间（此调整不影响能谱）。
   events.to_light_curve和lc_kwargs_dic中的参数correct_by_dead_time无效，都会修正死时间，只是总光变在取出数据（get_data, get_plot_data）时增加correct_by_dead_time来提取是否修正死时间的光变数据。
3. 细节问题修复

--v20230420

修复问题：

1. 修复evt中生成多探头光变的误差计算错误
2. 修正并道数据设置channel bin失效的问题
3. 能谱图中总谱、本底谱和净谱统一为修正死时间后的能谱，统一能谱为counts/s
4. T90计算异常的问题
5. GECAMC的入射角计算错误。

新增：

1. 新增各类型文件的统一函数用于(GECAM的事例数据、GECAM的并道数据)：多探头的画光变、画能谱、生成能谱文件、计算T90

   新增函数：

   evt.plot_light_curve_with_detectors

evt.plot_spectrum_with_detectors

evt.generate_spec_file_with_detecotrs

duration_obj.generate_net_light_curve_with_detectors

同时，原始函数将停止更新，过几个版本后删除

原始函数如下所示：

evt.plot_light_curve

evt.plot_spectrum

evt.generate_spec_file

duration_obj.generate_net_light_curve_with_detectors

cal_net_light_curve_cumsum

2. 新增自定义光变用于计算T90，不限于GECAM的数据（见本文末尾）

功能列表

1. 基础功能
   1.1 时间转换
2. 多探头批量分析
   2.1 显示光变
   2.2 显示能谱
   2.3 生成响应文件（依赖于GECAM CALDB）
   2.4 生成能谱文件
3. 单个探头的细致分析
   3.1 显示光变（查看并修正单个能段的本底拟合结果）
   3.2 显示能谱
   3.3 生成响应文件（依赖于GECAM CALDB）
   3.4 生成能谱文件
4. 爆发现象分析
   4.1 爆发持续时间估计 （T90, T50）,可使用自定义数据

极目GECAM卫星简介

极目卫星（GECAM）的简介和数据下载网站:链接

引力波暴高能电磁对应体全天监测器（Gravitational wave high-energy Electromagnetic Counterpart All-sky Monitor, 简称GECAM）

GECAM卫星是世界首个专门用于探测引力波暴高能电磁对应体的的小型空间高能望远镜。对黑洞、中子星等极端天体的剧烈爆发现象展开观测。这是我国首颗具有快速下传及发布天文警报的卫星。在轨准实时发现和定位高能爆发天体，及时向天文界发布观测警报，引导其他望远镜联合观测是GECAM卫星的重要功能。

GECAM-A/B数据简介

GECAM-A/B卫星的有效载荷由伽马射线探测器(GRD)、荷电粒子探测器(CPD)和载荷处理器(EBOX)组成。从结构组成来说，分成穹顶舱和电子学舱。GRD和CPD探测器均采用模块化设计。GRD包括25个探测器模块，每个模块均能探测从前部入射的伽马光子，视场～2π。25个GRD分别指向不同方向，在立体角上基本均匀分布(除了面向地球的方向)。这样的配置不仅使每颗星能监测除地球遮挡外的所有天区(～70%全天)，而且对各入射方向的接收面积相当，因此对各天区位置的灵敏度较为均匀，利用指向不同方向的GRD可计算源的入射方位，对暴发源进行定位。

探测器组成：25个GRD(高增益和低增益)；高低增益的能段覆盖范围不同

GECAM-C(HEBS)数据简介

GEACM-C和GECAM-A/B设计相似

探测器组成：12个GRD(高增益和低增益)；高低增益的能段覆盖范围不同。
其中GRD06和GRD12只有高增益

GECAM数据说明

1. GECAM PI道共498道: 0~497 (其中前448道为正常事例，后50道分别为25个探测器（区分高低增益）的超高事例)

1. 基础功能

1.1 获取GECAMTools的版本号

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

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

1.2 时间转换

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

```
trig_met=74431600.6
```

```
# met转时间字符串
trig_time_str=GecamMet(trig_met).iso
# met转datetime
trig_datetime=GecamMet(trig_met).datetime
# met转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)
```

```
# 时间字符串转met
met1=GecamMet.from_iso(trig_time_str)
# datetime转met
met2=GecamMet.from_datetime(trig_datetime)
# MJD转met
met3=GecamMet.from_mjd(trig_mjd)

met1,met2,met3
```

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

```
from gecam.time import GecamMet,HebsMet,HxmtMet

# 同理
# GECAM-C(HEBS)的时间转换（HebsMet）和GECAM类似
# HXMT(慧眼)的时间转换（HxmtMet）和GECAM类似
```

2. 多个探头批量分析

2.1 读取事例数据（1级daily或trigger事例数据）

```
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
```

```
# 事例文件路径
# /gecamfs/Archived-
DATA/GSDC/LEVEL1/triggers/2021/05/tn210511_112749_fb/gbg_evt_tn210511_112749_fb_
v00.fits

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)

# 查看ebounds
# print(evt_info.ebounds)
# 查看完整的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)
```

2.1.1 批量选取探头

```
# 示例：选取10个高增益，15个低增益
choose_det=[]

choose_det.extend([GRD(number=i, gain_type="high") for i in range(1,11)])

choose_det.extend([GRD(i,"low") for i in range(11,26)])

# [det.full_name for det in choose_det]
```

2.1.2 数据切片保存

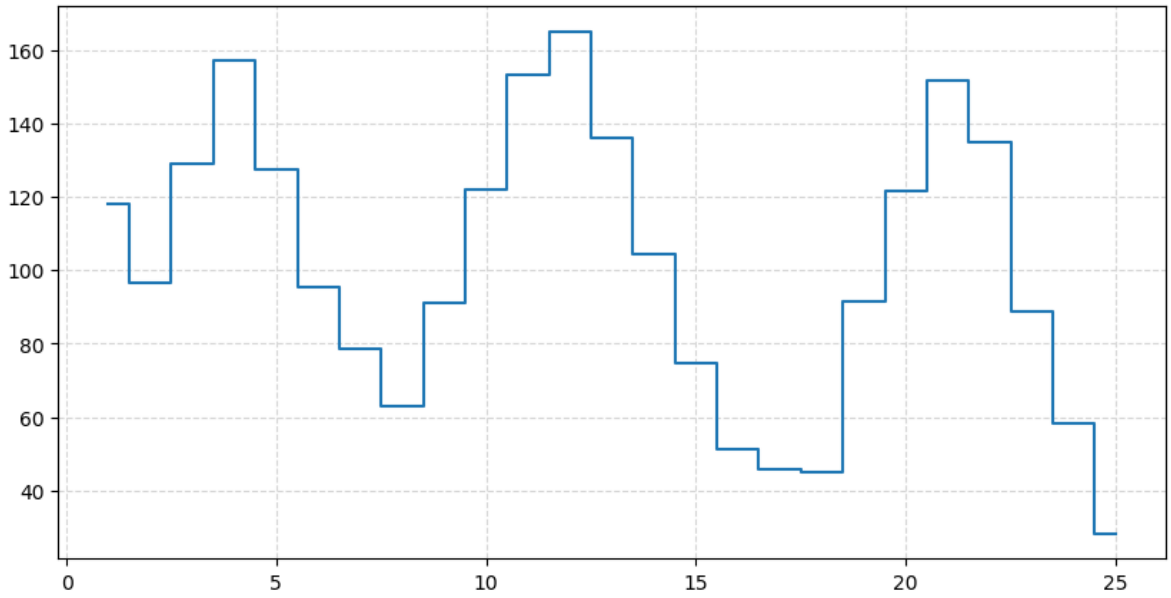
```
# GECAM的daily事例文件较大，如果日常分析只需要其中一段时间的数据，可裁切出一段数据进行保存。

trig_met = evt.info.trig_met

# 每个探头都只保存crop_time_range时间段的数据，其他信息保持不变
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 根据源的入射角选择探头

