ν-SQuIDS: A toolbox for neutrino oscillation experiments.[☆]

Carlos Alberto Argüelles Delgado[a,b], Jordi Salvado Serrat[a,b]

[a]*Department of Physics, University of Wisconsin, Madison, WI 53706, USA*
[b]*Wisconsin IceCube Particle Astrophysics Center, Madison, WI 53703, USA*

Abstract

The Neutrino Simple Quantum Integro-Differential equation Solver (ν-SQuIDS) is a C++ code based on SQuIDS that propagates an ensemble of neutrinos through a given media, e.g. Sun, Earth, Vacuum, etc, while considering, in a consistent way, the effect of neutrino oscillations, with coherent matter interactions, and non coherent interactions. The code has been design to be accurate and flexible, while at the same time maintain excellent performance. Furthermore, the user can easily change the neutrino oscillation parameters, propagation medium, and easily incorporate new oscillation physics.

Keywords: Neutrino oscillation, phenomenology, collective neutrino behavior, numerical techniques

---

Contents

[☆]The code can be found in `http://icecube.wisc.edu/~carguelles/nusquids`

Email addresses: `arguelles@wisc.edu` (Carlos Alberto Argüelles Delgado), `salvadoserra@wisc.edu` (Jordi Salvado Serrat)

1. Introduction

During the last decades a plethora of evidence that neutrinos change flavor as they propagate macroscopical distances due to the nonalignment of their mass and flavor eigenstates has been accumulated from solar, atmospheric, accelerator, and reactor experiments (Mohapatra et al., 2005; de Gouvea et al., 2013). More over the

neutrino theory (Akhmedov, 1999; Balantekin and Haxton, 2013) oscillation with matter effect theory (Blennow and Smirnov, 2013)

globes (Huber et al., 2007)

Oscillation parameters are set to global sterile (Kopp et al., 2013)

lsnd and miniboone searches (Conrad et al., 2013) fundamental physics white paper (Hewett et al., 2012), sterile white paper (Abazajian et al., 2012)

fukugita (?) concha (?) euro white paper (Agarwalla et al., 2012)

The rest of the paper is organized as follows: in section 2 we review neutrino oscillation theory and establish notation; in section 3 we describe the code and its performance; in section 4 we exemplify the code and test it in benchmark scenarios. Finally, section 5 presents concluding remarks.

2. Theory

We set $c = \hbar = 1$.

We can represent the neutrino state using the density matrix formalism, e.g. in the weak-interaction flavor eigenstate basis $\{|\nu_\alpha\rangle\}$ it can be written as

$$\rho = \sum_\alpha \phi_\alpha |\nu_\alpha\rangle \langle\nu_\alpha| \tag{1}$$

where ϕ_α specifies the flavor content. Another important representation are the mass eigenstates $\{|\nu_i\rangle\}$, which are related to the former by

$$|\nu_\alpha\rangle = \sum_i U_{\alpha i} |\nu_i\rangle \tag{2}$$

where U is the unitary lepton mixing matrix. It is customary to parametrize the mixing matrix U with mixing angles, $\{\theta_{ij}\}$, and CP phases, $\{\delta_{ij}\}$; for example when considering the standard 3 flavor paradigm the following parametrization is often used

$$U = \begin{pmatrix} c_{12}c_{13} & s_{12}c_{13} & s_{13}e^{-i\delta_{13}} \\ -s_{12}c_{23} - c_{12}s_{23}s_{13}e^{i\delta_{13}} & c_{12}c_{23} - s_{12}s_{23}s_{13}e^{i\delta_{13}} & s_{23}c_{13} \\ s_{12}s_{23} - c_{12}c_{23}s_{13}e^{i\delta_{13}} & -c_{12}s_{23} - s_{12}c_{23}s_{13}e^{i\delta_{13}} & c_{23}c_{13} \end{pmatrix}, \tag{3}$$

where $c_{ij} = \cos\theta_{ij}$, $s_{ij} = \sin\theta_{ij}$. In the 3 flavor scenario we use the aforementioned parametrization (with values from (Gonzalez-Garcia et al., 2012)) and when more flavors are considered we used the prescription given in Argüelles Delgado and Salvado Serrat (2014). Furthermore, the neutrino ensemble evolution is described by the following Liouville equation

$$\frac{\partial\rho(E)}{\partial x} = -i[H(E,x), \rho(E)]\ . \tag{4}$$

In general we can always split the Hamiltonian, H, into a time dependent and independent parts. In particular, for neutrino oscillations the following splitting is convenient

$$H(E,x) = H_0(E) + H_1(E,x) \tag{5a}$$

$$H_0(E) = \frac{1}{2E}\mathrm{diag}(0, \Delta m^2_{21}, \Delta m^2_{31}, ..., \Delta m^2_{i1}, ..., \Delta m^2_{n1}) \tag{5b}$$

$$H_1(E,x) = \sqrt{2}G_F U^\dagger \mathrm{diag}(N_e(x) - N_n(x)/2, -N_n(x)/2, -N_n(x)/2, 0, ..., 0)U \tag{5c}$$

where n is the number of neutrino flavors, G_F is the Fermi constant, Δm^2_{i1} are the neutrino mass splittings, and, finally, $N_e(x)$ and $N_n(x)$ are the electron and nucleon number densities along the neutrino path. On writing these equations we have used the convention that the first three flavor eigenstates corresponds to ν_e, ν_μ, and ν_τ, while the rest are assume to be sterile neutrinos. Furthermore, H_0 arrises from the neutrino kinetic term, where as H_1 incorporates the matter potential, i.e. coherent forward scattering interactions (Mikheev and Smirnov, 1985, 1986; Wolfenstein, 1978). Given this splitting it is convenient to change to the so called *interaction picture* , generated by H_0, defined by the following transformation for a given operator O

$$O \to \bar{O}(x) = exp(-iH_0x)Oexp(iH_0x) \tag{6}$$

and then the evolution equation is

$$\frac{\partial \bar{\rho}(E)}{\partial x} = -i[\bar{H}_1(E,x), \bar{\rho}(E)] \ . \tag{7}$$

