GammaLib - Action #535

Feature # 903 (Closed): Implement a sky region class

Preliminary discussion

10/09/2012 06:27 PM - Deil Christoph

Status:	Closed	**Start date:**	06/23/2013
Priority:	Normal	**Due date:**	
Assigned To:	Deil Christoph	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	HESS sprint #1		

Description

Introduction

Gammalib should provide region (a.k.a. shape) classes because many of the current TeV analysis methods are based on on- and off regions.
E.g. the standard method in HESS to produce spectra is the reflected region background method and the standard method to produce maps is the ring background method [1].

Coordinate transformations / integrating maps over regions is where most CPU time is spent in the HESS software, so it's very important to be efficient here for gammalib / ctools.

Most astronomy software contain regions, e.g. ds9 [2], the HESS software [3], Kapteyn [4, 5, 6], CIAO [7 - 10], pyregion [11], act-analysis [12], pyfact [13], CFITSIO [14 - 15].
I still have to look in detail what concepts / algorithms are useful for us.
Even if we implement everything from scratch these implementations can serve as a reference for unit tests / benchmarks.
Many ctools users will know X-ray data analysis, so playing well with the tools they already know (like dmregions and ds9) would be good.

Definition / Meaning of regions

Regions can either represent areas on the sky or in a 2D cartesian image.

If I understand correctly ds9 and CIAO only implement "image regions" and the HESS software only implements "sky regions".

In Kapteyn all shapes are subclasses of Polygon [5], because it implements the transformation of shapes from one image onto another, and when going e.g. from FOV coordinates like DETX--DETY to RA--DEC or e.g. from a CAR map to an AIT map, any simple geometrical shape like a circle will not be a simple geometrical shape on the new image any more and can only be represented by a polygon or mask.

This part is completely unclear to me, please comment:

- We need both "image regions" and "sky regions" in gammalib, right?
- Do we need ImageRing and SkyRing classes with the image version based on cartesian coordinates and the sky version on spherical coordinates, or can both cases be combined in one class?
- How should region classes (especially the mask) couple with the position and image classes in gammalib?
- Do we need region reprojection (resampling in the case of the mask region) onto other images?

Types of regions

To get started I would propose to only implement the geometrical regions commonly needed for TeV analysis:

- CircleSegment (part of a circle between an inner / outer radius and min / max position angle)
- Circle
- Ring
- Box (nice test case for rotations, because not radially symmetric around center)

as well as regions to specify irregular and complex shapes like e.g. exclusion regions:

- Polygon

- Mask (i.e. image with 0 and 1 entries)
- Compound (a list of regions)

Some, but not much code could be saved by using inheritance:

Region
-- CircleSegment -- Ring -- Circle
-- Polygon -- Box
-- Mask
-- Compound

It is unclear to me at the moment if the gains of using inheritance for Ring, Circle and Box (is-a relationship, avoid some code duplication) outweigh the drawbacks (most methods are re-implemented for the special cases anyways for speed, a Circle should not have a Get/SetOuterRadius method). As described above, in Kapteyn every region is a polygon.

Members

Regions should have the following members:

-- Position (either cartesian X, Y or sky lambda, beta)
-- Some floats for CircleSegment, Ring, Circle, Box
-- List of (x, y) floats for Polygon
-- List of regions for Compound
-- Binary image for Mask

Methods

Regions should implement the following methods:

-- Region::Region(region_string)
-- string Region::ToString()
-- Polygon Region.ToPolygon()

-- Region.GetCenter()
-- Region.GetArea()
-- Region.GetEquivalentCircleRadius()
-- Region.GetBoundingBox()

-- Region.Contains(point)
-- Region.Contains(other_region)
-- Region.Overlaps(other_region)

-- Region.Rotate(point, system)

Images should implement the following methods:

-- Image.GetSum(region) -- sum up pixel values in the region
-- Image.GetIntegral(region) -- like sum, but multiplied by bin area
-- Image.GetMask(region) -- pixels inside the region are 1, outside 0
-- Image.GetNPixelsInside(region) -- Image.GetMask(region).Sum()

I guess cubes (or any array with two spatial dimensions) should provide similar methods.

