

**GammaRay version 2.5.1**

# Introduction

GammaRay is a graphical user interface (GUI) that automates geostatistical workflows by driving and coordinating the several modules of the renowned Geostatistical Software Library (GSLib). GSLib was originally developed at Stanford University in 1992, under coordination of Prof. Andre Journel. Since then, several GSLib packages emerged, but the GSLib distribution currently considered as reference is developed and maintained by Prof. Dr. Clayton V. Deutsch and his fellow researchers at the Centre for Computational Geostatistics (CCG) of University of Alberta. GammaRay is named after the Greek letter gamma commonly used to represent the variogram, a model of spatial correlation for variables and for which geostatistics is broadly known.

One of Prof. Clayton’s courses on advanced multivariate geostatistics while I was an MSc student at the Mining Planning Laboratory (LPM) of the School of Engineering in Federal University of Rio Grande do Sul (UFRGS), advised by Prof. Dr. João Felipe, presented complex workflows involving several GSLib modules. The error-prone and time-consuming task of manually managing dozens of files, even with the help of shell/prompt scripts, and my Computer Science background inspired me to write GammaRay.

The main purpose of GammaRay is to add a user-friendly interface layer on top of the scientifically and numerically robust GSLib, greatly automating parameter file editing and module chaining so the practitioner can focus on geostatistics. GammaRay was conceived as a free and portable alternative to WinGSLib®, which is a commercial software available only for Microsoft Windows® users. GammaRay is also open source and thus is subject to code review and can receive contributions from other software developers as well as user feedback, suggestions and bug reports. GammaRay is built with the C++ programming language upon the famous Qt library to leverage the construction of a modern and platform independent graphical user interface.

What is GammaRay not? A) A tool for a naive user: GammaRay is not meant to throw in data and after some button pushing, grade tonnage curves or petroleum reservoir models are ready. Although GammaRay extensively helps with parameter setting and module chaining, it is not an abstraction layer over geostatistics, as it still requires full understanding of geostatistical concepts and full control of all GSLib modeling parameters. If you are new to geostatistics, please consider reading a geostatistics primer (REF). B) A toy program: despite being free and with a relatively simple interface, GammaRay can be used in research and industrial applications because all calculation is actually performed by the time-tested GSLib.

GammaRay is designed so the user or the IT support can easily repair her/his projects should things go awry. For instance, a GammaRay project is simply a directory with a file named gammaray.prj (a human readable text file) and all the project files in their original formats in it. There are no fancy binary databases, non-standard file formats, remote cloud storage or obscure registry settings to be concerned with. Also, GammaRay does not change the original data files to keep its project information nor converts them to another format for internal use so the data files remain readable by other software. The program keeps its information about files (e.g. which variables are X, Y and Z coordinates or grid dimensions) in separate human-readable text files called metadata files (.md extension) in the project directory. The user can also abort GammaRay in the middle of an on-going computation without fearing data corruption as it operates on temporary files before committing results to project files. Even though GammaRay was designed to minimize impacts of software misbehavior, the user must observe standard backup routines.

This manual presents only the practical aspects of geostatistics as to allow the user get started with the program functions. For theory and basics of geostatistics, there are already dozens of good tomes out there (this manual will not explain what a variogram is, for example). It is also advisable to be acquainted with the GSLib parameters and their meaning. If you are new to GSLib, you can learn by experimentation in GammaRay or referring to its manual (REF). The <http://www.statios.com/> website has on-line descriptions of GSLib programs (ex.: <http://www.statios.com/help/kt3d.html> ). Universities worldwide may also have GSLib material available online.

## Licensing

CC-BY-SA icon

GammaRay is licensed under the Creative Commons license of type Attribution/Share-alike (CC-BY-SA) version 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/deed.en>). Briefly, it means that you can freely use, redistribute, modify or create derivative work from it, even for commercial purposes. On the other hand, you must always give credit to the authors (the original author and the code contributors) and derivative work cannot have more restrictive licensing (e.g. CC-BY-ND or Copyright).

A derivative work is any work that uses GammaRay code or executable: a new code branch of GammaRay; software or printed work with part or the whole of its code; a software bundle that includes unmodified GammaRay code or executable are examples. The following are NOT derivative works for licensing purposes: graphics, variograms, models, figures, data files, reports, research papers, theses, maps, estimates, simulations, or any work produced by merely USING GammaRay.

## Disclaimer

The authors cannot be held responsible for any kind of loss caused by using the software. The software obviously does not have any warranty. GammaRay depends on GSLib and GhostScript to function, which are not part of the software and are not even maintained by the authors. Their acquisition, proper installation and normal operation check is entirely a user responsibility. GammRay will not perform all its functions if those software are not working properly or parts of them are missing.

The keepers of the repository cannot be held responsible for code contributions that were taken from another piece of work without express permission and/or credits given. Any identified and verified plagiarism will be removed from the code without prior notice with its contributors subject to being denied write access to the repository permanently. Conversely, reports or claims of plagiarism must be backed up by evidence that such code was published before the addition to the repository.

Without further notice, by publishing content to the repository, the contributors tacitly declare that the code was of her/his authorship.

## Donations

If you feel like making a money contribution to support further GammaRay development because it is helpful, please consider helping any charity project in your community or any worldwide humanitarian effort that is politically neutral such as Medecins Sans Frontieres. GammaRay continuity is assured by being open source, thus other people can make improvements, updates, bug fixes and executables for future operating system versions independently of financial support or authors availability.

Summary

[1 Introduction 1](#_Toc488569412)

[1.1 Licensing 2](#_Toc488569413)

[1.2 Disclaimer 2](#_Toc488569414)

[1.3 Donations 3](#_Toc488569415)

[2 Dependencies 8](#_Toc488569416)

[2.1 GSLib 8](#_Toc488569417)

[2.1.1 Note on GSLib compatibility 9](#_Toc488569418)

[2.2 Ghostscript 9](#_Toc488569419)

[2.3 Compile- and runtime libraries 9](#_Toc488569420)

[2.3.1 Developer tool set 9](#_Toc488569421)

[2.3.2 MSYS2: a GNU-like development environment for Windows 9](#_Toc488569422)

[2.3.3 Qt 5.3 10](#_Toc488569423)

[2.3.4 Boost 1.63 10](#_Toc488569424)

[2.3.5 Bug in Boost’s **matrix\_expression.hpp** 10](#_Toc488569425)

[2.3.6 Qwt 6.1.3 11](#_Toc488569426)

[2.3.7 VTK 6.3 11](#_Toc488569427)

[2.3.8 VTK compiling issues 12](#_Toc488569428)

[2.3.9 Windows: Debug x Release 13](#_Toc488569429)

[2.3.10 Windows with GCC: 32 or 64-bit 13](#_Toc488569430)

[3 Contributing with code 13](#_Toc488569431)

[3.1 The Git versioning system 13](#_Toc488569432)

[3.2 Downloading the sources repository 14](#_Toc488569433)

[3.3 Creating a branch 14](#_Toc488569434)

[3.4 Downloading an existing branch 15](#_Toc488569435)

[3.5 Keeping tool-specific files from version control 15](#_Toc488569436)

[3.6 Committing changes 15](#_Toc488569437)

[3.7 Uploading a branch 16](#_Toc488569438)

[3.8 Pull request 16](#_Toc488569439)

[3.8.1 Tolerated compiler warnings 17](#_Toc488569440)

[3.9 Updating your local repository 17](#_Toc488569441)

[3.10 File operations 17](#_Toc488569442)

[3.11 Issues using Git 18](#_Toc488569443)

[4 Getting GammaRay 18](#_Toc488569444)

[4.1 Installing (Windows users) 18](#_Toc488569445)

[4.2 Compiling (developers and non-Windows users) 18](#_Toc488569446)

[4.2.1 Configuring 19](#_Toc488569447)

[4.2.2 Building 20](#_Toc488569448)

[4.2.3 Deploying 20](#_Toc488569449)

[5 Getting started 21](#_Toc488569450)

[5.1 Configuring GammaRay 21](#_Toc488569451)

[5.2 Creating a project 22](#_Toc488569452)

[5.3 Adding data files to your project 23](#_Toc488569453)

[5.4 Removing data files from the project 25](#_Toc488569454)

[5.5 Freeing up memory by freeing loaded data 25](#_Toc488569455)

[5.6 Conventions 26](#_Toc488569456)

[5.7 Plot Dialog 26](#_Toc488569457)

[5.8 Parameters Dialog 27](#_Toc488569458)

[5.9 3D Viewer 28](#_Toc488569459)

[5.9.1 Known issues 29](#_Toc488569460)

[5.10 Known issues trying to launch the program 29](#_Toc488569461)

[6 Exploratory data analysis 29](#_Toc488569462)

[6.1 Map / data display 29](#_Toc488569463)

[6.1.1 Known issues 30](#_Toc488569464)

[6.2 Histogram 31](#_Toc488569465)

[6.2.1 Known issues 31](#_Toc488569466)

[6.3 Crossplot 31](#_Toc488569467)

[6.4 Probability plot 32](#_Toc488569468)

[6.5 Q-Q and P-P plots 33](#_Toc488569469)

[6.6 Univariate distribution modeling 34](#_Toc488569470)

[6.6.1 Known issues 35](#_Toc488569471)

[6.7 Bivariate distribution modeling 35](#_Toc488569472)

[6.7.1 Known issues 36](#_Toc488569473)

[6.8 Histograms of realizations 37](#_Toc488569474)

[6.9 Known issues with plots (**\*plt** programs in general) 38](#_Toc488569475)

[7 Data preparation 38](#_Toc488569476)

[7.1 Declustering 38](#_Toc488569477)

[7.1.1 Maximum or minimum? 39](#_Toc488569478)

[7.2 Transfer collocated values 40](#_Toc488569479)

[7.3 Creating a grid for estimation or simulation 40](#_Toc488569480)

[7.4 Convert a grid to point set 41](#_Toc488569481)

[7.5 Look for duplicate samples or samples too close 41](#_Toc488569482)

[7.5.1 Known issues 42](#_Toc488569483)

[7.6 Resampling large grids 42](#_Toc488569484)

[8 Data transforms 43](#_Toc488569485)

[8.1 Normal Score Transform 43](#_Toc488569486)

[8.2 Fourier Transform 43](#_Toc488569487)

[9 Data classification 44](#_Toc488569488)

[9.1 Category definition 44](#_Toc488569489)

[9.2 Univariate classification 45](#_Toc488569490)

[9.3 (CONSIDERING) Bivariate classification 46](#_Toc488569491)

[9.4 (CONSIDERING) Ternary diagram classification 46](#_Toc488569492)

[9.5 (CONSIDERING) Multivariate classification 46](#_Toc488569493)

[10 Variography 46](#_Toc488569494)

[10.1 Variogram map 47](#_Toc488569495)

[10.2 Experimental variogram 49](#_Toc488569496)

[10.3 Fitting a variogram model 51](#_Toc488569497)

[10.4 Cross variograms 51](#_Toc488569498)

[10.5 *A priori* variogram models 52](#_Toc488569499)

[10.6 Variograms of a realization ensemble in a single file 52](#_Toc488569500)

[10.7 Variograms of a realization ensemble across separate files 53](#_Toc488569501)

[11 Estimation 53](#_Toc488569502)

[11.1 Kriging 53](#_Toc488569503)

[11.1.1 Known issues 55](#_Toc488569504)

[11.2 Indicator kriging (IK) of a continuous variable 55](#_Toc488569505)

[11.2.1 Defining a threshold c.d.f. for a continuous variable 56](#_Toc488569506)

[11.2.2 Running IK for a continuous variable 57](#_Toc488569507)

[11.2.3 Post-processing the results 58](#_Toc488569508)

[11.2.4 Known issues 59](#_Toc488569509)

[11.3 IK for a categorical variable 59](#_Toc488569510)

[11.3.1 Defining a class p.d.f. for a categorical variable 60](#_Toc488569511)

[11.3.2 Running IK for a categorical variable 61](#_Toc488569512)

[11.3.3 Creating a maximum likelihood facies map 62](#_Toc488569513)

[11.3.4 Known issues 62](#_Toc488569514)

[11.4 Secondary data: soft indicator calibration 63](#_Toc488569515)

[11.4.1 Interpretation for categorical attributes 64](#_Toc488569516)

[11.4.2 Interpretation for continuous attributes 65](#_Toc488569517)

[11.4.3 Multiple secondary data 65](#_Toc488569518)

[11.4.4 Known issues 65](#_Toc488569519)

[11.5 Cokriging 65](#_Toc488569520)

[11.5.1 Cokriging variography 65](#_Toc488569521)

[11.5.2 The Linear Model of Coregionalization (LMC) 66](#_Toc488569522)

[11.5.3 Cokriging with **cokb3d** 67](#_Toc488569523)

[11.6 Estimation of unvalued cells of a grid (NDV estimation) 67](#_Toc488569524)

[11.7 (CONSIDERING) Highly multivariate estimation 68](#_Toc488569525)

[12 (PLANNED) Simulation 68](#_Toc488569526)

[12.1 Post-processing 68](#_Toc488569527)

[13 Advanced topics 69](#_Toc488569528)

[13.1 (CONSIDERING) Workflows 69](#_Toc488569529)

[13.2 (CONSIDERING) Extending GammaRay 69](#_Toc488569530)

[14 (WIP) References 69](#_Toc488569531)

# Dependencies

GammaRay depends on GSLib and GhostScript to fully perform its tasks. Furthermore, if you want to build GammaRay from the source code, you need to get some further software and libraries. Therefore, you must first get and install those software in your system.

