

POLITECNICO DI TORINO

Corso di Laurea
in Ingegneria Matematica

Tesi di Laurea Magistrale

**Modellizzazione della distribuzione della
popolazione tra città su reti spaziali mediante la
teoria cinetica dei sistemi multiagente**



Relatore
prof. Andrea Tosin

firma del relatore

.....

Candidato
Valerio Taralli

firma del candidato

.....

Anno Accademico 2025-2026

Ai miei genitori,
Elisabetta e Marco

SOMMARIO

La corrente tesi studia la distribuzione della popolazione mediante la teoria cinetica dei sistemi multiagente su reti spaziali, interpretando le città come agenti/vertici e rappresentando le connessioni interurbane come lati. Si propongono varie regole d'emigrazione per regolare le interazioni tra agenti, ricavando due principali risultati: l'adattamento lognormale bimodale della distribuzione della popolazione tra le città e l'adattamento di Pareto della coda. Nel complesso il modello restituisce dei risultati coerenti colla distribuzione reale dedotta dai dati ISTAT della Sardegna; per dipiù l'adattamento lognormale bimodale segue meglio la coda della distribuzione rispetto a quello di Pareto, sebbene nella prima metà coincidano.

ABSTRACT

The current thesis investigates population distribution through the kinetic theory of multi-agent systems on spatial networks, interpreting cities as agents/vertices and representing interurban links as edges. Various emigration rules have been proposed to regulate the interactions between agents, yielding two primary results: the bimodal log-normal fit of the total population distribution among cities and the Pareto fit of the tail. Overall, the model provides results consistent with the actual distribution derived from ISTAT data of Sardinia; furthermore, the bimodal log-normal fit follows the distribution tail more accurately than the Pareto one, although the two coincide in the first half.

INDICE

1 INTRODUZIONE	1
2 NOTE DI TEORIA DEI GRAFI	5
2.1 Definizioni miscellanee	5
2.2 Reti e città	6
2.3 Cenni sui dati	7
3 TEORIA CINETICA DEI SISTEMI MULTIAGENTE	13
3.1 Definizioni preliminari di probabilità	13
3.2 Descrizione cinetica di [tipo] Boltzmann	16
3.2.1 Equazione di Boltzmann omogenea	16
3.2.2 Equazione di Boltzmann disomogenea	21
3.2.3 Equazione di tipo Boltzmann omogenea	22
3.3 Descrizione cinetica urbana su reti	27
3.3.1 Equazione di tipo Boltzmann esatta	27
3.3.2 Equazione di tipo Boltzmann approssimata	33
3.3.3 Altri nessi distinguibile-indistinguibile	38
4 SIMULAZIONI	41
4.1 Metodo Monte Carlo	41
4.2 Rappresentazione dei risultati	42
4.2.1 Intervalli di confidenza	42
4.2.2 Struttura dei dati	43
4.2.3 Definizione dei grafici	45
4.2.4 Sulle fluttuazioni γ	49
4.3 Regole d'emigrazione	49
4.3.1 Regola taglia	51
4.3.2 Regola taglia-gradi	51
4.3.3 Regola frazionata	51
4.3.4 Regola taglia-forza	51
4.3.5 Interpretazioni	57
4.4 Altri casi notevoli	57
4.4.1 Varianti delle regole d'interazione	57
4.4.2 Il caso dell'Italia	57
4.4.3 Il caso della Valle d'Aosta	57
5 CONCLUSIONI	59
APPENDICE	61
A Codice	61
ELENCO DELLE FIGURE	123
ELENCO DELLE TABELLE	125
ELENCO DEI LISTATI	127
ELENCO DEGLI ALGORITMI	127
BIBLIOGRAFIA	129

1

INTRODUZIONE

Lo studio della distribuzione della popolazione tra le città è stato un fenomeno già analizzato [sporadicamente] in passato. Il primo fu Auerbach [1] che all'inizio del 19-esimo secolo notò una caratteristica poi formalizzata successivamente da Zipf¹ [19] quasi cinquant'anni dopo, seppure inizialmente in un contesto linguistico; difatti la legge di Zipf è una legge empirica di forma

$$f \propto \frac{1}{r} \quad \text{ove} \begin{cases} f \text{ è la frequenza della parola,} \\ r \text{ è il suo rango nella classifica;} \end{cases} \quad (1.1)$$

per dare un esempio concreto, se la 10 parola più frequente compare 2000 volte allora esiste una costante C tale che $2000 \approx C/10$. La stessa legge descritta in (1.1) si può ritrovare in molt'altri contesti, tra i quali figura la distribuzione della popolazione tra le città se in luogo di f si scrive la popolazione s [19, Fig. 9-2, p. 375]; tuttavia ciò è vero solo qualora $s \gg 1$, ossia nella coda della distribuzione, contrariamente al contesto linguistico originale di Zipf ove (1.1) vale sempre.

Più in generale la legge [empirica] di Zipf può essere descritta da una distribuzione di Pareto: sia S la variabile aleatoria legata alla taglia delle città e sia $f_S(s)$ la sua funzione di densità di probabilità², la quale permette d'interpretare $f_S(s)ds$ come il numero di città con $s \in (s, s+dt)$; allora si dice che S segue una distribuzione di Pareto nella coda se soddisfa

$$R(s) \equiv \int_s^{+\infty} f_S(t)dt \approx \frac{c}{s^\beta} \quad \text{se } s \gg 1, \quad (1.2)$$

per una certa costante $c \in \mathbb{R}_+$ e con $R(s)$ uguale alla funzione di ripartizione complementare di S : essa rappresenta matematicamente il numero di città con popolazione maggiore o uguale a s e, qualora l'indice di Pareto β fosse unitario, è analoga al rango r nella (1.1).

Più recentemente [8, 9] si è scoperto che al di fuori della coda la $f_S(s)$ è ben fittata dalla distribuzione lognormale con densità di probabilità

$$f_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right), \quad (1.3)$$

ove $X \sim \text{Lognormale}(\mu, \sigma)$; come suggerisce il nome, la peculiarità di tale distribuzione è che vale $\log X \sim \mathcal{N}(\mu, \sigma)$ (Fig. 1.1) in cui $\mathcal{N}(\mu, \sigma)$ indica la distribuzione gaussiana.

¹In realtà, come Zipf stesso ammette [19, p. 546], non fu il primo a scoprire tale legge; in ogni caso la popolarizzò a tal punto che oggi è conosciuta col suo nome.

²Affinché $f_S(s)$ esista bisogna rigorosamente anche supporre che S sia assolutamente continua, ma qui non è necessario entrare nei dettagli per i quali si rimanda ai §§ 3.1 e 3.3.

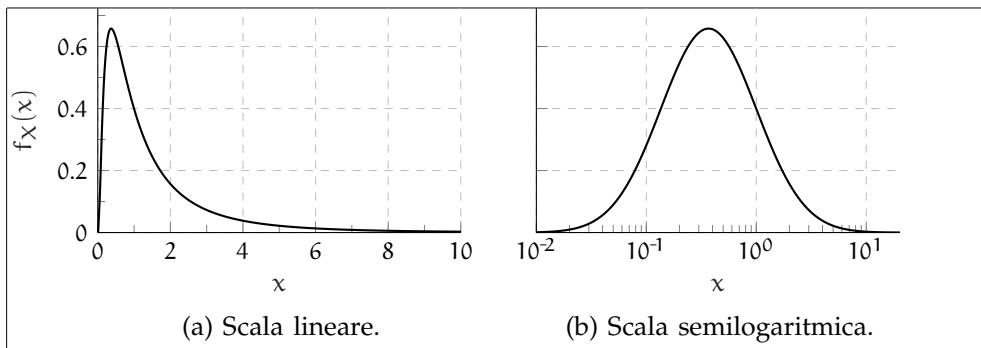


Figura 1.1: Funzione Lognormale(0,1).

Perdipiú da Gualandi *et al.* [9, 10] ci si è poi resi conto che l’intera distribuzione è ben fittata da una lognormale bimodale, vale a dire una combinazione convessa della (1.3)

$$f_S(s) \approx f_X(s) = \xi f_{S_1}(s) + (1 - \xi) f_{S_2}(s), \quad (1.4)$$

ove $\xi \in (0,1)$, $S_1, S_2 \sim \text{Lognormale}(\mu, \sigma)$ e $X \sim \text{BiLognormale}(\xi, \mu_1, \sigma_1, \mu_2, \sigma_2)$.

Pertanto l’obbiettivo di questa tesi è il seguente: si vuole simulare attraverso la Teoria Cinetica dei Sistemi MultiAgente (**TCSMA**) le interazioni tra città su un grafo spaziale, tentando di riprodurre una distribuzione della popolazione con caratteristiche analoghe a quelle realmente misurate (principalmente corpo lognormale e coda di Pareto); una volta raggiunto questo scopo si potrà poi meditare sulle implicazioni della legge d’interazione riguardo al fenomeno dell’emigrazione, che è alla base delle interazioni tra città.

Si sottolinea che qui le città verranno invece considerate come agenti, caratterizzati dalla loro popolazione, che interagiscono attraverso la struttura sottostante di un grafo [spaziale]; tale descrizione, salvo il grafo, non è affatto dissimile a quella classica delle molecole di un gas caratterizzate dalla loro velocità e posizione.

Difatti, in letteratura, perlomeno quella a conoscenza dell’autore, non esistono articoli che trattano contemporaneamente le teorie cinetiche, la distribuzione della città e i grafi, nonostante la rappresentazione di queste su un grafo sia del tutto naturale; più nel dettaglio gli articoli in letteratura

- ◊ o applicano la **TCSMA** alle città senza considerare una naturale topologia sottostante [10]³ o, in senso opposto, considerano la struttura da grafo senza applicare la **TCSMA** [4];
- ◊ altri vedono i nodi piú come luoghi ove vivono e interagiscono gli agenti, creando cosí un modello descritto da un sistema di equazioni di Boltzmann elevate e accoppiate [15];
- ◊ infine, l’articolo che piú si avvicina all’obbiettivo di questa tesi applica, tuttavia, la sua teoria nel contesto delle reti sociali [17].

Questa lacuna è probabilmente attribuibile al fatto che gli sviluppi della teoria dei grafi applicata alla **TCSMA** sono stati piuttosto recenti e prin-

³ In particolare [10], nonostante sia un articolo molto interessante e che giunge a conclusioni altrettanto importanti, astrae eccessivamente le interazioni tra città oltre a non definire con sufficiente chiarezza che cosa s’intende per *dintorni*.

60 cipalmente concentrati sulla prospettiva nodo-luogo anziché nodo-agente.
61 Dunque l'aspetto piú innovativo di questo lavoro è mostrare che la pro-
62 spettiva nodo-agente è altrettanto valida quanto quella piú comune e re-
63 cente del nodo-luogo; per il resto questa tesi dev'essere considerata come
64 un'esplorazione formale applicativa con pochi approfondimenti analitici.

65 Lo scritto sarà cosí suddiviso: nel secondo capitolo si affronteranno breve-
66 mente e [superficialmente] alcuni concetti della teoria dei grafi, soprattutto
67 quelli pertinenti alla topologia interurbana; quindi nel terzo la **TCSMA** è ap-
68 profondita prima da un punto di vista classico, e poi mediata dai grafi sia
69 esattamente che approssimativamente; infine negli ultimi due si illustreranno
70 i principali risultati concludendo con delle note finali e potenziali sviluppi
71 futuri.

72 Per il lettore interessato è anche presente un'appendice ove il codice di
73 Python dell'implementazione è spiegato a grandi linee.

2

NOTE DI TEORIA DEI GRAFI

In questo capitolo sono prima introdotti alcuni oggetti di base della teoria dei grafi, quindi si discute la natura della rete d'interesse, infine si descrivranno molto brevemente i dati usati.

2.1 DEFINIZIONI MISCELLANEE

Innanzitutto s'inizia dando la definizione di rete:

Definizione 2.1 (Grafo). Un grafo è formato dalla coppia $\mathcal{G} = (\mathcal{I}, \mathcal{E})$, ove \mathcal{I} è l'insieme degl'indici dei nodi mentre \mathcal{E} è l'insieme dei lati, vale a dire di coppie d'indici \mathcal{I} : due nodi $i, j \in \mathcal{I}$ sono connessi sse $(i, j) \in \mathcal{E}$.

Dall'insieme \mathcal{E} si può poi specializzare il concetto di grafo:

Definizione 2.2 (Grafo [in]diretto). Un grafo è indiretto sse dato $(i, j) \in \mathcal{E}$ allora $(j, i) \in \mathcal{E}$ e la direzione è trascurabile; altrimenti è detto diretto.

La trascurabilità della direzione sarà discussa poco dopo nella § 2.2, anche se è facilmente intuibile dalla Fig. 2.1.

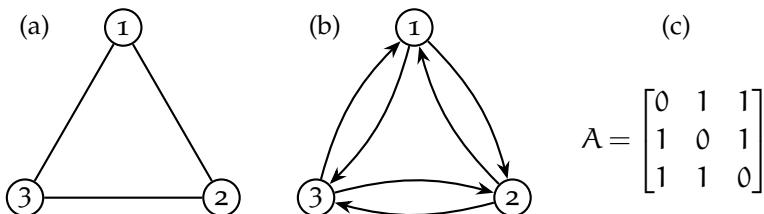


Figura 2.1: Esempio di un grafo indiretto, della sua equivalente forma diretta [simmetrica] e della loro [identica] matrice d'adiacenza.

Definizione 2.3 (Matrice d'adiacenza). Sia $N \equiv |\mathcal{I}| \in \mathbb{N}_+$ la cardinalità dell'insieme degl'indici, ossia il numero di vertici totali, si definisce la matrice d'adiacenza $\mathbf{M} \in \mathbb{R}^{N \times N}$ pesata come:

$$m_{i,j} \equiv \begin{cases} q_{i,j} & \text{se } (i,j) \in \mathcal{E}, \\ 0 & \text{altrimenti,} \end{cases} \quad (2.1)$$

ove $q_{i,j} \in \mathbb{R}$ è il peso associato al lato (i,j) ; nel caso in cui $q_{i,j} \equiv 1 \forall i,j \in \mathcal{I}$ si ha la matrice d'adiacenza unitaria $\mathbf{A} \in \mathbb{R}^{N \times N}$:

$$a_{i,j} \equiv \begin{cases} 1 & \text{se } (i,j) \in \mathcal{E}, \\ 0 & \text{altrimenti.} \end{cases} \quad (2.2)$$

Si osserva che, se non altrimenti detto, scrivendo solo «matrice d'adiacenza» si sottintende sempre quella unitaria.

Definizione 2.4 (Matrice trasposta). Data $\mathbf{M} \in \mathbb{R}^{N \times N}$, con $N \in \mathbb{N}_+$, allora s'indica \mathbf{M}^\top la matrice trasposta così definita:

$$m_{i,j}^\top = m_{j,i} \quad \forall i, j \in \{1, 2, \dots, N\}.$$

Definizione 2.5 (Matrice simmetrica). Data $\mathbf{M} \in \mathbb{R}^{N \times N}$, con $N \in \mathbb{N}_+$, allora

$$\mathbf{M} \text{ è simmetrica} \iff m_{i,j} = m_{j,i} = m_{i,j}^\top \iff \mathbf{M} = \mathbf{M}^\top. \quad (2.3)$$

Osservazione 2.1. Nei grafi indiretti sia \mathbf{M} che \mathbf{A} sono simmetriche mentre per quelli diretti non è detto lo siano.

Definizione 2.6 (Gradi e Forze). In un grafo diretto e pesato si definiscono la forza entrante e uscente come rispettivamente le quantità

$$w_i^- \equiv \sum_{j=1}^{|J|} m_{j,i} \quad \text{e} \quad w_i^+ \equiv \sum_{j=1}^{|J|} m_{i,j}, \quad \forall i \in J, \quad (2.4)$$

le quali se il grafo è simmetrico queste coincidono: $\mathbf{w} = \mathbf{w}^- = \mathbf{w}^+$. Nel caso unitario $\mathbf{M} \equiv \mathbf{A}$ si è soliti denominare le forze come gradi uscenti ed entranti:

$$k_i^- \equiv \sum_{j=1}^{|J|} a_{j,i} \quad \text{e} \quad k_i^+ \equiv \sum_{j=1}^{|J|} a_{i,j}, \quad \forall i \in J, \quad (2.5)$$

che, come prima, in una rete simmetrica coincidono: $\mathbf{k} = \mathbf{k}^- = \mathbf{k}^+$.

2.2 RETI E CITTÀ

Dopo aver illustrato un po' di teoria sui grafi sorge successivamente il problema di come rappresentare con essi la moltitudine di collegamenti possibili tra le città

Difatti vi sono molti modi di rappresentare le città mediante i grafi: i primi si pongono a livello *intraurbano*, o considerando le strade come lati e le loro intersezioni come nodi [3, § 3.1.3.1 p. 17], o rappresentando la rete di trasporto di tram, di bus e della metro [3, § 3.2.1 p. 22]; altri si pongono più propriamente a livello *interurbano* prendono singolarmente in considerazione varie reti dei trasporti (ferroviario [3, § 3.1.3.2 p. 17], navale [3, § 3.1.4 p. 19], aereo [3, § 3.1.2 p. 13], ecc.).

Tuttavia, da quanto detto nella Cap. 1, è chiaro che per predire la distribuzione della popolazione tra città valgono le seguenti osservazioni:

1. ogni rappresentazione intraurbana va scartata perché sono troppo fini, oltre a considerare movimenti limitatamente a una sola città;
2. d'altra parte tutte quelle interurbane vanno considerate contemporaneamente e non singolarmente siccome ciascun individuo può scegliere diversi trasporti per muoversi.

Ecco perché la corretta rete da considerare è quella legata ai movimenti pendolari [3, § 3.1.3.3 p. 18; 6] tra città che mostrano olisticamente tutti i possibili collegamenti tra le città a prescindere del trasporto scelto; inoltre

126 essa mostra collegamenti realistici associati a movimenti quotidiani anziché
 127 straordinari (vacanze, visite mediche, ecc.).

128 Una volta fissata la rappresentazione è necessario comprendere il tipo di
 129 grafo con cui si ha a che fare. Eppure qui la risposta è immediata dopo due
 130 osservazioni:

- 131 1. una città può interagire con un'altra senza che quest'ultima interagi-
 132 sca colla prima: è necessario considerare il senso di direzione;
- 133 2. ammesso che una città possa interagire con un'altra, allora è sempre
 134 possibile l'opposta: le potenziali interazioni sono simmetriche.

135 Dunque il grafo in questione è diretto ma con struttura indiretta (Def. 2.2),
 136 ovvero la sua matrice d'adiacenza A è simmetrica.

137 **Ipotesi 2.1.** Il grafo G è diretto e simmetrico (A è simmetrica).

138 Inoltre in questa trattazione vale la seguente fondamentale ipotesi:

139 **Ipotesi 2.2.** Il grafo G è statico: \mathcal{J} e \mathcal{E} sono costanti nel tempo.

140 In altre parole si è solo interessati a prevedere come la popolazione si di-
 141 stribuisce rispetto a una topologia prestabilita *a priori*; ovviamente si tratta
 142 di una forte semplificazione dato che nella realtà la topologia delle connes-
 143 sioni interurbane si è chiaramente coevoluta assieme allo sviluppo delle città
 144 stesse.

145 Infine si osserva che questo tipo di grafo è spaziale [3, 6], ovvero i suoi
 146 nodi occupano un punto nello spazio euclideo, seppure, al momento, sia una
 147 caratteristica alquanto formale; tuttavia servirà tenerlo a mente quando si
 148 definiranno alcune regole d'interazione nel Cap. 4. Perdipiù, contrariamente
 149 a quanto si possa pensare di primo acchito, tale grafo non è a invarianza di
 150 scala [2] a causa del tipo di grafo [3, 5]; ciò non esclude l'esistenza di nodi
 151 più centrali di altri, ma solo che il massimo grado di un nodo è limitato
 152 superiormente proprio dalla natura spaziale del grafo.

153 2.3 CENNI SUI DATI

154 La matrice di pendolarismo costruita è quella fornita dall'ISTAT del 1991
 155 [14] seguendo l'esempio di [6]. I dati sono di fatto un file di testo formato da
 156 una serie di righe di 29 numeri il cui significato è spiegato dalla Tab. 2.1.

157 I comuni considerati sono tutti quelli italiani (8100) nel 1991, quindi è ne-
 158 cessario estrapolare da essi le matrici di pendolarismo delle singole regioni
 159 e parallelamente quella complessiva dell'Italia. Si possono definire due tipi
 160 di matrici:

- 161 1. quella d'adiacenza unitaria in cui un lato esiste se almeno un pendo-
 162 lare (cioè il «[n]umero di persone» nella Tab. 2.1) si muove tra i due
 163 comuni e
- 164 2. quella d'adiacenza pesata che è analoga alla precedente ma con peso
 165 $q_{i,j}$ uguale alla somma totale dei pendolari tra due comuni.

¹ si v. il documento trapen91.txt per maggiori informazioni.

Dato	Col. iniziale	Lunghezza
Provincia di partenza	1	3
Comune di partenza	4	3
Sesso	7	1
Mezzo di trasporto	8	1
Cond. professionale	9	1
Orario di uscita	10	1
Tempo di percorrenza	11	1
Provincia di arrivo	12	3
Comune di arrivo	15	3
Numero di persone	18	12

Tabella 2.1: Formattazione di una sola riga nei dati ISTAT¹; le linee barrate corrispondono a dati trascurati.

166 Si notano tuttavia due principali difetti dei dati contenuti nel documento
 167 Pen_91It.txt:

- 168 1. in alcune linee come comune di destinazione è segnato il codice
 169 «022008» che però non è elencato nel documento elencom91.xls che
 170 riassume tutti gli 8100 comuni italiani nel 1991;
 171 2. in molte linee come comune di destinazione sono segnati dei codici
 172 incompleti contenenti solo l'ultima metà (codice comune) ma non
 173 la prima (codice provincia), e questi sono in totale 8: «215», «229»,
 174 «241», «216», «203», «224», «236», «246».

175 In entrambi i casi i dati relativi [a quelli che sono potenzialmente refusi]
 176 sono stati trascurati nella corrente tesi; il primo errore è già di per sé molto
 177 raro, mentre il secondo è eccessivamente ambiguo da risolvere come si può
 178 notare dalla linea successiva relativa al codice «215»:

179 00200714212215_____1,

180 ove _ indica uno spazio, per la quale, tralasciando il fatto che vi sono in tota-
 181 le 6 città italiane colo stesso codice comunale, se ci si limita alla regione del
 182 Piemonte (vale a dire la stessa del comune di partenza «002007») esistono in
 183 realtà ben due città condividenti lo stesso codice: «001215» e «004215», ep-
 184 pure entrambe non hanno lo stesso codice provinciale («002») del comune di
 185 partenza. Da questo piccolo estratto, quindi, non si può nemmeno ipotizza-
 186 re che il codice provinciale sia lo stesso tra il comune di partenza e d'arrivo;
 187 è allora chiaro che sia impossibile scegliere la prima metà per completare il
 188 codice del comune di destinazione.

189 **Osservazione 2.2.** A dire il vero si potrebbe scegliere, tra tutt'i comuni colo
 190 stesso codice comunale, quello che minimizza la distanza geografica (ossia
 191 euleriana) siccome le coordinate dei punti rappresentativi di ogni comune
 192 sono disponibili nell'ISTAT [13]. In ogni caso il numero così esiguo d'eccezio-
 193 ni rendono una tale "correzione" del tutto innecessaria, oltre che irrilevante
 194 come il Cap. 4 mostrerà.

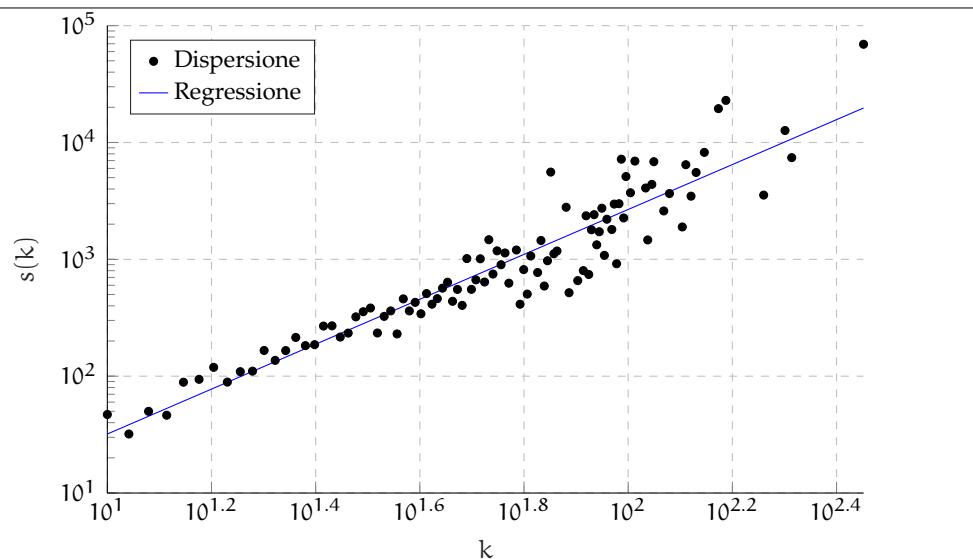


Figura 2.2: Forza contro grado per la Sardegna.

195 Infine nelle Figg. 2.2 e 2.3 sono stati anche riprodotti i risultati di [6] colle
196 seguenti osservazioni:

- 197 1. il coefficiente d'aggregazione medio $\langle C(k) \rangle = 0.453$ è quasi il doppio
198 di quello riportato da [6, p. 911] (0.26);
- 199 2. la Fig. 2.3c rispetto alla [6, Fig. 3b] presenta sia una forma legger-
200 mente diversa, seppure la tendenza a mezza luna sia la stessa, che
201 limiti superiori e inferiori più estesi;
- 202 3. il coefficiente d'aggregazione pesato rappresentato nella Fig. 2.3g
203 ha chiaramente una tendenza decrescente, anche se molto lenta, ra-
204 gion per cui si può considerare comunque «circa costante» come
205 afferma [6];
- 206 4. i primi due pesi più elevati, riportati nella Tab. 2.2, non corrispondo-
207 no: ciò si può giustificare molto probabilmente come refuso da parte
208 di [6] poiché i collegamenti «Cagliari-Sassari» e «Sassari-Olbia» so-
209 no molto distanti geograficamente², per cui è ragionevole che i flussi
210 siano bassi.

Connessione	Peso
Cagliari-Quartu SE	14709
Cagliari-Selargius	7995
Assemini-Selargius	4418
Porto Torres-Sassari	4149
Cagliari-Capoterra	3865

(a) Pesi maggiori correnti.

Connessione	Peso
Cagliari-Sassari	13953
Sassari-Olbia	7246
Cagliari-Assemini	4226
Porto Torres-Sassari	3993
Cagliari-Capoterra	3731

(b) Pesi maggiori di [6]

Tabella 2.2: Confronto con [6] dei pesi maggiori.

211 Queste discrepanze sono perché sono lievi e, per quanto concerne questo
212 elaborato, irrilevanti. Inoltre, l'unico risultato di rilievo è la correlazione

² Il primo equivale a percorrere l'intera lunghezza dell'isola ogni giorno.

²¹³ positiva tra la forza e il grado nella Fig. 2.2.

