ctools - Feature #2961

Explore the possibility to have a tool/script for spectral component separation

07/15/2019 12:08 PM - Tibaldo Luigi

Status:	Closed	**Start date:**	07/15/2019
Priority:	Normal	**Due date:**	
Assigned To:	Tibaldo Luigi	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.7.0		

Description

Spectral component separation makes it possible to derive the morphology of a component from the data based on assumptions about its spectrum (in the same way that you determine the spectrum based on hypotheses about the morphology when you make an SED).

History

#1 - 01/17/2020 09:59 AM - Tibaldo Luigi

- Assigned To set to Tibaldo Luigi

#2 - 01/23/2020 04:44 PM - Tibaldo Luigi

- Status changed from New to In Progress

- % Done changed from 0 to 10

Modified code generator to work with both Python 2 and 3

- print replaced with print function
- raw_input replaced with input from the builtins package

#3 - 01/28/2020 11:08 AM - Tibaldo Luigi

- % Done changed from 10 to 30

I have added in gammalib a 'flux' method for GModelSpatial that returns the flux integrated over a circular ROI. This is necessary to estimate the flux after analysis on a small ROI. For the moment the integral is always done on two pi in azimuth. A method to calculate the arclength of the overlap between two circular regions is already implemented in GCTASupport.

I have a minimally working script for binned and unbinned observations.

TO DO:

- decide whether to allow only uniform observation sets as for csspec SLICE method (would simplify script)
- OnOff analysis method
- parallelization
- upper limits
- likelihood profiles?
- test routines
- documentation (reference manual, tutorial)

#4 - 01/28/2020 06:09 PM - Tibaldo Luigi

- % Done changed from 30 to 40

The flux integration over circular ROI now is fully implemented. This required to move the method to calculate the arclength of a circle intersection with a circular ROI from GCTASupport to GTools. In this way the flux is computed only in the overlap region between model and ROI. I verified on a few test cases the results are consistent with expectations. A strange error from the test routines:

```
3 errors occured in module "sky":
 Error in Test GSkyRegions region access: *** ERROR in GPythonTestSuite::test: <class 'AttributeError'>'SwigPyObject' object has no attribute
 'name' (N16GPythonException10test_errorE)
 Error in Test GSkyRegions slicing: *** ERROR in GPythonTestSuite::test: <class 'AttributeError'>'SwigPyObject' object has no attribute 'name'
 (N16GPythonException10test_errorE)
 Error in Test sky class pickeling: Test pickeling of "GSkyRegions" class.: Error in pickeling "GSkyRegions" (can't pickle SwigPyObject objects). ()
```

Jürgen is looking into it.

For the cscript I implemented parallelization with the standard methods in mputils.
I refashioned the code to only accept in input homogeneous sets of either unbinned or binned observations. This makes it more consistent with csspec, slightly simplify the code, and makes it straightforward to implement the On/Off analysis (to be done next).

#5 - 01/28/2020 10:30 PM - Knödlseder Jürgen

The Python interface gets screwed up when adding

```
double flux(const GSkyRegionCircle& reg,
            const GEnergy&          srcEng = GEnergy(),
            const GTime&            srcTime = GTime()) const;
```

to GModelSpatial.i. Before adding this code the snipped

```
>>> import gammalib
>>> regions = gammalib.GSkyRegions()
>>> circle  = gammalib.GSkyRegionCircle()
>>> for i in range(10):
...     circle.name('%s' % i)
...     regions.append(circle)
...
```

gave

```
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cc00> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cba0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cc00> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cba0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cc00> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cba0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cc00> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cba0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cc00> >
```

<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x102b2cba0> >

but after adding the flux() method to the Python interface the same snipped gave

<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1c00>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1ba0>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1c00>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1ba0>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1c00>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1ba0>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1c00>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1ba0>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1c00>
<Swig Object of type 'GSkyRegionCircle *' at 0x1024d1ba0>