## GSLib

GSLib is normally distributed as just a set of executables, so installing is just a matter of downloading a compressed archive and decompressing them to some directory. GSLib may also be found as a set of FORTRAN source codes that will require compilation for your operating system in case you are unable to find precompiled executables for it. You may find several different GSLib distributions out there, with more or less programs, thus if an essential executable is missing, GammaRay will not be able to perform some of its functions. Here is the list of GSLib executables and versions that GammaRay currently works with:

* addcoord (3.000)
* bivplt (3.000)
* cokb3d (3.000)
* declus (3.000)
* gam (3.000)
* gammabar (2.000)
* gamv (3.000)
* getpoints
* histplt (3.000)
* histpltsim[[1]](#footnote-2) (4.200)
* histsmth (3.000)
* ik3d (3.000)
* kt3d (3.000)
* locmap (2.906)
* nscore (3.000)
* pixelplt (2.905)
* postik (3.000)
* probplt (3.000)
* qpplot (3.000)
* scatplt (3.000)
* scatsmth (3.000)
* vargplt (3.000)
* varmap (3.000)
* vmodel (3.000)

You can know the program version by simply calling it passing an invalid path to a parameter file, the program will quit with an error but will print the version. GammaRay may work program versions different from the listed, but be aware of possibly changes to the expected parameter file, which may hinder functionalities. Since there is no actual GSLib version (each program has its own version), it is recommended an original and recent GSLib distribution from CCG, but this requires being an affiliate. Therefore, if you are not an employee of a mining or petroleum company nor a student of some School of Mines, you likely have to search for older ones in the internet. GammaRay is expected to function with old versions since the required programs are among the basic ones that have been around for a while. An alternative source of GSLib code and executables (Windows 32/64, Linux 32/64 and SunOS 2.6) is <http://www.gslib.com/> .

### Note on GSLib compatibility

It is known that some GSLib programs have different versions that accept different parameter files. If this is the case, the program may quit with an error with the parameter file that GammaRay generates to interface to those programs. sgsim, for instance, is known to have at least two versions: one that requires the covariance matrix be specified and an older one that does not. GammaRay is currently made to interface to the newer versions of GSLib programs. Therefore, if you suspect of GSLib program incompatibility, run the suspected program without arguments to generate a parameter file example. Then you can compare this example with the corresponding template in the templates directory inside the project directory (you need to create a project – see Section 5.2 – to generate the parameter file templates). If they differ, please contact the authors (main menu -> Help -> About… or the LICENSE.md file), but, if you cannot wait, you must look for a compatible version of the program. Support for different parameter file versions is being considered, but this requires carefully mapping all different parameter file versions that may be out there.

Alternatively, you can use incompatible programs as custom GSLib executables, for which require you to specify parameter file templates (see Section 13.2).

## Ghostscript

GhostScript can be downloaded (precompiled executables/installers or source codes) from its website: <http://www.ghostscript.com/>. GammaRay does not require GhostView since it has its own internal viewer. GammaRay was tested to work properly with version 8.53 (32- or 64-bit), so you need this version or newer, though the software is expected to work with somewhat older versions. Old versions (at least version 5.9 or earlier) have different installation directory structures, therefore will cause plots to fail.

## Compile- and runtime libraries

Read this section if you plan to compile GammaRay.

### Developer tool set

First, you obviously need a developer tool set of your choice that features a C++ compiler (GCC, MinGW, MinGW64, MSYS2, Visual Studio, etc.). Any version that supports the main features of C++11 standard suffices.

### MSYS2: a GNU-like development environment for Windows

MSYS2 (<http://www.msys2.org>) is an interesting open-source alternative to Visual Studio for Windows development/building. It includes a POSIX interface (based on Cygwin) and a GCC-like development tool set (based on MinGW and MinGW64 compilers) as well as a package manager (pacman) that enables you to download Qt, Qwt and VTK libraries precompiled for your system, greatly reducing your build efforts. To search for the available packages in MSYS2, enter pacman -Ss <search\_text> (e.g. pacman -Ss Qt5) in MSYS’s shell.

When you launch MSYS2, you are in just a POSIX interface in Windows (it is not an emulation or virtual machine). You obtain software by using the package manager, evoked with the pacman command in MSYS2’s shell. Pacman takes care to resolve any dependency and the libraries are expected to be compatible with each other. Hence, to prepare MSYS2 to be used to compile GammaRay, you run the following commands in MSYS2 shell:

To install MinGW64 development tools (compiler, linker, debugger, etc.): pacman -S mingw-w64-x86\_64-toolchain. Make sure you also update zlib with a pacman -S /mingw-w64-x86\_64-zlib, if you get an error about missing entrypoint of inflateValidate in libpng16-16.dll.

Also, you can use a 32-bit tool set and libraries or use GCC tools that are not MinGW. The possibilities are quite diverse.

WARNING: VTK libraries managed by MSYS2 was observed to cause poor Fourier Transform (Section 8.2) performance.

### Qt 5.3

Download (<https://www.qt.io/download/>) and install/compile Qt 5.3 or newer, considering that support by VTK for the latest version may be incipient (see Section 0). Qt is designed to be able to detect your installed toolsets and therefore be able to generate the build files (ex.: a Makefile for GNU-compliant devtool sets) for them. Please, refer to Qt documentation regarding the compiler you plan to use.

It is important that the Qt installation/compilation installs the include directory (with the .h C++ headers) and the lib directory (with the .so/.a/.dll/.lib files) in addition to the Qt runtime. Hence, Qt runtime-only packages will not be useful. The useful packages are known as “devel” or “SDK” packages. Some precompiled Qt packages out there are bundled with a toolchain (compiler, linker, etc.), so you download two requirements at once.

To install Qt 5 in MSYS2 (compiled with MinGW64): pacman -S mingw-w64-x86\_64-qt5

### Boost 1.63

GammaRay uses the header-only Boost.Geometry library. It is recommended that you download (<http://www.boost.org/>) the entire set of libraries as other Boost libraries may be required in the future. As a header-only library, Boost.Geometry requires only that you download Boost (no need for compiling it for the current or previous GammaRay versions).

### Bug in Boost’s matrix\_expression.hpp

As of version 1.64.0 of Boost, you need to find and patch the file matrix\_expression.hpp: look for the code below (around line 2224):

if (it2\_ != it2\_end\_)

if (it2\_.index1 () <= i\_)

++ it2\_;

if (it2\_ != it2\_end\_) {

index2 = it2\_.index1 ();

}

Judging by the indentation, the topmost if lacks the opening curly braces, which delimits its scope. So you need to move the { from the second innermost if to the topmost if. You are encouraged to take part in the Boost’s community mailing list (1000’s messages) and report this bug.

### Qwt 6.1.3

GammaRay uses the Qt for Technical Applications library to leverage features such as charts, plots and other science-oriented graphical user interface elements. Its home page is <http://qwt.sourceforge.net/>, from there you can find download links and instructions to compile and install the libraries. You do not need to compile it if you find the headers and pre-compiled libraries for your platform. Since Qwt is a layer on top of Qt, keep in mind the correct Qwt version compatible with your Qt version.

To install Qwt in MSYS2 (compiled with MinGW64): pacman -S mingw-w64-x86\_64-qwt-qt5

### VTK 6.3

The program uses the Visualization Toolkit (<http://www.vtk.org/download/>) for its 3D viewer. Any version greater than 6.3 is expected to work, but since GammaRay interfaces VTK via a Qt widget, you must check for correct support of your Qt version (you need to enable Qt support in VTK configuration). You will likely also need to download and install CMake (<https://cmake.org/download/>) to build VTK for your environment, unless you find the pre-compiled VTK libraries. Though it is strongly recommended that you build it because VTK is highly customizable and pre-compiled libraries out there may lack features required by GammaRay. **Figure 1** shows a minimum VTK build configuration (non-advanced, non-grouped view) that works with GammaRay. Any CMake version able to build VTK 6.3 suffices. CMake version 3.8.1 was used to build the VTK libraries distributed with the pre-compiled GammaRay for Windows.

For those unfamiliar with CMake, briefly it is a meta-make (like Qt’s qmake) that works in cycles. When you start configuring (e.g. with CMake GUI) you start filling any missing values, enabling options, selecting choices and commanding “Configure” again to expand the configuration into new settings until you arrive at a stable and complete configuration like the one in **Figure 1**. Then you command “Generate” to get a usable Makefile or Visual Studio Solution necessary to build VTK libraries. Pay special attention when specifying the correct generator when working with CMake: for example, if you plan to use MinGW’s make (mingw32-make), then do not select “MSYS makefiles” or “GCC makefiles” as the Makefiles for each of the different makes out there have subtle differences.



**Figure 1** Minimum workable VTK build configuration as seen in CMake GUI (non-advanced, non-grouped mode).

To install VTK in MSYS2 (compiled with MinGW64): pacman -S mingw-w64-x86\_64-vtk. NOTE: MSYS2’ VTK was observed to cause poor Fourier Transform (Section 8.2) performance.

### VTK compiling issues

VTK is one of the most challenging libraries to build for the unexperienced user. Its highly customizable build combined with cross platform support often result in issues. Only the observed or reported uncommon issues will be listed here. By uncommon means anything not easy to find by searching Google, Stack Overflow, etc.

PROBLEM: Configuring VTK with CMake, the process crashes with this message (does not appear in log): gcc.exe: error: ARGS: No such file or directory .

POSSIBLE CAUSE: This error is likely caused by a combination of a recent gcc (versions 6 and later as of 2017) and a relatively older VTK source (versions 6 or earlier). The regular expression defined in files vtkCompilerExtras.cmake and GenerateExportHeader.cmake that extracts the compiler version number from the output of gcc --version command may not account for the latest versions of gcc. By editing the file, one can spot it defined with [345] or even only [34] to capture the major version of gcc.

SOLUTION: Either switch to a more recent version of VTK or tweak its source code by editing vtkCompilerExtras.cmake and GenerateExportHeader.cmake, updating the offending regular expression to [3456] or [34567] depending on the version of gcc being used. Take care to not break these scripts as CMake scripts are not very easy to debug, so keep a backup copy of the files if you decide to edit them.

PROBLEM: Configuring VTK with CMake, an error related to missing Qt WebKit occurs.

POSSIBLE CAUSE: Your Qt build/package is missing WebKit. Possibly because the build was done without it enabled (the person who build Qt enabled the -no-webkit option when configuring Qt compilation). Many people opt to compile Qt without the Webkit because this component is known to be troublesome.

SOLUTIONS: a) Obtain a pre-compiled Qt containing the WebKit libraries; b) If you compiled Qt, make sure you did not enable the -no-webkit option and rebuild Qt; c) Disable the Qt group in VTK configuration (**Figure 1**), run Configure, switch to advanced mode, then disable only the Module\_vtkGUISupportQtWebkit option individually and enable the other Module\_vtkGUISupportQt\* options. You also need to check whether the Module\_vtkInteractionWidgets and Module\_vtkImagingMath are enabled as switching to advanced mode may disable other libraries required by GammaRay. Qt WebKit is only necessary for the vtkQtRichTextView class, which is not used by GammaRay.

PROBLEM: Building with make results in an unspecified error with the first source file it encounters.

POSSIBLE CAUSE: This was observed in MSYS2’s gcc. The directory containing GCC’s libraries is not in PATH environment variable, so GCC is unable to load its dependencies, leading to a crash without any message detailing its cause.

SOLUTION: add the missing path to the PATH variable (e.g. export PATH=<path>:$PATH command in POSIX systems).

### Windows: Debug x Release

If you plan to develop GammaRay under Windows, especially if you plan to use debuggers, it is important to provide or build both runtime and debug versions of the libraries. In Windows, running the program compiled in debug mode will result in crashes and/or unspecified behavior when linked/run against release versions of the dependencies.

### Windows with GCC: 32 or 64-bit

GammaRay code is designed to be compiled as either a 32 or a 64-bit application. In Windows, the 32 and 64-bit platforms are referred to in Qt world as x86 and x64 (sometimes x86\_64) respectively. When you configure Qt, the same mkspec file (the file containing the toolchain/platform/architecture specifications) is used to build 32 or 64-bit executables in Windows using gcc/g++ compiler: win32-g++. The available platform specs can be assessed in mkspecs directory where you installed Qt.

# Contributing with code

Users can always request improvements and report bugs, but the response depends on developer availability. Hence, the best way to improve GammaRay is to take an active development role and gain knowledge in exchange. GammaRay was made open source for this very reason. Using an IDE is optional, though it is recommended to use Qt Creator, especially if you plan to make changes to GUI.

## The Git versioning system

An open source project, potentially involving people worldwide, surely requires a version control system to manage potentially concurrent code changes as well as to prevent bad code making into GammaRay. Git is one of such systems, it is robust, allows a distributed code development and control takes place without blocking individual source code files. Git was then selected to manage the GammaRay source code repository.

Hence, it is recommended that potential developers be acquainted with the Git versioning system (<https://git-scm.com/>). For those not familiar with it, Git has a vast array of functionalities and a complex command syntax. Only the basic operations will be presented here to enable first-time developers to contribute. For complex matters such as solving code conflicts (rare with Git, but they do occur), operations with remote branches, code audit, etc., please refer to Git’s website, as they have a comprehensive documentation as well as a live tutorial where you can safely experiment with Git commands while a display graphically shows the virtual repository state.

For those uncomfortable with command line interfaces, Git comes with portable GUI tools (evoked with the git gui and gitk commands). There are other Git GUI options out there tailored for your OS, please refer to Git’s website or search the internet.

The Windows version of Git comes with a command console interpreter that emulates bash in Linux systems so all the Git commands are available without making changes to the Windows environment.