Use cases

Reflected region background method

Implement reflected background analysis (see [1, 12, 13] above if you don't know how it works).

Ring background method

Implement ring background analysis (see [1, 12, 13] above if you don't know how it works).

Discussion

We will discuss on regions, on-off analysis and other issues:
http://www.doodle.com/yedn35ia7dkz8u3p

Please post any questions / comments / suggestions you have!

References

[1] http://adsabs.harvard.edu/abs/2007A%26A...466.1219B
[2] http://hea-www.harvard.edu/RD/ds9/ref/region.html
[3] http://www.mpi-hd.mpg.de/hfm/HESS/intern/Software/SASH-Docs/classTools_1_1Region.html
[4] http://www.astro.rug.nl/software/kapteyn/maputilstutorial.html#sky-polygons
[5] http://www.astro.rug.nl/software/kapteyn/shapes.html
[6] http://www.astro.rug.nl/software/kapteyn/maputilstutorial.html?highlight=flux#interactive-plotting-of-shapes-for-flux-etc
[7] http://cxc.harvard.edu/ciao/threads/regions/
[8] http://cxc.harvard.edu/ciao/ahelp/region.html
[9] http://cxc.harvard.edu/ciao/ahelp/dmregions.html
[10] http://cxc.harvard.edu/ciao/ahelp/dmfiltering.html
[11] http://leejjoon.github.com/pyregion/
[12] https://bitbucket.org/kosack/act-analysis
[13] https://github.com/mraue/pyfact/blob/develop/pyfact/map.py
[14] http://heasarc.gsfc.nasa.gov/fitsio/c/c_user/node102.html
[15] http://heasarc.gsfc.nasa.gov/lheasoft/ftools/headas/ftselect.html

History

#1 - 10/10/2012 10:15 AM - Deil Christoph

Apparently I can't edit the ticket description, so this is not a good place for a document that evolves while we discuss.

On github e.g. pull requests that never get merged are a good place for such documents:
https://github.com/astropy/astropy/pull/370

How should we do it for gammalib?
Wiki? gammlib repo (maybe in a branch that never gets merged)?

#2 - 10/10/2012 08:48 PM - Knödlseder Jürgen

- Target version set to HESS sprint #1

#3 - 10/10/2012 09:28 PM - Knödlseder Jürgen

Have you tried clicking on Update and the on More after Change properties? This should allow for modifications of the ticket descriptions. Please check.

Otherwise, there is always Wiki and also Forums for discussion.

Under https://cta-redmine.irap.omp.eu/projects/gammalib/wiki/Development_notes I started to create a master Wiki page for GammaLib development notes. One thing we have to decide upon is we need a specific HESS instrument class, or if we manage to make the code sufficiently general to digest both HESS and CTA data. I comment on this issue in feature #538 which we may use for discussing this issue.

#4 - 10/10/2012 09:48 PM - Deil Christoph

Jürgen Knödlseder wrote:

> Have you tried clicking on Update and the on More after Change properties? This should allow for modifications of the ticket descriptions. Please check.

This is exactly what I was looking for. Thanks!

#5 - 10/10/2012 10:27 PM - Knödlseder Jürgen

Christoph Deil wrote:

> Coordinate transformations / integrating maps over regions is where most CPU time is spent in the HESS software, so it's very important to be efficient here for gammalib / ctools.

Fully agree. Here pre computations and value caching a the key to get maximum speed. This is for example already implement in the GSkyDir which for example only transforms once from celestial to galactic coordinates and from then on stores both results.

> In Kapteyn all shapes are subclasses of Polygon [5], because it implements the transformation of shapes from one image onto another, and when going e.g. from FOV coordinates like DETX--DETY to RA--DEC or e.g. from a CAR map to an AIT map, any simple geometrical shape like a circle will not be a simple geometrical shape on the new image any more and can only be represented by a polygon or mask.

Maybe we need different forms and a generic interface to deal with all of them. In case we have a simple form, computations may be much faster when the analytic properties of the shape are used. Having a generic interfaces and then implement different shapes would also allow to make the development evolutive. We could start with a mask class for the most general region, and then implement gradually other shapes.