#6 - 01/29/2020 09:51 AM - Knödlseder Jürgen

Interestingly, when importing the model module before sky in __init__.py of gammalib, i.e.

```
# Import modules
from gammalib.app import *
from gammalib.base import *
from gammalib.fits import *
from gammalib.linalg import *
from gammalib.model import *       <= model first
from gammalib.numerics import *
from gammalib.obs import *
from gammalib.opt import *
from gammalib.sky import *         <= sky follows
from gammalib.support import *
```

the Proxy class does not exist:

<Swig Object of type 'GSkyRegionCircle *' at 0x10970bbd0>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bb70>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bbd0>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bb70>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bbd0>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bb70>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bbd0>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bb70>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bbd0>
<Swig Object of type 'GSkyRegionCircle *' at 0x10970bb70>

but when importing sky before model, i.e.

```
# Import modules
```

```
from gammalib.app import *
from gammalib.base import *
from gammalib.fits import *
from gammalib.linalg import *
#from gammalib.model import *
from gammalib.numerics import *
from gammalib.obs import *
from gammalib.opt import *
from gammalib.sky import *
from gammalib.model import *
from gammalib.support import *
```

is does

```
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0dbd0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0db70> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0dbd0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0db70> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0dbd0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0db70> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0dbd0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0db70> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0dbd0> >
<gammalib.sky.GSkyRegionCircle; proxy of <Swig Object of type 'GSkyRegionCircle *' at 0x10dc0db70> >
```

#7 - 01/29/2020 09:56 AM - Knödlseder Jürgen

I played a bit around with the output typemap in sky.i. The original typemap code is:

```
%typemap(out) GSkyRegion* {
    char classname[80];
    strcpy(classname, "_p_");
    strcat(classname, result->classname().c_str());
    swig_type_info *myinfo = SWIGTYPE_p_GSkyRegion;
    swig_cast_info *mycast = 0;
    mycast = myinfo->cast;
    while (mycast != 0) {
        if (strcmp(classname, mycast->type->name) == 0) {
            myinfo = mycast->type;
            break;
        }
```

```
        mycast = mycast->next;
    }
    $result = SWIG_NewPointerObj(SWIG_as_voidptr($1), myinfo, myinfo, 0 |  0);
}
```

and changing the SWIG_NewPointerObj attribute to

```
$result = SWIG_NewPointerObj(SWIG_as_voidptr($1), myinfo, SWIG_POINTER_OWN);
```

results in

```
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71c00>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71ba0>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71c00>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71ba0>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71c00>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71ba0>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71c00>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71ba0>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71c00>
swig/python detected a memory leak of type 'GSkyRegionCircle *', no destructor found.
<Swig Object of type 'GSkyRegionCircle *' at 0x10ac71ba0>
```

The behavior is explained here: https://stackoverflow.com/questions/7223327/what-does-the-last-argument-to-swig-newpointerobj-mean
Not sure it is of any help.

#8 - 01/29/2020 11:56 AM - Knödlseder Jürgen

- % Done changed from 40 to 50

Here is my analysis of the problem, which is still a guess of whats going on:

The current python module first imports the model module and then the sky module. Adding the flux() method leads to the addition of a new SWIG type with associated type cast from the GSkyRegionCircle class to the GBase class. An identical type is latter also implemented in the sky module, hence it actually exists twice. Specifically, the function

```
static void *_p_GSkyRegionCircleTo_p_GBase(void *x, int *SWIGUNUSEDPARM(newmemory)) {
    return (void *)((GBase *) (GSkyRegion *) ((GSkyRegionCircle *) x));
}
```

exists both in model_wrap.cpp and sky_wrap.cpp files and both are registered in the local

```
static swig_cast_info _swigc__p_GBase[] = { ... }
```

array.

