Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Triennale in Matematica Applicata

Tesi di laurea

Sviluppo e implementazione di un modello di arbitraggio statistico per un portafoglio di titoli azionari statunitensi

Candidato:
Davide Roznowicz

Relatore:
Prof. Dr. Alessandro Gnoatto

Anno Accademico 2019 – 2020

Indice

Elenco delle figure iii

Sommario iv

Introduzione vi

1 Clustering 1
- 1.1 Struttura dell'algoritmo di clustering *k-means* . . . 1
  - 1.1.1 Pseudocodice . . . 3
  - 1.1.2 Convergenza . . . 3
  - 1.1.3 Limiti e *k-means++* . . . 3
- 1.2 Struttura *PCA* . . . 5
  - 1.2.1 Costruzione algebrica . . . 5
  - 1.2.2 Codice MATLAB . . . 6
- 1.3 Analisi e costruzione dei clusters . . . 7
  - 1.3.1 Plot 2D della distribuzione dei rendimenti dei titoli . . . 7
  - 1.3.2 Individuazione di un opportuno numero di clusters . . . 9

2 Modelli 12
- 2.1 Modello per pairs-trading . . . 12
  - 2.1.1 Test di Dickey-Fuller . . . 13
- 2.2 Modello per statistical arbitrage tra baskets . . . 14
  - 2.2.1 Regressione di Engle-Granger & Test di Johansen . . . 15
- 2.3 Gestione ottima dei costi di transazione . . . 17

3 Setup & Backtest 22
- 3.1 Funzionamento backtest su piattaforma . . . 22
- 3.2 Step fondamentali per la costruzione del modello . . . 23
  - 3.2.1 Fase 1 : Aggregazione dati . . . 24
  - 3.2.2 Fase 2 : Clustering e gestione clusters . . . 25
  - 3.2.3 Fase 3 : Test e raccolta sottogruppi . . . 25
  - 3.2.4 Fase 4 : Score/Ranking dei sottogruppi . . . 26
  - 3.2.5 Fase 5 : Segnali e gestione posizioni . . . 28
- 3.3 Esecuzione backtest . . . 31
- 3.4 Considerazioni conclusive e possibili miglioramenti . . . 32

Glossario 35

Bibliografia 36

Elenco delle figure

1.1 Applicazione *kmeans++* e PCA: inizio=01/01/2013; fine=01/01/2017; numero di clusters=15; capitalizzazione minima=1B 9
1.2 Applicazione *kmeans++* e PCA: inizio=01/01/2013; fine=01/01/2017; numero di clusters=15; capitalizzazione minima=10B 10
1.3 *Akaike Information Criterion*: inizio=01/01/2013; durata: 60 giorni di contrattazione. 11

2.1 Esempio di *Distribuzione Stati del Sistema Cointegrato* costituito da *Chevron Corp (CVX)* e *Exxon Mobil Corp (XOM)*: inizio=01/01/2014; fine=01/04/2016; l'attrattore è costituito da una retta. 16
2.2 Esempio di *Frontiera efficiente* e *Traiettoria ottima* per tre possibili strategie: A ($\lambda = 2 \cdot 10^{-6}$), B ($\lambda = 0$), C ($\lambda = -2 \cdot 10^{-7}$) 20
2.3 Esempio di *Traiettoria ottima* in un portafoglio con due asset distinti nel caso di tre possibili strategie: A ($\lambda = 2 \cdot 10^{-6}$), B ($\lambda = 0$), C ($\lambda = -5 \cdot 10^{-8}$) . . . 21

3.1 Esempio di *relazione di cointegrazione* tra *Chevron Corp (CVX)* e *Exxon Mobil Corp (XOM)* relativo alla figura 2.1 30
3.2 Esempio di *relazione di cointegrazione* tra *'ALXN', 'SNY', 'ALNY'*; combinazione lineare generata automaticamente dall'algoritmo di backtesting nei 60 giorni di contrattazione precedenti al 05/10/2016. 30
3.3 Esempio di *Distribuzione Stati del Sistema Cointegrato* costituito da *'ALXN', 'SNY', 'ALNY'*; relativo alla figura 3.2; l'attrattore è costituito da un piano, con vettore di coefficienti $\beta = [1.0000 \quad -2.3404 \quad -0.6645]$. 31
3.4 Backtest dei due modelli con il setting descritto nel capitolo; non sono inclusi costi di transazione. 33
3.5 Statistiche descrittive del backtest presentate da QuantConnect relativamente al modello Pairs-Trading. 34

Sommario

Nella trattazione viene presentata la costruzione matematica di una strategia di arbitraggio statistico tra titoli azionari, con successiva implementazione in Python e MATLAB, tramite il supporto della piattaforma di backtesting QuantConnect che dà accesso ai dati necessari per le simulazioni di trading.
Verranno associati specifici algoritmi allo scopo di ottenere una quanto più rapida e accurata identificazione di cluster contenenti strumenti finanziari con comportamento storico assimilabile. Con successivi test si giungerà all'individuazione e al successivo utilizzo dei soli titoli che maggiormente esibiscono cointegrazione: ci si attenderà, dunque, che gli strumenti considerati siano sufficientemente inclini alla *mean-reversion*[1].
Tramite ulteriori tentativi di miglioramento della predittività del modello, insieme alla dinamica individuazione dei coefficienti e delle soglie di trading, si procederà a condurre una simulazione basata sui prezzi di chiusura giornaliera.
L'analisi effettuata sarà intervallata, al termine del secondo capitolo, da uno studio concernente l'ottimizzazione dei costi di transazione.

Introduzione

Molti strumenti finanziari evidenziano andamenti comparabili del processo di prezzo nel tempo (o meglio, del processo dei rendimenti). Tra alcuni di essi sussiste un'elevata affinità in termini di tipologia di attività che le società rappresentate svolgono. Spesso, ciò include anche analoghi rischi (economici, politici, concorrenziali...) cui queste ultime sono esposte.
Tuttavia ogni compagnia mantiene un certo grado di indipendenza dalle altre: infatti, in casi più o meno sporadici, può verificarsi un disallineamento delle quotazioni di un asset rispetto ad altri considerati simili. La pratica dell'arbitraggio statistico mira a colmare questa momentanea divergenza, assumendo che debba esserci una sorta di "equilibrio di medio-lungo termine", per cui il divario creatosi dovrà in qualche modo chiudersi.
Dal momento che questo tipo di strategia fa ampio affidamento sulla somiglianza storica dei titoli, si cerca di operare su gruppi di strumenti che abbiano saldamente manifestato la suddetta capacità di ricondursi all'equilibrio: ci si concentra, dunque, su quelli che risultano cointegrati. Con quest'ultima parola si fa riferimento alla *stazionarietà*[(2)] della differenza tra i rendimenti di una coppia di titoli, almeno in un certo intervallo di tempo. Il concetto di coppia, più tipico del *pairs-trading*[(3)], può chiaramente essere esteso ad una generica combinazione lineare di asset.
Si procederà infine, dopo aver valutato i più opportuni criteri di trading, a prendere una posizione *long*[(4)] sui titoli sotto-performanti, mentre si andrà *short*[(5)] sugli accoppiati corrispondenti.
Il vantaggio di un portafoglio costruito in questa maniera è che risulta molto meno esposto a fattori di rischio di mercato, riuscendo spesso a manifestare una correlazione molto bassa con gli indici azionari di riferimento. L'ottimizzazione dinamica delle varie fasi è fondamentale per ridurre al minimo l'esposizione a fattori quali volatilità e variabilità dei costi, cercando di massimizzare il profitto.

I capitoli 1 e 2 sono rispettivamente dedicati alla spiegazione delle tecniche di clustering e alla costruzione dei modelli di interesse. Invece, il terzo e ultimo capitolo si concentra sulla descrizione operativa delle fasi di trading dell'algoritmo di backtesting. Per avere un'idea di quali debbano essere le modalità di funzionamento e quindi gli aspetti di maggior interesse di tale algoritmo, si accennano fin da subito gli step fondamentali.

- FASE 1 : AGGREGAZIONE DATI
  - Costruzione della matrice dei prezzi X a partire dalle serie dei prezzi dei titoli azionari appartenenti all'insieme appositamente predisposto.
  - Conversione della matrice X in matrice di rendimenti normalizzata.
- FASE 2 : CLUSTERING E GESTIONE CLUSTERS
  - Applicazione dell'algoritmo di clustering *k-means++* su X: si ottengono dei cluster che raggruppano titoli con simili movimenti di mercato nella finestra temporale prestabilita.
  - Per ogni cluster si svolgono le seguenti operazioni:
    - Per i soli titoli appartenenti ad un medesimo cluster si costruisce una matrice X1 (sottomatrice di X) similmente a come si è fatto per X.

    - Si esegue *k-means++* su X1 per ottenere sottogruppi di pochi elementi su cui si potranno condurre agilmente i test.

- FASE 3 : TEST E RACCOLTA SOTTOGRUPPI
  - All'interno di ogni sottogruppo, individuato a seguito dell'applicazione di *k-means++* su X1, si effettuano gli opportuni test di cointegrazione tra i titoli (test di Dickey-Fuller o Johansen).
  - Si memorizzano in un dizionario le istanze relative ai sottogruppi considerati cointegrati: esse raccolgono informazioni rilevanti riguardanti i sottogruppi in questione.
- FASE 4 : SCORE/RANKING DEI SOTTOGRUPPI :
  - Si assegna uno score ad ogni istanza di sottogruppi cointegrati individuata.
  - Realizzazione di un dizionario di istanze ordinato in base allo score.
- FASE 5 : SEGNALI E GESTIONE POSIZIONI :
  - Si verifica la presenza di segnali di exit: se presenti su qualche sottogruppo, si procede alla liquidazione.
  - Si verifica la presenza di segnali di entry: se presenti su qualche sottogruppo, si procede all'aggiunta dei relativi titoli al portafoglio (aprendo opportunamente posizioni long/short), sempre che non si sia superato il numero massimo di posizioni disponibili.

Maggiori delucidazioni sulle fasi appena presentate verranno fornite successivamente, insieme a considerazioni di varia natura che contribuiranno ad avere una visione più chiara della trattazione in essere.

Capitolo 1

Clustering

1.1 Struttura dell'algoritmo di clustering *k-means*

Grazie alla sua semplicità ed efficacia, l'algoritmo denominato *k-means* è uno dei più noti per partizionare lo spazio di riferimento in clusters, ovvero gruppi di elementi che condividono determinate caratteristiche.
Molteplici sono state le varianti di *k-means* prodotte dalla ricerca accademica, nel tentativo di velocizzarlo o ridurne alcuni difetti. La struttura logica, tuttavia, è comune a tutte le versioni: vengono inizialmente scelti, in maniera più o meno casuale, k punti a rappresentare i *centroidi*[6] iniziali. Ogni elemento è dunque associato al centroide più vicino in base ad una misura di distanza scelta in precedenza. Una volta che i clusters sono stati costruiti, vengono aggiornati i centroidi. Si prosegue iterando tali passaggi fino a che non si raggiunge la convergenza, in accordo con qualche criterio prestabilito (ad esempio finchè i centroidi non si siano stabilizzati, oppure finchè non più di una certa percentuale di elementi cambi cluster di appartenenza).

Descriveremo ora la struttura in maniera più dettagliata.
Siano $x_1, x_2, ..., x_m \in \mathbb{R}^n$ un numero finito di elementi da ripartire in k clusters, $2 \leq k \leq m$. Data

$$f(W, Z) = \sum_{i=1}^{k} \sum_{j=1}^{m} w_{i,j} D(x_j, z_i) \tag{1.1}$$

ci si propone di determinare

$$\min_{W,Z} f(W, Z) \tag{1.2}$$

con il vincolo

$$\sum_{i=1}^{k} w_{i,j} = 1 \quad j = 1, 2, ..., m \tag{1.3}$$

in cui $w_{i,j} \in \{0, 1\} \quad i = 1, 2, ..., k, \;\; j = 1, 2, ..., m$
sapendo che: $W = [w_{i,j}] \in \mathbb{R}^{k \times m}$
$Z = [z_1, z_2, ..., z_k] \in \mathbb{R}^{n \times k}$
$z_i \in \mathbb{R}^n$ è il centro del cluster $i \quad i = 1, 2, ..., k$
$D(x_j, z_i)$ è una norma tra x_j e $z_i \quad i = 1, 2, ..., k, \;\; j = 1, 2, ..., m$

Una riga i di W esprime la relazione che intercorre tra l'elemento x_j (dove j è la colonna) ed il cluster i. Ciò significa che, all'interno di tale riga i, l'appartenenza all'i-esimo cluster sarà identificata come segue:

$$w_{i,j} = \begin{cases} 1 & \text{se l'elemento } x_j \text{ appartiene al cluster } i \\ 0 & \text{se l'elemento } x_j \text{ non appartiene al cluster } i \end{cases}$$

Conseguentemente, poichè un elemento può appartenere ad un solo cluster, si ha ciò che è espresso in 1.3.
Si consideri ora la definizione dell'insieme S:

Definizione 1 (insieme dei vincoli)**.** L' *insieme dei vincoli* per 1.2 è dato da : $S := \{ W \mid \sum_{i=1}^{k} w_{i,j} = 1 \quad j = 1, 2, ..., m, \;\; w_{i,j} \in \{0, 1\} \quad i = 1, 2, ..., k, \;\; j = 1, 2, ..., m \}$.

Dal momento che il problema 1.2 è non-convesso, esistono solitamente molti punti di minimo locale che non necessariamente corrispondono al minimo globale. Date le seguenti:

$$f(W^\star, Z^\star) \leq f(W, Z^\star) \quad \forall W \in S \tag{1.4}$$

$$f(W^\star, Z^\star) \leq f(W^\star, Z) \quad \forall Z \in \mathbb{R}^{n \times k} \tag{1.5}$$

Si può affermare:

Definizione 2 (soluzione parzialmente ottima)**.** Un punto $(W^\star, Z^\star)$ è una *soluzione parzialmente ottima* di 1.2 se soddisfa 1.4 e 1.5.

Per costruire una soluzione parzialmente ottima, consideriamo i problemi denotati P1 e P2:

P1 Dato $\hat{Z} \in \mathbb{R}^{n \times k}$, si intende determinare $\bar{W} \;\; t.c. \;\; f(\bar{W}, \hat{Z}) = \min_{W \in S} f(W, \hat{Z})$.

P2 Dato $\hat{W} \in S$, si intende determinare $\bar{Z} \;\; t.c. \;\; f(\hat{W}, \bar{Z}) = \min_{Z \in \mathbb{R}^{n \times k}} f(\hat{W}, Z)$.

Per stabilire la soluzione di P1 basta fare la seguente considerazione: per un j specifico, sicuramente $\exists r \in \{1, 2, ..., k\} \; t.c. \; D(x_j, z_r) \leq D(x_j, z_l) \quad l = 1, 2, ..., k$. Allora P1 si risolve ponendo:

$$w_{i,j} = \begin{cases} 1 & \text{se } i = r \\ 0 & \text{se } i \neq r \end{cases}$$

Chiaramente ciò si ripete per ogni $j, \;\; j = 1, 2, ..., m$.
Per quanto riguarda P2, si può determinare una soluzione esplicita dopo aver fissato la norma da utilizzare: scegliamo, anche pensando all'effettiva implementazione, la norma euclidea al quadrato. Perciò lo scopo è individuare z_i tale che minimizzi la somma dei quadrati dei residui all'interno di ogni cluster. Risulta possibile riscrivere la funzione da minimizzare in maniera molto più gestibile:

$$f(W, Z) = \sum_{i=1}^{k} \sum_{j=1}^{m} w_{i,j} D(x_j, z_i) = \sum_{i=1}^{k} \sum_{j=1}^{m} w_{i,j} \|x_j - z_i\|^2 = \sum_{i=1}^{k} \sum_{j=1}^{m} w_{i,j} (x_j - z_i)^2 = \sum_{i=1}^{k} \sum_{x_j \in Z_i} (x_j - z_i)^2 \tag{1.6}$$

dove con Z_i si denota il cluster i-esimo.

$$\begin{aligned} \frac{\partial}{\partial z_v} \sum_{i=1}^{k} \sum_{x_j \in Z_i} (x_j - z_i)^2 &= \sum_{i=1}^{k} \sum_{x_j \in Z_v} \frac{\partial}{\partial z_v} (x_j - z_v)^2 \\ &= - \sum_{x_j \in Z_v} 2 \cdot (x_j - z_v) \qquad v = 1, 2, ..., k \end{aligned}$$

$$\nabla_Z f(\hat{W}, Z) = 0 \quad \Leftrightarrow \quad |Z_v| \cdot z_v = \sum_{x_j \in Z_v} x_j \quad \Rightarrow \quad z_v = \frac{\sum_{x_j \in Z_v} x_j}{|Z_v|} \qquad v = 1, 2, ..., k$$

Da cui si evince che il miglior rappresentante per minimizzare la somma dei quadrati dei residui è la media dei punti all'interno del cluster.