So far we have only incorporated neutrino oscillation and matter effects through coherent interactions, but we now wish to extend this formalism to incorporate non coherent interactions and collective neutrino behavior. This problem has been extensively discussed in the literature, in particular in (Duan et al., 2010; Strack and Burrows, 2005; Zhang and Burrows, 2013) and (Cirelli et al., 2005; Blennow et al., 2008; Argüelles Delgado and Kopp, 2012), for definiteness we follow the formalism given in (Gonzalez-Garcia et al., 2005). In what follows we will suppress the *bar* symbol and assume that all operators, unless specified, are on the interaction basis.

$$\frac{\partial \rho(E,x)}{\partial x} = -i[H_1(E,x), \bar{\rho}(E)] - \{\Gamma(E,x), \rho\} + F(\Omega, L; E, x) \tag{8a}$$

$$\frac{\partial l(E,x)}{\partial x} = -\gamma(E,x) l + G(\Omega; E, x) \tag{8b}$$

where we have introduced $\Omega = \{\rho(E)\}$ and $L = \{l(E)\} = \{e(E), \mu(E), \tau(E)\}$ the neutrino and lepton ensembles respectively. Γ incorporates the effect of attenuation due to non coherent interactions in neutrinos. The F term contains the interactions between the neutrino collective and the leptons, similarly the G term incorporates the effects of neutrinos into leptons. In most scenarios the e and μ leptons lose energy too fast to contribute significantly into the latter neutrino flux, thus we shall only consider the τ leptons since they have a very short decay time (Halzen and Saltzberg, 1998). Thus, we write the right hand side terms of Eq. (8) explicitly as follows

$$\Gamma(E,x) = \sum_\alpha \frac{\Pi_\alpha}{2\lambda^\alpha_{\text{total}}} \tag{9a}$$

$$\gamma(E,x) = -\frac{1}{\lambda^\tau_{\text{dec}}(E,x)} \tag{9b}$$

$$\begin{aligned} F(\Omega, L; E, x) &= \int_E^\infty d\tilde{E} \sum_\alpha \frac{1}{2} \left\{ \frac{\Pi_\alpha}{\lambda^\alpha_{\text{NC}}(\tilde{E},x)}, \rho(\tilde{E},x) \right\} \frac{\partial N_{\text{NC}}(\tilde{E},E)}{\partial E} \\ &+ \int_E^\infty d\tilde{E} \frac{1}{\lambda^\tau(\tilde{E},x)} \tau(\tilde{E},x) \frac{\partial N_{\text{dec}}(\tilde{E},E)}{\partial E} \Pi_\tau \\ &+ \text{Br}_{\text{lep}} \int_E^\infty d\tilde{E} \frac{1}{\lambda^\tau(\tilde{E},x)} \tilde{\tau}(\tilde{E},x) \frac{\partial \tilde{N}_{\text{dec}}(\tilde{E},E)}{\partial E} \Pi_\tau \end{aligned} \tag{9c}$$

$$G(\Omega; E, x) = \int_E^\infty d\tilde{E} \frac{1}{\lambda^\tau_{CC}(\tilde{E},x)} \text{Tr}\left[\Pi_\tau, \rho(\tilde{E},x)\right] \frac{\partial N_{\text{CC}}(\tilde{E},E)}{\partial E} \tag{9d}$$

where we have introduced the flavor projectors Π_α and the sums run over the active neutrino flavors. Furthermore, $\lambda_{\rm CC}$, $\lambda_{\rm NC}$, and $\lambda_{\rm total} = \lambda_{\rm CC} + \lambda_{\rm NC}$ are the charge, neutral and total neutrino interactions lengths respectively. More over, $\frac{\partial N}{\partial E}$ represent the outgoing neutrino (or τ lepton) spectral distribution for charge, neutral neutrino interactions and τ decay. Finally, λ^{τ}_{dec} is the τ decay length, which is assumed to be much smaller than the relevant neutrino oscillation and interaction scales.

3. Description of the code

ν-SQuIDS is a C++ code build using the SQuIDS *class* and *framework* (Argüelles Delgado and Salvado Serrat, 2014). It is designed to propagate neutrinos through media while taking into account flavor oscillations and non coherent interactions. The program can run in two modes: *single* or *multiple* energy.

In the *single* energy mode only a fix neutrino energy is considered and no collective neutrino effects are implemented. In this case, only equation (7) is relevant for the neutrino propagation, and the `nuSQUIDS` *class* implements `SQUIDS::H0` as in equation (5b) and `SQUIDS::HI` as given in equation (5c).

In the *multiple* energy mode a statistical ensemble of neutrinos is considered. The ensemble is described by means of a set of `SQUIDS::SU_vector` *objects* located at fixed *energy nodes*, which can be either linearly or logarithmically spaced over the energy region under consideration. Besides defining `SQUIDS::H0` and `SQUIDS::HI`, as in the *single* energy mode, the following functions are also defined: `SQUIDS::GammaRho` by equation (9a), `SQUIDS::InteractionsRho` as in equation (9c), and `SQUIDS::InteractionsScalar` as equation (9d). Furthermore, τ regeneration is included under the approximation that the neutrino interaction scale is much larger than the τ decay length by periodically reinjecting the τ flux decay products to the neutrino statistical ensemble.

Even though the `nuSQUIDS` class implements all the necessary differential equations we must still specify the neutrino propagation environment, its trajectory, the relevant cross sections, and - if τ regeneration is considered - the properties of τ decay. Thus, for example, when interactions are considered the `nuSQUIDS` instance will automatically construct appropriate `NeutrinoCrossSections` and `TauDecaySpectra` objects to evaluate cross sections and τ physics respectively. On the other hand, the user must explicitly specify the neutrino propagation medium and trajectory through relevant specialization of `Body` and `Body::Track` classes.

