{{lastupdated_at}} by {{lastupdated_by}}

Development Guidelines

Scope

This page summarizes guidelines for the GammaLib development. We recommend that you follow these guidelines as closely as possible during your GammaLib developments.

Code optimization

This section summarizes guidelines for code optimization. The purpose of code optimization is to reduce computing time and memory requirements. Here a couple of useful links or documents:

- attachment:"Optimizing_software_in_C++.pdf"

Computations

GammaLib makes extensive use of mathematical functions, such as sin, cos, pow, log10, etc. Care has to be taken when using these functions, as improper usage may quickly lead to considerable speed penalties. Before you continue, you may learn about this by checking the [[Computation Benchmarks]] that were performed on various platforms.

From those benchmarks, the following coding rules were derived:

1. Only call mathematical function when necessary (cache once computed values). Mathematical functions are computing intensive.
2. Use double precision throughout all function calls. Modern FPUs are optimized for double precision, and you may get large speed penalties when using floating point versions of trigonometric functions (see details in [[Computation Benchmarks]]).
3. Use multiplication instead of division where possible. Even on modern systems, multiplications are twice as fast as divisions!
4. Use multiplication instead of pow for integer exponents. Multiplications are always faster than pow calls.
5. Avoid pow and exp when possible. For example, pow(b, 0.5*(log_b(x) + log_b(y))) can be written as sqrt(x*y), which is considerably faster.

Code structure

Method arguments

1. Arguments should be passed as const reference to avoid non necessary copying of values.
2. The number of arguments should be as small as possible. This reduces the overhead created when accessing a method. For a justification, read the section about [[Method-call overheads]].

Thread save coding

GammaLib makes use of OpenMP in various places. This imposes some thread save programming. Code sections that should not be executed in parallel, such as reading and writing from files, should be enclosed in a protective statement:

```
#pragma omp critical
{
   const_cast<GCTAObservation*>(this)->load(m_eventfile);
}
```

More reading

This section provides links to C++ coding guidelines:

- http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

Files

Optimizing_software_in_C++.pdf	870 KB	03/26/2013	Knödlseder Jürgen