> This part is completely unclear to me, please comment:
>
> - We need both "image regions" and "sky regions" in gammalib, right?

Why do we need sky regions? The regions are selected in data space (or image space), so I would guess we only would need image regions.

> - Do we need ImageRing and SkyRing classes with the image version based on cartesian coordinates and the sky version on spherical coordinates, or can both cases be combined in one class?

Again, what for do we need sky regions?

> - How should region classes (especially the mask) couple with the position and image classes in gammalib?

The GInstDir class (and its GCTAInstDir implementation) implements a direction in the data space. For the moment I'm using spherical coordinates everywhere, and they don't seem to be a major speed problem. Do we really need cartesian coordinates for the analysis?

Maybe here a note would help that describes the various coordinate systems we want to deal with. This would help to better understand what exactly should be done.

> - Do we need region reprojection (resampling in the case of the mask region) onto other images?

Maybe to in a first place, but probably at a later stage.

> To get started I would propose to only implement the geometrical regions commonly needed for TeV analysis:

- CircleSegment (part of a circle between an inner / outer radius and min / max position angle)
- Circle
- Ring
- Box (nice test case for rotations, because not radially symmetric around center)

as well as regions to specify irregular and complex shapes like e.g. exclusion regions:

- Polygon
- Mask (i.e. image with 0 and 1 entries)
- Compound (a list of regions)

Some, but not much code could be saved by using inheritance:

Region
-- CircleSegment -- Ring -- Circle
-- Polygon -- Box
-- Mask
-- Compound

It is unclear to me at the moment if the gains of using inheritance for Ring, Circle and Box (is-a relationship, avoid some code duplication) outweigh the drawbacks (most methods are re-implemented for the special cases anyways for speed, a Circle should not have a Get/SetOuterRadius method). As described above, in Kapteyn every region is a polygon.

I think the first step should be to design the interface for the Region class. Can we have a sufficiently generic interface that allows handling of regions in an abstract way (once they are defined, for example by an XML or a FITS file - maybe an XML file is better because directly editable).

Then the question is "a region of what"? Region on the sky, region in a data space? Is there any other use of the region classes in GammaLib, or are they specific to IACT analysis? One could eventually use them for aperture photometry of Fermi/LAT data, although this is rarely used (and often simple circles are used here ...).

Maybe we should define a region in GInstDir coordinates.

Members

Regions should have the following members:

-- Position (either cartesian X, Y or sky lambda, beta)
-- Some floats for CircleSegment, Ring, Circle, Box
-- List of (x, y) floats for Polygon
-- List of regions for Compound
-- Binary image for Mask

I think there should be a GRegion class (that defines a specific region) and a GRegions container. The GRegion class probably won't have any members, but just define the interface. The derived classes implementing the shapes then contain the region definitions.

If we really would need sky regions, we could have GInstRegion and GInstRegions and GSkyRegion and GSkyRegions classes.

Methods

Regions should implement the following methods:

-- Region::Region(region_string)
-- string Region::ToString()
-- Polygon Region.ToPolygon()

What is the region_string?

-- Region.GetCenter()

Does every region has a meaningful centre? Just as a note: the GammaLib convention would here be

Region.center()

i.e. no Get or Set prefixes.

-- Region.GetArea()
-- Region.GetEquivalentCircleRadius()
-- Region.GetBoundingBox()

Bounding Box or Bounding cercle?

-- Region.Contains(point)
-- Region.Contains(other_region)

Here the GammaLib convention would be Region.isin() (see e.g. in GEbounds).

-- Region.Overlaps(other_region)

-- Region.Rotate(point, system)

Images should implement the following methods:

What do you mean by images here? Sky maps (realized by the GSkymap class)? Does this imply that we may tie the region definition to the sky map definition? The Fermi/LAT and CTA classes use in fact the GSkymap class to store the data cubes. The GSkymap class supports WCS. This class could be certainly extended to support regions, implementing the methods below (with names following the GammaLib standards).

-- Image.GetSum(region) -- sum up pixel values in the region
-- Image.GetIntegral(region) -- like sum, but multiplied by bin area
-- Image.GetMask(region) -- pixels inside the region are 1, outside 0
-- Image.GetNPixelsInside(region) -- Image.GetMask(region).Sum()

I guess cubes (or any array with two spatial dimensions) should provide similar methods.

