

Credit Risk Analysis

Overview Of Quantum In Credit Risk Analysis:

Over the coming decades, quantum computing will not only revolutionize the way that humanity approaches many of its biggest problems but will also help to make businesses around the world more efficient. One of the clearest examples of this will be the benefits that financial institutions will obtain in the realm of credit risk analysis. In this report, we will dive into what credit risk analysis is, what value it provides, the current approaches/methods used, and the potential advantages quantum computing could provide.

Factors Considered in Credit Risk Assessments:

In its essence, credit risk analysis is the assessment of the likelihood that a debtor will default on a loan repayment from a creditor. Payment default is the main source of risk associated with granting debt.

For credit to individual borrowers and small businesses, financial institutions use slightly different credit risk evaluation metrics than those for large corporations.

Individuals and small businesses are evaluated based on what are often referred to as “the 5C’s of credit risk”:

- **Capacity**
 - A borrower’s ability to follow their repayment plan based on revenues, expenses, and cash flows.
- **Capital**
 - The amount of a borrower’s own capital that they were willing to risk by investing it into their idea/business.
- **Collateral:**
 - Alternative source of repayment that is given to a creditor to reduce their losses in case the borrower defaults, typically in the form of a major asset that is relatively liquid (ie, real estate).
- **Conditions**
 - Economic conditions, the competitive landscape within the borrower’s industry, among other factors which vary over time. A favorable set of circumstances decreases the likelihood of default.

- **Character**

- One's personal reliability as a debtor, including education, business experience, personal credit history, and more. Positive reputation within their industry gives creditors confidence.

Financial institutions evaluate large corporations' credit risk with a slightly different viewpoint, based on these factors:

- **Analyze financial position**

- Complete assessment of current and projected financial statements to gauge the borrower's repayment capacity. Company projections must be in line with the financial institution's market and economic expectations.

- **Sensitivity Analysis**

- Test the safety of a company's financial projections by considering potential adverse events in the economy or their industry. This could include a recession, a shortage in a supply chain, or a union forming.

- **Market Events**

- Consider the likelihood of a significant macroeconomic downturn or decline of specific sectors during the duration of a loan. For example, political risk is that of regulations/laws coming into place that could affect business operations.

- **Analyze Business Model/Strategy**

- Study the borrower's business plan, assessing the company's knowledge, experience, and capacity to operate successfully in the future. For example, if they over reliance on key clients, suppliers, or staff, as this makes any departure or relational conflict much more consequential to operational success.

- **Assess guarantees/Collateral**

- Typically includes a business's inventory, property, or some other major asset that is relatively liquid. A loan often involves guarantees of partial or full repayment from a third party if there is a default. In these cases, institutions' assessment of the guarantor is typically proportionate to the size of guarantee in relation to the loan.

All of these factors are taken into account by financial institutions when they configure how likely an applicant is to default on their loan using a classical simulation.

Classical Approaches to Quantifying Credit Risk:

Value at Risk (VaR) is an estimate of the maximum loss within a specified confidence level considering the statistical distribution of several credit risk factors (those listed above) and potential adverse movements in the relevance of those factors.

The following steps are used to calculate the “Value at Risk (VaR)” and cash flows generated from a loan using the classical **Monte Carlo simulation** technique:

- 1) Define loan characteristics, such as the loan principle, interest rate, term until maturity, among other loan-specific details.
- 2) Determine key credit risk factors, such as macroeconomic or industry-specific outlook, default probability indicators, among other information discussed in the “Factors Considered in Credit Risk Assessments” section.
- 3) Establish simulation framework to generate multiple scenarios for each of the credit risk factors over a desired time frame. These scenarios are determined by specifying the statistical models for each credit risk factor based on expert judgment, referencing historical data.
- 4) Simulate the loan’s performance under the different scenarios representing the occurrence of each risk factor, modelling the loan’s general performance under the various scenarios.
- 5) Calculate cash flows from the loan, such as principal and interest payments, under each simulated scenario.
- 6) Account for the likelihood of each scenario’s occurrence by assigning probability weights to each of them. This is also based on historical data and expert judgement considering both market factors and borrower-specific factors.
- 7) Summarize the simulated cash flows for all of the possible scenarios for each future time period, thus creating a distribution of potential outcomes for the loan’s cash flows.
- 8) Calculate Value-at-Risk. First, determine the confidence level you would like to use, then sort the simulated cash flows in ascending order and identify the value where the cumulative probability exceeds this confidence threshold – this represents the estimated VaR for the loan.

