UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Estudio del Análisis de Riesgo de Crédito utilizando Ordenadores Cuánticos

Manuel Soto Jiménez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Estudio del Análisis de Riesgo de Crédito utilizando Ordenadores Cuánticos

Autor: Manuel Soto Jiménez
Tutor: Francisco Javier Gómez Arribas, Luis de Pedro Sánchez

mayo 2021

Manuel Soto Jiménez
Estudio del Análisis de Riesgo de Crédito utilizando Ordenadores Cuánticos

Manuel Soto Jiménez
C\ Francisco de Navacerrada nº 39

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos, por aguantar casi un año de cháchara, dudas y debates.

To read our E-mail, how mean
of the spies and their quantum machine;
be comforted though,
they do not yet know
how to factorize twelve or fifteen...

Volker Strassen

Resumen

La memoria explica el proceso seguido para implementar una librería de análisis de riesgos financieros de un portfolio para K activos utilizando algoritmos cuánticos. Esta librería, `CreditRiskAnalysis`, implementada en Python y que utiliza Qiskit (IBM), utiliza como punto de partida un tutorial de Qiskit que actualmente solo puede analizar portfolios de dos activos.

Para llegar al análisis y mejora del tutorial de referencia, se desarrolla una introducción a la computación cuántica y al análisis de riesgos financieros. En esta parte introductoria se explican algoritmos como la estimación de amplitud cuántica (QAE) o el modelo económico a simular, el modelo de Dependencia Condicional Gaussiana (GCI).

Finalmente, se ejecutan diversos análisis de portfolios para $K \geq 2$ activos variando el algoritmo de estimación de amplitud a usar así como su ejecución desde simuladores a ordenadores cuánticos reales. Para este último caso, se estudia la viabilidad actual de los mismos y se analiza el futuro trabajo a realizar.

Palabras clave

Computación cuántica, Análisis de Riesgo de Crédito, Método de Montecarlo, Estimación de amplitud cuántica (QAE), Estimación de amplitud cuántica iterativa (iQAE), Estimación de amplitud cuántica por Máxima Verosimilitud (MLAE), Modelo de Dependencia Condicional Gaussiana (GCI), Valor en Riesgo (VaR), Qiskit

Abstract

The report explains the process followed to implement a portfolio financial risk analysis library for K assets using quantum algorithms. This library, called `CreditRiskAnalysis`, implemented in Python and based on Qiskit (IBM), uses as a starting point a Qiskit tutorial that can currently only analyze two-asset portfolios.

To get to the study and improvement of the reference tutorial, an introduction to quantum computing and financial risk analysis is developed. In this introductory part, algorithms such as quantum amplitude estimation (QAE) or the economic model to be simulated, the Gaussian Conditional Independence model (GCI), are explained.

Finally, several portfolio analysis are executed for assets of $K \geq 2$ varying the amplitude estimation algorithm to be used as well as its execution from simulators to real quantum computers. For the latter case, the current viability of these is studied and future work to be done is analyzed.

Keywords

Quantum computing, Credit Risk Analysis, Montecarlo Method, Quantum Amplitude Estimation (QAE), Iterative Quantum Amplitude Estimation (iQAE), Maximum Likelihood Amplitude Estimation (MLAE), Gaussian Conditional Independence Model (GCI), Value at Risk (VaR), Qiskit

ÍNDICE

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción, motivación y objetivos

La computación cuántica es un tema de considerable actualidad, popular tanto por sus propiedades intrínsecas que la diferencian de la computación clásica como por el aclamado potencial que tiene en la teoría y que parece tener de cara a las próximas décadas. En cierto modo, comparando cómo la parte teórica de la computación se situaba "en cabeza" frente a la situación tecnológica, nos encontramos en una época computacional análoga a la década de 1930. La aleatoriedad, la paralelización de ejecuciones o el entrelazamiento de estados cuánticos proporciona unas capacidades computacionales que aportan un nuevo punto de vista de cara a resolver problemas de solución clásica inexistente o inviable.

En este capítulo veremos cuál es el estado del arte actual para entender la comparación anterior, así como que de entre todas las ramas de las posibles aplicaciones de la computación cuántica introduciremos la que será implementada y estudiada en este trabajo.

1.1. Introducción

Se define la supremacía cuántica como la situación tecnológica futura en la cual los ordenadores cuánticos, además de poder resolver problemas que un ordenador clásico no pueda resolver, podrán encontrar la solución en tiempos considerablemente menores [1]. Una supremacía que, debido a las características de su unidad básica computacional, el *qubit*, requiere aún de suficiente investigación tanto física como teórica como para ser alcanzada.

Llegar a manipular *qubits* en condiciones físicas menos exigentes que las actuales a la par que precisas y escalables, así como diseñar algoritmos cuánticos que permitan superar la actual barrera clásica; marcan objetivos en distintas disciplinas que irán resultando en aceleraciones algorítmicas. Aceleraciones en materia de tiempos de ejecución frente a las alternativas clásicas de tal calibre que supondrá en un futuro cuantiosas ventajas y avances tecnológicos.

El ejemplo más conocido de estos potenciales avances es el problema de descomposición en factores primos. La alta complejidad con respecto a órdenes de ejecución de la notación $O(f(N))$ de Landau del mejor algoritmo de factorización clásico garantiza la seguridad en nuestras comunicaciones

cifradas. Esta seguridad se perderá en teoría una vez el *hardware* cuántico pueda implementar el conocido como Algoritmo de Shor, capaz de descomponer números en factores primos en tiempos polinomiales.

El camino teórico realizado hasta llegar hasta este algoritmo fue tan complejo como distinto de su actual alternativa clásica. En dicho camino se diseñaron subrutinas como la *Quantum Fourier Transform* y la Estimación de Fase [2, Capítulo 5], que disponen de un carácter lo suficientemente generalizado como para dar lugar a diversas aplicaciones para soluciones de problemas que van más allá de la Teoría de Números y la Criptografía.

La aplicación que a nosotros nos concierne es la del análisis de riesgos financieros, de donde actualmente dispondríamos de una aceleración cuadrática en tiempos de ejecución frente a las alternativas clásicas. En este trabajo estudiaremos uno de los algoritmos de análisis de riesgo de crédito propuestos, tanto en su implementación como en su proyección hacia un problema del mundo real.

1.2. Motivación del Problema y Estado del Arte

El mundo de la economía y las finanzas se caracteriza por una incertidumbre e importancia que ha dado lugar a un continuo cambio tanto en los modelos económicos planteados como en los cómputos necesarios para calcular y describir estos modelos. Como es de esperar en nuestro contexto tecnológico actual, se abre la línea de investigación hacia cómo las aceleraciones que aportan los ordenadores cuánticos podrían ayudar a predecir los modelos propuestos.

Una forma de abordar el problema de describir un modelo planteado es el de simularlo un número de veces y realizar un muestreo de sus resultados. Este método de simulación se conoce como el Método de Montecarlo y, como norma general, mientras que en mínimos tamaños del modelo es abordable la simulación clásica, a mayor tamaño de variables a tener en cuenta comienza a volverse inviable nuestra simulación. Comprendamos a continuación la motivación del problema del Análisis de Riesgo de Crédito y nuestra situación actual tanto teórica como tecnológica.

1.2.1. Análisis de Riesgo de Crédito: Motivación

Supongamos que somos una entidad bancaria que ofrece K diferentes préstamos a sus clientes, figurando en su portfolio financiero. Cada uno de dichos préstamos supondrá no disponer de cierta cantidad de dinero hasta el momento en el que nuestro cliente sea capaz de devolverlo, momento que no siempre llega. Por ello, por las posibles insolvencias de cada préstamo actual, el banco deberá disponer de cierto fondo económico para declararse solvente “casi seguramente”. Además, el correcto y convincente análisis de la viabilidad y robustez económica de una entidad financiera supone un mayor número de beneficios en materia de provisiones que puedan repartir organismos reguladores como los

bancos centrales; segundo motivo que aporta importancia a nuestro problema, conocido como análisis de riesgo de crédito [3].

1.2.2. Estado del Arte

Son múltiples tanto los modelos económicos para el análisis de riesgos financieros como las alternativas para ejecutar algoritmos cuánticos. Como ejemplo, la compañía de computación cuántica D-Wave analiza estos problemas de finanzas como la representación del modelo económico en un modelo físico energético, buscando la solución óptima como el estado cuántico de menor energía.

En nuestro caso, tomaremos como referencia la solución sugerida por IBM. De la mano de investigadores como Stefan Woerner y Daniel J. Egger [4] [5], se propone una alternativa cuántica a la ejecución del método de Montecarlo con una modelización que sigue las pautas de Basilea II [3]. Dicha alternativa, generalizable en la teoría para K activos, se caracteriza por el uso de algoritmos de estimación de amplitudes, que dan lugar a una aceleración cuadrática con respecto a la alternativa clásica. Una vez disponemos de la solución "en papel", urge el paso hacia un punto programable, preguntándonos qué herramientas para elaborar algoritmos cuánticos son utilizadas en la actualidad.

Son varias las librerías y lenguajes de programación existentes para ello, destacando Qiskit (IBM), una librería de Python. Elegimos Qiskit frente a otras alternativas sobre diseño programático de circuitos cuánticos por las siguientes razones:

1.– En primer lugar, se buscaba un *Software Development Kit* con acceso a procesadores cuánticos, motivo por el cual se reducía nuestra búsqueda a librerías como *ProjectQ*, *Forest*, *t|ket>*, *Circ* o, por supuesto, Qiskit.

2.– Partimos además eligiendo Qiskit con la ventaja de disponer de acceso a ordenadores cuánticos de mayor potencia gracias al Hub-CSIC.

3.– Nuestra familiarización con el lenguaje y *workflow* de Qiskit nos hace disponer de una base práctica por delante de cualquier otra alternativa; una base importante para manipulaciones avanzadas como las que han sido necesarias.

4.– Por último, el hecho de que el tutorial de Qiskit para el análisis de riesgo de crédito [6] estuviera codificado con ayuda de Qiskit nos hace obviar nociones de compatibilidad entre ambos códigos.

Actualmente Qiskit está pasando por numerables procesos de actualización y *debugging*, pero sigue siendo una de las alternativas más viables para manipular qubits, ya sean simulados o en ordenadores cuánticos reales (a los que nos referiremos como *backends* a lo largo de esta memoria). El algoritmo cuántico mencionado anteriormente está en parte implementado e incluso incluido en tutoriales de Qiskit [6], pero el mayor inconveniente es que fue programado para únicamente dos activos ($K = 2$); punto en el que entra nuestro trabajo a realizar.

1.3. Objetivos a completar

Con todo esto, son varios los objetivos que tendrán lugar a lo largo de este trabajo:

1.– Comprender y explicar los fundamentos de la computación cuántica para disponer de suficiente base como para analizar la alternativa cuántica al método de Monte Carlo: La estimación de amplitud cuántica (QAE) y sus variantes iterativa (iQAE) y por máxima verosimilitud (MLAE).

2.– Comprender y explicar conceptos del análisis de riesgo de crédito, pasando a analizarlos mediante la propuesta cuántica de nuestro *paper* de referencia de IBM.

3.– Partiendo de la *demo* del tutorial de análisis de riesgo de crédito de IBM, implementar en Qiskit una librería generalizable a K activos, capaz de simplificar para un usuario ajeno a la base teórica de este algoritmo el proceso de ejecución.

4.– Ejecutar en simuladores ejemplos de análisis de portfolios para dos y más de dos activos ($K \geq 2$), analizando precisión y longitudes de los circuitos.

5.– Estudiar la viabilidad de la ejecución de análisis de portfolios sobre ordenadores cuánticos reales y su proyección a futuros "problemas del mundo real".

1.4. Estructura del Documento

En primer lugar, el Capítulo 2 dará la introducción tanto a los algoritmos cuánticos que vamos a necesitar comprender y utilizar como a los conceptos de estadística y matemática financiera pertinentes para el análisis de riesgo de crédito.

Nos encontramos ante un trabajo sobre el cual confluyen dos ramas de conocimiento a primera instancia muy diferentes. Una vez introduzcamos lo suficiente cada una de estas dos, iremos comprendiendo cómo se acaban uniendo conceptos a la hora de ejecutar estas nuevas alternativas cuánticas para el análisis de riesgo de crédito; motivo por el cual se destaca el peso teórico inicial de este trabajo.

A continuación, ambas introducciones se unen en el Capítulo 3, tomando como referencia el *paper* [4] de IBM para el estudio de las puertas cuánticas que hemos de crear y sobre las que estimar sus amplitudes. Presentaremos a su vez la clase `CreditRiskAnalysis`, que toma como base el tutorial existente de Qiskit pero añade tanto un amplio abanico de parámetros y alternativas a la QAE como la generalización a K activos de un portfolio.

En el Capítulo 4 analizamos los resultados de nuestras ejecuciones, tanto en simuladores como ordenadores cuánticos reales; así como la viabilidad hacia futuros problemas reales que pueda resolver esta librería. Concluimos finalmente en el Capítulo 5.

2

Introducción a la Computación Cuántica y al Análisis de Riesgo de Crédito

Introduciremos a lo largo de este capítulo conceptos preliminares de las dos materias principales de esta memoria: primeramente acerca de la computación cuántica y los algoritmos de estimación de amplitud (QAE, iQAE y MLAE), finalmente sobre la estadística necesaria para plantear nuestro problema de análisis de riesgo de crédito y sobre los estadísticos que ayudan a conocer nuestro modelo de incertidumbre propuesto.

2.1. Introducción a la Computación Cuántica. Estimaciones de amplitud

Los algoritmos de estimaciones de amplitud cuánticos requieren de una exigente base teórica como para su completa comprensión. Conceptos como las propiedades físicas y matemáticas del *qubit*, superposición y entrelazamiento; así como el conocimiento de las puertas cuánticas y su composición en circuitos cuánticos constituyen esta base teórica necesaria.

Se proporciona en el Anexo, Apéndice A, la definición y explicación de lo anteriormente enumerado para que con su lectura se pueda ser capaz de pasar a aplicaciones más avanzadas como las de este trabajo.

2.1.1. Formalización del Problema de Estimación de Amplitud

Presentamos en esta sección un algoritmo prometedor para ser aplicado en el análisis de riesgos financieros y utilizable en bastantes más disciplinas. Se sigue lo explicado en el artículo que presenta el algoritmo en cuestión: *Quantum Amplitude Amplification and Estimation* [7], así como el artículo principal que seguimos a lo largo del trabajo [4, Apéndice B].

Consideremos una función booleana desconocida $\chi : X \mapsto \{0, 1\}$, que particiona X entre elementos “buenos” y “malos” (1 o 0, respectivamente). El problema que se plantea consiste en poder identificar qué elementos de X son “buenos” y “malos”, e.d., $\chi^{-1}(0)$ y $\chi^{-1}(1)$. Se entiende que cuanto

mayor es el cardinal de X, más difícil de descifrar es esta función de "caja negra", y si se hace inviable la determinación de la partición de X, el problema pasa a al menos ser capaz de estimar con precisión la probabilidad de, eligiendo al azar un elemento de X, que este vaya a parar al 1.

Clásicamente es clara la alternativa de simulación a fuerza bruta: Vamos a elegir elementos al azar de X y acabamos realizando un conteo del cardinal de las preimágenes de χ. Computacionalmente, este proceso acaba siendo costoso cuanto mayor sea el cardinal de X y más definidos estén los límites de los $x \in X$ "buenos".

Este problema se puede reinterpretar en el contexto de puertas cuánticas y operadores algebraicos. Sea $\mathcal{A}$ un algoritmo cuántico cuyo efecto sobre X es desconocido. Es decir, el efecto de $\mathcal{A}$ sobre $|0\rangle$ da lugar a

$$\mathcal{A}\,|0\rangle_{n+1} = \sum_{x\in X} \alpha_x\,|x\rangle = \sqrt{1-a}\,|\psi_0\rangle_n\,|0\rangle + \sqrt{a}\,|\psi_1\rangle_n\,|1\rangle\,, \tag{2.1}$$

obteniendo así una superposición cuántica de los elementos de X. Atendiendo a la relación entre amplitudes de un estado y probabilidades de medición, nos encontramos con la posible reexpresión del problema en el hecho de ahora querer estimar la probabilidad de medir 1 sobre cierta superposición creada $|\psi_x\rangle$, donde dicha probabilidad se denotará como a. Puesto que $\sqrt{a}^2 + \sqrt{1-a}^2 = 1$, podemos normalizar este valor entre 0 y 1 mediante senos y cosenos, de forma que el problema se transforma en una estimación de fase.

$$|\psi_x\rangle = \sqrt{1-a}\,|\psi_0\rangle\,|0\rangle + \sqrt{a}\,|\psi_1\rangle\,|1\rangle = \sin(\theta_a)\,|\psi_1\rangle\,|1\rangle + \cos(\theta_a)\,|\psi_0\rangle\,|0\rangle\,. \tag{2.2}$$

Los órdenes de ejecución de esta alternativa cuántica siguen estando empatados frente a la alternativa anterior: realmente estamos simulando el circuito las veces que sean necesarias para estimar esa probabilidad, puesto que en una superposición de estados es imposible determinar las amplitudes, obteniendo al medir alguno de los estados superpuestos. A continuación veremos dónde se experimenta la aceleración cuadrática en esta alternativa.

2.1.2. *QAE: Quantum Amplitude Estimation*

La estimación de amplitud cuántica (de ahora en adelante QAE por sus siglas en inglés), consigue aportar una aceleración en órdenes de ejecución con respecto a la precisión que se espera en $\frac{1}{p_x}$ ejecuciones (siendo p_x la probabilidad de obtener una x tal que $\chi(x) = 1$) [7]. La clave de este algoritmo reside en ser capaz de amplificar la amplitud existente en el estado $|\psi_x\rangle$ a partir de manipularlo con ciertas puertas lógicas.

Para poder pasar de la probabilidad de obtener un resultado bueno a la amplitud amplificada de $|\psi_x\rangle$, se presenta el siguiente operador cuántico

$$\mathcal{Q} = -\mathcal{A}\mathcal{S}_0\mathcal{A}^\dagger\mathcal{S}_{\psi_0} \tag{2.3}$$

Analicemos cada elemento que compone $\mathcal{Q}$:

1.– En primer lugar, recordamos que $\mathcal{A}$ es el operador que representa el problema de la función χ.

2.– La puerta lógica $\mathcal{S}_0 = I_{n+1} - 2\,|0\rangle_{n+1}\,\langle 0|_{n+1}$, analizando su expresión, vemos que invierte el signo del estado $|0\rangle$ y deja igual el resto de estados.

3.– $\mathcal{A}^\dagger$ es la inversa de $\mathcal{A}$ (principio de computación reversible).

4.– Por último, $\mathcal{S}_{\psi_0}$ es análogo al operador anterior e invierte el signo al estado "bueno", mientras que deja igual el resto de estados.

