The user manual: flavor Symmetry Embedded - GLoBES (FaSE-GLoBES)

Jian Tang[1*] and Tse-Chun Wang[1†]

[1] *School of Physics, Sun Yat-Sen University, Guangzhou 510275, China*

* tangjian5@mail.sysu.edu.cn

† wangzejun@mail.sysu.edu.cn

I. OVERVIEW FASE-GLOBES

Flavor **S**ymmetry **E**mbedded (**FaSE**) is a supplemental tool for **G**eneral **Lo**ng **B**aseline **E**xperiment **S**imulator (**GLoBES**) [1, 2] in order to analyse how a leptonic flavor symmetry model can be tested in neutrino oscillation experiments. **FaSE** is written in the `c/c++` language, and consist with two codes **FASE_GLoBES.c** and **model-input.c**. The user defines the model in to **model-input.c**, while **FASE_GLoBES.c** does not need to be touched.

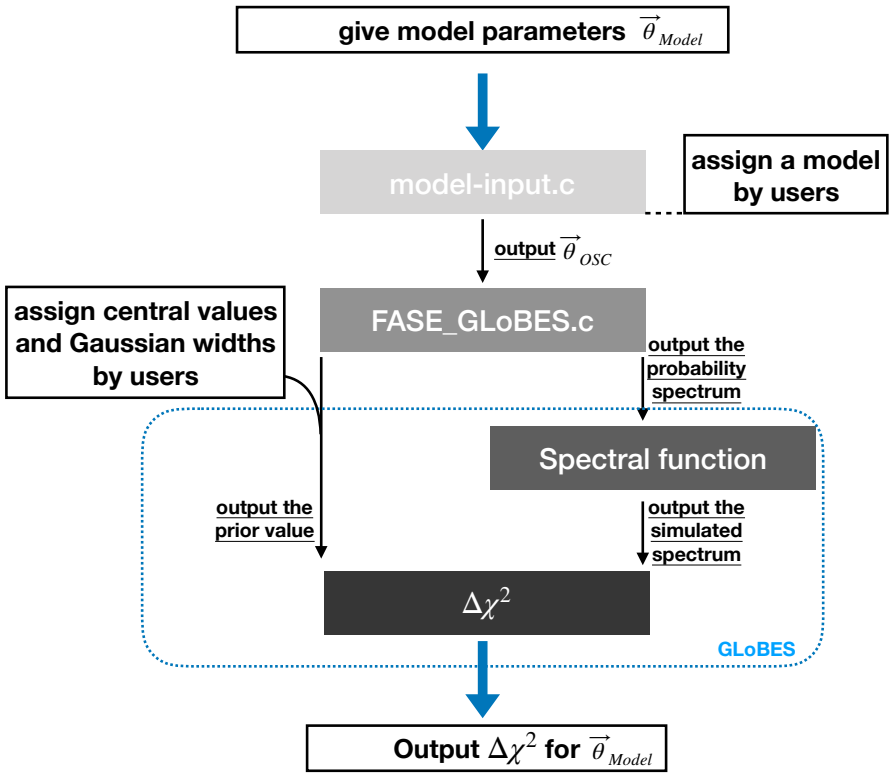

Figure 1: A scheme to correlate the model parameters with standard neutrino oscillation parameters. The error propagation is implemented in the simulation code up to the spectra analysis.

Combining **GLoBES** with **Fase** (we call it '**FaSE-GLoBES**'), the user can analyse flavor symmetry models with the simulated experimental configuration. The concept of

FaSE-GLoBES is shown in Fig. 1, in which three parts are shown: 1. **the parameter translation**, 2. **giving oscillation-parameter values**, and 3. **the χ^2-value calculation**. The idea behind this flow chart in Fig. 1 is that given a set of model parameters $\vec{\theta}_{Model}$, the corresponding values in standard oscillation parameters $\vec{\theta}_{OSC}$ are obtained by a translation function, which is assigned by the user in **model-input.c**. And then, through **FASE_GLoBES.c** these oscillation-parameter values are passed into **GLoBES** library to simulate the event spectra so that we can perform physics analysis in the newly-defined χ^2.

API functions in **FaSE** are listed:

1. `MODEL_init(`N_{para}`)`,
2. `FASE_glb_probability_matrix`,
3. `FASE_glb_set_oscillation_parameters`,
4. `FASE_glb_get_oscillation_parameters`,
5. `FASE_prior_OSC`,
6. `FASE_prior_model`.

The first one is to initialise **FaSE** with the number of input parameters N_{para}. The next three functions need to be included to replace the default **GLoBES** probability engine by the one that can read the output from **model-input.c**, as follows.

```
glbRegisterProbabilityEngine(6,
&FASE_glb_probability_matrix,
&FASE_glb_set_oscillation_parameters,
&FASE_glb_get_oscillation_parameters,
NULL);
```

This probability engine can work with oscillation or model parameters. It can be set by the user with the parameter `PARA`. If `PARA=STAN` (`PARA=MODEL`) the probability engine works with oscillation (model) parameters. The final two items on the API list are prior functions. Once the user gives the prior in oscillation (model) parameters, the user needs to call `FASE_prior_OSC` (`FASE_prior_model`) as follows.

```
glbRegisterPriorFunction(FASE_prior_OSC,NULL,NULL,NULL);
```

or

```
glbRegisterPriorFunction(FASE_prior_model,NULL,NULL,NULL);
```

We note that except for setting the probability engine and the prior function, the other parts in the main code should follow with the GLoBES manual. Many simulation packages are provided on internet. Some of them are also available in **GLoBES** website [3], while the working group in the DUNE experiment also releases their neutrino flux information and detector descriptions in a AEDL file, provided in [4]. We summarise AEDL files for some of most-discussed experiments in Table. I, including their source and reference.

