# lime

## *Release 1.0.0*

**Bing Gu**

**May 12, 2022**

# CONTENTS:

The goal is to provide a simple-to-use package to study how light interacts with matter.

Check docs/manual.pdf for theoretical details.

Main modules

- Nonlinear molecular spectroscopy
- Molecular quantum dynamics

---

- **Adiabatic wavepacket dynamics**
  - Split-operator method
  - Discrete variable representation
- **Nonadiabatic wavepacket dynamics**
  - Split-operator method - For the exact nonadiabatic dynamics of vibronic models in the diabatic representation.
  - RK4 - For the exact nonadiabatic wavepacket dynamics in the adiabatic representation.

# Semiclassical quantum trajectory method

# Quantum chemistry

CHAPTER

ONE

# OPEN QUANTUM SYSTEMS

- Lindblad quantum master equation
- Redfield theory
- second-order time-convolutionless master equation
- hierarchical equation of motion

# Quantum transport - Landauer transport

# CHAPTER TWO

# SOID STATE MATERIALS

- Band structure from tight-binding Hamiltonians

CHAPTER

# THREE

# PERIODICALLY DRIVEN MATTER

- Floquet spectrum

## 3.1 Users Guide

### 3.1.1 Nonlinear molecular spectroscopy

1. Sum-of-states (SOS) for multilevel system

   Pros: computationally cheap and intuitive

   **Note:** In the SOS method, decay is introduced phenomelogically.

2. correlation function approach – Direct computing the many-point correlation functions with the quantum regression theorem

   Valid for open quantum systems where environment effects can be rigirously described with quantum master equations.

   Visulize with double-sided Feynman diagrams instead of the time-loop diagrams.

3. non-perturbative approah – valid for ALL systems provided a quantum dynamics solver is provided.

   Simulating the laser-driven dynamics including explicitly all laser pulses

## 3.2 lime package

### 3.2.1 Subpackages

### 3.2.2 Submodules

### 3.2.3 lime.Floquet module

### 3.2.4 lime.FranckCondon module

Created on Thu Aug 12 14:50:39 2021

@author: Bing

`lime.FranckCondon.`**`FranckCondon`**(*Ln*, *Lm*, *d*)

Analytical formula for the Franck-Condon factors from

Chang, J.-L. Journal of Molecular Spectroscopy 232, 102–104 (2005).

**Parameters**

- **Ln** (*TYPE*) – DESCRIPTION.
- **Lm** (*TYPE*) – DESCRIPTION.
- **d** (*TYPE*) – DESCRIPTION.

**Returns** Franck-Condon overlap.

**Return type** float

`lime.FranckCondon.`**`dfactortial`**(*n*)

### 3.2.5 lime.backup module

### 3.2.6 lime.beam module

Created on Fri Oct 8 17:10:45 2021

@author: bing

`lime.beam.`**`beam`**(*rho*, *phi*, *z*, *k*, *pol=[1, 0, 0]*, *l=0*)

cylindrical vector beam

**Parameters**

- **rho** (*TYPE*) – DESCRIPTION.
- **phi** (*TYPE*) – DESCRIPTION.
- **z** (*TYPE*) – DESCRIPTION.
- **k** (*TYPE*) – DESCRIPTION.
- **pol** (*TYPE, optional*) – DESCRIPTION. The default is [1, 0, 0].
- **l** (*TYPE, optional*) – DESCRIPTION. The default is 0.

**Returns** DESCRIPTION.

**Return type** TYPE

### 3.2.7 lime.cavity module

### 3.2.8 lime.common module

`lime.common.`**`dagger`**(*H*)

`lime.common.`**`delta`**(*n*, *m*)

### 3.2.9 lime.coordinates module

### 3.2.10 lime.correlation module

Created on Wed Dec 18 20:36:55 2019

@author: bing

`lime.correlation.correlation_3p_1t`(*H*, *rho0*, *ops*, *c_ops*, *tlist*, *dyn*, **args*)

compute the time-translation invariant two-point correlation function in the density matrix formalism using quantum regression theorem

<AB(t)C> = Tr[ AU(t) B rho_0 C U^dag(t)]

the density matrix is stored in 'dm.dat' the correlation function is stored in 'corr.dat'

**input:** H: full Hamiltonian rho0: initial wavepacket ops: list of operators [A, B] dyn: dynamics method e.g. lindblad, redfield, heom args: dictionary of parameters for dynamics

**output:** t:

`lime.correlation.correlation_4p_2t`()

### 3.2.11 lime.fft module

`lime.fft.dft`(*x*, *f*, *k*)

Discrete Fourier transfrom at specified momentum

`lime.fft.dft2`(*x*, *y*, *f*, *kx*, *ky*)

Discrete Fourier transfrom at specified momentum

`lime.fft.fft`(*f*, *x=None*, *axis=- 1*, ***kwargs*)

customized fourier transform of function f

g(omega) = int dt f(t) * exp(- i * omega * t)

**Parameters**

- **f** (*ndarray*) – input data
- **x** (*TYPE, optional*) – the grid points. The default is None.
- **axis** (*int, optional*) – Axis over which to compute the FFT. If not given, the last axis is used.
- ****kwargs** (*TYPE*) – DESCRIPTION.

**Returns**

- **g** (*ndarray*) – the fourier transform of f
- **freq** (*1darray*) – frequencies where f are evaluated

`lime.fft.fft2`(*f*, *dx=1*, *dy=1*)

customized FFT for 2D function input:

**f: 2d array,** input array

**Returns**

**1d array** frequencies

**g: 2d array** fourier transform of f

**Return type** freq

`lime.fft.ifft(f, x=None, axis=- 1)`

customized fourier transform of function f g = int dt f(t) * exp(i * freq * t) :returns: frequencies where f are evaluated

g: the fourier transform of f

**Return type** freq

### 3.2.12 lime.group module

Created on Wed Mar 23 21:30:04 2022

Some trivial computations for the group theory

@author: bing

### 3.2.13 lime.gwp module

Created on Fri Apr 1 21:22:56 2022

Using Gaussian basis set to propagate the nonadiabatic molecular dynamics

@author: Bing Gu

`class lime.gwp.GWP(x, p, a, phase, coeff)`

Bases: `object`

`lime.gwp.H()`

construct the hamiltonian matrix Kinetic energy operator can be computed exactly. Potential energy operator - approximation Nonadiabatic coupling -

`class lime.gwp.NAMD(bases, dim=1)`

Bases: `object`

**`kmat()`**

**`overlap()`**

construct overlap matrix from GWPs defined by {a,x,p}

**`run()`**

**`vmat()`**

`lime.gwp.kin_1d(aj, qj, pj, sj, ak, qk, pk, sk, am)`

kinetic energy matrix elements between two multidimensional GWPs

`lime.gwp.kin_me(gj, gk)`

kinetic energy matrix elements between two multidimensional GWPs

`lime.gwp.kmat(bset)`

kinetic energy matrix

`lime.gwp.overlap`(*gj*, *gk*)
overlap between two GWPs defined by {a,x,p}

`lime.gwp.overlap_1d`(*aj*, *x*, *px*, *sj*, *ak*, *y*, *py*, *sk*)
overlap between two 1D GWPs

### 3.2.14 lime.liouville module

Created on Wed Jun 19 20:05:24 2019

@author: binggu

@aims: superoperator algebra in Liouville space

`lime.liouville.sort`(*eigvals*, *eigvecs*)

### 3.2.15 lime.mol module

Created on Tue Jun 30 21:16:53 2020

@author: Bing Gu

Basic module for many-level systems

**class** `lime.mol.JahnTeller`(*E*, *omega*, *kappa*, *truncate=24*)
Bases: `lime.mol.LVC`

E otimes e Jahn-Teller model with two degenerate modes and two degenerate electronic states (+ the ground state)

`APES`(*x*, *y*, *B=None*)

`buildH`(*B=None*)
Calculate the vibronic Hamiltonian.