Con los cálculos pertinentes se puede llegar a la ventaja que supone elevar a cierta potencia m el operador $\mathcal{Q}$:

$$\mathcal{Q}^m\,|\psi\rangle = \sin((2m+1)\theta_a)\,|\psi_1\rangle\,|1\rangle + \cos((2m+1)\theta_a)\,|\psi_0\rangle\,|0\rangle\,. \tag{2.4}$$

Este fenómeno, la amplificación de amplitud, proporciona el hecho de ser capaz de recrear una "malla" sobre la circunferencia goniométrica a partir de la cual iríamos granulando en qué celda se sitúa θ_a. Obteniendo dicho valor y aplicando una transformada de Fourier inversa (IQFT), hallamos el valor de a.

2.1.3. Mejoras de la *QAE*

Si bien en la teoría y en la simulación la QAE es bastante superior a las alternativas clásicas actuales, bien es sabido que un algoritmo cuántico en la práctica y con los medios actuales acaba viendo bastante reducido su potencial. Vemos que tenemos un circuito excesivamente largo (en gran parte por iterar $2^m - 1$ veces $\mathcal{Q}$). Esta longitud afecta a los tiempos de coherencia necesarios para conseguir ejecuciones estables, y veremos en los siguientes capítulos cómo con Qiskit se acaba haciendo inviable ejecutar la QAE con la precisión esperada en ordenadores cuánticos reales. Con ello, por la mejorable calidad de la precisión de las simulaciones ejecutadas mediante QAE, solo mostraremos ejemplos realizados con ayudas de las siguientes mejoras que explicamos a continuación.

Sin embargo, damos entrada en esta sección a las alternativas y mejoras que ha experimentado la QAE hacia algoritmos que han mejorado tiempos de ejecución y de coherencia.

iQAE: *Iterative Quantum Amplitude Estimation*

La estimación de amplitud iterativa se considera actualmente entre las mejores estimaciones de amplitud que no requieren de estimación de fase (*Quantum Phase Estimation*), un proceso especialmente costoso en número de puertas lógicas requeridas y número de *qubits*. [8]

Como idea general, este algoritmo diecinueve años más moderno que su antecesor se basa en establecer iterativamente un intervalo de confianza para la fase en la que se halla $|\psi\rangle$, de forma que se obtiene un rango para la amplitud como sigue

$$[a_l, a_u] = [sin^2(\theta_l), sin^2(\theta_u)], \tag{2.5}$$

Con u y l denotando *upper* y *lower*.

Es importante denotar dos parámetros nuevos con respecto a la QAE cuyas variaciones acaban influyendo considerablemente en la salida del algoritmo.

1.– Un parámetro $\alpha \in (0, 1)$ denota la confianza de los intervalos a calcular, cuyo porcentaje es $\frac{(1-\alpha)}{T}$. Con ello, α influirá en el tamaño de los intervalos de confianza resultantes, si bien no directamente como podríamos pensar debido al parámetro T.

2.– Un valor de precisión $\epsilon \in (0, \frac{1}{2})$. Influye en la precisión del algoritmo tanto para la condición de parada (pues en el momento en el que $\theta_u - \theta_l < 2\epsilon$ el algoritmo finaliza), como para la influencia del parámetro T sobre α. T, el número máximo de iteraciones, se calcula como $T = \lceil \log_2(\pi/4\epsilon) \rceil$.

La profundidad de la iQAE se reduce significativamente con respecto a la QAE por dos motivos. El primero reside en el hecho de no ejecutar todo el algoritmo “de una vez”, sino dividirlo en distintas iteraciones. El segundo motivo viene dado por realizar una búsqueda efectiva de la k_i-ésima potencia a la que se eleva el operador $\mathcal{Q}$ en lugar de calcular las distintas $2^m - 1$ potencias.

MLAE: *Maximum Likelihood Amplitude Estimation*

La estimación de amplitud por máxima verosimilitud está “empatada” con la iterativa en tiempo de ejecución y precisión en la mayoría de los estudios pertinentes. Coincide con la iterativa en obviar la estimación de fase [9].

La idea principal que reside en la MLAE está en, al igual que en la QAE, iterar potencias de $\mathcal{Q}^{m_k}$. Sin embargo, en lugar de estimar θ_a mediante el mallado de la circunferencia, se hace uso de la máxima verosimilitud (puesto que se pueden tratar las medidas de $\mathcal{Q}^{m_k}|\psi\rangle$ como una Bernoulli por existir cierta probabilidad $p \in (0, 1)$ de obtener 1 y $1 - p$ de obtener 0). Con ello, sea N_k el número de *shots* del circuito y G_k el número de mediciones buenas (*good*), la verosimilitud para cierto m_k es

$$VERO_k(G_k; \theta_a) = (\sin^2((2m+1)\theta_a))^{G_k} + (\cos^2((2m+1)\theta_a))^{N_k - G_k}, \quad k = 0, \cdots, M \tag{2.6}$$

Y, por último, MLAE une estas M funciones de verosimilitud en un productorio para poder estimar θ_a:

$$\hat{\theta_a} = argmax(\prod_{k=0}^{M} VERO_k(G_k;\theta_a)). \tag{2.7}$$

Por lo tanto, la amplitud estimada es $\hat{a} = \sin^2(\hat{\theta_a})$.

El parámetro que añade MLAE y que hemos de destacar es el número máximo de iteraciones, M. Es comprensible que, a mayor M, más sensible y precisa será nuestra estimación por máxima verosimilitud. De forma similar a la iQAE, el hecho de dividir las ejecuciones en distintas iteraciones resulta en circuitos de menores exigencias en tiempos de coherencia.

2.2. Introducción al Análisis de Riesgo de Crédito. Valor en Riesgo de un Portfolio

El riesgo de crédito de una entidad financiera se define como la probabilidad de perder dinero en un activo, bien sea por incapacidad de un deudor o por un resultado negativo en una inversión. Un banco ha de establecer ciertas medidas cualitativas y cuantitativas a una exposición con tal de poder "adelantarse" a los acontecimientos. Si bien queda fuera del alcance de este trabajo, se menciona el hecho de que factores como el destino del crédito, la información de ingresos de un cliente, la garantía o aval y (importante) la situación económica general son algunos ejemplos de estas medidas que al final nos llevan a una probabilidad de "acertar" o "fallar".

Una vez tengamos las probabilidades de los activos de la entidad, tendríamos una variable aleatoria donde cada posible pérdida almacenaría un valor computado en función del valor y riesgo de cada activo. Con ello, el problema a resolver en el análisis de riesgo de crédito consiste en obtener probabilidades y estudios estadísticos de dicha distribución con tal de determinar la posibilidad en la que una entidad se tendría que declarar insolvente.

Adelantamos que, de entre los estadísticos a presentar para el estudio de nuestra variable aleatoria de pérdidas, el valor a riesgo *alpha* o *Value at Risk* será el principal, que calcularemos mediante búsquedas en bisección y estimaciones de amplitud. Este estadístico supone un umbral recomendado para una entidad financiera para, no únicamente mostrarse solvente al final de un período de tiempo con confianza $\alpha \in (0,1)$ (por lo general se hace uso de $\alpha = 95\,\%$ e incluso $\alpha = 99{,}9$), sino además obtener beneficios tras analizar provisiones por parte de entidades supervisoras.

Serían múltiples las formas de abordar este problema: Algunos modelos se ajustarían mejor o

peor a lo actual, tendrían mayor o menor complejidad, predecirían mejor o peor valores futuros... Sin embargo, son uniformes y definidas las directrices establecidas desde gobernanzas centrales como el Banco Central Europeo o los acuerdos de Basilea (I, II y III), puesto que la insolvencia de un único banco podría suponer catastróficas consecuencias para el resto.

Específicamente, procederemos a presentar la formalización matemática de nuestro problema con tal de comprender mejor qué distribuciones aleatorias se generan a partir de computar estas probabilidades y en qué pueden ayudar a las entidades financieras. El modelo matemático explicado a continuación procede de [3], el *Internal Rated Bases* (IRB) del acuerdo segundo de Basilea, que describe el Modelo de dependencia Condicional Gaussiana, o *Gaussian Condicional Independence model* (GCI).

2.2.1. Formalización matemática. Modelo GCI

Procedemos a comprender la descripción estadística y modelización que aporta el modelo de dependencia condicional gaussiana descrito en el IRB.

Presentación de los parámetros

Son varios los parámetros a presentar en el modelo del IRB:

1.– Primeramente, necesitaremos definir el comportamiento de los K activos financieros que querramos evaluar (L_i), que serán variables aleatorias cuyo resultado podrá ser diario, mensual, anual, etc.

2.– Cada activo financiero supondrá una pérdida de cierto valor en caso de fallo λ_i.

3.– Recordando los parámetros que afectaban a la probabilidad de rendimiento de una exposición, hará falta tener en cuenta el estado económico actual y modelizarlo en cierta variable aleatoria (Z).

4.– Presentamos también un parámetro de sensibilidad frente a dicho estado económico para cada L_i, que denotaremos como $\rho_i \in [0, 1)$.

Por lo tanto, nuestro objetivo es describir matemáticamente el valor de $\mathcal{L} = \sum_{i=1}^{K} L_i$, e.d, el agregado de pérdidas.

Modelización de los parámetros

El hecho de que las L_i supongan "perder" o "ganar" nos da lugar a relacionar esta dicotomía con una distribución de Bernoulli. Sin embargo, faltaría añadir el λ_i para ponderar la pérdida resultante:

$$L_i = \lambda_i \cdot X_i, \quad X_i \sim Ber(p_i). \tag{2.8}$$

Claramente cada X_i toma el valor 0 o 1 por realización, mientras que cada L_i resultará en 0 o

λ_i. No obstante, queda por definir cada probabilidad de fallo p_i, que deberá atender a parámetros de partida como ingresos del cliente o el destino del crédito; pero también a la idiosincrasía económica, que con ejemplos como el año 2008 se deja clara su importancia en algunas exposiciones.

Con ello, dando origen al nombre del modelo GCI, se modeliza la situación económica general con una distribución normal estándar (o gaussiana) $Z \sim \mathcal{N}(0,1)$.

Expresamos a continuación esta dependencia condicionada, denotando como p_i^0 la probabilidad inicial de fallo del activo, $\phi(x) : \mathbf{R} \mapsto [0,1]$ como la función de distribución acumulada de la gaussiana y, por último, $z \in \mathbf{R}$ como una realización de la variable aleatoria Z.

$$p_i(z) = \mathbf{P}(X_i = 1 \mid Z = z) = \phi\left(\frac{\phi^{-1}(p_i^0) - \sqrt{\rho_i} z}{\sqrt{1-\rho_i}}\right) \tag{2.9}$$

Realizamos varias observaciones en la ecuación 2.9 para comprobar la correcta definición de $p_i(z) : \mathbf{R} \mapsto [0,1]$:

1.– En el hipotético caso de analizar activos sin dependencia alguna con respecto a un estado global realizado por Z, sustituyendo $\rho_i = 0$ en la definición de $p_i(z)$ vemos que $p_i(z) = \phi(\phi^{-1}(p_i^0)) = p_i^0$, comprobando que el parámetro p_i^0 en efecto representa la probabilidad inicial.

2.– Tomando límites en z apreciamos la monotonía creciente de $p_i(z)$:

$$\lim_{z \to -\infty} p_i(z) = \phi(-\infty) = 0; \quad \lim_{z \to +\infty} p_i(z) = \phi(+\infty) = 1 \tag{2.10}$$

En otras palabras, realizaciones "muy buenas" de z (cuando tienda a menos infinito, traduciéndose como épocas de bonanza económica) harán improbable el fallo de un activo y viceversa.

3.– Tomando límites en p_i^0 apreciamos la dependencia de $p_i(z)$ con respecto a la probabilidad inicial:

$$\lim_{p_i^0 \to 0} p_i(z) = \phi(-\infty) = 0; \quad \lim_{p_i^0 \to 1} p_i(z) = \phi(+\infty) = 1 \tag{2.11}$$

Comprobamos aquí que sigue permaneciendo la noción de la probabilidad inicial, en el sentido de que "activos muy arriesgados o muy seguros lo seguirán siendo".

Estadísticos a utilizar. *Value at Risk*

Por último, una vez conocemos la distribución de nuestro modelo de pérdidas $\mathcal{L}$, queda preguntarnos por cuáles estadísticos utilizaremos con tal de analizar posibles riesgos en la gobernanza de nuestros activos.

Primeramente hacemos hincapié en el hecho de que $\mathcal{L}$, con cada simulación, nos devolverá una cantidad monetaria de pérdidas en nuestros activos (en \$, €, millones de \$, etc.), luego será necesario que el banco mantenga una cantidad de pasivos mayor a lo recomendado por ciertos estadísticos para mantenerse solvente.

El primero y más obvio de los estadísticos es la esperanza de $\mathcal{L}$, $\mathbf{E}[\mathcal{L}] \in \left[0, \sum_{i=1}^{K} \lambda_i\right]$. El cálculo

de este valor es simple en el supuesto de independencia entre las L_i, calculándose como sigue

$$\mathbf{E}[\mathcal{L}] = \sum_{i=1}^{K} \mathbf{E}[L_i] = \sum_{i=1}^{K} \lambda_i p_i^0. \tag{2.12}$$

Este supuesto solo ocurre cuando $\rho_1 = \cdots = \rho_K = 0$. De lo contrario, el cálculo adquiere bastante complejidad al depender cada L_i de Z, motivo por el cual se simularán los sucesos y el cálculo de la esperanza.

Como es de esperar en una entidad financiera, las exposiciones de la misma suelen conllevar un riesgo bajo *a priori* y esto causa que, salvo extremas realizaciones de Z, la distribución de $\mathcal{L}$ presente unas colas muy ligeras al final de los intervalos de pérdidas y acumule la mayoría de sus probabilidades al inicio. Esto conlleva que vamos a obtener valores de $\mathbf{E}[\mathcal{L}]$ considerablemente bajos y que el fin de este estadístico será meramente informativo (pocos bancos querrían asumir el riesgo de mantenerse con unos fondos que rondaran la esperanza de las pérdidas).

Por este motivo, ajustando un $\alpha \in [0, 1]$, se define el Valor de Riesgo al nivel de confianza α o *Value at Risk* (VaR) como el máximo de las pérdidas de $\mathcal{L}$ que tienen cabida en el percentil de α, o bien como el mínimo de las pérdidas de probabilidad mayor que α [4, Sección II].

$$VaR_\alpha[\mathcal{L}] = \inf_{x \geq 0} \{x \mid \mathbf{P}(\mathcal{L} \leq x) \geq \alpha\}. \tag{2.13}$$

Como ejemplo, tomando de las finanzas una confianza usual como es el $99{,}9\,\%$, el $VaR_{99{,}9\,\%}[\mathcal{L}]$ [1] supone una cantidad financiera en el que el $99{,}9\,\%$ de los casos la entidad se mantendrá solvente.

En conclusión, mediante algoritmos cuánticos, vamos a querer obtener la siguiente información mediante estadísticos de la variable aleatoria $\mathcal{L}$:

1.– En primer lugar, vamos a simular las ocurrencias de $\mathcal{L}$ cuánticamente, si bien primeramente computaremos los resultados clásicamente para tener referencia de los márgenes de error. Llamaremos a este paso la "ejecución mixta".

2.– A continuación, estimaremos $E[\mathcal{L}]$ para obtener la noción de la pérdida media del portfolio.

3.– Después, opcionalmente y para cerciorarnos de la correctitud del paso último, representaremos gráficamente la función de distribución acumulada estimando $CDF_{\mathcal{L}}(x)$ para $x = 0, \cdots, \sum \lambda_i$.

4.– Por último, realizaremos una búsqueda en bisección sobre estos valores de $CDF_{\mathcal{L}}(x)$ para obtener el *Value at Risk* para un α dado.

[1] A lo largo de esta memoria hablaremos también de α como la probabilidad de errar y no como el nivel de confianza, en especial cuando pasemos a la implementación de este estadístico. Se entenderá que hablamos de este caso alternativo en valores como $VaR_{0{,}01\,\%}[\mathcal{L}]$, por ejemplo.

3

Estudio de Portfolios Financieros mediante Algoritmos Cuánticos. Diseño e Implementación de la solución

Previo a entrar en la librería de *qiskit* para dar resolución al problema, recapitularemos la algoritmia con la que se debería proceder para representar el modelo GCI y sus estadísticos, haciendo hincapié en conceptos de la computación cuántica que nos ayudarán a implementar este algoritmo.

3.1. Redefinición del problema orientada a la manipulación de *qubits*

Vamos a analizar los cuatro objetivos propuestos en el capítulo anterior y preguntarnos cómo implementar esto mediante recursos cuánticos:

1.– Para el primer paso, el modelo mixto solo requerirá de crear un operador cuántico que simule los $\{X_i\}_{i=1}^{K}$ (y con ello Z), para después juntar los resultados clásicamente en $\mathcal{L}$. Este operador cuántico tendrá por nombre $\mathcal{U}$ y daremos detalle en su creación más adelante.

2.– Para estimar la esperanza de $\mathcal{L}$, haremos uso de la QAE sobre un operador $\mathcal{A}$ transformable en $\mathcal{Q}$ tal y como figura en la Ecuación 2.3. Este operador será una composición de $\mathcal{U}$ junto con una suma ponderada $\mathcal{S}$ con tal de expresar $\mathcal{L} = \sum \lambda_i X_i$.

3.– Por último, estimaremos la función de distribución acumulada usando mismamente la QAE sobre $\mathcal{A}$ que, a diferencia del paso anterior, tendrá añadido un operador $\mathcal{C} = \mathcal{C}(x)$ que comparará el valor de $\mathcal{L}$ con una x dada y afectará a un *qubit* objetivo. Se entenderá como "bueno" cuando $x \geq \mathcal{L}$. El valor a riesgo α se buscará en bisección clásicamente.

Por lo tanto, vamos a analizar individualmente cada paso del operador $\mathcal{A} = \mathcal{CSU}$ del cual estimaremos su amplitud.

3.1.1. Modelo de Incertidumbre: $\mathcal{U}$

Recordemos que, para representar el *Gaussian Conditional Independence model*, será necesario simular una distribución normal $Z \sim \mathcal{N}(0, 1)$, cómo cada resultado z de Z afectará a cada $p_i(z)$ y, por último, simular cada ocurrencia de las distribuciones de Bernoulli $X_i \sim Ber(p_i(z))$.

Tomando como referencia la condición de normalización y las probabilidades de medición de un estado en función de su amplitud, resulta que un *qubit* puede representar fielmente una distribución de Bernoulli si lo rotamos debidamente sobre el eje Y y medimos sobre el eje X. Por ejemplo, podríamos simular un $X \sim Ber(1/2)$ midiendo sobre el eje X un *qubit* al que le hayamos aplicado una Hadamard (H).

