

UNIVERSITÀ DEGLI STUDI DI TORINO

Facoltà di Economia

Corso di Laurea in Finanza Aziendale e dei
Mercati Finanziari



TESI DI LAUREA

ABM e Distribuzione Alternativa dei Rendimenti: Analisi e
Risultati

Relatore: Prof. Pietro Terna

Correlatore: Prof. Sergio Margarita

Candidato: Andrea Romeo

Anno Accademico 2010/2011

Indice

Introduzione.....	3
Capitolo 1	
Inquadramento teorico.....	6
Capitolo 2	
Modello di simulazione ABM.....	17
2.1 NetLogo.....	17
2.1.1 Interfaccia.....	22
2.2 Modello NetLogo di Mercato Finanziario.....	24
Capitolo 3	
Strumenti software di analisi dati.....	56
3.1 Maxima.....	56
3.2 Elaborazione dati Maxima.....	58
Capitolo 4	
Analisi dati.....	74
4.1 Test sull'apparato statistico utilizzato.....	74
4.2 Analisi dati reali.....	76
4.3 Analisi dati simulati.....	79
4.3.1 Dimensione.....	81
4.3.2 Volatilità del prezzo di offerta.....	83
4.3.3 Stazionarietà.....	90
Conclusioni.....	92
Appendice A.....	98
Appendice B.....	104

Introduzione

La complessità dei sistemi che generano i fenomeni che noi osserviamo spesso è tale che non ci consente di prevedere in maniera deterministica ciò che accadrà nelle diverse situazioni.

Tuttavia per non essere del tutto impotenti proviamo a stimare quantomeno la probabilità che ogni dato evento ha di manifestarsi.

Osservando il mondo reale però ci accorgiamo che a volte l'improbabile si realizza.

Quando questo avviene ci troviamo di fronte ad un bivio: da una parte possiamo affermare di essere stati molto fortunati o molto sfortunati, dall'altra possiamo fare un esame di coscienza e cercare di capire se non abbiamo commesso errori nel valutare ex-ante la probabilità dell'evento in questione.

I mercati finanziari hanno funzioni fondamentali per l'intero sistema economico: fra le altre consentono di trasferire risorse da soggetti in surplus a soggetti in deficit, permettono una distribuzione migliore del rischio tra soggetti e offrono un metodo per attribuire un valore ad un bene, il cosiddetto prezzo di mercato appunto.

Quando avvengono eventi improbabili nei mercati finanziari, se scegliamo la prima strada, cioè pensiamo che l'evento sia frutto del gioco del caso, stiamo implicitamente rinunciando a comprendere il funzionamento di un nodo cruciale della "rete" economica.

La crisi finanziaria iniziata nel 2007 ha avuto però se non altro il merito di farci riflettere: ci siamo chiesti se non fosse giunto il momento di intraprendere la seconda via.

Il lavoro che sarà qui proposto è stato concepito partendo proprio da questa riflessione, e vuole proporsi come un incipit e un piccolo incoraggiamento per tutti coloro che intendono intraprendere l'arduo e lungo secondo cammino.

Il lavoro qui proposto infatti si muove all'interno di un quadro teorico alternativo a quello utilizzato in larga parte in passato: se le variazioni di prezzo sono spesso state viste e modellizzate come realizzazioni di variabili casuali Normali o affini, il nostro inquadramento teorico ipotizza che le suddette variazioni possano essere viste come realizzazioni di variabili aleatorie con distribuzione stabile.

Il termine alternativo non vuole essere inteso come una rottura netta con il passato, bensì è da leggersi nell'ottica della volontà di offrire uno strumento in più, di offrire la possibilità di scegliere tra una gamma più ampia di possibilità di approccio ai mercati.

Questa idea sta alla base anche della scelta di utilizzare in combinazione con la teoria un modello di simulazione ad agenti.

Vi è infatti la ferma convinzione che le simulazioni effettuate attraverso modelli ABM siano uno strumento importante e che esse siano in grado di affiancare i tradizionali modelli matematici per quello che riguarda la comprensione di fenomeni economici.

Il lavoro pertanto è stato svolto con l'intenzione di osservare i mercati finanziari da un altro punto di vista, nella speranza di ottenere elementi utili e utilizzabili per comprenderne meglio il funzionamento.

Nel primo capitolo sarà presentata dunque la teoria che sta alla base del lavoro.

Si parte in particolare dagli studi proposti fra gli altri da Mandelbrot, evidenziandone ed analizzandone gli elementi costituenti e gli elementi distintivi.

Essendo la distribuzione di probabilità Normale un caso particolare di distribuzione stabile, si cerca di offrire uno spunto alla creazione di modelli più generali.

È messo in luce in particolare il ruolo svolto dal parametro α delle distribuzioni stabili, che come mostreremo influisce sulla forma della distribuzione ed in particolar modo sullo spessore delle code, cioè sulla probabilità che si assegna agli eventi rari.

Nel secondo capitolo invece si affronta la costruzione del modello di simulazione, mostrando come le ipotesi di partenza si traducono, attraverso il codice di programmazione, nei comportamenti degli agenti.

Il modello è stato realizzato partendo da quanto proposto da Terna nel modello "baseCDAModel" [Terna, 2009].

Il linguaggio di programmazione utilizzato è NetLogo.

Il terzo capitolo riguarderà l'apparato statistico.

In questo capitolo si cercherà di tradurre in pratica quanto proposto nel primo capitolo, affiancando alla spiegazione del codice utilizzato per il calcolo il riferimento costante alla teoria.

Il software usato è Maxima.

I risultati ottenuti attraverso l'analisi statistica proposta nel capitolo 3 dei dati generati con il

modello descritto nel capitolo 2, confluiranno nel quarto e ultimo capitolo, così come le possibili chiavi di lettura.

Qui mostreremo in particolare come i dati possano ben adattarsi ad essere modellizzati tramite le distribuzioni stabili proposte nel capitolo 1.

Capitolo 1

Inquadramento teorico

Occupandosi di mercati finanziari, occorre in qualche modo confrontarsi con i comportamenti più svariati: i prezzi si comportano in un modo, i rendimenti in un altro, così come la volatilità.

Un fenomeno analizzabile in un mercato è, ad esempio, l'andamento nel tempo dei rendimenti.

Per rendimenti intendiamo la variazione di prezzo tra due istanti successivi.

Vi sono molti modi per formulare in termini matematici il rendimento; uno di quelli maggiormente usati fa riferimento al regime di capitalizzazione continua:

$P_{t+1} + D_{t+1} = P_t e^r$, se ipotizziamo che i dividendi siano 0 tra t e $t+1$:

$$\frac{P_{t+1}}{P_t} = e^r \quad \text{e quindi} \quad r = \ln \frac{P_{t+1}}{P_t} .$$

Questo implica anche che il rendimento su due periodi è:

$P_{t+2} = P_{t+1} e^{r_{t+2}} = P_t e^{r_{t+1}} e^{r_{t+2}} = P_t e^{r_{t+2} + r_{t+1}} = P_t e^{r_{t+2,2}}$, cioè il rendimento su più periodi è pari alla somma dei rendimenti uniperiodali.

L'analisi sui rendimenti incomincia spesso con l'analisi di una serie storica di dati, ordinati nel tempo.

Le frequenze con cui si manifestano le osservazioni all'interno del campione si ipotizza che siano indicatrici della probabilità che avremo di osservare una data osservazione, in questo caso un dato rendimento, all'interno del mercato da cui è estratto il campione.

Assumiamo che la variabile che stiamo osservando sia una variabile casuale: stiamo pertanto facendo inferenza sulla sua funzione di probabilità.

Ipotizzare che la frequenza con cui incontriamo un dato rendimento all'interno del campione possa essere assunta come la probabilità di ottenere quel dato rendimento all'interno del mercato, significa

dunque ipotizzare che vi sia stazionarietà, cioè che le caratteristiche della funzione di probabilità della variabile casuale non cambiano nel tempo.

A questo punto molte implicazioni operative necessitano che venga definita una funzione di probabilità.

La funzione di probabilità più usata nel campo dei mercati finanziari per descrivere il comportamento dei rendimenti è stata la funzione di probabilità di una variabile casuale normale.

Mandelbrot nel paper *The Variation of Certain Speculative Prices* [Mandelbrot, 1963], tuttavia, presenta un nuovo modello che descriva il comportamento dei prezzi in mercati speculativi.

Questo modello di comportamento, che si può chiamare “Paretiano stabile”, viene proposto come un'alternativa alla tradizionale distribuzione Gaussiana nel modellizzare le variazioni di prezzo tra due istanti successivi.

La necessità di trovare un modello alternativo nasce dal fatto che i dati reali non si adattano sufficientemente al modello tradizionale: in particolare alcune analisi [Tinter, 1940] mostrano come le distribuzioni empiriche di rendimenti presentino campane più alte e strette rispetto alla distribuzione Normale, nonché code più spesse e lunghe [Nolan, 2003b].

Il vantaggio di lavorare con un modello che ipotizza distribuzioni Normali è che la distribuzione Normale è stabile, cioè la somma di due variabili casuali Normali $N_1(\mu_1; \sigma_1^2)$ e $N_2(\mu_2; \sigma_2^2)$, è a sua volta una variabile casuale Normale:

$$aN_1 + bN_2 \sim N_3(a\mu_1 + b\mu_2; a^2\sigma_1^2 + b^2\sigma_2^2)$$

Nel caso di due variabili casuali Normali, G_1 e G_2 , con media zero e varianza σ_1^2 e σ_2^2 , la combinazione lineare delle due, G , si distribuisce come una variabile casuale Normale nel seguente modo:

$$G \sim N(0, \sigma_1^2 + \sigma_2^2)$$

Per ottenere una variabile casuale Normale standard, cioè $N(0,1)$, dalla combinazione lineare di altre due variabili casuali Normali standard, occorre che si verifichino le seguenti condizioni:

$$s_1U_1 + s_2U_2 = sU, \text{ dove } U \sim N(0,1),$$

e, poiché $G_1 = s_1U \sim N(0, s_1^2)$, $G_2 = s_2U \sim N(0, s_2^2)$ e $G = sU \sim N(0, s^2)$,

$$s_1^2 + s_2^2 = s^2 \quad .$$

La prima è la condizione di “stabilità” vera e propria mentre la seconda è una relazione ausiliaria che permette di esprimere il fattore di “scala” s in funzione di s_1 e s_2 .

Nel caso della distribuzione Gaussiana i fattori di “scala” s_1 , s_2 e s sono chiaramente collegati alla varianza, cioè al momento di ordine 2, che è finito.

Tuttavia nel caso di altre distribuzioni, i momenti delle quali sono infiniti, i fattori di “scala” non possono più essere espressi come funzione dei momenti.

Lévy [Lévy, 1925][Lévy e Borel, 1954] ha fornito l'espressione della funzione caratteristica delle distribuzioni L-stabili, cioè di quelle distribuzioni che sono soluzione della condizione di stabilità sopra citata:

$$\log \varphi(t) = i \delta t - \gamma |t|^\alpha \left[1 + i \beta \left(\frac{t}{|t|} \right) \tan \left(\frac{\alpha \pi}{2} \right) \right]$$

Sia la distribuzione Normale che quella di Cauchy si possono ottenere partendo dall'espressione precedente, imponendo rispettivamente $\alpha=2$ e $\alpha=1$, $\beta=0$.

Infatti imponendo $\alpha=2$, $\delta=\mu$ e $\sigma^2=2\gamma$ otteniamo:

$$\varphi(t) = \exp(i \delta t - \gamma |t|^\alpha \left[1 + i \beta \left(\frac{t}{|t|} \right) \tan \left(\frac{\alpha \pi}{2} \right) \right]) = \exp(i \delta t - \gamma t^2) = \exp(i \mu t - \frac{\sigma^2 t^2}{2}) \quad , \text{ cioè la funzione}$$

caratteristica della distribuzione Normale di media μ e varianza σ^2 .

Nell'espressione precedente i parametri possono essere interpretati nel modo seguente: δ è il parametro di “posizionamento”, γ è il parametro di “scala”, nel senso che determina la grandezza delle probabilità complessive, β è l'indice di asimmetria, mentre α è il parametro che determina lo spessore delle code [Nolan, 2003].

I parametri sono compresi nei range:

$$\alpha \in]0,2] \quad , \quad \beta \in [-1,1] \quad , \quad \gamma \in [0,\infty] \quad \text{e} \quad \delta \in \mathbb{R} \quad .$$

Nel caso $1 < \alpha \leq 2$, il valore atteso di U è finito, $E(U) < \infty$, e possiamo esprimere $\delta = E(U)$. Se inoltre $\beta = 0$, allora δ è anche la mediana della distribuzione.

α e β determinano il “tipo” di variabile aleatoria, che si può considerare standard nel caso in cui $\gamma=1$ e $\delta=0$.

Se U è una variabile casuale “stabile” standardizzata, sU sarà una variabile casuale “stabile” con gli

stessi valori α , β e δ di U , e con $\gamma = s^\alpha$.

Quindi la distribuzione della combinazione lineare di due variabili aleatorie “stabili” standardizzate è:

$$s_1 U_1 + s_2 U_2 = s U$$

con

$$s_1^\alpha + s_2^\alpha = s^\alpha.$$

La somma di N variabili aleatorie “stabili” i.i.d. è esprimibile come: $U_1 + U_2 + U_3 + \dots + U_N = S_N$.

La funzione caratteristica associata a S_n sarà:

$$\log \varphi(z) = i \delta N z - N \gamma |z|^\alpha \left[1 + i \beta \left(\frac{z}{|z|} \right) \tan \left(\frac{\alpha \pi}{2} \right) \right].$$

Nel caso particolare che si modellino gli incrementi di prezzo come una variabile casuale “stabile” con $\alpha=2$ (distribuzione Normale), se la deviazione standard degli incrementi giornalieri è $\sigma(1)$, la deviazione standard degli incrementi su T giorni è:

$$\sigma(T) = \left(\sum_{i=1}^T \sigma_i^2(1) \right)^{1/2} = (T \sigma^2(1))^{1/2} = T^{1/2} \sigma(1).$$

Un'altra misura della variabilità è basata sui quartili ed è fornita dalla differenza interquartile ($Q_3 - Q_1$).

Poiché tra Q_1 e Q_3 si trova il 50 per cento centrale della distribuzione, se la loro differenza è piccola, vuol dire che la variabilità è contenuta; se la differenza è ampia, la variabilità è elevata.

Nel caso più generale invece in cui si modellino gli incrementi di prezzo come una variabile casuale “stabile”, il valore atteso della differenza interquartile degli incrementi su T giorni può essere espresso come:

$$E[Q_3(T) - Q_1(T)] = T^{1/\alpha} E[Q_3(1) - Q_1(1)].$$

Lévy ha mostrato che le code delle distribuzioni “stabili” (eccetto la Normale), seguono una forma asintotica della Legge di Pareto.

Se prendiamo la distribuzione di Cauchy, utilizzando il teorema di De l'Hopital, possiamo calcolare il seguente limite:

$$\lim_{u \rightarrow \infty} u Pr(U < -u) = \lim_{u \rightarrow \infty} u Pr(U > u) = \lim_{u \rightarrow \infty} \frac{\left[\frac{1}{2} - \frac{1}{\pi} \arctan(u) \right]}{u^{-1}} = \lim_{u \rightarrow \infty} \left(\frac{\pi + \pi u^2}{u^2} \right)^{-1} = \frac{1}{\pi} .$$

Date due costanti $c_1 = \sigma^+$ e $c_2 = \sigma^-$, legate dalla relazione:

$$\beta = \frac{(c_1 - c_2)}{(c_1 + c_2)} , \text{ con } |\beta| \neq 1 ,$$

al tendere di u all'infinito le probabilità si comportano nella maniera seguente:

$$u^\alpha Pr(U > u) \rightarrow c_1 = \sigma^{+\alpha} , \quad u^\alpha Pr(U < u) \rightarrow c_2 = \sigma^{-\alpha} ,$$

dove σ^+ e σ^- sono i corrispettivi della deviazione standard nel caso di distribuzione Gaussiana e possono essere chiamati “deviazione standard positiva” e “deviazione standard negativa”.

Quando $|\beta|=1$, allora $c_1=0$ o $c_2=0$, e quindi rispettivamente la coda negativa o la coda positiva decrescono più velocemente di quanto farebbe la distribuzione di Pareto di parametro α .

La distribuzione Normale è utilizzata per modellizzare fenomeni in quanto il teorema centrale del limite afferma che se si ha una somma di variabili aleatorie U_n i.i.d. con media $E[U]$ e varianza $\sigma^2 = E[U_n - E[U]]^2$, entrambe finite, allora indipendentemente dalla forma distributiva di partenza, al tendere della dimensione campionaria a infinito la somma tende a distribuirsi come una variabile casuale Normale, cioè che:

$$\lim_{n \rightarrow \infty} N^{-1/2} \sigma^{-1} \sum_{n=1}^N [U_n - E[U]]$$

è una variabile casuale Normale standard.

Inoltre si può affermare che esistono due funzioni $A(N)$ e $B(N)$ tali che, al tendere di N a infinito, la somma

$$A(N) \sum_{n=1}^N U_n - B(N)$$

ha limite finito e che tale limite non è riducibile ad una costante non casuale.

Quando la varianza delle distribuzioni U_n non è finita, quest'ultima affermazione rimane valida, mentre ciò che afferma il teorema centrale del limite cessa di essere valido.

La funzione $A(N)$ può essere scelta come: $A(N) = N^{-1/\alpha}$.

Le condizioni di Pareto-Doeblin-Gnedenko [Gnedenko et altri, 1968], che introducono la notazione:

$$Pr(U > u) = Q'(u)u^{-\alpha} \quad e \quad Pr(U < -u) = Q''(u)u^{-\alpha}$$

richiedono che:

1. per u che tende a infinito, $Q'(u)/Q''(u)$ tenda al limite c_1/c_2
2. esista un valore di $\alpha > 0$ tale che, per ogni $k > 0$ e per u che tende a infinito, si abbia:

$$\frac{Q'(u) + Q''(u)}{Q'(ku) + Q''(ku)} \rightarrow 1$$

Le condizioni di Pareto-Doebelin-Gnedenko sono una generalizzazione della legge asintotica di Pareto: quest'ultima infatti è più stringente in quanto richiede che sia $Q'(u)$ che $Q''(u)$ abbiano limite al tendere di u a infinito.

A questo punto è possibile applicare il seguente teorema: se U_n sono variabili casuali i.i.d., possono

non esistere funzioni $A(N)$ e $B(N)$ tali per cui $A(N) \sum_{n=1}^N U_n - B(N)$ tenda a un limite vero e proprio. Tuttavia, se esistono, il limite sarà una delle soluzioni dell'equazione di "stabilità".

Più precisamente, il limite sarà una variabile aleatoria con distribuzione Normale se e solo se U_n sono variabili aleatorie con varianza finita; il limite sarà invece una variabile aleatoria con distribuzione "stabile" non Gaussiana se e solo se le condizioni di Pareto-Doebelin-Gnedenko sono

soddisfatte per un $\alpha \in (0,2)$. Quindi $\beta = \frac{(c_1 - c_2)}{(c_1 + c_2)}$ e $A(N)$ sarà ricavato dalla condizione

$$N Pr[U > u A^{-1}(N)] \rightarrow c_1 u^{-\alpha} .$$

Tuttavia i comportamenti asintotici spesso non sono rilevanti nelle applicazioni pratiche.

Sebbene eccetto in casi particolari (ad esempio distribuzione Normale e distribuzione di Cauchy) non si conosca una forma chiusa per l'espressione di densità delle distribuzioni "stabili", la teoria afferma che:

1. le funzioni di densità sono sempre unimodali, cioè raggiungono un solo valore massimo;
2. le funzioni di densità dipendono in maniera continua dai parametri;
3. se $\beta > 0$ e $\alpha \in (0,2)$, la coda positiva è la più spessa e la media è più grande della mediana e del valore più probabile.

Nel caso di distribuzioni "stabili" simmetriche $\beta = 0$, la rappresentazione di $Pr(U > u) = Pr(U < -u)$ in un grafico a doppia scala logaritmica ci mostra come il parametro α possa essere interpretato

come valore indicatore della “pendenza”, tenendo costanti gli altri parametri.

È possibile ipotizzare che le variazioni a un periodo dei prezzi

$$L(t,1) = \log Z(t+1) - \log Z(t)$$

siano variabili casuali con momenti della popolazioni infiniti oltre il momento di ordine uno.

Questo significa che:

$$\int_{-\infty}^{\infty} p(u)u \, du < \infty \quad \text{e} \quad \int_{-\infty}^{\infty} p(u)u^2 \, du = \infty \quad ,$$

cioè:

$$\lim_{u \rightarrow \infty} p(u)u^2 = 0 \quad \text{e} \quad \lim_{u \rightarrow \infty} p(u)u^3 = \infty \quad .$$

Un'altra ipotesi può essere che le variazioni dei prezzi da t a $t+T$

$$L(t,T) = \log Z(t+T) - \log Z(t)$$

dipendano solo da T , e quindi seguiranno lo stesso processo a meno del parametro T , che andrà a modificare la “scala” delle varie distribuzioni.

Un'ipotesi del genere ci riconduce al concetto di “stabilità” sopra enunciato.

Il modo più semplice per esprimere un comportamento di questo tipo delle code di una distribuzione è utilizzare una distribuzione che sia asintoticamente Paretiana.

È conveniente utilizzare una distribuzione Paretiana nella modellizzazione, poichè si sostituisce una distribuzione “stabile”, come quella Gaussiana, con un'altra “stabile”, mantenendo così inalterate le proprietà di scala.

Mandelbrot [Mandelbrot, 1963] afferma che per descrivere fenomeni come la variazione dei prezzi in un mercato, la distribuzione della ricchezza in una popolazione o la dimensione delle imprese è necessario utilizzare variabili aleatorie che presentano distribuzioni di probabilità con momenti superiori al primo infinito.

Possiamo ad esempio prendere in considerazione la distribuzione Paretiana, per quello che riguarda valori positivi, proponendola come alternativa nella modellizzazione di fenomeni in cui è importante usare distribuzioni invarianti a determinati tipi di trasformazioni.

In particolare la distribuzione Paretiana è invariante:

1. alla somma ponderata di variabili casuali Paretiane,

2. alla scelta del massimo tra variabili casuali Paretiane,
3. alla somma di variabili casuali Paretiane.

Supponiamo di avere una serie di variabili casuali paretiane U_n che si distribuiscono secondo:

$$Pr(U_n > u) \sim C_n u^{-\alpha} .$$

Se la variabile casuale U_w ha probabilità di distribuirsi come una variabile casuale U_n pari a p_n , con $1 < n < N$, U_w stessa sarà una variabile casuale asintoticamente Paretiana, con fattore di scala:

$$C_w = \sum_{n=1}^N p_n C_n .$$

Se la variabile casuale U_M è la variabile che a posteriori presenta il valore più grande delle N U_n variabili casuali Paretiane, essa stessa sarà una variabile casuale asintoticamente Paretiana, con fattore di scala:

$$C_M = \sum_{n=1}^N C_n .$$

Se la variabile casuale U_w è una somma delle N U_n variabili casuali Paretiane essa stessa sarà una variabile casuale asintoticamente Paretiana, con fattore di scala:

$$C_M = \sum_{n=1}^N C_n .$$

Questo significa che la somma di N variabili casuali Paretiane si comporta asintoticamente come la più grande di esse.

Il fatto di utilizzare distribuzioni con momenti superiori al primo infinito può essere giustificato in quei casi in cui il momento secondo campionario osservato, con l'ampliarsi del campione, non converga o si stabilizzi rapidamente attorno al valore corrispondente all'intero campione.

In questi casi non si ha una stima affidabile del momento secondo dell'intera popolazione.

Le cause possono essere molteplici, in particolare se ne possono citare due: o il campione è troppo piccolo per poter inferire in modo attendibile sui veri valori della popolazione, o semplicemente si può assumere che il momento secondo della popolazione possa assumere anche il valore infinito.

Se si considera accettabile la seconda spiegazione, un possibile lato positivo è che il modello non dipenderebbe in maniera eccessiva da un valore che potrebbe essere difficile da stimare.

Nel caso si assuma che il fenomeno da modellizzare segua una distribuzione con momento secondo infinito, questo vuol dire che al tendere di u ad infinito, la distribuzione delle code, cioè ad esempio $1 - F(u)$, decrescerà più lentamente di $1/u$ ma più velocemente di $1/u^2$.

In questo caso il comportamento di $F(u)$ nelle code è importante, e può essere approssimato da

$$Cu^{-\alpha}, \text{ con } 1 < \alpha < 2.$$

Nel caso invece si assuma che i momenti della distribuzione siano finiti, diciamo almeno fino al quarto, significa che il comportamento di $F(u)$ nelle code non molto è importante, e la distribuzione può essere approssimata ad esempio da una distribuzione Normale o log-Normale.

Un lato negativo nell'assumere che i momenti di ordine superiore al primo siano infiniti invece riguarda l'impossibilità di utilizzo di alcuni strumenti statistici e di inferenza, come quelli che si basano ad esempio sulla deviazione standard.

Il quadro teorico proposto è applicato da Mandelbrot alle variazioni di prezzi, ed in particolare l'analisi svolta nel paper *The Variation of Certain Speculative Prices* verte sui prezzi del cotone nel periodo che va dal 1880 al 1940.

I risultati mostrano come le variazioni di prezzi giornaliere sembrano seguire lo stesso processo delle variazioni mensili, a meno di un fattore di "scala".

Questo processo è ipotizzato da Mandelbrot come un distribuzione stabile con parametri $\alpha = 1.7$, $\delta = 0$ e $\beta = 0$.

Tuttavia lo stesso autore afferma che il parametro β potrebbe essere in effetti leggermente negativo, in quanto le code negative sono leggermente più spesse delle code positive.

Taleb [Taleb, 2009], partendo dal lavoro teorico di Mandelbrot, afferma che non ci sono prove contro le ipotesi della distribuzione power-law dei rendimenti.

L'obiettivo però dev'essere quello di applicare in pratica il quadro teorico proposto.

Taleb evidenzia quattro problemi che nascono dal confronto tra la teoria di Mandelbrot e i modelli di finanza quantitativa tradizionali:

1. l'uso di forme di casualità troppo dipendenti dalla distribuzione Normale anche per descrivere le forme di casualità definibili "selvagge";
2. l'uso di modelli che si basano su proprietà vere solo asintoticamente;
3. l'uso di modelli che ipotizzano indipendenza temporale e quindi applicabilità della

distribuzione Normale;

4. problemi relativi alla calibrazione di modelli.

