pyChemiQ

Release v1.0

OriginQC

Mar 13, 2024

pyChemiQ 安装

pyChemiQ 是一款由本源量子开发的 python 软件库, 它可以用于在真实量子计算机或虚拟机上实现费米子体系的模拟与计算。支持用户自定义映射、拟设、优化器，方便用户二次开发。该软件包为量子化学计算和方法开发提供了一个简单、轻量级且高效的平台。pyChemiQ 可用于在量子计算机上使用平均场和后平均场方法模拟分子来解决量子化学问题。pyChemiQ 简化了分子结构输入和量子电路之间的转换，最大限度地减少了进入该领域所需的领域专业知识，让感兴趣的学者更方便解决和研究量子计算机上的电子结构问题。

目前，pyChemiQ 支持输入分子结构得到二次量子化后的 Fermion Hamiltonian；映射方面，pyChemiQ 支持 Jordan-Wigner(JW) 变换、Bravyi-Kitaev(BK) 变换、Parity 变换和 Multilayer Segmented Parity(MSP) 变换方法将二次量子化的 Fermion Hamiltonian 算符映射成量子计算机上的 pauli Hamiltonian 算符；拟设方面，pyChemiQ 也支持不同的拟设构造量子电路，例如 Unitary Coupled Cluster（UCC）、Hardware-Efficient、symmetry-preserved 等拟设构造量子电路；优化器方面，pyChemiQ 提供了以下几种经典优化器来进行变分参数的优化：NELDER-MEAD、POWELL、COBYLA、L-BFGS-B、SLSQP 和 Gradient-Descent。通过自定义映射和拟设方式，用户也可以自行构造或优化量子线路，求得电子结构问题的最优基态能量解。

- 欢迎您在 Github 上 Fork 我们的项目或给我们留言：Github 网站链接 。
- 为特定量子化学问题定制 pyChemiQ 功能，请联系本源量子陈先生 18221003869 或给我们发送邮件 dqa@originqc.com。
- 体验全部最新功能的可视化教学工具 ChemiQ，请前往 官网 下载。
- 若您想在文献中引用 pyChemiQ 或 ChemiQ, 请按照如下格式: Wang Q, Liu H Y, Li Q S, et al. Chemiq: A chemistry simulator for quantum computer[J]. arXiv preprint arXiv:2106.10162, 2021.

Chapter 1

安装介绍

我们提供了 Linux,MacOS,Windows 上的 python 预编译包供安装，目前 pyChemiQ 支持 3.8 版本的 python，更多 python 版本的支持还在开发中。

如果您已经安装好了 python 环境和 pip 工具，pyChemiQ 可通过如下命令进行安装：

```
pip install pychemiq
```

Chapter 2

氢分子案例计算

为了快速上手 pyChemiQ，我们先来看下 H_2 分子的案例计算。在设置参数前，我们先来了解两个名词：映射和拟设。为了在量子计算机上模拟电子结构问题，我们需要一套转换方式，将电子的费米子算符编码到量子计算机的泡利算符，这就是映射 (mapping)。为了获得与体系量子终态相近的试验波函数，我们需要一个合适的波函数假设，我们称之为拟设 (Ansatz)。并且理论上，假设的试验态与理想波函数越接近，越有利于后面得到正确基态能量。

下面氢分子的例子中我们使用 sto-3g 基组，映射方式使用 JW 变换，拟设采用 UCCSD，经典优化器方法采用 SLSQP。这里我们暂不展开每种方法背后的理论背景，详细的介绍参见理论背景章节：

```
# 先导入需要用到的包
from pychemiq import Molecules,ChemiQ,QMachineType
from pychemiq.Transform.Mapping import jordan_wigner,MappingType
from pychemiq.Optimizer import vqe_solver
from pychemiq.Circuit.Ansatz import UCC
import numpy as np

# 初始化分子的电子结构参数，包括电荷、基组、原子坐标 (angstrom)、自旋多重度
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"

mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge)

# 利用 JW 变换得到泡利算符形式的氢分子哈密顿量
fermion_H2 = mol.get_molecular_hamiltonian()
```

(continues on next page)

(continued from previous page)

```
pauli_H2 = jordan_wigner(fermion_H2)

# 准备量子线路，需要指定的参数有量子虚拟机类型 machine_type，拟设映射类型 mapping_type，
# 泡利哈密顿量的项数 pauli_size，电子数目 n_elec 与量子比特的数目 n_qubits
chemiq = ChemiQ()
machine_type = QMachineType.CPU_SINGLE_THREAD
mapping_type = MappingType.Jordan_Wigner
pauli_size = len(pauli_H2.data())
n_qubits = mol.n_qubits
n_elec = mol.n_electrons
chemiq.prepare_vqe(machine_type,mapping_type,n_elec,pauli_size,n_qubits)

# 设置拟设类型，这里我们使用 UCCSD 拟设
ansatz = UCC("UCCSD",n_elec,mapping_type,chemiq=chemiq)

# 指定经典优化器与初始参数并迭代求解
method = "SLSQP"
init_para = np.zeros(ansatz.get_para_num())
solver = vqe_solver(
        method = method,
        pauli = pauli_H2,
        chemiq = chemiq,
        ansatz = ansatz,
        init_para=init_para)
result = solver.fun_val
n_calls = solver.fcalls
print(result,f"函数共调用{n_calls}次")
energies = chemiq.get_energy_history()
print(energies)
```

打印得到的结果为：

```
-1.1372838317140834 函数共调用 9 次
[-1.1167593073964257, -1.0382579032966825, -1.137282968297819, -1.137282968297819, -1.
↪1372838302540205, -1.137283647727291, -1.1372838297780967, -1.1372838317140834, -1.
↪1372838317140834]
```

我们将 pyChemiQ 打印出来的数据作图，与同基组下的经典 Full CI 进行对比。可以看到随着函数迭代次数的增加，电子能量逐渐收敛至 Full CI 的能量，如下图所示。而且当函数迭代到第二次时电子能量已经达到了化学精度 1.6×10^{-3} Hartree。

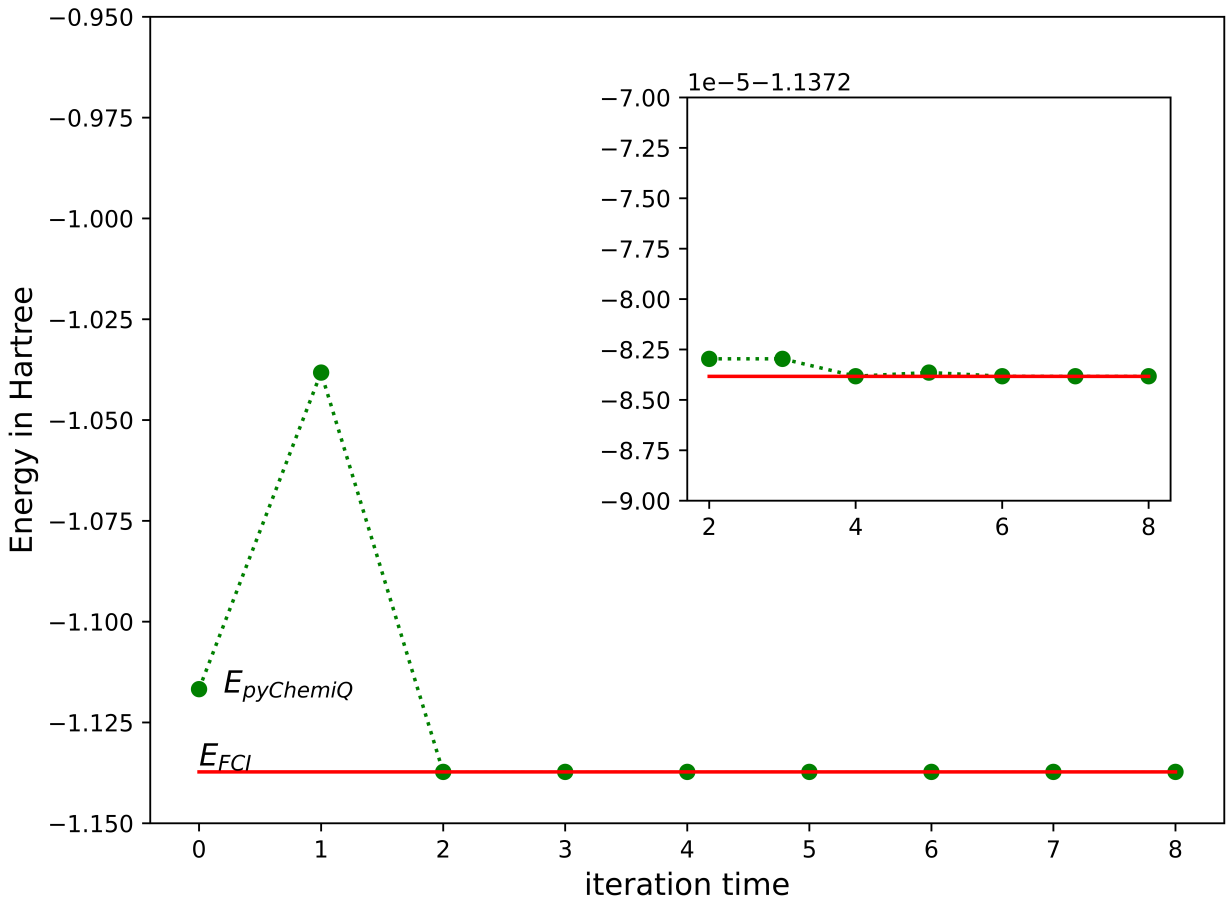
−0.950
−0.975
−1.000
−1.025
−1.050
−1.075
−1.100
−1.125
−1.150
Energy in Hartree
0
1
2
3
4
5
6
7
8
iteration time
$E_{pyChemiQ}$
E_{FCI}
1e−5−1.1372
−7.00
−7.25
−7.50
−7.75
−8.00
−8.25
−8.50
−8.75
−9.00
2
4
6
8

图: 氢分子能量收敛曲线

Chapter 3

哈密顿量教程

量子化学的基本目标之一是揭示分子结构和性质的关系。我们可以根据描述分子中电子的波函数计算出准确的分子性质，电子波函数满足电子薛定谔方程。系统的电子薛定谔方程可以写为式 (1)。$|\psi_{el}\rangle$ 是系统的电子波函数，$\hat{H}_{el}$ 是系统中电子的 Hamiltonian，为式 (2) 的形式 (原子单位)，式中 A 指的是核，i, j 指的是电子。第一项是电子的动能，第二项表示电子与核的库仑吸引力，r_{Ai} 是电子 i 与原子序数为 Z_A 的核 A 之间的距离，第三项表示电子与电子排斥作用，r_{ij} 是电子 i 与电子 j 之间的距离。该式的求解是建立在电子运动在固定不动的原子核周围之上，因为在原子中核的质量比电子质量大得多，运动比电子慢。所以可以忽略原子核的效应，假定电子运动在固定的原子核周围来近似。

$$\hat{H}_{el}|\psi_{el}\rangle = E_{el}|\psi_{el}\rangle \tag{1}$$

$$\hat{H}_{el} = -\frac{1}{2}\sum_i \nabla_i^2 - \sum_{A,i}\frac{Z_A}{r_{Ai}} + \sum_{i>j}\frac{1}{r_{ij}} \tag{2}$$

所以我们的核心目标可以描述为：求解在特定分子体系下定态电子薛定谔方程，得到的解为体系处于不同定态下的本征值，即电子能量。根据变分原理 (variational principle) 可知，求得的最小特征值为体系的基态能量 E_0 (ground-state energy), 对应的系统状态为基态 (ground-state)。

$$E = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} \geq E_0$$

在二次量子化的形式中 (second quantization)，电子波函数可以表示成占据数态 (occupation number state), 这里采用轨道序数从右向左依次递增的约定, 如式 (3) 所示。其中 N 为电子数，M 为自旋轨道数。当自旋轨道 χ_p 被电子占据时，n_p 为 1；反之，未被占据时，n_p 为 0。换句话说，占据数态是一组只包含 0 和 1 的二元的数串，其长度是自旋轨道的数量，每一位上的 1 表示该编号下的自旋轨道是被占据的。由此我们可以将一系列产生算符作用在真空态 $|\rangle$ 来构造任何系统的 Hartree-Fock 态。如式 (4) 所示。式中 $a_i^\dagger$ 为产生算符 (creation operator)，它的作用是在第 i 个自旋轨道上产生一个电子，同理，定义 a_j 为湮灭算符 (annihilation operator)，它的作用是在第 j 个自旋轨道上湮灭一个电子。

$$\Phi_{HF}(\chi_1, \chi_2, \cdots, \chi_N) = |n_{M-1}, n_{M-2}, \cdots, n_0\rangle \tag{3}$$

$$|n_{M-1}, n_{M-2}, \cdots, n_0\rangle = a_0^\dagger a_1^\dagger \cdots a_N^\dagger|\rangle = \prod_{i=1}^{N} a_i^\dagger|\rangle \tag{4}$$

二次量子化后，电子的 Hamiltonian 表示成式 (5) 的形式[1]，该式中第一项为是单粒子算符，第二项为双粒子算符，下标 $pqrs$ 分别代表不同电子自旋轨道，其中 h_{pq} 、h_{pqrs} 分别代表单、双电子积分。如果选定基组，我们就可以确定积分的具体值。

$$\hat{H}_{el} = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{pqrs} h_{pqrs} a_p^\dagger a_q^\dagger a_s a_r \qquad (5)$$

在 pyChemiQ 中，利用方法 get_molecular_hamiltonian() 可以得到分子的费米子哈密顿量。下面我们以氢分子为例，展示得到的哈密顿量。

```
from pychemiq import Molecules
```

在 pychemiq.Molecules 这个模块中，我们可以初始化分子的电子结构参数, 包括分子的几何构型 (坐标单位为 angstrom)、基组、电荷、自旋多重度，将分子信息存储在对象中。输入参数后执行经典 hartree-fock 计算。

```
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"
mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge)
```

通过调用 get_molecular_hamiltonian() 可以得到费米子形式的分子 Hamiltonian 子项及每项的系数。以下是示例代码及氢分子的费米子 Hamiltonian 打印结果。

```
fermion_H2 = mol.get_molecular_hamiltonian()
print(fermion_H2)


{
: 0.715104
0+ 0 : -1.253310
1+ 0+ 1 0 : -0.674756
1+ 0+ 3 2 : -0.181210
1+ 1 : -1.253310
2+ 0+ 2 0 : -0.482501
2+ 1+ 2 1 : -0.663711
2+ 1+ 3 0 : 0.181210
2+ 2 : -0.475069
3+ 0+ 2 1 : 0.181210
3+ 0+ 3 0 : -0.663711
3+ 1+ 3 1 : -0.482501
```

(continues on next page)

[1] Attila Szabo and Neil S Ostlund. *Modern quantum chemistry: introduction to advanced electronic structure theory*. Courier Corporation, 2012.

(continued from previous page)

```
3+ 2+ 1 0 : -0.181210
3+ 2+ 3 2 : -0.697652
3+ 3 : -0.475069
}
```

另外，pyChemiQ 也支持活性空间和冻结轨道的设置，通过限制电子和所激发的轨道，只考虑部分的电子激发到部分的轨道上的激发组态。适当使用活性空间和冻结轨道可以在保证化学精度的同时减少模拟所需要的量子比特数目。

活性空间方法 (即经典中的 CASSCF) 把分子轨道分为三个部分：电子占据轨道，活性轨道和空轨道。其中电子占据轨道始终保持电子双占据，空轨道则始终不占据电子。有限的电子只会在活性轨道里面自由跃迁。我们定义活性空间 [m,n]，其中 m 为活性轨道的数目，n 为活性电子的数目，此时的计算包含了 n 个电子在 m 个轨道上的所有排列。活性轨道的选取一般在是在 HOMO 与 LUMO 位置，因为处在这些分子轨道中的电子就像原子轨道中的价电子一样是化学反应中最活泼的电子，是化学反应的关键。下图中就是活性空间 [2,2] 的示例：

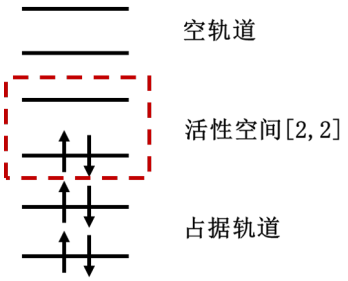

图 1: 活性空间设置中分子轨道的划分

在 pyChemiQ 中，我们在 pychemiq.Molecules 中通过参数 active 来指定活性空间。比如我们指定活性空间 [2,2] 来得到 LiH 的哈密顿量：

```
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = ["Li     0.00000000    0.00000000    0.37770300",
        "H      0.00000000    0.00000000   -1.13310900"]
active = [2,2]
mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge,
    active = active)
fermion_LiH = mol.get_molecular_hamiltonian()
```

pyChemiQ 对冻结轨道数目的设置是通过参数 nfrozen 来指定的，我们默认从能量最低的分子轨道算起开始冻结该轨道及轨道上的电子。比如下面这个例子我们冻结一个空间轨道来得到 LiH 的哈密顿量：

```
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = ["Li     0.00000000    0.00000000    0.37770300",
        "H      0.00000000    0.00000000   -1.13310900"]
nfrozen = 1
mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge,
    nfrozen = nfrozen)
fermion_LiH = mol.get_molecular_hamiltonian()
```

参考文献

Chapter 4

映射教程

为了在量子计算机上模拟电子结构问题，我们需要一个映射关系，将电子的费米子算符映射 (mapping) 到量子计算机的泡利算符 (pauli operator, 即 pauli X 矩阵, pauli Y 矩阵, pauli Z 矩阵, 单位矩阵 I)。目前，比较常见的映射有 Jordan-Wigner(JW) 变换[1]、Bravyi-Kitaev(BK) 变换[2] 和 Parity 变换[3] 等。不同的变换所得到的量子线路深度可能有所不同，但他们的功能都是一致的，都是为了将费米子系统映射到量子计算机中去。

这里，我们以 JW 变换为例展示三个自旋轨道的映射示意图，如图 1 所示。可以看出，在 JW 变换下，每一个量子比特标识一个费米轨道，占据态和非占据态分别映射到量子比特的 $|1\rangle$ 态和 $|0\rangle$ 态。此时，轨道和量子比特是一一对应的。

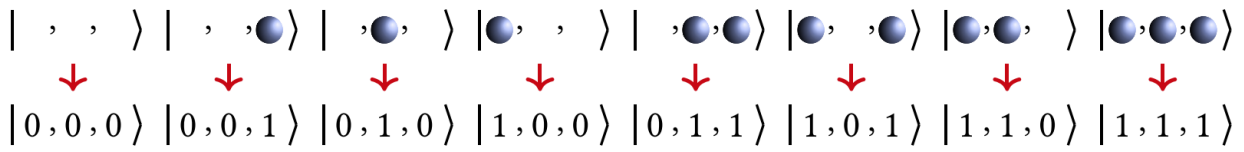

图 1: 三个自旋轨道的 JW 变换示意图. 图引自[4]

在 pychemiq.Transform 这个模块中，一个非常重要的子模块是 pychemiq.Transform.Mapping，实现的是就是把费米子算符映射成为泡利算符。目前 pyChemiQ 支持的映射方式有 Jordan-Wigner(JW) 变换、Bravyi-Kitaev(BK) 变换、Parity 变换和 Multilayer Segmented Parity(MSP) 变换[5] 。可以通过如下方式调用相应的包：

```
from pychemiq.Transform.Mapping import (
jordan_wigner,
```

(continues on next page)

[1] E Wigner and Pascual Jordan. Über das paulische äquivalenzverbot. *Z. Phys*, 47:631, 1928.

[2] Sergey B Bravyi and Alexei Yu Kitaev. Fermionic quantum computation. *Annals of Physics*, 298(1):210–226, 2002.

[3] Jacob T Seeley, Martin J Richard, and Peter J Love. The bravyi-kitaev transformation for quantum computation of electronic structure. *The Journal of chemical physics*, 137(22):224109, 2012.

[4] Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews* , 120(22):12685–12717, 2020.

[5] Qing-Song Li, Huan-Yu Liu, Qingchun Wang, Yu-Chun Wu, and Guo-Ping Guo. A unified framework of transformations based on the jordan–wigner transformation. *The Journal of Chemical Physics*, 157(13):134104, 2022.