## Downloading the sources repository

A repository is the directory tree with the source code and the data files maintained by Git to keep accurate track of the minutest changes. To set up your local GammaRay repository, simply go to the directory where you plan to keep your local copy (called a clone) of the GammaRay sources and enter the following command in your OS shell:

git clone https://github.com/PauloCarvalhoRJ/gammaray.git

The previous command creates a directory called GammaRay containing your local clone in it, ready to be tracked by Git.

## Creating a branch

A branch is a particular version of GammaRay code. Before starting making changes, it is necessary to create a branch, as any changes are only merged into the “official” source branch (called master) via pull requests, after the necessary code review. To create a new branch, go to the sources directory then enter:

git branch ticket123\_SolveKrigingDialogBug

Most times, you want to create a fresh branch derived from master. To make sure, enter git branch without arguments to check whether your current branch is master before creating your new branch. Of course, it is possible to create branches from non-master branches. If you are not in master branch, switch to it by issuing git checkout master.

The second parameter is the name given to your branch, normally named after the new enhancement to be added or the bug to be fixed. After that, it is necessary to switch to your branch (called a checkout):

git checkout ticket123\_SolveKrigingDialogBug

With the previous command, Git will keep record of any changes you make to the sources and assign them to the current branch. You can list the branches you checked-out at least once with a git branch without arguments.

## Downloading an existing branch

You can download other branches that may have been uploaded by other users to the remote repository with a git fetch command. For instance, a fellow developer may ask you to test a particular new feature she/he is working on or you wish to continue work on the same branch in another computer.

## Keeping tool-specific files from version control

Before making any changes, if you plan to use an integrated development environment (IDE) like Netbeans, Eclipse, Visual Studio or Qt Creator, it is a good idea to open, configure, build and run the unchanged project inside the IDE. This procedure will create any tool-specific files in your source tree. For instance, Qt Creator adds a file called GammaRay.pro.user and the text editor KWrite saves backup copies of the edited files with the pattern name.txt~ in the source tree that are not source files.

To quickly identify files kept by your tool set, simply enter a git status. Any files listed as untracked were created by your tool. You then need to edit .gitignore, adding each of these files making Git ignore them. After making the changes to .gitignore, issue a git status again to make sure no tool-specific file remains untracked.

Other IDE actions in the future may create non-source files, so pay attention to the git status output looking for possible non-source files created by your development tool. Keep in mind that branches containing non-source files will not be merged into master.

## Committing changes

Before committing changes, enter a

git status

This command may output either or both lists:

* “Changes not staged for commit”: the files listed here are files already belonging to the source file set that you changed.
* “Untracked files”: new files that you created in the source tree.

If your shell/command prompt supports colors, the file paths in those lists are displayed in red, meaning that they need to be staged. To stage, in Git jargon, is to prepare a commit, which is a set of individual changes. You stage the changed or created files by issuing a

git add <path>

command. Entering git add . (notice the period) will stage for commit all currently pending changes. After your changes are staged, you need to commit them to the current branch with:

git commit -m “if to prevent a divide-by-zero”

That command groups the set of current dangling changes together into a single unit called a commit. Without changes dangling, you are free to switch (checkout) to another branch or send it to the remote repository, for example. When you do a git checkout, Git changes all source files to reflect the source state represented by the target branch, thus it is important to not have uncommitted changes, otherwise you risk losing progress. Alternatively, dangling changes can be set aside for a later commit with a git stash so you clear the Git state as if you had committed them. Use git stash list to print the current stack of unfinished changes and git stash pop restores unfinished changes following a stack logic.

IMPORTANT: take care to not commit changes to your local master branch as to not risk losing work. Direct changes to master are not allowed in the remote repository. If you by accident make commits to your local master, then you will need to move them to another branch, a sensitive operation called “cherry picking” in Git jargon. Please refer the Git documentation regarding cherry picking. If you made accidental changes to master that you did not commit, you can issue a git checkout <path> to overwrite a changed file with the original content. For unwanted untracked (new) files, simply remove them using your OS commands. Be careful with recipes you may find in the internet that wipe out all untracked files, they may delete any important non-source files you or your tools keep in the source tree, compromising their functions.

## Uploading a branch

Once you completed the desired improvement to GammaRay, you can send the branch to the remote repository with:

git push origin ticket123\_SolveKrigingDialogBug

The previous command orders Git to send the branch to the original repository (in this case the remote one in GitHub).

## Pull request

After you uploaded a branch, you need go to the GitHub project home (<https://github.com/PauloCarvalhoRJ/gammaray>), log in, go to your branch (<https://github.com/PauloCarvalhoRJ/gammaray/branches>) and start a process called a pull request in GitHub in order to have your changes added to the “official” version of GammaRay (the master branch). This is similar to a peer-review process to publish scientific papers with serious journals. Likewise, the code in your branch will be reviewed and tested with two possible outcomes: acceptance or changes required.

If your branch is accepted, your contribution will become part of GammaRay permanently and be shared with the world, tacitly becoming an author agreeing to the GammaRay license (Section 1.1). Keep in mind that this action is irreversible and invalidates any claims of copyright or patent over the published code. If you want to remain the sole owner of the code produced, please, delete the branch on the remote repository **before** requesting a merge.

**WARNING:** Keep in mind that by publishing code, without further notice, you tacitly declare the code is of your authorship, being the sole responsible for possibly adding copyrighted code without express permission or prior published code without given the due credit. The git blame command can be used to identify individuals who contributed to code on a line-per-line basis. So do be careful before requesting merges.

If you are required to make corrections to your branch, simply make the necessary further changes to your local branch and do a git push origin again when done. To increase the likelihood of acceptance, please observe these guidelines:

* Try to keep your branches as small as possible as to have your contribution accepted in less time. Therefore, avoid fixing more than one bug and/or adding more than one feature in a single branch.
* Qt Designer .ui files: these are common sources of headaches. Avoid working on forms visual that you suspect other people are currently working on, as the conflicts are a pain to resolve because Designer tends to rearrange the XML in them on each save.
* Custom Qt Widgets: Do not place widgets not belonging to Qt or GammaRay (e.g. from Qwt) via Designer. Qt widgets of other libraries should be placed explicitly via C++ code. This practice avoids possible incompatibilities with the Qt Designer of other developers.
* Comment your code plentifully, as this facilitates understanding and shortens review time. Try to use Doxygen (<http://www.doxygen.org/>) comment syntax in the headers so any new API can be automatically documented for other developers.
* Keep code portability in mind. Avoid using hardware-, OS- or compiler-specific constructs in your code. If you do need them, please, try to replace with a corresponding Qt or ISO C++ abstraction where possible, otherwise try to provide the equivalent calls for all main OSes via #if #elif #else #endif pre-processor directives. Insertions in other programming languages (e.g. Assembly, Fortran, etc.) are out-of-question.
* If your changes require a new dependency, make sure it can be freely distributable and available at least to the main end-user OSes (Windows, Linux and MacOS). This applies to both runtime libraries and the SDK (headers and compile time libraries).
* Avoid pushing code that compiles with warnings (see Section 3.8.1 for tolerated compiler warnings). Warnings signal potential problems and are not meant to be ignored. If it is unavoidable, acknowledge it in a comment right above the line that is generating the warning.
* Branches that do not compile are rejected at once, so, please, complete your work before requesting a merge. Code must compile with C++11 compilers, so code in C++14 or later standard will be also rejected.
* Branches containing non-source files will be rejected. IDEs are known to keep such files in the source tree. Please, refer to Section 3.5 on how to prevent this.
* Branches are not limited to solving bugs or introducing new features. Commits made entirely by code comments, aesthetic improvements, performance tuning or refactoring are quite welcome.

After your branch is accepted, you can optionally delete your local copy of the branch with a:

git branch -d ticket123\_SolveKrigingDialogBug

### Tolerated compiler warnings

* dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing] when caused by calls to Util::almostEqual2sComplement().
* Warnings caused by dependency code (not belonging to GammaRay code).

## Updating your local repository

As you work, you may want to update your local copy of GammaRay code with the most recent changes. To do so, first switch to the master branch:

git checkout master

Then update your local master branch with:

git pull

If wish to update a branch of yours as well, first switch to the branch with a git checkout, then do:

git merge master

Keeping your local branches updated reduces the likelihood of code conflicts, which are a nuisance to solve.

## File operations

If you need to add new files, rename, move or delete files, it is advised to do via Git commands instead of native OS commands so these operations are accurately reproduced in other repositories.

git add: registers a newly created file with the version control (this command also stages dangling changes for commit if the file already exists).

git mv: performs a file renaming or moving, keeping track of it.

git rm: performs a file deletion, keeping track of it. WARNING: files deleted this way are still kept in history, so make sure to not keep sensitive information in version-controlled files.

Git treats the path as part of the file name, so it does not actually track directories. Hence, to add a directory to version control, you need to create and add at least one file in it.

After performing the necessary file operations, you can do a git commit to register such changes in your current branch.

You can keep files in the repository from version control by editing a file called .gitignore in your repository. This file itself is versioned, so keep in mind that it will be shared and may have rules set by other people.

## Issues using Git

PROBLEM: Switching to another branch (git checkout) hangs, I cancel (e.g. with CTRL+C), then when I try again, Git outputs an error message:

Another git process seems to be running in this repository, e.g.

an editor opened by 'git commit'. Please make sure all processes

are terminated then try again. If it still fails, a git process

may have crashed in this repository earlier:

remove the file manually to continue.

POSSIBLE CAUSE: Make sure you are not manipulating the repository from elsewhere (e.g. from Qt Creator). If this is not the case, Git misbehaved.

POSSIBLE SOLUTION: If you are sure the repository is not being used by another program, go to the repository directory and delete the .git/index.lock file.

## Issues with GDB (debugging)

PROBLEM: The debugger does not trap crashes caused by the Q\_ASSERT macro (e.g. array index out of range).

POSSIBLE CAUSE: GDB is not configured to stop with the exception in qFatal().

SOLUTION: enable the capture of qFatal exceptions. Please refer to GDB documentation or search the internet on how to do that in your tool set. If you are using Qt Creator, you can go to the menu Tools 🡪 Options 🡪 Debbuger 🡪 GDB Extended pane. Enable the “Stop when qFatal() is called” option.

# Getting GammaRay

## Installing (Windows users)

ZIP files with precompiled executables and libraries can be downloaded from <https://github.com/PauloCarvalhoRJ/gammaray/releases>. Simply unpack the ZIPs into any directory and run GammaRay.exe.

## Compiling (developers and non-Windows users)

Given the vast array of possible operating system options available, if you are a non-Windows user, you have to find precompiled executables/installers for your OS or download the source code ZIP or tarball (also found in <https://github.com/PauloCarvalhoRJ/gammaray/releases>) and compile GammaRay yourself, which I personally recommend.

Compiling GammaRay is not difficult, but this is not advised for a novice user should problems arise. Anyone who have compiled programs before will find compiling GammaRay easy to follow. GammaRay code was written to be cross platform, but some few possibly not-so-portable code may cause compiler errors in your platform. If you suspect this is the case, please, report it in the GammaRay development page (<https://github.com/PauloCarvalhoRJ/gammaray/issues>) to have it fixed.

The following instructions assume you are in a POSIX operating system (Unix, Linux, Apple OS X, BSD, etc.) and using GCC tools or using a GCC-like toolset/environment under Windows (MinGW, Cygwin, etc.), so unless you are in a rather exotic OS you may be able to complete the steps giving or taking small variations. Android and iOS are both POSIX, but you may find differences in those platforms that may require some research from your part. Now, if you are using something like OS-2 or Netware do not even try.

Even though GammaRay depends on GhostScript and GSLib to perform tasks, the program does not use them as libraries. You can compile the program without them in your system.

### Configuring

Before compiling GammaRay, it is advisable to keep (though you can change them to different paths) the following variables defined in Gammaray.pro file so the sources directory stays clean:

release:DESTDIR = ../GammaRay\_release/dest

release:OBJECTS\_DIR = ../GammaRay\_release/obj

release:MOC\_DIR = ../GammaRay\_release/moc

release:RCC\_DIR = ../GammaRay\_release/rcc

release:UI\_DIR = ../GammaRay\_release/ui

debug:DESTDIR = ../GammaRay\_debug/dest

debug:OBJECTS\_DIR = ../GammaRay\_debug/obj

debug:MOC\_DIR = ../GammaRay\_debug/moc

debug:RCC\_DIR = ../GammaRay\_debug/rcc

debug:UI\_DIR = ../GammaRay\_debug/ui

The settings above will redirect all meta-object compiler (moc), resource compiler, preprocessor, compiler and linker outputs to a separate directory called “GammaRay\_relase” or “GammaRay\_debug” in the parent directory where the source code directory is. This way you avoid cluttering the source code directory and organize the compilation products. Those directories will be created automatically if they do not exist.

The directories marked with the release: prefix are the ones that will receive the build products for the executable intended for distribution to end users. The ones marked with the debug: prefix receive build products intended for debugging purposes, which are mainly used by software developers.

You also need to define the BOOST\_INCLUDE, QWT\_INCLUDE, QWT\_LIB, VTK\_INCLUDE and VTK\_LIB environment variables with the paths to the directories containing the headers and libraries of Boost, Qwt and VTK (Qt’s headers and libraries are automatically resolved by Qt building system). The environment variable VTK\_VERSION\_SUFFIX must also be defined if your VTK libraries names are suffixed with the version of VTK (e.g. VTK\_VERSION\_SUFFIX=-7.1).

The standard GammaRay.pro contains the QMAKE\_CXXFLAGS += -m64 line. If you want to build a 32-bit program, just delete it or comment it out by prepending a # character.

