Tratamiento Digital de Señales Financieras

Pablo Marchesi
Junio 2024

Abstract

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.

"Algebra is like sheet of music. The important thing isn't can you read music, it's can you hear it. Can you hear the music, Robert?"

— Niels Bohr to J. Robert Oppenheimer, *Oppenheimer* (2023)

Índice

1. Introducción **4**

2. Conceptos Básicos del Tratamiento Digital de Señales **5**
2.1. Filtro digital 5
2.2. Respuesta al Impulso 5
2.3. Respuesta en Frecuencia 6
2.4. Función de Transferencia 6
2.5. Frecuencia de Corte 7

3. Filtros digitales **8**
3.1. SMA: Simple Moving Average 8
3.2. EMA: Exponential Moving Average 10
3.3. DMA: Double Moving Average 12
3.4. Butterworth 15
3.5. SuperSmoother 16

4. Trading Algorítmico **19**
4.1. Desarrollo de una Estrategia 19
4.2. Implementación en Python 20
4.3. Backtesting 20
4.4. Métricas clave 22
4.5. Resultados iniciales 24

5. Optimización **25**
5.1. Consideraciones Previas 25
5.2. Método Exhaustivo 26
5.3. Método Bayesiano 26
5.4. Resultados 26

6. Validación del sistema **27**
6.1. Análisis de sensibilidad 27
6.2. IS vs OOS 27
6.3. Análisis de Dependencia 27
6.4. Método Monte Carlo 27
6.5. Simulación con Comisiones 27
6.6. Simulación de Slippage Extremo 27
6.7. Simulación en Otros Mercados 27

7. Conclusiones **28**

Índice de figuras

1. Relación entre la señal de entrada $x[n]$ y la señal de salida $y[n]$ a través de un filtro digital. 5
2. Relación entre la señal de entrada $x[n]$ y la señal de salida $y[n]$ a través de la respuesta al impulso $h[n]$. 5
3. Filtro paso-bajo con $\omega_c = 0.5$ 8
4. Suavizado con una SMA de 20 periodos 8
5. Respuesta en frecuencia de una SMA de 20 periodos 9
6. Filtrado con una EMA donde $\alpha = 0.13$ 11
7. Respuesta en frecuencia de una EMA con $\alpha = 0.13$ 11
8. Filtrado con una DMA de periodo 14 13
9. Respuesta en frecuencia de un filtro DMA 13
10. Filtro Butterworth con $\omega_c = 0.044$ 15
11. Respuesta en frecuencia de un filtro Butterworth con $\omega_c = 0.044$ . . . 15
12. Filtrado con SuperSmoother para $\omega_c = 0.044$ 17
13. Respuesta en frecuencia de un filtro SuperSmoother con $\omega_c = 0.044$ . . 17
14. Comparación de las respuestas en frecuencia de los filtros 18
15. Cruce de dos filtros SMA con $\omega_{c1} = 0.015$ y $\omega_{c2} = 0.005$ 19
16. Ejecución del backtesting para la estrategia con SMA 21
17. Rentabilidades de las distintas estrategias en función del tiempo . . . 24
18. Drawdown de las diferentes estrategias 25

Índice de cuadros

1. Comparación de estrategias de trading 24

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 Monte Carlo 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 [7].

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 [8]:

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

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 $\delta[n]$.

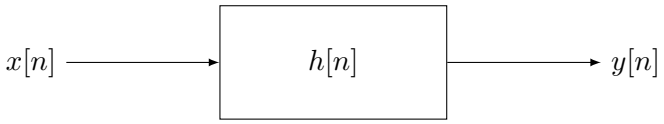

Figura 2: Relación entre la señal de entrada $x[n]$ y la señal de salida $y[n]$ a través de la respuesta al impulso $h[n]$.

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 [3].

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_i 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.

