IBM Quantum Challenge Fall 2021

Challenge 1: Optimizing your portfolio with quantum computers

We recommend that you switch to **light** workspace theme under the Account menu in the upper right corner for optimal experience.

Introduction: What is portfolio optimization?

Portfolio optimization is a crucial process for anyone who wants to maximize returns from their investments. Investments are usually a collection of so-called assets (stock, credits, bonds, derivatives, calls, puts, etc..) and this collection of assets is called a **portfolio**.

The goal of portfolio optimization is to minimize risks (financial loss) and maximize returns (financial gain). But this process is not as simple as it may seem. Gaining high returns with little risk is indeed too good to be true. Risks and returns usually have a trade-off relationship which makes optmizing your portfolio a little more complicated. As Dr. Harry Markowitz states in his Moderbn Portfolio Theory he created in 1952, "risk is an inherrent part of higher reward."

Modern Portfolio Theory (MPT)
An investment theory based on the idea that investors are risk-averse, meaning that when given two portfolios that offer the same expected return they will prefer the less risky one. Investors can construct portfolios to maximize expected return based on a given level of market risk, emphasizing that risk is an inherent part of higher reward. It is one of the most important and influential economic theories dealing with finance and investment. Dr. Harry Markowitz created the modern portfolio theory (MPT) in 1952 and won the Nobel Prize in Economic Sciences in 1990 for it.

Reference: Modern Portfolio Theory

Challenge

***Goal** Portfolio optimization is a crucial process for anyone who wants to maximize returns from their investments. In this first challenge, you will learn some of the basic theory behind portfolio optimization and how to formulate the problem so it can be solved by quantum computers. During the process, you will learn about Qiskit's Finance application class and methods to solve the problem efficiently. 1. **Challenge 1a**: Learn how to use the PortfolioOptimization() method in Qiskit's Finance module to convert the portfolio optimization into a*

*quadratic program. 2. **Challenge 1b**: Implement VQE to solve a four-stock portfolio optimization problem based on the instance created in challenge 1a. 3. **Challenge 1c**: Solve the same problem using QAOA with three budgets and double weights for any of the assets in your portfolio.*

*Before you begin, we recommend watching the [**Qiskit Finance Demo Session with Julien Gacon**](https://youtu.be/UtMVoGXlz04?t=2022) and check out the corresponding [**demo notebook**](https://github.com/qiskit-community/qiskit-application-modules-demo-sessions/tree/main/qiskit-finance) to learn about Qiskit's Finance module and its appications in portfolio optimization.*

1. Finding the efficient frontier

The Modern portfolio theory (MPT) serves as a general framework to determine an ideal portfolio for investors. The MPT is also referred to as mean-variance portfolio theory because it assumes that any investor will choose the optimal portfolio from the set of portfolios that

- *Maximizes expected return for a given level of risk; and*
- *Minimizes risks for a given level of expected returns.*

The figure below shows the minimum variance frontier of modern portfolio theory where the horizontal axis shows the risk and the vertical axis shows expected return.

Consider a situation where you have two stocks to choose from: A and B. You can invest your entire wealth in one of these two stocks. Or you can invest 10% in A and 90% in B, or 20% in A and 80% in B, or 70% in A and 30% in B, etc ... There is a huge number of possible combinations and this is a simple case when considering two stocks. Imagine the different combinations you have to consider when you have thousands of stocks.

The minimum variance frontier shows the minimum variance that can be achieved for a given level of expected return. To construct a minimum-variance frontier of a portfolio:

- *Use historical data to estimate the mean, variance of each individual stock in the portfolio, and the correlation of each pair of stocks.*
- *Use a computer program to find out the weights of all stocks that minimize the portfolio variance for each pre-specified expected return.*
- *Calculate the expected returns and variances for all the minimum variance portfolios determined in step 2 and then graph the two variables.*

Investors will never want to hold a portfolio below the minimum variance point. They will always get higher returns along the positively sloped part of the minimum-variance frontier. And the positively sloped part of the minimum-variance frontier is called the efficient frontier.

The efficient frontier is where the optimal portfolios are. And it helps narrow down the different portfolios from which the investor may choose.

2. Goal Of Our Exercise

The goal of this exercise is to find the efficent frontier for an inherent risk using a quantum approach. We will use Qiskit's Finance application modules to convert our portfolio optimization problem into a quadratic program so we can then use variational quantum algorithms such as VQE and QAOA to solve our optimization problem. Let's first start by looking at the actual problem we have at hand.

3. Four-Stock Portfolio Optimization Problem

Let us consider a portfolio optimization problem where you have a total of four assets (e.g. STOCK0, STOCK1, STOCK2, STOCK3) to choose from. Your goal is to find out a combination of two assets that will minimize the tradeoff between risk and return which is the same as finding the efficient frontier for the given risk.

