Università
Ca'Foscari
Venezia
Facoltà
di Economia

Corso di Laurea Magistrale
in Sviluppo Economico
e dell'Impresa

Prova finale di Laurea

"Approccio evolutivo e analisi delle serie storiche"

Relatore
prof. Pizzi Claudio

Laureando
Begotto Massimo
Matricola 823385

Anno Accademico
2013-2014

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

> La volontá vuole tutto sempre e di nuovo, la volontá spiega tutto. Una volontá che cessasse di volere non sarebbe piú tale.
>
> *Arthur Schopenhauer*

I problemi che coinvolgono l'ottimizzazione globale su spazi continui sono onnipresenti in in ambito scientifico [Storn R., Price K., 1997] e, piú in generale, applicazioni di questo tipo di problema possono essere esercitate in qualsiasi tipo di contesto, dalla finanza all'organizzazione logistica, dalle ricerche di mercato alla gestione di prezzi, redditi, e turni di lavoro da parte di un'azienda, e gli esempi potrebbero essere moltissimi.

Per ottimizzazione si intende la capacitá di riuscire a trovare il miglior valore possibile da assegnare ad una o piú variabili in relazione ad un criterio, il quale comunemente viene definito come una funzione da minimizzare (o massimizzare) rispettando eventuali ulteriori vincoli. L'approccio standard per la risoluzione dei problemi di ottimizzazione parte proprio dal definire una funzione obiettivo (o funzione fitness), ma in molti casi trovare l'ottimo globale di questa funzione risulta molto complesso [Berto S., 2011], come quando la ricerca avviene in uno spazio $\mathbb{R}^n$ con n molto grande, e si corre il rischio talvolta di trovare solo un punto di ottimo locale [Storn R., Price K., 1997], ottenendo dunque una soluzione che non é quella ricercata. Un esempio concreto per questo tipo di difficoltá é quello che si ha nel giá citato ambito finanziario, in cui spesso non é verosimile accettare ipotesi di normalitá della distribuzione di una data variabile, e

sono perció inutilizzabili i classici strumenti adoperati nella statistica tradizionale per la stima dei valori: basti pensare al problema di un portafoglio per il quale vanno massimizzati i rendimenti minimizzando i rischi; come ormai dimostrato [Cherubini U., Della Lunga G., 2001] non é corretto rappresentare l'aleatoritá dei rendimenti con una distribuzione normale, la quale sottovaluta la probabilitá che si verifichino eventi estremi. I rendimenti hanno una distribuzione definita leptocurtica, cioé con code molto piú pesanti, e dunque ipotizzare una distribuzione normale per risolvere questo tipo di problemi di ricerca del punto ottimo sarebbe una semplificazione fallace.

Oltre alla dimensionalitá del problema ed alla sua specificitá, un altro fattore da tenere in considerazione é il tempo a disposizione per la risoluzione di un problema di ottimizzazione [De Giovanni L.].

É stato anche per ovviare a complicazioni come quelle appena descritte che negli ultimi decenni hanno preso sempre piú piede e sono state sviluppate tecniche euristiche per l'ottimizzazione che traggono origine da concetti statistici, e che attraverso un algoritmo iterativo permettono di avvicinarsi molto alla meta, restando sempre legati alla minimizzazione di una funzione obiettivo. In particolare in questa trattazione descriveremo tre di questi metodi, che prendono spunto da processi biologici e cercano di implementarli adattandoli al contesto di ricerca; si tratta di:

- Algoritmi Evolutivi (AE), che partendo da un gruppo di possibili soluzioni le modificano in generazioni successive fino ad arrivare ad una soluzione soddisfacente;
- algoritmi della Swarm Intelligence (SI), metodologie che si ispirano in genere al comportamento di stormi di uccelli, sciami di api o colonie di formiche, i quali riescono, attraverso la condivisione delle informazioni raccolte da ogni singolo individuo, ad indirizzarsi verso la soluzione, e, iterando questo

meccanismo, a trovare finalmente la soluzione ottima per una specifica situazione, portando vantaggi all'intero gruppo.

La famiglia della prima classe di algoritmi é molto vasta, e qui prenderemo in esame il solo Differential Evolution (DE), appartenente alla prima generazione di algoritmi evolutivi.
Della seconda famiglia vedremo invece il Particle Swarm Optimization (PSO), appartenente agli Swarm Algorithms, ed il Il Fireworks Algorithm (FA), terza tecnica di cui tratteremo, anche questa classificata tra le Swarm Intelligence, che trae origine dall'immagine dei fuochi d'artificio, e dunque esula dal comportamento animale per quanto riguarda l'idea iniziale, ma per funzionamento e risultati fa parte di questa categoria.

Nel primo capitolo di questo documento introdurremo l'euristica e gli algoritmi in generale, nel secondo, terzo e quarto capitolo verrá presentata approfonditamente ognuna delle tre tecniche appena elencate, mentre nel quinto capitolo si cercherá di testare queste metodologie euristiche paragonando, quando possibile, la loro bontá nella stima dei parametri di alcuni modelli statistici rispetto alla stima effettuata con i piú classici metodi di stima.

Tutte le computazioni presenti in questo elaborato sono state effettuate con il software statistico *R* (`http://www.r-project.org/`).

Capitolo 1

Dall'euristica agli algoritmi

1.1 L'euristica

L'euristica, dal greco $\epsilon\mu\rho\acute{\iota}\sigma\kappa\omega$ (*heurisko*, letteralmente "scopro" o "trovo"), é una parte dell'epistemologia e del metodo scientifico. L'euristica é un tipo di ricerca non convenzionale, attuata con lo scopo di arrivare a conclusioni su un dato argomento, o su di una certa teoria, che andranno poi verificate, ma che si fermano, in quanto tali, al concetto di scoperte puramente empiriche, o che tutt'al piú possono aprire nuovi spazi di ricerca teorica. Per meglio dire, attraverso una ricerca euristica si ritiene avvicinarsi, con piú o meno precisione, alla risposta teorica esatta, che non siamo in grado di trovare precisamente in altra maniera, ed é per questo che la conclusione deve essere successivamente posta ad ulteriori indagini. La moderna accezione di questo termine si deve al filosofo tedesco Immanuel Kant (1724-1804) che riferendosi alle pseudoscienze, le descrive come aventi una *funzione euristica*: vale a dire che esse, seppur non appartenenti alle scienze propriamente dette a causa degli errori che presentano e le metodologie "non scientifiche" che utilizzano, non vanno comunque escluse, bensí sono da tenere in considerazione perché utili ad ampliare le conoscenze, come la metafisica che spinge l'uomo a cercare e trovare valide risposte alle proprie domande. Il procedimento euristico é dunque un approccio che non segue un percorso chiaro per accrescere la conoscenza o per generarne di nuova, ma che si affida preci-

puamente all'intuito, elaborando con esso lo stato passeggero del contesto che si viene creando attorno a chi indaga e che va mutando di istante in istante.
Ció detto appare chiara l'antitesi, rispetto all'euristica, del concetto di algoritmo, tutt'altro che guidato dall'intuito o da vie poco affidabili secondo i canoni scientifici, e anzi emblema di sistematicitá, rigorositá, organizzazione, efficienza ed affidabilitá, mediante il quale si arriva ad una soluzione dopo un numero finito di passi descritti con chiarezza. Eppure in questa tesi andremo a vedere un possibile utilizzo di tre tipi di algoritmi per problemi di ottimizzazione, procedimenti algoritmici facenti parte della piú ampia categoria delle tecniche euristiche; l'accostamento in questione puó apparire un ossimoro, ma é possibile trovare una spiegazione per questa contraddizione nei termini: in alcuni contesti non é sempre conveniente utilizzare metodi classici che danno risultati puntuali, precisi e che trovano risposte univoche, ma si ricorre a tecniche per l'appunto euristiche per avvicinarsi al risultato; nell'utilizzo peró delle suddette tecniche si fa ricorso ad algoritmi per verificare l'avvicinarsi al risultato e dare dei criteri di arresto alla ricerca. Si tratta quindi di utilizzare un procedimento algoritmico, sistematico e rigoroso, all'interno di un metodo di ricerca che si "accontenta" di arrivare nell'intorno desiderato.
I principali motivi che solitamente portano a non utilizzare metodi esatti, basati su modelli matematici, sono i seguenti [De Giovanni L.]:

- La necessitá di costruire un modello matematico ad hoc, modello che potrebbe anche non essere possibile formulare per la complessitá del problema da risolvere, o che se anche formulato puó essere difficle da maneggiare e non avere l'accuratezza ricercata;

- In alcuni casi, risolvere i problemi può richiedere tempi troppo lunghi, mentre di solito si cercano soluzioni in pochissimo tempo, addirittura in pochi istanti;

- Talvolta le soluzioni trovate con metodi esatti possono contenere comunque degli errori.

In applicazioni reali si cercano in sostanza soluzioni che approssimino il valore ignoto da noi cercato con una certa bontá ma che contemporaneamente siano raggiungibili in tempi di calcolo accettabili.
Ad ogni modo, in linea generale, affinchè un metodo sia un buon sostituto di una tecnica esatta e tradizionale, deve avere alcune caratteristiche [Storn R., Price K., 1997]:

- capacità di operare con funzioni non-differenziabili, non-lineari e multimodali;
- efficacia nel computare funzioni anche molto complesse;
- buone proprietà di convergenza (ottenere stimatori consistenti, che convergano all'ottimo in una serie di prove indipendenti);
- possibilità di lavorare contemporaneamente in punti diversi dello spazio parametrico;
- facili da utilizzare, con poche variabili di controllo ad orientare il processo.

Possiamo ora dare una breve, e probabilmente incompleta, classificazione dei metodi euristici [De Giovanni L.]:

- **Euristiche costruttive**: metodi coi quali si parte da un insieme vuoto, orientandosi a trovare almeno una soluzione ammissibile. I suoi passi fondamentali sono:
  1. *Inizializzazione* - Si sceglie un elemento di partenza per la costruzione della soluzione parziale S.
  2. *Selezione di un nuovo elemento da aggiungere alla soluzione parziale* - Si individua il criterio di selezione di un nuovo elemento da aggiungere alla soluzione parziale S.

3. *Criterio di arresto* - Se S é completa e, quindi, ammissibile, la procedura si interrompe, altrimenti si ritorna al passo 2.

- **Euristiche migliorative**: metodi con cui si cerca il miglioramento di una soluzione ammissibile, vengono detti anche **tecniche di ricerca locale**. Gli step sono i seguenti:

  1. *Inizializzazione* - Si sceglie una soluzione iniziale S di innesco del processo di ricerca.
  2. *Individuazione dell'intorno* - Si definisce una "mossa" ovvero una operazione che consente di individuare un intorno N(S) della soluzione corrente.
  3. *Scelta di una nuova soluzione* - Si individua la soluzione S' ∈ N(S) migliore dell'intorno.
  4. *Criterio di arresto* - Se S' é migliore di S si sostituisce, ponendo S=S', e si torna al passo 2; altrimenti la procedura si arresta.

- **Metodi Metaeuristici**: metodi che mixano diversi procedimenti per migliorare le soluzioni ottenibli con un metodo migliorativo, sono schemi algoritmici che lavorano a prescindere dal problema specifico che devono risolvere, facendo pochissime assunzioni su di esso o addirittura nessuna, e cercando all'interno di un ampio spazio di soluzioni candidate.

Tra le varie tecniche euristiche esistenti, ci indirizziamo ora verso gli algoritmi per l'ottimizzazione che andremo ad utilizzare.

1.2 Gli algoritmi evolutivi

Secondo quanto suggerito dalle loro classificazioni, ci sono molte varianti differenti di algoritmi evolutivi, ma l'idea di fondo comune dietro tutte queste tecniche é ovviamente la stessa: data una popolazione di individui, esiste una pressione

ambientale che provoca la cosiddetta selezione naturale (sopravvivenza del piú forte) e questo provoca un aumento dell'idoneitá della popolazione all'ambiente nelle generazioni successive. Data una funzione obiettivo da massimizzare - o minimizzare - possiamo creare in modo casuale una serie di soluzioni candidate, cioé elementi appartenenti al dominio della funzione, e applicare la funzione obiettivo come indicatore per la scelta dei candidati migliori. Sulla base di questo semplice meccanismo possiamo indirizzare le scelte dei candidati che daranno vita alla prossima generazione mediante meccanismi di *ricombinazione* e / o *mutazione* tra i candidati stessi. Questo processo puó essere ripetuto fino a quando non si arriva ad un candidato in una data generazione con qualitá sufficiente (valore della fitness accettabile), o finché un criterio di arresto precedentemente impostato viene raggiunto.

Dalla breve introduzione appena fatta si intuisce facilmente che gli Algoritmi Evolutivi, conosciuti con l'acronimo "E.A.", ed utilizzati per la risoluzione di problemi di ottimizzazione anche complessi, si reggono sulla teoria dell'evoluzione della specie, con esatto riferimento a quella esplicitata dal naturalista e geologo britannico Charles Darwin nella sua opera "Sull'origine della specie" (1859). É dunque sugli stessi principi che si basano anche gli algoritmi evolutivi, i principi di sopravvivenza degli individui piú adatti e di selezione naturale dell'ambiente circostante: questi due concetti sono legati nel contesto degli algortimi alla funzione obiettivo, come una forza che spinge in avanti la "qualitá", mentre la parte "casuale" presente della teoria darwinista, viene tradotta nelle operazioni di ricombinazione e mutazione che generano i cambiamenti osservabili tra due generazioni successive, creando le necessarie novità. Quelli appena richiamati sono i tratti piú importanti degli algoritmi evolutivi, ma ci sono anche altre caratteristiche ed assunzioni di base che vanno a dare un quadro piú preciso della teoria che sta alla base degli algoritmi evolutivi ed é per intero ispirata da Darwin; per entrare meglio nell'ottica degli E.A. ne citiamo alcune, tenendo conto che vanno

sempre pensate come riferite alla funzione fitness [Eiben A.E., Smith J.E., 2004]:

- piú gli individui di una specie possiedono fertilità elevata, piú chance avranno di portare le loro caratteristiche alle generazioni successive;
- in assenza di influenze esterne, la dimensione della popolazione di una specie rimane pressoché costante, cosí come le risorse alimentari saranno limitate, ma stabili nel tempo;
- dal momento che gli individui competono per queste risorse limitate, una lotta per la sopravvivenza ne consegue;
- alcune delle variazioni tra gli individui influenzerá la loro forma fisica e, di conseguenza, la loro capacitá di sopravvivere;
- una buona parte di queste variazioni sono ereditabili;
- gli individui meno adatti all'ambiente hanno meno probabilità di riprodursi, mentre gli individui piú adatti sopravvivono e producono prole con maggiori probabilità;
- gli individui che sopravvivono e si riproducono trasmetteranno i loro tratti alla loro prole;
- una specie cambierá e si adatterá lentamente e sempre di piú ad un dato ambiente; durante questo processo, e la conseguenza potrá essere una nuova specie.

A partire dagli anni '50 si é iniziato a pensare alla teoria evoluzionistica, con i particolari aspetti appena citati, per applicazioni in problemi che apparentemente potevano essere molto distanti: il merito é dello statistico inglese George E.P. Box (1919-2013) il quale ne introduce lo schema di base. L'applicazione combinata di variazione e selezione, concetti che presenteremo a breve, conduce in generale

a migliorare i valori di fitness in popolazioni susseguenti. É facile vedere tale processo come se l'evoluzione stesse ottimizzando, o almeno approssimando, avvicinandosi a valori sempre piú ottimali durante il suo corso. L'evoluzione é spesso vista come un processo di adattamento. Da questo punto di vista, la fitness non é vista come una funzione obiettivo da minimizzare, ma come espressione di requisiti ambientali. La corrispondenza tra questi requisiti, la loro intersecazione, comporta una maggiore "vitalità", che si riflette in un maggior numero di discendenti. Il processo evolutivo rende quindi la popolazione meglio adattabile al nuovo ambiente, al quale sopravvive con piú facilità. Va notato però che molte componenti di un processo evolutivo sono stocastiche. Durante la selezione, individui con una buona fitness hanno una maggiore probabilità di essere selezionati rispetto a quelli meno allenati, ma in genere anche gli individui deboli hanno la possibilità di sopravvivere. Nella ricombinazione la scelta di quali parti debbano essere ricombinate é casuale. Allo stesso modo per la mutazione, i pezzi che saranno mutati in una soluzione candidata, ed i nuovi pezzi che li sostituiscono, sono scelti aleatoriamente.

Di seguito, in Figura 1.1 lo pseudo-codice generico per un algortimo evolutivo, e successivamente in Figura 2.2 il suo diagramma.

```
INIZIO
 INIZIALIZZARE popolazione con candidati scelti casualmente
 VALUTARE ogni candidato
  WHILE ( condizione di arresto non soddisfatta) DO
   1 SELEZIONA   individui;
   2 RICOMBINA   coppie di individui;
   3 MUTARE      gli output creati;
   4 VALUTARE    i nuovi candidati;
  END WHILE
 FINE
```

Figura 1.1: Pseudo-codice Algoritmi Evolutivi

La funzione fitness rappresenta una valutazione euristica della qualità del candidato, ed il processo di ricerca è guidato dagli operatori variazione e selezione per mixare le informazioni di più individui. Gli algoritmi evolutivi sono basati sulla popolazione, calcolando su di un insieme di soluzioni candidate simultaneamente quale sia la migliore. A seconda del tipo di algoritmo evolutivo in gioco, i candidati possono essere rappresentati da stringhe di valori alfanumerici o semplici sequenze di caratteri (Algortimi Genetici), da vettori contenenti numeri reali (Strategie Evolutive) e così via. Lo schema di base è sempre il medesimo, la codifica cambia perchè alcune sono preferibili rispetto ad altre in quanto si prestano meglio al problema da rappresentare, ma tecnicamente la scelta di una piuttosto che dell'altra non è esiziale per lo sviluppo del processo algortimico.

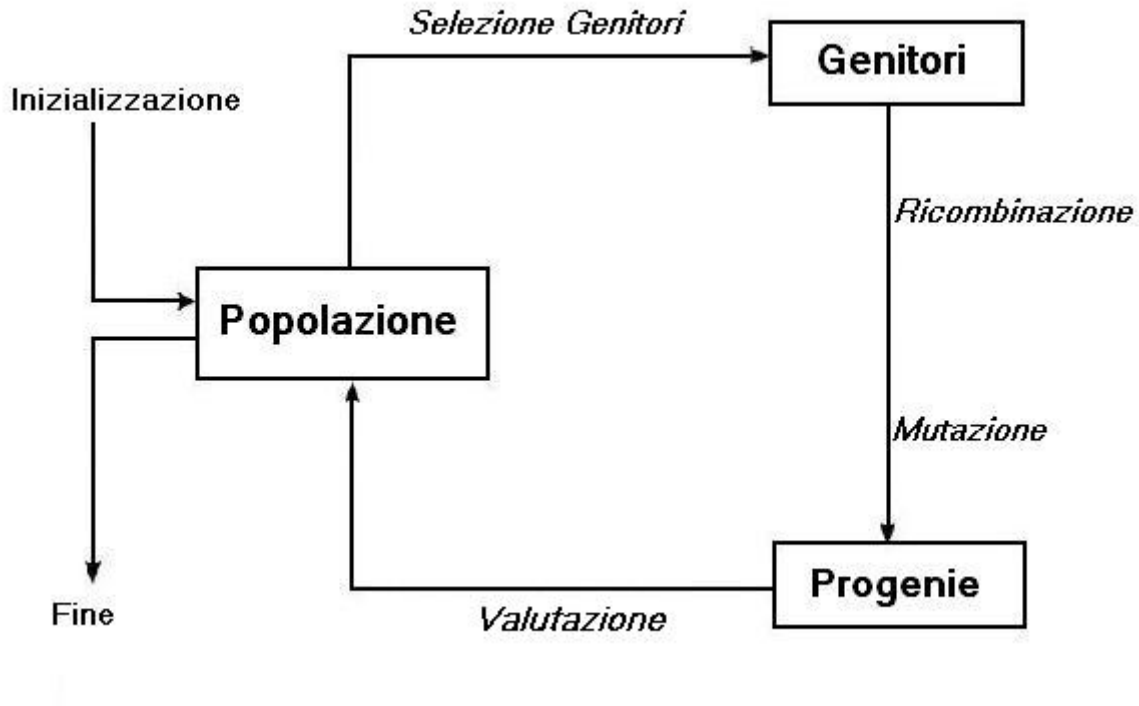

Figura 1.2: Diagramma Algoritmi Evolutivi

1.2.1 Le componenti di un Algoritmo Evolutivo

Ogni algorimo evolutivo ha come visto una struttura standard, fatta da determinate componenti che vanno specificate ogni qual volta si parla di algoritmi evolutivi per differenziarne un tipo dall'altro. Le fasi più importanti sono [Eiben A.E., Smith J.E., 2004]:

- rappresentazione (definizione degli individui);
- funzione fitness - valutazione;
- popolazione;
- meccanismo di scelta dei genitori;
- operatori ricombinazione e mutazione;
- meccanismo di selezione per la nuova generazione;

inoltre vanno sempre definite le condizioni di inizializzazione ed il criterio / i criteri di arresto dell'algoritmo.

1.2.1.1 Rappresentazione - Definizione degli individui

Il primo ineludibile passaggio è quello della individuazione dei candidati iniziali. È il momento in cui si connette il mondo reale, quello a cui appartiene il problema di ottimizzazione, al modello che abbiamo in mente per risolverlo, l'algoritmo prescelto, cercando di guidare all'interno dello spazio parametrico di risoluzione il contesto del problema. Bisogna quindi valutare quali sono gli aspetti fondamentali ed a quali requisiti o restrizioni debbano sottostare gli individui da prendere in considerazione.
Gli oggetti che formano le possibili soluzioni all'interno del contesto del problema originario sono indicati come fenotipi, mentre la loro codifica, gli individui all'interno della EA, sono chiamati genotipi. La prima fase di progettazione viene

comunemente chiamata la rappresentazione in quanto equivale a specificare una mappatura dei fenotipi su una serie di genotipi, creati appunto per rappresentare questi fenotipi. Per esempio, dato un problema di ottimizzazione sugli interi, dato insieme di numeri interi formerebbe l'insieme di fenotipi. Successivamente si potrebbe decidere di rappresentarli con il loro codice binario, quindi 18 sarebbe visto come un fenotipo e 10010 come un genotipo che lo rappresenta. È importante comprendere che lo spazio del fenotipo può essere molto diverso dallo spazio del genotipo, e che tutta la ricerca evolutiva avviene nello spazio del genotipo. Una soluzione - un buon fenotipo - è ottenuto dalla decodifica del miglior genotipo dopo la risoluzione. A tal fine, occorre rilevare che la soluzione ottimale per il problema in un fenotipo è rappresentato nel dato spazio del genotipo; altresì importante è sottolineare che la rappresentazione deve necessariamente essere invertibile: ad ogni genotipo deve corrispondere uno ed un solo fenotipo, e viceversa.

1.2.1.2 Funzione Fitness - Valutazione

Il ruolo della funzione fitness è quello di rappresentare i bisogni a cui ci si deve adattare. È l'elemento dal quale non si può prescindere e che sta alla base della selezione, facilitando di conseguenza i miglioramenti. Più nello specifico, definisce ciò che devono significare i miglioramenti, gli traccia la via. Tecnicamente è una funzione che assegna una misura di qualità al genotipo e tipicamente è composta da un valore nello spazio del fenotipo e dalla sua rappresentazione inversa. Per svolgere la sua mansione, è una funzione che va massimizzata, o minimizzata, in relazione al tipo di problema.

Nel caso di funzioni multiobiettivo [Seah C.W. et al., 2012] ci si rifà al concetto di "ottimo paretiano", valutando quindi con un valore fitness più elevato la coppia (se problema con due variabili) con la maggiore efficienza nella minimizzazione (massimizzazione), costruendo la frontiera di Pareto e valutando caso per caso.

1.2.1.3 Popolazione

Il ruolo della popolazione è quello di assumere la rappresentanza di possibili soluzioni. La popolazione è quindi un multiset di genotipi, ed è l'elemento attraverso il quale si attua l'evoluzione: gli individui infatti, visti di per sé, non sono il vero strumento, l'evoluzione è possibile solo perchè passa su larga scala, con variazioni che possono essere "testate" e dare risultati affidabili. Gli individui sono oggetti statici, è la popolazione che cambia e si adatta. Data una certa rappresentazione, definire una popolazione può voler dire anche semplicemente lo specificare quanti individui ne fanno parte, esprimendo la misura della grandezza della popolazione. In alcuni EA sofisticati invece, possono esserci spazi parametrici di più difficile definizione, con vincoli più stringenti e condizioni di relazione tra elementi da soddisfare, come determinate distanze tra individui: in questi casi è importante lavorare con accuratezza per definire la popolazione per poterla rappresentare minuziosamente, altrimenti il risultato dell'intera computazione avrebbe meno attendibilità. La diversità di una popolazione è misura del numero di differenti soluzioni presenti. Esistono molti tipi di misura della diversità all'interno di un insieme che ci indicano quanto un elemento è diverso dagli altri, basti pensare ad esempio a quanti tipi di distanze possono essere calcolate (Euclidea, Manhattan etc...). La popolazione di partenza preferibile per effettuare una ricerca con un algoritmo evolutivo deve essere il più eterogenea possibile, per cercare di coprire al meglio l'intero dominio delle soluzioni possibili: questa è una caratteristica decisiva per evitare gli ottimi locali e raggiungere l'ottimo globale.

1.2.1.4 Meccanismo di scelta dei genitori

Il ruolo della selezione dei genitori, o "mating selection", è di distinguere tra gli individui basandosi sulla loro qualità, per permettere ai migliori individui di diventare genitori delle nuove generazioni. Un individuo diventa genitore se viene

selezionato per passare alla fase di variazione. È la fase che genera miglioramenti intergenerazionali delle qualità, ed è tipicamente una fase probabilistica. La selezione seguirà i risultati delle valutazioni effettuate mediante la funzione fitness, andando a privilegiare l'individuo che meglio svolge il suo compito, quello di minimizzare o massimizzare la funzione.
Col termine privilegiare si intende che nella selezione di chi si riprodurrà, tutti avranno una probabilità di essere scelti direttamente proporzionale al valore ottenuto dalla massimizzazione (o minimizzazione) della fitness, e cioè tanto più un individuo sarà una buona soluzione tanto più alta sarà la sua probabilità di riprodursi, senza tuttavia escludere chi invece dovesse risultare un cattivo candidato; quest'ultimi però si presume verranno eliminati nel lungo periodo, con il ripetersi dei cicli dell'algoritmo che rappresentano la creazione di generazioni successive, proprio per ricalcare la teoria dell'evoluzione, permettendo all'algoritmo di avvicinarsi sempre di più all'intorno della soluzione ottimale. Come detto, anche chi non presenta le qualità richieste ha una probabilità definita, diversa da zero, di diventare genitore, ma la probabiltà assegnata a chi possiede i requisiti ottimali è senz'altro maggiore: questo per cercare di evitare che l'intera ricerca possa diventare troppo restrittiva andando a bloccarsi nell'intorno di un ottimo locale.

1.2.1.5 Gli operatori di variazione

La fondamentale funzione svolta dagli operatori di mutazione e ricombinazione è quella di creare nuovi individui da vecchi individui, e ciò corrisponde al creare nuovi candidati alla risoluzione del problema.

Mutazione Un operatore di variazione che lavora con un individuo alla volta (o una sua caratteristica) è definito *mutazione*. È applicato ad un genotipo e genera, a partire da esso, un suo mutante, una sua variazione. Un operatore di mutazione

è sempre stocastico: i suoi output dipendono da una o svariate scelte casuali, ma le variazioni sono sempre imparziali dipendendo dal caso, e non sono forzate. Ad ogni modo in alcuni tipi di algoritmi evolutivi il suo ruolo è quasi marginale, mentre in talune altre declinazioni svolge quasi da solo l'intera ricerca: è in questi casi che diventa basilare la regolazione della sua "potenza" da parte dell'utente, poichè dalla calibrazione dell'ampiezza possibile delle mutazioni (le variazioni ante e post mutazione) dipende il buon esito della ricerca.

Ricombinazione Un operatore di variazione che lavora con due o più individui simultaneamente è definito *ricombinazione* o *crossover*. Questo operatore mischia informazioni provenienti da due genitori dando vita ad uno o più figli; nell'ambiente della ricerca euristica, quindi per il ragionamento matematico, i genitori possono essere anche più di due, ma appare del tutto evidente che questo non possa avere una qualche corrispondenza in campo biologico, ed è anche per questo che casi simili sono meno diffusi di quelli "naturali".
Anche nella ricombinazione, come nella mutazione, l'aleatorietà gioca un ruolo decisivo nelle variazioni: la scelta di quali parti debbano esssere ricombinate, ed il modo in cui farlo, dipendono da estrazioni casuali. Il ruolo di questo operatore varia da algoritmo ad algoritmo, e come la mutazione in alcuni casi non ne troviamo traccia.
Il principio che sta dietro alla ricombinazione, quello di combinare le caratteristiche desiderate di due individui, è di per sé molto semplice ed ha alla sue spalle centinaia di anni di applicazioni e successi a supportare questa idea: da sempre allevatori e coltivatori hanno utilizzato ed utilizzano questo principio incrociando animali con le caratteristiche ricercate, e la stessa cosa è avvenuta e avviene per le piante. Negli algoritmi evolutivi si riprende questo concetto familiare, randomizzandolo.

