GammaLib - Action #576

Feature # 104 (Closed): Refactor CTA response class

Introduce GCTABackground class

10/16/2012 06:16 PM - Knödlseder Jürgen

Status:	Closed	**Start date:**	07/20/2012
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.0.0		

Description

This class should deal with the background extension that may be available in the response files (see action #537 and related discussions). As far as I understand, it should deal with BACKGROUND ACCEPTANCE, which is the probability of measuring a background event as function of energy and position in the FOV.

The BACKGROUND ACCEPTANCE is given versus reconstructed (or observed) energy.

There is also another extension called BACKGROUND RATE. I'm not sure what should be done about this.

History

#1 - 12/01/2012 02:48 AM - Knödlseder Jürgen

- Description updated

#2 - 12/01/2012 03:03 AM - Knödlseder Jürgen

Before starting the implementation, we should think about the name. Is GCTABackground really the right name, or should we have one of:

- GCTABackgroundAcceptance
- GCTABgdAcceptance
- GCTAAcceptance

#3 - 05/15/2013 09:27 AM - Knödlseder Jürgen

- Target version changed from HESS sprint #1 to 00-08-00

#4 - 10/03/2013 02:31 PM - Knödlseder Jürgen

I would propose to create a generic class that is inspired from GCTAModelRadialAcceptance, with the main difference being that the radial component GCTAModelRadial is replaced by a generic spatial model GModelSpatial. We could then use the existing classes GModelSpatialDiffuseMap and GModelSpatialDiffuseCube to handle either maps or map cubes. Note that this would even allow to handle directly geometric models (such as Gaussians, etc), which may make GCTAModelRadialAcceptance in the long run obsolete.

If we'd like to go this way we would need to investigate how the npred computation is done. See GCTAModelRadialAcceptance::npred for the actual computation in the GCTAModelRadialAcceptance class, which then would need to be generalized.

We could name this class indeed GCTABackground, but would this be the ultimate and only background class for CTA? I also like the idea to keep Model in the name, so that model classes are easy to identify. What about GCTAModelBackground in this case?

#5 - 10/07/2013 05:29 PM - Mayer Michael

I fully agree to this generic class. It allows way more flexibility in analysing arbitrary background shapes, which can become completely non-Gaussian in some energies and also zenith ranges.
It could be named GCTAModelBackground or GCTABackground. I would also like to propose GCTAFoVBackground, since this clarifies that the content of this class is strongly related to the instrument and not to something physical like e.g. diffuse emission.

#6 - 10/07/2013 05:47 PM - Knödlseder Jürgen

Michael Mayer wrote:

> I fully agree to this generic class. It allows way more flexibility in analysing arbitrary background shapes, which can become completely non-Gaussian in some energies and also zenith ranges.
> It could be named GCTAModelBackground or GCTABackground. I would also like to propose GCTAFoVBackground, since this clarifies that the content of this class is strongly related to the instrument and not to something physical like e.g. diffuse emission.

But FoV is also not very explicit in terms of content.

GCTAModelInstrumentBackground (very long sad.png
GCTAInstrumentBackground (no model in name, maybe ok)
GCTABackground (short and smart and analogue to GCTAResponse; as this says CTA background it's maybe implicit that the background is instrument related wink.png

#7 - 10/07/2013 11:12 PM - Mayer Michael

You are right, GCTABackground already implies that it is instrument-related. Let's continue with this name.

#8 - 10/08/2013 02:51 PM - Mayer Michael

GCTAModelBackgrond would also be fine. Concerning the npred method: As far as I understand it, we would need to define a new class GCTAModelBackground::roi_kern used to integrate over the spatial model. The eval method of this function should probably take a GSkyDir-object as argument. Do we need to define a new class GFunction2D to perform this integration? Or is a 2D-integration already implement somewhere else? In my understanding this integration is crucial to achieve this generalised background approach. What do you think?

#9 - 10/08/2013 03:42 PM - Knödlseder Jürgen

So far I do 2D integrations by nested 1D integrations. The experience in Fermi shows that numerical integration is a tricky thing, and the order of the integrals can be important (as at some level one always runs in numerical noise problems). Hence I would recommend to break the integral down in an azimuthal integral and an offset angle integral. I would put the azimuthal integral as the inner integral, as the azimuthal dependence is likely often pretty smooth.

#10 - 10/08/2013 03:55 PM - Mayer Michael

ok that sounds good to me. So we would basically need two classes: GCTAModelBackground::roi_kern_offset and GCTAModelBackground::roi_kern_azimuthal, where the latter one contains the first as member, right? Is there a class where you already use this, so that we can start from that?

#11 - 10/08/2013 04:12 PM - Knödlseder Jürgen

Yes, have a look at the files GCTAResponse.cpp and GCTAResponse_helpers.cpp.