El objetivo entonces es buscar una forma de dotar a cada X_i una representación con un *qubit* que disponga de un ángulo tal que $|\beta|^2 = p_i(z)$ (estableceremos así el "fallo" de un activo como la medición de 1 en un *qubit*). Tomando presente la normalización realizada en la Ecuación 2.2 y la expresión de probabilidades del GCI dada por la Ecuación 2.9, vemos que será necesario establecer una fase, θ_i, como sigue:

$$\theta_i(z) = \arcsin\left(\sqrt{\frac{\phi^{-1}(p_i^0) - \sqrt{\rho_i}z}{\sqrt{1-\rho_i}}}\right) \tag{3.1}$$

Sin embargo, una observación que hemos de tener en cuenta antes de intentar diseñar un *qubit* con una fase como la anterior está en el hecho de que z es un número real. Esto conlleva una notoria complejidad con respecto al número de *qubits* necesarios para representar este valor. Con ello, pasamos a realizar una discretización de la Gaussiana Z pasando a tener 2^{n_z} valores comprendidos en el intervalo $[-z_{max}, z_{max}]$ ($z_{max} = 2$ por defecto), con n_z parámetro que indica el usuario para establecer el número de *qubits* a usar.

Finalmente, aplicando una rotación (R_Y) sobre los n_z *qubits* dados (llamamos a esta parte $\mathcal{U}_Z$ [1]) y preparando rotaciones controladas sobre los K *qubits* que representan los X_i (además de una primera rotación basada en el p_i^0), conseguimos construir la puerta $\mathcal{U}$, que hace uso de $n_z + K$ *qubits* y aplica a los K últimos una rotación basada en la Ecuación 3.1, aplicada una aproximación lineal de funciones $\{a_i x + b_i\}_{i=0}^{n_z-1}$. Se adjunta un esquema del circuito de simulación de $\mathcal{U}$ en la Figura 3.1.

3.1.2. Agregado de los resultados: $\mathcal{S}$

Si bien veremos en los resultados de las simulaciones que ya podemos computar clásicamente las diferentes ocurrencias de $\mathcal{U}$ para tener una fiel representación del modelo, a continuación hemos de construir mediante puertas cuánticas cómo deberíamos sumar las ocurrencias de "activos fallidos" en función de su valor (λ_i).

Esta puerta cuántica tendrá como nombre $\mathcal{S}$, y tomará los K *qubits* de los resultados de $\mathcal{U}$ para guardar en un estado $|0\rangle_{n_S}$ el valor de la suma ponderada. La longitud del resultado, n_S, atiende al valor máximo resultante posible, es decir,

[1] Las rotaciones R_Y de $\mathcal{U}_Z$ han de realizarse con unos ángulos basados en $arcsin(\sqrt{p_i^Z})$, con p_i^Z la probabilidad de caer en la región i-ésima particionada de la normal Z. [10]

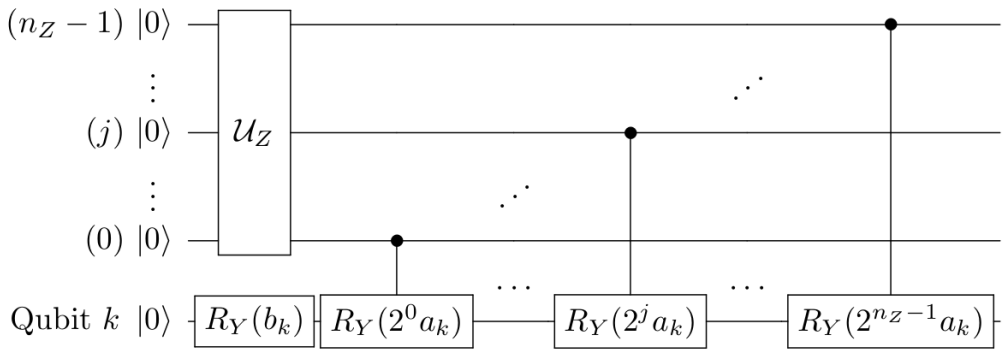


Figura 3.1: Circuito de simulación del modelo de incertidumbre $\mathcal{U}$ aplicado a un activo X_k. [4, Figura 2]

$$n_S = \lfloor \log_2(\lambda_1 + \lambda_2 + \cdots + \lambda_K) \rfloor + 1. \tag{3.2}$$

Con ello, la acción del operador $\mathcal{S}$ se puede resumir con la siguiente expresión:

$$\mathcal{S} : |x_1, \cdots, x_K\rangle_K |0\rangle_{n_S} \mapsto |x_1, \cdots, x_K\rangle_K \left| \sum_{i=1}^{K} \lambda_i x_i \right\rangle_{n_S}. \tag{3.3}$$

Construir una puerta cuántica que pueda llevarnos al valor de la suma ponderada, conocidos de antemano los valores de λ_i, se realiza primeramente mediante *flips* (e.d., del $|0\rangle$ al $|1\rangle$) controlados por cada x_i sobre los $\lfloor \log_2(\lambda_i) \rfloor + 1$ *qubits* necesarios, pasando a sumarse sobre el valor agregado total.

Conseguimos de esta forma establecer una fase sobre el *ket* inicial $|0\rangle_{n_S}$ que lo sitúa entre el valor nulo y el valor máximo $|\sum \lambda_i\rangle$, de forma que se puede interpretar como una amplitud $a \in [0, 1]$ cuya estimación nos resulta en el valor medio esperado de la suma computada por $\mathcal{S}$. En otras palabras, aplicando una QAE sobre el operador $\mathcal{A} = SU$ con respecto a los últimos n_S *qubits* es como se obtiene mediante alternativa cuántica al método de Montecarlo el valor de $E[\mathcal{L}]$.

3.1.3. Comparación con valores de x: $\mathcal{C}$

Por último, la última puerta cuántica que nos ayuda a estimar mediante estadísticos el valor a riesgo α, $\mathcal{C}$, consiste en un comparador que "voltea" un *qubit* inicializado a $|0\rangle$ en función de si el resultado de la suma anterior es mayor o menor que un x dado. Es decir, el operador $\mathcal{C}$ se define como sigue

$$\mathcal{C} : |i\rangle_{n_S} |0\rangle \mapsto \begin{cases} |i\rangle_{n_S} |0\rangle & \text{si } i \leq x, \\ |i\rangle_{n_S} |1\rangle & \text{en otro caso.} \end{cases} \tag{3.4}$$

Por lo tanto, nos es posible identificar la composición de los operadores anteriores junto a $\mathcal{C}$ como el modelo de caja negra que nos discierne un último *qubit* de entre los introducidos como "bueno" o "malo" en función del valor agregado resultante.

Esta capacidad de discernimiento resulta coincidir con la función de distribución acumulada de $\mathcal{L}$, y por lo tanto a la estimación de $VaR[\mathcal{L}]$.

Por último, realizamos hincapié en la composición de los tres operadores mostrados en estas secciones que dan lugar a $\mathcal{A}$, en el sentido de cómo parte de la salida de un operador se enlaza con la entrada del siguiente.

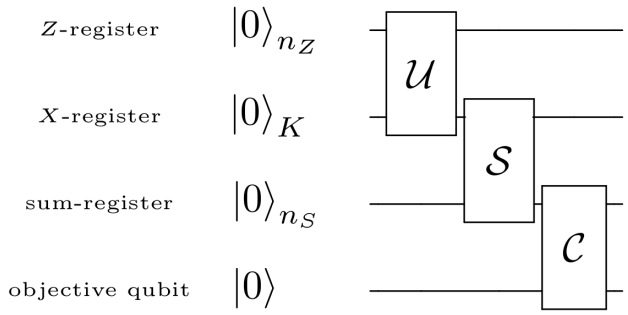


Figura 3.2: Composiciones de los operadores U, S y $\mathcal{C}$. [4, Figura 1]

3.2. Clase `CreditRiskAnalysis`

Hemos creado la clase `CreditRiskAnalysis` como interfaz entre un usuario que desee experimentar con estos métodos alternativos de análisis de riesgos financieros pero carezca de los conocimientos en Qiskit necesarios. Este código implementado veremos que se basa conceptualmente en una *demo* de uno de los ensayos seguidos en el trabajo [6], si bien conseguimos de esta forma poder computar resultados sin seguir el tutorial paso a paso, así como que resaltaremos diferencias pasadas y presentes existentes entre nuestro código y el tutorial. Recordemos haber realizado la justificación de nuestra elección de Qiskit para implementar la clase `CreditRiskAnalysis` en la Sección 1.2.2.

El código fuente de esta clase, junto con muestras de ejecuciones y gráficas relacionadas con la memoria, se encuentra subido como repositorio en Github [11].

Recalcando una vez más, comentamos el hecho de que el paso del diseño conceptual de secciones anteriores hacia el código en Qiskit ha sido dado como base a partir del tutorial ya existente; y que nuestro trabajo comienza tomándolo como punto de partida y construye hacia la generalización a K portfolios de lo hecho, hacia el funcionamiento de interfaz para un usuario sin conocimiento entre circuitos cuánticos y hacia una mayor precisión tras el estudio de parámetros de cada algoritmo alternativo

Otra destacable diferencia de la clase `CreditRiskAnalysis` reside en la alta configuración que deja al usuario de cara a ejecutar sus análisis. Proporcionamos en el Apéndice B mayor detalle con respecto a los parámetros disponibles. De entre estos parámetros; `p_zeros`, `rhos` y `lgd` son los tres argumentos principales, determinando las características del portfolio a analizar. Estas tres variables consistirán en tres listas (de igual longitud K) que albergan, respectivamente:

- Las probabilidades iniciales de cada *asset* del portfolio (p_k^0).
- El parámetro de dependencia entre el activo k-ésimo y la distribución normal simulada (en el modelo GCI esto es $\rho_k \in [0, 1)$).
- El valor económico de cada activo, es decir, cada λ_k.

A continuación se muestra un sencillo ejemplo de código en el que podemos crear un portfolio de tres activos y llamar a un método a partir del cual representamos gráficamente y resolvemos todos los estadísticos planteados (modelo mixto, esperanza, función de distribución y valor de riesgo) mediante la MLAE:

Código 3.1: Ejemplo de construcción de un portfolio.

```
p_zeros = [0.25, 0.1, 0.3]
rhos = [0.1, 0.15, 0.05]
lgd = [2,4,1]

cra = CreditRiskAnalysis(p_zeros,rhos,lgd,backend="qasm_simulator",ae_algorithm = 'mlae',
    num_ml_queries = 6)

cra.run_all()
```

Más adelante se mostrará la salida de un método como es `cra.run_all()`, pero primeramente mostraremos cómo se han preparado las puertas lógicas y operadores principales mediante Qiskit para cada punto de los objetivos planteados previamente.

3.2.1. Flujo de Trabajo de los Métodos Implementados

Dividiremos esta sección por cada uno de nuestros cuatro objetivos. Mostraremos detalles con respecto a su implementación y para la ejecución de un usuario, si bien los resultados y algunos ejemplos de salidas se mostrarán más adelante, en la Sección 4.1.

Simulación cuántica y computación clásica: Modelo mixto

Recordemos que el objetivo de nuestro llamado modelo mixto consiste en comprobar que la simulación cuántica del modelo de incertidumbre $\mathcal{U}$ generado por el portfolio es correcta; así como para obtener valores de los estadísticos computados clásicamente para compararlos con los que se obtengan cuánticamente más adelante.

Apreciamos de nuestra librería la "granularidad" disponible de la información que requiera un usuario, donde cada problema a resolver dispone de diversos métodos dependientes entre sí (por ejemplo, entre `run_U` y `graph_U`). Mayor detalle de dichas dependencias se encuentra ubicado en el Apéndice B.

Para ilustrar los métodos utilizados para crear funciones relativas a cada operador, se muestra a continuación `build_U`. Se aprovecha la librería de Qiskit `qiskit.aqua` para generar el Modelo de Dependencia Condicional Gaussiana como sigue (se adjunta una línea con el `import` pertinente) [2].

Código 3.2: Método de construcción del modelo de incertidumbre.

```
from aqua.components.uncertainty_models import GaussianConditionalIndependenceModel as GCI

def build_U(self):
    # construct circuit factory for uncertainty model (Gaussian Conditional Independence model)
    self.u = GCI(self.n_z, self.z_max, self.p_zeros, self.rhos)
    # determine the number of qubits required to represent the uncertainty model
    self.num_qubits = self.u.num_target_qubits
    # initialize quantum register and circuit
    q = QuantumRegister(self.num_qubits, name='q')
    self.qc = QuantumCircuit(q)
    # construct circuit
    self.u.build(self.qc, q)
```

Comprendemos con este ejemplo la noción de cómo `CreditRiskAnalysis` sirve de interfaz entre lo disponible por Qiskit y un usuario sin un mínimo de conocimientos del funcionamiento de esta librería: Si bien por un lado disponemos de la capacidad de generar un GCI mediante Qiskit, un usuario debería controlar el hecho de agregarlo a un circuito con los registros pertinentes y correctamente enlazados. Conforme la complejidad de los operadores aumente, como en la ejecución de una QAE, podremos apreciar en mayor medida la dificultad de "ensamblar" cada operador entre sí.

Estimación de la pérdida esperada mediante algoritmos cuánticos: $E[\mathcal{L}]$

De forma análoga a cómo hemos definido distintos métodos para el modelo de incertidumbre etiquetados como `U`, la estimación, cálculo y graficación de la pérdida esperada del portfolio se etiqueta como `E` (en referencia al operador $E[\mathcal{L}]$).

Para construir el operador $\mathcal{S}$, se dispone en Qiskit de la factoría `WeightedSumOperator`. Creamos la variable `agg` (de *aggregate*) en el código 3.3.

[2] Se omitirá gran parte del código a lo largo de la explicación del mismo por fines ilustrativos, indicándose como "`#[...]`".

Código 3.3: Construcción mediante Qiskit del sumador ponderado.

```
from aqua.circuits import WeightedSumOperator

def build_E(self):
    # determine number of qubits required to represent total loss
    self.n_s = WeightedSumOperator.get_required_sum_qubits(self.lgd)
    # create circuit factory (add Z qubits with weight/loss 0)
    self.agg = WeightedSumOperator(self.n_z + self.K, [0]*self.n_z + self.lgd) # se añaden los
        qubits que se utilizaban para Z

    #[ . . . ]
```

Remarcamos en el código 3.4 la diferencia entre el operador $\mathcal{A}$ utilizado para estimar la esperanza frente al que se construya para $CDF_{\mathcal{L}}$. Procedemos así con el método de Qiskit Aqua `PwlObjective` (*Piecewise Linear Objective Function*). Este método construye una función lineal a trozos si el usuario especifica los *breakpoints* (puntos x donde se cambia de función), pendientes y *offsets*; necesaria para estimar la fase de la pérdida esperada.

Código 3.4: Construcción mediante Qiskit de la fase a estimar para hallar la pérdida esperada.

```
from aqua.components.uncertainty_problems import UnivariatePiecewiseLinearObjective as
    PwlObjective

def build_E(self):

        # [ . . . ]

        breakpoints = [0]      # puntos de cambio
        slopes = [1]           # pendientes de cada segmento lineal (e.d, a en ax + b)
        offsets = [0]          # desviación de cada segmento (e.d, b en ax + b)
        f_min = 0              # minimo valor de la funcion
        f_max = sum(self.lgd)  # maximo valor de la funcion
        c_approx = 0.25        # un factor de aproximacion, que da lugar a "comprimir" cada
            segmento

        objective = PwlObjective(
            self.agg.num_sum_qubits, # n de qubits necesarios
            0,                  # minimo valor que se introduce en la funcion
            2**self.agg.num_sum_qubits-1,  # maximo valor alcanzable por el registro de qubits
            breakpoints, slopes, offsets, f_min, f_max, c_approx
        )
```

Como última sección de código para comprender la unión entre la clase implementada y Qiskit, se muestra en la celda 3.5 el proceso de selección de entre los distintos estimadores de amplitud y su ejecución.

Código 3.5: Discernimiento entre las distintas estimaciones de amplitud para la pérdida esperada.

```
from qiskit.aqua.algorithms import AmplitudeEstimation
from qiskit.aqua.algorithms import IterativeAmplitudeEstimation
from qiskit.aqua.algorithms import MaximumLikelihoodAmplitudeEstimation

def print_qae_E(self):

    # [ . . . ]

    if self.ae_algorithm == 'iqae':
        self.ae = IterativeAmplitudeEstimation(epsilon=self.epsilon, alpha=self.alpha,
            a_factory=self.multivariate)
    elif self.ae_algorithm == 'mlae':
        self.ae = MaximumLikelihoodAmplitudeEstimation(self.num_ml_queries,a_factory=self.
            multivariate,objective_qubits=[self.u.num_target_qubits])
    else:
        self.ae = AmplitudeEstimation(self.num_eval_qubits, self.multivariate)
    self.qae_result = self.ae.run(quantum_instance=self.backend, shots=self.shots)
    self.estimated_expected_loss = self.qae_result['estimation']

    # [ . . . ]
```

Comentar, por último, que `graph_E` devolverá dos gráficas: La primera consiste en un "zoom" hacia el valor teórico, el estimado y el intervalo de confianza de la estimación, mientras que la segunda da una visión más general del portfolio y la distribución de sus pérdidas, añadiendo la esperanza estimada.

Representación gráfica de la función de distribución acumulada: $CDF_{\mathcal{L}}$

El hallazgo de la función de distribución acumulada de $\mathcal{L}$ realmente se trata de una tarea secundaria cuando pensemos en aplicar estos algoritmos cuánticos en portfolios "del mundo real". Esta opcionalidad se debe a que, si bien nos aporta bastante información sobre la variable aleatoria, construirla requiere de mayores tiempos de ejecución por tener que ejecutar una estimación de amplitud sobre cada x perteneciente al dominio de $\mathcal{L}$.

Apreciamos en la construcción del operador $\mathcal{A}$, en este caso con el comparador $\mathcal{C}$, cómo ha de variar en función de un argumento `x_eval`, que consiste en la x sobre la que hemos de estimar $CDF_{\mathcal{L}}(x)$. El comparador `Comparator` consiste en una clase previa que hace uso del `IntegerComparator` de Qiskit.

De entre nuestros métodos disponibles para graficar $CDF_{\mathcal{L}}$, se destaca `graph_cdf()` para conocer en detalle la función de distribución acumulada; tanto la teórica como la estimación junto a sus intervalos de confianza; así como una línea sobre el nivel de confianza α para entender sobre qué valor de x se situará el VaR_{α}. Aparte, se muestra el valor *delta* del test de Kolmogorov-Smirnov y su p-valor, una prueba estadística que indica la probabilidad de que ambas funciones de distribución pertenezcan a la misma variable aleatoria. Con ello, a mayor p-valor mayor precisión en los resultados.