(continued from previous page)

```
bravyi_kitaev,
parity,
segment_parity)
```

例如我们使用 JW 变换将上节氢分子的费米子 Hamiltonian 映射成泡利形式，示例代码及打印结果如下：

```
# 先初始化得到氢分子的费米子 Hamiltonian
from pychemiq import Molecules
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"
mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge)
fermion_H2 = mol.get_molecular_hamiltonian()

# 使用 JW 变换将得到的氢分子的费米子 Hamiltonian 映射成泡利形式
pauli_H2 = jordan_wigner(fermion_H2)
print(pauli_H2)

{
"" : -0.097066,
"X0 X1 Y2 Y3" : -0.045303,
"X0 Y1 Y2 X3" : 0.045303,
"Y0 X1 X2 Y3" : 0.045303,
"Y0 Y1 X2 X3" : -0.045303,
"Z0" : 0.171413,
"Z0 Z1" : 0.168689,
"Z0 Z2" : 0.120625,
"Z0 Z3" : 0.165928,
"Z1" : 0.171413,
"Z1 Z2" : 0.165928,
"Z1 Z3" : 0.120625,
"Z2" : -0.223432,
"Z2 Z3" : 0.174413,
"Z3" : -0.223432
}
```

除此之外，pyChemiQ 也支持自行构建费米子算符或泡利算符来自定义哈密顿量。详见进阶教程第一节。

参考文献

Chapter 5

拟设教程

为了获得与体系量子终态相近的试验波函数，我们需要一个合适的波函数假设，我们称之为拟设 (Ansatz)。并且理论上，假设的试验态与理想波函数越接近，越有利于后面得到正确基态能量。实际上，使用 VQE 算法在量子计算机上模拟分子体系基态问题，最终都是转换到在量子计算机上对态进行演化，制备出最接近真实基态的试验态波函数。最后在对哈密顿量进行测量求得最小期望值，即基态能量。经典的传统计算化学领域已经发展了多种多样的波函数构造方法，比如组态相互作用法 (Configuration Interaction, CI)[1] , 耦合簇方法 (Coupled-Cluster, CC) 等[2] 。

目前，应用在 VQE 上的主流拟设主要分为两大类，一类化学启发拟设，如酉正耦合簇 (Unitary Coupled-Cluster, UCC)[3] ，另一类是基于量子计算机硬件特性构造的拟设，即 Hardware-Efficient 拟设[4] 。截至现在，pyChemiQ 支持的拟设有 Unitary Coupled Cluster(UCC)、Hardware-Efficient、Symmetry-Preserved[5] 来构造量子电路，通过自定义的方式构建量子线路拟设详见进阶教程的量子线路教程。

1. 通过 Unitary Coupled Cluster 拟设构建线路

在求解体系基态能量选用 Hartree-Fock 态作为初猜波函数时，由于 Hartree-Fock 态为单电子组态，没有考虑电子关联能，所以要将其制备成多电子组态 (也就是纠缠态)，以使测量结果达到化学精度。UCC 中的 CC 即是量子化学中的耦合簇算符 $e^{\hat{T}}$，它从 Hartree-Fock 分子轨道出发，通过指数形式的耦合算符得到真实体系的波函数。详细的理论请查看理论基础 的拟设小节。下面我们示例的是氢分子单激发算符 $a_2^\dagger a_0 - a_0^\dagger a_2$ 的量子线路。

[1] Peter J Knowles and Nicholas C Handy. A new determinant-based full configuration interaction method. *Chemical physics letters*, 111(4-5):315–321, 1984.

[2] Rodney J Bartlett. Many-body perturbation theory and coupled cluster theory for electron correlation in molecules. *Annual review of physical chemistry*, 32(1):359–401, 1981.

[3] Andrew G Taube and Rodney J Bartlett. New perspectives on unitary coupled-cluster theory. *International journal of quantum chemistry*, 106(15):3393–3401, 2006.

[4] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink,Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[5] Bryan T Gard, Linghua Zhu, George S Barron, Nicholas J Mayhall, Sophia E Economou, and Edwin Barnes. Efficient symmetry-preserving state preparation circuits for the variational quantum eigensolver algorithm. *npj Quantum Information*, 6(1):10, 2020.

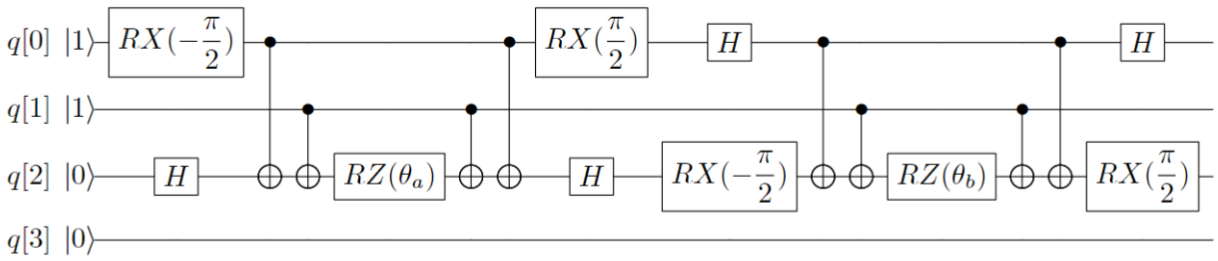

图 1: 单激发算符 $a_2^\dagger a_0 - a_0^\dagger a_2$ 的量子线路

2. 通过 Hardware-Efficient 拟设构建线路

基于量子计算机硬件特性构造的拟设是各种变分量子算法中经常使用的一种拟设。当我们对要解决的问题信息知之甚少时，我们可以使用基本的量子门来搜索目标。通常基于硬件特性构造的拟设由许多层组成，每一层的量子线路上都由两部分组成：一是每个量子比特上的单比特旋转门（RZRXRZ），二是相邻量子比特上两个纠缠比特形成的受控逻辑门（受控 RY 门）。单层的量子线路如下图所示：

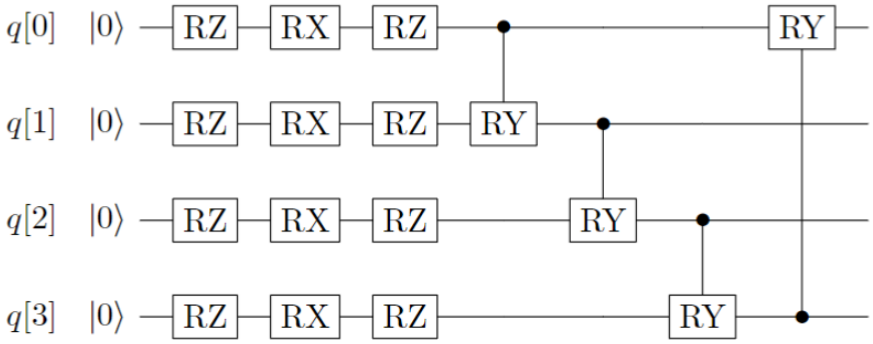

图 2: 基于 Hardware-Efficient 拟设的单层量子线路

3. 通过 Symmetry-Preserved 拟设构建线路

基于 Symmetry-Preserved 拟设是通过一个特定纠缠门 (entangling gate) $A(\theta, \phi)$ 作为组成单元，来构建保持粒子数守恒、时间反演对称性与自旋对称性的拟设。在 $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ 为基下，$A(\theta, \phi)$ 可以被表示为：

$$A(\theta,\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & e^{i\phi}\sin\theta & 0 \\ 0 & e^{-i\phi}\sin\theta & -\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

该纠缠门分解成单、双比特基础门为：

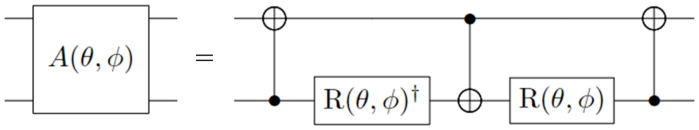

图 3: $A(\theta,\phi)$ 门的分解。其中:

$$R(\theta,\phi) = R_z(\phi+\pi)R_y(\theta+\pi/2), R_z(\theta) = exp(-i\theta\sigma_z/2), R_y(\phi) = exp(-i\phi\sigma_y/2)$$

由 $A(\theta,\phi)$ 与 X 门构建的 Symmetry-Preserved 拟设的量子线路如下图所示:

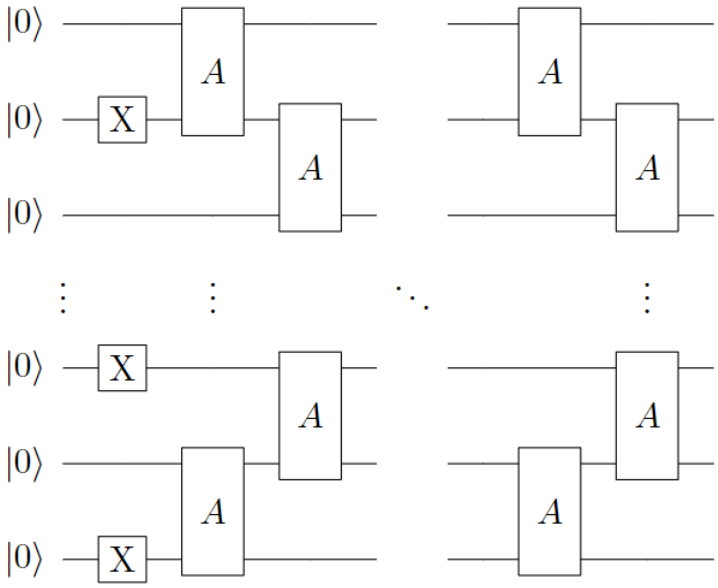

图 4: 基于 Symmetry-Preserved 拟设的单层量子线路

在基础教程中代码示例中使用的都是 UCCSD, 比如计算 LiH 分子:

```
from pychemiq import Molecules,ChemiQ,QMachineType
from pychemiq.Transform.Mapping import jordan_wigner,MappingType
from pychemiq.Optimizer import vqe_solver
from pychemiq.Circuit.Ansatz import UCC
import numpy as np

multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = ["Li     0.00000000    0.00000000    0.37770300",
        "H      0.00000000    0.00000000   -1.13310900"]
mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge)
fermion_LiH = mol.get_molecular_hamiltonian()
pauli_LiH = jordan_wigner(fermion_LiH)

chemiq = ChemiQ()
```

(continues on next page)

(continued from previous page)

```
machine_type = QMachineType.CPU_SINGLE_THREAD
mapping_type = MappingType.Jordan_Wigner
pauli_size = len(pauli_LiH.data())
n_qubits = mol.n_qubits
n_elec = mol.n_electrons
chemiq.prepare_vqe(machine_type,mapping_type,n_elec,pauli_size,n_qubits)

# 设置 ansatz 拟设类型，这里使用的是 UCCSD 拟设
ansatz = UCC("UCCSD",n_elec,mapping_type,chemiq=chemiq)
```

下面我们来演示如何使用 pyChemiQ 调用其他拟设：

```
# 使用 UCCS 拟设
from pychemiq.Circuit.Ansatz import UCC
ansatz = UCC("UCCS",n_elec,mapping_type,chemiq=chemiq)
# 使用 UCCD 拟设
ansatz = UCC("UCCD",n_elec,mapping_type,chemiq=chemiq)

# 使用 HardwareEfficient 拟设
from pychemiq.Circuit.Ansatz import HardwareEfficient
ansatz = HardwareEfficient(n_elec,chemiq = chemiq)

# 使用 SymmetryPreserved 拟设
from pychemiq.Circuit.Ansatz import SymmetryPreserved
ansatz = SymmetryPreserved(n_elec,chemiq = chemiq)
```

指定拟设类型后，就可以自动生成含参的量子线路，下一步就是指定经典优化器与初始参数并迭代求解：

```
method = "SLSQP"
init_para = np.zeros(ansatz.get_para_num())
solver = vqe_solver(
        method = method,
        pauli = pauli_LiH,
        chemiq = chemiq,
        ansatz = ansatz,
        init_para=init_para)
result = solver.fun_val
print(result)
```

在其他参数不变的情况下，使用不同拟设的结果如下：

ansatz	Energy(Hartree)
UCCS	-7.863382128921046
UCCD	-7.882121742611668
UCCSD	-7.882513551487563
HE	-7.8633821289210415
SP	-5.602230693394411

与同基组下的经典 Full CI 结果-7.882526376869 对比，我们发现 UCCD 与 UCCSD 拟设已经达到了化学精度 1.6×10^{-3} Hartree。

参考文献

Chapter 6

算法教程

自从量子计算概念提出以来，人们已经开发了数百种量子算法[1] 。虽然人们对在量子计算机上模拟化学问题十分感兴趣，但目前只有少数量子算法可以作为设计此类量子程序的基石。在应用到材料模拟、生物医药问题的相关量子算法中，许多都需要大量的量子资源，比如大量量子比特或深量子线路。而基于量子-经典混合的算法的变分量子特征值求解算法通过充分利用含噪声的量子比特的计算能力，来实现无完整纠错情况下相关问题的近似和解决。

变分量子特征值求解算法 (Variational Quantum Eigensolver, VQE) 是一种混合的量子经典算法，它使用参数化的量子线路来构造波函数，再通过经典计算机来优化这些参数，使哈密顿量的期望值最小化。它的基本流程如图 1 所示，整体流程分别包括量子初态制备、哈密顿子项 H_i 的测量、求和、收敛性判断和参数优化等过程，其中，量子初态制备、哈密顿子项的测量在量子计算机上进行，即图中淡蓝色部分，其它的步骤如求和和参数优化，由经典计算机完成，即图中淡黄色的部分。

[1] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2(1):1–8, 2016

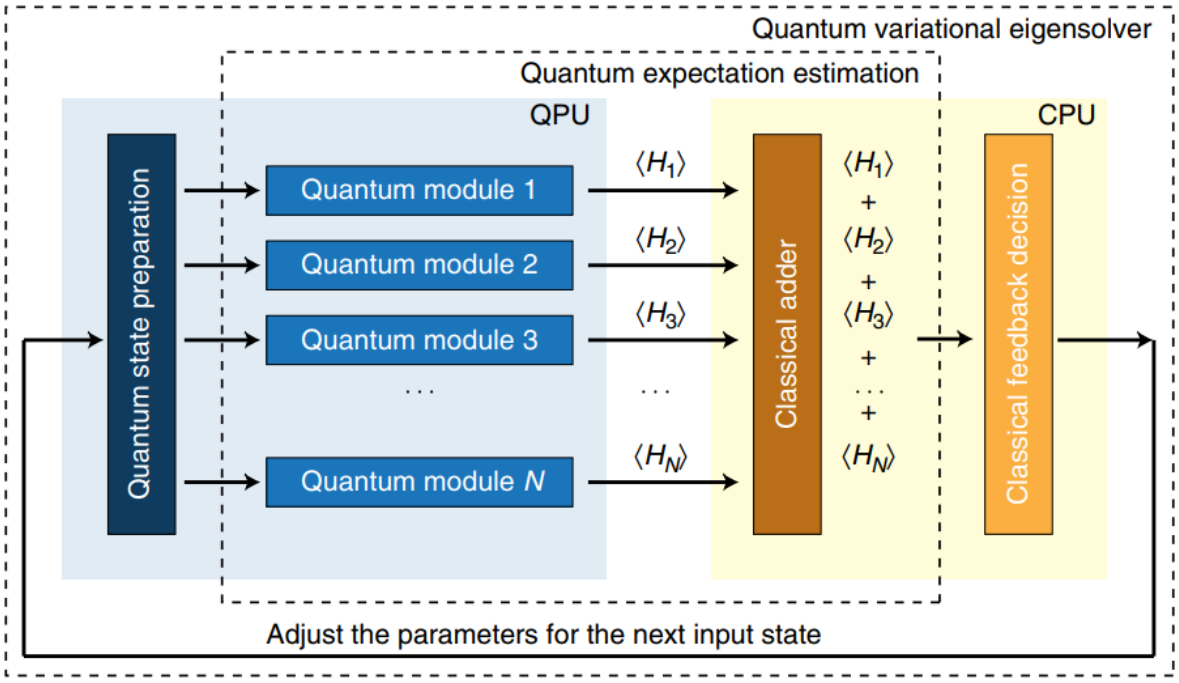

图 1: VQE 算法流程[2]

具体来讲，VQE 可以总结为以下步骤：

(i) 选一组随机参数 θ

(ii) 在量子计算机上制备试验波函数 $|\psi(\theta)\rangle$

(iii) 对哈密顿量的各个子项进行测量，然后在经典计算机上进行求和，得到 $|\psi(\theta)\rangle$ 的哈密顿量的期望值，即得到分子的能量

(iv) 判断该能量是否满足收敛条件，如果满足，则将该能量作为分子基态能量的近似值，终止计算；如果不满足，则变分优化参数，利用经典优化器产生一组新的参数 θ ，重新制备量子初态

(v) 重复，直到能量收敛

此时，参数化量子线路应该制备好哈密顿量的基态，或非常接近基态的状态。与量子相位估计算法 (QPE) 相比，VQE 需要更少的门数和更短的相干时间。它以多项式的重复次数换取所需的相干时间的减少。因此，它更适合于 NISQ 时代。

整个算法中，第一步量子初态制备对后面快速获得正确结果至关重要，特别是应用到化学体系，因为电子是费米子，所以必不可少要做一些额外的预处理。量子初态通常是 Hartree-Fock 态，它由经典计算得到，但不同的映射方法 (mapping) 也会对初态的构造有影响。常用的映射方法有 Jordan-Wigner(JW) 变换、Parity 变换及 Bravyi-Kitaev(BK) 变换等。在确定了映射方式对线路进行初态的构建后，为获得与体系量子终态相近的试验波函数 $|\psi(\theta)\rangle$ ，我们需要一个合适的波函数假设 (Ansatz)。

[2] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L Oąfbrien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014

目前，量子计算化学上试验态的制备方法，主要分为两大类，一类为传统计算化学启发拟设，如幺正耦合簇方法 (Unitary Coupled Cluster, UCC)，另一类是基于量子计算机硬件特性构造的拟设，即 Hardware-Efficient 拟设。一旦选择一个拟设，就可以将其对应的线路在量子计算机上执行，来计算目标函数值，在 VQE 中对应的结果是能量的期望。在经过多步迭代测量至收敛，此时得到的末态是该水平下最接近系统基态的状态，相应的能量也就是求得的基态能量。

在下面 H_2O 分子的案例计算中，我们使用 sto-3g 基组，活性空间 [4,4]，映射方式使用 BK 变换，拟设采用 UCCSD，经典优化器方法使用一阶导数优化方法 L-BFGS-B：

```
from pychemiq import Molecules,ChemiQ,QMachineType
from pychemiq.Transform.Mapping import bravyi_kitaev,MappingType
from pychemiq.Optimizer import vqe_solver
from pychemiq.Circuit.Ansatz import UCC
import numpy as np

# 初始化分子的电子结构参数，包括电荷、基组、原子坐标、自旋多重度、活性空间
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = ["O      0.00000000    0.00000000    0.12713100",
        "H      0.00000000    0.75801600   -0.50852400",
        "H      0.00000000   -0.75801600   -0.50852400"]
active = [4,4]

mol = Molecules(
    geometry = geom,
    basis    = basis,
    multiplicity = multiplicity,
    charge = charge,
    active = active)

# 通过 BK 变换得到泡利算符形式的水分子哈密顿量并打印结果
fermion_H2O = mol.get_molecular_hamiltonian()
pauli_H2O = bravyi_kitaev(fermion_H2O)
print(pauli_H2O)

# 准备量子线路，需要指定的参数有量子虚拟机类型 machine_type，拟设映射类型 mapping_type，
# 泡利哈密顿量的项数 pauli_size，电子数目 n_elec 与量子比特的数目 n_qubits
chemiq = ChemiQ()
machine_type = QMachineType.CPU_SINGLE_THREAD
mapping_type = MappingType.Bravyi_Kitaev
pauli_size = len(pauli_H2O.data())
n_qubits = mol.n_qubits
n_elec = mol.n_electrons
chemiq.prepare_vqe(machine_type,mapping_type,n_elec,pauli_size,n_qubits)
```

(continues on next page)

(continued from previous page)

```
# 设置簇算符需要的映射方法及簇算符类型，这里我们使用 UCCSD 拟设
ansatz = UCC("UCCSD",n_elec,mapping_type,chemiq=chemiq)

# 指定经典优化器与初始参数并迭代求解
method = "L-BFGS-B"
init_para = np.zeros(ansatz.get_para_num())
solver = vqe_solver(
    method = method,
    pauli = pauli_H2O,
    chemiq = chemiq,
    ansatz = ansatz,
    init_para=init_para)
result = solver.fun_val
n_calls = solver.fcalls
print(result,f"函数共调用{n_calls}次")

energies = chemiq.get_energy_history()
print(energies)
```

打印得到的结果为：

```
-74.97462360159876 调用函数 16 次
[-74.96590114589256, -74.93763769775363, -74.97445942068707, -74.97445942068707, -74.
↪97411682452937, -74.9746226763453, -74.9746226763453, -74.97462062772358, -74.
↪97462337673937, -74.97462337673937, -74.97462142026288, -74.97462351765488, -74.
↪97462351765488, -74.974622639902, -74.97462360159876, -74.97462360159876]
```

为了对比 pyChemiq 的计算精度，我们将结果与经典计算化学软件 PySCF[3] 的结果做了比较 (PySCF 的安装详见 官网 。在 PySCF 中我们使用了相同的基组和方法 (VQE 中 UCCSD ansatz 对应经典的 CISD 方法)，代码如下：

```
from pyscf import gto, scf

atom = '''
O                  0.00000000    0.00000000    0.12713100
H                  0.00000000    0.75801600   -0.50852400
H                  0.00000000   -0.75801600   -0.50852400
'''

mol = gto.M(atom=atom,    # in Angstrom
```

(continues on next page)

[3] Qiming Sun, Timothy C Berkelbach, Nick S Blunt, George H Booth, Sheng Guo, Zhendong Li, Junzi Liu, James D McClain, Elvira R Sayfutyarova, Sandeep Sharma, et al. Pyscf: the python-based simulations of chemistry framework. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 8(1):e1340, 2018.