1.2.1.6 Meccanismo di selezione per la nuova generazione

Il meccanismo di selezione per la nuova generazione è ancora una selezione in base alla qualità degli individui, ed è per questo molto simile alla selezione dei genitori, ma avviene in un'altra fase del ciclo algoritmico e presenta delle differenze. È chiamata in gioco nella fase dopo la generazione degli output a partire dai genitori e serve a farne una selezione: è noto che la popolazione rimane numericamente costante durante tutti i cicli, ma può avvenire che venga prodotto da una generazione di genitori un numero N di figli maggiore dei P genitori, ma per mantenere P costante nel ciclo successivo alcuni vanno scartati. La selezione in questione, e qui troviamo la fondamentale differenza dalla selezione dei genitori, avviene in maniera deterministica, cioè i meno adatti vengono scartati senza assegnazioni di probabilità in base alla fitness, e nemmeno una piccola chance viene concessa a chi è portatore delle caratteristiche meno desiderabili.
Il tipo di selezione appena descritto viene anche chiamato meccanismo di sostituzione, in quanto nel procedere delle elaborazioni è conveniente andare praticamente a sostituire nell'insieme popolazione i selezionati di questa fase, facendogli prendere il posto di un equivalente numero di individui, che possono essere anche i genitori stessi, e facendoli conseguentemente rientrare nel gruppo dei potenziali genitori della generazione che andrà ad essere prodotta. Talvolta la sostituzione può avvenire integralmente, selezionando un numero P tra i figli che va a sostituire l'intera popolazione, in altri casi invece la sostituzione avviene prendendo le indicazioni date dal valore della fitness, e dove non è conveniente per il miglioramento della soluzione, la sostituzione può avvenire in maniera molto ridotta oppure addirittura può non essere messa in pratica, ripartendo dalla popolazione precedente per formare una generazione diversa ed indirizzarsi verso i miglioramenti non ottenuti nell'ultimo ciclo.

1.2.1.7 Inizializzazione e condizioni di arresto

L'**inizializzazione** avviene solo nel primo ciclo degli EA. La popolazione iniziale viene generata casualmente, tenendo conto dei criteri specifici che incorpora il problema di ottimizzazione da risolvere, facendo sí che gli individui sottostiano alle condizioni imposte. Per alcuni algoritmi evolutivi si preferisce impostare una popolazione iniziale che abbia già buoni valori di fitness, forzandone la creazione, ma è molto più diffusa una inizializzazione del tutto casuale, lasciando il compito di avvicinarsi alla soluzione interamente all'algoritmo, e questa è evidentemente l'unica via da seguire nel caso non si abbia alcuna informazione utile riguardante i dati da computare.

Per quanto concerne le **condizioni di arresto**, possiamo fare nuovamente una distinzione tra l'eventualità in cui si abbiano informazioni sui dati, e la circostanza in cui non se ne abbiano. Nel primo caso una condizione d'arresto molto frequente è in relazione al valore della funzione obiettivo: se viene raggiunto un certo livello ritenuto accettabile, un valore che è sufficiente per ritenere il corrispondente individuo come una soluzione buona per l'ottimo globale, il processo si arresta; nella pratica degli EA però, spesso non si ha la certezza di ottenere un ottimo globale, per cui con questo criterio l'algoritmo potrebbe non avere un arresto.
Nell'eventualità in cui non si abbiano informazioni sui dati, i criteri di arresto più comuni sono i seguenti:

- raggiungimento del tempo massimo di elaborazione della CPU;
- raggiungimento di un limite dato di valutazioni della fitness;
- per un dato periodo di tempo, i miglioramenti della fitness rimangono al di sotto di una soglia fissata;

– la diversità della popolazione scende sotto una determinata soglia.

Anche la scelta del criterio di arresto influenza l'efficacia della ricerca dell'ottimo, ma questa scelta non può prescindere dal tempo a disposizione per l'ottenimento del risultato.

1.2.2 La famiglia degli Algoritmi Evolutivi

La famiglia degli algoritmi evolutivi è molto ampia e ne fanno parte svariate altre più piccole famiglie di algoritmi e metodi di risoluzione, in cui sono contenute a loro volta ulteriori tecniche euristiche. I principali sottogruppi facenti parte della famiglia degli Algoritmi Evolutivi sono [Weise T., 2009]:

- Algoritmi Genetici (GA)
- Learning Classifier System (LCS)
- Programmazione Evolutiva
- Evolution Strategies (ES)
- Programmazione Genetica (GP)

Uno dei tre algoritmi di cui tratteremo in questa tesi è il Differential Evolution, appartenente alla famiglia delle Evolution Strategies, e lo vedremo meglio nel Capitolo 2.

Un'altra importante famiglia di cui ci interessa dire qualcosa in più, legata agli Algoritmi Evolutivi pur non facendone propriamente parte, è la categoria della Swarm Intelligence.

1.3 Swarm Intelligence

La Swarm Intelligence (SI) è una categoria di ricerca euristica definita come "l'emergente intelligenza collettiva di gruppi di semplici agenti" [Bonabeau et al., 1999],

all'interno della quale l'intelligenza emergente è quella più complessa ed elevata dell'intero gruppo rispetto a quella molto più semplice dei singoli componenti. La SI è un tipo di ricerca basata sul comportamento di gruppi auto-organizzati, che non presentano un unico coordinatore, bensí vengono indirizzati dalla condivisione di informazioni raccolte da tutti i singoli componenti che interagiscono tra loro e con l'ambiente che li attornia, avendo dunque questo sistema la caratteristica di un controllo decentralizzato [Dorigo M., Birattari M., 2007]. L'espressione Swarm Intelligence, introdotta per la prima volta da Gerardo Beni, Susan Hackwood e Jing Wang nel 1988 nel contesto dei sistemi robotici cellulari, può riferirsi a sistemi sia artificiali che naturali. In effetti è una tecnica che prende spunto da un comportamento messo in atto da alcuni animali, e per carpirla è necessario il rimando all'origine biologica del concetto.

La Swarm Intelligence, traducibile come "intelligenza di sciame", è un'intelligenza collettiva, utilizzata da organizzazioni sociali come quelle presenti negli sciami di api, nei branchi di delfini, nelle colonie di formiche, negli stormi di uccelli o nelle mandrie di mammiferi per risolvere problemi che coinvolgono l'intera collettività; prendendo un singolo individuo di uno di questi gruppi, ad esempio una formica, si può dire che essa, pur dotata di una intelligenza propria, non ha la capacità di risolvere problemi complicati, tantomeno se essi sono di una dimensione che coinvolge l'intero gruppo, mentre con la condivisione di informazioni ed esperienze con tutte le altre formiche, è possibile per la colonia, nella sua totalità, muoversi nel modo migliore e più conveniente per ricercare cibo, per scappare da un pericolo potenziale o per costruire un formicaio.

Le formiche sono una testimonianza eloquente di quanto individui con intelligenza molto ridotta (come gli insetti) possano manifestare inaspettate e sorprendenti caratteristiche di gruppo, attraverso una forma di comunicazione implicita, utilizzata anche da altre specie, chiamata *stigmergia*, termine introdotto dallo zoologo francese Pierre-Paul Grassé nel 1959 col significato di "incitazione al lavoro" (stig-

mergia deriva dal greco *stigma* - traccia - ed *ergon* - lavoro).

"La stigmergia è una forma di comunicazione che avviene alterando lo stato dell'ambiente in un modo che influenzerà il comportamento degli altri individui per i quali l'ambiente stesso è uno stimolo" [Kennedy J., Eberhart R.C., 2001]: in concreto le formiche sono in grado di condividere informazioni attraverso il rilascio da parte di ognuna di loro, durante il percorso, di una sostanza chimica, il feromone, che in base alla propria intensità fornisce indicazioni alle altre formiche; questa sostanza però evapora, di conseguenza le formiche si muovono seguendo con maggiore incentivo gli itinerari più frequentati (feedback positivo), perchè essi restano segnati dai feromoni molto a lungo, tralasciando quelli con un segnale minore (feedback negativo).

La "coscienza" di gruppo appena descritta è ciò che viene definito Swarm Intelligence, una vera forma di intelligenza intangibile che deriva dalla somma delle singole, più semplici e limitate intelligenze, e che rende possibile lo svolgimento di compiti per i quali è richiesta una organizzazione collettiva di alto livello, senza peraltro che nessun tipo di direttiva arrivi da un grado gerarchico superiore, poichè nessun tipo di gerarchia è contemplata nella concezione di questa teoria.

La Swarm Intelligence è quindi una proprietà che appartiene ad alcuni gruppi, come ci indicano Beni e Watt con la definizione "proprietà di un sistema in cui il comportamento collettivo di agenti (non sofisticati) che interagiscono localmente con l'ambiente produce l'emergere di pattern funzionali globali nel sistema". Il principio fondamentale riconosciuto nei gruppi rientranti in questa definizione è l'auto-organizzazione: "un sistema auto-organizzante è un sistema che tende a migliorare le sue capacità nel corso del tempo organizzando meglio i suoi elementi per raggiungere l'obiettivo" [Klir G., 1991]. La comunità, partita spostandosi del tutto casualmente, si muove imparando dalle informazioni condivise ed avendo memoria delle azioni messe in atto in precedenza da tutti i componenti, valutando la soluzione migliore di volta in volta e reiterando questo meccanismo: esso

rappresenta l'unica struttura che governa le dinamiche ed imposta il miglioramento, sulla base dell'unica regola da seguire, quella della convenienza per tutto il collettivo. Da questo punto di vista è possibile notare la principale analogia con gli algoritmi evolutivi.
Dal contesto biologico, degli insetti in particolare, ci sono in definitiva delle caratteristiche che sintetizzano i comportamenti chiave degli animali oggetto di studio e danno ispirazione alla teoria della Swarm Intelligence per la risoluzione di problemi nei vari campi in cui viene applicata - come informatica, fisica, statistica, ingegneria elettrica, chimica - e sono alla base di tutti gli algoritmi che vengono creati all'interno delle SI [Dorigo M., Birattari M., 2007]:

- un insieme di insetti è considerato un sistema decentralizzato, senza un ente coordinatore, in cui il sistema stesso è una somma di risposte agli stimoli esterni ed interni da parte dei suoi componenti;
- gli individui sono tanti e omogenei tra loro (sono molto simili o appartengono a poche categorie);
- ogni individuo da solo non è in grado di trovare una soluzione, a differenza del collettivo;
- la stigmergia è assunta come strumento di comunicazione e permette l'interazione tra individui vicini, la comunicazione infatti avviene solo su piccola scala poichè gli individui possono esaminare solo lo spazio entro un breve raggio di distanza;
- la somma delle interazioni tra vicini è in grado di cambiare il sistema di riferimento, cambiamento che a sua volta rende diverse le interazioni e permette il progredire della colonia verso l'obiettivo;

- il sistema è visto come probabilistico, in quanto ogni individuo ha un comportamento stocastico che dipende dalla sua percezione della zona in cui agisce.

Le caratteristiche appena elencate sono i principi cardine a partire dai quali è stata concepita la teoria delle "intelligenze di sciame", applicabile per la risoluzione di problemi di svariata natura, ed ideati con la principale aspirazione di essere una affidabile alternativa nei problemi di ottimizzazione.
Dalle caratteristiche dei sistemi ispiratori si passa poi alla costruzione degli algoritmi di SI, che vengono ideati rispettando tre proprietà di modo che essi siano scalabili, paralleli e tolleranti agli errori:

1. **Scalabilità** significa che un sistema continua a funzionare pur aumentando la dimensione o la numerosità delle variabili, senza la necessità di ridefinire il modo in cui interagiscono le parti. Dato che in un sistema di Swarm Intelligence gli individui interagiscono solo con quelli più strettamente vicini, il numero di interazioni totali non tende a crescere eccessivamente con l'aumentare del numero dei componenti della popolazione, ed il comportamento degli individui è solo vagamente influenzato dalla dimensione del collettivo. La scalabilità è dunque fondamentale per questo tipo di sistemi poichè rende possibile computazioni con dimensioni maggiori, aumentando la probabilità di avvicinarsi alla soluzione, senza compromettere la velocità di analisi.

2. **Parallelismo**, cioè la possibilità per gli individui che compongono lo sciame di eseguire diverse azioni in luoghi diversi allo stesso tempo. Questa proprietà è molto utile perchè contribuisce a rendere il sistema più flessibile ed in grado di auto-organizzarsi in sottogruppi che si occupano contemporaneamente di diversi aspetti di un compito complesso.

3. La **tolleranza agli errori** è una proprietà intrinseca a questo tipo di sistemi a causa della natura decentrata ed auto-organizzante delle loro strutture di controllo. Il sistema è composto da molti individui intercambiabili e che vengono sostituiti spesso, ed assunto che nessuno di loro ha la facoltà di controllare il comportamento globale del sistema, un individuo che non è funzionale al sistema può essere facilmente rimpiazzato da uno completamente funzionale e funzionante. Questo permette di superare eventuali errori senza che essi creino particolari disagi alla computazione.

1.3.1 Swarm Intelligence ed Algoritmi Evolutivi

È stato già accennato che la categoria delle Swarm Intelligence è correlata agli Algoritmi Evolutivi, ed in effetti, per non cadere in errore, solo di relazione si può parlare, perchè secondo molte classificazioni presenti in letteratura la prima non rientra propriamente nella seconda famiglia di algoritmi, anche se secondo alcune altre classificazioni è idoneo far rientrare a pieno diritto gli algoritmi sull'intelligenza di sciame nella macro-famiglia degli Algoritmi Evolutivi.
Ad ogni modo, quale sia la classificazione più corretta non è scopo fondamentale di questa tesi, interessa piuttosto mostrare come gli argomenti siano in relazione, cogliendone opportunamente le analogie e le differenze, per verificare come gli algoritmi che andremo ad applicare per le stime parametriche abbiano delle similitudini e siano accostabili.

Le Analogie sono numerose, sia a livello teorico che pratico. Nel primo caso i punti sono sostanzialmente due: anzitutto l'ispirazione che guida la nascita di questi approcci deriva in entrambi i casi da meccanismi presenti in natura, gli EA sono guidati quasi pedissequamente dalla teoria darwinista dell'evoluzione della specie, le SI si sviluppano a partire dall'osservazione delll'intelligenza collettiva di alcuni gruppi di animali. La seconda similitudine concettuale, ma non meno

importante, è quella che riguarda l'approccio delle dinamiche di lavoro degli algoritmi, come affrontano il problema e come cercano la via verso la soluzione: entrambi i metodi partono da individui scelti più o meno casualmente, i quali rappresentano dei candidati ad essere soluzione del problema, ne verificano l'utilità testandoli e, da questo passaggio e dalle conseguenti informazioni ricavate, si indirizzano verso candidati preferibili, ottenendo miglioramenti di volta in volta ed avvicinandosi alla soluzione ottimale.

Le analogie dal punto di vista pratico si esplicitano in quelle che sono le componenti/fasi in cui si realizza lo sviluppo dell'algoritmo, le quali ricalcano inevitabilmente i principi teorici appena riepilogati sommariamente. Non sembra opportuno mettere la stessa enfasi messa per quanto riguarda gli Algoritmi Evolutivi nella descrizione di ogni singola componente delle Swarm Intelligence, in quanto il relativo approfondimento si risolverebbe in una fotocopia di quanto detto - sezione **1.2.1** e seguenti - ma basterà richiamare quali sono le componenti presenti anche nelle SI. La rappresentazione come definizione degli individui e dei criteri a cui devono sottostare i componenti del gruppo di soluzioni sulle quali si lavora è il punto di partenza in entrambi i casi, senza differenze da notare, così come per il ruolo e la definizione di popolazione a numerosità costante durante le iterazioni.

La più importante componente a rientrare tra le analogie è la funzione fitness: il suo ruolo è chiave in entrambi i meccanismi, perchè permette di assegnare un valore ad ogni candidato; il valore assegnato è il criterio principe che orienta l'intera ricerca euristica e permette di discriminare i candidati in base alla loro utilità per il miglioramento. Come accadrà nelle applicazioni che vedremo, le funzioni fitness saranno addirittura interscambiabili tra un procedimento e l'altro senza alcuna variazione.

In relazione alla funzione fitness, ci sono i procedimenti che portano alla selezione di alcuni individui od alla loro eliminazione: essa avviene come ribadito

ripetutamente in base al valore della funzione obiettivo per entrambe le teorie, ed in entrambe anche la natura della selezione è stocastica, in quanto pure nelle Swarm Intelligence si tende a non escludere nessun candidato, seppur con un valore basso della fitness. Questo accade per tenere ancora possibile la ricerca anche in spazi parametrici che in un primo momento non sembrano sede della soluzione ottimale: in poche parole è una caratteristica molto utile ad evitare di cadere in un ottimo locale, che renderebbe poco validi questi strumenti come metodi di risoluzione di qualsivoglia problema.
Infine anche l'inizializzazione ed i criteri di arresto sono praticamente identici, con la creazione iniziale casuale di individui che sottostia alle condizioni descritte dalla fase di rappresentazione del problema, ed i criteri di arresto influenzati sopratutto dal fattore tempo, cioè dalla lunghezza dell'elaborazione che solitamente si vuole entro un certo tempo limite.

Le Differenze possono anch'esse essere divise in teoriche e pratiche come fatto per le analogie. La differenza più astratta riguarda il concetto di ricerca della soluzione che guida gli algoritmi delle due categorie: gli Algoritmi Evolutivi basano il loro sviluppo sul cambiamento delle caratteristiche degli individui secondo i loro operatori di variazione, testando la prole prodotta e verificando se le variazioni effettuate a partire dai genitori hanno portato benefici. Questa è dunque una ricerca che si potrebbe definire statica, vale a dire che la ricerca avviene sfruttando i cambiamenti degli individui, a differenza di quanto avviene nelle Swarm Intelligence: la ricerca in questo caso è estremamente dinamica, avviene con veri e propri spostamenti degli individui all'interno dello spazio di ricerca, del dominio. È una effettiva ricerca empirica, un "inseguimento" pratico e concreto della soluzione attraverso la partecipazione di tutti gli individui che condividono le informazioni derivanti dai loro spostamenti e dalle successive conseguenze.

Se uno spostamento appare vantaggioso sarà tutto il gruppo ad andare in quella direzione perchè si intuisce che la soluzione è verso quell'intorno, mentre, pur tenendo conto dei segnali che danno le nuove generazioni - i quali sono le stesse indicazioni sulla direzione ottenute negli SI - negli EA la ricerca sfrutta gli accoppiamenti tra individui che vanno a sostituire la precedente generazione solo se più vicini alla soluzione, e che danno quindi vita ad individui che incrociandosi a loro volta si avvicineranno. Il movimento appare dunque più "rigido".
La seconda differenza, riferita alla struttura e perciò pratica, riguarda il modo con cui cambiano gli individui, gli operatori di variazione: come visto sono ricombinazione e mutazione, prettamente stocastici, a dare le direttive per questo passaggio fondamentale negli EA, mentre è differente il fenomeno nelle SI.
Nelle SI l'evoluzione dello spostamento si basa sugli spostamenti stessi: gli algoritmi tengono memoria di quanto ottenuto dalle nuove posizioni ottenute dalle generazioni appena create, creando le successive sulla base dei risultati verificati in ultima istanza. Semplificando, nel suo processo l'algoritmo si limita a registrare i risultati di tutte le posizioni ed agire di conseguenza, trasmettendo quanto appreso alle nuove generazioni. Gli spostamenti sono creati da specifiche relazioni tra le posizioni degli altri individui a seconda della caratteristica funzione di ogni algoritmo, che riflette il particolare contesto naturale o artificale di riferimento; inoltre a seconda del tipo di algoritmo le posizioni meno favorevoli trovate dagli individui possono influenzare con molto o poco peso le future posizioni, ma usualmente non vengono totalmente scartate.

Da questo conciso ragionamento su analogie e differenze, il risultato è che sono le prime ad essere numericamente maggiori, con quasi tutte le componenti ed alcune analogie sul modus operandi. Questo dunque, cercando di risolvere il dubbio sulla classificazione delle Swarm Intelligence come parte o meno degli

Algoritmi Evolutivi presentato in precedenza, può indurci a ritenere condivisibile la posizione di chi ritiene le gli EA comprendere al suo interno le SI. D'altra parte però è pur vero che andando a vedere quali siano le differenze, ci si rende conto di quanto esse coinvolgano punti fondamentali, riguardando l'idea di ricerca ed una componente decisiva. Detto ciò il dubbio sulla classificazione rimane intatto, ed abbiamo forse toccato le motivazioni che mantengono in vita il dualismo tra le due posizioni.

1.3.2 La famiglia delle Swarm Intelligence

Pur non chiarendo a quale famiglia appartengano le Swarm Intelligence, ma avendo sottolineato una sicura relazione con gli Algoritmi Evolutivi, si possono elencare quali siano gli algoritmi di ricerca appartenenti a questa classe, divenuti numerosissimi negli ultimi anni, specialmente dal 2005 in poi:

- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)
- Flock of Starlings Optimization (FSO)
- Artificial Bee Colony algorithm (ABC)
- The Bees Algorithm
- Artificial Immune Systems (AIS)
- Grey Wolf Optimizer (GWO)
- Bat Algorithm (BA)
- Gravitational Search Algorithm (GSA)
- Altruism algorithm
- Glowworm Swarm Optimization (GSO)

- Intelligent Water Drops (IWD)
- River Formation Dynamics (RFD)
- Self-Propelled Particles (SPP)
- Stochastic Diffusion Search (SDS)
- Multi-swarm optimization

Come si può notare dai loro nomi, la quasi totalità di questi algoritmi prende spunto da una particolare specie appartenente al mondo animale o da fenomeni presenti in natura. Gli algoritmi maggiormente utilizzati e sviluppati sono i primi tre dell'elenco, Particle Swarm Optimization, dal movimento di sciami d'api o banchi di pesci, Ant Colony Optimization, dal comportamento delle formiche, e Flock of Starlings Optimization che deriva dal volo di stormi di uccelli.
In questa tesi ci occuperemo nel Capitolo 3 del PSO per poi utilizzarlo in alcune ricerche.
Una tecnica più recente del PSO e che prende anch'essa ispirazione dalla Swarm Intelligence, sfruttando il movimento di un gruppo, è quella denominata Fireworks Algorithm (FA), ispirata dalle esplosioni dei fuochi d'artificio. È il terzo ed ultimo algoritmo che verrà utilizzato e valutato per risolvere problemi di ottimizzazione in questo documento e sarà presentato nel Capitolo 4.

Capitolo 2

Differential Evolution - DE

2.1 Introduzione

Il Differential Evolution è un metodo di ricerca matematica per l'ottimizzazione di funzioni multidimensionali, appartenente alla famiglia delle Strategie Evolutive. La nascita di questo algortimo stocastico si deve a Ken Price e Rainer Storm i quali lo idearono nel tentativo di risolvere un poliniomio di Čebyšëv. L'idea originale fu quella di utilizzare un vettore di differenze per perturbare il vettore popolazione, con il fine anzitutto che il procedimento fosse il più facile possibile da usare. Da quel momento, nel 1996, molte versioni del DE sono state implementate, in quanto grazie alla versatilità nell'utilizzo di questo metodo euristico, numerosi ricercatori lo hanno maneggiato apportandone migliorie nel corso del tempo. Nei primi anni di vita, il DE è stato testato anche in competizioni come l'IEEE's International Contest on Evolutionary Optimization (ICEO) nel 1996 e 1997 raccogliendo da subito un ottimo riscontro, ed è stato inserito come ottimizzatore in piattaforme come Mathematica e software come R; sue applicazioni sono molto utilizzate in ambito finanziario per la gestione di portafogli, ma oltre a questo contesto sono svariati gli ambiti in cui è conveniente sfruttarne il procedimento [Ardia D. et al., 2011].

Appartenendo agli EA, il DE è ispirato da processi biologici, ed infatti ricalca lo schema di base degli Algoritmi Evolutivi: a partire da una popolazione di

individui scelti casualmente e rientranti nei criteri suggeriti dall'impostazione del problema - il dominio - DE risolve i problemi di ottimizzazione evolvendo la popolazione di soluzioni candidate usando operatori di variazione quali mutazione e crossover (ricombinazione), creando ciclo dopo ciclo nuove generazioni, tenendo sempre in considerazione il principale scopo, vale a dire che le nuove popolazioni abbiano individui che migliorino il risultato della funzione obiettivo da minimizzare (o massimizzare).

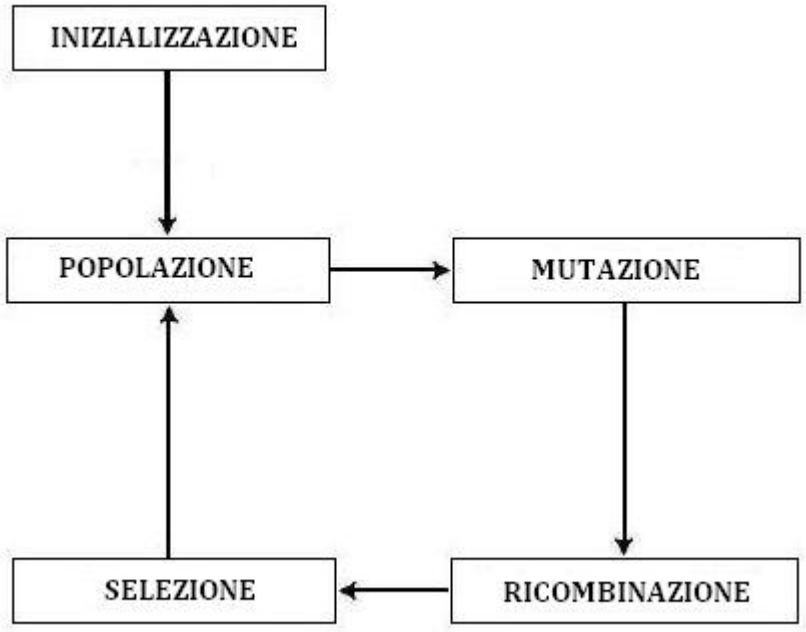

Figura 2.1: Sintesi algoritmo del Differential Evolution

2.2 Il DE ed il suo funzionamento

Il DE è un metodo che utilizza NP vettori parametrici di dimensione D. Sia

$$x_{i,G} \quad \text{con } i = 1, 2, 3, \ldots, NP \tag{2.1}$$

il vettore popolazione per ogni generazione G, ed NP non varia durante il processo algoritmico. Il vettore popolazione iniziale è scelto casualmente all'interno

dello spazio parametrico, e per la scelta randomizzata dei valori verrà utilizzata una variabile uniforme.

Il DE produce, a partire dalla popolazione iniziale, nuovi vettori parametrici sommando la differenza pesata tra due vettori-individui della popolazione ad un terzo vettore. L'operazione appena descritta è la cosiddetta *mutazione* del DE. I parametri mutati del vettore sono poi mixati con quelli di un altro vettore predeterminato appartenente alla popolazione, il vettore target, per generare il cosiddetto vettore trial: questa seconda operazione è definita *ricombinazione* o *crossover*. A questo punto si passa alla verifica per la fase detta *selezione*: vengono confrontati i valori della fitness rispettivamente del vettore target e del vettore trial, se il primo ha un valore migliore viene mantenuto all'interno della popolazione anche per la generazione successiva, viceversa il vettore trial andrà a prendere il posto del vettore target.

Le fasi appena elencate vengono ripetute in ogni ciclo per ogni componente della popolazione, di conseguenza verranno ripetute NP volte.

Le fasi del DE possono ora essere viste più nel dettaglio.

2.2.1 Mutazione

Per ogni vettore target $x_{i,G}$ con $i = 1, 2, 3, \ldots, NP$, viene generato un vettore mutante, secondo questa equazione

$$v_{i,G+1} = x_{r1,G} + F \cdot (x_{r2,G} - x_{r3,G}) \tag{2.2}$$

dove gli indici randomizzati r_1, r_2 ed $r_3 \in \{1, 2, 3, \ldots, NP\}$ sono numeri interi e mutualmente differenti, con $F > 0$; r_1, r_2 ed r_3 devono essere anche differenti da i, e ciò comporta la necessità di avere $NP \geq 4$. Il fattore F è una costante $\in [0, 2]$ che controlla il contributo della differenza $(x_{r2,G} - x_{r3,G})$ e della variazione della

mutazione di conseguenza. Un esempio bidimensionale che mostra la mutazione con la differenza pesata e la generazione di $v_{i,G+1}$ è visibile nella Figura 2.2.

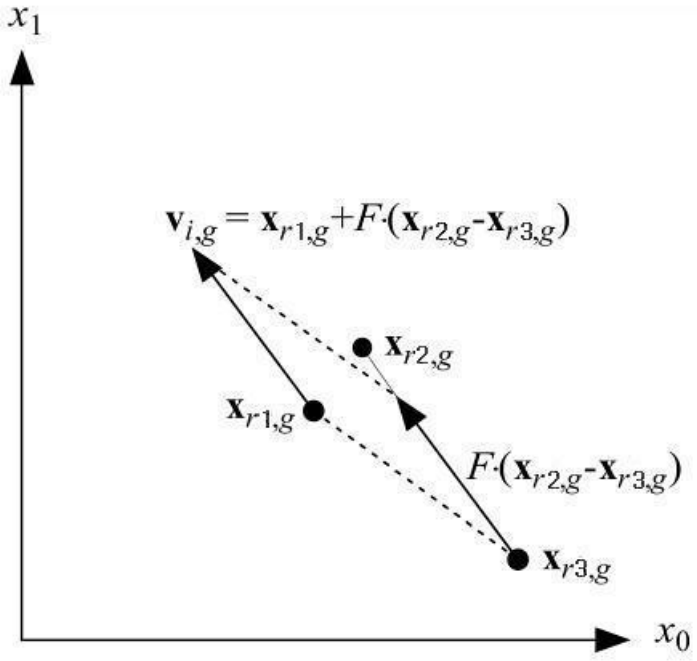

Figura 2.2: La generazione di $v_{i,G+1}$ a partire da $x_{r1,G}$ e dalla differenza pesata con F tra $x_{r2,G}$ e $x_{r3,G}$

2.2.2 Crossover

Per aumentare la diversità che si ottiene attraverso la mutazione nei parametri dei vettori, viene introdotto anche l'operatore crossover.

