UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Tratamiento Digital de Señales Financieras

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Marchesi Selma, Pablo

Tutor/a: Albiol Colomer, Alberto

CURSO ACADÉMICO: 2023/2024

Resumen Ejecutivo

El presente trabajo tiene como objetivo desarrollar estrategias de trading algorítmico empleando métodos propios del tratamiento digital de señales, en concreto, el filtrado mediante filtros paso-bajo. La eficacia de estas estrategias se evaluará a través de un backtesting utilizando datos históricos de cotizaciones de acciones. Posteriormente, se aplicarán técnicas de optimización numérica para mejorar su rendimiento. Finalmente, se llevará a cabo un análisis estadístico para verificar su validez.

Abstract

The present work aims to develop algorithmic trading strategies using methods from digital signal processing, specifically low-pass filtering. The effectiveness of these strategies will be evaluated through backtesting using historical stock price data. Subsequently, numerical optimization techniques will be applied to improve their performance. Finally, a statistical analysis will be conducted to verify their validity.

Resum Executui

El present treball té com a objectiu desenvolupar estratègies de trading algorísmic emprant mètodes propis del tractament digital de senyals, en concret, el filtratge mitjançant filtres passa-baix. L'eficàcia d'aquestes estratègies s'avaluarà a través d'un backtesting utilitzant dades històriques de cotitzacions d'accions. Posteriorment, s'aplicaran tècniques d'optimització numèrica per millorar el seu rendiment. Finalment, es durà a terme una anàlisi estadística per verificar la seva validesa.

RESUMEN EJECUTIVO

La memoria del TFG/TFM del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la ingeniería de telecomunicación.

CONCEPT (ABET)	CONCEPTO (traducción)	¿Cumple? (S/N)	¿Dónde? (páginas)
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	4
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	4
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	19
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	19-29
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	30-38
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	39
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	39

Índice

Índice de figuras

Índice de cuadros

1 Introducción

El Tratamiento Digital de Señales (TDS) es una rama de las telecomunicaciones que se encarga del análisis y el procesamiento de señales digitales como pueden ser audio, imagen o vídeo. El TDS permite extraer o modificar información relevante de las señales, por ejemplo, mediante el uso de filtros, que permiten extraer componentes específicos de las señales.

Podemos ver aplicaciones del TDS en diversos campos como la biomedicina, la ingeniería electrónica, las telecomunicaciones, en el control de sistemas, en multimedia... En nuestro caso, nos centraremos en su aplicación a las finanzas y, en concreto, en el trading algorítmico, que se puede definir como el desarrollo y ejecución de sistemas de compra y venta en los mercados financieros, mediante el uso de algorítmos informáticos.

La idea de aplicar el TDS en este campo en concreto surge debido a las similitudes en el tipo de señal respecto a las señales con las que se puede trabajar en telecomunicaciones. Las señales financieras, que en nuestro caso serán acciones de bolsa, son digitales por naturaleza, ya que los datos de cotizaciones están estructurados como datos muestreados en intervalos de tiempo uniformes (diarios, semanales, mensuales...). Además, fluctúan con el tiempo y presentan un carácter no determinista, es decir, son procesos estocásticos como podría ser una señal de voz.

Por tanto, las técnicas que se emplean en el TDS también pueden ser aplicadas a los mercados financieros. En este trabajo desarrollaremos varios indicadores técnicos, es decir, herramientas matemáticas para evaluar y predecir el comportamiento futuro de los precios de los activos financieros, que construiremos ayudándonos del TDS. En concreto, haremos uso de filtros paso-bajo como pueden ser el filtro media móvil o el butterworth, para generar señales de compra y venta de acciones.

Una vez desarrollemos la estrategia de trading, realizaremos un backtesting con datos históricos de cotizaciones de acciones para ver cómo se ha comportado nuestro sistema en el pasado. Además, llevaremos a cabo una optimización numérica para estimar los parámetros óptimos de los indicadores técnicos. Posteriormente validaremos dicho sistema mediante métodos estadísticos como la simulación de casos extremos o los análisis de dependencia y sensibilidad de la estrategia de trading. Todas estas pruebas nos ayudarán a decir si la estrategia es válida para implementarse en tiempo real.

Todos las gráficas de este trabajo son de elaboración propia, creadas con la librería de graficación *plotly*. Respecto a los datos de las cotizaciones de acciones, se han obtenido mediante la librería de Python *yfinance*.

2 Conceptos Básicos del Tratamiento Digital de Señales

En esta sección revisaremos los conceptos del TDS que usaremos a lo largo del trabajo. Para una visión más en profundidad de estos temas, consultar [9].

2.1. Filtro digital

Un filtro digital es un sistema que procesa señales digitales para modificar ciertas características de una señal de entrada $x[n]$ y obtener una señal de salida $y[n]$. Existen diversos tipos de filtros digitales: paso-bajo, paso-alto, paso-banda y elimina-banda. Cada uno de estos filtros está diseñado para permitir o atenuar ciertas frecuencias de la señal de entrada $x[n]$.

Figura 1: Relación entre la señal de entrada $x[n]$ y la señal de salida $y[n]$ a través de un filtro digital.

Alternativamente, podemos definir un filtro digital como la relación entre la señal de entrada, $x[n]$, y la de salida, $y[n]$, mediante una ecuación en diferencias con la siguiente forma [10]:

$$y[n] = \sum_{i=0}^{M} b_i x[n-i] - \sum_{j=1}^{N} a_j y[n-j] \tag{1}$$

donde b_i y a_j representan los coeficientes del filtro. La elección de estos coeficientes será clave a la hora de diseñar los filtros y es el tema central de este trabajo. La ecuación (1) implica que nuestro sistema debe de ser causal, lineal e invariable en el tiempo. Las aplicaciones que desarrollaremos haciendo uso de estos filtros cumplen dichas condiciones.

2.2. Respuesta al Impulso

Otra forma de caracterizar un filtro digital es mediante su respuesta al impulso $h[n]$, que es la respuesta del sistema al introducir un impulso unitario $\delta[n]$.

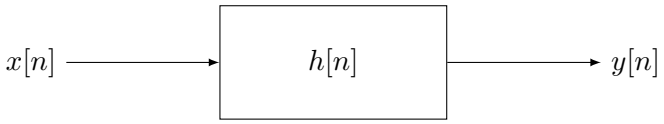

Figura 2: Relación entre la señal de entrada $x[n]$ y la señal de salida $y[n]$ a través de la respuesta al impulso $h[n]$.

Para obtener la señal de salida $y[n]$ a partir de la señal de entrada $x[n]$ y la respuesta al impulso $h[n]$ haremos uso de la famosa operación de la convolución, que en este contexto será discreta (o digital).

Definición 1 *La Convolución Discreta* $(x * h)[n]$ *de dos secuencias discretas* $x[n]$ *y* $h[n]$ *se define como:*

$$y[n] = (x * h)[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k] \tag{2}$$

donde $x[n]$ *y* $h[n]$ *son secuencias discretas y* $(x * h)[n]$ *representa el valor de la convolución discreta en el índice* n*.*

Por tanto, si conocemos la señal de entrada y la respuesta al impulso de nuestro filtro digital, podremos obtener la señal de salida a través de la convolución.

2.3. Respuesta en Frecuencia

Es una práctica muy común en el TDS analizar las señales en el dominio de la frecuencia. Gran parte de la utilidad del filtrado digital reside en que nos permite modificar ciertos componentes frecuenciales de la señal de entrada, luego es necesario examinar las señales en el dominio frecuencial además de en el dominio temporal. Para esta tarea, haremos uso de la Transformada de Fourier para señales discretas (la DTFT):

Definición 2 *La Transformada de Fourier en Tiempo Discreto (DTFT) de una secuencia* $x[n]$ *se define como:*

$$\mathcal{F}\{x[n]\} = X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n} \tag{3}$$

donde ω *es la frecuencia angular (en radianes por muestra) y* $X(e^{j\omega})$ *es la función de transformada de Fourier en tiempo discreto de* $x[n]$*.*

Una propiedad fundamental de las señales en tiempo discreto es que la convolución en el dominio del tiempo se convierte en multiplicación en el dominio de la frecuencia, expresada matemáticamente de esta forma:

$$\mathcal{F}\{(x * h)[n]\} = X(e^{j\omega}) \cdot H(e^{j\omega}) \tag{4}$$

Esta propiedad nos será muy útil a la hora de realizar el filtrado, ya que la forma de proceder será transformar las señales al dominio de la frecuencia, multiplicarlas, y transformarlas inversamente al dominio del tiempo. Esto es computacionalmente más eficiente que la convolución gracias a la FFT o Transformada Rápida de Fourier que consiste en un algoritmo optimizado para transformar señales discretas en tiempo discreto. Para más información sobre la FFT, consultar [5].

La función $H(e^{j\omega})$ en (4) es la transformada de la respuesta impulsiva del filtro, $h[n]$, y recibe el nombre de respuesta en frecuencia y nos servirá para caracterizar las modificaciones que realiza el filtro en los componentes frecuenciales de la señal.

2.4. Función de Transferencia

El número de coeficientes b_i y a_j de un filtro digital es proporcional a la complejidad de dicho filtro. Frecuentemente nos encontraremos con filtros que poseen numerosos coeficientes y lidiar con las ecuaciones en diferencias como en (1) resulta

tedioso en estos casos. Una forma alternativa de expresar los coeficientes del filtro es mediante la función de transferencia del filtro. Para definir este concepto, introduciremos primero la Transformada Z:

Definición 3 *La Transformada Z de una secuencia discreta $x[n]$ se define como:*

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]\, z^{-n} \tag{5}$$

donde z es una variable compleja y $X(z)$ es la función de Transformada Z de $x[n]$.

La Transformada Z es una generalización de la Transformada de Fourier en Tiempo Discreto y posee algunas de sus propiedades como la de que la convolución en el dominio del tiempo es una multiplicación en el dominio transformado (o dominio z):

$$\mathcal{Z}\{(x * h)[n]\} = X(z) \cdot H(z) \tag{6}$$

Equipados con la Transformada Z, podemos abordar el problema de las tediosas ecuaciones en diferencias para filtros complejos. Si transformamos la respuesta al impulso $h[n]$ a su dominio Z, obtenemos la llamada Función de Transferencia, $H(z)$:

Definición 4 *La función de transferencia $H(z)$ de un filtro digital se expresa como un cociente de polinomios $B(z)$ y $A(z)$ ponderado por los coeficientes del filtro b_i y a_j:*

$$H(z) = \frac{\sum_{i=0}^{M} b_i z^{-i}}{\sum_{j=0}^{N} a_j z^{-j}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}} = \frac{B(z)}{A(z)} \tag{7}$$

De esta forma simplificamos el diseño y el análisis del filtro digital al estudio de los coeficientes a_j y b_i en los polinomios $B(z)$ y $A(z)$. La función de transferencia es clave para el diseño del filtro, específicamente, la elección de los coeficientes, aunque de cara a la implementación en tiempo, seguiremos usando las ecuaciones en diferencias (1).

2.5. Frecuencia de Corte

A continuación, definiremos un concepto clave que será la base del diseño de los filtros digitales que llevemos a cabo y, por tanto, de los indicadores técnicos que usemos en los sistemas de trading algorítmico. Se trata de la frecuencia de corte.

Definición 5 *La frecuencia de corte, ω_c de un filtro es la frecuencia a la cual la respuesta del filtro se encuentra en un valor de atenuación igual a - 3dB respecto a la ganancia máxima o mínima del filtro.*

De cara a la implementación práctica, lo que haremos será emplear funciones de Python que nos permitan obtener los coeficientes de un filtro específico a partir de su frecuencia de corte ω_c. Por tanto, éste será el parámetro clave de nuestros sistemas de trading, y nuestra tarea será encontrar el ω_c óptimo que máximice la rentabilidad de la estrategia. Además, lo usaremos como parámetro común para armonizar los diferentes filtros y por tanto, las estrategias que desarrollemos. Hablaremos más sobre este tema en la sección 4.

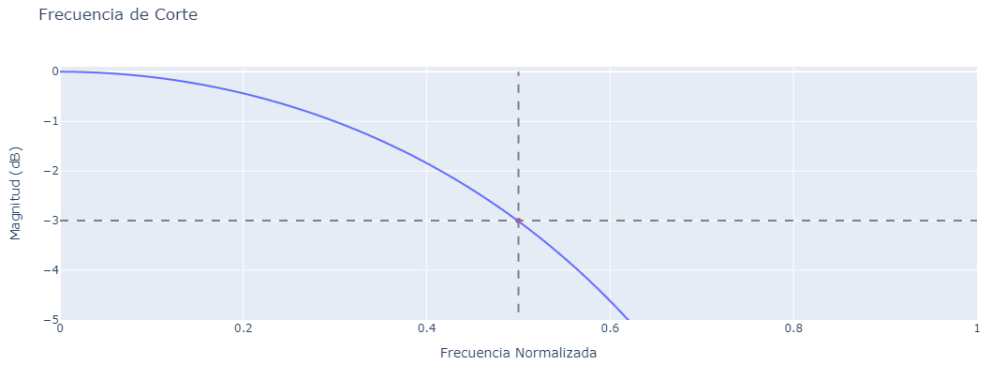

Figura 3: Filtro paso-bajo con $\omega_c = 0.5$

3 Filtros digitales

En esta sección nos centraremos en el diseño de los filtros digitales que usaremos posteriormente para nuestras estrategias de trading. Analizaremos su respuesta en frecuencia, que es la que más información nos proporcionará sobre su comportamiento, así como su implementación en tiempo y su función de transferencia. Todos los filtros con los que trabajaremos serán paso-bajo, ya que nuestro objetivo será suavizar la señal financiera (o de bolsa).

3.1. SMA: Simple Moving Average

Nuestro primer filtro es la media móvil simple o SMA (simple moving average). Se trata de un promediado temporal de las últimas N muestras de la señal. La respuesta al impulso de este filtro es la siguiente:

$$h[n] = \frac{1}{N} \sum_{k=0}^{N-1} \delta[n-k] \tag{8}$$

En la figura que se muestra a continuación podemos apreciar el filtrado que realiza la SMA sobre una señal de bolsa:

Figura 4: Suavizado con una SMA de 20 periodos

Cuanto mayor sea la ventana de promediado, o como se suele decir, el periodo de la media móvil, N, mayor será el suavizado de la señal (aunque también tendrá más *lag*). Es un filtro muy básico, simple de calcular, y altamente utilizado en el mundo del trading.

Otra forma de entender el filtrado que realiza la media móvil es mediante su respuesta en frecuencia, que viene dada por la siguiente expresión [7]:

$$|H(e^{j\omega})| = \left| \frac{1}{N} \frac{\sin\left(\frac{\omega N}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} \right| \tag{9}$$

Hemos tomado exclusivamente el modulo de $H(e^{j\omega})$ ya que en nuestro caso, y durante todo este trabajo, la información de la fase del filtro no nos será relevante. Representaremos la respuesta en frecuencia en decibelios de la siguiente forma:

$$A(\omega)(dB) = 10 \log_{10} |H(e^{j\omega})|^2 \tag{10}$$

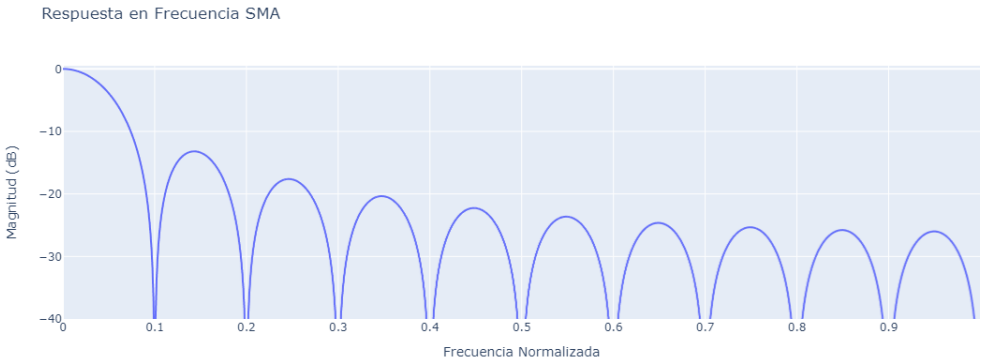

Figura 5: Respuesta en frecuencia de una SMA de 20 periodos

Respecto al diseño del filtro, la función de transferencia del filtro media móvil tiene la siguiente forma:

$$H(z) = \frac{1}{N} \sum_{n=0}^{N-1} z^{-n} \tag{11}$$

donde $A(z) = 1$ y $B(z)$ es un polinomio de orden N, cuyos coeficientes b_i son todos iguales y valen $\frac{1}{N}$. Como hemos comentado anteriormente, usaremos la frecuencia de corte ω_c para armonizar todos los filtros que diseñemos, es decir, compararemos los diferentes filtros digitales a partir de una misma frecuencia de corte. Por tanto, necesitamos diseñar el filtro SMA a partir de ω_c en lugar de con el periodo N. Si hacemos $|H(e^{j\omega})|^2 = \frac{1}{2}$ (el equivalente a -3dB) y haciendo algunas manipulaciones algebraicas [2], llegamos a:

$$\sin\left(\frac{N\omega_c}{2}\right) - \frac{N}{\sqrt{2}} \sin\left(\frac{\omega_c}{2}\right) = 0 \tag{12}$$

que podemos resolver mediante métodos numéricos. Para esta tarea, hemos hecho uso del método de Newton para obtener los ceros de la función (12) y despejar N. La implementación en Python es la siguiente:

Algoritmo 1 *Método de Newton para obtener el periodo N de la SMA a partir de la frecuencia de corte ω_c:*

```
# Metodo de Newton para obtener el perido de la SMA a
# partir de la frecuencia de corte

from math import pi, cos, sin, sqrt

def N_sma(wc):

    wc = wc*pi
    func = lambda N: sin(wc*N/2) - (N/sqrt(2))*(sin(wc/2))
    deriv = lambda N: (wc/2)*cos(wc*N/2)-(1/sqrt(2))*sin(wc/2)
    N_0 = pi/wc

    return int(np.round(newton(func, N_0, deriv)))
```

Para finalizar con el filtro SMA, lo implementaremos en el dominio del tiempo, y en función de la frecuencia de corte, de la siguiente forma:

Algoritmo 2 *Implementación temporal de un filtrado con una SMA a partir de la frecuencia de corte:*

```
# Filtro media movil en funcion de la frecuencia de corte

def sma(x, wc):
    N = N_sma(wc)
    y = pd.Series(x).rolling(N).mean()
    return y
```

3.2. EMA: Exponential Moving Average

La SMA es un caso específico de un filtro FIR o de respuesta al impulso finita. Otro tipo de media móvil es la media móvil exponencial o EMA (exponential moving average). Este filtro es de respuesta al impulso infinita, IIR, y pondera los datos más recientes, decayendo exponencialmente con el tiempo. Su respuesta al impulso tiene la siguiente expresión:

$$h[n] = \alpha(1-\alpha)^n \tag{13}$$

donde α es el factor de suavizamiento, un número entre 0 y 1. La EMA reacciona más rápido a los cambios en el precio que la SMA, aunque presenta un menor suavizado de la señal. Cuanto más cercano a 1 sea α, con mayor rapidez reaccionará el filtro a los cambios en el precio. Otra forma alternativa de expresar el filtro EMA es mediante la ecuación en diferencias que se muestra a continuación [7]:

$$y[n] = \alpha x[n] + (1-\alpha)y[n-1] \tag{14}$$

Se trata de un filtro recursivo ya que a parte de la señal, se emplean los cálculos previos del filtro para calcular el siguiente valor.

