General Long Baseline Experiment Simulator (GLoBES)

PAVAN POOT PANDEY

August 8, 2012

Abstract

GLoBES (General Long Baseline Experiment Simulator) is a flexible software package to simulate neutrino oscillation long baseline and reactor experiments. It allows to simulate experiments with stationary point source, where each experiment is assumed to have only one neutrino source. Here some of the basic features, terminology of GLoBES have been described and also how it works with C-program.

Contents

0.1 Introduction . . . 2
  0.1.1 Terms of usage of GLoBES . . . 3
0.2 GLoBES Installation . . . 3
  0.2.1 Prerequisites for GLoBES installation . . . 3
  0.2.2 Installation Instructions . . . 4
  0.2.3 Basic Installation . . . 4
  0.2.4 Installation without root privilege . . . 5
  0.2.5 Building and Using static versions of GLoBES . . . 5
  0.2.6 GSL requirements . . . 6
0.3 Writing a program in GLoBES . . . 6
  0.3.1 Initialisation of GLoBES . . . 6
  0.3.2 Units in GLoBES and the integrated luminosity . . . 8
  0.3.3 Handling oscillation parameter vectors . . . 9
  0.3.4 Computing the simulated data . . . 10
  0.3.5 Version control and debugging . . . 11
0.4 Writing a GLoBES program in C . . . 11
0.5 Plots and Results . . . 13

0.1 Introduction

GLoBES (General Long Baseline Experiment Simulator) is a flexible software package to simulate neutrino oscillation long baseline and reactor experiments. GLoBES allows to simulate experiments with stationary neutrino point sources, where each experiment is assumed to have only one neutrino source. Such experiments are neutrino beam experiments and reactor experiments. Geometrical effects of a source distribution, such as in the sun or the atmosphere, can not be described. In addition, sources with a physically significant time dependence, such as supernova, can not be studied. It is, however, possible to simulate beams with bunch structure, since the time dependence of the neutrino source is physically only important to suppress backgrounds. On the experiment definition side, either built-in neutrino fluxes (e.g., neutrino factory, β-Beam) or arbitrary fluxes can be used. Similarly, arbitrary cross sections, energy dependent efficiencies, the energy resolution function, the considered oscillation channels, backgrounds, and many other features can be specified.

For the systematics, energy normalization and calibration errors can be simulated in a straightforward way, or the systematics can be completely user-defined (Version 3.0 and higher). Note that the energy ranges and windows, as well as the bin widths can be (almost) arbitrarily chosen, which means that variable bin widths are allowed. Together with GLoBES comes a number of pre-defined experiments in order to demonstrate the capabilities of GLoBES and to provide prototypes for new experiments.

With the C-library, one can extract the $\triangle\chi^2$ for all defined oscillation channels for an experiment or any combination of experiments. Of course, also low-level information, such as oscillation probabilities or event rates, can be obtained. GLoBES includes the simulation of neutrino oscillations in matter with arbitrary matter density profiles, as well as it allows to simulate the matter density uncertainty. As one of the most advanced features of GLoBES, it provides the technology to project the $\triangle\chi^2$, which is a function of all oscillation parameters, onto any subspace of parameters by local minimization. This

approach allows the inclusion of multi-parameter-correlations, where external input (e.g., from solar experiments) can be imposed, too. Applications of the projection mechanism include the projections onto the $sin^2\theta_{13}$-axis and the $sin^2\theta_{13}$- δ_{CP}-plane. In addition, all oscillation parameters can be kept free to precisely localize degenerate solutions. In the newest version GLoBES 3.0 flexibility is introduced at all levels. At the probability level, the transition probabilities can be modified to introduce new physics. At the systematics level, user-defined systematical errors and correlations between sources or detectors can be simulated, and at the analysis level, arbitrary input from external measurements can be added. Therefore, GLoBES now provides solutions for new classes of problems.

