数值求解方法

- 前两种是基础的求解方法，实现起来比较简单。最后一种是使用神经网络的，如果没时间就算了。

归一化处理

- 为了防止数值溢出和满足实际的密度矩阵条件，读出或者计算的时候需要进行归一化处理。也就是秩条件，

$$\sum \rho_{l_0 l_0,l_1 l_1 ...,l_{M-1} l_{M-1}} = 1$$

- 密度条件的性质和演化方程都都会满足密度矩阵的对角线为实数，一般不需要特别处理成系数的模（但类型还是复数，需要注意一下）。
- 归一化很简单，若

$$\sum \rho_{l_0 l_0,...,l_{M-1} l_{M-1}} = Trace$$

那么归一化后

$$\rho' = \frac{1}{Trace}\rho$$

动态规划

- 初始化，现在还不太清楚怎样的初始化比较好，现在猜测可能比较好的初始方法是设

$$\rho_{l_0 k_0,...,l_{M-1} k_{M-1}} = \frac{1}{\prod_{j=0}^{M-1} n_j}$$

- 因为这个方程是齐次方程，为了避免最后的解全是零，强制设定

$$\rho_{00,...,00} = \frac{1}{\prod_{j=0}^{M-1} n_j}$$

仅在读出或者归一化时由归一化规则改变，其他过程应不改变它的值。

- 动态规划的方法很简单，很暴力，线性遍历存储密度矩阵的数组，假设是第t次迭代，则对每个元素是实现下列表达式：

$$\rho^{t+1}_{l_0 k_0,...,l_{M-1} k_{M-1}} = \alpha \frac{1}{P(l_0,k_0,\dots,l_{M-1},k_{M-1})} \sum_{i \in Neighbors} P(l_0^i,k_0^i,\dots,l_{M-1}^i,k_{M-1}^i)\rho^t_{l_0^i k_0^i,...,l_{M-1}^i k_{M-1}^i} + (1-\alpha)\rho^t_{l_0 k_0,...,l_{M-1} k_{M-1}}$$

α是超参数（其实这个方法是强化学习里的，所以肯定有参数给你调）。

这个过程，由于多维数组实际上是线性存储的，因此迭代时分块进行并行，共享一个存储数组。原则上，更新的顺序不会影响这个方法的结果，但由于共享内存，**需要注意不同线程访问同一个地址时可能出现访问冲突，要想办法解决。**

- 当元素的模超过一个预设参数（100差不多了），就做一次归一化处理。
- 预设一个最大迭代次数T_{max},和一个精度要求ε,当$t > T_{max}$或者对每个元素都有

$$\frac{\left|P(l_0,k_0,\dots,l_{M-1},k_{M-1})\rho_{l_0 k_0,...,l_{M-1} k_{M-1}} - \sum_{i \in Neighbors} P(l_0^i,k_0^i,\dots,l_{M-1}^i,k_{M-1}^i)\rho_{l_0^i k_0^i,...,l_{M-1}^i k_{M-1}^i}\right|}{Max(P(l_0,k_0,\dots,l_{M-1},k_{M-1})\rho_{l_0 k_0,...,l_{M-1} k_{M-1}},\varepsilon)} \leq \varepsilon$$

时停止迭代，最后作一次归一化，输出结果。

蒙特卡洛TD迭代

- 初始方法和上一个方法一样。
- 为了避免全零解，也需要上述的固定值处理方法。

- 但在这个方法里，我们随机生成蒙特卡洛路径去求解这个方程组。
- 我们比较关系对角元素的值，所以一般从对角元素出发，对第t次大迭代，即

  $$\rho^t_{l_0^0 l_0^0, l_1^0 l_1^0 \ldots, l_{M-1}^0 l_{M-1}^0}$$

  等权随机从它的邻居里面挑一个，行进到该邻居,记为

  $$\rho^t_{l_0^1 k_0^1, \ldots, l_{M-1}^1 k_{M-1}^1}$$

  然后更新前一项的值

  $$\rho^{t+1}_{l_0^0 l_0^0, l_1^0 l_1^0 \ldots, l_{M-1}^0 l_{M-1}^0} = \rho^t_{l_0^0 l_0^0, l_1^0 l_1^0 \ldots, l_{M-1}^0 l_{M-1}^0} + \alpha \left[\gamma \frac{P(l_0^1, k_0^1, \ldots, l_{M-1}^1, k_{M-1}^1)}{P(l_0^0, l_0^0, \ldots, l_{M-1}^0, l_{M-1}^0)} \rho^t_{l_0^0 k_0^1, \ldots, l_{M-1}^1 k_{M-1}^1} - \rho^t_{l_0^0 l_0^0, l_1^0 l_1^0 \ldots, l_{M-1}^0 l_{M-1}^0} \right]$$

