# Qiskit Nature

Qiskit Nature is an open-source framework that supports problems including ground state energy computations, excited states and dipole moments of molecule, both open and closed-shell.

The code comprises chemistry drivers, which when provided with a molecular configuration will return one and two-body integrals as well as other data that is efficiently computed classically. This output data from a driver can then be used as input in Qiskit Nature that contains logic which is able to translate this into a form that is suitable for quantum algorithms. The conversion first creates a FermionicOperator which must then be mapped, e.g. by a Jordan Wigner mapping, to a qubit operator in readiness for the quantum computation.

In [ ]:
```
!pip install qiskit-nature
```

## Optional Installs

To run chemistry experiments using Qiskit Nature, it is recommended that you install a classical computation chemistry software program/library interfaced by Qiskit. Several, as listed below, are supported, and while logic to interface these programs is supplied by Qiskit Nature via the above pip installation, the dependent programs/libraries themselves need to be installed separately.

1. Gaussian 16™, a commercial chemistry program
2. PSI4, a chemistry program that exposes a Python interface allowing for accessing internal objects
3. PyQuante, a pure cross-platform open-source Python chemistry program
4. PySCF, an open-source Python chemistry program

In [ ]:
```
!pip install pyscf
```

# Creating Your First Chemistry Programming Experiment in Qiskit

Now that Qiskit Nature is installed, let's try a chemistry application experiment using the VQE (Variational Quantum Eigensolver) algorithm to compute the ground-state (minimum) energy of a molecule.

In [4]:

```
from qiskit_nature.settings import settings
from qiskit_nature.drivers import UnitsType
from qiskit_nature.drivers.second_quantization import PySCFDriver
from qiskit_nature.problems.second_quantization.electronic import ElectronicS

settings.dict_aux_operators = True

# Use PySCF, a classical computational chemistry software
# package, to compute the one-body and two-body integrals in
# electronic-orbital basis, necessary to form the Fermionic operator
driver = PySCFDriver(atom='H .0 .0 .0; H .0 .0 0.735',
                     unit=UnitsType.ANGSTROM,
                     basis='sto3g')
problem = ElectronicStructureProblem(driver)

# generate the second-quantized operators
second_q_ops = problem.second_q_ops()
main_op = second_q_ops['ElectronicEnergy']

particle_number = problem.grouped_property_transformed.get_property("Particle

num_particles = (particle_number.num_alpha, particle_number.num_beta)
num_spin_orbitals = particle_number.num_spin_orbitals

# setup the classical optimizer for VQE
from qiskit.algorithms.optimizers import L_BFGS_B

optimizer = L_BFGS_B()

# setup the mapper and qubit converter
from qiskit_nature.mappers.second_quantization import ParityMapper
from qiskit_nature.converters.second_quantization import QubitConverter

mapper = ParityMapper()
converter = QubitConverter(mapper=mapper, two_qubit_reduction=True)

# map to qubit operators
qubit_op = converter.convert(main_op, num_particles=num_particles)

# setup the initial state for the ansatz
from qiskit_nature.circuit.library import HartreeFock
```

```
init_state = HartreeFock(num_spin_orbitals, num_particles, converter)

# setup the ansatz for VQE
from qiskit.circuit.library import TwoLocal

ansatz = TwoLocal(num_spin_orbitals, ['ry', 'rz'], 'cz')

# add the initial state
ansatz.compose(init_state, front=True, inplace=True)

# set the backend for the quantum computation
from qiskit import Aer

backend = Aer.get_backend('aer_simulator_statevector')

# setup and run VQE
from qiskit.algorithms import VQE

algorithm = VQE(ansatz,
                optimizer=optimizer,
                quantum_instance=backend)

result = algorithm.compute_minimum_eigenvalue(qubit_op)
print(result.eigenvalue.real)

electronic_structure_result = problem.interpret(result)
print(electronic_structure_result)
```

```
-1.8572750301557372
=== GROUND STATE ENERGY ===

* Electronic ground state energy (Hartree): -1.857275030156
  - computed part:      -1.857275030156
~ Nuclear repulsion energy (Hartree): 0.719968994449
> Total ground state energy (Hartree): -1.137306035707

=== MEASURED OBSERVABLES ===


=== DIPOLE MOMENTS ===

~ Nuclear dipole moment (a.u.): [0.0  0.0  1.3889487]
```

The program above uses a quantum computer to calculate the ground state energy of molecular Hydrogen, H2, where the two atoms are configured to be at a distance of 0.735 angstroms. The molecular input specification is processed by the PySCF driver. This driver is wrapped by the ElectronicStructureProblem. This problem instance generates a list of second-quantized operators which we can map to qubit operators with a QubitConverter. Here, we chose the parity mapping in combination with a 2-qubit reduction, which is a precision-preserving optimization removing two qubits; a reduction in complexity that is particularly advantageous for NISQ computers.

The qubit operator is then passed as an input to the Variational Quantum Eigensolver (VQE) algorithm, instantiated with a classical optimizer and a RyRz ansatz (TwoLocal). A Hartree-Fock initial state is used as a starting point for the ansatz.

The VQE algorithm is then run, in this case on the Qiskit Aer statevector simulator backend. Here we pass a backend but it can be wrapped into a QuantumInstance, and that passed to the run instead. The QuantumInstance API allows you to customize run-time properties of the backend, such as the number of shots, the maximum number of credits to use, settings for the simulator, initial layout of qubits in the mapping and the Terra PassManager that will handle the compilation of the circuits. By passing in a backend as is done above it is internally wrapped into a QuantumInstance and is a convenience when default setting suffice.

In the end, you are given a result object by the VQE which you can analyze further by interpreting it with your problem instance.

# Further examples

Learning path notebooks may be found in the Nature Tutorials section of the documentation and are a great place to start

Jupyter notebooks containing further Nature examples may be found in the following Qiskit GitHub repositories at qiskit-nature/docs/tutorials.