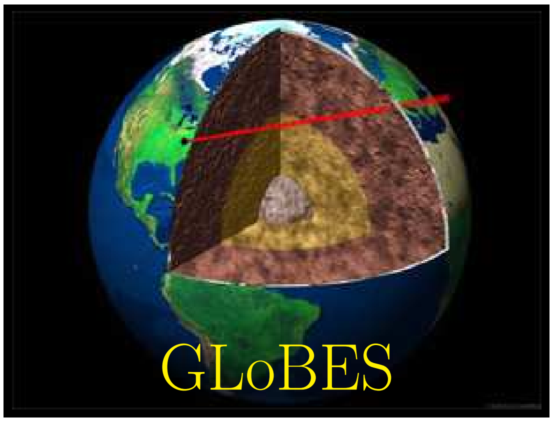
GLoBES

General Long Baseline Experiment Simulator

User's and experiment definition manual

Patrick Huber[a], Joachim Kopp[b], Manfred Lindner[b], Mark Rolinec[c], Walter Winter[d]



Version from March 29, 2007 for GLoBES 3.0

[a] *University of Wisconsin, Physics Department, 1150 University Av., Madison, WI 53706, USA*
[b] *Max–Planck–Institut für Kernphysik, Postfach 10 39 80, D–69029 Heidelberg, Germany*
[c] *Technische Universität München, Institut für Theoretische Physik, Physik–Department, James–Franck–Strasse, D–85748 Garching, Germany*
[d] *Universität Würzburg, Lehrstuhl für theoretische Physik II, Institut für theoretische Physik und Astrophysik, Am Hubland, D-97074 Würzburg, Germany*

What is GLoBES?

GLoBES ("General Long Baseline Experiment Simulator") is a flexible software package to simulate neutrino oscillation long baseline and reactor experiments. On the one hand, it contains a comprehensive abstract experiment definition language (AEDL), which allows to describe most classes of long baseline experiments at an abstract level. On the other hand, it provides a C-library to process the experiment information in order to obtain oscillation probabilities, rate vectors, and $\Delta\chi^2$-values. Currently, GLoBES is available for GNU/Linux. Since the source code is included, the port to other operating systems is in principle possible. The software as well as up-to-date versions of this manual can be found at this URL: `http://www.mpi-hd.mpg.de/~globes`

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 supernovæ, 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. Furthermore, experiments with discrete numbers of sources and detectors can be implemented by user-defined systematics in GLoBES 3.0 and higher.

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 $\Delta\chi^2$ 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 $\Delta\chi^2$, 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 $\sin^2 2\theta_{13}$-axis and the $\sin^2 2\theta_{13}$-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.

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 you use GLoBES to produce a publication or a talk indicate that you 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,

but *not* this manual. This manual itself is not a scientific publication and will not be submitted to a scientific journal. It will evolve during time since it is intended for regular revision. Besides that, many of the data which are used by GLoBES and distributed together with it should be properly referenced. For details see below.

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 wouldn't 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 user's 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. [3, 4].

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 (*cf.*, Chapter 12). It is recommended that you use the same style for your own input files, since then, in case they are distributed, everybody will know how to correctly reference your work.

What is new in GLoBES?

Here we briefly summarize the main changes of the new GLoBES version. For details, please refer to the respective parts of the manual. Please note that any new GLoBES version is compatible with older versions, *i.e.*, old application software and AEDL files should, with minor modifications, run with the new version as well. However, some function names and features will evolve during time, which means that outdated features may not be documented anymore.

Version 3.0

Here comes a summary of the most important changes in this version for users of earlier versions of GLoBES.

New features

- User-defined systematics, which can be used to simulate reactor experiments *etc.*; see Secs. 3.2 and 11.6
- User-defined priors to include arbitrary external information in the χ^2 before marginalization over the oscillation parameters; see Sec. 4.5
- Non-standard physics support; see Chapter 8
- Beta beam fluxes available as built-in fluxes; see Sec. 11.1
- Enhanced support for parallelization, such as Condor; see, *e.g.*, page 118
- Updated AEDL files; see Table 2.1
- New AEDL features, such as the support of lists as variables and an interpolation routine; see Sec. 10.4
- Clean-up of inconsistencies, such as an overall (internal) normalization factor in the flux files; see, *e.g.*, page 139
- Faster probability engine, easier installation (internal changes)

- Experimental feature: Alternative minimizer provided, which is usually faster than the standard minimizer; see Chapter 9

Major changes

Most of the modifications should not require that old pieces of software be changed. However, the following changes could be relevant:

- `glbSetDensityParams` has to be used with `glbDefineParams` and `glbSetDensityProjectionFlag` together with `glbDefineProjection`, because unexpected pre-defined behavior should be avoided.

- Functions `glbSetFilter`, `glbGetFilter`, `glbSetFilterState`, and `glbGetFilterState` replaced by functions `...InExperiment`.

- AEDL requires now that `$version` be used to define the minimum version number this AEDL file is to be used with. With this requirement one can easily avoid that new AEDL files with new features be used with old versions of GLoBES which may not recognize these features.

- Some of the earlier AEDL files have been updated, changed names, or have been removed. In addition, new files have been added. Although old AEDL files will run as usual for compatibility, they will not be supported by the GLoBES Team anymore. You should make sure to keep these files when updating GLoBES.

- The implementation of the tilt (systematics) has slightly changed. The tilt also works for variable bin widths. Therefore, you will obtain slightly different results when you run the same AEDL between older and newer versions of GLoBES.

Minor changes

Here we document the most important changes which should not affect older software:

- Functions and constants renamed for consistency:
  - `glbChiTheta` → `glbChiTheta13`
  - `glbChiThetaDelta` → `glbChiTheta13Delta`
  - `glbChiDms` → `glbChiDm21`
  - `glbChiDm` → `glbChiDm31`
  - `GLB_DM_SOL` → `GLB_DM_21`
  - `GLB_DM_ATM` → `GLB_DM_31`
  - `glbSetStartingValues` → `glbSetCentralValues`
  - `glbGetStartingValues` → `glbGetCentralValues`

  - `glbGetProfileData` → `glbGetProfileDataInExperiment`

- Rate access changed; see Sec. 6.3
- Systematics concept changed, concept of error dimensions removed; `@backgroundcenter` removed, central values for all systematics parameters now zero; see Sec. 11.6

Contents

How to use this manual

As it is illustrated in Fig. 1, GLoBES consists of several modules.

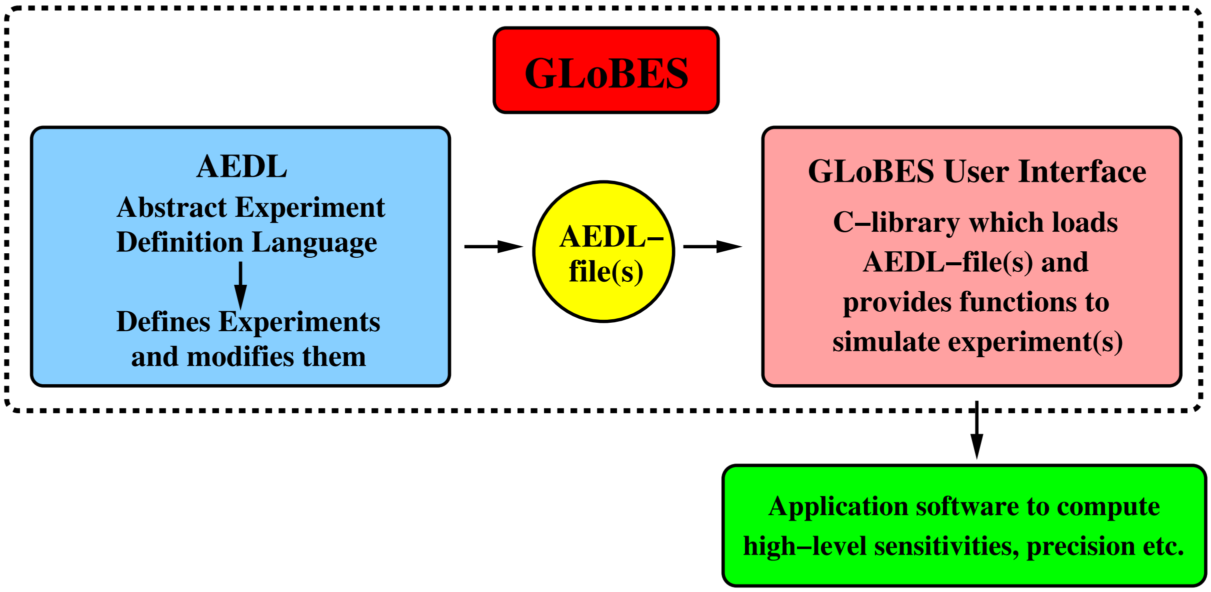


Figure 1: Different modules in GLoBES.

AEDL("Abstract Experiment Definition Language") is a language to define experiments in the form of ordinary text files. One or more of the resulting AEDL files can then be processed together with supporting flux or cross section files by the user interface. The user interface is a C-library, which loads one or more AEDL file(s) containing the experiment definition(s). The user interface is linked against the application software, and provides the user interface functions for the intended experiment simulation.

The application software is, except for some example files, not part of GLoBES, since the evaluation of the experiment performance is often a matter of taste and definition. In addition, the algorithms depend, especially for high-precision instruments, very much on the oscillation parameters. In general, it is quite simple to simulate superbeams and reactor experiments. However, because of the more complicated topology, the simulation of neutrino factories is much more difficult. In order to demonstrate some of these difficulties, we present in this manual mainly examples with neutrino factories. These examples can be found in Part I within the boxed pages. As complete files, they are also available in the GLoBES software package.

The GLoBES software may have two target groups: Physicists, who are mainly interested in optimizing the potential of specific experimental setups, and others, who are mainly interested in the physics potential of different experiment types from a theoretical point of view. For the first group, AEDL could be the most interesting aspect of GLoBES, where the user interface is only a tool to obtain specific parameter sensitivities. In this case, GLoBES could serve as a unified tool for the comparison and optimization of different experiment setups on equal footing, where it is the primary objective to simulate the experiments as accurate as possible. In addition, changes in experimental parameters, such as efficiencies or energy resolutions, can quickly be tested. For the second user group, the pre-defined experiment definition files might already be sufficient to test new conceptual approaches, and the user interface is the most interesting aspect for sophisticated applications including correlations, degeneracies, and multi-experiment setups. In either case, the GLoBES software could serve as a platform for the exchange of experiment definitions, and for an efficient splitting of work between experimentalists and theorists.

The user interface functions are described in Part I of this manual, which is the "user's manual". In there, first of all a short GLoBES tour is given in Chapter 1 in order to have an overview over GLoBES. After that, the user interface is successively introduced from very basic to more sophisticated functions. Eventually, it is demonstrated how one can change many experiment parameters at running time (such as baseline or target mass), and how one can obtain low-level information. We recommend that everybody interested in GLoBES should become familiar at least with the concepts in Chapter 1 and some of the examples on the boxed pages. The examples can be directly compiled from the respective directory in the GLoBES software package. The corresponding figures are produced with the Mathematica Notebook `DocPlots.nb`, which can be found in the example directory as well.

In Part II of the manual, AEDL is described. After an introductory chapter, all functions are defined in greater detail. This part might be more interesting for the experimental users who want to modify or create AEDL files. A useful tool in this context is the executable program `globes`, which returns event rates and other information for individual AEDL files without further programming. For example, flux normalizations can with this tool be easily adjusted to reproduce the event rates of a specific experiment. It is described in the last chapter of Part II.

In this version of the manual, introductory topics and advanced topics are mixed if they belong to the same subject. Therefore, we have marked more advanced material by a star (*). This material can be skipped in a first reading of the manual. In some cases, it may be even recommendable to do so because knowledge of AEDL is required (which is introduced in the second part).

Part I

User's manual

Chapter 1

A GLoBES tour

In this first chapter, we show a GLoBES tour illustrating the main features of GLoBES. The complete example can be found as `example-tour.c` in the `example` subdirectory of your GLoBES distribution. The output is written to `stream`, which can be either `stdout`, or a file.[1] Details about how to use GLoBES with C can found in Chapter 2 and the following chapters. You can also find a summary of the most important GLoBES χ^2-functions in Table 1.1. Note that this chapter can be skipped without loss of relevant information.

Initialize the GLoBES library:

```
glbInit(argv[0]);
```

Define my standard oscillation parameters:

```
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;
```

Load one neutrino factory experiment:

```
glbInitExperiment("NFstandard.glb",&glb_experiment_list[0],
                  &glb_num_of_exps);
```

Initialize a number of parameter vectors we are going to use later:

```
glb_params true_values = glbAllocParams();
glb_params fit_values = glbAllocParams();
glb_params central_values = glbAllocParams();
glb_params input_errors = glbAllocParams();
glb_params minimum = glbAllocParams();
```

[1] Note that the output in this section can be slightly different from yours depending on the current version of the probability engine, systematics implementation, and AEDL file used.

Function	Purpose	Parameters → Result
Systematics only:		
`glbChiSys`	χ^2 with systematics only	`(glb_params in, int exp, int rule)` → `double` χ^2
Projections onto axes:		
`glbChiTheta13`	Projection onto θ_{13}-axis	`(glb_params in, glb_params out, int exp)` → `double` χ^2
`glbChiDelta`	Projection onto δ_{CP}-axis	`(glb_params in, glb_params out, int exp)` → `double` χ^2
`glbChiTheta23`	Projection onto θ_{23}-axis	`(glb_params in, glb_params out, int exp)` → `double` χ^2
`glbChiDm31`	Projection onto Δm^2_{31}-axis	`(glb_params in, glb_params out, int exp)` → `double` χ^2
`glbChiDm21`	Projection onto Δm^2_{21}-axis	`(glb_params in, glb_params out, int exp)` → `double` χ^2
Projection onto plane:		
`glbChiTheta13Delta`	Projection onto θ_{13}-δ_{CP}-plane	`(glb_params in, glb_params out, int exp)` → `double` χ^2
Projection onto any hyper-plane:		
`glbChiNP`	Projection onto any n-dimensional hyper-plane	`(glb_params in, glb_params out, int exp)` → `double` χ^2 Needs `glbSetProjection` before!
Localization of degeneracies:		
`glbChiAll`	(Local) Minimization over all parameters	`(glb_params in, glb_params out, int exp)` → `double` χ^2

Table 1.1: The GLoBES standard function to obtain a χ^2-value with systematics only or systematics and correlations. The parameters `rule` and `exp` can either be `GLB_ALL` for all initialized experiment or the experiment number (0 to `glb_num_of_exps-1`) for a specific experiment. The format of `glb_params` is discussed in detail in Chapter 2. Note that all functions but `glbChiSys` are using minimizers which have to be initialized with `glbSetInputErrors` and `glbSetCentralValues` first.

Assign values to our standard oscillation parameters and the standard matter density scaling factors:

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

Compute the simulated data with our standard parameters:

```
glbSetOscillationParameters(true_values);
glbSetRates();
```

Return the oscillation probabilities in vacuum and matter for the electron neutrino as initial flavor:

```
int i;
fprintf(stream,"\nOscillation probabilities in vacuum:  ");
for(i=1;i<4;i++) fprintf(stream,"1->%i:  %g",i,
                    glbVacuumProbability(1,i,+1,50,3000));
fprintf(stream,"\nOscillation probabilities in matter:  ");
for(i=1;i<4;i++) fprintf(stream,"1->%i:  %g ",i,
                    glbProfileProbability(0,1,i,+1,50));
```

→ Output:

Oscillation probabilities in vacuum: 1->1: 0.999955 1->2: 2.58628e-05 1->3: 1.92142e-05
Oscillation probabilities in matter: 1->1: 0.999965 1->2: 2.01364e-05 1->3: 1.49644e-05

Now assign fit values, where we will test the fit value $\sin^2 2\theta_{13} = 0.0015$:

```
glbCopyParams(true_values,fit_values);
glbSetOscParams(fit_values,asin(sqrt(0.0015))/2,GLB_THETA_13);
```

Compute χ^2 with systematics only for all experiments and rules:

```
chi2 = glbChiSys(fit_values,GLB_ALL,GLB_ALL);
fprintf(stream,"chi2 with systematics only:  %g\n\n",chi2);
```

→ Output:

chi2 with systematics only: 22.433

This we would obtain from the first appearance channel only:

```
chi2 = glbChiSys(fit_values,0,0);
fprintf(stream,"This we would have from the CP-even appearance
channel only:  %g\n\n",chi2);
```

→ Output:

This we would have from the CP-even appearance channel only: 21.1569

The sum over all rules again gives:

```
chi2 = glbChiSys(fit_values,GLB_ALL,0)+ glbChiSys(fit_values,GLB_ALL,1)+
       glbChiSys(fit_values,GLB_ALL,2)+ glbChiSys(fit_values,GLB_ALL,3);
fprintf(stream,"The sum over all rules gives again:  %g\n\n",chi2);
```

→ Output:

The sum over all rules gives again: 22.433

Let's prepare the minimizers for taking into account correlations. Set errors for external parameters, too: 10% for each of the solar parameters, and 5% for the matter density.

```
glbDefineParams(input_errors,theta12*0.1,0,0,0,sdm*0.1,0);
glbSetDensityParams(input_errors,0.05,GLB_ALL);
glbSetCentralValues(true_values);
glbSetInputErrors(input_errors);
```

Then we can calculate χ^2 including the full multi-parameter correlation, and show where GLoBES actually found the minimum (note that this takes somewhat longer than systematics only). This corresponds to a projection onto the $\sin^2 2\theta_{13}$-axis:

```
chi2 = glbChiTheta13(fit_values,minimum,GLB_ALL);
fprintf(stream,"chi2 with correlations:  %g \n",chi2);
fprintf(stream,"Position of minimum:  theta12, theta13, theta23,
   delta, sdm, ldm, rho\n");
glbPrintParams(stream,minimum);
fprintf(stream,"Note that s22theta13 is unchanged/kept fixed:
   %g!  \n\n", pow(sin(2*glbGetOscParams(minimum,GLB_THETA_13)),2));
```

→ Output:

chi2 with correlations: 1.99794
Position of minimum: theta12,theta13,theta23,delta,sdm,ldm,rho
0.541226 0.0193698 0.746156 1.74968 6.64399e-05 0.00200514
1.00341
Iterations: 1988
Note that s22theta13 is unchanged/kept fixed: 0.0015!

Instead of including the full correlation, we can take the correlation with every parameter except for δ_{CP}, *i.e.*, we keep (in addition to θ_{13}) δ_{CP} fixed. This corresponds to a projection onto the $\sin^2 2\theta_{13}$-δ_{CP}-plane:

```
chi2 = glbChiTheta13Delta(fit_values,minimum,GLB_ALL);
fprintf(stream,"chi2 with correlations other than with deltacp:
%g \n\n",chi2);
```

→ Output:

chi2 with correlations other than with deltacp: 4.02974

Similarly, we can only take into account the correlation with δ_{CP}. For this, we need to define our own (user-defined) projection, where only δ_{CP} (and the matter density) is a free parameter:

```
glb_projection myprojection = glbAllocProjection();
glbDefineProjection(myprojection,GLB_FIXED, GLB_FIXED, GLB_FIXED,
   GLB_FREE, GLB_FIXED, GLB_FIXED);
glbSetDensityProjectionFlag(myprojection,GLB_FREE,GLB_ALL);
glbSetProjection(myprojection);
chi2 = glbChiNP(fit_values,minimum,GLB_ALL);
fprintf(stream,"chi2 with correlation only with deltacp:
   %g \n\n",chi2);
glbFreeProjection(myprojection);
```

→ Output:

chi2 with correlation only with deltacp: 2.72943

We can also switch of the systematics and compute the statistics χ^2 only:

```
glbSwitchSystematics(GLB_ALL,GLB_ALL,GLB_OFF);
chi2 = glbChiSys(fit_values,GLB_ALL,GLB_ALL);
glbSwitchSystematics(GLB_ALL,GLB_ALL,GLB_ON);
fprintf(stream,"chi2 with statistics only:
   %g\n\n",chi2);
```

→ Output:

chi2 with statistics only: 37.9736

Let us now locate the exact position[2] of the sgn-degeneracy:

```
glbDefineParams(input_errors,theta12*0.1,0,0,0,sdm*0.1,ldm/3);
glbDefineParams(central_values,theta12,theta13,theta23,
   deltacp,sdm,-ldm);
glbSetDensityParams(input_errors,0.05,GLB_ALL);
glbSetDensityParams(starting_errors,1.0,GLB_ALL);
glbSetCentralValues(central_values);
```

[2]For an exact definition of inverted hierarchy, see page 47.

```
glbSetInputErrors(input_errors);
chi2=glbChiAll(central_values,minimum,GLB_ALL);
fprintf(stream,"chi2 at minimum:  %g \n",chi2);
fprintf(stream,"Position of minimum:
   theta12,theta13,theta23,delta,sdm,ldm,rho\n");
glbPrintParams(stream,minimum);
```

→ Output:

```
chi2 at minimum: 4.2137
Position of minimum: theta12,theta13,theta23,delta,sdm,ldm,rho
0.585006 0.0275073 0.725679 1.05251 7.80256e-05 -0.00191273
0.979049
Iterations: 3184
```

After testing these functions with only one experiment, let us now go to a two-experiment setup with two different neutrino factory baselines. Since the GLoBES parameter vectors depend on the number of experiments, we have to free them first:

```
glbFreeParams(true_values);
glbFreeParams(fit_values);
glbFreeParams(central_values);
glbFreeParams(input_errors);
glbFreeParams(minimum);
```

Then we clear the experiment list and load the new experiments:

```
fprintf(stream,"\nNOW: TWO-EXPERIMENT SETUP
   NuFact at 3000km+NuFact at 7500km\n\n");

glbClearExperimentList();

glbInitExperiment("NFstandard.glb",&glb_experiment_list[0],
   &glb_num_of_exps);
glbInitExperiment("NFstandard.glb",&glb_experiment_list[0],
   &glb_num_of_exps);
```

→ Output:

```
NOW: TWO-EXPERIMENT SETUP NuFact at 3000km+NuFact at 7500km
```

Then we need to change the baseline of the second experiment, where we set the density to the average density of this baseline:

```
double* lengths;
double* densities;
```

```
glbAverageDensityProfile(7500,&lengths,&densities);
fprintf(stream,"Magic baseline length:  %g,
   Density:  %g\n\n",lengths[0],densities[0]);
glbSetProfileDataInExperiment(1,1,lengths,densities);
free(lengths);
free(densities);
```

→ Output:

Magic baseline length: 7500, Density: 4.25286

Now we can re-initialize our parameter vectors again:

```
true_values = glbAllocParams();
fit_values = glbAllocParams();
central_values = glbAllocParams();
input_errors = glbAllocParams();
minimum = glbAllocParams();
```

In addition, we repeat the procedure for the simulated rates and the fit parameter vector:

```
glbDefineParams(true_values,theta12,theta13,theta23,deltacp,sdm,ldm);
glbSetDensityParams(true_values,1.0,GLB_ALL);
glbSetOscillationParameters(true_values);
glbSetRates();

glbCopyParams(true_values,fit_values);
glbSetOscParams(fit_values,asin(sqrt(0.0015))/2,GLB_THETA_13);
```

Here comes the χ^2 with systematics only for all experiments and rules:

```
chi2 = glbChiSys(fit_values,GLB_ALL,GLB_ALL);
fprintf(stream,"chi2 with systematics for all exps:
   %g\n",chi2);
```

→ Output:

chi2 with systematics for all exps: 30.5678

Compute χ^2 for each experiment and compute the sum:

```
chi2 = glbChiSys(fit_values,0,GLB_ALL);
fprintf(stream,"chi2 with systematics for 3000km:  %g\n",chi2);
chi2b = glbChiSys(fit_values,1,GLB_ALL);
fprintf(stream,"chi2 with systematics for 7500km:  %g\n",chi2b);
fprintf(stream,"The two add again to:
   %g\n\n",chi2+chi2b);
```

→ Output:

```
chi2 with systematics for 3000km: 22.433
chi2 with systematics for 7500km: 8.1348
The two add again to: 31.0797
```

Similarly, compute the χ^2 with correlations for each experiment and their combination. Compare it to the χ^2 for all experiments: the sum of the individual results is not equal to the χ^2 of the combination anymore. Note that there are now two densities in the output vectors.

```
glbDefineParams(input_errors,theta12*0.1,0,0,0,sdm*0.1,0);
glbSetDensityParams(input_errors,0.05,GLB_ALL);
glbSetCentralValues(true_values);
glbSetInputErrors(input_errors);
chi2 = glbChiTheta13(fit_values,minimum,0);
fprintf(stream,"chi2 with correlations for 3000km:  %g \n",chi2);
glbPrintParams(stream,minimum);
chi2b = glbChiTheta13(fit_values,minimum,1);
fprintf(stream,"\nchi2 with correlations for 7500km:
   %g \n",chi2b);
glbPrintParams(stream,minimum);
chi2sum = glbChiTheta13(fit_values,minimum,GLB_ALL);
fprintf(stream,"\nchi2 with correlations for combination:
   %g \n",chi2sum);
glbPrintParams(stream,minimum);
fprintf(stream,"\nThe sum of the two chi2s is %g,
   whereas the total chi2 is %g !\n\n",chi2+chi2b,chi2sum);
```

→ Output:

```
chi2 with correlations for 3000km: 1.99794
0.541226 0.0193698 0.746156 1.74968 6.64399e-05 0.00200514
1.00341 1
Iterations: 1988

chi2 with correlations for 7500km: 0.803787
0.556503 0.0193698 0.771812 4.75971 7.00813e-05 0.0020012
1 1.01201
Iterations: 798

chi2 with correlations for combination: 3.68133
0.543335 0.0193698 0.77004 1.75576 6.59505e-05 0.00199913
1.00333 1.03269
Iterations: 2187
```

The sum of the two chi2s is 2.80173, whereas the total chi2 is 3.68133!

Now find the $\mathrm{sgn}(\Delta m^2_{31})$-degeneracy for the neutrino factory at the short baseline and test if it is still present in the combination of the two baselines:

```
glbDefineParams(input_errors,theta12*0.1,theta13,theta23,
   deltacp,sdm*0.1,ldm/3);
glbDefineParams(central_values,theta12,theta13,theta23,
   deltacp,sdm,-ldm);
glbSetDensityParams(input_errors,0.05,GLB_ALL);
glbSetDensityParams(central_values,1.0,GLB_ALL);
glbSetCentralValues(central_values);
glbSetInputErrors(input_errors);

chi2=glbChiAll(central_values,minimum,0);
fprintf(stream,"chi2 at minimum, L=3000km:  %g \n",chi2);
glbPrintParams(stream,minimum);

chi2=glbChiAll(minimum,minimum,GLB_ALL);
fprintf(stream,"\nchi2 for combination at minimum of L=3000km:
   %g \n",chi2);
glbPrintParams(stream,minimum);
```

→ Output:

chi2 at minimum, L=3000km: 4.82021
0.584568 0.026488 0.727714 1.09015 7.78747e-05 -0.00191163
0.978867 1
Iterations: 3298

chi2 for combination at minimum of L=3000km: 59.4879
0.599908 0.0172467 0.768342 1.39186 8.198e-05 -0.00189508
0.970733 1.17125
Iterations: 2327

Finally, we have to free the parameter vectors again:

```
glbFreeParams(true_values);
glbFreeParams(fit_values);
glbFreeParams(central_values);
glbFreeParams(input_errors);
glbFreeParams(minimum);
```

Chapter 2

GLoBES basics

In this first chapter of the user's manual, we assume that the GLoBES software is readily installed on your computer system. For the installation, see Appendix A and the `INSTALL` file in the software package. We demonstrate how to load pre-defined experiments and introduce the basic concepts of GLoBES. We do not go into details of the programming language, which means that standard parts of the program code common to all of the examples in the following chapters are, in general, omitted. An example of a minimal GLoBES program in C can be found on page 14. Furthermore, the files of the examples in this part can be found in the `example` subdirectory of your GLoBES distribution. After the installation of GLoBES, they 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 you've just to type `make` and you're done.[1] This `Makefile` very well serves as a template for your own applications.

We will in this part not go into details of the experiment definition. The pre-defined experiment prototypes in the `data` subdirectory are summarized in Table 2.1 and described in Appendix B. They correspond (except from minor modifications) to the experiments in the respective references in the table. These files are installed to the directory `${prefix}/share/globes` which usually defaults to `/usr/local/share/globes`. It is useful to add this path to the value of `GLB_PATH`.

2.1 Initialization 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 14).*

[1] The data files (AEDL and supporting files) needed by the examples are already in place.

Example: Using GLoBES *with C*

Here comes the C-code skeleton, which is (more or less) common to all of our GLoBES examples:

```
#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 */

/* 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 */

   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) */
   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


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


   /* ...  CODE ...  */


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


   exit(0);
}
```

Experiment	Filename	Short description	Ref.
Superbeam experiments:			
T2K	`T2K.glb`	J-PARC to Super-K, 2 yr ν- and 6 yr $\bar{\nu}$-running	[5, 6]
T2HK	`T2HK.glb`	J-PARC to Hyper-K, 4 yr ν- and 4 yr $\bar{\nu}$-running (T2K upgrade)	[5, 6]
NOνA	`NOvA.glb`	FermiLab NuMI beamline off-axis, 3 yr ν- and 3 yr $\bar{\nu}$-running	[7, 8]
SPL	`SPL.glb`	CERN to Fréjus, 2 yr ν- and 8 yr $\bar{\nu}$-running	[9–11]
Reactor experiments:			
Reactor-I	`Reactor1.glb`	Small reactor exp., $\mathcal{L} = 400\,\mathrm{t\,GW\,yr}$	[12]
Reactor-II	`Reactor2.glb`	Large reactor exp., $\mathcal{L} = 8\,000\,\mathrm{t\,GW\,yr}$	[12]
DoubleChooz	`D-Chooz_near.glb`	Double Chooz near detector, 5 years data taking	[13]
	`D-Chooz_far.glb`	Double Chooz far detector, 5 years data taking	
β-Beams:			
Low γ	`BB_100.glb`	$\gamma = 100$ CERN to Fréjus baseline scenario, 4 yr ν- and 4 yr $\bar{\nu}$-running	[9]
Medium γ	`BB_350.glb`	$\gamma = 350$, refurbished SPS scenario, 4 yr ν- and 4 yr $\bar{\nu}$-running	[14]
Variable γ	`BBvar_WC.glb`	Variable β-beam, 4 yr ν- and 4 yr $\bar{\nu}$-running, AEDL-Variables `gammafactor`, `EXP_FACTOR` and `baselinefactor`, 500 kt WC detector	[15]
	`BBvar_TASD.glb`	As `BBvar_WC.glb`, but with a 50 kt TASD	[15]
Neutrino factories:			
Standard	`NFstandard.glb`	Standard neutrino factory, 4 yr ν- and 4 yr $\bar{\nu}$-running	[6]
Variable E_μ	`NFvar.glb`	Variable E_μ neutrino factory, 4 yr ν- and 4 yr $\bar{\nu}$-running, AEDL-Variables `emax` and `BASELINE`, disapp. channels *without* CID	[6, 16]
Gold + Silver	`NF_GoldSilver.glb`	As `NFvar.glb`, plus 5 kt ECC detector for Silver Channel measurement	[6, 16, 17]
Hybrid det.	`NF_hR_lT.glb`	As `NFvar.glb`, but lower threshold and higher energy resolution	[6, 16]

Table 2.1: Pre-defined experiment prototypes, their filenames (to be used in `glbInitExperiment`), their short descriptions, and the references in which they are originally used and discussed (except from minor modifications, such as a different implementation of the energy threshold function). Note that some of these experiments may want to be adjusted in terms of integrated luminosity, baseline, flux, efficiencies, or other factors. In any case these file are installed along with GLoBES. For more details, see Appendix B.

In principle, the GLoBES user interface can currently handle up to 32 of different long-baseline 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.[2] 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 currently 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`.

A list of pre-defined experiment prototypes, their filenames, their short descriptions, and the references of their definitions can be found in Table 2.1; for more details, see Appendix B. If the program cannot find these files, or some of them are syntactically not correct, it will break at this place.

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.*

[2]Note that the global variable `glb_num_of_exps` must not be modified by the user.

Quantities	Examples	Units
Angles	θ_{13}, θ_{12}, θ_{23}, δ_{CP}	Radians
Mass squared differences	Δm_{21}^2, Δm_{31}^2	eV^2
Matter densities	ρ_i	$\mathrm{g/cm}^3$
Baseline lengths	L_i	km
Energies	E_ν	GeV
Fiducial masses	m_{Det}	t (reactor exp.) or kt (accelerator exp.), depends on experiment definition
Time intervals	t_{run}	yr
Source powers	P_{Source}	Useful parent particle decays/yr (Neutrino factory, β-Beam), GW thermal power (reactor exps.), or MW target power (superbeams); depends on flux definition
Cross sections/E	σ_{CC}/E	$10^{-38}\,\mathrm{cm}^2/\mathrm{GeV}^2$

Table 2.2: Quantities used in GLoBES, examples of these quantities, and their standard units in the application software.

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.

2.2 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. Table 2.2 summarizes the units of the most important quantities. In principle, the event rates are proportional to the product of source power $\times$ target mass $\times$ 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 100% of 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). Since the pre-defined experiments in Table 2.1 are given for specific luminosities, it is useful to read out and change these parameters of the individual 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 definition 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 detector 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.