Limitations and Quantum Advantage:

The Monte Carlo method of analyzing an applicant’s credit risk is fairly comprehensive and effective at predicting default risk, but it does have some limitations.

Firstly, the time and computational requirements involved in running a large number of random trials limits the speed of overall analysis for a given loan. For financial institutions performing a high frequency of credit risk assessments, this can lead to a significant time cost. Using their superior technological abilities, quantum computers can help financial institutions speed up the process by running simulations faster. This ends up saving them money when run at scale and allows users to become more adaptive to market changes.

Secondly, Monte Carlo simulations typically assume independence among the variables being modelled, so many of the calculations do not account for how the variables are related, thus impeding the results by varying degrees. Quantum algorithms can create entangled systems that can more accurately account for these connections between variables, essentially eliminating this problem.

The **Parametric method** is another popular technique used by financial institutions to comprehensively evaluate an entity's credit risk considering all the factors listed in the section labelled "Factors Considered in Credit Risk Assessments". It quantifies the likelihood of default and estimates the potential losses associated with a given borrower or loan portfolio using mathematical and statistical techniques. The step-by-step approach is as follows:

- 1) Estimate Probability of Default (PD). This represents the likelihood that a borrower will default on their loan within a specified time frame. It is estimated by experts based on borrower characteristics, historical data, and more important information.
- 2) Estimate Loss Given Default (LGD). This percentage value represents the portion of a loan that is expected to be unrecovered in the case that a borrower defaults. It is generally estimated by experts based on historical recovery rates for similar loans in conjunction with predictive statistical models.
- 3) Estimate Exposure at Default (EAD). This term refers to the outstanding loan balance or credit limit granted to the borrower at the time of default – how much exposure a lender has to the defaulting borrower. EAD is used in calculating the potential loss in case of default.
- 4) Calculate Expected Loss (EL). This value is the product of the first three values (PD, LGD, and EAD), and it represents the anticipated loss on a specific loan or loan portfolio over a given time frame. To calculate EL, one multiplies $PD \times LGD \times EAD$ together. This provides an average credit risk associated with the loan or portfolio at hand.

Limitations and Quantum Advantage:

Although the parametric method of evaluating credit risk is one of the best options available to financial institutions today, it does have some limitations. One of its biggest weaknesses is its assumption that credit

risk factors follow a normal distribution, when many risk events (ie, a default) often exhibit fat tail curves, as well as skewness or kurtosis. These are not captured effectively by normal distributions, so parametric models may underestimate the probability of extreme events, in turn leading to inaccurate risk estimates. Quantum computers' superior ability to handle probability distributions could create more accurate distribution models which capture fat-tailed and skewed characteristics. Given that these characteristics are very often observed in credit risk events, quantum algorithms would likely provide a more realistic representation of many events, enhancing the accuracy of risk estimations. Another weakness of this method is that two of the values estimated when using the parametric method, PD and LGD, require a comprehensive data analysis. This would include consideration of historical events, similar past loans, borrower characteristics, and other factors discussed in the "Factors Considered in Credit Risk Assessment" section. Comprehensive and reliable data can be difficult to attain, especially for lesser-known borrowers that are in niche industries. Insufficient or unreliable data can lead to skewed risk estimates, and in turn inaccurate predictions. With their increased computational power over classical computers, quantum computers could more efficiently process massive amounts of data regarding risk factors and historical information. This quantum advantage could help to significantly reduce the limitations on data availability by improving both the quantity and quality of data used in assessing a borrower's level of risk on a given loan.