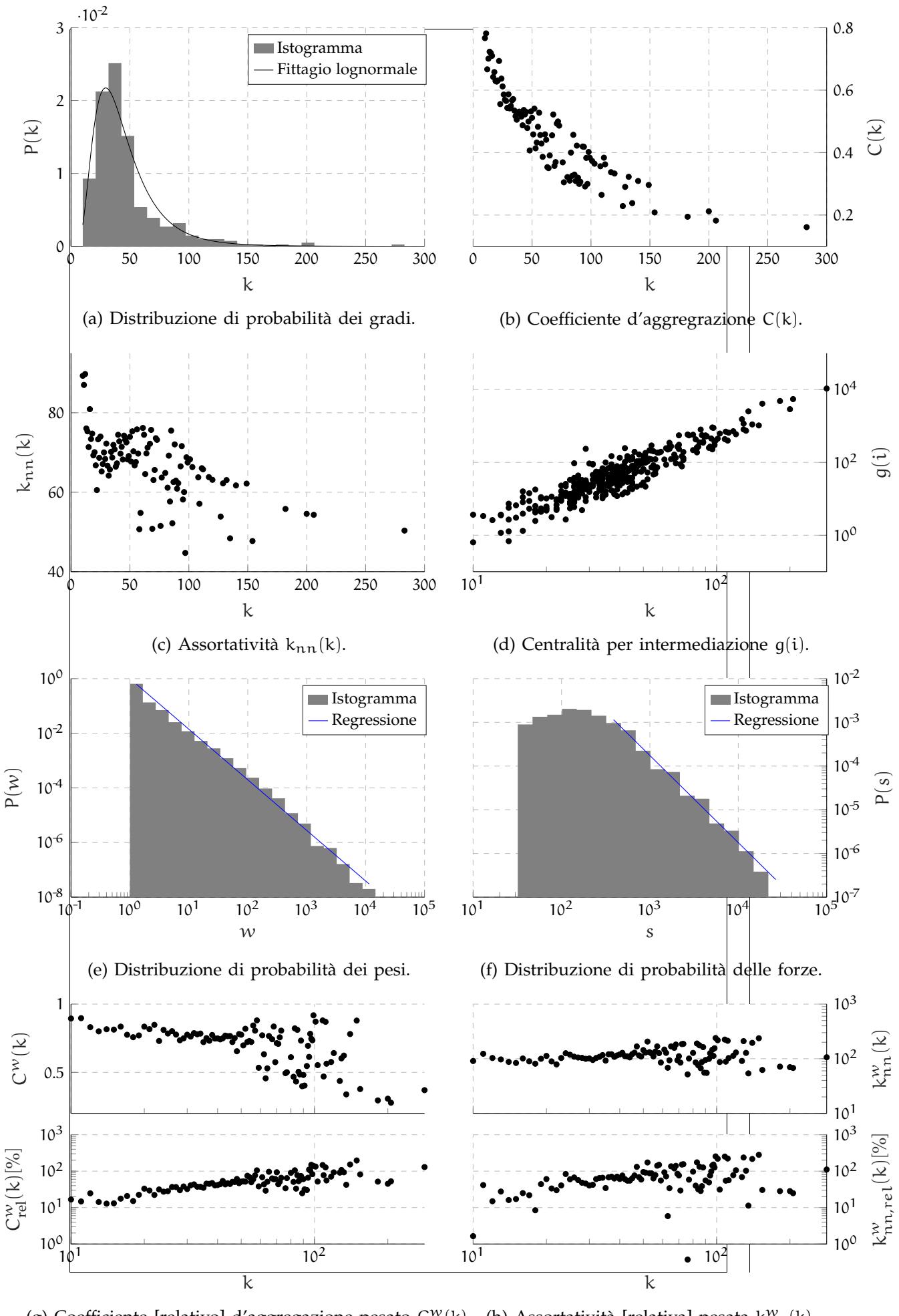


Figura 2.3: Riproduzione dei principali grafici di [6] [a cui si rimanda per maggiori dettagli sui vari coefficienti] relativi alla topologia della rete del pendolarismo sarda.

3

TEORIA CINETICA DEI SISTEMI MULTIAGENTE

In questo capitolo è discussa la [TCSMA](#) necessaria per ottenere e discutere i risultati nel Cap. 4. S'inizia prima dando delle nozioni generali di teoria della probabilità dalle quali si deriva successivamente la celeberrima equazione di Boltzmann a partire da una descrizione stocastica delle particelle di un gas; quindi si passa a generalizzare tali risultati per mediare le interazioni tramite una struttura topologica sottostante gli agenti, sia in modo esatto che approssimato; successivamente si descrivono le equazioni cinetiche nel caso corrente; infine si conclude analizzando le ipotesi semplificative che portano dalla presente teoria a quella classica di Boltzmann.

3.1 DEFINIZIONI PRELIMINARI DI PROBABILITÀ

Si annoverano ora alcuni concetti e risultati di teoria della probabilità avvisando, però, che per molti di essi è impossibile entrare eccessivamente nei dettagli senza spiegare un intero corso; in ogni caso si rimanda a [7] per gl'interessati.

Definizione 3.1 (Semiretta intera/reale positiva). In tutto questo elaborato si fanno uso dei seguenti sottinsiemi dei numeri reali e naturali:

$$\mathbb{R}_+ \equiv \{x \in \mathbb{R} : x \geq 0\} \subset \mathbb{R}, \\ \mathbb{N}_+ \equiv \{n \in \mathbb{N} : n \geq 1\} \subset \mathbb{N}.$$

Definizione 3.2 (Variabile aleatoria). Una variabile aleatoria X è una funzione misurabile $X: \Omega \rightarrow \mathbb{R}$, il cui dominio è uno spazio astratto Ω di eventi; quest'ultimo definisce a sua volta uno spazio di probabilità $(\Omega, \mathcal{F}, \mathbb{P})$ composto da una σ -algebra \mathcal{F} e una misura \mathbb{P} di massa unitaria.

Siccome Ω è uno spazio astratto poco trattabile, in questo scritto si definisce una variabile aleatoria semplicemente evidenziando l'appartenenza al suo codominio e sottintendendo il suo dominio: $X \in \mathbb{R}$.

Definizione 3.3 (Densità di probabilità di X). Sia $X \in \mathbb{R}$ una variabile aleatoria assolutamente continua, allora la densità di probabilità $f_X: \mathbb{R} \rightarrow \mathbb{R}_+$ è una funzione misurabile che rappresenta la legge \mathbb{P}_X di X :

$$\mathbb{P}_X(A) = \mathbb{P}(X \in A) = \int_A f_X(x) dx \quad \forall A \subseteq \mathbb{R} \text{ misurabile.}$$

Osservazione 3.1. Nella definizione precedente si assume che la variabile aleatoria X sia assolutamente continua, in tal modo $f_X(x)$ è la sua derivata di Radon-Nikodym, ma la teoria sviluppata in questo paragrafo vale anche per misure più astratte rispetto le quali $f_X(x)dx$ va inteso come $f_X(dv)$.

Definizione 3.4 (Densità congiunta di probabilità di \mathbf{X}). Siano $n \in \mathbb{N}^+$ e

$$\mathbf{X} = (X_1, X_2, \dots, X_n) \in \mathbb{R}^n$$

un vettore aleatorio assolutamente continuo, allora la densità congiunta di probabilità $f_{\mathbf{X}}: \mathbb{R}^n \rightarrow \mathbb{R}_+$ è una funzione misurabile che rappresenta la legge $\mathbb{P}_{\mathbf{X}}$ di \mathbf{X} :

$$\begin{aligned}\mathbb{P}_{\mathbf{X}}(A) &= \mathbb{P}(\mathbf{X} \in A) = \int_A f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \\ &= \int_A f_{\mathbf{X}}(\mathbf{x}) dx_1 dx_2 \cdots dx_n \quad \forall A \subseteq \mathbb{R}^n \text{ misurabile.}\end{aligned}$$

Definizione 3.5 (Densità marginale di probabilità di X_i). Siano $n \in \mathbb{N}^+$ e $\mathbf{X} \in \mathbb{R}^n$ un vettore aleatorio assolutamente continuo, allora la densità marginale di probabilità $f_{X_i}: \mathbb{R} \rightarrow \mathbb{R}_+$ è una funzione misurabile così definita:

$$f_{X_i}(x_i) = \int_{\mathbb{R}^{n-1}} f_{\mathbf{X}}(\mathbf{x}) dx_1 dx_2 \cdots dx_{i-1} dx_{i+1} \cdots dx_n \quad \forall i \in \{1, 2, \dots, n\}; \quad (3.1)$$

essa è a tutti gli effetti la densità di probabilità della variabile aleatoria X_i secondo la Def. 3.3.

Osservazione 3.2. Per brevità, da qui in avanti s'indicheranno le precedenti densità sottintendendo che siano relative alla probabilità: $f_{\mathbf{X}}$ è dunque la densità di \mathbf{X} , f_X la densità congiunta di \mathbf{X} e f_{X_i} la densità marginale di X_i .

Inoltre, qualora il contesto non dia ambiguità, si sottintendono le variabili o i vettori aleatori cui fanno riferimento le relative densità: dunque f è la densità di \mathbf{X} , X e X_i , a seconda del caso; in alternativa si possono usare anche forme più leggere, come f_i per f_{X_i} , se necessario.

Proprietà 3.1 (Densità congiunta di variabili aleatorie indipendenti). Siano $n \in \mathbb{N}^+$ e $\mathbf{X} \in \mathbb{R}^n$ un vettore aleatorio assolutamente continuo, allora se tutte le variabili aleatorie che lo compongono sono tra loro indipendenti allora la densità congiunta di \mathbf{X} equivale alla produttoria di quelle marginali:

$$X_i \perp\!\!\!\perp X_j \quad \forall i \neq j \in \{1, 2, \dots, n\} \implies f_{\mathbf{X}} = \prod_{i=1}^n f_{X_i}.$$

Definizione 3.6 (Valore atteso). Sia $X \in \mathbb{R}$ una variabile aleatoria assolutamente continua e data una funzione misurabile $\varphi: \mathbb{R} \rightarrow \mathbb{R}$ arbitraria, allora il valore atteso di $\varphi(X)$ è definito come

$$\mathbb{E}[\varphi(X)] \equiv \int_{\mathbb{R}} \varphi(x) f_X(x) dx,$$

e un simile discorso vale anche per un vettore aleatorio ma con $\varphi: \mathbb{R}^n \rightarrow \mathbb{R}$, preso $n \in \mathbb{N}^+$:

$$\mathbb{E}[\varphi(\mathbf{X})] \equiv \int_{\mathbb{R}^n} \varphi(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = \int_{\mathbb{R}^n} \varphi(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) dx_1 dx_2 \cdots dx_n.$$

Definizione 3.7 (Valore atteso condizionato). Sia $X \in \mathbb{R}$ una variabile aleatoria assolutamente continua a media finita $\mathbb{E}[X] \in \mathbb{R}$ e sia $\mathcal{G} \subseteq \mathcal{F}$ una sotto- σ -algebra, allora l'attesa di X condizionata a \mathcal{G} è una variabile aleatoria Y tale che

274 AC₁ $\sigma(Y) \in \mathcal{G}$, ossia Y è \mathcal{G} -misurabile e

275 AC₂ per ogni $A \in \mathcal{G}$ la misura di X e Y tramite \mathbb{P} è la stessa:

$$\mathbb{P}(X \in A) = \int_A X d\mathbb{P} = \int_A Y d\mathbb{P} = \mathbb{P}(Y \in A)$$

276 L'insieme delle Y che soddisfanno le AC₁ e AC₂ sono indicate col simbolo
 277 $\mathbb{E}[X|\mathcal{G}]$ che ne è anche il suo rappresentante; in effetti una generica Y si può
 278 indicare anche direttamente come $\mathbb{E}[X|\mathcal{G}]$.

279 Un'analogia definizione vale anche per i vettori aleatori e per i caso misti
 280 con X scalare e Y vettoriale.

281 **Osservazione 3.3.** Presa una variabile aleatoria $Y \in \mathbb{R}$, nella precedente defi-
 282 nizione \mathcal{G} può essere anche la σ -algebra definita da Y :

$$\sigma(Y) \equiv \sigma(Y^{-1}(B) \mid B \in \mathcal{B}(\mathbb{R})),$$

283 ove $\mathcal{B}(\mathbb{R})$ è la σ -algebra di Borel relativa allo spazio dei numeri reali mentre
 284 $\sigma(A)$ è la più piccola σ -algebra che contiene $A \subset \mathbb{R}_+(\Omega)$.

285 Un simile risultato vale anche per i vettori aleatori $\mathbf{Y} \in \mathbb{R}^n$.

286 **Proposizione 3.1.** *Dati*

287 1. $n, h, k \in \mathbb{N}_+$ tali che $n = h + k$;

288 2. un vettore aleatorio $\mathbf{Z} = (\mathbf{X}, \mathbf{Y}) \in \mathbb{R}^n$ assolutamente continuo costituito da
 289 $\mathbf{X} \in \mathbb{R}^h$ e $\mathbf{Y} \in \mathbb{R}^k$ con densità congiunta $f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y})$ e marginali $f_{\mathbf{X}}$ e $f_{\mathbf{Y}}$;

290 3. una funzione $\varphi: \mathbb{R}^n \rightarrow \mathbb{R}$ misurabile e tale che

$$\mathbb{E}[\varphi(\mathbf{Z})] = \mathbb{E}[\varphi(\mathbf{X}, \mathbf{Y})] < \infty;$$

291 allora si può definire $h: \mathbb{R}^k \rightarrow \mathbb{R}$ tramite la seguente equazione

$$\int_{\mathbb{R}^h} h(\mathbf{y}) f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} = \int_{\mathbb{R}^h} \varphi(\mathbf{x}, \mathbf{y}) f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} \quad \forall \mathbf{y} \in \mathbb{R}^k, \quad (3.2)$$

292 che soddisfa $h(\mathbf{Y}) \in \mathbb{E}[\varphi(\mathbf{X}, \mathbf{Y})|\mathbf{Y}]$.

293 *Dimostrazione.* Per la AC₁ è sufficiente esplicitare la forma di $\sigma(h(\mathbf{Y}))$:

$$\sigma(h(\mathbf{Y})) \equiv \sigma(Y^{-1}(h^{-1}(B)) \mid B \in \mathcal{B}(\mathbb{R})) \subseteq \sigma(Y^{-1}(B) \mid B \in \mathcal{B}(\mathbb{R}^h)),$$

294 a parole si considera la controimmagine tramite \mathbf{Y} dei borelliani filtrati da
 295 h , ragion per cui la σ -algebra non potrà che essere contenuta in quella non
 296 filtrata.

297 Per la AC₂ si verifica la condizione $\forall A \in \sigma(\mathbf{Y})$ avvalendosi della caratteriz-
 298 zazione di $h(\mathbf{Y})$ tramite (3.2):

$$\begin{aligned} \int_A h(\mathbf{Y}) d\mathbb{P} &= \int_B h(\mathbf{y}) f_{\mathbf{Y}}(\mathbf{y}) d\mathbf{y} \stackrel{(3.1)}{=} \int_B h(\mathbf{y}) \int_{\mathbb{R}^h} f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \int_B \int_{\mathbb{R}^h} h(\mathbf{y}) f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \stackrel{(3.2)}{=} \int_B \int_{\mathbb{R}^h} \varphi(\mathbf{x}, \mathbf{y}) f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \int_{\mathbb{R}^n} \chi_B(\mathbf{y}) \varphi(\mathbf{x}, \mathbf{y}) f_{\mathbf{Z}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} = \int_{\Omega} \chi_B(\mathbf{Y}) \varphi(\mathbf{X}, \mathbf{Y}) d\mathbb{P} \\ &= \int_{\Omega} \chi_C(\mathbf{Z}) \varphi(\mathbf{X}, \mathbf{Y}) d\mathbb{P} = \int_{\Omega} \chi_A \varphi(\mathbf{X}, \mathbf{Y}) d\mathbb{P} = \int_A \varphi(\mathbf{X}, \mathbf{Y}) d\mathbb{P}, \end{aligned}$$

ove χ_A , χ_B e χ_C sono funzioni indicatrici dei rispettivi insiemi, e in piú vale
 $B \equiv Y(A) \subseteq \mathbb{R}^k$ e $C \equiv Z(\mathbb{R}^h \times A) \subseteq \mathbb{R}^n$ dimodoché

$$A = Y^{-1}(B) = Z^{-1}(C).$$

□

Osservazione 3.4. La precedente dimostrazione svela un'interpretazione piú pratica dell'attesa condizionata rispetto alla sua definizione alquanto teorica.

Si consideri la (3.2), allora $h(y)$ si può cosí riformulare:

$$h(y) = \int_{\mathbb{R}^h} \varphi(x, y) f_{X|Y=y}(x) dx = \mathbb{E}[\varphi(X, Y)|Y=y] \quad \forall y \in \mathbb{R}^k, \quad (3.3)$$

in cui

$$f_{X|Y=y} \equiv \begin{cases} f_Z(x, y)/f_Y(y) & \text{se } f_Y(y) \neq 0 \\ 0 & \text{se } f_Y(y) = 0 \end{cases}$$

è la densità di X condizionata dall'evento $Y=y$.

La (3.3), in pratica, si calcola svolgendo la media di $\varphi(X, Y)$ rispetto a X dopo aver "fissato" l'evento $Y=y$, vale a dire considerando la Y come fosse un parametro uguale alla realizzazione y .

Successivamente, se s'impone $A=\Omega$ (sempre lecito essendo \mathcal{G} una σ -algebra) allora dalla AC2 vale

$$\mathbb{E}[\varphi(X, Y)] = \mathbb{E}[\mathbb{E}[\varphi(X, Y)|Y]], \quad (3.4)$$

il che significa che l'attesa di $\varphi(X, Y)$ si calcola mediando rispetto a Y la variabile aleatoria $\mathbb{E}[\varphi(X, Y)|Y]$, ricavata dalla (3.3) ma considerando y arbitrario.

Definizione 3.8 (Processo stocastico). Data una dimensione $n \in \mathbb{N}_+$, un processo stocastico è un insieme di vettori aleatori X_t parametrizzati da un indice $t \in I \subseteq \mathbb{R}$:

$$\{X_t \in \mathbb{R}^n \mid t \in I\} = \{X_t\}_{t \in I},$$

la cui densità congiunta è

$$\mathbb{P}_{X_t}(A) = \mathbb{P}(X_t \in A) = \int_A f_{X_t}(x, t) dx \quad \forall A \subseteq \mathbb{R}^n \text{ misurabile.}$$

Comunemente s'interpreta l'indice t come il tempo ponendo $I \equiv \mathbb{R}_+$, e cosí sarà fatto in questo scritto.

3.2 DESCRIZIONE CINETICA DI [TIPO] BOLTZMANN

3.2.1 Equazione di Boltzmann omogenea

Descrizione classica

Innanzitutto è necessario caratterizzare alcune proprietà del gas da modelizzare.

Ipotesi 3.1. Si consideri un gas composto da particelle che soddisfaccia le successive importanti ipotesi:

- 327 G₁ è uniformemente distribuito nello spazio cosicché in ogni punto la
 328 distribuzione statistica delle velocità è la stessa;
 329 G₂ è rarefatto, da cui segue che solo interazioni binarie possono aver
 330 luogo o, precisamente, sono più frequenti;
 331 G₃ è composto da particelle indistinguibili e aventi ugual massa;
 332 G₄ le collisioni sono elastiche per cui si ha conservazione dell'impulso
 333 e dell'energia esprimibile, grazie alle G₂ e G₃, binariamente come

$$\mathbf{v}' + \mathbf{v}'_* = \mathbf{v} + \mathbf{v}_*, \quad (\text{CI})$$

$$|\mathbf{v}'|^2 + |\mathbf{v}'_*|^2 = |\mathbf{v}|^2 + |\mathbf{v}_*|^2, \quad (\text{CE})$$

334 ove $\mathbf{v}', \mathbf{v}'_* \in \mathbb{R}^3$ sono le velocità poscollisionali che collidono colle
 335 velocità precollisionali $\mathbf{v}, \mathbf{v}_* \in \mathbb{R}^3$, delle due particelle interagenti.¹

336 Dalla G₄ si possono in realtà esprimere le velocità poscollisionali a partire
 337 da quelle precollisionali, definendo quelle che sono le regole d'interazione.

338 **Proposizione 3.2** (Regole d'interazione). *Esistono due funzioni*

$$\psi, \psi_* : \mathbb{R}^3 \rightarrow \mathbb{R}^3$$

339 *lineari che soddisfanno (CI, CE) e tali che*

$$\begin{aligned} \mathbf{v}' &= \psi(\mathbf{v}, \mathbf{v}_*) \equiv \mathbf{v} + [(\mathbf{v}_* - \mathbf{v}) \cdot \mathbf{n}] \mathbf{n}, \\ \mathbf{v}'_* &= \psi_*(\mathbf{v}, \mathbf{v}_*) \equiv \mathbf{v}_* + [(\mathbf{v} - \mathbf{v}_*) \cdot \mathbf{n}] \mathbf{n}, \end{aligned} \quad (3.5)$$

340 *ove $\mathbf{n} \in \mathbb{S}^2$ è un qualunque vettore appartenente alla sfera unitaria di \mathbb{R}^3 .*

341 *Dimostrazione.* Assumendo l'ansatz

$$\mathbf{v}' = \mathbf{v} - \gamma \mathbf{n} \quad \text{e} \quad \mathbf{v}'_* = \mathbf{v}_* + \gamma \mathbf{n}, \quad (3.6)$$

342 in cui $\gamma \in \mathbb{R}$ è un parametro da determinare, si può notare che la (CI) è già
 343 verificata, mentre imponendo la (CE) si ha

$$\begin{aligned} |\mathbf{v}'|^2 + |\mathbf{v}'_*|^2 &= |\mathbf{v} - \gamma \mathbf{n}|^2 + |\mathbf{v}_* + \gamma \mathbf{n}|^2 \\ &= (\mathbf{v} - \gamma \mathbf{n}) \cdot (\mathbf{v} - \gamma \mathbf{n}) + (\mathbf{v}_* + \gamma \mathbf{n}) \cdot (\mathbf{v}_* + \gamma \mathbf{n}) \\ &= |\mathbf{v}|^2 - 2\gamma(\mathbf{v} \cdot \mathbf{n}) + \gamma^2 + |\mathbf{v}_*|^2 + 2\gamma(\mathbf{v}_* \cdot \mathbf{n}) + \gamma^2 \\ &= |\mathbf{v}|^2 + |\mathbf{v}_*|^2 - 2\gamma[(\mathbf{v}_* - \mathbf{v}) \cdot \mathbf{n} + 2\gamma] \\ &= |\mathbf{v}|^2 + |\mathbf{v}_*|^2 \implies \gamma[(\mathbf{v}_* - \mathbf{v}) \cdot \mathbf{n} + 2\gamma] = 0, \end{aligned}$$

344 ma escludendo il caso banale di $\gamma = 0$ (nessuna interazione) si ha

$$\gamma = \Gamma(\mathbf{v}, \mathbf{v}_*) \equiv (\mathbf{v} - \mathbf{v}_*) \cdot \mathbf{n},$$

345 per cui γ è in realtà una funzione delle velocità precollisionali; la (3.6) diventa allora

$$\mathbf{v}' = \mathbf{v} + [(\mathbf{v}_* - \mathbf{v}) \cdot \mathbf{n}] \mathbf{n} \quad \text{e} \quad \mathbf{v}'_* = \mathbf{v}_* + [(\mathbf{v} - \mathbf{v}_*) \cdot \mathbf{n}] \mathbf{n}. \quad (3.7)$$

¹ In questo contesto non è necessario distinguere quale delle due sia quella interagente e quale quella ricevente, contrariamente a quello delle città.

348 Si noti come la precedente dimostrazione lasci arbitrario $n \in S^2$, seppure
 349 da un punto di vista fisico sia ragionevole porlo parallelo al vettore passante
 350 per i centri delle particelle collidenti.

351 Si osservi come le regole d'interazione nella 3.5 sono bilineari e simmetri-
 352 che secondo la seguente definizione:

353 **Definizione 3.9.** Se le regole d'interazione del tipo (3.5) verificano

$$\begin{aligned}\psi(\mathbf{v}, \mathbf{v}_*) &= \psi_*(\mathbf{v}_*, \mathbf{v}) & \mathbf{v}' &= \psi_*(\mathbf{v}_*, \mathbf{v}) \\ \psi_*(\mathbf{v}, \mathbf{v}_*) &= \psi(\mathbf{v}_*, \mathbf{v}) & \mathbf{v}'_* &= \psi(\mathbf{v}_*, \mathbf{v})\end{aligned}$$

354 allora si dicono simmetriche.

355 *Descrizione statistica*

356 Per proseguire è però necessaria una descrizione statistica del gas in esame:
 357 siano allora $\{\mathbf{V}_t\}_{t \in \mathbb{R}_+}$ e $\{\mathbf{V}_t^*\}_{t \in \mathbb{R}_+}$ i processi stocastici delle velocità precollisio-
 358 nali, dove $\mathbf{V}_t, \mathbf{V}_t^* \in \mathbb{R}^3$ sono variabili aleatorie di cui le velocità precedenti, \mathbf{v}
 359 e \mathbf{v}_* , sono due realizzazioni al tempo t ; un simile discorso vale per i processi
 360 stocastici delle velocità poscollisionali $\{\mathbf{V}'_t\}_{t \in \mathbb{R}_+}$ e $\{\mathbf{V}'_t^*\}_{t \in \mathbb{R}_+}$.

361 **Osservazione 3.5.** Le variabili aleatorie \mathbf{V}_t e \mathbf{V}_t^* non sono indicizzate (per es.
 362 \mathbf{V}_t^i) proprio per l'indistinguibilità delle particelle assunta dalla G3, da cui si
 363 deduce anche che le leggi di \mathbf{V}_t e \mathbf{V}_t^* sono identiche:

$$f_{\mathbf{V}_t}(\mathbf{v}, t) = f_{\mathbf{V}_t^*}(\mathbf{v}, t) = f(\mathbf{v}, t) \quad \forall \mathbf{v} \in \mathbb{R}^3.$$

364 Le regole d'interazione nella (3.5) diventano

$$[RI]_V \begin{cases} \mathbf{V}' = \psi(\mathbf{V}, \mathbf{V}_*) \equiv \mathbf{V} + [(\mathbf{V}_* - \mathbf{V}) \cdot \mathbf{n}] \mathbf{n}, \\ \mathbf{V}'_* = \psi_*(\mathbf{V}, \mathbf{V}_*) \equiv \mathbf{V}_* + [(\mathbf{V} - \mathbf{V}_*) \cdot \mathbf{n}] \mathbf{n}, \end{cases}$$

365 in cui anche \mathbf{n} risulta aleatoriamente distribuito; usualmente s'ipotizza $\mathbf{n} \sim$
 366 $\mathcal{U}(S^2)$, vale a dire che non vi sono direzioni preferenziali essendo queste
 367 uniformemente distribuite sulla sfera unitaria.

368 Tuttavia l'interazione tra due particelle non è detto che avvenga, aspetto
 369 formulabile introducendo

$$\Theta \sim \text{Bernoulli}(B((\mathbf{V}_t^* - \mathbf{V}_t) \cdot \mathbf{n}) \Delta t) \quad [\text{Ber}]_V$$

370 che è una variabile aleatoria di Bernoulli, indipendente da \mathbf{V}_t e \mathbf{V}_t^* , la cui
 371 probabilità è descritta da due termini:

- 372 1. la funzione $B: \mathbb{R} \rightarrow \mathbb{R}_+$, detta nucleo di collisione, che permette d'a-
 373 vere una maggiore espressività sulle collisioni molecolari le quali
 374 possono influenzare il tasso d'interazione tra le molecole²;
- 375 2. un passo temporale $\Delta t > 0$ per discretizzare il tempo.

376 **Osservazione 3.6.** Per definizione di Θ , deve valere $B((\mathbf{V}_t^* - \mathbf{V}_t) \cdot \mathbf{n}) \Delta t \leq 1$
 377 condizione soddisfacibile anche prendendo un passo temporale Δt adattivo;
 378 si vedrà tra poco, però, che, per quanto concerne il modello analitico, tale
 379 condizione sarà sempre verificata.

2 Si noti come B dipenda da $(\mathbf{V}_* - \mathbf{V}) \cdot \mathbf{n}$, termine ripreso dalla Prop. 3.2, cosa che permette d'interpretare \mathbf{n} come la direzione lungo la quale le interazioni sono più frequenti.

380 Colle [RI] \mathbf{v} e [Ber] \mathbf{v} , lo stato dei due agenti al tempo $t + \Delta t$ successivo è
381 dunque dato dal seguente algoritmo d'interazione:

$$[AR]_{\mathbf{v}} \begin{cases} \mathbf{V}_{t+\Delta t} = (1-\Theta)\mathbf{V}_t + \Theta\mathbf{V}'_t, \\ \mathbf{V}'_{t+\Delta t} = (1-\Theta)\mathbf{V}'_t + \Theta\mathbf{V}''_t, \end{cases}$$

382 che di fatto è una regola che descrive se gli agenti interagiscono a coppie
383 (modellizzato dalla Θ) e, nel caso, come interagiscono (modellizzato dalle
384 \mathbf{V}'_t e \mathbf{V}''_t) modificando il loro stato al tempo successivo.