**Parameters** **nums** (*list of integers*) – size for the Fock space of each mode

**Returns** Hamiltonian

**Return type** 2d array

**class** `lime.mol.LVC`(*E*, *modes*)
Bases: `lime.mol.Mol`

linear vibronic coupling model in Fock space

`APES`(*x*)

`add_coupling`(*coupling*)
add additional coupling terms to the Hamiltonian such as Stark and Zeeman effects

**Parameters** **coupling** (*list, [[a, b], strength]*) – describe the coupling, a, b labels the electronic states

**Returns** updated H.

**Return type** ndarray

**buildH**()

Calculate the vibronic Hamiltonian.

**Parameters** **nums** (*list of integers*) – size for the Fock space of each mode

**Returns** Hamiltonian

**Return type** 2d array

**buildop**(*i*, *f=None*, *isherm=True*)

construct electronic operator

ket{f}ra{i}

**if isherm:** return ket{f}ra{i} + ket{i}ra{f}

**Parameters**

- **i** (*int*) – initial state.
- **f** (*int, optional*) – final state. Default None. If None, set f = i.
- **isherm** (*bool, optional*) – indicator of whether the returned matrix is Hermitian or not Default: True

**Returns** DESCRIPTION.

**Return type** 2darray

**calc_edip**()

**dpes**(*q*)

**groundstate**()

return the ground state

**Returns** DESCRIPTION.

**Return type** TYPE

**promote**(*A*, *which='el'*)

**rdm_el**(*psi*)

Compute the electronic reduced density matrix.

**Parameters** **psi** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

**vertical**(*n=1*)

generate the initial state created by vertical excitation

**Parameters** **n** (*int, optional*) – initially excited state. The default is 1.

**Returns** **psi** – DESCRIPTION.

**Return type** TYPE

**wavepacket_dynamics**(*method='RK4'*)

class lime.mol.**Mode**(*omega: float*, *couplings: list*, *truncate: int = 2*)

Bases: `object`

**couplings: list**

**omega: float**

**truncate: int = 2**

**class** lime.mol.**Mol**(*H*, *edip=None*, *edip_rms=None*, *gamma=None*)

Bases: object

**DQC**()

**ETPA**()

**PE**(*pump*, *probe*, *t2=0.0*, ***kwargs*)

alias for photon_echo

**PE2**(*omega1*, *omega2*, *t3=0.0*, ***kwargs*)

2D photon echo signal at -k1+k2+k3 Transforming t1 and t2 to frequency domain.

**Parameters**

- **pump** (*TYPE*) – DESCRIPTION.
- **probe** (*TYPE*) – DESCRIPTION.
- **t2** (*TYPE, optional*) – DESCRIPTION. The default is 0.0.
- ****kwargs** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

**TPA**()

**absorption**(*omegas*, *method='sos'*, ***kwargs*)

Linear absorption of the model.

**Parameters**

- **omegas** (*TYPE*) – DESCRIPTION.
- **method** (*TYPE, optional*) – DESCRIPTION. The default is 'SOS'.
- **normalize** (*TYPE, optional*) – DESCRIPTION. The default is True.

**Returns** DESCRIPTION.

**Return type** TYPE

**cars**(*shift*, *omega1*, *t2=0.0*, *plt_signal=False*, *fname=None*)

**driven_dynamics**(*pulse*, *dt=0.001*, *Nt=1*, *obs_ops=None*, *nout=1*, *t0=0.0*)

wavepacket dynamics in the presence of laser pulses

**Parameters**

- **pulse** (*TYPE*) – DESCRIPTION.
- **dt** (*TYPE, optional*) – DESCRIPTION. The default is 0.001.
- **Nt** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **obs_ops** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **nout** (*TYPE, optional*) – DESCRIPTION. The default is 1.

- **t0** (*float*) – initial time

**Return type** None.

**eigenenergies**()

**eigenstates**(*k=6*)

**Parameters** **k** (*integer*) – number of eigenstates to compute, < dim

**Returns**

- **eigvals** (*vector*)
- **eigvecs** (*2d array*)

**eigvals**()

**energy**(*psi*)

**evolve**(*psi0*, *dt=0.001*, *Nt=1*, *e_ops=None*, *nout=1*, *t0=0.0*, *pulse=None*)

quantum dynamics under time-independent Hamiltonian

**Parameters**

- **pulse** (*TYPE*) – DESCRIPTION.
- **dt** (*TYPE, optional*) – DESCRIPTION. The default is 0.001.
- **Nt** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **obs_ops** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **nout** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **t0** (*float*) – initial time

**Return type** None.

**get_dip**()

**get_dm**()

**get_edip**()

**get_nonhermH**()

**get_nonhermitianH**()

**groundstate**(*method='trivial'*)

**photon_echo**(*pump*, *probe*, *t2=0.0*, ***kwargs*)

2D photon echo signal at -k1+k2+k3

**Parameters**

- **pump** (*TYPE*) – DESCRIPTION.
- **probe** (*TYPE*) – DESCRIPTION.
- **t2** (*TYPE, optional*) – DESCRIPTION. The default is 0.0.
- ****kwargs** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

**quantum_dynamics**(*psi0*, *dt=0.001*, *Nt=1*, *obs_ops=None*, *nout=1*, *t0=0.0*)

quantum dynamics under time-independent hamiltonian

**Parameters**

- **pulse** (*TYPE*) – DESCRIPTION.
- **dt** (*TYPE, optional*) – DESCRIPTION. The default is 0.001.
- **Nt** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **obs_ops** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **nout** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **t0** (*float*) – initial time

**Return type** None.

**set_decay**(*gamma*)

Set the decay rate for all excited states.

**Parameters** **gamma** (*TYPE*) – DESCRIPTION.

**Return type** None.

**set_decay_for_all**(*gamma*)

Set the decay rate for all excited states.

**Parameters** **gamma** (*TYPE*) – DESCRIPTION.

**Return type** None.

**set_dephasing**(*gamma*)

set the pure dephasing rate for all coherences

**Parameters** **gamma** (*TYPE*) – DESCRIPTION.

**Return type** None.

**set_dip**(*dip*)

**set_dipole**(*dip*)

**set_edip**(*edip*, *pol=None*)

**set_lifetime**(*tau*)

**set_mdip**(*mdip*)

**tcl2**()

second-order time-convolutionless quantum master equation

**class** lime.mol.**Result**(*description=None*, *psi0=None*, *rho0=None*, *dt=None*, *Nt=None*, *times=None*, *t0=None*, *nout=None*)

Bases: object

**expect**()

**class** lime.mol.**SESolver**(*H=None*)

Bases: object

**correlation_2op_1t**()

**correlation_3op_1t**(*psi0*, *oplist*, *dt*, *Nt*)

<AB(t)C>

**Parameters**

- **psi0** –
- **oplist** –
- **dt** –
- **Nt** –

**correlation_3op_2t**(*psi0*, *oplist*, *dt*, *Nt*, *Ntau*)

<A(t)B(t+tau)C(t)> :param oplist: [a, b, c] :type oplist: list of arrays :param psi0: initial state :type psi0: array :param dt: :param nt: number of time steps for t :type nt: integer :param ntau: time steps for tau :type ntau: integer

