# PhotosynthesisExample

March 21, 2024

## 1 Ground State Energy Estimation for Photosynthesis with Quantum Circuits

Artificial photosynthesis mimics natural photosynthesis by absorbing solar light and splitting water into O2, protons (H+) and electrons (e).

The electrons extracted from water can reduce protons or CO2 to produce energy carrier fuels such as H2 or hydrocarbons

Water oxidation reaction 2H2O -> O2 + 4(H+) + 4e

Proton reduciton reaction: 2(H+) + 2e -> H2

CO2 reduciton reactions: - CO2 + 2(H+) + 2e -> CO + H2O (CO2 Reduction 1)

- CO2 + 6(H+) + 6e -> CH3OH + H2O (CO2 Reduction 2)
- CO2 + 8(H+) + 8e -> CH4 + 2H2O (CO2 Reduction 3)

Typical examples of water oxidation: Co_4O_4 catalysis (https://pubs.acs.org/doi/10.1021/ja202320q)

[ ]:
```
import re
import sys
import time
import numpy as np
from openfermionpyscf import run_pyscf
from openfermion.chem import MolecularData
from pyLIQTR.PhaseEstimation.pe import PhaseEstimation
from qca.utils.utils import extract_number, circuit_estimate
```

[ ]:
```
def load_pathway_xyz(fname, pathway=None):
    coordinates_pathway = []
    with open(fname) as f:
        data = f.readlines()
        coordinates_pathway = []
        for k, line in enumerate(data):
            if 'multiplicity' in line or 'charge' in line:
                coords_list = []
                geo_name = None
                if len(line.split(',')) > 2:
```

```
                    geo_name = line.split(',')[2]

                # Use a regular expression to extract the multiplicity value
                match = re.search(r"multiplicity\s*=\s*(\d+)", line)
                if match:
                    multiplicity = int(match.group(1))  # Convert the string to␣
 ↪an integer
                match = re.search(r"charge\s*=\s*(\d+)", line)
                if match:
                    charge = int(match.group(1))  # Convert the string to an␣
 ↪integer

                nat = int(data[k-1].split()[0])
                print(f'{geo_name} Multiplicity: {multiplicity}  total no. of␣
 ↪atoms={nat}, charge = {charge}')
                coords_list.append([nat, charge, multiplicity])
                for i in range(nat):
                    tmp = data[k+1+i].split()
                    aty = tmp[0]
                    xyz = [float(tmp[i]) for i in range(1,4)]
                    coords_list.append([aty, xyz])

                if geo_name is not None and pathway is not None:
                    order = extract_number(geo_name)
                    if order is not None and order in pathway:
                        coordinates_pathway.append(coords_list)
                else:
                    coordinates_pathway.append(coords_list)
    return coordinates_pathway
```

[ ]:
```
molecular_hamiltonians = []
pathway0 = [5, 10, 28, 29, 30, 31, 32, 33]
pathway1 = [2, 1, 14, 15, 16, 17, 18, 19]
pathway2 = [3, 1, 14, 15, 16, 20, 21, 22, 23]
pathway3 = [27, 1, 14, 15, 16, 24, 25, 26]
# Water oxidation via Co4O4 catalyst.
# water oxidation
coordinates_pathway = load_pathway_xyz('../data/water_oxidation_Co4O4.xyz',␣
 ↪pathway=pathway0)

# co2 reduction
coordinates_pathway = load_pathway_xyz('../data/CO2_reduciton_CoPc.xyz')

# Set calculation parameters.
run_scf = 1
run_mp2 = 0
run_cisd = 0
```

```
run_ccsd = 0
run_fci = 0

# Set molecule parameters.
basis = 'sto-3g'
multiplicity = 1
n_points = 40
bond_length_interval = 3.0 / n_points
active_space_frac = 5 # 1 over n
```

