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:

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

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

Density matrices instead of state vectors

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

$\rightarrow$ Get rid of Environment (keyword: partial trace)

$\rightarrow$ Just consider degrees of freedom of interest

Consequence: Decoherence

▶ 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

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

$\Rightarrow$ Need effective description!

Time evolution of the density matrix

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

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

▶ Why multiple ϱ_j? E.g.: One per energy bin!

▶ $\hat{H} = \hat{H}_0 + \hat{H}_1(t)$: Unitary evolution

Time evolution of the density matrix

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

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

▶ Why multiple ϱ_j? E.g.: One per energy bin!

▶ $\hat{H} = \hat{H}_0 + \hat{H}_1(t)$: Unitary evolution

▶ Γ: Decoherence, Attenuation

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_j? E.g.: One per energy bin!

▶ $\hat{H} = \hat{H}_0 + \hat{H}_1(t)$: Unitary evolution

▶ Γ: Decoherence, Attenuation

▶ F: Other non-linear effects (coupling between ϱ_j)

Simple Example: ν Oscillations in Vacuum

Neutrino Experiment:

- ▶ Fixed baseline L
- ▶ N energy bins $\{E_j\}_j$
- ▶ $\hat{H}^j = \hat{H}_0^j = E_j \cdot \mathbb{I} + \frac{1}{2E_j}\mathbb{M}^2$
- ▶ $\Gamma \equiv 0$
- ▶ $F \equiv 0$
- ▶ $\varrho_j(t) = \sum_{\alpha,\beta=e}^{\tau} \phi_{\alpha\beta}^j(t)|\nu_\alpha\rangle\langle\nu_\beta|$

Simple Example: ν Oscillations in Vacuum

Neutrino Experiment:

- Fixed baseline L
- N energy bins $\{E_j\}_j$
- $\hat{H}^j = \hat{H}^j_0 = E_j \cdot \mathbb{I} + \frac{1}{2E_j}\mathbb{M}^2$
- $\Gamma \equiv 0$
- $F \equiv 0$
- $\varrho_j(t) = \sum_{\alpha,\beta=e}^{\tau} \phi^j_{\alpha\beta}(t)|\nu_\alpha\rangle\langle\nu_\beta|$

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

$$= \sum_{\alpha,\beta=e}^{\tau} (\mathbb{M}_1)_{\alpha\beta}|\nu_\alpha\rangle\langle\nu_\beta|$$

$$\mathbb{M}_0 = \begin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{pmatrix}$$

$$\mathbb{M}_1 = U^\dagger_{\mathrm{PMNS}}\mathbb{M}_0 U_{\mathrm{PMNS}}$$

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

- ▶ Consider in mass basis: $\hat{H}_0^j$ is diagonal
- ▶ Depends only on commutator!

$$[A, \mathbb{I}] = 0 \quad \Rightarrow \quad [\hat{H}_0^j, \varrho_j(t)] = [\hat{H}_0^j - \epsilon_j \mathbb{I}, \varrho_j(t)]$$

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

▶ Consider in mass basis: $\hat{H}_0^j$ is diagonal

▶ Depends only on commutator!

$$[A, \mathbb{I}] = 0 \quad \Rightarrow \quad [\hat{H}_0^j, \varrho_j(t)] = [\hat{H}_0^j - \epsilon_j\mathbb{I}, \varrho_j(t)]$$

$$\epsilon_j := \langle \nu_1 | \hat{H}_0^j | \nu_1 \rangle = E_j + \frac{m_1^2}{2E_j} \quad \text{(Ground state energy)}$$

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

▶ Consider in mass basis: $\hat{H}_0^j$ is diagonal

▶ Depends only on commutator!

$$[A, \mathbb{I}] = 0 \quad \Rightarrow \quad [\hat{H}_0^j, \varrho_j(t)] = [\hat{H}_0^j - \epsilon_j \mathbb{I}, \varrho_j(t)]$$

$$\epsilon_j := \langle \nu_1 | \hat{H}_0^j | \nu_1 \rangle = E_j + \frac{m_1^2}{2E_j} \quad \text{(Ground state energy)}$$

$$\Rightarrow \tilde{H}^j = \frac{1}{2E_j} \begin{pmatrix} 0 & 0 & 0 \\ 0 & \Delta m_{21}^2 & 0 \\ 0 & 0 & \Delta m_{31}^2 \end{pmatrix}$$

Some Important Considerations (ctd.)

