Open_Nonlocal Documentation

The purpose of this python package is to explore the dynamics of Nonlocality (both Mermin and Svetlinchny's inequality) and entanglement in an open quantum system for a given Hamiltonian, initial state, and collapse operators.

The dependencies required are as follows:

- Python 2.7
- QuTip 2.0.0
  - A python module which contains many routines useful for quantum mechanics, especially a linear algebra formalism, a monte-carlo solver, and a master equation solver.
- Numpy
  - A numeric package for python.
- ParalelPython (pp) 1.6.3
  - A parallelization routine.
- Scipy
  - A powerful scientific program that contains plotting and optimization routines.
- S_maximize_3q
  - Custom code, requiring most of the above, which finds the maximum Svetlinchny violation for a given state.
- M_maximize_3q
  - Custom code, requiring most of the above, which finds the maximum Mermin violation for a given state.
- DM_S_M_maximize_3q
  - Custom code, requiring most of the above dependencies and the additional Open_Habtom custom dependency, which finds the maximum Svetlinchny and Mermin violation for a given density matrix.

In order to install the program, one must navigate to the folder containing the Open_3 directory. Once there, run the 'python setup.py install' command from the terminal (after doing the same for all the dependencies, of course).

To run the program, a file called Opentestrun.py is given. It allows one to set up a series of trials, parallel python job cluster instance, and execute all the trials from one program. Be aware that the program can take many hours to execute, even in parallel on a cluster. The program details a particular order of trials, but in general to use Opentestrun.py one inputs into run list a series of runs. One then defines the external nodes for the job cluster (if any) and the number of local CPU's. After this, and ensuring that the Hamiltonian is the correct one, the program will execute, calculating everything desired and plotting all the figures. Data will be saved in the /Data/Calculated folder, which can then be read again at a later date. Also note that the file structure found in the folder with the Open_3 directory must be replicated exactly, as the program will look for certain folders to save files.

Throughout the program, various calculated lists and initial states are handled through the Data object. The different components of this object are as follows:

- Data.avgSlist=List of the Maximum violation of Svetlinchny's inequality
- Data.avgMlist= List of the Maximum violation of Mermin's inequality
- Data.avgtglist= List of the three tangle
- Data.avgcr123list= List of the concurrence between qubit 1 and the joint system of qubits 2 and 3
- Data.avgcr231list= List of the concurrence between qubit 2 and the joint system of qubits 1 and 3
- Data.avgcr312list= List of the concurrence between qubit 3 and the joint system of qubits 1 and 2
- Data.avgvnlist= List of the von Neumann entropy
- Data.psi0= The initial state
- Data.infolist= A list of all the initial parameters for the hamilton
- Data.keylist= A list of strings of the names of the entries in the above list
- Data.name= The name of the trial
- Data.H= The hamiltonian
- Data.S_Maxlistlist= The list of vectors which maximized Svetlinchny's operator
- Data.M_Maxlistlist= The list of vectors which maximized Mermin's operator
- Data.raw= A numpy list of the states or density matrices from solve or mastersolve
- Data.opts= The program options, given in Opentestrun.py
- Data.tlist= A numpy array of all timesteps
- Data.slist= All collapse operators
- Data.pathname= The path where the file is being run
- Data.rawq= The states in qutip, rather than numpy form (identical to Data.raw except for this)
- Data.entire_raw= The entire output of solve or mastersolve

Below is a list of functions from the program which may be useful to those writing custom code:

- call_from_file(filename)
    - Returns: Data
    - This calls an file, creates a Data object, and then places the unpickled data from the file to the Data object.
- data_def()
    - Returns: Data
    - Creates a Data object
- save(Data)
    - Saves the Data in the Data object. Can choose, through opts, to save the raw data or not.
- call(Data)
    - Will call a file with the same information as is stored in Data (initial information only)
- mastersolve(Data)
    - Will solve a master equation and write the output to the Data instance. This function saves the entire output of the solution to Data.entire_raw, saves only the list of density matrices to Data.rawq, and saves a numpy version in Data.raw. It is the numpy version which is used later by the program.
- solve(Data,ntraj)
    - Will solve a monte-carlo trajectory simulation for ntraj number of trajectories. This function saves the entire output of the solution to Data.entire_raw, saves only the list of density matrices to Data.rawq, and saves a nummpy version in Data.raw. It is the numpy version which is used later by the program.
- Run_record(Data,opts,name,know,H,psi0,ntraj,gamma1,gamma2,gamma3,g12,g23,g31,Y,t0,tf,tsteps,guess)
    - Takes the above variables and writes them to a Data instance.
- Calculate(opts,name,know,H,psi0,ntraj,gamma1,gamma2,gamma3,g12,g23,g31,Y,t0,tf,tsteps,guess,job_server)
    - This will calculate the entanglement and Nonlocality for a given trial. Opts is the variable which determines which quantities are calculated.
- DC_Run(opts,name,know,H,psi0,ntraj,gamma1,gamma2,gamma3,g12,g23,g31,Y,t0,tf,tsteps,guess,job_server)
    - This code runs a trial and includes calculating, saving, and graphing all the data calculated.
- Test(opts,job_server)
    - This program tests, for a given opts, whether or not the dependencies are installed correctly. If the trial finishes without a hitch, it worked.

- Entangle_calc(qth_tstep,ntraj)
    - Returns C123,C231,C312,vn,Tgl
    - Calculates the entanglement for the qth_tstep of the monte-carlo simulation
- S_check_Single_DM(rho,guess,opts)
    - Returns Sm,maxlist
    - Calculates the maximum violation of Svetlinchny's inequality for a given density matrix, number of guesses, and program options. Also, if desired, will output a list of the vectors that maximized the inequality.
- S_calc_DM(rawdm,guess,opts,job_server)
    - Returns: avgSlist,S_Maxlistlist
    - Calculates the maximum violation of Svetlinchny's inequality for a series of density matrices, number of guesses, and program options. Also, if desired, will output a list of the vectors that maximized the inequality.
- M_check_Single_DM(rho,guess,opts)
    - Returns: M,maxlist
    - Calculates the maximum violation of Mermin's inequality for a given density matrix, number of guesses, and program options. Also, if desired, will output a list of the vectors that maximized the inequality.
- M_calc_DM(rawdm,guess,opts,job_server)
    - Returns: avgMlist, M_Maxlistlist
    - Calculates the maximum violation of Mermin's inequality for a series of density matrices, number of guesses, and program options. Also, if desired, will output a list of the vectors that maximized the inequality.
- S_check_Single(ith_traj,guess,tsteps,opts)
    - Returns: smlist,maxlistlist
    - Calculates the maximum violation of Svetlinchny's inequality for a given trajectory at each time step for the number of guesses, number of time steps, and program options. Also, if desired, will output a list of the vectors that maximized the inequality.
- S_calc(raw,guess,ntraj,tsteps,name,opts,job_server)
    - Returns: avgSlist, maxlistlist
    - Calculates the maximum violation of Svetlinchny's inequality for all trajectories in a monte-carlo simulation at each time step for the number of guesses, number of time steps, and program options. Also, if desired, will output a list of the vectors that maximized the inequality.
- M_check_Single(ith_traj,guess,tsteps,opts)
    - Returns: Mlist

    - Calculates the maximum violation of Mermin’s inequality for a given trajectory at each time step for the number of guesses, number of time steps, and program options.
- M_calc(raw,guess,ntraj,tsteps,name,opts,job_server)
    - Returns: avgMlist
    - Calculates the maximum violation of Mermin’s inequality for all trajectories in a monte-carlo simulation at each time step for the number of guesses, number of time steps, and program options.