4. Formulation

How can we formulate this problem?
The function which describes the efficient frontier can be formulated into a quadratic program with linear constraints as shown below.
The terms that are marked in red are associated with risks and the terms in blue are associated with returns. You can see that our goal is to minimize the tradeoff between risk and return. In general, the function we want to optimize is called an objective function.

$$\min_{x \in \{0, 1\}^n}: $ $q x^n\Sigma x$ - $\mu^n x$$$

$$subject$ $to: 1^n x = B$$

- x *indicates asset allocation.*
- $Σ$ *(sigma) is a covariance matrix. A covariance matrix is a useful math concept that is widely applied in financial engineering. It is a statistical measure of how two asset prices are varying with respect to each other. When the covariance between two stocks is high, it means that one stock experiences heavy price movements and is volatile if the price of the other stock changes.*
- q *is called a risk factor (risk tolerance), which is an evaluation of an individual's willingness or ability to take risks. For example, when you use the automated financial advising services, the so-called robo-advising, you will*

usually see different risk tolerance levels. This q value is the same as such and takes a value between 0 and 1.

- *μ (mu) is the expected return and is something we obviously want to maximize.*
- *n is the number of different assets we can choose from*
- *B stands for Budget. And budget in this context means the number of assets we can allocate in our portfolio.*

Goal:

Our goal is to find the x value. The x value here indicates which asset to pick (x[i]=1) and which not to pick (x[i]=0).

Assumptions:

We assume the following simplifications:

- *all assets have the same price (normalized to 1),*
- *the full budget B has to be spent, i.e. one has to select exactly B assets.*
- *the equality constraint $1^n x = B$ is mapped to a penalty term $(1^n x - B)^2$ which is scaled by a parameter and subtracted from the objective function.*

Step 1. Import necessary libraries

In [2]:
```
#Let us begin by importing necessary libraries.
from qiskit import Aer
from qiskit.algorithms import VQE, QAOA, NumPyMinimumEigensolver
from qiskit.algorithms.optimizers import *
from qiskit.circuit.library import TwoLocal
from qiskit.utils import QuantumInstance
from qiskit.utils import algorithm_globals
from qiskit_finance import QiskitFinanceError
from qiskit_finance.applications.optimization import PortfolioOptimization
from qiskit_finance.data_providers import *
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit_optimization.applications import OptimizationApplication
from qiskit_optimization.converters import QuadraticProgramToQubo
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import datetime
import warnings
from sympy.utilities.exceptions import SymPyDeprecationWarning
warnings.simplefilter("ignore", SymPyDeprecationWarning)
```

```
{'1': 1024}
```

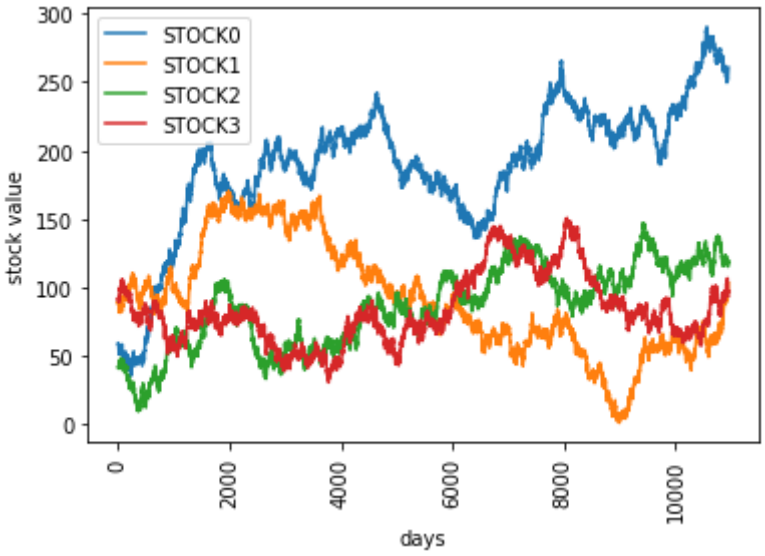
Step 2. Generate time series data (Financial Data)

Let's first generate a random time series financial data for a total number of stocks n=4. We use RandomDataProvider for this. We are going back in time and retrieve financial data from November 5, 1955 to October 26, 1985.

In [2]:
```
# Set parameters for assets and risk factor
num_assets = 4     # set number of assets to 4
q = 0.5                   # set risk factor to 0.5
budget = 2           # set budget as defined in the problem
seed = 132     #set random seed

# Generate time series data
stocks = [("STOCK%s" % i) for i in range(num_assets)]
data = RandomDataProvider(tickers=stocks,
                  start=datetime.datetime(1955,11,5),
                  end=datetime.datetime(1985,10,26),
                  seed=seed)
data.run()
```

In [19]:
```
# Let's plot our finanical data
for (cnt, s) in enumerate(data._tickers):
    plt.plot(data._data[cnt], label=s)
plt.legend()
plt.xticks(rotation=90)
plt.xlabel('days')
plt.ylabel('stock value')
plt.show()
```



***WARNING** Please do not change the start/end dates that are given to the RandomDataProvider in this challenge. Otherwise, your answers will not be graded properly.*

Step 3. Quadratic Program Formulation

Let's generate the expected return first and then the covariance matrix which are both needed to create our portfolio.

