Práctica de Algoritmos Heurísticos

Selección y evaluación de portafolios de inversión

Aragón Aboyte Mariana Elisa
Enríquez Nares Horacio
Guerra Alcalá Daniela
Hernández Mota Rodrigo

September 2016

Índice general

Capítulo 1

Generalidades

1.1. Objetivo

Utilizar algoritmos heurísticos para optimizar el proceso de selección y evaluación de portafolios de inversión.

1.2. Introducción

El diseño de portafolio presenta un reto en finanzas, además la simulación del comportamiento de un portafolio en el tiempo es un buen ejercicio para tener una idea del desempeño del portafolio elegido.

La selección de un portafolio por medio de la teoría de Markowitz, supone que el portafolio será válido en un periodo de tiempo sin modificación de las ponderaciones de los activos. Por otra parte, existen algoritmos de compra y venta de acciones de manera automática - trading - siguiendo ciertas reglas para tomar la decisión de realizar una operación. Uno de los algoritmos más sencillos se basa en la comparación del promedio móvil de la acción con el precio de la acción a la cual se le aplicará la operación. Las reglas a seguir son simples: Si el precio de la acción cruza positivamente el promedio móvil, entonces se realiza una operación de compra de acciones; por otro lado, si el precio de la acción tiene un cruce negativo del promedio móvil, se realiza una operación de venta.

En la primera parte del documento, se presenta una práctica sobre la teoría de Markowitz, mientras que en la segunda sección se muestra una aplicación de un portafolio utilizando algoritmos de trading.

1.3. Activos financieros utilizados

Para el desarrollo del presente documento, se utilizaron 6 activos cotizando en la Bolsa Mexicana de Valores (BMV) cuyos precios históricos diarios pueden ser consultados utilizando la plataforma Yahoo Finance.

- **GFNORTEO.MX**: Nombre Comercial: Grupo Financiero Banorte

  Banorte es una institución financiera en México, fundada en la ciudad de Monterrey en 1899. Ofrece una gran variedad de productos y servicios financieros a través de casa de bolsa, compañías de pensiones y seguros, Afores, sociedades de inversión, empresas de arrendamiento, factoraje y almacenadora. Está posicionada en los primeros lugares en las categorías de instituciones bancarias más grandes de México, proveedores más importantes en colocación de créditos a gobiernos, bancos más importantes en créditos hipotecarios, cartera comercial y tarjetas de crédito.

- **LIVERPOLC-1.MX**: Comercial: El Puerto de Liverpool

  El Puerto de Liverpool, fundada en México en 1847, opera una de las cadenas de tiendas departamentales en el país. Inició su cotización en la Bolsa Mexicana de Valores en 1965. A lo largo de los años ha adquirido empresas importantes como Salinas y Rocha y Fábricas de Francia, además de centros comerciales en ciudades importantes de México, y aumentado el número de tiendas. Además de la venta de ropa, novedades y artículos para el hogar, Liverpool ofrece tarjetas de crédito, productos de seguros y servicios.

- **HERDEZ.MX**: Nombre Comercial: Grupo Herdez

  Grupo Herdez es una compañía que se encarga de la manufactura, compra, distribución y venta de alimentos envasados y enlatados dentro de México y Estados Unidos. Sus productos van desde salsas caseras y vegetales en conserva hasta burritos, atún y especias. Sus negocios, además de estar en el sector de alimentos, están en los inmobiliarios. Uno de sus principales objetivos es ser una empresa líder en comida mexicana a nivel internacional, siendo Estados Unidos su mercado principal.

- **BIMBOA.MX**: Nombre Comercial: Grupo Bimbo

  Bimbo es una empresa que se encuentra en el giro de alimentos. Fundada en 1945 en la Ciudad de México. Grupo Bimbo inició su expansión internacional en 1990 y hoy, es la empresa de panificación más grande del mundo. Sus principales líneas de productos incluyen pan de caja fresco y congelado, bollos, galletas, pastelitos, muffins, bagels, productos empacados, tortillas, botanas saladas y confitería, entre otros. Grupo Bimbo fabrica más de 10,000 productos, y tiene una plantilla laboral superior a los 127,000 colaboradores. Desde 1980, las acciones de Grupo Bimbo se

cotizan en la Bolsa Mexicana de Valores (BMV) bajo la clave de pizarra BIMBO.

- **SANMEXB.MX**: Nombre Comercial: Grupo Financiero Santander México

  Grupo Financiero Santander es una institución financiera que ofrece servicios a personas físicas y morales principalmente en México. Opera en Banca Comercial y Corporativa Global, ofreciendo a sus clientes productos de depósito, préstamos, tarjetas de crédito, hipotecas y corretaje de seguros. Por otro lado se encuentran los préstamos corporativos, financiamiento de capital social y de trabajo, de comercio exterior, gestión de efectivo y servicios de banca móvil, entre otros. Otras de sus actividades incluyen los mercados de capitales de deuda, financiamiento de proyectos, estructuración de capital, derivados de renta variable, los negociados en bolsa, acciones y operaciones de bonos.

- **ALSEA.MX**: Nombre Comercial: Alsea

  Alsea es el operador de restaurantes líder en América Latina y España con marcas de reconocimiento dentro de los segmentos de comida rápida, cafeterías, comida casual y restaurante familiar. Cuenta con un portafolio de multimarcas integrado por Domino´s Pizza, Starbucks, Burger King, Chili´s, California Pizza Kitchen, P.F. Chang´s, Italianni´s, TheCheesecake Factory, Vips, El Portón, Foster´s Hollywood, La Vaca Argentina, Cañas y Tapas e II Tempietto. La compañía opera más de 2,950 unidades y cuenta cerca de 60,000 colaboradores en México, Argentina, Chile, Colombia, Brasil y España.

Los activos financieros antes mencionados fueron seleccionados de forma arbitraria. Su propósito es meramente ilustrativo y como ejercicio de aplicación.

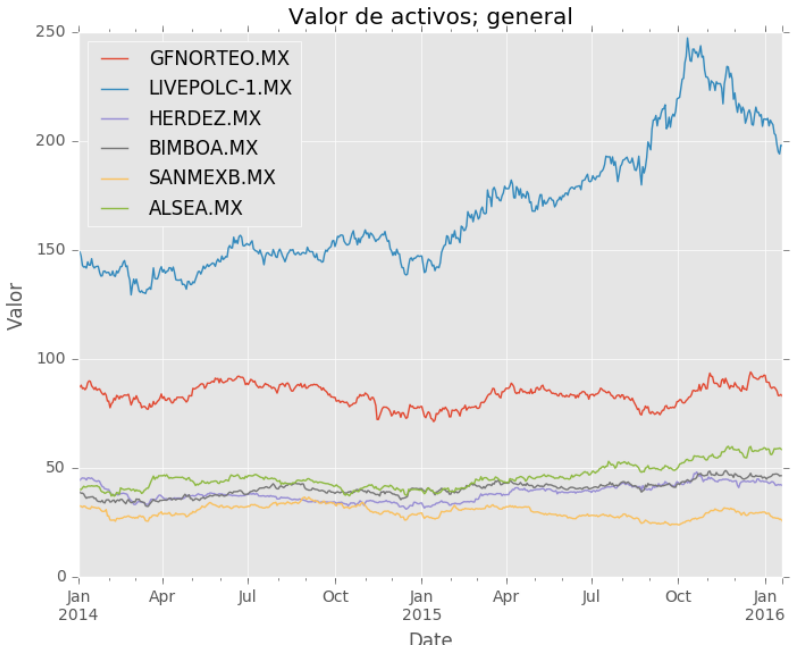


Figura 1.1: Valor de activos financieros

Capítulo 2

Selección de portafolio: Teoría de Markowitz

En esta sección se utiliza la teoría de Markowitz para definir las ponderaciones $x_0, x_1, \ldots, x_n$ de los activos subyacentes propuestos en la sección anterior ($n = 6$).

Se utiliza el algoritmo heurístico PSO -*Particle Swarm Optimization*- para encontrar las ponderaciones ideales basadas en la teoría de Markowitz según el perfil del inversionista. Con esta teoría se hace referencia a buscar aquel portafolio que maximice su rendimiento y, al mismo tiempo, minimice el riesgo.

El PSO, empleado para encontrar las mejores ponderaciones de los activos en cuestión, es un algoritmo heurístico que optimiza pero a diferencia del genético, aquí no se descarta a ninguno de los datos utilizados y no hay un límite en el rango empleado. Este algoritmo busca el mejor dentro de números aleatorios (no necesariamente tienen que ser enteros) que se van moviendo de tal forma que siguen al que va teniendo un mejor desempeño. El movimiento de las partículas (los números en los cuales busca) va acorde a una ecuación de movimiento:

$$Vp_{k+1} = Vp_k + c_1 * rand() * (Xpg_k - Xp_k) + c_2 * rand() * (Xpl_k - Xp_k)$$

$$Xp_{k+1} = Xp_k + Vp_{k+1}$$

Siguiendo a estas ecuaciones, las partículas se mueven según el líder y los saltos en el movimiento son cada vez más pequeños de tal forma que converja al óptimo. Entre más partículas se utilicen, el resultado se encuentra más rápido.

La función a minimizar por el algorítmo tiene la siguiente forma:

$$z = -\beta_1 r + \beta_2 \sigma + c \tag{2.1}$$

En donde r es el rendimiento de un posible portafolio, σ es su riesgo y c es una penalización en caso de que las proporciones del portafolio no cumplan con $x_i \in [0, 1]$ y $\sum x_i = 1$ para todo $i = 1, 2, ..., n$.

Los parámetros β_1 y β_2 son constantes mayores a cero que se ajustan según el perfil del inversionista, es decir, se establecen según la importancia que se le quiera dar a maximizar el rendimiento o minimizar el riesgo. Entre mayor sea β_1, el algoritmo buscará valores de $x_0, x_1, \ldots, x_n$ que generen rendimientos altos; por el contrario, incrementar β_2 genera que el algoritmo busque bajo riesgo. Esto es debido a que la función a minimizar (z) decrece cuando el rendimiento aumenta (i.e. $-\beta_1 r$) y aumenta cuando el riesgo disminuye (i.e. $\beta_2 \sigma$).

Según la teoría de Markowitz, si se tiene un conjunto de portafolios creados en base a los mismos activos subyacentes que comparten el mismo nivel de riesgo (σ); un inversionista racional debería escoger aquel que presente el mayor rendimiento.

Para procurar congruencia con esta teoría, se debe seleccionar β_1 y β_2 de tal forma que sean afines a tal definición. En este caso se han elegido $\beta_1 =$ y $\beta_2 =$.

2.1. Teoría de Markowitz

La teoría moderna de la selección de carteras fue originada por Harry Markowitz en 1952. Markowitz parte sobre la base del comportamiento racional del inversor en donde este desea la rentabilidad y rechaza el riesgo. Para ello demostró que la clave para diversificar un portafolio no estaba solamente en el número de acciones que lo componen, sino también en la correlación que existe entre los retornos de las acciones que lo conforman.

Obtuvo el Premio Nobel de Economía en 1990 compartido con Merton H. Miller y William F. Sharpe por su trabajo denominado ¡U+FFFD¿¡U+FFFD¿¡U+FFFD¿Teoría de Selección de Carteras¡U+FFFD¿¡U+FFFD¿¡U+FFFD¿. Dicha teoría fue la primera formalización matemática en materia de Finanzas en la diversificación de inversiones, en la cual, el Riesgo puede reducirse sin cambiar el rendimiento esperado de la cartera.

Un portafolio eficiente, según Markowitz, es aquel que tiene un mínimo riesgo, para un retorno dado o, equivalentemente un portafolio con un máximo retorno para un nivel de riesgo dado.

El rendimiento del portafolio está dado por:

$$R_p = \begin{bmatrix} r_0 & r_1 & \dots & r_n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{bmatrix}$$

Mientras que el riesgo del portafolio viene dado por:

$$\sigma_p^2 = \begin{bmatrix} w_0 & w_1 & \dots & w_n \end{bmatrix} \begin{bmatrix} \sigma_0^2 & \sigma_{0,1} & \dots & \sigma_{0,n} \\ \sigma_{1,0} & \sigma_{1,1}^2 & \dots & \sigma_{1,n} \\ \vdots & \dots & \dots & \vdots \\ \sigma_{n,0} & \sigma_{n,1} & \dots & \sigma_{n,n}^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

2.2. Ponderaciones

Llevando a cabo lo mencionado anteriormente, al ejecutar el algoritmo de optimización se obtienen los siguientes resultados:

Proporciones óptimas para los activos en cuestión Los pesos correspondientes para cada activo son:

Activo	Ponderación
GFNORTEO.MX	0.05471346
LIVERPOLC-1.MX	0.30778373
HERDEZ.MX	0.13873603
BIMBOA.MX	0.18363016
SANMEXB.MX	0.02029841
ALSEA.MX	0.29838743

Cuadro 2.1: Tabla con ponderaciones según Teoría Markowitz

Ver algoritmo en anexos.

2.3. Valuación del portafolio

Con las ponderaciones anteriores, se obtiene un portafolio con rendimiento y riesgo de:

- **Rendimiento**: 0,0005059139641425607

- **Riesgo**: $9{,}30415417912355x10^{-05}$

Resultado gráfico en figura 2.1.

