# GammaLib - Bug #1556

## Flux-methods of spectral models give incorrect results

10/19/2015 01:57 PM - Mayer Michael

| | | | | |
|---|---|---|---|---|
| **Status:** | Closed | | **Start date:** | 10/19/2015 |
| **Priority:** | Urgent | | **Due date:** | |
| **Assigned To:** | Knödlseder Jürgen | | **% Done:** | 100% |
| **Category:** | | | **Estimated time:** | 0.00 hour |
| **Target version:** | 1.0.0 | | | |

**Description**

Trying, e g. the following code in python:

```
import gammalib
m = gammalib.GModels("$GAMMALIB/test/data/model_point_eplaw.xml")
flux = m[0].spectral().flux(gammalib.GEnergy(1.0,"TeV"), gammalib.GEnergy(2.0,"TeV"))
print flux
```

Currently returns 465088315870.0, which is way too large. I have tested the same thing with the log parabola model and also get unrealistically large numbers in the flux() and eflux() method. It seems that the integration is corrupt since the power law model (where the integration is done analytically) is fine.
Could this be a problem of the integrator with small numbers?

**History**

**#1 - 10/26/2015 11:02 PM - Knödlseder Jürgen**

*- File test.py added*

*- File model_point_eplaw.xml added*

*- File model_point_logparabola.xml added*

*- Status changed from New to In Progress*

*- Assigned To set to Knödlseder Jürgen*

*- % Done changed from 0 to 10*

I tried to reproduce the problem by executing the attached test script (your code):

```
python -c 'import test'
3.31592415503e-12
```

Given that the prefactor is 5.7e-16 ph/cm2/s/MeV and a 1e6 integration range (1 TeV) the result looks reasonable. Also the attached log parabola looks reasonable.

Can you double check with the attached script and models?

**#2 - 10/27/2015 10:51 AM - Mayer Michael**

This is interesting and puzzling: on my Mac (OS X 10.11), I get the same number you obtain above.
However, on the batch farm at DESY Zeuthen, I get this large number for the exponential cut off. For the log parabola model, the integral doesn't converge:

+++ WARNING in GIntegral::romberg(1e+06, 2e+06, 5): Integration uncertainty nan exceeds absolute tolerance of nan after 21 iterations. Result -nan is inaccurate.

I made a clean new clone of the repository and recompiled. Still I get the same incorrect result. The operation system is "Scientific Linux release 6.6 (Carbon)".

**#3 - 10/27/2015 12:15 PM - Knödlseder Jürgen**

I did the same on a slightly older Scientific Linux version here (6.2) and got

```
$ python -c 'import test'
3.31592415503e-12
```

I'm wondering whether this could be related to storing references in the integration kernel instead of values (I think I had this kind of problem before). Could you go in GModelSpectralExpPlaw.hpp and change the code as follows (remove the & and the const for m_norm and m_index):

```
// Photon flux integration kernel
class flux_kernel : public GFunction {
public:
    flux_kernel(const double& norm,
            const double& index,
            const double& pivot,
            const double& ecut) :
            m_norm(norm),
            m_index(index),
            m_inv_pivot(1.0/pivot),
            m_inv_ecut(1.0/ecut) {}
    double eval(const double& eng);
protected:
    double   m_norm;      //!< Normalization
    double   m_index;     //!< Index
    double        m_inv_pivot; //!< 1 / Pivot energy
    double        m_inv_ecut;  //!< 1 / Cut off energy
};
```

**#4 - 10/27/2015 01:32 PM - Mayer Michael**

*- % Done changed from 10 to 80*

Thanks for the solution. I changed the code and recompiled - now it works fine!
Result for Exponential cut off is: 3.31592415503e-12

I had to run make clean and rebuild before it actually worked. Do we have to adapt all flux and eflux methods of the spectral models to follow this scheme?

**#5 - 10/27/2015 02:12 PM - Knödlseder Jürgen**

I think that this would be more save.

References are heavily used to store parameters for kernel classes, also within the CTA interface.

The problem occurs if a reference goes out of scope (i.e. the value is actually ill defined). Sometimes the corresponding memory cell keeps the old value, but sometimes a new value will be written in the cell, leading to corruption. The behavior is typically compiler / machine dependent, that's I think why I could not reproduce the problem.

So far I have taken the position to carefully code so that no problems occur (you basically have to make sure at the client side that the reference is always valid). It's not fully clear to me why there is a problem in the specific case you discovered, but the fact that the results were corrupted is worrisome.