Table I: Summary of available AEDL files for some of most-discussed experiments.

exp.	source	ref.
T2K	T2K.glb on [3]	[5]
NOvA	NOvA.glb on [3]	[6]
T2HK	sys-T2HK.glb on [3]	[7]
DUNE	[4]	[4]
MOMENT	on FaSE website	[8]

In the following, we give more details about using **FASE-GLoBES**. In Sec. I A, we will give the instruction to download and install **GLoBES** and **FASE**, and we will introduce how to compile and run the code. More complicated is setting the model, and the way to do this will be described in Sec. I B. Finally, we will discuss how to set up the prior in Sec. I C.

A. Download, compile and run

The user can download **FaSE** simply from `https://github.com/tcwphy/FASE_GLoBES.git` directly. Or, the user can also download it with

```
git clone https://github.com/tcwphy/FASE_GLoBES.git
```

As **FaSE** works based on **GLoBES**, the user should use the **GLoBES** `Makefile`, but include the binary file of **FaSE**. To do so, we include the script in the `Makefile`,

```
{makefile_execution}:  my_program.o FASE_GLoBES.o model-input.o
gcc my_program.o FASE_GLoBES.o model-input.o -o my_executable $(LDFLAGS)
```

`$(local_LDFLAGS)`

where `{compile_execusion}` is the makefile commend for the program `my_program.o` and the execution `my_executable` is the output. After giving this script in the **GLoBES** `Makefile`, the user needs to makefile the program `my_program` on the terminal.

```
makefile {makefile_execution}
```

And, the user can execute the program by typing the following commend on the terminal.

```
./my_executable
```

The user can also compile the program without the **GLoBES** `Makefile` with the following script.

```
gcc my_program.o FASE_GLoBES.o model-input.o -lglobes -LGLB_DIR/lib/ -o
my_executable
```

And, the execution is also in the same way: `./my_executable`.

Finally we need to initialize **FaSE** by giving N_{para} which is the number of model parameters. To do so, we need to include the following script in the main code.

`MODEL_init(`N_{para}`);`

We note that N_{para} should not be larger than the number of standard oscillation parameters, 6.

B. Model setting

The function `MtoS` can do the translator from model parameters $\vec{\theta}_{Model}$ to oscillation parameters $\vec{\theta}_{OSC}$. After the user gives the array $\vec{\theta}_{Model}$ to the function `MtoS`, the output is the corresponding oscillation parameter $\vec{\theta}_{OSC}$, of which components are θ_{12}, θ_{13}, θ_{23}, δ, Δm^2_{21}, and Δm^2_{31}. For the first four components, values are given in the unit of **rad**, while the other two are in $\mathbf{eV}^2$. These values will be passed in to **FaSE_GLoBES** to simulate the experimental spectra and compute the prior value.

To do the translation from $\vec{\theta}_{Model}$ to $\vec{\theta}_{OSC}$, the user can assign the relation between the oscillation and model parameter sets, or define the mass matrix in model parameters, which will be diagonalised by the function `STAN_OSC` to obtain the corresponding oscillation-

parameter values. In the way of directly giving the relation between oscillation and model parameter sets, the user needs to provide

$$\vec{\theta}_{Model} = \vec{f}(\vec{\theta}_{OSC}) \tag{1}$$

in the function `MtoS`.

The oscillation parameters can also be obtained in the way based on

$$U^{\dagger}\mathcal{M}\mathcal{M}^{\dagger}U = \mathbf{M}^2, \text{ where } \mathbf{M}^2_{\alpha\beta} = m^2_{\alpha}\delta_{\alpha\beta}, \tag{2}$$

where $\mathcal{M}$ ($\mathbf{M}$) is the neutrino mass matrix in the flavor (mass) state. The matrix $\mathcal{M}$ is given by user with model parameters $\vec{\theta}_{Model}$. The matrix U is the neutrino mixing matrix, and can be used for getting mixing angles. The difference between any two diagonal elements of $\mathbf{M}$ ($\mathbf{M}_{ii} - \mathbf{M}_{jj} = m^2_i - m^2_j$) is the mass-squared difference (Δm^2_{ij}). This diagnolisation Eq. (2) can be done with the function `STAN_OSC`, which outputs directly the vector $\vec{\theta}_{OSC}$. The user needs to call it in `MtoS` with

`STAN_OSC({`M`},{`$\vec{\theta}_{OSC}$`});`,

where `{`M`}` and `{`$\vec{\theta}_{OSC}$`}` are the input neutrino mass matrix and the output array of oscillation parameter values, respectively. More details about how to assign a model will be introduced in Sec. II A.