GCTAResponse::npred_radial uses the integrand cta_npred_radial_kern_rho (defined in GCTAResponse_helpers.cpp) for the offset angle integration. cta_npred_radial_kern_rho uses cta_npred_radial_kern_omega as kernel for azimuth angle computation. The npred function call for a point source is in cta_npred_radial_kern_omega. There a similar examples for other model types ...

#12 - 10/08/2013 05:25 PM - Mayer Michael

Ok great. Thanks for the references. I would set up the necessary header and source-files and check them in. From them we could create sub features which have to be implemented, like e.g. the npred-method, etc...

#13 - 10/09/2013 10:04 AM - Mayer Michael

While setting up the files, I encountered two issues, we probably have to think about:

1. the GCTModelBackground::npred-method maybe should be subdivided in npred_radial, npred_elliptical and npred_diffuse. Each function integrating the respective spatial model properly. The other possibility is to add an npred-Method to the GModelSpatial-class which gets overwritten by the derived classes like GModelSpatialRadial, GModelSpatialElliptical and GModelSpatialDiffuse.
2. The GModelSpatial classes have to be extended by the function omega(), returning the solid angle of the spatial model. This value is used in the mc-method for normalisation
3. Do we always want to integrate from the instrument pointing direction, or get the integration minimum from the center of gravity fro the model itself?

#14 - 10/09/2013 12:05 PM - Mayer Michael

concerning point 1, I will start to implement that using a general integration in *theta* and *phi*. If we realise that this is unsuited for some models, we can still change that.

#15 - 10/09/2013 12:37 PM - Knödlseder Jürgen

Michael Mayer wrote:

> While setting up the files, I encountered two issues, we probably have to think about:
>
> 1. the GCTModelBackground::npred-method maybe should be subdivided in npred_radial, npred_elliptical and npred_diffuse. Each function integrating the respective spatial model properly. The other possibility is to add an npred-Method to the GModelSpatial-class which gets overwritten by the derived classes like GModelSpatialRadial, GModelSpatialElliptical and GModelSpatialDiffuse.
> 2. The GModelSpatial classes have to be extended by the function omega(), returning the solid angle of the spatial model. This value is used in the mc-method for normalisation
> 3. Do we always want to integrate from the instrument pointing direction, or get the integration minimum from the center of gravity fro the model itself?

Good questions. I need some time to think about these points.

#16 - 10/10/2013 05:35 PM - Mayer Michael

- File GCTAModelBackground.cpp added

- File GCTAModelBackground.hpp added

- File GCTAModelBackground.i added

I set up the files GCTAModelBackground.hpp and GCTAModelBackground.cpp (see attachment). The npred-method is implemented generally but should be reviewed extensively. For this, I somewhat used the general integration from GCTAResponse::npred_elliptical and GCTAResponse::npred_diffuse of course leaving out the irf folding. In the mc-method, I preliminary fixed the solid angle of the model to 1.0. There, we should either implement the function GModelSpatial::omega or think about something else (see my previous point 2).
For now the integration is done starting from the pointing direction.
Note, that these files are only a first draft - documentation might be wrong and bugs are mandatory smile.png
It should compile however...

#17 - 10/10/2013 05:46 PM - Knödlseder Jürgen

Great!

If you work under a specific branch I recommend that you just push this branch into the Git repo so that everybody can see it.

In case you have not yet done so, we probably should add also a unit test case so that the new class can be tested immediately.

#18 - 10/11/2013 07:29 AM - Mayer Michael

- File cta_background_model_test.xml added

- Status changed from New to Feedback

- % Done changed from 0 to 50

- Remaining (hours) set to 0.0

I pushed a new branch named *576-introduce-GCTAModelBackground* into the central repo. A unit test would be very good. Do we need to create a new subfeature for that?
I attached an exemplary *.xml* file how such a model could be defined.

#19 - 10/11/2013 09:45 AM - Knödlseder Jürgen

- Estimated time set to 0.00

Michael Mayer wrote:

> I pushed a new branch named *576-introduce-GCTAModelBackground* into the central repo. A unit test would be very good. Do we need to create a new subfeature for that?

Thanks. Typically I create the unit test together with the functionality (key word: test driven development), hence I don't create a separate feature for this. The basic idea behind this is that a new feature should be tested anyways before merging into the trunk, hence a test goes together with a function.

> I attached an exemplary *.xml* file how such a model could be defined.

Looks good. Concerning the type, I would call this "CTABackground" to make clear that this is a CTA specific background model. There is some redundancy here with the instrument="CTA" attribute, but this would allow to create background models also for other instruments without type

conflicts (the types must be unique among all models).