Quantum Amplitude Estimation:

In an industry dealing in vast amounts of capital, any marginal improvement in predictive modelling is significant. Quantum algorithms provide an opportunity for this industry to enhance their analysis of credit risk.

Quantum Amplitude Estimation (QAE) is a quantum algorithm used to represent the amplitudes of certain quantum states. The amplitude of a quantum state is a complex number representation of its probability. QAE is often used in Quantum Machine Learning and optimization algorithms, which financial institutions can leverage for a technological advantage. The use of QAE in credit risk analysis algorithms can provide a quadratic speedup over classical methods, and in turn vastly increase the accuracy of analysis. Estimating the probability of default or of certain events occurring can be represented as amplitudes of quantum states. By implementing quantum superposition, QAE algorithms can provide more accurate results.

In the following report, the necessary Qiskit concepts and classes associated with different implementations of QAE algorithms will be thoroughly analyzed and compared.

Operators

Overview

Operators in Quantum Mechanics refer to a mathematical representation (Often via a matrix) of some observable that performs an action on a quantum state. They describe how a physical quantity, such as position or energy, is measured over time. Operators can have a variety of effects on a state such as measurement, transformation, or superposition and entanglement. In QAE the most important operator is the Grover operator which will be discussed in detail later on.

Instantiation

There are many different implementations of quantum operators, a basic example are Paul matrices used to describe fundamental observables.

A general method for constructing custom unitary operators involves:

1. Defining the transformation you want to apply
2. Identifying the dimensionality of the quantum system where the operator will act
3. Setting orthogonal states to span the vector space of the system (Common ones are $|0\rangle$ and $|1\rangle$)
4. Create a Unitary matrix to represent the transformation

[img 1]

```
46 # Define the custom transformation
47 transformation = np.array([[1, 1j], [1j, 1]]) / np.sqrt(2)
48
49 # Check if the transformation matrix is unitary
50 is_unitary = np.allclose(np.conj(transformation.T) @ transformation, np.identity(2))
51
52 # Print the transformation matrix and unitarity check
53 print("Custom Transformation Matrix:")
54 print(transformation)
55 print("\nIs Unitary:", is_unitary)
56
```

Grover Operator

Overview

The Grover Operator plays a central role in the process of amplifying the amplitudes of the desired state and increasing the probability of measuring them. It utilizes Grover's search algorithm which involves a few key steps:

- Initialize equal superposition of all possible states
- Apply the Oracle Operator
- Apply a Diffuser
- Repeat the operator applications for root (total number of states/number of marked states)
- Finally perform a measurement on the quantum register with a higher probability of it providing the solution to the search problem.