Expected Return μ

Expected return of a portfolio is the anticipated amount of returns that a portfolio may generate, making it the mean (average) of the portfolio's possible return distribution. For example, let's say stock A, B and C each weighted 50%, 20% and

30% respectively in the portfolio. If the expected return for each stock was 15%, 6% and 9% respectively, the expected return of the portfolio would be:

$$\mu = (50\% \times 15\%) + (20\% \times 6\%) + (30\% \times 9\%) = 11.4\%$$

For the problem data we generated earlier, we can calculate the expected return over the 30 years period from 1955 to 1985 by using the following `get_period_return_mean_vector()` *method which is provided by Qiskit's RandomDataProvider.*

In [58]:
```
from abc import ABC, abstractmethod
from typing import Tuple, Optional, List, cast
import logging
from enum import Enum

import numpy as np
import fastdtw

from qiskit.utils import algorithm_globals


class StockMarket(Enum):
    """Stock Market enum"""

    LONDON = "XLON"
    EURONEXT = "XPAR"
    SINGAPORE = "XSES"


class BaseDataProvider(ABC):
    """The abstract base class for data_provider modules within Qiskit's finance

    To create add-on data_provider module subclass the BaseDataProvider class in
    Doing so requires that the required driver interface is implemented.

    To use the subclasses, please see
    https://github.com/Qiskit/qiskit-finance/blob/main/docs/tutorials/11_time_ser

    """

    @abstractmethod
    def __init__(self) -> None:
        self._data: Optional[List] = None
        self._n = 0  # pylint: disable=invalid-name
        self.period_return_mean: Optional[np.ndarray] = None
        self.cov: Optional[np.ndarray] = None
        self.period_return_cov: Optional[np.ndarray] = None
        self.rho: Optional[np.ndarray] = None
        self.mean: Optional[np.ndarray] = None

    @abstractmethod
    def run(self) -> None:
        """Loads data."""
        pass

    # it does not have to be overridden in non-abstract derived classes.
    def get_mean_vector(self) -> np.ndarray:
        """Returns a vector containing the mean value of each asset.

        Returns:
            a per-asset mean vector.
        Raises:
```

```
            QiskitFinanceError: no data loaded
        """
        try:
            if not self._data:
                raise QiskitFinanceError(
                    "No data loaded, yet. Please run the method run() first to lo
                )
        except AttributeError as ex:
            raise QiskitFinanceError(
                "No data loaded, yet. Please run the method run() first to load t
            ) from ex
        self.mean = cast(np.ndarray, np.mean(self._data, axis=1))
        return self.mean

    @staticmethod
    def _divide(val_1, val_2):
        if val_2 == 0:
            if val_1 == 0:
                return 1
            logger.warning("Division by 0 on values %f and %f", val_1, val_2)
            return np.nan
        return val_1 / val_2

    # it does not have to be overridden in non-abstract derived classes.
    def get_period_return_mean_vector(self) -> np.ndarray:
        """
        Returns a vector containing the mean value of each asset.

        Returns:
            a per-asset mean vector.
        Raises:
            QiskitFinanceError: no data loaded
        """
        try:
            if not self._data:
                raise QiskitFinanceError(
                    "No data loaded, yet. Please run the method run() first to lo
                )
        except AttributeError as ex:
            raise QiskitFinanceError(
                "No data loaded, yet. Please run the method run() first to load t
            ) from ex
        _div_func = np.vectorize(BaseDataProvider._divide)
        period_returns = _div_func(np.array(self._data)[:, 1:], np.array(self._da
        self.period_return_mean = cast(np.ndarray, np.mean(period_returns, axis=1
        return self.period_return_mean

    # it does not have to be overridden in non-abstract derived classes.
    def get_covariance_matrix(self) -> np.ndarray:
        """
        Returns the covariance matrix.

        Returns:
            an asset-to-asset covariance matrix.
        Raises:
            QiskitFinanceError: no data loaded
        """
        try:
            if not self._data:
                raise QiskitFinanceError(
                    "No data loaded, yet. Please run the method run() first to lo
                )
        except AttributeError as ex:
            raise QiskitFinanceError(
```

```
                "No data loaded, yet. Please run the method run() first to load t
            ) from ex
        self.cov = np.cov(self._data, rowvar=True)
        return self.cov

    # it does not have to be overridden in non-abstract derived classes.
    def get_period_return_covariance_matrix(self) -> np.ndarray:
        """
        Returns a vector containing the mean value of each asset.

        Returns:
            a per-asset mean vector.
        Raises:
            QiskitFinanceError: no data loaded
        """
        try:
            if not self._data:
                raise QiskitFinanceError(
                    "No data loaded, yet. Please run the method run() first to lo
                )
        except AttributeError as ex:
            raise QiskitFinanceError(
                "No data loaded, yet. Please run the method run() first to load t
            ) from ex
        _div_func = np.vectorize(BaseDataProvider._divide)
        period_returns = _div_func(np.array(self._data)[:, 1:], np.array(self._da
        self.period_return_cov = np.cov(period_returns)
        return self.period_return_cov

    # it does not have to be overridden in non-abstract derived classes.
    def get_similarity_matrix(self) -> np.ndarray:
        """
        Returns time-series similarity matrix computed using dynamic time warping

        Returns:
            an asset-to-asset similarity matrix.
        Raises:
            QiskitFinanceError: no data loaded
        """
        try:
            if not self._data:
                raise QiskitFinanceError(
                    "No data loaded, yet. Please run the method run() first to lo
                )
        except AttributeError as ex:
            raise QiskitFinanceError(
                "No data loaded, yet. Please run the method run() first to load t
            ) from ex
        self.rho = np.zeros((self._n, self._n))
        for i_i in range(0, self._n):
            self.rho[i_i, i_i] = 1.0
            for j_j in range(i_i + 1, self._n):
                this_rho, _ = fastdtw.fastdtw(self._data[i_i], self._data[j_j])
                this_rho = 1.0 / this_rho
                self.rho[i_i, j_j] = this_rho
                self.rho[j_j, i_i] = this_rho
        return self.rho

    # gets coordinates suitable for plotting
    # it does not have to be overridden in non-abstract derived classes.
    def get_coordinates(self) -> Tuple[np.ndarray, np.ndarray]:
        """Returns random coordinates for visualisation purposes."""
        # Coordinates for visualisation purposes
        x_c = np.zeros([self._n, 1])
```

```
        y_c = np.zeros([self._n, 1])
        x_c = (algorithm_globals.random.random(self._n) - 0.5) * 1
        y_c = (algorithm_globals.random.random(self._n) - 0.5) * 1
        # for (cnt, s) in enumerate(self.tickers):
        # x_c[cnt, 1] = self.data[cnt][0]
        # y_c[cnt, 0] = self.data[cnt][-1]
        return x_c, y_c

    def per(self) -> np.ndarray:
        """
        Returns a vector containing the mean value of each asset.

        Returns:
            a per-asset mean vector.
        Raises:
            QiskitFinanceError: no data loaded
        """
        try:
            if not self._data:
                raise QiskitFinanceError(
                    "No data loaded, yet. Please run the method run() first to lo
                )
        except AttributeError as ex:
            raise QiskitFinanceError(
                "No data loaded, yet. Please run the method run() first to load t
            ) from ex
        _div_func = np.vectorize(BaseDataProvider._divide)
        #period_returns = _div_func(np.array(self._data)[:, 1:], np.array(self._d
        #self.period_return_mean = cast(np.ndarray, np.mean(period_returns, axis=

        period_total_return = _div_func(np.array(self._data)[:, -1], np.array(sel
        self.period_total_return = cast(np.ndarray, period_total_return)
        return np.array(self._data)
```