Código 3.6: Construcción del circuito de estimación de la función de distribución acumulada.

```
from qiskit.circuit.library import IntegerComparator

def get_cdf_operator_factory(self,x_eval):
    if not hasattr(self, 'agg'):
        self.build_E()
    # comparator as objective
    cdf_objective = Comparator(self.agg.num_sum_qubits, x_eval+1, geq=False)
    # define overall uncertainty problem
    multivariate_cdf = MultivariateProblem(self.u, self.agg, cdf_objective) # A: U, S y C

    return multivariate_cdf
```

Búsqueda en bisección del valor de riesgo: $VaR_\alpha[\mathcal{L}]$

La creación de $\mathcal{A}$ y estimaciones de amplitud cuánticas sobre $CDF_\mathcal{L}$, si bien se puede utilizar como hemos hecho anteriormente para pasar a conocer la variable aleatoria por completo; se utilizará para realizar una búsqueda en bisección sobre las posibles pérdidas con tal de encontrar el valor a riesgo α. Destacamos que, procediendo de esta forma, pasamos a realizar en el peor de los casos $\mathcal{O}(\log(K))$ estimaciones de amplitud; luego obtenemos un valor significativo y explicativo para $\mathcal{L}$ en un mucho menor tiempo.

Con ello, un usuario podrá obtener el valor a riesgo α de un portfolio dado llamando a `print_var`. Podrá especificar un argumento de verbosidad con tal de conocer la ejecución de la búsqueda en bisección, aparte de comprobar la evolución de este proceso gráficamente mediante `graph_var`.

3.2.2. Cambios en la solución implementada con respecto a la existente

Hacemos acopio en esta sección de los distintos cambios conceptuales y prácticos que experimentó la clase `CreditRiskAnalysis`, justificados por habernos situado en un contexto tan dinámico como volátil; ya sea por la librería Qiskit Aqua como por los tutoriales de Qiskit sobre análisis de riesgos financieros.

Diferencias entre el código implementado y el tutorial de Qiskit

La clase `CreditRiskAnalysis` basa el cauce de su ejecución en el mismo que el tutorial de Qiskit `09_credit_risk_analysis` [6], sin embargo, son dos las diferencias principales a destacar.

La primera de ellas consiste en la falta de generalización a K portfolios por parte del tutorial. Si bien se ofrece la posibilidad de cambiar el portfolio inicial en una de las celdas, el código de computación de los resultados falla para diversos portfolios de incluso dos activos, ya sea en resultados como en errores de indexación que puedan aparecer, abriéndose una *Issue* en el Github de Qiskit Aqua en

Octubre de 2020 [12]. El tutorial fue actualizado días después, aunque no se han terminado de pulir los errores existentes en dicha generalización.

Un portfolio con `lgd = [2, 2]` es capaz de asertar esta diferencia, funcionando correctamente en nuestro código y fallando en el tutorial de Qiskit (errores de indexación en la computación de resultados para $\mathcal{U}$, amplitudes mal estimadas tanto para la esperanza como para la búsqueda en bisección).

Por otro lado, la segunda diferencia consiste en la creación de los circuitos a ejecutar. El tutorial de Qiskit actualizado pasó a crear los circuitos "a mano", en el sentido de ir añadiendo a un `QuantumCircuit` los operadores pertinentes. La clase `CreditRiskAnalysis`, sin embargo, hace uso de las conocidas como `CircuitFactory`, clases con métodos para construir dichos circuitos cuánticos. Si bien la `CircuitFactory` de Qiskit Aqua proporciona una forma más generalizada de construir circuitos y es el método que nos ha dado resultados satisfactorios, veremos a continuación que dicha forma de construir circuitos la propia librería la ha declarado como deprecada.

Diferencias entre el código implementado y la librería Qiskit Aqua

Como se ha mencionado anteriormente, las factorías de circuitos cuánticos de Qiskit se avisan como deprecadas por parte de Qiskit Aqua. Este hecho actualmente no afecta al código de la librería más allá de aparecer un *warning*, pero requerirá considerarse para un mantenimiento a largo plazo en el cambio del uso de estos circuitos. Sin embargo, podemos ver en un *branch* del Github de Aqua [13] cómo actualmente no existen sustitutos para las factorías de circuitos para problemas multivariable (como nuestro operador $\mathcal{A}$).

Para solucionar temporalmente esta deprecación, las dependencias de `CreditRiskAnalysis` se importan desde una copia de seguridad local de la versión `0.8.0` de Aqua (Aqua se sitúa actualmente en su versión `0.9.0`). En caso de no disponer de dicha copia de seguridad, se importa la versión más actual. Comentar además que no únicamente se encuentra deprecada esta parte de Qiskit Aqua, sino también Qiskit Aqua en sí, realizándose a fecha de Abril de 2021 una migración de sus funcionalidades hacia Qiskit Terra.

Destacamos también entre las diferencias del código implementado y Qiskit Aqua la evolución de disponer únicamente de la QAE a incluir en la clase las versiones iterativa y de máxima verosimilitud, presentadas anteriormente. No únicamente evolucionó el código por encontrarnos ante versiones más optimizadas y precisas, sino también por no funcionar la `AmplitudeEstimation` de Qiskit Aqua durante los primeros meses de codificación del trabajo [14].

4

Resultados de Ejecución en Simuladores y Ordenadores Cuánticos.

En este último capítulo previo a conclusiones pasamos a analizar nuestros resultados de ejecución de la clase `CreditRiskAnalysis`. Los dividiremos según el tipo de *backend* al que fueron enviadas las tareas de ejecución, ya sea en simuladores, donde veremos complejos ejemplos que muestran nuestra satisfactoria generalización a K activos, bien sea en ordenadores cuánticos reales. En estos últimos previamente haremos un estudio con respecto al tamaño de nuestros circuitos y la precisión de estas ejecuciones, comprendiendo cierto trabajo a futuro que haga obtener resultados análogos a los simulados para ordenadores reales.

4.1. Resultados de Ejecución en Simuladores. Ejemplo de Generalización a K activos

Las ejecuciones en simuladores, a diferencia de lo que mostremos más adelante en ordenadores cuánticos, proporcionan resultados de muy considerable precisión, recordemos generalizados a K activos de cualesquiera valores. El simulador sobre el que se ejecutarán los ejemplos es el `qasm_simulator`. Se procede a mostrar y comentar las gráficas con los resultados de dos ejemplos:

1.– El primer ejemplo consistirá en un portfolio mayor, de nueve activos, que mostrarán cómo este análisis ha sido generalizado con límites comprobados hasta tiempos computacionales asequibles. Se analizará utilizando la iQAE (`epsilon` y `alpha` por defecto).

2.– El segundo ejemplo corresponderá a un portfolio sencillo de tres activos, analizado usando la MLAE (con `num_ml_queries = 6`). De esta forma mostramos que hemos conseguido ejecutar análisis de portfolios financieros de más de dos activos, con resultados precisos.

4.1.1. Primer ejemplo: Portfolio de nueve activos con iQAE

Se muestra a continuación los parámetros del portfolio a analizar:

Este portfolio es una muestra de la generalización posible a K activos de la que provee nuestra clase `CreditRiskAnalysis`. Intentar ejecutar un portfolio similar al dado en los tutoriales de Qiskit Aqua resultaría en errores de ejecución y en resultados erróneos.

Activo (i)	Valor del activo (λ_i)	Probabilidad inicial (p_i^0)	Factor de dependencia (ρ_i)
0	2	0.15	0.1
1	2	0.25	0.05
2	4	0.1	0.1
3	1	0.3	0.2
4	6	0.05	0.25
5	5	0.1	0.15
6	3	0.2	0.1
7	5	0.15	0.05
8	1	0.3	0.1

Tabla 4.1: Datos del Portfolio del primer ejemplo

Ejecución del Modelo Mixto: $\mathcal{U}$

Procedemos en primer lugar a ejecutar nuestro modelo mixto, recordando nuestras razones tanto de obtener una información sobre la que comparar el resto de resultados como de analizar la "calidad" del *backend sobre el que ejecutamos*. Como habíamos adelantado previamente, son tres las gráficas mostradas tras llamar a `graph_U()`. La salida por pantalla es la siguiente:

```
-------- RESULTS OF U IN qasm_simulator --------
Expected Loss E[L]: 3.77636719
Value at Risk VaR[L]: 11.00000000
P[L <= VaR[L]]: 0.96679688
```

Vemos cómo la esperanza y el valor a riesgo se sitúan a lo largo de la distribución de pérdidas de $\mathcal{L}$ en la primera gráfica resultante.

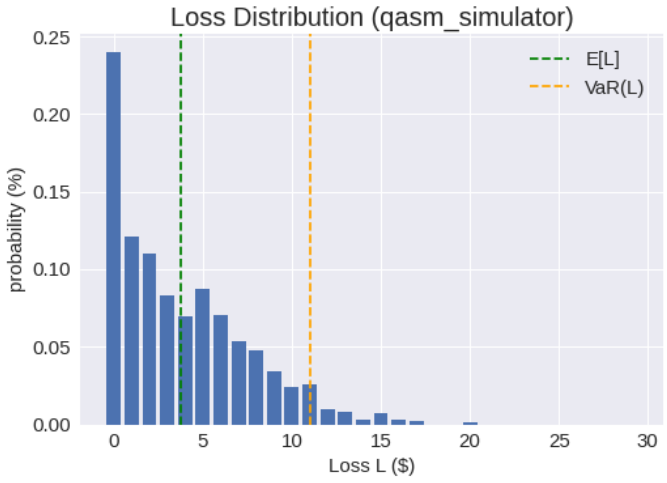
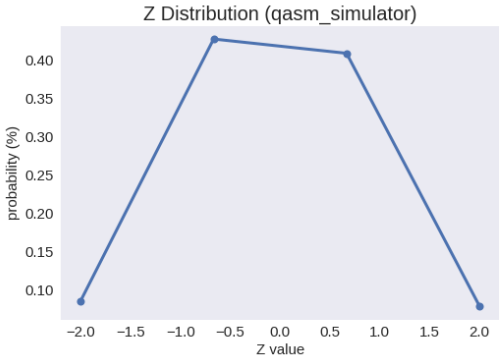


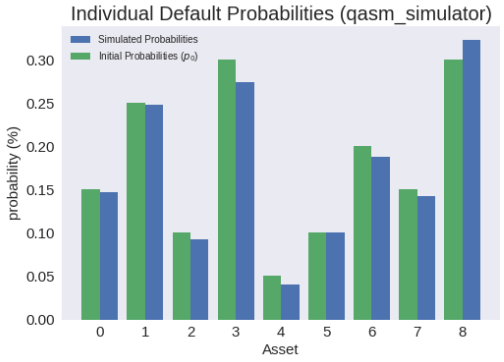
Figura 4.1: Distribución de pérdidas del portfolio del ejemplo primero según el modelo mixto.

Observamos (y adelantamos) que la mayoría de funciones de distribución de un modelo de pérdidas del GCI seguirá en parte una densidad parecida a la de una distribución exponencial.

Las siguientes dos gráficas que genera este método representan la simulación de la distribución de la normal estándar discretizada, así como una comparación entre las probabilidades simuladas de cada *asset* frente a las iniciales establecidas p_i^0.



(a) Simulación de Z en el primer portfolio ejemplo.



(b) Comparación entre probabilidades teóricas y simuladas en el primer ejemplo.

Figura 4.2: Análisis de la calidad del simulador con respecto a la generación del modelo de incertidumbre.

La muy leve asimetría de la distribución de Z nos da la información de situarnos en un *backend* donde el ruido no parecerá ser un problema. Esto es evidente para estas ejecuciones puesto que son simuladores, pero apreciaremos más adelante algunos *backends* cuyo ruido cuántico nos dificultará extraer resultados satisfactorios. La comparación entre probabilidades sirve como segunda forma para cerciorarnos de la calidad de la simulación. En caso contrario, en función del lado hacia el que se incline la asimetría de Z apreciaremos unas probabilidades mayores o menores frente a las iniciales.

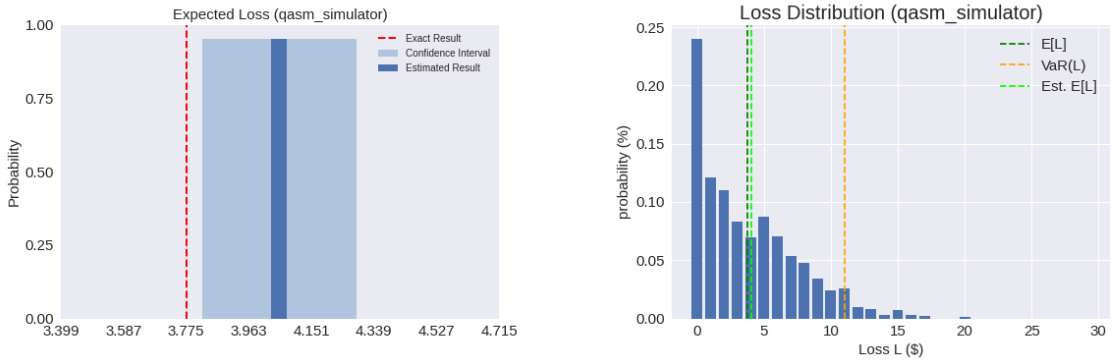
Ejecución de la QAE para la estimación de la esperanza: $E[\mathcal{L}]$

La salida por pantalla tras ejecutar `graph_E()` nos compara el valor de referencia anterior (el simulado de forma mixta, que nos atenemos en parte a la nomenclatura de tutorial y lo llamamos exacto [6]) frente al estimado mediante la iQAE y su intervalo de confianza.

```
--- RESULTS OF IQAE FOR E IN qasm_simulator ---
Reference value: 3.77636719
Estimated value: 4.05453145
Confidence interval: [3.82256534, 4.28649756]
```

Apreciamos un error cometido del orden de décimas. En nuestra primera gráfica, el *zoom* hacia la diferencia entre el valor exacto y el estimado, podremos dudar de la calidad de esta ejecución, pero

en la segunda gráfica, análoga a la primera de `graph_U()`, apreciaremos la alta proximidad entre estos dos valores. Importante comentar el hecho de que el valor simulado clásicamente (línea roja discontinua) no ha de ser el exacto teórico al simularse un número finito de veces, motivo por el cual no se tenga por qué entrar este valor en el intervalo de confianza. Esta observación sobre llamar "exacto" a los valores de referencia se aplica para el resto de resultados y gráficas del trabajo.



(a) Comparación entre valor exacto y estimado de la esperanza del primer portfolio.

(b) Distribución de las pérdidas indicando la esperanza estimada mediante QAE.

Figura 4.3: Comparación entre valor exacto y estimado en la esperanza del primer portfolio.

Función de distribución acumulada: $CDF_{\mathcal{L}}$

La ejecución de `graph_cdf()` nos informará del p-valor del test de Kolmogorov-Smirnov, y comprobaremos que las probabilidades de errar con respecto a la similitud entre la exacta y la estimada mediante una iQAE con `epsilon = 0.01` se va al orden de milésimas. Ocurre con la estimación de amplitud sobre un simulador que sus intervalos de confianza son excesivamente estrechos, luego apenas se podrán apreciar en la gráfica. Más adelante consultaremos ejecuciones donde se pueda apreciar una "banda" sobre la que se estiman situar los posibles valores de $CDF_{\mathcal{L}}(x)$.

Esta visualización de la función de distribución acumulada (Figura 4.4) nos adelanta el hecho de que el $VaR_{95\,\%}[\mathcal{L}]$ está en $x = 11$, al ser la x cuyo valor de $CDF_{\mathcal{L}}$ se sitúa más cerca y por encima del $95\,\%$.

Valor en riesgo: $VaR_{\alpha}[\mathcal{L}]$

Vemos en la salida por pantalla de `graph_var()` un error cometido del orden de centésimas con respecto a la probabilidad estimada del valor en riesgo resultante.

```
--- RESULTS OF IQAE FOR VaR IN qasm_simulator ---
Estimated Value at Risk: 11
Reference Value at Risk: 11
Estimated Probability: 0.95680988
```

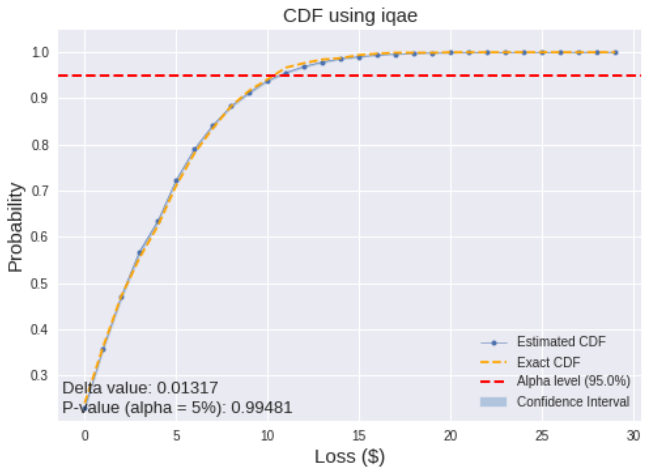
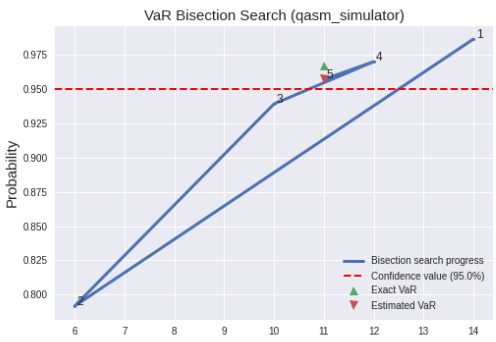


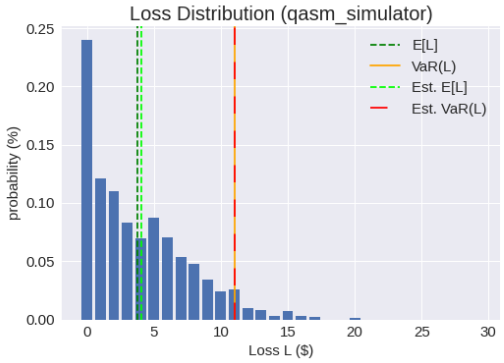
Figura 4.4: Función de distribución acumulada del portfolio primero.

```
Reference Probability: 0.96679688
```

Mientras que la segunda gráfica mostrada por esta función es análoga a las anteriores que mostraban la distribución de pérdidas, en la primera se muestra mediante la línea azul el recorrido que realiza la búsqueda en bisección (etiquetando cada punto con el orden en el que fue explorado), marcando finalmente el valor en riesgo estimado y exacto, a una altura que depende de la probabilidad de los mismos.



(a) Búsqueda en bisección del valor en riesgo del primer ejemplo.



(b) Distribución de las pérdidas del primer portfolio añadiendo el VaR estimado.

Figura 4.5: Búsqueda en bisección del valor en riesgo y vista general de las pérdidas del portfolio y sus estimaciones.

4.1.2. Segundo ejemplo: Portfolio de tres activos con MLAE

Incorporamos un segundo ejemplo, de menor tamaño, que realiza hincapié en la comparación entre nuestro código implementado frente al tutorial de Qiskit Aqua al ser el mismo que el de la *demo* si bien añadiendo un activo más que induciría a error en el tutorial.

Se adjunta en el Apéndice C una explicación de los resultados análoga a la del primer ejemplo. En dicho segundo portfolio, de tres activos como ejemplo de la frontera del tutorial base, es analizado mediante la MLAE.

