Solver

> All the following parts should be implemented through Python.

Initial Value

Problem

> Given a initial-state (folk state) and a certain normalized operator (not in multiplied form), output its expectation value.
>
> For example:
>
> $$< 1_a, 2_b|a^\dagger a b^\dagger b|1_a, 2_b >= 2 =< a^\dagger a b^\dagger b >$$
>
> Recall the basic relation of the basic operators:
>
> $$\beta^\dagger|n_\alpha >= \delta_{\alpha\beta}(n_\alpha + 1)|n_\alpha + 1 >$$
> $$\beta|n_\alpha >= \delta_{\alpha\beta} n_\alpha|n_\alpha - 1 >$$
> $$< n_\alpha|n_\beta >= \delta_{\alpha\beta}\delta_{n_\alpha n_\beta}$$

Solution

> All the Input states should be represented in array. The *i*-th number in the array represent the initial-photon numbers of the *i*-th mode.
>
> For example:
>
> $$|1_a, 2_b, 0_c, 5_d >\rightarrow [1, 2, 0, 5]$$
>
> - Step One:
>
>   Scanning the operator from left to right, since the operator is normalized, it is obvious that the basic operators in the same mode will be gathered together. For certain mode, record the number of the creation operator and annihilation operator as n and m.
>
> - Step Two:
>
>   If n is not equal to m, the expectation value of this mode will be zero.
>
>   if n is larger than the photon number (marked as n_alpha) of current mode, the expectation value of this mode will be zero.
>
>   Otherwise, the expectation value of this mode will be
>
> $$n_\alpha!/(n_\alpha - n)!$$
>
> - Final Step:
>
> Multiply the expectation values of each modes together.

Evolution

> We derive the expressions for the evolution of the expectation value of certain operator based on the Lindblad master equation, that is

$$\frac{d\rho}{dt} = -\frac{i}{\hbar}[H,\rho] + \Sigma_n \frac{C_{Cn}}{2}[2O_n \rho O_n^\dagger - \rho O_n^\dagger O_n - O_n^\dagger O_n \rho](1)$$
$$\frac{d<A_i>}{dt} = tr(A_i \frac{d\rho}{dt})(2)$$
$$H = \hbar \Sigma_i C_{Hi} O_i (3)$$
$$Collapse\ Operator: \sqrt{C_{Cn}} O_n (4)$$

where *C_i* is coefficient and *O_i* is operator.

Then we can derive the evolution for the expectation value of certain operator:

$$\frac{d<A_i>}{dt} =< \frac{i}{\hbar}[H, A_i] > + \Sigma_n < \frac{C_{Cn}}{2}[2O_n^\dagger A_i O_n - O_n^\dagger O_n A_i - A_i O_n^\dagger O_n] > (5)$$

Problem

Given an operator, H and the collapse operators, output the expression for the evolution of the expectation value of the given operator in the form of operator tree.

Solution

We perform two operations to generate the expression for evolution. One to deal with Hamiltonian operator and the other to deal with collapse operators. In order to perform *Cluster-Expansion*, we request the client to input the max-length of single Operator

- Hamiltonian Operator

We request the client to input the Hamiltonian Operator as a list whose elements are the operators represented in the form of array and their coefficients.

Step One:

Following the rule of *Poisson bracket*, we can derive that:

$$< \frac{i}{\hbar}[H, A_i] >= i\Sigma_j C_{Hj}(< O_j A_i > - < A_i O_j >)$$

Travel through the list, follow this expression to connect the operators and normalize them, insert the result to the final expression in the form of Operator Tree.

Step Two:

Travelling through the Operator Tree, for the single operator whose length larger than the max-length, delete the operator, do *Cluster-Expansion* approximation for the operator and insert the result to the Operator Tree.

- Collapse Operators

We request the client to input the list **in the form of expression (4)** as a list. The operation is similar to the one for Hamiltonian Operator, just following the following expression:

$$\Sigma_n < \frac{C_{Cn}}{2}[2O_n^\dagger A_i O_n - O_n^\dagger O_n A_i - A_i O_n^\dagger O_n] >$$

Main Program

Request

The user should input the following elements to launch the solver

> - The initial-state in the form of array, folk state only.
> - Hamiltonian Operator and Collapse Operators.
> - The max-length of single operator and the tracking operators.
> - Time nodes as a list for tracking evolution.

> Basically, we will use a differential equation solver provided by *scipy* to solve the differential equation we derived based on cluster-expansion approach. We will output the expectation value associate with the input time nodes of the tracking operators as an array.

Derive and Assign

> - Step One:
>
> we create a dynamic array (motivated by classic dichotomy) to store the current value and also the reference of its evolution-expression tree and its node in the tracking tree of each tracking operator. Also, an *Operator Tree* whose node' s value equal to the operator 's position in the dynamic array.
>
> - Step Two:
>
> Add the initial tracking operators provided by the user to the Operator Tree and distribute space for them in the dynamic array. Calculate the initial value for them, store them in the dynamic array. And assign their nodes in tracking tree to the dynamic array.
>
> - Step Three:
>
> Travelling through dynamic array, if the value of the address of its evolution-expression is null, derive the expression for it. Do DFS in the evolution-expression tree. If we come across any single operator which is not exist in the tracking tree, repeat step two for it.
>
> - Finial Step:
>
> Repeat Step Three until all the values of the references of operators ' evolution-expressions are not null.

Calculate the Evolution

> Create a new dynamic array whose size is equal to the existed one to store the increment (This is needed in most of the differential equation solver).
>
> For each tracking operator, do DFS in its evolution-expression tree. For single operator, multiply its current value with its coefficient. For multiplied operators, multiply all the current value of single operator which it consist of ad its coefficient together.
>
> Add up the sum and store it in the new dynamic array.
>
> Separate its real part and imagine part and store in the new array whose size is twice of the new dynamic array. (This is needed as most of the differential equation solver do not support complex number)

Solve the Problem

> Connect the Solver with the differential equation solver and store the evolution of each tracking operator inputted by user in the array as output.

API

```
class CEBSolver:
    "聚类分解求解器"
    def
_init_(self,InitialState,HamilitonOperator,CollapseOperator,TrackingOperator):
        return IsSuccess
    def InitialValue(self,Operators)
        return InitialValues
    def Derive(self,Operator):
        return ExpressionTree
    def Assign(self,MaxLength):
        return self.TrackingArray
    def EvolutionF(self,CurrentValue,t):
        return dydt
    def Solver(self,t_list):
        return sol
```