(continued from previous page)

```
    basis='STO-3G',
    charge=0,
    spin=0)
myhf = mol.RHF().run()
mycas = myhf.CASCI(4, 4).run()
E_CISD = mycas.e_tot
print(E_CISD)
```

得到的结果如下：

```
converged SCF energy = -74.9659011458929
CASCI E = -74.9746354406465  E(CI) = -6.11656024435146  S^2 = 0.0000000
-74.9746354406465
```

我们将 pyChemiQ 打印出来的数据作图，与同水平下的经典 CISD 结果进行对比。可以看到随着函数迭代次数的增加，电子能量逐渐收敛至经典结果的能量，如图 2 所示。而且当函数迭代到第五次时电子能量已经达到了化学精度 1.6×10^{-3} Hartree。

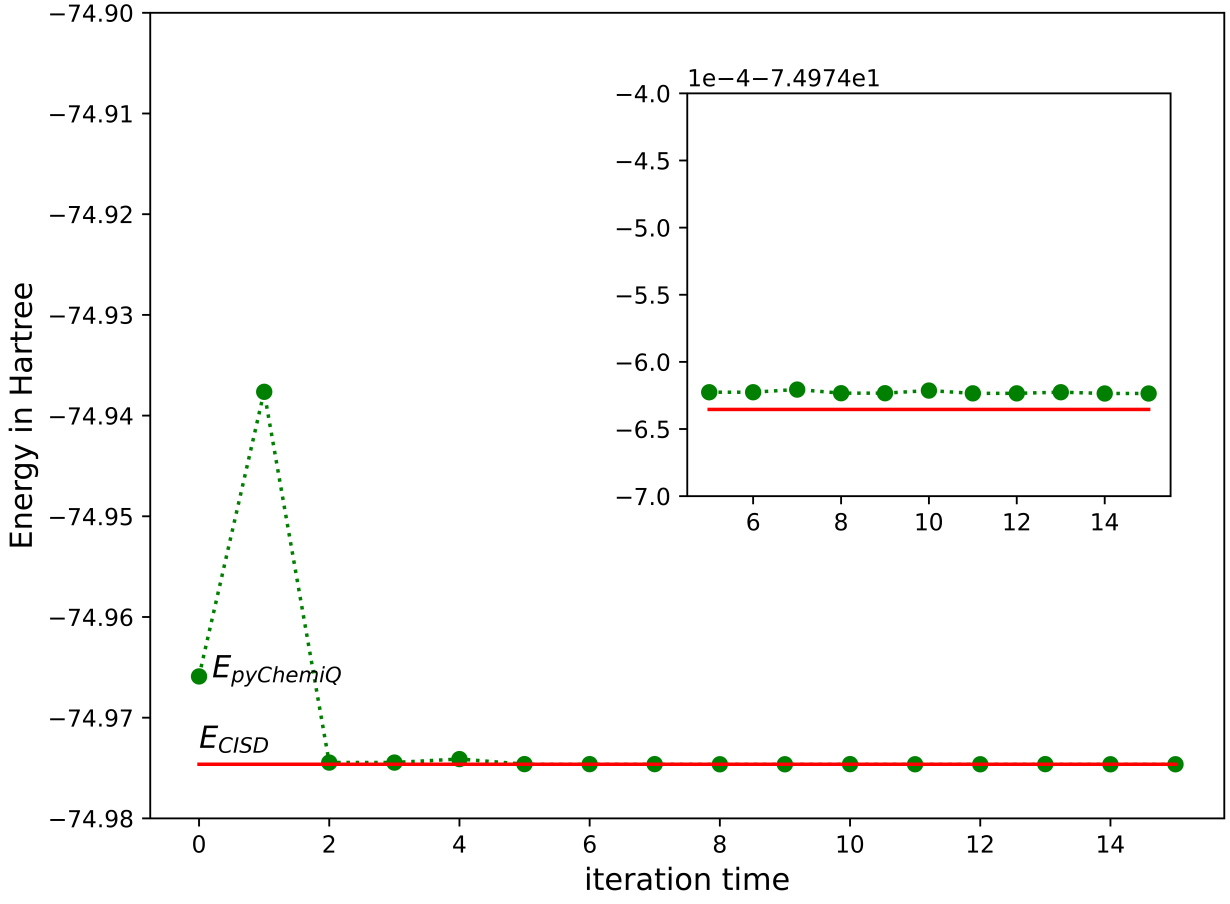

图 2: 水分子能量收敛曲线

参考文献

Chapter 7

功能教程

我们都知道化学反应的本质是旧化学键断裂和新化学键形成的过程。而化学键的“断裂”和“形成”只是我们宏观上对其定性的描述。实际上，我们之所以说化学键“形成”，是因为分子间原子沿着反应坐标的方向相互靠近，处于某一相对位置时整个体系的能量最低，最稳定。而化学键“断裂”则是在外界输入能量后，分子间的原子克服了能垒，沿着反应坐标的方向相互远离。在计算化学中，我们可以用势能面 (Potential Energy Surface, PES) 来描述分子在其原子的不同位置的能量，从而帮助我们理解化学反应的进程。

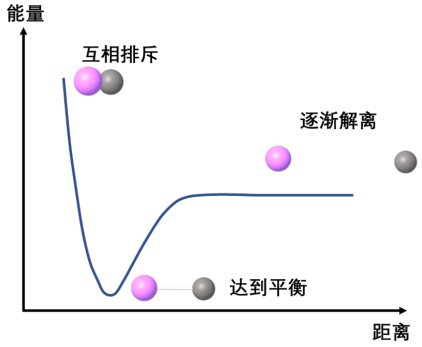

图 1: 双分子原子的势能面

比如，随着分子内某一根键的增长，能量会随着变化，做能量-键长的变化曲线，称为势能曲线，如图 1；如果做分子的势能随两种坐标参数变化的图像，你会发现这是一个面（因为共有 3 个量：两种坐标变量加能量，组成三维空间），就叫势能面，如图 2；以此类推，整个分子势能随着所有可能的原子坐标变量变化，是一个在多维空间中的复杂势能面 (hypersurface)，统称势能面[1] 。

[1] Baidu. https://baike.baidu.com/item/%E5%8A%BF%E8%83%BD%E9%9D%A2/6295493, last access on 6th January, 2023

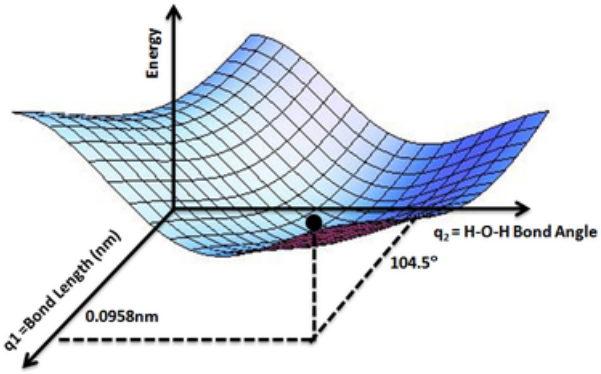

图 2: 水分子势能面: 势能最低点对应优化后的水分子结构，O-H 键长 0.0958 nm，H-O-H 夹角为 104.5° 。图引自[2]

在这个教程中，我们会演示如何使用 pyChemiQ 得到分子的势能面的数据，并使用 matplotlib 绘制势能面。这里我们以双原子分子 H_2 为例，映射方式使用 JW 变换，拟设采用 UCCSD，经典优化器使用 SLSQP，最后我们再将 pyChemiQ 的势能曲线与 PySCF 的势能曲线进行对比：

```
# 导入所需的包
from pychemiq import Molecules,ChemiQ,QMachineType
from pychemiq.Transform.Mapping import jordan_wigner,MappingType
from pychemiq.Optimizer import vqe_solver
from pychemiq.Circuit.Ansatz import UCC
import numpy as np
from pyscf import gto, scf, fci
import matplotlib.pyplot as plt

# 进行势能面扫描：先初始化参数，再构建不同键长下的分子体系，进行多次单点能计算
basis = 'sto-3g'
multiplicity = 1
charge=0

## 定义步长间隔、步数
bond_length_interval = 0.1
n_points = 40
bond_lengths = []
energies = []
for point in range(3, n_points + 1):
    bond_length = bond_length_interval * point
    bond_lengths += [bond_length]
    geometry = ["H 0 0 0", f"H 0 0 {bond_length}"]

    mol = Molecules(
```

(continues on next page)

[2] Wikipedia. Potential energy surface. https://en.wikipedia.org/wiki/Potential_energy_surface, last access on 6th January, 2023

(continued from previous page)

```
        geometry = geometry,
        basis    = basis,
        multiplicity = multiplicity,
        charge = charge)

    fermion_H2 = mol.get_molecular_hamiltonian()
    pauli_H2 = jordan_wigner(fermion_H2)

    chemiq = ChemiQ()
    machine_type = QMachineType.CPU_SINGLE_THREAD
    mapping_type = MappingType.Jordan_Wigner
    pauli_size = len(pauli_H2.data())
    n_qubits = mol.n_qubits
    n_elec = mol.n_electrons
    chemiq.prepare_vqe(machine_type,mapping_type,n_elec,pauli_size,n_qubits)
    ansatz = UCC("UCCSD",n_elec,mapping_type,chemiq=chemiq)

    method = "SLSQP"
    init_para = np.zeros(ansatz.get_para_num())
    solver = vqe_solver(
            method = method,
            pauli = pauli_H2,
            chemiq = chemiq,
            ansatz = ansatz,
            init_para=init_para)
    energy = solver.fun_val
    energies += [energy]

# 使用经典计算化学软件 PySCF 的 FCI 方法来计算氢分子在不同键长下的能量
pyscf_energies = []
bond_length_interval = 0.1
n_points = 40
for point in range(3, n_points + 1):
    bond_length = bond_length_interval * point
    atom = f'H 0 0 0; H 0 0 {bond_length}'

    mol = gto.M(atom=atom,   # in Angstrom
            basis='STO-3G',
            charge=0,
            spin=0)
    myhf = scf.HF(mol).run()
    cisolver = fci.FCI(myhf)
    pyscf_energies += [cisolver.kernel()[0]]
```

(continues on next page)

(continued from previous page)

```
# 最后我们使用 matplotlib 来绘制氢分子势能面
plt.figure()
plt.plot(bond_lengths, energies, '-g',label='pyChemiQ')
plt.plot(bond_lengths, pyscf_energies, '--r',label='PySCF')
plt.ylabel('Energy in Hartree')
plt.xlabel('Bond length in angstrom')
plt.legend()
plt.show()
```

得到的氢分子势能图对比如下图所示，由于二者计算结果过于接近，势能面大部分处于重合的状态。

图 3: pyChemiQ 与 PySCF 得到的氢分子势能面

参考文献

Chapter 8

算符类教程

8.1 费米子算符类

我们在基础教程的第一节已经介绍过费米子的产生算符 $a_p^\dagger$ 和湮灭算符 a_q 。如果我们要自定义哈密顿量或者拟设，可以使用 pyChemiQ 中的费米子算符类与泡利算符类。在 pyChemiQ 的费米子算符类中，$P+$ 表示产生算符 $a_p^\dagger$ ，用 Q 表示湮灭算符 a_q 。例如”1+ 3 5+ 1”代表 $a_1^\dagger a_3 a_5^\dagger a_1$ 。FermionOperator 类也可以同时存储这一项的系数，例如构建费米子项 $2a_0a_1^\dagger$ 、$3a_2^\dagger a_3$ 和 $a_1^\dagger a_3 a_5^\dagger a_1$ ：

```
from pychemiq import FermionOperator
a = FermionOperator("0 1+", 2)
b = FermionOperator("2+ 3", 3)
c = FermionOperator("1+ 3 5+ 1",1)
```

若要构建多个费米子项的加减，则采用字典序的形式构建多个表达式。例如 p0 构建 $2a_0a_1^\dagger + 3a_2^\dagger a_3$ 。除此之外，我们还可以构造一个空的费米子算符类 p1，里面不包含任何产生和湮没算符及常数项；或者只构造电子之间排斥能常数项，例如 p2；也可以通过已经构造好的费米子算符来构造它的一份副本，例如 p3。

```
p0 = FermionOperator({"0 1+": 2, "2+ 3": 3})
p1 = FermionOperator()
p2 = FermionOperator(2)
p3 = p2
```

FermionOperator 类也提供了费米子算符之间加、减和乘的基础的运算操作。计算返回的结果还是一个费米子算符类。该类同样也支持打印功能，可以直接将费米子算符类打印输出到屏幕上。

```
plus = a + b
minus = a - b
multiply = a * b
print("a + b = {}".format(plus))
```

(continues on next page)

(continued from previous page)

```
print("a - b = {}".format(minus))
print("a * b = {}".format(multiply))
```

通过使用上述示例代码，$a+b$，$a-b$ 和 $a*b$ 的计算结果如下：

```
a + b = {
0 1+ : 2.000000
2+ 3 : 3.000000
}
a - b = {
0 1+ : 2.000000
2+ 3  : -3.000000
}
a * b = {
0 1+ 2+ 3 : 6.000000
}
```

还可以通过 normal_ordered 接口对费米子算符进行整理。在这个转换中规定所作用的轨道编码从高到低进行排序，并且产生算符出现在湮没算符之前。整理规则如下：对于两个相同数字，交换湮没和产生算符，等价于单位 1 减去正规序。如果同时存在两个相同的产生或湮没算符，则该项表达式等于 0(泡利不相容原理)；对于不同的数字，整理成正规序，需要将正规序的系数变为相反数 (费米子算符的反对易关系)。例如表达式“0 1+ 2+ 3”，整理成正规序“2+ 1+ 3 0”，相当于不同数字交换了 4 次，系数不变。

```
print(multiply.normal_ordered())
{
2+ 1+ 3 0 : 6.000000
}
```

此外，费米子算符类还提供了 data 接口，可以返回费米子算符内部维护的数据。

```
print("data = {}".format(a.data()))
data = [(([(0, False), (1, True)], '0 1+'), (2+0j))]
```

8.2 泡利算符类

泡利算符是一组三个 2×2 的酉正厄密复矩阵，又称酉矩阵。我们一般都以希腊字母 σ 来表示，记作 σ_x，σ_y，σ_z。在量子线路中我们称它们为 X 门，Y 门，Z 门。它们对应的矩阵形式如下所示：

Pauli-X 门：

$$X = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Pauli-Y 门：

$$Y = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Pauli-Z 门：

$$Z = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

以上三个泡利 (Pauli matrices) 有时也被称作自旋矩阵 (spin matrices)。可观察到 Pauli-X 门相当于 NOT 门，其将 $|0\rangle \to |1\rangle, |1\rangle \to |0\rangle$ 。Pauli-Y 门的作用相当于绕布洛赫球的 Y 轴旋转角度 π 。Pauli-Z 门的作用相当于绕布洛赫球的 Z 轴旋转角度 π 。

泡利算符具有如下性质：

1. 泡利算符与自身相乘得到是单位矩阵

$$\sigma_x\sigma_x = I$$
$$\sigma_y\sigma_y = I$$
$$\sigma_z\sigma_z = I$$

2. 顺序相乘的两个泡利算符跟未参与计算的泡利算符是 i 倍的关系

$$\sigma_x\sigma_y = i\sigma_z$$
$$\sigma_y\sigma_z = i\sigma_x$$
$$\sigma_z\sigma_x = i\sigma_y$$

3. 逆序相乘的两个泡利算符跟未参与计算的泡利算符是 $-i$ 倍的关系

$$\sigma_y\sigma_x = -i\sigma_z$$
$$\sigma_z\sigma_y = -i\sigma_x$$
$$\sigma_x\sigma_z = -i\sigma_y$$

pyChemiQ 中实现了泡利算符类 PauliOperator。我们可以很容易的构造泡利算符类，例如构造一个空的泡利算符项，如 p1；或者构造带系数的泡利算符直积项 $2\sigma_z^0\sigma_z^1$, 如 p2。这里泡利算符右上角的数字代表作用在具体的量子比特，这一项代表的意义的是一个 Pauli-Z 门作用在量子比特 0 张乘一个 Pauli-Z 门作用在量子比特 1 上，该项的系数为 2；若要构建多个泡利算符直积项的加和，可以采用字典序的形式，如 p3 构建的是 $2\sigma_z^0\sigma_z^1 + 3\sigma_x^1\sigma_y^2$ ；或者构造一个如 p4 的单位矩阵，其系数为 5，也可以用如 p5 的形式来构建，二者等价。

```
from pychemiq import PauliOperator
p1 = PauliOperator()
p2 = PauliOperator("Z0 Z1", 2)
p3 = PauliOperator({"Z0 Z1": 2, "X1 Y2": 3})
p4 = PauliOperator(5)
p5 = PauliOperator("", 5)
```

注: 构造泡利算符类的时候，字符串里面包含的字符只能是空格、*X*、*Y* 和 *Z* 中的一个或多个，包含其它字符将会抛出异常。另外，同一个字符串里面同一泡利算符的比特索引不能相同，例如：*PauliOperator(* "*Z0 Z0*"，*2)* 将会抛出异常。

同费米子算符类一样，泡利算符类之间可以做加、减、乘等操作，计算返回结果还是一个泡利算符类。而且也支持打印功能，我们可以将泡利算符类打印输出到屏幕上，方便查看其值。

```
a = PauliOperator("Z0 Z1", 4)
b = PauliOperator("X5 Y6", 3)
plus = a + b
minus = a - b
muliply = a * b
print(plus)
```

在实际使用的时候，我们常常需要知道该泡利算符项操作了多少个量子比特，这时候我们通过调用泡利算符类的接口 get_max_index() 得到。如果是空的泡利算符项调用 get_max_index() 接口则返回 SIZE_MAX（具体值取决于操作系统），否则返回其最大索引值。在下面的例子里，前者输出的值为 1，后者输出的值为 6。

```
a = PauliOperator("Z0 Z1", 2)
b = PauliOperator("X5 Y6", 3)
print(a.get_max_index())
print(b.get_max_index())
```

此外，泡利算符类也提供了 data 接口，可以返回泡利算符内部维护的数据。

```
print("data = {}".format(a.data()))
data = [(({0: 'Z', 1: 'Z'}, 'Z0 Z1'), (2+0j))]
```

Chapter 9

优化器教程

在拟设教程中我们构造了含参数的拟设线路，进行量子期望估计后，需要不断迭代优化 Ansatz 中涉及的参数以获取最低的能量，并以此能量最低的叠加态作为当前分子模型的基态。VQE 中对这些参数的优化是利用经典优化器来处理的，截止目前，pyChemiQ 提供了以下几种优化器：NELDER-MEAD、POWELL、COBYLA、L-BFGS-B、SLSQP 和 Gradient-Descent。其中无导数优化方法为 NELDER-MEAD、POWELL、COBYLA；一阶方法为 L-BFGS-B、SLSQP 和 Gradient-Descent。在 pyChemiQ 中，我们通过在最后的 VQE 求解器中的 method 来指定不同的经典优化器。

```
# 用 method 指定经典优化器与初始参数并迭代求解
#method = "NELDER-MEAD"
#method = "POWELL"
#method = "COBYLA"
#method = "L-BFGS-B"
#method = "Gradient-Descent"
method = "SLSQP"
init_para = np.zeros(ansatz.get_para_num())
solver = vqe_solver(
        method = method,
        pauli = pauli_H2,
        chemiq = chemiq,
        ansatz = ansatz,
        init_para=init_para)
result = solver.fun_val
n_calls = solver.fcalls
print(result,f"函数共调用{n_calls}次")
```

除了使用 pyChemiQ 自带的优化器外，我们也可以调用外部的 python 库来实现经典优化部分，这里我们以调用 scipy.optimize 为例。首先我们要先要利用 chemiq.getLossFuncValue() 来得到损失函数：

```
# 优化器使用外部 scipy 库的 SLSQP，这时我们需要先定义损失函数
```

(continues on next page)

(continued from previous page)

```
def loss(para,grad,iters,fcalls):
    res = chemiq.getLossFuncValue(0,para,grad,iters,fcalls,pauli,chemiq.qvec,ansatz)
    return res[1]

def optimScipy():
    import scipy.optimize as opt
    options = {"maxiter":2}
    init_grad = np.zeros(ansatz.get_para_num())
    method = "SLSQP"
    res = opt.minimize(loss,init_para,
            args=(init_grad,0,0),
            method = method,
            options = options)
    final_results = {
            "energy":res.fun,
            "fcalls":f"函数共调用{res.nfev}次",
    }
    print(final_results)

# 在主函数中指定 loss 函数中的泡利哈密顿量 pauli 和优化器的初始参数 init_para
if __name__ == "__main__":
    pauli = pauli_H2
    init_para = np.zeros(ansatz.get_para_num())
    optimScipy()
```

如果不想使用现有的优化器，我们也可以自己定义优化器，下面我们以梯度下降法为例。首先我们要先利用 chemiq.getExpectationValue() 得到期望值来定义损失函数：

```
# 优化器使用自定义的梯度下降法，首先定义损失函数：
def loss3(para,grad):
    new_para[:] = para[:]
    global result
    result = chemiq.getExpectationValue(0,0,0,chemiq.qvec,H,new_para,ansatz,False)
    if len(grad) == len(para):
        for i in range(len(para)):
            new_para[i] += delta_p
            result_p = chemiq.getExpectationValue(0,0,0,chemiq.qvec,H,new_para,ansatz,
↪False)
            grad[i] = (result_p - result)/delta_p
            new_para[i] -= delta_p
    return result
def GradientDescent():
    para_num = ansatz.get_para_num()
    seed = 20221115
```

(continues on next page)

(continued from previous page)

```
    np.random.seed(seed)
    para = np.zeros(para_num)
    grad = np.zeros(para_num)
    lr = 0.1
    result_previous = 0
    result = 0
    threshold = 1e-8
    for i in range(1000):
        result_previous = result
        result = loss3(para,grad)
        para -= lr*grad
        if abs(result - result_previous) < threshold:
            print("final_energy :",result)
            print("iterations :",i+1)
            break
# 在主函数中指定 loss 函数中的泡利哈密顿量 H 和参数 delta_p，以及初始参数 new_para
if __name__ == "__main__":
    new_para = np.zeros(ansatz.get_para_num())
    delta_p = 1e-3
    H = pauli_H2.to_hamiltonian(True)
    GradientDescent()
```