1.1.1 Pseudocodice

Algorithm 1 k-means

procedure
step1
si sceglie un punto iniziale $Z^0 \in \mathbb{R}^{n \times k}$
$W^0 \leftarrow P1(f, Z^0)$
set $r = 0$
step 2
$Z^{r+1} \leftarrow P2(f, W^r)$
if $|f(W^r, Z^{r+1}) - f(W^r, Z^r)| > \varepsilon$ **then**
goto *step 3*
else
$(W^\star, Z^\star) \leftarrow (W^r, Z^{r+1})$
return $(W^\star, Z^\star)$
step 3
$W^{r+1} \leftarrow P1(f, Z^{r+1})$
if $|f(W^{r+1}, Z^{r+1}) - f(W^r, Z^{r+1})| > \varepsilon$ **then**
$r \leftarrow r + 1$
goto *step 2*
else
$(W^\star, Z^\star) \leftarrow (W^{r+1}, Z^{r+1})$
return $(W^\star, Z^\star)$

1.1.2 Convergenza

Viene ora illustrata la dimostrazione della convergenza tratta da [11].

Teorema 1. L'algoritmo presentato converge ad una *soluzione parzialmente ottima* di 1.2 in un numero finito di iterazioni.

Dimostrazione. Per prima cosa si procede mostrando che un determinato elemento di S viene visitato una sola volta prima che l'algoritmo si fermi. Supponiamo, per assurdo, che ciò non sia vero. Allora è possibile trovare $W^{r_1} = W^{r_2}$ per alcuni $r_1, r_2,\ r_1 \neq r_2$. L'esecuzione di *step 2* produce Z^{r_1+1} e Z^{r_2+1} come soluzioni ottimali per W^{r_1} e W^{r_2} rispettivamente. Da cui:

$$\begin{aligned} f(W^{r_1}, Z^{r_1+1}) &= f(W^{r_2}, Z^{r_1+1}) && \text{poichè } W^{r_1} = W^{r_2} \\ &= f(W^{r_2}, Z^{r_2+1}) && \text{da } \textit{step 2} \end{aligned}$$

Ma la sequenza di $f(\cdot, \cdot)$ generata da P2 in *step 2* risulta strettamente decrescente. Perciò $f(W^{r_2}, Z^{r_2+1}) < f(W^{r_2}, Z^{r_1+1})$ contraddice ciò che è stato affermato appena sopra. Ne deriva $W^{r_1} \neq W^{r_2}$.
Inoltre, dal momento che ci sono un numero finito di elementi in S, l'algoritmo raggiungerà una *soluzione parzialmente ottima* in un numero finito di iterazioni.

□

1.1.3 Limiti e *k-means++*

Come si è intuito dalla trattazione condotta finora, l'attenzione nello sviluppare una tale tipologia di algoritmo non si focalizza nel determinare la soluzione esatta di 1.2, ovvero il minimo globale

di 1.1. Fare ciò, infatti, sarebbe equivalente a tentare di risolvere un problema $\mathcal{NP}$. Piuttosto, tramite vari accorgimenti, ci si propone di determinare una *soluzione parzialmente ottima* soddisfacente. Questo tipo di limitazione, tuttavia, permette di far sì che la complessità temporale sia lineare rispetto alle variabili m, k, n, i, le quali, assumendo il significato precedentemente loro attribuito, fanno rispettivamente riferimento al numero di:

- m : elementi nello spazio di riferimento
- k : clusters
- n : dimensioni di ogni elemento
- i : iterazioni per raggiungere la convergenza

Più esplicitamente, come affermato in [10], il tempo di esecuzione si può approssimare con $O(m \cdot k \cdot n \cdot i)$. Solitamente, se gli elementi nello spazio presentano, almeno vagamente, un raccoglimento in possibili clusters, *k-means* converge alquanto rapidamente. Tuttavia, si possono identificare casi in cui la complessità temporale risulta superpolinomiale nel numero di iterazioni se si lascia che l'algoritmo operi fino all'esatta convergenza, come testimoniato in [2]. Dunque la maggior parte delle volte si preferisce fissare i fin dall'inizio.
Eppure, simili accorgimenti non garantiscono alcunchè sull'accuratezza dell'algoritmo. Infatti, ci sono determinati fattori che possono seriamente pregiudicare la buona performance di *k-means*. Tra i più rilevanti:

1. Scelta dei centroidi iniziali
2. Stima del numero di clusters

La gestione della prima questione risulta particolarmente significativa per evitare di incorrere in scarsi risultati di clustering. Piuttosto che inizializzare i k centroidi in maniera casuale, una loro scelta iniziale più ragionata può apportare enormi miglioramenti alla precisione dell'algoritmo. Nel lavoro [3], gli autori Arthur e Vassilvitskii presentano una modalità di selezione dei centroidi iniziali caratterizzata da un semplice approccio probabilistico: il primo centroide z_1 viene scelto con probabilità uniforme tra tutti gli elementi; dopodichè, uno per volta, si procede alla determinazione dei successivi $z_i,\ i = 2, 3, ..., k$ secondo una distribuzione che, ad ogni elemento $x_j,\ j = 1, 2, ..., m$ associa un peso pari a:

$$p(x_j) := \frac{D(x_j)^2}{\sum_{j=1}^{m} D(x_j)^2}$$

dove, immaginando che ad un determinato istante siano già stati scelti l centroidi, $1 \leq l \leq k$, si definisce $D(x_j) := min_{1 \leq i \leq l} \|x_j - z_i\|$ per una opportuna norma. Intuitivamente, viene attribuita una probabilità maggiore ad elementi che localizzerebbero i centroidi successivi lontano da quelli precedentemente individuati. Una volta stabiliti tutti i k centroidi iniziali, si esegue la normale precedura per *k-means*, partendo da *step 1*, ma ovviamente ignorando la linea che richiede l'inizializzazione casuale di Z^0.
Tale struttura, arricchita dalla scelta ponderata di Z^0, individua un algoritmo denominato *k-means++*. Esso apporta benefici consistenti sia dal punto di vista della velocità d'esecuzione, sia, come già detto, dal punto di vista dell'accuratezza.

La seconda questione fondamentale, ovvero l'individuazione del numero di clusters più adatto agli scopi che ci siamo prefissi, sarà esaminata più in dettaglio nell'ultima sezione di questo capitolo.

1.2 Struttura *PCA*

Nella gestione di un numero elevato di dimensioni caratterizzanti ogni singolo elemento, può risultare estremamente arduo comprendere se *k-means* stia partizionando lo spazio in maniera ragionevole. Infatti, poichè tale algoritmo necessita del numero di clusters k come dato in input, bisognerebbe poter verificare, almeno qualitativamente, con quale $\bar{k}$ si riesca a "spiegare" meglio la data distribuzione degli x_j. La costruzione del numero più adatto di clusters potrebbe essere cruciale per garantire il funzionamento dell'intera strategia.
Chiaramente il controllo "manuale" per constatare che alcuni elementi simili siano stati raccolti nel medesimo cluster può essere d'aiuto; tuttavia, in presenza di una quantità maggiore di x_j, la suddetta verifica si rivela inefficace. Una procedura di supporto in tale contesto è costituita dalla cosiddetta *Principal Component Analysis*, altresì chiamata PCA. Essa consiste in una riduzione dimensionale del set di dati di partenza, pur continuando a garantire una rappresentazione efficace delle caratteristiche dello spazio originario. Più specificatamente, a partire da un un gran numero di variabili correlate, permette di ottenere delle nuove variabili, tra loro non correlate, che trattengano la massima quantità possibile di varianza delle variabili iniziali. Tali nuove variabili sono ottenute tramite opportune combinazioni lineari delle variabili originarie: il vantaggio che ne consegue riguarda il fatto che, poichè vengono appositamente costruite in ordine decrescente rispetto alla varianza detenuta, anche solo un numero esiguo delle prime nuove variabili fornisce una buona approssimazione di qual è la disposizione reciproca degli elementi dello spazio di partenza.

1.2.1 Costruzione algebrica

Denotiamo con n il numero di variabili originarie, raccolte in un vettore $x = [x_1, x_2, ..., x_n]^T$. Inizialmente ci si propone di determinare un vettore di coefficienti $\alpha = [\alpha_1, \alpha_2, ..., \alpha_n]^T$ tale che la nuova variabile $\alpha^T \bullet x = \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n x_n = \sum_{j=1}^{n} \alpha_j x_j$ trattenga la massima varianza possibile. Per conseguire tale obiettivo, bisogna massimizzare $\mathbb{V}ar(\alpha^T x) = \alpha^T \Sigma \alpha$, dove Σ è la matrice delle covarianze. Si mostra necessario, in tale contesto, l'utilizzo di un vincolo affinchè tale massimizzazione avvenga attribuendo coefficienti finiti ad α. Solitamente si normalizza α ponendo $\alpha^T \alpha = 1$. Dopodichè si applicano i moltiplicatori di Lagrange:

$$\alpha^T \Sigma \alpha - \lambda(\alpha^T \alpha - 1)$$

con $\lambda \in \mathbb{R}$; dunque si procede con le derivate parziali:

$$\begin{aligned} \frac{\partial}{\partial \alpha}\{\alpha^T \Sigma \alpha - \lambda(\alpha^T \alpha - 1)\} &= \Sigma\alpha - \lambda\alpha = 0 \\ &= (\Sigma - \lambda\mathbb{I})\alpha = 0 \end{aligned}$$

È evidente da questa scrittura che λ svolge il ruolo di autovalore di Σ, mentre α quello di relativo autovettore. Per stabilire quale sia l'autovettore da scegliere tra gli n possibili, bisogna notare che, come discende da sopra:

$$\alpha^T \Sigma \alpha = \alpha^T \lambda \alpha = \lambda \alpha^T \alpha = \lambda$$

Perciò, affinchè sia massima la varianza, dobbiamo imporre che l'autovalore sia il più grande possibile. Quindi α è proprio l'autovettore relativo all'autovalore massimo. Per semplicità, indichiamo con λ_k il k-esimo autovalore in ordine di grandezza e con μ_k l'autovettore ad esso corrispondente, dove $k = 1, 2, ..., n$. La prima nuova variabile, anche chiamata *Primo Componente Principale* (*PC1*), sarà dunque $z_1 = \mu_1 x$ con $\mathbb{V}ar(z_1) = \lambda_1$.
Ora ci si propone di determinare ognuna delle altre nuove variabili in modo che abbiano, ancora

una volta, la massima varianza possibile, seppur chiaramente non possano trattenere tanta varianza quanto *PC1*. Indichiamo con β il vettore dei coefficienti per la seconda nuova variabile e costruiamo *PC2* richiedendo sia che $z_2 = \beta^T \Sigma \beta$ massimizzi $\mathbb{V}ar(z_2)$ sia che $\mathbb{C}ov(z_1, z_2) = 0$, in modo che le nuove variabili risultino non correlate. Si ricava:

$$\mathbb{C}ov(z_1, z_2) = \mathbb{C}ov(\alpha^T x, \beta^T x) = \alpha^T \Sigma \beta = \beta^T \Sigma \alpha = \beta^T \lambda_1 \alpha = \lambda_1 \beta^T \alpha = \lambda_1 \alpha^T \beta = 0$$

Dopo aver imposto il vincolo appena ottenuto $\beta^T \alpha = 0$ e applicato la normalizzazione $\beta^T \beta = 1$ come in precedenza, si procede con i moltiplicatori di Lagrange:

$$\beta^T \Sigma \beta - \lambda(\beta^T \beta - 1) - \psi \beta^T \alpha$$

con $\lambda, \psi \in \mathbb{R}$. Tramite le derivate parziali: $\frac{\partial}{\partial \beta}\{\beta^T \Sigma \beta - \lambda(\beta^T \beta - 1) - \psi \beta^T \alpha\} = \Sigma\beta - \lambda\beta - \psi\alpha = 0$
Ora moltiplicando ciò che abbiamo ottenuto per α^T, discende:

$$\alpha^T \Sigma \beta - \lambda \alpha^T \beta - \psi \alpha^T \alpha = 0$$

da cui, valendo la normalizzazione $\alpha^T \alpha = 1$, ne consegue $\psi = 0$. Pertanto:

$$\Sigma\beta - \lambda\beta = (\Sigma - \lambda\mathbb{I})\beta = 0$$

Nuovamente λ si rivela l'autovalore più grande disponibile (escluso λ_1), ovvero λ_2; mentre μ_2 è il corrispondente autovettore (assumendo che Σ abbia tutti autovalori distinti).
Similmente, in generale:

$$z_k = \mu_k x \qquad\qquad \mathbb{V}ar(z_k) = \lambda_k$$

Avvalendosi di questa tecnica, verranno effettuati dei plot efficaci e utili per supportare la costruzione del modello nel suo complesso.
A dir la verità, l'idea iniziale era di usare PCA anche per pre-processare gli elementi dello spazio di riferimento, prima di applicare *k-means*. Avrebbe dunque svolto il ruolo di algoritmo utile a ridurre la dimensionalità dello spazio per accelerare la fase di clustering. Tuttavia, ci si è accorti che il solo *k-means* è sufficientemente veloce (seppur non veloce quanto PCA+*k-means*) ai nostri scopi, nel caso di una finestra temporale non troppo lunga. Invece, all'aumentare della lunghezza della finestra, l'onere di pre-processare lo spazio con PCA supera ampiamente i benefici da esso generati, in quanto il costo computazionale di determinare gli autovalori è proporzionale, appunto, al cubo del numero di periodi dell'intervallo temporale scelto. Di conseguenza si utilizzerà solamente *k-means++* sull'intero spazio; supporto a tale scelta si può trovare anche in [8].

1.2.2 Codice MATLAB

Segue l'implementazione MATLAB dell'algoritmo per *PCA*.

```
function [PCs,eigval,eigvect]=myPCA(X)
% suppongo di avere n variabili e d dimensioni (oppure d osservazioni)
%
% input:    X matrice di dati dxn
%
% output:   PCs:        principal components, ovvero la rappresentazione
%                       delle nuove variabili nel nuovo spazio
%           eigval:     autovalori relativi alla matrice delle
%                       covarianze: corrispondono alla varianza
%                       trattenuta da ogni nuova nuova variabile
%           eigvect:    autovettori relativi alla matrice delle
```

```
%                       covarianze: corrispondono ai vettori di
%                       coefficienti che
%                       permettono di costruire i PCs

Cov=cov(X);
[eigvect,eigval]=eig(Cov);
eigval=flip(diag(eigval));
eigvect=(flip(eigvect'))';
PCs=X*eigvect;
end
```

1.3 Analisi e costruzione dei clusters

Lo scopo principale di questa sezione consiste nello svolgere una quanto più accurata analisi della distribuzione degli elementi nello spazio: di conseguenza, ci si propone di determinare un opportuno numero di clusters, tenendo conto, durante i ragionamenti, di quali siano le esigenze specifiche per il modello di arbitraggio in costruzione.
Poichè QuantConnect rende disponibile soltanto dati di titoli quotati negli exchange americani, ci si limiterà ad operare con questi. Precisamente, in tale parte della trattazione, si procederà nelle suddette indagini avvalendosi del supporto di MATLAB, poichè la piattaforma di backtesting non fornisce sufficiente libertà nelle rappresentazioni grafiche, apparentemente vietando l'utilizzo di *matplotlib*. Si è perciò deciso di accedere ai dati necessari richiamando l'API di *Yahoo finance*; allo scopo, sono state scaricate delle opportune funzioni di gestione di tali chiamate e degli eventuali errori da esse generati, reperibili al seguente link: https://it.mathworks.com/matlabcentral/fileexchange/68361-yahoo-finance-and-quandl-data-downloader. Per accedere alla serie storica relativa ad uno specifico titolo azionario risulta fondamentale avere a disposizione il relativo *ticker symbol*[7]: come rappresentanti della maggior parte delle azioni americane quotate più rilevanti, sono stati indentificati due exchange, NYSE e NASDAQ. Da https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nyse&render=download e da https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nasdaq&render=download si ottengono i file .csv da cui estrarre ticker symbols e capitalizzazione per l'analisi seguente. Essendo consapevoli di possibili errori nei dati prelevati da Yahoo e volendo inoltre evitare titoli poco scambiati, si procede filtrando fin da principio nel seguente modo:

- Non vengono esaminati i titoli con dati mancanti/incompleti
- Escludiamo i titoli con capitalizzazione inferiore al miliardo di dollari (in data 04/04/2020)
- Consideriamo una sola volta titoli che sono quotati in entrambi gli exchange

In maniera del tutto casuale viene scelta una finestra temporale di 4 anni, precisamente dal 01/01/2013 fino al 01/01/2017, e vengono raccolte in una matrice le serie storiche contenenti i prezzi di chiusura per ogni azione precedentemente selezionata.

