

输入输出格式

输入

密度矩阵初始化

```
def initialRho(mode:int,dimension:list)
```

`mode` 模式数;

`dimension[mode]` 表示每个模式的维度

输入哈密顿量和坍塌算符

```
def defineEvo1(Hamiltonian:list,Collapse:list)
```

`Hamiltonian/Collapse=[[Coefficient:complex,'Operators':str],...]`

e.g. `H=[[1,'AAa'],[2j,'aBB']]`

- 规定输入时单个整体算符内，每个模式的算符一定会分开（即不会出现AABa）；且对每个模式的算符块，产生算符一定会出现在坍塌算符前（即不会出现aAaA）。

调用求解方法

- 蒙特TD法

```
def MonteTD(Tmax:int,epsilon:double,hyperParam:list)
```

`Tmax` 大迭代次数。

`epsilon` 相对精度值

`hyperParam` 超参数

- 动态规划

```
def DynamicProgram(Tmax:int,epsilon:double,hyperParam:list)
```

同上

- *梯度下降

```
def RLGradient(...)
```

比较复杂，等到时候具体实现时再讨论

输出

约化还原归一

首先，由于前面为了方便，实际上对矩阵元做了一些变换，所以现在要还原回去。还原方法为

$$\rho_{n_0 m_0, \dots, n_{M-1} m_{M-1}}^{(out)} = \prod_{i=0}^{M-1} \sqrt{n_i! m_i!} \rho_{n_0 m_0, \dots, n_{M-1} m_{M-1}}$$

还原时为了节省计算量，可以把各组的阶乘边存边算，按顺序来。

然后，对其进行前面所说的迹归一， $\rho^{(out)} = \frac{1}{Trace} \rho$

张量积大矩阵输出

```
def RowRho()
```

把多维矩阵整合成一个二维矩阵输出，其中索引变换为：对矩阵元 $\rho_{n_{large}, m_{large}} \rho_{n_0 m_0, \dots, n_{M-1} m_{M-1}}$

$$n_{large} = \prod_{i=0}^{M-1} n_i; m_{large} = \prod_{i=0}^{M-1} m_i$$

部分模式约化矩阵输出

```
def PartialRho(traceMode:list)
```

`traceMode` 要求和掉的模式集合 $\{M_i\}$,假设总数为 P

则有

$$\rho_{n_j m_j, \dots; j \notin \{M_j\}}^{(ptrace)} = \sum_{i \in M_j} \sum_{k_i=0}^{Dimension_i-1} \rho_{n_j m_j, \dots, k_i k_i, \dots}$$

然后按张量积大矩阵输出就行了。

模式均值矩

```
def AvgMoment(Ordere:list)
```

`Order` 表明每个模式需要的矩的次数 o_j

输出一个实数值

$$Moment = \sum_{All\ n} \left(\prod_{j=0}^{M-1} n_j^{o_j} \right) \rho_{n_0 n_0, \dots, n_{M-1} n_{M-1}}$$