COMPUTACIÓN CUÁNTICA APLICADA A LAS FINANZAS

TRABAJO FIN DE GRADO
CURSO 2022-2023

AUTOR
CINTIA Mª HERRERA ARENAS

DIRECTORES
GUILLERMO BOTELLA JUAN
ALBERTO ANTONIO DEL BARRIO GARCÍA

GRADO EN INGENIERÍA DE COMPUTADORES
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

ÍNDICE

ÍNDICE DE FIGURAS

Resumen

En este trabajo fin de grado, se aborda el desafío de asignar eficientemente un presupuesto limitado a una serie de acciones financieras para maximizar los beneficios, utilizando el algoritmo de optimización variacional cuántica (VQE) y se estudian diferentes optimizadores.

Se desarrollan tres versiones del algoritmo VQE para abordar esta problemática de manera integral. En la primera versión, se emplea un enfoque binario, donde se decide si se escoge o no una acción. En la segunda versión, las acciones se codifican con un peso que indica su proporción. Por último, en la versión final (híbrida), se extraen los datos del historial de acciones de IBM y se combinan los dos enfoques anteriores. En esta versión, se aplica inicialmente el algoritmo de variables binarias para seleccionar las acciones óptimas y, posteriormente, se aplica el algoritmo sobre variables enteras para distribuir el presupuesto (Budget) de manera óptima entre las acciones escogidas.

El presente TFG busca aprovechar la capacidad de los computadores cuánticos reales y los métodos clásicos de optimización para abordar problemas financieros complejos, ofreciendo una solución innovadora y prometedora en el campo de las finanzas cuánticas.

Palabras clave

Computación cuántica, VQE, finanzas, banco, acciones.

Abstract

In this TFG, we study how to efficiently distribute a limited budget to financial stocks to maximize benefits, using the quantum variational optimization algorithm (VQE), and the study of different optimizers.

There are three versions of the VQE algorithm developed to comprehensively approach this issue. In the first version, a binary approach is used, where it is decided if a stoke is or not chosen. In the second version, the stokes are coded with a weight indicating their proportion. Finally, in the final version (hybrid), the IBM stock history data is extracted, and the two previous approaches are combined. In this version, we first apply the binary variables algorithm to select the optimal stocks and, later, we apply the algorithm on integer variables to distribute the budget optimally among the chosen actions.

This TFG looks to take advantage of the capacity of real quantum computers and classical optimization methods to approach complex financial problems, offering an innovative and promising solution in the sector of quantum finance.

Keywords

Quantum computing, VQE, finance, bank, stocks.

Capítulo 1: Introducción

En este capítulo se describe cómo surge la idea de necesitar un programa que ayude y guíe al usuario a la hora de invertir en acciones.

Para ello, se recurre a la computación cuántica que, como se explicará más adelante, presenta una mayor ventaja ante la computación clásica.

Además, se detallan los objetivos para realizar este proyecto.

1.1. Motivación

En este proyecto se busca encontrar un programa que ayude al usuario a la elección sobre qué acciones invertir y de qué manera hacerlo. Se ha propuesto apostar por un algoritmo cuántico, debido a los beneficios que presenta.

La computación cuántica presenta una mayor potencia de cálculo y, junto a la superposición y otras propiedades que se explicarán en capítulos posteriores, puede romper con muchos algoritmos de ciberseguridad actuales. Por ello, cada vez más empresas están empezando a interesarse en incorporar tecnologías criptográficas que protejan la información tanto en computadoras clásicas como cuánticas.

Además, gracias al escalado que los computadores cuánticos presentan, son capaces de explorar con más facilidad el conjunto de todas las soluciones posibles. Esto es útil en el ámbito de las finanzas, en la que influyen múltiples dimensiones para tener en cuenta para tomar la mejor decisión.

1.2. Objetivos

El objetivo de este proyecto es realizar una prueba de concepto de la computación cuántica en el contexto de la optimización de portfolio.

Dentro de la teoría de carteras, optimizar una cartera de inversiones se torna más complejo a medida que aumenta el número de activos.

Se estudia el algoritmo Variational Quantum Eigensolver (VQE), utilizado para este tipo de optimización, y se pone a prueba en simuladores proporcionados por IBM. Para tener una perspectiva más amplia, se contrastan estos resultados con los obtenidos en algoritmos clásicos bajo las mismas condiciones.

1.3. Plan de trabajo

Dado que el proyecto es individual, no han sido necesarias reuniones para planificar y repartir el trabajo.

Para el estudio del VQE con variables binarias, se han requerido 2 meses aproximadamente.

Para el estudio del VQE con variables enteras, 1 mes.

Dedicando el resto de los meses al VQE con acciones reales y la comparación de éste con algoritmos clásicos.

Capítulo 2: Estado del Arte

• 2.1. Quantum Computing

La computación cuántica es una rama que fusiona aspectos de las ciencias de la física, matemática y computación (Johnston. 2019).

La computación cuántica, explora el procesamiento de información aprovechando principios cuánticos fundamentales. En sistemas informáticos convencionales, la representación de datos se realiza mediante bits binarios que existen en estados definidos de 0 o 1 (Hidary. 2019). En contraste, la arquitectura cuántica opera mediante qubits, que, gracias a la superposición cuántica, pueden existir en una combinación lineal de ambos estados simultáneamente. Añadiendo a esto fenómenos como el entrelazamiento cuántico, las máquinas cuánticas presentan capacidades computacionales potencialmente exponenciales en comparación con sus homólogas clásicas. (Carrascal G. d., 2021) Estas propiedades permiten a las computadoras cuánticas abordar y potencialmente resolver problemas computacionales de clase NP-Completo, que son intratables para la computación clásica, como son el caso del algoritmo de Grover y el algoritmo de Shor.

El algoritmo de Grover consiste en encontrar un elemento en una lista desordenada (algoritmo de búsqueda). En el caso de resolver dicho problema de manera clásica, se tiene un coste de O(n), siendo n el número de elementos de la lista. Mientras que, si se resuelve de manera cuántica, el coste es O(√n). (Grover., 1996)

Por otra parte, el algoritmo de Shor se usa para resolver la etapa más compleja de la factorización de un número entero de n bits. Si dicho algoritmo se resuelve de manera clásica, se obtiene un coste de O(e^(cn^(1/3)*(log n)^(2/3))), mientras que de forma cuántica el coste sería O(n²(log n)(log log n)). (Shor, 1995)

A continuación, se puede observar la esfera de Bloch (Ilustración 1), que es una representación de un qubit. (Gines Carrascal, 2023)

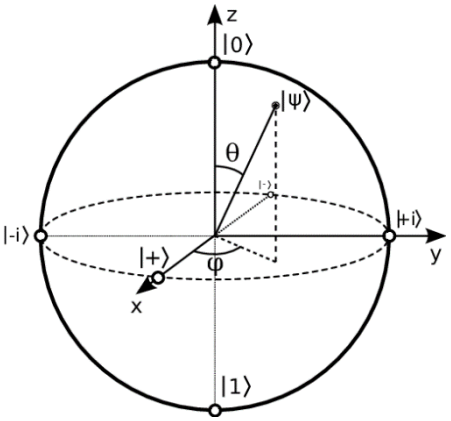

Ilustración 1. Esfera de Bloch.

Representándose un estado cuántico como:

$|\varphi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\theta}\sin(\theta/2)\,|1\rangle$

Por tanto, un qubit se puede representar como una combinación lineal de los estados $|0\rangle$ y $|1\rangle$:

$|\varphi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$

resultando, $|\alpha|^2 + |\beta|^2 = 1$

A continuación, se enumeran los principios básicos de la computación cuántica.

Superposición: este principio establece que un qubit puede estar en una combinación de dos o más estados simultáneamente.

Entrelazamiento: esta propiedad ocurre cuando dos sistemas (o qubits) se entrelazan o vinculan, de forma que ambos comparten información sobre el otro, sin importar cuánto de separados estén.

Interferencia: este principio es una consecuencia de la superposición. Se produce cuando dos qubits se superponen, y su estado final es "0".

Gracias a estas ventajas que ofrece la computación cuántica, es común su uso en sectores como las finanzas o la simulación de sistemas químicos, ya que, permiten abordar problemas de manera más eficiente que con otros algoritmos clásicos.

• 2.2. VQE

El VQE, Variational Quantum Eigensolver, es un algoritmo usado en computación cuántica. Dicho algoritmo permite estimar el valor mínimo correspondiente a una matriz. Este algoritmo utiliza el principio variacional para calcular la energía del estado fundamental hamiltoniano. (Jules Tilly, 2022)

En mecánica cuántica, un Hamiltoniano es el observable que representa la energía total del sistema, es decir, es un operador autoadjunto (es decir, cuando el dominio de un operador hermítico y el de su operador adjunto coinciden totalmente) definido sobre un dominio denso en el espacio de Hilbert del sistema.

Una de sus principales ventajas frente a otros algoritmos es, poder resolver el problema, pero, con un circuito menos profundo.

Como se puede observar en la Ilustración 2, primero se realizan algunos cálculos previos a través de un algoritmo clásico. Se calcula la matriz Hamiltoniana y se genera el estado inicial.

A continuación, se pasa a la parte cuántica. Aquí, se prepara el estado cuántico dependiendo del problema y parámetros dados.

Tras la ejecución, se miden y evalúan los resultados obtenidos. En caso de haber encontrado la solución al problema, hemos terminado. En caso

contrario, se debe de repetir el proceso. (Carrascal de las Heras, Hernamperez Manso, Botella, & del Barrio, 2023). Para ello, se reajustan los parámetros de entrada, y se repite hasta que se encuentre la solución. (Carrascal G. R., 2023)

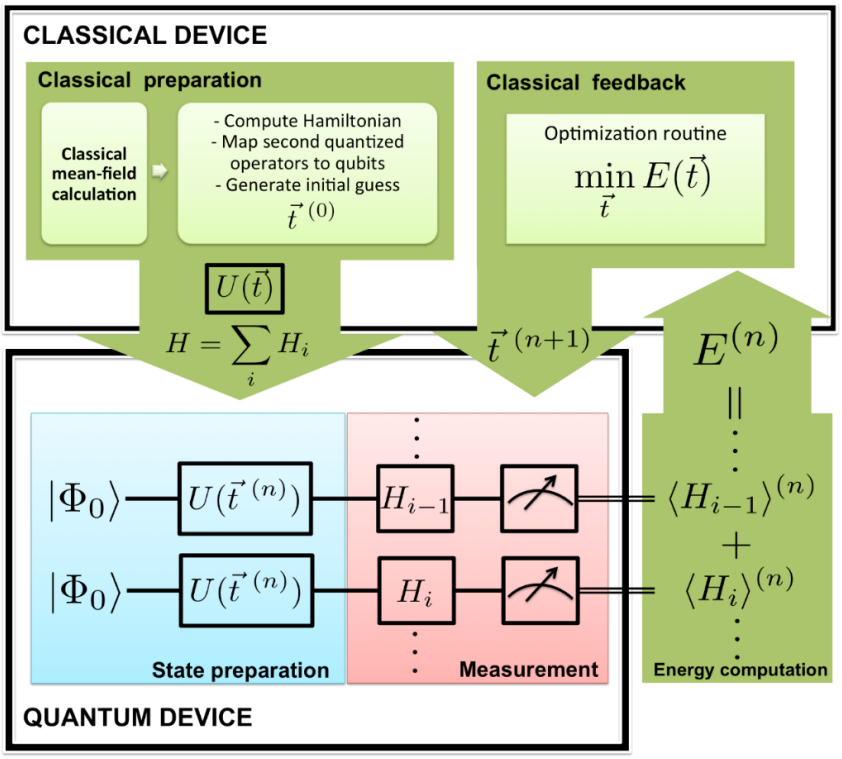

Ilustración 2. VQE Esquema I.

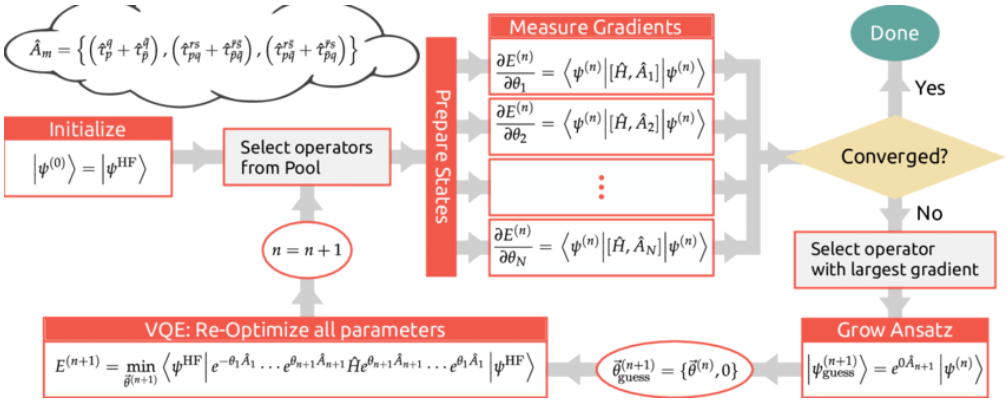

Ilustración 3. VQE Esquema II.

● 2.3. Finanzas

Las finanzas son una rama de la economía que estudia la gestión del dinero y capital. Además, analiza los posibles riesgos que conlleva invertir en acciones (posible pérdida de dinero…). Por tanto, siempre se tiende a minimizar las pérdidas y maximizar las ganancias.