#6 - 10/11/2012 01:42 PM - Deil Christoph

Jürgen Knödlseder wrote:

> Why do we need sky regions? The regions are selected in data space (or image space), so I would guess we only would need image regions.

I think in HAP the ReflectedBgMaker runs without ever having an image object.
Probably it could just as well run in image space with an image e.g. representing a tangential projection with center at the pointing position.

The RingBgMaker has two modes, a fast one where a ring in image coordinates is used and a slow one where a ring on the sky is used, i.e. each pixel center coordinate is tested if it is contained in a ring on the sky.
We probably have to think a bit how to make a correct ring-correlated map at high latitudes.

Karl should definitely comment here, but as far as I can see, yes, we don't really need "sky regions" and not having them in addition to "image regions" would make regions much simpler.

> I think there should be a GRegion class (that defines a specific region) and a GRegions container. The GRegion class probably won't have any members, but just define the interface. The derived classes implementing the shapes then contain the region definitions.

The relationship Compound is a Region has the advantage that the user can e.g. have an off region like this (see http://cxc.harvard.edu/ciao/threads/regions/):

```
unix% cat exclude_one.reg
circle(4086,4084,20)
-circle(4102,4128,100)
circle(4074,4120,200)
```

Intuitively it makes sense to me that Regions is a list of Region objects, and not a subclass of Region. But then code working with e.g. On and Off regions always has to handle the two cases, which I think might be annoying.

> What do you mean by images here? Sky maps (realized by the GSkymap class)? Does this imply that we may tie the region definition to the sky map definition? The Fermi/LAT and CTA classes use in fact the GSkymap class to store the data cubes. The GSkymap class supports WCS. This class could be certainly extended to support regions, implementing the methods below (with names following the GammaLib standards).

Yes, I think sky maps should have methods that take regions as arguments. And probably the mask region would be implemented as a sky map, so regions and sky maps depend on each other.
This is the case in the HESS software. Do you think this would be a good design for gammalib?

I have to think some more about use cases for regions and where GInstDir and where GSkyDir should be used in the regions API.

#7 - 10/11/2012 01:53 PM - Deil Christoph

- File RegionDoxygen.pdf added

- File Region.hh added

- File Polygon.hh added

- File RegionTools.hh added

- File Region.C added

- File Polygon.C added

- File RegionTools.C added

I attached the HESS Tools::Region code for reference. The equivalent Stash::Coordinate in gammalib would be GSkyDir and GInstDir, the equivalent of Plotters::SkyHist would be GSkymap, I think. I still have to get familiar with gammalib and also review the HESS region code.

#8 - 10/11/2012 03:26 PM - Deil Christoph

- File Mathieu_AnalysisRegion.pdf added

#9 - 10/11/2012 03:46 PM - Knödlseder Jürgen

Christoph Deil wrote:

> The relationship Compound is a Region has the advantage that the user can e.g. have an off region like this (see http://cxc.harvard.edu/ciao/threads/regions/):
> [...]
>
> Intuitively it makes sense to me that Regions is a list of Region objects, and not a subclass of Region. But then code working with e.g. On and Off regions always has to handle the two cases, which I think might be annoying.

I guess we would then simply split in inclusion and exclusion regions.

#10 - 10/11/2012 03:47 PM - Knödlseder Jürgen

Christoph Deil wrote:

> Yes, I think sky maps should have methods that take regions as arguments. And probably the mask region would be implemented as a sky map, so regions and sky maps depend on each other.
> This is the case in the HESS software. Do you think this would be a good design for gammalib?

Right, I would have implement a mask region as a sky map.

#11 - 10/11/2012 10:36 PM - Knödlseder Jürgen

Thinking a little more about the region definition, I would argue that it should be okay to define a region in sky coordinates, as we will use the class to define regions in the sky. The fact that we may use regions not in physical but in data space is here not really relevant. What is relevant is that a point in a region is represented as a sky direction GSkyDir. Note in this respect that an instrument direction for an imaging telescope (implemented by GInstDir) is in general also only a copy of a GSkyDir object.

