

SQuIDS: A Tool to Solve Time Evolution in finite dimensional (open) Quantum Systems

An Application to Neutrino Oscillations

arxiv:1412.3832

Dominik Hellmann

TU Dortmund WG Päs

May 3, 2023



Outline:

- 1. Introduction (Quantum Evolution with Density Matrices)
- 2. SQuIDS (Overview and Exercises)
 - 2.1 The Const class (Overview + Exercise)
 - 2.2 The SU_vector class (Overview + Exercise)
 - 2.3 The SQuIDS class (Overview + Exercise)

Motivation

Task: Solve time evolution of finite dimensional quantum (sub-)systems:

- ► Flavor oscillations
- Quantum computation
- ► Systems with finitely many energy levels
- ► Spins

Time evolution of closed quantum system: Schrödinger equation

$$i\frac{\partial}{\partial t}|\psi\rangle = \hat{H}|\psi\rangle \qquad (\hbar = 1)$$
 (1)



Density matrices instead of state vectors

Often: finite dimensional system S coupled to a complicated (but uninteresting) environment E

- → Get rid of Environment (keyword: partial trace)
- → Just consider degrees of freedom of interest

Consequence: Decoherence

- lacktriangle Subsystem cannot be described by pure state $|\psi\rangle$
- ▶ Mixed state: Described by density matrix $\varrho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$



Example: Neutrino oscillations in matter

$\mathcal{S}\simeq\mathbb{C}^3$	E
flavor degrees of freedom $ \psi\rangle = \sum_{\alpha=e}^{\tau} \psi_{\alpha} \nu_{\alpha}\rangle$	all remaining d.o.f (momenta, spins,) \rightarrow infinite dimensional

- ▶ We are not at all interested in E
- Only the ν flavor composition is interesting to us
- **But**: E significantly influences flavor d.o.f.
- ⇒ Need effective description!

Time evolution of the density matrix

Master equation(s): (multiple density matrices possible)

$$\frac{\mathrm{d}\varrho_j}{\mathrm{d}t} = -i[\hat{H}_j(t),\varrho_j(t)] + \{\Gamma_j(t),\varrho_j(t)\} + F_j[\{\varrho_k\}_k,t]$$

- ▶ Why multiple ϱ_i ? E.g.: One per energy bin!
- $\hat{H} = \hat{H}_0 + \hat{H}_1(t)$: Unitary evolution
- Γ: Decoherence
- ▶ F: Other non-linear effects (coupling between ϱ_i)



Simple Example: ν Oscillations in Vacuum

Neutrino Experiment:

- Fixed baseline L
- \triangleright N energy bins $\{E_i\}_i$
- $\hat{H}^j = \hat{H}_0^j = E_j \cdot \mathbb{I} + \frac{1}{2E} \mathbb{M}^2$
- $\Gamma = 0$
- F=0
- $\rho_i(t) = \sum_{\alpha=0}^{\tau} \phi_{\alpha}^j(t) |\nu_{\alpha}\rangle\langle\nu_{\alpha}|$

$$\mathbb{M} = \sum_{j,k=1}^{3} (\mathbb{M}_0)_{jk} |\nu_j\rangle\langle\nu_k|$$

$$= \sum_{lpha,eta=e}^{ au} (\mathbb{M}_1)_{lphaeta} |
u_lpha
angle\langle
u_eta|$$
 $\mathbb{M}_0 = egin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{pmatrix}$
 $\mathbb{M}_1 = U_{\mathrm{PMNS}}^{\dagger} \mathbb{M}_0 U_{\mathrm{PMNS}}$



Some Important Considerations

The master equation simplifies to

$$\frac{\mathrm{d}\varrho_j}{\mathrm{d}t} = -i[\hat{H}_0^j, \varrho_j(t)]$$

Further simplifications

- ightharpoonup Consider in mass basis: \hat{H}_0^j is diagonal
- ► Depends only on commutator!
 - $[A, \mathbb{I}] = 0 \Rightarrow [\hat{H}_0^j, \varrho_i(t)] = [\hat{H}_0^j \epsilon_i \mathbb{I}, \varrho_i(t)]$

$$ilde{H}^j = rac{1}{2 E_j} egin{pmatrix} 0 & 0 & 0 & 0 \ 0 & \Delta m_{21}^2 & 0 \ 0 & 0 & \Delta m_{31}^2 \end{pmatrix}$$



Some Important Considerations (ctd.)