1.3.1 Plot 2D della distribuzione dei rendimenti dei titoli

Si è inizialmente tentato di studiare la distribuzione dei rendimenti giornalieri di tutte le azioni che superavano la selezione descritta in precedenza. Prima di eseguire *k-means++* su tutti i titoli dell'insieme considerato, si procede ad una normalizzazione delle variabili, ovvero delle serie storiche dei rendimenti relative ad ogni azione: sottrendo la media e dividendo per lo scarto quadratico medio ci si riconduce infatti ad una distribuzione $\sim \mathcal{N}(0,1)$; tramite questa operazione

la costruzione dei clusters avviene correttamente, in accordo con il criterio di somiglianza della distribuzione dei rendimenti.

```
% Si costruisce il plot 2D : si applica k-means++ sull' intero spazio;
% successivamente, si rappresentano i primi due PCs per avere un'idea
% della distribuzione dei titoli, rappresentando con uno stesso colore
% elementi che appartengono al medesimo cluster.

% startdate='01/01/2013';
% enddate='01/01/2017';
% interval='1d';
MinCap=1000;  % capitalizzazione minima di 1 miliardo $
kclusters=15; % numero di clusters che si vuole ottenere (ad esempio)

load 'DataMatrix' % contiene la matrice X: 1523 azioni x 1007 giorni

capsel=find(MktCap<MinCap); % seleziono per capitalizzazione
MktCap(capsel)=[]; symbols(capsel)=[]; X(capsel,:)=[];
[m,n]=size(X);

for k=1:n-1 % computo rendimenti giornalieri
    X(:,k)=(X(:,k+1)-X(:,k))./X(:,k);
end
X=X(:,1:n-1); % non ho rendimenti giornalieri per ultimo giorno
X=(X-mean(X,2))./std(X'); % normalizzo ogni serie riconducendola
% ad una distribuzione N(0,1)
[m,n]=size(X); % m stocks; n timeseries

% chiediamo di ripetere kmeans 5 volte
% e prendiamo il tentativo che meglio minimizza la somma dei quadrati dei
% residui degli elementi di ogni cluster
idx = kmeans(X,kclusters,'Replicates',5);

[PCs,eigval,eigvect]=myPCA(X);
PCX=[PCs(:,1) PCs(:,2)]; % consideriamo solo i primi due PCs per il plot

figure;
for j=1:kclusters % stesso colore per elementi dello stesso cluster
    C = linspecer(kclusters); % per avere colori distinguibili
    plot(PCX(idx==j,1),PCX(idx==j,2),'.','MarkerSize',12,...
        'color',C(j,:),'linewidth',3);
    hold on;
end
xlabel('PC1')
ylabel('PC2')
title ('Clusters')
hold off
```

Per queste prime figure il numero di clusters è a scopo illustrativo; a breve si determinerà quale numero è più appropriato per il dataset in esame e le esigenze imposte dal modello.
Potrebbe rivelarsi utile stabilire un ottimo numero di clusters anche per i titoli con capitalizzazione non inferiore a 10 miliardi di dollari, sia per confermare la bontà della tecnica che si utilizzerà sia per, eventualmente, restringere l'insieme dei titoli su cui si intende operare.

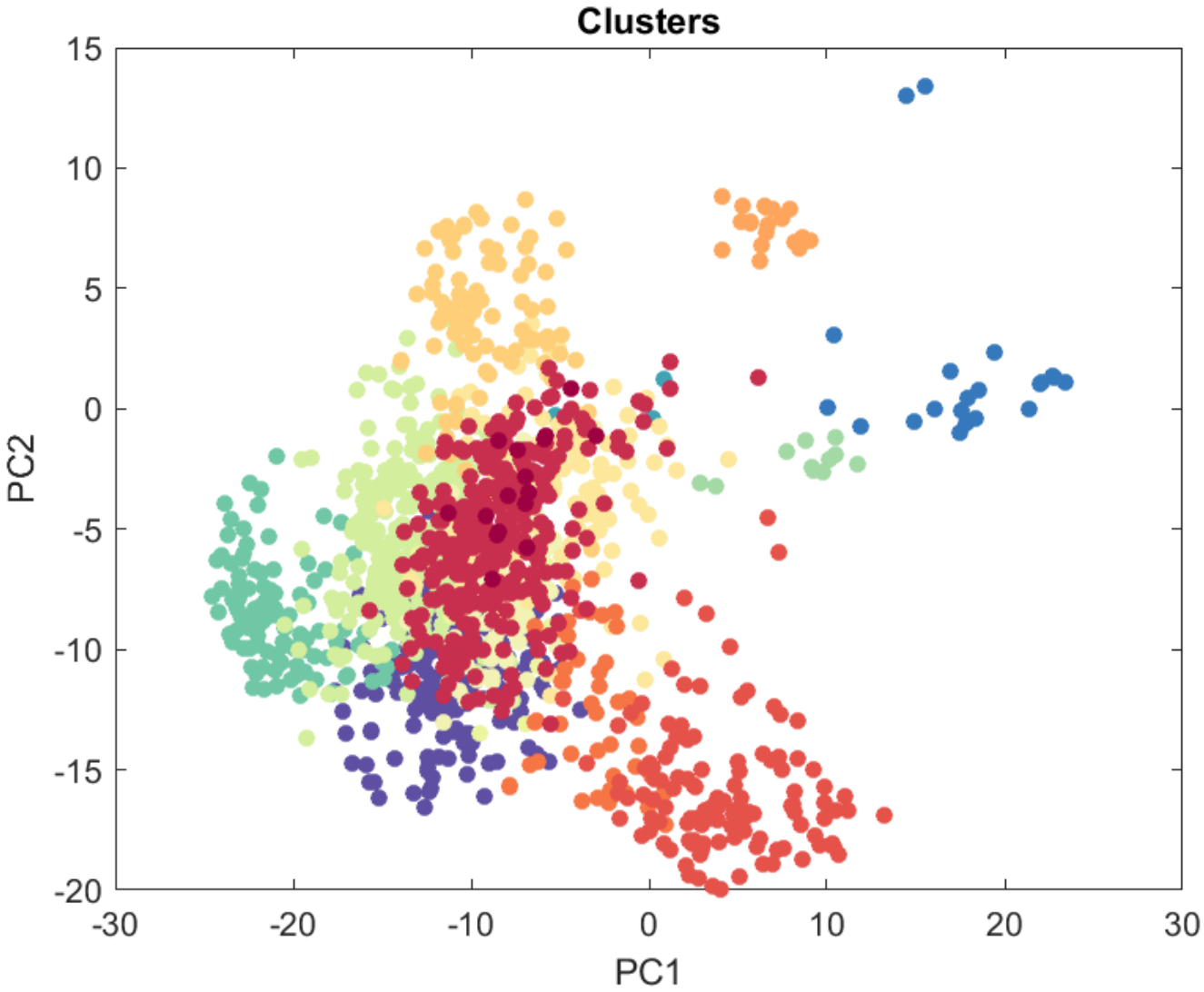


Figura 1.1 – Applicazione *kmeans++* e PCA: inizio=01/01/2013; fine=01/01/2017; numero di clusters=15; capitalizzazione minima=1B

1.1 e 1.2 fanno entrambe riferimento all'intera lunghezza della serie temporale a disposizione; tuttavia, poichè la *rolling window*[8] che si adotterà durante il backtest sarà compresa tra 2 e 3 mesi, è preferibile analizzare i titoli su di un arco di tempo inferiore. Senza discostarsi molto dalla scelta che molti paper di ricerca tendono a compiere, si fisserà la finestra temporale a circa 60 giorni di contrattazioni (poco meno di 3 mesi reali): dovrebbe essere un tempo sufficientemente lungo per verificare l'eventuale cointegrazione e svolgere successive analisi in merito.

1.3.2 Individuazione di un opportuno numero di clusters

In [4] si fa riferimento ad alcune funzioni nella variabile k da massimizzare/minimizzare allo scopo di determinare un numero di clusters $\bar{k}$ che si ritenga adeguato per il dataset in esame. Ne valuteremo un paio verificando nel contempo che, come ragionevolmente ci si potrebbe attendere al fine di costruire un buon modello, ogni raggruppamento non contenga più di poche decine di elementi.
Per primo è stato studiato il Calinski-Harabasz-Index.

$$CH(k) = \frac{\frac{BSS(k)}{k-1}}{\frac{WSS(k)}{m-k}} \tag{1.7}$$

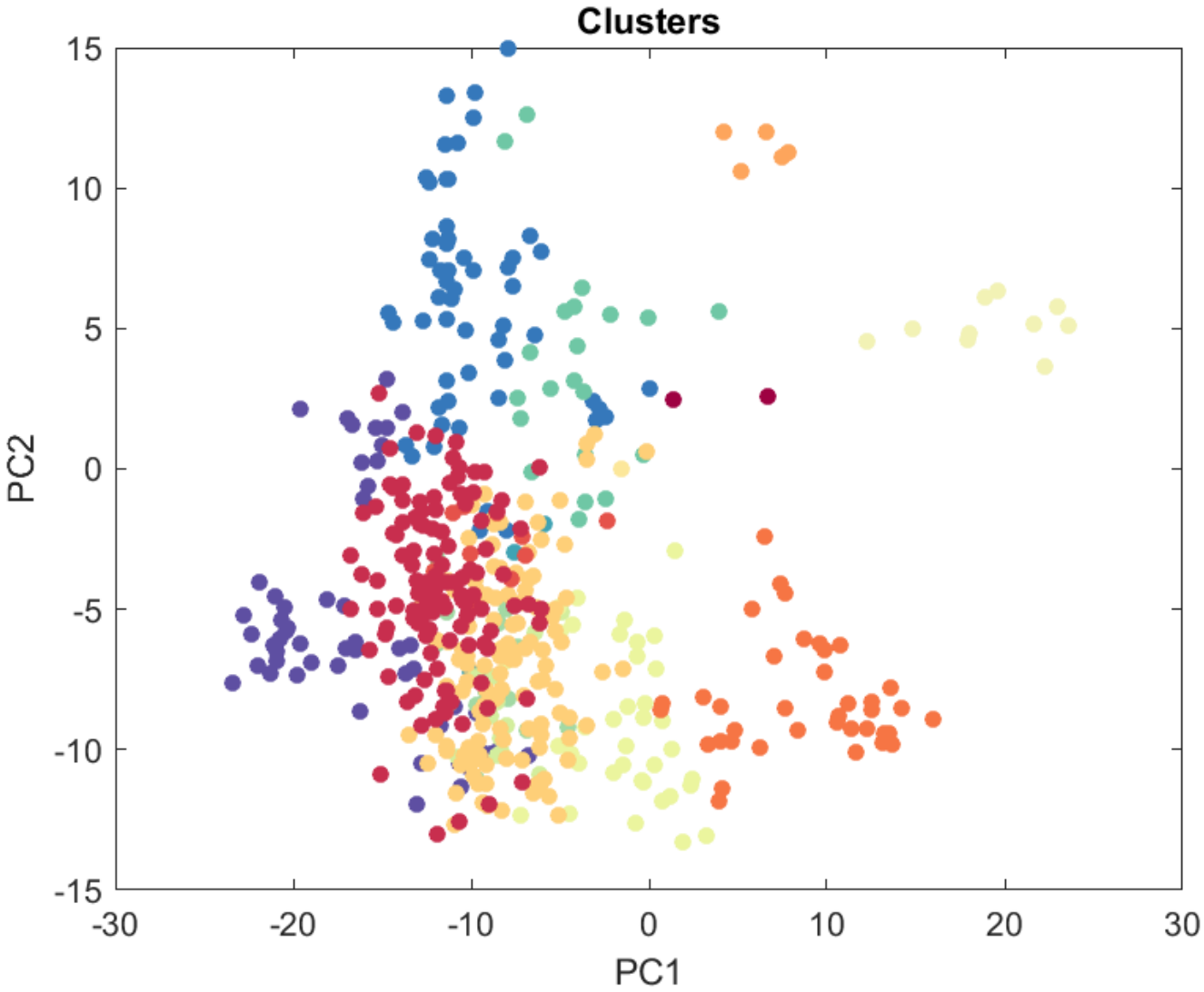


Figura 1.2 – Applicazione *kmeans++* e PCA: inizio=01/01/2013; fine=01/01/2017; numero di clusters=15; capitalizzazione minima=10B

$$BSS(k) = \sum_{i=1}^{k} |Z_i| \cdot (z_i - \frac{1}{m} \sum_{j=1}^{m} x_j)^2 \tag{1.8}$$

Il suo funzionamento è piuttosto intuitivo in quanto cerca di definire il numero di clusters=k in modo da massimizzare la funzione 1.7. Per raggiungere tale scopo, attribuisce un punteggio maggiore al diminuire di *Within cluster sums of squares* (WSS), ovvero 1.6, e all'aumentare di *Between cluster sums of squares* (BSS), cioè 1.8, inserendo degli aggiustamenti in base al valore di k (notazioni di inizio capitolo).
Tuttavia, ricordando la conformazione del dataset, la quale non presenta eccessiva separazione tra i clusters per alcun numero specifico k, e tenendo a mente il fatto che si è poco interessati ad avere i clusters ben distinti quanto piuttosto ad avere una gran omogeneità interna, ovvero gran somiglianza tra gli elementi appartenenti al medesimo cluster, si preferisce non utilizzare tale funzione come riferimento. L'esito era in ogni caso uno score decrescente in k, per il primo motivo appena indicato.
Una funzione che si accorda molto meglio con gli obiettivi della trattazione è:

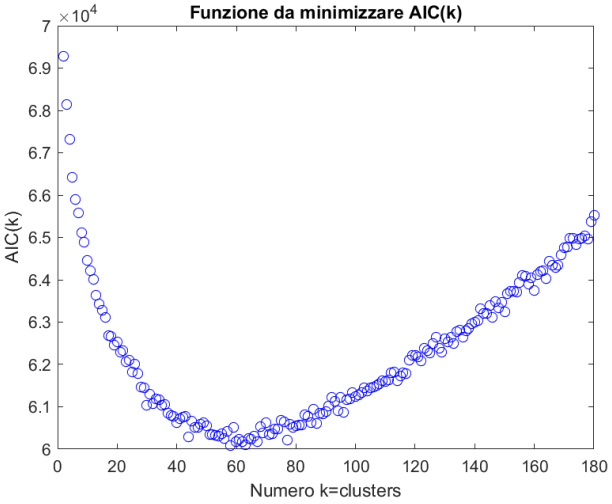
$$AIC(k) = WSS(k) + 2nk \tag{1.9}$$

dove n indica il numero di dimensioni del dataset. La 1.9, detta *Akaike Information Criterion*, individua il numero di clusters ottimo k come quel valore che la minimizza. Essa risulta diret-

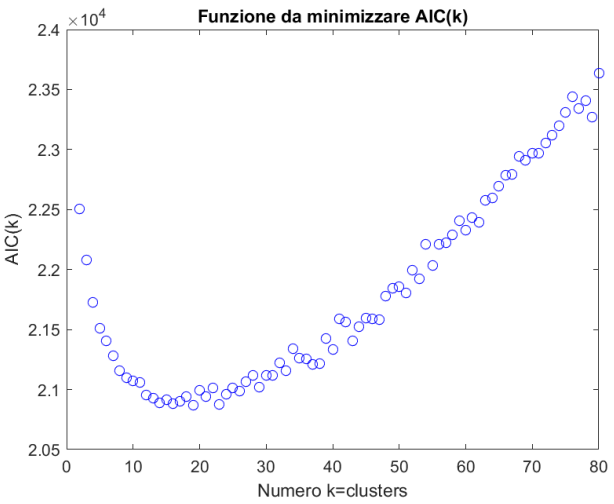
tamente proporziale sia a WSS che a k, ma non introduce nessuna penalizzazione per BSS piccolo.

```
function [AIC]=myAIC(X,k)
[~,n]=size(X);
[idx,C] = kmeans(X,k,'Replicates',5);
WSS=0;
for j=1:k
    pos=find(idx==j); % identifico gli elementi del cluster j
    WSSj=0;
    for s=1:length(pos)
        WSSj=WSSj+(C(j,:)-X(pos(s),:))*(C(j,:)-X(pos(s),:))';
    end
    WSS=WSS+WSSj;
end
AIC=2*n*k+WSS;

end
```



(a) capitalizzazione minima=1B



(b) capitalizzazione minima=10B

Figura 1.3 – *Akaike Information Criterion*: inizio=01/01/2013; durata: 60 giorni di contrattazione.

Nei grafici in 1.3 si visualizzano chiaramente le regioni di minimo e quindi gli intorni di k per cui le funzioni sono minimizzate. In particolare, riferendosi a 1.3 (grafico a sinistra), si nota ciò che si sperava: l'intervallo per k tra 50 e 70 determina il minimo, per cui, mediamente, ci saranno tra i 20 e i 30 elementi per cluster a seconda della scelta precisa di k; per quanto riguarda 1.3 (grafico a destra), invece, il numero ottimo di clusters è compreso approssimativamente tra 15 e 20. Probabilmente verrà preferito un numero non elevato di titoli per cluster, optando dunque per $k \approx 70$ nel caso di circa 1500 elementi nel dataset ($k \approx 20$ nel caso di circa 500 elementi). La scelta dipenderà da quante saranno le azioni totali messe a disposizione da QuantConnect.

Capitolo 2

Modelli

I primi modelli di arbitraggio statistico tra titoli azionari si focalizzavano sull'analizzare le relazioni intercorrenti tra coppie di essi: ciò portava alla costruzione di portafogli di *pairs-trading*, in cui potevano figurare varie posizioni aperte su coppie diverse, in modo da favorire la diversificazione. Ancora oggi, la maggior parte dei paper di ricerca in merito, in campo accademico, è orientata ad analisi di questo tipo, in cui vengono suggerite e implementate possibili tecniche per ovviare a specifiche problematiche.
In questa sede verrà, innanzitutto, presentato un modello adeguato per la realizzazione di un portafoglio di *pairs-trading*, che sarà successivamente implementato in QuantConnect.
Tuttavia, dal momento che la relativa semplicità nella realizzazione della suddetta strategia ne ha facilitato un'ampia adozione, i rendimenti ad essa legati sono scesi notevolmente nell'ultimo decennio. Ecco il motivo per cui l'attenzione alla gestione dei costi, trattata nell'ultima sezione del capitolo, risulta estremamemte rilevante nella pratica, insieme all'individuazione, seppur ardua, di soluzione efficaci a problematiche che in tale contesto ancor di più affliggono la performance, come la presenza di relazioni spurie tra i titoli.
In ogni caso, si ritiene che, invece di limitarsi ad operare con una singola coppia di azioni, si possa ampliare la visione in modo da considerare una opportuna combinazione lineare tra titoli costituenti un sistema cointegrato. Tale osservazione nasce dall'aspettativa che un simile modello sia in grado di generare ritorni più consistenti o quantomeno contribuire alla riduzione della volatilità della strategia. Pertanto la seconda sezione del capitolo si concentrerà sulla modellizzazione di un portafoglio di arbitraggio statistico tra baskets di azioni. La sua effettiva attuazione avverrà, poi, tramite la piattaforma di backtesting in uso.