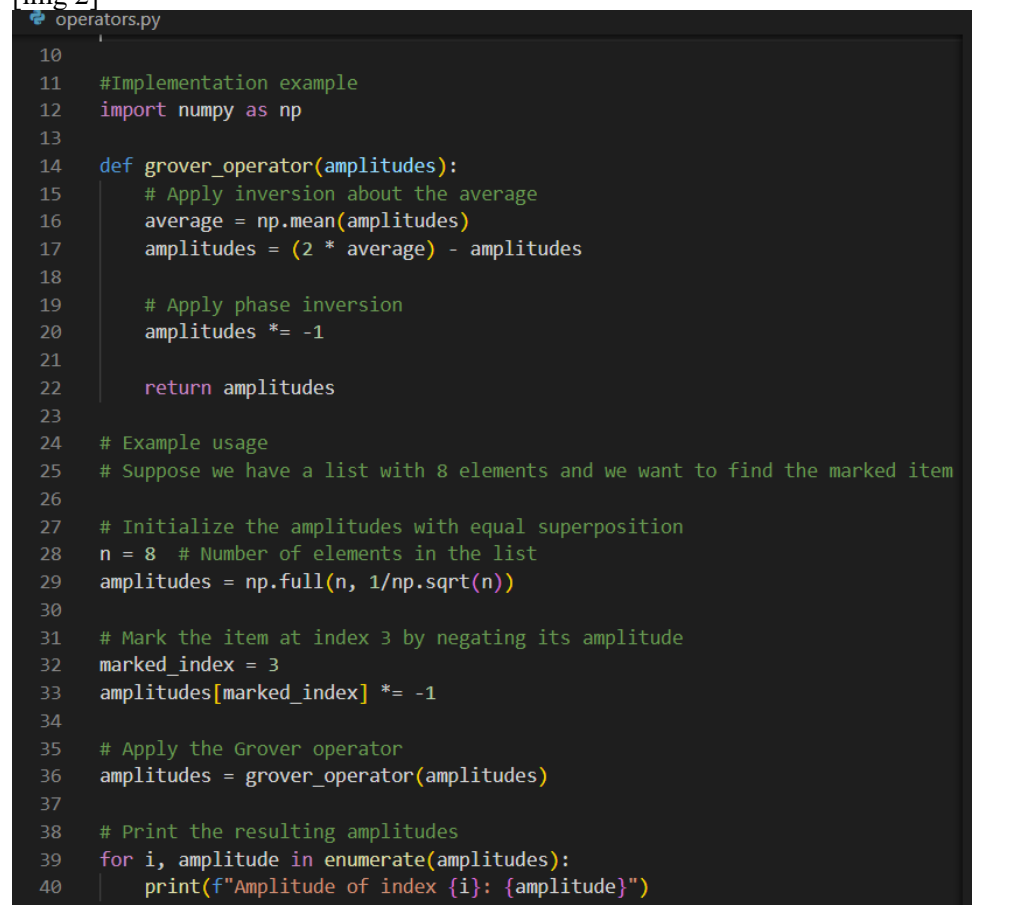
The Oracle and Diffuser operators mentioned above are the two components that make up the Grover Operator. The Oracle operator encodes the solution being searched and marks the desired states in the superposition of states (Like evaluating true for desired state and false for the rest). The diffuser

operator transforms the amplitudes of the states around the mean amplitude. It reflects the amplitudes of the states across the mean amplitude, this results in the amplitude of the desired states being magnified.

In relation to QAE, the Grover Operator is used in QAE to amplify the amplitude of a target state and suppress the amplitudes of other states in superposition.

Instantiation

[img 2]



```
10
11 #Implementation example
12 import numpy as np
13
14 def grover_operator(amplitudes):
15     # Apply inversion about the average
16     average = np.mean(amplitudes)
17     amplitudes = (2 * average) - amplitudes
18
19     # Apply phase inversion
20     amplitudes *= -1
21
22     return amplitudes
23
24 # Example usage
25 # Suppose we have a list with 8 elements and we want to find the marked item
26
27 # Initialize the amplitudes with equal superposition
28 n = 8 # Number of elements in the list
29 amplitudes = np.full(n, 1/np.sqrt(n))
30
31 # Mark the item at index 3 by negating its amplitude
32 marked_index = 3
33 amplitudes[marked_index] *= -1
34
35 # Apply the Grover operator
36 amplitudes = grover_operator(amplitudes)
37
38 # Print the resulting amplitudes
39 for i, amplitude in enumerate(amplitudes):
40     print(f"Amplitude of index {i}: {amplitude}")
41
```

Amplitude Estimation Workflow

Overview

Qiskit's Quantum Amplitude Estimation (QAE) algorithms all derive from the `AmplitudeEstimator` interface, containing the `estimate` method. This method, regardless of which implementation is used, requires an `EstimationProblem` object and will return an `AmplitudeEstimatorResult` object. This consistency allows for different methods of QAE to follow a consistent workflow.

EstimationProblem

```
CLASS EstimationProblem (state_preparation, objective_qubits, grover_operator=None,  
                          post_processing=None, is_good_state=None) ¶
```

[SOURCE]

Overview

This class contains the specific information needed to run a QAE algorithm.