385 *Derivazione modello*

386 L'idea successiva, al fine di ricavare un modello dell'evoluzione di f , è di me-
387 diare le [AR] \mathbf{v} attraverso una funzione arbitraria $\varphi: \mathbb{R}^3 \rightarrow \mathbb{R}$, detta quantità
388 osservabile, computabile dalle realizzazioni o di $\mathbf{V}_{t+\Delta t}$ o di $\mathbf{V}'_{t+\Delta t}$; pertanto
389 una volta applicato $\varphi(\cdot)$ alle [AR] \mathbf{v} ,

$$\begin{aligned} \varphi(\mathbf{V}_{t+\Delta t}) &= \varphi((1-\Theta)\mathbf{V}_t + \Theta\mathbf{V}'_t), \\ \varphi(\mathbf{V}'_{t+\Delta t}) &= \varphi((1-\Theta)\mathbf{V}'_t + \Theta\mathbf{V}''_t), \end{aligned}$$

390 e mediando $\mathbb{E}[\cdot]$ si arriva a

$$\begin{aligned} \mathbb{E}[\varphi(\mathbf{V}_{t+\Delta t})] &= \mathbb{E}[\varphi((1-\Theta)\mathbf{V}_t + \Theta\mathbf{V}'_t)], \\ \mathbb{E}[\varphi(\mathbf{V}'_{t+\Delta t})] &= \mathbb{E}[\varphi((1-\Theta)\mathbf{V}'_t + \Theta\mathbf{V}''_t)]; \end{aligned}$$

391 per espandere la media, siccome Θ dipende da $(\mathbf{V}_t, \mathbf{V}'_t, \mathbf{n})$, bisogna avvalersi
392 dell'attesa condizionata, e in particolare della Oss. 3.4, che porta a

$$\begin{aligned} \mathbb{E}[\varphi(\mathbf{V}_{t+\Delta t})] &= \mathbb{E}[\varphi((1-\Theta)\mathbf{V}_t + \Theta\mathbf{V}'_t)], \\ &= \mathbb{E}[\mathbb{E}[\varphi((1-\Theta)\mathbf{V}_t + \Theta\mathbf{V}'_t) | \mathbf{V}_t, \mathbf{V}'_t, \mathbf{n}]], \\ &= \mathbb{E}[\varphi(\mathbf{V}_t)(1 - B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})\Delta t)] \\ &\quad + \mathbb{E}[\varphi(\mathbf{V}'_t)B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})]\Delta t, \end{aligned} \tag{3.8}$$

393 e similmente per $\mathbb{E}[\varphi(\mathbf{V}'_{t+\Delta t})]$; riordinando i termini si ricava

$$\begin{aligned} \frac{\mathbb{E}[\varphi(\mathbf{V}_{t+\Delta t})] - \mathbb{E}[\varphi(\mathbf{V}_t)]}{\Delta t} &= \mathbb{E}[B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})(\varphi(\mathbf{V}'_t) - \varphi(\mathbf{V}_t))], \\ \frac{\mathbb{E}[\varphi(\mathbf{V}'_{t+\Delta t})] - \mathbb{E}[\varphi(\mathbf{V}'_t)]}{\Delta t} &= \mathbb{E}[B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})(\varphi(\mathbf{V}'_t) - \varphi(\mathbf{V}'_t))], \end{aligned}$$

394 e passando formalmente al tempo continuo col limite $\Delta t \rightarrow 0^+$ ³ si ha

$$\begin{aligned} \frac{d\mathbb{E}[\varphi(\mathbf{V}_t)]}{dt} &= \mathbb{E}[B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})(\varphi(\mathbf{V}'_t) - \varphi(\mathbf{V}_t))], \\ \frac{d\mathbb{E}[\varphi(\mathbf{V}'_t)]}{dt} &= \mathbb{E}[B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})(\varphi(\mathbf{V}'_t) - \varphi(\mathbf{V}'_t))], \end{aligned}$$

395 che, dopo aver espanso i valori attesi secondo le loro definizioni, diventano

$$\begin{aligned} \frac{d}{dt} \int_{\mathbb{R}^3} \varphi(\mathbf{v}) f(\mathbf{v}, t) d\mathbf{v} &= \int_{\mathbb{R}^6} \langle B(\mathbf{v}, \mathbf{v}', \mathbf{n})(\varphi(\mathbf{v}') - \varphi(\mathbf{v})) \rangle f_V(\mathbf{v}, \mathbf{v}', t) d\mathbf{v} d\mathbf{v}', \\ \frac{d}{dt} \int_{\mathbb{R}^3} \varphi(\mathbf{v}') f(\mathbf{v}', t) d\mathbf{v}' &= \int_{\mathbb{R}^6} \langle B(\mathbf{v}, \mathbf{v}', \mathbf{n})(\varphi(\mathbf{v}') - \varphi(\mathbf{v})) \rangle f_V(\mathbf{v}, \mathbf{v}', t) d\mathbf{v} d\mathbf{v}', \end{aligned} \tag{3.9}$$

³ Ciò permette di soddisfare la condizione descritta nel'3.6 per qualunque valore di $B((\mathbf{V}'_t - \mathbf{V}_t) \cdot \mathbf{n})$.

396 in cui

$$\langle \cdot \rangle \equiv \frac{1}{4\pi} \int_{S^2} \cdot d\Omega$$

397 indica la media rispetto a Ω , $B(\mathbf{v}, \mathbf{v}_*, \Omega)$ è una forma breve per $B((\mathbf{v} - \mathbf{v}_*) \cdot \Omega)$.
 398 e si è posto $\mathbf{V} \equiv (\mathbf{V}_t, \mathbf{V}_t^*)$ la cui densità congiunta è $f_V(\mathbf{v}, \mathbf{v}_*, t)$. È infine
 399 necessario sommare le due equazioni in (3.9). S'inizi dal primo membro:

$$\frac{d}{dt} \int_{\mathbb{R}^3} \varphi(\mathbf{v}) f(\mathbf{v}, t) d\mathbf{v} + \frac{d}{dt} \int_{\mathbb{R}^3} \varphi(\mathbf{v}^*) f(\mathbf{v}_*, t) d\mathbf{v}_* = 2 \frac{d}{dt} \int_{\mathbb{R}^3} \varphi(\mathbf{v}) f(\mathbf{v}, t) d\mathbf{v}, \quad (3.10)$$

400 infatti dal'Oss. 3.5 le due leggi sono di fatto identiche come lo è il domi-
 401 nio d'integrazione, quindi basta il cambio di variabile $\mathbf{v}_* = \mathbf{v}$ nel secondo
 402 integrale per rendersi conto dell'equivalenza. Per il secondo membro sono
 403 necessarie due ulteriori fondamentali ipotesi:

404 **Ipotesi 3.2.** Si assume che il nucleo di collisione sia una funzione pari:

$$B((\mathbf{v} - \mathbf{v}_*) \cdot \Omega) = B((\mathbf{v}_* - \mathbf{v}) \cdot \Omega) \quad \forall \mathbf{v}, \mathbf{v}_* \in \mathbb{R}^3; \quad (3.11)$$

405 una scelta tipica è il valore assoluto $|\cdot|$: $B((\mathbf{v} - \mathbf{v}_*) \cdot \Omega) \equiv |(\mathbf{v} - \mathbf{v}_*) \cdot \Omega|$.

406 **Ipotesi 3.3** (Caos molecolare). Le particelle interagenti secondo le [AR] V so-
 407 no campionante indipendentemente. Tal'ipotesi è più facile da giustificare
 408 matematicamente che fisicamente perché semplifica di molto conti; in ogni
 409 caso, la G2 la corrobora poiché in un gas rarefatto è naturale che se due par-
 410 ticelle interagiscono, prima che ricollidano, avranno perso ogni vicendevole
 411 dipendenza a causa delle innumerevoli altre interazioni colle altre particelle.

412 **Osservazione 3.7.** Dal'Ip. 3.3, unitamente alla proprietà 3.1, si deduce che la
 413 densità congiunta del vettore aleatorio $\mathbf{V} \equiv (\mathbf{V}_t, \mathbf{V}_t^*)$ è data dal prodotto

$$f_V(\mathbf{v}, \mathbf{v}_*, t) = f_{V_t}(\mathbf{v}, t) f_{V_t^*}(\mathbf{v}_*, t) = f(\mathbf{v}, t) f(\mathbf{v}_*, t) \quad \forall \mathbf{v}, \mathbf{v}_* \in \mathbb{R}^3.$$

414 In tal modo il termine moltiplicato a $\varphi(\mathbf{v}_*)$ nel secondo membro della
 415 seconda equazione della (3.9) può essere così riformulato:

$$\int_{\mathbb{R}^6} \langle (\varphi(\mathbf{v})) B(\mathbf{v}, \mathbf{v}_*, \Omega) \rangle f(\mathbf{v}, t) f(\mathbf{v}_*, t) d\mathbf{v} d\mathbf{v}_*, \quad (3.12)$$

416 seguendo la medesima logica della (3.10). Applicando i due risultati illu-
 417 strati nelle (3.10), (3.12) si perviene finalmente all'equazione di Boltzmann
 418 omogenea asimmetrica in forma debole:

$$\frac{d}{dt} \int_{\mathbb{R}^3} \varphi f d\mathbf{v} = \frac{1}{4\pi} \int_{\mathbb{R}^6} \int_{S^2} B(\mathbf{v}, \mathbf{v}_*, \Omega) \left(\frac{\varphi' + \varphi_*'}{2} - \varphi \right) f f_* d\Omega d\mathbf{v} d\mathbf{v}_*, \quad (3.13)$$

419 ove, per brevità di notazione, si sono sottintese le dipendenze per tutte le
 420 funzioni, espresse, salvo per il nucleo di collisione, da apici ' e asterischi *
 421 (per una spiegazione più dettagliata si legga poco dopo nel § 3.2.3).

422 Tuttavia, qualora le regole d'interazione siano simmetriche secondo la
 423 Def. 3.9, come in questo caso data la Prop. 3.2, sempre con un cambio di
 424 variabili e sfruttando la parità del nucleo di collisione vale

$$\int_{\mathbb{R}^6} (\langle B(\mathbf{v}, \mathbf{v}_*, \Omega) \varphi' \rangle) f f_* d\mathbf{v} d\mathbf{v}_* = \int_{\mathbb{R}^6} (\langle B(\mathbf{v}, \mathbf{v}_*, \Omega) \varphi'_* \rangle) f f_* d\mathbf{v} d\mathbf{v}_*,$$

425 da cui segue l'equazione di Boltzmann omogenea simmetrica in forma de-
426bole⁴:

$$\frac{d}{dt} \int_{\mathbb{R}^3} \varphi f d\mathbf{v} = \frac{1}{4\pi} \int_{\mathbb{R}^6} \int_{S^2} B(\mathbf{v}, \mathbf{v}_*, n) (\varphi' - \varphi) f f_* dnd\mathbf{v} d\mathbf{v}_*. \quad [\text{EtB}]_V$$

427 Si può anche ricavare la forma forte della [\[EtB\]_V](#) considerando le regole d'in-
428 terazione inverse della [\[RI\]_V](#) [\[16, § 2.5, p. 15\]](#), ma il conto esula dagli scopi di
429 questo elaborato; ciononostante si riporta come riferimento per il paragrafo
430 a venire:

$$\frac{\partial f}{\partial t} = \frac{1}{4\pi} \int_{\mathbb{R}^3} \int_{S^2} B(\mathbf{v}, \mathbf{v}_*, n) (f' f'_* - f f_*) dnd\mathbf{v}_*. \quad (3.14)$$

431 3.2.2 Equazione di Boltzmann disomogenea

432 L'equazione originariamente ricavata da Boltzmann non è la [\(3.14\)](#), bensì
433 è quella disomogenea la cui unica differenza è la presenza di un termine
434 avvettivo $\mathbf{v} \cdot \nabla f$ a primo membro:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = \frac{1}{4\pi} \int_{\mathbb{R}^3} \int_{S^2} B(\mathbf{v}, \mathbf{v}_*, n) (f' f'_* - f f_*) dnd\mathbf{v}_*, \quad (3.15)$$

435 dove l'operatore $\nabla \equiv (\partial_{x_1}, \partial_{x_2}, \partial_{x_3})$ è il gradiente spaziale e la distribuzione
436 dipende ora anche dallo spazio: $f \equiv f(\mathbf{x}, \mathbf{v}, t)$. Essa è un'equazione integro-
437 differenziale della distribuzione statistica delle posizioni e delle velocità a
438 tempo dato.

439 Il primo membro della [\(3.15\)](#) non è altro che la derivata materiale del-
440 la distribuzione f : posto $B(\mathbf{v}, \mathbf{v}_*, t) \equiv 0$, ovvero in mancanza di collisioni,
441 l'equazione involve in una del trasporto lineare con soluzione nota.

442 **Proposizione 3.3.** *La soluzione dell'equazione del trasporto lineare*

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = 0 \quad (3.16)$$

443 ha come soluzione $f(\mathbf{x}, \mathbf{v}, t) = f_0(\mathbf{x} - \mathbf{v}t, \mathbf{v})$ in cui $f_0 = f(\mathbf{x}, \mathbf{v}, 0)$ è la distribuzione
444 iniziale.

445 *Dimostrazione.* Si procede usando il metodo delle caratteristiche: nella [\(3.16\)](#)
446 la velocità \mathbf{v} nell'operatore avvettivo non dipende né dallo spazio né dal
447 tempo, quindi si possono prendere le curve $\mathbf{x}(t)$ tali che

$$\frac{d\mathbf{x}}{dt} = \mathbf{v} \implies \mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}t \quad (3.17)$$

448 dette, appunto, curve caratteristiche della [\(3.16\)](#); valutando allora la distri-
449 buzione $f(\mathbf{x}, \mathbf{v}, t)$ lungo di esse si può definire $\hat{f}(t) = f(\mathbf{x}(t), \mathbf{v}, t)$ che derivata
450 equivale a

$$\frac{d\hat{f}}{dt} = \frac{\partial f}{\partial t} + \nabla f \cdot \frac{d\mathbf{x}}{dt} = \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = 0 \implies \hat{f}(t) = \text{costante} \quad \forall t \geq 0$$

⁴ Si dice debole siccome essa vale per una funzione test φ arbitraria.

da cui si deduce che f è costante lungo le caratteristiche; ma dalla (3.17) si può scrivere, trascurando la dipendenza dal tempo, $\mathbf{x}_0 = \mathbf{x} - \mathbf{v}t$, dunque la soluzione in un generico punto $(\mathbf{x}, \mathbf{v}, t)$ sarà data da

$$f(\mathbf{x}, \mathbf{v}, t) = \hat{f}(t) = \hat{f}(0) = f_0(\mathbf{x}_0, \mathbf{v}) = f_0(\mathbf{x} - \mathbf{v}t, \mathbf{v}),$$

considerando nella prima uguaglianza la caratteristica che al tempo t passa per \mathbf{x} . \square

Ciò significa che in assenza di collisioni la distribuzione iniziale f_0 semplicemente trasla rigidamente nello spazio allo scorrere del tempo lungo la direzione della velocità costante \mathbf{v} , movimento che rappresenta la variazione medi statistica delle posizioni molecolari.

D'altra parte il secondo membro della (3.15) rappresenta la variazione media statistica dalle velocità molecolari a causa delle collisioni (viene perciò anche chiamato operatore collisionale).

Pertanto la (3.15) descrive una chiara separazione di effetti: il primo membro altera solo la posizione delle particelle per il trasporto libero, mentre il secondo comporta esclusivamente variazioni della velocità per le collisioni.

L'obbiettivo di Boltzmann tramite la (3.15) era di raccordare due mondi: il mondo macroscopico, che all'equilibrio termodinamico restituisce quantità fisiche stazionarie e ben definite, e quello microscopico la cui descrizione molecolare implica una continua e caotica variazione delle stesse quantità fisiche anche in condizioni di equilibrio termodinamico (agitazione termica).

Ciò è stato possibile grazie all'introduzione dei momenti statistici, ossia di quantità macroscopiche calcolabili a partire dalla distribuzione f soddisfacente la (3.15); i principali sono

$$\begin{aligned} \rho(\mathbf{x}, t) &\equiv \int_{\mathbb{R}^3} f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} && \text{densità,} \\ \mathbf{u}(\mathbf{x}, t) &\equiv \frac{1}{\rho(\mathbf{x}, t)} \int_{\mathbb{R}^3} \mathbf{v} f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} && \text{velocità massica,} \\ E(\mathbf{x}, t) &\equiv \frac{1}{\rho(\mathbf{x}, t)} \|\mathbf{v}\|^2 \int_{\mathbb{R}^3} f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} && \text{energia totale,} \\ e(\mathbf{x}, t) &\equiv \frac{1}{\rho(\mathbf{x}, t)} \int_{\mathbb{R}^3} \|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)\|^2 f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} && \text{energia interna,} \\ \theta(\mathbf{x}, t) &\equiv \frac{1}{3} e(\mathbf{x}, t) && \text{temperatura.} \end{aligned} \quad (3.18)$$

Tali quantità sono definite da f seppure questa sia in genere infattibile da ricavare dalla (3.15), vista la difficile trattabilità analitica dell'equazione; ep pure, tramite una sua attenta riformulazione in determinati regimi limite [16, 18], è possibile svicolarsi del tutto dall'evoluzione puntuale della f ricavando un sistema chiuso dei soli momenti (3.18). La convergenza di questi a un valore stazionario è il nesso tra i due mondi attraverso la descrizione mesoscopica indotta dalla distribuzione f .

3.2.3 Equazione di tipo Boltzmann omogenea

La derivazione dettagliata nel § 3.2.1 è valida in realtà per una classe di problemi molto più generali, detti di tipo Boltzmann omogenei, la cui teoria è stata sviluppata da Pareschi *et al.* [18]; sono chiamati così per due ragioni:

- 485 1. sono formulati per mezzo di equazioni integro-differenziali analoghe strutturalmente alla [EtB]_v e
 486 2. sono applicati a contesti⁵ molto distanti da quello originale delle velocità di particelle di un gas.

489 Si ripropongono in questo paragrafo molti risultati analoghi a quelli già visti nel § 3.2.1 senza però riscrivere tutti i passaggi.

491 **Descrizione e derivazione**

492 Sia $\{X_t\}_{t \in \mathbb{R}_+}$ il processo stocastico relativo allo stato microscopico dell'agente di un sistema, dove $X_t \in I \subseteq \mathbb{R}^n$ è il vettore aleatorio⁶ che descrive i suoi stati microscopici al tempo t ; allora le Ipp. 3.1 e 3.3 si possono così riformulare:

496 **Ipotesi 3.4.** Gli agenti sono caratterizzati dalle seguenti ipotesi:

- 497 B₁ la distribuzione statistica dei microstati è uniforme nello spazio, e quindi non dipende da esso;
 498 B₂ gli agenti interagiscono solo binariamente, ovvero le interazioni a coppia sono le più frequenti;
 499 B₃ gli agenti sono indistinguibili cosicché ognuno è rappresentativo del loro insieme;
 500 B₄ due agenti interagenti sono statisticamente indipendenti.

504 D'altra parte gli stati poscollisionali (postati), X'_t e $X^{*'}_t$, si legano con quelli precollisionali (prestati), X_t e X^*_t , tramite

$$[RI]_X \begin{cases} X'_t = \psi(X_t, X^*_t, y), \\ X^{*'}_t = \psi_*(X_t, X^*_t, y_*), \end{cases}$$

506 ove ora

$$\psi: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^h \rightarrow \mathbb{R}^n$$

$$\psi_*: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^{h_*} \rightarrow \mathbb{R}^n$$

507 sono generiche regole d'interazione, non necessariamente lineari o simmetriche (Def. 3.11), mentre $y \in \mathbb{R}^h$ e $y_* \in \mathbb{R}^{h_*}$, con $h, h_* \in \mathbb{N}$, rappresentano dei coefficienti potenzialmente stocastici.

510 L'interazione tra di essi avviene secondo una variabile aleatoria di Bernoulli del tipo

$$\Theta \sim \text{Bernoulli}(\mu(X_t, X^*_t, w, t) \Delta t), \quad [\text{Ber}]_X$$

512 con tasso/nucleo d'interazione

$$\mu(X_t, X^*_t, w, t): \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^k \times [0, +\infty) \rightarrow \mathbb{R}_+, \quad [\text{NI}]_X$$

513 per controllare quant'è "probabile" che i due agenti interagiscano, e coefficienti $w \in \mathbb{R}^k$, con $k \in \mathbb{N}$, potenzialmente stocastici.

5 Esempi importanti sono quello sociofisico, come i vari modelli sulle opinioni, ed econofisico, come quello qui considerato delle città

6 Si noti che il dominio I degli stati non coincide necessariamente col'intero spazio reale \mathbb{R}^n .

515 **Osservazione 3.8.** Se non si sa o non si vuole toccare quell'aspetto modelli-
 516 stico si può semplicemente porre unitario:

$$\mu \equiv 1, \quad \forall (\mathbf{x}, \mathbf{x}_*, \mathbf{w}, t) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^k \times [0, +\infty)$$

517 sottintendendo in tal modo un'uniforme "probabilità" d'interazione.

518 Nel complesso i due agenti s'interfacciano mediante l'algoritmo d'intera-
 519 zione

$$[\text{AR}]_{\mathbf{x}} \begin{cases} \mathbf{X}_{t+\Delta t} = (1-\Theta)\mathbf{X}_t + \Theta\mathbf{X}'_t, \\ \mathbf{X}'_{t+\Delta t} = (1-\Theta)\mathbf{X}'_t + \Theta\mathbf{X}_t, \end{cases}$$

520 dove $\Delta t \in \mathbb{R}_+$ è il passo temporale atto a discretizzare il tempo.

521 **Osservazione 3.9.** l'acronimo $[\text{AR}]_{\mathbf{x}}$ sta per «Azione-Reazione» poiché si
 522 suppone che ogn'interazione (azione), ossia $\Theta = 1$, necessariamente modi-
 523 fica entrambi gli stati degli agenti coinvolti (reazione); in alcuni contesti [17]
 524 questa regola può essere alterata dimodoché solo l'agente interagente sia
 525 modificato portando alle interazioni «Azione-Azione».

526 Infine, riapplicando un ragionamento analogo a quello svolto nel § 3.2.1
 527 colla quantità osservabile $\varphi: \mathbb{R}^n \rightarrow \mathbb{R}$, si perviene all'equazione di tipo Bol-
 528 tzmann omogenea generale

$$\frac{d}{dt} \int_{\mathbb{R}^n} \varphi f d\mathbf{x} = \int_{\mathbb{R}^{2n}} \left\langle \mu(\mathbf{x}, \mathbf{x}_*, \mathbf{w}, t) \frac{\varphi' + \varphi'_* - \varphi - \varphi_*}{2} \right\rangle f f_* d\mathbf{x} d\mathbf{x}_*, \quad [\text{EtB}]_{\mathbf{x}}$$

529 nella quale $\langle \cdot \rangle$ rappresenta la media rispetto a tutt'i coefficienti potenzial-
 530 mente stocastici (\mathbf{y} , \mathbf{y}_* e \mathbf{w}), mentre il termine $-\varphi - \varphi_*$ discende dal fatto
 531 che non si è ipotizzato che il tasso d'interazione μ sia pari.

532 **Osservazione 3.10** ($[\text{EtB}]_{\mathbf{x}}$ simmetrica). Qualora $[\text{NI}]_{\mathbf{x}}$ sia pari (Def. 3.10) e
 533 le regole d'interazione $[\text{RI}]_{\mathbf{x}}$ siano simmetriche (Def. 3.11), la $[\text{EtB}]_{\mathbf{x}}$ si può
 534 riformulare come

$$\frac{d}{dt} \int_{\mathbb{R}^n} \varphi f d\mathbf{x} = \int_{\mathbb{R}^{2n}} \langle \mu(\mathbf{x}, \mathbf{x}_*, \mathbf{w}, t) (\varphi' - \varphi) \rangle f f_* d\mathbf{x} d\mathbf{x}_*, \quad (3.19)$$

535 mediante un cambio di variabili atto a invertire \mathbf{x} e \mathbf{x}_* solo per la differenza
 536 $\varphi'_* - \varphi_*$.

537 **Definizione 3.10** ($[\text{NI}]_{\mathbf{x}}$ pari). Se il tasso d'interazione in $[\text{NI}]_{\mathbf{x}}$ verifica

$$\mu(\mathbf{x}, \mathbf{x}^*, \mathbf{w}, t) = \mu(\mathbf{x}^*, \mathbf{x}, \mathbf{w}, t) \quad \forall \mathbf{x}, \mathbf{x}^* \in I \subseteq \mathbb{R}^n \text{ e } \forall \mathbf{w} \in \mathbb{R}^k,$$

538 allora μ si dice pari.

539 **Definizione 3.11** ($[\text{RI}]_{\mathbf{x}}$ simmetriche). Se le regole d'interazione del tipo $[\text{RI}]_{\mathbf{v}}$
 540 verificano

$$\begin{aligned} \psi(\mathbf{x}, \mathbf{x}^*, \mathbf{y}) &= \psi_*(\mathbf{x}^*, \mathbf{x}, \mathbf{y}) & \forall \mathbf{x}, \mathbf{x}_* \in I \subseteq \mathbb{R}^n \text{ e } \forall \mathbf{y}, \mathbf{y}_* \in \mathbb{R}^h, \\ \psi_*(\mathbf{x}, \mathbf{x}^*, \mathbf{y}_*) &= \psi(\mathbf{x}^*, \mathbf{x}, \mathbf{y}_*) \end{aligned}$$

541 allora si dicono simmetriche. In altre parole, la simmetria implica

$$\mathbf{X}'_t = \psi_*(\mathbf{X}'_t, \mathbf{X}_t, \mathbf{y}) \quad \text{e} \quad \mathbf{X}'^*_t = \psi(\mathbf{X}'_t, \mathbf{X}_t, \mathbf{y}_*).$$

542 **Notazione**

543 Nei previ paragrafi si è fatto spesso uso, specie per le varie equazioni [di
 544 tipo] Boltzmann (3.13–3.15), [EtB]_v e [EtB]_x, di una notazione abbreviata per
 545 tutti gli oggetti salvo il tasso d’interazione μ . In questo breve paragrafo si de-
 546 finiscono in maniera più esplicita queste abbreviazioni facendo riferimento
 547 al contesto generale sviluppato poco fa.

548 Innanzitutto la densità congiunta f da sola sottintende la variabile dell’agente
 549 interagente x mentre con un pedice f_* quella dell’agente ricevente
 550 x_* :

$$f \equiv f(x, t) \quad \text{e} \quad f_* \equiv f(x_*, t);$$

551 dopodiché le quantità osservabili φ seguono la medesima logica di prima
 552 ma coll’aggiunta di un apice per distinguere le variabili postinterazionali:

$$\begin{aligned} \varphi &\equiv \varphi(x) & \text{e} & \varphi_* \equiv \varphi(x_*), \\ \varphi' &\equiv \varphi(x') & \text{e} & \varphi'_* \equiv \varphi(x'_*). \end{aligned}$$

553 Ovviamente questa notazione può essere generalizzata a una qualunque
 554 funzione scalare $g: \mathbb{R}^n \rightarrow \mathbb{R}$ funzione del microstato degli agenti:

$$\begin{aligned} g &\equiv g(x) & \text{e} & g_* \equiv g(x_*), \\ g' &\equiv g(x') & \text{e} & g'_* \equiv g(x'_*). \end{aligned}$$

555 Infine si osserva che qualora vi siano altri pedici o apici nelle funzioni a ve-
 556 nire, essi *non* fanno riferimento a questa notazione corta, se non altrimenti
 557 specificato. Per esempio giusto nel prossimo paragrafo si lavora con del-
 558 le densità f_i , in cui i è l’indice del nodo del grafo associato all’agente; è
 559 comunque possibile, anche per questi casi, usare la notazione precedente
 560 riposizionando lievemente i pedici e gli apici:

$$f^i \equiv f(x, t) \quad \text{e} \quad f_i^* \equiv f(x_*, t);$$

561 **Analisi dimensionale**

562 Finora tutte le equazioni passate fanno riferimento a un tempo adimensiona-
 563 le, sebbene l’introduzione di un tempo sia una modifica alquanto indolore e
 564 che lasci trasparire un’interessante interpretazione del passo temporale Δt .

565 Sia dunque $\tilde{t} = \tau t$ il tempo dimensionale scomposto nel prodotto di in uno
 566 adimensionale t e della dimensione τ .

567 **Osservazione 3.11.** Seguendo la notazione ISO [11, § 6.2] \tilde{t} dev’essere scrit-
 568 to $\tilde{t} = \{t\} \times [t]$ ma, per leggerezza di notazione, si omettono le parentesi po-
 569 nendo $\tau \equiv [t]$: l’importante è vedere t come il valore numerico variabile e
 570 adimensionale, mentre τ come la dimensione costante e fissa.