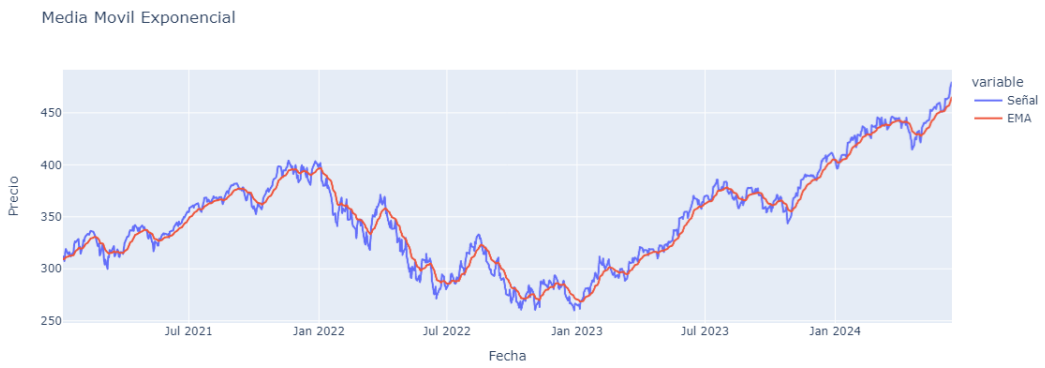

Figura 6: Filtrado con una EMA donde $\alpha = 0.13$

Por otro lado, la respuesta en frecuencia de la EMA es la siguiente [2]:

$$|H(e^{j\omega})| = \frac{\alpha}{\sqrt{1 - 2(1-\alpha)\cos(\omega) + (1-\alpha)^2}} \tag{15}$$

A diferencia de la SMA, que introducía ceros en ciertas frecuencias de la señal (9), la EMA tiene una respuesta que decae exponencialmente sin llegar a introducir ningún cero. Esto cambia el comportamiento del filtro de forma notable, haciéndolo más ruidoso o con una mayor sensibilidad ante los cambios de la señal.

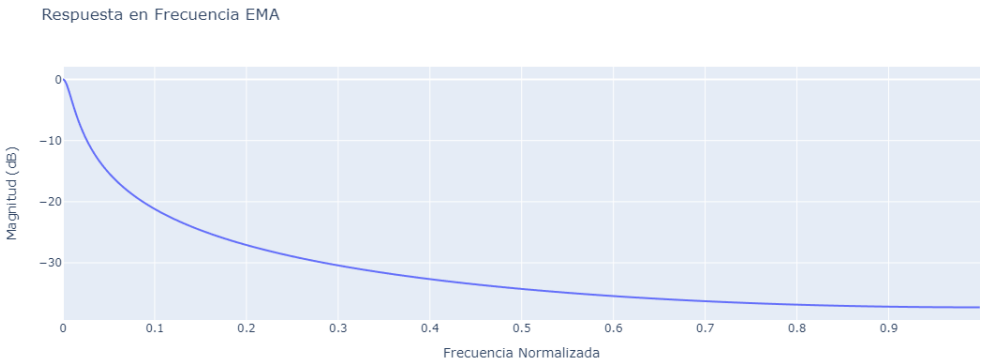

Figura 7: Respuesta en frecuencia de una EMA con $\alpha = 0.13$

Respecto al diseño del filtro, podemos definir su función de transferencia como:

$$H(z) = \frac{\alpha z}{z + (\alpha - 1)} \tag{16}$$

Como se puede observar, es un filtro de tres coeficientes con $b_0 = \alpha$, $a_0 = 1$ y $a_1 = (\alpha - 1)$. De nuevo, surge la problemática de implementar el filtro en función de la frecuencia de corte ω_c, en lugar de con el factor de suavizado α. Haciendo $|H(e^{j\omega})|^2 = \frac{1}{2}$ en (15) obtenemos:

$$\alpha^2 + 2 \cdot (1 - \cos(\omega_c))\alpha + 2 \cdot (\cos(\omega_c) - 1) = 0 \tag{17}$$

que volveremos a resolver mediante el método de Newton para despejar α, de la siguiente forma:

Algoritmo 3 *Método de Newton para obtener α a partir de ω_c:*

```
# Metodo de Newton para obtener el parametro alpha de la
# EMA a partir de la frecuencia de corte

def alpha_ema(wc):
    wc = wc*pi
    B = 2*(1-cos(wc)); C = 2*(cos(wc)-1)
    func = lambda alpha: alpha**2 + B*alpha + C
    deriv = lambda alpha: 2*alpha + B
    alpha_0 = 0.5
    return newton(func, alpha_0, deriv)
```

Para finalizar, implementaremos el filtro EMA según la ecuación en diferencias (14), a partir de la frecuencia de corte.

Algoritmo 4 *Implementación temporal del filtrado con una EMA a partir de la frecuencia de corte:*

```
# Filtro media movil exponencial en funcion de la
# frecuencia de corte

def ema(x, wc):
    alpha = alpha_ema(wc)
    y = []

    for n in range(len(x)):
        if n == 0:
            y.append(x[0])
        else:
            y_n = alpha*x[n] + (1-alpha)*y[n-1]
            y.append(y_n)
    return y
```

3.3. DMA: Double Moving Average

El siguiente filtro que analizaremos es la media móvil doble o DMA (Double Moving Average). Consiste en hacer pasar la salida de un filtro media móvil simple por otro filtro SMA del mismo periodo, para obtener aún más suavizado (aunque mayor *lag*). Podemos entender la respuesta al impulso de este filtro como una convolución de dos filtros SMA:

$$h_{dma}[n] = (h_{sma} * h_{sma})[n] \tag{18}$$

El filtro equivalente (FIR) será una función triangular con $2(N-1)$ muestras no nulas, es decir, los coeficientes b_i, ponderando con un mayor peso los valores centrales de la ventana de filtrado (a diferencia de la SMA que ponderaba a todos los valores por igual).

Figura 8: Filtrado con una DMA de periodo 14

Aplicando la propiedad (4) de la Transformada de Fourier, podemos obtener la respuesta en frecuencia de forma sencilla:

$$\mathcal{F}\{(h_{sma} * h_{sma})[n]\} = H_{sma}(e^{j\omega}) \cdot H_{sma}(e^{j\omega}) = H_{sma}(e^{j\omega})^2$$

Por tanto, la respuesta en frecuencia de la DMA será:

$$|H(e^{j\omega})| = \left| \frac{1}{N} \frac{\sin\left(\frac{\omega N}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} \right|^2 \tag{19}$$

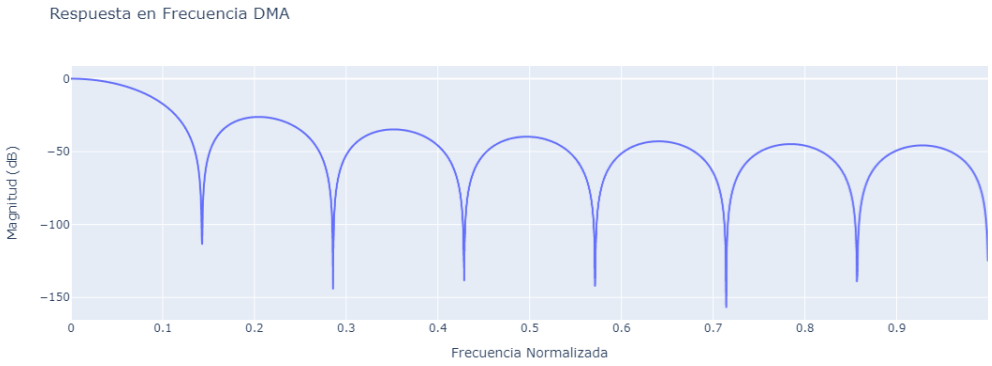

Figura 9: Respuesta en frecuencia de un filtro DMA

Respecto a la función de transferencia, se vuelve a cumplir la propiedad (6), dando lugar a:

$$\mathcal{Z}\{(h_{sma} * h_{sma})[n]\} = H_{sma}(z) \cdot H_{sma}(z) = H_{sma}(z)^2$$

y desarrollando la expresión (11) se obtiene:

$$H_{sma}(z) = \frac{1}{N} \sum_{k=0}^{N-1} z^{-k} = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}}$$

Por tanto la expresión de la función de transferencia del filtro DMA es:

$$H(z) = \left(\frac{1}{N}\frac{1-z^{-N}}{1-z^{-1}}\right)^2 = \frac{1}{N^2}\left(\frac{1-z^{-N}}{1-z^{-1}}\right)^2 \tag{20}$$

Y como hemos hecho anteriormente, obtendremos el periodo, N, a partir de la frecuencia de corte, ω_c, mediante el método de Newton. Si procedemos como en (12), llegamos a:

$$\sin\left(\frac{N\omega_c}{2}\right) - \frac{N}{\sqrt[4]{2}}\sin\left(\frac{\omega_c}{2}\right) = 0 \tag{21}$$

que podemos solucionar con el siguiente algoritmo en Python:

Algoritmo 5 *Método de Newton para obtener el periodo de la DMA a partir de la frecuencia de corte:*

```
# Metodo de Newton para obtener el perido de la DMA a
# partir de la frecuencia de corte

def N_dma(wc):
    wc = wc*pi
    func = lambda N: sin(wc*N/2) - (N/(2**(1/4)))*(sin(wc/2))
    deriv = lambda N: (wc/2)*cos(wc*N/2) - (1/(2**(1/4)))*sin(
       wc/2)
    N_0 = pi/wc
    return int(np.round(newton(func, N_0, deriv)))
```

Además, implementaremos el filtro en el dominio temporal de la siguiente forma:

Algoritmo 6 *Filtrado temporal de una señal a usando una DMA:*

```
# Filtrado temporal con DMA

    def dma(x, wc):
    N = N_dma(wc)
    h = (1/N)*np.ones(N,)
    B = np.convolve(h, h, mode='full')

    y = []
    for n in range(len(x)):
        if n < len(B):
            y.append(np.NAN)
        else:
            y_n = np.dot(B,x[n-len(B):n])
            y.append(y_n)

    return y
```

3.4. Butterworth

A continuación analizaremos la versión digital de un famoso filtro analógico, el filtro Butterworth. Una característica muy relevante de dicho filtro es que la respuesta en frecuencia es plana a frecuencias muy bajas, y luego decae de forma suave a partir de la frecuencia de corte [7]. A la hora de diseñar el filtro, podemos escoger el orden de este, que es proporcional al número de coeficientes b_i y a_j. En nuestro caso y para mayor simplicidad, usaremos un filtro Butterworth de orden 1. Dicho filtro se puede expresar mediante la siguiente ecuación en diferencias:

$$y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1] \tag{22}$$

Para este caso tenemos que $b_1 = b_0$. La siguiente imagen muestra cómo filtra el Butterworth una señal de bolsa:

Figura 10: Filtro Butterworth con $\omega_c = 0.044$

Por otro lado, la magnitud o el modulo de la respuesta en frecuencia de dicho filtro es la siguiente:

$$|H(e^{j\omega})| = \left| \frac{\omega_c(1 + e^{-j\omega})}{2 + \omega_c + (2 - \omega_c)e^{-j\omega}} \right| \tag{23}$$

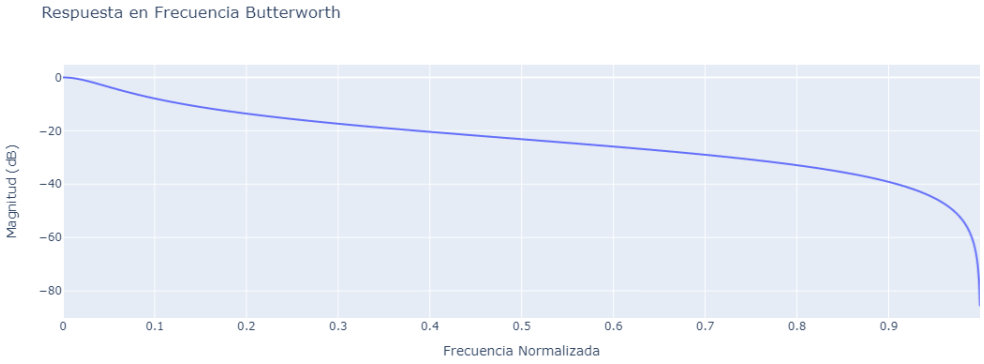

Figura 11: Respuesta en frecuencia de un filtro Butterworth con $\omega_c = 0.044$

Partiendo de la expresión (23) podemos deducir su función de transferencia, haciendo $z^{-1} = e^{-j\omega}$, por tanto, llegamos a:

$$H(z) = \frac{\omega_c(1 + z^{-1})}{2 + \omega_c + (2 - \omega_c)z^{-1}} \tag{24}$$

De cara a la implementación en tiempo, usaremos la ecuación (22), quedando el código en Python de esta forma:

Algoritmo 7 *Filtrado con un Butterworth de orden 1 a partir de la frecuencia de corte:*

```
# Filtro butterworth en funcion de la frecuencia de corte

def butterworth(x,wc):
    N = 1
    B,A = butter(N,wc)
    y = []

    for n in range(len(x)):
        if n == 0:
            y.append(x[0])
        else:
            y_n = B[0]*(x[n] + x[n-1]) - A[1]*y[n-1]
            y.append(y_n)
    return y
```

3.5. SuperSmoother

El último filtro que analizaremos es el SuperSmoother, una versión modificada de un filtro Butterworth, optimizada para aplicaciones de trading. Este filtro se ha tomado de la referencia [7]. Se invita a los lectores a consultarla para una visión más en profundidad ya que aquí daremos solo unas pinceladas. Consideremos un filtro Butterworth de orden 2, su ecuación en diferencias es la siguiente:

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \tag{25}$$

donde a_0, a_1, a_2, b_0, b_1 y b_2 son los coeficientes del filtro para una frecuencia de corte dada ω_c. Si hacemos $b_1 = b_2 = 0$ obtenemos un filtro con menor *lag*, más óptimo para aplicaciones de trading. Por tanto, la ecuación anterior queda tal que así:

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] + b_0 x[n] \tag{26}$$

A continuación, y con el objetivo de añadir un cero a en la frecuencia de Nyquist (π radianes), introduciremos una media móvil de dos periodos en el filtro. Esto permite suavizar aún más la señal ya que estamos cancelando la máxima frecuencia. Por tanto, la ecuación del filtro SuperSmoother es la siguiente:

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] + \frac{b_0}{2}(x[n] + x[n-1]) \tag{27}$$

Si considreamos dos variables auxiliares k_1 y k_2 tales que:

$$k_1 = \exp\left(-\sqrt{2}\cdot\omega_c\cdot\frac{\pi}{2}\right), \quad k_2 = 2k_1\cdot\cos\left(\sqrt{2}\cdot\omega_c\cdot\frac{\pi}{2}\right)$$

podemos obtener los coeficientes del filtro de esta forma:

$$a_1 = k_2, \quad a_2 = -(k_1)^2, \quad b_0 = 1 - a_1 - a_2$$

Si representamos el filtrado en tiempo, obtenemos la siguiente imagen:

Figura 12: Filtrado con SuperSmoother para $\omega_c = 0.044$

Y la respuesta en frecuencia tiene el siguiente aspecto:

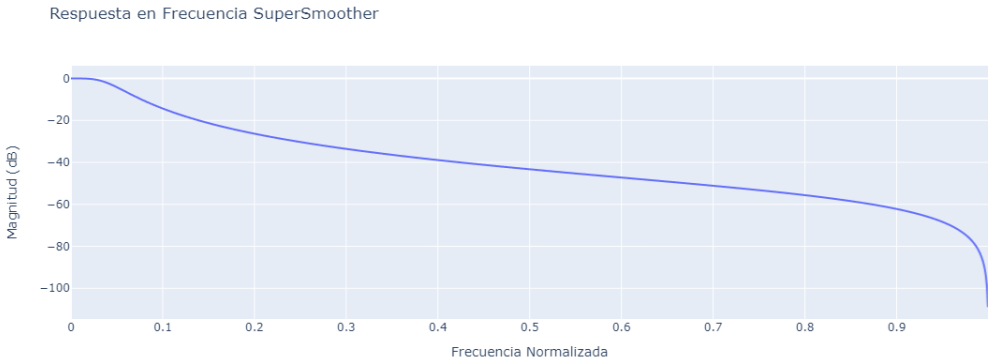

Figura 13: Respuesta en frecuencia de un filtro SuperSmoother con $\omega_c = 0.044$

Esta respuesta en frecuencia es muy parecida a la que tiene un filtro Butterworth de orden 1. Sin embargo, en el SuperSmoother la atenuación aumenta mucho más rápido que en el Butterworth. Las bajas frecuencias se mantienen con ganancia cercana a la unidad (0 dB) pero rápidamente, al subir en frecuencia, se atenúan considerablemente. En la siguiente sección analizaremos si realmente esta característica del SuperSmoother propicia resultados mejores en el trading algorítmico.

