Plataforma de inversión bursátil con fines de investigación

adquisición de datos, creación y simulación de estrategias, optimización de parámetros y generación de resultados

Javier Matos Odut

Resumen

Este proyecto consiste en el desarrollo de una plataforma de inversión bursátil con fines de investigación. El objetivo de la plataforma es el de automatizar aquellos procesos que intervienen en el estudio de las estrategias de negociación de acciones o estrategias de trading: la adquisición de datos desde los proveedores de información bursátil, la creación y simulación de estrategias basadas en diferentes modelos matemáticos, la optimización de los parámetros de las estrategias y la generación de resultados descriptivos.

La plataforma de inversión bursátil es una herramienta de gran valor para la investigación. Permite medir, analizar y manipular cualquier aspecto relacionado con las estrategias de trading. Más allá de proporcionar una base, esta plataforma presenta una arquitectura pensada para ser extendida de acuerdo a las necesidades.

La plataforma también es una herramienta útil para el inversor. Gracias al simulador que implementa se puede estudiar el comportamiento de las estrategias sin la necesidad de operar en bolsa. De esta manera, el inversor puede comprobar con anticipo las consecuencias de sus decisiones para evitar aquellas inversiones más arriesgadas y propensas a incurrir en pérdidas.

Las aportaciones más notables e innovadoras en el desarrollo de la plataforma han sido las estrategias de trading compuestas, con las que se pueden combinar otras estrategias para conseguir mejores resultados; el mecanismo de optimización de parámetros, que empléa, entre otros, un algoritmo genético distribuido para el proceso de optimización; y el sistema de generación de resultados, capaz de representar el espacio de búsqueda para estudiar las relaciones entre los valores óptimos de las estrategia de trading.

Índice general

1. Introducción **17**

1.1. Motivación 18
1.2. Objetivos 18
1.3. Metodología de trabajo 19
1.4. Recursos utilizados 19
1.4.1. Hardware y equipamiento 19
1.4.2. Software para desarrollo 20
1.4.3. Software para redactar la documentación 20
1.4.4. Justificación de las elecciones 20
1.5. Aportaciones realizadas 21
1.6. Estructura del documento 21

2. Inversión bursátil **23**

2.1. ¿Qué es una acción? 23
2.2. La negociación de acciones 24
2.2.1. Compra 24
2.2.2. Venta 25
2.2.3. Posición larga 25
2.2.4. Posición corta 25
2.2.5. Fluctuaciones del precio 26
2.3. La Bolsa de valores 26
2.3.1. Características 27
2.3.2. Participantes 27
2.3.3. Función económica 27
2.4. El problema de la negociación de acciones 28
2.5. Maneras de enfrentar el problema 29
2.5.1. Predicción de series temporales 29
2.5.2. Herramientas del campo de las finanzas 30

3. Plataforma de inversión bursátil **33**

3.1. Adquisición de datos 35
3.1.1. La clase `DataSerie` 36

3.2. Máquinas de trading . . . 37
3.2.1. La clase `AlgoTrader` . . . 38
3.3. Optimización de parámetros . . . 40
3.3.1. El método `optimize` . . . 41
3.4. Generación de resultados . . . 48
3.5. El archivo de configuración `Settings.m` . . . 48

4. Conexión con los proveedores de datos **49**
4.1. Google Finance . . . 50
4.1.1. Interacción con el proveedor . . . 50
4.1.2. Estructura de datos remota . . . 52
4.1.3. Transferencia de datos . . . 52
4.1.4. Extracción de la información . . . 53
4.1.5. Almacenamiento local . . . 53
4.1.6. La clase `GoogleDataSerie` . . . 53
4.2. Yahoo! Finance . . . 53
4.2.1. Interacción con el proveedor . . . 54
4.2.2. Estructura de datos remota . . . 56
4.2.3. Transferencia de datos . . . 56
4.2.4. Extracción de la información . . . 57
4.2.5. Almacenamiento local . . . 57
4.2.6. La clase `YahooDataSerie` . . . 57
4.3. VisualChart V . . . 57
4.3.1. Interacción con el proveedor . . . 58
4.3.2. Estructura de datos remota . . . 59
4.3.3. Transferencia de datos . . . 59
4.3.4. Extracción de la información . . . 59
4.3.5. Almacenamiento local . . . 60
4.3.6. La clase `VisualChartDataSerie` . . . 60
4.3.7. La biblioteca de enlace dinámico `VisualChartConnector.dll` . . . 61

5. Estrategias de trading **63**
5.1. Basadas en indicadores técnicos . . . 64
5.1.1. Media móvil . . . 66
5.1.2. Media móvil desplazada . . . 68
5.1.3. Media móvil con umbral . . . 71
5.1.4. Cruce de medias móviles . . . 73
5.1.5. Cruce de medias móviles con umbral . . . 78
5.1.6. Bandas de Bollinger . . . 79
5.2. Basadas en osciladores técnicos . . . 81
5.2.1. Convergencia/divergencia de medias móviles . . . 82

5.2.2. Índice de fuerza relativa . . . 86
5.2.3. Cruce del índice de fuerza relativa . . . 90
5.2.4. Estocástico . . . 91
5.2.5. Cruce del estocástico . . . 93
5.2.6. Oscilador de momento . . . 96
5.2.7. Cruce del oscilador de momento . . . 97
5.3. Basadas en otros conceptos . . . 100
5.3.1. Falso óptimo . . . 100
5.3.2. Estrategia aleatoria . . . 102
5.3.3. Últimas n variaciones del precio . . . 103
5.3.4. Últimas n variaciones del precio ponderadas . . . 107
5.3.5. Proporción relativa de las últimas n variaciones del precio . . . 108

6. Estrategias de trading compuestas 113
6.1. Conjunto . . . 114
6.2. Intersección . . . 116
6.3. Intersección elitista . . . 116
6.4. Votación . . . 120
6.5. Fragmentación . . . 123
6.6. Máquinas compuestas interpretadas como operadores . . . 127
6.6.1. Operadores de intersección, intersección elitista y votación . . . 127
6.6.2. Operador de fragmentación . . . 128
6.6.3. Combinando operadores . . . 128

7. Optimización de parámetros 131
7.1. Sobre la naturaleza del problema . . . 131
7.2. Funciones de evaluación . . . 132
7.2.1. Beneficio/Pérdida . . . 133
7.2.2. Pérdida . . . 134
7.2.3. Mínimo beneficio/pérdida esperado . . . 135
7.3. Funciones de selección . . . 137
7.4. Métodos de optimización . . . 137
7.4.1. Método enumerativo . . . 138
7.4.2. Búsqueda aleatoria . . . 141
7.4.3. Algoritmo genético . . . 141

8. Generación de resultados 149
8.1. Ámbito individual . . . 149
8.1.1. Señal de posición . . . 149
8.1.2. Beneficio/pérdida . . . 150
8.1.3. Inversión por posición . . . 151
8.1.4. Resultado por posición . . . 151

8.1.5. Funciones de evaluación . . . 152
8.2. Ámbito colectivo . . . 153
8.2.1. Espacio de búsqueda . . . 154
8.2.2. Estadísticos . . . 159

9. Conclusiones y trabajo futuro **161**
9.1. Conclusiones . . . 161
9.2. Trabajo futuro . . . 162

A. La clase `DataSerie` **163**
A.1. Propiedades . . . 163
A.1.1. `Close` . . . 164
A.1.2. `CompressionType` . . . 164
A.1.3. `CompressionUnits` . . . 164
A.1.4. `DateTime` . . . 164
A.1.5. `DiffSerie` . . . 164
A.1.6. `EndDateTime` . . . 165
A.1.7. `High` . . . 165
A.1.8. `InitDateTime` . . . 165
A.1.9. `Length` . . . 165
A.1.10. `Low` . . . 165
A.1.11. `Open` . . . 165
A.1.12. `Serie` . . . 165
A.1.13. `SymbolCode` . . . 166
A.1.14. `Volume` . . . 166
A.2. Métodos . . . 166
A.2.1. `DataSerie` . . . 166
A.2.2. `drawClose` . . . 167
A.2.3. `drawDiffSerie` . . . 167
A.2.4. `drawHigh` . . . 168
A.2.5. `drawLow` . . . 168
A.2.6. `drawOpen` . . . 169
A.2.7. `drawSerie` . . . 169
A.2.8. `drawSeries` . . . 169
A.2.9. `drawVolume` . . . 170
A.2.10. `equals` . . . 170
A.2.11. `firstFrom` . . . 171
A.2.12. `get` . . . 171
A.2.13. `lastUntil` . . . 172
A.2.14. `plot` . . . 172
A.2.15. `plotClose` . . . 173

A.2.16. plotDiffSerie . . . 173
A.2.17. plotHigh . . . 173
A.2.18. plotLow . . . 174
A.2.19. plotOpen . . . 174
A.2.20. plotSerie . . . 175
A.2.21. plotSeries . . . 175
A.2.22. plotVolume . . . 176
A.2.23. plotWrapper . . . 176
A.3. El sistema de dibujado . . . 177

B. La clase AlgoTrader **179**
B.1. Propiedades . . . 179
B.1.1. AllowedPositions . . . 180
B.1.2. CurrentFunds . . . 180
B.1.3. DataSerie . . . 180
B.1.4. InitialFunds . . . 180
B.1.5. InvertSignal . . . 180
B.1.6. InvestmentLimit . . . 180
B.1.7. ProfitLossTestSet . . . 181
B.1.8. ProfitLossTrainingSet . . . 181
B.1.9. Signal . . . 181
B.1.10. TestSet . . . 181
B.1.11. TradingCost . . . 181
B.1.12. TrainingSet . . . 181
B.2. Métodos . . . 182
B.2.1. AlgoTrader . . . 182
B.2.2. clone . . . 182
B.2.3. computeSignal . . . 183
B.2.4. diffProfitLossSerie . . . 183
B.2.5. drawDiffProfitLoss . . . 184
B.2.6. drawPosition . . . 185
B.2.7. drawProfitLoss . . . 185
B.2.8. drawSeriePosition . . . 186
B.2.9. drawSignal . . . 186
B.2.10. fitness . . . 187
B.2.11. fitnessFunctionWrapper . . . 187
B.2.12. fitnessStatistics . . . 188
B.2.13. optimize . . . 189
B.2.14. plot . . . 189
B.2.15. plotDiffProfitLoss . . . 190
B.2.16. plotPosition . . . 190

B.2.17. `plotProfitLoss` . . . 191
B.2.18. `plotSearchSpace` . . . 191
B.2.19. `plotSearchSpace123` . . . 192
B.2.20. `plotSeriePosition` . . . 192
B.2.21. `plotSignal` . . . 193
B.2.22. `plotWrapper` . . . 194
B.2.23. `positionSerie` . . . 194
B.2.24. `positionsLog` . . . 195
B.2.25. `printPositionsLog` . . . 196
B.2.26. `profitLoss` . . . 197
B.2.27. `profitLossSerie` . . . 197
B.2.28. `resetSignal` . . . 199
B.2.29. `signal2positions` . . . 199
B.2.30. `updateSignal` . . . 200
B.3. El sistema de dibujado . . . 200

C. Ejemplos de uso **203**
C.1. Toma de contacto . . . 203
C.2. Trabajando con series de datos . . . 203
C.3. Utilizando las máquinas de trading . . . 203
C.4. Diseñando una máquina de trading compuesta . . . 203
C.5. Optimizando el desempeño de las máquinas . . . 204
C.6. Analizando los resultados . . . 204

Bibliografía **205**

Índice de figuras

3.1. Bloques funcionales construidos sobre la plataforma de inversión bursátil . . . 34
3.2. Elementos que intervienen en la llamada al método `optimize` 47

4.1. Información sobre una acción bursátil en Google Finance 50
4.2. Cotización de una acción bursátil en Google Finance 51
4.3. Información sobre una acción bursátil en Yahoo! Finance 54
4.4. Cotización de una acción bursátil en Yahoo! Finance 55
4.5. Interfaz de usuario de la aplicación VisualChart V 58

5.1. Gráfico de una máquina de trading 65
5.2. Media móvil exponencial 68
5.3. Gráfico de la máquina de trading `MovingAverage` 70
5.4. Media móvil exponencial desplazada 71
5.5. Media móvil exponencial con umbral 72
5.6. Gráfico de la máquina de trading `MovingAverageThreshold` 74
5.7. Cruce de medias móviles exponenciales 75
5.8. Gráfico de la máquina de trading `MovingAveragesCrossing` 77
5.9. Cruce de medias móviles exponenciales con umbral 78
5.10. Bandas de Bollinger 81
5.11. Convergencia/divergencia de medias móviles 84
5.12. Gráfico de la máquina de trading `MovingAverageConvergenceDivergence` . . 87
5.13. Índice de fuerza relativa 88
5.14. Cruce del índice de fuerza relativa 91
5.15. Estocástico 92
5.16. Gráfico de la máquina de trading `Stochastic` 94
5.17. Cruce del estocástico 95
5.18. Oscilador de momento 97
5.19. Gráfico de la máquina de trading `Momentum` 98
5.20. Cruce del oscilador de momento 99
5.21. Falso óptimo 102
5.22. Gráfico de la máquina de trading `Random` 104

5.23. Gráfico de la máquina de trading `Random` sin las restricciones de longitud mínima y máxima en la duración de la posición 105
5.24. Últimas n variaciones del precio 107
5.25. Proporción relativa de las últimas n variaciones del precio 110

6.1. Gestión de propiedades dinámicas en la clase `Joint` 114
6.2. Intersección de las posiciones de las estrategias 117
6.3. Intersección de señales de posición 118
6.4. Intersección elitista de las posiciones de las estrategias 119
6.5. Intersección elitista de señales de posición 120
6.6. Votación de las posiciones de las estrategias 122
6.7. Votación de señales de posición 123
6.8. Valor óptimo de la media móvil según intervalo 124
6.9. División de una serie de datos en los intervalos de entrenamiento y test 124
6.10. División de una serie de datos en múltiples intervalos de entrenamiento y test 124
6.11. Representación en árbol del operador de conjunto 127
6.12. Representación en árbol del operador de fragmentación 128
6.13. Representación en árbol de una combinación de operadores 129

7.1. Esquema de un algoritmo genético 143
7.2. Esquema de un algoritmo genético distribuido en islas 145

8.1. Señal de posición de una máquina de trading 150
8.2. Curva de beneficio/pérdida en los diferentes intervalos 152
8.3. Volumen de capital invertido en los diferentes intervalos 153
8.4. Espacio de búsqueda para una función de dos dimensiones 155
8.5. Espacio de búsqueda de la máquina `MovingAveragesCrossing` y la función de evaluación `profitLoss` 156
8.6. Espacio de búsqueda de la máquina `RelativeStrengthIndexCrossing` y la función de evaluación `profitLoss` 156
8.7. Espacio de búsqueda de la máquina `MovingAveragesCrossing` y la función de evaluación `minProfitLossExpected` 157
8.8. Espacio de búsqueda generado por la máquina `MovingAveragesCrossing` y la función de evaluación `loss` 157
8.9. Representación de un espacio de búsqueda de tres dimensiones 158
8.10. Representación de un espacio de búsqueda de tres dimensiones reducido a un espacio de dos dimensiones 158
8.11. Histograma de una máquina del tipo `ThreeMovingAverages` 160
8.12. Histograma de una máquina del tipo `MovingAveragesCrossing` 160

Índice de tablas

3.1. Tabla de registros de posición 43

4.1. Contenido del archivo de datos descargado desde Google Finance 52
4.2. Contenido del archivo de datos descargado desde Yahoo! Finance 56
4.3. Descripción de la estructura de datos provista por el conector VisualChart V . 60

6.1. División de una serie de datos en múltiples intervalos de entrenamiento y test 125

7.1. Descripción de las columnas de la tabla de registros de posición 133
7.2. Resultados de la función `index2indexArray` 139
7.3. Valores por defecto del algoritmo genético distribuido 147

8.1. Representación de la tabla de registros de posición 154

Índice de fragmentos de código

5.1. Método `computeSignal` de la clase `MovingAverage` 69
5.2. Método `computeSignal` de la clase `MovingAveragesCrossing` 76
5.3. Método `computeSignal` de la clase `MovingAverageConvergenceDivergence` . 85
5.4. Cálculo de estadísticas con el método `profitLossStatistics` 106
5.5. Método `computeSignal` de la clase `LastNWeighted` 109
7.1. Función de evaluación `profitLoss` 134
7.2. Función de evaluación `loss` 136
7.3. Función de evaluación `minProfitLossExpected` 137
7.4. Función de optimización `exhaustive` 140
7.5. Función de optimización `randomSearch` 142

Capítulo 1

Introducción

En este primer capítulo se presenta el proyecto realizado, que consiste en el desarollo de una plataforma de inversión bursátil con fines de investigación. El propósito perseguido es el de crear un entorno potente con el que se puedan emprender estudios y análisis rigurosos sobre las estrategias de trading[1]. Para ello, la plataforma ofrece una solución integral que cubre todas las estapas del proceso: adquisición de datos desde los proveedores de información bursátil, creación y simulación de estrategias, optimización de parámetros y generación de resultados.

En conjunto con la plataforma se han implementado las estrategias de trading más populares. Con estas, el usuario puede comenzar sus estudios o utilizarlas de punto de partida para crear nuevas estrategias. También se incluyen unas estrategias especiales con las que el usuario puede combinar un grupo de estrategias para lograr resultados más elaborados.

La plataforma fue desarrollada con fines de investigación, y ofrece una gran flexibilidad y productividad a expensas de un pequeño coste en forma de curva de aprendizaje. Hay que resaltar que esta no es una herramienta destinada al uso profesional.

La culminación del proyecto ha requerido del empleo de algunas de las herramientas más vanguardistas del campo de las ciencias computacionales. En concreto, se han utilizado técnicas de metaprogramación, modelos de aprendizaje para la predicción de series temporales y algoritmos basados en metaheurísticas para el cálculo de los parámetros óptimos.

El resto del capítulo proporciona más información sobre el proyecto realizado. En la sección 1.1 se habla de la motivación de abordar un problema sobre la inversión en bolsa: la esperanza de conseguir grandes beneficios. La sección 1.2 menciona los objetivos que se plantean. La sección 1.3 describe la metodología de trabajo seguida durante la ejecución del proyecto. Los recursos hardware y software utilizados quedan recogidos en la sección 1.4. En la sección 1.5 se enumeran las aportaciones del proyecto. Por último, en la sección 1.6 se detalla la estructura del documento.

[1] Una estrategia de trading consiste en un conjunto de reglas que establecen la manera en que se negocia con ciertas acciones de la bolsa. Esto no es lo mismo que las estrategias de inversión, que se refieren al conjunto de reglas diseñadas para orientar al inversor en la selección de una cartera de inversiones.

1.1. Motivación

La inversión en bolsa se erige como una actividad lucrativa con potencial para producir grandes beneficios. Empujadas por estas expectativas, muchas personas se dedican a invertir en bolsa. Esta situación da lugar a grandes movimientos de capital, en los que cada participante se ve recompensado o penalizado de forma monetaria según el grado de acierto de las decisiones con las que intervino en el mercado. En este sentido, sería interesante contar con estrategias que indiquen cómo operar en bolsa para maximizar el beneficio y minimizar las pérdidas. La motivación principal de este proyecto consiste en desarrollar una plataforma con la que enfrentar el reto de generar estrategias de trading competitivas.

Acercar el problema de la creación de estrategias de trading al mundo académico permite contar con herramientas formales que introducen medida y control. A partir de la medida es posible cuantificar y atribuir valor a cada aspecto del problema: la importancia de la medida radica en que aporta objetividad en la comprensión del problema. El control permite articular soluciones al problema con conocimiento y previsión.

Otra motivación consiste en el uso de metaheurísticas para optimizar los resultados de las estrategias. Las metaheurísticas son técnicas potentes que resuelven problemas de computación generales de manera eficiente. La fortaleza de estas técnicas radica en su efectividad para resolver problemas de optimización no resolubles con otros métodos.

1.2. Objetivos

Los objetivos del proyecto son cada una de las partes que conforman la plataforma de inversión, que se complementan para ofrecer una solución integral. Estas partes aparecen listadas a continuación:

Plataforma de inversión bursátil Entorno que provee la funcionalidad base requerida para el diseño, implementación y estudio de cualquier estrategia de trading: facilita la conectividad con los proveedores de información, ofrece gráficos representativos para los datos de entrada y las estrategias de trading, cuenta con un motor de simulación para evaluar las estrategias... Define interfaces para extender cualquier aspecto de la plataforma gracias al mecanismo de herencia de clases.

Conexión con los proveedores de datos Sistema de recepción de datos desde los proveedores de información bursátil. Implementa la conectividad con dos proveedores gratuitos como son Yahoo! Finance y Google Finance. También incorpora conectividad con la aplicación VisualChart V, que permite obtener datos con mayor resolución (en intervalos de tiempo más cortos).

Estrategias de trading Conjunto de estrategias creadas sobre la plataforma. Están basadas principalmente en técnicas recogidas en el análisis técnico. Para cada una de estas estrategias se explica cómo funciona, el modelo matemático en que se basa y la manera en que ha sido implementada.

Estrategias de trading compuestas Conjunto de estrategias concebidas para la creación de nuevas estrategias. Pueden interpretarse como elementos constructores que toman unas estrategias de partida y dan lugar a nuevas estrategias producto de la combinación.

Optimización de parámetros Mecanismo para el cálculo de los parámetros óptimos en las estrategias de trading. Define funciones de evaluación, funciones de selección y métodos de optimización que pueden ser empleados para mejorar el desempeño de las estrategias. Las posibilidades del mecanismo de optimización soy muy amplias, como consecuencia de la manera en que ha sido implementado.

Generación de resultados Métodos para generar resultados descriptivos del funcionamiento de las estrategias de trading. Es una parte esencial de la plataforma, pues la generación de resultados es vital en el análisis y estudio de las estrategias. Es necesario implementar una gran cantidad de funciones y métodos con el objetivo de recopilar información de casi cualquier aspecto de las estrategias.

1.3. Metodología de trabajo

Se ha seguido la metodología de trabajo del diseño incremental. Consiste en un proceso de desarrollo basado en iteraciones sucesivas que en cada ocasión incrementa la funcionalidad del software producido. Inicialmente, se contó con la definición del problema y el conjunto de objetivos a cumplir. Luego, se descompusieron los objetivos en un conjunto de subobjetivos que correspondían con funcionalidades concretas. En cada iteración del desarrollo se seleccionaban un grupo de funcionalidades que eran implementadas.

Para las fases de trabajo se detectaron las dependencias entre los subobjetivos, estableciéndose un orden relativo entre ellos. Por ejemplo, existían subobjetivos consistentes en obtener información bursátil de los proveedores y también otros que trataban sobre representar gráficamente la información adquirida. Como es necesario contar con la información antes de poder representarla, entonces el subobjetivo relativo a esta funcionalidad fue alcanzado antes que el subobjetivo relativo a la representación gráfica de la información.

1.4. Recursos utilizados

Esta sección se incluye para ofrecer una idea del entorno de desarrollo utilizado en la realización del proyecto. Se facilita así la posibilidad de reproducir los resultados obtenidos utilizando los mismos medios. También, se incluye un apartado dedicado a justificar el porqué de esta elección del software de desarrollo en detrimento de cualquier otra.

1.4.1. Hardware y equipamiento

- Estación de trabajo con capacidad de procesamiento paralelo
- Conexión a Internet

1.4.2. Software para desarrollo

- Sistema operativo Microsoft Windows 7 Ultimate Service Pack 1 (64 bits)
- Entorno de desarrollo MATLAB R2011a (64 bits)
- Entorno de desarrollo Microsoft Visual Studio 2010 Professional Service Pack 1
- Software para trading VisualChart V

1.4.3. Software para redactar la documentación

- LyX 2.0.0
- MiKTeX 2.9

1.4.4. Justificación de las elecciones

Considerando la naturaleza del problema y su caracter matemático, el entorno de desarrollo elegido ha sido MATLAB. Este potente entorno ofrece una alta productividad y permite extender sus capacidades a partir de la utilización de las diferentes *toolboxes* con las que cuenta. Existen en el mercado alternativas como Octave, Scilab o Sage, aunque no son productos lo suficientemente maduros como para permitir el desarrollo de este proyecto.

Dada la intención de obtener datos con información bursátil de calidad (con la precisión deseada y en el intervalo que se considere oportuno), se hace uso del software de trading VisualChart V. De cuantos programas fueron analizados, es el único gratuito que ofrece una API (*Application Programming Interface*) con la que descargar datos del mercado bursátil. El primer problema que tiene este programa es que sólo funciona bajo el sistema operativo Windows. El segundo problema es que la interacción con su API se hace por medio de la interfaz COM (*Component Object Model*). A consecuencia de esto último, MATLAB no puede tratar directamente con VisualChart V, ya que las capacidades COM que implementa MATLAB son sólo un subconjunto de las propuestas por la interfaz, y no alcanza a reconocer ciertos tipos de datos.

Para superar los problemas de conectividad entre MATLAB y VisualChart V se utilizan las tecnologías de Microsoft. La plataforma .NET, desarrollada por Microsoft, ofrece soporte nativo con la interfaz COM, también creada por esta misma compañía. Además, sucede que la integración de .NET con MATLAB es tal que incluso se pueden crear y gestionar instancias de clases .NET desde la consola de MATLAB. De esta manera, se desarrolló un programa en el lenguaje C# que se encarga de las comunicaciones entre MATLAB y VisualChart V. Este programa es imperceptible para el usuario, pues las instancias producidas durante su ejecución se controlan de forma transparente por MATLAB.

1.5. Aportaciones realizadas

La principal aportación del proyecto es la plataforma de inversión, que es una herramienta de investigación de gran utilidad. Con esta plataforma se simplifica todo el proceso de creación, análisis y estudio de las estrategias de trading. Más allá de este software, las contribuciones del proyecto son las siguientes:

- La definición formal del problema de la negociación de acciones. Habitualmente, los trabajos que tratan sobre este problema omiten cualquier tipo de definición precisa y la suponen como conocida.

- El mecanismo de optimización de parámetros. Esta es una parte muy importante de la plataforma, en cuya implementación se han utilizado técnicas de metaprogramación y patrones de diseño que favorecen la genericidad y modularidad. La ventaja de este enfoque es que permite optimizar cualquier característica medible de la estrategia con los algoritmos de optimización elegidos. El mecanismo de optimización tiene la capacidad de detectar los parámetros que tienen las estrategias de trading (implementadas como máquinas de trading) y optimizar los valores que toman.

- La implementación de un algoritmo genético distribuido en MATLAB. Con este método de optimización paralelo se pueden afrontar los problemas de optimización de parámetros más duros.

- Las estrategias de trading compuestas. Son una extensión de las estrategias de trading que surgen para paliar algunos de sus problemas. Con ellas se pueden combinar otras estrategias para producir mejores resultados. El objetivo detrás de la combinación de varias estrategias es el de potenciar las fortalezas y reducir las debilidades. Esta aportación en particular introduce una infinidad de posibilidades para generar nuevas estrategias de trading.

- La generación de resultados. Se ofrecen resultados muy elaborados y expresivos: es posible dibujar el espacio de búsqueda asociado a los valores de varios parámetros para apreciar de forma visual el comportamiento de una estrategia en cada caso. De esta forma, se evidencian las relaciones existentes entre aquellos conjuntos de valores de los parámetros que mejores resultados ofrecen. Así mismo, también se proporcionan estudios estadísticos derivados de los espacios de búsqueda.

1.6. Estructura del documento

El documento está estructurado de la siguiente manera:

Capítulo 1 Introduce el proyecto desarrollado: la descripción general, los objetivos perseguidos, la metodología de trabajo aplicada, los recursos utilizados y las aportaciones realizadas.

Capítulo 2 Describe el contexto en el que se formula el problema de finanzas para el que se ha creado la plataforma: explica en qué consiste la inversión bursátil, qué son las acciones, cómo se negocia con ellas y qué es la Bolsa de valores. Define el problema de la negociación de acciones de forma concisa. Recoge las maneras en que se ha intentado resolver este problema desde diferentes enfoques.

Capítulo 3 Ofrece una vista global de la plataforma de inversión: enumera las partes que la componen y explica cómo se relacionan. Los detalles concernientes a cada una de las partes de la plataforma están recogidos en los siguientes capítulos.

Capítulo 4 Explica en detalle el proceso de conexión con los proveedores de datos. Para cada uno de ellos se cuenta cómo sucede la interacción, cuál es la estructura de datos, de qué manera se efectúa la transferencia, cómo se extrae la información y la forma en que se almacena.

Capítulo 5 Explica las estrategias de trading implementadas en la plataforma: describe el fundamento en que se sustentan, presenta los modelos matemáticos en se basan, revela la manera en que se calculan las señales de indicación para invertir en el mercado e introduce las clases en que se implementan.

Capítulo 6 Documenta todos los aspectos relativos a las estrategias de trading compuestas: el motivo por el que surgen, los problemas que resuelven, los principios en que se sostienen y la manera en que se implementan. Analiza las posibilidades de interpretar las estrategias de trading compuestas como operadores en el sentido matemático.

Capítulo 7 Describe el mecanismo de optimización centrándose en la explicación de sus partes: las funciones de evaluación, las funciones de selección y los métodos de optimización. Comenta la manera en que colaboran estos elementos durante el proceso de optimización.

Capítulo 8 Analiza las posibilidades de generación de resultados que ofrece la plataforma. Comenta la manera en que se pueden agregar nuevas funciones para medir características adicionales de las máquinas de trading.

Capítulo 9 Recoge las conclusiones derivadas del proyecto y las líneas de trabajo futuro.

Apéndice A Documenta los aspectos técnicos de la clase `DataSerie` que implementa el contenedor de datos bursátiles. Enumera las propiedades y métodos de la clase.

Apéndice B Documenta las características técnicas de la clase `AlgoTrader` con la que se implementan las estrategias de trading. Enumera las propiedades y métodos de la clase.

Apéndice C Proporciona referencias hacia los ejemplos de uso que se incluyen en conjunto con la plataforma de inversión bursátil. Estos ejemplos ilustran el potencial que tiene la plataforma desarrollada como herramienta para la investigación de estrategias de trading.

Capítulo 2

Inversión bursátil

En finanzas, la inversión consiste en destinar una cantidad de capital a la adquisición de bienes con la intención de obtener unos ingresos o rentas a lo largo del tiempo. De esta manera, se renuncia a un consumo actual y cierto a cambio de conseguir unos beneficios futuros y distribuidos en el tiempo.

Los factores más relevantes que condicionan las decisiones de inversión son los siguientes:

Rendimiento esperado Es la compensación que se espera obtener por la inversión, su rentabilidad.

Riesgo aceptado Incertidumbre sobre cuál será el rendimiento real que se obtendrá al final de la inversión, que incluye además la estimación de la capacidad de pago (si la inversión podrá pagar los resultados al inversor).

Horizonte temporal Periodo durante el cual se mantiene la inversión.

El término bursátil se refiere a la Bolsa de valores, que es una organización privada que brinda las facilidades necesarias para que sus miembros, atendiendo al mandato de sus clientes, introduzcan las órdenes y realicen negociaciones de compra y venta de valores. Los tipos de valores que se comercian son acciones de sociedades o compañías anónimas, bonos públicos y privados, certificados, títulos de participación y una amplia variedad de instrumentos de inversión.

En consecuencia, la inversión bursátil consiste en el empleo de un capital para la compra de títulos (principalmente acciones) que se negocian en el mercado bursátil con la intención de generar beneficios a partir de las variaciones en el precio y el cobro de dividendos.

2.1. ¿Qué es una acción?

Una acción es una parte del capital social de una sociedad anónima. Representa la propiedad que una persona tiene de una parte de esa sociedad. Normalmente, las acciones son transmisibles de forma libre y otorgan derechos económicos y políticos a su titular. La emisión de acciones es el principal medio utilizado por las empresas para captar el capital requerido para el desarrollo de sus actividades.

Como inversión, una acción es un instrumento de renta variable, dado que no tiene un retorno fijo establecido por contrato, sino que depende de la buena marcha de dicha empresa.

Hay dos maneras de hacer dinero con las acciones: una es a través del pago de dividendos que el propietario de la acción recibe (en efectivo o en títulos cuando se reparten las utilidades de la compañía), la otra es mediante la diferencia entre el precio al que compra una acción y al que la vende más tarde.

2.2. La negociación de acciones

Salvo prohibición, las acciones de una empresa pueden ser transferidas desde el accionista (el titular en posesión de las acciones) hacia terceros, mediante su venta u otros mecanismos. Existen rigurosas leyes y reglamentos que rigen las transferencias, sobre todo si el emisor es una entidad que cotiza en bolsa.

El deseo de los accionistas de comerciar con sus acciones ha llevado a la creación de bolsas de valores. La Bolsa de valores es una organización que ofrece un mercado para la negociación de acciones, derivados y otros productos financieros. En la actualidad, los inversionistas suelen estar representados por un agente de bolsa que compra y vende acciones de una amplia gama de empresas en las bolsas. Una empresa puede cotizar sus acciones en el mercado bursátil si cumple los requisitos impuestos por la Bolsa de valores.

2.2.1. Compra

Son varias las maneras de comprar acciones. La forma mas común es a través de un corredor de bolsa que organiza la transferencia de acciones desde un vendedor hasta un comprador. La mayoría de transferencias se hacen por mediación de los corredores de bolsa que operan en la Bolsa de valores.

Existen muchos corredores de bolsa entre los que se puede elegir, los cuales se clasifican en dos grupos: los corredores de servicio completo, que cobran más por ofrecer consejos de inversión y un servicio más personalizado; y los corredores de descuento, que cobran menos a expensas de obviar los consejos de inversión. Otro tipo de corredor podría ser un banco o una cooperativa de crédito, que disponen de un acuerdo sea con un corredor de servicio completo o uno de descuento.

Hay dos maneras de financiar la compra de acciones: pagándolas con dinero en posesión del comprador o con la compra a crédito. Comprar acciones a crédito significa adquirirlas con dinero prestado por el corredor de bolsa, utilizando otras acciones que se tienen en la cuenta como aval. Estas acciones movilizadas en la cuenta son la garantía de que el comprador puede pagar el préstamo; de lo contrario, el corredor de bolsa tiene el derecho de vender las acciones de garantía para recuperar el dinero prestado. Este servicio no es gratis, sino que el corredor de bolsa cobra intereses por el préstamo. La compra de acciones a crédito está relacionada con apalancamiento.

2.2.2. Venta

La venta de acciones es un procedimiento similar al de la compra. La intención del inversor es la de comprar barato y vender caro, aunque una serie de razones pueden inducir a vender en pérdidas, como por ejemplo el evitar incurrir en pérdidas mayores.

Al igual que con la compra de acciones, hay una tarifa que cobra el corredor de bolsa por transferir las acciones del vendedor al comprador. Esta tarifa puede ser más alta o más baja, dependiendo del tipo de corredor que se encargue de la transferencia.

Una vez realizada la venta, el vendedor tiene plenos derechos sobre el dinero conseguido. Una parte importante en la venta de acciones es el seguimiento de los ingresos. Las ganancias así conseguidas, de producirse, están sometidas al pago de impuestos según esté establecido.

2.2.3. Posición larga

Una posición larga consiste en comprar acciones, esperar a que se produzca una apreciación en el valor de estas y venderlas. Para el caso, la operación produce beneficios si el precio de la acción sube y pérdidas si baja.

En una posición larga la secuencia de pasos que se toman es la siguiente:

1. Comprar las acciones en el mercado (se abre la posición larga).
2. Seguir la evolución del precio con el propósito de maximizar los beneficios y minimizar las pérdidas.
3. Vender las acciones en el mercado (se cierra la posición larga).

2.2.4. Posición corta

La manera en que se opera en la bolsa no tiene por qué ser primero comprando acciones y luego vendiéndolas. En este sentido, una posición corta se refiere a la venta de acciones que no se poseen, con la esperanza de que su precio se vea devaluado, para luego comprarlas y devolverlas al prestamista. De esta manera, la operación acarrea beneficios si el precio de la acción baja y pérdidas si sube.

La secuencia de pasos que se suceden en una posición corta es la siguiente:

1. Pedir prestadas las acciones a un prestamista, generalmente un corredor de bolsa.
2. Vender las acciones en el mercado (se abre la posición corta).
3. Seguir la evolución del precio con el propósito de maximizar los beneficios y minimizar las pérdidas.
4. Comprar las acciones en el mercado (se cierra la posición corta).
5. Devolver las acciones tomadas del prestamista con las compradas.
6. Pagar intereses por el préstamo concedido.

Las posiciones cortas son más arriesgadas que las largas: en el caso de las posiciones largas, el peor escenario consiste en la pérdida de todo el capital invertido, que es una cantidad acotada; sin embargo, en las posiciones cortas no hay cota máxima de la pérdida que se puede sufrir.

2.2.5. Fluctuaciones del precio

El precio de una acción fluctúa debido fundamentalmente a la teoría de la oferta y la demanda. Al igual que todos los productos en el mercado, el precio de una acción es sensible a la demanda. Por su parte, hay muchos factores que influyen en la demanda de una acción en particular.

El precio de las acciones puede verse influido por las previsiones de los analistas de la empresa, los competidores de la empresa, las perspectivas para el segmento del mercado, las agencias de calificación, los factores macroeconómicos...

Con el análisis fundamental y el análisis técnico se intentan entender las condiciones del mercado que dan lugar a cambios en los precios, o incluso predecir los precios en el futuro.

2.3. La Bolsa de valores

La Bolsa de valores es una organización privada que brinda las facilidades necesarias para que sus miembros, atendiendo al mandato de sus clientes, introduzcan órdenes y realicen negociaciones de compra y venta de valores. Los tipos de valores que se comercian son acciones de sociedades o compañías anónimas, bonos públicos y privados, certificados, títulos de participación y una amplia variedad de instrumentos de inversión.

La negociación de los valores en los mercados bursátiles se hace tomando como base unos precios conocidos y fijados en tiempo real, en un entorno seguro para la actividad de los inversionistas, donde el mecanismo de las transacciones está totalmente regulado, lo que garantiza la legalidad, la seguridad y la transparencia.

Las bolsas fortalecen al mercado de capitales e impulsan el desarrollo económico y financiero en la mayoría de los países del mundo, donde existen en algunos casos desde hace siglos, a partir de la creación de las primeras entidades de este tipo aparecidas en los primeros años del siglo XVII.

La institución Bolsa de valores intenta satisfacer tres grandes intereses:

- El de la empresa, porque al colocar sus acciones en el mercado y ser adquiridas por el público, obtiene de este el financiamiento necesario para cumplir sus fines y generar riqueza.

- El de los ahorradores, porque se convierten en inversionistas y en la medida de su participación obtienen beneficios por la vía de los dividendos que le reportan sus acciones.

- El del Estado, porque con la bolsa dispone de un medio para financiarse y hacer frente al gasto público, así como adelantar nuevas obras y programas de alcance social.

Los participantes que interactúan en la bolsa son los demandantes de capital (empresas, organismos públicos o privados y otras entidades), los oferentes de capital (ahorradores, inversionistas) y los intermediarios.

La negociación de valores en las bolsas se efectúa a través de los miembros de la bolsa, conocidos habitualmente con el nombre de brokers (terminología anglosajona), corredores, sociedades de corretaje de valores, casas de bolsa, agentes o comisionistas, de acuerdo a la denominación que reciben en las leyes de cada país, quienes hacen su labor a cambio de una comisión.

2.3.1. Características

Las características de la bolsa son las siguientes:

Rentabilidad Siempre que se invierte en bolsa se pretende obtener un rendimiento, y este se puede lograr de dos maneras: la primera es con el cobro de dividendos y la segunda con la diferencia entre el precio de venta y el de compra de los títulos.

Seguridad La bolsa es un mercado de renta variable, es decir, la cotización de los valores cambia a lo largo del tiempo, lo cual implica un riesgo. Para minimizar el riesgo es conveniente diversificar y no concentrar toda la inversión en un mismo valor de la bolsa.

Liquidez Facilidad de este tipo de inversiones de comprar y vender rápidamente.

2.3.2. Participantes

Los participantes que toman parte en las bolsas de valores son de tres clases:

Intermediarios Casas de bolsa, sociedades de corretaje, agencias de valores...

Inversionistas Arriesgan mucho buscando altas rentabilidades (inversionistas a corto plazo); buscan rentabilidad a través de dividendos y ampliaciones de capital (inversionistas a largo plazo); o invierten preferiblemente en valores de renta fija (inversionistas adversos al riesgo).

Empresas y estados Empresas, organismos públicos o privados y otros entes.

2.3.3. Función económica

Las bolsas de valores cumplen las siguientes funciones:

- Canalizan el ahorro hacia la inversión, favoreciendo el desarrollo económico.
- Ponen en contacto a las empresas y entidades del Estado necesitadas de recursos de inversión con los ahorradores.
- Confieren liquidez a la inversión, de manera que los poseedores de títulos pueden convertir en dinero sus acciones u otros valores con facilidad.

- Certifican precios de mercado.
- Favorecen una asignación eficiente de los recursos.
- Contribuyen a la valoración de activos financieros.

Por otra parte, las bolsas están sujetas a los riesgos de los ciclos económicos y sufren los efectos de los fenómenos psicológicos que pueden elevar o reducir los precios de los títulos y acciones.

2.4. El problema de la negociación de acciones

El problema de la negociación de acciones se define como sigue: elegidas las acciones de una empresa o entidad, maximizar la ganancia que se obtiene como consecuencia de las operaciones de compra y venta de estas acciones a lo largo del tiempo.

Maximizar la ganancia significa maximizar los beneficios de las posiciones largas y cortas que se toman en el mercado bursátil. Para calcular el beneficio potencial por cada acción negociada en una posición larga se procede como aparece en la ecuación 2.1, siendo $precio_{t_0}$ el precio al que la acción fue adquirida en el instante de tiempo t_0 y $precio_{t_n}$ el precio al que fue vendida en el instante t_n. Por su parte, para calcular el beneficio potencial por cada acción negociada en una posición corta se hace como indica la ecuación 2.2.

$$beneficio\ potencial\ por\ acción\ en\ posición\ larga = precio_{t_n} - precio_{t_0} \tag{2.1}$$

$$beneficio\ potencial\ por\ acción\ en\ posición\ corta = precio_{t_0} - precio_{t_n} \tag{2.2}$$

La dificultad del problema estriba en que para cada instante de tiempo sólo se conocen los precios pasados de la acción. En consecuencia, cada posición que se toma en el mercado refleja la esperanza del inversor porque las variaciones en el precio se produzcan en sentido favorable: esto es, suban al tomar una posición larga y bajen en posición corta.

Las ecuaciones anteriores son una simplificación para ilustrar la manera en que se calcula el beneficio (o pérdida) potencial por acción adquirida. Una fórmula más fiel a la realidad debe tener en cuenta el número de acciones que se negocian, los costes de operación impuestos por el intermediario y los intereses asociados al préstamo de acciones en el caso de una posición corta. Las ecuaciones 2.3 y 2.4 estiman estos resultados de forma más rigurosa para una negociación con un volumen de N acciones.

$$\begin{aligned} beneficio\ posición\ larga = \quad & N \cdot (precio_{t_n} - precio_{t_0}) \\ & -2 \cdot coste\ de\ operación \end{aligned} \tag{2.3}$$

$$\begin{aligned} beneficio\ posición\ corta = \quad & N \cdot (precio_{t_0} - precio_{t_n}) \\ & -2 \cdot coste\ de\ operación \\ & -intereses\ del\ préstamo \end{aligned} \tag{2.4}$$

Los datos primordiales en este problema son los precios de la acción en los instantes pasados. No obstante, los inversores emplean cualquier tipo de información relacionada (y no tan relacionada) con la que cuentan: indicadores macroeconómicos, sensaciones de los inversores en el mercado (miedo, incertidumbre, pesimismo, confianza, seguridad, optimismo), perspectivas futuras... Parte de la dificultad de este problema radica en seleccionar aquellos indicadores más representativos e importantes que intervienen en la estimación del precio futuro de la acción.

2.5. Maneras de enfrentar el problema

La obtención de un modelo preciso para resolver el problema de la negociación de acciones supondría unas compensaciones económicas muy elevadas. Dada la cuantiosa retribución, se han creado y estudiado una gran cantidad de modelos desde perspectivas muy variadas. Estos modelos no resuelven de forma explícita el problema de la negociación de acciones, sino que buscan solución a problemas equivalentes cuyos resultados se pueden aplicar de forma directa en la resolución del problema de partida. Uno de estos problemas equivalentes es el de la predicción de series temporales.

En esta sección se recogen algunos de los modelos surgidos en relación con el problema a resolver. El contenido se divide en dos bloques: el primero se centra en los modelos de predicción de series temporales y el segundo en las herramientas propias del campo de las finanzas.

2.5.1. Predicción de series temporales

Una serie temporal es una secuencia de datos, medidos normalmente en instantes de tiempo sucesivos espaciados entre sí de manera uniforme. Ejemplos de series temporales son el precio diario de cierre del índice de Dow Jones o el caudal anual del río Nilo en Aswan. El análisis de series temporales comprende métodos para analizar los datos de las series temporales con el fin de extraer estadísticas significativas y otras características de los valores. La predicción de series temporales consiste en el uso de un modelo para predecir los hechos futuros basándose en eventos conocidos del pasado: trata de predecir el valor futuro de la serie antes de la medición.

Análisis

Hay varios tipos de análisis de datos disponibles para las series temporales, cada uno de los cuales es apropiado para un propósito concreto:

Exploración general La exploración general comprende el examen gráfico de las series de datos, el análisis de la autocorrelación para estimar la dependencia de la serie y el análisis espectral para detectar el comportamiento cíclico relacionado con la estacionalidad.

Descripción Con la descripción se descompone la serie en sus partes constituyentes: la tendencia, el ciclo, la estacionalidad y la aleatoriedad.

Predicción y pronóstico Consiste en la utilización de los modelos estadísticos para la simulación estocástica, a fin de generar versiones alternativas de la serie de datos que representen los valores que esta puede tomar en el futuro. Esto es, emplear modelos estadísticos para describir el resultado probable de la serie de datos en el futuro inmediato, dados los resultados más recientes y pasados.

Modelos

Los modelos de series temporales de datos pueden tener muchas formas con las que representar a los procesos estocásticos. Las tres clases de modelos según importancia práctica son el modelo autorregresivo (AR), el modelo integrado (I) y el modelo de la media móvil (MA). Estas tres clases modelan la serie temporal con una función lineal aplicada a los datos previos. Con la combinación de estos modelos se producen el modelo autorregresivo de media móvil (ARMA) [1] y el modelo autorregresivo integrado de media móvil (ARIMA) [1]. El modelo autorregresivo fraccional integrado de media móvil (ARFIMA) es una generalización de todos los anteriores [2, 3, 4]. Así mismo, hay extensiones de estos modelos que manipulan series de vectores, que se conocen como series temporales multivariable. Otro tipo de extensiones se basan en considerar una serie temporal externa que condiciona el resultado de la serie temporal modelada.

Además de los modelos lineales están aquellos que contemplan relaciones no lineales en los valores de la serie respecto a sus anteriores. Con estos modelos se pueden generar series temporales caóticas. Más importante aún, los estudios sugieren que las predicciones de los modelos no lineales son superiores a la de los lineales.

Algunos de estos modelos no lineales se basan en representar el cambio de la varianza a lo largo del tiempo (heterocedasticidad). Estos son los modelos de heterocedasticidad condicional autorregresivos (ARCH), que cuentan con una amplia variedad de representaciones (GARCH, TARCH, EGARCH, FIGARCH, CGARCH...) [5]. Aquí, los cambios en las varianza se relacionan o predicen a partir de los valores pasados observados en la serie.

Otras herramientas que se han mostrado especialmente útiles en el modelado y predicción de series temporales son las redes neuronales [6, 7, 8], las redes Bayesianas [9] y el modelo oculto de Márkov [10].

2.5.2. Herramientas del campo de las finanzas

En el pasado, las decisiones de inversión se tomaban sin mucho fundamento, sólo justificadas por las intuiciones y sensaciones del inversor. Las transacciones eran cada vez de mayor cuantía y el riesgo asumido más alto. Se urgía la creación de herramientas que impusieran rigor en la toma de decisiones. Las aportaciones realizadas en este sentido sobre el campo de las finanzas han sido múltiples. Estos esfuerzos se ven justificados por las promesas de suculentas ganancias.

Se parte del análisis bursátil, que tiene como objetivo el estudio del comportamiento de los mercados financieros y de los valores que lo constituyen. Con este análisis se pretende conseguir información relevante que ayude en las decisiones de inversión en situaciones de incertidumbre. Se divide en una doble categoría de análisis, cada una de las cuales parte de suposiciones diferentes a la hora de interpretar el comportamiento de los mercados financieros. Estas dos categorías son:

Análisis fundamental

En el análisis fundamental se analizan aspectos clave de una empresa para evaluar su situación: estado y salud financiera, forma de gestión, funcionamiento, ventajas competitivas, competidores y mercado en que opera. El objetivo de este tipo de análisis es conocer el auténtico valor del título o acción, llamado valor fundamental. Un aspecto negativo de este tipo de análisis es que los resultados que arroja están condicionados por la apreciación del analista y por tanto son subjetivos.

Este tipo de análisis fue introducido por Benjamin Graham y David Dodd [11].

Análisis técnico

El análisis técnico consiste en el estudio de la acción del mercado, principalmente a través del uso de gráficas, con el propósito de predecir las tendencias en el precio.

La acción del mercado queda reflejada por las tres fuentes principales de información disponibles para el analista técnico: el precio o cotización, el volumen de transacciones y el interés abierto (representa el número de contratos que permanecen abiertos al cierre del periodo).

El análisis técnico tuvo sus orígenes en la teoría de Dow, creada por Charles H. Dow a finales del siglo XIX. Adquirió gran impulso con la teoría de las ondas de Elliot propuesta por Ralph N. Elliott [12]. Los principios y herramientas que proporciona este tipo de análisis son aplicables al estudio de cualquier instrumento financiero.

Entre las fortalezas del análisis técnico está el hecho de que se puede adaptar a cualquier instrumento financiero y dimensión de tiempo. No hay ninguna área dentro de los mercados financieros en la que los principios del análisis técnico no sean aplicables. Se pueden seguir simultáneamente tantos mercados como se desee, aplicando los mismos principios con adaptaciones menores de acuerdo al comportamiento de cada uno.

La principal virtud del análisis técnico reside en la utilización de fórmulas bien definidas que excluyen la subjetividad en los estudios. Gracias a esta característica se puede ejecutar este tipo de análisis sobre máquinas.

Predictibilidad

Existe recelo en la comunidad académica respecto a las capacidades de predicción del análisis funcional y el análisis técnico.

El análisis fundamental parece bien sostenido en la teoría, y muchos analistas lo usan en la actualidad. Sin embargo, los resultados arrojados por la comunidad académica concluyen que el análisis fundamental no es mejor que el análisis técnico a la hora de predecir el futuro de las acciones [13].

En lo que al análisis técnico se refiere, los estudios académicos sugieren que el poder de predicción que tiene es limitado [13]. Los economistas Arnold Moore y Eugene F. Fama lograron determinar que sólo alrededor del 3 % de la variación diaria en el precio puede ser explicada a partir de los precios pasados de la acción. Es por eso que la información contenida en el histórico del precio pasado de un determinado título es insuficiente para predecir el precio futuro [14, 15, 16]. Burton G. Malkiel, inspirador de la teoría del paseo aleatorio, es otro especialista que examinó el éxito predictivo del análisis técnico [17].

Otros estudios sugieren que hay posibilidades de obtener beneficios por encima del promedio. Por ejemplo, la predicción no lineal usando redes neuronales ha demostrado obtener resultados estadisticamente significativos [18].

Existen varias hipótesis que explicarían las dificultades del análisis técnico y otro tipo de análisis bursátiles para predecir las cotizaciones futuras:

- Una de ellas tiene que ver con la hipótesis de los mercados eficientes, según la cual toda la información disponible influye rápidamente en la cotización de un determinado título. Esto haría que fuera imposible aprovechar las subvaloraciones o sobrevaloraciones del precio.

- Otro tipo de evidencia fue encontrada por Gary Belsky y Thomas Gilovich. Estos señalan que los mercados tienen periodos de paseo aleatorio en que básicamente no pasa nada determinate, interrumpidos por euforias y pánicos abruptos. En sus estudios consideraron cotizaciones desde 1963 a 1993, lo que incluía 7802 días de negocio, en los que la rentabilidad media anual fue algo inferior al 12 %. Si se descontaban los 40 mejores días, entonces la rentabilidad media anual sólo era ligeramente superior al 7 %. Esto significa que el 0,5 % de las sesiones son responsables de casi el 5 % de la rentabilidad, siendo el 99,5 % restante responsable de sólo el 7 %. Dado el caracter extremadamente abrupto de las euforias y los pánicos, estos autores sugieren que difícilmente las técnicas de pronóstico actuales puedan ser demasiado útiles [19].

Capítulo 3

Plataforma de inversión bursátil

Este trabajo parte del desarrollo de una plataforma de creación de estrategias de trading. La plataforma automatiza todo el proceso relativo a la creación de estrategias de trading en el mercado de acciones: adquisición de información desde los proveedores de datos, normalización de la información de entrada, creación y simulación de estrategias, optimización de parámetros y generación de resultados. Su necesidad se fundamentó en el vacío existencial de entornos adecuados para tal propósito. Previo al desarrollo de la plataforma se consideraron dos alternativas:

- Utilizar una aplicación de inversión bursátil que permita crear estrategias. Analizadas las opciones que había en el mercado, resultó que los mecanismos que ofrecían para implementar estrategias eran toscos: sistemas basados en reglas, lenguajes propietarios y otras alternativas igual de limitadas. La falta de expresividad de estas herramientas hizo que se descartaran.

- Hacer uso de alguna API para interactuar con una aplicación de inversión bursátil. Si es posible interactuar con una aplicación de inversión pero desde fuera de ella, entonces se evitan las limitaciones que impone. El problema en este caso es que las APIs no están pensadas para crear estrategias y las que sí lo permiten no son muy flexibles. Una cuestión adicional es el elevado coste de estas interfaces.

Dado que ninguna de las dos opciones era adecuada, se optó por desarrollar la plataforma.