If we agree on this, it makes sense to add the region class to the sky module of GammaLib, as the region class will be used to select regions in a sky map.

Then, a region may be an inclusion or an exclusion region. One option would be to manage this by a member attribute (which specifies whether a region should be included or excluded). An alternative would be to have a collection of inclusion and exclusion regions. To illustrate the latter case, one could have a method

```
doSomething(const GRegions& inclusion, const GRegions& exclusion);
```

that takes both sets of regions as argument. But maybe this is too complex and always would require to carry to sets of regions. Having a flag that determines how a region should be handled may be more appropriate. Note that the flag could be used internally by the GRegion class so that

```
bool GRegion::isin(const GSkyDir& dir);
```

returns true if the sky direction is within the region for an inclusion region, and false for an exclusion region. The GRegions class would also have a

```
bool GRegions::isin(const GSkyDir& dir);
```

method, that would first scan all exclusion regions and return false if the direction is within one of those, and then scan all the inclusion regions and return true if the direction is within one of those.

Thinking a little more about it, it may even be better to handle this only at the GRegions level. GRegions would be a container that contains a list of GRegion objects and a list of flags, specifying whether a GRegion is included or excluded. This would mean that GRegion would not know if it is a inclusion or exclusion region. I think this is more logical and universal.

It would be at the point when a GRegion is appended to a GRegions container that the decision about inclusion or exclusion region is taken.

#12 - 10/15/2012 11:11 AM - Deil Christoph

Jürgen Knödlseder wrote:

> Thinking a little more about the region definition, I would argue that it should be okay to define a region in sky coordinates, as we will use the class to define regions in the sky. The fact that we may use regions not in physical but in data space is here not really relevant. What is relevant is that a point in a region is represented as a sky direction GSkyDir. Note in this respect that an instrument direction for an imaging telescope (implemented by GInstDir) is in general also only a copy of a GSkyDir object.
>
> If we agree on this, it makes sense to add the region class to the sky module of GammaLib, as the region class will be used to select regions in a sky map.

I think it is clear that GRegion will be closely related to GSkyMap, so I also think it should be in the sky module.
Just to clarify: GInstDir is not a GSkyDir subclass, so if we use GSkyDir in the GRegion API we can't work with GInstDir objects, right?
Do we need GRegion to work with GInstDir?

> Then, a region may be an inclusion or an exclusion region. One option would be to manage this by a member attribute (which specifies whether a region should be included or excluded). An alternative would be to have a collection of inclusion and exclusion regions. To illustrate the latter case, one could have a method
> [...]
> that takes both sets of regions as argument. But maybe this is too complex and always would require to carry to sets of regions. Having a flag that determines how a region should be handled may be more appropriate. Note that the flag could be used internally by the GRegion class so that
> [...]
> returns true if the sky direction is within the region for an inclusion region, and false for an exclusion region. The GRegions class would also have a
> [...]
> method, that would first scan all exclusion regions and return false if the direction is within one of those, and then scan all the inclusion regions and return true if the direction is within one of those.
>
> Thinking a little more about it, it may even be better to handle this only at the GRegions level. GRegions would be a container that contains a list of GRegion objects and a list of flags, specifying whether a GRegion is included or excluded. This would mean that GRegion would not know if it is a inclusion or exclusion region. I think this is more logical and universal.

I think the "region purpose" like on, off, exclusion should not be part of the region class, but handled by classes working with regions.

> It would be at the point when a GRegion is appended to a GRegions container that the decision about inclusion or exclusion region is taken.

In the HESS software we have a CompoundRegion, which is used e.g. for exclusion and off regions. It always represents the union of the regions in the compound, i.e. there's no way to e.g. define a box and cut out a circle by making a compound region where the box has is "include" and the circle is "exclude".

I'm not sure if is is necessary / useful to handle inclusion / exclusion at the GRegions level or if it should be handled by higher-level classes and tools.
I'm also not sure how useful GRegions will be because it is not a subclass of GRegion (see my comments on compound regions above).

I think these are the most difficult design decisions we should discuss this afternoon:

- GCompoundRegion versus GRegions and where exclusion / inclusion is handled.
- "sky regions", i.e. regions in sky coordinates versus "image regions", i.e. regions in image coordinates.