El objetivo final de las finanzas es conseguir un buen uso y gestión del dinero y capital, ya sea para una empresa o individuo, de forma que éste consiga beneficios económicos.

Es importante destacar la optimización de carteras, la cual consiste en seleccionar la mejor combinación de inversiones para maximizar el retorno y minimizar el riesgo, según las características y aversión al riesgo del cliente. El objetivo es encontrar el equilibrio ideal entre riesgo y retorno, considerando factores como objetivos de inversión, condiciones del mercado y preferencias individuales. Esto implica analizar riesgos y retornos de activos como acciones, bonos y bienes raíces.

El proceso incluye definir objetivos de inversión, identificar y analizar activos disponibles, determinar la combinación óptima de activos y monitorear regularmente la cartera. A pesar de que la optimización matemática es útil, tiene limitaciones, ya que se basa en supuestos y datos históricos, y no garantiza rendimientos futuros. Además, los modelos suelen tener una perspectiva estática del mercado.

En el Capítulo 7, se realiza una comparación del VQE (algoritmo cuántico) con tres algoritmos clásicos en distintos escenarios.

Los algoritmos clásicos con los que se han realizado el estudio han sido los siguientes:

- SMA Strategy
- Mean-Variance optimization
- Convex optimization

A continuación, se explica en qué consiste cada uno.

● 2.3.1. SMA Strategy

Los datos históricos relacionados con el precio de mercado de un activo tienen indicios de estar relacionados con cómo variará su precio en un futuro, por tanto, merece la pena prestar atención a su historial a la hora de invertir en acciones.

La estrategia SMA es una herramienta de análisis técnico ampliamente utilizada para predecir tendencias de precios futuras mediante el análisis de datos de precios históricos.

El SMA es el precio medio de mercado de un valor durante un período específico. Se le conoce como el 'moving' average ya que se traza en un gráfico barra por barra y forma una línea que se mueve a lo largo del gráfico a medida que cambia el precio promedio.

El SMA se calcula sumando el precio de un valor durante un período y luego dividiendo esa cifra por el número de períodos.

Por ejemplo, sumar los precios de cierre de un valor del mes anterior y luego dividir el total por el número de días del mes.

La fórmula para un promedio móvil simple es:

$$SMA = \frac{(A_1 + A_2 + A_3 + \ldots + A_N)}{N}$$

Ai = el precio del activo en el período i

N = el número de períodos totales (personalizable)

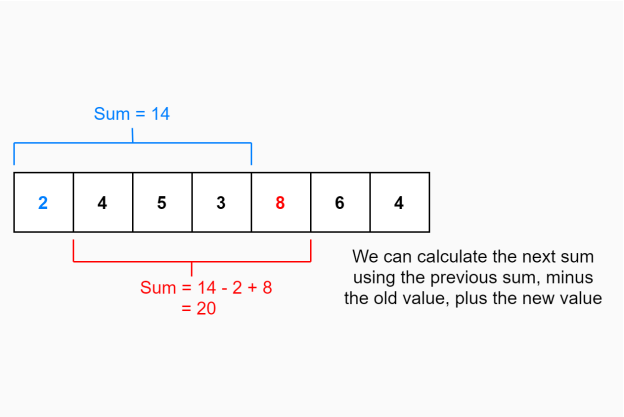

Ilustración 4. SMA Ejemplo.

SMA beneficia a los inversores a corto y largo plazo. Suaviza la volatilidad promediando el precio del valor durante un período determinado y ayuda a identificar tendencias.

- ## 2.3.2. Mean-Variance optimization

La optimización de la varianza media es un elemento clave de la inversión basada en datos . Es el proceso de medir el riesgo de un activo frente a su rendimiento probable e invertir en función de esa relación riesgo/rendimiento.

Un análisis de media-varianza es una herramienta que los inversores utilizan para ayudar a distribuir el riesgo en sus carteras , que consta de dos elementos clave: riesgo (risk), expresado como varianza (cuánto ha cambiado el precio del activo y qué tan rápido lo ha hecho durante un período de tiempo), y recompensa (reward), expresada como retorno.

En él, el inversor mide el riesgo de un activo, y luego lo compara con el rendimiento probable del activo. El objetivo de una optimización de varianza media es maximizar la recompensa de una inversión en función de su riesgo.

Una cartera optimizada debería generar el mayor rendimiento posible en función de la cantidad de riesgo, y generar la menor cantidad de riesgo para el retorno.

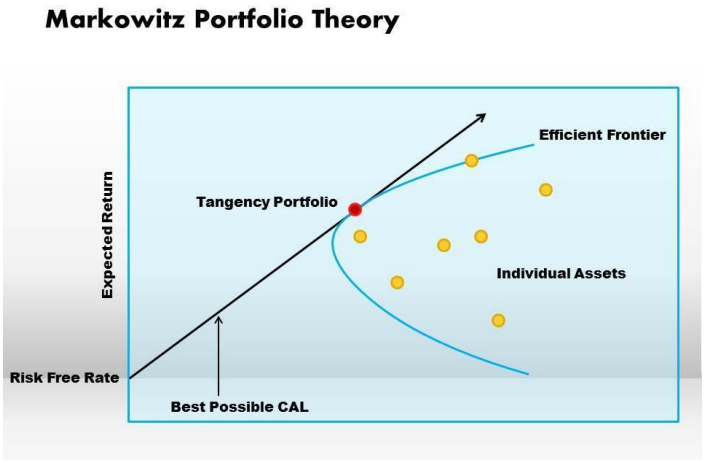

Ilustración 5. Mean-Variance optimization.

- ## 2.3.3. Convex optimization

A diferencia de los análisis descriptivos y predictivos, que se centran en comprender datos pasados y predecir resultados futuros, los análisis prescriptivos tienen como objetivo optimizar los procesos de toma de decisiones.

La optimización convexa es un subcampo de la optimización matemática que estudia el problema de minimizar funciones convexas sobre conjuntos convexos. Muchas clases de problemas de optimización convexa admiten algoritmos de tiempo polinomial, mientras que la optimización matemática es en general NP-difícil .

Un problema de optimización convexa es un problema de optimización en el que la función objetivo es una función convexa y el conjunto factible es un conjunto convexo. Su objetivo es encontrar un punto que maximice o minimice la función objetivo.

En general, se cumple que:

- todo mínimo local es un mínimo global.
- el conjunto óptimo es convexo.
- si la función objetivo es estrictamente convexa, entonces el problema tiene como máximo un punto óptimo.

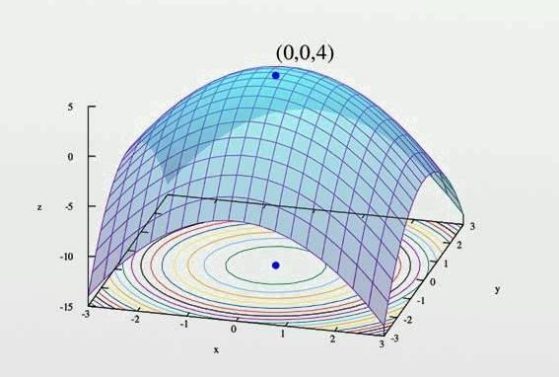

Ilustración 6. Convex optimization.

● 2.3.4. IBM Finance

IBM es una de las empresas que facilita computadores cuánticos al usuario. De esta forma, se pueden ejecutar programas ya sea en un entorno de simulación, o computador real (es decir, con ruido).

IBM Finance es una parte de IBM que se encarga de integrar, organizar y supervisar las transacciones financieras.

Dentro de IBM, podemos encontrar qiskit, que es un kit de desarrollo de software creado por IBM para trabajar con computadoras cuánticas a nivel de circuitos, pulsos y algoritmos.

Si nos vamos al sector de finanzas, se encuentra qiskit finance, que es un framework de código abierto que contiene componentes de incertidumbre para problemas de acciones/valores, aplicaciones para problemas financieros, como la optimización de cartera, y proveedores de datos para obtener datos reales o aleatorios para experimentos financieros.

Capítulo 3: Problema con variables binarias

Un problema con variables binarias es aquel en el que las variables enteras están restringidas entre los valores 0 y 1, representando el 0 la opción de no escoger una variable, y 1 si se ha escogido.

Como primer acercamiento al problema, se verá un enfoque simplificado, en el cual se tiene un número pequeño de acciones (en este caso 6), y se tiene que decidir si se escoge una acción o no (0 o 1).

El algoritmo es el siguiente: Primero, se indican las variables iniciales y las restricciones del problema y lo que se quiere (en este caso, que el número de acciones a escoger sea igual al Budget). A continuación, se ejecutará el circuito n veces (haciendo uso del VQE) para conseguir el resultado óptimo.

Como se ha mencionado, el número de acciones es 6, y el Budget 3. Cabe destacar que, en este caso, al ser un problema de variables binarias, el Budget es, a parte del dinero que se dispone, el número de acciones a escoger.

Para conseguir quedarse con la mejor solución, se recurre a la variable "penalty" (o penalización), la cual "premia" los resultados buenos, de esta manera, se descartan soluciones lejanas a la deseada.

La variable *mu* (μ), representa los retornos esperados de los activos, y *sigma* las covarianzas (o desviación típica) entre los activos.

En la Ilustración 7 se puede observar un posible ejemplo de cómo serían los parámetros iniciales del problema.

```python
# prepare problem instance
n = 6             # number of assets
q = 0.5           # risk factor
budget = n // 2   # budget
penalty = 2*n     # scaling of penalty term
```

```python
# instance from [1]
mu = np.array([0.7313, 0.9893, 0.2725, 0.8750, 0.7667, 0.3622])
sigma = np.array([
    [ 0.7312, -0.6233,  0.4689, -0.5452, -0.0082, -0.3809],
    [-0.6233,  2.4732, -0.7538,  2.4659, -0.0733,  0.8945],
    [ 0.4689, -0.7538,  1.1543, -1.4095,  0.0007, -0.4301],
    [-0.5452,  2.4659, -1.4095,  3.5067,  0.2012,  1.0922],
    [-0.0082, -0.0733,  0.0007,  0.2012,  0.6231,  0.1509],
    [-0.3809,  0.8945, -0.4301,  1.0922,  0.1509,  0.8992]
])
```

Ilustración 7. Problema con variables binarias. Ejemplo parámetros iniciales.

A continuación, se puede ver cómo se crea el modelo (clásico) del problema. En dicho modelo, se indican las restricciones del problema. En este caso, se limita cada variable entre 0 y 1, ya que, como se ha explicado, en este primer modelo solo importa escoger o no una acción. También se indica que la suma del precio de todas las acciones debe ser igual al Budget, ya que, como se trabaja con variables binarias y se busca maximizar el resultado, esto solo se consigue si el número de acciones seleccionadas es igual al Budget.

```python
# create docplex model
mdl = Model('portfolio_optimization')
x = mdl.binary_var_list('x{}'.format(i) for i in range(n))
objective = mdl.sum([mu[i]*x[i] for i in range(n)])
objective -= q * mdl.sum([sigma[i,j]*x[i]*x[j] for i in range(n) for
mdl.maximize(objective)
mdl.add_constraint(mdl.sum(x[i] for i in range(n)) == budget)

# case to
qp = from_docplex_mp(mdl)
```

Ilustración 8. Problema con variables binarias. Modelo.

Finalmente, se crea el problema cuadrático y se muestra como quedarían los parámetros iniciales.

Aquí se puede ver como se detallan las restricciones que se indican, así como el cálculo para poder conseguir el máximo beneficio.

```
1 print(qp.export_as_lp_string())
```

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: portfolio_optimization

Maximize
 obj: 0.731300000000 x0 + 0.989300000000 x1 + 0.272500000000 x2
      + 0.875000000000 x3 + 0.766700000000 x4 + 0.362200000000 x5 + [
      - 0.731200000000 x0^2 + 1.246600000000 x0*x1 - 0.937800000000 x0*x2
      + 1.090400000000 x0*x3 + 0.016400000000 x0*x4 + 0.761800000000 x0*x5
      - 2.473200000000 x1^2 + 1.507600000000 x1*x2 - 4.931800000000 x1*x3
      + 0.146600000000 x1*x4 - 1.789000000000 x1*x5 - 1.154300000000 x2^2
      + 2.819000000000 x2*x3 - 0.001400000000 x2*x4 + 0.860200000000 x2*x5
      - 3.506700000000 x3^2 - 0.402400000000 x3*x4 - 2.184400000000 x3*x5
      - 0.623100000000 x4^2 - 0.301800000000 x4*x5 - 0.899200000000 x5^2 ]/2
Subject To
 c0: x0 + x1 + x2 + x3 + x4 + x5 = 3

Bounds
 0 <= x0 <= 1
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1
 0 <= x5 <= 1