Los objetivos a alcanzar durante la creación de la plataforma fueron los de conseguir un producto específico en funcionalidades y flexible en tanto a las posibilidades de extensión. Para satisfacer estas dos premisas se planteó una arquitectura que, a través de un proceso de refinamiento en sus sucesivas versiones, dio lugar a un resultado que favorecía el encapsulamiento, la coherencia, la cohesión y la genericidad. Alcanzar esta madurez en la arquitectura requirió de una versión prototipo, dos versiones mayores y múltiples versiones menores. Una cuestión crítica, resuelta durante esta etapa, fue la de definir puntos de extensión en la plataforma. Los puntos de extensión son un mecanismo para personalizar y extender el funcionamiento de la plataforma sin necesidad de modificar ningún aspecto interno. A lo largo del documento se ilustran casos en que, a través de los puntos de extensión, se logra el comportamiento deseado.

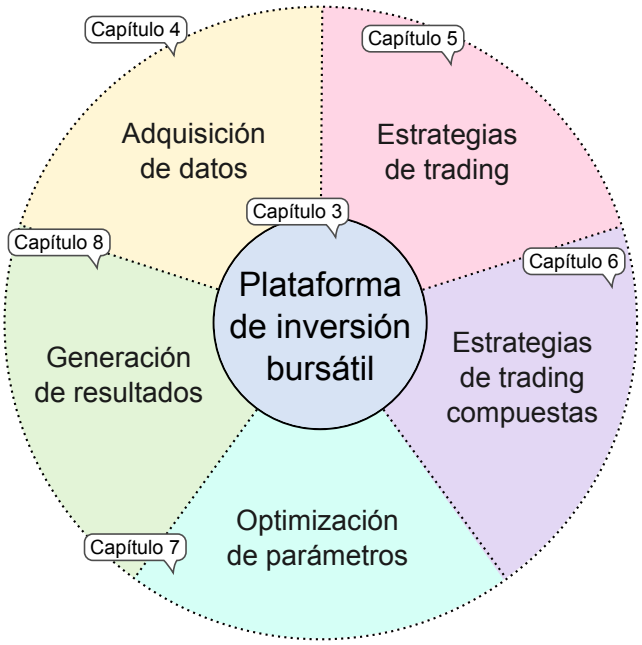

Figura 3.1: Bloques funcionales construidos sobre la plataforma de inversión bursátil

En el aspecto técnico, la plataforma está constituida por una serie de bloques funcionales que interactúan entre sí. En la figura 3.1 se muestran de forma esquemática los bloques funcionales de la plataforma. Para cada uno de ellos se indica el capítulo del documento en que se explican.

Dentro de la plataforma, la clase en torno a la cual sucede todo es `AlgoTrader`, que implementa los algoritmos de trading, algoritmos automaticos de trading, estrategias de trading o cualquier otro término equivalente con que se quiera designar al concepto. Las funcionalidades relativas a la optimización de parámetros y generación de estadísticas están implementadas como métodos en esta clase. El enfoque dado para el mecanismo de optimización de parámetros es súmamente potente y depende de funciones externas, si bien esos detalles se revelan en la secciones pertinentes. Por su parte, `DataSerie` es la clase encargada de la obtención, normalización, manipulación y representación de la información de los proveedores de datos. A través de estas dos clases la plataforma ofrece un entorno homogéneo, intuitivo, fácil de utilizar, potente y con un alto nivel de productividad.

La implementación de la plataforma se ha realizado casi en su totalidad con MATLAB, utilizando el paradigma de programación orientada a objetos. Las bondades de este paradigma permiten una implementación elegante de la plataforma para que cumpla con todos los requisitos deseables. Además de un lenguaje de programación, MATLAB es un entorno de computación numérica ámpliamente utilizado en ciencia, ingeniería y economía. Como característica adicional, MATLAB ofrece primitivas para la ejecución de código en paralelo: estas primitivas han sido utilizadas en la implementación de la plataforma. Donde más beneficio se obtiene del uso del paralelismo es en las funciones de optimización, que son las más intensivas en términos computacionales.

En el resto del capítulo se hace una presentación de los elementos de la plataforma y se describe la manera en que interactúan para dar vida a la aplicación. La sección 3.1 explica en qué consiste el proceso de adquisición de datos y cómo la clase `DataSerie` facilita la tarea de incorporar proveedores de información bursátil. En la sección 3.2 se describe la clase `AlgoTrader`, que es la parte central de la plataforma; también se indica cómo funcionan las estrategias de trading y de qué manera esta clase ofrece una solución versátil para implementarlas. La sección 3.3 introduce el enfoque innovador con que se incorporan las técnicas de optimización para calcular los parámetros óptimos de las estrategias. La generación de resultados se trata en la sección 3.4 y es parte fundamental para el análisis de las máquinas. Por último, en la sección 3.5, se comenta sobre el archivo `Settings.m` que almacena configuraciones y valores por defecto.

3.1. Adquisición de datos

Dado que el problema trata sobre generar estrategias de trading, entonces los datos de partida son todos los relacionados con las acciones de los mercados bursátiles. Contar con datos reales supone una ventaja considerable que aporta realismo al trabajo realizado, sin embargo, no es una situación que se produzca en todos los casos, a consecuencia de la complejidad que puede existir para recabarlos. Estas carencias, en caso de producirse, intentan ser suplidas por medio de la utilización de sistemas que generan datos lo más fieles posibles a la situación que modelan. Como aclaración, en el trabajo se han utilizado en todo momento datos reales obtenidos desde los proveedores.

¿Qué información sobre las acciones se necesita? Respecto a esta pregunta, hay que hacer primero la distinción entre información básica y derivada. En los mercados bursátiles se comercia con acciones; las operaciones de comercio con cada una de estas acciones da lugar a un registro de precios y volúmenes de transacción a lo largo del tiempo, que constituye la información básica. Por su parte, la información derivada se origina a partir de la información básica, a través de la aplicación de fórmulas y modelos matemáticos. La información derivada permite estudiar, resumir y evidenciar ciertas características de la información básica. La adquisición de datos consiste en obtener la información básica de la acción en forma de series temporales[1] de precios y volúmenes con que se comerció en el mercado de acciones.

Se requieren varias series temporales sobre una acción para ofrecer información precisa. En una serie temporal, cada valor de la misma se corresponde con una medición realizada a una variable en un instante de tiempo. Si el espacio temporal que se utiliza entre mediciones sucesivas es grande, entonces se pierde gran cantidad de información. Así, por ejemplo, en el intervalo de tiempo entre mediciones es posible que el valor de la variable comience con una medición, alcance máximos, mínimos y termine en otra medición distinta, ¿qué valor se toma entonces? Este problema se agudiza si la variable observada es de naturaleza cambiante. Para reducir la falta de representatividad de una única serie temporal se utilizan varias series

[1] Una serie temporal es una secuencia de valores medidos en el tiempo, ordenados de forma cronológica y habitualmente espaciados entre sí de manera uniforme.

temporales referidas a los precios de apertura, más alto, más bajo y de cierre de la acción. También se considera la serie temporal relativa al volumen de transacciones de la acción.

La información sobre las acciones se consigue por medio de los proveedores de datos. Estos proveedores son empresas o instituciones que tienen conexión directa con el mercado bursátil y recogen entre otros los registros de información sobre los precios y volúmenes de las acciones. Posteriormente, se encargan de difundir la información por diferentes canales de comunicación, incluido Internet. En el capítulo 4 se explican los tres proveedores de información bursátil incorporados en la plataforma.

3.1.1. La clase `DataSerie`

La clase abstracta `DataSerie` es la manera en que la plataforma desarrollada contempla el uso de información relativa a las acciones; puede entenderse como un contenedor donde se vuelca la información obtenida desde un proveedor de datos. Además de almacenar información, es capaz de representarla gráficamente. Aunque integrada en la plataforma, la clase es independiente y se puede utilizar para cualquier otro propósito.

Esta clase define las estructuras de datos necesarias para almacenar la información de las acciones. Para el caso de las series temporales, hay que notar que consisten en secuencias de valores referenciados por los instantes de tiempo en que fueron tomados. En consecuencia, son necesarios dos vectores para almacenar toda la información relativa a una serie: un vector para los valores de la serie y otro vector para los instantes de tiempo. La información que hay que guardar de una acción es la siguiente: identificador de la acción, tipo y unidades de compresión temporal, vector de instantes temporales, vector de precios de apertura, vector de precios más altos, vector de precios más bajos, vector de precios de cierre y vector de volumenes de transacción. El tipo y unidades de compresión temporal determinan la frecuencia con que se toman los datos: un minuto, cinco días, dos meses... Los detalles de implementación sobre las estructuras de datos se indican en la sección A.1.

Respecto a sus capacidades, la clase `DataSerie` ofrece gráficos elaborados de la información que contiene. Con las invocaciones a los métodos pertinentes, se generan las representaciones visuales de la información almacenada en el objeto. Las series temporales que representan las cotizaciones de la acción se muestran en forma de curva sobre el plano, donde el eje x o de abscisa representa el tiempo y el eje y o de ordenada el valor de la cotización. Por su parte, el volumen de transacciones, también contenido en una serie temporal, se muestra en forma de gráfico de barras. Los métodos de dibujado así como las demás funcionalidades provistas por la clase `DataSerie` se explican en la sección A.2.

¿Cómo incorporar proveedores de datos en la plataforma?

Ya se ha comentado suficiente sobre la clase `DataSerie` como para dar una idea del rol que cumple dentro de la plataforma y las capacidades que tiene. Esta clase implementa todos los métodos necesarios para gestionar la información que almacena sobre la acción bursátil, aunque hay uno en concreto, el metodo abstracto `get`, cuya implementación delega al progra-

mador. Para incorporar un nuevo proveedor de datos hay que crear una clase que herede de `DataSerie` e implemente el método `get`. Al contener un método abstracto, la clase `DataSerie` es abstracta, por lo que no se pueden crear instancias a partir de ella. La función `get` y sus argumentos de entrada y salida se explican en la sección A.2.12. En esencia, esta función toma como argumentos de entrada el identificador de la acción, el tipo y unidades de compresión temporal y dos fechas para el intervalo de tiempo del que solicita la información; como resultado, devuelve seis vectores columnas de igual longitud con la información de los instantes temporales, las cotizaciones de apertura, más alta, más baja y cierre junto al volumen de transacciones. La signatura de la función `get` es la siguiente:

```
[dateTime, open, high, low, close, volume]
= get(symbolCode,compressionType,compressionUnits,initDateTime,endDateTime)
```

Organización de la documentación sobre la clase `DataSerie`

Los detalles sobre la clase `DataSerie` se encuentran repartidos a lo largo del documento. En la sección 3.1 se introduce la clase, mencionando qué información de la acción almacena. También se comentan las capacidades de representación gráfica que tiene. Por ultimo, se explica cómo hacer para incorporar nuevos proveedores con la implementación del método abstracto `get`.

Si bien queda introducida la metodología para incorporar nuevos proveedores, no es hasta el capítulo 4 donde se explican los proveedores de datos incluidos en la plataforma. Cada uno de ellos se accede de forma distinta y la manera en que se obtiene la información es bien diferente; sin embargo, todos los proveedores encajan de forma adecuada en la clase `DataSerie`, que permite un tratamiento homogeneo.

Para comprender las características técnicas de la clase `DataSerie` es necesario apoyarse en el apéndice A. Aquí aparece un listado de las propiedades o atributos que componen la estructura de datos y los métodos que implementa. También explica el sistema de dibujado con el que se confeccionan los gráficos. Los ejemplos de uso de la clase se encuentran en el apéndice C.

3.2. Máquinas de trading

Las máquinas de trading son programas que implementan estrategias de trading; además, ofrecen mecanismos para manipular estas estrategias y obtener información de su desempeño.

Como punto de partida, la máquina de trading tiene acceso a toda la información sobre una acción bursátil: estos datos son proporcionados por un objeto de la clase `DataSerie`. Después de procesar la entrada según su lógica interna, la máquina produce la salida, que consiste en las señales de compra y venta de acciones. La salida de la máquina se codifica como un vector de posiciones que indica, para cada instante, la posición que tiene la máquina respecto al mercado: posición larga si ha comprado, corta si ha vendido o sin posición si está fuera del mercado.

Una máquina de trading puede interpretarse como un agente inteligente que percibe estímulos del entorno (cotizaciones de la acción bursátil), procesa tales percepciones y actúa (decisiones de compra y venta de acciones bursátiles) de manera racional, es decir, de manera correcta y tendiendo a optimizar el resultado esperado.

Muchas de las estrategias de trading de la actualidad se basan en los modelos matemáticos comentados en la sección 2.5. Estas estrategias y sus modelos constituyen el nucleo de las máquina de trading. En los capítulos 5 y 6 se explican algunas de las máquinas desarrolladas sobre la plataforma.

3.2.1. La clase `AlgoTrader`

Las máquinas de trading se implementan a partir de la clase abstracta `AlgoTrader`. Consiste en una clase que aporta todos los métodos necesarios para inicializar, simular, optimizar, analizar y visualizar las estrategias de trading: puede entenderse como un entorno donde se implementan las estrategias que se quieren estudiar.

La clase `AlgoTrader` define las estructuras de datos necesarias con las que ofrecer una base a partir de la cual implementar las máquinas de trading. Cuenta con propiedades o atributos relevantes para configurar las estrategias: un apuntador hacia un objeto de la clase `DataSerie` para conseguir los datos de partida, un vector donde almacenar la salida generada por la estrategia, los índices de comienzo y fin de los conjuntos de entrenamiento y test, el capital inicial que se invierte, el tope por inversión... Para un listado completo de las estructuras de datos y propiedades de la clase `AlgoTrader` se ha de consultar la sección B.1.

Cada vez que cambia alguno de los parámetros de la clase `AlgoTrader`, es necesario actualizar el estado interno de la máquina de trading. La actualización puede ser un proceso muy costoso, así que se ha implementado un sistema de actualización perezosa que computa el nuevo estado sólo en caso de que sea necesario. Para lograr este comportamiento se utilizan eventos que, ante la modificación de parámetros de la estrategia, marcan el estado del objeto como no actualizado. El objeto mantiene su estado de no actualizado hasta el instante en que se necesitan los resultados; en ese momento, el estado del objeto se actualiza y se devuelven los resultados. La ventaja de hacerlo así está en que si se cambian varios parámetros de la estategia no se actualiza el estado con cada parámetro alterado, sino que se espera hasta cuando se requieren los resultados. Este sistema de actualización perezosa es transparente al usuario, que sólo necesita indicar qué parámetros de la estrategia cambian el valor de la señal de posiciones para que sean tenidos en cuenta.

Una máquina de trading tiene que poner a disposición del usuario la información que pueda necesitar, después de todo, esta es una herramienta destinada al estudio y el análisis. El grueso la clase lo constituyen los métodos que manipulan la estrategia y realizan cálculos para extraer información sobre su desempeño. Para cuestiones muy específicas sobre las máquinas de trading que no se encuentren implementadas, está la posibilidad de extender la clase e implementar los métodos necesarios o redefinir alguno de los ya existentes. Los métodos de la clase están documentados en la sección B.2.

La información gráfica es de vital importancia, pues condensa una gran cantidad de información y la muestra de forma conveniente. La clase `AlgoTrader` implementa muchos métodos que ofrecen diferentes vistas sobre las estrategias de trading. Supone un gran apoyo el contar con elementos visuales descriptivos: así se facilita más aún el estudio de las estrategias. El mecanismo para representar gráficamente las máquinas de trading es idéntico al que utiliza la clase `DataSerie`. Este sistema de generación de gráficos consiste en un conjunto de métodos desacoplados entre sí que colaboran dedicándose a implementar partes concretas de los gráficos. La ventaja de este sistema de dibujado es que es fácil de extender y mantener por medio del mecanismo de herencia de clases. Los métodos del sistema de dibujado se explican en la sección B.3.

¿Cómo incorporar máquinas de trading en la plataforma?

Para implementar máquinas de trading en la plataforma hay que heredar de la clase abstracta `AlgoTrader` e implementar su único método abstracto `computeSignal`. Este método no toma ni devuelve ningún argumento[2] y se encarga de aplicar la estrategia para producir las señales de posición en el mercado. El método `computeSignal` obtiene la información de partida del objeto y almacena el resultado también en el objeto: actualiza su estado interno. En la sección B.2.3 se encuentra una descripción detallada del método abstracto `computeSignal`.

Organización de la documentación sobre la clase `AlgoTrader`

La documentación referente a la clase `AlgoTrader` está distribuida en todo el documento. En la sección 3.2 se comenta sobre las máquinas de trading explicando qué son, qué datos de partida requieren y qué resultados producen. Las características de estas máquinas las asemejan a agentes inteligentes que interactúan con su entorno. También se trata el sobre la inclusión de nuevas máquinas de trading a través del mecanismo de herencia y la implementación del método abstracto `computeSignal`.

En los capítulos 5 y 6 se explican las máquinas de trading desarrolladas en el proyecto. Estas máquinas utilizan estrategias basadas en modelos matemáticos que, aunque diferentes, tienen cabida en la plataforma. También se describe cómo utilizar los puntos de extensión para alterar el funcionamiento de las máquinas y lograr en cada caso el comportamiento deseado.

La descripción técnica de los elementos de la clase `AlgoTrader` se hace en el apéndice B. Ahí aparece un listado de las propiedades o atributos que configuran la estructura de datos y los métodos que implementa. También se explica el sistema de dibujado que da lugar a los gráficos que genera la clase. Por último, en el apéndice C aparecen los ejemplos de uso de la clase.

[2] En MATLAB cuando se utiliza el paradigma de programación orientada a objetos los métodos de una clase tienen que declarar de forma obligatoria como primer argumento de entrada una instancia de esa misma clase. Esto es así porque MATLAB permite invocar los métodos de dos formas diferentes: con el separador punto entre la variable del objeto y el nombre del método como por ejemplo `objeto.método(...)` o con la sintaxis habitual de las funciones como en `método(objeto, ...)`.

3.3. Optimización de parámetros

Las estrategias de trading se apoyan en modelos matemáticos que utilizan parámetros. La experiencia capacita para seleccionar valores adecuados de los parámetros con la esperanza de mejorar los resultados, no obstante, este procedimiento está sólo al alcance de los expertos y ni siquiera así se garantizan resultados cercanos al óptimo. ¿Cómo hacer entonces para calcular los valores óptimos? Los modelos matemáticos en que se basan las estrategias de trading pueden ser simulados en la plataforma para cuantificar su desempeño; de esta forma, es posible comparar los resultados obtenidos en las diferentes asignaciones y atribuir a los parámetros los valores que proporcionan el mejor resultado.

Desde el punto de vista matemático, una estrategia de trading puede concebirse como una función g, con los mismos parámetros de entrada a que la estrategia y que devuelve como resultado las decisiones d sobre cómo invertir. Así mismo, es necesario emplear una segunda función f que, dadas las decisiones d, calcula el desempeño p de la estrategia; tal evaluación la hace sobre el conjunto de datos de entrenamiento utilizando algún criterio: beneficio obtenido, riesgo asumido, número de operaciones exitosas... La utilidad de la función f radica en que permite cambiar el criterio de evaluación aplicado a la estrategia. La ecuación 3.1 muestra cómo se hace el cálculo del desempeño de la estrategia.

$$f(g(a)) = p \tag{3.1}$$

A partir de las definiciones anteriores, el problema de la asignación de valores para los parámetros se puede tratar como un problema de optimización. Como indica la ecuación 3.2, en un problema de optimización el objetivo es seleccionar el conjunto de valores a^*, dentro del espacio de búsqueda A, tales que ningún otro conjunto de valores produzca mejores resultados. En la ecuación se utiliza el operador $\succeq$, que compara valores expresados en las mismas medidas e indica que el valor de la izquierda es mejor que el de la derecha. El término *mejor* se refiere indistintamente al mínimo en caso de minimizar o al máximo en caso de maximizar y depende del criterio considerado.

$$f(g(a^*)) \succeq f(g(a)) \quad , \forall a \in A \tag{3.2}$$

Para calcular el conjunto de valores óptimos de los parámetros se utilizan los métodos de optimización. En el contexto de este trabajo, un método de optimización m consiste en un procedimiento que, a partir de la estrategia de trading g, la función de selección s, la función de evaluación f y el espacio de búsqueda A, calcula el conjunto de valores óptimos a^* que mejoran el resultado de la estrategia de trading sobre el conjunto de datos de entrenamiento. La función de selección s puede ser `máximo` o `mínimo` según se busque maximizar o minimizar el valor devuelto por la función de evaluación f. Todo lo anterior se resume en la ecuación 3.3, que descompone el problema de la optimización de parámetros en partes bien definidas.

$$m(g, s, f, A) = a^* \tag{3.3}$$

La descomposición del problema en las partes mencionadas es crucial para lograr un sistema extensible y escalable. Así, por ejemplo, en caso de implementar un número M, G y F de métodos de optimización, estrategias de trading y funciones de evaluación respectivamente, las dos alternativas extremas son:

Solución empotrada En este caso, la solución consiste en un código monolítico que implementa uno de los métodos de optimización para una de las estrategias de trading y ante una función de evaluación. Como resultado, y para cubrir todas las combinaciones posibles, es necesario implementar $M \cdot G \cdot F$ procedimientos diferentes. Sobra decir que este enfoque es inviable en una plataforma escalable.

Solución modular En la solución modular cada componente se implementa por separado, así se evita la explosión combinatoria de procedimientos que es preciso implementar que, en esta situación, es de $M + G + F$. Luego, en tiempo de ejecución, se combinan los módulos para que trabajen en conjunto. Esta es la opción seguida en la plataforma.

3.3.1. El método `optimize`

El cálculo de los valores óptimos de los parámetros lo realiza el método `optimize` de la clase `AlgoTrader`. Tal método es el responsable de lanzar el método de optimización con la función de selección, la función de evaluación y el espacio de búsqueda. La forma que tiene la instrucción de llamada al método `optimize` es la mostrada en el código siguiente:

```
algoTraderInstance.optimize(@evaluationFunction, ...
                        @selectionFunction, ...
                        @optimizationMethod, ...
                        'Parameter1', Domain1, ...
                        'Parameter2', Domain2, ...
                        ...
                        'ParameterN', DomainN)
```

Los elementos que intervienen en la llamada son los mismos que aparecen en la ecuación 3.3, aunque el orden es diferente. La máquina de trading `algoTraderInstance` (g en las ecuaciones anteriores) es una instancia de clase y por eso se utiliza la notación punto. Los tres primeros parámetros que son `evaluationFunction` para la función de evaluación (f en las ecuaciones anteriores), `selectionFunction` para la función de selección (s en las ecuaciones anteriores) y `optimizationMethod` para el método de optimización (m en las ecuaciones anteriores) utilizan el prefijo `@` en la instrucción, pues es la manera en que MATLAB referencia funciones. Los pares `Parameter*` y `Domain*` sirven para indicar el nombre de los parámetros que se someten al proceso de optimización y el conjunto de valores en donde se buscan los óptimos (se corresponden con el espacio de búsqueda A en las ecuaciones anteriores)[3]. El nombre del parámetro consiste en una cadena de texto que identifica una propiedad de la máquina

[3]El espacio de búsqueda consiste en todas las combinaciones de valores de los parámetros que se pueden generar. Si, por ejemplo, se quieren optimizar los parámetros *ParametroA*, *ParametroB* y *ParametroC*

de trading. Respecto al conjunto de valores en donde se busca el óptimo, se especifica como un array de tipo numérico (notación `[]` en MATLAB) o como un array de celdas (admite números y cadenas de texto, notación `{}` en MATLAB), según sean los valores que se utilicen.

Como resultado de una llamada al método `optimize`, se actualizan los parámetros especificados de la estrategia, tomando estos los mejores valores que haya sido capaz de encontrar el método de optimización. En tanto a los métodos de optimización, no están obligados a encontrar el conjunto de valores óptimos para los parámetros, aunque sí un conjunto de valores adecuados. Para su funcionamiento, el método de optimización requiere de la función `evaluationFunction` que evalúa el desempeño de la estrategia sobre el conjunto de datos de entrenamiento: es en esta fase donde se utiliza el simulador de estrategias implementado en la plataforma. Una descripción técnica del método `optimize` se encuentra en la sección B.2.13. Los ejemplos de uso del método aparecen en el apéndice C.

¿Cómo incorporar funciones de evaluación de estrategias en la plataforma?

Una función de evaluación mide el desempeño de una estrategia de trading atendiendo a algún criterio: beneficio obtenido, riesgo asumido, número de operaciones exitosas... La función de evaluación parte de las decisiones tomadas por la estrategia, que consisten en una serie de registros que describen cada una de las posiciones adoptadas en el mercado. Estos registros están compuestos por los siguientes campos:

Tipo de posición Se refiere al tipo de operación con que se posiciona la estrategia en el mercado: puede ser una operación de compra (identificada con el valor 1) o de venta (-1). También se contempla el caso en que la estrategia permanece fuera del mercado (en cuyo caso se identifica la no posición con el valor 0).

Índices de apertura y cierre Índices a partir de los cuales se abre y cierra la posición en el mercado. Estos índices son relativos a la serie de precios de `DataSerie`.

Fechas de apertura y cierre Fechas correspondientes a la apertura y cierre de la operación de mercado. Se calculan a partir de los índices de apertura y cierre sobre la propiedad `DateTime` de la clase `DataSerie`. El formato en que está codificada la fecha es aquel utilizado por la propiedad `DateTime`.

Precios de apertura y cierre Precios en los instantes de tiempo en que se abre y cierra la posición en el mercado. Son el resultado de acceder a las posiciones indicadas por los índices de apertura y cierre sobre la propiedad `Serie` de la clase `DateTime`.

Beneficio/Pérdida Factor que multiplica a los fondos con que cuenta la estrategia para calcular los nuevos fondos tras la operación bursátil.

---

cuyos conjuntos de valores permitidos son $A = \{a_1, a_2, ..., a_i\}$, $B = \{b_1, b_2, ..., b_j\}$ y $C = \{c_1, c_2, ..., c_k\}$ respectivamente, entonces el espacio de búsqueda generado es $E = A \times B \times C$. Este espacio E tiene tres dimensiones y el número de estados que contiene es de $i \cdot j \cdot k$. Cualquier terna (a_r, b_s, c_t) con $r \in \{1, ..., i\}$, $s \in \{1, ..., j\}$ y $t \in \{1, ..., k\}$ es un elemento del espacio de búsqueda E.

Tipo	Índice$_1$	Índice$_N$	Fecha$_1$	Fecha$_N$	Precio$_1$	Precio$_N$	Beneficio/Pérdida
1	58	80	731979	732133	9.24	8.50	0.9199
-1	80	112	732133	732357	8.50	9.57	0.8882
0	112	120	732357	732413	9.57	9.16	1
1	120	133	732413	732504	9.16	10.70	1.1681
-1	133	147	732504	732602	10.70	9.37	1.1419
1	147	161	732602	732700	9.37	9.89	1.0555
0	161	169	732700	732756	9.89	9.53	1

Tabla 3.1: Tabla de registros de posición: contiene la información de las posiciones adoptadas por la máquina de trading en el mercado

Los registros se empaquetan en forma de tabla para ser pasados como argumento a la función de evaluación de las máquinas de trading. Las filas de la tabla se corresponden con los registros y se ordenan de forma cronológica; las columnas se refieren a los campos que componen los registros: tipo de posición, índice de apertura, índice de cierre, fecha de apertura, fecha de cierre, precio de apertura, precio de cierre y beneficio/pérdida. En la tabla 3.1 se muestra la estructura de los datos relativos a las posiciones adoptadas en el mercado. A partir de esta información, la función de evaluación calcula un valor numérico que se corresponde con una medición del rendimiento de la estrategia de acuerdo a cierto criterio. La función de evaluación debe respetar la siguiente signatura:

```
performance = evaluationFunction(positionsLog, ...)
```

En este caso, la función `evaluationFunction` toma como primer parámetro la tabla de datos, pudiendo aceptar además una cantidad adicional de argumentos para realizar la computación. Esta función no la invoca el usuario de forma directa, sino que está pensada para ser llamada desde el método `optimize`, el método `fitness` y el método `fitnessStatistics`. Como el usuario no realiza la llamada, entonces los parámetros adicionales que se desean enviar a la función `evaluationFunction` han de ser indicados en el método `optimize` (lo mismo es aplicable sobre los otros dos métodos). Por ejemplo, si se desean incluir los parámetros de entrada `Arg1` hasta `ArgN` en la función de evaluación, entonces es necesario sustituir en la llamada al método `optimize` el argumento correspondiente a la función de evaluación `@evaluationFunction` por una construcción más expresiva, como es `{@evaluationFunction, Arg1, ..., ArgN}`. De esta forma, se puede pasar una cantidad arbitraria de parámetros a la función de evaluación. Como resultado de la computación, la función devuelve un argumento `performance` con el valor de la medición asociada al criterio implementado.

Un ejemplo de función de evaluación consiste en aquella que mide el beneficio o pérdida de una estrategia, que se calcula multiplicando los valores de la columna de beneficio/pérdida de la tabla. Las funciones de evaluación implementadas en la plataforma se detallan en la sección 7.2.

¿Cómo incorporar métodos de optimización en la plataforma?

En lo que a este trabajo se refiere, un método de optimización consiste en un procedimiento que calcula valores para los parámetros de las máquinas de trading, con el objetivo de mejorar su desempeño. No es necesario que el método de optimización garantice el alcanzar valores óptimos para los parámetros, pues la condición de optimalidad en los resultados es demasiado fuerte e incompatible con muchos de ellos.

Para incorporar un método de optimización en la plataforma, el único requisito consiste en respetar el siquiente esquema en la signatura de la función:

```
bestIndexArray = optimizationMethod(@selectionFunction,
                     @fitnessFunction,
                     searchSpaceSize, ...)
```

Se aprecia que la función que implementa el método acepta un mínimo de tres argumentos: `selectionFunction` para la función de selección, `fitnessFunction` para la función de aptitud y `searchSpaceSize` para especificar las dimensiones y tamaño del espacio de búsqueda. Es posible declarar argumentos de entrada adicionales para utilizar con el método. La función devuelve en el vector `bestIndexArray` el mejor resultado hallado por el método de optimización. Al igual que en el caso de la función de evaluación, el usuario no realiza la llamada a la función de forma directa, sino que esta se hace desde del método `optimize` de la clase `AlgoTrader`. La descripción de los parámetros de entrada y salida ayuda a comprender cómo tienen que implementarse los métodos de optimización:

Función de selección Se trata de la función que marca la dirección que sigue el proceso de optimización: puede ser maximizar o minimizar. Es el primer argumento del método de optimización y se identifica en la instrucción anterior con `selectionFunction`, que está precedida por el carácter `@` al tratarse de una función MATLAB. Los valores que toma son `max` o `min` según se busque maximizar o minimizar la característica que mide la función de evaluación. Las funciones de selección se explican en la sección 7.3.

Función de aptitud Es la función encargada de medir una característica de la estrategia de trading. Ocupa la segunda posición entre los argumentos y en la instrucción se identifica con `fitnessFunction`, precedida por el carácter `@`. Esta función la genera de forma automática el método `optimize`, a partir de la función de evaluación `evaluationFunction` y del espacio de búsqueda constituido por los pares `Parameter*` y `Domain*`. Consiste en una función envoltorio que conoce los detalles sobre la función de evaluación y el espacio de búsqueda. Sirve para desacoplar al método de optimización de las características del problema que resuelve. La función de aptitud toma como argumento de entrada un vector de números enteros cuya longitud es igual al número de dimensiones del espacio de búsqueda y hace con esto una asignación de valores a los parámetros que se busca optimizar; después, la función mide y devuelve el desempeño de la estrategia, fruto de la asignación de valores a los parámetros utilizando la función de evaluación. Las funciones de evaluación que encapsula la función de aptitud aparecen en la sección 7.2.

Tamaño del espacio de búsqueda Queda determinado por el número de dimensiones y el tamaño de cada una de ellas. Este argumento está en la tercera posición y se identifica con `searchSpaceSize`. Se codifica como un vector de números enteros con tantos elementos como dimensiones tenga el espacio de búsqueda y donde el valor de cada elemento se corresponde con el tamaño de la dimensión asociada, siendo el primer elemento el responsable de indicar el tamaño de la primera dimensión, el segundo elemento de la segunda dimensión y así sucesivamente.

Array de índices El método de optimización devuelve un array de índices. Estos índices referencian a los valores de los parámetros que hacen que la estrategia consiga el mejor resultado. Es el argumento de salida de la función y se identifica con `bestIndexArray` en la instrucción. Con este array de índices, el método `optimize` selecciona los valores de los parámetros de la estrategia.

La parte más complicada del proceso consiste en comprender cómo se utiliza la función de aptitud y de qué manera su argumento de entrada está relacionado con el tamaño del espacio de búsqueda. Para aclarar estos detalles, se muestra un ejemplo ilustrativo que consiste en la ejecución de una llamada al método `optimize`, donde se utiliza: la máquina de trading basada en la media móvil explicada en la sección 5.1, el método de optimización por enumeración de la sección 7.4 y la función de evaluación de beneficios y pérdidas de la sección 7.2. La instrucción de partida es la que aquí aparece:

```
ma.optimize(@profitLoss, @max, @exhaustive, ...
       'Samples', 5:35, 'Mode', {'e', 's'})
```

Supuesta la existencia de una variable con nombre `ma` que sea instancia de la clase `MovingAverage`, entonces la instrucción anterior calcula el valor óptimo del parámetro `Samples` utilizando el conjunto de valores `5:35` y del parámetro `Mode` en el conjunto `{'e', 's'}`, aplicando el método de optimización `exhaustive`. Al concluir la ejecución del método `optimize`, estos dos parámetros toman aquellos valores que más mejoran el desempeño de la estrategia: en particular, el proceso de optimización busca maximizar el beneficio que logra la estrategia, propósito que se consigue con las funciones `profitLoss` y `max` facilitadas como argumentos del método.

Antes de lanzar el proceso de optimización `exhaustive` hay que crear la función de aptitud, que se le pasa como uno de los argumentos. Esta función se genera a partir de la función de evaluación `profitLoss` y de todos los pares de parámetros y conjuntos de valores introducidos en la llamada al método `optimize`: `Samples` con los valores `5:35` y `Mode` con los valores `{'e', 's'}`. Para el ejemplo que se trata, el resultado es una función de aptitud que toma como argumento de entrada un vector de dos elementos, donde el primero referencia los valores asociados a `Samples` y el segundo los relativos a `Mode`. Si esta función de aptitud `fitnessFunction` toma el argumento de entrada `[1 1]`, entonces evalúa el desempeño de la estrategia para el caso en que `Samples = 5` y `Mode = 'e'`, si el argumento de entrada es sin embargo `[10 2]`, entonces mide el desempeño para el caso en que `Samples = 14` y

`Mode = 's'`. En cada una de las evaluaciones que hace la función de aptitud, se utiliza el simulador de máquinas de trading de la plataforma. Como se aprecia, esta función generada de forma automática es un elemento clave en el sistema de optimización de parámetros: primero, desacopla al método de optimización de conocer detalles sobre el tipo de los valores que maneja, que en el ejemplo abordado consiste en números y cadenas de caracteres; segundo, oculta la complejidad inherente a la ejecución del sistema de simulación de máquinas de trading; y tercero, hace posible cambiar el criterio para medir el desempeño de la máquina de trading al utilizar diferentes funciones de evaluación. Los demás parámetros de la máquina, cuyos valores no se busca optimizar, se consideran como constantes y no se alteran.

El último argumento obligatorio que hay que pasar al método de optimización es el tamaño del espacio de búsqueda. Sucede que la función de aptitud toma un vector de elementos que referencia valores de los parámetros; para no introducir en este vector índices que estén fuera del rango es necesario saber cuáles son los límites. Siguiendo con el ejemplo, el tamaño del espacio de estados es `[31, 2]` pues 31 son los valores considerados para el parámetro `Samples` y 2 los del parámetro `Mode`. Con la información sobre el tamaño del espacio de búsqueda, el método de optimización sabe cuáles son los valores de los índices que puede introducir en la función de aptitud, sin provocar una excepción por referirse a valores fuera de rango.

Tras generar los argumentos que consume la función de optimización, el método `optimize` llama a dicha función. Siguiendo con el ejemplo, la llamada a la función de optimización es:

```
bestIndexArray = exhaustive(@max, @fitnessFunction, [31, 2])
```

Finalizada la ejecución del método de optimización que implementa la función `exhaustive`, se devuelven en el vector `bestIndexArray` los índices de aquellos valores de los parámetros que más mejoran el resultado de la estrategia de trading, atendiendo al objetivo marcado por las funciones `max` y `fitnessFunction`. Para concluir, el método `optimize` toma el vector de índices devuelto y efectúa las asignaciones sugeridas.

Un resumen gráfico de las partes que interactúan en el proceso de optimización de parámetros aparece en la figura 3.2. Ahí se resalta cuáles son los argumentos del método `optimize`: la función de evaluación, la función de selección, el método de optimización y el espacio de búsqueda. Aparece también en la figura la función de fitness, que se identifica con `fitnessFunction` y que requiere de la función de evaluación y del espacio de búsqueda. A su vez, el espacio de búsqueda se utiliza para calcular su tamaño, que aparece como `searchSpaceSize`. Por último, se aprecia cómo todos estos elementos conforman los argumentos del método de optimización, que aparece como `optimizationMethod`.

Al igual que en el caso de la función de evaluación, se pueden pasar parámetros al método de optimización por medio de la llamada al método `optimize`. También aquí la razón para permitir tal posibilidad es la misma: el usuario no lanza de forma directa el método de optimización, luego ha de permitirse un canal para indicar parámetros en caso de ser necesarios. La manera en que se hace es utilizando una llamada que siga un patron como el siguiente:

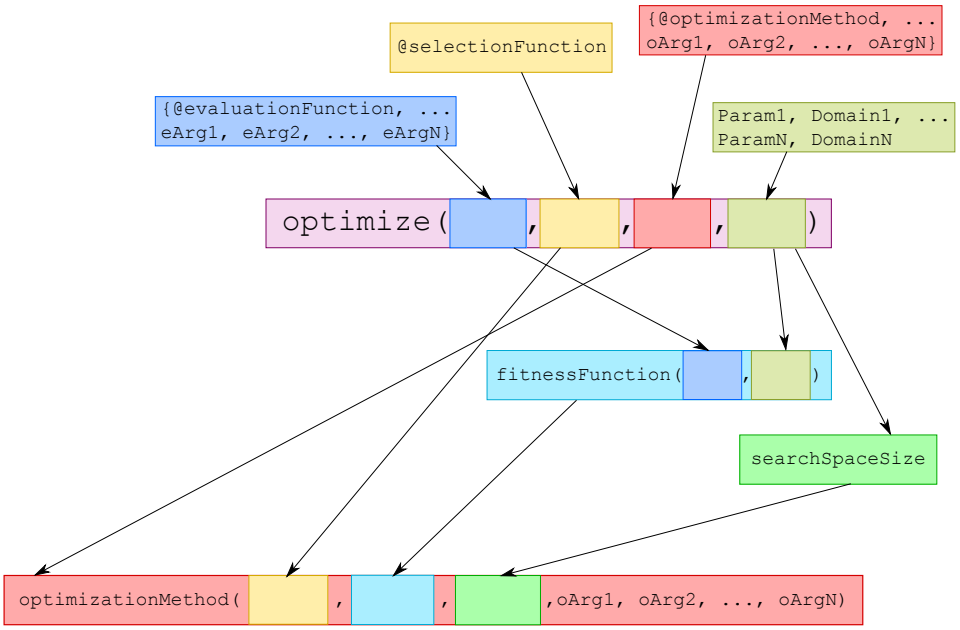

Figura 3.2: Elementos que intervienen en la llamada al método `optimize`

```
algoTraderInstance.optimize(@evaluationFunction, @selectionFunction, ...
                        {@optimizationMethod, Arg1, ..., ArgN}, ...
                        'Parameter1', Domain1, ..., 'ParameterN', DomainM)
```

De esta manera, todos los parámetros `Arg1`..., `ArgN` se pasan al método de optimización. Para transmitir estos parámetros, la llamada que hace el método `optimize` hacia la función de optimización mantiene el formato mostrado a continuación:

```
bestIndexArray = optimizationMethod(@max, @fitnessFunction, ...
                        [SizeDomain1, ..., SizeDomainM], ...
                        Arg1, ..., ArgN)
```

Los métodos de optimización implementados se describen en la sección 7.4.

Organización de la documentación sobre el método `optimize`

Los elementos que intervienen en el método `optimize` son la función de evaluación, la función de selección, el método de optimización, la función de fitness y el espacio de búsqueda. En la sección anterior se explicó de forma global el funcionamiento del método `optimize` y los elementos que toman parte en la computación. En tanto a las partes más complejas del método, son explicadas en secciones posteriores. Las funciones de evaluación están documentadas en la sección 7.2 y las de selección en la sección 7.3. Los métodos de optimización se recogen en la sección 7.4.

3.4. Generación de resultados

Una herramienta de investigación debe facilitar el realizar experimentos y recoger los resultados obtenidos de ellos. En el estudio de las máquinas de trading hay un sinfín de características que pueden ser objeto de interés por parte del investigador: el rendimiento de la máquina de acuerdo al beneficio (o pérdida) conseguido, la influencia de los costes de operación (el coste de comprar y vender acciones), la mejor racha (posiciones consecutivas exitosas que adopta la máquina en el mercado), la peor racha (posiciones consecutivas fallidas que toma la máquina en el mercado), los posibles rendimientos de la máquina de acuerdo a los valores que toman sus parámetros...

Los bloques funcionales que extienden a la plataforma fueron mostrados en la figura 3.1. Cada uno de ellos genera resultados de acuerdo a los aspectos en que se centra. A lo largo del documento se describen los resultados que estas partes producen. Las propiedades, métodos y funciones que generan resultados se recogen en el capítulo 8, luego de que hayan sido explicados los bloques funcionales.

3.5. El archivo de configuración `Settings.m`

En el archivo `Settings.m` se almacenan las configuraciones y los valores por defecto de las clases de la plataforma. Con esta solución se facilita el aplicar cambios sin manipular los archivos de código. Los aspectos que permite modificar son los siguientes:

- Los valores por defecto de las propiedades de la clase `AlgoTrader`, como son: el tipo de posiciones que puede adoptar la máquina (sólo posiciones largas, sólo posiciones cortas o ambas), los intervalos de entrenamiento y test, los fondos iniciales con que cuenta la máquina de trading, la cantidad máxima que se puede invertir y el coste de operar en el mercado. Las propiedades de la clase `AlgoTrader` están recogidas en el apéndice B.

- Los valores por defecto de las propiedades particulares de cada una de las máquinas de trading. Así, se evita la molestia de conocer los argumentos del constructor de la máquina. El único argumento obligatorio en todo caso es la serie de datos, que consiste en una instancia de la clase `DataSerie` que se pasa al constructor como primer parámetro; el resto son opcionales. Las máquinas de trading y sus propiedades están documentadas en los capítulos 5 y 6.

- Los colores y tipos de gráficos utilizados en el sistema de dibujado. Muchas de las características de los gráficos que generan las diferentes clases están definidas en este archivo de configuración. Alterando estos valores se cambian las representaciones que ofrecen las clases que heredan de `DataSerie` o `AlgoTrader`.

Capítulo 4

Conexión con los proveedores de datos

La lectura de datos reales es imprescindible; en este sentido, se han tenido en cuenta los diferentes proveedores del mercado, centrándose en aquellos más económicos y de fácil acceso. En concreto, se ha optado por la inclusión de dos proveedores gratuitos como son Yahoo! Finance y Google Finance. El tercer proveedor es VisualChart V, que requiere del registro de una cuenta de usuario gratuita. Este último proveedor es capaz de adquirir datos con mayor resolución que los dos anteriores, aunque como contrapartida, es necesario esperar hasta el cierre de los mercados para tener acceso a los datos del día. Esta es la limitación de uso que impone la cuenta de usuario gratuita.

Aunque diferentes, los proveedores de datos son utilizados de forma homogénea gracias a la interfaz propuesta por la clase abstracta `DataSerie`. Esta pieza fundamental de la plataforma permite extender el desarrollo más allá de los proveedores aquí implementados. Contar con un tratamiento homogéneo de la información hace posible que el código escrito para la clase `AlgoTrader` sea más elegante y esté desacoplado de los detalles propios de cada proveedor. Detalles como este hacen manifiestas las buenas prácticas de programación seguidas durante el desarrollo de la plataforma. Para una descripción técnica de la clase `DataSerie` se puede consultar el apéndice A.

El resto del capítulo trata sobre la implementación concreta de los proveedores de datos seleccionados. Se puede observar que el proceso seguido es el mismo para cada uno de ellos, si bien difieren en la metodología, a consecuencia de las particularidades de cada uno. Las fases del proceso son las siguientes: analizar cómo se interactúa con el proveedor, estudiar cómo es la estructura de datos a la que da acceso, implementar un sistema de transferencia de datos y por último almacenar la información de forma local. Todas las fases del proceso, a excepción del almacenamiento local que es automático, se implementan en el método `get`. Los pasos seguidos son los mismos para todos los proveedores implementados en el capítulo. En la sección 4.1 se implementa el proveedor de datos bursátiles de Google. Luego, en la sección 4.2 se implementa el proveedor de Yahoo!, que es muy parecido al anterior. Por último, en la sección 4.3 se trata sobre la implementación del proveedor de VisualChart V, que es el más versátil y complejo. La implementación de este último requiere de un pequeño desarrollo sobre la plataforma .NET para conectar VisualChart V con MATLAB.

Figura 4.1: Información sobre una acción bursátil en Google Finance

4.1. Google Finance

Google Finance es un sitio web producto de la empresa Google. El servicio ofrece noticias sobre las decisiones financieras de empresas y estados, así como otros grandes eventos recogidos por la prensa. El sitio web está integrado con los servicios Google News y Google Blog Search para incorporar información adicional sobre las corporaciones. Este sitio web dispone de información sobre las cotizaciones bursátiles, los precios del cambio de divisas, el rendimiento del mercado de Estados Unidos agrupado por sectores... Respecto a las cotizaciones bursátiles, almacena datos históricos de empresas de los Estados Unidos para los últimos 40 años.

4.1.1. Interacción con el proveedor

La interacción con Google Finance se hace a través de su sitio web[1]. Utilizando el buscador de acciones bursátiles que incorpora, se accede a la información sobre las cotizaciones. Así por ejemplo, la figura 4.1 muestra la cotización de las acciones de la empresa Ford, según fue buscada en este sitio web. En la figura se aprecia un enlace hacia los datos históricos, que ha sido señalado con una flecha. Al acceder hacia la página de los datos históricos, se encuentra una tabla ordenada de forma cronológica con la información de las cotizaciones. Lo interesante de esta página está en el enlace para descargar la información, que se obtiene como una hoja de cálculos con los datos históricos solicitados. La figura 4.2 muestra la captura descrita, indicando con una flecha el enlace que contiene la información en un archivo de hoja de cálculos.

[1] `http://google.com/finance`

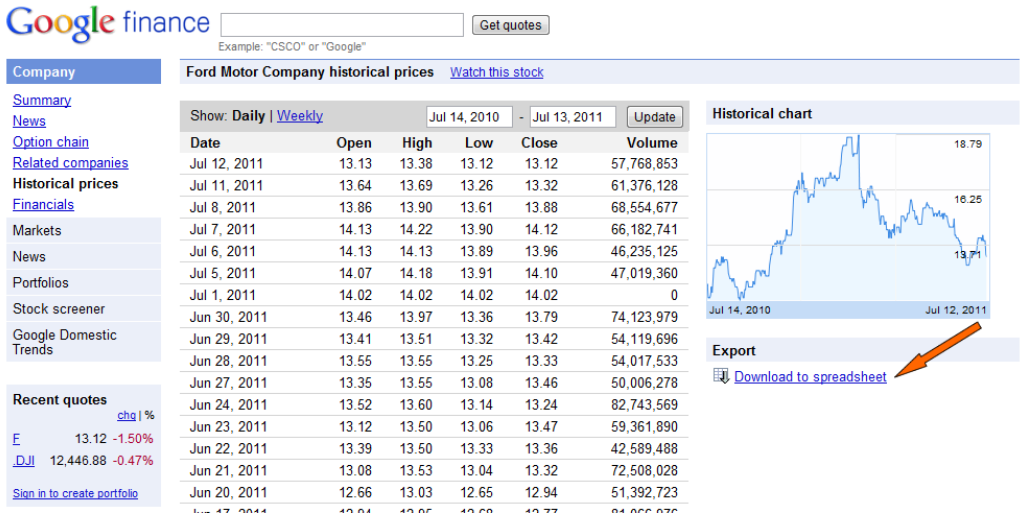

Figura 4.2: Cotización de una acción bursátil en Google Finance

Para automatizar este proceso, se analiza la URL (*Uniform Resource Locator*) que apunta hacia el archivo con los datos históricos. Tras una serie de pruebas, se concluyó cuáles eran las cadenas a utilizar para formar la URL que apunta hacia la información de interés. Tal dirección consiste en realidad en una consula que se pasa al proveedor para que genere el archivo de datos oportuno. Esta consulta está compuesta por partes individuales que, combinadas a través de un caracter de separación, especifican todos los criterios de búsqueda deseados. Las componentes que se combinan para formar la dirección para la petición del archivo se describen a continuación:

http://google.com/finance/historical? Prefijo de la cadena que compone la dirección que apunta hacia la hoja de cálculos.

& Símbolo separador. Es necesario utilizarlo para combinar los patrones y componer la dirección.

q=VALOR Patrón con el que se especifica la acción concreta de la que se quiere obtener los datos históricos. La subcadena *VALOR* se refiere a la cadena de texto utilizada por Google Finance para identificar a la acción.

output=csv Formato en que se desean obtener los datos históricos. El formato csv (*Comma-separated values*) consiste en un archivo de texto que con saltos de líneas y comas distingue las filas y las columnas de una tabla de valores. Es el único formato de salida que admite Google Finance.

histperiod=(daily|weekly) Patrón para indicar el tipo de compresión con que se piden los datos históricos: puede ser *daily* para datos diarios o *weekly* para semanales. Agrupando los datos es posible conseguir datos mensuales, anuales o de cualquier otro tipo con resolución menor a la diaria. En caso de omisión, los datos devueltos son diarios.

Date,	Open,	High,	Low,	Close,	Volume
12-Jul-11,	13.13,	13.38,	13.12,	13.12,	57768853
11-Jul-11,	13.64,	13.69,	13.26,	13.32,	61376128
8-Jul-11,	13.86,	13.90,	13.61,	13.88,	68554677
7-Jul-11,	14.13,	14.22,	13.90,	14.12,	66182741
6-Jul-11,	14.13,	14.13,	13.89,	13.96,	46235125
5-Jul-11,	14.07,	14.18,	13.91,	14.10,	47019360
1-Jul-11,	14.02,	14.02,	14.02,	14.02,	0
30-Jun-11,	13.46,	13.97,	13.36,	13.79,	74123979
29-Jun-11,	13.41,	13.51,	13.32,	13.42,	54119696

Tabla 4.1: Contenido del archivo de datos descargado desde Google Finance

startdate=YYYY-MM-DD Patrón para especificar la fecha de comienzo del intervalo de los datos históricos. En caso de omisión, se toma la fecha más antigua a partir de la cual se disponga de datos históricos.

enddate=YYYY-MM-DD Patrón para especificar la fecha de final del intervalo de los datos históricos. En caso de omisión, se toma la fecha más reciente hasta la que se disponga de datos históricos.

La cadena `http://google.com/finance/historical?q=NYSE:F&output=csv` es un ejemplo de cadena bien formada que contiene la dirección de un archivo de datos históricos. Esta cadena es la correspondiente al enlace señalado la flecha de la figura 4.2.

4.1.2. Estructura de datos remota

La estructura de datos remota consiste en un archivo con valores separados por saltos de linea y comas que se interpreta como una hoja de cálculo. Los saltos de línea sirven para discriminar entre filas y las comas, entre columnas. En la tabla 4.1 aparece el contenido de uno de estos archivo. La primera linea consiste en la descripción de los datos contenidos en cada columna: fecha, precio de apertura, precio más alto, precio más bajo, precio de cierre y volumen de transacciones. Las siguientes líneas contienen los datos ordenados de forma cronológica, comenzando por los más recientes.

4.1.3. Transferencia de datos

Para conseguir los datos hay que generar la URL con las opciones adecuadas y descargar el archivo al que apunta. Esta operación se realiza utilizando la función `urlread` de MATLAB, que lee el contenido apuntado por una dirección URL y devuelve como resultado una cadena de texto. Hay un problema con el proveedor Google Finance y es que sólo permite descargar archivos con un contenido limitado a 4000 líneas. Ante peticiones de mayor tamaño, no se recibirán todos los datos solicitados. Una manera de arreglar esta limitación consiste en hacer descargas secuenciales hasta satisfacer el proceso de recolección de datos. También es importante ignorar la primera linea de los archivos descargados, pues sólo contendrá la descripción de cada una de las columnas.

4.1.4. Extracción de la información

El formato del archivo de datos es regular y la extracción de la información es inmediata. En particular, se utilizan las expresiones regulares de MATLAB para leer y extraer la información del archivo de datos. Como resultado de la extracción, se obtienen vectores de valores.

Además de extraer la información, en esta fase también se produce la normalización de los valores. De manera nativa, Google Finance ofrece datos agrupados en forma de días o semanas, ¿cómo es posible conseguir otro tipo de agrupación? La manera en que se hace es procesando los datos extraídos y agrupándolos según sea necesario. Si por ejemplo se piden con una resolución de tres días, entonces se descargan con una frecuencia diaria y se agrupan de tres en tres.

4.1.5. Almacenamiento local

El método `get` realiza los pasos anteriores hasta la extracción de datos. El resultado que este método devuelve consiste en los vectores de información de las cotizaciones. En el proceso de almacenamiento local se toman los vectores de información devueltos por el método `get` y se almacenan en las propiedades del objeto que hereda de la clase `DataSerie`. La llamada al método `get` sólo la realiza el constructor de la clase durante la creación de una nueva instancia. Llegados a este punto el objeto contiene la información solicitada, que se determina a partir de los argumentos en la llamada al constructor de la clase.

4.1.6. La clase `GoogleDataSerie`

Siguiendo las indicaciones anteriores, se implementa en la plataforma la conexión hacia el proveedor de datos Google Finance. El desarrollo se encuentra en la clase `GoogleDataSerie`, que hereda de la clase `DataSerie` e implementa el método abstracto `get`. Gracias a la clase `DataSerie`, la cantidad de código escrita para implementar la clase `GoogleDataSerie` ha sido mínima. Una descripción detallada de las posibilidades que ofrece la clase `GoogleDataSerie` a partir de sus propiedades y métodos aparece en el apéndice A.

4.2. Yahoo! Finance

Yahoo! Finance es un servicio web propiedad de Yahoo! que proporciona información financiera. Es el sitio web con noticias y estudios financieros más utilizado en los Estados Unidos. Ofrece información sobre cotizaciones bursátiles, bolsas de valores, comunicados de prensa e informes financieros. Incluye algunas herramientas para gestionar las finanzas personales. Este portal opera en muchos países con la información de las bolsas nacionales.