Finally, `nuSQUIDS` provides a set functions to evaluate the neutrino ensemble flavor and mass composition as well as the capability to store the calculation in an HDF5 (Folk et al., 1999) file for later study.

3.1. Body & Track

`Body` and `Body::Track` are virtual C++ classes which are used to represent the environment where the neutrino propages (`Body`) and the neutrino path inside it (`Body::Track`). Along this document we will sometime use the short hand `Track` for `Body::Track`, since its clear that the former is meaningless without its corresponding environment, and often we will refer to it as the *neutrino trajectory*. The `Body` class has two important *virtual* functions which are evaluated along the neutrino trajectory

- Density

  ```
  virtual double density(std::shared_ptr<Track>);
  ```

  Returns density at a give `Track` position in gr/cm^3.

- Electron fraction

  ```
  virtual double ye(std::shared_ptr<Track>);
  ```

  Returns the electron fraction at a give `Track` position.

Furthermore, the `Track` object has the following members and functions

- Basic members

  ```
  double x;
  double xini;
  double xned;
  ```

  `x` represent the current position along the neutrino path, while `xini` and `xend` are the initial and final position in natural units.

- Functions

  ```
  double GetX(void);
  double GetInitialX(void);
  double GetFinalX(void);
  ```

  These functions return `x`, `xini`, and `xend` respectively.

Since `Body` and `Track` are abstract classes they themselves do not perform any task, but rather their specializations specify the real neutrino propagation environment and how it relates to its trajectory. ν-SQuIDS implements the most common used environments and trajectory configurations, but the *user* is free (and encouraged) to create new *classes* in order to extend ν-SQuIDS applicability.

The `Body` classes specializations implemented in ν-SQuIDS are the following: `Vacuum`, `ConstantDensity`, `VariableDensity`, `Earth`, `EarthAtm`, and `Sun`.

3.1.1. Vacuum

- `Vacuum`

  ```
  Vacuum(void);
  ```

  Initializes a `Vacuum` environment.

- `Vacuum::Track`

  ```
  Vacuum::Track(double xini,double xend);
  ```

  Initialize the corresponding `Track` setting the initial (`xini`) and final (`xend`) neutrino position in eV^{-1}.

3.1.2. ConstantDensity

- `ConstantDensity`

  ```
  ConstantDensity(double rho, double ye);
  ```

  Initializes a `ConstantDensity` environment with constant density `rho`, in gr/cm^3, and electron fraction `ye`.

- `ConstantDensity::Track`

  ```
  ConstantDensity::Track(double xini,double xend);
  ```

  Initialize the corresponding `Track` setting the initial (`xini`) and final (`xend`) neutrino position in eV^{-1}.

3.1.3. VariableDensity

- `VariableDensity`

  ```
  VariableDensity(std::vector<double> x,std::vector<double> density,
  std::vector<double> ye);
  ```

  Initializes a `VariableDensity` environment given three equal size arrays specifying the density and electron fraction at given positions. An object will be created that interpolates using `gsl_spline` (Gough, 2009) along the `x` array to get the density and electron fraction as continuous functions.

- `VariableDensity::Track`

  ```
  VariableDensity::Track(double xini,double xend);
  ```

  Initialize the corresponding `Track` setting the initial (`xini`) and final (`xend`) neutrino position in eV^{-1}.

3.1.4. Earth

- `Earth`

  ```
  Earth(void);
  ```

  Initializes an `Earth` environment as defined by the PREM (Dziewonski and Anderson, 1981).

  ```
  Earth(string filepath);
  ```

Initializes an `Earth` environment as defined by a table given in the file specified by `filepath`. The table should have three columns: radius (where 0 is center and 1 is surface), density (gr/cm^3), and y_e (dimensionless). `gsl_spline` (Gough, 2009) is used to interpolate ρ and y_e as a function of radius to the earth center.

- `Earth::Track`

```
Earth::Track(double xini,double xend, double L);
```

Initialize the corresponding `Track` setting the initial (`xini`) and final (`xend`) neutrino position along a baseline `L`.

3.1.5. *EarthAtm*

- `EarthAtm`

```
EarthAtm(void);
```

Initializes an `EarthAtm` environment as defined by the PREM (Dziewonski and Anderson, 1981).

```
EarthAtm(string filepath);
```

Initializes an `EarthAtm` environment as defined by a table given in the file specified by `filepath`. The table should have three columns: radius (where 0 is center and 1 is surface), density (gr/cm^3), and ye (dimensionless). `gsl_spline` (Gough, 2009) is used to interpolate ρ and y_e as a function of radius to the earth center.

- `EarthAtm::Track`

```
EarthAtm::Track(double phi);
```

Initialize the corresponding `Track` by specifying the zenith angle in radians.

3.1.6. *Sun*

- `Sun`

```
Sun(void);
```

Initializes an `Sun` environment as defined by the *Standard Solar Model* (Bahcall et al., 2005).

- `Sun::Track`

```
Sun::Track(double xini, double xend);
```

Initialize the corresponding `Track` by the initial position in the sun `xini` and `xend` along the solar radius.