**correlation_4op_1t**(*psi0*, *oplist*, *dt=0.005*, *Nt=1*)

<AB(t)C(t)D>

**Parameters**

- **psi0** –
- **oplist** –
- **dt** –
- **Nt** –

**correlation_4op_2t**(*psi0*, *oplist*, *dt=0.005*, *Nt=1*, *Ntau=1*)

**Parameters**

- **psi0** (*vector*) – initial state
- **oplist** (*list of arrays*) –

**propagator**(*dt*, *Nt*)

**run**(*psi0=None*, *dt=0.01*, *Nt=1*, *e_ops=None*, *nout=1*, *t0=0.0*, *edip=None*, *pulse=None*)

quantum dynamics under time-independent and time-dependent Hamiltonian

**Parameters**

- **psi0** (*1d array*) – initial state
- **pulse** (*callable/list of callable*) – external laser pulses
- **dt** (*TYPE, optional*) – DESCRIPTION. The default is 0.001.
- **Nt** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **obs_ops** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **nout** (*TYPE, optional*) – DESCRIPTION. The default is 1.
- **t0** (*float*) – initial time. The default is 0.

**Return type** None.

lime.mol.**driven_dynamics**(*H*, *psi0*, *dt=0.001*, *Nt=1*, *e_ops=None*, *nout=1*, *t0=0.0*, *return_result=True*, *sparse=True*)

Laser-driven dynamics in the presence of laser pulses

**Parameters**

- **ham** (*2d array*) – Hamiltonian of the molecule
- **dip** (*TYPE*) – transition dipole moment
- **psi0** (*1d array*) – initial wavefunction
- **pulse** (*TYPE*) – laser pulse
- **dt** (*TYPE*) – time step.
- **Nt** (*TYPE*) – timesteps.
- **e_ops** (*list*) – observable operators to compute
- **nout** (*int*) – Store data every nout steps

**Return type** None.

lime.mol.**mls**(*dim=3*)

lime.mol.**polar**(*x*, *y*)

lime.mol.**read_input**(*fname_e*, *fname_edip*, *g_included=True*)

Read input data from quantum chemistry output.

**Parameters**

- **fname_e** (*str*) – filename for the energy levels.
- **fname_edip** (*list*) – filenames for the electric dipole moment.
- **g_included** (*TYPE, optional*) – DESCRIPTION. The default is True.

**Returns** **mol** – DESCRIPTION.

**Return type** TYPE

lime.mol.**test_sesolver**()

### 3.2.16 lime.noise module

Created on Fri Aug 3 16:27:19 2018

@author: Bing Gu

Colored Gaussian noise

lime.noise.**cnoise**(*nstep*, *nsample*, *dt=0.001*, *tau=0.0025*, *ave=0.0*, *D=0.0025*)

store several series of Gaussian noise values in array EPS.

This is based on the algorithm in R. F. Fox et al. Phys. Rev. A 38, 11 (1988). The generated noise satisfy <eps(t) eps(s)> = D/tau * exp(-|t-s|/tau), and the initial distribution is Gaussian N(0, sigma) with sigma**2 = D/tau

**INPUT:** dt: timestep, default 0.001 tau: correlation time, default 0.0025 ave: average value, default 0.0 D: strength of the noise, default 0.0025

**OUTPUT:** eps: eps[nstep, nsample] colored Gaussian noise

lime.noise.**corr**(*eps*)

calculate the autocorrelation function in variable MEAN.

lime.noise.**cross_corr**(*a*, *b*)

calculate the cross-correlation function in variable MEAN.

### 3.2.17 lime.optics module

Created on Tue Mar 26 17:26:02 2019

@author: binggu

`class lime.optics.Analyser(E, t)`

Bases: `object`

`FROG(w=None, use_fft=False)`

`plot_spectrogram()`

`spectrogram()`

`class lime.optics.Biphoton(omegap, bw, Te, p=None, q=None, phase_matching='sinc')`

Bases: `object`

`bandwidth(which='signal')`

Compute the bandwidth of the signal/idler mode

**Parameters** `which` (`TYPE, optional`) – DESCRIPTION. The default is 'signal'.

**Return type** None.

`detect()`

two-photon detection amplitude in a temporal grid defined by the spectral grid.

**Returns**

- **t1** (*1d array*)
- **t2** (*1d array*)
- **d** (*detection amplitude in the temporal grid (t1, t2)*)

`detect_is()`

`detect_si()`

`g2()`

`get_jsa()`

**Returns** **jsa** – joint spectral amplitude

**Return type** array

`get_jta()`

Compute the joint temporal amplitude J(ts, ti) over a temporal meshgrid.

**Returns**

- **ts** (*1d array*) – signal time grid
- **ti** (*1d array*) – idler temporal grid
- **jta** (*2d array*) – joint temporal amplitude

`jta(ts, ti)`

`plt_jsa(xlabel=None, ylabel=None, fname=None)`

**pump**(*bandwidth*)
pump pulse envelope :param bandwidth:

**rdm**(*which='signal'*)

**set_grid**(*p, q*)

class lime.optics.**ChirpedPulse**(*omegac=0.11024710050125681, tau=206.70686668237954, tc=0, amplitude=0.001, cep=0.0, beta=0*)

Bases: `lime.optics.Pulse`

**efield**(*t*)

**Parameters** **t** (*TYPE*) – DESCRIPTION.

**Return type** electric field at time t.

**spectrum**(*omega*)
Fourier transform of the Gaussian pulse

class lime.optics.**GaussianPulse**(*omegac=0.11024710050125681, tau=206.70686668237954, tc=0, delay=0.0, amplitude=0.001, cep=0.0, beta=0*)

Bases: `object`

**efield**(*t*)

**Parameters** **t** (*TYPE*) – DESCRIPTION.

**Return type** electric field at time t.

**envelop**(*t*)

**field**(*t*)
electric field

**plt_efield**()

**spectrogram**(*efield*)

**spectrum**(*omega*)
Fourier transform of the Gaussian pulse

class lime.optics.**Pulse**(*omegac=0.11024710050125681, tau=206.70686668237954, tc=0, delay=0.0, amplitude=0.001, cep=0.0, beta=0*)

Bases: `object`

**efield**(*t, return_complex=False*)

**Parameters** **t** (*TYPE*) – time.

**Returns** complex-valued electric field at time t. Take the real part for the electric field.

**Return type** complex

**envelop**(*t*)

**field**(*t*)
electric field

**plt_efield**()

**spectrogram**(*efield*)

**spectrum**(*omega*)

Fourier transform of the Gaussian pulse

lime.optics.**field_to_intensity**(*E*)

lime.optics.**fwhm_to_std**(*fwhm*)

lime.optics.**hom**(*p*, *q*, *f*, *tau*)

HOM coincidence probability

**Parameters**

- **p** –
- **q** –
- **f** –
- **tau** –
- **method** (*str*) – "brute": directly integrating the JSA over the frequency grid "schmidt": compute the signal using the Schmidt modes of the

  entangled light
- **nmodes** –

**Returns** **prob** – coincidence probability

**Return type** 1d array

lime.optics.**hom_schmidt**(*p*, *q*, *f*, *method='rdm'*, *nmodes=5*)

HOM signal with Schmidt modes

**Parameters**

- **p** –
- **q** –
- **f** –
- **nmodes** –

lime.optics.**intensity_to_field**(*I*)

transform intensity to electric field

E = sqrt(

rac{2I}{c epsilon_0})

**I** [TYPE] intensity, in units of W/m^2

**None.** electric field in atomic units

lime.optics.**jta**(*t2*, *t1*, *omegap*, *sigmap*, *Te*)

Analytical form for the joint temporal amplitude for SPDC type-II two-photon state.