The actual code that gets executed is

```
    // Setup integration kernel
    flux_kernel integrand(m_norm.value(),  m_index.value(),
                m_pivot.value(), m_ecut.value());
    GIntegral integral(&integrand);
```

Here we have to make sure that the references of m_norm.value() and m_index.value() are valid. The value() method returns a value (not a reference), hence I would expect that the reference to these values are also valid after the

```
flux_kernel integrand(m_norm.value(),  m_index.value(), m_pivot.value(), m_ecut.value());
```

call.

I had resolved this problem when I discovered it in the past (need to find the corresponding issue) by saving the actual values explicitly in local variables, in this case this would be

```
    // Setup integration kernel
    double norm = m_norm.value();
    double index = m_index.value();
    flux_kernel integrand(norm,  index,
                m_pivot.value(), m_ecut.value());
    GIntegral integral(&integrand);
```

This code makes sure that the references to norm and index are valid through the integration step, as variables always remain valid in a given scope (brackets). Variables being provided as function arguments apparently do not need to stay valid after the function is called.

You may try if this alternative fix also solves your problem.

We should however create an issue to change all references to values (the question is still how to handle "big" objects; we don't want to make a copy of a GCTAObservation). Maybe a reasonable rule would be to store pointers for "big" objects and values for integers, doubles, etc. (the fact of storing a pointer makes it more explicit that the client needs to take care about the pointer being valid; specifically this avoids encountering the reference problem we have here).

**#6 - 10/27/2015 02:57 PM - Mayer Michael**

Thanks for the detailed explanations. The alternative solution you proposed (storing the values into local variables) did also work fine.
I am not sure why only the norm and index are stored by reference and the other two parameters m_pivot and m_ecut aren't.

Do we really want to change all references to values? Sounds like a lot of effort especially on the code verification side. I guess as long such a problem can be fixed by storing the values to local variables it should be ok, right?

**#7 - 10/27/2015 03:32 PM - Knödlseder Jürgen**

Mayer Michael wrote:

> Thanks for the detailed explanations. The alternative solution you proposed (storing the values into local variables) did also work fine.
> I am not sure why only the norm and index are stored by reference and the other two parameters m_pivot and m_ecut aren't.

Because they are stored as their inverse values, hence I cannot use their original references.

> Do we really want to change all references to values? Sounds like a lot of effort especially on the code verification side. I guess as long such a problem can be fixed by storing the values to local variables it should be ok, right?

The problem is that we have no way to systematically track the problem. There could be response computations that are wrong on some machines, but the user may not easily recognize this. I think we should go on the save side.

I'm not worried about code verification. This are done pretty automatically, I'm also about to implement an automatic science validation in ctools which generates all pull distributions that are needed to assure the code is okay (I think we can live with typically 100 pulls, reducing the time for science verification to a few days; this is done as a background job in Jenkins).

**#8 - 10/27/2015 03:39 PM - Mayer Michael**

Ok sounds reasonable to me. How do we proceed? Do we fix the problem right now by using the solution of storing the values to local variables. Then setup an issue to change the storage from references to values? Or you prefer to skip the first step?

**#9 - 10/27/2015 04:18 PM - Knödlseder Jürgen**

I prefer to skip the local variables and change directly the const references to actual values. This also reduces the code changes (you just need to redefine the interface in the header files and recompile).

**#10 - 10/27/2015 05:02 PM - Mayer Michael**

Ok I understand. Here a list of spectral models that need to be adapted:

- GModelSpectralSuperExpPlaw.hpp
- GModelSpectralLogParabola.hpp
- GModelSpectralGauss.hpp
- GModelSpectralExpPlaw.hpp

I am not sure regarding the response computations. Probably most of it are contained in GCTAResponse_helpers.cpp and in GCTAModel*.hpp files, right? I'll probably have to look through other instrument modules as well?

**#11 - 10/27/2015 05:13 PM - Knödlseder Jürgen**

Most is in indeed in the response helpers, but also the GCTAResponseXXX classes.