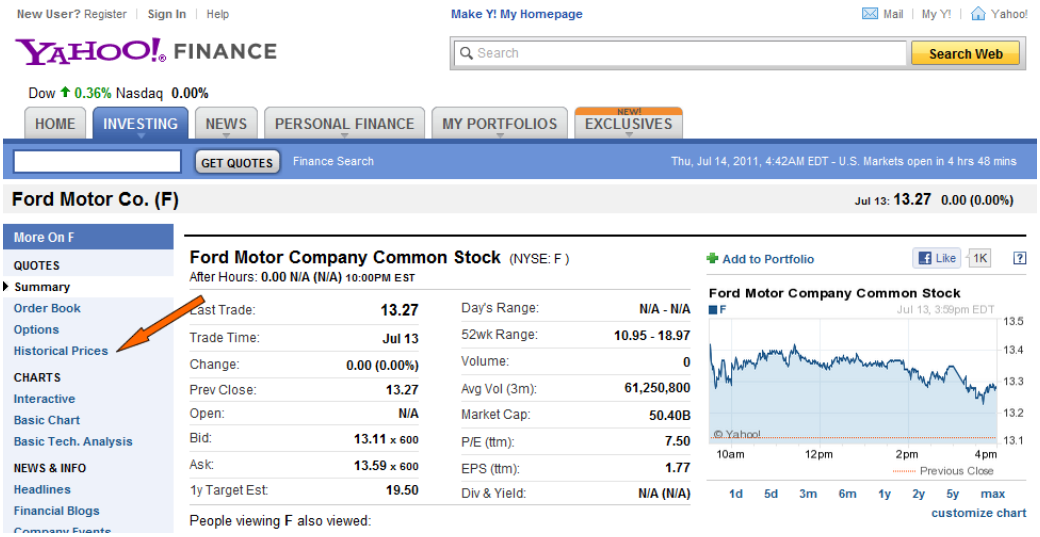

Figura 4.3: Información sobre una acción bursátil en Yahoo! Finance

4.2.1. Interacción con el proveedor

Al igual que en el caso anterior, también se interactúa con Yahoo! Finance por medio de su sitio web[2]. Un buscador conduce hacia la información sobre las acciones bursátiles, incluyendo sus cotizaciones. Si se efectúa una búsqueda, se llega hasta una página que aglutina en forma de resumen la información más relevante respecto a una acción. La figura 4.3 muestra la página en cuestión, donde se ha incluído una flecha para señalar el enlace hacia la información histórica de la acción mostrada. Este enlace conduce a otra página donde aparecen las cotizaciones de la acción en forma de tabla de datos. Hacia el final de esta página, se encuentra un enlace para descargar archivo con la información de las cotizaciones. La figura 4.4 muestra el sitio web con una flecha apuntando al enlace donde se encuentra el archivo.

La dirección del enlace es en realidad un conjunto de instrucciones que se le pasan al servidor web del proveedor para que genere el archivo con los datos solicitados. Es necesario comprender la manera en que se confecciona la dirección URL para generar las consultas de forma automatizada. La estructura de la cadena consiste en la combinación de varios elementos por medio de un carácter separador: primero está la dirección del servidor remoto, luego aparecen las instrucciones para efectuar la consulta. Las partes de la dirección se describen a continuación:

http://ichart.finance.yahoo.com/table.csv? Prefijo de la cadena de dirección que apunta hacia el servidor web donde se realizan las peticiones de datos.

& Carácter separador. Se utiliza para combinar las diferentes partes de la cadena.

s=VALOR Patrón para indicar la acción de la que se solicita la información del proveedor. La subcadena *VALOR* se refiere a la cadena de texto utilizada por Yahoo! Finance para identificar a la acción.

[2] `http://finance.yahoo.com`

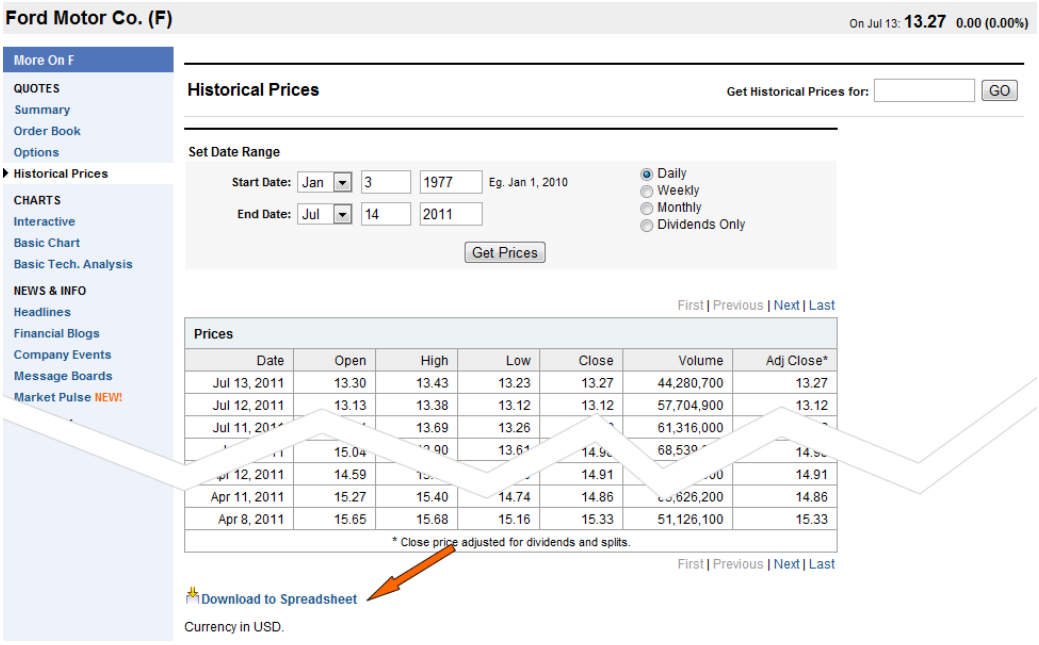

Figura 4.4: Cotización de una acción bursátil en Yahoo! Finance

ignore=.csv Formato de salida de los datos históricos. El formato csv consiste en un archivo de texto que almacena información dispuesta en forma de tabla. Utiliza saltos de linea y comas para identificar las diferentes filas y columnas de la tabla. Es el único formato de salida que admite Yahoo! Finance.

g=(d|w|m) Cadena con la que se indica el tipo de compresión con que se solicitan los datos históricos. Los valores permitidos son *d* para valores diarios, *w* para semanales y *m* para mensuales. A partir de la agrupación es posible conseguir otros tipos de compresión. Los datos generados son diarios en caso de omisión.

b=DD&a=MM&c=YYYY Patrón para especificar la fecha de comienzo del intervalo de los datos históricos. La subcadena *DD* se sustituye por el día, la subcadena *MM* se sustituye por los meses, teniendo en cuenta que el primer mes se representa con el valor 0, y la subcadena *YYYY* se sustituye por el año. En caso de omisión, se toma la fecha más antigua a partir de la cual se disponga de datos históricos.

e=DD&d=MM&f=YYYY Patrón para especificar la fecha final del intervalo de los datos históricos. La sustitución de las subcadenas se hace siguiendo el mismo procedimiento que en el caso anterior. En caso de omisión, se toma la fecha más reciente hasta la que se disponga de datos históricos.

La cadena `http://ichart.finance.yahoo.com/table.csv?s=F&g=d&ignore=.csv` es un ejemplo de cadena bien formada, con la dirección y las instrucciones necesarias para producir un archivo con datos históricos. Esta cadena es similar a la del enlace que aparece señalado en la figura 4.4.

Date,	Open,	High,	Low,	Close,	Volume,	Adj Close
2011-07-13,	13.30,	13.43,	13.23,	13.27,	44280700,	13.27
2011-07-12,	13.13,	13.38,	13.12,	13.12,	57704900,	13.12
2011-07-11,	13.64,	13.69,	13.26,	13.32,	61316000,	13.32
2011-07-08,	13.86,	13.90,	13.61,	13.88,	68539000,	13.88
2011-07-07,	14.13,	14.22,	13.90,	14.12,	66168700,	14.12
2011-07-06,	14.13,	14.13,	13.89,	13.96,	46218000,	13.96
2011-07-05,	14.07,	14.18,	13.91,	14.10,	47003800,	14.10
2011-07-01,	13.92,	14.13,	13.73,	14.02,	66304800,	14.02
2011-06-30,	13.46,	13.97,	13.36,	13.79,	74076900,	13.79
2011-06-29,	13.41,	13.51,	13.32,	13.42,	54105500,	13.42
2011-06-28,	13.55,	13.55,	13.25,	13.33,	54013200,	13.33
2011-06-27,	13.35,	13.55,	13.09,	13.46,	49987600,	13.46
2011-06-24,	13.52,	13.60,	13.14,	13.24,	82707400,	13.24
2011-06-23,	13.12,	13.50,	13.06,	13.47,	59323000,	13.47

Tabla 4.2: Contenido del archivo de datos descargado desde Yahoo! Finance

4.2.2. Estructura de datos remota

Ante la petición enviada al servidor, se obtiene como resultado un archivo de texto con los valores. El contenido del archivo se asemeja a la tabla de valores mostrada en la figura 4.4. Utilizando saltos de línea y comas se organizan los valores en filas y columnas. El contenido de uno de estos archivos de texto aparece en la tabla 4.2. En la primera linea, se incluye una descripción de los datos que contiene cada columna: fecha, precio de apertura, precio más alto, precio más bajo, precio de cierre, volumen de transacciones y precio de cierre ajustado. La columna de precio de cierre ajustado tiene en cuenta todos los fraccionamientos[3] y dividendos de la acción, aplicando las modificaciones pertimentes de manera retroactiva. Las líneas siguientes contienen los datos ordenados de forma cronológica, con los más recientes ocupando las posiciones superiores.

4.2.3. Transferencia de datos

Los datos se solicitan al proveedor por medio de la dirección URL. Esta dirección es una consulta con las indicaciones sobre los datos requeridos. El resultado de la consulta da lugar a un archivo de texto con la estructura antes descrita. El contenido del archivo se lee con la función `urlread` de MATLAB.

[3] El fraccionamiento o escisión de una acción (en inglés un *split*) ocurre cuando la compañía emite un número de acciones en una determinada proporción a partir de las ya existentes. Así por ejemplo si el fraccionamiento es 2-1 entoces cada accionista duplica su número de acciones aunque el capital total se conserva por lo que el precio de una acción es recortado hasta la mitad. También son posibles los fraccionamientos a la inversa en cuyo caso el número de acciones se reduce a la par que se incrementa el valor por unidad para mantener el capital total.

4.2.4. Extracción de la información

Para extraer la información a partir del archivo con la cadena de texto, se lee el contenido utilizando una expresión regular. No hay complicación alguna en esta fase, pues las líneas que componen el texto siguen todas el mismo patrón. Tras procesar el archivo de datos se obtienen vectores de datos en el formato de MATLAB.

Luego, se realiza la normalización y adecuación de los valores. A través de Yahoo! Finance los datos pueden ser solicitados con un tipo de compresión diaria, semanal o mensual. Si se requiere una agrupación diferente de los valores, entonces se modifican de la forma pertinente. Por ejemplo, si se piden valores con una frecuencia trimestral, entonces estos se solicitan con una frecuencia mensual y se agrupan de tres en tres.

4.2.5. Almacenamiento local

El método `get` realiza los pasos anteriores hasta la extracción de datos. El resultado que este método devuelve consiste en los vectores de información de las cotizaciones. En el proceso de almacenamiento local se toman los vectores de información devueltos por el método `get` y se almacenan en las propiedades del objeto que hereda de la clase `DataSerie`. La llamada al método `get` sólo la realiza el constructor de la clase durante la creación de una nueva instancia. Llegados a este punto el objeto contiene la información solicitada, que se determina a partir de los argumentos en la llamada al constructor de la clase.

4.2.6. La clase `YahooDataSerie`

La clase `YahooDataSerie` es la realización del proceso explicado en los apartados anteriores. Con ella se obtienen datos provinientes de la plataforma Yahoo! Finance. La clase `YahooDataSerie` hereda de la clase abstracta `DataSerie` e implementa el método abstracto `get`. Al heredar de la clase `DataSerie`, se reduce el esfuerzo de implementación y se tiene acceso a todas las funcionalidades provistas por la clase. Las posibilidades que ofrece la clase `DataSerie`, y por ende la clase `YahooDataSerie`, se recogen en el apéndice A.

4.3. VisualChart V

VisualChart V es un software de uso profesional propiedad de Visual Chart Group. Este programa obtiene información bursátil desde los mercados y permite emitir operaciones de compra y venta en la bolsa. Se necesita conexión a un broker para poder operar en la bolsa. La aplicación ofrece un entorno integral orientado al análisis técnico y al análisis chartista. Como cuestión avanzada, este software aporta herramientas para implementar estrategias de trading personalizadas utilizando el lenguaje de programación Visual Basic.

La aplicación VisualChart V sólo funciona bajo el sistema operativo Windows. Esta restricción ha condicionado el desarrollo del trabajo. Las conexiones anteriores no cuentan con la citada limitación y funcionan sobre cualquier sistema operativo sobre el que sea instalado MATLAB.

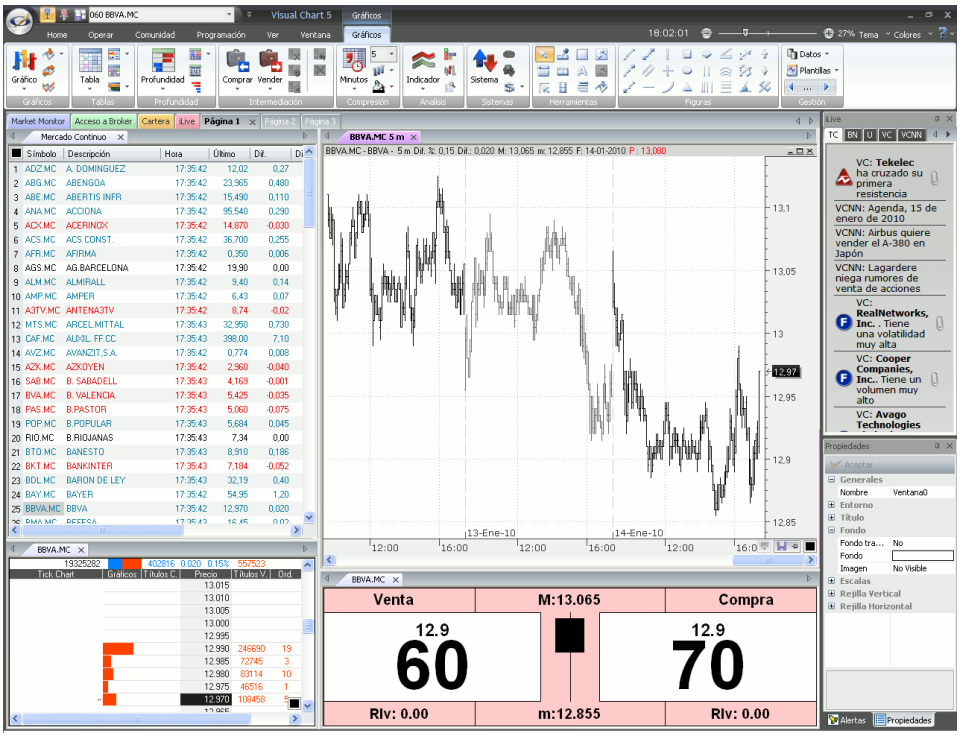

Figura 4.5: Interfaz de usuario de la aplicación VisualChart V

4.3.1. Interacción con el proveedor

La aplicación VisualChart V proporciona toda la información sobre el mercado bursátil por medio de una interfaz de usuario, como la mostrada en la figura 4.5. El problema, sin embargo, es que toda esa información ha de ser transferida hacia MATLAB para ser utilizada. Lo que sucede en este caso es que la aplicación VisualChart V hace de intermediaria entre el proveedor de información y MATLAB. En consecuencia, el proceso de interacción está dividido en dos fases: en la primera MATLAB interactúa con VisualChart V y en la segunda VisualChart V lo hace con el proveedor de información bursátil.

Para acometer la primera fase del proceso, se utiliza la interfaz COM que declara la aplicación VisualChart V. Por su parte, MATLAB es capaz de utilizar de forma nativa la interfaz COM, para así dialogar con VisualChart V. Sucede, sin embargo, que los tipos de datos devueltos por la interfaz COM de VisualChart V son incompatibles con la versión del interfaz COM implementado en MATLAB. El salto entre las aplicaciones se superó creando un proyecto con la tecnología .NET e implementado en C#. Tal proyecto consiste en una clase en forma de biblioteca de enlace dinámico que interactúa con VisualChart V y es utilizada desde MATLAB. Este conflicto se comentó en la sección 1.4.

La segunda fase del proceso que comunica VisualChart V con el mercado bursátil es transparente y se logra por medio de las llamadas a la clase creada en la biblioteca de enlace dinámico. Esta clase se encarga de gestionar el estado de las comunicaciones de VisualChart V: abre la conexión con el proveedor, solicita los datos, descarga los resultados de las peticiones y cierra la conexión.

4.3.2. Estructura de datos remota

La estructura de datos remota puede concebirse como una base de datos sobre la que se pueden realizar consultas. Las consultas, al igual que en los casos anteriores, necesitan el identificador de la acción, el tipo de compresión, las unidades de compresión y el intervalo temporal. Sobre el tipo de compresión, VisualChart V permite obtener datos con las frecuencias de ticks, minutos, días, semanas o meses. Para gestionar las fuentes de datos originadas por las consultas, la interfaz COM ofrece la clase `VCDS_DataSourceManager`. El resultado de cada petición es un objeto de tipo `VCDS_DataSerie`, que almacena un conjunto de registros. Cada registro se representa por medio de la clase `VCDS_BarValue` de la interfaz y contiene la fecha, el precio de apertura, el precio más alto, el precio más bajo, el precio de cierre, el volumen de transacciones de la acción y el interés abierto[4]. El conjunto de registros ordenados de forma cronológica es la forma en que VisualChart V codifica las series temporales de datos bursátiles.

La clase envoltorio a través de la que opera MATLAB es `DataSourceManager`. Esta hace uso de las clases `DataSerie` y `BarValue`. Las clases son homólogas a aquellas provistas por la interfaz COM, cuyo nombre comienza por el prefijo `VCDS_`. Estas clases envoltorio ofrecen un subconjunto de la funcionalidad de las clases originarias, que es la necesaria para llevar los datos hasta MATLAB. Es importante destacar que existen en el trabajo dos clases con el nombre `DataSerie`: una es la clase abstracta implementada en MATLAB, a partir de la cual se hereda para incorporar conexiones a proveedores de datos; y la otra es la del desarrollo en C#, necesaria para comunicar MATLAB con VisualChart V.

4.3.3. Transferencia de datos

La transferencia de datos la realiza el servidor de comunicaciones de VisualChart V al utilizar la interfaz COM. Se requiere de conexión a Internet y del registro de una cuenta de usuario gratuita. La aplicación debe estar en ejecución para poder realizar las peticiones.

4.3.4. Extracción de la información

Para extraer la información, se leen los datos desde las clases de la interfaz COM y se almacenan en las clases homólogas creadas en C#. A su vez, para extraer la información desde estas clases de C#, se definen una serie de propiedades de sólo lectura que son accesibles por MATLAB. En particular, la clase `DataSerie` en C# implementa la propiedad de sólo lectura `Matrix`, que devuelve una matriz de datos donde las filas contienen registros de datos y las columnas se refieren a los diferentes campos de los registros. Una descripción de las columnas de la matriz se muestra en la tabla 4.3.

[4]El interés abierto se refiere al número de órdenes de compra que existen antes de que el mercado de acciones inicie una sesión.

#columna	Descripción
1	Año
2	Mes
3	Día
4	Hora
5	Minuto
6	Segundo
7	Precio de apertura
8	Precio más alto
9	Precio más bajo
10	Precio de cierre
11	Volumen de transacciones
12	Interés abierto

Tabla 4.3: Descripción de las columnas de la propiedad `Matrix` de la clase `DataSerie`

4.3.5. Almacenamiento local

El método `get` realiza los pasos anteriores mediante el uso de la biblioteca de enlace dinámico `VisualChartConnector.dll`. Este archivo contiene todas las clases C# necesarias para mediar en el proceso de comunicación entre MATLAB y VisualChart V. Una vez importados los valores desde la propiedad `Matrix` de la clase `DataSerie` (clase del desarrollo en C#), ya se tiene todo lo necesario para instanciar el objeto de la clase que hereda de `DataSerie` (clase abstracta que se hereda para incorporar la conexión al proveedor de datos).

En el proceso de almacenamiento local, se toman los vectores de información devueltos por el método `get` y se almacenan en las propiedades del objeto. La llamada al método `get` sólo la realiza el constructor de la clase durante la creación de una nueva instancia. Llegados a este punto el objeto contiene la información solicitada, que se determina a partir de los argumentos en la llamada al constructor de la clase.

4.3.6. La clase `VisualChartDataSerie`

La clase `VisualChartDataSerie` consiste en la implementación de las indicaciones anteriores, para incorporar el proveedor de datos bursátiles de VisualChart V. El desarrollo se encuentra en la clase `VisualChartDataSerie`, que hereda de `DataSerie` e implementa el método abstracto `get`. Respecto a la biblioteca de enlace dinámico `VisualChartConnector.dll`, tiene que ser almacenada en la misma carpeta donde se encuentra la clase `VisualChartDataSerie`.

Aunque más compleja, la implementación de la conexión a este proveedor de datos ha seguido el mismo esquema que en los casos anteriores. No es casual la idoneidad con que la clase abstracta `DataSerie` se adapta a su labor, pues es el resultado del perfeccionamiento de sucesivas versiones anteriores. Con indiferencia del origen de los datos con que se genera la clase `VisualChartDataSerie`, el resultado es una clase muy similar a las anteriores, que da la sensación de homogeneidad al usuario. Para una descripción detallada de las posibilidades que ofrece la clase `VisualChartDataSerie` es preciso consultar el apéndice A.

4.3.7. La biblioteca de enlace dinámico `VisualChartConnector.dll`

Esta biblioteca de enlace dinámico hace de pasarela para comunicar MATLAB con VisualChart V. La manera en que lo consigue es utilizando el patrón envoltorio. La interfaz COM que implementa MATLAB es un subconjunto de la utilizada por parte de VisualChart V, así que su comunicación directa no es posible. Se eligió la tecnología .NET y el lenguaje C# porque la interfaz COM se soporta de forma nativa. Además, MATLAB es capaz de utilizar las clases .NET a través del intérprete.

La clase `BarValue`

Esta clase surge a partir de la clase `VCDS_BarValue` de la interfaz COM. Contiene información sobre los precios, el volumen de transacciones y el interés abierto en un instante de tiempo dado.

La clase `DataSerie`

Clase homóloga a `VCDS_DataSerie` de la interfaz COM. Consiste en un contenedor que almacena un conjunto de objetos `BarValue`, ordenados de forma cronológica. Esta clase se utiliza para almacenar la información de las series temporales de precios, volúmen e interés abierto de la acción.

El enumerado `enumerates`

Tipo enumerado que sirve para indicar el tipo de compresión temporal con que VisualChart V solicita los datos a su proveedor. Los valores posibles son `Ticks`, `Minutes`, `Days`, `Weeks`, `Months`.

La clase `DataSourceManager`

Clase creada a partir de la clase `VCDS_DataSourceManager` de la interfaz COM. El cometido de la clase `DataSourceManager` es el de gestionar el proceso de creación de objetos del tipo `DataSerie`.

La clase `SafeDataSourceManager`

La clase `SafeDataSourceManager` es una implementación segura derivada a partir de la clase `DataSourceManager`. Esta versión utiliza el patrón singleton[5] para forzar la unicidad y asegurar resultados correctos ante entornos concurrentes.

[5]El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Capítulo 5

Estrategias de trading

En finanzas, una estrategia de trading consiste en un conjunto de reglas, comportamientos y procedimientos basados en modelos matemáticos, que tiene por objetivo orientar al inversor que opera en la bolsa. La estrategia de trading produce señales de compra y venta que ayudan al inversor en la toma de decisiones.

No hay que confundir las estrategias de trading con las estrategias de inversión: en estas últimas, el propósito consiste en elegir una cartera de inversiones, compuesta por una combinación de instrumentos financieros de renta fija y renta variable (entre los que se encuentran las acciones), para ofrecer alta rentabilidad con un nivel de riesgo aceptable. Así, una estrategia de inversión puede elegir invertir parte del capital que gestiona en acciones de una empresa, utilizando las indicaciones de compra y venta arrojadas por una estrategia de trading.

La plataforma ofrece, por medio de la clase `AlgoTrader`, una manera sencilla, rápida y eficiente de implementar cualquier estrategia de trading. Para implementar tales estrategias el usuario sólo debe programar una clase que herede de `AlgoTrader`. Esta clase fue introducida en la sección 3.2 y es detallada de forma técnica en el apéndice A.

El resto del capítulo presenta las estrategias de trading que fueron implementadas en la plataforma. En la sección 5.1 se introducen las estrategias basadas en indicadores técnicos. La sección 5.2 explica los osciladores técnicos y las estrategias de trading implementadas a partir de ellos. Por último, en la sección 5.3 se hace una recopilación de otras estrategias de trading interesantes confeccionadas durante el desarrollo del proyecto.

Gráficos de las máquinas de trading

Antes de comenzar con las máquinas de trading, se explican los gráficos que estas generan y que aparecen con frecuencia a lo largo del capítulo.

La información visual constituye un apoyo imprescindible que aglutina un ingente de datos en apenas espacio. A través del uso de representaciones visuales adecuadas, se busca hacer simple la comprensión global de los resultados. Además, los gráficos entre máquinas son similares, con la intención de facilitar las comparaciones. Estos gráficos están constituidos por diferentes recuadros, cada uno de los cuales se centra en la representación de ciertos valores. Los recuadros aparecen dispuestos en columna y comparten el intervalo de tiempo: de

esta forma, se sigue la progresión de todos los valores dibujados trazando una línea vertical imaginaria que avanza hacia la derecha. Como ejemplo, en la figura 5.1 se muestra el gráfico de una máquina de trading donde, como se aprecia, aparecen tres recuadros:

Serie/Position Este recuadro muestra la curva del precio de la acción a lo largo del tiempo.

Position Aquí aparecen, representadas como rectángulos, las posiciones en el tiempo que adopta la estrategia: estas pueden ser largas o cortas, aunque también se contempla la posibilidad de permanecer fuera del mercado. La longitud de los rectángulos es relativa al tiempo que dura la posición en el mercado; por su parte, la altura es proporcional a la cantidad que ha invertido la máquina.

Profit/Loss Recuadro que contiene la curva del beneficio o pérdida acumulada en el tiempo. Representa la cantidad en tanto por uno, donde la unidad corresponde a la inversión inicial.

El código de colores para el dibujado de las curvas y los rectángulos es importante pues, para cada instante, indica la posición que mantiene la máquina de trading: el verde se usa para indicar posición larga, el rojo para corta y el negro señala que se permanece fuera del mercado. En lo referente a la escala, se utiliza la lineal en todos los recuadros.

Además de los anteriores, cada máquina de trading define sus propios recuadros para mostrar la información conveniente. En la sección 5.2 por ejemplo, las máquinas definen un nuevo recuadro en el que presentan el valor del oscilador en cada instante.

5.1. Basadas en indicadores técnicos

Un indicador técnico consiste en una serie de valores calculados al aplicar una fórmula sobre el precio de la acción. La información del precio incluye cualquier combinación de los precios de apertura, más alto, más bajo y de cierre en un periodo de tiempo. Algunos indicadores sólo emplean los precios de cierre, mientras que otros incorporan el volumen y el interés abierto en sus fórmulas. Los indicadores técnicos se basan en los principios y suposiciones del análisis técnico.

Con un indicador técnico se ofrece una perspectiva diferente desde la que analizar la acción del precio. Algunos indicadores, como la media móvil, se obtienen con fórmulas simples y su interpretación es fácil de entender. Otros, como el estocástico, se calculan con fórmulas más complejas y requieren más tiempo de estudio para entenderlos por completo. Con independencia de la complejidad de la fórmula, los indicadores técnicos ofrecen una perspectiva única sobre la fuerza y dirección de la acción del precio.

Los indicadores técnicos cumplen tres funciones principales: alertar sobre eventos importantes que se producen como consecuencia de las variaciones en el precio de la acción, confirmar los resultados de otros indicadores técnicos para comprender la acción del precio y predecir la dirección futura del precio.

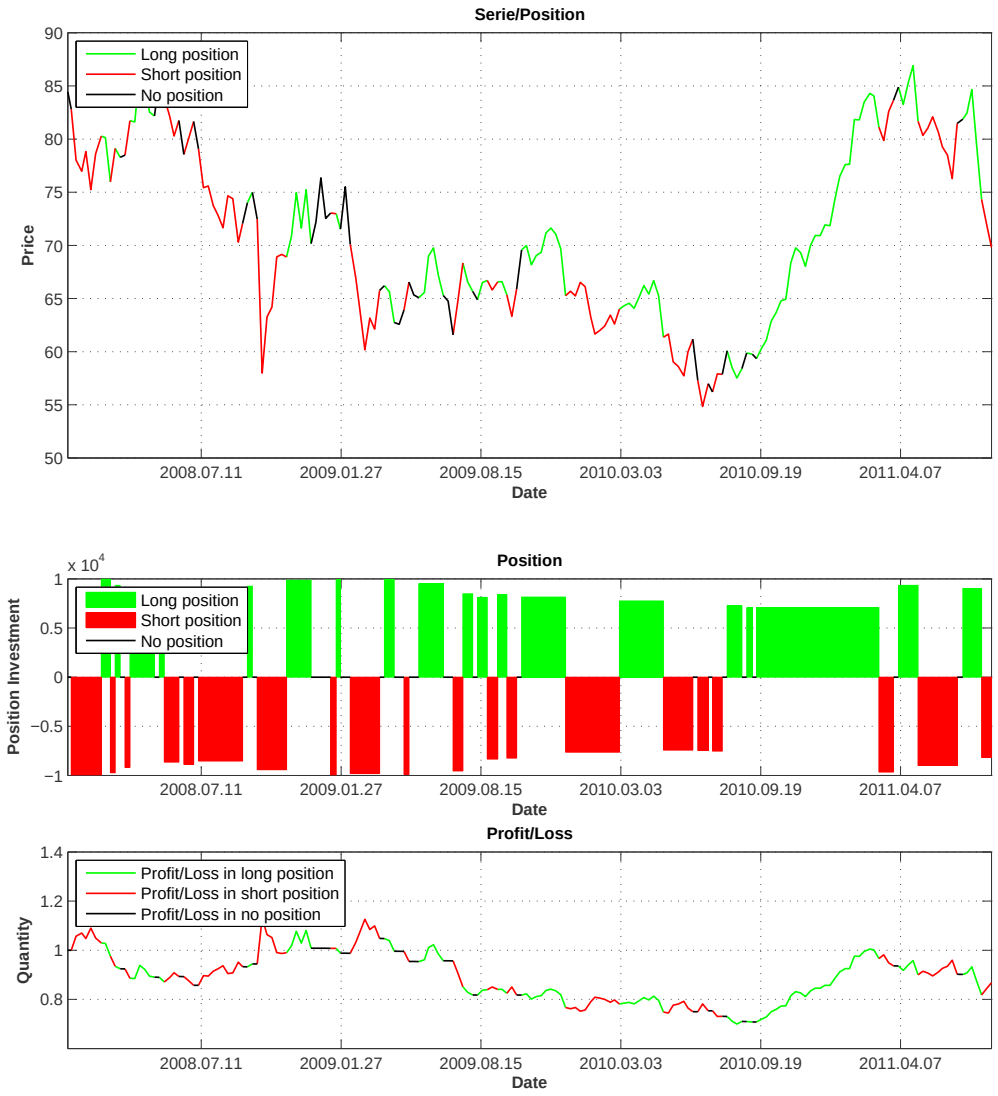

Figura 5.1: Gráfico de una máquina de trading

En esta parte del documento se introducen algunos de los indicadores técnicos más utilizados, y para cada uno de ellos se explica cómo es la clase que lo implementa.

En lo que a la programación se refiere, todas las clases que implementan estrategias basadas en indicadores técnicos heredan de la clase `Indicator` o de algún descendiente suyo, que a su vez hereda de `AlgoTrader`. De esta manera, cualquier cambio que se realiza sobre la clase `Indicator` afecta a todas las máquinas de esta familia.

5.1.1. Media móvil

La media móvil es un indicador técnico que identifica la dirección en la tendencia del precio de la acción [20, 21]: tal tendencia en el precio de una acción puede ser alcista, bajista o lateral. No predice la dirección del precio, sino que define la dirección de la tendencia con un retraso. El retraso se debe a que la media móvil utiliza valores pasados del precio. A pesar del retraso, la media movil ayuda a suavizar la serie de valores del precio y filtra el ruido en los datos. Por ser seguidora de tendencias, resulta más útil cuando existe una dirección clara del mercado, es decir, cuando hay una tendencia alcista o bajista. En momentos de tendencia lateral es preferible guiarse por las señales que ofrecen los osciladores. Los sistemas seguidores de tendencia se caracterizan por estar siempre en mercado: pueden estar posicionados largos o cortos, pero nunca estan fuera del mercado.

La media móvil es el punto de partida de otros muchos indicadores y osciladores técnicos. Los dos tipos más comunes de medias móviles son la media móvil simple y la media móvil exponencial. En inglés se conocen como *Simple Moving Average* o SMA y *Exponential Moving Sverage* o EMA.

Formulación del modelo

La media móvil simple se forma calculando el precio medio de la acción en un número específico de periodos. Se utiliza como base el precio de cierre de la acción. Como su nombre indica, la media móvil es una media que se mueve: nuevos resultados se producen con la entrada de nuevos datos. Esto hace que la media se mueva con el tiempo. La ecuación 5.1 recoge el comportamiento mencionado, siendo p el número de periodos.

$$SMA_i = \frac{x_{i-p+1} + \ldots + x_{i-1} + x_i}{p} \tag{5.1}$$

En la media móvil simple todos los datos de entrada x_j, que se corresponden con precios de la acción en los diferentes periodos, tienen el mismo peso, es decir, no se discrimina entre más recientes y más antiguos, por lo que la contribución a la media de cada valor es la misma. La media móvil exponencial reduce el tiempo que tarda en adaptarse al precio respecto a la simple asignando un mayor peso a los valores del precio más recientes. El peso asignado a los valores de precio más recientes depende del número de periodos que se toman. El cálculo de la media móvil exponencial es recursivo y para el caso base emplea el valor de la media móvil simple. La ecuación 5.2 muestra la manera en que se calcula el caso base y el caso recursivo, con p el número de periodos.

$$EMA_i = \begin{cases} SMA_i = \frac{x_{i-p+1}+...+x_{i-1}+x_i}{p} & ,\text{si } (i = p) \\ x_i \cdot K + EMA_{i-1} \cdot (1-K) \quad ,K = \frac{2}{1+p} & ,\text{si } (i > p) \end{cases} \tag{5.2}$$

Tanto en el caso de la media móvil simple como en la exponencial, los valores que toman para aquellos casos en que $i < p$ no están definidos.

Generación de la señal

El precio de la acción se mueve con mayor velocidad que la media móvil. Si el precio está por debajo de la media móvil y en un instante cruza hacia arriba, entonces esta podría ser una indicación de que la tendencia en el mercado va a cambiar y comenzará a ser alcista. De misma manera, también se puede pensar que cambiará la tendencia si el precio de la acción está por encima del valor de la media móvil y luego la cruza hacia abajo, siendo el inicio de una tendencia bajista. Estos cruces entre la media móvil y el precio de la acción se pueden utilizar para generar señales de compra y venta con las que posicionarse largo o corto en el mercado. La señal se genera tal y como se indica en la ecuación 5.3 para cada periodo. Se utiliza el parámetro XMA_i para referirse indistintamente a SMA_i y a EMA_i.

En la figura 5.2 aparece la representación del valor de una acción a lo largo del tiempo acompañanda de la media móvil exponencial en color azul. Como se aprecia, la curva que representa al precio es más abrupta y sus movimientos más rápidos; por el contario, la media móvil describe una curva más suave que marca la tendencia del precio. Mientras el precio de la acción está por encima del valor de la media móvil, la señal que produce la estrategia es de posición larga; si se da el caso contrario donde el precio de la acción está por debajo de la media móvil, entonces la estrategia adopta una posición corta. En los intervalos temporales en que no existe una tendencia clara del precio los resultados de esta estrategia son deficientes.

Sobre la figura 5.2, cabe destacar que la curva relativa a la media móvil no comienza a la misma par que la curva del precio de la acción. Esto es así por la manera en que se calcula la media móvil, pues el primer valor que define la fórmula es aquel con índice $i = p$, siendo p el periodo. Para los índices tales que $i < p$, el valor es indefinido. Esta situación se da en todas las estrategias que utilizan fórmulas cuyo cálculo requiere de valores pasados.

$$Señal_i = \begin{cases} +1 & ,\text{si } precio_i > XMA_i \\ -1 & ,\text{si } precio_i < XMA_i \end{cases} \tag{5.3}$$

La clase `MovingAverage`

Esta clase implementa la estrategia de trading basada en la media móvil simple o la media móvil exponencial. La clase hereda de `Indicator`, que a su vez hereda de la clase `AlgoTrader`. El conjunto de propiedades y métodos de la clase es el siguiente:

`Mode` Propiedad que indica el tipo de media móvil, con el valor `'s'` para la simple y `'e'` para la exponencial.

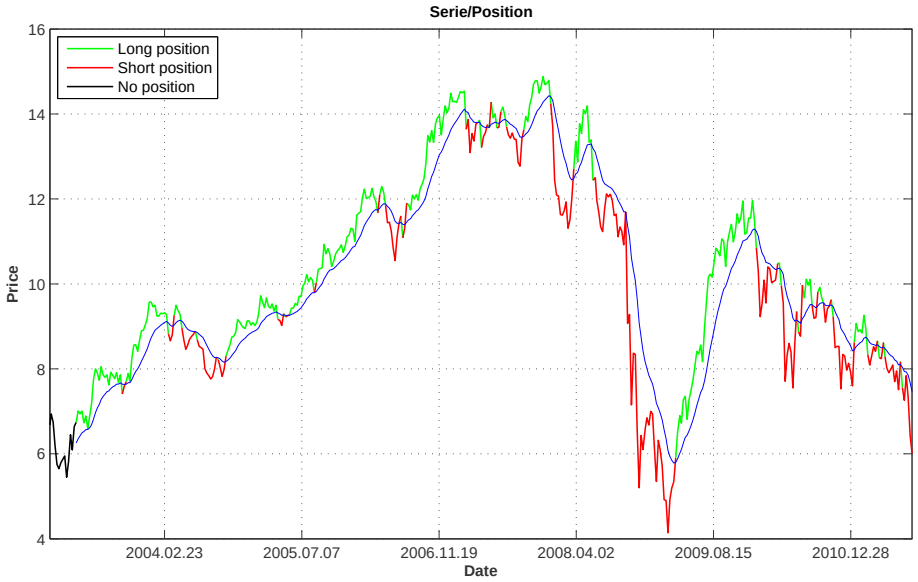

Figura 5.2: Media móvil exponencial

`Samples` Propiedad para establecer el número de periodos en el calculo de la media móvil. Se corresponde con el parámetro p que aparece en las ecuaciones 5.1 y 5.2.

`computeSignal` Método de la clase que computa la señal de posición según aparece en la ecuación 5.3. Los precios de cotización de la acción se leen desde el objeto de tipo `DataSerie` asociado a la máquina. El código 5.1 muestra una manera de implementar el método. Para el cálculo de los diferentes tipos de medias móviles se utiliza la función `movavg`, incluida la plataforma.

`drawSeriePosition` Método que decora los gráficos de la curva de precios superponiendo el trazado de la media móvil. De esta manera, queda personalizada la representación visual que ofrecen los métodos `plot` y `plotSeriePosition`. El gráfico que presenta la máquina de trading, por medio del método `plot`, se muestra en la figura 5.3. Por su parte, el método `plotSeriePosition` produce gráficos como el de la figura 5.2.

5.1.2. Media móvil desplazada

Consiste en una media móvil ajustada en el tiempo por medio de un desplazamiento [20, 21, 22]. Se construye tomando la media móvil y desplazandola un número de intervalos. El resultado consiste en una curva suave y desplazada que acompaña al precio, como aparece en la figura 5.4. Esta estategia introduce un retardo en el efecto que tienen las variaciones del precio de la acción.

```
function computeSignal(algoTrader)

% Parámetros
serie = algoTrader.DataSerie.Serie;
mode = algoTrader.Mode;
samples = algoTrader.Samples;

% Media móvil
movingAverage = movavg(serie, mode, samples);

% Señal
signal = zeros(1, length(serie));
signal(serie>movingAverage) = 1;
signal(serie<movingAverage) = -1;

% Actualiza el valor de la propiedad Signal
algoTrader.Signal = signal;

end
```

Código 5.1: Método `computeSignal` de la clase `MovingAverage`

Formulación del modelo

El cálculo es igual al que indica la ecuación 5.1 para la media móvil simple y la ecuación 5.2 para la media móvil exponencial. La repercusión del desplazamiento afecta al cálculo de la señal.

Generación de la señal

La señal se genera de forma similar a como se hace en la media móvil, pero tomando un valor pasado de la media para comparar con el precio vigente de la acción, tal y como indica la ecuación 5.4. El desplazamiento al que se somete la media móvil es de un número de d intervalos.

$$Señal_i = \begin{cases} +1 & \text{, si } precio_i > XMA_{i-d} \\ -1 & \text{, si } precio_i < XMA_{i-d} \end{cases} \tag{5.4}$$

La clase `MovingAverageDisplaced`

La clase `MovingAverageDisplaced` implementa la estrategia descrita en esta sección. Esta clase tiene como antecesores a las clases `Indicator` y `AlgoTrader` ya que hereda de `MovingAverage`. Los siguientes son los métodos y propiedades que implementa la clase:

`Displacement` Propiedad que determina el desplazamiento de la media móvil. No puede tomar valores negativos, pues esto supone conocer los valores futuros del precio de la acción. En la ecuación 5.4, esta propiedad se corresponde con el parámetro d.

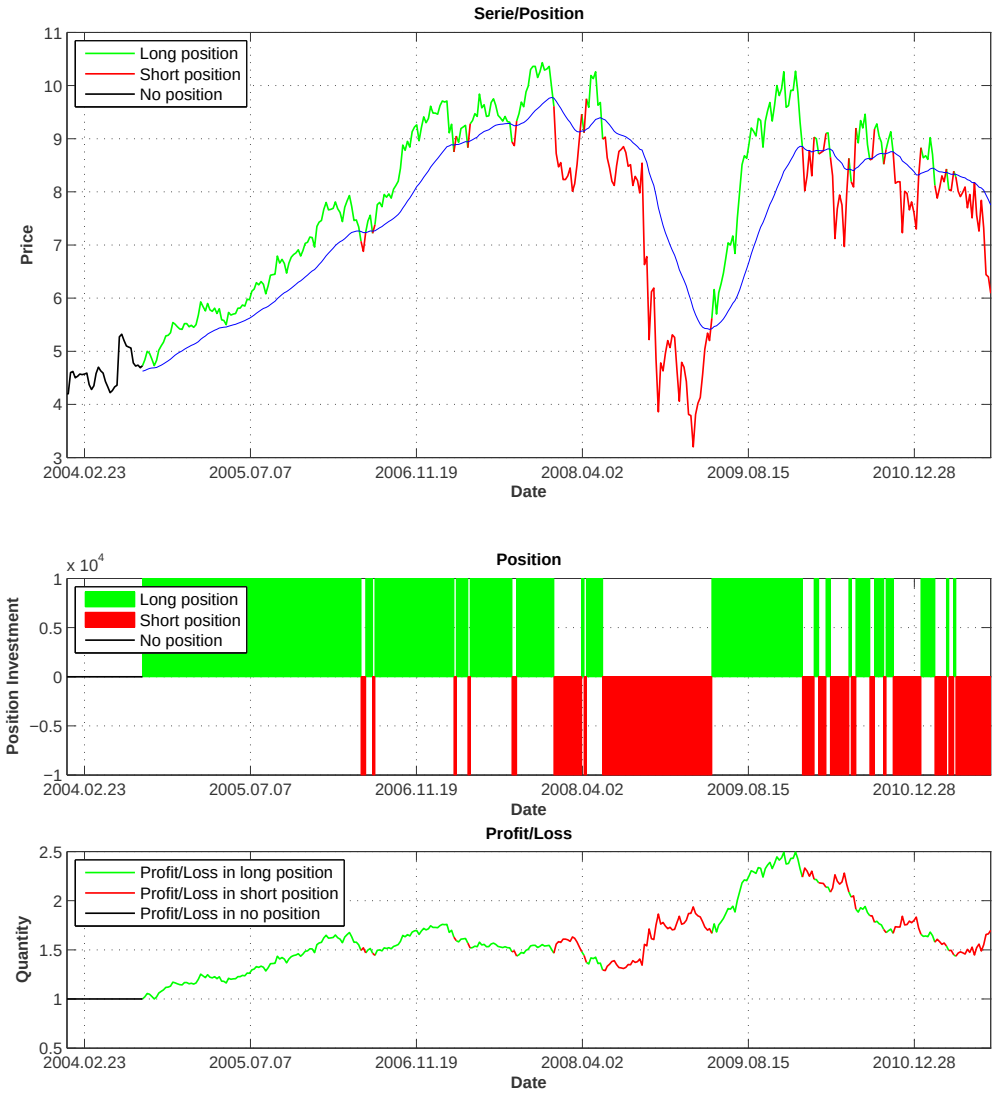

Figura 5.3: Gráfico de la máquina de trading `MovingAverage`

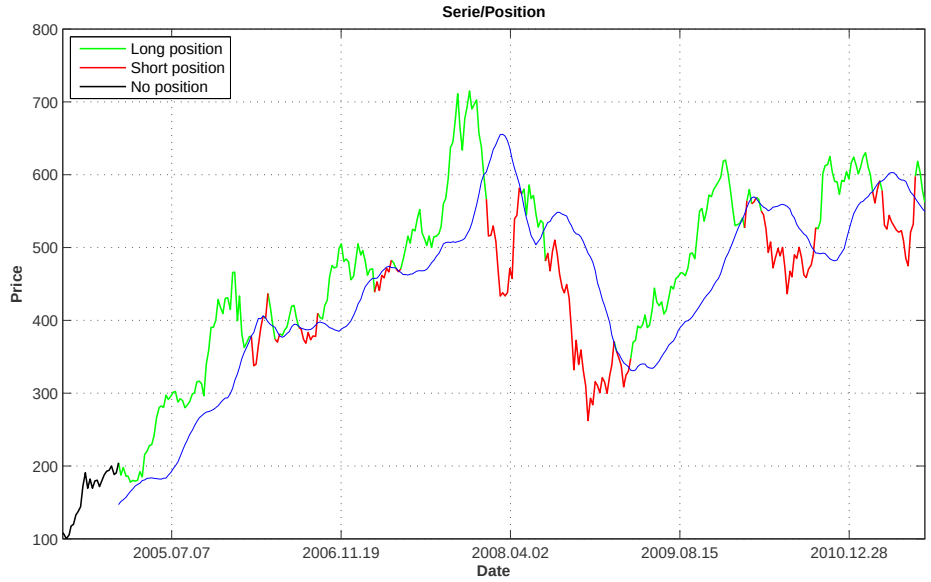

Figura 5.4: Media móvil exponencial desplazada

`computeSignal` Método que computa la señal de la máquina de trading según la ecuación 5.4.

`drawSeriePosition` Método que decora los gráficos de la serie de precios con la media móvil desplazada para ofrecer un resultado como el que aparece en la figura 5.4. Altera el comportamiento de los métodos `plotSeriePosition` y `plot`.

5.1.3. Media móvil con umbral

La media móvil con umbral, que es una mejora de la media móvil, introduce un nivel de tolerancia en los cambios de posición de la señal [20, 21]. De esta manera, se reduce el efecto negativo de las oscilaciones en el precio de la acción, que cruzando una y otra vez sobre la media móvil no se separa de ella de forma considerable: reduce el problema de la tendencia lateral. Como extensión, esta técnica hereda las demás bondades y limitaciones de la media móvil.

Formulación del modelo

El cálculo es idéntico al de la media móvil. Se mantienen las dos modalidades: media móvil simple, calculada a partir de la ecuación 5.1 y media móvil exponencial, con la ecuación 5.2. La aplicación de los umbrales no interviene durante el proceso del cálculo de la media móvil, sino que toma parte en el instante de generar la señal.

Generación de la señal

La fórmula del cálculo de la señal, que cambia respecto a la utilizada para la media móvil, hace uso de un par de umbrales: con ellos se regula la tolerancia a las pequeñas fluctuaciones

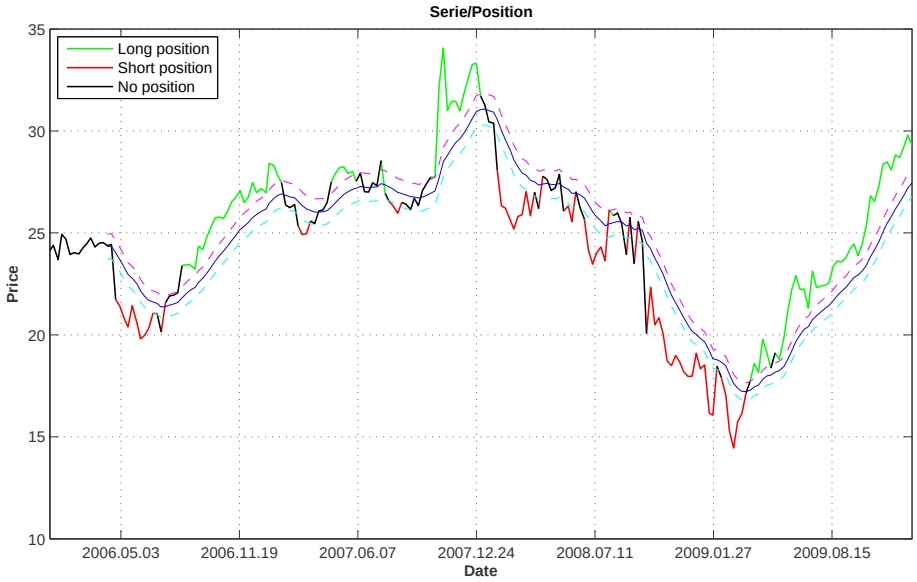

Figura 5.5: Media móvil exponencial con umbral

que se producen durante los intervalos de tendencia lateral. Un umbral es el de subida, que multiplica a la media móvil para establecer una cota inferior. Si esta cota es superada por el precio, entonces la estrategia adopta una posición larga. El otro umbral es el de descenso, que fija la cota superior al multiplicar a la media móvil. Si el precio de la acción es inferior a la cota superior, entonces la estrategia toma una posición corta. La ecuación 5.5 resume la manera en que la señal es generada en cada periodo.

En la figura 5.5 se muestra el precio de la acción acompañando de la media móvil exponencial en color azul y las cotas superior e inferior en colores cian y magenta respectivamente. Cabe resaltar que la cota inferior está por encima de la cota superior. Esto es así porque la cota inferior marca el mínimo valor a partir del cual se toma una posición larga, mientras que la cota superior indica el máximo valor por debajo del cual se mantiene una posición corta. La media móvil suaviza el movimiento de los precios de la acción y las cotas ayudan a evitar que la estrategia entre en el mercado en momentos de alta volatilidad e incertidumbre.

Si se asignan valores de umbral tales que $umbralSubida > 1 > umbralDescenso$, entonces la estrategia no está siempre en el mercado, sino que adopta un comportamiento más conservador. Esto se aprecia en la figura 5.5, donde las secciones en negro de la curva del precio indican momentos en que la estrategia está fuera del mercado. Si el umbral de subida es inferior al de descenso, entonces la máquina de trading no se posiciona en el mercado.

$$Señal_i = \begin{cases} +1 & \text{, si } precio_i > cota\ inferior = umbralSubida \cdot XMA_i \\ -1 & \text{, si } precio_i < cota\ superior = umbralDescenso \cdot XMA_i \\ 0 & \text{, en otro caso} \end{cases} \tag{5.5}$$

La clase `MovingAverageThreshold`

La clase `MovingAverageThreshold` implementa la estrategia de trading de media móvil con umbral. Esta clase hereda de `MovingAverage`, luego sus antecesores son las clases `Indicator` y `AlgoTrader`. Las propiedades y los métodos de la clase aparecen a continuación:

`RiseThreshold` Propiedad que indica el umbral de subida. En la ecuación 5.5 se corresponde con el parámetro *umbralSubida*.

`FallThreshold` Propiedad que establece el umbral de descenso. En la ecuación 5.5 aparece como el parámetro *umbralDescenso*.

`Threshold` Propiedad dependiente de sólo escritura que, al asignarle un valor, actualiza las propiedades `RiseThreshold` y `FallThreshold` asignando a ambas dicho valor. Además de unificar ambos umbrales en un mismo valor, esta propiedad sirve para desplazar a lo largo del eje vertical el efecto de la media móvil. De esta manera, puede relajarse o hacerse más estricta la condición para entrar en posiciones largas o cortas.

`computeSignal` Método que computa la señal de posición de acuerdo a la ecuación 5.5.

`drawSeriePosition` Método para personalizar la representación visual de la información provista por la máquina. Modifica el comportamiento del método `plotSeriePosition`, que produce gráficos como el de la figura 5.5. También altera el funcionamiento del método `plot`, que genera una representación como la que se ve en la figura 5.6.

5.1.4. Cruce de medias móviles

Esta estrategia se basa en dos medias móviles que difieren en el valor de sus periodos: el valor mayor corresponde a la media móvil larga y el valor menor es el de la media móvil corta. Consiste en una extensión de la estrategia de media móvil donde la media móvil larga identificar la tendencia y la media móvil corta aporta estabilidad a la estrategia [20, 21]; así se reduce el impacto negativo de las oscilaciones del precio de la acción. Como en el caso de la estrategia de la media móvil, esta estrategia es deficiente cuando la tendencia en el mercado es lateral.

Formulación del modelo

Para el cálculo de las dos medias móviles, una vez más se utiliza la descripción provista por la ecuación 5.1 para el caso de la media móvil simple y la ecuación 5.2 para la exponencial. El número de periodos de la media móvil corta tiene que ser menor que el de la media móvil larga.