However, the sky_wrap.cpp file later implements also a type cast from the GSkyRegionCircle class to the GSkyRegion class. This leads to a notable difference. In model_wrap.cpp one can find

```
static swig_cast_info _swigc__p_GSkyRegion[] = {{&_swigt__p_GSkyRegion, 0, 0, 0},{0, 0, 0, 0}};
```

while in sky_wrap.cpp there is

```
static swig_cast_info _swigc__p_GSkyRegion[] = {  {&_swigt__p_GSkyRegionMap, _p_GSkyRegionMapTo_p_GSkyRegion, 0, 0},
 {&_swigt__p_GSkyRegionCircle, _p_GSkyRegionCircleTo_p_GSkyRegion, 0, 0},  {&_swigt__p_GSkyRegion, 0, 0, 0},{0, 0, 0, 0}};
```

.

Adding explicitly in model_wrap.cpp the following

```
//static swig_cast_info _swigc__p_GSkyRegion[] = {{&_swigt__p_GSkyRegion, 0, 0, 0},{0, 0, 0, 0}};
static void *_p_GSkyRegionCircleTo_p_GSkyRegion(void *x, int *SWIGUNUSEDPARM(newmemory)) {
    return (void *)((GSkyRegion *)  ((GSkyRegionCircle *) x));
}
static swig_cast_info _swigc__p_GSkyRegion[] = {  {&_swigt__p_GSkyRegionCircle, _p_GSkyRegionCircleTo_p_GSkyRegion, 0, 0},
 {&_swigt__p_GSkyRegion, 0, 0, 0},{0, 0, 0, 0}};
```

solves the issue.
It therefore seems that the problem is that no type cast from GSkyRegionCircle to GSkyRegion exists in model_wrap.cpp, and somehow the Python module seems to use the first type cast information it can find. Interestingly, the type cast for GSkyRegionMap does not get corrupted:

```
>>> import gammalib
>>> regions = gammalib.GSkyRegions()
>>> map  = gammalib.GSkyRegionMap()
>>> for i in range(10):
...     map('%s' % i)
...     regions.append(map)
...
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566c00> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566ba0> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566c00> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566ba0> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566c00> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566ba0> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566c00> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566ba0> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566c00> >
<gammalib.sky.GSkyRegionMap; proxy of <Swig Object of type 'GSkyRegionMap *' at 0x104566ba0> >
```

.

#9 - 01/29/2020 12:08 PM - Knödlseder Jürgen

A relevant reading is here: http://www.swig.org/Doc3.0/SWIGDocumentation.html (see section 11.12). Specifically:

Another issue needing to be addressed is sharing type information between multiple modules. More explicitly, we need to have ONE swig_type_info for each type. If two modules both use the type, the second module loaded must lookup and use the swig_type_info structure from the module already loaded. Because no dynamic memory is used and the circular dependencies of the casting information, loading the type information is somewhat tricky, and not explained here. A complete description is in the Lib/swiginit.swg file (and near the top of any generated file).

#10 - 01/29/2020 12:25 PM - Knödlseder Jürgen

Interestingly, one can switch on debugging of the SWIG module initializer using

```
//#if 0
#define SWIGRUNTIME_DEBUG
//#endif
```

This gives

```
SWIG_InitializeModule: size 49
...
SWIG_InitializeModule: look cast _p_GSkyRegionCircle
SWIG_InitializeModule: found cast _p_GSkyRegionCircle
SWIG_InitializeModule: skip old cast _p_GSkyRegionCircle
...
SWIG_InitializeModule: type 14 _p_GSkyRegion
SWIG_InitializeModule: cast type _p_GSkyRegionMap
SWIG_InitializeModule: cast type _p_GSkyRegionCircle
SWIG_InitializeModule: cast type _p_GSkyRegion
---- Total casts: 3
...
```

upon importing GammaLib. So it looks like the casting array is setup up properly, but it still somehow finds the old cast. Here is what is written in the swiginit.swg header:

```
 * There are three cases to handle:
 *  1) If the cast->type has already been loaded AND the type we are adding
 *     casting info to has not been loaded (it is in this module), THEN we
 *     replace the cast->type pointer with the type pointer that has already
 *     been loaded.
 *  2) If BOTH types (the one we are adding casting info to, and the
 *     cast->type) are loaded, THEN the cast info has already been loaded by
 *     the previous module so we just ignore it.
 *  3) Finally, if cast->type has not already been loaded, then we add that
 *     swig_cast_info to the linked list (because the cast->type) pointer will
 *     be correct.
```

and I'm wondering whether we are actually in case 2: GSkyRegion and GSkyRegionCircle exist already.