Note that two single-photon electric field prefactors are neglected.

**Parameters**

- **t2** (*TYPE*) – DESCRIPTION.

- **t1** (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.optics.rdm(f, dx=1, dy=1, which='x')`

Compute the reduced density matrix by tracing out the other dof for a 2D wavefunction

**Parameters**

- **f** (*2D array*) – 2D wavefunction
- **dx** (*float, optional*) – DESCRIPTION. The default is 1.
- **dy** (*float, optional*) – DESCRIPTION. The default is 1.
- **which** (*str*) – indicator which rdm is required. Default is 'x'.

**Returns** **rho1** – Reduced density matrix

**Return type** TYPE

`lime.optics.schmidt_decompose(f, dp, dq, nmodes=5, method='rdm')`

kernel method f: 2D array,

input function to be decomposed

**nmodes: int** number of modes to be kept

**method: str** rdm or svd

`lime.optics.std_to_fwhm(tau)`

Transformation between standard deviation to FWHM for a Gaussian pulse

**Parameters** **tau** (*float*) – std

**Returns** FWHM.

**Return type** float

### 3.2.18 lime.oqs module

### 3.2.19 lime.phys module

`class lime.phys.HarmonicOscillator(omega, mass=1, x0=0)`

Bases: `object`

basic class for harmonic oscillator

`eigenstate(x, n=0)`

`potential(x)`

`class lime.phys.Morse(D, a, re, mass=1)`

Bases: `object`

basic class for Morse oscillator

`eigenstate(x, n=0)`

`eigval`(*n*)

Given an (integer) quantum number v, return the vibrational energy.

**Parameters** `n` (*TYPE*) – DESCRIPTION.

**Return type** None.

`potential`(*x*)

`lime.phys.anticomm`(*A*, *B*)

`lime.phys.anticommutator`(*A*, *B*)

`lime.phys.basis`(*N*, *j*)

**Parameters**

- `N` (*int*) – Size of Hilbert space for a multi-level system.
- `j` (*int*) – The j-th basis function.

**Returns** j-th basis function for the Hilbert space.

**Return type** 1d complex array

`lime.phys.basis_transform`(*A*, *v*)

transformation rule: A_{ab} = <a|i><i|A|j><j|b> = Anew = v^dag A v input:

A: matrix of operator A in old basis v: basis transformation matrix

**output:** Anew: matrix A in the new basis

`lime.phys.boson`(*omega*, *n*, *ZPE=False*)

`lime.phys.coh_op`(*j*, *i*, *d*)

return a matrix representing the coherence j, i

**Parameters**

- `j` (*TYPE*) – DESCRIPTION.
- `i` (*TYPE*) – DESCRIPTION.
- `d` (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.phys.comm`(*A*, *B*)

`lime.phys.commutator`(*A*, *B*)

`lime.phys.coth`(*x*)

`lime.phys.dag`(*a*)

`lime.phys.dagger`(*a*)

`lime.phys.destroy`(*N*)

Annihilation operator for bosons.

**Parameters** `N` (*int*) – Size of Hilbert space.

**Return type** 2d array complex

`lime.phys.`**`driven_dissipative_dynamics`**(*ham*, *dip*, *rho0*, *pulse*, *dt=0.001*, *Nt=1*, *obs_ops=None*, *nout=1*)

Laser-driven dynamics in the presence of laser pulses

**Parameters**

- **ham** (*2d array*) – Hamiltonian of the molecule
- **dip** (*TYPE*) – transition dipole moment
- **rho0** (*2d array complex*) – initial density matrix
- **pulse** (*TYPE*) – laser pulse
- **dt** (*float*) – DESCRIPTION.
- **Nt** (*TYPE*) – DESCRIPTION.
- **obs_ops** (*list*) – observable operators to compute
- **nout** (*int*) – Store data every nout steps

**Return type** None.

`lime.phys.`**`driven_dynamics`**(*ham*, *dip*, *psi0*, *pulse*, *dt=0.001*, *Nt=1*, *obs_ops=None*, *nout=1*, *t0=0.0*)

Laser-driven dynamics in the presence of laser pulses

**Parameters**

- **ham** (*2d array*) – Hamiltonian of the molecule
- **dip** (*TYPE*) – transition dipole moment
- **psi0** (*1d array*) – initial wavefunction
- **pulse** (*TYPE*) – laser pulse
- **dt** (*TYPE*) – time step.
- **Nt** (*TYPE*) – timesteps.
- **obs_ops** (*list*) – observable operators to compute
- **nout** (*int*) – Store data every nout steps

**Return type** None.

`lime.phys.`**`eig_asymm`**(*h*)

Diagonalize a real, *asymmetrix* matrix and return sorted results.

Return the eigenvalues and eigenvectors (column matrix) sorted from lowest to highest eigenvalue.

`lime.phys.`**`expm`**(*A*, *t*, *method='EOM'*)

**exponentiate a matrix at t** U(t) = e^{A t}

**Parameters**

- **A** (*TYPE*) – DESCRIPTION.
- **t** (*float or list*) – times
- **method** (*TYPE, optional*) – DESCRIPTION. The default is 'EOM'.

  **EOM: equation of motion approach.** d/dt U(t) = A U(t) This can be generalized for time-dependent Hamiltonians A(t)

  diagonalization: diagonalize A

for Hermitian matrices only, this is prefered

**Returns** **Ulist** – DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`fermi`**(*E*, *Ef=0.0*, *T=0.0001*)

Fermi-Dirac distribution function INPUT:

E : Energy Ef : Fermi energy T : temperture (in units of energy, i.e., kT)

**OUTPUT:** f(E): Fermi-Dirac distribution function at energy E

`lime.phys.`**`fftfreq`**(*times*)

get the spectral range corresponding to the temporal grid (a.u.)

**Parameters** **`times`** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`gaussian`**(*x*, *sigma=1.0*)

normalized Gaussian distribution

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`get_index`**(*array*, *value*)

get the index of element in array closest to value

`lime.phys.`**`gwp`**(*x*, *sigma=1.0*, *x0=0.0*, *p0=0.0*)

complex Gaussian wavepacket

**Parameters**

- **`x`** (*TYPE*) – DESCRIPTION.
- **`sigma`** (*TYPE, optional*) – DESCRIPTION. The default is 1..
- **`x0`** (*TYPE, optional*) – DESCRIPTION. The default is 0..
- **`p0`** (*TYPE, optional*) – DESCRIPTION. The default is 0..

