Program structure for Genetic algorithm:

1 Initialize :

Full\_system\_instance = full\_system()

2 construct\_full\_system\_Hamiltonian\_part\_I

3 Initialize\_Genetic\_algorithm

assign customary fitness function designed by your self

run genetic algorithm

4 output result.

Genetic algorithm is only responsible for mutation, crossover and migration the parameter genetic code, the physical content is contained in class:

full\_system.

Full system have subclass: detector, we will first introduce this:

Detector class:

Main function:

1 initialize

2 construct\_Detector Hamiltonian part I

(i) construct state energy and diagonal part

(ii) compute off diagonal coupling number for detector, this number will output for Genetic algorithm to generate genetic code and parameter

3 construct\_Detector\_Hamiltonian part II

(i) read off diagonal coupling element and assign it to

self.offdiag\_coupling\_element\_list

(ii) add off diagonal coupling element into dmat.

extra function will be called in function.cpp

(1) construct initial wave function

(2) Reverse dmat to diagonal part

Function for Full\_system class:

1 Initialize function:

(i) In full\_system \_\_init\_\_() function, we also have to initialize two detectors: detector 1, detector 2

(ii) declare member variables.

Here one new feature is we construct matrix solely for photon, detector1 and detector2.

2 . Construct\_full\_system\_Hamiltonian\_part1c

(i) call detector function: construct Hamiltonian part I

(ii) construct full system diagonal part of Hamiltonian and their state.

Also corresponding matrix for photon Hamiltonian, detector1 Hamiltonian and detector2 Hamiltonian.

(iii) construct full system off-diagonal coupling number.

Here when we construct Hamiltonian index irow, icol.

we record lower triangular part of Hamiltonian matrix as well as upper triangular part of Hamltonian matrix at the same time.

(1) self.irow

(2) self.icol

At the same time, we record coupling index betwen photon and detector, also between detector and detector in list:

(1) inter\_photon\_detector\_coupling\_irow

(2) inter\_photon\_detector\_coupling\_icol

(iv) . compute position for intra detector coupling:

This function construct coupling in full matrix which corresponds to coupling in detector 1 and detector 2.

Here we have two extra list to record these element:

(1) detector1\_irow , detector1\_icol : this list include matrix element position both for diagonal and off diagonal part of matrix.

(2) detector1\_coupling\_irow, detector1\_coupling\_icol:

this list only include matrix element position for off diagonal part of matrix.

(3) We also construct matrix element location for irow, icol:

irow, icol for coupling in detector 1

irow, icol for coupling in detector 2

3 output off diagonal matrix element number

4 construct Hamiltonian part II:

(i) Reverse mat:

reverse matrix for

(1) detector1 Hamiltonian

(2) detector 2 Hamiltonian

(3) whole matrix

(4) matrix for detector 1 in full matrix

(5) matrix for detector 2 in full matrix.

(ii) Read off diagonal matrix element:

read off diagonal matrix element value from Genetic algorithm module.

This list can be split into 3 part:

Part I: detector 1. Part II. detector 2. Part III. coupling between detector and photon.

(iii) construct off-diagonal coupling in full matrix.

We first construct off diagonal coupling value between detector , detector and between detector photon.

Then in read\_off diagonal matrix elemnt function, we already let detector construct their off diagonal part.

Thus we can utilize off diagonal part of detector to construct its counterpart in full matrix.

We also construct off diagonal part for :

self.mat\_detector1

self.mat\_detector2

(This is matrix for detector in full Hamiltonian)

(iv) initialize wave function:

Every time we have to re-initialize wave function to do simulation.

Here we first initialize wave function for detector.

Then we initialize wave function for full system.

This will utilize initial detector 1 state and initial detector 2 state and initial photon state input into matrix.

(v) Shift Hamiltonian:

This module is essential for speed up simulation. Especially when energy window is small.

The step size for SUR algorithm is strictly restricted by largest matrix element in Hamiltonian.

5 Evolve Dynamics :

Using SUR algorithm to evolve dynamics

time step in SUR algorithm: 0.2 / Max\_element.

We also compute photon energy, detector 1 energy and detector 2 energy using matrix:

self.mat\_detector1 , self.detector1\_irow, self.detector1\_icol

self.mat\_detector2 , self.detector2\_irow, self.detector2\_icol

Combine full system class with Genetic algorithm module:

Implement Genetic algorithm:

1 Declare variable for detector 1 and detector 2 and full system.

2 Initialize full system. Here we call construct\_Hamiltonian\_part I.

3 initialize variable for Genetic algorithm.

this include mutation prob, cross\_over prob number of species in each process.