Nella fase di crossover si crea il trial vector:

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1}) \tag{2.3}$$

formato in questo modo

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{se} \quad (randb(j) \leq CR) \quad \text{o} \quad j = rnbr(i) \\ x_{ji,G} & \text{se} \quad (randb(j) > CR) \quad \text{e} \quad j \neq rnbr(i) \end{cases} \tag{2.4}$$

$$j = 1, 2, \ldots, D$$

Nella (2.4), *randb*(j) è la j-esima estrazione casuale di un numero appartenente all'intervallo [0,1]; CR è la costante di crossover, anch'essa appartenente all'intervallo [0,1], che deve però essere impostata dall'utente utilizzatore del DE; *rnbr*(i) è un indice estratto casualmente tra i numeri 1, 2, . . . , D che serve ad assicurare che $u_{i,G+1}$ prenda almeno un parametro da $v_{i,G+1}$.
In Figura 2.3 viene descritta schematicamente l'operazione del crossover nel caso di un vettore di parametri con $D = 7$.

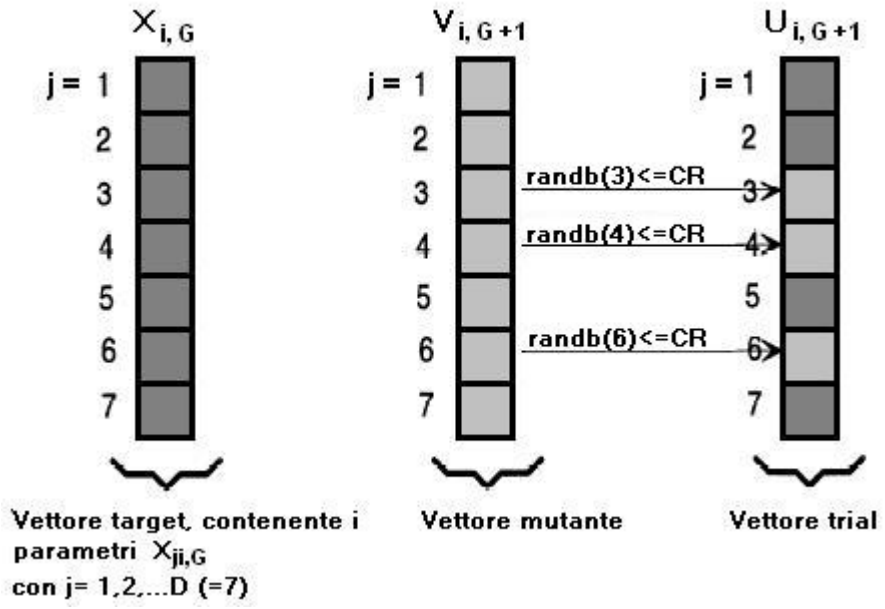

Figura 2.3: Esempio di crossover per un vettore con $D = 7$

2.2.3 Selezione

Dopo le operazioni di mutazione e crossover viene effettuata la selezione, che avviene sulla base di quanto indicato dal valore della funzione fitness. Decidere se il vettore trial $u_{i,G+1}$ entrerà a far parte della generazione G+1 o se verrà scartato, dipende unicamente dal confronto di $u_{i,G+1}$ con $x_{i,G}$: se col nuovo vettore la funzione fitness ottiene un valore migliore di quanto ottenuto con $x_{i,G}$, il vettore

trial entrerà nella popolazione $G + 1$, altrimenti il vettore $x_{i,G}$ non verrà sostituito e resterà nella generazione successiva.

Nella Figura 2.4 viene riepilogato graficamente un intero ciclo di DE, che viene eseguito per ogni individuo appartenente alla popolazione G ed in seguito per ogni generazione successiva fino a raggiungere uno dei criteri od il singolo criterio di arresto.

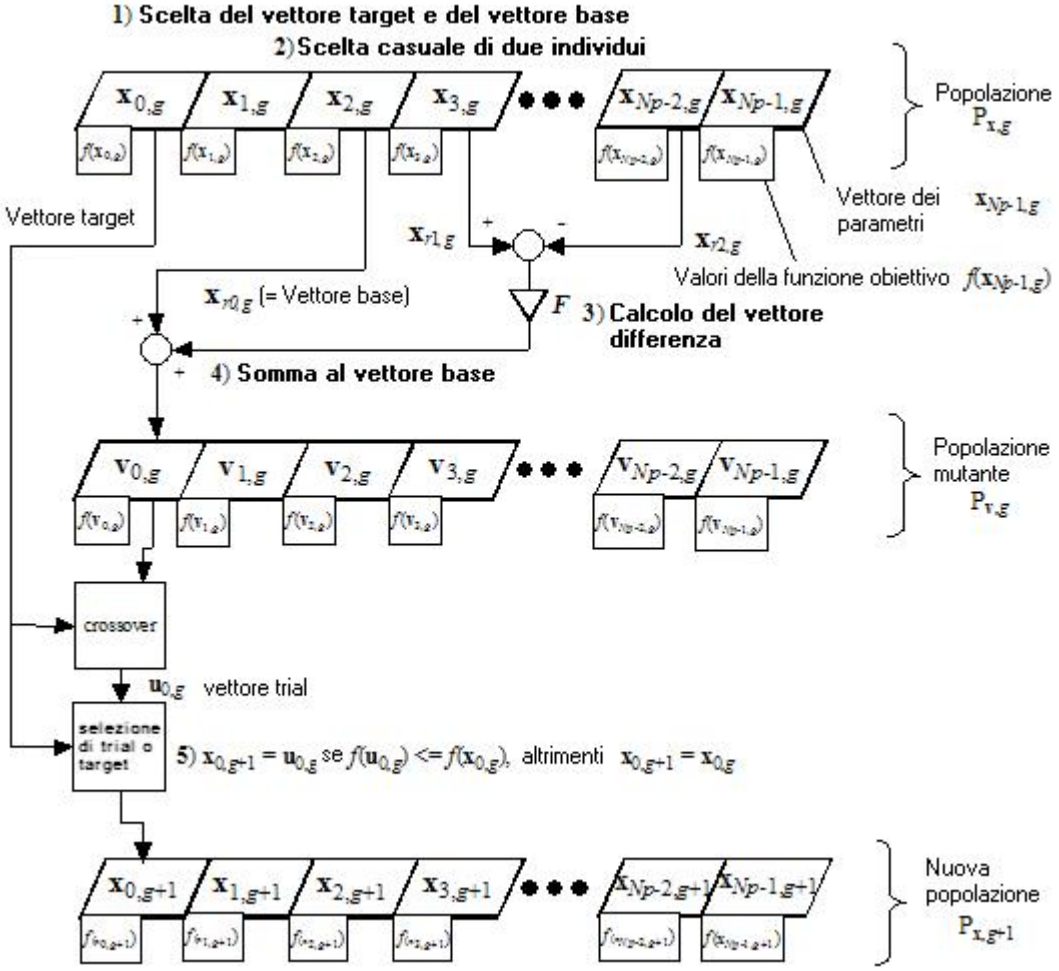

Figura 2.4: Schema completo di un ciclo dell'algoritmo del Differential Evolution

2.3 Varianti del DE

La struttura e le componenti presentate finora appartengono alla versione base del Differential Evolution, caratterizzata dal tipo di mutazione che abbiamo visto;

la suddetta viene indicata con la notazione $DE \setminus rand \setminus 1$. Esistono però molte altre versioni, che si discostano dalla $DE \setminus rand \setminus 1$ per una diversa conformazione della mutazione, la quale non avviene sempre con la 2.2.
Nella tabella seguente un riepilogo delle possibili versioni e rispettive mutazioni:

Notazione	**Formula della mutazione**
$DE \setminus rand \setminus 1$	$v_{i,G+1} = x_{r1,G} + F \cdot (x_{r2,G} - x_{r3,G})$
$DE \setminus rand \setminus 2$	$v_{i,G+1} = x_{r1,G} + F \cdot (x_{r2,G} - x_{r3,G}) + F \cdot (x_{r4,G} - x_{r5,G})$
$DE \setminus best \setminus 1$	$v_{i,G+1} = x_{Best,G} + F \cdot (x_{r1,G} - x_{r2,G})$
$DE \setminus best \setminus 2$	$v_{i,G+1} = x_{Best,G} + F \cdot (x_{r1,G} - x_{r2,G}) + F \cdot (x_{r3,G} - x_{r4,G})$
$DE \setminus current - to - best \setminus 1$	$v_{i,G+1} = x_{i,G} + F \cdot (x_{Best,G} - x_{i,G}) + F \cdot (x_{r1,G} - x_{r2,G})$
$DE \setminus current - to - best \setminus 2$	$v_{i,G+1} = x_{i,G} + F \cdot (x_{Best,G} - x_{i,G}) + F \cdot (x_{r1,G} - x_{r2,G}) + F \cdot (x_{r3,G} - x_{r4,G})$

Tabella 2.1: Varianti del DE e relative mutazioni.

Nelle ricerche che verranno effettuate, utilizzeremo sia la $DE \setminus rand \setminus 1$ che la $DE \setminus best \setminus 2$, nella quale $x_{Best,G}$ indica il vettore che nella generazione G ha ottenuto il miglior valore della fitness.

2.4 Utilizzo del DE con R

È stato già accennato che tutte le computazioni di questa tesi che vedremo nei successivi capitoli sono state effettuate col software statistico R. All'interno di questo software esiste un pacchetto grazie al quale è possibile utilizzare l'algoritmo del Differential Evolution, ed è il DEoptim ad implementarlo. Nelle analisi svolte però si è preferito creare l'algoritmo ex-novo senza ricorrere al DEoptim, per studiarne meglio il funzionamento ed avere un maggiore dimistichezza nel controllo delle funzionalità dell'algoritmo; un ulteriore vantaggio derivante dalla creazione di un algoritmo che esula da quello fornito dal software è il fatto che è stato possibile indirizzarsi con più decisione al tipo di modelli per i quali il DE è

stato applicato.

In figura 2.5 viene proposto lo pseudo-codice dell'algoritmo del DE.

```
INIZIO
FISSA PARAMETRI
   Fissa valore di F ∈[0,2]
   Fissa valore di CR ∈[0,1]
   Fissa valore di NP
INIZIALIZZA ALGORITMO
    Contatore g=0
    Crea NP individui casuali
        WHILE (criterio di arresto)
            FOR(i in 1:NP)
                \\MUTAZIONE//
                scelta casuale di 3 vettori x1 , x2, x3
                v = x1 + F (x2 - x3)
                \\CROSSOVER//
                generazione casuale di rnbr
                  FOR (z in 1:D)
                    generazione casuale randb(0,1)
                    IF  randb<=CR o z = rnbr
                     u_i = v_i
                    else
                     u_i = x_i
                    end IF
                  end FOR
                \\SELEZIONE//
                IF  f(u_i)< f(x_i)
                  x_{i,g+1} = u_i
                else
                  x_{i,g+1} = x_{i,g}
                end IF
            end FOR
              g=g+1
        end WHILE
FINE
```

Figura 2.5: Pseudo-codice del DE

Lo pseudo-codice riportato in Figura 2.5 si riferisce alla *DE* \ *rand* \ 1 mentre per l'altra versione del DE utilizzata, la *DE* \ *best* \ 2, ci sarebbero naturalmente delle differenze nella fase della mutazione.

Capitolo 3

Particle Swarm Optimization - PSO

3.1 Introduzione

Il Particle Swarm Optimization è un algoritmo in grado di ottimizzare un problema non lineare e multidimensionale, che raggiunge in genere buone soluzioni pur richiedendo una parametrizzazione minima [Pereira G., 2011].
L'algoritmo ed il concetto di Particle Swarm Optimization sono stati introdotti da James Kennedy e Russel Ebhart nel 1995. Tuttavia, le sue origini sono meno recenti, in quanto il principio di base di ottimizzazione attraverso il "comportamento di sciame" si ispira a precedenti tentativi di riprodurre i comportamenti osservati di animali nel loro habitat naturale, come stormi di uccelli o banchi di pesci, e, quindi, in ultima analisi, le sue origini sono la natura stessa. Queste radici nei processi naturali di gruppi di animali portano alla classificazione dell'algoritmo come appartenente alle Swarm Intelligence.
Il concetto di base dell'algoritmo è quello di creare uno sciame di particelle che si muovono nello spazio intorno a loro (spazio parametrico) alla ricerca del loro obiettivo o del luogo che meglio si adatta alle loro esigenze, indicate da una funzione di fitness. La metafora del comportamento degli uccelli è presto detta: uno stormo di uccelli vola all'interno del proprio ambiente, cercando il posto migliore per riposare; il posto migliore può essere una combinazione di caratteristiche come uno spazio sufficiente per tutto lo stormo, la facilità di accesso al

cibo, all'acqua o qualsiasi altra risorsa rilevante.

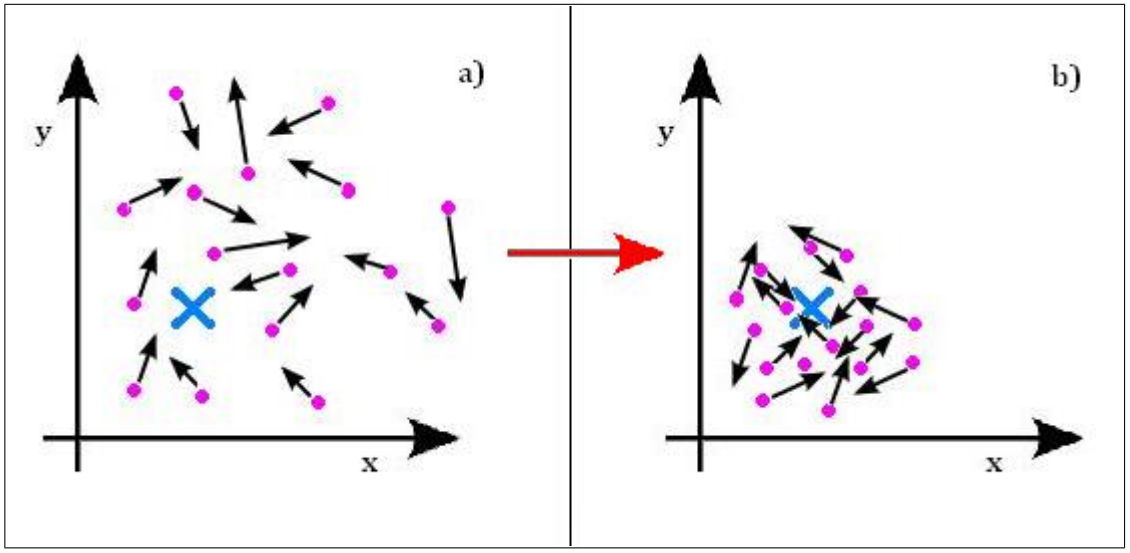

Figura 3.1: Esempio bidimensionale del movimento delle particelle: nella *a*) inizia il movimento, che via via tende a far spostare le particelle verso la posizione ottima all'interno dello spazio, come nella *b*).

Prendendo come base il movimento dei pesci o di sciami di api, riprodotto dalle particelle come in Figura 3.1, ci sono tre idee principali che stanno dietro le proprietà di ottimizzazione del PSO [Pereira G., 2011]:

1. una singola particella, che rappresenta una possibile soluzione del problema, può determinare quanto buona sia la propria posizione attuale. La misura della qualità della propria posizione viene data mettendo in relazione con quanto esplorato dalla particella in questione, ma anche con quanto "conosciuto" dalle altre particelle, grazie alla condivisione delle informazioni;

2. ogni particella ha "memoria" dei propri stati nelle posizioni precedenti, ed anche di quelli delle particelle vicine;

3. nella velocità che fa muovere ogni particella all'interno della regione di esplorazione, esiste un fattore stocastico che le muove anche verso spazi non esplorati.

Questa struttura, combinata con una buona distribuzione iniziale dello sciame, consente una vasta esplorazione dello spazio parametrico e dà una probabilità molto alta di trovare le migliori soluzioni in modo efficiente.
Dopo l'introduzione del PSO diverse varianti sono state ideate, ma il concetto di fondo rimane comunque il medesimo: una volta che lo spazio in cui risolvere il problema è stato definito, un insieme di particelle viene generato in esso, e, mentre lo esplora, le loro posizioni e velocità sono aggiornati iterativamente secondo l'algoritmo specifico.
Oltre al modo in cui vengono aggiornate posizioni e velocità delle particelle, un'altra variabile in gioco che cambia da applicazione ad applicazione del PSO è la funzione fitness. Come in qualsiasi altro caso di ricerca euristica, le specifiche di questa funzione dipendono dal problema da ottimizzare, soprattutto per quanto riguarda le sue dimensioni D.
Anche se il PSO ha dimostrato di essere un algoritmo efficiente e con buoni risultati, la sua struttura non garantisce che la soluzione migliore venga trovata, dato che si basa sull'esplorazione dello spazio, ma questa d'altronde è una caratteristica intrinseca delle tecniche euristiche, o metaeuristiche in questo caso, sebbene esse offrano una buona certezza sul fatto che ci si avvicini all'ottimo globale senza fermarsi ad uno locale.

3.2 Il PSO ed il suo funzionamento

Il PSO nasce cercando di imitare un comportamento molto complesso, ma da esso trae le più semplici indicazioni possibili, ed infatti è di per sé un algoritmo dal facile utilizzo. Esistono sostanzialmente due componenti che generano gli spostamenti: velocità e posizione.
Trattandosi di problemi multi-dimensionali, genericamente si ha il vettore velocità $v_i = (v_{i,0}, v_{i,1}, \ldots, v_{i,D})$ ed il vettore posizione $x_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,D})$. Essi sono calcolati come segue

$$x_{i,d}(G+1) = x_{i,d}(G) + v_{i,d}(G+1) \tag{3.1}$$

e

$$\begin{aligned} v_{i,d}(G+1) = \; & v_{i,d}(G) \\ + \; & C_1 \cdot Rnd(0,1) \cdot [pb_{i,d}(G) - x_{i,d}(G)] \\ + \; & C_2 \cdot Rnd(0,1) \cdot [gb_d(G) - x_{i,d}(G)] \end{aligned} \tag{3.2}$$

in cui abbiamo:

i *indice delle particelle*;

d *dimensione considerata, ogni particella ha una posizione ed una velocità per ogni dimensione*;

G *generazione, cioè il ciclo dell'algoritmo*;

$x_{i,d}$ *posizione della particella i nella dimensione d*;

$v_{i,d}$ *velocità della particella i nella dimensione d*;

C_1 *costante di accelerazione per la componente cognitiva*;

$pb_{i,d}$ *la locazione nella dimensione d con il miglior valore della fitness tra tutte quelle visitate in questa dimensione dalla particella i*;

C_2 *costante di accelerazione per la componente sociale*;

gb_d *la locazione nella dimensione d con il miglior valore di fitness tra tutte le posizioni visitate in quella dimensione da tutte le particelle.*

Dall'equazione 3.1 abbiamo che la posizione alla nuova generazione per ogni particella sarà la somma tra la posizione attuale e la velocità. La velocità a sua volta si calcola con la 3.2, che comprende diverse componenti: nella prima riga abbiamo l'ultima velocità registrata per quella particella; alla seconda riga c'è la componente detta cognitiva, che tiene conto della distanza tra la miglior posizione

trovata autonomamente dalla particella e la sua posizione attuale; infine la terza riga è la componente sociale, che guarda invece alla distanza tra posizione attuale e la migliore trovata tra tutte quelle esplorate dai componenti dello sciame.
Come ampiamente discusso nella sezione sulle Swarm Intelligence (1.3), nella parte introduttiva di questo capitolo e da quanto si vede nelle equazioni, ribadiamo ora che uno dei punti di forza del PSO è la capacità degli individui di condividere le informazioni, ed il fatto che conseguentemente il comportamento di ognuno di essi venga influenzato da quanto esplorato dagli altri componenti della popolazione. Nel caso specifico della forma originale del PSO, è il miglior punto trovato in generale dal gruppo ad influenzare gli spostamenti degli individui, oltre che la miglior posizione trovata singolarmente da ognuno di essi, poichè sarà con altissima probabilità il più vicino alla soluzione del problema: l'ottimo globale.
Le equazioni 3.1 e 3.2 sono auto-aggiornanti essendo ricorsive, e permettono un miglioramento progressivo delle posizioni di tutte le particelle: proprio a causa della ricorsività, un'altro aspetto importante è l'inizializzazione delle particelle della prima iterazione, dato che da esse dipende inevitabilmente ogni generazione successiva ed hanno un grande impatto sulla buona riuscita della ricerca. Normalmente le posizioni iniziali vengono estratte da una variabile uniforme, di modo che esse coprano al meglio lo spazio parametrico. Le velocità vengono inizializzate casualmente, con la cura di settarle ad un valore basso perchè sono generalmente più adeguate per evitare grandi spostamenti iniziali.

In Figura 3.2 vediamo l'algoritmo con il suo diagramma di flusso. Inizialmente oltre alle velocità ed alle posizioni vanno inizializzate anche le costanti per la parte sociale e cognitiva, che non variano mai durante l'esecuzione della ricerca, ed hanno anch'esse un grosso peso specifico, controllando quanto pesino negli spostamenti le componenti cognitiva e sociale. Per la corretta implemen-

tazione dell'algoritmo è basilare aggiornare prima le velocità delle particelle, e successivamente le loro posizioni.

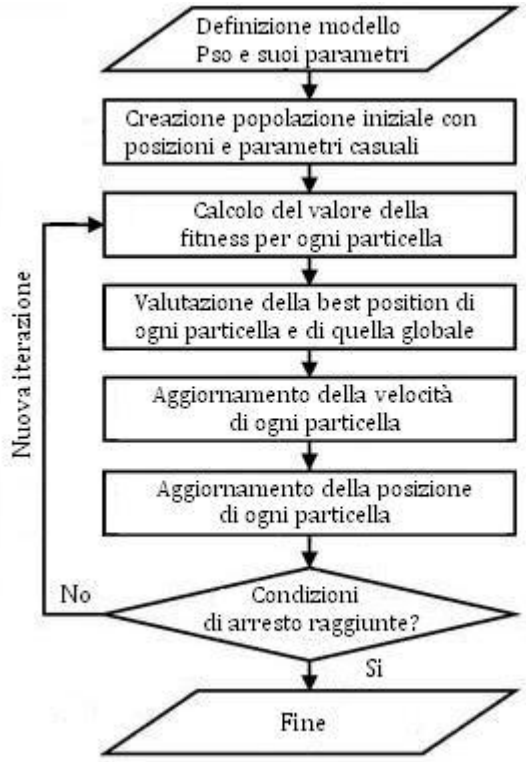

Figura 3.2: Diagramma di flusso dell'algoritmo del PSO.

3.3 Varianti del PSO

La versione appena descritta è quella originale del PSO, definita del *Global Best*, ma esistono molte varianti che cercano di migliorarlo. Noi non utilizzeremo la versione originale, che pur ha sempre presentato buoni risultati, bensì una delle due varianti presentate di seguito. Le varianti si possono basare su come vengono inizializzate velocità e posizioni (ad esempio le velocità iniziali possono essere poste uguali a zero anzichè essere inizializzate casualmente) oppure su quanto venga influenzata ogni particella dalla miglior posizione globale rispetto a quella delle particelle vicine.

Le due varianti principali sono il *Local Best* e l' *Inertia Weight*.

Local Best è una variante nella quale, a differenza della versione base, si riduce ad una zona più ristretta lo scambio di informazioni. Le particelle infatti vengono divise in sotto-gruppi, e la condivisione di quale sia la miglior posizione avviene solo tra loro. Gli spostamenti sono quindi maggiormente influenzati dal "vicinato" rispetto a quanto avviene nel *Global Best*. Ciò comporta che la convergenza all'ottimo globale è più lenta, ma il lato positivo è che la copertura dello spazio di ricerca avviene con maggiore efficacia, riducendo la probabilità di restare in un ottimo locale, ed aumentando quella di arrivare all'ottimo globale. La scelta tra le due varianti dipende quindi dal tempo a disposizione, poichè in termini di costi si ha una perdita solo da questo punto di vista a fronte però di una ricerca più affidabile. La suddivisione avviene perlopiù in due modi: statisticamente, guardando all'indice della particella, o sulla base delle distanze.

Inertia Weigth è una variante che mira a bilanciare due possibili tendenze del PSO, quella di sfruttare l'intorno di soluzioni note e quella di esplorare nuove aree dello spazio di ricerca. A tale scopo essa si focalizza sulla prima componente $v_{i,d}(G)$ della (3.2), che tiene memoria della velocità precedente, moltiplicandola per una costante **w**. Questa è la versione del PSO che verrà utilizzata nei calcoli. L'equazione (3.2) diventa quindi

$$\begin{aligned} v_{i,d}(G+1) = \quad & \mathbf{w} \cdot v_{i,d}(G) \\ + \quad & C_1 \cdot Rnd(0,1) \cdot [pb_{i,d}(G) - x_{i,d}(G)] \\ + \quad & C_2 \cdot Rnd(0,1) \cdot [gb_d(G) - x_{i,d}(G)] \end{aligned} \tag{3.3}$$

Si noti che togliendo questa componente il movimento sarebbe costante attorno ad un punto, unico candidato come soluzione. L'idea dell' *Inertia Weight* si traduce nel dare un peso bilanciato all'inerzia del movimento, moltiplicando la componente per una costante **w**, cercando di dare più importanza all'esplorazione di nuove aree di quanto non accada con la versione base del PSO. Per

fare questo c'è la necessità di contrastare il movimento precedente, spingendo le particelle in nuove direzioni, con la moltiplicazione di $v_{i,d}(G)$ per la costante.

3.4 Utilizzo del PSO con R

Algoritmo 1 Inizializzazione

FOR (i in 1:S)
  FOR (d in 1:D)
    \\INIZIALIZZA POSIZIONE E VELOCITA' DI OGNI PARTICELLA//
    $x_{i,d} = Rnd(x_{min}, x_{max})$
    $v_{i,d} = Rnd(-v_{max}/3, v_{max}/3)$
  end FOR

  \\INIZIALIZZA LA MIGLIOR POSIZIONE DI OGNI PARTICELLA//
  $pb_i = x_i$
  \\AGGIORNA LA MIGLIOR POSIZIONE GLOBALE//
  IF $f(pb_i) < f(gb)$
    $gb = pb_i$
  end IF
end FOR

Algoritmo 2 Particle Swarm Optimization (Global Best)

\\INIZIALIZZA TUTTE LE PARTICELLE//
$it = 1$
WHILE (criterio di arresto)
  FOR (i in 1:S)
    \\AGGIORNA LA MIGLIOR POSIZIONE DI OGNI PARTICELLA//
    IF $f(x_i) < f(pb_i)$
      $pb_i = x_i$
    end IF
    \\AGGIORNA LA MIGLIOR POSIZIONE GLOBALE//
    IF $f(pb_i) < f(gb)$
      $gb = pb_i$
    end IF
  end FOR

  \\AGGIORNA POSIZIONE E VELOCITA' DI OGNI PARTICELLA//
  FOR (i in 1:S)
    FOR (d in 1:D)
      $v_{i,d} = v_{i,d} + C_1 * Rnd(0,1) * [pb_{i,d} - x_{i,d}] + C_2 * Rnd(0,1) * [gb_d - x_{i,d}]$
      $x_{i,d} = x_{i,d} + v_{i,d}$
    end FOR
  end FOR
  $it = it + 1$
end WHILE

Figura 3.3: Pseudocodice per il PSO (Global Best), in cui S è il numero delle particelle e D il numero delle loro dimensioni.

Anche per il PSO, come pure per il Differential Evolution, esiste un pacchetto

che ne implementa l'algoritmo in R. Nella fattispecie si tratta del pacchetto PSOptim. Anche per questo metodo però, per lo svolgimento delle analisi raccolte nella seguente tesi, l'algoritmo è stato creato ex-novo senza l'utilizzo del pacchetto disponibile. Nella Figura 3.3 lo pseudocodice per ricostruirlo.

Capitolo 4

Fireworks Algorithm - FA

4.1 Introduzione

Il Fireworks Algorithm (FA) è un algoritmo basato sul concetto di Swarm Intelligence, di recente sviluppato a partire dalla simulazione del processo di esplosione dei fuochi d'artificio, proseguendo su di un meccanismo già tracciato da PSO e dalle altre SI. In analogia con i fuochi d'artificio che esplodono illuminando il cielo notturno, le particelle del FA sono rilasciate nel potenziale spazio di ricerca e per ogni fuoco d'artificio è avviato un processo di esplosione ed una pioggia di scintille riempie lo spazio locale intorno ad esso. I fuochi d'artificio, vale a dire le posizioni di partenza, così come le scintille appena generate, rappresentano potenziali soluzioni nello spazio di ricerca. Similmente ad altri algoritmi di ottimizzazione, l'obiettivo è trovare una buona soluzione (idealmente l'ottimo globale) per $min_{x \in \Omega} f(x)$ dove $f : \mathbb{R}^N \rightarrow \mathbb{R}$ è una funzione non-lineare.
Il FA, introdotto per la prima volta nel 2010 da Ying Tan e Yuanchun Zhu, presenta un nuovo metodo di ricerca dentro lo spazio potenziale mediante un processo stocastico di esplosione all'interno di uno spazio locale. Questa tecnica di ricerca metaeuristica funziona come segue: inizialmente, N fuochi artificiali vengono inizializzati in modo casuale, e la loro qualità (cioè il valore fitness) viene valutata per determinare l'ampiezza dell'esplosione e il numero di scintille per ogni fuoco. Successivamente, i fuochi d'artificio esplodono e generano diversi tipi di scintille

all'interno del loro spazio locale. Infine, N fireworks candidati sono selezionati tra il gruppo di candidati, che comprende le scintille appena generate, nonchè gli N fuochi originali [Tan Y., Zhu Y., 2010].