3.2. NeutrinoCrossSections

This object contains neutrino cross section information used when considering neutrino interactions. The given cross sections are a pQCD *deep inelastic* calculation using the CTEQ6 parton distributions functions on an isoscalar target and are valid for $E_\nu > O(10\text{GeV})$. Furthermore, the ν_e and ν_μ cross sections are the same, where as the ν_τ includes the τ final state mass suppression; the same holds true for the respective antineutrino cross sections. The cross sections are loaded from tables included in `nuSQUIDS/data/generate/`. The following quantities are given

- Total neutrino (antineutrino) cross section: $\sigma_\alpha^{CC/NC}(E_\nu)$
- Single differential neutrino (antineutrino) cross section: $\frac{d\sigma_\alpha^{CC}}{dE_\nu}(E_\nu, E_{lep})$, $\frac{d\sigma_\alpha^{NC}}{dE_\nu}(E_\nu, \tilde{E}_\nu)$

3.2.1. Constructors and Initializing Functions

- Standard void constructor.

```
NeutrinoCrossSections();
```

- Constructor and initializing function with memory reservation.

```
NeutrinoCrossSections(double Emin,double Emax, int Esize);
void Init(double,double,int);
```

This constructor and initialization functions calculate and store the cross sections on logarithmically spaced energy nodes between `Emin` and `Emax` with `Esize` divisions. For the total cross section `gsl_spline` (Gough, 2009) is used to interpolate in neutrino energy, where as for the differential cross section simple bilinear interpolation has been implemented.

3.2.2. Functions

The following functions assume that the τ and $\bar{\tau}$ have the same decay distribution.

- Total neutrino cross sections

```
double sigma_CC(int e1,int flv,int nt);
double sigma_NC(int e1,int flv,int nt);
```

`sigma_CC` (`sigma_NC`) returns the total deep inelastic neutrino charge (neutral) current cross section at an energy node `e1` with neutrino flavor specified by `flv` (`0 =` e, `1 =` μ, `2 =` τ), and `nt` toggles between neutrinos (`0`) and antineutrinos (`1`).

- Neutrino single differential cross sections

```
double dsde_CC(int e1,int e2,int flv, int nt);
double dsde_NC(int e1,int e2,int flv, int nt);
```

`dsde_CC` (`dsde_NC`) returns the neutrino single differential charge (neutral) current cross section between energy nodes `e1` and `e2`. Furthermore, the neutrino flavor specified by `flv` (`0 =` e, `1 =` μ, `2 =` τ), and `nt` toggles between neutrinos (`0`) and antineutrinos (`1`).

3.3. TauDecaySpectra

This object contains the information about the τ decay into leptons and hadrons. The formulas implemented in this class were taken from (Dutta et al., 2000) and are implemented in natural units. It is only used when *tau regeneration* is considered and it returns the following quantities on the energy nodes

$$\frac{dN_{dec}^{lep/had}(E_\tau, E_\nu)}{dE_\nu}, \frac{d\tilde{N}_{dec}^{lep/had}(E_\tau, E_\nu)}{dE_\nu} \tag{10}$$

i.e. the neutrino and antineutrino spectral distributions from τ lepton (hadron) decay mode.

3.3.1. Constructors and Initializing Functions

- Standard void constructor.

  ```
  TauDecaySpectra();
  ```

- Constructor and initializing function with memory reservation.

  ```
  TauDecaySpectra(double Emin,double Emax, int Esize);
  void Init(double,double,int);
  ```

  This constructor and initialization functions calculate and store the τ decay spectra on logarithmically spaced energy nodes between `Emin` and `Emax` with `Esize` divisions.

3.3.2. Functions

The following functions assume that the τ and $\bar{\tau}$ have the same decay distribution.

- (Anti)Neutrino spectra with respect to neutrino energy

  ```
  double dNdEnu_All(int e1,int e2);
  ```

  Returns neutrino decay spectra evaluated between energy nodes `e1` and `e2` when τ decays into leptons or hadrons.

  ```
  double dNdEnu_Lep(int e1,int e2);
  ```

  Returns neutrino decay spectra evaluated between energy nodes `e1` and `e2` when τ decays into leptons.

- (Anti)Neutrino spectra with respect to τ energy

  ```
  double dNdEle_All(int e1,int e2);
  ```

  Returns neutrino decay spectra evaluated between energy nodes `e1` and `e2` when τ decays into leptons or hadrons. with respect to the initial τ energy.

  ```
  double dNdEle_Lep(int e1,int e2);
  ```

  Returns neutrino decay spectra evaluated between energy nodes `e1` and `e2` when τ decays into leptons. with respect to the initial τ energy.

3.4. nuSQUIDS

This object is an specialization of the `SQUIDS` class (Argüelles Delgado and Salvado Serrat, 2014) that implements the differential equations as described in Sec. 2. In particular, it is used to specify the propagation `Body` and its associated `Track`. Moreover it uses the `NeutrinoCrossSections` and `TauDecaySpectra` in order to evaluate the neutrino cross sections and τ decay spectra; the latter is only used then τ regeneration is enabled. Furthermore, through the `SQUIDS::Set` function it enables the user to modify the neutrino oscillation parameters as well as the differential equation numerical precision. Finally, it also has the capability to create and read HDF5 files that store the program results and configuration.

3.4.1. Constructors and Initializing Functions

- Standard void constructor.

  ```
  nuSQUIDS();
  ```

- Single energy mode constructor.

  ```
  nuSQUIDS(int numneu, string NT);
  Init(int,string);
  ```

  This constructor and initialization function initializes `nuSQUIDS` in the *single energy mode*. `numneu` specifies the number of neutrino flavors which can go from 2 to 6, while `NT` can be set to `"neutrino"` or `"antineutrino"`.

- Multiple energy mode constructor.

  ```
  nuSQUIDS(double Emin,double Emax,int Esize,int numneu,
           string NT,bool elogscale,bool iinteraction);
  Init(double,double,int,int,string,bool,bool);
  ```

  This constructor and initialization function initializes `nuSQUIDS` in the *multiple energy mode*. `Emin`, `Emax` and `Esize` define the energy nodes to be used spaced in either `logaritmic` or `linear` scales depending on the value of `elogscale (true or false)`. Furthermore, `numneu` specifies the number of neutrino flavors which can go from 2 to 6, while `NT` can be set to `"neutrino"` or `"antineutrino"` or `"both"`. Finally, `iinteraction` toggles the neutrino interactions `on (true)` and `off (false)`.