Finché non sappiamo se e dove la distribuzione dei rendimenti è limitata superiormente o troncata, è molto più sicuro modellarla utilizzando una distribuzione power-law, piuttosto che una Normale o una Lognormale, che potrebbero portare a errori, anche per quello che riguarda eventuali stress tests.

Questi quattro problemi portano tre conseguenze:

1. momenti non finiti non permettono mai derivazioni basate su espansioni e sul lemma di Ito;
2. i processi in generale non convergono necessariamente ad un ambito dove la distribuzione Gaussiana sia applicabile;
3. la parametrizzazione di modelli non è così semplice come potrebbe esserlo nel caso si ipotizzi una distribuzione Normale.

Taleb mostra come sia quasi impossibile stimare parametri consistenti da un campione finito di dati, che segue una distribuzione che ha momenti di ordine n o superiori non finiti, anche se i parametri da stimare si riferiscono a momenti di ordine inferiore a n .

Aggiunge anche che i momenti non finiti influiscono sulla formula di Black-Scholes-Merton [Black e Scholes, 1973], e lo fa mostrando l'andamento degli hedging errors nel caso si ipotizzi che le variazioni del sottostante seguano una power-law con varianza finita, una Normale e mostrando l'andamento degli hedging errors con dati reali.

Le variazioni di prezzo nei modelli tradizionali sono ipotizzate come un processo stocastico con le caratteristiche di un moto browniano geometrico:

$$\frac{dS}{S} = m dt + \sigma dZ$$

Sebbene questo sia conveniente dal punto di vista matematico, in quanto il prezzo al tempo $t + \Delta t$ diventa:

$$S_{t+\Delta t} = S_t e^{a+bz}$$

e il suo valore atteso è:

$$E[S_{t+\Delta t}] = S_t \int e^{a+bz} \phi(z) dz$$

le sue ipotesi sono spesso inaccettabili e ciò si riflette sulla realistica del modello.

Proposte alternative per quello che riguarda la modellizzazione della variazione dei prezzi sono un processo aritmetico:

$$S_{t+\Delta t} = a + bz + S_t$$

e un processo geometrico:

$$S_{t+\Delta t} = S_t(1 + a + bz)$$

Sebbene entrambi possano generare prezzi negativi, si rivelano in ogni caso più in linea con i dati osservati nella realtà.

Per quello che riguarda la dipendenza temporale invece, si fa notare che la distribuzione power-law mantiene invariato l'esponente α a prescindere dalla scala temporale che si utilizza, cioè a prescindere dalla dimensione di Δt che si sta prendendo in considerazione.

Questo significa che la distribuzione dei rendimenti è la medesima sia che lavoriamo su dati giornalieri, settimanali, mensili o di altra lunghezza temporale.

Non si è tuttavia a conoscenza di un modo per esprimere il processo matematicamente, né computazionalmente, senza ricorrere ad un qualsiasi processo che converga all'ambito gaussiano.

Si potrebbe d'altro canto provare a porre rimedio a questa difficoltà nel pricing di titoli, evitando di ricorrere a processi, e limitandosi a lavorare con distribuzioni discrete tra due periodi.

Capitolo 2

Modello di simulazione ABM

2.1 NetLogo

NetLogo è un ambiente di modellazione programmabile per simulare fenomeni naturali e sociali.

Il progetto è stato iniziato tra gli altri da Uri Wilensky nel 1999 ed è in continuo sviluppo al Center for Connected Learning and Computer-Based Modeling.

NetLogo è particolarmente adatto per la modellizzazione di sistemi complessi che si sviluppano nel tempo. Chi modella può dare istruzioni a centinaia o migliaia di "agenti", tutti operanti in maniera indipendente. In questo modo è possibile esplorare il collegamento tra il livello micro relativo al comportamento degli individui e il macro livello di schemi che emergono dall'interazione di molti individui.

NetLogo è un software semplice grazie al quale si possono facilmente eseguire simulazioni o addirittura costruirle. È inoltre abbastanza avanzato per essere utilizzato come un potente strumento per i ricercatori in molti campi.

NetLogo ha un'ampia documentazione e molte esercitazioni. Inoltre è dotato di una biblioteca di modelli, cioè di una grande raccolta di simulazioni che può essere utilizzata e modificata. Queste simulazioni riguardano svariate aree di ricerca scientifica, sia per quello che riguarda le scienze naturali, sia in relazione alle scienze sociali; vi sono per l'appunto modelli inerenti la biologia e la medicina, la fisica e la chimica, la matematica e l'informatica, l'economia e la psicologia sociale.

NetLogo fa parte della nuova generazione della serie di linguaggi multi-agente di modellazione che è iniziata con StarLogo. NetLogo aggiunge nuove importanti funzioni e un linguaggio e un'interfaccia utente ridisegnata.

StarLogo è un linguaggio di simulazione agent-based sviluppato da Mitchel Resnick, Eric Klopfer e altri al MIT Media Lab del MIT. Si tratta di un'estensione del linguaggio di programmazione Logo, scritto in Lisp e progettato per l'istruzione.

NetLogo è eseguito sulla macchina virtuale Java, in questo modo può funzionare su tutte le principali piattaforme (Mac, Windows, Linux, etc). I modelli possono anche essere eseguiti come applet Java in un browser web.

Il Java è un linguaggio di programmazione orientato agli oggetti, creato da James Gosling e altri ingegneri di Sun Microsystems.

I programmi scritti in linguaggio Java sono destinati all'esecuzione sulla piattaforma Java, ovvero saranno lanciati su una Java Virtual Machine e, a tempo di esecuzione, avranno accesso alle API della libreria standard. Ciò fornisce un livello di astrazione che permette alle applicazioni di essere interamente indipendenti dal sistema su cui esse saranno eseguite.

Il Lisp (List Processor) è un linguaggio di programmazione con implementazioni sia compilate sia interpretate, usato nei progetti di intelligenza artificiale. È stato ideato nel 1958 da John McCarthy come linguaggio formale, per studiare le equazioni di ricorsione in un modello computazionale. È un linguaggio di programmazione che si basa sul concetto di programma come funzione.

Questi agenti mobili chiamati "turtles" si muovono su una griglia di "patch", che sono a loro volta agenti programmabili.

Tutti gli agenti possono interagire tra di loro ed eseguire più operazioni contemporaneamente.

NetLogo inoltre può scambiare dati con altre applicazioni: il linguaggio comprende comandi che permettono di leggere o scrivere qualsiasi tipo di file di testo.

Ci sono anche strumenti per l'esportazione e l'importazione di dati in formati standard.

Lo stato completo del mondo può essere salvato e ripristinato in un formato che può essere facilmente aperto e analizzato con altri software.

Graficamente i dati possono essere esportati per essere analizzati con altri strumenti. Il contenuto della finestra grafica, o dell'intera interfaccia del modello, può essere salvato come immagine. Eventualmente è possibile trasformare le immagini in un filmato.

I modelli creati possono poi essere pubblicati sul web come applet Java.

NetLogo ha le sue origini in StarLisp [Lasser e Omohundro, 1986] e Logo [Papert, 1980], che a sua volta è un membro della famiglia Lisp.

Da Logo, è stata ereditata la "turtle".

Tuttavia in Logo, il programmatore controlla una singola tartaruga, mentre in NetLogo il modello può averne migliaia.

NetLogo segue la filosofia di Logo anche per quello che riguarda la facilità di utilizzo, fornendo un'interfaccia semplice e facilitazioni d'uso per i nuovi utenti.

Da StarLisp, un applicativo Lisp parallelo del 1980, è stato ereditato il multi-agent.

Il design attuale di NetLogo si basa sull'esperienza con ambienti precedenti, come ad esempio

StarLogoT [Wilensky, 1997].

Rispetto a StarLogoT sono state ridisegnate sia la lingua che l'interfaccia utente. NetLogo comprende quasi tutte le funzioni StarLogoT, oltre molte altre nuove. Molte delle nuove caratteristiche di NetLogo sono destinate a utenti che hanno come finalità la ricerca scientifica.

NetLogo è stato sviluppato dal 1999 ed ora è un software maturo, stabile e affidabile.

Anche se la base di utenti si è ampliata nel tempo, il tasso di segnalazioni di bug è andato decrescendo.

Rispetto alle prime versioni, i modelli ora sono eseguiti molto più velocemente.

L'accettazione di NetLogo nell'ambito della ricerca scientifica e a scopi educativi è ampia ed in crescita.

Un esempio di quanto appena affermato è il fatto che un numero crescente di corsi universitari sono ora insegnati, in tutto o in parte, utilizzando NetLogo.

Alcuni di questi corsi e workshop producono materiali resi disponibili online a fini consultativi e di sviluppo.

La pagina web di NetLogo infatti ha un'area apposita dove gli utenti possono caricare i modelli da condividere con la comunità.

Come linguaggio, le due grandi novità di NetLogo rispetto a Logo sono la possibilità di avere una pluralità di agenti e il calcolo simultaneo e interattivo.

Sebbene Logo non si limita alle applicazioni grafiche, è meglio conosciuto per la sua "grafica della turtle" in cui un essere virtuale o "turtle" si muove lasciando una scia dietro di sé.

NetLogo generalizza il concetto, essendo in grado di supportare centinaia o migliaia di tartarughe che possono muoversi e interagire.

Il mondo in cui la tartaruga si muove è una griglia di "patch", anch'esse programmabili.

Sia le tartarughe che le patch sono "agenti".

Tutti gli agenti possono interagire tra loro e svolgere più attività contemporaneamente.

NetLogo include anche un terzo tipo di agente, chiamato "observer".

In NetLogo è possibile avere solo un osservatore.

Nella maggior parte dei modelli, l'osservatore è il punto d'inizio della simulazione, attraverso l'emissione di istruzioni per le tartarughe e per le patch.

È possibile definire diverse "breed" di tartarughe, e variabili e comportamenti differenti possono essere associati ad ogni razza.

Un'importante caratteristica del linguaggio NetLogo è il cosiddetto "Agentset", o raccolta di agenti. Per esempio, l'insieme di tutte le tartarughe e l'insieme di tutte le patch sono agentsets. È inoltre possibile creare agentsets personalizzati, per esempio l'insieme di tutte le tartarughe rosse, o una

colonna di patch (l'insieme di patch con una data coordinata X). Gli Agentsets sono responsabili di gran parte della potenza espressiva di NetLogo.

Uno degli obiettivi principali per il progetto NetLogo è che i risultati siano scientificamente riproducibili, quindi è importante che i modelli operino in modo deterministico.

Questo significa che se si inserisce nel generatore di numeri casuali lo stesso "seed", il modello NetLogo eseguirà sempre gli stessi passi nello stesso ordine e produrrà esattamente i medesimi risultati, indipendentemente dal computer sul quale viene eseguita la simulazione.

Questo è garantito dal fatto che le librerie matematiche della piattaforma Java sono indipendenti.

Oltre a costruzioni speciali per il supporto della modellizzazione multi-agent, NetLogo comprende anche linee di programmazione standard, quali procedure, cicli, condizioni, ricorsione, stringhe, liste e così via.

Nella parte in alto della schermata di NetLogo sono presenti tre "etichette", denominate "Interface", "Information" e "Procedures".

Ognuna fa riferimento ad una diversa schermata del medesimo modello, ed è possibile visualizzare solamente una di queste tre schermate alla volta.

nAgents: 900
 p-rat: 0.5
 passLevelEso: 0.70

\$: 100
 passLevelRandom: Yes
 strategies: random

seed: 100
 rational: last close
 irrational: last close

vol-a: 1
 t: 161

mean [a] of turtles: 0.491
 cash of rat: 368450
 stocks of rat: -338
 wealth of rat: -81090

item 0 exePrice: 1330

item 0 logB: 1324.1214
 item 0 logS: 1331.3048

%-rat: 0.51

turtles with no stocks: 34

3D ticks: 500

exePrice

2.1.1 Interfaccia

Nella schermata di interfaccia è possibile guardare l'esecuzione del modello. Sono presenti in questa schermata anche strumenti che consentono di controllare e modificare ciò che accade all'interno del modello.

Quando si esegue NetLogo, la schermata di interfaccia è vuota.

La barra degli strumenti della schermata di interfaccia contiene i pulsanti che consentono di modificare, eliminare e creare elementi visualizzati nella stessa, e un menu che consente di selezionare diversi elementi dell'interfaccia, come ad esempio i pulsanti e gli slider.

I pulsanti, quando cliccati, possono eseguire un comando sia una volta sola, sia in maniera continuativa. Nel secondo caso il comando è eseguito fino a quando si fa nuovamente clic sul pulsante per interrompere l'azione.

Gli slider rappresentano variabili globali, che sono accessibili da tutti gli agenti. Sono utilizzati nei modelli come un modo rapido per cambiare il valore di una variabile senza dover modificare la procedura ogni volta. L'utente, infatti, può spostare il cursore ad un valore desiderato, e osservare immediatamente ciò che accade nel modello.

Gli switch sono una rappresentazione visiva di una variabile vero\falso. L'utente può impostare la variabile su true, attivando l'interruttore, o su false, disattivandolo.

I chooser consentono all'utente di scegliere un valore da assegnare ad una variabile globale, da un elenco di opzioni, presentate in un menu a discesa.

Le caselle di input sono variabili globali che contengono stringhe o numeri. L'utente può inserire il valore che intende assegnare alla variabile semplicemente digitandolo.

I monitor mostrano il valore di qualsiasi espressione. L'espressione potrebbe essere una variabile o un'espressione complessa. I monitor sono aggiornati automaticamente diverse volte al secondo.

I plot sono grafici in tempo reale dei dati che il modello sta generando.

Gli altri controlli nella barra degli strumenti di interfaccia consentono di modificare a piacimento gli aggiornamenti a vista e altre varie proprietà del modello.

Il Command Center permette di impartire comandi direttamente all'interno dell'interfaccia, senza aggiungere le procedure al modello. Questo è utile per il controllo e la manipolazione ad hoc di

agenti, senza modificare le procedure del modello.

2.2 Modello NetLogo di Mercato Finanziario

```
breed [IRRs IRR]
```

```
breed [RATLs RATL]
```

```
breed [RATSS RATS]
```

```
turtles-own[buy sell pass price cash stocks
```

```
           passLevel rat sl tp a in-mkt]
```

```
globals  [logB logS exePrice exePriceD
```

```
          returns returnsD orders t turns]
```

I modelli sviluppati con NetLogo consentono l'utilizzo di tre tipologie di variabili o liste: quelle di input dell'utente, quelle possedute dagli agenti e quelle globali.

All'interno del codice di programmazione è necessario definire soltanto le ultime due tipologie, in quanto le variabili di input sono già implicitamente definite nell'interfaccia, attraverso strumenti come sliders e choosers, che ne permettono anche l'assegnazione dei valori.

Nel modello di simulazione sono state assegnate alle “turtles”, cioè agli agenti, differenti breed, al fine di facilitarne la programmazione.

La breed è una variabile preesistente in NetLogo che possiedono tutti gli agenti; traducibile come specie, essa consente la suddivisione dell'insieme degli agenti in sottoinsiemi, che possono essere omogenei dal punto di vista di determinate caratteristiche degli agenti che li compongono. Ciò permette di semplificare molto il codice di programmazione in quanto consente di riferirsi in ogni momento solo al sottoinsieme degli agenti di interesse, piuttosto che alla sua totalità indistinta.

A seconda della breed che possiede un'agente, quest'ultimo si comporterà in una determinata

maniera.

La breed “IRRs” include tutti quegli agenti che non sono vincolati nell'immissione ordini nel mercato, cioè quegli agenti che possono immettere liberamente una volta ogni ciclo un'ordine nel mercato, sia in acquisto che in vendita.

La breed “RATLs” include tutti quegli agenti che hanno aperto una posizione lunga sul mercato, e che quindi devono chiudere.

Il loro comportamento è vincolato da ciò, in quanto fintanto che una turtle appartiene all'agentset “RATLs” potrà immettere sul mercato solo ordini di vendita, ad un prezzo che viene determinato a seconda della strategia scelta dall'utente (“stop-loss take-profit”, “take-profit only”).

La breed “RATs” include tutti quegli agenti che hanno aperto una posizione corta sul mercato.

Una turtle che appartiene all'agentset “RATs” potrà immettere sul mercato solo ordini di vendita, ad un prezzo che viene determinato anche in questo caso dalla strategia scelta dall'utente.

La lista turtles-own contiene tutte le variabili assegnate agli agenti, che possono assumere valori differenti per ogni agente. Alcune delle variabili contenute in turtle-own, nello specifico “passLevel” e “rat”, assumono un solo valore durante il corso della simulazione, mentre le altre possono assumere valori che cambiano nel corso della simulazione.

“Buy”, “sell” e “pass” sono variabili che cambiano nel tempo e possono assumere solo i valori True e False.

Queste tre variabili determinano se ad ogni ciclo l'agente decide di immettere un ordine in acquisto, in vendita o di non immettere ordini. Questo implica che ad ogni ciclo soltanto una delle tre variabili assumerà il valore True, mentre le restanti due avranno valore False.

Queste tre variabili vengono modificate ad ogni ciclo durante l'esecuzione dei blocchi “set-price”, “change-breed-RATL” e “change-breed-RATS”.

La variabile “price” ad ogni ciclo contiene il prezzo relativo all'ordine che l'agente intende immettere nel mercato.

Questa variabile viene modificata ad ogni ciclo durante l'esecuzione dei blocchi “set-price” e “set-sl-tp-price”.

La variabile “cash” contiene l'ammontare di denaro, positivo o negativo, di cui l'agente dispone per operare.

Il fatto che l'ammontare può essere anche negativo implica che in questo modello sono permesse

vendite allo scoperto e che si può liberamente prendere a prestito ad un tasso pari a 0.

Questa variabile viene modificata ad ogni ciclo durante l'esecuzione del blocco “book”.

La variabile “stocks” indica il numero di titoli che l'agente in ogni momento detiene in portafoglio, e può essere una quantità positiva o negativa.

Questa variabile viene modificata ad ogni ciclo durante l'esecuzione del blocco “book”.

“PassLevel” è una variabile contenente un valore compreso tra 0 e 1, fisso nel tempo, che indica la probabilità di ogni agente ad ogni ciclo di non immettere ordini nel mercato.

La variabile “rat” indica se l'agente è o meno “razionale” e può assumere i valori 0 e 1; nel caso assuma valore 1 l'agente può essere classificato come “razionale”, in quanto segue un qualche tipo di strategia per chiudere le posizioni aperte. Il valore della variabile “rat” è fisso nel tempo.

Le variabili “sl” e “tp” indicano i valori rispettivamente di stop-loss e take-profit che l'agente “razionale” utilizza per la chiusura della posizione aperta.

Queste tre variabili vengono modificate ad ogni ciclo durante l'esecuzione dei blocchi “change-breed-RATL” e “change-breed-RATS”.

La variabile “a” può assumere un valore compreso tra 0 e 1, valore che può cambiare nel corso della simulazione.

La variabile “a” è da intendersi come indicatrice della propensione al rischio dell'agente: a valori più elevati di “a” vi sarà la tendenza dell'agente a fare offerte con prezzo più distante da quello di mercato in quel determinato istante, così come a valori più elevati di “a” vi sarà la tendenza dell'agente “razionale” di adottare stop-loss e take-profit più ampi.

Questa variabile viene modificata ad ogni ciclo durante l'esecuzione del blocco “change-breed-IRR”.

La variabile “in-mkt” può assumere valore 0 o 1 ed assume valore 1 quando un agente RATL o RATS immette un ordine nel book, a prescindere che l'ordine venga eseguito immediatamente, successivamente o non venga eseguito.

Questa variabile è più che altro strumentale al modo con cui si è programmato il modello, non ha dunque un significato economico di grande interesse.

Questa variabile viene modificata ad ogni ciclo durante l'esecuzione dei blocchi “go”, “change-breed-IRR” e “set-sl-tp-price”.

Come si può notare non si fa nessun riferimento alle quantità di titoli negoziabili con ogni ordine, in

quanto nel modello gli agenti possono comprare o vendere soltanto un titolo alla volta.

Globals è usato per definire variabili, liste o costanti che devono essere utilizzate in molte parti del programma.

Questa parola chiave, così come altre parole chiave quali `breed`, `<breeds>-own`, `patch-own`, e `turtles-own`, può essere utilizzata solo all'inizio di un programma, prima di ogni definizione di funzione. Essa definisce nuove variabili globali.

Le variabili globali sono denominate tali perché sono accessibili da tutti gli agenti e possono essere utilizzate ovunque in un modello.

“LogB” e “logS” sono due liste utilizzate nel codice come se fossero i due lati del book: in ogni istante all'interno delle due liste troviamo tutti gli ordini, in acquisto o in vendita, immessi nel mercato ma non ancora eseguiti.

La lista “exePrice” contiene tutti i prezzi a cui gli scambi sono stati eseguiti: ad ogni transazione avvenuta, il prezzo è registrato all'interno di questa lista.

Se ci trovassimo nel mondo reale questa lista rappresenterebbe la registrazione di tutti i movimenti di prezzo di un titolo, tick-by-tick.

La lista “exePriceD” contiene tutti i prezzi con cui si conclude ciascun ciclo di simulazione.

Se ci trovassimo nel mondo reale questa lista rappresenterebbe la registrazione di tutti i prezzi di chiusura di un titolo, giorno per giorno.

In questo caso stiamo assumendo che ogni tick della simulazione, cioè ogni ciclo, sia assimilabile ad un giorno di negoziazione.

La lista “returns” contiene tutti i rendimenti istantanei del titolo.

La lista “returnsD” contiene tutti i rendimenti giornalieri del titolo.

La lista “orders” contiene tutti gli ordini immessi fino a quel momento nel mercato.

Questa lista è utile per un'analisi approfondita di quanto avvenuto fino a quel momento all'interno della simulazione¹.

Ogni elemento contenuto nella lista “orders”, infatti, contiene le seguenti informazioni, riferibili temporalmente all'istante in cui è stato generato e immesso l'ordine nel book:

- prezzo al quale viene immesso l'ordine,
- numero identificativo dell'agente che immette l'ordine,
- ticks, cioè ciclo di simulazione durante il quale è stato generato e immesso l'ordine
- t, cioè istante, all'interno di quel determinato tick, in cui è stato generato e inserito

¹ Un esempio sull'utilizzo della lista “orders” può essere il seguente: inserendo all'interno del Command Center il comando “foreach orders [if item 1 ? = 834 [show ?]]” è possibile visualizzare tutti gli ordini immessi dall'agente 834 durante tutto il corso della simulazione.

l'ordine,

- valore della variabile “a” dell'agente che ha generato l'ordine, nell'istante in cui lo ha generato,
- valore della variabile “rat” dell'agente che ha generato l'ordine,
- ultimo prezzo scambiato sul mercato,
- prezzo del miglior bid\ask, cioè prezzo del miglior ordine opposto presente in quell'istante nel mercato,
- open\close, cioè un valore pari a 1 nel caso l'ordine sia immesso per aprire una posizione o sia immesso da un agente “irrazionale”, o un valore pari a 0 nel caso l'ordine serva per chiudere una posizione aperta da un agente “razionale”,
- buy\sell, cioè un valore pari a 1 nel caso l'ordine sia in acquisto, pari a 0 nel caso sia in vendita.

La variabile t può assumere valori interi nell'intervallo $[0, \infty]$, ed all'interno di ogni ciclo rappresenta il numero di scambi avvenuti fino a quel momento.

Questa variabile è azzerata all'inizio di ciascun ciclo di simulazione.

Se ci trovassimo nel mondo reale, questa variabile rappresenterebbe i tick all'interno di una giornata di negoziazione.

La lista “turns” contiene il nome di tutti gli agenti che verranno chiamati ad immettere un ordine nel mercato, nell'ordine con cui verranno chiamati a farlo.

```
to setup
```

```
ca
```

```
random-seed seed
```

```
set exePrice []
```

```
set exePriceD []
```

```
set exePrice fput 1000 exePrice
```

```
set exePriceD fput 1000 exePriceD
```

```
set orders []
```

```

create-IRRs nAgents

let side sqrt nAgents

let step max-pxcor / side

ask IRRs
[set shape "person"
 set size 1
 set stocks 0
 set cash 0
 set in-mkt 0
 set a random-float 1
 ifelse random-float 1 > p-rat [set rat 0 set color white]
                               [set rat 1 set color grey]
 ifelse passLevelRandom = "Yes" [set passLevel random-float 1]
                               [set passLevel passLevelEso]]

let an 0
let x 0
let y 0

while [an < nAgents]
  [if x > (side - 1) * step [set y y + step
                           set x 0]

   ask turtle an

```

```
[setxy x y]

set x x + step

set an an + 1

]

end
```

“Setup” è il blocco di procedure che imposta le condizioni del modello allo stato iniziale. Questo implica fra le altre cose la creazione degli agenti nel numero e nel tipo stabilito dall'utente e l'assegnazione dei valori di partenza alle variabili.

Il comando `ca`, abbreviazione di `clear-all`, è usato per cancellare tutti i risultati e le modifiche derivanti dalle precedenti simulazioni, al fine dunque di iniziarne una nuova in un ambiente che presenti le condizioni desiderate.

Il comando `random-seed` imposta il seme del generatore di numeri pseudo-casuali. Il seme può essere un numero intero nell'intervallo supportato dal NetLogo.²

I numeri casuali generati da NetLogo sono i cosiddetti numeri "pseudo-casuali". Questo è tipico nella programmazione informatica. Ciò significa che appaiono casuali, ma in realtà sono generati da un processo deterministico.

Con processo deterministico intendiamo un processo che genera lo stesso risultato ogni volta, se le condizioni di partenza del processo sono le medesime, se cioè in questo caso specifico si inizia la simulazione con le stesse condizioni iniziali e il medesimo seme.

Nel contesto di modelli scientifici, i numeri pseudo-casuali sono realmente desiderabili. Questo perché è importante che un esperimento scientifico sia riproducibile, cioè che chiunque possa riprodurre la simulazione e ottenere gli stessi risultati che sono stati ottenuti da altri nelle medesime condizioni.

Pertanto il generatore di numeri casuali presente in NetLogo può essere avviato con un certo valore di seme ed una volta che il generatore è stato impostato con il comando `random-seed`, genera, da quel momento in poi, sempre la stessa sequenza di numeri casuali.

Si noti, tuttavia, che è garantita la riproducibilità della simulazione solo se si utilizza la stessa

² Da -9007199254740992 a 9007199254740992.

versione di NetLogo. È possibile, infatti, che il generatore di numeri casuali sia differente a seconda della versione che si utilizzi. Nella versione di NetLogo 4.1.3, utilizzata per la creazione del modello di simulazione, è presente un algoritmo di generazione di numeri pseudo-casuali noto come Mersenne Twister.³

Se non si imposta il seme casuale, NetLogo lo imposta in automatico ad un valore basato sulla data e ora correnti. Poiché non c'è modo di scoprire in quest'ultimo caso quale sia il seme casuale scelto, se si desidera che il modello sia riproducibile, è necessario impostare il seme casuale.