```
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

# 读取posatt文件
posatt_path=r"test_gecam_data/gb_posatt_tn210511_112749_v00.fits"
posatt_obj = PosAtt.open(posatt_path)

# 选定met时间，或使用触发时间 evt.info.trig_met
choose_met = evt.info.trig_met
# satallite: b/c
choose_satellite="b"

# 设定源坐标（J2000）
# 或取出触发evt文件的header中存储的源坐标: ra,dec,err_radius=evt.info.loc
ra,dec,err_radius=evt.info.loc


# 获取时间点对应的卫星姿态（四元数）
# TODO: 注意：
# For GECAMC(HEBS)， MET(59537900)前后的姿态四元数的格式错误，暂未修复
# pre_quat = ['Q2', 'Q3', 'Q4', 'Q1'] # until 59537900
# quat = ['Q1', 'Q2', 'Q3', 'Q4'] # from 59538144
quat = posatt_obj.get_quat(choose_met)

# 如果GECAMC的met<59537900
# 需要手动调整，quat=[quat[3],quat[0],quat[1],quat[2]]
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)
    # 必须设置探头所属的卫星， 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')
```



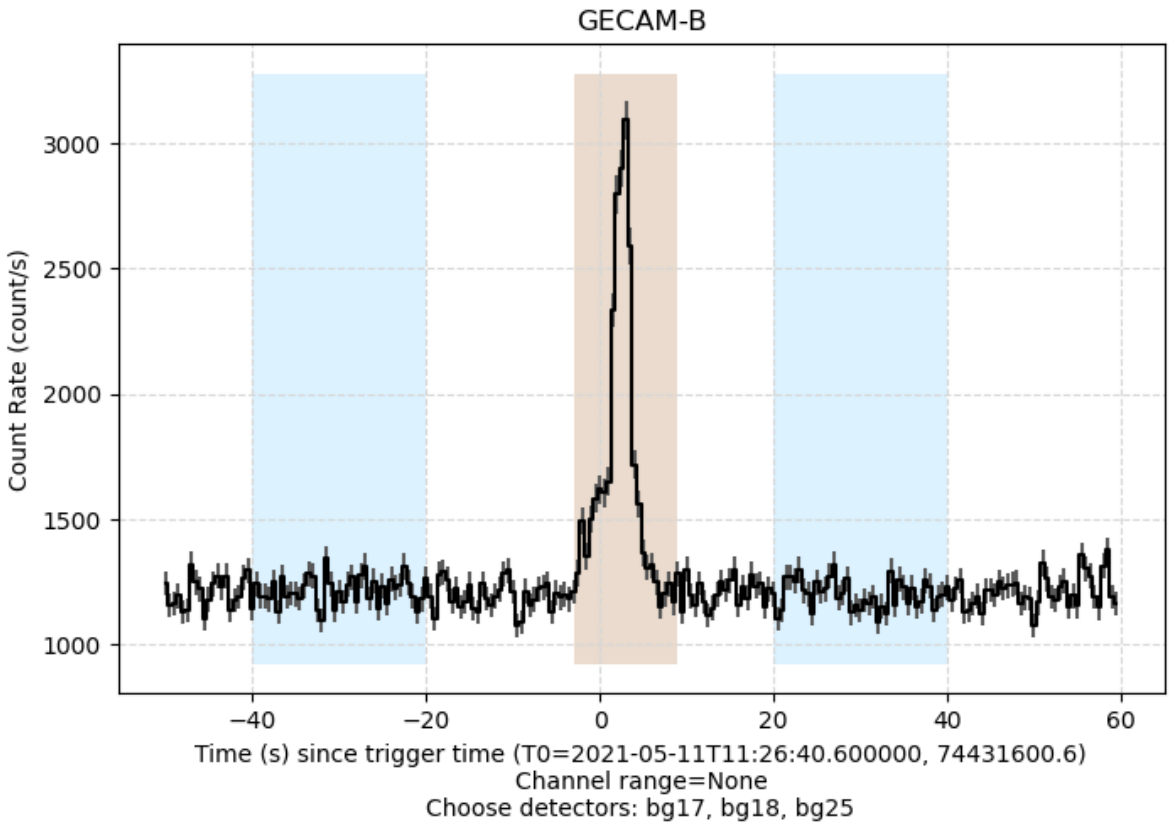
选择源入射角度最小的几个探头：[17,18,25]

2.2 查看多个探头的叠加光变

```
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],# 自定义分能段的光变
```

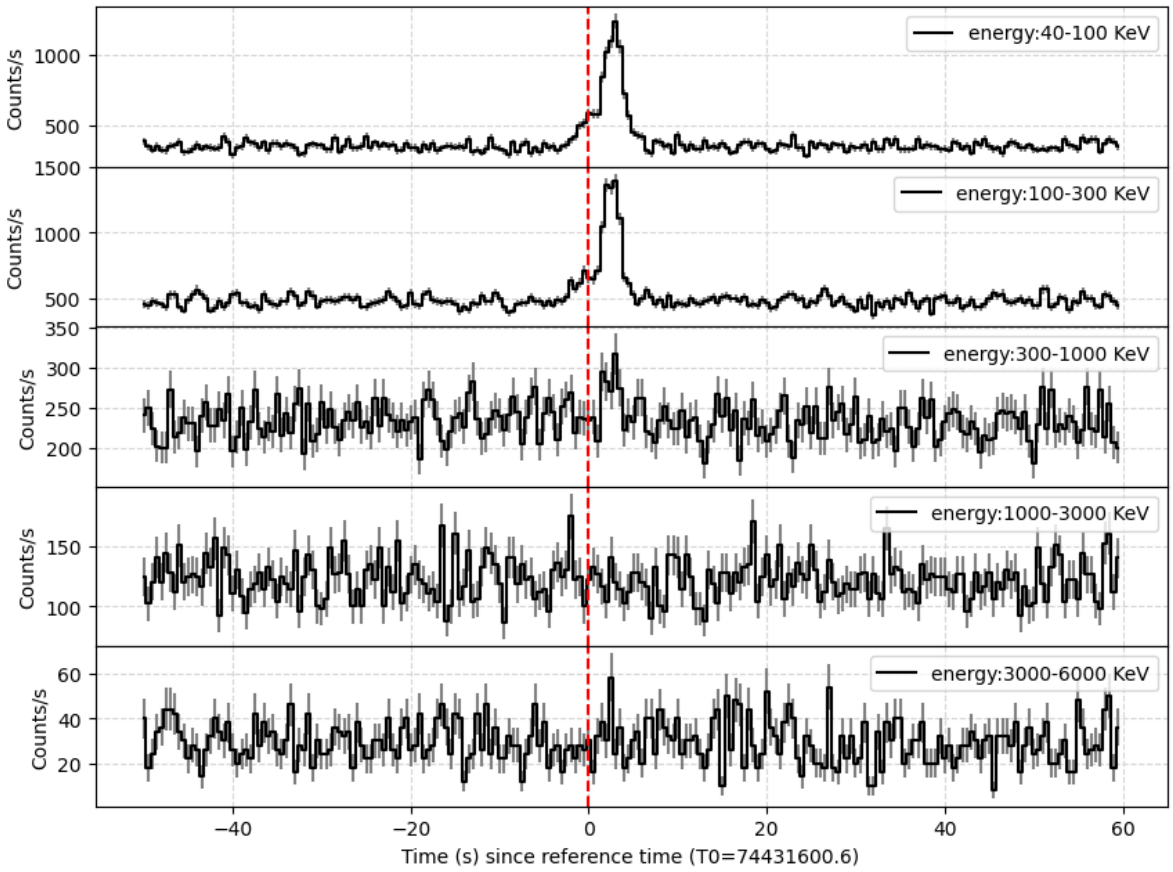
```
    # "channel_bin":1  # 自定义分能道的光变，当energy_bin和channel_bin同时存在时，
channel_bin优先级更高
}
# 画出源区间（src_range）和本底区间（bg_range）以辅助选取时间范围
fig_kwargs_dic = {
    "bg_range": [[trig_met - 40, trig_met - 20],
                 [trig_met + 20, trig_met + 40]],
    "src_range": [trig_met - 3, trig_met + 9]
}