571 Detto ciò, mediante la derivata dimensionale

$$\frac{d \cdot}{dt} = \tau \frac{d \cdot}{d \tilde{t}}$$

572 e le riformulazioni

$$\tilde{f}(x, \tilde{t}) \equiv f(x, \tilde{t}/\tau), \quad \text{e} \quad \tilde{f}_*(x_*, \tilde{t}) \equiv f(x_*, \tilde{t}/\tau),$$

abbreviate con \tilde{f} e \tilde{f}_* rispettivamente, la [EtB]_X diventa

$$\frac{d}{d\tilde{t}} \int_{\mathbb{R}^n} \varphi \tilde{f} d\mathbf{x} = \frac{1}{\tau} \int_{\mathbb{R}^{2n}} \left\langle \mu(\mathbf{x}, \mathbf{x}_*, \mathbf{w}, \tilde{t}/\tau) \frac{\varphi' + \varphi'_* - \varphi - \varphi_*}{2} \right\rangle \tilde{f} \tilde{f}_* d\mathbf{x} d\mathbf{x}_*. \quad (3.20)$$

Per capire come varia il tasso d'interazione μ è sufficiente dimensionalizzare la [Ber]_X:

$$\Theta \sim \text{Bernoulli} \left(\mu(\mathbf{X}_t, \mathbf{X}_t^*, \mathbf{w}, \tilde{t}/\tau) \frac{\Delta\tilde{t}}{\tau}, \right),$$

da cui si deduce il tasso d'interazione dimensionale $\tilde{\mu}$:

$$\tilde{\mu}(\mathbf{x}, \mathbf{x}_*, \mathbf{w}, \tilde{t}) \equiv \frac{1}{\tau} \mu(\mathbf{x}, \mathbf{x}_*, \mathbf{w}, \tilde{t}/\tau),$$

che trasforma la (3.20) nell'equazione di tipo Boltzmann dimensionale

$$\frac{d}{d\tilde{t}} \int_{\mathbb{R}^n} \varphi \tilde{f} d\mathbf{x} = \int_{\mathbb{R}^{2n}} \left\langle \tilde{\mu}(\mathbf{x}, \mathbf{x}_*, \mathbf{w}, \tilde{t}) \frac{\varphi' + \varphi'_* - \varphi - \varphi_*}{2} \right\rangle \tilde{f} \tilde{f}_* d\mathbf{x} d\mathbf{x}_*. \quad (3.21)$$

Osservazione 3.12 (frequenza d'interazione). Dall'identità $\Delta\tilde{t} = \tau\Delta t$ si può interpretare Δt come la frazione della dimensione τ dopo la quale può avvenire un'interazione, da cui si deduce che $\Delta\tilde{t}$ è l'intervallo di tempo tra due interazioni.

Di conseguenza si può vedendo l'[AR]_X come un fenomeno periodico di periodo $\Delta\tilde{t}$ e frequenza

$$\tilde{f} \equiv \frac{1}{\Delta\tilde{t}} = \frac{1}{\tau\Delta t} \equiv \frac{1}{\tau} f,$$

in cui, come prima $1/\tau$ è la dimensione della frequenza adimensionale f .

Ecco che il reciproco del passo temporale Δt prende il significato della frequenza d'interazione del fenomeno, nell'unità di tempo sia adimensionale che dimensionale mediante $1/\tau$.

Osservazione 3.13 (Scelta di τ). La precedente osservazione permette anche di esplicitare un metodo con cui decidere la dimensione temporale τ : si consideri $\Delta t = 0.01$, allora la frequenza è di 100 interazioni nell'unità di tempo. In un fenomeno come quello urbano si possono allora escludere due scale:

- 592 ◊ quella dei mesi o superiori, perché è irrealistico che in un arco tem-
593 porale così lungo vi siano solo 100 interazioni tra le città;
- 594 ◊ similmente quella dei secondi o inferiori per un ragionamento simile
595 a quello di prima ma opposto.

È pertanto naturale che la scala ideale in tal contesto sia quella dei giorni.

Ciononostante, in questo scritto non si studia un'equazione di tipo Boltzmann dimensionale analoga alla (3.21) per tre principali motivazioni:

- 599 1. la dimensione va considerata solo se si è interessati a studiare il
600 transitorio, non la distribuzione stazionaria;
- 601 2. lo studio di quest'ultimo è incompatibile coll'Ip. 2.2 di grafo statico;
- 602 3. mancano i dati storici, sia attendibili che su scale temporali adeguate,
603 per poter confrontare il transitorio simulato con quello reale.

Tali sono le ragioni per cui il tempo sarà sempre considerato adimensionale da qui in poi: l'obiettivo è studiare la distribuzione stazionaria raggiunta, non in quanto tempo si raggiunge (tempo di convergenza).

607 3.3 DESCRIZIONE CINETICA URBANA SU RETI

608 Si analizza adesso il caso del tessuto interurbano descritto nel § 2.2, considerando le città sia come agenti che come nodi di una rete che ne regola le 609 interazioni: due città possono interagire se sono connessi; è anche ovvio che 610 l'interazione modifichi, in qualche modo da definire, la loro popolazione. 611

612 In questo paragrafo si applicano i concetti della teoria sviluppata nei §§ 2.1 e 613 3.2.3 e, in particolare, la notazione usata e definita. S'inizia mostrando una 614 derivazione esatta la quale, cioè, considera la topologia esatta; quindi si 615 approfondisce un'approssimazione nella quale si "perde" la topologia indotta 616 dalla rete; infine, si considerano i collegamenti tra la derivazione esatta e 617 quella di tipo Boltzmann sotto determinate ipotesi.

618 3.3.1 Equazione di tipo Boltzmann esatta

619 *Descrizione e derivazione*

620 Visto che si assume che valgono le Ip. 3.4⁷, dalla B₃ sorge un problema non 621 di poco conto: gli agenti sono indistinguibili ma la rete sottostante li rende 622 distinti per via degl'indici \mathcal{I} .

623 Per dipanare la questione è sufficiente considerare l'indice come una 624 variabile aleatoria $I \in \mathcal{I}$ così da definire il processo stocastico come

$$\{\mathbf{X}_t\} \equiv \{(I, S_t)\}_{t \in \mathbb{R}_+},$$

625 in cui $S_t \in \mathbb{R}_+$ è la variabile aleatoria relativa alla popolazione dell'agente 626 rappresentativo al tempo t ed è l'unica componente del processo stocastico 627 a variare nel tempo per l'Ip. 2.2.

628 **Osservazione 3.14** (Sulla natura numerica della popolazione). È naturale 629 che non esistano frazioni di persone e che quindi rigorosamente $S_t \in \mathbb{N}$ ma 630 non sempre la scelta più realistica è più modellisticamente agevole; infatti 631 trattare S_t come una variabile aleatoria discreta impone un codominio, ap- 632 punto, discreto che è in genere più difficile da manipolare matematicamente 633 rispetto a un intervallo continuo. Tal'è la ragione nel scegliere S_t reale: dopo 634 aver fatto i conti normalmente si può poi approssimare per eccesso o difetto 635 ricavando la taglia intera effettiva; l'errore così commesso è di al più una 636 persona.

637 Così ragionando si può nel complesso descrivere statisticamente lo stato 638 microscopico $\mathbf{X}_t \equiv (I, S_t)$ dell'agente rappresentativo colla densità congiunta

$$f(i, s, t) : \mathcal{I} \times \mathbb{R}_+ \times [0, +\infty) \rightarrow \mathbb{R}_+,$$

639 che è discreta in $i \in \mathcal{I}$ e continua in $s \in \mathbb{R}_+$, uguale a

$$f(i, s, t) \equiv \frac{1}{N} \sum_{j \in \mathcal{I}} f_j(s, t) \otimes \delta(i - j), \quad (3.22)$$

⁷ Perdipiù la B₂ è valida a un tempo t , ma nell'unità di tempo dimensionale ſ le interazioni si possono considerare come non binarie, come già discusso alla fine del § 3.2.3.

dove $N \equiv |\mathcal{J}|$ è il numero totale d'agenti del grafo mentre $\delta(\cdot)$ denota la delta di Dirac centrata all'origine; d'altra parte

$$f_i = f_i(s, t) : \mathbb{R}_+ \times [0, +\infty) \rightarrow \mathbb{R}_+$$

è la densità di S_t dell'agente i -esimo.

In tal modo la densità congiunta f è effettivamente indistinguibile rispetto a tutti gli agenti, soddisfacendo la [B3](#), e integra sul suo dominio a uno:

$$\begin{aligned} \mathbb{P}_t(\mathcal{J} \times \mathbb{R}_+) &= \int_{\mathcal{J}} \int_{\mathbb{R}_+} f(i, s, t) ds di = \frac{1}{N} \sum_{i \in \mathcal{J}} \int_{\mathcal{J}} \int_{\mathbb{R}_+} f_i(s, t) ds \otimes \delta(i-j) di \\ &= \frac{1}{N} \sum_{i \in \mathcal{J}} \int_{\mathcal{J}} \delta(i-j) di = \frac{1}{N} \sum_{i \in \mathcal{J}} 1 = \frac{N}{N} = 1, \quad \forall t \geq 0; \end{aligned}$$

mentre le densità f_i preservano la naturale distinguibilità degli agenti indotta dalla topologia sottostante.

Sempre come conseguenza del grafo, la [\[Ber\]x](#) deve dipendere dalla matrice d'adiacenza A :

$$\Theta \sim \text{Bernoulli}(A(I, I_*) \Delta t), \quad [\text{Ber}]_S^A$$

ove $A : \mathcal{J} \times \mathcal{J} \rightarrow \mathbb{R}$ è una funzione che a ogni coppia d'indici (I, I_*) associa la relativa entrata nella matrice d'adiacenza

$$A(I, I_*) \equiv a_{I, I_*} = \begin{cases} 1 & \text{se } (I, I_*) \in \mathcal{E}, \\ 0 & \text{altrimenti,} \end{cases}$$

secondo la [\(2.2\)](#); in tal modo l'interazione non ha possibilità d'avvenire se i due agenti non sono connessi.

Osservazione 3.15. Rispetto al caso generale [\[Ber\]x](#) il tasso d'interazione ha forma

$$\mu(i, i_*) \equiv A(i, i_*), \quad (3.23)$$

esso, cioè, è per definizione costante sia rispetto agli stati S_t e S_t^* che rispetto al tempo, oltre a non avere coefficienti⁸ w .

Osservazione 3.16. Per definizione della distribuzione di Bernoulli, è necessario imporre $A(I, I^*) \Delta t \leq 1$, ma dato che $A(I, I^*) \in \{0, 1\}$ ciò si traduce nella naturale condizione che $\Delta t \leq 1$.

D'altro canto le regole d'interazione nella [\[RI\]x](#) diventano

$$[\text{RI}]_S \left\{ \begin{array}{l} S'_t = \psi(S_t, I, S_t^*, I_*, \gamma), \\ S_t^{*\prime} = \psi_*(S_t, I, S_t^*, I_*, \gamma). \end{array} \right.$$

Osservazione 3.17. Confrontato a [\[RI\]x](#), solo la funzione della città interagente ψ presenta un coefficiente stocastico $y \equiv \gamma \in \mathbb{R}$, mentre quella relativa alla città ricevente ψ_* non dipende da potenziali coefficienti⁹ y_* .

⁸ Questa situazione si può simbolicamente rappresentare ponendo $k = 0$.

⁹ Come nell'Oss. [3.15](#), tale condizione si può simbolicamente rappresentare ponendo $h_* = 0$.

664 Unendo le $[Ber]_S^A$ e $[RI]_S$, le $[AR]_X$ si scrivono

$$[AR]_S \begin{cases} S_{t+\Delta t} = (1-\Theta)S_t + \Theta S'_t, \\ S_{t+\Delta t}^* = (1-\Theta)S_t^* + \Theta S_t^{**}, \end{cases}$$

665 coerentemente col fatto che in un cointesto urbano, qualora una città-nodo
666 interagisca con un'altra, entrambe debbano variare il loro stato (Oss. 3.9).

667 **Osservazione 3.18.** Paragonato a $[AR]_X$ ci si potrebbe chiedere perché non
668 si considera l'intero vettore aleatorio X_t , come pure nella $[RI]_S$; la ragione è
669 che I è una componente statica che non varia nel tempo.

670 In ogni caso, considerando I_t e I_t^* momentaneamente dinamiche, si possono
671 definire le regole d'interazione

$$[RI]_I \begin{cases} I'_t = \psi(S_t, I, S_t^*, I_*) \equiv I_t, \\ I_t^{**} = \psi_*(S_t, I, S_t^*, I_*) \equiv I_t^*, \end{cases}$$

672 da cui segue l'algoritmo d'interazione

$$[AR]_I \begin{cases} I_{t+\Delta t} = (1-\Theta)I_t + \Theta I'_t = (1-\Theta)I_t + \Theta I_t = I_t, \\ I_{t+\Delta t}^* = (1-\Theta)I_t^* + \Theta I_t^{**} = (1-\Theta)I_t^* + \Theta I_t^* = I_t^*, \end{cases}$$

673 che soddisfa la staticità di I e I_* e completa, assieme a $[AR]_S$, la formulazione
674 $[AR]_X$ più generale.

675 Sia $\Phi: \mathcal{I} \times \mathbb{R}_+ \rightarrow \mathbb{R}$ un'arbitraria quantità osservabile, dalla $[Ber]_S^A$ l'equa-
676 zione di tipo Boltzmann omogenea $[EtB]_X^A$ ha forma

$$\frac{d}{dt} \int_{\mathcal{I}} \int_{\mathbb{R}_+} \Phi f dv di = \int_{\mathcal{I}^2} \int_{\mathbb{R}_+^2} A(i, i_*) \frac{\langle \Phi' + \Phi_*' - \Phi - \Phi_* \rangle}{2} ff_* ds ds_* di di_*, \quad [EtB]_S^A$$

677 ove si è usata la notazione abbreviata illustrata nel § 3.2.3, mentre $\langle \cdot \rangle$ indica
678 il valore atteso rispetto alla variabile aleatoria γ nelle $[RI]_S$.

679 **Osservazione 3.19.** Anche se la derivazione della $[EtB]_S^A$ è già stata spiegata
680 nel dettaglio nel § 3.2.1, rimane interessante riportare la forma del passaggio
681 più difficile, ossia quella della media condizionata (3.8) (qui mostrato solo
682 per $S_{t+\Delta t}$):

$$\begin{aligned} \mathbb{E}[\Phi(I, S_{t+\Delta t})] &= \mathbb{E}[\mathbb{E}[\Phi(I, (1-\Theta)S_t + \Theta\psi(S_t, I, S_t^*, I_*, \sigma)A(I, I_*)\Delta t) | I, I_*]] \\ &= \mathbb{E}[\mathbb{E}[\Phi(I, S_t)(1 - A(I, I_*)\Delta t) \\ &\quad + \Phi(I, \psi(S_t, I, S_t^*, I_*, \sigma))A(I, I_*)\Delta t]]. \end{aligned}$$

683 Si noti come il condizionamento sia solo rispetto a I a I_* , proprio poiché Θ
684 in $[Ber]_S^A$, tramite il tasso d'interazione μ nella (3.23), dipende solo da essi.

685 *Analisi delle regole d'interazione*

686 Si approfondiscono in questo paragrafo le regole d'interazione $[RI]_S$.

687 Innanzitutto, il perché nella $[AR]_S$ si considerano interazioni «Azione-
688 Reazione» (Oss. 3.9) discende da un vincolo fisico: se una città interagisce
689 con un'altra, scambiando popolazione, necessariamente anche l'altra varia il
690 proprio stato.

Dopodiché, passando alle realizzazioni di tutte le variabili aleatorie considerate, vale a dire scrivendole in minuscolo e omettendo la dipendenza dal tempo, le regole d'interazione [RI]_S sono così definite¹⁰

$$[\text{RE}] \begin{cases} \psi(s, i, s_*, i_*; \gamma) = s(1 - E(s, i, s_*, i_*) + \gamma) \\ \psi_*(s, i, s_*, i_*) = s_* + sI(s, i, s_*, i_*) \end{cases}$$

ove

$$E: \mathbb{R}_+ \times \mathbb{R}_+ \times \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}_+,$$

$$I: \mathbb{R}_+ \times \mathbb{R}_+ \times \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}_+,$$

sono rispettivamente i tassi d'emigrazione e immigrazione, i cui argomenti sono per brevità sottintesi da qui in poi, mentre γ rappresenta fluttuazioni stocastiche. Si caratterizza ulteriormente E :

Ipotesi 3.5. Per avere un postato s fisicamente sensato, ossia positivo, quando $\gamma = 0$, si assume che il tasso d'emigrazione E sia superiormente limitato dalla costante $\lambda \equiv \sup E$, tale che $\lambda \in (0, 1)$, e inferiormente limitato dall'origine:

$$0 \leq E \leq \lambda < 1, \quad \forall s, \forall s_*, \forall i, \forall i_*.$$

D'altra parte il postato s'_* [per il momento] non si considera poiché è per definizione sempre positivo qualunque sia il tasso d'immigrazione $I \in [0, +\infty)$.

Se invece γ è non nullo, dev'essere anch'esso limitato:

$$s' = s(1 - E + \gamma) > 0 \implies \gamma > E - 1, \quad \forall E \in [0, \lambda], \quad (3.24)$$

La scelta di $>$ anziché \geq nella (3.24) è ben fondata: di fatto si sta supponendo che le fluttuazioni non possano spopolare una città¹¹; tale condizione discende dal rapporto s_*/s contenuto nelle regole d'emigrazione proposte nel § 4.3.

Perdipiù, le fluttuazioni rappresentano a grandi linee quei fenomeni complessivi di nascita e di morte che vengono considerati ma non direttamente modellati; pertanto valgono le seguenti ipotesi:

Ipotesi 3.6. γ deve soddisfare le seguenti caratteristiche:

F₁ può assumere sia valori positivi che negativi, ma non minori del vincolo imposto da (3.24);

F₂ la media è scelta arbitrariamente posta allo zero, ossia $\langle \gamma \rangle = 0$;

F₃ seppure non vi siano limiti superiori per l'entità della fluttuazione, è chiaro che più grande questa è meno è probabile.

Con queste si possono allora analizzare alcune distribuzioni continue:

1. la distribuzione normale $N(\mu, \sigma^2)$ non soddisfa la F₁ poiché può assumere valori reali arbitrari con probabilità non nulla;

¹⁰ Sono state in parte ispirate dalla [10], (2.1) p. 223.

¹¹ Ciò non significa che il modello non possa prevedere un tale fenomeno, poiché la taglia può arbitrariamente avvicinarsi a zero ma mai esserne uguale, raggiungendolo solo *a posteriori* dopo l'approssimazione dai numeri reali a quelli interi (Oss. 3.14).

- 721 2. la distribuzione uniforme $\mathcal{U}([a,b])$ è adeguata solo per intervalli finiti
 722 e diventa degenere quando un suo estremo diverge, per cui non
 723 soddisfa né **F₂** né **F₃**, mentre **F₁** sì;
- 724 3. la distribuzione esponenziale $\text{Exp}(\lambda)$ è quella più promettente per-
 725 ché riflette sia **F₂** (dopo un'opportuna traslazione dei valori campio-
 726 nati) che **F₃**, ma sfortunatamente non **F₁** perché la densità è non
 727 nulla al valore estremo $\gamma = E - 1$;
- 728 4. l'unica distribuzione che soddisfa tutte e tre le caratteristiche ricercate
 729 è proprio la distribuzione $\text{Gamma}(\alpha, \beta)$.

730 Si assuma allora $\hat{\gamma} \sim \text{Gamma}(\alpha, \beta)$ con densità

$$f_{\hat{\gamma}}(x) = \frac{1}{\beta^{\alpha} \Gamma(\alpha)} x^{\alpha-1} \exp\left(-\frac{x}{\beta}\right),$$

731 ove α e β sono i parametri rispettivamente di forma e di scala, mentre

$$\Gamma(\alpha) : \mathbb{R}_+ \rightarrow \mathbb{R}_+$$

732 è la funzione gamma

$$\Gamma(\alpha) \equiv \int_0^{+\infty} y^{\alpha+1} e^{-y} dy.$$

733 Imponendo la **F₁** si ottiene

$$\langle \hat{\gamma} \rangle = 1 - E = \alpha\beta \quad \text{e} \quad \text{var}(\hat{\gamma}) \equiv \sigma^2 = \alpha\beta^2,$$

734 in cui σ^2 indica la varianza, da cui

$$\alpha = \frac{(1-E)^2}{\sigma^2} \quad \text{e} \quad \beta = \frac{\sigma^2}{1-E}.$$

735 La **F₂** si può semplicemente soddisfare traslando i valori campionanti della
 736 $\hat{\gamma}$ di $-\langle \hat{\gamma} \rangle$, ossia si considera la distribuzione $\gamma \sim \hat{\gamma} - \langle \hat{\gamma} \rangle$:

$$\langle \gamma \rangle = \langle \hat{\gamma} - \langle \hat{\gamma} \rangle \rangle = \langle \hat{\gamma} \rangle - \langle \hat{\gamma} \rangle = 0.$$

737 D'altra parte per la **F₁** bisogna salvaguardarsi dai casi degeneri della distri-
 738 buzione gamma: essa infatti se $\alpha = 1$ diventa $\text{Exp}(\beta)$, mentre se $\alpha < 1$ diverge
 739 all'origine; per avere quindi probabilità nulla di campionare da $\hat{\gamma}$ l'origine
 740 [e quindi $-\langle \hat{\gamma} \rangle$ dopo la traslazione] è necessario porre

$$\alpha > 1 \implies \frac{(1-E)^2}{\sigma^2} \geq \frac{(1-\lambda)^2}{\sigma^2} > 1 \implies \sigma < |1-\lambda| = 1-\lambda. \quad (3.25)$$

741 La (3.25) implica quindi che non è possibile avere una varianza arbitraria per
 742 poter soddisfare la **F₁**, ma che questa è limitata superiormente dal massimo
 743 tasso d'emigrazione λ , interpretabile come attrattività: più grande è λ più
 744 piccola è la varianza, e viceversa.

745 In realtà la σ non può davvero assumere valori arbitrari tra 0 e $1-\lambda$: difatti
 746 se $\sigma \rightarrow 1-\lambda$ vale $\alpha \rightarrow 1$ e dunque la distribuzione gamma tende a un'espo-
 747 nenziale; questo ha l'effetto d'introdurre una pericolosa asimmetria nella
 748 densità, come si può notare nella la Fig. 3.1: una volta centrata rispetto al-
 749 la media la moda (il valore più frequente) ha valore negativo portando, in

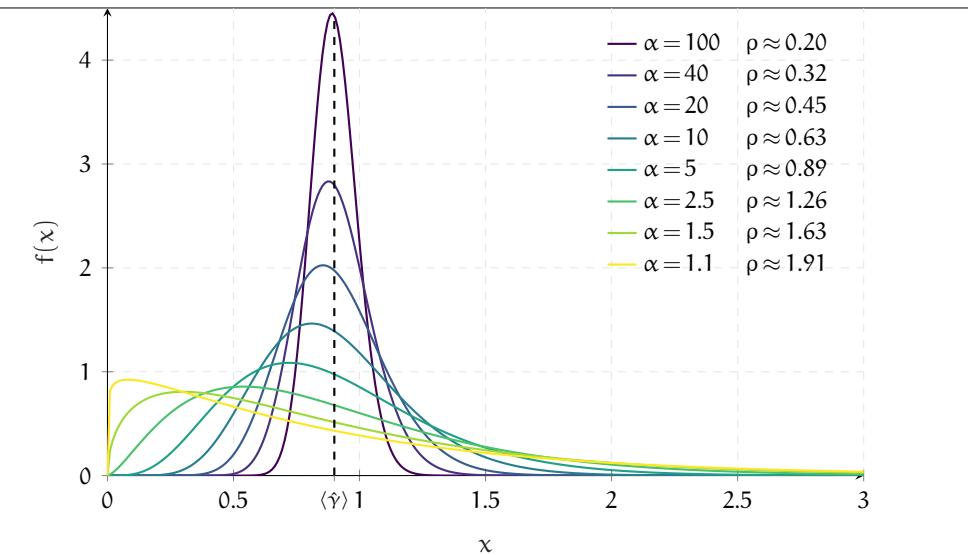


Figura 3.1: Transizione della densità gamma verso l'asimmetria con $\lambda = 0.1$ ed $E = \lambda$, da cui $\langle \hat{\gamma} \rangle = 0.9$ e $\beta = \langle \hat{\gamma} \rangle / \alpha$.

un orizzonte temporale finito, le perturbazioni negative a essere quelle più frequenti.

Il risultato è che con σ molto elevato le città tendono a spopolarsi perché v'è una preferenza per fluttuazioni negative. Per ovviare al problema bisogna ulteriormente vincolare σ dimodoché l'indice di asimmetria

$$\rho \equiv \frac{2}{\sqrt{\alpha}}$$

sia sufficientemente piccolo; dalla Fig. 3.1 si può perciò imporre $\rho \leq 0.2$, cioè

$$\alpha \geq 100 \implies \frac{(1-E)^2}{\sigma^2} \geq 100 \implies \sigma \leq \frac{1-\lambda}{10}, \quad (3.26)$$

dove, come nella (3.25), si è imposto nell'ultima implicazione il caso peggiore $E = \lambda$. Dunque il vero limite superiore, per preservare la simmetria, è di un ordine di grandezza inferiore a quello precedentemente stimato.

Osservazione 3.20. Quello qui mostrato è solo un possibile modello stocastico perturbativo; infatti, modificando le Ip. 3.6, se ne può scegliere un altro: ad esempio si può imporre una distribuzione uniforme oppure si può troncare una gaussiana affinché soddisfi il vincolo di positività (3.24).

Non manca che caratterizzare il tasso d'immigrazione I:

Ipotesi 3.7. Si postula che l'interazione 14 conservi [in media] la popolazione totale:

$$s + s_* = \langle s + s_* \rangle = \langle s' + s'_* \rangle \stackrel{F2}{=} s - sE + s_* + sI,$$

da cui $E \equiv I$, che è ragionevole siccome l'emigrazione e l'immigrazione sono fenomeni relativi (invertendo s ed s_* sarebbe l'opposto).

Osservazione 3.21. Le 14 non sono ancora complete siccome non si è esplorata la regola d'emigrazione, la cui scelta, tuttavia, si può dunque vedere come la regola d'interazione stessa: è l'ultimo tassello del mosaico. Pertanto

771 si lascia quest'ultima definizione al § 4.3 nel quale se ne propongono varie,
 772 si discutono quindi con vari studi parametrici per poi infine interpretarle
 773 analizzando cosa dicono sul fenomeno della migrazione.

774 3.3.2 Equazione di tipo Boltzmann approssimata

775 L'approssimazione si fonda sulla seguente matrice:

776 **Definizione 3.12** (Matrice dei gradi \mathbf{B}). Sia $\mathbf{B} \in \mathbb{R}^{N \times N}$ una matrice di rango
 777 uno

$$\mathbf{B} \equiv \frac{\mathbf{k}^+(\mathbf{k}^-)^\top}{D_N}$$

778 definite tramite il prodotto diadico dei vettori

$$\begin{aligned}\mathbf{k}^+ &\equiv (k_1^+, k_2^+, \dots, k_N^+)^\top = \mathbf{A}\mathbf{1} \in \mathbb{R}^N, \\ \mathbf{k}^- &\equiv (k_1^-, k_2^-, \dots, k_N^-)^\top = \mathbf{A}^\top \mathbf{1} \in \mathbb{R}^N,\end{aligned}\quad (3.27)$$

779 rispettivamente dei gradi uscenti ed entranti, e la costante¹²

$$D_N \equiv \sum_{i \in J} k_i^+ = \mathbf{1}^\top \mathbf{k}^+ = \sum_{i \in J} k_i^- = \mathbf{1}^\top \mathbf{k}^- = \mathbf{1}^\top \mathbf{A} \mathbf{1} = \|\mathbf{A}\|_1. \quad (3.28)$$

780 Allora \mathbf{B} approssima la matrice d'adiacenza \mathbf{A} poiché per definizione vale

$$\left. \begin{aligned}\mathbf{B}\mathbf{1} &= \frac{1}{D_N} \mathbf{k}^+(\mathbf{k}^-)^\top \mathbf{1} = \mathbf{k}^+ \frac{D_N}{D_N} = \mathbf{k}^+ \\ \mathbf{B}^\top \mathbf{1} &= \frac{1}{D_N} \mathbf{k}^-(\mathbf{k}^+)^\top \mathbf{1} = \mathbf{k}^- \frac{D_N}{D_N} = \mathbf{k}^-\end{aligned}\right\} \implies \mathbf{B} \approx \mathbf{A}.$$

781 Inoltre, poiché \mathbf{A} è simmetrica (Ip. 2.1) vale $\mathbf{k}^+ = \mathbf{k}^- = \mathbf{k}$ e quindi

$$\mathbf{B} = \frac{\mathbf{k}\mathbf{k}^\top}{D_N}.$$

782 Tramite la Def. 3.12, la $[\text{Ber}]_S^A$ diventa

$$\Theta \sim \text{Bernoulli}(B(I, I^*) \Delta t), \quad [\text{Ber}]_S^B$$

783 da cui la $[\text{EtB}]_S^A$ si legge

$$\frac{d}{dt} \int_J \int_{\mathbb{R}_+} \Phi g dv di = \int_J \int_{\mathbb{R}_+^2} B(i, i_*) \frac{\langle \Phi' + \Phi'_* - \Phi - \Phi_* \rangle}{2} gg_* ds ds_* didi_*, \quad [\text{EtB}]_S^B$$

784 dove g e g_* sono densità definite come la (3.22) ma indicate diversamente
 785 per distinguerle da quelle esatte f ed f_* .