Al fine di garantire la diversità e bilanciare la ricerca globale e locale, l'ampiezza

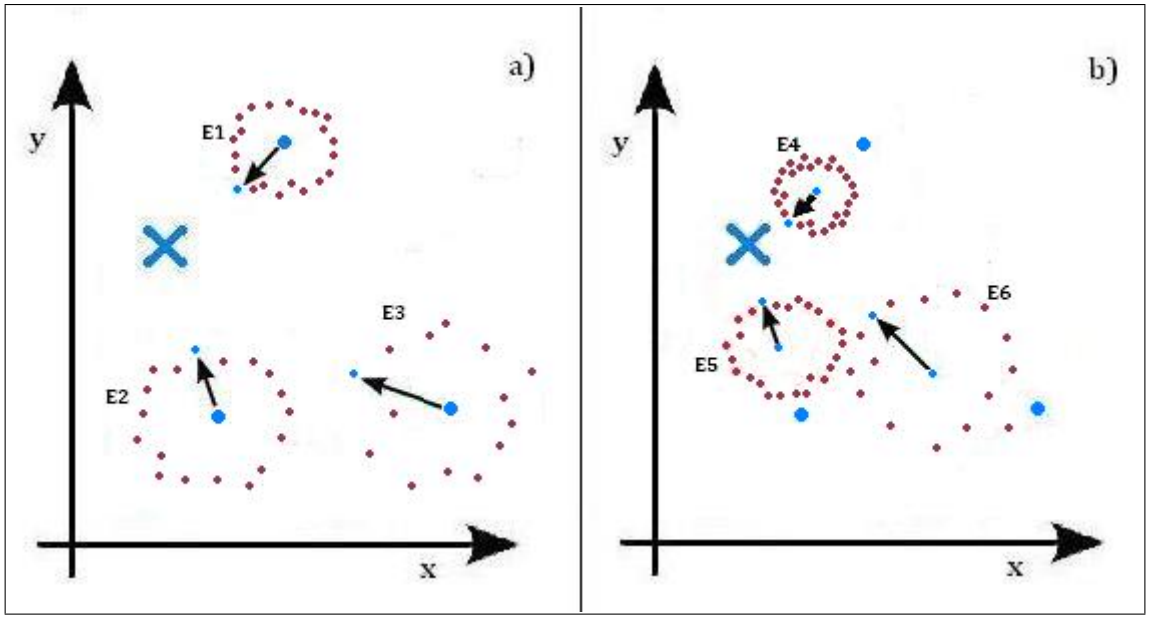

Figura 4.1: Esemplificazione bidimensionale della ricerca spaziale che avviene tramite il FA dell'ottimo globale: nella *a*) avvengono le prime tre esplosioni dagli $N = 3$ fuochi d'artificio, nella *b*) avvengono le successive tre, con punto di partenza la miglior posizione trovata con l'esplosione precedente.

dell'esplosione e la numerosità delle scintille esplose variano tra i fuochi d'artificio. Un fuoco iniziale con una migliore idoneità - giudicata tramite la funzione fitness - può generare un numero maggiore di scintille all'interno di un intervallo più piccolo, cioè, con un'esplosione che avviene con minore ampiezza. Al contrario, fuochi d'artificio con minore idoneità possono generare solo una popolazione più piccola in un raggio più ampio. Nella Figura 4.1 viene esemplificata una esplosione su un spazio bidimensionale, dando l'idea di quanto possa accadere con una ricerca tramite il Fireworks Algorithm: in questo caso gli n fuochi di partenza sono tre, con tre differenti esplosioni. Nell'esplosione E1 vediamo una maggiore numerosità di scintille prodotte all'interno di un'area più ristretta rispetto alle altre due esplosioni E2 ed E3, e questo perchè la E1 è più vicina

all'ottimo globale indicato in figura. Nella figura 4.1 *b*) si osservano le successive esplosioni: n è costante e rimane pari a tre, prendendo per ognuna delle tre esplosioni precedenti la scintilla con il migliore valore della fitness. Si noti come via via che ci si avvicina all'ottimo globale la numerosità delle esplosioni cresce e contemporaneamente sono le ampiezze delle E4, E5 ed E6 a decrescere rispetto alle relative esplosioni precedenti.

Questa tecnica permette di bilanciare tra le capacità di esplorazione e sfruttamento (*exploration* ed *exploitation*) dell'algoritmo: l'esplorazione si riferisce alla capacità dell'algoritmo di esplorare varie regioni dello spazio di ricerca al fine di individuare aree con buone soluzioni potenziali, lo sfruttamento si riferisce alla capacità di condurre una ricerca approfondita in un'area più piccola già riconosciuta come potenzialmente promettente al fine di trovare la soluzione ottimale. L'esplorazione è attuata da quei fuochi d'artificio che hanno una grande ampiezza di esplosione (quindi con fitness inferiore), in quanto hanno la capacità di "fuggire" dai minimi locali. Lo sfruttamento si ottiene da quei fuochi artificiali che hanno una piccola ampiezza d'esplosione (cioè alta fitness), in quanto è più forte la capacità di ricerca locale in settori promettenti.

È inoltre di fondamentale importanza la fase che si verifica dopo questa prima esplosione, nella quale viene generato anche un altro tipo di scintille su base del tutto casuale che nulla ha a che fare col valore della fitness e dunque nemmeno con la posizione all'interno dello spazio di ricerca. L' idea di fondo di questo passaggio è quella di garantire ulteriormente la diversità delle particelle create, al fine di migliorare l'efficacia della ricerca.

4.2 Il FA ed il suo funzionamento

Come nei casi già affrontati del DE e del PSO, gli algoritmi creati per essere un'alternativa nei problemi di ottimizzazione sono modellati per essere molto semplici. È così anche per il FA, il cui diagramma di flusso è visibile in Figura

4.2. La ricerca si effettua nell'intorno di ognuno degli n fuochi d'artificio, a partire

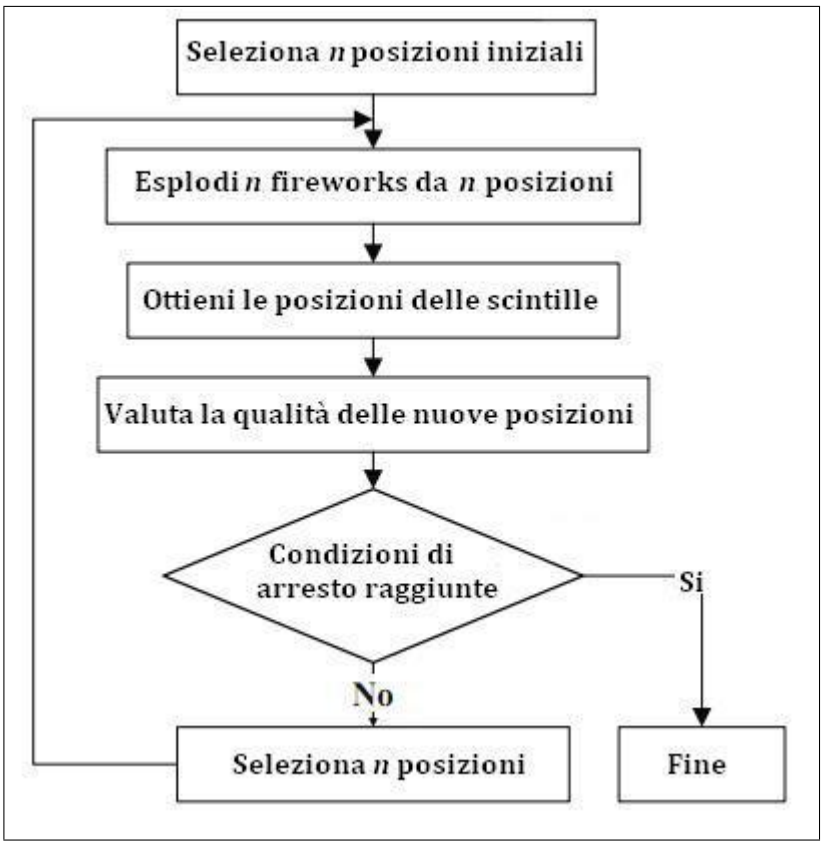

Figura 4.2: Diagramma di flusso del FA.

dai quali viene fatta partire una pioggia di scintille che cadono entro un certo raggio, come in Figura 4.1. Successivamente una seconda esplosione crea un altro gruppo di scintille attorno al fuoco. A questo punto si ottiene una generazione di possibili candidati di numerosità non costante, tra i quali si scelgono gli n migliori candidati dai quali far ripartire un successivo ciclo di doppie esplosioni; una volta raggiunte le condizioni di arresto, la ricerca finisce. Le condizioni di arresto possono essere incentrate sia sulla qualità della soluzione che sul numero di iterazioni completate.

4.2.1 Esplosioni del primo tipo

Le esplosioni del primo tipo sono le prime generate dai fuochi d'artificio, e variano in base al valore di fitness ottenuto dalla posizione in cui essi si trovano, cioè,

traducendo in termini di ricerca all'interno dello spazio parametrico, variano in relazione alla loro distanza dall'ottimo globale.
Dato un problema di minimizzazione del tipo

$$Min \;\; f(x) \;\; \in \mathbb{R},\; x_{min} \leq x \leq x_{max}$$

dove $x = x_1, x_2, x_3, \ldots, x_d$ denota la posizione all'interno dello spazio di ricerca, $f(x)$ è la funzione obiettivo (funzione fitness), ed x_{min} ed x_{max} sono i limiti inferiore e superiore dello spazio potenziale, le due variabili che cambiano in base alla posizione sono il numero di scintille create e l'ampiezza delle esplosioni.

Numero di scintille Il numero di scintille generate da ogni fuoco d'artificio x_i è definito come segue

$$s_i = m \cdot \frac{y_{max} - f(x_i) + \xi}{\sum_{i=1}^{n}(y_{max} - f(x_i)) + \xi} \tag{4.1}$$

dove m è un parametro che controlla il numero totale di scintille generate dagli n fuochi d'artificio ed è impostato dall'utente, $y_{max} = max(f(x_i))$ con $i = 1, 2, 3, \ldots, n$ è il valore massimo (quindi il peggiore) della funzione obiettivo, ottenuto tra tutti gli n fuochi, mentre ξ è una costante piccolissima, tendente allo zero, utilizzata per evitare che il denominatore sia zero.
Per non avere problemi di numerosità di s_i e non incorrere in situazioni che rallenterebbero la ricerca, come quelli derivanti da un numero troppo elevato di scintille, vengono definiti anche i limiti di s_i

$$\hat{s}_i = \begin{cases} a \cdot m & \text{se } s_i < a \cdot m \\ b \cdot m & \text{se } s_i > b \cdot m,\; a < b < 1 \\ s_i & \text{altrimenti} \end{cases} \tag{4.2}$$

dove a e b sono costanti.

La differenza di numero di scintille è verificabile dalla 4.1, infatti se il denominatore per il calcolo di tutte le s_i corrispondenti ad ogni x_i rimane costante, il numeratore sarà più grande per quelle posizioni dei fuochi aventi un valore della funzione fitness $f(x_i)$ migliore, cioè più basso, con la conseguenza di ottenere un s_i più elevato.

Ampiezza delle esplosioni Al contrario del numero di scintille, l'ampiezza dell'esplosione sarà più elevata a partire da una posizione peggiore rispetto a quella che parte da una buona posizione. L'ampiezza dell'esplosione è definita come segue

$$A_i = \hat{A} \cdot \frac{f(x_i) - y_{min} + \xi}{\sum_{i=1}^{n}(f(x_i) - y_{min}) + \xi} \tag{4.3}$$

dove $\hat{A}$ denota l'ampiezza massima dell'esplosione, impostata dall'utente, e $y_{min} = max(f(x_i))$ con $i = 1, 2, 3, \ldots, n$ è il minimo valore (quindi il migliore) della funzione fitness ottenuto tra tutti gli n fuochi d'artificio.
Come si nota, facendo il ragionamento inverso rispetto alla numerosità delle scintille, avremo un denominatore sempre costante, ma il numeratore sarà più grande stavolta per le posizioni con una fitness peggiore, aumentandone la relativa ampiezza rispetto a quella più bassa delle posizioni migliori.
Il problema che si cerca di risolvere con gli algoritmi euristici può essere ad una dimensione o multidimensionale. Nel caso del FA per ottenere maggiore randomizzazione nelle esplosioni, non tutte le dimensioni vengono coinvolte. Prendendo un generico problema d-dimensionale, viene calcolato z secondo la (4.4)

$$z = d \cdot rand(0, 1) \tag{4.4}$$

in cui d è la dimensionalità di x_i, mentre $rand(0, 1)$ è una variabile uniforme

distribuita nell'intervallo [0,1]. Successivamente vengono scelte casualmente le z dimensioni tra le d esistenti: parlando da un punto di vista geometrico, solo per le z dimensioni estratte avverrà uno spostamento all'interno dello spazio di ricerca, quindi le esplosioni coinvolgeranno solo alcune dimensioni. Lo spostamento h per ognuna delle dimensioni z viene calcolato tenendo conto dell'ampiezza A_i relativa alla posizione del fuoco x_i

$$h = A_i \cdot rand(-1, 1) \tag{4.5}$$

in cui $rand(-1, 1)$ è una variabile uniforme distribuita nell'intervallo [-1,1]. Questo ulteriore fattore permette di variare l'ampiezza calcolata per ogni posizione da dimensione a dimensione dello stesso fuoco d'artificio. Nella Figura 4.3 vediamo un esempio bidimensionale di esplosione con e senza il fattore $rand(-1, 1)$.

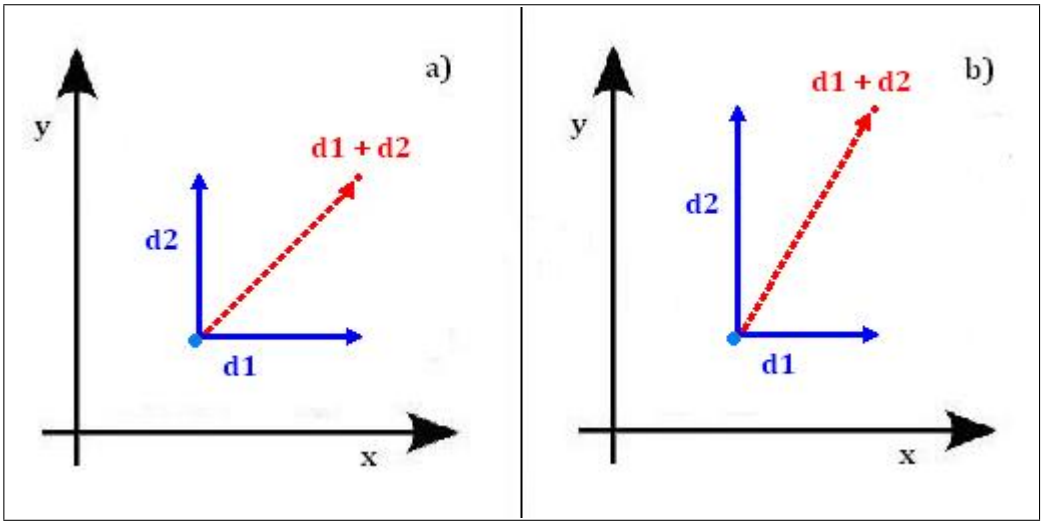

Figura 4.3: Esempio bidimensionale di spostamento in seguito all'esplosione. Nella *a*) la scintilla si muove sulle due dimensioni con la stessa ampiezza A_i senza il fattore $rand(-1, 1)$. Nella *b*) cambia l'ampiezza dello spostamento tra le due dimensioni grazie al fattore $rand(-1, 1)$.

Se l'ampiezza non cambiasse tra dimensione e dimensione avremmo, nel caso bidimensionale, delle esplosioni del primo tipo che formano una circonferenza perfetta, poichè il raggio non varierebbe mai. Col fattore $rand(-1, 1)$ a moltiplicare l'ampiezza su ogni dimensione, la ricerca è in grado di ottenere posizioni differenti

per esplorare meglio lo spazio di ricerca.

Definita quindi con $\tilde{x}_k^j$ la posizione nello spazio della j-esima dimensione, tra quelle selezionate, del fuoco d'artificio x_k, lo spostamento avverrà ricorsivamente in questo modo

$$\tilde{x}_{k+1}^j = \tilde{x}_k^j + h \tag{4.6}$$

e così per ognuna delle z dimensioni del fuoco selezionate precedentemente.

Se in seguito agli spostamenti la posizione dovesse uscire dai limiti imposti dal tipo di problema, la posizione ottenuta sarebbe modificata per farla rientrare nello spazio parametrico.

4.2.2 Esplosioni del secondo tipo

Allo scopo di aumentare la variabilità tra le scintille generate, vengono fatte esplodere dagli n fuochi d'artificio anche delle ulteriori scintille. Questa volta il meccanismo è più semplice, non essendo influenzato dalla posizione e richiedendo solo un passaggio. Il numero di scintille generate è impostato dall'utente, col parametro $\hat{m}$. L' ampiezza invece è casuale come vedremo. Anche in questo caso, come nelle esplosioni del primo tipo, non tutte le dimensioni vengono coinvolte. Esattamente allo stesso modo, viene impostato un valore di z tramite la (4.4) e successivamente z dimensioni vengono scelte tra le d del fuoco.

Definita con $\hat{x}_k^j$ la posizione nello spazio della j-esima dimensione, tra quelle selezionate, del fuoco d'artificio x_k, lo spostamento avverrà in questo modo

$$\hat{x}_{k+1}^j = \hat{x}_k^j \cdot G \tag{4.7}$$

in cui G è un coefficiente estratto da una variabile Gaussiana (1,1). La (4.7) viene ripetuta per ognuna delle dimensioni per un numero $\hat{m}$ di volte.

Infine, alla stregua del procedimento precedente, se in seguito agli spostamenti

la posizione dovesse uscire dai limiti imposti dal tipo di problema, la posizione ottenuta sarebbe modificata per farla rientrare nello spazio parametrico.

4.2.3 Selezione delle scintille

Alla fine di ogni doppio ciclo di esplosioni, nel caso in cui le condizioni di arresto non si fossero verificate, vanno sempre selezionate le nuove posizioni, cioè i nuovi fuochi d'artificio, da cui far partire le successive esplosioni. Il numero dei fuochi d'artificio è sempre costante ed è uguale ad n. Tra tutte le scintille generate, l'unica ad essere sicuramente utilizzata nel ciclo susseguente è quella corrispondente alla migliore posizione $\mathbf{x}^\star$, nella quale la funzione obiettivo $f(\mathbf{x}^\star)$ ha il miglior valore tra tutte le posizioni trovate. Rimangono quindi altre $n-1$ posizioni da selezionare, e questo procedimento avviene in base alla distanza tra ogni posizione e le altre. In generale, la distanza tra una posizione x_i e le altre è definita come segue:

$$R(x_i) = \sum_{j \in K} d(x_i, x_j) = \sum_{j \in K} \|x_i - x_j\| \tag{4.8}$$

in cui K è l'insieme di tutte le posizioni, sia fuochi che scintille.
Ciò detto, la probabilità di una posizione x_i di essere selezionata è calcolata così:

$$p(x_i) = \frac{R(x_i)}{\sum_{j \in K} R(x_j)} \tag{4.9}$$

Nel calcolo della distanza può essere utilizzato qualsiasi tipo di metodo come la distanza di Manhattan, la distanza Euclidea, la distanza Euclidea ponderata, la distanza di Mahalanobis etc...

4.3 L'utilizzo del FA con R

Per l'utilizzo del FA con R non è possibile utilizzare alcun pacchetto a differenza di quanto detto per DE e PSO, in quanto, data la recente nascita di questo algoritmo e la non vastissima diffusione, non ne è stato ancora implementato nessuno. È

stato necessario di conseguenza implementare in R l'ottimizzatore a partire dal seguente pseudo-codice.

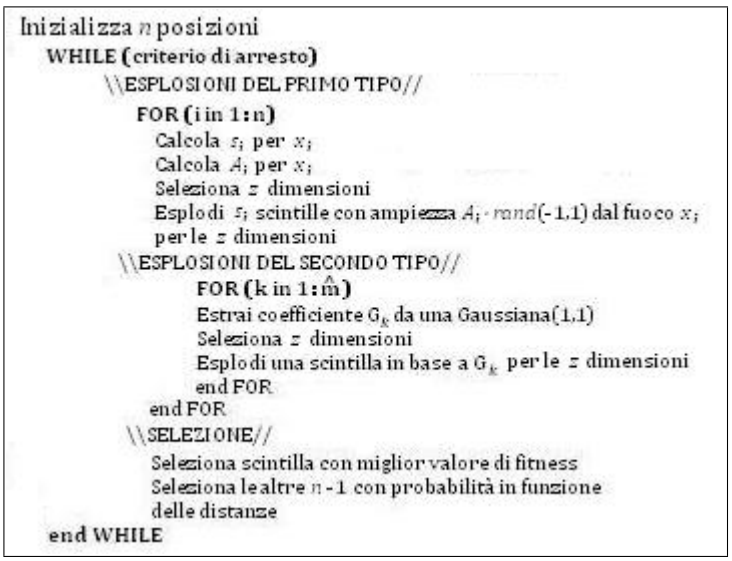

Figura 4.4: Pseudo-codice del FA.

Capitolo 5

L'applicazione degli algoritmi

Dopo aver presentato i tre algoritmi ed il loro funzionamento possiamo ora vederli applicati ad un caso concreto, per la stima dei parametri di un modello autoregressivo.

I modelli autoregressivi vengono impiegati nella descrizione di serie temporali. In un generico processo autoregressivo $AR(p)$ di ordine p, il valore y_t all'istante t è descritto dai valore agli istanti precedenti $(t-1, t-2, \ldots, t-n)$ moltiplicati per determinati parametri Φ_p. L'equazione di un generico $AR(p)$ che descrive la serie storica $\{y_t\}_{t=1}^{n}$ si presenta come la seguente

$$y_t = \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \Phi_3 y_{t-3} + \cdots + \Phi_p y_{t-p} + \epsilon_t, \quad t = 1, \ldots, n \tag{5.1}$$

in cui ϵ_t è la parte stocastica, un processo *white noise* a media nulla e varianza costante σ^2, spesso ipotizzato gaussiano.

Una volta individuato il tipo di modello capace di spiegare l'andamento della serie storica, l'unico problema è quello di riuscire ad individuare i parametri con una stima. Nel caso dell' $AR(p)$ si tratta dunque di stimare i Φ_p parametri: è questo lo scopo per il quale vengono applicati gli algoritmi evolutivi come metodo alternativo. In particolare, il funzionamento dei tre algoritmi presentati, permetterà di "ipotizzare" dei parametri-soluzione, verificando la loro idoneità e andando poi a migliorarli con iterazioni successive. L'idoneità di cui sopra non è altro che il valore della fitness. Dalla (5.1) è possibile ricavare la funzione

obiettivo da minimizzare alla quale sarà legata la ricerca algoritmica. L'obiettivo è minimizzare la componente *white noise*, vale a dire i residui, gli elementi di disturbo rispetto alla relazione tra y_t e gli istanti precenti y_{t-i} con $i = 1, 2, \ldots, p$. La funzione da minimizzare sarà perciò

$$\epsilon_t = y_t - \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \Phi_3 y_{t-3} + \cdots + \Phi_p y_{t-p} \tag{5.2}$$

L'obiettivo degli algoritmi sarà quello di trovare i parametri coi quali si ottiene il più piccolo valore possibile di $\sum_{t=1}^{n} \epsilon_t^2$ tra tutti quelli generati.
Quanto ottenuto verrà confrontato con la stima dei parametri ottenuta da R attraverso la risoluzione del sistema di equazioni normali di Yule-Walker, un tipo ricorsivo di risoluzione per i modelli autoregressivi $AR(p)$ che stima i Φ_p a partire dai coefficienti di autocorrelazione.

Dopo un primo confronto numerico sulle stime ottenute, si passerà ad utilizzare il metodo bootstrap, cercando di approssimare la distribuzione delle stime per calcolarne gli intervalli di confidenza.

5.1 Cenni sul metodo bootstrap

Il metodo *bootstrap*, ideato nel 1979 da Bradley Efron, è utilizzato per stimare caratteristiche della distribuzione di probabilità di uno stimatore, quando questa è sconosciuta. Questa tecnica permette di stimare la distribuzione campionaria di quasi qualsiasi statistica con metodi molto semplici. Generalmente, rientra nella classe più ampia dei metodi di ricampionamento: la pratica del metodo infatti si basa su di un campione osservato di dati, il quale viene ricampionato un numero sufficientemente elevato di volte per ottenere risultati affidabili.
Il metodo bootstrap prevede che da una popolazione iniziale, considerata come una variabile casuale distribuita secondo una funzione $F(x, \theta)$, venga estratto con o senza reinserimento un campione inziale $X = (X_1, \ldots, X_{CC})$ di grandezza CC, e

da questo venga stimato il parametro θ mediante $\hat{\theta} = \hat{\theta}(X_1, \ldots, X_{CC})$. Dal campione iniziale vengono estratti con reinserimento altri campioni $X_b^\star = (X_{b1}^\star, \ldots, X_{bCC}^\star)$ un numero di volte B abbastanza elevato, e vengono calcolate sui campioni estratti le stime di θ con lo stesso stimatore $\hat{\theta}_b^\star = \hat{\theta}(X_b^\star)$. Se questo procedimento viene ripetuto un numero sufficiente di volte, si otterrà una buona approssimazione dell'universo di tutte le possibili estrazioni dal campione iniziale. Così facendo sarà possibile fare inferenza sulla caratteristica di interesse della popolazione universo di partenza da cui è stato estratto il campione iniziale $X = (X_1, \ldots, X_{CC})$, e successivamente gli altri campioni bootstrap $X_b^\star = (X_{b1}^\star, \ldots, X_{bCC}^\star)$, considerando la distribuzione delle stime $\hat{\theta}$ ottenute in seguito ai ricampionamenti [Morana M.T., Porcu M.].

Nel nostro caso, il metodo bootstrap verrà utilizzato come segue: inizialmente sarà simulata una serie storica, sulla quale si stimerà con un algoritmo il parametro, o i parametri per il caso AR(2); quindi si calcoleranno i residui del parametro stimato. L'insieme dei residui rappresenterà il campione iniziale appena descritto. Dal campione iniziale si estrarranno con reinserimento 1000 campioni bootstrap di residui dai quali si ricostruiranno 1000 serie bootstrap, tutte sempre con lo stesso parametro calcolato alla prima applicazione. Su ognuna delle serie ricostruite verranno applicati gli algoritmi, i quali produrranno 1000 stime dei parametri. Fatto ciò, ritenendo una ripetizione di 1000 volte un numero sufficientemente elevato, si guarderà la distribuzione delle stime, costruendo degli intervalli di confidenza.

5.2 Applicazione con un AR(1)

La prima applicazione degli algoritmi viene effettuata per la stima del parametro Φ_1 di un modello $AR(1)$. Data una serie storica di lunghezza n, la funzione che ne

descrive l'andamento nel caso di un $AR(1)$ è

$$y_t = \Phi_1 y_{t-1} + \epsilon_t \qquad t = 1, 2, \ldots, n \tag{5.3}$$

Dalla (5.3) va poi definita la specifica funzione fitness per l'applicazione degli algoritmi nel caso di un AR(1). I residui si calcolano con la (5.4) per ogni parametro

$$\epsilon_t = y_t - \Phi_1 y_{t-1} \qquad t = 1, 2, \ldots, n \tag{5.4}$$

Dunque il valore della fitness si otterrà minimizzando la loro somma al quadrato

$$min \sum_{t=1}^{n} \epsilon_t^2 = min(\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 + \cdots + \epsilon_n^2) \tag{5.5}$$

Per il confronto procederemo con la simulazione di cento serie storiche con lo stesso valore impostato di Φ_1, verificando attraverso il metodo di stima tradizionale il valore calcolato per il parametro, mentre prenderemo per ognuna delle serie storiche simulate il valore medio di cento applicazioni di tutti e quattro gli algoritmi - PSO, FA e le due varianti del DE ($DE \setminus rand \setminus 1$ e $DE \setminus best \setminus 2$).
Con R simuliamo le cento serie di numerosità $n = 200$. Il parametro autoregressivo è posto pari a $\Phi = 0.8, -0.5, 0.2$ e 0.1.

```
x<-arima.sim (n = 200, list(ar = 0.8))
```

mentre la stima del metodo Yule-Walker per una serie AR(1) è calcolabile con la funzione

```
ar(x, order.max = 1)
```

Prima di passare alle applicazioni degli algoritmi, riepiloghiamo con la Tabella 5.1 i parametri impostati per ognuno di essi. I parametri rimarranno immutati anche nelle applicazioni successive, comprese quelle che vedremo per i modelli

autoregressivi di ordine 2 e per il SETAR.

Algoritmo	**Parametri**					
DE \ *rand* \ 1	CR=0.6	F=1.1	NP=13			
DE \ *best* \ 2	CR=0.6	F=1.1	NP=13			
PSO	w=0.7	NP=20				
FA	n=5	m=50	a=0.04	b=0.8	$\hat{A}$=40	$\hat{m}$=5

Tabella 5.1: I parametri impostati per gli algoritmi. NP sta per numero di particelle, per le altre costanti si rimanda ai relativi capitoli.