### Building

After making the necessary changes to Gammaray.pro, you can start compiling GammaRay. Open a command prompt window or shell, go to the directory where the source code was copied and enter the command:

qmake GammaRay.pro

The previous command is expected to generate the Makefile file in your source code directory, readying the project to be compiled using your platform environment. Some tool sets may generate the Makefile in a different directory (the Qt-MinGW bundle is known to do this), hence you need to find Makefile and go to its directory. You can then simply enter:

make

The previous command reads the Makefile script and starts the compilation process that is expected to take 2-3 minutes to complete. The GammaRay executable (called GammaRay in POSIX systems or GammaRay.exe in Windows) will be left in one of the output file directories, depending on the changes you made to GammaRay.pro. If you defined the DESTDIR directory, then the executable can be found there.

Recall that the mentioned commands are not part of the operating system. qmake is part of Qt, so it will not work if Qt is not installed. make is part of a GNU developer toolset and will not work if the said package is not installed.

### Deploying

If you want to run GammaRay only in the computers where it was compiled and if the Qt libraries were made globally available (environment variables PATH (Windows), LD\_LIBRARY\_PATH (Linux), etc.) in those systems, then one can start GammaRay by simply running the executable.

If you plan to distribute the recently compiled GammaRay to end users, you must copy the Qt libraries along the executable and possibly make changes to the aforementioned environment variables. Please, refer to how to deploy Qt programs here: <http://doc.qt.io/qt-5/deployment.html> or here: <http://doc.qt.io/qt-5/windows-deployment.html> . The latest Windows versions of Qt come with a tool called windeployqt.exe that helps in creating a directory containing the executable and all the DLL dependencies for easy Windows deployment.

The compiler you used likely introduces particular library dependencies that also need to be shipped along the application executable. Please, refer to your compiler documentation regarding the libraries that must be distributed with the executable.

In case you need to make executables for a broad audience such as in a large company, university or the internet, you may consider making installers that automate the deployment process. Inno Setup and Nullsoft’s NSIS are good examples to make Windows installers. For Linux systems, it depends on the specific distribution. For example, RedHat-like distributions use the RPM installation system and Debian-like OSes use the DEB system.

If you get error messages trying to run GammaRay regarding undefined symbols and/or missing libraries, you may consider using the ldd command for POSIX systems or the Dependency Walker tool (<http://www.dependencywalker.com/>) for Windows to analyze the problem. Missing symbols with all the required libraries already present in the distribution directory may signal that you copied the wrong versions of libraries. For instance, if you are compiling GammaRay with MinGW 32-bit under Windows, you must copy the Qt libraries from <QtInstallDir>\<QtVersion>\mingw<mingwVersion>\_32\bin and not those found in <QtInstallDir>\<QtVersion>\winrt\_x64\bin for instance.

# Getting started

The first time you run GammaRay, its default main screen (**Figure 2**) will appear. It is a very simple interface and tasks are carried out by evoking context menus on the various possible combinations of object selection in the project tree (**Figure 2**, A). For instance, to plot a scatter plot, select two or three variables of the same data file, right-click on them and click on the appropriate option.

From now on, the term “right-click” refers to context menu calling. The actual gesture to trigger this action varies depending on OS, window manager, user preferences, available input devices and accessibility features.



C

B

A

**Figure 2** GammaRay main window. A: project tree; B: content panel; C: output messages panel.

(**Figure 2**, B) is the content area, where more complex interfaces like the Workflow Designer (see section) and the 3D Viewer are docked. (**Figure 2**, C) is the output panel where GSLib and GammaRay output messages are displayed. Informative messages are displayed in blue, warning messages, in dull yellow and error messages/output, in red.

## Configuring GammaRay

Before commencing work, it is necessary to tell GammaRay where to find the GSLib executables and the GhostScript PostScript renderer. Go to File->Settings… menu to open the settings dialog (**Figure 3**), which consists of two fields containing directories. For the GSLib directory, simply locate where the GSLib executables are.



**Figure 3** the program settings dialog.

For the GhostScript, you need to select its home directory. The GhostScript home directory is normally its installation directory, which at least contains a bin directory with the GhostScript executable (gswin\*.exe in Windows, gs in other OSes) in it. Sometimes, GhostScript is a system-wide tool in Unix-like OSes, so bin/gs can be found in /usr. When done, click OK to commit the changes.

In the setting dialog, you can also set the maximum number of cell grids for viewing grids in the 3D Viewer (Section 5.9). The default is 2 million cells, which is safe for a mildly dimensioned system (4GB of RAM, mid standard graphics card). You can adjust it down to 500 000 and up to 100 million. If a grid exceeds this threshold, the program subsamples it until the cell count falls below this setting to allow a safe visualization. Subsampling occurs only for visualization purposes, as the data are not changed.

## Creating a project

The project is how the files of your study are organized. Physically, a project is just a directory with a text file called gammaray.prj in it. Creating a project is very simple, go to the menu File -> New/Open Project, then select a directory. GammaRay then creates a gammaray.prj file in the selected directory along with a few subdirectories.

The project is shown in GammaRay with some top-level items visible in the project tree (**Figure 4**). The project name will be the directory name in file system, so if you wish a meaningful name for you project, just choose/create a directory named as desired using operating system functions.



**Figure 4** the project tree top-level items.

GammaRay does not touch any files that may be already present in the project directory, so you can simply point to a directory containing all your data files. If you want to use a new directory, you must first create it via OS commands/interface.

There is no need to issue a “save project”, as the project is updated automatically. Closing the project (menu File->Close Project) is also not necessary so the program automatically re-opens the project that was open when the software was closed for the last time. GammaRay also saves the project tree state so the expanded or collapsed items are shown exactly as they were when the program was closed. GammaRay also keeps a list of recently opened projects in the File menu, so, to re-open a previously opened project, click on one of the names.

The user can quickly inspect the project directory by right-clicking on the project name label, on the top of the project tree, then selecting “Open project directory…”. Depending on the underlying window manager, some actions can be performed in the dialog such as opening a file in an external editor. You can also quickly assess the project directory path by selecting “See project path” in the same context menu.

In addition to the gammaray.prj file, GammaRay creates two sub-directories: templates that contains parameter file templates for each GSLib file and tmp, which contains intermediary files generated during GammaRay execution.

IMPORTANT: GammaRay does not clear the tmp directory so the user can review intermediary results and possibly save them elsewhere. It is up to the user to remove the temporary files either externally or by right-clicking on the project name label and choosing “Clear temporary files”. Cleansing via menu command presents a confirmation dialog with the current size of the temporary files directory in megabytes.

## Adding data files to your project

GammaRay supports data files in the format compatible with GSLib, that is, a simplified form of Geo-EAS format. To import data files to your project, right-click on the Data Files item in the project tree, then click on “Add data file…” option. An OS-dependent file selection dialog (Windows Explorer, Konqueror, Dolphin, Commander, etc.) will appear enabling you to browse the file system directories and select a data file. Even for data files already present in the project directory, this action is required. You can also drag and drop a file onto the program’s main window to add it as a data file to the project. Support for drag and drop files depend on your operating system and may be unavailable or not work as expected.

After a file is selected (or dropped), a second dialog (**Figure 5**) is presented, where the user can review a file sample and declares which type the data file is (regular grid or point set). Depending on the data type informed, a third dialog (**Figure 6** or **Figure 7**) appears so the user fills in the data file details. It is important to provide accurate information, so the automation provided by GammaRay works properly.



**Figure 5** data file type declaration dialog.



**Figure 6** point set file information dialog.



**Figure 7** regular grid file information dialog.

Pay special attention to the no-data value. This value signals to GammaRay which variable value corresponds to uninformed data, so it can accurately configure GSLib parameters to treat them so. The no-data value can be changed anytime later by right-clicking on a data file, then choose the “Set no-data value” option. To unset the value, leave the field empty.

The added data file appears as a sub-item in the Data Files item in the project tree. To see all your data files, simply expand the Data Files item clicking on the plus sign to the left. Files of different types are displayed with different icons. **Figure 8** shows an example of project with some data files. The files can be already inside the project directory and files outside the project directory are automatically copied into it before being added.



**Figure 8** example of a project tree with two data files.

The variables within each data file also appear in the project tree as child items. A number between square brackets appears to the left of the variable name to indicate its sequence within the file, as the variables may appear alphabetically sorted or arranged in more complex structures. This is the same number used to refer to variables in GSLib parameter files.

An accessory file containing information about the added file is automatically created and updated by GammaRay in the project directory, this is the metadata file. The metadata file, a human-readable text file, has the same name of the file it is referring to, but with the added .md extension. You can see the metadata file contents by right-clicking on a data file in the project tree, then choosing “See metadata”. No-data-value

## Removing data files from the project

Right-click on a data file, then choose either “Remove from project” or “Remove and delete”. The first option simply removes the file reference from the project so it does not appear in the project tree anymore, but the physical file itself is kept in the project directory. The second option also removes the file from the file system, so be careful, as this action is irreversible. BUG NOT DELETING MD FILES

## Freeing up memory by freeing loaded data

As you work, GammaRay keeps loaded data in memory so they do not need to be loaded again (unless the file changes in disk). This is especially useful for larger data files, when repeated time-consuming data reloading can become a nuisance. However, they may accumulate with time, so you may wish to free up RAM for GSLib or other applications by unloading any loaded data.

To do so, simply right-click on the project name label, on the top of the project tree, then select “Free loaded data (frees up RAM)”. Depending on the amount of data loaded, you may not notice any change.

## Conventions

As GammaRay is built on top of GSLib, it follows the GSLib conventions regarding geostatistical modeling.

**Angles:** azimuthal convention, also known as geologist’s convention. 0° corresponds to North and angle values increase clockwise. Azimuths are noted as N###E, indicating this convention. For example, N090E is 90° following the azimuthal convention, which corresponds to East.

**Value-grid alignment**: regular grids are always cell centered. This means that a property value is constant inside a given cell, instead of property values assigned to grid vertexes (corner-point grids). **Figure 9** illustrates the relation between the data value locations and the grid geometry per GSLib gridding convention.



**Figure 9** GSLib regular grid convention for a 3x3 2D regular grid.

**Grid origin:** The first cell in the Cartesian grid data file is the westernmost, southernmost, topmost cell. The origin coordinate is the center of the first cell.

**Grid scan:** Cell order in data files follows first the East->West order. Once a row is completed, then South->North order is followed to the next row. Once all rows are scanned, then Up->Down order is followed on to the next slice. If the file has more than one realization, then order goes on to the next realization, starting over to the position of the first cell and so on.

## Plot Dialog

GammaRay code was designed with code reuse in mind, so the same dialogs will be used across the different program functions. This is not only a software engineering decision; it also means that the user will find familiar operation procedures across the different program functions.

**Figure 10** shows the Plot Dialog containing a histogram. The Plot Dialog is presented whenever a graphical output needs to be displayed. Its features are available to all the different GSLib plot programs (histplt, pixelplt, scatplt, etc.). The dialog’s buttons are explained in **Figure 10**.

The Plot Dialog allows the user to display plots in a desired resolution in dpi units (dots per inch). For convenience, the user can click on one of the standard resolution buttons (**Figure 10**, B) to set current plot resolution to 80dpi (for viewing on the screen), 150dpi (recommended for on-line publication), 300dpi (recommended for printed black-and-white publication) and 600dpi (recommended for printed color publication).



A

B

C

D

E

F

**Figure 10** the Plot Dialog showing a histogram. A: buttons to increase/decrease plot resolution in 10dpi steps. B: buttons to set plot resolution to standard resolutions in dpi. C: button to capture current image to clipboard. D: button to toggle crosshairs under the mouse pointer. E: button to call the Parameters Dialog to review and change plot settings. F: button to save the plot file (PostScript) in the project.

## Parameters Dialog

The Parameters Dialog (**Figure 11**) is presented whenever the user wants to change the parameters of some GSLib program. The fields in this dialog are dynamically created from the definitions of a parameter template file (more about parameter templates in Section 13.2) located in the templates subdirectory in the project directory. There is one template for each GSLib program, each resulting in a different dialog configuration.

GammaRay fills all the fields so usually good starting results can be obtained by changing a few or no parameters at all. This enables a smooth and faster assessment of data, specially during the exploratory data analysis. Explaining each parameter of each GSLib program is out of the scope of this document. To get introduced to parameters meaning, REF and <http://www.statios.com/> are good references.



**Figure 11** the Parameters Dialog configured for the pixelplt program.

## 3D Viewer

The 3D Viewer (**Figure 12**) is a widget (GUI element) that by default occupies the content panel (**Figure 2, B**) in the program’s main window. It complements the standard GSLib plots as it allows a quick visualization and easy navigation through the whole data set in all three spatial dimensions.

To use the 3D Viewer, simply drag-and-drop any object from the project tree into the object list, located to the left of the 3D Viewer. Drag the mouse over the viewing panel to manipulate the scene. Try different mouse buttons and combine them with keyboard mode keys (ALT, SHIFT, CTRL) to achieve different sets of manipulation movements. If your pointing device (e.g. mouse) features a scroll wheel or similarly purposed input, it can be used to zoom in and out.

Trying to view currently unsupported object types will cause an error message to appear in the output message panel of the main window. Object type support will improve in future versions of the program. To remove objects from the scene, right-click on an object in the 3D Viewer’s list, then choose “Remove from view” in the context menu. To the right of the viewer, a set of buttons can be used to issue special scene changes such as camera reset to global view our repositioning the camera to face one of the Cartesian planes.

The standard VTK key commands are available: pressing ‘W’ switches rendering to wireframe mode (‘S’ switches back to surface); ‘P’ with the mouse pointer over an object shows the object’s bounding box (hit ‘P’ over empty space to hide it).