# # 画多探头的叠加光变
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 提取分能段的光变数据并绘图

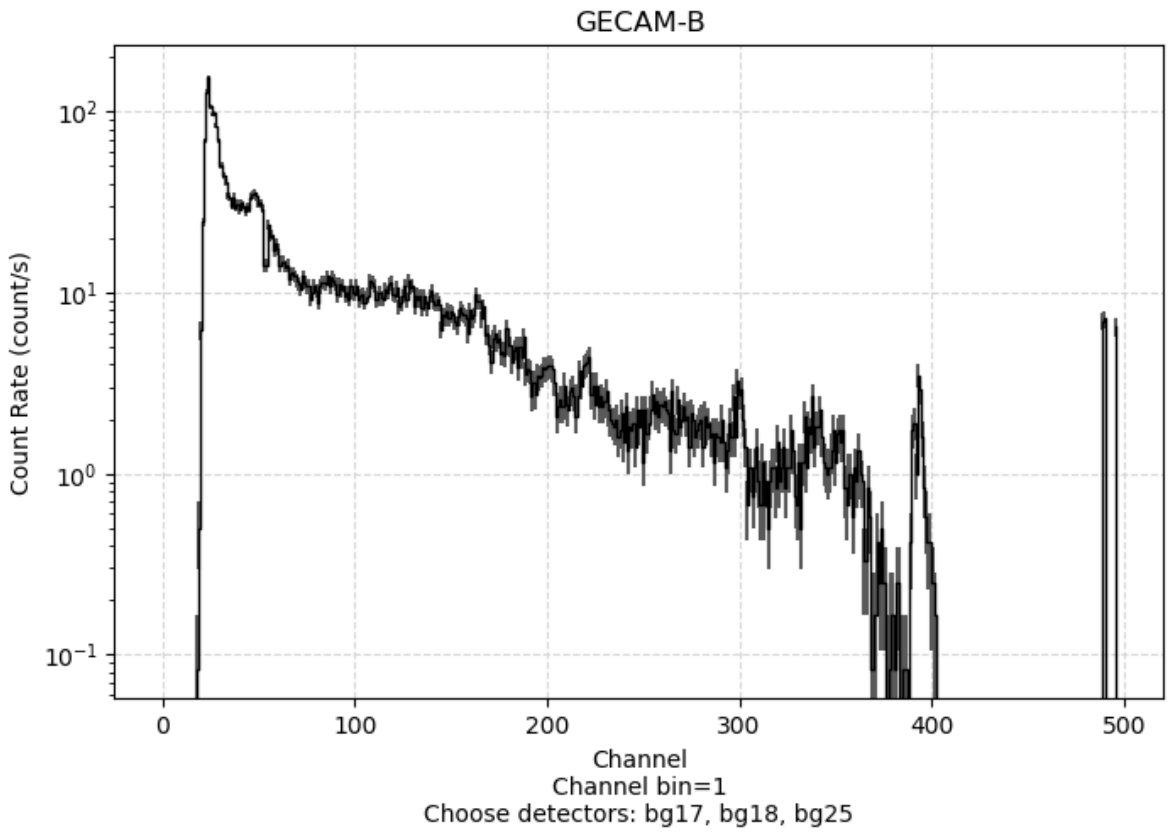
```
import numpy as np

det_y_list = []
for det_lc_obj in dets_lc_list:
    # 获取分能段的光变曲线，correct_by_dead_time=True，表示修正死时间
    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"]

# 合并多个探头的光变
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
    # 转换为计数率
    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 查看多个探头的叠加能谱

```
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 = {}

# 画多探头的叠加的能谱
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 生成多个探头的响应文件

```python
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
# 读取posatt文件
posatt_path=r"test_gecam_data/gb_posatt_tn210511_112749_v00.fits"
posatt_obj = PosAtt.open(posatt_path)

# 选定met时间，或使用触发时间 evt.info.trig_met
choose_met = evt.info.trig_met
# satallite: b/c
choose_satellite="b"

# 选择探头(必须设定增益,high 或者 low)
choose_det = GRD(18, "high")
# 必须设置探头所属的卫星， a/b
choose_det.set_satellite(choose_satellite)

# 设定源坐标（J2000）
```

```
# 或取出触发evt文件的header中存储的源坐标: ra,dec,err_radius=evt.info.loc
ra,dec,err_radius=evt.info.loc

# 获取时间点对应的卫星姿态（四元数）
# TODO: 注意:
# For GECAMC(HEBS), MET(59537900)前后的姿态四元数的格式错误，暂未修复
# pre_quat = ['Q2', 'Q3', 'Q4', 'Q1'] # until 59537900
# quat = ['Q1', 'Q2', 'Q3', 'Q4'] # from 59538144
quat = posatt_obj.get_quat(choose_met)

# 如果GECAMC的met<59537900
# 需要手动调整，quat=[quat[3],quat[0],quat[1],quat[2]]
if choose_satellite=="c" and choose_met<=59537900:
    quat=[quat[3],quat[0],quat[1],quat[2]]


# 根据源的坐标和卫星姿态计算入射角（deg）
theta, phi = radec_to_thetaphi(ra, dec, quat,satellite=choose_satellite)

# 必须区分事例数据还是并道数据， 事例：evt 能谱并道: bspec 时间并道：btime
# evt/bspec/btime
event_type="evt"

# 设置响应文件输出的文件夹
out_dir = "./test_rsp/"

# 生成单个探头（单个响应）响应文件，返回响应文件的绝对路径
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': 744316000.6}
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'
```

```
# 生成多个探头的响应文件（必须区分高低增益）
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 生成多个探头的能谱文件