2.3 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 $(\theta_{12}, \theta_{13}, \theta_{23}, \delta_{\mathrm{CP}}, \Delta m_{21}^2, \Delta m_{31}^2)$ 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 $\hat{\rho}_i$, 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.*, $\hat{\rho}_i \equiv \rho_i/\bar{\rho}_i$. 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 (*cf.*, Chapter 4) 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)` *assigns 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)` *returns the value of the oscillation parameter* `which` *in the structure* `in`*.*

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}, Δm^2_{21}, Δm^2_{31}. Alternatively to the number, the constants `GLB_THETA_12`, `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)` *returns the value of the density parameter* `which` *in the structure* `in`*.*

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 */

glbFreeParams(vector1);
/* ... more vectors freed ... */

/* ... end of program or glbClearExperimentList ... */
```

2.4 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 vacuum 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`*.*

A complete example for a minimal GLoBES program can be found on Page 14.

2.5 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.

Chapter 3

Calculating χ^2 with systematics only

Calculating a χ^2-value including systematics, but without correlations and degeneracies, is the simplest and fastest possibility to obtain high-level information on an experiment. Here we describe the use of built-in systematics, and the rather advanced topic of user-defined systematics. Note that the matter density is treated as an oscillation parameter in GLoBES, which means that it is not dealt with at the systematics level.

3.1 Built-in systematics

Keeping all oscillation parameters and matter density scaling factors fixed, one can use the following builtin functions to obtain the total χ^2 of all specified oscillation channels including systematics:

Function 3.1 `double glbChiSys(const glb_params in,int exp, int rule)` *returns the* χ^2 *for the (fixed) oscillation parameters* `in`*, the experiment number* `exp`*, and the rule number* `rule`*. For all experiments or rules, use* `GLB_ALL` *as parameter value.*

Note that the result of `glbChiSys` for all experiments or rules corresponds to the sum of all of the individual `glbChiSys` calls. This equality will not hold for the minimizers in the next chapters anymore. An example how to use `glbChiSys` can be found on page 24.

The treatment of systematics in GLoBES is performed by the so-called *pull method* with the help of nuisance systematics parameters. They are taken to be completely uncorrelated among different rules, and treated with simple Gaußian statistics. In general, a rule is a prescription for summing up experimentally indistinguishable signal and background events from different oscillation channels. For more details on the rule concept, see Part II of this manual, and for the treatment of systematics, see Sec. 11.6.

One example for a systematics parameter is the signal normalization error, *i.e.*, an error on the overall normalization of the signal. For illustration, we assume that the signal event rate in the ith bin s_i^0 of one oscillation channel is altered by the overall normalization nuisance parameter of this channel, *i.e.*,

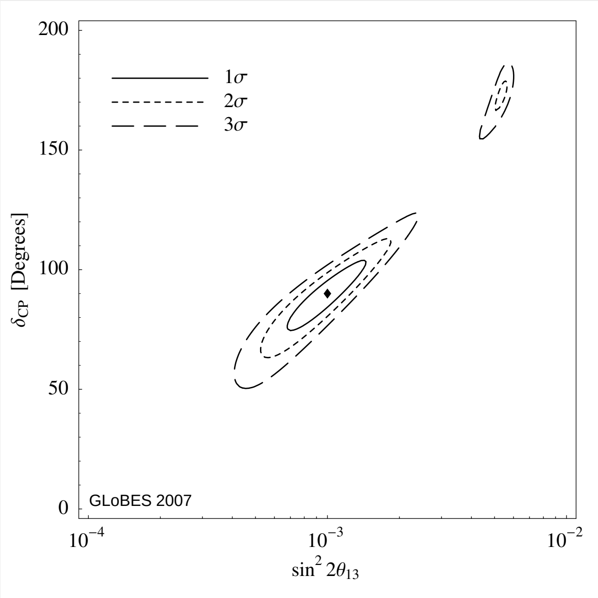
$$s_i = s_i(n_s) = s_i^0 \cdot (1 + n_s), \tag{3.1}$$

Example: Correlation between $\sin^2 2\theta_{13}$ *and* $\delta_{\rm CP}$

A typical and fast application for `glbChiSys` is the visualization of two-parameter correlations using systematics only. For example, to calculate the two-parameter correlation between $\sin^2 2\theta_{13}$ and $\delta_{\rm CP}$ at a neutrino factory, one can use the following code excerpt from `example1.c`:

```
/* Initialize parameter vector(s) and compute simulated data */
glbDefineParams(true_values,theta12,theta13,theta23,deltacp,sdm,ldm);
glbSetDensityParams(true_values,1.0,GLB_ALL);
glbCopyParams(true_values,test_values);
glbSetOscillationParameters(true_values); glbSetRates();

/* Iteration over all values to be computed */
for(x=-4.0;x<-2.0+0.01;x=x+2.0/50)
for(y=0.0;y<200.0+0.01;y=y+200.0/50)
{
    /* Set parameters in vector of test values */
    glbSetOscParams(test_values,asin(sqrt(pow(10,x)))/2,GLB_THETA_13);
    glbSetOscParams(test_values,y*M_PI/180.0,GLB_DELTA_CP);

    /* Compute Chi2 for all loaded experiments and all rules */
    res=glbChiSys(test_values,GLB_ALL,GLB_ALL);
    AddToOutput(x,y,res);
}
```

The resulting data can then be plotted as a contour plot (2 d.o.f.):



where n_s is the signal normalization parameter. The total number of events in the ith bin x_i also includes the background event rates b_i, *i.e.*, $x_i = s_i + b_i$, which may have their own systematics parameters. In order to implement an overall signal normalization error σ_{n_s}, the χ^2, which includes all event rates x_i of all bins, is minimized over the nuisance parameter n_s:

$$\hat{\chi}^2 = \min_{n_s} \left(\chi^2(n_s, \ldots) + \frac{(n_s)^2}{\sigma_{n_s}^2} \right). \tag{3.2}$$

This minimization is done independently for all nuisance parameters of the rule. The total χ^2 for the considered experiment is finally obtained by repeating this procedure for all rules and adding their χ^2-values. In general, the situation is more complicated because of the usage of many systematical errors. More details about systematics parameters and the definition of signal, background, and oscillation channels can be found in Sec. 11.6, too.

The systematics minimization of an experiment can be easily switched on and off with `glbSwitchSystematics`, *i.e.*, one can also compute the χ^2 with statistics only. In addition, several options for systematics are available, such as only using total event rates without spectral information. For details, we refer to Chapter 7.

3.2 User-defined systematics calculation*

For some experimental setups, the built-in systematics functions are not sufficient because they cannot handle systematical errors that are correlated between different parts of a multi-detector setup. Therefore GLoBES allows the user to override the default χ^2 calculation. The reader who wants to use this feature should be familiar with Secs. 6.3, 7.1, 11.4, and 11.6 of the manual. Note that this is an advanced topic, which requires GLoBES 3.0 or higher. It can be omitted in a first reading of the manuscript.

User-defined systematics is, compared to the built-in systematics, a compound between the AEDL definition and some source code defining the behavior. In the simplest case, the AEDL file defines user-defined systematics for a specific rule, gives it a name/identifier, and defines the systematical errors. The source code has then to match this AEDL definition – therefore, make sure to use only unique identifiers. Ones needs to register the user-defined systematics in the source code (typically after `glbInit`) with

Function 3.2 `int glbDefineChiFunction(glb_chi_function chi_func, int dim, const char *name, void* user_data)` *tells* GLoBES *to register the user-defined systematics χ^2 function* `chi_func` *identified by the string* `name` *with* `dim` *systematics parameters. The parameter* `user_data` *is an arbitrary pointer being transferred to* `chi_func`*. It can, for instance, be used to avoid global variables. Returns zero, if successful.*

The user-defined systematics function itself has to have the pre-defined type

Function 3.3 `double (*glb_chi_function)(int exp, int rule, int dim, double *params, double *errors, void* user_data)`*. In this function type,* `exp` *is the experiment number,* `rule` *is the rule number,* `dim` *is the number of systematics parameters,* *Example: Systematics for a reactor setup with near and far detector*

The following code fragment from the much more extensive `example5.c` calculates χ^2 for a reactor setup with two detectors including five different systematical errors: the flux normalization of the reactor (`x[0]`), the fiducial mass uncertainties of the far (`x[1]`) and near (`x[2]`) detector, and energy calibration errors for the far (`x[3]`) and near (`x[4]`) detectors. The calculation follows Eq. (3.3), but includes the energy calibration.

```
double likelihood(double true_rate, double fit_rate, double sqr_sigma)
{ if (sqr_sigma > 0) return square(true_rate - fit_rate) / sqr_sigma;
else return 0.0; }

double chiDCNorm(int exp, int rule, int np, double *x, double *errors,
   void* user_data)
{
   const EXP_FAR = 0; const EXP_NEAR = 1;
   int n_bins = glbGetNumberOfBins(EXP_FAR);
   double *true_rates_N = glbGetRuleRatePtr(EXP_NEAR, 0);
   double *true_rates_F = glbGetRuleRatePtr(EXP_FAR, 0);
   double signal_fit_rates_N[n_bins]; double signal_fit_rates_F[n_bins];
   double signal_norm_N, signal_norm_F;
   int ew_low, ew_high, i;
   double emin, emax, fit_rate; double chi2 = 0.0;

   /* Request simulated energy interval and analysis energy window */
   glbGetEminEmax(exp, &emin, &emax);
   glbGetEnergyWindowBins(exp, rule, &ew_low, &ew_high);

   /* Apply energy calibration error */
   glbShiftEnergyScale(x[3], glbGetSignalFitRatePtr(EXP_FAR, 0),
      signal_fit_rates_F, n_bins, emin, emax);
   glbShiftEnergyScale(x[4], glbGetSignalFitRatePtr(EXP_NEAR, 0),
      signal_fit_rates_N, n_bins, emin, emax);

   /* Loop over all bins in energy window */
   signal_norm_F = 1.0 + x[0] + x[1];
   signal_norm_N = 1.0 + x[0] + x[2];
   for (i=ew_low; i <= ew_high; i++)
   {
     /* Statistical part of chi2 for far detector */
     fit_rate = signal_norm_F * signal_fit_rates_F[i];
     chi2 += likelihood(true_rates_F[i], fit_rate, true_rates_F[i]);


     /* Statistical part of chi2 for near detector */
     fit_rate = signal_norm_N * signal_fit_rates_N[i];
     chi2 += likelihood(true_rates_N[i], fit_rate, true_rates_N[i]);
   }

   /* Systematical part of chi2 (= priors) */
   for (i=0; i < np; i++) chi2 += square(x[i]/errors[i]);

   return chi2;
}
```

`params` *is an array of the systematics nuisance parameters themselves, and* `errors` *is an array with the systematical errors. The parameter* `user_data` *is set as defined with* `glbDefineChiFunction`*. Note the the central values for* `params` *are* 0*, which means that* 0 *corresponds to the un-modified rates. The array indices run from* 0 *to* `dim`*-1.*

Note that this function will be called many times by the GLoBES minimizers. Therefore the function should be as efficient as possible. In addition, note that complicated systematics with many parameters may introduce complicated (unknown) topologies for the minimization, which means that the minimizers may end up in a local minimum instead of the global minimum. GLoBES provides the function `glbSetSysStartingValuesList` to change the starting values of the minimizer in the case of convergence problems (see below). A typical implementation may look like the following code, where "`chiDCNorm`" is the identifier assigned in the AEDL definition:

```
double chiDCNorm(int exp, int rule, int dim, double *x,
   double *errors, void* user_data)
{
   double chi2 = 0.0;
   int i;

   ...  /* Some code to calculate the chi2 */

   /* Here the systematics priors (penalties) are added:  */
   for(i=0;i<dim;i++) chi2 += (x[i]*x[i])/(errors[i]*errors[i]);
   return chi2;
}

int main(int argc,char*argv[])
{
   ...
   glbInit(agrv[0]);
   glbDefineChiFunction(&chiDCNorm,5,"chiDCNorm",NULL);
   ...
}
```

During running time, the systematics can be changed with `glbSetChiFunction` and `glbGetChiFunction` as described in Sec. 7.1. In addition, especially useful in the context of user-defined systematics, a pointer to the systematics function can be returned either by name, or by experiment and rule selection:

Function 3.4 `glb_chi_function glbGetChiFunctionPtr(const char *name)` *returns a pointer to the systematics* χ^2 *function with a specified name* `name`*.*

Function 3.5 `glb_chi_function glbGetChiFunctionPtrInExperiment(int exp, int rule, int on_off)` *returns a pointer to the systematics χ^2 function of experiment* `exp` *and rule* `rule`*. Systematics on or off can be accessed by* `on_off`*.*

The user-defined χ^2 function of type `glb_chi_function` may use a number of helper functions for for the χ^2 calculation. A very useful function to include energy calibration errors is

Function 3.6 `glbShiftEnergyScale(double b, double *rates_in, double *rates_out,int bins, double emin, double emax)` *shifts the energy scale in* `rates_in` *by the relative amount* `b` *and stores the result in* `rates_out`*. The parameters* `emin` *and* `emax` *denote the minimal and maximal energy as obtained with* `glbGetEminEmax`*. Requires constant energy bin widths!*

For details on energy calibration errors, see Sec. 11.6. In addition, in order to implement user-defined systematics, one usually needs low-level access to the event rate vectors using `glbGetRuleRatePtr`, `glbGetSignalFitRatePtr`, `glbGetBGFitRatePtr` and possibly other functions described in Sec. 6.3. Additionally, the functions `glbGetEminEmax`, `glbGetEnergyWindow`, `glbGetEnergyWindowBins`, and `glbGetNumberOfBins` from Sec. 6.2 are useful. We recommend that you familiarize yourself with these functions at this point.

Let us now discuss a simple example. Typical applications for user-defined systematics are reactor neutrino experiments, where systematics play a crucial role. For a setup with near and far detectors, the Gaussian formula for χ^2 is

$$\chi^2 = \sum_{i=1}^{\#\text{ of bins}} \sum_{d=N,F} \frac{(O_{d,i} - (1 + a_R + a_d)T_{d,i})^2}{O_{d,i}} + \frac{a_R^2}{\sigma_R^2} + \frac{a_N^2}{\sigma_N^2} + \frac{a_F^2}{\sigma_F^2} \tag{3.3}$$

where $O_{N,i}$ and $O_{F,i}$ are the event rates for the i-th bin in the near and far detector, calculated for the assumed "true" values of the oscillation parameters. $T_{d,i}$ are the event rates for the parameter values that are currently being tested. a_R, a_N and a_F parameterize the small uncertainties in the reactor flux and the fiducial mass of the two detectors. In this example, their central values are assumed to be zero, while their standard deviations are σ_R, σ_N and σ_F. The first line of Eq. (3.3) is the standard χ^2 expression for the Gauss distribution, while the terms in the second line are penalties for deviations of the systematics parameters from their central values. We use two AEDL files for this experiment, one for the far detector and one for the near detector. The corresponding rule definition (ν_e disappearance) in the far detector AEDL-file is

```
rule(#rule0)<
   @signal = 1.0@#nu_e_disappearance_CC
   @background = 0.0@#nu_e_disappearance_CC /* No background */
   @energy_window = 0.0015 :  0.01

   @sys_off_function = "chiNoSysSpectrum"
```

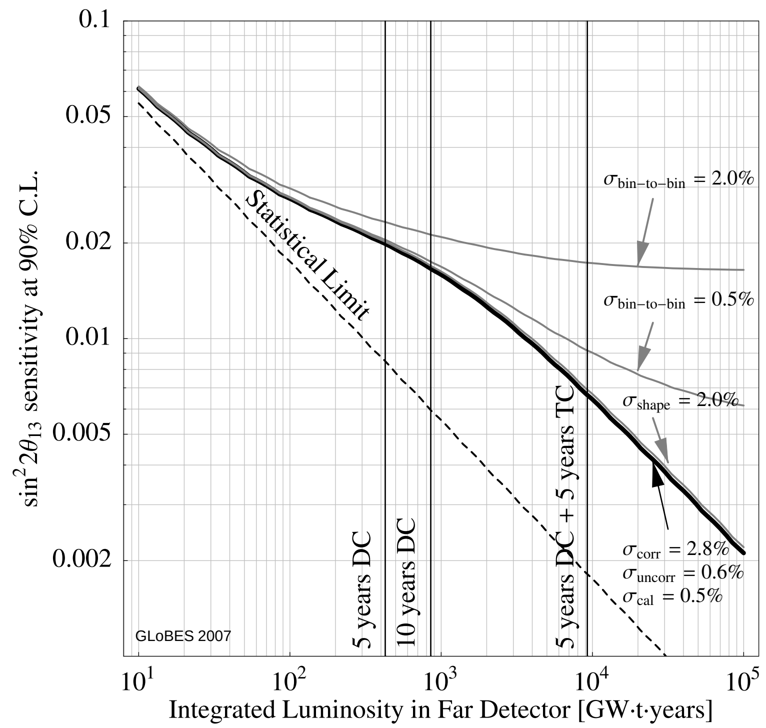


Figure 3.1: The result from `example5.c` (figure similar to the one in Ref. [13]). The thick curve corresponds to the systematics on page 26.

```
    @sys_off_errors = { }
    @sys_on_function = "chiDCNorm"
    @sys_on_errors = { 0.028, 0.006, 0.006, 0.005, 0.005 }
     // { Flux, Fid.  mass FD, Fid.  mass ND, Energy FD, Energy ND }
>
```

In this case, the systematics is called "`chiDCNorm`" for systematics on, whereas systematics off is computed with spectral information, but without systematical errors. For the near detector AEDL-file, we have instead

```
rule(#rule0)<
    @signal = 1.0@#nu_e_disappearance_CC
    @background = 0.0@#nu_e_disappearance_CC /* No background */
    @energy_window = 0.0015 :  0.01

    @sys_off_function = "chiNoSysSpectrum"
    @sys_off_errors = { }
    @sys_on_function = "chiZero"
    @sys_on_errors = { }
>
```

In this case, the systematics `chiZero` is used for systematics on, which means that there will be no active χ^2 calculation in this rule. With this definition, the user-defined systematics will only be called once for the far detector. However, the rates from the near detector will be passively provided for the common χ^2 function. You can find the corresponding files `dchooz-near.glb` and `dchooz-far.glb` you will need for `example5.c`, in the example directory. We show the implementation of the χ^2 function in the example on page 26, where we in addition include an uncorrelated energy calibration error. See Fig. 3.1 for the result of this example.

In cases where the systematics minimization does not converge fast enough or ends up in a local minimum, one can change the starting values of the minimizer, *i.e.*, the starting point from which the local minimizer rolls into the local minimum.

Function 3.7 `int glbSetSysStartingValuesList(int exp, int rule, int on_off, const double *sys_list)` *sets the starting values of the local systematics minimizer in experiment* `exp` *and rule* `rule` *to the values in* `sys_list`. *This change can be performed for systematics on or systematics off by using* `GLB_ON` *or* `GLB_OFF` *for* `on_off`. *Usually, zero is used for all starting values.*

Function 3.8 `double *glbGetSysStartingValuesListPtr(int exp, int rule, int on_off, const double *sys_list)` *returns the starting values of the local systematics minimizer in experiment* `exp` *and rule* `rule`. *This list can be obtained for systematics on or systematics off by using* `GLB_ON` *or* `GLB_OFF` *for* `on_off`.

A useful trick is often to use the minimum from the last run as starting values for the next one if the input parameters are only slightly changed. One can obtain the minimum from the last run from the last call of the χ^2 function (*cf.*, `example5.c`).

Chapter 4

Calculating χ^2-projections: how one can include correlations

This chapter deals with the rather complicated issue of n-parameter correlations. It is one of the greatest strengths of this software to include the full n-parameter correlation in the high-dimensional parameter space with reasonable effort. Of course, calculating χ^2-projections is somewhat more complicated than using systematics only. Therefore, we use a simple step by step introduction to the problem.

4.1 Introduction

In principle, the precision of an individual parameter measurement including correlations in the χ^2-approach can be obtained as the projection of the n-dimensional fit manifold onto the respective axis. Similarly, one can project the fit manifold onto a plane, such as the $\sin^2 2\theta_{13}$-δ_{CP}-plane, if one wants to show the allowed region in this plane with all the other parameter correlations included. In practice, this projection (or marginalization) is very difficult: a grid-based method would need $(N_{\mathrm{grid}})^n$ function calls of `glbChiSys` to calculate the projection onto one dimension including the full n-parameter correlation, where N_{grid} is the number of points in each direction of the lattice. For example, taking only $N_{\mathrm{grid}} = 20$ and $n = 7$ (six oscillation parameters and matter density) would mean more than one billion function calls of `glbChiSys`. One can easily imagine that these would be too many for any reasonable application.

The solution to this problem is using a n-dimensional, local minimizer for the projection instead of a grid-based method, where we will illustrate this minimization process later. It turns out that such a minimizer can include a full 6-parameter correlation with of the order of 1 000 function calls of `glbChiSys`. For the minimization we use a derivative free method due to Powell in a modified [18] version[1].

[1] Not to need derivatives is highly desired, since the event rate depends in a non-linear way on the oscillation parameters. Thus, there is no easy analytical way to obtain derivatives of the χ^2 function.

Thus, for each point on the projection axis/plane, one can obtain a result within several seconds on a modern computer, which means that the complete measurement precision for one fixed true parameter set can be obtained in a few minutes. One can easily imagine that such a minimizer makes more sophisticated applications possible with the help of overnight calculations, such as showing the dependencies on the true parameter values.

This approach also has one major disadvantage: There is *no* such thing as a global minimization algorithm or even an algorithm which guarantees to find *all* local minima of a function. In practice this means using a local minimizer, one may end up in an unwanted local minimum and not in the investigated (possibly global) one or one may miss a local minimum which affects the results[2]. The only way out of this dilemma is to use some heuristic approach, *i.e.*, one can use schemes which work in most cases and announce their failure loudly. In order to use such a heuristic some (analytical or numerical) knowledge on the topology of the fit manifold is necessary. With this knowledge it is possible to obtain an approximate position for each local minimum and thus to start the local minimizer close enough to the investigated minimum. Fortunately, this can be done quite straightforwardly in most cases, since the structure of the neutrino oscillation formulas does not cause very complicated topologies of the fit manifolds. Especially the simulation of reactor experiments and conventional beams or superbeams is rather simple with purely numerical approaches. Neutrino factories have, especially for small values of θ_{13}, a much more complicated topology. In this case, results of the many analytical discussions of this issue can be used. This means that one can implicitly use the analytical knowledge to obtain better predictions for the location of a minimum. One can easily imagine that the used methods then also depend on the region of the parameter space. In this manual, we mainly use examples with a neutrino factory, since some of these complications can be illustrated there. Note that the methods described here are neither complete, nor will they work everywhere in the parameter space. It is in any case up to the user to make sure that the results are what he/she thinks.

Some more words of warning with respect to results obtained by projecting the χ^2: The results obtained this way are always only an *upper* bound on the value of the projected χ^2 function, *i.e.* an undiscovered minimum decreases the value of the the projected χ^2 function. If the value of the χ^2 function in the missed minimum is larger than the previously found ones it will not influence the value of the projected value. Thus, one can only run the danger to obtain a too optimistic solution if one does not find the other local minima appearing below the chosen confidence level. With this approach and proper usage, it should not be possible to produce a too pessimistic solution. However, if one is not careful enough to locate all local minima, one can easily produce too optimistic solutions. This danger can be summarized as follows:

$$\text{Too pessimistic result} < \underbrace{\text{Real result}}_{\text{Located by careful usage}} \leq \textsf{GLoBES}\,\text{result} < \text{Too optimisitic result}$$

[2]NB – Implementing a grid-based method which guarantees to find all local minima is not straightforward either, to say the least.

In many cases, the fit manifold is restricted by the knowledge from earlier experiments. For example, the knowledge on the solar parameters will in most cases be supplied by the solar neutrino experiments. If, at the time of the measurement, the external precision of a parameter is better than the one of the considered experiment itself, one usually will use this better, external knowledge and impose a corresponding constraint on this parameter. The external knowledge may reduce the extension of the n-dimensional fit manifold in the respective direction. In the most extreme case, keeping all parameters but the measured one fixed in the analysis is equivalent to the assumption that all parameters are determined externally with infinitively high precision. As this is quite a strong assumption, one should *always* check the consequences of relaxing it and using realistic errors. Only if such a test has demonstrated that the impact of the uncertainty on a given fit parameter is negligible, it can safely be assumed as fixed. The inclusion of external input in GLoBES is done by the use of Gaußian *priors*: We assume that an external measurement has determined the measured parameter to be at the *central value* with a 1σ Gaußian error (which we call *input error*). The explicit definition of these priors will be shown in the next section.

4.2 The treatment of external input

It is one of the strengths of the GLoBES software to use external input in order to reduce the extension of the fit manifold with the knowledge from external (earlier) measurements. The treatment of external input is done by the addition of Gaußian so-called *priors* to the systematics-minimized χ^2-function. For example, for the matter density, one obtains as the final projected χ^2_F after minimization over the matter density scaling factor $\hat{\rho}$

$$\chi^2_F = \min_{\rho} \left(\chi^2(\hat{\rho}) + \frac{(\hat{\rho} - \hat{\rho}^0)^2}{\sigma^2_{\hat{\rho}}} \right). \tag{4.1}$$

This example is a very simple one, since in fact the minimization is simultaneously performed over all priors and free oscillation parameters. In Eq. (4.1), $\hat{\rho}^0$ is the *central value* of the prior, and $\sigma_{\hat{\rho}}$ the 1σ absolute (half width) *input error*. Thus, it is assumed that an external measurement has determined the matter density with a precision (input error) $\sigma_{\hat{\rho}}$ at the central value $\hat{\rho}^0$. Usually, the central value is fixed at the best-fit value, and the input error is chosen as the 1σ half width of the external measurement. For the matter density, $\hat{\rho}^0$ is usually set to 1.0, corresponding to the actual matter density profile such as given by the experiment definition file, and $\sigma_{\hat{\rho}}$ to the relative matter density uncertainty (*e.g.*, 0.05 for 5% uncertainty).

In principle, one can set the priors for the matter density and all oscillation parameters. For example, if the disappearance channels of the experiment determine the leading oscillation parameters with unprecedented precisions, one can omit the respective input errors. In GLoBES, an input error value of 0 corresponds[3] to neglecting the prior. If, however, earlier external measurements provide better information, one can set their absolute

[3] To be precise, a value for the error in between -10^{-12} and $+10^{-12}$

precisions with the input errors. The central values are usually chosen to be the best-fit values of this external experiments, such as for the input from solar experiments. In some cases, it may be necessary to adjust them, such as for artificial constraints to the oscillation parameters. For example, for the investigation of the opposite-sign solution, one can use the prior to constrain Δm^2_{31} in order to force the minimizer not to fall into the (unwanted) true-sign solution. In this case, the central value of Δm^2_{31} would be set to $\rho^0_{\Delta m^2_{31}} = -\Delta m^2_{31}$, and a $\sigma_{\Delta m^2_{31}}$ of the order of Δm^2_{31} would be imposed. For the algorithm, it would then be rather difficult to converge into the unwanted true-sign solution. However, note that one should in this case check that the actually determined value for Δm^2_{31} after minimization is close enough to the guessed value $-\Delta m^2_{31}$ in order to avoid significant artifical contributions of the priors to the final χ^2. Alternatively one could re-run the minimizer with the position of the previously found minimum as starting position but now with switching off the constraint on Δm^2_{31}.

In order to set the central values and input errors, two function have to be called *before the usage of any minimizer*:

Function 4.1 `int glbSetCentralValues(const glb_params in)` *sets the central values for all of the following minimizer calls to* `in`*.*

Function 4.2 `int glbSetInputErrors(const glb_params in)` *sets the input errors for all of the following minimizer calls to* `in`*. An input error of* 0 *corresponds to not taking into account the respective prior.*

Accordingly, there are functions to return the actually set central values and input errors:

Function 4.3 `int glbGetCentralValues(glb_params out)` *writes the currently set central values to* `out`*.*

Function 4.4 `int glbGetInputErrors(glb_params out)` *writes the currently set input errors to* `out`*.*

All functions take or return as many matter density parameters as there are initialized experiments. In addition, they return -1 if the operation was not successful.

Eventually, a typical initialization of the external input with 10% external precisions for the solar parameters[4], and 5% matter density uncertainties for all experiments looks like this:

```
glbDefineParams(input_errors,theta12*0.1,0,0,0,sdm*0.1,0);
glbSetDensityParams(input_errors,0.05,GLB_ALL);
glbSetCentralValues(true_values);
glbSetInputErrors(input_errors);
```

[4]In fact, accelerator-based long-baseline experiments are primarily sensitive to the product $\sin 2\theta_{12} \cdot \Delta m^2_{21}$, which means that these errors effectively add up to an error of this product of about 15%.

In this example, the central values are set to the true (simulated) values.

Though the priors are an elegant way to treat external input, there are also some complications with priors. The following hints are for the more advanced GLoBES user:

1. The priors are only added once to the final χ^2, no matter how many experiments there are simulated. This is already one reason (besides the minimization) why the sum of all projected χ^2's of the individual experiments cannot correspond to the χ^2 of the combination of all experiments.

2. Priors are not used for parameters which are not minimized over, *i.e.*, kept fixed. This will be important together with arbitrary projections using `glbChiNP`. A more subtle consequence is the comparison of fit manifold sections and projections for the solutions where the absolute minimum χ^2 is larger than zero, *i.e.*, degeneracies other than the true solution. In this case, the sections and projections are not comparable if not corrected by the prior contributions, where the correction can be obtained as the χ^2-difference at the minimum. For example, projecting the $\mathrm{sgn}(\Delta m^2_{31})$-degeneracy onto the θ_{13}-δ_{CP}-plane and comparing it with the section (all other parameters fixed), the section region would in many cases be larger than the projection region if the priors were not added to the section. At the true solution, this problem usually does not occur because the prior contributions are close to zero.

3. Currently, GLoBES only supports Gaußian pre-defined priors for the individual oscillation parameters. Especially for the solar parameters, this is only an approximation, since they are imposed on θ_{12} and not on Δm^2_{31} or $\sin^2\theta_{12}$. GLoBES 3.0 and higher provide an alternative to that by the concept of user-defined priors (*cf.*, Sec. 4.5).

4.3 Projection onto the $\sin^2 2\theta_{13}$- or δ_{CP}-axis

The projection onto the $\sin^2 2\theta_{13}$- (or δ_{CP}-) axis is performed by fixing $\sin^2 2\theta_{13}$ (or δ_{CP}) and minimizing the χ^2-function over all free fit parameters and the matter densities. We illustrate this method by the example of the projection of the two-dimensional manifold in the $\sin^2 2\theta_{13}$-δ_{CP}-plane onto the $\sin^2 2\theta_{13}$-axis in Fig. 4.1. In this figure, the left-hand plot shows the correlation in the $\sin^2 2\theta_{13}$-δ_{CP}-plane computed with `glbChiSys`. The right-hand plot illustrates the projection of this two-dimensional manifold onto the $\sin^2 2\theta_{13}$ axis by minimizing χ^2 over δ_{CP}. In this simple example, the minimization is done along the vertical gray lines in the left hand plot. The obtained minima are located on the thick gray curve, which means that the right-hand plot represents the χ^2-value along this curve. In fact, one can easily see that one obtains the correct projected 3σ errors in this example (*cf.*, arrows). This figure illustrates the projection of a two-parameter correlation. In general, the full n-parameter correlation is treated similarly by the simultaneous (local) minimization over all free fit parameters.

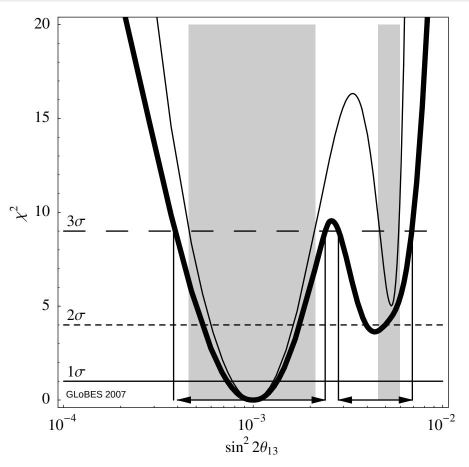
The following functions are some of the simplest minimizers provided by GLoBES:

Example: Projection of two- and n-dimensional manifold onto $\sin^2 2\theta_{13}$*-axis*

This example demonstrates how to project the fit manifold onto the $\sin^2 2\theta_{13}$-axis, *i.e.*, how one can include correlations. We compute two sets of data: one for keeping all parameters but δ_{CP} fixed (two-parameter correlation), and one for keeping all parameters free (multi-parameter correlation). However, we impose external precisions for the solar parameters and the matter density. The following code excerpt is from `example2.c`:

```
/* Set central values and input errors for all projections */
glbDefineParams(input_errors,theta12*0.1,0,0,0,sdm*0.1,0);
glbSetDensityParams(input_errors,0.05,GLB_ALL);
glbSetCentralValues(true_values); glbSetInputErrors(input_errors);

/* Define my own two-parameter projection for glbChiNP: Only deltacp is free!  */
glbDefineProjection(th13_projection,GLB_FIXED,GLB_FIXED,GLB_FIXED,GLB_FREE,GLB_FIXED,GLB_FIXED);
glbSetDensityProjectionFlag(th13_projection,GLB_FIXED,GLB_ALL);
glbSetProjection(th13_projection);

/* Iteration over all values to be computed */
double x,res1,res2;
for(x=-4;x<-2.0+0.001;x=x+2.0/50)
{
    /* Set fit value of stheta */
    glbSetOscParams(test_values,asin(sqrt(pow(10,x)))/2,1);

    /* Guess fit value for deltacp in order to safely find minimum */
    glbSetOscParams(test_values,200.0/2*(x+4)*M_PI/180,3);

    /* Compute Chi2 for user-defined two-parameter correlation */
    res1=glbChiNP(test_values,NULL,GLB_ALL);

    /* Compute Chi2 for full correlation:  minimize over all but theta13 */
    res2=glbChiTheta13(test_values,NULL,GLB_ALL);