- Constructing from a ν-SQuIDS-HDF5 file

  ```
  nuSQUIDS(string filepath);
  Init(string);
  ```

  This constructor and initialization function initializes `nuSQUIDS` from a previously generated ν-SQuIDS-HDF5 file. The result `nuSQUIDS` object will be given in *single* or *multiple* energy mode depending on the HDF5 file configuration. `filepath` must specify the full path of the HDF5 file.

3.4.2. Functions

Brief description of the `nuSQUIDS` public functions given in alphabetical order.

- Flavor composition evaluator (single energy mode)

  ```
  double EvalFlavor(int flv);
  ```

  Returns the content a given neutrino flavor specified by `flv` (`0 =` e, `1 =` μ, `2 =` τ, ...). This function can only be use in the *single energy mode*.

- Flavor composition evaluator (multiple energy mode)

  ```
  double EvalFlavorAtNode(int flv,int ie,int rho = 0);
  double EvalFlavor(int flv,double Enu,int rho = 0);
  ```

  `EvalFlavorAtNode` returns the content a given neutrino flavor specified by `flv` (`0 =` e, `1 =` μ, `2 =` τ`, ...`) at an energy node `ie`. Furthermore, `EvalFlavor` returns the approximate content of a given flavor for a specific neutrino energy `Enu` by interpolating in the interaction basis. In each function, when considering `NT = "both"`, the parameter `rho` toggles between `neutrino (0)` and `antineutrino (1)`.

- Mass composition evaluator (single energy mode)

  ```
  double EvalMass(int eig);
  ```

  Returns the content a given neutrino mass eigenstate specified by `eig` (`0 =` ν_1, `1 =` ν_2, `2 =` ν_3`, ...`). This function can only be use in the *single energy mode*.

- Mass composition evaluator (multiple energy mode)

  ```
  double EvalMassAtNode(int eig,int ie,int rho = 0);
  double EvalMass(int eig,double Enu,int rho = 0);
  ```

  `EvalMassAtNode` returns the content a given neutrino mass eigenstate specified by `eig` (`0 =` ν_1, `1 =` ν_2, `2 =` ν_3`, ...`) at an energy node `ie`. Furthermore, `EvalMass` returns the approximate content of a given mass eigenstate for a specific neutrino energy `Enu` by interpolating in the interaction basis. In each function, when considering `NT = "both"`, the parameter `rho` toggles between `neutrino (0)` and `antineutrino (1)`.

- Evolve ν state

  ```
  void EvolveState(void);
  ```

  Once the neutrino propagation problem has been setup this function evolves the neutrino state (in either *single* energy or *multiple* energy mode) from its initial position to its final position.

- Get energy nodes values

  ```
  std::vector<double> GetERange(void);
  ```

Returns a vector containing the energy nodes positions given in natural units, i.e. eV.

- Get number of energy nodes

  ```
  int GetNumE(void);
  ```

  Returns the number of energy nodes.

- Get number neutrino flavors

  ```
  int GetNumNeu(void);
  ```

  Returns the number of neutrino flavors.

- Get Hamiltonian at current position

  ```
  SU_vector GetHamiltonian(std::shared_ptr<Track> track, double E,
                           int rho = 0);
  ```

  Returns the `SU_vector` that represents the (anti)neutrino Hamiltonian at a position specified by a `Track` and given neutrino energy `Enu` (in eV). Furthermore, `rho` specifies whether the neutrino or antineutrino Hamiltonian is returned.

- Get `Body`

  ```
  std::shared_ptr<Body> GetBody(void);
  ```

  Returns the `Body` instance currently stored in the `nuSQUIDS` object.

- Get `Track`

  ```
  std::shared_ptr<Track> GetTrack(void);
  ```

  Returns the `Track` instance currently stored in the `nuSQUIDS` object.

- Set `Body`

  ```
  void Set_Body(std::shared_ptr<Body>);
  ```

  Sets the `Body` instance in which the neutrino propagation will take place.

- Set `Track`

  ```
  void Set_Track(std::shared_ptr<Track>);
  ```

  Sets the `Track` instance which describes the neutrino propagation inside a given `Body`.

- Set the initial state

  ```
  void Set_initial_state(array1D, string basis);
  void Set_initial_state(array2D, string basis);
  void Set_initial_state(array3D, string basis);
  ```

`Set_initial_state` sets the initial neutrino (and antineutrino) state. The states can be specified for the single and multiple energy modes can be specified using the different `C++` signatures.

  - `array1D state`: Can only be used in single energy mode and is defined by `state[`α`] = ` ϕ_α where α is a flavor or mass eigenstate index.
  - `array2D state`: Can only be used in multiple energy mode and is defined by `state[ei][`α`] = ` $\phi_\alpha(E$`[ei]`$)$, i.e. the flavor (mass) eigenstate composition at a given energy node `ei`.
  - `array3D state`: Can only be used in multiple energy mode and is defined by `state[ei][`ρ`][`α`] = ` $\phi^\rho_\alpha(E$`[ei]`$)$, i.e. the flavor (mass) eigenstate composition at a given energy node `ei`, and where $\rho = 0 \equiv$ neutrino and $\rho = 0 \equiv$ antineutrino.

- Write the state into an HDF5 file.

```
void WriteStateHDF5(string);
```

  Writes the current `nuSQUIDS` configuration and state into an HDF5 file for later use.

- Read from an HDF5 file.

```
void ReadStateHDF5(string);
```

  Reads an previously generated HDF5 file and sets the `nuSQUIDS` object accordingly, i.e. it configures it and loads the saved state.

4. Examples and benchmarks

4.1. Single energy mode

In this section we illustrate the usage of the *single* energy mode. A full written example can be found in `nuSQUIDS/examples/osc_example.cpp`; here only code snippets will be shown.