Can solve H_0 evolution analytically!

Pass to interaction picture:

$$\begin{split} \tilde{\varrho}(t) &:= \exp(iH_0t)\varrho \exp(-iH_0t) \\ \Rightarrow \dot{\varrho} &= -i[H_0, \rho] + \exp(-iH_0t)\dot{\tilde{\varrho}} \exp(iH_0t) \end{split}$$

- Can subtract $-i[H_0, \varrho]$ on both sides of master equation
- Must transform all terms to interaction picture (SQuIDS does that automatically and efficiently)

Some Important Considerations (ctd.)

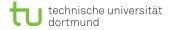
All matrices in our system are hermitian: $A^{\dagger} = A$

- ▶ Hermitian $n \times n$ matrices form $N = n^2$ dimenional real vector space
- ► Convenient basis: SU(n) generators σ_i (e.g. n=2: Pauli matrices + identity)
- ▶ Decompose: $\varrho = \sum_{i=0}^{n^2-1} \rho_i \cdot \sigma_i$
- ▶ Components ρ_i form n^2 dimensional vector called SU_vector in the following

Summary

What did we learn so far (in general):

- 1. We passed to density matrix formulation (allows for mixed states)
- 2. Formulated master equation
- 3. Can subtract $\epsilon_0 \cdot \mathbb{I}$ from \hat{H} (only energy diff. important)
- 4. Can solve \hat{H}_0 exactly (interaction picture) $\varrho \to e^{i\hat{H}_0t}\varrho e^{-i\hat{H}_0t}$
- 5. Can represent ϱ, H, \ldots as n^2 dimensional, real vector (SU_vector)!
 - → Efficient and preserves hermiticity automatically!





Clone the Repo!

Git Repository includes all needed files (slides, code templates)

Instructions

cd to the location where you want to place the repo
git clone https://github.com/BObsen/sm_to_bsm_neutrino.git
cd sm_to_bsm_neutrino



Overview

SQuIDS mainly consists out of 3 interconnected classes:

- 1. squids::Const
 - ► Implements all sorts of constants of nature
 - ► Conversion between natural units and other unit systems
 - ► Stores system parameters (mixing angles, energy differences, ...)



Overview

SQuIDS mainly consists out of 3 interconnected classes:

- 1. squids::Const
 - ► Implements all sorts of constants of nature
 - ► Conversion between natural units and other unit systems
 - ► Stores system parameters (mixing angles, energy differences, ...)
- 2. squids::SU_vector
 - Represents hermitian matrices efficiently
 - ► Implements all sorts of operations on them (Unitary transformations, trace, commutator, . . .)



Overview

SQuIDS mainly consists out of 3 interconnected classes:

- 1. squids::Const
 - ▶ Implements all sorts of constants of nature
 - ► Conversion between natural units and other unit systems
 - ► Stores system parameters (mixing angles, energy differences, ...)
- 2. squids::SU_vector
 - Represents hermitian matrices efficiently
 - ► Implements all sorts of operations on them (Unitary transformations, trace, commutator, ...)
- 3. squids::SQuIDS
 - ► Abstract base class, uses squids::Const and squids::SU_vector
 - ▶ Implements time evolution of the system of density matrices
 - Includes methods for taking expectation values etc



SQuIDS - The Const Class



Const - Construction of objects

Can only be default constructed:

```
squids::Const units;
```

- ► Constructs squids::Const object called units
- ► This object contains
 - ▶ Different physical constants $(G_F, N_A, G, m_p, ...)$
 - ► Values of km, s, J, kg, etc. in natural units
 - Yet unspecified values for:
 - **b** basis change from B_0 to B_1 (e.g. mass and flavor basis)
 - energy differences which can be used for the hamiltonian \hat{H}_0