    AddToOutput(x,res1,res2);
}
```

The two lists of data then represent the $\sin^2 2\theta_{13}$ precisions with two-parameter correlations (gray-shaded) and multi-parameter correlations (arrows):



(Same parameters as on page 24 and in Fig. 4.1, but 1 d.o.f.)

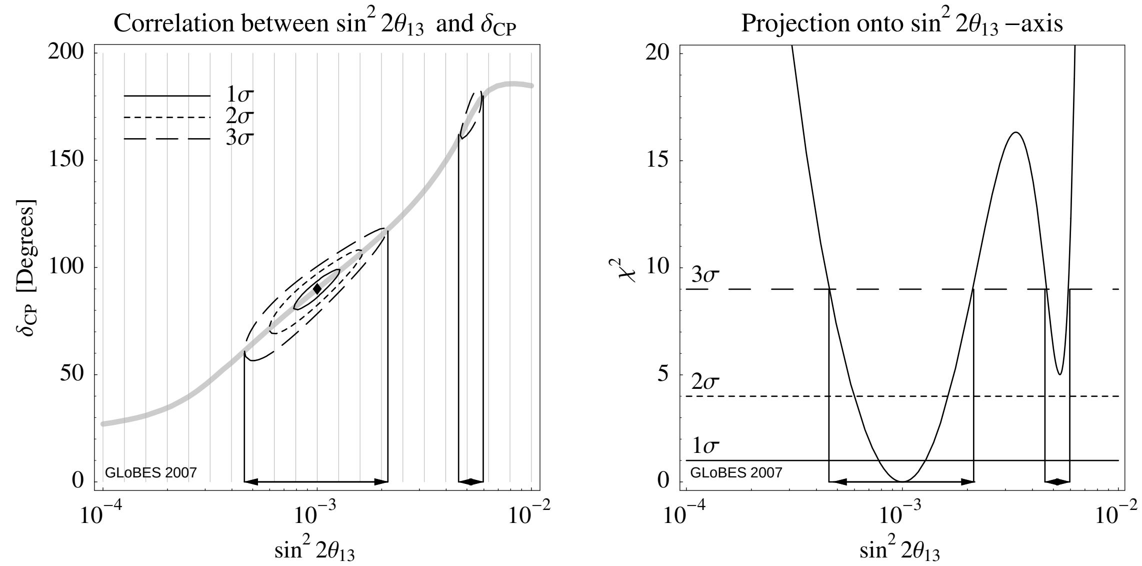


Figure 4.1: Left plot: The correlation between $\sin^2 2\theta_{13}$ and δ_{CP} as calculated in the example on page 24, but for 1 d.o.f. only. Right plot: The χ^2-value of the projection onto the $\sin^2 2\theta_{13}$-axis as function of $\sin^2 2\theta_{13}$. The projection onto the $\sin^2 2\theta_{13}$-axis is obtained by finding the minimum χ^2-value for each fixed value of $\sin^2 2\theta_{13}$ in the left-hand plot, *i.e.*, along the gray vertical lines. The thick gray curve marks the position of these minima in the left-hand plot. The arrows mark the obtained fit ranges for $\sin^2 2\theta_{13}$ at the 3σ confidence level (1 d.o.f.), *i.e.*, the precision of $\sin^2 2\theta_{13}$.

Function 4.5 `double glbChiTheta13(const glb_params in, glb_params out, int exp)` *returns the projected χ^2 onto the θ_{13}-axis for the experiment* `exp`*. For the simulation of all initialized experiments, use* `GLB_ALL` *for* `exp`*. The values in* `in` *are the guessed fit values for the minimizer (all parameters other than θ_{13}) and the fixed fit value of θ_{13}. The actually determined parameters at the minimum are returned in* `out`*, where θ_{13} is still at its fixed value. If* `out` *is set to* `NULL`*, this information will not be returned.*

Function 4.6 `double glbChiDelta(const glb_params in, glb_params out, int exp)` *returns the projected χ^2 onto the δ_{CP}-axis for the experiment* `exp`*. For the simulation of all initialized experiments, use* `GLB_ALL` *for* `exp`*. The values in* `in` *are the guessed fit values for the minimizer (all parameters other than δ_{CP}) and the fixed fit value of δ_{CP}. The actually determined parameters at the minimum are returned in* `out`*, where δ_{CP} is still at its fixed value. If* `out` *is set to* `NULL`*, this information will not be returned.*

All of the minimization functions have a similar parameter structure: The fixed fit parameter value and the guessed starting point of the minimizer, *i.e.*, the guessed position of the minimum, are transferred in the list `in`. Part of this list are the matter density scaling factors of all experiments, which are also minimized over. The minimizer is then started at the guessed point and runs into the local minimum, where the fit parameter of the projection axis is fixed. For the true solution, it is usually sufficient to start the minimizer at the true parameter values. However, the convergence speed might be better by starting it slightly off this point. In addition, there are problems in many cases with more complicated topologies, which means that better guesses for the position of the minimum are needed. The position of the minimum is then returned in `out` together with the number of iterations used for the minimization. It is very often useful to print the output of the minimization with `glbPrintParams` in order to check that the minimum is the appropriate one. For example, if the minimizer ends up in the wrong-sign solution in Δm^2_{31}, priors can be used to force it into the tested minimum. In addition, the number of iterations used allows an optimization of the convergence speed. Note that before any minimization, `glbSetCentralValues` and `glbSetInputErrors` have to be used at least once. In addition, note that the resulting χ^2 of `glbChiTheta13` (or `glbChiDelta`) for the combination of more than one experiment is not equal to the sum of the individual χ^2-values anymore. This has two reasons: First, the topology of the fit manifold is altered by the addition of χ^2-values of different experiments. Thus, after the minimization, the position of the minimum can be different to the ones of the individual experiments. Second, the priors for the external knowledge on the parameters are only added once – independent of the number of experiments.

The output of the minimizer in `out` carries as many matter density scaling factors as there are experiments. Either one (for the simulation of one experiment) or all (for the simulation of all experiments) are different from 1.0 if matter density uncertainties are present, since each experiment may face other matter density conditions. The minimizers of individual experiments "know" which experiment they are currently treating, which means that they only return the matter density scaling factor of the appropriate experiment. For example, calculating `glbChiTheta13` for the last experiment number, the last density value will be returned. This approach turns out to be extremely useful together with the simulation of more than one experiment. One can, for instance, locate the degeneracies of all individual experiments. In order to test if these degeneracies are still present in the combination of all experiments (which has a very different topology), one can test the combination of experiments with the output `out` from the individual experiments. In this case, even the correct matter density scaling factor output is used.

The example on page 36 demonstrates how one can obtain Fig. 4.1 (right) with keeping all parameters but δ_{CP} fixed, as well as how one can include the full n-parameter correlation with external input. It also demonstrates how these two compare to each other. One can easily read off this example that there is a substantial impact of the correlation with oscillation parameters other than δ_{CP}. Note that it uses the function `glbChiNP` for arbitrary projections from the next section for the minimization over δ_{CP}. In addition, there is one interesting feature in guessing the oscillation parameters in this example: In order to avoid falling into the wrong minimum, the fit value of δ_{CP} is guessed from Fig. 4.1 (left). This quite sophisticated "guessing" is typical for neutrino factories because of the $(\delta_{\mathrm{CP}}, \theta_{13})$-degeneracy, whereas it is for superbeams often sufficient to use the true values. A strong indication for a wrong guessing are discontinuous jumps in the projected χ^2-function, where the minimizer jumps from one minimum to another. In such cases, the starting point of the minimizer has to be adjusted to help it find the true minimum. Other examples for projections onto a parameter axis while keeping exactly one parameter fixed are `glbChiTheta23`, `glbChiDm31`, and `glbChiDm21`, which can be found in Table 1.1 on page 4.

Since the number of different parameter vectors used by GLoBES may be a bit confusing, we summarize the most common parameter sets used in the calculations:

The simulated/true values are the values set in `glbSetRates`.

The fit values are the values in the first parameter `in` of `glbChiTheta13` (and the other minimization functions) which are *fixed* by definition (such as θ_{13} for `glbChiTheta13`).

The starting point/educated guess for the minimizer are the values in the first parameter `in` of `glbChiTheta13` (and the other minimization functions) which are *free* by definition (such as all but θ_{13} for `glbChiTheta13`).

The minimization result of the marginalization process are the values in the second parameter `out` of `glbChiTheta13` (and the other minimization functions) which are *free* by definition (such as all but θ_{13} for `glbChiTheta13`). The other values in `out` still correspond to the fit values.

The position of external priors, *i.e.*, the best-fit values of external input, is set by `glbSetCentralValues`.

The magnitude of external errors, *i.e.*, the errors of external input, is set by `glbSetInputErrors`.

4.4 Projection onto any hyperplane*

In general, one can show the measurement result in any k-dimensional hyperplane, where k is smaller than the dimension of the parameter space n, and thus the dimension of the fit manifold. In this case, k parameters are fixed and $n-k$ parameters are minimized over. One such example is the projection of the fit manifold onto the $\sin^2 2\theta_{13}$-δ_{CP}-plane, *i.e.*, $k=2$ here. In this case, the two parameters $\sin^2 2\theta_{13}$ and δ_{CP} are kept fixed, and the others are minimized over. The corresponding function is

Function 4.7 `double glbChiTheta13Delta(const glb_params in, glb_params out, int exp)` *returns the projected χ^2 onto the θ_{13}-δ_{CP}-plane for the experiment* `exp`*. For the simulation of all initialized experiments, use* `GLB_ALL` *for* `exp`*. The values in* `in` *are the guessed fit values for the minimizer (all parameters other than θ_{13} and δ_{CP}) and the fixed fit values of θ_{13} and δ_{CP}. The actually determined parameters at the minimum are returned in* `out`*, where θ_{13} and δ_{CP} are still at their fixed values. If* `out` *is set to* `NULL`*, this information will not be returned.*

This function works analogously to the ones in the last section. It can, for example, be used to obtain a figure similar to Fig. 4.1, left, but with all other parameters marginalized

Function	Purpose	Parameters → Result
`glbAllocProjection`	Allocate projection vector	`()`
`glbFreeProjection`	Free projection vector `stale`	`(glb_projection stale)`
`glbDefineProjection`	Assign projection vector `in`	`(glb_projection in, int theta12, int theta13, int theta23, int delta, int dms, int dma)`
`glbCopyProjection`	Copy vector `source` to `dest`	`(const glb_projection source, glb_projection dest)`
`glbPrintProjection`	Print vector `in` to file `stream`	`(FILE* stream, const glb_projection in)`
`glbSetProjectionFlag`	Set flag for oscillation parameter `which` in vector `in` to value `flag`.	`(glb_projection in, int flag, int which)`
`glbGetProjectionFlag`	Return flag for oscillation parameter `which` in vector `in`.	`(const glb_projection in, int which)` → `int` flag
`glbSetDensity-ProjectionFlag`	Set flag for density parameter `which` in vector `in` to value `flag`.	`(glb_projection in, int flag, int which)`
`glbGetDensity-ProjectionFlag`	Return flag for density parameter `which` in vector `in`.	`(const glb_projection in, int which)` → `int` flag

Table 4.1: Different functions handling the `glb_projection` type. Flags are either `GLB_FIXED` or `GLB_FREE`. The (un-shown) return values of the `Set-` and `Define-` functions point either to the assigned vector if successful, or they are `NULL` if unsuccessful.

over. The example on page 36 illustrates then the result of the projection of the "eggs" within the $\sin^2 2\theta_{13}$-δ_{CP}-plane onto the θ_{13}-axis. Though the running time for one call of these functions is somewhat shorter than the one for the $\sin^2 2\theta_{13}$- or δ_{CP}-projections, one has to compute a two-dimensional array for such a figure (instead of a one-dimensional list). Therefore, the overall computational effort is much higher, *i.e.*, in the order of hours. In many cases, it is therefore convenient to run `glbChiSys` first to obtain a picture of the manifold and to adjust the parameter ranges. Then, one can run `glbChiTheta13Delta` for a complete evaluation of the problem including correlations.

In principle, one can also use three- or more-dimensional projections. In addition, one may want to use a different set of parameters for single- or two-parameter projections. The very flexible function `glbChiNP` is designed for this purpose. However, because of its flexibility, it involves more sophistication.

In order to define arbitrary projections, we introduce the vector `glb_projection`, which is very similar to the oscillation parameter vector `glb_params`. Normally, the user does not need to access this type directly: A set of function similar to the ones for `glb_params` is provided. The purpose of `glb_projection` is to tell GLoBES which parameters are fixed, and which are minimized over. Thus, in comparison to `glb_params`, it does not take values for the parameters, but flags `GLB_FIXED` or `GLB_FREE`. For example, the projection onto the θ_{13}-axis `glbChiTheta13` is nothing else than a special case of `glbChiNP` with θ_{13} fixed and all the other parameters free. Similar to `glb_params`, the type `glb_projection` has to be allocated first, and freed later. The access functions for `glb_projection` are summarized in Table 4.1. Since the complete set is very similar to the one for `glb_params`, we do not go into greater details here.

As soon as we have defined a projection, we can assign it:

Function 4.8 `int glbSetProjection(const glb_projection in)` *sets the projection to* `in`*. The return value is* 0 *if successful, and* -1 *if unsuccessful.*

Similarly, the currently assigned projection can be returned with:

Function 4.9 `int glbGetProjection(glb_projection out)` *writes the currently set projection to* `out`*. The return value is* 0 *if successful, and* -1 *if unsuccessful.*

After setting the central values, input errors, and the projection, we can run the minimizer:

Function 4.10 `double glbChiNP(const glb_params in, glb_params out, int exp)` *returns the projected* χ^2 *onto the hyperplane specified by* `glbSetProjection` *for the experiment* `exp`*. For the simulation of all initialized experiments, use* `GLB_ALL` *for* `exp`*. The values in* `in` *are the guessed fit values for the minimizer (all free parameters) and the fit values on the hyperplane (all fixed parameters). The actually determined parameters at the minimum are returned in* `out`*, where the fixed parameters are still at their input values. If* `out` *is set to* `NULL`*, this information will not be returned.*

As an example, the projection sequence for a minimization over δ_{CP} only (and the matter density parameters) looks like this:

```
glb_projection th13_projection = glbAllocProjection();
glbDefineProjection(th13_projection,GLB_FIXED,GLB_FIXED,GLB_FIXED,
          GLB_FREE,GLB_FIXED,GLB_FIXED);
glbSetDensityProjectionFlag(t13_projection,GLB_FIXED,GLB_ALL);
glbSetProjection(th13_projection);
res1=glbChiNP(test_values,NULL,GLB_ALL);
glbFreeProjection(th13_projection);
```

In this case, only the correlation with δ_{CP} is taken into account. Note that in the example on page 36 this projection is compared with the result including the full multi-parameter correlation.

4.5 User-defined priors*

User-defined priors are an advanced concept of GLoBES 3.0 and higher. Therefore, this section can be skipped in a first reading of the manual. They allow for arbitrary priors

Example: Non-Gaussian external solar input

This examples demonstrates how to use a non-Gaussian error for the external solar mixing angle input instead of a Gaussian error. The user-defined prior function reads as follows and is very similar to the standard prior:

```
double my_prior(const glb_params in, void* user_data)
{
    glb_params central_values = glbAllocParams();
    glb_params input_errors = glbAllocParams();
    glb_projection p = glbAllocProjection();
    glbGetCentralValues(central_values);
    glbGetInputErrors(input_errors);
    glbGetProjection(p);
    int i;
    double pv = 0.0;
    double fitvalue,centralvalue,inputerror;

    /* Add oscillation parameter priors */
    for(i=0;i<6;i++)
    if(glbGetProjectionFlag(p,i)==GLB_FREE)
    {
       fitvalue=glbGetOscParams(in,i);
       centralvalue=glbGetOscParams(central_values,i);
       inputerror=glbGetOscParams(input_errors,i);
       if(inputerror>1e-12)       {
          if(i==GLB_THETA_12) /* Prior on sin^2 θ_12 */
             pv+=square((startvalue-square(sin(fitvalue)))/inputerror);
          else
             pv+=square((startvalue-fitvalue)/inputerror);
       }
    }

    /* Add matter parameter priors */
    for(i=0;i<glb_num_of_exps;i++)
    if(glbGetDensityProjectionFlag(p,i)==GLB_FREE)
    {
       fitvalue=glbGetDensityParams(in,i);
       centralvalue=1.0;
       inputerror=glbGetDensityParams(input_errors,i);
       if(inputerror>1e-12)
          pv+=square((centralvalue-fitvalue)/inputerror);
    }

    glbFreeParams(central_values);
    glbFreeParams(input_errors);
    glbFreeProjection(p);
    return pv;
}
```

Note that this prior interprets the central values and input errors in terms of $\sin^2\theta_{12}$ instead of θ_{12}.

which depend on the oscillation parameters only (as opposed to systematics; *cf.*, Eq. (4.1)). Therefore, compared to user-defined systematics, they depend on the oscillation parameters only, but not on the systematics parameters. Examples for applications are non-Gaussian external input, the combination with externally simulated experiments, and the constraint to certain parameter-subspaces (such as a specific octant).

The following function replaces the standard priors by user defined ones and has to be used after `glbInit`:

Function 4.11 `glbRegisterPriorFunction(double (*prior)(const glb_params, void *user_data), int (*central)(const glb_params, void *user_data), int (*error)(const glb_params, void *user_data), void *user_data)` *registers a user-defined prior function* `prior`*. In addition, it is possible to register two functions* `central` *and* `error` *being called every time* `glbSetCentralValues` *or* `glbSetInputErrors` *are called with the same argument. For example, the central values and input errors can be stored in global variables for a faster access. These pointers can also be* `NULL`*. The pointer* `user_data` *can be optionally used to circumvent the use of global variables. It is set by* `glbRegisterPriorFunction` *and transferred to the registered functions whenever called.*

The prior function should expect the parameter structure `glb_params` containing the current fit values and the `void`-pointer `user_data` as arguments. We show an example for a user-defined prior on page 42. This prior is simply registered with

```
glbInit(argv[0]);
glbRegisterPriorFunction(my_prior,NULL,NULL,NULL);
```

and replaces the standard prior of GLoBES. It behaves exactly as the standard prior, but the solar mixing angle is interpreted as $\sin^2\theta_{12}$ instead of θ_{12}. Therefore, a Gaussian error is imposed on $\sin^2\theta_{12}$ instead of θ_{12}, and the central values and input errors are interpreted in terms of $\sin^2\theta_{12}$ as well. Note that one does not need to know if the prior was called for one experiment only or all experiments, since this information is implicitly given by the different density projection flags being set accordingly.

Chapter 5

Locating degenerate solutions

Here we describe how one can locate degenerate solutions in GLoBES, and we discuss several techniques for the application software.

5.1 Minimization over all oscillation parameters

In the last chapter, we introduced marginalizations over different parameters to obtain measurement precisions. Similarly, one can minimize over *all* n parameters to find the local minimum close to any starting point. This approach is very useful for the exact numerical location of a degeneracy if its approximate position is known. For the determination of the approximate position, one can use analytical approaches or an educated guess. Though the usage of the all-parameter minimizers is quite simple, one should keep in mind that they are local minimizers. Therefore, one may need a very sophisticated application software to find all degenerate solutions.

The function to perform the all-parameter minimization is `glbChiAll`:

Function 5.1 `double glbChiAll(const glb_params in, glb_params out, int exp)` *returns the minimized* χ^2 *over all parameters for the experiment* `exp`*. For the simulation of all initialized experiments, use* `GLB_ALL` *for* `exp`*. The values in* `in` *are the guessed fit values for the minimizer. The actually determined parameters at the minimum are returned in* `out`*. If* `out` *is set to* `NULL`*, this information will not be returned.*

This function takes the suspected position of the local minimum and returns its actual position in `out`, as well as the χ^2-value at the minimum as return value. Thus, the return value can be immediately used to judge whether the located degeneracy appears at the chosen confidence level.

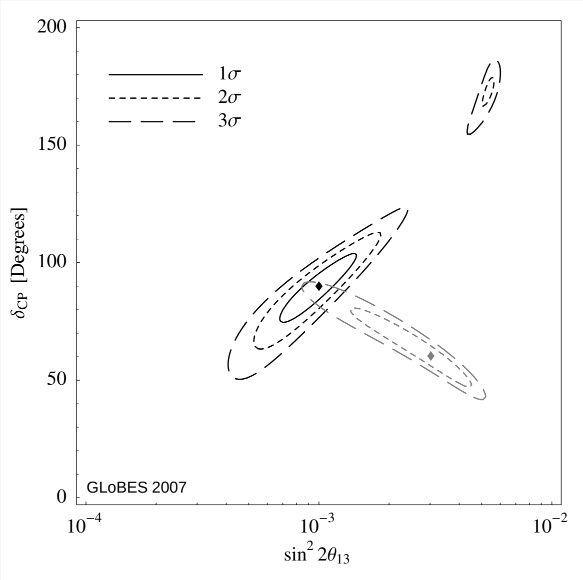
The example on page 46 illustrates how to locate the $\mathrm{sgn}(\Delta m_{31}^2)$-degeneracy and show the corresponding degenerate solution in the $\sin^2 2\theta_{13}$-δ_{CP}-plane together with the original solution. In this case, the position of the degeneracy can be easily guessed to be at the true parameter values but with inverted Δm_{31}^2. The minimizer then runs off the plane of these parameters into the local minimum. It is very important to take into account the position

Example: Finding the sgn(Δm_{31}^2)*-degeneracy*

In many cases, one can find the exact position of the sgn(Δm_{31}^2)-degeneracy with **glbChiAll**, where one starts the local minimizer at the suspected position and lets it run into the minimum. The following code excerpt corresponds to finding the degenerate solution for the example on page 24, and it is from `example3.c`:

```
/* Set starting vales to suspected position at opposite sign of ldm */
glbDefineParams(central_values,theta12,theta13,theta23,deltacp,sdm,-ldm);
glbSetDensityParams(central_values,1.0,GLB_ALL);

/* Set input errors for external input, where some information on ldm
    is imposed in order to avoid falling into the right-sign solution */
glbDefineParams(input_errors,theta12*0.1,0,0,0,sdm*0.1,ldm/3);
glbSetDensityParams(input_errors,0.05,GLB_ALL);

glbSetCentralValues(central_values);
glbSetInputErrors(input_errors);

/* Localize degenerate solution by minimization over all parameters
*/
double CL=glbChiAll(central_values,deg_pos,GLB_ALL);

/* Now:  CL is the chi2 of the deg. solution and deg_pos the position */
```

Using `deg_pos` to obtain a section of the degeneracy in the $\sin^2 2\theta_{13}$-$\delta_{\rm CP}$-plane (*cf.*, `example3.c`), one can plot it as a contour plot in addition to the original solution (2 d.o.f., gray curves):



of the degeneracy off this plane, since the actual χ^2 in the minimum is probably lower than in the plane. Thus, the degeneracy may not even appear at the chosen confidence level in the plane, but it does appear at the real minimum. The two sections through the fit manifold shown in the figure on page 46 therefore do not appear at the same oscillation parameter values.[1]

Note: Inverting the mass hierarchy is not precisely the same as changing from $\Delta m_{31}^2 \rightarrow -\Delta m_{31}^2$. In this case the absolute value of Δm_{32}^2 changes also, which introduces a new frequency to the problem. Therefore, if we assume normal hierarchy whenever $\Delta m_{31}^2 > 0$, the corresponding point in parameters space for inverted hierarchy is given by $\Delta m_{31}^2 \rightarrow -\Delta m_{31}^2 + \Delta m_{21}^2$, because with this definition the absolute value of Δm_{32}^2 is unchanged and no new frequency is introduced.

5.2 Advanced tricks for degeneracy localization*

For the more advanced reader, a number of tricks can be useful for the numerical localization of degenerate solutions. Here we give a qualitative, incomplete list:

Minimum χ^2 larger than threshold. If a located degeneracy has a minimum χ^2 larger than the corresponding confidence level threshold for the discussed quantity of interest, the degeneracy can be immediately ignored. This saves a lot of computation time.

Locating degeneracies in more complicated topologies. For more complicated topologies, such as for neutrino factories, it is often useful to use multi-step location procedures or analytical knowledge. For example, for a numerical procedure, one may first of all switch off the systematics and keep $\sin^2 2\theta_{13}$ or δ_{CP} fixed, *i.e.*, use `glbChiTheta13`, where $\sin^2 2\theta_{13}$ or δ_{CP} is fixed at the best-fit value. The result can then be used as a starting point for `glbChiAll` for the individual experiments with the systematics switched on again.

Forcing the minimizer into the targeted solution. In addition to switching off the systematics, it can be useful to reduce the input errors during some steps of the localization process in order to prevent the minimizer from running away too far from the targeted solution. The example on page 46 illustrates this with the input error for Δm_{31}^2: Since the guessed starting point might be quite far away from the real degeneracy, the algorithm may in some cases find the original solution instead of the degeneracy (which can be immediately seen from the output vector). The input error for Δm_{31}^2 gives the algorithm a "bias" against the original solution. However, note

[1]The discussed figure on page 46 is produced by `glbChiSys` and thus only represents a section through the fit manifold. For the projection including correlations, one may rather want to use `glbChiTheta13Delta`.

that the input error must not be too small in order to avoid a significant contribution of the prior to the final χ^2. Alternatively, one could once again run `glbChiAll` with the located minimum as `in` vector, and Δm^2_{31} kept free.

Finding degeneracies with multiple experiments. For multiple experiments, it turns out to be useful to locate the degeneracies for individual experiments first. Then, all of the found degeneracies below the threshold can be tested for the combination of all experiments.

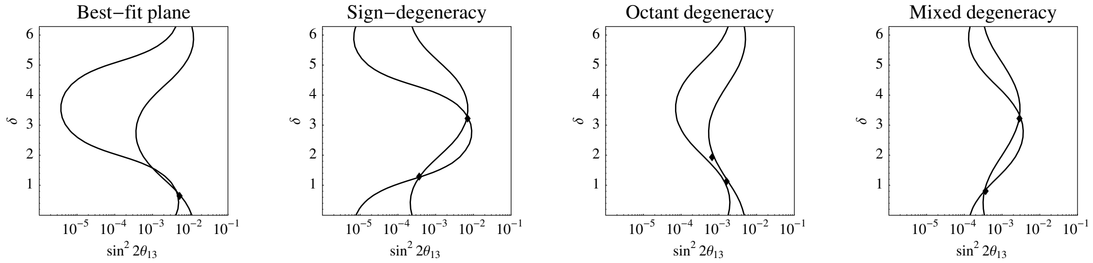
Tracking algorithm. If one scans a large portion of the parameter space with different input values, it is often useful to use the output from the previous minimization as educated guess for the next minimization. This works often very well if the degeneracy can be located in a part of the parameter space and the input parameter values change slowly enough (adiabatic transformation of the fit manifold).

Pre-scanning the parameter space. In some cases, a very fast procedure can be a pre-scan of the relevant parameters using the very fast `glbChiSys`. For example, for the intrinsic degeneracy, the location of the degeneracy in the $(\delta_{\mathrm{CP}}, \theta_{13})$ plane can easily and quickly found with `glbChiSys` while keeping the other parameters fixed. Use this location then as a starting point for `glbChiAll`.

Manual scan of a subspace. In some cases, the minimizer easily ends up in the wrong/unwanted solution, which is in most cases the already known best-fit solution. For example, when locating the octant degeneracy, it is difficult to prevent the minimizer from running into the best-fit octant. In this case, one can scan one parameter (such as θ_{23}) on a grid by choosing only valid values, and use `glbChiNP` to marginalize over the other ones (while keeping *e.g.* θ_{23} fixed). Therefore, the 6-parameter minimization is split into a 1-parameter grid-based minimization and a 5-parameter simultaneous minimization. Advantage: Runs with all GLoBES versions. Disadvantage: Very slow.

Using Schwetz-priors (GLoBES 3.0 and higher). An elegant and fast method to prevent the minimizer from running into parts of the parameter space is to use user-defined priors and add a penalty as soon as a taboo zone is entered (*cf.*, Sec. 4.5). This method was initially suggested by Thomas Schwetz.

Using total rates. In order to systematically locate all degeneracies (the eight-fold degeneracy), relatively reliable methods use the total appearance neutrino and antineutrino event rates of the experiment to determine educated guesses. The method goes as follows: Plot the curves with equal total rate in the $\sin^2 2\theta_{13}$-δ_{CP}-plane for both neutrinos and antineutrinos using the same rates as in the best-fit point. The curves will intersect at the intrinsic degeneracy (if it all). Plot the same curves for the same rates with $\mathrm{sgn}(\Delta m^2_{31})$ flipped. Again you will find a maximum of two intersection points. Now do the same for $\pi/2 - \theta_{23}$ flipped, and for $\mathrm{sgn}(\Delta m^2_{31})$ combined with $\pi/2 - \theta_{23}$ flipped (mixed degeneracy). You will find at most two more intersection points in each case. The results should look somewhat like this, where the best-fit point is not marked[2]:



Altogether, there is a maximum of eight intersection points in the $\sin^2 2\theta_{13}$-δ_{CP}-plane, one of which is the best-fit point. These points can be used as starting points for the minimizer to locate the eight-fold degeneracy. Note that similar methods using the χ^2 instead of the total rates have also been successfully used in the past. In this case, one would scan for local minimas disconnected from the best-fit solution.

Finally, note that any degenerate solution below the confidence level threshold, which cannot be located, makes the result appear better than it actually is. Thus, one should be careful with the determination of the degenerate solutions in order to find all of them.

[2]The marks correspond to the points where the discrete degeneracies are located according to this specific algorithm. Note that one also wants to find the positions of close-to-intersections, because statistical errors may be as large as that the corresponding degenerate solution may still be present at the chosen confidence level. Therefore, in the third plot these close-to-intersections are marked as well.

Chapter 6

Obtaining low-level information

In this chapter, we discuss possibilities to obtain low-level information in GLoBES, *i.e.*, oscillation probabilities, rates, and other information lower than on the χ^2-level.

6.1 Oscillation probabilities

GLoBES can compute the probabilities in vacuum with the following function:

Function 6.1 `double glbVacuumProbability(int l, int m, int panti,double E, double L)` *returns the neutrino oscillation probability* $\nu_l \to \nu_m$ *for the energy* `E` *and the baseline* `L` *in vacuum. The parameter* `panti` *is* $+1$ *for neutrinos and* -1 *for antineutrinos.*

Note that for this and the other probability functions $1 \leq$ `l`, `m` ≤ 3. In addition, the oscillation probabilities in matter can be obtained with:

Function 6.2 `double glbProfileProbability(int exp, int l, int m, int panti,double E)` *returns the neutrino oscillation probability* $\nu_l \to \nu_m$ *for the energy* `E` *in matter, where the matter density profile is the one of experiment* `exp`. *The parameter* `panti` *is* $+1$ *for neutrinos and* -1 *for antineutrinos.*

This function ignores the filter state, and it does not use the filter if switched on. For a constant matter density profile, it is sufficient to specify the oscillation channel with

Function 6.3 `double glbConstantDensityProbability(int l, int m, int panti, double E, double L, double rho)` *returns the neutrino oscillation probability* $\nu_l \to \nu_m$ *for the energy* `E` *in constant matter, where the matter density profile has the constant density* `rho` *and the baseline is* `L`. *The parameter* `panti` *is* $+1$ *for neutrinos and* -1 *for antineutrinos.*

If one in addition wants to use the low-pass filter feature in GLoBES (see Sec. 11.5), one can can use

Function 6.4 `double glbFilteredConstantDensityProbability(int exp, int l, int m, int panti, double E)` *returns the neutrino oscillation probability $\nu_l \to \nu_m$ for the energy* `E` *in constant matter, where the matter density and baseline, as well as the filter properties are taken from experiment* `exp`*. The parameter* `panti` *is $+1$ for neutrinos and -1 for antineutrinos.*

This function uses the filter depending on the filter state, *i.e.*, if it is switched off, it will not be used.

6.2 Information from AEDL files*

In some cases, it is necessary to obtain information from the loaded AEDL files. This and the next sections are marked as advanced because knowledge of AEDL is required, *i.e.*, the reader should be familiar with Part II of the manual. Very basic information is supplied with

Function 6.5 `int glbGetEminEmax(int experiment, double *emin, double *emax)` *returns the energy range [*`emin`*,* `emax`*] covered by the binning as defined in the* AEDL *file for the experiment* `experiment`*.*

Function 6.6 `int glbGetEnergyWindow(int exp, int rule, double *low, double *high)` *returns the energy window [*`low`*,* `high`*] defined in the* AEDL *file for the experiment* `exp` *and rule* `rule`*.*

Function 6.7 `int glbGetEnergyWindowBins(int exp, int rule, int *lowbin, int *highbin)` *returns the energy window in terms of bin numbers [*`lowbin`*,* `highbin`*] corresponding to the energy window defined in the* AEDL *file for the experiment* `exp` *and rule* `rule`*.*

Function 6.8 `int glbGetNumberOfBins(int exp)` *returns the number of bins for the experiment* `exp`*.*

Function 6.9 `int glbGetNumberOfSamplingPoints(int exp)` *returns the number of sampling points for the experiment* `exp`*.*

Function 6.10 `double *glbGetBinSizeListPtr(int exp)` *returns a pointer to the array of bin widths for the experiment* `exp`*.*

Function 6.11 `double *glbGetSamplingStepsizeListPtr(int exp)` *returns a pointer to the array of sampling step sizes for the experiment* `exp`*.*

Function 6.12 `double *glbGetBinCentersListPtr(int exp)` *returns a pointer to the array of mean bin energies for the experiment* `exp`*.*

Function 6.13 `double *glbGetSamplingPointsListPtr(int exp)` *returns a pointer to the array of sampling points for the experiment* `exp`*.*

In order to obtain information on the structure of the rules, a number of functions are provided:

Function 6.14 `int glbGetNumberOfRules(int exp)` *returns the number of rules in experiment* `exp`*.*

Function 6.15 `int glbGetLengthOfRule(int exp, int rule, int signal)` *returns the length of rule* `rule` *in experiment* `exp`*. The parameter* `signal` *can be either* `GLB_SIG` *for the number of signal components or* `GLB_BG` *for the number of background components.*

Function 6.16 `int glbGetChannelInRule(int exp, int rule, int pos, int signal)` *returns the channel number in rule* `rule` *at the position* `pos` *of the experiment* `exp`*. The parameter* `signal` *refers to signal (*`GLB_SIG`*) or background (*`GLB_BG`*).*

Function 6.17 `double glbGetCoefficientInRule(int exp, int rule, int pos, int signal)` *returns the coefficient of the component* `pos` *in rule* `rule` *of the experiment* `exp`*. The parameter* `signal` *refers to signal (*`GLB_SIG`*) or background (*`GLB_BG`*).*

Similarly to the rules, one can find the number of channels of an experiment:

Function 6.18 `int glbGetNumberOfChannels(int exp)` *returns the number of channels of experiment* `exp`*.*

For each channel, the efficiencies and backgrounds can be returned with the following functions:

Function 6.19 `double *glbGetEfficiencyPtr(int exp, int ch, int pre_post)` *returns a pointer to the efficiency list for experiment* `exp` *and channel* `ch`*. The pre- or post-smearing efficiencies are returned with* `pre_post` *set to* `GLB_PRE` *and* `GLB_POST`*, respectively.*

Function 6.20 `double *glbGetBackgroundPtr(int exp, int ch, int pre_post)` *returns a pointer to the background list for experiment* `exp` *and channel* `ch`*. The pre- or post-smearing backgrounds are returned with* `pre_post` *set to* `GLB_PRE` *and* `GLB_POST`*, respectively.*

Since in AEDL rules, cross section, fluxes etc. carry a 'name' by which they can be referred to, while in C they carry only an integer index, it is sometimes difficult to figure out the correct correspondence. Therefore the information about this correspondence obtained during parsing is stored and can be accessed within C by the following two functions.

Function 6.21 `int glbNameToValue(int exp, const char* context, const char *name)` *Converts an* AEDL *name given as argument* `name` *into the corresponding C index. The variable* `context` *describes wether this name belongs to a rule, channel, flux, energy, or cross type environment.* `exp` *is the number of the experiment and can not be* `GLB_ALL`*. It returns either the index in case of success or* `-1` *in case the name was not found.*

Function 6.22 `const char *glbValueToName(int exp,const char* context, int value)` *Converts a C index given as argument* `value` *into the corresponding* AEDL *name. The variable* `context` *describes wether the index belongs to a rule, channel, flux, energy or cross type environment.* `exp` *is the number of the experiment and can not be* `GLB_ALL`*. It returns either the name in case of success or* `NULL` *in case the name was not found. The returned string must not be modified.*

6.3 Event rates*

One can also return event rates in GLoBES, but this feature requires some knowledge about the experiment definition. In fact, many of these functions are very advanced, which means that the reader who wants to use them should be familiar with Secs. 11.4 and Sec. 11.6 of the AEDL manual. Note that parts of the event rate access have changed in GLoBES 3.0, because user-defined systematics require very fast access, which was not possible with the old method.

A very simple function is for the total rate

Function 6.23 `double glbTotalRuleRate(int exp, int rule, int pos, int effi, int bgi, int coeffi, int signal)` *returns the total rates. A specific experiment* `exp` *and a specific rule* `rule` *have to be chosen, as well as the signal or background rate* `signal` *(either* `GLB_SIG` *or* `GLB_BG`*). The position* `pos` *refers to the component within the signal or background, and can also be* `GLB_ALL`*. The function may return the rates with (*`GLB_W_COEFF`*) or without (*`GLB_WO_COEFF`*) overall efficiency coefficient, as it is specified by* `coeffi`*. In addition, it may contain the post-smearing efficiencies (set* `effi` *to* `GLB_W_EFF` *or* `GLB_WO_EFF`*), and the post-smearing backgrounds (set* `bgi` *to* `GLB_W_BG` *or* `GLB_WO_BG`*). The pre-smearing efficiencies and backgrounds cannot be accessed at the rule level.*

The function `glbTotalRuleRate` is especially useful if one wants to draw bi-rate graphs with total event rates, or look for the $(\delta_{\mathrm{CP}}, \theta_{13})$-degeneracy by the intersection of neutrino and antineutrino constant event rate curves.

There are several functions to directly print or save the event rate information:

Function 6.24 `int glbShowRuleRates(FILE* stream, int exp, int rule, int pos, int effi, int bgi, int coeffi, int signal)` *prints the binned rule rates as a list with energy and event rate to the file* `stream` *(either an open file, or* `stdout`*). A specific experiment* `exp` *and a specific rule* `rule` *have to be chosen, as well as the* *signal or background rate* signal *(either* GLB_SIG *or* GLB_BG*). The position* pos *refers to the component within the signal or background, and can also be* GLB_ALL*. The function may return the rates with (*GLB_W_COEFF*) or without (*GLB_WO_COEFF*) overall efficiency coefficient, as it is specified by* coeffi*. In addition, it may contain the post-smearing efficiencies (set* effi *to* GLB_W_EFF *or* GLB_WO_EFF*), and the post-smearing backgrounds (set* bgi *to* GLB_W_BG *or* GLB_WO_BG*). The pre-smearing efficiencies and backgrounds cannot be accessed at the rule level. The return value is* 0 *if successful, and* −1 *if unsuccessful.*

Function 6.25 int glbShowChannelRates(FILE *stream, int exp, int channel, int smearing, int effi, int bgi) *prints the binned channel rates as a list with energy and event rate to the file* stream *(either an open file, or* stdout*). A specific experiment* exp *and a specific channel* channel *have to be chosen. The function may return the rates before (*GLB_PRE*) or after (*GLB_POST*) the energy smearing, as it is specified by* smearing*. In addition, it may contain the pre- and post-smearing efficiencies (set* effi *to* GLB_W_EFF *or* GLB_WO_EFF*), and the pre- and post-smearing backgrounds (set* bgi *to* GLB_W_BG *or* GLB_WO_BG*). Note that the post-smearing efficiencies and backgrounds cannot be taken into account if the rates are returned before the energy smearing. The return value is* 0 *if successful, and* −1 *if unsuccessful.*

For rate vectors, GLoBES currently supports rule-based and channel-based event rate functions, where typically pointers on the rate vectors are returned. The following pointer-based functions are currently supported:

Function 6.26 double *glbGetChannelRatePtr(int exp, int ch, int pre_post) *returns a pointer to the simulated rate vector of experiment* exp *and channel* ch*. Either pre-smearing (*pre_post *is* GLB_PRE*) or post-smearing (*pre_post *is* GLB_POST*) rates can be accessed.*

Function 6.27 double *glbGetRuleRatePtr(int exp, int rule) *returns a pointer to the simulated rate vector of experiment* exp *and rule* rule*.*

Function 6.28 double *glbGetSignalRatePtr(int exp, int rule) *returns a pointer to the simulated signal rate vector of experiment* exp *and rule* rule*.*

Function 6.29 double *glbGetBGRatePtr(int exp, int rule) *returns a pointer to the simulated background rate vector of experiment* exp *and rule* rule*.*

Function 6.30 double *glbGetChannelFitRatePtr(int exp, int ch, int pre_post) *returns a pointer to the fit rate vector of experiment* exp *and channel* ch*. Either pre-smearing (*pre_post *is* GLB_PRE*) or post-smearing (*pre_post *is* GLB_POST*) rates can be accessed.*

Function 6.31 double *glbGetSignalFitRatePtr(int exp, int rule) *returns a pointer to the fit signal rate vector of experiment* exp *and rule* rule*.*

Function 6.32 `double *glbGetBGFitRatePtr(int exp, int rule)` *returns a pointer to the fit background rate vector of experiment* `exp` *and rule* `rule`*.*

A simple example how to use these functions to print a rate vector is