[ ]:
```
print('\nGenerating the electronic Hamiltonain along the reaction pathway!\n')

if len(coordinates_pathway) > 0:
    # generate the Hamiltonians
    for coords in coordinates_pathway:
        nat, charge, multi = [int(coords[0][j]) for j in range(3)]
        print(f'\nGenerating the qubit Hamiltonian for the molecule: \n␣
 ↪{coords}\n')

        # set molecular geometry in pyscf format
        basis = 'cc-pvdz'
        #basis = "STO3G"
        geometry = []
        for k, coord in enumerate(coords[1:]):
            coord_str = ' '.join(map(str, coord[1]))
            if k == nat-1:
                coord_str = f'{coord[0]} {coord_str}'
            else:
                coord_str = f'{coord[0]} {coord_str};\n'
            atom = (coord[0], tuple(coord[1]))
            geometry.append(atom)

        molecule = MolecularData(geometry, basis, multi, charge=charge,␣
 ↪description='catalyst')

        print(f'no, of atoms = {len(geometry)}')
        # Run pyscf.
        molecule = run_pyscf(molecule,
                             run_scf=run_scf,
                             run_mp2=run_mp2,
                             run_cisd=run_cisd,
                             run_ccsd=run_ccsd,
                             run_fci=run_fci)

        print(f'number of orbitals           = {molecule.n_orbitals}')
        print(f'number of electrons          = {molecule.n_electrons}')
```

```
        print(f'number of qubits            = {molecule.n_qubits}')
        print(f'Hartree-Fock energy         = {molecule.hf_energy}')
        nocc = molecule.n_electrons // 2
        nvir = molecule.n_orbitals - nocc
        sys.stdout.flush()

        # get molecular Hamiltonian
        active_space_start =  nocc - nocc // active_space_frac # start index of␣
 ↪active space
        active_space_stop = nocc + nvir // active_space_frac   # end index of␣
 ↪active space

        print(f'active_space start = {active_space_start}')
        print(f'active_space stop  = {active_space_stop}')


        molecular_hamiltonian = molecule.get_molecular_hamiltonian(
        occupied_indices=range(active_space_start),
        active_indices=range(active_space_start, active_space_stop)
        )

        # shifted by HF energy
        molecular_hamiltonian -= molecule.hf_energy
        molecular_hamiltonians.append(molecular_hamiltonian)
        sys.stdout.flush()
        sys.exit()
```

[ ]:
```
E_min = -4000
E_max = -3000
omega = E_max - E_min
t = 2*np.pi/omega
phase_offset = E_max * t
for i in molecular_hamiltonians:
    molecular_hamiltonian = molecular_hamiltonians[i]
    trotter_order = 2
    trotter_steps = 1
    n_qubits = molecular_hamiltonian.n_qubits
    # note that we would actually like within chemical precision
    # which should take > 10 bits of precision, it just takes a
    # really long time to run so a scaling argument will be needed
    bits_precision = 1

    gse_args = {
        'trotterize' : True,
        'mol_ham'    : molecular_hamiltonian,
        'ev_time'    : 1,
        'trot_ord'   : trotter_order,
```

```
    'trot_num'    : trotter_steps
}


init_state = [0] * n_qubits

print('starting')
t0 = time.perf_counter()
gse_inst = PhaseEstimation(
    precision_order=bits_precision,
    init_state=init_state,
    phase_offset=phase_offset,
    include_classical_bits=False,
    kwargs=gse_args
)
gse_inst.generate_circuit()
t1 = time.perf_counter()
print(f'Co4O4 time to generate high level number {i} : {t1 - t0}')
gse_circuit = gse_inst.pe_circuit

print('Estimating Co4O4 circuit {i}')
t0 = time.perf_counter()
circuit_estimate(gse_circuit,
                 outdir='GSE/',
                 circuit_name=f'Co4O4_{i}',
                 trotter_steps=trotter_steps,
                 write_circuits=True
                 )
t1 = time.perf_counter()
print(f'Time to estimate Co4O4: {t1-t0}')
```