4.1.1. Example 1

In this first example we will define a `nuSQUIDS` instance, set the mixing angles, and then calculate $P(\nu_\mu \to \nu_e)$ for a 500 km baseline experiment in the Earth.

```
// create a nuSQUIDS object setting number of flavors = 3
nuSQUIDS nus(3,"neutrino");

// We can change the mixing angles from their defaults
nus.Set("th12",0.563942);
nus.Set("th13",0.154085);
nus.Set("th23",0.785398);
// square mass differences
nus.Set("dm21sq",7.65e-05);
```

```cpp
nus.Set("dm31sq",0.00247);
// CP phase
nus.Set("delta1",0.0);

// We must set the neutrino energy
nus.Set_E(10.0*nus.units.GeV);

// For this example lets consider a long baseline experiment
double baseline = 500.0*nus.units.km;
// create a body object, in this case the Earth
std::shared_ptr<Earth> earth = std::make_shared<Earth>();
// create a trajectory on the body. In this case the Earth
// trajectory is given by three quantities: Initial position,
// Final position, and Baseline.
std::shared_ptr<Earth::Track> earth_track = std::make_shared<Earth
    ::Track>(0.0,baseline,baseline);
// Then we set the body and trajectory in the nuSQUIDS object.
nus.Set_Body(earth);
nus.Set_Track(earth_track);

// We must specify the initial neutrino flavor composition
// In this case we will start with a pure nu_mu state.
std::vector<double> ini_state{0,1,0};
nus.Set_initial_state(ini_state,"flavor");

// Now that we have all the pieces in place we can tell the
// nuSQUIDS object to evolve the given state.
nus.EvolveState();

// Then we can evaluate, for example, the nu_e content
nus.EvalFlavor(0);
```

4.1.2. Example 2

Even though we recommend to use the *multiple* energy mode when considering a range of energies this can also be performed in the single energy mode. Lets assume that a set of energies is provided in `std::vector<double> energy_range`, then the following could be done (results of this can be seen in Fig. 4.1.2)

```cpp
nuSQUIDS nus(3,"neutrino");
double phi = acos(-1.0);
std::vector<double> ini_state{0,1,0};
nus.Set_Body(std::make_shared<EarthAtm>());
nus.Set_Track(std::make_shared<EarthAtm::Track>(phi));
for( double enu : energy_range ){
        nus.Set_E(enu);
        nus.Set_initial_state(ini_state,"flavor");
        nus.EvolveState();
        std::cout << nus.EvalFlavor(1) << std::endl;
}
```

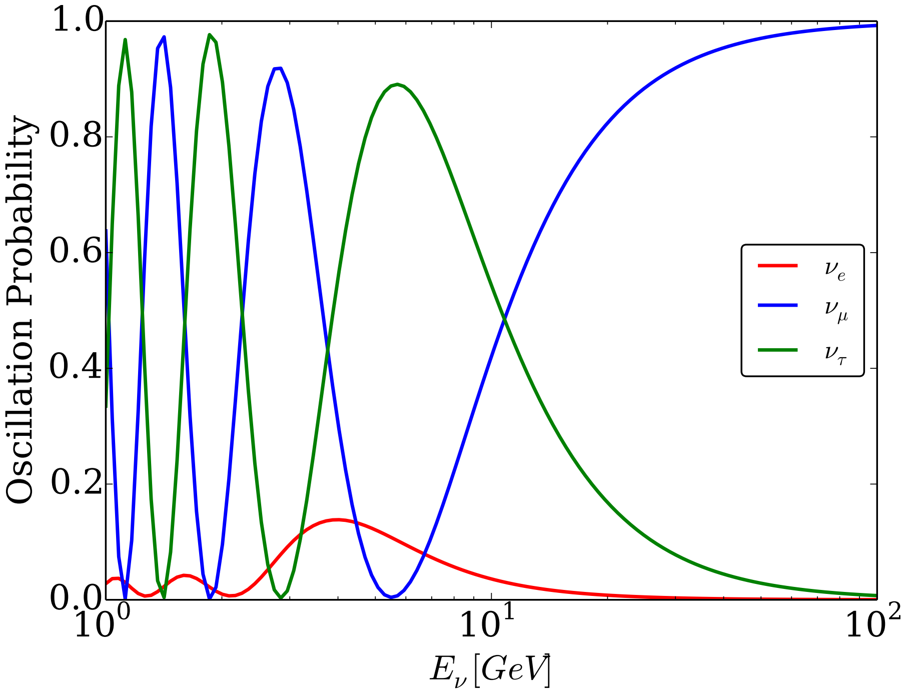

Figure 4.1: Single mode neutrino oscillation probability in a constant density matter environment.

In the above example we could had rather fix the energy and reset the `Track` in order to explore the probability as a function of baseline.

4.1.3. Example 3

We can also use the custom environments provided, such as `ConstantDensity` and `VariableDensity`. For example in case we want to consider $\rho = 13\text{gr/cm}^3$ and $y_e = 0.5$ we can use the following code snipped, which result is shown in Fig. 4.1.3,

```
nuSQUIDS nus(3,"neutrino");
double phi = acos(-1.0);
std::vector<double> ini_state{0,1,0};
nus.Set_Body(std::make_shared<ConstantDensity>(13.0,0.5));
nus.Set_Track(std::make_shared<ConstantDensity::Track>(0.0,1000.0*
    nus.units.km));
for( double enu : energy_range ){
        nus.Set_E(enu);
        nus.Set_initial_state(ini_state,"flavor");
        nus.EvolveState();
        std::cout << nus.EvalFlavor(0) << std::endl;
}
```