In [65]:

```
# Created a class object
object = BaseDataProvider.per(data)
import pandas as pd
df = pd.DataFrame(columns = ['S0', 'S1', 'S2', 'S3'])
df.S0 = object[0]
df.S1 = object[1]
df.S2 = object[2]
df.S3 = object[3]
df
```

Out[65]:

	S0	S1	S2	S3
0	59.275111	88.263504	41.967196	91.118385
1	59.497896	87.782812	41.718319	92.048618
2	57.832825	89.591339	41.085444	92.698362
3	57.668238	89.568783	41.132155	91.809670
4	58.795860	88.779449	41.699691	91.735292
...	...	...	...	...
10943	255.597289	99.027342	118.430643	96.861760
10944	256.304594	99.864607	117.961994	96.303711
10945	259.386138	101.674332	118.615580	94.617105
10946	260.367798	102.605393	118.551716	93.774948
10947	260.703867	103.238090	119.452349	93.539939

10948 rows × 4 columns

In [67]:
```
!pip install PyPortfolioOpt
```

```
Collecting PyPortfolioOpt
  Downloading PyPortfolioOpt-1.5.1-py3-none-any.whl (61 kB)
     |████████████████████████████████| 61 kB 30 kB/s s eta 0:00:01
Requirement already satisfied: cvxpy<2.0.0,>=1.1.10 in /opt/conda/lib/python3.8/site-packages (from PyPortfolioOpt) (1.1.18)
Requirement already satisfied: pandas>=0.19 in /opt/conda/lib/python3.8/site-packages (from PyPortfolioOpt) (1.4.0)
Requirement already satisfied: numpy<2.0,>=1.12 in /opt/conda/lib/python3.8/site-packages (from PyPortfolioOpt) (1.22.1)
Requirement already satisfied: scipy<2.0,>=1.3 in /opt/conda/lib/python3.8/site-packages (from PyPortfolioOpt) (1.7.3)
Requirement already satisfied: scs>=1.1.6 in /opt/conda/lib/python3.8/site-packages (from cvxpy<2.0.0,>=1.1.10->PyPortfolioOpt) (3.1.0)
Requirement already satisfied: osqp>=0.4.1 in /opt/conda/lib/python3.8/site-packages (from cvxpy<2.0.0,>=1.1.10->PyPortfolioOpt) (0.6.2.post5)
Requirement already satisfied: ecos>=2 in /opt/conda/lib/python3.8/site-packages (from cvxpy<2.0.0,>=1.1.10->PyPortfolioOpt) (2.0.10)
Requirement already satisfied: qdldl in /opt/conda/lib/python3.8/site-packages (from osqp>=0.4.1->cvxpy<2.0.0,>=1.1.10->PyPortfolioOpt) (0.1.5.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.8/site-packages (from pandas>=0.19->PyPortfolioOpt) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /opt/conda/lib/python3.8/site-packages (from pandas>=0.19->PyPortfolioOpt) (2.8.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.8/site-packages (from python-dateutil>=2.8.1->pandas>=0.19->PyPortfolioOpt) (1.16.0)
Installing collected packages: PyPortfolioOpt
Successfully installed PyPortfolioOpt-1.5.1
```

