Proyecto: "Dynamic Mode Decomposition: Aplicación en Trading Algorítmico"

Pedro Salomone y Santiago Vazquez

UNC Universidad Nacional de Córdoba

FAMAF Facultad de Matemática, Astronomía, Física y Computación

23 de Noviembre de 2022

Resumen

El presente estudio tiene como finalidad la aplicación del modelo de descomposición en modos dinámicos (DMD) como estrategia de trading algoritmico. Este método presenta la ventaja de poder caracterizar sistemas dinámicos complejos, como es el caso de los mercados financieros, de una manera libre de ecuaciones, ya que descompone el estado del sistema en términos de rango bajo cuyos coeficientes temporales en el tiempo son conocidos. Al extraer el método portfolios, entendido como una estructura temporal clave coherente, en su ventana muestral, éste provee una regresión que mejor se ajusta al sistema dinámico lineal. Éste modelo basado en datos permite entonces descubrir patrones del mercado para poder informar inversiónes de inversión de comprar, vender o mantener. Un punto crítico del método es el algoritmo asociado que busca optimizar la ventana muestral y de predicción, identificando *trading hot spots* que permitan obtener mejores resultados.

Keywords: Dynamic Mode Decomposition, sistemas dinámicos, trading financiero

1. Introducción

Nuestro trabajo tiene su justificación en la gran importancia que tiene el *trading algoritmico* en el mercado bursátil mundial (especialmente en los mercados de valores estadounidenses que son los de mayor amplitud y profundidad), ya que según estadísticas a Noviembre del 2022, el trading algorítmico el comercio algorítmico representa alrededor del 60-75 % del comercio total de acciones en USA.[1]. Este tipo de trading se basa en modelos matemáticos que buscan capitalizar sobre los patrones del mercado, de modo de informar o ejecutar en el caso que sean automatizados, decisiones de comprar, vender o mantener una acción o conjunto de acciones. Nuestro objetivo en este artículo es desarrollar un esquema de trading aplicando el modelo DMD (*Dynamic Mode Decomposition*) para capitalizar sobre los patrones de mercado que el modelo pueda identificar. La ventaja de este modelo es que puede integrar el poder de las series de tiempo con el Análisis de Componentes Principales (PCA), ya que, si bien el DMD es un algoritmo de reducción de dimensionalidad, a diferencia de algorítmos como el PCA los modos calculados tienen comportamientos temporales intrínsecos y no predeterminados. El supuesto que se realiza para poder utilizar este modelo es que el mercado bursátil es un sistema dinámico complejo que exhibe fenómenos multiescala no estacionarios. El modelo DMD

[1] https://therobusttrader.com/what-percentage-of-trading-is-algorithmic/

a diferencia de otros métodos aplicados a sistemas dinámicos no prescribe un modelo para explicar el sistema, sino que las dinámicas del mismo se reconstruyen directamente de los datos muestrales para una ventana temporal específica. El modelo descompone la información del portfolio de acciones en caracteristicas de rango bajo que se comportan con una dinámica temporal prescrita. Luego, el sistema dinámico lineal de ajuste por mínimos cuadrados permite predecir estados futuros del sistema a corto plazo, las cuales pueden usarse para formular estrategias de trading. Otra ventaja del modelo es que el mismo es adaptativo, ya que actualiza la descomposición a medida que el mercado cambia en el tiempo. También calcula la ventana temporal muestral y de predicción óptima para realizar la predicción, llamados "hot spots", es decir, la ventana temporal que mayor probabilidad de acierto en la predicción tiene. Nuestro trabajo aplicará el modelo para el índice $S\&P$ 500 discriminando el mismo por sector, siguiendo principalmente la metodología propuesta por Kutz. El presente paper se articula de la siguiente manera: En el punto 2 se describirá el trabajo realizado, explicando los puntos más importantes del modelo, los supuestos realizados para su aplicación y la metodología utilizada. Luego en la sección 3 mencionaremos las conclusiones a las que arribamos en nuestro estudio. Los códigos y archivos utilizados se adjuntarán en los Anexos A y B respectivamente.

2. Trabajo realizado

Como se mencionó en la introducción, el algoritmo DMD es un método libre de ecuaciones para aproximar las dinámicas no lineales de un sistema dinámico complejo. Formalmente:

$$\frac{dx}{dt} = N(x, t, \mu) \tag{1}$$

donde para nuestro trabajo, x es el portfolio de compañías seleccionadas. *N(.)* es un proceso dinámico desconocido generalmente no lineal y dependiente del tiempo, y μ como la variable que contiene los parámetros del sistema.