```
int i;
int n_bins = glbGetNumberOfBins(EXP_FAR);
double *true_rates_N = glbGetRuleRatePtr(0, 0);
printf("Simulated rates, experiment 0, rule 0:  \n");
for(i=0;i<n_bins;i++) printf("% g ",true_rates_N[i]);
printf("\n");
```

6.4 Fluxes and cross sections*

Another piece of low-level information, which can be returned by GLoBES, are the numbers from the loaded fluxes and cross sections. The following functions interpolate on the loaded fluxes and cross sections, *i.e.*, any value in the allowed energy range can be given as input:

Function 6.33 `double glbFlux(int exp, int ident, double E, double distance, int l, int anti)` *returns the flux of flux number* `ident` *of the experiment* `exp` *for the flavor ν_l and polarity* `anti` *($+1$: neutrinos, -1: antineutrinos) at the energy* `E` *and distance* `distance`*.*

Function 6.34 `double glbXSection(int exp, int ident, double E, int l, int anti)` *returns the cross section of experiment* `exp`*, cross section number* `ident` *for the flavor ν_l and polarity* `anti` *($+1$: neutrinos, -1: antineutinos) at the energy* `E`*.*

The number of fluxes can be obtained with

Function 6.35 `int glbGetNumberOfFluxes(int exp)` *returns the number of fluxes defined in experiment* `exp`*.*

Chapter 7

Changing experiment parameters at running time

Many of the parameters in experiment definitions can be changed at running time. For example, we have introduced in Sec. 2.2 possibilities to change the integrated luminosity, which consists of source power, running time, and target mass. In this chapter, we discuss more sophisticated experiment changes. However, since GLoBES computes a lot of information only once when an experiment is loaded, many parameters can not be changed at running time. For example, the energy resolution function or the number of bins are used to compute the smearing matrix already at the initialization of the experiment, which saves a lot of computation time for most applications. In Sec. 7.3, we introduce a mechanism how one can even change these AEDL parameters during running time.

7.1 Systematics

Changing the systematics at running time can be useful to investigate the impact factors affecting the measurement. In GLoBES, the systematics is defined rule-based, *i.e.*, each rule has its own systematics. In addition, GLoBES supports dual systematics, *i.e.*, AEDL requires that it has to be defined in each rule what "Systematics on" and "Systematics off" means. In principle, these two sets correspond to two completely different systematics implementations, and it is up to the AEDL authors to define what that means. From the point of view of the API, it is very simple to switch the systematics on and off, *i.e.*, to switch between the two systematics modes:

Function 7.1 `int glbSwitchSystematics(int exp, int rule, int on_off)` *switches the systematics in experiment* `exp` *and rule* `rule` *on (*`on_off` *is* `GLB_ON`*) or off (*`on_off` *is* `GLB_OFF`*). For the experiment or rule index, one can also use* `GLB_ALL`*.*

One can also return the systematics state with

Function 7.2 `int glbGetSysOnOffState(int exp, int rule)` *returns the systematics state in experiment* `exp` *and rule* `rule`*.*

In the example on page 59, the application of `glbSwitchSystematics` is demonstrated to show the impact of systematics, correlations, and degeneracies.

The following material requires knowledge of AEDL, which means that it can be skipped at a first reading. During running time, it is possible to change the systematics of an experiment or rule (as compared to the systematics assigned in the AEDL file) with the following function:

Function 7.3 `int glbSetChiFunction(int exp, int rule, int on_off, const char *name, double *errors)` *tells* GLoBES *to use the registered user-defined systematics identified by the string* `name` *or any built-in systematics to calculate* χ^2 *for the experiment* `exp` *and the rule* `rule`*. Both of the parameters* `exp` *and* `rule` *can take the value* `GLB_ALL` *to specify that the given systematics function should be used for each experiment or each rule. The parameter* `on_off` *determines if the systematics function should be used when systematical errors are switched on (*`GLB_ON`*), or when they are switched off (*`GLB_OFF`*). The array* `errors` *sets the systematical errors in the order in which they are expected by the systematics function (indices run from 0 to the number of systematics parameters-1). The function returns zero, if successful.*

Note that user-defined systematics functions have to be registered with `glbDefineChiFunction` first. One can also request the systematics function by

Function 7.4 `int glbGetChiFunction(int exp, int rule, int on_off, char *sys_id, size_t max_len)` *returns the name of the systematics* χ^2 *function of a given experiment* `exp` *and rule* `rule` *for systematics on or off as given by* `on_off`*. The name is copied to the string* `sys_id`*, the maximum length of which is specified by* `max_len`*. If* `max_len` *is too small, or if any other error occurs, the return value is* < 0*.*

Except from the general treatment of systematics, one can read out and change the signal and background errors for standard pre-defined systematics during running time:

Function 7.5 `int glbSetSignalErrors(int exp, int rule, double norm, double tilt)` *sets the signal errors of experiment* `exp` *and rule* `rule` *to* `norm` *(normalization error) and* `tilt` *(tilt/calibration error).*

Function 7.6 `int glbGetSignalErrors(int exp, int rule, double* norm, double* tilt)` *writes the signal errors of experiment* `exp` *and rule* `rule` *to* `norm` *(normalization error) and* `tilt` *(tilt/calibration error).*

Function 7.7 `int glbSetBGErrors(int exp, int rule, double norm, double tilt)` *sets the background errors of experiment* `exp` *and rule* `rule` *to* `norm` *(normalization error) and* `tilt` *(tilt/calibration error).*

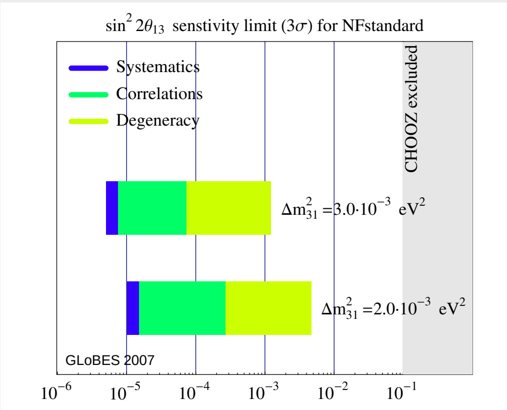
Function 7.8 `int glbGetBGErrors(int exp, int rule, double* norm, double* tilt)` *writes the background errors of experiment* `exp` *and rule* `rule` *to* `norm` *(normalization error) and* `tilt` *(tilt/calibration error).*

Example: The impact of systematics, correlations, and degeneracies

Here, we demonstrate how systematics, correlations, and degeneracies can be successively included in the calculation of the $\sin^2 2\theta_{13}$-sensitivity limit. The following code fragment shows how systematics can be switched off in order to compute the sensitivity limit from statistics only:

```
/* Calculate chi2 with statistics only */
double CalcNoSystematics(double theldm,double thex)
{
   /* Switch systematics off for all exps and all rules */
   glbSwitchSystematics(GLB_ALL,GLB_ALL,GLB_OFF);

   /* Calculate Chi2-list as if systematics were on */
   double res=CalcSystematics(theldm,thex);

   /* Switch systematics on for all exps and all rules */
   glbSwitchSystematics(GLB_ALL,GLB_ALL,GLB_ON);

   return res;
}
```

The complete code is very advanced and can be found in `example4.c`. It includes many concepts from earlier examples, and, in addition, it uses a little trick: It avoids falling into the wrong solution with `glbChiTheta13` by using the fit value of δ_{CP} from the step before as prediction for the position of the minimum in the current calculation.
The returned lists of data from the example represent χ^2 as function of the fit value of $\sin^2 2\theta_{13}$. The intersections of these curves with the line $\chi^2 = 9$ give the $\sin^2 2\theta_{13}$ sensitivity limits at the 3σ confidence level. Note that, in the following plot, the $\mathrm{sgn}(\Delta m_{31}^2)$- and $(\delta_{\mathrm{CP}}, \theta_{13})$-degeneracies are not included in the sensitivity limit with correlations only (green bar), but only in the limit with degeneracies (yellow bar):



As a more flexible concept, the systematical errors may be given as lists for both user-defined and built-in systematics. For example, the standard signal and background pre-defined errors can be accessed as lists with four elements (signal normalization, signal tilt, background normalization, background tilt) if built-in systematics is used. The corresponding functions are:

Function 7.9 `int glbSetSysErrorsList(int exp, int rule, int on_off, const double *sys_list)` *changes the systematical errors defined in the* AEDL *file in experiment* `exp` *and rule* `rule` *to the values in* `sys_list`*. This change can be perfomed for systematics on or systematics off by using* `GLB_ON` *or* `GLB_OFF` *for* `on_off`*.*

Function 7.10 `double* glbGetSysErrorsListPtr(int exp, int rule, int on_off)` *returns a pointer to the list of systematical errors as defined in the* AEDL *file in experiment* `exp` *and rule* `rule`*. Choose systematics on or systematics off by using* `GLB_ON` *or* `GLB_OFF` *for* `on_off`*.*

As usual, all these functions return -1 if they were not successful. For user-defined systematics, see Sec. 3.2, and for the definitions of these quantities, see Sec. 11.6.

7.2 Baseline and matter density profile*

In order to change the baseline of an experiment, it is important to keep in mind that each experiment has a profile type defined in the AEDL file (average density, PREM profile with a given number of steps, or arbitrary profile; *cf.*, Table 11.2). One can check the currently used profile type with

Function 7.11 `int glbGetProfileTypeInExperiment(int exp)` *returns the matter density profile type of experiment* `exp`*.*

For each profile type, one can easily change the baseline with `glbSetBaselineInExperiment`, where the average density or the PREM profile are re-computed, or the steps in the arbitrary profile are re-scaled. If this behavior is not the desired one, one has to use `glbSetProfilDataInExperiment` as explained below.

Function 7.12 `int glbSetBaselineInExperiment(int exp, double baseline)` *sets the baseline length in experiment* `exp` *to* `baseline`*. The function returns* -1 *if it was not successful.*

Note that `glbSetBaselineInExperiment` does not change the profile type in the experiment. The counterpart of this function is:

Function 7.13 `double glbGetBaselineInExperiment(int exp)` *returns the baseline length currently used for experiment* `exp`*.*

One can not change the profile type of an experiment manually during running time. However, one can change the matter density profile, where the profile type is automatically switched to 3, *i.e.* arbitrary matter density profile. In addition, a number of functions are provided to compute possible matter density profiles (average density, PREM profile). In general, a matter density profile in GLoBES with N layers is represented by a list of lengths

$$\text{Lengths} = (x_1, x_2, \ldots, x_N) \tag{7.1}$$

and a list of densities

$$\text{Densities} = (\rho_1, \rho_2, \ldots, \rho_N), \tag{7.2}$$

where the baseline is given by

$$L = \sum_{i=1}^{N} x_i. \tag{7.3}$$

In C, lists are represented as pointers to the first element:

```
double* lengths;
double* densities;
```

Many of the GLoBES baseline functions take and return such lists as parameters, and are therefore more sophisticated to handle. In general, any function *returning* lists allocates the memory for them. It is then up to the user to free this memory! In addition, they normally provide the length of the lists N by means of an additional argument which is a pointer to `size_t`. Normally, it is enough to declare a variable of the type `size_t` and to give its address to the function. The following functions return matter density profiles:

Function 7.14 `int glbLoadProfileData(const char* filename, size_t *layers, double **lengths, double **densities)` *loads a density file from the file* `filename`*. It returns the number of layers* `layers`*, the list of lengths* `lengths`*, and the list of densities* `densities`*.*

The file should contain in each line a length and density for one layer, which are separated by an empty space.

Function 7.15 `int glbStaceyProfile(double baseline, size_t layers, double **lengths, double **densities)` *creates a PREM/Stacey matter density profile with a number of* `layers` *steps for the baseline* `baseline`*. The list of lengths* `lengths` *and the list of densities* `densities` *are returned.*

Note that this function does not interpolate or average within individual layers.

Function 7.16 `glbAverageDensityProfile(double baseline, double **lengths, double **densities)` *creates a average matter density profile from the PREM/Stacey profile with one step for the baseline* `baseline`*. The list of lengths* `lengths` *and the list of densities* `densities` *are returned.*

The average matter density $\bar{\rho}(L)$ for a matter density profile $\rho(x)$ along the baseline L baseline is defined as

$$\bar{\rho}(L) = \frac{1}{L} \int\limits_0^L \rho(x) dx = \frac{1}{L} \int\limits_0^L \tilde{\rho}\left(d(x)\right) dx \,, \tag{7.4}$$

where $\tilde{\rho}(d)$ is the PREM matter density as function of the distance d to the Earth's core, and $d(x) = \sqrt{x^2 + R^2 - 2xR\cos\theta}$ is the purely geometrical relationship between d and x with the Earth radius R and the nadir angle $\cos\theta = L/(2R)$.

Function 7.17 `int glbGetProfileDataInExperiment(int exp,size_t *layers, double** lengths, double** densities)` *returns the matter density profile currently used for experiment* `exp`*. The number of layers* `layers`*, the list of lengths* `lengths`*, and the list of densities* `densities` *are returned.*

All these functions return -1 if they were not successful.

The counterpart of these functions to assign a specific matter density profile to an experiment is

Function 7.18 `int glbSetProfileDataInExperiment(int exp, size_t layers,const double* lengths, const double* densities)` *sets the matter density of experiment* `exp` *to an arbitrary profile with* `layers` *steps. The density layers are specified by the lists* `lengths` *and* `densities`*. The function returns -1 if it was not successful.*

Finally, let us take a look at two examples. This example changes the baseline length to 7 500 km, where the average matter density is manually computed:

```
double* lengths;
double* densities;
glbAverageDensityProfile(7500,&lengths,&densities);
glbSetProfileDataInExperiment(0,1,lengths,densities);
free(lengths);
free(densities);
```

In the second example, we change the baseline to a PREM profile with 100 matter density steps and print them:

```
double* lengths;
double* densities;
glbStaceyProfile(7500,100,&lengths,&densities);
int i;
for(i=0;i<100;i++) printf("%g %g \n",lengths[i],densities[i]);
glbSetProfileDataInExperiment(0,100,lengths,densities);
free(lengths);
free(densities);
```

7.3 External parameters in AEDL files*

Using external parameters in AEDL files is a very powerful feature to change experiment parameters at running time, which requires, however, that the experiment be re-initialized. For example, one can change the energy resolution function or the number of energy bins. However, in some cases, there might be complications, such that the number of pre- or post-smearing efficiencies does not correspond to the number of energy bins anymore. Therefore, this feature needs to be used with care.

In order to use external parameters in AEDL files, one simply introduces them. For example, an energy resolution function

```
energy(#EnergyResolution1)<
      type = 1
      @sigma_e = { myres ,0,0 }
>
```

might be defined in AEDL, where the energy resolution is proportional to `myres` $\times$ energy.

In order to use the user-defined variable, one has to assign it with `glbDefineAEDLVariable` *before* the experiment is initialized with `glbInitExperiment`:

Function 7.19 `void glbDefineAEDLVariable(const char* name, double value)` *assigns the value* `value` *to the* AEDL *variable* `name`*.*

In our energy resolution example, one could now loop over the energy resolution such as with

```
int i;
for(i=5;i<20;i++)
{
      glbClearExperimentList();
      glbDefineAEDLVariable("myres",0.01*i);
      glbInitExperiment(...);

      /* do something */
}
```

Note that one does not have to re-initialize the oscillation parameter vectors every time within the loop as long as the number of experiments does not change.

Similar to a simple AEDL variable, one can transfer the value of an AEDL list with (*cf.*, Sec. 10.3)

Function 7.20 `glbDefineAEDLList(const char *name, double *list, size_t length)` *assigns the list* `list` *of length* `length` *to the* AEDL *variable* `name`*.*

See Sec. 7.2 for how to use such lists. In order to clear the external variable stack if one is excessively using it, one can use

Function 7.21 `void glbClearAEDLVariables()` *clears the* AEDL *variable list.*

This function is called automatically upon exit of the program.

7.4 Algorithm parameters: Filter functions*

The oscillation frequency filters to filter fast oscillations can also be accessed by the user interface. For details of the filter functions, we refer to Sec. 11.5 of the AEDL manual.

In particular, there are a number of functions:

Function 7.22 `int glbSetFilterStateInExperiment(int exp, int on_off)` *sets the filter state in experiment* `exp` *to on (*`GLB_ON`*) or off (*`GLB_OFF`*).*

Function 7.23 `int glbGetFilterStateInExperiment(int exp)` *returns the filter state of experiment* `exp`*.*

Analogously, the filter value can be accessed:

Function 7.24 `int glbSetFilterInExperiment(int exp, double filter)` *sets the filter in experiment* `exp` *to the value* `value`*.*

Function 7.25 `double glbGetFilterInExperiment(int exp)` *returns the filter value of experiment* `exp`*.*

The return value of all `Set-` functions is -1 if they were not successful.

Chapter 8

Simulating non-standard physics*

In this chapter, we discuss how to simulate non-standard physics with GLoBES, *i.e.*, physics beyond the standard three-flavor neutrino oscillation scenario. For the first time, this feature was used as experimental feature in Ref. [19], and it has become a standard feature of GLoBES starting from version 3.0. Since the computation of oscillation probabilities, and therefore the flavor transition probabilities, is the core basic element of GLoBES, the introduction of non-standard physics requires familiarity with the probability calculation in GLoBES. It is therefore an advanced topic. Below, we will demonstrate how and where to do the necessary modifications, and how to simulate non-standard physics in the application software.

8.1 Modification of GLoBES

GLoBES 3.0 and higher does not require a re-compilation of the software to simulate non-standard physics. However, the probability engine has to be changed, which one would usually do by copying the respective parts from `glb_probability.c` (in the `source` subdirectory). Therefore, for sophisticated applications, familiarity with this file is required. A number of comments are provided in this file to keep it self-explanatory. The key function types are

Function 8.1 `int (*glb_set_oscillation_parameters_function)(glb_params p, void* user_data)` *is used to pass the fundamental oscillation parameters* `p` *to the probability engine. The function should store them into variables accessible to* `glb_probability_matrix_function`*. In addition, it can be used to pre-compute quantities which are dependent on the oscillation parameters only, such as the mixing matrix. In order to circumvent global variables, arbitrary additional parameters can be passed to the function in* `user_data`*, which is set by* `glbRegisterProbabilityEngine`*.*

Function 8.2 `int (*glb_get_oscillation_parameters_function)(glb_params p, void* user_data)` *reads the fundamental oscillation parameters from the internal variables of the probability engine, and writes them into* `p`*. In order to circumvent global* *variables, arbitrary additional parameters can be passed to the function in* `user_data`*, which is set by* `glbRegisterProbabilityEngine`*.*

Function 8.3 `int (*glb_probability_matrix_function)(double P[3][3], int cp_sign, double E, int psteps, const double *lengths, const double *densities, double filter_sigma, void* user_data)` *calculates the neutrino oscillation probability matrix and returns it in* `P`*. The* `cp_sign` *is +1 for neutrinos and -1 for antineutrinos. In addition, the matter density profile to be used is characterized by the number of steps* `psteps`*, the lengths of the matter density layers in the list* `lengths`*, and the corresponding densities in the list* `densities`*. The parameter* `filter_sigma` *defines the width of the low-pass filter to be used, or no filter usage if negative. In order to circumvent global variables, arbitrary additional parameters can be passed to the function in* `user_data`*, which is set by* `glbRegisterProbabilityEngine`*.*

These three function types correspond to the standard implementation functions `glb_set_oscillation_parameters`, `glb_get_oscillation_parameters`, and `glb_probability_matrix` in `glb_probability.c` of the source code of GLoBES, where one can find the standard behavior and use it by cut and paste (including the variables and code used by these functions).

In order to use non-standard physics in GLoBES, one needs to re-define the three above functions and register them after `glbInit` with

Function 8.4 `int glbRegisterProbabilityEngine(int n_parameters, glb_probability_matrix_function prob_func, glb_set_oscillation_parameters_function set_params_func, glb_get_oscillation_parameters_function get_params_func, void* user_data)` *registers a probability engine for the simulation of non-standard physics with* `n_parameters` *oscillation parameters (*`n_parameters` ≥ 6*). The three functions* `prob_func`*,* `set_params_func`*, and* `get_params_func` *will be used with the function types defined above. In order to circumvent global variables, an arbitrary pointer* `user_data` *can be defined, that will be passed to the probability engine in each function call.*

The number of oscillation parameters can, at any time, be obtained with

Function 8.5 `int glbGetNumOfOscParams()` *returns the number of oscillation parameters.*

Let us now illustrate the implementation of non-standard physics with a simple example, which can be found as `example6.c` in the example directory. This example is a simplified version of Sec. 4 of Ref. [19]. It uses an analytical probability calculation for a reactor experiment with a baseline being treated in vacuum for simplicity. The non-standard effect is the loss of coherence because of wave packet decoherence or any other such effect, *i.e.*, we have one additional oscillation parameter. We need to define two functions to access a set of global oscillation parameters

```
double th12, th13, th23, deltacp, sdm, ldm, sigma_E;

int my_set_oscillation_parameters(glb_params p, void *user_data)
{
   th12 = glbGetOscParams(p, GLB_THETA_12);
   th13 = glbGetOscParams(p, GLB_THETA_13);
   th23 = glbGetOscParams(p, GLB_THETA_23);
   deltacp = glbGetOscParams(p, GLB_DELTA_CP);
   sdm = glbGetOscParams(p, GLB_DM_21) * 1.0e-18;
   ldm = glbGetOscParams(p, GLB_DM_31) * 1.0e-18;
   sigma_E = glbGetOscParams(p, GLB_SIGMA_E);

   return 0;
}

int my_get_oscillation_parameters(glb_params p, void *user_data)
{
   glbSetOscParams(p, th12, GLB_THETA_12);
   glbSetOscParams(p, th13, GLB_THETA_13);
   glbSetOscParams(p, th23, GLB_THETA_23);
   glbSetOscParams(p, deltacp, GLB_DELTA_CP);
   glbSetOscParams(p, sdm*1.0e18, GLB_DM_21);
   glbSetOscParams(p, ldm*1.0e18, GLB_DM_31);
   glbSetOscParams(p, sigma_E, GLB_SIGMA_E);

   return 0;
}
```

Furthermore, we need to define the probability matrix calculation itself – in this case, we perform an analytical calculation:

```
int my_probability_matrix(double P[3][3], int cp_sign, double E,
int psteps, const double *length, const double *density, double
filter_sigma, void *user_data)
{
   int i, j;
   double L, Delta21, Delta31, Delta32;
   double D21, D31, D32, s13, c13, s12, c12, t;

   /* Set all probabilities to zero initially */
   for (i=0; i < 3; i++) for (j=0; j < 3; j++) P[i][j] = 0.0;

   /* Calculate total baseline */
   L = 0.0; for (i=0; i < psteps; i++) L += length[i];
```

```
   L = KM_TO_EV(L) * 1.0e9; /* Convert to GeV^-1 */

   /* Compute P_ee */
   s12 = sin(th12); c12 = cos(th12);
   s13 = sin(th13); c13 = cos(th13);
   t = L / (4.0 * E);
   Delta21 = sdm * t; Delta31 = ldm * t;
   Delta32 = Delta31 - Delta21;
   t = M_SQRT2 * sigma_E / E;
   D21 = exp(-square( Delta21 * t ));
   D31 = exp(-square( Delta31 * t ));
   D32 = exp(-square( Delta32 * t ));
   P[0][0] = square(square(c13)) * ( 1 - 2.0*square(s12*c12)*
     (1 - D21*cos(2.0*Delta21)) )      + 2.0*square(s13*c13) *
     ( D31*square(c12)*cos(2.0*Delta31)
     + D32*square(s12)*cos(2.0*Delta32) ) + square(square(s13));

   return 0;
}
```

Now we can register the probability engine after `glbInit`:

```
glbInit(argv[0]);
glbRegisterProbabilityEngine(7, /* Number of parameters */
     &my_probability_matrix,
     &my_set_oscillation_parameters,
     &my_get_oscillation_parameters,
     NULL);
```

We will demonstrate in the next section how to access the non-standard physics parameter in the application software.

8.2 Using non-standard physics in the application software

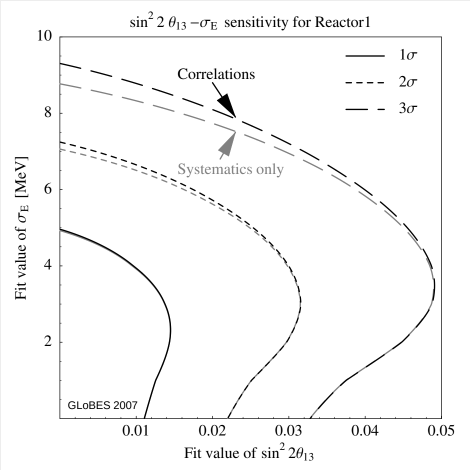
Using more than six parameters in GLoBES, you will have to maintain the additional parameters. For example, there are no standard routines in GLoBES which define projections including more than six parameters, which means that you should only use `glbChiNP` further on in order to have a defined behavior of the projection. In addition, you can still use `glbDefineParams`, but this will only access the six standard parameters. You will need to set the additional ones manually using `glbSetOscParams`. Do not forget to maintain your non-standard parameters, since any negligence will be punished by un-predicted behavior!

Example: Decoherence in neutrino oscillations

The following code fragment from `example6.c` calculates a fit region on $\sin^2 2\theta_{13}$-σ_E-space, where σ_E is the non-standard decoherence parameter (*cf.*, Ref. [19]):

```
const int GLB_SIGMA_E = 6;

/* Fix theta13 and sigma_E, and marginalize over all other parameters
except deltaCP and th23, which do not enter P_ee */
glbDefineProjection(myproj, GLB_FREE, GLB_FIXED, GLB_FIXED, GLB_FIXED,
    GLB_FREE, GLB_FREE);
glbSetDensityProjectionFlag(myproj, GLB_FIXED, GLB_ALL);
glbSetProjectionFlag(myproj, GLB_FIXED, GLB_SIGMA_E);
glbSetProjection(myproj);

for(x=0; x < 0.05+0.001; x+=0.005) /* th13 loop */
for(y=0.0; y < 0.010+0.001; y+=0.001) /* sigma_E loop */
{
    /* Set vector of test=fit values */
    thetheta13=asin(sqrt(x))/2.0;
    glbSetOscParams(test_values,thetheta13,GLB_THETA_13);
    glbSetOscParams(test_values,y,GLB_SIGMA_E);

    /* Compute Chi2 with correlations */
    res=glbChiNP(test_values,NULL,GLB_ALL);

    AddToOutput(x,y,res);
}
```

The result is represented by the dark curves in the following figure (similar to Ref. [19]):



In order to use the non-standard parameters, GLoBES creates any set of oscillation parameters (type `glb_params`) with the additional parameters. To access the non-standard parameters, you can use the following functions as usual:

- `glbSetOscParams`
- `glbGetOscParams`
- `glbSetProjectionFlag`
- `glbGetProjectionFlag`

In all these cases, the last parameter `which` can now run from 0 to `glbGetNumOfOscParams()-1`. For instance,