#13 - 10/15/2012 11:16 AM - Deil Christoph

- File BgStats.hh added

- File ReflectedBgMaker.hh added

- File BgStats.C added

- File ReflectedBgMaker.C added

Adding HESS BgStats class and ReflectedBgMaker to illustrate how on / off / exclusion regions are handled in the HESS software.

#14 - 10/15/2012 11:33 AM - Deil Christoph

Jürgen Knödlseder wrote:

> Christoph Deil wrote:
>
> > -- Region.Contains(point)
> > -- Region.Contains(other_region)
>
> Here the GammaLib convention would be Region.isin() (see e.g. in GEbounds).

I think this is a misnomer. If you want to use isin, I think

direction.isin(region)

would be semantically correct, whereas

region.isin(direction)

is semantically incorrect.

But of course GSkyDir is a more fundamental class than GRegion and thus should not depend on it.
So I'm not proposing to implement GSkyDir.isin(GRegion), but GRegion.contains(GSkyDir) instead.

#15 - 10/15/2012 12:01 PM - Deil Christoph

How should GRoi relate to GRegion?

As far as I can see the GRoi base at the moment is purely class boilerplate code (i.e. doesn't define an ROI- or region-specific API) and the GCTARoi and GLATRoi classes define a circle (center and radius members), but no methods like area(), contains(), ...

We should avoid code duplication.

- Do we even need extra ROI classes when we have regions?
- Do ROI classes have to be instrument-specific? (as far as I can see at the moment GCTARoi and GLATRoi are the same?)

For the HESS survey I do use FITS masks to define the pixels (basically large rectangles with sometimes bright sources I model separately cut out) I want to include in my binned morphology fits for crowded regions containing several sources. I call this ROI, but I don't know if it would fall under the ROI concept in gammalib / ctools.

#16 - 10/15/2012 12:30 PM - Knödlseder Jürgen

Christoph Deil wrote:

> I think it is clear that GRegion will be closely related to GSkyMap, so I also think it should be in the sky module.
> Just to clarify: GInstDir is not a GSkyDir subclass, so if we use GSkyDir in the GRegion API we can't work with GInstDir objects, right?
> Do we need GRegion to work with GInstDir?

For imaging instruments, GInstDir is basically a copy of GSkyDir, but for non-imaging instruments it may be substantially different (for a coded mask detector it would be just a detector pixel coordinate, for a Compton telescope it would be a 3D quantity).

I think we don't need this for GInstDir, let's just do it in physical space.

By the way: GSkyDir can transparently handle RA,DEC and GLON,GLAT. You may use whichever you want in the region definition (or even use mixed definitions; you can always read back in a given coordinate system, GSkyDir will automatically make the transformation, and also cache values, so that a coordinate transformation is done at most once).

> I think the "region purpose" like on, off, exclusion should not be part of the region class, but handled by classes working with regions.

Agree.

> > It would be at the point when a GRegion is appended to a GRegions container that the decision about inclusion or exclusion region is taken.

> In the HESS software we have a CompoundRegion, which is used e.g. for exclusion and off regions. It always represents the union of the regions in the compound, i.e. there's no way to e.g. define a box and cut out a circle by making a compound region where the box has is "include" and the circle is "exclude".
>
> I'm not sure if is is necessary / useful to handle inclusion / exclusion at the GRegions level or if it should be handled by higher-level classes and tools.
> I'm also not sure how useful GRegions will be because it is not a subclass of GRegion (see my comments on compound regions above).
>
> I think these are the most difficult design decisions we should discuss this afternoon:
>
> - GCompoundRegion versus GRegions and where exclusion / inclusion is handled.
> - "sky regions", i.e. regions in sky coordinates versus "image regions", i.e. regions in image coordinates.

I used combinations of inclusion and exclusion regions for SPI on INTEGRAL (for pointing selections). This was very useful. I think having both types of regions gives you a lot of freedom to specify the region you're interested in.

If you don't use inclusion and exclusion, how does CompoundRegion handle a ring background from which you exclude some areas due to the presence of sources?