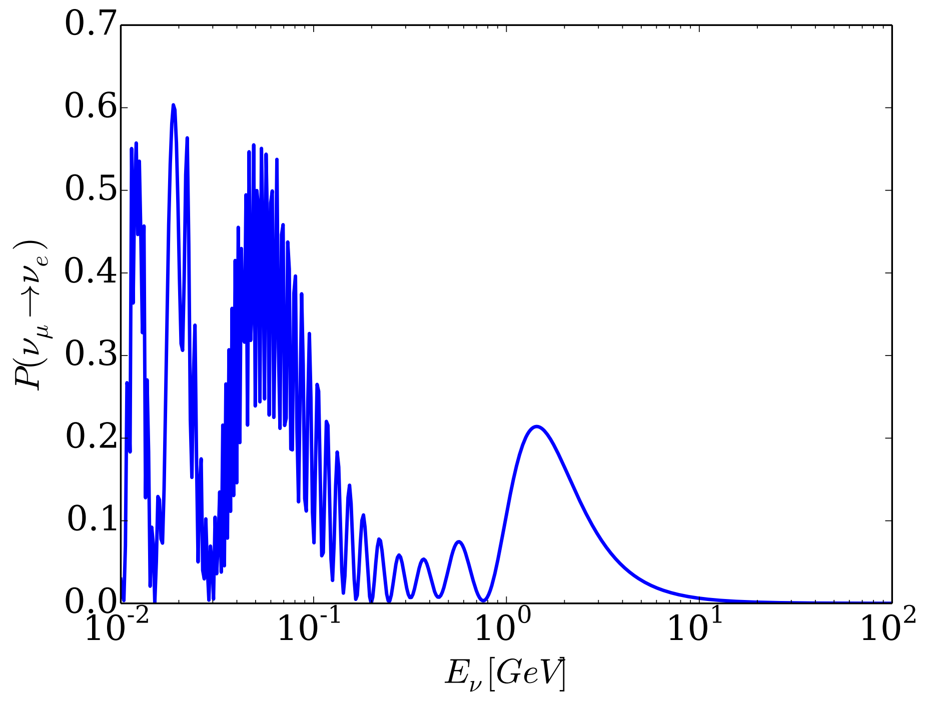

Figure 4.2: Single mode neutrino oscillation probability in a constant density matter environment.

4.2. Multiple energy mode

As explained in the text the *multiple* energy mode can consider a *set* of neutrino energies. The simplest thing that can be done is to

4.2.1. Example 1

In this mode we can actually input a neutrino flux and propagate it in its environment. For example we could propagate $\phi(E) = \phi_0 E^{-1}\Pi_\mu$, i.e. power law pure ν_μ flux, through the Earth. The following piece of code demonstrates this

4.2.2. Example 2

In this mode we can actually input a neutrino flux and propagate it in its environment. For example we could propagate $\phi(E) = \phi_0 E^{-1}\Pi_\mu$, i.e. power law pure ν_μ flux, through the Earth. The following piece of code demonstrates this

4.2.3. Example 3

We can further consider the the propagation of $\phi(E) = \phi_0 E^{-1}(\Pi_\mu + \Pi_\tau)$, again through the Earth, but considering τ regeneration.

5. Conclusions & Acknowledgements

Abazajian, K. N., Acero, M. A., Agarwalla, S. K., Aguilar-Arevalo, A. A., Albright, C. H., Antusch, S., Argüelles, C. A., Balantekin, A. B., Barenboim, G., Barger, V., Bernardini, P., Bezrukov, F., Bjaelde, O. E., Bogacz, S. A., Bowden, N. S., Boyarsky, A., Bravar, A., Berguno, D. B., Brice, S. J., Bross, A. D., Caccianiga, B., Cavanna, F., Chun, E. J., Cleveland, B. T., Collin, A. P., Coloma, P., Conrad, J. M., Cribier, M., Cucoanes, A. S., D'Olivo, J. C., Das, S., de Gouvea, A., Derbin, A. V., Dharmapalan, R., Diaz, J. S., Ding, X. J., Djurcic, Z., Donini, A., Duchesneau, D., Ejiri, H., Elliott, S. R., Ernst, D. J., Esmaili, A., Evans, J. J., Fernandez-Martinez, E., Figueroa-Feliciano, E., Fleming, B. T., Formaggio, J. A., Franco, D., Gaffiot, J., Gandhi, R., Gao, Y., Garvey, G. T., Gavrin, V. N., Ghoshal, P., Gibin, D., Giunti, C., Gninenko, S. N., et al. (129 additional authors not shown), 04 2012. Light sterile neutrinos: A white paper.
URL http://arxiv.org/abs/1204.5379

Agarwalla, S., Akhmedov, E., Blennow, M., Coloma, P., Donini, A., et al., 2012. Euronu-wp6 2010 report.

Akhmedov, E. K., 1999. Neutrino physics, 103–164.

Argüelles Delgado, C. A., Kopp, J., 2012. Sterile neutrinos and indirect dark matter searches in IceCube. JCAP 1207, 016.

Argüelles Delgado, C. A., Salvado Serrat, J., 2014. A simple quantum integro differential solver (squids).

Bahcall, J. N., Serenelli, A. M., Basu, S., 2005. New solar opacities, abundances, helioseismology, and neutrino fluxes. The Astrophysical Journal Letters 621 (1), L85.

Balantekin, A. B., Haxton, W. C., 03 2013. Neutrino oscillations.
URL http://arxiv.org/abs/1303.2272

Blennow, M., Edsjo, J., Ohlsson, T., 2008. Neutrinos from WIMP annihilations using a full three-flavor Monte Carlo. JCAP 0801, 021.