**Returns** **psi** – DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`gwp2`**(*x*, *y*, *sigma=array([[1.0, 0.0], [0.0, 1.0]])*, *xc=[0, 0]*, *kc=[0, 0]*)

generate a 2D Gaussian wavepacket in grid :param x0: float, mean value of gaussian wavepacket along x :param y0: float, mean value of gaussian wavepacket along y :param sigma: float array, covariance matrix with 2X2 dimension :param kx0: float, initial momentum along x :param ky0: float, initial momentum along y :return: float 2darray, the gaussian distribution in 2D grid

`lime.phys.`**`gwp_k`**(*k*, *sigma*, *x0*, *k0*)

analytical fourier transform of gauss_x(x), above

`lime.phys.`**`ham_ho`**(*freq*, *n*, *ZPE=False*)

Hamiltonian for harmonic oscilator

**input:** freq: fundemental frequency in units of Energy n : size of matrix ZPE: boolean, if ZPE is included in the Hamiltonian

**output:** h: hamiltonian of the harmonic oscilator

`lime.phys.heaviside(x)`

`lime.phys.hilbert_dist(A, B)`

Returns the Hilbert-Schmidt distance between two density matrices A & B.

**Parameters**

- **A** (*qobj*) – Density matrix or state vector.
- **B** (*qobj*) – Density matrix or state vector with same dimensions as A.

**Returns** **dist** – Hilbert-Schmidt distance between density matrices.

**Return type** float

### Notes

See V. Vedral and M. B. Plenio, Phys. Rev. A 57, 1619 (1998).

`lime.phys.interval(x)`

`lime.phys.is_positive_def(A)`

`lime.phys.isdiag(M)`

Check if a matrix is diagonal.

**Parameters** **M** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.isherm(a)`

`lime.phys.jump(f, i, dim=2, isherm=True)`

`lime.phys.ket2dm(psi)`

convert a ket into a density matrix

**Parameters** **psi** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.ldo(b, A)`

linear differential operator Ab

**Parameters**

- **b** (*TYPE*) – DESCRIPTION.
- **A** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.lindbladian(l, rho)`

lindblad superoperator: l rho l^dag - 1/2 * {l^dag l, rho} l is the operator corresponding to the disired physical process e.g. l = a, for the cavity decay and l = sm for polarization decay

`lime.phys.`**`liouvillian`**(*rho*, *H*, *c_ops*)
lindblad quantum master eqution

`lime.phys.`**`lorentzian`**(*x*, *width=1.0*)

**Parameters**

- **x** (*TYPE*) – DESCRIPTION.
- **x0** (*float*) – center of the Lorentzian
- **width** (*float*) – Half-wdith half-maximum

**Return type** None.

`lime.phys.`**`lowering`**(*dims=2*)

`lime.phys.`**`meshgrid`**(**args*)
fix the indexing of the Numpy meshgrid

**Parameters** ***args** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`morse`**(*r*, *D*, *a*, *re*)

**Morse potential** D * (1. - e^{-a * (r - r_e)})**2

**Parameters**

- **r** (*float/1darray*) – DESCRIPTION.
- **D** (*float*) – well depth
- **a** (*TYPE*) – 'width' of the potential.
- **re** (*TYPE*) – equilibrium bond distance

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`multi_spin`**(*onsite*, *nsites*)
construct the hamiltonian for a multi-spin system params:

onsite: array, transition energy for each spin nsites: number of spins

`lime.phys.`**`multiboson`**(*omega*, *nmodes*, *J=0*, *truncate=2*)
construct the hamiltonian for a multi-spin system

**Parameters**

- **omegas** (*1D array*) – resonance frequenies of the boson modes
- **nmodes** (*integer*) – number of boson modes
- **J** (*float*) – hopping constant
- **truncation** (*TYPE, optional*) – DESCRIPTION. The default is 2.

**Returns**

- **ham** (*TYPE*) – DESCRIPTION.
- **lower** (*TYPE*) – DESCRIPTION.

`lime.phys.`**`multimode`**(*omegas*, *nmodes*, *J=0*, *truncate=2*)

construct the direct tensor-product Hamiltonian for a multi-mode system

**Parameters**

- **omegas** (*1D array*) – resonance frequenies of the boson modes
- **nmodes** (*integer*) – number of boson modes
- **J** (*float*) – nearest-neighbour hopping constant
- **truncate** (*list*) – size of Fock space for each mode

**Returns**

- **ham** (*TYPE*) – DESCRIPTION.
- **xs** (*list*) – position operators in the composite space for each mode

`lime.phys.`**`norm`**(*psi*, *dx=1*)

normalization of the wavefunction

N = int dx psi^*(x) psi(x)

**Parameters** **psi** (*1d array, complex*) – DESCRIPTION.

**Return type** float, L2 norm

`lime.phys.`**`norm2`**(*f*, *dx=1*, *dy=1*)

L2 norm of the 2D array f

**Parameters**

- **f** (*TYPE*) – DESCRIPTION.
- **dx** (*TYPE*) – DESCRIPTION.
- **dy** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.`**`obs`**(*psi*, *a*)

**Parameters**

- **psi** (*1d array*) – wavefunction.
- **a** (*2d array*) – operator a.

**Returns** Expectation of operator a.

**Return type** complex

`lime.phys.`**`obs_dm`**(*rho*, *d*)

observables for operator d

`lime.phys.`**`pauli`**()

`lime.phys.`**`pdf_normal`**(*x*, *mu=0*, *sigma=1.0*)

`lime.phys.`**`project`**(*P*, *a*)

reduce the representation of operators to a subspace defined by the projection operator P

**Parameters**

- **P** (*TYPE*) – DESCRIPTION.

- `a` (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.phys.ptrace`(*rho*, *dims*, *which='B'*)

partial trace of subsystems in a density matrix defined in a composite space

**Parameters**

- `rho` (*ndarray*) – DESCRIPTION.
- `which` (*TYPE, optional*) – DESCRIPTION. The default is 'B'.

**Returns** **rhoA** – DESCRIPTION.

**Return type** TYPE

`lime.phys.quadrature`(*n*)

Quadrature operator of a photon mode X = (a + a+)/sqrt{2}

**Parameters** `n` (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.quantum_dynamics`(*ham*, *psi0*, *dt=0.001*, *Nt=1*, *obs_ops=None*, *nout=1*, *t0=0.0*, *output='obs.dat'*)

Laser-driven dynamics in the presence of laser pulses

**Parameters**

- `ham` (*2d array*) – Hamiltonian of the molecule
- `psi0` (*1d array*) – initial wavefunction
- `dt` (*float*) – time step.
- `Nt` (*int*) – timesteps.
- `obs_ops` (*list*) – observable operators to compute
- `nout` (*int*) – Store data every nout steps

**Return type** None.

`lime.phys.raising`(*dims=2*)

raising operator for spin-1/2 :param dims: Hilbert space dimension :type dims: integer

**Returns** **sp** – raising operator

**Return type** 2x2 array

`lime.phys.rect`(*x*)

`lime.phys.resolvent`(*omega*, *Ulist*, *dt*)

compute the resolvent 1/(omega - H) from the Fourier transform of the propagator omega: float

frequency to evaluate the resolvent

**Ulist: list of matrices** propagators

dt: time-step used in the computation of U

`lime.phys.rgwp(x, x0=0.0, sigma=1.0)`
real Gaussian wavepacket

**Parameters**

- **x** (*TYPE*) – DESCRIPTION.
- **x0** (*float*) – central position
- **sigma** (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.phys.rk4(rho, fun, dt, *args)`
Runge-Kutta method

`lime.phys.rotate(angle)`

`lime.phys.sigmax()`

`lime.phys.sigmay()`

`lime.phys.sigmaz()`

`lime.phys.sinc(x)`
sinc(x) = sin(x)/x

**Parameters** **x** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.phys.sort(eigvals, eigvecs)`
sort eigenvalues and eigenvectors from low to high

**Parameters**

- **eigvals** (*TYPE*) – DESCRIPTION.
- **eigvecs** (*TYPE*) – DESCRIPTION.