**Figure 12** Main window featuring the 3D Viewer showing an attribute of a point set.

### Known issues

Problem: displaying a large grid causes the program to crash. This crash may also ensue with critically large grids when switching the display mode to wireframe (pressing ‘W’ key), which is more memory demanding.

Possible cause: low system requirements. Launch the program from a command prompt/shell and test, looking for a message containing std::bad\_alloc. This message indicates the program was unable to allocate memory necessary to carry out visualization.

Solutions: Close other applications and/or unnecessary OS services; free loaded data (Section 5.5); remove other objects from visualization; reduce the maximum grid cell count for safe visualization (Section 5.1) to enable subsampling of the offending grid.

## Known issues trying to launch the program

PROBLEM: GammaRay may turn impossible to launch (normally with an OS error message dialog or SEGFAULT message in shell) after it had crashed following an error with the bivplt program (see Section 6.7). This may arise with other programs. Apparently, the project becomes corrupted after the first crash, and when GammaRay tries to open the project again upon launching, it crashes again, becoming useless.

POSSIBLE SOLUTION: Try to rename the directory containing the offending project and launch GammaRay again.

# Exploratory data analysis

The functions described in this section are related to the first step in the geostatistical modeling study: data analysis.

## Map / data display

Right-click on a variable, then choose “Map” to display the map plot. For regular grids (pixelplt program – **Figure 14**), the user can change the plot parameters to display slices along the XZ and YZ planes, differently from the default XY plane. A similar effect can be achieved for point sets (locmap program – **Figure 13**) if the user manually changes the default x, y, z in the parameter dialog (click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button), for example, assigning the z variable as y coordinate results in a sideways view to the samples instead of a geographic map.

Categorical variables (C:\Users\paulocarvalho\Desktop\GammaRay\art\catvar16.png icon) in Cartesian grids are displayed as such, using the associated category definition to set appropriate plot parameters. Variables containing categorical codes but not set as one (C:\Users\paulocarvalho\Desktop\GammaRay\art\variable.png icon) can be displayed as categorical by choosing one of the available category definitions (Section 9.1) after choosing “Map as” in the context menu. Even categorical variables can be displayed with a different category definition by means of the “Map as” sub-menu item. Due to limitation of the locmap progam, categorical variables in point sets can only be displayed as continuous.



**Figure 13** map plot of a point set file.



**Figure 14** plot of a XY slice (map) of a regular grid.

### Known issues

Problem: Plotting a Cartesian grid is blank in the Plot Dialog.

Possible cause: Large data set. Certain versions of pixelplt cannot open grids with too many cells. Check the output messages panel (**Figure 2**, C) for a “MAXV too small” message, which is sent by pixelplt as a normal message (blue characters) instead of an error (red characters).

Solutions: a) Find a pixelplt executable that works by trial and error. b) Recompile pixelplt from source, after changing the MAXV variable to a greater value. c) View the grid in the 3D Viewer (Section 5.9).

## Histogram

Right-click on a variable, then choose “Histogram”. After calling the GSLib program histplt with an appropriately generated parameter file and GhostScript, GammaRay presents the Plot Dialog displaying the histogram (**Figure 10**). If you wish to change the suggested histplt parameters, simply click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to open the Parameters Dialog, make the necessary adjustments, then click OK to run histplt again and the plot is updated. For instance, it may be necessary to change the horizontal scale to logarithmic for gold grade or permeability data or you may wish to plot the cumulative histogram instead of the default density histogram.

### Known issues

Problem: Histogram plot becomes blank when changing to logarithmic scale.

Possible cause: There was an attempt to compute log(0) in the GSLib program.

Solution: Try setting trimming limits different from those set automatically by GammaRay: for example, range not containing zero.

Problem: Histogram computation takes forever. You can use Task Manager (Windows) or top/htop commands (Posix OSes) to see whether the histplt process keeps at high CPU usage.

Possible cause: If the data file is a large Cartesian grid, its sheer size may cause histplt to misbehave.

Solution: Use the OS tools to abort the histplt process, then try resampling the grid (Section 7.6).

## Crossplot

Select two or three variables of a same data file, right-click, then choose “Cross plot Var1 X Var2” or “Cross plot Var1 X Var2 X Var3” option to display the cross plot (**Figure 15**). Notice that order is important. The variable selected first will be the X-axis, the second will be the Y-axis, and the third, if present, will set the gray scale color of the cross plot points.



**Figure 15** a 3-variable crossplot.

## Probability plot

A probability plot is used to check a variable for normality or log-normality and to assess the behavior of the extreme values. The probability plot is a special type of crossplot with the values in linear or logarithmic scale in the x-axis while the normal c.d.f. is in the y-axis. Thus, normal or log-normal data appear as a near-straight line covering the entire plot intervals in a probability plot. The GSLib program probplt is used to compute this plot.

To display the probability plot (**Figure 16**), simply right-click on a variable, then choose the “Probability plot” option of the context menu to run probplt.



**Figure 16** an example of the probability plot for a logarithmic variable. The vertical axis is the normal c.d.f. values.

## Q-Q and P-P plots

These two plots are used to compare the distributions of two different data files. The Q-Q plot compares the quantiles whereas the P-P plot serves to compare the cumulative probabilities. To display the Q-Q plot (**Figure 17**), select two variables of different files, right-click, then choose “Q-Q/P-P plot”. Both variables must represent the same random function, otherwise the plots do not make sense. If you want to display the P-P plot (**Figure 18**), click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button after the Plot Dialog appears and change the parameters to do so. The program qpplt is used to generate both plots.



**Figure 17** a Q-Q plot of a logarithmic variable in two different data files showing similar distributions.



**Figure 18** the P-P plot of the same data of Figure 17

## Univariate distribution modeling

The histsmth program allows the user to fit a non-parametric smooth distribution model to a histogram, which can be especially useful for simulations (Section 12) where the available data do not allow a clear distribution characterization. To do this, the user right-clicks on a variable and selects the “Model a distribution…” option to open the distribution modeling dialog (**Figure 19**).



**Figure 19** the smooth distribution modeling dialog.

To start modeling, the user clicks on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the suggested histsmth parameters and then click on OK in the Parameters Dialog to run the program and display the result (**Figure 20**). You can click on the  button to display the last result again as needed. The histsmth program is also used to plot the result, so you can perform the fitting cycles in the Plot Dialog. When done, click on the  button to save the smooth distribution model in the project.

Before the file is saved, a dialog (**Figure 21**) is presented to set roles for each column of the distribution file, as the contents of file vary. It is possible to cancel this dialog, but doing so hinders GammaRay automation and the lack of visual differentiation in the project tree can cause confusion.



**Figure 20** plot of the histsmth result showing a smooth distribution (solid line) fit to a histogram (bars).



**Figure 21** the dialog used to set the roles for the distribution file columns.

The saved univariate distributions appear under the Distribution Files group in the project tree.

### Known issues

Problem: Univariate distribution modeling fails when changing to logarithmic scale.

Possible cause: There was an attempt to compute log(0) in the GSLib program.

Solution: Try setting the smoothing limits to values such that log(0) does not occur.

## Bivariate distribution modeling

The scatsmth program lets the user model a non-parametric smooth bivariate distribution in the same fashion as the univariate distribution modeling. bivplt is the separate program able to prepare the complex plot combining scatsmth and histsmth results. Bivariate distributions are typically used in co-simulation methods (REF). To start bivariate distribution modeling, select two variables from the same file, right-click and choose “Model bidistribution <name of first variable> X <name of second variable>” to open the bivariate distribution modeling dialog (**Figure 22**).

Notice that it is necessary to have univariate smooth distributions for both variables in your project beforehand. Please refer to Section 6.6 for univariate smooth distributions.

**Figure 23** shows an example of bivariate distribution fit to data.



**Figure 22** the bivariate smooth distribution modeling dialog.



**Figure 23** a smooth bivariate distribution fit to data and their smooth distributions.

**Figure 24** shows an example of a complete set of smooth distributions generated for two variables.



**Figure 24** part of a project tree showing the final result of smooth distributions fit to two variables.

### Known issues

Problem: after scatsmth execution, the plot appears blank or is not updated when changes are made.

Possible cause 1: incorrect column index parameters when one or both variables are in logarithmic scale.

Solution1 : in the plot window, open de parameters dialog (C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button) and change the column indexes for the bidistribution file. The defaults are 1, 2 and 3. The GEO-EAS file column that carries the probability values normally is the last one, so the correct index is likely to be 4 or 5, depending whether one or both variables are in logarithmic scale.

Possible cause 2: there was an attempt to compute log(0) in the GSLib program.

Solution 2: set the trimming limits to values such that a log(0) is not computed.

Possible cause 3: inadequate color scale.

Solution 3: set the color scale limits such that all probability values lie within the range. Try starting with 0.0 and 1.0.

## Histograms of realizations

Recall that this feature requires the histpltsim program, which is non-standard GSLib, so make sure the directory containing the GSLib programs has it, otherwise an error message will appear in the message panel. After a simulation run, a common practice is to validate the realizations. One validation is to plot the histograms of the realization ensemble in the same chart and often compare them with a reference histogram.

In GammaRay this can be simply done by right-clicking on a variable of a Cartesian grid then activating the “Realizations histograms” option in the context menu. A reference histogram can be plotted if a variable of a point set (supposedly the hard data used to condition the simulation) is also selected along the Cartesian grid containing the realizations before evoking the context menu. **Figure 25** shows an example of histograms of realizations plotted with a reference histogram.



**Figure 25** Histograms of a realization ensemble (black) plotted along with a reference histogram (red).

## Known issues with plots (\*plt programs in general)

PROBLEM: The Plot Dialog does not show a plot (gray panel in Windows) or an incomplete plot, even with \*plt parameters checked and the message panel shows that \*plt completed without error messages or interruptions in its execution.

POSSIBLE CAUSE: User-entered strings (e.g. plot title, legend, etc.) happen to contain PostScript symbols or variable names (e.g. variograms: when plotting with vargplt) that corrupt rendering by GhostScript, causing it to fail silently or delivering an image with parts missing.

SOLUTION: Check your custom plot texts for suspicious words (e.g. line) and characters (e.g. parenthesis) that might be used as symbols and instructions in PostScript files. You can open a PostScript file in a text editor to have an idea of what symbols and reserved words appear in them.

# Data preparation

Data preparation functions modify or transform the original data so they become compatible with some modeling algorithm, de-biased, etc. Other functions presented in this section are operations to data files that may help the user during the modeling workflow.

## Declustering

Declustering is a very important step in any geostatistical study because irregular sampling may distort its statistics. Declustering is a method to mitigate location bias from irregular data. For example, oil wells tend to be drilled at the best production targets, therefore porosity data, for example, may have greater mean than reality.

To start declustering, right-click on a variable in a point set file, then choose “Decluster…” to open the Declustering Dialog (**Figure 26**).



**Figure 26** the Declustering dialog.

Click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the suggested parameters then click OK in the Parameters Dialog to run the declus program. Evaluate the result by displaying the declus report (button), the resulting statistics ( button) or the map of computed weights ( button). If not satisfied or whish to search in a narrower window, repeat the steps, otherwise click save ( button) to add the computed weights to the point set file. The weights become a new variable in the selected point set file. The declustering weights computed this way are represented in the project tree as a child item under the original variable (**Figure 27**) to emphasize the dependency relation between the variable and its weight.



**Figure 27** a project tree showing two variables with declustering weights attached (the variables represented by the "W" icon).

The declustered weights shown as attached to their variables are automatically assigned for GSLib programs that use them. If you have declustering weights shown in the project tree as independent variable, then you have to manually configure the corresponding parameter. If you have more than one weight for a same variable, then you must also manually change the parameter if you wish to use a different weight than the default.

### Maximum or minimum?

Sometimes you have, for example, ore grade samples and you may assume samples are biased towards high values, thus you configure declus to search for the lowest average just to find an inconsistent declustering weights map like **Figure 28**. It is inconsistent because only part of the clustered samples are getting weights departing from 1.0. This signals that you must probably switch optimization to the opposite way.



**Figure 28** Map of inconsistent declustering weights.

## Transfer collocated values

Values from a regular grid can be transferred to the collocated (that is, same place) locations of a point set file using the getpoints program. This can be useful to plot cross plots between estimated or simulated values and the original sample values for validation. You can perform this task by simply selecting a point set file and a Cartesian grid file, right-click and select the “Transfer colocated values from … to …” option. This action is fully automated so it does not open the Parameters Dialog. All variables from the regular grid are transferred, as getpoints does not allow a selection.

## Creating a grid for estimation or simulation

GammaRay has a tool to quickly create a grid appropriate for estimating or simulating a variable. To do so, right-click on a point set file, then choose “Create estimation/simulation grid…” to open the grid creation dialog (**Figure 29**). The program suggests initial grid parameters based on the area/volume occupied by the data. Normally the user sets round values for the grid resolution (xsiz, ysiz, zsiz) and origin (xmn, ymn, zmn). The “nx,ny,nz = “ button can be used to compute new cell count (nx, ny, nz) based on the values entered and the space occupied by data. The user can preview the grid geometry by clicking on the “Preview” button, for which GammaRay generates and plots a temporary regular grid containing a binary variable with a checkerboard pattern.



**Figure 29** the dialog used to create grids for estimations or simulations.

The user can check the amount of variance loss with the current grid resolution with respect to a variogram model and an intended block discretization for block kriging by running gammabar ( button). Leave block discretization the default 1 x 1 x 1 for point kriging. If the loss is much greater than the nugget effect, then the user might consider increasing the grid resolution. Conversely, if the loss is too close or equal to the nugget effect, the grid resolution might be higher than necessary.