Para finalizar, hemos implementado el SuperSmoother de la siguiente forma:

Algoritmo 8 *Filtrado temporal con un SuperSmoother a partir de la frecuencia de corte:*

```
# Filtro supersmoother en funcion de la frecuencia de corte

def smooth(x,wc):

    a = np.exp(-np.sqrt(2)*np.pi*wc/2)
    b = 2*a*np.cos(np.sqrt(2)*np.pi*wc/2)

    c2 = b
    c3 = -a*a
    c1 = 1-c2-c3

    y = []

    for n in range(len(x)):
        if n == 0:
            y.append(x[0])
        else:
            y_n = (c1/2)*(x[n] + x[n-1]) + c2*y[n-1] + c3*y[n
                -2]
            y.append(y_n)
    return y
```

Si unificamos todos los filtros bajo la misma frecuencia de corte ω_c, podemos comparar las diferentes respuestas en frecuencia, dando lugar a la siguiente imagen:

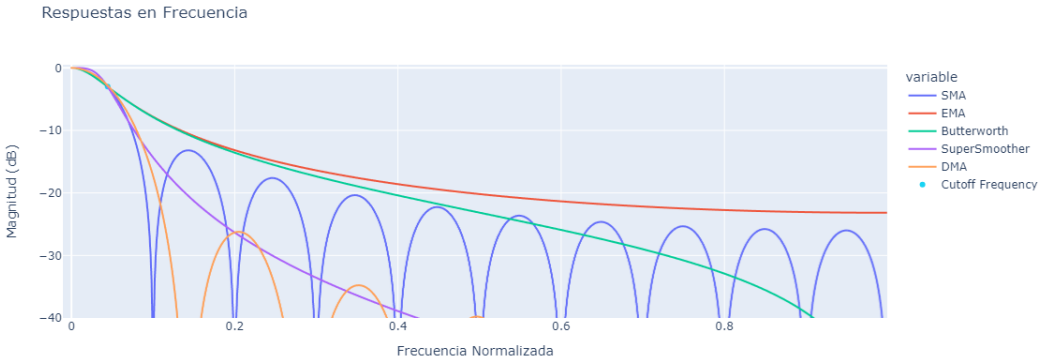

Figura 14: Comparación de las respuestas en frecuencia de los filtros

De cara a la siguiente sección, lo que haremos será definir una ω_c arbitraria y posteriormente optimizar las estrategias de trading para obtener la ω_c óptima que maximice los beneficios del sistema.

4 Trading Algorítmico

Una vez tenemos los filtros listos, vamos a desarrollar nuestras estrategias de trading algorítmico. Desarrollaremos una estrategia común, y la probaremos con los 5 filtros paso-bajo que hemos propuesto en el capitulo anterior. Para esta tarea, nos ayudaremos de las referencias [8] y [6].

4.1. Desarrollo de una Estrategia

La estrategia que vamos a emplear se suele conocer como un cruce de medias, y consiste en filtrar la señal dos veces, con un mismo filtro, pero con dos frecuencias de corte diferentes. De esta forma obtenemos dos señales suavizadas distintas, que cuando se crucen nos darán las señales de compra y venta. En nuestro sistema, podemos operar tanto en largo (comprar la acción, esperando que suba) como en corto (vender la acción sin tenerla en propiedad, esperando que baje). A continuación, se detalla la lógica de la estrategia:

1. Elegir un filtro: SMA, EMA, DMA, Butterworth o SuperSmoother.
2. Definir dos frecuencias de corte (arbitrarias), ω_{c1} y ω_{c2} de forma que $\omega_{c1} > \omega_{c2}$.
3. Filtrar la señal de bolsa $x[n]$ con el filtro elegido obteniendo $y_1[n]$ e $y_2[n]$.
4. Si $y_1[n-1] > y_2[n-1]$ y $y_1[n] \leq y_2[n]$, cerramos el corto, abrimos un largo.
5. Si $y_1[n-1] < y_2[n-1]$ y $y_1[n] \geq y_2[n]$, cerramos el largo, abrimos un corto.

De forma informal y para hacer más intuitiva la explicación de la estrategia, nos referiremos a $y_1[n]$ como el filtro rápido y a $y_2[n]$ como el filtro lento. La idea que subyace en esta estrategia es que cuando el filtro rápido cruza de abajo a arriba al filtro lento, es un indicador de que el precio puede estar cambiando de tendencia y comenzar a subir (interesa ponerse largos). De forma alternativa, cuando el filtro rápido cruza de arriba a abajo al lento, indica que el precio puede estar a punto de bajar (interesa ponerse cortos).

Figura 15: Cruce de dos filtros SMA con $\omega_{c1} = 0.015$ y $\omega_{c2} = 0.005$

En la figura anterior se observan dos cruces, el primero indicaría señal de venta o corto y el segundo una compra o largo.

4.2. Implementación en Python

De cara a la implementación en Python, usaremos la libreria *backtesting.py* [3]. Nos será de gran ayuda ya que nos proporcionará el motor de backtesting y los métodos de optimización. Sin embargo, debemos diseñar la lógica de la estrategia (definida anteriormente). Haremos esto para cada uno de los 5 filtros. A continuación se muestra la estrategia para el filtro SMA:

Algoritmo 9 *Estrategia para el filtro SMA segun la librería backtesting.py:*

```
from backtesting import Strategy, Backtest
from backtesting.lib import crossover

# Estrategia SMA
class sma_strat(Strategy):

    # Parametros iniciales
    wc1 = wc1_0; wc2 = wc2_0

    def init(self):
        # Calcula los indicadores
        self.filt1 = self.I(sma, self.data.Close, self.wc1)
        self.filt2 = self.I(sma, self.data.Close, self.wc2)

    def next(self):
        # Cierra el corto, abre un largo
        if crossover(self.filt1, self.filt2):
            self.position.close()
            self.buy()

        # Cierra el largo, abre un corto
        elif crossover(self.filt2, self.filt1):
            self.position.close()
            self.sell()
```

4.3. Backtesting

Una vez diseñada e implementada la lógica de nuestra estrategia, es el momento de ponerla a prueba en datos de cotizaciones históricas, para ver cómo se ha comportado en el pasado. Este proceso recibe el nombre de backtesting. Si la estrategia ha dado una buena rentabilidad en el pasado, es un indicador de que podría seguir haciéndolo en el futuro. A la hora de realizar el backtesting hemos tenido en cuenta lo siguiente:

- Usaremos como señal los datos de cotizaciones de la empresa Microsoft (con ticker MSFT), desde el año 2010 hasta el presente
- Supondremos que las comisiones de compra y venta son del 0 %
- Tomaremos como frecuencias de corte iniciales $\omega_{c1} = 0.044$ y $\omega_{c2} = 0.0088$
- Comenzaremos con un capital inicial V_i, de \$10.000
- La tasa libre de riesgo r es constante y vale $r = 5.5\,\%$

El código del backtesting se puede ver a continuación (de nuevo, hemos usado la librería *backtesting.py*):

Algoritmo 10 *Backtesting de las estrategias de trading algorítmico a través de la libreria backtesting.py:*

```
from backtesting import Backtest
import yfinance as yf
import pandas as pd
import numpy as np

# Cotizaciones
x = yf.download('MSFT', start = '2010-01-01', progress = False)

# Parametros del backtesting
iCash = 10000; com = 0; r = 0.055

# Parametros iniciales filtros
wc1_0 = 0.044; wc2_0 = 0.0088

# Ejecucion de los backtestings
bt_sma = Backtest(x, sma_strat, cash = iCash, commission = com)
bt_ema = Backtest(x, ema_strat, cash = iCash, commission = com)
bt_butter = Backtest(x, butter_strat, cash = iCash, commission
    = com)
bt_smooth = Backtest(x, smooth_strat, cash = iCash, commission
    = com)
bt_dma = Backtest(x, dma_strat, cash = iCash, commission = com)
```

Una de las ventajas de usar la librería de backtesting mencionada es que podemos visualizar gráficamente y de una forma sencilla la ejecución de la estrategia de trading, así como sus resultados. En la siguiente figura se puede ver la ejecución de las operaciones y arriba del todo el rendimiento porcentual de la estrategia:

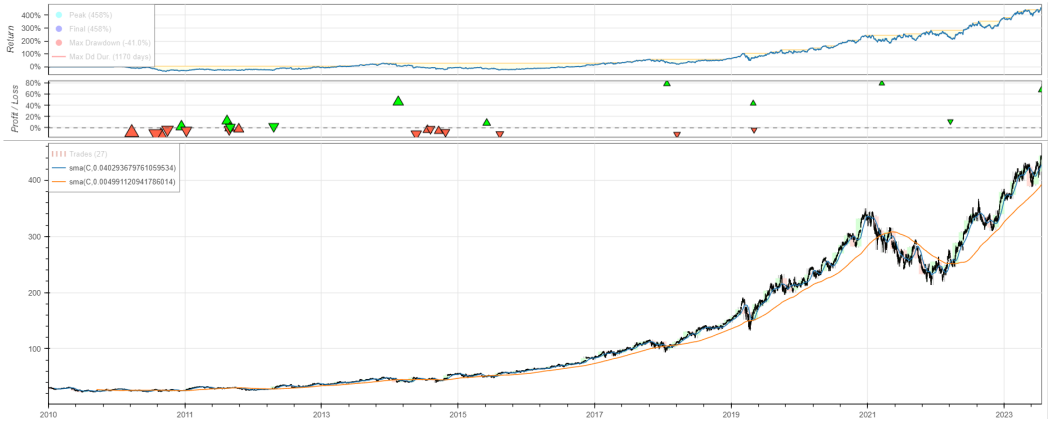

Figura 16: Ejecución del backtesting para la estrategia con SMA

Aunque la visualización gráfica puede ser útil para evaluar el éxito del backtesting, en la práctica solo la utilizaremos para asegurarnos de que las operaciones se están ejecutando según la lógica deseada. Por otro lado, el análisis del rendimiento del backtesting lo haremos a través de las métricas estadísticas que comentaremos en la siguiente sección.

4.4. Métricas clave

A la hora de evaluar el éxito de un backtesting, debemos tener en cuenta una serie de métricas estadísticas que nos dan información acerca de este. A continuación de detallan las más relevantes:

1. **Retornos (%)**: rendimiento porcentual de la estrategia al final del periodo.

2. **Retornos Anualizados (%)**: tasa de retorno anual media durante el periodo de estudio. Esta métrica es de especial relevancia ya que se suele comparar con los retornos anualizados del mercado. Históricamente, la rentabilidad anualizada del mercado de acciones americano ha sido del 10 %. Nuestro objetivo será encontrar estrategias de trading que puedan superar esta rentabilidad. La fórmula para el cálculo del retorno anualizado es:

$$\mu = \left(\frac{V_f}{V_i}\right)^{\frac{1}{n}} - 1$$

   donde V_f es el valor final de la cartera y n el número de años.

3. **Volatilidad Anualizada (%)**: representa la variabilidad de los retornos de la estrategia de trading durante un año. Se puede entender de manera más informal como el riesgo de pérdida que asumimos al invertir en la estrategia durante un año. Podemos calcularla a partir de la desviación estándar de los retornos de la estrategia para un periodo dado:

$$\sigma = \sigma_{\text{periodo}} \times \sqrt{n}$$

   donde n es el número de periodos empleados para el cálculo. Por ejemplo, si tenemos los retornos diarios de la estrategia, tendremos que $n = 252$ por ser dicho número los días que cotiza la acción o los días en los que nuestro sistema opera en el mercado.

4. **Ratio de Sharpe**: aunque los retornos anualizados son una buena métrica para medir el éxito de un backtesting (es decir, lo mucho que gana la estrategia) es necesario tener en cuenta también el riesgo que asumimos al invertir en ella. No es lo mismo ganar un 15 % anual asumiendo un 15 % de riesgo que ganar ese mismo 15 % asumiendo un riesgo del 30 %. El Ratio de Sharpe mide la rentabilidad ajustada al riesgo, teniendo en cuenta la tasa libre de riesgo del mercado (que suele ser la rentabilidad del bono del tesoro americano a 10 años). La fórmula es la siguiente:

$$S = \frac{\mu - r}{\sigma}$$

   Buscaremos tener un Ratio de Sharpe superior a 1, lo que implica tener una rentabilidad superior al riesgo que asumimos, teniendo en cuenta la tasa libre de riesgo del mercado. Un Ratio de Sharpe negativo implica que es más rentable invertir en los mercados monetarios o en bonos del tesoro que en nuestra estrategia.

5. **Número de *Trades***: el total de operaciones que ha arrojado la estrategia al final del backtesting. Nos interesa tener un número elevado de *trades* para que la estrategia tenga validez estadística.

6. **Máximo Drawdown (%)**: la máxima pérdida acumulada durante el periodo de backtesting. Es una métrica de riesgo que nos habla de cuánto podemos llegar a perder para el total del periodo de análisis (a diferencia de la volatilidad anualizada que nos indica el riesgo para un año).

7. **Retornos/Máximo Drawdown**: esta es una métrica similar al Ratio de Sharpe en cuanto a que ajusta la rentabilidad al riesgo, pero en este caso no lo calculamos de forma anual, sino usando todo el periodo de estudio. Para calcular este ratio, lo dividiremos entre el número de años que dure el backtesting. Buscaremos que este cociente sea lo más alto posible.

8. **Trades/Años**: número de operaciones anuales (de media). Es una métrica que nos aporta más información que el número de *trades* totales ya que de esta forma tenemos en cuenta la duración del backtesting.

Usaremos estas métricas para evaluar el rendimiento de las estrategias y comprobar qué filtro es el que genera una mayor rentabilidad. A continuación mostramos el cálculo en Python de estas métricas:

Algoritmo 11 *Calculo de los parámetros de los diferentes backtestings:*

```
# Estadisticas de los backtestings
stats_sma = bt_sma.run()
stats_ema = bt_ema.run()
stats_butter = bt_butter.run()
stats_smooth = bt_smooth.run()
stats_dma = bt_dma.run()

stats = {'SMA':stats_sma,'EMA':stats_ema, 'Butterworth':
    stats_butter,'SuperSmoother':stats_smooth, 'DMA':stats_dma}

# Parametros relevantes
mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
    maxDD = []; rets = []; retsDD = [];

for name, stat in stats.items():
    mu.append(stat['Return (Ann.) [%]'])
    sigma.append(stat['Volatility (Ann.) [%]'])
    ntrades.append(stat['# Trades'])
    ytrades.append(365*stat['# Trades']/len(x))
    sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
        Volatility (Ann.) [%]'])
    maxDD.append(stat['Max. drawdown [%]'])
    rets.append(stat['Return [%]'])
    retsDD.append(-365*stat['Return [%]']/(stat['Max. drawdown
        [%]']*len(x)))

res = pd.DataFrame({'Estretegia':stats.keys(), 'Retornos (%)':
    rets,'Retornos Anualiz. (%)':mu, 'Volatilidad Anualiz. (%)':
    sigma, 'Trades':ntrades,'Trades/Ano':ytrades, 'Ratio Sharpe'
    :sharpe, 'Max drawdown (%)': maxDD, 'Retornos/drawdown':
    retsDD,'wc1':wc1_0, 'wc2': wc2_0})

# Curvas de equity
```

```
df_equity = pd.DataFrame({name: stats['_equity_curve']['Equity']
    ] for name, stats in stats.items()})

# Curvas de rendimiento en %
df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
    iCash - 1 for name, stats in stats.items()})

# Curvas de drawdown
df_DD = pd.DataFrame({name: stats['_equity_curve']['drawdownPct
    '] for name, stats in stats.items()})
```

4.5. Resultados iniciales

Para acabar la sección mostramos, los resultados de los backtestings en forma de tabla:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
SMA	123.21	5.72	26.93	33	3.31	0.01	-50.70	0.24
EMA	55.00	3.08	26.16	60	6.02	-0.09	-58.94	0.09
Butterworth	90.76	4.58	26.62	60	6.02	-0.03	-59.09	0.15
SuperSmoother	9.95	0.66	26.28	40	4.01	-0.18	-54.78	0.02
DMA	136.51	6.14	26.98	31	3.11	0.02	-55.99	0.24

Cuadro 1: Comparación de estrategias de trading

Como se puede observar, la estrategia DMA es la que mayor Ratio de Sharpe tiene, y mayor rentabilidad total y ajustada al drawdown. La SMA tiene una menor rentabilidad que la DMA pero mejora en cuanto a riesgo al tener una menor volatilidad anualizada y menor drawdown. Sin embargo, como nuestro objetivo es buscar estrategias que ofrezca la mayor rentabilidad ajustada al riesgo, nos quedaremos con la estrategia DMA.