786 Sorge però spontanea una domanda non di poco conto: perché non si
 787 può scegliere un'altra matrice $\mathbf{C} \in \mathbb{R}^{N \times N}$ per approssimare la \mathbf{A} , secondo
 788 altri criteri analoghi o dissimili alla (3.27)? La risposta non è immediata ma
 789 discende in essenza su come la $[\text{EtB}]_S^B$ può essere riformulata. A tal scopo si
 790 necessita di un osservazione:

¹² La $\|\cdot\|$ è la norma uno applicata alle matrici: $\|\mathbf{A}\|_1 \equiv \sum_{i,j \in J} |a_{i,j}|$.

Osservazione 3.22 (Teoria e Pratica). Come detto nel Cap. 1, l'interesse è di studiare la distribuzione della popolazione tra città; perciò, non a torto, si può vedere la densità f nella (3.22) come eccessivamente dettagliata, contenendo informazioni legate ai vertici: ai fini pratici è quindi sufficiente ricavare in qualche modo la densità marginale $\bar{f}(s,t) \equiv \int_J f(i,s,t) di, \forall t \geq 0$, che è esattamente quanto fatto nel § 4.1 per i risultati delle venture simulazioni.

Dall'Oss. 3.22 si possono pertanto specializzare gli osservabili:

Ipotesi 3.8 (Osservabili puntuali). Si considera la seguente classe di osservabili: $\Phi(i,s) \equiv \delta(i-j)\varphi(s), \forall j \in J$, dove $\varphi : \mathbb{R}_+ \rightarrow \mathbb{R}$ è una funzione arbitraria.

Con tale scelta è possibile recuperare un sistema di equazioni debole per le g_i : usando, infatti, l'Ip. 3.8 nella [EtB]_S^B porta a

$$\begin{aligned} \frac{d}{dt} \int_{\mathbb{R}_+} \varphi g_j ds &= \frac{1}{N} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' - \varphi \rangle}{2} g_j \sum_{i \in J} B(j,i) g_i^* ds ds_* \\ &\quad + \frac{1}{N} \int_{\mathbb{R}_+^2} \frac{\langle \varphi'_* - \varphi_* \rangle}{2} s_j^* \sum_{i \in J} B(i,j) g_i ds ds_*, \quad \forall j \in J, \end{aligned}$$

ma essendo B simmetrica vale

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi g_j ds = \frac{1}{N} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} g_j \sum_{i \in J} B(j,i) g_i^* ds ds_* \quad \forall j \in J,$$

che in forma matriciale, introducendo le densità vettoriali

$$\mathbf{g} \equiv (g_i)_{i \in J} \quad \text{e} \quad \mathbf{g}^* \equiv (g_i^*)_{i \in J},$$

si legge

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi \mathbf{g} ds = \frac{1}{N} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} \mathbf{g} \odot \mathbf{B} \mathbf{g}^* ds ds_*, \quad (3.29)$$

ove \odot indica il prodotto di Hadamard (prodotto per elementi).

Il prossimo passo è introdurre la densità globale delle taglie, ossia la densità marginale della (3.22) rispetto alla popolazione s :

$$\bar{g}_N(s,t) \equiv \int_J g(i,s,t) di = \frac{1}{N} \sum_{i \in J} g_i = \frac{1}{N} \mathbf{1}^\top \mathbf{g}, \quad (3.30)$$

ove $\mathbf{1} \equiv (1, 1, \dots, 1) \in \mathbb{R}^N$, che non è che la media tra le densità di S_t tra tutti i vertici¹³; premoltiplicando la (3.29) per $\frac{1}{N} \mathbf{1}^\top$ e usando la (3.30) si ha

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi \bar{g} ds = \frac{1}{N^2} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} \mathbf{g}^\top \mathbf{B} \mathbf{g}^* ds ds_*, \quad [\mathbf{E}\bar{g}]$$

Osservazione 3.23. La [E \bar{g}] si può anche ottenere usando, in luogo dell'Ip. 3.8, direttamente un osservabile indipendente dall'indice i : $\Phi(i,s) \equiv \varphi(s)$, confermando quanto premesso nell'Oss. 3.22, cioè che la [E \bar{g}] si ricava perdendo le informazioni legati agli indici negli osservabili.

¹³ La medesima definizione vale anche per \bar{f}_N della [EtB]_S^A.

814 La [Eq] non è ancora un'equazione chiusa per \bar{g} per via del secondo mem-
 815 bro nel quale $\mathbf{g}^\top \mathbf{B} \mathbf{g}_*$ richiede sia di conoscere nel dettaglio le connessio-
 816 ni sottostanti del grafo che le distribuzioni della popolazione di ogni città.
 817 Tuttavia, avvalendosi della definizione della \mathbf{B} , la [Eq] diventa

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi \bar{g}_N ds = \frac{1}{N^2} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2D_N} (\mathbf{k}^\top \mathbf{g})(\mathbf{k}^\top \mathbf{g}_*) ds ds_*, \quad (3.31)$$

818 che richiede d'introdurre una nuova densità di probabilità:

819 **Definizione 3.13** (Densità dei gradi \mathbf{k}). Sia

$$d_N : \mathbb{R}_+ \times \{0, 1, \dots, N\} \times \mathbb{R}_+ \rightarrow \mathbb{R}_+,$$

820 la densità di probabilità dell'evento che al tempo t un agente abbia popola-
 821 zione s e grado k , definizione che la lega alle g_i dalla relazione

$$d_N(s, k, t) = \frac{1}{N} \sum_{i \in \mathcal{J}_k} g_i(s, t),$$

822 ove

$$\mathcal{J}_k \equiv \{i \in \mathcal{J} \mid k_i = k\} \subseteq \mathcal{J} \quad (3.32)$$

823 è l'insieme dei nodi con grado k ; da essa ne seguono altre due:

$$\begin{aligned} \bar{g}_N(s, t) &= \frac{1}{N} \sum_{i \in \mathcal{J}} g_i(s, t) = \sum_{k=0}^N \frac{1}{N} \sum_{i \in \mathcal{J}_k} g_i(s, t) = \sum_{k=0}^N d_N(s, k, t), \\ \mathbf{k}^\top \mathbf{g}(s, t) &= \sum_{i \in \mathcal{J}} k_i g_i(s, t) = \sum_{k=0}^N k \sum_{i \in \mathcal{J}_k} g_i(s, t) = N \sum_{k=0}^N k d_N(s, k, t). \end{aligned} \quad (3.33)$$

824 Un analogo discorso vale per le f_i in luogo delle g_i .

825 E dalla Def. 3.13 segue la sua normalizzazione:

826 **Definizione 3.14** (Densità dei gradi \mathbf{k} normalizzati). Siano

$$\hat{k}_i \equiv \frac{k_i}{N} \in \mathcal{K}, \quad \forall i \in \mathcal{J},$$

827 i gradi normalizzati di un generico nodo rappresentativo di \mathcal{G} , in cui

$$\mathcal{K} \equiv \left\{ \frac{k}{N} \mid k = 0, 1, \dots, N \right\} \subseteq [0, 1]$$

828 è l'insieme discreto dei gradi normalizzati; allora la densità dei gradi norma-
 829 lizzati

$$\hat{d}_N : \mathbb{R}_+ \times \mathcal{K} \times \mathbb{R}_+ \rightarrow \mathbb{R}_+,$$

830 è così definita

$$\hat{d}_N(s, \hat{k}, t) \equiv N d_N(s, N \hat{k}, t). \quad (3.34)$$

831 **Osservazione 3.24.** Visto che due elementi consecutivi dell'insieme \mathcal{K} sono
 832 separati da un passo costante e uguale a $1/N$, si può definire $\Delta \hat{k} \equiv 1/N$ che
 833 da (3.34) implica

$$\sum_{\hat{k} \in \mathcal{K}} \int_{\mathbb{R}_+} \hat{d}_N(s, \hat{k}, t) ds \Delta \hat{k} = \sum_{k \in \mathcal{K}} \int_{\mathbb{R}_+} d_N(s, k, t) ds = 1, \quad \forall t \geq 0,$$

834 cosa che permette di vedere la \hat{d}_N come funzione costante a tratti dei gradi
 835 normalizzati \hat{k} , e similmente per d_N (basta definire $\Delta k \equiv 1$) essendone un
 836 numero riscalamento.

837 **Osservazione 3.25.** L'Oss. 3.24 dà la possibilità anche d'interpretare la (3.33)
 838 come un uguaglianza tra densità marginali: $\bar{g}_N = \bar{d}_N = \bar{d}_N$.

839 Con tutte queste definizioni e osservazioni, la (3.31) si può riformulare in
 840 termini di \hat{d}_N

$$\frac{d}{dt} \sum_{k=0}^N \int_{\mathbb{R}_+} \varphi d_N ds = N^2 \sum_{k, k_*=0}^N \int_{\mathbb{R}_+^2} \frac{kk_* \langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{D_N} d_N d_N^* ds ds_*,$$

841 dove k_* è il grado associato a g_N^* , e poi in termini di \hat{d}_N

$$\underbrace{\frac{d}{dt} \sum_{k \in \mathcal{K}} \int_{\mathbb{R}_+} \varphi d_N ds \Delta k}_{\textcircled{I}} = \underbrace{\sum_{k, k_* \in \mathcal{K}} \int_{\mathbb{R}_+^2} \frac{kk_* \langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{D_N} d_N d_N^* ds ds_*}_{\textcircled{II}}.$$

842 Il secondo membro si può ulteriormente manipolare moltiplicando e divi-
 843 dendo per N^2 e definendo il grado normalizzato medio

$$\bar{D}_N \equiv \frac{D_N}{N^2} = \sum_{k \in \mathcal{K}} \int_{\mathbb{R}_+} k d_N ds \Delta k, \quad (3.35)$$

844 da cui

$$\textcircled{II} = \sum_{k, k_* \in \mathcal{K}} \int_{\mathbb{R}_+^2} \frac{kk_* \langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{\bar{D}_N} d_N d_N^* ds ds_* \Delta k \Delta k_*.$$

845 Valutando il limite $N \rightarrow \infty$ alla (3.35) e ai membri \textcircled{I} e \textcircled{II}

$$\begin{aligned} (3.35) &\xrightarrow{N \rightarrow \infty} \bar{D} \equiv \int_0^1 \int_{\mathbb{R}_+} k d_N ds dk, \quad \textcircled{I} \xrightarrow{N \rightarrow \infty} \frac{d}{dt} \int_0^1 \int_{\mathbb{R}_+} \varphi d_N ds dk \\ &\textcircled{II} \xrightarrow{N \rightarrow \infty} \int_0^1 \int_0^1 \int_{\mathbb{R}_+^2} \frac{kk_* \langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{\bar{D}} d_N d_N^* ds ds_* dk dk_*, \end{aligned}$$

846 si perviene infine a un'equazione di tipo Boltzmann:

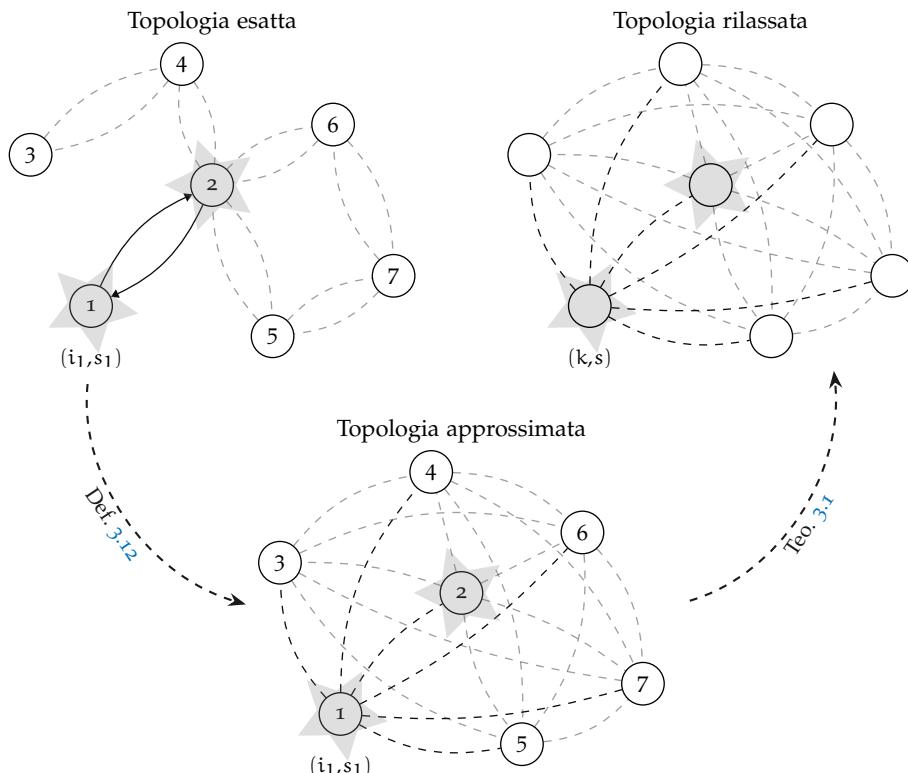
$$\frac{d}{dt} \int_0^1 \int_{\mathbb{R}_+} \varphi d_N ds dk = \int_0^1 \int_0^1 \int_{\mathbb{R}_+^2} \frac{kk_* \langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{\bar{D}} d_N d_N^* ds ds_* dk dk_*. \quad [\text{EtB}]_S^k$$

847 Si è dunque dimostrato il seguente teorema:

848 **Teorema 3.1** (di rilassamento della topologia). La $[\text{EtB}]_S^B$ è formalmente equi-
 849 valente nel limite $N \rightarrow \infty$ a un'equazione di tipo Boltzmann di forma $[\text{EtB}]_X$ con
 850 microstato $X_t \equiv \{\mathbf{k}, S_t\}$, ove $\mathbf{k} \in [0, 1]$ è la variabile aleatoria dei gradi normalizzati,
 851 osservabili $\varphi(s)$ indipendenti da \mathbf{k} e nucleo d'interazione $\mu(\mathbf{k}, \mathbf{k}_*) \equiv (kk_*)/\bar{D}$.

852 Il significato del Teo. 3.1 è profondo: approssimare $[\text{EtB}]_S^A$ con $[\text{EtB}]_S^B$ coin-
 853 cide col perdere la distinguibilità indotta dal grafo, rilassando in tal modo
 854 la topologia la quale non scompare ma rimane solo come distribuzione dei
 855 gradi; tale riduzione è come sfocare i dettagli delle connessioni tra i vertici;
 856 si passa da uno specifico grafo a una classe di grafi. Nella Fig. 3.2 è presente
 857 uno schema riassuntivo del corrente paragrafo.

$$\begin{array}{ccc}
 \bar{\mathbf{d}}_N & \xrightarrow{N \rightarrow \infty} & \bar{\mathbf{d}} \\
 \text{Teo. 3.1} \parallel & & \parallel \\
 \bar{\mathbf{g}}_N & \xrightarrow{N \rightarrow \infty} & \bar{\mathbf{g}} \\
 \text{Def. 3.12} \parallel & & \parallel \\
 \bar{\mathbf{f}}_N & \xrightarrow{N \rightarrow \infty} & \bar{\mathbf{f}}
 \end{array}
 \begin{array}{c}
 \text{Topologia rilassata} \\
 \text{Topologia esatta (B)} \\
 \text{Topologia esatta (A)}
 \end{array}$$

Figura 3.2: Schema riassuntivo del § 3.3.2.**Figura 3.3:** Rappresentazione grafica dell'approssimazione.

858 **Osservazione 3.26.** Il Teo. 3.1 motiva anche il perché non si possa considerare una generica matrice \mathbf{C} che approssima la \mathbf{A} : non è detto che rilassi la
859 topologia portando a un'equazione analoga alla $[EtB]_S^k$.
860

861 **Osservazione 3.27.** La Fig. 3.2 contiene al suo interno $\hat{\mathbf{f}}$ che ha un'analogia
862 definizione alla Def. 3.14, ma normalizzando gli indici: siano

$$\hat{i} \equiv \frac{i}{N} \in \mathcal{J}, \quad \forall i \in \mathcal{I},$$

863 gli indici normalizzati di un generico nodo rappresentativo di \mathcal{G} , in cui

$$\mathcal{J} \equiv \left\{ \frac{i}{N} \mid i=0,1,\dots,N \right\} \subseteq [0,1]$$

864 è l'insieme discreto degli indici normalizzati. Allora la (3.22) normalizzata,
865 per analogia colla (3.34), si può definire

$$\hat{\mathbf{f}}(\hat{i}, s, t) = N \mathbf{f}(N \hat{i}, s, t) = \sum_{j \in \mathcal{J}} f_j(s, t) \otimes \delta(N \hat{i} - j),$$

ma per la condizione di normalità $\mathbb{P}([0,1] \times \mathbb{R}_+) = 1, \forall t \geq 0$, deve valere¹⁴

$$f_j(s, t) = N f_j(s, t) \quad \text{ove } j = N_i,$$

anch'essa analoga alla (3.34); difatti con questa la f si scrive

$$f(i, s, t) = \frac{1}{N} \sum_{j \in \mathcal{J}} f_j(s, t) \otimes \delta(N(i - j)) = \frac{1}{N} \sum_{j \in \mathcal{J}} f_j(s, t) \otimes \delta(i - j).$$

Con tali definizioni la densità marginale \bar{f} si può riformulare mediante il cambio di variabili $i = Ni$:

$$\bar{f}_N = \int_{\mathcal{J}} f(i, s, t) di = \int_{\mathcal{J}} N f(i, s, t) di = \int_{\mathcal{J}} f(i, s, t) di = \bar{f}_N = \frac{1}{N} \sum_{i \in \mathcal{J}} f_i(s, t),$$

e, definendo il passo $\Delta i \equiv 1/N$ tra due indici normalizzati consecutivi, si arriva al limite $N \rightarrow \infty$ a

$$\bar{f}_N = \bar{f}_N = \sum_{i \in \mathcal{J}} f_i(s, t) \Delta i \xrightarrow{N \rightarrow \infty} \bar{f} = \bar{f} = \int_0^1 f_i(s, t) di,$$

mentre la f diventa

$$f(i, s, t) = \int_0^1 f_j(s, t) \otimes \delta(i - j) dj = f_i(s, t).$$

Similmente vale per \bar{g} .

Osservazione 3.28 (Sulla bontà di $A \approx B$). Su un aspetto è al momento non ci si può esprimere: quanto bene la B approssima la matrice A in generale? Sarebbe necessario elaborare ulteriormente la teoria per trovare una risposta, obiettivo, come già detto, non di questa tesi. Tuttavia, euristicamente, si può riflettere in questo modo: con $N \gg 1$ le singole connessioni sono meno importanti, per cui vengono meno i dettagli, mentre la panoramica può essere ragionevolmente colta da B ; ciò suggerirebbe che più sono i nodi migliore è l'approssimazione $A \approx B$. Tale breve riflessione è però solo un intuito, *non* una dimostrazione rigorosa, e dunque anche potenzialmente falsa.

Osservazione 3.29. Si veda [17] per la dimostrazione del Teo. 3.1 in un contesto delle reti sociali e nel caso di un grafo diretto, non necessariamente simmetrico, oltre che di $[RI]_X$ lineari e simmetriche.

3.3.3 Altri nessi distinguibile-indistinguibile

Nel precedente paragrafo si è esplorato come si può "perdere" la struttura sottostante indotta dal grafo passando a una densità dipendente dal grado dei nodi, recuperando in tal modo un'equazione classica di tipo Boltzmann. Risulta perciò stimolante esplorare se sussitano certe ipotesi che permettano di trasformare la $[E\bar{g}]$ in una forma della $[EtB]_X$, approfondendo così i legami tra la teoria retale sviluppata nei due previ due paragrafi e la teoria di tipo Boltzmann.

¹⁴ Si ricordi che l'indice j non è un argomento della densità f_j , quindi può essere cambiato con un altro indice purché sia distinto da tutti gli altri secondo la trasformazione scelta.

894 **Ipotesi semplificative**

895 A questo scopo valgono le seguenti:

896 **Ipotesi 3.9.** Esistono tre principali ipotesi semplificative della [E \bar{g}]:

897 **S1** Il grafo è completamente connesso con matrice d'adiacenza unitaria:

$$\mathbf{A} \equiv \mathbf{1} \iff a_{i,j} \equiv 1 \quad \forall i,j \in \mathcal{J}.$$

898 **S2** Gli agenti sono indistinguibili rispetto al grafo:

$$f_i(s,t) = f_j(s,t) = f(s,t) \quad \forall i,j \in \mathcal{J},$$

899 **S3** Le regole d'interazione [RI] $_S$ sono simmetriche (Def. 3.11).

900 **Analisi della S1**

901 Notando che $\mathbf{A} = \mathbf{1} = \mathbf{1}\mathbf{1}^\top$, la [E \bar{g}] diventa

$$\begin{aligned} \frac{d}{dt} \int_{\mathbb{R}_+} \varphi \bar{f} ds &= \frac{1}{N^2} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} \mathbf{f}^\top \mathbf{1}\mathbf{1}^\top \mathbf{f}_* ds ds_* \\ &= \frac{1}{N^2} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} (\mathbf{1}^\top \mathbf{f})(\mathbf{1}^\top \mathbf{f}_*) ds ds_* \\ &= \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} \left(\frac{1}{N} \mathbf{1}^\top \mathbf{f} \right) \left(\frac{1}{N} \mathbf{1}^\top \mathbf{f}_* \right) ds ds_*, \end{aligned}$$

902 ma ricordando (3.30) si arriva a

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi \bar{f} ds = \int_{\mathbb{R}_+^2} \frac{\langle \varphi' - \varphi + \varphi'_* - \varphi_* \rangle}{2} \bar{f} \bar{f}_* ds ds_*, \quad (3.36)$$

903 la quale è analoga alla classica equazione di tipo Boltzmann [EtB] $_X$ con mi-
904 crostato $X_t \equiv S_t$ scalare, regole d'interazione [RI] $_S$ e nucleo d'interazione
905 unitario¹⁵ $\mu \equiv 1$ nella [Ber] $_X$. Ciò significa che con **S1** gli agenti, nonostan-
906 te siano distinti per il grafo, si possono vedere come indistinguibili purché
907 si consideri la distribuzione media (3.30).

908 **Analisi della S2**

909 Sotto tal'ipotesi, che equivale coll'assumere

$$\mathbf{f} = \mathbf{1}\mathbf{f} \quad \text{e} \quad \mathbf{f}_* = \mathbf{1}\mathbf{f}_*,$$

910 la [E \bar{g}] diventa

$$\begin{aligned} \frac{d}{dt} \int_{\mathbb{R}_+} \varphi \bar{f} ds &= \frac{1}{N^2} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} \mathbf{f} \mathbf{1}^\top \mathbf{A} \mathbf{1} \mathbf{f}_* ds ds_* \\ &= \frac{\mathbf{1}^\top \mathbf{A} \mathbf{1}}{N^2} \int_{\mathbb{R}_+^2} \frac{\langle \varphi' + \varphi'_* - \varphi - \varphi_* \rangle}{2} \mathbf{f} \mathbf{f}_* ds ds_* \end{aligned}$$

¹⁵Tale risultato implica anche che un nucleo di collisione unitario nella [EtB] $_X$ è il corrispettivo di una matrice unitaria nella [EtB] $_S$.

e rimembrando (3.28) si ottiene

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi f ds = \int_{\mathbb{R}_+^2} \frac{D_N}{N^2} \frac{\langle \varphi' - \varphi + \varphi'_* - \varphi_* \rangle}{2} ff^* ds ds_*.$$

In questo contesto il rapporto $D_N/N^2 \in [0, 1]$ (grado medio normalizzato della rete) rappresenta quanto la rete è topologicamente simile a una completamente connessa¹⁶.

Per il resto l'equazione è analoga alla classica equazione di tipo Boltzmann [EtB]_X con microstato $X_t \equiv S_t$ scalare, regole d'interazione [RI]_S e nucleo d'interazione nella [Ber]_X costante $\mu \equiv D_N/N^2 = \bar{D}_N$.

Dunque l'indistinguibilità degli agenti ha una notevole conseguenza sulla [Eg]: riassume l'effetto complessivo del grafo al solo coefficiente \bar{D}_N il quale, dunque, ne rappresenta gli ultimi bagliori prima di una sua totale scomparsa per la S1.

Analisi della S1, S2 e S3

Visto che vale la S1 si può partire dalla (3.36) nella quale la densità media (3.30) diventa per S2

$$\bar{f}(s, t) = \frac{1}{N} \sum_{i \in \mathcal{I}} f_i(s, t) \stackrel{S2}{=} \frac{1}{N} \sum_{i \in \mathcal{I}} f(s, t) = f(s, t),$$

ossia la \bar{f} coincide con quella di tutti gli agenti¹⁷, essendo questi, appunto, indistinguibili.

In tal modo la (3.36) diventa

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi f ds = \int_{\mathbb{R}_+^2} \frac{\langle \varphi' - \varphi + \varphi'_* - \varphi_* \rangle}{2} ff^* ds ds_*,$$

che unita all'S3 porta all'equivalenza

$$\int_{\mathbb{R}_+^2} \langle \varphi'_* - \varphi_* \rangle ff_* ds ds_* = \int_{\mathbb{R}_+^2} \langle \varphi' - \varphi \rangle ff^* ds ds_*,$$

dalla quale si ha

$$\frac{d}{dt} \int_{\mathbb{R}_+} \varphi f ds = \int_{\mathbb{R}_+^2} \langle \varphi' - \varphi \rangle ff^* ds ds_*.$$

Quest'equazione, come per gli altri casi, è analoga alla classica equazione di tipo Boltzmann simmetrica (3.19) con microstato $X_t \equiv S_t$ scalare, regole d'interazione [RI]_S e nucleo d'interazione nella [Ber]_X unitario $\mu \equiv 1$.

¹⁶ Difatti D_N è interpretabile come il numero di lati presenti in un grafo diretto e che ha come limite superiore proprio N^2 , ossia il numero totale di coppie [e quindi lati] dati N nodi.

¹⁷ Si noti anche che la perdita della dipendenza della distribuzione media dal numero di nodi N è coerente col limite $N \rightarrow \infty$, sempre ben approssimabile nei casi classici con un numero di agenti $N \gg 1$ [come lo studio di un gas].

4

SIMULAZIONI

In questo capitolo si applica tutta la teoria affrontata in quello precedente. Innanzitutto si descrive l'algoritmo con cui sono stati svolte le simulazioni; quindi si spiegano le formule che definiscono i grafici per analizzare i risultati per poi motivare rapidamente il perché le fluttuazioni possono [anzi devono] essere trascurate. Per le simulazioni si propongono varie leggi d'emigrazione che studiate tramite dei studi parametrici, cercando successivamente d'interpretare cosa queste dicono sul fenomeno della migrazione in sé. Infine si mostra brevemente una simulazione per l'Italia, confermando che i risultati migliorano col numero d'agenti.

4.1 METODO MONTE CARLO

Nel contesto delle TCSMA l'obbiettivo, com'è chiaro dal Cap. 3, è quello di ricavare la densità $f(\mathbf{x}, t)$ nella $[EtB]_X$ al variare del tempo. Si potrebbe allora pensare di discretizzare quest'ultima equazione mediante il Metodo delle Differenze Finite, degli Elementi Finiti o dei Volumi Finiti; tuttavia, vi sono due principali problemi:

1. l'impossibilità, in generale, di ricavare la forma forte della $[EtB]_X$, specie nel caso di $[RI]_X$ non lineari;
2. anche ipotizzando di trovare la forma forte a cui applicare i precedenti metodi, la sua natura integro differenziale la rende complessa da manipolare dato che l'operatore collisionale¹, come

$$Q(f, f)(\mathbf{v}, t) \equiv \frac{1}{4\pi} \int_{\mathbb{R}^3} \int_{S^2} B(\mathbf{v}, \mathbf{v}_*, \mathbf{n}) (f(\mathbf{v}', t)f(\mathbf{v}'_*, t) - f(\mathbf{v}, t)f(\mathbf{v}_*, t)) d\mathbf{n} d\mathbf{v}_*$$

nella (3.15), dipende dalla densità medesima.