## Convert a grid to point set

Sometimes it is necessary to convert a regular grid into a point set file. For example, changing datum/projection (e.g. WGS-84/UTM) requires a re-projection of each grid value location separately and then a new regular grid (see Section 7.3) must be defined in the new coordinate system. Another example: the indicator kriging program, ik3d, accepts secondary data (soft indicators) as point set files instead of the usual regular grid.

This conversion is done by adding coordinates (addcoord program) to the values arranged in a grid. You can do this by right-clicking on a Cartesian grid, then select the “Convert to point set” option. The user then enters the realization number (default is 1) and the name for the new point set file. After that, addcoord runs and a new point set file is added to the project.

## Look for duplicate samples or samples too close

Duplicate or too close sample points may cause some GSLib programs to fail. kt3d, a kriging program, is known to output grids with all-non-data-value estimates due to numerical instability or divisions by zero caused by small separation between samples. Data processing, re-sampling, or file mergers are common causes of duplicate samples. To do this, right-click on a point set file, then choose “Look for duplicate/close samples”.

The program will request you to enter two values: tolerance and distance. First, you enter the tolerance. Tolerance is the size of a bounding box around each data point used to arrange them in a data structure optimized for spatial searches. The default is a good number, so normally you just confirm. Larger values may degrade the performance of the spatial searches. Then you enter the distance. Distance is the criterion used to tell two samples are too close to each other. Enter a small value greater than zero, as entering zero may prevent you to locate duplicate samples due to floating point number inaccuracies. The default distance value is a small enough figure assuming a typical mining or petroleum application. If your study involves a spatially small domain (e.g. biology), you may consider entering a smaller value.

After entering tolerance and distance, the program will present a report in the main window’s message output panel (**Figure 2**, C) showing the file lines where duplicate or near-miss samples were found. You then have to edit the data file in another program, deciding how to solve the reported issues. The program offers a quick way to edit a project file: right-click on a file, choose “Open with external program”. This obviously depends on your OS, system settings and programs installed.

### Known issues

The program may crash due to an internal error in Boost’s R-Tree spatial index used in GammaRay to look for duplicate data points. An R-Tree is a data structure used for fast spatial searches, which is built depending on several parameters set in GammaRay code. This crash is known to occur with certain data files, requiring adjustment of R-Tree parameters in the program’s code. Please, contact the program developers (contact info can be seen in menu Help->About or in the program’s page in GitHub), providing the data file that caused the crash so the R-Tree parameters can be fine tuned.

## Resampling large grids

Sometimes GSLib programs are unable to handle large data files (e.g. 10’s of millions of points/cells). Even GammaRay itself can crash with such large grids depending on memory constraints of the system. If you experience crashes or extremely slow response from either GammaRay, GSLib programs or GhostScript with a large grid, you may wish to resample it. Sometimes an external program outputs a large grid and the user wishes to validate the results using GammaRay/GSLib (e.g. plotting histograms and variograms for the obtained models), so, a smaller, resampled model, is enough to validate the original result. Other possible application for resampled grids are using them as proxy models.

To perform resampling, simply right-click on a Cartesian grid, then select the “Resample” option. A small dialog will pop-up asking the user to enter the sampling rates for each of the topological directions (I/X, J/Y or K/Z). If you leave all as 1, the grid will be simply copied. If you set 2 for J, for instance, the program will create a new grid, taking a sample in the J (Y-aligned) direction every 2 cells, resulting in a file approximately half the size. The program also adjusts cell geometry so as the resulting model occupies the same spatial extent of the original data. You must find by trial-and-error sampling rates small enough as to allow the GSLib programs to handle the data, avoiding large values that may distort, for example, histograms and variograms.

IMPORTANT: Keep in mind that resampling is not the same as upscaling. Resampling is a simple computational procedure to obtain a smaller data set by skipping cells. Resampling is fast though less accurate than upscaling. Resampled grids can be used to get histograms, variograms and for proxy modeling. Nevertheless, for economic studies, resource estimation, etc. with smaller models, upscaling or partitioning are preferred methods.

# Data transforms

Sometimes it is necessary to transform the data to bring it within certain assumptions (e.g. normality) or to change its domain (e.g. to frequency domain) to perform filtering.

## Normal Score Transform

The normal score transform (nscore) transforms a variable so its distribution becomes a Gaussian distribution with zero mean and unitary standard deviation. Some GSLib modeling programs can automatically n-score the input data as needed, but sometimes it is necessary to have more control over the transformations in the workflow or some operation (e.g. decorrelation) must be performed in normal space prior to the estimation/simulation step.

To do this transform, simply right-click on a variable, then choose “Normal score…” to open the N-Score dialog (**Figure 30**). Click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the suggested parameters and then OK in the Parameters Dialog to run nscore.



**Figure 30** the normal score transform dialog.

Check the normal variable statistics by clicking on the  button. The  button is used to add the normal variable to the data file and to save the resulting transform table in the project directory. Normal variables computed from inside GammaRay appear in the project tree as a child item of the transformed variable ( icon), so the user can easily keep track of the transform stack that may develop as the work progresses. Transform tables are not visible items in the project tree, but GammaRay keeps track of them via the metadata files as to facilitate back transform in the later steps of a study.

## Fourier Transform

The Fourier Transform (FT) changes the data from the spatial domain to the frequency domain. FT is available as, due to the discrete nature of grid computation, a Discrete Fourier Transform (DFT), implemented via a Fast Fourier Transform algorithm. To perform a FT, right-click on a variable of a Cartesian grid (of any dimension), then choose “FFT”. The program will prompt the user to enter the name for the new Cartesian grid containing both the real and imaginary parts (complex numbers) of the data in frequency domain. The resulting grid is topologically equivalent to the input grid. The real part is the amplitude spectrum while the imaginary part is the phase spectrum.

**Figure 31** shows a typical Fourier image obtained with FFT. It can be regarded as a spectrum with frequency zero (DC) is at the center of image, with frequency increasing outwards in all directions.



**Figure 31** An FFT image of a 2D Cartesian grid.

### Known issues with FT

PROBLEM: FT seems to be slow or takes forever for a large grid.

POSSIBLE CAUSE: FT is expected to take about a minute for a 7-million cell grid in a quad-core mid-range notebook computer (plus another minute to save the results to file). GammaRay uses VTK to process images (a grid can be viewed as an image). The program can be loading the debug version of VTK or VTK built with inappropriate compiler options, which is a potential risk with precompiled libraries.

SOLUTION: If you are under Windows, use the libraries packed with the GammaRay executable in its GitHub homepage (see Section 4). If you are building GammaRay, it is recommended to build VTK too (Section 2.3.7). The VTK libraries managed by MSYS2 system, for instance, were observed to cause poor FT performance.

## Reverse Fourier Transform

You can obtain a grid in real domain from a Fourier grid by simply selecting two variables of a Cartesian grid, right-clicking and selecting the “rev. FFT: real = …; imag. = …” option. Notice that variable selection order is important: the first variable will be the real part (amplitude spectrum) and the second will be the imaginary part (phase spectrum).

# Data classification

Some geostatistical methods work with categorical data, which means that the usual continuous measurements must be transformed into category or class integer IDs. For example, one needs to estimate a map of sandstone and shale from gamma ray well logs. Hence, it is necessary to convert the gamma ray records into two codes: one assigned to sandstone and the other assigned to shale. These codes can be 1 and 2, for instance. Noteworthy, this transformation is not reversible, meaning that it is not possible to obtain gamma ray values from sandstone and shale codes deterministically.

Data classification is a vast field of study itself and it is beyond the scope of this document to present the subject with adequate depth. Only the classification methods available in GammaRay will be described herein.

## Category definition

Before any classification takes place, it is necessary to define the categories. In GammaRay this means to create at least one list of triplets in your project: code, color and name. To create a category definition, simply right-click on the “Resource Files” top-level item in the project tree, then select “Create categories definition…”. The category definition dialog (**Figure 32**) will open. Click on the “+” button to add any number of categories, then specify an integer code, a GSLib color and a name for each category. When done, click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\save16.png button to save the definitions to your project.



**Figure 32** The category definitions dialog.

You can review and edit any category definition later by right-clicking on an item with a C:\Users\paulocarvalho\Desktop\GammaRay\art\catdef16x16.png icon and selecting the “Edit” option. You can remove and add categories and change the existing ones.

## Univariate classification

Univariate classification can be simply done by right clicking on a variable of a point set, then selecting “Classify into” or “Classify with” and choosing one of the category definitions or one of the category classifications available (**Figure 33**).



**Figure 33** Single-variable classification context menu.

An editor dialog is shown (**Figure 34**), where the user enters value ranges to be mapped into categories. The histogram (Section 6.2) can help in defining the ranges. Saving the mapping (category classification file) is optional, though it can be useful if you wish to adjust the ranges or to perform the classification procedure with another data file or variable. Clicking OK the classification will take place, after which the user will be asked to enter the new variable name containing the categorical values to be added to the selected file.



**Figure 34** The dialog used to specify ranges of values to be mapped into categories.

## (CONSIDERING) Bivariate classification

## (CONSIDERING) Ternary diagram classification

## (CONSIDERING) Multivariate classification

# Variography

Statistics study random variables. If the random variable varies with spatial location, then it is called a regionalized variable, studied with geostatistics. As related to spatial location, regionalized random variables may exhibit a spatial structure or continuity, such as many natural phenomena such as ore bodies, spreading of pollutants, distribution of temperatures, etc.

A variogram is a model of the spatial continuity, or correlation, of a regionalized variable (direct or autovariogram) or between two regionalized variables (cross variogram). Two variables that possess a shared spatial behavior or correlation (cross variogram) are said to be coregionalized. All variogram-based geostatistical modeling programs require a variogram model so they can yield models that reproduce the spatial behavior observed in data. The usual workflow is to compute an experimental variogram from the input data and the user fits a theoretical variogram model to it. This model is then entered as parameters in variogram-based estimation and simulation programs.

To start variogram modeling, right-click on a variable (or two variables for cross-variography, see Section 10.4), then choose “Variogram analysis…” to open the Variogram Analysis dialog (**Figure 35**).



**Figure 35** the Variogram Analysis dialog.

## Variogram map

A variogram analysis can start by identifying the variable anisotropy after computing the experimental variogram in all directions (variogram map/volume). The variogram map is a regular grid (2D or 3D) that allows quick identification of the main axes of the anisotropy ellipsoid for a future variogram model, hence the user only needs to compute experimental variograms along the axes. In the Variogram Analysis dialog, click on the topmost C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the suggested parameters then click OK in the Parameters dialog to run varmap and display the resulting grid (**Figure 36** and **Figure 37**). Click on the topmost  button to display the grid again as needed.



**Figure 36** a variogram map (XY slice) showing an anisotropy with a semi-major axis along the N165E azimuth.



**Figure 37** a variogram map showing anisotropy with semi-major axis along the N070E azimuth.

The variogram map grid can be saved in the project by clicking on the topmost  button so the user can assess other experimental variogram attributes computed by varmap such as number of pairs.

A “good” variogram map uses lags only long enough as to periodic bands of blank cells or “salt-and-pepper” do not appear in the resulting grid. Bands or grid patterns of blank cells indicate that the specified lags are shorter than sample spacing (**Figure 38**). “Salt-and-pepper” maps (**Figure 39**) result from critically short lags, which can hinder interpretation. Conversely, excessively long lags may result in low resolution maps.



**Figure 38** a variogram map with a lag shorter than sample spacing in east-west direction, resulting in vertical bands of unvalued cells.



**Figure 39** a "salt-and-pepper" variogram map computed with critically short lags. This map was computed from the same data as the one in **Figure 36**

It is also important pay attention to the color scale. GammaRay by default sets the color scale to cover the entire range of values. Therefore, the user may check the global variance and set the maximum value in scale to this value (if the variogram map was computed in traditional semivariogram mode or covariances), which is the sill of stationary data sets. A constant color assigned to values greater than the variogram sill can help in interpreting anisotropy.

## Experimental variogram

Once the main anisotropy axes are known, the user can compute experimental variograms along the axes (by setting appropriate azimuth, dip and roll angles) to assess the variogram behavior in detail. Click on the middle C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the suggested parameters then click OK in the Parameters dialog to run gam (if you selected a variable of a grid) or gamv (for point sets).

One suggestion is to compute and optionally save (middle  button) at least three experimental variograms (for a 3D case):

1. With an omnidirectional variogram (**Figure 40**) that will allow the identification of the variogram structures;
2. With two areal directional variograms (**Figure 41**) to assess the horizontal ranges;
3. With a vertical variogram (**Figure 42**), especially for petroleum reservoirs and thin orebodies where the vertical ranges and lags are expected to be in much smaller scale than the areal lags and ranges.

The program generates a text file used by the vargplt program to display a color legend in the plot. The generated text file contains a legend relating the curve colors to the directions used (azimuth and dip for irregular data and X, Y and Z grid steps for regular data). The legend helps the user in the variogram modeling step.



**Figure 40** an omnidirectional experimental variogram.



**Figure 41** an experimental variogram along two azimuths. The color legend is automatically generated by GammaRay to help the user in the variogram modeling step.



**Figure 42** an experimental vertical variogram.

## Fitting a variogram model

After computing an experimental variogram, the user can fit a variogram model to it by clicking on the bottommost C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to open the Parameters dialog for the vmodel program. To facilitate variogram model fitting, GammaRay sets up the variogram plot parameters (vargplt program) so the user can view the experimental variogram (as dots by default) with the theoretical variogram (as solid lines by default) being fitted. When done, the variogram model can be saved in the project by clicking on the bottommost  button.