2.1 Modello per pairs-trading

Consideriamo una coppia di titoli azionari (X,Y) le cui serie storiche dei prezzi siano rispettivamente identificate da x_t e y_t. Affinchè sussista una relazione di "somiglianza" che possa perdurare, bisogna aspettarsi che, per un opportuno $\beta \in \mathbb{R}$, la differenza $e_t = y_t - \beta x_t$ sia un processo stazionario, ovvero ci sia cointegrazione. In generale, il processo di prezzo di un'azione non è stazionario bensì un random walk: prima di verificare la cointegrazione, bisogna accertarsi dell'integrazione. Si consideri un modello AR(1), ovvero un modello autoregressivo con un solo lag:

$$z_t = \varphi + \rho z_{t-1} + \varepsilon_t \tag{2.1}$$

dove $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ iid. In riferimento a 2.1, z_t si definisce integrata di ordine 1, scritta $I(1)$, se $\rho = 1$, ovvero nel caso di random walk. Il concetto di integrazione risulta legato al numero di volte che occorre differenziare una serie per renderla stazionaria: nel caso in esame, infatti, tramite una sola differenziazione, si ottiene che $\Delta z_t = z_t - z_{t-1}$ è stazionario, ovvero $I(0)$. In ogni caso, tali concetti verranno esplicitati in un quadro molto più generale nella prossima sezione.
In generale, è difficile poter utilizzare liberamente 2.1: spesso, infatti, gli errori ε_t sono autocor-

relati, ovvero $\mathbb{C}ov(\varepsilon_t, \varepsilon_{t-1}) \neq 0$. Per ovviare a questo problema, è pratica comune aggiungere un numero sufficiente di differenze prime in modo da garantire l'assenza di autocorrelazione.
Si intende dunque esprimere e_t come un processo AR(1), scritto in differenze, ed avente un opportuno numero di termini ritardati per annullare la correlazione tra gli errori. Ricaviamo, considerando ν_t errore potenzialmente autocorrelato:

$$
\begin{aligned}
& e_t = \theta + \alpha e_{t-1} + \nu_t \\
& e_t - e_{t-1} = \theta + \alpha e_{t-1} - e_{t-1} + \nu_t && \text{sottraendo da entrambe le parti } e_{t-1} \\
& \Delta e_t = \theta + \omega e_{t-1} + \nu_t && \text{sostituendo } \omega = \alpha - 1 \\
& \Delta e_t = \theta + \omega e_{t-1} + \sum_{s=1}^{p} \gamma_s \Delta e_{t-s} + \varepsilon_t && \text{sostituendo } \nu_t \text{ autocorrelato con } p \text{ termini ritardati}
\end{aligned}
$$

dove, nuovamente, $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$. La mancanza di un termine di trend non è casuale, bensì un'apposita scelta dettata dall'aspettativa che il processo delle differenze e_t sia stazionario.
Inoltre, non potendo osservare direttamente e_t, tale quantità sarà rimpiazzata da una stima di regressione $\hat{e}_t$.
Chiaramente occorre stabilire un modo per stimare un adeguato numero p di ritardi. In questo contesto, vengono in aiuto criteri d'informazione quali AIC o SC (*Schwarz Criterion*):

$$AIC = \log\left(\frac{SQR}{N}\right) + \frac{2K}{N} \tag{2.2}$$

$$SC = \log\left(\frac{SQR}{N}\right) + \frac{K \log N}{N} \tag{2.3}$$

dove N è la numerosità campionaria; K è il numero di parametri stimati per il modello in esame; $SQR = \sum_{i=1}^{N} \hat{e}_i^2$ è la somma dei quadrati dei residui.
Lo scopo è scegliere p in modo tale da minimizzare 2.2 oppure 2.3, ognuna delle quali si prefigge di rendere minima la somma dei quadrati dei residui, introducendo una penalità all'aumentare di K. Tuttavia, ai fini del modello, specie pensando al caso multivariato della prossima sezione, conviene dover gestire il numero più basso possibile di ritardi: infatti, quando i coefficienti non sono più scalari ma matrici, un solo ritardo aggiuntivo equivale a computare d^2 parametri, assunto d come dimensione della matrice. In questa ottica, si preferisce adottare 2.3 poichè penalizza maggiormente l'aggiunta di parametri: infatti, si ha $\frac{K \log N}{N} > \frac{2K}{N}$ per $N \geq 8$.

2.1.1 Test di Dickey-Fuller

Partendo dal modello AR precedentemente costruito:

$$\Delta \hat{e}_t = \theta + \omega \hat{e}_{t-1} + \sum_{s=1}^{p} \gamma_s \Delta \hat{e}_{t-s} + \varepsilon_t \tag{2.4}$$

e richiamando la stima dei residui $\hat{e}_t = y_t - \beta x_t$, il test di Dickey-Fuller consiste in una verifica di significatività di ω. Precisamente, data l'ipotesi nulla H_0 e l'alternativa H_1, e ricordando $\omega = \alpha - 1$:

$$
\begin{aligned}
H_0 : \alpha = 1 &\iff H_0 : \omega = 0 \\
H_1 : \alpha < 1 &\iff H_1 : \omega < 0
\end{aligned}
$$

Dopo aver computato i coefficienti con OLS, si calcola la statistica $\tau = \frac{\hat{\omega} - 0}{\hat{\sigma}}$, dove $\hat{\sigma}$ è lo scarto quadratico legato al coefficiente di $\hat{\omega}$ nella regressione. Poichè in questo contesto la distribuzione

asintotica di riferimento non è standard, siamo obbligati a far riferimento a valori critici appositamente computati e presenti in molti libri sull'argomento.
In ogni caso, denotando con τ_c il valore critico, possiamo dedurre:

$$\tau \leq \tau_c \implies \hat{e}_t \text{ è stazionaria}$$
$$\tau > \tau_c \implies \hat{e}_t \text{ non è stazionaria}$$

2.2 Modello per statistical arbitrage tra baskets

L'estensione naturale del modello AR nel contesto multivariato è il VAR. Tramite un modello vettoriale si è in grado di tenere in costante monitoraggio la dinamica evoluzione di molteplici variabili d'interesse.
Si consideri il seguente vettore relativo al processo di prezzo di un insieme di titoli al tempo t: $X_t = [x_{1t}, x_{2t}, ..., x_{dt}]^T$ dove x_{it}, con $i = 1, 2, ..., n$, è la variabile che fa riferimento al prezzo dell'i-esima azione in t. Seguendo i passi svolti nel caso univariato della sezione precedente, non risulta dunque difficile ricondursi ad una scrittura simile a 2.4:

$$X_t = \Theta + AX_{t-1} + \nu_t$$
$$X_t - X_{t-1} = \Theta + AX_{t-1} - X_{t-1} + \nu_t \qquad \text{sottraendo da entrambe le parti } X_{t-1}$$
$$\Delta X_t = \Theta + \Omega X_{t-1} + \nu_t \qquad \text{sostituendo } \Omega = A - \mathbb{I}$$
$$\Delta X_t = \Theta + \Omega X_{t-1} + \sum_{s=1}^{p} \Gamma_s \Delta X_{t-s} + \varepsilon_t \qquad \text{sostituendo } \nu_t \text{ autocorrelato con } p \text{ termini ritardati}$$

dove $\Theta, A, \Omega, \Gamma_s$, per $s = 1, 2, ..., p$ sono matrici $\in \mathbb{R}^{d \times d}$; $\nu_t = [\nu_{1t}, \nu_{2t}, ..., \nu_{dt}]^T$ è il vettore di residui potenzialmente autocorrelati; $\varepsilon_t = [\varepsilon_{1t}, \varepsilon_{2t}, ..., \varepsilon_{dt}]^T$ è il vettore di residui in cui $\varepsilon_t \sim \mathcal{N}(0^{d \times 1}, \Sigma)$, per $i = 1, 2, ..., d$, con Σ matrice delle covarianze di ε_t. Svolgendo tali passaggi, si è passati da una scrittura che ben poco si mostrava adeguata alla descrizione di processi non-stazionari (ma appropriata a quelli stazionari) ad una che si adatta correttamente alla rappresentazione di sistemi non-stazionari in presenza di cointegrazione. L'ultima riscrittura, infatti, prende il nome di VECM, ovvero *Vector Error Correction Model*, la quale sarà a breve resa ancora più esplicita tramite l'individuazione dell'equilibrio del sistema.
Si vuole ora dare una più ampia formalizzazione dei concetti brevemente espressi nella sezione precedente.

Definizione 3 (Lag Operator)**.** È un operatore lineare L tale che, data una serie storica X_t: $LX_t = X_{t-1} \quad \forall t > 1$.

Immediatamente si ricava anche: $L^k X_t = X_{t-k}, \quad k \in \mathbb{Z}$

Definizione 4 (Difference Operator)**.** Si definisce *First Difference Operator* $\Delta X_t := X_t - X_{t-1}$

Sfruttando la definizione 3, si può riscrivere: $\Delta X_t = X_t - X_{t-1} = (1 - L)X_t$. Di conseguenza $\Delta^k X_t = (1 - L)^k X_t, \quad k \in \mathbb{Z}$, si denomina *k-th Difference Operator*. A questo punto risulta conveniente sfruttare il polinomio caratteristico (che viene ricavato nel modo mostrato sotto) del modello VECM per meglio collocare il concetto di integrazione.

$$(1 - L)X_t = \Theta + \Omega L X_t + (1 - L)X_t \sum_{s=1}^{p} \Gamma_s L^s + \varepsilon_t \qquad \text{Riscrivo il VECM con l'ausilio dell'operatore } L$$

$$((1 - L) - \Omega L - (1 - L) \sum_{s=1}^{p} \Gamma_s L^s) X_t = \Theta + \varepsilon_t$$

$$\Omega(L) = (1 - L) - \Omega L - (1 - L) \sum_{s=1}^{p} \Gamma_s L^s \qquad \Omega(L) \text{ è il polinomio caratteristico cercato}$$

Il determinante relativo a $\Omega(L)$ ha al più $(p+1)d$ radici. Se indichiamo con ρ_i le radici di $\det(\Omega(L)) = 0$, è possibile riscrivere $\det(\Omega(L)) = \prod_{i=1}^{(p+1)d}(1 - L\frac{1}{\rho_i})$.

Definizione 5 (Integrazione)**.** Un processo $(X_t)_t$ VAR, il cui determinante ha radici ρ_i unitarie oppure al di fuori del disco di raggio $R = 1$ è detto integrato di ordine d, ovvero $I(d)$, $\iff$ $\Delta^d X_t$ è stazionario ma $\Delta^{d-1} X_t$ non lo è.

Definizione 6 (Cointegrazione)**.** Un processo $(X_t)_t > 0$ VAR è detto cointegrato di ordine (d, b), ovvero $CI(d, b)$ $\iff$ esiste almeno un vettore $\beta = [\beta_1, \beta_2, ..., \beta_d]^T \neq 0^{d\times 1}$, $\beta \in \mathbb{R}^{d\times 1}$, tale che $\beta^T X_t$ sia $I(d - b)$.

L'interesse è rivolto al caso in cui le serie x_{it} siano $I(1)$. Dunque l'obiettivo del contesto in esame risulta ottenere che $\beta^T X_t$ sia stazionario.
Più in dettaglio, se il polinomio caratteristico $\Omega(L)$ presenta una radice unitaria, allora $\Omega(1) = -\Omega$ è singolare, di rango r, dove $0 \leq r < d$; di conseguenza l'intero processo non è stazionario. Tuttavia, ogni matrice di rango ridotto è scomponibile nel seguente modo: $\Omega = \alpha\beta^T$, dove $\alpha, \beta \in \mathbb{R}^{d\times r}$. Si giunge dunque al seguente modello:

$$\Delta X_t = \Theta + \alpha\beta^T X_{t-1} + \sum_{s=1}^{p} \Gamma_s \Delta X_{t-s} + \varepsilon_t \tag{2.5}$$

Grazie al *Granger Representation Theorem*, che non sarà enunciato in questa sede, seppur se ne forniscano adeguate fonti d'approfondimento in [5], ci è permesso legare univocamente l'esistenza di un sistema cointegrato ad una rappresentazione tramite VECM di serie integrate. In tale contesto, nonostante il fatto che X_t sia un random walk per la presenza di almeno una radice unitaria, il suddetto teorema aiuta ad identificare in $\{X \mid \beta^T X = 0\} = span\{\beta\perp\}$ l'equilibrio dinamico di lungo termine, ossia l'*attrattore*[9] del sistema considerato. Per $\beta \in \mathbb{R}^{d\times r}$, si ha che $\{X \mid \beta^T X = 0\}$ delinea un iperpiano di dimensione $d - r$. La matrice α, invece, costituisce la cosiddetta matrice di *loadings*, la quale descrive come il modello reagisce a deviazioni di breve termine del sistema rispetto all'attrattore, innescate dalle dinamiche di Γ_s.

2.2.1 Regressione di Engle-Granger & Test di Johansen

Mentre nel caso univariato il test evidenziava due soli esiti ammissibili, ora si aprono molti casi intermedi:

- $r = 0$: $\Omega = 0^{d\times d}$ per cui X_t è un random walk multivariato e non sussiste cointegrazione
- $r = n$: X_t non è $I(1)$, bensì $I(0)$, poichè Ω ha rango massimo
- $0 < r < n$: X_t è un sistema cointegrato

L'interesse è rivolto a quest'ultimo caso. Inoltre, in questa trattazione si ritiene che il livello di cointegrazione più consistente sia raggiungibile se ogni titolo è cointegrato con ognuno degli altri costituenti il sistema da studiare. Ciò è ottenibile solamente nel caso di unico vettore di cointegrazione, vale a dire $\beta \in \mathbb{R}^{d\times 1}$. A tal fine ci si concentrerà sulla costruzione di sistemi con matrice Ω di rango $r = 1$.
La stima di un vettore β di un VECM è resa possibile dal metodo suggerito da Engle e Granger, che consiste nel regredire tramite OLS una serie x_{it}, con $i \in \{1, 2, ..., d\}$, su tutte le altre serie univariate che costituiscono il vettore X_t. Non è detto che la prima serie scelta sia la più adatta a mostrare la cointegrazione sperata. Si ripeterà il tentativo di regressione prendendo a turno le altre x_{j_t}, $j \neq i$, come riferimento. Il controllo del p-value più basso dovrebbe denotare la miglior scelta per il caso in esame.
Ovviamente la metodologia appena enunciata è applicabile nel caso si sia già dimostrato con

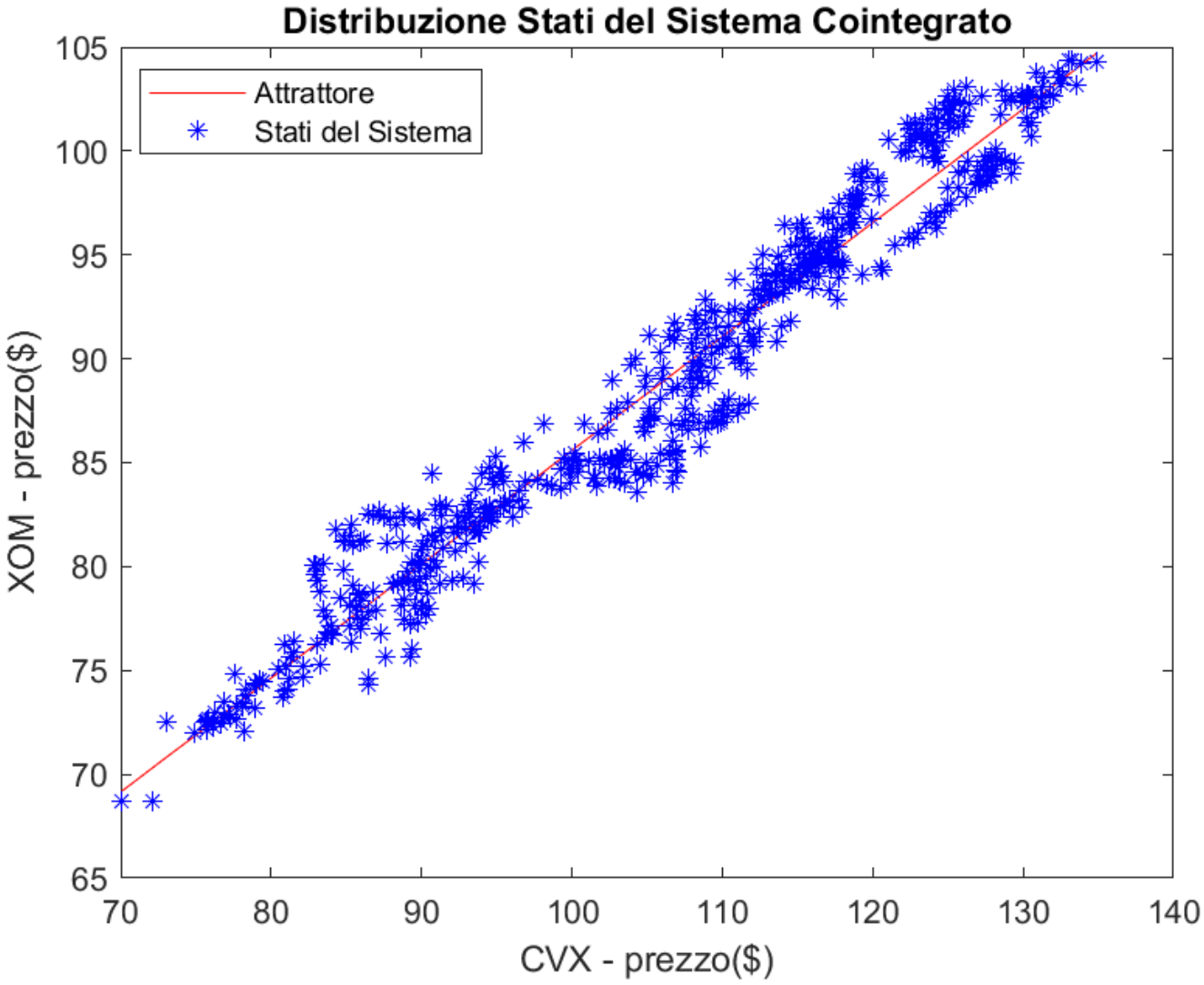


Figura 2.1 – Esempio di *Distribuzione Stati del Sistema Cointegrato* costituito da *Chevron Corp (CVX)* e *Exxon Mobil Corp (XOM)*: inizio=01/01/2014; fine=01/04/2016; l'attrattore è costituito da una retta.