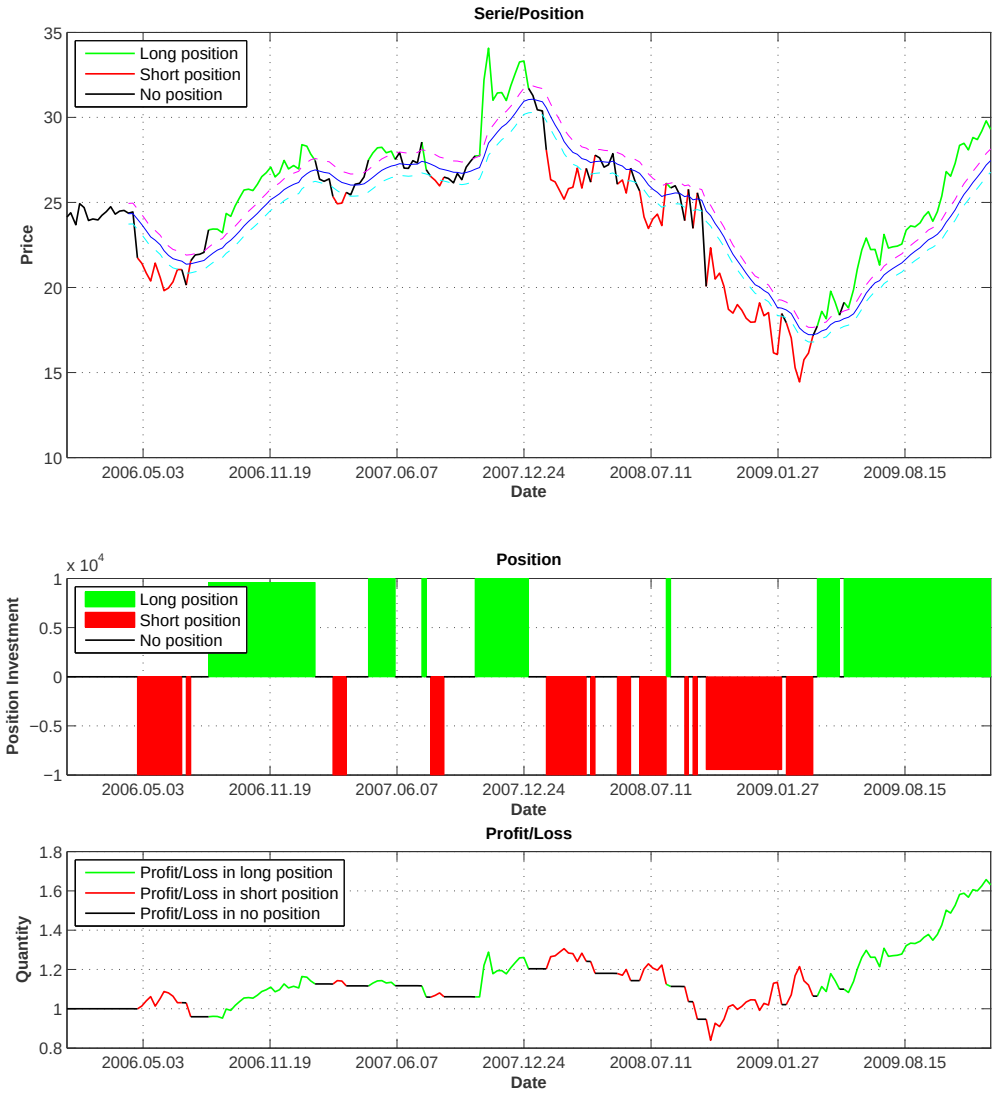

Figura 5.6: Gráfico de la máquina de trading `MovingAverageThreshold`

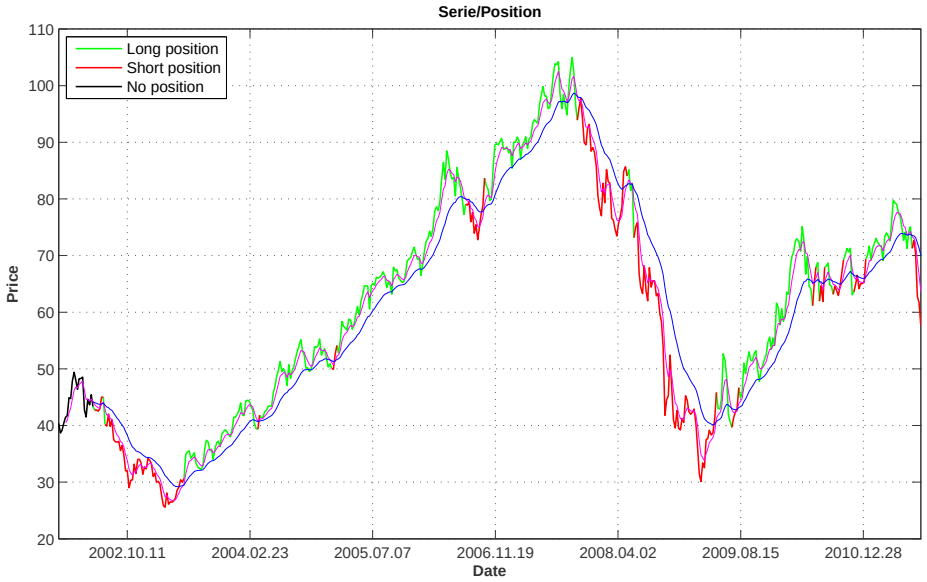

Figura 5.7: Cruce de medias móviles exponenciales

Generación de la señal

El precio de la acción suele presentar fuertes oscilaciones. La media móvil corta suaviza un poco estos movimientos bruscos, para así conseguir una curva más estable. La media móvil larga, con una cantidad mayor de periodos, consigue suavizar más aún la curva del precio y marca además la tendencia que se presenta. En combinación, los cruces entre estas medias sirven para generar la señal con que la máquina de trading interviene en el mercado. La estrategia adopta una posición larga cuando la media móvil corta es mayor a la media móvil larga y una posición corta cuando la relación es la contraria, es decir, cuando la media móvil corta es menor a la media móvil larga. La especificación sobre cómo se genera la señal se muestra de manera formal en la ecuación 5.6.

El comportamiento de la estrategia se observa en el ejemplo mostrado en la figura 5.7. La curva del precio de la acción aparece como la más irregular, acompañada por la media móvil corta, en color magenta, y la media móvil larga, en color azul. Se comprueba que los cruces entre las dos medias originan el cambio de posición en la estrategia. Debido al número de periodos de cada una, las medias móviles comienzan en instantes diferentes en la figura.

$$Señal_i = \begin{cases} +1 & ,\text{si } XMA\_corta_i > XMA\_larga_i \\ -1 & ,\text{si } XMA\_corta_i < XMA\_larga_i \end{cases} \tag{5.6}$$

La clase `MovingAveragesCrossing`

En la clase `MovingAveragesCrossing` se implementa la estrategia del cruce de medias móviles. Esta clase hereda de `Indicator`, que a su vez lo hace de `AlgoTrader`. Las propiedades y métodos de la clase son:

```
function computeSignal(algoTrader)

% Parámetros
serie = algoTrader.DataSerie.Serie;
mode = algoTrader.Mode;
lead = algoTrader.Lead;
lag = algoTrader.Lag;

% Media móvil
leadMovingAverage = movavg(serie, mode, lead);
lagMovingAverage = movavg(serie, mode, lag);

% Señal
signal = zeros(1, length(serie));
signal(leadMovingAverage>lagMovingAverage) = 1;
signal(leadMovingAverage<lagMovingAverage) = -1;

% Actualiza el valor de la propiedad Signal
algoTrader.Signal = signal;

end
```

Código 5.2: Método `computeSignal` de la clase `MovingAveragesCrossing`

`Mode` Propiedad para indicar el tipo de media móvil: toma el valor `'s'` para la media móvil simple y `'e'` para la media móvil exponencial. En la ecuación 5.6, XMA_corta_i y XMA_larga_i se refieren a las medias móviles corta y larga con indiferencia del tipo.

`Lead` Propiedad con el número de periodos de la media móvil corta. Se corresponde con el parámetro p que aparece en las ecuaciones 5.1 y 5.2.

`Lag` Propiedad que guarda la cantidad de periodos de la media móvil larga. Se refiere al parámetro p de las ecuaciones 5.1 y 5.2. Si no se cumple la condición $Lead < Lag$, entonces la máquina permanece fuera del mercado en todo momento.

`computeSignal` Método que computa la señal de la máquina de acuerdo a la ecuación 5.6. Una posible implementación del método se encuentra en el código 5.2.

`drawSeriePosition` Método para decorar los gráficos. Superpone las curvas de las medias móviles al gráfico del precio de la acción. Altera el comportamiento del método `plotSeriePosition`, que pinta gráficos como el de la figura 5.7. También modifica el método `plot`, que dibuja la información de acuerdo a como aparece en la figura 5.8.

Figura 5.8: Gráfico de la máquina de trading `MovingAveragesCrossing`

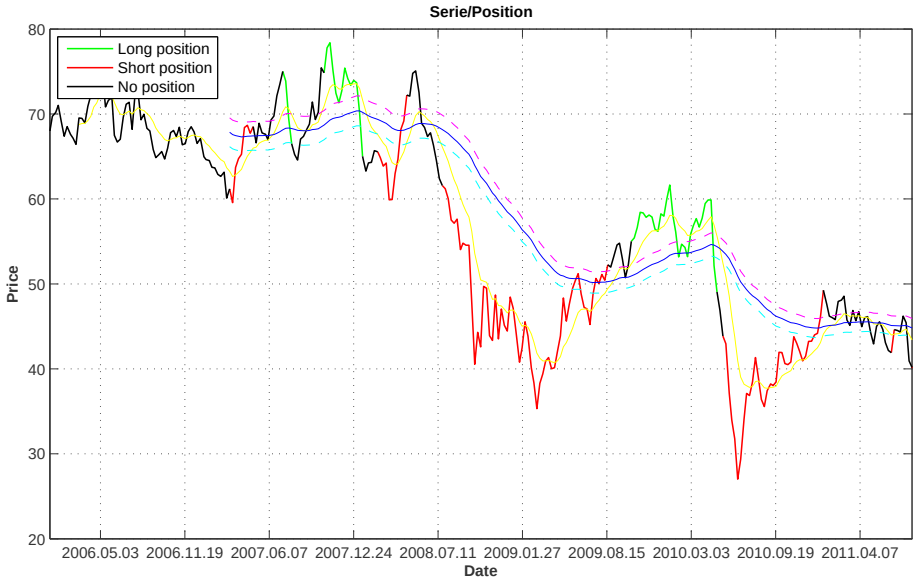

Figura 5.9: Cruce de medias móviles exponenciales con umbral

5.1.5. Cruce de medias móviles con umbral

El cruce de medias móviles con umbral es una extensión del cruce de medias móviles. Hace uso de dos valores de umbral para admitir cierta tolerancia y evitar oscilaciones indeseables de la media móvil corta sobre la media móvil larga. El uso del umbral reduce el efecto negativo de las fuertes oscilaciones del precio durante la tendencia lateral. De esta manera, la máquina de trading evita el oscilar en posiciones opuestas en un espacio de tiempo corto.

Formulación del modelo

El componente base de la estrategia sigue siendo la media móvil, con la ecuación 5.1 para la simple y la ecuación 5.2 para la exponencial. Igual que en el caso de la estrategia anterior, la media móvil corta tiene que utilizar un número de periodos menor al de la media móvil larga. La aplicación del umbral se hace durante el cálculo de la señal.

Generación de la señal

En la generación de la señal intervienen la media móvil corta, la larga, el umbral de subida y el de descenso. La manera de generar la señal es la siguiente: si la media móvil corta es mayor a la media móvil larga multiplicada por el umbral de subida, entonces se toma una posición larga; en caso de que la media móvil corta sea menor a la media móvil larga multiplicada por el umbral de descenso, entonces se adopta una posición corta. Esto se resume en la ecuación 5.7.

En la figura 5.9 se muestra el comportamiento de la estrategia. La curva de diferentes colores corresponde al precio. En amarillo aparece la media móvil corta y en azul, la larga. Las líneas discontinuas magenta y cian son las cotas inferior y superior.

$$Señal_i = \begin{cases} +1 & \text{, si } XMA\_corta_i > cota\ inferior = umbralSubida \cdot XMA\_larga_i \\ -1 & \text{, si } XMA\_corta_i < cota\ superior = umbralDescenso \cdot XMA\_larga_i \\ 0 & \text{, en otro caso} \end{cases} \tag{5.7}$$

La clase `MovingAveragesCrossingThreshold`

La clase `MovingAveragesCrossingThreshold`, que implementa esta estrategia, hereda de la clase `MovingAveragesCrossing`. Cuenta con los métodos y propiedades siguientes:

`RiseThreshold` Propiedad que contiene el valor del umbral de subida. Se refiere al parámetro *umbralSubida* de la ecuación 5.7.

`FallThreshold` Propiedad con el umbral de descenso que interviene en el cálculo de la señal. En la ecuación 5.7 se refiere a *umbralDescenso*.

`Threshold` Propiedad de sólo lectura que unifica el valor de las propiedades `RiseThreshold` y `FallThreshold` al asignado. Sirve para desplazar arriba o abajo la media móvil, esto es, relaja o refuerza la condición para entrar en posiciones largas o cortas.

`computeSignal` Método que calcula la señal de posición de acuerdo a la ecuación 5.7.

`drawSeriePosition` Método que decora la representación visual de la clase. Dibuja las curvas de las medias móviles y las cotas en los gráficos. Altera el comportamiento del los métodos `plotSeriePosition` y `plot`. El gráfico que aparece en la figura 5.9 es consecuencia del método `plotSeriePosition`.

5.1.6. Bandas de Bollinger

Las Bandas de Bollinger representan la volatilidad y se sitúan por encima y por debajo de la media móvil. La volatilidad se define en términos de la desviación estándar sobre el precio de la acción. El propósito de las Bandas de Bollinger es facilitar una definición relativa de alto y bajo. Por definición, los precios son altos en la banda superior y bajos en la banda inferior. Las Bandas de Bollinger determinan la fuerza de la tendencia en el precio de la acción.

Formulación del modelo

La media móvil que utiliza esta técnica es la simple, cuya fórmula aparece en la ecuación 5.1. La banda superior se calcula a partir de la ecuación 5.8 y la banda inferior, con la ecuación 5.9. En estas fórmulas aparece el factor K, que regula la amplitud de la zona encerrada entre las bandas. En ambos casos se necesita calcular la desviación estándar, que se hace según aparece en la ecuación 5.10, donde $\overline{precio_{i-p+1:i}}$ es la media de los precio de la acción en los últimos p periodos. Esta ecuación se puede definir en términos de la media móvil simple, como aparece en la última igualdad.

$$BandaSuperior_i = SMA_i + K \cdot \sigma_i \tag{5.8}$$

$$BandaInferior_i = SMA_i - K \cdot \sigma_i \tag{5.9}$$

$$\sigma_i = \sqrt{\frac{1}{p} \cdot \sum_{j=0}^{p-1} (precio_{i-j} - \overline{precio_{i-p+1:i}})^2} = \sqrt{\frac{1}{p} \cdot \sum_{j=0}^{p-1} precio_{i-j}^2 - SMA_i} \tag{5.10}$$

Generación de la señal

Las Bandas de Bollinger reflejan la dirección con la media móvil simple y la volatilidad con las bandas superior e inferior. Como tal, se pueden utilizar para determinar si el precio está relativamente alto o bajo. De acuerdo a su creador, John Bollinger [23], gran parte del movimiento del precio de la acción está contenido entre las bandas, luego aquellos movimientos que se producen fuera de ellas son significativos. En el sentido técnico, los precios están relativamente altos cuando sobrepasan la banda superior y bajos cuando caen por debajo de la banda inferior.

Una manera en que se utiliza esta técnica es produciendo señales de venta cuando el precio sobrepasa la banda superior, ante la creencia de que la acción está sobrecomprada y hay una burbuja. Por el contrario y siguiendo un razonamiento similar, se emite una señal de compra cuando el precio cae por debajo de la banda inferior, pues la acción está sobrevendida y tiene un precio inferior al que le corresponde. Sin embargo, los precios pueden ser altos o bajos por alguna razón subyacente. Como con los otros indicadores, las Bandas de Bollinger no deben utilizarse solas sino en combinación. En resumen, el cálculo de la señal se realiza según indica la ecuación 5.11.

Una imágen representativa del uso de esta estrategia y la señal que produce aparece en la figura 5.10. Ahí se muestra el precio como una curva en colores verde, rojo y negro, que identifica las diferentes posiciones que adopta la máquina de trading en el tiempo. Decorando el gráfico aparece la curva azul de la media móvil simple. En magenta aparece la curva de la banda superior y en cian la de la banda inferior.

$$Señal_i = \begin{cases} +1 & \text{, si } precio_i < BandaInferior_i \\ -1 & \text{, si } precio_i > BandaSuperior_i \\ 0 & \text{, en otro caso} \end{cases} \tag{5.11}$$

La clase `BollingerBands`

La clase `BollingerBands` implementa la estrategia basada en las Bandas de Bollinger. Esta clase hereda de `Indicator`, luego tiene como antecesor a la clase `AlgoTrader`. Las propiedades y métodos que implementa la clase son:

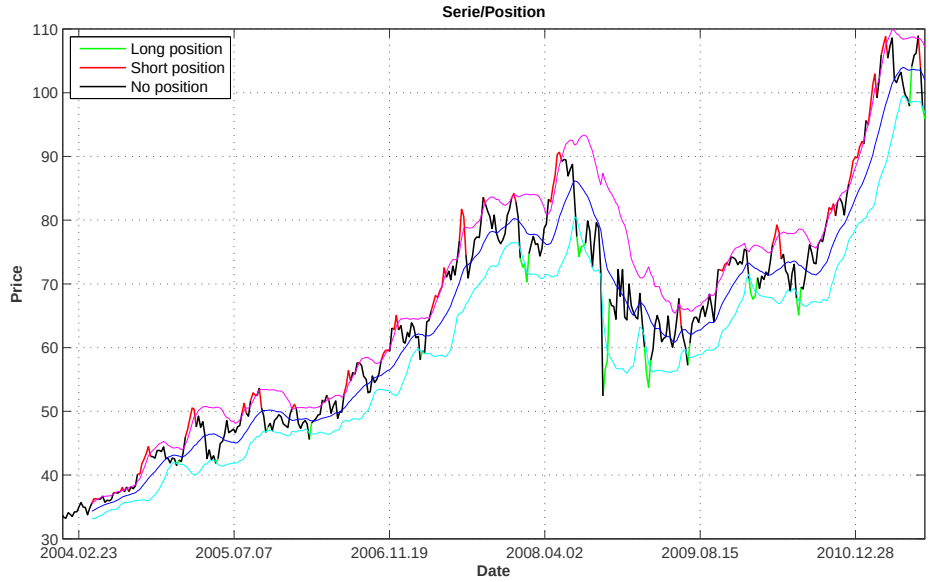

Figura 5.10: Bandas de Bollinger

`Samples` Propiedad que especifica el número de periodos a tomar en el cálculo de la media móvil y la desviación estándar. Se corresponde con el parámetro p en la ecuación 5.10.

`K` Propiedad que contiene el valor del factor que multiplica a la desviación estándar en el cálculo de las bandas. Aparece como K en las ecuaciones 5.8 y 5.9.

`computeSignal` Método que aplica la estrategia descrita en el cálculo de la señal de posición. Lo hace de acuerdo a las indicaciones de la ecuación 5.11.

`drawSeriePosition` Método que decora el gráfico de la curva del precio de la acción con las bandas de Bollinger y la media móvil. Altera la representación provista por los métodos `plotSeriePosition` y `plot`. En particular, la figura 5.10 es producto del método `plotSeriePosition`.

5.2. Basadas en osciladores técnicos

Los osciladores son modelos matemáticos aplicados al precio, basados en alguna observación específica sobre el comportamiento del mercado. Son un tipo especial de indicadores que oscilan en el tiempo por encima y por debajo de una línea central o entre unos niveles. Los osciladores técnicos se sustentan en las hipótesis sostenidas por el análisis técnico.

Normalmente, la curva del oscilador no se dibuja superpuesta a la del precio, sino que se representa en un recuadro independiente. Los ociladores no se expresan en las mismas unidades que el precio de la acción, sino que los valores que toman carecen de unidades y están acotados.

Esta parte del documento trata sobre la explicación de algunos de los osciladores técnicos más utilizados y su implementación como máquinas de trading.

Todas las máquinas de esta familia, que implementan estrategias basadas en osciladores técnicos, heredan de la clase abstracta `Oscillator` o de alguna clase heredera suya. En la jerarquía de clases de la plataforma, la clase `Oscillator` hereda de `AlgoTrader`. Al compartir estas máquinas una misma clase antecesora, los cambios aplicados a la clase `Oscillador` repercuten en todas ellas.

La clase abstracta `Oscillator` es algo más que el antecesor en común de las estrategias basadas en osciladores. Esta clase redefine y aporta métodos propios respecto a la clase `AlgoTrader`. Los métodos en cuestión son los siguientes:

`plot` Este método dibuja el resultado de la máquina de trading. El gráfico generado es similar al del método `plot` de la clase `AlgoTrader`, con la salvedad de que incorpora un recuadro adicional para mostrar la curva de valores relativa al oscilador. El contraste se aprecia al comparar las figuras 5.8 y 5.12.

`plotOscillator` Crea el recuadro con la representación gráfica del oscilador. El método de dibujado `plot` lo utiliza para la creación de los graficos de la máquina.

`plotSeriePositionOscillator` Muestra un gráfico donde sólo aparecen los recuadros del precio de la acción y del oscilador, colocados uno sobre el otro. La vista que ofrece es similar a la del método `plot`, pero ignorando los dos últimos recuadros.

`drawOscillator` Método abstracto que realiza el dibujado de la curva dentro del recuadro del oscilador. Es responsabilidad de las clases que heredan de `Oscillator` el implementarlo. Este método de dibujado se invoca desde el método `plotOscillator` para pintar el gráfico de dentro del recuadro.

5.2.1. Convergencia/divergencia de medias móviles

Esta técnica, que fue ideada por Gerald Appel [24, 25] y que se conoce como *Moving Average Convergence-Divegence* o MACD, da lugar a uno de los indicadores de momento más simples y efectivos. Convierte dos indicadores seguidores de tendencia, que son las medias móviles, en un oscilador de momento, sustrayendo la media móvil larga de la corta. Como resultado, la técnica ofrece lo mejor de los dos mundos: seguimiento de tendencia y momento. El MACD fluctúa por encima y por debajo de la linea del cero en tanto las medias móviles convergen, se cruzan y divergen. Como los valores del MACD no están acotados, no es particularmente útil para identificar niveles de sobrecompra o sobreventa.

Formulación del modelo

El MACD utiliza tres componentes: el MACD en sí mismo, que consiste en la diferencia entre una media móvil exponencial corta y una larga, calculado a partir de la ecuación 5.12;

la señal, que no hay que confundir con la señal que genera la máquina de trading, correspondiente a la media móvil exponencial del MACD, como aparece en la ecuación 5.13; y el histograma, que es la diferencia entre el MACD y la señal, según se recoge en la ecuación 5.14. En las ecuaciones se utiliza una notación diferente a la seguida hasta ahora: en este caso, los parámetros $MACD$, $Señal^{MACD}$, $Histograma$ y $precio$ son vectores; EMA es la función que computa la media móvil exponencial; y $periodos_{mcorta}$, $periodos_{mlarga}$ y $periodos_{señal}$ los números que determinan los diferentes periodos.

$$MACD = EMA(precio, periodos_{mcorta}) - EMA(precio, periodos_{mlarga}) \tag{5.12}$$

$$Señal^{MACD} = EMA(MACD, periodos_{señal}) \tag{5.13}$$

$$Histograma = MACD - Señal^{MACD} \tag{5.14}$$

Generación de la señal

En el MACD se trata la convergencia y divergencia de las dos medias móviles. La convergencia ocurre cuando las medias móviles se aproximan entre sí. La media móvil corta es más rápida y es responsable de la mayoría de los movimientos del MACD. La media móvil larga es más lenta y menos reactiva a los cambios del precio de la acción.

Tal y como resalta la figura 5.11, el MACD oscila por encima y por debajo de la linea del cero. Los valores positivos del MACD, que se incrementan en tanto las medias móviles se distancian, indican que la media móvil corta está por encima de la larga, es decir, el impulso alcista está aumentando. Valores negativos del MACD, que también aumentan en magnitud conforme las medias móviles se distancian, indican que la media móvil corta está por debajo de la larga, esto es, el impulso bajista se está imponiendo.

Los cruces de la señal con el MACD son la forma más habitual de producir señales con las que adoptar posiciones en el mercado. La señal de esta estrategia consiste en la media móvil exponencial del MACD. Esta señal, que se calcula como la media móvil del indicador, sigue el recorrido del MACD y hace más fácil reconocer sus cambios. Un cruce alcista ocurre cuando el MACD cruza hacia arriba la línea de la señal; un cruce bajista sucede cuando el MACD cruza hacia abajo la línea de la señal. Este proceso, que aparece resumido en la ecuación 5.15, es el seguido para producir la señal de posición de la estrategia. Se insiste en que no hay que confundir la señal del MACD, que aparece explicada en la ecuación 5.13, con la señal de posición de la máquina, que se muestra en la ecuación 5.15.

$$Señal_i = \begin{cases} +1 & \text{, si } MACD_i > Señal_i^{MACD} \\ -1 & \text{, si } MACD_i < Señal_i^{MACD} \\ 0 & \text{, en otro caso} \end{cases} \tag{5.15}$$

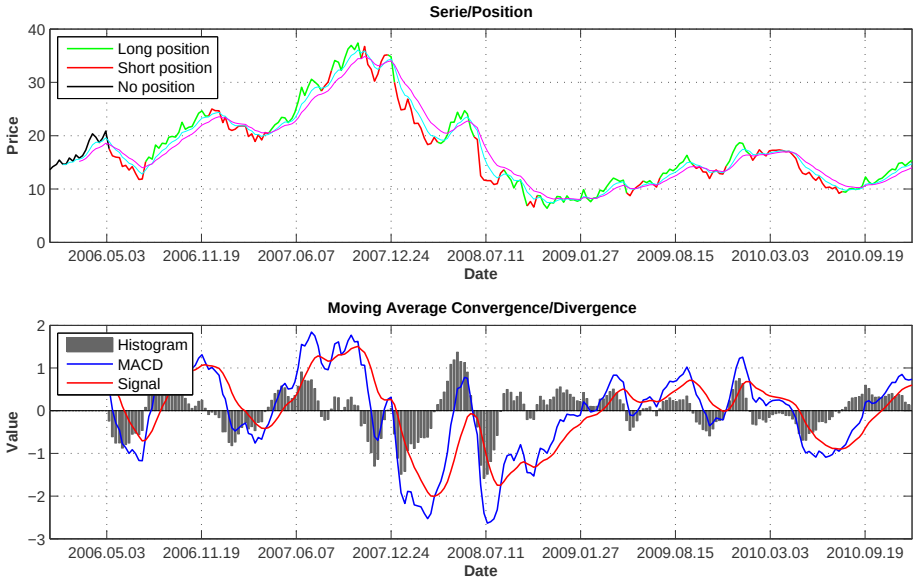

Figura 5.11: Convergencia/divergencia de medias móviles

La clase `MovingAverageConvergenceDivergence`

La clase `MovingAverageConvergenceDivergence`, que hereda de la clase `Oscillator`, implementa la estrategia descrita. Las propiedades y métodos de la clase aparecen a continuación:

`Mode` Propiedad para indicar el tipo de media móvil que se utiliza en los cálculos: aunque en las ecuaciones 5.12 y 5.13 aparece la media móvil exponencial, pues es como está establecido para esta técnica, se puede aplicar la simple. Esto se hace empleando el valor `'s'` en vez del valor `'e'`.

`Lead` Propiedad referida al número de periodos de la media móvil corta. Aparece en la ecuación 5.12 como el parámetro $periodos_{mcorta}$.

`Lag` Propiedad que contiene el número de periodos de la media móvil larga. En la ecuación 5.12 se corresponde al parámetro $periodos_{mlarga}$.

`Samples` Propiedad para almacenar el número de periodos con que se calcula la media móvil que da lugar a la señal del método. Se refiere al parámetro $periodos_{señal}$ de la ecuación 5.13.

`computeSignal` Método que computa la señal de posición de la máquina de trading. No hay que confundir la señal de posición de la máquina, calculada a partir de la ecuación 5.15, con la señal propia del método, que aparece en la ecuación 5.13. Una posible implementación del método aparece en el código 5.3. La función `movavg`, incluida en la plataforma, es la responsable de calcular los diferentes tipos de medias móviles.

```
function computeSignal(algoTrader)

% Parámetros
serie = algoTrader.DataSerie.Serie;
mode = algoTrader.Mode;
lead = algoTrader.Lead;
lag = algoTrader.Lag;
signalMode = algoTrader.SignalMode;
signalSamples = algoTrader.SignalSamples;

% Media móvil corta y larga
leadMovingAverage = movavg(serie, mode, lead);
lagMovingAverage = movavg(serie, mode, lag);

% MACD
macd = leadMovingAverage-lagMovingAverage;

% Señal MACD
signalmacd = movavg(macd, signalMode, signalSamples);

% Histograma
% histogram = macd-signalmacd;

% Señal
signal = zeros(1, length(serie));
signal(macd>signalmacd) = 1;
signal(macd<signalmacd) = -1;

% Actualiza el valor de la propiedad Signal
algoTrader.Signal = signal;

end
```

Código 5.3: Método `computeSignal` de la clase `MovingAverageConvergenceDivergence`

`drawSeriePosition` Método de la clase que decora el gráfico de la curva del precio. En concreto, superpone las curvas de la media móvil corta y la larga. El resultado de modificar esta representación aparece en el recuadro superior de la figura 5.11, donde las medias móviles están dibujadas en cian y magenta. Los métodos cuya representación se ve afectada por este método son `plot`, `plotSeriePosition` y `plotSeriePositionOscillator`.

`drawOscillator` Método de la clase responsable de dibujar el recuadro de los gráficos donde se visualiza el valor del oscilador. Tal recuadro aparece en la parte inferior de la figura 5.11 donde, con la misma escala temporal, se muestran las variaciones en el oscilador. Este método es la base de `plotOscillator`, que dibuja el oscilador en singular, y del método `plotSeriePositionOscillator`, que también dibuja el oscilador pero en compañía de la curva del precio de la acción, cuya salida aparece en la figura 5.11. También altera la salida del método `plot` que aparece en la figura 5.12.

5.2.2. Índice de fuerza relativa

El índice de fuerza relativa, también conocido como *Relative Strength Index* en inglés o símplemente RSI, es un oscilador de momento que mide la velocidad y el cambio de los movimientos del precio. Muestra la fuerza del precio mediante una comparación de los movimientos individuales al alza o a la baja [26, 27].

Formulación del modelo

Para simplificar la explicación del cálculo, el RSI se ha descompuesto en sus componentes básicos: RS, $GananciaMedia$ y $PérdidaMedia$. En el cálculo del RSI se utiliza un periodo, así se define el intervalo durante el cual se acumulan las ganancias y pérdidas que experimenta el precio de la acción. En particular, el parámetro $GananciaMedia$ de la ecuación 5.17 consiste en el valor medio de los incrementos del precio de la acción en los últimos p periodos. Por su parte, $PérdidaMedia$ se define como la media del decremento en el precio de la acción en los anteriores p periodos: este valor siempre se expresa como una cantidad positiva.

El valor de RS representa cuántas veces es mayor el movimiento ascendente en el precio de la acción que el movimiento descendente. La fórmula del RSI, que aparece en la ecuación 5.16, normaliza el valor de RS y lo convierte en un oscilador, cuyos valores fluctúan entre 0 y 100. En tanto a su interpretación, de acuerdo a su autor, John W. Wilder, se considera sobrecompra cuando su valor es superior a 70 y sobreventa cuando está por debajo de 30 [26].

$$RSI = 100 - \frac{100}{1 + RS} \tag{5.16}$$

$$RS = \frac{GananciaMedia}{PérdidaMedia} \tag{5.17}$$

Figura 5.12: Gráfico de la máquina de trading `MovingAverageConvergenceDivergence`

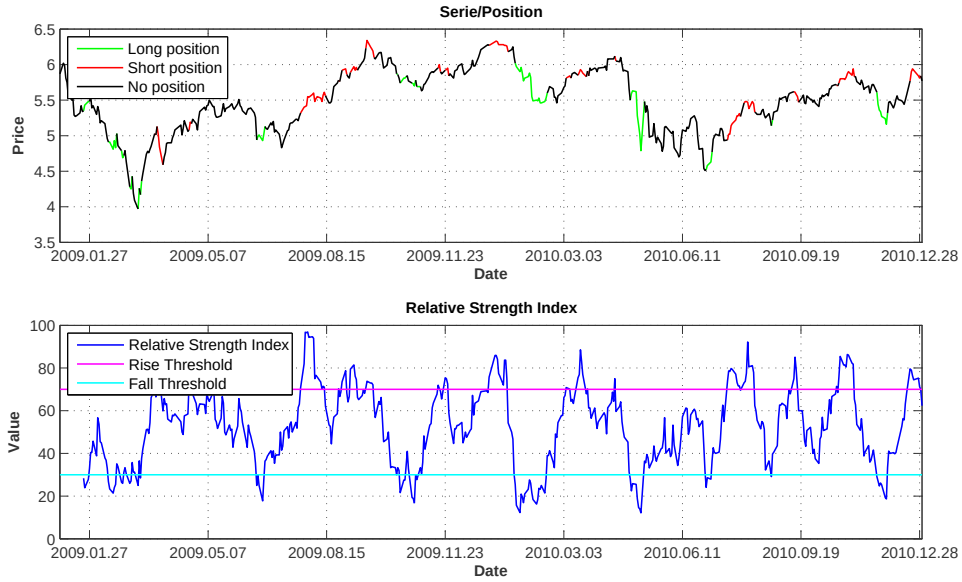

Figura 5.13: Índice de fuerza relativa

Como aparece en la ecuación 5.18, es posible desarrollar la fórmula para conseguir un resultado más fácil de interpretar. Con este nuevo enfoque, se aprecia que el RSI calcula qué proporción de las variaciones en el precio se corresponden con subidas. Si se da el caso de que $GananciaMedia/(GananciaMedia + PérdidaMedia) \approx 0$, entonces la situación que sucede es $GananciaMedia \ll PérdidaMedia$, es decir, la tendencia del precio es bajista; por el contrario si $GananciaMedia/(GananciaMedia + PérdidaMedia) \approx 1$, entonces el caso que ocurre es $GananciaMedia \gg PérdidaMedia$, es decir, la tendencia del precio es alcista. En la figura 5.13 se aprecia gráficamente la forma de este oscilador de acuerdo al valor del precio.

$$\begin{aligned} RSI &= 100 - \frac{100}{1+RS} \\ &= \frac{100\cdot(1+RS)-100}{1+RS} \\ &= 100 \cdot \frac{RS}{1+RS} \\ &= 100 \cdot \frac{GananciaMedia}{GananciaMedia+PérdidaMedia} \end{aligned} \tag{5.18}$$

Generación de la señal

Para el cálculo de la señal se razona de la siguiente manera: si las subidas del precio son mucho más abundantes que las bajadas, entonces la acción está probablemente sobrecomprada y existe una burbuja[1]; por el contrario, si las bajadas en el precio son muy superiores a las

[1] Una burbuja económica (también llamada burbuja especulativa, burbuja de mercado o burbuja financiera) es un fenómeno que se produce en los mercados, en buena parte debido a la especulación, que se caracteriza

subidas, entonces la acción está sobrevendida. Asumiendo cierta veracidad en la hipótesis anterior, la forma de proceder es sencilla: hay que comprar o posicionarse en largo cuando la acción está sobrevendida, pues el precio tiene que subir para corregirse; por el otro lado, se tiene que vender o posicionarse en corto cuando la acción está sobrecomprada, pues en este caso el precio tiene que bajar para corregirse. El comportamiento descrito queda recogido en la ecuación 5.19. En realidad, la hipótesis anterior es bastante deficitaria, ya que el precio de la acción posee en ocasiones cierta inercia que le hace mantener la tendencia. De nuevo se remite a la figura 5.13, donde el recuadro inferior muestra el valor del oscilador en con los umbrales de subida y descenso. Como se aprecia, la inercia del precio provoca que, en ocasiones, suceda lo contrario a lo esperado de acuerdo al razonamiento que propone esta técnica: la sobrecompra no da lugar a bajadas en el precio y la sobreventa no corrige el precio al alza.

$$Señal_i = \begin{cases} -1 & \text{, si } RSI_i > umbralSubida \\ +1 & \text{, si } RSI_i < umbralDescenso \\ 0 & \text{, en otro caso} \end{cases} \tag{5.19}$$

La clase `RelativeStrengthIndex`

La clase `RelativeStrengthIndex` implementa la estrategia antes descrita. Esta clase hereda de `Oscillator` que a su vez lo hace de `AlgoTrader`. El conjunto de propiedades y métodos implementados es el siguiente:

`Samples` Propiedad con el valor del número de periodos que considera la estrategia durante el cálculo. En particular, se refiere al número de periodos p que toma parte en la ecuación 5.17 para calcular los valores $GananciaMedia$ y $PérdidaMedia$.

`RiseThreshold` Propiedad que marca el umbral de subida a partir del cual se produce la situación de sobrecompra. En la ecuación 5.19 se refiere al parámetro $umbralSubida$.

`FallThreshold` Propiedad que indica el umbral de descenso que, si no es alcanzado por el valor del RSI, representa una situación de sobreventa. Se corresponde al parámetro $umbralDescenso$ de la ecuación 5.19.

`computeSignal` Método para computar la señal de posición generada por la estrategia. Lo hace de acuerdo a las indicaciones de la ecuación 5.19.

`drawOscillator` Método que dibuja el oscilador en compañía de los umbrales de subida y descenso. Se utiliza en los métodos `plotOscillator`, `plotSeriePositionOscillator` y `plot`. En particular, la figura 5.13 es fruto de la representación generada por el método `plotSeriePositionOscillator`.

por una subida anormal y prolongada del precio de un activo o producto, de forma que dicho precio se aleja cada vez más del valor real o intrínseco del producto.

5.2.3. Cruce del índice de fuerza relativa

El cruce del índice de fuerza relativa es una extensión de la estrategia anterior. La extensión consiste en incluir una media móvil y tomar los cruces con el RSI para generar las señales de posición. De esta manera, se detecta cuándo las sobrecompras o sobreventas de la acción se aceleran.

Formulación del modelo

La formulación del modelo emplea alguna de las medias móviles, tanto en su variante simple como exponencial. El otro elemento utilizado es el RSI, que ya fue explicado en la sección anterior.

Generación de la señal

Esta estrategia se basa en la hipótesis del RSI, que toma posiciones largas en situaciones de sobreventa para anticipar las correcciones al alza y adopta posiciones cortas en situaciones de sobrecompra en anticipo a la corrección a la baja. La inclusión de la media móvil sirve para detectar y medir las sobrecompras y sobreventas que experimenta una acción. Así, cuando más se acentúan estas situaciones es cuando la estrategia decide posicionarse. En consecuencia, la estrategia toma una posición corta cuando el RSI supera a su media móvil y cambia a posición larga cuando el RSI es se sitúa por debajo de la media móvil. El proceso se recoge en la ecuación 5.20.

El resultado de aplicar esta estrategia sobre una acción aparece en la figura 5.14. En el recuadro del oscilador aparece el RSI en azul, el umbral de subida en magenta, el umbral de descenso en cian y la media móvil del RSI en verde. Se comprueba a partir del color de la curva del precio que esta estrategia está siempre en el mercado debido a que utiliza el cruce como señal para cambiar de posición.

$$Señal_i = \begin{cases} -1 & \text{, si } RSI_i > XMA_i \\ +1 & \text{, si } RSI_i < XMA_i \\ 0 & \text{, en otro caso} \end{cases} \tag{5.20}$$

La clase `RelativeStrengthIndexCrossing`

La clase `RelativeStrengthIndexCrossing` implementa la estrategia descrita en esta sección. Esta clase hereda de `RelativeStrengthIndex`. Las propiedades y métodos de la clase se listan a continuación:

`CMode` Propiedad que indica el tipo de media móvil utilizada: con el valor `'s'` se refiere a la simple y con el valor `'e'` a la exponencial.

`CSamples` Propiedad para fijar el número de periodos en el cálculo de la media móvil.

`computeSignal` Método que computa la señal de posición en la máquina de trading. Realiza los cálculos de acuerdo a las indicaciones que aparecen en la ecuación 5.20.

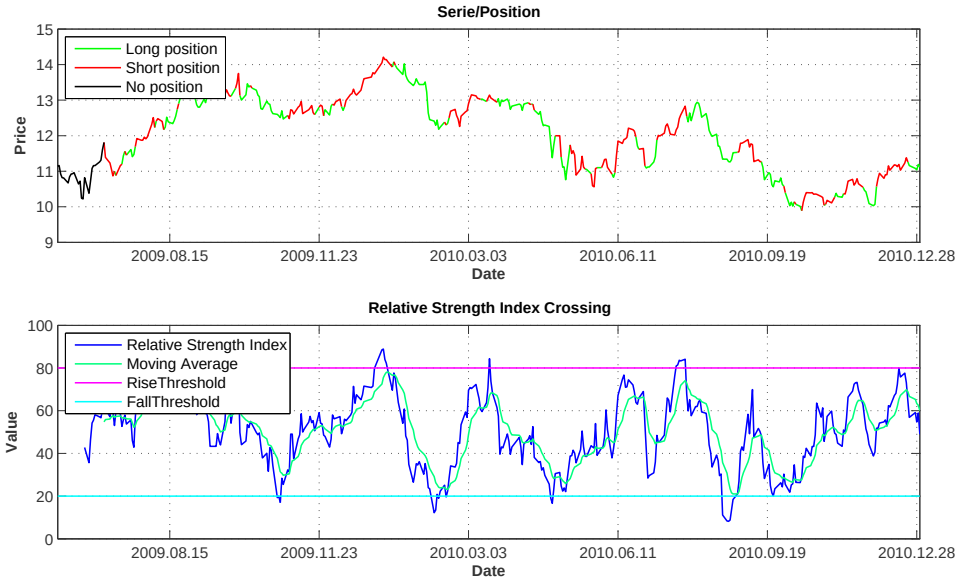

Figura 5.14: Cruce del índice de fuerza relativa

`drawOscillator` Método que dibuja los gráficos dentro del recuadro del oscilador: pinta el RSI, los umbrales de subida y descenso y la media móvil. Altera la representación provista por los métodos `plotOscillator`, `plotSeriePositionOscillator` y `plot`. A partir de tal modificación, el método `plotSeriePositionOscillator` ofrece gráficos como el de la figura 5.14.

5.2.4. Estocástico

El estocástico es una variable estadística que se basa en la posición de la cotización con respecto a los máximos y mínimos en un intervalo [20, 21]. Según George C. Lane [28], el oscilador no sigue el precio, no sigue el volumen ni nada de eso; este oscilador sigue la velocidad del impulso del precio; como regla general, el impulso cambia de dirección antes que el precio.

Formulación del modelo

El estocástico mide el nivel del precio de cierre respecto al rango determinado por el precio más alto y el mas bajo. Como aparece en la ecuación 5.21, se multiplica por 100 para restringir su valor entre 0 y 100. Los parámetros $precioBajo_{i-s+1:i}$ y $precioAlto_{i-s+1:i}$ se refieren, respectivamente, al precio más bajo y al más alto que sucede en el intervalo de periodos $i-s+1:i$. La ecuación 5.22 calcula el valor de $\%K$, que consiste en la media móvil simple del estocástico: sirve para suavizar la serie de valores del estocástico. Por su parte, en la ecuación 5.23 se calcula $\%D$, que consiste en la media móvil de $\%K$.

En el recuadro inferior de la figura 5.15 se recogen los elementos que toman parte en la estrategia: en color rojo el valor de $\%K$, en azul el valor de $\%D$ y en magenta y cian los umbrales de subida y descenso.

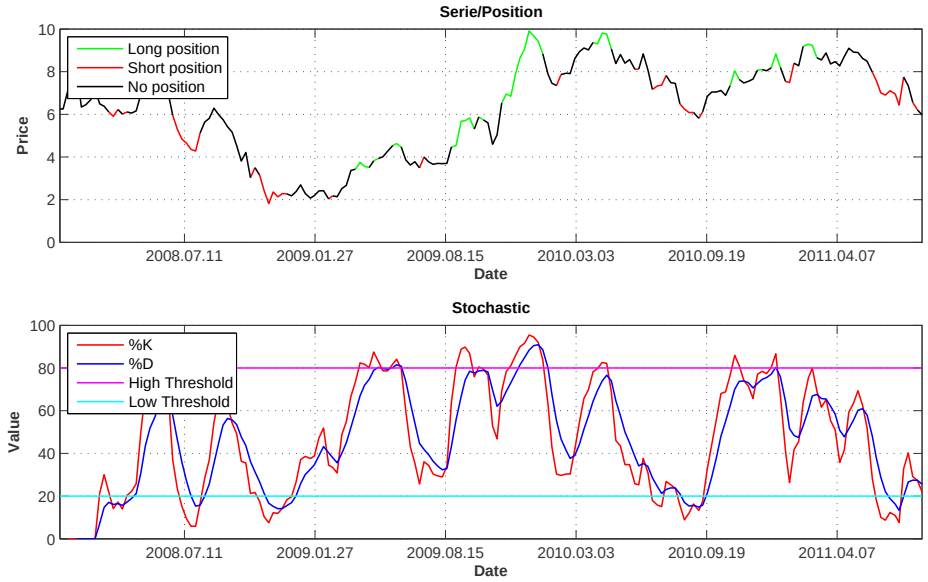

Figura 5.15: Estocástico

A partir de los elementos definidos, se pueden utilizar diferentes criterios para generar señales de posición para la máquina de trading: el cruce del estocástico por encima o debajo de valores de umbral, el cruce del estocástico con $\%K$, el cruce entre $\%K$ y $\%D$...

$$Estocástico = 100 \cdot \frac{precioCierre_i - precioBajo_{i-s+1:i}}{precioAlto_{i-s+1:i} - precioBajo_{i-s+1:i}} \tag{5.21}$$

$$\%K = SMA(Estocástico, periodos_k) \tag{5.22}$$

$$\%D = SMA(\%K, periodos_d) \tag{5.23}$$

Generación de la señal

En lo que a esta estrategia se refiere, el criterio para generar señales de posición consiste en el cruce del valor de $\%K$ con un par de umbrales. La descripción exacta del método está recogida en la ecuación 5.24: si $\%K$ supera la cota inferior impuesta por $umbralSubida$, entonces el precio de la acción está próximo al máximo y se adopta una posición larga en respuesta a la tendencia alcista; si sucede que $\%K$ está por debajo del valor de la cota superior $umbralDescenso$, entonces el precio está cercano al más bajo de los últimos periodos, por lo que se abre una posición corta como consecuencia de la tendencia bajista.

$$Señal_i = \begin{cases} +1 & \text{, si } \%K_i > umbralSubida \\ -1 & \text{, si } \%K_i < umbralDescenso \\ 0 & \text{, en otro caso} \end{cases} \tag{5.24}$$

La clase `Stochastic`

La clase `Stochastic` implementa la estrategia del estocástico con el cruce de los umbrales. Esta clase hereda de `Oscillator`. Las propiedades y métodos que define son los siguientes:

`Samples` Propiedad que indica el número de periodos que se toma en el cálculo de los precios más alto y más bajo. En la ecuación 5.21 este valor aparece como el parámetro s.

`RiseThreshold` Propiedad que determina el umbral de subida. Si el estocástico supera este valor, entonces se considera que el mercado presenta un impulso alcista. Esta propiedad atribuye valor al parámetro

`FallThreshold` Propiedad que fija el umbral de descenso. Por debajo de este valor el estocástico sugiere un impulso bajista en el mercado. Esta propiedad se refiere al parámetro *umbralDescenso* de la ecuación 5.24.

`Mode` Propiedad para seleccionar la media móvil que se utiliza en los cálculos: aunque en las ecuaciones 5.22 y 5.23 aparece la media móvil simple, que es la manera en que está definida esta estrategia, también se puede aplicar la exponencial. Los valores que toma esta propiedad son `'s'` para la media móvil simple y `'e'` para la exponencial.

`KSamples` Propiedad para establecer el número de periodos que se utilizan en la media móvil para calcular $\%K$. En la ecuación 5.22 este parámetro aparece como $periodos_k$.

`DSamples` Propiedad que indica el número de periodos empleados en la media móvil para el cálculo de $\%D$. Aparece como el parámetro $periodos_d$ en la ecuación 5.23.

`computeSignal` Método que calcula la señal de posición para la máquina según la ecuación 5.24.

`drawOscillator` Método que dibuja los diferentes elementos que utiliza la estrategia en el recuadro del oscilador. Modifica el comportamiento de los métodos `plotOscillator`, `plotSeriePositionOscillator` y `plot`. Como muestras, la figura 5.15 es fruto del método `plotSeriePositionOscillator` y la figura 5.16 del método `plot`.

5.2.5. Cruce del estocástico

Existen múltiples criterios para producir señales en el caso del estocástico. Aquí se contempla el caso en que la señal se genera a consecuencia de un cruce. Por todo lo demás, esta estrategia es similar a la anterior.

Formulación del modelo

La formulación del modelo de esta máquina es idéntica al de la máquina de la sección anterior, pues ambas se basan en el estocástico.

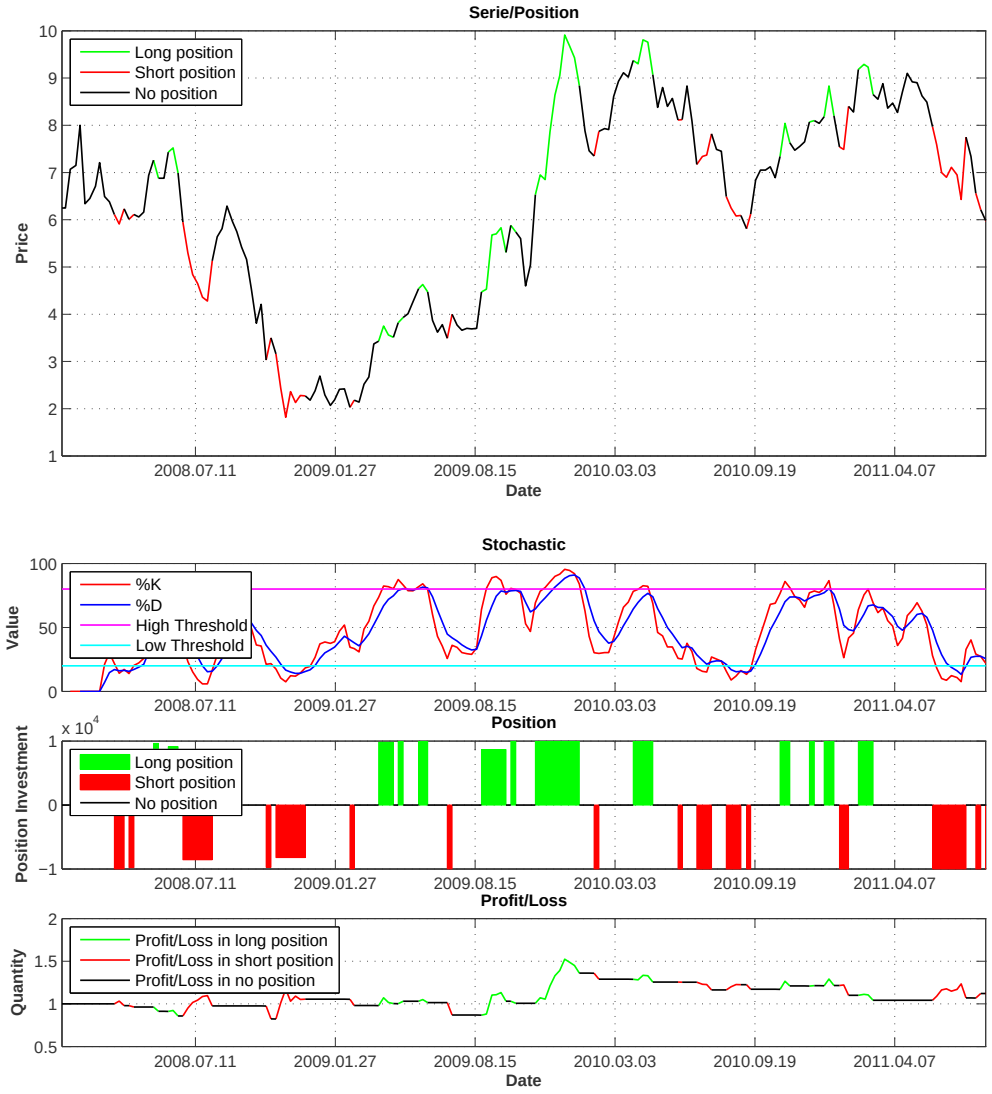

Figura 5.16: Gráfico de la máquina de trading `Stochastic`

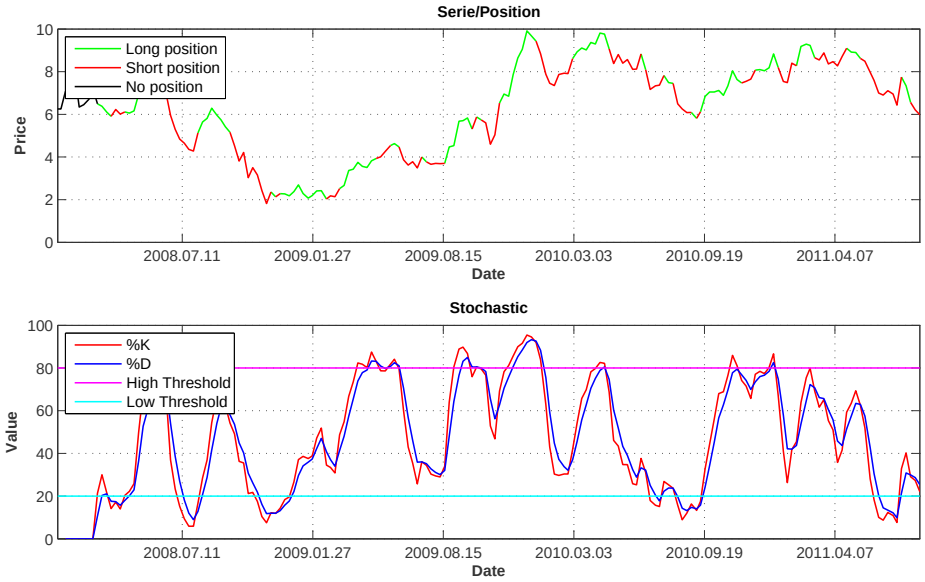

Figura 5.17: Cruce del estocástico

Generación de la señal

La opción que se explora en esta máquina consiste en utilizar el cruce entre dos valores para generar la señal de posición. En concreto, se contrastan los valores $\%K$ y $\%D$ para evaluar la aceleración y deceleración del impulso en el precio. El principio en que se basa esta estrategia es el siguiente: si el valor de $\%K$ es superior a $\%D$, entonces la cotización experimenta un impulso alcista con una aceleración proporcional a la distancia entre los dos valores; por su parte, si el valor de $\%K$ es inferior a $\%D$, entonces la cotización sufre un impulso bajista de aceleración proporcional a la diferencia entre estos dos valores. Dicho de otra forma: el cruce entre $\%K$ y $\%D$ detecta el refuerzo en las tendencias del precio. De esta manera, se aprovechan aquellos momentos en que el impulso se hace más fuerte para producir señales de posición en la máquina de trading. Como resumen, la generación de la señal se realiza de acuerdo a la ecuación 5.25.