C. Prior setting

Given a set of values for model parameters, **FASE_GLoBES.c** will obtain the corresponding oscillation-parameter values from **model-input.c**, and will pass these values to simulate event spectra and to compute the prior value. Two gaussian prior functions are provided in **FaSE**: `FASE_prior_OSC` and `FASE_prior_model`. These two functions are for different purposes. If the user give the prior in oscillation (model) parameters, the user should register `FASE_prior_OSC` (`FASE_prior_model`) with the **GLoBES** function `glbRegisterPriorFunction`, as we introduced in the beginning of this section. The user also needs to assign the parameter `PARA=STAN` (`PARA=Model`), when the user prefers to give the prior in oscillation (model) parameters. The Gaussian prior is

$$\chi^2_{prior} = \sum_i \frac{(\theta_i - \theta^c_i)^2}{\sigma^2_i}, \tag{3}$$

where θ_i is one of parameters constrained by prior, θ_i^c (σ_i) is the central value (Gaussian width) of the prior for θ_i. We note that θ_i can be either model ($\vec{\theta}_{Model}$) or oscillation parameters ($\vec{\theta}_{OSC}$). The values of θ_i^c and σ_i need to be given by the user through three arrays: `Central_prior`, `UPPER_prior`, and `LOWER_prior`, in which there are six components. To treat asymmetry of width for upper ($\theta_i > \theta_i^c$) and lower ($\theta_i < \theta_i^c$) Gaussian widths, we give values in two arrays `UPPER_prior`, and `LOWER_prior`, respectively. If the user gives the prior in model parameters, the order of each component follows with the setup of input for the probability engine. If the user gives the prior in oscillation parameters, the six components in order are θ_{12}, θ_{13}, θ_{23}, δ, Δm^2_{21}, and Δm^2_{31}. The first four parameter are in **rad**, and the final two are in $\mathbf{eV}^2$.

Finally, some restrictions are imposed by the studied flavor symmetry model. We set up these restrictions in the function `model_restriction` in `model-input.c`. In the function `model_restriction`, the user needs to *return* 1 once the restriction is broken. For example, if the normal ordering is imposed, we give " `if (DMS31<0) { return 1;}` " in `model_restriction`, where `DMS31` is the variable for Δm^2_{31}. If there is no any restrictions, we simply *return* 0 in `model_restriction` as follows.

```
double model_restriction(double model []){ return 0;}
```

D. The definition of χ^2 function

The user can use **FaSE-GLoBES** to constrain model parameters. Suppose we have the measurement $\vec{x}$ and the likelihood function $L(\vec{\theta}) = P(\vec{x}|\vec{\theta})$ for a set of parameters $\vec{\theta} = (\theta_1, ..., \theta_N)$, where $P(\vec{x}|\vec{\theta})$ is the probability function for data $\vec{x}$ in favour of the hypothesis $\vec{\theta}$. The constraint of model parameters can be obtained with the statistis parameter $\chi^2 \equiv \ln L(\vec{\theta})$. The expression χ^2 is used as the default **GLoBES** setting. In more detail, the χ^2 function, following the Poisson distribution, is constructed based on a log-likelihood ratio,

$$\begin{aligned}\chi^2(\vec{\theta}^{hyp}, \xi_s, \xi_b) =& 2\sum_i \left(\eta_i(\vec{\theta}^{hyp}, \xi_s, \xi_b) - n_i + n_i \ln \frac{n_i}{\eta_i(\vec{\theta}^{hyp}, \xi_s, \xi_b)} \right) \\ &+ p(\xi_s, \sigma_s) + p(\xi_b, \sigma_b) + \chi^2_{prior}, \end{aligned} \tag{4}$$

where i runs over the number of bins, $\eta_i(\vec{\theta}, \xi_s, \xi_b)$ is the assumed event rate in the ith bin and E_i is the central value in this energy bin. The vector $\vec{\theta}$ consists of model or oscillation parameters. The parameters ξ_s and ξ_b are introduced to account for the systematic uncertainties in the normalisation for the signal (subscript $_s$) and background (subscript $_b$) components of the event rate, and are allowed to vary in the fit as nuisance parameters. For a given set of parameters $\vec{\theta}$, the event rate in the ith energy bin is calculated as

$$\eta_i(\vec{\theta}^{hyp}, \xi_s, \xi_b) = (1 + \xi_s) \times s_i + (1 + \xi_b) \times b_i, \tag{5}$$

where s_i and b_i are the expected number of signal and background events in ith energy bin, respectively. The nuisance parameters are constrained by the Gaussian prior $p(\xi, \sigma) = \xi^2/\sigma^2$ with corresponding uncertainties σ_s and σ_b for the signal and background, respectively. Finally, χ^2_{prior} is a set of Gaussian priors for hypothesis, and is expressed as Eq. (3). After doing all minimisations, the user obtains the χ^2 value for a specific hypothesis $\vec{\theta}^{hyp}$, $\chi^2(\vec{\theta}^{hyp})$.

Based on the χ^2 function Eq. (4), we can study how model parameters can be constrained and whether a flavor-symmetry neutrino model is excluded by simulated experiments. In the following we will demonstrate how it works.