sufficiente affidabilità che $r = 1$. A questo scopo interviene il test di Johansen.
L'ideatore, Søren Johansen, propone due test simili, ma reputa comunque più attendibile uno dei due, il cosiddetto *Trace Test*. In maniera intuitiva, il suddetto test verifica la significatività degli autovalori di una matrice semidefinita positiva (così che gli autovalori siano reali non negativi) che ha lo stesso rango di Ω. Si opera un test di azzeramento degli autovalori $\hat{\lambda}_i$, $i = 1, 2, ..., d$, in ordine dal più piccolo al più grande. Si inizia quindi a testare il più piccolo, ovvero $\hat{\lambda}_d$. Se si rifiuta l'ipotesi nulla $H_0 : \hat{\lambda}_d = 0$ (in quanto $\hat{\lambda}_d$ è positivo), allora tutti gli autovalori sono positivi, per cui Ω ha rango massimo. Invece, se non si rifiuta H_0, si procede con test successivi. Partendo da $r^* = d-2$ e continuando (nel caso di non rifiuto di H_0) con $r^* = d-3, ..., 1$, si effettua un test di azzeramento congiunto dei più piccoli $d - r^*$ autovalori:

- $H_0 : r^* = r < d$ se non si rifiuta H_0, si continua con r^* successivo
- $H_1 : r^* = r = d$ se si rifiuta H_0, ci si ferma e si prende $r = r^* + 1$ come rango stimato

La statistica test da utilizzare ad ogni step successivo è:

$$J_{trace} = -N \sum_{i=r^*+1}^{d} \log(1 - \hat{\lambda}_i)$$

dove N è la numerosità campionaria. La distribuzione asintotica dei valori critici di riferimento è non-standard; questi ultimi si ricavano mediante approssimazioni numeriche.

2.3 Gestione ottima dei costi di transazione

In un contesto realistico di arbitraggio statistico, caratterizzato da un'operatività a medio-alta frequenza e dalla presenza di molteplici titoli in portafoglio, la gestione dei costi di transazione impatta in maniera consistente sulla performance. Basandosi sul lavoro di Almgren-Chriss in [1], identifichiamo, in questa sede, quali siano gli elementi di maggior rilievo presenti nella loro trattazione, cercando di sottolineare gli aspetti che, potenzialmente, hanno maggior influenza sui nostri modelli di arbitraggio.
Tuttavia, per mancanza di infrastruttura adeguata, non procederemo all'implementazione dei modelli di ottimizzazione dei costi; peraltro, essi non risulterebbero particolarmente efficaci nel backtest da noi effettuato, in quanto esso opera con una frequenza giornaliera, come si leggerà nel capitolo successivo. Tuttavia, si ritiene che, operando con frequenze più elevate e con strategie di arbitraggio più sofisticate e precise di quelle da noi adottate, i modelli di Almgren-Chriss possano complessivamente garantire una diminuzione consistente dei costi associati.
Come si deduce dalle prime righe di [1], il documento è finalizzato a ottimizzare il processo di transizione di un portafoglio da una composizione iniziale ad una finale in un determinato lasso temporale, sia che ciò comporti vendita o acquisto di titoli. In particolare, l'attenzione è rivolta allo studio dell'impatto sul prezzo di mercato derivante dalla vendita/acquisto di grossi blocchi di titoli. Il focus sulla liquidità è dunque preponderante: elevati volumi di vendita su azioni ritenute poco liquide possono portare a rapide diminuzioni di prezzo e conseguente minor "incasso" rispetto all'attesa; viceversa, un simile controvalore di vendita, nel caso di azioni molto liquide, può avere un impatto quasi trascurabile sul prezzo di riferimento. Per tale motivo, la trattazione si concentra sulla minimizzazione dei costi, associando una opportuna penalità al rischio/incertezza. Si consegue ciò costruendo, per un dato livello di tolleranza al rischio, una frontiera efficiente che ricalchi la strategia ottima per minimizzare i costi.
Modelliamo le dinamiche di prezzo come dei random walk con incrementi indipendenti. Tale indipendenza dei rendimenti permette che una strategia statica risulti ottima anche nel caso l'operatore decida di modificare il suo programma, già avviato, di vendite/acquisti.
In primo luogo ripercorriamo i passaggi necessari a costruire le frontiere efficienti nel caso di un'unica azione; solo in seguito si delineerà un'estensione ad un portafoglio di titoli. Senza perdita di generalità consideriamo, dunque, un programma di liquidazione per un numero di azioni X (di uno specifico titolo) da compiersi entro un tempo futuro T. Dividiamo T in un numero finito N di intervalli di lunghezza $\tau = \frac{T}{N}$ e sia $t_k = k\tau$ dove $k = 0, ..., N$.

Definizione 7 (traiettoria di trading)**.** Una *traiettoria di trading* è un vettore $[x_0, ..., x_N]^T$ dove x_k è il numero di azioni di X che abbiamo in programma di detenere in portafoglio al tempo t_k.

Inizialmente si ha $x_0 = X$; alla fine $x_N = 0$. Sia $n_k = x_{k-1} - x_k$ il numero di azioni vendute tra t_{k-1} e t_k. Allora $x_k = X - \sum_{j=1}^{k} n_j = \sum_{j=k+1}^{N} n_j, \quad k = 0, ..., N$.

Definizione 8 (strategia di trading)**.** Una *strategia di trading* è una regola per determinare n_k in termini dell'informazione disponibile al tempo t_{k-1}.

Identifichiamo la volatilità come una forza esogena che agisce in maniera casuale e indipendente dalle nostre operazioni; invece, guardiamo all'impatto sul mercato come ad un fattore endogeno. Distinguiamo tra due tipi di impatto derivante dalle nostre operazioni:

- impatto permanente : consiste in cambiamenti dell'equilibrio del prezzo del titolo che si protraggono anche in seguito alla nostra completa liquidazione.

- impatto temporaneo : fa riferimento a temporanei squilibri tra domanda e offerta che possono momentaneamente deviare il prezzo dell'asset dall'equilibrio.

Assumiamo che il prezzo dell'asset evolva nel seguente modo:

$$S_k = S_{k-1} + \sigma\tau^{1/2}\xi_k - \tau g\left(\frac{n_k}{\tau}\right) \tag{2.6}$$

dove $k = 1, ..., N$; σ rappresenta la volatilità del titolo; ξ_j sono variabili i.i.d. con media 0 e varianza unitaria; $g(v)$ è funzione proporzionale al livello medio di azioni vendute nell'intervallo tra t_{k-1} e t_k e fa riferimento all'impatto permanente. Con $h(v)$, invece, designeremo l'impatto temporaneo: esso influenza solo il prezzo ricavato dalla vendita all'istante successivo, ma il suo effetto non è in grado di modificare S_k:

$$\tilde{S_k} = S_{k-1} - h\left(\frac{n_k}{\tau}\right) \tag{2.7}$$

dove con $\tilde{S_k}$ identifichiamo il vero prezzo a cui avviene la vendita. Dunque, se si volesse riassumere l'incasso complessivo una volta terminato l'intero processo di liquidazione:

$$\sum_{k=1}^{N} n_k\tilde{S_k} = XS_0 + \sum_{k=1}^{N}\left(\sigma\tau^{1/2}\xi_k - \tau g\left(\frac{n_k}{\tau}\right)\right)x_k - \sum_{k=1}^{N} n_k h\left(\frac{n_k}{\tau}\right) \tag{2.8}$$

Nella riscrittura sopra si può riconoscere XS_0 come valore iniziale in \$ dell'asset, tenendo conto della quantità detenuta; $\sum_{k=1}^{N}\sigma\tau^{1/2}\xi_k x_k$ si riferisce alla variazione di prezzo del titolo dovuta alle fluttuazioni di mercato; infine, i due termini $-\sum_{k=1}^{N}\tau g\left(\frac{n_k}{\tau}\right)x_k - \sum_{k=1}^{N} n_k h\left(\frac{n_k}{\tau}\right)$ sono indicativi dei costi sostenuti, globalmente, per le transazioni eseguite, incorporando entrambi gli impatti sopra considerati. Per cui ci si attende che i costi totali siano, in valore assoluto, pari a:

$$\mathbb{E}[x] = \sum_{k=1}^{N}\tau g\left(\frac{n_k}{\tau}\right)x_k + \sum_{k=1}^{N} n_k h\left(\frac{n_k}{\tau}\right) \tag{2.9}$$

mentre la varianza ad essi associata è:

$$\mathbb{V}ar(x) = \sigma^2\sum_{k=1}^{N}\tau {x_k}^2 \tag{2.10}$$

Non sarebbe necessario, ma per trattare con maggior facilità le equazioni di interesse, ipotizziamo, in maniera assolutamente plausibile, che le funzioni di impatto siano lineari nel proprio argomento. Ne deriverebbe, per l'impatto permanente:

$$g(v) = \gamma v \tag{2.11}$$

mentre per l'impatto temporaneo:

$$h\left(\frac{n_k}{\tau}\right) = \varepsilon sign(n_k) + \frac{\eta}{\tau}n_k \tag{2.12}$$

In 2.12 una stima per ε è rappresentata dal costo fisso di vendita (ad esempio metà dello spread tra bid e ask in aggiunta ad altre spese fisse); η, invece, è più difficile da quantificare potendo essere associata ad aspetti non-lineari caratterizzanti la microstruttura del mercato. Tramite

semplificazioni algebriche:

$$\begin{aligned}\sum_{k=1}^{N}\tau g\left(\frac{n_k}{\tau}\right)x_k &= \gamma\sum_{k=1}^{N}x_k n_k = \gamma\sum_{k=1}^{N}x_k(x_{k-1}-x_k)\\ &= \frac{1}{2}\gamma\sum_{k=1}^{N}\left({x_{k-1}}^2-{x_k}^2-(x_k-x_{k-1})^2\right)\\ &= \frac{1}{2}\gamma X^2-\frac{1}{2}\gamma\sum_{k=1}^{N}{n_k}^2\end{aligned}$$

Introducendo $\tilde{\eta}=\eta-\frac{1}{2}\gamma\tau$, si ottiene:

$$\mathbb{E}[x]=\frac{1}{2}\gamma X^2+\varepsilon\sum_{k=1}^{N}|n_k|+\frac{\tilde{\eta}}{\tau}\sum_{k=1}^{N}{n_k}^2 \tag{2.13}$$

Ora consideriamo:

Definizione 9 (strategia ottima)**.** Una strategia si definisce ottima se non ne esiste un'altra tale che, a parità (o minor) di varianza, abbia inferiori costi di transazione attesi.

Mettendo assieme i risultati per il valore atteso 2.13 e per la varianza 2.10, e tenendo a mente la definizione 9, è possibile impostare il problema di minimizzazione vincolato:

$$\min_{x|\mathbb{V}ar(x)\leq V^\star}\mathbb{E}[x] \tag{2.14}$$

Essendo $V^\star$ convesso, anche $\{\mathbb{V}ar(x)\leq V^\star\}$ è convesso; dal momento che $\mathbb{E}[x]$ è strettamente convesso, ne consegue che $\exists!x^\star$ tale che realizzi 2.14. Risolviamo esplicitamente il problema di minimizzazione vincolato introducendo un moltiplicatore di Lagrange λ:

$$\min_x\left(\mathbb{E}[x]+\lambda\mathbb{V}ar(x)\right) \tag{2.15}$$

dove λ è anche interpretabile come misura di avversione al rischio nel contesto di una funzione di utilità, ovvero quanto si penalizza la varianza in relazione al valore atteso. Data la convessità di $U(x):=\mathbb{E}[x]+\lambda\mathbb{V}ar(x)$ per $\lambda>0$, imponiamo:

$$\frac{\partial U}{\partial x_j}=2\tau\left\{\lambda\sigma^2x_j-\tilde{\eta}\frac{x_{j-1}-2x_j+x_{j+1}}{\tau^2}\right\}=0$$

j=1,...,N-1. Tale condizione per la ricerca del minimo globale equivale a scrivere:

$$\frac{1}{\tau^2}\left(x_{j-1}-2x_j+x_{j+1}\right)=\tilde{\kappa}^2x_j \tag{2.16}$$

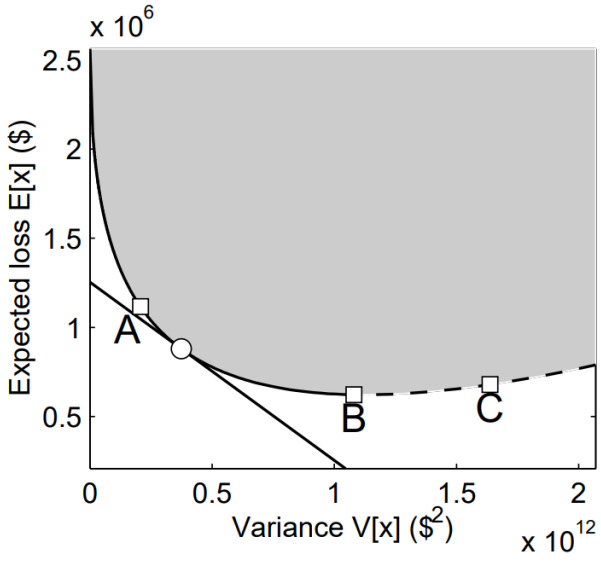
dove $\tilde{\kappa}^2=\frac{\lambda\sigma^2}{\tilde{\eta}}$. Senza dilungarsi nei passaggi necessari ad ottenere la forma alquanto compatta di Almgren-Chriss, presentiamo la soluzione:

$$x_j=\frac{\sinh\left(\kappa\left(T-t_j\right)\right)}{\sinh(\kappa T)}X\quad j=0,...,N \tag{2.17}$$

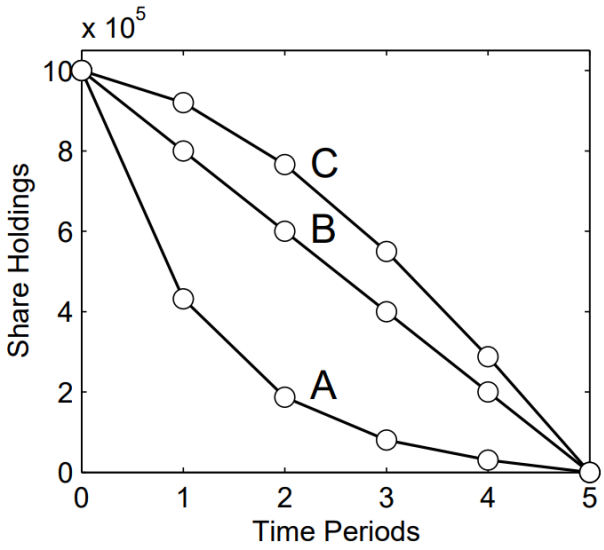
Similmente:

$$n_j=\frac{2\sinh\left(\frac{1}{2}\kappa\tau\right)}{\sinh(\kappa T)}\cosh\left(\kappa\left(T-\left(j-\frac{1}{2}\right)\tau\right)\right)X\quad j=1,...,N \tag{2.18}$$

Dunque 2.17 e 2.18 permettono di tracciare le traiettorie ottime per ogni profilo di rischio, relativamente alla specifica attitudine dell'investitore.
Riferendosi alle figure in 2.2, evidenziamo le differenze tra le strategie adottate:



(a) *Frontiera efficiente* mostrante il livello di valore atteso e varianza ottimo per tre possibili strategie A,B,C.



(b) *Traiettoria ottima* mostrante le quantità n_k da liquidare ad ogni istante t_k per tre possibili strategie A,B,C.