#20 - 10/11/2013 09:59 AM - Mayer Michael

Yes, I should have done the tests right away. I'll try to create the unit test as soon as possible. Unfortunately (shame on me), I've never done this before.
I agree to include *CTA* in the name

#21 - 10/11/2013 10:26 AM - Knödlseder Jürgen

Michael Mayer wrote:

> Yes, I should have done the tests right away. I'll try to create the unit test as soon as possible. Unfortunately (shame on me), I've never done this before.
> I agree to include *CTA* in the name

Just look in the existing unit test files test_CTA.cpp/test_CTA.hpp and add some test cases. Maybe you want to inspire your tests from TestGCTAResponse::test_response_irf_diffuse and/or TestGCTAResponse::test_response_npred_diffuse.

#22 - 10/11/2013 10:29 AM - Mayer Michael

Thanks, I will do that.

#23 - 10/18/2013 09:29 AM - Mayer Michael

- % Done changed from 50 to 70

I defined a test-case for the npred method of this class and pushed it into the feature branch: Using a Gaussian spatial model and setting the roi of the GCTAObservation to 1sigma allows to check wether the integration works. The integral of a 2D Gaussian within one standard deviation gives analytically {{latex(\left(1-\frac{1}{\sqrt{e}}\right))}}.
For now, I took the default integration precision of 1e-6 which looks good. Probably, we have to investigate the trade-off between computation speed and accuracy.
Do we need more test cases for this class? We also still have to think about the mc-method, wether we want to add functions returning the solid angle to GModelSpatial.
P.S. The test_COM fails in binned_observations, should I report a bug or is that already known?

#24 - 10/31/2013 02:38 PM - Mayer Michael

I extensively tested this class using a GModelSpatialDiffuseCube as spatial model and fitting it to off data (from HESS). I found out that using fits-cubes, the npred-method, which is called during the fit takes way too long to calculate (~30s - 1min per call). The resulting fit values look good but unless there is no smarter way to integrate over the spatial part of the model, we may consider using a sub-class for this specific case. My suggestion would be a GCTAModelFITSBackground or GCTAModelBackgroundCube, etc. This sub-class could specifically store the pre-calculated integration values in each energy bin and return it as needed. What do you think?

#25 - 10/31/2013 10:41 PM - Knödlseder Jürgen

Mayer Michael wrote:

> I extensively tested this class using a GModelSpatialDiffuseCube as spatial model and fitting it to off data (from HESS). I found out that using fits-cubes, the npred-method, which is called during the fit takes way too long to calculate (~30s - 1min per call). The resulting fit values look good but unless there is no smarter way to integrate over the spatial part of the model, we may consider using a sub-class for this specific case. My suggestion would be a GCTAModelFITSBackground or GCTAModelBackgroundCube, etc. This sub-class could specifically store the pre-calculated integration values in each energy bin and return it as needed. What do you think?

Is it correct that you used GModelSpatialDiffuseCube as spatial model for GCTAModelBackground?

One thing I recognized is that GObservation::npred_temp will perform a temporal integration that is not really necessary. GObservation::npred_temp just checks for the special case of a sky model and replaces the integration by a multiplication in case that the temporal model is a constant. Now we would need a more general method to handle this case. One solution would be to add a isconstant() method to GModel so that GObservation::npred_temp can make a general check. For GModelSky and GCTAModelBackground, this method would simply test whether the temporal component is a constant or not.

For the rest I agree that pre-computation would be best. The only question is where to store the precomputed values. The Npred values depend on the FITS cube content and the position of the ROI. The latter is associated to the observation, the former to the model. As a model could in principle apply to several observations, the precomputed values cannot be stored simply with the model.

I implemented in fact a caching logic for IRF and Npred value for diffuse models in GCTAResponse. See for example GCTAResponse::npred_diffuse and the section that starts with #if defined(G_USE_NPRED_CACHE). Here, the integration is skipped when a Npred value is found in the cache. The cache consists of Npred values as function of model name, photon energy and time (see GCTAResponse.hpp). This gave a tremendous speed-up for the diffuse models.

Note that the conditional computation allows for easy cross-checking.

I guess a pretty equivalent approach would be to add an Npred cache to GCTAModelBackground::npred, where the cache values are a function of the observation id, the energy and the time. For the id you may even combine instrument type and id to make sure that the result is unique (although there is so far no check on uniqueness in GammaLib, but this should be added). For the response cache, I use
std::string id = source.name() + "::" + obs.id();
to build a unique name.

Another thing is that you do not set the integration precision, so it's 1e-6 by default. In GCTAResponse::npred_diffuse I set the precision explicitly to 1e-4 for the theta integration and 1e-4 for the phi integration (see cta_npred_diffuse_kern_theta::eval). This has an important impact on the speed. You may check [[GCTAResponse]] for results with different precision values. You may do an equivalent study. In case you know the exact answer you may tune the values to a reasonable precision.

