# HQCS 2024 - Group Assignment

**Rules of the game:**

- Each solution has to be properly explained, otherwise no points will be granted;
- Only one person from the group should perform the submission;
- The use of generative AI for anything different from rephrasing text **is not permitted**;
- Submissions coming after the deadline will not be considered;
- For any question, feel free to reach out by using the TUWEL forum;
- Please submit your solution using the following format: HQCS-GROUPN-NAME-SURNAME-ASSIGNMENT3.{zip, tar, bz2, gz}.

## 1 Introduction

This assignment aims to introduce you to the principles and practical implementation of hybrid quantum-classical systems. You will leverage IBM's Qiskit framework to compute molecular energies using quantum algorithms and integrate these results into a classical molecular dynamics (MD) simulation.

Working in groups of up to three, your team will:

- Use Qiskit to implement a Variational Quantum Eigensolver (VQE) for calculating the ground-state energy of a simple molecule.
- Develop a classical molecular dynamics simulation using quantum-derived energies to model molecular vibrations.

Your program will be constituted of two parts: a quantum and a classical part.

## 2 Quantum Part

Execution of this part requires installation of `qiskit-nature` package. You can do so by running `pip install qiskit-nature`.

In the quantum part, you will perform the calculation of the ground state energy for a molecule of Lithium Hidride, $LiH$, and of Oxygen, $O_2$. Here is how you can obtain different values that are needed for the calculation:

```
from qiskit_nature.drivers.second_quantization import PySCFDriver
from qiskit_nature.second_q.formats.molecule_info import MoleculeInfo
```

```python
from qiskit_nature.second_q.mappers import ParityMapper

molecule = MoleculeInfo(
        # Coordinates in Angstrom
        symbols=["Li", "H"],
        coords=([0.0, 0.0, 0.0], [dist, 0.0, 0.0]),
        multiplicity=1,  # = 2*spin + 1
        charge=0,
    )

driver = PySCFDriver.from_molecule(molecule)

# Get properties
properties = driver.run()
# Now you can get the reduced electronic structure problem
problem = FreezeCoreTransformer(
    freeze_core=True, remove_orbitals=[-3, -2]
).transform(properties)

num_particles = problem.num_particles
num_spatial_orbitals = problem.num_spatial_orbitals

mapper = ParityMapper(num_particles=num_particles)
qubit_op = mapper.map(problem.second_q_ops()[0])
```

**Exercise 1:** (60 Pt.)

you must calculate the ground state using two algorithms and compare the results. The algorithms are `VQE` and `AdaptVQE`, both available in the package `qiskit_algorithms.minimum_eigensolvers`. You should compare the results of the two algorithms with a classical version, for reference.

For each algorithm, you have to try different configurations, in terms of (1) initial state, (2) optimizer, and (3) ansatz. More precisely:

1. You have to compare three types of initializations: random initialization, all-zero initialization, and Hartree-Fock initialization. For the first two, we mean that you should modify accordingly the `initial_point` parameter of VQE. For the Hartree-Fock initialization, you are supposed to create a Hartree-Fock state using the `HartreeFock` class. Please note that the initialization using Hartree Fock state will work only with UCC ansatz. (10 Pt.)

2. You are supposed to compare the `UCC` ansatz, provided in the class `UCCSD` in Qiskit Nature, with at least one hardware efficient ansatz, such as `EfficientSU2`, `RealAmplitudes`, `PauliTwoDesign`, and `ExcitationPreserving`. All these circuits are available in the package `qiskit.circuit.library`. You can choose the configurations that you prefer. (20-35 Pt.)

3. You will compare now the use of at least two different optimizers for the VQAs. We suggest the use of `COBYLA`, `SPSA`, and `L_BFGS_B`, but feel free to use any other available in `qiskit_algorithms.optimizers`. (10-15 Pt.)

4. We expect you to compare the different configurations considering: (20 Pt.)

    - Running time, intended as wall clock time from the initialization of the molecule until the calculation of the minimum eigenvalue (Only for the quantum configurations)

- Convergence per iteration, namely, showing a plot where you visualize the energy with respect to iteration (Only for the quantum configurations);
- Error in comparison with the classical implementation.