I comandi di NetLogo con "random" nei loro nomi, come ad esempio random, random-float, e così via, non sono gli unici che usano numeri pseudo-casuali. Molte altre operazioni richiedono delle scelte casuali. Per esempio, gli agentsets sono sempre in ordine casuale, così in questo modo i comandi one-of e n-of estraggono agenti dalla totalità in modo casuale.

Successivamente si procede a definire le liste "exePrice", "exePriceD" e "orders" come liste vuote.

All'interno delle liste "exePrice" e "exePriceD" è inserito l'elemento 1000, che è il prezzo di partenza di default della simulazione, a prescindere dai settaggi impostati dall'utente.

I comandi fput e lput consentono di inserire un elemento all'inizio o alla fine di una lista.

Il comando create-IRRs genera gli agenti, nel numero impostato dall'utente, che corrisponde nel linguaggio di programmazione alla variabile "nAgents"⁴.

La variabile "nAgents" può assumere valori interi da 1 a 900 e viene controllata attraverso uno slider presente nell'interfaccia.

In seguito si creano e definiscono due variabili, "side" e "step", che sono utilizzate per ottenere una disposizione spaziale ordinata degli agenti nell'interfaccia.

Per la definizione di queste due variabili è stato utilizzato il comando let, che crea una nuova variabile locale e le assegna un determinato valore. Una variabile o lista locale si differenzia da una globale in quanto esiste solo all'interno del blocco di comandi in cui è creata.

Per blocco di comandi si intende una parte di codice che inizia con il comando to e termina con il comando end.

Le righe di codice successive si riferiscono a comandi e variabili di ogni singolo agente, e proprio

³ Mersenne Twister è un algoritmo per la generazione di numeri pseudocasuali di tipo lineare congruenziale sviluppato nel 1997 da Makoto Matsumoto e Takuji Nishimura [Matsumoto e Nishimura, 1998].

⁴ L'utilizzo del comando create-IRRs in NetLogo è alternativo ai comandi create-turtles e ask turtles [set breed IRRs], in quanto con entrambi si ottiene il medesimo risultato. In linea di principio si è cercato, come in questo caso, di programmare il codice nella maniera più snella e sintetica possibile, ma che fosse ciò nonostante al contempo di immediata comprensibilità.

per questo motivo all'inizio è usato il comando ask con riferimento al sottoinsieme "IRRs".⁵

Le variabili a cui viene assegnato un valore sono shape e size, che sono due variabili preesistenti in NetLogo e indicano la forma e la dimensione visualizzata degli agenti, "stocks", "cash", "in-mkt", "a", "rat" e "passLevel".

La variabile shape riceve valore "person".

La variabile size è impostata sul valore 1.

Le variabili "stocks", "cash" e "in-mkt" prendono valore 0.

La variabile "a" riceve un valore casuale⁶ compreso tra 0 e 1, differente per ogni agente.

La riga di codice successiva assegna, ad ogni agente, il valore 1 alla variabile rat con probabilità pari a p-rat, oppure 0 altrimenti.

Infine viene assegnato un valore alla variabile passLevel, a seconda che l'utente scelga se essa sia impostata esogenamente o generata casualmente dal modello.

Nel primo caso la variabile "passLevel" assumerà un valore pseudo-casuale compreso tra 0 e 1, differente per ogni agente.

Nel secondo caso la variabile "passLevel" avrà valore fisso nel tempo pari al valore attribuito nell'interfaccia alla variabile "passLevelEso".

Successivamente si creano le variabili locali "an", "x" e "y".

Il comando while crea un ciclo, cioè fintanto che una condizione si verifica, si ripete il comando associato al ciclo.

In questo caso la condizione è che la variabile "an", che rappresenta il numero identificativo di ogni agente, sia minore del numero totale degli agenti, cioè si esegue il comando all'interno del ciclo per ciascuno degli agenti creati.

Il comando eseguito per ogni agente dispone gli agenti in maniera ordinata nello spazio, riempiendo il mondo per file orizzontali partendo dall'angolo in basso a sinistra (origine degli assi) e terminando nell'angolo in alto a destra.

Per fare ciò all'interno del ciclo while agli agenti viene chiesto di impostare le proprie coordinate

⁵ In questo particolare caso, poiché le turtles all'inizio sono create tutte con breed "IRRs", sarebbe stato indifferente riferirsi all'insieme di tutti gli agenti, e pertanto utilizzare il comando ask turtles.

⁶ Ogni volta che viene attribuito un valore casuale utilizzando il comando random-float <n>, si sta generando un numero pseudo-casuale compreso nell'intervallo $(0, n)$, utilizzando una distribuzione di probabilità uniforme.

Peraltro si avrà: $F(x) = \frac{x}{n}$ per $x \in (0, n)$ e $f(x) = \frac{1}{n}$ per $x \in (0, n)$.

spaziali ai valori “x” e “y”, che vengono opportunamente aumentati ogni volta che un singolo agente viene disposto nello spazio.

La distanza tra ciascun agente, sia orizzontalmente che verticalmente, è sempre pari al valore della variabile “step”.

Il comando end chiude il blocco di comandi, in questo caso quello denominato setup.

```
to go

set logB []
set logS []
ask turtles [set in-mkt 0]
set t 0
set turns []
ask IRRs [set turns fput who turns]

while [not empty? turns]
[ask turtle item 0 turns
  [ifelse breed = IRRs
    [set-price]
    [set-sl-tp-price]
  book]
  set turns but-first turns
  if strategies = "stop-loss take-profit"
    [if (any? RATLs with [in-mkt = 0 and (sl > item 0 exePrice or
tp < item 0 exePrice)])]
```

```

    or (any? RATSS with [in-mkt = 0 and (tp > item 0 exePrice or
sl < item 0 exePrice)])

    [ask (turtle-set RATLs with [in-mkt = 0 and (sl > item 0
exePrice or tp < item 0 exePrice)]

        RATSS with [in-mkt = 0 and (tp > item 0
exePrice or sl < item 0 exePrice)])

    [set turns fput who turns

    set turns remove-duplicates turns]]]

if strategies = "take-profit only"

[if (any? RATLs with [in-mkt = 0 and (tp < item 0 exePrice)])

    or (any? RATSS with [in-mkt = 0 and (tp > item 0 exePrice)])

    [ask (turtle-set RATLs with [in-mkt = 0 and (tp < item 0
exePrice)]

        RATSS with [in-mkt = 0 and (tp > item 0 exePrice)])

    [set turns fput who turns

    set turns remove-duplicates turns]]]]

set exePriceD fput item 0 exePrice exePriceD

tick

end

```

Il blocco denominato “go” contiene tutte le procedure necessarie a far sì che il modello generi la simulazione, quindi è possibile definirlo come il “motore” del modello.

Inizialmente si procede a svuotare il book del nostro mercato.

Si imposta la variabile “in-mkt” di ogni agente su 0, ciò significa impostare ogni agente come se non abbia ancora operato sul mercato.

È azzerata la variabile “t”, ed è definita la lista “turns”.

La lista “turns” rappresenta l'ordine che gli agenti seguono per immettere ordini nel mercato.

Ciò significa che il primo elemento di tale lista, in ogni istante, identificherà il prossimo agente ad immettere un ordine nel mercato.

La lista “turns” è poi riempita con i numeri identificativi di ogni agente IRRs, seguendo un ordine pseudo-casuale.

All'inizio di ogni ciclo infatti gli agenti che possono immettere ordini nel mercato sono sia gli agenti cosiddetti “irrazionali”, sia gli agenti “razionali” che non abbiano posizioni già aperte. Questo insieme di agenti richiade perfettamente nella definizione di “IRRs”.

A questo punto è presente un ciclo while, che sicuramente non appare di immediata comprensione ma che tuttavia svolge un ruolo primario per il funzionamento del modello.

Il comando while è strutturato in NetLogo nel seguente modo: mentre [condizioni] [comandi].

Non appena le condizioni diventano false, si esce dal ciclo. In caso contrario, sono eseguiti e ripetuti i comandi.

Nel nostro caso specifico, la condizione che chiude il ciclo è che non ci siano più agenti che debbano inserire ordini nel mercato, cioè che non ci siano più agenti nella lista “turns”.

In caso contrario, si chiede all'agente corrispondente al primo elemento della lista “turns”, di formare un prezzo relativo all'ordine che intende immettere nel mercato, e successivamente di immettere tale ordine.

La condizione ifelse breed = IRRs serve per differenziare la formazione del prezzo nel caso l'agente sia “IRRs”, cioè voglia aprire una posizione, o nel caso l'agente sia “RATLs” o “RATSs”, cioè l'agente debba chiudere la posizione, poiché sono scattati stop-loss o take-profit.

Nel primo caso si fa riferimento al blocco “set-price”, mentre nel secondo al blocco “set-sl-tp-price”.

Il blocco denominato “book” contiene al suo interno tutte le procedure che permettono l'immissione di un ordine e la sua eventuale esecuzione.

Appena è terminata l'esecuzione dei comandi contenuti in “book”, si elimina l'agente che ha appena operato dalla lista “turns”, eliminandone appunto il primo elemento attraverso il comando but-first.

Poiché al termine del blocco “book” c'è la possibilità che il prezzo del titolo sia cambiato, in seguito ad uno scambio avvenuto, si va a controllare se per caso siano scattati stop-loss o take-profit negli

agenti “RATLs” e “RATs”.

Questa intenzione si traduce nel codice in due gruppi analoghi di procedure.

La differenza sta nel fatto che, come possiamo notare dalle condizioni iniziali di ciascuno dei due gruppi: `if strategies = “stop-loss take-profit”` e `if strategies = “take-profit only”`, il primo gruppo viene utilizzato quando la strategia che gli agenti “razionali” seguono è impostata su “stop-loss take-profit”, mentre il secondo quando quando la strategia che gli agenti “razionali” seguono è impostata su “take-profit only”.

Attraverso la condizione `if any?` si chiede a NetLogo di verificare se ci sia qualche agente “razionale” che debba chiudere una posizione poiché il prezzo ha superato uno stop-loss o un take-profit.

Nel caso la condizione sia vera, si chiede all'insieme di tutti questi agenti che hanno posizioni da chiudere di inserire il numero identificativo, in ordine casuale, all'inizio della lista “turns”.

Ciò significa che questi agenti saranno i primi ad operare.

Per raggruppare agenti con caratteristiche e breed differenti in un unico insieme o agentset è stato utilizzato il comando `turtle-set`.

Per richiamare agenti in base a caratteristiche specifiche, in questo caso in base ad un confronto tra ultimo prezzo eseguito e stop-loss e/o take-profit, è stato utilizzato il comando `with`.

Poiché in linea teorica è possibile che lo stesso agente venga richiamato più volte attraverso questi procedimenti, è stato inserito il comando `remove-duplicates` in riferimento alla lista “turns”.

Ricapitolando quanto detto finora in altre parole, la lista “turns” contenente i turni che gli agenti seguono per immettere ordini nel mercato, è generata una prima volta, all'inizio di ogni ciclo di simulazione, dagli agenti “IRRs” che vogliono aprire una posizione. Successivamente ogni volta che è eseguito un ordine alla lista sono aggiunti i numeri identificativi degli agenti “RATLs” e “RATs” che devono chiudere la propria posizione.

Quando si interrompe il ciclo `while` significa che non ci sono più ordini da immettere nel mercato, poiché la lista “turns” è vuota, e quindi è come se ipoteticamente terminasse la giornata di negoziazione.

Il blocco “go” si conclude con l'inserimento del prezzo di chiusura della giornata di negoziazione all'interno della lista “`exePriceD`” e con l'avanzamento dei tick del modello.

to book

```
if not pass
  [let tmp []
    set tmp lput price tmp
    set tmp lput who tmp
    set tmp lput ticks tmp
    set tmp lput t tmp
    set tmp lput a tmp
    set tmp lput rat tmp
    set tmp lput item 0 exePrice tmp
    ifelse buy [
      ifelse not empty? logS [set tmp lput item 0 (item 0 logS)
tmp]
      [set tmp lput "e" tmp]][
      ifelse not empty? logB [set tmp lput item 0 (item 0 logB)
tmp]
      [set tmp lput "e" tmp]]
    ifelse breed = IRRs [set tmp lput 1 tmp][set tmp lput 0 tmp]
    ifelse buy [set tmp lput 1 tmp][set tmp lput 0 tmp]
    set orders fput tmp orders

    if buy [set logB lput tmp logB]
    set logB reverse sort-by [item 0 ?1 < item 0 ?2] logB
    if (not empty? logB and not empty? logS) [
```

```

if item 0 (item 0 logB) >= item 0 (item 0 logS)
  [set exePrice fput item 0 (item 0 logS) exePrice
  let agB item 1 (item 0 logB)
  let agS item 1 (item 0 logS)
  set t t + 1
  graph
  ask turtle agB
    [set stocks stocks + 1
    set cash cash - item 0 exePrice
    if strategies != "random" [
      ifelse breed = IRRs
        [change-breed-RATL]
        [change-breed-IRR]]
  ask turtle agS
    [set stocks stocks - 1
    set cash cash + item 0 exePrice
    if strategies != "random" [
      ifelse breed = IRRs
        [change-breed-RATS]
        [change-breed-IRR]]
  set logB but-first logB
  set logS but-first logS]]

if sell [set logS lput tmp logS]
set logS sort-by [item 0 ?1 < item 0 ?2] logS

```

```

if (not empty? logB and not empty? logS) [
  if item 0 (item 0 logB) >= item 0 (item 0 logS)
    [set exePrice fput item 0 (item 0 logB) exePrice
     let agB item 1 (item 0 logB)
     let agS item 1 (item 0 logS)
     set t t + 1
     graph
     ask turtle agB
       [set stocks stocks + 1
        set cash cash - item 0 exePrice
        if strategies != "random" [
          ifelse breed = IRRs
            [change-breed-RATL]
            [change-breed-IRR]]]
     ask turtle agS
       [set stocks stocks - 1
        set cash cash + item 0 exePrice
        if strategies != "random" [
          ifelse breed = IRRs
            [change-breed-RATS]
            [change-breed-IRR]]]
     set logB but-first logB
     set logS but-first logS]]
]

```

end

Il blocco denominato “book” è il nucleo del modello, in quanto si occupa dell'inserimento degli ordini nel mercato, nonché della loro esecuzione.

Nel mercato reale, per book si intende l'insieme delle proposte di acquisto e di vendita, ordinate in base al prezzo (decescente se in acquisto e crescente se in vendita) presenti in ogni istante in un determinato mercato per un determinato prodotto finanziario.

Per tutte le singole proposte in acquisto e in vendita, il book riporta l'indicazione degli operatori proponenti (rappresentata da un codice), il codice dello strumento da trattare e le modalità di esecuzione delle proposte, in relazione alla quantità, alle condizioni di prezzo e al tempo. A parità di prezzo vale la priorità dell'orario di immissione della proposta.

Se una proposta viene modificata per la quantità o per il prezzo, essa perde la priorità temporale acquisita.

Durante le contrattazioni gli ultimi prezzi scambiati subiscono variazioni istante per istante e questo vale per tutti i titoli quotati. Il prezzo può passare in un istante da 33.10 a 33.15 per poi ritornare a 33.10: questi sono i prezzi degli ultimi scambi che corrispondono al prezzo in denaro o a quello in lettera, per cui in ogni momento esistono due prezzi per ogni titolo, il miglior denaro (a cui si può vendere subito) e la migliore lettera (a cui si può acquistare subito).

Esistono regole ben precise che riguardano il prezzo degli ordini inseriti con il limite di prezzo. Ad esempio non è possibile inserire un ordine in acquisto o in vendita a 31.22 ma bisogna scegliere fra 31.20 e 31.25 in quanto ad esempio il tick minimo corrisponde a 0.05 (per quotazioni superiori a 30 euro).

I tick di negoziazione sono fissati in base al prezzo di apertura e sono strettamente legati a quel prezzo.

Il blocco, così come altre caratteristiche del modello, è stato concepito ispirandosi fortemente alle peculiarità appena esposte di un reale book di mercato, come si noterà approfondendone l'analisi.

Innanzitutto gli agenti che hanno accesso al book sono quelli che hanno intenzione di acquistare o vendere, cioè coloro i quali hanno la propria variabile “pass” che presenta valore False, come si può notare dalla condizione if che precede l'intero insieme di procedure.

Nel caso in cui la suddetta condizione sia vera, si procede alla creazione dell'offerta.

L'offerta è creata, attraverso il comando `let`, come una lista denominata “`tmp`” composta da dieci elementi:

1. “`price`”, cioè il prezzo al quale viene immesso l'ordine,
2. `who`, cioè il numero identificativo dell'agente che immette l'ordine,
3. `ticks`, cioè ciclo di simulazione durante il quale è stato generato e immesso l'ordine
4. “`t`”, cioè istante in cui è stato generato e inserito l'ordine,
5. “`a`”, cioè il valore della variabile “`a`” dell'agente in quel determinato istante,
6. “`rat`”, cioè il valore della variabile “`rat`” dell'agente,
7. `item 0 exePrice`, cioè l'ultimo prezzo scambiato sul mercato,
8. `item 0 (item 0 logS)\item 0 (item 0 logB)`⁷, prezzo del miglior `ask\bid`,
9. `1\0`, cioè un valore pari a 1 nel caso l'ordine sia immesso per aprire una posizione o sia immesso da un agente “irrazionale”, o un valore pari a 0 nel caso l'ordine serva per chiudere una posizione aperta da un agente “razionale”,
10. `1\0`, cioè un valore pari a 1 nel caso l'ordine sia in acquisto, pari a 0 nel caso sia in vendita.

L'ordine così generato, prima di essere immesso nel mercato e processato, è immagazzinato all'interno della lista “`orders`”.

La quantità di informazione che si è deciso inserire in ogni singolo ordine non è necessaria per il corretto funzionamento del blocco “`book`”, ma è dovuta all'intenzione di poter analizzare i risultati della simulazione avendo un potente strumento aggiuntivo. La lista “`orders`”, nella quale sono registrati tutti gli ordini che sono generati nel corso della simulazione, grazie al suo contenuto informativo funge quasi da “scatola nera” del modello: ci aiuta cioè nel ricostruire cosa sia successo in ogni determinato istante della simulazione.

A questo punto il blocco si biforca in due sottogruppi di procedure, simili e a tratti speculari, a seconda che l'ordine appena generato sia in acquisto o in vendita.

Nel caso sia un ordine in acquisto, l'ordine viene inserito nel `book` dal lato denaro, cioè la lista “`tmp`” viene inserita come elemento in fondo alla lista “`logB`”.

La lista “`logB`” viene poi ordinata in ordine decrescente di prezzo, attraverso il comando `reverse sort-by`, e, una volta verificato che sia la lista “`logB`” sia la lista “`logS`” non siano vuote, si procede alla ricerca di un ordine opposto da far combaciare con quello appena inserito.

In particolare, se esiste un ordine in vendita con prezzo inferiore o uguale a quello di acquisto appena inserito, si procede all'esecuzione dell'ordine:

⁷ Nel caso le liste “`logS`” o “`logB`” siano vuote, è inserito all'interno dell'ordine, in posizione 7, il carattere “`e`”.

- il prezzo, che in questo caso sarà quello di vendita, essendo l'ordine eseguito “al meglio”, viene inserito all'inizio della lista “exePrice”,
- vengono richiamati gli agenti che hanno inserito rispettivamente l'ordine in acquisto e in vendita, grazie al numero identificativo presente negli ordini, item 1 (item 0 logB) e item 1 (item 0 logS),
- si aumenta di uno il valore della variabile “t”, cioè si fanno avanzare di una unità gli ipotetici tick del mercato,
- si disegna il nuovo prezzo eseguito, attraverso il blocco “graph”,
- si provvede allo scambio di denaro e titolo, agendo opportunamente sulle variabili cash e stocks,
- nel caso la strategia impostata dall'utente sia diversa da quella “random”, la procedura continua in maniera differente a seconda che l'agente abbia breed “IRRs” o meno. Nel primo caso, si va a richiamare il blocco “change-breed-RATL” o “change-breed-RATS”, si va a vedere cioè se l'agente è dotato di “razionalità” e se perciò deve cambiare la sua breed in “RATL” o “RATS”, mentre nel secondo caso l'agente, che è obbligatoriamente un agente “RATS” o “RATL”, avendo chiuso la sua posizione, ritornerà ad essere un agente “IRR” attraverso le procedure presenti nel blocco “change-breed-IRR”
- infine si provvede all'eliminazione dal book degli elementi rappresentanti gli ordini appena eseguiti, attraverso i comandi but-first riferiti alle liste “logB” e “logS”.

Nel caso l'ordine appena immesso sia un ordine in vendita, si procede in modo analogo e speculare.

L'ordine viene inserito nel book dal lato lettera, cioè in fondo alla lista logS.

La lista logS viene poi ordinata in ordine crescente di prezzo e, verificato che sia la lista logB sia la lista logS non siano vuote, si procede alla ricerca di un ordine opposto da far combaciare con quello appena inserito.

In particolare, se esiste un ordine in acquisto con prezzo superiore o uguale a quello di vendita appena inserito, si procede all'esecuzione dell'ordine:

- il prezzo, che in questo caso sarà quello di acquisto, essendo l'ordine eseguito “al meglio”, viene inserito all'inizio della lista exePrice,
- vengono richiamati gli agenti che hanno inserito rispettivamente l'ordine in acquisto e in vendita, grazie al numero identificativo presente negli ordini, item 1 (item 0 logB) e item 1

(item 0 logS),

- si aumenta di uno il valore della variabile “t”, cioè si fanno avanzare di una unità gli ipotetici tick del mercato,
- si disegna il nuovo prezzo eseguito, attraverso il blocco “graph”,
- si provvede allo scambio di denaro e titolo, agendo opportunamente sulle variabili cash e stocks,
- nel caso la strategia impostata dall'utente sia diversa da quella “random”, la procedura continua in maniera differente a seconda che l'agente abbia breed “IRRs” o meno. Nel primo caso, si va a richiamare il blocco “change-breed-RATL” o “change-breed-RATS”, si va a vedere cioè se l'agente è dotato di “razionalità” e se perciò deve cambiare la sua breed in “RATL” o “RATS”, mentre nel secondo caso l'agente, che è obbligatoriamente un agente “RATS” o “RATL”, avendo chiuso la sua posizione, ritornerà ad essere un agente “IRR” attraverso le procedure presenti nel blocco “change-breed-IRR”
- infine si provvede all'eliminazione dal book degli elementi rappresentanti gli ordini appena eseguiti, attraverso i comandi but-first riferiti alle liste “logB” e “logS”.

```
to graph
```

```
set-current-plot "exePrice"
```

```
plot item 0 exePrice
```

```
set-current-plot "vol-a"
```

```
plot mean [a] of turtles
```

```
end
```

Il blocco “graph” svolge la funzione di rappresentare graficamente nel tempo l'andamento delle variabili di interesse: nello specifico nel modello sviluppato sono rappresentati l'andamento nel tempo del prezzo del titolo e l'evoluzione del parametro a nel corso della simulazione.

Il comando `set-current-plot` imposta come grafico di riferimento quello che è specificato nel comando: nel primo caso “`exePrice`” e nel secondo “`vol-a`”.

Tutti i comandi successivi si riferiscono al grafico selezionato, mediante il comando `set-current-plot`.

Successivamente si provvede alla rappresentazione vera e propria, utilizzando il comando `plot`, che disegna il valore della variabile indicata in base alle caratteristiche del grafico a cui fa riferimento.

I valori che di volta in volta sono disegnati sono rispettivamente il primo elemento della lista “`exePrice`” (item 0 `exePrice`) e il valore della media della variabile “ a ” di tutti gli agenti in quel determinato istante (`mean [a] of turtles`).

```
to save_results

let exePrice2 exePrice
set exePrice remove-item ((length exePrice) - 1) exePrice
set exePrice2 remove-item 0 exePrice2
set returns []

(foreach exePrice exePrice2 [set returns lput precision (ln ?1 -
ln ?2) 4 returns])

file-open "C:\\Users\\Andrea\\Desktop\\prova.txt"
foreach returns [file-print ?]

file-close
```

end

Il blocco denominato “save_results” serve per la generazione di un file di testo contenente le informazioni che si intendono analizzare al di fuori dell'ambiente NetLogo.

In questo caso è generato un file di testo (.txt) contenente la lista dei rendimenti istantanei generati dalla simulazione, che verrà analizzato con il software Maxima.

All'inizio del blocco è creata una lista temporanea denominata “exePrice2”, strumentale alla generazione dei rendimenti.

La lista è creata identica alla lista “exePrice”, che contiene tutti i prezzi eseguiti.

Successivamente alla lista “exePrice” è rimosso l'ultimo elemento, che per costruzione è il prezzo 1000, mentre alla lista “exePrice2” è rimosso il primo elemento, cioè l'ultimo prezzo eseguito.

Le liste così modificate contengono la prima i prezzi in ogni istante, la seconda i prezzi dell'istante precedente.

Detto in altre parole, ad ogni posizione, la lista “exePrice” conterrà il prezzo corrispondente all'istante “t”, mentre la lista “exePrice2” nella medesima posizione conterrà il prezzo all'istante “t-1”.

Sfruttando ciò, è generata la lista dei rendimenti, sottraendo al logaritmo di ogni elemento della prima lista, il logaritmo dell'elemento corrispondente della seconda lista.

Ad esempio sottraendo al logaritmo dell'elemento 0 della lista “exePrice” il logaritmo dell'elemento 0 della lista “exePrice2”, otterremo il rendimento istantaneo in “t”, seguendo la formula

$$r_t = \ln P_t - \ln P_{t-1} .$$

Il comando foreach esegue proprio questa procedura, aggiungendo mano a mano elementi alla lista “returns”, mentre il comando precision fa in modo che i dati siano omogenei per quello che riguarda la loro lunghezza in cifre: i rendimenti così generati avranno tutti 4 posti dopo la virgola.

Il comando file-open interpreta una stringa di testo come il percorso di un file, e apre quel determinato file.

Il comando file-print scrive il valore richiesto in un file aperto, in questo caso quello aperto grazie al

comando file-open.

Il valore nel caso del comando file-print è seguito da un ritorno a capo.

Il comando file-print è inserito all'interno di un ciclo foreach, poiché il comando file-print dev'essere eseguito per ogni elemento della lista “returns”.