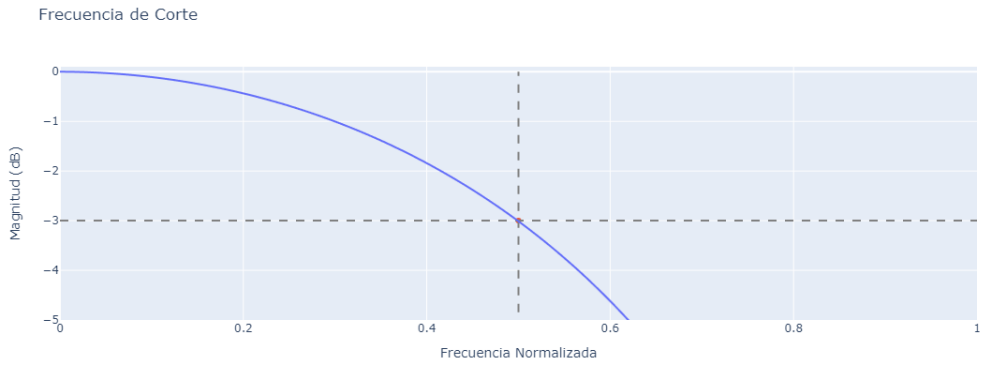

Figura 3: Filtro paso-bajo con $\omega_c = 0.5$

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 [5]:

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

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:

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

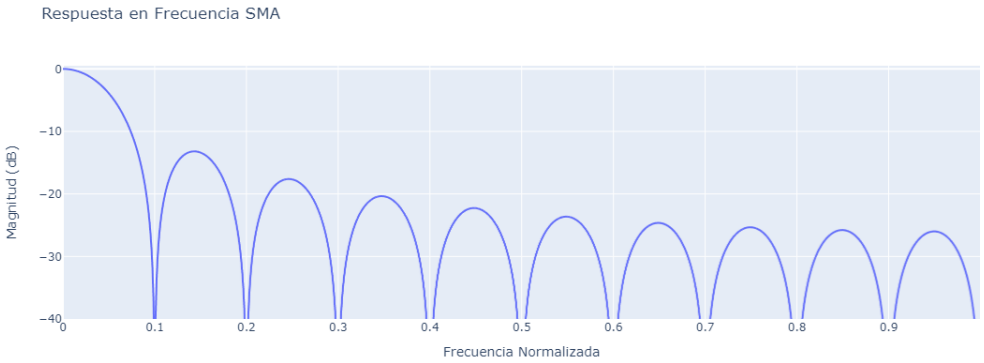

Figura 5: Respuesta en frecuencia de una SMA de 20 periodos

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 [1], 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 [5]:

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

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.

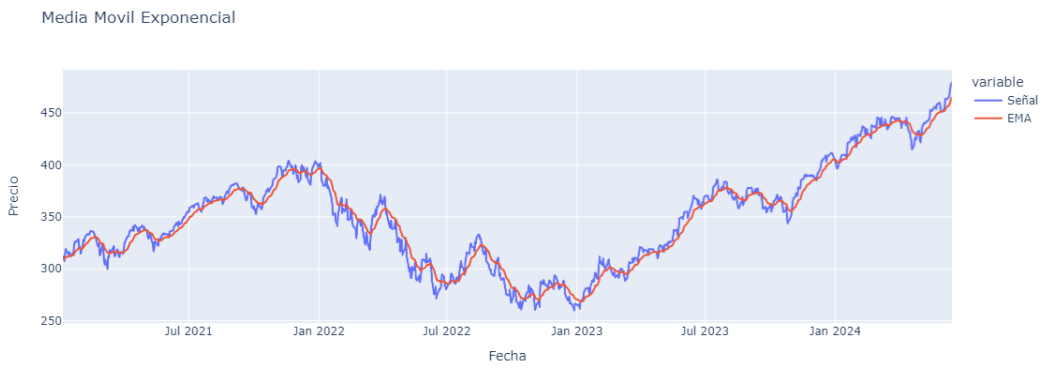

Figura 6: Filtrado con una EMA donde $\alpha = 0.13$

Por otro lado, la respuesta en frecuencia de la EMA es la siguiente [1]:

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

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.