Parameters

- `state_preparation`: QuantumCircuit
 - This parameter prepares and represents the input state, which is also called ‘A’.
- `objective_qubits`: int| list[int]
 - This parameter represents the qubit index, or list of qubit indices to be measured.
 - The ‘is_good_state’ function will be applied to the selected qubit(s).
- `grover_operator`: QuantumCircuit | None
 - This parameter represents the Grover operator, which is also called ‘Q’.
- `post_processing`: Callable[[float], float] | None
 - This parameter represents the mapping applied to the result of the algorithm.
 - The default value is the identity matrix.
- `is_good_state`: Callable[[str], bool] | None
 - This parameter represents the function called on selected qubits, to check whether the qubit string represents a good state.
 - The default value is $|1\rangle$

Instantiation

[img 3]

```
problem = EstimationProblem(  
    state_preparation=A, # A operator  
    grover_operator=Q, # Q operator  
    objective_qubits=[0], # the "good" state Psi1 is identified as measuring |1> in qubit 0  
)
```


The instantiation of the EstimationProblem object ‘problem’ detailed in [img 3] provides the first, second, and third parameters and allows for the remaining two to be set by the default values.

Methods

- rescale (scaling_factor: float)
 - This method accepts a single parameter representing a scaling factor.
 - It rescales the good state amplitude with this value.

Implementations

[img 4]

```
#Implementation for Canonical Amplitude Estimation
ae_result = ae.estimate(problem)

#Implementation for Iterative Amplitude Estimation
iae_result = iae.estimate(problem)

#Implementation for Maximum Likelihood Amplitude Estimation
mlae_result = mlae.estimate(problem)

#Implementation for Faster Amplitude Estimation
fae_result = fae.estimate(problem)
```

As explained prior, all Quantum Amplitude Estimation algorithms derive from the AmplitudeEstimator interface. This allows for a straightforward and consistent workflow of applications requiring QAE as shown in [img 4].

Sampler

Overview

The Sampler class is an abstract base class that serves as an interface used to execute circuits and obtain probabilities. It provides a unified way to interact with a quantum device on the backend. This class is commonly instantiated directly by using concrete implementations such as the AerSimulator class.

Implementations

Creating an instance of AerSimluator automatically implements the Sampler interface, allowing the object to be used as a sampler for executing quantum circuits.

The first method of implementation is as shown in [img 5]

```
from qiskit import Aer

# Get a simulator backend
backend = Aer.get_backend('qasm_simulator')
```

[img 5]

The three types of simulators are as follows:

- ‘qasm_simulator’
 - This simulator emulates the execution of quantum circuits and returns measurement outcomes as bit strings.
- ‘statevector_simluator’
 - This simulator returns the final quantum state vector.
 - Is used for simulations without measurements.
- ‘unitary_simulator’
 - This simulator returns the unitary matrix representation of the quantum circuit.

A second method of implementation is as shown in [img 6]

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator

# Get a simulator backend
backend = AerSimulator()
```


[img 6]

When accessing the AerSimulator backend from the qiskit_aer module, the program automatically returns a ‘qasm_simulator” backend. To override this you can add a parameter as in [img 7].

```
backend= AerSimulator(method='statevector')
```

[img 7]

AmplitudeEstimation

```
CLASS AmplitudeEstimation(num_eval_qubits, phase_estimation_circuit=None, iqft=None,  
                           quantum_instance=None, sampler=None) 
```

[\[SOURCE\]](#)

Overview

This class implements the original Quantum Amplitude Estimation (QAE) algorithm. This canonical version uses Quantum Phase Estimation (QPE) and Grover's Algorithm. The estimation error bound of QAE scales as $O(1/M)$, where M denotes the number of samples, providing a quadratic speedup over current Monte Carlo simulations which scale at $O(1/\sqrt{M})$.

The QAE algorithm provides a general framework for amplitude estimation and is adaptable to various applications and problems. It can be used to estimate the probability that the output of another algorithm satisfies a particular target property.

Parameters

- `num_eval_qubits` : int
 - This parameter represents the number of evaluations qubits.
- `phase_estimation_circuit` : QuantumCircuit | None
 - This parameter represents the phase estimation circuit used to run the algorithm.
 - It is a default parameter that defaults to the standard phase estimation circuit from the circuit library.
- `iqft` : QuantumCircuit | None
 - This parameter represents the inverse quantum Fourier transform component.
 - It is a default parameter that defaults to a standard implementation.
- `sampler` : BaseSampler | None
 - This parameter represents a sampler primitive to evaluate the circuits.

Instantiation

[img 8]