Il comando file-close chiude il file aperto in precedenza, salvando le modifiche effettuate.

```
to save_resultsDay

let exePriceD2 exePriceD
set exePriceD remove-item ((length exePriceD) - 1) exePriceD
set exePriceD2 remove-item 0 exePriceD2
set returnsD []
(foreach exePriceD exePriceD2 [set returnsD lput precision (ln ?1
- ln ?2) 4 returnsD])
file-open "C:\\Users\\Andrea\\Desktop\\prova2.txt"
foreach returnsD [file-print ?]
file-close

end
```

Il blocco denominato “save_resultsDay” serve per la generazione di un altro file di testo contenente la lista dei rendimenti “giornalieri” generati dalla simulazione, che verrà analizzato anch'esso con il software Maxima.

La struttura del blocco è analoga a quella del blocco “save_results”.

All'inizio del blocco è creata una lista temporanea denominata “exePrice2D”, strumentale alla generazione dei rendimenti.

La lista è creata identica alla lista “exePriceD”, che contiene tutti i prezzi di chiusura.

Successivamente alla lista “exePriceD” è rimosso l'ultimo elemento, che per costruzione è il prezzo 1000, mentre alla lista “exePrice2D” è rimosso il primo elemento, cioè l'ultimo prezzo di chiusura.

Le liste così modificate contengono la prima i prezzi di chiusura ad ogni tick, cioè ad ogni ciclo di simulazione, la seconda i prezzi del tick precedente.

La lista dei rendimenti è generata nuovamente sottraendo al logaritmo di ogni elemento della prima lista, il logaritmo dell'elemento corrispondente della seconda lista, seguendo la formula

$$r_T = \ln P_T - \ln P_{T-1} .$$

Il comando foreach aggiunge mano a mano elementi alla lista “returnsD”.

Il comando file-open apre un altro file, il comando file-print scrive il valore richiesto nel file aperto ed il comando file-close chiude il file aperto in precedenza, salvando le modifiche effettuate.

```
to set-price
```

```
if breed = IRRs [  
  ifelse random-float 1 < passLevel  
    [set pass True][set pass False]  
    ifelse not pass  
      [ifelse random-float 1 < 0.5 [set buy True set sell False]  
        [set sell True set buy False]]  
      [set buy False set sell False]  
  ifelse rat = 0 [if irrational = "last close"  
    [set price item 0 exePriceD + int (random-normal
```

```

0 s * a)

    if price < 1 [set price 1]]
if irrational = "last price"

[set price item 0 exePrice + int (random-normal

0 s * a)

    if price < 1 [set price 1]]
if irrational = "1000"

[set price int (random-normal 1000 s * a)

    if price < 1 [set price 1]]]
[if rational = "last close"

[set price item 0 exePriceD + int (random-normal

0 s * a)

    if price < 1 [set price 1]]
if rational = "last price"

[set price item 0 exePrice + int (random-normal

0 s * a)

    if price < 1 [set price 1]]
if rational = "1000"

[set price int (random-normal 1000 s * a)

    if price < 1 [set price 1]]] ]

end

```

Il blocco denominato “set-price” ha due funzioni: assegnare un valore alla variabile “price” e assegnare valori alle variabili “buy”, “sell” e “pass”.

Detto in altre parole, attraverso il blocco “set-price” gli agenti formulano le condizioni a cui intendono fare offerte, in acquisto o in vendita, con il relativo prezzo, che poi immetteranno nel mercato; oppure decidono se non immettere offerte durante quel particolare ciclo.

Le procedure, come si può notare dalla prima riga, vengono eseguite soltanto dagli agenti “IRRs”, ossia dal sottoinsieme di agenti irrazionali e di agenti razionali che non hanno posizioni aperte.

Gli agenti razionali con posizioni aperte, infatti, seguiranno altre strategie: in particolare strategie funzionali alla chiusura delle posizioni aperte con profitto o limitando le perdite.

Nelle due righe di codice successive, è utilizzato il comando ifelse. Questo comando, utilizzato spesso nella programmazione di questo modello, è molto simile al comando if, in quanto consente l'esecuzione di comandi al verificarsi delle condizioni specificate. Tuttavia a differenza di if, ifelse prevede l'esecuzione di comandi anche nel caso in cui le condizioni non si verificano.

Ciò consente la programmazione in due modi differenti nel caso di due eventi complementari.

Nello specifico viene assegnato valore True alla variabile “pass” con probabilità pari al valore della variabile “passLevel”, oppure valore False con probabilità pari a $1 - \text{“passLevel”}$.

Successivamente, nel caso in cui la variabile “pass” non assuma valore True, si assegna alla variabile “buy” o alla variabile “sell” valore True, e alla variabile “sell” o alla variabile “buy” valore False, in maniera equiprobabile.

Nel caso “pass” assuma valore True, entrambe le variabili “buy” e “sell” sono settate su False.

Si può quindi affermare che ogni agente avrà in ogni istante soltanto una di queste tre variabili impostata sul valore True, e questa ne determinerà il comportamento.

Infine il prezzo relativo all'ordine che ogni agente con breed IRRs intende immettere nel mercato può essere generato in tre modi differenti.

Il valore che l'utente attribuisce alle variabili “rational” e “irrational” determina il comportamento che ciascun tipo di agente adotterà nella formulazione di un valore da attribuire alla propria variabile “price”, sempre nel caso in cui occorra aprire una posizione o l'agente sia “irrazionale”, cioè sempre nel caso in cui l'agente in questione abbia breed IRR.

Poiché l'utente può attribuire tre valori alle variabili “rational” e “irrational”, cioè “last close”, “last price” e “1000”, ci sono tre modalità di generazione del valore da attribuire alla variabile “price”:

- il valore è generato come la realizzazione di una variabile pseudo-casuale normale⁸, di media pari all'ultimo prezzo di chiusura, e deviazione standard pari al valore della variabile “s”, impostabile a piacimento dell'utente nell'interfaccia, moltiplicato per il valore della variabile “a”, differente per ogni agente e indicatrice della propensione al rischio dell'agente,
- il valore è generato come la realizzazione di una variabile pseudo-casuale normale, di media pari all'ultimo prezzo eseguito, e deviazione standard pari al valore della variabile “s” moltiplicato per il valore della variabile “a”,
- il valore è generato come la realizzazione di una variabile pseudo-casuale normale, di media pari a 1000, e deviazione standard pari al valore della variabile “s” moltiplicato per il valore della variabile “a”.

Per comodità il prezzo è generato in modo che sia un valore intero, avendo considerato che questa restrizione trova comunque corrispondenza nella realtà, dove i prezzi sono un insieme discontinuo e discreto.

Inoltre è stata aggiunta la condizione che in qualsiasi caso il prezzo deve essere strettamente positivo: infatti se il valore che dovrebbe essere attribuito alla variabile “price” è un numero minore di uno, in automatico la variabile “price” assumerà valore 1, che è pertanto il minimo prezzo offribile sul mercato.

```
to set-sl-tp-price
```

```
  set in-mkt 1
```

```
  if breed = RATLs [if item 0 exePrice < sl and
```

```
    strategies = "stop-loss take-profit"
```

```
    [set price int sl - 1
```

⁸ Utilizzando una distribuzione di probabilità Normale, si avrà: $F(x) = \frac{1}{2} \left(1 + \operatorname{erf} \frac{x - \mu}{\sigma \sqrt{2}} \right)$ per $x \in \mathbb{R}$ e

$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$ per $x \in \mathbb{R}$, con $\sigma = s \cdot a$ e $\mu = (\text{item 0 exePriceD}), (\text{item 0 exePrice}), 1000$.

```

        if price < 1 [set price 1]]
if item 0 exePrice > tp
[set price int tp - 1
    if price < 1 [set price 1]]]
if breed = RATSS [if item 0 exePrice > sl and
    strategies = "stop-loss take-profit"
[set price int sl + 1
    if price < 1 [set price 1]]
if item 0 exePrice < tp
[set price int tp + 1
    if price < 1 [set price 1]]]

end

```

Il blocco “set-sl-tp-price” serve per assegnare un valore alla variabile “price” degli agenti che hanno posizioni aperte che si trovano in stop-loss o take-profit.

Innanzitutto si assegna valore 1 alla variabile “in-mkt”, in quanto una volta che l'agente ha eseguito le procedure contenute all'interno del blocco “set-sl-tp-price”, il medesimo agente eseguirà subito dopo le procedure contenute in “book”.

Ciò significa che quando un agente esegue il blocco “set-sl-tp-price”, sarà sicuro di immettere l'ordine nel mercato, e pertanto gli va attribuito valore 1 alla variabile “in-mkt”, per distinguerlo da quegli agenti che non hanno ancora operato.

Poiché gli agenti che possono eseguire questo blocco appartengono agli agentset “RATLs” e “RATSS”, le procedure sono differenti per i due tipi di breed.

Nel caso l'agente sia “RATL”, nel caso sia stato rotto il suo livello di stop-loss e nel caso la variabile “strategies” sia stata impostata dall'utente sul valore “stop-loss take-profit”, la sua

variabile “price” riceverà come valore il valore maggiore tra “sl”-1 e 1.

Stessa cosa avviene nel caso sia stato raggiunto il livello di take-profit: la variabile “price” riceverà valore “tp”-1.

Nel caso invece che l'agente sia “RATS”, nel caso sia stato rotto il suo livello di stop-loss e nel caso la variabile “strategies” sia stata impostata dall'utente sul valore “stop-loss take-profit”, la sua variabile “price” assumerà come valore “sl”+1.

Allo stesso modo, nel caso sia stato raggiunto il livello di take-profit, la variabile “price” assumerà il valore maggiore tra “tp”+1 e 1.

```
to change-breed-IRR
```

```
if rat = 1 [set in-mkt 0
            set breed IRRs set color grey
            set sl 0 set tp 0 set shape "person"
            let anew random-float 1 ifelse cash < 0
            [if a > anew [set a anew]]
            [if a < anew [set a anew]]]
```

```
end
```

La funzione del blocco “change-breed-IRR” è duplice: da un lato serve a riportare gli agenti con variabile “rat” uguale a 1 che hanno chiuso un'operazione, quindi agenti “RATLs” o “RATs”, allo stato iniziale, cioè alla breed “IRR”; dall'altro serve per cambiare il valore della variabile “a” di ogni agente “razionale” in base al proprio comportamento, “adattandola” o “calibrandola” nel

tempo.

Si chiede innanzi tutto all'agente di impostare la variabile “in-mkt” al valore 0, la variabile breed a IRR, la variabile color a grey, le variabili “sl” e “tp” al valore 0, nonché la variabile shape a “person”.

Successivamente si crea una variabile locale denominata “anew”, con valore pseudo-casuale compreso tra 0 e 1.

Se la variabile cash è positiva, se cioè l'agente ha operato ottenendo un profitto nel tempo, si ipotizza che potrebbe essere più tollerante verso il rischio.

Pertanto si confronta il valore della variabile “a” con quello della variabile “anew”, ed il nuovo valore di “a” sarà il maggiore tra i due.

Viceversa, se la variabile cash è negativa, se cioè l'agente ha operato ottenendo una perdita nel tempo, si ipotizza che potrebbe essere meno tollerante verso il rischio.

Pertanto si confronta il valore della variabile “a” con quello della variabile “anew”, ed il nuovo valore di “a” sarà il minore tra i due.

```
to change-breed-RATS
```

```
if rat = 1 [set breed RATSS
            set color green set shape "person"
            set buy True set sell False set pass False
            set sl (( item 0 exePrice ) * (1 + (a ^ 2)))
            set tp (( item 0 exePrice ) * (1 - (a ^ 2)))]
```

```
end
```

Nel caso un agente con variabile “rat” uguale a 1 abbia aperto una posizione corta, si attiva il blocco “change-breed-RATS”.

Si chiede innanzi tutto all'agente di impostare la variabile breed a RATS, la variabile color a green, la variabile shape a “person”.

Le variabili “buy”, “sell” e “pass” ricevono rispettivamente valori True, False e False.

Infine si impostano gli stop-loss e i take-profit.

Le regole di generazione di stop-loss e take profit sono le seguenti:

- $sl_{short} = p_t \cdot (1 + a^2)$,
- $tp_{short} = p_t \cdot (1 - a^2)$.

```
to change-breed-RATL
```

```
if rat = 1 [set breed RATLs
    set color red set shape "person"
    set buy False set sell True set pass False
    set tp (( item 0 exePrice ) * (1 + (a ^ 2)))
    set sl (( item 0 exePrice ) * (1 - (a ^ 2)))]
end
```

Quando un agente “razionale” apre una posizione lunga, si eseguono le procedue presenti nel blocco “change-breed-RATL”.

Si chiede innanzi tutto all'agente di impostare la variabile *breed* a RATL, la variabile *color* a red, la variabile *shape* a “person”.

Le variabili “buy”, “sell” e “pass” ricevono rispettivamente valori False, True e False.

Successivamente si impostano gli stop-loss e i take-profit.

Le regole di generazione di stop-loss e take profit sono le seguenti:

- $sl_{long} = p_t \cdot (1 - a^2)$,
- $tp_{long} = p_t \cdot (1 + a^2)$.

Capitolo 3

Strumenti software di analisi dati

3.1 Maxima

Maxima è un Computer Algebra System (CAS) in grado di eseguire calcoli numerici, simbolici, grafici e altre operazioni correlate.

Con Maxima è possibile trattare ad esempio espressioni simboliche e numeriche, derivate, integrali, espansioni in serie di Taylor, trasformate di Laplace, equazioni differenziali, sistemi di equazioni lineari, polinomi, insiemi, liste, vettori e matrici.

La precisione dei risultati numerici di Maxima è flessibile, a seconda delle esigenze dell'utente, ma può essere estremamente alta.

Si possono inoltre disegnare grafici sia in due che in tre dimensioni.

Maxima può funzionare con varie modalità:

1. è utilizzabile come terminale che interpreta comandi in modo interattivo;
2. è utilizzabile come un vero e proprio linguaggio di programmazione (interpretato, non compilato come il c);
3. ha un'interfaccia grafica semplice dal terminale, di nome xmaxima;
4. ha un'interfaccia grafica più evoluta, wxmaxima, che facilita l'inserimento dei comandi dal terminale anche agli utenti non esperti.

Il codice sorgente di Maxima può essere compilato in molti ambienti operativi, come Windows, Linux, e MacOS X.

Maxima discende da Macsyma, il Computer Algebra System sviluppato negli anni 60 al

Massachusetts Institute of Technology.

Macsyma era visto come una rivoluzione in quegli anni, e molti sistemi più recenti, come Maple e Mathematica, si sono ispirati ad esso.

Tuttavia Maxima è il solo sistema basato su Macsyma che sia offerto al pubblico gratuitamente e che abbia una comunità di utenti attiva per quello che riguarda il suo continuo sviluppo.

Dal 1982 al 2001 la sezione Maxima di Macsyma ha avuto come punto di riferimento William Schelter, che nel 1998 ottenne il permesso di rilasciare i codici sorgente sotto la licenza GNU General Public License (GPL).

Ci sono due modalità di funzionamento di Maxima: interattiva e di interpretazione di una lista di comandi (programma, o file batch).

Nella modalità interattiva si scrivono i comandi sulle righe che cominciano per (%i1), (%i2), ... o simili. Queste sono le righe di input, e sono numerate progressivamente. Alla fine di una riga di input, si preme il tasto "Invio" e Maxima interpreta i comandi producendo, se è il caso, una riga di output, indicata con (%o1), (%o2), ... numerata progressivamente.

Nella compilazione di file Maxima, occorre sempre terminare le righe di comando, cioè i comandi o le combinazioni di comandi, con un ";".

Se si omette il ";", il programma entra in condizione di errore.

Si può anche terminare un comando con "\$", che però non mostra l'eventuale output del comando.

3.2 Elaborazione dati Maxima

```
(%i173) load(numericalio)$
```

```
(%i174) a: read_list("C:/Users/Andrea/Desktop/prova.txt")$
```

```
(%i175) ad: read_list("C:/Users/Andrea/Desktop/prova2.txt")$
```

```
(%i176) alphamc: matrix([2,2,2,2,2,2,2],  
  [1.916,1.924,1.924,1.924,1.924,1.924,1.924],  
  [1.808,1.813,1.829,1.829,1.829,1.829,1.829],  
  [1.729,1.73,1.737,1.745,1.745,1.745,1.745],  
  [1.664,1.663,1.663,1.668,1.676,1.676,1.676],  
  [1.563,1.56,1.553,1.548,1.547,1.547,1.547],  
  [1.484,1.48,1.471,1.46,1.448,1.438,1.438],  
  [1.391,1.386,1.378,1.364,1.337,1.318,1.318],  
  [1.279,1.273,1.266,1.25,1.21,1.184,1.15],  
  [1.128,1.121,1.114,1.101,1.067,1.027,0.973],  
  [1.029,1.021,1.014,1.004,0.974,0.935,0.874],  
  [0.896,0.892,0.887,0.883,0.855,0.823,0.769],  
  [0.818,0.812,0.806,0.801,0.780,0.756,0.691],  
  [0.698,0.695,0.692,0.689,0.676,0.656,0.595],  
  [0.593,0.59,0.588,0.586,0.579,0.563,0.513])$
```

```
(%i177) vamc: [2.439,2.5,2.6,2.7,2.8,3,3.2,3.5,4,5,6,8,10,15,25]$  
  vbmc: [0.0,0.1,0.2,0.3,0.5,0.7,1]$
```

```
(%i179) load (descriptive)$
```

```
(%i180) m: mean (a);  
  var (a);
```

```

s: sqrt (var(a));
qr :qrange(a);

(%i184) m2: mean (ad);
var (ad);
s2: sqrt (var(ad));
qr2 :qrange(ad);

(%i188) s/sqrt(2),numer;
qr/2;

(%i190) q95: quantile(a, 0.95)$
q05: quantile(a, 0.05)$
q75: quantile(a, 0.75)$
q25: quantile(a, 0.25)$
q50: quantile(a, 0.50)$

(%i195) q95d: quantile(ad, 0.95)$
q05d: quantile(ad, 0.05)$
q75d: quantile(ad, 0.75)$
q25d: quantile(ad, 0.25)$
q50d: quantile(ad, 0.50)$

(%i200) valpha: (q95 - q05)/(q75 - q25);
vbeta: (q95 + q05 - 2*q50)/(q95 - q05);

(%i202) valpha2: (q95d - q05d)/(q75d - q25d);
vbeta2: (q95d + q05d - 2*q50d)/(q95d - q05d);

(%i204) vamc1: vamc$
vamc1: abs(vamc1 - valpha)$
lmin(vamc1)$

(%i207) stemp: 0$

```

```

    for i: 1 unless vamc1 [i] = lmin(vamc1) do stemp: i$
    stemp: stemp+1;

(%i210) vbmc1: vbmc$
    vbmc1: abs(vbmc1 - vbeta)$
    lmin(vbmc1)$

(%i213) rtemp: 0$
    for j: 1 unless vbmc1 [j] = lmin(vbmc1) do rtemp: j$
    rtemp: rtemp+1;

(%i216) alpha1: alphamc [stemp][rtemp];

(%i217) vamc2: vamc$
    vamc2: abs(vamc2 - valpha)$
    lmin(vamc2)$

(%i220) stemp: 0$
    for i: 1 unless vamc2 [i] = lmin(vamc2) do stemp: i$
    stemp: stemp+1$

(%i223) vbmc2: vbmc$
    vbmc2: abs(vbmc2 - vbeta)$
    lmin(vbmc2)$

(%i226) rtemp: 0$
    for j: 1 unless vbmc2 [j] = lmin(vbmc2) do rtemp: j$
    rtemp: rtemp+1$

(%i229) alpha2: alphamc [stemp][rtemp];

(%i230) b: continuous_freq (a, 30);

```

```
(%i231) bd: continuous_freq (ad, 30);
```

```
(%i232) c: first (b)$
```

```
    load(basic)$
```

```
    pop (c)$
```

```
(%i235) d: second (b)$
```

```
(%i236) g: reverse(c)$
```

```
(%i237) h: reverse(second(b))$
```

```
(%i238) fa1: [first(d) , second(d) , third(d) , fourth(d) , fifth  
    (d) , sixth(d) , seventh(d) , eighth(d) , ninth(d) ,  
    tenth(d)];
```

```
    fa2: [first(h) , second(h) , third(h) , fourth(h) , fifth  
    (h) , sixth(h) , seventh(h) , eighth(h) , ninth(h) ,  
    tenth(h)];
```

```
(%i240) fc1: [first(d), first(d) + second(d), first(d) + second(d)  
    + third(d), first(d) + second(d) + third(d) + fourth(d),  
    first(d) + second(d) + third(d) + fourth(d) + fifth (d),  
    first(d) + second(d) + third(d) + fourth(d) + fifth(d) +  
    sixth(d),  
    first(d) + second(d) + third(d) + fourth(d) + fifth (d) +  
    sixth(d) + seventh(d), first(d) + second(d) + third(d) +  
    fourth(d) + fifth (d)+ sixth(d) + seventh(d) + eighth(d),  
    first(d) + second(d) + third(d) + fourth(d) + fifth (d)+  
    sixth(d) + seventh(d) + eighth(d) + ninth(d), first(d) +  
    second(d) + third(d) + fourth(d) + fifth (d)+ sixth(d) +  
    seventh(d) + eighth(d) + ninth(d) + tenth(d)];
```

```
(%i241) fc2: [first(h), first(h) + second(h), first(h) + second(h)  
    + third(h), first(h) + second(h) + third(h) + fourth(h),
```

```

first(h) + second(h) + third(h) + fourth(h) + fifth (h),
first(h) + second(h) + third(h) + fourth(h) + fifth(h) +
sixth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h) +
sixth(h) + seventh(h), first(h) + second(h) + third(h) +
fourth(h) + fifth (h)+ sixth(h) + seventh(h) + eighth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h)+
sixth(h) + seventh(h) + eighth(h) + ninth(h),first(h) +
second(h) + third(h) + fourth(h) + fifth (h)+ sixth(h) +
seventh(h) + eighth(h) + ninth(h) + tenth(h)];

```

```
(%i242) n: length (a)$
```

```
(%i243) f1: create_list(i / n,i,fc1);
      f2: create_list(i / n,i,fc2);
```

```
(%i245) v: [first(c) , second(c) , third(c) , fourth(c) , fifth
(c) , sixth(c) , seventh(c) , eighth(c) , ninth(c) ,
tenth(c)];
      u1: abs(v);
```

```
(%i247) u2: [first(g) , second(g) , third(g) , fourth(g) , fifth
(g) , sixth(g) , seventh(g) , eighth(g) , ninth(g) ,
tenth(g)];
```

```
(%i248) c: first (bd)$
      pop (c)$
```

```
(%i250) d: second (bd)$
```

```
(%i251) g: reverse (c)$
```

```
(%i252) h: reverse(second(bd))$
```

```
(%i253) fa1d: [first(d) , second(d) , third(d) , fourth(d) , fifth
(d) , sixth(d) , seventh(d) , eighth(d) , ninth(d) ,
tenth(d)];
```

```
fa2d: [first(h) , second(h) , third(h) , fourth(h) , fifth
(h) , sixth(h) , seventh(h) , eighth(h) , ninth(h) ,
tenth(h)];
```

```
(%i255) fc1d: [first(d), first(d) + second(d), first(d) +
second(d) + third(d), first(d) + second(d) + third(d) +
fourth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d),
first(d) + second(d) + third(d) + fourth(d) + fifth(d) +
sixth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d) +
sixth(d) + seventh(d), first(d) + second(d) + third(d) +
fourth(d) + fifth (d)+ sixth(d) + seventh(d) + eighth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d)+
sixth(d) + seventh(d) + eighth(d) + ninth(d),first(d) +
second(d) + third(d) + fourth(d) + fifth (d)+ sixth(d) +
seventh(d) + eighth(d) + ninth(d) + tenth(d)];
```

```
(%i256) fc2d: [first(h), first(h) + second(h), first(h) +
second(h) + third(h), first(h) + second(h) + third(h) +
fourth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h),
first(h) + second(h) + third(h) + fourth(h) + fifth(h) +
sixth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h) +
sixth(h) + seventh(h), first(h) + second(h) + third(h) +
fourth(h) + fifth (h)+ sixth(h) + seventh(h) + eighth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h)+
sixth(h) + seventh(h) + eighth(h) + ninth(h),first(h) +
second(h) + third(h) + fourth(h) + fifth (h)+ sixth(h) +
seventh(h) + eighth(h) + ninth(h) + tenth(h)];
```

```

(%i257) nd: length (ad)$

(%i258) f1d: create_list(i / nd,i,fc1d);
      f2d: create_list(i / nd,i,fc2d);

(%i260) vd: [first(c) , second(c) , third(c) , fourth(c) , fifth
      (c) , sixth(c) , seventh(c) , eighth(c) , ninth(c) ,
      tenth(c)];
      u1d: abs(v);

(%i262) u2d: [first(g) , second(g) , third(g) , fourth(g) , fifth
      (g) , sixth(g) , seventh(g) , eighth(g) , ninth(g) ,
      tenth(g)];

(%i263) load(distrib)$

(%i264) n1: create_list(cdf_normal(i,m,s),i,v),numer$
      n2: create_list(1 - cdf_normal(i,m,s),i,u2)$

(%i266) n: append(n1,n2)$
      un: append(u1,u2)$

(%i268) cc1: create_list(cdf_cauchy(i,m,(qr / 2)),i,v)$
      cc2: create_list(1 - cdf_cauchy(i,m,(qr / 2)),i,u2)$

(%i270) cc: append(cc1,cc2)$

(%i271) plot2d([[discrete,u1 , f1],[discrete,u2 , f2],[discrete,un
      , n],[discrete,un , cc]], [style, points],[logy] , [logx]);

(%i272) plot2d([[discrete,u1 , f1],[discrete,u2 , f2],
      [discrete,u1d , f1d],[discrete,u2d , f2d]], [style, points],
      [logy] , [logx]);

```

```

(%i273) u11: create_list(log(i/(last(u1))),i,u1)$

(%i274) u22: create_list(log(i/(last(u1))),i,u2)$

(%i275) X11 : matrix (u11)$
      X22 : matrix (u22)$

(%i277) F11 : matrix (fa1)$
      F22 : matrix (fa2)$

(%i279) transpose (F11)$
      transpose (F22)$

(%i281) alpha1: 1 + ((last(fc1))/ (X11 . F11));

(%i282) alpha2: 1 + ((last(fc2))/ (X22 . F22));

(%i283) pl1:(u1 / last(u1))^(1 - alpha1);

(%i284) pl2:(u2 / last(u2))^(1 - alpha2);

(%i285) fp11: f1 / last(f1);
      fp12: f2 / last(f2);

(%i287) plot2d([[discrete,u1 , fp11],[discrete,u2 , fp12],
      [discrete,u1 , pl1],[discrete,u2 , pl2]], [style, points],
      [logy] , [logx]);

(%i288) ullog : create_list(log(x),x,u1)$

(%i289) u2log : create_list(log(x),x,u2)$

(%i290) fllog : create_list(log(x),x,f1)$

```

```

(%i291) f2log : create_list(log(x),x,f2)$

(%i310) unlog : create_list(log(x),x,u1)$

(%i293) ntemp: length(n1)$
      n1: delete(0.0,n1)$
      nlog : create_list(log(x),x,n1)$

(%i311) unlog1: unlog;
      for i: 1 thru (ntemp - length(n1)) do pop(unlog)$

(%i297) cclog : create_list(log(x),x,cc1)$

(%i313) M11 : matrix (ullog, f1log)$
      M12 : matrix (u2log, f2log)$
      M1n : matrix (unlog, nlog)$
      M1cc : matrix (unlog1, cclog)$

(%i317) transpose (M11)$
      transpose (M12)$
      transpose (M1n)$
      transpose (M1cc)$

(%i321) lsquares_estimates (M11, [y,x], y = A+B*x, [A,B]),numer;
      lsquares_estimates (M12, [y,x], y = A+B*x, [A,B]),numer;
      lsquares_estimates (M1n, [y,x], y = A+B*x, [A,B]),numer;
      lsquares_estimates (M1cc, [y,x], y = A+B*x, [A,B]),numer;

```

Per analizzare i dati di output del modello creato in ambiente NetLogo è stato utilizzato Maxima, programmando in modo tale che i dati siano raccolti ed elaborati al fine di ottenere valori e grafici esplicativi e commentabili.