#17 - 10/15/2012 12:33 PM - Knödlseder Jürgen

Christoph Deil wrote:

> Jürgen Knödlseder wrote:
>
> > Christoph Deil wrote:
> >
> > > -- Region.Contains(point)
> > > -- Region.Contains(other_region)
> >
> > Here the GammaLib convention would be Region.isin() (see e.g. in GEbounds).
>
> I think this is a misnomer. If you want to use isin, I think
> [...]
> would be semantically correct, whereas
> [...]
> is semantically incorrect.
>
> But of course GSkyDir is a more fundamental class than GRegion and thus should not depend on it.
> So I'm not proposing to implement GSkyDir.isin(GRegion), but GRegion.contains(GSkyDir) instead.

I don't understand. You want to check if a sky direction is in a region (e.g. is an event or a pixel in a region). In this case you would use Region.isin().

But I see. You mean when you read from left to right, isin() makes no sense, but we should use contains(). Point taken. Needs some revision of the existing interface smile.png

#18 - 10/15/2012 12:39 PM - Knödlseder Jürgen

Christoph Deil wrote:

> How should GRoi relate to GRegion?
>
> As far as I can see the GRoi base at the moment is purely class boilerplate code (i.e. doesn't define an ROI- or region-specific API) and the GCTARoi and GLATRoi classes define a circle (center and radius members), but no methods like area(), contains(), ...
>
> We should avoid code duplication.
>
> - Do we even need extra ROI classes when we have regions?
> - Do ROI classes have to be instrument-specific? (as far as I can see at the moment GCTARoi and GLATRoi are the same?)
>
> For the HESS survey I do use FITS masks to define the pixels (basically large rectangles with sometimes bright sources I model separately cut out) I want to include in my binned morphology fits for crowded regions containing several sources. I call this ROI, but I don't know if it would fall under the ROI concept in gammalib / ctools.

GRoi is definitely defined in the data space. For the moment it's only implemented for CTA, and here we have an imaging instrument, hence image space = data space.

The ROI is used for unbinned analysis, and it defines the region over which the model needs to be integrated. In this concept, we could use a simple circular region to define the ROI, i.e. once the region class is available, we can use it for GCTARoi. Having more complex forms will probably be an implementation nightmare, as the actual code uses the fact that the Roi is a circle. Hence when we use GRegion for this, we would need to check that it's a circle. Should also not be a problem. Or we use directly the GRegionCircle class for ROI implementation.

#19 - 10/15/2012 12:49 PM - Deil Christoph

Jürgen Knödlseder wrote:

> You mean when you read from left to right, isin() makes no sense, but we should use contains(). Point taken. Needs some revision of the existing interface smile.png

Yes. I looked at Java lists to come up with the name contains .
Generally Java is probably a good guide because it is very object-oriented and clean. C++ stdlib containers can't be our guide for names because they have the algorithms separate from the containers (std::find) and Python lists have a language construct (element in list).

#20 - 10/15/2012 01:02 PM - Deil Christoph

Jürgen Knödlseder wrote:

> If you don't use inclusion and exclusion, how does CompoundRegion handle a ring background from which you exclude some areas due to the presence of sources?

For spectral analyses these regions in BgStats are used:

```
Tools::Compound fOnRegion;            //< On region, possibly compound
Tools::Compound fOffRegion;        //< Off region, possibly compound
Tools::Compound fOnExclusionRegion;//< Exclusion region, possibly compound
Tools::Compound fOffExclusionRegion;//< Exclusion region, possibly compound
```

We don't use the RingBgMaker to make spectra. For making maps the RingBgMaker multiplies the CountMap and ExposureMap with the ExclusionMap (all SkyHists, not regions), and then creates a temporary Ring region object for each pixel and sums the content of the (Count x Exclusion) and (Exposure x Exclusion) maps.

#21 - 10/15/2012 01:08 PM - Knödlseder Jürgen

Christoph Deil wrote:

> Jürgen Knödlseder wrote:
>
> > If you don't use inclusion and exclusion, how does CompoundRegion handle a ring background from which you exclude some areas due to the presence of sources?
>
> For spectral analyses these regions in BgStats are used:
> [...]
>
> We don't use the RingBgMaker to make spectra. For making maps the RingBgMaker multiplies the CountMap and ExposureMap with the ExclusionMap (all SkyHists, not regions), and then creates a temporary Ring region object for each pixel and sums the content of the (Count x Exclusion) and (Exposure x Exclusion) maps.