Another way to perform model fitting (**Figure 44**) is to right-click on a previously saved experimental variogram (files with the  icon) to open the Variogram Analysis dialog with only the variogram fitting buttons (**Figure 43**).



**Figure 43** the Variogram Analysis dialog configured for variogram fitting to a previously saved experimental variogram.



**Figure 44** directional variogram fitting. Experimental variogram as dots and variogram model as solid lines.

## Cross variograms

Cross variography is performed the same way as direct (or auto-) variography, except that the user must select two variables of the same data file, then right-click and choose the “Cross variography VAR1 x VAR2” menu option. ATTENTION: order is important to compute cross variograms. The variable selected first will be the head variable (z(u)) and the other will be the tail variable (z(u+h)). The user can check the head and tail selection in the Variogram Analysis dialog.

## *A priori* variogram models

The user can create a variogram model without going through the steps in Variogram Analysis dialog by right-clicking on the “Variogram Files” top-level group in the project tree. Select the “Create variogram model…” option to open the Parameters Dialog with default vmodel parameters (**Figure 45**).



**Figure 45** The Parameters Dialog configured for the vmodel program. The fields highligthed in green are for visualization only. The actual variogram model parameters are highlighted in red.

Review the parameters and click “OK” to plot the variogram model or “Cancel” to abort the task. Once in the Plot Dialog, if you click “Cancel”, signaling that you are not satisfied with the current model, the Parameters Dialog pops up again with the current vmodel parameter values for another modeling cycle. If you click “OK” in the Plot Dialgo, signaling that you finished the model, you will be asked to save or discard the variogram model.

Saved variogram models can be reviewed later by right-clicking on a variogram model (files with the  icon) and choosing “Review” to open the vmodel parameters in a dialog. The user can change the model or simply plot it following the same rationale for creating new variogram models.

## Variograms of a realization ensemble in a single file

The user can plot several experimental variograms in a single chart, which is useful for validating simulations. If the user selects a Cartesian grid that contains more than one realization, the C:\Users\paulocarvalho\Desktop\GammaRay\art\iconsHD\varnreals32.png button will appear in the Variogram Analysis Dialog, signaling that this feature is available. Clicking on it, a small dialog pops up with all the realizations selected by default.

After a selection is made, the Parameters Dialog appears, prompting the user to set the gam parameters for all the selected realizations (realization number will be ignored). After that, variogram computing and plotting take place, which may take a while, depending on the grid size and the number of realizations. **Figure 46** is an example of variograms computed for several realizations along two directions.



**Figure 46** Variograms of several realizations of a grid. Each color is a realization. Each line style is a direction.

## Variograms of a realization ensemble across separate files

If the simulation program delivers realizations as separate files, there is a different procedure outside the Variogram Analysis Dialog. To plot variograms this way, you select Attributes of Cartesian files (they can be of the same file) in the Project Tree. Then, right-click and choose the “Multiple variograms” option in the pop-up menu. A simple dialog will be displayed, where the user can check the selected Files/Variables and with a C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to bring the Parameters Dialog for the gam program.

In the Parameter Dialog, the user enters parameters such as directions steps and number of lags. Recall that parameters that vary with the different input Attributes/Files will be set automatically, ignoring what you might set, for instance: grid parameters, trimming limits and input file paths. After confirming the parameters, the program will run gam several times (may take a while), and then the plot will be displayed. The resulting plot looks similar to the single-file case (**Figure 46**).

# Estimation

To estimate is to compute the expected value of a random function in a location where it is unknown using existing samples around it as basis. GSLib offers estimation in the form of geostatistical methods, which mean that the estimates are computed based on statistical characteristics and following a model of spatial continuity, normally expressed by a variogram model. A typical application of geostatistical estimation is to produce a map of ore grades from scattered drill hole samples. A distinctive feature of geostatistical estimation is that it yields the estimation error in the form of kriging variance.

## Kriging

The GSLib program kt3d is used to perform estimation of a continuous variable (e.g. porosity). With kt3d, it is possible to perform simple kriging (SK), ordinary kriging (OK), kriging with locally varying mean (LVM) and kriging with an external drift (KED). These last two require a secondary variable or trend model.

SK is recommended for variables with a global constant mean (stationary), or the variable does not have a global trend. OK is recommended when a trend is observed. For instance, the experimental variogram of non-stationary data will not settle in a sill along specific azimuths.

If an extensive secondary variable is available, it can be used as a local mean to better inform the estimation. Hence, it is important that the secondary variable be in the same unit and scale of the primary. Otherwise, you can perform some kind of calibration to convert the secondary, but be aware of introducing bias. LVM is a simple kriging in which the constant mean is replaced with a value present in each estimation location. KED is an ordinary kriging in which the estimated local mean is computed from the secondary variable values, which tend to have a greater count than the primary. To decide between LVM and KED, the relative stationarity can be used, for which cross variograms can be computed to test whether it settles in a sill in all directions (both variables are relatively stationary). The secondary variable grid file does not need to be the same used for the estimation grid, but their geometry must match (same position, cell count and cell size) otherwise kt3d will quit with an error. If the secondary variable is missing in the samples file, you can transfer the collocated values from its grid file using the feature described in Section 7.2. If the variables are relatively stationary, you can use LVM, otherwise, KED.

To start kriging a continuous variable, go to the main menu “Estimation”, then select “Kriging (continuous)” to open the Kriging Dialog (**Figure 47**). Select the input files for the estimation then click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the parameters. The estimation starts after you close the Parameters Dialog.



**Figure 47** the kriging dialog (continuous variable).

After the estimation completes, the Plot Dialog will pop-up automatically showing the estimation result. If you run the estimation again, GammaRay will retain your variogram settings, unless you change to another variogram model. This is because the user may tune the variogram parameters during the estimation attempts. However, GammaRay will always re-read the data (primary and secondary) and estimation grid settings from the files chosen in the kriging dialog.

After you run the estimation at least once, you can optionally click on the  button to run kt3d again in cross validation mode and a cross plot will be displayed so the user can assess the quality of the estimates. Estimates with a correlation coefficient close to 1.0 with respect to the sample values are deemed satisfactory.

When done, click on the first  button to save the estimates and, optionally, the kriging variance to the selected estimation grid file. You can optionally click on the second  button to save the kriging variances (estimation error). The third  button serves to save or update the variogram model with the current variogram parameters used to compute the estimation. If you give a new name, you will create a new variogram model, otherwise you will update an existing one.

### Known issues

Problem: Kriging seems to complete normally, but the estimates grid is not displayed.

Possible cause: Estimates grid filled with all non-data-values. If you made sure the configured search strategy is adequate for your problem, then duplicate samples or samples too close to each other are likely causing numerical instability during kriging.

Solution: Look for duplicate samples (Section 7.5) and treat them by editing the sample file in an external editor.

## Indicator kriging (IK) of a continuous variable

The estimation values and kriging variances seen in Section 11.1 can be regarded as means and variances of local Gaussian distributions conditioned to data if one assumes or verifies the multi-Gaussianity hypothesis (multi-Gaussian kriging or MGK). The bigaus program, for instance, can be used to verify two-point (variogram based) normality. These estimated local Gaussian distributions are an early form of uncertainty modeling.

An indicator is a special variable that indicates an impossibility (0.0), a certainty (1.0) or a likelihood (values in between). Indicator values between 0.0 and 1.0 are often referred to as soft indicators. At a sample point (no estimation uncertainty), an indicator obviously assumes a binary value (0/1) that indicates whether the sample value is below (1.0) or above (0.0) a certain threshold. At an estimation location, due to uncertainty, the indicator can assume values between 0 and 1, to indicate the likelihood that the continuous variable at that location is below a threshold. The objective of IK is to estimate these indicator values to quantify such uncertainties.

Therefore, IK is interesting as it allows one to estimate arbitrary (non-parametric) distributions, meaning that no assumption is made. IK for continuous variable yields maps of local c.d.f.’s for each threshold, which alone are not usable values. Consequently, an IK post-processing program is normally used in tandem to obtain usable products. These post-processed products are normally for decision making and risk assessment, such as a probability map of ore grade being above a threshold. The GSLib program postik is used to post-process IK local c.d.f. estimates.

For a continuous variable, the user provides a series of increasing thresholds (e.g. 0.5 1.0 2.5 5.0 10.0) and an associated global cumulative distribution function (c.d.f.), for example: 0.12 0.29 0.50 0.74 0.88.

Since usually there are more than one threshold, IK performs an equivalent number of estimations at each estimation location, consequently the geomodeler must provide a variogram for each threshold/category. You can enable the median IK (mIK) mode, which requires just one variogram model, but assumes that all thresholds have the same spatial structure. This opens interesting modeling possibilities, for instance, the user can provide different variograms for each threshold to reflect their expected spatial behavior (high grades can be more erratic, for instance). To model variograms for this purpose, one selects the adequate variogram calculation option (indicator variogram for continuous variable) during the variogram modeling step (Section 10).

### Defining a threshold c.d.f. for a continuous variable

You may use the cumulative histogram (Section 6.2) or the probability plot (Section 6.4) to help in defining the thresholds-cumulative probability pairs (**Figure 48**).



**Figure 48** The crosshairs over a cumulative histogram of your samples can assist in defining threshold-cumulative probability pairs for Indicator Kriging.

A threshold c.d.f. can be created by right-clicking on the “Resource Files” group in the project tree and choosing the “Create threshold c.d.f. …” option. Then the value pairs editor (**Figure 49**) pops up enabling the user to maintain threshold c.d.f.’s in the project. Keeping c.d.f. files is not required for IK operation, but they help in organizing your data, otherwise the user would probably have to keep records elsewhere.



**Figure 49** The threshold-cumulative probability pairs editor used to keep record of threshold c.d.f.'s.

Upon saving, a new item with the given name appears under the “Resource Files” group in the project tree.

A c.d.f. can be reviewed or edited later by right-clicking on a  file in the project tree (under the “Resources” group), then clicking on the “Edit” menu item.

### Running IK for a continuous variable

For a continuous variable, ik3d outputs a grid of local c.d.f.’s for each threshold. Go to the “Estimation” menu of the program’s main window, then select the “Indicator Kriging (continuous)” item. The IK dialog (**Figure 50**) pops up, presenting the most relevant options for an easier configuration of the ik3d GSLib program, which has a particularly complex parameter set. Recall that the variogram models should ideally be fit (Section 10.3) to experimental variograms computed in indicator mode. Also, mind the order of the variograms: the first variogram from left to right is the variogram for the first threshold in the selected c.d.f..



**Figure 50** The IK dialog for a continuous variable.

Click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png icon to review the parameters and then click “OK” in the Parameters Dialog to run the estimation. Upon completion, the program will plot the local c.d.f. maps of all thresholds (**Figure 51**). Click on the first  button to save the new variables to the estimation grid file.

**ATTENTION**: if you plan to post-process the IK estimates (Section 11.2.3), then you must activate the second  button to save a new Cartesian grid containing only the estimates in the correct order (a requirement for postik).



**Figure 51** Examples of local c.d.f. maps for three thresholds estimated with IK. Notice the effect of the different variograms.

### Post-processing the results

If you saved the ik3d output grid for post-processing, you can proceed to post-process the indicator kriging estimates to get risk assessment products. Activate the “Estimation” option in the main window’s menu, then choose the “IK Post-processing” menu item to bring the indicator kriging post-processing dialog (**Figure 52**).



**Figure 52** The indicator kriging post-processing dialog.

You have obviously to select an ik3d output grid containing only the IK estimates in the correct order (selecting a generic grid with the IK estimates will not work) and the threshold c.d.f. used during the IK run. The c.d.f. will provide the threshold values required by postik, though the cumulative frequencies are not used.

You can optionally provide a data file to better characterize the cumulative distribution between the thresholds, if you enable the third option for the tails and in-between conditional c.d.f. interpolation modes. postik uses these data to tabulate quantiles to linearly interpolate the cumulative distribution between them in addition to the selected thresholds. Providing a declustering weight (Section 7.1) is optional, but be aware that irregularly sampled data may result in unrealistic distributions (sampling bias).

Click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png button to review the default parameters, selecting the desired output option depending on your study objectives. Click OK on the Parameters Dialog to run postik and display the results. The results depend on the selected post-processing product.

If you select **E-type** as post-processing option, then postik outputs a grid with the mean of the local cumulative distributions. If you select **Prob. and mean above**, you need to enter a threshold, then the tested version of postik outputs three fields: mean above, mean below and probability above. If you select **quantile**, you need to enter a p-value (ex.: 0.5 for P50 or median) between 0.0 and 1.0, then postik outputs the value corresponding to the quantile. If you select **variance**, a grid with the conditional variance will be computed.

If you enable volume support correction, two options will be available. The **affine** option means affine correction and the **indirect** option means indirect correction through permanence of a lognormal distribution. You also need to enter a variance reduction factor between 0.0 and 1.0.

Once you are satisfied with the presented post-processed product, click on  button to save it as a new grid to the project. You can go back to the parameters, then run postik again to generate another post-processed product, then save, and so on until you collect all the desired ones.

### Known issues

Problem: the kriging seems to complete normally, but the plots of the local c.d.f. values for each threshold are not displayed.

Possible cause: the resulting grids are filled with the constant values of the global c.d.f., causing the program to set the color scale beginning and ending at the same value. To test this, open the parameters of the Plot Dialog and set some valid color scale. You may see the grid painted in some solid color.