Chapter 10

量子线路教程

在基础教程中的拟设教程我们展示了如何使用 Unitary Coupled-Cluster、Hardware-Efficient、Symmetry-Preserved 接口来直接搭建量子线路，这个教程展示的是如何使用用户自定义的方式来构建量子线路拟设。

调用 pychemiq.Circuit.Ansatz 模块下的 UserDefine 函数，我们可以通过以下两种方式自定义构建线路：第一种是通过将 originIR 格式的代码输入到 circuit 参数中自定义量子线路，第二种是通过输入耦合簇激发项的费米子算符 fermion 参数来构建量子线路。该函数的接口介绍如下：

UserDefine(*n_electrons*, *circuit=None*, *fermion=None*, *chemiq=None*)

使用用户自定义的方式构建量子线路拟设。

Parameters

- **n_electrons** (*int*) –输入分子体系的电子数。
- **circuit** (*str*) –构建量子线路的 originIR 字符串。
- **fermion** (FermionOperator) –构建量子线路的费米子算符类。
- **chemiq** (ChemiQ) –指定 chemiq 类。详见 pychemiq.ChemiQ。

Returns

输出自定义拟设的 AbstractAnsatz 类。

1. 通过将 originIR 输入到 circuit 参数中自定义量子线路

我们可以通过两种方法获得 originIR 格式的量子线路，第一种方法是通过本源量子云平台的图形化编程界面搭建量子线路。如下图，先在本源量子云平台 图形化编程界面 搭建量子线路。

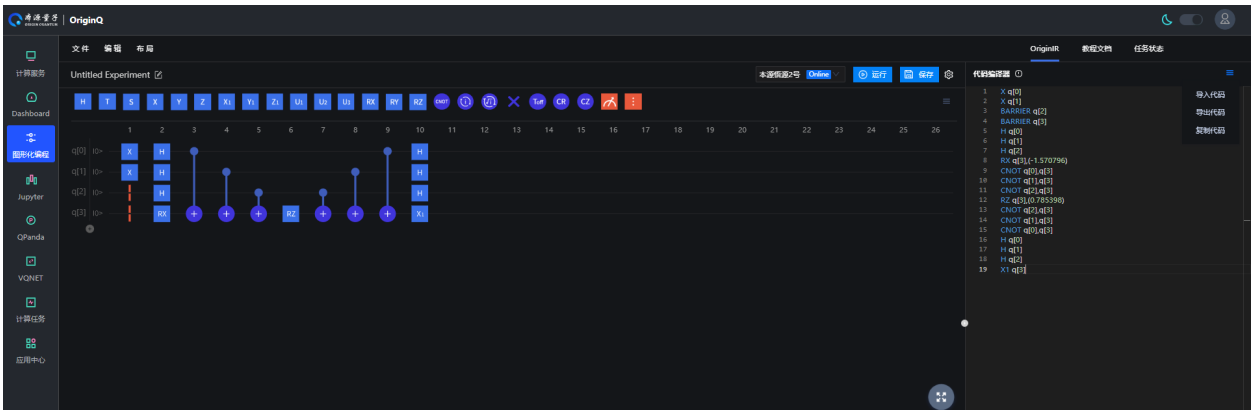

图 1: 通过本源量子云平台图形化编程界面搭建量子线路

搭建完量子线路后导出如下的 originIR 格式的字符串，再将其输入到 UserDefine 函数中的 circuit 参数即可。

```
X q[0]
X q[1]
BARRIER q[2]
BARRIER q[3]
H q[0]
H q[1]
H q[2]
RX q[3],(-1.570796)
CNOT q[0],q[3]
CNOT q[1],q[3]
CNOT q[2],q[3]
RZ q[3],(0.785398)
CNOT q[2],q[3]
CNOT q[1],q[3]
CNOT q[0],q[3]
H q[0]
H q[1]
H q[2]
X1 q[3]
```

接口示例：

```
from pychemiq import Molecules,ChemiQ,QMachineType
from pychemiq.Transform.Mapping import jordan_wigner,MappingType
from pychemiq.Optimizer import vqe_solver
from pychemiq.Circuit.Ansatz import UserDefine
import numpy as np

multiplicity = 1
```

(continues on next page)

(continued from previous page)

```
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"
mol = Molecules(
      geometry = geom,
      basis    = basis,
      multiplicity = multiplicity,
      charge = charge)
fermion_H2 = mol.get_molecular_hamiltonian()
pauli_H2 = jordan_wigner(fermion_H2)

chemiq = ChemiQ()
machine_type = QMachineType.CPU_SINGLE_THREAD
mapping_type = MappingType.Jordan_Wigner
pauli_size = len(pauli_H2.data())
n_qubits = mol.n_qubits
n_elec = mol.n_electrons
chemiq.prepare_vqe(machine_type,mapping_type,n_elec,pauli_size,n_qubits)

# 使用自定义量子线路，将 originIR 格式的字符串输入到 circuit 参数中
circuit = """
    X q[0]
    X q[1]
    BARRIER q[2]
    BARRIER q[3]
    H q[0]
    H q[1]
    H q[2]
    RX q[3],(-1.5707963)
    CNOT q[0],q[3]
    CNOT q[1],q[3]
    CNOT q[2],q[3]
    RZ q[3],(0.785398)
    CNOT q[2],q[3]
    CNOT q[1],q[3]
    CNOT q[0],q[3]
    H q[0]
    H q[1]
    H q[2]
    RX q[3],(1.5707963)
"""
ansatz = UserDefine(n_elec, circuit=circuit, chemiq=chemiq)
```

(continues on next page)

(continued from previous page)

```
# 最后指定经典优化器与初始参数并迭代求解
method = "SLSQP"
init_para = np.zeros(ansatz.get_para_num())
solver = vqe_solver(
        method = method,
        pauli = pauli_H2,
        chemiq = chemiq,
        ansatz = ansatz,
        init_para=init_para)
result = solver.fun_val
print(result)
```

打印得到的结果为：0.7151043390810803。这里的两个 RX 门为固定参数，不参与变分线路的参数优化。对于有参数的旋转门，默认参数不为 $\pi/2$ 或 $-\pi/2$ 的是待优化的参数。

第二种获得 originIR 格式的量子线路是通过 pyqpanda 中的 convert_qprog_to_originir 函数获得，详细教程见 量子程序转化 OriginIR 。这里我们以拟设教程中提到的 Hardware-Efficient 拟设的单层线路为例，演示如何通过量子程序获得 OriginIR。下面我们先构建 HE 拟设线路 QProg，再通过函数 convert_qprog_to_originir(prog, machine) 将其转换成 originIR 格式。

```
import pyqpanda as pq
import numpy as np

def HE_ansatz(machine_type,qn, para):
    machine = pq.init_quantum_machine(machine_type)
    qlist=pq.qAlloc_many(qn)

    # 构建 HE 拟设线路
    prog = pq.QProg()
    for i in range(qn):
        prog.insert(pq.RZ(qlist[i], para[4*i]))
        prog.insert(pq.RX(qlist[i], para[4*i+1]))
        prog.insert(pq.RZ(qlist[i], para[4*i+2]))

    for j in range(qn-1):
        ry_control = pq.RY(qlist[j+1], para[4*j+3]).control(qlist[j])
        prog.insert(ry_control)

    ry_last = pq.RY(qlist[0], para[4*qn-1]).control(qlist[qn-1])
    prog.insert(ry_last)

    #print(prog)
    OriginIR=pq.convert_qprog_to_originir(prog, machine)
    print(OriginIR)
```

(continues on next page)

(continued from previous page)

```
    return OriginIR
```

下面我们定义主函数来获得该参数下的 originIR 格式的量子线路。这里我们以四个量子比特为例：

```
if __name__ == "__main__":
    machine_type = pq.QMachineType.CPU
    qn=4
    para=np.random.random(4*qn)
    HE_ansatz(machine_type,qn, para)
```

打印得到的结果为：

```
QINIT 4
CREG 0
RZ q[0],(0.6639123)
RX q[0],(0.69876429)
RZ q[0],(0.87923246)
RZ q[1],(0.50633782)
RX q[1],(0.57366393)
RZ q[1],(0.51500428)
RZ q[2],(0.41510053)
RX q[2],(0.58136057)
RZ q[2],(0.60506401)
RZ q[3],(0.99153126)
RX q[3],(0.89568316)
RZ q[3],(0.6493124)
CONTROL q[0]
RY q[1],(0.011800026)
ENDCONTROL
CONTROL q[1]
RY q[2],(0.92157183)
ENDCONTROL
CONTROL q[2]
RY q[3],(0.64791654)
ENDCONTROL
CONTROL q[3]
RY q[0],(0.50756615)
ENDCONTROL
```

将前两行删去后即可输入到 UserDefine 函数中的 circuit 参数中，如第一种方式所示。

2. 通过输入耦合簇激发项的费米子算符 fermion 参数来构建量子线路

第二种方式是通过输入耦合簇激发项的费米子算符 fermion 参数来构建量子线路。例如，对于 4 个量子比特，2 电子体系的双激发耦合簇算符，自旋轨道 0 和 1 为占据态，激发后的耦合簇项为：01->23。

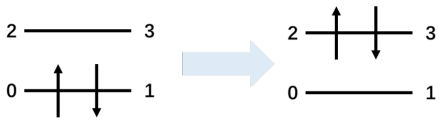

图 2: 四个自旋轨道的氢分子体系由基态到双激发态

如要构建如上的激发费米子算符我们需要用 FermionOperator 来构建或者通过调用 pychemiq.Utils 模块中的函数 get_cc() 来构建。

```
from pychemiq import FermionOperator
a = FermionOperator("3+ 2+ 1 0", 1)
print(a)

from pychemiq.Utils import get_cc_n_term,get_cc
import numpy as np
n_para = get_cc_n_term(4,2,"CCD")
para = np.ones(n_para)
cc_fermion = get_cc(4,2,para,"CCD")
print(cc_fermion)
```

二者打印的结果都为:

```
{
3+ 2+ 1 0 : 1.000000
}
```

将得到的激发费米子算符输入到 UserDefine 函数中的 fermion 参数即可。这里我们以氢分子为例:

接口示例:

```
from pychemiq import Molecules,ChemiQ,QMachineType,FermionOperator
from pychemiq.Transform.Mapping import jordan_wigner,MappingType
from pychemiq.Optimizer import vqe_solver
from pychemiq.Circuit.Ansatz import UserDefine
import numpy as np

multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"
mol = Molecules(
      geometry = geom,
      basis    = basis,
      multiplicity = multiplicity,
      charge = charge)
```

(continues on next page)

(continued from previous page)

```
fermion_H2 = mol.get_molecular_hamiltonian()
pauli_H2 = jordan_wigner(fermion_H2)

chemiq = ChemiQ()
machine_type = QMachineType.CPU_SINGLE_THREAD
mapping_type = MappingType.Jordan_Wigner
pauli_size = len(pauli_H2.data())
n_qubits = mol.n_qubits
n_elec = mol.n_electrons
chemiq.prepare_vqe(machine_type,mapping_type,n_elec,pauli_size,n_qubits)

# 使用自定义量子线路，将自定义的激发费米子算符输入到 fermion 参数中
a = FermionOperator("3+ 2+ 1 0", 1)
ansatz = UserDefine(n_elec, fermion=a, chemiq=chemiq)

# 最后指定经典优化器与初始参数并迭代求解
method = "SLSQP"
init_para = np.zeros(ansatz.get_para_num())
solver = vqe_solver(
        method = method,
        pauli = pauli_H2,
        chemiq = chemiq,
        ansatz = ansatz,
        init_para=init_para)
result = solver.fun_val
print(result)
```

打印得到的结果为：-1.1372838304374302

Chapter 11

计算化学

随着量子化学理论的不断完善，计算化学已经成了化学工作者解释实验现象、预测实验结果、指导实验设计的重要工具，在药物的合成、催化剂的制备等方面有着广泛的应用。但是，面对计算化学所涉及的巨大计算量，经典计算机在计算精度、计算尺寸等方面显得能力有限，这就在一定程度上限制了计算化学的发展。而费曼曾提出：可以创造一个与已知物理系统条件相同的系统，让它以相同的规律演化，进而获得我们自己想要的信息。费曼的这一猜想提示我们——既然化学所研究的体系是量子体系，我们何不在量子计算机上对其进行模拟呢?

就目前的有噪声的中等规模量子 (NISQ) 计算机而言，可以通过变分量子特征值求解算法 (Variational Quantum Eigensolver，简称 VQE)，在量子计算机上实现化学模拟。该算法作为用于寻找一个较大矩阵的特征值的量子与经典混合算法，不仅能保证量子态的相干性，其计算结果还能达到化学精度。

计算化学，顾名思义，就是利用数学方法通过计算机程序对化学体系进行模拟计算，以解释或解决化学问题。早期由于计算能力较弱，化学研究主要以理论和实验交互为主。但随着科学技术的蓬勃发展、量子化学理论的不断完善，计算已经成为一种独立的科研手段，与理论和实验相为验证，密不可分。如今，计算化学对于化学工作者来说，已经成了解释实验现象、预测实验结果、指导实验设计的重要工具，在材料科学、纳米科学、生命科学等领域得到了广泛的应用。

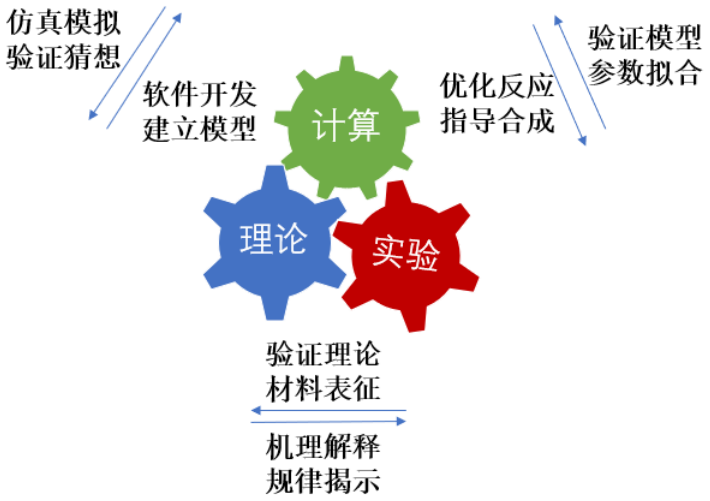

图 1: 计算、理论与实验的相互关系

计算化学主要包括两个分支——量子化学与分子模拟，其理论基础分别源自量子力学 (Quantum Mechanics, QM) 与分子力学 (Molecular Mechanics, MM)。

理论量子化学主要是使用量子力学 (QM) 来计算分子系统中电子的状态和性质。通俗意义上讲就是通过求解薛定谔方程来得到分子系统中电子的运动状态，而这些运动状态决定了其分子的化学性质。它们决定了一个化学反应是否能发生。精确求解电子 Hamiltonian 需要巨大的算力，对于小分子求解尚可进行，但是无法计算大分子。这时就需要基于经验参数的分子力场 (Molecular Field) 方法。

在分子力场或分子力学 (MM) 方法中，分子被视为机械连接的原子系统。与量子力学方法不同的是，电子的具体分布没有被精确地计算，而是与原子核一起作为有效的原子 (effective atoms)。由于分子力场的方法中，每个键的类型和参数都是在计算前根据体系被选定好的，它可以在典型的键合距离 (bonding distance) 下很好地描述大型的分子系统。但不可能描述发生键断裂和形成的反应或转化。换句话说，分子力场可以模拟大分子系统，但是无法精确求解其具体电子的分布，而这正是化学反应的关键所在。

这里我们主要聚焦量子化学方法。化学行为的本质是分子中电子运动的结果，而电子的运动遵循量子力学，所以量子化学是一个量子计算天然的用武之地。传统的量子化学主要使用经典计算机来计算模拟量子体系，但量子系统的波函数的复杂度随粒子数呈指数增长。这种指数级的增长使得经典计算机在处理类似问题时很棘手，比如当前人们对多体物理和强关联系统等问题还无法进行精确求解。基于量子力学原理的量子计算机给计算化学开辟了新的方向，由于量子比特的量子性质，可以实现叠加和纠缠，因此可以有效地存储量子系统的波函数。同时我们需要在计算化学中运用量子算法来加快运算效率，从而研究用经典算法有限甚至不可能计算的系统。

Chapter 12

理论基础

12.1 2.1 电子结构问题

在对一个 N 电子体系计算模拟时，往往需要求解薛定谔方程，这就涉及到了 $3N$ 维波函数的求解。含时 (time-dependent) 薛定谔方程如式 (1) 所示[1]，其中 $\hat{\mathbf{H}}$ 是系统的 Hamiltonian，它描述了整个系统的全部信息. $\psi(x,t)\rangle$ 是系统随时间变化的多体波函数。当粒子处于与时间无关的势场中时, 即势能只与粒子的位置 x 有关。在经典力学中，若势能与时间无关，总能量就是一个运动常量；在量子力学中则存在着能量完全确定的态。薛定谔方程中粒子坐标 x 和时间 t 独立。分离变量后，我们可以得到不含时 (time-independent) 的薛定谔方程，即定态 (stationary states) 薛定谔方程 (物理上喜欢叫它能量本征方程，因此定态也叫能量本征态), 如式 (2) 所示。这种形式的薛定谔方程是化学中最常见的也是最感兴趣的形式。

$$\hbar\frac{\partial}{\partial t}|\Psi(x,t)\rangle = \hat{H}|\Psi(x,t)\rangle \qquad (1)$$

$$\hat{H}|\psi(x)\rangle = E|\psi(x)\rangle \qquad (2)$$

所以我们的核心目标可以描述为：求解在特定分子体系下不含时的、非相对论情况下的定态薛定谔方程，得到的解为体系处于不同定态下的本征值，即能量。根据变分原理 (variational principle) 可知，求得的最小特征值为体系的基态能量 (ground-state energy), 对应的系统状态为基态 (ground-state)。

假设原子核和电子是质点并忽略自旋-轨道相互作用 (spin-orbit interaction)，则对于一个多电子的分子体系而言，Hamiltonian 可以写成式 (3) 的形式 (原子单位)，式中 A, B 指的是核，i, j 指的是电子。其中第一项是电子的动能，第二项是核的动能，第三项表示电子与核的库仑吸引力，r_{Ai} 是电子 i 与原子序数为 Z_A 的核 A 之间的距离，第四项表示核与核的库仑排斥作用，R_{AB} 是核 A 与核 B 之间的距离，最后一项表示电子与电子排斥作用，r_{ij} 是电子 i 与电子 j 之间的距离。

$$\hat{H} = -\frac{1}{2}\sum_i \nabla_i^2 - \sum_A \frac{1}{2M_A}\nabla_A^2 - \sum_{A,i}\frac{Z_A}{r_{Ai}} + \sum_{A>B}\frac{Z_A Z_B}{R_{AB}} + \sum_{i>j}\frac{1}{r_{ij}} \qquad (3)$$

在原子中往往核的质量比电子质量大得多，运动比电子慢。因此，可以忽略原子核的效应，假定电子运动在固定不动的原子核周围，这就是 Born-Oppenheimer 近似。在这个近似下，式 (3) 中核的动能项可忽略不

[1] Ira N. Levine. *Quantum chemistry* . Pearson Prentice Hall, Upper Saddle River, NJ, 5th edition, 2000.