```
glbDefineParams(true_values,0.55,0.15,0.78,0.0,0.000082,0.0022);
glbSetDensityParams(true_values,1.0,GLB_ALL);
glbSetOscParams(true_values,0.0,6);
```

sets all of the oscillation parameters if you have one additional parameter. In addition, the function `glbCopyParams` can be used to copy all parameters inluding the non-standard ones. For projections, it is highly recommended not to use any of the `glbChi...` functions anymore except from `glbChiNP`, in order to have a predictive behavior of the "extra" dimensions.[1] Similar to the oscillation parameters, you can define the marginalization over the extra parameter(s) by `glbSetProjectionFlag`. You can find a simple example using non-standard physics with the code from the last section on page 69.

[1]It is easy to write headers for new functions with a self-defined behavior with respect to the new dimensions using `glbChiNP`.

Chapter 9

Experimental features*

Here we describe experimental features currently being implemented in GLoBES. These features have not yet been tested extensively and should be used with care. They may evolve into standard features in the future, or they may not be supported anymore at a certain point. In general, the number of experimental features is small in a new release version, and increases towards a new version number.

GLoBES 3.0 and higher

One can change the minimization algorithm in GLoBES with

Function 9.1 `int glbSelectMinimizer(int minimizer_ID)` *selects the minimizer* `GLB_MIN_NESTED_POWELL`*,* `GLB_MIN_DEFAULT`*, or the hybrid minimizer* `GLB_MIN_POWELL` *by the parameter* `minimizer_ID`*.* `GLB_MIN_NESTED_POWELL` *is the currently implemented standard algorithm,* `GLB_MIN_DEFAULT` *chooses the standard algorithm at any given time, and* `GLB_MIN_POWELL` *is a faster, yet rarely tested hybrid minimizer.*

Compared to the standard minimization which is performed for systematics first and then for the oscillation parameters, the hybrid minimizer mixes the systematics and oscillation parameter minimizations. This method is much faster, but correlations between systematics parameters and oscillation parameters might lead to a different convergence behavior in some situations. Thus, when switching to the hybrid minimizer in an application program, one has to reconsider the question whether all degeneracies are found. Since one can change the minimizer at any time, it is recommended that one cross check the minimization for a particular experiment. The danger of modifying the convergence behavior in existing application programs is another reason why the new, faster minimizer is still declared as experimental and not used by default. Note that the meaning of the number of iterations in `glb_params` changes for the hybrid minimizer. Since the systematics and oscillation parameter minimizations are not strictly separated anymore, the minimizer does not count the iterations separately, and the total number of iterations is returned. Therefore, it appears that the hybrid minimizer needs more iterations, but, in fact, the default minimizer only counts the oscillation parameter level.

Part II

The Abstract Experiment Definition Language – AEDL

Chapter 10

Getting started

Here, the general concept of AEDL is described and illustrated by an example. In addition, a short introduction to the syntax of AEDL is given.

10.1 General concept of the experiment simulation

The goal of AEDL is to describe a large number of complex and very different experiments by a limited number of parameters. It allows a representation of very different setups within one data structure, and thus implements universal rate and χ^2 computation methods. For experiment simulations, usually a new piece of code is written and compiled for each different experiment. In many cases, even parameter changes, such as the number of bins, require the recompilation of the source code. However, such a technique soon reaches its limits when the simulated experiments are rather complex, or more than one type of experiment is studied simultaneously. Furthermore, it is very difficult to verify the correctness of the obtained results, since every time a new piece of code is added to deal with a new experiment type, new errors will be introduced.

Thus, a general and flexible experiment description language is needed. The description of a neutrino experiment can be split into three parts: Source, oscillation, and detection. The neutrino sources within GLoBES are assumed to be stationary point sources, where each experiment has only one source. This restricts the classes of neutrino sources which can be studied with GLoBES:

- Experiments using many point-like sources can only be approximated. One example are reactor experiments using many distant reactor blocks.
- Geometrical effects of a source distribution, such as in the sun or the atmosphere, can not be described.
- Sources with a physically significant time dependency can not be studied, such as supernovæ. It is, however, possible to study beams with bunch structure, since the time dependence of the neutrino source is physically only important to suppress backgrounds.

The description of the neutrino oscillation physics is, at least numerically, relatively simple. We use the *evolution operator method* (see, *e.g.*, Ref. [20]) to compute the neutrino oscillation probabilities and divide the matter density profile into layers of constant matter density. For each of these layers, the Hamiltonian in matter is diagonalized in order to propagate the neutrino transition amplitudes. Since this step is computationally expensive, a specialized algorithm is used [21]. Finally, the transition probability is obtained as the absolute square of the total neutrino transition amplitudes. Depending on the precision of the studied experiment, this approach turns out to be precise enough in Earth matter even if only a small number of matter density steps is used. Since we allow an uncertainty of the matter density profile, it is, in fact, in most cases sufficient to consider only one density step with the average matter density together with a matter density uncertainty [22]. Note that this approach may not be applicable to quickly varying extraterrestrial matter density profiles.

While it is comparatively simple to define a general neutrino source and to compute the oscillation physics, the general properties of a detector simulation are much more complicated. The basic assumption in building an abstract detector description is *linearity*, *i.e.*, that two neutrino events do not interfere with each other. Furthermore it is assumed that all information on the oscillation physics is given by the *reconstructed* flavor and energy of a neutrino event. The term "reconstructed" implies that the well-defined energy of the incident neutrino, which can not be directly observed, translates via secondary particles and the detection properties into a distribution of possible energy values. This process is illustrated in Fig. 10.1 for the energy variable. The same, in principle, applies to the

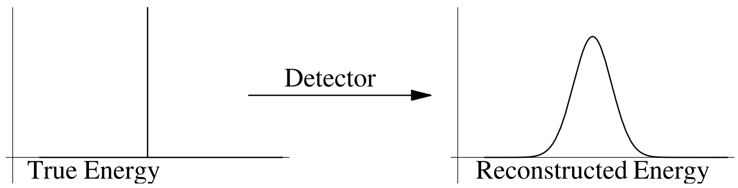


Figure 10.1: A detector maps a true parameter value onto a distribution of reconstructed parameter values. This is illustrated here for there energy.

nature of the neutrino flavor. However, in this case, only discrete values are applicable. Note that the reconstructed neutrino energy and the neutrino flavor are the only observables in GLoBES.

This picture can also be formulated in a more mathematical way. Let us define x as the true parameter value and x' as the reconstructed parameter value. Similarly, $f(x)$ is the distribution of true parameters values and $p(x')$ is the distribution of reconstructed parameter values. Then the detector function $D(x,x')$, which describes the mapping performed by the detector, is given by

$$p(x') = \int dx\, f(x) \cdot D(x,x')\,. \tag{10.1}$$

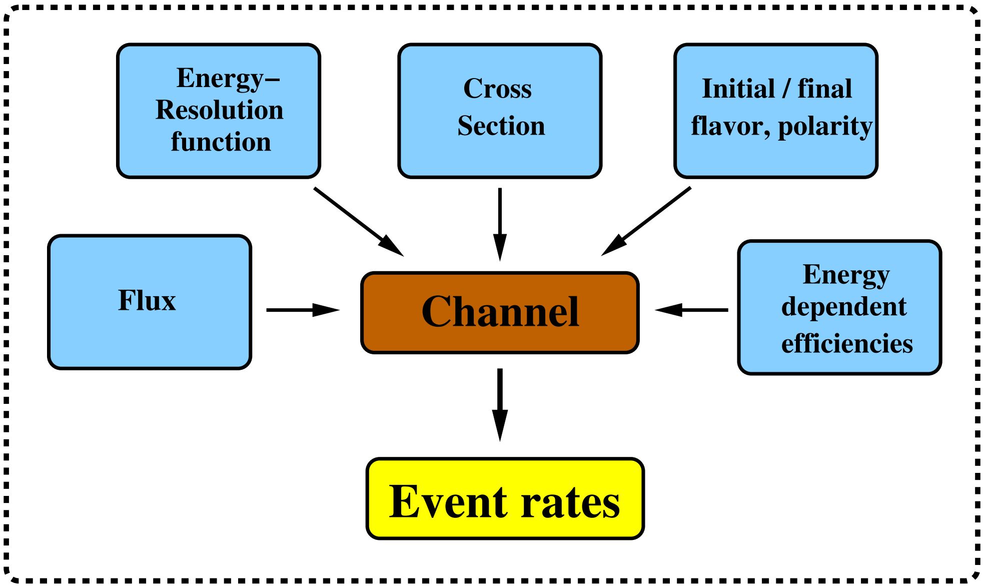


Figure 10.2: General concept of a "channel".

Obviously Eq. (10.1) only describes the detector properly if the linearity condition is fulfilled. Within this model, a detector is completely specified by a set of $D(E, E')$ for the energy variable E, and a set $D(F, F')$ for the flavor variable F. In general, $D(E, E', F)$ also depends on the incident neutrino flavor F, as well as $D(F, F', E)$ depends on the incident neutrino energy E. These sets of mapping functions usually are obtained from a full detector simulation and can be obtained by using as input distribution $f(x)$ a delta distribution $\delta(x - x_0)$.

In order to implement an experiment definition including various sources of systematical errors, we use several abstraction levels. The first level is the so-called "channel", which is the link between the oscillation physics and the detection properties for a specific oscillation pattern (*cf.*, Fig. 10.2). A channel specifies the mapping of a specific neutrino flavor produced by the source onto a reconstructed neutrino flavor. For example, a muon neutrino oscillates into an electron neutrino and subsequently interacts via quasi-elastic charged current scattering. The measured energy and direction of the secondary electron in the detector then allows to reconstruct the neutrino energy. The connection from the source flux of the muon neutrino, via the probability to appear as a electron neutrino, to its detection properties (such as cross sections and energy smearing) is encapsulated into the channel.

The channels are the building blocks for the so-called "rules". In general, a rule consists of one or more "signal" and "background" oscillation channels, which are normalized with efficiencies (*cf.*, Fig. 10.3). The event numbers from these channels are added *before* the $\Delta\chi^2$-value is calculated.[1] In addition, each rule implements *independent* systematics, such

[1]Note that in this manual, the χ^2 and $\Delta\chi^2$ are equal, since for simulated data $\Delta\chi^2 = 0$ at the best-fit

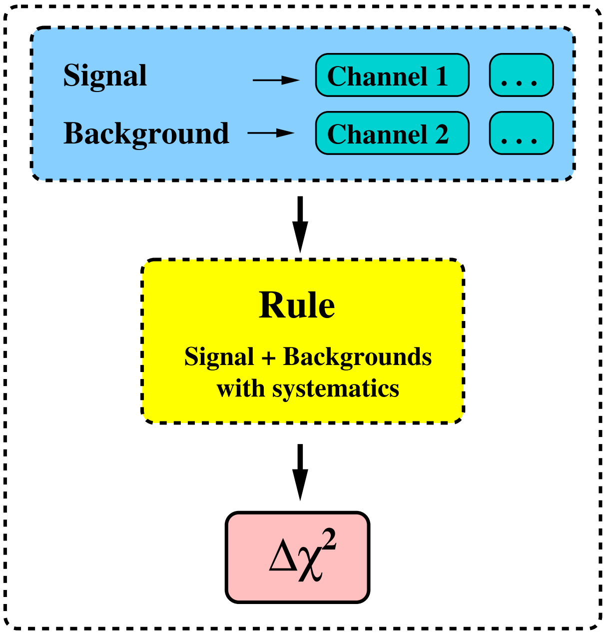


Figure 10.3: General concept of a "rule".

as signal and background normalization errors. Eventually, each rule gives a $\Delta\chi^2$-value, and the total $\Delta\chi^2$ of one experiment is obtained by adding the $\Delta\chi^2$'s of all rules (*cf.*, Fig. 10.4). Note that each experiment shares a common matter density profile. An example for a rule could look like this: We want to detect electron neutrino appearance ("signal"), where the overall efficiency for quasi-elastics electron neutrino events is 0.4. There is a fraction of 0.01 of all neutral current events which are mis-identified as quasi-elastic electron neutrino events ("background"). The neutral current fraction is only known within 10% ("background uncertainty") and there is an energy scale uncertainty of 100 MeV ("energy calibration error"). All this systematics is independent of the other rules. Thus, a rule connects the event rates to the calculation of a $\Delta\chi^2$ which properly includes systematical errors. The resulting $\Delta\chi^2$ is then the starting point for the oscillation physics analysis. Note again that

- Within each rule, the event numbers from different channels are added.
- Within each rule, the systematics is treated independently from the other rules.
- For each rule, the $\Delta\chi^2$ is computed; the $\Delta\chi^2$'s from all rules are added in an experiment.

Of course, an abstract experiment definition language can not simulate all possible types of experiments. As we have seen, there are several assumptions for source and detector. However, it turns out that GLoBES can be applied to a large number of experiment

point. Thus, we are using χ^2 and $\Delta\chi^2$ as equal quantities.

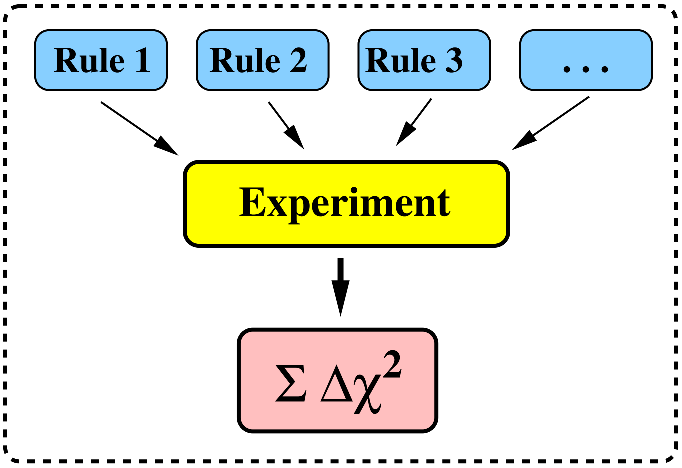


Figure 10.4: General concept of an “experiment”.

types, such as conventional beams, superbeams, neutrino factories, β-Beams, and reactor experiments.

10.2 A simple example for AEDL

Experiments in GLoBES are defined by the Abstract Experiment Definition Language (AEDL). The experiment definition is written into a text file using the AEDL syntax. Currently, a number of pre-defined experiment definition files are provided with GLoBES, which have to be modified manually in order to define new experiments. The application software then uses this text file to initialize the experiment, where other secondary files might be read for source fluxes, cross sections *etc.*. In this section, we show the definition of a very simple neutrino factory in AEDL, where we do not go into details. In the next chapter, we will discuss each of the individual steps in detail.

The first line of every experiment definition file has to be

```
!%GLoBES
```

in order not to confuse it with some other file format. In addition, GLoBES 3.0 and higher requires the identification of the minimum GLoBES version the AEDL file can be used with:

```
$version="3.0.0"
```

First, we instruct GLoBES to use the built-in source flux for a neutrino factory originating from stored μ^+'s. This is achieved by setting the `@builtin` variable to 1. Next, we specify the muon energy to be 50 GeV by the `@parent_energy` variable. We assume that there will be $5.33 \cdot 10^{20}$ useful muon decays per year and that this luminosity is available for 8 years, *i.e.*, a total number of $4.264 \cdot 10^{21}$ muons is stored:

```
/* beam */
nuflux(#mu_plus)<
  @builtin = 1
  @parent_energy = 50.0
  @stored_muons = 5.33e+20
  @time = 8.0
>
```

Note that we tell GLoBES that we want to refer to this neutrino source later as as `#mu_plus`. Let us now define a very simple detector with a target mass of 50 kt and 20 energy bins between 4 GeV and 50 GeV:

```
$target_mass = 50
$bins = 20
$emin = 4.0
$emax = 50.0
```

Then, we specify the file which contains the cross sections we want to use:

```
/* cross section */
cross(#CC)<
  @cross_file = "XCC.dat"
>
```

The command `cross` tells the parser that a cross section environment begins. It has the name `#CC`, which can later be used to refer to this specific environment, and thus to the file `XCC.dat`. Note that each name begins with a leading `#`. Of course, the baseline and matter profile have to be specified, too, where we use an arbitrary matter density profile here:

```
/* baseline */
$profiletype = 3
$densitytab = {3.5}
$lengthtab = {3000.0}
```

The curly brackets used for the definition of `$densitytab` and `$lengthtab` refer to a list of numbers. Here, the lists contain only one element each, because we only use one density layer: We specify a baseline length of 3000 km with a constant matter density of 3.5 g/cm^3. As another ingredient, we have to define the energy resolution function:

```
/* energy resolution */
energy(#MINOS)<
  @type = 1
  @sigma_e = {0.15,0.0,0.0}
>
```

The energy command starts the energy environment, which has the name #MINOS here. Out of several possibilities, it uses algorithm one, the simplest and fastest one. The actual energy resolution is specified by the energy resolution variable, which is a list of three elements. Each element is one parameter of the general resolution function as defined in Eq. 11.12. Now we have all pieces to be able to define the appearance and the corresponding disappearance channel of a neutrino factory: $\nu_e \to \nu_\mu$ and $\bar{\nu}_\mu \to \bar{\nu}_\mu$ (μ^+ stored).

```
/* channels */
channel(#appearance)<
  @channel = #mu_plus:  +:  electron:  muon:  #CC: #MINOS
>
channel(#disappearance)<
  @channel = #mu_plus:  -:  muon:  muon:  #CC: #MINOS
>
```

The first element is the name of the flux, which we have defined above. The second element "±" determines whether neutrinos or anti-neutrinos are taken from the flux table (two different polarities allowed). The third position defines the initial flavor, and the forth position the final flavor, followed by the name of the cross section and energy resolution function as defined before. The last step is to encapsulate the channels into a rule:

```
/* rules */
rule(#rule1)<
  @signal = 0.45 @ #appearance
  @signalerror = 0.025 :  0.0001
  @background = 1.0e-05 @ #disappearance
  @backgrounderror = 0.2 :  0.0001
  @sys_on_function = "chiSpectrumTilt"
  @sys_off_function = "chiNoSysSpectrum"
  @energy_window = 4.0 :  50.0
>
```

The @signal refers to the "signal" in our experiment. We use the above defined channel named #appearance with a constant overall efficiency of 0.45 (in a more realistic simulation, one would introduce an energy threshold function). The signal error variable has two components: The first one is the normalization error of the signal, here 2.5%. The second one refers to the energy calibration error of the signal, which is defined in Sec. 11.5. The background variable specifies the composition of the beam background. In this (simplified) case, we use the fraction $1 \cdot 10^{-5}$ of the channel named #disappearance, *i.e.*, the muon neutrinos with a mis-identified charge. The background error variable is defined in the same way as the signal error variable, *i.e.*, we have a 20% background uncertainty and a very small energy calibration error. In addition, the systematics treatment is specified in @sys_on_function (for systematics switched on) and @sys_off_function (for systematics switched off) – see Table 11.3.

The experiment defined here represents a first simplified version of a neutrino factory experiment. It still lacks the correct energy dependence of the efficiencies, the antineutrino disappearance rule, and the channels and rules for the symmetric operation with μ^- stored. However, it may serve as a simple, introductory example. In the next chapter, we will demonstrate that AEDL is much more powerful than illustrated here.

10.3 Introduction to the syntax of AEDL

We now give a short introduction to the syntax of AEDL. The first eight characters have to be `%!GLoBES` in order to avoid parsing megabytes of chunk and producing thousands of error messages. In addition, the minimum GLoBES version that the AEDL file is supposed to run with has to be defined by a `$version` statement, such as `$version="3.0.0"`. Comments can be used in the same way as in C:

```
/* This starts a comment
and here the comment ends */
// Another comment
```

There are pre-defined variables which all start with `$`. Their range is also checked. For example, `$bins` can be only between 0 and 500.[2] If one uses a `float` quantity where an `int` is expected, the `float` will be converted to an `int` in the same way as in C. For example, we have scalar variables

```
$bins = 10
$baseline = 1200.0
```

and simple lists

```
$densitytab={1.0,2.2343,3.3432}
```

Since there are often groups of data which we want to refer to later, environments can be used. This is illustrated with the channel definition part:

```
channel(#ch1)<
  ...
>
```

The first part is the type of environment, which is `channel` here. There are the following types of environments in AEDL:

```
nuflux
cross
channel
energy
rule
```

[2]The upper limit is only there for safety reasons, the memory is allocated dynamically.

Besides the environment type, there is a user-defined name beginning with # in the above example: #ch1. It can be used later to refer to the channel defined in `<...>`. Those names are so-called "automatic variables" and have to start with #. Note that these names have to be unique and can only be referred to after their definition. However, similar to C, one can give a declaration without definition before:

```
channel(#ch2)<>
```

Now one can refer to the name #ch2, while the actual channel definition comes later. The internal representation of this automatic variable is a number, which obtains its value from a counter for each type of environment. For example, for channel the counter is numofchannels. The counter keeps track of how many different names there are for one type of environment, which means that it counts the number of channels, rules, energy resolution functions, *etc.*. Thus, the automatic variables are numbered in the order of their definition, and the number can later be used to refer to them in the C code (from 0 to numof... -1). In order to facilitate the the mapping from names in AEDL to indices in C there are two functions glbNameToValue and glbValueToName which make this transition (see Sec. 6.2, page 52).

Within each environment type, there are several variables beginning with @, which can only be used within the appropriate type of environment. In many cases, they have a special syntax, such as @channel.

If you want to have several experiments in one file, separate the different experiments by

```
#NEXT#
```

This command resets the counters for channels, rules, fluxes, cross section and energy resolution environments. All variables have their scope limited by either %!GLoBES, #NEXT# or EOF. This allows a consistent treatment of various experiments in one file.

As another feature of AEDL, can use include files with the include command. Includes can be nested up to MAX_INCLUSION_DEPTH, which is currently set to 10. Error reporting works for nested includes, too. The included file is not required to begin with %!GLoBES to facilitate cut and paste:

```
include "./file_1"
```

With this include mechanism, one can use constructions such as

```
include "Exp1.glb"
#NEXT#
include "Exp2.glb"
```

in order to initialize a combined analysis of the experiments defined in the files Exp1.glb and Exp2.glb. Note that one has to use quotation marks for filenames in AEDL. Even if one uses the automatic variable #CC in both experiments, but the cross section data are different (for example, because of different target nuclei), the correct cross section data will be applied to each of the experiments. Note that, alternatively, one can also load both files successively by two separate calls of `glbInitExperiment`.

Furthermore, one can define constants such as

```
Pi = 3.14159
```

These constants can not only be defined within one AEDL file, but also by the calling C program, which allows to use a simple but powerful variable substitution mechanism as described in Sec. 7.3.

In addition, some simple algebraic manipulations are possible, such as

```
Pi+1
Pi^2
sin(Pi/2)
```

The following mathematical functions from `<math.h>` are available: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `log`, `log10`, `exp`, `sqrt`. These functions can be used everywhere, where otherwise only a scalar number would appear.

Finally, note that a line feed character `\n` is necessary at the end of the input – alternatively you can put a comment at the end.

10.4 More advanced AEDL features*

In GLoBES 3.0 and higher, a number of new features can be used. The most important one are lists as variables in AEDL. They start with `%`, such as

```
%effs = { 0.2, 0.4, 0.6, 0.8, 1.0, 1.0, 1.0 }
```

Functions can be threaded over lists, *i.e.*, they will be applied to each element of list, and return a list. Note that the original list will be destroyed by this process. Therefore, it is necessary to create a copy of your list if you want to use the original and the threading result. For that purpose, the `copy` function is provided:

```
listb = copy(lista)
listb := lista // Alternative method in environments
```

You will also need to use `copy` when you assign a list to an experiment structure (see below).

Two helper functions `bincenter()` and `samplingbincenter()` return lists with the central energies of the bins or sampling points, respectively. For example,

```
%bc=bincenter()
```

A very useful new features is an interpolation function which can directly interpolate a number of points and evaluate them at a different set of places. For example,

```
%energ = { 4.0,20.0,50.0 }
%effs = { 0.0,1.0,1.0 }
%ires = interpolation(%energ,%effs,1,%bc)
```

interpolates the points with x-values `%energ` and y-values `%effs` with the interpolation order one (linear interpolation, third parameter) and evaluates the interpolation result at the bin centers obtained above, *i.e.*, it returns a list of the y-values at the places specified by the last parameter. The only allowed interpolation orders are "1" (linear) and "2" (cubic splines). This example creates a neutrino factory energy threshold function linearly climbing from 0 to 1 between 4 GeV and 20 GeV. It can be directly used in a channel definition, *e.g.*,

```
channel(#nu_mu_appearance)<
   @channel = #mu_plus:  +:  electron:  muon:  #CC: #MINOS
   @post_smearing_efficiencies = copy(%ires)
>
```

To simplify debugging of lists and numbers, GLoBES now supports output directly from AEDL files:

```
R = 1.15
echo(R) // Print without line feed
line(2) // Two line feeds
echon(ires) // Print with line feed
```

Chapter 11

Experiment definition with AEDL

In this chapter, we give a detailed description of the AEDL features. We also show the underlying mathematical concepts where applicable. We do not exactly follow the separation of source, oscillation, and detection properties, since most issues more or less involve the detection. Instead, we illustrate many of the features of the GLoBES simulation successively in the logical order of their definition, and demonstrate how they translate into AEDL.

11.1 Source properties and integrated luminosity

As we have discussed before, GLoBES can only deal with point sources. Thus, it is not possible to study effects of the finite size of the neutrino production region, such as in the sun or in reactor experiments with many neutrino sources (*e.g.*, KamLAND). Therefore, a neutrino source in GLoBES can, in general, be characterized by the flux spectrum for each neutrino flavor, the CP sign (neutrinos or antineutrinos), and the total luminosity of the source.

Before we come to the definition of the source properties, let us discuss the total integrated luminosity of the experiment. In GLoBES, the total number of events is in general proportional to the product of

$$N \times \text{Fid. det. mass [kt/t]} \times \text{Running time [yr]} \times \left\{ \begin{array}{c} \text{Source power [MW/GW]} \\ \text{Useful parent decays [yr}^{-1}] \end{array} \right. \tag{11.1}$$

with N being a normalization constant. Thus, the source power corresponds to either the amount of energy produced per time frame in the target (such as for nuclear reactors or sources based on pion decay), or the useful parent particle decays per time frame (neutrino factories, beta beams). In addition, the definition of the source power makes only sense together with the flux normalization, the running time, and the fiducial detector mass in order to define the total integrated luminosity. Therefore, one can, in principle, use arbitrary units for these components as long as their product gives the wanted neutrino flux. However, it is recommended to use normalizations such that the source power units are

@builtin	Description	Parameters	Min. version
1	Neutrino factory μ^+ decay	`@parent_energy` [GeV], `@stored_muons`	2.0
2	Neutrino factory μ^- decay	`@parent_energy` [GeV], `@stored_muons`	2.0
3	Beta beam, inv. beta decay	`@end_point` [GeV], `@stored_ions`, `@gamma`	3.0
4	Beta beam, beta decay	`@end_point` [GeV], `@stored_ions`, `@gamma`	3.0

Table 11.1: Built-in fluxes currently supported by GLoBES. For details on the beta beams, see Ref. [15].

MW for a proton-based beam, and $\mathrm{GW}_{\mathrm{thermal}}$ for a reactor experiment. Correspondingly, the detector mass units should be kilotons for a proton-based beam, and tons for a reactor experiment. In any case it is a good idea to document the choices made by the user by corresponding comments in AEDL. For more details on the luminosity implementation, see Appendix C.

The quantity which can be used to scale the overall integrated luminosity of an experiment, is the fiducial detector mass. For example,

```
$target_mass = 50.0
```

defines a 50 kt detector for a neutrino factory.

There are two principal ways to initialize a neutrino flux: Either one can use a built-in source, or one can provide a file. In both cases, a flux is defined by the environment `nuflux`, such as

```
nuflux(#name)<
  ...
  @time = 8.0
>
```

with a running time of 8 years. Note that the running time is used within the `nuflux` environment. This feature can be used to load the neutrino and antineutrino fluxes in an accelerator experiment separately, in order to combine them with different running times in the respective operation modes. The name of the flux `#name` will later be referred to in the channel definitions. Note that GLoBES versions older than 3.0 use the still supported `flux` environment, which is different from `nuflux` by an undocumented normalization factor 5.2 for user-defined fluxes. This difference is explained in Appendix C.

For a built-in neutrino source, one has to specify which built-in spectrum should be used, as well as its parameters. The software will then automatically calculate the neutrino spectrum. Note that in this case, there is no degree of freedom in the choice of the source units. The currently available fluxes are described in Table 11.1. For example, two built-in neutrino factory fluxes are available: μ^+-decay (`@builtin = 1`) and μ^--decay (`@builtin = 2`). In these cases, the muon energy (energy of the parent particle) has to be specified together with the number of useful decays per year. Thus, an example to set up a neutrino factory flux is

```
nuflux(#mu_plus)<
  @builtin = 1
  @parent_energy = 50.0
  @stored_muons = 5.33e+20
  @time = 8.0
>
```

Furthermore, two beta beam neutrino fluxes are available: inverse beta decay (`@builtin = 3`, *i.e.* from the decay of ^{18}Ne-isotopes) and beta decay (`@builtin = 4`, *i.e.* from the decay of ^{6}He-isotopes). For a beta beam file, using the beta beam built-in fluxes, the end point energy of the decay (`@end_point`), the number of decays per year (`@stored_muons`), and the acceleration factor of the ions γ (`@gamma`) have to be specified. An example for setting up a beta beam neutrino flux from the decay of ^{18}Ne isotopes is

```
nuflux(#nu_e_flux)<
  @builtin = 3
  @gamma = 130.0
  @end_point = 0.0034
  @stored_muons = 2.2e+18
  @time = 4.0
>
```

For a user-defined flux, one has to specify the file name:

```
nuflux(#user)<
  @flux_file = "user_file_1.dat"
  @time = 2.0
  @power = 4.0
  @norm = 1e+8
>
```

In this case, the `@norm` variable is an overall normalization which defines a conversion factor from the fluxes in the file to the units in GLoBES. In general, there are many ways to give the source power of a neutrino source, such as neutrinos per proton on target per area per time frame. Right now, each flux has its own normalization factor, which is not always straightforward to calculate. Often, one has to take into account many things, such as the number of target particles per unit mass. In addition, the fluxes will be rescaled by $1/L^2$, which means that the normalization must contain a factor L_0^2. Here L_0 is the distance

`$profiletype`	Additional variables	Description
1	`$baseline`	Average density (constant)
2	`$baseline`, `$densitysteps`	PREM profile with given number of equidistant steps
3	`$lengthtab`, `$densitytab`	Arbitrary profile (table of layer thicknesses, table of densities)

Table 11.2: Different matter density profiles which can be used with GLoBES.

from the source for which the flux is given to the actual neutrino production region. At the end, it is left to the user to ensure that the numbers in the flux file give, after the multiplication with `@norm`, the proper numbers of produced neutrinos corresponding to the chosen target power `@power`. Usually this adjustment of `@norm` is performed by comparison with known energy spectra for a specific experiment. For more details on the flux definition, see Appendix C.

The software assumes that the given flux file has seven columns and 501 lines with equidistant energies. The format is:

$$E \quad \Phi_{\nu_e} \quad \Phi_{\nu_\mu} \quad \Phi_{\nu_\tau} \quad \Phi_{\bar{\nu}_e} \quad \Phi_{\bar{\nu}_\mu} \quad \Phi_{\bar{\nu}_\tau}$$

In order to access fluxes at arbitrary energies, linear interpolation is used by GLoBES. In general, it is advisable to provide the flux between `$sampling_min` and `$sampling_max` (*cf.*, Sec. 11.5), since this is the energy range considered in the simulation. However, if part of this interval is omitted in the flux file, zero will be used there. If some neutrino flavors are not used in the simulation, the corresponding columns in the flux file have to be filled nevertheless, *e.g.* with zeros.

The flux files accept one-line comments, which start with `#` and end with the linefeed character '`\n`', they are not counted as a line and their content is discarded. These comments are useful to provide meta information about the fluxes such as units or the origin of the information. This is also the default method to point the user to the references that should be cited when using a particular flux file.

11.2 Baseline and matter density profile

Besides the energy and the involved flavors, the neutrino oscillation physics depends on the baseline and the matter density profile. All neutrino oscillation parameters are defined at running time.

The baseline is given by

```
$baseline = 3000.0
```

Note that baseline lengths are always assumed to be in kilometers.

Furthermore, the matter density profile along the baseline has to be specified. The simplest profile is a constant matter density equal to the average matter density from the PREM [3, 4] onion shell model of the earth:

```
$profiletype=1
```

If you are using this option please cite Refs. [3, 4].

For a better approximation of the realistic earth matter density profile, one can use an arbitrary number of equidistant steps of the PREM profile:

```
$profiletype=2
$densitysteps=20
```

Note that the value of `$densitysteps` is time-critical, since the computation time of oscillation probabilities is directly proportional to the number of layers. As a third possibility, one can specify the matter density profile manually with a list of thicknesses and densities of the matter density layers. This example uses two density steps with two different densities:

```
$profiletype=3
$densitytab={2.8, 3.5}
$lengthtab={1000.0, 2000.0}
```

It is important that both lists have the same length and that the thicknesses given in `$lengthtab` add up to the length of the baseline, which does not have to be explicitely specified anymore. In addition, matter densities are always given in g/cm^3. This approach can also be used for a constant matter density profile with a specific matter density:

```
$profiletype=3
$densitytab={3.5}
$lengthtab={3000.0}
```

The possible options for matter density profiles are summarized in Table 11.2.

11.3 Cross sections

Cross sections will later be used as part of the channel definition (see Sec. 11.4). Similar to the source fluxes, they are provided by the user as a data file:

```
cross(#name)<
  @cross_file ="user_file_1.dat"
>
```

This cross section can later be referred to by `#name`.

Cross sections in GLoBES are given as total cross section divided by energy:

$$\hat{\sigma}(E) = \sigma(E)/E \left[10^{-38}\,\frac{\mathrm{cm}^2}{\mathrm{GeV}}\right] \tag{11.2}$$

The software assumes that the cross section files are text files with seven columns and 1001 lines of the form

$$\log_{10}E \quad \hat{\sigma}_{\nu_e} \quad \hat{\sigma}_{\nu_\mu} \quad \hat{\sigma}_{\nu_\tau} \quad \hat{\sigma}_{\bar{\nu}_e} \quad \hat{\sigma}_{\bar{\nu}_\mu} \quad \hat{\sigma}_{\bar{\nu}_\tau}$$

Here the logarithms of the energy values have to be equidistant. For arbitrary energies, linear interpolation is used. If the energy leaves the range of values given in the file, 0.0 will be assumed. In general, it is advisable to provide the cross sections in the range between `$sampling_min` and `$sampling_max` (*cf.*, Sec. 11.5). Cross sections for unused neutrino flavors have to be filled with zeros, and can not just be omitted.

Like the flux files, the cross section files accept one-line comments, which start with `#` and end with the linefeed character '`\n`'; they are not counted as a line and their content is discarded. These comments are useful to provide meta information about the cross sections like units or the origin of the information. This is also the default method to point the user to the references he/she should cite when using a particular cross section file.

11.4 Oscillation channels

Channels in GLoBES represent an intermediate level between the pure oscillation physics given by the oscillation probability $P_{\alpha\beta}$, and the final event rates composed of signal and background. A channel describes the path from one initial neutrino flavor in the source to the event rates in the detector for one specific interaction type (IT) and final flavor. Therefore, a channel contains the description of the initial neutrino flavor, its CP eigenvalue (neutrino or antineutrino), the detected neutrino flavor, the interaction cross sections for the chosen interaction type, and the energy resolution function of the detector.

Before we come to the definition of channels in AEDL, we introduce the general concept for the calculation of event rates. The first step is to compute the number of events for each IT in the detector for each initial and final neutrino flavor and energy bin. The second step is to include the detector effects coming from the insufficient knowledge in the event reconstruction. The combination of these two steps leads to the differential event rate spectrum for each initial and final flavor and IT as seen by the detector, which we call the "channel". In this section, we focus on the first step, *i.e.*, we discuss the definition of the energy resolution function in the next section, since this is a rather comprehensive issue.

The differential event rate for each channel is given by

$$\begin{aligned}
\frac{dn_\beta^{\mathrm{IT}}}{dE'} = \quad & N \int_0^\infty \int_0^\infty dE\, d\hat{E} \quad \underbrace{\Phi_\alpha(E)}_{\text{Production}} \times \\
& \underbrace{\frac{1}{L^2} P_{(\alpha\to\beta)}(E, L, \rho; \theta_{12}, \theta_{13}, \theta_{23}, \Delta m_{31}^2, \Delta m_{21}^2, \delta_{\mathrm{CP}})}_{\text{Propagation}} \times \\
& \underbrace{\sigma_f^{\mathrm{IT}}(E) k_f^{\mathrm{IT}}(E - \hat{E})}_{\text{Interaction}} \times \\
& \underbrace{T_f(\hat{E}) V_f(\hat{E} - E')}_{\text{Detection}},
\end{aligned} \tag{11.3}$$

where α is the initial flavor of the neutrino, β is the final flavor, $\Phi_\alpha(E)$ is the flux of the initial flavor at the source, L is the baseline length, N is a normalization factor, and ρ is the matter density. The energies in this formula are given as follows:

- E is the incident neutrino energy, *i.e.*, the actual energy of the incoming neutrino (which is not directly accessible to the experiment)
- $\hat{E}$ is the energy of the secondary particle
- E' is the reconstructed neutrino energy, *i.e.*, the measured neutrino energy as obtained from the experiment

The interaction term is composed of two factors, which are the total cross section $\sigma_\beta^{\mathrm{IT}}(E)$ for the flavor f and the interaction type IT, and the energy distribution of the secondary particle $k_\beta^{\mathrm{IT}}(E-\hat{E})$. The detector properties are modeled by the threshold function $T_\beta(\hat{E})$, coming from the the limited resolution or the cuts in the analysis, and the energy resolution function $V_\beta(\hat{E}-E')$ of the secondary particle.

Since it is computationally very expensive to solve this double integral numerically, we split up the two integrations. The integral over $\hat{E}$ depends only on the terms containing $\hat{E}$, i.e., on $k_\beta^{\mathrm{IT}}(E-\hat{E})$, $T_\beta(\hat{E})$, and $V_\beta(\hat{E}-E')$. These terms do not depend on the oscillation parameters, so they will not vary during the fit, and the $\hat{E}$ integral can be pre-computed in the initialization phase. We define:

$$R_\beta^{\mathrm{IT}}(E,E')\,\epsilon_\beta^{\mathrm{IT}}(E') \equiv \int\limits_0^\infty d\hat{E} \quad T_\beta(\hat{E})\,k_\beta^{\mathrm{IT}}(E-\hat{E})\,V_\beta(\hat{E}-E')\,. \tag{11.4}$$

Thus, $R_\beta^{\mathrm{IT}}(E,E')$ describes the energy response of the detector, *i.e.*, a neutrino with a (true) energy E is reconstructed with an energy between E' and $E'+dE'$ with a probability $R_\beta^{\mathrm{IT}}(E,E')dE'$. The function $R(E,E')$ is often called "energy resolution function". Actually, its internal representation in the software is a smearing matrix. The function $\epsilon_\beta^{\mathrm{IT}}(E')$ will later be referred to as "post-smearing efficiencies", since it will allow us to define cuts and threshold functions *after* the smearing is performed, *i.e.*, as function of E'. The detailed definition and initialization of the energy resolution function is described in Sec. 11.5.

Eventually, we can write down the number of events per bin i and channel c as

$$n_i^c = \int_{E_i-\Delta E_i/2}^{E_i+\Delta E_i/2} dE' \quad \frac{dn_\beta^{\mathrm{IT}}}{dE'}(E') \tag{11.5}$$

where ΔE_i is the bin size of the ith energy bin. This means that one has to solve the integral

$$n_i^c = N/L^2 \int_{E_i-\Delta E_i/2}^{E_i+\Delta E_i/2} dE' \quad \int\limits_0^\infty dE\ \Phi^c(E)\,P^c(E)\,\sigma^c(E)\,R^c(E,E')\,\epsilon^c(E')\,. \tag{11.6}$$

Note that the events are binned according to their *reconstructed* energy.

A simple channel definition in GLoBES consists of the flux, the CP-sign of the initial state, the initial flavor, the final flavor, the cross sections, and the energy resolution function. In order to refer to the fluxes, cross sections, and energy resolution functions, they have to be defined first with their `#name` in the respective environments. A simple definition of a channel is