  α, γ是超参数。假设第i步行进到索引$Index_i^t$,同理有随机挑选下一个元素使得

  $$Index_{i+1}^t \in Neighbors(Index_i^t)$$

  然后运用上述表达式更新值

  $$\rho^{t+1}_{Index_i^{t+1}} = \rho^{t+1}_{Index_i^t} + \alpha \left[\gamma N_{neighbors} \frac{P(Index_{i+1}^t)}{P(Index_i^t)} - \rho^t_{Index_i^t} \right]$$

  其中$N_{neighbors}$是邻居的总数，直到行走到规定索引外，则重新开始，**遍历下一个对角元作为初始点（总之每次大迭代都要对每个对角元来一遍）**，重复迭代。不可避免的可能会一个点重复走过，但没关系，还是按照上面的表达式覆盖更新就行了。
- 重复上述，直到$t > T_{max}$,或者对每个对角元有

  $$\frac{\left| P(l_0, l_0, \ldots, l_{M-1}, l_{M-1}) \rho_{l_0 l_0, \ldots, l_{M-1} l_{M-1}} - \sum_{i \in Neighbors} P(l_0^i, k_0^i, \ldots, l_{M-1}^i, k_{M-1}^i) \rho_{l_0^i k_0^i, \ldots, l_{M-1}^i k_{M-1}^i} \right|}{Max(P(l_0, l_0, \ldots, l_{M-1}, l_{M-1}) \rho_{l_0 l_0, \ldots, l_{M-1} l_{M-1}}, \varepsilon)} \leq \varepsilon$$

- 这里有两个地方需要并行处理
  - 一个在一次大迭代t中，遍历不同对角元初始点需要并行，类似动态规划法，需要小心内存共享。
  - 不同的大迭代可以**并行进行**，**内存不共享**，最后的结果是**对不同线程的结果求平均**

*强化梯度下降

这个要用神经网络，建议用tensorflow实现。

- 目标是训练一个这样的函数

  $$\mathrm{P}(\vec{C}, \vec{n}, \vec{m}; \vec{W}) = \rho^{(\vec{C})}_{\vec{n}, \vec{m}}$$

  其中$\vec{n}, \vec{m}$对应密度矩阵元的索引,$\vec{C}$是输入的哈密顿量或者坍塌算符的系数向量（因为我们希望求出不同系数的解）。

  由于矩阵元是一个复数，所以需要实部和虚部分离，即

  $$\vec{\mathrm{P}}(\vec{C}, \vec{n}, \vec{m}; \vec{W}) = \begin{pmatrix} Re[\rho^{(\vec{C})}_{\vec{n}, \vec{m}}] \\ Im[\rho^{(\vec{C})}_{\vec{n}, \vec{m}}] \end{pmatrix}$$

- cost-function很好定义，为

  $$VE(\vec{W}) = \left(\vec{P}_{ss} - \hat{\vec{P}} \right)^2 = \vec{P}^2_{diff}$$

- 那么梯度下降自然如下

$$\vec{W}_{t+1}^{RE} = \vec{W}_{t}^{RE} + \alpha[P_{diff,1}]\nabla P_1$$
$$\vec{W}_{t+1}^{Im} = \vec{W}_{t}^{Im} + \alpha[P_{diff,2}]\nabla P_2$$

其实就是最后输出有两个神经元。

- 但不同于一般的神经网络，我们这里没有正确答案label，所以我们采用label的无偏估计来代替，这就是强化学习的核心，即

$$\rho_{\vec{n},\vec{m}}^{ss} \approx \frac{1}{P(\vec{n},\vec{m})} \sum_{(\vec{l},\vec{k})\in Neighbor(\vec{n},\vec{m})} P(\vec{l},\vec{k})\hat{\rho}_{\vec{l},\vec{k}}$$

等式右边是当前神经网络给出的解，因此我们基本上不需要任何训练数据，直接把参数输入，让他跑就行了。