Applications

The user of **FaSE-GLoBES** is able to study how model parameters can be constrained by the simulated experiments. To do so, the user needs to simulate the true event spectrum n_i with a set of model ($\vec{\theta}^{true}_{Model}$) or oscillation parameters ($\vec{\theta}^{true}_{OSC}$), *i.e.* set up $n_i(\vec{\theta}^{true}_{Model})$ or $n_i(\vec{\theta}^{true}_{OSC})$. The hypothesis $\vec{\theta}^{hyp}_{Model}$ predicts the tested event spectrum $\eta_i(\vec{\theta}^{hyp}_{Model}, \xi_s, \xi_b)$. With the default settings for χ^2 function as Eq. (3) in **FaSE-GLoBES**, the user computes the statistical quantity,

$$\chi^2(\vec{\theta}^{hyp}_{Model}), \text{ with } n_i(\vec{\theta}^{true}_{Model}) \text{ or } n_i(\vec{\theta}^{true}_{OSC}). \tag{6}$$

We note that the minimum of χ^2 in the whole parameter space ($\chi^2_{min.}$) may not be 0. Therefore, to get the precision of model parameters, the user should use the value $\Delta\chi^2(\vec{\theta}^{hyp}_{Model}) \equiv \chi^2(\vec{\theta}^{hyp}_{Model}) - \chi^2_{min.}$, instead of $\chi^2(\vec{\theta}^{hyp}_{Model})$ itself. By varying different hypotheses $\vec{\theta}^{hyp}_{Model}$, we will obtain the allowed region of model parameters with the statistical quantity $\Delta\chi^2(\vec{\theta}^{hyp}_{Model})$.

The user can also study how well a flavor symmetry model explains the computed data, or predict whether the simulated experiment can exclude this model or not. In other words, the user studies the minimum of χ^2 for the flavor symmetry model $\vec{\theta}_{Model}$ as a hypothesis, by assuming different true oscillation values, *i.e.* different $\vec{\theta}^{true}_{OSC}$. To do so, one can compute the same statistical quantity in Eq. (6), while the true spectrum is varied with different true values $\vec{\theta}^{true}_{OSC}$. All model parameters are allowed to be varied with the user-defined prior. Finally, the user might adopt Wilk's theorem to interpret results [9]. When we compare nested models, the $\Delta\chi^2$ test statistics is a random variable asymptotically distributed as a χ^2-distribution with the number of degrees of freedom, which is equal to the difference in the number of free model parameters.

In Secs. II D and II E, we will present examples to demonstrate how the user can do with **FaSE-GLoBES** to constrain the model parameter and to exclude a model by the simulated experiment configuration, respectively.

II. EXAMPLES

In this manual, we now take the tri-direct littlest seesaw (TDLS) [11–13] and a S_4 modular flavor symmetry model [14] as examples, and simulate the event spectra for Deep Underground Neutrino Experiment (DUNE) [15] and the MuOn-decay MEdium baseline NeuTrino beam experiment (MOMENT) [8] by GLoBES. In TDLS model, the atmospheric and solar flavon vacuum alignments are $\langle\phi_{\rm atm}\rangle \propto (1,\omega^2,\omega)^T$ and $\langle\phi_{\rm sol}\rangle \propto (1,x,x)^T$, where ω stands for a cube root of unity and the parameter x is real because of the imposed CP symmetry. Under this model, the light left-handed Majorana neutrino mass matrix in the flavor basis is given by

$$m_\nu = m_a \begin{pmatrix} 1 & \omega & \omega^2 \\ \omega & \omega^2 & 1 \\ \omega^2 & 1 & \omega \end{pmatrix} + e^{i\eta} m_s \begin{pmatrix} 1 & x & x \\ x & x^2 & x^2 \\ x & x^2 & x^2 \end{pmatrix} , \tag{7}$$

where x, η, m_a, and the ratio $r \equiv m_s/m_a$ are four parameters and will be constrained by experimental data. And based on Eq. (7), the relation between the model and oscillation parameters are predicted as Table. II. We note that from Eq. (7), $m_1 = 0$ and the normal mass ordering are imposed, and will need to be imposed in **FaSE-GLoBES**. Therefore, the restrictions in this model are $m_a > 0$ and $r > 0$.

Table II: A summary of the relation between oscillation parameters and TDLS model parameters [10]. Two requirements are imposed by TDLS: the smallest mass state $m_1 = 0$ and the normal mass ordering. The sign of $\sin\delta$ depends on the sign of $x\cos\psi$: "+" ("−") is for $x\cos\psi > 0$ (< 0).