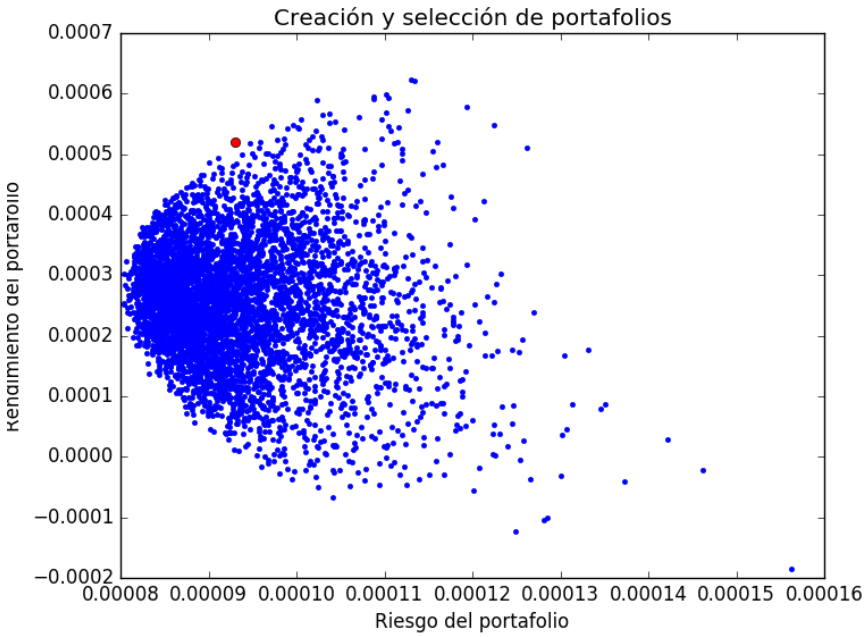


Figura 2.1: Ilustración de teoría de selección de portafolios basada en Markowitz

Capítulo 3

Algorítmo de Trading para creación de portafolio

En este capítulo se utilizará un algorítmo de trading que utiliza los mismos activos que la optimización realizada con la teoría de Markowitz en el capítulo anterior.

3.1. Algorítmos de trading: promedio móvil

Un algoritmo de trading es aquel algoritmo preestablecido que se utiliza para operar en los mercados financieros. Su función es realizar compras y ventas de activos financieros (siguiendo ciertas restricciones o condicionales) sin necesidad de que esté una persona haciéndolo, ya que la computadora lo lleva a cabo instantáneamente.

En este caso, se procederá a aplicar un algoritmo de trading basado en el promedio móvil. Se busca crear un portafolio con los mismos activos que en la sección anterior determinando las nuevas ponderaciones para cada activo pero ahora tomando en cuenta que el rendimiento estará basado en el promedio móvil.

La ventana de tiempo a utilizar será diferente para cada uno de los activos en cuestión, ya que ésta se determinará en base a un proceso de optimización (empleando un algoritmo heurístico; el genético), buscando aquella ventana que nos otorgue un mayor rendimiento promedio del balance y menor riesgo promedio del balance. Esto resultará de las operaciones de compra-venta para cada activo.

Al mismo tiempo, con el algoritmo Particle Swarm Optimization se pretende encontrar las ponderaciones óptimas para cada activo dadas las ventanas de

tiempo óptimas de los mismos.

Una vez obtenidos estos datos, se evaluará el portafolio para calcular su rendimiento y riesgo, mismos que se compararán con la evaluación del portafolio formado previamente con la teoría de Markowitz.

La relación entre el precio del activo y el valor del promedio móvil define las instrucciones de compra o venta. Tal relación se representa bajo las siguientes condiciones;

Operación de compra:

$$(p_{t-1} < pm_{t-1})\&(p_t > pm_t)$$

Operación de venta:

$$(p_{t-1} > pm_{t-1})\&(p_t < pm_t)$$

En donde t representa el tiempo, p representa el valor del activo y pm representa el valor del promedio móvil.

Por otro lado, es importante decir que entre menor sea la ventana de tiempo a tomar, se asemejará más a la real.

Para la creación del portafolio y obtención de rendimientos se considera que se invertirá la cantidad de 1,000,000 de pesos, repartido en cada acción pero no es necesario que se invierta todo el monto disponible. Además el tiempo de simulación del algoritmo de trading será de dos años (Enero de 2014 al día de hoy).

Aplicación de Métodos heurísticos

Primero que nada es importante de definir que un método heurístico son estrategias generales de resolución de problemas, basadas en la experiencia previa con problemas similares. Los métodos más utilizados son los algoritmos genéticos y el PSO (Particle Swarm Optimization).

El primero, es un algoritmo de optimización que está inspirado en la genética y selección natural, ya que los datos utilizados se ponen a prueba y sólo los mejores "sobrevivenz pasan a la siguiente generación. Este algoritmo tiene una serie de pasos para poder dar solución al problema; inicialización, selección, cruzamiento y mutación. Por otro lado, el método es capaz de ser aplicado para resolver un rango muy amplio de problemas.

Se utiliza este algoritmo genético para resolver el problema de optimización del portafolio mediante algoritmos de trading. En particular, se propone una función de desempeño f que depende de la ponderación de cada activo y de la ventana de tiempo del algoritmo de trading (por cada activo).

Para poder aplicar este algoritmo, es necesario inicializar la población (que en este caso cada poblador representa al conjunto de ponderaciones de cada activo y de ventanas de tiempo a probar) y a estos individuos iniciales se les aplica una codificación binaria en variables de reales. El conjunto de individuos forman una población.

Ciertas consideraciones deben ser tomadas en cuenta a la hora de aplicar este algoritmo. Debido a que estamos optimizando respecto a un conjunto de variables de diferentes tipos (pertenecientes a dos grupos), el espacio de búsqueda y el numero de bits necesarios de cada variable tiene que ser definido independientemente:

La siguiente función representa la distancia o salto entre un valor y el siguiente en un espacio de búsqueda dado (debido a la discretización del mismo).

$$\Delta = \frac{ls - li}{2^n - 1}$$

Donde ls y li representan los límites superior e inferior del rango y n es el número de bits.

En el caso de la ventana de tiempo, se determinan estos parámetros de tal forma que la siguiente expresión sea verdad:

$\Delta = 1$

Por lo tanto:

$ls = 128\ li = 1\ n = 7$

Es decir, el algoritmo de trading podrá buscar valores en un ventana entre 1 y 128 días. Con el número de bits igual a 7, se genera una discretización del espacio de búsqueda tal que solo se puede acceder a valores enteros.

En el caso de las variables que representan la ponderación de cada activo se tiene: $ls = 1\ li = 0\ n = 11$

Debido a que la variable x_i solamente debe tomar valores entre $[0, 1]$. El valor de $n = 11$ nos permite ser más precisos en la discretización del espacio de búsqueda, permitiendo acceso a más valores del mismo.

Versión Alternativa

En la versión alternativa para resolver este problema se comenzó por encontrar las ventanas óptimas para cada activo. Es decir, el tamaño de las ventas es independiente al nivel de capital (y por lo tanto; ponderación) que se utiliza para crear el portafolio e invertir el dinero.

Esta versión permite simplificar el código. Habiendo calculado las ventanas de

tiempo, se utiliza optimización por enjambre de partículas para determinar los valores de los pesos.

Ver código en anexos.

3.2. Ponderaciones

Proporciones óptimas para los activos en cuestión

Los pesos correspondientes para cada activo son:

Activo	Ponderación
GFNORTEO.MX	0.0870446911918
LIVERPOLC-1.MX	0.41447398072
HERDEZ.MX	0.1162624572072
BIMBOA.MX	0.2197448888
SANMEXB.MX	0.01458201987
ALSEA.MX	0.12608911062

Cuadro 3.1: Tabla con ponderaciones según algoritmo de trading mediante optimización por algoritmo genético

Ver algoritmo en anexos.

3.3. Valuación del portafolio

El rendimiento y el riesgo del portafolio obtenido son:

Con las ponderaciones anteriores, la valuación del portafolio resulta:

- **Rendimiento**: 0,00045789
- **Riesgo**: $3{,}99482624^{-05}$

Capítulo 4

Comparación y análisis de resultados

En este último capítulo se comparan los parámetros relevantes entre ambos portafolios.

4.1. Visualización y comparación

Los resultados del portafolio usando la teoría de Markowitz son los siguentes:

Activo	Ponderación
GFNORTEO.MX	0.05471346
LIVERPOLC-1.MX	0.30778373
HERDEZ.MX	0.13873603
BIMBOA.MX	0.18363016
SANMEXB.MX	0.02029841
ALSEA.MX	0.29838743

Cuadro 4.1: Tabla con ponderaciones según Teoría Markowitz

- **Rendimiento**: 0,0005059139641425607
- **Riesgo**: 9,30415417912355$x10^{-05}$

Por otra parte, el portafolio construido con el algoritmo de Trading tiene las siguentes características:

Activo	Ponderación	Ventana de tiempo	Rend.prom y std
GFNORTEO.MX	0.0870446911918	68	(0.0119,0.9399)
LIVERPOLC-1.MX	0.41447398072	84	(0.0779,1.3111)
HERDEZ.MX	0.1162624572072	127	(0.0245,1.1455)
BIMBOA.MX	0.2197448888	96	(-0.0053,1.1029)
SANMEXB.MX	0.01458201987	100.0	(0.0301,1.1036)
ALSEA.MX	0.12608911062	3.0	(0.0820,1.1424)

Cuadro 4.2: Tabla con datos utilizando algoritmo de Trading

- **Rendimiento**: 0,00045789
- **Riesgo**: $3{,}99482624x10^{-05}$

4.2. Conclusiones

La teoría de selección de portafolios con Markowitz nos permite visualizar la relación riesgo-rendimiento de un portafolio construido con diferentes ponderaciones sobre cierto conjunto de activos. Sabiendo esto, es congruente que un inversionista eliga aquel portafolio que ofrezca el mayor nivel de rendimiento dado un nivel de riesgo. En esta práctica se realizó la simulación de Markowitz y posteriormente se encontró el mejor portafolio disponible (con el algoritmo de optimización Heurístico; PSO) para un inversionista con ciertas preferencias para riesgo y rendimiento.

El portafolio que se obtuvo mediante esta técnica, aunque congruente, muestran un rendimiento relativamente bajo. Esto es debido a que la selección de los activos fue aleatoria. Una selección inteligente (analizar correlaciones entre variables y rendimientos) podría haber generado mejores portafolios. Aunado a esto, por simplicidad se omitieron las covarianzas entre los activos por cuestión de simplicidad, lo cual en práctica no es recomendable.

Entre otras limitaciones, el algoritmo PSO intenta optimizar una función de 6 dimensiones lo cual requiere de un gran número de partículas, iteraciones y una constante revisión de los parámetros de velocidad y restricciones. Para solucionar esto se recomendaría usar un algoritmo previo que considerando la covarianza y nivel de rendimiento de cada activo, busque la mejor combinación posible reduciendo el numero de activos.

Por otra parte, este reporte considera otra forma de seleccionar y crear un portafolio; mediante algoritmos de trading. El algoritmo utilizado en esta práctica se basa en un promedio móvil simple para decidir operaciones de compra y venta. Según el activo que se esté manejando, el capital disponible y la ventana de tiempo, este algoritmo genera un rendimiento diario. Al aplicar tal procedimiento a un conjunto de activos, se puede crear un portafolio.

Se utilizó el algoritmo heurístico de optimización genético para calcular la ventana de tiempo requerida y la ponderación de los activos (para repartir el capital inicial y construir el portafolio). El resultado obtenido no fue satisfactorio. El algoritmo encontró las variables requeridas bajo las restricciones impuestas, sin embargo el rendimiento final fue cercano a cero y de valor negativo. Analizando los rendimientos individuales de cada activo gestionado por el algoritmo de trading, podemos observar que gran parte de ellos también fue negativo.

Por otra parte, se usó versión alternativa a este algoritmo que consta en calcular las ventas previamente y realizar con esa información optimización por enjambre de partículas. Esta versión logró un rendimiento cercano al que se obtuvo con Markowitz.

Las desventajas que muestra el algoritmo de optimización empleado en este particular problema es que la función objetivo (algoritmo de trading y creación de

portafolio) era complicada y tardada de optimizar. En particular, considerando las ventanas de tiempo y la ponderación de cada activo, la función objetivo (o de desempeño) era de 12 dimensiones. Por esta razon y por falta de tiempo disponible, se utilizó un numero de pobladores e iteraciones relativamente bajos. Probablemente al aumentar estos parámetros el algoritmo sea capaz de encontrar una mejor solución; tal vez hasta igual o mejor que los resultados de Markowitz.

Habiendo analizado las consideraciones de ambos casos, se llega a la conclusión de que en la metodologá para crear un portafolio hay ciertas ventajas y desventajas respecto a los algoritmos que se pueden emplear en este proceso. En un principio se deberán escoger los activos subyacentes de forma inteligente. Posteriormente, bajo ciertas restricciones y considerando el tiempo disponible, la creación de portafolios mediante la teoría de Markowitz y optimización por enjambre de partículas parece mostrar mejores resultados que la alterantiva mostrada en este reporte. No obstante, no se descarta por completa la opción de utilizar algoritmos de trading con optimización mediante algoritmos genéticos. Sin embargo, se propone que aumentar el número de iteraciones, pobladores y utilizar un diferentes métodos (más eficientes) para la decisión de compra y venta. Se recomienda así mismo procurar la eficiencia del código; programación funcional, vectorización, entre otras para así lograr un mejor rendimiento y eficiencia.