4.2. Resultados de Ejecución en Ordenadores Cuánticos. Escalado hacia un problema real

Procedemos a avanzar las ejecuciones desde simuladores hacia ordenadores cuánticos reales. Veremos que, a día de hoy, analizar portfolios como los anteriores en un *backend* real supone realizar labores secundarias de limitación de la profundidad del circuito y de corrección de ruido. Estos análisis secundarios, llevados a cabo a partir del estudio del cambio de parámetros del constructor `CreditRiskAnalysis`, son debidos a las características actuales de los ordenadores cuánticos de IBMQ, con significativas limitaciones en comparación a un simulador.

Agradecemos al Hub-CSIC la posibilidad de acceso a ordenadores cuánticos más potentes de IBMQ, permitiéndonos haber podido analizar portfolios en el estado actual tecnológico.

4.2.1. Análisis del tamaño de circuitos

La primera realidad a la que nos enfrentamos al momento de querer escalar hacia un "problema del mundo real" un análisis de portfolios está en el excesivo tamaño de los circuitos conforme aumentamos el número de activos a analizar. Recordemos que el tamaño de un circuito (y del ordenador cuántico que lo alberga), se puede considerar en anchura (número de *qubits*) y profundidad (número de puertas utilizadas o tiempo de coherencia requerido).

Anchura de un circuito cuántico: Número de *qubits*

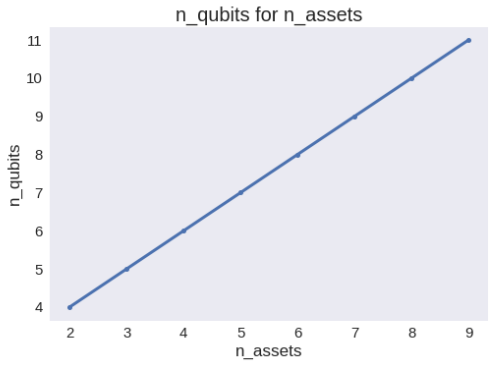
El número de *qubits* de nuestros circuitos cuánticos resultan no ser el mayor de los problemas, en el sentido de que experimentan un crecimiento lineal con respecto al número de activos ($\mathcal{O}(K)$). Dividimos este análisis en dos casos: Anchura del circuito para la ejecución del modelo mixto, y anchura del circuito en estimaciones de amplitud.

En el primero de estos casos, la anchura de $\mathcal{U}$ aumenta de uno en uno por cada activo que tenga

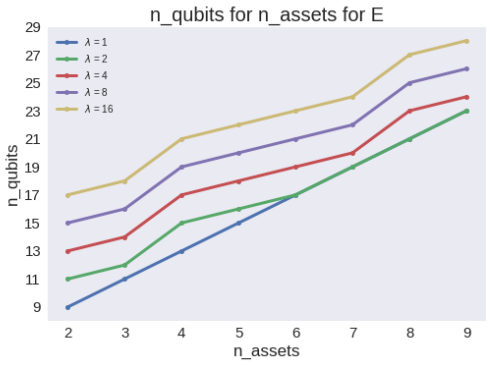
(Figura 4.6(a)), puesto que solo se necesita un *qubit* para representar cada distribución de Bernoulli $\{X_i\}_{i=1}^{K}$ correspondiente. Recordando que se establece un parámetro, n_Z, para utilizarlo como número de *qubits* destinados a representar la distribución normal, es fácil comprobar que el número de *qubits* para $\mathcal{U}$ es:

$$n_U = n_Z + K \tag{4.1}$$

Por otro lado, la anchura de $\mathcal{A}$ (el objetivo de las QAEs pertinentes), aumenta linealmente (Figura 4.6(b)), si bien el crecimiento de esta dependerá del número de *bits* necesarios para representar en binario el valor de cada λ_i, añadido los *qubit* ancilla necesarios para que la suma ponderada sea reversible. Cabe de esperar que podremos realizar "reglas de tres" sobre los valores reales de los activos con tal de poder reducir en anchura el circuito.



(a) Número de *qubits* por cada activo añadido en $\mathcal{U}$



(b) Número de *qubits* por cada activo añadido en $\mathcal{A}$

Figura 4.6: Análisis del aumento de anchura de los circuitos con respecto a cada activo nuevo añadido.

En conclusión, los circuitos que pretendemos ejecutar no suponen problema alguno "a lo ancho" en la mayoría de los casos. Es evidente que en *backends* como `ibmq_santiago` o `ibmq_vigo`, que no superan los cinco *qubits*, estos circuitos no podrán ser ejecutados y son rechazados al momento de intentar enviar la tarea al servidor, pero mediante otros servicios a los que se tiene acceso desde el CSIC, como `ibmq_manhattan` o `ibmq_toronto`, de 65 y 27 *qubits* respectivamente, se podrán ejecutar portfolios incluso mayores que los de nuestras simulaciones con respecto a la anchura.

Profundidad de un circuito cuántico

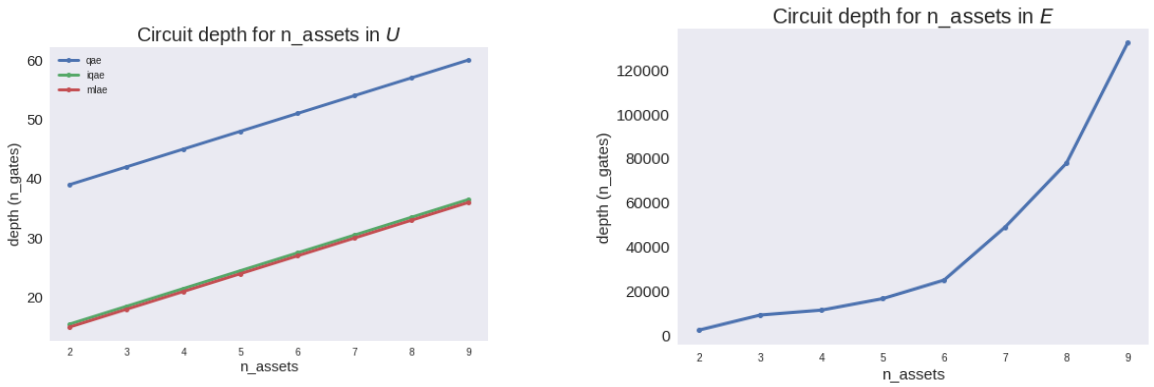
La profundidad de los circuitos creados y ejecutados resultan tener la mayor problemática con respecto a la capacidad de los ordenadores cuánticos reales utilizados. Una estimación de amplitud resulta ser un circuito, si bien de una anchura aceptable, de una exigencia en tiempos de coherencia alta. Esto se debe a que, incluso ya resultando larga una ejecución de $\mathcal{Q}$, sabemos que es necesario

además llegar a potencias del orden de $\mathcal{Q}^{2^k}$.

Ha sido bastante común a lo largo de la codificación y pruebas de nuestra clase `CreditRiskAnalysis` el siguiente error, mostrado tras intentar ejecutarse un trabajo enviado al servidor:

```
ERROR_RUNNING_JOB: Circuit runtime is greater than the device
repetition rate [8020]
```

Traduciendo el mensaje del error, nos cercioramos de que nuestros circuitos a ejecutar son excesivamente largos para la capacidad actual de los ordenadores cuánticos ofrecidos. Apreciamos la diferencia entre la linealidad de la profundidad con respecto al número de activos para simular $\mathcal{U}$ frente al crecimiento exponencial de $\mathcal{A}$ en la Figura 4.7, donde la profundidad de un circuito se conoce como el número de puertas lógicas de una base de puertas universales en ensamblador: rotaciones sobre los tres ejes $\{U_i\}_{i=1}^{3}$, la $CNOT$ y la identidad, a partir de las cuales se construye cualquier otra puerta lógica.



(a) Profundidad del circuito por cada activo añadido en $\mathcal{U}$. Se desglosa para los tres algoritmos de QAE disponibles

(b) Profundidad del circuito por cada activo añadido en $\mathcal{A}$ para la iQAE.

Figura 4.7: Análisis del aumento de profundidad de los circuitos con respecto a cada activo nuevo añadido.

Analizando el desglose de la Figura 4.7(a), explicamos las tres decisiones a tomar con tal de conseguir ejecutar nuestros circuitos en *backends* reales:

1.– En primer lugar, a día de hoy, descartar las ejecuciones en ordenadores reales utilizando la QAE "primitiva" como algoritmo. El hecho de ejecutar simultáneamente las potencias $\mathcal{Q}^{2^k}$ hacen exigir unos tiempos de coherencia actualmente inviables. Además, como se indicó en el capítulo anterior, el hecho de que no haya recibido este algoritmo el mantenimiento adecuado por parte de Qiskit Aqua, hace que, en el supuesto de conseguir ejecutar circuitos, los resultados no sean nada satisfactorios.

2.– Por otro lado, de la iQAE se ha realizado un exhaustivo análisis con respecto a sus parámetros `epsilon` y `alpha`, consiguiendo una serie de valores a partir de los cuales la profundidad del circuito se ha visto significativamente reducida sin por ello afectar en exceso a la precisión de la ejecución. Veremos que hemos conseguido resultados para la iQAE, pero finalmente por motivos de ruido estos han tenido que desecharse.

3.– Por último, si bien la MLAE no es un algoritmo tan recomendado por parte de Qiskit Aqua como la iQAE, es con el cual hemos podido ejecutar con menores problemas. Veremos más adelante que, pese a reducir los problemas de ruido cuántico, la deprecación de las `CircuitFactory` parecen ser la causa de una especie de *bias* en el error cometido, que induce a resultados no del todo satisfactorios.

Con ello, pasamos a continuación a mostrar los resultados del análisis realizado sobre la iQAE y a mostrar las ejecuciones de la MLAE.

4.2.2. iQAE: Análisis de precisión y profundidad

La estimación de amplitud cuántica iterativa, explicada en la Sección 2.1.3, recordemos que consiste en una QAE que, en un bucle controlado por la precisión de la estimación, busca con distintas potencias del operador $\mathcal{Q}^k$ la k que aporte el resultado satisfactorio. Con ello, variar los parámetros `epsilon` y `alpha` supone variar la k encontrada para un mismo circuito y por lo tanto variar la profundidad del mismo.

Adelantamos las consecuencias de aumentar o disminuir los dos parámetros que podemos indicar al construir un objeto de la clase `CreditRiskAnalysis`:

1.– `epsilon` ($\epsilon \in [0{,}01, 0{,}5]$): Recordemos que indica la precisión exigida con respecto a la fase, no con respecto a la amplitud. A mayor *epsilon*, menor será la potencia de k encontrada, pero el resultado irá distando con mayor probabilidad con respecto al exacto.

2.– `alpha` ($\alpha \in [0{,}01, 1)$): Indica el opuesto a la confianza del intervalo de amplitudes resultante $(1-\alpha)$. Con ello, a mayor *alpha*, menor será el intervalo de confianza por aumentar lo probabilidad de salir del mismo; traduciéndose en una mayor exigencia sobre la precisión.

Las estimaciones de amplitud a partir de las cuales se ha realizado este análisis consiste en la obtención de $CDF_{\mathcal{L}(x)}$, donde $x = VaR_{5\,\%}[\mathcal{L}] - 1 = 7$ [1] y $\mathcal{L}$ la variable aleatoria resultante de analizar mediante el modelo GCI el siguiente portfolio financiero:

Activo (i)	Valor del activo (λ_i)	Probabilidad inicial (p_i^0)	Factor de dependencia (ρ_i)
0	2	0.15	0.1
1	1	0.2	0.05
2	6	0.3	0.15

Tabla 4.2: Portfolio de estudio de la profundidad de la iQAE según parámetros ϵ y α.

Comentamos por último que cambiar el valor de la x sobre la que estimar o el portfolio en el que se estudia dará indistintamente unos resultados análogos a los que vamos a observar y analizar.

[1] Se toma una x previa justo al valor de riesgo para comprender la importancia de una estimación de amplitud precisa. Teniendo unos valores exactos alrededor del $0{,}9475$, apreciamos que con intervalos de confianza excesivamente "holgados" nos inducimos al error de establecer un valor de riesgo menor del correspondiente.

Variación de precisión y profundidad con respecto a *epsilon*

Se muestran cuatro gráficas en la Figura 4.8 que nos cercioran de lo adelantado anteriormente con respecto a este parámetro.

Destacamos varias conclusiones a partir de estos resultados:

1.– En primer lugar, nos cercioramos de la correspondencia 1-1 entre la potencia k-ésima adecuada para estimar el operador $\mathcal{Q}$ y la profundidad del circuito, luego no parece haber causas secundarias que afecten a la profundidad de la iQAE.

2.– Por otro lado, la apreciación de la reducción significativa en la profundidad a partir de un `epsilon` por encima de $0{,}05$, donde el error requerido para la fase pasa a ser tan "holgado" que en la primera iteración del bucle, con $\mathcal{Q}$ únicamente, nos basta para estimar la amplitud.

3.– Por último con respecto a profundidad, explicamos la no linealidad entre profundidad y `epsilon` dado por el hecho de que, al estar comparando fases en la circunferencia goniométrica, ciertos valores de k pueden "encajar" mejor en el mallado realizado y por ende no proseguir con la búsqueda.

4.– Con respecto a las precisiones, vemos claramente cómo el cambio en la caída de profundidad para $\epsilon \geq 0{,}05$ se ha de "pagar" con una caída significativa en la precisión del intervalo de confianza.

5.– Finalmente, apreciamos que la diferencia entre el *epsilon* exigido y el resultante (*actual epsilon* en la Figura 4.8(d)) se mantiene constante a partir del $0{,}025$, justo la mitad del umbral observado, debido a que la condición del bucle *while* de la iQAE depende de 2ϵ.

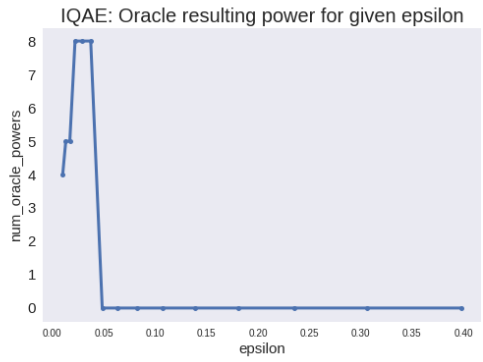
Surgen dos ideas concluyentes con respecto a la variación de *epsilon* en caso de querer ejecutar nuestras estimaciones de amplitud iterativas en ordenadores reales. La primera idea consiste en mantener $\epsilon = 0{,}01$ para no dañar la precisión resultante, si bien adelantamos que con este valor no podemos actualmente ejecutar circuitos. La segunda opción, causa de unos resultados no convenientes, está en establecer $\epsilon = 0{,}05$, donde el tamaño del circuito será el menor posible, pero ya conocemos de adelanto la excesiva longitud del intervalo de confianza dado.

Variación de precisión y profundidad con respecto a *alpha*

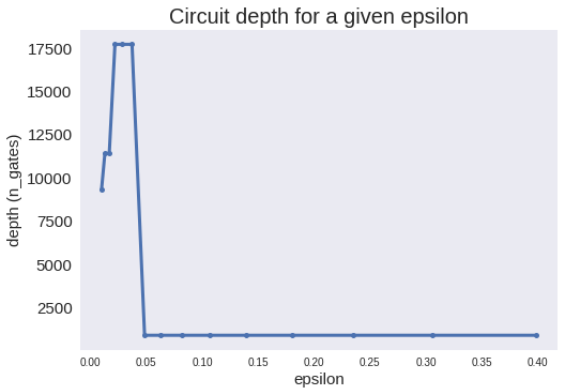
Mostramos gráficas con el mismo concepto de las de la sección anterior para comentar valores óptimos de *alpha*.

Son varios los puntos a destacar, análogos a los observados en la sección anterior. Primeramente, nos cercioramos una vez más de la correspondencia entre $\mathcal{Q}^k$ (Figura 4.9(a)) y la profundidad del circuito (Figura 4.9(b)), esta vez con un trazado que parece ser más irregular a primera vista, pero que se explica al compararlo con $x + sin(x)$ y recordar esa búsqueda del valor sobre la circunferencia goniométrica.

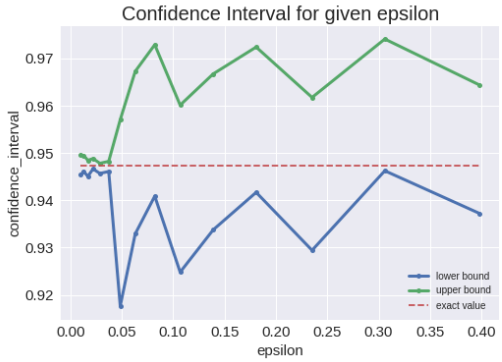
Por otro lado, observamos lo adelantado con respecto a la precisión del intervalo de confianza. Esto parece ser contradictorio con respecto a lo aprendido en estadística sobre los niveles de confianza, pero que se explica observando el código fuente de los métodos utilizados por Qiskit Aqua para devolver estos intervalos al calcular un *epsilon* inversamente proporcional a *alpha* [15][método `_chernoff_confint`].



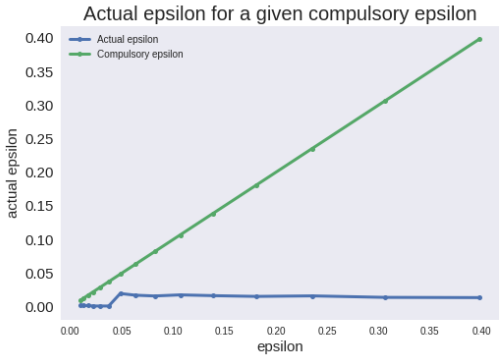
(a) Potencia de k resultante para una misma $\mathcal{Q}$ (oráculo) en función de ϵ para la iQAE.



(b) Profundidad del circuito de la iQAE en función del ϵ dado.



(c) Evolución del intervalo de confianza para una estimación de amplitud con iQAE con la variación de ϵ.



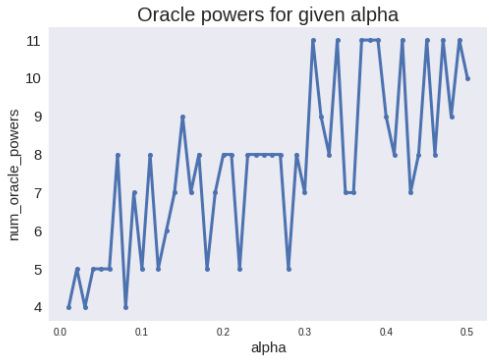
(d) Error en la fase resultante con respecto al pedido (*actual epsilon*)

Figura 4.8: Análisis de precisión y profundidad de la iQAE con respecto a la variación de *epsilon*.