Binaries
 x0 x1 x2 x3 x4 x5
End
```

Ilustración 9. Problema con variables binarias. Restricciones.

Antes de ejecutar el algoritmo cuántico, se debe aplicar el penalty y convertir el problema en un problema linear.

A continuación, se procede a ejecutar el VQE. Como se sabe, los algoritmos cuánticos requieren ejecutarse varias veces, ya que, debido al ruido, los resultados pueden verse afectados, y tenemos que quedarnos con el valor que más veces se repita, ya que, ese tendrá más probabilidad de ser el

correcto. En este caso, se lanza 100 veces. En futuras versiones se ejecutará un mayor número de veces.

Como optimizador, escogemos el COBYLA (más adelante se mostrará un estudio de cómo actúa el algoritmo con distintos optimizadores).

Cabe destacar que, en esta primera aproximación, se lanza el circuito en un backend local.

El *alpha* escogido para este problema es de 0.35 (cuanto porcentaje de los extremos de la campana Normal de Gauss despreciamos).

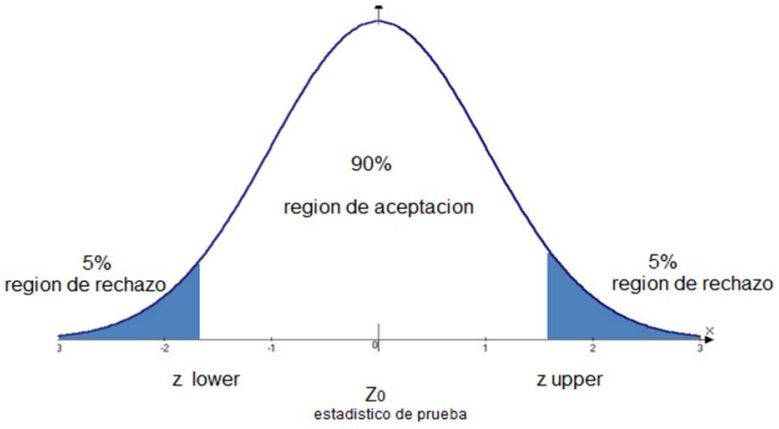

Ilustración 10. Ejemplo alpha=0.05 (Campana de Gauss).

Se aplica el VQE con los parámetros indicados y, se calcula el valor mínimo. Como se puede observar, los resultados obtenidos con el VQE y el circuito clásico son parecidos.

Al ser un problema "pequeño" (problema binario y número de acciones = 6), se puede ejecutar clásicamente, para tener un referente del resultado óptimo, y ver cuánto se ha alejado con el algoritmo cuántico.

Resultados algoritmo clásico:

```
optimal function value: 1.27835
optimal value: [1. 1. 0. 0. 1. 0.]
status: SUCCESS
```

Ilustración 11. Problema con variables binarias. Resultados algoritmos clásico.

Resultados algoritmo cuántico:

```
fval=1.27835, x0=1.0, x1=1.0, x2=0.0, x3=0.0, x4=1.0, x5=0.0, status=SUCCESS
```

Ilustración 12. Problema con variables binarias. Resultados algoritmo cuántico.

Como se puede observar, el valor mínimo coincide con la búsqueda realizada en un algoritmo clásico. De estos resultados se puede concluir que, para el Budget dado, la solución óptima será escoger las acciones x0, x1, x3 y, descartar x2 y x5.

Capítulo 4: Estudio de distintos optimizadores

Para concluir cual optimizador era el más adecuado para este problema, se realiza un estudio con varios optimizadores y, modificándose los valores de *alpha* y *maxiter* (número de vueltas que se ejecuta el algoritmo cuántico), para determinar cuál es mejor, cual es más rápido, o cual encuentra la solución con un número de vueltas menor.

Algunos de los resultados fueron los siguientes:

● 4.1. COBYLA

COBYLA es la abreviatura de Constrained Optimization BY Linear Aproximation, es decir, se basa en las aproximaciones lineales para optimizar el circuito. (Gines Carrascal, G. B. ,2021)

COBYLA se usa sobre todo en problemas que tienen restricciones (como es nuestro caso), y cuya derivada de la función objetivo es desconocida.

La gráfica de este optimizador tiende a estabilizarse cuanto más próximo está de la solución.

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 80 y 90 se encuentra el resultado correcto.

  NOTA: El rango del *maxiter* se indica la solución óptima en menor tiempo. Siempre que se aumente el *maxiter* va a seguir encontrando la solución, mientras que, si se disminuye, puede no encontrarla.

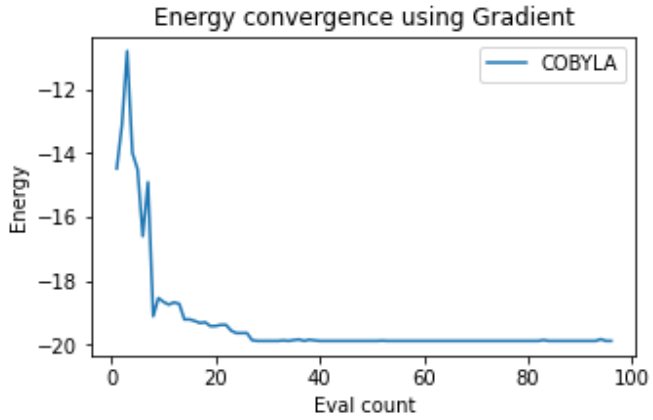

Ilustración 13. COBYLA Gráfica (maxiter=100, alpha=0.35).

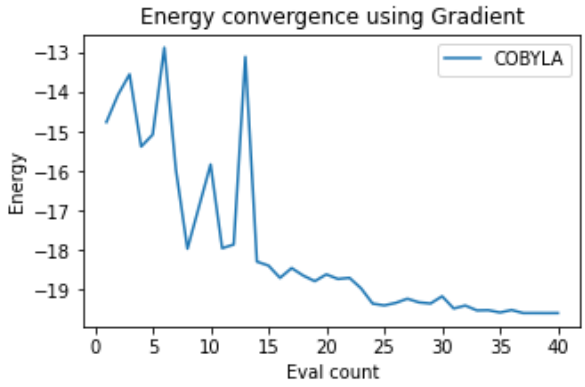

Ilustración 14. COBYLA Gráfica (maxiter=50, alpha=0.35).

- Con *maxiter* = 100 y variando *alpha*.
  Conclusión: A mayor *alpha*, más le cuesta estabilizarse y encontrar la solución. Esto tiene sentido, ya que, al ser *alpha* el porcentaje de datos que desprecia de la Normal, puede descartar soluciones válidas.

  En general, siempre va a mostrar mejores resultados aquellos problemas que usen un *alpha* lo más cercano a cero.

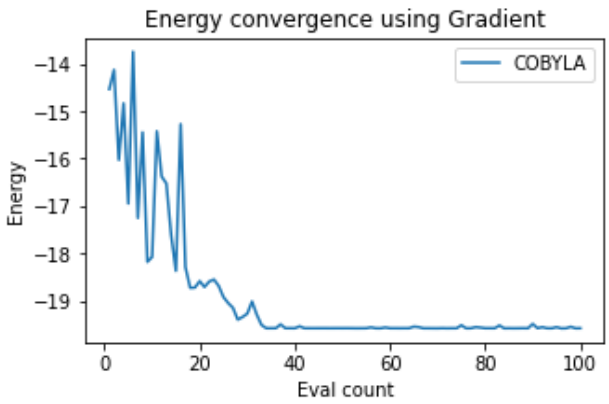

Ilustración 15. COBYLA Gráfica (maxiter=100, alpha=0.65).

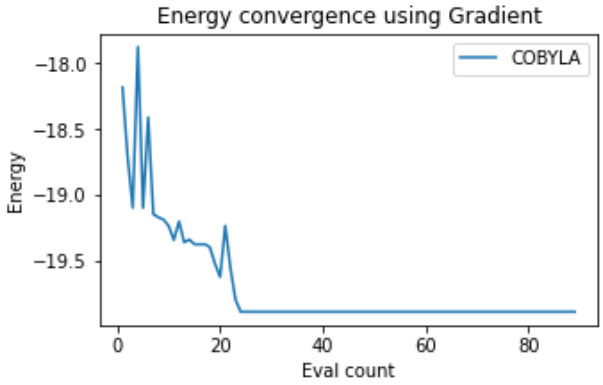

Ilustración 16. COBYLA Gráfica (maxiter=100, alpha=0.15).

- Variando *alpha* y *maxiter*.
  Conclusión: Si disminuimos *alpha* (~0.15), podemos encontrar la solución con un *maxiter* menor (50), hecho que no sucedía con *alpha* = 0.35.

● 4.2. GSLS

GSLS es la abreviatura de Gaussian-smoothed Line Search, es decir, se basa en la búsqueda de línea suavizada por el método de Gauss. (Nesterov. 2017)

La gráfica de este optimizador tiende a estabilizarse cuanto más próximo está de la solución.

- Con *alpha* = 0.35 y variando *maxiter*.

Conclusión: Con *maxiter* entre 60 y 70 se encuentra el resultado correcto.
El GSLS es más estable al encontrar la solución (converge antes), como el COBYLA, pero tarda más (2 min).

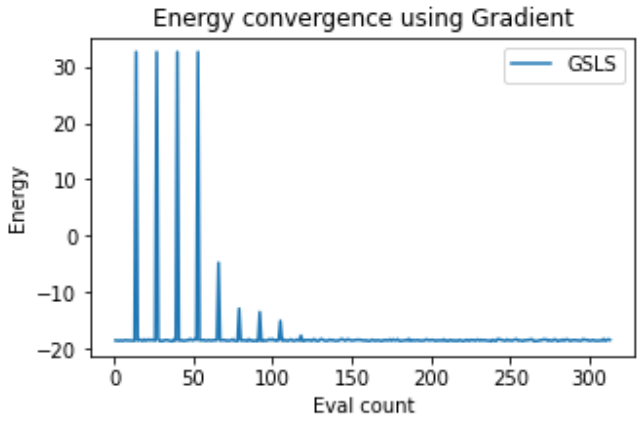

Ilustración 17. GSLS Gráfica (maxiter=100, alpha=0.35).

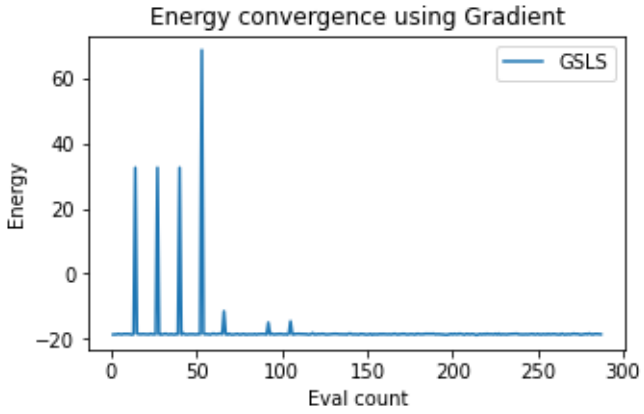

Ilustración 18. GSLS Gráfica (maxiter=70, alpha=0.35).

- Con *maxiter* = 80 y variando *alpha*.
  Conclusión: Con menor *alpha*, se necesita un menor *maxiter* para encontrar la solución. También se muestra una gráfica más estable.

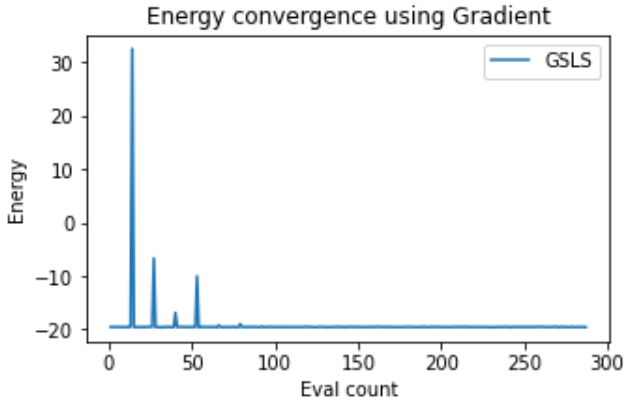

Ilustración 19. GSLS Gráfica (maxiter=80, alpha=0.15).

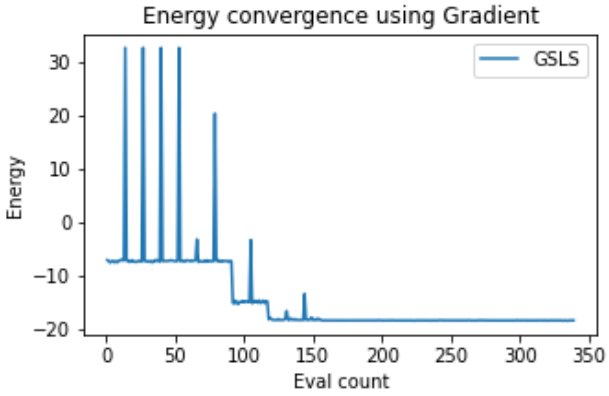

Ilustración 20. GSLS Gráfica (maxiter=80, alpha=0.65).

- Variando *alpha* y *maxiter*.
  Conclusión: Si se disminuye *alpha*, puedes disminuir *maxiter* para encontrar la solución óptima.
  Con *alpha* = 0.35, se encuentra la solución con *maxiter* entre 60 y 70.
  Con *alpha* = 0.15, se encuentra la solución con *maxiter* entre 20 y 30.

● 4.3. L_BFGS_B

L_BFGS_B es la abreviatura de Limited-memory BFGS Bound (BFGS es debido a las siglas de Broyden-Fletcher-Goldfarb-Shanno, sus creadores). (Gonglin Yuan, 2011)

Este es un optimizador cuasi-Newton de funciones con un gran número de parámetros o de gran complejidad.

Este optimizador trata de minimizar el valor de una función escalar diferenciable con restricciones simples.

La gráfica de este optimizador tiende a estabilizarse cuanto más próximo está de la solución.

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 110 y 120 encuentra la solución.
  En la gráfica, se puede apreciar cómo converge el algoritmo.
  Tarda un poco menos que el GSLS en ejecutarse, pero encuentra la solución con un *maxiter* mayor.

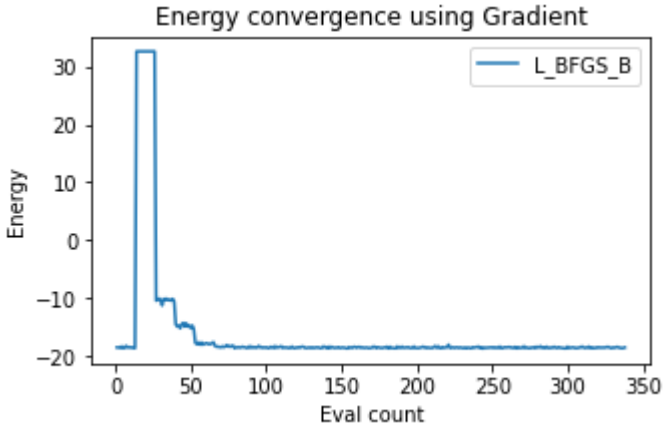

Ilustración 21. L_BFGS_B Gráfica (maxiter=100, alpha=0.35).

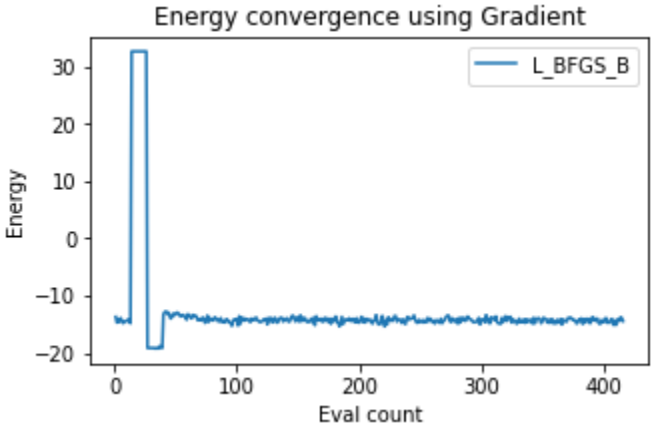

Ilustración 22. L_BFGS_B Gráfica (maxiter=120, alpha=0.35).

- Con *maxiter* = 120 y variando *alpha*.
  Conclusión: Si se disminuye *alpha* por debajo de 0.25, no encuentra la solución.
  Como se puede apreciar, la gráfica es más inestable cuanto mayor sea *alpha*.

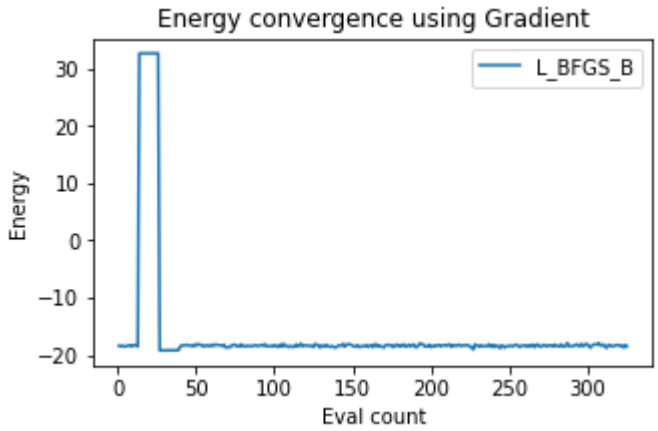

Ilustración 23. L_BFGS_B Gráfica (maxiter=120, alpha=0.15).

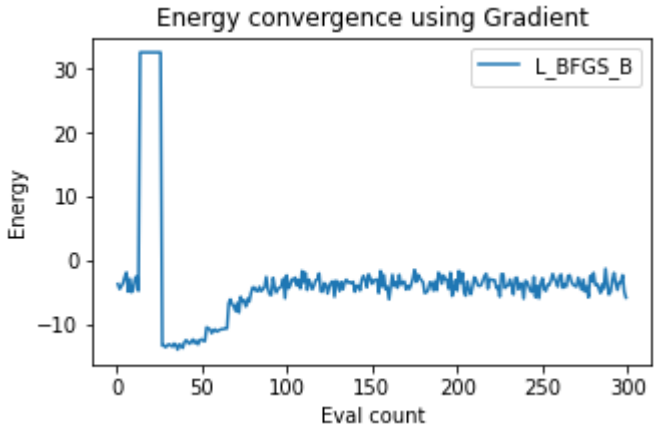

Ilustración 24. L_BFGS_B Gráfica (maxiter=120, alpha=0.65).

- Variando *alpha* y *maxiter*.
  Conclusión: Con *alpha* = 0.15, encuentra la solución con un *maxiter* entre 80 y 90.
  Cuanto mayor sea *alpha*, más tarda en ejecutarse.

● 4.4. NELDER_MEAD

Se usa para optimizar un circuito que no requiere restricciones (debido a esto, no podemos aplicarlo en nuestro problema). (Nelder, 1965)

Consiste en encontrar el mínimo de una función objetivo en un espacio multidimensional. En la Ilustración 25 se muestra el proceso para encontrar el valor mínimo.

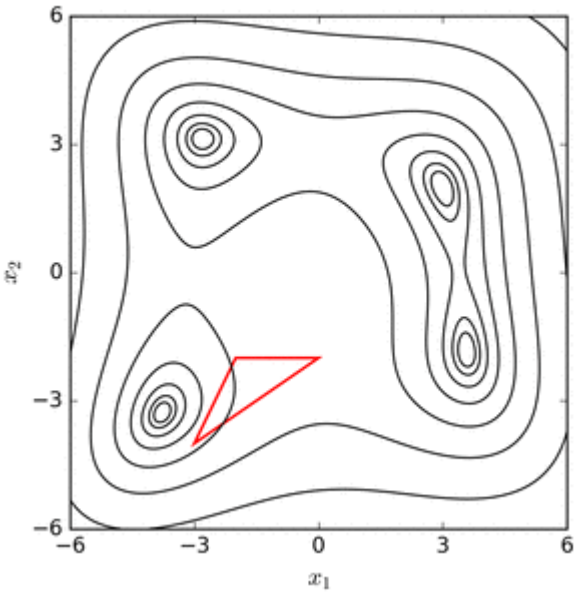

Ilustración 25. NELDER MEAD. Búsqueda del valor mínimo en la función de Himmelblau.

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 50 y 60 encuentra la solución (mejor que GSLS). Tarda en ejecutarse parecido al GSLS.

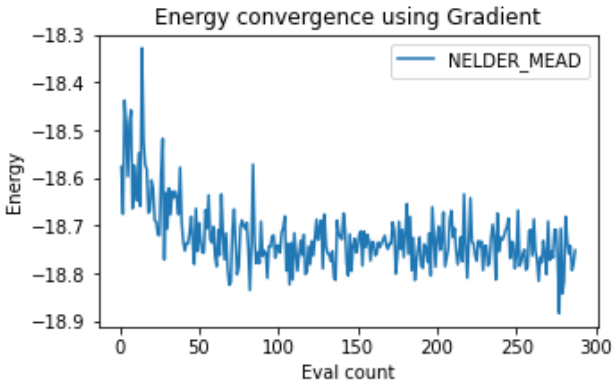

Ilustración 26. NELDER_MEAD Gráfica (maxiter=100, alpha=0.35).

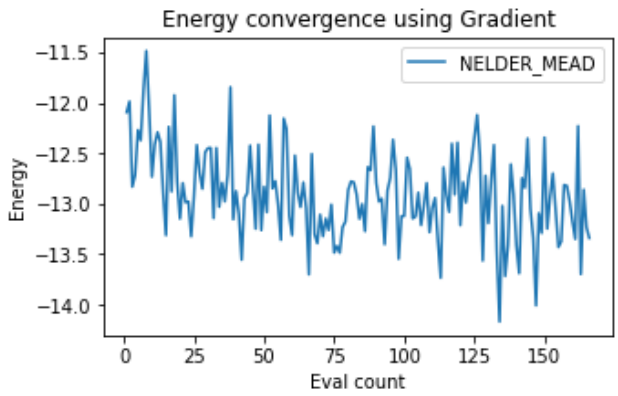

Ilustración 27. NELDER_MEAD Gráfica (maxiter=60, alpha=0.35).

- Variando *alpha* y *maxiter*.
  Conclusión: Parece que no afecta tanto el valor de alpha, si no el del *maxiter* a la hora de encontrar la solución.

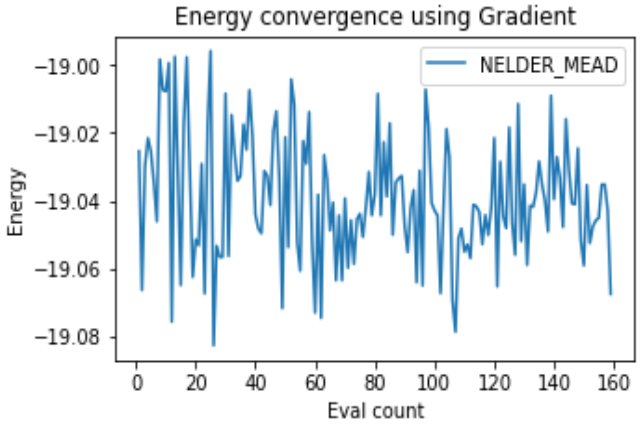

Ilustración 28. NELDER_MEAD Gráfica (maxiter=60, alpha=0.15).

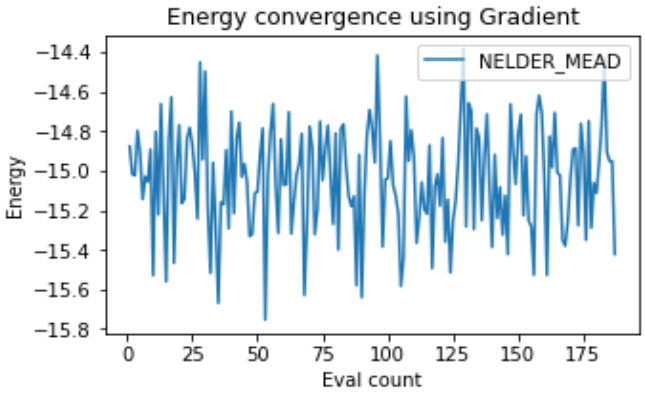

Ilustración 29. NELDER_MEAD Gráfica (maxiter=60, alpha=0.65).

● 4.5. ADAM

Este optimizador basado en gradientes se basa en estimaciones adaptativas de momentos de orden inferior. Además, no necesita mucha memoria y es invariable al cambio de escala diagonal de los gradientes.

Dicho algoritmo es usado en redes neuronales para mejorar el proceso de aprendizaje de un modelo. Este utiliza una estimación del momento y de la magnitud de los gradientes anteriores para actualizar los parámetros del modelo en cada iteración, adaptando la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y de la magnitud del gradiente. De esta forma, se consigue que el modelo se ajuste de manera más eficiente a los datos de entrenamiento, aumentando su precisión de predicción. (Raul Murillo, 2020)

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 110 y 120 encuentra la solución.
  Tarda más en ejecutarse que otros optimizadores (5 min).

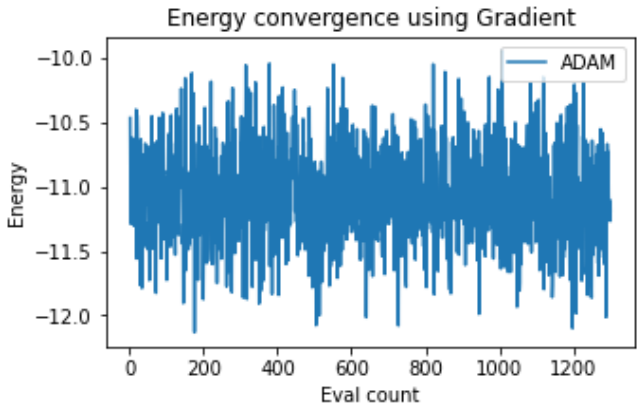

Ilustración 30. ADAM Gráfica (maxiter=100, alpha=0.35).