model parameters	$x,\ \eta,\ r,\ m_a$
combinations of model parameters	$y = \frac{5x^2+2x+2}{2(x^2+x+1)}(m_a + e^{i\eta}m_s)$, $z = -\frac{\sqrt{5x^2+2x+2}}{2(x^2+x+1)}\left[(x+2)m_a - x(2x+1)e^{i\eta}m_s\right]$, $w = \frac{1}{2(x^2+x+1)}\left[(x+2)^2 m_a + x^2\,(2x+1)^2\,e^{i\eta}m_s\right]$, $\sin\psi = \frac{\Im(y^*z+wz^*)}{\lvert y^*z+wz^*\rvert}$, $\quad \cos\psi = \frac{\Re(y^*z+wz^*)}{\lvert y^*z+wz^*\rvert}$. $\sin 2\theta = \frac{2\lvert y^*z+wz^*\rvert}{\sqrt{(\lvert w\rvert^2-\lvert y\rvert^2)^2+4\lvert y^*z+wz^*\rvert^2}}$, $\cos 2\theta = \frac{\lvert w\rvert^2-\lvert y\rvert^2}{\sqrt{(\lvert w\rvert^2-\lvert y\rvert^2)^2+4\lvert y^*z+wz^*\rvert^2}}$.
oscillation parameters	$\Delta m^2_{21} = m_2^2 = \frac{1}{2}\left[\lvert y\rvert^2 + \lvert w\rvert^2 + 2\lvert z\rvert^2 - \frac{\lvert w\rvert^2-\lvert y\rvert^2}{\cos\theta}\right]$, $\Delta m^2_{31} = m_3^2 = \frac{1}{2}\left[\lvert y\rvert^2 + \lvert w\rvert^2 + 2\lvert z\rvert^2 + \frac{\lvert w\rvert^2-\lvert y\rvert^2}{\cos\theta}\right]$, $\sin^2\theta_{12} = 1 - \frac{3x^2}{3x^2+2(x^2+x+1)\cos^2\theta}$, $\sin^2\theta_{13} = \frac{2\left(x^2+x+1\right)\sin^2\theta}{5x^2+2x+2}$, $\sin^2\theta_{23} = \frac{1}{2} + \frac{x\sqrt{3(5x^2+2x+2)}\sin 2\theta\sin\psi}{2[3x^2+2(x^2+x+1)\cos^2\theta]}$, $\cos\delta = \frac{\cot 2\theta_{23}\left[3x^2-\left(4x^2+x+1\right)\cos^2\theta_{13}\right]}{\sqrt{3}\lvert x\rvert\sin\theta_{13}\sqrt{(5x^2+2x+2)\cos^2\theta_{13}-3x^2}}$, $\sin\delta = \pm\csc 2\theta_{23}\sqrt{1 + \frac{(x^2+x+1)^2\cot^2\theta_{13}\cos^2 2\theta_{23}}{3x^2[3x^2\tan^2\theta_{13}-2(x^2+x+1)]}}$.

In the S_4 modular model, the neutrino mass is predicted:

$$
\begin{aligned}
m_\nu = {} & (\mu_1\hat{c}_R^2 + \mu_2\hat{s}_R^{*2})\begin{pmatrix} 1 & -2\omega^2 & -2\omega \\ -2\omega^2 & 4\omega & 4 \\ -2\omega & 4 & 4\omega^2 \end{pmatrix} + (\mu_1\hat{s}_R^2 + \mu_2\hat{c}_R^{*2})\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix} \\
& + (\mu_1\hat{c}_R\hat{s}_R - \mu_2\hat{c}_R^*\hat{s}_R^*)\begin{pmatrix} 0 & -1 & 1 \\ -1 & 4\omega^2 & 2i\sqrt{3} \\ 1 & 2i\sqrt{3} & -4\omega \end{pmatrix},
\end{aligned} \tag{8}
$$

where $\hat{c}_R$ and $\hat{s}_R$ are $\cos\theta_R \times \mathrm{e}^{i\alpha_2}$ and $\sin\theta_R \times \mathrm{e}^{i\alpha_3}$, respectively. As ω is fixed at $-\frac{1}{2} + i\frac{\sqrt{3}}{2}$, this model has 5 model parameters: $\mu_1,\ \mu_2,\ \theta_R,\ \alpha_2,\ \alpha_3$.

In the following, all examples are mainly presented for TDLS, except for Sec. II E. However, we remind the user that though we use TDLS as an example, **FaSE-GLoBES** should

not be limited in any flavor model. We therefore in the **FaSE** distribution, show the warped flavor symmetry (WSF) [16], trimaximal mixing sum rule (TM1) [17] as examples, which will not be further introduced in this manual. We summarise our examples for **FaSE-GLoBES** and references for those theoretical models in Tabl III. And in the rest of this section, we demonstrate the **example 1** and **example 4**.

Table III: Summary of available AEDL files for some of most-discussed experiments.

example	model	application	reference
example1	TDLS	parameter constraint	[11–13]
example2	WSF	parameter constraint	[16]
example3	TM1 sum rule	model testing	[17]
example4	S_4 modular	model testing	[14]

A. Setup for a model

If the user prefers to assign the model with the mass matrix, the user will need to define the complex matrix `Mass_Matrix` with Eq. (7) in the function `MtoS`. The user can therefore assign the model as follows.

```
  int MtoS(double OSC_PARAMS[6], double M_para[])
{
double x=M_para[0];
double eta=M_para[1];
double r=M_para[2];
double ma=M_para[3];
double ms=ma*r;

  double complex Mass_Matrix[] = {ma+ms*(cos(eta) + I*sin(eta)),
ma*(cos(6.6666e-1*M_PI) + I*sin(6.6666e-1*M_PI))+x*ms*(cos(eta) +
I*sin(eta)), ma*(cos(6.6666e-1*M_PI) + I*sin(6.6666e-1*M_PI))*(cos(6.6666e-1*M_PI)
+ I*sin(6.6666e-1*M_PI))+x*ms*(cos(eta) + I*sin(eta)),
ma*(cos(6.6666e-1*M_PI) + I*sin(6.6666e-1*M_PI))+x*ms*(cos(eta) +
I*sin(eta)), ma*(cos(6.6666e-1*M_PI) + I*sin(6.6666e-1*M_PI))*(cos(6.6666e-1*M_PI)
```

```
+ I*sin(6.6666e-1*M_PI))+x*x*ms*(cos(eta) + I*sin(eta)),
ma+x*x*ms*(cos(eta) + I*sin(eta)), ma*(cos(6.6666e-1*M_PI) +
I*sin(6.6666e-1*M_PI))*(cos(6.6666e-1*M_PI) + I*sin(6.6666e-1*M_PI))+x*ms*(cos(eta)
+ I*sin(eta)), ma+x*x*ms*(cos(eta) + I*sin(eta)), ma*(cos(6.6666e-1*M_PI) +
I*sin(6.6666e-1*M_PI))+x*x*ms*(cos(eta) + I*sin(eta))};

  STAN_OSC(Mass_Matrix,OSC_PARAMS);

  return 0;
}
```