I found the following in the instrument modules:

```
grep -R GFunction inst/*
inst/cta/include/GCTABackgroundPerfTable.hpp:    class integrand : public GFunction {
inst/cta/include/GCTAEdisp2D.hpp:    class edisp_kern : public GFunction {
inst/cta/include/GCTAEdispRmf.hpp:    class edisp_kern : public GFunction {
inst/cta/include/GCTAModelAeffBackground.hpp:    class npred_roi_kern_theta : public GFunction {
inst/cta/include/GCTAModelAeffBackground.hpp:    class npred_roi_kern_phi : public GFunction {
inst/cta/include/GCTAModelIrfBackground.hpp:    class npred_roi_kern_theta : public GFunction {
inst/cta/include/GCTAModelIrfBackground.hpp:    class npred_roi_kern_phi : public GFunction {
inst/cta/include/GCTAModelRadialAcceptance.hpp:    class roi_kern : public GFunction {
inst/cta/include/GCTAModelRadialGauss.hpp:    class integrand : public GFunction {
inst/cta/include/GCTAModelRadialPolynom.hpp:    class integrand : public GFunction {
inst/cta/include/GCTAModelRadialProfile.hpp:    class integrand : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_npsf_kern_rad_azsym : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nedisp_kern : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_kern : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_irf_radial_kern_rho : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_irf_radial_kern_omega : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_radial_kern_rho : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_radial_kern_omega : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_irf_elliptical_kern_rho : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_irf_elliptical_kern_omega : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_elliptical_kern_rho : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_elliptical_kern_omega : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_irf_diffuse_kern_theta : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_irf_diffuse_kern_phi : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_diffuse_kern_theta : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_nroi_diffuse_kern_phi : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_radial_kern_rho : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_radial_kern_omega : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_radial_kern_delta : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_radial_kern_phi : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_elliptical_kern_rho : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_elliptical_kern_omega : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_diffuse_kern_delta : public GFunction {
inst/cta/src/GCTAResponse_helpers.hpp:class cta_psf_diffuse_kern_phi : public GFunction {
```

and the following in the include directory

```
grep -R GFunction include
include/GModelSpectralExpPlaw.hpp:    class flux_kernel : public GFunction {
include/GModelSpectralExpPlaw.hpp:    class eflux_kernel : public GFunction {
include/GModelSpectralGauss.hpp:    class eflux_kernel : public GFunction {
include/GModelSpectralLogParabola.hpp:    class flux_kern : public GFunction {
include/GModelSpectralSuperExpPlaw.hpp:    class flux_kernel : public GFunction {
include/GModelSpectralSuperExpPlaw.hpp:    class eflux_kernel : public GFunction {
include/GObservation.hpp:    class model_func : public GFunction {
include/GObservation.hpp:    class npred_kern : public GFunction {
include/GObservation.hpp:    class npred_spec_kern : public GFunction {
include/GObservation.hpp:    class npred_func : public GFunction {
include/GResponse.hpp:    class edisp_kern : public GFunction {
```

When going through all files we should carefully consider how to change const refs. I would in a first pass only recommend to change the doubles, integers, etc. to avoid duplication of potentially large structures.

If you prefer you can just do the spectral models and I can go over the CTA response part.

**#12 - 10/27/2015 05:32 PM - Mayer Michael**

If you prefer you can just do the spectral models and I can go over the CTA response part.

As you wish, you are the boss :)
Just let me know the branch you are working on. Feel free to also pass some of the response files to me.

When going through all files we should carefully consider how to change const refs. I would in a first pass only recommend to change the doubles, integers, etc. to avoid duplication of potentially large structures.

Agreed.

**#13 - 10/27/2015 06:30 PM - Knödlseder Jürgen**

I started to work on the CTA part on a local branch (we better do not work on the same branch as this might lead to conflicts). I will also work on GResponse and GObservation.

Just create a new branch to work on the model part.

**#14 - 10/27/2015 09:09 PM - Knödlseder Jürgen**

*- % Done changed from 80 to 90*

I finally did the job for everything (hope you did not already start working on that).

All unit tests run successfully. I'm about to integrate the code.

**#15 - 10/27/2015 09:51 PM - Knödlseder Jürgen**

*- Status changed from In Progress to Feedback*
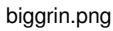
*- % Done changed from 90 to 100*

Merged into devel.

Can you check if everything is okay on the DESY batch farm?

**#16 - 10/28/2015 01:40 PM - Mayer Michael**

Thanks a lot. Indeed I did not start to work on that. I've checked the flux and eflux methods of all spectral models and they all return reasonable results now. Many thanks for the quick integration.

**#17 - 10/28/2015 02:12 PM - Knödlseder Jürgen**

*- Status changed from Feedback to Closed*

This was the last action to close for the GammaLib 1.0 release  biggrin.png

**#18 - 10/28/2015 02:15 PM - Mayer Michael**

Excellent!  smile.png

## Files

| | | | |
|---|---|---|---|
| test.py | 213 Bytes | 10/26/2015 | Knödlseder Jürgen |
| model_point_eplaw.xml | 870 Bytes | 10/26/2015 | Knödlseder Jürgen |
| model_point_logparabola.xml | 848 Bytes | 10/26/2015 | Knödlseder Jürgen |