```
ae = AmplitudeEstimation(
    num_eval_qubits=3, # the number of evaluation qubits specifies circuit width and accuracy
    sampler=sampler,
)
```

The instantiation of the AmplitudeEstimation object ‘ae’ detailed in [img 8] provides the first and last parameters and allows for the remaining two to be set by the default values.

Methods

- estimate (estimation_problem: EstimationProblem)
 - This method accepts a single parameter and returns an object of type AmplitudeEstimationResult.

```
ae_result = ae.estimate(problem)
```

[img 9]

The ‘estimate’ method call detailed in [img 9] acting on ‘ae’, returns an AmplitudeEstimationResult object stored as ‘ae_results’. It accepts an object of type EstimationProblem created in [--img number--]. The ‘ae_results’ variable stores important information about the amplitude estimation and can be used in the ‘compute_confidence_interval’ and ‘compute_mle methods’.

- compute_confidence_interval (result: AmplitudeEstimationResult, alpha: float, kind: str)
 - This method accepts three parameters.
 - These parameters are used to return the (1-alpha) confidence interval of the selected kind.

[img 10]

```
ae_confidence_interval = ae.compute_confidence_interval(ae_result,0.15,"fisher")
```

The ‘compute_confidence_interval’ method call detailed in [img 10] acting on ‘ae’ returns the (1-alpha) confidence interval stored in ‘ae_confidence_interval’.

- compute_mle (result: AmplitudeEstimationResult, apply_post_processing: bool)
 - This method accepts two parameters.
 - The second parameter checks whether to apply post processing before returning.
 - The method computes and returns the Maximum Likelihood Estimator (MLE) value of the AmplitudeEstimationResult object.

```
ae_mle = ae.compute_mle(ae_result,False)
```

[img 11]

The ‘compute_mle’ method call detailed in [img 11] acting on ‘ae’ returns the Maximum Likelihood Estimator (MLE) value of the AmplitudeEstimationResult object. This value is computed without restriction to a sampling grid, thus, is an improved and more accurate estimator.

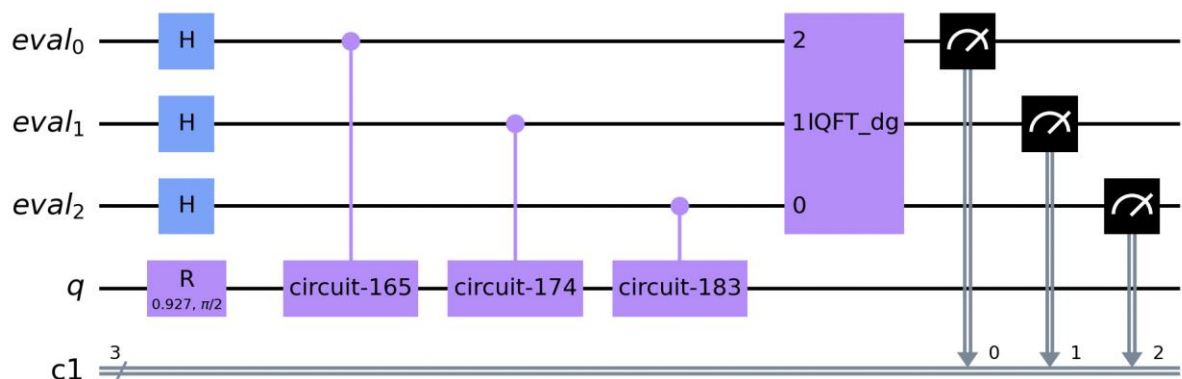
- construct_circuit (estimation_problem: EstimationProblem, measurement: bool)
 - This method accepts two parameters.
 - The second parameter checks whether to include measurements.
 - The method constructs and returns a QuantumCircuit object.