0.1.1 Terms of usage of GLoBES

Referencing the GLoBES software: GLoBES is developed for academic use. Thus, the GLoBES Team would appreciate being given academic credit for it. Whenever one uses GLoBES to produce a publication or a talk indicate that he/she have used GLoBES and please cite the following references [1, 2] P. Huber, M. Lindner and W. Winter Simulation of long baseline neutrino oscillation experiments with GLoBES, Comput. Phys. Commun. 167 (2005) 195, arXiv:hep-ph/0407333, P. Huber, J. Kopp, M. Lindner, M. Rolinec, and W. Winter New features in the simulation of neutrino oscillation experiments with GLoBES 3.0, arXiv:hep-ph/0701187. Besides that, many of the data which are used by GLoBES and distributed together with it should be properly referenced. Apart from that, GLoBES is free software and open source, i.e., it is licensed under the GNU Public License.

Referencing the data in GLoBES: GLoBES wouldnt be useful without having high quality input data. Much of these input data have been published elsewhere and the authors of those publications would appreciate to be cited whenever their work is used. It is solely the users responsibility to make sure that he understands where the input material for GLoBES comes from and if additional work has to be cited in addition to the GLoBES papers [1, 2]. To assist with this task, we provide the necessary information for the data coming along together with GLoBES. When using the built-in Earth matter density profile, the original source is Refs. All files ending with .dat or .glb in the data subdirectory of the GLoBES tar-ball have on top a comment field which clearly indicates which studies should be cited when using a certain file. Make sure that dependencies are correctly tracked, i.e., in some cases files included by other files need to be checked, too (for example, cross section or flux files). One can use the -v3 option to the globes command to see which files are included. It is recommended that one uses the same style for one's own input files, since then, in case they are distributed, everybody will know how to correctly reference one's work.

0.2 GLoBES Installation

This is GLoBES, a collection of numerical routines for the simulation and analysis of neutrino oscillation experiments.

GLoBES is free software, one can redistribute it and/or modify it under the terms of the GNU General Public License.

The GNU General Public License does not permit this software to be redistributed in proprietary programs.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

0.2.1 Prerequisites for GLoBES installation

For GLoBES installation, besides the usual things like a working libc one needs to have

gcc The GNU compiler collection
gcc.gnu.org

GSL The GNU Scientific Library
www.gnu.org/software/gsl/

The library ‘libglobes’ should in principle compile with any C/C++ compiler but the ‘globes’ binary uses the ‘argp’ facility of ‘glibc’ to parse its command line options. However, on platforms where ‘argp’ is lacking GLoBES has replacement code, thus it should also work there. GLoBES is, however, using the C99 standard in order to handle complex numbers, but that is the only feature of C99 used.

GSL is also available as rpm’s from the various distributors of GNU/Linux, see their web sites for downloads. Chances are that gcc and GSL are already part of your installation. For building GLoBES from source, however, not only working libaries for above packages are needed but also the headers, especially for GSL. Depending on your installation, eg. on RedHat/Fedora, of GSL this may require to additionally install a rpm-package named ‘gsl-devel’. If GSL has been installed from the tar-ball as provided by gnu.org no problems should occur. Furthmore you need a working ‘make’ to build and install GLoBES.

0.2.2 Installation Instructions

GLoBES follows the standard GNU installation procedure. To compile GLoBES you will need an ANSI C-compiler. After unpacking the distribution the Makefiles can be prepared using the configure command,

./configure

You can then build the library by typing,

make

A shared version of the library will be compiled by default.
The libraries and modules can be installed using the command,

make install

The install target also will install a program with name globes to /usr/local/bin.

The default install directory prefix is /usr/local. Consult the ”Further Information” section below for instructions on installing the library in another location or changing other default compilation options.

Moreover a config-script called ‘globes-config’ will be installed. This script displays all information necessary to link any program with GLoBES. For building static libaries and linking against them see the corresponding section of this file.

0.2.3 Basic Installation

These are generic installation instructions.

The ‘configure’ shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a ‘Makefile’ in each directory of the package. It may also create one or more ‘.h’ files containing system-dependent definitions. Finally, it creates a shell script ‘config.status’ that you can run in the future to recreate the current configuration, a file ‘config.cache’ that saves the results of its tests to speed up reconfiguring, and a file ‘config.log’ containing compiler output (useful mainly for debugging ‘configure’).