Capítulo 5

Bibliografía

Bibliografía

[1] Bloomberg.(2016). Obtenido de Grupo Financiero Banorte. En `http://www.bloomberg.com/quote/GFNORTEO:MM` [consultado el 23 de septiembre de 2016]

[2] Bloomberg.(2016). Obtenido de El Puerto de Liverpool. En `http://www.bloomberg.com/quote/LIVEPOLC:MM` [consultado el 23 de septiembre de 2016]

[3] Herdez. (2016). *Grupo Herdez.* Obtenido de Historia. En `http://grupoherdez.mx/conocenos/estrategia/` [consultado el 23 de septiembre de 2016]

[4] Liverpool. (2016). *Liverpool.* Obtenido de Acerca de Liverpool: En `http://elpuertodeliverpool.mx/historia.html` [consultado el 23 de septiembre de 2016]

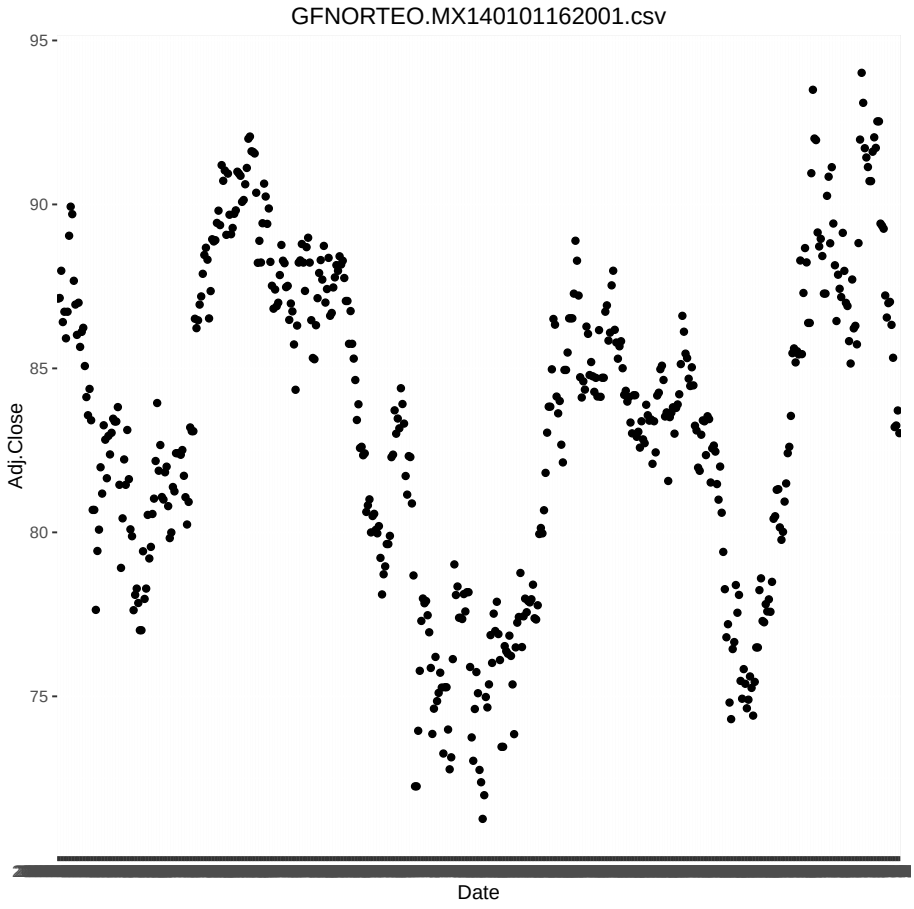
[5] Adams, A. T., Booth, P. M., Bowie, D. C., y Freeth, D. S. (2003). *Investment Mathematics.* Inglaterra: John Wiley & Sons Ltd. [Consultado el 23 de septiembre de 2016]

Capítulo 6

Anexos

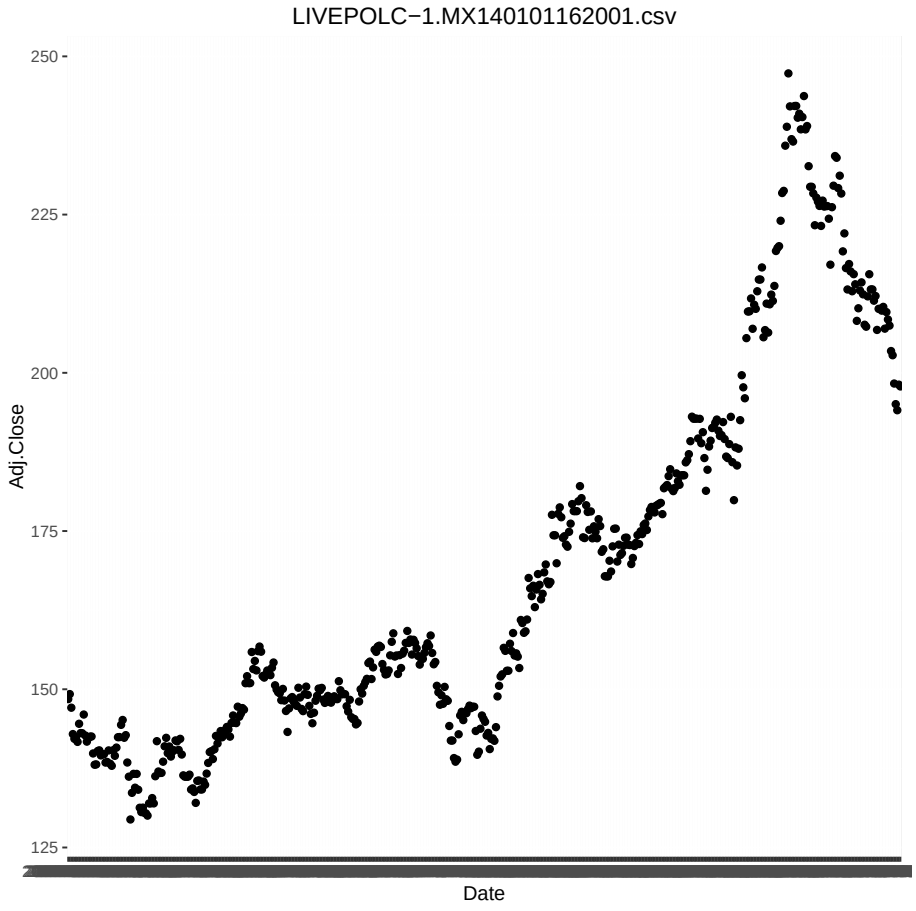
6.1. Datos históricos

```
## GFNORTEO.MX140101162001.csv
## H E A D
##         Date  Open  High   Low Close  Volume Adj.Close
## 1 2014-01-01 91.36 91.36 91.36 91.36       0   87.1285
## 2 2014-01-02 91.35 94.35 90.64 91.38 8798700   87.1475
## 3 2014-01-03 91.38 93.00 91.21 92.25 3984600   87.9772
## 4 2014-01-06 92.25 92.42 90.30 90.61 4491200   86.4132
## 5 2014-01-07 90.50 91.59 90.21 90.94 2629300   86.7279
## 6 2014-01-08 90.41 91.80 89.02 90.09 3596400   85.9173
## T A I L
##           Date  Open  High   Low Close   Volume Adj.Close
## 531 2016-01-13 89.70 89.70 88.01 88.62  8402800   86.3278
## 532 2016-01-14 88.05 88.76 87.40 87.59  7017400   85.3244
## 533 2016-01-15 87.00 87.00 85.04 85.42  8865700   83.2105
## 534 2016-01-18 85.30 85.97 85.00 85.47  1167200   83.2593
## 535 2016-01-19 86.18 86.30 85.05 85.94 12462300   83.7171
## 536 2016-01-20 85.90 85.90 84.06 85.23  8655500   83.0255
```



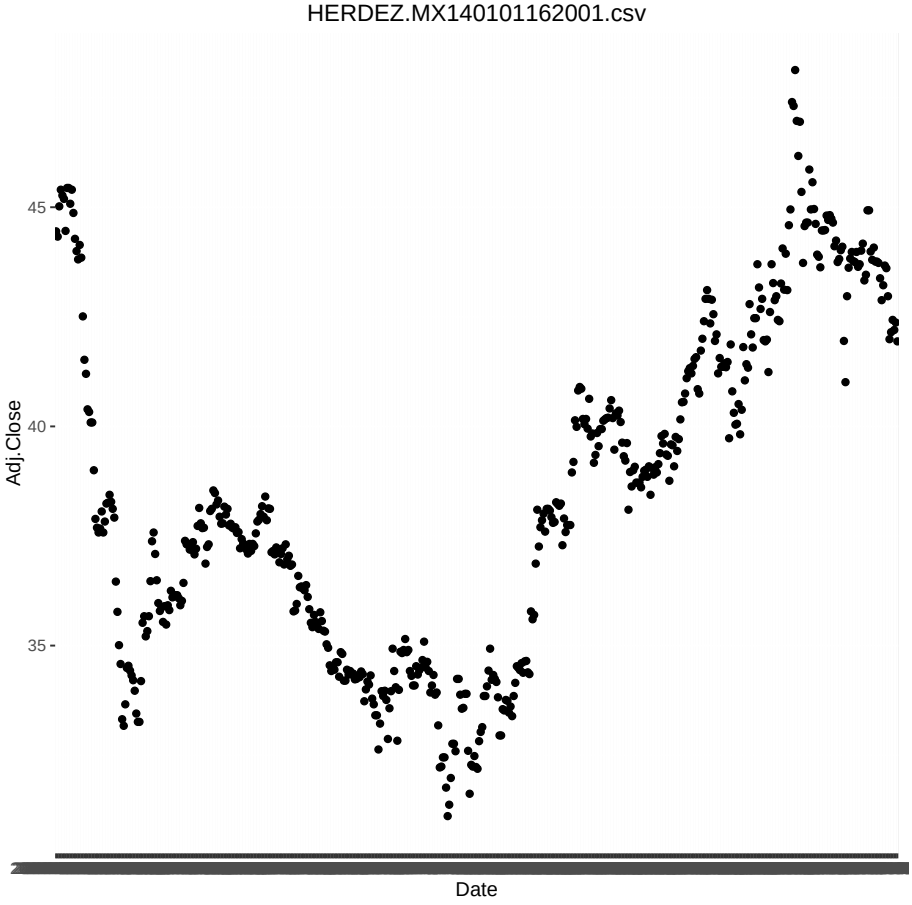

```
## LIVEPOLC-1.MX140101162001.csv
## H E A D
##         Date   Open   High    Low  Close Volume Adj.Close
## 1 2014-01-01 149.07 149.07 149.07 149.07      0    148.45
## 2 2014-01-02 150.50 150.50 147.52 149.86 116400    149.24
## 3 2014-01-03 147.75 149.69 145.45 147.70 137100    147.09
## 4 2014-01-06 149.70 149.70 142.43 143.52 263200    142.92
## 5 2014-01-07 142.80 144.33 142.42 142.76 162400    142.17
## 6 2014-01-08 142.60 144.42 142.12 142.97 343900    142.38
## T A I L
##           Date   Open   High    Low  Close  Volume Adj.Close
## 531 2016-01-13 204.21 205.00 198.01 202.78  571200    202.78
## 532 2016-01-14 203.99 206.99 197.02 198.31 1176500    198.31
## 533 2016-01-15 197.00 197.99 193.54 195.05  428200    195.05
## 534 2016-01-18 194.89 198.75 190.92 194.09  212300    194.09
## 535 2016-01-19 195.50 202.98 195.45 198.13  652000    198.13
```

```
## 536 2016-01-20 196.00 198.91 195.00 197.85 1024400    197.85
```