The user can also provide the relation between model and oscillation parameters Eq. (1), such as Table. II. The user, who prefers to do so, can assign a model as follows.

```
  double MtoS(double osc_para[6], double M_para[])
{
/*example:  tri-direct*/

double x=M_para[0];
double eta=M_para[1];
double r=M_para[2];
double ma=M_para[3];

osc_para[0]=TDth12(x,eta,r, ma);
osc_para[1]=TDth13(x,eta,r, ma);
osc_para[2]=TDth23(x,eta,r, ma);
osc_para[3]=TDdCP(x,eta,r, ma);
osc_para[4]=TDdm21(x,eta,r, ma);
osc_para[5]=TDdm31(x,eta,r, ma);

return 0;
}
```

The functions `TDth12`, `TDth13`, `TDth23`, `TDdCP`, `TDdm21`, and `TDdm31` are included for

obtaining the value of θ_{12}, θ_{13}, θ_{23}, δ, Δm^2_{21}, and Δm^2_{31} according to Table II, respectively.

The restrictions $m_a > 0$ and $r > 0$ need to be setup in `model_restriction` as follows.

```
double model_restriction(double model [])
{
double x=model[0];
double eta=model[1];
double r=model[2];
double ma=model[3];

if(ma<0) {return 1;}
if(r<0) {return 1;}

return 0;
}
```

B. Initialise the code

In the beginning of main code, we need to initialise **GLoBES** and **FaSE**, and to include the AEDL file for MOMENTas follows.

```
glbInit(argv[0]);
glbInitExperiment("exp/MOMENT_FIX_FLUX_150KM_addATM_NC.glb",&glb_experiment_list[0],
&glb_num_of_exps);
MODEL_init(4);
```

Then we register the probability engine.

```
glb_init_probability_engine();
glbRegisterProbabilityEngine(6,
&FASE_glb_probability_matrix,
&FASE_glb_set_oscillation_parameters,
&FASE_glb_get_oscillation_parameters,
NULL);
```

In the following we define the model-parameter values (x, η, r, and m_a), and set the true value in model parameters `true_values` to simulated the true event spectrum. Here we note that for setting up the true spectrum with model (oscillation) parameters, the user needs to make `PARA` equal to `MODEL` (`STAN`).

```
float x_true,eta_true,r_true,ma_true;
x_true=-3.65029; eta_true=1.13067*M_PI; r_true=0.511325;
ma_true=3.71199e-3;
glb_params true_values = glbAllocParams();
glbDefineParams(true_values,x_true,eta_true,r_true,ma_true,0,0);
glbSetDensityParams(true_values,1.0,GLB_ALL);
PARA=MODEL;
glbSetOscillationParameters(true_values);
glbSetRates();
```

C. Prior setup

To set up the prior, we need to register the prior function. As mentioned, when we use the prior in oscillation and model parameters, we register `FASE_prior_OSC` and `FASE_prior_model`, respectively. Here we present an example setting up priors in oscillation parameters according to NUFIT4.0 Table IV.

Table IV: The best fit and 3σ uncertainty, in the results of NuFit4.0 [18].

Parameter	$\theta_{12}/^\circ$	$\theta_{13}/^\circ$	$\theta_{23}/^\circ$	$\delta/^\circ$	$\Delta m^2_{21}/10^{-5}\mathrm{eV}^2$	$\Delta m^2_{31}/10^{-3}\mathrm{eV}^2$
best fit	33.82	8.61	49.6	215	7.39	2.525
3σ Range	$31.61 - 36.27$	$8.22 - 8.99$	$40.3 - 52.4$	$125 - 392$	$6.79 - 8.01$	$2.47 - 2.625$

We therefore register `FASE_prior_OSC` for the prior function as follows.

```
glbRegisterPriorFunction(FASE_prior_OSC,NULL,NULL,NULL);
```

And, then we give the value for the central values, and upper and lower Gaussian widths as follows.

```
UPPER_prior[0]=36.27*degree; LOWER_prior[0]=31.61*degree;
Central_prior[0]=33.82*degree;
UPPER_prior[1]=8.99*degree; LOWER_prior[1]=8.22*degree;
Central_prior[1]=8.61*degree;
UPPER_prior[2]=52.4*degree; LOWER_prior[2]=40.3*degree;
Central_prior[2]=49.6*degree;
UPPER_prior[3]=392*degree; LOWER_prior[3]=125*degree;
Central_prior[3]=215*degree;
UPPER_prior[4]=8.01e-5; LOWER_prior[4]=6.79e-5; Central_prior[4]=7.39e-5;
UPPER_prior[5]=2.625e-3; LOWER_prior[5]=2.427e-3; Central_prior[5]=2.525e-3;

int i;
for (i=0;i<6;i++) {UPPER_prior[i]=fabs(UPPER_prior[i]-Central_prior[i])/3;
LOWER_prior[i]=fabs(LOWER_prior[i]-Central_prior[i])/3;}
for (i=0; i<6; i++) glbSetOscParams(centers,0,i);
glbSetDensityParams(centers,1.0,GLB_ALL); glbCopyParams(centers,input_errors);
glbSetCentralValues(centers); glbSetInputErrors(input_errors);
```

The parameter `degree` is defined as $\pi/180$.