El comportamiento de esta estrategia, frente a la curva de precios de una acción, se muestra en la figura 5.17. Los elementos que aparecen en el recuadro del oscilador son $\%K$ en color rojo, $\%D$ en azul y los umbrales en colores magenta y cian.

$$Señal_i = \begin{cases} +1 & ,\text{si } \%K_i > \%D_i \\ -1 & ,\text{si } \%K_i < \%D_i \\ 0 & ,\text{en otro caso} \end{cases} \tag{5.25}$$

La clase `StochasticCrossing`

La estrategia se implementa en la clase `StochasticCrossing`, que hereda de `Stochastic`. Como sólo cambia el criterio para producir la señal, entonces sólo implementa un método:

`computeSignal` Método que genera la señal de posición de la máquina de trading. Lleva a cabo los cálculos de acuerdo a la ecuación 5.25.

5.2.6. Oscilador de momento

El oscilador de momento es un indicador técnico que, dada una acción del mercado, mide cuánto varía su precio en un lapso de tiempo. Este oscilador cuantifica la velocidad con la que cambia el precio y ofrece un indicador de la tendencia.

Formulación del modelo

El cálculo para el caso de esta estrategia es bastante sencillo: consiste en hallar la diferencia entre el precio más reciente y aquel que tenía la acción hace cierta cantidad de periodos. La ecuación 5.26 muestra cómo se realiza el cálculo, siendo s el número de periodos que se saltan hacia el pasado.

Con indicadores como este, que consiste en una resta, se obsevan tres tipos de eventos significativos: valores extremos, que suceden cuando la distancia entre los precios es máxima y señalan condiciones de sobrecompra o sobreventa; divergencia, que es una advertencia de un posible cambio en la tendencia del precio; y cruce de la línea central, que proporciona una indicación importante en relación con la dirección de la tendencia del precio. En el caso de esta estrategia, se opta por el tercer criterio para generar señales de posición en la máquina de trading.

$$Momento_i = precio_i - precio_{i-s} \tag{5.26}$$

Generación de la señal

Para calcular la señal de posición en cada periodo, se procede como indica la ecuación 5.27: el valor en el lado derecho de ambas inecuaciones hace referencia a la línea central. La estrategia opta por posicionarse en largo cuando el precio más reciente es mayor al más antiguo, es decir, cuando el mercado presenta una tendencia alcista. Por el contrario, se posiciona en corto si el precio actual es inferior al precio pasado, pues este es un indicador de tendencia bajista.

En la figura 5.18 se presenta el resultado de esta estrategia sobre la cotización de una acción. En el recuadro del oscilador aparecen el oscilador de momento en azul y la línea del cero en negro.

$$Señal_i = \begin{cases} +1 & \text{, si } Momento_i > 0 \\ -1 & \text{, si } Momento_i < 0 \\ 0 & \text{, en otro caso} \end{cases} \tag{5.27}$$

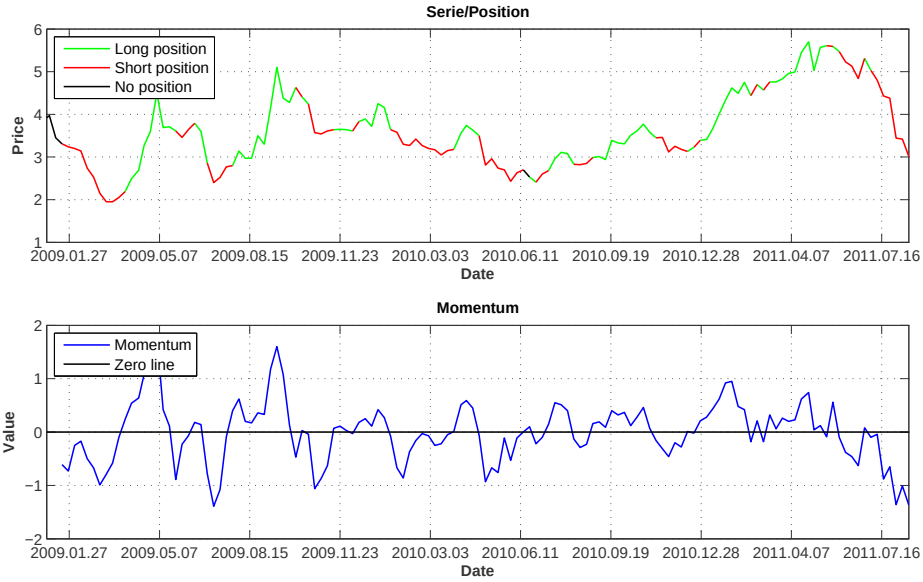

Figura 5.18: Oscilador de momento

La clase `Momentum`

Es en la clase `Momentum`, que hereda de `Oscillator`, donde se implementa la estrategia del oscilador de momento. Las propiedades y métodos de la clase son:

`Delay` Propiedad para fijar la cuantía, en número de periodos, del salto que hace la máquina en el cálculo del oscilador de momento. El valor de esta propiedad es el que se aplica en la ecuación 5.26 por medio del parámetro s.

`computeSignal` Método que calcula la señal de posición según indica la ecuación 5.27.

`drawOscillator` Método que dibuja los gráficos del recuadro del oscilador. En el caso de esta clase, plasma la curva del oscilador de momento y resalta la línea del cero. Este método modifica las representaciones provistas por los métodos `plotOscillator`, `plotSeriePositionOscillator` y `plot`. En particular, un gráfico representativo del método `plotSeriePositionOscillator` aparece en la figura 5.18. Por su parte, el método `plot` crea gráficos como el que se muestra en la figura 5.19.

5.2.7. Cruce del oscilador de momento

Una alternativa de uso para el oscilador de momento consiste en emplear el cruce entre el indicador y su media móvil. Este enfoque saca provecho de algunas características del oscilador que la anterior estrategia no es capaz de explotar.

Figura 5.19: Gráfico de la máquina de trading `Momentum`

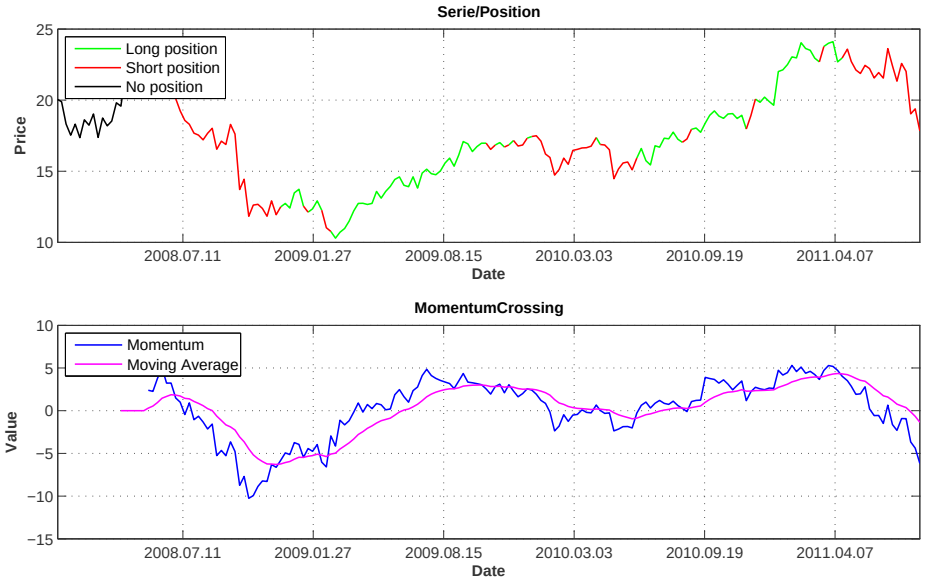

Figura 5.20: Cruce del oscilador de momento

Formulación del modelo

Se omite desarrollar la formulación del modelo para esta máquina, pues es idéntica a la empleada para la máquina de la sección anterior. Tanto esta estrategia como la previa se basan en el oscilador de momento.

Generación de la señal

Además del caso contemplado en la estrategia del oscilador de momento, se puede utilizar el cruce del oscilador con su media móvil como un segundo criterio para producir señales de posición. El razonamiento en este caso es el siguiente: si el oscilador de momento mide la tendencia a partir de la variación en el precio, entonces la media móvil sobre este valor sirve para discriminar entre tendencias cortas y largas (o principales y secundarias). Dada esta diferenciación entre tendencias cortas y largas, su cruce sirve para generar señales. Si el valor del oscilador de momento es superior al de su media móvil, entonces se produce una variación al alza en la tendencia; por su parte, si el valor del oscilador de momento es inferior al de su media móvil, la variación que ocurre es a la baja. La detección en el cambio de la tendencia es lo que utiliza esta estrategia para generar la señal de posición. Como resumen, el proceso de cálculo de la señal aparece en la ecuación 5.29, donde la media móvil se calcula según indica la ecuación 5.28.

El resultado de aplicar esta técnica aparece en la figura 5.20. En el recuadro inferior se muestra el oscilador de momento en color azul y su media móvil en magenta. Se puede ver el efecto de esta estrategia tomando nota de los colores en los trazos de la curva de precios del recuadro superior.

$$MediaMóvil^{Momento} = XMA(Momento, periodos) \tag{5.28}$$

$$Señal_i = \begin{cases} +1 & \text{, si } Momento_i > MediaMóvil_i^{Momento} \\ -1 & \text{, si } Momento_i < MediaMóvil_i^{Momento} \\ 0 & \text{, en otro caso} \end{cases} \tag{5.29}$$

La clase `MomentumCrossing`

La clase `MomentumCrossing`, que hereda de `Momentum`, implementa la estrategia de trading del cruce del oscilador de momento. Las propiedades y métodos de la clase son:

`Mode` Propiedad para seleccionar el tipo de media móvil: con `'s'` se refiere a la simple y con `'e'` a la exponencial. Determina cómo se calcula $MediaMovil^{Momento}$ en la ecuación 5.28.

`Samples` Propiedad que contiene el número de periodos con que se calcula la media movil. Con ella se determina el valor del parámetro *periodos* en la ecuación 5.28.

`computeSignal` Método que computa la señal de la máquina de trading. El cálculo que realiza este método concuerda con el de la ecuación 5.29.

`drawOscillator` Método que dibuja el oscilador de momento junto a la media móvil. Es utilizado por los métodos `plotOscillator`, `plotSeriePositionOscillator` y `plot` para pintar los gráficos dentro del recuadro del oscilador. Como ejemplo, hace que el método `plotSeriePositionOscillator` genere gráficos similares al de la figura 5.20.

5.3. Basadas en otros conceptos

Además de los indicadores y osciladores técnicos, cualquier otro modelo puede ser adecuado para implementar máquinas de trading. En esta sección del capítulo se exploran otras posibilidades. Algunas de las máquinas resultantes tienen usos especiales que se comentan para cada caso.

5.3.1. Falso óptimo

Entre las máquinas concebidas en el desarrollo del trabajo, la que mejores resultados ofrece es la del falso óptimo. Sin embargo, sucede que esta estrategia se basa en la trampa de mirar hacia adelante el precio de la acción para decidir qué posición adoptar. Así, si en el futuro el precio de una acción sube, hay que posicionarse en largo; mientras que si baja, hay que hacerlo en corto. Por motivos evidentes, esta estrategia no se puede utilizar en la realidad, pero su implementación resulta interesante desde el punto de vista de la investigación.

A partir del resultado de esta máquina se obtiene una cota inferior del beneficio máximo que se consigue invirtiendo en acciones de una empresa determinada. El nombre de falso óptimo es dado en alusión a que no calcula de manera óptima el beneficio máximo, sino una mera aproximación que sirve como cota inferior. Esta medición cuantifica el potencial que tienen las diferentes acciones del mercado para generar beneficios. De esta manera, se puede establecer una ordenación entre acciones para elegir las más provechosas sobre las que invertir con las máquinas de trading.

Formulación del modelo y generación de la señal

Utilizar información del precio futuro de la acción es la manera más sencilla y adecuada de dar lugar a una máquina de trading rentable. El problema es que en situaciones reales no se dispone de tan preciada información. No obstante, es posible relajar esta restricción en la plataforma.

La serie de precios de la acción en un intervalo temporal se almacena en un objeto de la clase `DataSerie`. Durante el proceso de generación de la señal de posición, se viola la restricción de no mirar precios futuros de la acción si para generar el valor de $Seal_i$ se toma el valor de $precio_j$, donde $i < j$. Es justo esta la situación que se produce en la máquina de trading del falso óptimo, que genera la señal según se indica en la ecuación 5.30.

Una propiedad interesante de esta fórmula es que dispone de un parámetro s para ajustar el tamaño del salto en número de periodos hacia el futuro. Si el precio por operación en el mercado es despreciable, entonces el mejor resultado se alcanza con $s = 1$. Esto se debe a la desigualdad triangular, que establece que $|precio_i - precio_k| < |precio_i - precio_j| + |precio_j - precio_k|$ con $i < j < k$. En consecuencia, un valor más pequeño de s hace que esta máquina aproveche más cantidad de variaciones en el precio de la acción y genere beneficios más cuantiosos.

El resultado que arroja esta máquina de trading se aprecia en la figura 5.21. El recuadro superior representa el precio de la acción a lo largo del tiempo. El recuadro inferior contiene la curva de beneficios y pérdidas, característica en esta máquina de trading por ser monótona creciente para indicar que siempre se producen incrementos en el beneficio.

$$Señal_i = \begin{cases} +1 & \text{, si } precio_i < precio_{i+s} \\ -1 & \text{, si } precio_i > precio_{i+s} \\ 0 & \text{, en otro caso} \end{cases} \tag{5.30}$$

La clase `FakeOptimum`

La clase `FakeOptimum` hereda de la clase `AlgoTrader`. La lista de propiedades y métodos de esta clase es la siguiente:

`Foresee` Propiedad que indica el número de periodos que se salta hacia el futuro para la comparación de los precios de la acción. En la ecuación 5.30 se corresponde con s.

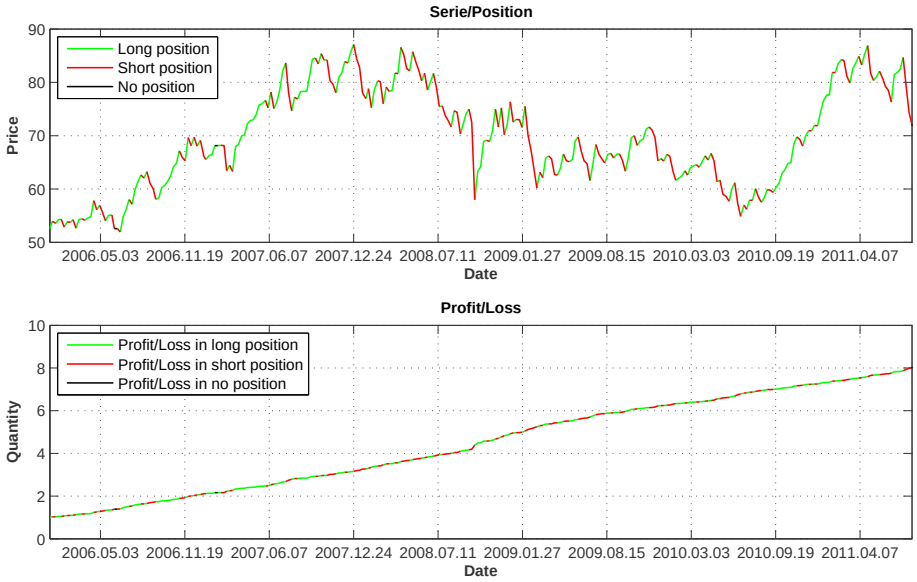

Figura 5.21: Falso óptimo

`plotSeriePositionProfitLoss` Método que dibuja la curva del precio de la acción en el tiempo junto a la curva de beneficios y pérdidas de la máquina. Genera un gráfico como el que se muestra en la figura 5.21.

`computeSignal` Método que genera la señal de posición según la ecuación 5.30.

5.3.2. Estrategia aleatoria

La estrategia aleatoria consiste en adoptar posiciones sin utilizar más criterio que el generador de números pseudoaleatorios. Esta estrategia puede parecer innecesaria, aunque su importancia es bastante notoria de cara a realizar un análisis exhaustivo de los resultados: simula el comportamiento de los inversores de bolsa que carecen de conocimientos, arroja resultados significativos sobre el retorno que se espera al invertir de forma aleatoria en el mercado de acciones, sirve como estrategia base para comparar con las demás...

Formulación del modelo y generación de la señal

El modelo que sigue la estrategia aleatoria es el de asignar a cada elemento del vector *Señal* un valor aleatorio y equiprobable del conjunto $\{+1, -1, 0\}$, tal y como se desprende de la ecuación 5.31. El inconveniente de este método es que mantiene la posición que adopta durante escasos periodos: la probabilidad que tiene esta estrategia de mantenerse en una misma posición durante p periodos es de $1/3^{p-1}$. Para soslayar esta limitación, se introducen las variables *longitudMínima* y *longitudMáxima*, que determinan el número mínimo y máximo de periodos durante los cuales la estrategia aleatoria mantiene una misma posición.

Una instancia de la máquina de trading aleatoria muestra sus resultados en la figura 5.22. La máquina de esta figura tiene en cuenta las variables de longitud en el cálculo de la señal.

El efecto de estas variables es evitar situaciones donde la máquina de trading aleatoria opera como aparece en la figura 5.23, donde la duración de las posiciones es muy corta.

$$Señal_i = random(\{+1, -1, 0\}) \tag{5.31}$$

La clase `Random`

`Random` es la clase que implementa la estrategia aleatoria. Esta clase hereda de `AlgoTrader`. Las propiedades y los métodos que implementa son los siguientes:

`MinimumPeriodLength` Propiedad que fija la longitud mínima en número de periodos que tiene una posición adoptada por la máquina aleatoria. Esta propiedad se refiere a la variable *longitudMínima*.

`MaximumPeriodLength` Propiedad para determinar la longitud máxima en número de periodos que tiene una posición tomada por la máquina de trading. Esta propiedad alude a la variable *longitudMáxima*.

`Dummy` Propiedad para volver a computar la señal de posición de manera aleatoria. Esta propiedad no conserva el valor que se le asigna sino que lo ignora y mantiene su estado de no asignación. Como efecto lateral, esta propiedad sustituye la señal de posición por una nueva también calculada de forma aleatoria. La utilidad de la propiedad se pone de relieve cuando se utiliza en combinación con algunos métodos, como pueden ser `profitLossStatistics`, `plotSearchSpace` y `plotSearchSpace123`. Por ejemplo, con el método `profitLossStatistics` se pueden obtener un conjunto de valores estadísticos sobre el rendimiento de la estrategia en término de los beneficios y las pérdidas. Este ejemplo de utilización de la propiedad `Dummy` se muestra en el código 5.4.

`computeSignal` Método que computa, de forma aleatoria, la señal de posición de la máquina de trading. Cada posición que adopta la máquina se mantiene un mínimo de *longitudMínima* de periodos y un máximo de *longitudMáxima* de periodos. Estos parámetros se ajustan en la máquina alterando los valores de las propiedades `MinimumPeriodLength` y `MaximumPeriodLength`.

5.3.3. Últimas n variaciones del precio

Esta estrategia se basa en tomar posiciones en el mercado de acuerdo a las variaciones del precio en el tiempo. Como en el caso de la media móvil, esta estrategia sigue la tendencia.

Formulación del modelo y generación de la señal

La manera en que se averigua la tendencia que sigue el precio de una acción es comparando los valores que toma a lo largo del tiempo. A partir de este hecho, se comprueba que en ocasiones la tendencia en el precio posee cierta inercia. Tal característica es la que hace que

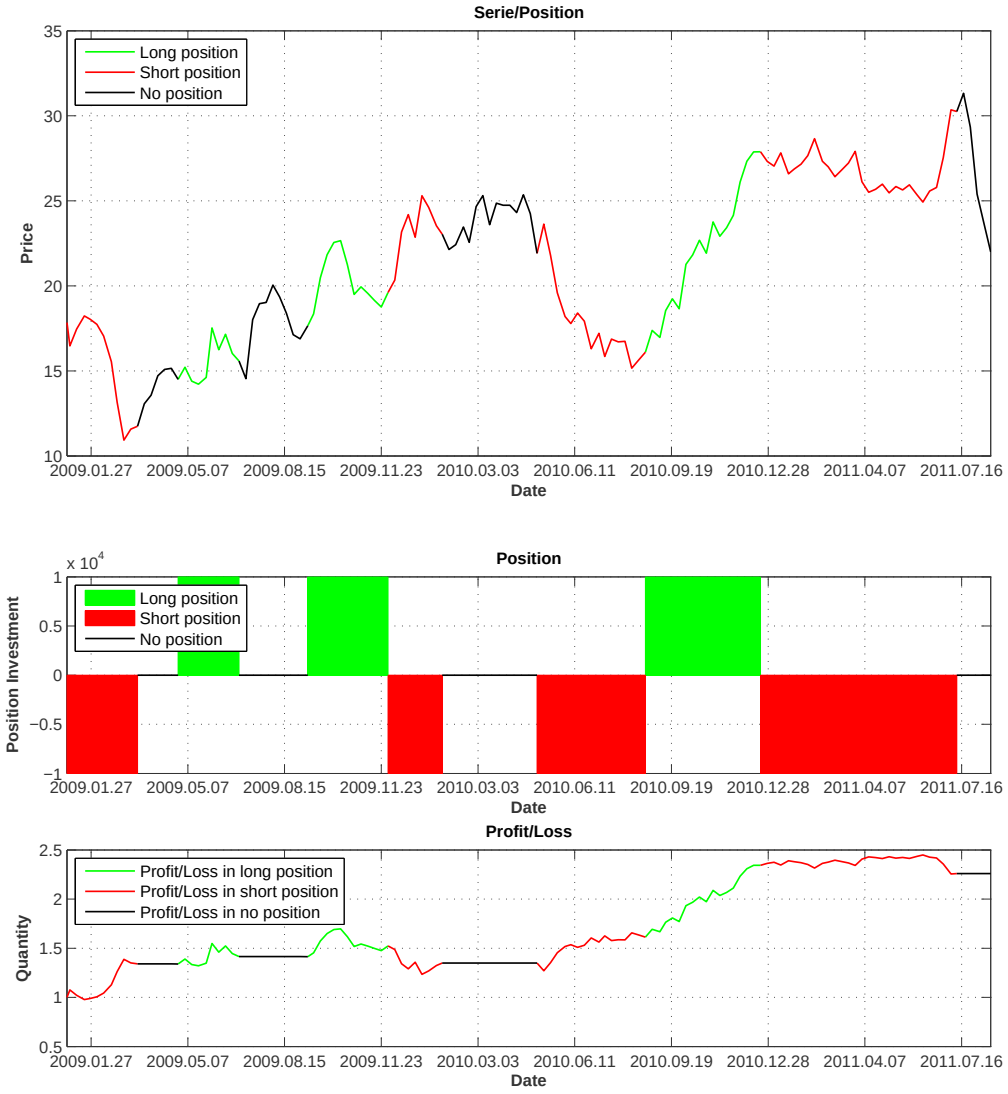

Figura 5.22: Gráfico de la máquina de trading `Random`

Figura 5.23: Gráfico de la máquina de trading `Random` sin las restricciones de longitud mínima y máxima en la duración de la posición

```
% Serie de datos
ds = YahooDataSerie('xom','weeks',1,'2006-01-01');

% Máquina de trading basada en la estrategia aleatoria
r = Random(ds,1,10);

% Cálculo del mínimo, máximo, media y desviación típica
% del beneficio y pérdida de la máquina de trading. Se
% repite el experimento 1000 veces como consecuencia
% del uso que se hace de la propiedad Dummy.
[min max mean std]=r.profitLossStatistics('Dummy',1:1000)

% Resultado obtenido en la computación
% min =
%     0.3228
% max =
%     2.0706
% mean =
%     0.9960
% std =
%     0.2554
```

Código 5.4: Cálculo de estadísticas con el método `profitLossStatistics` de la máquina `Random`

se produzcan largas progresiones ascendentes o descendentes en el precio de la acción. No obstante, la inercia en la tendencia del precio no tiene por qué estar presente, o puede que de producirse sea demasiado irregular como para aprovecharla. En cualquier caso, es interesante comprobar en qué medida la inercia interviene en la predicción del movimiento en el precio. Por ejemplo, si el nivel de inercia en la tendencia es alto, entonces las variaciones en el precio son poco oscilantes y aparecen largos periodos durante los cuales la tendencia mantiene su racha; si por el contrario la inercia es inexistente, entonces el precio de la acción está caracterizado por la alta volatilidad y las constantes oscilaciones. La manera en que esta estrategia se plasma en fórmula para calcular la señal viene recogida en la ecuación 5.32.Con valor de s se elige el tamaño del salto en número de periodos para realizar la comparación de los precios.

La figura 5.24 muestra el resultado de utilizar esta estrategia con una acción del mercado. La situación en que más provecho obtiene es cuando la inercia en la tendencia es alta.

$$Señal_i = \begin{cases} +1 & \text{, si } \sum_{j=1}^{s}(precio_{i-j+1} - precio_{i-j}) > 0 \\ -1 & \text{, si } \sum_{j=1}^{s}(precio_{i-j+1} - precio_{i-j}) < 0 \\ 0 & \text{, en otro caso} \end{cases} \tag{5.32}$$

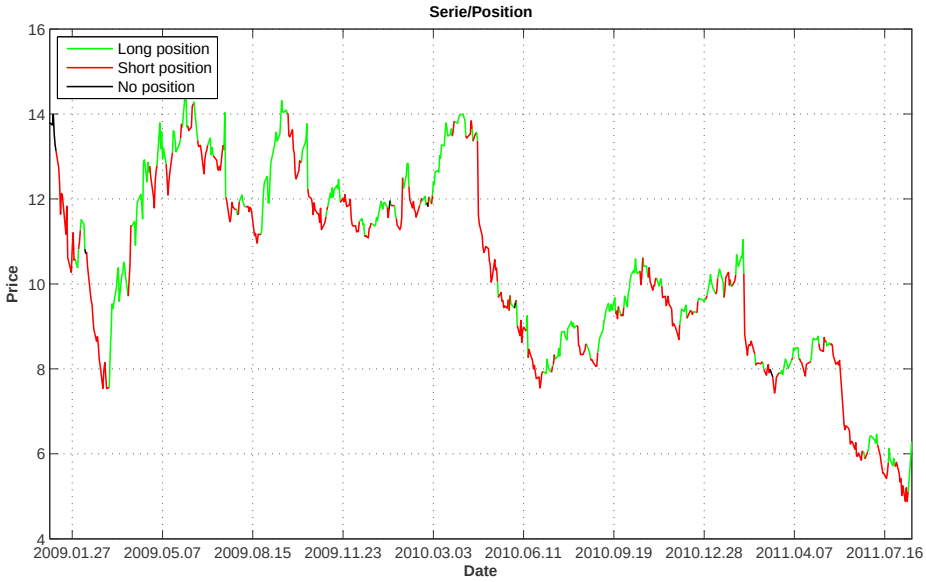

Figura 5.24: Últimas n variaciones del precio

La clase `LastN`

La clase `LastN` hereda de `AlgoTrader`. Las propiedades y métodos que implementa son:

`Samples` Propiedad que indica el número de periodos que se toma hacia el pasado para calcular la señal que genera la máquina. En la ecuación 5.32 se corresponde con s.

`computeSignal` Método para calcular la señal de la máquina conforme a la ecuación 5.32.

5.3.4. Últimas n variaciones del precio ponderadas

Esta estrategia es una extensión de la anterior. En este caso se considera un vector de pesos que da mayor importancia a los precios más recientes frente a los más antiguos.

Formulación del modelo y generación de la señal

El precio de la acción puede variar en gran medida con el paso del tiempo, es importante considerar los valores pasados, aunque es vital dar mayor importancia a los valores presentes. Esta estrategia utiliza el vector de pesos para este propósito, donde la elección de cada elemento del vector depende del usuario. Sin embargo, es recomendable atribuir un valor mayor a los últimos elementos del vector, pues son los que multiplican al precio más reciente. En resumen, el cálculo de la señal se hace conforme indica la ecuación 5.33. En ella, el vector de pesos aparece en disposición de columna y contiene un total de s elementos. Para reducir los precios y el vector de pesos a un número se utiliza el producto escalar.

$$Señal_i = \begin{cases} +1 & \text{, si } [(precio_{i-s+1} - precio_{i-s}) \ldots (precio_i - precio_{i-1})] \times \begin{bmatrix} peso_1 \\ \vdots \\ peso_s \end{bmatrix} > 0 \\ -1 & \text{, si } [(precio_{i-s+1} - precio_{i-s}) \ldots (precio_i - precio_{i-1})] \times \begin{bmatrix} peso_1 \\ \vdots \\ peso_s \end{bmatrix} < 0 \\ 0 & \text{, en otro caso} \end{cases} \tag{5.33}$$

La clase `LastNWeighted`

`LastNWeighted` es la clase que implementa esta estrategia. Hereda de `LastN`, por lo que tiene como antecesor a la clase a la clase `AlgoTrader`. Las propiedades y métodos que implementa esta clase aparecen a continuación:

`Weight` Propiedad que consiste en un vector de pesos por el que se multiplican los precios de la acción en el cálculo de la señal. El vector de pesos aparece en forma de vector columna en la ecuación 5.33. El parámetro s de la ecuación es la longitud este vector.

`computeSignal` Método que calcula la señal de la máquina a partir de la ecuación 5.33. Una posible implementación del método es la que aparece en el código 5.5.

5.3.5. Proporción relativa de las últimas n variaciones del precio

Esta estrategia es similar a la estrategia de las últimas n variaciones del precio. La diferencia radica en que las últimas n variaciones suman los beneficios y restan las pérdidas, mientras tanto, en esta estrategia se utiliza la proporción de las variaciones positivas frente a las negativas y viceversa. El resultado es una estrategia más potente que se ve perjudicada en menor manera durante una tendencia lateral.

Formulación del modelo y generación de la señal

La estrategia de las últimas n variaciones tiene un problema grave: este consiste en que durante una tendencia lateral las oscilaciones del precio hacen que se pase rápidamente de una posición a su opuesta, provocando inestabilidad y gastos derivados de las operaciones de mercado. Una manera de reducir el impacto de este problema es a través de la introducción de un umbral. Este umbral viene dado en forma de dos parámetros: uno fija el mínimo de la proporción de subidas del precio frente a bajadas para abrir una posición larga y el otro la proporción mínima de bajadas del precio frente a subidas para comenzar una posición corta. Tales parámetros aparecen en la ecuación 5.34, empleando $ratio_{rf}$ para denotar la proporción de subidas frente a bajadas y $ratio_{fr}$ para la proporción de bajadas frente a subidas. El

```
function computeSignal(algoTrader)

% Parámetros
serie = algoTrader.DataSerie.Serie;
weight = algoTrader.Weight;
samples = length(weight);

N = length(serie);

diffSerie = diff(serie);
nPriceDiff = zeros(1, N);

for i = 1:samples
        nPriceDiff = ...
                + nPriceDiff ...
                + [ ...
                        zeros(1,i) ...
                        diffSerie(1:end-i+1)* ...
                        weight(samples-i+1) ...
                  ];
end

% Señal
signal = zeros(1, N);
signal(nPriceDiff>0) = 1;
signal(nPriceDiff<0) = -1;

% Actualiza el valor de la propiedad Signal
algoTrader.Signal = signal;

end
```

Código 5.5: Método `computeSignal` de la clase `LastNWeighted`

Figura 5.25: Proporción relativa de las últimas n variaciones del precio

parámetro p de la ecuación indica el número de periodos que se consideran hacia el pasado. Así, el principio en que se basa la estrategia es el de frenar las operaciones de mercado cuando la tendencia sea lateral y favorecerlas cuando la tendencia sea alcista o bajista, como consecuencia de la fuerte inercia en el movimiento del precio de la acción.

En la figura 5.25 se ilustra la manera en que funciona la estrategia. Si los trazos de la curva del precio son de color verde cuando sube y rojos cuando baja, entonces la estrategia acierta en su predicción. A partir de la figura se observa que la estrategia no está siempre dentro del mercado.

$$Señal_i = \begin{cases} +1 & \text{, si } \frac{\sum_j precio_j}{\sum_k precio_k} > ratio_{rf},\ \forall j,k \in [i-p,i] \mid precio_j > 0 \wedge precio_k < 0 \\ -1 & \text{, si } \frac{\sum_j precio_j}{\sum_k precio_k} > ratio_{fr},\ \forall j,k \in [i-p,i] \mid precio_j < 0 \wedge precio_k > 0 \\ 0 & \text{, en otro caso} \end{cases} \tag{5.34}$$

La clase `LastNRatio`

La clase `LastNRatio` hereda de `AlgoTrader`. En la implementación de esta clase se introducen las siguientes propiedades y métodos:

`Samples` Propiedad para fijar el número de periodos hacia el pasado que la máquina de trading considera en el cálculo de la señal. Aparece en la ecuación 5.34 como el parámetro p.

`RiseFallRatio` Propiedad que indica la proporción mínima de subidas del precio frente a bajadas para tomar una posición larga. Se refiere al parámetro $ratio_{rf}$ en la ecuación 5.34.

`FallRiseRatio` Propiedad que marca la proporción mínima de bajadas del precio frente a subidas para adoptar una posición corta. Se corresponde con $ratio_{fr}$ en la ecuación 5.34.

`computeSignal` Método que computa la señal de posición para la estrategia. La implementación de este método se hace según establece la ecuación 5.34.

Capítulo 6

Estrategias de trading compuestas

Cada estrategia de trading presentada en el capítulo 5 explota alguna característica de la serie de precios de la acción: la media móvil sigue la tendencia, el indice de fuerza relativa mide la velocidad y el cambio de los movimientos del precio, el estocástico sigue la velocidad del impulso del precio... No obstante, hay que ser consciente de las limitaciones. En este sentido, los autores señalan que las estrategias deben emplearse con rigor y en combinación con otros criterios. Esto es así porque al centrarse cada estrategia en una característica de la serie de precios, pasa a obviar todas las demás. En consecuencia, ninguna de las estrategias se muestra infalible por sí sola.

Otra cuestión que reduce el éxito de las estrategias es su comportamiento estático ante un mercado cambiante. Una estrategia de trading consiste en una herramienta de apoyo al inversor que emite señales para orientarle en la toma de decisiones. Las estrategias utilizan parámetros en sus procedimientos de cálculo. El valor de un parámetro puede ser adecuado ante cierta situación en el mercado, sin embargo, conforme cambia la situación no se puede esperar que el mismo valor siga siendo igual de efectivo, sino que debe adaptarse.

Este capítulo explica la manera en que se afrontan los casos mencionados en los párrafos anteriores con la plataforma. En la sección 6.1 se examina la asociación de estrategias de trading como un conjunto que funciona de forma armónica. Este principio de asociación es la base de las estrategias siguientes. En la sección 6.2 se hace trabajar al conjunto de manera colaborativa a través de la intersección de sus posiciones en el mercado. La sección 6.3 utiliza las estrategias del conjunto de forma competitiva, por medio de la intersección de los mejores resultados. Otra posibilidad se estudia en la sección 6.4, donde las estrategias deciden por votación las posiciones que toman como conjunto. La sección 6.5 se centra en la adaptación de los parámetros a la situación del mercado. Por último, en la sección 6.6 se discute sobre la interpretación de las máquinas compuestas como operadores en el sentido matemático. Esta sección proporciona un enfoque muy intuitivo para comprender las composiciones.

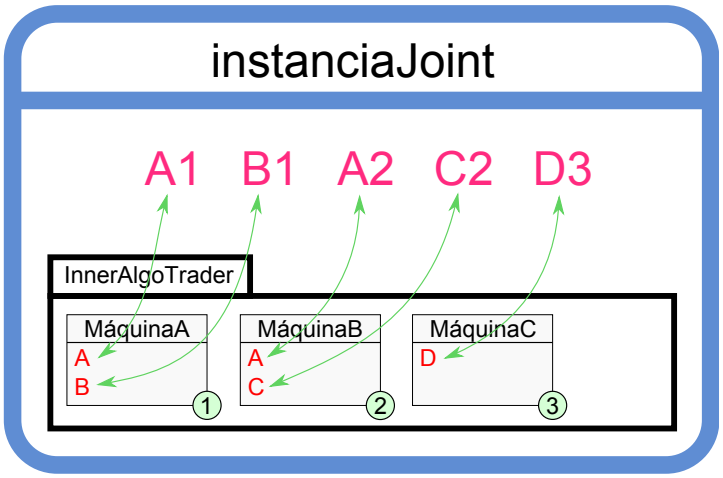

Figura 6.1: Gestión de propiedades dinámicas en la clase `Joint`

6.1. Conjunto

Dado que las estrategias de trading exhiben debilidades al ser utilizadas de forma individual, cabe preguntarse cómo será su comportamiento en conjunto. El principio que aparece tras esta propuesta es el siguiente: si en un conjunto de máquinas de trading cada una de ellas sufre de ciertas debilidades y posee ciertas fortalezas, entonces pueden existir maneras de combinarlas para que se complementen. Se busca que como conjunto reduzcan sus debilidades y potencien sus fortalezas, para así obtener mejores resultados.

En resumen, la agrupación de estrategias da lugar a una estrategia compuesta, la cual calcula la señal de posición a partir de las contribuciones individuales.

La clase `Joint`

La clase abstracta `Joint` es la manera en que se representa al conjunto de estrategias que operan de forma solidaria. Esta clase hereda de `AlgoTrader` y su objetivo es el de servir como contenedor. Todas aquellas estrategias que se albergan en la clase `Joint` deben compartir la misma fuente de datos, es decir, han de operar sobre la misma instancia de la clase `DataSerie`.

Una característica especial de esta clase es que facilita la interacción con las estrategias que acoge. En particular, la clase `Joint` crea y gestiona propiedades dinámicas que enlazan con aquellas de las máquinas de trading del conjunto: cualquier operación de lectura o escritura sobre estas propiedades de enlace repercuten sobre la propiedad enlazada. Por ejemplo, si la máquina de trading `MáquinaA` cuenta con las propiedades `A` y `B`, la máquina `MáquinaB` con las propiedades `A` y `C`, y la máquina `MáquinaC` con la propiedad `D`, entonces una instancia de la clase `Joint` que incluya estas máquinas genera las propiedades dinámicas `A1`, `B1`, `A2`, `C2` y `D3`. Tal situación se ilustra en la figura 6.1, donde aparece una instancia de esta clase almacenando las máquinas en la estructura de tipo array `InnerAlgoTrader`. En este caso, la propiedad `A` de la máquina `MáquinaB` se accede desde `instanciaJoint.InnerAlgoTrader{2}.A`, o como `instanciaJoint.A2` si se hace uso de las propiedades dinámicas de enlace.

La clase `Joint` y el mecanismo de propiedades dinámicas abre un nuevo horizonte, permitiendo entre otros la optimización simultanea de varias máquinas de trading, no para obtener sus óptimos individuales, sino para computar el óptimo del colectivo.

Más allá de implementar un contenedor, esta clase no define la manera en que se combinan las señales de posición de las máquinas. Es responsabilidad del usuario extender la clase e implementar el método abstracto `computeSignal` para especificar cómo se calcula la señal de posición del conjunto.

Las propiedades y métodos de la clase `Joint` se listan a continuación:

`InnerAlgoTrader` Propiedad que se refiere a la estructura de datos donde se almacenan las máquinas de trading. Consiste en una estructura de acceso lineal de tipo array.

`DynamicProperties` Propiedad privada utilizada en la gestión de las propiedades dinámicas. El proceso de creación y eliminación de propiedades dinámicas es transparente al usuario. Bajo ningún concepto se precisa de utilizar la información contenida en esta propiedad.

`add` Método que sirve para agregar nuevas máquinas de trading al conjunto. Tras las incorporaciones se generan las propiedades dinámicas pertinentes. Las nuevas máquinas de trading tienen que utilizar la misma serie de datos que las demás. Este método almacena copias de las máquinas introducidas como argumentos.

`clone` Método para producir una copia de la máquina contenedor y de todas las máquinas que contiene. El proceso de copia es recursivo y afecta a todas las estrategias contenidas en la jerarquía.

`recursiveUpdate` Método privado que actualiza el valor de las propiedades de las máquinas de trading en el conjunto. De esta forma, los cambios en las propiedades `TrainingSet`, `TestSet`, `InitialFunds`, `InvestmentLimit` y `TradingCost` de la instancia de la clase `Joint` se transmiten hacia aquellas máquinas que contiene.

`remove` Método que elimina máquinas de trading del conjunto. Para suprimir una máquina hay que referenciarla sobre la estructura de datos, es decir, hay que indicar la posición que ocupa en el array `InnerAlgoTrader`.

`computeSignal` Método abstracto de la clase `Joint` destinado a implementar en las clases herederas el mecanismo de generación de la señal de posición.

`updateDynamicProperties` Método privado que actualiza las propiedades dinámicas de la instancia de clase. Actúa en las llamadas a los métodos `add` y `remove`.

`updateSignal` Método privado que actualiza la señal de posición con cada cambio.

6.2. Intersección

Una manera de utilizar varias máquinas de trading consiste en adoptar posiciones en el mercado sólo cuando exista consenso entre ellas, es decir, cuanto todas las máquinas sugieran adoptar la misma posición. Este procedimiento para combinar resultados es el más conservador, pues requiere de la unanimidad en los resultados de las estrategias.

Generación de la señal

El cálculo de la posición de la máquina de intersección es como sigue: si todas las máquinas del conjunto están en largo, entonces esta máquina también se posiciona en largo; si por el contrario todas las máquinas toman posiciones cortas, entonces esta máquina abre posición corta; en los demás casos, está fuera del mercado. Este comportamiento queda reflejado en la ecuación 6.1.

Un ejemplo de funcionamiento de la estrategia se ilustra en la figura 6.2. Ahí aparecen cuatro recuadros, donde los tres superiores se refieren a las posiciones de las estrategias del conjunto y el inferior es el resultado de su combinación con la operación de intersección. En cada recuadro, los rectángulos en verde representan posiciones largas y los rojos cortas. Las líneas verticales discontinuas sirven como apoyo. Desde la perspectiva visual, los rectángulos de posiciones de esta estrategia son fruto de la intersección de los rectángulos de las estrategias del conjunto. La vista en detalle de cómo se combinan las señales aparece en la figura 6.3.

$$Señal_i = \begin{cases} +1 & \text{, si } Señal_{m,i} = +1, \quad \forall m \in \text{“Máquinas de trading”} \\ -1 & \text{, si } Señal_{m,i} = -1, \quad \forall m \in \text{“Máquinas de trading”} \\ 0 & \text{, en otro caso} \end{cases} \tag{6.1}$$

La clase `Intersection`

La clase `Intersection` implementa la máquina de intersección. Hereda de la clase `Joint` e implementa un único método:

`computeSignal` Método que computa la señal de posición en la estrategia. Realiza los cálculos conforme aparece en la ecuación 6.1.

6.3. Intersección elitista

En un conjunto de estrategias cada una de ellas tiene un rendimiento diferente: unas tienen más acierto y contribuyen de manera positiva al conjunto, mientras que otras son nefastas y perjudican al resto. Sin las herramientas adecuadas, estudiar este comportamiento colectivo es una tarea compleja.

La intersección elitista es una extensión de la estrategia anterior: consiste en tomar sólo un subconjunto de las estrategias en el cálculo de la señal de posición. De esta manera, aquellas estrategias cuya aportación sea negativa pueden ser deshabilitadas.

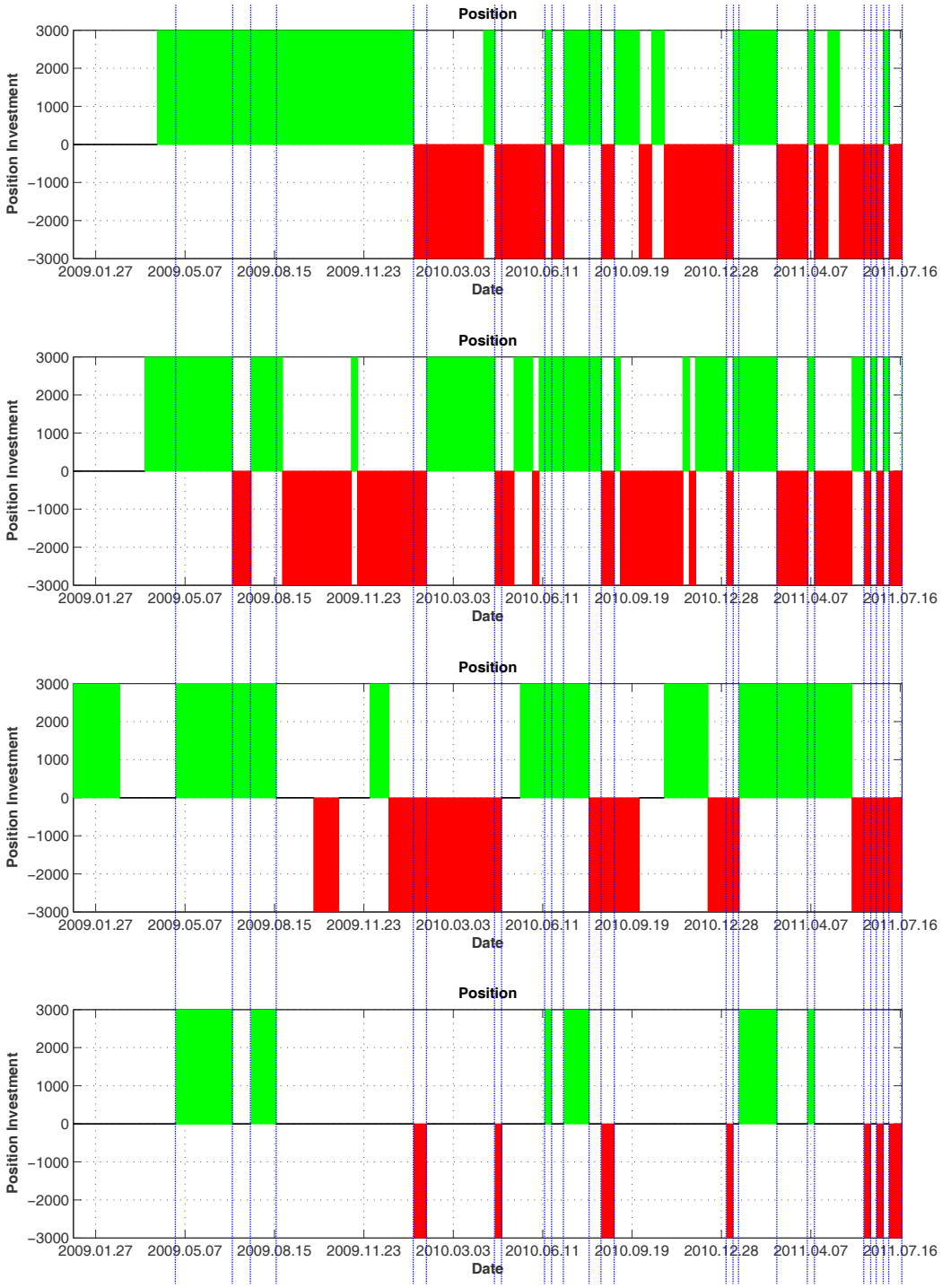

Figura 6.2: Intersección de las posiciones de las estrategias

Figura 6.3: Intersección de señales de posición

Un aspecto importante de la intersección elitista es que introduce un nuevo problema: la selección del subconjunto óptimo de estrategias que mejor desempeño obtiene. Este problema consiste en tomar las estrategias que mejor se complementan con la intersección de sus señales. El número de maneras en que puede ser elegido este subconjunto es 2^n, con n el número de máquinas en el conjunto.

Generación de la señal

El cálculo de la señal de posición conjunta es similar al de la estrategia anterior: la diferencia consiste en que la intersección se hace con las señales de posición del subconjunto de máquinas seleccionadas. La fórmula para computar la señal se muestra en la ecuación 6.2. Esta máquina relaja las exigencias para tomar posiciones en el mercado, pues la intersección de un subconjunto es menos restrictiva que la del conjunto al completo.

El ejemplo de actuación de esta estrategia se encuentra en la figura 6.4. Los tres recuadros superiores marcan las posiciones de las estrategias del conjunto. El segundo recuadro expresa con el color gris que la estrategia está fuera del subconjunto seleccionado: se ha deshabilitado. Como resultado, en el recuadro inferior aparece la señal de posición del conjunto, que consiste en la intersección de las máquinas primera y tercera. La figura 6.5 ofrece una imágen descriptiva del proceso de cálculo.

$$Señal_i = \begin{cases} +1 & \text{, si } Señal_{\hat{m},i} = +1, \quad \forall \hat{m} \in \text{“Máquinas de trading seleccionadas”} \\ -1 & \text{, si } Señal_{\hat{m},i} = -1, \quad \forall \hat{m} \in \text{“Máquinas de trading seleccionadas”} \\ 0 & \text{, en otro caso} \end{cases} \tag{6.2}$$

La clase `ElitistIntersection`

En la clase `ElitistIntersection` se implementa la estrategia de la intersección elitista. Esta clase hereda de `Intersection` y extiende de forma significativa su funcionalidad a partir de las propiedades y métodos siguientes:

`Selection` Propiedad que se utiliza para marcar las máquinas del conjunto como habilitades o deshabilitadas. Con esta propiedad se establece el subconjunto seleccionado. Consiste en un array de tipo binario cuya longitud es el número de máquinas

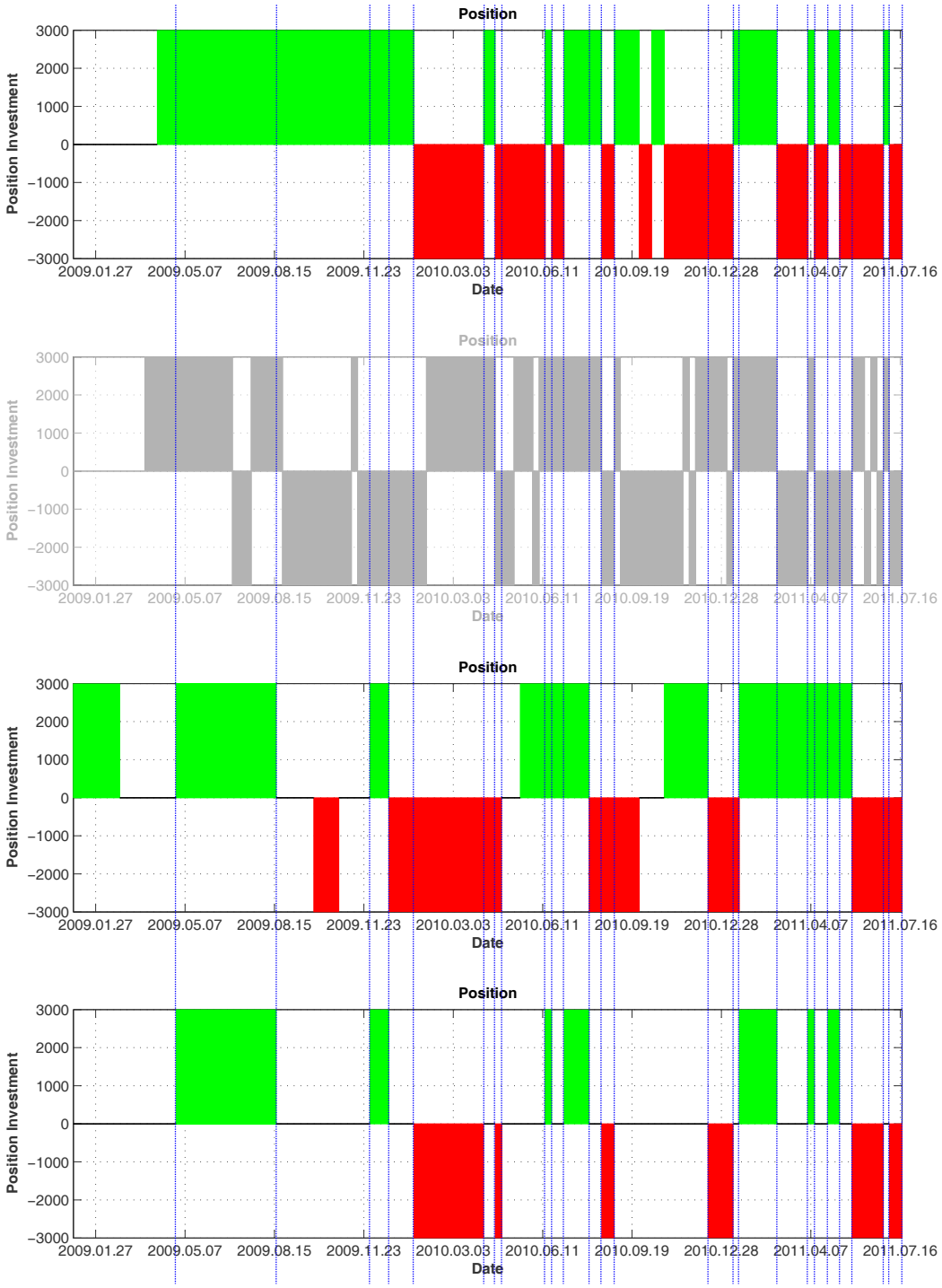

Figura 6.4: Intersección elitista de las posiciones de las estrategias primera y tercera

Figura 6.5: Intersección elitista de señales de posición

almacenadas en la propiedad `InnerAlgoTrader`. Una máquina contenida en la posición i del array `InnerAlgoTrader` es parte del subconjunto sólo si el valor de la posición i del array `Selection` es `true`; si este valor es `false`, entonces la máquina se excluye.