If you need to do unusual things to compile the package, please try to figure out how ‘configure’ could check whether to do them, and mail diffs or instructions to the address given in the ‘README’ so they can be considered for the next release. If at some point ‘config.cache’ contains results you don’t want to keep, you may remove or edit it.

The file ‘configure.in’ is used to create ‘configure’ by a program called ‘autoconf’. You only need ‘configure.in’ if you want to change it or regenerate ‘configure’ using a newer version of ‘autoconf’.

The simplest way to compile this package is:

1. ‘cd’ to the directory containing the package’s source code and type ‘./configure’ to configure the package for your system. If you’re using ‘csh’ on an old version of System V, you might need to type ‘sh ./configure’ instead to prevent ‘csh’ from trying to execute ‘configure’ itself.

Running ‘configure’ takes awhile. While running, it prints some messages telling which features it is checking for.

2. Type 'make' to compile the package.

3. Type 'make install' to install the programs and any data files and documentation.

4. You can remove the program binaries and object files from the source code directory by typing 'make clean'. To also remove the files that 'configure' created (so you can compile the package for a different kind of computer), type 'make distclean'. There is also a 'make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

5. Since you've installed a library don't forget to run 'ldconfig'!

0.2.4 Installation without root privilege

Install GLoBES to a directory of your choice GLB_DIR. This is done by

configure –prefix=GLB_DIR

and then follow the usual installation guide. The only remaining problem is that you have to tell the compiler where to find the header files, and the linker where to find the library. Furthermore you have to make sure that the shared object files are found during execution. Running 'configure' also produces a 'Makefile' in the examples subdirectory which can serve as a template for the compilation and linking process, since all necessary flags are correctly filled in. Another solution is to set the environment variable LD_RUN_PATH during linking to GLB_DIR/lib/. Best thing is to add this to your shell dot-file (e.g. .bashrc). Then you can use: A typical compiler command like

gcc -c my_program.c -IGLB_DIR/include/

and a typical linker command like

gcc my_program.o -lglobes -LGLB_DIR/lib/ -o my_executable

More information on this issue can be obtained by having a look into the output of make install.

CAVEAT: It is in principle possible to have many installations on one machine, especially the situation of having an installation by root and by a user at the same time might occur. However it is strictly warned against this possibility since it is *extremely* likely to create some versioning problem at some time!

0.2.5 Building and Using static versions of GLoBES

In certain circumstances it may be useful to use a static version of libglobes or any of the binaries, e.g. when running on a cluster.

The 'configure' script accepts the option '–disable-shared', in which case only static objects are built, i.e. only a static version of libglobes. In case your system does not support shared libraries the 'configure' script recognizes this. If you give no options to 'configure', both shared and static versions are built and will be installed. All binaries, however, will use dynamic linking. If you want to build static binaries, use LDFLAGS='-all-static' for building them.

Sometimes it is convenient, eg. for debugging purposes, to have a statically linked version of a program using GLoBES, which is easiest achieved by just linking with 'libglobes.a'. If you need a completely statically linked version, please, have a look at the Makefile in the 'examples' directory.

make example-static

produces a statically linked program that should in principle run on most Linuxes. It should be straightforward to adapt this example to your needs.

All these options rely on a working gcc installation. It seems that gcc 3.x is broken in subtle way which makes it necessary to add a symbolic link in the gcc library directory. The diagnostic for this requirement is that building static programs fails with the error message 'cannot find -lgcc_s'. In those cases, find 'libgcc.a' and add a symbolic link in the same directory where you found it (this requires probably root privileges)

ln -s libgcc.a libgcc_s.a

If you can not write to this directory just use the following work around. Add the same link as above to the directory where you installed GLoBES into cd prefix/lib

ln -s path_to_libgcc.a/libgcc.a libgcc_s.a

and then change back into the 'examples' directory and type

make LDFLAGS=-Lprefix/lib example-static and you are done.