```
ae_circuit = ae.construct_circuit(problem, True)

ae_circuit.decompose().draw("mpl")
```

[img 12]

The ‘construct_circuit’ method call detailed in [img 12] acting on ‘ae’ returns a QuantumCircuit object stored in ‘ae_circuit’. The ‘ae_circuit’ variable stores important information about the current quantum circuit and provides methods which can access and modify this data. This variable can also be used to deconstruct and draw the circuit , providing a visual representation as shown in [img 13]



[img 13]

- evaluate_measurements (circuit_results: dict[str, int] | np.ndarray, threshold: float)
 - This method accepts two parameters.

- The `circuit_results` parameter can be a dict, state vector, or a quasi-probability dict
- The `threshold` parameters marks the threshold for discarding probabilities.

The ‘evaluate measurements’ method call returns a tuple containing dictionaries with grid points and their respective probabilities. This can be used to visualize and organize the probability of different measurements.

MaximumLikelihoodAmplitudeEstimation

```
CLASS MaximumLikelihoodAmplitudeEstimation(evaluation_schedule, minimizer=None,  
                                             quantum_instance=None, sampler=None) ⓘ
```

[\[SOURCE\]](#)

Overview

The canonical implementation of the QAE algorithm uses a combination of Grovers algorithm and QPE, which involves complex quantum circuits and controlled operations. This can make implementation challenging. An alternative approach is the Maximum Likelihood Quantum Amplitude Estimation algorithm, which uses much simpler circuits while achieving the same quadratic speedup over Monte Carlo Simulations.

Parameters

- `evaluation_schedule` : list[int] | int
 - If a list, the powers applied to the Grover operator.
- `minimizer` : MINIMIZER | None
 - A minimizer that is used to find the minimum of the function.
 - Minimizer takes a function a list of (float float) tuples (for bounds) as parameters. It will return a float of found minimum.
 - Defaults to a brute search.
- `sampler` : BaseSampler | None
 - This parameter represents a sampler primitive to evaluate the circuits.

Instantiation

```
miae = MaximumLikelihoodAmplitudeEstimation(
    evaluation_schedule=3, # log2 of the maximal Grover power
    sampler=sampler,
)
```

[img 14]

The instantiation of the MaximumLikelihoodAmplitudeEstimation detailed in [img 14] provides the first and last parameters and allows for the remaining parameter to be set by the default values.

Methods

- `compute_mle` (circuit_results: list[dict[str, int] | np.ndarray], estimation_problem: EstimationProblem, num_state_qubits: int | None), return_counts: bool)
 - This method accepts four parameters.
 - The `circuit_results` parameter represents a list of circuit outcomes, as counts or state vectors.
 - The `num_state_qubits` parameter represents the number of state qubits, required for state vector simulations and defaults to None.
 - The `return_counts` parameter when true, returns the good counts.

Maximum Likelihood Quantum Amplitude Estimation is calculated via grid-search, if sufficient grid points are provided.

The following method follows the same structure as the “`compute_confidence_interval`” method in `AmplitudeEstimation`, with an extra parameter used for potential post processing.

- `compute_confidence_interval` (result: AmplitudeEstimationResult, alpha: float, kind: str, apply_post_processing: bool)
 - this method accepts an additional parameter that when true, will apply post processing to the confidence interval.

```
miae_confidence_interval = miae.compute_confidence_interval(miae_result,0.15,"fisher",False)
```

The following two methods follow the same structure as the methods in `AmplitudeEstimation`

- `estimate` (estimation_problem: EstimationProblem)

```
miae_result = miae.estimate(problem)
```

- `construct_circuit` (estimation_problem: EstimationProblem, measurement: bool)
 - This method accepts two parameters.