#26 - 10/31/2013 11:48 PM - Knödlseder Jürgen

Knödlseder Jürgen wrote:

> One thing I recognized is that GObservation::npred_temp will perform a temporal integration that is not really necessary. GObservation::npred_temp just checks for the special case of a sky model and replaces the integration by a multiplication in case that the temporal model is a constant. Now we would need a more general method to handle this case. One solution would be to add a isconstant() method to GModel so that GObservation::npred_temp can make a general check. For GModelSky and GCTAModelBackground, this method would simply test whether the temporal component is a constant or not.

I just added an isconstant() method to GModel and the derived classes and use this method now in GObservation to check whether a temporal integration is needed. You need to implement this method now also for GCTAModelBackground. Code is in devel.

#27 - 11/06/2013 10:54 AM - Mayer Michael

> Is it correct that you used GModelSpatialDiffuseCube as spatial model for GCTAModelBackground?

Yes, I built a cube containing the expected background rate [#/s/TeV/sr] for a HESS run, similar to the Fermi Galactic diffuse model. If the model matches the data, I expect a normalisation of the power-law spectral component around 1 and a spectral index around 0 (analogous to Fermi). This works pretty well so far. However, I needed to comment out the line, where the spatial cube gets normalised in the GModelSpatialDiffuseCube::read-function. This still needs some thinking, how this can be done automatically.

> I implemented in fact a caching logic for IRF and Npred value for diffuse models in GCTAResponse. See for example GCTAResponse::npred_diffuse and the section that starts with #if defined(G_USE_NPRED_CACHE). Here, the integration is skipped when a Npred value is found in the cache. The cache consists of Npred values as function of model name, photon energy and time (see GCTAResponse.hpp). This gave a tremendous speed-up for the diffuse models.
>
> Note that the conditional computation allows for easy cross-checking.
>
> I guess a pretty equivalent approach would be to add an Npred cache to GCTAModelBackground::npred, where the cache values are a function of the observation id, the energy and the time. For the id you may even combine instrument type and id to make sure that the result is unique (although there is so far no check on uniqueness in GammaLib, but this should be added). For the response cache, I use
> std::string id = source.name() + "::" + obs.id();
> to build a unique name.

I followed and implemented this approach and it really gave an incredible speed up. Fitting the cube model to ~2000 events took only a few seconds. Actually, in case of a FITS-cube, we could speed-up the computation even more since the spatial integration only has to be done once per ebin, right? Or does the GModelSpatialDiffuseCube, i.e. the GSkymap-cube also interpolate in energy?

> Another thing is that you do not set the integration precision, so it's 1e-6 by default. In GCTAResponse::npred_diffuse I set the precision explicitly to 1e-4 for the theta integration and 1e-4 for the phi integration (see cta_npred_diffuse_kern_theta::eval). This has an important impact on the speed. You may check [[GCTAResponse]] for results with different precision values. You may do an equivalent study. In case you know the exact answer you may tune the values to a reasonable precision.

In fact, I quickly tested the precision down to 1e-2, still giving reasonable numbers. I will perform a test and put it to the wiki, in order to find the best value.

I also implemented the isconstant() in the GCTAModelBackground-class.

#28 - 11/13/2013 11:22 PM - Deil Christoph

Chia-Chun and I would like to start testing this way of modeling the background, too.
Is there an ETA for this in the devel branch?
Is there something missing that we might be able to help implement?

#29 - 11/14/2013 05:40 AM - Knödlseder Jürgen

As soon as Michael gives green light I'll pull in the changes.

#30 - 11/14/2013 05:52 AM - Knödlseder Jürgen

Mayer Michael wrote:

> Yes, I built a cube containing the expected background rate [#/s/TeV/sr] for a HESS run, similar to the Fermi Galactic diffuse model. If the model matches the data, I expect a normalisation of the power-law spectral component around 1 and a spectral index around 0 (analogous to Fermi). This works pretty well so far. However, I needed to comment out the line, where the spatial cube gets normalised in the GModelSpatialDiffuseCube::read-function. This still needs some thinking, how this can be done automatically.

We could add a flag that enables/disables normalisation. This flag could be set as an attribute in the XML file.

We could even avoid the necessity to add this attribute to the XML file by adding some code to GCTAModelBackground::read that sets the corresponding attribute in the spatial component of the model (the XML tree is loaded in GCTAModelBackground::read, hence we should have full control over its content at this level).

> I followed and implemented this approach and it really gave an incredible speed up. Fitting the cube model to ~2000 events took only a few seconds. Actually, in case of a FITS-cube, we could speed-up the computation even more since the spatial integration only has to be done once per ebin, right? Or does the GModelSpatialDiffuseCube, i.e. the GSkymap-cube also interpolate in energy?

Practically it interpolates in energy, so as long as the few seconds are acceptable I would leave it as is.

> In fact, I quickly tested the precision down to 1e-2, still giving reasonable numbers. I will perform a test and put it to the wiki, in order to find the best value.