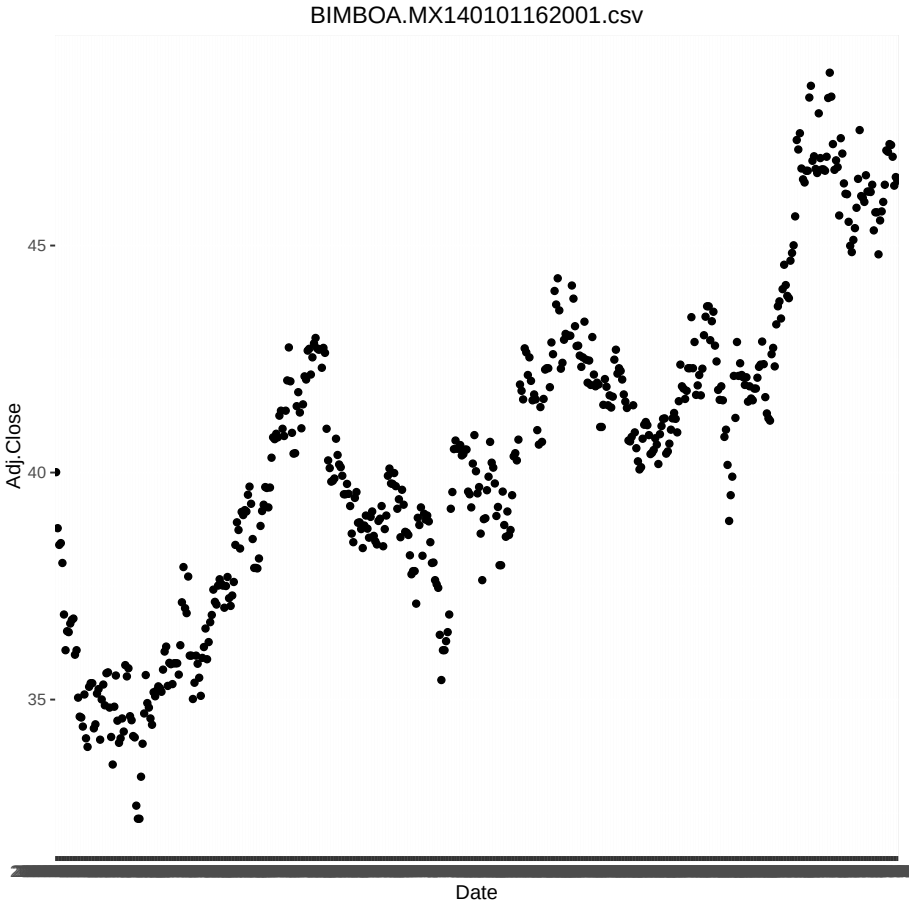

```
## HERDEZ.MX140101162001.csv
## H E A D
##         Date  Open  High   Low Close Volume Adj.Close
## 1 2014-01-01 46.47 46.47 46.47 46.47      0     44.45
## 2 2014-01-02 46.07 47.70 46.05 46.34  25900     44.33
## 3 2014-01-03 46.86 47.52 46.34 47.07  50000     45.02
## 4 2014-01-06 47.67 48.08 47.28 47.46  80500     45.40
## 5 2014-01-07 47.50 48.07 47.06 47.33 185900     45.27
## 6 2014-01-08 47.35 47.45 47.05 47.24  68100     45.19
## T A I L
##           Date  Open  High   Low Close  Volume Adj.Close
## 531 2016-01-13 43.99 44.42 42.49 42.96  130200     41.99
## 532 2016-01-14 43.06 44.00 42.62 43.12  140300     42.15
```

```
## 533 2016-01-15 43.02 43.99 42.68 43.41  463300     42.43
## 534 2016-01-18 43.50 43.60 43.00 43.17   21300     42.20
## 535 2016-01-19 43.45 43.90 42.31 43.35  241100     42.37
## 536 2016-01-20 43.00 43.00 41.25 42.91 1180400     41.94
```



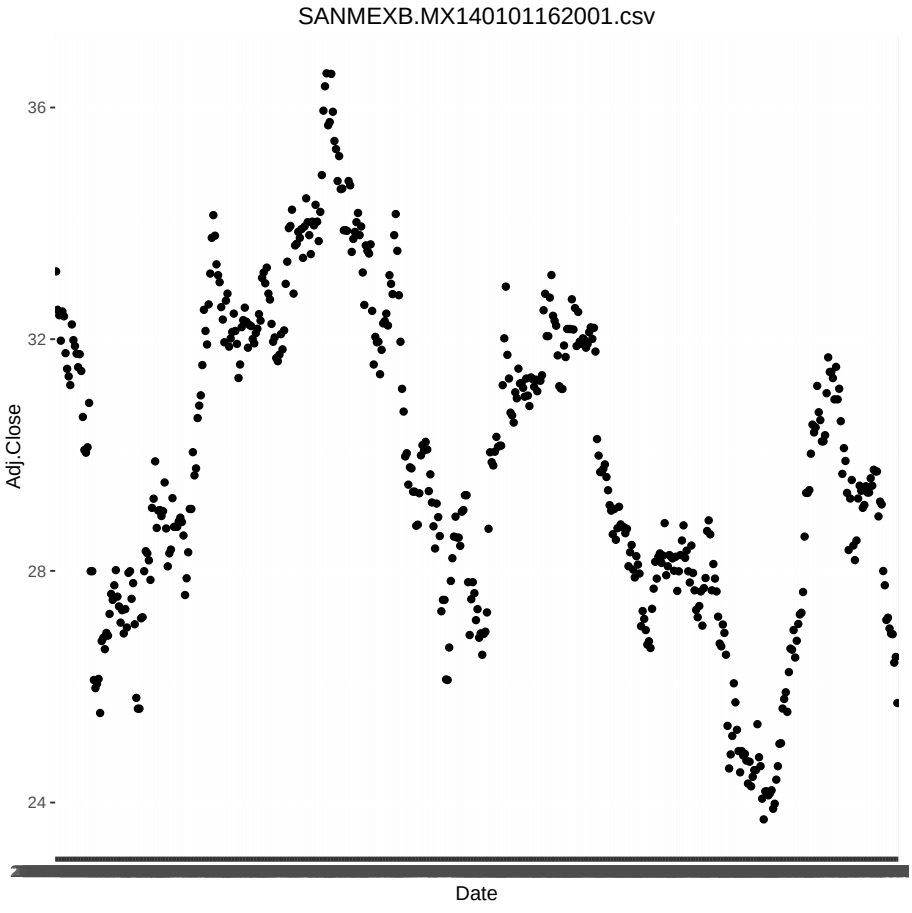

```
## BIMBOA.MX140101162001.csv
## H E A D
##         Date  Open  High   Low Close  Volume Adj.Close
## 1 2014-01-01 40.20 40.20 40.20 40.20       0   40.0094
## 2 2014-01-02 40.00 40.35 38.69 38.96 1465300   38.7753
## 3 2014-01-03 39.00 39.03 38.11 38.59 2105700   38.4070
## 4 2014-01-06 38.60 39.18 38.50 38.63 1915300   38.4468
## 5 2014-01-07 38.50 38.73 38.02 38.19 2398300   38.0089
## 6 2014-01-08 38.01 38.33 37.00 37.05 5156400   36.8743
## T A I L
```

```
##           Date  Open  High   Low Close  Volume Adj.Close
## 531 2016-01-13 46.74 47.96 46.65 47.46 3088900   47.2350
## 532 2016-01-14 47.60 47.99 46.00 47.44 2032100   47.2151
## 533 2016-01-15 47.50 47.51 46.11 47.18 2952400   46.9563
## 534 2016-01-18 47.00 47.13 46.50 46.54  622800   46.3193
## 535 2016-01-19 46.97 47.25 46.00 46.73 2249700   46.5084
## 536 2016-01-20 46.01 47.11 45.86 46.65 1766700   46.4288
```



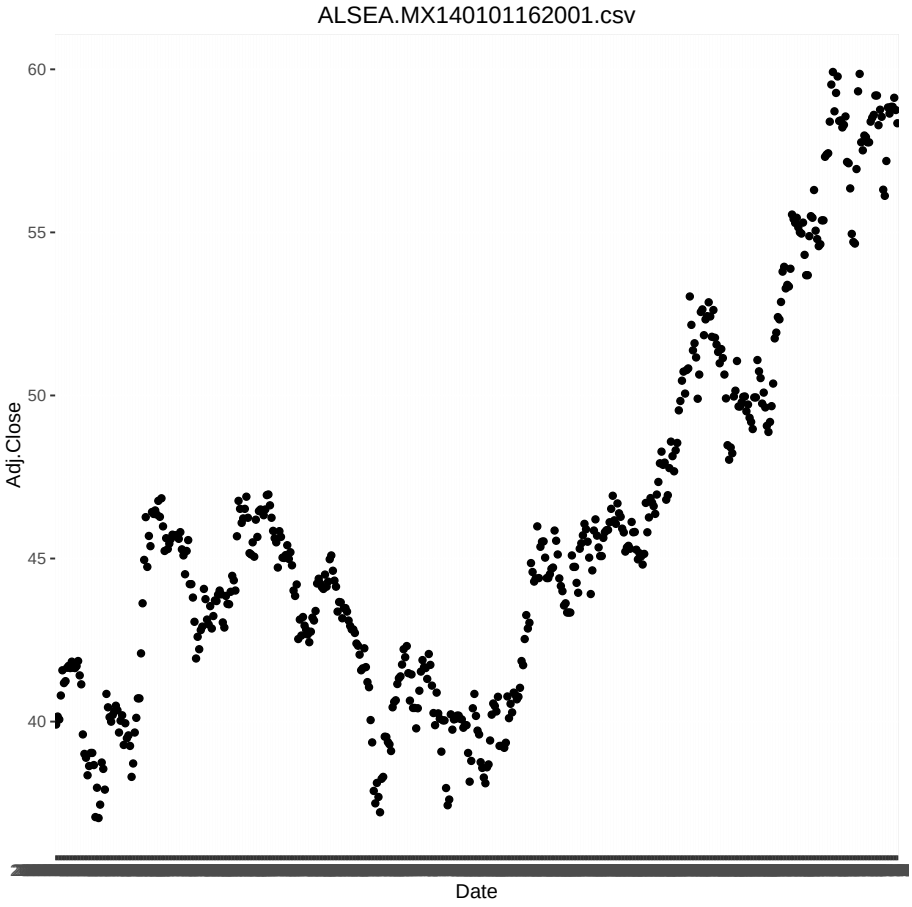

```
## SANMEXB.MX140101162001.csv
## H E A D
##         Date  Open  High   Low Close  Volume Adj.Close
## 1 2014-01-01 35.50 35.50 35.50 35.50       0  33.17082
## 2 2014-01-02 35.40 35.40 34.65 34.79 2721400  32.50741
## 3 2014-01-03 34.79 35.40 34.57 34.69 3194100  32.41397
## 4 2014-01-06 35.05 35.05 34.14 34.22 2225700  31.97481
```

```
## 5 2014-01-07 34.44 34.84 34.21 34.76 2444800  32.47937
## 6 2014-01-08 34.58 34.97 34.35 34.67 2141900  32.39528
## T A I L
##           Date  Open  High   Low Close  Volume Adj.Close
## 531 2016-01-13 27.90 28.05 27.21 27.49 2952900  27.00552
## 532 2016-01-14 28.00 28.00 27.05 27.40 1858700  26.91711
## 533 2016-01-15 27.30 27.50 26.95 27.39 6600900  26.90729
## 534 2016-01-18 27.09 27.34 26.73 26.89  657100  26.41610
## 535 2016-01-19 27.29 27.56 26.73 26.99 4032400  26.51434
## 536 2016-01-20 27.03 27.07 26.05 26.18 4674300  25.71861
```




```
## ALSEA.MX140101162001.csv
## H E A D
##         Date  Open  High   Low Close  Volume Adj.Close
## 1 2014-01-01 40.79 40.79 40.79 40.79       0  39.91006
```

```
## 2 2014-01-02 40.79 41.21 40.40 41.04 1704700  40.15466
## 3 2014-01-03 41.04 41.25 40.40 40.95  820300  40.06661
## 4 2014-01-06 41.57 41.90 41.18 41.70 1153800  40.80043
## 5 2014-01-07 41.70 42.50 41.67 42.49 1020800  41.57338
## 6 2014-01-08 42.41 42.60 41.83 42.09 1370600  41.18201
## T A I L
##           Date  Open  High   Low Close  Volume Adj.Close
## 531 2016-01-13 59.28 60.15 58.62 59.29 1890100  58.64372
## 532 2016-01-14 59.98 59.99 58.40 59.50 1279400  58.85143
## 533 2016-01-15 58.75 59.85 58.25 59.50 1708900  58.85143
## 534 2016-01-18 59.20 59.85 59.13 59.78  244500  59.12838
## 535 2016-01-19 59.85 60.50 58.98 59.40 1505300  58.75252
## 536 2016-01-20 59.00 59.40 58.75 58.99 1615400  58.34699
```



6.2. Código de algorítmo PSO: generalizado

```
# -*- coding: utf-8 -*-

'''

Optimizaci\'{ o}n por Enjambre de Part\'{ i}culas

Particle Swatm Optimization (PSO)



'''

import numpy as np


def algo_pso(npart, c1, c2, func, nvar, desc):

    ini = float("inf")
    desc = (-1) * desc

    # numero de interaciones de b\'{ u}squeda
    niter = 3000

    # inicializaci\'{ o}n del enjambre
    prtl = []
    for i in range(nvar):
        prtl.append(np.random.rand(npart))

    # Mejores locales iniciales y desempe\~{n}o
    prtl_ml = prtl
    fpl     = ini * np.ones(npart)


    # Mejor global inicial y desempe\~{n}o
    prtl_mg = [0] * nvar
    fpg = ini;
    lfpg = []

    # velocidad
    prtl_v = []
    for i in range(nvar):
        prtl_v.append(np.zeros(npart))
```

```
for k in range(niter):

    # funci\'{ o}n para calcular desempe\~{n}o del enjambre
    fp = desc * func(prtl)

    # Encontrar mejor global (m\'{ i}nimo)
    index = np.argmin(fp)

    # comparaci\'{ o}n para determinar m\'{ i}nimo global
    if fp[index] < fpg:
        fpg = fp[index]
        for j in range(nvar):
            prtl_mg[j] = prtl[j][index]

    # encontrar mejores locales
    for ind in range(npart):
        if fp[ind] < fpl[ind]:
            fpl[ind] = fp[ind]
            for j in range(nvar):
                prtl_ml[j][ind] = prtl[j][ind]

    # movimiento de part\'{ i}culas
    for j in range(nvar):
        prtl_v[j] = prtl_v[j] + c1 * np.random.rand(npart) * (prtl_mg[j] - prtl[j]) + c2
        prtl[j]   = prtl[j] + prtl_v[j]
    lfpg.append(fpg)



fp = func(prtl)
'''
import matplotlib.pyplot as plt
plt.plot(lfpg)
'''

return prtl_mg, desc * fpg, fp, prtl
```