Innanzitutto si chiede al software di caricare il pacchetto `numericalio`.

`Numericalio` è un insieme di funzioni per leggere e scrivere file e flussi di dati. Le funzioni per testi in input e in output sono in grado di leggere e scrivere numeri (interi o con la virgola), simboli e stringhe. Le funzioni per l'input e l'output binario sono in grado di leggere e scrivere solo numeri con la virgola.

Se esiste già un elenco, una matrice oggetto o una matrice per memorizzare dati di input, le funzioni di input di `numericalio` sono in grado di scrivere dati in quell'oggetto. Altrimenti, `numericalio` può intuire, in una certa misura, la struttura di un oggetto per memorizzare i dati, e restituire l'oggetto.

Con il comando `read_list` sono caricate le liste di rendimenti generati con NetLogo e contenute nei file di testo di output del modello ad agenti.

La lista dei rendimenti high-frequency è definita all'interno del codice come "a", mentre la lista dei rendimenti giornalieri è definita come "ad".

In seguito è generata una matrice di valori, "alphamc", che fanno riferimento alla tabella per il calcolo della stima del valore α di una distribuzione stabile.

Il metodo utilizzato è quello proposto da McCulloch, e prevede l'interpolazione in una tabella di due valori stimati [McCulloch, 1986].ⁱ

Le due liste successive, "vamc" e "vbmc", contengono i valori da interpolare.

Successivamente è caricato il pacchetto `descriptive`.

Il pacchetto `descriptive` contiene una serie di funzioni per fare calcoli statistici descrittivi e grafici.

Con "m" è definita la variabile che rappresenta la media di "a", calcolata tramite il comando `mean`.ⁱⁱ

Con "s" invece è definita la deviazione standard di "a", calcolata come la radice quadrata della varianza di "a".ⁱⁱⁱ

"Qr" definisce invece la variabile range interquartile di "a", calcolata tramite il comando `qr`.^{iv}

Subito sotto è effettuato un confronto tra due valori relativi alla distribuzione: il primo è il valore $s/\sqrt{2}$, mentre il secondo è $qr/2$.

Nel caso di distribuzioni stabili, uno dei parametri della funzione caratteristica è γ , che è il parametro di "scala", nel senso che determina la grandezza delle probabilità complessive.

Se U è una variabile casuale "stabile" standardizzata, cU sarà una variabile casuale "stabile" con gli stessi valori α , β e δ di U, e con $\gamma = c^\alpha$.

Nel caso di una distribuzione Normale con $\alpha = 2$, γ vale c^2 , mentre nel caso di una distribuzione di Cauchy con $\alpha = 1$, γ vale $2c$. La teoria ci dice anche che per la prima γ vale $s^2/2$, mentre per la seconda vale γ vale $qr/2$. Con una semplice sostituzione otteniamo che:

- $c = qr/2$ nel caso di $\alpha = 1$,
- $c = s/\sqrt{2}$ nel caso di $\alpha = 2$.

A questo punto sono calcolati alcuni percentili delle due distribuzioni di rendimenti, che saranno strumentali alla stima del valore α attraverso il metodo di McCulloch.

I percentili stimati sono il novantacinquesimo, il quinto, il settantacinquesimo, il venticinquesimo, e il cinquantesimo, cioè la mediana.

I due valori, “ v_{α} ” e “ v_{β} ”, che serviranno all'interpolazione sono calcolati infatti come funzione dei suddetti percentili:

$$v_{\alpha} = \frac{P_{95} - P_{05}}{P_{75} - P_{25}} \quad \text{e} \quad v_{\beta} = \frac{P_{95} + P_{05} - 2P_{50}}{P_{95} - P_{05}} .$$

Il passo successivo riguarda i comandi che eseguono l'interpolazione richiesta.

Innanzitutto si genera una lista dove sono presenti gli scarti tra il valore trovato v_{α} e i valori indice della tabella “ $v_{\alpha mc}$ ”, in valore assoluto, e se ne individua il minore. Questo significa andare a trovare il valore della lista “ $v_{\alpha mc}$ ” che più si avvicina al nostro valore “ v_{α} ”.

Poi, attraverso una procedura ricorsiva, richiamata in Maxima tramite il comando `do`, si individua la posizione del numero indice che più si avvicina a “ v_{α} ”.

Il numero di righe della tabella proposta da McCulloch e riportata nel codice è 15. Pertanto il numero indice della riga, denominato nel codice “`stemp`”, assumerà valore compreso tra 1 e 15.

Lo stesso procedimento è eseguito per il parametro v_{β} .

Il numero di colonne della tabella proposta da McCulloch è 10, pertanto il numero indice di colonna, denominato nel codice “`rtemp`”, assumerà valore compreso tra 1 e 10.

A questo punto l' α stimato non è nient'altro che l'elemento della matrice “ $\alpha_{\alpha mc}$ ” con riga “`stemp`” e colonna “`rtemp`”.

La medesima procedura è eseguita per il calcolo della stima del valore α sia nel caso di rendimenti high-frequency, sia nel caso di rendimenti giornalieri.

Ne l caso i valori “ α_1 ” e “ α_2 ” siano vicini o addirittura uguali abbiamo evidenza del fatto

che la distribuzione è invariante di scala.

Il comando `continuous_freq` raggruppa i dati di una lista in classi, in numero variabile a scelta dell'utente (nel nostro caso 30) e genera una lista contenente due sottoliste: nella prima sono contenuti i valori degli estremi degli intervalli (il primo valore è il primo estremo inferiore, i successivi sono estremi inferiori\superiori, l'ultimo valore è l'ultimo estremo superiore), nella seconda le corrispondenti frequenze assolute.

La lista così generata è definita “b” nel caso di rendimenti high-frequency e “bd” nel caso di rendimenti giornalieri, la prima sottolista “c” mentre la seconda “d”.

Poiché la prima sottolista contiene 31 valori anziché 30, dovuto al fatto che il primo elemento è l'estremo inferiore, si procede all'eliminazione del suddetto elemento tramite il comando `pop`, che restituisce la lista di partenza senza il primo elemento.

Il comando `pop` è utilizzabile richiamando il pacchetto di `Maxima basis`.

Le liste “g” e “h” contengono gli stessi elementi delle liste “c” e “d”, però in ordine inverso.

Le frequenze assolute delle code della distribuzione campionaria, cioè delle dieci classi superiori e delle dieci classi inferiori della distribuzione, sono contenute nelle liste “fa1” e “fa2”.

Le frequenze assolute cumulate delle code della distribuzione sono generate nelle liste “f1” e “f2”.

Ogni lista contiene dieci elementi, e le liste sono create nel seguente modo: il primo elemento della lista “f1” è il primo elemento della lista “d”, il secondo elemento di “f1” è la somma dei primi due elementi di “d”, il terzo elemento di “f1” è la somma dei primi tre elementi di “d”, e così via.

Per la lista “f2” il procedimento è il medesimo seguito per creare “f1”, ma facendo riferimento agli elementi di “h” invece che agli elementi di “d”.

Il passo successivo consiste nel cambiare le frequenze contenute in “f1” e “f2” da assolute a relative, e per fare ciò si dividono gli elementi delle due liste per il numero di osservazioni presenti nel campione, cioè si divide per “n”, che è il nome assegnato al valore della lunghezza della lista “a”, che contiene tutte le osservazioni.

Per analizzare le code della distribuzione, oltre alle liste “f1” e “f2” contenenti i valori delle frequenze relative corrispondenti ad ogni classe, abbiamo bisogno delle liste contenenti i valori delle classi.

Queste due liste sono denominate “u1” e “u2” e, come le liste “f1” e “f2” a cui sono associate, contengono dieci elementi ciascuna.

I valori relativi alle classi della coda destra della distribuzione, sono tutti valori positivi, mentre quelli relativi alle classi della coda sinistra sono tutti negativi.

Poiché l'intento è un confronto grafico diretto e immediato dei valori relativi alle due code, i valori delle classi relativi alla coda sinistra della distribuzione sono presi in valore assoluto, come si può notare nel codice. La lista "v" dunque è una lista puramente strumentale all'operazione, e ci permette di costruire la lista "u1".

I valori inseriti nelle liste "u1" e "u2" saranno quindi tutti positivi, e vengono presi rispettivamente dalla lista "v" (che a sua volta è stata creata prendendo gli elementi dalla lista "c") e dalla lista "g".

Tutta la procedura è ripetuta per i rendimenti giornalieri, e le liste contenenti le frequenze e i valori campionari associati hanno gli stessi nomi del caso precedente, con una "d" postposta al fine di renderli distinguibili.

A questo punto viene chiesto di caricare il pacchetto `distrib`, che contiene un insieme di funzioni utilizzabili per il calcolo delle probabilità in modelli univariati. Le distribuzioni di probabilità possono essere sia continue che discrete.

Il pacchetto `distrib` utilizza una convenzione di denominazione. Il nome di ogni funzione ha due parti, la prima fa riferimento alla funzione o al parametro che vogliamo calcolare, mentre la seconda è un riferimento esplicito al modello probabilistico che intendiamo usare.

Il passi successivi sono mossi dall'intento di confrontare il comportamento dei dati che noi osserviamo, cioè il comportamento delle code della distribuzione del campione dei rendimenti, con il comportamento che la teoria suggerisce a proposito di questa distribuzione.

Poiché la teoria classica ipotizza che la distribuzione dei rendimenti in un mercato efficiente sia normale, creiamo i valori relativi ad una distribuzione normale che abbia caratteristiche compatibili con ciò che noi abbiamo osservato nei dati (cioè prendiamo in considerazione una distribuzione normale con parametri media e varianza pari ai valori campionari che abbiamo precedentemente stimato).

Gli elementi delle liste denominate "n1" e "n2" sono appunto i valori teorici di probabilità di una distribuzione normale con media "m" e varianza "s", in corrispondenza dei valori degli estremi delle classi.

Le liste "n1" e "n2" sono create infatti utilizzando il comando `create_list`, che crea una nuova lista partendo da una precedentemente definita. Gli elementi della nuova lista saranno gli elementi della vecchia lista valutati attraverso una particolare funzione inserita all'interno del comando.

Nel nostro caso ad esempio la lista “n1” contiene gli elementi della lista “v”, valutati dalla funzione di distribuzione di una variabile casuale normale. La lista “n1” avrà quindi lo stesso numero di elementi della lista di partenza “v”.

La funzione `cdf_normal(x, m, s)` del pacchetto `distrib` fornisce il valore numerico della funzione di distribuzione di una variabile aleatoria normale di media m e varianza s , valutata in x .^v

I due comandi successivi creano le liste “n” e “un” utilizzando il comando `append`.

Il comando di Maxima `append` genera una lista partendo dagli elementi di due o più liste, inserendo nella nuova lista prima tutti i singoli elementi della prima lista, poi tutti quelli della seconda lista, e così via.

Per avere un ulteriore elemento di paragone, il procedimento di creazione dei valori teorici è stato fatto analogamente anche nel caso si ipotizzi che la distribuzione dei rendimenti segua una distribuzione di Cauchy.

Gli elementi delle liste denominate “cc1” e “cc2” sono i valori teorici di probabilità di una distribuzione di Cauchy con parametri “m” e “qr/2”, in corrispondenza dei valori degli estremi delle classi.

La funzione `cdf_cauchy(x, a, b)` del pacchetto `distrib` fornisce il valore numerico della funzione di distribuzione di una variabile aleatoria normale di parametri a e b , valutata in x .^{vi}

A questo punto è stato generato tutto ciò che è necessario per eseguire un confronto grafico.

Il confronto grafico è proposto attraverso la costruzione di un grafico cartesiano a due dimensioni, a doppia scala logaritmica.

Il primo confronto grafico rappresenta le due code della distribuzione campionaria dei rendimenti high-frequency, e le due serie dei loro valori teorici nel caso di distribuzione normale e di Cauchy.

Il secondo confronto grafico invece mostra congiuntamente le due code della distribuzione dei rendimenti high-frequency e le due code della distribuzione dei rendimenti giornalieri.

Il comando `plot2d` mostra uno o più grafici in due dimensioni.

Il primo input da inserire nel comando è cosa disegnare: nel nostro caso inseriamo la coppia di liste “u1” e “f1”, la coppia di liste “u2” e “f2”, la coppia di liste “un” e “n” e la coppia di liste “un” e “cc”.

Un grafico può essere definito anche nella forma discreta. La forma discreta è usata per disegnare un insieme di punti di coordinate date. Un grafico discreto è definito da una lista che inizia con la

keyword “discrete”, seguita da una o due liste di valori. Nel caso siano specificate due liste, devono essere della medesima lunghezza, e la prima sarà interpretata come l'insieme delle coordinate x dei punti che si vogliono rappresentare, mentre la seconda come l'insieme delle coordinate y.

L'opzione [style, points] è utilizzata poiché vogliamo che i punti siano rappresentati separatamente, senza alcuna linea di congiungimento.

Le opzioni [logy] e [logx] sono utilizzate per generare il grafico in doppia scala logaritmica.

La rappresentazione grafica dei punti è interessante in quanto possiamo confrontare immediatamente quanto i punti che rappresentano le code della distribuzione del campione di rendimenti si discostino dai punti che rappresentano il valore teorico ipotizzando una distribuzione normale.

La teoria suggerisce che, nel caso la distribuzione estratta da un campione che si ipotizza stabile non sia Normale, da un certo valore in poi le code della distribuzione seguono una qualche forma di power law.

Seguendo questo principio, proviamo a individuare il valore dello stimatore di massima verosimiglianza per il parametro α di una power law [Hall, 1982][Newman, 2004][Mittnik e Paolella, 1999].^{vii}

La formula utilizzata per il calcolo è:

$$\hat{\alpha}_{ML} = 1 + n \left[\sum_{i=1}^n \ln \left(\frac{x_i}{x_{min}} \right) \right]^{-1} .$$

In seguito vengono generati i valori teorici di probabilità cumulata riferibili a due power law di parametro α stimato (da notare è che è stimato un α diverso per ciascuna delle due code, e che l' α stimato dipende dalla scelta del parametro x_{min}).

I valori appena generati vengono quindi nuovamente confrontati con i valori osservati, ancora una volta in un grafico a doppia scala logaritmica.

Ritornando al primo grafico generato, e considerando quattro rette che interpolano i quattro insiemi di punti (coda sinistra campionaria, coda destra campionaria, valori teorici della distribuzione normale e valori teorici della distribuzione di Cauchy), un parametro che possiamo confrontare è la pendenza delle quattro rette.

Il parametro in questione è stimato ricorrendo a una regressione lineare.

Poiché il grafico è in doppia scala logaritmica, il parametro andrà stimato utilizzando il logaritmo

dei dati di partenza.

Sono create quindi sette liste, “u1log”, “u2log”, “f1log”, “f2log”, “unlog”, “nlog” e “cclog”, a partire dalle liste “u1”, “u2”, “f1”, “f2”, “un”, “n1” e “cc1”. Il procedimento utilizzato prevede anche questa volta l'uso del comando `create_list`.

Successivamente sono generate quattro matrici, “M11”, “M12”, “M1n” e “M1cc”, attraverso il comando `matrix`.

Il comando di Maxima `matrix` è in grado di generare matrici rettangolari partendo da N liste.

Ogni lista diventa una riga della matrice così generata. Le liste di partenza devono essere tutte della stessa lunghezza M: la matrice generata infatti avrà dimensioni NxM.

Nel nostro caso le matrici sono generate a partire dalle coppie di liste “u1log” e “u2log”, “f1log” e “f2log”, “unlog” e “nlog”, “unlog1” e “cclog”.

Alcune righe di comando sono state inserite per evitare errori nel calcolo. Per valori “estremi” di x una variabile casuale Normale ha probabilità talmente piccola che il software la assimila a 0. Poiché sono calcolati i logaritmi di questi valori, vi sarebbe stato un errore in quanto sarebbe stato richiesto il calcolo di $\log(0)$.

Al fine di evitare ciò sono eliminati dalla lista “nlog” tutti gli elementi per cui sarebbe stato calcolato $\log(0)$, e dalla lista “unlog” sono eliminati gli elementi loro corrispettivi.

Le matrici sono poi trasposte, utilizzando il comando `transpose`.

A questo punto si esegue la regressione lineare su ognuna delle quattro matrici, utilizzando il metodo dei minimi quadrati.^{viii}

Capitolo 4

Analisi dati

Facendo riferimento al quadro teorico su cui si basa il lavoro svolto, è possibile proporre, fra gli altri, tre spunti di analisi:

1. osservare se la serie di rendimenti è stazionaria,
2. osservare se la serie di rendimenti è stabile,
3. stimare il valore dell' α della distribuzione teorica a cui facciamo riferimento.

4.1 Test sull'apparato statistico utilizzato

Prima di incominciare a fare inferenza sui dati generati dalle simulazioni, si è preferito testare l'apparato statistico e di calcolo programmato e utilizzato.

Per fare ciò sono state generate alcune serie di dati ad hoc da analizzare, in modo da osservare i risultati che si ottengono e verificare che siano corretti.

Le serie di dati sono generate come realizzazioni di variabili casuali che hanno distribuzioni stabili simmetriche standard, cioè come realizzazioni di variabili aleatorie

$$X \sim S(\alpha, \beta, \gamma, \delta) = S(\alpha, 0, 1, 0) .$$

Per generare questi dati è stato utilizzato il metodo di simulazione proposto da Chambers [Chambers et altri, 1976].

Date due variabili casuali indipendenti, U e V , con U uniformemente distribuita nell'intervallo $(-\pi/2, \pi/2)$ e V esponenzialmente distribuita con media 1, la variabile casuale simmetrica Z , con

$$Z = \frac{\sin(\alpha U)}{(\cos U)^{1/\alpha}} \left[\frac{\cos((\alpha-1)U)}{V} \right]^{(1-\alpha)/\alpha}$$

ha distribuzione $Z \sim S(\alpha, 0, 1, 0)$, con $0 < \alpha \leq 2$ e $\alpha \neq 1$.

α	X_1	X_2	$X_1 + X_2$	σ_1	σ_2	σ_{12}
2,0	1,98	1,98	1,96	0,035	0,035	0,043
1,9	1,93	1,9	1,89	0,058	0,072	0,085
1,8	1,81	1,8	1,79	0,044	0,025	0,056
1,7	1,71	1,72	1,69	0,047	0,040	0,033
1,6	1,63	1,6	1,59	0,049	0,063	0,049
1,5	1,49	1,52	1,5	0,048	0,042	0,053
1,4	1,42	1,39	1,41	0,046	0,050	0,063
1,3	1,29	1,29	1,28	0,035	0,036	0,000
1,2	1,22	1,22	1,23	0,079	0,079	0,073
1,1	1,1	1,12	1,13	0,048	0,030	0,003

Tabella 1: Questa tabella illustra il valore medio dell' α stimato di 10 serie di numerosità 1045. Nelle prime due colonne è indicato il valore dell' α inserito nell'algoritmo di generazione della serie. Nella terza il valore dell' α della serie generata dalla somma di due variabili aleatorie stabili, con il medesimo α . Le rimanenti tre colonne indicano la deviazione standard delle 10 prove dal valore medio. Fonte dati: elaborazione personale.

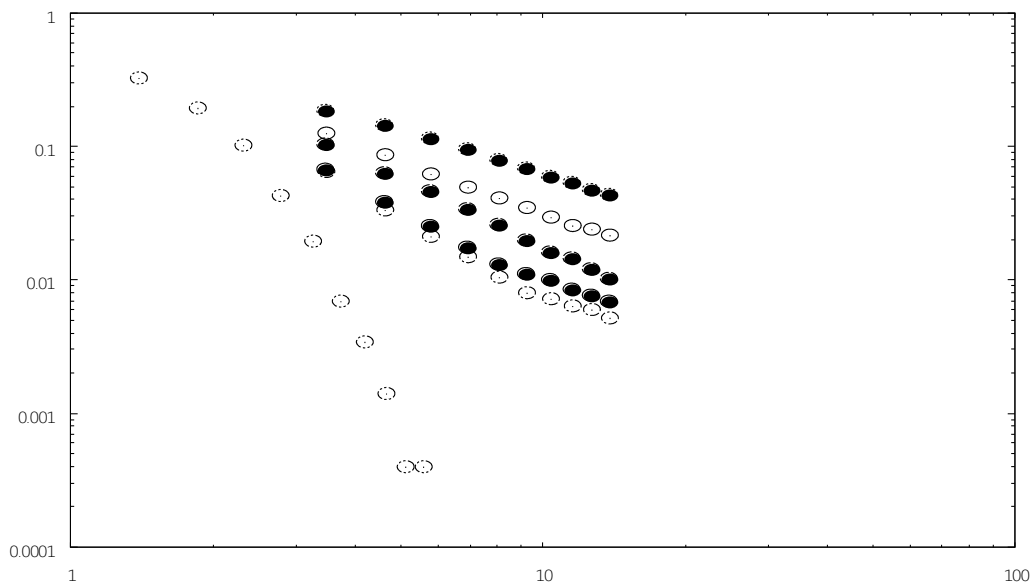


Illustrazione 1: Rappresentazione grafica in doppia scala logaritmica delle code delle distribuzioni di serie di dati stabili generate per vari livelli di α . Le serie sono generate come insieme di numeri pseudocasuali provenienti da variabili aleatorie stabili standard con parametri $\beta = 0$, $\gamma = 1$, $\delta = 0$, $\alpha = 2, 1,8, 1,6, 1,4, 1,2$ e 1 . Fonte dati: elaborazione personale.

4.2 Analisi dati reali

L'analisi su dati reali è utile per identificare evidenze di comportamenti che rientrano nell'ambito teorico proposto.

In particolare si cerca di osservare se si possa ipotizzare che i rendimenti di un bene scambiato sul mercato siano assimilabili alle realizzazioni di una variabile aleatoria con distribuzione di probabilità stabile.

Nel caso qui proposto, si analizza una serie di rendimenti di un titolo azionario, il titolo JPMorgan, quotato presso il New York Stock Exchange.

L'idea della modellizzazione di una serie osservata basandosi sull'assunto che essa sia l'insieme delle realizzazioni di una variabile aleatoria è molto utile nella pratica, ma necessita che la serie osservata sia stazionaria.

Mandelbrot [Mandelbrot, 1963] utilizza un metodo di comparazione grafica per individuare sintomi di stazionarietà delle serie utilizzate.

Applicando quanto proposto al nostro caso, abbiamo inizialmente diviso la serie dei rendimenti giornalieri del titolo JPMorgan in due parti: la prima parte va all'incirca dal 2001 al 2005, mentre la seconda parte va dal 2006 al 2010.

Successivamente abbiamo provveduto ad effettuare un'analisi comparata tra le due sottoserie così generate.

Nel caso l'intera serie sia stazionaria, le due sottoserie dovrebbero presentare comportamenti molto simili, in quanto stazionarietà significa che le caratteristiche della funzione che genera la serie rimangono immutate nel tempo.

Dall'analisi svolta è emerso che l'ipotesi di stazionarietà non può essere rifiutata, in quanto sia l' α stimato per le due sottoserie, sia il comportamento delle code, sono molto simili.

	α	v_α	v_β
2001-2005	1,273	4,039	0,053
2006-2010	1,391	3,548	-0,056

Tabella 2: La tabella indica i valori stimati dell' α nel caso di rendimenti giornalieri, dal 2001 al 2005 per la prima serie e dal 2006 al 2010 per la seconda serie, e i valori stimati di v_α e v_β utilizzati per ricavare la stima di α . Fonte dati: Yahoo!Finance.

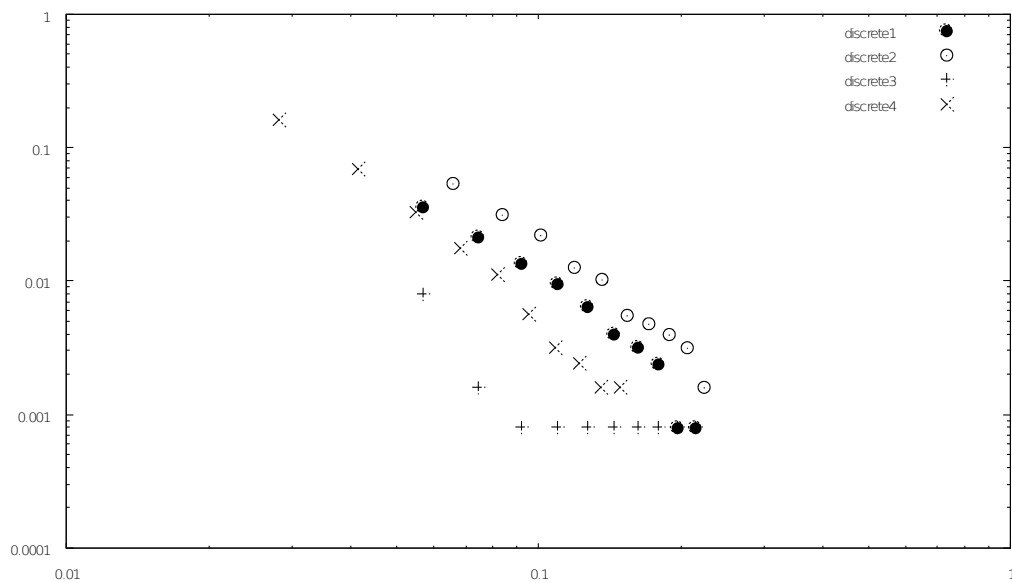


Illustrazione 2: Rappresentazione grafica in doppia scala logaritmica delle code delle distribuzioni dei rendimenti giornalieri del titolo JPMorgan dal 2001 al 2005 e dei rendimenti giornalieri dal 2006 al 2010. Si noti come le due serie di rendimenti abbiano la medesima pendenza e valori molto simili. Un comportamento di questo genere può portare a ipotizzare che la serie nella sua totalità sia stazionaria nel tempo. Fonte dati: Yahoo!Finance.