The question is of course how "reasonable" is defined. I basically wanted "better than 1%" for the response so that the numerical precision is small in comparison with anticipated systematic uncertainties in the IRF. For background this may be a little less relevant, on the other hand, you need it fast, not necessarily fastest, so I would choose the highest precision that you feel is still acceptable in computing time. We can always reconsider these values once more bells and whistles are added and the code becomes slower.

> I also implemented the isconstant() in the GCTAModelBackground-class.

Great!

#31 - 11/14/2013 10:08 AM - Mayer Michael

Knödlseder Jürgen wrote:

> We could add a flag that enables/disables normalisation. This flag could be set as an attribute in the XML file.
>
> We could even avoid the necessity to add this attribute to the XML file by adding some code to GCTAModelBackground::read that sets the corresponding attribute in the spatial component of the model (the XML tree is loaded in GCTAModelBackground::read, hence we should have full control over its content at this level).

This means we would need to add the attribute to the respective spatial model (starting from the base class). But in reality, I guess the GModelSpatialDiffuseCube would be the only model where we have this choice. The other spatial models are normalised by definition right? Therefore, I don't know wether we should only add a flag to the function GModelSpatialDiffuseCube::read, or approach this issue more general. What is your preference for this?

> Practically it interpolates in energy, so as long as the few seconds are acceptable I would leave it as is.

ok, great.

> The question is of course how "reasonable" is defined. I basically wanted "better than 1%" for the response so that the numerical precision is small in comparison with anticipated systematic uncertainties in the IRF. For background this may be a little less relevant, on the other hand, you need it fast, not necessarily fastest, so I would choose the highest precision that you feel is still acceptable in computing time. We can always reconsider these values once more bells and whistles are added and the code becomes slower.

ok, I'll roughly check for different precisions. I think 1e-3 should also be fast enough.

We still need to think how to handle the GCTAModelBackground::mc method. Currently, it will give wrong results, since the return value has to be scaled by the solid angle of the model. We would probably need a function returning this value in GModelSpatial-base class (implemented in the inherited classes).

#32 - 11/16/2013 01:56 AM - Knödlseder Jürgen

I thought about your comments.

I concluded that a diffuse map cube should not be normalised like a diffuse sky map, as the diffuse map cube contains both spatial and spectral information. The purpose for normalising a diffuse sky map was that the spectral fitting result can be directly used to compute the total photon or energy flux. For a diffuse map cube, however, the total flux is given by the model itself (modulo a scaling factor).

I thus modified the GModelSpatialDiffuseCube in that sense. I also changed the eval() and eval_gradients() method so that the actual intensity value is derived by interpolation in energy, not by selecting a particular map. This certainly slows down a little bit the computations, but avoids artificial jumps in the spectrum at the end.

Furthermore I implemented the Monte Carlo cache correctly. I had in fact to modify also GModelSky::mc() a bit as the diffuse map cube is in fact an exception with respect to all other models in that it is not fully factorised. To handle this, I introduced a spectral nodes function as member of GModelSpatialDiffuseCube and filled this node function by integration over the sky maps in the cube. This spectral nodes function is then recovered by GModelSky::mc() where it is multiplied with the spectral function of the model. This combined spectral model is then used for MC sampling (see lines 832ff). You would need to implement a similar thing in GCTAModelBackground.

Also note the GModelSpatialDiffuseCube::set_mc_cone() method that limits the MC pre-computations to a circular region. This speeds up simulations in cases where the map is much larger than the region of interest, and also assures that the spectrum is consistent with the selected region from the map.

I put all the new code together with a small test script test/test_diffuse_cube.py into the branch 576-adapt-diffuse-cube. You may merge this branch into your feature branch to see if you can combine it with the GCTAModelBackground class.

Please also note that GModelSpatialDiffuseCube assumes that the maps are given in units of ph/cm2/s/sr/MeV. Most of the factors (cm2, s, sr) are basically scaling factors (sr may vary slightly over the map, though), hence they are not really an issue. The per MeV unit needs however to be considered carefully, otherwise your spectra will get skewed.

#33 - 11/18/2013 06:06 PM - Mayer Michael

Thanks for the comprehensive explanations. I agree, the GModelSpatialDiffuseCube should not be normalised at all. The interpolation in energy is definitely the better way than to just return the values from the respective maps. I already tested it and analysed a few runs - the computation speed is not really affected by the new interpolation.
I implemented the GCTAModelBackground::mc-method in the same manner as in GModelSky. While testing, I realised that a cube where one map is fully zero creates nan in the GModelSpectralNodes-parameters. Therefore, I added a simple if-statement to GModelSpatialDiffuseCube::set_mc_cone() which allows to only append a node if the flux is >0

> I put all the new code together with a small test script test/test_diffuse_cube.py into the branch 576-adapt-diffuse-cube. You may merge this branch into your feature branch to see if you can combine it with the GCTAModelBackground class.