In [68]:
```
import pandas as pd
from pypfopt import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns

# Calculate expected returns and sample covariance
mu = expected_returns.mean_historical_return(df)
S = risk_models.sample_cov(df)
```

```
# Optimize for maximal Sharpe ratio
ef = EfficientFrontier(mu, S)
raw_weights = ef.max_sharpe()
cleaned_weights = ef.clean_weights()
ef.save_weights_to_file("weights.csv")  # saves to file
print(cleaned_weights)
ef.portfolio_performance(verbose=True)
```

```
OrderedDict([('S0', 0.94529), ('S1', 0.0), ('S2', 0.05471), ('S3', 0.0)])
Expected annual return: 3.4%
Annual volatility: 10.6%
Sharpe Ratio: 0.13
```

Out[68]:
```
(0.0341209016579502, 0.10562818294211969, 0.13368498126762185)
```

In [69]:
```
from pypfopt.expected_returns import mean_historical_return
from pypfopt.risk_models import CovarianceShrinkage


mu = mean_historical_return(df)
S = CovarianceShrinkage(df).ledoit_wolf()
from pypfopt.efficient_frontier import EfficientFrontier

ef = EfficientFrontier(mu, S)
weights = ef.max_sharpe()

cleaned_weights = ef.clean_weights()
print(dict(cleaned_weights))
ef.portfolio_performance(verbose=True)
```

```
{'S0': 0.92874, 'S1': 0.0, 'S2': 0.07126, 'S3': 0.0}
Expected annual return: 3.4%
Annual volatility: 12.4%
Sharpe Ratio: 0.11
```

Out[69]:
```
(0.03395013901806946, 0.12368532528498227, 0.11278734147262068)
```

In [44]:
```
#Let's calculate the expected return for our problem data

mu = data.get_period_return_mean_vector()   # Returns a vector containing the mea

print(mu)
```

```
[1.59702144e-04 4.76518943e-04 2.39123234e-04 9.85029012e-05]
```

Covariance Matrix Σ

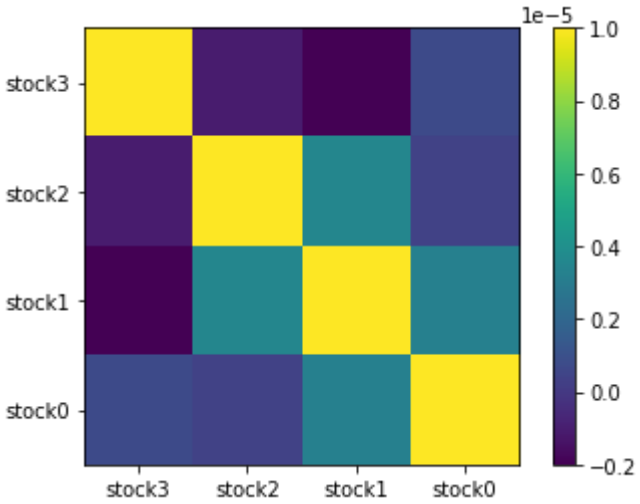
Covariance Σ is a statistical measure of how two asset's mean returns vary with respect to each other and helps us understand the amount of risk involved from an investment portfolio's perspective to make an informed decision about buying or selling stocks.

If you have 'n' stocks in your porfolio, the size of the covariance matrix will be n x n. Let us plot the covariance marix for our 4 stock portfolio which will be a 4 x 4 matrix.