`add` Método con el que se agregan nuevas máquinas de trading en el conjunto. Las nuevas máquinas están inicialmente habilitadas.

`optimizeSelection` Método que computa la selección del subconjunto óptimo de máquinas que mejoran el resultado. Admite como argumentos la función de evaluación, la función de selección y el método de optimización. Más allá de estos, no precisa de ningún argumento adicional. Las llamadas a este método mantienen el esquema que se sigue con el método `optimize`, que fue explicado en la sección 3.3.

`remove` Método que elimina máquinas de trading del conjunto. Para suprimir una máquina hay que referenciarla sobre la estructura de datos, es decir, hay que indicar la posición que ocupa en el array `InnerAlgoTrader`.

`computeSignal` Método que computa la señal de posición de la máquina de trading de intersección elitista. El cálculo es similar al de la estrategia anterior, con la salvedad de que toma sólo las máquinas del subconjunto seleccionado, es decir, aquellas que están habilitadas. En la ecuación 6.2 se recoge el método de cálculo seguido.

6.4. Votación

La estrategia de votación es la última que se define para conjuntos de estrategias. Esta se basa en decidir la señal de posición que toma el conjunto a partir de votaciones. En consecuencia, la posición que respalda el agregado es aquella más popular entre las estrategias individuales. La estrategia de votación guarda relación con la estrategia de la intersección: esta última es un caso especial de la primera donde las posiciones se adoptan ante el consenso en la votación.

Generación de la señal

En este caso, el cálculo de la señal resultante es algo más complejo, debido a que hay que idear un sistema de votación para las estrategias. En la votación se procede de la siguiente manera: si hay una mayoría de estrategias que votan por una misma posición y además el umbral del mínimo de votos es superado, entonces la posición que se adopta es la votada. Por otra parte, se permanece fuera del mercado si se produce un empate o si no se alcanzan los votos mínimos para aprobar una posición. El umbral de votos se utiliza para exigir un nivel de acuerdo mínimo entre las estrategias antes de abrir una posición. Este proceso queda recogido en la ecuación 6.3. Los elementos que aparecen en dicha ecuación son: el número de votos de posición larga S_i^+, el número de votos de posición corta S_i^-, el número de votos para no entrar en el mercado $S_i^\varnothing$ y el umbral del mínimo de votos V. La descripción matemática de estos elementos aparece en las ecuaciones 6.4, 6.5, 6.6 y 6.7.

El sistema de votación aparece ejemplificado en la figura 6.7. Una instancia real de la máquina que implementa esta estrategia se muestra en la figura 6.6. De nuevo, los tres recuadros superiores son las estrategias del conjunto, mientras que el último consiste en el resultado alcanzado mediante la votación, donde el mínimo para alcanzar un acuerdo son dos votos.

$$Señal_i = \begin{cases} +1 & \text{, si } S_i^+ > S_i^- \wedge S_i^+ > S_i^\varnothing \wedge S_i^+ \geq V \\ -1 & \text{, si } S_i^- > S_i^+ \wedge S_i^- > S_i^\varnothing \wedge S_i^- \geq V \\ 0 & \text{, en otro caso} \end{cases} \tag{6.3}$$

$$S_i^+ = |\{m \mid Señal_{m,i} = +1,\ \forall m \in \text{“Máquinas de trading”}\}| \tag{6.4}$$

$$S_i^- = |\{m \mid Señal_{m,i} = -1,\ \forall m \in \text{“Máquinas de trading”}\}| \tag{6.5}$$

$$S_i^\varnothing = |\{m \mid Señal_{m,i} = 0,\ \forall m \in \text{“Máquinas de trading”}\}| \tag{6.6}$$

$$V = r \cdot |\text{“Máquinas de trading”}| \tag{6.7}$$

La clase `Voting`

La clase `Voting` hereda de `Joint` e implementa la estrategia de votación. Las propiedades y métodos que define aparecen listados a continuación:

`Ratio` Propiedad con que se indica la proporción del mínimo de votos para acordar una posición. El mínimo de votos (parámetro V) se calcula multiplicando esta proporción (parámetro r) por la cantidad de máquinas en el conjunto, según refleja la ecuación 6.7.

`computeSignal` Método que computa la señal de posición con la votación de cada máquina. El procedimiento que sigue aparece en la ecuación 6.3.

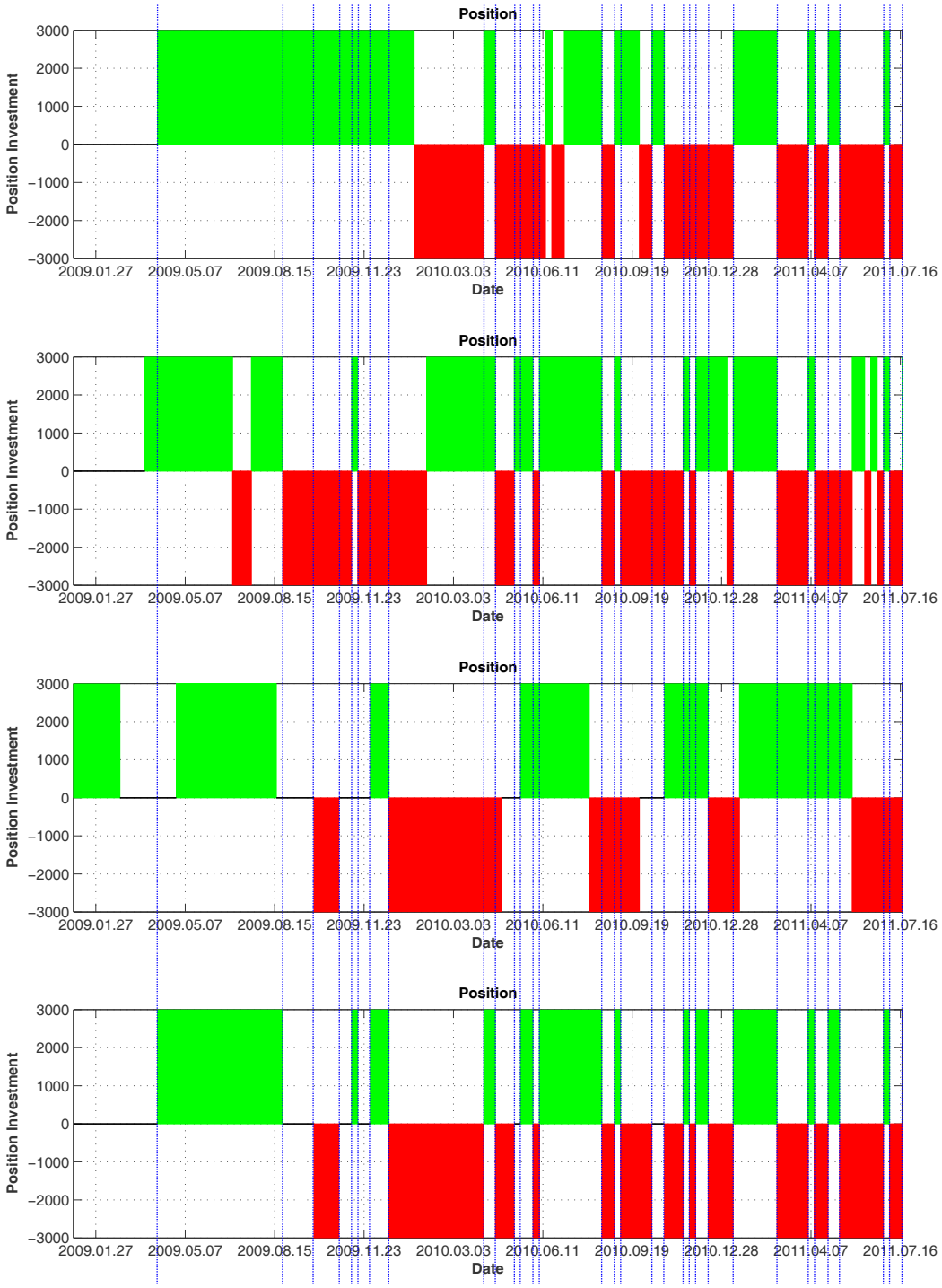

Figura 6.6: Votación de las posiciones de las estrategias

Figura 6.7: Votación de señales de posición

6.5. Fragmentación

El mercado bursátil es un ente vivo que absorbe las noticias sobre finanzas. En él, las novedades que afectan al precio de una acción se descuentan de inmediato. En consecuencia, los valores de los parámetros de las estrategias han de adaptarse para obtener el máximo provecho.

Sirva como ejemplo la figura 6.8, cuyos dos recuadros contienen la misma curva de precios. Se aplica en ambas series la estrategia de la media móvil y se optimiza el número de periodos para mejorar el resultado, aunque con una diferencia: en el recuadro superior la optimización se hace sobre la primera mitad de la serie y en el inferior sobre la segunda mitad. Existe una diferencia sustancial en el número óptimo de periodos en cada caso.

El proceso de adaptación de los parámetros consiste en adecuar sus valores para hacerlos cercanos al óptimo. En este sentido, el esquema aplicado en la plataforma aparece en la figura 6.9, donde la serie de datos con el precio de la acción en el tiempo se divide en dos intervalos disjuntos: el de entrenamiento y el de test. Luego de la división, el proceso de optimización calcula los valores óptimos en el intervalo de entrenamiento y los utiliza en el intervalo de test. Los problemas principales de este enfoque son dos:

- Los valor óptimos así computados se vuelven más inapropiados conforme los datos sobre los que se aplican se alejan en el tiempo del intervalo de entrenamiento. Esto sucede porque la fórmula del cálculo de la señal de posición toma valores actuales del precio y utiliza parámetros cuyos valores óptimos fueron computados para valores pasados del precio.

- El tamaño del intervalo de entrenamiento repercute de manera negativa si es demasiado grande. Como el mercado se ve inmerso en diferentes situaciones a lo largo del tiempo, entonces un intervalo de entrenamiento demasiado ámplio contiene situaciones muy dispares. Esto significa que los valores óptimos calculados en este intervalo no son apropiados para ninguna situación concreta en el mercado.

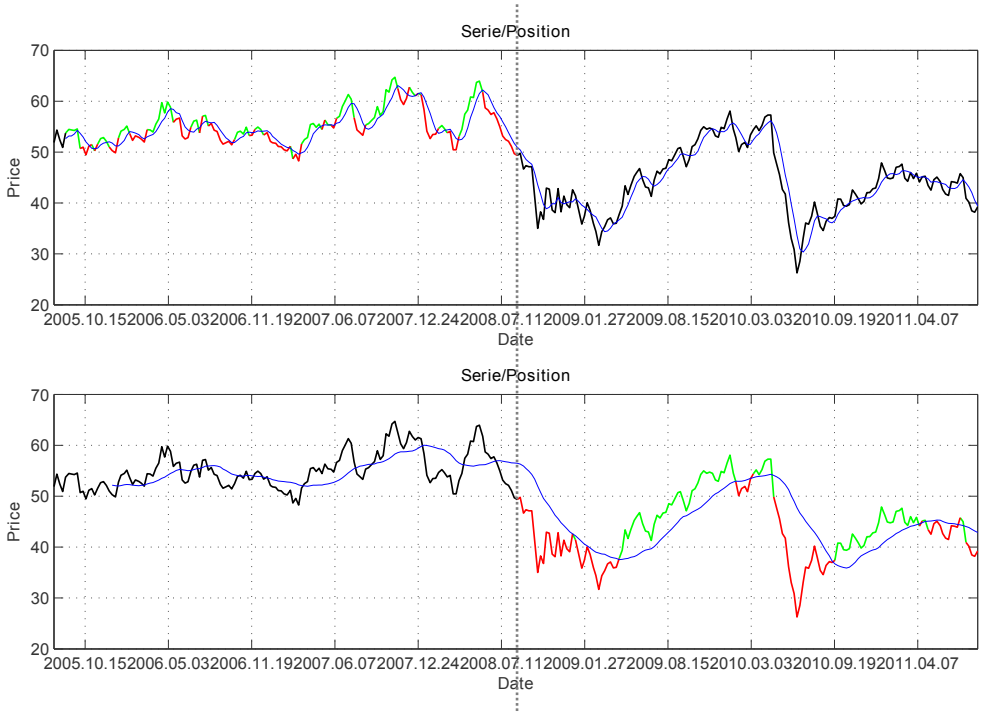

Figura 6.8: Valor óptimo de la media móvil según intervalo

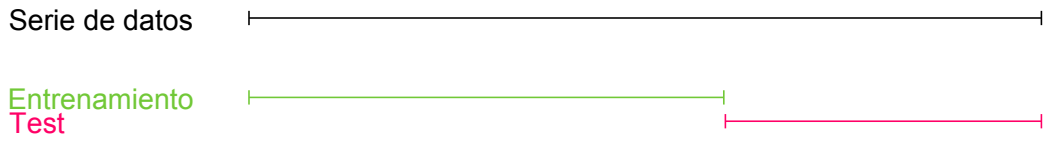

Figura 6.9: División de una serie de datos en los intervalos de entrenamiento y test

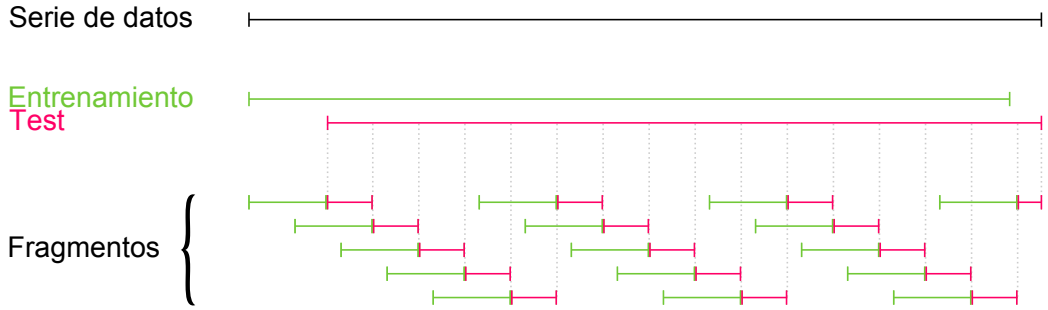

Figura 6.10: División de una serie de datos en múltiples intervalos de entrenamiento y test

#fragmento	Intervalo de entrenamiento	Intervalo de test
1	[1 40]	[41 70]
2	[31 70]	[71 100]
3	[61 100]	[101 130]
4	[91 130]	[131 160]
5	[121 160]	[161 190]
6	[151 190]	[191 200]

Tabla 6.1: División de una serie de datos de 200 valores en intervalos de entrenamiento de longitud 40 e intervalos de test de longitud 30 para cada fragmento

Para superar estas dos dificultades la plataforma proporciona la máquina de fragmentación. La manera en que esta máquina se sobrepone a las limitaciones explicadas es mediante la partición de los intervalos de entrenamiento y test. En contraposición a lo que ocurre en la figura 6.9, la máquina de fragmentación realiza una división de estos intervalos como la que se muestra en la figura 6.10. Los múltiples intervalos de entrenamiento pueden solaparse, si bien los de test han de ser contiguos. Las ventajas de la fragmentación son las siguientes:

- Los intervalos de entrenamiento y test son pequeños y se adaptan mejor a la situación que atraviesa el mercado bursátil.
- Los valores óptimos de cada intervalo de entrenamiento se aplican sólo en el intervalo de test correspondiente: se incrementa la validez y calidad del valor óptimo usado para el parámetro.
- Si se tiene en cuenta el paso del tiempo, entonces la estrategia de fragmentación calcula antes los óptimos y los emplea más rápido que en la versión no fragmentada. Esto se debe a que en la fragmentación se intercalan los intervalos de entrenamiento y test.
- El proceso de fragmentación es transparente y se puede aplicar a cualquier estrategia.

Para implementar este mecanismo, la máquina de fragmentación crea múltiples copias de la máquina de trading de partida y utiliza cada una de ellas sobre un intervalo de entrenamiento y test diferente. Estas copias reciben el nombre de fragmentos.

En la tabla 6.1 aparece un ejemplo que muestra cómo es la partición de una serie de datos con 200 valores del precio de una acción tomando fragmentos con intervalos de entrenamiento de longitud 40 e intervalos de test de longitud 30. Es importante que los intervalos de entrenamiento y test sean disjuntos, pues de lo contrario el proceso de entrenamiento conoce algunos datos del intervalo de test y el resultado queda falseado.

Generación de la señal

En el proceso de generación de la señal de posición, la máquina de fragmentación une las señales que las máquinas de fragmentos han creado. El proceso de reconstrucción consiste en copiar de cada máquina fragmento la señal de posición que ha computado para su intervalo de test. Este proceso se resume en la ecuación 6.8.

$$Señal_i = Señal_{m,i}, \quad i \in \text{“Intervalo de test del fragmento } m\text{”} \tag{6.8}$$

La clase `Fragmented`

Con la clase `Fragmented` se implementa el mecanismo de fragmentación de máquinas de trading. Esta clase hereda de `AlgoTrader`. El conjunto de propiedades y métodos que crea es el siguiente:

`Fragment` Propiedad que consiste en una estructura de datos lineal donde se almacenan las máquinas fragmento con los diferentes intervalos de entrenamiento y test.

`InnerAlgoTrader` Propiedad que almacena una copia de la máquina de trading sobre la que se aplica el proceso de fragmentación. Se utiliza para crear los fragmentos que se guardan en `Fragment`. Cada vez que cambia el tamaño del intervalo de entrenamiento o el de test se vuelven a generar las máquinas alojadas en la estructura `Fragment`.

`TrainingSetSize` Propiedad que indica el tamaño de los intervalos de entrenamiento.

`TestSetSize` Propiedad que determina el tamaño de los intervalos de test.

`clone` Método que crea una copia de la máquina de fragmentación y de sus fragmentos.

`fitnessStatistics` Método que calcula estadísticas acerca de la función de evaluación.

`optimize` Método que computa el valor óptimo de las propiedades de la máquina de fragmentación y de sus fragmentos.

`plotSearchSpace` Método que representa el espacio de búsqueda de donde se obtienen los valores óptimos de los parámetros. Recrea espacios de cualquier dimensión descomponiéndolos en subespacios de dos dimensiones.

`plotSearchSpace123` Método que representa el espacio de búsqueda en donde se quieren hallar los valores óptimos de los parámetros. Es parecido al método `plotSearchSpace`, con la salvedad de que sólo dibuja espacios de una, dos y tres dimensiones.

`resetFragment` Método privado que se utiliza para eliminar la estructura de datos contenida en la propiedad `Fragment`. Se llama cuando hay que recrear las máquinas fragmento con un tamaño diferente para el intervalo de entrenamiento o test.

`computeSignal` Método que computa la señal de posición de la máquina de trading de fragmentación. Construye esta señal como se recoge en la ecuación 6.8.

`updateFragment` Método privado que actualiza la estructura de datos `Fragment`. Es responsable de crear los fragmentos con sus respectivos intervalos de entrenamiento y test.

`updateSignal` Método privado que actualiza la señal de posición con cada cambio.

`wideGet` Método que recoge el valor de una propiedad desde los múltiples fragmentos de la máquina de fragmentación.

`wideSet` Método que asigna un valor a la propiedad de todos los fragmentos.

6.6. Máquinas compuestas interpretadas como operadores

Todas las máquinas de trading de este capítulo se definen a partir otras. Así, la máquina de conjunto y las que heredan de esta (la máquina de intersección, la de intersección elitista y la de votación) requieren de una colección de máquinas iniciales. Por su parte, la máquina de fragmentación precisa de una única máquina de partida.

Dada esta dependencia, las máquinas compuestas se pueden interpretar como operadores entre máquinas de trading. Estos operadores se encargan de combinar las máquinas de trading de alguna manera para dar como resultado una nueva máquina. Con esta visión se entiende a la combinación de máquinas como una extensión natural de las máquinas de trading.

6.6.1. Operadores de intersección, intersección elitista y votación

El primer grupo de operadores se debe a aquellas máquinas del tipo conjunto: intersercción, intersección elitista y votación. Estos operadores toman un número variable de máquinas de trading y devuelven otra máquina, tal y como se recoge en la ecuación 6.9.

$$Joint : M_1 \times M_2 \times \ldots \times M_n \rightarrow M \tag{6.9}$$

Los operadores se representan gráficamente como árboles de dos niveles, donde la raíz es el operador y las hojas las máquinas que toma. En la figura 6.11 se enseña cómo es la imagen que simboliza a todos los operadores de esta familia.

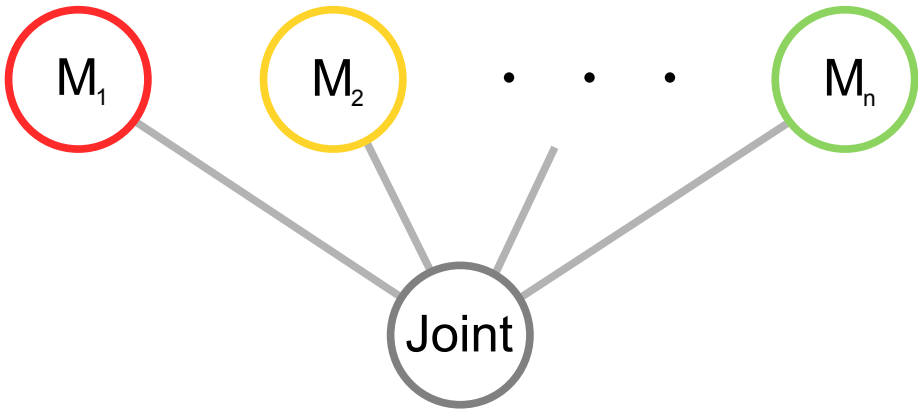

Figura 6.11: Representación en árbol del operador de conjunto

6.6.2. Operador de fragmentación

Este operador es consecuencia de la máquina de fragmentación. A diferencia de los anteriores, el operador de fragmentación es unario y toma una única máquina de trading para devolver otra fragmentada. Esta situación queda reflejada en la ecuación 6.10.

$$Fragmented : M \rightarrow M \tag{6.10}$$

El operador de fragmentación se representa como un árbol de dos niveles: en el primer nivel el operador y en el segundo la máquina a fragmentar. Este gráfico se muestra en la figura 6.12.

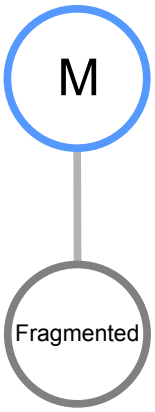

Figura 6.12: Representación en árbol del operador de fragmentación

6.6.3. Combinando operadores

Al igual que sucede en matemáticas, los operadores se pueden combinan en expresiones. Estos operadores toman cualquier tipo de máquina de trading, sea esta una simple o una compuesta, y devuelven una nueva máquina. No hay limitación en cuanto al nivel de anidamiento de máquinas, aunque combinaciones con demasiados niveles pierden el sentido y la utilidad.

Como ejemplo para este último apartado se presenta la figura 6.13, que es la representación en forma de árbol de una máquina de trading compuesta. Esta máquina consiste en la combinación de seis máquinas de trading básicas por medio de los operadores de intersección, fragmentación y votación. En los árboles como este, las hojas siempre contienen máquinas no compuestas, pues de lo contrario dichos nodos no serian hojas.

Además de la representación en árbol, la composición también se puede describir como una expresión matemática. Para hacerlo hay que escribir los operadores en forma de funciones con los argumentos entre paréntesis. En la ecuación 6.11 aparece la expresión cuya información es equivalente a la de la figura 6.13.

$$Voting(Intersection(M_1, M_2, M_3), Fragmented(M_4), Fragmented(M_5), M_6) \tag{6.11}$$

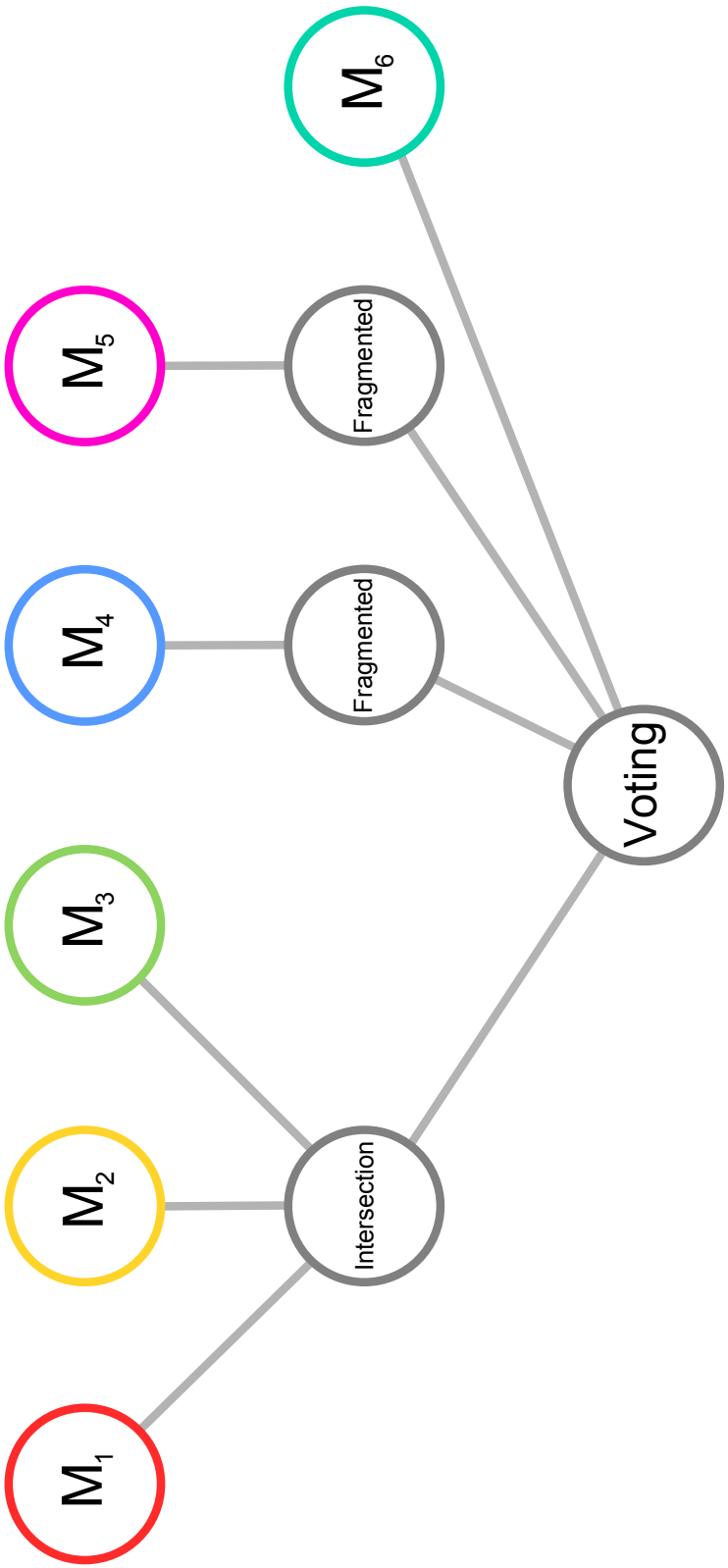

Figura 6.13: Representación en árbol de una combinación de operadores

Capítulo 7

Optimización de parámetros

En la sección 3.3 se habló sobre la optimización de parámetros como parte de la funcionalidad provista por la plataforma. Además, se explicó el proceso para optimizar máquinas de trading, aplicable a aquellas desarrolladas en los capítulos 5 y 6. Como se dijo, el método `optimize` es el responsable de lanzar el proceso de optimización, para lo cual precisa de tres funciones: la función de evaluación, que cuantifica el desempeño de la estrategia de acuerdo a cierto criterio; la función de selección, que sirve para indicar si se busca maximizar o minimizar el resultado; y la función que implementa el método de optimización. A modo de recordatorio, se muestra la forma de una llamada al método `optimize`:

```
algoTraderInstance.optimize(@evaluationFunction, ...
                        @selectionFunction, ...
                        @optimizationMethod, ...
                        'Parameter1', Domain1, ...
                        'Parameter2', Domain2, ...
                        ...
                        'ParameterN', DomainN)
```

En este capítulo se profundiza en la explicación del mecanismo de optimización y sus partes constituyentes. La sección 7.1 discute la categoría del problema así como su nivel de complejidad. En la sección 7.2 se desarrollan algunas de las funciones de evaluación implementadas en la plataforma. Luego, en la sección 7.3, se habla de las dos funciones de selección. Los métodos de optimización, que constituyen la parte más compleja del capítulo, se describen en profundidad en la sección 7.4.

7.1. Sobre la naturaleza del problema

El problema de asignar valores a los parámetros de una máquina de trading, para mejorar su desempeño según cierto criterio, es un problema de optimización combinatoria. En términos de complejidad computacional, este problema es de la clase *NP-hard* [29].

En matemáticas aplicadas y en ciencias de la computación, la optimización combinatoria es una rama que consiste en encontrar la solución óptima de entre un conjunto de soluciones. Los problemas que resuelve la optimización combinatoria tienen como característica que su conjunto de soluciones factibles es discreto o se puede reducir a un conjunto discreto. Es requisito para resolverlos el contar con una función que cuantifique la calidad de una solución.

En el contexto de este trabajo, las soluciones son las diferentes asignaciones de valores a los parámetros de la máquina de trading: de entre todas ellas, hay que encontrar la asignación óptima que logra el mejor desempeño de la máquina. En este proceso interviene la función de evaluación para medir la calidad de las soluciones y la función de selección para escoger la solución adecuada [30, 31].

7.2. Funciones de evaluación

Como se ha repetido en diferentes ocasiones, las funciones de evaluación miden el desempeño de la estrategia de trading de acuerdo con algún criterio. Son necesarias en el proceso de optimización de parámetros, pues guían al método de optimización en la búsqueda de aquellas soluciones (en forma de asignaciones de valores a los parámetros de la máquina) que mejor satisfacen los requisitos.

En esta sección se discuten las funciones de evaluación implementadas, el criterio que tienen en consideración y la manera en que son calculadas. Cabe recordar que, como punto de partida, todas las funciones de evaluación han de cumplir la signatura que aparece a continuación:

```
performance = evaluationFunction(positionsLog, ...)
```

El argumento de salida `performance` es del tipo numérico y mide el desempeño de la estrategia. Los argumentos de entrada que admite la función son múltiples, donde el primero, identificado con `positionsLog`, que es obligatorio, se refiere a la tabla con los registros de las posiciones tomadas en el mercado. El resto de argumentos son opcionales y sirven para enviar valores que pueda requerir la función de evaluación.

La tabla de registros que se pasa como argumento ya fue explicada en la sección 3.3. Esta consiste en una matriz de dos dimensiones cuyas filas son los registros ordenados de forma cronológica y las columas los diferentes campos. El orden y la descripción de las columnas es el que aparece en la tabla 7.1. Las columnas de la tabla se calculan con el método `positionsLog` de la máquina de trading.

Para aplicar las funciones de evaluación sobre las máquinas de trading hay que utilizar el método `fitness`. El primer argumento del método es la función de evaluación y el resto de argumentos, que son opcionales, son los que se pasan a la función de evaluación. Información más detallada sobre la utilización de este método aparece en la sección B.2.10.

#columna	Descripción
1	Tipo de posición
2	Índice de apertura
3	Índice de cierre
4	Fecha de apertura
5	Fecha de cierre
6	Precio de apertura
7	Precio de cierre
8	Beneficio/Pérdida

Tabla 7.1: Descripción de las columnas de la tabla de registros de posición

7.2.1. Beneficio/Pérdida

Una máquina de trading opera en el mercado bursátil mediante la compra y venta de acciones. Como resultado de su actividad, esta genera beneficios o incurre en pérdidas. Desde el prisma de las finanzas, la maquina de trading puede entenderse como una inversión con un nivel de riesgo que proporciona rentabilidad sobre el dinero que se invierte. En consecuencia, un criterio para medir el rendimiento de estas máquinas consiste en calcular la rentabilidad de la inversión, que se hace como aparece en la ecuación 7.1.

La función de evaluación de los beneficios y las pérdidas es un procedimiento que calcula la razón entre los fondos finales que conserva la máquina de trading y los fondos iniciales con que contó. A esta razón se la llama el factor de rentabilidad y su cálculo se realiza de acuerdo a la ecuación 7.2. Con la fórmula se deduce lo siguiente: si el factor de rentabilidad es mayor a la unidad, entonces la inversión produce beneficios; en caso contrario, la inversión va asociada a una pérdida de capital. Esta función de evaluación se usa en combinación con la función de selección `max`, pues es una medida que se busca maximizar en las máquinas. El factor de rentabilidad está relacionado de forma estrecha con la rentabilidad, como se desprende de las ecuaciones 7.3 y 7.4.

$$Rentabilidad = \frac{Beneficio/Pérdida}{Inversión} \cdot 100 \tag{7.1}$$

$$Factor\ de\ rentabilidad = \frac{Capital\ final}{Capital\ inicial} = \frac{Beneficio/Pérdida + Inversión}{Inversión} \tag{7.2}$$

$$Rentabilidad = (Factor\ de\ rentabilidad - 1) \cdot 100 \tag{7.3}$$

$$Factor\ de\ rentabilidad = (Rentabilidad/100) + 1 \tag{7.4}$$

Los datos de partida para calcular el factor de rentabilidad de la máquina de trading están contenidos en la tabla de registros, que se pasa a las funciones de evaluación como argumento. En particular, la octava columna de esta tabla almacena el factor de rentabilidad de cada una

```
function performance = profitLoss(positionsLog)

% [ ...
%     positionType, ...
%     openIndex, ...
%     closeIndex, ...
%     openDate, ...
%     closeDate, ...
%     openPrice, ...
%     closePrice, ...
%     profitLoss ...
% ] ...
% = positionsLog;

performance = prod(positionsLog(:,8));

end
```

Código 7.1: Función de evaluación `profitLoss`

de las posiciones que fueron adoptadas por la máquina. Para calcular el factor de rentabilidad de la máquina hay que multiplicar entre sí los factores de rentabilidad de cada una de las posiciones. Esto se resume en la ecuación 7.5 para una tabla de registros con n filas.

$$\begin{aligned} \textit{Factor de rentabilidad} = \; & \textit{Factor de rentabilidad}_1 \\ & \cdot \textit{Factor de rentabilidad}_2 \\ & \vdots \\ & \cdot \textit{Factor de rentabilidad}_n \end{aligned} \tag{7.5}$$

La función `profitLoss`

La función de evaluación `profitLoss` calcula el rendimiento de una máquina de trading según se ha explicado en esta sección. Una posible manera de implementar la función es como aparece en el código 7.1. Se aprecia que sólo es necesaria la última columna de la tabla de registros de posición para el cálculo del valor.

7.2.2. Pérdida

El factor de rentabilidad es una buena medida para cuantificar la calidad de una máquina de trading, sin embargo, sucede que tal medida no aporta mucha información sobre el factor de rentabilidad de cada una de las posiciones que adoptó la máquina en el mercado de acciones. Así, por ejemplo, las máquinas de trading A y B pueden tener un mismo factor de rentabilidad, aunque el factor de rentabilidad de cada una de sus posiciones sea muy diferente, como queda reflejado en la ecuación 7.6. En una situación como esta, aunque ambas máquinas tengan el mismo factor de rentabilidad, se prefiere a la máquina A, pues tiene un comportamiento más

estable. La estabilidad de una máquina se relaciona con el riesgo que se asume al invertir en ella: cuanto más estables son los resultados de la máquina, menor es el riesgo asumido. Este es un factor muy importante que condiciona las decisiones de inversión en finanzas. La rentabilidad y el riesgo son objetivos que están en conflicto el uno con el otro: mayor rentabilidad implica mayor riesgo y viceversa.

En este sentido, un criterio más conservador consiste en medir únicamente las pérdidas. Para calcular este valor, primero hay que explicar la descomposición que se hace del factor de rentabilidad, resumida en la ecuación 7.7. Como punto de partida se tiene la ecuación 7.5, donde el valor se calcula a partir del producto de los factores de rentabilidad de cada posición adoptada por la máquina. A partir de esta ecuación, se agrupan los factores de rentabilidad en dos grupos: por un lado aquellos mayores a la unidad, cuyo producto da lugar al factor de beneficio; por el otro los menores a la unidad, que se multiplican para obtener el factor de pérdida. De esta manera, la función de evaluación de las pérdidas devuelve el factor de pérdida asociada a la máquina de trading, es decir, el producto de aquellas posiciones con factor de rentabilidad menor a la unidad.

$$\begin{aligned} Factor\ de\ rentabilidad_A &= 1{,}8 = 1{,}5 \cdot 1{,}2 \\ Factor\ de\ rentabilidad_B &= 1{,}8 = 0{,}75 \cdot 4 \cdot 0{,}6 \end{aligned} \tag{7.6}$$

$$Factor\ de\ rentabilidad = Factor\ de\ beneficio \cdot Factor\ de\ pérdida \tag{7.7}$$

La función `loss`

La función de evaluación que calcula el factor de pérdidas es `loss`. Para implementar esta función se puede hacer como aparece en el código 7.2. Existe un caso de excepción en el código que se da cuando no hay posiciones con pérdidas, situación ante la cual se devuelve cero.

7.2.3. Mínimo beneficio/pérdida esperado

En el primer criterio de evaluación, consistente en medir el factor de rentabilidad, no se tiene en cuenta la estabilidad de la máquina de trading, pues se ignora el riesgo; en el segundo criterio, que mide el factor de pérdida, se ofrece una visión sesgada al pasar por alto aquellas posiciones que generan beneficios, centrándose sólo en las que dan pérdidas.

Una buena función de evaluación debe tener en cuenta los beneficios y pérdidas de todas las posiciones adoptadas en el mercado (maximizar la rentabilidad) y también la variación de estos valores (minimizar el riesgo). En este sentido, se define la función de evaluación del mínimo beneficio/pérdida esperado como la diferencia entre la media y la desviación estándar de los factores de rentabilidad de las posiciones de mercado. Esta definición queda recogida en la ecuación 7.8. En la fórmula, la media contribuye a medir la rentabilidad, que se quiere maximizar, y la desviación estándar cuantifica la inestabilidad, que se desea minimizar. Esta función promociona a las máquinas de trading más rentables a la par que penaliza a aquellas con comportamiento más impredecible, que a la postre son las que entrañan mayor riesgo.

```
function performance = loss(positionsLog)

% [ ...
%     positionType, ...
%     openIndex, ...
%     closeIndex, ...
%     openDate, ...
%     closeDate, ...
%     openPrice, ...
%     closePrice, ...
%     profitLoss ...
% ] ...
% = positionsLog;

loss = positionsLog(positionsLog(:,8)<1,8);

if isempty(loss)
    performance = 0;
else
    performance = prod(loss);
end

end
```

Código 7.2: Función de evaluación `loss`

$$Mínimo\ beneficio/pérdida\ esperado = Media - Desviación\ estándar \tag{7.8}$$

$$Media = \frac{1}{n} \cdot \sum_{i=1}^{n} Factor\ de\ rentabilidad_i \tag{7.9}$$

$$Desviación\ estándar = \sqrt{\frac{1}{n} \cdot \left(\sum_{i=1}^{n} Factor\ de\ rentabilidad_i^2 \right) - Media^2} \tag{7.10}$$

La función `minProfitLossExpected`

La función de evaluación descrita es `minProfitLossExpected`. En el código 7.3 aparece cómo implementarla. De nuevo, sólo se necesita utilizar la información contenida en la ultima columna de la tabla.

```
function performance = minProfitLossExpected(positionsLog)

% [ ...
%     positionType, ...
%     openIndex, ...
%     closeIndex, ...
%     openDate, ...
%     closeDate, ...
%     openPrice, ...
%     closePrice, ...
%     profitLoss ...
% ] ...
% = positionsLog;

profitLoss = positionsLog(:,8);

meanProfitLoss = sum(profitLoss)/length(profitLoss);

stdProfitLoss = ...
    sqrt(sum((profitLoss-meanProfitLoss).^2) ...
        /length(profitLoss));

performance = meanProfitLoss - stdProfitLoss;

end
```

Código 7.3: Función de evaluación `minProfitLossExpected`

7.3. Funciones de selección

Las funciones de selección determinan la dirección en la búsqueda de la solución. Las dos funciones de selección son `max`, que se usa para maximizar, y `min`, que se empléa para minimizar. Estas funciones están relacionadas con las funciones de evaluación, pues las complementan para definir de forma precisa el objetivo que persigue el algoritmo de optimización. Las funciones de evaluación son las responsables de hacer las mediciones, mientras que las funciones de selección establecen una relación de orden entre estas. El orden inducido entre las mediciones hace posible las comparaciones y la selección del óptimo. Las funciones `max` y `min` son parte del conjunto de funciones de MATLAB.

7.4. Métodos de optimización

Como se menciona en la sección 7.1, se enfrenta un problema de optimización combinatoria cuando se busca optimizar el valor de los parámetros de la máquina de trading. Existen multitud de algoritmos que resuelven este problema de manera más o menos satisfactoria. Esta sección recoge aquellos métodos de optimización desarrollados para utilizar en la plataforma.

Los métodos de optimización requieren de la función de selección, la función de fitness (que la genera el método `optimize`) y las características del espacio de búsqueda (en particular, el tamaño de cada dimensión). Como resultado, el método devuelve la mejor solución encontrada en el espacio de búsqueda. La llamada a los métodos de optimización tienen el siguiente formato:

```
bestIndexArray = optimizationMethod(@selectionFunction,
                    @fitnessFunction, searchSpaceSize, ...)
```

La elección de los métodos implementados en la plataforma se debe a las características del problema que se afronta. En muchas ocasiones, el espacio de búsqueda suele ser relativamente manejable, en cuyo caso, el examinar todas las combinaciones de valores de los parámetros es algo factible. Por el contrario, también hay veces en que el espacio de búsqueda es mucho más grande y un recorrido exhaustivo no es factible. En esta situación, hay métodos de optimización más eficientes que se pueden emplear, como por ejemplo aquellos basados en metaheurísticas.

Un uso especial que se puede dar a un método de optimización es el de simular la conducta de un inversor en el momento de asignar valores a los parámetros de la máquina. El inversor actúa probando diferentes valores para los parámetros y conservando aquella configuración que mejor resultado produce. La utilidad de un método así estriba en proporcionar resultados que sirven de base para comparar con otros métodos o estimar el logro que alcanza un inversor con cierto nivel de habilidad. Para incluir esta posibilidad, la plataforma implementa el método de busqueda aleatoria.

7.4.1. Método enumerativo

El método de optimización enumerativo es tal que, para encontrar la mejor solución a la instancia de un problema, examina todas y cada una de las soluciones candidatas del espacio de búsqueda. Este método encuentra siempre la mejor solución, aunque su complejidad temporal lo hace impracticable para instancias de problemas donde el espacio de búsqueda es relativamente grande.

En el contexto de la optimización de parámetros para las máquinas de trading, el procedimiento que aplica el método es el de probar todas y cada una de las combinaciones de valores para los parámetros. De entre ellas, conserva la asignación que da lugar a los mejores resultados.

El método enumerativo está implementado en la función `exhaustiveSequential`. También existe una versión paralela que, para hacer uso de todas las unidades de procesamiento de la máquina, divide el espacio de búsqueda en partes. La función `exhaustive` contiene dicha implementación paralela.

La importancia de este método es notable: la evaluación de todas las soluciones del espacio de búsqueda se utiliza en el análisis estadístico del método `fitnessStatistics` y en la representación del espacio de búsqueda de los métodos `plotSearchSpace` y `plotSearchSpace123`.

Instrucción de llamada	Valor devuelto
`index2indexArray([5 7 10 3],1)`	`[1 1 1 1]`
`index2indexArray([5 7 10 3],2)`	`[2 1 1 1]`
⋮	⋮
`index2indexArray([5 7 10 3],525)`	`[5 7 5 2]`
⋮	⋮
`index2indexArray([5 7 10 3],1049)`	`[4 7 10 3]`
`index2indexArray([5 7 10 3],1050)`	`[5 7 10 3]`

Tabla 7.2: Resultados de la función `index2indexArray`

La función `exhaustiveSequential`

El método enumerativo para la optimización combinatoria está desarrollado en la función `exhaustiveSequential`. Su implementación es idéntica a la que aparece en el código 7.4, salvo por el detalle de que el bucle principal de la función, en lugar de ser `parfor`, es `for`.

La función auxiliar `index2indexArray` transforma un índice del espacio unidimensional a un conjunto de índices del espacio multidimensional. Conocidos los tamaños de las diferentes dimensiones del espacio multidimensional, esta función sirve para recorrer dicho espacio con el índice de un bucle. En la tabla 7.2 se muestra el valor devuelto conforme a los argumentos de entrada. El primer argumento de la función es un vector con los tamaños de cada una de las dimensiones, el segundo es el índice unidimensional a convertir en el conjunto de indices del espacio multidimensional. El valor de este segundo argumento ha de estar comprendido dentro del intervalo $[1\ \ n]$, donde n es el producto de los tamaños de las dimensiones del espacio de búsqueda, que en el ejemplo es $5 \cdot 7 \cdot 10 \cdot 3 = 1050$.

La función `exhaustive`

La versión paralela del método enumerativo está en la función `exhaustive`, cuya implementación aparece en el código 7.4. El paralelismo se alcanza en este caso utilizando la versión paralela del bucle `for`, que se introduce en el código con la palabra reservada `parfor`. Tal funcionalidad la proporciona la *Parallel Computing Toolbox* de MATLAB.

El código que implementa este método esconde sutilezas para particionar de forma adecuada el espacio de búsqueda. En particular, MATLAB puede referenciar elementos de una estructura de datos multidimensional utilizando un vector de indices, uno para cada dimensión, o un único índice global. No obstante, el proceso de conversión del vector de indices al índice global y viceversa ha de ser realizado con cuidado para que, en todo momento, ambos referencien al mismo elemento de la estructura. Esta tarea es justo la que realiza la función auxiliar `index2indexArray`. La utilidad de la conversión reside en que el bucle `parfor` de MATLAB no aprovecha bien el paralelimo del hardware con estructuras multidimensionales, pero sí lo hace de forma óptima sobre estructuras unidimensionales.

```
function [bestIndexArray, searchSpace] = exhaustive( ...
    selectionFunction, fitnessFunction, searchSpaceSize)

if length(searchSpaceSize) == 1
    searchSpace = zeros(searchSpaceSize,1);
else
    searchSpace = zeros(searchSpaceSize);
end

parfor i = 1:prod(searchSpaceSize)
    indexArray = index2indexArray(searchSpaceSize,i);
    searchSpace(i) = feval(fitnessFunction,indexArray);
end

% Get first best value
[~, index] = selectionFunction(searchSpace(:));
bestIndexArray = index2indexArray(searchSpaceSize,index);

end

function indexArray = index2indexArray(searchSpaceSize, n)

n = n-1;

N = length(searchSpaceSize);
indexArray = zeros(1, N);

for i = N:-1:1
    baseValue = prod(searchSpaceSize(1:i-1));
    idiv = floor(n/baseValue);
    n = n - idiv*baseValue;
    indexArray(i) = idiv;
end

indexArray = indexArray+1;

end
```

Código 7.4: Función de optimización `exhaustive` basada en el método enumerativo

7.4.2. Búsqueda aleatoria

El método de búsqueda aleatoria, también conocido como *Random Search* en la literatura inglesa, se atribuye a Leonid A. Rastrigin [32]. La forma en que funciona este método es generando soluciones de manera aleatoria y conservando, a lo largo de las iteraciones que dura el proceso, la mejor solución encontrada.

Este método, a diferencia del enumerativo, no asegura encontrar la solución óptima, sin embargo, proporciona una solución de compromiso y lo hace de forma rápida con independencia del tamaño del espacio de búsqueda. La calidad del resultado en este caso depende del tamaño del espacio de búsqueda y del número de iteraciones que se realizan. Más aún, si hay una solución óptima en un espacio de búsqueda con n soluciones, entonces la probabilidad de encontrar tal solución efectuando i iteraciones se calcula con la ecuación 7.11. Se observa que esta probabilidad es mayor cuantas más iteraciones se hacen y menor es el espacio de búsqueda.

$$P(\textit{encontrar solución óptima}) = 1 - \left(1 - \frac{1}{n}\right)^{i} \tag{7.11}$$

La función `randomSearch`

La optimización por el método de búsqueda aleatoria se hace con la función `randomSearch`, cuya implementación aparece en el código 7.5. Esta función utiliza un argumento de entrada específico que es `iterations`, con el que establece el número de iteraciones que realiza el método: para pasarle un valor, se utiliza la construcción `{@random, numeroIteraciones}` como tercer argumento del método `optimize`, siendo los dos primeros la función de evaluación y la de selección.

7.4.3. Algoritmo genético

Un algoritmo genético, o *Genetic Algorithm* en inglés, es un método de búsqueda que se inspira en el proceso de la evolución natural. Esta heurística se utiliza habitualmente para generar soluciones a problemas de optimización y búsqueda. Desde su invención por parte de John H. Holland [33], ha sido aplicada de forma exitosa en un ámplio conjunto de problemas.

Metodología

En un algoritmo genético, una población de cadenas (llamadas cromosomas o el genotipo del genoma), que codifican soluciones candidatas (llamadas invidividuos, criaturas o fenotipo) de un problema de optimización, evolucionan hacia mejores soluciones. De forma tradicional, las soluciones se representan con cadenas binarias, aunque otros tipos de codificaciones son posibles. La evolución, que comienza desde una población de individuos generados de forma aleatoria, ocurre con el paso de las generaciones. En cada paso generacional se selecciona de forma estocástica un subconjunto de individuos de la población (de acuerdo a su aptitud) y se modifican (mediante recombinaciones e incluso mutaciones aleatorias) para formar la

```
function bestIndexArray = randomSearch( ...
    selectionFunction, fitnessFunction, searchSpaceSize, ...
    iterations)

% Número de iteraciones predeterminado
if ~exist('iterations','var'); iterations = 500; end;

% Aplicar función de selección
if strcmp(func2str(@max),func2str(selectionFunction))
    objectiveMaximize = true;
    bestFitness = -Inf;
else
    objectiveMaximize = false;
    bestFitness = Inf;
end

bestIndexArray = [];

% Bucle principal
for i = 1:iterations
    indexArray = ones(1, ...
        length(searchSpaceSize)) ...
        +round(rand(1, ...
            length(searchSpaceSize)).*(searchSpaceSize-1));

    currentFitness = fitnessFunction(indexArray);

    if objectiveMaximize && bestFitness<currentFitness
        bestIndexArray = indexArray;
        bestFitness = currentFitness;
    elseif ~objectiveMaximize && bestFitness>currentFitness
        bestIndexArray = indexArray;
        bestFitness = currentFitness;
    end
end

end
```

Código 7.5: Función de optimización `randomSearch` basada en el método de la búsqueda aleatoria

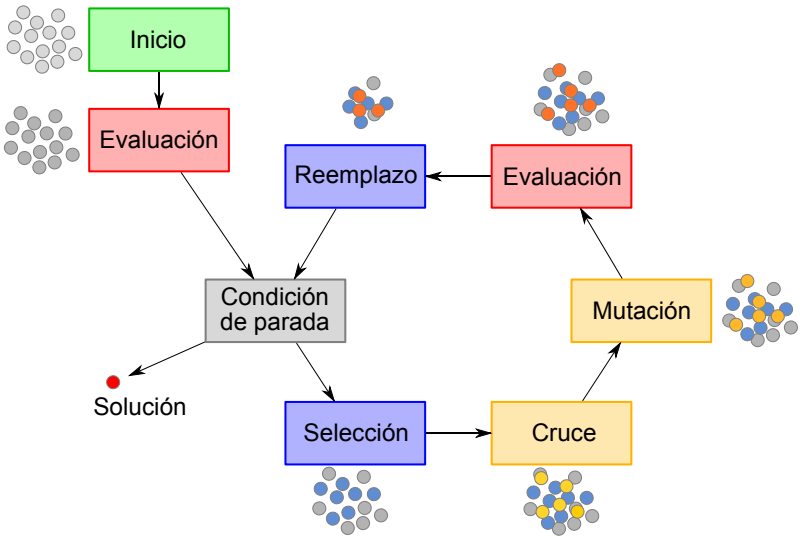

Figura 7.1: Esquema de un algoritmo genético

nueva población. Esta nueva población se utiliza en la siguiente iteración del algoritmo. Por lo general, el algoritmo termina cuando alcanza el número máximo de generaciones o cuando se logra un nivel de aptitud satisfactorio en la población.

La representación habitual de las soluciones se hace con cadenas de bits. También es posible utilizar cadenas con tipos y estructuras diferentes de la misma manera. La característica principal que hace convenientes a estas representaciones genéticas es que sus partes se pueden alinear con facilidad debido a su tamaño fijo, lo cual facilita el cruce entre individuos.

La función de fitness se define sobre la representación genética y mide la calidad de la solución representada. Esta función siempre es dependiente del problema. En el caso de este trabajo, la función de fitness se genera de forma automática con la función de evaluación y el espacio de búsqueda. De esta manera, el usuario se limita a crear funciones de evaluación en términos del problema y no se ve expuesto a la situación de entender cómo funcionan los métodos de optimización o qué codificación es preciso utilizar.

Para entender en profundidad los algoritmos evolutivos es necesario comprender todas las fases a las que se someten los individuos de la población. En la figura 7.1 aparece la representación del esquema de funcionamiento de un algoritmo genético. Las fases de que consta son las siguientes:

Inicio Al comienzo del algoritmo, los individuos de la población se generan de forma aleatoria. El tamaño de la población depende de la naturaleza del problema: a problemas más duros y con más óptimos locales, mayor es el tamaño de población que se suele utilizar. Lo habitual es generar la población aleatoriamente para así cubrir el mayor rango posible de soluciones en el espacio de búsqueda, no obstante, los individuos también pueden ser generados en zonas donde es probable que se encuentren las soluciones óptimas. Para este segundo caso es habitual utilizar algoritmos ávidos o soluciones almacenadas de ejecuciones anteriores.

Evaluación A cada uno de los individuos de la población se le aplica la función de aptitud para cuantificar la bondad de la solución que codifica. Este es, por lo general, el proceso más costoso en términos computacionales para aquellas metaheurísticas basadas en poblaciones.

Selección En cada generación, una proporción de la población vigente se selecciona para crear a la nueva población. Los individuos son elegidos a través de un proceso estocástico donde las soluciones más aptas (según establece la función de fitness) tienen mayor probabilidad de ser escogidas. Ciertos métodos de selección eligen preferiblemente a los mejores individuos; otros métodos lo hacen de manera aleatoria, pues este proceso puede llegar a ser costoso y tener una repercusión demasiado negativa en el algoritmo.