Infine, per quanto riguarda le condizioni di arresto, va detto che in tutti i casi sono state impostate con riguardo unicamente al raggiungimento di un limite di iterazioni ed ad un miglioramento sufficiente della fitness da un ciclo all'altro.

5.2.1 Serie con $\Phi_1 = 0.8$

Come detto sono state simulate cento serie storiche di numerosità $n = 200$ e tutte con lo stesso parametro $\Phi_1 = 0.8$. Per ogni serie sono stati applicati cento volte i quattro algoritmi, creando dunque quattro matrici, una per algoritmo, di dimensioni (100x100), in cui in ogni riga vengono riportate le cento stime effettuate sulla serie corrispondente. La variabilità presente all'interno di ogni riga delle matrici con le stime effettuate è, come prevedibile, molto bassa, di conseguenza appare poco significativo riportarne i valori calcolati. A titolo esemplificativo riportiamo la summary della prima riga della matrice contenente le stime per i quattro algoritmi:

```
> summary(stime_de[1,])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
 0.8632  0.8632  0.8632  0.8632  0.8632  0.8632
> summary(stime_debest[1,])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.8631  0.8632  0.8632  0.8632  0.8632  0.8633
> summary(stime_pso[1,])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.8628  0.8631  0.8632  0.8632  0.8633  0.8634
> summary(stime_fw[1,])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.8619  0.8630  0.8632  0.8632  0.8633  0.8646
```

I dati riportati, pressochè identici, sono in ordine relativi alle stime effettuate sulla prima serie con DE versione *DE* \ *rand* \ 1, DE versione *DE* \ *best* \ 2, PSO e FA. Questi sono i risultati, mentre il parametro calcolato con Yule - Walker è ottenuto come segue

```
> ar(serie[1,],order.max=1)
Call:
ar(x = serie[1, ], order.max = 1)
Coefficients:

     1
0.8546
Order selected 1  sigma^2 estimated as  1.113
```

I risultati sono molto simili anche per le rimanenti 99 serie simulate, seppur con una distribuzione concentrata attorno ad un altro valore medio, vicino a 0.8 ma diverso da 0.86. Per un confronto sulle distribuzioni delle stime, consideriamo le medie aritmetiche dei valori stimati per ognuna delle serie storiche simulate,

calcolando perciò 100 valori medi (uno per serie), mettendoli in relazione con quanto calcolato con Yule - Walker.

Come era lecito aspettarsi nei quattro casi i cento valori si distribuiscono in

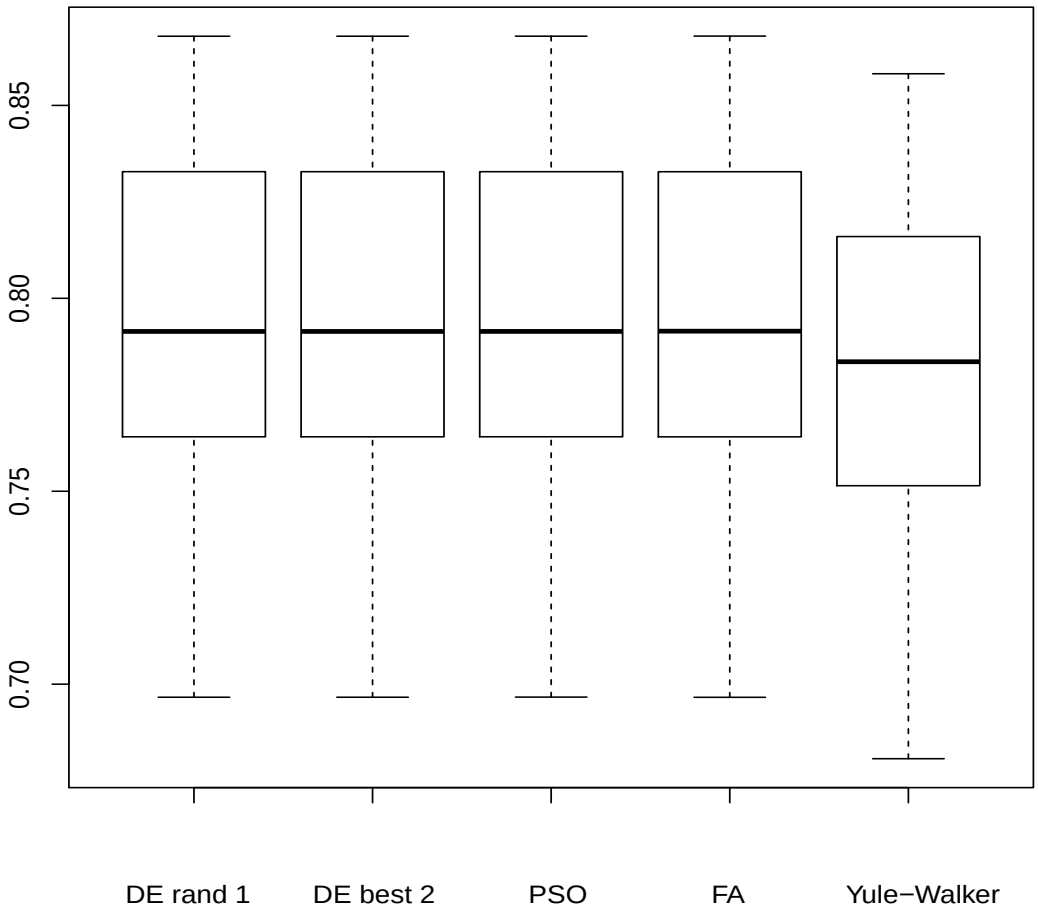

Figura 5.1: Boxplot delle medie delle stime - $\Phi_1 = 0.8$ - e delle stime con Yule-Walker.

modo quasi analogo, partendo da dati a loro volta molto simili. Quì però è possibile vedere una maggiore variabilità , con i massimi che superano lo 0.85 ed i minimi inferiori allo 0.70: ciò è dovuto non ad errori degli algoritmi, ma alle serie simulate, tutte diverse anche se con lo stesso parametro impostato. Infine, rispetto a Yule - Walker il risultato è leggermente migliore. A dimostrazione del

fatto di seguito si riportano i dati.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ *rand* \ 1	0.6966	0.7646	0.7914	0.7939	0.8328	0.8679
DE \ *best* \ 2	0.6966	0.7646	0.7914	0.7939	0.8328	0.8679
PSO	0.6966	0.7646	0.7914	0.7939	0.8328	0.8679
FA	0.6966	0.7646	0.7915	0.7939	0.8328	0.8680
Yule-Walker	0.6807	0.7518	0.7836	0.7817	0.8152	0.8582

Tabella 5.2: Summary medie - $\Phi_1 = 0.8$ - e stime con Yule-Walker.

Quanto visto graficamente coi boxplot è riportato in Tabella 5.2 : con le applicazioni degli algoritmi otteniamo dati pressochè identici. Nel confronto col metodo Yule-Walker poi, a parità di intervallo di variazione (sempre intorno allo 0.17), vediamo che la media ottenuta dagli algoritmi si avvicina leggermente di più allo 0.8 impostato.

Quanto visto finora si basa su un campione ristretto, infatti abbiamo preso in considerazione solo la distribuzione di cento medie ottenute a partire da cento stime ottenute su cento simulazioni diverse. Per cercare di costruire la distribuzione empirica dello stimatore, consideriamo il metodo bootstrap.
La prima stima, cioè il valore mediante il quale si ricostruiranno tutte le serie bootstrap, viene stimato a partire da una nuova serie storica, simulata con lo stesso $\Phi_1 = 0.8$ ma diversa da quelle su cui abbiamo già lavorato.

DE \ rand\ 1	DE \ best \ 2	PSO	FA	Yule-Walker
0.7559056	0.7559086	0.7558416	0.7566812	0.7334381

Tabella 5.3: Stime sulla serie iniziale non bootstrappata.

Le stime presenti in tabella 5.3 sono quelle che verranno utilizzate come parametro per calcolare i residui necessari per generare delle serie bootstrap. Il numero di repliche bootstrap è stato fissato a mille per tutti e quattro gli algoritmi.

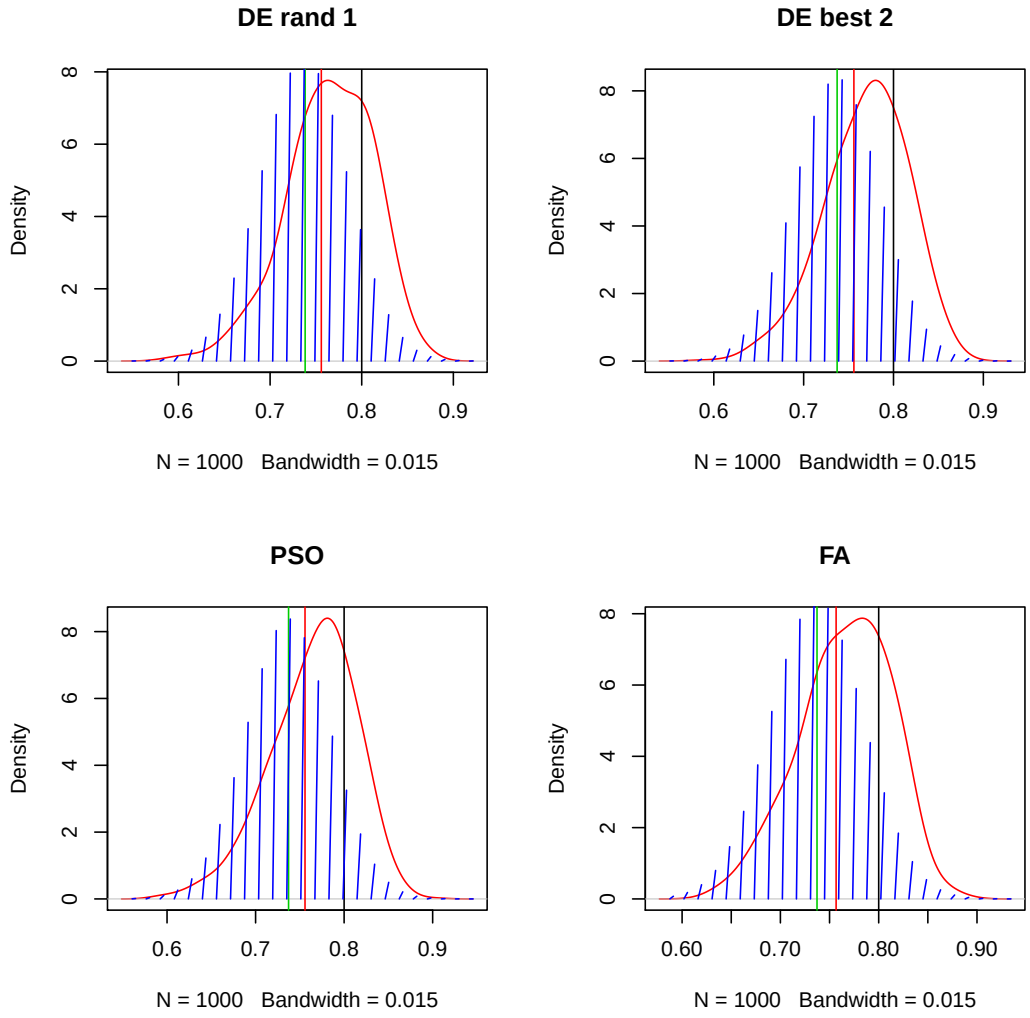

Figura 5.2: Distribuzioni delle stime sulle serie bootstrap a partire da un AR(1) con $\Phi_1 = 0.8$.

In Figura 5.2 vengono illustrati graficamente i risultati delle stime sulle serie bootstrap. La curva rossa è una stima mediante kernel della distribuzione dei valori stimati nei quattro casi, mentre la linea rossa verticale rappresenta la stima del parametro effettuata sulla serie simulata. Quest'ultimi corrispondono ai valori riportati in tabella 5.3. La linea verticale nera è invece sullo 0.8, quindi il valore con cui è stata simulata la serie. La linea verde verticale è sul valore calcolato

con le equazioni di Yule-Walker, il metodo tradizionale, e le linee blu spaziano all'interno dell'area in cui si distribuirebbe una normale con media sulla linea verde e varianza calcolata sempre sulla base del metodo Yule-Walker. Da notare che la distribuzione delle stime calcolate con gli algoritmi si avvicina in media al valore prefissato 0.8. Riportiamo quindi le relative summary.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ *rand* \ 1	0.5829	0.7350	0.7670	0.7650	0.8010	0.8765
DE \ *best* \ 2	0.5844	0.7399	0.7732	0.7695	0.8022	0.8856
PSO	0.5936	0.7341	0.7709	0.7647	0.7980	0.9007
FA	0.6219	0.7361	0.7700	0.7658	0.8003	0.8895

Tabella 5.4: Summary medie Φ_1 bootstrap.

Calcoliamo ora l'intervallo di confidenza per la stima di Φ_1. Possiamo utilizzare il metodo del percentile, avendo utilizzato nel procedimento bootstrap dei sub-campioni sufficientemente grandi (ricordiamo che la numerosità dei campioni di residui per ogni serie è di 199), un metodo che si basa sull'utilizzo dei percentili della distribuzione bootstrap cumulata di $\hat{\Phi}_1^\star$, stimatore di Φ_1 [Bonanomi A., 2007]. Ad un livello di confidenza $1-2\alpha = 0.95$, fissando $\alpha = 0.025$, gli estremi dell'intervallo di confidenza sono:

$$[\hat{\Phi}_1^{\star(\alpha)}; \hat{\Phi}_1^{\star(1-\alpha)}]$$

e quindi per costruire l'intervallo di confidenza si prenderanno i percentili di ordine α e $1-\alpha$ della distribuzione bootstrap delle stime.
Gli intervalli sono in definitiva i seguenti

Algoritmo	**Intervallo**	σ

DE \ *rand* \ 1	$0.6621772 < \Phi_1 < 0.8433711$	0.04742217
DE \ *best* \ 2	$0.6647457 < \Phi_1 < 0.8497591$	0.04649845
PSO	$0.6564595 < \Phi_1 < 0.8449813$	0.04814068
FA	$0.6675989 < \Phi_1 < 0.8409381$	0.0462729
Yule-Walker	$0.6387541 < \Phi_1 < 0.8281221$	0.04830817

Tabella 5.5: Intervalli di confidenza per Φ_1 ottenuti dalla distribuzione delle stime su serie bootstrap e con metodo tradizionale e relative standard deviations.

Per il calcolo dell'intervallo sul parametro ottenuto col metodo Yule-Walker è stata presa la varianza risultante dall'applicazione con R del procedimento, costruendo l'intervallo assumendo che la variabile sia distribuita come una normale.

5.2.2 Serie con $\Phi_1 = -0.5$

Lo stesso procedimento verrà ora applicato per serie simulate con un valore di $\Phi_1 = -0.5$, per vedere se quanto visto con un valore più alto e positivo si ripete anche in questo caso.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ *rand* \ 1	-0.6114	-0.5368	-0.5003	-0.4983	-0.4577	-0.3868
DE \ *best* \ 2	-0.6115	-0.5367	-0.5004	-0.4982	-0.4578	-0.3868
PSO	-0.6113	-0.5368	-0.5005	-0.4982	-0.4577	-0.3868
FA	-0.6114	-0.5368	-0.5003	-0.4983	-0.4577	-0.3868
Yule-Walker	-0.6086	-0.5357	-0.5002	-0.4978	-0.4582	-0.3854

Tabella 5.6: Summary medie - $\Phi_1 = -0.5$ - e stime con Yule-Walker.

Dalla media delle stime con tutti e quattro gli algoritmi utilizzati si ha un risultato soddisfacente, infatti la mediana è quasi perfettamente sul valore -0.5 come peraltro la media (vedi Tabella 5.6). I risultati fin qui sono anche migliori rispetto alle stime su serie con valore 0.8. Come visto in precedenza, anche in questo caso i risultati sono quasi gli stessi. Non si notano differenze nemmeno con quanto evidenziato dal metodo Yule-Walker. Venendo al metodo bootstrap,

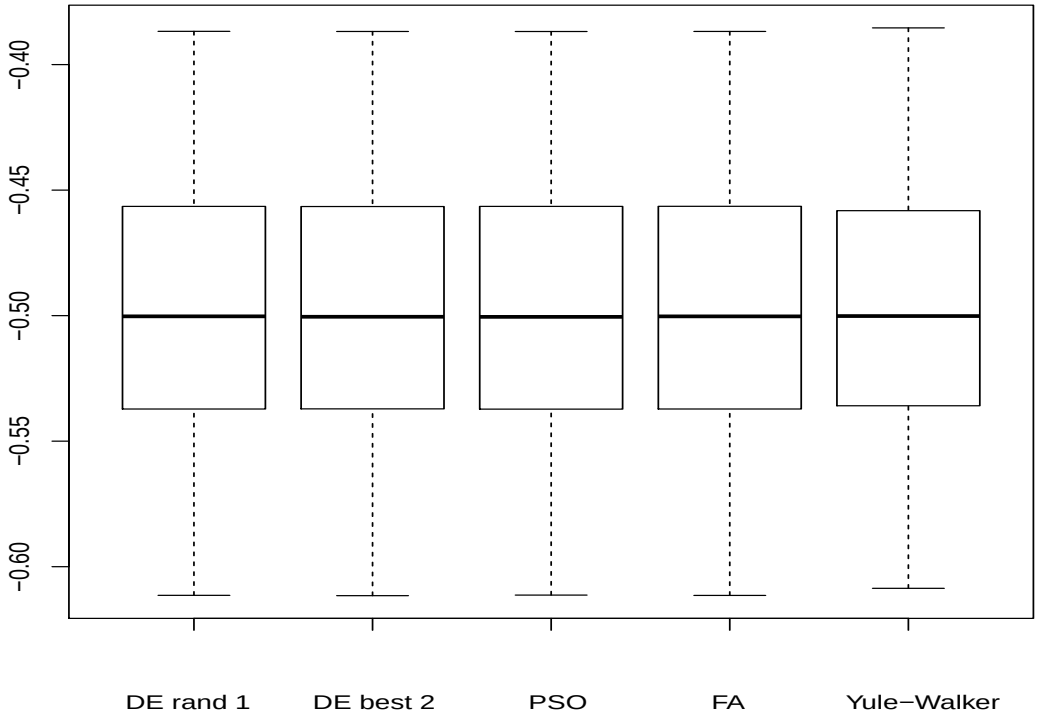

Figura 5.3: Boxplot delle medie delle stime - $\Phi_1 = -0.5$ per le due versioni del DE, PSO e FA e delle stime con Yule-Walker.

iniziamo guardando il valore calcolato sulla nuova serie simulata che andremo ad utilizzare.

DE \ rand\ 1	DE \ best \ 2	PSO	FA	Yule-Walker
-0.3677996	-0.3687018	-0.3681593	-0.3679251	-0.3683749

Tabella 5.7: Stime sulla serie iniziale non bootstrappata.

Rispetto al caso precedente, qui la distribuzione delle stime (curva rossa) non

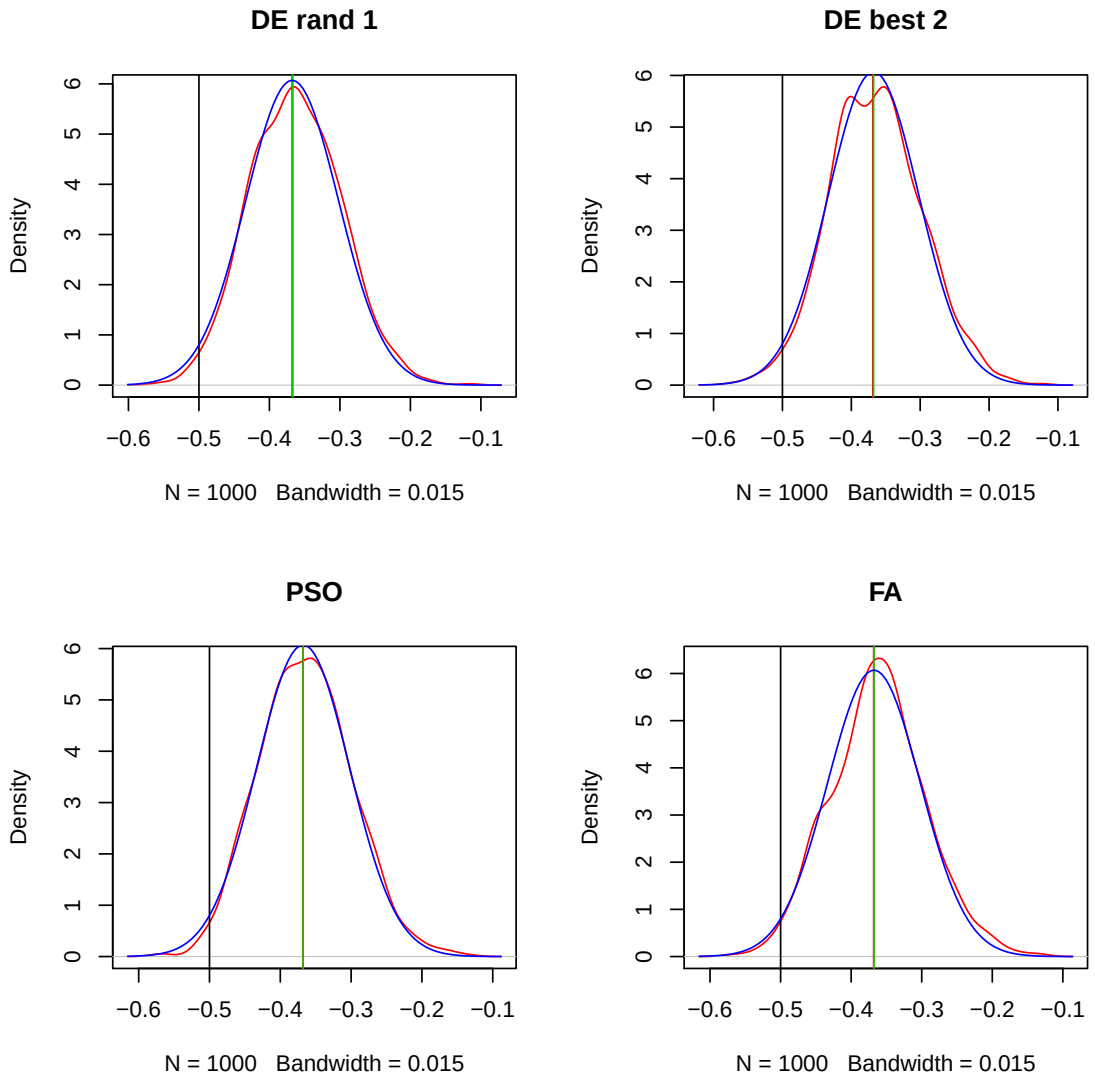

Figura 5.4: Distribuzioni delle stime sulle serie bootstrap a partire da un AR(1) con $\Phi_1 = -0.5$.

è ugualmente vicina al valore impostato.

Come si vede però, la distribuzione ottenuta e quella di una normale con media e varianza calcolate col metodo classico (curva blu) vanno quasi a sovrapporsi in tutte e quattro le applicazioni effettuate.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ *rand* \ 1	-0.5561	-0.4092	-0.3643	-0.3629	-0.3197	-0.1162
DE \ *best* \ 2	-0.5760	-0.4098	-0.3651	-0.3641	-0.3203	-0.1237
PSO	-0.5705	-0.4100	-0.3648	-0.3642	-0.3216	-0.1331
FA	-0.5705	-0.4072	-0.3626	-0.3622	-0.3202	-0.1312

Tabella 5.8: Summary medie Φ_1 bootstrap.

Gli intervalli risultanti, con lo stesso livello di confidenza del 95%, sono elencati nella Tabella 5.9:

Algoritmo	Intervallo	σ
DE \ *rand* \ 1	-0.4826649 < Φ_1 < -0.2340271	0.06418243
DE \ *best* \ 2	-0.4925205 < Φ_1 < -0.226447	0.06657661
PSO	-0.4810852 < Φ_1 < -0.2294247	0.06501269
FA	-0.4863658 < Φ_1 < -0.2253339	0.06674077
Yule-Walker	-0.4978707 < Φ_1 < -0.2388791	0.06606929

Tabella 5.9: Intervalli di confidenza per Φ_1 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

A differenza di quanto riscontrato con il parametro 0.8, in questo caso il valore non è compreso nel range dell'intervallo, ma va anche notato che gli intervalli e la varianza ottenuti dagli algoritmi sono molto vicini a quelli del metodo tradizionale.

5.2.3 Serie con $\Phi_1 = 0.2$

Quanto fatto finora viene ripetuto con serie simulate a parametro 0.2.

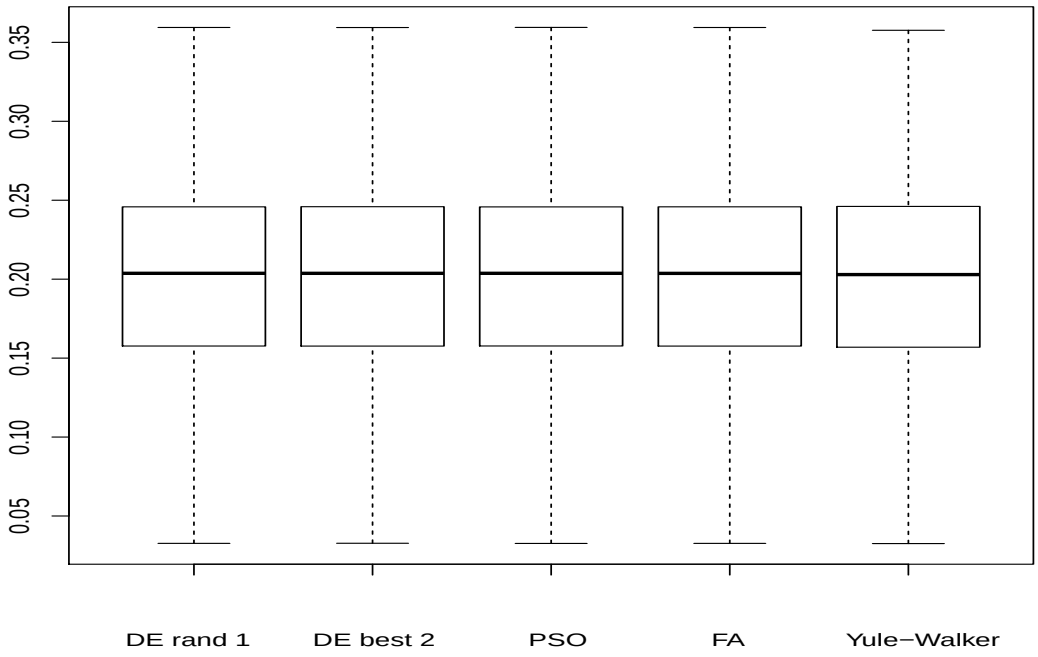

Figura 5.5: Boxplot delle medie delle stime - $\Phi_1 = 0.2$ e delle stime con Yule-Walker.

Come già visto le medie delle 100 stime su ognuna delle 100 serie simulate danno risultati buoni, infatti abbiamo uno scarto interquantilico abbastanza ridotto e concentrato attorno al valore esatto.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ rand \ 1	0.03267	0.15770	0.20370	0.20080	0.24580	0.35940
DE \ best \ 2	0.03273	0.15770	0.20370	0.20080	0.24590	0.35940
PSO	0.03262	0.15770	0.20370	0.20070	0.24580	0.35950
FA	0.03265	0.15770	0.20370	0.20080	0.24580	0.35940
Yule-Walker	0.03253	0.15700	0.20290	0.20090	0.24590	0.35760

Tabella 5.10: Summary medie - $\Phi_1 = 0.2$ - e stime con Yule-Walker.

DE \ rand\ 1	DE \ best \ 2	PSO	FA	Yule-Walker
0.2124261	0.2124685	0.2124784	0.2125789	0.1961968

Tabella 5.11: Stime sulla serie iniziale non bootstrappata.