Como usualmente no es posible hallar la solución general para (1), se utilizan soluciones numéricas que buscan modelar los estados futuros del sistema. Como el DMD es un método libre de ecuaciones, asume que el lado derecho de la ecuación es desconocido, por ende, sólo se usan las observaciones tomadas y las condiciones iniciales para aproximar las dinámicas y predecir los estados futuros. El DMD realiza una aproximación lineal de la evolución

$$\frac{d\tilde{x}}{dt} = A\tilde{x} \tag{2}$$

con condición incial $\tilde{x}(0) = \tilde{x}_0$, cuya solución es:

$$\tilde{x}(t) = \sum_{i=1}^{K} b_k \psi_k exp(\omega_k t) \tag{3}$$

donde ψ_k y ω_k son los autovectores y autovalores de la matriz A, y los coeficientes b_k son las coordenadas de $x(0)$ en la base del autovector. La interpretación de esta solución en finanzas es importante ya que modos con una parte real positiva de ω_k son soluciones exponencialmente crecientes, es decir, pueden generar dinero en el caso de un portfolio, mientras que aquellos con parte real negativa son exponencialmente decreciente, perdiendo dinero en este caso. Las estrategias de trading se basaran en los valores de ω_k.

El objetivo final de este algoritmo es contruir la matriz A de forma tal que la solución real y aproximada estén próximas en el sentido de minimos cuadrados:

$$\|x(t) - \tilde{x}(t)\| \ll 1 \tag{4}$$

dentro de la ventana temporal en la cual A es construida.

El proceso de recolección de datos involucra dos parámetros:

$$N = \text{Numero de compañías del portfolio}$$
$$M = \text{Numero de instantáneas tomadas}$$

En nuestro trabajo consideramos tomar las instantáneas de los precios de las acciones del portfolio considerado en espacios regulares de tiempo (diarios) para simplificar el proceso.

Luego, los datos se almacenan en una matriz donde sus filas son las empresas y sus columnas los precios diarios tomados para la cantidad de días de la ventana temporal escogida. Por ejemplo, para el caso de un portfolio con acciones del sector financiero del $S\&P$ 500 tendríamos:

$$X = \begin{bmatrix} ... & 'BLK' & ... \\ ... & 'C' & ... \\ ... & 'GS' & ... \\ ... & . & ... \\ ... & . & ... \end{bmatrix} \tag{5}$$

donde de izquierda a derecha tendremos los precios diarios y de arriba a abajo las compañías del portfolio. Luego, con esta información construimos dos matrices *X1* y *X2*:

$$X_1 = \begin{bmatrix} | & | & ... & | \\ x_1 & x_2 & ... & x_{m-1} \\ | & | & ... & | \end{bmatrix} \tag{6}$$

$$X_2 = \begin{bmatrix} | & | & ... & | \\ x_2 & x_3 & ... & x_m \\ | & | & ... & | \end{bmatrix} \tag{7}$$

donde X_2 es la matriz desplazada en el tiempo.

El sistema puede seguir una dinámica no lineal, aunque lo que se busca aquí es hallar la aproximación lineal local óptima. Esta aproximación lineal local puede escribirse en términos de estas dos matrices como:

$$X_2 = AX_1 \tag{8}$$

donde la matriz A es el operador lineal de Koopman, cuya mejor aproximación se determina como:

$$A = X_2 {X_1}^{\dagger} \tag{9}$$

donde † es la pseudoinversa de Moore-Penrose. Por ende, para obtener esta aproximación de A necesitamos resolver la pseudoinversa de X_1. Una forma eficiente de hacerlo es usando la descomposición SVD de la matriz X_1:

$$X_1 = U\Sigma V^* \tag{10}$$

donde * denota la conjugada transpuesta, Σ es matriz diagonal donde sus valores representan la contribución de cada modo y U es ortogonal.