```
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:对应于选择的探头的响应文件，区分高低增益
    "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()
```

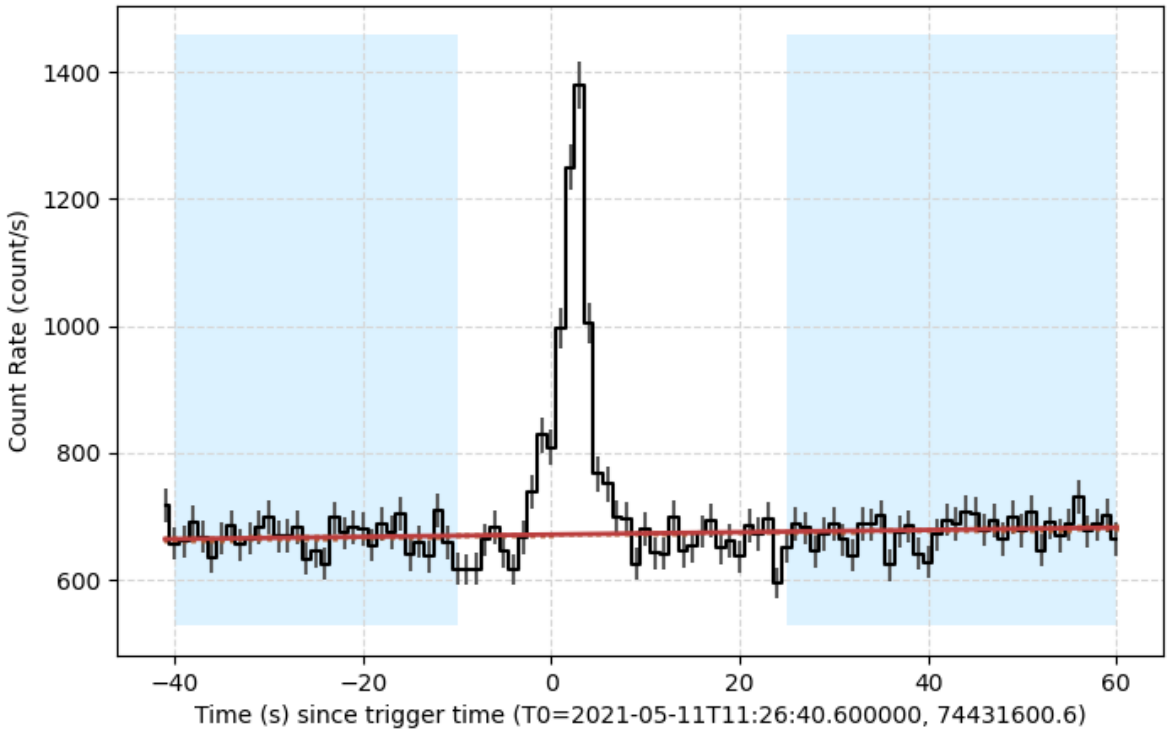
```
dict_keys(['bg18H', 'bg18L', 'bg25H'])
```

```
# 查看bg18H的能谱数据
spec_data_bg18H = spec_data.get("bg18H")
# 查看生成能谱前的总光变和本底光变
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")

# 查看第一个时间段源的能谱数据
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>]
```

```
# 示例：检查bg18H在拟合光变时是否存在错误（参考后续的细致分析）
det_sliced_c_lc_fig = LightCurveFigure(bg18H_lc.get_plot_data(),
trig_time=trig_met)
# 标记本底范围的阴影
det_sliced_c_lc_fig.add_background(bg18H_bg_lc.get_plot_data(),
                                   bg_time_range=bg18H_bg_lc.bg_time_range,
label="bg")

# 查看bg18H的本底拟合评价
# channel_range=[10,200]
# bg18H_bg_lc.show_fitting_quality(channel_range=channel_range)
# plt.show()
```




```
# 能谱数据
# channel的每个bin的边界
channel_bins = spec_bg18H.channel_bins
# energy的每个bin的边界
energy_bins = spec_bg18H.energy_bins
# 每个channel bin的计数
counts = spec_bg18H.counts
# 每个channel bin的计数误差
count_err = spec_bg18H.counts_err
```

3 单个探头细致分析

3.1 读取事例文件

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

3.2 过滤出单个探头的数据

```
# 过滤出探头18的数据
det_events = evt.select_detector(18)
```

3.3 对于单个探头的事例，进一步过滤

3.3.1 如果后续将使用该数据生成能谱文件，则必须过滤增益，不能选择全增益

```
# 对于探头数据进一步的数据过滤，增益、时间范围、能量范围、能道范围、是否只使用推荐事例

# 根据增益过滤（如需用于生成能谱，必须设定增益），both:全增益， high:高增益， low：低增益
# gain_type=None,

# 根据时间过滤
#time_range=None,

# 过滤能量范围或能道范围，不同时进行，如果channel_range不为None,则只过滤channel
#energy_range=None,
#channel_range=None,

# 是否只选取推荐事例
#only_recommend=True
det_sliced_events = det_events.slice(gain_type="high", only_recommend=True)
```

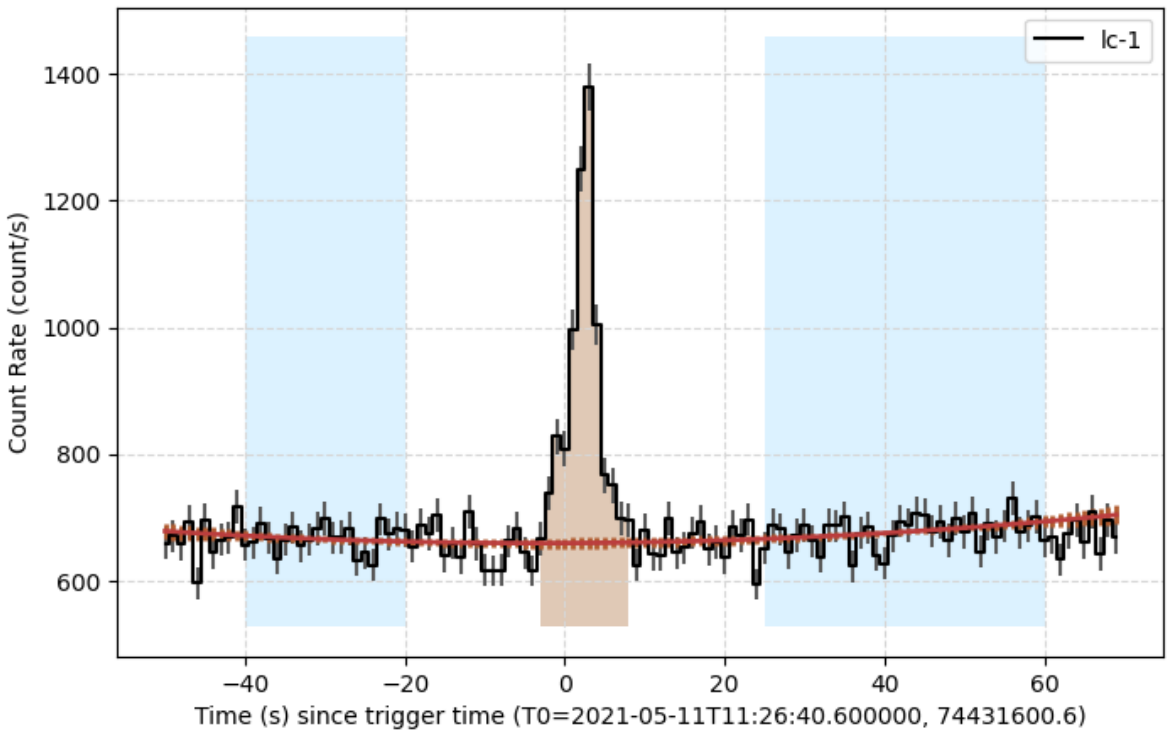
3.4 从过滤后的数据中提取分能段的光变

```
# 提取触发时间
trig_met = evt.info.trig_met

# 光变的总时间范围（绝对时间）
lc_time_range = (trig_met - 50, trig_met + 70)

# 定义源时间段
src_time_range = [trig_met - 3, trig_met+9]
# 定义本底时间段
bg_time_range_list = [[trig_met - 40, trig_met - 20],
                      [trig_met + 25, trig_met + 60]]