#11 - 01/29/2020 02:31 PM - Knödlseder Jürgen

Note that the _p_GSkyRegionCircleTo_p_GBase function in model_wrap.cpp is only included if

%import(module="gammalib.sky") "GSkyRegionCircle.i";

is added to GModelSpatial.i.

#12 - 01/29/2020 04:56 PM - Knödlseder Jürgen

Concluding now the investigation, the only solution I could up with is changing the order of import of Python modules in __init__.py.in. I changed the order and pushed the modifications into the devel branch. Rebasing your branch on the devel branch should fix your issue.

#13 - 01/30/2020 04:53 PM - Tibaldo Luigi

Thanks. This indeed solves the problem. As discussed online I changed the method interface so that it accepts in input a generic GSkyRegion. For the moment only the case of a circle is implemented.
I also added a minimal test with a disk model and three circular regions (not overlapping, fully containing the model, partially overlapping). This part from my point of view is ready and could already be merged in gammalib devel.

Going back to the component separation script now.

#14 - 01/30/2020 06:36 PM - Tibaldo Luigi

- % Done changed from 50 to 60

Implemented On/Off analysis and upper limit computation.

#15 - 01/31/2020 03:55 PM - Knödlseder Jürgen

- % Done changed from 60 to 70

user#266 wrote:

> Thanks. This indeed solves the problem. As discussed online I changed the method interface so that it accepts in input a generic GSkyRegion. For the moment only the case of a circle is implemented.
> I also added a minimal test with a disk model and three circular regions (not overlapping, fully containing the model, partially overlapping). This part from my point of view is ready and could already be merged in gammalib devel.
>
> Going back to the component separation script now.

I merged the code into GammaLib devel.

#16 - 01/31/2020 06:45 PM - Tibaldo Luigi

Implemented methods to pipe results in memory in Python. Cleaned up logging.
Will now focus on testing and documentation.

#17 - 02/03/2020 12:16 PM - Tibaldo Luigi

Unbinned analysis does not work, waiting on #2695 to be implemented.

#18 - 02/06/2020 10:09 AM - Tibaldo Luigi

I implemented in csphagen a snippet that automatically look for the HDU 'EXCLUSION' in the exclusion map file, unless the user has specified the extension number of name. If no extension is specified and there is no extension named 'EXCLUSION' the primary HDU is used.
This enables the user to pass the fits in output from ctskymap directly to csphagen (and all scripts that use csphagen) without worrying about specifying the HDU name.

#19 - 02/13/2020 03:03 PM - Tibaldo Luigi

I realized that there was a problem in the implementation of GModelSpatial.flux for point sources. I modified my branch 2961-csc to address this: now the point source case is treated separately.