Const - Unit conversion

Easily convert between SI and natural units:

```
The units are to be read as [unit we want] / [eV^{\alpha}], e.g.: km / [eV^{-1}]: [Value in eV^{-1}] = [Value in km] \cdot km / [eV^{-1}] [Value in km] = L / km
```



Const - Setting / Getting Mixing Angles

Furthermore you can store system parameters:

```
\label{eq:squids::Const params;} $$ \sup_{0.2} = 24^\circ$ $$ params.SetMixingAngle(0, 1, 24 * params.degree); $$ \sum_{0.2} \delta_{13} = 2^\circ$ $$ params.SetPhase(0, 2, 2 * params.degree); $$ \sum_{0.2} \Delta E_{10} = 7eV$ $$ params.SetEnergyDifference(1, 7 * params.eV); $$
```

- ► Substitute Set for Get: returns corresponding value
- ▶ Energy differences: Only convenience parameters simplifying definition of \hat{H}_0



SQuIDS - The Const Class: Exercise

Const class exercise

- 1. Declare a default constructed const class object
- 2. Answer the following questions:
 - 2.1 How many eV^{-1} correspond to 300 km
 - 2.2 How many radians correspond to 25°
 - 2.3 If you are 24 years old, how many eV^{-1} are you old?
- 3. Set the mixing parameters for three neutrino generations to:
 - $\theta_{12} = 33.48^{\circ}$
 - $\theta_{13} = 8.55^{\circ}$
 - $\theta_{23} = 42.3^{\circ}$
- 4. Set the energy differences to:
 - $\Delta m_{21}^2 = 7.5 \cdot 10^{-5} \,\mathrm{eV}^2$
 - $\Delta m_{31}^{21} = 2.45 \cdot 10^{-3} \, \text{eV}^2$



SQuIDS - The $SU_vector\ Class$



SU Vector



SQuIDS - The SU_vector Class: Exercises



SU vector exercise

- 1. Declare an empty SU vector corresponding to a 3D Hilbert space
- 2. Initialize an array of projectors for the three mass eigenstates (B_0)
- 3. Rotate them to the flavor basis (B_1)
- 4. Initialize a SU vector corresponding to the matrix (B_0)

$$\Delta \mathbb{M}^2 := egin{pmatrix} 0 & 0 & 0 \ 0 & \Delta m_{21}^2 & 0 \ 0 & 0 & m_{31}^2 \end{pmatrix}$$



SQuIDS - The SQuIDS Class



SQuIDS



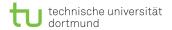
SQuIDS - The SQuIDS Class: Exercises



SQuIDS application: Neutrino oscillations in vacuum







BACK UP



Installation

What do we need for this tutorial?

- ► A unix-like (sub-)system
 - ► Linux
 - ► Mac (+ Xcode developer tools!)
 - On Windows: WSL
- ► A C++ compiler
- ► Make, wget, Git

Use scripts install_gsl.sh and install_SQuIDS.sh from the repo!



Installation (GSL)

```
cd $HOME
mkdir -p smToBsmLibs/gsl
wget ftp://ftp.gnu.org/gnu/gsl/gsl-latest.tar.gz
tar -zxvf gsl-latest.tar.gz
rm gsl-latest.tar.gz
cd $(find gsl-* | head -n 1)
./configure --prefix=$HOME/smToBsmLibs/gsl
make
make check
make install
LD_LIBRARY_PATH=$HOME/smToBsmLibs/gsl/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
cd $HOME
rm -rf $(find gsl-* | head -n 1)
```



Installation (SQuIDS)

```
cd $HOME
mkdir -p smToBsmLibs/SQuIDS
git clone https://github.com/jsalvado/SQuIDS.git
cd $(find SQuIDS* | head -n 1)
./configure --with-gsl-incdir=$HOME/smToBsmLibs/gsl/include \
--with-gsl-libdir=$HOME/smToBsmLibs/gsl/lib \
--prefix=$HOME/smToBsmLibs/SQuIDS
make
make test
make install
LD_LIBRARY_PATH=$HOME/smToBsmLibs/SQuIDS/lib:$LD_LIBRARY_PATH # linux only
export LD_LIBRARY_PATH # linux only
cd $HOME
rm -rf $(find SQuIDS* | head -n 1)
```