6.3. app_2v1.py Código de algoritmo genético: promedio móvil

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 20 21:36:24 2016

@author: Rodrigo


General procedure:

Genetic_algorithm
    optimize: function h(f,x)
    f is the vector or 'time intervals'
    x is the vector of weights


"""

print('\nReading dependencies... \n\n')
import numpy as np
import matplotlib.pyplot as plt
import gen_fun as fu
import findata as fd

#%%
# Definici\'{ o}n de la funci\'{ o}n de inicializaci\'{ o}n
def initpob(npob,nbits,xmin,xmax):
    # xint = np.round(((2**nbits)-1)*np.random.rand(npob,1));
    #
    # npob = numero de pobladores
    # nbits = numero de bits del codigo genetico
    # xmin = limite inferior del espacio de busqueda
    # xmax = limite superior del espacio de busqueda
    xint = np.random.randint(2**nbits,size=(npob,1))
    xpob = ((xmax-xmin)/(np.double(2**nbits)-1))*xint+xmin;
    return xpob,xint

#%%
# Definici\'{ o}n de la funci\'{ o}n de selecci\'{ o}n
def selecbest(npadres,xpob,xint,ypob,minmax):
    # Ordenar los pobladores seg\'{ u}n su desempe\~{n}o.
    #
```

```
    # npadres = numero de padres que se quieren salvar
    # xpob = pobladores en el espacio de busqueda
    # xint = cromosoma en decimal
    # ypob = desempe\~{n}o de los pobladores
    # minmax = 1 si se quiere maximizar y -1 si se quiere minimizar
    npob,ntemp = np.shape(xpob);
    temp = np.zeros((npob,3));
    temp[:,0] = ypob#ypob[:,0];
    temp[:,1] = xpob[:,0];
    temp[:,2] = xint[:,0];
    temp = temp[temp[:,0].argsort(),:];
    if minmax == 1:
        xpadres=temp[npob-npadres:npob,1];
        xintpadres=temp[npob-npadres:npob,2];
    elif minmax == -1:
        xpadres=temp[0:npadres,1];
        xintpadres=temp[0:npadres,2];
    else:
        print('No se eligio un argumento valido para minmax')
        return -1,1
    xpadres = xpadres.reshape((npadres,1));
    xintpadres = xintpadres.reshape((npadres,1));
    return xpadres,xintpadres

#%% M\'{ e}todos de cruzamiento

# Un punto cruce
def un_pc(ind,cromx,cromy,nbits):
    crombit = np.random.randint(nbits-1)+1;
    if ind%2 == 0:
        cromhijo = cromx[0:crombit]+cromy[crombit:nbits];
    else:
        cromhijo = cromy[crombit:nbits]+cromx[0:crombit];
    return cromhijo

# Doble punto cruce
def do_pc(ind,cromx,cromy,nbits):
    i = np.random.randint(nbits//2-1)+1;
    j = nbits - (np.random.randint(nbits//2-2)+1);

    if ind%2 == 0:
        cromhijo = cromx[0:i]+cromy[i:j]+cromx[j:nbits]
    else:
        cromhijo = cromy[0:i]+cromx[i:j]+cromy[j:nbits]
    return cromhijo
```

```
# Cruzamiento uniforme
def cr_un(ind,cromx,cromy,nbits):
    cromhijo = ''
    for i in range(nbits):
        if np.random.rand(1) > 0.5:
            cromhijo = cromhijo + cromx[i]
        else:
            cromhijo = cromhijo + cromy[i]
    return cromhijo

# Cruzamiento aritm\'{ e}tico
def cr_ar(ind,cromx,cromy,nbits):
    x = int(cromx,2)
    y = int(cromy,2)
    z = x + y
    zbin = np.binary_repr(z,nbits)
    cromhijo = zbin[0:nbits]
    return cromhijo


#%%
#Definici\'{ o}n de la funci\'{ o}n de cruzamiento
def cruzamiento(xint,nhijo,nbits,xmin,xmax,met_cruz):
    # Realizacion de nhijo numero de pobladores nuevos a partir de xint
    #
    # xint = cromosoma en decimal
    # nhijo = numero de hijos que se quieren generar
    # nbits = numero de bits del cromosoma de los pobladores
    # xmin = limite inferior del espacio de busqueda
    # xmax = limite superior del espacio de busqueda

    npadre,ntemp = np.shape(xint);
    xinthijo=np.zeros((nhijo,1));
    xhijo=xinthijo;
    if npadre==1:
        print("Precuaci\'{ o}n: Con un solo padre no se tiene recombinaci\'{ o}n de genes")
    px = np.tile(np.arange(0,npadre),nhijo//npadre+1);

    for ind in range(0,nhijo):
        cromx = np.binary_repr(np.int(xint[px[ind],0]),width=nbits);
        cromy = np.binary_repr(np.int(xint[px[ind+1],0]),width=nbits);

        if met_cruz == 1:
            # Metodo 1 para cruzamiento
            cromhijo = un_pc(ind,cromx,cromy,nbits)
        elif met_cruz == 2:
```

```
            #Metodo 2 para cruzamiento
            cromhijo = do_pc(ind,cromx,cromy,nbits)
        elif met_cruz == 3:
            #Metodo 3 para cruzamiento
            cromhijo = cr_un(ind,cromx,cromy,nbits)
        elif met_cruz == 4:
            #Metodo 4 para cruzamiento
            cromhijo = cr_ar(ind,cromx,cromy,nbits)

        xinthijo[ind,0]=int(cromhijo,2);
        xhijo[ind,0] = ((xmax-xmin)/(np.double(2**nbits)-1))*xinthijo[ind,0]+xmin;
    return xhijo,xinthijo

#%%
#Definici\'{ o}n de la funci\'{ o}n de cruzamiento
def mutacion(xint,nmut,nbits,xmin,xmax):
    # Realizacion de nmut numero de mutaciones en los pobladores xint
    #
    # xint = cromosoma en decimal
    # nmut = numero de mutantes que se queiren generar
    # nbits = numero de bits del cromosoma de los pobladores
    # xmin = limite inferior del espacio de busqueda
    # xmax = limite superior del espacio de busqueda
    nhijo,ntemp = np.shape(xint);
    for ind in range(0,nmut):
        nhijmut = np.random.randint(nhijo);
        nbitmut = np.random.randint(nbits);
        crom = np.binary_repr(np.int(xint[nhijmut,0]),width=nbits);
        if crom[nbitmut] == '1':
            crom = crom[0:nbitmut]+'0'+crom[nbitmut+1:nbits];
        else:
            crom = crom[0:nbitmut]+'1'+crom[nbitmut+1:nbits];
        xint[nhijmut,0]=int(crom,2);
    xmut = ((xmax-xmin)/(np.double(2**nbits)-1))*xint+xmin;
    return xmut,xint

# %%

def gen_algo(xmin, delta, nbits, npob, npadres, nvar, func, desc, met_cruz, price, returns):
    '''
    La funci\'{ o}n gen_algo(...) aplica la metodolog\'{ i}a de optimizaci\'{ o}n mediante
    algor\'{ i}tmos gen\'{ e}ticos.

    I N P U T:
    - xmin:
    - delta:
```

```
- nbits:
- npob:
- npadres:
- nvar:
- func:
- desc:
- met_cruz:

O U T P U T
-
'''

# Se determina el l\'{ i}mite superior de la regi\'{ o}n
#xmax = xmin+delta;

# Se determina el n\'{ u}mero de hijos
nhijos  = npob - npadres;

# N\'{ u}mero de iteraciones e inicializaci\'{ o}n del promedio.
niter = 50;
yprom = np.zeros(niter);

# Inicio de poblaci\'{ o}n en varialbe pobl
# caracter\'{ i}sticas: pobl es una lista que contiene listas.
# ejemplo: pobl[i][j] accede a la lista j de la variable i+1 en donde
# j solamente puede adquirir valores de {0, 1}
# e.g. pobl[0][0] regresa el valor real de la variable 1.
# e.g. pobl[0][1] regresa el valor entero de la variable 1.
'''
Para esta aplicaci\'{ o}n es necesario generar 6 fechas y 6 ponderaciones
'''
pobl1 = []
pobl2 = []
for i in range(0,int(nvar/2)):
    pobl1.append(initpob(npob,7,1,128))
    pobl2.append(initpob(npob,10,0,1))

pobl = pobl1+pobl2

# Ciclo para comenzar desarrollo y selecci\'{ o}n de la poblaci\'{ o}n
for ind in range(0,niter):
    # Evaluaci\'{ o}n del 'performance' de acuerdo al criterio func
    yp = func(pobl, price, returns);
    yprom[ind]=np.mean(yp)

    # Selecci\'{ o}n de los individuos m\'{ a}s aptos (futuros padres)
```

```
        padr=[]
        for i in range(0,nvar):
            padr.append(selecbest(npadres,pobl[i][0],pobl[i][1],yp,desc))

        # Recombinzaci\'{ o}n gen\'{ e}tica de los individuos con mejor desepe\~{n}o.
        # (generaci\'{ o}n de hijos)
        hijs1 = []
        hijs2 = []
        nhijos = npob - npadres#int((npob - npadres)/2)
        for i in range(0,int(nvar/2)):
            #
            hijs1.append(cruzamiento(padr[i][1],nhijos,7,1,128,3))
            hijs2.append(cruzamiento(padr[i][1],nhijos,11,0,1,3))

        hijs = hijs1+hijs2

        # Mutaci\'{ o}n aleatoria cada 'n' generaciones
        if ind%10==0:
            hijs1 = []
            hijs2 = []
            for i in range(0,int(nvar/2)):
                hijs1.append(mutacion(hijs[i][1],4,7,1,128))
                hijs2.append(mutacion(hijs[i][1],4,11,0,1))
            hijs = hijs1+hijs2

        # Eliminar a la poblaci\'{ o}n con bajo desempe\~{n}o.
        # Sobreescribir 'padres' e 'hijos'.
        pobl = []
        for i in range(0,nvar):
            aux0 = np.concatenate((padr[i][0],hijs[i][0]))
            aux1 = np.concatenate((padr[i][1],hijs[i][1]))
            pobl.append([aux0,aux1])

        print('Evaluating: {} %'.format(100*(ind+1) /niter))

    yp = func(pobl, price, returns);
    return pobl, yp, yprom

# %%

def func(pobl, price, returns):
    x = pobl
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    a,b,c = np.shape(X)
```

```
    X = np.reshape(X,(a,b))
    mX = np.asmatrix(X)


    #for i in range(int(np.shape(X)[0]/2)):
    #    X[i] = [int(j) for j in X[i]]
    br = int(np.shape(X)[0]/2)
    for i in range(mX.shape[1]):
        z = fu.prom_mov2(mX[br:, ].T,mX[:br, ].T,price, returns,1)


    return z


# %%Number assets for portfolio

nacci = 6
nvar = nacci*2

# %% Descarga de datos
acc = ["GFNORTEO.MX","LIVEPOLC-1.MX","HERDEZ.MX","BIMBOA.MX", "SANMEXB.MX", "ALSEA.MX"]
#n_acc, temp = np.shape(X)
acc = acc[:nvar]# acc[:n_acc]
price, returns = fd.download(acc)

xmin = 1; xmax = 128; delta= xmin-xmax; nbits=7;
npob = 40; npadres = 12;
desc = -1
print('Running genetic algorithm optimization... \n\n')
pobl, yp, yprom = gen_algo(xmin, delta, nbits, npob, npadres, nvar, func, desc, 3, price, re

print('\nShowing Results: ')



#%%
# BEFORE PRINTING THE RESULTS...
results = {}
c = 0
aux = []
for j in range(nvar):
    aux.append(pobl[j][0])
for i in yp:
    aux2 = []
    for k in range(nvar):
        aux2.append(aux[k][c])
```

```
    results[np.double(i)] = aux2
    c += 1

print('With an xmin of :',0, 'and a delta of: ', 1, 'and a window from:',1,' days to:',128,'

# PRINT RESULTS
if desc == 1:
    valu = max(results.keys())
    minmax = 'max.'
else:
    valu = min(results.keys())
    minmax = 'min'

indx = list(results.keys()).index(valu)
xval = list(results.values())[indx]

rs = '\n'
for i in range(len(xval)):
    aux = '    x'+str(i)+' = '+str(xval[i][0])+'\n'
    rs = rs + aux

print('The',minmax,'value found was:', valu,'in the region {x0,x1,...,xn} : \n',rs)

# PLOT RESUTLS (Average)
plt.plot(yprom)
plt.title('Average performance per generation')
plt.xlabel('Number of generations')
plt.ylabel('Average performance (function evaluated)')
plt.show()


# %%
'''
xval has te answer...

'''

print('\n\nResults in context: \n')

br = int(nvar/2)
mx = np.matrix(xval)
r, s = fu.prom_mov3g(mx[br:, ].T,mx[:br, ].T,price, returns,0)

print('\n\nReturn: {}'.format(r))
print('Standard Deviation: {}'.format(s))
```

```
print('\n sum: {}'.format(sum(mx[br:, ])))
```

6.4. Código para descarga de datos

```
# -*- coding: utf-8 -*-

'''

script to download financial data...