Cruce El cruce genera nuevos individuos en la población a partir de los individuos seleccionados. Para crear cada nuevo individuo, un par de "padres" se combinan de acuerdo a las reglas establecidas por el operador de cruce. El "hijo", producto del cruce, comparte muchas de las características de sus "padres". Este proceso de reproducción se repite sobre los individuos seleccionados hasta dar lugar a una cierta cantidad de nuevos individuos. Aunque los métodos de reproducción basados en dos padres son más fieles a la biología, también es posible considerar un número diferente de progenitores.

Mutación La mutación es un proceso que altera la configuración genética de los individuos. Así, se diversifica la población para evitar la convergencia prematura[1] y se alcanzan zonas del espacio de búsqueda posiblemente no exploradas. Consiste en realizar variaciones en los genes de un individuo. Puede ser más o menos agresiva en virtud del número de genes modificados y la cuantía de la modificación.

Reemplazo El número de individuos de la población se mantiene constante a lo largo de las generaciones. Después del cruce, se producen nuevos individuos que compiten con los de la población actual para formar parte de la nueva población. El reemplazo es una fase donde se determina de qué manera los nuevos individuos y los de la población actual se seleccionan para configurar la nueva población. Entre las opciones más comunes figuran la sustitución de la población al completo o la selección de los mejores individuos de entre los descendientes y la población actual.

Condición de parada El criterio para detener la ejecución y devolver el resultado se establece en la condición de parada. Tal condición se evalúa en cada paso generacional, donde el algoritmo decide si ha de proseguir con el ciclo o si por el contrario lo detiene. Las condiciones habituales para detener la computación son el número de pasos generacionales, el tiempo de ejecución, la convergencia de la población o un nivel de aptitud satisfactorio en la población.

[1] El término convergencia prematura se refiere a que la población ha convergido demasiado pronto, siendo la consecuencia un resultado subóptimo.

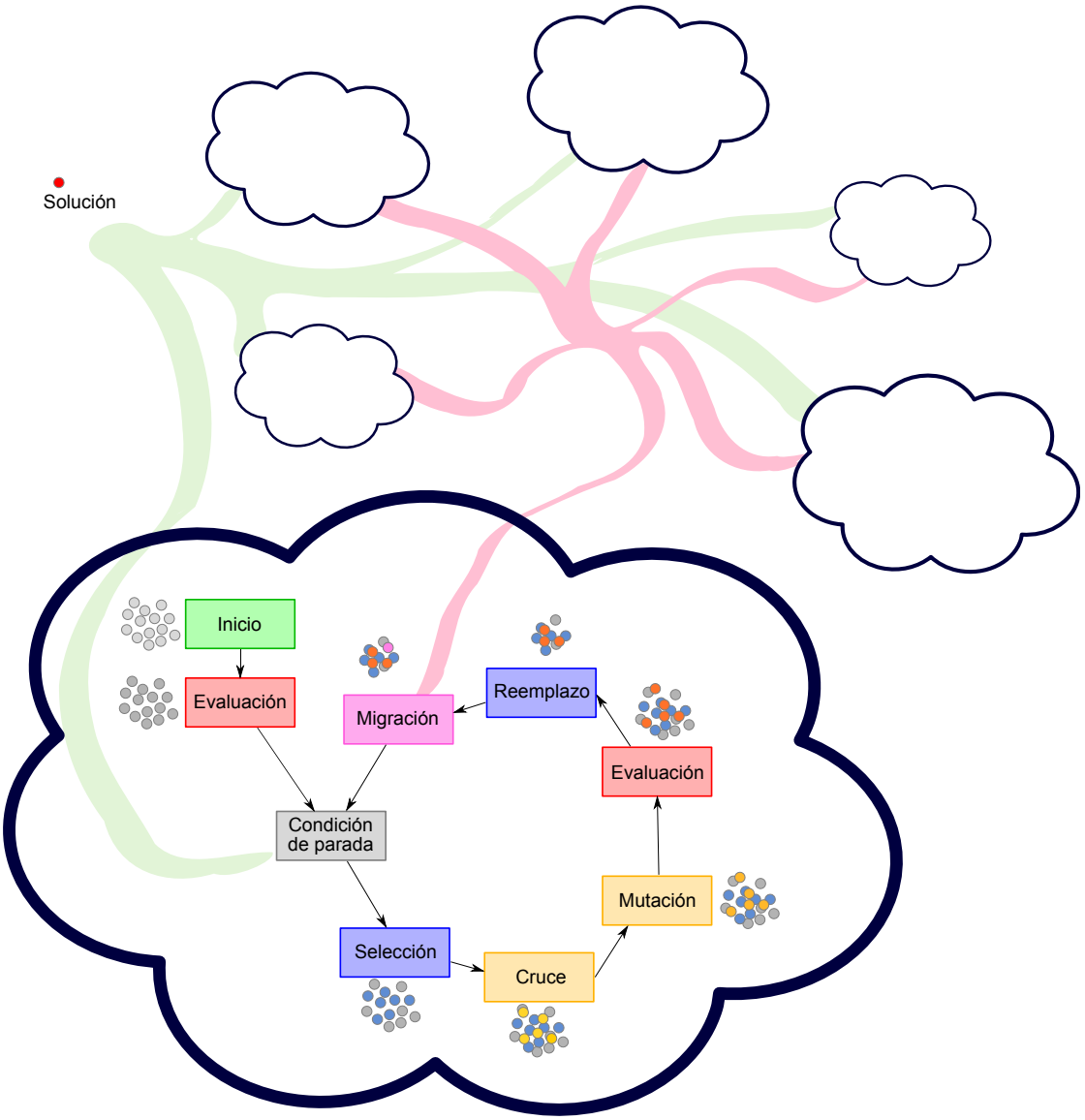

Figura 7.2: Esquema de un algoritmo genético distribuido en islas

Los algoritmos genéticos, al igual que tantas otras metaheurísticas, permiten aprovechar el paralelismo del hardware de forma natural: a esta característica se la conoce como *paralelismo implícito*. Para ello, basta con lanzar un algoritmo genético por unidad de procesamiento. A partir de ahí, el paralelismo puede aprovecharse de dos maneras: ejecuciones independientes que compiten entre sí para alcanzar el mejor resultado o ejecuciones dependientes que colaboran compartiendo individuos en el proceso de optimización. En el segundo caso, hay que establecer un fase de migración en el algoritmo original. Al algoritmo resultante se lo conoce como algoritmo genético distribuido en islas, cuyo esquema de funcionamiento aparece en la figura 7.2.

Migración La migración es el proceso por el cual diferentes instancias del algoritmo genético intercambian individuos, para resolver el problema de forma colaborativa. El proceso

migratorio se lanza cuando en una población el mejor individuo no se ve superado a lo largo de los pasos generacionales. En este caso, dicha población recibe a un individuo de otra población, con la esperanza de que la variabilidad genética que trae consigo permita generar individuos más óptimos.

La función `geneticAlgorithm`

La función `geneticAlgorithm` implementa el algoritmo genético distribuido en islas. El algoritmo se sustenta sobre las primitivas de la *Parallel Computing Toolbox*. El número de islas que se crean es variable y depende del número de sesiones MATLAB para computación en paralelo[2]. Las características de este algoritmo, respecto a las fases del algoritmo genético son las siguientes:

- En el inicio se crean los individuos de forma aleatoria. Esto, trasladado al contexto de las máquinas de trading, significa que se crean máquinas donde cada parámetro a optimizar toma un valor cualquiera del conjunto de valores admisibles. El número de poblaciones y el de individuos por población se fijan con los parámetros `nPopulations` y `nIndividuals`, cuyos valores por defecto aparecen en la tabla 7.3.
- Durante la fase de evaluación, los individuos (que se refieren a máquinas de trading con parámetros concretos) miden su nivel de aptitud con la función de fitness. Tal función la genera de manera automática el método `optimize` a partir de una función de evaluación.
- El proceso de selección de individuos para el cruce no tiene en cuenta la aptitud: todos los individuos tienen la misma probabilidad de ser seleccionados. De esta manera, se propicia la variabilidad genética y la exploración del espacio de búsqueda.
- El cruce de dos individuos se hace escogiendo con igual probabilidad un gen de un progenitor o del otro. Por término medio, un "hijo" hereda la mitad de sus características de cada progenitor. La proporción de descencientes respecto a la población que se generan en cada paso lo regula el argumento `descendantsRatio`, con el valor por defecto que muestra la tabla 7.3.
- La mutación consiste en la variación de los genes del individuo de acuerdo a una probabilidad: cada gen del individuo tiene cierta probabilidad de ser cambiado en esta fase.
- Al final de cada ciclo generacional, la nueva población se compone a partir de los mejores individuos, sean estos nuevos descendientes o de la población actual. Así, el reemplazo consiste en la selección elitista de los mejores individuos.

[2]MATLAB permite crear sesiones para realizar computaciones en paralelo. Una sesión es una entidad con capacidades de computación. De esta manera, se pueden crear tantas sesiones locales como unidades de procesamiento tenga la máquina, e incluso más. Una vez abiertas las sesiones, las funciones programadas con las primitivas paralelas de la *Parallel Computing Toolbox* experimentan un incremento de la velocidad, al utilizar más recursos hardware. Las sesiones MATLAB se abren y cierran con el comando `matlabpool`.

Descripción	Valor
`nPopulations`	máx(1, *sesiones interactivas MATLAB*)
`nIndividuals`	40
`descendantsRatio`	0,5
`timeTrigger`	60
`iterationTrigger`	∞
`convergenceTrigger`	20
`migrationTrigger`	5
P(*selección de individuo para cruce*)	1/*número de individuos*
P(*heredar gen de progenitor*$_1$)	0,5
P(*heredar gen de progenitor*$_2$)	0,5
P(*mutación en gen*)	0,5

Tabla 7.3: Valores por defecto del algoritmo genético distribuido

- Como condición de parada para el algoritmo se contemplan diferentes opciones: tiempo máximo de ejecución, cantidad límite de pasos generacionales y número de pasos generacionales sin mejoras en la población. Se puede utilizar cualquier combinación de estos criterios. Los argumentos de la función para fijar estos valores son `timeTrigger`, `iterationTrigger` y `convergenceTrigger`, con los valores por defecto que aparecen en la tabla 7.3.

- Para la fase de migración, cada instancia del algoritmo cuya población esté estancada tras una cantidad de pasos generacionales recibe el mejor individuo de otra población cualquiera. Este número de pasos sin mejora, que inicia el proceso de migración, se establece con el argumento `migrationTrigger`, cuyo valor por defecto se muestra en la tabla 7.3.

La función que implementa el algoritmo admite una serie de argumentos con los que establece las características y condiciones de la ejecución. Los valores por defecto que toman estos argumentos aparecen recogidos en la tabla 7.3. Por su parte, la signatura de la función es como sigue:

```
bestIndexArray = geneticAlgorithm(selectionFunction, ...
                     fitnessFunction, searchSpaceSize, ...
                     nPopulations, nIndividuals, descendantsRatio, ...
                     timeTrigger, iterationTrigger, ...
                     convergenceTrigger, migrationTrigger)
```

Capítulo 8

Generación de resultados

Una cuestión fundamental en la plataforma consiste en la generación de resultados. El sistema de generación de resultados comprende a todas las propiedades, métodos y funciones que ofrecen información relevante sobre las máquinas de trading. Estos datos proporcionan perspectivas complementarias sobre las máquinas para dar al usuario una vista completa.

La necesidad de contar con información relevante y precisa es máxima. La creación de máquinas de trading efectivas no es un proceso ciego, sino que se sustenta sobre un sólido estudio de las características medibles de las máquinas.

En este capítulo, se presenta una recopilación de las características cuantificables de las máquinas así como las funciones y métodos que las miden. Algunas de estas funciones ya fueron introducidas en la sección 7.2, por lo que sólo serán mencionadas.

La redacción del capítulo se divide en dos bloques principales que se refieren a ámbitos complementarios. La sección 8.1 se centra en las máquinas de trading y en toda la información que proporcionan. Por su parte, la sección 8.2 toma una visión más ámplia y explica la comparación relativa de instancias de máquinas del mismo tipo con el mecanismo de optimización.

8.1. Ámbito individual

Una máquina de trading proporciona resultados a partir de sus propiedades y métodos. Con esta información, el usuario puede manipular la máquina para alterar su funcionamiento.

Entre los resultados más importantes sobre una máquina se incluyen: la señal de posición en el mercado, el rendimiento de la máquina en términos de su beneficio/pérdida, la cantidad invertida en cada posición y los resultados conseguidos por posición adoptada.

8.1.1. Señal de posición

La señal de posición almacena las posiciones que la máquina adopta en el mercado para cada instante de tiempo. En los capítulos 5 y 6 se describieron las máquinas de trading y las fórmulas que utilizan para el cálculo de la señal de posición. Esta es el resultado de aplicar la estrategia de trading sobre los precios de la acción en el tiempo. Toda la información obtenida de las máquinas de trading se derivada en un sentido u otro de la señal de posición.

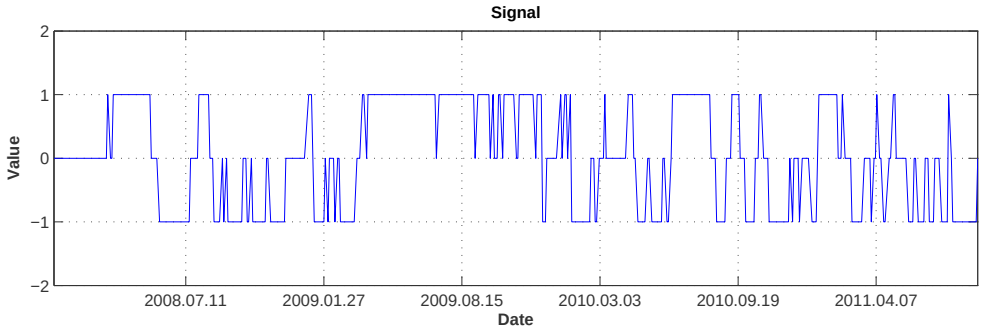

Figura 8.1: Señal de posición de una máquina de trading

El valor de la señal de posición se encuentran alojado en la propiedad `Signal` de la máquina de trading. Para visualizar la señal se utiliza el método `plotSignal`, cuyo resultado se ofrece en la figura 8.1. La curva tiene forma de señal discreta con tres valores diferentes: el uno para indicar posición larga, el menos uno para cortas y el cero para señalar que se permanece fuera del mercado. Acerca de este método, toma como argumento el identificador del intervalo respecto al que hay que dibujar la señal. Al intervalo de entrenamiento corresponde el valor `0`, al de test el `1`, a la unión de ambos intervalos el valor `2` y a todo el conjunto de datos el `3`. Los intervalos de entrenamiento y test no tienen por qué cubrir toda la serie de datos, de ahí la diferenciación entre los dos últimos casos.

8.1.2. Beneficio/pérdida

El beneficio/pérdida es la principal medida del éxito o fracaso de una máquina de trading. Este resultado se refiere al logro de la máquina desde el punto de vista monetario tras invertir según la señal de posición. La escala en que se representa esta medida otorga el valor de la unidad a la inversión inicial: de esta manera, si la variable toma por ejemplo el valor de dos, entonces la máquina de trading logra doblar la inversión inicial; sin embargo, si este valor es un medio, entonces la máquina pierde la mitad de la inversión.

Las propiedades y métodos de la máquina de trading que ofrecen información de esta característica son los siguientes siguientes:

`ProfitLossTrainingSet` Propiedad dependiente de sólo lectura que indica el beneficio/pérdida de la máquina de trading durante el intervalo de entrenamiento. El mecanismo de optimización examina esta propiedad durante la búsqueda de los valores de los parámetros que más mejoran el resultado de la máquina.

`ProfitLossTestSet` Propiedad dependiente de sólo lectura que señala el beneficio/pérdida de la máquina durante el intervalo de test. Esta propiedad muestra el beneficio/pérdida potencial de la máquina en el mercado. El proceso de optimización desconoce el comportamiento del precio de la acción en el intervalo de test, de manera que este valor no está falseado en tanto los intervalos de entrenamiento y test no estén solapados.

`profitLoss` Método que computa el beneficio/pérdida de la máquina en los diferentes intervalos. Como sucede con el método `plotSignal`, este método toma un argumento con el que discrimina entre intervalos. Los valores de las propiedades `ProfitLossTrainingSet` y `ProfitLossTestSet` se calculan con este método.

`profitLossSerie` Método que computa la curva del beneficio/pérdida de acuerdo a la escala temporal de la serie de datos. A diferencia del método `profitLoss`, este otro no se limita al valor final, sino que computa la evolución del beneficio/pérdida a lo largo del tiempo. Este método admite como argumento el identificador del intervalo respecto al que realizar los cálculos.

`plotProfitLoss` Método que dibuja la curva del beneficio/pérdida generada por el método `profitLossSerie`. Toma un argumento con el identificador del intervalo del que se desea realizar la gráfica. Un ejemplo de esta representación aparece en la figura 8.2, donde cada recuadro resalta la evolución del beneficio/pérdida en los diferentes intervalos. En el intervalo de entrenamiento la curva es mucho más pronunciada, consecuencia de la optimización de los valores de los parámetros.

8.1.3. Inversión por posición

La inversión por posición se refiere a la cantidad de capital que la máquina de trading invierte en el mercado de acciones en cada una de las posiciones tomadas. Este valor se relaciona con la señal de posición, los fondos iniciales y la inversión máxima permitida: la señal de posición fija los instantes en que las posiciones se abren y cierran, los fondos iniciales se refieren al capital inicial con que cuenta la máquina para invertir y la inversión máxima permitida consiste en el límite de capital que se puede invertir por posición.

El método `positionSerie` calcula el capital invertido en el mercado por la máquina de trading a lo largo del tiempo. Esta información se representa de forma gráfica, como muestra la figura 8.3, con el método `plotPosition`. En ambos casos, los métodos admiten un argumento con el identificador del intervalo al que se restringen.

8.1.4. Resultado por posición

Para conseguir información detallada del funcionamiento de la máquina en el mercado de acciones está la tabla de registros de posición. En ella aparece un listado de todas las posiciones que se adoptan a lo largo del tiempo. Cabe recordar que las columnas de la tabla fueron descritas en la sección 3.3, en la parte donde se explica cómo incorporar funciones de evaluación. Más aún, se mostró un ejemplo de esta estructura en la tabla 8.1. Esta tabla es la que se pasa a las funciones de evaluación desarrolladas en la sección 7.2.

A modo de recapitulación, los datos que se almacenan para cada posición tomada en el mercado son los siguientes: el tipo de posición (que puede ser larga o corta), los indices de apertura y cierre, las fechas de apertura y cierre, los precios de apertura y cierre, y el

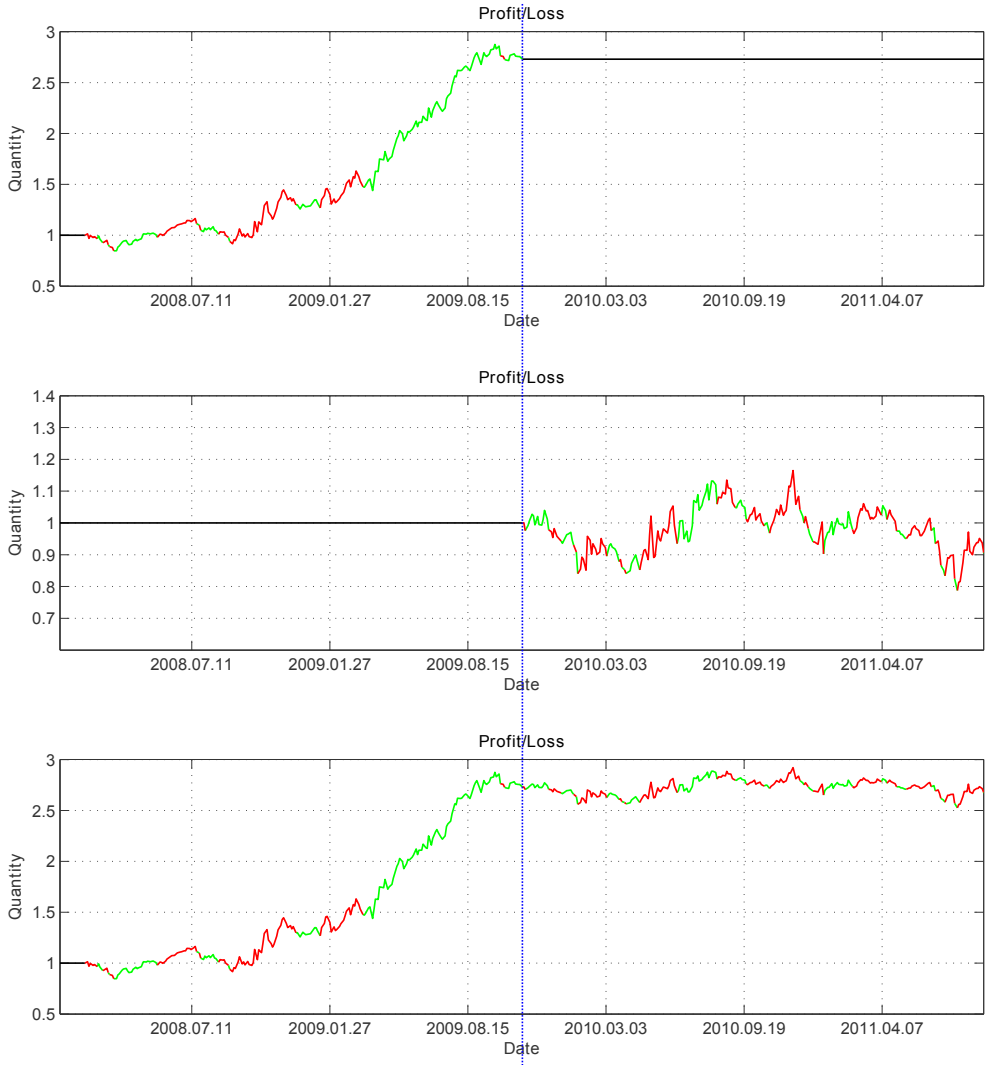

Figura 8.2: Curva de beneficio/pérdida durante el intervalo de entrenamiento (recuadro superior), el intervalo de test (recuadro intermedio) y la unión de ambos (recuadro inferior)

beneficio/pérdida asociado a la operación. Todos ellos aparecen como columnas en la tabla de registros de posición.

Las columnas de la tabla se computan con el método `positionsLog`. Por su parte, el método `printPositionsLog` muestra por pantalla la información de estos registros, proporcionando un resultado similar al que aparece en la tabla 8.1. Estos métodos aceptan un argumento de entrada con el que seleccionar el intervalo al que se limitan.

8.1.5. Funciones de evaluación

Las funciones de evaluación sirven para medir una característica en particular de la máquina de trading. Estas funciones fueron explicadas en la sección 7.2. A partir de ellas, se provee al usuario de un mecanismo para introducir nuevos criterios de medición.

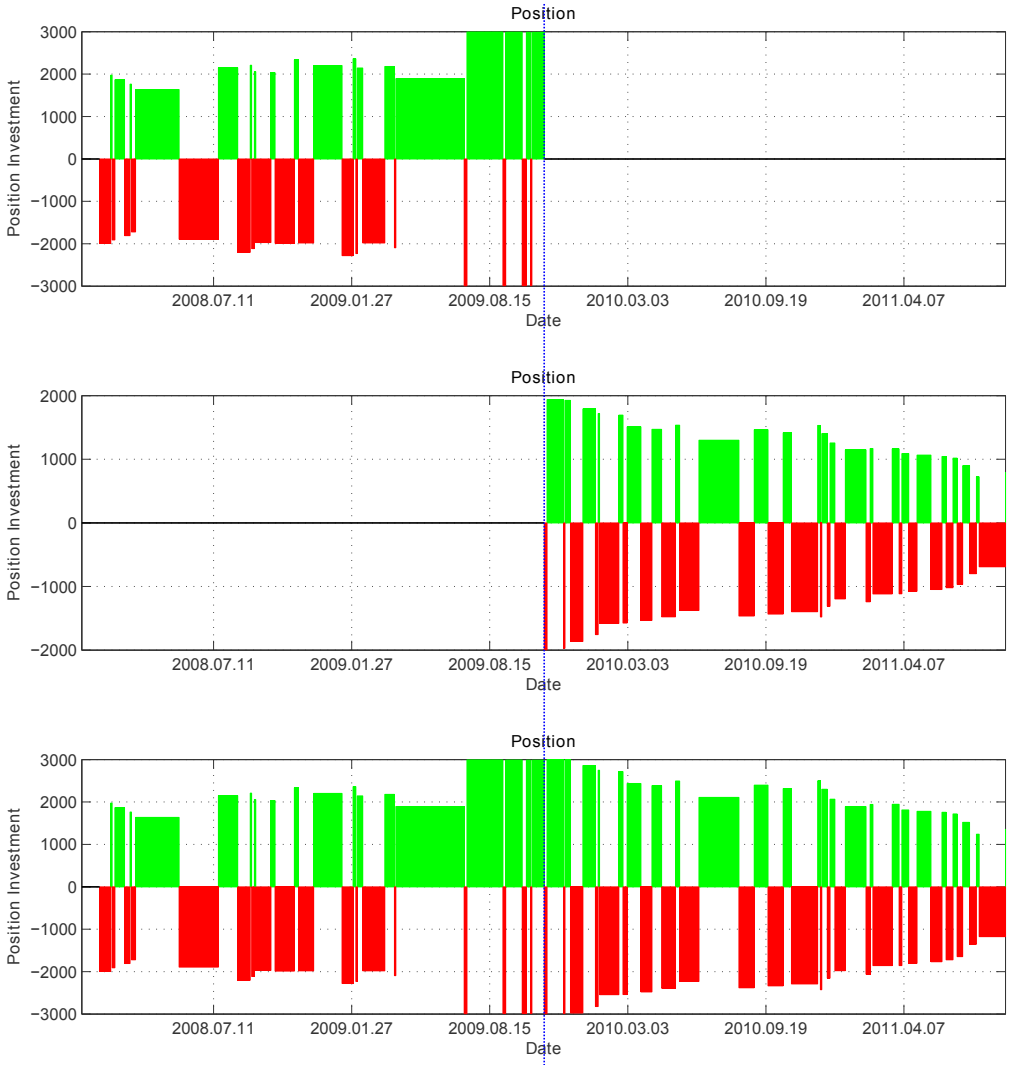

Figura 8.3: Volumen de capital invertido por posición en el mercado durante el intervalo de entrenamiento (recuadro superior), el intervalo de test (recuadro intermedio) y la unión de ambos (recuadro inferior)

8.2. Ámbito colectivo

La aportación del mecanismo de optimización es notable en cuanto a la generación de resultados. Este hace que se adopte un enfoque más ámplio, donde múltiples instancias de máquinas de un mismo tipo se comparan entre sí. Los resultados así obtenidos no se refieren a la instancia de una máquina en particular, sino a un agregado de ellas. Estos resultados sobre conjuntos de instancias de una misma máquina de trading permiten inferir de forma más certera las propiedades y características generales del tipo de máquina.

Los resultados más destacables que se consiguen son la representación del espacio de búsqueda y el cálculo de estadísticos. Estos procesos computan los resultados de un conjunto de instancias de máquinas y organizan los resultados de acuerdo a sus objetivos. Aquella característica que miden de las instancias depende de la función de evaluación empleada.

+1	[	11	16	]	[	2008.01.15	2008.01.22	]	[	9.44	8.6	]	0.91063
+1	[	17	18	]	[	2008.01.23	2008.01.24	]	[	8.19	8.72	]	1.06411
-1	[	63	71	]	[	2008.03.27	2008.04.08	]	[	9.37	9.76	]	0.95791
-1	[	91	95	]	[	2008.05.07	2008.05.13	]	[	10.57	10.5	]	1.00608
+1	[	104	108	]	[	2008.05.26	2008.05.30	]	[	9.88	10.03	]	1.01463
+1	[	109	110	]	[	2008.06.02	2008.06.03	]	[	9.81	9.8	]	0.99845
-1	[	149	151	]	[	2008.07.29	2008.07.31	]	[	9.2	9.35	]	0.98319
+1	[	213	215	]	[	2008.10.27	2008.10.29	]	[	5.2	5.8	]	1.11473
+1	[	228	234	]	[	2008.11.17	2008.11.25	]	[	4.68	4.5	]	0.96245
+1	[	267	274	]	[	2009.01.16	2009.01.27	]	[	4.8	4.52	]	0.94120
+1	[	289	296	]	[	2009.02.17	2009.02.26	]	[	4.11	4.12	]	1.00190
-1	[	311	317	]	[	2009.03.19	2009.03.27	]	[	4.17	4.28	]	0.97311
-1	[	327	331	]	[	2009.04.14	2009.04.20	]	[	5.35	5.25	]	1.01813
-1	[	342	343	]	[	2009.05.06	2009.05.07	]	[	6.19	6.11	]	1.01238
-1	[	344	347	]	[	2009.05.08	2009.05.13	]	[	6.4	5.95	]	1.06974
-1	[	397	407	]	[	2009.07.22	2009.08.05	]	[	8	8.7	]	0.91292
-1	[	425	426	]	[	2009.08.31	2009.09.01	]	[	9.47	9.26	]	1.02160
-1	[	481	483	]	[	2009.11.17	2009.11.19	]	[	10.44	10.35	]	1.00808

Tabla 8.1: Representación de la tabla de registros de posición

8.2.1. Espacio de búsqueda

En optimización, el espacio de búsqueda se refiere al dominio de la función a ser optimizada. En el caso de los algoritmos de búsqueda, que manejan espacios discretos, se refiere al conjunto de todas las posibles soluciones candidatas a un problema.

En la figura 8.4 se muestra el aspecto de un espacio de búsqueda para una función de dos variables. Esta consiste en la aplicación de la función de evaluación del beneficio/pérdida sobre la máquina de trading del cruce de medias móviles. Las dos variables que se enfrentan en el gráfico se refieren a las propiedades `Lead` y `Lag` de la máquina.

Muchas conclusiones se pueden hacer con la información que proporciona el espacio de búsqueda. La utilidad de una herramienta tan potente como esta es múltiple:

- Muestra la distribución de los valores de los parámetros que optimizan el desempeño de la máquina: si estos están formando grupos, entonces los parámetros de la máquina admiten cierto grado de tolerancia en la elección unos valores que sean adecuados; de lo contrario, se hace patente el riesgo de utilizar los valores óptimos para los parámetros, pues ofrecen poca fiabilidad en los resultados que lograrán ante nuevos datos de entrada. De esta manera se puede comprender el riesgo que entraña la utilización de la máquina de trading. Se puede advertir este contraste entre las figuras 8.5 y 8.6.

- Manifiesta la relación existente entre los valores que más mejoran el rendimiento de la máquina: en ocasiones, los mejores valores de varios parámetros están relacionados entre sí de alguna manera. Comprender estas relaciones es esencial para estudiar las características de la máquina de trading. Tal situación se produce entre las propiedades `Lead` y `Lag` de la máquina `MovingAveragesCrossing`, como se muestra en la figura 8.5.

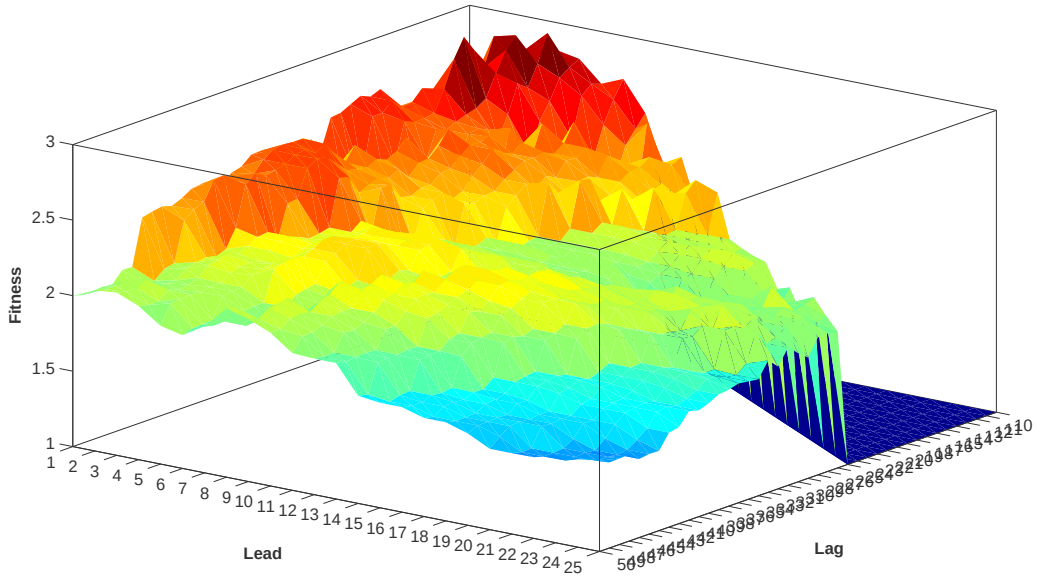

Figura 8.4: Espacio de búsqueda para una función de dos dimensiones

- Resalta la forma del espacio de búsqueda, que es consecuencia de la función de evaluación. Así, se pueden comparar las funciones de evaluación en virtud de las características de los espacios que generan: continuidad, diferencia entre los valores máximos y mínimos, óptimos locales... Este hecho se evidencia al contrastar los espacios de búsqueda obtenidos con respecto a la máquina `MovingAveragesCrossing`: en la figura 8.5 aparece el espacio consecuencia de la función de evaluación `profitLoss`, en la figura 8.7 el de la función `minProfitLossExpected` y en la figura 8.8 el de la función `loss`.

Los métodos responsables de dibujar el espacio de búsqueda son dos: `plotSearchSpace` y `plotSearchSpace123`. El primero representa espacios de cualquier dimensión descomponiendolos en múltiples subespacios de dos dimensiones; el segundo representa espacios de una, dos y hasta tres dimensiones, haciendo corresponder las dimensiones del espacio de búsqueda con las dimensiones espaciales de los gráficos. Un mismo espacio de búsqueda aparece representado por el método `plotSearchSpace123` en la figura 8.9 y por el método `plotSearchSpace` en la figura 8.10. La función de evaluación se incluye como primer argumento de estos métodos, antes del nombre de las propiedades a analizar en el dominio de valores introducido. Por ejemplo, el comando que genera la representación de la figura 8.8 a partir de `mac`, que es una instancia de `MovingAveragesCrossing`, es el siguiente:

```
mac.plotSearchSpace(@loss, ...
        'Lead',1:25, ...
        'Lag',10:50);
```

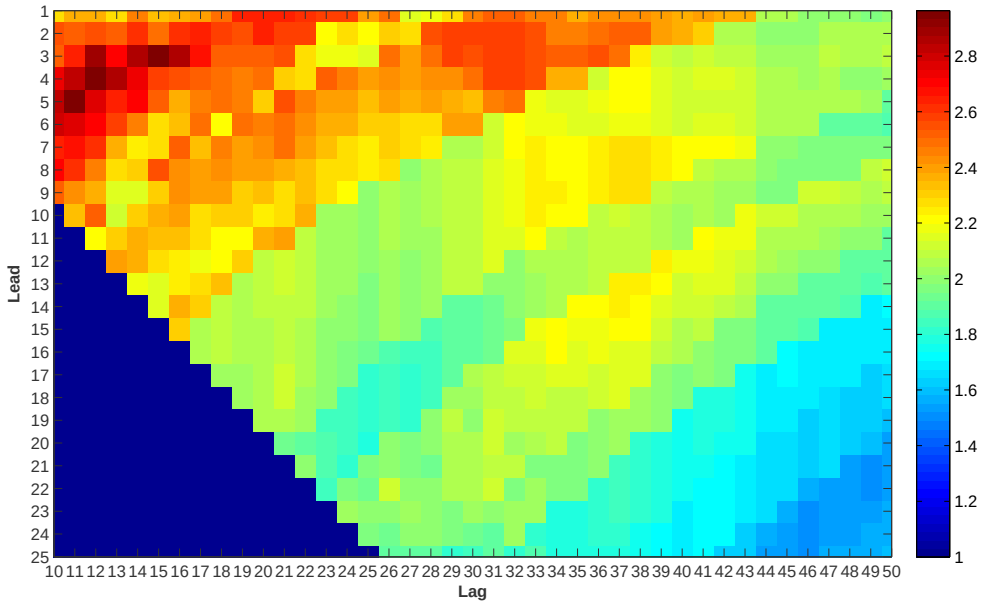

Figura 8.5: Espacio de búsqueda originado a partir de las propiedades `Lead` y `Lag` de la máquina `MovingAveragesCrossing` y la función de evaluación `profitLoss`

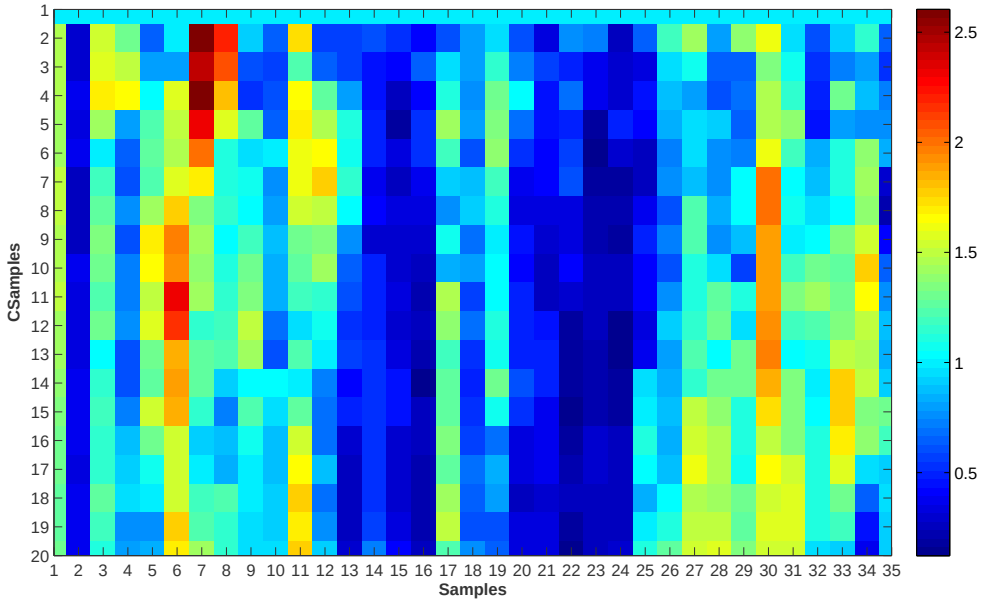

Figura 8.6: Espacio de búsqueda originado a partir de las propiedades `Samples` y `CSamples` de la máquina `RelativeStrengthIndexCrossing` y la función de evaluación `profitLoss`

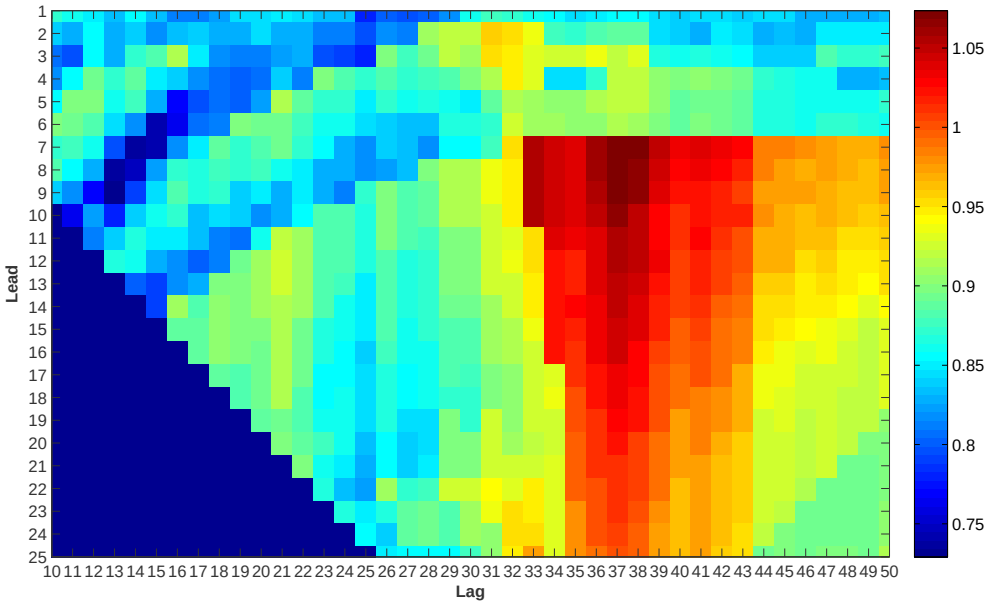

Figura 8.7: Espacio de búsqueda originado a partir de las propiedades `Lead` y `Lag` de la máquina `MovingAveragesCrossing` y la función de evaluación `minProfitLossExpected`

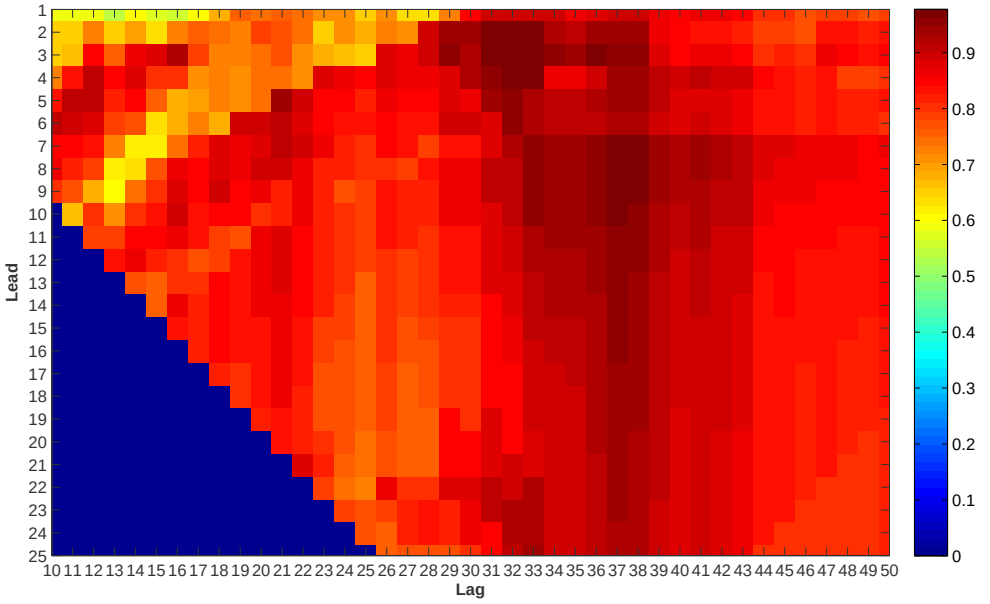

Figura 8.8: Espacio de búsqueda originado a partir de las propiedades `Lead` y `Lag` de la máquina `MovingAveragesCrossing` y la función de evaluación `loss`

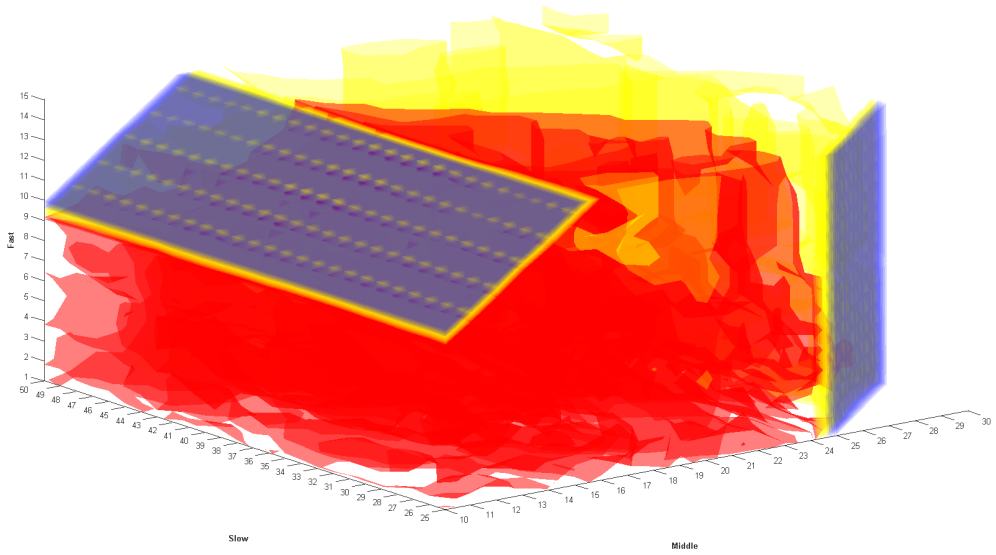

Figura 8.9: Espacio de búsqueda originado a partir de las propiedades `Fast`, `Middle` y `Slow` de la máquina `ThreeMovingAverages`, utilizando el método `plotSearchSpace123` y la función de evaluación `profitLoss`

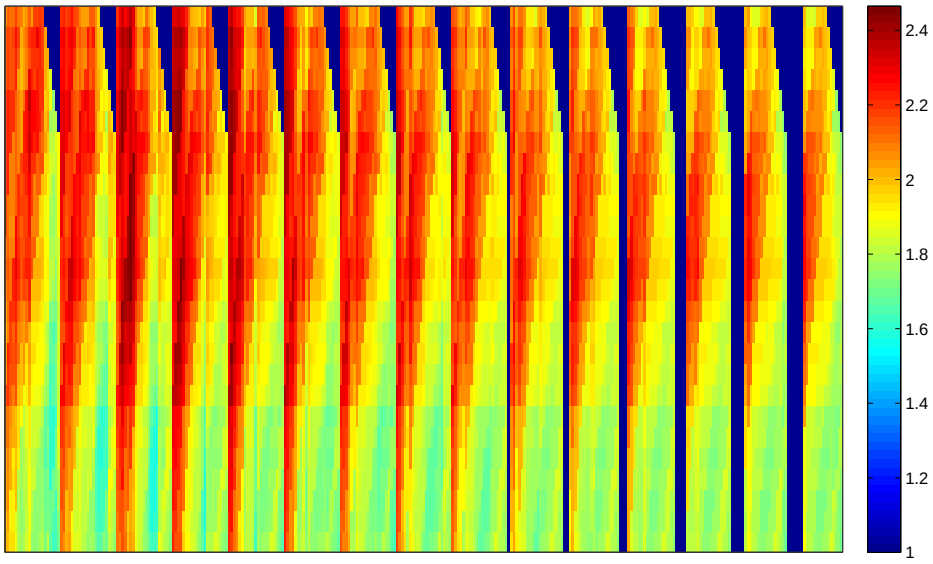

Figura 8.10: Espacio de búsqueda originado a partir de las propiedades `Fast`, `Middle` y `Slow` de la máquina `ThreeMovingAverages`, utilizando el método `plotSearchSpace` y la función de evaluación `profitLoss`

8.2.2. Estadísticos

La información que contiene el espacio de búsqueda consiste en las mediciones realizadas por la función de evaluación sobre el conjunto de instancias de máquinas de trading. Todos estos resultados se pueden sintetizar en una serie de valores representativos: el mínimo, el máximo, la media y la desviación típica. Con ellos, el usuario puede decidir si utiliza una máquina de trading o no. En particular, si la función de evaluación aplicada en el cálculo es la del beneficio/pérdida, entonces el valor mínimo, el máximo y la media sirven para estimar los casos peor, mejor y promedio.

El gráfico que aparece en la figura 8.9 es una representación gráfica del espacio de búsqueda originado por la función de evaluación `ProfitLoss` y la máquina `ThreeMovingAverages` al asignar diferentes valores a sus propiedades `Fast`, `Middle` y `Slow`. En este gráfico el color rojo se refiere a configuraciones adecuadas para la máquina de trading, el amarillo a configuraciones de rendimiento promedio y el azúl a configuraciones deficitarias.

Es difícil tomar decisiones ante tan compleja representación, así que estos datos se reorganizan en forma de histograma como el de la figura 8.11. La información así provista es sencilla de entender e interpretar y en el caso del ejemplo, sugiere que la máquina de trading no pierde dinero con independencia de la asignación de valores que se haga a las propiedades `Fast`, `Middle` y `Slow`, pues el valor de la función de evaluación nunca es inferior a la unidad. En la figura 8.12 se observa que la situación es diferente, y aquí la máquina pierde dinero para la mayoría de configuraciones que puede tomar.

El método `fitnessStatistics` genera histogramas como el de la figura 8.11. Como argumento de entrada, este método admite una función de evaluación. En cuanto a los argumentos de salida, devuelve los valores mínimo, máximo, promedio y desviación típica del espacio de búsqueda. El formato de una llamada a este método es similar a la de los métodos para dibujar el espacio de búsqueda.

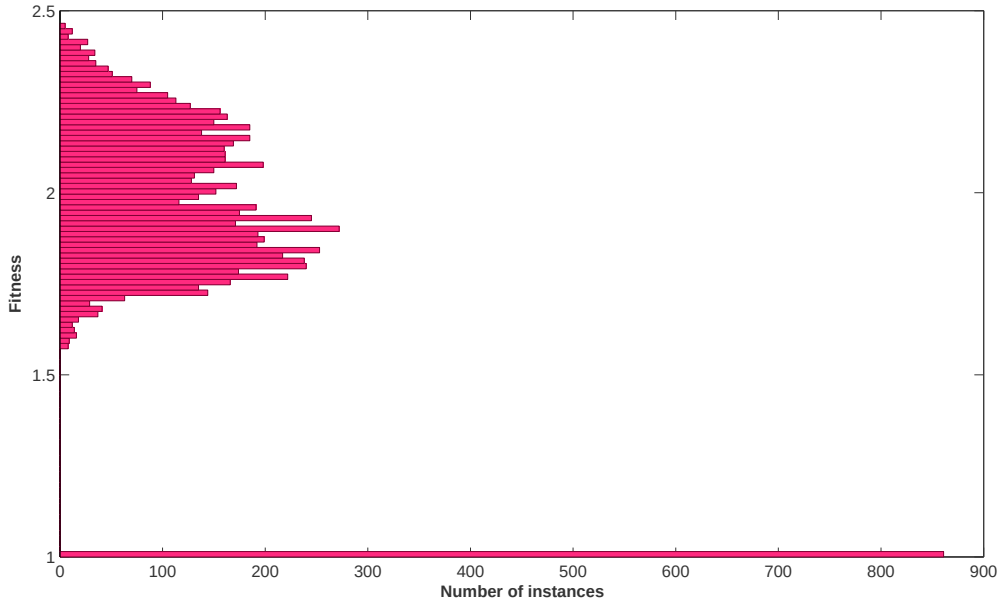

Figura 8.11: Histograma con el número de instancias de máquinas de trading agrupadas de acuerdo a su desempeño: este gráfico es consecuencia de aplicar el método `fitnessStatistics` en una máquina del tipo `ThreeMovingAverages` para evaluar el resultado al asignar diferentes valores a sus propiedades `Fast`, `Middle` y `Slow`

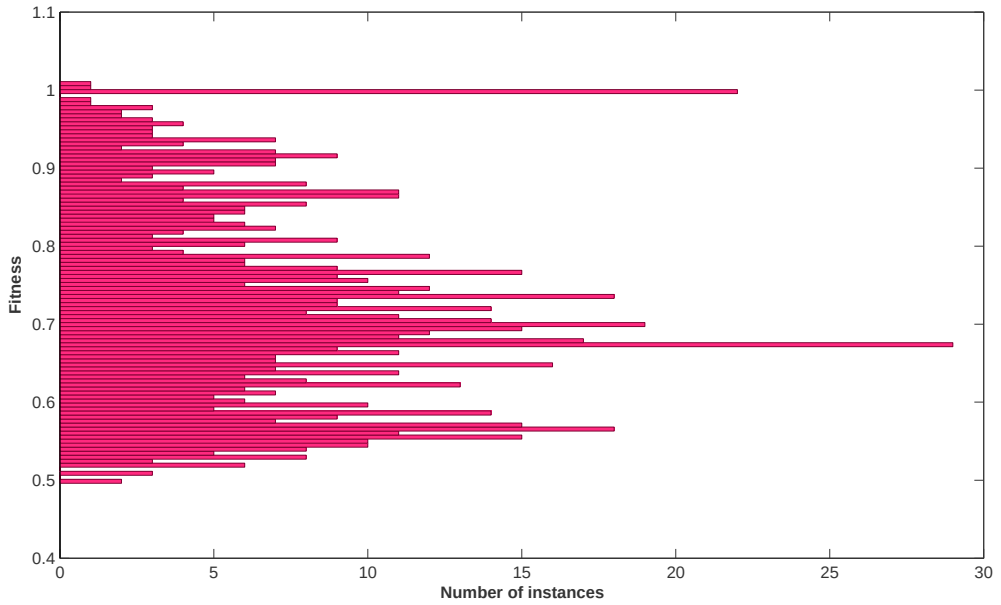

Figura 8.12: Histograma con el número de instancias de máquinas de trading agrupadas de acuerdo a su desempeño: este gráfico es consecuencia de aplicar el método `fitnessStatistics` en una máquina del tipo `MovingAveragesCrossing` para evaluar el resultado al asignar diferentes valores a sus propiedades `Lead` y `Lag`

Capítulo 9

Conclusiones y trabajo futuro

En este capítulo se recogen las conclusiones generales y se mencionan las líneas de trabajo más provechosas con las que extender la plataforma de inversión bursátil.

9.1. Conclusiones

Hasta ahora los trabajos de investigación sobre estrategias de trading han partido desde cero, sin utilizar ninguna herramienta. Esto quiere decir que el esfuerzo de los investigadores no se ha centrado en el objetivo de la investigación, sino que se ha repartido debido a la necesidad de implementar soluciones *ad hoc* con las que experimentar con las estrategias. Existe una gran cantidad de procesos en común en todos estos experimentos (adquisición de datos, optimización de parámetros, generación de resultados...), pero no se han dado pasos en el sentido de desarrollar un marco de trabajo estandarizado.

Una situación que sucede al estudiar las estrategias de trading es que el investigador (o el inversor) tiene formación en un área concreta y suele desconocer las herramientas formales de otras áreas, aún cuando estas podrían serle de gran ayuda. Por ejemplo, un economista puede entender el funcionamiento de las estrategias de trading e ignorar los métodos de optimización que servirían para mejorar el desempeño de la estrategia.

Con la plataforma de inversión bursátil se superan todas estas dificultades. Por un lado, la plataforma implementa todos los procesos habituales que toman parte en la experimentación con estrategias de trading, para así servir de punto de partida en los trabajos futuros. Por otro lado, la plataforma ofrece un entorno integral en el que los investigadores (e inversores) disponen de herramientas potentes traídas desde las diferentes áreas para abordar el problema.