Can solve H_0 evolution analytically!

▶ Switch to interaction picture:

$$\tilde{\varrho}(t) := \exp(iH_0 t)\varrho \exp(-iH_0 t)$$
$$\Rightarrow \dot{\varrho} = -i[H_0, \rho] + \exp(-iH_0 t)\dot{\tilde{\varrho}}\exp(iH_0 t)$$

▶ 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) $\varrho \to e^{i\hat{H}_0 t} \varrho e^{-i\hat{H}_0 t}$
5. Can represent $\varrho, H, \ldots$ as n^2 dimensional, real vector (`SU_vector`)!
   - $\to$ Efficient and preserves hermiticity automatically!

SQuIDS

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/B0bsen/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

technische universität
dortmund

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 ($G_F, N_A, G, m_p, \ldots$)
  ▶ Values of km, s, J, kg, etc. in natural units
  ▶ Yet unspecified values for:
    ▶ 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:

```
squids::Const units;
double L = 10 * units.cm; \\ 506773 eV^{-1}
double T = 1 * units.year; \\ 4.79116e+22 eV^{-1}
double GF = units.GF * (units.GeV * units.GeV); \\
1.16638e-05 GeV^{-2}
```

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:

```
squids::Const params;
\\ Sets θ₁₂ = 24°
params.SetMixingAngle(0, 1, 24 * params.degree);
\\ Sets δ₁₃ = 2°
params.SetPhase(0, 2, 2 * params.degree);
\\ Sets ΔE₁₀ = 7eV
params.SetEnergyDifference(1, 7 * params.eV);
```

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

technische universität
dortmund

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:
   - ► $\theta_{12} = 33.48°$
   - ► $\theta_{13} = 8.55°$
   - ► $\theta_{23} = 42.3°$
4. Set the energy differences to:
   - ► $\Delta m_{21}^2 = 7.5 \cdot 10^{-5}\,eV^2$
   - ► $\Delta m_{31}^2 = 2.45 \cdot 10^{-3}\,eV^2$

SQuIDS - The SU_vector Class

SU Vector - Contructors

Construct a SU Vector:

▶ Default: No entries, no size

```
squids::SU_vector rho;
```

▶ Defined Dimension: entries are zero, size is $\dim(\mathcal{H}) > 0$

```
squids::SU_vector rho(dim); \\size = dim²
```

And more: construct from `array`, `gsl_matrix_complex`, `std::vector`
...

SU Vector - Special Matrices

Construct a SU Vector to Standard Form:

▶ Identity: Corresponding matrix is the identity $(Id)_{ij} = \delta_{ij}$

```
squids::SU_vector Id
= squids::SU_vector::Identity(dim);
```

▶ Projector on state k_0: only the (k_0, k_0) element is 1, i.e.
$(\mathbb{P}_{k_0})_{ij} = \delta_{ik_0}\delta_{jk_0}$

```
squids::SU_vector P_k0
= squids::SU_vector::Projector(dim, k0);
```

And more: See documentation (also in the repo)

SU Vector - Functions

Manipulate a SU Vector:

▶ Rotate from one basis to another ($B_0 \leftrightarrow B_1$): (using angles specfied in params)

```
rho.RotateToB1(params); \\B0 to B1
rho.RotateToB0(params); \\B1 to B0
```

▶ Return dim of Hilbert space, Size of vector:

```
rho.Dim(); \\returns dim(H)
rho.Size(); \\returns number of elements
```

Furthermore: You can add, subtract multiply (also with scalars), assign, access elements via rho[] etc.