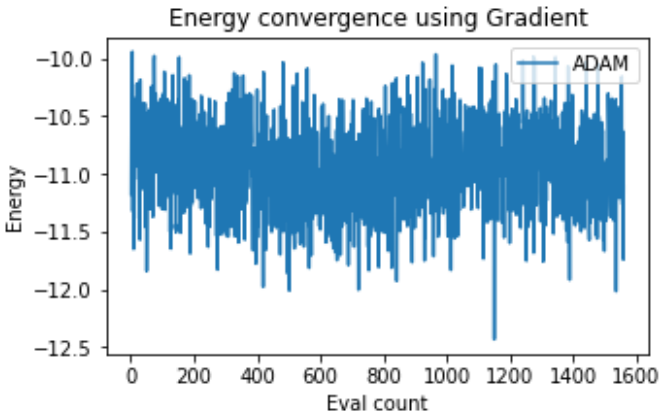

Ilustración 31. ADAM Gráfica (maxiter=120, alpha=0.35).

- Variando *alpha* y *maxiter*.
  Conclusión: Con *alpha* menor de 0.20 no encuentra la solución.

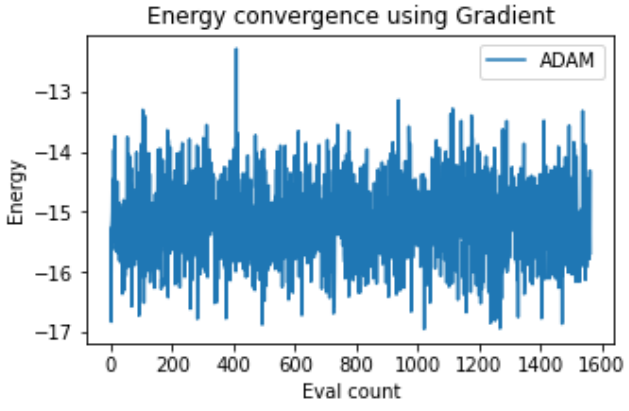

Ilustración 32. ADAM Gráfica (maxiter=120, alpha=0.15).

● 4.6. CG

CG es la abreviatura de Conjugate Gradient optimizer. CG es un algoritmo para la solución numérica de sistemas de ecuaciones lineales cuyas matrices son simétricas y definidas positivas.

Dicho algoritmo iterativo utiliza la suposición inicial para generar una secuencia de mejoras de soluciones aproximadas para un problema, en el cual, cada aproximación se deriva de las anteriores.

Este optimizador es bueno para problemas sin restricciones (debido a esto, no lo podemos aplicar a nuestro problema). (Quintana, 2022)

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 100 y 110 encuentra la solución.
  Tarda más que otros optimizadores en ejecutarse.

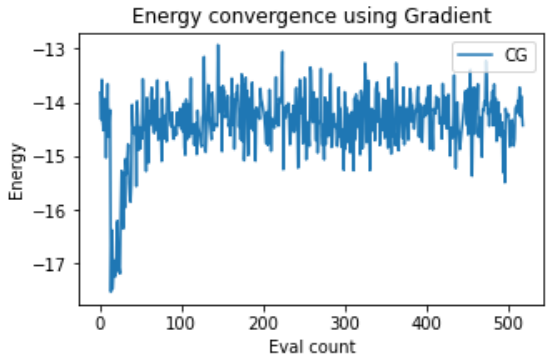

Ilustración 33. CG Gráfica (maxiter=100, alpha=0.35).

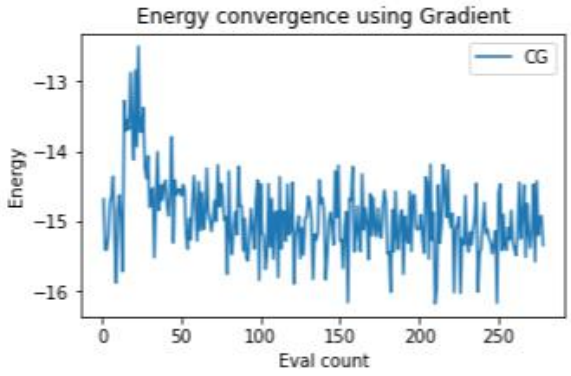

Ilustración 34. CG Gráfica (maxiter=120, alpha=0.35).

- Variando *alpha* y *maxiter*.
  Conclusión: Parece que no afecta tanto el valor de *alpha*, si no el del maxiter a la hora de encontrar la solución.

● 4.7. SLSQP

SLSQP es la abreviatura de Sequential Least SQuares Programming optimizer, es decir, se basa en los cuadrados mínimos para encontrar la solución mínima. (Hwang, 1989)

Este optimizador consiste en minimizar una función de varias variables con cualquier combinación de límites, restricciones de igualdad y desigualdad.

SLSQP se suele usar para problemas en los que la función objetivo y las restricciones son dos veces diferenciables continuamente. Sin embargo, este optimizador realiza una conversión de los valores en flotantes, lo que hace que trabaje con valores infinitos (aumenta el número de qubits).

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 110 y 120 encuentra la solución. A mismo *alpha* que en COBYLA, necesitas mayor *maxiter* para encontrar la solución.

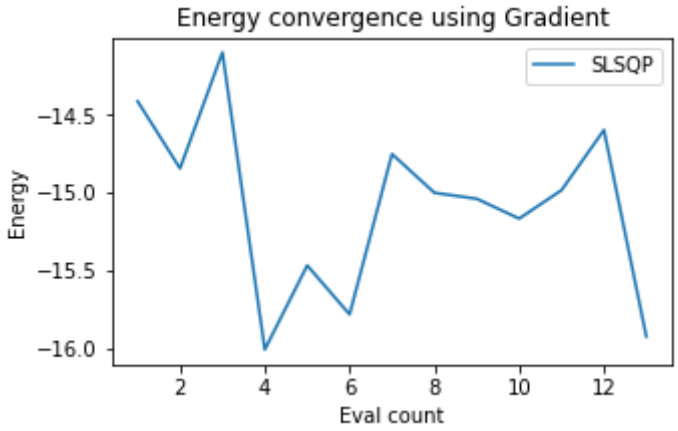

Ilustración 35. SLSQP Gráfica (maxiter=100, alpha=0.35).

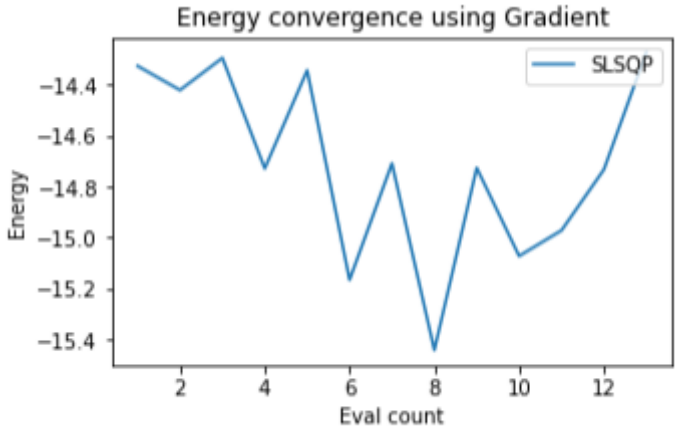

Ilustración 36. SLSQP Gráfica (maxiter=150, alpha=0.35).

- Con *maxiter* = 120 y variando *alpha*.
  Conclusión: A diferencia de otros optimizadores como el COBYLA, parece que, si el *maxiter* es lo suficientemente grande, el *alpha* no afecta para encontrar la solución.

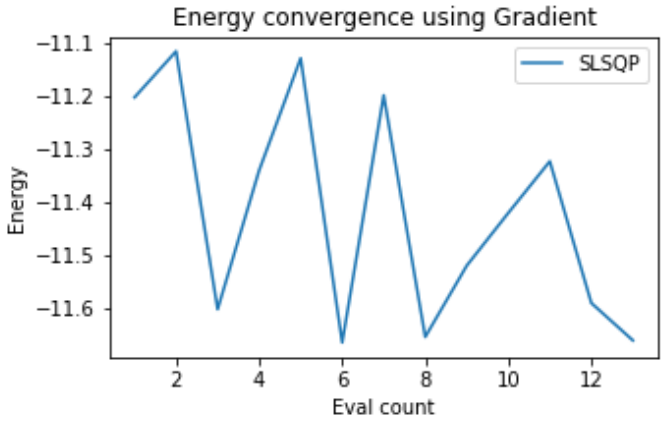

Ilustración 37. SLSQP Gráfica (maxiter=120, alpha=0.65).

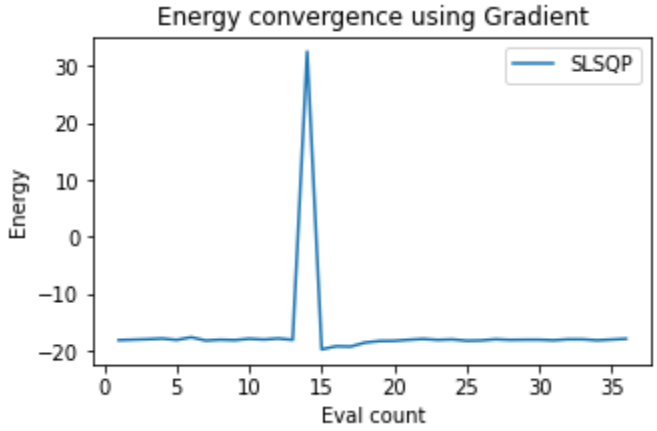

Ilustración 38. SLSQP Gráfica (maxiter=120, alpha=0.15).

- Variando *alpha* y *maxiter*.
  Conclusión: Si se disminuye *alpha*, podemos encontrar la solución con menor *maxiter*.

● 4.8. TNC

TNC es la abreviatura de Truncated Newton, es decir, se basa en un algoritmo de Newton truncado para minimizar una función con variables sujetas a límites. (Stephen G. Nash, 2000)

Este optimizador utiliza información de gradiente, al igual que el CG. Se diferencia del optimizador CG, en que este envuelve una

implementación en C y permite que cada variable tenga límites superiores e inferiores.

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 110 y 120 encuentra la solución. Necesita mayor *maxiter* para encontrar la solución (peor que COBYLA y GSLS), y tarda más que el COBYLA en ejecutarse.

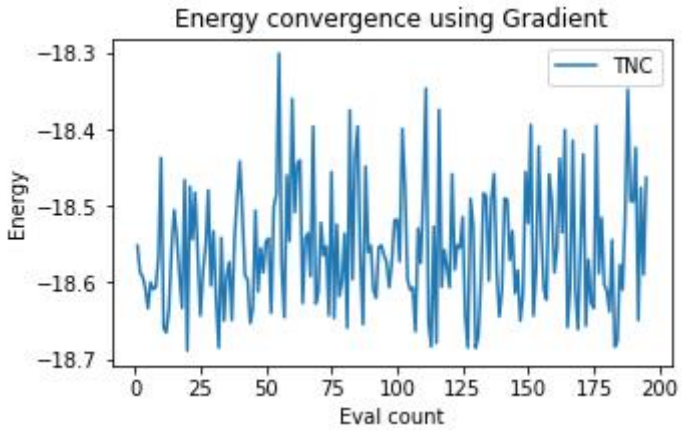

Ilustración 39. TNC Gráfica (maxiter=100, alpha=0.35).

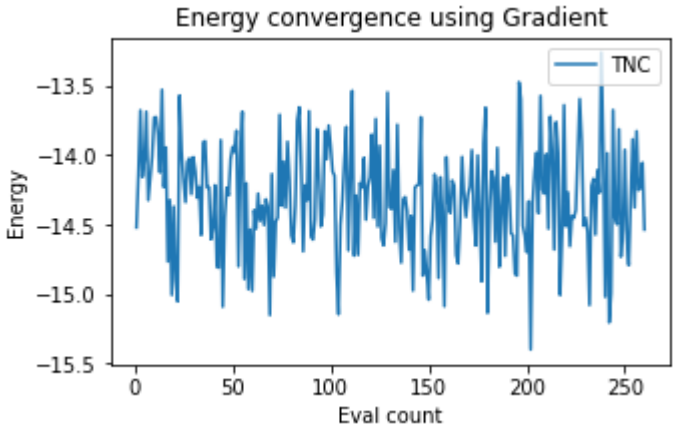

Ilustración 40. TNC Gráfica (maxiter=120, alpha=0.35).

- Variando *alpha* y *maxiter*.
  Conclusión: Disminuyendo *alpha*, se puede encontrar la solución con un menor *maxiter*.

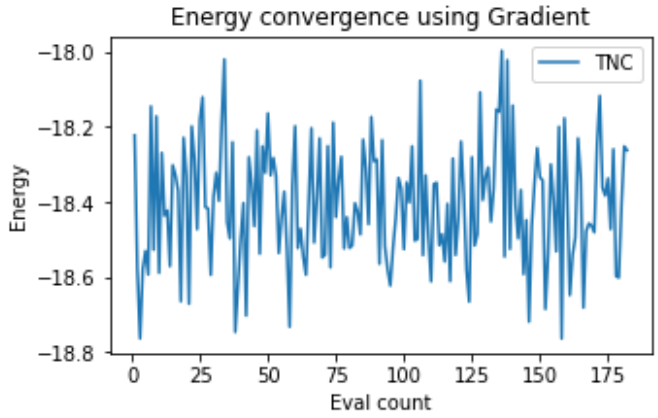

Ilustración 41. TNC Gráfica (maxiter=120, alpha=0.15).

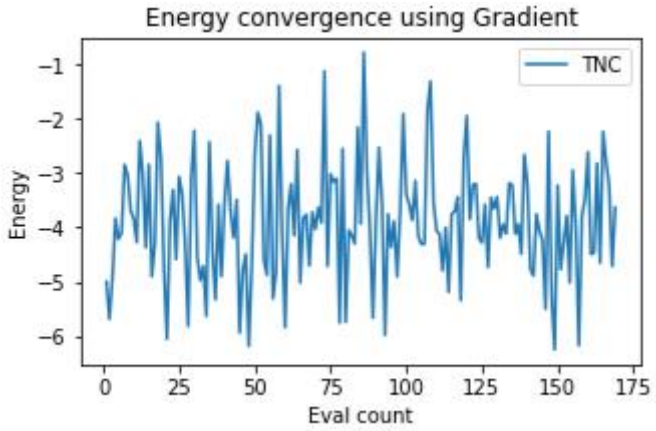

Ilustración 42. TNC Gráfica (maxiter=120, alpha=0.65).

● 4.9. POWELL

Este optimizador realiza una optimización sin restricciones e ignora los límites (debido a esto, no se puede aplicar a nuestro problema).

POWELL es un método de dirección conjunta, es decir, realiza una minimización unidimensional secuencial a lo largo de cada vector direccional, que se actualiza en cada iteración del bucle de minimización principal. Además, la función que se minimiza no necesita ser derivable y no se toman derivadas. (Vassiliadis, 2001)

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 90 y 100 encuentra la solución.
  Tarda más en ejecutarse que otros optimizadores (3 min).

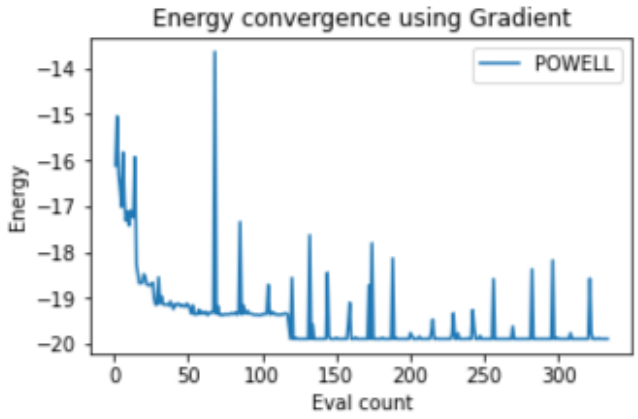

Ilustración 43. POWELL Gráfica (maxiter=100, alpha=0.35).

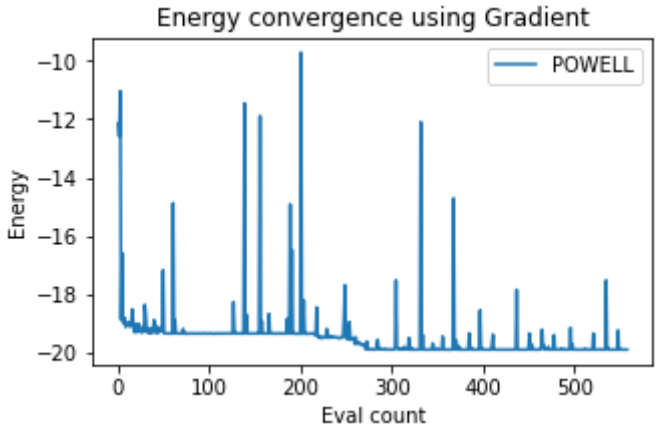

Ilustración 44. POWELL Gráfica (maxiter=90, alpha=0.35).

- Con *maxiter* = 100 y variando *alpha*.
  Conclusión: Cuanto más pequeño sea *alpha*, mejor encuentra la solución.

- Variando *alpha* y *maxiter*.
  Conclusión: Con *alpha* = 0.15, encuentra la solución con *maxiter* = 80.
  Cuanto menor sea *alpha*, más estable parece la gráfica.

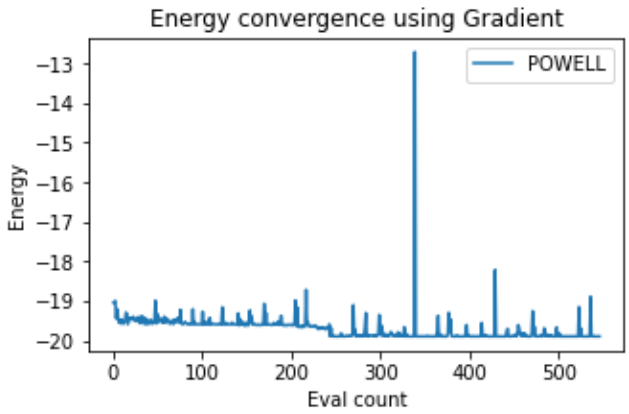

Ilustración 45. POWELL Gráfica (maxiter=90, alpha=0.15).

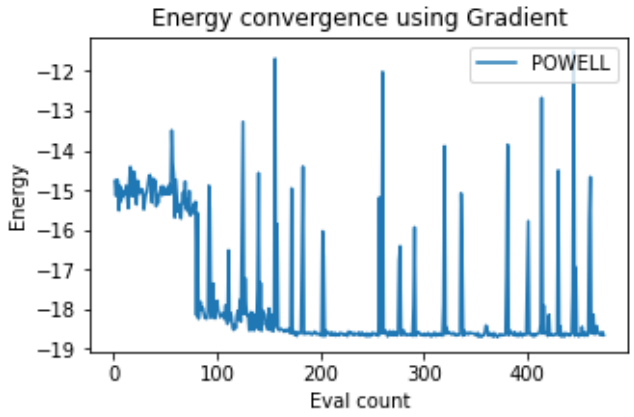

Ilustración 46. POWELL Gráfica (maxiter=90, alpha=0.65).

● 4.10. NFT

NFT es la abreviatura de Nakanishi-Fujii-Todo. Utiliza un método de optimización mínima secuencial para algoritmos híbridos cuánticos-clásicos, que posee la ventaja de converger más rápido, es robusto contra errores estadísticos , y además, está libre de hiperparámetros. (Nakanishi, 2020)

- Con *alpha* = 0.35 y variando *maxiter*.
  Conclusión: Con *maxiter* entre 120 y 130 encuentra la solución.
  Tarda más que otros optimizadores en ejecutarse.

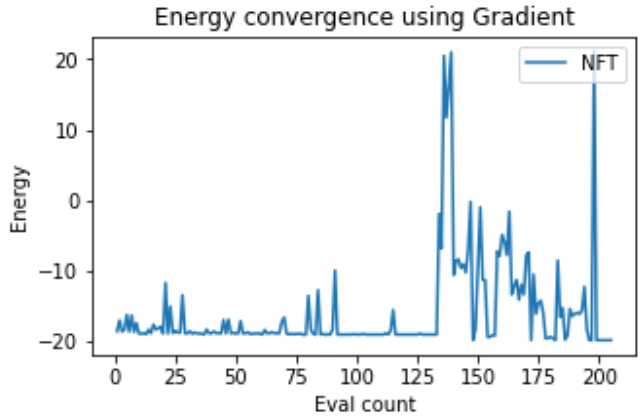

Ilustración 47. NFT Gráfica (maxiter=100, alpha=0.35).

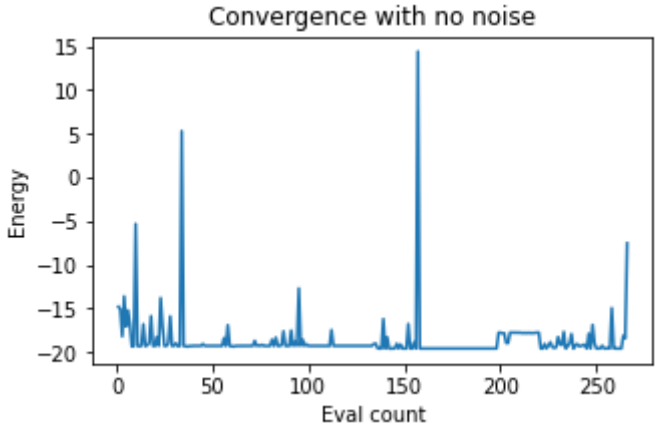

Ilustración 48. NFT Gráfica (maxiter=130, alpha=0.35).

- Con *maxiter* = 130 y variando *alpha*.
  Conclusión: No encuentra la solución con *alphas* muy pequeños (<0.15), ni con *alphas* mayores que 0.5.

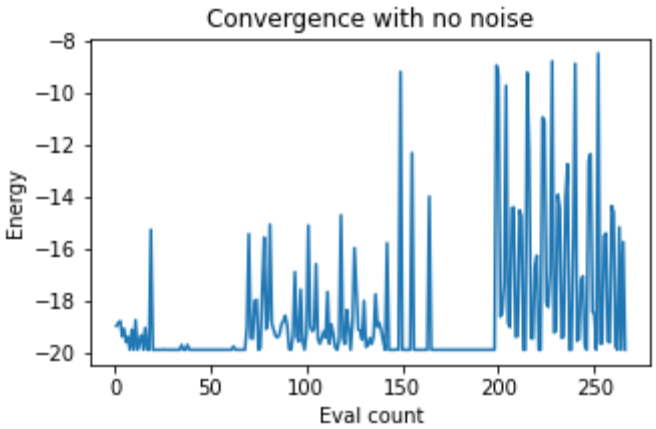

Ilustración 49. NFT Gráfica (maxiter=130, alpha=0.15).

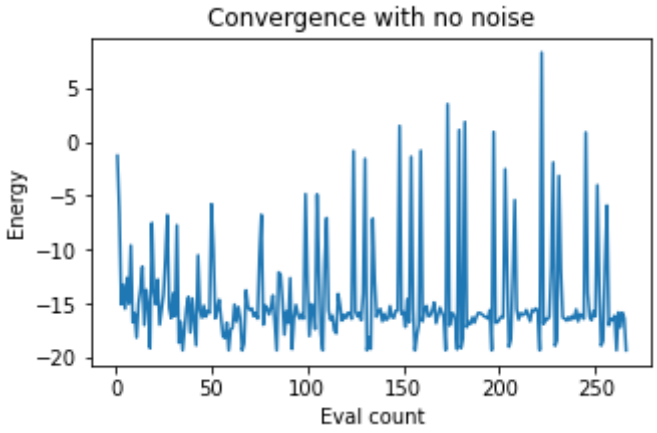

Ilustración 50. NFT Gráfica (maxiter=130, alpha=0.65).

- Variando *alpha* y *maxiter*.
  Conclusión: A mayor *alpha*, peor más inestables las gráficas.
  Con *alpha* menores encuentra mejores resultados.

4.11. Conclusiones.

A continuación, se enumeran los optimizadores que mejores resultados han mostrado al ejecutar el problema:

- COBYLA
- GSLS
- NELDER_MEAD
- L_BFGS_B

De los anteriores, a excepción del NELDER_MEAD, los optimizadores tienden a estabilizar su gráfica a medida que se acercan al mínimo de la solución.

Capítulo 5: Problema con variables enteras

En esta segunda versión, se pueden escoger varias acciones con distinto peso cada una. Es decir, ahora en vez de decidir si se escoge una acción o no (1 o 0), se puede escoger una fracción para cada acción. De esta manera, se aproxima el problema a un problema real, aunque se aumenta considerablemente el número de qubits.

A continuación, se explican los cambios respecto a la anterior versión.

En este caso, el número de acciones es 5 (se ha disminuido 1 para poder adaptarlo al número de qubits). El *Budget* real es 10000 (*budgetR*), un valor más realista, mientras que *budget* será el *Budget* con el que se trabajará. Para ajustar los cálculos, se hará una conversión tras realizar el algoritmo para adaptar las cantidades al *Budget real*.

