How to Install GLoBES & A Simple Example for Using GLoBES

Yinyuan Huang, Aiyu Bai

1 Install GLoBES

Before you begin to install GLoBES, please make sure you have a Linux system, a C(C++) compiler, and GSL.

```
# In Linux system
sudo apt update
sudo apt install build-essential libgsl-dev
```

Option 1: Install from Website(not recommended)

You can download GLoBES from its website:

```
wget https://www.mpi-hd.mpg.de/personalhomes/globes/download/globes-3.2.18.tar.gz
```

After finishing downloading, you should unzip the file

```
tar -zxf globes-3.2.18.tar.gz
cd globes-3.2.18
```

and run configure

```
./configure
```

then you should compile and install

```
sudo make
sudo make install
sudo ldconfig
```

You can test the installation by compiling an example:

```
cd /path/to/globes-3.2.18/examples
make
./example1
```

If the compilation and running finishes with no errors, the installation is successful and you will get the output file test1.dat.

Option 2: Install from GitHub(recommended)

Another option to install GLoBES is to use GitHub. After struggling with the outdated compilation method for days (or even years), we have made GLoBES available on GitHub and adopted CMake as the build system to simplify the compilation process.
You can download the example program from GitHub:

```
git clone https://github.com/WeiMXi/globes-cmake.git
cd globes-cmake
mkdir build
cd build
cmake ..
make
cd examples
./example1
```

then you will get the output file test1.dat. If you want to use GLoBES as a third party library, you can download this program as a template.

2 A Simple Example

Example 1 in GLoBES demonstrates how to perform a two-parameter scan to explore the correlation between the mixing angle θ_{13} and the CP-violating phase δ_{CP}. It covers the full pipeline: experiment loading, parameter definition, rate simulation, and χ^2 calculation with systematic errors.

Program Structure and Functionality

Example 1 computes the surface of χ^2 in the $(\sin^2(2\theta_{13}), \delta_{\mathrm{CP}})$ plane. The following code blocks highlight key steps.

1. Library and experiment initialization

This code initialized the GLoBES library and loaded the experiment file `NFstandard.glb`:

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

2. Output preparation

A custom output routine opens the file `test1.dat` and writes a header. The file will store three columns: $\log_{10}(\sin^2\theta_{13})$, δ_{CP}, and χ^2.

```
InitOutput(MYFILE,"Format: Log(10,s22th13) deltacp chi^2 \n");
```

3. Definition of true and test parameters

This code defines the oscillation parameters with values representative of current knowledge (In there they are only for illustrative purposes and do not correspond to actual physical measurements):

```
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;
```

two parameter vectors are allocated and filled with these values:

```
glb_params true_values = glbAllocParams();
glb_params test_values = glbAllocParams();

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

The vector `true_values` is used to simulate the data from observation; `test_values` represents theoretical data and will be varied during the scan.

4. Data simulation

The true parameters are applied, and the expected event rates are computed:

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

5. Two-dimensional scan

This code is a nested loop that scans $\log_{10}(\sin^2 2\theta_{13})$ and δ_{CP}:

```
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){
        thetheta13=asin(sqrt(pow(10,x)))/2;
        glbSetOscParams(test_values,thetheta13,GLB_THETA_13);
        glbSetOscParams(test_values,y*M_PI/180.0,GLB_DELTA_CP);
        res=glbChiSys(test_values,GLB_ALL,GLB_ALL);
        AddToOutput(x,y,res);
    }
}
```

At each grid point, x is converted back to θ_{13}, the test parameter vector is updated, and the total χ^2 (including systematic errors) is calculated with `glbChiSys()`. The triplet (x, y, χ^2) is written in the output file `test1.dat`. After running example 1, this file will be generated in the current directory, and you can plot it to visualize the χ^2 landscape.
More detailed information can be obtained from GLoBES's user manual.

3 Scientific Plotting

Today, one can easily use AI to draw a scientific graph. However, there are still some points that should be noticed during plotting. A good science graph should contain the following:

- **Legend:** The symbols, line styles, and colors in the legend must match the data curves, and the description text should be concise and unambiguous. Place the legend preferentially in an empty area within the chart.
- **Tick marks:** All four axes (top, bottom, left, right) of the chart must have tick marks. All tick marks (major and minor) must point inward, toward the data area. Set appropriate axis limits so that the data curves occupy the main area of the graph.
- **Image format:** Save as vector graphics (PDF, SVG, EPS) whenever possible.

An example plot code for test1.dat using Python is shown below.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import matplotlib.ticker as ticker
from matplotlib.lines import Line2D

import scienceplots
plt.style.use(['science', 'no-latex'])

# ========== read data ==========
data = np.loadtxt('test1.dat', skiprows=1)
x_log = data[:, 0] # log10(sin^2 2theta13)
y = data[:, 1]     # delta_CP (degrees)
z = data[:, 2]     # chi^2

delta = z - np.min(z)

x = 10**x_log

x_min, x_max = 1e-4, 1e-2
y_min, y_max = 0, 200
n_grid = 500

x_grid = np.logspace(np.log10(x_min), np.log10(x_max), n_grid)
y_grid = np.linspace(y_min, y_max, n_grid)
X_grid, Y_grid = np.meshgrid(x_grid, y_grid, indexing='ij')

points = np.column_stack((x, y))
Z_grid = griddata(points, delta, (X_grid, Y_grid), method='cubic')

# ========== plot ==========
fig, ax = plt.subplots(figsize=(8, 6))

levels = [2.30, 6.18, 11.83]      # 1sigma, 2sigma, 3sigma (2 d.o.f.)
linestyles = ['-', '-.', '--']
labels = [r'$1\sigma$', r'$2\sigma$', r'$3\sigma$']

for level, ls, label in zip(levels, linestyles, labels):
    ax.contour(X_grid, Y_grid, Z_grid, levels=[level],
               colors='black', linewidths=2.5, linestyles=ls)
```

```
legend_elements = [
    Line2D([0], [0], color='black', lw=2.5, linestyle='-', label=r'$1\sigma$'),
    Line2D([0], [0], color='black', lw=2.5, linestyle='-.', label=r'$2\sigma$'),
    Line2D([0], [0], color='black', lw=2.5, linestyle='--', label=r'$3\sigma$')
]

idx_min = np.argmin(z)
best_x = x[idx_min]
best_y = y[idx_min]
ax.plot(best_x, best_y, 'r*', markersize=14,
        label=f'Best fit: ({best_x:.2e}, {best_y:.0f}$^\circ$)')

legend_elements.append(Line2D([0], [0], color='red', marker='*', linestyle='None',
                              markersize=12, label=f'Best fit'))
ax.legend(handles=legend_elements, loc='upper left', frameon=False)

ax.set_xscale('log')
ax.set_xlim(x_min, x_max)
ax.set_ylim(0, 200)

ax.xaxis.set_major_locator(ticker.LogLocator(base=10.0, numticks=5))
ax.xaxis.set_major_formatter(ticker.LogFormatterMathtext())

ax.tick_params(top=True, right=True, labeltop=False, labelright=False)

# ========= labels ==========
ax.set_xlabel(r'$\sin^2 2\theta_{13}$', fontsize=16)
ax.set_ylabel(r'$\delta_{\mathrm{CP}}$ [Degrees]', fontsize=16)

ax.grid(False)

# ========== save ==========
plt.tight_layout(pad=0.5)
plt.savefig('chi2_log_linestyles.pdf', dpi=300)
plt.savefig('chi2_log_linestyles.png', dpi=300)
plt.show()
```