D. Constraint of model parameters

We show an example for how the model parameter can be constrained by MOMENT experiment, assuming the true model values $x = -3.65$, $\eta = 1.13\pi$, $r = 0.511$, and $m_a = 3.71$ meV. To get the constraint of x and η, we need to compute the minimum of χ^2 value as the following script in the main code.

```
glbSetProjection(free);
float res0=glbChiNP(true_values,NULL,GLB_ALL);
```

The **GLoBES** projector `free` allows all model parameters to vary, in order to find the value of `res0`, which is for χ^2_{min}. Then, we set two loops varying different values of x and η in $\vec{\theta}^{hyp}_{Model}$ to get $\Delta\chi^2$ values for different hypotheses. We note here there is degenerate

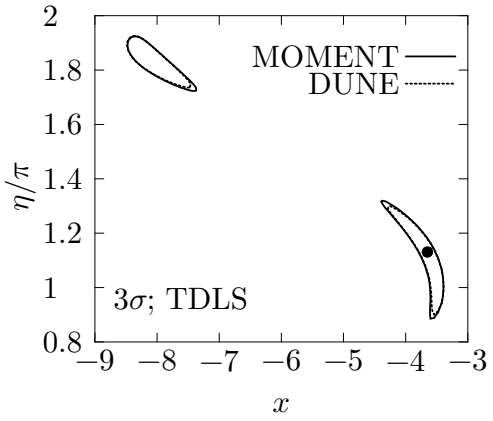

Figure 2: Example for using **FaSE-GLoBES** to obtain the constraints of model parameters for tri-direct littlest seesaw, with simulated DUNE and MOMENT data. The result is assumed the normal ordering. And, the dot denotes the model prediction with with NuFit4.0 results.

solutions where $x \sim -8$, $\eta \sim 1.8\pi$, $r \sim 0.1$, and $m_a \sim 3.8$ meV. To find this degenerate solutions we set the entry with $r = 0$.

```
float x,eta,r,ma,dx,deta;
float lower_x,upper_x,lower_eta,upper_eta;
FILE* File=fopen("data/constraint_x_eta.dat", "w");
lower_x=-9; upper_x=-3; lower_eta=0.8*M_PI; upper_eta=2*M_PI;
dx=(upper_x-lower_x)/100; deta=(upper_eta-lower_eta)/100;
glbSetProjection(projection);
for (x=lower_x;x<=upper_x;x=x+dx){
for (eta=lower_eta;eta<=upper_eta;eta=eta+deta){
glbSetOscParams(test_values,x,0); glbSetOscParams(test_values,eta,1);
glbSetOscParams(test_values,r_true,2);
float res=glbChiNP(test_values,NULL,GLB_ALL);
glbSetOscParams(test_values,0.,2);
float res2=glbChiNP(test_values,NULL,GLB_ALL);
if(res2<res) {res=res2;}
fprintf(File,"%f %f %f \n",x,eta/M_PI,res-res0);
} fprintf(File,"\n");}
```

The **GLoBES** projector `projection` fixes values of x and η, but allows the other parameters to vary. The output (`data/contraint_x_eta.dat`) is a set of $\Delta\chi^2(\vec{\theta}^{hyp})$ (the third column) values with the changed x (the first column), η (the second column). We use these data, and obtain the predicted 1σ, 2σ, and 3σ constraints of x and η by MOMENT shown in Fig. 2. The code for this example is provided as **x_eta.c** in **example1** of the **FaSE** distribution.

E. Model testing

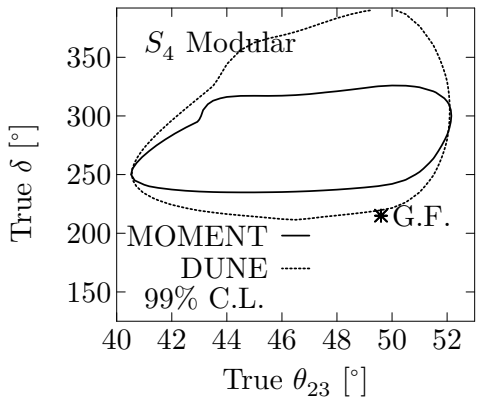

Figure 3: Example for using **FaSE-GLoBES** to obtain the 2-D exclusion contour at 99% C.L. on the plane of true values of θ_{23} and δ for a S_4 modular symmetry model with DUNE and MOMENT. The result is assumed the normal ordering. And, the dot (star) denotes the model prediction with NuFit4.0 results (best fit of NuFit4.0).