```
# prepare problem instance
n = 5           # number of assets
q = 0.5          # risk factor
budget = 16  # budget
budgetR = 100000  # budget
penabudget = 16  # budgetlty = 2*n    # scaling of penalty term
```

```
# example instance
s = np.array([ 1, 1, 1,1,1])
sR = np.array([ 103.2, 215.1, 19.5, 45.6,487.3])
mu = np.array([ 0.8750, 0.7667, 0.3622,0.8750, 0.7667])
sigma = np.array([
    [ 3.5067,  0.2012,  1.0922,3.5067,  0.2012],
    [ 0.2012,  0.6231,  0.1509,3.5067,  0.2012],
    [ 1.0922,  0.1509,  0.8992,3.5067,  0.2012],
    [ 2.5067,  0.2012,  1.1922,3.5067,  0.2012],
    [ 0.2042,  0.6231,  0.1559,3.5067,  0.2012]
])
```

Ilustración 51. Problema con variables enteras. Ejemplo parámetros iniciales.

Nuevamente, se indican las restricciones para el algoritmo. Al trabajar con un *Budget* de 16, se le indicará que la suma de las acciones no debe ser superior a éste.

Para poder adaptarse al número de qubits, se limitará el valor a escoger de cada acción entre 0 y *Budget* -1. Se realiza esta resta porque en caso contrario, se necesitaría un qubit más para poder representar el valor. El resto de las restricciones es igual que en la anterior versión.

A diferencia de la versión anterior, en esta no se podrá ejecutar el código de un algoritmo clásico para ver cuál sería el resultado óptimo y comparar resultados, ya que, requeriría mucho tiempo debido a la cantidad de cómputo (ventaja del algoritmo cuántico frente al clásico).

```python
# create docplex model
a=int(budget/np.min(s)) #la accion mas barata vale 1 y el budget es 10
mdl = Model('portfolio_optimization')
x = mdl.integer_var_list(('x{}'.format(i) for i in range(n)), lb=0, ub=(a-1))
objective = mdl.sum([mu[i]*x[i] for i in range(n)])
objective -= q * mdl.sum([sigma[i,j]*x[i]*x[j] for i in range(n) for j in range(n)])
mdl.maximize(objective)
mdl.add_constraint(mdl.sum(s[i]*x[i] for i in range(n)) <= budget)

# case to
qp = from_docplex_mp(mdl)
```

```python
print(qp.export_as_lp_string())
```

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: portfolio_optimization

Maximize
 obj: 0.875000000000 x0 + 0.766700000000 x1 + 0.362200000000 x2
      + 0.875000000000 x3 + 0.766700000000 x4 + [ - 3.506700000000 x0^2
      - 0.402400000000 x0*x1 - 2.184400000000 x0*x2 - 6.013400000000 x0*x3
      - 0.405400000000 x0*x4 - 0.623100000000 x1^2 - 0.301800000000 x1*x2
      - 3.707900000000 x1*x3 - 0.824300000000 x1*x4 - 0.899200000000 x2^2
      - 4.698900000000 x2*x3 - 0.357100000000 x2*x4 - 3.506700000000 x3^2
      - 3.707900000000 x3*x4 - 0.201200000000 x4^2 ]/2
Subject To
 c0: x0 + x1 + x2 + x3 + x4 <= 16

Bounds
       x0 <= 15
       x1 <= 15
       x2 <= 15
       x3 <= 15
       x4 <= 15

Generals
 x0 x1 x2 x3 x4
End
```

Ilustración 52. Problema con variables enteras. Modelo.