Per verificare che la serie sia stabile si possono cercare prove di invarianza alla somma.

Nel caso in analisi questo è possibile farlo sia confrontando i valori stimati dell' α , sia mediante un confronto grafico delle code della distribuzione campionaria.

	α	v_α	v_β
1m	1,664	2,869	-0,003
day	1,279	3,792	-0,009
week	1,279	3,790	-0,047
month	1,484	3,142	-0,013

Tabella 3: La tabella indica i valori stimati dell' α nel caso di rendimenti a un minuto, giornalieri, settimanali e mensili, e i valori stimati di v_α e v_β utilizzati per ricavare la stima di α . Fonte dati: Yahoo!Finance.

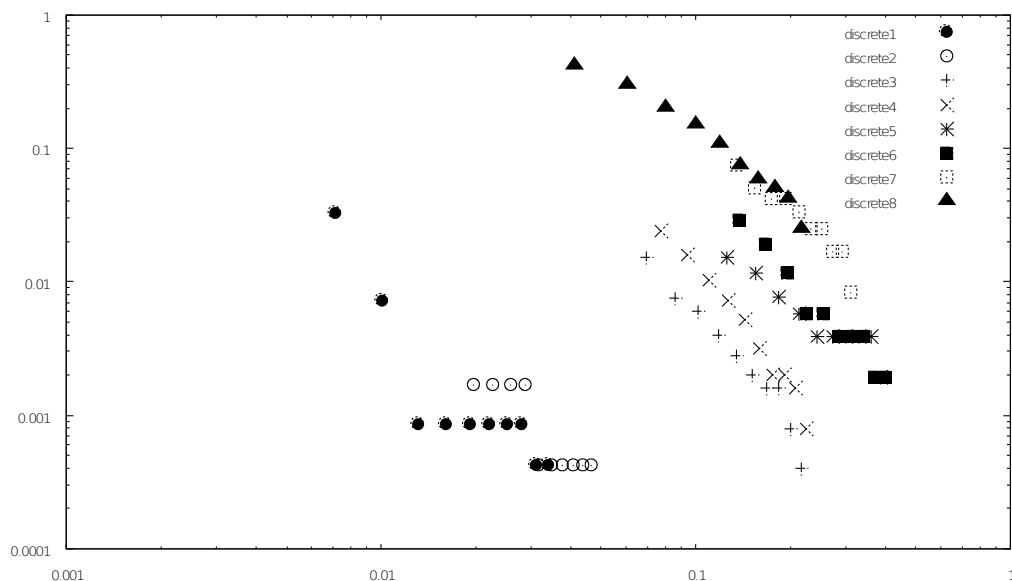


Illustrazione 3: Rappresentazione grafica in doppia scala logaritmica delle code delle distribuzioni dei rendimenti minuto per minuto del titolo JPMorgan dal 2/3/2009 al 10/3/2009 e dei rendimenti giornalieri, settimanali e mensili del titolo JPMorgan dal 1/1/2001 al 31/12/2010. Si può osservare come le tre serie di rendimenti giornalieri, settimanali e mensili siano traslazioni nel piano l'una dell'altra, mantenendo però invariata la pendenza della distribuzione: questo evidenzia il fatto che le tre serie si possono ipotizzare stabili, in quanto generate dalla stessa distribuzione, a meno di un fattore di scala. Fonte dati: hftradingbook.com e Yahoo!Finance.

4.3 Analisi dati simulati

Il modello ABM di mercato finanziario proposto genera serie di rendimenti che possono essere analizzate.

Il suo funzionamento consiste nel far combaciare ordini di acquisto e di vendita immessi nel mercato da agenti.

Questi agenti possono formulare i propri ordini in diversi modi.

Avendo costruito il modello con due tipologie di agenti, da una parte abbiamo agenti “irrazionali” che immettono ordini senza basarsi su quanto hanno compiuto in passato, dall'altra abbiamo agenti “razionali” che immettono ordini sul mercato cercando di ottenere un profitto e/o limitare le perdite, in funzione di ciò che hanno fatto in passato.

Il modello di simulazione può tuttavia prescindere totalmente dalla parte sul comportamento degli agenti, in quanto se la variabile “strategies” è impostata su “random”, gli agenti saranno tutti “irrazionali”.

Per “irrazionali” intendiamo agenti che immettono ordini sul mercato senza osservare le posizioni aperte e senza modificare il proprio comportamento nel tempo, senza cioè nessuna forma di apprendimento né qualche regola prefissata.

Si può tuttavia dividere la popolazione di agenti in due gruppi: il primo può all'interno della simulazione formulare il prezzo della propria offerta basandosi sull'ultimo prezzo eseguito, mentre il secondo può formulare la propria offerta facendo riferimento all'ultimo prezzo di chiusura, cioè al prezzo di chiusura del ciclo precedente.

In formule:

$$1. \quad P_t \sim N(P_{t-1}^e, sa) \quad \text{o}$$

$$2. \quad P_t \sim N(P_{t-1}^c, sa) \quad ,$$

dove:

P_t è il prezzo che ogni agente offre all'istante t , P_{t-1}^e è l'ultimo prezzo eseguito e P_{t-1}^c è l'ultimo prezzo di chiusura.

Riassumendo, è possibile svolgere simulazioni con agenti “irrazionali” che:

- immettono tutti ordini generati dall'ultimo prezzo eseguito,
- immettono tutti ordini generati dall'ultimo prezzo di chiusura,

- alcuni immettono ordini generati dall'ultimo prezzo eseguito, mentre altri immettono ordini generati dall'ultimo prezzo di chiusura.

Le percentuali di agenti di un tipo piuttosto che di un altro sono regolate attraverso le variabili “p-rat”, “irrational” e “rational”.

Come evidenzia la tabella 7, la simulazione effettuata con agenti “razionali” che seguono una strategia “take-profit only” ha generato i dati che si prestano meglio ad essere modellizzati attraverso una distribuzione stabile.

Confrontando il valore dell' α delle due serie, tick-by-tick e giornaliera, sembrano esserci evidenze di invarianza alla somma.

Inoltre le deviazioni standard sono le più piccole registrate.

	α	σ
tick-by-tick	1,428	0,046
day	1,602	0,141

Tabella 4: La tabella indica i valori medi stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri, e i valori stimati della deviazione standard. Il numero di prove effettuate è 20. Ciascuna prova consiste in una simulazione della durata di 350 cicli. I “seed” utilizzati sono: 100, 13, 2, 1986, 1, -2, 99, 522, 10, 11, 12, -25, -30, -444, -10000, -155, 99999, -3334, -337, 337. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo di chiusura.

	α	σ
tick-by-tick	1,307	0,050
day	1,752	0,129

Tabella 5: La tabella indica i valori medi stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri, e i valori stimati della deviazione standard. Il numero di prove effettuate è 20. Ciascuna prova consiste in una simulazione della durata di 350 cicli. I “seed” utilizzati sono: 100, 13, 2, 1986, 1, -2, 99, 522, 10, 11, 12, -25, -30, -444, -10000, -155, 99999, -3334, -337, 337. La metà degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, l'altra metà seguendo l'ultimo prezzo di chiusura.

	α	σ
tick-by-tick	1,522	0,059
day	1,602	0,146

Tabella 6: La tabella indica i valori medi stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri, e i valori stimati della deviazione standard. Il numero di prove effettuate è 10. Ciascuna prova consiste in una simulazione della durata di 250 cicli. I “seed” utilizzati sono:

11, 10, 23, -991, -1, 675487, -567, -5, -100089, 9999. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito.

	α	σ
tick-by-tick	1,489	0,063
day	1,544	0,096

Tabella 7: La tabella indica i valori medi stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri, e i valori stimati della deviazione standard. Il numero di prove effettuate è 10. Ciascuna prova consiste in una simulazione della durata di 250 cicli. I "seed" utilizzati sono: 1, 10, 23, -991, -1, 675487, -567, -5, -100089, 9999. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, gli agenti "razionali" seguono una strategia di chiusura delle posizioni aperte del tipo "take-profit only".

	α	σ
tick-by-tick	1,531	0,085
day	1,721	0,136

Tabella 8: La tabella indica i valori medi stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri, e i valori stimati della deviazione standard. Il numero di prove effettuate è 10. Ciascuna prova consiste in una simulazione della durata di 250 cicli. I "seed" utilizzati sono: 1, 10, 23, -991, -1, 675487, -567, -5, -100089, 9999. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, gli agenti "razionali" seguono una strategia di chiusura delle posizioni aperte del tipo "stop-loss take-profit".

4.3.1 Dimensione

Il modello di simulazione può generare un campione di dati di dimensione variabile, a piacere dell'utente.

È interessante analizzare il comportamento dei rendimenti generati a diversi istanti della simulazione, verificando come varia il parametro α di interesse, anche a seconda delle condizioni impostate all'inizio della simulazione.

Dai risultati empirici ottenuti si evince che si manifesta una convergenza verso un determinato livello di α con l'aumentare della dimensione del campione, cioè con l'aumentare dei cicli di simulazione realizzati.

Questo fenomeno è coerente anche con quanto si osserva nel confronto tra rendimenti tick-by-tick, giornalieri, settimanali e mensili.

Provenendo i dati dal medesimo campione, ed essendo i dati che compongono le serie le somme gli uni degli altri, aumentando l'orizzonte temporale dei rendimenti si riduce la numerosità di dati che si analizzano.

Da notarsi che nel caso di dati generati dal modello di simulazione, la dimensione del campione di rendimenti tick-by-tick generati non è costante mantenendo costante il numero di cicli eseguiti, in quanto il numero di contratti eseguiti in ogni ciclo è casuale ed in più dipende dalle condizioni iniziali del modello.

Si può aggiungere inoltre che, nel caso gli agenti “razionali” agiscano seguendo la strategia “take-profit only”, si generano serie di dati con α decisamente più piccolo rispetto alle altre due strategie, cioè con rendimenti molto grandi più frequenti.

n. tick	α	α
	tick-by-tick	day
300	1,56	1,48
500	1,48	1,56
1000	1,39	1,66
1500	1,39	1,66
2000	1,39	1,66

Tabella 9: La tabella indica i valori stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri. Ciascuna prova consiste in una simulazione della durata di 300, 500, 1000, 1500, 2000 cicli. Il “seed” utilizzato è 9999. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, gli agenti “razionali” seguono una strategia di chiusura delle posizioni aperte del tipo “stop-loss take-profit”.

n. tick	α	α
	tick-by-tick	day
300	1,48	1,48
500	1,39	1,56
1000	1,39	1,56
1500	1,48	1,66
2000	1,39	1,66

Tabella 10: La tabella indica i valori stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri. Ciascuna prova consiste in una simulazione della durata di 300, 500, 1000, 1500, 2000 cicli. Il “seed” utilizzato è 9999. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito ed una strategia “random”.

n. tick	α	α
	tick-by-tick	day
300	1,56	1,66
500	1,56	1,66
1000	1,48	1,56
1500	1,28	1,48
2000	1,13	1,48

Tabella 11: La tabella indica i valori stimati dell' α nel caso di rendimenti tick-by-tick e di rendimenti giornalieri. Ciascuna prova consiste in una simulazione della durata di 300, 500, 1000, 1500, 2000 cicli. Il "seed" utilizzato è 9999. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, gli agenti "razionali" seguono una strategia di chiusura delle posizioni aperte del tipo "take-profit only".

4.3.2 Volatilità del prezzo di offerta

È interessante analizzare come si comportano le serie di rendimenti al variare della varianza dei prezzi offerti sul mercato dagli agenti.

Con varianza dei prezzi offerti sul mercato s'intende il parametro "s" del modello di simulazione, e proprio su questo si va ad agire.

Il prezzo al quale gli agenti intendono formulare l'offerta da immettere nel mercato, infatti, è concepito nel modello come la realizzazione di una variabile casuale.

La distribuzione scelta per questa variabile aleatoria è Normale.

I parametri che influiscono sulla distribuzione della variabile casuale Normale sono "s" e "a", mediante la relazione $\sigma_p = s \cdot a$.

Il secondo parametro è un valore compreso tra 0 e 1, differente per ogni agente, e si può intendere come la componente "personale" di variabilità dell'offerta.

Il parametro "a" è modificato nel tempo dagli agenti "razionali", quando questi seguono una strategia differente da quella "random", attraverso un meccanismo di prova ed errore.

Il parametro "a" può aumentare se l'agente in passato ha ottenuto profitti dalle sue operazioni di compravendita e, specularmente, può diminuire se l'agente in passato ha ottenuto perdite.

Il primo parametro invece è uguale per tutti gli agenti, ed è impostato a piacimento dall'utente all'inizio della simulazione.

Esso può essere inteso dunque come la componente “di mercato”, nella sua totalità, di variabilità.

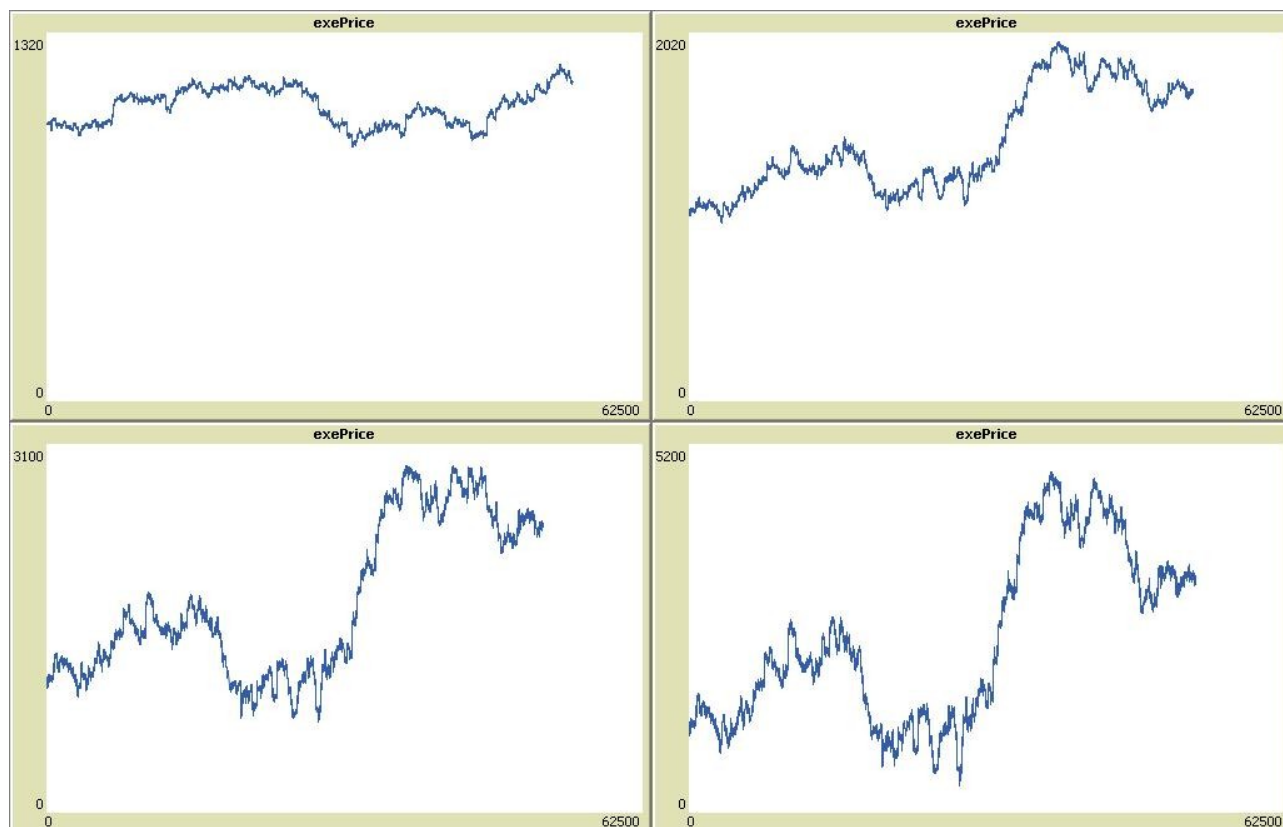


Illustrazione 4: Rappresentazione grafica dell'andamento del prezzo tick-by-tick. Ciascuna prova consiste in una simulazione della durata di 300 cicli, con valore di partenza del parametro “s” rispettivamente di: 10, 20, 50 e 100. Il “seed” utilizzato è -56. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito. La strategia degli agenti è “random”. Si può osservare come le variazioni del parametro “s” sembra influiscano soltanto sulla dimensione delle variazioni di prezzo, piuttosto che sull'andamento del prezzo stesso.

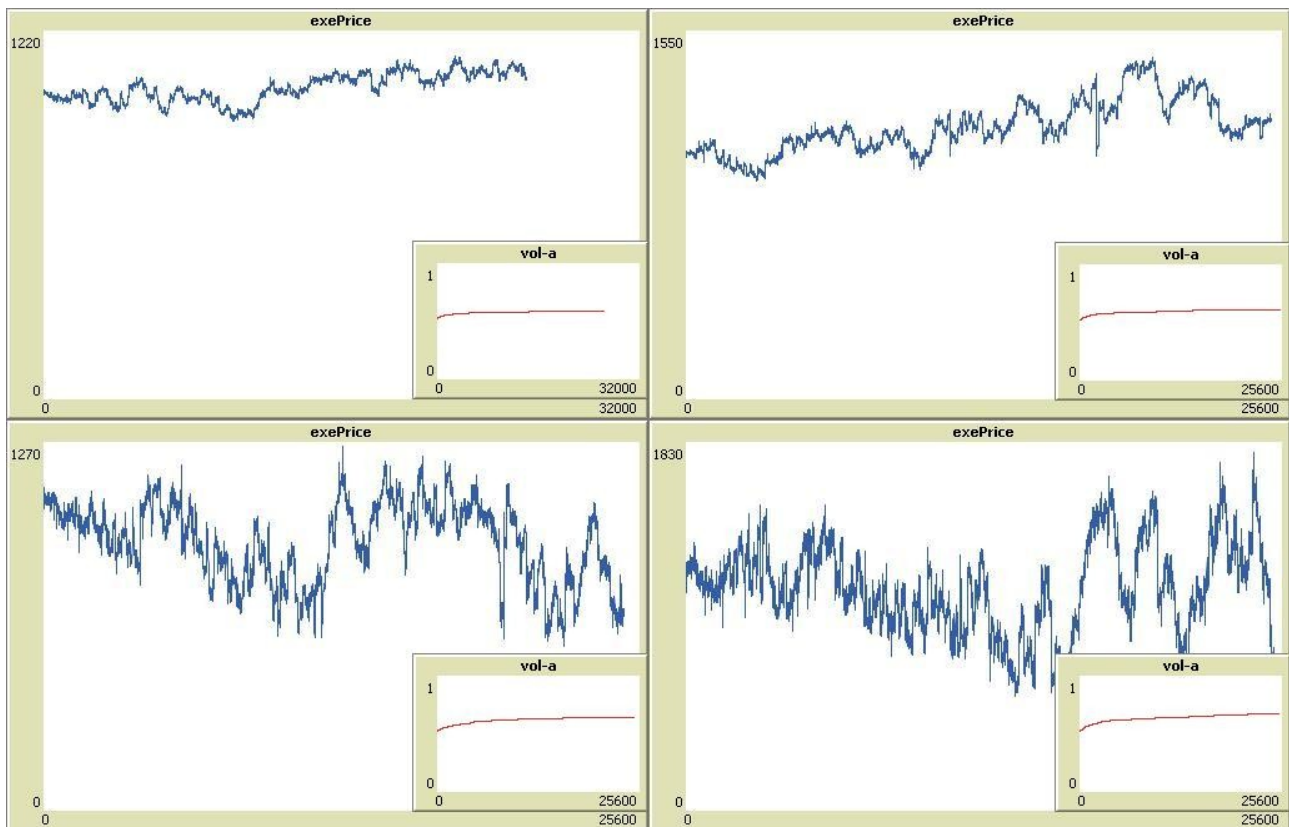


Illustrazione 5: Rappresentazione grafica dell'andamento del prezzo tick-by-tick e dell'andamento nel tempo della media del valore del parametro “a” di tutti gli agenti operanti sul mercato. Ciascuna prova consiste in una simulazione della durata di 300 cicli, con valore di partenza del parametro “s” rispettivamente di: 10, 20, 50 e 100. Il “seed” utilizzato è -56. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito. Gli agenti “razionali” seguono una strategia di chiusura delle posizioni aperte del tipo “take-profit only”. Si può osservare come le variazioni del parametro “s” sembra influiscano soltanto sulla dimensione delle variazioni di prezzo, piuttosto che sull'andamento del prezzo stesso, e come le variazioni sembra diventino più grandi con il passare del tempo. Il parametro “a” sembra modificarsi in maniera più accentuata nel corso del tempo quando si parte da valori più elevati di “s”.

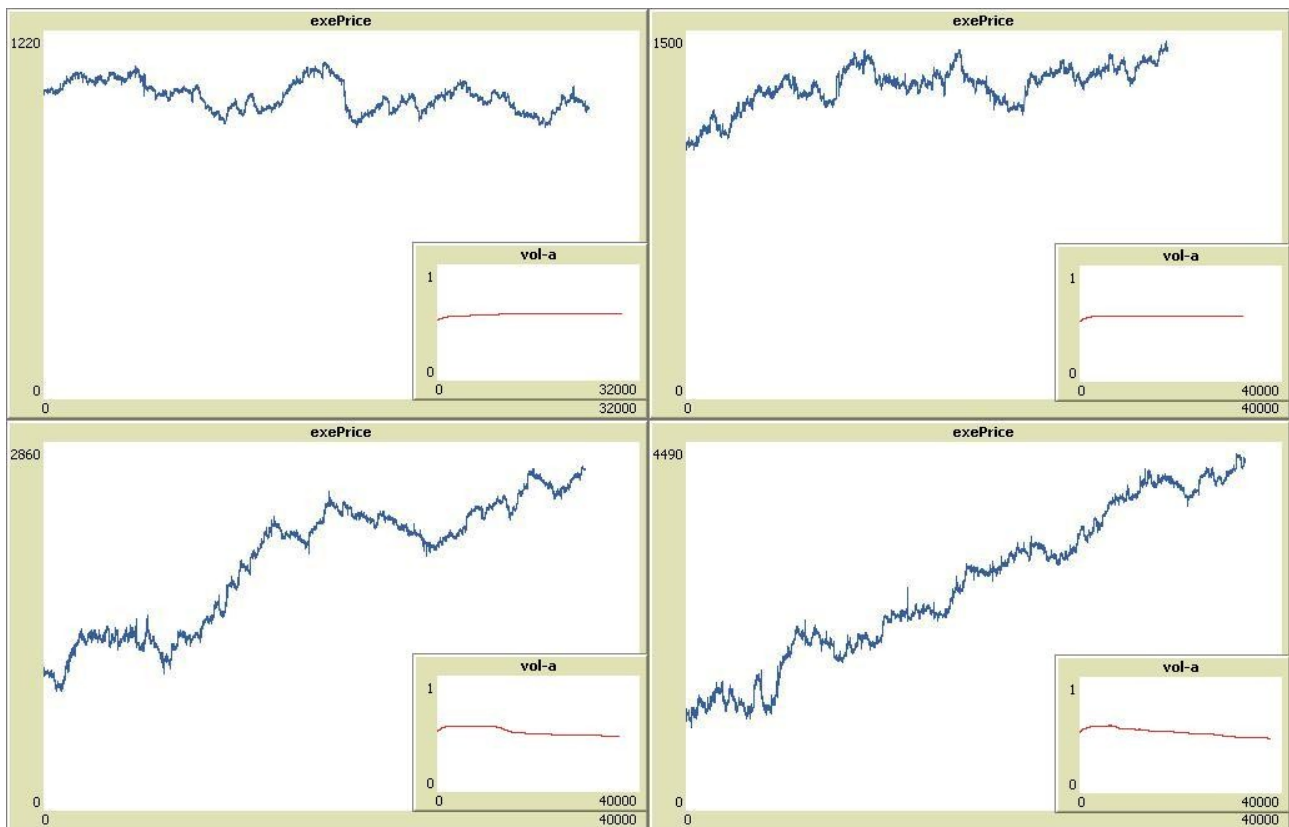


Illustrazione 6: Rappresentazione grafica dell'andamento del prezzo tick-by-tick e dell'andamento nel tempo della media del valore del parametro “a” di tutti gli agenti operanti sul mercato. Ciascuna prova consiste in una simulazione della durata di 300 cicli, con valore di partenza del parametro “s” rispettivamente di: 10, 20, 50 e 100. Il “seed” utilizzato è -56. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito. Gli agenti “razionali” seguono una strategia di chiusura delle posizioni aperte del tipo “stop-loss take-profit”. Si può osservare come le variazioni del parametro “s” sembra influiscano non soltanto sulla dimensione delle variazioni di prezzo, ma anche sull'andamento del prezzo stesso, a differenza di quanto sembra avvenire nei casi illustrati in precedenza. Anche in questo caso, come nel precedente, il parametro “a” sembra avere variazioni più accentuate nel corso del tempo quando si parte da valori maggiori di “s”.