Anche nella distribuzione delle stime sulle serie bootstrap, l'applicazione de-

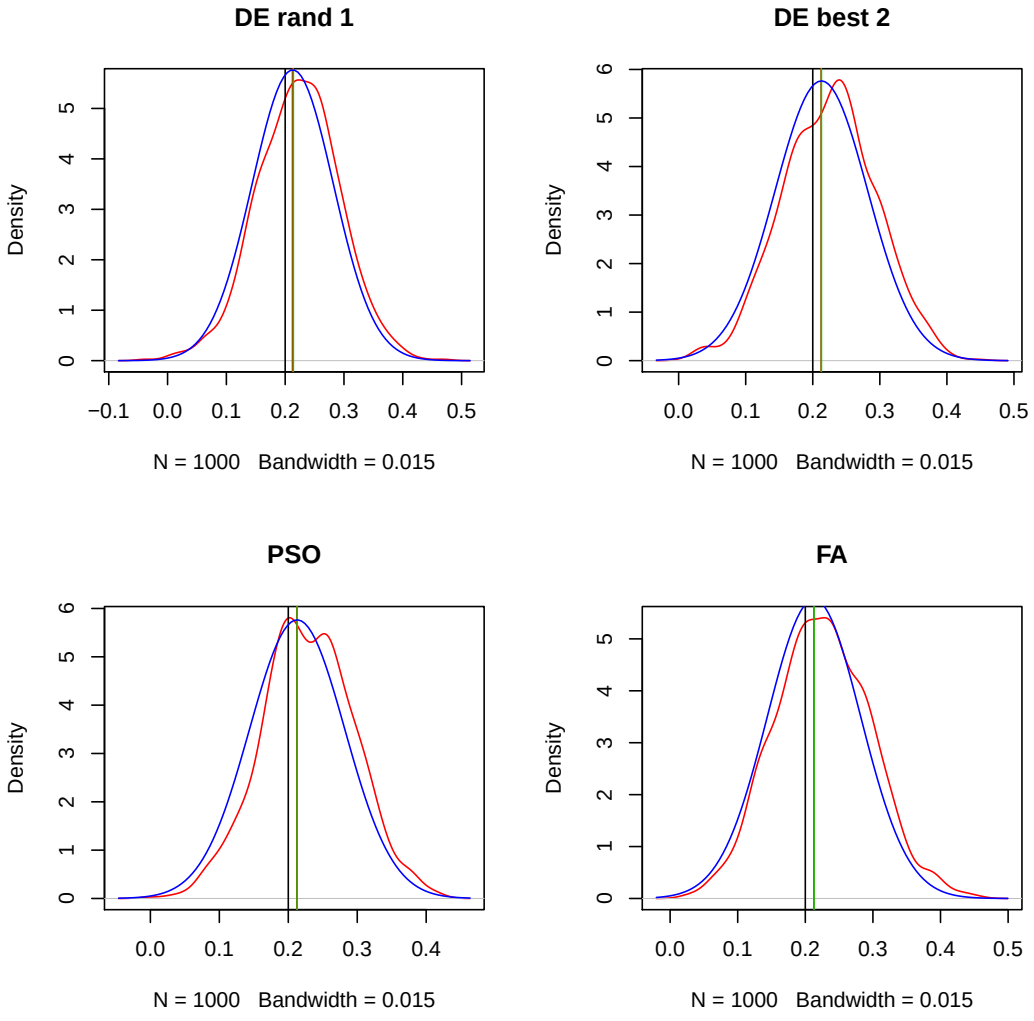

Figura 5.6: Distribuzioni delle stime sulle serie bootstrap a partire da un AR(1) con $\Phi_1 = 0.2$.

gli algoritmi si dimostra essere una valida alternativa allo stimatore tradizionale.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ *rand* \ 1	-0.03838	0.17420	0.22310	0.22240	0.26790	0.46950
DE \ *best* \ 2	0.01178	0.17560	0.22800	0.22470	0.27030	0.44590
PSO	0.02465	0.17660	0.22460	0.22530	0.27520	0.45470
FA	-0.00117	0.18450	0.22750	0.22840	0.27300	0.41870

Tabella 5.12: Summary medie Φ_1 bootstrap.

Le distribuzioni sono molto simili, come gli intervalli: quello calcolato con Yule-Walker non ha gli stessi estremi degli altri quattro, ma ha un intervallo di ampiezza quasi equivalente.

Algoritmo	Intervallo	σ
DE \ *rand* \ 1	$0.07795964 < \Phi_1 < 0.3588861$	0.07021698
DE \ *best* \ 2	$0.09337058 < \Phi_1 < 0.3627233$	0.06971589
PSO	$0.09511734 < \Phi_1 < 0.3589927$	0.06628258
FA	$0.09109586 < \Phi_1 < 0.3720056$	0.07053974
Yule-Walker	$0.05961285 < \Phi_1 < 0.3327807$	0.06968569

Tabella 5.13: Intervalli di confidenza per Φ_1 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

5.2.4 Serie con $\Phi_1 = 0.1$

Ancora una volta le stime sono buone e le distribuzioni dei cento valori quasi identiche.

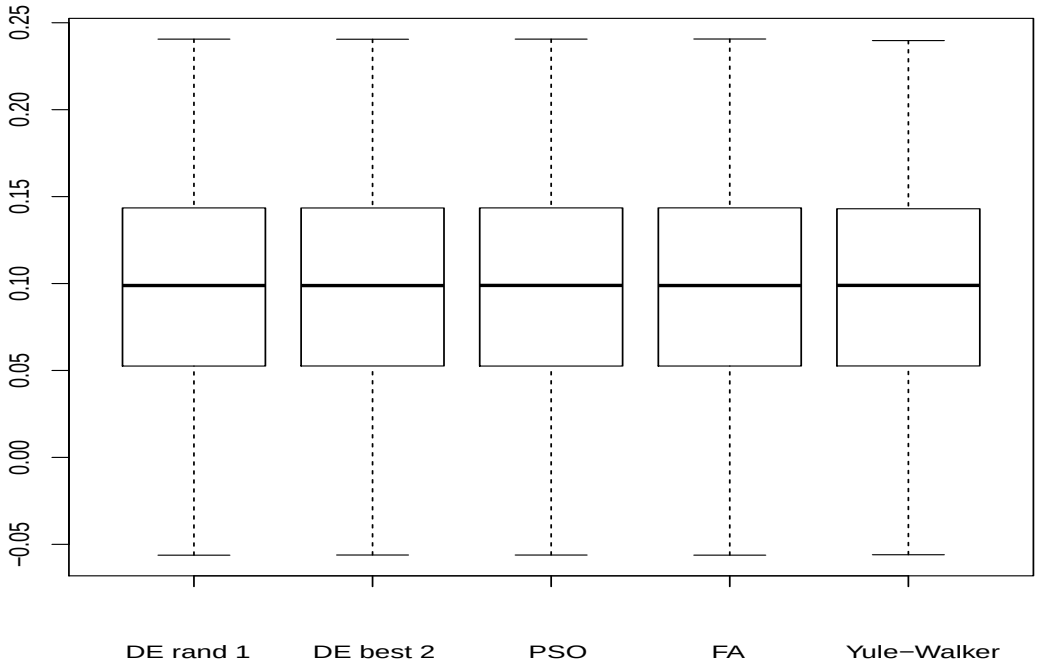

Figura 5.7: Boxplot delle medie delle stime - $\Phi_1 = 0.1$ - e delle stime con Yule-Walker.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ *rand* \ 1	-0.05623	0.05266	0.09886	0.09886	0.14350	0.24060
DE \ *best* \ 2	-0.05614	0.05274	0.09884	0.09885	0.14350	0.24050
PSO	-0.05615	0.05266	0.09892	0.09884	0.14350	0.24060
FA	-0.05622	0.05269	0.09886	0.09886	0.14350	0.24060
Yule-Walker	-0.05598	0.05288	0.09894	0.09881	0.14290	0.23970

Tabella 5.14: Summary medie - $\Phi_1 = 0.1$ - e stime con Yule-Walker.

DE \ rand\ 1	DE \ best \ 2	PSO	FA	Yule-Walker
0.09724947	0.09686527	0.09702974	0.09679347	0.09599608

Tabella 5.15: Stime sulla serie iniziale non bootstrappata.

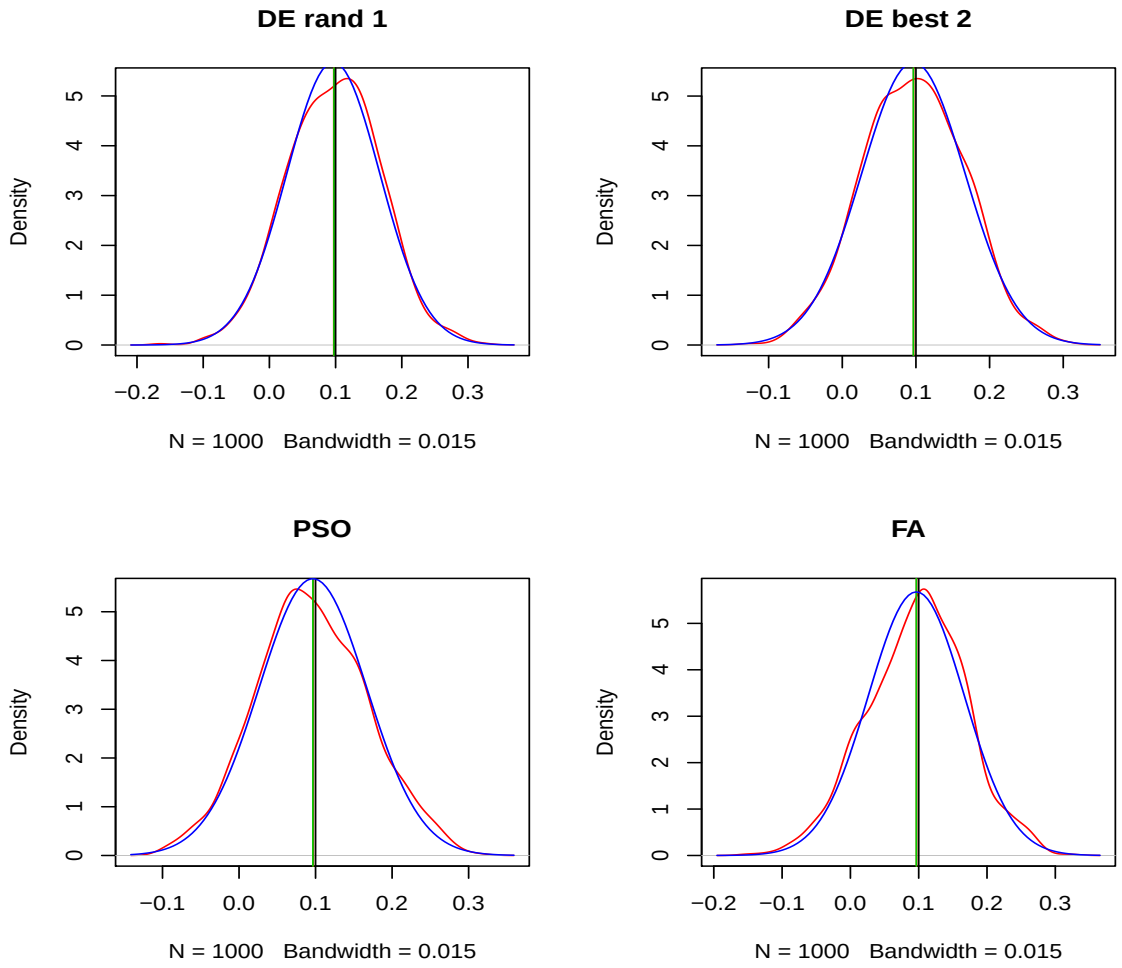

Figura 5.8: Distribuzioni delle stime sulle serie bootstrap a partire da un AR(1) con $\Phi_1 = 0.1$.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ *rand* \ 1	-0.16420	0.04902	0.09852	0.09781	0.14610	0.32450
DE \ *best* \ 2	-0.12510	0.05040	0.09715	0.09735	0.14620	0.30500
PSO	-0.09629	0.04717	0.09241	0.09534	0.14520	0.31430
FA	-0.15040	0.04744	0.10200	0.09680	0.14590	0.32050

Tabella 5.16: Summary medie Φ_1 bootstrap - $\Phi_1 = 0.1$.

Algoritmo	**Intervallo**	σ
DE \ *rand* \ 1	-0.03857618 < Φ_1 < 0.2295847	0.070019
DE \ *best* \ 2	-0.03492021 < Φ_1 < 0.2281232	0.06895033
PSO	-0.04514994 < Φ_1 < 0.2429346	0.07222256
FA	-0.05054489 < Φ_1 < 0.2354624	0.07197224
Yule-Walker	-0.0417263 < Φ_1 < 0.2337185	0.07026652

Tabella 5.17: Intervalli di confidenza per Φ_1 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

5.3 Applicazione con un AR(2)

La seconda applicazione degli algoritmi viene effettuata per la stima dei parametri Φ_1 e Φ_2 di un modello $AR(2)$. Data una serie storica di lunghezza n, la funzione che ne descrive l'andamento nel caso di un $AR(2)$ è

$$y_t = \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \epsilon_t \qquad t = 1, 2, \ldots, n \tag{5.6}$$

Dalla (5.6) si definisce la funzione fitness da impostare per applicare gli algoritmi nel caso di un AR(2). Dopo aver trovato i residui con la (5.7)

$$\epsilon_t = y_t - \Phi_1 y_{t-1} - \Phi_2 y_{t-2} \qquad t = 1, 2, \ldots, n \tag{5.7}$$

il valore fitness si ottiene minimizzandoli

$$min \sum_{t=1}^{n} \epsilon_t^2 = min(\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 + \cdots + \epsilon_n^2) \tag{5.8}$$

Dopo aver impostato la funzione fitness, verranno simulate serie con due valori per i rispettivi $\Phi_{1,2}$, mentre il resto del procedimento sarà il medesimo di quello

visto per i modelli AR(1), quindi con una iniziale simulazione di cento serie con numerosità pari a 200, e poi la ricostruzione di mille serie bootstrap e su di esse l'applicazione con l'ottenimento di mille valori e le considerazioni sulla loro distribuzione.

5.3.1 Serie con $\Phi_1 = -0.3$ e $\Phi_2 = -0.8$

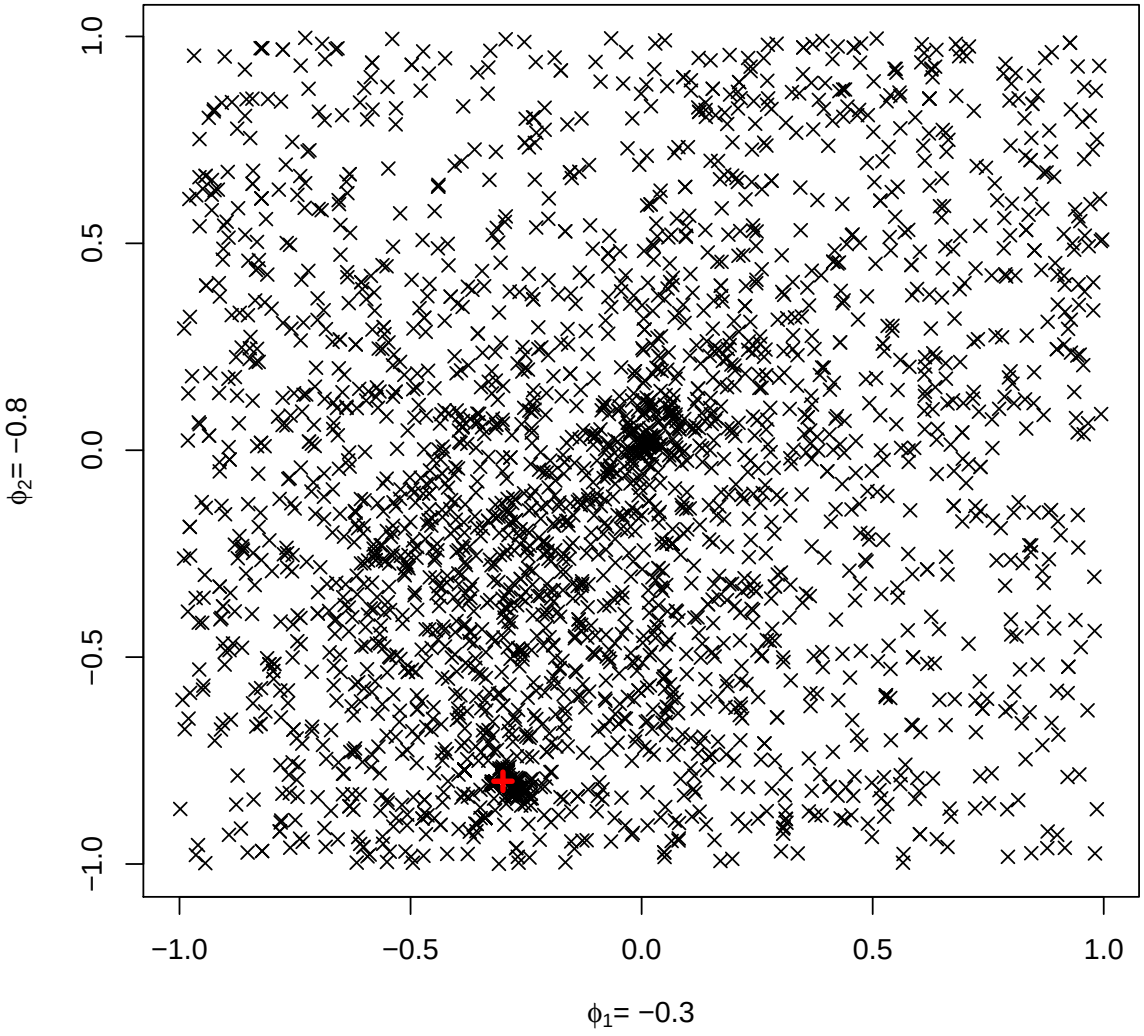

Figura 5.9: Esempio di ricerca all'interno dello spazio parametrico effettuato col Fireworks Algorithm per un AR(2) con parametri $\Phi_1 = -0.3$ e $\Phi_2 = -0.8$. Ogni crocetta rappresenta una scintilla che per coordinate ha in ascissa la stima di Φ_1 ed in ordinata la stima di Φ_2. Nel caso specifico le scintille generate sono state 4059. La croce rossa è il punto di coordinate (-0.3, -0.8.)

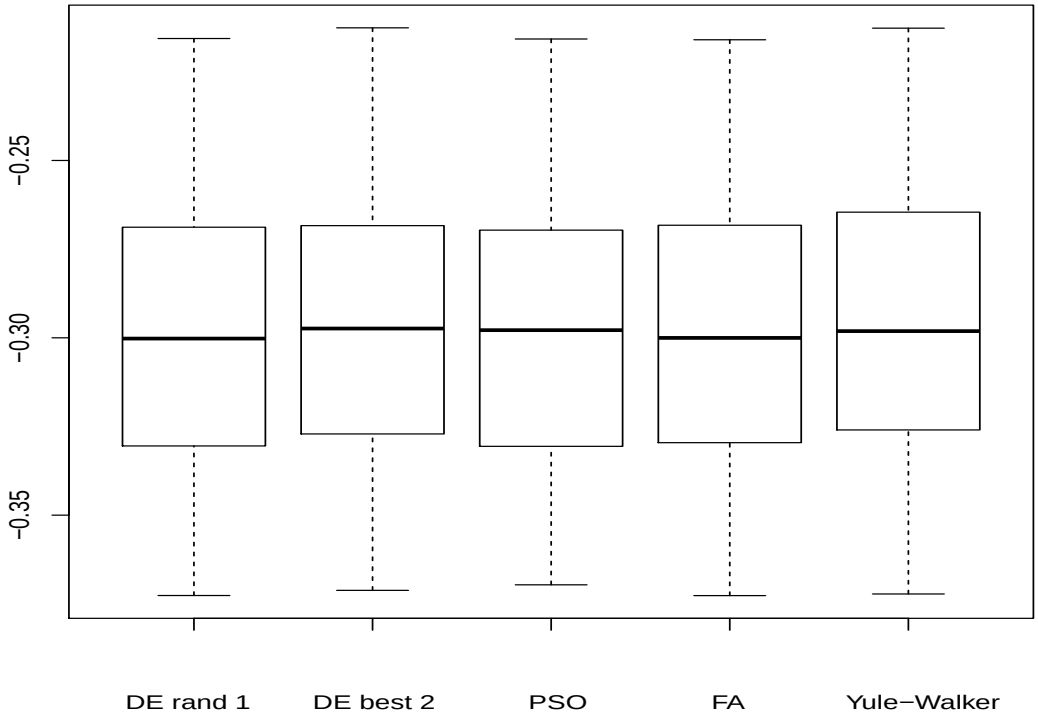

Figura 5.10: Boxplot delle medie delle stime per il primo parametro $\Phi_1 = -0.3$ di un AR(2) e delle stime con Yule-Walker.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ *rand* \ 1	-0.5122	-0.3797	-0.3495	-0.3488	-0.3180	-0.2005
DE \ *best* \ 2	-0.5285	-0.3813	-0.3473	-0.3466	-0.3125	-0.1905
PSO	-0.4818	-0.3730	-0.3449	-0.3432	-0.3138	-0.1875
FA	-0.4986	-0.3730	-0.3434	-0.3412	-0.3104	-0.1375
Yule-Walker	-0.3722	-0.3259	-0.2981	-0.2965	-0.2647	-0.2127

Tabella 5.18: Summary medie - $\Phi_1 = -0.3$ - e stime con Yule-Walker.

Rispetto alle stime col metodo tradizionale, in questo caso dall'applicazione degli algoritmi sono state ottenute medie maggiormente distanti dal valore impostato, confrontando questo anche con quanto visto negli AR(1), inoltre, evidenza

in relazione con la media, i valori agli estremi sono più distanti, creando un intervallo di variazione di ampiezza maggiore a quello che si trova con le stime Yule-Walker.

Nelle stime del secondo parametro invece troviamo di nuovo dei dati molto

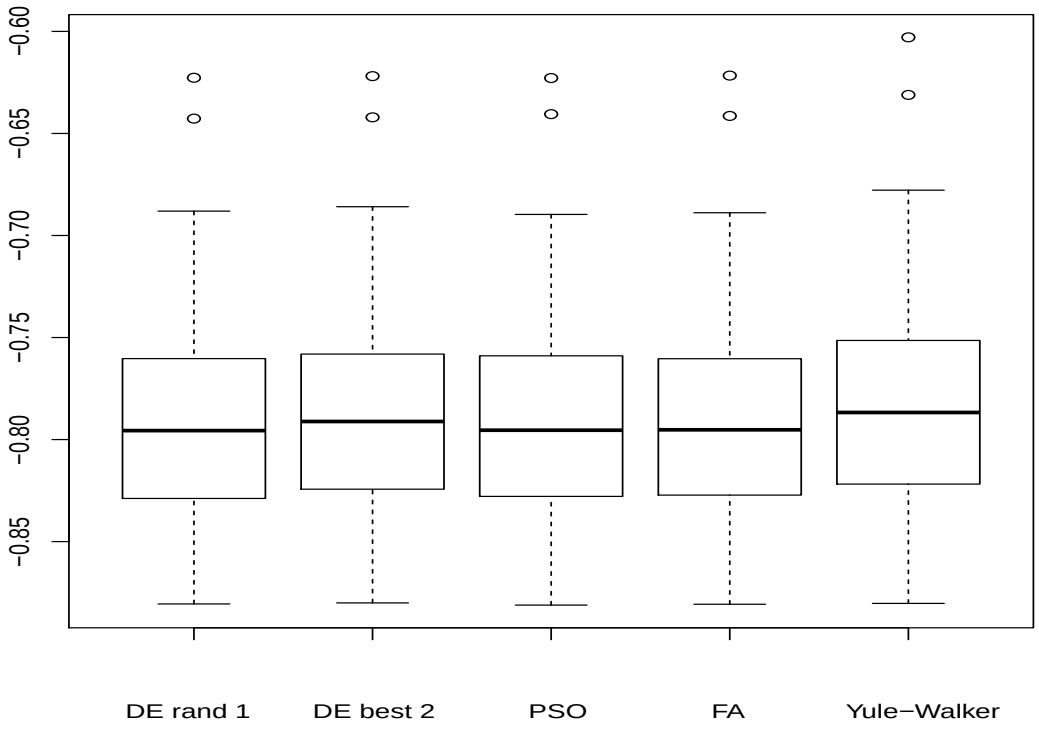

Figura 5.11: Boxplot delle medie delle stime per il secondo parametro $\Phi_2 = -0.8$ di un AR(2) e delle stime con Yule-Walker.

simili per i quattro algoritmi e per il metodo tradizionale.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ rand \ 1	-0.8983	-0.8080	-0.7785	-0.7753	-0.7467	-0.5900
DE \ best \ 2	-0.9078	-0.8071	-0.7759	-0.7726	-0.7391	-0.5707
PSO	-0.8905	-0.8060	-0.7773	-0.7731	-0.7452	-0.5885
FA	-0.8900	-0.7983	-0.7694	-0.7631	-0.7326	-0.5758
Yule-Walker	-0.8803	-0.8218	-0.7867	-0.7835	-0.7516	-0.6029

Tabella 5.19: Summary medie - $\Phi_1 = -0.8$ - e stime con Yule-Walker.

Per il metodo bootstrap ci muoviamo esattamente come prima.

	DE \ rand\ 1	**DE \ best \ 2**	**PSO**	**FA**	**Yule-Walker**
Φ_1	-0.3501773	-0.3601257	-0.3460505	-0.3432367	-0.3465728
Φ_2	-0.7830146	-0.7731453	-0.779691	-0.7689368	-0.7723367

Tabella 5.20: Stime sulla serie iniziale non bootstrappata.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ rand \ 1	-0.5122	-0.3797	-0.3495	-0.3488	-0.3180	-0.2005
DE \ best \ 2	-0.5285	-0.3813	-0.3473	-0.3466	-0.3125	-0.1905
PSO	-0.4818	-0.3730	-0.3449	-0.3432	-0.3138	-0.1875
FA	-0.4986	-0.3730	-0.3434	-0.3412	-0.3104	-0.1375

Tabella 5.21: Summary medie Φ_1 bootstrap - $\Phi_1 = -0.3$.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ rand \ 1	-0.8983	-0.8080	-0.7785	-0.7753	-0.7467	-0.5900
DE \ best \ 2	-0.9078	-0.8071	-0.7759	-0.7726	-0.7391	-0.5707
PSO	-0.8905	-0.8060	-0.7773	-0.7731	-0.7452	-0.5885
FA	-0.8900	-0.7983	-0.7694	-0.7631	-0.7326	-0.5758

Tabella 5.22: Summary medie Φ_2 bootstrap - $\Phi_2 = -0.8$.

Dai grafici sulle distribuzioni delle stime del primo parametro vediamo una fotografia di quanto visto coi numeri sulle prime cento medie: la media non

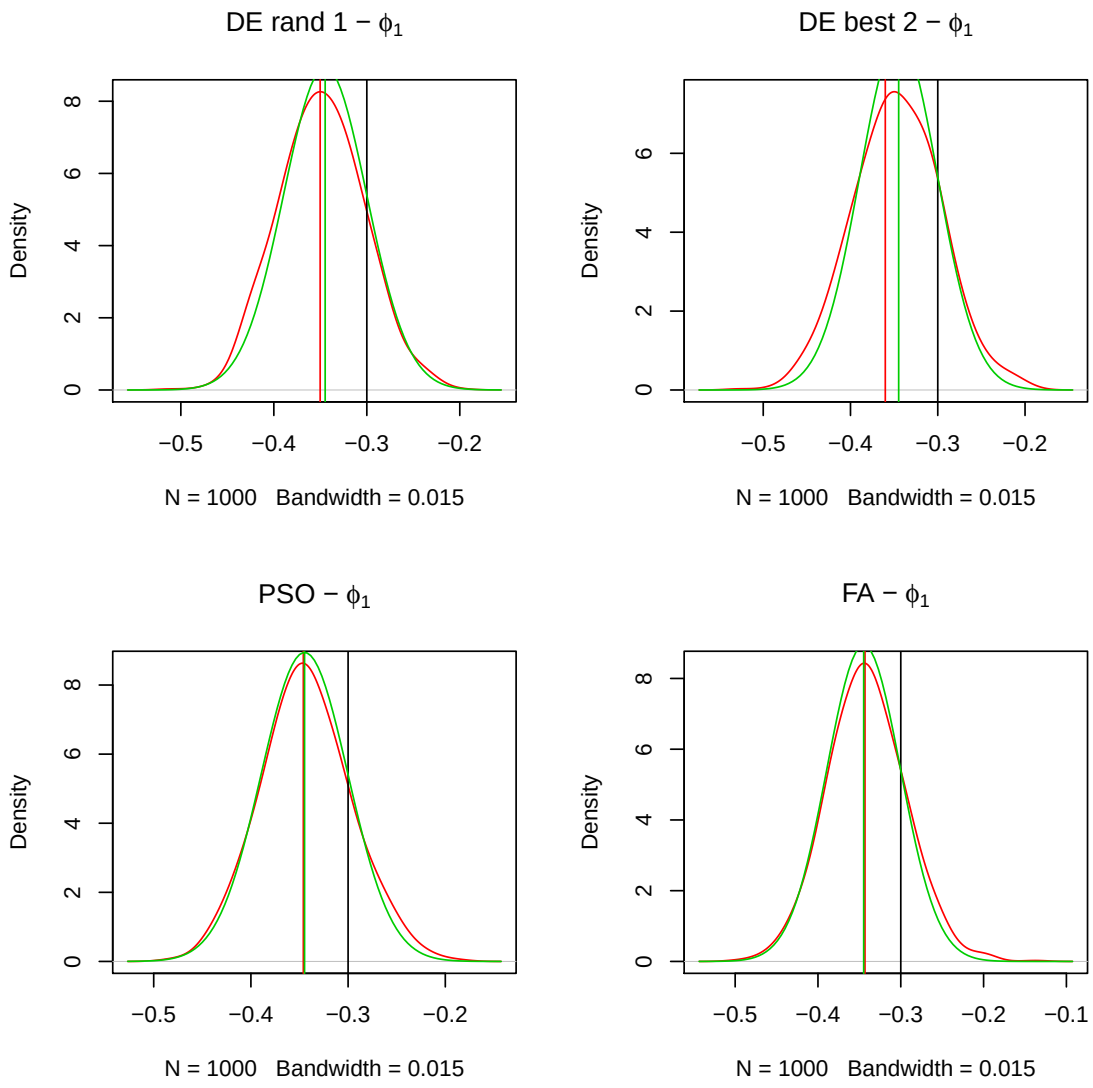

Figura 5.12: Distribuzioni delle stime sulle serie bootstrap per il primo parametro Φ_1 di un AR(2).

è centrata sul valore medio della simulazione -0.3, ma su un valore vicino; è importante però sottolineare come le due distribuzioni (curve verdi e curve rosse) siano molto simili.

Per il parametro Φ_2 si ripete ancora molto similmente quanto visto in precedenza, anche se il grafico ci mostra due curve non sovrapposte quasi perfettamente come per il parametro Φ_1. Gli intervalli di confidenza a livello 95% vengono riportati nelle Tabelle 5.23 e 5.24.