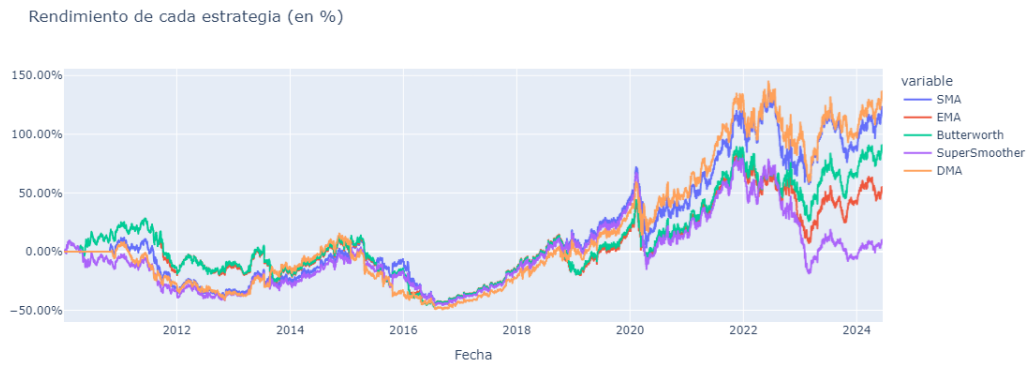

Figura 17: Rentabilidades de las distintas estrategias en función del tiempo

Si atendemos a las rentabilidades anualizadas y las comparamos con activos libres de riesgo o con el mercado, podemos afirmar que estas estrategias apenas baten la rentabilidad de, por ejemplo, los bonos del tesoro americano a un año (que en Junio de 2024 dan una rentabilidad de alrededor del 5.1 %). Además, quedan muy lejos de la rentabilidad del mercado (que es de cerca del 10 %). Por tanto, no las tomamos como válidas, luego no invertiríamos en ellas.

Por otro lado, recordemos que hemos establecido las frecuencias de corte de forma arbitraria. En la siguiente sección, nos encargaremos de buscar las frecuencias de corte óptimas que maximicen las rentabilidades de estas estrategias.

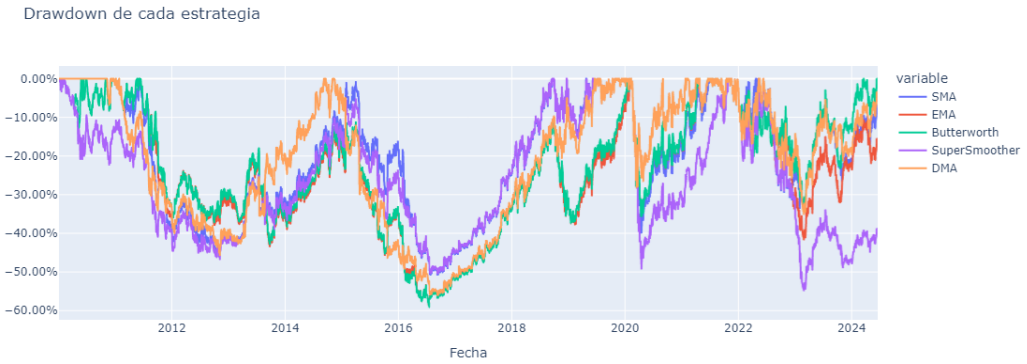

Figura 18: Drawdown de las diferentes estrategias

5 Optimización

En esta sección explicaremos en detalle el proceso de optimización de nuestras estrategias de trading algorítmico. Además, hablaremos de los sesgos más frecuentes que debemos evitar a la hora de optimizar y propondremos dos métodos, el exhaustivo y el Bayesiano siendo este último más sofisticado que el primero.

5.1. Consideraciones Previas

Podemos definir la optimización de una estrategia de trading como la búsqueda de la mejor combinación de parámetros que permita que los resultados sean lo más elevados y estables posible a lo largo del tiempo [6]. En nuestro caso, consistirá en buscar los parámetros ω_{c1} y ω_{c2} óptimos.

El mayor riesgo que tenemos al optimizar es caer en el llamado *data snooping bias* o sesgo de sobreoptimización. Esto sucede cuando afinamos tanto los parámetros del sistema que llegamos a modelar el ruido de los datos, dando lugar a resultados satisfactorios en el backtesting, pero catastróficos en en el mercado real. Por tanto, debemos de tener mucho cuidado a la hora de optimizar la estrategia. Algunas de las prácticas más comunes para evitar la sobreoptimización y hacer un buen backtesting son:

- Usar sistemas sencillos con pocos indicadores. Cuanto menor sea el número de variables a optimizar menor probabilidad de sobreoptimización.

- Definir unos rangos bien acotados para los parámetros a optimizar. De esta forma nos ahorraremos tiempo de computación.
- Definir un paso o un incremento Δs adecuado para el rango de los parámetros. Un Δs muy pequeño da lugar a tiempos de computación elevados y es probable que genere sobreoptimización.
- Definir un objetivo claro de optimización. Puede ser buscar el máximo retorno, la mayor rentabilidad ajustada al riesgo, la menor volatilidad, minimizar el drawdown... Todo dependerá del objetivo de la estrategia y de nuestra tolerancia al riesgo.

En nuestro caso, nos limitaremos a optimizar los dos parámetros que hemos mencionado anteriormente, con un rango bastante acotado tal que:

$$\omega_{c1}, \omega_{c2} \in [0.0044, 0.089]$$

Además, definiremos incremento $\Delta s = 0.0025$, de forma que el número total de combinaciones será igual a 1156 (apenas unos minutos de computación). Por último, nuestro objetivo será maximizar el Ratio de Sharpe, S, es decir, obtener la mayor rentabilidad ajustada al riesgo.

5.2. Método Exhaustivo

El primer método que usaremos para optimizar el el método exhaustivo o *grid search* [4]. En términos más coloquiales, es un método por fuerza bruta, es decir, vamos a probar todas las combinaciones posibles de nuestras frecuencias de corte para ver la combinación de estas que maximiza el parámetro que deseamos (el Ratio de Sharpe). Tras optimizar por este método, obtenemos los siguientes resultados:

Estrategia	ω_{c1}	ω_{c2}
SMA	0.00693	0.00443
EMA	0.00693	0.00443
Butterworth	0.00693	0.00443
SuperSmoother	0.01443	0.00693
DMA	0.01443	0.00443

Cuadro 2: Valores de ω_{c1} y ω_{c2} óptimos para diferentes estrategias

Un aspecto relevante de la optimización es que los valores de ω_{c2} en algunas estrategias se han situado en el límite inferior del rango que hemos establecido. Esto sucede porque la acción que estamos usando como señal presenta una tendencia alcista a muy largo plazo, por lo que el retorno por comprar y mantener la acción va a ser bastante elevado. Por tanto, el sistema se optimiza haciendo el menor número de operaciones posibles, para estar la mayor parte del tiempo comprado (como si tratase de replicar el retorno de comprar la acción y mantenerla). Para conseguir esto, se establece una frecuencia de corte muy baja de forma que el filtro suavice mucho la señal y hayan pocas operaciones pero que se prolonguen mucho en el tiempo.

Podemos afirmar, por tanto, que el valor de ω_{c2} tenderá a buscar siempre la cota inferior que establezcamos, por el tipo de señal tan alcista que estamos usando.

Esto es un factor de sobreoptimización claro. En la siguiente sección presentaremos algunas técnicas de validación estadística que nos arrojarán más información sobre este aspecto. A continuación se muestra un mapa de calor de la optimización de parámetros de la estrategia SMA. Como se puede apreciar, los mejores resultados se dan para un ω_{c2} bajo.

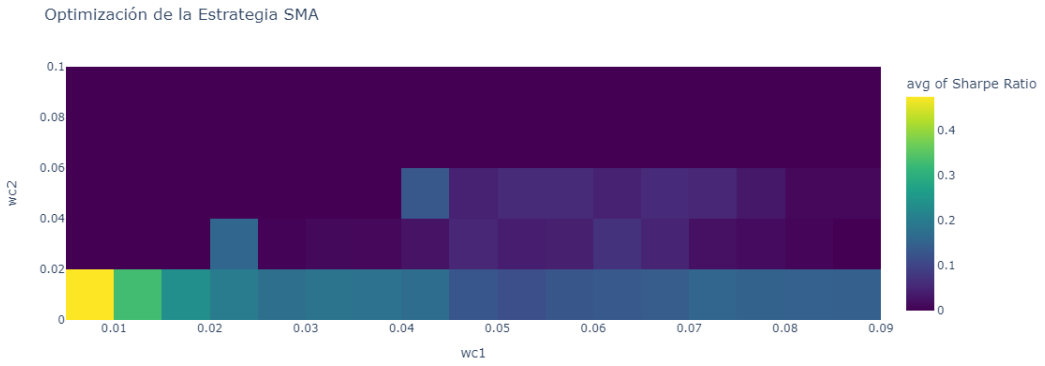

Figura 19: Mapa de calor de la optimización de la estrategia SMA

Por otro lado, los resultados de la estrategia optimizada son los siguientes:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
SMA	707.89	15.57	29.25	11	1.10	0.34	-41.03	1.73
EMA	630.23	14.76	29.25	14	1.40	0.32	-41.96	1.51
Butterworth	535.09	13.66	28.88	14	1.40	0.28	-53.91	0.99
SuperSmoother	725.02	15.74	29.37	16	1.60	0.35	-47.80	1.52
DMA	710.05	15.59	29.03	10	1.00	0.35	-45.98	1.55

Cuadro 3: Comparación de estrategias de trading optimizadas (método exhaustivo)

Como se puede observar, se ha optimizado el Ratio de Sharpe, S, a costa de realizar menos operaciones (por lo que hemos comentado anteriormente). Los retornos anualizados son más que aceptables ya que superan el *benchmark* del mercado que hemos situado en un 10 % anual. Sin embargo, el número de operaciones por año es muy bajo, lo que pone en duda la validez estadística del sistema (más adelante analizaremos este aspecto).

Viendo estos resultados, nos quedaríamos con la estrategia SuperSmoother, por tener un mayor S, el mayor número de trades y además porque es la única que cuyo parámetro ω_{c2} no ha buscado el límite del rango, que es una buena señal de que estamos evitando la sobreoptimización.

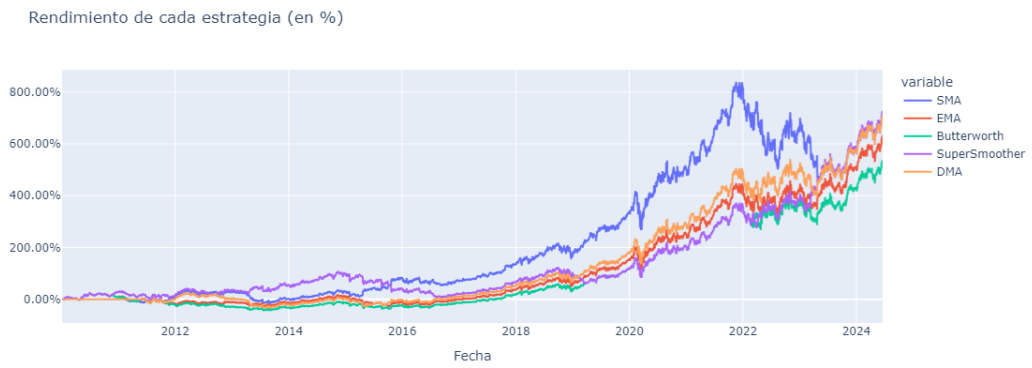

Figura 20: Retornos de las estrategias optimizadas (método exhaustivo)

5.3. Método Bayesiano

En este apartado haremos uso de un método de optimización más sofisticado que el anterior, que reducirá el tiempo de computación. Se trata del método Bayesiano u optimización Bayesiana. Esta técnica es especialmente útil para funciones ruidosas y sin solución analítica (como en nuestro caso). Además, es bastante eficiente para funciones costosas de evaluar. La idea principal de este método es crear un modelo probabilístico de la función objetivo e iterar probando varias combinaciones de parámetros, actualizando el modelo para tomar decisiones informadas acerca de qué parámetros evaluar a continuación.

Está demostrado que este método reduce el tiempo de computación en comparación con el método exhaustivo, que hemos expuesto anteriormente. Para más información, consultar [1]. Tras optimizar los parámetros con este método, hemos obtenido:

Estrategia	ω_{c1}	ω_{c2}
SMA	0.009155	0.005872
EMA	0.011849	0.007573
Butterworth	0.017365	0.004527
SuperSmoother	0.024695	0.004901
DMA	0.020196	0.007021

Cuadro 4: Valores de ω_{c1} y ω_{c2} para cada estrategia

En este caso, ningún parámetro ha buscado el límite del rango de optimización, lo que indica que no se está sobreoptimizando el sistema, lo cual es muy buena señal. Esto es una mejora notable respecto del método de optimización exhaustiva.

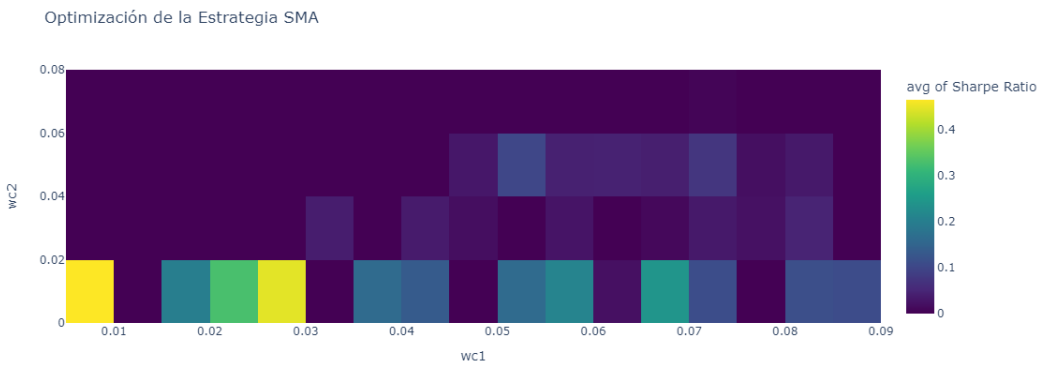

Figura 21: Mapa de calor de la optimización de la estrategia SMA

Atendiendo a los resultados de la optimización, podemos observar lo siguiente:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
SMA	498.56	13.19	28.35	15	1.50	0.27	-29.53	1.69
EMA	212.21	8.20	27.32	28	2.81	0.10	-63.84	0.33
Butterworth	324.29	10.53	27.92	30	3.01	0.18	-58.87	0.55
SuperSmoother	390.74	11.65	28.32	24	2.41	0.22	-52.40	0.75
DMA	454.25	12.59	28.22	21	2.11	0.25	-44.06	1.03

Cuadro 5: Comparación de estrategias optimizadas (método Bayesiano)

El sistema que mejor rinde es el de SMA, aunque presenta muy pocas operaciones. Algo similar pasa con el DMA. Por tanto, tomaremos el sistema SuperSmoother por tener un número aceptable de operaciones y tener una rentabilidad ajustada al riesgo, S, adecuada. Además, tiene un retorno anualizado por encima del mercado, que es lo que buscamos.

5.4. Conclusiones de la Optimización

Tras emplear dos métodos de optimización distintos, nos quedaremos con el Bayesiano debido a que proporciona más operaciones en cada estrategia (lo cual hace los resultados estadísticamente más significativos), además, los parámetros de optimización no han buscado los límites del rango con esta técnica, indicando escasa sobreoptimización. Pese a que los retornos y el Ratio de Sharpe son más bajos, las estrategias aparentan ser más robustas con este segundo método. Por último, tomaremos el sistema SuperSmoother como el mejor por lo que hemos comentado anteriormente.