# 时间bin，单位秒
time_bin = 1
# 能道分bin，整数为均匀分bin，一维列表为自定义分bin
channel_bin = 1

det_sliced_lc = det_sliced_events.to_light_curve(lc_time_range, time_bin,
channel_bin)
# 本底拟合，拟合阶次为2
det_sliced_bg_lc = det_sliced_lc.fit_background(bg_time_range_list, fit_order=2)
# 提取源时间段的光变
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(
```

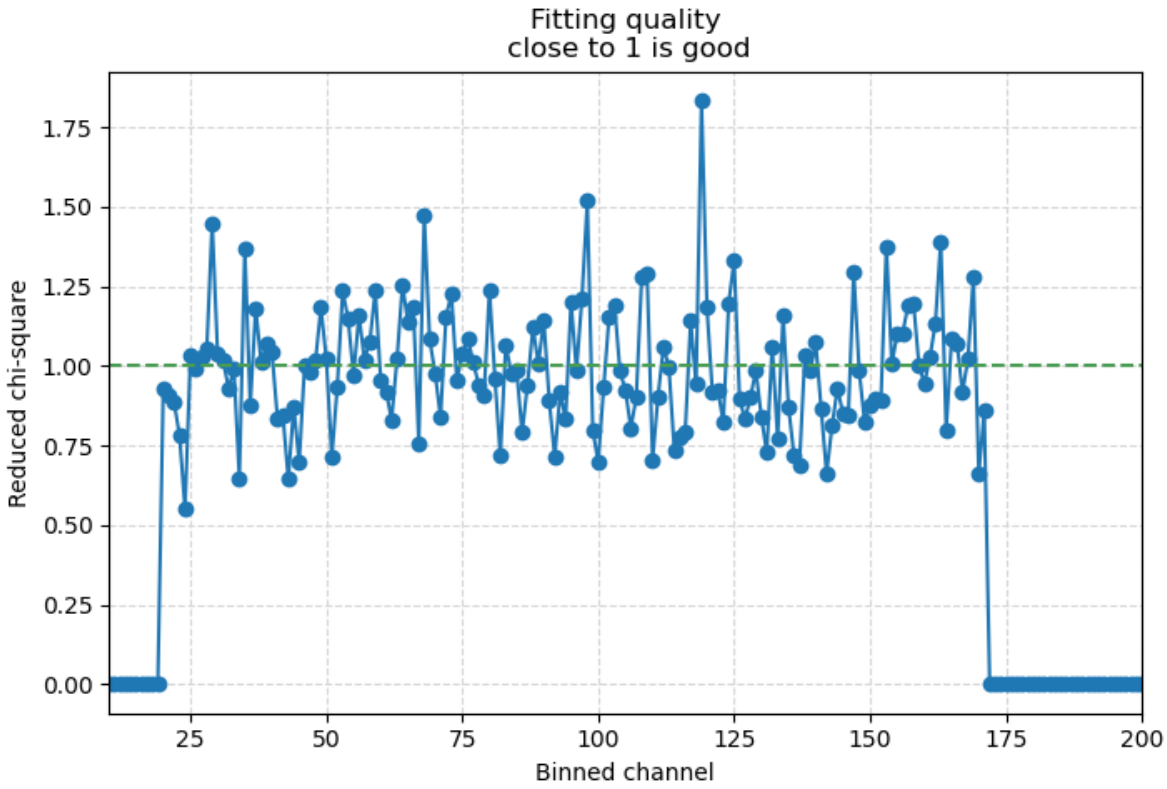
3.5 画出合并所有能段的光变

```
# 画出总时长光变（合并各个能段）
det_sliced_lc_fig = LightCurveFigure(det_sliced_lc.get_plot_data(),
trig_time=trig_met, dpi=100)
# 画出本底
det_sliced_lc_fig.add_background(det_sliced_bg_lc.get_plot_data(),
                                 bg_time_range=det_sliced_bg_lc.bg_time_range)
# 画出源时间段的阴影
det_sliced_lc_fig.add_selection(det_src_lc.get_plot_data())
det_sliced_lc_fig.show_legend()
```



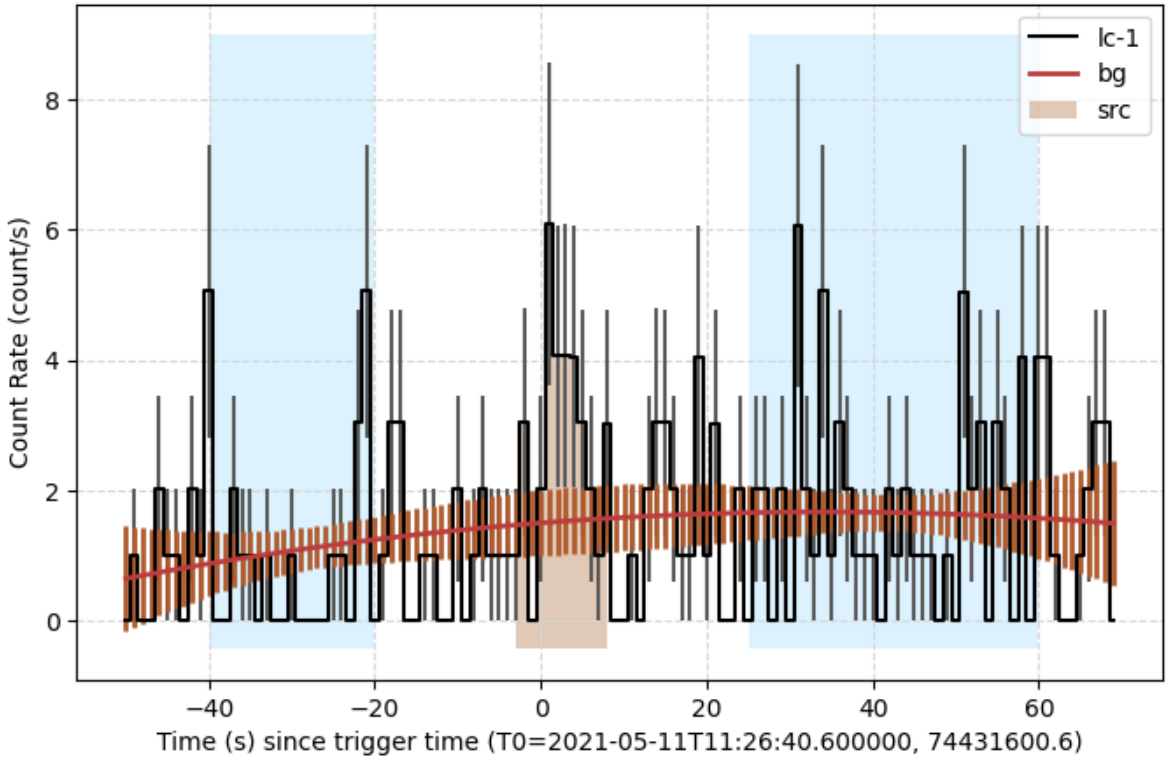
3.6 查看各个能段的拟合结果评价

```
# 选择查看，并道后的第10至200道的拟合结果评价指标，None:为查看所有道
# 当前指标越接近1越好
channel_range=[10,200]
quality_data=det_sliced_bg_lc.show_fitting_quality(channel_range=channel_range)