0.2.6 GSL requirements

Sometimes the GNU scientific library is not available or is installed in a non-standard location. This situation can arise in an installation without root privileges. In this case one can specify '–with-gsl-prefix=path_to_gsl' as option to the 'configure' script. If one wants to use a shared version of 'libgsl' then one has to make sure that the linker find the library at run-time. This can be achieved by setting the environment variable LD_LIBRARY_PATH correctly, i.e. (in bash) export LD_LIBRARY_PATH='path_to_gsl' You also can use a static version of GSL by either building GLoBES with LDFLAG='-all-static' or by configuring GSL with '–disable-shared'. In both cases no further actions like setting any environment variables is necessary.

0.3 Writing a program in GLoBES

Here i will show, how to write a program in C-language and how to load pre-defined experiments and also will introduce to the basic concepts of GLoBES. A GLoBES progarm written in C-language in given here. After the the installation of GLoBES, it can be compiled using the Makefile in the examples directory. The Makefile has been correctly setup by the configure script to take into account details of the installation on your system. Thus youve just to type make and youre done.1 This Makefile very well serves as a template for your own applications.

0.3.1 Initialisation of GLoBES

Before one can use GLoBES, one has to initialize the GLoBES library :

Function 2.1: void glbInit(char *name) initializes the library libglobes and has to be called in the beginning of each GLoBES program. It takes the name name of the program as a string to initialize the error handling functions. In many cases, it is sufficient to use the first argument from the command line as the program name (such as in the example on page 10 below).

****
This is a basic C-code for all GLoBES programs.

```
// A C−code format for almost all type of GLoBES program

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

#include <globes/globes.h> /* Include GLoBES library */
#include "myio.h"     /* Include “housemade” I/O-routines */
                                                                      10

/* If filename given, write to file; if empty, to screen:*/
char MYFILE[]="testX.dat";


int main(int argc, char *argv[])
```

```
{
   glbInit(argv[0]);       /* Initialize GLoBES library */
                                                                        20

  glbInitExperiment("NFstandard.glb",&glb_experiment_list[0],
  &glb_num_of_exps); /* Initialize experiment NFstandard.glb */

/* Initialize housemade output function */
  InitOutput(MYFILE,"Format: . . . . . . . . . \n");

/* Initialize parameter vector(s) */                                    30
 glb_params true_values = glbAllocParams();
/* . . . */

/* Assign: theta12,theta13,theta23,deltacp,dm2solar,dm2atm */
 glbDefineParams(true_values,
               asin(sqrt(0.8))/2,asin(sqrt(0.001))/2,M_PI/4,M_PI/2,7e−5,2e−3);
 glbSetDensityParams(true_values,1.0,GLB_ALL); // Matter scaling
                                                                        40
/* The simulated data are computed */
 glbSetOscillationParameters(true_values);
 glbSetRates();

/* . . .CODE . . .*/

/* Free parameter vector(s) */                                          50
 glbFreeParams(true_values);
/* . . . */

 exit(0);

}
```
****

In principle, the GLoBES user interface can currently handle up to 32 of different longbaseline experiments simultaneously, where the number of existing experiment definition files can, of course, be unlimited. Note that each experiment assumes a specific matter density profile, which means that it makes sense to simulate different operation modes within one experiment definition, and physically different baselines, in different definitions. For details of the rate computation and simulation techniques, we refer at this place to Part II. Though the simplest case of simulating one experiment may be most often used, using several experiments is useful in many cases. For example, combinations of experiments can be tested for complementarity and competitiveness by equal means within one program. In general, many GLoBES functions take the experiment number as a parameter, which runs from 0 to glb_num_of_exps-1 in the order of their initialization in the program. In addition, using the parameter value GLB_ALL as experiment number (for example, in the glbChi... functions) initiates a combined analysis of all loaded experiments.

For storing the experiments, GLoBES uses the initially empty list of experiments glb_experiment_list. To add a pre-defined experiment to this list, one can use the function glbInitExperiment:

Function 2.2: int glbInitExperiment(char *infile, glb_exp *ptr, int *counter) adds a single experiment with the filename infile to the list of cur- rently loaded experiments. The counter is a pointer to the variable containing the number of experiments, and the experiment ptr points to the beginning of the experiment list. The function returns zero if it was successful.