```
miae_circuit = miae.construct_circuits(problem)
```

IterativeAmplitudeEstimation

Overview

In the search for QAE variants that don't require dependence on QPE, the Iterative Quantum Amplitude Estimation, or for short IQAE, provides an efficient solution to the classical MC simulations. Qiskit allows for relatively easy implementation of this algorithm by creating a class that can be imported from Qiskit's Algorithms module.

Parameters:

- `epsilon_target`: float
 - This parameter allows the user to pick a target precision which represents the desired accuracy for the estimation. It has values between 0 and 0.5.
- `alpha`: float
 - Represents the confidence level, where the target probability is $1 - \alpha$
 - User inputs value between 0 and 1
- `confint_method`: str
 - "Beta" is the default for the confidence intervals which uses the Clopper-Pearson intervals.
 - Other option is "chernoff", representing Chernoff intervals.
 - Chernoff isn't as common and is used for niche scenarios where the distribution decays exponentially, considering that the exponential tail bounds for the probability of deviation from the mean
- `min_ratio`: float = 2
 - For FindNextK helper algorithm where (K_{i+1} / K_i)
- `quantum_instance`: QuantumInstance | Backend | None = None

- Currently depending depreciation, could be replaced shortly
 - Quantum Instance or a Backend to be used, allowing for more flexibility
- sampler: BaseSampler | None = None
 - Mentioned above

Instantiation:

```
iae = IterativeAmplitudeEstimation(epsilon_target=0.5,alpha=0.25, sampler=sampler,)
```

****need to import IAE from qiskit.algorithms****

```
from qiskit.algorithms import IterativeAmplitudeEstimation
```

Methods:

- estimate
 - Method used to find the target amplitude for a given problem
 - Accepts problem from EstimationProblem class, returning the target amplitude
 - ****explained previously****

```
iae_result = iae.estimate(problem)
```
- construct_circuit
 - Common method ****explained previously****

```
iae_circuit = iae.construct_circuit(problem,k=3)
```
- _find_next_k
 - Accepts 4 parameters, returning a tuple of the next power k and a boolean flag for the interval

The importance of this method lies in ensuring that the next estimation iteration lies in whichever desired half of the circle is wanted ($[0, \pi]$ or $[\pi, 2\pi]$). This ensures accuracy and convergence of the algorithm. Additionally, the next k determines the next set of bounds for theta, which is critical in refining the estimation.

```
next_k = iae._find_next_k(3, upper_half_circle=True, theta_interval=(0.0, 1.0))
```

- _good_state_probability
 - Method accepts 3 parameters, returns tuple or float depending on parameters
 - Returns probability of '1' in the last qubit

This ‘good state’ probability is a very important metric, which can help users assess the success and quality of an estimate by capturing the meaningful information from the last qubit. This method available in the IAE class grants users ...

```
probability = iae._good_state_probability(problem, state_vec, num_state_qubits)
```

References:

- [1] “The 5 cs of credit,” Navy Federal Credit Union, <https://www.navyfederal.org/makingcents/business/the-5-cs-of-credit.html#:~:text=Lenders%20score%20your%20loan%20application,Capacity> (accessed Jun. 29, 2023).
- [2] M. B. Ansari, “Risk analysis using Monte Carlo Simulation,” Medium, <https://medium.com/analytics-vidhya/risk-analysis-using-monte-carlo-simulation-e35bf047fa20> (accessed Jun. 29, 2023).
- [3] N. Krastev, “Expected loss and its components,” 365 Financial Analyst, <https://365financialanalyst.com/knowledge-hub/credit-analysis/expected-loss-and-its-components-probability-of-default-loss-given-default-and-exposure-at-default/> (accessed Jun. 29, 2023).
- [4] D. Grinko, J. Gacon, C. Zoufal, and S. Woerner, “Iterative quantum amplitude estimation,” Nature News, <https://www.nature.com/articles/s41534-021-00379-1> (accessed Jun. 29, 2023).
- [5] A. Callison and D. E. Browne, “Adam Callison and Dan E. Browne 1department of physics and ... - arxiv.org,” Improved maximum-likelihood quantum amplitude estimation, <https://arxiv.org/pdf/2209.03321.pdf> (accessed Jun. 30, 2023).
- [6] “Quantum amplitude estimation¶,” Quantum Amplitude Estimation - Qiskit Finance 0.3.4 documentation, https://qiskit.org/ecosystem/finance/tutorials/00_amplitude_estimation.html (accessed Jun. 29, 2023).