Blennow, M., Smirnov, A. Y., 06 2013. Neutrino propagation in matter.
URL http://arxiv.org/abs/1306.2903

Cirelli, M., Fornengo, N., Montaruli, T., Sokalski, I., Strumia, A., Vissani, F., 2005. Spectra of neutrinos from dark matter annihilations.
URL http://arxiv.org/abs/hep-ph/0506298

Conrad, J. M., Louis, W. C., Shaevitz, M. H., 06 2013. The lsnd and miniboone oscillation searches at high m^2.
URL http://arxiv.org/abs/1306.6494

de Gouvea, A., Pitts, K., Scholberg, K., Zeller, G., Alonso, J., Bernstein, A., Bishai, M., Elliott, S., Heeger, K., Hoffman, K., Huber, P., Kaufman, L., Kayser, B., Link, J., Lunardini, C., Monreal, B., Morfin, J., Robertson, H., Tayloe, R., Tolich, N., Abazajian, K., Akiri, T., Albright, C., Asaadi, J., Babu, K., Balantekin, A., Barbeau, P., Bass, M., Blake, A., Blondel, A., Blucher, E., Bowden, N., Brice, S., Bross, A., Carls, B., Cavanna, F., Choudhary, B., Coloma, P., Connolly, A., Conrad, J., Convery, M., Cooper, R., Cowen, D., da Motta, H., de Young, T., Lodovico, F. D., Diwan, M., Djurcic, Z., Dracos, M., Dodelson, S., Efremenko, Y., Ekelof, T., Feng, J., Fleming, B., Formaggio, J., Friedland, A., Fuller, G., Gallagher, H., Geer, S., Gilchriese, M., Goodman, M., Grant, D., Gratta, G., Hall, C., Halzen, F., Harris, D., Heffner, M., et al. (100 additional authors not shown), 10 2013. Neutrinos.
URL http://arxiv.org/abs/1310.4340

Duan, H., Fuller, G. M., Qian, Y.-Z., 01 2010. Collective neutrino oscillations.
URL http://arxiv.org/abs/1001.2799

Dutta, S. I., Reno, M. H., Sarcevic, I., 2000. Tau neutrinos underground: Signals of muon-neutrino $\rightarrow$ tau-neutrino oscillations with extragalactic neutrinos. Phys.Rev. D62, 123001.

Dziewonski, A. M., Anderson, D. L., 1981. Preliminary reference earth model. Physics of the earth and planetary interiors 25 (4), 297–356.

Folk, M., Cheng, A., Yates, K., 1999. Hdf5: A file format and i/o library for high performance computing applications. In: Proceedings of Supercomputing. Vol. 99.

Gonzalez-Garcia, M., Halzen, F., Maltoni, M., 2005. Physics reach of high-energy and high-statistics icecube atmospheric neutrino data. Phys.Rev. D71, 093010.

Gonzalez-Garcia, M., Maltoni, M., Salvado, J., Schwetz, T., 2012. Global fit to three neutrino mixing: critical look at present precision. Journal of High Energy Physics 2012 (12), 1–24.

Gough, B., 2009. GNU scientific library reference manual. Network Theory Ltd.

Halzen, F., Saltzberg, D., 1998. Tau neutrino appearance with a 1000 megaparsec baseline.
URL http://arxiv.org/abs/hep-ph/9804354

Hewett, J., Weerts, H., Brock, R., Butler, J., Casey, B., Collar, J., de Gouvea, A., Essig, R., Grossman, Y., Haxton, W., Jaros, J., Jung, C., Lu, Z., Pitts, K., Ligeti, Z., Patterson, J., Ramsey-Musolf, M., Ritchie, J., Roodman, A., Scholberg, K., Wagner, C., Zeller, G., Aefsky, S., Afanasev, A., Agashe, K., Albright, C., Alonso, J., Ankenbrandt, C., Aoki, M., Arguelles, C., Arkani-Hamed, N., Armendariz, J., Armendariz-Picon, C., Diaz, E. A., Asaadi, J., Asner, D., Babu, K., Bailey, K., Baker, O., Balantekin, B., Baller, B., Bass, M., Batell, B., Beacham, J., Behr, J., Berger, N., Bergevin, M., Berman, E., Bernstein, R., Bevan, A., Bishai, M., Blanke, M., Blessing, S., Blondel, A., Blum, T., Bock, G., Bodek, A., Bonvicini, G., Bossi, F., Boyce, J., Breedon, R., Breidenbach, M., Brice, S., Briere, R., Brodsky, S., et al. (403 additional authors not shown), 05 2012. Fundamental physics at the intensity frontier.
URL http://arxiv.org/abs/1205.2671

Huber, P., Kopp, J., Lindner, M., Rolinec, M., Winter, W., 2007. New features in the simulation of neutrino oscillation experiments with GLoBES 3.0: General Long Baseline Experiment Simulator. Comput.Phys.Commun. 177, 432–438.

Kopp, J., Machado, P. A., Maltoni, M., Schwetz, T., 2013. Sterile neutrino oscillations: the global picture. Journal of High Energy Physics 2013 (5), 1–52.

Mikheev, S., Smirnov, A. Y., 1985. Resonance Amplification of Oscillations in Matter and Spectroscopy of Solar Neutrinos. Sov.J.Nucl.Phys. 42, 913–917.

Mikheev, S., Smirnov, A. Y., 1986. Resonant amplification of neutrino oscillations in matter and solar neutrino spectroscopy. Nuovo Cim. C9, 17–26.

Mohapatra, R., Antusch, S., Babu, K., Barenboim, G., Chen, M.-C., Davidson, S., de Gouvea, A., de Holanda, P., Dutta, B., Grossman, Y., Joshipura, A., Kayser, B., Kersten, J., Keum, Y., King, S., Langacker, P., Lindner, M., Loinaz, W., Masina, I., Mocioiu, I., Mohanty, S., Murayama, H., Pascoli, S., Petcov, S., Pilaftsis, A., Ramond, P., Ratz, M., Rodejohann, W., Shrock, R., Takeuchi, T., Underwood, T., Wolfenstein, L., 2005. Theory of neutrinos: A white paper.
URL http://arxiv.org/abs/hep-ph/0510213

Strack, P., Burrows, A., 2005. Generalized boltzmann formalism for oscillating neutrinos.
URL http://arxiv.org/abs/hep-ph/0504035

Wolfenstein, L., 1978. Neutrino Oscillations in Matter. Phys.Rev. D17, 2369–2374.

Zhang, Y., Burrows, A., 10 2013. Transport equations for oscillating neutrinos.
URL http://arxiv.org/abs/1310.2164