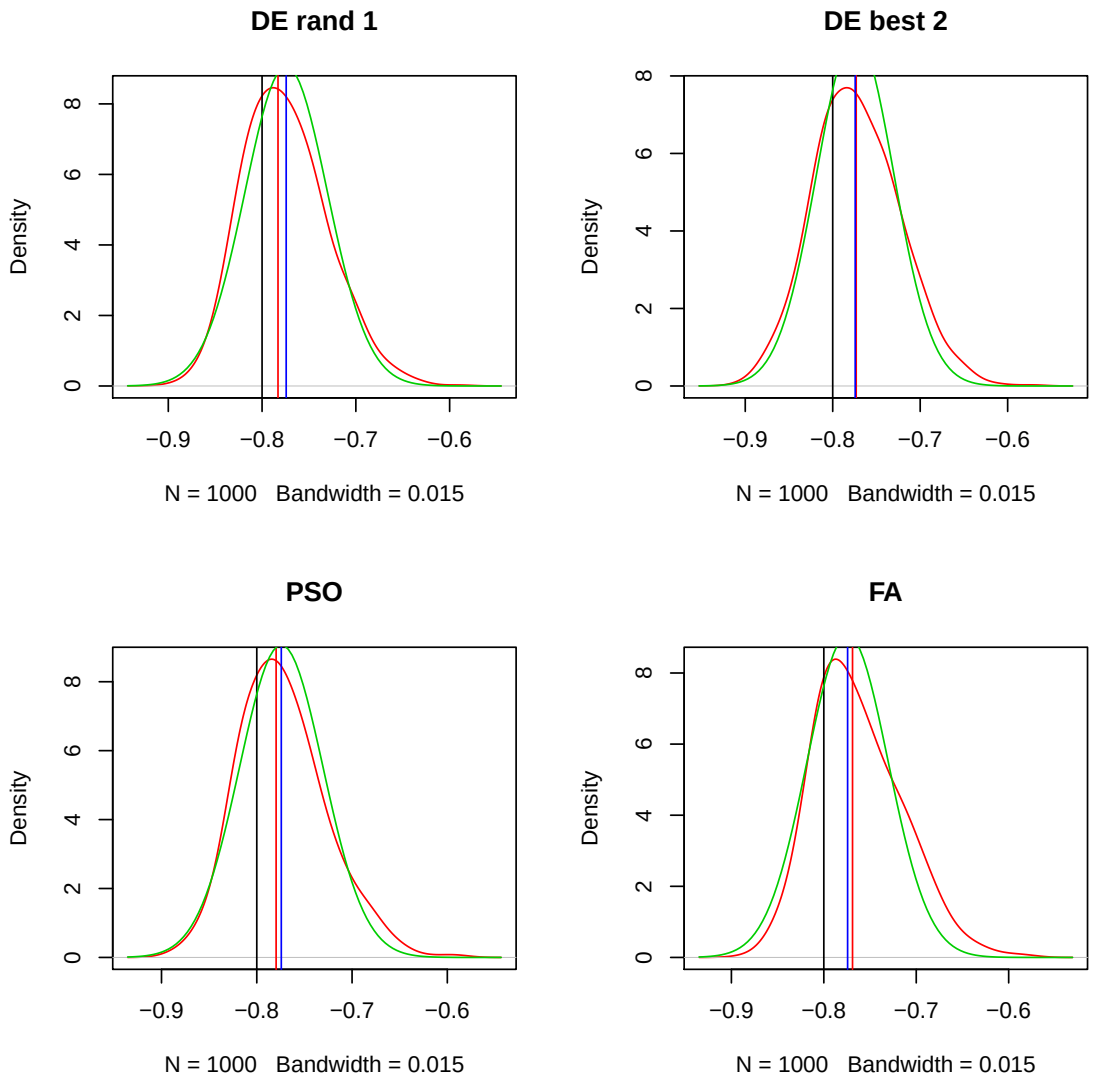

Figura 5.13: Distribuzioni delle stime sulle serie bootstrap per il secondo parametro Φ_2 di un AR(2).

Algoritmo	**Intervallo**	σ
DE \ *rand* \ 1	-0.4301195 < Φ_1 < -0.2563662	0.04518788
DE \ *best* \ 2	-0.4456655 < Φ_1 < -0.2438387	0.05041669
PSO	-0.4306802 < Φ_1 < -0.2503328	0.04599554
FA	-0.429659 < Φ_1 < -0.2477616	0.04719707

Yule-Walker	-0.4352766< Φ_1 < -0.2578689	0.04525707

Tabella 5.23: Intervalli di confidenza per Φ_1 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

Algoritmo	**Intervallo**	σ
DE \ *rand* \ 1	-0.8518837< Φ_1 < -0.6798939	0.04447134
DE \ *best* \ 2	-0.8642088 < Φ_1 < -0.6697111	0.04935789
PSO	-0.8518999 < Φ_1 < -0.6743477	0.04567294
FA	-0.8435784 < Φ_1 <-0.6618634	0.04798053
Yule-Walker	-0.8610406 < Φ_1 < -0.6836329	0.04525707

Tabella 5.24: Intervalli di confidenza per Φ_2 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

Come per il primo parametro così per il secondo non ci sono grosse differenze negli intervalli, quello che risulta dalle stime col metodo Yule-Walker è anch'esso molto vicino a quelli calcolati con i metodi di stima euristici.

5.3.2 Serie con $\Phi_1 = 0.9$ e $\Phi_2 = -0.9$

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ *rand* \ 1	0.8148	0.8786	0.8960	0.8958	0.9140	0.9715
DE \ *best* \ 2	0.8147	0.8787	0.8957	0.8958	0.9144	0.9724
PSO	0.8136	0.8780	0.8961	0.8954	0.9141	0.9732
FA	0.8152	0.8779	0.8950	0.8949	0.9136	0.9694

Yule-Walker	0.8089	0.8630	0.8870	0.8838	0.9060	0.9366

Tabella 5.25: Summary medie - $\Phi_1 = 0.9$ - e stime con Yule-Walker.

Nelle simulazioni con parametri $\Phi_1 = 0.9$ e $\Phi_2 = -0.9$ le stime effettuate con gli algoritmi si rivelano leggermente migliori.

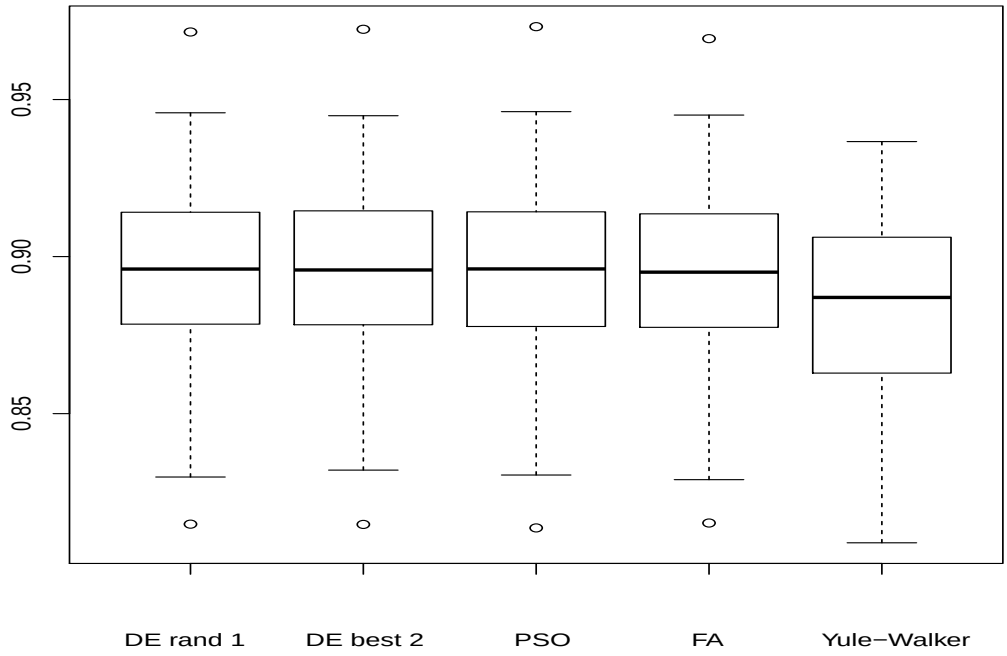

Figura 5.14: Boxplot delle medie delle stime per il primo parametro $\Phi_1 = 0.9$ di un AR(2) e delle stime con Yule-Walker.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ *rand* \ 1	-0.9557	-0.9083	-0.8946	-0.8897	-0.8758	-0.8065
DE \ *best* \ 2	-0.9570	-0.9090	-0.8933	-0.8894	-0.8768	-0.8040
PSO	-0.9569	-0.9087	-0.8940	-0.8893	-0.8753	-0.8068
FA	-0.9521	-0.9082	-0.8940	-0.8891	-0.8758	-0.8065
Yule-Walker	-0.9568	-0.8969	-0.8812	-0.8778	-0.8636	-0.7940

Tabella 5.26: Summary medie - $\Phi_2 = -0.9$ - e stime con Yule-Walker.

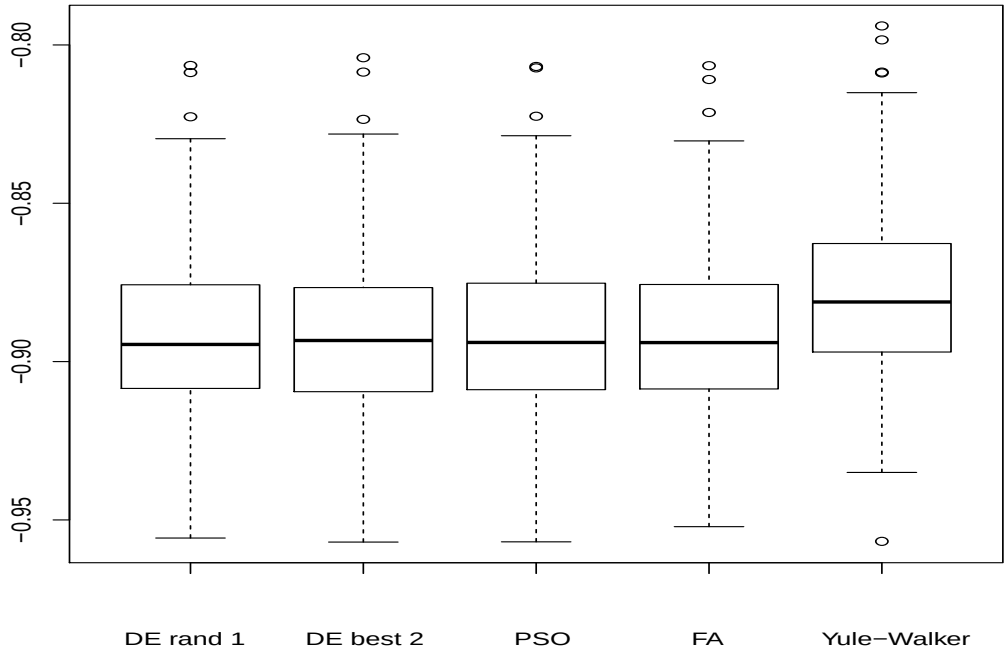

Figura 5.15: Boxplot delle medie delle stime per il primo parametro $\Phi_2 = -0.9$ di un AR(2) e delle stime con Yule-Walker.

	DE \ rand\ 1	**DE \ best \ 2**	**PSO**	**FA**	**Yule-Walker**
Φ_1	0.9023770	0.9176196	0.912004	0.9251369	0.9033657
Φ_2	-0.9087009	-0.8967873	-0.9122879	-0.9196397	-0.8919924

Tabella 5.27: Stime sulla serie iniziale non bootstrappata.

Algoritmo	**Min**	**1° quart.**	**Mediana**	**Media**	**3° quart.**	**Max**
DE \ *rand* \ 1	0.7912	0.8774	0.9000	0.8990	0.9226	0.9871
DE \ *best* \ 2	0.7908	0.8771	0.8987	0.8989	0.9218	0.9925
PSO	0.7721	0.8900	0.9108	0.9084	0.9293	0.9861
FA	0.8453	0.9081	0.9281	0.9251	0.9437	0.9913

Tabella 5.28: Summary medie Φ_1 bootstrap - $\Phi_1 = 0.9$.

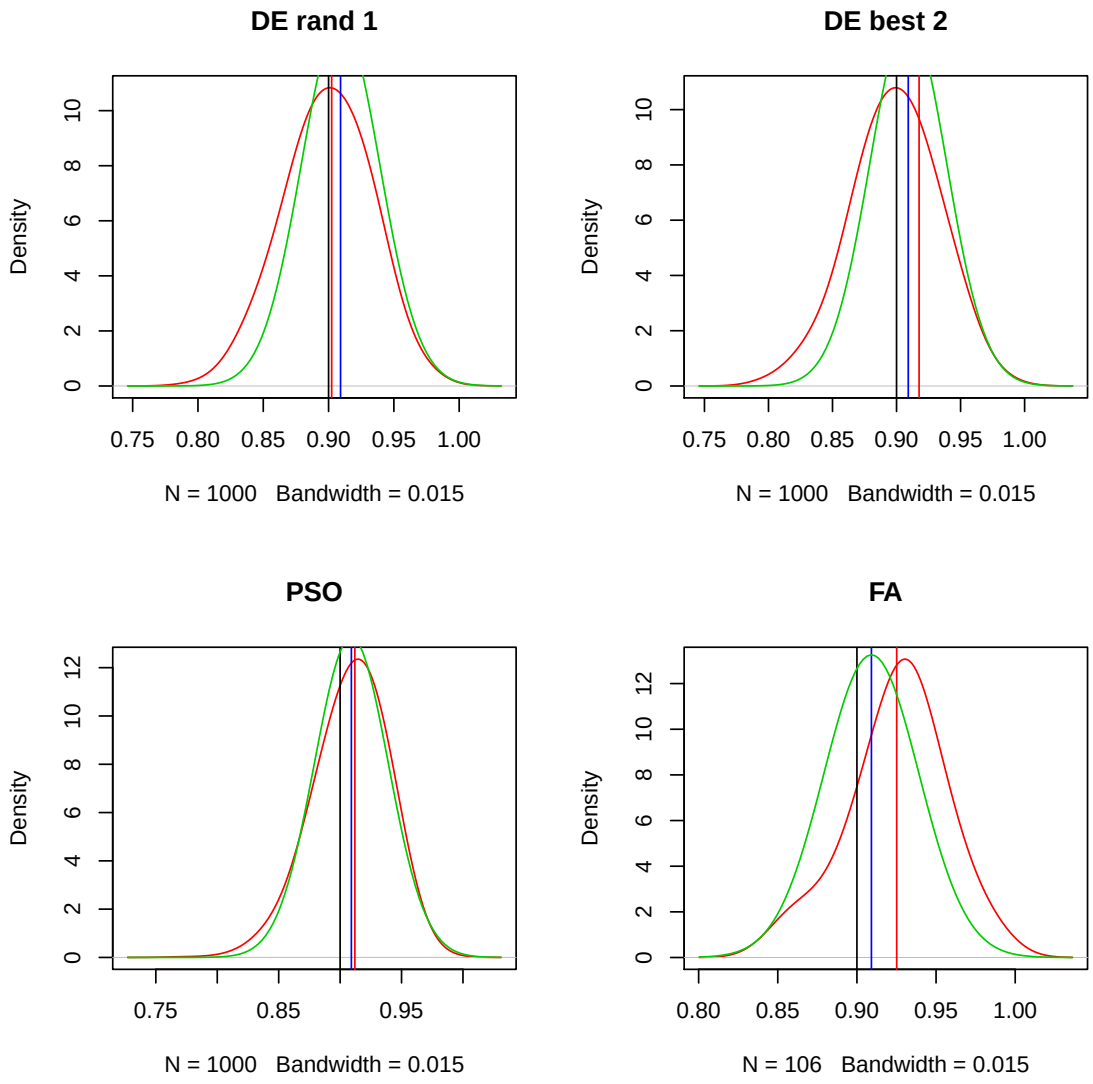

Figura 5.16: Distribuzioni delle stime sulle serie bootstrap per il primo parametro Φ_1 di un AR(2).

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE \ rand \ 1	-0.9783	-0.9231	-0.9040	-0.9001	-0.8797	-0.7610
DE \ best \ 2	-0.9783	-0.9231	-0.9040	-0.9001	-0.8797	-0.7610
PSO	-0.9709	-0.9284	-0.9103	-0.9068	-0.8879	-0.7637
FA	-0.9824	-0.9349	-0.9111	-0.9122	-0.8941	-0.8346

Tabella 5.29: Summary medie Φ_2 bootstrap - $\Phi_2 = -0.9$.

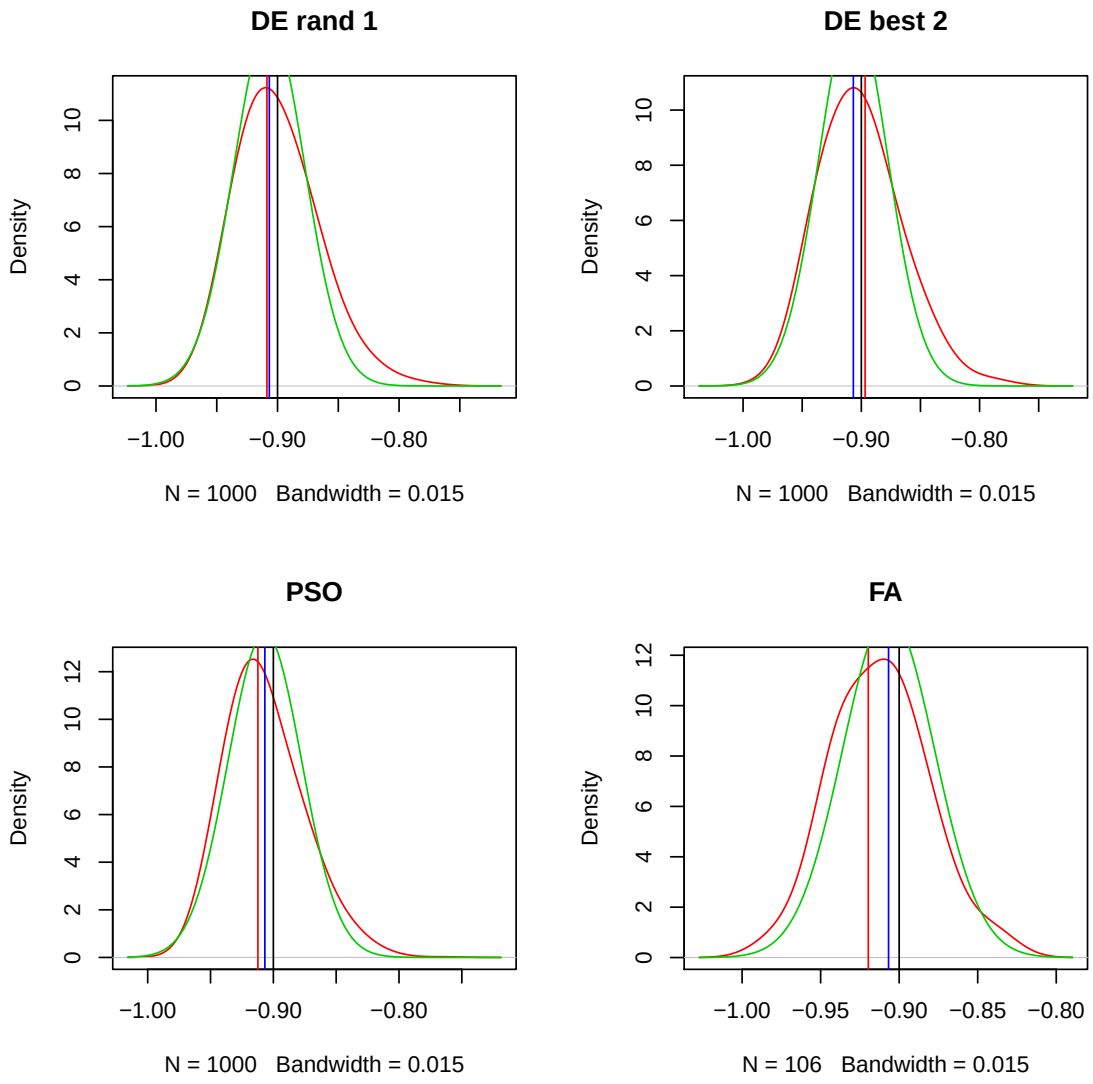

Figura 5.17: Distribuzioni delle stime sulle serie bootstrap per il primo parametro Φ_2 di un AR(2).

Per entrambi i parametri col FA otteniamo il risultato peggiore, seppur non molto distante dal valore ricercato. La stima è molto buona con gli altri tre algoritmi.

Algoritmo	**Intervallo**	σ
$DE \setminus rand \setminus 1$	$0.8346101 < \Phi_1 < 0.9588175$	0.03247947
$DE \setminus best \setminus 2$	$0.82907 < \Phi_1 < 0.9613857$	0.03365022

PSO	$0.8440692 < \Phi_1 < 0.9570904$	0.02911997
FA	$0.8569967 < \Phi_1 < 0.981854$	0.03034353
Yule-Walker	$0.8402395 < \Phi_1 < 0.9664919$	0.03220725

Tabella 5.30: Intervalli di confidenza per Φ_1 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

Algoritmo	**Intervallo**	σ
DE \ *rand* \ 1	$-0.9553116 < \Phi_1 < -0.8235434$	0.03319245
DE \ *best* \ 2	$-0.9582292 < \Phi_1 < -0.8278092$	0.03457108
PSO	$-0.9539174 < \Phi_1 < -0.840184$	0.02988602
FA	$-0.9661128 < \Phi_1 < -0.8451557$	0.02975404
Yule-Walker	$-0.9551186 < \Phi_1 < -0.8288662$	0.03220725

Tabella 5.31: Intervalli di confidenza per Φ_2 e varianze ottenute dalla distribuzione delle stime su serie bootstrap e con Yule-Walker.

Gli intervalli maggiormente ristretti vengono calcolati a partire dalle stime ottenute con le due varianti del Differential Evolution e con il metodo di stima Yule-Walker.

5.4 Applicazione con un SETAR(2,1,1)

Le applicazioni viste finora con i modelli Autoregressivi di primo e secondo ordine ci hanno dato dei buoni risultati. Per avere ulteriori evidenze sulla bontà delle stime calcolate dagli algoritmi euristici trattati in questa tesi, li testeremo non più con dei modelli lineari come fatto in precedenza, ma con un modello non-lineare.

Il modello preso in considerazione rientra nei modelli autoregressivi a soglia (*TAR – Threshold AutoRegressive model*) introdotti per la prima volta da Tong nel 1978 e successivamente sviluppati dallo stesso Tong e da altri autori (Tong-Lim, 1980; Chan e Tong, 1986; Tsay, 1989; Granger e Teräsvirta, 1993; Montgomery, 1998).

L'idea su cui si fondano i modelli a soglia è quella secondo la quale se una variabile appartenente ad una serie ha un comportamento complessivo non-lineare ciò non esclude che in alcuni tratti essa assuma un comportamento lineare.

Il concetto di "linearità a tratti" comporta che la variabile si muova a seconda del tratto in cui si trova; ogni tratto, meglio definito come regime, avrà proprie determinate caratteristiche di media, varianza e autocorrelazione.

In un dato istante t, l'appartenenza della variabile ad un regime piuttosto che ad un altro dipende dalla soglia (o dalle soglie) fissate.

Nel caso in cui in un modello TAR il passaggio della variabile da un regime all'altro dipenda dai valori precedenti della serie, si parla di modello *SETAR* (*Self – Exciting Threshold AutoRegressive model*) [Somera A., 2008].

Nel nostro caso prenderemo in esame un modello SETAR (2,1,1), quindi un modello con due regimi, entrambi AR(1), con un ritardo e una soglia r.

Un generico modello con queste caratteristiche è descritto dalla (5.9):

$$y_t = \begin{cases} \Phi_0^{(1)} + \Phi_1^{(1)} y_{t-1} + \epsilon_t^{(1)} & \text{se} \quad y_{t-1} \leq r \\ \\ \Phi_0^{(2)} + \Phi_1^{(2)} y_{t-1} + \epsilon_t^{(2)} & \text{se} \quad y_{t-1} > r \end{cases} \tag{5.9}$$

in cui $\Phi_0^{(1)}$, $\Phi_1^{(1)}$ e $\Phi_0^{(2)}$, $\Phi_1^{(2)}$ sono i parametri relativi al primo ed al secondo regime, $\epsilon_t^{(1)}$ e $\epsilon_t^{(2)}$ sono due variabili derivanti da un processo *white noise* a media nulla e varianza σ^2 costante spesso ipotizzata gaussiana, ed r è il parametro soglia.

Nell'applicazione che seguirà verrà impostata per le parti stocastiche di entrambi i regimi una varianza $\sigma^2 = 1$.

Per questa parte, non avendo rilevato significative differenze nei risultati tra le due versioni del Differential Evolution, verrà utilizzata unicamente la versione base $DE \setminus rand \setminus 1$.

Essendo infine di difficile applicazione uno stimatore tradizionale per modelli non-lineari, ci si limiterà alla verifica delle stime con gli algoritmi col medesimo procedimento già visto nelle sezioni precedenti di questo capitolo.

5.4.1 Serie con $\Phi_1^{(1)} = 0.75$, $\Phi_1^{(2)} = -0.75$ ed $r = -0.5$

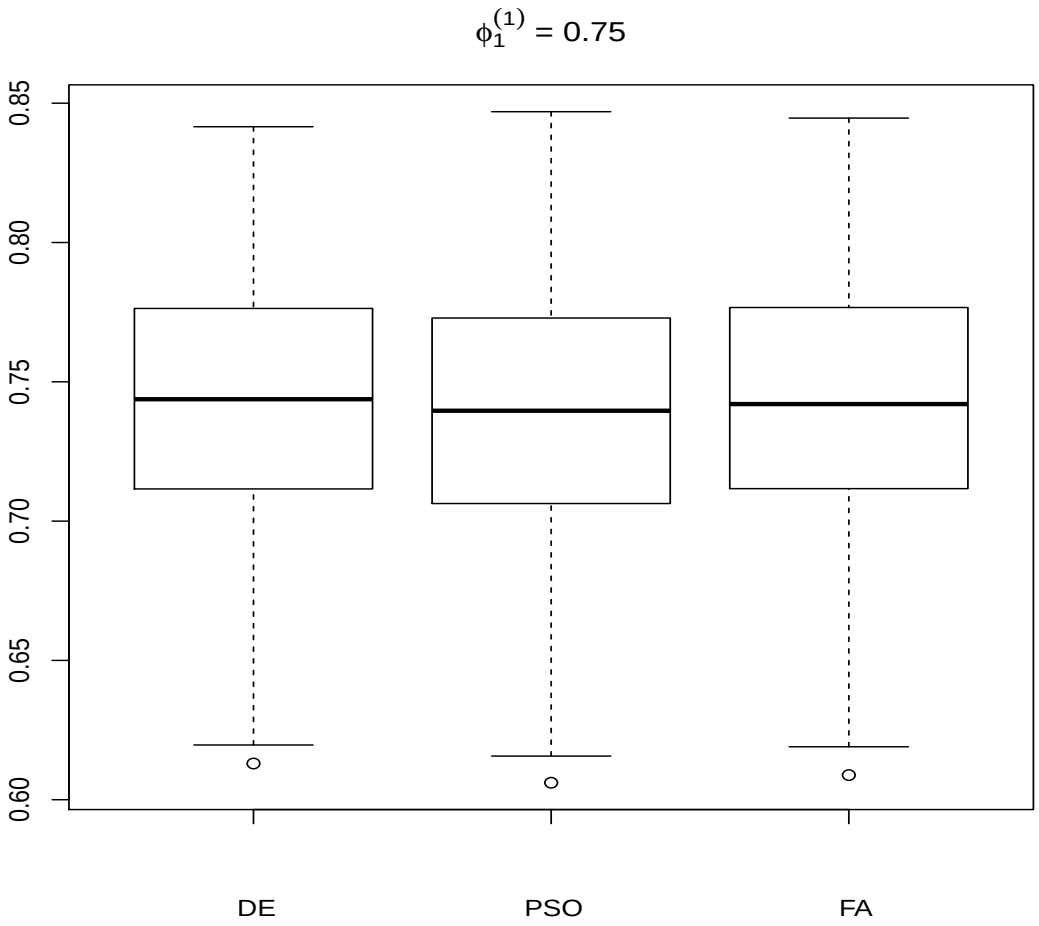

Figura 5.18: Distribuzioni delle medie delle stime sulle cento serie simulate per il parametro $\Phi_1^{(1)}$ del primo regime di un SETAR(2,1,1).

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE	0.6130	0.7121	0.7437	0.7426	0.7748	0.8416
PSO	0.6061	0.7066	0.7396	0.7394	0.7720	0.8470
FA	0.6088	0.7119	0.7420	0.7430	0.7766	0.8447

Tabella 5.32: Summary medie - $\Phi_1^{(1)} = 0.75$.