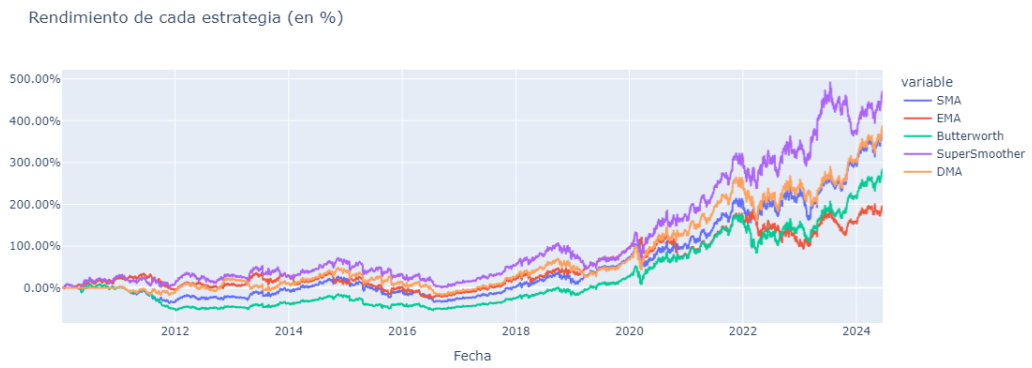

Figura 22: Retornos de las estrategias optimizadas (método Bayesiano)

6 Validación del Sistema

En esta última sección del trabajo haremos uso de técnicas estadísticas para validar nuestro sistema de trading. Para ello, elegiremos el mejor sistema tras la optimización, que como hemos comentado, es el sistema SuperSmoother. A partir de ahora trabajaremos exclusivamente con esta estrategia.

La validación consiste en poner a prueba el sistema creando unas condiciones que simulen el mercado real, de forma que si pasa estos tests, podemos considerar que el sistema es apto para operar en tiempo real. Además de esto, también someteremos a la estrategia a situaciones extremas y a escenarios nuevos para comprobar su robustez. Realizaremos las siguientes pruebas:

- **Análisis en otros mercados**: comprobaremos el desempeño de la estrategia en otras acciones, mercados y clases de activos. Si la estrategia se comporta bien, será un indicador de que es robusta.
- **Análisis con comisiones**: introduciremos una pequeña comisión al hacer cada operación para hacer más realista la operativa.
- **Análisis de sensibilidad**: variaremos los parámetros del sistema para comprobar si sigue siendo efectivo. Si al cambiar los parámetros el sistema pierde, será un indicador de sobreoptimización.
- **Análisis de *slippage* extremo**: en los mercados poco líquidos, las operaciones suelen ejecutarse a un precio peor del establecido. Simularemos este escenario para comprobar la robustez de la estrategia ante eventos extremos de *slippage*.
- **Análisis de dependencia**: estudiaremos los retornos de cada operación e identificaremos si hay algún *trade* que aporte unas ganancias muy por encima de la media. Buscaremos que el retorno medio de las operaciones sea elevado y que no dependa de unas pocas operaciones positivas.
- **Análisis del drawdown máximo**: simularemos de forma aleatoria el orden de ejecución de las operaciones para estimar la peor racha de pérdidas posible (aunque el beneficio final será el mismo para todas las iteraciones). Esta prueba es clave para estimar el máximo drawdown posible del sistema.

- **Análisis IS/OOS**: probaremos el sistema en otro marco temporal (OOS) diferente al que hemos usado para optimizar sistema (IS). Esta prueba es una simulación del mercado real porque el sistema tratará con datos que nunca ha visto.

No es necesario que el sistema pase todas las pruebas a las que lo vamos a someter, pero si que es importante que su desempeño sea adecuado en la mayoría de ellas. Además, al hacer estos tests, caracterizaremos muy bien cómo se comporta la estrategia ante rachas perdedoras, situaciones extremas o eventos de poca liquidez, cambios en la volatilidad y el régimen de mercado...

En definitiva, la validación del sistema es un paso clave en el desarrollo de cualquier estrategia de trading algorítmico porque nos garantiza que la estrategia es válida para operar en tiempo real. En el anexo se puede encontrar el código correspondiente a esta sección. Por último, hemos usado de guía para este apartado la referencia [6].

6.1. Análisis en otros mercados

En primer lugar, probaremos el sistema SuperSmoother en otras acciones, mercados y activos. Hemos elegido para esta tarea las acciones de Santander (SAN), British Petroleum (BP) y Amazon (AMZN) que pertenecen a los mercados español, británico y americano respectivamente. Probaremos también la estrategia en los ETFs del oro (GLD), Nasdaq100 (QQQ) y S&P500 (SPY). A continuación se muestran los resultados:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
QQQ	170.58	7.13	21.52	19	1.90	0.08	-40.10	0.43
SPY	6.67	0.45	17.55	16	1.60	-0.29	-57.39	0.01
SAN	62.52	3.42	30.53	20	2.00	-0.07	-71.14	0.09
GLD	12.74	0.83	14.92	33	3.31	-0.31	-53.38	0.02
BP	-48.61	-4.50	23.81	34	3.41	-0.42	-70.03	-0.07
AMZN	211.86	8.19	35.10	20	2.00	0.08	-55.15	0.39

Cuadro 6: Resultados de la estrategia en diferentes activos

Como se puede observar, los activos que mejor se comportan son el QQQ y AMZN. Esto se debe a que hemos entrenado la estrategia con los datos de MSFT, que es una empresa tecnológica como podría ser AMZN. Además el QQQ es un índice de las 100 empresas tecnológicas más relevantes de Estados Unidos, luego se comporta similar a AMZN y a MSFT. En el resto de activos no acaba de comportarse de forma adecuada (salvo quizás en SAN). Podemos concluir que la estrategia está ligeramente sobreoptimizada para las acciones tecnológicas americanas.

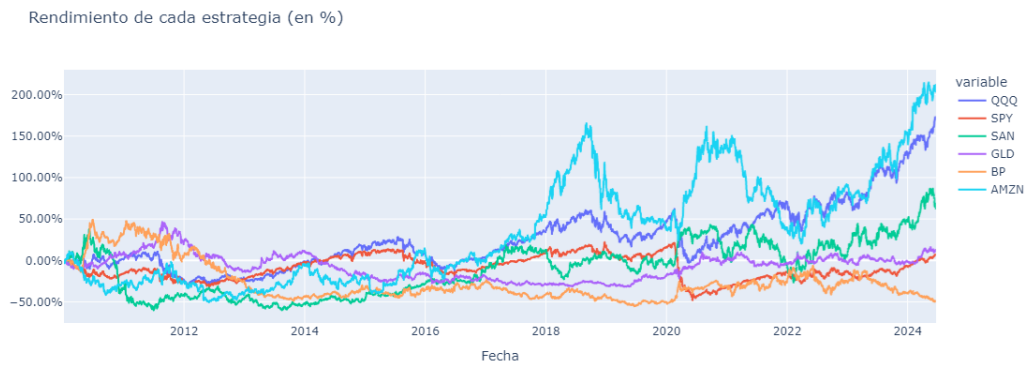

Figura 23: Comparativa de los retornos de la estrategia en diferentes mercados

6.2. Análisis con Comisiones

A continuación, supondremos una comisión del 1 % por operación, un escenario mucho más realista que el 0 % que habíamos supuesto inicialmente. Los resultados son:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
Sin Comisión	388.20	11.60	28.30	24	2.41	0.22	-52.40	0.74
Con Comisión	277.06	9.62	27.90	24	2.41	0.15	-60.51	0.46

Cuadro 7: Comparación de estrategias de trading con y sin comisión

Claramente los resultados empeoran, y lo hacen en función al número de operaciones realizadas. En algunos brokers se pueden conseguir comisiones por debajo del 1 %, así que en ese caso la estrategia no empeoraría tanto.

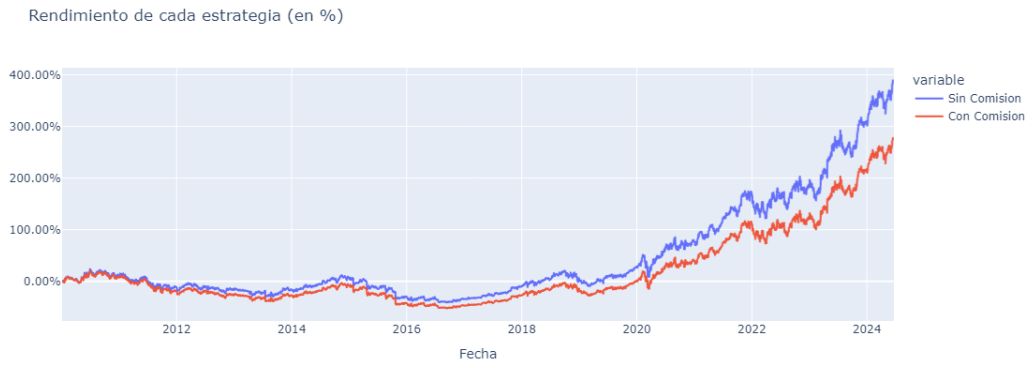

Figura 24: Rentabilidad con y sin comisiones

6.3. Análisis de Sensibilidad

Continuaremos con el análisis de sensibilidad. En este apartado, variaremos los parámetros ω_{c1} y ω_{c2} ligeramente y estudiaremos cómo cambia el rendimiento la estrategia. Si no empeora mucho, será una prueba de que es una estrategia robusta. Llevaremos a cabo 10 iteraciones, que hemos obtenido tomando valores de una gaussiana de media nula y desviación estándar igual a 0.1:

	$\Delta\omega_{c1}(\%)$	$\Delta\omega_{c2}(\%)$	μ
0	-6.79	-9.09	9.49
1	-5.22	-10.82	8.01
2	28.02	1.89	6.93
3	6.50	10.66	7.97
4	-10.37	-2.53	10.43
5	18.98	2.01	9.54
6	0.67	6.25	9.10
7	3.57	-1.33	11.24
8	-15.79	-7.20	8.35
9	-1.78	-1.79	10.69

Cuadro 8: Comparación de retornos anualizados para diferentes parámetros

Como era de esperar, si variamos mucho los parámetros, los retornos bajan, a mayor variación de parámetros, peor desempeño, por lo general. Sorprendentemente, en algún caso el rendimiento ha mejorado ligeramente, esto podría indicar que el sistema no está excesivamente sobreoptimizado ya que hay más máximos de rendimiento para otros parámetros. En la siguiente figura se puede apreciar mejor el presente análisis.

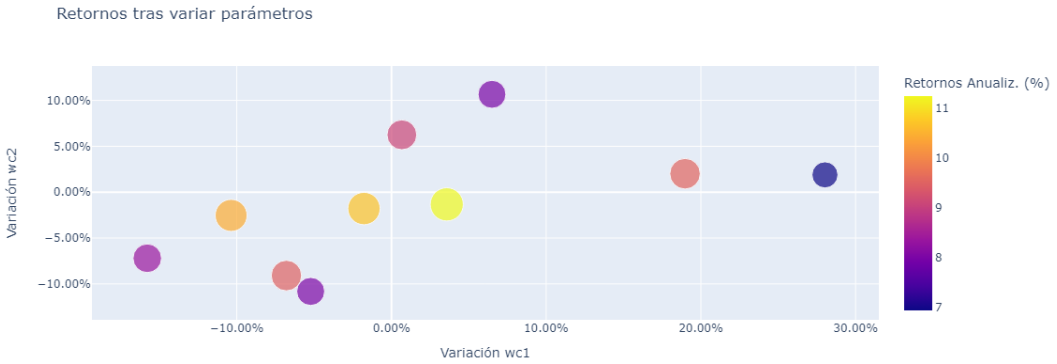

Figura 25: Análisis tras variar parámetros

A medida que nos alejamos del origen de coordenadas, los colores de los puntos se oscurecen, pero el rendimiento no baja excesivamente, luego damos por válido este test (sobre todo porque hemos conseguido mejorar el rendimiento, que es un signo de que el sistema no está sobreoptimizado).

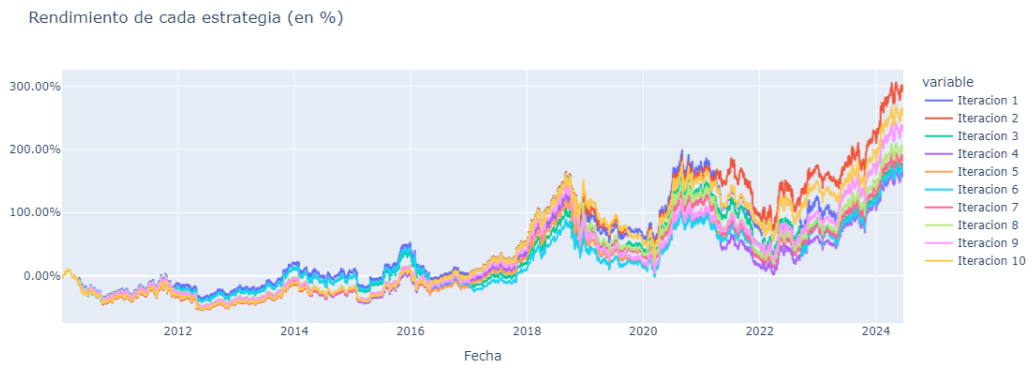

Figura 26: Comparativa del retorno tras variar parámetros

6.4. Simulación de *Slippage* Extremo

El siguiente paso será simular un caso extremo de *slippage*. Supongamos que el mercado sufre un *crash* bursátil. En este escenario, la liquidez se retira y los *spreads* (diferencias entre los precios de compra y venta) se ensanchan. Esto produce que a la hora de operar, los *trades* se ejecuten a un precio peor de lo esperado (porque no hay contrapartida para nuestra operación a ese precio exacto, en ese momento preciso).

Vamos a simular el escenario descrito. Supongamos que el *slippage* se modela con una gaussiana de media nula y desviación estándar del 0.015, es decir, que en un 68 % de los casos el *slippage* será de menos del 1.5 %. Asumiremos este contexto para todas las operaciones del sistema. Este es claramente un escenario extremo y no ocurre a menudo, pero nos sirve para testear el sistema en un escenario inusual. Tomaremos solo valores negativos de esta distribución (ya que el *slippage* implica siempre peores precios y nunca mejores). Tras simular se pueden observar los siguientes resultados:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
Sin Slippage	388.205	11.599	28.305	24	2.406	0.215	-52.400	0.743
Con Slippage	233.075	8.684	27.727	24	2.406	0.115	-63.883	0.366

Cuadro 9: Comparativa de estrategias con y sin *slippage*

donde claramente el caso con *slippage* es peor y empeora en función al número de operaciones. En este caso, obtendríamos una rentabilidad anualizada por debajo de la del mercado, pero recordemos que se trata de un caso extremo.

Si se opera en mercados muy líquidos como el americano, en horario normal de mercado, no deberíamos tener a penas *slippage* (sería cercano al 0 %), y si además se evita operar en eventos de mercado extremos, podríamos conseguir que nuestra estrategia opere sin este factor limitante.

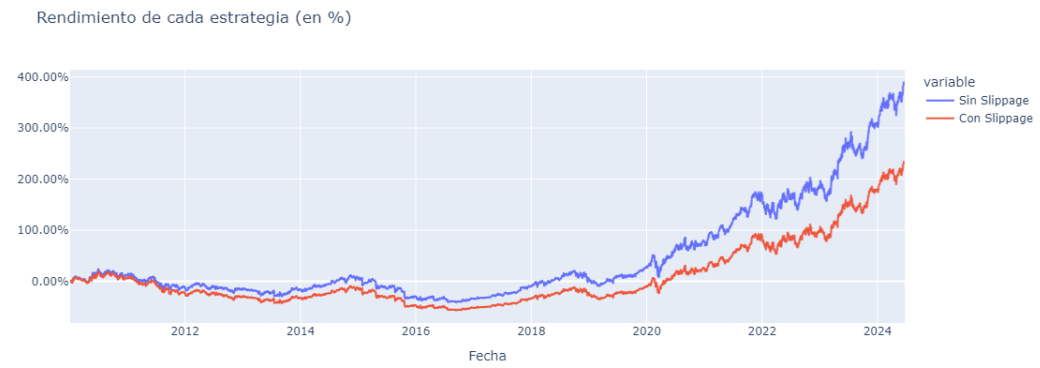

Figura 27: Comparativa retornos con y sin *slippage*

6.5. Análisis de Dependencia

A continuación realizaremos un análisis de dependencia. Esto consiste en tomar los retornos de las operaciones e identificar aquellos que se desvíen de la media, para comprobar si están influenciando en exceso al resultado final del sistema. Si la estrategia pierde o deja de ganar mucho si ignoramos estos *trades*, esto implicará que el sistema es áltamente dependiente de operaciones puntuales (que pueden deberse al azar de los mercados).