Per tali ragioni si procede in maniera più semplice: conoscendo $[AR]_X$, che governa le interazioni binarie tra agenti, si possono quindi direttamente simulare tutte le molteplici collisioni mediante un metodo di Monte Carlo di tipo Nanbu-Babovsky, descritto nel dettaglio nell'Alg. 1 nel quale $T > 0$ è il tempo finale di simulazione, mentre P è la popolazione totale.

Osservazione 4.1. Nella linea 18, \bar{f} è una densità arbitraria: va intesa come \bar{f} se l'approccio è esatto e come \bar{g} se è approssimato.

Osservazione 4.2. Più in generale, nella linea 1, \mathcal{S}_0 si può campionare da una densità $\bar{\mathcal{S}}_0$ iniziale; tuttavia non conoscendone alcuna per la distribuzione della popolazione, si è deciso di definire \mathcal{S}^0 come un vettore uniforme rispetto a una popolazione totale P iniziale.

¹ Vale a dire la parte integrale della forma forte dell'equazione di tipo Boltzmann, in genere scritta a secondo membro.

Algoritmo 1: Algoritmo [AR]_S di tipo Nanbu-Babovsky

```

Dati:  $N \in \mathbb{N}_+$ ,  $\Delta t \leq 1$ ,  $\sigma$ ,  $T > 0$ ,  $P$ ,  $\mathbf{A}$  e  $\mathbf{B}$  ;
1  $\mathcal{S}^0 \leftarrow (s_1^0, s_2^0, \dots, s_N^0) \equiv (P/N)\mathbf{1} \in \mathbb{R}_+^N$ ;
2 per  $n = 0, 1, 2, \dots, \lfloor T/\Delta t \rfloor - 1$  fai
3    $P \leftarrow$  permutazione indipendente di  $\{1, 2, \dots, N\}$ ;
4   per  $i = 1, 2, \dots, \lfloor N/2 \rfloor$  fai
5      $j \leftarrow \lfloor N/2 \rfloor + i$ ,  $s_i^n \leftarrow P(i)$  e  $s_j^n \leftarrow P(j)$ ;
6     se esatto allora
7        $\Theta \leftarrow \text{Bernoulli}(A(i,j)(\Delta t))$ ;
8     altrimenti # è approssimato
9        $\Theta \leftarrow \text{Bernoulli}(B(i,j)(\Delta t))$ ;
10    se  $\Theta = 1$  allora
11       $E \leftarrow E(s_i^n, i, s_j^n, j)$ ;
12       $\gamma \leftarrow \text{Gamma}((1-E)^2/\sigma^2, \sigma^2/(1-E))$ ;
13       $s_i^{n+1} \leftarrow s_i^n(1-E+\gamma)$ ; # città interangente
14       $s_j^{n+1} \leftarrow s_j^n + s_i^n E$ ; # città ricevente
15    altrimenti
16       $s_i^{n+1} \leftarrow s_i^n$  e  $s_j^{n+1} \leftarrow s_j^n$ ;
17     $\mathcal{S}^{n+1} \leftarrow (s_1^{n+1}, s_2^{n+1}, \dots, s_N^{n+1})$ ;
18     $\bar{l}(s, (n+1)\Delta t) \leftarrow$  istogramma di  $\mathcal{S}^{n+1}$ ;

```

966 4.2 RAPPRESENTAZIONE DEI RISULTATI

967 Per analizzare i risultati delle simulazioni è opportuno descrivere nel dettaglio le funzioni usate per descriverli; per farlo, però, si devono prima approfondire le conseguenze della natura stocastica dell'Alg. 1 e la struttura dei risultati. Verso la fine, si motiva anche l'omissione delle fluttuazioni γ .

971 4.2.1 Intervalli di confidenza

972 Innanzitutto simulando la [EtB]_S mediante Alg. 1, e quindi l'algoritmo [AR]_X,
973 si sta introducendo nei risultati un rumore di natura stocastica: più simulazioni
974 daranno risultati diversi per cui la singola non ha rilevanza statistica;
975 bisogna cioè calcolarne molteplici e valutare gli intervalli di confidenza per
976 conoscere l'incertezza sulla stima della media.

977 Sia $R \in \mathbb{N}_+$ il numero di simulazioni eseguite e $\mathbf{S} \in \mathbb{R}_+^R$ il vettore aleatorio
978 della popolazione di una città relativa a ciascuna simulazione.

979 **Ipotesi 4.1** (Numero di simulazioni). In questo elaborato si assume $R = 100$
980 per avere un numero statisticamente significativo di dati.

981 Per l'Alg. 1 le componenti (S_1, S_2, \dots, S_R) di \mathbf{S} sono indipendenti e identi-
982 camente distribuite dalla densità q , proprietà che permette di definire

$$\bar{S} \equiv \frac{1}{R} \sum_{r=1}^R S_r \quad \text{e} \quad V \equiv \frac{1}{R-1} \sum_{r=1}^R (S_r - \bar{S})^2,$$

983 rispettivamente la media e la varianza campionarie. Allora, data μ la *vera*²
 984 media della densità l , la distribuzione T di Student con parametro $R-1$ si
 985 scrive

$$T = \frac{\bar{S} - \mu}{\sqrt{V/R}} \sim \text{Student}(R-1).$$

986 Da questa si può definire l'intervallo di confidenza [simmetrico] con livello
 987 di confidenza α ponendo

$$\mathbb{P}\left(-t_{R-1}^{\alpha/2} \leq T \leq t_{R-1}^{\alpha/2}\right) = 1 - \alpha \quad (4.1)$$

988 ove $t_{R-1}^{\alpha/2} \in \mathbb{R}$ è quel valore reale tale che

$$\mathbb{P}\left(T < t_{R-1}^{\alpha/2}\right) = 1 - \frac{\alpha}{2}.$$

989 Esplicitando la T nella (4.1) e isolando la media μ si ha

$$\mathbb{P}\left(\bar{S} - t_{R-1}^{\alpha/2} \sqrt{\frac{V}{R}} \leq \mu \leq \bar{S} + t_{R-1}^{\alpha/2} \sqrt{\frac{V}{R}}\right) = 1 - \alpha,$$

990 da cui

$$IC_R^\alpha(\mathbf{S}) \equiv \left[\bar{S} - t_{R-1}^{\alpha/2} \sqrt{\frac{V}{R}}, \bar{S} + t_{R-1}^{\alpha/2} \sqrt{\frac{V}{R}} \right]$$

991 è la stima intervallare che definisce l'intervallo di confidenza ricercato, espri-
 992 mibile anche più compattamente come

$$IC_R^\alpha(\mathbf{S}) \equiv \bar{S} \pm t_{R-1}^{\alpha/2} \sqrt{V/R}. \quad (4.2)$$

993 **Ipotesi 4.2** (livello di confidenza). Si sceglie $\alpha = 0.05$ così d'avere un interval-
 994 lo con livello di confidenza 0.95.

995 Il significato dell'intervallo di confidenza in essenza è l'errore statistico
 996 commesso: esso valuta quanto è probabile che la stima intervallare (4.2) con-
 997 tenga il parametro μ ; in altre parole IC_R^α misura l'incertezza sulla stima della
 998 media: preso un campione s , più l'intervallo di confidenza è esteso più la
 999 media campionaria \bar{s} è una stima incerta dell'effettiva media μ ; viceversa più
 1000 è stretto, più la stima \bar{s} è precisa nel senso che μ si trova in un intorno picco-
 1001 lo della media campionaria. Sotto questo punto di vista è concettualmente
 1002 analogo alla precisione di uno strumento di misura.

1003 **Osservazione 4.3.** La stima intervallare (4.2) appena ricavata vale tanto per
 1004 il vettore aleatorio \mathbf{S} che per le sue realizzazioni s , le quali sono, appunto, il
 1005 risultato delle R simulazioni.

1006 4.2.2 Struttura dei dati

1007 I dati presentano una struttura di un tensore del quart'ordine $s \in \mathbb{R}_+^{N_f \times 2 \times N \times R}$
 1008 i cui indici hanno il seguente significato

$$s_{i,r}^{n,a} \begin{cases} n = \text{istante temporale}, & a = \text{tipo di simulazione}, \\ i = \text{indice della città}, & r = \text{numero della simulazione}. \end{cases}$$

1009 L'istante temporale è definito tramite tre parametri in \mathbb{N}_+ :

² Vale a dire μ non è una variabile aleatoria ma l'esatto parametro della media di l .

- 1010 ◊ $N_t \equiv \lfloor T/\Delta t \rfloor$ è il numero totale di tempi simulati;
 1011 ◊ $N_s \ll N_t$ è il numero di catture dai tempi simulati;
 1012 ◊ $N_f < N_s$ è il numero di tempi ridotti dalle catture.

1013 Sia le catture che la riduzione sono campinate in intervalli equispaziati
 1014 con passi

$$\Delta s = N_t / N_s \quad \text{e} \quad \Delta f = N_s / N_f$$

1015 ove si suppone, per semplicità, che N_s e N_f siano divisori ordinatamente di
 1016 N_t e N_s , da cui

$$N_t / N_s - \lfloor N_t / N_s \rfloor = 0 \quad \text{e} \quad N_s / N_f - \lfloor N_s / N_f \rfloor = 0,$$

1017 e ugualmente si assume per Δt e T .

1018 Tramite l'Alg. 1 si simulano in totale N_t tempi con passo Δt di cui N_s
 1019 sono salvati nel tensore $\mathbf{s} \in \mathbb{R}_+^{N_s \times 2 \times N \times R}$ delle catture:

$$\mathbf{s}_{\cdot, r}^{n, a} \equiv \mathcal{S}_{\cdot, r}^{n \Delta s, a} \quad \forall n, \forall a, \forall r,$$

1020 dove $\mathcal{S}_{\cdot, r}^{n \Delta s, a}$ va intesa come il vettore delle popolazioni predette nell' r -esima
 1021 simulazione esatta, se $a=1$, o approssimata, se $a=2$; successivamente si
 1022 convolve \mathbf{s} rispetto all'indice del tempo:

$$s_{i, r}^{n, a} \equiv \frac{1}{N_f} \sum_{j=1}^{N_f} s_i^{(n-1)N_f + j, a} r, \quad \forall n, \forall a, \forall i, \forall r,$$

1023 vale a dire si mediano ogni Δf elementi delle N_s catture. Tale mollificazione
 1024 è necessaria per rendere \mathbf{s} meno rumoroso rispetto a \mathbf{s} e quindi più leggibile
 1025 una volta raffigurato.

1026 Per quanto riguarda gl'istanti temporali considerati, si hanno tre forme a
 1027 seconda di come s'intende l'indice n :

$$\begin{aligned} t_n &= n \Delta t, & \forall n \in \{1, 2, \dots, N_t\}, \\ t_n^s &= t_n \Delta s, & \forall n \in \{1, 2, \dots, N_s\}, \\ t_n^f &= t_n^s \Delta f, & \forall n \in \{1, 2, \dots, N_f\}, \end{aligned}$$

1028 rispettivamente per i tempi discretizzati, campionanti e ridotti; a prescindere
 1029 vale comunque $t_{N_t} = t_{N_s}^s = t_{N_f}^f = T$.

1030 **Osservazione 4.4.** I parametri temporali N_t , N_s ed N_f non contano il tempo
 1031 iniziale perché fa riferimento alla distribuzione iniziale $\mathbf{s}_0 \in \mathbb{R}_+^N$.

1032 **Ipotesi 4.3.** In questa trattazione si considerano $\Delta t = 0.01$, $N_s = 1000$, $N_f = 50$
 1033 mentre N_t viene scelto per essere poco superiore al tempo di convergenza
 1034 della simulazione, se i risultati convergono, ma sicuramente è un multiplo
 1035 di 10 per garantire che N_s sia un suo divisore.

1036 **Osservazione 4.5.** Per la maggior parte dei grafici si considera esclusiva-
 1037 mente la distribuzione al tempo finale T senza convoluzione, cosicché, per
 1038 leggerezza di notazione, si può impropriamente denotare con $\mathbf{s}^T \in \mathbb{R}_+^{2 \times N \times R}$
 1039 il tensore del terz'ordine tale che $\mathbf{s}^T \equiv \mathbf{s}^{N_s}$.

1040 **4.2.3 Definizione dei grafici**

1041 **Istogrammi**

1042 I primi grafici considerano gl'istogrammi [normalizzati] della distribuzione
1043 al tempo finale T .

1044 Si inizia considerando il massimo e il minimo elemento del tensore \mathbf{s}^T ,

$$s_{\max} \equiv \max_{a,i,r} s_{i,r}^{T,a} \quad \text{e} \quad s_{\min} \equiv \min_{a,i,r} s_{i,r}^{T,a},$$

1045 coi quali si può definire una griglia comune equispaziata su cui costruire
1046 gl'istogrammi. Siano N_c il numero di intervalli della griglia, ossia il
1047 numero di classi, allora definita Istogramma: $\mathbb{R}^N \rightarrow \mathbb{R}^{N_c}$ come la funzione
1048 che restituisce i valori [normalizzati] delle classi dell'istogramma, il tensore
1049 $\mathbf{h} \in \mathbb{R}_+^{2 \times N_c \times R}$ si scrive

$$\mathbf{h}_{\cdot,r}^a \equiv \text{Istogramma}(\mathbf{s}_{\cdot,r}^{T,a}), \quad \forall a \text{ e } \forall r,$$

1050 Valutati gl'intervalli di confidenza

$$IC_R^{0.05}(\mathbf{h}_c^a) = \bar{h}_c^a \pm t_{R-1}^{0.025} \sqrt{\frac{v_c^a}{R}}, \quad \forall a \text{ e } \forall c,$$

1051 si possono rappresentare con \bar{h}^a l'istogramma medio della simulazione esatta
1052 e approssimata, e con $\pm t_{R-1}^{0.025} \sqrt{v_c^a/R}$ l'errore stocastico sulle stime dei
1053 valori medi delle classi.

1054 **Lognormale bimodale**

1055 Per quanto riguarda i fittaggi lognormali bimodali la logica è simile agl'isto-
1056 grammi: sia

$$\mathcal{L}(x; \mathbf{s}_{\cdot,r}^{T,a}) : \mathbb{R}_+ \rightarrow \mathbb{R}_+, \quad \forall a \text{ e } \forall r,$$

1057 la densità di una distribuzione lognormale bimodale, con densità (1.4), fittata
1058 dal vettore $\mathbf{s}_{\cdot,r}^{T,a}$ e sia

$$\mathcal{L}(x; \mathbf{s}^{T,a}) : \mathbb{R}_+ \rightarrow \mathbb{R}_+^R$$

1059 la funzione vettoriale tale che

$$\mathcal{L}(x; \mathbf{s}^{T,a}) \equiv [\mathcal{L}(x; \mathbf{s}_{\cdot,1}^{T,a}) \quad \mathcal{L}(x; \mathbf{s}_{\cdot,2}^{T,a}) \quad \dots \quad \mathcal{L}(x; \mathbf{s}_{\cdot,R}^{T,a})]^T,$$

1060 che in essenza raccoglie puntualmente tutti i fittaggi in un unico vettore.
1061 Allora gl'intervalli di confidenza hanno forma

$$IC_R^{0.05}(\mathcal{L}(x; \mathbf{s}^{T,a})) = \bar{\mathcal{L}}^a(x) \pm t_{R-1}^{0.025} \sqrt{\frac{v^a(x)}{R}}, \quad \forall x \in [s_{\min}, s_{\max}] \text{ e } \forall a,$$

1062 in cui, come prima, $\bar{\mathcal{L}}^a(x)$ è la funzione media mentre $\pm t_{R-1}^{0.025} \sqrt{v^a(x)/R}$ la
1063 sua incertezza stocastica.

1064 **Osservazione 4.6.** Essendo x continuo, l'insieme degl'intervalli di confiden-
1065 za forma per le funzioni un fascio di confidenza.

1066 Si nota, in conclusione, che i tre grafici sono in scala semilogaritmica per
1067 far emergere la distribuzione normale dalla lognormale bimodale.

1068 **Funzione di ripartizione empirica**

1069 La distribuzione di Pareto è fittata a partire dalla funzione di ripartizione
 1070 complementare (FRC) empirica che quindi va prima analizzata, ma solo
 1071 nella coda della distribuzione della popolazione.

1072 Il dominio in questo caso viene definito mediante l'ultimo quartile, ossia
 1073 quel valore $s_{3/4}$ tale che $\mathbb{P}(S \leq s_{3/4}) = 3/4$, data una qualunque variabile alea-
 1074 toria S ; ciò corrisponde nella pratica a trovare quell'elemento di $\mathbf{s}_{\cdot,r}^{T,a}$ tale
 1075 che

$$\left| \left\{ s_{i,r}^{T,a} \geq s_{3/4} \mid i \in \{1, 2, \dots, N\} \right\} \right| = \left\lceil \frac{N}{4} \right\rceil, \quad \forall a \text{ e } \forall r,$$

1076 che a parole vuol dire che *almeno* $1/4$ di tutti gli elementi di $\mathbf{s}_{\cdot,r}^{T,a}$ sono
 1077 superiori a $s_{3/4}$. Allora l'intervallo $[s_{3/4}, s_{\max}]$ caratterizza la coda.

1078 La FRC empirica, che approssima la FRC $\mathbb{P}(S > x)$, si scrive

$$\mathcal{F}(x; \mathbf{s}_{\cdot,r}^a) \equiv 1 - \frac{1}{N} \sum_{i=1}^N \chi_{[0,x]}(s_{i,r}^{T,a}), \quad \forall a \text{ e } \forall r, \quad (4.3)$$

1079 dove $\chi_{[0,x]}: \mathbb{R} \rightarrow \{0,1\}$ è la funzione indicatrice dell'insieme $[0,x]$:

$$\chi_{[0,x]}(w) \equiv \begin{cases} 1 & \text{se } w \in [0,x], \\ 0 & \text{altrimenti.} \end{cases}$$

1080 Tuttavia, siccome la (4.3) dev'essere valutata in $x \in [s_{3/4}, s_{\max}]$, e in particolare
 1081 nel tensore $\mathbf{p} \in \mathbb{R}^{2 \times N_{1/4} \times R}$ di \mathbf{s} ristretto alla coda

$$p_{i,r}^a \equiv s_{N-N_{1/4}+i,r}^{T,a} \quad \forall a, \forall i \in \{1, 2, \dots, N_{1/4}\} \text{ e } \forall r$$

1082 nel quale $N_{1/4} \equiv \lceil N/4 \rceil$, vale il seguente riscalamento:

$$\begin{aligned} \mathcal{F}(x; \mathbf{s}_{\cdot,r}^{T,a}) &= 1 - \frac{N - N_{1/4}}{N} - \frac{1}{N} \sum_{i=1}^{N_{1/4}} \chi_{[0,x]}(s_{N-N_{1/4}+i,r}^{T,a}) \\ &= \frac{N_{1/4}}{N} \left[1 - \frac{1}{N_{1/4}} \sum_{i=1}^{N_{1/4}} \chi_{[0,x]}(p_{i,r}^{T,a}) \right] = \frac{N_{1/4}}{N} \mathcal{F}(x; \mathbf{p}_{\cdot,r}^a), \quad \forall a \text{ e } \forall r, \end{aligned}$$

1083 Vale a dire che $\mathcal{F}(x; \mathbf{p}_{\cdot,r}^a)$ è uguale a $\mathcal{F}(x; \mathbf{s}_{\cdot,r}^{T,a})$ a meno di un riscalamento
 1084 $N_{1/4}/N$. Tuttavia l'indice di Pareto è invariante rispetto ai riscalamenti:

1085 **Osservazione 4.7.** In scala logaritmica la (1.2) si scrive

$$\log(R) = \log(c) - \beta \log(s),$$

1086 che equivale a una retta traslata di $\log(c)$; indicando con R' il rapporto R/c
 1087 segue

$$\log(R') \equiv \log(R) - \log(c) = -\beta \log(s),$$

1088 per la quale il coefficiente di Pareto β è invariato.

1089 Da quest'osservazione si può pertanto scegliere $\mathcal{F}(x; \mathbf{s}_{\cdot,r}^{T,a})$ in scala logarit-
 1090 mica per una maggiore leggibilità dei valori i quali altrimenti, con $\mathcal{F}(x; \mathbf{p}_{\cdot,r}^a)$,

1091 sarebbero bassi. L'uso di tale scala introduce però un lieve problema: l'ultimo
 1092 elemento $p_{N_{1/4},r}^a$ ha ordinata nulla ed è dunque non visibile; per ovviare
 1093 il problema si può traslare la FRC empirica in scala lineare

$$\mathcal{F}(x; \mathbf{p}_{\cdot,r}^a) + \frac{1}{2N_{1/4}}, \quad \forall a \text{ e } \forall r, \quad (4.4)$$

1094 ciò corrisponde in scala logaritmica a una traslazione non uniforme dei valori
 1095 che quindi vengono falsati, seppure di poco essendo $1/2N_{1/4}$ una quantità
 1096 piccola; per tale ragione tale modifica è valida *solo graficamente* e non per il
 1097 calcolo dell'indice di Pareto.

1098 Di conseguenza, nel complesso, la FRC empirica è raffigurata in scala
 1099 logaritmica dal diagramma a dispersione delle coppie

$$\left(\frac{N}{N_{1/4}} \mathcal{F}(\bar{p}_i^a; \mathbf{s}_{\cdot,r}^{T,a}) + \frac{1}{2N_{1/4}}, \bar{p}_i^a \right) = \left(\mathcal{F}(\bar{p}_i^a; \mathbf{p}_{\cdot,r}^a) + \frac{1}{2N_{1/4}}, \bar{p}_i^a \right) \quad \forall a \text{ e } \forall i,$$

1100 ove \bar{p}_i^a viene dagl'intervalli di confidenza

$$IC_R^{0.05}(\bar{p}_i^a) = \bar{p}_i^a \pm t_{R-1}^{0.025} \sqrt{\frac{v_i^a}{R}}, \quad \forall a \text{ e } \forall i.$$

1101 **Osservazione 4.8.** Si noti che, contrariamente ai casi precedenti, l'intervallo
 1102 di confidenza è orizzontale e non verticale; difatti l'immagine della FRC em-
 1103 pirica è invariata rispetto alle simulazioni che modificano solo le popolazioni
 1104 dei centri maggiori, ossia l'ascissa della FRC empirica.

1105 *Pareto e relativo indice*

1106 Per quanto riguarda il fintaggio della distribuzione di Pareto il ragionamento
 1107 è analogo a quello visto per la lognormale bimodale: sia

$$\mathcal{P}(x; \mathbf{p}_{\cdot,r}^a): \mathbb{R}_+ \rightarrow \mathbb{R}_+, \quad \forall a \text{ e } \forall r,$$

1108 la densità di distribuzione di Pareto, con FRC (1.2), fissata dal vettore $\mathbf{p}_{\cdot,r}^a$ e
 1109 sia

$$\mathcal{P}(x; \mathbf{p}^a): \mathbb{R}_+ \rightarrow \mathbb{R}_+^R$$

1110 la funzione vettoriale tale che

$$\mathcal{P}(x; \mathbf{p}^a) \equiv \begin{bmatrix} \mathcal{P}(x; \mathbf{p}_{\cdot,1}^a) & \mathcal{P}(x; \mathbf{p}_{\cdot,2}^a) & \dots & \mathcal{P}(x; \mathbf{p}_{\cdot,R}^a) \end{bmatrix}^\top.$$

1111 Allora gl'intervalli di confidenza hanno forma

$$IC_R^{0.05} \left(\frac{N}{N_{1/4}} \mathcal{P}(x; \mathbf{p}^a) \right) = \bar{\mathcal{P}}^a(x) \pm t_{R-1}^{0.025} \sqrt{\frac{v^a(x)}{R}}, \quad \forall x \in [s_{3/4}, s_{\max}] \text{ e } \forall a,$$

1112 ove $N/N_{1/4}$ è il fattore di riscalamento descritto nel § 4.2.3 necessario per
 1113 confrontare l'adattamento colla FRC empirica.

1114 **Osservazione 4.9.** È fissando i dati \mathbf{p} alla distribuzione di Pareto che si ricava
 1115 il relativo indice β , non dalla FRC empirica: ecco perché, essendo puramente
 1116 grafiche, le modifiche nella (4.4) sono tollerabili.

Evoluzione della taglia media

Si esprima con $\bar{s} \in \mathbb{R}_+^{N_f \times 2 \times R}$ il tensore delle taglie medie rispetto ai nodi

$$\bar{s}_r^{n,a} \equiv \frac{1}{N} \sum_{i=1}^N s_{i,r}^{n,a}, \quad \forall n, \forall a \text{ e } \forall r,$$

allora gl'intervallo di confidenza di \bar{s} sono

$$IC_R^{0.05}(\bar{s}^{n,a}) = \bar{s}^{n,a} \pm t_{R-1}^{0.025} \sqrt{\frac{v^{n,a}}{R}}, \quad \forall n \text{ e } \forall a.$$

L'andamento della taglia media totale è raffigurato interpolando linearmente le coppie

$$(t_n^f, \bar{s}^{n,a}), \quad \forall n \text{ e } \forall a,$$

a istanti temporali contingui; lo stesso vale per gli estremi degl'intervalli di confidenza $\pm t_{R-1}^{0.025} \sqrt{v^{n,a}/R}$, che quindi diventano un fascio di confidenza.

Taglie medie vs gradi

S'indichi ora con $\bar{s} \in \mathbb{R}_+^{2 \times R}$ il tensore delle taglie medie rispetto alle simulazioni

$$\bar{s}_i^a \equiv \frac{1}{R} \sum_{r=1}^R s_{i,r}^{T,a}, \quad \forall a \text{ e } \forall i,$$

allora dalle coppie

$$(k_i, \bar{s}_i^a), \quad \forall a \text{ e } \forall i,$$

si può definire il grafico a dispersione che lega la taglia media tra tutte le simulazioni di un nodo al suo grado.

Osservazione 4.10. In questo caso non si disegnano gl'intervalli di confidenza erché il grafico è già molto denso e aggiungere ulteriori barre verticali/orizzontali lo renderebbe difficilmente leggibile.

Evoluzioni delle taglie medie delle classi dei gradi

Sia $\mathcal{K} \subset \{1, 2, \dots, N\}$ l'insieme dei gradi distinti di k , si denoti con $\bar{s} \in \mathbb{R}^{N_f \times 2 \times |\mathcal{K}|}$ il tensore

$$\bar{s}_k^{n,a} \equiv \frac{1}{|\mathcal{J}_k|} \sum_{i \in \mathcal{J}_k} \frac{1}{R} \sum_{r=1}^R s_{i,r}^{n,a}, \quad \forall n, \forall a \text{ e } \forall k,$$

ove \mathcal{J}_k viene dalla (3.32) e rappresenta l'insieme degl'indici con grado k . Pertanto interpolando linearmente le coppie

$$(t_n^f, \bar{s}_k^{n,a}), \quad \forall n, \forall a \text{ e } \forall k,$$

a istanti temporali contigui, si può rappresentare l'andamento temporale della popolazione media della classe k -esima tra tutte le simulazioni.

Si conclude notando che non si disegnano gl'intervalli di confidenza per la medesima ragione chiarita nell'Oss. 4.10.

1142 **4.2.4 Sulle fluttuazioni γ**

1143 Nelle [RE] è stato introdotto per completezza il termine γ legato alle fluttuazioni,
 1144 ossia a quei fenomeni di morte e nascita che caratterizzano la naturale
 1145 variazione di una popolazione intorno a una media. Si argomenta adesso
 1146 che è possibile trascurarle per due ragioni:

- 1147 1. si è interessati solo alla predizione della distribuzione stazionaria da
 confrontare con quella reale del 1991 e
- 1149 2. dal vincolo (3.26) le uniche perturbazioni simmetriche sono per loro
 natura piccole, essendo concentrate in un intorno dell'origine.

1151 In effetti la Fig. 4.1 conferma che γ introduce solo una lieve incertezza sui
 1152 risultati, simulati coi parametri nella Tab. 4.1 e colla regola d'interazione

$$E(s, s_*, i, i_*) \equiv (1 - \zeta)\lambda \frac{\frac{(s_*/s)(w_{i_*}/w_i)}{\alpha}}{1 + \frac{(s_*/s)(w_{i_*}/w_i)}{\alpha}} + \zeta\lambda \frac{\frac{(s/s_*)(w_i/w_{i_*})}{\alpha}}{1 + \frac{(s/s_*)(w_i/w_{i_*})}{\alpha}}, \quad (4.5)$$

1153 che è spiegata implicitamente nel corso del § 4.3 ed è brevemente discussa
 1154 più nel dettaglio nel § 4.4.1.