Solution: Set the trimming limits (in the ik3d parameters) to values that are not near the c.d.f.. The trimming limits may be reasonable, marking the minimum and maximum of your data, but they may be inside the value interval of the estimated c.d.f.’s. If GammaRay sets, for example 15.0 and 50.0 based on your data, try setting 0.0 and 100.0.

## IK for a categorical variable

At a sample point (no estimation uncertainty), an indicator assumes 1.0 if it belongs to a certain class and 0.0 otherwise. At an estimation location, due to uncertainty, the indicator can assume values between 0 and 1, to indicate the likelihood that the categorical variable belongs to a class. For a categorical variable, the user provides a list of all possible class codes (e.g. 3 5 1 2 4) and an associated global probability density function (p.d.f.), for example: 0.4 0.2 0.1 0.25 0.05.

IK for categorical variables yields probability fields for each category. Goovaerts (<https://www.mail-archive.com/ai-geostats@jrc.it/msg02054.html>) suggests, for instance, to simply pick the class with the greatest likelihood to get a usable facies map using a spreadsheet software. The probability fields can be used, for example, with SNESIM (Single Normal Equation Simulation) to produce multiple-point facies simulations. Usable facies maps can also be obtained by using these fields as local facies proportions with GTSIM (Truncated Gaussian Simulation).

Like the continuous variable case, usually there are more than category, IK then performs an equivalent number of estimations at each estimation location, consequently the geomodeler must provide a variogram for each category. To model variograms for this purpose, one selects the variogram calculation option to indicator variogram for categorical variable during the variogram modeling step (Section 10).

### Defining a class p.d.f. for a categorical variable

Plotting the histogram (Section 6.2) of a categorical variable can help in defining the p.d.f (**Figure 53**). If your data does not have categorical values, the program has tools to create categorical variables, please refer to Section 9.



**Figure 53** The histogram of a categorical variable can help in defining a p.d.f.

A category p.d.f. can be defined by right-clicking on a category definition file (files with the C:\Users\paulocarvalho\Desktop\GammaRay\art\catdef16x16.png icon) in the project tree and choosing the “Create category p.d.f. …” option. Please, refer to Section 9.1 to create category definitions. Then the value pair editor (**Figure 54**) pops up enabling the user to enter the probabilities for each category. A p.d.f. can be reviewed or edited later by right-clicking on a  file in the project tree (under the “Resource Files” group), then clicking on the “Edit” menu item. Recall that the probabilities must sum up 100%.



**Figure 54** Defining a p.d.f. for a categorical variable

### Running IK for a categorical variable

For a categorical variable, ik3d outputs a grid of local p.d.f.’s, that is: probabilities of each class. First, you need to classify (that is, convert values into integer category/class identifiers – see Section 7.5) your sample values, unless your data already features categorical variables. To run IK, go to the “Estimation” menu of the program’s main window, then select the “Indicator Kriging (categorical)” item. The IK dialog (**Figure 55**) pops up, presenting the most relevant options for an easier configuration of the ik3d GSLib program, which has a particularly complex parameter set. Recall that the variogram models should ideally be fit (Section 10.3) to experimental variograms computed in indicator mode. Also, mind the order of the variograms: the first variogram from left to right is the variogram for the first class in the selected category p.d.f..



**Figure 55** The IK dialog for estimating a categorical variable.

Click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\settings16x16.png icon to review the parameters and then click “OK” in the Parameters Dialog to run the estimation. Upon completion, the program will plot the probability fields of all classes (**Figure 56**). Click on the  button to save the new variables to the estimation grid file.



**Figure 56** Example of probability fields for three categories obtained with IK. Notice the complex geometry resulting from the multiple variogram models.

### Creating a maximum likelihood facies map

If you click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\faciesmap16x16.png button, the program will create a grid containing category codes. The codes in each cell are selected based on which has the greatest probability. After clicking the button, the categorical map is displayed (**Figure 57**). If you click on the OK button of the Plot Dialog, you will be asked to enter the name of the categorical variable to be added to estimation grid.



**Figure 57** Example of a maximum likelihood facies map computed with IK. Notice the complex facies geometry resulted from the multiple variogram models.

### Known issues

PROBLEM: GammaRay crashes following an error reported by ik3d : error in soft data file.

LIKELY CAUSE: The soft indicator file has more data points than the sample data file.

SOLUTION: Create a soft indicator file with less points or the same number of points of the sample data file.

PROBLEM: GammaRay crashes following an error reported by ik3d : domain error with sqrt (the square root function in Fortran) while processing the soft indicator file.

LIKELY CAUSE: The soft indicator file data points too dense.

SOLUTION: Create a soft indicator file with more sparse points.

## Secondary data: soft indicator calibration

Indicator-based algorithms require that secondary data be converted into soft indicators. One can create soft indicators by right-clicking on a variable of a point set, then choosing the “Soft indicator calibration…” item. This brings the Soft Indicator Calibration dialog. The appearance of the calibration curves depend on the type of the target soft indicators. If you choose a source of category definition (category p.d.f. file or category definition file), then the curves are shown in categorical mode (**Figure 58**), otherwise in continuous mode (**Figure 59**).



**Figure 58** Soft indicator calibration for a categorical attribute.



**Figure 59** Soft indicator calibration for the thresholds of a continuous attribute.

You set the calibration curves simply by dragging the curve points up and down according to your interpretation relating the category/threshold likelihoods to the selected secondary data. Click the  button to preview the results. Click on the C:\Users\paulocarvalho\Desktop\GammaRay\art\thrcdf16x16.png or the C:\Users\paulocarvalho\Desktop\GammaRay\art\catpdf16x16.png button to see the resulting global c.d.f. or p.d.f.. Once you are satisfied, you can click on the  button to save the computed soft indicators to the data file. Due to restrictions imposed by the ik3d program, GammaRay will create a new data file containing only the X, Y, Z coordinates and the soft indicators.

### Interpretation for categorical attributes

The geomodeler must think that, for each of all possible secondary data values, a set of soft indicators will be computed, summing up exactly 1.0 (100%). The example in **Figure 60** shows that for 30.0 secondary value, the probability of limestone will be 22%. The vertical extent of the filled areas means the likelihood of the corresponding category, varying within the range of the secondary data. The areas are filled with the colors assigned to their respective categories.



**Figure 60** Interpreting the calibration curves for the categorical case: for 30.0 value, the probability of limestone is about 22% (red line).

### Interpretation for continuous attributes

The probabilities are now cumulative, meaning that for each threshold, you set the probability of the continuous attribute being less than the given threshold along the entire range of the secondary data.



**Figure 61** Interpreting the calibration curves for the continuous case: for the 1000.0 value (thick red line), the resulting c.d.f. for the thresholds (25.0, 35.0 and 45.0). is shown to the right.

### Multiple secondary data

One way to use multiple secondary data is to perform the soft indicator calibration on each of them separately, then use some probability integration model such as the Tau Model or Bayesian Updating to compute a single set of soft indicators for the algorithm.

### Known issues

PROBLEM: Calibration seems to run successfully, but the preview shows a blank plot.

LIKELY CAUSE: Soft indicator field with a constant value, resulting from a flat calibration curve (default curve). The all-constant values result in incorrect automatic color scale, compromising the plot.

SOLUTION: Make sure you change at least one point of the calibration curves. Alternatively, you can manually configure color scale in the Plot Dialog to display the points.

## Cokriging

LVM and KDE use secondary data to inform only the estimation mean. Cokriging uses secondary data to inform the estimation itself, without requiring the secondary data to be in the same unit and scale of the primary. It also avoids the risk of bias caused by a regression used to convert the secondary data into the primary data unit.

### Cokriging variography

The downside of cokriging is that, now that the secondary data takes part on the kriging equation, it demands fitting another variogram model. In addition, cross-variograms between the primary and secondary are necessary to complete the covariance matrix for the cokriging equation. This means, for two variables, four variogram models, at least for a rigorous cokriging procedure. This complication leads to a variogram modeling decision.

Furthermore, the mixing of possibly very different variogram models may lead to numerical instabilities with the covariance matrix. Hence, the several variogram models must form a Linear Model of Coregionalization (LMC) to assure the same required properties with a single variogram, namely positive definiteness.

You can assume no lag effect (also known as delay effect) to make the primary-secondary and secondary-primary cross-variograms the same, reducing the variogram count to three. It is also possible to assume the variograms share the same structures (models and ranges) differing only in the contributions, thus it is only necessary to model a variogram for one of the variables.

If the secondary data is extensive, you can use only the secondary data already located in the estimation locations to inform the estimation, an operation called Collocated Cokriging (the normal cokriging is called Full Cokriging). Collocated cokriging requires only the variogram model of the primary (Markov Model 1) or of the secondary (Markov Model 2).

### The Linear Model of Coregionalization (LMC)

To prevent numerical instabilities in the kriging computation, the set of variograms between any combination of two variables in the problem must form a LMC. That is, the two direct variograms and the cross variogram (assuming no lag-effect) must be linear combinations of the same nested structures (nugget effect γ0 + two structures γ1, γ2 in the following example):

Where γU(h), γV(h), γUV(h) are the direct and cross variograms of the variables U and V. *u*, *v* and *w* are weights or coefficients that can be negative. These coefficients are the non-standardized contributions of each nested structure, that is, their covariance contributions not scaled to [0.0 1.0] range. The example LMC can be rewritten as three matrix operations, for each structure:

Finally, to ensure positive definiteness, the following conditions must be observed for this case:

* The *u* and *v* coefficients must be positive for the three nested structures.
* *u*∙*v* > *w*² for each nested structure, that is, the determinants of the three coefficient matrices must be positive.

In addition, the following must also be checked:

* All variogram models must be composed by the same basic structures, angles and ranges.
* The power model is not allowed as imposed by cokb3d.

The user can check the variography validity for cokriging by pressing the “LMC Check” button in the cokriging dialog (**Figure 62**).

### Cokriging with cokb3d

The cokb3d program supports only full cokriging using a general LMC. Even though the co-located option is available, it is not implemented in canon version of cokb3d.

To start cokriging with cokb3d, go to the menu Estimation->Cokriging (cokb3d) option in the main window to bring the cokriging dialog. It is important to check whether the selected variograms form a LMC by clicking the “LMC Check” button. The program will either confirm goodness or will report the detected problems in the output messages panel in the main window, prompting the user to make the necessary adjustments to the variogram models. Please, refer to Section 10.5 on how to edit existing variogram models.



**Figure 62** The cokriging dialog.

To save the estimates to the selected estimation grid, simply click on the first  button. Optionally, click on the second  button to save the kriging variances.

## Estimation of unvalued cells of a grid (NDV estimation)

NDV stands for no-data-value, a value defined for a file that signals absence of data (see Section 5.3). Sometimes data in form of Cartesian grids or grids produced by geomodeling software come with “holes” in them. Some algorithms require that the input grid has vales in all of its cells (e.g. Fourier Transform – Section 8.2) or simply the user wants a map without discontinuities. To perform this kind of estimation, right-click on a variable of Cartesian grid, then activate the “NDV estimation” option to open the appropriate dialog (**Figure 63**).



**Figure 63** The dialog used to estimate unvalued cells in a grid.

Due to potentially large number of cells to estimate, GammaRay performs kriging with a code tailored for this purpose, which is faster than exporting the entire grid as a point set (Section 7.4) and using the more generic kt3d alternative. NDV estimation used only the values already in the grid for estimation. The search neighborhood is a parallelepiped defined in cell units for each topological direction (IJK) around the estimation cell. The “use as default value (SK or OK)” option enables the user to set a value for cells which cannot be estimated due to absence of values in its neighborhood, otherwise NDV will be used. The other options in the dialog are usual options required by other kriging algorithms.

After clicking “Run”, the user enters the name of the new Cartesian grid to contain the estimates. Valued cells are simply copied instead of being estimated. NDV estimation is a point kriging procedure considering the values located at the centers of grid cells, following GSLib convention. Keep in mind variable block size issues (block-to-block variance) if the Cartesian grid is the data source of an irregular grid.

## (CONSIDERING) Highly multivariate estimation

Cokriging with more than two variables is a daunting task. Fitting six variograms for a three-variable case is practically prohibitive. You can go back to a bivariate problem by merging the multiple secondary variables into a supersecondary (REF).

Another way to deal with a highly multivariate problem is to decorrelate the variables, estimate them separately using univariate kriging and then reverse the decorrelation to obtain the correlated estimations. Decorrelation can be done at h=0 (collocated decorrelation) with Principal Component Analysis (PCA, REF) or at a given h>0 (spatial decorrelation) with Minimum/Maximum Autocorrelation Factors (MAF, REF).

# (PLANNED) Simulation

Variance inflation (Intrinsic Col-CoK).

## Post-processing

# Advanced topics

## (CONSIDERING) Workflows

Complex geostatistical modeling workflows involving multiple data transforms, estimation, and simulation steps can be designed with the Workflow Designer. Using the Workflow Designer allows the user to graphically design a workflow much like drawing flow charts. A graphic presentation, like in the famous MathWorks Simulink ®, also allows a broad view of what the workflow does, which can help the user to debug problems.

## (CONSIDERING) Extending GammaRay

GammaRay is not limited to the standard GSLib programs. The user can employ non-standard programs with GammaRay, given it accepts parameter files following the GSLib standard. The user can even create a custom GSLib-like program to use with GammaRay using any computer language since GammaRay interfaces the programs via parameter files, which are just text files.

To use non-standard programs, the user must write a parameter template file following a syntax, which allows GammaRay to interface to such programs.

SYNTAX

# (WIP) References

1. histpltsim is a non-standard GSLib-like program, it may be more difficult to obtain or you may find one not compatible with GammaRay-generated parameter file for it. [↑](#footnote-ref-2)