# 对于某个能道拟合效果不好，
# 则后续程序，可查看该能段的光变，或重新单独拟合该能段的本底
```



3.7 画出能段序号为100的光变

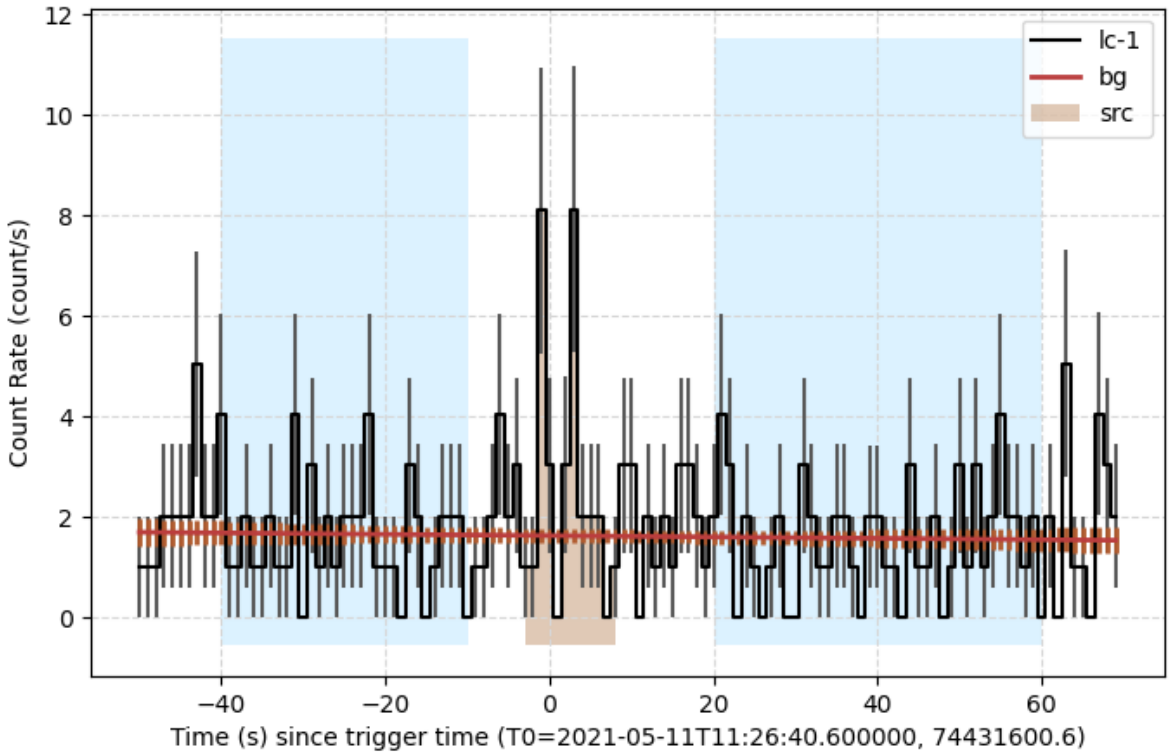
```
# 设置能段序号为100 （如果生成分能段光变时有进行并道，则选取并道后的能段序号）
choose_channel_index = 119

# 过滤选取能段的光变
channel_lc = det_sliced_lc.get_channel_lc(choose_channel_index) # 总长度光变
channel_src_lc = det_src_lc.get_channel_lc(choose_channel_index) # 源区间光变
channel_bg_lc = det_sliced_bg_lc.get_channel_lc(choose_channel_index) #本底区间光变

# 画光变
det_sliced_c_lc_fig = LightCurveFigure(channel_lc.get_plot_data(),
trig_time=trig_met)
# 标记本底范围的阴影
det_sliced_c_lc_fig.add_background(channel_bg_lc.get_plot_data(),
                                   bg_time_range=channel_bg_lc.bg_time_range,
label="bg")
#  标记源时间段
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 调整单个能段的本底拟合

```
# 选择能段序号为100，（如果生成分能段光变时有进行并道，则选取并道后的能段序号）
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]
```



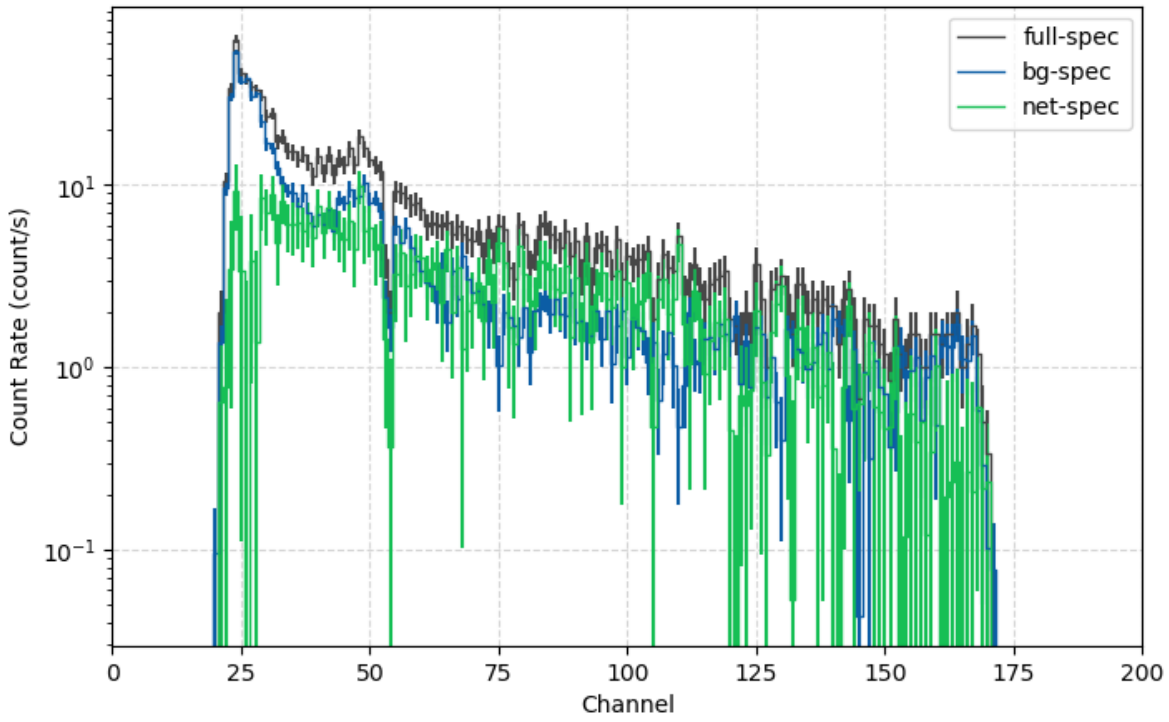
3.9 生成当前探头的能谱

```
# 生成能谱文件的数据,时间分解谱
# det_sliced_lc:  单个探头的光变
# det_sliced_bg_lc:  单个探头的本底光变
from gecam.data.spec import SpecFile

spec_file = SpecFile(det_sliced_lc, det_sliced_bg_lc)

# 添加第一个源时间段
src_time_range=(trig_met -1, trig_met + 5)
spec, bg_spec, net_spec = spec_file.add_src(src_time_range)

# 添加第二个源时间段
src_time_range2 = (trig_met + 5, trig_met + 9)
spec2, bg_spec2, net_spec2 = spec_file.add_src(src_time_range2)
```

3.10 查看能谱

```
from gecam.plot.spectrum import SpectrumFigure

# 画出能谱文件的数据（当前选择第一段源时间段的能谱）
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)

# 截断显示
spec_fig.set_xlim([0, 200])
# spec_fig.set_yscale("linear")
spec_fig.show_legend()
```