Una de las aportaciones más creativas en la plataforma son las estrategias de trading compuestas. Estas se muestran como extensiones para combinar de manera efectiva un conjunto de estrategias con el objetivo de potenciar sus fortalezas y reducir sus debilidades. Con ellas, las posibilidades de generar nuevas estrategias en la plataforma son virtualmente ilimitadas.

Otro elemento de gran notoriedad en la plataforma de inversión bursátil es el mecanismo de optimización de parámetros: esta es una pieza muy valiosa de la plataforma debido a la flexibilidad que exhibe y la potencia de los algoritmos de optimización que implementa. Este

mecanismo permite a un investigador (o inversor) optimizar cualquier estrategia desarrollada con los métodos más sofisticados. En particular, la plataforma incorpora un algoritmo genético distribuido para problemas de optimización duros.

En lo que al análisis de resultados se refiere, la plataforma ofrece la posibilidad de representar los espacios de búsqueda de los problemas asociados y resumir los resultados con los estadísticos. Esta aportación es crucial y permite al investigador (o inversor) entender de forma visual el comportamiento de la estrategia de trading en virtud de los valores de sus parámetros.

El campo de la inversión bursátil está saturado de técnicas de dudosa efectividad y los mercados se encuentran plagados de inversores movidos por no mucho más que sus propios impulsos. Una herramienta como la desarrollada en este proyecto puede utilizarse para arrojar luz y realizar experimentos serios y rigurosos acerca de las inversiones bursátiles.

9.2. Trabajo futuro

Algunas de las líneas de trabajo futuro se relacionan con las contribuciones del proyecto.

Una propuesta puede ser la de implementar nuevas estrategias de trading para la plataforma. En particular, las estrategias de mayor interés para ser incorporadas son aquellas basadas en herramientas propias del campo de la inteligencia artificial, como las redes neuronales, las redes de Bayesianas o el modelo oculto de Markov. Estas técnicas se han mostrado efectivas en el modelado y predicción del precio de las acciones de la bolsa. Igual de interesante puede ser además combinar estos métodos con las estrategias de trading compuestas.

Otra línea de trabajo consiste en idear estrategias de trading compuestas que utilicen a las estrategias de partida de alguna manera más sofisticada. Por ejemplo, se pueden desarrollar estrategias compuestas que ponderen el peso de cada estrategia en la toma de decisiones según su desempeño. Otra idea podría ser combinar un conjunto de máquinas fragmentadas para que en cada fragmento sólo se considere la decisión de aquella máquina que obtuvo el mejor resultado en el intervalo del fragmento anterior.

Una opción también atractiva sería la de emprender un estudio de las estrategias implementadas en la plataforma. La idea aquí consiste en recopilar un ingente de datos para estimar el éxito de algunas estrategias para producir beneficios frente a la incapacidad de otras. Un análisis estadístico exhaustivo arrojaría conclusiones sobre el rendimiento relativo de las estrategias.

La última línea propuesta consistiría en utilizar la plataforma como un generador de instancias de problemas de optimización combinatoria para estudiar los resultados de los algoritmos de optimización.

Para concluir, cabe mencionar que casi cualquier aspecto de la plataforma puede ser extendido en el grado que se desee siguiendo las indicaciones recogidas en el documento.

Apéndice A

La clase `DataSerie`

La clase abstracta `DataSerie` es la encargada de almacenar y representar la información relativa a una acción del mercado bursátil. Ofrece una interfaz común para todas aquellas clases que se encargan de conectar con los diferentes proveedores de información bursátil. Está implementada en el archivo `DataSerie.m` como una clase MATLAB de tipo *handle*[1]. A partir del mecanismo de herencia es como se extiende a esta clase abstracta. Es necesario que la clase heredera implemente el método abstracto `get`, que es el encargado de comunicar con el proveedor de información bursátil concreto. La clase `DataSerie` fue introducida en la sección 3.1 y utilizada en el capítulo 4 para implementar la comunicación con los diferentes proveedores disponibles en la plataforma.

Esta parte del documento contiene la información técnica relativa a la clase `DataSerie`. En la sección A.1 quedan recogidas las propiedades[2] o atributos. Los métodos son documentados en la sección A.2. Finalmente, la sección A.3 comenta el sistema de dibujado de gráficos.

A.1. Propiedades

Las propiedades de la clase `DataSerie` se pueden separar según su naturaleza en no dependientes y dependientes. Las propiedades no dependientes son aquellas que reservan memoria donde almacenar el valor. Las propiedades dependientes, según se definen en MATLAB, no reservan memoria: en caso de lectura, la propiedad dependiente calcula el valor a devolver a partir de una función asociada (es similar a un método sin argumentos que devuelve un valor); para operaciones de escritura, la propiedad dependiente invoca a una segunda función asociada que altera alguna de las propiedades no dependientes del objeto (se asemeja a un método con el nuevo valor de la propiedad como argumento de entrada que modifica el estado interno del objeto).

[1] Una clase de tipo *handle* se define en MATLAB como aquella que se manipula a través de un puntero y no de forma directa. Las diferencias entre una clase normal y una clase de tipo *handle* son sutiles y afectan al comportamiento de los objetos de la clase.

[2] Hay que aclarar que el uso que MATLAB hace del término propiedad se corresponde con el concepto de atributo según se define en los lenguajes de programación orientada a objetos.

A.1.1. Close

La propiedad `Close` se corresponde con un vector columna de tipo *double* que contiene los precios de cierre de la acción. El instante de tiempo correspondiente al elemento i del vector `Close` se encuentra en la posición i del vector `DateTime`. De esta manera, se almacena tanto el precio de cierre como el instante de tiempo de la serie temporal.

A.1.2. CompressionType

La propiedad `CompressionType` almacena en forma de cadena de caracteres el tipo de resolución con que están almacenados los datos en el objeto: puede ser `ticks`, `minutes`, `days`, `weeks` o `months`. Sirve, junto a la propiedad `CompressionUnits`, para entender la magnitud y frecuencia de los datos almacenados en el objeto. Esta propiedad toma el valor que se pasa al método `get` como argumento. De esta manera, las peticiones realizadas a los proveedores pueden proporcionar información con diferente grado de resolución. Por ejemplo, si la compresión temporal es `months` y las unidades de compresión 3, entonces los datos tomados del proveedor son empaquetados en intervalos de tres meses.

A.1.3. CompressionUnits

La propiedad `CompressionUnits` guarda un número de tipo entero que cuantifica la frecuencia con que se encuentran almacenados los datos de la serie. Sirve, junto a la propiedad `CompressionType`, para entender la magnitud y frecuencia de los datos almacenados en el objeto. Esta propiedad toma el valor que se pasa como argumento al método `get`.

A.1.4. DateTime

La propiedad `DateTime` se corresponde con un vector columna de tipo *double* que contiene los instantes de tiempo donde se realizaron las mediciones de los precios de la acción. Se utiliza junto a otros vectores para definir las series temporales relacionadas con las cotizaciones de una acción. Las fechas se codifican en diferentes formatos dentro de MATLAB: como cadena de caracteres con la función `datestr`, como vector con `datevec` o como número con `datenum`. Este vector codifica las fechas como números, donde el valor 1 se corresponde con la fecha 1 de enero del año 0 a las 00:00:00. El incremento de días supone un aumento del valor numérico en una unidad. Por su parte, también se consideran incrementos menores a un dia (horas, minutos y segundos), en cuyo caso se utilizan números fraccionarios. Por ejemplo, el dia 3 de enero del año 0 a las 06:00:00 se codifica como 3.25.

A.1.5. DiffSerie

La propiedad dependiente de sólo lectura `DiffSerie` devuelve un vector columna con las variaciones relativas en tanto por uno que sufren los valores del vector `Serie`. En concreto, el primer elemento toma el valor 0 y el resto se calcula a partir de la división entre dos valores consecutivos del vector `Serie`.

A.1.6. EndDateTime

Esta propiedad dependiente de sólo lectura devuelve la última fecha del vector `DateTime`, que marca el último instante para el que se tiene información sobre los precios de la acción.

A.1.7. High

La propiedad `High` se corresponde con un vector columna de tipo *double* que contiene los precios más altos de la acción. El instante de tiempo correspondiente al elemento i del vector `High` se encuentra en la posición i del vector `DateTime`. De esta manera, se almacena tanto el precio más alto como el instante de tiempo de la serie temporal.

A.1.8. InitDateTime

Propiedad dependiente de sólo lectura que devuelve la primera fecha en el vector `DateTime`. Se refiere al inicio del intervalo para el cual se tiene información de los precios de la acción.

A.1.9. Length

Esta propiedad dependiente de sólo lectura devuelve la longitud de los vectores que representan series temporales almacenadas en el objeto. Las longitudes de todos estos vectores son las mismas.

A.1.10. Low

La propiedad `Low` se corresponde con un vector columna de tipo *double* que contiene los precios mas bajos de la acción. El instante de tiempo correspondiente al elemento i del vector `Low` se encuentra en la posición i del vector `DateTime`. De esta manera, se almacena tanto el precio más bajo como el instante de tiempo de la serie temporal.

A.1.11. Open

La propiedad `Open` se corresponde con un vector columna de tipo *double* que contiene los precios de apertura de la acción. El instante de tiempo correspondiente al elemento i del vector `Open` se encuentra en la posición i del vector `DateTime`. De esta manera, se almacena tanto el precio de apertura como el instante de tiempo de la serie temporal.

A.1.12. Serie

La propiedad dependiente de sólo lectura `Serie` devuelve la propiedad `Close`. Se utiliza como apuntador para desacoplar a la plataforma de inversión de una serie temporal concreta. Las máquinas de inversión suelen realizar los cálculos a partir de la información contenida en la propiedad `Serie`. Se puede hacer que la propiedad `Serie` apunte hacia cualquiera de los vectores: `Open`, `High`, `Low` y `Close`.

A.1.13. SymbolCode

La propiedad `SymbolCode` guarda el identificador de la acción bursátil en forma de cadena de texto. Almacena la misma cadena que utiliza el proveedor de datos para identificar a la acción. Para una misma acción de la bolsa este identificador puede cambiar según el proveedor.

A.1.14. Volume

La propiedad `Volume` se corresponde con un vector columna de tipo *double* que contiene los volúmenes de transacción de la acción. El instante de tiempo correspondiente al elemento i del vector `Volume` se encuentra en la posición i del vector `DateTime`. De esta manera, se almacena tanto el volumen de transacción como el instante de tiempo de la serie temporal.

A.2. Métodos

Es necesario comentar la manera en que MATLAB permite organizar el código para implementar una clase: una forma consiste en crear un único archivo monolítico que contenga toda la definición e implementación de la clase; otra posibilidad es crear un archivo con el nombre de la clase y almacenarlo en un directorio con el mismo nombre que este archivo, aunque sin la extensión y precedido del carácter @. La clase `DataSerie` se organiza de acuerdo a la segunda opción. Utilizando tal posibilidad, se crean los métodos de la clase como archivos independientes contenidos en el directorio `@DataSerie`. En este caso, hay que especificar en el archivo de la clase los atributos del método: estos son las restricciones de acceso (`public`, `private`, `protected`) y sus características (`static`, `abstract`). Los métodos de las clases en MATLAB, a excepción del constructor, tienen como primer argumento la instancia de la clase. Esto es así porque MATLAB permite utilizar los métodos de dos formas diferentes: la primera consiste en emplear el separador '.', colocando delante la instancia de la clase y detrás el método; la segunda posibilidad es usar el método como si fuera una función, donde el primer argumento es la instancia de la clase.

A.2.1. DataSerie

Constructor de la clase `DataSerie`.

Sintaxis

```
dataSerie = DataSerie(symbolCode, compressionType, compressionUnits, ...
                      initDateTime, endDateTime)
```

Argumentos

`symbolCode` Cadena de caracteres que identifica a la acción bursátil según la reconoce el proveedor de datos.

`compressionType` Cadena de caracteres que indica la magnitud de tiempo considerada en las series temporales. Puede tomar los valores `'ticks'`, `'minutes'`, `'days'`, `'weeks'` o `'months'` según estén o no implementados en el proveedor.

`initDateTime` Fecha comienzo del intervalo a partir de la cual se quiere obtener la información sobre las series temporales de la acción bursátil.

`endDateTime` Fecha final del intervalo hasta la cual se quiere obtener la información sobre las series temporales de la acción bursátil.

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

A.2.2. `drawClose`

Dibuja la gráfica del precio de cierre de la acción. Los datos que visualiza son los de la propiedad `Close` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawClose(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.3. `drawDiffSerie`

Dibuja la gráfica del diferencial en tanto por uno de alguno de los precios de la acción (apertura, más alto, más bajo o cierre) según apunte la propiedad `Serie`. Los datos que visualiza por defecto son los del diferencial de la propiedad `Serie` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawDiffSerie(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.4. `drawHigh`

Dibuja la gráfica del precio más alto de la acción. Los datos que visualiza son los de la propiedad `High` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawHigh(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.5. `drawLow`

Dibuja la gráfica del precio más bajo de la acción. Los datos que visualiza son los de la propiedad `Low` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawLow(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.6. drawOpen

Dibuja la gráfica del precio de apertura de la acción. Los datos que visualiza son los de la propiedad `Open` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawOpen(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.7. drawSerie

Dibuja la gráfica de alguno de los precios de la acción (apertura, más alto, más bajo o cierre) según apunte la propiedad `Serie`. Los datos que visualiza por defecto son los de la propiedad `Close` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawSerie(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.8. drawSeries

Dibuja la gráfica con todos los precios de la acción (apertura, más alto, más bajo y cierre). Los datos que visualiza son los de las propiedades `Open`, `High`, `Low` y `Close` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawSeries(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.9. `drawVolume`

Dibuja la gráfica del volumen de transacciones de la acción. Los datos que visualiza son los de la propiedad `Volume` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawVolume(dataSerie, axesHandle, initIndex, endIndex)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`initIndex` Índice del primer valor de las series temporales a partir del cual dibujar las gráficas.

`endIndex` Índice del último valor de las series temporales hasta el cual dibujar las gráficas.

A.2.10. `equals`

Compara dos objetos de clase `DataSerie` e indica si son o no iguales.

Sintaxis

```
boolean = equals(lhs, rhs)
```

Argumentos

`lhs` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rhs` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`boolean` Resultado de la comparación de igualdad.

A.2.11. `firstFrom`

Calcula el índice del primer elemento del vector de instantes temporales `DateTime` que cumpla ser menor o igual que la fecha indicada.

Sintaxis

```
index = firstFrom(dataSerie, dateTime)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`dateTime` Fecha para realizar la búsqueda del índice sobre `DateTime`.

`index` Índice del vector de instantes temporales `DateTime`.

A.2.12. `get`

Obtiene los precios de cotización de la acción desde un proveedor de datos. Es un método abstracto que ha de ser implementado desde la clase que hereda de `DataSerie`. Este método no se utiliza directamente, sino que es invocado por el constructor para obtener los datos del proveedor en el momento de la creación del objeto.

Sintaxis

```
[dateTime, open, high, low, close, volume]
= get(symbolCode,compressionType,compressionUnits,initDateTime,endDateTime)
```

Argumentos

`symbolCode` Cadena de caracteres que identifica a la acción bursátil según el proveedor de datos.

`compressionType` Cadena de caracteres que indica la magnitud de tiempo considerada en las series temporales. Puede tomar los valores `ticks`, `minutes`, `days`, `weeks` o `months` según estén o no implementados en el proveedor.

`initDateTime` Fecha comienzo del intervalo a partir de la cual se quiere obtener la información sobre las series temporales de la acción bursátil.

`endDateTime` Fecha final del intervalo hasta la cual se quiere obtener la información sobre las series temporales de la acción bursátil.

`dateTime` Vector columna con los instantes de tiempo en que se han obtenido los valores de las series temporales.

`open` Vector columna con los precios de apertura de la acción bursátil.

`high` Vector columna con los precios más altos de la acción bursátil.

`low` Vector columna con los precios más bajos de la acción bursátil.

`close` Vector columna con los precios de cierre de la acción bursátil.

`volume` Vector columna con los volúmenes de transacción de la acción bursátil.

A.2.13. `lastUntil`

Calcula el índice del último elemento del vector de instantes temporales `DateTime` que cumpla ser mayor o igual que la fecha indicada.

Sintaxis

```
index = lastUntil(dataSerie, dateTime)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`dateTime` Fecha para realizar la búsqueda del índice sobre `DateTime`.

`index` Índice del vector de instantes temporales `DateTime`.

A.2.14. `plot`

Genera el gráfico con que se representa a la acción bursátil. Crea y configura la ventana, y mediante llamadas sucesivas al método `plotWrapper` produce los gráficos deseados.

Sintaxis

```
fig = plot(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.15. plotClose

Genera el gráfico con que se representa al precio de cierre de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawClose` como argumento.

Sintaxis

```
fig = plotClose(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.16. plotDiffSerie

Genera el gráfico con que se representa al diferencial de la serie predeterminada de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawDiffSerie` como argumento.

Sintaxis

```
fig = plotDiffSerie(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.17. plotHigh

Genera el gráfico con que se representa al precio más alto de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawHigh` como argumento.

Sintaxis

```
fig = plotHigh(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.18. `plotLow`

Genera el gráfico con que se representa al precio más bajo de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawLow` como argumento.

Sintaxis

```
fig = plotLow(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.19. `plotOpen`

Genera el gráfico con que se representa al precio de apertura de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawOpen` como argumento.

Sintaxis

```
fig = plotOpen(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.20. `plotSerie`

Genera el gráfico con que se representa la serie predeterminada de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawSerie` como argumento.

Sintaxis

```
fig = plotSerie(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.21. `plotSeries`

Genera el gráfico con que se representa a todos los precios de la acción (apertura, más alto, más bajo y cierre). Realiza llamadas sucesivas al método `plotWrapper` utilizando como argumento los métodos de dibujado `drawOpen`, `drawHigh`, `drawLow` y `drawClose`.

Sintaxis

```
fig = plotSeries(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.22. plotVolume

Genera el gráfico con que se representa el volumen de transacciones de la acción bursátil. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawVolume` como argumento.

Sintaxis

```
fig = plotVolume(dataSerie, rangeInit, rangeEnd)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

A.2.23. plotWrapper

Inicializa y configura la ventana donde se muestran los gráficos. Se encarga de establecer la manera en que se interactúa con los gráficos a partir de la función de *panning*, que cambia la sección del gráfico mostrado en pantalla ante eventos del ratón, y *zooming*, que aumenta o disminuye la escala en que se visualiza el gráfico. A partir de la llamada al método `plotWrapper`, utilizando como argumento alguno de los métodos de dibujado `draw*`, se consigue la representación de la información de la acción bursátil. Este método es utilizado por los métodos `plot*` para generar los gráficos.

Sintaxis

```
fig = plotWrapper(dataSerie, customizer, axesHandle,
                  rangeInit, rangeEnd, varargin)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`customizer` Método de dibujado para representar la información de la acción bursátil.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`varargin` Resto de argumentos que se pasan sin modificar hacia el método `draw*`. Permite extender el comportamiento del método que dibuja la información.

`fig` Apuntador a la figura generada.

A.3. El sistema de dibujado

El sistema de dibujado está dividido en partes que colaboran en la construcción de los gráficos: de esta manera se consigue un código fácil de mantener, alterar y extender. Este proceso está dividido en dos etapas: primero se crea y configura la ventana contenedor y después se dibujan los gráficos que representan la información. La manera de acometer esta separación es utilizando el método `plotWrapper`, que se encarga de la inicialización y toma como argumento un método `draw*`, cuyo objetivo es dibujar el gráfico. Para ocultar esta complejidad de cara al usuario se utilizan los métodos `plot*`, que generan el gráfico a través de una llamada a `plotWrapper` en combinación con un método de dibujado `draw*`. Por ejemplo, la clase `DataSerie` implementa el método `plotClose`, que crea la ventana con la curva de cotizaciones de cierre de la acción combinando los métodos `plotWrapper` y `drawClose`. La ventaja de esta separación de roles en los métodos es que la implementación del método `plotWrapper` es la más compleja (al lidiar con los aspectos específicos del entorno MATLAB para configurar la ventana), mientras que la implementación de los métodos `plot*` y `draw*` es trivial.

Apéndice B

La clase `AlgoTrader`

Las máquinas de trading heredan de la clase abstracta `AlgoTrader`. Esta clase facilita la programación de máquinas gracias a los métodos que implementa y la interfaz que ofrece. Se encuentra en el archivo `AlgoTrader.m`, siendo una clase MATLAB de tipo *handle*[1]. La máquina de trading que hereda de esta clase tiene que implementar el método abstracto `computeSignal`, que es el que calcula la señal de posiciones de la máquina en el mercado. Esta clase se introdujo en la sección 3.2 y se usó de forma extensiva en los capítulos 5 y 6 para implementar máquinas de trading. En el capítulo 7 se explicaron sus características en relación a la optimización de parámetros. Por último, en el capítulo 8 de describieron las posibilidades de generación de resultados para el estudio de las máquinas de trading.

Aquí se documenta la información técnica de la clase `AlgoTrader`. Primero, en la sección B.1, se enumeran las propiedades. La sección B.2 contiene un listado de los métodos de la clase. Por último, el sistema de dibujado se explica en la sección B.3.

B.1. Propiedades

Las propiedades de la clase `AlgoTrader` se pueden separar según su naturaleza en no dependientes y dependientes. Las propiedades no dependientes son aquellas que reservan memoria donde almacenar el valor. Las propiedades dependientes, según se definen en MATLAB, no reservan memoria: en caso de lectura, la propiedad dependiente calcula el valor a devolver a partir de una función asociada (su comportamiento es similar al de un método sin argumentos que devuelve un valor); para operaciones de escritura, la propiedad dependiente invoca a una segunda función asociada que altera alguna de las propiedades no dependientes del objeto (en este caso se asemeja a un método con el nuevo valor de la propiedad como argumento de entrada que modifica el estado interno del objeto).

[1] Una clase de tipo *handle* se define en MATLAB como aquella que se manipula a través de un puntero y no de forma directa. Las diferencias entre una clase normal y una clase de tipo *handle* son sutiles y afectan al comportamiento de los objetos de la clase.

B.1.1. AllowedPositions

La propiedad `AllowedPositions` indica qué tipo de posiciones se le permite adoptar a la máquina de trading: con valor 0 limita su acción a posiciones largas, con 1 a posiciones cortas y con 2 permite cualquier tipo de posición. El valor de esta propiedad afecta a la señal que genera la máquina, contenida en la propiedad `Signal`.

B.1.2. CurrentFunds

Con la propiedad dependiente de sólo lectura `CurrentFunds` la máquina de trading muestra la cantidad de capital que tiene tras haber operado en el mercado durante el intervalo de test. Este valor se expresa en las mismas unidades que el capital inicial, contenido en la propiedad `InitialFunds`. Si el valor de esta propiedad es mayor que el de la propiedad `InitialFunds`, entonces la máquina ha generado beneficios; si es menor, entonces ha incurrido en pérdidas.

B.1.3. DataSerie

La propiedad `DataSerie` contiene una instancia de la clase `DataSerie`. Es de esta instancia de donde se obtiene la información de las series de precios de la acción para realizar las computaciones. El valor de esta propiedad se fija en el instante de creación de la máquina y no se puede alterar durante el tiempo de vida del objeto.

B.1.4. InitialFunds

La cantidad de fondos de que dispone una máquina se indica con `InitialFunds`: representa el capital que utiliza para operar en el mercado bursátil. Si se modifica esta cantidad, entonces se vuelven a calcular los beneficios y pérdidas consecuencia de las posiciones tomadas por la máquina. También se actualiza el valor de la propiedad `CurrentFunds`.

B.1.5. InvertSignal

Con la propiedad `InvertSignal` se invierte el tipo de posiciones que toma la máquina de trading. Si su valor es `false`, entonces la señal se calcula de la forma habitual según determina la estrategia de la máquina de trading; si su valor es `true`, entonces la señal queda invertida, es decir, las posiciones largas tornan en cortas y las cortas en largas.

B.1.6. InvestmentLimit

La propiedad `InvestmentLimit` marca la cantidad máxima que invierte la máquina de trading en una operación de mercado: si el capital que resta a la máquina es menor a esta cantidad, entonces lo invierte por completo; sino, entonces invierte el máximo indicado por este valor.

B.1.7. `ProfitLossTestSet`

La propiedad dependiente de sólo lectura `ProfitLossTestSet` indica el capital que posee la máquina después de invertir durante el intervalo de test. Esta cantidad está normalizada, de manera que el valor 1 representa a la inversión inicial.

B.1.8. `ProfitLossTrainingSet`

La propiedad dependiente de sólo lectura `ProfitLossTrainingSet` indica el capital que posee la máquina después de invertir durante el intervalo de entrenamiento. Esta cantidad está normalizada, de manera que el valor 1 representa a la inversión inicial.

B.1.9. `Signal`

La propiedad `Signal` almacena las posiciones que adopta la máquina en el mercado a lo largo del tiempo. Consiste en un vector fila de misma longitud que la serie de precios del objeto `DataSerie` referenciado por la máquina de trading. El valor de esta propiedad es calculado por el método `computeSignal` de acuerdo a la estrategia de trading.

El sistema de actualización perezosa hace que el valor de esta propiedad sólo se calcule cuando se lo requiere y no cuando cambian los parámetros que alteran su valor.

B.1.10. `TestSet`

La propiedad `TestSet` establece el intervalo de test en los datos de entrada. Consiste en un vector de dos elementos donde el primero marca el inicio del intervalo y el segundo el final. El valor de estos elementos se corresponde con índices de las series de datos del objeto de tipo `DataSerie` de la máquina de trading.

B.1.11. `TradingCost`

Con la propiedad `TradingCost` se fija el coste de emitir una operación al mercado de acciones. Cada apertura o cierre de posición conlleva una operación de mercado. El coste de operación depende del capital transferido y se descompone en una cantidad fija y otra variable. Esta propiedad consiste en una tabla de tres columnas donde la primera es la cantidad máxima por debajo de la cual se aplica cierta tarifa, la segunda representa la comisión fija y la tercera la comisión variable. Si hay varias filas, entonces se ordenan de manera creciente de acuerdo a la primera columna: en este caso, la tarifa se aplica por franjas. Una última fila debe utilizar el valor `Inf` en la primera columna para así cubrir todas las posibilidades.

B.1.12. `TrainingSet`

La propiedad `TrainingSet` establece el intervalo de entrenamiento en los datos de entrada. Consiste en un vector de dos elementos donde el primero marca el inicio del intervalo y el segundo el final. El valor de estos elementos se corresponde con índices de las series de datos del objeto de tipo `DataSerie` de la máquina de trading.

B.2. Métodos

Es necesario comentar la manera en que MATLAB permite organizar el código para implementar una clase: una forma consiste en crear un único archivo monolítico que contenga toda la definición e implementación de la clase; otra posibilidad es crear un archivo con el nombre de la clase y almacenarlo en un directorio con el mismo nombre que este archivo, aunque sin la extensión y precedido del carácter @. La clase `AlgoTrader` se organiza de acuerdo a la segunda opción. Utilizando tal posibilidad, se crean los métodos de la clase como archivos independientes contenidos en el directorio `@AlgoTrader`. En este caso, hay que especificar en el archivo de la clase los atributos del método: estos son las restricciones de acceso (`public`, `private`, `protected`) y sus características (`static`, `abstract`). Los métodos de las clases en MATLAB, a excepción del constructor, tienen como primer argumento la instancia de la clase. Esto es así porque MATLAB permite utilizar los métodos de dos formas diferentes: una de ellas consiste en emplear el separador '.', colocando delante la instancia de la clase y detrás el método; la otra posibilidad es usar el método como si fuera una función, donde el primer argumento es la instancia de la clase.

B.2.1. `AlgoTrader`

Constructor de la clase `AlgoTrader`.

Sintaxis

```
algoTrader = AlgoTrader(dataSerie)
```

Argumentos

`dataSerie` Instancia de la clase `DataSerie` con la información de la acción bursátil.

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

B.2.2. `clone`

Hace una copia de la máquina de trading. Las variables que maneja MATLAB son apuntadores hacia instancias de máquinas de trading, de manera que una asignación copia el apuntador, pero no la instancia.

Sintaxis

```
newObj = clone(this)
```

Argumentos

`this` Instancia de la clase `AlgoTrader` a copiar.

`newObj` Nueva instancia creada. Consiste en una copia de la instancia pasada en el parámetro `this`.

B.2.3. `computeSignal`

Calcula la señal de posición de la máquina de trading. Es un método abstracto que ha de ser implementado por la clase que hereda de `AlgoTrader` para definir la manera en que se calcula la señal de posición.

Sintaxis

```
computeSignal(algoTrader)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

B.2.4. `diffProfitLossSerie`

Calcula la diferencia entre valores sucesivos de la serie de beneficio/pérdida.

Sintaxis

```
[diffProfitLossSerie, positionSerie, priceSerie, dateTimeSerie, ...
longPosition, shortPosition, noPosition] ...
= diffProfitLossSerie(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`diffProfitLossSerie` Serie con los datos del diferencial entre valores sucesivos de la serie de beneficio/pérdida.

`positionSerie` Serie con los datos del capital invertido en el mercado por la máquina de trading.

`profitLossSerie` Serie con los datos del beneficio/pérdida consecuencia de la inversión de la máquina de trading.

`priceSerie` Serie con los datos del precio de la acción.

`datetimeSerie` Serie con los datos del instante en que se tomaron las observaciones en las series temporales.

`longPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición larga en la máquina de trading. La primera columna se refiere al índice de comienzo del intervalo y la segunda columna al índice de final del intervalo. La matriz tiene tantas filas como posiciones largas hayan sido efectuadas por la máquina de trading.

`shortPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición corta en la máquina de trading.

`noPosition` Matriz de dos columnas para indicar el intervalo durante el que la máquina permanece fuera del mercado.

B.2.5. `drawDiffProfitLoss`

Dibuja la gráfica de la diferencia en los valores sucesivos de la serie de beneficio/pérdida. Los datos que visualiza son los calculados por el método `diffProfitLoss` a lo largo del intervalo temporal determinado por `DateTime`. Se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawDiffProfitLoss(algoTrader, axesHandle, setSelector, initIndex, endIndex)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`initIndex` Índice a partir del cual efectuar los cálculos en el método.

`endIndex` Índice hasta el cual efectuar los cálculos.

B.2.6. drawPosition

Dibuja la gráfica de la posición adoptada en el mercado por la máquina de trading. Los datos que visualiza son los calculados por el método `positionSerie` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawPosition(algoTrader, axesHandle, setSelector, initIndex, endIndex)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`initIndex` Índice a partir del cual efectuar los cálculos en el método.

`endIndex` Índice hasta el cual efectuar los cálculos.

B.2.7. drawProfitLoss

Dibuja la gráfica del beneficio/pérdida logrado por la máquina de trading. Los datos que visualiza son los calculados por el método `profitLossSerie` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawProfitLoss(algoTrader, axesHandle, setSelector, initIndex, endIndex)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`initIndex` Índice a partir del cual efectuar los cálculos en el método.

`endIndex` Índice hasta el cual efectuar los cálculos.

B.2.8. drawSeriePosition

Dibuja la gráfica de la cotización de la acción coloreando los trazos según la posición que toma la máquina respecto al mercado. Los datos que visualiza son los de la propiedad `Serie` del objeto de tipo `DataSerie` asociado a la máquina de trading a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawSeriePosition(algoTrader, axesHandle, setSelector, initIndex, endIndex)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`initIndex` Índice a partir del cual efectuar los cálculos en el método.

`endIndex` Índice hasta el cual efectuar los cálculos.

B.2.9. drawSignal

Dibuja la gráfica de la señal que determina la posición que adopta la máquina de trading en el mercado. Los datos que visualiza son los de la propiedad `Signal` a lo largo del intervalo temporal determinado por `DateTime`. Este método se utiliza como argumento de la función `plotWrapper`.

Sintaxis

```
drawSignal(algoTrader, axesHandle, setSelector, initIndex, endIndex)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`initIndex` Índice a partir del cual efectuar los cálculos en el método.

`endIndex` Índice hasta el cual efectuar los cálculos.

B.2.10. `fitness`

Efectúa la medición de una característica de la máquina de trading. La característica concreta que mide depende de la función de evaluación pasada como argumento.

Sintaxis

```
performance = fitness(algoTrader, eFunction)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`eFunction` Función de evaluación para calcular el desempeño de la máquina de trading.

`performance` Valor de la medición según establece la función de evaluación.

B.2.11. `fitnessFunctionWrapper`

Método protegido (sólo accesible desde clases que heredan de `AlgoTrader`) con cuya inicialización parcial se crea la función de aptitud que utilizan los métodos de optimización.

Sintaxis

```
performance = fitnessFunctionWrapper(algoTrader, ...
eFunction, propertyName, propertyDomain, indexArray)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`eFunction` Función de evaluación para calcular el desempeño de la máquina de trading.

`propertyName` Estructura de acceso lineal (array de celdas en MATLAB) que contiene el nombre de aquellas propiedades cuyos valores óptimos se calculan durante el proceso de optimización.

`propertyDomain` Estructura de acceso lineal (array de celdas en MATLAB) que contiene el dominio de valores de cada una de las propiedades cuyos valores óptimos se calculan durante el proceso de optimización.

`indexArray` Estructura de acceso lineal (array en MATLAB) que referencia los valores de las propiedades con que se evalúa la máquina de trading. La longitud de este array es igual al número de propiedades de la máquina que se buscan optimizar. Cada elemento de este array indica, según su posición, el valor de la propiedad con que se mide el desempeño de la máquina.

`performance` Valor de la medición según establece la función de evaluación.

B.2.12. `fitnessStatistics`

Aglutina la información del espacio de búsqueda en una serie de valores estadísticos. Dibuja un histograma donde las máquinas de trading son agrupadas de acuerdo a su desempeño. El espacio de búsqueda es inducido por la función de evaluación sobre los valores de los parámetros con que se llama al método.

Sintaxis

```
[minFitness, maxFitness, meanFitness, stdFitness] ...
= fitnessStatistics(algoTrader, varargin)
```

```
[minFitness, maxFitness, meanFitness, stdFitness] ...
= fitnessStatistics(algoTrader, eFunction, varargin)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`eFunction` Función de evaluación para calcular el desempeño de la máquina de trading.

`varargin` Resto de argumentos que se pasan al método. Sirve para establecer las configuraciones de la máquina de trading que se someten al proceso de evaluación en el cálculo de estadísticos. Para cada parámetro de la máquina que se quiere examinar hay que escribir primero su nombre y luego el conjunto de valores en donde se busca el óptimo. La forma de introducir estos pares es idéntica a como se hace para el caso del método `optimize`.

`minFitness` Mínimo valor de las mediciones hechas con la función de evaluación.

`maxFitness` Máximo valor de las mediciones hechas con la función de evaluación.

`meanFitness` Valor promedio de las mediciones hechas con la función de evaluación.

`stdFitness` Desviación típica de las mediciones hechas con la función de evaluación.

B.2.13. `optimize`

Optimiza los valores de los parámetros de la máquina de trading. Este método es el responsable de ejecutar el algoritmo de optimización con las funciones de evaluación y selección sobre el espacio de búsqueda especificado.

Sintaxis

```
optimize(algoTrader, varargin)

optimize(algoTrader, eFunction, varargin)

optimize(algoTrader, eFunction, sFunction, varargin)

optimize(algoTrader, eFunction, sFunction, oFunction, varargin)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`eFunction` Función de evaluación para calcular el desempeño de la máquina de trading.

`sFunction` Función de selección para elegir la máquina de trading óptima.

`oFunction` Función que implementa el método de optimización a emplear.

`varargin` Resto de argumentos que se pasan al método. Sirve para establecer las posibles configuraciones de la máquina de trading de donde se busca aquella óptima. Para cada parámetro de la máquina que se quiere examinar hay que escribir primero su nombre y luego el conjunto de valores en donde se busca el óptimo. Por ejemplo, si se quiere optimizar la máquina `m` hallando el valor óptimo del parámetro `a` en el conjunto `[1 2 3]` y del parámetro `b` en `{'x', 'y', 'z'}`, entonces la llamada al método es `m.optimize('a',[1 2 3],'b',{'x', 'y', 'z'})`.

B.2.14. `plot`

Genera el gráfico con que se representa a la máquina de trading. Crea y configura la ventana, y mediante llamadas sucesivas al método `plotWrapper` produce los gráficos deseados.

Sintaxis

```
fig = plot(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

B.2.15. `plotDiffProfitLoss`

Genera el gráfico con que se representa al diferencial de los valores sucesivos de la serie de beneficio/pérdida. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawDiffProfitLoss` como argumento.

Sintaxis

```
fig = plotDiffProfitLoss(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual dibujar las gráficas.

`rangeEnd` Fecha en la serie de datos hasta la cual dibujar las gráficas.

`fig` Apuntador a la figura generada.

B.2.16. `plotPosition`

Genera el gráfico con que se representa a las posiciones que toma la máquina de trading en el mercado. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawPosition` como argumento.

Sintaxis

```
fig = plotPosition(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`fig` Apuntador a la figura generada.

B.2.17. `plotProfitLoss`

Genera el gráfico con que se representa la curva del beneficio/pérdida conseguido por la máquina de trading a lo largo del tiempo. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawProfitLoss` como argumento.

Sintaxis

```
fig = plotProfitLoss(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`fig` Apuntador a la figura generada.

B.2.18. `plotSearchSpace`

Genera el gráfico que representa al espacio de búsqueda inducido por la función de evaluación y el dominio de valores de los parámetros. Descompone el espacio de búsqueda resultante para obtener un equivalente en dos dimensiones que mostrar por pantalla.

Sintaxis

```
fig = plotSearchSpace(algoTrader, varargin)
```

```
fig = plotSearchSpace(algoTrader, eFunction, varargin)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`eFunction` Función de evaluación para calcular el desempeño de la máquina de trading.

`varargin` Resto de argumentos que se pasan al método. Sirve para establecer las configuraciones de la máquina de trading que se someten al proceso de evaluación en la generación del espacio de búsqueda. Para cada parámetro de la máquina que se quiere representar hay que escribir primero su nombre y luego el conjunto de valores que toma. La forma de introducir estos pares es idéntica a como se hace para el caso del método `optimize`.

`fig` Apuntador a la figura generada.

B.2.19. `plotSearchSpace123`

Genera el gráfico que representa al espacio de búsqueda inducido por la función de evaluación y el dominio de valores de los parámetros. Muestra espacios de búsqueda de hasta tres dimensiones.

Sintaxis

```
fig = plotSearchSpace123(algoTrader, varargin)

fig = plotSearchSpace123(algoTrader, eFunction, varargin)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`eFunction` Función de evaluación para calcular el desempeño de la máquina de trading.

`varargin` Resto de argumentos que se pasan al método. Sirve para establecer las configuraciones de la máquina de trading que se someten al proceso de evaluación en la generación del espacio de búsqueda. Para cada parámetro de la máquina que se quiere representar hay que escribir primero su nombre y luego el conjunto de valores que toma. La forma de introducir estos pares es idéntica a como se hace para el caso del método `optimize`.

`fig` Apuntador a la figura generada.

B.2.20. `plotSeriePosition`

Genera el gráfico con que se representa al precio de la acción y las posiciones adoptadas por la máquina. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawSeriePosition` como argumento.

Sintaxis

```
fig = plotSeriePosition(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`fig` Apuntador a la figura generada.

B.2.21. `plotSignal`

Genera el gráfico con que se representa a la señal de posición producida por la máquina de trading. Realiza una llamada al método `plotWrapper` utilizando el método de dibujado `drawSignal` como argumento.

Sintaxis

```
fig = plotSignal(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`fig` Apuntador a la figura generada.

B.2.22. plotWrapper

Inicializa y configura la ventana donde se muestran los gráficos. Se encarga de establecer la manera en que se interactúa con los gráficos a partir de la función de *panning*, que cambia la sección del gráfico mostrado en pantalla ante eventos del ratón, y *zooming*, que aumenta o disminuye la escala en que se visualiza el gráfico. A partir de la llamada al método `plotWrapper`, utilizando como argumento alguno de los métodos de dibujado `draw*`, se consigue la representación de la información de la acción bursátil. Este método es utilizado por los métodos `plot*` para generar los gráficos.

Sintaxis

```
fig = plotWrapper(algoTrader, customizer, axesHandle, ...
                  setSelector, rangeInit, rangeEnd, varargin)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`customizer` Método de dibujado para representar la información de la máquina de trading.

`axesHandle` Apuntador al eje de la figura en el que dibujar.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`varargin`

`fig` Apuntador a la figura generada.

B.2.23. positionSerie

Calcula las posiciones que toma la máquina de trading en el mercado.

Sintaxis

```
[positionSerie, priceSerie, dateTimeSerie, ...
longPosition, shortPosition, noPosition] ...
= positionSerie(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`diffProfitLossSerie` Serie con los datos del diferencial entre valores sucesivos de la serie de beneficio/pérdida.

`positionSerie` Serie con los datos del capital invertido en el mercado por la máquina de trading.

`profitLossSerie` Serie con los datos del beneficio/pérdida consecuencia de la inversión de la máquina de trading.

`priceSerie` Serie con los datos del precio de la acción.

`datetimeSerie` Serie con los datos del instante en que se tomaron las observaciones en las series temporales.

`longPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición larga en la máquina de trading. La primera columna se refiere al índice de comienzo del intervalo y la segunda columna al índice de final del intervalo. La matriz tiene tantas filas como posiciones largas hayan sido efectuadas por la máquina de trading.

`shortPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición corta en la máquina de trading.

`noPosition` Matriz de dos columnas para indicar el intervalo durante el que la máquina permanece fuera del mercado.

B.2.24. `positionsLog`

Calcula las columnas de la tabla de registros de posición. En ella se almacenan los resultados conseguidos por la máquina de trading para cada una de las posiciones adoptadas en el mercado.

Sintaxis

```
[positionType, openIndex, closeIndex, ...
openDate, closeDate, openPrice, closePrice, profitLoss] ...
= positionsLog(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`positionType` Tipo de posición tomada por la máquina de trading. Se corresponde con los datos de la columna del tipo de posición en la tabla de registros de posición.

`openIndex` Índice de apertura de posición. Se corresponde con los datos de la columna del índice de apertura en la tabla de registros de posición.

`closeIndex` Índice de cierre de posición. Se corresponde con los datos de la columna del índice de cierre en la tabla de registros de posición.

`openDate` Fecha de apertura de posición. Se corresponde con los datos de la columna de fecha de apertura en la tabla de registros de posición.

`closeDate` Fecha de cierre de posición. Se corresponde con los datos de la columna de fecha de cierre en la tabla de registros de posición.

`openPrice` Precio de la acción en la apertura de la posición. Se corresponde con los datos de la columna de precio de apertura en la tabla de registros de posición.

`closePrice` Precio de la acción en el cierre de la posición. Se corresponde con los datos de la columna de precio de cierre en la tabla de registros de posición.

`profitLoss` Beneficio/pérdida consecuencia de la posición adoptada. Se corresponde con los datos de la columna de beneficio/pérdida en la tabla de registros de posición.

B.2.25. `printPositionsLog`

Escribe por pantalla la tabla de registros de posición.

Sintaxis

```
printPositionsLog(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

B.2.26. `profitLoss`

Calcula el beneficio/pérdida logrado por la máquina de trading.

Sintaxis

```
profitLoss = profitLoss(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`profitLoss` Beneficio/pérdida consecuencia de la actuación de la máquina de trading en la bolsa.

B.2.27. `profitLossSerie`

Calcula la serie de beneficio/pérdida que refleja la evolución del resultado conseguido por la máquina de trading en el mercado.

Sintaxis

```
[profitLossSerie, positionSerie, priceSerie, dateTimeSerie, ...
longPosition, shortPosition, noPosition] ...
= profitLossSerie(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`diffProfitLossSerie` Serie con los datos del diferencial entre valores sucesivos de la serie de beneficio/pérdida.

`positionSerie` Serie con los datos del capital invertido en el mercado por la máquina de trading.

`profitLossSerie` Serie con los datos del beneficio/pérdida consecuencia de la inversión de la máquina de trading.

`priceSerie` Serie con los datos del precio de la acción.

`datetimeSerie` Serie con los datos del instante en que se tomaron las observaciones en las series temporales.

`longPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición larga en la máquina de trading. La primera columna se refiere al índice de comienzo del intervalo y la segunda columna al índice de final del intervalo. La matriz tiene tantas filas como posiciones largas hayan sido efectuadas por la máquina de trading.

`shortPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición corta en la máquina de trading.

`noPosition` Matriz de dos columnas para indicar el intervalo durante el que la máquina permanece fuera del mercado.

B.2.28. `resetSignal`

Método privado que maneja los eventos que reinician la señal de posición almacenada en la propiedad `Signal`. Este método es parte del mecanismo de actualización perezosa implementado en la clase `AlgoTrader`. Sólo se ejecuta cuando se altera alguno de los parámetros de la máquina de trading que interviene en el cálculo de la señal.

Sintaxis

```
resetSignal(algoTrader, src, event)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`src` Fuente desde la que se produce el evento (parámetro obligatorio en aquellos métodos que manejan eventos).

`event` Evento que produce la ejecución del método (parámetro obligatorio en aquellos métodos que manejan eventos).

B.2.29. `signal2positions`

Transforma la señal de posición en un conjunto de intervalos. Cada intervalo representa el instante inicial y el final en que la máquina mantiene una posición en el mercado.

Sintaxis

```
[longPosition, shortPosition, noPosition, initIndex, endIndex] ...
= signal2positions(algoTrader, setSelector, rangeInit, rangeEnd)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`setSelector` Indica el intervalo de la serie de datos sobre el que se aplica el método: con `0` para el de entrenamiento, `1` para el de test, `2` para la unión de ambos y `3` para todo el intervalo del cual se disponga de datos.

`rangeInit` Fecha en la serie de datos a partir de la cual calcular la serie.

`rangeEnd` Fecha en la serie de datos hasta la cual calcular la serie.

`longPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición larga en la máquina de trading. La primera columna se refiere al índice de comienzo del intervalo y la segunda columna al índice de final del intervalo. La matriz tiene tantas filas como posiciones largas hayan sido tomadas por la máquina de trading.

`shortPosition` Matriz de dos columnas para indicar el intervalo de duración de una posición corta en la máquina de trading.

`noPosition` Matriz de dos columnas para indicar el intervalo durante el que la máquina permanece fuera del mercado.

`initIndex` Índice a partir del cual efectuar los cálculos en el método.

`endIndex` Índice hasta el cual efectuar los cálculos.

B.2.30. `updateSignal`

Método privado que maneja los eventos que inician el proceso de cálculo de la señal de posición. Este método es parte del mecanismo de actualización perezosa implementado en la clase `AlgoTrader`. Se ejecuta cuando se requiere leer el valor de la señal de posición de la máquina de trading. Lo que hace este método es computar el valor de la propiedad `Signal` en caso de que alguno de los parámetros de los que depende haya alterado su valor.

Sintaxis

```
updateSignal(algoTrader, src, event)
```

Argumentos

`algoTrader` Instancia de la clase `AlgoTrader` que implementa la estrategia de trading.

`src` Fuente desde la que se produce el evento (parámetro obligatorio en aquellos métodos que manejan eventos).

`event` Evento que produce la ejecución del método (parámetro obligatorio en aquellos métodos que manejan eventos).

B.3. El sistema de dibujado

El sistema de dibujado está dividido en partes que colaboran en la construcción de los gráficos: de esta manera se consigue un código fácil de mantener, alterar y extender. Este proceso está dividido en dos etapas: primero se crea y configura la ventana contenedor y después se dibujan los gráficos que representan la información. La manera de acometer esta separación es utilizando el método `plotWrapper`, que se encarga de la inicialización y toma como argumento un método `draw*`, cuyo objetivo es dibujar el gráfico. Para ocultar esta complejidad de cara al usuario se utilizan los métodos `plot*`, que generan el gráfico a través de una llamada a `plotWrapper` en combinación con un método de dibujado `draw*`. Por ejemplo, la clase `AlgoTrader` implementa el método `plotProfitLoss`, que crea la ventana con la curva de beneficios y pérdidas de la máquina de trading combinando los métodos `plotWrapper`

y `drawProfitLoss`. La ventaja de esta separación de roles en los métodos es que la implementación del método `plotWrapper` es la más compleja (al lidiar con los aspectos específicos del entorno MATLAB para configurar la ventana), mientras que la implementación de los métodos `plot*` y `draw*` es trivial.

Apéndice C

Ejemplos de uso

En este último apéndice se referencian algunos ejemplos de uso de la plataforma de inversión bursátil. Los ejemplos están programados como *scripts* en el lenguaje MATLAB. Con ellos se proporcionan ejemplos de uso para demostrar el potencial de la plataforma de inversión.

C.1. Toma de contacto

Consiste en una toma de contacto para familiarizarse con la plataforma de inversión. Se realizan algunas operaciones básicas sobre las series de datos y las máquinas de trading. Para cada objeto creado se muestran sus propiedades y se utilizan los métodos de generación de gráficos para mostrar su representación. El primer ejemplo se implementa en el archivo `Ejemplo1.m`.

C.2. Trabajando con series de datos

En este caso, se comparan las series de datos obtenidas desde dos proveedores de información bursátil respecto a la misma acción de la bolsa: el objetivo es resaltar las diferencias entre los proveedores. Este ejemplo se implementa en el archivo `Ejemplo2.m`.

C.3. Utilizando las máquinas de trading

En el tercer ejemplo se trabaja con un grupo de máquinas de trading para resaltar y evidenciar algunas de sus características. Se trata la cuestión de la modificación de parámetros para observar la forma en que afecta al comportamiento de las máquinas. El ejemplo está implementado en el archivo `Ejemplo3.m`.

C.4. Diseñando una máquina de trading compuesta

Trata sobre la creación de una máquina de trading compuesta de varios niveles. Este es un ejemplo ilustrativo para resaltar la flexibilidad de las máquinas compuestas. Se advierte

que las máquinas compuestas de varios niveles son construcciones demasiado artificiosas que carecen de sentido práctico. En el archivo `Ejemplo4.m` se implementa el ejemplo.

C.5. Optimizando el desempeño de las máquinas

Aquí se hace uso intensivo del mecanismo de optimización para sacar todo el provecho posible a las máquinas de trading. Se prueban diferentes funciones de evaluación y selección, y se comparan los métodos de optimización implementados. También se comprueba el tiempo de ejecución en virtud del número de procesadores utilizados. El quinto ejemplo se implementa en el archivo `Ejemplo5.m`.

C.6. Analizando los resultados

El análisis de resultados es parte vital de una herramienta destinada al uso en investigación. El ejemplo consiste en el análisis de los resultados provistos por algunas máquinas de trading: se observan las relaciones entre los valores de los parámetros que mejores resultados proporcionan, se calculan los estadísticos para resumir con ellos el comportamiento de la máquina, se muestran en un histograma las configuraciones de la máquina de trading agrupadas de acuerdo a su desempeño... El último ejemplo de uso está implementado en el archivo `Ejemplo6.m`.

Bibliografía

[1] George Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control.* Prentice-Hall, third edition, 1994.

[2] Clive W. J. Granger and Roselyne Joyeux. An introduction to long-memory time series nodels and fractional differencing. *Journal of Time Series Analysis*, 1:15–30, 1980.

[3] Jonathan R. M. Hosking. Fractional differencing. *Biometrika*, 68:165–176, 1981.

[4] Peter M. Robinson. *Time Series with Long Memory.* Oxford University Press, 2003.

[5] Robert F. Engle. *ARCH: Selected Readings.* Oxford University Press, 1995.

[6] Ramon Lawrence. Using neural networks to forecast stock market prices. 1997.

[7] Marijana Zekic. Neural network applications in stock market predictions - a methodology analysis. 1998.

[8] Birgul Egeli, Meltem Ozturan, and Bertan Badur. Stock market prediction using artificial neural networks. 2003.

[9] Jangmin O, Jae Won Lee, Sung-Bae Park, and Byoung-Tak Zhang. Stock trading by modelling price trend with dynamic bayesian networks. 2004.

[10] Yingjian Zhang. Prediction of financial time series with hidden markov models. Master's thesis, Shandong University, 2004.

[11] Benjamin Graham and David Dodd. *Security Analysis: Principles and Techniques.* McGraw-Hill, second edition, 2002.

[12] Ralph N. Elliott. *R.N. Elliott's Masterworks: The Definitive Collection.* New Classics Library, 2004.

[13] Thomas E. Kida. *Don't Believe Everything You Think.* Prometheus Books, 2006.

[14] Paul H. Cootner. *The Random Character of Stock Market Prices.* Cambridge, Mass., M.I.T. Press, 1964.

[15] Eugene F. Fama. The behaviour of stock market prices. *Journal of Business*, 38:34, 1965.

[16] William A. Sherden. *The Fortune Sellers: The Big Business of Buying and Selling Predictions.* John Wiley and Sons, 1998.

[17] Burton G. Malkiel. *A Random Walk Down Wall Street.* W. W. Norton & Company, 2003.

[18] Andrew Skabar and Ian Cloete. Neural networks, financial trading and the efficient markets hypothesis. *Australian Computer Science Communications*, 24, 2002.

[19] Gary Belsky and Thomas Gilovich. *Why Smart People Make Big Money Mistakes And How To Correct Them.* Simon & Schuster, 2000.

[20] John J. Murphy. *Technical Analysis of the Financial Markets.* New York Institute of Finance, 1999.

[21] Martin Pring. *Introduction to Technical Analysis.* McGraw-Hill, 1997.

[22] Joe DiNapoli. *Trading with DiNapoli Levels.* Coast Investment Software Inc., 1969.

[23] John A. Bollinger. *Bollinger on Bollinger Bands.* McGraw-Hill, 2001.

[24] Gerald Appel. *Technical Analysis.* FT Press, 2005.

[25] Gerald Appel and Edward Dobson. *Understanding MACD.* Traders Press Inc., 2008.

[26] John W. Wilder. *New Concepts in Technical Trading Systems.* Trend Research, 1978.

[27] Constance Brown. *Technical Analysis for the Trading Professional.* McGraw-Hill, 1999.

[28] George C. Lane. *Using Stochastics, Cycles & RSI.* published under the auspices of Investment Educators, 1986.

[29] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[30] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization.* Wiley-Interscience, 1997.

[31] Alexander Schrijver. *Combinatorial Optimization (3 volume, A,B, & C).* Springer, 2003.

[32] Leonid A. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automation and Remote Control*, 24:1337–1342, 1963.

[33] John H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, 1992.