Debido a que la matriz A suele ser grande en este tipo de problemas, realizar su auto-descomposición resulta computacionalmente costoso, por ende podemos resolver esto para la matriz $\tilde{A}$ la cual es una aproximación de A. Ésta puede hallarse utilizando las ecuaciones (9) y (10) truncando la descomposición SVD de la matriz X_1 en r valores singulares, basado en la varianza capturada por los valores singulares, llegando a la siguiente expresión

$$\tilde{A} = U^* X_2 V \Sigma^{-1} \tag{11}$$

Consideramos entonces el problema del autovalor asociado a $\tilde{A}$:

$$\tilde{A}y_r = \mu_r y_r \quad r = 1, 2, ..., R \tag{12}$$

donde R es el rango de la aproximación que elegimos. Los autovalores μ_r contienen información sobre la dinámica temporal del sistema, y los vectores propios se pueden usar para reconstruir los modos DMD:

$$\psi_r = Uy_r \tag{13}$$

Con la aproximación de rango bajo de los autovalores y autovectores, la proyección de la solución futura puede ser construida para todo tiempo en el futuro, escribiendose como:

$$x_{DMD}(t) = \sum_{r=1}^{R} b_r(0)\psi_r(x)exp(\omega_r t) = \Psi diag(exp(\omega t))b \tag{14}$$

donde $\omega_r = ln(\mu_k)/\Delta t$, $b_k(0)$ es la amplitud inicial de cada modo, Ψ es la matriz cuyas columnas son los autovectores ψ_r, $diag(exp(\omega t)$ es una matriz diagonal cuyas entradas son la exponencial de los autovalores, y b es el vector de coeficientes b_r. Para calcular los valores de los coeficientes iniciales $b_k(0)$ fijamos $t = 0$ para la primera instantánea x_1, lo que nos resulta en $x_1 = \Psi b$. Pudiendo resolver la ecuación mediante la pseudo inversa:

$$b = \Psi^{\dagger} x_1 \tag{15}$$

Uniendo (14) y (15) nos quedaría la siguiente ecuación de proyección futura del estado del sistema para cualquier tiempo t:

$$x_{DMD}(t) = \Psi diag(exp(\omega t))\Psi^{\dagger} x(0) \tag{16}$$

Resumiendo, entrenaremos una matriz $\tilde{A}$ en un subconjunto de datos cuyos valores y vectores propios contienen la información necesaria para realizar predicciónes en un período de tiempo determinado.

El algoritmo puede resumirse como:

(1) Se toma una muestra de N acciones para M cantidad de días. Los datos se almacenan en una matriz X.

(2) Se divide la matrix X en dos submatrices $X1^{M-1}$ y $X2^{M}$.

(3) Se realiza la descomposición SVD para $X1^{M-1}$.

(4) Se deifne la matriz $\tilde{A}$ y se computan sus autovalores y autovectores.

(5) Se calcula el estado inicial del sistema mediante la pseudo inversa.

(6) Se computa la solución en cualquier tiempo t.

Para poder realizar el algoritmo de trading utilizando DMD, debemos considerar dos parámetros:

m = numero de días pasados de precios obtenidos
l = numero de días futuros predichos

es decir, utilizaremos m días pasados para predecir l días futuros. Utilizaremos información histórica para determinar las combinaciónes (m, l) que arrojen los mejores valores predictivos. Las combinaciones con mejores predicciones las llamaremos "hot spots". Entonces nuestro trabajo tiene dos puntos importantes: (i) determinar los hot spots en base a la información historica y (ii) implementar el algoritmo de trading con los hot spots obtenidos.

Nuestra metodología de aplicación será la siguiente:
Para la implementación del algoritmo utilizaremos el índice $S\&P$ 500 dividiendo el mismo en sectores, aplicaremos el DMD sobre una cartera teorica de acciones de cada sector en la cual todas las acciones tienen la misma ponderación, y evaluaremos, utilizando los hot spots identificados para cada sector, si el modelo predice que el sector subirá o bajará, comprando si sube o no comprando si baja, multiplicando el valor de la cartera por el rendimiento real. Luego, se compararán los rendimientos obtenidos para determinada ventana temporal para cada sector con el rendimiento del índice $S\&P$ 500 (por medio del ETF SPY), de modo de comparar si la estrategia de trading DMD tuvo una mejor performance que simplemente comprar el ETF y tenerlo durante ese tiempo. Nuestro trading será diario, de modo de simplificar el análisis. Tampoco consideraremos los costos de comisión por compra.

Por ejemplo, tomaremos el sector *Consumer Discretionary* del $S\&P$ 500 para demostrar el funcionamiento del modelo. El código utilizado para la implementación puede consultarse en el anexo A del presente informe.