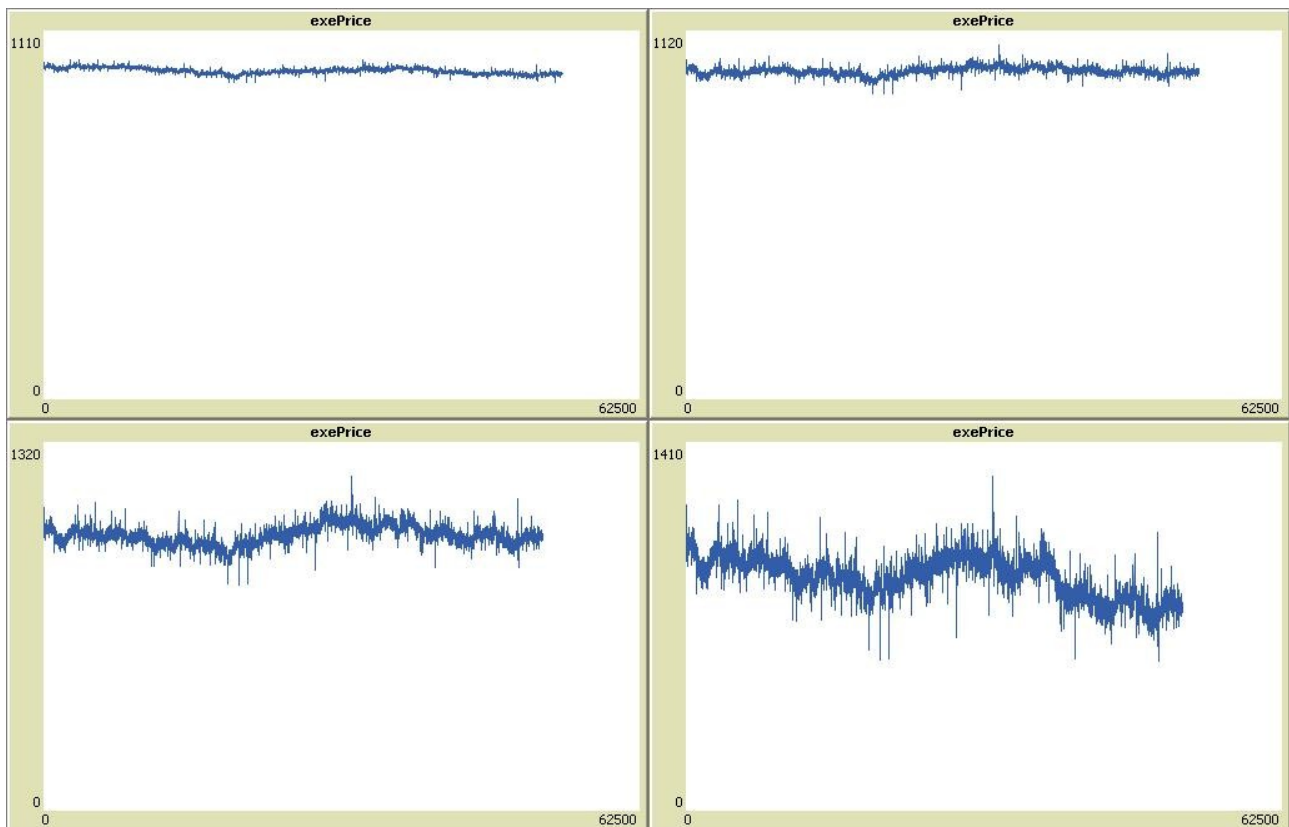


Illustrazione 7: Rappresentazione grafica dell'andamento del prezzo tick-by-tick. Ciascuna prova consiste in una simulazione della durata di 300 cicli, con valore di partenza del parametro “s” rispettivamente di: 10, 20, 50 e 100. Il “seed” utilizzato è -56. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo di chiusura. La strategia degli agenti è “random”. Si può osservare come le variazioni del parametro “s” sembra influiscano anche in questo caso in modo marcato sulla dimensione delle variazioni di prezzo.

Le prove effettuate evidenziano una relazione inversa tra varianza dei prezzi offerti e parametro α dei rendimenti generati.

Questo implica che se gli agenti immettono nel mercato offerte a prezzi più eterogenei, meno concentrati intorno all'ultimo prezzo eseguito o all'ultimo prezzo di chiusura, sarà più probabile incontrare rendimenti grandi.

Uno spunto d'analisi di questo tipo può essere utile per la spiegazione di alcuni fenomeni. Assumiamo che il modello rappresenti bene la realtà e analizziamo poi il mondo reale e il comportamento dei rendimenti.

Ci sono situazioni nelle quali si manifestano grandi variazioni di prezzo con frequenza anomala: tipici sono i cosiddetti momenti di “turbolenza” o le “bolle”.

Il modello tende a spiegare queste situazioni con un aumento della variabilità dei prezzi delle offerte immesse.

Se i prezzi immessi sul mercato rispecchiano il valore che le persone attribuiscono al titolo negoziato, un aumento della varianza di questi prezzi offerti potrebbe significare incertezza delle persone nell'attribuire un valore al bene in questione. Queste situazioni di incertezza sono appunto caratteristiche dei momenti di “turbolenza” e delle bolle.

Il modello ad agenti pertanto spiegherebbe l'aumentare di α anche attraverso l'aumentare di “s”.

Gli stimatori della varianza, sebbene spesso non consistenti in quanto non convergono al vero momento della popolazione se α è minore di 2, sono comunque influenzati dal variare di “s”, come dimostra la tabella 12.

Nel caso di $\alpha = 2$, la varianza dei rendimenti è legata al parametro γ della funzione caratteristica.

In questo caso specifico perciò un aumento della varianza comporta un aumento di γ , cioè della “scala” della distribuzione.

strategy	α	α	α	α	σ	σ	s
	tick-by-tick	day	week	month	tick-by-tick	day	
“random”	1,563	1,563	2	1,829	0,0014	0,0144	10
“random”	1,279	1,484	2	1,548	0,0017	0,0207	20
“random”	1,128	1,391	1,663	1,364	0,0032	0,0401	50
“random”	1,128	1,279	1,386	1,273	0,0072	0,0617	75
“random”	1,128	1,279	1,386	1,25	0,0082	0,0689	100
“random”	1,484	1,664	1,663	2	0,0016	0,0134	10
“random”	1,279	1,563	1,386	1,813	0,0026	0,0214	20
“random”	1,279	1,484	1,386	1,121	0,005	0,0406	50
“random”	1,128	1,391	1,279	1,273	0,0075	0,0604	75
“random”	1,128	1,273	1,273	1,114	0,0087	0,0712	100
“random”	1,563	1,563	1,56	2	0,0019	0,0176	10
“random”	1,279	1,563	1,386	2	0,0043	0,0406	20
“random”	1,029	1,128	1,266	2	0,1108	0,3499	50
“random”	1,128	1,279	1,266	2	0,1037	0,2784	75
“random”	1,029	1,128	1,121	2	0,1435	0,3584	100
“stop-loss take profit”	1,484	1,663	1,829	2	0,0015	0,0101	10
“stop-loss take profit”	1,279	1,729	2	1,813	0,0027	0,0187	20
“stop-loss take profit”	1,029	1,48	1,663	1,916	0,0043	0,0269	50
“stop-loss take profit”	1,029	1,279	1,391	2	0,0092	0,0569	75
“stop-loss take profit”	1,029	1,56	1,563	1,829	0,0069	0,0429	100
“take profit only”	1,563	1,808	1,729	1,279	0,0021	0,0138	10
“take profit only”	1,391	1,563	2	2	0,0046	0,0312	20
“take profit only”	1,279	1,484	1,391	2	0,0167	0,117	50
“take profit only”	1,279	1,484	1,729	2	0,0161	0,0986	75
“take profit only”	1,128	1,484	1,563	1,484	0,0446	0,1894	100

Tabella 12: La tabella indica i valori stimati dell' α nel caso di rendimenti tick-by-tick, di rendimenti giornalieri, settimanali e mensili, e i valori stimati di s nel caso di rendimenti tick-by-tick e rendimenti giornalieri. Ciascuna prova consiste in una simulazione della durata di 500 cicli. I “seed” utilizzati sono 45, -980, 111, 111 e 111. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito. Nei primi tre casi la strategia degli agenti “razionali” è “random”. Nel quarto caso gli agenti “razionali” seguono una strategia di chiusura delle posizioni aperte del tipo “stop-loss take-profit”. Nel quinto caso gli agenti “razionali” seguono una strategia di chiusura delle posizioni aperte del tipo “take-profit only”

4.3.3 Stazionarietà

Per verificare la stazionarietà di una serie di dati è possibile utilizzare il metodo proposto del confronto grafico [Mandelbrot, 1963].

Questa metodologia prevede inizialmente la suddivisione della serie in sottoserie di dati.

Successivamente si esegue un'analisi su ciascuna delle sottoserie così generate.

Nel caso in cui la serie di partenza sia stazionaria le sottoserie si devono comportare tutte nello stesso modo.

Come si può vedere dall'illustrazione 8 e dall'illustrazione 9, se una serie è stazionaria le sue sottoserie hanno andamenti uguali, ed i risultati delle singole analisi sono gli stessi.

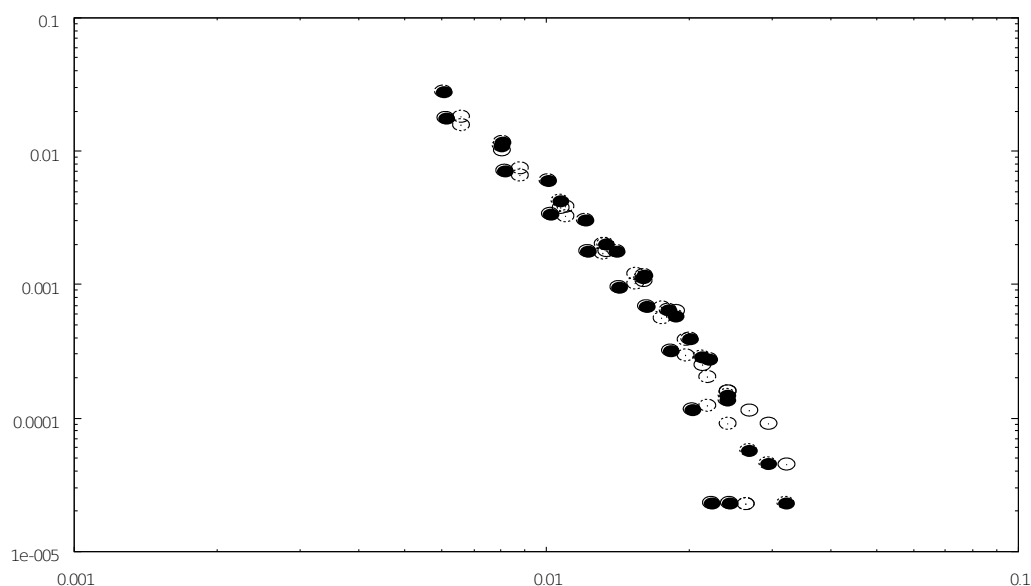


Illustrazione 8: Rappresentazione grafica in doppia scala logaritmica delle code delle distribuzioni dei rendimenti tick-by-tick generati attraverso il modello di simulazione. Le sei serie provengono tutte dalla medesima. La serie generata è stata divisa in due e quattro parti: ciascuna serie rappresentata nel grafico è una di queste parti. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, gli agenti "razionali" seguono una strategia di chiusura delle posizioni aperte del tipo "take-profit only". Il valore di "s" iniziale è 10 e il numero di cicli eseguiti è 2000. Il "seed" utilizzato è 7897. Si noti come le sei serie di rendimenti abbiano la medesima pendenza e valori molto simili. Un comportamento di questo genere può portare a ipotizzare che la serie nella sua totalità sia stazionaria nel tempo.

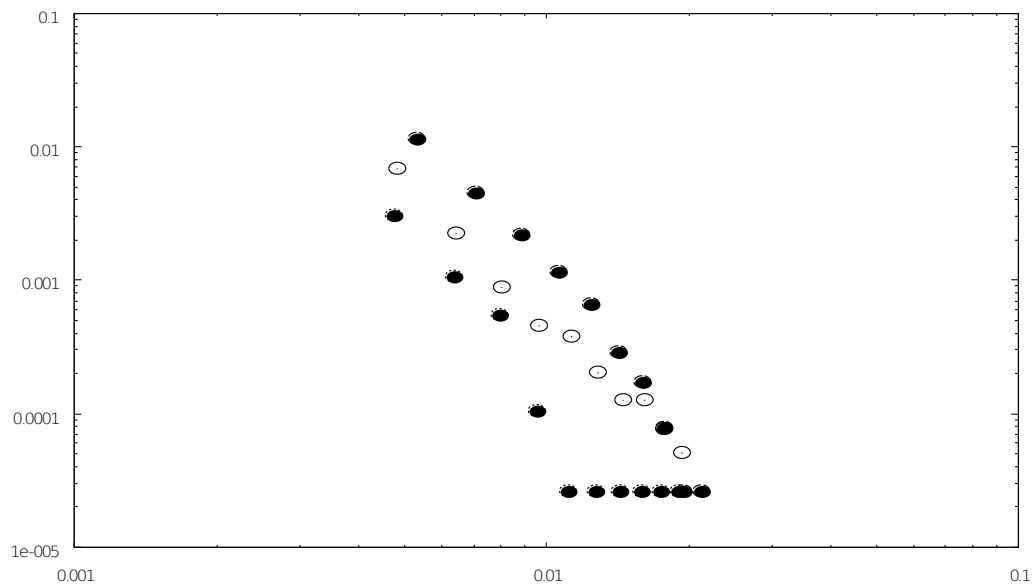


Illustrazione 9: Rappresentazione grafica in doppia scala logaritmica delle code delle distribuzioni dei rendimenti tick-by-tick generati attraverso il modello di simulazione. Le tre serie provengono tutte dalla medesima. La serie generata è stata divisa in tre parti: ciascuna serie rappresentata nel grafico è una di queste parti. La totalità degli agenti ha generato le proprie offerte seguendo l'ultimo prezzo eseguito, gli agenti "razionali" seguono una strategia di chiusura delle posizioni aperte del tipo "stop-loss take-profit". Il valore di "s" iniziale è 10 e il numero di cicli eseguiti è 1400. Il "seed" utilizzato è 7897. Si noti come le tre serie di rendimenti abbiano la medesima pendenza.

Conclusioni

Il lavoro svolto ha visto l'utilizzo combinato della simulazione ad agenti per la riproduzione stilizzata di un mercato finanziario e del software di calcolo per l'analisi dei dati, il tutto muovendosi sempre all'interno della cornice teorica proposta.

La modellizzazione ad agenti ha consentito di assumere ipotesi semplici riguardo il comportamento degli agenti, con risultati aggregati tuttavia dalle interpretazioni assolutamente non banali.

Il software di calcolo programmato ad hoc si rivela essere uno strumento altamente flessibile, molto potente e preciso, prezioso per lo studio di serie temporali di rendimenti.

Esso fornisce in particolare le basi per interpretazioni di carattere ad esempio economico o comportamentale dei fenomeni presi in analisi.

L'inquadramento teorico invece è il punto di partenza per uno studio dei mercati finanziari differente da quello che è stato svolto in larga parte fino agli anni 2000.

Le crisi finanziarie che si sono susseguite nel decennio 2000-2010 hanno messo in luce la necessità di intraprendere nuove strade nell'approcciarsi all'analisi della borsa.

Tutto ciò ovviamente con la speranza di riuscire ad ottenere nuovi e migliori modelli e strumenti, matematici e statistici, in grado di spiegare il comportamento dei mercati e in grado di prevedere con buona approssimazione il loro comportamento futuro.

Tuttavia la necessità più grande è quella di riuscire ad ottenere modelli robusti anche a periodi di "turbolenza" o crisi, capaci cioè di aiutare a comprendere cause e effetti anche quando siamo in presenza di fenomeni anomali, che riteniamo abbiano bassa probabilità di manifestarsi.

Gli strumenti utilizzati in passato, facendo largo uso di ipotesi di partenza quali quella di una distribuzione di frequenza Normale dei rendimenti, tendono a sottostimare il rischio di eventi rari e, cosa ancor più grave, gli effetti che questi eventi potrebbero avere.

La crisi finanziaria iniziata nel 2007 ha messo in evidenza, fra le altre cose, proprio i limiti degli strumenti utilizzati in finanza, che spesso si sono rivelati non solo inutili, ma addirittura dannosi.

Proponendoci un approccio ai mercati che parte dall'ipotesi che i rendimenti abbiano una distribuzione stabile, abbiamo innanzi tutto il vantaggio di avere un modello più generale⁹, in secondo luogo poniamo ipotesi meno restrittive e più verosimili¹⁰ ed infine abbiamo la possibilità

⁹ La distribuzione Normale è appunto un caso particolare di distribuzione stabile.

¹⁰ Eccetto il caso di $\alpha = 2$, le distribuzioni stabili hanno momenti secondi e superiori infiniti, pertanto non è richiesto che aumentando la numerosità del campione gli stimatori di questi momenti convergano al vero valore della

non banale di osservare il problema da un'altra angolazione, complementare e/o alternativa¹¹.

Ciò che comunque contraddistingue il lavoro qui svolto è proprio il tentativo di combinare teoria, simulazione e analisi, cercando di ottenere sinergie e mettere in luce le potenzialità dei singoli strumenti e del loro uso congiunto.

I risultati dell'analisi svolta sui dati reali e simulati ci mostrano come l'inquadramento teorico proposto abbia solide basi, sia verosimile e abbia buone potenzialità di sviluppo.

Un risultato interessante è aver ottenuto infatti rendimenti simulati che presentano distribuzioni di frequenza compatibili con le distribuzioni “stabili”, pur non avendo inserito alcuna ipotesi di questo tipo nel modello ad agenti, e che hanno distribuzioni simili e compatibili con quelle ottenute partendo da dati reali.

Dall'analisi delle prove di simulazione effettuate è emersa chiaramente anche una relazione tra la variabilità delle offerte che gli agenti inseriscono nel mercato e la probabilità di ottenere variazioni di prezzo ampie e anomali: si mostra infatti come all'aumentare del parametro “s” del modello ad agenti si riduca il valore stimato di α , indicatore proprio della probabilità del verificarsi di eventi rari.

Le analisi sulla stazionarietà delle serie evidenziano inoltre risultati incoraggianti: il mercato, simulato o reale, tende a comportarsi nello stesso modo con l'andare del tempo.

Questo ci garantisce il punto di partenza per costruire modelli e strumenti che, calibrati su quanto è avvenuto in passato, sono in grado di aiutarci a comprendere cosa potrebbe accadere in futuro.

Così come lavorando con serie stazionarie il nostro scopo è quello di creare qualcosa che funzioni sia in-sample che soprattutto out-of-sample, allo stesso modo lavorando con modelli di simulazione l'obiettivo dev'essere quello di creare qualcosa che possa essere applicato non solo al mondo artificiale, ma anche a quello reale.

Gli usi del modello sono molteplici. In primo luogo il modello può essere utilizzato come strumento a se stante di generazione di dati ad altissima frequenza, tick-by-tick.

Questi dati sono relativamente difficili da reperire, e possono avere anche un costo notevole.

Poiché il modello di simulazione è stato concepito nel modo più semplice possibile, le ipotesi di partenza possono essere modificate a piacimento, anche a livello di codice, mantenendo inalterato tuttavia il suo funzionamento.

Il cuore del modello infatti è una riproduzione il più fedele possibile del reale funzionamento del book di un mercato.

È possibile pertanto partire da una molteplicità di assunzioni differenti ed ottenere comunque dati

popolazione [Mandelbrot, 2009].

¹¹ Il parametro α ci dà un'indicazione sulla probabilità che abbiamo di ottenere realizzazioni di una variabile casuale molto distanti dal valore atteso.

generati allo stesso modo di dati reali.

Nel nostro caso specifico infatti, il modello è servito per generare dati simulati ad alta frequenza, partendo da un'ipotesi sulla volatilità delle offerte intorno all'ultimo prezzo eseguito o di chiusura, e da alcune possibili ipotesi sul comportamento degli agenti inseriti nel mercato.

L'idea è quella di fare ipotesi sul comportamento degli agenti che operando generano le variazioni di prezzo, piuttosto che sulle variazioni di prezzo stesse.

Queste ultime sono il risultato emergente dall'interazione continua di agenti con modi operandi più o meno simili tra loro.

Si può pertanto provare a simulare la volatilità futura dei prezzi e dei rendimenti partendo dalla volatilità degli ordini inseriti nel book.

Si sfrutta cioè la relazione individuata tra volatilità delle offerte e la volatilità dei prezzi.

Come detto, poiché la volatilità delle offerte è un parametro che assegniamo agli agenti e che gli agenti stessi possono modificare in modo adattivo durante la simulazione, la sua interpretazione è ricca di significato e può riguardare svariate discipline, quali ad esempio la finanza comportamentale.

Inoltre, facendo un'analisi a ritroso, si può provare a stimare il tipo di mercato in cui ci troviamo, con agenti più o meno razionali, con differenti tipi di comportamento, etc.

Per fare ciò si può provare a ricostruire dati simulati che siano verosimili ai dati che abbiamo osservato nella realtà.

Il modello allo stato attuale può essere costituito da una o due categorie di agenti, ma potrebbe comunque essere sviluppato introducendo altri tipi di agenti che seguono altri tipi di comportamenti.

Il modello “Artificial Financial Market” [Goncalves, 2003] ad esempio prende in considerazione il ruolo dell'informazione esogena e come questa sia valutata, utilizzando anche algoritmi che permettono agli agenti di imitare il comportamento degli altri.

Per quello che riguarda l'apprendimento nel tempo, nel modello ABM proposto gli agenti “razionali” imparano dal passato modificando il proprio parametro “a”, seguendo un meccanismo di prova ed errore: tuttavia ampliando il campo di ricerca è possibile pensare a sistemi più sofisticati di apprendimento.

Il software di calcolo inoltre, ha dimostrato di essere in grado di stimare parametri con buona approssimazione, come mostrano anche i test effettuati.

Proseguendo e sviluppando la strada intrapresa ci auspichiamo di riuscire ad ottenere strumenti che, basandosi su assunzioni teoriche e ipotesi coerenti con quelle qui proposte, siano in grado di garantire applicazioni pratiche fondamentali per chi opera sui mercati.

Siano capaci cioè di svolgere funzioni quali ad esempio l'asset pricing o che possano aiutare a guidare le scelte di composizione di portafoglio.

Lo stesso discorso vale per quello che riguarda la valutazione di opzioni o la gestione del rischio.

I mercati finanziari possono essere visti come le “macchine” dove sovente si decide buona parte del benessere dell'umanità.

Il problema tuttavia è che abbiamo conoscenze limitate e spesso confuse sul loro funzionamento.

In un mondo sempre più collegato in rete, la diffusione dei fenomeni è ampia e di portata sempre maggiore, le nostre conoscenze però sono ancora limitate sia per quello che riguarda le cause dei comportamenti, sia per ciò che riguarda la possibilità di controllarli.

Va detto in ogni caso che questo è solo l'incipit di un nuovo modo di approcciarsi ai mercati: la strada da percorrere è ancora lunga e proprio per questo motivo è necessario fare tesoro anche dei piccoli risultati incoraggianti.

Bibliografia

- : Black, F. and Scholes, M., *The pricing of options and corporate liabilities*, 1973, *The journal of political economy*, 637--654, JSTOR
- : Chambers, J.M., Mallows, C.L. e Stuck, B.W., *A method for simulating stable random variables*, 1976, *Journal of the American Statistical Association*, 340--344, JSTOR
- : Fama, E.F. and Roll, R., *Parameter estimates for symmetric stable distributions*, 1971, *Journal of the American Statistical Association*, 331--338, JSTOR
- : Gnedenko, B.V., Kolmogorov, A.N., Chung, K.L. e Doob, J.L., *Limit distributions for sums of independent random variables*, 1968, Addison-Wesley Reading, MA
- : Hall, P., *On some simple estimates of an exponent of regular variation*, 1982, *Journal of the Royal Statistical Society*, 37--42, JSTOR
- : Lasser, C. e Omohundro, S.M., *The essential Star-lisp manual*, 1986, Thinking Machines Corporation
- : Lévy, P., *Calcul des probabilités*, 1925, 9, Gauthier-Villars Paris
- : Lévy, P. e Borel, É., *Théorie de l'addition des variables aléatoires*, 1954, 1, Gauthier-Villars Paris
- : Mandelbrot, B., *The variation of certain speculative prices*, 1963, *The journal of business*, 36, 4, 394--419, JSTOR
- : Mandelbrot, B., "New Methods of Statistical Economics," revisited: *Short versus long tails and Gaussian versus power-law distributions*, 2009, *Complexity*, 14, 3, 55--65, Wiley Online Library
- : Matsumoto, M. and Nishimura, T., *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*, 1998, *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8, 1, 3--30, ACM
- : McCulloch, J.H., *Simple consistent estimators of stable distribution parameters*, 1986, *Communications in Statistics: Simulation and Computation*, 15, 4, 1109--1136,
- : Mittnik, S. e Paoletta, M.S., *A simple estimator for the characteristic exponent of the stable Paretian distribution*, 1999, *Mathematical and computer modelling*, 29, 10-12, 161--176, Elsevier
- : Newman, M.E.J., *Power laws, Pareto distributions and Zipf's law*, 2004, Arxiv preprint cond-mat/0412004

- : Nolan, J.P., *Stable distributions: Models for heavy-tailed data*, 2003
- : Nolan, J.P., *Modelling financial data with stable distributions*, 2003, *Handbook of Heavy Tailed Distributions in Finance*, 106--129, Elsevier-North Holland, Amsterdam
- : Papert, S., *Mindstorms*, 1980, Harvester Pr.
- : Taleb, N.N., *Finiteness of variance is irrelevant in the practice of quantitative finance*, 2009, *Complexity*, 14, 3, 66--76, Wiley Online Library
- : Tinter, G., *Business Cycles in the United States, 1919-1932*, 1940
- : Wilensky, U., *StarLogoT*, 1997, Center for Connected Learning and Computer-Based Modeling, Northwestern University