Great. I merged my branch with the new code. I also created a test script in the CTA module test_cta_bgcube.py. I directly adapted this script from yours. The script does not need any additional files, so the size of the repo stays the same.

> Please also note that GModelSpatialDiffuseCube assumes that the maps are given in units of ph/cm2/s/sr/MeV. Most of the factors (cm2, s, sr) are basically scaling factors (sr may vary slightly over the map, though), hence they are not really an issue. The per MeV unit needs however to be considered carefully, otherwise your spectra will get skewed.

For the analysis of cubes, it had units of per TeV, which also worked. But for the simulation, I agree, we need a fixed unit. Do you think it would make sense to read the values according to the unit specified in the "ENERGIES" extension? I mean someone who stores the energies in TeV would probably also like to store the values of the FITS cube in the same unit.

In addition, I adjusted the unit test test_GModel.cpp to be compatible with the new constructor of GModelSpatialDiffuseCube. By the way: why exactly did you change the GEnergies to std::vector<GEnergy> there?

#34 - 11/18/2013 10:00 PM - Knödlseder Jürgen

Mayer Michael wrote:

> Thanks for the comprehensive explanations. I agree, the GModelSpatialDiffuseCube should not be normalised at all. The interpolation in energy is definitely the better way than to just return the values from the respective maps. I already tested it and analysed a few runs - the computation speed is not really affected by the new interpolation.
> I implemented the GCTAModelBackground::mc-method in the same manner as in GModelSky. While testing, I realised that a cube where one map is fully zero creates nan in the GModelSpectralNodes-parameters. Therefore, I added a simple if-statement to GModelSpatialDiffuseCube::set_mc_cone() which allows to only append a node if the flux is >0

Indeed, GModelSpectralNodes computes the log of the intensities hence it does not like zero fluxes. There is nothing that catches this problem. I created bug #990 for this.

Adding the if statement as you did is certainly a work around. At the same time, having a diffuse cube with an empty map does not make really sense (model values also need to be positive), hence one could also think about removing these maps internally. We should at least watch if this case has no other side effects.

> > I put all the new code together with a small test script test/test_diffuse_cube.py into the branch 576-adapt-diffuse-cube. You may merge this branch into your feature branch to see if you can combine it with the GCTAModelBackground class.
>
> Great. I merged my branch with the new code. I also created a test script in the CTA module test_cta_bgcube.py. I directly adapted this script from yours. The script does not need any additional files, so the size of the repo stays the same.
>
> > Please also note that GModelSpatialDiffuseCube assumes that the maps are given in units of ph/cm2/s/sr/MeV. Most of the factors (cm2, s, sr) are basically scaling factors (sr may vary slightly over the map, though), hence they are not really an issue. The per MeV unit needs however to be considered carefully, otherwise your spectra will get skewed.
>
> For the analysis of cubes, it had units of per TeV, which also worked. But for the simulation, I agree, we need a fixed unit. Do you think it would make sense to read the values according to the unit specified in the "ENERGIES" extension? I mean someone who stores the energies in TeV would probably also like to store the values of the FITS cube in the same unit.

I was checking the code to see where the exact unit used is in fact important. In fact, for storing log10(E) I assume that energies are in MeV (as everywhere in GammaLib where I have to deal explicitly with energy units), but this is generally not an issue since the back-conversion to a GEnergy object is done correctly, hence the internal unit is generally not really relevant. At the end, I think there is no code in GModelSpatialDiffuseCube where I explicitly assume that the map is per MeV and not per TeV.

I think the only point which counts is in fact the code ewidth.MeV() in the test_diffuse_cube.py Python script. If you had a model per TeV, then you would need to replaced this by ewidth.TeV().

> In addition, I adjusted the unit test test_GModel.cpp to be compatible with the new constructor of GModelSpatialDiffuseCube. By the way: why exactly did you change the GEnergies to std::vector<GEnergy> there?

Apparently something got mixed up in the merging. Initially I had std::vector<GEnergy> but then I introduced the GEnergies class. I see that the 576-introduce-CTAModelBackground branch still contained the old version. I thus merged in again the correct version using GEnergies and re-changed the test_GModel.cpp file and pushed back to origin smile.png

#35 - 11/18/2013 10:17 PM - Knödlseder Jürgen

Mayer Michael wrote:

> Great. I merged my branch with the new code. I also created a test script in the CTA module test_cta_bgcube.py. I directly adapted this script from yours. The script does not need any additional files, so the size of the repo stays the same.

I had to slightly adjust the test file access path and to use the scatter plot instead of hist2d to make the script work. The former was probably related to your specific gammalib installation, the latter is due to my probably too old matplotlib version. I changed the code slightly so that it will check whether hist2d is available. If yet it will use this routine, otherwise it will generate a scatter plot.