In [21]:
```
# Let's plot our covariance matrix Σ (sigma)
sigma = data.get_period_return_covariance_matrix() #Returns the covariance matrix
print(sigma)
fig, ax = plt.subplots(1,1)
im = plt.imshow(sigma, extent=[-1,1,-1,1])
x_label_list = ['stock3', 'stock2', 'stock1', 'stock0']
y_label_list = ['stock3', 'stock2', 'stock1', 'stock0']
```

```
ax.set_xticks([-0.75,-0.25,0.25,0.75])
ax.set_yticks([0.75,0.25,-0.25,-0.75])
ax.set_xticklabels(x_label_list)
ax.set_yticklabels(y_label_list)
plt.colorbar()
plt.clim(-0.000002, 0.00001)
plt.show()
```

```
[[ 4.88319903e-05 -1.07868619e-06 -2.12961489e-06  7.06600109e-07]
 [-1.07868619e-06  9.97360142e-04  3.51594354e-06  3.68715793e-07]
 [-2.12961489e-06  3.51594354e-06  2.87365468e-04  3.20819120e-06]
 [ 7.06600109e-07  3.68715793e-07  3.20819120e-06  1.92316728e-04]]
```



The left-to-right diagnoal values (yellow boxes in the figure below) show the relation of a stock with 'itself'. And the off-diagonal values show the deviation of each stock's mean expected return with respect to each other. A simple way to look at a covariance matrix is:

- ***If two stocks increase and decrease simultaneously then the covariance value will be positive.***
- ***If one increases while the other decreases then the covariance will be negative.***

You may have heard the phrase "Don't Put All Your Eggs in One Basket." If you invest in things that always move in the same direction, there will be a risk of losing all your money at the same time. Covariance matrix is a nice measure to help investors diversify their assets to reduce such risk.

Now that we have all the values we need to build our portfolio for optimization, we will look into Qiskit's Finance application class that will help us contruct the quadratic program for our problem.

Qiskit Finance application class

In Qiskit, there is a dedicated `PortfolioOptimization` application to construct the quadratic program for portfolio optimizations.

PortfolioOptimization class creates a porfolio instance by taking the following five arguments then converts the instance into a quadratic program.

Arguments of the PortfolioOptimization class:

- ***expected_returns***
- ***covariances***
- ***risk_factor***
- ***budget***
- ***bounds***

Once our portfolio instance is converted into a quadratic program, then we can use quantum variational algorithms suchs as Variational Quantum Eigensolver (VQE) or the Quantum Approximate Optimization Algorithm (QAOA) to find the optimal solution to our problem.

We already obtained expected_return and covariances from Step 3 and have risk factor and budget pre-defined. So, let's build our portfolio using the `PortfolioOptimization` ***class.***

Challenge 1a: Create the portfolio instance using PortfolioOptimization class

*****Challenge 1a****
Complete the code to generate the portfolio instance using the [**PortfolioOptimization**] (https://qiskit.org/documentation/finance/stubs/qiskit_finance.applications.PortfolioO class. Make sure you use the **five arguments** and their values which were obtained in the previos steps and convert the instance into a quadratic program **qp**.

*****Note:** A binary list [1. 1. 0. 0.] indicates a portfolio consisting STOCK2 and STOCK3.***

In [26]:
```
##############################
# Provide your code here
portfolio = PortfolioOptimization(expected_returns = mu, covariances = sigma,
                                  risk_factor = q, budget = budget)
qp = portfolio.to_quadratic_program()

##############################
print(qp)
```

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

Minimize
 obj: - 3.398201220000 x_0 - 0.169657730000 x_1 - 1.846326660000 x_2
      - 0.026575910000 x_3 + [ 0.000048831990 x_0^2 - 0.000002157372 x_0*x_1
      - 0.000004259230 x_0*x_2 + 0.000001413200 x_0*x_3 + 0.000997360142 x_1^2
      + 0.000007031887 x_1*x_2 + 0.000000737432 x_1*x_3 + 0.000287365468 x_2^2
      + 0.000006416382 x_2*x_3 + 0.000192316728 x_3^2 ]/2
Subject To
 c0: x_0 + x_1 + x_2 + x_3 = 2

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1

Binaries
 x_0 x_1 x_2 x_3
End
```

If you were able to successfully generate the code, you should see a standard representation of the formulation of our qudratic program.

In [7]:
```
# Check your answer and submit using the following code
from qc_grader import grade_ex1a
grade_ex1a(qp)
```

```
Submitting your answer for 1a. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

Minimum Eigen Optimizer

Interestingly, our portfolio optimization problem can be solved as a ground state search of a Hamiltonian. You can think of a Hamiltonian as an energy function representing the total energy of a physical system we want to simulate such as a molecule or a magnet. The physical system can be further represented by a mathemetical model called an Ising model which gives us a framework to convert our binary variables into a so called spin up (+1) or spin down (-1) state.

When it comes to applyting the optimization algorithms, the algorithms usually require problems to satisfy certain criteria to be applicable. For example, variational algorithms such as VQE and QAOA can only be applied to Quadratic Unconstrained Binary Optimization (QUBO) problems, thus Qiskit provides converters to automatically map optimization problems to these different formats whenever possible.

Solving a QUBO is equivalent to finding a ground state of a Hamiltonian. And the Minimum Eigen Optimizer translates the Quadratic Program to a Hamiltonian, then calls a given Mimimum Eigensolver such as VQE or QAOA to compute the ground states and returns the optimization results for us.

This approach allows us to utilize computing ground states in the context of solving optimization problems as we will demonstrate in the next step in our challenge exercise.

Step 5. Solve with classical optimizer as a reference

Lets solve the problem. First classically...