'''

import numpy as np
import pandas as pd
import pandas.io.data as wb
import matplotlib.pyplot as plt
import datetime as dt
import os.path

def download(acc):
    '''
    function to automate the process of downloading financial data from yahoo
    I N P U T S
    acc = list of strings containing the stock names.

    '''

    # define starting and ending time
    start = dt.datetime(2014,1,1)
    end = dt.datetime(2016,1,20) #dt.datetime.now()

    # determine id...
    '''
    lid = []
    lid.append(dt.date.today())
    fid = str(lid[0])
    '''
    lid1 = []; lid2 = []
    lid1.append(start)
    lid2.append(end)
    fid = lid1[0].strftime('%y%d%m') + lid2[0].strftime('%y%d%m')
```

```
    req_do = 0
    stocks = []
    for i in acc:
        if os.path.isfile(i+fid+'.csv'):
            stocks.append(pd.read_csv(i+fid+'.csv'))
        else:
            req_do = 1
    if req_do == 1:
        # read data...
        stocks = [wb.DataReader(i,'yahoo',start,end) for i in acc]
        # save data...
        j = 0
        for i in acc:
            stocks[j].to_csv(i+fid+'.csv')
            j += 1

    # get prices and calc returns
    price = pd.DataFrame()
    returns = pd.DataFrame()
    j = 0
    for i in acc:
        price[i] = stocks[j]['Adj Close']
        returns[i] = (price[i] / price[i].shift(1) - 1)[1:]
        j += 1

    return price, returns

def parameters(price, returns):
    '''
    function that calculates the parameters required  construct a portfolio.
    '''
    # mean and covar. of each stock
    returns = np.matrix(returns)
    rst = np.mean(returns, 0)
    cst = np.matrix(np.cov(returns, rowvar=False))

    # participation of stocks in portfolio (for markowitz)
    npart = 5000
    ntemp, nst = np.shape(returns)
    part = np.random.rand(npart,nst)
    spart = np.sum(part, 1)
    for i in range(nst):
        part[:,i] = part[:,i] / spart;

    return part, rst, cst
```

```
def markowitz(part, rst, cst):
    '''
    function that applies the markowitz portfolio simulation
    '''
    part = np.matrix(part)

    # determines the returns...
    rp = part * np.transpose(rst)

    # determines the risk (var)
    riskp = part*cst*np.transpose(part)
    n,t = np.shape(part)
    te = np.arange(0,n)
    riskp = np.transpose(riskp[te,te])


    return np.array(rp), np.array(riskp)

def graf_prec(price,returns, acc):
    import matplotlib
    matplotlib.style.use('ggplot')
    plt.figure
    price.plot.line()
    plt.title('Valor de activos; general')
    plt.ylabel('Valor')
    plt.show()
    for i in acc:
        plt.figure
        plt.subplot(3,1,1)
        price[i].plot.line()
        plt.title('Valor del activo: {}'.format(i))
        plt.ylabel('Valor')
        plt.subplot(3,1,3)
        plt.bar(returns[i].index,returns[i].values, color = 'black')
        plt.title('\nRendimiento')
        plt.tick_params(axis='x',which='both',bottom='off',top='off',labelbottom='off')
        plt.show()
    return 0

'''
# Example
acc = ["GFNORTEO.MX","LIVEPOLC-1.MX","HERDEZ.MX","BIMBOA.MX", "SANMEXB.MX", "ALSEA.MX"]
price, returns = download(acc)
graf_prec(price,returns, acc)
part, rst, cst = parameters(price, returns)
rp, riskp = markowitz(part, rst, cst)
```

```
plt.plot(riskp, rp, 'b.')
plt.title('Markowitz simulation')
plt.xlabel('Risk (sd)')
plt.ylabel('Returns')
plt.show()
'''
```

6.5. Código de funciones generales

```
# -*- coding: utf-8 -*-
'''
@author: Rodrigo Hern\'{ a}ndez Mota
'''

import numpy as np
import findata as fd

def f4(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x

    xm = np.matrix(X)
    xm = xm.T

    # costo
    c = np.matrix([170, 160, 175, 180, 195])
    c = c.T

    # funci\'{ o}n de gasto
    z = xm.dot(c)

    # reestric.
    nn = xm.shape[0]
    r = np.zeros((nn,10))
    alpha = 10000

    if len(a) == 1:
        # 0
        index = (-X[0] + 48 > 0)
```

```
        r[:,0] = alpha * index * (-X[0] + 48)
        # 1
        index = (-X[1] - X[0] + 79 > 0)
        r[:,1] = alpha * index * (-X[1] - X[0] + 79)
        # 2
        index = (65 - X[0] - X[1] > 0)
        r[:,2] = alpha * index * (65 - X[0] - X[1])
        #3
        index = (87 - X[0] - X[1] - X[2] > 0)
        r[:,3] = alpha * index * (87 - X[0] - X[1] - X[2])
        # 4
        index = (64 - X[1] - X[2] > 0)
        r[:,4] = alpha * index * (64 - X[1] - X[2])
        # 5
        index = (73 - X[2] - X[3] > 0)
        r[:,5] = alpha * index * (73 - X[2] - X[3])
        #6
        index = (82 - X[2] - X[3] > 0)
        r[:,6] = alpha * index * (82 - X[2] - X[3])
        #7
        index = (43 - X[3] > 0)
        r[:,7] = alpha * index * (43 - X[3])
        #8
        index = (52 - X[3] - X[4] > 0)
        r[:,8] = alpha * index * (52 - X[3] - X[4])
        #9
        index = (15 - X[4] > 0)
        r[:,9] = alpha * index * (15 - X[4])

    re = np.sum(r, axis = 1)
    z = np.array(z.T) + re
    return z[0]

# Funci\'{ o}n ej3 (2 variables)
def f3(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x

    # comienza fun
    z = []

    s = np.size(X[0])
    rest1 = np.zeros(s)
    rest2 = np.zeros(s)
```

```
    for i,j in zip(X[0], X[1]):
        if i+j >= 1:
            z.append(i**2 + j**2)
        else:
            z.append(-10*(i+j)+20)

    a1 = 10000
    a2 = 10000

    if len(a) == 1:
        index = (-X[0]-1>= 0)
        rest1[index] = -X[0][index]-1
        index = (-X[1]-1>= 0)
        rest2[index] = -X[1][index]-1



    return z + a1 * rest1 + a2 * rest2
```

```
# Funci\'{ o}n y = x^1
def func_1v(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    # evaluate and return
    return X[0] ** 2

# Funci\'{ o}n y = x1^2 + x2^2
def func_2v(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    # evaluate and return
    return X[0] ** 2 + X[1] ** 2

# Funci\'{ o}n  y = -x1*(10+100*x1) - x2*(5+40*x1) - x3*(5+50*x3)
def func1_3v(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
```

```
    else: X = x
    # evaluate and return
    return -X[0]*(10+100*X[0]) - X[1]*(5+40*X[1]) - X[2]*(5+50*X[2]);

# Funci\'{ o}n y = ...
def func2_3v(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    # evaluate and return
    u = X[0]**2-2*X[0]+1-10*np.cos(X[0]-1)+X[1]**2+X[1]+\
    0.25-10*np.cos(X[0]+0.5)+X[2]**2-10*np.cos(X[2])
    return u

# Funci\'{ o}n de Rast...
def func_rast(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    # evaluate and return
    for i in range(len(X)):
        u = 10+ X[i]**2-10*np.cos(5*X[i])
    return u

# Funci\'{ o}n de Ackley
def func_ackley(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    # evaluate and return
    for i in range(len(X)):
        u = -10*np.exp(-np.sqrt(X[i]**2)) - np.exp(np.cos(5*X[i]))
    return u

# Funci\'{ o}n de Rosen
def func_rosen(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    # evaluate and return
    for i in range(len(X)-1):
        u = 100 * (X[i+1]-X[i])**2+(1-X[i+1])**2
```

```
    return u

# Markowitz function
def markowitz_1(X):
    '''
    function that applies the markowitz portfolio simulation
    '''

    # download data and calc. parameters
    import findata as fd
    n_acc, temp = np.shape(X)
    acc = ["GFNORTEO.MX","LIVEPOLC-1.MX","HERDEZ.MX", "BIMBOA.MX", "SANMEXB.MX", "ALSEA.MX"]
    acc = acc[:n_acc]
    price, returns = fd.download(acc)
    pa, rst, cst = fd.parameters(price, returns)

    '''
    # conditional for genetic algo...
    aa = np.shape(X[0])
    if len(aa) > 1:
        aux = X[0]
        for i in range(1,len(X)):
            aux = np.hstack([aux,X[i]])
        X = np.transpose(aux)
    '''

    # use part as matrix
    X = np.transpose(np.matrix(X))

    # determines the returns...
    rp = X * np.transpose(rst)

    # determines the risk (var)
    riskp = X*cst*np.transpose(X)
    n,t = np.shape(X)
    te = np.arange(0,n)
    riskp = np.transpose(riskp[te,te])


    return np.array(rp), np.array(riskp), np.array(X.T), pa

def markowitz_2(rp, riskp, X):
    '''
    This function takes the two results of markowitz and...

    import matplotlib.pyplot as plt
```

```
    plt.plot(riskp, rp, 'b.')
    plt.title('Markowitz simulation')
    plt.xlabel('Risk (sd)')
    plt.ylabel('Returns')
    plt.show()
    '''

    b1 = 10000
    b2 = 100000
    alpha1 = 1000
    alpha2 = 50
    z = -b1*rp + b2*riskp

    i,j = X.shape
    rest = np.zeros(j)


    for act in X:
        # x > 0
        rest = rest + alpha1 * np.abs(act) * (act < 0)
        # x < 1
        rest = rest + alpha1 * np.abs(act) * (act > 1)


    X2 = np.transpose(np.matrix(X))
    X2 = np.array(X2)

    aux = []
    for act2 in X2:
        # sum x == 1
        aux.append(alpha2 * np.abs(np.sum(act2) - 1))
    rest = rest + np.array(aux)

    z = z.T + rest

    return z[0]

def markowitz(x):
    # determine if x is data for gen.algo (==2) or gen.pso
    '''
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    '''
    X = x
```

```
    rp, riskp, X, pa = markowitz_1(X)
    z = markowitz_2(rp, riskp, X)

    return z

def simtrading_prom(data,nmovil,cash0,accion0,com):
    #nmovil = 10; #numero de dias del promedio movil
    #cash0 = 10000; #dinero disponible para comprar acciones
    #accion0 = 0; #numero de acciones disponibles para vender
    #com = 0.0029; # porcentaje de cobro de comision por operacion
    ndata,temp = np.shape(data); #numero de datos disponibles para simular
    promovil = np.zeros((ndata,1)); # iniciar el vector donde se guardara el promedio movil
    numaccion = np.ones((ndata+1,1))*accion0; # iniciar el vector donde se guardara el numer
    cash = np.ones((ndata+1,1))*cash0; # iniciar el vector donde se guardara el numero de ac
    balance = np.ones((ndata+1,1))*(cash+accion0*data[nmovil-1,1]);

    for k in range(np.int(nmovil),np.int(ndata)):
        #calculo del promedio movil
        promovil[k,0] = np.mean(data[k-nmovil:k,1]);

        # simulacion de compra y venta
        if data[k,1]>=promovil[k,0]:
            #compra
            temp = np.floor(cash[k,0]/(data[k,1]*(1+com))); #acciones para que me alcanzan
            numaccion[k+1,0] = numaccion[k,0]+temp; # actualizo el numero de acciones
            cash[k+1,0] = cash[k,0]-temp*data[k,1]*(1+com); #actualizo el cash
            balance[k+1,0] = cash[k+1,0]+numaccion[k+1,0]*data[k,1];
        else:
            #vende
            numaccion[k+1,0] = 0;
            cash[k+1,0] = cash[k,0]+numaccion[k,0]*data[k,1]*(1-com);
            balance[k+1,0] = cash[k+1,0]+numaccion[k+1,0]*data[k,1];

    # La funcion regresa el promedio movil, el balance de la cuenta simulada,
    # el comportamiento del cash de la cuenta y el comportamiento de las acciones
    return promovil,balance,cash,numaccion

def prom_mov(x,f,price, returns, re):
    '''
    re = {0, 1}, apply resstrictions (1), or not (0)
    '''
    X = x
    cash_tot = 1000000

    pa, rst, cst = fd.parameters(price, returns)
```

```
X = np.transpose(np.matrix(x))

#Uso de la funcion
#nmovil = v; #numero de dias del promedio movil
accion0 = 0; #numero de acciones disponibles para vender
com = 0.0029; # porcentaje de cobro de comision por operacion
rport = np.array([])
sigma_port = np.array([])
for i in range(np.shape(x)[1]):
    r_ind = []
    sigma_ind = []

    for j in range(np.shape(x)[0]):
        nmovil = f[j]
        pricegrum = price.values[:,j];
        ndata = np.size(pricegrum);
        data = np.reshape(pricegrum,(ndata,1));
        data = np.append(np.reshape(np.arange(0,ndata),(ndata,1)),data,axis=1);


        cash0 = cash_tot * x[j][i] #dinero disponible para comprar acciones
        promovil,balance,cash,numaccion = simtrading_prom(data,nmovil,cash0,accion0,com)