计，核间的排斥项可视为常数项，原子核和核外电子的运动就可以分开。系统的电子薛定谔方程可以写为式 (4)，其中电子的 Hamiltonian 为式 (5) 的形式。

$$\hat{H}_{el}|\psi_{el}\rangle = E_{el}|\psi_{el}\rangle \qquad (4)$$

$$\hat{H}_{el} = -\frac{1}{2}\sum_i \nabla_i^2 - \sum_{A,i}\frac{Z_A}{r_{Ai}} + \sum_{i>j}\frac{1}{r_{ij}} \qquad (5)$$

12.2 2.2 二次量子化

前面的描述都是基于一次量子化，表达式比较复杂繁琐，下面我们引入简洁的二次量子化 (second quantization) 表示[2] 。

首先，我们定义 $a_i^\dagger$ 为产生算符 (creation operator)，它的作用是在第 i 个自旋轨道上产生一个电子，同理，定义 a_j 为湮灭算符 (annihilation operator)，它的作用是在第 j 个自旋轨道上湮灭一个电子。电子是费米子 (fermion)，其产生算符和湮灭算符遵循泡利不相容原理 (Pauli exclusion principle) 且服从反对易 (anticommutation) 关系, 如式 (6) 所示。

$$\begin{aligned}\{a_i, a_j\} &= \{a_i^\dagger, a_j^\dagger\} = 0 \\ \{a_i, a_j^\dagger\} &= a_i a_j^\dagger + a_j^\dagger a_i = \delta_{ij}\end{aligned} \qquad (6)$$

$$\Phi_{HF}(\chi_1, \chi_2, \cdots, \chi_N) = |n_{M-1}, n_{M-2}, \cdots, n_0\rangle \qquad (7)$$

$$|n_{M-1}, n_{M-2}, \cdots, n_0\rangle = a_0^\dagger a_1^\dagger \cdots a_N^\dagger|\rangle = \prod_{i=1}^{N} a_i^\dagger|\rangle \qquad (8)$$

在二次量子化的框架下，slater determinant 可以表示成占据数态 (occupation number state), 这里采用轨道序数从右向左依次递增的约定, 如式 (7) 所示。其中 N 为电子数，M 为自旋轨道数。当自旋轨道 χ_p 被电子占据时，n_p 为 1 时 (此时该自旋轨道 χ_p 在 Φ 的表达式中)；反之，未被占据时，n_p 为 0(此时该自旋轨道不在 Φ 的表达式中)。换句话说，占据数态是一组只包含 0 和 1 的二元的数串，其长度是自旋轨道的数量，每一位上的 1 表示该编号下的自旋轨道是被占据的。由此我们可以将一系列产生算符作用在真空态 $|\rangle$ 来构造任何系统的 Hartree-Fock 态。如式 (8) 所示。下面我们来举个例子理解一下这两个算符。

例 1 考虑有四个自旋轨道的氢分子体系。为了方便起见这里对自旋轨道进行编号，按照自旋轨道能量由低到高且将自旋向上 (*alpha spin*) 的电子序数排在自旋向下 (*beta spin*) 前面，其占据数态可表示为 $|n_3 n_2 n_1 n_0\rangle$ 。我们知道每个氢原子都有一个电子，所以一个氢分子体系有两个电子。根据分子轨道理论，这两个电子填充在能量较低的成键轨道上，如图 1 所示。该体系的 *Hartree-Fock* 态可构造为式 (9) 的形式。若此时，我们将算符 a_1 作用在该态上，应用式 (6) 和真空态的性质，我们可以得到如式 (10) 的结果。

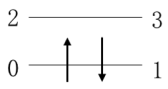

图 1: 四个自旋轨道的氢分子体系

[2] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics* , 92(1):015003, 2020.

$$\Phi_{HF}(\chi_1,\chi_2) = a_0^\dagger a_1^\dagger|\rangle = |0011\rangle \qquad (9)$$

$$a_1|0011\rangle = a_1 a_0^\dagger a_1^\dagger|\rangle = -a_0^\dagger a_1 a_1^\dagger|\rangle = -a_0^\dagger(1-a_1^\dagger a_1)|\rangle = -a_0^\dagger|\rangle = -|0001\rangle \qquad (10)$$

实际上，产生与湮灭算符作用在占据数态上可以表达成式 (11) 的形式，其中 $(-1)^{\sum_{i=0}^{p-1} n_i}$ 是相位因子，也称宇称 (parity)。$\oplus$ 表示模二加法 ($0\oplus 1 = 1, 1\oplus 1 = 0$)。

$$\begin{aligned}
&a_p\,|n_{M-1}, n_{M-2}, \ldots, n_0\rangle \\
&= \delta_{n_p,1}(-1)^{\sum_{i=0}^{p-1} n_i}\,|n_{M-1}, n_{M-2}, \ldots, n_p \oplus 1, \ldots, n_0\rangle \\
&a_p^\dagger\,|n_{M-1}, n_{M-2}, \ldots, n_0\rangle \\
&= \delta_{n_p,0}(-1)^{\sum_{i=0}^{p-1} n_i}\,|n_{M-1}, n_{M-2}, \ldots, n_p \oplus 1, \ldots, n_0\rangle
\end{aligned} \qquad (11)$$

例 2 为了更好理解以上的公式，我们试着考虑图 2 的单电子激发态。可以看到对比图 1 处于基态的情况，此时的激发态一个电子由轨道 1 激发到轨道 3 上，即从轨道 1 上湮灭，在轨道 3 上产生。用算符表示为 $a_3^\dagger a_1|0011\rangle$ 。首先来看 a_1 作用在态 $|0011\rangle$ 上时，会将这个态变为负的 $|0001\rangle$ 。这是因为初始状态为 $|0011\rangle$ ，按 $|n_3 n_2 n_1 n_0\rangle$ 的编码顺序 n_1 是 1，那么 a_1 去作用在 1 这个态上，会将其变为 0，然后计算相位因子，发现相位因子等于 1($n_0 = 1$)，这样就产生一个负号。同理当 $a_3^\dagger$ 作用在态 $|0001\rangle$ 上时，其相位因子等于 1($n_0+n_1+n_2 = 1+0+0 = 1$)，如下所示。

$$\begin{aligned}
a_1|0011\rangle &= \delta_{1,1}(-1)^1|0001\rangle = -|0001\rangle \\
-a_3^\dagger|0001\rangle &= -\delta_{0,0}(-1)^{1+0+0}|1001\rangle = |1001\rangle
\end{aligned} \qquad (12)$$

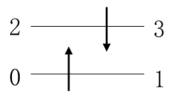

图 2: 四个自旋轨道的氢分子体系的单电子激发态

二次量子化后，电子的 Hamiltonian 表示成式 (13) 的形式[3] ，该式中第一项为是单粒子算符，第二项为双粒子算符，下标 $pqrs$ 分别代表不同电子自旋轨道，其中 h_{pq} 、h_{pqrs} 分别代表单、双电子积分，计算公式如式 (14) 所示。如果选定基组，我们就可以确定积分的具体值。

$$\hat{H}_{el} = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2}\sum_{pqrs} h_{pqrs} a_p^\dagger a_q^\dagger a_s a_r \qquad (13)$$

$$\begin{aligned}
h_{pq} &= \int dr x_p(r)^* \left(-\frac{1}{2}\nabla^2 - \sum_A \frac{Z_A}{|r_A - r|}\right) x_q(r) \\
h_{pqrs} &= \int dr_1 dr_2 \frac{1}{|r_1 - r_2|} x_p(r_1)^* x_q(r_2)^* x_r(r_1) x_s(r_2)
\end{aligned} \qquad (14)$$

从 h_{pq} 的计算公式中，可以发现大括号中的两项正是一次量子化 Hamiltonian 中的电子动能项 $-\frac{1}{2}\nabla^2$ 和核与电子之间的引力势能项 $-\sum_A \frac{Z_A}{|r_A - r|}$ ；从 h_{pqrs} 的计算公式中，可以发现 $\frac{1}{|r_1 - r_2|}$ 正是一次量子化 Hamiltonian 中的电子间排斥能项。由此可见，h_{pq} 和 h_{pqrs} 起到了联系二次量子化 Hamiltonian 与一次量化 Hamiltonian 的作用。

[3] Attila Szabo and Neil S Ostlund. *Modern quantum chemistry: introduction to advanced electronic structure theory* .Courier Corporation, 2012.

12.3 2.3 映射

由二次量子化我们得到了 Hamiltonian 的费米子形式，是不是就意味着可以直接输入进量子计算机中进行计算了呢？其实不然，因为量子计算机是以量子比特的语言运行的，这里的量子比特是一组可区分的粒子。而电子是费米子，它是以费米子算符的形式表示的全同粒子。因此，为了在量子计算机上模拟电子结构问题，我们需要一个映射关系，将电子的费米子算符映射 (mapping) 到量子计算机的泡利算符 (pauli operator)。目前，比较常见的映射有 Jordan-Wigner(JW) 变换[4] 、Bravyi-Kitaev(BK)[5] 变换和 Parity 变换[6] 等。不同的变换所得到的量子线路深度可能有所不同，但他们的功能都是一致的，都是为了将费米子系统映射到量子计算机中去。随着研究的不断推进，在这些映射基础上，人们也发展了各式各样的新型映射，但应用较广的还主要是这些。接下来，我们以 JW 变换为例进行介绍。

在 JW 变换中，每个分子自旋轨道的占据信息记录在量子比特的计算基 (computational basis states) $|0\rangle$ 、$|1\rangle$ 中，即 $|0\rangle$ 表示不占据，$|1\rangle$ 表示占据。因此，对于 M 个自旋轨道的系统，采用右边开始编码约定，存在如式 (15) 的映射关系。我们回顾一下费米子的产生算符和湮灭算符的作用，产生算符是将粒子从未占据态从 $|0\rangle$ 变为占据态 $|1\rangle$ ；湮灭算符将粒子从占据态 $|1\rangle$ 态变为未占据态 $|0\rangle$ 。为了在量子计算机中实现这种操作，我们需要进行人为的构建。我们可以发现，通过利用 Pauli X 矩阵和 Pauli Y 矩阵进行组合，可以在量子计算机上实现与费米子类似的操作，如式 (16) 所示。这里的 $Q_j^\dagger, Q_j$ 是量子比特的产生与湮灭算符，X_j, Y_j 表示 Pauli X 和 Y 矩阵作用在第 j 个量子比特上。

$$|n_{M-1}, n_{M-2}, \cdots, n_0\rangle \to |q_{M-1}\rangle \otimes |q_{M-2}\rangle \otimes \cdots \otimes |q_0\rangle \qquad q_j = n_j \in \{0, 1\} \tag{15}$$

$$Q_j = |0\rangle\langle 1| = \frac{X_j + iY_j}{2} \qquad Q_j^\dagger = |1\rangle\langle 0| = \frac{X_j - iY_j}{2} \tag{16}$$

$$\begin{aligned} a_j &= I^{\otimes n-j-1} Q_j \otimes Z_{j-1} \otimes \cdots \otimes Z_0 \\ a_j^\dagger &= I^{\otimes n-j-1} Q_j^\dagger \otimes Z_{j-1} \otimes \cdots \otimes Z_0 \end{aligned} \tag{17}$$

有了 $Q_j^\dagger, Q_j$ 对于模拟费米子来说还是不够的。费米子算符实现的除了产生、湮灭算符带来的占据态 (occupation) 信息的转变外，其相位因子 (phase factor) 还记录了体系的宇称信息。因此，要想在量子计算机上模拟费米子，还必须考虑记录体系的宇称信息的一个相位因子。对于 JW 变换，这个相位因子可以用一串 Pauli-Z 矩阵来等效替代。因此，在 JW 变换中，费米子的产生、湮灭算符可以表示为式 (17)，其中 n 为自旋轨道数，也是量子比特数；j 表示算符作用的子空间，即量子比特的序号。$Q_j^\dagger, Q_j$ 改变自旋轨道的占据态，一串 Pauli-Z 矩阵来实现相位因子。JW 变换的主要思想是将费米子轨道的占据信息局域存储在量子比特中，但宇称信息非局域存储，因为式中每个 Pauli -Z 矩阵依次作用在不同的量子比特上。这也说明了对于 JW 变换，Pauli 的权重随着自旋轨道数 M 的增加呈线性增长。

例 3 这里我们来具体举例说明在 JW 变换中费米子算符是如何转换成泡利算符的。以例 2 中提及的算符 $a_3^\dagger$ 和 a_1 为例。(以下泡利算符之间都是直积形式，张量积符号已省略)

$$\begin{aligned} a_3^\dagger &= Q_3^\dagger Z_2 Z_1 Z_0 = \frac{1}{2}(X_3 - iY_3) Z_2 Z_1 Z_0 = \frac{1}{2} X_3 Z_2 Z_1 Z_0 - \frac{i}{2} Y_3 Z_2 Z_1 Z_0 \\ a_1 &= I_3 I_2 Q_1 Z_0 = \frac{1}{2} I_3 I_2 (X_1 + iY_1) Z_0 = \frac{1}{2} I_3 I_2 X_1 Z_0 + \frac{i}{2} I_3 I_2 Y_1 Z_0 \end{aligned} \tag{18}$$

[4] E Wigner and Pascual Jordan. Über das paulische äquivalenzverbot. *Z. Phys*, 47:631, 1928

[5] Sergey B Bravyi and Alexei Yu Kitaev. Fermionic quantum computation. *Annals of Physics* ,298(1):210–226, 2002

[6] Jacob T Seeley, Martin J Richard, and Peter J Love. The bravyi-kitaev transformation for quantum computation of electronic structure. *The Journal of chemical physics* ,137(22):224109, 2012.

这里，我们展示三个自旋轨道的 JW 变换示意图，如图 3 所示。可以看出，在 JW 变换下，每一个量子比特标识一个费米轨道，占据态和非占据态分别映射到量子比特的 $|1\rangle$ 态和 $|0\rangle$ 态。此时，轨道和量子比特是一一对应的。

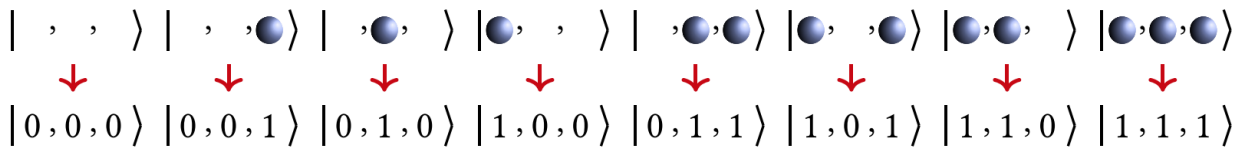

图 3: 三个自旋轨道的 JW 变换示意图. 图引自[7]

为了让大家更好地理解 JW 变换，我们从数学上具体推导下对上面 3 比特的变换例子。我们假设基态的轨道占据情况是 $|010\rangle$，那么 $a_2^\dagger$ 作用在这个态上时，根据公式 (11)，会将这个态变为负的 $|110\rangle$。对于 $a_2^\dagger$ 这个算符来说，经过 JW 变换，变为了一串 $Q_2^\dagger Z_1 Z_0$ 的直积形式，作用在初态 $|010\rangle$ 上，按照序号逐一作用上去发现，Z_0 作用在 0 态上不产生效果，Z_1 作用在态上产生一个负号，$Q_2^\dagger$ 作用在 0 态上，会将其变为 1。因此我们发现，在 JW 变换下，系统前后是等价的，这也验证了 JW 变换的有效性——将费米子系统在量子计算机上等效地表示出来了。

$$
\begin{aligned}
a_2^\dagger|010\rangle &= \delta_{0,0}(-1)^{0+1}|110\rangle = -|110\rangle \\
Q_2^\dagger \otimes Z_1 \otimes Z_0|010\rangle &= -|110\rangle \qquad (19)
\end{aligned}
$$

12.4 2.4 拟设

为了获得与体系量子终态相近的试验波函数，我们需要一个合适的波函数假设，我们称之为拟设 (Ansatze)。并且理论上，假设的试验态与理想波函数越接近，越有利于后面得到正确基态能量。实际上，在量子计算机上模拟分子体系基态问题，最终都是转换到在量子计算机上对态进行演化，制备出最接近真实基态的试验态波函数。经典的传统计算化学领域已经发展了多种多样的波函数构造方法，比如组态相互作用法 (configuration interaction, CI), 耦合簇方法 (coupled-cluster, CC) 等。目前，应用在 VQE 上拟设主要分为两大类，一类化学启发拟设，如酉正耦合簇 (unitary coupled-cluster, UCC)，另一类是基于量子计算机硬件特性构造的拟设，即 Hardware-Efficient 拟设。

Hardware-Efficient 拟设

Hardware-Efficient 直接将 $|00\cdots0\rangle$ 演化成纠缠态 (可以看成是叠加态的特殊情形，其特征是不能分解成两个态的张量积)，不再经过 Hartree-Fock 态。该拟设的量子线路的结构一般包括许多重复、密集的模块，每个模块由特定类型的含参数的量子门构成，这些量子门在目前含噪声的中型量子器件 (NISQ) 上更容易实现，因为其更能满足现有量子计算机的特点——较短的相干时间与受限的量子门结构。这一拟设被应用在小分子 VQE 的实验演示中[8][9]，但是并不适用于更大的体系。因为它并不具体考虑模拟的实际化学体系，制备出了许

[7] Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews* , 120(22):12685–12717, 2020.

[8] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* , 549(7671):242–246, 2017.

[9] Abhinav Kandala, Kristan Temme, Antonio D Córcoles, Antonio Mezzacapo, Jerry M Chow, and Jay M Gambetta. Error mitigation extends the computational reach of a noisy quantum processor. *Nature* , 567(7749):491–495, 2019.

多物理上不应存在的量子态，从而引入了巨大的参数量，使优化变得繁琐甚至无法进行[10]。以氢分子为例，它仅含两个电子，若使用最小基组，它的 Hartree-Fock 态可以写成 $|0011\rangle$ ，如例 1 所示。因此在不考虑自旋禁阻的情况下，只可能存在 $C_4^2 - 1 = 5$ 种激发态 (包括单激发态和双激发态)。但是，在经过 Hardware-Efficient 拟设后，会产生 $|0111\rangle$ 甚至 $|1111\rangle$ 等电子数或总自旋量子数不守恒的激发态，这就增加了经典优化器需要优化的参数、提高了陷入“高原贫瘠”状态的可能性。

酉耦合簇拟设

在求解体系基态能量时，若选用 Hartree-Fock 态作为初猜波函数，由于 Hartree-Fock 态为单电子组态，没有考虑电子关联能，所以要将其制备成多电子组态 (也就是纠缠态)，以使测量结果达到化学精度。UCC 中的 CC 即是量子化学中的耦合簇算符 $e^{\hat{T}}$，它从 Hartree-Fock 分子轨道出发，通过指数形式的耦合算符得到真实体系的波函数, 如式 (20) 所示。这里的 $|\psi_{HF}\rangle$ 即为 HF 波函数，是参考态。$\hat{T}$ 即耦合簇理论中的电子簇算符，由子簇算符加和而成，其中 $\hat{T}_1$ 包含所有单激发的算符，$\hat{T}_2$ 包含所有双激发的算符，其余项以此类推。由于在一个多电子体系中，三激发、四激发发生的概率很小，所以通常在双激发处进行“截断”，最终只剩 $\hat{T}_1$ 和 $\hat{T}_2$ 两项，由产生算符与湮灭算符表示如式 (21) 所示。在例 2 中我们展示了氢分子单电子激发的一种情况，实际上在不考虑自旋禁阻与自旋对称的情况下，该体系的单双激发簇算符分别为式 (22) 所示。

$$
\begin{aligned}
&|\psi_{CC}\rangle = e^{\hat{T}}|\psi_{HF}\rangle \\
&\hat{T} = \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \cdots
\end{aligned} \tag{20}
$$

$$
\begin{aligned}
\hat{T}_1 &= \sum_{r}^{vir}\sum_{a}^{occ} t_a^r a_r^\dagger a_a \\
\hat{T}_2 &= \sum_{r,s}^{vir}\sum_{a,b}^{occ} t_{ab}^{rs} a_r^\dagger a_s^\dagger a_b a_a
\end{aligned} \tag{21}
$$

$$
\begin{aligned}
\hat{T}_1 &= t_0^2 a_2^\dagger a_0 + t_0^3 a_3^\dagger a_0 + t_1^2 a_2^\dagger a_1 + t_1^3 a_3^\dagger a_1 \\
\hat{T}_2 &= t_{01}^{23} a_3^\dagger a_2^\dagger a_0 a_1
\end{aligned} \tag{22}
$$

但是 e^T 并不是酉算子，无法直接通过 JW 变换、BK 变换等方法映射到量子比特上，所以需要构造出酉算子版本的指数耦合簇算符，即酉耦合簇算符，如式 (23) 所示。

$$
\hat{U} = e^{\hat{T} - \hat{T}^\dagger} \tag{23}
$$

$$
\hat{U}(\theta) = e^{\hat{T}_1(\theta) + \hat{T}_2(\theta) - \hat{T}_1^\dagger(\theta) - \hat{T}_2^\dagger(\theta)} \tag{24}
$$

若 UCC 中的簇算符 $\hat{T}$ 只含 $\hat{T}_1$ 这一项, 则称这一算符为单激发耦合簇 (UCCS) 算符; 若 UCC 中的簇算符 $\hat{T}$ 含有 $\hat{T}_1$ 和 $\hat{T}_2$ 两项, 则称这个算符为单双激发耦合簇 (UCCSD) 算符, 如式 (24) 所示。其中 $\hat{T}_1(\theta) = \sum_{ij}\theta_{ij}a_i^\dagger a_j$, $\hat{T}_2(\theta) = \sum_{ijkl}\theta_{ijkl}a_i^\dagger a_j^\dagger a_k a_l$ 。这里的 θ_{ij}⍰θ_{ijkl} 就是需要通过优化器来优化的参数且均为实数，对应经典簇算符系数 t_a^r 、t_{ab}^{rs} 与 UCCS 相比，UCCSD 比 UCCS 多考虑了双电子激发态，因此演化线路就变得更为复杂，计算也更加耗时，但随之而来的计算精度也有所提升。

在含噪声的中型量子器件 (NISQ) 上，利用变分量子算法 (如 VQE 算法) 进行化学模拟，其模拟效果很大程度上取决于用于制备试验态的含参拟设线路的高效性。而拟设线路是否高效，一般可以通过线路含参个数、线路深度、双量子逻辑门的个数来判断。若线路含参个数过多，经典优化器在对线路参数进行优化时，容易

[10] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications* , 9(1):1–6, 2018.

陷入“高原贫瘠”状态；若线路过深 (特别是双量子逻辑门过多)，演化时间就越长，所制备出的试验态的保真度就越低。所以设计拟设线路不仅要考虑到其结果的精度，其在线路上的效率也是在实际应用中要着重考虑的方面。针对拟设线路进行适当的截断或者优化也是目前许多学者的研究方向，比如 ADAPT-VQE[11] ，根据各个算符的梯度自适应地选择构建拟设，大大减少优化参数和约化线路深度，做到同时节约计算资源并提高计算效率。VQE 的其它改进方法还有很多，这里不再赘述，有兴趣的可查阅相关文献[Page 50, 21213]。

12.5 2.5 Trotter 分解

上节我们讲了酉耦合簇拟设，但在加载进量子线路上进行拟设线路的构造之前，还需要的一个关键技术就是 Trotter 分解[14] (Lie-Trotter-Suzuki decomposition)，又称渐近近似定理。

在式 (24) 中，指数项是由一系列簇算符构成，使用 Trotter 分解，即考虑一阶近似下，$e^{\hat{A}+\hat{B}} \approx e^{\hat{A}}e^{\hat{B}}$ ，则式 (24) 可以写为:

$$\hat{U}(\theta) = exp\left(\sum_{ij}\theta_{ij}(a_i^\dagger a_j - a_j^\dagger a_i)\right) \times exp\left(\sum_{ijkl}\theta_{ijkl}(a_i^\dagger a_j^\dagger a_k a_l - a_l^\dagger a_k^\dagger a_j a_i)\right) \tag{25}$$

再使用一次一阶近似下的 Trotter 分解，上式可以写为：

$$\hat{U}(\theta) = \prod_{ij} exp(\theta_{ij}(a_i^\dagger a_j - a_j^\dagger a_i)) \times \prod_{ijkl} exp(\theta_{ijkl}(a_i^\dagger a_j^\dagger a_k a_l - a_l^\dagger a_k^\dagger a_j a_i)) \tag{26}$$

参考文献

[11] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. Adaptvqe: An exact variational algorithm for fermionic simulations on a quantum computer. *arXiv preprint arXiv:1812.11173* , 2018.

