Objects

hamiltonian
source
The coefficients are stored as a NumPy array; the datatype is complex128
The Pauli terms are stored in a PauliList object, which is a Qiskit class. This class is made up of NumPy arrays and Python lists. This class represents quantum operations using classical data structures.

Object Type = Classical

ansatz
source
A classical representation of a quantum circuit. It represents a circuit structure and does not use quantum software for execution (at this point). Once execution is performed, quantum software/hardware will interact with it.

Object Type = Classical

backend
source
In Qiskit, selecting a backend involves classical operations that retrieve data about backend availability and queue status, enabling the choice of the least busy quantum hardware. The .target method is utilised to pass backend constraints to a pass manager. The Target object itself is a data structure residing in classical memory. It uses classical data types like dictionaries, lists, and custom objects defined within Qiskit to represent the backend's capabilities and limitations.

The backend is also used to configure the session object, which configures the estimator object. The backend, although classical, will facilitate communication with the selected quantum hardware and, therefore, should be assigned the <<Quantum Driver>> stereotype.

Object Type = <<Quantum Driver>>

pm
source
Used to transform the ansatz circuit to be backend compatible. The information held to make this transformation remains classical. The transformation process relies solely on classical information processing and does not require interaction with the quantum hardware during the pm.run() execution. The pass manager acts as a classical pre-processing step, preparing the circuit for execution on the quantum device.

Object Type = Classical

ansatz_isa

This is the transformed ansatz circuit.

Object Type = Classical

hamiltonian_isa
The method .apply_layout() is applied to the Hamiltonian, using the newly created ansatz_isa circuit layout as a guide. This all remains classical.

Object Type = Classical

x0
A NumPy array containing the original guess of parameters to use, where the search process will be to find the parameters in a search space that result in the lowest energy estimate of the Hamiltonian.

Object Type = Classical

cost_func

This is a method that contains and uses the EstimatorV2 (estimator) instance. The estimator.run() method submits circuits, observables, and parameters to the Estimator primitive ([source]()). It returns a [RuntimeJobV2,]() which tracks and manages the status of the request submitted—in this case, an estimation job. Calling .result() on the RuntimeJobV2 object returns the job results; these are extracted and added to a dictionary to refer to later.

The backend will manage the communication between the quantum and classical software. The cost_func method is a <<Quantum Request>>, making a call to the backend via estimator.run() to obtain an estimation of the circuit's energy.

Object Type = <<Quantum Request>>

estimator
[source]()

This is a quantum algorithm that estimates the expectation values of quantum circuits and observables. It should be assigned the <<Quantum>> stereotype.

Object Type = <<Quantum>>

session
[source]()

The session object provides a context for grouping and potentially prioritising the execution of quantum jobs on a specific backend. It is configured with information from the backend, which describes the capabilities and status of the associated quantum hardware. The backend, a classical entity, manages the translation of classical instructions to quantum operations and the communication of results back to the classical realm. Therefore, the session will receive information regarding execution management from the backend in a classical format.

Object Type = Classical

res
source
The SciPy Minimize method is a classical method for finding the minimum result, which will be the parameters that produce the lowest energy estimate. This outer loop takes the cost_func into its inner loop. It will configure its result on the classically translated result of the cost_func method, so it is a classical entity.

Object Type = Classical

cost_history_dict

A dictionary that stores the classically translated results from the cost_func iterations.

Object Type = Classical

fig
A matplotlib object visualising the results

Object Type = Classical

Messages

hamiltonian -> ansatz: .num_qubits **Classical**
backend -> pm: pass constraints and optimisation level **Classical**
ansatz -> pm: transform ansatz backend compatible **Classical**
pm -> ansatz_isa: pm.run(ansatz) **Classical**
ansatz_isa -> hamiltonian: transform hamiltonian backend compatible **Classical**
hamiltonian -> hamiltonian_isa: hamiltonian.apply_layout(layout=ansatz_isa.layout) **Classical**
x0 -> minimize: pass current params guess **Classical**
ansatz_isa -> minimize: pass ansatz circuit **Classical**
hamiltonian_isa -> minimize: pass hamiltonian **Classical**
backend -> session: create backend-compatible resource management session **Classical**
session -> estimator: configure estimator **Classical**
estimator -> minimize: pass estimator **Classical**
minimize -> cost_func: call cost_func() **Classical**
cost_func -> cost_func: create variable pub(ansatz, [hamiltonian], [params]) **Classical**
estimator <- cost_func: run estimation on pub **<<Quantum Request>>**
estimator --> cost_func: extract energy estimate **Classical**
cost_func -> cost_history_dict: record & print results (params, energy estimate, loop iteration) **Classical**

minimize -> x0: update params **Classical**
cost_func -> cost_func: return energy estimate **Classical**
cost_history_dict <- minimize: verify params and interations **Classical**
cost_history_dict --> minimize: match = True **Classical**
cost_history_dict --> minimize: match = False **Classical**
cost_history_dict -> figure: plot iterations and energy estimates **Classical**