You should test each configuration using at least 500 shots and 500 iterations for the optimizers. Concerning execution, we expect you to use a classical simulator.

**Exercise 2:**

40 Now, you have to integrate the calculation that you performed in the quantum within a classical simulation. You are supposed to use the **Verlet integration method**, which allows you to simulate the trajectory of a molecule in the space using the following formula:

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{1}$$

Where $x(t)$ is the position of the molecule at time step $t$, $v(t)$ the velocity, and $a(t)$ the acceleration, given by $\frac{F}{m}$, where $m$ is the mass of the nucleus and $F$ is the gradient of the energy. The exact formula for it is

$$F = -\frac{\partial E}{\partial x} \tag{2}$$

where $E(x)$ is the energy for the atom at the position $x$, calculated in the quantum part, and $x$ is the position of the molecule. However, it can be approximated by

$$F \approx -\frac{E(x + \delta) + E(x - \delta)}{2\delta}. \tag{3}$$

Regarding updates of the velocity, we can compute it as

$$v(t + \Delta t) = v(t) + \frac{1}{2}(a(t) + a(t + \Delta t))\Delta t \tag{4}$$

1. Use the energy, calculated with a configuration of VQE/AdaptVQE of your choice, to update the position of the molecule in the space. Please note that the trajectories are 3D. For simplification, you can make your own assumption for missing values (i.e., mass, velocity and so on): the point is not to provide an accurate molecular dynamics simulation, but more to give an example of a hybrid quantum-classical application. Your implementation should be able to simulate at least 10 steps; (20 Pt.)

2. Compare the results of the hybrid quantum-classical simulation with a pure classical simulation, i.e., where you use only classical functions to calculate the energy; (20 Pt.)

3. **Bonus**: You should also visualize the position of the molecules in the space using matplotlib. (10Pt.)

# 3 Hints

In the following, we show a possible pseudocode for the application:

Also, note that quantum execution is the bottleneck in this case, so I suggest that you do not run the quantum part from within the simulation loop. A possible solution would be to precalculate the values beforehand, and use them in the loop instead of running the quantum part from within it.

**Algorithm 1** 3D Molecular Dynamics Simulation with Verlet Integration

**Require:**

Mass $m$
Time step $\Delta t$
Total steps $N_{\text{steps}}$
Initial position $\mathbf{r} = [x, y, z]$
Initial velocity $\mathbf{v} = [v_x, v_y, v_z]$
Force function `force_function`$(\mathbf{r})$

**Ensure:** Trajectory `trajectory` containing positions over time

$\mathbf{a} \leftarrow \frac{\texttt{force_function}(\mathbf{r})}{m}$ ▷ Initial acceleration
`trajectory` $\leftarrow [\mathbf{r}]$ ▷ Store initial position
**for** step $\leftarrow 1$ to $N_{\text{steps}}$ **do**
    $\mathbf{r}_{\text{new}} \leftarrow \mathbf{r} + \mathbf{v} \cdot \Delta t + 0.5 \cdot \mathbf{a} \cdot (\Delta t)^2$ ▷ Update position
    $\mathbf{f}_{\text{new}} \leftarrow$ `force_function`$(\mathbf{r}_{\text{new}})$ ▷ Compute new force
    $\mathbf{a}_{\text{new}} \leftarrow \frac{\mathbf{f}_{\text{new}}}{m}$ ▷ Compute new acceleration
    $\mathbf{v} \leftarrow \mathbf{v} + 0.5 \cdot (\mathbf{a} + \mathbf{a}_{\text{new}}) \cdot \Delta t$ ▷ Update velocity
    $\mathbf{r} \leftarrow \mathbf{r}_{\text{new}}$ ▷ Update position for next step
    $\mathbf{a} \leftarrow \mathbf{a}_{\text{new}}$ ▷ Update acceleration for next step
    `trajectory.append`$(\mathbf{r})$ ▷ Store position
**end for**
**return** `trajectory`

# 4 Deliverables

You have to submit:

- A report in PDF, where you explain your solution and comment the results;
- The **commented** source code of the application;
- The requirements.txt file with all the libraries needed to execute the code;
- A README file with all the instructions to run the program.