	λ	α	ζ	σ	N_t
Senza γ	0.1	1	0.1	–	10^6
Con γ	0.1	1	0.1	0.05	10^6

Tabella 4.1: Parametri della Fig. 4.1.

1155 Dunque γ introduce solo del lieve rumore che, sebbene abbia il nobile
 1156 scopo di rendere più realistico il modello, non aiuta a interpretare i risultati
 1157 rispetto a quelli reali.

1158 **4.3 REGOLE D'EMIGRAZIONE**

1159 In questo paragrafo si specializza la regola d'emigrazione E definita nelle
 1160 [RE] per completare le [RI]_s; tutte le forme qui presenti condividono comun-
 1161 que tre aspetti: sono simmetriche, non lineari e associano i microstati (i, s) e
 1162 (i_*, s_*) rispettivamente alla città interagente e ricevente.

1163 Per brevità si limita lo studio a venire su tre aspetti:

- 1164 1. si considera solo la regione della Sardegna sia perché molti risultati
 di [6] sono già stati riprodotti nel § 2.3, sia poiché per le altre 19
 1166 regioni sono stati trovati risultati analoghi;
- 1167 2. si analizza esclusivamente la configurazione che meglio fitta i dati
 1168 reali, almeno tra quelle esplorate dall'autore, e
- 1169 3. si approfondiscono, vista la loro lunghezza, due soli studi parame-
 trici per l'ultima regola d'emigrazione proposta.

1171 Pertanto, data l'importanza della Sardegna, si elencano nella Tab. 4.2 i dati
 1172 principali della regione assieme ai parametri adattati dell'indice di Pareto e
 1173 della lognormale bimodale.

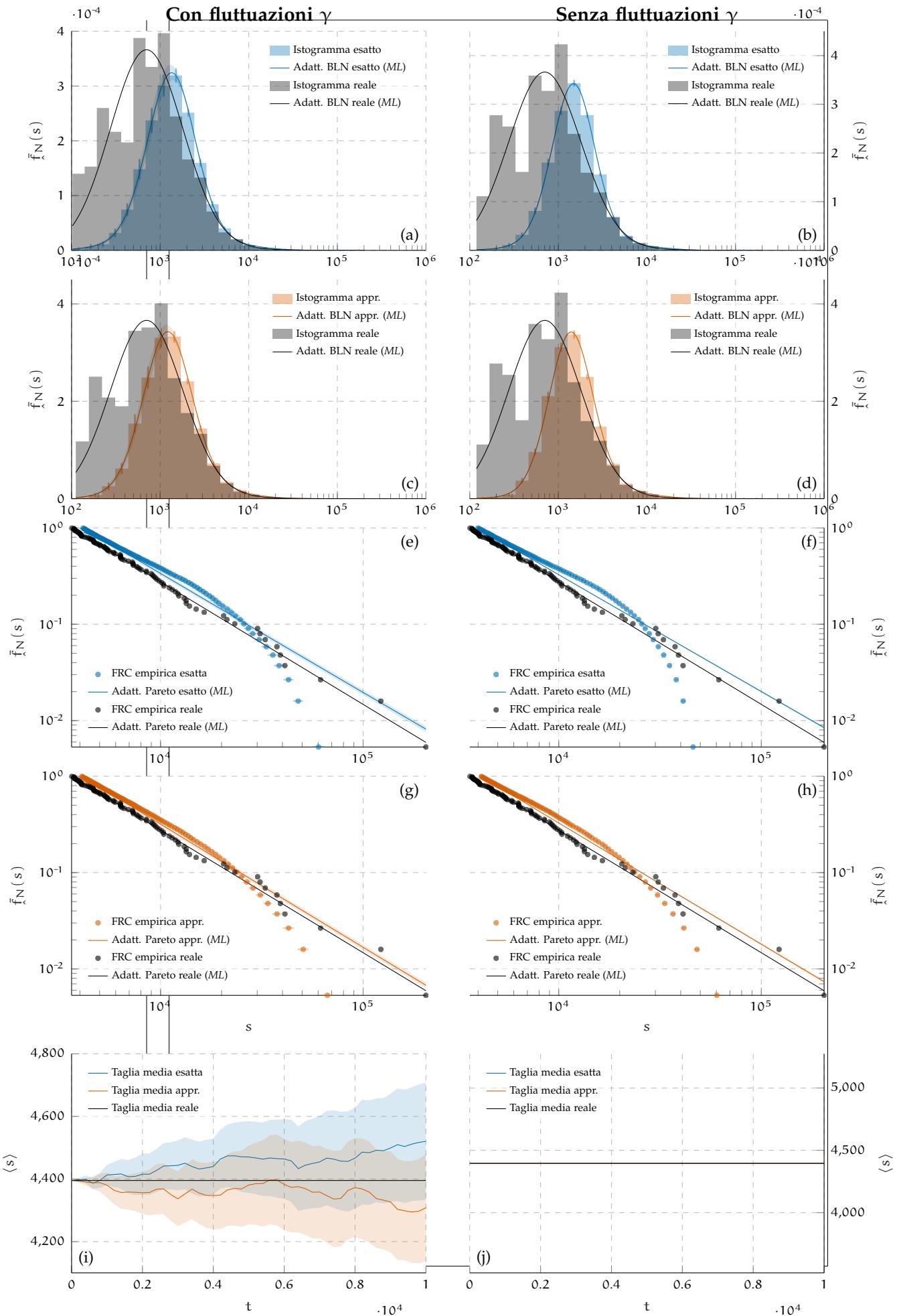


Figura 4.1: Confronto di una simulazione con e senza fluttuazioni; la regola d'immigrazione è la (4.5) mentre i parametri sono illustrati nella Tab. 4.1.

N	P	β	ξ
375	$\approx 1.65 \times 10^6$	1.27	0.93

Tabella 4.2: Dati e parametri della Sardegna.

Ciononostante, nei §§ 4.4.2 e 4.4.3 si studiano comunque la Valle d'Aosta e l'Italia essendo casi estremi rispetto al numero di nodi.

4.3.1 Regola taglia

Una prima possibilità consiste nella

$$E(s, s_*) \equiv \lambda \frac{(s_*/s)^\alpha}{1 + (s_*/s)^\alpha}, \quad [RE]_S$$

in cui $\lambda \in (0,1)$ e $\alpha \in \mathbb{R}^+$; in essenza è la [10, (2.2), § 2, p. 223] modificata mediante la [10, (4.5), § 4, p. 228], ossia è una funzione di Hill di ordine α , in cui v'è un tasso di emigrazione maggiore verso città con popolazione relativa, data dal rapporto s_*/s , maggiore.

I risultati nella {§}, ottenuti tramite i parametri nella Tab. 4.3, non sono purtroppo corretti dato che non solo mostrano dei centri abitati spopolarsi, ma che questi sono più per la maggior parte i le città più popolate.

	λ	α	N_t	R
Stabile	0.1	0.5	10^7	100
Instabile	0.1	1.5	10^5	10

Tabella 4.3: Dati e parametri della $[RE]_S$.

4.3.2 Regola taglia-gradi

$$E_{SK}(s, s_*, i, i_*) \equiv \lambda \frac{[(s_*/s)(k_{i_*}/k_i)]^\alpha}{1 + [(s_*/s)(k_{i_*}/k_i)]^\alpha}, \quad [RE]_{SK}$$

λ	α	N_t
0.15	0.63	2×10^6

Tabella 4.4: Dati e parametri della $[RE]_{SK}$.

4.3.3 Regola frazionata

$$E_{SK}^f(s, s_*, i, i_*) \equiv (1 - \zeta) E_{SK}(s, s_*, k_i, k_{i_*}) + \zeta E_{SK}(s_*, s, k_{i_*}, k_i) \quad [RE]_{SK}^f$$

4.3.4 Regola taglia-forza

$$E_{SW}(s, s_*, i, i_*) \equiv \lambda \frac{[(s_*/s)(w_{i_*}/w_i)]^\alpha}{1 + [(s_*/s)(w_{i_*}/w_i)]^\alpha}, \quad [RE]_{SW}$$

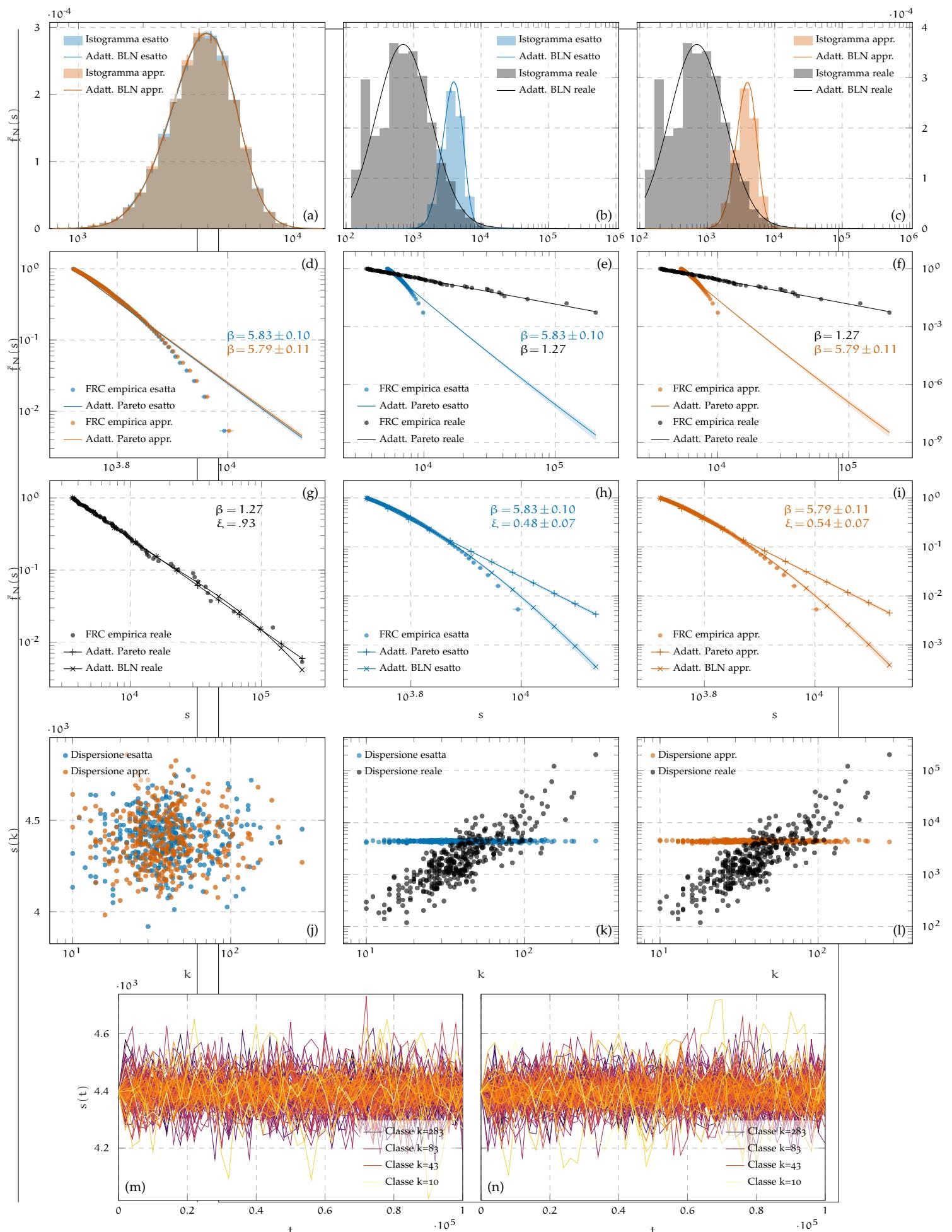


Figura 4.2: Studio della configurazione di riferimento stabile della [RE]_S con parametri dalla Tab. 4.3; per la spiegazione dei grafici si veda il § 4.2.3.

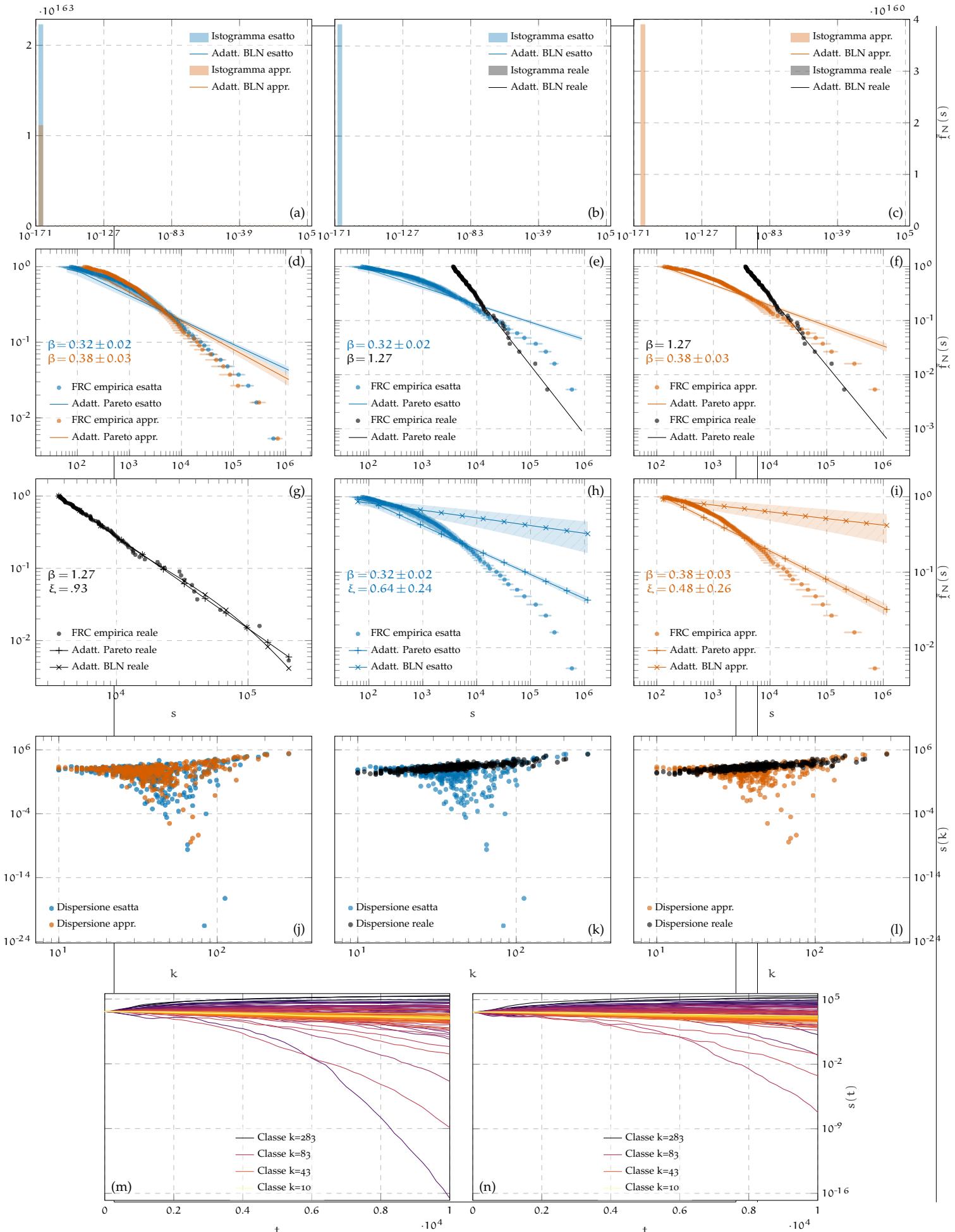


Figura 4.3: Studio della configurazione di riferimento instabile della $[RE]_S$ con parametri dalla Tab. 4.3; per la spiegazione dei grafici si veda il § 4.2.3.

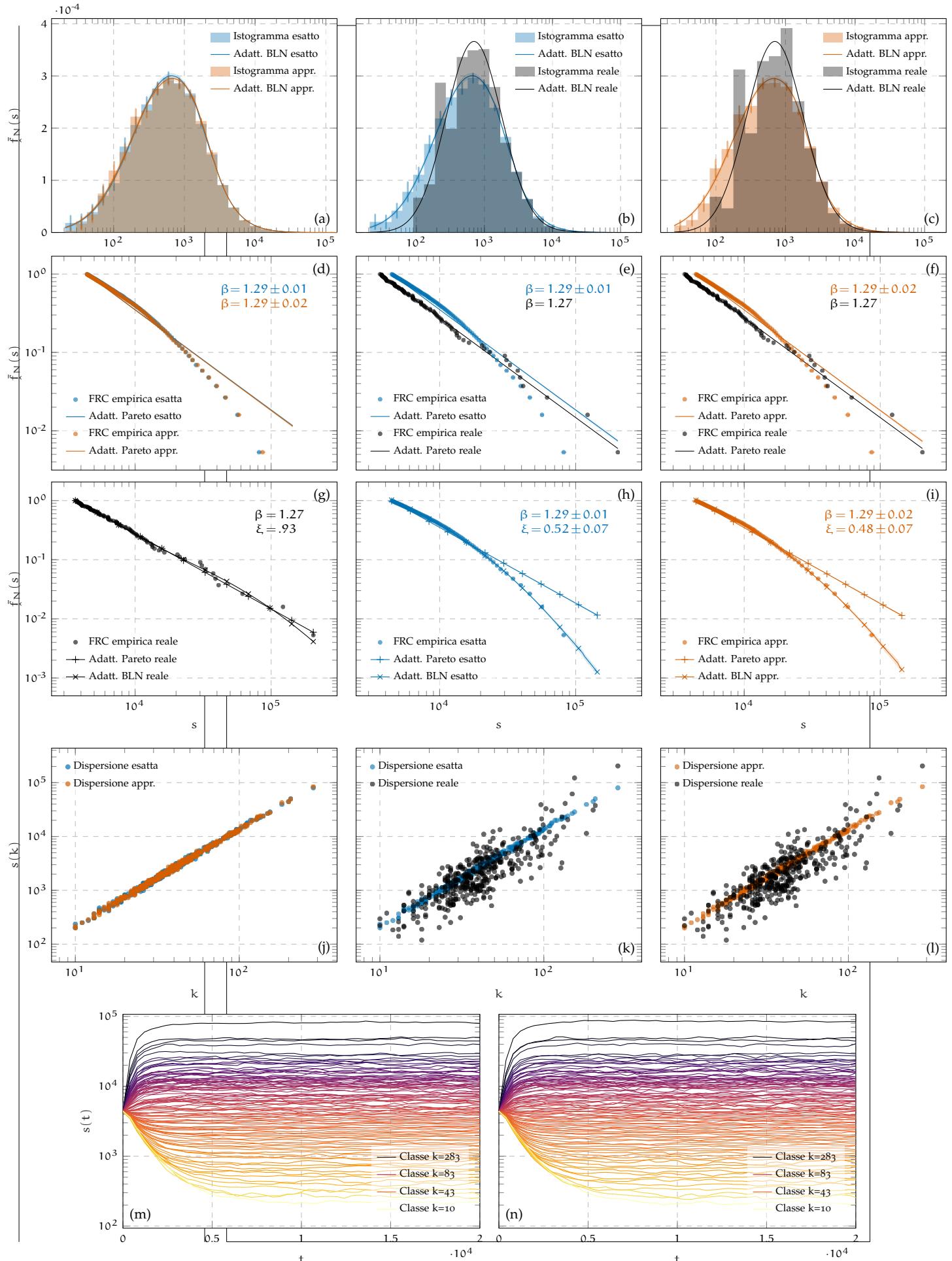


Figura 4.4: Studio della configurazione di riferimento della (RE_{SK}) con parametri dalla Tab. 4.4; per la spiegazione dei grafici si veda il § 4.2.3.

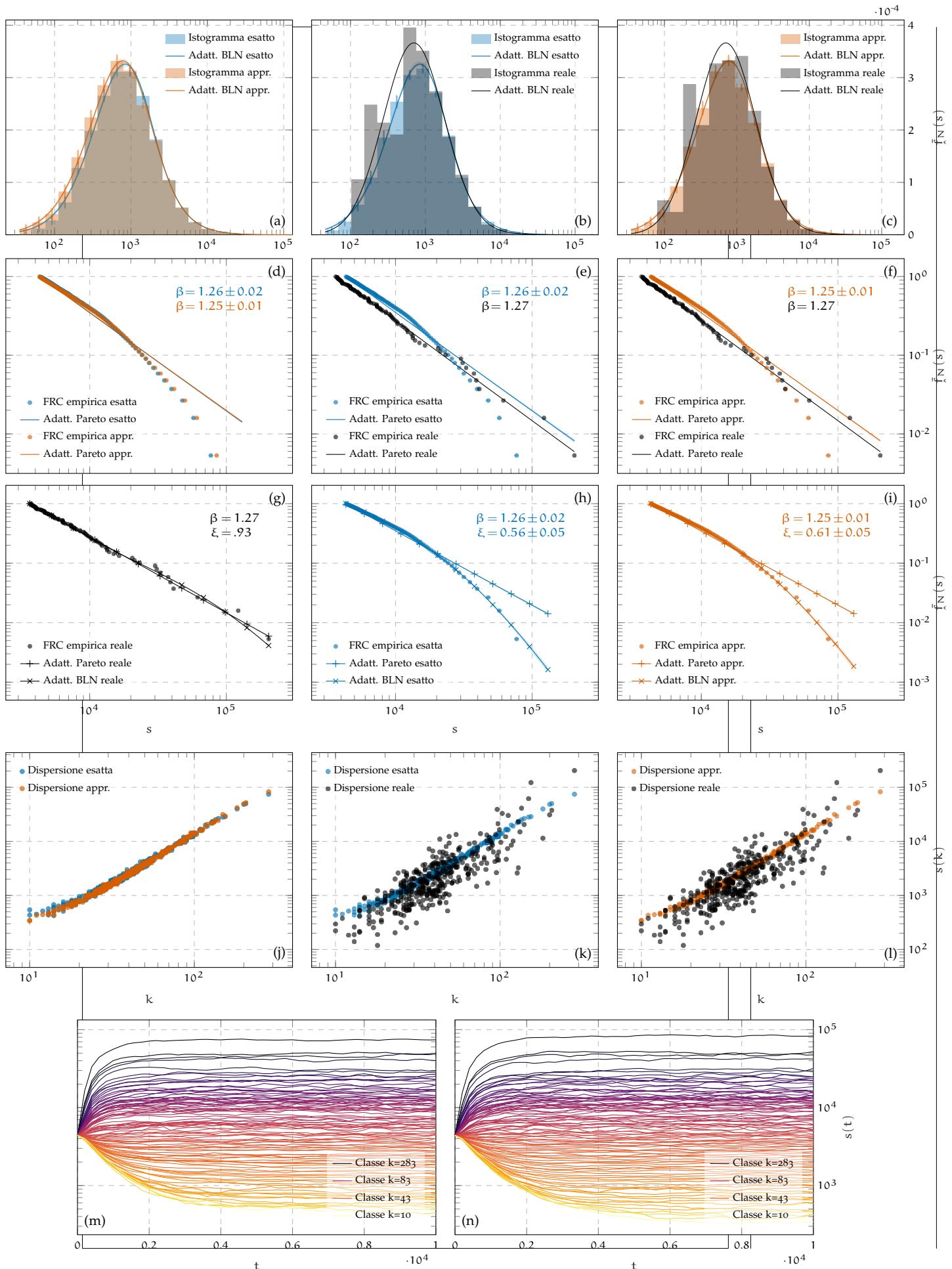


Figura 4.5: Studio della configurazione di riferimento della $(\text{RE})_{SK}^f$ con parametri dalla Tab. 4.5; per la spiegazione dei grafici si veda il § 4.2.3.

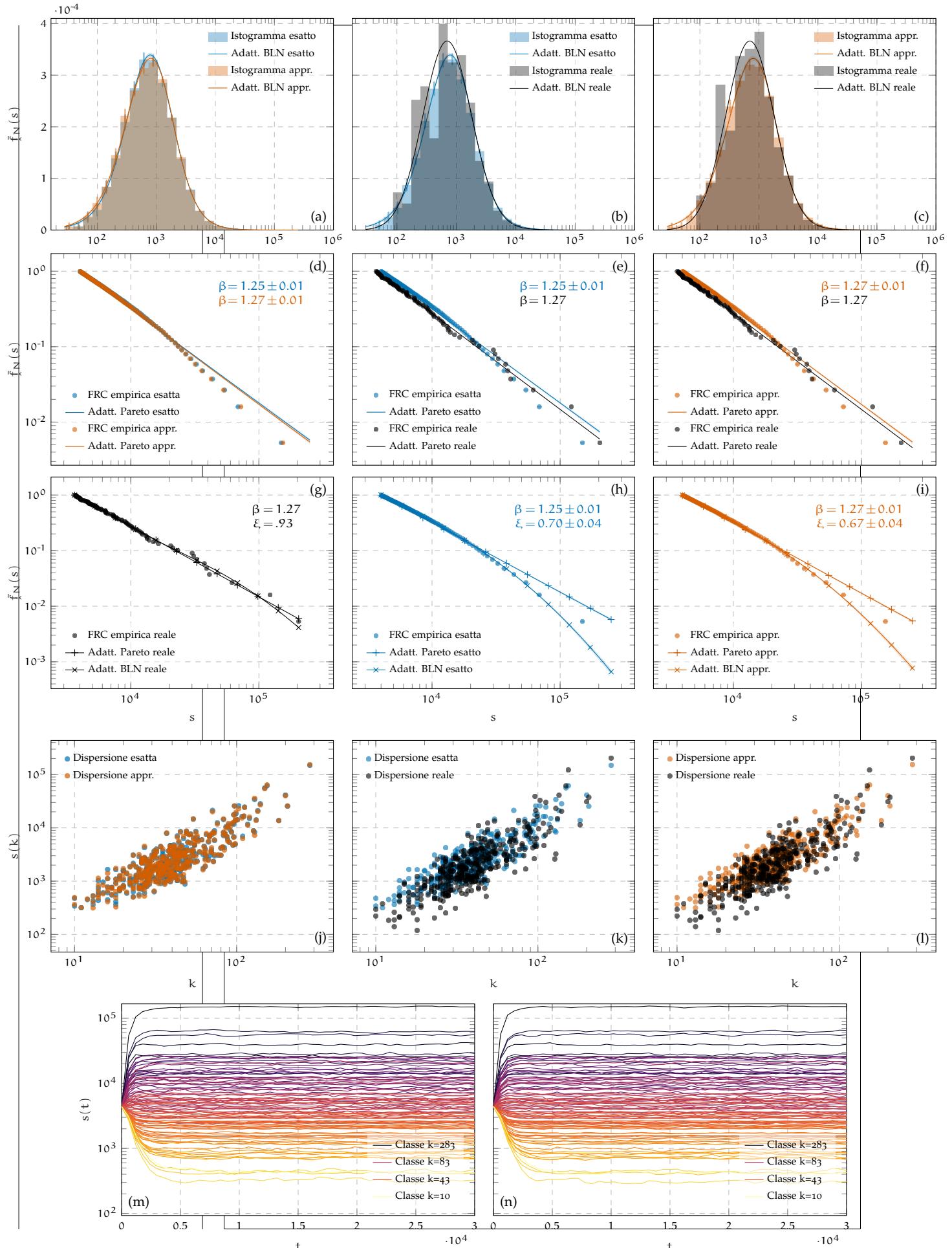


Figura 4.6: Studio della configurazione di riferimento della $[RE]_{SW}$ con parametri dalla Tab. 4.6; per la spiegazione dei grafici si veda il § 4.2.3.

λ	α	ζ	N_t
0.15	0.67	0.01	10^6

Tabella 4.5: Dati e parametri della $[RE]_{SK}^f$.

λ	α	N_t
1	1	1

Tabella 4.6: Dati e parametri della $[RE]_{SW}$.

1188 4.3.5 Interpretazioni

1189 In poche parole la ($[RE]_S$) descrive la tendenza degl'individui di aggregarsi
 1190 per i piú svariati motivi: lavoro, sicurezza, famiglia, eccetera.

1191 I due principali parametri del modello possono quindi essere cosí inter-
 1192 pretati:

- 1193 ◇ $\lambda \in (0,1)$ rappresenta l'attrattività dei poli, ossia la frazione che le
 1194 città piú popolose riescono al massimo ad attrarre in un'interazione;
- 1195 ◇ $\alpha \in \mathbb{R}^+$ indica la rapidità d'emigrazione e influenza quanto rapidamente
 1196 il rapporto s_*/s raggiunge la massima attrattività λ .