Here most important function linking Genetic algorithm and our quantum measurement simulation is fitness function.

Fitness function is declared in fitness.py.

4 Run Genetic algorithm

5 Evaluate result of Genetic algorithm:

best simulation result plot.

parameter for best simulation result.

fitness function for last generation.

Now we focus on Genetic algorithm part:

Extend\_Genetic\_algorithm :

This is class inherited from pyeasyga.GeneticAlgorithm

All extra variables we want to send into Genetic algorithm should through seed\_data variable.

1 Genetic\_algorithm.run() function ,

we first use create\_first\_generation() to create first generation.

Each generation:

(1) if generation % immigration\_generation\_number, we do immigration.

(2) create next generations.

2 create\_first\_generation:

(1) initialize randomly genetic code.

This is through self.create\_initial\_population() function

(2) calculate population fitness.

(i) if genes equivalent to what we already generated, we will just use fitness value computed before.

(ii) if genes is new, we call fitness function (Which will call SUR algorithm and analyze simulation result).

(3) rank population according to fitness function of genes.

3 create next generations:

(1) create\_new\_population() .

Here we will use mutation and crossover.

Here we will also use selection function to select individuals.

We use tournament\_selection method to select individuals:

(i) generate small populations with size tournament size.

(ii) choose the one with best fitness function among them.

This will we tends to choose inidividuals with best fitness function.

We use self.crossover\_function () and self.mutate\_function() to do crossover and mutation.

crossover function:

(this part may have bug, double check.)

part1[:index] + part2[index:]

part2[:index] + part1[index:]

4 immigration function:

we randomly select individuals from generations, send it to process0.

In process 0 we shuffle these indivudals and then scatter them back to process.

In each process, we them replace these chosen indivudal with individual from other process.

Now we focus on fitness function part: (fitness function is essential for success of genetic algorithm, so you have to make sure the fitness function you choose make sense.)

In fitness function, we do following thing:

1 we convert genetic algorithm bit into off-diagonal coupling parameter set.

This is achieved by convert\_bit\_to\_parameter function

2 we do simulation using SUR algorithm by adding off diagonal coupling

This is achieved in Evolve\_full\_system\_and\_return\_energy\_change function

3 Analyze energy for detector and system we get, and output information :

(1) First peak duration (2) maximum energy exchange

(3) localization duration

4 use fitness function to combine above three variables into one fitness function value we want to optimize.

This program targeted at optimizing parameters layer by layer.

We have already finished part : optimizing parameters all by once. Now

we need to classify different states and couplings , write a large iteration of genetic algorithm to optimize parameters layer by layer.

General idea:

1 You should have initial detector state to begin with.

2 search layers for connectivity . (Layer is determined by corresponding mode number. We first fit 2 mode and then 3 mode and 4 mode etc. ) Sort them according to their layer.

[ [index1, index2, ^ layer1] [ index1, index2, ^,layer2],[] ]

3 Our program should be able to

(1) return the index of coupling parameters of the choice (between layer 0 and layer 1. layer 1 and layer2 etc.) and the state these coupling corresponds to [state1, state2]

(2) record fitted parameter of previous layer as long as index & state of coupling [state1, state2]

(3) We have three kinds of coupling parameters:

1 fitted layer index nd fitted parameter value will feed into Genetic algorithm and full system class.

2 parameters to fit: Still index is needed and we fit these index according to Genetic algorithm

3 coupling with higher order layer : Set to 0.

For each module, they should providing following implementation:

(1) Full system class module (Return simulation result etc):

(i) sort the layer of state

(ii) return state mode number in different layer

(iii) can read parameters to set off-diagonal coupling.

This amounts to:

1. set all parameters to 0

2. read fixed parameter and their index and set coupling to fixed parameter

3. read parameter for fitting and their index to set it.

(iv) determine coupling parameter index between each layer.

Meanwhile output them to fit into Genetic algorithm

(2) Genetic algorithm module :

(i) Read fixed parameter set and their index +

fitting parameter set and their index (optimize in Genetic algorithm)

Fixed parameter can inherit from previous iteration.