SQuIDS – The SU_vector Class: Exercise

From SM to BSM - 2023

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)

$$\Delta\mathbb{M}^2 := \begin{pmatrix} 0 & 0 & 0 \\ 0 & \Delta m_{21}^2 & 0 \\ 0 & 0 & m_{31}^2 \end{pmatrix} \qquad (2)$$

SQuIDS - The SQuIDS Class

SQuIDS - General Overview pt. I

`squids::SQuIDS` is a so called base class:

▶ You derive your own class from it
▶ It provides fundamental functionality through member functions
  ▶ Integration of the master equation
  ▶ Efficient memory management and storage of density matrices
▶ Already includes several members storing basic data of your system
▶ Virtual member functions:
  ▶ Some you have to define for the class to work (`H0`, constructors, `init`, ...)
  ▶ Some you can ignore if your system doesn't need them

SQuIDS - General Overview pt. II

Distribution of density matrices across the nodes [arxiv:1412.3832]:
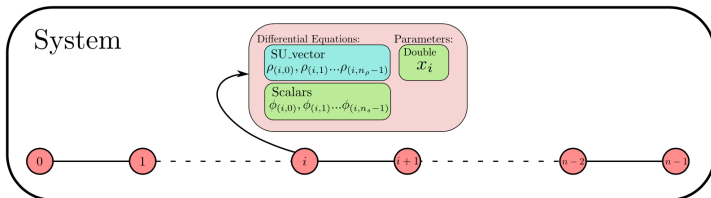
- ▶ n nodes x_i (energy bins, angles, whatever feature of your system)
- ▶ At each node:
    - ▶ nrho density matrices
    - ▶ ns scalars (not important for us)

$\rightarrow$ Specify initial data, H0, HI, GammaRho, etc.

$\Rightarrow$ SQuIDS evolves the whole system ($+$ can take expectation values of observables)

SQuIDS - Constructors

Constructors / Initializors: (Set bkg. params and allocate memory)

```
SQuDIS(); \\ default
SQuDIS(uint nx, uint dim, unit nrho, uint nscalar,
double ti = 0);
void ini(uint nx, uint dim, unit nrho, uint nscalar,
double ti = 0);
```

▶ nx: Number of x nodes x_i
▶ dim: Dimensions of Hilbert space
▶ nrho / nscalar: # density matrices / scalars *per node*
▶ ti: initial time, defaults to zero

Call them in your own constructor with the system parameters you need!

SQuIDS - Member variables

SQuIDS is set up to include the following member variables:

- ▶ `std::vector<double> x`: x range
- ▶ `uint nsun`: Dim. of Hilbertspace
- ▶ `Const params`: `squids::Const` object containing system parameters
- ▶ And many more!

You can (and should) access these from within your own member functions!

SQuIDS - Handy functions

Some very useful predefined member functions are:

```
int Set_xrange(double xini, double xend, std::string
scale); \\ Sets x = {xini, ..., xend} with lin or log
scale
```

SQuIDS - Handy functions

Some very useful predefined member functions are:

```
int Set_xrange(double xini, double xend, std::string
scale); \\ Sets x = {xini, ..., xend} with lin or log
scale
double Get_x(uint i) const; \\ Returns x[i]
```

SQuIDS - Handy functions

Some very useful predefined member functions are:

```
int Set_xrange(double xini, double xend, std::string
scale); \\ Sets x = {xini, ..., xend} with lin or log
scale
double Get_x(uint i) const; \\ Returns x[i]
const * Const Get_params() const; \\ Returns Const member
```

SQuIDS - Handy functions

Some very useful predefined member functions are:

```
int Set_xrange(double xini, double xend, std::string
scale); \\ Sets x = {xini, ..., xend} with lin or log
scale
double Get_x(uint i) const; \\ Returns x[i]
const * Const Get_params() const; \\ Returns Const member
int Evolve(double dt); \\ Evolves System by dt
```