        #calculo del rendimiento promedio del balance final
        rend = (balance[nmovil+1:ndata]/balance[nmovil:ndata-1])-1;
        r_ind.append(np.mean(rend))
        sigma_ind.append(np.std(rend))

    x_reng = X[i,:]
    r_reng = np.matrix(r_ind).T
    sigma_reng = np.matrix(sigma_ind).T
    sigma_reng = np.power(sigma_reng,2)
    x_reng2 = np.power(x_reng,2)

    rport = np.append(rport, np.array(x_reng.dot(r_reng)))
    sigma_port = np.append(sigma_port, np.array(x_reng2.dot(sigma_reng)))

X = np.array(X.T)

# reestric
if re == 1:

    b1 = 10000
    b2 = 100000
    alpha1 = 1000
    alpha2 = 50
```

```
        z = -b1*rport + b2*sigma_port

        i,j = X.shape
        rest = np.zeros(j)


        for act in X:
            # x > 0
            rest = rest + alpha1 * np.abs(act) * (act < 0)
            # x < 1
            rest = rest + alpha1 * np.abs(act) * (act > 1)


        X2 = np.transpose(np.matrix(X))
        X2 = np.array(X2)

        aux = []
        for act2 in X2:
            # sum x == 1
            aux.append(alpha2 * np.abs(np.sum(act2) - 1))
        rest = rest + np.array(aux)

        z = z.T + rest
    else:
        z = [rport, sigma_port]

    return z

def port_val(x,f, price, returns):

    cash_tot = 1000000
    pa, rst, cst = fd.parameters(price, returns)
    #Uso de la funcion
    #nmovil = v; #numero de dias del promedio movil
    accion0 = 0; #numero de acciones disponibles para vender
    com = 0.0029; # porcentaje de cobro de comision por operacion
    rport = np.array([])
    sigma_port = np.array([])
    r_ind = []
    sigma_ind = []
    for i in range(len(f)):
        nmovil = f[i]
        pricegrum = price.values[:,i];
        ndata = np.size(pricegrum);
```

```
        data = np.reshape(pricegrum,(ndata,1));
        data = np.append(np.reshape(np.arange(0,ndata),(ndata,1)),data,axis=1);


        cash0 = cash_tot * x[i] #dinero disponible para comprar acciones
        promovil,balance,cash,numaccion = simtrading_prom(data,nmovil,cash0,accion0,com);

        #calculo del rendimiento promedio del balance final
        rend = (balance[nmovil+1:ndata]/balance[nmovil:ndata-1])-1;
        r_ind.append(np.mean(rend))
        sigma_ind.append(np.std(rend))

    x_reng = np.asmatrix(x)
    r_reng = np.matrix(r_ind).T
    sigma_reng = np.matrix(sigma_ind).T
    sigma_reng = np.power(sigma_reng,2)
    x_reng2 = np.power(x_reng,2)

    rport = np.append(rport, np.array(x_reng.dot(r_reng)))
    sigma_port = np.append(sigma_port, np.array(x_reng2.dot(sigma_reng)))


    z = [rport, sigma_port]

    return z

def prom_mov2(x,f,price, returns, re):
    '''
    re = {0, 1}, apply resstrictions (1), or not (0)
    '''
    X = x
    cash_tot = 1000000

    pa, rst, cst = fd.parameters(price, returns)

    #X = np.transpose(np.matrix(x))

    #Uso de la funcion
    #nmovil = v; #numero de dias del promedio movil
    accion0 = 0; #numero de acciones disponibles para vender
    com = 0.0029; # porcentaje de cobro de comision por operacion
    rport = np.array([])
    sigma_port = np.array([])
    for i in range(np.shape(x)[0]):
        r_ind = []
        sigma_ind = []
```

```
        for j in range(np.shape(x)[1]):
            nmovil = np.int(f[i,j])
            pricegrum = price.values[:,j];
            ndata = np.size(pricegrum);
            data = np.reshape(pricegrum,(ndata,1));
            data = np.append(np.reshape(np.arange(0,ndata),(ndata,1)),data,axis=1);


            cash0 = cash_tot * x[i,j] #dinero disponible para comprar acciones
            promovil,balance,cash,numaccion = simtrading_prom(data,nmovil,cash0,accion0,com)

            #calculo del rendimiento promedio del balance final
            rend = (balance[nmovil+1:ndata]/balance[nmovil:ndata-1])-1;
            r_ind.append(np.mean(rend))
            sigma_ind.append(np.std(rend))

        x_reng = X[i,:]
        r_reng = np.matrix(r_ind).T
        sigma_reng = np.matrix(sigma_ind).T
        sigma_reng = np.power(sigma_reng,2)
        x_reng2 = np.power(x_reng,2)

        rport = np.append(rport, np.array(x_reng.dot(r_reng)))
        sigma_port = np.append(sigma_port, np.array(x_reng2.dot(sigma_reng)))

    X = np.array(X.T)

    # reestric
    if re == 1:

        b1 = 10000
        b2 = 100000
        alpha1 = 1000
        alpha2 = 1000


        z = -b1*rport + b2*sigma_port

        i,j = X.shape
        rest = np.zeros(j)


        for act in X:
            # x > 0
            rest = rest + alpha1 * np.abs(act) * (act < 0)
```

```
            # x < 1
            rest = rest + alpha1 * np.abs(act) * (act > 1)


        X2 = np.transpose(np.matrix(X))
        X2 = np.array(X2)

        aux = []
        for act2 in X2:
            # sum x == 1
            aux.append(alpha2 * np.abs(np.sum(act2) - 1))
        rest = rest + np.array(aux)

        z = z.T + rest
    else:
        z = [rport, sigma_port]

    return z


def prom_mov3g(x,f,price, returns, re):
    '''
    re = {0, 1}, apply resstrictions (1), or not (0)
    '''
    X = x
    cash_tot = 1000000

    pa, rst, cst = fd.parameters(price, returns)

    #X = np.transpose(np.matrix(x))

    #Uso de la funcion
    #nmovil = v; #numero de dias del promedio movil
    accion0 = 0; #numero de acciones disponibles para vender
    com = 0.0029; # porcentaje de cobro de comision por operacion
    rport = np.array([])
    sigma_port = np.array([])
    for i in range(np.shape(x)[0]):
        r_ind = []
        sigma_ind = []

        for j in range(np.shape(x)[1]):
            nmovil = np.int(f[i,j])
            pricegrum = price.values[:,j];
            ndata = np.size(pricegrum);
            data = np.reshape(pricegrum,(ndata,1));
```

```
        data = np.append(np.reshape(np.arange(0,ndata),(ndata,1)),data,axis=1);


        cash0 = cash_tot * x[i,j] #dinero disponible para comprar acciones
        promovil,balance,cash,numaccion = simtrading_prom(data,nmovil,cash0,accion0,com)
        rend = (balance[nmovil+1:ndata]/balance[nmovil:ndata-1])-1;
        rendm = np.mean(rend);
        riskm = np.std(rend);
        rendf = (balance[-1,0]/cash0)-1;
        print('Rendm = %.4f, Riskm = %.4f, Rendf = %.4f' % (rendm*100,riskm*100,rendf*10

        # graf
        ndata,temp = np.shape(data);
        t = np.reshape(np.arange(0,ndata),(ndata,1));
        t1 = np.reshape(np.arange(0,ndata+1),(ndata+1,1));
        import matplotlib.pyplot as plt
        plt.figure(1);
        plt.subplot(3,1,1);
        plt.plot(data[nmovil:,0],data[nmovil:,1],'b-',t[nmovil:,0],promovil[nmovil:,0],'
        plt.ylabel('precio');
        plt.grid(color='k', linestyle='--');
        plt.subplot(3,1,2);
        plt.plot(t1[nmovil:,0],numaccion[nmovil:,0],'b-');
        plt.ylabel('acciones');
        plt.grid(color='k', linestyle='--');
        plt.subplot(3,1,3);
        plt.plot(t1[nmovil:,0],balance[nmovil:,0],'b-');
        plt.ylabel('balance');
        plt.xlabel('d\'{ i}a');
        plt.grid(color='k', linestyle='--');
        plt.show();


        #calculo del rendimiento promedio del balance final
        r_ind.append(np.mean(rend))
        sigma_ind.append(np.std(rend))

    x_reng = X[i,:]
    r_reng = np.matrix(r_ind).T
    sigma_reng = np.matrix(sigma_ind).T
    sigma_reng = np.power(sigma_reng,2)
    x_reng2 = np.power(x_reng,2)

    rport = np.append(rport, np.array(x_reng.dot(r_reng)))
    sigma_port = np.append(sigma_port, np.array(x_reng2.dot(sigma_reng)))
```

```
    X = np.array(X.T)

    # reestric
    if re == 1:

        b1 = 10000
        b2 = 100000
        alpha1 = 1000
        alpha2 = 1000


        z = -b1*rport + b2*sigma_port

        i,j = X.shape
        rest = np.zeros(j)


        for act in X:
            # x > 0
            rest = rest + alpha1 * np.abs(act) * (act < 0)
            # x < 1
            rest = rest + alpha1 * np.abs(act) * (act > 1)


        X2 = np.transpose(np.matrix(X))
        X2 = np.array(X2)

        aux = []
        for act2 in X2:
            # sum x == 1
            aux.append(alpha2 * np.abs(np.sum(act2) - 1))
        rest = rest + np.array(aux)

        z = z.T + rest
    else:
        z = [rport, sigma_port]

    return z
```

6.6. app_1.py Código de aplación: Markowitz

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 20 21:36:24 2016
```

```
@author: Rodrigo
"""
import numpy as np
import matplotlib.pyplot as plt
import func_pso as pso
import gen_fun as f
import findata as fd

# %% funci\'{ o}n para evaluar
def psoeval_general(func, nvar, desc):
    print('\n\nEvaluating pso-algorithm... \n')
    # n\'{ u}mero de part\'{ i}culas
    npart = 2000
    # par\'{ a}metros de movimiento
    c1 = 0.01; c2 = 0.01
    # evalucaci\'{ o}n y slgor\'{ i}tmos pso
    prtl_mg, fpg, fp, prtl = pso.algo_pso(npart, c1, c2, func, nvar, desc)
    print('RESULTS: \n\n')
    # mostrar resultados.
    st = 'm\'{ i}nimo'
    if desc == 1: st = 'm\'{ a}ximo'
    string = '\n'
    for i in range(nvar):
        string = string+'    x'+str(i)+' = '+str(prtl_mg[i])+'\n'


    print('El',st,'encontrado es de',fpg, 'en la regi\'{ o}n {x0,x1,...,xn} :',string)


    return prtl_mg

# %% gr\'{ a}ficos de acciones
print('\nReading data and dependencies...')
acc = ["GFNORTEO.MX","LIVEPOLC-1.MX","HERDEZ.MX","BIMBOA.MX", "SANMEXB.MX", "ALSEA.MX"]
price, returns = fd.download(acc)
fd.graf_prec(price,returns, acc)

# %% Encontrar proporciones de activos utilizando PSO
nact = 6
print('\n\n-----------------------------------------------------------------------')
print('Dado un portafolio de n = {} activos se determinan las proporciones '.format(nact))
print('de inversi\'{ o}n de cada uno mediante optimizaci\'{ o}n por PSO. \n\n')
x = psoeval_general(f.markowitz,nact,-1)
```

```
# %% Evaluar y graficar markowitz
print('\n\n----------------------------------------------------------------------')
print('Usando las ponderaciones determinadas por el algoritmo PSO, se calcula')
print('el rendimiento y riesgo del portafolio y se grafica junto con simulaciones')
print('de diferentes portafolios. \n\n')
X = np.array([[i] for i in x])
rp, riskp, X, pa     = f.markowitz_1(X)
rpo, riskpo, Xo, pao = f.markowitz_1(pa.T)

f1 = plt.figure()
plt.plot(riskpo, rpo, 'b.', riskp, rp, 'ro')
plt.title('Creaci\'{ o}n y selecci\'{ o}n de portafolios')
plt.xlabel('Riesgo del portafolio')
plt.ylabel('Rendimiento del portafolio')
#f1.axes.get_xaxis().set_ticks([])
#f1.axes.get_yaxis().set_ticks([])
plt.show(f1)

print('\n\nDatos del portafolio seleccionado:\n')
print('\t Rendimiento: {}'.format(rp[0][0]))
print('\t Riesgo: {}\n\n'.format(riskp[0][0]))
```

6.7. app_2

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 20 21:36:24 2016

@author: Rodrigo


General procedure:

Genetic_algorithm
    optimize: function h(f)
    f is the vector or 'time intervals'
    h(f) = PSO
        optimize: function z(x)
        x is the vector or weights
            z(x) contains function prom_mov()

"""
```

```
import numpy as np
import matplotlib.pyplot as plt
import func_psof as pso
import gen_fun as fu
import func_agf as ga
import findata as fd

# %%Number assets for portfolio

nvar = 6

# %% Descarga de datos
acc = ["GFNORTEO.MX","LIVEPOLC-1.MX","HERDEZ.MX","BIMBOA.MX", "SANMEXB.MX", "ALSEA.MX"]
#n_acc, temp = np.shape(X)
acc = acc[:nvar]# acc[:n_acc]
price, returns = fd.download(acc)

npart = 1
# %% determinar mejor ventana

reg = {}
prtl = []
for j in range(nvar):
    prtl.append(np.random.rand(npart))
for i in range(1,128):


    reg[i] = fu.prom_mov3(prtl,[i]*nvar,price, returns, 0)
#print(reg[100])
# escoger mejor ventana por activo
rega = {}
for j in range(nvar):
    rega[j] = [0, -float('inf')]
for i in range(1, 128):
    for j in range(6):
        if rega[j][1] < reg[i][0][j]:
            rega[j] = [i, reg[i][0][j]]

f = [rega[i][0] for i in range(nvar)]

print('\n\nVentanas:\n' ,f)
# prom_mov3(prtl,[1]*6,price, returns, 0)