Por tanto, lo que buscamos es que la estrategia tenga unos retornos medios consistentes, con pocas desviaciones de la media y sin retornos extremos. En la siguiente imagen se puede apreciar el retorno por operación de la estrategia:

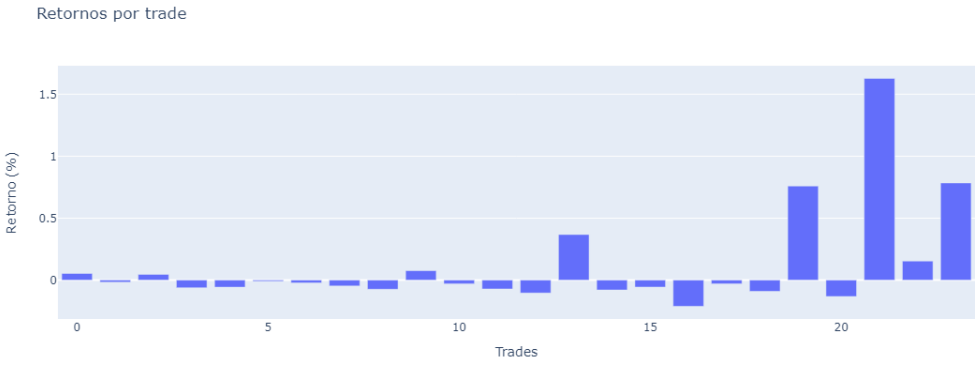

Figura 28: Retornos por trade

Se observan tres operaciones que superan claramente al resto de *trades* en cuanto a retorno. Si calculamos la media de los retornos, obtenemos un valor del 11.6 %, un valor muy por debajo de estas operaciones que acabamos de mencionar. La mediana se sitúa en el -2.87 %, lo cual nos indica que el sistema es altamente dependiente de unos pocos *trades* que hacen que la media pase a ser positiva. Si graficamos la distribución de los retornos podemos ver esto más claro:

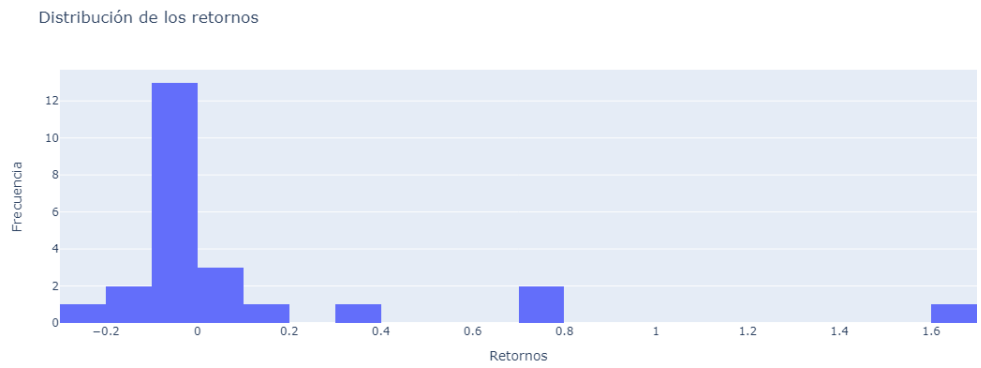

Figura 29: Distribución de los retornos

La distribución está centrada por debajo del 0 % y podemos ver unas pocas operaciones que se desvían mucho de ahí. Por tanto no damos por válida esta prueba. En algunos casos es admisible tener sistemas que arriesguen poco capital y tengan muchas operaciones perdedoras y unas pocas en las que se gane mucho, aunque en este caso hay que hacer una gestión del riesgo muy precisa.

6.6. Análisis del Drawdown Máximo

En este apartado estudiaremos el drawdown máximo al que se puede enfrentar nuestra estrategia. Para ello tomaremos los *trades* de la operativa y los desordenaremos, repitiendo este proceso de forma iterativa mediante una simulación de Monte Carlo. Las rentabilidades iniciales y finales de todas las trayectorias serán las mismas, pero podremos encontrar escenarios en los que el drawdown aumenta considerablemente (seguido de una racha ganadora después). De esta forma, podremos estimar el drawdown máximo del sistema.

Para ello, realizaremos 300 iteraciones con las operaciones desordenadas, como se muestra en la siguiente figura, y estudiaremos aquel caso en el que el drawdown supere al resto.

Figura 30: Análisis del drawdown máximo

Tras esta simulación, tenemos que el drawdown máximo es del 66.67 %. Este método es de especial importancia si decidimos operar con el sistema en tiempo real, ya que podríamos construir un intervalo de confianza a partir de una simulación como la que hemos realizado, para poder estimar la probabilidad de que se presente una racha perdedora muy notable. De esta forma, si el sistema presenta unas pérdidas que son estadísticamente muy improbables en base a este estudio, significará que la estrategia está fallando o que el mercado ha cambiado.

6.7. Análisis IS vs OOS

Por último, realizaremos la prueba IS/OOS donde IS significa *In Sample* y OOS *Out Of Sample*. Este método consiste en dividir un conjunto de datos en entrenamiento (IS) y test (OOS) y analizar si el rendimiento del sistema en los datos OOS es consistente con el rendimiento en IS. Por lo general interesa que la muestra de datos IS sea mayor que la de OOS.

En nuestro caso, los datos IS son los que hemos usado para optimizar el sistema. Por tanto, lo que haremos será designar otro marco temporal diferente al seleccionado para la optimización y probar el rendimiento del sistema en dicho periodo. Si los resultados OOS son iguales o mejores que los resultados IS, daremos por válida la prueba y significará que el sistema no está sobreoptimizado y es robusto ante nuevos datos (ya que la estrategia no ha tratado nunca con los datos OOS). En definitiva, el test en datos OOS simula la ejecución de la estrategia en un mercado real.

Para realizar este test, usaremos los datos de la cotización de MSFT desde el 01-01-2000 hasta el 01-01-2010. A continuación se muestra un gráfico de la cotización de MSFT en ese periodo.

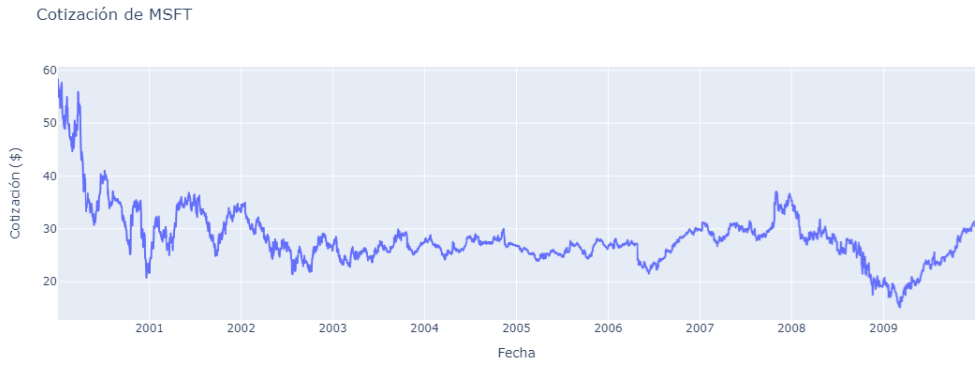

Figura 31: Cotización MSFT para el periodo OOS

Si realizamos el backtesting con los datos OOS, obtenemos los resultados que se muestran en la siguiente tabla:

Estrategia	Retornos	μ	σ	Trades	$\frac{\text{Trades}}{\text{Años}}$	S	Drawdown	$\frac{\text{Retornos}}{\text{Drawdown}}$
IS	388.20	11.60	28.30	24	3.48	0.22	-52.40	1.08
OOS	-90.94	-21.39	21.24	25	3.63	-1.27	-95.49	-0.14

Cuadro 10: Comparativa IS vs OOS

Como se observa, el test en datos OOS es realmente malo. Esto se puede deber a que la estrategia está optimizada para un periodo temporal, a partir de 2010 (figura 4.3), muy alcista, que es radicalmente distinto al de los datos OOS. Por lo tanto, o bien la estrategia no es apta para tendencias laterales (como en los datos OOS) o el sistema está sobreoptimizado para los datos IS.

Teniendo en cuenta que la estrategia ha tenido cierto éxito en los test de sensibilidad y de otros mercados, hay indicios para pensar que no está excesivamente sobreoptimizada. Por este motivo, podemos afirmar que la estrategia no rinde bien en periodos laterales, es decir, que se trata de una estrategia de seguimiento de tendencias.

Para evitar las pérdidas ocasionadas por operar en regímenes de mercado laterales, habría que añadir algún indicador que haga que el sistema se apague en estos momentos, por ejemplo, un filtro macroeconómico que active las operaciones en periodos de crecimiento económico como el periodo IS. Sin embargo, dicha mejora se escapa del alcance de este trabajo.

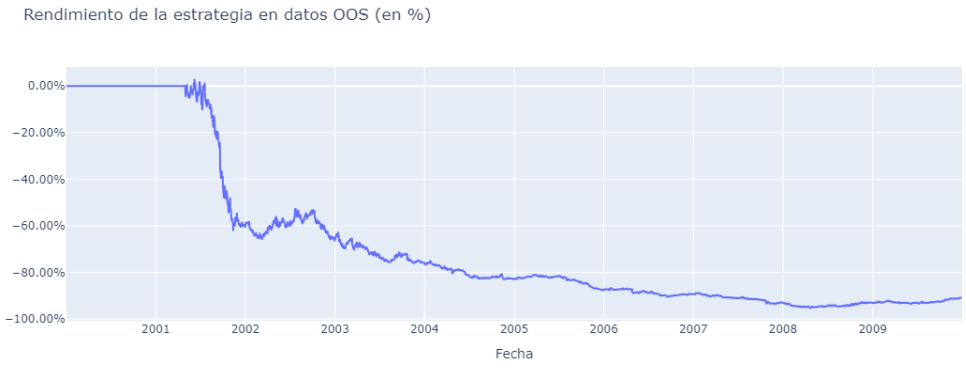

Figura 32: Rendimiento de la estrategia en datos OOS

7 Conclusiones

Para concluir este trabajo, haremos un repaso de los resultados que hemos obtenido hasta el momento. En primer lugar, hemos conseguido diseñar e implementar con éxito cinco filtros digitales paso-bajo aptos para usarse como indicadores de trading algorítmico.

Posteriormente, hemos diseñado una estrategia de trading en base a la combinación de dos filtros (iguales) con distintas frecuencias de corte ω_c, que da lugar a dos suavizados de la señal de bolsa que al cruzarse proporcionan indicadores de compra y venta. Hemos implementado dicha lógica y hemos creado cinco sistemas de trading, uno con cada filtro, y tras designar unas frecuencias de corte arbitrarias hemos podido comparar el rendimiento de las diferentes estrategias, mediante el estudio de diversas métricas.

A continuación hemos optimizado las estrategias en base a dos métodos, el exhaustivo y el Bayesiano, encontrado las frecuencias de corte óptimas. Hemos decidido tomar el segundo método por arrojar un mayor número de *trades* y por dar indicios de escasa sobreoptimización (ya que en el método exhaustivo los parámetros buscaban el límite del rango de optimización). Tras analizar los resultados hemos elegido el sistema con el filtro SuperSmoother por tener una combinación adecuada de rentabilidad ajustada al riesgo, número de *trades* y retornos anualizados por encima del mercado.

El siguiente paso ha sido validar el sistema a través de varios métodos estadísticos. De esta parte del trabajo se puede destacar como aspectos positivos que la estrategia funciona adecuadamente para el mercado americano, y en especial para acciones tecnológicas y que los parámetros ω_c no están excesivamente optimizados (como se ha comprobado en el análisis de sensibilidad). Respecto a la parte negativa, hemos averiguado que la estrategia funciona mal para periodos laterales, es decir, se trata de una estrategia de seguimiento de tendencias, y que además los beneficios finales dependen en exceso de unos pocos *trades* muy lucrativos, lo que indica escasa robustez. Además, el número de operaciones del sistema ha sido bajo y esto pone en duda la validez estadística de los resultados.

Como conclusión de este estudio, podemos afirmar que el uso de filtros digitales en el trading algorítmico es posible, aunque no hemos encontrado evidencias significativas de que tengan un poder predictivo superior al de otras estrategias de trading. Finalmente, tras estudiar los resultados de la validación, concluimos que la estrategia de SuperSmoother no es apta para operar en mercado real, a menos que se combine con algún indicador adicional para garantizar la operativa en tendencia.

Referencias

[1] Bayesian optimization with skopt, 2024, Junio. https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html.

[2] Digital filters and systems, 2024, Junio. https://tttapa.github.io/Pages/Mathematics/Systems-and-Control-Theory/Digital-filters/index.html.

[3] Documentación librería backtesting.py, 2024, Junio. https://kernc.github.io/backtesting.py/doc/backtesting/#gsc.tab=0.

[4] Wikipedia: Hyperparameter optimization, 2024, Junio. https://en.wikipedia.org/wiki/Hyperparameter_optimization.

[5] Antonio Albiol Colomer. Tratamiento digital de la señal: teoría y aplicaciones. *Valencia: Universidad Politecnica de Valencia*, 2007.

[6] Martí Castany Aparicio. *Guía de iniciación al trading cuantitativo: diseña paso a paso tus estrategias de inversión ganadoras*. Libros de Cabecera, 2019.

[7] John F Ehlers. *Cycle Analytics for Traders: Advanced Technical Trading Concepts*. John Wiley & Sons, 2013.

[8] Yves Hilpisch. *Python for Finance: Analyze big financial data*. O'Reilly Media, Inc., 2014.

[9] John G Proakis. *Digital signal processing: principles, algorithms, and applications, 4/E*. Pearson Education India, 2007.

[10] Li Tan and Jean Jiang. *Digital signal processing: fundamentals and applications*. Academic press, 2018.

A Código Filtros Digitales

```
#---------------------------------------------
# Tratamiento Digital de Senales Financieras
# Pablo Marchesi
# Junio 2024
#---------------------------------------------

from scipy.signal import butter
import pandas as pd
import numpy as np
from math import pi, cos, sin, sqrt, acos
from scipy.optimize import newton

# Siguiente potencia de dos
def nextpow2(N):
    n = 1
    while n < N: n *= 2
    return n

# Metodo de Newton para obtener el perido de la SMA a partir de la frecuencia de corte
def N_sma(wc):

    wc = wc*pi
    func = lambda N: np.sin(wc*N/2) - (N/np.sqrt(2))*(np.sin(wc/2))
    deriv = lambda N: (wc/2)*np.cos(wc*N/2) - (1/np.sqrt(2))*np.sin(wc/2)
    N_0 = pi/wc
    return int(np.round(newton(func, N_0, deriv)))

# Metodo de Newton para obtener el la frecuencia de corte a partir del periodo
def wc_sma(N, **kwargs):
    func = lambda w: sin(N*w/2) - N/sqrt(2) * sin(w/2)
    deriv = lambda w: cos(N*w/2) * N/2 - N/sqrt(2) * cos(w/2) / 2
    omega_0 = pi/N
    return newton(func, omega_0, deriv, **kwargs)/pi

# Filtro media movil en funcion de la frecuencia de corte
def sma(x, wc):

    N = N_sma(wc)
    y = pd.Series(x).rolling(N).mean()
    return y

# Metodo de Newton para obtener la frecuencia de corte de la EMA a partir de alpha
def wc_ema(alpha):
    return (acos((alpha**2 + 2*alpha - 2) / (2*alpha - 2)))/pi

# Metodo de Newton para obtener el parametro alpha de la EMA a partir de la frecuencia de corte
```

```
47 def alpha_ema(wc):
48     wc = wc*pi
49     B = 2*(1-cos(wc)); C = 2*(cos(wc)-1)
50     func = lambda alpha: alpha**2 + B*alpha + C
51     deriv = lambda alpha: 2*alpha + B
52     alpha_0 = 0.5
53     return round(newton(func, alpha_0, deriv),3)
54 
55 # Filtro butterworth en funcion de la frecuencia de corte
56 def butterworth(x,wc):
57     N = 1
58     B,A = butter(N,wc)
59     y = []
60 
61     for n in range(len(x)):
62         if n == 0:
63             y.append(x[0])
64         else:
65             y_n = B[0]*(x[n] + x[n-1]) - A[1]*y[n-1]
66             y.append(y_n)
67     return y
68 
69 # Filtro supersmoother en funcion de la frecuencia de corte
70 def smooth(x,wc):
71 
72     a = np.exp(-np.sqrt(2)*np.pi*wc/2)
73     b = 2*a*np.cos(np.sqrt(2)*np.pi*wc/2)
74 
75     c2 = b
76     c3 = -a*a
77     c1 = 1-c2-c3
78 
79     y = []
80 
81     for n in range(len(x)):
82         if n == 0:
83             y.append(x[0])
84         else:
85             y_n = (c1/2)*(x[n] + x[n-1]) + c2*y[n-1] + c3*y[n
                -2]
86             y.append(y_n)
87     return y
88 
89 # Filtro media movil exponencial en funcion de la frecuencia de
     corte
90 def ema(x, wc):
91     alpha = alpha_ema(wc)
92     y = []
93 
94     for n in range(len(x)):
95         if n == 0:
96             y.append(x[0])
97         else:
98             y_n = alpha*x[n] + (1-alpha)*y[n-1]
99             y.append(y_n)
100     return y
```

```
101
102 # Metodo de Newton para obtener el perido de la DMA a partir de
        la frecuencia de corte
103 def N_dma(wc):
104     wc = wc*pi
105     func = lambda N: sin(wc*N/2) - (N/(2**(1/4)))*(sin(wc/2))
106     deriv = lambda N: (wc/2)*cos(wc*N/2) - (1/(2**(1/4)))*sin(
            wc/2)
107     N_0 = pi/wc
108     return int(np.round(newton(func, N_0, deriv)))
109
110 # Metodo de Newton para obtener la frecuencia de corte de la
        DMA a partir del periodo
111 def wc_dma(N):
112     func = lambda wc: sin(N*wc/2) - (N/(2**(1/4))) * sin(wc/2)
113     deriv = lambda wc: cos(N*wc/2) * N/2 - (N/(2**(1/4))) * cos
            (wc/2) / 2
114     omega_0 = pi/N
115     return newton(func, omega_0, deriv)/pi
116
117 def dma(x, wc):
118     N = N_dma(wc)
119     h = (1/N)*np.ones(N,)
120     B = np.convolve(h, h, mode='full')
121
122     y = []
123     for n in range(len(x)):
124         if n < len(B):
125             y.append(np.NAN)
126         else:
127             y_n = np.dot(B,x[n-len(B):n])
128             y.append(y_n)
129
130     return y
```