The final contour generated by example 1 looks like figure 1.

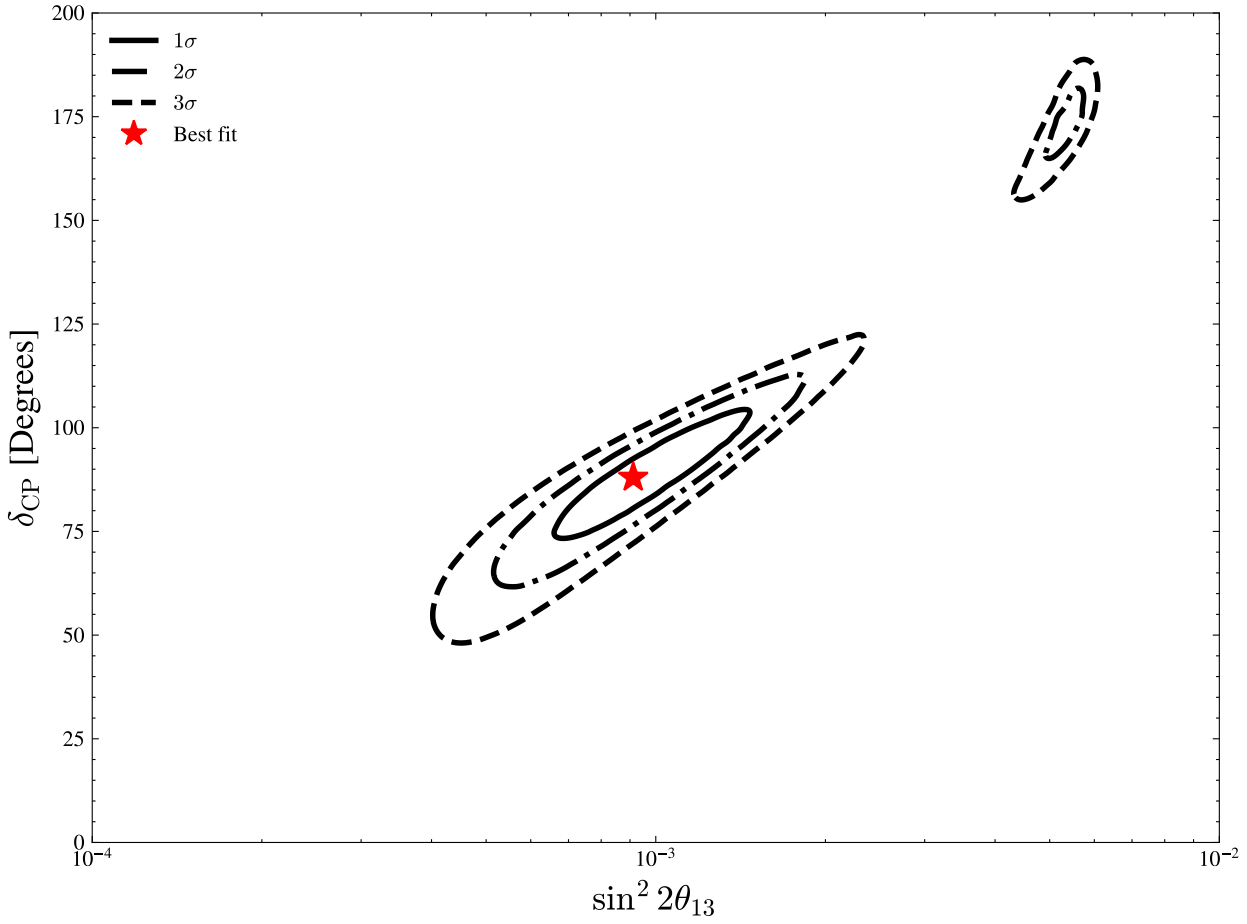

Figure 1: $\sin^2 2\theta_{13} - \delta_{\rm CP}$ contour generated by example 1.