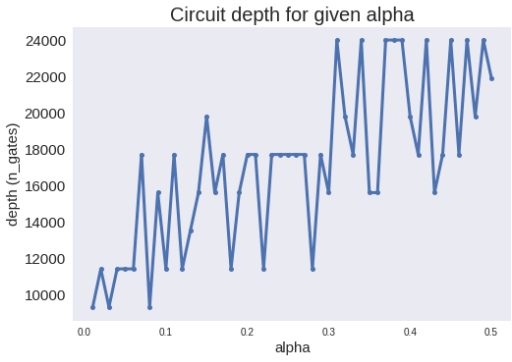
Concluimos, por lo tanto, en el hecho de que podremos reducir `alpha` a valores menores para conseguir una reducción sobre la profundidad del circuito, a la par que no parece verse tan afectada la precisión.

iQAE: Resultados parciales

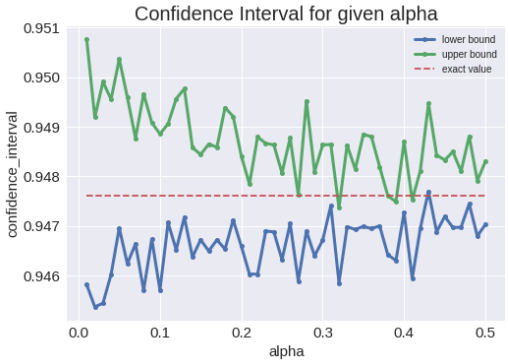
En esta subsección vemos mediante resultados cómo el ruido cuántico acaba afectando significativamente a nuestras soluciones, más cuando hemos tenido que realizar un ajuste en los parámetros `alpha` y `epsilon` para poder ejecutar los circuitos correspondientes a cambio de perder en precisión. Las siguientes gráficas han sido generadas tras ejecutar en el *backend* de IBMQ más grande disponible para usuarios cualesquiera. Este ordenador es el `ibmq_16_melbourne` (15 *qubits*, volumen cuántico de 8), que "sacrifica" su capacidad en anchura por unos tiempos de coherencia más bajos de lo habitual. Más adelante, para la MLAE, ejecutaremos en *backends* de IBM provistos para usuarios del CSIC.



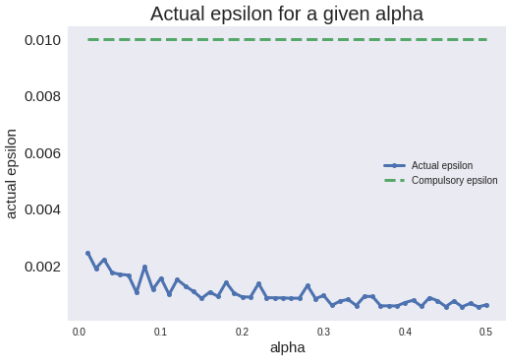
(a) Potencia de k resultante para una misma $\mathcal{Q}$ en función de α para la iQAE.



(b) Profundidad del circuito de la iQAE en función del α dado.



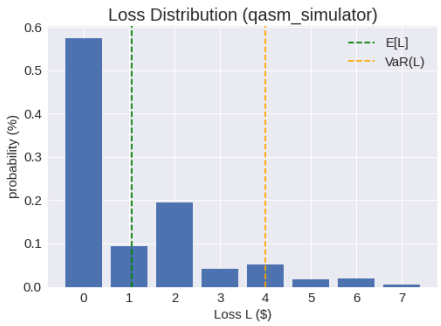
(c) Evolución del intervalo de confianza para una estimación de amplitud con iQAE con la variación de α.



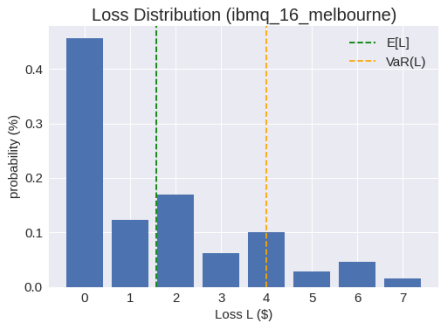
(d) Error en la fase resultante con respecto al pedido (*actual alpha*)

Figura 4.9: Análisis de precisión y profundidad de la iQAE con respecto a la variación de *alpha*.

Nuestro portfolio a analizar es el mismo que el desarrollado en nuestro Ejemplo 2 de la Sección 4.1.1 (Apéndice C). Primeramente comparamos su ejecución del modelo mixto frente a la de Melbourne en la Figura 4.10.



(a) Distribución del portfolio según `qasm_simulator`



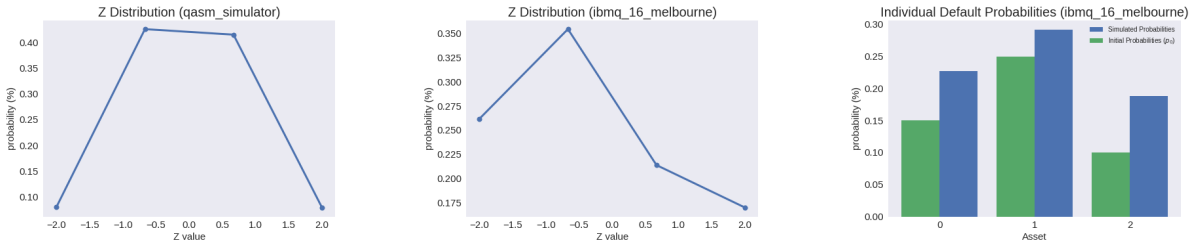
(b) Distribución del portfolio en `ibmq_16_melbourne`.

Figura 4.10: Comparación de ejecuciones del modelo mixto entre el simulador y un *backend* real.

Apreciamos que ambas formas de la distribución de $\mathcal{L}$ son similares, si bien los valores parecen acentuarse menos en el caso de Melbourne. Vemos que esto sucede por la cantidad de ruido cuántico que distorsiona lo que debería ser una distribución normal (Figura 4.11(a)), y cómo ello afecta a las probabilidades de cada activo.

Aunque este exceso de ruido cuántico no parezca haber afectado mucho a la creación de $\mathcal{U}$, recordemos que este modelo mixto es el que menores tiempos de coherencia requiere y, por lo tanto, el que menor problema supone en todo lo analizado previamente. Se muestra a continuación en la Figura 4.12(a) cómo dista una simple estimación de amplitud por iQAE de $E[\mathcal{L}]$ del valor teórico y cómo queda una gráfica de una función de distribución acumulada calculada en Melbourne en la Figura 4.12(b).

Estos resultados tan distantes de lo esperado nos hace replantearnos los algoritmos a utilizar y, como adelantamos al inicio de este capítulo, "aparcar" actualmente el uso de la iQAE con nuestros parámetros estudiados y personalizados. Pasamos así a nuestro último algoritmo disponible y al intento de obtención de ejecuciones satisfactorias.

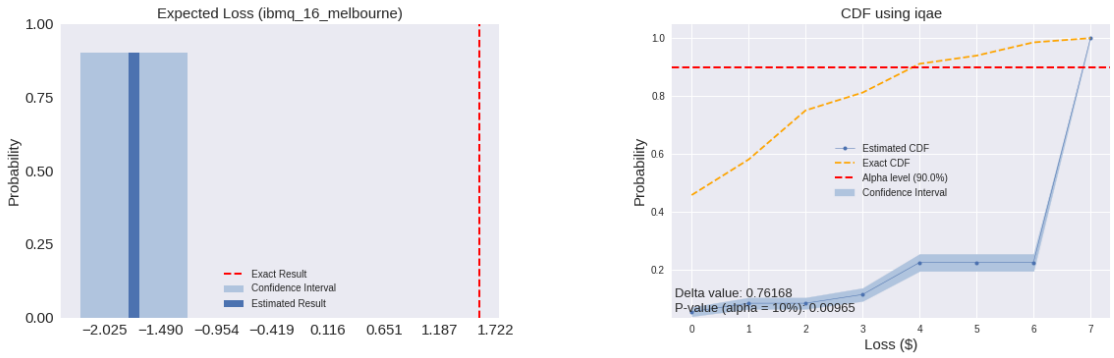


(a) Distribución normal Z simulada por el `qasm_simulator`

(b) Distribución normal Z ejecutada en `ibmq_16_melbourne`

(c) Probabilidades p_i^0 de cada activo en `ibmq_16_melbourne`

Figura 4.11: Análisis de la calidad de la distribución normal Z simulada y de las p_i^0 iniciales.



(a) Estimación de E

(b) Estimación de CDF

Figura 4.12: Estimaciones de amplitud para la esperanza del portfolio y para la función de distribución acumulada resultantes de `ibmq_16_melbourne`

4.2.3. MLAE: Ejecución en ordenadores cuánticos

La *Maximum Likelihood Amplitude Estimation*, introducida en la Sección 2.1.3, presenta dos ventajas frente a la iQAE en este contexto actual de búsqueda de circuitos ejecutables y precisos. Ambas ventajas tienen como causa el hecho de tener fijas las potencias de $\mathcal{Q}^k$ que ejecutar y analizar, puesto

que supone en primer lugar poder forzar la certeza de elevar a potencias que mantengan tiempos de coherencia factibles y, en segundo lugar, que posibles errores de ruido no den lugar a pasar a otras potencias en función del error comentido, como ocurre en el bucle *while* de la iQAE.

Con ello, conseguimos analizar portfolios mediante la ejecución de la MLAE en ordenadores cuánticos reales. Analizaremos cómo el modelo mixto se ejecuta sin problema alguno, se consigue una QAE para la esperanza que no se desvía en exceso del resultado teórico, y analizaremos cuáles serían las posibles causas de los errores en la estimación de $CDF_{\mathcal{L}}$ y del $VaR_{\alpha}[\mathcal{L}]$.

A lo largo de la siguiente sección compararemos tres *backends* provistos por IBMQ: `ibmq_paris`, `ibmq_manhattan` e `ibmq_toronto`. Estos tres ordenadores cuánticos son de 27, 65 y 27 *qubits* respectivamente, mientras que sus volúmenes cuánticos (una métrica de IBM para clasificar ordenadores según sus *qubits* y tiempos de coherencia) son de 32. El portfolio a analizar es el siguiente:

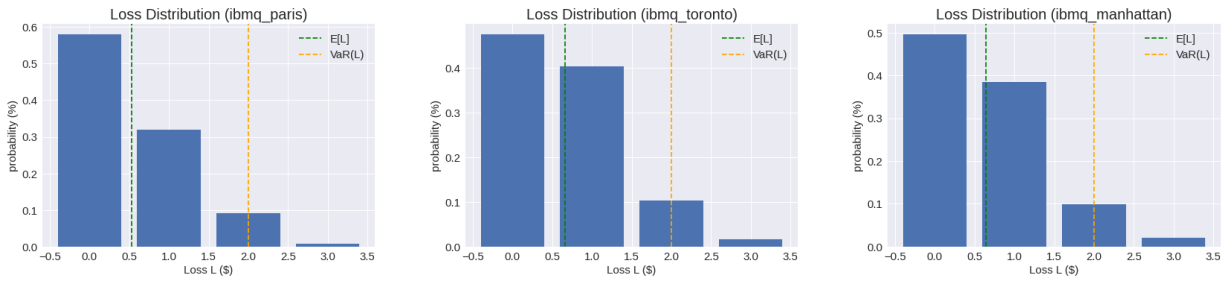
Activo (i)	Valor del activo (λ_i)	Probabilidad inicial (p_i^0)	Factor de dependencia (ρ_i)
0	1	0.25	0.1
1	1	0.1	0.15
2	1	0.15	0.2

Tabla 4.3: Portfolio analizado por ordenadores cuánticos reales

Comparación de resultados del modelo mixto

Si bien sabemos que en el modelo mixto el algoritmo de estimación de amplitud usado no influye en los resultados, puesto que calculamos clásicamente los valores que posteriormente se estiman mediante algoritmos cuánticos, dedicamos esta subsección a analizar la calidad de los tres *backends* propuestos. Apreciamos de esta forma en la Figura 4.13 las diferencias entre las distribuciones calculadas de una misma variable aleatoria, como causa del ruido cuántico de los tres *backends* utilizados.

Apreciamos en los resultados que varía en $\pm 0{,}1$ el valor de $P(\{\mathcal{L}=0\})$, diferencia que parece arrastrarse hacia $P(\{\mathcal{L}=1\})$. Pese a estas variaciones, no se ve lo suficientemente afectado el valor teórico de la esperanza ni del valor en riesgo.

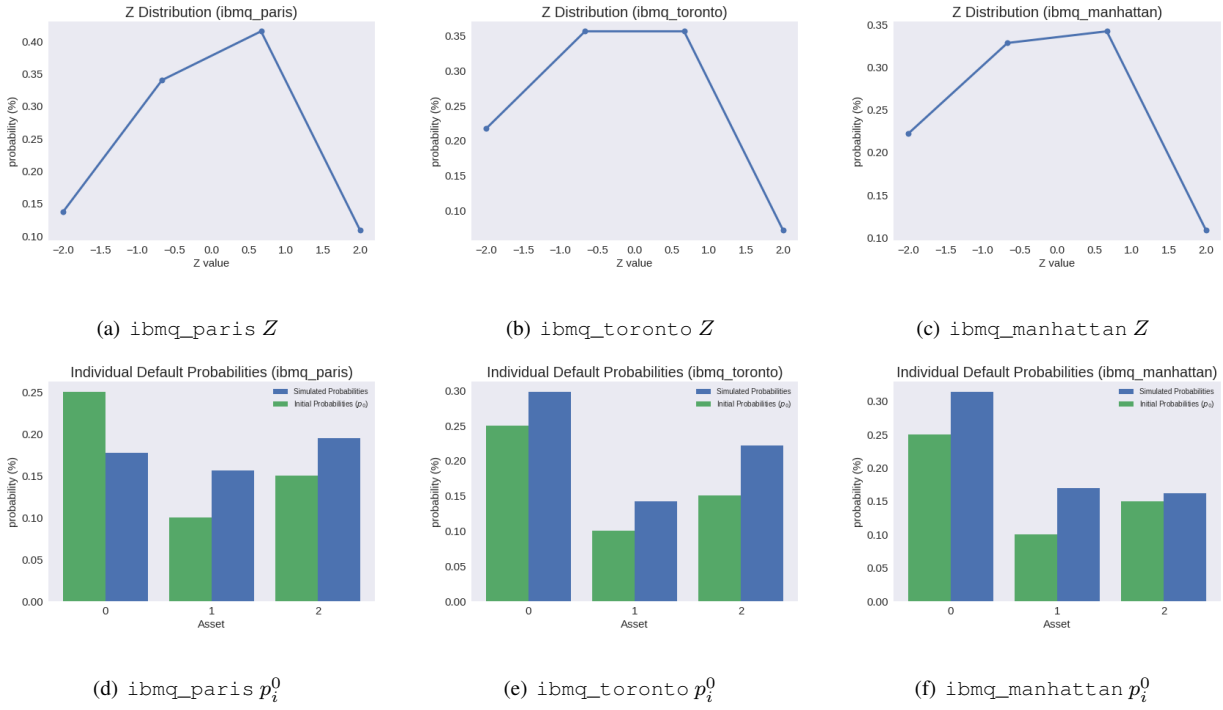


(a) `ibmq_paris` (b) `ibmq_toronto` (c) `ibmq_manhattan`

Figura 4.13: Análisis de resultados para el modelo mixto en los diferentes *backends* de IBMQ.

A continuación, en la Figura 4.14 observamos las diferencias causadas por el ruido cuántico de cada *backend* en la simulación de la distribución de Z y en cómo dista cada probabilidad de cada *asset* frente a la teórica establecida.

Concluimos con varias ideas con respecto a estas ejecuciones en *backends* de mayor potencia y capacidad. La primera de las ideas está en el hecho de que, si bien las simulaciones mixtas carecen de aplicación cuando se escale hacia problemas "del mundo real", funcionan correctamente en diversos ordenadores cuánticos pese a pequeñas variaciones de ruido y esto tiene diversas aplicaciones tanto ilustrativas como educativas. Por otro lado, podemos considerar que el nivel de ruido presente en nuestros tres *backends* presentados es bastante similar, ya sea por los resultados de esperanza y valor en riesgo parecidos como por presentar diferencias del mismo calibre entre las p_i^0. Apreciamos que es bastante usual un cierto *bias* en nuestros n_Z *qubits* de Manhattan y Toronto de medirse como $|00\rangle$, motivo por el cual el valor $-2{,}00$ es bastante más probable frente al $2{,}00$, luego destacamos de París ser el *backend* con menor ruido cuántico.



(a) `ibmq_paris` Z (b) `ibmq_toronto` Z (c) `ibmq_manhattan` Z

(d) `ibmq_paris` p_i^0 (e) `ibmq_toronto` p_i^0 (f) `ibmq_manhattan` p_i^0

Figura 4.14: Análisis de la calidad de la incertidumbre simulada en los diferentes *backends* de IBMQ.

Análisis de resultados de la MLAE para la estimación de la esperanza

Mientras que para el modelo mixto hemos acusado las diferencias entre valores obtenidos y teóricos al ruido cuántico, un problema que acaece sobre la mayoría de experimentos cuánticos actuales, llega un punto en el que se ha de ubicar un error fuera de estos motivos. Este hecho ocurre en el momento en el que apreciamos en la Tabla 4.4 que las tres estimaciones de amplitud por MLAE dan lugar a un mismo valor que dista entre sí por órdenes de centésimas.

Backend	Valor de Referencia	Valor Obtenido
`ibmq_paris`	0.6611	1.4279
`ibmq_toronto`	0.5273	1.4221
`ibmq_manhattan`	0.6435	1.4166

Tabla 4.4: Análisis de resultados para la estimación de $E[\mathcal{L}]$ en los diferentes *backends* de IBMQ.

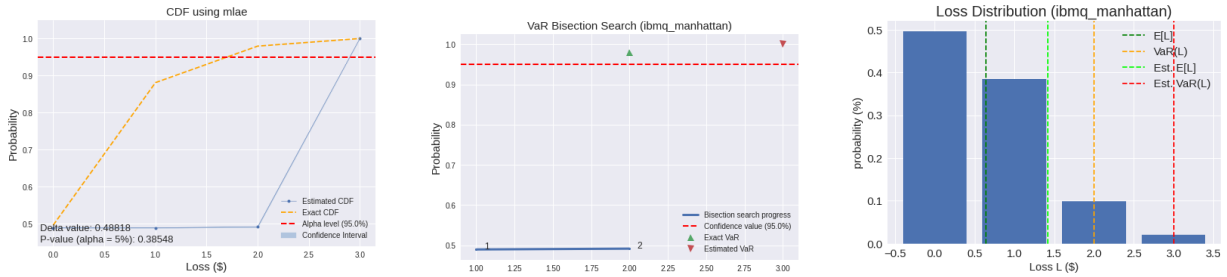
Tal y como mencionamos en la Sección 3.2.2, las diferencias entre nuestra clase `CreditRiskAnalysis` frente a las del actual tutorial de Qiskit probablemente albergarán las causa de este *bias* que desplaza la esperanza de $\mathcal{L}$ estimada en un error $\epsilon \sim 0{,}9$. Sin embargo, la dificultad de implementaciones de estos circuitos "a mano"; así como tomar de referencia el tutorial actual de Qiskit nos dificulta considerablemente generalizar nuestros portfolios a K activos, hace reconocer esta posible causa del error y considerarlo como la futura mejora de esta librería de mayor urgencia.