**Returns**

- **eigvals** (*TYPE*) – DESCRIPTION.
- **eigvecs** (*TYPE*) – DESCRIPTION.

`lime.phys.tdse(wf, h)`

`lime.phys.tensor(*args)`
Calculates the tensor product of input operators.

Build from QuTip.

**Parameters** **args** (*array_like*) – `list` or `array` of quantum objects for tensor product.

`lime.phys.tensor_power(a, n: int)`
kron(a, kron(a, …))

`lime.phys.thermal_dm(n, u)`
return the thermal density matrix for a boson n: integer

dimension of the Fock space

**u: float** reduced temperature, omega/k_B T

`lime.phys.tracedist(A, B)`

Calculates the trace distance between two density matrices.. See: Nielsen & Chuang, "Quantum Computation and Quantum Information"

Parameters ———-!= A : 2D array (N,N)

Density matrix or state vector.

**B** [2D array (N,N)] Density matrix or state vector with same dimensions as A.

**tol** [float] Tolerance used by sparse eigensolver, if used. (0=Machine precision)

**sparse** [{False, True}] Use sparse eigensolver.

**Returns tracedist** – Trace distance between A and B.

**Return type** float

#### Examples

```
>>> x=fock_dm(5,3)
>>> y=coherent_dm(5,1)
>>> tracedist(x,y)
0.9705143161472971
```

`lime.phys.transform(A, v)`

transformation rule: A_{ab} = <a|i><i|A|j><j|b> = Anew = v^dag A v input:

A: matrix of operator A in old basis v: basis transformation matrix

**output:** Anew: matrix A in the new basis

### 3.2.20 lime.qnm module

### 3.2.21 lime.quadrature module

Created on Wed Oct 27 15:43:40 2021

@author: Bing

Computing multidimensional Gaussian integral with Gaussian-Hermite quadrature

`class lime.quadrature.Quadrature`

Bases: `object`

`gauss_hermite(n, mu=None, sigma=None)`

Compute int f(x) exp(-x^2) dx = sum_{i=0}^n w[i] * f(x[i]) using Gaussian-Hermite quadrature

**Parameters**

- **n** (*TYPE*) – DESCRIPTION.
- **mu** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **sigma** (*TYPE, optional*) – DESCRIPTION. The default is None.

**Returns**

- *TYPE* – DESCRIPTION.
- **w** (*TYPE*) – DESCRIPTION.

`integrate(f)`

`lime.quadrature.covfunc(x)`

`lime.quadrature.f(x)`

`lime.quadrature.gauss_hermite_quadrature(n, mu=None, sigma=None)`

Compute int f(x) exp(-x^2) dx = sum_{i=0}^n w[i] * f(x[i]) using Gaussian-Hermite quadrature

**Parameters**

- **n** (*TYPE*) – DESCRIPTION.
- **mu** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **sigma** (*TYPE, optional*) – DESCRIPTION. The default is None.

**Returns**

- *TYPE* – DESCRIPTION.
- **w** (*TYPE*) – DESCRIPTION.

### 3.2.22 lime.spin_boson module

`class lime.spin_boson.Spin_boson(beta, omegac=1.0, reorg=2.0)`

Bases: `object`

`pure_dephasing(t)`

**Compute the exact decoherence function for the pure-dephasing spin_boson** model H = eps0* sigmaz/2 + sigmaz *(g_k a^dag_k + g_k^* a_k) + omega_k a_k^dag a_k

**INPUT:** beta: inverse temperature omegac: cutoff frequency reorg: reorganization energy

**OUTPUT: Decoherence function at time t** ln Phi(t) = - int_0^infty J(omega)coth(beta*omega/2) domega J(omega) = sum_k |g_k|^2 delta(omega - omega_k)

`spectral_density(omega, omegac, reorg, name)`

Spectral density

`update_temp(beta)`

update inverse temperature INPUT:

beta: 1/(k_B T) in units of Energy

### 3.2.23 lime.style module

lime.style.**color_code**(*x*, *y*, *z*, *fig*, *ax*, *cbar=False*)

lime.style.**curve**(*x*, *y*, ***kwargs*)
simple 1D curve plot

**Parameters**

- **x** (*TYPE*) – DESCRIPTION.
- **y** (*TYPE*) – DESCRIPTION.

**Return type** None.

lime.style.**export**(*x*, *y*, *z*, *fname='output.dat'*, *fmt='gnuplot'*)
export 3D data to gnuplot format

**Parameters**

- **x** (*TYPE*) – DESCRIPTION.
- **y** (*TYPE*) – DESCRIPTION.
- **z** (*TYPE*) – DESCRIPTION.
- **fname** (*TYPE, optional*) – DESCRIPTION. The default is 'output.dat'.
- **fmt** (*str, optional*) – The target format. The default is 'gnuplot'.

**Return type** None.

lime.style.**imshow**(*x*, *y*, *f*, *vmin=None*, *vmax=None*, *ticks=None*, *output='output.pdf'*, *xlabel='X'*, *ylabel='Y'*, *diverge=False*, *cmap='viridis'*, ***kwargs*)

**Parameters**

- **f** (*2D array*) – array to be plotted.
- **extent** (*list [xmin, xmax, ymin, ymax]*) –

**Return type** Save a fig in the current directory.

lime.style.**level_scheme**(*E*, *ylim=None*, *fname=None*)
plot the energy levels :param E:

lime.style.**matplot**(*x*, *y*, *f*, *vmin=None*, *vmax=None*, *ticks=None*, *output='output.pdf'*, *xlabel='X'*, *ylabel='Y'*, *diverge=False*, *cmap='viridis'*, ***kwargs*)

**Parameters**

- **f** (*2D array*) – array to be plotted.
- **extent** (*list [xmin, xmax, ymin, ymax]*) –

**Returns**

- *Save a fig in the current directory.*
- *To be deprecated. Please use imshow.*

lime.style.**plot_surface**(*x*, *y*, *surface*)

lime.style.**plot_surfaces**(*x*, *y*, *surfaces*) `lime.style.set_style(fontsize=12)`

`lime.style.subplots(nrows=1, ncols=1, figsize=(4, 3), sharex=True, sharey=True, **kwargs)`

`lime.style.surf(f, x, y, fname='output.png', xlabel='X', ylabel='Y', zlabel='Z', title=None, method='matplotlib')`

`lime.style.test_level_scheme()`

`lime.style.two_scales(x, yl, yr, xlabel=None, ylabels=None, xlim=None, yllim=None, yrlim=None, yticks=None, fname='output.pdf')`

### 3.2.24 lime.superoperator module

Created on Thu Jun 25 22:01:00 2020

@author: Bing

Modules for computing signals with superoperator formalism in Liouville space

Instead of performing open quantum dynamics, the Liouvillian is directly diagonalized

**Possible improvements:**

1. merge the Qobj class with QUTIP

**class** `lime.superoperator.Lindblad_solver(H, c_ops=None)`

Bases: `object`

**correlation_2op_1t**(*rho0, ops, tlist*)

Compute <A(t)B> by diagonalizing the Liouvillian.

**Returns** correlation function.

**Return type** 1D array.

**correlation_2op_1w**(*rho0, ops, w*)

Compute S(w) = <A(w)B> = int_0^infty <A(t)B> exp(iwt) dt by diagonalizing the Liouvillian.