Normally, a typical call of glbInitExperiment is

glbInitExperiment("NFstandard.glb",&glb_experiment_list[0],

&glb_num_of_exps);

In this case, the experiment in the file NFstandard.glb is added to the internal global list of experiments, and the experiment counter is increased. The experiment then has the number glb_num_of_exps-1. The elements of the experiment list have the type glb_exp, which the user will not need to access directly in most cases. The experiment definition files, which usually end with .glb, and any supporting files, are first of all searched in the current directory, and then in the path given in the environment variable GLB_PATH.

One can also remove all experiments from the evaluation list at running time:

Function 2.3: void glbClearExperimentList() removes all experiments from the internal list and resets all counters.

Note that changing the number of experiments requires a new initialization of all parameters of the types glb_params and glb_projection if the number of experiments changes, since these parameter structures internally carry lists for the matter densities of all experiments. Similarly, once should never call glbAlloc... before the experiment initialization.

0.3.2 Units in GLoBES and the integrated luminosity

While interacting with the user interface of GLoBES, parameters are transferred to and from the GLoBES library. In GLoBES, one set of units for each type of quantity is used in order to avoid confusion about the definition of individual parameters. In principle, the event rates are proportional to the product of source power x target mass x running time, which we call integrated luminosity. Since especially the definition of the source power depends on the experiment type, the quantities of the three luminosity components are not unique and depend on the experiment definition. Usually, one uses detector masses in kilotons for beam experiments, and detector masses in tons for reactor experiments. Running times are normally given in years, where it is often assumed that the experiment runs 100of the year. Thus, for shorter running periods, the running times need to be renormalized. Source powers are usually useful parent particle decays per year (neutrino factories, β-beams), target power in mega watts (superbeams), or thermal reactor power in giga watts (reactor experiments).

Function 2.4: void glbSetSourcePower(int exp, int fluxno, double power) sets the source power of experiment number exp and flux number fluxno to power. The defi- nition of the source power depends on the experiment type as described above.

Function 2.5: double glbGetSourcePower(int exp, int fluxno) returns the source power of experiment number exp and flux number fluxno.

Function 2.6: void glbSetRunningTime(int exp, int fluxno, double time) sets the running time of experiment number exp and flux number fluxno to time years.

Function 2.7: double glbGetRunningTime(int exp, int fluxno) returns the running time of experiment number exp and flux number fluxno.

Function 2.8: void glbSetTargetMass(int exp, double mass) sets the fiducial detec- tor mass of experiment number exp to mass tons or kilotons (depending on the experiment definition).

Function 2.9: double glbGetTargetMass(int exp) returns the fiducial detector mass of experiment number exp.

Thus, these functions also demonstrate how to use the assigned experiment number and others. These numbers run from 0 to the number of experiments-1, fluxes-1, etc., where the individual elements are numbered in the order of their appearance. Note that the source power and running time are quantities defined together with the neutrino flux, whereas the target mass scales the whole experiment. Thus, if one has, for instance, a neutrino and an antineutrino running mode, one can scale them independently.

0.3.3 Handling oscillation parameter vectors

Before we can set the simulated event rates or access any oscillation parameters, we need to become familiar with the concept GLoBES uses for oscillation parameters. In order to transfer sets of oscillation parameter vectors (θ_{12}, θ_{13}, θ_{23}, δ_{CP}, $\triangle m^2_{21}$, $\triangle m^2_{31}$) as well as some other information, the parameter type glb_params is used. In general, this type is often transferred to and from GLoBES functions. Therefore, the memory for these vectors has to be reserved (allocated) before they can be used, and it has to be returned (freed) afterwards. GLoBES functions usually use pointers of the type glb_params for the input or output to the functions. As an input parameter, the pointer has to be initialized with the address of a valid parameter structure, where the oscillation parameters are read from. As an output parameter, it has to be initialized with the address of a structure which the return values will be written to. This parameter transfer concept seems to be very sophisticated, but, as we will see in the next chapters, it hides a lot of complicated parameter mappings which otherwise need to be done by the user. For example, not only the oscillation parameters are stored in the glb_params structure, but also information on the matter densities of all