Análisis de resultados de la MLAE para el valor en riesgo

Mediante tres gráficas de la Figura 4.15 explicamos cómo los errores encontrados en las estimaciones de amplitud de la MLAE, comentados en la sección anterior, da lugar a estimar un VaR erróneo frente al simulado clásicamente. Recordemos que las siguientes ejecuciones se muestran en Manhattan, si bien en el resto de los *backends* los resultados han sido semejantes.

En primer lugar, observamos en la Figura 4.15(a) cómo la función de distribución acumulada parece quedarse casi constante, causa análoga a la de las esperanzas estimadas erróneamente. El último valor posible para x se establece a 1 desde la librería (ahorramos una estimación de amplitud obvia, puesto que $CDF_{\mathcal{L}}(\sum \lambda_i) = 1$ para todo portfolio a analizar).

A continuación, vemos cómo ese error se traslada a la búsqueda en bisección del VaR en la Figura 4.15(b), donde el VaR de referencia se ubica en $x = 2$ pero nosotros lo establecemos en el último valor posible, donde valía 1. Podemos ver un resumen de nuestras desviaciones realizadas en las estimaciones, tanto de la esperanza como del valor a riesgo en la Figura 4.15(c).



(a) Función de distribución acumulada (b) Búsqueda en bisección del VaR (c) Resultados finales.

Figura 4.15: Análisis de la estimación de $VaR_\alpha[\mathcal{L}]$ en `ibmq_manhattan`

5

Conclusiones y Trabajo Futuro

Concluimos esta memoria enumerando las aportaciones que ha realizado este trabajo, relacionándolas con los objetivos planteados en la Sección 1.3:

- Se ha analizado el código aportado por la *demo* de nuestro *paper* de referencia [6] y se ha implementado partiendo de esta base una clase, `CreditRiskAnalysis`, cuyas diferencias y ventajas volvemos a resaltar:
  1.– Muy importante, la capacidad de generalización del portfolio a K activos. Capacidad que se ha demostrado faltar en el tutorial dado previamente [12].
  2.– La alta parametrización disponible para el usuario: *backends*, *epsilon*, *alpha*, datos sobre el portfolio a analizar, algoritmo a utilizar (QAE, iQAE o MLAE)...
  3.– El hecho de actuar como interfaz esta clase entre un usuario que desconozca sobre la computación cuántica y un portfolio a analizar mediante este método.
  4.– Disponer de un sumario gráfico más avanzado y detallado frente a lo actualmente aportado por el tutorial: Gráfica de la CDF, intervalos de confianza de las AE, búsqueda en bisección del VaR...
- En segundo lugar, con respecto a los objetivos primero y segundo, se ha reunido suficiente base teórica como para comprender y analizar exhaustivamente nuestro *paper* de referencia sobre análisis de riesgo de crédito mediante avanzados algoritmos cuánticos [4].
- En tercer lugar, como parte del cuarto objetivo completado, se han ejecutado análisis realizados por nuestra clase ejecutados en simuladores y se ha demostrado tanto su generalización a K activos como su alta precisión obtenida.
- Por último, completando nuestro quinto objetivo, tras realizar un detallado estudio sobre el tamaño de nuestros circuitos y ordenadores cuánticos reales y cómo distintos parámetros afectan a la profundidad, se ha mostrado el estado actual de nuestras ejecuciones en *backends* reales para comprender la línea de progreso sobre la que se sitúa nuestro trabajo futuro.

Si bien queda como trabajo futuro la adaptación de nuestros estimadores de amplitud a las últimas versiones de Qiskit Aqua con tal de obtener resultados reales satisfactorios, consideramos haber cometido un significativo avance dado por la implementación de la clase `CreditRiskAnalysis`, especialmente por su generalización a K activos de un portfolio (de la cual recordemos carece la solución actual implementada por Qiskit [6]).

En conclusión, una introducción al álgebra compleja, la manipulación de *qubits* mediante circuitos, conocer algoritmos de estimaciones de amplitud y conceptos en matemática financiera y estadística nos han conducido a una implementación ejecutable en ordenadores cuánticos. La clase, `CreditRiskAnalysis`, volviendo al símil de nuestra introducción con respecto a la década de 1930, nos lleva a analizar lo que suponen estos resultados en potencia (nuestras simulaciones), frente a la realidad a la que nos encontramos causada por los ordenadores cuánticos disponibles actualmente; motivación principal para nuestro trabajo futuro a realizar.

Bibliografía

[1] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, Oct 2019.

[2] M. A. N. . I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[3] M. Rutkowski and S. Tarca, "Regulatory capital modelling for credit risk," *International Journal of Theoretical and Applied Finance*, vol. 18, 12 2014.

[4] D. J. Egger, R. Garcia Gutierrez, J. Cahue Mestre, and S. Woerner, "Credit risk analysis using quantum computers," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[5] S. Woerner and D. Egger, "Quantum risk analysis," *npj Quantum Information*, vol. 5, 12 2019.

[6] IBM, "Qiskit credit risk analysis tutorial." `https://qiskit.org/documentation/tutorials/finance/09_credit_risk_analysis.html`, 2021.

[7] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," *AMS Contemporary Mathematics Series*, vol. 305, 06 2000.

[8] D. Grinko, J. Gacon, C. Zoufal, and S. Woerner, "Iterative quantum amplitude estimation," 12 2019.

[9] Y. Suzuki, S. Uno, R. Raymond, T. Tanaka, T. Onodera, and N. Yamamoto, "Amplitude estimation without phase estimation," *Quantum Information Processing*, vol. 19, 01 2020.

[10] L. Grover and T. Rudolph, "Creating superpositions that correspond to efficiently integrable probability distributions," 09 2002.

[11] M. Soto Jiménez, "Repositorio `TFG_Informatica_CreditRiskAnalysis`." `https://github.com/SrSoto/TFG_Informatica_CreditRiskAnalysis`, 2021.

[12] M. Soto Jiménez, "Qiskit aqua github: Indexerror in credit risk analysis tutorial." `https://github.com/Qiskit/qiskit-aqua/issues/1360`, 2020.

[13] J. Gacon, "Qiskit aqua github: Deprecate the circuitfactory and derived types." `https://github.com/Qiskit/qiskit-aqua/pull/1348`, 2020.

[14] S. Wood, "Qiskit aqua github: Ae result has changed." `https://github.com/Qiskit/qiskit-aqua/pull/1348`, 2020.

[15] Q. Aqua, "Qiskit aqua github: Iterativeamplitudeestimation source code." `https://qiskit.org/documentation/_modules/qiskit/aqua/algorithms/amplitude_estimators/iqae.html#IterativeAmplitudeEstimation`, 2021.

[16] IBM, "Ibmq experience. página principal." `https://quantum-computing.ibm.com/`, 2018.

Apéndices

A

Introducción a la Computación Cuántica

Se proporciona este capítulo del anexo a un lector que necesite predisponerse e introducirse a la computación cuántica previo a entrar en algoritmos avanzados como la QAE.

La "ruta" a trazar a lo largo de este apéndice consistirá en comprender la ventaja que supone utilizar el *qubit* como unidad básica computacional frente al *bit*. A continuación, tras introducir notación matemática para manipular estos *qubits*, comprenderemos lo que es una puerta lógica cuántica y cómo se construye un circuito. Por último, especificaremos de estos circuitos los dos necesarios para comprender el funcionamiento del algoritmo cuántico principal a utilizar, llegando finalmente a enlazar con el análisis de riesgos financieros. La primera parte de este recorrido, previa a especificar aplicaciones de algunos algoritmos, realiza un resumido seguimiento de [2].

A.1. El *Qubit*. Definición, propiedades y diferencias frente al *bit*

Previamente a entender lo que es un *qubit*, es preciso recordar la definición de *bit* como *"unidad básica computacional que puede tomar el valor 0 o 1"*. Cursos impartidos a lo largo de la carrera han servido para comprender que de bastantes formas uno puede interpretar un sistema físico para realizar álgebra booleana (es decir, manipulaciones clásicas): Un transistor que a partir de un umbral de voltaje se interpreta como 1, una onda y su ausencia o presencia de la misma, la activación o pausa de un sensor, etc.

Con ello, el *qubit* es *"la unidad constitutiva de la información almacenada y manipulada en la computación cuántica"*. Recordando que en la mecánica cuántica se estudian los fenómenos observables y sus probabilidades, pasamos a comprender que la diferencia entre un *bit* y un *qubit* reside en la capacidad del *qubit* de encontrarse en estados superpuestos entre el 0 y el 1 (de aquí la conocida y quizás confusa frase de "poder ser 1 y 0 a la vez"). Introduciendo la notación de Dirac (que se utilizará a menudo a lo largo de la memoria), decimos que disponer de un *qubit* significa disponer de un modelo físico descrito por la mécanica cuántica con los estados $|0\rangle$ y $|1\rangle$.

En términos físicos y matemáticos, mientras que en el *bit* únicamente almacenamos el 0 o el 1, un *qubit* $|\psi\rangle$ se interpreta como una combinación lineal de sus dos estados con dos números complejos cuya suma es de módulo 1:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = (a + b\cdot i)\cdot|0\rangle + (c + d\cdot i)\cdot|1\rangle\,; \quad \alpha,\beta \in \mathbf{C}; \quad |\alpha|^2 + |\beta|^2 = 1. \tag{A.1}$$

Esta expresión del *qubit* da a entender la ventaja de almacenaje que presenta frente a su alternativa clásica, si bien, recordando la limitación presente entre el mundo macroscópico y los fenómenos cuánticos al realizar observaciones, solo podremos realizar mediciones de un *qubit* obteniendo $|0\rangle$ o $|1\rangle$ como resultado. Con ello, encontrar al *qubit* $|\psi\rangle$ en uno de los dos estados dependerá de las amplitudes de onda de ambas componentes de la superposición o, en términos probabilísticos,

$$P(\{|\psi\rangle = 0\}) = |\alpha|^2, \quad P(\{|\psi\rangle = 1\}) = |\beta|^2, \tag{A.2}$$

Comprendiendo así la restricción anterior sobre estos números complejos para que su suma valga 1. Además, geométricamente podemos interpretar esta condición de suma uno como la restricción de los vectores estado $|\psi\rangle$ a una esfera normalizada de longitud unitaria. Realizando un pertinente cambio de coordenadas apreciaríamos esta correspondencia geométrica, llegando a la famosa interpretación de los *qubits* como elementos de la Esfera de Bloch.

La Esfera de Bloch consiste en una esfera unitaria $\mathbf{S}^2 \subset \mathbf{R}^3$ que ayuda a representar geométricamente el estado de un *qubit* $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, no únicamente para visualizar la influencia de los números complejos α, β, sino para comprender geométricamente el efecto de las distintas rotaciones que podemos ejercer sobre un *qubit*: R_X, R_Y, R_Z o bien U_1, U_2, U_3, correspondientes a rotar sobre los tres ejes de $\mathbf{R}^3$.

Comprendamos paso a paso cómo a partir de dos números complejos α, β podemos llegar a una subvariedad de $\mathbf{R}^3$:

1.– α y β, por ser números complejos, se pueden escribir en función de cuatro números reales en forma exponencial en lugar de diferenciar entre parte real e imaginaria:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = r_\alpha e^{i\phi_\alpha}\,|0\rangle + r_\beta e^{i\phi_\beta}\,|1\rangle\,. \tag{A.3}$$

2.– Puesto que al medir estados cuánticos añadir una fase global resulta en mismos resultados probabilísticos, podemos añadir la fase $e^{-i\phi_\alpha}$ con tal de retirar uno de los parámetros reales.

$$\psi' \;= e^{-i\phi_\alpha}\,|\psi\rangle = r_\alpha\,|0\rangle + r_\beta e^{i\phi}\,|1\rangle\,; \quad \phi = \phi_\beta - \phi_\alpha. \tag{A.4}$$

3.– A continuación, recordando nuestra condición de normalización probabilística (Ecuación A.2), escribimos β en coordenadas cartesianas complejas $x + iy$, de forma que se obtiene la ecuación de una esfera.

$$1 = |\alpha|^2 + |\beta|^2 = r_\alpha^2 + x^2 + y^2. \tag{A.5}$$

4.– Por último, para comprender mejor las rotaciones que podamos realizar sobre un *qubit* en la Esfera de Bloch, reescribimos en forma polar la esfera de coordenadas (r_α, x, y), reajustando ángulos para garantizar la unicidad de los estados.

$$|\psi\rangle = cos(\theta/2)\,|0\rangle + sin(\theta/2)e^{i\phi}\,|1\rangle\,; \quad 0 \leq \theta \leq \pi, 0 \leq \phi \leq 2\pi. \tag{A.6}$$

En la Figura A.1 podemos apreciar la representación de un *qubit* en la Esfera de Bloch y recordar el papel de los ángulos θ y ϕ para determinar el punto.

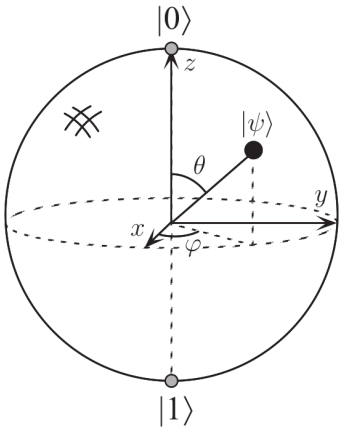


Figura A.1: Representación de un qubit en la Esfera de Bloch [2, Figura 1.3].

A.2. Puertas lógicas cuánticas. Entrelazamiento de *qubits*

Sabiendo que el más bajo nivel de la computación clásica se basaba en "los cables" y "las puertas lógicas", queda comprender cómo podemos operar con los *qubits* de forma análoga. Podríamos comenzar con una de las funcionalidades lógicas más simples, la puerta NOT. Esta función debería convertir las amplitudes de $|0\rangle$ en las de $|1\rangle$ y viceversa. Vemos de esta forma que una puerta lógica cuántica opera sobre los coeficientes α y β de un *qubit*, y este ejemplo quedaría como sigue

$$\alpha\,|0\rangle + \beta\,|1\rangle \mapsto \alpha\,|1\rangle + \beta\,|0\rangle\,. \tag{A.7}$$

Con ello, podemos darnos cuenta de una conveniente forma de interpretar las puertas lógicas como matrices ($\mathbf{M}_{2\times 2}(\mathbf{C})$). En el ejemplo anterior apreciamos que la puerta lógica NOT se corresponde con el siguiente operador lineal

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{A.8}$$

Una de las pocas restricciones (bastante importante) que se le impone a una matriz compleja para considerarse puerta lógica cuántica está en el hecho de que ha de ser unitaria. Es decir, sea U puerta lógica, es necesario que $U^\dagger U = I$. [1] Esta restricción de unitariedad da lugar a una importante propiedad en la computación cuántica: La reversibilidad. Por ello, podríamos revertir cualquier proceso computacional en un ordenador cuántico y podemos hablar de circuitos invertidos. Esta última observación es importante para este trabajo, en el sentido de que se apreciará la aparición de los llamados *qubits ancilla* para que la computación sea reversible en operaciones como sumas o productos.

Previo a comprender algoritmos cuánticos de mayor dificultad que una puerta NOT, es preciso conocer ciertas herramientas matemáticas para aumentar el número de *qubits* que interactúan en un circuito. Se define el *producto de Kronecker* como el operador sobre matrices (y con ello vectores) A, B de dimensiones $m \times n$, $p \times q$ que resulta en la siguiente matriz $mp \times nq$

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}. \tag{A.9}$$

El producto de Kronecker equivaldrá a la hora de elaborar circuitos cuánticos como "colocar en paralelo", ya sean *qubits* o puertas lógicas. Abreviaremos $|0\rangle \otimes |0\rangle$ como $|00\rangle$ y de forma análoga con el resto de elementos de la base computacional multidimensional, compuesta por 2^n elementos. Por ejemplo, un estado cuántico de dos *qubits* $|\psi\rangle$ estará definido por cuatro amplitudes diferentes

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle\,, \tag{A.10}$$

con la nueva condición de normalización $\sum_{x\in\{0,1\}} |\alpha_x|^2 = 1$.

Por último, se presenta un ejemplo simple para comprender la correlación entre el diagrama de un circuito cuántico y su notación matricial. Los *estados de Bell* o *base de Bell* son cuatro estados cuánticos conocidos por tratarse de un ejemplo simple del entrelazamiento cuántico (donde la observación de un *qubit* influye directamente en el segundo, siendo la base teórica del conocido como Internet cuántico [2]).

$$\begin{cases} |\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \\ |\beta_{01}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \\ |\beta_{10}\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \\ |\beta_{11}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \end{cases} \tag{A.11}$$

[1] Se utilizará a lo largo de la memoria † para simbolizar la adjunta de una matriz, e.d., el transpuesto de su conjugado.

El ejemplo a tratar consistirá en un circuito que transforma la base “de ceros y unos” en la base de Bell. Destacamos dos pasos:

1.– Sobre el primer *qubit* x, realizamos una *rotación de Hadamard*, H, donde el *qubit* realiza una rotación de pi radianes en el eje Z, y otra de $\frac{\pi}{2}$ radianes por el eje Y (recordando la interpretación de un *qubit* en la esfera de Bloch)

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{A.12}$$

Puesto que sobre el *qubit* y no se hace nada, el primer paso equivale gracias al producto de Kronecker a $I \otimes H$, aumentando la dimensión en dos *qubits*.

2.– Por último, hacemos uso de una puerta lógica de dos *qubits*, la $CNOT$, análoga a la clásica, donde el segundo *qubit* se “niega” en caso de que el primero esté a 1 (aplicado para el caso cuántico a las amplitudes pertinentes).

$$\text{CNOT} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{A.13}$$

Identificamos las matrices identidad y NOT en las submatrices A_{00} y A_{11} de la $CNOT$, comprendiendo así la condicionalidad explicada.

Con ello, podemos expresar la acción del circuito de este ejemplo sobre un estado $|x, y\rangle ; x, y \in \{0, 1\}$ como la composición de las matrices de ambos pasos:

$$(CNOT) \cdot (H \otimes I) |x, y\rangle = \frac{|0, y\rangle + (-1)^x |1, \bar{y}\rangle}{\sqrt{2}}, \tag{A.14}$$

donde $\bar{y}$ es la negación de y. Sustituyendo x e y por las combinaciones de 0 y 1 llegaríamos al β_{xy} de la base anterior. De forma más simple podemos comprender el funcionamiento de un circuito como este a partir de su diagrama:

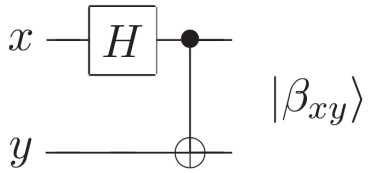


Figura A.2: Circuito de creación de bases de Bell. [2, Figura 1.12]

B

Uso de la clase CreditRiskAnalysis

Procedemos en este anexo a detallar ciertos parámetros y métodos disponibles desde la clase `CreditRiskAnalysis` para que un potencial usuario comprenda qué llamadas a funciones utilizar en función de la información que desea obtener.