#20 - 02/14/2020 11:53 AM - Tibaldo Luigi

- % Done changed from 70 to 80

Added test unit and filled reference manual

#21 - 02/18/2020 04:47 PM - Tibaldo Luigi

- File Figure_1.png added

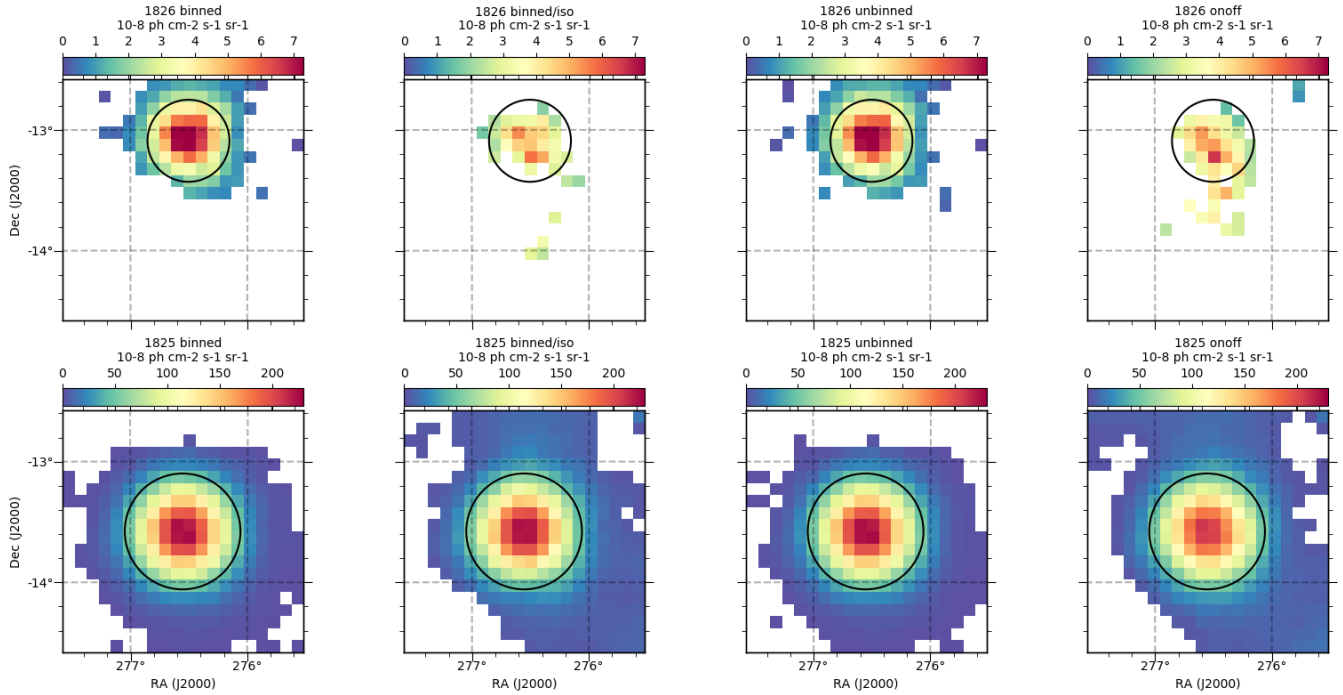
- % Done changed from 80 to 90

I performed a detailed test using the DC-1 dataset. I analysed the region of HESS 1825 and HESS 1826 and derived a flux map for the two sources based on their different spectra.

I selected ~ 15h of observations around the two sources (the dataset is limited in order to be able to compare all analysis methods in a reasonable amount of time). I used as input the true sky model, notably for the source's spectra.

Below you can see flux maps on a 2 deg x 2 deg region for four different test cases. In all cases the final flux map grid spacing is 0.1 deg, and the radius of the ROI for component separation is 0.2 deg. The energy range is 0.1 to 160 TeV.
- Binned analysis (binned cube over a 4 deg x 4 deg region with 0.01 deg spatial grid and 60 logarithmic energy bins).
- Binned analysis in which the true source morphology was not used as prior, but instead within each ROI for component separation emission from the two sources is assumed to be isotropic (binned analysis configuration was the same as before).
- Unbinned analysis.
- On/Off analysis (exclusion map generated with ctskymap over a 4 deg x 4 deg region, with ROI radius of 0.3 deg, inner/outer radius of background ring of 0.5 deg and 0.8 deg, two iterations and a threshold of 5 sigma; WSTAT analysis without background model).

The flux is shown only for bins in which the source TS is > 4. The circles show the 1-sigma contour of the source true Gaussian morphology.