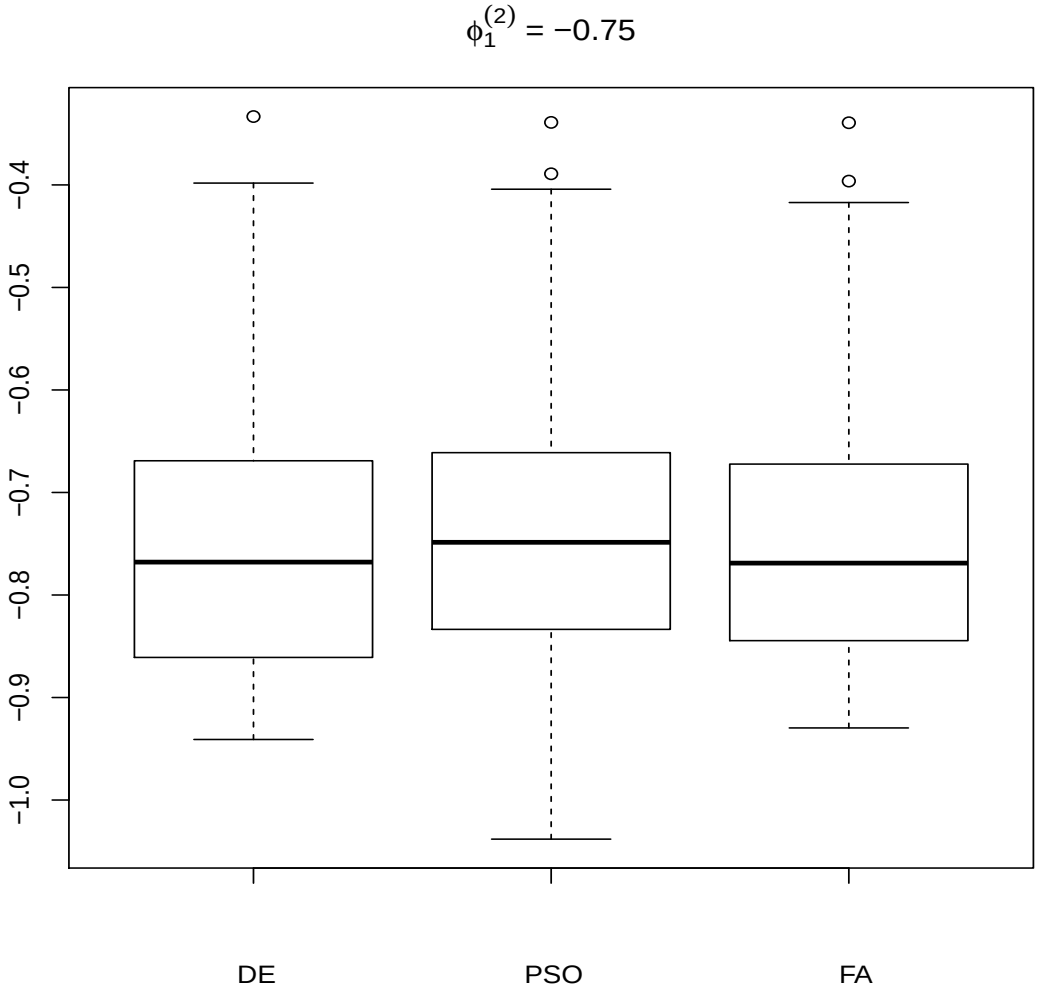

Figura 5.19: Distribuzioni delle medie delle stime sulle cento serie simulate per il parametro $\Phi_1^{(2)}$ del secondo regime di un SETAR(2,1,1).

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE	-0.9409	-0.8606	-0.7679	-0.7514	-0.6693	-0.3333
PSO	-1.0380	-0.8334	-0.7486	-0.7475	-0.6612	-0.3390
FA	-0.9297	-0.8433	-0.7689	-0.7492	-0.6728	-0.3393

Tabella 5.33: Summary medie - $\Phi_1^{(2)} = -0.75$.

Le stime ottenute per i due parametri $\Phi_1^{(1)}$ e $\Phi_1^{(2)}$ sono simili e buone per tutti e tre gli algoritmi. Per la stima della soglia invece, il PSO si rivela meno efficace.

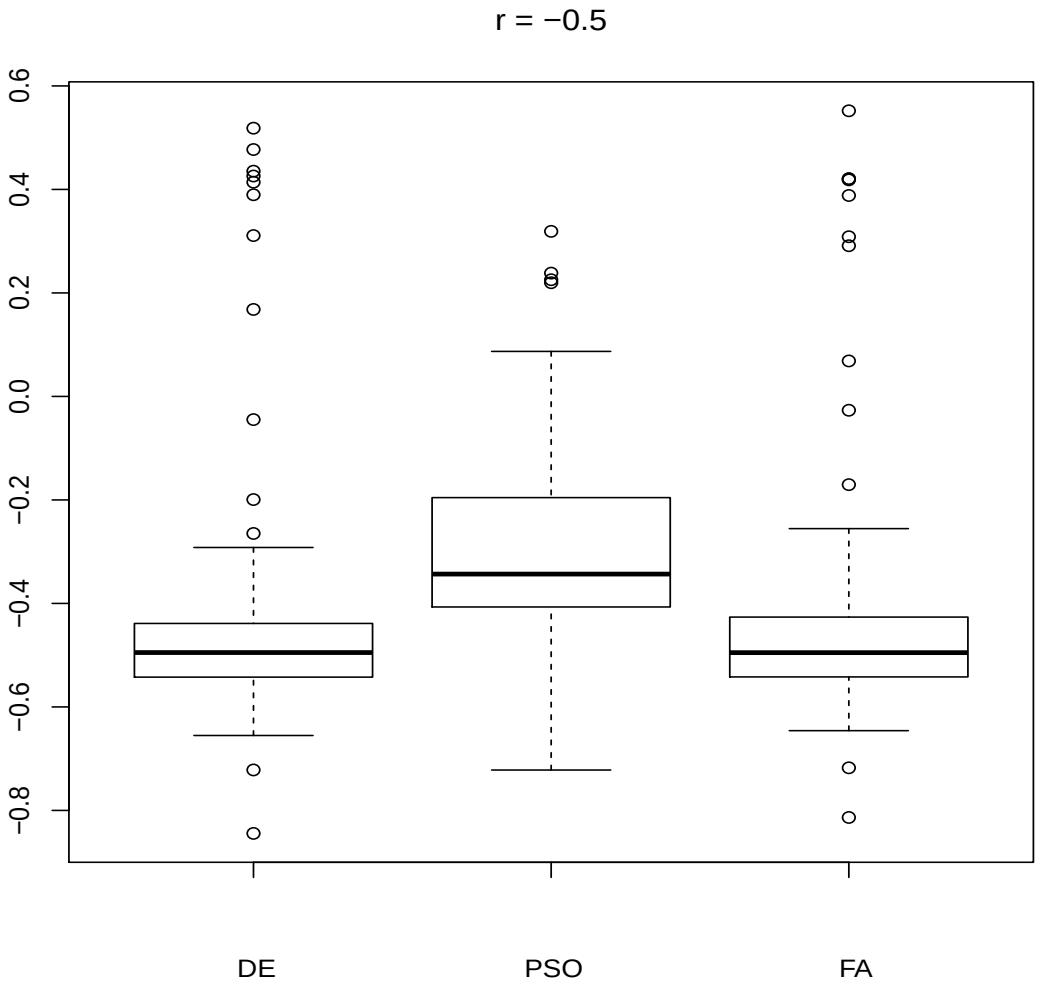

Figura 5.20: Distribuzioni delle medie delle stime sulle cento serie simulate per il parametro *r* indicante la soglia di un SETAR(2,1,1).

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE	-0.8444	-0.5424	-0.4950	-0.4254	-0.4404	0.5184
PSO	-0.7219	-0.4064	-0.3433	-0.2839	-0.1966	0.3190
FA	-0.8136	-0.5417	-0.4951	-0.4207	-0.4269	0.5520

Tabella 5.34: Summary medie stime di $r = -0.5$.

Visti i risultati delle medie delle cento applicazioni sulle cento serie storiche per i tre algoritmi, utilizziamo il metodo bootstrap.

	DE	PSO	FA
Φ_1^1	0.7202753	0.7117615	0.7135358
Φ_1^2	-0.7821964	-0.7872991	-0.7281866
r	-0.5790948	-0.2908507	-0.4689145

Tabella 5.35: Stime sulla serie iniziale non bootstrappata.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE	0.4664	0.6642	0.7083	0.7018	0.7440	0.8715
PSO	0.3747	0.6283	0.6705	0.6663	0.7083	0.8054
FA	0.4508	0.6574	0.6996	0.6948	0.7378	0.8602

Tabella 5.36: Summary $\Phi_1^{(1)}$ bootstrap - $\Phi_1^{(1)} = 0.75$.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE	-0.9992	-0.8850	-0.7845	-0.7784	-0.6897	-0.2310
PSO	-1.2970	-0.8508	-0.7555	-0.7577	-0.6676	-0.2718
FA	-0.9990	-0.8265	-0.7307	-0.7231	-0.6268	-0.2484

Tabella 5.37: Summary $\Phi_1^{(2)}$ bootstrap - $\Phi_1^{(2)} = -0.75$.

Algoritmo	Min	1° quart.	Mediana	Media	3° quart.	Max
DE	-0.9476	-0.5467	-0.4990	-0.4251	-0.4372	0.9796
PSO	-0.83160	-0.39360	-0.24920	-0.23350	-0.08724	0.62320
FA	-0.97096	-0.5431	-0.4974	-0.4093	-0.4230	0.9880

Tabella 5.38: Summary r bootstrap - $r = -0.5$.

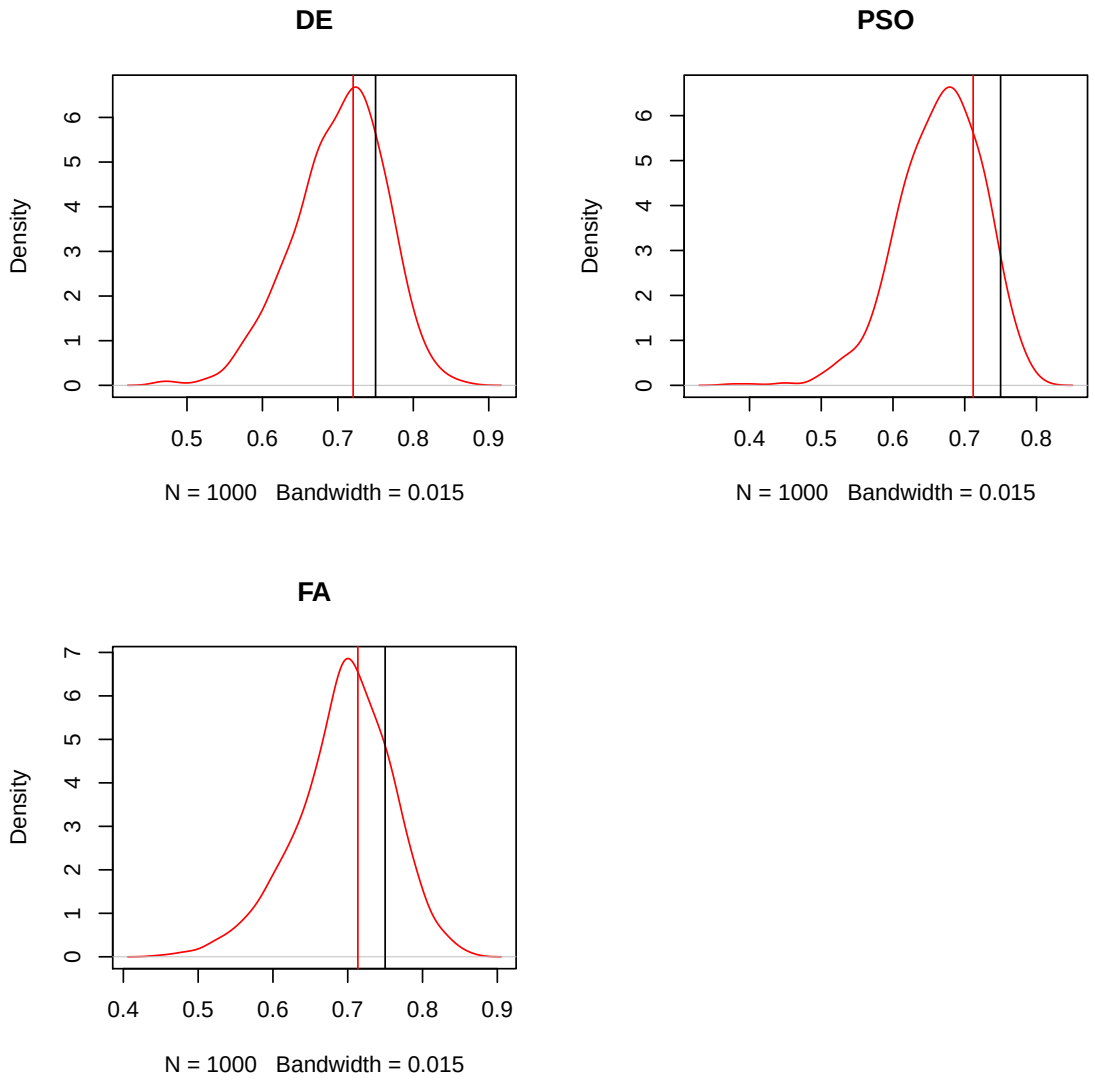

Figura 5.21: Distribuzioni delle stime sulle serie bootstrap per il parametro $\Phi_1^{(1)}$ del primo regime di un SETAR(2,1,1).

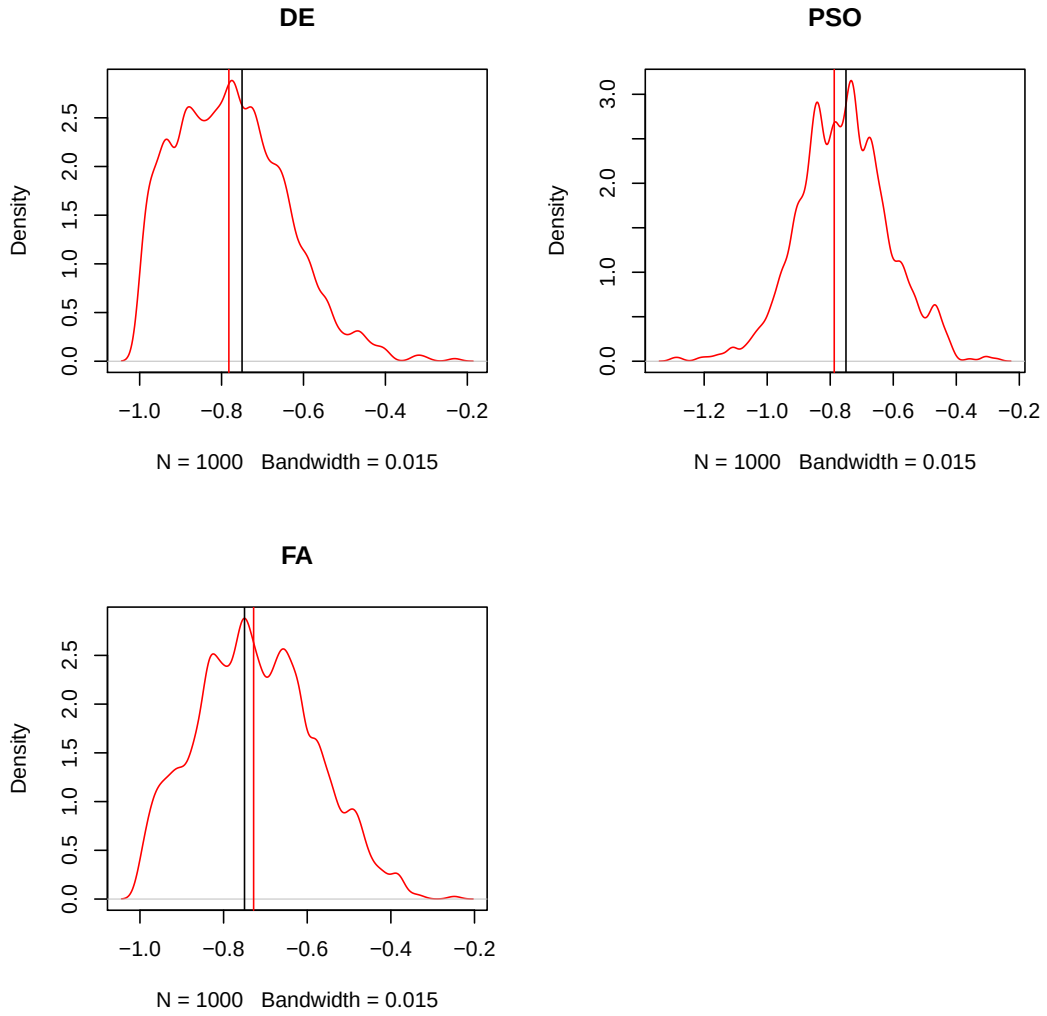

Figura 5.22: Distribuzioni delle stime sulle serie bootstrap per il parametro $\Phi_1^{(2)}$ del secondo regime di un SETAR(2,1,1).

A differenza di quanto visto coi modelli AR(1) e AR(2) in questo caso non abbiamo il confronto con uno stimatore tradizionale, di conseguenza nelle immagini si vedono una curva rossa, che ci mostra una stima mediante kernel della distribuzione dei valori stimati nei quattro casi, una linea verticale rossa in corrispondenza del valore stimato sulla serie simulata ed una linea verticale nera sul valore impostato nella simulazione per il parametri autoregressivi e per la soglia.

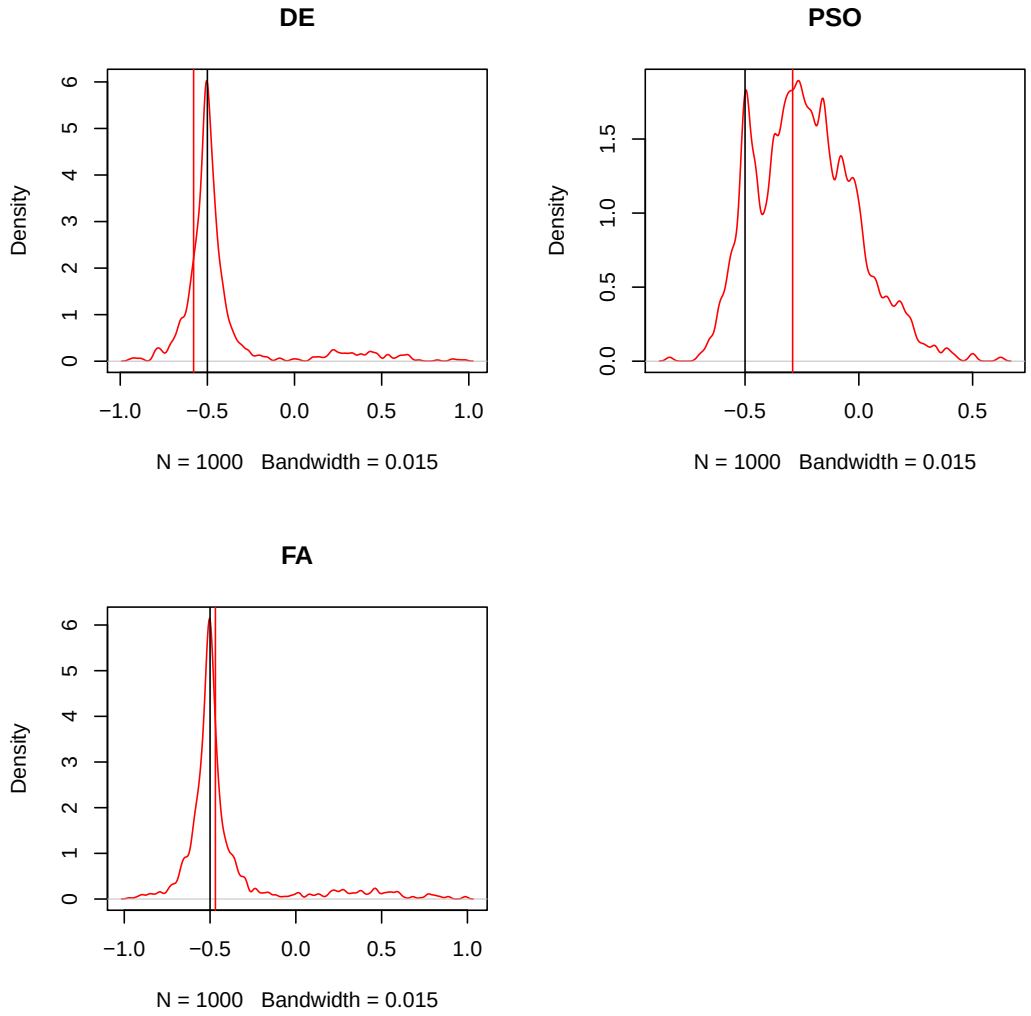

Figura 5.23: Distribuzioni delle stime sulle serie bootstrap per la soglia r di un SETAR(2,1,1).

Algoritmo	**Intervallo**	σ
DE	$0.5737084 < \Phi_1^{(1)} < 0.8066493$	0.06056461
PSO	$0.5375281 < \Phi_1^{(1)} < 0.7654728$	0.05854647
FA	$0.5518934 < \Phi_1^{(1)} < 0.8053581$	0.064073

Tabella 5.39: Intervalli di confidenza per $\Phi_1^{(1)}$ e varianze ottenute dalla distribuzione delle stime su serie bootstrap.

Algoritmo	**Intervallo**	σ
DE	-0.9844306< $\Phi_1^{(2)}$ < -0.4866904	0.132791
PSO	-1.036868 < $\Phi_1^{(2)}$ < -0.4643942	0.143324
FA	-0.9703462 < $\Phi_1^{(2)}$ <-0.4429327	0.1392272

Tabella 5.40: Intervalli di confidenza per $\Phi_1^{(2)}$ e varianze ottenute dalla distribuzione delle stime su serie bootstrap.

Algoritmo	**Intervallo**	σ
DE	-0.723494< r < 0.4903167	0.2851438
PSO	-0.5893161 < r < 0.2263564	0.217871
FA	-0.7364747 < r <0.5568585	0.3065764

Tabella 5.41: Intervalli di confidenza per la soglia r e varianze ottenute dalla distribuzione delle stime su serie bootstrap.

Gli intervalli ottenuti per il parametro del primo regime sono migliori rispetto agli intervalli ottenuti per il parametro del secondo regime e per il parametro soglia. Questo è dovuto alla distribuzione delle stime che hanno una standard deviation più elevata sia per il parametro $\Phi_1^{(2)}$ (σ è circa il doppio rispetto a σ relativa alla distribuzione di $\Phi_1^{(1)}$) che per il parametro soglia (σ è circa il quadruplo rispetto a σ relativo alla distribuzione di $\Phi_1^{(1)}$)

Capitolo 6

Conclusioni

Il fine di questo documento è stato quello di provare a verificare l'adeguatezza di strumenti statistici alternativi per la stima dei parametri in problemi di ottimizzazione. Contestualmente, l'altro scopo è stato il loro confronto, ove possibile, con metodi tradizionali.

I problemi di ottimizzazione o, più in particolare, della stima di parametri, sono di larghissima diffusione e si presentano continuamente in una moltitudine di settori, siano essi per la stima di un parametro o multi-obiettivo. La versatilità dei metodi euristici permette loro di essere quasi sempre una possibile fonte di soluzione, essendo utilizzabili in tutti i contesti in cui si presenta un problema di ottimizzazione, a prescindere dalle caratteristiche specifiche del problema.

In virtù di quanto premesso, la curiosità di andare a toccare con mano quanto i metodi euristici potessero essere preziosi strumenti di calcolo, anche in relazione ai metodi tradizionali, ha dato la principale motivazione alla redazione di questa tesi.

Inoltre, con questo elaborato si è inteso creare un mezzo di diffusione per queste tecniche di ricerca ancora poco diffuse, cercando di "raccontarle" nel modo più semplice ed intuitivo possibile, caratteristiche che peraltro appartengono intrinsecamente ai metodi in questione.

Nella presentazione degli algoritmi si è potuto vedere quali fossero analogie e

differenze nel metodo di ricerca euristica, ed il confronto sui risultati dati dalle applicazioni ha certamente rivelato alcune indicazioni interessanti.

Partendo dal risultato forse meno significativo, la prima cosa da dire riguarda le due versioni proposte del Differential Evolution: in seguito ai risultati applicativi è possibile affermare che tra le due varianti $DE \setminus rand \setminus 1$ e $DE \setminus best \setminus 2$ non esistono particolari differenze, o, per meglio dire, non si ottengono stime migliori con l'uno piuttosto che con l'altro, e questo ha comportato che nella fase di analisi sui modelli a soglia si è preferito non perdere tempo, utilizzando solo la versione di base $DE \setminus rand \setminus 1$.

Ad ogni modo, le evidenze più importanti riguardano il confronto coi metodi di stima tradizionali: la conclusione principale è che i metodi di stima euristici, rappresentati nel nostro caso dai tre algoritmi scelti, rappresentano senza ombra di dubbio un valido strumento alternativo nella risoluzione di problemi di ottimizzazione ed in particolar modo nella stima di parametri per la definizione dell'andamento di serie storiche. Ciò è stato messo in risalto sia per quanto riguarda gli AR(1) che per gli AR(2), modelli nei quali le stime ottenute su svariati valori dei parametri, possono dirsi generalmente precise tanto quanto quelle ottenute col metodo Yule-Walker, ed anzi, non sono affatto mancati i casi in cui fossero addirittura più vicine ai parametri impostati i risultati degli algoritmi piuttosto di quelli raggiunti col metodo classico.

Venendo ai modelli non-lineari, i risultati sono stati ancora soddisfacenti. Le stime sono state migliori con DE e Fa rispetto al PSO ma ciò che importa evidenziare è che gli algoritmi hanno funzionato anche quì. Purtroppo, per problemi di tempistiche, non è stato possibile né il confronto con metodi più tradizionali né una più esaustiva ricerca, con applicazioni su simulazioni di ulteriori serie storiche con altri parametri oltre a quella proposta, ma l'indicazione di massima è che pure in modelli non-lineari gli approcci utilizzati rappresentino una valida via verso la soluzione ricercata.

Infine due parole vanno spese per la velocità di esecuzione degli algoritmi: coi modelli lineari la rapidità di calcolo è stata maggiore per il DE ed il PSO, ed il FA il più lento, mentre coi modelli non-lineari il PSO si è rivelato il più lento, il DE il più veloce ed il FA nel mezzo; va detto però che questo è dipeso anche da come gli algoritmi sono stati implementati, dunque sono risultanze eventualmente confutabili.

In ultima istanza, i metodi presentati sono senz'altro strumenti alternativi di ricerca che possiedono ottime capacità di risoluzione dei problemi, e ciò che si auspica è che essi vengano sempre più utilizzati ed approfonditi.

Bibliografia

Ardia D., Boudt K., Carl P., Mullen K.M., Peterson B.G., 2011, Differential Evolution with DEoptim, *R Journal*, 3(1), http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Ardia~et~al.pdf

Ardia D., Mullen K.M., Ulrich J, Peterson B.G. , *DEoptim: An R Package for Global Optimization by Differential Evolution*, http://127.0.0.1:17563/library/DEoptim/doc/DEoptim.pdf

Berto S., 2011, *Nuove tecniche di ottimizzione: Evolution Strategies e metodo Cross-Entropy a confronto*, Tesi di laurea magistrale in Statistica per l' Impresa, Università Ca' Foscari di Venezia

Bonabeau E., Dorigo M., Theraulaz G., 1999, Swarm Intelligence: From Natural to Artificial System, *Oxford University Press*, New York

Bonanomi A., 2007, Intervalli di confidenza bootstrap: una veduta d'insieme e una proposta per un indice di cograduzione, *Quaderno di Dipartimento N.3*, 133, 1-26, http://hdl.handle.net/10807/20037

Cherubini U., Della Lunga G., 2001, *Il Rischio Finanziario*, The McGraw-Hill Company

De Giovanni L., *Metodi euristici di ottimizzazione combinatoria*, Università di Padova, http://www.math.unipd.it/~luigi/courses/metmodoc1011/m08.meta.pdf

Dorigo M., Birattari M., 2007, Swarm Intelligence, *Scholarpedia*, 2(9), 1462, revision n.138640, `http://www.scholarpedia.org/article/Swarm_intelligence`

Eiben A.E., Smith J.E., 2004, *Introduction to Evolutionary Computing*, Springer, 15 - 35

Kennedy J., Eberhart R.C., 2001, *Swarm Intelligence*, Morgan Kaufmann

Klir G., 1991, Generalized information theory, *Fuzzy Sets and Systems*, 40(1), 127-142.

Morana M.T., Porcu M., *Il bootstrap. Un'applicazione informatica per un problema di ricampionamento*, `http://veprints.unica.it/444/1/q1_02.pdf`

Pereira G., 2011, Particle Swarm Optimization, *INESC- ID and Instituto Superior Técnico*, `http://web.ist.utl.pt/gdgp/VA/data/pso.pdf`

Seah C.W., Ong Y.S., Tsang I.W., Jiang S., 2012, Pareto Rank Learning in Multi-objective Evolutionary Algorithms, *Evolutionary Computation (CEC)*, 1-8

Somera A., 2008, *Test di non linearità: un'esplorazione Monte Carlo*, Tesi di laurea specialistica in Scienze Statistiche, Economiche, Finanziarie ed Aziendali, Università degli studi di Padova

Storn R., Price K., 1997, Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization*, 11 , 341 - 359

Tan Y., Zhu Y., 2010, Fireworks Algorithm for Optimization, *Advances in Swarm Intelligence*, 335-364

Varian H., 2005 , Bootstrap Tutorial, *Mathematica Journal*, 9, 768-775, http://www.mathematica-journal.com/issue/v9i4/contents/BootstrapTutorial/BootstrapTutorial.pdf?origin=publication_detail

Wang C.X., Li C.H., Dong H., Zhang F., 2013, An Efficient Differential Evolution Algorithm For Function Optimization, *Information Technology Journal*, 12, 444-448, http://scialert.net/abstract/?doi=itj.2013.444.448

Weise T., 2009, *Global Optimization Algorithms - Theory and Application*, http://www.it-weise.de/projects/book.pdf

Ringraziamenti

Ringrazio la mia famiglia per avermi supportato, il professor Pizzi per la disponibilità senza la quale non avrei potuto finire questo percorso, la Fondazione Iglesias e Giulia.