We can now use the Operator we built above without regard to the specifics of how it was created. We set the algorithm for the NumPyMinimumEigensolver so we can have a classical reference. Backend is not required since this is computed classically not using quantum computation. The result is returned as a dictionary.

In [23]:
```
exact_mes = NumPyMinimumEigensolver()
exact_eigensolver = MinimumEigenOptimizer(exact_mes)
result = exact_eigensolver.solve(qp)

print(result)
```

```
optimal function value: -5.2443619108857975
optimal value: [1. 0. 1. 0.]
status: SUCCESS
```

The optimal value indicates your asset allocation.

Challenge1b: Solution using VQE

Variational Quantum Eigensolver (VQE) is a classical-quantum hybrid algorithm which outsources some of the processing workload to a classical computer to efficiently calculate the ground state energy (lowest energy) of a Hamiltonian). As we discussed earlier, we can reformulate the quadratic program as a ground state energy search to be solved by VQE where the ground state corresponds to the optimal solution we are looking for. In this challenge exercise, you will be asked to find the optimal solution using VQE.

> ***Challenge 1b***
> *Find the same solution by using Variational Quantum Eigensolver (VQE) to solve the problem. We will specify the optimizer and variational form to be used.*

> ***HINT:** If you are stuck, check out [**this qiskit tutorial**] (https://qiskit.org/documentation/finance/tutorials/01_portfolio_optimization.html) and adapt it to our problem:*

Below is some code to get you started.

In [35]:
```
optimizer = SLSQP(maxiter=1000)
algorithm_globals.random_seed = 1234
backend = Aer.get_backend('statevector_simulator')

ry = TwoLocal(num_assets, 'ry', 'cz', reps=3, entanglement='full')
```

```
quantum_instance = QuantumInstance(backend=backend, seed_simulator=seed, seed_tra

##############################
# Provide your code here

vqe = VQE(ry, optimizer=optimizer, quantum_instance=quantum_instance)

##############################

vqe_meo = MinimumEigenOptimizer(vqe)  #please do not change this code

result = vqe_meo.solve(qp)  #please do not change this code

print(result)    #please do not change this code
```

```
optimal function value: -0.00023285626449450194
optimal value: [1. 0. 1. 0.]
status: SUCCESS
```

In [36]:
```
# Check your answer and submit using the following code
from qc_grader import grade_ex1b
grade_ex1b(vqe, qp)
```

```
Submitting your answer for 1b. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

VQE should give you the same optimal results as the reference solution.

Challenge 1c: Portfolio optimization for B=3, n=4 stocks

In this exercise, solve the same problem where one can allocate double weights (can allocate twice the amount) for a single asset. (For example, if you allocate twice for STOCK3 one for STOCK2, then your portfolio can be represented as [2, 1, 0, 0]. If you allocate a single weight for STOCK0, STOCK1, STOCK2 then your portfolio will look like [0, 1, 1, 1])
Furthermore, change the constraint to B=3. With this new constraint, find the optimal portfolio that minimizes the tradeoff between risk and return.

> ***Challenge 1c****
> ***Complete the code to generate the portfolio instance using the PortfolioOptimization class.***
> ***Find the optimal portfolio for budget=3 where one can allocate double weights for a single asset.***
> ***Use QAOA to find your optimal solution and submit your answer.***

> ***HINT:** Remember that any one of STOCK0, STOCK1, STOCK2, STOCK3 can have double weights in our portfolio. How can we change our code to accommodate integer variables?*

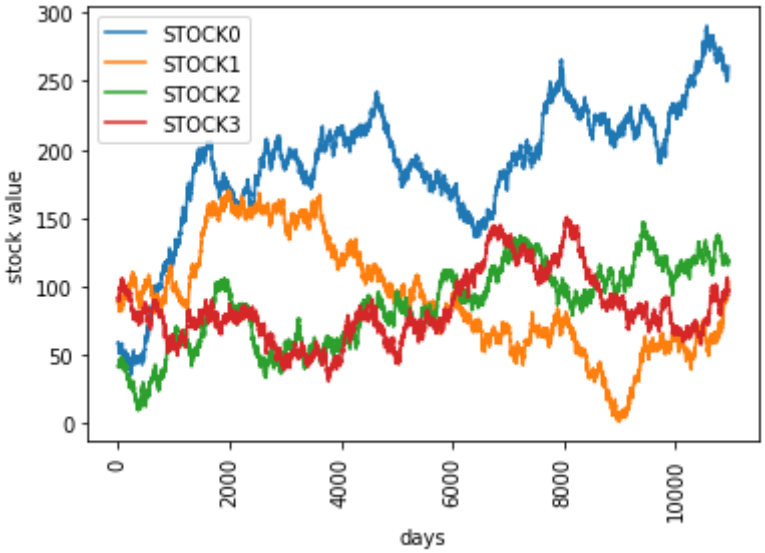
Step 1: Import necessary libraries

In [74]:
```
#Step 1: Let us begin by importing necessary libraries
import qiskit
from qiskit import Aer
from qiskit.algorithms import VQE, QAOA, NumPyMinimumEigensolver
from qiskit.algorithms.optimizers import *
from qiskit.circuit.library import TwoLocal
from qiskit.utils import QuantumInstance
from qiskit.utils import algorithm_globals
from qiskit_finance import QiskitFinanceError
from qiskit_finance.applications.optimization import *
from qiskit_finance.data_providers import *
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit_optimization.applications import OptimizationApplication
from qiskit_optimization.converters import QuadraticProgramToQubo
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import datetime
import warnings
from sympy.utilities.exceptions import SymPyDeprecationWarning
warnings.simplefilter("ignore",SymPyDeprecationWarning)
```

Step 2: Generate Time Series Data (Financial Data)

In [24]:
```
# Step 2. Generate time series data for four assets.
# Do not change start/end dates specified to generate problem data.
seed = 132
num_assets = 4
stocks = [("STOCK%s" % i) for i in range(num_assets)]
data = RandomDataProvider(tickers=stocks,
                 start=datetime.datetime(1955,11,5),
                 end=datetime.datetime(1985,10,26),
                 seed=seed)
data.run()
```

In [25]:
```
# Let's plot our finanical data (We are generating the same time series data as i
for (cnt, s) in enumerate(data._tickers):
    plt.plot(data._data[cnt], label=s)
plt.legend()
plt.xticks(rotation=90)
plt.xlabel('days')
plt.ylabel('stock value')
plt.show()
```



Step 3: Calculate expected return mu and covariance sigma

In [77]:
```
# Step 3. Calculate mu and sigma for this problem