SQuIDS - Handy functions

Some very useful predefined member functions are:

```
int Set_xrange(double xini, double xend, std::string
scale); \\ Sets x = {xini, ..., xend} with lin or log
scale
double Get_x(uint i) const; \\ Returns x[i]
const * Const Get_params() const; \\ Returns Const member
int Evolve(double dt); \\ Evolves System by dt
double GetExpectationValue(SU_vector op, uint irho, uint
ix) const; \\ Calculates exp. val. of op at node ix with
density matrix irho at current time
```

SQuIDS - Handy functions

Some very useful predefined member functions are:

```
int Set_xrange(double xini, double xend, std::string
scale); \\ Sets x = {xini, ..., xend} with lin or log
scale
double Get_x(uint i) const; \\ Returns x[i]
const * Const Get_params() const; \\ Returns Const member
int Evolve(double dt); \\ Evolves System by dt
double GetExpectationValue(SU_vector op, uint irho, uint
ix) const; \\ Calculates exp. val. of op at node ix with
density matrix irho at current time
```

Of course there are more but these are most important for us!

SQuIDS - Evolution functions

Last but not least: The time independent Hamiltonian H0

```
SU_vector H0(double x, uint irho) const;
```

▶ Returns $\hat{H}_0$ as SU_vector object
▶ Assumes that $\hat{H}_0$ diagonal (i.e. given in mass basis B0 for ν oscillations)
▶ Cannot modify but can read member variables (const)
▶ irho: H0 for density matrix at node x_i

technische universität
dortmund

SQuIDS - The SQuIDS Class: Exercise

SQuIDS application: Neutrino oscillations in vacuum

General set up: Vacuum Oscillations Experiment with . . .

▶ Fixed baseline L ($\hat{=}$ time variable)

▶ `n` logarithmic energy bins: $E \in [10\,\mathrm{MeV}, 10\,\mathrm{GeV}]$

▶ All neutrinos are produced as electron neutrinos

Use: `x` nodes as energy nodes, one density matrix per node, no scalar functions, only `H0` non-zero

SQuIDS application: Neutrino oscillations in vacuum

1. Declare vacuum class publically derived from squids::SQuIDS class in `vac/src/vacuum.hpp` with methods
   - ▶ Default constructor
   - ▶ Initializing constructor
   - ▶ `H0`
   - ▶ `GetProbabilities`
   - ▶ Destructor if needed

2. Define the corresponding member functions in `vac/src/vacuum.cpp`
   - ▶ `nbins` x-nodes corresponding to number of E bins (log scaled)
   - ▶ One density function per node, no scalar functions
   - ▶ `nflavor` neutrino flavors
   - ▶ Initial condition $\varrho_i(t=0) = \mathbb{P}_e$ for all $i \in \{1, \ldots, \mathtt{nbins}\}$
   - ▶ Work in mass basis!!!

SQuIDS application: Neutrino oscillations in vacuum

3. Initialize an object from your class in `vac/src/main.cpp` (nbins = 1000, `nflavor` = 3, $E \in [10\,\mathrm{MeV}, 10\,\mathrm{GeV}]$)
4. Evolve the system for $L = 300\ \mathrm{km}$
5. Save the oscillation probabilities $P_{e\alpha}(E_j, L)$ to file(s) $\alpha = e, \mu, \tau$
6. Plot them against the energy

Outlook

- ▶ So far we only scratched the surface of SQuIDS' abilities
- ▶ Setting $\hat{H}_1(t, \varrho), \Gamma(t, \varrho), F(t, \varrho) \neq 0$ opens up possibilities to include decoherence, interactions, etc.
- ▶ These extra terms are solved numerically (using GSL)
- ▶ Examples for neutrinos:
  - ▶ Wave packet decoherence
  - ▶ Neutrino propagation through the sun / earth
  - ▶ Collective neutrino oscillations (early universe, supernovae, ...)
  - ▶ Active sterile oscillations by using more than three generations

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)
```