The user can also study how well a flavor symmetry model explains the computed data as predicting how the simulated experiment can exclude this model. As an example, we present testing a S_4 modular symmetry model, assuming various true values of θ_{23} and δ. The code for this example is provided as **th23_delta.c** in the **example4** of **FaSE** distribution. We compute the minimal χ^2 value for this S_4 modular model with all model parameters, varied with the prior in Eq. (3) according to NuFit4.0 results. The studied statistics function is exactly Eq. (4), but the true event rate n_i is predicted by a set of oscillation parameters, which will be varied in the code. And all four model parameter

can be varied with the prior given by Eq. (3). Except for θ_{23} and δ, which are varied between $40° < \theta_{23} < 52°$ and $125° < \delta < 392°$, we fix the other oscillation parameters at the current best fit result Table IV. Finally, in the code we adopt Wilk's theorem [9]: textttgsl_cdf_chisq_Qinv(gsl_cdf_chisq_Q(fabs(res),dof),1).

```
float th23,dCP,dth23,ddCP,lower_th23,upper_th23,lower_dCP,upper_dCP;
FILE* File=fopen("data/S4Modular_th23_dCP(DUNE)_v3.dat", "w");
lower_th23=40; upper_th23=52.5; lower_dCP=125; upper_dCP=395;
dth23=.5; ddCP=5.;

int dof=1;
glbSetProjection(free);
for (th23=lower_th23;th23<=upper_th23;th23=th23+dth23){
for (dCP=lower_dCP;dCP<=upper_dCP;dCP=dCP+ddCP){
glbSetOscParams(true_values,th23*degree,GLB_THETA_23);
glbSetOscParams(true_values,dCP*degree,GLB_DELTA_CP);
PARA=STAN;
glbSetOscillationParameters(true_values);
glbSetRates();
PARA=MODEL;
glbDefineParams(test_values,mu1,mu2,theta_R,alpha1,alpha2,0);
float res=glbChiNP(test_values,out,GLB_ALL);

for(double alpha2_test=25; alpha2_test<400; alpha2_test=alpha2_test+300){
glbSetOscParams(test_values,alpha2_test*degree,4);

for(double alpha1_test=60; alpha1_test<400; alpha1_test=alpha1_test+300){
glbSetOscParams(test_values,alpha1_test*degree,3);

glbSetOscParams(test_values,40*degree,2);
float res2=glbChiNP(test_values,out2,GLB_ALL);
```

```
if(res>res2) res=res2; glbCopyParams(out2,out);

glbSetOscParams(test_values,20*degree,2);
res2=glbChiNP(test_values,out2,GLB_ALL);
if(res>res2) res=res2; glbCopyParams(out2,out);
}}

 fprintf(File,"%f %f %f\n",th23,dCP,\
 gsl_cdf_chisq_Qinv(gsl_cdf_chisq_Q(fabs(res),dof),1));
 } fprintf(File,"\n");}
```

We note that hoops for variables `alpha2_text`, `alpha1_test`, and switch the entry value of θ_R between 20° and 40° is for finding the degenerate solutions, according to Ref. [14]. The output `data/S4Modular_th23_dCP_test.dat` contains true values of θ_{23} (the first column) and δ (the second column) with χ^2_{min} for TDLS (the third column). With these data, we then have the results shown in Fig. 3.

Bibliography

[1] P. Huber, M. Lindner, and W. Winter, Comput. Phys. Commun. **167**, 195 (2005), hep-ph/0407333.

[2] P. Huber, J. Kopp, M. Lindner, M. Rolinec, and W. Winter, Comput. Phys. Commun. **177**, 432 (2007), hep-ph/0701187.

[3] *GLoBES website, https://www.mpi-hd.mpg.de/personalhomes/globes/index.html*, URL `(https://www.mpi-hd.mpg.de/personalhomes/globes/index.html)`.

[4] T. Alion et al. (DUNE) (2016), 1606.09550.

[5] P. Huber, M. Lindner, and W. Winter, Nucl. Phys. **B645**, 3 (2002), hep-ph/0204352.

[6] I. Ambats et al. (NOvA) (2004), hep-ex/0503053.

[7] P. Huber, M. Mezzetto, and T. Schwetz (????).

[8] J. Cao et al., Phys. Rev. ST Accel. Beams **17**, 090101 (2014), 1401.8125.

[9] S. S. Wilks, Annals Math. Statist. **9**, 60 (1938).

[10] G.-J. Ding, S. F. King, and C.-C. Li, JHEP **12**, 003 (2018), 1807.07538.

[11] S. F. King, JHEP **07**, 137 (2013), 1304.6264.

[12] S. F. King, JHEP **02**, 085 (2016), 1512.07531.

[13] S. F. King and C. Luhn, JHEP **09**, 023 (2016), 1607.05276.

[14] I. de Medeiros Varzielas, S. F. King, and Y.-L. Zhou, Phys. Rev. D **101**, 055033 (2020), 1906.02208.

[15] R. Acciarri et al. (DUNE) (2015), 1512.06148.

[16] P. Chen, G.-J. Ding, A. D. Rojas, C. Vaquera-Araujo, and J. Valle, JHEP **01**, 007 (2016), 1509.06683.

[17] C. H. Albright and W. Rodejohann, Eur. Phys. J. C **62**, 599 (2009), 0812.0436.

[18] I. Esteban, M. C. Gonzalez-Garcia, A. Hernandez-Cabezudo, M. Maltoni, and T. Schwetz, JHEP **01**, 106 (2019), 1811.05487.