Figura 2.2 – Esempio di *Frontiera efficiente* e *Traiettoria ottima* per tre possibili strategie: A ($\lambda = 2 \cdot 10^{-6}$), B ($\lambda = 0$), C ($\lambda = -2 \cdot 10^{-7}$)

- A : rispecchia un profilo avverso al rischio in cui, pur di ridurre l'incertezza cui si è esposti, si preferisce incorrere in costi attesi più elevati, liquidando rapidamente la maggior parte delle quote di titolo azionario possedute.
- B : corrisponde ad un profilo neutrale al rischio in cui si liquidano gradualmente le azioni detenute. Precisamente prevede che ad ogni istante t_k vengano vendute un uguale numero di azioni pari a $n_k = \frac{X}{N}$. Ciò corrisponde ad attuare la strategia che minimizza il costo atteso complessivo.
- C : rispecchia un profilo propenso al rischio in cui la vendita viene ritardata, incorrendo sia in un'elevata incertezza, sia in costi attesi cospicui dato che ci si ritrova a liquidare rapidamente verso la fine del periodo di riferimento.

Nella figura a sinistra di 2.2, il punto di intersezione tra la linea tangente e la frontiera fa riferimento ad un profilo "leggermente" avverso al rischio con $\lambda = 10^{-6}$.

Tuttavia, un'importante considerazione deve essere condotta in relazione al punto B. Certamente in esso si ha la minimizzazione dei costi attesi; nonostante ciò, muovendosi in un intorno di tale minimo globale, si ottiene un risultato ben più interessante.
Siano $(\mathbb{E}_0, \mathbb{V}_0)$ le coordinate del punto che, relativamente alla figura 2.2, corrisponde a B. Sviluppiamo la funzione $\mathbb{E}(\mathbb{V})$ con Taylor centrato in $\mathbb{E}_0$. Dal momento che $\frac{d\mathbb{E}}{d\mathbb{V}}\big|_{\mathbb{V}=\mathbb{V}_0} = 0$, abbiamo:

$$\mathbb{E} - \mathbb{E}_0 \sim \frac{1}{2}(\mathbb{V} - \mathbb{V}_0)^2 \frac{d^2\mathbb{E}}{d\mathbb{V}^2}\bigg|_{\mathbb{V}=\mathbb{V}_0} \tag{2.19}$$

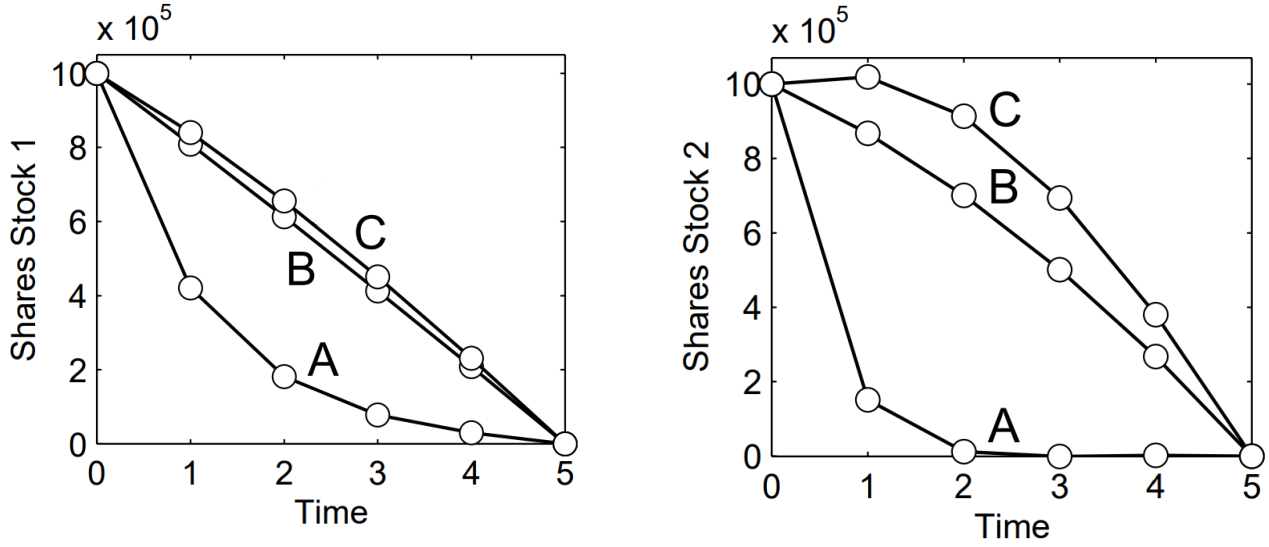
dove $\frac{d^2\mathbb{E}}{d\mathbb{V}^2}\big|_{\mathbb{V}=\mathbb{V}_0} > 0$ per la convessità della frontiera. Si deduce che una diminuzione della varianza al primo ordine è attuabile incorrendo solo in un aumento dei costi al secondo ordine. Di conseguenza, si ritiene sia sempre vantaggioso impiegare una strategia che rispecchi un profilo almeno "leggermente" avverso al rischio.

Come annunciato in precedenza, estendiamo i concetti sopra appresi al caso di un portafoglio contenente m titoli azionari diversi. Sia $x_k = [x_{1k}, ..., x_{mk}]^T$ il vettore del numero di quote detenute per ogni asset al tempo t_k. Inoltre, $x_{jk} < 0$ significa che abbiamo una posizione short sul titolo j; similmente, se $n_{jk} = x_{k-1} - x_k < 0$, allora stiamo comprando l'asset j tra t_{k-1} e t_k. Similmente a 2.6, assumiamo che S_k segua un random walk multivariato privo di drift. Sia $C \in \mathbb{R}^{m\times m}$ la matrice delle covarianze. Considerando un modello con impatti lineari, scriviamo:

$$g(v) = \Gamma v$$

$$h(v) = \varepsilon sign(v) + Hv$$

dove $\Gamma, H \in \mathbb{R}^{m\times m}$ e $\varepsilon \in \mathbb{R}^{m\times 1}$. L'elemento ij di Γ, H rappresenta l'impatto sul prezzo dell'asset i causato dalla vendita del titolo j ad un tasso unitario.
Come precisamente presentato in [1], tramite opportuni raccoglimenti, cambi di variabili e riscritture, si ottengono delle formule per $\mathbb{E}[x]$ e per $\mathbb{V}ar(x)$ ed infine, risolvendo l'analogo problema di minimizzazione, si giunge ad una forma esplicita per x_{jk}.



(a) *Traiettoria ottima* per il titolo *stock 1* nel caso di tre possibili strategie A,B,C.

(b) *Traiettoria ottima* per il titolo *stock 2* nel caso di tre possibili strategie A,B,C.

Figura 2.3 – Esempio di *Traiettoria ottima* in un portafoglio con due asset distinti nel caso di tre possibili strategie: A ($\lambda = 2 \cdot 10^{-6}$), B ($\lambda = 0$), C ($\lambda = -5 \cdot 10^{-8}$)

Ciò che preme sottolineare è rappresentato dal fatto che, in un portafoglio, più i titoli sono correlati tra loro e più le traiettorie ottime sono simili.
Dalle figure in 2.3, inoltre, risulta lampante come la liquidità sia il fulcro del lavoro di Almgren-Chriss. Rispetto a *stock 1*, infatti, *stock 2* è un titolo più liquido e meno volatile. Ecco perchè il primo asset necessiti di essere venduto in maniera molto più graduale rispetto al secondo, in modo da evitare di impattare eccessivamente il mercato.

Capitolo 3

Setup & Backtest

Di seguito si intende fornire una accurata descrizione del funzionamento dell'algoritmo di trading vero e proprio, senza doversi troppo dilungare sulla trattazione dei metodi di clustering o dei modelli, a cui sono stati dedicati i due capitoli precedenti. Perciò ci si soffermerà sugli aspetti che sono stati accennati in precedenza, mentre si darà solo menzione degli argomenti già approfonditi. L'algoritmo è stato integralmente scritto in Python e, per poter effettuare il backtest, si è proceduto alla stesura del codice all'interno della piattaforma di QuantConnect servendosi della libreria messa da loro a disposizione. Operare in questo modo ha inoltre consentito di avere accesso a dati azionari affidabili, senza la necessità di reperirli, talvolta imprecisi, da fonti terze.
Nelle sezioni successive si tenterà di esplicitare i passi fondamentali di cui è costituito il codice, distinguendo, in primo luogo, tra i metodi richiesti dalla piattaforma per condurre il backtest e quelli utilizzati, invece, per supportare il lavoro di costruzione dei modelli in esame. Si cercherà, dunque, di rendere quanto più comprensibile la logica dell'algoritmo leggendo il solo capitolo.
Il codice è facilmente accessibile tramite il seguente link github: https://github.com/DavideRoznowicz/Trading_Models

3.1 Funzionamento backtest su piattaforma

Per condurre il backtest è obbligatorio operare all'interno di una classe predisposta da QuantConnect dotata di due metodi la cui presenza è indispensabile. La prima non deve avere alcun nome particolare, perciò la si denoterà semplicemente con il nome del modello; i metodi, invece, al fine di essere riconosciuti correttamente, dovranno chiamarsi `Initialize` e `OnData`. Al momento dell'esecuzione dell'algoritmo, viene creata un'istanza della classe relativa allo specifico backtest utilizzando come costruttore il metodo `Initialize`. Quest'ultimo conterrà perciò gli attributi che l'oggetto deve possedere fin dall'inizio; alcuni di essi non sono visibili, ma pur sempre accessibili: vengono dichiarati dalla piattaforma per aggregare i dati, facilitandone la reperibilità all'interno della classe. Inoltre, risulta necessario definire, tramite alcuni comandi interni, certi parametri chiave su cui si innesca il backtest. Tra tali comandi, si annoverano:

- `SetStartDate(anno, mese, giorno)` : definisce l'inizio del backtest.
- `SetEndDate(anno, mese, giorno)` : definisce il termine del backtest.
- `SetCash(denaro)` : indica la quantità di denaro di partenza. Nonostante il fatto che il nostro sia un portafoglio long-short, ovvero mediamente tenda a finanziare le posizioni long con il denaro ricavato dalle posizioni short, vogliamo evitare di costruire posizioni eccessivamente a leva. Perciò, per scelta, si terrà come riferimento il denaro impostato in `SetCash(denaro)`: per ogni posizione aperta su un titolo, sia long o short, imponiamo che la somma dei valori assoluti del denaro dedicato ad ogni posizione non possa superare tale soglia prestabilita.

- `AddEquity(ticker, frequenza)` : viene aggiunta all'istanza dell'algoritmo un riferimento al fatto che il titolo, legato al *ticker* in input, debba essere incluso nell'insieme di azioni su cui poter operare. Conseguentemente viene predisposta la costruzione di un opportuno oggetto contenente informazioni riguardanti il titolo aggiunto. Inoltre, con il secondo parametro in input si danno informazioni sulla frequenza dei dati che si vogliono avere a disposizione nell'oggetto di cui si è appena parlato; in particolar modo, si stabilisce su quale scala si intende effettuare il backtest (giornaliera, oraria, a minuti, ...).

Nel nostro caso, all'interno di `Initialize` è anche utile definire aggiornamenti periodici, secondo una determinata frequenza (potenzialmente diversa da quella definita con AddEquity, ma nel nostro caso uguale, ovvero giornaliera), dei dati raccolti nella rolling window senza dover integralmente ricaricarli tutti ad ogni passo del backtest (e quindi ogni giorno nel caso in esame): ciò è conseguibile tramite la funzione accessoria `def OnDataConsolidated(self, sender, bar)`. Precisamente, data una rolling window di n periodi, viene eliminata l'informazione più datata e caricata quella più recente, non appena essa si sia resa disponibile.
Per poter costruire la matrice dei prezzi si rende necessario tenere traccia dei dati della rolling window relativa ad ogni titolo: tale processo è reso molto più agile attraverso una classe implementata allo scopo di questa trattazione che, insieme ad altre, verrà presentata nella prossima sezione.
Avendo `def Initialize(self)` uno scopo simile al metodo built-in di Python per il costruttore (`__init__`), il suo unico input deve essere l'istanza stessa (`self`). Mentre tale funzione viene dunque "utilizzata" solo una volta, all'inizio, per definire l'ambiente di partenza ed i vari aggiornamenti da effettuare con la frequenza prestabilita, `def OnData(self, data)`, invece, è costantemente richiamata (ogni giorno): essa agisce modificando l'istanza e gli oggetti creati dalla piattaforma con lo scopo di raccogliere statistiche descrittive riguardanti il backtest; l'input `data` è un oggetto aggiornato internamente che dà la possibilità di impiegare dati continuamente aggiornati concernenti i titoli precedentemente aggiunti tramite `AddEquity`. Principalmente in `OnData` avviene la costruzione effettiva della strategia, con l'individuazione dei titoli su cui operare e la successiva immissione degli ordini di mercato, insieme alla gestione delle posizioni aperte.

3.2 Step fondamentali per la costruzione del modello

Oltre alla classe e ai metodi obbligatori, ne sono stati implementati altri per raggiungere i nostri specifici obiettivi di backtest.
L'intera effettiva strategia risulta scritta all'interno di `def OnData(self, data)`, in cui la costruzione delle specifiche fasi è resa più agile dall'utilizzo di dizionari e oggetti generati dalle seguenti classi appositamente realizzate:

- `class SymbolData(object)` : facilita l'utilizzo delle serie di dati di particolare interesse per il backtest in oggetto.

- `class Cluster(object)` : definisce un'entità Cluster rendendo più agevole la distinzione dei vari cluster e di ciò che contengono dopo aver eseguito *k-means++*.

- `class TradableGroup(object)` : gestisce sottogruppi di titoli in cui si è verificata cointegrazione: contiene varie informazioni ricavate dall'elaborazione dei dati, comprese le caratteristiche della combinazione lineare generata. Comprende, inoltre, varie funzioni accessorie per il ranking e l'individuazione di segnali.

- `class PositionManagement(object)` : permette di gestire le posizioni attualmente aperte e di monitorare i sottogruppi che manifestano cointegrazione ma non sono ancora una posizione in portafoglio.

Le sottosezioni seguenti consentiranno di vedere le fasi fondamentali della strategia, accostando alle spiegazioni di queste un grado di dettaglio maggiore sulle classi appena menzionate. Per scelta, in questo documento si mostreranno esplicitamente i codici di tali classi, ma non delle fasi di `OnData`, le quali saranno semplicemente descritte. Infatti, una volta compreso il funzionamento delle classi, si ritiene non sia particolarmente complicato capire le varie fasi, le quali si riducono ad un utilizzo piuttosto estensivo delle classi stesse, in modo da avere sempre a disposizione i dati di interesse. Si consiglia nuovamente, dunque, di leggere le spiegazioni delle fasi guardando al codice disponibile al link github precedentemente indicato, non appena si avesse qualche perplessità in merito all'implementazione.

3.2.1 Fase 1 : Aggregazione dati

In `Initialize` è presente una lista di *ticker* relativa ai titoli che vogliamo facciano parte del nostro insieme di partenza, ovvero quelli cui poter attingere in ogni momento (tale insieme si denota *Universe*). Essi sono stati generati separatamente per semplicità; corrispondono a poco meno di 500 azioni americane a più alto volume giornaliero scambiato e con prezzo superiore a $ 10 relativamente al giorno di inizio del backtest. Sono proprio tali titoli ad esser poi "aggiunti" con il comando `AddEquity`.

```
class SymbolData(object):

    def __init__(self, symbol, barPeriod, windowSize):
        self.Symbol = symbol    # oggetto Symbol predisposto da
        # QuantConnect: contiene vari dati del titolo (ad esempio il
        # ticker relativo al codice identificativo univoco)
        self.BarPeriod = barPeriod    # Frequenza della rolling window :
        # 1 giorno
        self.Bars = RollingWindow[TradeBar](windowSize)    # oggetto
        # RollingWindow (TradeBar: tipo di QuantConnect)
        self.PriceSeries = None    # serie dei prezzi
        self.ReturnSeries = None    # serie dei rendimenti normalizzati

    # verifica se l'oggetto Bars (e quindi la rolling window) sia stato
    # aggiornato: restituisce un boolean
    def IsReady(self):
        return self.Bars.IsReady    # IsReady: attributo interno di Bars
        # che verifica l'aggiornamento dell'oggetto

    # verifica che l'aggiornamento sia appena avvenuto
    def WasJustUpdated(self, current):
        return self.Bars.Count > 0 and self.Bars[0].Time == current - \
               self.BarPeriod
```

Tramite `class SymbolData(object)` è piuttosto facile tenere traccia delle serie di dati relative ad un singolo titolo, in modo da poter essere riutilizzate successivamente. La necessità è, tuttavia, quella di avere a disposizione le serie relative a tutti i titoli dello *Universe*: a questo scopo si è introdotto un dizionario chiamato `Data` che ad un codice identificativo (key) fa corrispondere un oggetto `SymbolData` (value) relativo ad una specifica azione. In questo modo risulta agevole richiamare i dati di qualunque titolo tramite tale codice. Quest'ultimo non è altro che un'identificazione univoca dell'asset predisposta da QuantConnect che non subisce variazioni nemmeno se la società quotata decide di cambiare appunto *ticker*. Ad esempio, per la compagnia *Apple inc.* il ticker relativo è "AAPL" mentre il codice identificativo assume la forma "AAPL R735QTJ8XC9X".
Nella fase 1, di conseguenza, ci si riduce ad utilizzare il dizionario `Data` per costruire una matrice

X che contenga nelle righe le serie dei prezzi di chiusura di tutti i titoli dello *Universe*. Dopodichè la si modifica, convertendola in matrice di rendimenti; infine, si procede a normalizzare ogni sua riga sottraendo la media e dividendo per lo scarto quadratico.

