Code Structure:

Even after I sort out the code , it’s still a little bit messy. Also, the idea in code to construct Hamiltonian involve some details in several paper and also based on professor Logan’s note which is an extension of 1990 Logan Wolynes’s theory (https://doi.org/10.1063/1.458637).

So we should set up a meeting to discuss details of code.

I.

Below is a brief introduction of code:

This code is intended for simulating IVR (vibrational energy redistribution ) for two coupled electronic states.

We assume vibrational states in two electronic states can delocalize through anharmonic coupling in state space, we are interested at what happened if there are tunneling between two electronic states.

Here we use eq.(7) in Logan’s note as model, which have one vibrational mode related to electronic state according to \lambda (b\_0 + b\_0^{+}).

The idea of this code is straightforward:

1 We first construct states represented by basis set (n0, n1, n2, n3, ... n\_{N}) in state space. Here we use various criteria to shrink state space size (as vibrational quatna are boson and it’s quantum number can go to infinity... we should construct a small state set around state are we interested at.)

2 Then we construct anharmonic coupling between these states according to scaling relationship.

(See <https://www.pnas.org/content/95/11/5960> eq.(4)(5) or

<https://aip.scitation.org/doi/10.1063/5.0043665> eq.(1)(2) )

If you are interested the origin of this formula, see: <https://aip.scitation.org/doi/10.1063/1.473253>)

With Harmonic oscillator’s Hamiltonian (H0) + anharmonic coupling (V) , we have our Hamiltonian H .

3 We then choose vibrational state |n0> we interested at , initialize wave function to choose |n0> as initial state and Evolve Schrodinger equation according to SUR algorithm (leap frog algorithm to evolve Shcrodinger equation).

We can compute various quantity to characterize this system during evolution, for example IPR. (See eq.[1] in <https://www.pnas.org/content/95/11/5960> ) .

PS: there are many variables not used in this code, I adapt old code for this project, so some function and variables are not in use.

PPS:

Although the structure of this code is simple, the code is messy because using MPI, you have to take care of what every process are doing. (for example: sending and receiving data from other process, what is MPI designed for).

II.

How to run the code:

1 Use git for version control, this is very useful. (I think you probably already know :) )

2 This C++ code use Cmake to compile and link different files.

CMakeList.txt contain codes to compile files.

When download this code in your computer, please change

directory path in CMakeList.txt :

include\_directories("path to folder /MPI\_Code")

3 input\_format.txt :

this file contains information about input.txt we use for code.

An example of input.txt in in ./result folder.

Please see code for meaning of these variables.

4 You have to install MPI to make this code works.

For linux this should be straightforward. If you encounter problems, we can discuss it. (I install it long ago, there may be subtle technical problems.)

5

Create Debug and Release folder. Then compile code in Debug and Release folder as follow:

cmake -DCMAKE\_BUILD\_TYPE=Release .. (Release folder)

make

cmake -DCMAKE\_BUILD\_TYPE=Debug .. (Debug folder)

make

See:

<https://stackoverflow.com/questions/7724569/debug-vs-release-in-cmake>

6 to run code in parallel.

mpirun -np num\_proc ./project name.

For me, if I want to use 5 cores to run it:

mpirun -np 5 ./4\_point\_correlation\_calculation\_clean\_version

For debugging MPI code See:

<https://www.open-mpi.org/faq/?category=debugging>

I usually use command below to debug:

mpirun -np 2 xterm -e gdb (my\_mpi\_application)

(You need to have xterm installed.   
https://zoomadmin.com/HowToInstall/UbuntuPackage/xterm)