De la misma forma que se hacía en la versión binaria, se debe adaptar el circuito, y convertirlo a un problema lineal y eliminar desigualdades.

De forma análoga a la versión anterior, se elige el optimizador (en este caso COBYLA).

Se procede a lanzar el circuito. Todo es igual a la versión anterior, excepto que aquí se ha lanzado en un simulador (ibmq_qasm_simulator, de 32 qubits) en vez de ejecutarlo en local.

Se procede a "deshacer" el cambio que se hizo al principio sobre ajustar el *Budget*, y queda la cantidad a invertir en cada acción.

```
1 res1 = results.x/budget
2 res = res1*budgetR
3 print(res)
```

```
[6250.    0. 6250.    0. 6250.]
```

```
1 cantAcc=(res/sR).astype(int)
2 print(cantAcc)
```

```
[ 60   0 320   0  12]
```

Ilustración 53. Problema con variables enteras. Resultados algoritmo cuántico.

Capítulo 6: Problema con acciones reales

En esta versión final, los datos son extraídos del historial de acciones de IBM. Se combinan la primera y segunda versión. Primero se aplica el algoritmo de variables binarias para escoger las acciones y, después, se aplica el algoritmo sobre variables enteras para la repartición del *Budget*.

Hay que resaltar, que es necesario tener una cuenta en IBM e identificarse con el ID de la cuenta de IBM para poder lanzar el código.

Primero se extraen los datos sobre las acciones. Se usan los datos sobre la primera mitad del 2021. En este caso, se escoge un *Budget* de valor 5 (número de acciones máximo a escoger).

A continuación, se muestran los datos obtenidos en formato tabla y diagrama, para poder apreciar cómo cambió cada acción a lo largo de los meses.

Date	ACS.MC	ACX.MC	AENA.MC	AMS.MC	ANA.MC	BBVA.MC	BKT.MC	CABK.MC	CLNX.MC	COL.MC	...	MAP.MC	MEL.MC	MRL.MC	MTS.M
2021-01-04	22.673164	8.169370	138.500000	58.060001	110.788498	3.527420	2.784623	1.880836	47.286839	7.459796	...	1.293591	5.355	6.376487	19.03960
2021-01-05	23.042517	8.185191	138.300003	58.220001	110.220840	3.523960	2.879495	1.885253	46.584106	7.483210	...	1.305017	5.350	6.503004	19.03767
2021-01-06	23.546179	8.383844	141.100006	59.580002	113.059143	3.727232	3.055136	1.992148	43.948837	7.520673	...	1.340927	5.590	6.490353	19.53683
2021-01-07	23.285954	8.631721	139.300003	56.799999	119.303421	3.764427	3.183342	2.067240	43.236851	7.436382	...	1.344192	5.535	6.549394	20.04854
2021-01-08	23.344715	8.536790	140.699997	57.099998	121.006401	3.698688	3.191675	2.068124	43.865620	7.436382	...	1.341744	5.635	6.498788	19.59476
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2021-04-26	23.348934	10.477608	136.899994	59.700001	136.711700	3.873848	3.743247	2.250112	46.639744	7.928082	...	1.446210	7.000	7.746251	24.89534
2021-04-27	23.417206	10.385313	138.550003	58.980000	138.320068	3.931261	3.784795	2.302234	47.138885	7.942130	...	1.430295	6.954	7.759747	24.64431
2021-04-28	23.246527	10.271045	140.449997	58.360001	137.279358	3.937835	3.866594	2.311953	47.069004	7.885936	...	1.451923	7.046	7.786737	24.33052
2021-04-29	23.101448	10.271045	142.850006	58.119999	137.468567	3.988674	4.117216	2.347290	46.639744	7.960862	...	1.449475	6.980	7.812039	24.03122
2021-04-30	23.152653	10.147985	144.699997	56.700001	136.900909	4.093858	4.108201	2.356124	46.959194	7.899986	...	1.461309	6.816	7.754685	23.38916

83 rows × 27 columns

Ilustración 54. Problema con acciones reales. Ejemplo IBEX acciones.

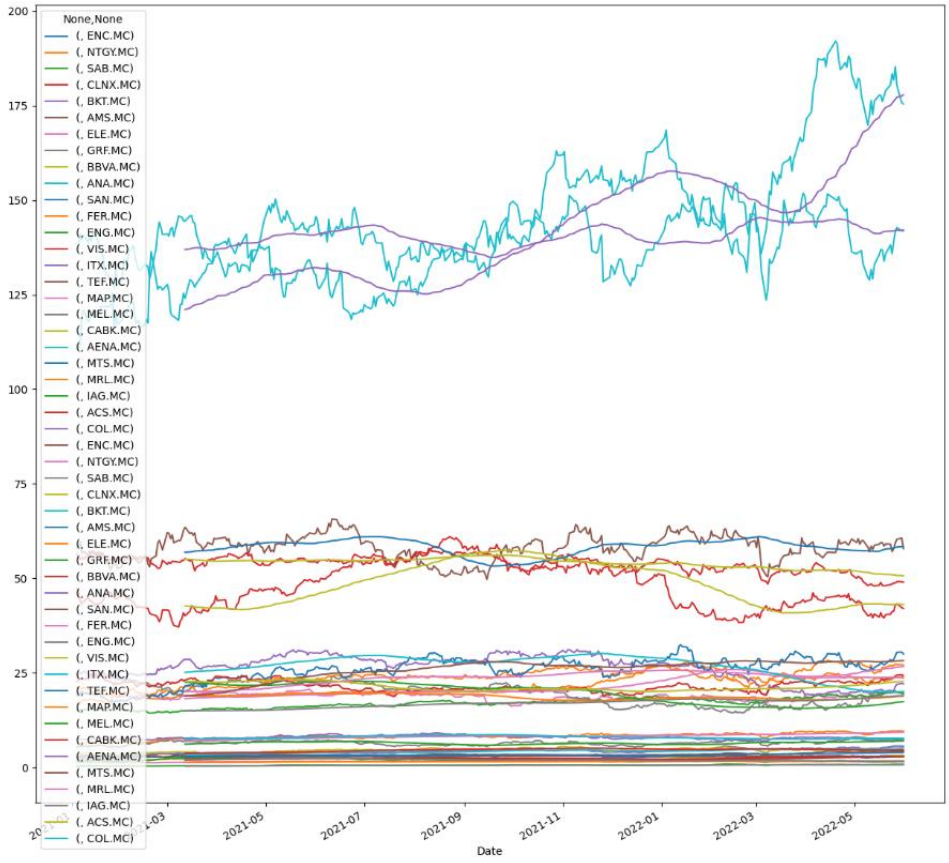

Ilustración 55. Problema con acciones reales. Ejemplo IBEX acciones gráfica.

A continuación, se explica el algoritmo con el VQE.

La función recibe como parámetros de entrada el *Budget*, *maxiter* (número de iteraciones), *alpha* y el *factor de riesgo* (probabilidad de que un rendimiento sea menor a lo esperado, es decir, la inversión realizada no proporcione la rentabilidad esperada o que la pérdida supere la inversión inicial).

En la primera fase (parte binaria), se aplica el modelo con los valores iniciales de mu y *sigma*, y se aplica el VQE. De esta forma, el algoritmo escogerá las acciones candidatas para la siguiente fase (parte entera).

Por tanto, para cada mes, el algoritmo seleccionará 5 de entre todas las acciones que reciba, escogiendo las mejores que crea que pueden dar mejor beneficio.

Como se ha explicado, ahora nos debemos quedar solo con las acciones escogidas en la parte binaria. Por tanto, se recalcula *mu* y *sigma* y el número de acciones para la siguiente fase. Para ello, se calculan dichos valores para el número de acciones candidatas de la fase anterior. Se vuelve a aplicar el modelo con el nuevo valor de *sigma*, *mu* y número de acciones, y se aplica el VQE.

Ahora que nos hemos quedado con los datos de las 5 acciones que interesan, el algoritmo repartirá el *Budget* de la forma más rentable y beneficiosa posible.

Al igual que se hizo en la versión de variables enteras, se debe "deshacer" el cambio para poder obtener los resultados correctos. Se procede a recalcular las acciones totales, es decir, el número de acciones vuelve a ser el original.

● 6.1. Resultados obtenidos.

Se ha ejecutado el código anterior con los siguientes parámetros:

- *alpha* = 0.35
- *budget* = 5
- Optimizador: COBYLA

- Maxiter (número de vueltas): 500

El resultado final y la gráfica son los siguientes:

Ilustración 56. Problema con acciones reales. Resultados gráfica.

1	Stat,quantumOptimizer
2	Start,2021-01-03
3	End,2022-05-31
4	Risk-free rate,0.00%
5	,,,,,,,,,,,,,,,,,,,
6	Total Return,-3.49%
7	Daily Sharpe,0.02
8	Daily Sortino,0.03
9	CAGR,-2.50%
10	Max Drawdown,-32.15%
11	Calmar Ratio,-0.08
12	,,,,,,,,,,,,,,,,,,,
13	MTD,3.48%
14	3m,6.17%
15	6m,11.91%
16	YTD,4.48%
17	1Y,-14.11%
18	3Y (ann.),-
19	5Y (ann.),-
20	10Y (ann.),-
21	Since Incep. (ann.),-2.50%
22	,,,,,,,,,,,,,,,,,,,
23	Daily Sharpe,0.02
24	Daily Sortino,0.03
25	Daily Mean (ann.),0.43%
26	Daily Vol (ann.),24.13%
27	Daily Skew,-0.13
28	Daily Kurt,3.48
29	Best Day,7.80%
30	Worst Day,-6.04%
31	,,,,,,,,,,,,,,,,,,,
32	Monthly Sharpe,-0.05
33	Monthly Sortino,-0.09
34	Monthly Mean (ann.),-0.98%
35	Monthly Vol (ann.),19.17%
36	Monthly Skew,0.88
37	Monthly Kurt,2.31
38	Best Month,14.24%
39	Worst Month,-9.55%
40	,,,,,,,,,,,,,,,,,,,
41	Yearly Sharpe,-
42	Yearly Sortino,-
43	Yearly Mean,4.48%
44	Yearly Vol,-
45	Yearly Skew,-
46	Yearly Kurt,-
47	Best Year,4.48%
48	Worst Year,4.48%
49	,,,,,,,,,,,,,,,,,,,
50	Avg. Drawdown,-9.91%
51	Avg. Drawdown Days,101.75
52	Avg. Up Month,4.64%
53	Avg. Down Month,-2.91%
54	Win Year %,100.00%
55	Win 12m %,0.00%

Ilustración 57. Problema con acciones reales. Resultados finales.

De la Ilustración 57 se pueden observar datos como las ganancias/pérdidas al año, las ganancias que se consiguieron en el mejor mes, el período en el que se lanza el algoritmo, etc.

Por último, se muestra una sección de la ejecución del código, para ver los resultados intermedios.

```
Phase 1 - Problem solved:
##########
Phase 1 - results:
[0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0]
Phase 2 - Num Qubits:
            37
Phase 2 - Optimization parameters:
            74
Phase 1 - Problem solved:
##########
Phase 1 - results:
[0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0]
Phase 2 - Num Qubits:
            37
Phase 2 - Optimization parameters:
            74
Phase 1 - Problem solved:
##########
Phase 1 - results:
[0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
Phase 2 - Num Qubits:
            11
Phase 2 - Optimization parameters:
            22
```

Ilustración 58. Problema con acciones reales. Resultados iteración.

La Phase 1 corresponde con los resultados obtenidos en la selección de acciones mediante acciones binarias. Se puede observar un array. Cada casilla corresponde a una acción, respectivamente. Si el valor es 1, la acción fue elegida.

Por el contrario, la Phase 2 corresponde con los resultados obtenidos en la parte de acciones enteras. Aquí, el algoritmo decidía que peso dar a cada acción (cuánto dinero repartir).

Dicho proceso, se repite hasta completar el periodo de tiempo indicado. Siendo un mes por cada par de fases.

Capítulo 7: Comparación VQE con algoritmos clásicos

En este apartado, se ejecuta el algoritmo anterior junto con varios algoritmos clásicos, y se comparan los resultados.

Por el elevado tiempo de ejecución, se ha tenido que disminuir el *Budget* a 4 (número de acciones a escoger en la fase del VQE).

Como se explicó en capítulos anteriores, los algoritmos clásicos con los que se han realizado el estudio del VQE han sido los siguientes:

- SMA Strategy
- Mean-Variance optimization
- Convex optimization

Se ha ejecutado el VQE con los siguientes parámetros:

- *alpha* = 0.35
- *budget* = 4
- Optimizador: COBYLA
- *Maxiter* (número de vueltas): 250

7.1. Resultados obtenidos

Se han realizado pruebas en distintos escenarios: usando 27 acciones y 15 acciones del historial de IBM. Cabe destacar que, para las ejecuciones de 27 acciones, el tiempo ha sido de 3-4 días, mientras que en el caso de 15 acciones el tiempo de ejecución ha sido de 1-2 días.

El backend utilizado en ambos casos ha sido *ibmq_qasm_simulator*, y se han realizado las pruebas en entornos sin ruido.

A continuación, se muestran algunos de los resultados obtenidos.