[12] Dmitry A Fedorov, Bo Peng, Niranjan Govind, and Yuri Alexeev. Vqe method: A short survey and recent developments. *Materials Theory* , 6(1):1–21, 2022.

[13] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews* , 119(19):10856–10915, 2019.

[14] Hale F Trotter. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society*, 10(4):545–551, 1959.

Chapter 13

VQE 流程简介

13.1 3.1 变分原理

对于一个 n 阶的方阵，如果想找到它的特征值 $\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_n$ ，可以利用 VQE 算法。在量子化学中，该算法被用于寻找描述某一体系 (如多电子体系) 哈密顿量的特征值 $E_0, E_1, E_2, \ldots, E_n$ ，根据变分原理 (Variational Principle)，其可求得的最小特征值即为体系的基态能量 E_0 。这里的变分原理起源于数学领域的泛函分析，最初见于著名的例子最速曲线问题[1]，随后被物理和化学学者应用到各自领域中，意在解决无法直接解析的极值函数问题。变分原理的基础是 Euler-Lagrangian 方程，它对应着泛函的临界点。在量子化学中常用于解电子薛定谔方程以得到目标能量。变分法求解基态能量可表述如下[2]：

设量子体系的哈密顿量为 $\hat{H}$ ，$|\psi\rangle$ 态为空间中任意未知的态矢量 (称为试验态矢量), 并且 $|\psi\rangle$ 满足归一化条件 $\langle\psi|\psi\rangle = 1$。我们将 $\hat{H}$ 在态 $|\psi\rangle$ 中的平均值 $\langle\hat{H}\rangle$ 看作态矢量 $|\psi\rangle$ 的泛函。

$$\langle\hat{H}\rangle = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} \qquad (1)$$

考虑一个任意的物理体系，其哈密顿量 $\hat{H}$ 与时间无关。设体系包括 $\hat{H}$ 在内的一组力学量完全集的共同本征态为 $\{|\psi_i\rangle\}$ ，相对应的能量本征值为 $E_0 \leq E_1 \leq E_2 \leq \ldots$, 展开试验态矢量 $|\psi\rangle = \Sigma_i a_i|\psi_i\rangle$ 。于是

$$\langle\psi|\hat{H}|\psi\rangle = \sum_{i=0}^{\infty} |a_i|^2 E_i \qquad (2)$$

且有 $\langle\psi|\psi\rangle = \sum_{i=0}^{\infty} |a_i|^2 = 1$ 。对于 $i \geq 0$ ，有 $E_i \geq E_0$ 且满足 $|a_i|^2 E_i \geq |a_i|^2 E_0$ ，因此

$$\langle\hat{H}\rangle = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} = \frac{\sum_i |a_i|^2 E_i}{\sum_i |a_i|^2} \geq \frac{\sum_i |a_i|^2 E_0}{\sum_i |a_i|^2} = E_0 \qquad (3)$$

上式说明，用变分原理求出的能量极值 $\langle\hat{H}\rangle$ 总是大于或等于体系的准基态能量 (exact ground state energy)，它给出了体系基态能量的一个上界 (upper bound)。

[1] Herman Erlichson. Johann bernoulli’s brachistochrone solution using fermat’s principle of least time. *European journal of physics* , 20(5):299, 1999.

[2] Ira N. Levine. *Quantum chemistry* . Pearson Prentice Hall, Upper Saddle River, NJ, 5th edition, 2000.

从式 (3) 中可以看出，如果所选择的试验态 $|\psi\rangle$ 正好就是体系的基态 $|\psi_0\rangle$，那么不等式中的等号成立，我们直接得到了体系的基态能量 E_0；但往往更多的情况是，选择的试验态 $|\psi\rangle$ 与体系的基态相比有一定差距，导致计算得到的 E 大于 E_0 很多，这时就需要引入一组参数 $\vec{\theta}$，通过不断迭代参数来调节试验态波函数 $|\psi(\vec{\theta})\rangle$，使其最终非常接近体系的基态。通过上述过程求解系统基态能量的方法称为变分法，也是 VQE 之所以是 Variational 的原因。

$$\langle\hat{H}\rangle = E(\vec{\theta}) = \frac{\langle\psi(\vec{\theta})|\hat{H}|\psi(\vec{\theta})\rangle}{\langle\psi(\vec{\theta})|\psi(\vec{\theta})\rangle} \geq E_0 \qquad (4)$$

13.2 3.2 VQE 流程概述

变分量子特征值求解算法 (Variational Quantum Eigensolver, VQE) 是一种经典-量子混合算法，它使用参数化的量子线路来构造波函数，利用经典计算机来优化这些参数，使哈密顿量的期望值最小化，得到的最小能量即所求的基态能量。它的基本流程如图 1 所示，具体流程包括量子态制备、哈密顿量子项 H_i 的测量、求和、收敛性判断和参数优化等过程，其中，量子态制备 (quantum state preparation)、哈密顿量子项的测量 (也称量子期望估计 quantum expectation estimation) 是在量子计算机上进行的，即图中淡黄色部分，其它的步骤如求和、参数优化由经典计算机完成，即图中淡蓝色的部分。

具体来讲，VQE 算法流程可以总结为以下步骤：

(i) 选一组随机初始参数 $\theta_1^k, \theta_2^k, \theta_3^k \ldots \theta_n^k$

(ii) 在虚拟机或量子计算机上制备试验波函数 $|\psi(\vec{\theta})\rangle$

(iii) 对哈密顿量的各个子项进行测量，然后在经典计算机上进行求和，得 $|\psi(\vec{\theta})\rangle$ 的哈密顿量的期望值，即分子的能量

(iv) 判断该能量是否满足收敛条件，如果满足，则将该能量作为分子基态能量的近似值，终止计算；如果不满足，则变分优化参数，利用经典优化器产生一组新的参数 $\theta_1^{k+1}, \theta_2^{k+1}, \theta_3^{k+1} \ldots \theta_n^{k+1}$，重新制备量子态

(v) 重复 2-4 步，直到能量收敛。

此时，理论上参数化量子线路已制备好哈密顿量的基态，或非常接近基态的状态。与量子相位估计算法相比，VQE 需要更少的门数和更短的相干时间。它以多项式的重复次数换取所需的相干时间的减少。因此，它更适合于 NISQ 时代。

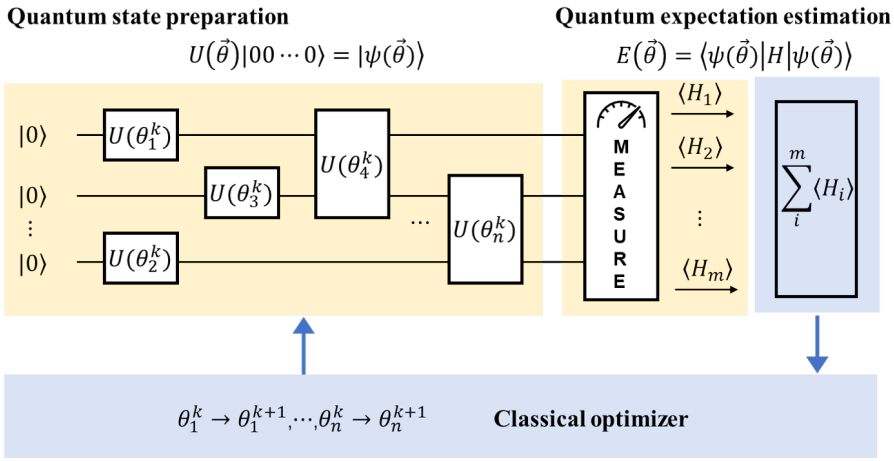

图 1: 经典-量子混合的 VQE 算法流程 (图改自[3])

图 2 展示的是 VQE 流程中量子线路的部分，即图 1 淡黄色的部分。量子态的制备部分我们以 UCC 拟设为例来进行线路构造。使用 UCC 拟设来制备试验态通常是先在线路上搭建好参照态 $|\psi_{ref}\rangle$，然后添加上节中式 (23) 的 $\hat{U}(\theta)$ 酉耦合簇算符演化生成试验波函数 $|\psi(\theta)\rangle$。

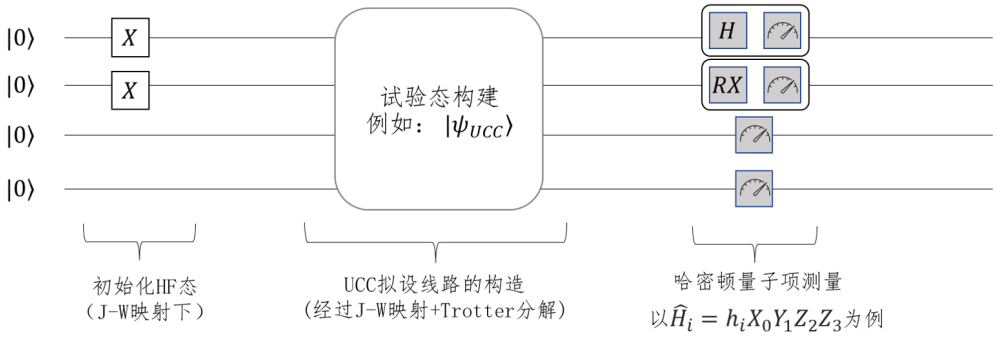

图 2: VQE 量子线路部分的流程

13.3 3.3 量子线路初态的构造

我们来看一下线路初态的具体构造。实际的演化线路中，开始每个比特的状态都默认为 $|0\rangle$ 态，若要构造 $|1\rangle$ 态的比特，我们首先需要对它翻转。回顾基础量子门的作用，Pauli-X 门正好可以实现这个操作，所以只需对 $|0\rangle$ 态的对应比特施加 X 门即可，如式 (5) 所示。因此，JW 映射下四个自旋轨道的氢分子体系的 Hartree-Fock 初态如图 3 所示，对第 0、1 比特施加 X 门就将 $|0000\rangle$ 的量子状态变成了所需的 $|0011\rangle$ 态。

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{5}$$

[3] Qingchun Wang, Huan-Yu Liu, Qing-Song Li, Ye Li, Yahui Chai, Qiankun Gong, Haotian Wang, Yu-Chun Wu, Yong-Jian Han, Guang-Can Guo, et al. Chemiq: A chemistry simulator for quantum computer. *arXiv preprint arXiv:2106.10162* , 2021.

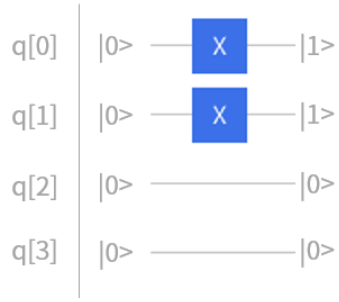

图 3: JW 映射下四个自旋轨道的氢分子初态构建

事实上，对于任意一个含有 M 个自旋分子轨道的 N 电子体系，它的 Hartree-Fock 态都可以这样简单的表示。只要在量子线路中给定 M 个量子比特，然后在前 N 个量子线路上加上 Pauli-X 门即可得到所需要的 N 电子体系的 Hartree-Fock 态。

$$|\psi_{HF}\rangle = |\underbrace{0\ldots0}_{\text{M 个量子比特}}\overbrace{11\ldots11}^{\text{N 个电子}}\rangle$$

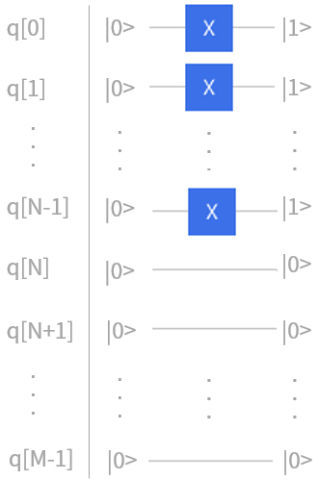

图 4: JW 映射下 M 个自旋轨道的 N 电子体系初态构建

13.4 3.4 拟设线路的构造

在理论基础章节讲完映射和拟设之后，原则上，只要我们给定费米子形式的簇算符，就可以将其转化成 Pauli 算符串，然后加载进量子线路上进行拟设线路的构造。接下来，我们看看如何根据这些 Pauli 算符串来得到最终的量子线路，然后进行演化。首先，考虑两个 Pauli Z 门的指数情况，在线路的情况如图 5 所示，这里的第一个 CNOT 门是用来纠缠两个量子比特，然后应用 Rz 门，再然后是第二个 CNOT 门。通过使用附加的 CNOT 门，这种线路结构可以推广到更多的量子比特。比如对于三个 Pauli Z 门的张量积情况，其线路如图 6 所示。所以可以看出，对于更多量子比特的情况，其线路结构也是很容易拓展得到的。

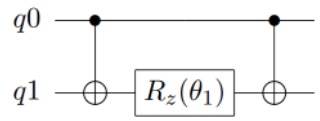

图 5: 两个 Pauli Z 门的量子线路

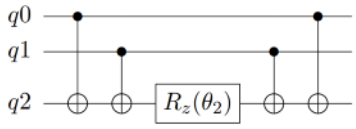

图 6: 三个 Pauli Z 门的量子线路

刚才，我们考虑的都是只有 Pauli-Z 门的。实际线路中，还存在许多含有其它 Pauli 算符的情况，这时我们就需要对 X、Y 基进行转换，转到 Z 基上再通过上述的线路实现。比如，对 Pauli-X 矩阵而言，可以通过在前后加一个 H 门来完成基的变换；而对于 Pauli-Y 门，则通过在前后加两个旋转 X 门来完成，如式 (6) 所示。因此，对于指数上含有 Pauli-X 门和 Pauli-Z 门的直积的情况，如 $e^{-i\theta_3(\sigma_x^0\otimes\sigma_z^1)}$，只需在 Pauli-X 门所作用的比特前后分别加 H 门就可以实现模拟，具体的量子线路如图 7 所示。类似的，当指数上有 Pauli-Y 门的时候，如 $e^{-i\theta_4(\sigma_z^0\otimes\sigma_y^1)}$，需要在其作用的量子比特前后各加一个旋转 X 门。具体的量子线路如图 8 所示。

$$\begin{aligned}\sigma_x &= H\sigma_z H \\ \sigma_y &= \mathrm{RX}(-\frac{\pi}{2})\sigma_z \mathrm{RX}(\frac{\pi}{2}) \end{aligned} \tag{6}$$

其中，

$$\mathrm{RZ}(\theta) = e^{-i\theta Z/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \mathrm{RX}(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$$

如果对于一些更加复杂的 Pauli 算符串，同时含有多个 Pauli-X、Pauli-Y 门时，利用这些规律进行扩展即可相应模拟线路。例如对于这个有 4 个 Pauli 算符串 $e^{-i\theta_5(\sigma_x^0\otimes\sigma_z^1\otimes\sigma_y^2\otimes\sigma_x^3)}$，其中包含了 2 个 Pauli-X 门、一个 Pauli-Y 门、一个 Pauli-Z 门，转换后得到它的量子线路如图 9 所示。

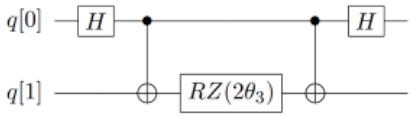

图 7: 指数上含 Pauli-X 门的量子线路

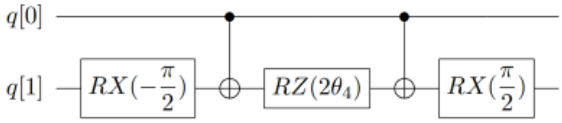

图 8: 指数上含 Pauli-Y 门的量子线路

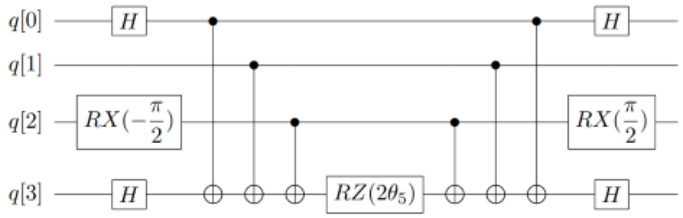

图 9: 指数上更复杂的 Pauli 算符串的量子线路

通过对上述过程进行推广，我们由此可以模拟复杂的酉正簇算符，让其在线路上对初态进行演化，从而制备出最接近真实基态的试验态波函数。下面我们就来看一下如何在量子线路上构造上一节例 2 的单激发簇算符。在上一节的例 3 中我们已经通过 JW 变换将簇算符从费米子形式转换成泡利直积的形式，即：

$$a_3^\dagger = \frac{1}{2}X_3 \otimes Z_2 \otimes Z_1 \otimes Z_0 - \frac{i}{2}Y_3 \otimes Z_2 \otimes Z_1 \otimes Z_0$$
$$a_1 = \frac{1}{2}I_3 \otimes I_2 \otimes X_1 \otimes Z_0 + \frac{i}{2}I_3 \otimes I_2 \otimes Y_1 \otimes Z_0$$

根据泡利算符的性质，则该单激发簇算符 $a_3^\dagger a_1$ 为四项：

$$\frac{i}{4}Y_1 \otimes Z_2 \otimes X_3 + \frac{1}{4}X_1 \otimes Z_2 \otimes X_3 + \frac{1}{4}Y_1 \otimes Z_2 \otimes Y_3 - \frac{i}{4}X_1 \otimes Z_2 \otimes Y_3$$

如我们在 2.4 节中所讲的，只有是厄米矩阵才可以放在指数上进行线路演化，我们需要构造出酉算子版本的指数耦合簇算符，如上一节式 (23) 所示。这里 $a_3^\dagger a_1$ 要减去其共轭转置 $a_1^\dagger a_3$ ，即：

$$-\frac{i}{4}Y_1 \otimes Z_2 \otimes X_3 + \frac{1}{4}X_1 \otimes Z_2 \otimes X_3 + \frac{1}{4}Y_1 \otimes Z_2 \otimes Y_3 + \frac{i}{4}X_1 \otimes Z_2 \otimes Y_3$$

故最后留下两项，即

$$\frac{i}{2}(Y_1 \otimes Z_2 \otimes X_3 - X_1 \otimes Z_2 \otimes Y_3)$$

根据上述构造线路的方法，我们在量子线路上构造下式，其量子线路如图 10 所示。

$$e^{\theta_{13}(a_3^\dagger a_1 - a_1^\dagger a_3)} = e^{(i\theta_a/2)\sigma_y^1 \otimes \sigma_z^2 \otimes \sigma_x^3} e^{(-i\theta_b/2)\sigma_x^1 \otimes \sigma_z^2 \otimes \sigma_y^3}$$

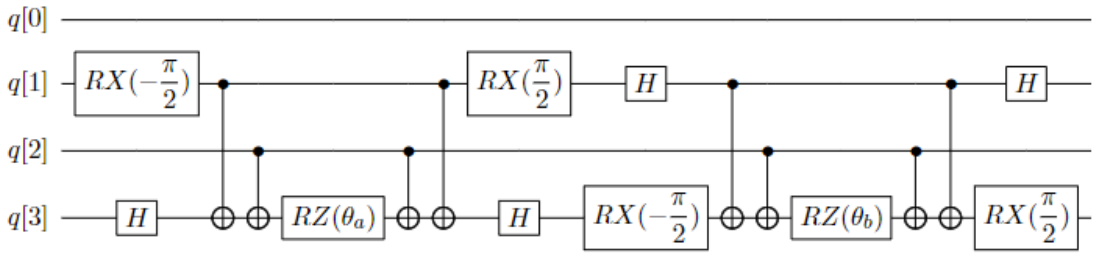

图 10: 单激发算符 $a_3^\dagger a_1 - a_1^\dagger a_3$ 的量子线路

我们接着来看 UCCSD 整体拟设线路的构造。对四个自旋轨道的氢分子，不考虑自旋禁阻与自旋对称，它最终参数化的费米簇算符有五项，如上一节式 (22) 所示，分别列于图 11 线路上五个模块中。图 10 的线路

即是图 11 中第二个模块的线路实现，描述的是位于轨道 1 的电子到轨道 3 上的激发。通过这五个模块的线路演化就获得了试验态。

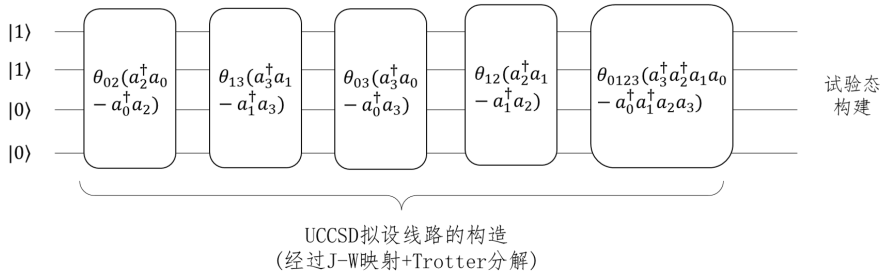

图 11: JW 映射下氢分子 UCCSD 拟设线路的构造

图 10 展示的仅仅是一项单激发簇算符拟设线路的构造，双激发簇算符转换成泡利形式一共有 16 项，而且对于更大的基组，更多的电子数，单、双激发算符会更多，自然线路深度和待优化的参数量就会随之增长，正如我们在 2.4 一节最后提到的那样，针对拟设线路进行适当的截断或者优化对于模拟更大化学的化学体系具有重要的意义。