#36 - 11/18/2013 10:19 PM - Knödlseder Jürgen

I was wondering whether you have tested the new code already on real data (to see whether all normalizations are done correctly)?

#37 - 11/18/2013 10:43 PM - Mayer Michael

Knödlseder Jürgen wrote:

> I was wondering whether you have tested the new code already on real data (to see whether all normalizations are done correctly)?

Yes. I was running my scripts which analysed a large amount of HESS off runs. The results looked very promising, i.e normalisations were around 1 and spectral index around 0. The code seems to work and give the same results as before.

> Indeed, GModelSpectralNodes computes the log of the intensities hence it does not like zero fluxes. There is nothing that catches this problem. I created bug #990 for this.
> Adding the if statement as you did is certainly a work around. At the same time, having a diffuse cube with an empty map does not make really sense (model values also need to be positive), hence one could also think about removing these maps internally. We should at least watch if this case has no other side effects.

Of course, non-positive model values don't make sense. Especially, if we had an event where the model predicts zero, the fit might have a problem. Currently, I circumvent this problem by applying a safe energy threshold with ctselect. Probably, removing these maps internally is the safest way to deal with that. However, this might cause confusion when reading in such a map and saving it afterwards - the file will have changed without any obvious reason to the user...

> I had to slightly adjust the test file access path and to use the scatter plot instead of hist2d to make the script work. The former was probably related to your specific gammalib installation, the latter is due to my probably too old matplotlib version. I changed the code slightly so that it will check whether hist2d is available. If yet it will use this routine, otherwise it will generate a scatter plot.

Ok great, I didn't know that hist2d was specific to a newer version. I looked on my system: I use version 1.3.0

#38 - 11/19/2013 02:20 AM - Knödlseder Jürgen

See the ctools feature #988:

I removed the area computation from GCTAModelBackground::mc as the spectrum extracted from the map takes already into account the solid angle of the pixels, hence units are cts/s.

#39 - 11/19/2013 10:23 AM - Knödlseder Jürgen

- % Done changed from 70 to 100

I did some rework of the GCTARoi and GCTAInstDir classes to allow for an abstract check of containment of an event within an ROI. This code is also used in the new ctools version that is in branch 988-support-GCTABackground-in-ctobssim.

It looks to me that the code is now pretty much ready. Let me know if it's okay for you to merge the feature into devel.

#40 - 11/19/2013 11:21 AM - Mayer Michael

- Status changed from Feedback to Pull request

Great. All changes we made in the last days did not affect the ctlike results. Therefore, in my opinion as well, the code could now be merged. Of course, there are still some things we need to keep in mind:

- do we always integrate the model starting from the pointing position
- check integration precision vs speed
- in the future, should this class also support arbitrary models (e.g. FITS-files) given in instrument coordinates?

I think, however, these items can be subject to future issues.

#41 - 11/19/2013 11:39 AM - Knödlseder Jürgen

Mayer Michael wrote:

> Great. All changes we made in the last days did not affect the ctlike results. Therefore, in my opinion as well, the code could now be merged. Of course, there are still some things we need to keep in mind:
>
> - do we always integrate the model starting from the pointing position

Are you sure that you integrate from the pointing position and not the ROI centre? I guess in all the test done so far both positions are identical, but in the code I see that the ROI centre goes in the rotation matrix that transforms local coordinates to celestial coordinates. The local coordinates should therefore be in the ROI system.

I'll do some tests ...

> - check integration precision vs speed

Agree.

> - in the future, should this class also support arbitrary models (e.g. FITS-files) given in instrument coordinates?

Can you think about more arbitrary models than a diffuse map cube?

> I think, however, these items can be subject to future issues.

Agree also. We want to get the code in the trunk so that others can use it.

#42 - 11/19/2013 11:50 AM - Mayer Michael

> Are you sure that you integrate from the pointing position and not the ROI centre? I guess in all the test done so far both positions are identical, but in the code I see that the ROI centre goes in the rotation matrix that transforms local coordinates to celestial coordinates. The local coordinates should therefore be in the ROI system.

Oups, yes I meant ROI - which was the same as pointing in my thinking. I mean, if. e.g. a model peaks not in the center but is slightly off, maybe it would be more accurate to always integrate from the model peak. This would cost additional computation time to find the model peak. Therefore, I think for now it should be fine like this.

> Can you think about more arbitrary models than a diffuse map cube?

I guess I had an error in my thinking. I meant to have the model stored in absolut instrument coordinates x and y and not a sky image. But I guess since the rotation is already given this shouldn't be an issue.

> Agree also. We want to get the code in the trunk so that others can use it.

affirmative smile.png

#43 - 11/19/2013 12:21 PM - Knödlseder Jürgen

Mayer Michael wrote:

> Are you sure that you integrate from the pointing position and not the ROI centre? I guess in all the test done so far both positions are identical, but in the code I see that the ROI centre goes in the rotation matrix that transforms local coordinates to celestial coordinates. The local coordinates should therefore be in the ROI system.