**Returns** correlation function.

**Return type** 1D array.

**correlation_3op_1t**(*rho0, ops, t*)

Compute <A(t)B> by diagonalizing the Liouvillian.

**Returns** correlation function.

**Return type** 1D array.

**correlation_3op_1w**(*rho0, ops, w*)

Compute <A(t)B> by diagonalizing the Liouvillian.

**Returns** correlation function.

**Return type** 1D array.

**correlation_3op_2t**(*rho0, ops, tlist, taulist, k=None*)

Compute <A(t)B(t+tau)C(t)> by diagonalizing the Liouvillian.

**Returns** correlation function.

**Return type** 1D array.

**correlation_4op_2t**(*rho0*, *ops*, *tlist*, *taulist*, *k=None*)
Compute <A(t)B(t+tau)C(t)> by diagonalizing the Liouvillian.

**Returns** correlation function.

**Return type** 1D array.

**eigenstates**(*k=None*)

**evolve**(*rho0*, *tlist*, *e_ops*)

**liouvillian**()

class lime.superoperator.**Qobj**(*data=None*, *dims=None*)
Bases: `object`

**conjugate**()

**dot**(*b*)

**to_linblad**(*gamma=1.0*)

**to_super**(*type='commutator'*)

**to_vector**()

lime.superoperator.**absorption**(*mol*, *omegas*, *c_ops*)
superoperator formalism for absorption spectrum

**Parameters**

- **mol** (*TYPE*) – DESCRIPTION.
- **omegas** (*vector*) – detection window of the spectrum
- **c_ops** (*TYPE*) – list of collapse operators

**Return type** None.

lime.superoperator.**cdot**(*a*, *b*)
matrix product of a.H.dot(b)

**Parameters**

- **a** (*TYPE*) – DESCRIPTION.
- **b** (*TYPE*) – DESCRIPTION.

**Return type** None.

lime.superoperator.**dm2vec**(*rho*)
transform an operator/density matrix to a vector in Liouville space

**Parameters** **A** (*TYPE*) – DESCRIPTION.

**Return type** None.

lime.superoperator.**kraus**(*a*)
Kraus superoperator a rho a^dag = a^dag_R a_L

**Parameters** **a** (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`lime.superoperator.left(a)`

`lime.superoperator.lindblad_dissipator(l)`

`lime.superoperator.liouville_space(N)`

constuct liouville space out of N Hilbert space basis |ij>

`lime.superoperator.liouvillian(H, c_ops)`

Construct the Liouvillian out of the Hamiltonian and collapse operators

**Parameters**

- **H** (*TYPE*) – DESCRIPTION.
- **c_ops** (*TYPE*) – DESCRIPTION.

**Returns** l – DESCRIPTION.

**Return type** TYPE

`lime.superoperator.mat2vec_index(N, i, j)`

Convert a matrix index pair to a vector index that is compatible with the matrix to vector rearrangement done by the mat2vec function.

From Qutip.

`lime.superoperator.obs(rho, a)`

`lime.superoperator.op2sop(a, kind='commutator')`

`lime.superoperator.operator_to_superoperator(a, kind='commutator')`

promote an operator/density matrix to an superoperator in Liouville space

**Parameters** **A** (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.superoperator.operator_to_vector(rho)`

transform an operator/density matrix to an superoperator in Liouville space

**Parameters** **A** (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.superoperator.resolvent(omega, L)`

Resolvent of the Lindblad quantum master equation

**Parameters**

- **omega** (*TYPE*) – DESCRIPTION.
- **L** (*2d array*) – full liouvillian

**Return type** None.

`lime.superoperator.right(a)`

`lime.superoperator.sort(eigvals, eigvecs)`

`lime.superoperator.to_super(a, kind='commutator')`

`lime.superoperator.trace(rho)`

`lime.superoperator.vec2mat_index(N, I)`

Convert a vector index to a matrix index pair that is compatible with the vector to matrix rearrangement done by the vec2mat function.

From Qutip.

### 3.2.25 lime.susceptibility module

### 3.2.26 lime.units module

`class lime.units.AtomicUnits`

Bases: `object`

### 3.2.27 lime.wigner module

Created on Sat Jul 24 22:23:15 2021

@author: bing

@Source: https://www.frank-zalkow.de/en/the-wigner-ville-distribution-with-python.html

`lime.wigner.nextpow2(p)`

`lime.wigner.spectrogram(x, d=1)`

Wigner transform of an input signal with FFT.

W(w, t) = int d au x(t + au/2) x^*(t - au/2) e^{i w tau}

**Parameters**

- **x** (*1d array*) – The time-domain signal.
- **d** (*TYPE, optional*) – DESCRIPTION. The default is 1.

**Returns**

- *TYPE* – spectrogram in (f, t)
- **freqs** (*1d array*) – sample frequencies.

`lime.wigner.wigner(x, d=1)`

Wigner transform of an input signal with FFT. W(t, w) = int dtau x(t + tau/2) x^*(t - tau/2) e^{i w tau}

**Parameters** **x** (*TYPE*) – DESCRIPTION.

**Returns**

- *TYPE* – DESCRIPTION.
- *TYPE* – DESCRIPTION.
- *TYPE* – DESCRIPTION.

`lime.wigner.wvd(audioFile, t=None, N=None, trace=0, make_analytic=True)`

### 3.2.28 lime.wpd module

Created on Mon Jan 4 23:44:15 2021

Wave packet dynamics solver for wavepacket dynamics with N vibrational modes (N = 1 ,2)

For linear coordinates, use SPO method For curvilinear coordinates, use RK4 method

@author: Bing Gu

`lime.wpd.KEO`(*psi*, *kx*, *ky*, *G*)

compute kinetic energy operator K * psi

**Parameters**

- **psi** (*TYPE*) – DESCRIPTION.
- **dt** (*TYPE*) – DESCRIPTION.

**Return type** None.

`lime.wpd.PEO`(*psi*, *v*)

V |psi> :param dt: float

time step

**Parameters**

- **v_2d** – float array the two electronic states potential operator in grid basis
- **psi_grid** – list the two-electronic-states vibrational state in grid basis

**Returns** psi_grid(update): list the two-electronic-states vibrational state in grid basis after being half time step forward

`lime.wpd.S0`(*x*, *y*)

`lime.wpd.S1`(*x*, *y*)

**class** `lime.wpd.SPO`(*x*, *mass=1*, *ns=2*)

Bases: `object`

`build`(*dt*)

`k_evolve`(*psi_x*)

one time step for exp(-i * K * dt)

`run`(*psi0*, *dt*, *Nt=1*, *t0=0*, *nout=1*)

Time-dependent Schrodinger Equation for wavepackets on a single PES.

**Parameters**

- **psi0** (*1d array, complex*) – initial wavepacket
- **t0** (*float*) – initial time
- **dt** (*float*) – the small time interval over which to integrate
- **nt** (*float, optional*) – the number of intervals to compute. The total change in time at the end of this method will be dt * Nsteps. default is N = 1

`set_grid`(*xmin=- 1*, *xmax=1*, *npts=32*) `set_potential`(*potential*)

`x_evolve`(*psi*)

one time step for exp(-i * V * dt)

**Parameters** `psi` (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

`x_evolve_half`(*psi*)

one time step for exp(-i * V * dt)

**Parameters** `psi` (*TYPE*) – DESCRIPTION.

**Returns** DESCRIPTION.

**Return type** TYPE

**class** `lime.wpd.SPO2`(*x*, *y*, *masses*, *ns=2*, *coords='linear'*, *G=None*)

Bases: `object`

second-order split-operator method for nonadiabatic wavepacket dynamics in the diabatic representation with two-dimensional nuclear coordinate

For time-independent Hamiltonian

e^{-i H Delta t} = e^{- i V Delta t/2} e^{-i K Delta t} e^{-iVDelta t/2}

**For time-dependent H,** TBI

`build`(*dt*, *inertia=None*)

Setup the propagators appearing in the split-operator method.