Estudio con 27 acciones

En las siguientes gráficas se pueden apreciar los siguientes algoritmos, VQE (rojo), SMA Strategy o above50sma (azul), Convex optimization (verde), Mean-Variance optimization o classic optimizar (naranja).

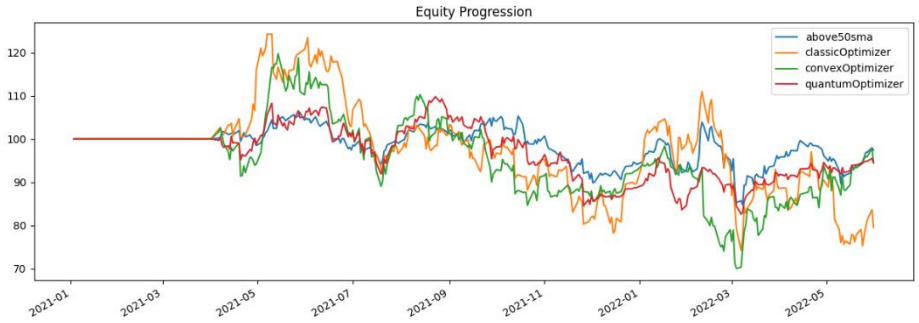

Ilustración 59. Comparación VQE con algoritmos clásicos (27 acciones I).

En esta gráfica se puede apreciar cómo el VQE (rojo) predice de forma correcta en qué acciones invertir comparado con los otros algoritmos clásicos e, incluso, está por encima en algunos meses.

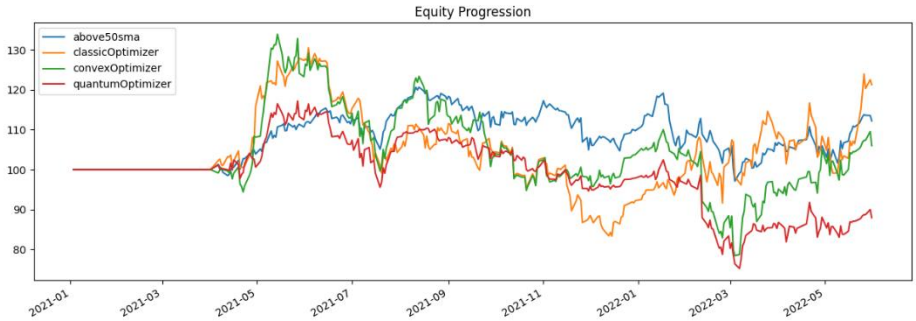

Ilustración 60. Comparación VQE con algoritmos clásicos (27 acciones II).

En esta gráfica el VQE (rojo) presenta un peor comportamiento comparado con los resultados anteriores. Esto puede ser debido a la elección de las 4 acciones que realiza en la primera fase, o modificando algún parámetro adicional como puede ser *alpha*, *maxiter, factor de riesgo*, etc., se puede llegar a controlar estas pérdidas.

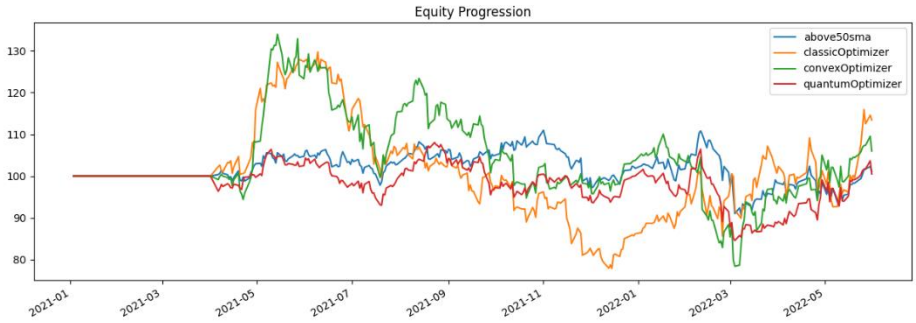

Ilustración 61. Comparación VQE con algoritmos clásicos (27 acciones III).

En esta otra gráfica se puede apreciar que, mientras que algoritmos clásicos como el Convex optimization (verde) y el Mean-Variance optimization (naranja) presentan una amplia fluctuación a lo largo del intervalo de los meses, el VQE (rojo) apenas presenta inestabilidad.

Estudio con 15 acciones

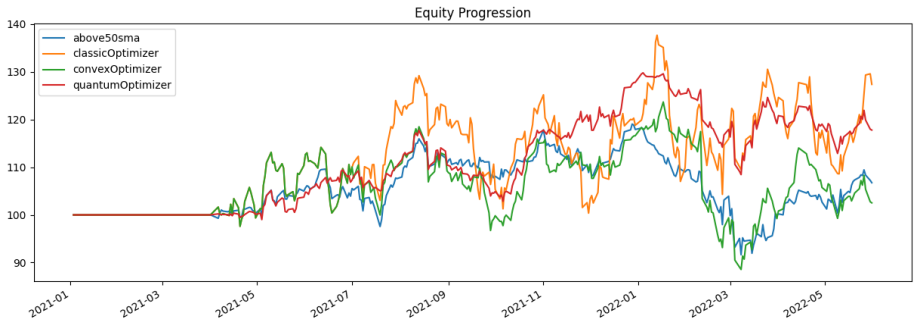

Ilustración 62. Comparación VQE con algoritmos clásicos (15 acciones I).

En esta gráfica se puede observar como el VQE (rojo) presenta una tendencia semejante al algoritmo clásico Mean-Variance optimization (naranja) a lo largo de los meses, indicando que, en caso de haber invertido en cualquiera de ellos, se hubieran obtenido beneficios semejantes.

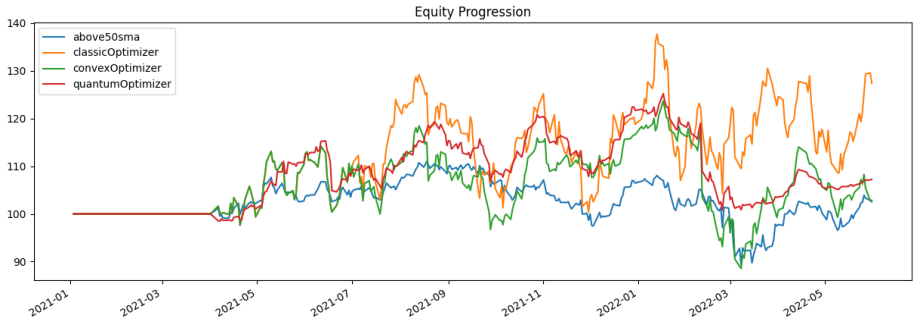

Ilustración 63. Comparación VQE con algoritmos clásicos (15 acciones II).

En estas gráficas el VQE sigue las mismas tendencias que el resto de los algoritmos clásicos, a excepción de los últimos meses del año.

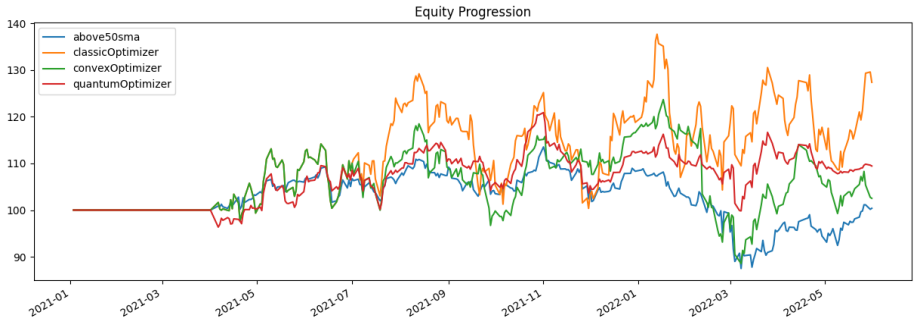

Ilustración 64. Comparación VQE con algoritmos clásicos (15 acciones III).

Capítulo 8: Conclusiones y Trabajo Futuro

En este capítulo se describe la retrospectiva del proyecto, haciendo ver la conclusión, así como posibles mejoras futuras.

8.1. Conclusiones

Se ha implementado un algoritmo cuántico usando el VQE, que recibe una serie de acciones del historial de IBM, y selecciona cada mes cuáles son las cuatro acciones candidatas más rentables para invertir, y cuánto se debe invertir en cada una de ellas.

Se han realizado pruebas entre distintos optimizadores, siendo el COBYLA el que mejores resultados ha presentado, y con el que se ha decidido realizar el resto de las ejecuciones.

Además, se ha realizado un estudio del algoritmo VQE junto a otros algoritmos clásicos, mostrando cómo se comportan en distintos escenarios. Como resultado, se obtiene que el VQE tiene un comportamiento semejante en la mayoría de los casos, siendo este más complejo y completo que el algoritmo binario presentado en el Capítulo 3.

8.2. Trabajo Futuro

Como propuestas para un futuro, se enumeran algunas líneas de trabajo para mejorar el proyecto:

Realizar el problema de acciones reales con los optimizadores propuestos en el Capítulo 4, y estudiar cuál se adecúa mejor para nuestro problema.

Ampliar/disminuir el número de acciones de la selección de IBM, y estudiar cómo se comportan los algoritmos.

Realizar cambios en los parámetros del código como pueden ser *alpha*, *factor de riesgo*, *maxiter*, etc

En caso de disponer de un computador potente, se podría ampliar el número de acciones a escoger (*budget*), aunque actualmente estamos limitados por el número de qubits que nos ofrecen.

Chapter 8: Conclusions and Future Work

This chapter describes the retrospective of the project, showing the conclusion, as well as possible future improvements.

8.1. Conclusions

A quantum algorithm has been implemented using VQE, which receives a series of stocks from IBM's history, and selects each month which are the four most profitable candidate stocks to invest, and how much we should invest in each of them.

Tests have been performed among different optimizers, being COBYLA the one who has presented the best results and use it to carry out the rest of the executions.

Furthermore, a study of the VQE algorithm has been carried out with other classical algorithms, showing how they behave in different scenarios. In conclusion, we obtained that the VQE has a similar behave in most cases, being the VQE more complex and complete than the binary algorithm presented in Chapter 3.

8.2. Future Work

As proposals for the future, listed below are some lines of work to improve the project:

Repeat the problem of real actions with the optimizers proposed in Chapter 4, and study which one is best suited for our problem.

Extend/decrease the number of stocks in the IBM selection, and study how the algorithms behave.

Make changes to code parameters such as *alpha*, *risk factor, maxiter*, etc.

In case of having a powerful computer, the number of stocks to choose (*budget*) could be increased, although we are currently limited by the number of qubits that they offer us.

Bibliografía

Carrascal de las Heras, G., Hernamperez Manso, P., Botella, G., & del Barrio, A. A. (2023). *Backtesting quantum computing algorithms for portfolio optimization. TechRxiv. Preprint* .

Carrascal, G. d. (2021). *First experiences of teaching quantum computing. J Supercomput 77, 2770–2799* .

Carrascal, G. R. (2023). *Differential Evolution VQE for Crypto-currency Arbitrage. Quantum Optimization with many local minima.*

Eric R. Johnston, N. H.-S. (2019). *Programming Quantum Computers, 1st edition, O'Reilly, .*

Gines Carrascal, G. B. (2023). *EPJ Quantum Technol., 10 1, 13.*

Grover., L. K. (1996). *A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing (STOC '96). Association for Computing Machinery, New York, NY, USA, 212–219.*

Hidary, J. D. (2019). *Quantum Computing: An Applied Approach, 1st edition, Springer.*

Jules Tilly, H. C. (2022). *The Variational Quantum Eigensolver: A review of methods and best practices, Physics Reports, Volume 986. Pages 1-128.*

Quintana, D. M. (2022). *"Leveraging Posits for the Conjugate Gradient Linear Solver on an Application-Level RISC-V Core," KTH Royal Institute of Technology, Tech. Rep.,.*

Raul Murillo, A. A. (2020). *Deep PeNSieve: A deep learning framework based on the posit number system, Digital Signal Processing, Volume 102, 102762.*

Shor, P. (1995). *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Rev., 41, 303-332.*

Nesterov, Y. and Spokoiny, V. (2017) Random gradient-free mini-mization of convex functions. Foundations of Computa-tional Mathematics, 17(2):527–566.

Gonglin Yuan, Xiwen Lu, (2011) An active set limited memory BFGS algorithm for bound constrained optimization, Applied Mathematical Modelling, Volume 35, Issue 7, Pages 3561-3573, ISSN 0307-904X,

Nelder, J.A. and Mead, R. (1965), "A simplex method for function minimization", Comput. J., 7, pp. 308–313

D.M. Hwang, C.T. Kelley, (1989) Sequential Quadratic Programming for Parameter Identification Problems, IFAC Proceedings Volumes, Volume 22, Issue 4, Pages 259-263, ISSN 1474-6670,

Stephen G. Nash, (2000) A survey of truncated-Newton methods, Journal of Computational and Applied Mathematics, Volume 124, Issues 1–2, Pages 45-59, ISSN 0377-0427

Vassiliadis, V.S., Conejeros, R. (2001). Powell Method . In: Floudas, C.A., Pardalos, P.M. (eds) Encyclopedia of Optimization. Springer, Boston, MA.

Nakanishi, K. M., Fujii, K., & Todo, S. (2020). Sequential minimal optimization for quantum-classical hybrid algorithms. Physical Review Research, 2(4), 043158.

Gines Carrascal, G. B. (2021) Paper: Portfolio using COBYLA Optimizer.