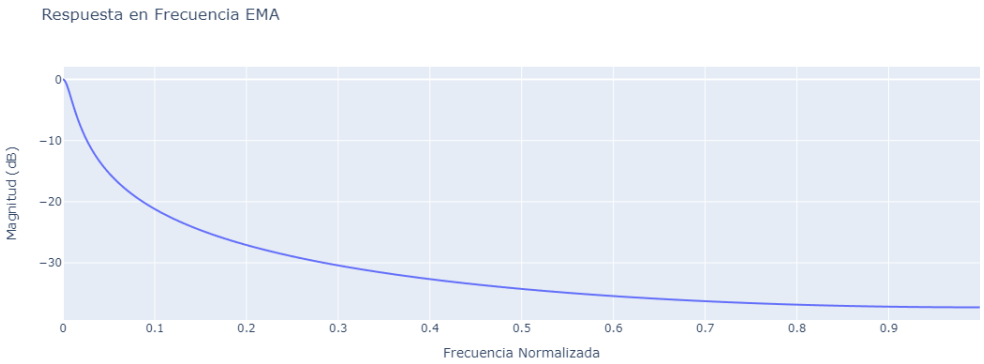

Figura 7: Respuesta en frecuencia de una EMA con $\alpha = 0.13$

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:

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

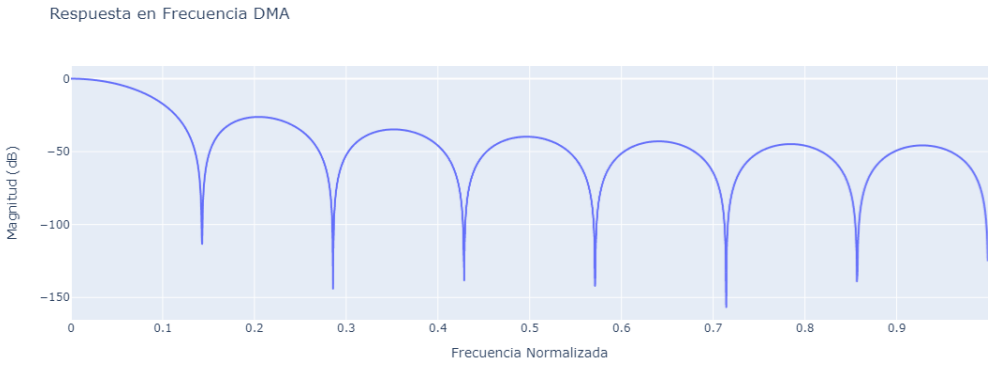

Figura 9: Respuesta en frecuencia de un filtro DMA

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 [5]. 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:

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

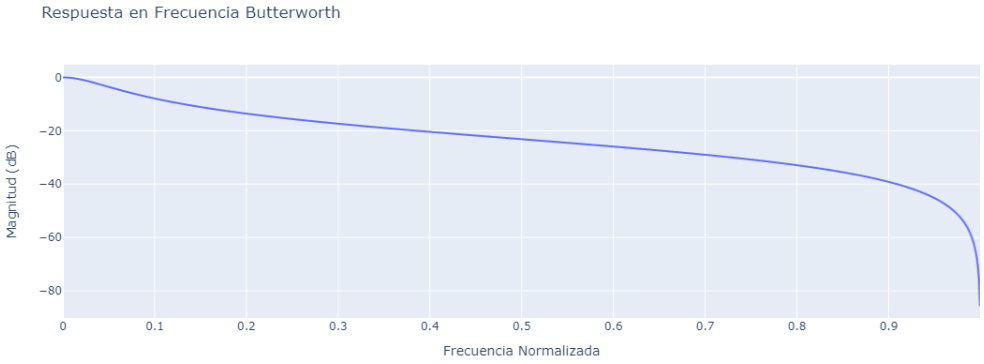

Figura 11: Respuesta en frecuencia de un filtro Butterworth con $\omega_c = 0.044$