B Código Backtesting, Optimización y Validación

```
1 #---------------------------------------------
2 # Tratamiento Digital de Senales Financieras
3 # Pablo Marchesi
4 # Junio 2024
5 #---------------------------------------------
6 from backtesting import Strategy, Backtest
7 from backtesting.lib import crossover
8 import yfinance as yf
9 import pandas as pd
10 import plotly.express as px
11 import numpy as np
12 from filt_funcs import sma, ema, butterworth, smooth, dma,
       wc_sma
13
14 # %% [markdown]
15 # ## Inicializaci n
16
```

```
# %%
#  Se al
x = yf.download('MSFT', start = '2010-01-01', progress = False)

# Parametros del backtesting
iCash = 10000; com = 0; r = 0.055

# Parametros iniciales filtros
wc1_0 = wc_sma(20); wc2_0 = wc_sma(100)

# %% [markdown]
# ## Definici n de las Estrategias

# %%
# Estrategia SMA
class sma_strat(Strategy):

    # Parametros iniciales
    wc1 = wc1_0; wc2 = wc2_0

    def init(self):
        # Calcula los indicadores
        self.filt1 = self.I(sma, self.data.Close, self.wc1)
        self.filt2 = self.I(sma, self.data.Close, self.wc2)

    def next(self):
        # Cierra el corto, abre un largo
        if crossover(self.filt1, self.filt2):
            self.position.close()
            self.buy()

        # Cierra el largo, abre un corto
        elif crossover(self.filt2, self.filt1):
            self.position.close()
            self.sell()

# Estrategia EMA
class ema_strat(Strategy):

    # Parametros iniciales
    wc1 = wc1_0; wc2 = wc2_0

    def init(self):
        # Calcula los indicadores
        self.filt1 = self.I(ema, self.data.Close, self.wc1)
        self.filt2 = self.I(ema, self.data.Close, self.wc2)

    def next(self):
        # Cierra el corto, abre un largo
        if crossover(self.filt1, self.filt2):
            self.position.close()
            self.buy()

        # Cierra el largo, abre un corto
        elif crossover(self.filt2, self.filt1):
            self.position.close()
```

```
73              self.sell()
74
75  # Estrategia Butterworth
76  class butter_strat(Strategy):
77
78      # Parametros iniciales
79      wc1 = wc1_0; wc2 = wc2_0
80
81      def init(self):
82          # Calcula los indicadores
83          self.filt1 = self.I(butterworth, self.data.Close, self.
                wc1)
84          self.filt2 = self.I(butterworth, self.data.Close, self.
                wc2)
85
86      def next(self):
87          # Cierra el corto, abre un largo
88          if crossover(self.filt1, self.filt2):
89              self.position.close()
90              self.buy()
91
92          # Cierra el largo, abre un corto
93          elif crossover(self.filt2, self.filt1):
94              self.position.close()
95              self.sell()
96
97  # Estrategia SuperSmoother
98  class smooth_strat(Strategy):
99
100     # Parametros iniciales
101     wc1 = wc1_0; wc2 = wc2_0
102
103     def init(self):
104         # Calcula los indicadores
105         self.filt1 = self.I(smooth, self.data.Close, self.wc1)
106         self.filt2 = self.I(smooth, self.data.Close, self.wc2)
107
108     def next(self):
109         # Cierra el corto, abre un largo
110         if crossover(self.filt1, self.filt2):
111             self.position.close()
112             self.buy()
113
114         # Cierra el largo, abre un corto
115         elif crossover(self.filt2, self.filt1):
116             self.position.close()
117             self.sell()
118
119 # Estrategia DMA
120 class dma_strat(Strategy):
121
122     # Parametros iniciales
123     wc1 = wc1_0; wc2 = wc2_0
124
125     def init(self):
126         # Calcula los indicadores
```

```
127         self.filt1 = self.I(dma, self.data.Close, self.wc1)
128         self.filt2 = self.I(dma, self.data.Close, self.wc2)
129 
130     def next(self):
131         # Cierra el corto, abre un largo
132         if crossover(self.filt1, self.filt2):
133             self.position.close()
134             self.buy()
135 
136         # Cierra el largo, abre un corto
137         elif crossover(self.filt2, self.filt1):
138             self.position.close()
139             self.sell()
140 
141 # %% [markdown]
142 # ## Ejecuci n del Backtesting y Estad sticas
143 
144 # %%
145 # Ejecucion de los backtestings
146 bt_sma = Backtest(x, sma_strat, cash = iCash, commission = com)
147 bt_ema = Backtest(x, ema_strat, cash = iCash, commission = com)
148 bt_butter = Backtest(x, butter_strat, cash = iCash, commission
    = com)
149 bt_smooth = Backtest(x, smooth_strat, cash = iCash, commission
    = com)
150 bt_dma = Backtest(x, dma_strat, cash = iCash, commission = com)
151 
152 # Estadisticas de los backtestings
153 stats_sma = bt_sma.run()
154 stats_ema = bt_ema.run()
155 stats_butter = bt_butter.run()
156 stats_smooth = bt_smooth.run()
157 stats_dma = bt_dma.run()
158 
159 stats = {'SMA':stats_sma, 'EMA':stats_ema, 'Butterworth':
    stats_butter, 'SuperSmoother':stats_smooth, 'DMA': stats_dma
    }
160 
161 # Parametros relevantes
162 mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
    maxDD = []; rets = []; retsDD = [];
163 
164 for name, stat in stats.items():
165     mu.append(stat['Return (Ann.) [%]'])
166     sigma.append(stat['Volatility (Ann.) [%]'])
167     ntrades.append(stat['# Trades'])
168     ytrades.append(365*stat['# Trades']/len(x))
169     sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
        Volatility (Ann.) [%]'])
170     maxDD.append(stat['Max. Drawdown [%]'])
171     rets.append(stat['Return [%]'])
172     retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown
        [%]']*len(x)))
173 
174 res = pd.DataFrame({'Estretegia':stats.keys(), 'Retornos (%)':
    rets,'Retornos Anualiz. (%)':mu, 'Volatilidad Anualiz. (%)':
```

```
    sigma,
175                     'Trades':ntrades, 'Trades/A o':ytrades, '
                            Ratio Sharpe':sharpe, 'Max Drawdown (%)'
                            : maxDD,
176                     'Retornos/Drawdown': retsDD,'wc1':wc1_0, '
                            wc2': wc2_0})
177 # Curvas de equity
178 df_equity = pd.DataFrame({name: stats['_equity_curve']['Equity'
    ] for name, stats in stats.items()})
179
180 # Curvas de rendimiento en %
181 df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
    iCash - 1 for name, stats in stats.items()})
182
183 # Curvas de drawdown
184 df_DD = pd.DataFrame({name: stats['_equity_curve']['DrawdownPct
    '] for name, stats in stats.items()})
185
186 # %% [markdown]
187 # ### Resultados Iniciales
188
189 # %%
190 # Resumen de las estadisticas de los backtestings
191 res
192
193 # %%
194 # Plot curvas equity
195 fig = px.line(df_equity, title = 'Rendimiento de cada
    estrategia (en $)')
196 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
197 fig.show()
198
199 # %%
200 # Plot curvas rendimiento en %
201 fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (
    en %)')
202 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
203 fig.update_yaxes(tickformat=".2%")
204 fig.show()
205
206 # %%
207 # Plot del drawdown
208 fig = px.line(-df_DD, title = 'Drawdown de cada estrategia')
209 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
210 fig.update_yaxes(tickformat=".2%")
211 fig.show()
212
213 # %% [markdown]
214 # ## Optimizaci n
215
216 # %%
217 # Parametros de la optimizacion (metodo exahustivo)
218 wc_range = list(np.arange(wc_sma(200), wc_sma(10), 0.0025))
219 iter = len(wc_range)**2; print(f'Calculando {iter}
    combinaciones, en el rango [{round(wc_sma(200), 4)}, {round(
    wc_sma(10), 4)}]')
```

```
220 goal = 'Sharpe Ratio'
221 cond = lambda param: param.wc1 > param.wc2
222 method = 'grid'
223
224 # Optimizaciones
225 stats_sma_opt, heatmap_sma = bt_sma.optimize(wc1 = wc_range,
        wc2 = wc_range, maximize = goal, constraint = cond, method =
         method, return_heatmap=True)
226 stats_ema_opt, heatmap_ema = bt_ema.optimize(wc1 = wc_range,
        wc2 = wc_range, maximize = goal, constraint = cond, method =
         method, return_heatmap=True)
227 stats_butter_opt, heatmap_butter = bt_butter.optimize(wc1 =
        wc_range, wc2 = wc_range, maximize = goal, constraint = cond
        , method = method, return_heatmap=True)
228 stats_smooth_opt, heatmap_smooth = bt_smooth.optimize(wc1 =
        wc_range, wc2 = wc_range, maximize = goal, constraint = cond
        , method = method, return_heatmap=True)
229 stats_dma_opt, heatmap_dma = bt_dma.optimize(wc1 = wc_range,
        wc2 = wc_range, maximize = goal, constraint = cond, method =
         method, return_heatmap=True)
230
231 # Estadisticas tras optimizar
232 stats_opt = {'SMA':stats_sma_opt,'EMA':stats_ema_opt,'
        Butterworth':stats_butter_opt,
233              'SuperSmoother':stats_smooth_opt, 'DMA':
                    stats_dma_opt}
234
235 heatmaps = {'SMA':heatmap_sma,'EMA':heatmap_ema,'Butterworth':
        heatmap_butter,
236              'SuperSmoother':heatmap_smooth, 'DMA': heatmap_dma}
237
238 # Parametros relevantes
239 mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
        maxDD = []; rets = []; retsDD = []; wc1 = []; wc2 = []
240
241 for name, stat in stats_opt.items():
242     mu.append(stat['Return (Ann.) [%]'])
243     sigma.append(stat['Volatility (Ann.) [%]'])
244     ntrades.append(stat['# Trades'])
245     ytrades.append(365*stat['# Trades']/len(x))
246     sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
            Volatility (Ann.) [%]'])
247     maxDD.append(stat['Max. Drawdown [%]'])
248     rets.append(stat['Return [%]'])
249     retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown
            [%]']*len(x)))
250
251 for name, heatmap in heatmaps.items():
252     wcs = heatmap.sort_values(ascending = False).iloc[0:1].
            index[0]
253     wc1.append(wcs[0]); wc2.append(wcs[1])
254
255 df_params = pd.DataFrame({'Estrategia':stats_opt.keys(), 'wc1':
        wc1, 'wc2':wc2})
256
```

```
257 res_opt = pd.DataFrame({'Estretegia':stats_opt.keys(), '
    Retornos (%)': rets,'Retornos Anualiz. (%)':mu, 'Volatilidad
     Anualiz. (%)':sigma,
258                         'Trades':ntrades, 'Trades/A o':ytrades
                               , 'Ratio Sharpe':sharpe, 'Max
                               Drawdown (%)': maxDD, 'Retornos/
                               Drawdown': retsDD})
259
260 # Curvas de equity
261 df_equity = pd.DataFrame({name: stats['_equity_curve']['Equity'
    ] for name, stats in stats_opt.items()})
262
263 # Curvas de rendimiento en %
264 df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
    iCash - 1 for name, stats in stats_opt.items()})
265
266 # Curvas de drawdown
267 df_DD = pd.DataFrame({name: stats['_equity_curve']['DrawdownPct
    '] for name, stats in stats_opt.items()})
268
269 # %%
270 # Par metros optimos de la estrategia
271 df_params
272
273 # %%
274 # Estad sticas del backtesting optimizado
275 res_opt
276
277 # %%
278 df = pd.DataFrame(heatmap_sma).reset_index()
279 fig = px.density_heatmap(df, x='wc1', y='wc2', z='Sharpe Ratio'
    ,
280                          color_continuous_scale = 'Viridis',
                               histfunc = 'avg',
281                          title = 'Optimizaci n de la
                               Estrategia SMA', nbinsx=35, nbinsy
                               = 5)
282
283 fig.show()
284
285 # %%
286 # Plot curvas equity
287 fig = px.line(df_equity, title = 'Rendimiento de cada
    estrategia (en $)')
288 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
289 fig.show()
290
291 # %%
292 # Plot curvas rendimiento en %
293 fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (
    en %)')
294 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
295 fig.update_yaxes(tickformat=".2%")
296 fig.show()
297
298 # %%
```

```
299 # Plot del drawdown
300 fig = px.line(-df_DD, title = 'Drawdown de cada estrategia')
301 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
302 fig.update_yaxes(tickformat=".2%")
303 fig.show()
304
305 # %%
306 # Parametros de la optimizacion (metodo bayesiano)
307 method = 'skopt'
308
309 # Optimizaciones
310 stats_sma_opt, heatmap_sma = bt_sma.optimize(wc1 = wc_range,
        wc2 = wc_range, maximize = goal, constraint = cond, method =
         method, return_heatmap=True)
311 stats_ema_opt, heatmap_ema = bt_ema.optimize(wc1 = wc_range,
        wc2 = wc_range, maximize = goal, constraint = cond, method =
         method, return_heatmap=True)
312 stats_butter_opt, heatmap_butter = bt_butter.optimize(wc1 =
        wc_range, wc2 = wc_range, maximize = goal, constraint = cond
        , method = method, return_heatmap=True)
313 stats_smooth_opt, heatmap_smooth = bt_smooth.optimize(wc1 =
        wc_range, wc2 = wc_range, maximize = goal, constraint = cond
        , method = method, return_heatmap=True)
314 stats_dma_opt, heatmap_dma = bt_dma.optimize(wc1 = wc_range,
        wc2 = wc_range, maximize = goal, constraint = cond, method =
         method, return_heatmap=True)
315
316 # Estadisticas tras optimizar
317 stats_opt = {'SMA':stats_sma_opt,'EMA':stats_ema_opt,'
        Butterworth':stats_butter_opt,
318              'SuperSmoother':stats_smooth_opt, 'DMA':
                    stats_dma_opt}
319
320 heatmaps = {'SMA':heatmap_sma,'EMA':heatmap_ema,'Butterworth':
        heatmap_butter,
321              'SuperSmoother':heatmap_smooth, 'DMA': heatmap_dma}
322
323 # Parametros relevantes
324 mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
        maxDD = []; rets = []; retsDD = []; wc1 = []; wc2 = []
325
326 for name, stat in stats_opt.items():
327     mu.append(stat['Return (Ann.) [%]'])
328     sigma.append(stat['Volatility (Ann.) [%]'])
329     ntrades.append(stat['# Trades'])
330     ytrades.append(365*stat['# Trades']/len(x))
331     sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
            Volatility (Ann.) [%]'])
332     maxDD.append(stat['Max. Drawdown [%]'])
333     rets.append(stat['Return [%]'])
334     retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown
            [%]']*len(x)))
335
336 for name, heatmap in heatmaps.items():
337     wcs = heatmap.sort_values(ascending = False).iloc[0:1].
            index[0]
```

```
338     wc1.append(wcs[0]); wc2.append(wcs[1])
339
340 df_params = pd.DataFrame({'Estrategia':stats_opt.keys(), 'wc1':
        wc1, 'wc2':wc2})
341
342 res_opt_2 = pd.DataFrame({'Estretegia':stats_opt.keys(), '
        Retornos (%)': rets,'Retornos Anualiz. (%)':mu, 'Volatilidad
         Anualiz. (%)':sigma,
343                           'Trades':ntrades, 'Trades/A o':ytrades
                                , 'Ratio Sharpe':sharpe, 'Max
                               Drawdown (%)': maxDD,
344                           'Retornos/Drawdown': retsDD})
345
346 # Curvas de equity
347 df_equity = pd.DataFrame({name: stats['_equity_curve']['Equity'
        ] for name, stats in stats_opt.items()})
348
349 # Curvas de rendimiento en %
350 df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
        iCash - 1 for name, stats in stats_opt.items()})
351
352 # Curvas de drawdown
353 df_DD = pd.DataFrame({name: stats['_equity_curve']['DrawdownPct
        '] for name, stats in stats_opt.items()})
354
355 # %%
356 df = pd.DataFrame(heatmap_sma).reset_index()
357 fig = px.density_heatmap(df, x='wc1', y='wc2', z='Sharpe Ratio'
        ,
358                          color_continuous_scale = 'Viridis',
                              histfunc = 'avg',
359                          title = 'Optimizaci n de la
                              Estrategia SMA', nbinsx=35, nbinsy
                              = 5)
360
361 fig.show()
362
363 # %%
364 # Parametros optimizados
365 df_params
366
367 # %%
368 # Estadisticas del backtesting optimizado
369 res_opt_2
370
371 # %%
372 # Plot curvas equity
373 fig = px.line(df_equity, title = 'Rendimiento de cada
        estrategia (en $)')
374 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
375 fig.show()
376
377 # %%
378 # Plot curvas rendimiento en %
379 fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (
        en %)')
```

```
fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
fig.update_yaxes(tickformat=".2%")
fig.show()

# %%
# Plot del drawdown
fig = px.line(-df_DD, title = 'Drawdown de cada estrategia')
fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
fig.update_yaxes(tickformat=".2%")
fig.show()

# %% [markdown]
# ## Validaci n del Sistema
# - Analisis de sensibilidad
# - Analisis en otros mercados
# - Analisis IS/OOS
# - Analisis con comisiones
# - Analisis slippage extremo
# - Analisis Monte Carlo
# - Analisis de sensibilidad
#

# %% [markdown]
# ### Analisis IS

# %%
# Se al
x = yf.download('MSFT', start = '2010-01-01', progress = False)

# Parametros del backtesting
iCash = 10000; com = 0; r = 0.055

# Parametros iniciales filtros
wc1_0 = df_params.loc[3]['wc1']; wc2_0 = df_params.loc[3]['wc2'
    ]


# %%
# Estrategia SuperSmoother
class smooth_strat(Strategy):

    # Parametros iniciales
    wc1 = wc1_0; wc2 = wc2_0

    def init(self):
        # Calcula los indicadores
        self.filt1 = self.I(smooth, self.data.Close, self.wc1)
        self.filt2 = self.I(smooth, self.data.Close, self.wc2)

    def next(self):
        # Cierra el corto, abre un largo
        if crossover(self.filt1, self.filt2):
            self.position.close()
            self.buy()

        # Cierra el largo, abre un corto
```

```
435         elif crossover(self.filt2, self.filt1):
436             self.position.close()
437             self.sell()
438 
439 bt_smooth_IS = Backtest(x, smooth_strat, cash = iCash,
        commission = com)
440 stats_IS = bt_smooth_IS.run()
441 
442 # %% [markdown]
443 # ### Analisis en otros mercados
444 
445 # %%
446 tickers = ['QQQ', 'SPY', 'SAN', 'GLD', 'BP', 'AMZN']
447 
448 stats_tickers = {}
449 
450 for ticker in tickers:
451     x = yf.download(ticker, start = '2010-01-01', progress =
            False)
452     bt_i = Backtest(x, smooth_strat, cash = iCash, commission =
            com)
453     stats_i = bt_i.run()
454     stats_tickers[ticker] = stats_i
455 
456     # Parametros relevantes
457     mu = []; sigma = []; ntrades = []; ytrades = []; sharpe =
            []; maxDD = []; rets = []; retsDD = [];
458 
459     for name, stat in stats_tickers.items():
460         mu.append(stat['Return (Ann.) [%]'])
461         sigma.append(stat['Volatility (Ann.) [%]'])
462         ntrades.append(stat['# Trades'])
463         ytrades.append(365*stat['# Trades']/len(x))
464         sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
                Volatility (Ann.) [%]'])
465         maxDD.append(stat['Max. Drawdown [%]'])
466         rets.append(stat['Return [%]'])
467         retsDD.append(-365*stat['Return [%]']/(stat['Max.
                Drawdown [%]']*len(x)))
468 
469 res_tickers = pd.DataFrame({'Estretegia':stats_tickers.keys(),
        'Retornos (%)': rets,'Retornos Anualiz. (%)':mu, '
        Volatilidad Anualiz. (%)':sigma,
470                             'Trades':ntrades, 'Trades/A o':ytrades
                                , 'Ratio Sharpe':sharpe, 'Max
                                Drawdown (%)': maxDD, 'Retornos/
                                Drawdown': retsDD})
471 
472 # Curvas de rendimiento en %
473 df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
        iCash - 1 for name, stats in stats_tickers.items()})
474 
475 # %%
476 # Resultados otros mercados
477 res_tickers
478 
```

```
# %%
# Plot curvas rendimiento en %
fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (en %)')
fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
fig.update_yaxes(tickformat=".2%")
fig.show()

# %% [markdown]
# ### Analisis con comisiones

# %%
com = 0.01
x = yf.download('MSFT', start = '2010-01-01', progress = False)
bt_smooth_com = Backtest(x, smooth_strat, cash = iCash, commission = com)
stats_com = bt_smooth_com.run()

stats = {'Sin Comision':stats_IS, 'Con Comision':stats_com}

# Parametros relevantes
mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = []; maxDD = []; rets = []; retsDD = [];

for name, stat in stats.items():
    mu.append(stat['Return (Ann.) [%]'])
    sigma.append(stat['Volatility (Ann.) [%]'])
    ntrades.append(stat['# Trades'])
    ytrades.append(365*stat['# Trades']/len(x))
    sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['Volatility (Ann.) [%]'])
    maxDD.append(stat['Max. Drawdown [%]'])
    rets.append(stat['Return [%]'])
    retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown [%]']*len(x)))

res_com = pd.DataFrame({'Estretegia':stats.keys(), 'Retornos (%)': rets,'Retornos Anualiz. (%)':mu, 'Volatilidad Anualiz. (%)':sigma,
                        'Trades':ntrades, 'Trades/A o':ytrades, 'Ratio Sharpe':sharpe, 'Max Drawdown (%)': maxDD, 'Retornos/Drawdown': retsDD})

# Curvas de rendimiento en %
df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/iCash - 1 for name, stats in stats.items()})

# %%
# Comparacion con comisiones
res_com

# %%
# Plot curvas rendimiento en %
fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (en %)')
```

```
fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
fig.update_yaxes(tickformat=".2%")
fig.show()

# %% [markdown]
# ### Analisis de sensibilidad

# %%
from numpy.random import normal

# Parametros iniciales filtros
wc1_range = wc1_0*(1+normal(0,.1,10))
wc2_range = wc2_0*(1+normal(0,.1,10))
stats_sens = {}

for i in range(len(wc1_range)):
    class smooth_strat(Strategy):

        # Parametros iniciales
        wc1 = wc1_range[i]; wc2 = wc2_range[i]

        def init(self):
            # Calcula los indicadores
            self.filt1 = self.I(smooth, self.data.Close, self.
                wc1)
            self.filt2 = self.I(smooth, self.data.Close, self.
                wc2)

        def next(self):
            # Cierra el corto, abre un largo
            if crossover(self.filt1, self.filt2):
                self.position.close()
                self.buy()

            # Cierra el largo, abre un corto
            elif crossover(self.filt2, self.filt1):
                self.position.close()
                self.sell()

    bt_i = Backtest(x, smooth_strat, cash = iCash, commission =
        com)
    stats_i = bt_i.run()
    stats_sens[f'Iteracion {i+1}'] = stats_i

# Parametros relevantes
mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
    maxDD = []; rets = []; retsDD = [];

for name, stat in stats_sens.items():
    mu.append(stat['Return (Ann.) [%]'])
    sigma.append(stat['Volatility (Ann.) [%]'])
    ntrades.append(stat['# Trades'])
    ytrades.append(365*stat['# Trades']/len(x))
    sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
        Volatility (Ann.) [%]'])
    maxDD.append(stat['Max. Drawdown [%]'])
```

```
574     rets.append(stat['Return [%]'])
575     retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown
            [%]']*len(x)))
576
577 res_sens = pd.DataFrame({'Estretegia':stats_sens.keys(), '
        Retornos (%)': rets,'Retornos Anualiz. (%)':mu, 'Volatilidad
         Anualiz. (%)':sigma,
578                           'Trades':ntrades, 'Trades/A o':ytrades
                                , 'Ratio Sharpe':sharpe, 'Max
                                Drawdown (%)': maxDD, 'Retornos/
                                Drawdown': retsDD})
579
580 df_wc = pd.DataFrame({'wc1/wc1_0':(wc1/wc1_0-1), 'wc2/wc2_0':
        wc2/wc2_0-1, 'Retornos Anualiz. (%)':mu})
581
582 # Curvas de rendimiento en %
583 df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
        iCash - 1 for name, stats in stats_sens.items()})
584
585 # %%
586 # Variaci n de parametros
587 df_wc
588
589 # %%
590 fig = px.scatter(df_wc, x='wc1/wc1_0', y='wc2/wc2_0', color = '
        Retornos Anualiz. (%)', title='Retornos tras variar
        par metros',
591                  labels={'wc1/wc1_0': 'Variaci n wc1', 'wc2/
                      wc2_0': 'Variaci n wc2'}, size='Retornos
                      Anualiz. (%)', size_max=25)
592
593 fig.update_yaxes(tickformat=".2%"); fig.update_xaxes(tickformat
        =".2%")
594
595 fig.show()
596
597 # %%
598 # Resultados tras la variacion de parametros
599 res_sens
600
601 # %%
602 # Plot curvas rendimiento en %
603 fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (
        en %)')
604 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
605 fig.update_yaxes(tickformat=".2%")
606 fig.show()
607
608 # %% [markdown]
609 # ### Analisis OOS
610
611 # %%
612 # Analisis IS/OOS
613 x = yf.download('MSFT', start = '2000-01-01', end = '2010-01-01
        ', progress = False)
```

```
614 bt_smooth_OOS = Backtest(x, smooth_strat, cash = iCash,
        commission = com)
615 stats_OOS = bt_smooth_OOS.run()
616
617 stats = {'IS':stats_IS, 'OOS':stats_OOS}
618
619 # Parametros relevantes
620 mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
        maxDD = []; rets = []; retsDD = [];
621
622 for name, stat in stats.items():
623     mu.append(stat['Return (Ann.) [%]'])
624     sigma.append(stat['Volatility (Ann.) [%]'])
625     ntrades.append(stat['# Trades'])
626     ytrades.append(365*stat['# Trades']/len(x))
627     sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
            Volatility (Ann.) [%]'])
628     maxDD.append(stat['Max. Drawdown [%]'])
629     rets.append(stat['Return [%]'])
630     retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown
            [%]']*len(x)))
631
632 res_OOS = pd.DataFrame({'Estretegia':stats.keys(), 'Retornos
        (%)': rets,'Retornos Anualiz. (%)':mu, 'Volatilidad Anualiz.
         (%)':sigma,
633                         'Trades':ntrades, 'Trades/A o':ytrades
                            , 'Ratio Sharpe':sharpe, 'Max
                            Drawdown (%)': maxDD, 'Retornos/
                            Drawdown': retsDD})
634
635
636 # %%
637 # Gr fico
638 fig = px.line(x['Close'], title = 'Cotizaci n de MSFT')
639 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '
        Cotizaci n ($)', showlegend = False)
640
641 fig.show()
642
643 # %%
644 # Resultados OOS
645 res_OOS
646
647 # %%
648 # Plot curvas rendimiento en %
649 fig = px.line(stats_OOS['_equity_curve']['Equity']/iCash - 1 ,
        title = 'Rendimiento de la estrategia en datos OOS (en %)')
650 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '',
        showlegend = False)
651 fig.update_yaxes(tickformat=".2%")
652 fig.show()
653
654 # %% [markdown]
655 # ### Analisis de slippage extremo
656
657 # %%
```

```
658 def pos_norm(n, loc, scale):
659     pos_values = []
660 
661     while len(pos_values) < n:
662         # Generar valores de una distribuci n normal
663         values = np.random.normal(loc, scale, size=n)
664 
665         # Filtrar solo los valores positivos
666         pos_values.extend(values[values > 0])
667 
668     pos_values = pos_values[:n]
669 
670     return pos_values
671 
672 # %%
673 import math
674 # Media geometrica para sacar el slippage medio por trade
675 def geo_mean(list):
676 
677     prod = 1
678     for num in list:
679         prod *= num
680 
681     return math.pow(prod, 1 / len(list))
682 
683 # %%
684 # Simulamos con una gaussiana un escenario de slippage extremo
685 df_trades = stats_IS['_trades']
686 slippage = pos_norm(len(df_trades), 0, 0.015)
687 slippage = [x + 1 for x in slippage]
688 slippage_mean = geo_mean(slippage)-1
689 slippage_mean
690 
691 # %%
692 com = slippage_mean
693 x = yf.download('MSFT', start = '2010-01-01', progress = False)
694 bt_smooth_slip = Backtest(x, smooth_strat, cash = iCash,
        commission = com)
695 stats_slip = bt_smooth_slip.run()
696 
697 stats = {'Sin Slippage':stats_IS, 'Con Slippage':stats_slip}
698 
699 # Parametros relevantes
700 mu = []; sigma = []; ntrades = []; ytrades = []; sharpe = [];
      maxDD = []; rets = []; retsDD = [];
701 
702 for name, stat in stats.items():
703     mu.append(stat['Return (Ann.) [%]'])
704     sigma.append(stat['Volatility (Ann.) [%]'])
705     ntrades.append(stat['# Trades'])
706     ytrades.append(365*stat['# Trades']/len(x))
707     sharpe.append((stat['Return (Ann.) [%]']-r*100)/stat['
          Volatility (Ann.) [%]'])
708     maxDD.append(stat['Max. Drawdown [%]'])
709     rets.append(stat['Return [%]'])
```

```
710     retsDD.append(-365*stat['Return [%]']/(stat['Max. Drawdown
            [%]']*len(x)))
711
712 res_slip = pd.DataFrame({'Estretegia':stats.keys(), 'Retornos
        (%)': rets,'Retornos Anualiz. (%)':mu, 'Volatilidad Anualiz.
         (%)':sigma,
713                            'Trades':ntrades, 'Trades/A o':ytrades
                                , 'Ratio Sharpe':sharpe, 'Max
                                Drawdown (%)': maxDD, 'Retornos/
                                Drawdown': retsDD})
714
715 # Curvas de rendimiento en %
716 df_ROI = pd.DataFrame({name: stats['_equity_curve']['Equity']/
        iCash - 1 for name, stats in stats.items()})
717
718 # %%
719 # Comparativa con y sin slippage
720 res_slip
721
722 # %%
723 # Plot curvas rendimiento en %
724 fig = px.line(df_ROI, title = 'Rendimiento de cada estrategia (
        en %)')
725 fig.update_layout(xaxis_title = 'Fecha', yaxis_title = '')
726 fig.update_yaxes(tickformat=".2%")
727 fig.show()
728
729 # %% [markdown]
730 # ### Analisis de dependencia
731
732 # %%
733 # Analizamos la rentabilidad por trade
734 stats_IS = bt_smooth_IS.run()
735 df_trades = stats_IS['_trades']
736 fig = px.bar(df_trades['ReturnPct'], title = 'Retornos por
        trade')
737 fig.update_layout(xaxis_title = 'Trades', yaxis_title = '
        Retorno (%)', showlegend = False)
738
739
740 # %%
741 # Distribucion de los retornos
742 med = np.median(df_trades['ReturnPct'])
743 mean = np.mean(df_trades['ReturnPct'])
744 std = np.std(df_trades['ReturnPct'])
745 print(med, mean, std)
746 fig = px.histogram(df_trades['ReturnPct'], nbins=25, title = '
        Distribuci n de los retornos')
747 fig.update_layout(xaxis_title = 'Retornos', yaxis_title = '
        Frecuencia', showlegend = False)
748
749 # %% [markdown]
750 # ### Analisis del drowdawn m ximo
751
752 # %%
753 from numpy.random import shuffle
```

```
754 import plotly.graph_objects as go
755
756 stats_IS = bt_smooth_IS.run()
757 df_trades = stats_IS['_trades']
758 trades = df_trades['PnL']
759 iter = 300
760 fig = go.Figure()
761
762
763 for i in range(iter):
764
765     shuffled_trades = trades.copy()
766     shuffle(shuffled_trades)
767     shuffled_trades = pd.concat([pd.Series([0]),
            shuffled_trades], ignore_index=True)
768     sum = np.cumsum(shuffled_trades)/iCash
769     fig.add_trace(go.Scatter(x = np.arange(1, len(df_trades)+2)
            , y=sum, mode='lines'))
770
771
772
773 fig.update_layout(title='An lisis del Drawdown', xaxis_title='
        N mero de Trades',yaxis_title='Rentabilidad', showlegend =
        False)
774 fig.update_yaxes(tickformat=".2%")
775 fig.update_yaxes(range=[-1, 5])
776
777 fig.show()
```