Utilizando como ventana temporal los años 2018-2022, podemos ver los modos de cada sector para poder elegir el rango de truncamiento (desarrollar más y poner imágen de los autovalores). Luego, determinamos la mejor combinación de (mp, mf) realizando el análisis de hotspots, por ejemplo para el sector *Consumer Discretionary* tenemos la siguiente matriz de hotspots:

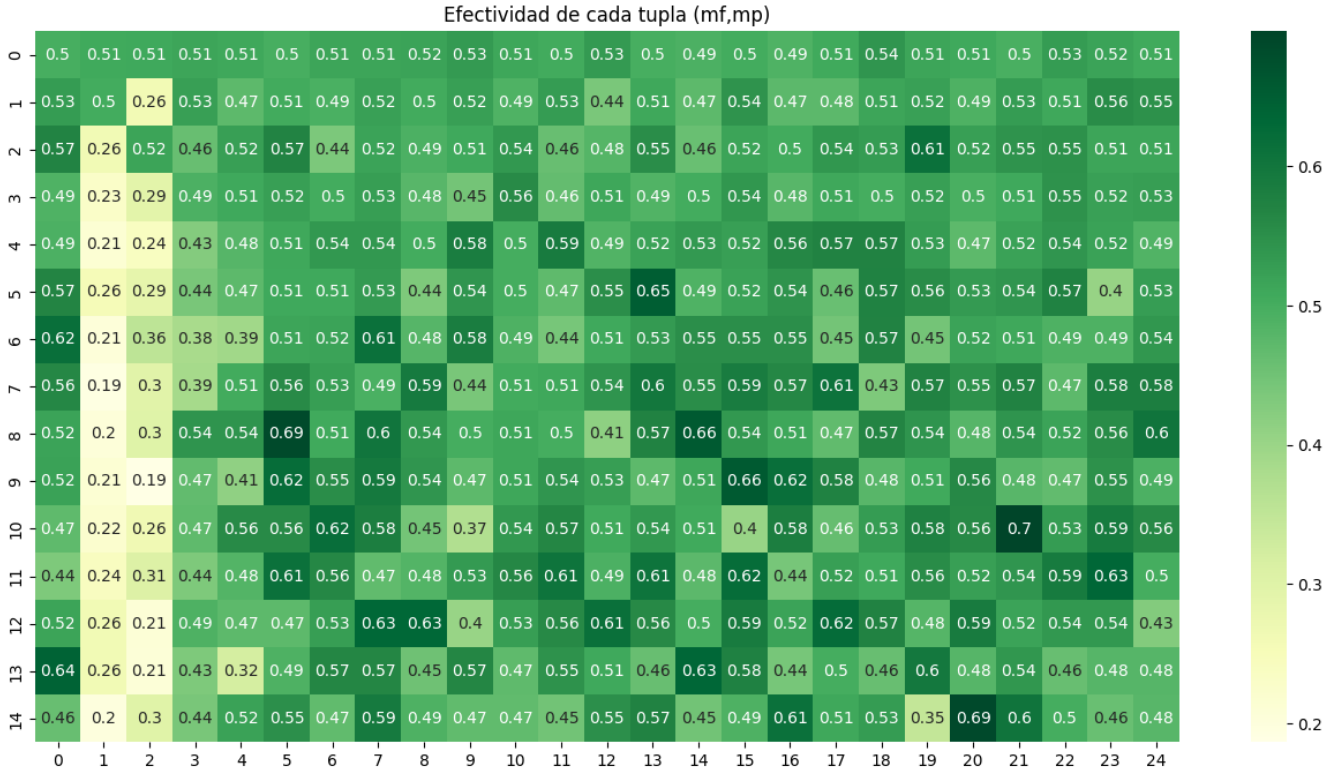

Figura 1: Hotspots

Podemos observar que los las combinaciones que mayor probabilidad de tener una predicción mas precisa son las que están pintadas de verde más oscuro, y, siguiendo a Kutz, es preferible tomar una combinación que tenga alta probabilidad de precisión y que además esté rodeada de combinaciones con alta probabilidad también.
Tomando una combinación de $(mp, mf) = (7, 10)$, es decir, 7 días del pasado para predecir 10 días futuros, aplicamos el algoritmo DMD para la ventana temporal elegida, calculamos los retornos acumulados de la cartera y finalmente comparamos los resultados con el ETF SPY (que replica el $S\&P$ 500). Podemos observar la performance del algoritmo para cada uno de los sectores elegidos para el análisis, donde para cada uno se eligió el hotspot correspondiente:

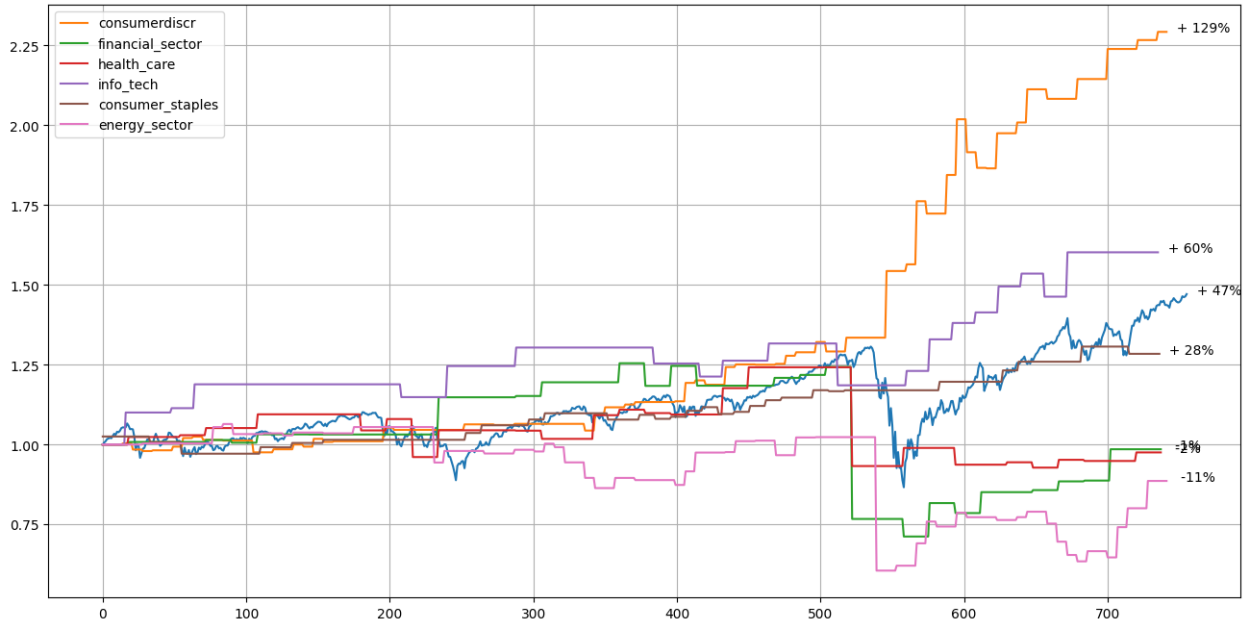

Figura 2: Rendimiento acumulado sectores/SPY. Podemos observar que, aún eligiendo los mejores hotspots para los distintos sectores, el algoritmo no produce mejores resultados que el SPY en todos los casos, sino que eso ocurre solo para el sector *Consumer Discretionary* e *Information Technology*. Esto se puede deber a que el algoritmo no detecta un hotspot claro que permita calcular una ventana temporal predictiva (mp, mf) consistente en el tiempo.

3. Conclusiones

Como conclusiones a nuestra aplicación del algoritmo DMD a las finanzas y especificamente el trading, podemos decir que el modelo provee una estrategia adaptativa y robusta que capitaliza sobre los patrones del mercado, y además es flexible debido a que no depende una ecuación sino que es una estrategia puramente basada en datos, por ende, los modos pueden reconstruirse de manera periódica, ajustando el algoritmo y no perder poder de predicción.
Se podrían modificar y cambiar ciertas caracteristicas de la implementación del algoritmo presentada en este informe, de modo que éste sea mas preciso y más realista. Una de las mejoras sería el cálculo de la tupla (mp, mf) cada cierto período de tiempo, de modo de que el algoritmo gane precisión recalculando la ventana óptima de días pasados para predecir días futuros en lugar de calcularlo al inicio y mantenerlo.
Otra posible mejora sería la predicción de suba y baja de cada uno de los activos en cartera, en lugar de utilizar el promedio de suba o baja de toda la cartera. De esta forma se podrían tomar decisiones de compra o venta para cada una de las acciones y ser así más precisos.
Por último, en el cálculo de la rentabilidad se podrían añadir los costos de comisiones cobrados

por cada transacción, esto sería más realista y se evitaría también sobreestimar la rentabilidad generada por el algoritmo.