of the initialized experiments. Since GLoBES treats the matter density as a free parameter known with some external precision to include matter density uncertainties, the minimizers also use fit values and external errors for the matter densities of all loaded experiments. More precisely, the matter density profile of each experiment i is multiplied by a scaling factor ρ_i, which is stored in the density information of glb_params. For a constant matter density, it is simply the ratio of the matter density and the average matter density specified in the experiment definition, i.e., $\rho_i \equiv \rho_i/\bar{\rho}_i$. For a matter density profile, it acts as an overall normalization factor: The matter density in each layer is multiplied by this factor. In most cases one wants to take a scaling factor of 1.0 here, which simply means taking the matter density profile as it is given in the experiment definition. For the treatment of correlations, however, an external precision of the scaling factor might be used to include the correlations with the matter density uncertainty. Note that the glb_params structures must not be initialized before all experiments are loaded, since the number of matter densities can only be determined after the experiments are initialized. Similarly, any change in the number of experiments requires that the parameter structures be re-initialized, i.e., freed and allocated again.

Another piece of information that will be returned from the minimizers and transferred into the glb_params structure is the number of iterations used for the minimization, which is proportional to the running time of the minimizer. In general, the user does not need to access the elements in glb_params directly. A number of functions is provided to handle these parameter structures:

Function 2.10: glb_params glbAllocParams() allocates the memory space needed for a parameter vector and returns a pointer to it. All parameter values are initially set to nan (not a number).

Function 2.11: void glbFreeParams(glb_params stale) frees the memory needed for a parameter vector stale and sets the pointer to NULL.

Function 2.12: glb_params glbDefineParams(glb_params in, double theta12, double theta13,double theta23, double delta, double dm21, double dm31) as- signs the complete set of oscillation parameters to the vector in, which has to be allocated before. The return value is the pointer to in if the assignment was successful, and NULL otherwise.

Function 2.13: glb_params glbCopyParams(const glb_params source, glb_params dest) copies the vector source to the vector destination. The return value is NULL if the assignment was not successful.

Function 2.14: void glbPrintParams(FILE *stream, const glb_params in) prints the parameters in in to the file stream. The oscillation parameters, all density values, and the number of iterations are printed as pretty output. Use stdout for stream if you want to print to the screen.

In addition to these basic functions, there are functions to access the individual parameters within the parameter vectors:

Function 2.15: glb_params glbSetOscParams(glb_params in, double osc, int which) sets the oscillation parameter which in the structure in to the value osc. If the assignment was unsuccessful, the function returns NULL.

Function 2.16: double glbGetOscParams(glb_params in, int which) value of the oscillation parameter which in the structure in. returns the In both of these functions, the parameter which runs from 0 to 5 (or the number of oscillation parameters-1), where the parameters in GLoBES always have the order θ_{12}, θ_{13}, θ_{23}, δ_{CP}, $\triangle m^2_{21}$, $\triangle m^2_{31}$. Alternatively to the number, the constants GLB_THETA_12, 21 31 GLB_THETA_13, GLB_THETA_23, GLB_DELTA_CP, GLB_DM_21, or GLB_DM_31 can be used. Similarly, the density parameters or number of iterations (returned by the minimizers) can be accessed:

Function 2.17: glb_params glbSetDensityParams(glb_params in, double dens, int which) sets the density parameter which in the structure in to the value dens. If the assignment was unsuccessful, the function returns NULL. If GLB_ALL is used for which, the density parameters of all experiments will be set accordingly.

Function 2.18: double glbGetDensityParams(glb_params in, int which) the value of the density parameter which in the structure in. returns

Function 2.19: glb_params glbSetIteration(glb_params in, int iter) sets the number of iterations in the structure in to the value iter. If the assignment was unsuccessful, the function returns NULL.

Function 2.20: int glbGetIteration(glb_params in) returns the value of the number of iterations in the structure in.