```
channel(#channel_1)<
  @channel = #flux :  +:  muon:  muon:  #cross:  #energy
>
```

It is also possible to define a channel as no-oscillation by using the prefix `NOSC_` in either the initial flavor or the final flavor, like this

```
channel(#channel_1)<
  @channel = #flux :  +:  NOSC_muon:  muon:  #cross:  #energy
>
```

In this case all diagonal probabilities $P_{\alpha\alpha}$ are unity, and all off-diagonal probabilities $P_{\alpha\beta}$ are zero. This is, for instance, useful for neutral current events, since these do not depend on any oscillation parameters[1]. The channels marked as `NOSC_` are already computed by `glbSetRates` and do not have to be recomputed in the subsequent fit (which calls the undocumented function `glbSetNewRates`). Therefore this feature can be used to speed up the rate computation considerably, especially in cases where a large set of channels exist which are only used for the computation of backgrounds. Usually, it is an excellent approximation to treat backgrounds as if they were not affected by oscillations[2].

Note that the energy environment will be described in the next section. In addition, one can define pre- and post-smearing effects together with the channels, which will also be introduced together with the energy resolution function in the next section.

11.5 Energy resolution function

The definition and implementation of the energy resolution function is rather sophisticated in GLoBES. In particular, the choice of the proper parameters depends on the experiment and the frequencies of the involved neutrino oscillations. This choice also greatly influences the speed of the calculation.

In this section, we first discuss the principles of the energy smearing, where it is assumed that the reader is familiar with Sec. 11.4. Then we introduce an automatic energy smearing algorithm, which is fairly simple to understand and applicable to most beam-based experiments. In most cases, the reader may want to proceed to the next section

[1]At least in the absence of sterile neutrinos

[2]In the case that the backgrounds have a sizeable dependence on the oscillation parameters, they carry information on the oscillation parameters, and therefore behave more like a signal.

after reading these two subsections. In the third subsection, we describe a more elaborate (and slower) smearing algorithm, which is useful to avoid aliasing effects if the neutrino oscillations are rather fast compared to the bin size, as is the case for (solar) reactor experiments. Eventually, we show how one can use a manual smearing matrix instead of using one of the implemented algorithms.

11.5.1 Introduction and principles

The energy resolution function $R^c(E, E')$ and the post-smearing efficiencies $\epsilon(E')$ have already been introduced in Sec. 11.4, where a definition has been given in Eq. (11.4). Instead of using Eq. (11.4) directly, we apply a slightly different definition of the post-smearing efficiencies $\epsilon(E')$. In general, they have to be determined by means of a Monte Carlo simulation of the experiment. This usually involves a binning of the simulated events in the reconstructed energy E'. Therefore, one can simplify Eq. (11.6) by

$$\int_{E_i-\Delta E_i/2}^{E_i+\Delta E_i/2} dE'\, R^c(E,E')\,\epsilon^c(E') \simeq \hat{\epsilon}_i^c \cdot \int_{E_i-\Delta E_i/2}^{E_i+\Delta E_i/2} dE' \quad R^c(E,E')\,. \tag{11.7}$$

Here the $\hat{\epsilon}_i^c$ are the binned "post-smearing" efficiencies, which will be set within the corresponding `channel` environment (see below). From Eq. (11.6) it is obvious that the integration with respect to the reconstructed energy E' can be performed independently of the oscillation parameters. We define the "bin kernel" K_i^c for the ith bin as

$$K_i^c(E) \equiv \int_{E_i-\Delta E_i/2}^{E_i+\Delta E_i/2} dE' \quad R^c(E,E')\,. \tag{11.8}$$

With this definition, Eq. (11.6) can be re-written as

$$n_i^c = N/L^2\,\hat{\epsilon}_i^c \int_0^\infty dE \quad \underbrace{\Phi^c(E)\,P^c(E)\,\sigma^c(E)\,K_i^c(E)}_{f(E)}\,. \tag{11.9}$$

There is no principle reason why one should not evaluate this integral directly by the usual numerical methods. However, it turns out that this is very slow in many cases. Therefore, we will introduce two different approximation schemes for different applications in the next two subsections. In either case, the integrand in Eq. (11.9) has to be evaluated at fixed "sampling points". These sampling points have to directly or indirectly be defined by the user.

Before we come to the calculation algorithms, it is useful to understand the general evaluation algorithm. As it is illustrated in Fig. 11.1, GLoBES uses several levels with respect to the energy ranges:

Sampling point level This level is used internally to evaluate the integrand in Eq. (11.9) at all sampling points. The energy scale is the actual incident neutrino energy E. For a manual definition of the sampling points, use

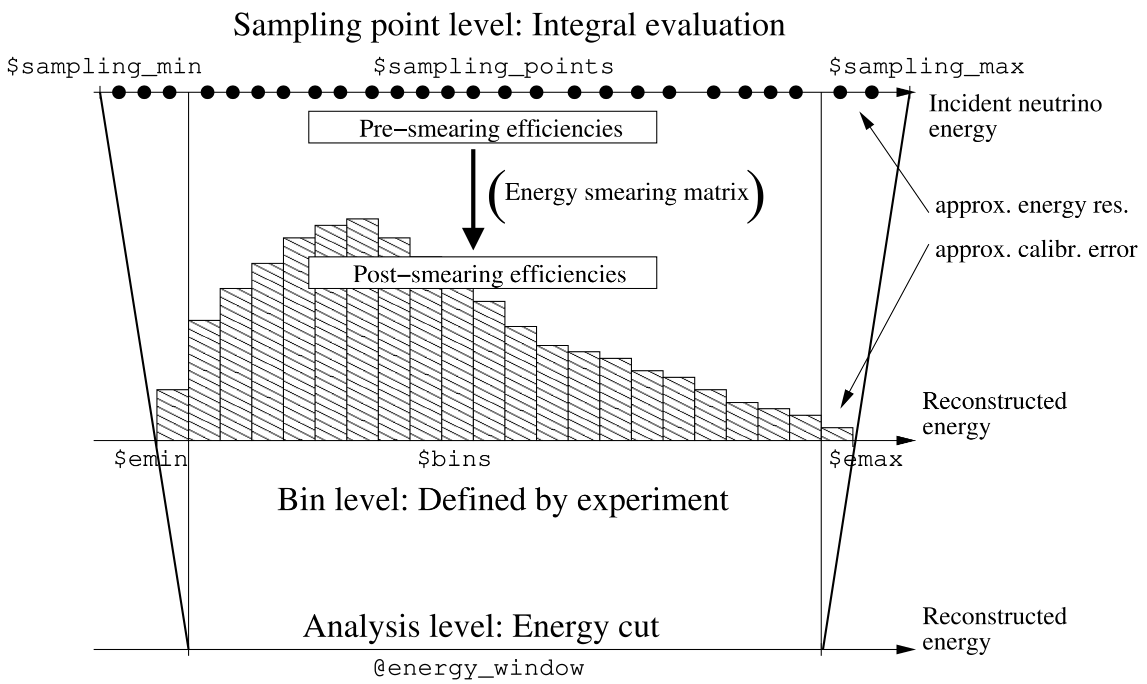


Figure 11.1: The different evaluation levels for the energy smearing in GLoBES.

```
$sampling_points = 20
$sampling_min = 4.0
$sampling_max = 50.0
```

for equidistant sampling points. If no values are given for these variables, they are assumed to be equal to their corresponding counterparts at the bin level, *i.e.*, `$sampling_points = $bins`, `$sampling_min = $emin` and `$sampling_max = $emax`.

Arbitrarily spaced sampling points can be specified with `$sampling_stepsize`

```
$sampling_stepsize={1.0,2.0,3.0,4.0,5.0,...}
```

The choice of the sampling point configuration strongly depends on the experiment and required accuracy. Ideally, the integrand of Eq. 11.9 is zero outside the sampling range. If this cannot be achieved, it is usually sufficient that the sampling range is by about three times the energy resolution (evaluated at `$emin` and `$emax`, respectively) larger than the bin range. The spacing of the sampling points should be somewhat smaller than the finest details of the integrand (a factor $\simeq 2$ usually is more than enough).

Bin level This level is determined by the experiment and its analysis. Note that energy bin sizes much smaller than the energy resolution will not improve the results. The energy bin range and the number of energy bins always have to be specified. For the case of large values of the integrand in Eq. (11.9) at the energy range limits, it is recommended to exceed the analysis energy window by about three times the energy calibration error in order to avoid cutoff effects. In addition, note that an energy resolution larger than about the bin range will distribute events out of this range. Therefore, the normalization will be affected.

In order to define a range between E_{min} and E_{max} divided by a certain number of equidistant bins, use

```
$emin = 4.0
$emax = 50.0
$bins = 20
```

For arbitrary bins, use E_{min} and E_{max} and the size of each bin ΔE_i:

```
$emin = 4.0
$emax = 50.0
$binsize = { 15.0 , 5.0 , 20.0, 6.0 }
```

The number of bins will be automatically computed by GLoBES. Note that the bin sizes have to add up to the energy range `$emax-$emin`.

The choices at bin level are mainly determined by optimizing the performance of the experiment.

Analysis level On the analysis level, an energy window can be defined within each rule. For details, see next chapter.

In general, the energy smearing happens between the sampling point and bin levels, which means that the energy smearing matrix will have `$sampling_points` columns and `$bins` rows.

As illustrated in Fig. 11.1, an interesting feature in combination with the channels are pre- and post-smearing effects. Pre-smearing effects are taken into account on the sampling point level, and post-smearing effects on the bin level. Examples for these effects are energy dependent efficiencies and (non-beam) backgrounds. Efficiencies are multiplicative factors, whereas backgrounds are added to the event rates. These components can be introduced before or after the integration in Eq. (11.9) is done. If they are introduced before, we call them `@pre_smearing_efficiencies` or `@pre_smearing_background`. If they are introduced afterwards, we call them `@post_smearing_efficiencies` or `@post_smearing_background`. Note that pre-smearing components are always a function of the incident neutrino energy E. Thus, there have to be as many elements as there are sampling points. Examples for pre-smearing quantities are non-beam backgrounds, such as from geophysical neutrinos. The post-smearing components are always a function of the reconstructed neutrino energy E', such as the post-smearing efficiencies $\epsilon_{\beta}^{\mathrm{IT}}(E')$ in Eq. (11.4). Examples for post-smearing efficiencies are cuts and detection threshold functions. All post-smearing components have to have as many elements as there are energy bins. Efficiencies are multiplicative and their default value is 1, whereas backgrounds are additive and their default value is 0. Thus, a more elaborate channel can be defined as

```
channel(#channel_1)<
  @channel = #flux :  +:  muon:  muon:  #cross:  #energy
  @pre_smearing_background = {1,2,3,4,5,6,7,8,9,10}
  @post_smearing_efficiencies = {0.1,0.2,0.3,0.4,0.5}
>
```

This experiment uses 10 sampling points and 5 bins.

In the following subsections we will explain the energy resolution function. All energy resolution functions are defined within an `energy` environment and can be referred to by `#name`:

```
energy(#name)<
  ...
>
```

The individual parameters of the environment will be defined below and depend on the algorithm used.

11.5.2 Bin-based automatic energy smearing

This algorithm is the simplest of the built-in algorithms for the evaluation of Eq. (11.9). It is applicable to most of the experiments which can be simulated with GLoBES.

The key idea is to use a "flat" model, *i.e.* the integrand of Eq. 11.9 is well approximated by being piecewise constant in each sampling step. This is a good approximation as long as

- No details are lost, *i.e.* the spacing of sampling points is smaller than the energy resolution.
- The edges are treated correctly.
- The neutrino oscillations are slow on a scale of the sampling point distance.

In this case, Eq. (11.9) is reduced to

$$n_i^c = N/L^2 \sum_{j=1}^{N} \Phi^c(E_j)\, P^c(E_j)\, \sigma^c(E_j)\, K_i^c(E_j)\, \Delta E_j\,. \tag{11.10}$$

The advantages of this algorithm are obvious: All factors independent of the oscillation parameters have to be only evaluated once at values of E which are known in advance, which means that they can be put into a look-up table. In addition, the probability has to be only evaluated at previously known values of the energy, which makes it possible to compute the transition amplitudes for all channels simultaneously. One assumption is that all involved factors are piece-wise constant, *i.e.*, they hardly change within each bin. This assumption seems to be very restrictive, which is however not quite correct. First of all, if one analyzes simulated data (which are simulated with the same algorithm), the errors will cancel between the simulated and fitted data. Second, and more important, this algorithm is just a very basic integration routine[3] and converges to the true result for decreasing step size. Thus if the number of sampling points is large enough this algorithm is very accurate. Bin-based energy smearing is selected by

```
@type = 1
```

within the `#energy` environment. The computation of the bin kernel K_i^c is performed by GLoBES. Thus, it requires that the number of bins `$bins` and the minimum energy `$emin` and maximum energy `$emax` are given in case of equidistant bins. As far as the parameterization for the energy resolution function $R^c(E, E')$ in Eq. (11.8) is concerned, the algorithm uses a Gaußian

$$R^c(E, E') = \frac{1}{\sigma(E)\sqrt{2\pi}} \, e^{-\frac{(E-E')^2}{2\sigma^2(E)}} \,. \tag{11.11}$$

There are several energy resolution functions available, where by default `#standard` is used:

```
@sigma_function = #standard
```

The energy resolution function `#standard` is defined by

$$\sigma(E) = \alpha \cdot E + \beta \cdot \sqrt{E} + \gamma \,, \tag{11.12}$$

where the parameters α, β and γ are provided by the user:

```
@sigma_e = {0.15, 0.0, 0.0}
```

Currently, another possible choice for `@sigma_function` is `#inverse_beta`, which only uses the parameter α. It is defined by

$$\sigma(E) = \begin{cases} \alpha \cdot \sqrt{1000}^{\,-1} \sqrt{x - 8 \cdot 10^{-4}} \,, & \text{for } x > 1.8 \cdot 10^{-3} \\ \alpha \cdot 10^{-3} \,, & \text{for } x \leq 1.8 \cdot 10^{-3} \end{cases} \tag{11.13}$$

The somewhat complicated form is due to the fact that inverse β-decay has a neutrino threshold of 1.8 MeV and that a neutrino at threshold already produces $\simeq 1$ MeV visible energy in the detector (for more details see *e.g.* [12]).

[3] It is planned for the future to implement something like a Gauß-Kronrod scheme as an alternative here.

In the actual implementation of the algorithm, the sum in Eq. (11.10) is only evaluated for the E_j's with $K(E_j)$ above a certain threshold, which is, by default, 10^{-7}. This threshold is defined at compilation time.

Eventually, a complete energy resolution definition with bin-based automatic energy smearing is, for example,

```
energy(#name)<
  @type = 1
  @sigma_function = #standard
  @sigma_e = {0.15 ,0.0 ,0.0}
>
```

11.5.3 Low-pass filter*

In order to ensure that very fast oscillations do not lead to aliasing, it is possible to impose a low-pass filter already during the calculation of the probabilities itself. This highly experimental feature will be called "filter" in the following. The calculation of oscillation probabilities is, in principle, a computation of phase differences. Restricting the maximum admissible size of those phase differences effectively filters the high frequency component of the oscillation probability. This idea is implemented according to

$$P_{\alpha\beta}(E) \quad = \quad \sum_{ij} U_{\alpha j} U^*_{\beta j} U^*_{\alpha i} U_{\beta i} e^{-i\Phi_{ij}} \times e^{-\Phi^2_{ij}/\sigma_f(E)^2}\,, \tag{11.14}$$

where $\Phi_{ij} := \Delta m^2_{ij} L/2E$ is the usual phase difference and the last term is a Gaußian filter with width $\sigma_f(E)$. Choosing $\sigma_f(E) := \sigma^0_f \cdot E$ ensures that this filter behaves approximately as an energy resolution function with constant width $\sigma_e = \sqrt{2}/\sigma^0_f$, *i.e.*

$$\int d\tilde{E} \quad P(\tilde{E}) \frac{1}{\sigma_e \sqrt{2\pi}}\, e^{-\frac{(E-\tilde{E})^2}{2\sigma_e^2}}\,. \tag{11.15}$$

The relationship between Eqs. (11.14) and (11.15) is not obvious and connected to the properties of $P_{\alpha\beta}$: see Refs. [23, 24]. This feature works *only* for vacuum and constant densities and is controlled by the filer state variable. In addition, σ_e is set by the filter value variable:

```
$filter_state = 1
$filter_value = 2.0
```

would switch the filter feature on and set the width to 2.0 GeV. The setting of `$filter_state` is ignored whenever a density profile with more than one layer is used.

With a type 1 (`@type = 1`) energy resolution function, σ_e contributes to the energy resolution function of the detector $\sigma_c(E)$ according to

$$\sigma_{\text{eff}}(E)^2 \simeq \sigma_e^2 + \sigma_c(E)^2\,. \tag{11.16}$$

Sometimes this behavior is unwanted, and therefore one can try to 'subtract' the filtering from the energy resolution function by splitting the energy resolution function $\sigma(E)_{\mathrm{eff}}$ into two parts by

$$\sigma_{\mathrm{eff}}(E)^2 = \underbrace{\sigma_c(E)^2 - \sigma_e^2}_{\tilde{\sigma}_c^2(E)} + \sigma_e^2\,, \tag{11.17}$$

where the truncated energy resolution function $\tilde{\sigma}_c(E)$ is used instead of $\sigma_c(E)$ in computing the smearing data. Thus one obtains as effective energy resolution

$$\sigma_{\mathrm{eff}}(E)^2 \simeq \sigma_c(E)^2\,. \tag{11.18}$$

This scheme is used by choosing as type for the energy resolution

```
@type = 2
```

11.5.4 Manual energy smearing*

In some cases, one may want to use the output of a detector Monte Carlo simulation directly. Then one can use "manual" energy smearing instead of the automatic energy smearing algorithms.

The energy smearing matrix K_{ij} has `$bins` rows and `$sampling_points` columns, which are numbered from 0 to `$bins`−1 resp. `$sampling_points`−1. It is equivalent to the bin- and sampling-point-based kernel in Eq. (11.8):

$$K_{ij} = K_i^c(E)|_{E=E_j}\,, \tag{11.19}$$

where E_j is the energy of the jth sampling point. In general, many of the entries in this matrix are zero, which means that it is convenient to evaluate the integrand in Eq. (11.9) only at positions where K_{ij} is non-zero. The corresponding "sampling range" of non-zero matrix entries in K_{ij} for the ith energy bin is defined to run from column k_l^i ("lower index") to column k_u^i ("upper index"). An example for a smearing matrix is

$$K_{ij} = \underbrace{\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & & & & \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & & & \\ & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & & \\ & & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & \\ & & & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ & & & \uparrow & & \ddots & & \uparrow \\ & & & k_l^i & & & & k_u^i \end{pmatrix}}_{\texttt{\$sampling\_points}\text{ columns}} \quad \longleftarrow \quad \texttt{\$bins}\text{ rows}\,, \tag{11.20}$$

where the un-shown entries are zero. Thus, the values of K_{ij} have to be specified between k_l^i and k_u^i in the form $\{k_l^i, k_u^i, K_{i\,k_l^i}, K_{ik_l^i+1}, \ldots, K_{ik_u^i}\}$:

```
energy(#name)<
  @energy = {0,2, 0.8634265, 0.0682827, 4e-06}:
    {0,4, 0.1507103, 0.6965592, 0.1507103, 0.00101, 1e-07}:
    ...
    {40,42, 0.1507103, 0.6965592, 0.1507103};
>
```

The last line has to be terminated by a semicolon ';'. Note that the sum of all entries in each *column* should be equal to unity, since all of the incoming neutrinos should be assigned to energy bins. In many practical cases, however, the definition of the energy smearing can lead to sums smaller than unity, such as in the case of truncated Gaußian distributions. The sum of entries in each *row* is not defined, since the events might be unevenly distributed into the energy bins according to the energy resolution function.

11.6 Rules and the treatment of systematics

The set of rules for an experiment is the final link between the event rate computation and the statistical analysis. The information in the rules specifies how the χ^2 is computed based upon the raw event rates given by the channels and possible systematical errors. Therefore a rule has two parts: The first part describes how signal and background events are composed out of the channels, and the second part specifies which systematical errors are considered, as well as their values. For a rule, the splitting into signal and background is useful for the treatment of systematics, as we will see later. Each rule will lead to a $\Delta\chi^2$-value, which means that all $\Delta\chi^2$'s of the different rules will be added for the whole experiment. Within each rule, the event rates are added, and the systematics is considered to be independent of the other rules (unless user-defined systematics specifies a dependence). Thus, it is convenient to combine the previously defined channels for different oscillation patterns and interaction types into one logical construction, which is the rule. For example, a superbeam usually has two rules: One for the ν_e-appearance rates, and one for the ν_μ-disappearance rates. In each case, contributions of several interaction types, as well as from the ν_e-contamination of the beam will lead to a number of contributing signal and background event channels.

For each rule, the signal event rate s_i in the ith bin can be composed out of one or more channels according to

$$s_i = \alpha_{c_{s1}} \cdot n_i^{c_{s1}} + \alpha_{c_{s2}} \cdot n_i^{c_{s2}} + \ldots \tag{11.21}$$

where the α's are overall normalization factors/efficiencies determined by the properties of the detector. Note that bin-based (energy-dependent) efficiencies can be defined with the post-smearing efficiencies in the last section. In addition, note that in most cases, it makes sense to have only one signal channel and to assign all sorts of perturbations to the background. Similarly, the background event rate b_i in the ith bin can be composed out of one or more channels:

$$b_i = \beta_{c_{b1}} \cdot n_i^{c_{b1}} + \beta_{c_{b2}} \cdot n_i^{c_{b2}} + \ldots\,, \tag{11.22}$$

where the channels can be any combination of the ones in the signal rate and additional ones. The background normalization factors very often have a specific meaning. For example, they may correspond to a fraction of mis-identified events (charge or flavor mis-identification). These basic building blocks of each rule are, within the rule environment, for example defined by

```
@signal = 0.5 @ #channel_1
@background = 0.001 @ #channel_2 :  0.005 @ #channel_3
```

For the analysis of the systematical errors, the so called "pull method" is used [25][4]. For the pull method, k systematical errors are included by introducing k additional variables ζ_k, which are the so-called "nuisance parameters". The nuisance parameters describe the dependence of the event rates on the various sources of systematical errors. For example, an error on the total normalization is included by multiplying the expected number of events in each bin by a factor $(1+\zeta_1)$. The variation of ζ_1 in the fit is constrained by adding a penalty p_1 to the χ^2-function. In case of a Gaußian distributed systematical error, this penalty is given by

$$p_i = \frac{\zeta_i^2}{\sigma_{\zeta_i}^2}\,, \tag{11.23}$$

where σ_{ζ_i} is the standard deviation of the corresponding nuisance parameter. In the following, we will refer to the standard deviation as the "error", since it corresponds to the actual systematical uncertainty. Note that the central values of all penalties are zero in GLoBES 3.0 and higher. The resulting χ^2 is then minimized with respect to all nuisance parameters ζ_i, which leads to χ^2_{pull}

$$\chi^2_{\mathrm{pull}}(\boldsymbol{\lambda}) := \min_{\{\zeta_i\}} \left(\chi^2(\boldsymbol{\lambda}, \zeta_1, \ldots, \zeta_k) + \sum_{j=1}^{k} p_j(\zeta_j) \right)\,. \tag{11.24}$$

Here $\boldsymbol{\lambda}$ refers to the oscillation parameters including the matter density ρ. One advantage of the pull method is that whenever the number N of data points is much larger than k, it is numerically easier to compute χ^2_{pull} than to invert the $N \times N$ covariance matrix. For the experiments considered here, N is typically 20 and $k \sim 4$, which means that the pull method is numerically much faster. Moreover, it is more flexible and allows the inclusion of systematical errors also for a Poissonian χ^2-function. In Ref. [25], it has been demonstrated that the pull method and the covariance based approach are equivalent for a Gaußian and linear model. In general, there is a separate $(\chi^2_{\mathrm{pull}})^r$ for each rule r, *i.e.*, pair of signal and background spectra, with a separate set of nuisance parameters ζ_i^r. Thus, χ^2_{pull} is the sum of all individual $(\chi^2_{\mathrm{pull}})^r$'s. By the minimization, the dependence on the k nuisance parameters is eliminated from χ^2_{pull}.

Now, we can introduce the different systematical errors. The two most important and most easily parameterized systematical errors are the normalization and energy calibration

[4] In fact the pull method was employed already in Ref. [6] before Ref. [25] appeared.

errors. These errors are assumed to be independent between the signal events and the background events, which means that this systematics treatment defines the grouping into signal or background. The implementation of the normalization error is straightforward:

$$s_i(a) := (1+a) \cdot s_i \tag{11.25}$$

with an analogous definition for the background events. Here, a is the "nuisance" parameter, which will be minimized over later.

For the parameterization of an energy calibration error, two possibilities are implemented. The first one (method "T") is somewhat simpler, whereas the second one (method "C") is more accurate, but it requires a careful choice of parameters. The first option (method "T") is[5]

$$s_i(a,b) \equiv s_i(a) + b \cdot s_i \, (E'_i - \bar{E}')/(E'_{\mathrm{max}} - E'_{\mathrm{min}}), \tag{11.26}$$

where E'_{min} and E'_{max} correspond to `$emin` and `$emax`, $\bar{E}' = \frac{1}{2}(E'_{\mathrm{max}} + E'_{\mathrm{min}})$ is the median of this energy interval, and E'_i is the mean (reconstructed) energy of the ith bin. This method is often referred to as a "tilt" of the spectrum, since it describes a linear distortion of the event rate spectrum. It also works for a variable bin width. The second option (method "C") is closer to an actual energy calibration error, which means that one should test this option whenever one suspects a large impact of this systematical error. It is based upon replacing the events in the ith bin by the ones at the energy $(1+b) \cdot E'_i$. If the target energy does not exactly hit a (discrete) bin energy E_k, linear interpolation is used. We use the following approximation:

$$\begin{aligned} s_i(a,b) &= (1+b) \cdot \left[(s_{k+1}(a) - s_k(a)) \cdot (\delta - k) + s_k(a) \right] , \\ \delta &= b \cdot (i + t_0 + 1/2) + i \, , \\ k &= \mathrm{div}(\delta, 1) \, , \\ t_0 &= E'_{\mathrm{min}}/\Delta E_0 \, . \end{aligned} \tag{11.27}$$

Here, ΔE_0 is the bin width (`$emax-$emin`)/`$bins`, and "div" refers to the integer part of the division. It is important to keep in mind that this definition of the energy calibration error makes sense only for constant bin widths, so the corresponding χ^2 functions should not be used in conjunction with the `$binwidth` directive. The factor $(1+b)$ in Eq. (11.27) comes from a renormalization of the bin width, since also the bin width is altered by the replacement of the energies. Furthermore, special attention has to be given to the limits $k < 1$ or $k+1 > N_{\mathrm{bins}}$, since there s_k or s_{k+1} may not have been calculated. By default, it is assumed that s_k is zero in those cases. However, if the event rates are still large at the limits, this will introduce errors, leading to a wrong estimate of the impact of the calibration error. In this case, one should truncate the analysis range by a few bins at the boundaries and thus ensure that only those s_i, whose index k is within the range $0, \ldots, N_{\mathrm{bins}} - 1$ (*cf.*, Fig. 11.1), are used. Therefore, it is possible to constrain the analysis energy range with each rule to an energy window:

[5]Note that this behavior has slightly changed compared to previous GLoBES releases.

Systematics function	a	b	c	d	Tilt/Calibr.	Dim.	Remarks
Standard systematics:							
`chiSpectrumTilt`	+	+	+	+	T	0	Systematics with tilt
`chiNoSysSpectrum`	-	-	-	-	-	2	No systematics, but spectral information
`chiTotalRatesTilt`	+	+	+	+	T	4	Total rates
`chiSpectrumOnly`	∞	-	-	-	-	7	Spectrum only
`chiNoSysTotalRates`	-	-	-	-	-	8	Total rates, no syst.
`chiSpectrumCalib`	+	+	+	+	C	9	Systematics with calibr.
User-defined systematics:							
`chiZero`	-	-	-	-	-	n/a	Passive rule (χ^2 returns zero)
Any other name	User-defined behavior					n/a	User-defined syst.

Table 11.3: Possible systematics χ^2 functions in GLoBES and their meaning. If a parameter is designated with +, it will be marginalized over, and therefore the corresponding error needs to have a non-zero value. In the cases with "total rates" in the remarks, the summation over the bins is performed *before* computing the χ^2, *i.e.*, no spectral information is used. The function `chiSpectrumOnly` leaves the normalization free ($\sigma_a = \sigma_c = \infty$), and therefore only the spectral information is used. As a consequence, the settings for the normalization error will be ignored (designated with the symbol ∞). In addition, the corresponding error dimension from earlier versions of GLoBES is shown in the column "Dim.".