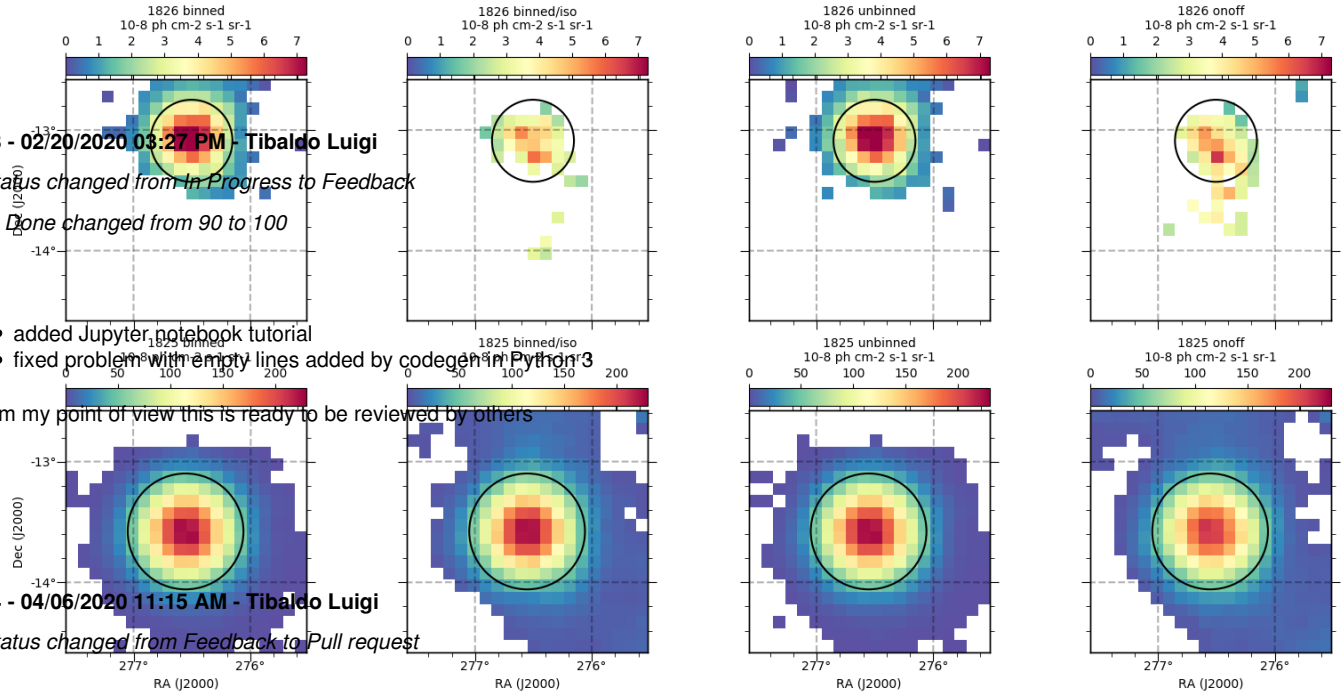


In all cases the script correctly identifies the region from which each spectral component is originated. Even in the two cases in which the prior is uniform (binned/iso and On/Off, the latter by default), the emission region is correctly identified. The On/Off analysis yields somewhat lower peak fluxes for HESS 1825.

#22 - 02/18/2020 06:25 PM - Tibaldo Luigi

- File Figure_1.png added

Colorbar units were wrong, here's the corrected figure.



#23 - 02/20/2020 03:27 PM - Tibaldo Luigi

- Status changed from In Progress to Feedback

- % Done changed from 90 to 100

- added Jupyter notebook tutorial
- fixed problem with empty lines added by codegen in Python 3

From my point of view this is ready to be reviewed by others

#24 - 04/06/2020 11:15 AM - Tibaldo Luigi

- Status changed from Feedback to Pull request

No feedback received, I'm setting this to pull request

#25 - 04/06/2020 12:39 PM - Knödlseder Jürgen

- Status changed from Pull request to Closed

Merged into devel

Files

Figure_1.png	125 KB	02/18/2020	Tibaldo Luigi
Figure_1.png	125 KB	02/18/2020	Tibaldo Luigi