3.2.2 Fase 2 : Clustering e gestione clusters

Dopo aver applicato *k-means++* su X (con 20 clusters) e aver perciò ottenuto il numero di clusters prestabilito, si procede costruendo una matrice X1 per ognuno di essi. Per cui, per ogni cluster specifico, X1 contiene le sole righe di X relative ai titoli che tale cluster contiene. Si esegue nuovamente *k-means++* su X1 al fine di ottenere sottogruppi di dimensione molto ridotta ($\approx$ 3-4 elementi per pairs-trading, $\approx$ 5-6 per statistical arbitrage) al cui interno si verificherà la presenza di titoli cointegrati, altrimenti si passerà subito a controllare il sottogruppo generato successivo. Questa seconda applicazione dell'algoritmo di clustering ha l'obiettivo di facilitare il processo di testing successivo.

```
class Cluster(object):

    def __init__(self):
        self.SymbolElements = None    # simboli (ovvero "codici ID")
        # dei titoli che appartengono ad un medesimo cluster
        self.SubGroups = {}    # dizionario di sottogruppi definiti
        # dalla classe TradableGroup

    def NumberOfElements(self):
        return len(self.SymbolElements)
```

Ogni istanza `Cluster()`, quindi, permette di raccogliere la lista di "codici ID" relativi ad ogni azione inclusa; principalmente, all'interno del dizionario `SubGroups` si raduneranno (come value) oggetti di tipo `TradableGroup()` (descritti in seguito) di titoli che avranno manifestato cointegrazione.

3.2.3 Fase 3 : Test e raccolta sottogruppi

A questo punto, all'interno di ogni sottogruppo individuato, si eseguono i test descritti nel capitolo 2:

- Test di Dickey-Fuller : si effettua nel caso il setting del backtest sia quello per il pairs-trading. Si procede individuando l'elemento corrispondente al centroide: quindi si verifica la cointegrazione di questo con un altro titolo del sottogruppo. Se la coppia esibisce cointegrazione, viene accettata e le sue statistiche descrittive di rilievo vengono raccolte in un'istanza `TradableGroup()`; altrimenti si passa a verificare la cointegrazione del primo elemento con un altro ancora appartenente al sottogruppo, nel caso ne siano rimasti.

- Test di Johansen : si effettua nel caso il setting del backtest sia quello per lo statistical arbitrage. Si procede individuando come elemento di partenza quello corrispondente al centroide: si verifica che il sistema costituito da una coppia di azioni abbia matrice di rango 1; altrimenti si riprova con un altro titolo del sottogruppo. Nel caso in cui sia stata evidenziata cointegrazione nella coppia di elementi, bisogna assicurarsi che il sistema costituito dalla coppia e da un altro elemento del sottogruppo sia cointegrato con matrice di rango 1. Continuando in questo modo, si accetta e memorizza l'insieme di elementi ottenuto se il sistema comprende almeno 3 titoli e non più di 4 (se si raggiungono 4 elementi la ricerca viene interrotta in quanto non si intende avere troppi elementi).

L'unica differenza tra l'algoritmo di backtesting per pairs-trading rispetto a quello per statistical arbitrage è rappresentato, appunto, dalla modalità con cui viene condotta la terza fase.

```
# Gestione sottogruppi di azioni (coppie se Pairs-Trading) su cui si puo'
# effettivamente operare analizzando le serie delle opportune
# combinazioni lineari;
# La classe seguente NON e' completa: mancano alcuni metodi che
# saranno presentati nelle sottosezioni successive
class TradableGroup(object):

    def __init__(self):
        self.SymbolTradableElements = None  # simboli dei titoli su cui
        # si puo' effettivamente fare trading: basta attendere il
        # segnale giusto
        self.Coeff = None  # coefficienti della combinazione lineare
        # generata
        self.Mu = None  # media dei residui
        self.Sigma = None  # deviazione standard dei residui
        self.Res = None  # serie stimata dei residui (== combinazione
        # lineare generata)
        self.Score = None  # score tramite ZeroCrossing(self, limit)
        self.Direction = None  # direzione del segnale sulla combinazione
        # lineare ("Long" o "Short" se c'e' segnale)
```

L'istanza `TradableGroup()` è forse la più rilevante: essa raccoglie le informazioni utili per valutare lo stato della combinazione lineare individuata dai coefficienti beta, i quali definiscono l'attrattore. Per gli elementi di un sistema (coppia per pairs-trading) ritenuto cointegrato, viene appunto creata una relativa istanza che è successivamente raccolta in un dizionario denominato `TradableGroupDictionary`, tramite cui si tiene memoria di tutti i sistemi cointegrati scovati all'interno di ogni specifico cluster. Tale dizionario viene poi successivamente assegnato all'attributo `SubGroups` di `Cluster()`, dei quali si è precedentemente parlato. Infine, ogni istanza `Cluster()` è raccolta in un dizionario `ClusterDictionary`.

3.2.4 Fase 4 : Score/Ranking dei sottogruppi

Nella quarta fase, partendo dal dizionario `ClusterDictionary`, costruiamo un dizionario che contenga semplicemente i sottogruppi precedentemente identificati su cui si è in attesa di segnale, liberandosi dell'appartenenza al cluster. Perciò tale dizionario conterrà, come value, oggetti del tipo `TradableGroup()`, i quali vengono ordinati in modo che a value più a destra nel dizionario corrisponda uno `Score` più basso.

```
# metodo appartenente alla classe TradableGroup() ;
# Agisce modificando l'attributo Score dell'istanza
def ZeroCrossing(self, limit):
    # limit e' il livello, rispetto a 0, dopo il quale si innesca il
    # segnale di trading nella combinazione lineare (2*sigma)
    beta1 = self.Mu
    x = self.Res - beta1 * np.ones(len(self.Res))  # centriamo i residui
    # attorno alla media 0
    nzeros = 0  # nzeros==1 significa che la combinazione ritorna a 0
    # una sola volta
    posSignals = np.where(np.abs(x) > limit)  # identifichiamo le
    # posizioni dei segnali
```

```
    posSignals = posSignals[0]  # e' una tupla di un elemento
    k = 0
    len_x = len(x)
    len_posSignals = len(posSignals)
    timesteps = 0
    if len_posSignals > 1:  # altrimenti c'e' un numero insufficiente di
        # segnali
        while k < len_posSignals - 1:

            if (posSignals[k + 1] - posSignals[k]) == 1:
                if (x[posSignals[k]] > 0 and x[posSignals[k + 1]] < 0)\
                        or (x[posSignals[k]] < 0 and
                            x[posSignals[k + 1]] > 0):
                    nzeros += 1
                    timesteps += 1

            else:
                for j in range(posSignals[k] + 1, posSignals[k + 1]):

                    if (x[posSignals[k]] > 0 and x[j] < 0) or \
                            (x[posSignals[k]] < 0 and x[j] > 0):
                        nzeros += 1
                        timesteps += j - posSignals[k]
                        break
                    if (j == posSignals[k + 1] - 1):
                        timesteps += posSignals[k + 1] - posSignals[k]
                        if (x[posSignals[k]] > 0 and
                            x[posSignals[k + 1]] < 0) or \
                                (x[posSignals[k]] < 0 and
                                 x[posSignals[k + 1]] > 0):
                            nzeros += 1
            k += 1
        if posSignals[len_posSignals - 1] != (len_x - 1):
            for j in range(posSignals[len_posSignals - 1] + 1, len_x):
                if (x[posSignals[len_posSignals - 1]] > 0 and x[j] < 0) \
                        or (x[posSignals[len_posSignals - 1]] < 0 and
                            x[j] > 0):
                    nzeros += 1
                    timesteps += j - posSignals[k]
                    break
        if nzeros > 0:
            AvgTime = timesteps / nzeros  # stima tempo medio per
            # mean-reversion
            self.Score = 1 / AvgTime
        else:
            self.Score = 0
    else:
        self.Score = 0  # se i segnali sono pochi, mettiamo il
        # sottogruppo in fondo alla classifica
    return
```

Tramite `def ZeroCrossing(self, limit)`, metodo accessorio di `class TradableGroup(object)`, assegniamo un punteggio ad ogni sottogruppo. Il primo passo per determinare lo score è quello di valutare quante volte, in seguito al manifestarsi di un segnale sulla combinazione lineare, si

ha che quest'ultima ritorna alla media 0. Contemporaneamente conteggiamo quanti periodi sono necessari affinchè ciò avvenga. Si computa a questo punto il tempo medio di ritorno `AvgTime`: per cui, con Score=1/AvgTime, si favoriscono i titoli che manifestano una maggiore velocità di *mean-reversion*.
Si è tuttavia consapevoli del fatto che, nel nostro contesto, in cui la finestra temporale su cui si svolgono le analisi di cointegrazione è piuttosto ristretta (60 periodi), difficilmente si hanno un numero di segnali sufficientemente significativi da determinare uno score che funga veramente da fattore di discernimento affidabile.
Nonostante ciò, l'attesa sarebbe che ad uno score più alto corrisponda con maggior probabilità un processo (la combinazione lineare generata) stazionario, data la continua tendenza ad un rapido ritorno a 0.
Un'altra considerazione sull'utilità di `AvgTime` può essere trovata nella sezione conclusiva.

3.2.5 Fase 5 : Segnali e gestione posizioni

```
class PositionManagement(object):

    def __init__(self):
        self.ReadySubGroups = {}  # dizionario di sottogruppi ammissibili
        # ma su cui si e' in attesa del segnale
        self.OpenPositions = {}  # dizionario di sottogruppi su cui si
        # sono aperte delle posizioni che attendono di essere chiuse
        # al momento opportuno
```

Un oggetto costruito a partire da `class PositionManagement(object)` è dotato di due attributi:

- `ReadySubGroups` : definisce un dizionario in cui si raccolgono i soli sottogruppi cointegrati (sotto forma di istanze `TradableGroup()`) con `Score` strettamente positivo; su di essi non è ancora stata aperta una posizione, ma vengono monitorati costantemente alla ricerca di un segnale di entrata prima di essere inseriti nel portafoglio.

- `OpenPositions` : definisce un dizionario in cui si raccolgono i soli sottogruppi che sono attualmente in portafoglio (sotto forma di istanze `TradableGroup()`); su di essi si è in attesa del manifestarsi di un eventuale segnale di uscita, in modo da poterli liquidare.

`OpenPositions` è l'unico dizionario che viene controllato e aggiornato quotidianamente. Infatti, per non rischiare che il backtest impieghi un tempo eccessivamente lungo, quasi tutte le operazioni delle fasi precedenti sono eseguite ogni 30 giorni, mantenendo la rolling window a 60 giorni. Ci si aspetta che la cointegrazione continui ad essere valida in questo periodo extra; poi, passati i 30 giorni, si ricostruiscono tutte le relazioni di cointegrazione ed i sottogruppi che costituiscono il dizionario `ReadySubGroups`.
Perciò, in quest'ultima fase si verifica preliminarmente la presenza di segnali di uscita, predisponendo la liquidazione dei titoli interessati nel caso in cui tale requisito sia soddisfatto. Dopodichè, se il portafoglio non è ancora pieno, si scorre il dizionario `ReadySubGroups` alla ricerca di sottogruppi che manifestino un segnale di entrata: non appena se ne sia trovato uno, si aggiungono al portafoglio i rispettivi titoli, in accordo con il peso individuato dai coefficienti dell'attrattore β.
Per scelta, ogni nuovo sottogruppo aggiunto alle posizioni aperte non deve operare su titoli già presenti in portafoglio.
Ad ogni sottogruppo è dedicato un eguale quantitativo di denaro: inizialmente l'idea era di costruire un portafoglio con 10 sottogruppi; in seguito, vedendo che, come verrà successivamente mostrato, non ci sono così tanti sottogruppi in un setting a frequenza giornaliera, si è optato per 3.

```
# metodi appartenenti alla classe TradableGroup()

# confronta prezzo attuale per determinare un eventuale segnale
# di entry : restituisce un boolean
def EntrySignal(self, data):
    signal = False
    Res = self.Res - self.Mu  # viene tolta la media: i residui hanno
    # ora media 0
    totsum = 0
    self.Direction = "Short"  # direzione della posizione da assumere
    # sulla combinazione di titoli
    for k in range(0, len(self.Coeff)):
        totsum += self.Coeff[k] * \
                  data[self.SymbolTradableElements[k]].Price
    currentPrice = totsum - self.Mu  # valore attuale della
    # combinazione lineare
    if (np.abs(currentPrice) > 2 * self.Sigma) and (
            np.abs(currentPrice) < 2.25 * self.Sigma):  # segnale
        # sulla combinazione
        signal = True
        if currentPrice < 0:  # segnale long sulla combinazione
            self.Direction = "Long"
    direction = self.Direction

    return signal, direction


# confronta prezzo attuale per determinare un eventuale segnale di
# exit : restituisce un boolean
def ExitSignal(self, data):
    takeProfit = 0.5  # livello rispetto alla media 0 per cui si esce
    # dalla posizione aperta (e' espresso come coefficiente di Sigma)
    signal = False
    Res = self.Res - self.Mu  # viene tolta la media: i residui hanno
    # ora media 0
    totsum = 0
    for k in range(0, len(self.Coeff)):
        totsum += self.Coeff[k] * \
                  data[self.SymbolTradableElements[k]].Price
    currentPrice = totsum - self.Mu  # valore attuale della
    # combinazione lineare

    if np.abs(currentPrice) > 2.5 * self.Sigma:  # segnale di exit
        # sulla combinazione
        signal = True

    if self.Direction == "Long":  # la posizione (ancora aperta) e'
        # long sulla combinazione di titoli
        if currentPrice > (-1) * takeProfit * self.Sigma:
            signal = True

    if self.Direction == "Short":  # la posizione (ancora aperta) e'
        # short sulla combinazione di titoli
        if currentPrice < takeProfit * self.Sigma:
```

```
55                signal = True

    return signal
```

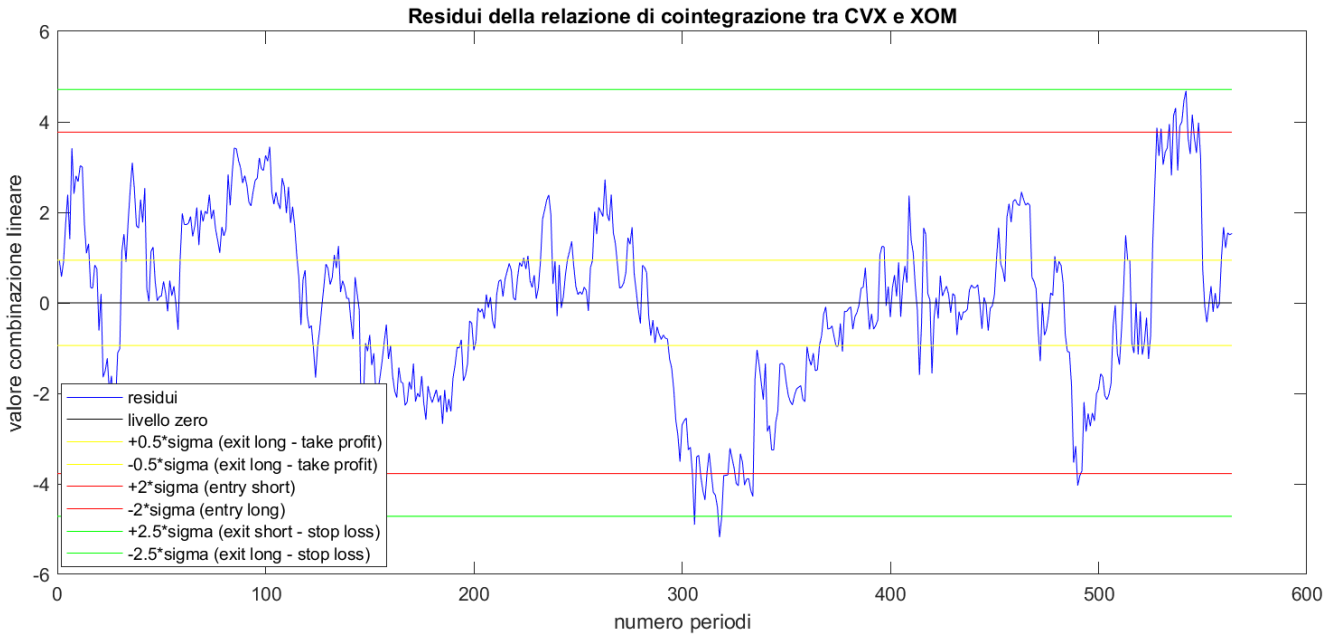


Figura 3.1 – Esempio di *relazione di cointegrazione* tra *Chevron Corp (CVX)* e *Exxon Mobil Corp (XOM)* relativo alla figura 2.1

Il funzionamento dei segnali di entry/exit è sufficientemente chiaro dal codice e dalla figura esemplificativa 3.1; l'unica precisazione riguarda l'entrata di una posizione: sia essa long o short, l'entrata avviene al superamento della soglia, in valore assoluto, di $2 \cdot std(\texttt{Res})$ e non oltre $2.25 \cdot std(\texttt{Res})$ (dove `Res` è l'attributo di `TradableGroup()` indicante la serie della combinazione lineare generata).