3.10 输出能谱到能谱文件中

```
# 能谱对应的响应文件
rsp_path = "test.rsp"
out_dir=r"./"
spec_file.write(out_dir, rsp_path=rsp_path)
```

爆发现象分析

爆发时间估计 (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)
```

生成净光变的累计计数曲线

```
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]#推荐的能量范围
}
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
```

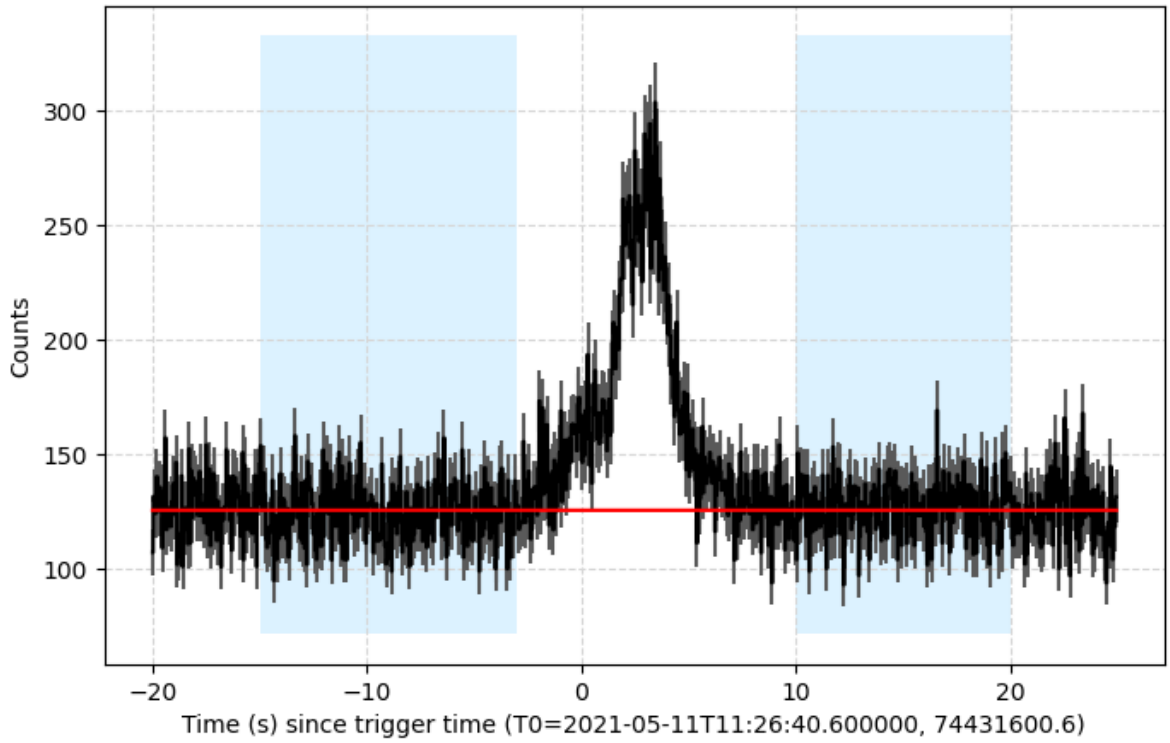
```
lc_dic.keys()
```

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

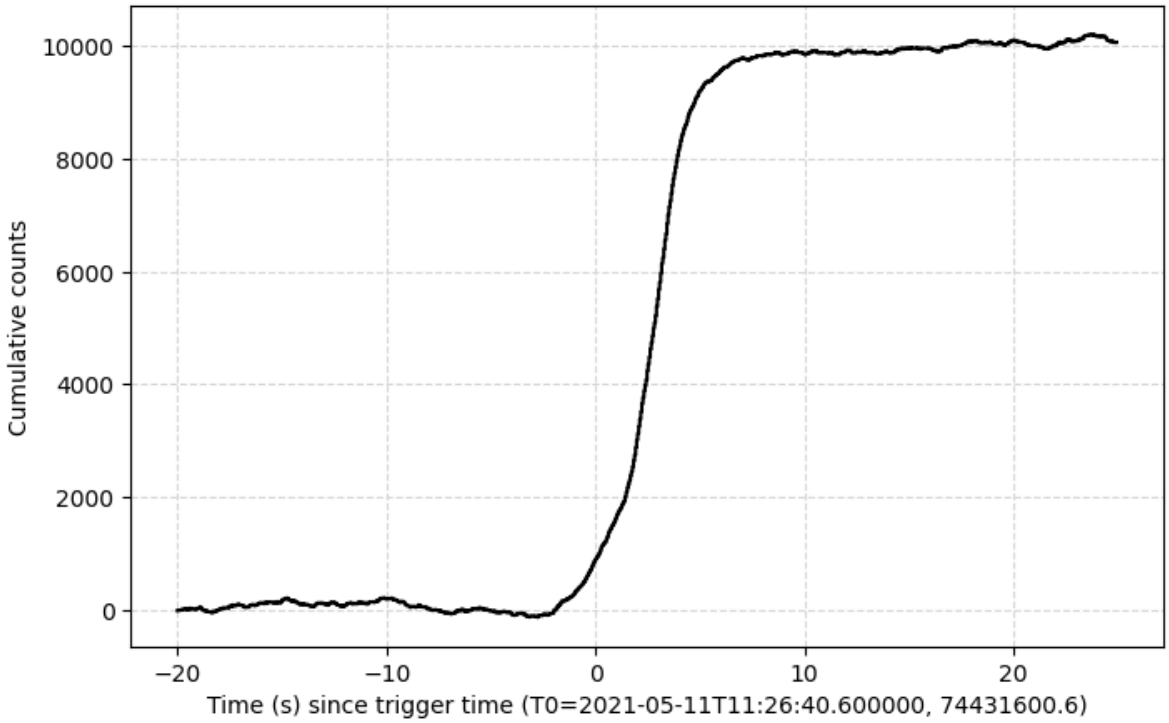
```
# lc_dic["bg18H"].total_lc.dead_time_on_bins
```

```
## 查看净光变

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")
```




```
# 查看净光变的累计计数曲线，方便选取累计曲线的本底范围
set_time_range=None
cumsum_lc_fig =
duration_obj.plot_light_curve_cumsum(set_time_range=set_time_range)
```



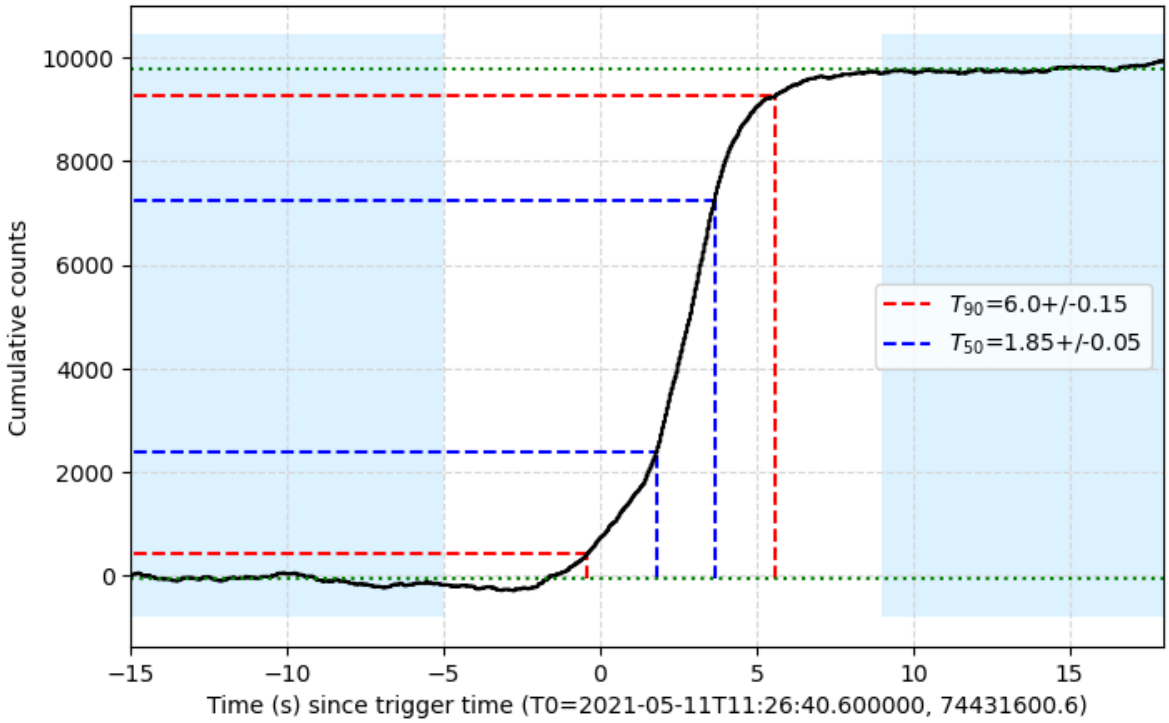
计算T90,T50