Oups, yes I meant ROI - which was the same as pointing in my thinking. I mean, if. e.g. a model peaks not in the center but is slightly off, maybe it would be more accurate to always integrate from the model peak. This would cost additional computation time to find the model peak. Therefore, I think for now it should be fine like this.

Finding the model peak looks quite tricky. I just went through the code and recognized that the pointing is not used at all in the mc() method, so I threw out the retrieval of the pointing.

For the Npred method I have still to check if things are working as expected.

> Can you think about more arbitrary models than a diffuse map cube?

I guess I had an error in my thinking. I meant to have the model stored in absolut instrument coordinates x and y and not a sky image. But I guess since the rotation is already given this shouldn't be an issue.

I see. We can indeed see how this could be implemented.

> Agree also. We want to get the code in the trunk so that others can use it.

affirmative smile.png

#44 - 11/19/2013 01:41 PM - Knödlseder Jürgen

I made some test runs where I simulated events within a 1 deg radius around the centre of the diffuse map cube and where I set the ROI to a 0.5 deg radius region slightly offset from the diffuse map centre (taken as the pointing).

The original code gave the following ctlike results:

```
2013-11-19T12:25:21: === GCTAPointing ===
2013-11-19T12:29:06:  Pointing direction .....: (RA,Dec)=(84.1726,22.0144)
2013-11-19T12:29:06: === GCTARoi ===
2013-11-19T12:29:06:  ROI centre .....: RA=84, DEC=22.2
2013-11-19T12:29:06:  ROI radius .....: 0.5 deg
...
2013-11-19T12:37:29:  Maximum log likelihood ....: -506.611
2013-11-19T12:37:29:  Observed events  (Nobs) ...: 662
2013-11-19T12:37:29:  Predicted events (Npred) ..: 662 (Nobs - Npred = 5.25365e-08)
...
2013-11-19T12:37:29:  Number of spectral par's ..: 3
2013-11-19T12:37:29:   Prefactor .....: 2.05303 +/- 0.118683 [0.001,1000] ph/cm2/s/MeV (free,scale=1,gradient)
2013-11-19T12:37:29:   Index .....: -0.238619 +/- 0.0500757 [-5,5]  (free,scale=1,gradient)
2013-11-19T12:37:29:   PivotEnergy .....: 1e+06 [10000,1e+09] MeV (fixed,scale=1e+06,gradient)
```

Obviously, the prefactor is pretty off, also the index is not as expected.

I then modified the code in GCTAModelBackground to integrate only within the ROI. This can be define for the moment by enabling the compile option G_NPRED_AROUND_ROI. This leads to the following ctlike results:

```
2013-11-19T12:25:21: === GCTAPointing ===
2013-11-19T12:25:21:  Pointing direction .....: (RA,Dec)=(84.1726,22.0144)
2013-11-19T12:25:21: === GCTARoi ===
2013-11-19T12:25:21:  ROI centre .....: RA=84, DEC=22.2
2013-11-19T12:25:21:  ROI radius .....: 0.5 deg
...
2013-11-19T12:39:41:  Maximum log likelihood ....: -832.651
2013-11-19T12:39:41:  Observed events  (Nobs) ...: 662
2013-11-19T12:39:41:  Predicted events (Npred) ..: 662 (Nobs - Npred = 6.52904e-09)
...
2013-11-19T12:39:41:  Number of spectral par's ..: 3
2013-11-19T12:39:41:   Prefactor .....: 1.01222 +/- 0.0585154 [0.001,1000] ph/cm2/s/MeV (free,scale=1,gradient)
2013-11-19T12:39:41:   Index .....: 0.0125696 +/- 0.0500757 [-5,5]  (free,scale=1,gradient)
2013-11-19T12:39:41:   PivotEnergy .....: 1e+06 [10000,1e+09] MeV (fixed,scale=1e+06,gradient)
```

Now, everything is nominal.

I'll make a commit with the G_NPRED_AROUND_ROI compile option. We may later remove the old code and the compile option.

#45 - 11/19/2013 02:19 PM - Knödlseder Jürgen

- Status changed from Pull request to Closed

Merged into devel.

#46 - 12/11/2013 10:18 PM - Knödlseder Jürgen

- Target version deleted (00-08-00)

#47 - 07/11/2014 04:15 PM - Knödlseder Jürgen

- Target version set to 1.0.0

Files

GCTAModelBackground.cpp	40.7 KB	10/10/2013	Mayer Michael
GCTAModelBackground.hpp	7.91 KB	10/10/2013	Mayer Michael
GCTAModelBackground.i	3.64 KB	10/10/2013	Mayer Michael
cta_background_model_test.xml	838 Bytes	10/11/2013	Mayer Michael