Appendice A

Codice di programmazione NetLogo del modello di simulazione

Di seguito è riportato il codice utilizzato per le procedure in NetLogo 4.1.3:

```
breed [IRRs IRR]
breed [RATLs RATL]
breed [RATSSs RATS]

turtles-own[buy sell pass price cash stocks
            passLevel rat sl tp a in-mkt]
globals    [logB logS exePrice exePriceD
            returns returnsD orders t turns]

to setup

  ca
  random-seed seed
  set exePrice []
  set exePriceD []
  set exePrice fput 1000 exePrice
  set exePriceD fput 1000 exePriceD
  set orders []

  create-IRRs nAgents

  let side sqrt nAgents

  let step max-pxcor / side

ask IRRs
[set shape "person"
 set size 1
 set stocks 0
 set cash 0
 set in-mkt 0
 set a random-float 1
 ifelse random-float 1 > p-rat [set rat 0 set color white]
                               [set rat 1 set color grey]
 ifelse passLevelRandom = "Yes"[set passLevel random-float 1]
                               [set passLevel passLevelEso]]

let an 0
let x 0
let y 0
```

```

while [an < nAgents]
  [if x > (side - 1) * step [set y y + step
    set x 0]
  ask turtle an
  [setxy x y]
  set x x + step
  set an an + 1
]
end

to go

set logB []
set logS []
ask turtles [set in-mkt 0]
set t 0
set turns []
ask IRRs [set turns fput who turns]

while [not empty? turns]
[ask turtle item 0 turns
  [ifelse breed = IRRs
    [set-price]
    [set-sl-tp-price]
  book]
  set turns but-first turns
  if strategies = "stop-loss take-profit"
  [if (any? RATLs with [in-mkt = 0 and (sl > item 0 exePrice or tp < item 0
exePrice)])
    or (any? RATSS with [in-mkt = 0 and (tp > item 0 exePrice or sl < item 0
exePrice)])
    [ask (turtle-set RATLs with [in-mkt = 0 and (sl > item 0 exePrice or tp <
item 0 exePrice)]
      RATSS with [in-mkt = 0 and (tp > item 0 exePrice or sl < item 0
exePrice)])
    [set turns fput who turns
      set turns remove-duplicates turns]]]
  if strategies = "take-profit only"
  [if (any? RATLs with [in-mkt = 0 and (tp < item 0 exePrice)])
    or (any? RATSS with [in-mkt = 0 and (tp > item 0 exePrice)])
    [ask (turtle-set RATLs with [in-mkt = 0 and (tp < item 0 exePrice)]
      RATSS with [in-mkt = 0 and (tp > item 0 exePrice)])
    [set turns fput who turns
      set turns remove-duplicates turns]]]
set exePriceD fput item 0 exePrice exePriceD
tick

end

to book

```

```

if not pass
[let tmp []
set tmp lput price tmp
set tmp lput who tmp
set tmp lput ticks tmp
set tmp lput t tmp
set tmp lput a tmp
set tmp lput rat tmp
set tmp lput item 0 exePrice tmp
ifelse buy [
  ifelse not empty? logS [set tmp lput item 0 (item 0 logS) tmp]
    [set tmp lput "e" tmp]][
  ifelse not empty? logB [set tmp lput item 0 (item 0 logB) tmp]
    [set tmp lput "e" tmp]]
ifelse breed = IRRs [set tmp lput 1 tmp][set tmp lput 0 tmp]
ifelse buy [set tmp lput 1 tmp][set tmp lput 0 tmp]
set orders fput tmp orders

if buy [set logB lput tmp logB]
set logB reverse sort-by [item 0 ?1 < item 0 ?2] logB
if (not empty? logB and not empty? logS) [
  if item 0 (item 0 logB) >= item 0 (item 0 logS)
  [set exePrice fput item 0 (item 0 logS) exePrice
  let agB item 1 (item 0 logB)
  let agS item 1 (item 0 logS)
  set t t + 1
  graph
  ask turtle agB
  [set stocks stocks + 1
  set cash cash - item 0 exePrice
  if strategies != "random" [
  ifelse breed = IRRs
  [change-breed-RATL]
  [change-breed-IRR]]]
  ask turtle agS
  [set stocks stocks - 1
  set cash cash + item 0 exePrice
  if strategies != "random" [
  ifelse breed = IRRs
  [change-breed-RATS]
  [change-breed-IRR]]]
  set logB but-first logB
  set logS but-first logS]]

if sell [set logS lput tmp logS]
set logS sort-by [item 0 ?1 < item 0 ?2] logS

```

```

if (not empty? logB and not empty? logS) [
  if item 0 (item 0 logB) >= item 0 (item 0 logS)
    [set exePrice fput item 0 (item 0 logB) exePrice
     let agB item 1 (item 0 logB)
     let agS item 1 (item 0 logS)
     set t t + 1
     graph
     ask turtle agB
       [set stocks stocks + 1
        set cash cash - item 0 exePrice
        if strategies != "random" [
          ifelse breed = IRRs
            [change-breed-RATL]
            [change-breed-IRR]]]
     ask turtle agS
       [set stocks stocks - 1
        set cash cash + item 0 exePrice
        if strategies != "random" [
          ifelse breed = IRRs
            [change-breed-RATS]
            [change-breed-IRR]]]
     set logB but-first logB
     set logS but-first logS]]
]

end

to graph

set-current-plot "exePrice"
plot item 0 exePrice
set-current-plot "vol-a"
plot mean [a] of turtles

end

to save_results

let exePrice2 exePrice
set exePrice remove-item ((length exePrice) - 1) exePrice
set exePrice2 remove-item 0 exePrice2
set returns []
(foreach exePrice exePrice2 [set returns lput precision (ln ?1 - ln ?2) 4
returns])
file-open "C:\\Users\\Andrea\\Desktop\\prova.txt"
foreach returns [file-print ?]
file-close

end

```

```

to save_resultsDay

let exePriceD2 exePriceD
set exePriceD remove-item ((length exePriceD) - 1) exePriceD
set exePriceD2 remove-item 0 exePriceD2
set returnsD []
(foreach exePriceD exePriceD2 [set returnsD lput precision (ln ?1 - ln ?2) 4
returnsD])
file-open "C:\\Users\\Andrea\\Desktop\\prova2.txt"
foreach returnsD [file-print ?]
file-close

end

```

```

to set-price

```

```

if breed = IRRs [
  ifelse random-float 1 < passLevel
    [set pass True][set pass False]
  ifelse not pass
    [ifelse random-float 1 < 0.5 [set buy True set sell False]
      [set sell True set buy False]]
    [set buy False set sell False]
  ifelse rat = 0 [if irrational = "last close"
    [set price item 0 exePriceD + int (random-normal 0 s * a)
    if price < 1 [set price 1]]
    if irrational = "last price"
    [set price item 0 exePrice + int (random-normal 0 s * a)
    if price < 1 [set price 1]]
    if irrational = "1000"
    [set price int (random-normal 1000 s * a)
    if price < 1 [set price 1]]]
    [if rational = "last close"
    [set price item 0 exePriceD + int (random-normal 0 s * a)
    if price < 1 [set price 1]]
    if rational = "last price"
    [set price item 0 exePrice + int (random-normal 0 s * a)
    if price < 1 [set price 1]]
    if rational = "1000"
    [set price int (random-normal 1000 s * a)
    if price < 1 [set price 1]]] ]

```

```

end

```

```

to set-sl-tp-price

```

```

  set in-mkt 1

```

```

if breed = RATLs [if item 0 exePrice < sl and
  strategies = "stop-loss take-profit"
  [set price int sl - 1
  if price < 1 [set price 1]]
  if item 0 exePrice > tp
  [set price int tp - 1
  if price < 1 [set price 1]]]
if breed = RATSS [if item 0 exePrice > sl and
  strategies = "stop-loss take-profit"
  [set price int sl + 1
  if price < 1 [set price 1]]
  if item 0 exePrice < tp
  [set price int tp + 1
  if price < 1 [set price 1]]]

end

to change-breed-IRR
if rat = 1 [set in-mkt 0
  set breed IRRs set color grey
  set sl 0 set tp 0 set shape "person"
  let anew random-float 1 ifelse cash < 0
  [if a > anew [set a anew]]
  [if a < anew [set a anew]]]

end

to change-breed-RATS
if rat = 1 [set breed RATSS
  set color green set shape "person"
  set buy True set sell False set pass False
  set sl (( item 0 exePrice ) * (1 + (a ^ 2)))
  set tp (( item 0 exePrice ) * (1 - (a ^ 2)))]

end

to change-breed-RATL
if rat = 1 [set breed RATLs
  set color red set shape "person"
  set buy False set sell True set pass False
  set tp (( item 0 exePrice ) * (1 + (a ^ 2)))
  set sl (( item 0 exePrice ) * (1 - (a ^ 2)))]

end

```

Appendice B

Codice di programmazione del software Maxima

Di seguito è riportato il codice utilizzato per l'analisi dati in ambiente wxMaxima 0.8.7 (Maxima 5.25.1):

```
load(numericalio)$

a: read_list("C:/Users/Andrea/Desktop/prova.txt")$

ad: read_list("C:/Users/Andrea/Desktop/prova2.txt")$

alphamc: matrix([2,2,2,2,2,2,2],
[1.916,1.924,1.924,1.924,1.924,1.924,1.924],
[1.808,1.813,1.829,1.829,1.829,1.829,1.829],
[1.729,1.73,1.737,1.745,1.745,1.745,1.745],
[1.664,1.663,1.663,1.668,1.676,1.676,1.676],
[1.563,1.56,1.553,1.548,1.547,1.547,1.547],
[1.484,1.48,1.471,1.46,1.448,1.438,1.438],
[1.391,1.386,1.378,1.364,1.337,1.318,1.318],
[1.279,1.273,1.266,1.25,1.21,1.184,1.15],
[1.128,1.121,1.114,1.101,1.067,1.027,0.973],
[1.029,1.021,1.014,1.004,0.974,0.935,0.874],
[0.896,0.892,0.887,0.883,0.855,0.823,0.769],
[0.818,0.812,0.806,0.801,0.780,0.756,0.691],
[0.698,0.695,0.692,0.689,0.676,0.656,0.595],
[0.593,0.59,0.588,0.586,0.579,0.563,0.513])$

vamc: [2.439,2.5,2.6,2.7,2.8,3,3.2,3.5,4,5,6,8,10,15,25]$
vbmc: [0.0,0.1,0.2,0.3,0.5,0.7,1]$

load (descriptive)$

m: mean (a);
var (a);
s: sqrt (var(a));
qr :qrange(a);

m2: mean (ad);
var (ad);
s2: sqrt (var(ad));
qr2 :qrange(ad);

s/sqrt(2),numer;
qr/2;

q95: quantile(a, 0.95)$
q05: quantile(a, 0.05)$
q75: quantile(a, 0.75)$
q25: quantile(a, 0.25)$
q50: quantile(a, 0.50)$

q95d: quantile(ad, 0.95)$
q05d: quantile(ad, 0.05)$
```

```

q75d: quantile(ad, 0.75)$
q25d: quantile(ad, 0.25)$
q50d: quantile(ad, 0.50)$

valpha: (q95 - q05)/(q75 - q25);
vbeta: (q95 + q05 - 2*q50)/(q95 - q05);

valpha2: (q95d - q05d)/(q75d - q25d);
vbeta2: (q95d + q05d - 2*q50d)/(q95d - q05d);

vamc1: vamc$
vamc1: abs(vamc1 - valpha)$
lmin(vamc1)$

stemp: 0$
for i: 1 unless vamc1 [i] = lmin(vamc1) do stemp: i$
stemp: stemp+1;

vbmcl: vbmc$
vbmcl: abs(vbmcl - vbeta)$
lmin(vbmcl)$

rtemp: 0$
for j: 1 unless vbmcl [j] = lmin(vbmcl) do rtemp: j$
rtemp: rtemp+1;

alpha1: alphamc [stemp][rtemp];

vamc2: vamc$
vamc2: abs(vamc2 - valpha)$
lmin(vamc2)$

stemp: 0$
for i: 1 unless vamc2 [i] = lmin(vamc2) do stemp: i$
stemp: stemp+1$

vbmc2: vbmc$
vbmc2: abs(vbmc2 - vbeta)$
lmin(vbmc2)$

rtemp: 0$
for j: 1 unless vbmc2 [j] = lmin(vbmc2) do rtemp: j$
rtemp: rtemp+1$

alpha2: alphamc [stemp][rtemp];

b: continuous_freq (a, 30);

bd: continuous_freq (ad, 30);

c: first (b)$
load(basic)$
pop (c)$

d: second (b)$

g: reverse(c)$

h: reverse(second(b))$

```

```

fal: [first(d) , second(d) , third(d) , fourth(d) , fifth (d) ,
sixth(d) , seventh(d) , eighth(d) , ninth(d) , tenth(d)];
fa2: [first(h) , second(h) , third(h) , fourth(h) , fifth (h) ,
sixth(h) , seventh(h) , eighth(h) , ninth(h) , tenth(h)];

fc1: [first(d), first(d) + second(d), first(d) + second(d) + third(d),
first(d) + second(d) + third(d) + fourth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d), first(d) +
second(d) + third(d) + fourth(d) + fifth(d) + sixth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d) + sixth(d) +
seventh(d), first(d) + second(d) + third(d) + fourth(d) + fifth (d)+
sixth(d) + seventh(d) + eighth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d)+ sixth(d) +
seventh(d) + eighth(d) + ninth(d),first(d) + second(d) + third(d) +
fourth(d) + fifth (d)+ sixth(d) + seventh(d) + eighth(d) + ninth(d) +
tenth(d)];

fc2: [first(h), first(h) + second(h), first(h) + second(h) + third(h),
first(h) + second(h) + third(h) + fourth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h), first(h) +
second(h) + third(h) + fourth(h) + fifth(h) + sixth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h) + sixth(h) +
seventh(h), first(h) + second(h) + third(h) + fourth(h) + fifth (h)+
sixth(h) + seventh(h) + eighth(h),
first(h) + second(h) + third(h) + fourth(h) +fifth (h)+ sixth(h) +
seventh(h) + eighth(h) + ninth(h),first(h) + second(h) + third(h) +
fourth(h) + fifth (h)+ sixth(h) + seventh(h) + eighth(h) + ninth(h) +
tenth(h)];

n: length (a)$

f1: create_list(i / n,i,fc1);
f2: create_list(i / n,i,fc2);

v: [first(c) , second(c) , third(c) , fourth(c) , fifth (c) , sixth(c) ,
seventh(c) , eighth(c) , ninth(c) , tenth(c)];
u1: abs(v);

u2: [first(g) , second(g) , third(g) , fourth(g) , fifth (g) , sixth(g) ,
seventh(g) , eighth(g) , ninth(g) , tenth(g)];

c: first (bd)$
pop (c)$

d: second (bd)$

g: reverse(c)$

h: reverse(second(bd))$

fal2d: [first(d) , second(d) , third(d) , fourth(d) , fifth (d) ,
sixth(d) , seventh(d) , eighth(d) , ninth(d) , tenth(d)];
fa2d: [first(h) , second(h) , third(h) , fourth(h) , fifth (h) ,
sixth(h) , seventh(h) , eighth(h) , ninth(h) , tenth(h)];

fc1d: [first(d), first(d) + second(d), first(d) + second(d) + third(d),
first(d) + second(d) + third(d) + fourth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d), first(d) +
second(d) + third(d) + fourth(d) + fifth(d) + sixth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d) + sixth(d) +

```

```

seventh(d), first(d) + second(d) + third(d) + fourth(d) + fifth (d)+
sixth(d) + seventh(d) + eighth(d),
first(d) + second(d) + third(d) + fourth(d) + fifth (d)+ sixth(d) +
seventh(d) + eighth(d) + ninth(d),first(d) + second(d) + third(d) +
fourth(d) + fifth (d)+ sixth(d) + seventh(d) + eighth(d) + ninth(d) +
tenth(d)];

fc2d: [first(h), first(h) + second(h), first(h) + second(h) + third(h),
first(h) + second(h) + third(h) + fourth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h), first(h) +
second(h) + third(h) + fourth(h) + fifth(h) + sixth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h) + sixth(h) +
seventh(h), first(h) + second(h) + third(h) + fourth(h) + fifth (h)+
sixth(h) + seventh(h) + eighth(h),
first(h) + second(h) + third(h) + fourth(h) + fifth (h)+ sixth(h) +
seventh(h) + eighth(h) + ninth(h),first(h) + second(h) + third(h) +
fourth(h) + fifth (h)+ sixth(h) + seventh(h) + eighth(h) + ninth(h) +
tenth(h)];

nd: length (ad)$

f1d: create_list(i / nd,i,fc1d);
f2d: create_list(i / nd,i,fc2d);

vd: [first(c) , second(c) , third(c) , fourth(c) , fifth (c) , sixth(c) ,
seventh(c) , eighth(c) , ninth(c) , tenth(c)];
uld: abs(v);

u2d: [first(g) , second(g) , third(g) , fourth(g) , fifth (g) , sixth(g) ,
seventh(g) , eighth(g) , ninth(g) , tenth(g)];

load(distrib)$

n1: create_list(cdf_normal(i,m,s),i,v),numer$
n2: create_list(1 - cdf_normal(i,m,s),i,u2)$

n: append(n1,n2)$
un: append(u1,u2)$

cc1: create_list(cdf_cauchy(i,m,(qr / 2)),i,v)$
cc2: create_list(1 - cdf_cauchy(i,m,(qr / 2)),i,u2)$

cc: append(cc1,cc2)$

plot2d([[discrete,u1 , f1],[discrete,u2 , f2],[discrete,un , n],
[discrete,un , cc]], [style, points],[logy] , [logx]);

plot2d([[discrete,u1 , f1],[discrete,u2 , f2],[discrete,uld , f1d],
[discrete,u2d , f2d]], [style, points],[logy] , [logx]);

u11: create_list(log(i/(last(u1))),i,u1)$

u22: create_list(log(i/(last(u1))),i,u2)$

X11 : matrix (u11)$
X22 : matrix (u22)$

F11 : matrix (fa1)$
F22 : matrix (fa2)$

```

```

transpose (F11)$
transpose (F22)$

alpha1: 1 + ((last(fc1))/ (X11 . F11));
alpha2: 1 + ((last(fc2))/ (X22 . F22));

p11:(u1 / last(u1))^(1 - alpha1);
p12:(u2 / last(u2))^(1 - alpha2);

fp11: f1 / last(f1);
fp12: f2 / last(f2);

plot2d([[discrete,u1 , fp11],[discrete,u2 , fp12],[discrete,u1 , p11],
[discrete,u2 , p12]], [style, points],[logy] , [logx]);

ullog : create_list(log(x),x,u1)$
u2log : create_list(log(x),x,u2)$
f1log : create_list(log(x),x,f1)$
f2log : create_list(log(x),x,f2)$
unlog : create_list(log(x),x,u1)$

ntemp: length(n1)$
n1: delete(0.0,n1)$
nlog : create_list(log(x),x,n1)$

unlog1: unlog;
for i: 1 thru (ntemp - length(n1)) do pop(unlog)$

cclog : create_list(log(x),x,cc1)$

M11 : matrix (ullog, f1log)$
M12 : matrix (u2log, f2log)$
M1n : matrix (unlog, nlog)$
M1cc : matrix (unlog1, cclog)$

transpose (M11)$
transpose (M12)$
transpose (M1n)$
transpose (M1cc)$

lsquares_estimates (M11, [y,x], y = A+B*x, [A,B]),numer;
lsquares_estimates (M12, [y,x], y = A+B*x, [A,B]),numer;
lsquares_estimates (M1n, [y,x], y = A+B*x, [A,B]),numer;
lsquares_estimates (M1cc, [y,x], y = A+B*x, [A,B]),numer;

```

i Stima di α

Il metodo di stima proposto da McCulloch nel paper “Simple consistent estimators of stable distribution parameters” prevede la stima dei quattro parametri di una distribuzione stabile attraverso la stima di cinque quantili campionari e l'utilizzo di tabelle di valori precalcolati.

Lo stimatore proposto per il parametro α è simile a quello proposto da Fama e Roll [Fama e Roll, 1971], e prevede la stima di α in un intervallo $[0,6 ; 2]$, sia nel caso di distribuzione simmetrica, $\beta = 0$, che nel caso di distribuzione asimmetrica.

La relazione a cui l'autore fa riferimento è:

$$\alpha = \Psi_1(u_\alpha, u_\beta) \quad \text{con} \quad \Psi_1(u_\alpha, -u_\beta) = -\Psi_1(u_\alpha, u_\beta) \quad ,$$

La tabella di valori da utilizzare per effettuare il confronto e l'estrapolazione della stima del valore di α è la seguente:

v_α	v_β	0.0	0.1	0.2	0.3	0.5	0.7	1.0
2.4	2	2	2	2	2	2	2	2
2.5	1.916	1.924	1.924	1.924	1.924	1.924	1.924	1.924
2.6	1.808	1.813	1.829	1.829	1.829	1.829	1.829	1.829
2.7	1.729	1.73	1.737	1.745	1.745	1.745	1.745	1.745
2.8	1.664	1.663	1.663	1.668	1.676	1.676	1.676	1.676
3	1.563	1.56	1.553	1.548	1.547	1.547	1.547	1.547
3.2	1.484	1.48	1.471	1.46	1.448	1.438	1.438	1.438
3.5	1.391	1.386	1.378	1.364	1.337	1.318	1.318	1.318
4	1.279	1.273	1.266	1.25	1.21	1.184	1.15	1.15
5	1.128	1.121	1.114	1.101	1.067	1.027	0.973	0.973
6	1.029	1.021	1.014	1.004	0.974	0.935	0.874	0.874
8	0.896	0.892	0.887	0.883	0.855	0.823	0.769	0.769
10	0.818	0.812	0.806	0.801	0.780	0.756	0.691	0.691
15	0.698	0.695	0.692	0.689	0.676	0.656	0.595	0.595

ii Mean

Il comando mean fornisce la media campionaria di una lista di dati, definita come:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad .$$

iii Var

Il comando var fornisce la varianza campionaria di una lista di dati, definita come:

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 .$$

iv Qrange

Il comando qrange fornisce la differenza interquartile di una lista di dati, definita come:

$$IQR = Q_{0,75} - Q_{0,25} = F^{-1}(0,75) - F^{-1}(0,25) .$$

v cdf_normal

Il comando cdf_normal fornisce il valore della funzione di distribuzione di una variabile aleatoria normale di media m e varianza s^2 , definita come:

$$F(x) = P(X \leq x) = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{x - m}{s \sqrt{2}} \right) \right]$$

erf è la funzione degli errori, che si definisce come:

$$\operatorname{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt .$$

La sua funzione caratteristica è:

$$\varphi(t) = \exp \left(i \mu t - \frac{1}{2} \sigma^2 t^2 \right) ,$$

da cui segue che la varianza è pari a 2γ .

vi cdf_cauchy

Il comando cdf_cauchy fornisce il valore della funzione di distribuzione di una variabile aleatoria di Cauchy di parametri a e b , definita come:

$$F(x) = \frac{1}{\pi} \arctan \left(\frac{x - a}{b} \right) + \frac{1}{2} .$$

La funzione dei quantili, cioè la funzione inversa della funzione di distribuzione, è definita come:

$$Q(p) = a + b \tan \left[\pi \left(p - \frac{1}{2} \right) \right],$$

da cui segue che la differenza interquartile è data da $2b$, cioè nella notazione adottata comunemente da 2γ .

vii stimatore di massima verosimiglianza per il parametro α di una power law

Data una distribuzione di probabilità D , con funzione di massa (o densità, se continua) di probabilità L_D , caratterizzata da un parametro ν , dato un campione di dati osservati $\{x_i\}_{i=1}^n$ di dimensione n si può calcolare la probabilità associata ai dati osservati:

$$P(\{x_i\}_{i=1}^n | \nu) = L_D(\nu | \{x_i\}_{i=1}^n)$$

D'altra parte, può darsi che il parametro ν sia ignoto, sebbene sia noto che il campione è estratto dalla distribuzione D . Un'idea per stimare ν è allora utilizzare i dati a nostra disposizione: $\{x_i\}_{i=1}^n$ per ottenere informazioni su ν .

Il metodo della massima verosimiglianza ricerca il valore più verosimile di ν , ossia ricerca il valore del parametro che massimizza la probabilità di aver ottenuto il campione dato. Da un punto di vista matematico,

$L_D(\nu | \{x_i\}_{i=1}^n)$ è detta funzione di verosimiglianza, e lo stimatore di massima verosimiglianza è ottenuto come:

$$\hat{\nu} = \max_{\nu} L_D(\nu | x_1, \dots, x_n).$$

La massimizzazione della funzione di verosimiglianza è equivalente a massimizzarne il logaritmo.

Nel nostro caso, se assumiamo che i rendimenti sono i.i.d.:

$$p(x) = \frac{(\alpha - 1)}{x_{min}} \left(\frac{x}{x_{min}} \right)^{-\alpha}$$

$$L_D(\nu | \{x_i\}_{i=1}^n) = L_D(\alpha, x_{min} | \{x_i\}_{i=1}^n) = \prod_{i=1}^n p(x_i) = \prod_{i=1}^n \frac{(\alpha - 1)}{x_{min}} \left(\frac{x_i}{x_{min}} \right)^{-\alpha}$$

$$\ln(L_D(\nu | \{x_i\}_{i=1}^n)) = n \ln \frac{(\alpha - 1)}{x_{min}} - \alpha \sum_{i=1}^n \ln \left(\frac{x_i}{x_{min}} \right)$$

$$\max_{\alpha, x_{min}} \ln(L_D(\{x_i\}_{i=1}^n)) = \max_{\alpha, x_{min}} n \ln \frac{(\alpha-1)}{x_{min}} - \alpha \sum_{i=1}^n \ln \left(\frac{x_i}{x_{min}} \right)$$

Le condizioni del primo ordine sono:

- $\frac{\delta L_D}{\delta \alpha} = \frac{n}{\hat{\alpha}-1} - \sum_{i=1}^n \ln \frac{x_i}{\hat{x}_{min}} = 0$
- $\frac{\delta L_D}{\delta x_{min}} = n \frac{\hat{\alpha}-1}{\hat{x}_{min}} = 0$,

da cui deriva:

$$\hat{\alpha}_{ML} = 1 + n \left[\sum_{i=1}^n \ln \left(\frac{x_i}{x_{min}} \right) \right]^{-1} .$$

La formula presente nel codice Maxima utilizza la notazione matriciale, dove:

$$XII = \left[\ln \left(\frac{x_1}{x_{min}} \right) \quad \dots \quad \ln \left(\frac{x_n}{x_{min}} \right) \right] \quad \text{e} \quad FII = [f_1 \quad \dots \quad f_i \quad \dots \quad f_n]$$

con:

$$\hat{\alpha}_{ML} = 1 + n [XII FII']^{-1}$$

viii metodo dei minimi quadrati

Il metodo dei minimi quadrati OLS è una tecnica di ottimizzazione che permette di trovare una funzione, detta curva di regressione, che si avvicini il più possibile ad un insieme di dati (tipicamente punti del piano). In particolare la funzione trovata deve essere quella che minimizza la somma dei quadrati delle distanze tra i dati osservati e quelli della curva che rappresenta la funzione stessa. Questo metodo va distinto da quelli per l'interpolazione, dove si richiede che la curva relativa alla funzione contenga i punti dati.

Le assunzioni OLS sono:

$$Y_i = A + B X_i + u_i$$

con:

- $E(u_i|X_i)=0$
- $(X_i, Y_i) \sim i.i.d.$
- (X_i, u_i) hanno momenti quarti finiti non nulli

Gli stimatori OLS sono:

$$\hat{B} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^n (X_i - \bar{X})^2}$$

e

$$\hat{A} = \bar{Y} - \hat{B} \bar{X} .$$