```
# 累计曲线的本底范围
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
```

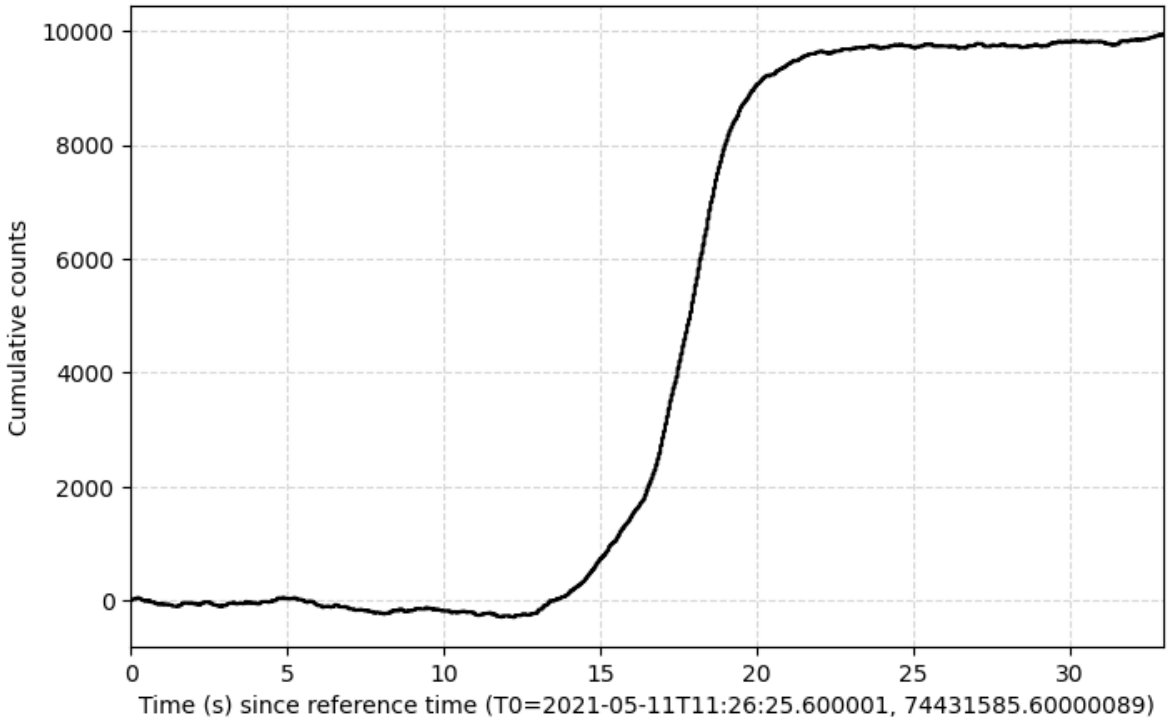
画出T90, T50的范围

```
# set_time_range 用于限制图像的起始时间

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)
```



自定义数据来估计T90, T50

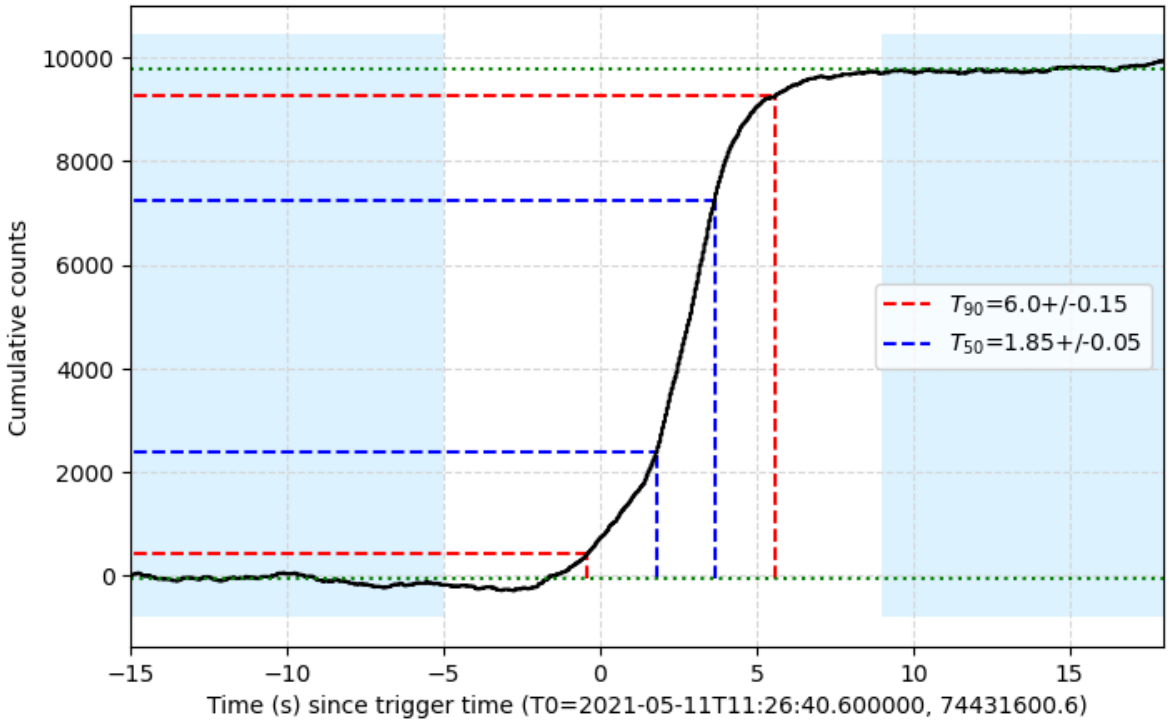
```
duration_obj3 = BurstDuration()

# 准备数据：总光变 total_lc 和净光变 net_lc (当前借用上节生成的光变作为示例)
# 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)

# 光变的时间可能不是默认的GECAM的met时间，可重写xlabel
cumsum_lc_fig3_2.set_xlabel(f"Time (s) since trigger time (T0=
{GecamMet(trig_met).iso}, {trig_met})")
# 保存图像
# 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