13.5 3.5 量子期望估计

拟设线路构造完成，制备好了试验态，我们就可以测量哈密顿量的期望，此过程又叫做量子期望估计。VQE 中，哈密顿量作为可观测量用于测量过程。需要注意的是，经过映射后，尽管簇算符和哈密顿算符都会以泡利形式书写，但是哈密顿算符不需要经过指数化操作放在线路上演化，而是直接作用在最后的测量中，所以线路实现和前面所讲有些许不同。比如，对于一个哈密顿量 $\hat{H}_i = h_i X_0 Y_1 Z_2 Z_3$ ，它的测量线路如图 12 所示，其中类似秒表一样的符号是测量门操作。可以看到，第 2，3 比特上只有 Z 门，可以直接测量返还该态在计算基上的期望。但是第 0，1 号比特上存在 X，Y 门，这时需要分别使用 H 门和 $RX(-\pi/2)$ 门旋转换基到到 Z 方向再测得。

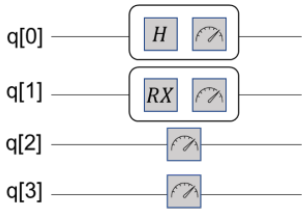

图 12: 哈密顿量用于态矢测量

接着我们来看测量值的读取。与 QPE 中使用二进制编码来存储相位信息不同，在 VQE 中，测量的结果是 0、1 构成的量子态，态的每一位比特存储的是自旋信息，这是由于“测量”代表使用 Pauli-Z 作用在末态上，此时量子位坍缩为 $|0\rangle$ 或 $|1\rangle$ 态。态 $|1\rangle$ 对应 Pauli-Z 中密度矩阵 $|1\rangle\langle 1|$ 的系数是-1，态 $|0\rangle$ 对应密度矩阵 $|0\rangle\langle 0|$ 的系数是 1，所以，单次测量返还的结果如式 (7) 所示。真实结果通常要求反复多次测量，取若干次测量进行加权得到期望。假设进行 m 次测量，得到-1 的次数为 k，那么最终的期望 E_i 可以按式 (8) 计算得到。

$$|j_1 j_2 \dots j_n\rangle = (-1)^{\text{number of 1 in n}} \qquad (7)$$

$$E_i = \langle H_i \rangle = h_i * \frac{-1 * k + (m - k)}{m} \tag{8}$$

例假设一哈密顿量 $\hat{H} = h_0 XY + h_1 ZZ$ ，其中系数 h_0, h_1 为单、双电子积分常数，由经典计算积分而来，为上节中式 (14) 中的 h_{pq} 与 h_{pqrs} ，表示每个子项的能量贡献值。在量子线路上，测量 1000 次的结果如下表所示。可以看到 h_0 在 $|00\rangle$ ，$|01\rangle$ ，$|10\rangle$ ，$|11\rangle$ 四态出现的次数分别为 200，200，100，500，h_1 出现的次数分别为 0，500，500，0。所以，根据式 (8) 计算，最终结果为 $0.4h_0 - h_1$ 。

	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$
$h_0 XY$	200	200	100	500
$h_1 ZZ$	0	500	500	0

图 13: 某哈密顿量测量 1000 次的结果

13.6 3.6 经典优化器参数优化

在 3.1 节中我们构造了含参的拟设线路，进行量子期望估计后，需要不断迭代优化 Ansatze 中涉及的参数以获取最低的能量，并以此能量最低的叠加态作为当前分子模型的基态。VQE 中对这些参数的优化是利用经典优化器来处理的，截止目前，pyChemiQ 提供了以下几种优化器：NELDER-MEAD、POWELL、COBYLA、L-BFGS-B、SLSQP 和 Gradient-Descent。其中无导数优化方法为 Nelder-Mead, Powell, COBYLA; 一阶方法为 L-BFGS-B、SLSQP 和 Gradient-Descent。

NELDER-MEAD 算法

NELDER-Mead 方法是无导数优化方法，它可以用来解决求给定非线性方程最小值的经典无约束优化问题。对于具有 n 个参数的函数，该算法保留一个 $n + 1$ 个点的集合，而这些点就是 n 维空间中多面体的顶点。这个方法通常被称为“单纯形算法”。

POWELL 算法

POWELL 又称方向加速法，它由 POWELL 于 1964 年提出，是利用共轭方向可以加快收敛速度的性质形成的一种搜索方法。该方法不需要对目标函数进行求导，当目标函数的导数不连续的时候也能应用，因此 POWELL 算法是一种十分有效的直接搜索法。POWELL 算法可用于求解一般无约束优化问题，对于维数 $n < 20$ 的目标函数求优化问题，此法可获得较满意的结果。不同于其他的直接法，POWELL 法有一套完整的理论体系，故其计算效率高于其他直接法。该方法使用一维搜索，而不是跳跃的探测步。同时，POWELL 法的搜索方向不一定为下降方向。

COBYLA 算法

线性近似的约束优化算法，简称 COBYLA 算法。该算法是一个顺序信赖域法。根据单纯形方法，问题的约束条件被转换为包含解的闭包单纯形。根据目前迭代最优解一定出现在闭包顶点上的理论，不断运用信赖域方法来优化和缩小可行性区域的闭包，最终求得满足精度要求的闭包和相应解。COBYLA 算法是一个不需求导支持和线性约束的优化算法。

L-BFGS-B 算法

Limited-memory-BFGS-Bounded 算法，简称 L-BFGS-B 算法，是一种拟牛顿算法。根据梯度方法的思路，可以得到牛顿法（二阶梯度法），然后使用正定矩阵来近似二阶导数矩阵以减少得到拟牛顿法所需的计算次数。通过继续简化正定矩阵的构建过程，构建近似矩阵的复杂度也得到降低，最终通过适应边界条件以获取该算法。L-BFGS-B 算法是一个需要求导的有界无约束优化算法。

SLSQP 算法

Sequential Least SQuares Programming optimization，简称 SLSQP 算法。该算法是一个依赖 KKT 条件的顺序最小二乘规划算法。从本质上来讲，它是一个用来求解二次规划问题的顺序 (或逐步) 的拟牛顿法。SLSQP 算法是一个需要使用求导方法的有界约束优化算法。它将一般的优化问题转化为二次规划问题。

GRADIENT-DESCENT 算法

梯度下降 (gradient descent) 是一种常见的一阶 (first-order) 优化方法，是求解无约束优化问题最简单、最经典的方法之一。梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是“最速下降法”。最速下降法越接近目标值，步长越小，前进越慢。

参考文献

Chapter 14

接口介绍

以下为 pyChemiQ 所有接口及配置文件的详细参数介绍。

Modules：

14.1 pychemiq.Transform

14.1.1 Module Contents

- *pychemiq.Transform.mapping*

将费米子算符映射为泡利算符的 mapping 子模块。在使用 UCC 拟设搭建参数化的量子线路来制备试验波函数时，我们需要输入酉耦合簇算符的映射类型。这时我们通过 pychemiq.Transform.mapping.MappingType 来指定酉耦合簇算符的映射类型。这里的映射类型需要与哈密顿量的映射方式保持一致，也就是说在计算中我们要保持同一套映射方式。

Classes

class pychemiq.Transform.mapping.**MappingType**

MappingType 这个枚举类有四个不同的取值，分别为：

Bravyi_Kitaev

Jordan_Wigner

Parity

SegmentParity

接口示例：

```
from pychemiq import ChemiQ,QMachineType
from pychemiq.Transform.Mapping import MappingType
from pychemiq.Circuit.Ansatz import UCC

chemiq = ChemiQ()
machine_type = QMachineType.CPU_SINGLE_THREAD
# 使用 JW 映射方式
mapping_type = MappingType.Jordan_Wigner
# 使用 BK 映射方式
# mapping_type = MappingType.Bravyi_Kitaev
# 使用 Parity 映射方式
# mapping_type = MappingType.Parity
# 使用 SegmentParity 映射方式
# mapping_type = MappingType.SegmentParity
chemiq.prepare_vqe(machine_type,mapping_type,2,1,4)
ansatz = UCC("UCCSD",2,mapping_type,chemiq=chemiq)
```

Functions

pychemiq.Transform.mapping.**jordan_wigner**(*fermion*)

将输入的费米子算符通过 Jordan Wigner 变换映射成为泡利算符。

Parameters

fermion (FermionOperator) –输入待映射的费米子算符。

Returns

映射后的泡利算符。泡利算符类。

pychemiq.Transform.mapping.**bravyi_kitaev**(*fermion*)

将输入的费米子算符通过 Bravyi Kitaev 变换映射成为泡利算符。

Parameters

fermion (FermionOperator) –输入待映射的费米子算符。

Returns

映射后的泡利算符。泡利算符类。

pychemiq.Transform.mapping.**parity**(*fermion*)

将输入的费米子算符通过 Parity 变换映射成为泡利算符。

Parameters

fermion (FermionOperator) –输入待映射的费米子算符。

Returns

映射后的泡利算符。泡利算符类。

pychemiq.Transform.mapping.**segment_parity**(*fermion*)

将输入的费米子算符通过 segment_parity 变换映射成为泡利算符。

Parameters

fermion (FermionOperator) –输入待映射的费米子算符。

Returns

映射后的泡利算符。泡利算符类。

接口示例:

下面这个例子中我们使用以上四种映射方式来将二次量子化后氢分子哈密顿量从费米子算符映射成为泡利算符的形式。首先，初始化分子的电子结构参数，得到费米子形式的哈密顿量。

```
from pychemiq import Molecules

multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"

mol = Molecules(
geometry = geom,
basis    = basis,
multiplicity = multiplicity,
charge = charge)
fermion_H2 = mol.get_molecular_hamiltonian()
```

通过 JW 变换得到泡利形式的氢分子哈密顿量并打印结果。

```
from pychemiq.Transform.Mapping import jordan_wigner
pauli_H2 = jordan_wigner(fermion_H2)
print(pauli_H2)
```

```
{
"" : -0.097066,
"X0 X1 Y2 Y3" : -0.045303,
"X0 Y1 Y2 X3" : 0.045303,
"Y0 X1 X2 Y3" : 0.045303,
"Y0 Y1 X2 X3" : -0.045303,
"Z0" : 0.171413,
"Z0 Z1" : 0.168689,
"Z0 Z2" : 0.120625,
"Z0 Z3" : 0.165928,
"Z1" : 0.171413,
```

(continues on next page)

(continued from previous page)

```
"Z1 Z2" : 0.165928,
"Z1 Z3" : 0.120625,
"Z2" : -0.223432,
"Z2 Z3" : 0.174413,
"Z3" : -0.223432
}
```

通过 BK 变换得到泡利形式的氢分子哈密顿量并打印结果。

```
from pychemiq.Transform.Mapping import bravyi_kitaev
pauli_H2 = bravyi_kitaev(fermion_H2)
print(pauli_H2)
```

```
{
"" : -0.097066,
"X0 Z1 X2" : 0.045303,
"X0 Z1 X2 Z3" : 0.045303,
"Y0 Z1 Y2" : 0.045303,
"Y0 Z1 Y2 Z3" : 0.045303,
"Z0" : 0.171413,
"Z0 Z1" : 0.171413,
"Z0 Z1 Z2" : 0.165928,
"Z0 Z1 Z2 Z3" : 0.165928,
"Z0 Z2" : 0.120625,
"Z0 Z2 Z3" : 0.120625,
"Z1" : 0.168689,
"Z1 Z2 Z3" : -0.223432,
"Z1 Z3" : 0.174413,
"Z2" : -0.223432
}
```

通过 Parity 变换得到泡利形式的氢分子哈密顿量并打印结果。

```
from pychemiq.Transform.Mapping import parity
pauli_H2 = parity(fermion_H2)
print(pauli_H2)
```

```
{
"" : -0.097066,
"X0 Z1 X2" : 0.045303,
"X0 Z1 X2 Z3" : 0.045303,
"Y0 Y2" : 0.045303,
"Y0 Y2 Z3" : 0.045303,
```

(continues on next page)

(continued from previous page)

```
"Z0" : 0.171413,
"Z0 Z1" : 0.171413,
"Z0 Z1 Z2" : 0.120625,
"Z0 Z1 Z2 Z3" : 0.120625,
"Z0 Z2" : 0.165928,
"Z0 Z2 Z3" : 0.165928,
"Z1" : 0.168689,
"Z1 Z2" : -0.223432,
"Z1 Z3" : 0.174413,
"Z2 Z3" : -0.223432
}
```

通过 SP 变换得到泡利形式的氢分子哈密顿量并打印结果。

```
from pychemiq.Transform.Mapping import segment_parity
pauli_H2 = segment_parity(fermion_H2)
print(pauli_H2)
```

```
{
"" : -0.097066,
"X0 Z1 X2" : 0.045303,
"X0 Z1 X2 Z3" : 0.045303,
"Y0 Z1 Y2" : 0.045303,
"Y0 Z1 Y2 Z3" : 0.045303,
"Z0" : 0.171413,
"Z0 Z1" : 0.171413,
"Z0 Z1 Z2" : 0.165928,
"Z0 Z1 Z2 Z3" : 0.165928,
"Z0 Z2" : 0.120625,
"Z0 Z2 Z3" : 0.120625,
"Z1" : 0.168689,
"Z1 Z2 Z3" : -0.223432,
"Z1 Z3" : 0.174413,
"Z2" : -0.223432
}
```

14.2 pychemiq.Circuit

14.2.1 Module Contents

- *pychemiq.Circuit.Ansatz*

构建量子线路拟设的 Ansatz 子模块。

Functions

pychemiq.Circuit.Ansatz.**UCC**(*ucc_type*, *n_electrons*, *mapping_type*, *chemiq=None*)

使用酉耦合簇算符构建量子线路拟设。

Parameters

- **ucc_type** (*str*) –输入酉耦合簇的激发水平。目前可选：UCCS、UCCD、UCCSD。
- **n_electrons** (*int*) –输入分子体系的电子数。
- **mapping_type** (MappingType) –输入酉耦合簇算符的映射类型。详见 pychemiq.Transform.Mapping。
- **chemiq** (ChemiQ) –指定 chemiq 类。详见 pychemiq.ChemiQ。

Returns

输出指定激发水平的 AbstractAnsatz 类。

pychemiq.Circuit.Ansatz.**HardwareEfficient**(*n_electrons*, *chemiq=None*)

使用 HardwareEfficient 构建量子线路拟设。

Parameters

- **n_electrons** (*int*) –输入分子体系的电子数。
- **chemiq** (ChemiQ) –指定 chemiq 类。详见 pychemiq.ChemiQ。

Returns

输出该拟设的 AbstractAnsatz 类。

pychemiq.Circuit.Ansatz.**SymmetryPreserved**(*n_electrons*, *chemiq=None*)

使用 SymmetryPreserved 构建量子线路拟设。

Parameters

- **n_electrons** (*int*) –输入分子体系的电子数。
- **chemiq** (ChemiQ) –指定 chemiq 类。详见 pychemiq.ChemiQ。

Returns

输出该拟设的 AbstractAnsatz 类。

pychemiq.Circuit.Ansatz.**UserDefine**(*n_electrons*, *circuit=None*, *fermion=None*, *chemiq=None*)

使用用户自定义的方式构建量子线路拟设。

Parameters

- **n_electrons** (*int*) –输入分子体系的电子数。
- **circuit** (*str*) –构建量子线路的 originIR 字符串。
- **fermion** (FermionOperator) –构建量子线路的费米子算符类。
- **chemiq** (ChemiQ) –指定 chemiq 类。详见 pychemiq.ChemiQ。

Returns

输出自定义拟设的 AbstractAnsatz 类。

Note: Ansatz 模块前三个函数的详细调用示例请参见基础教程中的拟设教程。最后一个函数的调用示例请参见进阶教程中的量子线路教程。

14.3 pychemiq.Optimizer

14.3.1 Module Contents

Functions

pychemiq.Optimizer.**vqe_solver**(*method='NELDER-MEAD'*, *ansatz=None*, *pauli=None*, *init_para=None*, *chemiq=None*, *Learning_rate=0.1*, *Xatol=0.0001*, *Fatol=0.0001*, *MaxFCalls=200*, *MaxIter=200*)

该类为 VQE 求解器，在参数中需要指定经典优化器方法、拟设、分子的泡利哈密顿量、初始参数、chemiq 类。

Parameters

- **method** (*str*) –指定经典优化器方法。目前 pyChemiQ 支持的方法有 NELDER-MEAD、POWELL、COBYLA、L-BFGS-B、SLSQP 和 Gradient-Descent。若不指定，默认使用 NELDER-MEAD 优化器。
- **ansatz** (*AbstractAnsatz*) –指定拟设类。详见 pychemiq.Circuit.Ansatz。
- **pauli** (PauliOperator) –指定分子的泡利哈密顿量。泡利算符类。详见 pychemiq.PauliOperator。
- **init_para** (*list[float]*) –指定初始参数。
- **chemiq** (ChemiQ) –指定 chemiq 类。详见 pychemiq.ChemiQ。

- **Learning_rate** (*float*) –指定学习率。选择与梯度相关的优化器方法需要此参数。默认为 0.1。
- **Xatol** (*float*) –变量的收敛阈值。默认为 0.0001。
- **Fatol** (*float*) –函数值的收敛阈值。默认为 0.0001。
- **MaxFCalls** (*int*) –函数最大可调用次数。默认为 200。
- **MaxIter** (*int*) –最大优化迭代次数。默认为 200。

Returns

QOptimizationResult 类。详见 pyqpanda.QOptimizationResult 。

Note: 优化器的更多示例以及调用外部的 scipy.optimize 库来实现经典优化等方式，请参见进阶教程中的优化器教程。

14.4 pychemiq.Utils

14.4.1 Module Contents

Functions

pychemiq.Utils.**get_cc_n_term**(*n_qubits*, *n_elec*, *excited_level*)

得到指定激发水平的耦合簇算符项数。例如：对于 4 个量子比特，2 电子体系的单双激发耦合簇算符，自旋轨道 0 和 1 为占据态，故耦合簇项数为五项：0->2,0->3,1->2,1->3,01->23。

Parameters

- **n_qubits** (*int*) –输入计算所需的量子比特数。
- **n_elec** (*int*) –输入分子体系的电子数。
- **excited_level** (*str*) –输入耦合簇算符的激发水平。目前可选：CCS、CCD、CCSD。

Returns

输出指定激发水平的耦合簇算符项数。整数型。

pychemiq.Utils.**get_cc**(*n_qubits*, *n_elec*, *para*, *excited_level='SD'*)

得到含参的指定激发水平的耦合簇算符。例如：对于 4 个量子比特，2 电子体系的单双激发耦合簇算符，自旋轨道 0 和 1 为占据态，故激发后的耦合簇项为：0->2,0->3,1->2,1->3,01->23。输出的费米子算符为：{ { “2+ 0” :para[0]},{ “3+ 0” :para[1]},{ “2+ 1” :para[2]},{ “3+ 1” :para[3]}, { “3+ 2+ 1 0” :para[4]} }

Parameters

- **n_qubits** (*int*) –输入计算所需的量子比特数。

- **n_elec** (*int*) –输入分子体系的电子数。
- **para** (*list[float]*) –输入初始参数列表。
- **excited_level** (*str*) –输入耦合簇算符的激发水平。目前可选：CCS、CCD、CCSD。默认为单双激发耦合簇算符 (CCSD)。

Returns

输出指定激发水平的耦合簇算符。费米子算符类。

pychemiq.Utils.**transCC2UCC**(*Pauli*)

只有酉算子才可以放在量子线路上进行模拟。该函数在耦合簇算符的基础上，将其中不是厄米矩阵的算子删去，构造出“酉算子版本”的耦合簇算符。

Parameters

Pauli (PauliOperator) –输入耦合簇算符。泡利算符类。

Returns

输出酉耦合簇算符。泡利算符类。

接口示例：

下面这个例子中，我们计算 4 个量子比特，2 电子体系的单双激发耦合簇算符，并将其转化为可以直接构建量子线路拟设的酉耦合簇算符。

```
from pychemiq.Utils import get_cc_n_term,get_cc,transCC2UCC
from pychemiq.Transform.Mapping import jordan_wigner
import numpy as np

# 计算 4 个量子比特，2 电子体系的单双激发耦合簇算符的项数来初始化参数列表，这里我们先令初参为全 1 的列表
n_para = get_cc_n_term(4,2,"CCSD")
para = np.ones(n_para)

# 打印设定初参后，4 个量子比特，2 电子体系的单双激发耦合簇算符
cc_fermion = get_cc(4,2,para,"CCSD")
print(cc_fermion)
```

打印的结果为：

```
{
2+ 0 : 1.000000
3+ 0 : 1.000000
2+ 1 : 1.000000
3+ 1 : 1.000000
3+ 2+ 1 0 : 1.000000
}
```

```
# 接着使用 JW 映射将费米子算符映射成泡利算符
cc_pauli = jordan_wigner(cc_fermion)
# 将非酉耦合簇算符删去，留下酉耦合簇算符
ucc_pauli = transCC2UCC(cc_pauli)
print(ucc_pauli)
```