B.1. Constructor: `CreditRiskAnalysis`

El punto más importante del *workflow* de preparación y ejecución del análisis de un portfolio financiero reside en la llamada al constructor `CreditRiskAnalysis`. En dicha llamada, un usuario establecerá los parámetros principales de ejecución, ya sea del algoritmo a utilizar y sus factores de precisión o ya sea del *backend* sobre el que ejecutar y el número de muestras a tomar.

Por ello, enumeramos y explicamos cada uno de los argumentos opcionales que se encuentran disponibles para el usuario; recordando que los únicos tres obligatorios, ya explicados, hacen referencia al propio portfolio a analizar.

1.– `backend`: El *backend* sobre el que ejecutar los circuitos cuánticos a diseñar. Podrá ser desde un simulador de vectores estado (`statevector_simulator`) o simulador de QASM; hasta cualquiera de los *backends* disponibles desde IBMQ Experience [16]. Iremos alternando entre simuladores y ordenadores reales para estudiar la escalabilidad de la resolución de estos problemas hacia el mundo real.

2.– `shots`: Número de "disparos" (intentos) a realizar en el simulador QASM o en cualquier *backend* real.

3.– `n_z` y `z_max`: Número de *qubits* a utilizar para la simulación de la distribución normal $\mathcal{N}(0,1)$ e intervalo sobre el que simula. Por defecto `n_z` está en usar dos *qubits*, y veremos más adelante cómo aumentar esta cantidad irá "afinando" la precisión de los resultados. Por otro lado, el intervalo sobre el que se simula la gaussiana por defecto es el $[-2,2]$.

4.– `alpha`: Niveles de confianza tanto para el intervalo de las amplitudes estimado como para el valor umbral del $VaR_\alpha[\mathcal{L}]$. Por defecto se ajusta al 5 %.

5.– `ae_algorithm`: Algoritmo de estimación de amplitud a elegir. Existen tres disponibles: QAE, iQAE y MLAE.

6.– `epsilon`: Parámetro de precisión con respecto a las fases estimadas en la iQAE. Por defecto se establece en el $0{,}01$.

7.– `num_ml_queries`: Parámetro para la MLAE que indica el número de potencias de $\mathcal{Q}^{m_k}$ con el que realizar un mallado de fases. Por defecto es cinco.

8.– `num_eval_qubits`: Parámetro para la QAE que indica el número de qubits utilizados para evaluar el valor de la fase estimada en la QAE. Por defecto es cinco.

B.2. Granularidad de la información: Métodos `build`, `run`, `compute`, `print` y `graph`

Una ventaja que ofrece nuestra librería `CreditRiskAnalysis` consiste en el hecho de ocultar al usuario el *workflow* obligado por parte de Qiskit, a veces de excesiva complejidad cuando queremos obtener el máximo de información posible.

Como norma general y apreciable en el tutorial de Qiskit sobre finanzas [6], un usuario debe seguir los siguientes pasos:

1.– En primer lugar, construir el circuito cuántico pertinente realizando diversas llamadas a constructores y uniendo las puertas lógicas entre sí.

2.– A continuación, mandar ejecutar dicho circuito creado previamente; ya sea ejecutando el circuito como tal o creando un *wrapper* como pueden ser los estimadores de amplitud. Es decir, en dicho caso nosotros crearíamos $\mathcal{A}$ y con el constructor adecuado se prepara el circuito de AE alrededor de $\mathcal{Q}$.

3.– Por último, tratar la estructura de datos pertinente a la salida (que suele consistir en un diccionario de numerosas *keys*) y decidir si imprimir los resultados por pantalla o generar una gráfica, poniendo en este punto "a prueba" los conocimientos del usuario actual sobre `matplotlib` o similares.

En vista de que dicho flujo de trabajo complica innecesariamente el código para cada vez que el usuario pretenda ejecutar un análisis; y más acentuada esta diferencia cuando desee variar algún parámetro como `epsilon` o `alpha`, se aprecia la ventaja que aporta el uso de la clase `CreditRiskAnalysis`.

Con ello, en función de la granularidad de información que requiera el usuario son varios los métodos que puede mandar a ejecutar y, muy importante, sin tener que atender a la secuencia de pasos enumerada anteriormente, en el sentido de que, a modo de pila, llamar por ejemplo al método tercero de la siguiente enumeración supondrá llamar al segundo y al primero.

Tomamos como ejemplo el proceso de construcción, ejecución, cómputo y salida por pantalla de resultados con respecto al modelo de incertidumbre simulado ($\mathcal{U}$, en el código como `U`):

1.– `build_U`: Construye el circuito cuántico. Es utilizado únicamente en caso de querer comprobar que el dibujo del `cra.qc` coincide con el mencionado en la teoría, o porque el usuario desee tener guardado en la clase el circuito en sí.

2.– `run_U`: Manda ejecutar el circuito en el *backend* que se haya establecido previamente. Considerable para usuarios que deseen manipular libremente el diccionario de resultados.

3.– `compute_U`: Método opcional que no en todos los problemas planteados aparece (`U`, `E`, `cdf` y `var`). Computa clásicamente sus resultados, obteniendo así las variables teóricas con las que comparar.

4.– `print_U`: Imprime los resultados teóricos obtenidos por pantalla, informando en todo momento del *backend*

y, cuando sea pertinente, del estimador de amplitud utilizado.

5.– `graph_U`: Genera tres gráficas para comprender la distribución del modelo de incertidumbre a analizar y la calidad de la simulación cuántica. Para otros problemas planteados las gráficas a generar han sido explicadas en la Sección 3.2.1 y ejemplificadas en la Sección 4.1

En vista de lo ejemplificado con respecto al problema de simulación de la incertidumbre, uno puede adelantar el nombre de los siguientes métodos restantes para la clase `CreditRiskAnalysis`:

1.– `*_E`: Para construir, ejecutar, graficar... con respecto al problema de hallar el valor de $E[\mathcal{L}]$.

2.– `*_cdf(x_eval)`: Destacamos en este caso el hecho de aparecer un argumento obligatorio, que consiste en la x sobre la cual se evalúa $CDF_{\mathcal{L}}(x)$. La única excepción es `graph_cdf()`, que representa gráficamente por completo la distribución acumulada.

3.– `*_var()`: Métodos relacionados con el problema de búsqueda del valor en riesgo *alpha* de $\mathcal{L}$.

Por último, recordar al usuario la disponibilidad del método `cra.run_all()`, que se encarga de realizar el sumario gráfico de cada uno de los problemas anteriores y cuya salida por pantalla será análoga a la de los ejemplos provistos y explicados en esta memoria.

C

Ejemplo de Resultados de Ejecución en Simuladores

Adjuntamos en este anexo ejemplos de resultados en simuladores análogos a los introducidos en el cuerpo del documento, esta vez con distintos algoritmos y parámetros.

C.1. Segundo ejemplo: Portfolio de tres activos con MLAE

Se muestra a continuación los parámetros del portfolio a analizar:

Activo (i)	Valor del activo (λ_i)	Probabilidad inicial (p_i^0)	Factor de dependencia (ρ_i)
0	1	0.15	0.1
1	2	0.25	0.05
2	4	0.1	0.1

Tabla C.1: Características del Portfolio del segundo ejemplo.

Este portfolio toma como dos primeros activos los mismo que en nuestro *paper* de referencia [4], y añade un tercero de mayor valor pero menor probabilidad que, además, induciría a errores de ejecución en el tutorial *demo* que comparábamos anteriormente.

C.1.1. Ejecución del Modelo Mixto: $\mathcal{U}$

La salida por pantalla es la siguiente:

```
-------- RESULTS OF U IN qasm_simulator --------
Expected Loss E[L]: 1.05859375
Value at Risk VaR[L]: 4.00000000
P[L <= VaR[L]]: 0.95703125
```

Vemos cómo la esperanza y el valor a riesgo se sitúan a lo largo de la distribución de pérdidas de $\mathcal{L}$ en la primera gráfica resultante.

Las siguientes dos gráficas que genera este método representan la simulación de la distribución

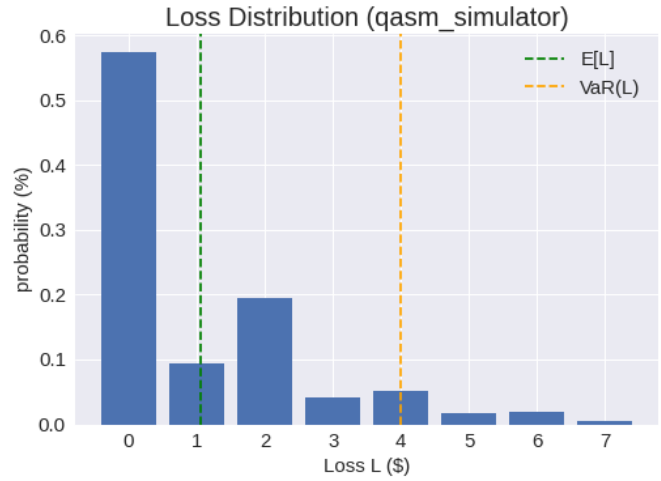


Figura C.1: Distribución de pérdidas del portfolio del ejemplo segundo según el modelo mixto.

de la normal estándar discretizada, así como una comparación entre las probabilidades simuladas de cada *asset* frente a las iniciales establecidas p_i^0.

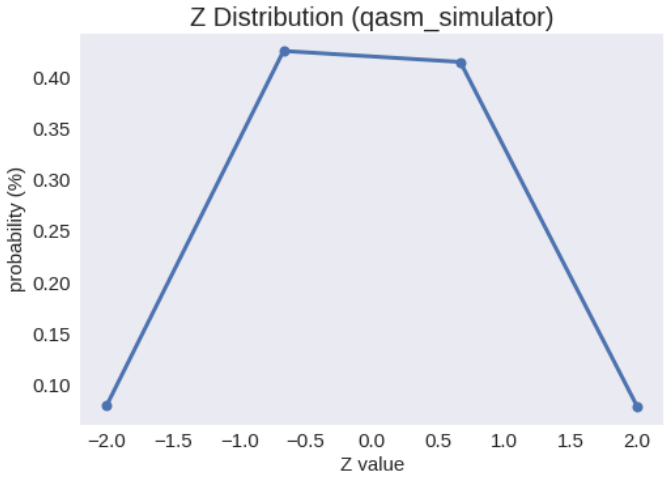


Figura C.2: Simulación de Z en el segundo portfolio ejemplo.

C.1.2. Ejecución de la QAE para la estimación de la esperanza: $E[\mathcal{L}]$

La salida por pantalla tras ejecutar `graph_E()` nos compara el valor "exacto" anterior (realmente se trata del simulado de forma mixta) frente al estimado mediante la MLAE y su intervalo de confianza.

```
   --- RESULTS OF MLAE FOR E IN qasm_simulator ---
Reference value: 1.05859375
Estimated value: 1.07666126
Confidence interval: [1.06971473, 1.08360780]
```

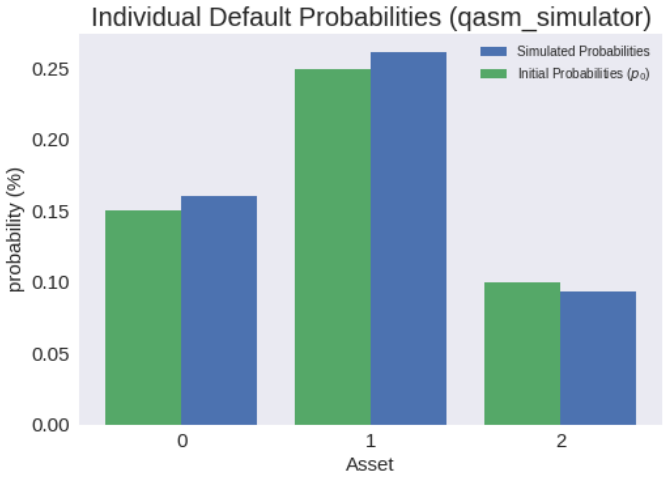


Figura C.3: Comparación entre probabilidades teóricas y simuladas en el segundo ejemplo

Mostramos a continuación una gráfica con el *zoom* hacia la diferencia entre el valor estimado y el real, junto a una vista general con ambos valores a lo largo de la distribución de pérdidas del portfolio.

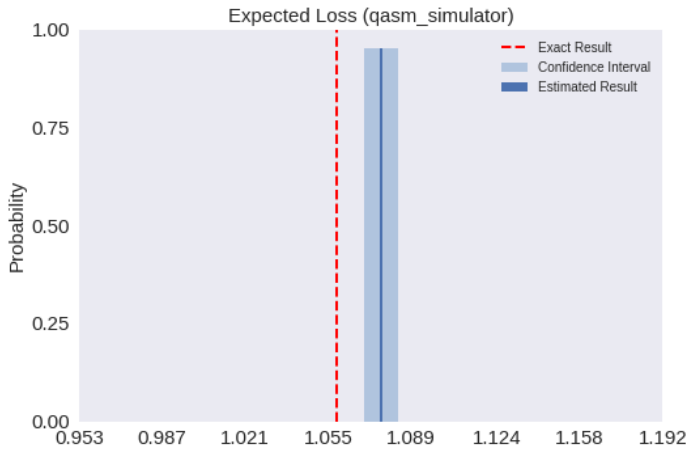


Figura C.4: Comparación entre valor exacto y estimado de $E[\mathcal{L}]$ en el segundo ejemplo.

C.1.3. Función de distribución acumulada: $CDF_{\mathcal{L}}$

Se muestra a continuación la salida por pantalla tras llamar a `graph_cdf`.

Esta visualización de la función de distribución acumulada nos adelanta el hecho de que el $VaR_{\alpha}[\mathcal{L}]$ está en $x = 4$, al ser la x cuyo valor de $CDF_{\mathcal{L}}$ se sitúa más cerca y por encima del $95\,\%$.

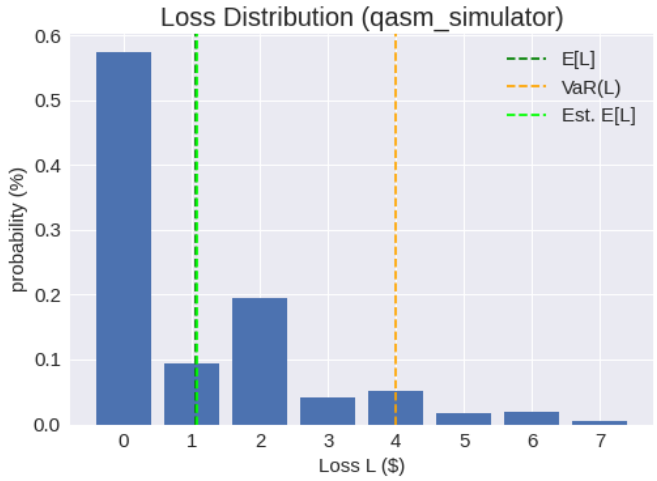


Figura C.5: Distribución de las pérdidas indicando la esperanza estimada mediante QAE.

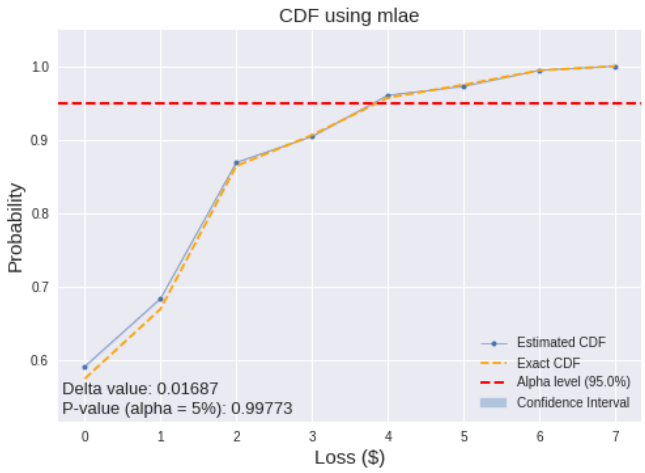


Figura C.6: Función de distribución acumulada del portfolio segundoo.

C.1.4. Valor en riesgo: $VaR_\alpha[\mathcal{L}]$

Vemos en la salida por pantalla de `graph_var()` un error cometido del orden de centésimas con respecto a la probabilidad estimada del valor en riesgo resultante.

```
   --- RESULTS OF MLAE FOR VaR IN qasm_simulator ---
Estimated Value at Risk: 4
Reference Value at Risk: 4
Estimated Probability: 0.96060196
Reference Probability: 0.95703125
```

Se muestra a continuación la exploración realizada por la búsqueda en bisección para hallar $VaR[\mathcal{L}]$.

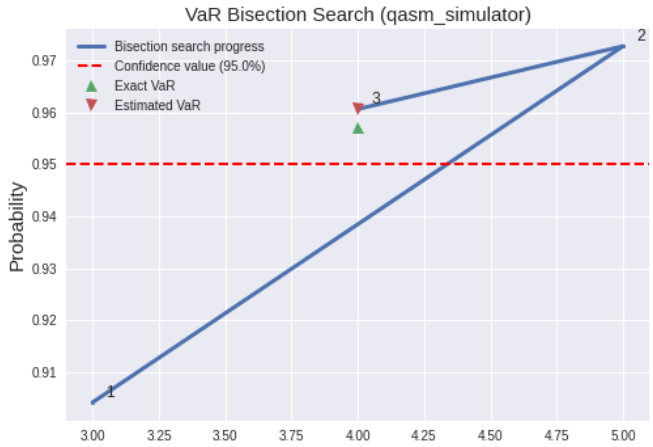


Figura C.7: Búsqueda en bisección del valor en riesgo del segundo ejemplo.

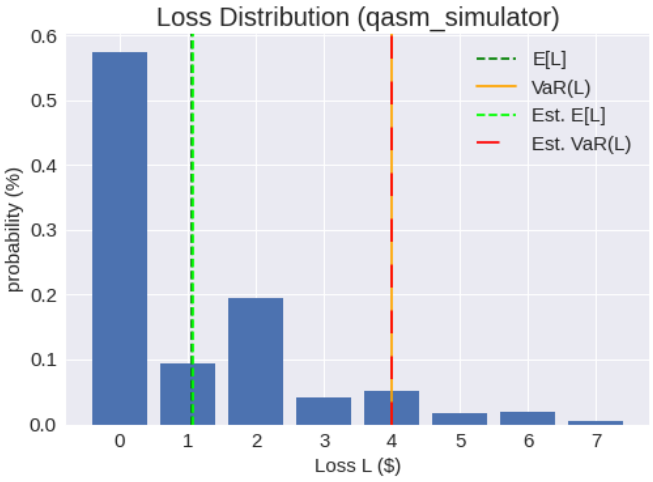


Figura C.8: Distribución de las pérdidas del segundo portfolio añadiendo el VaR estimado..


UAM
UNIVERSIDAD AUTONOMA
DE MADRID