# %% Funci<c3><b3>n para evaluar PSO

def psoeval_general(nvar, desc, f, price, returns):
```

```
    print('\n\nEVALUATING PSO ALGORITHM... \n')
    # n<c3><ba>mero de part<c3><ad>culas
    npart = 50
    # par<c3><a1>metros de movimiento
    c1 = 0.01; c2 = 0.01
    # evalucaci<c3><b3>n y slgor<c3><ad>tmos pso
    prtl_mg, fpg, fp, prtl = pso.algo_psof(npart, c1, c2, fu.prom_mov, nvar, desc, f, price,
    print('\n RESULTS: \n\n')
    # mostrar resultados.
    st = 'm<c3><ad>nimo'
    if desc == 1: st = 'm<c3><a1>ximo'
    string = '\n'
    for i in range(nvar):
        string = string+'    x'+str(i)+' = '+str(prtl_mg[i])+'\n'


    print('El',st,'encontrado es de',fpg, 'en la regi<c3><b3>n {x0,x1,...,xn} :',string)


    return prtl_mg
'''
def h(x, price, returns):
    a = np.shape(x[0])
    if len(a) > 1: X = [x[i][0] for i in range(len(x))]
    else: X = x
    a,b,c = np.shape(X)
    X = np.reshape(X,(a,b))
    mX = np.asmatrix(X)

    h = {}
    for i in range(mX.shape[1]):
        fv = [int(mX[i,0]) for i in range(mX.shape[0])]
        prtl_mg = psoeval_general(3,-1, fv,price, returns)
        rport, sigma_port = fu.port_val(fv,prtl_mg,price, returns)
        h[(rport, sigma_port)] = [fv, prtl_mg]


    return h,

xmin = 1; xmax = 128; delta= xmin-xmax; nbits=7;
npob = 10; npadres = 4;
pobl, yp, yprom = ga.gen_algo(xmin, delta, nbits, npob, npadres, nvar, psoeval_general, 1, 3
'''


prtl_mg = psoeval_general(6, -1, f, price, returns)
```

```
print('\n Evaluating portfolio: \n')
mf = np.matrix(f)
mprtl_mg = np.matrix(prtl_mg)
r, s = fu.prom_mov3g(mprtl_mg,mf,price, returns,0)

print('\n\nReturn: {}'.format(r))
print('Standard Deviation: {}'.format(s))

print('\n sum: {}'.format(sum(prtl_mg)))
```

6.8. Figuras

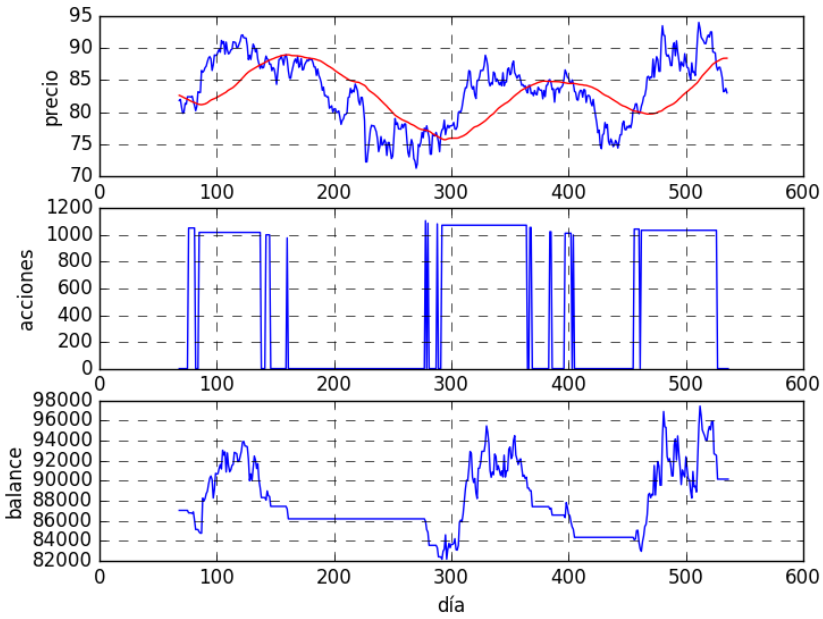


Figura 6.1: pendiente agregar

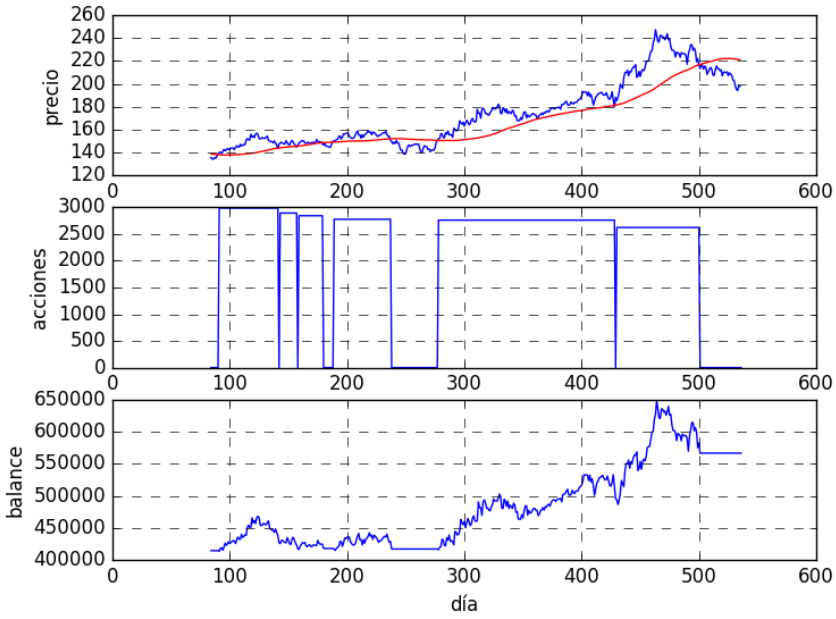


Figura 6.2: pendiente agregar

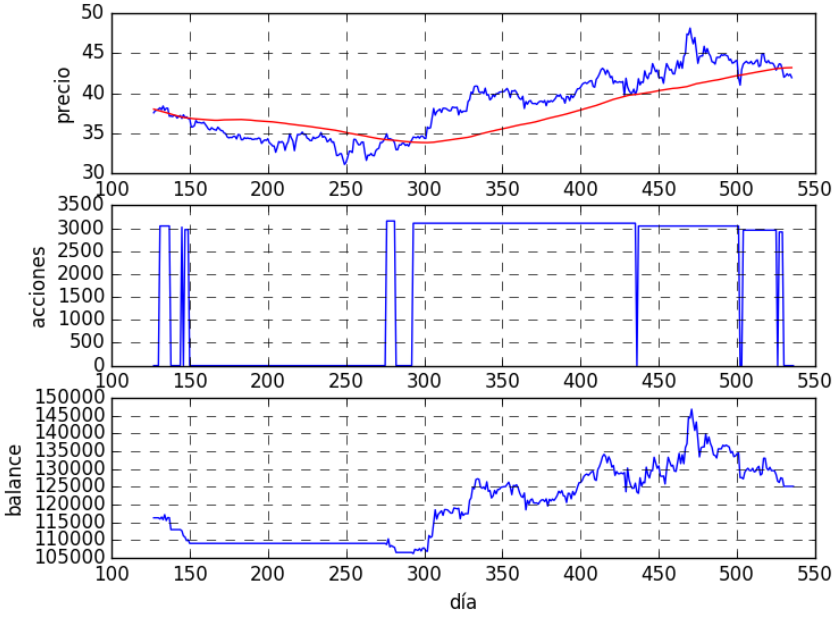


Figura 6.3: pendiente agregar

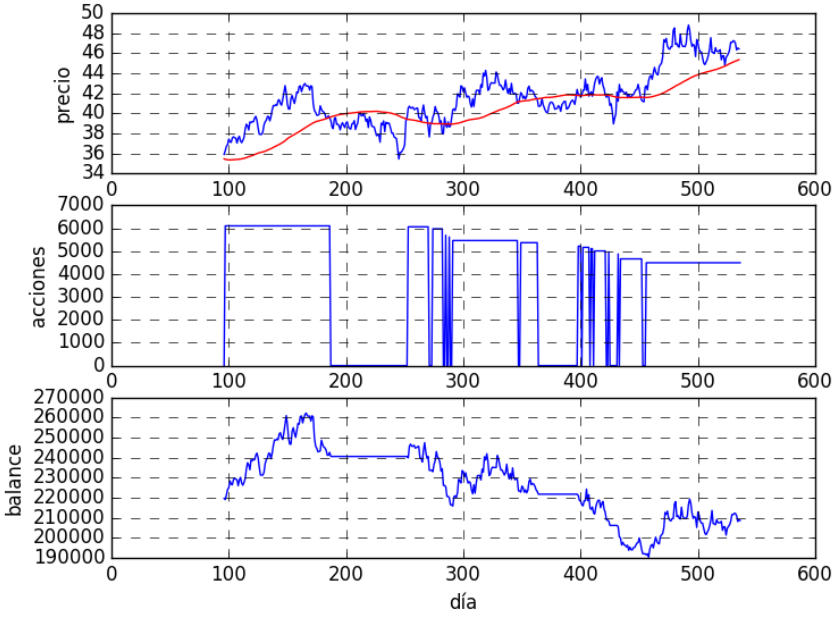


Figura 6.4: pendiente agregar

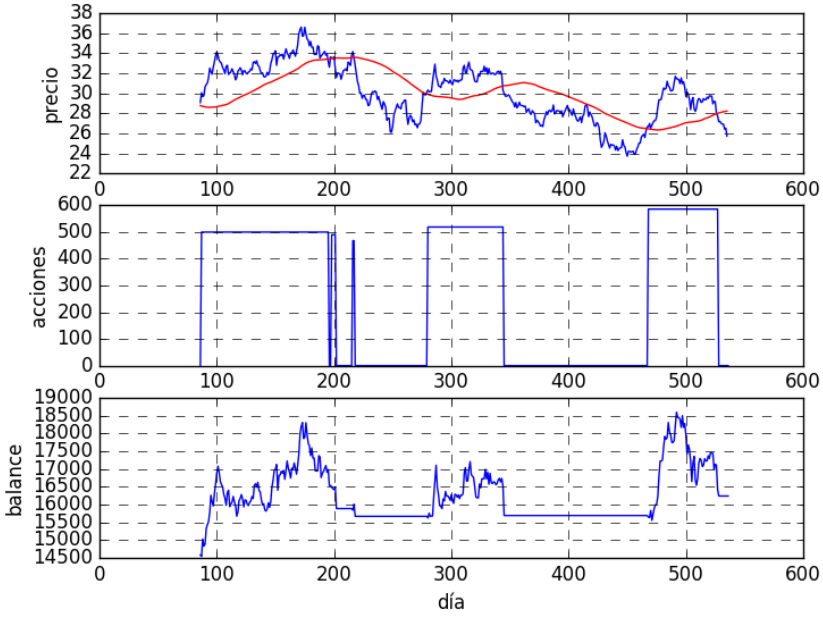


Figura 6.5: pendiente agregar

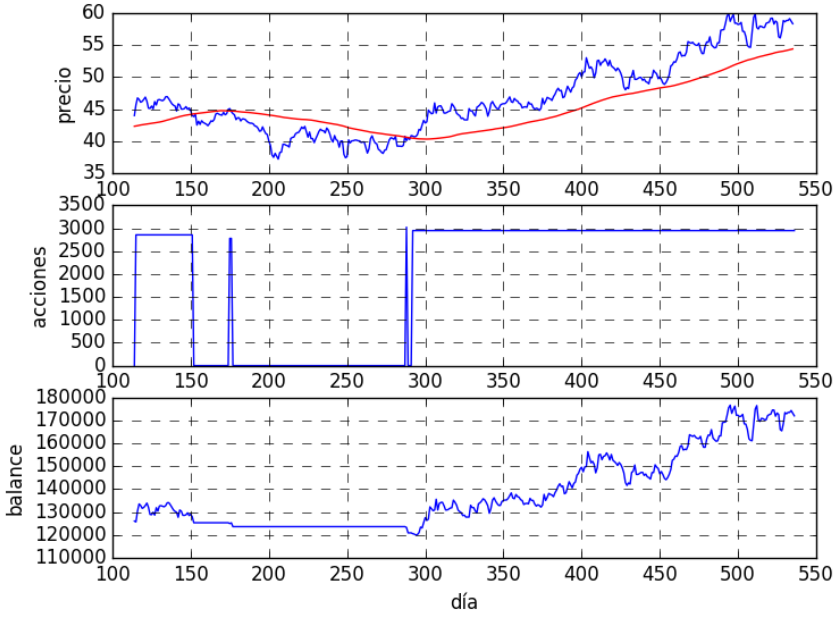


Figura 6.6: pendiente agregar