打印的结果为：

```
{
"X0 X1 X2 Y3" : -0.125000,
"X0 X1 Y2 X3" : -0.125000,
"X0 Y1 X2 X3" : 0.125000,
"X0 Y1 Y2 Y3" : -0.125000,
"X0 Z1 Y2" : 0.500000,
"X0 Z1 Z2 Y3" : 0.500000,
"X1 Y2" : 0.500000,
"X1 Z2 Y3" : 0.500000,
"Y0 X1 X2 X3" : 0.125000,
"Y0 X1 Y2 Y3" : -0.125000,
"Y0 Y1 X2 Y3" : 0.125000,
"Y0 Y1 Y2 X3" : 0.125000,
"Y0 Z1 X2" : -0.500000,
"Y0 Z1 Z2 X3" : -0.500000,
"Y1 X2" : -0.500000,
"Y1 Z2 X3" : -0.500000
}
```

Classes：

14.5 pychemiq.Molecules

14.5.1 Classes

class Molecules(*geometry=None*, *basis=None*, *multiplicity=None*, *charge=0*, *active=None*, *nfrozen=None*)

初始化分子的电子结构参数，包括电荷、基组、原子坐标、自旋多重度等

Parameters

- **geometry** (*str*) –输入分子中原子的类型和坐标。可以为字符串类型或者字符串列表。例如：geometry = “H 0 0 0,H 0 0 0.74” 或者 geometry = [“H 0 0 0” ,” H 0 0 0.74”]
- **basis** (*str*) –输入执行计算的基组水平。目前支持的基组为 MINI、sto-3G、sto-6G、3-21G、6-31G 等高斯型函数基组。不支持极化与弥散基组。

- **multiplicity** (*int*) –输入分子体系的自旋多重度。与分子总自旋量子数的关系为 M=2S+1。目前 pyChemiQ 只支持 RHF 单重态计算，UHF 以及 ROHF 正在开发中。
- **charge** (*int*) –输入分子体系的电荷。
- **active** (*list[int]*) –分子体系的活性空间，格式为 [m,n]，其中 m 为活性轨道的数目，n 为活性电子的数目。默认不设置活性空间。
- **nfrozen** (*int*) –分子体系的冻结轨道数目，从能量最低的分子轨道算起开始冻结该轨道及轨道上的电子。默认不设置冻结轨道。

Returns

输出执行 HF 计算的结果。

Attributes

n_atoms

得到分子体系中的原子数

n_electrons

得到分子体系中的电子数

n_orbitals

得到分子体系的总分子轨道数

n_qubits

得到计算所需要的总量子比特数 (即自旋轨道数量，2* 分子轨道数)

hf_energy

得到 HF 计算的能量 (单位:Hartree)

nuclear_repulsion

得到分子体系的核间斥力 (单位:Hartree)

canonical_orbitals

得到分子体系的正则轨道系数 (即分子轨道系数)

orbital_energies

得到体系每个分子轨道的能量

overlap_integrals

得到分子体系的重叠积分

one_body_integrals

得到分子体系的单电子积分

two_body_integrals

得到分子体系的双电子积分

Methods

get_molecular_hamiltonian()

得到初始化后的分子体系的哈密顿量

接口示例：

```
from pychemiq import Molecules
multiplicity = 1
charge = 0
basis =  "sto-3g"
geom = "H 0 0 0,H 0 0 0.74"
mol = Molecules(
      geometry = geom,
      basis    = basis,
      multiplicity = multiplicity,
      charge = charge)
```

调用以下接口得到该分子体系的信息：

```
print("The number of atoms is", mol.n_atoms)
print("The number of electrons is", mol.n_electrons)
print("The number of orbitals is", mol.n_orbitals)
print("The number of qubits is", mol.n_qubits)
print("The Hartree-Fock energy is", mol.hf_energy)
print("The nuclear repulsion is", mol.nuclear_repulsion)
```

```
The number of atoms is 2
The number of electrons is 2
The number of orbitals is 2
The number of qubits is 4
The Hartree-Fock energy is -1.1167593072992057
The nuclear repulsion is 0.7151043390810812
```

```
print("The canonical orbitals are\n", mol.canonical_orbitals)
print("The orbital energies are", mol.orbital_energies)
print("The overlap integrals are\n", mol.overlap_integrals)
```

```
The canonical orbitals are
 [[-0.54884228  1.21245192]
 [-0.54884228 -1.21245192]]

The orbital energies are [-0.57855386  0.67114349]

The overlap integrals are
```

(continues on next page)

(continued from previous page)

```
 [[1.         0.65987312]
 [0.65987312 1.        ]]
```

```
print("The one body integrals are\n", mol.one_body_integrals)
print("The two body integrals are\n", mol.two_body_integrals)
```

```
The one body integrals are
 [[-1.25330979e+00  0.00000000e+00]
 [ 4.16333634e-17 -4.75068849e-01]]

The two body integrals are
 [[[[ 6.74755927e-01 -1.11022302e-16]
    [-8.32667268e-17  6.63711401e-01]]

   [[-3.46944695e-17  1.81210462e-01]
    [ 1.81210462e-01  0.00000000e+00]]]


 [[[-4.85722573e-17  1.81210462e-01]
    [ 1.81210462e-01 -2.22044605e-16]]

   [[ 6.63711401e-01 -2.22044605e-16]
    [-1.66533454e-16  6.97651504e-01]]]]
```

```
print("The molecular hamiltonian is", mol.get_molecular_hamiltonian())
```

```
The molecular hamiltonian is {
: 0.715104
0+ 0 : -1.253310
1+ 0+ 1 0 : -0.674756
1+ 0+ 3 2 : -0.181210
1+ 1 : -1.253310
2+ 0+ 2 0 : -0.482501
2+ 1+ 2 1 : -0.663711
2+ 1+ 3 0 : 0.181210
2+ 2 : -0.475069
3+ 0+ 2 1 : 0.181210
3+ 0+ 3 0 : -0.663711
3+ 1+ 3 1 : -0.482501
3+ 2+ 1 0 : -0.181210
3+ 2+ 3 2 : -0.697652
3+ 3 : -0.475069
```

(continues on next page)

(continued from previous page)

```
}
```

14.6 pychemiq.ChemiQ

14.6.1 Classes

class ChemiQ

封装好的量子线路类，为 VQE 算法中在量子计算机/虚拟机上进行的部分。包括搭建参数化的量子线路来制备试验波函数和对哈密顿量的各个子项进行测量与求和。

prepare_vqe(*machine_type*, *mapping_type*, *n_elec*, *pauli_size*, *n_qubits*)

准备 VQE 算法量子线路的函数。

Parameters

- **machine_type** (*QMachineType*) –输入量子模拟器类型。目前 pyChemiQ 仅支持单线程 CPU，即 QMachineType.CPU_SINGLE_THREAD。含噪声量子模拟器的接入还在进行中。该类的介绍详见 pyqpanda.QMachineType 。
- **mapping_type** (MappingType) –输入映射类型。详见 pychemiq.Transform.Mapping。
- **n_elec** (*int*) –输入分子体系的电子数。
- **pauli_size** (*int*) –输入泡利哈密顿量的项数。
- **n_qubits** (*int*) –输入计算所需要的总量子比特数。

Returns

void

getExpectationValue(*index*, *fcalls*, *task_index*, *qvec*, *hamiltonian*, *para*, *ansatz*, *extra_measure*)

对线路进行测量得到哈密顿量期望值。

Parameters

- **index** (*int*) –输入体系编号。
- **fcalls** (*int*) –输入函数调用次数。
- **task_index** (*int*) –任务编号。
- **qvec** (*QVec*) –存储量子比特的数组。该类的介绍详见 pyqpanda.QVec 。
- **hamiltonian** (*Hamiltonian*) –输入 Hamiltonian 类。PauliOperator 中的哈密顿量是字符串形式，不利用后续的处理，Hamiltonian 在存储方式上将泡利算符其转换成自定义的 Hamiltonian 类，方便提取每一项的信息。

- **para** (*list[float]*) –指定初始待优化参数。
- **ansatz** (*AbstractAnsatz*) –指定拟设类。详见 pychemiq.Circuit.Ansatz。
- **extra_measure** (*bool*) –用于区分噪声模拟和非噪声模拟。布尔值。设置 False 为非噪声模拟。

Returns

哈密顿量期望值，即基态能量。双精度浮点数。

getLossFuncValue(*index*, *para*, *grad*, *iters*, *fcalls*, *pauli*, *qvec*, *ansatz*)

得到损失函数的值。

Parameters

- **index** (*int*) –输入体系编号。
- **para** (*list[float]*) –指定初始待优化参数。
- **grad** (*list[float]*) –指定初始待优化梯度。
- **iters** (*int*) –输入函数迭代的次数。
- **fcalls** (*int*) –输入函数调用的次数。
- **pauli** (PauliOperator) –指定分子的泡利哈密顿量。泡利算符类。详见 pychemiq.PauliOperator。
- **qvec** (*QVec*) –存储量子比特的数组。该类的介绍详见 pyqpanda.QVec 。
- **ansatz** (*AbstractAnsatz*) –指定拟设类。详见 pychemiq.Circuit.Ansatz。

Returns

损失函数的值。dict 类型。

get_energy_history()

Returns

每一次函数迭代后的能量值。双精度浮点数数组。

接口示例：

```
from pychemiq import Molecules,ChemiQ,QMachineType,PauliOperator
from pychemiq.Transform.Mapping import MappingType
from pychemiq.Circuit.Ansatz import UCC

chemiq = ChemiQ()
machine_type = QMachineType.CPU_SINGLE_THREAD
mapping_type = MappingType.Jordan_Wigner
chemiq.prepare_vqe(machine_type,mapping_type,2,1,4)
```

(continues on next page)

(continued from previous page)

```
ansatz = UCC("UCCD",2,mapping_type,chemiq=chemiq)
pauli = PauliOperator("Z0 Z1 ",0.1)
# 期望值函数与代价函数
H = pauli.to_hamiltonian(True)
result1 = chemiq.getExpectationValue(0,0,0,chemiq.qvec,H,[0],ansatz,False)
result2 = chemiq.getLossFuncValue(0,[0],[0],0,0,pauli,chemiq.qvec,ansatz)
energies = chemiq.get_energy_history()

print(result1)
print(result2)
print(energies)
```

打印得到的结果为：

```
0.1
('', 0.1)
[0.1]
```

14.7 pychemiq.FermionOperator

14.7.1 Classes

class FermionOperator(*{fermion_string: coefficient}*)

Parameters

- **fermion_string** (*str*) –字符形式的费米算符。
- **coefficient** (*float*) –该项费米算符的系数。

Returns

费米算符类。

normal_ordered()

normal_ordered 接口对费米子算符进行整理。在这个转换中规定所作用的轨道编码从高到低进行排序，并且产生算符出现在湮没算符之前。

data()

费米子算符类还提供了 data 接口，可以返回费米子算符内部维护的数据。

Note: 该类的详细介绍请参见进阶教程中的算符类教程。

14.8 pychemiq.PauliOperator

14.8.1 Classes

class PauliOperator(*{pauli_string: coefficient}*)

Parameters

- **pauli_string**(*str*) –字符形式的泡利算符。
- **coefficient**(*float*) –该项泡利算符的系数。

Returns

泡利算符类。

get_max_index()

得到最大索引值。如果是空的泡利算符项调用 get_max_index() 接口则返回 SIZE_MAX（具体值取决于操作系统），否则返回其最大索引值。

data()

泡利算符类提供了 data 接口，可以返回泡利算符内部维护的数据。

Note: 该类的详细介绍请参见进阶教程中的算符类教程。

Chapter 15

配置文件参数介绍

除了调用 pyChemiQ 的基础接口进行计算，您也可以设置配置文件直接运算，更多高级功能开放在配置文件里使用。您可以通过使用内置优化方法缩短量子线路，减少运行时间，还有切片数设置、拟设截断、MP2 初参设置等丰富功能的接口可调用。如果您想试用 pyChemiQ 这些高级功能，请前往 官网 申请授权码。ChemiQ 和 pyChemiQ 的 license 是通用的，如果您已有 ChemiQ 的 license，直接填进配置文件中全局设置的 license 参数即可。在设置完配置文件后，在终端输入下面的命令行即可运行计算：

```
from pychemiq import directly_run_config
directly_run_config("test.chemiq")  # 将 test 替换成您配置文件的名称
```

配置文件通常为.chemiq 为后缀的文件，主要分五个方面进行设置，详细的参数介绍及默认参数如下：

1. **全局设置 (general settings)**
    - task : 设置计算类型 [energy/MD]，即单点能计算/分子动力学模拟。默认值 energy。
    - backend : 设置执行计算的后端类型 [CPU_SINGLE_THREAD]。目前仅支持单线程 CPU，更多后端类型的接入还在进行中。
    - print_out : 设置是否打印 scf 迭代过程。默认为 F。
    - print_iters : 设置是否打印优化器迭代过程。默认为 F。
    - console_level : 设置终端打印日志级别，0 为输出，6 为不输出。默认值为 0。
    - logfile_name : 设置日志文件名，默认为空。例如，设置的日志名为 chemiq.log, 则最终输出的日志名为 chemiq-当天日期.log。只有设置了日志文件名，才可以输出日志文件。
    - logfile_level : 设置文件输出日志级别，0 为输出，6 为不输出。默认值为 6。
    - license : 设置授权序列号。请前往 官网 申请授权码。
2. **分子参数设置 (Molecule specification)**
    - geoms : 设置分子坐标，其中原子类型和原子坐标以空格进行分隔。
    - bohr : 坐标单位是否设置为 bohr。布尔值。默认为 F，用 angstrom 为单位。

- charge : 设置体系的电荷数，电荷数为正时无正号，为负时写负号，默认值 0。
- spin : 设置自旋多重度 (M=2S+1)，默认值 1。
- basis : 设置计算所需的基组水平 [MINI/sto-3G/sto-6G/3-21G/6-31G]。
- pure : 使用球谐型还是笛卡尔型 Gaussian 函数。默认为 T, 即使用球谐型 Gaussian 函数。
- local : 是否局域化 HF 轨道。默认为 F, 即不局域化 HF 轨道。
- active : 设置活性空间，以逗号为分隔符，默认值不设置活性空间。例如 active = 4, 4 中第一个 4 表示 4 个空间活性轨道，第二个 4 表示在活性空间中有 4 个电子。
- nfrozen : 设置冻结空间轨道数目，默认为 0。注：active 与 nfrozen 不能同时使用。
- mix_scf : 设置此参数可以有效解决 SCF 不收敛问题。方法原理为阻尼方法 (Damping)。将构建第 n+1 步 Fock 矩阵的密度矩阵 D(n+1) 变为 w*D(n-1)+(1-w)*D(n)，此处的 w 即为设置的参数。参数平均化后的密度矩阵削弱了当前步与上一步密度矩阵之间的差异，使密度矩阵随迭代变化更为平滑，帮助收敛。该参数取值为 [0.0, 1.0], 默认为 0.5。

3. **拟设参数设置 (ansatz settings)**

   设置量子线路拟设类型 [UCC/Hardware-efficient/Symmetry-preserved/User-define]。选择前三个类型的拟设，量子线路自动生成，选择最后一个线路拟设需要在参数 circuit 中输入 originIR 格式的量子线路。详见：originIR 格式介绍 。

   - mapping : 设置映射 [JW/P/BK/SP]。这几种映射方法分别为 Jordan-Wigner Transform，Parity Transform, Bravyi-Kitaev Transform, Segment Parity Transform。
   - excited_level : 设置激发水平 [S/D/SD]，仅当 ansatz 为 UCC 时生效。
   - restricted : 对激发项进行限制，制备更少组态波函数的叠加态以缩短线路。默认为 T。仅当 ansatz 为 UCC 时生效。
   - cutoff : 根据 MP2 的初参对 UCC 拟设的激发项进行截断。仅当 ansatz 为 UCC 且 init_para_type=MP2 时生效。默认为 F。
   - reorder : 按顺序排列量子比特，前半部分量子比特编码自旋向上，后半部分量子比特编码自旋向下，此参数设置为 T 时可以减少量子比特的使用。当 mapping 为 P 和 BK 时生效。默认为 F。
   - circuit : 通过 originIR 字符串设置线路，仅当 ansatz 为 User-define 时生效。

4. **优化器设置 (optimizer settings)**

   设置经典优化器类型 [Nelder-Mead/Powell/Gradient-Descent/COBYLA/L-BFGS-B/SLSQP]。

   - init_para_type : 设置构造初始参数的方式 [Zero/Random/input/MP2]，其中 Zero 表示初参为全零，Random 表示初参为 [0,1) 区间内的随机数，input 表示自定义初参，MP2 表示为二阶微扰得到的初参结果。其中 MP2 只在拟设为 UCCD 和 UCCSD 时可用。初参默认为 Zero。
   - slices : 设置切片数，即量子线路重复次数，默认值 1。
   - learning_rate : 设置学习率。默认值 0.1。
   - iters : 设置迭代次数，默认值 1000。

- fcalls : 设置函数调用次数，默认值 1000。
- xatol : 设置变量收敛阈值，默认值 1e-4。
- fatol : 设置期望值收敛阈值，默认值 1e-4。

5. **分子动力学参数设置 (molecular dynamics parameter settings)**

- HF : 设置关联采样方法。默认为 1。
- axis : 以字符串形式设置体系沿特定方向运动，格式为”x y z”。
- save_trajectory : 设置保存分子坐标文件的名称。默认为”traj.csv”。
- save_topology : 设置保存分子拓扑文件的名称。默认为”topology.txt”。
- velocity : 设置原子的初始速度，原子间以逗号分隔，”0.1 0.2 0.3, -0.1 -0.2 -0.3"，单位 A/fs，默认值全 0。
- step_size : 设置步长，大于 0，单位 fs，默认 0.2。
- step_number : 设置总步数，大于 1，默认 100。
- delta_r : 设置差分坐标大小，大于 0，默认 0.001。

下面我们给出一个使用配置文件计算氢分子单点能的案例。基组使用 sto-3G，拟设使用 UCCSD，映射使用 BK，优化器使用 NELDER-MEAD。初参为 MP2。

```
general = {
    task    = energy
    backend = CPU_SINGLE_THREAD
    license = XXXXX
}

mole = {
    geoms = {
        H 0 0 0
        H 0 0 0.74
    }
    bohr    = F
    charge  = 0
    spin    = 1
    basis   = sto-3G
    pure    = T
    local   = F
}

ansatz = UCC {
    excited_level = SD
    restricted    = T
```

(continues on next page)

(continued from previous page)

```
    cutoff        = T
    mapping       = BK
    reorder       = F
}

optimizer = NELDER-MEAD {
    learning_rate                = 0.1
    init_para_type               = MP2
    slices                       = 1
    iters                        = 1000
    fcalls                       = 1000
    xatol                        = 1e-6
    fatol                        = 1e-6
}
```

第二个示例我们计算氢分子的势能曲线，这里我们以扫描五个点为例，每个点间隔 0.1 angstrom。基组使用 sto-3G，活性空间使用 [2，2]，拟设使用自定义线路，映射使用 parity，优化器使用 SLSQP。初参为零。

```
general = {
    task    = energy
    backend = CPU_SINGLE_THREAD
    license = XXXXX
}

mole = {
    geoms = {
        H 0 0 0
        H 0 0 0.54;
        H 0 0 0
        H 0 0 0.64;
        H 0 0 0
        H 0 0 0.74;
        H 0 0 0
        H 0 0 0.84;
        H 0 0 0
        H 0 0 0.94
    }
    bohr    = F
    charge  = 0
    spin    = 1
    basis   = sto-3G
    pure    = T
    local   = F
    active = 2,2
```

(continues on next page)

(continued from previous page)

```
}

ansatz = User-define {
    circuit = {
        QINIT 4
        CREG 4
        CNOT q[1],q[0]
        CNOT q[2],q[1]
        CNOT q[3],q[2]
        H q[1]
        H q[3]
        S q[1]
}
    mapping       = P
    reorder       = T
}

optimizer = SLSQP {
    learning_rate                = 0.1
    init_para_type               = Zero
    slices                       = 1
    iters                        = 1000
    fcalls                       = 1000
    xatol                        = 1e-6
    fatol                        = 1e-6
}
```

第三个示例我们计算氢化锂分子的分子动力学轨迹。基组使用 3-21G，活性空间使用 [4，4]，拟设使用 Hardware-efficient，映射使用 JW，优化器使用 L-BFGS-B。初参为随机数。

```
general = {
    task    = MD
    backend = CPU_SINGLE_THREAD
    license = XXXXX
}

mole = {
    geoms = {
        H 0 0 0.38
        Li 0 0 -1.13
    }
    bohr    = F
    charge  = 0
    spin    = 1
```

(continues on next page)

(continued from previous page)

```
    basis    = 3-21G
    pure     = T
    local    = F
    active = 4,4
}

ansatz = Hardware-efficient {
    mapping       = JW
    reorder       = F
}

optimizer = L-BFGS-B {
    learning_rate                = 0.1
    init_para_type               = Random
    slices                       = 1
    iters                        = 1000
    fcalls                       = 1000
    xatol                        = 1e-6
    fatol                        = 1e-6
}

MD = 1 {
    velocity           = 0.0
    step_size          = 0.2
    step_number        = 100
    delta_r            = 0.001
}
```

Chapter 16

问题与建议

如果您在使用 pyChemiQ 遇到了问题或者希望给我们提出改进建议，请联系 dqa@originqc.com，或者通过 GitHub Issues 来提交问题与建议。

Chapter 17

FAQ

Q: pyChemiQ 对其他函数包有依赖吗?

A: pyChemiQ 依赖 numpy 函数包（版本大于 1.19）。若使用 Linux 版本安装，需保证系统 glibc 版本大于等于 2.29。

Python Module Index

p

Index

O

P

S

T

U

V