mu2 = data.get_period_return_mean_vector()   #Returns a vector containing the mea
sigma2 = data.get_period_return_covariance_matrix() #Returns the covariance matri
print(mu2, sigma2)
```

```
[1.59702144e-04 4.76518943e-04 2.39123234e-04 9.85029012e-05] [[ 4.88319903e-05 -
1.07868619e-06 -2.12961489e-06  7.06600109e-07]
 [-1.07868619e-06  9.97360142e-04  3.51594354e-06  3.68715793e-07]
 [-2.12961489e-06  3.51594354e-06  2.87365468e-04  3.20819120e-06]
 [ 7.06600109e-07  3.68715793e-07  3.20819120e-06  1.92316728e-04]]
```

Step 4: Set parameters and constraints based on this challenge 1c.

In [78]:
```
# Step 4. Set parameters and constraints based on this challenge 1c

################################
# Provide your code here

q2 =  0.5    #Set risk factor to 0.5
budget2 = 3 #Set budget to 3

################################
```

Step 5: Complete code to generate the portfolio instance

In [85]:
```
# Step 5. Complete code to generate the portfolio instance


################################
# Provide your code here
bounds = [[0, 2] for i in range(num_assets)]
portfolio2 = PortfolioOptimization(expected_returns = mu2, covariances = sigma2,
                                   risk_factor = q2, budget = budget2, bounds = bo
```

```
qp2 = portfolio2.to_quadratic_program()

###############################
```

Step 6: Let's solve the problem using QAOA

Quantum Approximate Optimization Algorithm (QAOA) is another variational algorithm that has applications for solving combinatorial optimization problems on near-term quantum systems. This algorithm can also be used to calculate ground states of a Hamiltonian and can be easily implemented by using Qiskit's QAOA application. (You will get to learn about QAOA in detail in challenge 4. Let us first focus on the basic implementation of QAOA using Qiskit in this exercise.)

In [86]:
```
# Step 6. Now let's use QAOA to solve this problem.

optimizer = SLSQP(maxiter=1000)
algorithm_globals.random_seed = 1234
backend = Aer.get_backend('statevector_simulator')

###############################
# Provide your code here

quantum_instance = QuantumInstance(backend=backend, seed_simulator=seed, seed_tra
qaoa = QAOA(optimizer=optimizer, reps=3, quantum_instance=quantum_instance)

###############################

qaoa_meo = MinimumEigenOptimizer(qaoa) #please do not change this code

result2 = qaoa_meo.solve(qp2) #please do not change this code

print(result2) #please do not change this code
```

```
optimal function value: -0.00032144003832501437
optimal value: [2. 0. 1. 0.]
status: SUCCESS
```

Note: The QAOA execution may take up to a few minutes to complete.

Submit your answer

In [87]:
```
# Check your answer and submit using the following code
from qc_grader import grade_ex1c
grade_ex1c(qaoa, qp2)
```

```
Submitting your answer for 1c. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

Further Reading:

For those who have successfully solved the first introductory level challenge, congratulations!
I hope you were able to learn something about optimizing portfolios and how you can use Qiskit's Finance module to solve the example problem.
If you are interested in further reading, here are a few literature to explore:

1. *Quantum optimization using variational algorithms on near-term quantum devices. Moll et al. 2017*
2. *Improving Variational Quantum Optimization using CVaR. Barkoutsos et al. 2019.*

Good luck and have fun with the challenge!

Additional information

Created by: Yuri Kobayashi

Version: 1.0.0