A. Anexo con código

```
#Funcion que me da la Matriz de Datos de los precios de cierre en la ventana temporal indicada

#Importamos los tickers de un archivo excel
f = open('consumerdiscr.csv', 'r')
reader = csv.reader(f)
tickers = [i.split(',')[0] for i in f.readlines()]

#Descargamos la data
#Formato fecha debe estar en mm-dd-yy
df = yf.download(tickers, start = "2018-01-01", end="2021-01-01")['Adj Close']
dft = df.T

#Conviertimos el panda dataframe en un vector con los precios de cada empresa en fila
consumerdiscr = dft.values

#Con este codigo verificamos que no haya Nans en los precios
x, y = consumerdiscr.shape
for i in range(x):
    for j in range(y):
        if np.isnan(consumerdiscr[i,j]) == True:
            print(f'Nan en {(i,j)}')
```

```
# Algorimto DMD
# Convertimos la informacin de los precios en dos matrices,
#las cuales son iguales excepto que X2 esta un dt adelantada de la X1

def dmd(X1, X2, truncate=None):

    #Realizamos la descomposicion SVD de la matriz X1
    U2,Sig2,Vh2 = np.linalg.svd(X1, False) # SVD of input matrix

    #Definimos donde truncare las matrices para trabajar con matrices cuadradas
    r = len(Sig2) if truncate is None else truncate # Rank truncation

    U = U2[:,:r]                    #
    Sig = np.diag(Sig2)[:r,:r]      #Truncamos las matrices de la descomposicion SVD
    V = Vh2.conj().T[:,:r]          #

    Atil = U.conj().T @ X2 @ V @ np.linalg.inv(Sig) # Construyo A tilde

    #Autodescomposicion de A tilde
    mu ,W = np.linalg.eig(Atil)

    Phi = X2 @ V @ np.linalg.inv(Sig) @ W # Construyo los DMD modes

```

```
    return mu, Phi
```

```
# Funcin que predice el estado del sistema en un tiempo t

def x_t(t, Phi, mu, dt, A):
    return np.real(np.dot(np.dot(np.dot(Phi,np.power(np.diag(mu),t/dt)),
    np.linalg.pinv(Phi)), A[:,0]))
```

```
#Funcion que dados dos estados de tiempo (vectores),
#me devuelve la variacion promedio entre los elementos del
# primero y el ultimo

def variacion(a, b):
    variacion_elementos = []
    for i in range(len(a)):
        variacion_elementos.append((b[i] - a[i]) / a[i])
    return sum(variacion_elementos) / len(variacion_elementos)
```

```
def acertividad(variacion_predecida, precios_actuales, precios_pasados):
    #Definimos una variable entera, la cual sera 0 si el algoritmo hizo una mala prediccion
    #y sera 1 en caso contrario
    t = 0

    #Calculamos la variacion que tuvo el mercado
    variacion_real = variacion(precios_pasados, precios_actuales)

    #Comparamos si la tendencia que predijo el algoritmo es
    #la misma que ocurrio en realidad
    if np.sign(variacion_real) == np.sign(variacion_predecida):
            t = 1
    return t
```

```
#Funcion a la que le pasamos los dias en el pasado que
#queremos tomar en cuenta (mp) para predecir
# el estado de las acciones mf dias en el futuro desde el dia mp

def resultado(mf, mp, Precios):

    #Definimos dos variables, la primera, por cuanto se multiplica el valor
    #de nuestra cartera, la segunda
    # sera una lista de la evolucion de el valor de la misma
    estado = 1
    estados = []

    #El tiempo inicial sera el ultimo dato de precio que tiene el sistema, la columna
    #de la matriz mp
    t = mp

    #Definimos el numero de iteraciones que hara el algoritmo
    #iteraciones = int(np.floor(Precios.shape[1] / mf))