In total, the parameter vector handling in a program normally has the following order:

glbInitExperiment(...);

/* ... more initializations ...*/

glb_params vector1 = glbAllocParams();

/* ... more vectors allocated ... */

/* Program code:assign and use vectors */

0.3.4 Computing the simulated data

Compared to existing experiments, which use real data, future experiments use simulated data. Thus, the true parameter values (or simulated parameter values) are used to calculate the reference event rate vectors corresponding to the simulated experiment result. After setting the true parameter values, the fit parameter values can be varied in order to obtain information on the measurement performance for the given set of true parameter values. Therefore, it is often useful to show the results of a future measurement as function of the true parameter values for which the reference rate vectors are computed at least within the currently allowed ranges. The true parameter values for the vacuum neutrino oscillation parameters have to be set by the function glbSetOscillationParameters, and the reference rate vector, i.e. the data, has to be computed by a call to glbSetRates. This has to be done before any evaluation function is used and after the experiments have been initialized and also the experiment parameters have been adjusted which could change the rates (such as baseline or target mass). This means that after any change of an experiment parameter, glbSetRates has to be called. Matter effects are automatically included as specified in the experiment definition. We have the following functions to assign and read out the vacuum oscillation parameters:

Function 2.21: int glbSetOscillationParameters(const glb_params in) sets the vacuum oscillation parameters to the ones in the vector in.

Function 2.22: int glbGetOscillationParameters(glb_params out) returns the vac- uum oscillation parameters in the vector out. The result of the function is 0 if the call was successful.

The reference rate vector is then computed with:

Function 2.23: void glbSetRates() computes the reference rate vector for the neutrino oscillation parameters set by glbSetOscillationParameters.

0.3.5 Version control and debugging

In order to keep track of the used version of GLoBES, the software provides a number of functions to check the GLoBES and experiment versions. It is up to the user to implement mechanisms into the program and AEDL files to check whether

- The program should only run with this specific version of GLoBES.
- The program can only run up to a certain GLoBES version.
- The program can only run with a minimum version of GLoBES.
- The program and AEDL file versions are compatible.

However, note that GLoBES 3.0 and higher requires that the GLoBES version be at least as new as the version the AEDL file was written for. The functions in GLoBES for version control are:

Function 2.24: int glbTestReleaseVersion(const char *version) returns 0 if the version string of the format X.Y.Z is exactly the used GLoBES version, 1 if it is older, and 1 if it is newer.

Function 2.25: const char* glbVersionOfExperiment(int experiment) returns the version string of the experiment number experiment (set by version in AEDL). The version string is allocated within the experiment structure, which means that it cannot be altered and must not be freed by the user. A useful function to debug GLoBES source code is Function 2.26 int glbSetVerbosityLevel(int level) sets the verbosity level for GLoBES messages. The following verbosity levels are currently used:

0 Display no messages

1 (Standard) Display error messages

2 Display warnings

3 Display file access history

4 Display search paths

Note that always messages with the chosen verbosity level and lower are displayed.