```
@energy_window = 4.0 :  50.0
```

The default energy window is given by the minimal and maximal reconstructed energies `$emin` and `$emax`. To be on the safe side, one should reduce the analysis window compared to the bin range on each side by about three times the energy calibration error.

Eventually, the total event rate x_i in a bin i is given by

$$x_i(a,b,c,d) = s_i(a,b) + b_i(c,d)\,, \tag{11.28}$$

and is thus a function of four parameters. The four parameters a,b,c,d have been introduced in order to describe systematical uncertainties and are the nuisance parameters. Each of the four parameters has a corresponding systematical error. They are called signal normalization (a), signal tilt/calibration (b), background normalization (c) and background tilt/calibration (d). Their default (central) values are zero.[6] The errors for the normalization and the values of tilt/calibration are always regarded as pairs, *i.e.*, they are given in the form `normalization : tilt`. For example, we have

```
@signalerror = 0.001 :  0.01
@backgrounderror = 0.001 :  0.01
```

[6]The old parameter `@backgroundcenter` should not be used anymore. A background normalization center of 1.0 will be interpreted as zero central value.

The user has the possibility to choose the set $\{\zeta_i\}$ of nuisance parameters which are minimized over. This choice is specified with the systematics functions `sys_on_function` and `sys_off_function` corresponding two systematics modes "systematics on" and "systematics off".[7] The different possibilities are shown in Table 11.3. Since the dual systematics modes define the behavior of the experiment for systematics on and off, it is useful to have a matching pair of systematics functions for each rule (see also Sec. 7.1). The signal and background errors specified by `@signalerror` and `@backgrounderror` will then be used, if applicable. For example, one may define

```
@signalerror = 0.001 :  0.01
@backgrounderror = 0.001 :  0.01
@sys_on_function = "chiSpectrumTilt"
@sys_off_function = "chiNoSysSpectrum"
```

For user-defined systematics (see, Sec. 3.2), one can use arbitrary names for the systematics functions which are not pre-defined. In this case, one would specify the systematical errors in lists, such as

```
@sys_on_function = "chiMySystematics"
@sys_on_errors = { 0.2, 0.3, 0.5 } // Uses three syst.  errors
@sys_off_function = "chiNoSysSpectrum"
@sys_off_errors = { }
```

Note that here, the systematics on and systematics off models can be used as two different, fully functional systematics modes with different systematical errors. The interpretation of the systematical errors in `@sys_on_errors` and `@sys_off_errors` is left to the application software code and the user. In addition, the application software has to register the user-defined systematics χ^2 function with `glbDefineChiFunction` and uniquely identify the name given by `@sys_on_function` or `@sys_off_function` to avoid confusing different systematics routines. In addition, it is possible to define a rule with passive systematics using `glbChiZero`. In this case, the contribution to χ^2 from this rule will always be set to zero, but the corresponding rate vectors will be calculated and provided for indirect access by other systematics functions. For example, a reactor experiment with correlated systematics between near and far detectors may define the user-defined systematics `chiReactor` for the far detector and use `chiZero` for the near detector. The routine assigned to the far detector will then perform the χ^2 calculation, which, of course, also involves the rates of the near detector. However, defining `chiReactor` in both the near and far detectors would result in a double call of the χ^2 function, *i.e.*, the resulting χ^2 would be too large by a factor two and the program would be slower by a factor of two. Note that mixed declarations of `@sys_on_errors`, `@sys_off_errors`, `@signalerror`, and `@backgrounderror` are possible. In this case, `@sys_on_errors` and `@sys_off_errors` have priority. In addition, `@sys_on_errors` and `@sys_off_errors` can be used for built-in systematics in

[7] In earlier versions before GLoBES 3.0 the "error dimension" was used. The corresponding parameters `errordim_sys_on` and `errordim_sys_off` are still supported but should not be used anymore.

the order: signal normalization, signal tilt (energy calibration), background normalization, background tilt (energy calibration). If one or both of these are not present, `@signalerror` or `@backgrounderror` will be used.

Eventually, a complete rule may look like this:

```
rule(#rule_1)<
  @signal = 0.5 @ #channel_1
  @background = 0.001 @ #channel_2 : 0.005 @ #channel_3
  @signalerror = 0.001 : 0.01
  @backgrounderror = 0.001 : 0.01
  @sys_on_function = "chiSpectrumTilt"
  @sys_off_function = "chiNoSysSpectrum"
  @energy_window = 4.0 : 50.0
>
```

Chapter 12

Testing & debugging of AEDL files

AEDL is a powerful language to describe a variety of different experiments. This chapter demonstrates how to test an AEDL file in order to check if it really describes a given experiment. For this application, the GLoBES package contains the program `globes`. It can either be regarded as an AEDL debugger, or as a simple command-line oriented tool to convert the rather abstract AEDL experiment description into more accessible event rates.

12.1 Basic usage of the `globes` binary

The `globes` binary is installed together with the library, but into the directory `$prefix/bin/`. In order to use the `globes` utility, this directory has to be in the path of the shell used to call the program.[1]

As an argument, `globes` takes a `.glb`-file. While parsing it, it prints any warnings and errors which occur while reading the file. Then it uses the experiment description in the file to compute the event rates at a certain point in parameter space. Finally, it displays the result based on the options used to call `globes`. The options of `globes` follow the GNU standard. Thus, there is a `--help` option to display all other options together with short descriptions.

Calling `globes` without any options and with a `.glb`-file as argument produces an event summary at rule level. In this case, the full experiment description in the file is taken into account, *i.e.*, all efficiencies, backgrounds, and energy resolution effects. Thus, the returned event rates are the ones which will be actually used to compute the χ^2 later. By default, the oscillation parameters used to calculate the transition probability are

$$\begin{aligned} \sin^2 2\theta_{12} &= 0.8 & \Delta m_{21}^2 &= 7 \cdot 10^{-5}\,\mathrm{eV}^2\,, \\ \sin^2 2\theta_{23} &= 1.0 & \Delta m_{31}^2 &= 3 \cdot 10^{-3}\,\mathrm{eV}^2\,, \\ \delta &= 0 & \sin^2 2\theta_{13} &= 0.1\,. \end{aligned} \tag{12.1}$$

[1] This is automatically the case if no options are given to `configure`, and `make install` was executed with root-privilege, *i.e.*, a standard installation was done.

Of course, it is possible to change these default values either by using the option `-p` on a call by call basis, or by setting the environment variable `GLB_CENTRAL_VALUES`:

```
globes -p'0.55,0,0.785,0,0.0008,0.0025'
globes --parameters='0.55,0,0.785,0,0.0008,0.0025'
```

For example, `GLB_CENTRAL_VALUES` can be defined within the shell session or in the shell profile:

```
export GLB_CENTRAL_VALUES='0.55,0,0.785,0,0.0008,0.0025'
```

Note that in the case of additional non-standard parameters, these can not be included in `globes` (the command behaves as for the standard three-flavor case, since otherwise a re-compilation of the software were necessary). Furthermore it is possible to switch off oscillations with the `-N` option and to switch them on again with `-O` (the default). The effect of `-N` is the same as to use `NOSC_` in all oscillation channels. This feature is useful if one wants to normalize the flux in an experimental to a given number of un-oscillated events.

The AEDL parser and interpreter have basically three levels of messages to the user: Warnings, errors and fatal errors. Fatal errors are always reported and lead to a program exit with status '1'. Usually only errors and no warnings are reported. The verbosity level can be chosen by the `-v` option, where `-v1` is default, *i.e.*, only errors and fatal errors are reported. The level -v0 corresponds to reporting fatal errors only, and `-v2` will print warnings in addition to fatal errors. It is recommended to test any new `.glb`-file with `-v2` to check the warnings at least once, and to decide whether there is a problem to be fixed. With `-v3` all files read by `globes` are displayed together with their path, and with `-v4` all files which have been attempted to be read are shown. These two setting are useful to clarify path resolution issues and shadowing of file names.

12.2 Testing AEDL files

In the process of defining a new experiment, the default output of `globes` at rule level is the final step. However, in order to arrive at this level it is often necessary to review the intermediate steps in the event rate calculation. The `globes` utility offers many possibilities to do this based on the rate access functions described in Sec. 6.3.

By default, `globes` returns total rates corresponding to the `-t` option. This can be changed to to a full spectrum by using `-s`. The spectral rates are shown in a table where the first column always gives the central energy of the corresponding bin or the sampling point.

If there is more than one experiment in a file, *i.e.*, there is at least one `#NEXT#` command, only the event rates for one experiment will be shown. This experiment can be chosen with the `-e` option, which takes as a mandatory argument the number of the experiment (starting with zero). The default is `-e0`.

Channel level

As a first step, one may want to check if each channel produces the anticipated output. Channel rates are returned if the `-c` option is used. This option takes as an optional argument the channel number (starting at zero). If no argument is given, all channels are displayed. By default, the sum of the event rates in each channel is shown. Each column has as first line the same channel name as in the file.

It is also possible to switch off one detector effect after the other. First, one can switch off the post-smearing efficiencies (`-f`) and the post-smearing backgrounds (`-g`). Next, one can switch off the energy resolution function with (`-b`) and view the rates before smearing. If the `-s` option is also used, the number of lines in the output will be given by `$sampling_points`. Another effect of the `-b` option is that the post-smearing efficiencies and backgrounds are no longer taken into account. Therefore, the `-g` and `-f` options now apply to the *pre*-smearing efficiencies and the *pre*-smearing backgrounds. Thus,

```
globes -c -b -g -f FILE
```

produces the raw event rate corresponding to the convolution of flux, probability, and cross section, neglecting all detector effects.

Rule level

The next logical step after checking the channel rates is to investigate the rule rates. The rule rates are returned with the option `-r`. This option takes as an optional argument the rule number (starting at zero). If no argument is given, all rules will be displayed. By default, the signal and background rates in each rule are shown separately, together with their decomposition into the different contributing channels. Each rule is preceeded by a line with the same rule name as in the file.

Also for the rules, it is possible to switch off one detector effect after the other – with the limitation that rules only make sense after the energy resolution function has been applied to each channel. Therefore, it is *not* possible to use `-b` together with `-r`, or to switch off any pre-smearing efficiencies or backgrounds. One can, however, switch off the post-smearing efficiencies (`-f`) and the post-smearing backgrounds (`-g`) for each channel. Since the definition of a rule also contains so-called “coefficients”, it is possible to switch them off with `-i`.

Output

The default output stream is `stdout`. The output can be re-directed to a file using the `-o` option, which takes as mandatory argument the file name. The default output format aims at maximal readability for a human eye. In many cases however, the output of `globes` is produced as input for other programs. There are some features to adjust the output format. Usually one would like to omit the channel and rule names by using simple printing `-S` instead of pretty printing `-P`.

There are special options for certain special formats: `-m` produces Mathematica[2] list output, which can be directly visualized by `MultipleListPlot`. The option `-u` uses in principle the same formatting as `-m`, but it allows to specify the left, middle, and right delimiters in constructing the list, such as

```
left
left 1 middle 2 middle 3 right
middle
left
left 1 middle 2 middle 3 right
right
```

This is, with `left` = '{', `middle` = ',' and `right` = '}', equivalent to the list $\{\{1,2,3\},\{1,2,3\}\}$. The delimiters can be set by `-L`, `-M` and `-R` as in the following example:

```
globes -Su -R$'\n' --Middle=" " -L" " ...
```

Here `$'\n'` is the escape sequence in the shell for ANSI C-like characters, such as linefeed '`\n`'. The above example produces a a two column file such as

1.0 0.12
1.2 0.14
1.3 0.18
...

where the first column is the central energy of the bin or the sampling point, and the second column gives the event rate. Usually, the output is a concatenation of many such two columns tables, where each rule part or channel part has its own table. Thus one can, by using `-u` and user-defined delimiters, construct many different output formats.

AEDL external variable substitution*

Some `.glb`-files use external AEDL variables in order to allow special purpose studies (such as the energy resolution-dependence). If the external variables are not explicitely specified, they are interpreted by the parser as zeros. Thus, it is impossible to properly parse any files with `globes` which contain such undefined variables. Hence, there is the possibility to define AEDL variables by using the define option `-D`. The example

```
globes -DBASELINE=3000 -D%BLUE=\{8, 15\} ...
```

defines the AEDL variable `BASELINE` to be 3000 and the AEDL variable list `%BLUE` to be {8,15} (please note the syntax for the brackets!).

[2]Mathematica is a trademark of Wolfram Inc.

Acknowledgments

We would like to thank Martin Freund, who wrote the very first version of a three-flavor matter profile treatment many years ago, and Thomas Schwetz, who has been pushing the software to the edge in the past few years. Furthermore, we would like to thank Tommy Ohlsson, Toshihiko Ota, and Julian Skrotzki for using and testing unpublished new features of the software. PH is especially thankful for the invaluable advice of Thomas Fischbacher on many design issues in the early stage of the project. Finally, thanks to all the people who have been pushing this project for many years, to the ones who have been continuing asking for the publication of the software, and the referees of several of our papers for suggestions which lead to improvements in the software.

This work and the development of GLoBES have over the years been supported by (in chronological order):

- Technische Universität München [All authors]
- Max-Planck-Institut für Physik, München [PH]
- Sonderforschungsbereich 375 für Astro-Teilchenphysik der Deutschen Forschungsgemeinschaft [All authors]
- Studienstiftung des Deutschen Volkes [JK, WW]
- Institute for Advanced Study, Princeton [WW]
- W. M. Keck Foundation [WW]
- National Science Foundation [WW]
- University of Wisconsin, Madison [PH]
- Max-Planck-Institut für Kernphysik, Heidelberg [JK, ML]
- Emmy Noether-Programm der Deutschen Forschungsgemeinschaft [WW]
- Universität Würzburg [WW]

Appendix A

GLoBES installation

A.1 Prerequisites for installation of GLoBES

Besides the usual things like a working libc you need 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 libraries for the above packages are needed, but also the headers, especially for GSL. For some installations of GSL, eg. on RedHat/Fedora, 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. Furthermore you need a working `make` to build and install GLoBES.

A.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 libraries and linking against them see the corresponding section of this file.

Basic Installation

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 have installed a library don't forget to run `ldconfig`!

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!

Building and Using static versions of GLoBES

Under 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 a 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.

GLoBES and Condor

> Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management.

A Condor (www.cs.wisc.edu/condor/) cluster is very well suited to run large GLoBES-based computation. The nature of the problems addressed with GLoBES is such that one typically ends up with a so called 'embarrassingly parallel' program. That means, that one repeats the same task N times, where each execution is independent of the other $N-1$. Therefore, this execution should become M times faster if one uses M processors. For this class of problems running on a dedicated cluster will not improve performance (but may reduce latency and such).

In order to fully exploit the functionality offered by Condor one should submit the jobs into the so called 'standard universe'. To do this, it is necessary to re-link the application with the Condor-library (this assumes that Condor is installed)

```
condor_compile gcc your_object_files -static `globes-config --libs`
```

It may be necessary to prefix the call of globes-config with the path to it, in case that this location is not in $PATH.

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 can 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.

Distributions

RedHat (all versions)

The standard rpm-based installation of GSL does not provide any header files for GSL, which are however needed to compile GLoBES. You have to install an additional rpm-package called `gsl-devel`. Alternatively you can install GSL from a tar-ball and use the `--with-gsl-prefix` option to the configure script of GLoBES.

Platforms

GLoBES builds and installs on 64bit Linux systems. GLoBES should work on Mac OS.

Windows

Currently GLoBES is only able to work under Cygwin `www.cygwin.com`. Inside Cygwin GLoBES needs to be built with these commands

```
configure
make LDFLAGS=-no-undefined'
```

Compilers and Options

Some systems require unusual options for compilation or linking that the `configure` script does not know about. You can give `configure` initial values for variables by setting them in the environment. Using a Bourne-compatible shell, you can do that on the command line like this

```
CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure
```

Or on systems that have the `env` program, you can do it like this

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of `make` that supports the `VPATH` variable, such as GNU `make`. `cd` to the directory where you want the object files and executables to go and run the `configure` script. `configure` automatically checks for the source code in the directory that `configure` is in and in `..`.

If you have to use a `make` that does not supports the `VPATH` variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use `make distclean` before reconfiguring for another architecture.

Installation Names

By default, `make install` will install the package's files in `/usr/local/bin`, `/usr/local/man`, etc. You can specify an installation prefix other than `/usr/local` by giving `configure` the option `--prefix=PATH`.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give `configure` the option `--exec-prefix=PATH`, the package will use `PATH` as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like `--bindir=PATH` to specify different values for particular kinds of files. Run `configure --help` for a list of the directories you can set and what kinds of files go in them.

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving `configure` the option `--program-prefix=PREFIX` or `--program-suffix=SUFFIX`.

Optional Features

Some packages pay attention to `--enable-FEATURE` options to `configure`, where `FEATURE` indicates an optional part of the package. They may also pay attention to `--with-PACKAGE` options, where `PACKAGE` is something like `gnu-as` or `x` (for the X Window System). The `README` should mention any `--enable-` and `--with-` options that the package recognizes.

For packages that use the X Window System, `configure` can usually find the X include and library files automatically, but if it doesn't, you can use the `configure` options

`--x-includes=DIR` and `--x-libraries=DIR` to specify their locations.

Building a perl extension

This feature is experimental and your mileage may vary!

This feature allows to build a perl binding of GLoBES i.e. you will in the end have a perl module from which you can use GLoBES from within any perl program.

If(!) everything works as intended, all you have to do is to provide `--enable-perl` to `configure` and type `make install`. Now have a look at `globes/example.pl` and you should see how that works in principle.

The trick here is, that we use SWIG (`www.swig.org`) to generate a wrapper file for GLoBES. The wrapper file is part of the GLoBES tar-ball (`globes/globes_perl.c`) and hence you should not need SWIG to be installed on your system.

All the tricks employed to get perl extension working should in some form be applicable to building other extensions, like python. If you want to try that you will need SWIG.

Building RPMs

This feature is experimental and your mileage may vary!

Many people find binary RPMs useful, therefore we provide an optional feature `--enable-rpm-rules` which should produce all the necessary Makefile rules for RPM building. To actually build RPMs requires that your system is properly setup for that. You can learn how to do that at `http://www.rpm.org`. You then can use `make rpm`, most likely you will need to be root to do that (`sudo` won't work!).

NOTE to people packaging GLoBES RPMs: Please, use the provided spec file and do include the headers!

Specifying the System Type

There may be some features `configure` can not figure out automatically, but needs to determine by the type of host the package will run on. Usually `configure` can figure that out, but if it prints a message saying it can not guess the host type, give it the `--host=TYPE` option. `TYPE` can either be a short name for the system type, such as `sun4`, or a canonical name with three fields