Okay, I understand. Won't it be more logical to have both inclusion and exclusion regions in a single GRegions object? These regions are just elements that define the compound region at the end. But it's not a simple container, so maybe another structure would be more appropriate:

- GRegion abstract base class for a region
- GRegionCompound or GCompoundRegion contains inclusion and exclusion elements, derives from GRegion (I still have no clear preference for whether putting the attribute before or after the base class name; most classes put the attribute behind, so that when listing things, classes are together)
- GRegionCircle or GCircleRegion to implement a circle, derives from GRegion

Objects taking regions just take then the GRegion reference ...

#22 - 05/02/2013 01:40 PM - Martin Pierrick

- File GSkyRegions.hpp added

- File GSkyRegion.hpp added

- File GSkyRegionCircle.hpp added

I have made draft header files for the sky region container, the sky region interface, and a circular sky region. At the moment it is the bare minimum and no high-level functionalities are implemented (such as integration of any quantity within the region...). These will soon be uploaded to the classical-analysis branch of the git repository.

#23 - 05/02/2013 01:59 PM - Knödlseder Jürgen

Great ! Some questions:

Pierrick Martin wrote:

> I have made draft header files for the sky region container, the sky region interface, and a circular sky region. At the moment it is the bare minimum and no high-level functionalities are implemented (such as integration of any quantity within the region...). These will soon be uploaded to the classical-analysis branch of the git repository.

GSkyRegion

- What is the purpose of the name attribute of a region?
- What is the purpose of the id attribute of a region?
- I guess the access operators should return a reference to a value instead of the value (is quicker)
- Same for at, and you also want a non-const at method.

#24 - 06/21/2013 03:43 PM - Martin Pierrick

Sorry, I had not seen the comment earlier...
I replaced the name attribute by a type attribute which will be Circle,Ring,Triangle...etc depending on the derived class.
The need for the id attribute is not clear. Its initial role was to connect a region to an observation run (because in ON-OFF analysis, the OFF regions can differ for each run), but then we said that GSkyRegion should be a class independently of its use in ON-OFF analysis.
Access operators now return pointers. The at() method was removed.

#25 - 06/23/2013 02:14 PM - Martin Pierrick

- Tracker changed from Feature to Action

- Subject changed from Add Regions / Shape classes to Preliminary discussion

- Status changed from New to Resolved

- Estimated time set to 0.00

- Position deleted (46)

#26 - 06/23/2013 02:17 PM - Martin Pierrick

- Parent task set to #903

#27 - 06/23/2013 02:42 PM - Martin Pierrick

- % Done changed from 0 to 100

#28 - 06/23/2013 10:23 PM - Knödlseder Jürgen

- Status changed from Resolved to Closed

Files

RegionDoxygen.pdf	1.04 MB	10/11/2012	Deil Christoph
Region.hh	16 KB	10/11/2012	Deil Christoph
Polygon.hh	1.59 KB	10/11/2012	Deil Christoph
RegionTools.hh	441 Bytes	10/11/2012	Deil Christoph
Region.C	41.9 KB	10/11/2012	Deil Christoph
Polygon.C	8.83 KB	10/11/2012	Deil Christoph
RegionTools.C	12.8 KB	10/11/2012	Deil Christoph
Mathieu_AnalysisRegion.pdf	533 KB	10/11/2012	Deil Christoph
BgStats.hh	6.62 KB	10/15/2012	Deil Christoph
ReflectedBgMaker.hh	2.16 KB	10/15/2012	Deil Christoph
BgStats.C	24 KB	10/15/2012	Deil Christoph
ReflectedBgMaker.C	17.8 KB	10/15/2012	Deil Christoph
GSkyRegions.hpp	6.39 KB	05/02/2013	Martin Pierrick
GSkyRegion.hpp	6.17 KB	05/02/2013	Martin Pierrick
GSkyRegionCircle.hpp	5.65 KB	05/02/2013	Martin Pierrick