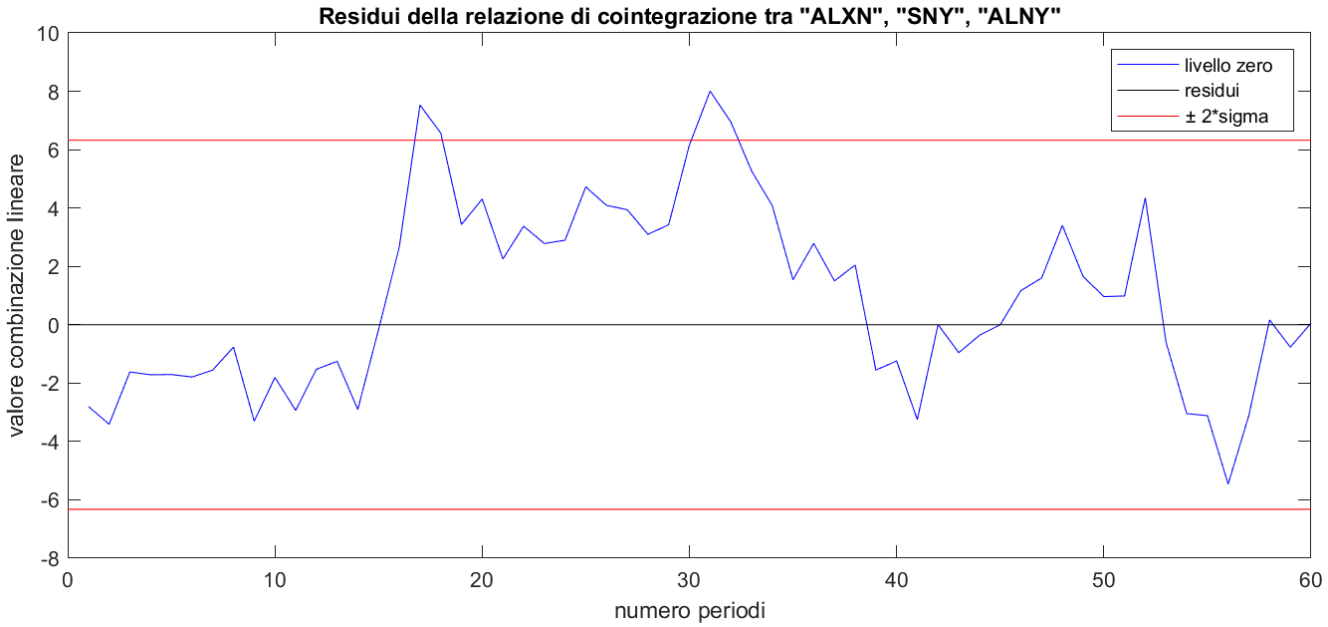


Figura 3.2 – Esempio di *relazione di cointegrazione* tra *'ALXN', 'SNY', 'ALNY'*; combinazione lineare generata automaticamente dall'algoritmo di backtesting nei 60 giorni di contrattazione precedenti al 05/10/2016.

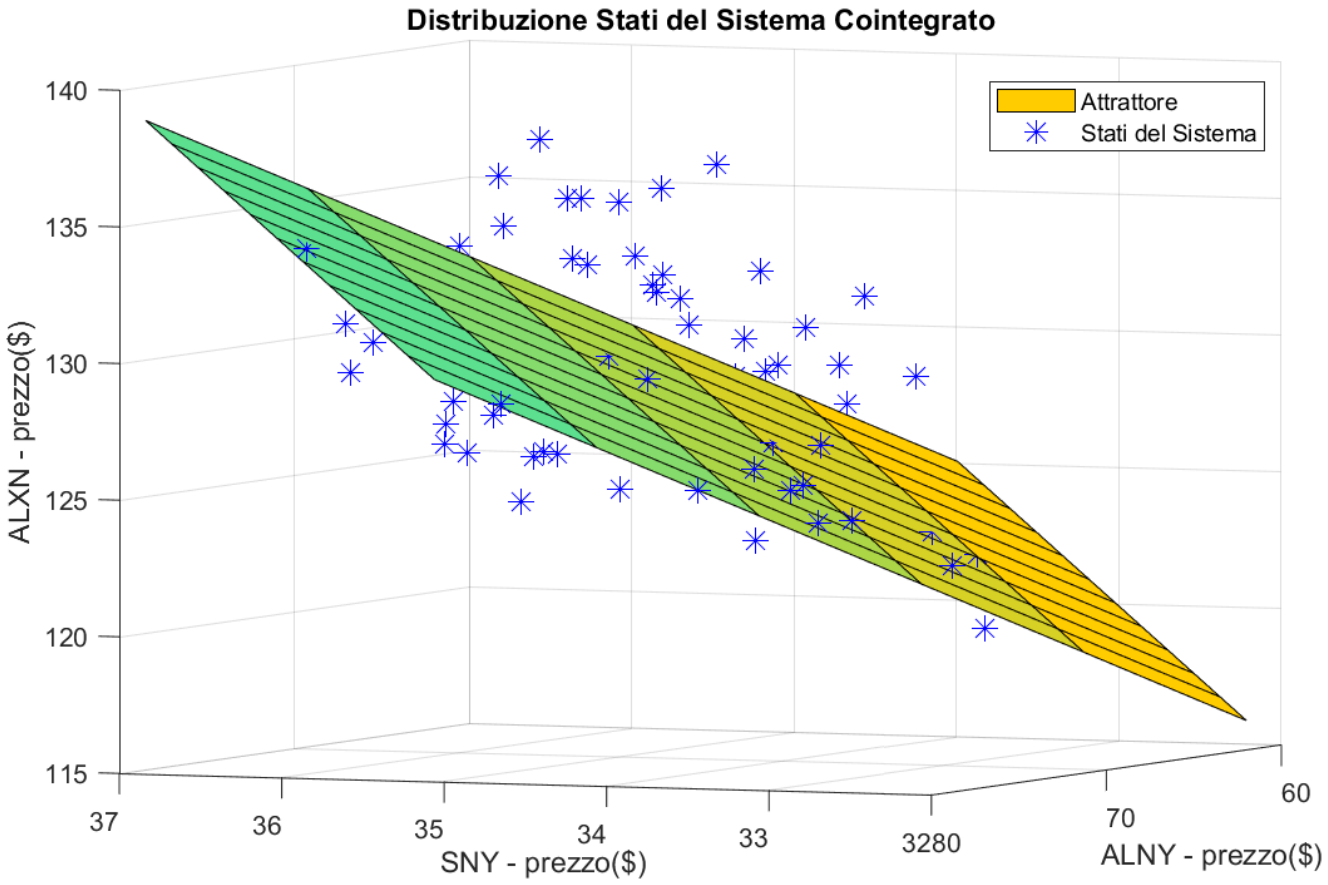


Figura 3.3 – Esempio di *Distribuzione Stati del Sistema Cointegrato* costituito da *'ALXN', 'SNY', 'ALNY'*; relativo alla figura 3.2; l'attrattore è costituito da un piano, con vettore di coefficienti $\beta = [1.0000 \quad -2.3404 \quad -0.6645]$.

Tuttavia, la figura 3.1 ha un numero di periodi certamente superiore al nostro contesto. L'illustrazione 3.2 rappresenta, invece, una serie generata dall'algoritmo di backtesting per statistical arbitrage. In 3.3 è mostrato anche l'attrattore che ci si attende rappresenti un equilibrio per il sistema cointegrato rilevato in 3.2.

3.3 Esecuzione backtest

Il backtest verrà eseguito separatamente per il modello pairs-trading e statistical arbitrage, con denaro di partenza impostato a $ 100000. Si segnalano, di seguito, alcune problematiche rilevate. Per le modalità di funzionamento della piattaforma, se si imposta frequenza giornaliera gli ordini vengono immessi all'apertura dei mercati il giorno successivo. Dunque, un segnale verificatosi alla chiusura di un giorno si trasforma in ordine di mercato non appena iniziano le contrattazioni il giorno dopo: ciò potrebbe implicare delle variazioni rispetto al prezzo istantaneamente rilevato al momento del segnale.
A causa di una qualche anomalia che non si è riusciti ad individuare, ci sono alcuni rari giorni in cui la matrice X non viene aggiornata e rimane dichiarata a zero: ciò fa scattare errori che bloccano il funzionamento del backtest. Si è ipotizzato che un'eventuale *delisting* o scorporo di qualche società possa impedire di "completare" la matrice di prezzi X. Per aggirare tale problema, si preferisce selezionare un intervallo di backtest (che sia comunque piuttosto recente) in cui tale questione non si manifesti minimamente: precisamente `self.SetStartDate(2015, 3, 1)` e `SetEndDate(2018, 8, 1)`.
Inoltre, per quanto riguarda il modello di statistical arbitrage, si è notato che, mantenendo lo stesso setting di pairs-trading, viene effettuato un numero veramente esiguo di operazioni. Ciò

è comunque legato al fatto che risulta più difficile stabilire cointegrazione efficace per più di due titoli, specie in un setting a frequenza giornaliera. Per mostrare ugualmente che l'algoritmo relativo funziona, si "allentano" alcuni parametri: la rolling window viene posta a 30 giorni; anche sottogruppi con `Score` $= 0$ vengono accettati; l'intervallo per i segnali di entrata diventa tra $2 \cdot std(\texttt{Res})$ e $2.5 \cdot std(\texttt{Res})$ mentre l'uscita è impostata a $3 \cdot std(\texttt{Res})$. Si è consapevoli del fatto che queste modifiche possano peggiorare l'affidabilità del processo, ma per completezza si vuole mostrare anche il backtest relativo allo statistical arbitrage.
Infine, si intende sottolineare che i risultati del backtest non siano, in generale, replicabili e che tali risultati possano differire anche in maniera sostanziale: l'inizializzazione probabilistica dei centroidi effettuata da *k-means++*, infatti, impedisce che in differenti istanze di backtest si effettuino esattamente le stesse operazioni, allo stesso momento. In ogni caso, la logica di esecuzione è la stessa, in quanto l'algoritmo sottostante non varia.

Si tiene a precisare che, come si può notare in 3.4, nel primo periodo dell'intervallo di backtest non vengano eseguite operazioni: ciò è dovuto al fatto che la piattaforma necessita di appunto 60 giorni di contrattazione per raccogliere i dati iniziali per la rolling window.

Nel complesso, il numero di operazioni eseguite dal modello di pairs-trading non è molto elevato, come invece ci si attendeva, ed il portafoglio non è sempre pieno. Certamente, incrementare la frequenza, in modo da operare non solo sulla base di prezzi di chiusura ma piuttosto ogni ora o minuto, permetterebbe di generare un turnover di operazioni molto maggiore. Tuttavia, il tempo necessario a condurre un simile backtest inizierebbe a diventare proibitivo per gli scopi di questa trattazione, specie se, per assicurarsi una maggiore cointegrazione, si optasse per una rolling window più lunga.

3.4 Considerazioni conclusive e possibili miglioramenti

Come si era ammesso nella parte introduttiva del nostro documento, i modelli di pairs-trading sono stati ampiamente sfruttati negli ultimi decenni, portando in generale ad una enorme diminuzione dei rendimenti. Ciò è confermato dal nostro modello, il quale fatica a fare profitti, specie ricordandosi che, per semplicità, sono stati ignorati i costi di transazione, i quali possono gravare molto su strategie attive come quelle analizzate.
Solo modelli più elaborati, costruiti riservando particolare attenzione a dettagli minimi che in molti casi non sono nemmeno stati presi in considerazione nelle nostre considerazioni, possono ancora sopravvivere in questo ambiente.
Nonostante ciò, si ritiene sia stato raggiunto lo scopo di realizzare un algoritmo basato sui modelli descritti ai capitoli precedenti che segua quanto più fedelmente le regole imposte ed enunciate in quest'ultimo capitolo.
Sicuramente sono molti gli aspetti che avrebbero potuto influenzare in maniera più o meno significativa il backtest, oltre a quelli già accennati in precedenza. Innanzitutto, in molti casi le relazioni individuate tra titoli ritenuti cointegrati non lo erano veramente: si trattava spesso di relazioni spurie o comunque piuttosto deboli sul lungo periodo che fattori quali una rolling window piuttosto breve e movimenti di prezzo simili in uno stesso periodo non contribuivano certo a smascherare. Sarebbe dunque ragionevole chiedersi se si possa in qualche modo regolarizzare il processo di detection delle relazioni di cointegrazione in modo da limitare i falsi positivi. Similmente, studi approfonditi potrebbero esser condotti nel tentativo di verificare se particolari eventi ricorrenti, quali sbalzi di volatilità nei mercati possano determinare rotture strutturali nelle sottostanti relazioni di cointegrazione fra titoli. Eventualmente, modelli di previsione della variazione di volatilità a breve termine potrebbero venir implementati come miglioramenti.
Per quanto concerne le modalità di utilizzo dell'algoritmo di clustering *k-means*, si è rilevato, in

(a) Pairs-Trading

(b) Statistical Arbitrage

Figura 3.4 – Backtest dei due modelli con il setting descritto nel capitolo; non sono inclusi costi di transazione.

fase di costruzione del modello di backtest, che si sarebbero potuti escogitare modi alternativi e forse più efficaci a raggiungere gli scopi prefissati: ad esempio realizzare un algoritmo di cluste-

Overview			
Overall Statistics			
Total Trades	160	Average Win	0.40%
Average Loss	-0.32%	Compounding Annual Return	1.489%
Drawdown	1.400%	Expectancy	0.153
Net Profit	3.896%	Sharpe Ratio	0.778
PSR	34.463%	Loss Rate	49%
Win Rate	51%	Profit-Loss Ratio	1.25
Alpha	0.012	Beta	0.005
Annual Standard Deviation	0.016	Annual Variance	0
Information Ratio	-0.788	Tracking Error	0.118
Treynor Ratio	2.675	Total Fees	$0.00

Figura 3.5 – Statistiche descrittive del backtest presentate da QuantConnect relativamente al modello Pairs-Trading.

ring simile a *k-means* ma con il vincolo di suddividere direttamente l'intero spazio di elementi in cluster (di cui non importa il numero) di dimensione uguale e prestabilita ($\approx$ 2-3 elementi per pairs-trading). Ciò avrebbe forse parzialmente migliorato la probabilità di individuare vera cointegrazione tra gli elementi di uno stesso cluster.
Variazioni plausibili sarebbero state attuabili anche per l'individuazione dei segnali di uscita per stop loss: adottando `AvgTime` come tempo medio di ritorno allo zero della combinazione lineare, si è in grado, nel caso di un numero significativamente elevato di segnali, di studiare la distribuzione dei tempi di *mean-reversion* dei diversi segnali di uno stesso titolo, allo scopo di identificarne un intervallo di confidenza. Esso potrebbe fungere da riferimento per un'eventuale segnale di uscita dipendente dal tempo in cui il titolo si trova in portafoglio.
Fondamentale, come si è gia detto, sarebbe sviluppare l'intero modello attorno ai costi di transazione, di cui si può cogliere la rilevanza nell'analisi sviluppata nell'ultima sezione del capitolo precedente. Si ritiene, tuttavia, che per far ciò si necessiti di controllare l'intero processo in maniera estremamente più efficace di come si sia tentato di fare nella nostra trattazione.

Glossario

1. ***mean-reversion*** : comportamento del titolo che tende, in un tempo non eccessivo, a ricondursi ad una sorta di prezzo di equilibrio di medio-lungo termine.

2. ***stazionarietà*** : una serie storica $(y_t)_{t=0,\dots,T}$ si dice stazionaria se: $\mathbb{E}[y_t] = \mu$, $\mathbb{V}ar(y_t) = \sigma^2$, $\mathbb{C}ov(y_t, y_{t+s}) = \mathbb{C}ov(y_t, y_{t-s}) = \gamma_s \quad \forall t$

3. ***pairs-trading*** : arbitraggio statistico in cui vengono associati titoli in coppia.

4. ***long*** : posizione di acquisto assunta su di un asset.

5. ***short*** : posizione di vendita assunta su di un asset. Essa precede l'acquisto, che è differito ad un secondo momento, ma indispensabile a coprire la posizione presa in precedenza.

6. ***centroide*** : dato uno spazio d-dimensionale, il centroide è inteso come la media aritmetica, computata per ogni coordinata, di tutti gli elementi in tale spazio.

7. ***ticker*** : abbreviazione utilizzata per identificare un titolo quotato. Non è detto, tuttavia, che tale rappresentazione sia univoca.

8. ***rolling window*** : finestra temporale mobile in cui i dati contenuti sono utilizzati per "training" del modello (in-sample period), la cui capacità predittiva sarà valutata in un periodo successivo (out-of-sample period).

9. ***attrattore***: è un insieme verso il quale evolve un sistema dinamico dopo un tempo sufficientemente lungo.

Bibliografia

[1] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

[2] David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, 2006.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[4] Chandan K. Reddy Charu C. Aggarwal. *Data Clustering.* CRC Press, Taylor & Francis Group, LLC, 2014.

[5] Robert F Engle and Clive WJ Granger. Co-integration and error correction: representation, estimation, and testing. *Econometrica: journal of the Econometric Society*, pages 251–276, 1987.

[6] R Carter Hill, William E Griffiths, and Guay C Lim. *Principi di econometria.* Zanichelli, 2013.

[7] Søren Johansen. Cointegration: an overview, 2004.

[8] I.T. Jolliffe. *Principal Component Analysis.* Springer, second edition, 2010.

[9] Riccardo Lucchetti. Appunti di analisi delle serie storiche, 2015.

[10] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval.* Cambridge university press, 2008.

[11] Shokri Z Selim and Mohamed A Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on pattern analysis and machine intelligence*, (1):81–87, 1984.