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 [5]. 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:

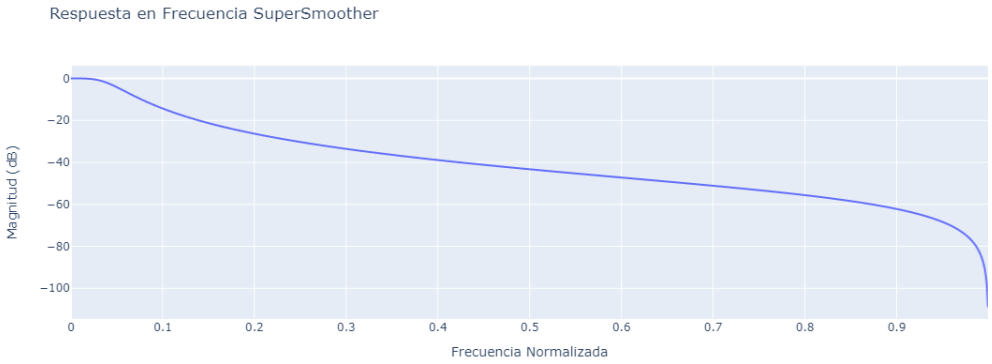

Figura 13: Respuesta en frecuencia de un filtro SuperSmoother con $\omega_c = 0.044$

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:

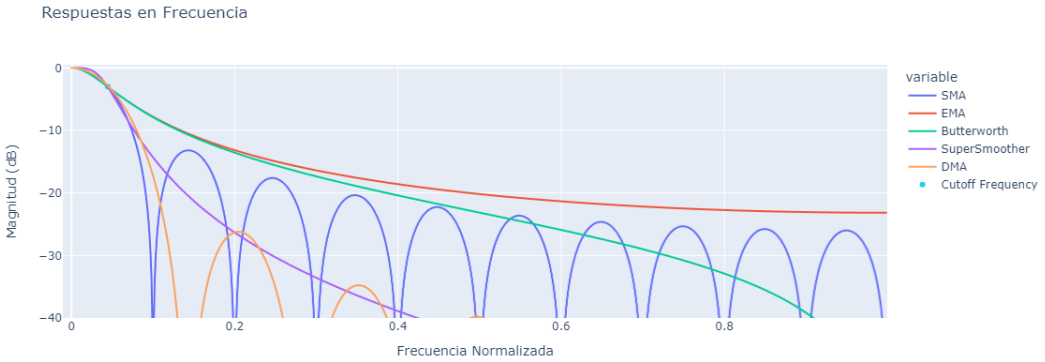

Figura 14: Comparación de las respuestas en frecuencia de los filtros

De cara a la siguiente sección, lo que haremos será definir una ω_c arbitraria y posteriormente optmizar las estrategias de trading para obtener la ω_c^* óptima que máximice 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 [6] y [4].

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* [2]. 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:

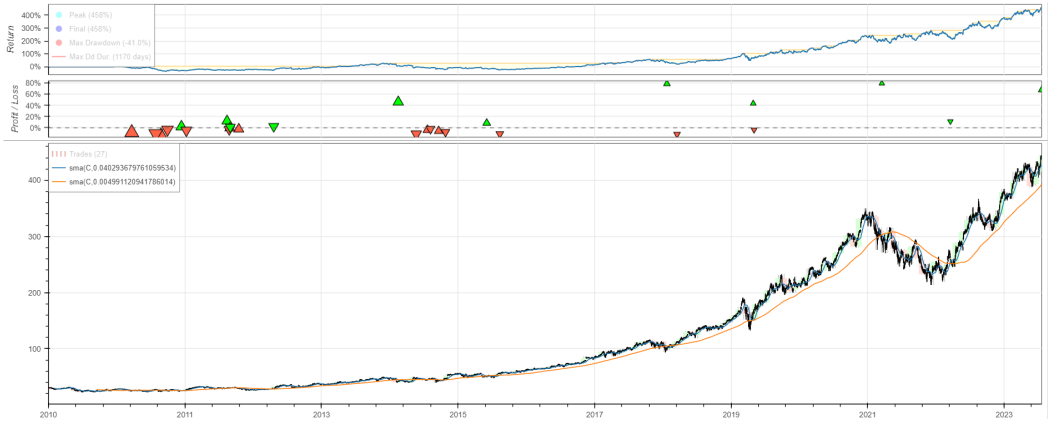

Figura 16: Ejecución del backtesting para la estrategia con SMA

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 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 es 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.

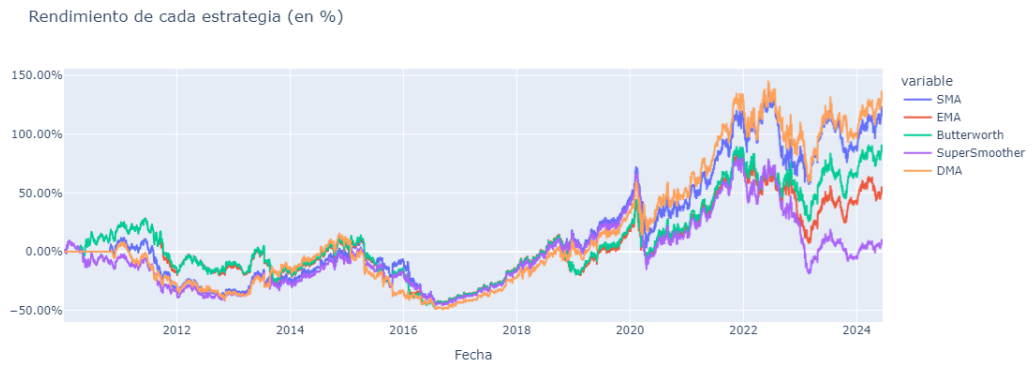

Figura 17: Rentabilidades de las distintas estrategias en función del tiempo

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.

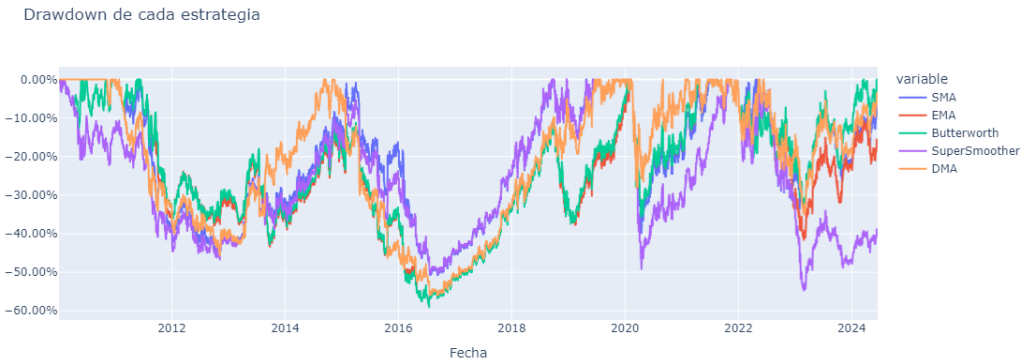

Figura 18: Drawdown de las diferentes estrategias

5 Optimización

5.1. Consideraciones Previas

5.2. Método Exhaustivo

5.3. Método Bayesiano

5.4. Resultados

6 Validación del sistema

6.1. Análisis de sensibilidad

6.2. IS vs OOS

6.3. Análisis de Dependencia

6.4. Método Monte Carlo

6.5. Simulación con Comisiones

6.6. Simulación de Slippage Extremo

6.7. Simulación en Otros Mercados

7 Conclusiones

Referencias

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

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

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

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

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

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

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

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