ANEXO I. RELACIÓN DEL TRABAJO CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE DE LA AGENDA 2030

Anexo al Trabajo de Fin de Grado y Trabajo de Fin de Máster: Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030.

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				*
ODS 2. **Hambre cero.**				*
ODS 3. **Salud y bienestar.**				*
ODS 4. **Educación de calidad.**	*			
ODS 5. **Igualdad de género.**				*
ODS 6. **Agua limpia y saneamiento.**				*
ODS 7. **Energía asequible y no contaminante.**				*
ODS 8. **Trabajo decente y crecimiento económico.**	*			
ODS 9. **Industria, innovación e infraestructuras.**	*			
ODS 10. **Reducción de las desigualdades**.				*
ODS 11. **Ciudades y comunidades sostenibles**.				*
ODS 12. **Producción y consumo responsables.**				*
ODS 13. **Acción por el clima.**				*
ODS 14. **Vida submarina.**				*
ODS 15. **Vida de ecosistemas terrestres.**				*
ODS 16. **Paz, justicia e instituciones sólidas.**				*
ODS 17. **Alianzas para lograr objetivos.**				*

Descripción de la alineación del TFG/TFM con los ODS con un grado de relación más alto.

***Utilice tantas páginas como sea necesario.

El TFG se alinea principalmente con el objetivo de crecimiento económico, ya que contribuye a mejorar la eficiencia de los mercados financieros, ya que el trading algorítmico es una actividad que proporciona liquidez a estos.

Editor: Secretaria General / UPV · D.L.: V-5092-2006 · ISSN: 1887-2298 · www.upv.es/secgen

Anexo al Trabajo de Fin de Grado y Trabajo de Fin de Máster: Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030. (Numere la página)

El TFG se alinea principalmente con el objetivo de crecimiento económico, ya que contribuye a mejorar la eficiencia de los mercados financieros, ya que el trading algorítmico es una actividad que proporciona liquidez a estos.

Editor: Secretaria General / UPV · D.L.: V-5092-2006 · ISSN: 1887-2298 · www.upv.es/secgen