For the kinetic energy operator with Jacobi coordinates

K =

rac{p_r^2}{2mu} + rac{1}{I(r)} p_ heta^2

Since the two KEOs for each dof do not commute, it has to be factorized as

e^{-i K delta t} = e{-i K_1 delta t} e^{- i K_2 delta t}

where $p_ heta = -i pa_ heta$ is the momentum operator.

**dt** [TYPE] DESCRIPTION.

**intertia: func** moment of inertia, only used for Jocabi coordinates.

None.

`plot_surface`(*style='2D'*)

`plt_wp`(*psilist*, ***kwargs*)

`population`(*psi*)

return the electronic populations

**Parameters** `psi` (*TYPE*) – DESCRIPTION.

**Return type** None.

**run**(*psi0*, *e_ops=[]*, *dt=0.01*, *Nt=1*, *t0=0.0*, *nout=1*, *return_states=True*)

**setG**(*G*)

**set_DPES**(*surfaces*, *diabatic_couplings*, *abc=False*)

set the diabatic PES and diabatic couplings

**Parameters**

- **surfaces** (*TYPE*) – DESCRIPTION.
- **diabatic_couplings** (*TYPE*) – DESCRIPTION.
- **abc** (`boolean, optional`) – indicator of whether using absorbing boundary condition. This is often used for dissociation. The default is False.

**Returns** **v** – DESCRIPTION.

**Return type** TYPE

**set_grid**(*x*, *y*)

**set_masses**(*masses*)

class lime.wpd.**SPO3**

Bases: `object`

**evolve**(*psi0*, *dt*, *Nt=1*)

**set_grid**(*x*, *y*)

**set_potential**(*v*)

class lime.wpd.**Solver**

Bases: `object`

**set_obs_ops**(*obs_ops*)

lime.wpd.**adiabatic_2d**(*x*, *y*, *psi0*, *v*, *dt*, *Nt=0*, *coords='linear'*, *mass=None*, *G=None*)

propagate the adiabatic dynamics at a single surface

**Parameters**

- **dt** – time step
- **v** – 2d array potential matrices in 2D
- **psi** – list the initial state

**mass: list of 2 elements** reduced mass

**Nt: int**

**the number of the time steps, Nt=0 indicates that no propagation has been done,** only the initial state and the initial purity would be the output

**G: 4D array nx, ny, ndim, ndim** G-matrix

**Returns** psi_end: list the final state

**G: 2d array** G matrix only used for curvilinear coordinates lime.wpd.**density_matrix**(*psi_grid*)

compute electronic purity from the wavefunction

lime.wpd.**diabatic**(*x*, *y*)

PESs in diabatic representation :param x_range_half: float, the displacement of potential from the origin

in x

**Parameters**

- **y_range_half** – float, the displacement of potential from the origin in y
- **couple_strength** – the coupling strength between these two potentials
- **couple_type** – int, the nonadiabatic coupling type. here, we used: 0) no coupling 1) constant coupling 2) linear coupling

**Returns** v: float 2d array, matrix elements of the DPES and couplings

lime.wpd.**diabatic_coupling**(*x*, *y*)

lime.wpd.**dpsi**(*psi*, *kx*, *ky*, *ndim=2*)

Momentum operator operates on the wavefunction

**Parameters**

- **psi** (*2D complex array*) – DESCRIPTION.
- **ndim** (*int, default 2*) – coordinates dimension

**Returns** **kpsi** – DESCRIPTION.

**Return type** (nx, ny, ndim)

lime.wpd.**dxpsi**(*psi*)

Momentum operator operates on the wavefunction

**Parameters** **psi** (*2D complex array*) – DESCRIPTION.

**Returns** **kpsi** – DESCRIPTION.

**Return type** (nx, ny, ndim)

lime.wpd.**dypsi**(*psi*)

Momentum operator operates on the wavefunction

**Parameters** **psi** (*2D complex array*) – DESCRIPTION.

**Returns** **kpsi** – DESCRIPTION.

**Return type** (nx, ny, ndim)

lime.wpd.**gauss_k**(*k*, *a*, *x0*, *k0*)

analytical fourier transform of gauss_x(x), above

lime.wpd.**gauss_x_2d**(*sigma*, *x0*, *y0*, *kx0*, *ky0*)

generate the gaussian distribution in 2D grid :param x0: float, mean value of gaussian wavepacket along x :param y0: float, mean value of gaussian wavepacket along y :param sigma: float array, covariance matrix with 2X2 dimension :param kx0: float, initial momentum along x :param ky0: float, initial momentum along y :return: gauss_2d: float array, the gaussian distribution in 2D grid

lime.wpd.**hpsi**(*psi*, *kx*, *ky*, *v*, *G*) lime.wpd.**k_evolve_2d**(*dt*, *masses*, *kx*, *ky*, *psi*)

propagate the state in grid basis a time step forward with H = K :param dt: float, time step :param kx: float, momentum corresponding to x :param ky: float, momentum corresponding to y :param psi_grid: list, the two-electronic-states vibrational states in

grid basis

**Returns** psi_grid(update): list, the two-electronic-states vibrational states in grid basis

lime.wpd.**potential_2d**(*x_range_half*, *y_range_half*, *couple_strength*, *couple_type*)

generate two symmetric harmonic potentials wrt the origin point in 2D :param x_range_half: float, the displacement of potential from the origin

in x

**Parameters**

- **y_range_half** – float, the displacement of potential from the origin in y
- **couple_strength** – the coupling strength between these two potentials
- **couple_type** – int, the nonadiabatic coupling type. here, we used: 0) no coupling 1) constant coupling 2) linear coupling

**Returns**

v_2d: float list, a list containing for matrices: v_2d[0]: the first potential matrix v_2d[1]: the potential coupling matrix

between the first and second

**v_2d[2]: the potential coupling matrix** between the second and first

v_2d[3]: the second potential matrix

lime.wpd.**square_barrier**(*x*, *width*, *height*)

lime.wpd.**theta**(*x*)

**theta function :** returns 0 if x<=0, and 1 if x>0

lime.wpd.**x_evolve_2d**(*dt*, *psi*, *v*)

propagate the state in grid basis half time step forward with H = V :param dt: float

time step

**Parameters**

- **v_2d** – float array the two electronic states potential operator in grid basis
- **psi_grid** – list the two-electronic-states vibrational state in grid basis

**Returns** psi_grid(update): list the two-electronic-states vibrational state in grid basis after being half time step forward

### 3.2.29 Module contents

## 3.3 Floquet

### 3.3.1 Floquet package

#### Submodules

#### Floquet.Floquet module

#### Module contents

## 3.4 heom module

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## l

# INDEX

## E



## F



## G



## H



## I



## J

## K



## L



## M



## N

## O



## P



## Q



## R



## S

## T



## U



## V



## W



## X