```
CPU-COMPANY-SYSTEM
```

See the file `config.sub` for the possible values of each field. If `config.sub` isn't included in this package, then this package doesn't need to know the host type.

If you are building compiler tools for cross-compiling, you can also use the `--target=TYPE` option to select the type of system they will produce code for and the `--build=TYPE` option to select the type of system on which you are compiling the package.

Sharing Defaults

If you want to set default values for `configure` scripts to share, you can create a site shell script called `config.site` that gives default values for variables like `CC`, `cache_file`, and `prefix`. `configure` looks for `PREFIX/share/config.site` if it exists, then `PREFIX/etc/config.site` if it exists. Or, you can set the `CONFIG_SITE` environment variable to the location of the site script. A warning: not all `configure` scripts look for a site script.

Appendix B

Catalogue of AEDL-Files

Along with the GLoBES package comes a catalogue of pre-defined experiment AEDL files for different future experiments and different beam and detector technologies. These include the planned superbeam experiments and their possible upgrades, different reactor experiment setups, different β-beam setups, and different neutrino factory setups. A complete list of all pre-defined experiment files can be found in Table 2.1. More detailed descriptions of the corresponding files, the assumptions, requirements, and references are given in the following.

B.1 Superbeam Experiments

Experiment	File	Runtime	Power	Baseline	Detector	Mass
T2K	`T2K.glb`	2 yr ν/6 yr $\bar{\nu}$	0.77 MW	295 km	WC	22.5 kt
T2HK	`T2HK.glb`	4 yr ν/4 yr $\bar{\nu}$	4 MW	295 km	WC	500 kt
NOνA	`NOvA.glb`	3 yr ν/3 yr $\bar{\nu}$	1.12 MW	812 km	TASD	25 kt
SPL	`SPL.glb`	2 yr ν/8 yr $\bar{\nu}$	4 MW	130 km	WC	500 kt

T2K – `T2K.glb`

The T2K experiment can be simulated with the file `T2K.glb`. This file tries to approximate as closely as possible the LOI [5], and the basic version was used within [6]. These references should be cited if the file `T2K.glb` is used for a scientific publication or a talk. For calculations that involve `T2K.glb`, the following additional files are required:

- `JHFplus.dat` (neutrino flux from J-PARC – ν_μ)
- `JHFminus.dat` (neutrino flux from J-PARC – $\bar{\nu}_\mu$)
- `XCC.dat` (charged current cross sections)
- `XNC.dat` (neutral current cross sections)

- `XQE.dat` (quasi elastic cross sections)

The T2K neutrino beam is produced at J-PARC and directed towards the Super-Kamiokande detector. The target power is 0.77 MW and 2 years ν-running and 6 years $\bar{\nu}$-running is assumed. The fiducial mass of the Super-Kamiokande Water Cerenkov detector is taken to be $\mathrm{m_{det}} = 22.5\,\mathrm{kt}$ at a baseline of $\mathrm{L} = 295\,\mathrm{km}$. The appearance measurement involves the total rates data from all CC events and the spectral data from the QE sample with a free normalization at an energy resolution of $\sigma_\mathrm{e} = 0.085\,\mathrm{GeV}$ due to the Fermi Motion. The normalization of the QE samples is kept free in order to avoid double counting of events since all QE events are also contained in the CC samples. The free normalization is introduced within a rule with the line `@signalerror = 10.:0.0001`. In case of systematics switched on the systematics functions `chiSpectrumTilt` for the QE sample and `chiTotalRatesTilt` for the CC sample are used. However, for systematics switched off, only the systematics function for the CC sample is changed to `chiNoSysTotalRates`, but the systematics function for the QE sample stays `chiSpectrumTilt`. Note, that otherwise the free normalization would be switched off and all events from the QE sample would be counted twice. A detailed discussion of this QE/CC sample splitting can be found in [6]. For the disappearance channels only the QE sample is used since statistics is already quite large and a treatment as for the appearance channels would only slightly modify the results. The quantitative treatment of systematics is similar to [26]. The following rules are defined within `T2K.glb`:

Disappearance (+)			σ_norm	σ_cal
Signal	$0.9 \otimes (\nu_\mu \to \nu_\mu)_\mathrm{QE}$		0.025	10^{-4}
Background	$0.0056 \otimes (\nu_\mu \to \nu_x)_\mathrm{NC}$		0.2	10^{-4}
Appearance (+) – Spectrum				
Signal	$0.505 \otimes (\nu_\mu \to \nu_e)_\mathrm{QE}$		10.0	10^{-4}
Background	$0.0056 \otimes (\nu_\mu \to \nu_x)_\mathrm{NC}$	$3.3 \cdot 10^{-4} \otimes (\nu_\mu \to \nu_\mu)_\mathrm{CC}$	0.05	0.05
Beam background	$0.505 \otimes (\nu_e \to \nu_e)_\mathrm{CC}$	$0.505 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_\mathrm{CC}$	0.05	0.05
Appearance (+) – Total Rates				
Signal	$0.505 \otimes (\nu_\mu \to \nu_e)_\mathrm{CC}$		0.05	10^{-4}
Background	$0.0056 \otimes (\nu_\mu \to \nu_x)_\mathrm{NC}$	$3.3 \cdot 10^{-4} \otimes (\nu_\mu \to \nu_\mu)_\mathrm{CC}$	0.05	10^{-4}
Beam background	$0.505 \otimes (\nu_e \to \nu_e)_\mathrm{CC}$	$0.505 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_\mathrm{CC}$	0.05	10^{-4}
Disappearance (–)				
Signal	$0.9 \otimes (\nu_\mu \to \nu_\mu)_\mathrm{QE}$		0.025	10^{-4}
Background	$0.0056 \otimes (\nu_\mu \to \nu_x)_\mathrm{NC}$		0.2	10^{-4}

Appearance (–) – Spectrum			σ_{norm}	σ_{cal}
Signal	$0.505 \otimes (\bar{\nu}_\mu \to \bar{\nu}_e)_{\text{QE}}$		10.0	10^{-4}
Background	$0.0056 \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\text{NC}}$	$3.3 \cdot 10^{-4} \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\text{CC}}$	0.05	0.05
Beam background	$0.505 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\text{CC}}$	$0.505 \otimes (\nu_e \to \nu_e)_{\text{CC}}$	0.05	0.05
Appearance (–) – Total Rates				
Signal	$0.505 \otimes (\bar{\nu}_\mu \to \bar{\nu}_e)_{\text{CC}}$		0.05	10^{-4}
Background	$0.0056 \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\text{NC}}$	$3.3 \cdot 10^{-4} \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\text{CC}}$	0.05	10^{-4}
Beam background	$0.505 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\text{CC}}$	$0.505 \otimes (\nu_e \to \nu_e)_{\text{CC}}$	0.05	10^{-4}

T2HK – `T2HK.glb`

T2HK is the superbeam upgrade of the T2K experiment and can be simulated with the file `T2HK.glb`. The target power is 4 MW and 4 years ν-running and 4 years $\bar{\nu}$-running is assumed. The fiducial mass of the Water Cerenkov detector is taken to be $\mathrm{m_{det}} = 500\,\mathrm{kt}$ at the same baseline as the T2K experiment. Besides these changes, the file `T2HK.glb` is similar to `T2K.glb` and the same additional files are required. The basic version was used within [6] which should be cited if the file `T2K.glb` is used for a scientific publication or a talk.

NOνA – `NOvA.glb`

The NOνA experiment can be simulated with the file `NOvA.glb`. The description of the disappearance channels is taken from [8] and the description of the appearance channels follows the proposal [7]. These references should be cited if the file `NOvA.glb` is used for a scientific publication or a talk. For calculations that involve `NOvA.glb`, the following additional files are required:

- `NOvAplus.dat` (NuMI neutrino flux – ν_μ)
- `NOvAminus.dat` (NuMI neutrino flux – $\bar{\nu}_\mu$)
- `XCC.dat` (charged current cross sections)
- `XNC.dat` (neutral current cross sections)

The NOνA experiment uses a neutrino beam from the Fermilab NuMI beamline. The source power is 10^{21} pot yr^{-1}, corresponding to a (naively computed) target power of 1.12 MW. For the running times, 3 years ν-running and 3 years $\bar{\nu}$-running are assumed. The fiducial mass of the Totally Liquid Scintillator Detector (TASD) is taken to be $\mathrm{m_{det}} = 25\,\mathrm{kt}$ at a baseline of $\mathrm{L} = 812$ km approximately 12 km off-axis to the beamline. The energy resolution is $\sigma_{\mathrm{e}} = 10\% \cdot \sqrt{\mathrm{E}}$ for electrons and $\sigma_{\mathrm{e}} = 5\% \cdot \sqrt{\mathrm{E}}$ for muons. The following rules are defined within `NOvA.glb`:

Disappearance (+)			σ_{norm}	σ_{cal}
Signal	$0.8 \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$		0.05	0.025
Background	$0.0015 \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$		0.05	0.025
Appearance (+)				
Signal	$0.24 \otimes (\nu_\mu \to \nu_e)_{\mathrm{CC}}$		0.05	0.025
Background	$0.0015 \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$	$1.0 \cdot 10^{-4} \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$	0.05	0.025
Beam background	$0.12 \otimes (\nu_e \to \nu_e)_{\mathrm{CC}}$		0.05	0.025
Disappearance (–)				
Signal	$0.8 \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\mathrm{CC}}$		0.05	0.025
Background	$0.0015 \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\mathrm{NC}}$		0.05	0.025
Appearance (–)				
Signal	$0.37 \otimes (\bar{\nu}_\mu \to \bar{\nu}_e)_{\mathrm{CC}}$		0.05	0.025
Background	$0.0037 \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\mathrm{NC}}$	$1.0 \cdot 10^{-4} \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\mathrm{CC}}$	0.05	0.025
Beam background	$0.12 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\mathrm{CC}}$		0.05	0.025

SPL - `SPL.glb`

The SPL experiment can be simulated with the file `SPL.glb`. This file was used in [9] and follows the experiment description from [10, 11]. These references should be cited if the file `SPL.glb` is used for a scientific publication or a talk. For calculations that involve `SPL.glb`, the following additional files are required:

- `SPLplus.dat` (neutrino flux from CERN – ν_μ)
- `SPLminus.dat` (neutrino flux from CERN – $\bar{\nu}_\mu$)
- `Mig_WC_numu.dat` (migration matrix – ν_μ)
- `Mig_WC_numubar.dat` (migration matrix – $\bar{\nu}_\mu$)
- `Mig_WC_nue.dat` (migration matrix – ν_e)
- `Mig_WC_nuebar.dat` (migration matrix – $\bar{\nu}_e$)
- `XCC_spl.dat` (charged current cross sections)
- `XNC_spl.dat` (neutral current cross sections)

The SPL experiment uses a neutrino beam from the CERN to Fréjus. The target power is 4 MW and 2 years ν-running and 8 years $\bar{\nu}$-running is assumed. The fiducial mass of the Water Cerenkov detector is taken to be $\mathrm{m}_{\mathrm{det}} = 500\,\mathrm{kt}$ at a baseline of $\mathrm{L} = 130$ km.

The energy resolution is introduced manually by four migration matrices (for $\nu_e, \bar{\nu}_e, \nu_\mu, \bar{\nu}_\mu$) that describe energy smearing due to Fermi Motion. The following rules are defined within `SPL.glb`:

Disappearance (+)			σ_{norm}	σ_{cal}
Signal	$1.0 \otimes (\nu_\mu \to \nu_\mu)_{\text{CC}}$	(energy dep. efficiency)	0.02	10^{-4}
Background	$4.3 \cdot 10^{-5} \otimes (\nu_\mu \to \nu_\mu)_{\text{CC}}$		0.02	10^{-4}
Appearance (+)				
Signal	$0.707 \otimes (\nu_\mu \to \nu_e)_{\text{CC}}$		0.02	10^{-4}
Background	$6.5 \cdot 10^{-4} \otimes (\nu_\mu \to \nu_x)_{\text{NC}}$	$5.4 \cdot 10^{-4} \otimes (\nu_\mu \to \nu_\mu)_{\text{CC}}$	0.02	10^{-4}
	$0.7 \otimes (\bar{\nu}_\mu \to \bar{\nu}_e)_{\text{CC}}$		0.02	10^{-4}
Beam Background	$0.677 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\text{CC}}$	$0.707 \otimes (\nu_e \to \nu_e)_{\text{CC}}$	0.02	10^{-4}
Disappearance (–)				
Signal	$1.0 \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\text{CC}}$	(energy dep. efficiency)	0.02	10^{-4}
Background	$4.3 \cdot 10^{-5} \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\text{CC}}$		0.02	10^{-4}
Appearance (–)				
Signal	$0.677 \otimes (\bar{\nu}_\mu \to \bar{\nu}_e)_{\text{CC}}$		0.02	10^{-4}
Background	$0.0025 \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\text{NC}}$	$5.4 \cdot 10^{-4} \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\text{CC}}$	0.02	10^{-4}
	$0.7 \otimes (\nu_\mu \to \nu_e)_{\text{CC}}$		0.02	10^{-4}
Beam Background	$0.677 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\text{CC}}$	$0.707 \otimes (\nu_e \to \nu_e)_{\text{CC}}$	0.02	10^{-4}

B.2 Reactor Experiments

Experiment	File	Runtime	th. Power	Baseline	Det.	Mass
Small	`Reactor1.glb`	5 yr $\bar{\nu}$	4 GW	1.7 km	LS	20 t
Large	`Reactor.glb`	8 yr $\bar{\nu}$	10 GW	1.7 km	LS	100 t
Double Chooz	`D-Chooz_near.glb`	5 yr $\bar{\nu}$	2×4.2 GW	0.1 km	LS	10.16 t
	`D-Chooz_far.glb`	5 yr $\bar{\nu}$	2×4.2 GW	1.05 km	LS	10.16 t

Small Reactor Experiment – `Reactor1.glb`

The file `Reactor1.glb` allows to simulate a small $\bar{\nu}_e$-disappearance reactor experiment. The basic version of this file was used within [12] which should be cited if the file `Reactor1.glb` is used for a scientific publication or a talk. For calculations that involve `Reactor1.glb`, the following additional files are required:

- `Reactor.dat` (neutrino flux from reactor)

- `XCCreactor.dat` (charged current cross sections for low energies)

The neutrino source is the core of a nuclear power reactor. The integrated luminosity is assumed to be $\mathcal{L} = 400\,\mathrm{t\,GW\,yr}$, such as for a 20 t detector, a reactor with a thermal power of 4 GW, and a running period of 5 years. As detector technology, a liquid scintillator detector is assumed, a far detector at a baseline of L = 1.7 km and a near detector which is assumed to be identical to the far detector (maybe apart from the size) in order to minimize the impact of systematical uncertainties. The near detector is simulated implicitly by lower effective systematical errors, and there are no external backgrounds assumed. The normalization error used in the file `Reactor1.glb` has to be considered as an effective error, receiving contributions from individual uncertainties (see Ref. [12]). The energy resolution is $\sigma_\mathrm{e} = 5\% \cdot \sqrt{\mathrm{E_{vis}}}$ and the choice for `@sigma_function` is `#inverse_beta`. The following rules are defined within `Reactor1.glb`:

Disappearance		σ_norm	σ_cal
Signal	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_\mathrm{CC}$	0.008	0.005
Background	$5.8 \cdot 10^{-5} \otimes (\bar{\nu}_e \to \bar{\nu}_e)_\mathrm{CC}$	10^{-6}	10^{-6}

Large Reactor Experiment – `Reactor2.glb`

The file `Reactor2.glb` allows to simulate a large $\bar{\nu}_e$-disappearance reactor experiment. The basic version of this file was used within [12] which should be cited if the file `Reactor2.glb` is used for a scientific publication or a talk. The integrated luminosity is assumed to be $\mathcal{L} = 8000\,\mathrm{t\,GW\,yr}$, *e.g.* a 100 t detector, a reactor with a thermal power of 10 GW and a running period of 8 years. Besides the higher integrated luminosity, the attributes of `Reactor2.glb` are similar to the ones of `Reactor1.glb`.

Double Chooz – `D-Chooz_near.glb` and `D-Chooz_far.glb`

The files `D-Chooz_near.glb` and `D-Chooz_far.glb` allow to simulate the DoubleChooz reactor experiment in France. They require user-defined systematics and GLoBES 3.0 or higher, where the user-defined systematics function can be found in the header of `D-Chooz_near.glb` or `example5.c`. The basic versions of these files were used within [13], which should be cited if the files `D-Chooz_near.glb` and `D-Chooz_far.glb` are used for a scientific publication or a talk. For calculations that involve `D-Chooz_near.glb` and/or `D-Chooz_far.glb`, the following additional files are required:

- `Reactor.dat` (neutrino flux from reactor)
- `XCCreactor.dat` (charged current cross sections for low energies)

The DoubleChooz experiment is located at the Chooz reactor complex, and the two reactor cores serve as $\bar{\nu}_e$ neutrino source, so the thermal power is $2 \cdot 4.2\,\mathrm{GW}$. Two identical liquid scintillator detectors with a fiducial mass of $\mathrm{m_{det}} = 10.16\,\mathrm{t}$ are used as near and far detector. The far detector is planned to be located in the CHOOZ cavern at a baseline of L = 1.05 km from the two reactor cores and the near detector is assumed to be located at a distance of 0.1 km to the cores. The total running time of the experiment is assumed to be 5 years. So, the integrated luminosity at the far detector yields $\mathcal{L} \approx 427\,\mathrm{t\,GW\,yr}$. Here, the total running time of 5 years is assumed within `D-Chooz_near.glb` and `D-Chooz_far.glb`, so near and far detector are assumed to start the mode of operation simultaneously. For the simulation of DoubleCHOOZ and considering a delayed start of data taking at the near detector the file `D-Chooz_near.glb` has to be modified. The cancellation of systematical uncertainties is considered by the manual definition of a χ^2 as described in [13] with the treatment of user-defined systematics as described in Sec. 3.2 The energy resolution is $\sigma_\mathrm{e} = 5\% \cdot \sqrt{\mathrm{E_{vis}}}$ and the choice for `@sigma_function` is `#inverse_beta`. The following rules are defined within `D-Chooz_near.glb` and `D-Chooz_far.glb`:

Disappearance			σ_{flux}	σ^N_{fid}	σ^F_{fid}	σ^N_{cal}	σ^F_{cal}
Signal	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\mathrm{CC}}$		0.02	0.006	0.006	0.005	0.005
Background	neglected	(sys. dominates)	–	–	–	–	–

B.3 Beta Beam Experiments

Experiment	File	Runtime	γ	Baseline	Det.	Mass
Low γ	`BB_100.glb`	4 yr ν/4 yr $\bar{\nu}$	100	130 km	WC	500 kt
Medium γ	`BB_350.glb`	4 yr ν/4 yr $\bar{\nu}$	350	730 km	WC	500kt
Variable γ	`BBvar_WC.glb`	4 yr ν/4 yr $\bar{\nu}$	variable	variable	WC	500 kt
Variable γ	`BBvar_TASD.glb`	4 yr ν/4 yr $\bar{\nu}$	variable	variable	TASD	50 kt

CERN-Fréjus Baseline Scenario – `BB_100.glb`

The $\gamma = 100$ β-beam baseline scenario from CERN to Fréjus can be simulated with the file `BB_100.glb`. The basic version of this file was used within [9]. This reference should be cited if the file `BB_100.glb` is used for a scientific publication or a talk. For calculations that involve `BB_100.glb`, the following additional files are required:

- `BB100flux_Ne.dat` (β-beam neutrino flux – ^{18}Ne stored at $\gamma = 100$)
- `BB100flux_He.dat` (β-beam neutrino flux – ^{6}He stored at $\gamma = 100$)
- `BeamBckg_100.dat` (beam background)
- `AtmBckg_100.dat` (atmospheric background)
- `Mig_WC_numu.dat` (migration matrix – ν_μ)
- `Mig_WC_numubar.dat` (migration matrix – $\bar{\nu}_\mu$)

- Mig_WC_nue.dat (migration matrix – ν_e)
- Mig_WC_nuebar.dat (migration matrix – $\bar{\nu}_e$)
- XCC_Nuance.dat (charged current cross sections)
- XNC_Nuance.dat (neutral current cross sections)
- Null.dat (auxiliary file)

The neutrino beam is produced at CERN and directed towards a a megaton Water Cerenkov detector at Fréjus. The neutrinos originate from the decays of accelerated isotopes ^{18}Ne (ν_e) and ^{6}He ($\bar{\nu}_e$). The acceleration factor is $\gamma = 100$ for both types of isotopes and $2.2 \cdot 10^{18}$ ^{18}Ne decays per year and $5.8 \cdot 10^{18}$ ^{6}He decays per year are assumed. The CERN-Fréjus baseline is L = 130 km, the fiducial mass of the detector is $m_{\mathrm{det}} = 500$ kt and 4 years ν-running and 4 years $\bar{\nu}$-running are assumed. The energy resolution is introduced manually by four migration matrices (for $\nu_e, \bar{\nu}_e, \nu_\mu, \bar{\nu}_\mu$) that describe the energy smearing due to Fermi Motion. The following rules are defined within BB_100.glb:

Disappearance – ^{18}Ne stored			σ_{norm}	σ_{cal}
Signal	$0.707 \otimes (\nu_e \to \nu_e)_{\mathrm{CC}}$		0.02	10^{-4}
Background	$4.3 \cdot 10^{-5} \otimes (\nu_e \to \nu_e)_{\mathrm{CC}}$		0.02	10^{-4}
Appearance – ^{18}Ne stored				
Signal	$1.0 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$	(energy dep. efficiency)	0.02	10^{-4}
Background	$1.0 \otimes (\nu_e \to \nu_x)_{\mathrm{NC}}$	(from external file)	0.02	10^{-4}
Atm. background		(from external file)	0.02	10^{-4}
Disappearance – ^{6}He stored				
Signal	$0.677 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\mathrm{CC}}$		0.02	10^{-4}
Background	$4.3 \cdot 10^{-5} \otimes (\bar{\nu}_e \to \nu_e)_{\mathrm{CC}}$		0.02	10^{-4}
Appearance – ^{6}He stored				
Signal	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{CC}}$	(energy dep. efficiency)	0.02	10^{-4}
Background	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_{\mathrm{NC}}$	(from external file)	0.02	10^{-4}
Atm. background		(from external file)	0.02	10^{-4}

Higher Gamma Scenario – BB_350.glb

A $\gamma = 350$ medium gamma β-beam scenario involving a megaton Water Cerenkov detector can be simulated with the file BB_350.glb. This file tries to approximate as closely as possible the scenario Setup III from [14]. This reference should be cited if the file BB_350.glb is used for a scientific publication or a talk. For calculations that involve BB_350.glb, the following additional files are required:

- `BB350flux.dat` (β-beam neutrino flux – $\gamma = 350$)
- `NeEffMig350.dat` (migration matrix – ν_μ)
- `NeBckgRej350.dat` (migration matrix – background)
- `NeDisEff350.dat` (migration matrix – ν_e)
- `HeEffMig350.dat` (migration matrix – $\bar{\nu}_\mu$)
- `HeBckgRej350.dat` (migration matrix – background)
- `HeDisEff350.dat` (migration matrix – $\bar{\nu}_e$)
- `XCC.dat` (charged current cross sections)
- `XNC.dat` (neutral current cross sections)

The neutrinos originate from the decays of accelerated isotopes ^{18}Ne (ν_e) and ^{6}He ($\bar{\nu}_e$). The acceleration factor is $\gamma = 350$ for both types of isotopes and $2.2 \cdot 10^{18}$ ^{18}Ne decays per year and $5.8 \cdot 10^{18}$ ^{6}He decays per year are assumed. The ion acceleration would require either a refurbished SPS (with superconducting magnets) or a more powerful accelerator, such as the Tevatron or LHC. The baseline is L = 730 km, the fiducial mass of the detector is $m_{\mathrm{det}} = 500\,\mathrm{kt}$ and 4 years ν-running and 4 years $\bar{\nu}$-running are assumed. The energy resolution is introduced manually by six migration matrices (for $\nu_e, \bar{\nu}_e, \nu_\mu, \bar{\nu}_\mu$, and the background from NC events for ^{18}Ne and ^{6}He) that describes energy smearing. These migration matrices also already include energy dependent efficiencies and background rejection factors. They are taken from the appendix of [14]. The following rules are defined within `BB_350.glb`:

Disappearance – ^{18}Ne stored			σ_{norm}	σ_{cal}
Signal	$1.0 \otimes (\nu_e \to \nu_e)_{\mathrm{CC}}$	(migration matrix)	0.025	10^{-4}
Background	neglected	(systematic uncertainty dominates)	–	–
Appearance – ^{18}Ne stored				
Signal	$1.0 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$	(migration matrix)	0.025	10^{-4}
Background	$1.0 \otimes (\nu_e \to \nu_x)_{\mathrm{NC}}$	(migration matrix)	0.05	10^{-4}
Disappearance – ^{6}He stored				
Signal	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\mathrm{CC}}$	(migration matrix)	0.025	10^{-4}
Background	neglected	(systematic uncertainty dominates)	–	–
Appearance – ^{6}He stored				
Signal	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{CC}}$	(migration matrix)	0.025	10^{-4}
Background	$1.0 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_{\mathrm{NC}}$	(migration matrix)	0.05	10^{-4}

Variable Beta Beam (Water Cerenkov) – BBvar_WC.glb

A variable β-beam scenario involving a megaton Water Cerenkov detector can be simulated with the file BBvar_WC.glb. The basic file was used within [15]. This reference should be cited if the file BBvar_WC.glb is used for a scientific publication or a talk. For calculations that involve BBvar_WC.glb, the following additional files are required:

- BckgMig_var.dat (migration matrix – background)
- XCC.dat (charged current cross sections)
- XNC.dat (neutral current cross sections)
- XQE.dat (quasi elastic cross sections)

and the values of the following AEDL-Variables have to be set:

- gammafactor (acceleration factor γ)
- EXP_FACTOR (parameter of ion decay scaling)
- baselinefactor (baseline parameter L/γ [km])

The neutrinos originate from the decays of accelerated isotopes ^{18}Ne (ν_e) and ^{6}He ($\bar{\nu}_e$). The acceleration factor is $\gamma =$gammafactor for both types of isotopes and $(100/\gamma)^\alpha \cdot 2.2 \cdot 10^{18}$ ^{18}Ne decays per year and $(60/\gamma)^\alpha \cdot 5.8 \cdot 10^{18}$ ^{6}He decays per year are assumed where $\alpha =$ EXP_FACTOR is a parameter that describes ion decay scaling. As default value EXP_FACTOR=0 should be chosen. Technically EXP_FACTOR can be chosen completely free, but the value should not deviate far from zero to stay meaningful. See [15] for a detailed discussion of this parameter. The γ value has to be chosen above 50. BBvar_WC.glb is optimized for $\gamma \lesssim 350$ but should be satisfactory up to $\gamma = 500$. The baseline is L = baselinefactor$\cdot\gamma$ km, the fiducial mass of the detector is $\mathrm{m_{det}} = 500\,\mathrm{kt}$ and 4 years ν-running and 4 years $\bar{\nu}$-running are assumed. The AEDL-Variable baselinefactor must be chosen such that the baseline lies in the interval $1\,\mathrm{km} \lesssim \mathrm{L} \lesssim 2 \cdot \mathrm{R_{EARTH}}$. The appearance measurement involves the total rates data from all CC events and the spectral data from the QE sample with a free normalization at an energy resolution of $\sigma_\mathrm{e} = 0.085\mathrm{GeV}$ due to the Fermi Motion identical to the treatment of systematics within the T2K and T2HK files. Note, that here also the systematics function for the appearance sample is chiSpectrumTilt in case of systematics on *and* off to avoid double counting of the QE events for systematics switched off. The following rules are defined within BBvar_WC.glb:

Disappearance – ^{18}Ne stored		σ_norm	σ_cal
Signal	$0.55 \otimes (\nu_e \to \nu_e)_\mathrm{QE}$	0.025	10^{-4}
Background	$0.003 \otimes (\nu_e \to \nu_x)_\mathrm{NC}$	0.05	10^{-4}

Appearance – ^{18}Ne stored – Spectrum		σ_{norm}	σ_{cal}
Signal	$0.55 \otimes (\nu_e \to \nu_\mu)_{\mathrm{QE}}$	10.0	10^{-4}
Background	$0.003 \otimes (\nu_e \to \nu_x)_{\mathrm{NC}}$	0.05	10^{-4}
Appearance – ^{18}Ne stored – Total Rates			
Signal	$0.55 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$	0.025	10^{-4}
Background	$0.003 \otimes (\nu_e \to \nu_x)_{\mathrm{NC}}$	0.05	10^{-4}
Disappearance – ^{6}He stored			
Signal	$0.75 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_{\mathrm{QE}}$	0.025	10^{-4}
Background	$0.0025 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_{\mathrm{NC}}$	0.05	10^{-4}
Appearance – ^{6}He stored – Spectrum			
Signal	$0.75 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{QE}}$	10.0	10^{-4}
Background	$0.0025 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_{\mathrm{NC}}$	0.05	10^{-4}
Appearance – ^{6}He stored – Total Rates			
Signal	$0.75 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{CC}}$	0.025	10^{-4}
Background	$0.0025 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_{\mathrm{NC}}$	0.05	10^{-4}

Variable Beta Beam (TASD) – `BBvar_TASD.glb`

A variable β-beam scenario involving a NOνA-like TASD detector can be simulated with the file `BBvar_TASD.glb`. The basic file was used within [15]. This reference should be cited if the file `BBvar_TASD.glb` is used for a scientific publication or a talk. For calculations that involve `BBvar_TASD.glb`, the following additional files are required:

- `XCC.dat` (charged current cross sections)
- `XNC.dat` (neutral current cross sections)

and the values of the following AEDL-Variables have to be set:

- `gammafactor` (acceleration factor γ)
- `EXP_FACTOR` (parameter of ion decay scaling)
- `baselinefactor` (baseline parameter L/γ [km])

The neutrinos originate from the decays of accelerated isotopes ^{18}Ne (ν_e) and ^{6}He ($\bar{\nu}_e$). The acceleration factor is γ =`gammafactor` for both types of isotopes and $(100/\gamma)^\alpha \cdot 2.2 \cdot 10^{18}$ ^{18}Ne decays per year and $(60/\gamma)^\alpha \cdot 5.8 \cdot 10^{18}$ ^{6}He decays per year are assumed where α = `EXP_FACTOR` is a parameter that describes ion decay scaling. As default value

EXP_FACTOR=0 should be chosen. Technically EXP_FACTOR can be chosen completely free, but the value should not deviate far from zero to stay meaningful. See [15] for a detailed discussion of this parameter. The γ value has to be chosen above 80. The baseline is $\mathrm{L} =$ baselinefactor$\cdot\gamma$ km, the fiducial mass of the detector is $\mathrm{m_{det}} = 50\,\mathrm{kt}$ and 4 years ν-running and 4 years $\bar{\nu}$-running are assumed. The AEDL-Variable baselinefactor must be chosen such that the baseline lies in the interval $1\,\mathrm{km} \lesssim \mathrm{L} \lesssim 2 \cdot \mathrm{R_{EARTH}}$. The energy resolution is $\sigma_\mathrm{e} = 6\% \cdot \sqrt{\mathrm{E}}$ for electrons and $\sigma_\mathrm{e} = 3\% \cdot \sqrt{\mathrm{E}}$ for muons. The following rules are defined within BBvar_TASD.glb:

Disappearance – ^{18}Ne stored		σ_norm	σ_cal
Signal	$0.2 \otimes (\nu_e \to \nu_e)_\mathrm{CC}$	0.025	10^{-4}
Background	$0.001 \otimes (\nu_e \to \nu_x)_\mathrm{NC}$	0.05	10^{-4}
Appearance – ^{18}Ne stored			
Signal	$0.8 \otimes (\nu_e \to \nu_\mu)_\mathrm{CC}$	0.025	10^{-4}
Background	$0.001 \otimes (\nu_e \to \nu_x)_\mathrm{NC}$	0.05	10^{-4}
Disappearance – ^{6}He stored			
Signal	$0.2 \otimes (\bar{\nu}_e \to \bar{\nu}_e)_\mathrm{CC}$	0.025	10^{-4}
Background	$0.001 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_\mathrm{NC}$	0.05	10^{-4}
Appearance – ^{6}He stored			
Signal	$0.8 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_\mathrm{CC}$	0.025	10^{-4}
Background	$0.001 \otimes (\bar{\nu}_e \to \bar{\nu}_x)_\mathrm{NC}$	0.05	10^{-4}

B.4 Neutrino Factory Experiments

Experiment	File	Runtime	E_μ	Baseline	Det.	Mass
Standard	NFstandard.glb	4 yr ν 4 yr $\bar{\nu}$	50 GeV	3000 km	MID	50 kt
Variable E_μ	NFvar.glb	4 yr ν 4 yr $\bar{\nu}$	variable	variable	MID	50 kt
Gold+Silver	NF_GoldSilver.glb	4 yr ν 4 yr $\bar{\nu}$	variable	variable	MID ECC	50 kt 5 kt
Hybrid det.	NF_hR_lT.glb	4 yr ν 4 yr $\bar{\nu}$	variable	variable	Hybrid	50 kt

Standard Neutrino Factory – NFstandard.glb

A standard neutrino factory scenario can be simulated with the file NFstandard.glb. The basic version was used within [6] (NuFact-II scenario, but some changes in normalization errors). This reference should be cited if the file `NFstandard.glb` is used for a scientific publication or a talk. For calculations that involve `NFstandard.glb`, the following additional files are required:

- `XCC.dat` (charged current cross sections)
- `XNC.dat` (neutral current cross sections)

The neutrino beam is produced by the decay of muons stored in a storage ring at a parent energy of $E_\mu = 50\,\mathrm{GeV}$. A number of $1.06 \cdot 10^{21}$ useful muon decays per year is assumed in each polarity (corresponding to $5.3 \cdot 10^{20}$ useful muon decays per year and polarity for simultaneous operation with both polarities) and 4 years ν-running and 4 years $\bar{\nu}$-running is assumed. The fiducial mass of the MID detector is taken to be $m_{\mathrm{det}} = 50\,\mathrm{kt}$ at a baseline of $L = 3000$ km. The energy resolution is $\sigma_e = 15\% \cdot E$. The following rules are defined within `NFstandard.glb`:

Disappearance – μ^+-stored			σ_{norm}	σ_{cal}
Signal	$0.35 \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$1.0 \cdot 10^{-5} \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\mathrm{NC}}$		0.2	10^{-4}
Appearance – μ^+-stored				
Signal	$0.45 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$5.0 \cdot 10^{-6} \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\mathrm{NC}}$	$5.0 \cdot 10^{-6} \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\mathrm{CC}}$	0.2	10^{-4}
Disappearance – μ^--stored				
Signal	$0.45 \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$1.0 \cdot 10^{-5} \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$		0.2	10^{-4}
Appearance – μ^--stored				
Signal	$0.35 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$5.0 \cdot 10^{-6} \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$	$5.0 \cdot 10^{-6} \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$	0.2	10^{-4}

Variable E_μ Neutrino Factory – `NFvar.glb`

A variable neutrino factory scenario can be simulated with the file `NFvar.glb`. The basic version was used within [16] and follows the neutrino factory scenarios from [6]. These references should be cited if the file `NFvar.glb` is used for a scientific publication or a talk. For calculations that involve `NFvar.glb`, the following additional files are required:

- `XCC.dat` (charged current cross sections)
- `XNC.dat` (neutral current cross sections)

and the values of the following AEDL-Variables have to be set:

- `emax` (parent energy of the stored muons [km])
- `BASELINE` (experiment baseline [GeV])

The neutrino beam is produced by the decay of muons stored in a storage ring at a parent energy of $E_\mu =$ `emax`. The parent energy of the muons can be appropriately set in the range $10\,\mathrm{GeV} \lesssim E_\mu \lesssim 80\,\mathrm{GeV}$. The baseline of the scenario is L = `BASELINE` and has to be chosen within the interval $1\,\mathrm{km} \lesssim L \lesssim 2 \cdot R_{\mathrm{EARTH}}$. Besides these settings the other attributes of `NFvar.glb` are similar to the ones from `NFstandard.glb`. Only the treatment of the disappearance channels is different. In the disappearance channels of `NFstandard.glb`, the charge identification is used to reduce the background from the appearance neutrinos. In `NFvar.glb`, however, a threshold similar to the MINOS experiment [27] is applied and the appearance and disappearance rates are assumed to be indistinguishable for the disappearance channel. The following rules are defined within `NFvar.glb`:

Disappearance – μ^+-stored			σ_{norm}	σ_{cal}
Signal	$0.9 \otimes (\bar\nu_\mu \to \bar\nu_\mu)_{\mathrm{CC}}$	$0.9 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$	0.025	10^{-4}
Background	$1.0 \cdot 10^{-5} \otimes (\bar\nu_\mu \to \bar\nu_x)_{\mathrm{NC}}$		0.2	10^{-4}
Appearance – μ^+-stored				
Signal	$0.45 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$5.0 \cdot 10^{-6} \otimes (\bar\nu_\mu \to \bar\nu_x)_{\mathrm{NC}}$	$5.0 \cdot 10^{-6} \otimes (\bar\nu_\mu \to \bar\nu_\mu)_{\mathrm{CC}}$	0.2	10^{-4}
Disappearance – μ^--stored				
Signal	$0.9 \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$	$0.9 \otimes (\bar\nu_e \to \bar\nu_\mu)_{\mathrm{CC}}$	0.025	10^{-4}
Background	$1.0 \cdot 10^{-5} \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$		0.2	10^{-4}
Appearance – μ^--stored				
Signal	$0.35 \otimes (\bar\nu_e \to \bar\nu_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$5.0 \cdot 10^{-6} \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$	$5.0 \cdot 10^{-6} \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$	0.2	10^{-4}

Variable Feutrino Factory with Silver Channel – `NF_GoldSilver.glb`

A variable neutrino factory scenario that includes the golden and silver appearance channels can be simulated with the file `NF_GoldSilver.glb`. The basic version was used within [16] and the golden channel follows the neutrino factory scenarios from [6] and the description of the silver channel follows [17]. These references should be cited if the file `NFvar_GoldSilver.glb` is used for a scientific publication or a talk. For calculations that involve `NFvar_GoldSilver.glb`, the following additional files are required:

- `XCC.dat` (charged current cross sections)

- `XNC.dat` (neutral current cross sections)

and the values of the following AEDL-Variables have to be set:

- `emax` (parent energy of the stored muons [GeV])
- `BASELINE` (experiment baseline [km])

The beam and golden channel attributes are similar to `NFvar.glb`. The parent energy of the muons can be appropriately set in the range $10\,\mathrm{GeV} \lesssim \mathrm{E}_\mu \lesssim 80\,\mathrm{GeV}$ (technically it is possible to arrange `emax` for $\mathrm{E}_\mu = 5\,\mathrm{GeV}$). The baseline is set by the AEDL-Variable `BASELINE` and has to be chosen within the interval $1\,\mathrm{km} \lesssim \mathrm{L} \lesssim 2 \cdot \mathrm{R}_{\mathrm{EARTH}}$. For the silver channel, an additional ECC detector with a fiducial mass $\mathrm{m}_{\mathrm{ECC}} = 5\,\mathrm{kt}$ is assumed to be located at the same baseline as the MID detector. The energy resolution of the silver channel is set to $\sigma_\mathrm{e} = 20\% \cdot \mathrm{E}$. The following additional rule compared to `NFvar.glb` is introduced in `NF_GoldSilver.glb`:

τ-Appearance – μ^+-stored			σ_{norm}	σ_{cal}
Signal	$0.096 \otimes (\nu_e \to \nu_\tau)_{\mathrm{CC}}$		0.15	10^{-4}
Background	$3.1 \cdot 10^{-8} \otimes (\nu_e \to \nu_e)_{\mathrm{CC}}$	$2.0 \cdot 10^{-8} \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$	0.2	10^{-4}
	$3.7 \cdot 10^{-6} \otimes (\bar\nu_\mu \to \bar\nu_\mu)_{\mathrm{CC}}$	$1.0 \cdot 10^{-3} \otimes (\bar\nu_\mu \to \bar\nu_\tau)_{\mathrm{CC}}$	0.2	10^{-4}
	$7.0 \cdot 10^{-7} \otimes (\bar\nu_\mu \to \bar\nu_x)_{\mathrm{NC}}$	$7.0 \cdot 10^{-7} \otimes (\nu_e \to \nu_x)_{\mathrm{NC}}$	0.2	10^{-4}

High Resolution/Low Threshold Neutrino Factory – `NF_hR_lT.glb`

A variable neutrino factory hybrid detector scenario can be simulated with the file `NF_hR_lT.glb`. The basic version was used within [16] and follows the neutrino factory scenarios from [6]. These references should be cited if the file `NF_hR_lT.glb` is used for a scientific publication or a talk. For calculations that involve `NF_hR_lT.glb`, the same additional files as for `NFvar.glb` are required and the AEDL-Variables `emax` and `BASELINE` have to be set within the same constraints as for `NFvar.glb`. `NF_hR_lT.glb` implements a lower threshold ($\sim 1\,\mathrm{GeV}$) at a higher energy resolution $\sigma_\mathrm{e} = 15\% \cdot \mathrm{E} + 0.085\,\mathrm{MeV}$, where the constant term represents the effects from Fermi Motion. The background rejection is energy dependent according to $10^{-3}/\mathrm{E}^2$, and matches `NFvar.glb` at higher energies. The following rules are defined within `NF_hR_lT.glb`:

Disappearance – μ^+-stored			σ_{norm}	σ_{cal}
Signal	$0.9 \otimes (\bar\nu_\mu \to \bar\nu_\mu)_{\mathrm{CC}}$	$0.9 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$	0.025	10^{-4}
Background	$1.0 \otimes (\bar\nu_\mu \to \bar\nu_x)_{\mathrm{NC}}$		0.2	10^{-4}

Appearance – μ^+-stored				
Signal	$0.5 \otimes (\nu_e \to \nu_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$1.0 \otimes (\bar{\nu}_\mu \to \bar{\nu}_x)_{\mathrm{NC}}$	(energy dep. rejection)	0.2	10^{-4}
	$1.0 \otimes (\bar{\nu}_\mu \to \bar{\nu}_\mu)_{\mathrm{CC}}$	(energy dep. rejection)	0.2	10^{-4}
Disappearance – μ^--stored				
Signal	$0.9 \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$	$0.9 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{CC}}$	0.025	10^{-4}
Background	$1.0 \cdot 10^{-5} \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$		0.2	10^{-4}
Appearance – μ^--stored				
Signal	$0.5 \otimes (\bar{\nu}_e \to \bar{\nu}_\mu)_{\mathrm{CC}}$		0.025	10^{-4}
Background	$1.0 \otimes (\nu_\mu \to \nu_x)_{\mathrm{NC}}$	(energy dep. rejection)	0.2	10^{-4}
	$1.0 \otimes (\nu_\mu \to \nu_\mu)_{\mathrm{CC}}$	(energy dep. rejection)	0.2	10^{-4}

Appendix C

Flux normalization in GLoBES*

A common issue with GLoBES is confusion about the proper units for the input flux files for use in AEDL experiment descriptions. Source of the confusion is an undocumented factor 5.2 with which the fluxes are multiplied in GLoBES versions older than 3.0 (see below). In Version 3.0 and higher, the alternative flux environment `nuflux` is provided, which does not contain this factor. The following material is based on the old environment `flux`. For the use of `nuflux`, replace the factor 5.2 by unity.

Historical problem

One problem for the design of AEDL was initially that meaningful units for flux data strongly depend on the given type of experiment, but also on relatively arbitrary decisions. For accelerator beams based on pion decay, one frequently defines the beam luminosity in *protons on target (pot)* since this number has a one-to-one correspondence with the number of neutrinos produced. Another sensible unit could be *megawatts on target (MW)*, again this number is directly correlated with the number of neutrinos and moreover there is a unique relation to *pot* for a given accelerator. Of course, what matters is the integrated luminosity. In some cases the neutrino flux is given per 10^7 s. However, most experiments will run for several years, hence also this number has to enter somewhere. For neutrino factories the proper number is useful muon decays per unit time, and for reactor experiments it is the thermal power of the reactor *asf.* This demonstrates that it is reasonable to keep the flux definition flexible.

Implementation in GLoBES

In understanding how one still can figure out what the correct units are for each case, it is a good starting point to look at what GLoBES does with the input files. The cross section in the file is given as differential cross section divided by energy $x = \sigma/E$, and the flux file gives f. The differential number of events per GeV n as computed in GLoBES without oscillation and efficiencies is given by

$$\begin{aligned} n &= 5.2 \times x \times E \times f \times \\ &\quad \texttt{@norm} \times \texttt{@power} \times \texttt{@stored\_muons} \times \texttt{@time} \times \texttt{\$target\_mass} \times (\texttt{\$baseline})^{-2} \end{aligned}$$

Note that 5.2 is a undocumented fudge factor!

It is the sole responsibility of the author of the AEDL file and its supporting files, to ensure that the result makes sense. In principle, it is possible to divide, for example, `@time` by π and fix that by redefining the flux file by multiplying it with π. Modifications like that have happened in the past and still happen, and many of them are not properly commented.

Writing AEDL files

The task is to choose the value of `@norm` such that all the variables in the AEDL file have the proper units, *e.g.*, `@time` has proper unit years.

GLoBES assumes that the cross section x is given in $10^{-38}\,\mathrm{cm}^2$ and that all fluxes are given at a distance of 1 km. In addition, it assumes that the number of target nuclei τ (or protons or whatever applies to the given cross section) per unit target mass m_u (which usually is kt) are properly accounted for.[1] Assuming that in the flux file the data is given as number of neutrinos per unit area A and energy bin of width ΔE at a distance L from the source, one obtains

$$\texttt{@norm} = \frac{1}{5.2}\left(\frac{\mathrm{GeV}}{\Delta E}\right)\left(\frac{\mathrm{cm}^2}{A}\right)\left(\frac{L}{\mathrm{km}}\right)^2\left(\frac{\tau}{m_u}\right) \times 10^{-38} \times \left(\frac{\mathcal{L}_u}{\mathcal{L}}\right) \tag{C.1}$$

where $\mathcal{L}$ absorbs all factors in the flux file related to the integrated luminosity, and $\mathcal{L}_u$ is the unit chosen for it. The concept of integrated luminosity is nicely described in the GLoBES manual in Sec. 11.1. A little example illustrates this concept: The flux is given for $10^{21}\,\mathrm{pot\,y}^{-1}$ of 10 GeV protons, thus a good choice for the units $\mathcal{L}_u$ is $\mathrm{MW\,y}^{-1}$, which means that $\mathcal{L}/\mathcal{L}_u$ is given by (assuming a $10^7\,\mathrm{s}$ year)

$$\frac{\mathcal{L}}{\mathcal{L}_u} = \frac{10\,\mathrm{GeV}\,10^{21}\,\mathrm{pot\,y}}{10^7\,\mathrm{s}} \times (\mathrm{MW\,y}^{-1})^{-1} = 0.16\ldots \tag{C.2}$$

Moving from `flux` to `nuflux`

In order to change the older `flux` environment to the new `nuflux` (GLoBES 3.0 and higher), replace all user-defined fluxes, such as

```
flux(#user)<
  @flux_file = "user_file_1.dat"
```

[1]Note that the cross sections which are delivered with GLoBES always are per nucleon.

```
    @time = 2.0
    @power = 4.0
    @norm = 1e+8
>
```

by

```
FF=5.1989

nuflux(#user)<
    @flux_file = "user_file_1.dat"
    @time = 2.0
    @power = 4.0
    @norm = FF*1e+8
>
```

This replacement is not necessary for neutrino factory built-in fluxes, and built-in beta beam fluxes were not supported by earlier versions of GLoBES.

Appendix D

The GNU General Public License

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

    (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

    The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

    If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or

otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix E

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled ”History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled ”History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the ”History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled ”Acknowledgements” or ”Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled ”Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled ”Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled ”Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled ”Acknowledgements”, ”Dedications”, or ”History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Bibliography

[1] P. Huber, M. Lindner, and W. Winter, *Simulation of long-baseline neutrino oscillation experiments with GLoBES*, Comput. Phys. Commun. **167** (2005), 195, `hep-ph/0407333`.

[2] P. Huber, J. Kopp, M. Lindner, M. Rolinec, and W. Winter, *New features in the simulation of neutrino oscillation experiments with GLoBES 3.0*, `hep-ph/0701187`.

[3] A. M. Dziewonski and D. L. Anderson, *Preliminary reference earth model*, Phys. Earth Planet. Interiors **25** (1981), 297–356.

[4] F. D. Stacey, *Physics of the earth*, 2nd ed., Wiley, 1977.

[5] Y. Itow et al., *The jhf-kamioka neutrino project*, Nucl. Phys. Proc. Suppl. **111** (2001), 146–151, `hep-ex/0106019`.

[6] P. Huber, M. Lindner, and W. Winter, *Superbeams versus neutrino factories*, Nucl. Phys. **B645** (2002), 3–48, `hep-ph/0204352`.

[7] NOvA, I. Ambats et al., *Nova proposal to build a 30-kiloton off-axis detector to study neutrino oscillations in the fermilab numi beamline*, (2004), `hep-ex/0503053`.

[8] NOvA, T. Yang and S. Woijcicki, *Study of physics sensitivity of ν_μ disappearance in a totally active version of nova detector*, (2004), `Off-Axis-Note-SIM-30`.

[9] J. E. Campagne, M. Maltoni, M. Mezzetto, and T. Schwetz, *Physics potential of the cern-memphys neutrino oscillation project*, (2006), `hep-ph/0603172`.

[10] J. E. Campagne and A. Cazes, *The theta(13) and delta(cp) sensitivities of the spl-frejus project revisited*, Eur. Phys. J. **C45** (2006), 643–657, `hep-ex/0411062`.

[11] M. Mezzetto, *Physics potential of the spl super beam*, J. Phys. **G29** (2003), 1781–1784, `hep-ex/0302005`.

[12] P. Huber, M. Lindner, T. Schwetz, and W. Winter, *Reactor neutrino experiments compared to superbeams*, Nucl. Phys. **B665** (2003), 487–519, `hep-ph/0303232`.

[13] P. Huber, J. Kopp, M. Lindner, M. Rolinec, and W. Winter, *From double chooz to triple chooz: Neutrino physics at the chooz reactor complex*, JHEP **05** (2006), 072, `hep-ph/0601266`.

[14] J. Burguet-Castell, D. Casper, E. Couce, J. J. Gomez-Cadenas, and P. Hernandez, *Optimal beta-beam at the cern-sps*, Nucl. Phys. **B725** (2005), 306–326, `hep-ph/0503021`.

[15] P. Huber, M. Lindner, M. Rolinec, and W. Winter, *Physics and optimization of beta-beams: From low to very high gamma*, Phys. Rev. **D73** (2006), 053002, `hep-ph/0506237`.

[16] P. Huber, M. Lindner, M. Rolinec, and W. Winter, *Optimization of a neutrino factory oscillation experiment*, Phys. Rev. **D74** (2006), 073003, `hep-ph/0606119`.

[17] D. Autiero et al., *The synergy of the golden and silver channels at the neutrino factory*, Eur. Phys. J. **C33** (2004), 243–260, `hep-ph/0305185`.

[18] R. P. Brent, *Algorithms for minimization without derivatives*, Prentice-Hall, 1973.

[19] M. Blennow, T. Ohlsson, and W. Winter, *Damping signatures in future neutrino oscillation experiments*, JHEP **06** (2005), 049, `hep-ph/0502147`.

[20] T. Ohlsson and H. Snellman, *Neutrino oscillations with three flavors in matter: Applications to neutrinos traversing the earth*, Phys. Lett. **B474** (2000), 153–162, `hep-ph/9912295`, Erratum ibidem **B480**, 419(E) (2000).

[21] J. Kopp, *Efficient numerical diagonalization of hermitian 3x3 matrices*, (2006), `physics/0610206`.

[22] T. Ohlsson and W. Winter, *The role of matter density uncertainties in the analysis of future neutrino factory experiments*, Phys. Rev. **D68** (2003), 073007, `hep-ph/0307178`.

[23] K. Kiers, S. Nussinov, and N. Weiss, *Coherence effects in neutrino oscillations*, Phys. Rev. **D53** (1996), 537–547, `hep-ph/9506271`.

[24] C. Giunti, *Coherence and wave packets in neutrino oscillations*, Found. Phys. Lett. **17** (2004), 103–124, `hep-ph/0302026`.

[25] G. L. Fogli, E. Lisi, A. Marrone, D. Montanino, and A. Palazzo, *Getting the most from the statistical analysis of solar neutrino oscillations*, Phys. Rev. **D66** (2002), 053010, `hep-ph/0206162`.

[26] M. Ishitsuka, T. Kajita, H. Minakata, and H. Nunokawa, *Resolving neutrino mass hierarchy and cp degeneracy by two identical detectors with different baselines*, Phys. Rev. **D72** (2005), 033003, `hep-ph/0504026`.

[27] MINOS, E. Ables et al., *P-875: A long baseline neutrino oscillation experiment at fermilab*, FERMILAB-PROPOSAL-0875.

Appendix F

Indices

API functions

API constants & macros

AEDL reference

Index