0.4 Writing a GLoBES program in C

This is a typical program written in C-language. This program calculates sensitivity of detector using near and far detectors.

```
==========================================================================
/* GLoBES – General LOng Baseline Experiment Simulator
 * (C) 2002 - 2007, The GLoBES Team
 *
 * GLoBES is mainly intended for academic purposes. Proper
 * credit must be given if you use GLoBES or parts of it. Please
 * read the section 'Credit' in the README file.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or          10
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA      20
 */

/*
 * Example: Correlation between s22th13 and deltacp
 * Compile with ``make th13delta''
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>                                                              30
#include <string.h>

#include <globes/globes.h> /* GLoBES library */
#include "myio.h"          /* my input-output routines */

/* If filename given, write to file; for empty filename write to screen */
char MYFILE[]="test1.dat";

int main(int argc, char *argv[])
{                                                                              40
  /* Initialize libglobes */
  glbInit(argv[0]);

  /* Initialize experiment NFstandard.glb */
  glbInitExperiment("NFstandard.glb",&glb_experiment_list[0],&glb_num_of_exps);

  /* Intitialize output */
  InitOutput(MYFILE,"Format: Log(10,s22th13) deltacp chi^2 \n");

  /* Define standard oscillation parameters */                                 50
  double theta12 = asin(sqrt(0.8))/2;
  double theta13 = asin(sqrt(0.001))/2;
  double theta23 = M_PI/4;
  double deltacp = M_PI/2;
  double sdm = 7e-5;
  double ldm = 2e-3;

  /* Initialize parameter vector(s) */
  glb_params true_values = glbAllocParams();
  glb_params test_values = glbAllocParams();                                   60

  glbDefineParams(true_values,theta12,theta13,theta23,deltacp,sdm,ldm);
  glbSetDensityParams(true_values,1.0,GLB_ALL);
  glbDefineParams(test_values,theta12,theta13,theta23,deltacp,sdm,ldm);
  glbSetDensityParams(test_values,1.0,GLB_ALL);

  /* The simulated data are computed */
  glbSetOscillationParameters(true_values);
  glbSetRates();
                                                                               70
```

```
/* Iteration over all values to be computed */
double thetheta13,x,y,res;

for(x=0.0;x<15.0+0.01;x=x+2.0/50)
for(y=0.0;y<360.0+0.01;y=y+200.0/50)
{
    /* Set vector of test values */
    thetheta13=asin(sqrt(pow(10,x)))/2;
    glbSetOscParams(test_values,thetheta13,GLB_THETA_13);
    glbSetOscParams(test_values,y*M_PI/180.0,GLB_DELTA_CP);            80

    /* Compute Chi^2 for all loaded experiments and all rules */
    res=glbChiSys(test_values,GLB_ALL,GLB_ALL);
    AddToOutput(x,y,res);
}

/* Destroy parameter vector(s) */
glbFreeParams(true_values);
glbFreeParams(test_values);
                                                                        90
exit(0);
}
```

==

To compile we use following command:

- "make th13delta" (this is to compile the above program)
- "./th13delta" (this is to execute it)

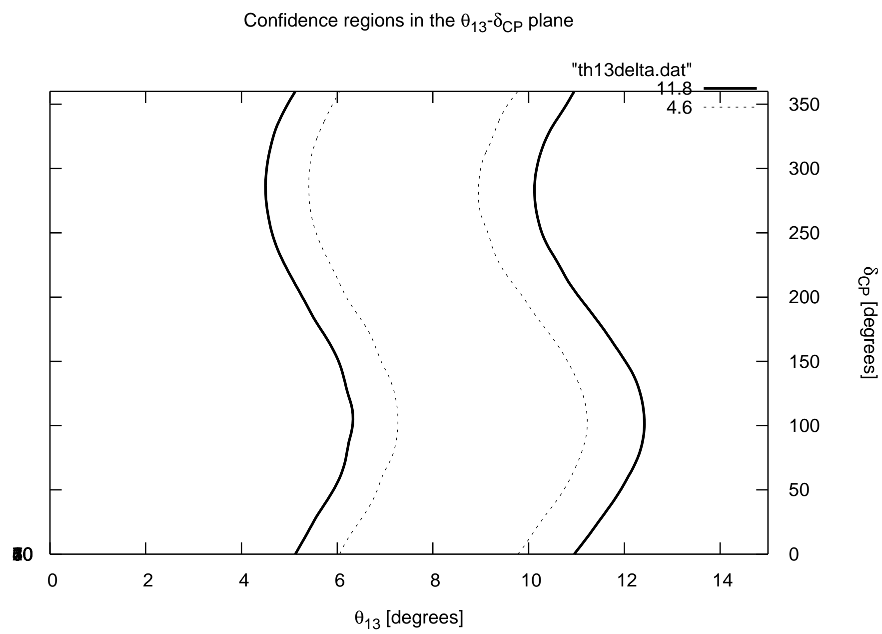
Before compiling it, we must modify our "make" file accordingly.

0.5 Plots and Results

Data obtained in ".dat" file is then plotted gnuplot. Following plot was obtained:

Result:

- Basics of GLoBES have been studied and its installation is understood.
- A program in C is written to understand the features of GLoBES and it was found that GLoBES is a nice tool to understand neutrino experiments and its physics.
- Correlation between θ_{13} and δ_{CP} was studied and understood.



(a) (1)

Figure 1: Correlation between θ_{13} and δ_{CP}