```

```
    for i in range(0, Precios.shape[1] - mp, mf):
        #Definimos las matrices con las que construiremos los DMD Modes
        X1 = Precios[ : , i : i + mp]
        X2 = Precios[ : , i + 1 : i + mp + 1]

        #Realizamos la descomposicion DMD de nuestras matrices
        mu, Phi = dmd(X1,X2)

        #El salto de tiempo sera la cantidad de dias que predeciremos en el futuro
        dt = mf

        #Actualizamos t al tiempo en el que queremos precedir el
        #precio de nuestra cartera
        t = t + mf

        #Este if evita un error de out of range cuando calculo la variacion real
        if i + mf + mp < Precios.shape[1]:

            #Calculamos los precios que tendra la cartera en el nuevo tiempo t
            xt = x_t(t, Phi, mu, dt, X1)

            #Calculamos la variacion predecida
            variacion_predecida = variacion(X2[ : , -1], xt)

            #Calculamos la varaicion real
            variacion_real = variacion(X2[ : , -1], Precios[ : , i + mf + mp])

            #Este if chequea que el t no este fuera de rango y si
            #el algoritmo predijo que el precio va a subir
            # a la vez que el precio real efectivamente subio
            if t < Precios.shape[1] and variacion_predecida > 0:

                #El algoritmo compra, por lo que nuestra cartera va a sufrir
                #la variacion que tenga el mercado
                estado = estado + estado * variacion_real

            #Con este for simplemente lleno la lista estados para luego poder
            #plotearlos a la par de otros activos
            for j in range(mf):
                estados.append(estado)

    return(estados)
```

```
# Funcion que devolvera un heatmap con la acerividad que tuvo el algoritmo

def HotSpots(MF,MP, Precios):
    comofue = np.zeros((MF, MP))

    for mp in range(1, MP + 1):
        for mf in range(1, MF + 1):

            #Definimos dos variables, las cuales, dividiendolas obtendremos
            #la acertividad del algoritmo
```

```
            casos_favorables = 0
            casos_totales = 0

            #El tiempo inicial sera el ultimo dato de precio que tiene el sistema,
            #la columna de la matriz mp
            t = mp

            #Definimos el numero de iteraciones que hara el algoritmo
            #iteraciones = int(np.floor(Precios.shape[1] / mf))

            for i in range(0, Precios.shape[1] - mp, mf):
                #Definimos las matrices con las que construiremos los DMD Modes
                X1 = Precios[ : , i : i + mp]
                X2 = Precios[ : , i + 1 : i + mp + 1]

                #Realizamos la descomposicion DMD de nuestras matrices
                mu, Phi = dmd(X1,X2)

                #El salto de tiempo sera la cantidad de dias que predeciremos
                #en el futuro
                dt = mf

                #Actualizamos t al tiempo en el que queremos
                #precedir el precio de nuestra cartera
                t = t + mf

                #Este if evita un error de out of range
                if i + mf + mp < Precios.shape[1]:

                    #Calculamos los precios que tendra la cartera en el nuevo tiempo
    t
                    xt = x_t(t, Phi, mu, dt, X1)

                    #Calculamos la variacion predecida
                    variacion_predecida = variacion(X2[ : , -1], xt)

                    #Calculamos la varaicion real
                    variacion_real = variacion(X2[ : , -1], Precios[ : , i + mf +
    mp])

                    #Este if chequea que el t no este fuera de rango y si el
    algoritmo predijo que el precio va a subir
                    # a la vez que el precio real efectivamente subio
                    if t < Precios.shape[1] and np.sign(variacion_predecida) ==
    np.sign(variacion_real):

                        casos_favorables += acertividad(variacion_predecida,
    Precios[:, t], X2[:,-1])
                        casos_totales += 1
                    else:
                        casos_totales += 1

            comofue[mf - 1, mp - 1] = casos_favorables / casos_totales
```

```
59
60      fig, ax = plt.subplots(figsize=(16,8))
61      ax = sns.heatmap(comofue, annot=True, cbar=True, cmap='YlGn')
62      plt.title(f"Efectividad de cada tupla (mf,mp)")
63      plt.show()
```

Para una revisión más detallada y actualizada del código pueden visitar el siguiente link: DMD code

Referencias

[1] Kutz, J. N., Brunton, S. L., Brunton, B. W., Proctor, J. L. *Dynamic mode decomposition: data-driven modeling of complex systems*. Society for Industrial and Applied Mathematics 2016.

[2] Taylor, R. *Dynamic Mode Decomposition in Python.*. Mathematics Python.