1197 4.4 ALTRI CASI NOTEVOLI

1198 4.4.1 Varianti delle regole d'interazione

1199 4.4.2 Il caso dell'Italia

1200 4.4.3 Il caso della Valle d'Aosta

1201

5 | CONCLUSIONI

- 1202 1. Maggiori sviluppi teorici, specialmente per ricavare un'equazione di
1203 Fokker-Planck per la distribuzione stazionaria.
- 1204 2. Maggiori sviluppi pratici per abbandonare l'ipotesi semplificativa
1205 della rete statica.
- 1206 3. Applicare questa teoria anche a reti europee o internazionali.

1207 APPENDICE

1208 A CODICE

1 **Listato A.1: Codice «main.py».**

```
1  from multiprocessing import freeze_support
2
3  def main():
4      import libGUIs
5      import libData as libD
6      import libNetworks as libN
7      import libKTMAS as libK
8
9
10     ### Graphic User Interface ###
11     clsGUI = libGUIs.ParametersGUI()
12     clsPrm = clsGUI.GatherParameters() # Parameters
13
14
15     if clsPrm.simFlag:
16
17         ### Data extraction ###
18         if clsPrm.extraction: libD.ExtractRegionData()
19
20
21         ### Matrices extraction ###
22         clsReg = libD.LoadRegionData(clsPrm.region)
23
24
25         ### Network analysis ###
26         if clsPrm.analysis:
27             clsNA = libN.NetworkAnalysis(clsPrm,clsReg)
28
29             clsNA.DegreeDistributionFig()
30             clsNA.WeightDistributionFig()
31             clsNA.StrengthDistributionFig()
32
33             clsNA.BetweennessCentralityFig()
34             clsNA.StrengthVsDegreeFig()
35
36             clsNA.AClusteringCoefficientFig()
37             clsNA.WClusteringCoefficientFig()
38
39             clsNA.AAssortativityFig()
40             clsNA.WAssortativityFig()
41
42             # clsNA.ShowFig()
43
44
45         ### Kinetic simulation ###
46         if clsPrm.parametricStudy:
47             clsKS = libK.ParametricStudy(clsPrm,clsReg)
48         else:
49             clsKS = libK.KineticSimulation(clsPrm,clsReg)
50
51         clsKS.MonteCarloSimulation()
52
53         clsKS.SizeDistrFittingsFig()
54         clsKS.AverageSizeFig()
55         clsKS.SizeVsDegreeFig()
56         clsKS.SizeDistrEvolutionFig()
57         clsKS.SizeEvolutionsFig()
```

```

58
59     # clsKS.ShowFig()
60
61 if __name__ == "__main__":
62     freeze_support()
63     main()
64
65
66     Listato A.2: Codice «libParameters.py».
67
68     # Library to more easily define/change parameters
69
70
71     from dataclasses import dataclass
72     from typing import Any
73
74     from copy import deepcopy
75     import numpy as np
76
77     from pathlib import Path
78
79
80     ### Module attributes ###
81
82
83     mainFolder      = Path(__file__).resolve().parent
84     projectFolder   = mainFolder.parent
85     dataFolder       = projectFolder/'Dati'
86
87     matrixZipPath   = dataFolder/'MatriciPendolarismo1991.zip'
88     sizeZipPath     = dataFolder/'CensimentoRegioni1991.zip'
89     coordZipPath    = dataFolder/'LimitiRegioni1991.zip'
90     shpFilePath     = f"zip://{{coordZipPath}}!Limiti1991_g/Com1991_g/Com ]
91      ↵ 1991_g_WGS84.shp"
92
93
94     regDataZipPath = dataFolder/'DatiRegioni1991.zip'
95     simDataZipFile = dataFolder/'DatiSimulazione.zip'
96
97
98     parameters = {
99         "population": {
100             "text": "S",
101             "val": 1
102         },
103         "attractivity": {
104             "text": "\u03bb",
105             "val": 1.0
106         },
107         "convincibility": {
108             "text": "\u03b1",
109             "val": 1.0
110         },
111         "deviation": {
112             "text": "\u03c3",
113             "val": 1.0
114         },
115         "region": {
116             "text": "Region selected",
117             "val": "region"
118         },
119         "zetaValue": {
120             "text": "\u03b6",
121             "val": 1.0
122         },
123         "timestep": {
124             "text": "\u0394t",
125             "val": 1.0
126         },
127         "timesteps": {
128             "text": "Nt",
129             "val": 1
130         }
131     },
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158

```

```

59     "iterations": {
60         "text": "Ni",
61         "val": 1
62     },
63     "progressBar": {
64         "text": "Progress Bar",
65         "val": True
66     },
67     "extraction": {
68         "text": "Extract data",
69         "val": False
70     },
71     "analysis": {
72         "text": "Network analysis",
73         "val": False
74     },
75     "edgeWeights": {
76         "text": "Edge weights",
77         "val": False
78     },
79     "fluctuations": {
80         "text": "Fluctuations",
81         "val": True
82     },
83     "zetaFraction": {
84         "text": "Fraction",
85         "val": False
86     },
87     "interactingRule": {
88         "text": "Interacting law",
89         "val": "law"
90     },
91     "PdfPopUp": {
92         "text": "Open PDF",
93         "val": False
94     },
95     "LaTeXConversion": {
96         "text": "LaTeX Conversion",
97         "val": False
98     },
99     "snapshots": {
100         "text": "Ns",
101         "val": 100
102     },
103     "smoothingFactor": {
104         "text": "Sf",
105         "val": 10
106     },
107     "parametricStudy": {
108         "text": "Parametric study",
109         "val": False
110     },
111     "studiedParameter": {
112         "text": "Studied parameter",
113         "val": "study"
114     },
115     "startValuePrmStudy": {
116         "text": "Start",
117         "val": 1.0
118     },
119     "endValuePrmStudy": {
120         "text": "End",
121         "val": 1.0
122     },
123     "numberPrmStudy": {
124         "text": "Nv",
125         "val": 1

```

```

126     }
127 }
128
129 regionList = [
130     'Piemonte',
131     "Valle d'Aosta",
132     'Lombardia',
133     'Trentino-Alto Adige',
134     'Veneto',
135     'Friuli-Venezia Giulia',
136     'Liguria',
137     'Emilia-Romagna',
138     'Toscana',
139     'Umbria',
140     'Marche',
141     'Lazio',
142     'Abruzzo',
143     'Molise',
144     'Campania',
145     'Puglia',
146     'Basilicata',
147     'Calabria',
148     'Sicilia',
149     'Sardegna',
150     'Italia'
151 ]
152
153 intRuleList = [
154     ' $\lambda(rs^\alpha)/(1+rs^\alpha)$ ', # 0
155     ' $\lambda(rsk/\alpha)/(1+rsk/\alpha)$ ', # 1
156     ' $\lambda(rsk^\alpha)/(1+rsk^\alpha)$ ', # 2
157     ' $\lambda[rsk/(1+rsk)]^\alpha$ ', # 3
158 ]
159
160 caseStudies = {
161     "selected": "Default",
162     "list": {
163         "Default": {
164             "attractivity": .18, #.18,
165             "convincibility": .44, #.44,
166             "deviation": 0.05,
167             "region": 19,
168             "zetaValue": 0.01,
169             "timestep": 0.01,
170             "timesteps": int(1e6),
171             "iterations": 100,
172             "progressBar": False,
173             "extraction": False,
174             "analysis": False,
175             "edgeWeights": True,
176             "fluctuations": False,
177             "zetaFraction": True,
178             "interactingRule": 2,
179             "PdfPopUp": False,
180             "LaTeXConversion": False,
181             "snapshots": 1000,
182             "smoothingFactor": 50,
183             "studiedParameter": 1,
184             "startValuePrmStudy": 6.5e-1,
185             "endValuePrmStudy": 7.5e-1,
186             "numberPrmStudy": 10,
187             "parametricStudy": False
188         },
189         " $\lambda(rsk/\alpha)/(1+rsk/\alpha)$ ": {
190             "attractivity": 0.05,
191             "deviation": 0.05,
192             "region": 19,

```

```

193     "zetaValue": 0.1,
194     "timestep": 0.01,
195     "timesteps": int(5e5),
196     "iterations": 15,
197     "progressBar": False,
198     "extraction": False,
199     "analysis": False,
200     "edgeWeights": False,
201     "fluctuations": True,
202     "zetaFraction": False,
203     "interactingRule": 1,
204     "PdfPopUp": False,
205     "LaTeXConversion": False,
206     "snapshots": 100,
207     "smoothingFactor": 10,
208     "studiedParameter": 0,
209     "startValuePrmStudy": 1.0,
210     "endValuePrmStudy": 1.0,
211     "numberPrmStudy": 1,
212     "parametricStudy": False
213 },
214 "(1- $\zeta$ )efl_k/ $\alpha$ + $\zeta$ efs_k": {
215     "attractivity": 0.05,
216     "deviation": 0.05,
217     "region": 19,
218     "zetaValue": 0.1,
219     "timestep": 0.01,
220     "timesteps": int(1e7),
221     "iterations": 15,
222     "progressBar": False,
223     "extraction": False,
224     "analysis": False,
225     "edgeWeights": False,
226     "fluctuations": True,
227     "zetaFraction": True,
228     "interactingRule": 1,
229     "PdfPopUp": False,
230     "LaTeXConversion": False,
231     "snapshots": 100,
232     "smoothingFactor": 10,
233     "studiedParameter": 0,
234     "startValuePrmStudy": 0.01,
235     "endValuePrmStudy": 0.1,
236     "numberPrmStudy": 5,
237     "parametricStudy": True
238 },
239 " $\lambda(rsk^\alpha)/(1+rsk^\alpha)$ 
```

```

250         "startValuePrmStudy": 0.1,
251         "endValuePrmStudy": 1,
252         "numberPrmStudy": 10,
253         "parametricStudy": False
254     },
255     " $\lambda[rsk/(1+rsk)]^{\alpha}$ ": {
256         "attractivity": 0.05,
257         "convincibility": 0.3,
258         "deviation": 0.05,
259         "region": 19,
260         "zetaValue": 0.1,
261         "timestep": 0.01,
262         "timesteps": int(1e7),
263         "iterations": 15,
264         "progressBar": False,
265         "extraction": False,
266         "analysis": False,
267         "edgeWeights": False,
268         "fluctuations": True,
269         "zetaFraction": False,
270         "interactingRule": 3,
271         "PdfPopUp": False,
272         "LaTeXConversion": False,
273         "snapshots": 100,
274         "smoothingFactor": 10,
275         "studiedParameter": 1,
276         "startValuePrmStudy": 0.3,
277         "endValuePrmStudy": 1,
278         "numberPrmStudy": 3,
279         "parametricStudy": False
280     }
281   }
282 }
283
284 prmStudyList = [
285   parameters['attractivity']['text'],
286   parameters['convincibility']['text'],
287   parameters['zetaValue']['text']
288 ]
289
290 regPopDict = { # Italian region sizes in 1991
291   'Piemonte':int(4302565),
292   "Valle d'Aosta":int(115938),
293   'Lombardia':int(8856074),
294   'Trentino-Alto Adige':int(890360),
295   'Veneto':int(4380797),
296   'Friuli-Venezia Giulia':int(1197666),
297   'Liguria':int(1676282),
298   'Emilia-Romagna':int(3909512),
299   'Toscana':int(3529946),
300   'Umbria':int(811831),
301   'Marche':int(1429205),
302   'Lazio':int(5140371),
303   'Abruzzo':int(1249054),
304   'Molise':int(330900),
305   'Campania':int(5630280),
306   'Puglia':int(4031885),
307   'Basilicata':int(610528),
308   'Calabria':int(2070203),
309   'Sicilia':int(4966386),
310   'Sardegna':int(1648248),
311   'Italia':int(56778031)
312 } # See Table 6.1 on p. 488 of «ISTAT Popolazione e abitazioni 1991
313   ↪ {04-12-2025}.pdf»
314
315 workersShMTTemplate = {
316   'handles':[],
```

```

326     'parameters':{
327         'Mdt': None,
328         'wOdt': None,
329         'wI': None,
330         'di': None,
331         'idi': None,
332         'ns': None
333     },
334     'gui':{
335         'progress': np.int64,
336         'elapsed': np.float64,
337         'done': np.int8
338     }
339 }
340
341 #### Main classes and function ####
342
343 @dataclass(eq=False)
344 class Parameter:
345     text: Any = None
346     val: Any = None
347     var: Any = None
348     lbl: Any = None
349     wid: Any = None
350     frame: Any = None
351     list: Any = None
352     cbid: Any = None
353
354 class Parameters():
355     def __init__(self,**kwargs):
356         for text,value in kwargs.items():
357             setattr(self,text,value)
358
359 class ComboBoxList():
360     def __init__(self,attribute,list):
361         attribute.list = list
362         self.list = list
363         self.code = {
364             r:i for i,r in enumerate(list)
365         }
366
367 def CopyWorkerShMTTemplate(): return deepcopy(workersShMTTemplate)

```

Listato A.3: Codice «libData.py».

```

1 # Library to handle [especially writing and reading] data
2
3 import zipfile as zf
4 from zipfile import ZipFile as ZF
5
6 from io import TextIOWrapper as tiow
7 from io import BytesIO as bio
8 from io import StringIO as sio
9
10 import pandas as pd
11 import csv,json
12 import geopandas as gpd
13
14 import numpy as np
15
16 import libParameters as libP
17
18 #### Main functions ####
19
20 def ExtractRegionData():
21     dictMun,dictReg = ReadMunRegCodes()

```

```

23     BuildAdjacencyMatrices(dictMun,dictReg)
24     BuildSizeDistributions(dictReg)
25
26     WriteRegionData(dictReg)
27
28     def LoadRegionData(code):
29         el = {
30             'A':'txt',
31             'W':'txt',
32             'li2Name':'.json',
33             'li2Coord':'.txt',
34             'name2li':'.json',
35             'sizeDistr':'.txt',
36             'Nc':'.json'
37         }
38         parameters = {}
39
40         with ZF(libP.regDataZipPath) as z:
41             for data,ext in el.items():
42                 path = f'{code:02d}/{data}{ext}'
43                 with z.open(path,'r') as f:
44                     match data:
45                         case 'A' | 'W' | 'li2Coord' | 'sizeDistr':
46                             parameters[data] = np.loadtxt(
47                                 f,
48                                 encoding='utf-8',
49                                 delimiter=",",
50                                 dtype=int
51                             )
52
53                         case 'li2Name':
54                             parameters[data] = {
55                                 int(k): v for k, v in
56                                 json.load(f).items()
57                             } # This is necessary since the keys of a
58                             # «.json» file are always strings, hence
59                             # they have to be converted to integer if
60                             # originally they were such
61
62                         case 'Nc' | 'name2li':
63                             parameters[data] = json.load(f)
64
65             return libP.Parameters(**parameters)
66
67
68     ### Auxiliary functions ###
69
70     def xls2csv(
71         xlsFile,
72         zipFile=None
73     ):
74         if zipFile is None:
75             b = xlsFile
76         else:
77             with ZF(zipFile) as z:
78                 b = z.read(xlsFile) # Extract raw bytes from
79                 # «elencom91.xls»
80
81         streamXls = bio(b) # Create a binary stream in RAM from the
82         # bytes «b»
83         df = pd.read_excel( # Read the Excel content
84             streamXls,
85             # from «streamXLS»
86             dtype=str,
87             # treating columns as strings
88             engine="xlrd",
89             # using «xlrd»
90             sheet_name=0 # for only the first worksheet
91         ) # «df» stands for «DataFrame» from «pandas.DataFrame»
92         streamCsv = sio() # Allocate a text buffer in RAM that
93         # behaves like a writable text file (UTF-8 encoding)

```

```

13 df.to_csv(          # Serialize «df» as «.csv»
14     streamCsv,      # into «streamCSV» and
15     index=False      # without the DataFrame's row numbers
16 )
17 streamCsv.seek(0)   # Reset the text buffer cursor to the start
18     ↳ so it can be read from the beginning
19
20 return streamCsv
21
22 def ReadMunRegCodes(): # Read Municipality-Region Codes
23     # Conversion of «fileName» from the old format «.xls» to a more
24     ↳ manageable «.CSV»
25     matrixCSV = xls2csv('elencom91.xls',libP.matrixZipPath)
26
27     # These two dictionary are necessary to link municipalities and
28     ↳ regions via their codes defined in «file», which will be
29     ↳ useful later on to extract the actual data for the
30     ↳ adjacency matrices
31     dictMun = {} # Dictionary to link municipality codes with
32     ↳ region codes
33     dictReg = {
34         i+1:{
35             'li2Name':{}, # Dictionary to link local indices with
36             ↳ the municipality name
37             'li2Coord':{}, # Dictionary to link local indices with
38             ↳ the municipality representative point
39             'name2li':{}, # Dictionary to link local indices with
40             ↳ the municipality name
41             'code2li':{}, # Dictionary to link municipality codes
42             ↳ with local indices
43             'Nc':0 # Number of cities in a region
44         } for i in range(21)
45     } # The index 21 is arbitrarily associated to Italy viewed as
46     ↳ the 21th region, hence its local index is actually the
47     ↳ global one
48
49     gdf = gpd.read_file(libP.shpFilePath).set_index('PRO_COM_T').g_
50     ↳ eometry.representative_point()
51     reader = csv.reader(matrixCSV)
52     next(reader) # Skip header line containing metadata labels
53     for row in reader:
54         try: # If the code is not empty
55             codeReg = int(row[0]) # Region code
56             codeMun = row[3]      # Municipality code
57             nameMun = row[4]      # Municipality name
58
59             dictMun[codeMun] = codeReg
60             # In reality «codeMun» it's more like «(Province
61             ↳ code)+(Municipality code)»
62
63             UpdateDictionary(
64                 dictReg,
65                 codeReg,
66                 nameMun,
67                 codeMun,
68                 gdf
69             )
70
71             UpdateDictionary(
72                 dictReg,
73                 21,
74                 nameMun,
75                 codeMun,
76                 gdf
77             )
78
79         except ValueError:
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135

```

```

136         continue # Ignore it otherwise
137
138     return dictMun, dictReg
139
140 def UpdateDictionary(
141     dictReg,
142     codeReg,
143     nameMun,
144     codeMun,
145     gdf
146 ):
147     lgi = dictReg[codeReg]['Nc'] # Local/Global index
148     dictReg[codeReg]['li2Name'][lgi] = nameMun
149     dictReg[codeReg]['li2Coord'][lgi] =
150         ↪ [gdf[codeMun].x,gdf[codeMun].y]
151     dictReg[codeReg]['name2li'][nameMun] = lgi
152     dictReg[codeReg]['code2li'][codeMun] = lgi
153
154     dictReg[codeReg]['Nc'] = lgi+1 # Update local/global number of
155         ↪ cities
156     # Local (codeReg!=21) index
157     # Global (codeReg==21) index
158
159 def BuildAdjacencyMatrices(
160     dictMun,
161     dictReg
162 ):
163
164     for r in dictReg:
165         Nc = dictReg[r]['Nc']
166
167         for (M,numTyp) in [
168             ('A',np.uint8), # 'A' == [Unitary] Adjacency matrix
169             ('W',np.int64) # 'W' == Weighted adjacency matrix
170         ]:
171             dictReg[r][M] = np.zeros((Nc,Nc),dtype=numTyp)
172
173             li2Coord = np.empty((Nc,2),np.float64)
174             for i in range(Nc):
175                 li2Coord[i,0] = dictReg[r]['li2Coord'][i][0]
176                 li2Coord[i,1] = dictReg[r]['li2Coord'][i][1]
177             dictReg[r]['li2Coord'] = li2Coord
178
179             with ZF(libP.matrixZipPath) as z, z.open('Pen_91It.txt') as f:
180                 for line in tiow(f,encoding="utf-8"):
181                     oMun = line[:6]           # Origin municipality
182                     dMun = line[11:17]        # Destination municipality
183                     commuters = int(line[17:-1]) # Edge weight (commuters)
184
185                     if oMun != dMun and ' ' not in dMun: # and ' ' not in
186                         ↪ oMun
187                         try:
188                             oReg = dictMun[oMun] # Origin region
189                             dReg = dictMun[dMun] # Destination region
190
191                             if oReg == dReg: # and commuters!=0
192                                 UpdateMatrices(
193                                     dictReg,
194                                     commuters,
195                                     oReg,dReg,
196                                     oMun,dMun
197                                 )
198
199             UpdateMatrices(
200                 dictReg,
201                 commuters,
202                 21,21,
203             )

```

```

200                     oMun,dMun
201             )
202         except Exception:
203             continue
204
205         # else:
206         #     if oMun != dMun:
207         #         prova = f'{oMun[:3]>{dMun[:3]}'
208         #         try:
209         #             dictMun[prova]
210         #         except KeyError:
211         #             print(line)
212
213         # There are in total two majors typos in the data:
214         #     1. There is a municipality of code «'022008'» which isn't
215         #         listed in «elencom91.xls», raising always a KeyError
216         #     2. Some lines in «Pen_91It.txt» contain a dMun which is
217         #         incomplete, lacking in particular the first three numbers
218         #         for the province code; these are in total:
219         #             '215  ','229  ','241  ','216  ','203  ','224
220         #             ','236  ','246 '
221         #     Moreover, nothing can be done about it, not even
222         #     assuming that dReg shares the same province code of oMun,
223         #     as some of these are ambiguous, even in the same region
224         #     For instance, the very first one '215  ' appears in
225         #     the line «'00200714212215      1\n'»; leaving aside
226         #     the fact that there are in total 6 cities with the same
227         #     municipality code accross the italian peninsula, even just
228         #     considering the Piedmont region, i.e. the same one of
229         #     «oMun='002007'» («ASIGLIANO VERCELLESE»), there exists
230         #     actually two cities with such a code: «'001215'» («RIVA
231         #     PRESSO CHIERI») and «'004215'» («SAVIGLIAN»O), and both do
232         #     not have the same province code («'002'») of oMun. In plain
233         #     words it's impossible which one has to be chosen
234
235         # Therefore the if statement excludes municipalities whose
236         #     codes are only partially written due to, I presume, typos
237         #     from ISTAT
238         # while, the try statement catches the few cases where the code
239         #     does not match any municipality (only one: «022008»)
240
241     def UpdateMatrices(
242         dictReg,commuters,
243         oReg,dReg,
244         oMun,dMun
245     ):
246         oI = dictReg[oReg]['code2li'][oMun] # Local/Global origin index
247         dI = dictReg[dReg]['code2li'][dMun] # Local/Global destination
248         # index
249         # The index is considered global iff «oReg=dReg=21»
250
251         dictReg[oReg]['A'][oI,dI] = 1
252         dictReg[dReg]['A'][dI,oI] = 1
253         dictReg[oReg]['W'][oI,dI] += commuters
254         dictReg[dReg]['W'][dI,oI] += commuters
255         # The sum in «matricesReg[oReg]['W'][oI,dI] += commuters» is
256         # necessary as there are repeating origin-destination links
257         # in the dataset
258
259     def BuildSizeDistributions(
260         dictReg
261     ):
262         sizeDistribution = {}
263
264         with ZF(libP.sizeZipPath) as z:
265             Nc = dictReg[21]['Nc']
266             sizeDistribution[21] = np.zeros((Nc,), dtype=np.int64)

```



```

306                         delimiter=", "
307                     )
308
309                 case '.json':
310                     json.dump(
311                         dictReg[r+1][data],
312                         buf,
313                         indent=3 if data != 'Nc' else 0
314                     )
315
316                     z.writestr(path,buf.getvalue())
317
318             def WriteSimulationData(
319                 vrtState,
320                 snapshots,
321                 siVrtState,
322                 typ,
323                 lbl,
324                 li2Name,
325                 Ni,
326                 sid
327             ):
328                 lbl = [t.replace('.','') for t in lbl]
329
330                 mode = 'a' if sid is not None and sid != 1 else 'w'
331                 with ZF(
332                     libP.simDataZipFile,mode,
333                     compression=zf.ZIP_DEFLATED,
334                     compresslevel=9
335                 ) as z:
336                     for r in range(Ni):
337                         dicName2SortedPop = {
338                             t:{
339                                 li2Name[i]:vrtState[r,i,t] for i in
340                                     ↳ siVrtState[r,:,:-1,t]
341                                 } for t in typ
342                             }
343
344                         folder = (
345                             f'{}' if sid is None else f's{sid}/'
346                             f'{0 if r+1<10 else ""}{r+1}'
347                         )
348                         for t in typ:
349                             buf = bio()
350                             path = f'{folder}/{lbl[t]}CitySizesFinal.npy'
351                             np.save(buf,vrtState[r,:,:t])
352                             z.writestr(path,buf.getvalue())
353
354                             buf = sio()
355                             path = f'{folder}/{lbl[t]}CitySizesSorted.json'
356                             json.dump(dicName2SortedPop[t],buf,indent=3)
357                             z.writestr(path,buf.getvalue())
358
359                             buf = bio()
360                             path = f'{folder}/{lbl[t]}S snapshots.npy'
361                             np.save(buf,snapshots[r,:,:t])
362                             z.writestr(path,buf.getvalue())
363
364             def SetParameters(cls):
365                 parameters = libP.parameters
366                 for name,kwarg in parameters.items():
367                     setattr(cls,name,libP.Parameter(**kwarg))
368
369             def LoadCaseStudies(cls):
370                 data = libP.caseStudies
371
372                 caseStudies = data['list']

```

```

372     selectedCS = data['selected']
373
374     listCS = list(caseStudies.keys())
375     dictCS = {}
376
377     for name,study in caseStudies.items():
378         dictCS[name] = {}
379
380         for key,val in study.items():
381             prm = getattr(cls,key)
382             dictCS[name][prm] = val
383
384         # After all parameters have been loaded the external lists
385         # are saved as well
386         for (prmName,prmlist) in [
387             ('region','regionList'),
388             ('interactingRule','intRuleList'),
389             ('studiedParameter','studiedPrmList')
390         ]:
391             prm = getattr(cls,prmName)
392             value = dictCS[name][prm]
393             dictCS[name][prm] = getattr(cls,prmlist).list[value]
394
395     return dictCS, selectedCS, listCS

```

Listato A.4: Codice «libFigures.py».

```

1  # Library to create figures
2
3  # import os, sys
4  from pathlib import Path
5  import subprocess
6
7  import numpy as np
8
9  from scipy.io import savemat
10 from scipy import stats
11 from scipy.optimize import minimize
12
13 import matplotlib.pyplot as plt
14 from matplotlib.pyplot import savefig
15 # import mplcursors
16
17 from libParameters import projectFolder
18
19
20 ### Main class and functions ####
21
22 # Figure data
23 class FigData():
24     def __init__(self,clsPrm,folder):
25         self.folder = folder
26         self.projectFolder = projectFolder
27
28         self.regCode = str(clsPrm.region)
29         self.PdfPopUp = clsPrm.PdfPopUp
30         self.LaTeXConversion = clsPrm.LaTeXConversion
31
32         for ext in ('.pdf','.mat','.tex'):
33             if ext=='.pdf' or self.LaTeXConversion:
34                 setattr(self,f'{ext[1:]}FolderPath',self.CreateFolders(ext))
35                 # for p in folder.iterdir():
36                 #     if p.is_file():
37                 #         p.unlink()
38
39     def SetFigs(self,nRow=1,nCol=1,size=None):
40         nFig = nRow*nCol; self.nFig = nFig

```

```

#1
#2     if not isinstance(nFig,int) or nFig <= 0:
#3         raise ValueError('The number of figures must be a
#4             → integer positive number')
#5
#6     for i in range(nFig):
#7         setattr(self,f'fig{i+1}', {'plots':[], 'style':{}})
#8
#9     if nFig == 1:
#10        self.fig = self.fig1
#11        return plt.figure()
#12    else:
#13        return plt.subplots(
#14            nRow,nCol,
#15            figsize=size,
#16            gridspec_kw=dict(
#17                wspace=0.2,
#18                hspace=0.2
#19            )
#20        )
#21
#22
#23 def SaveFig(self,name):
#24     self.SaveFig2pdf(name)
#25
#26     self.names = [
#27         f'{name}fig{f+1}' for f in range(self.nFig)
#28     ] if self.nFig>1 else [name]
#29
#30     if self.LaTeXConversion:
#31         for f in range(self.nFig):
#32             self.SaveFig2mat(f)
#33             self.SaveFig2tex(f)
#34
#35 def SaveFig2pdf(self,name):
#36     format = '.pdf'
#37     pdfFilePath = self.FigPath(name,format)
#38
#39     savefig(
#40         pdfFilePath,
#41         dpi=300,
#42         bbox_inches='tight'
#43     )
#44
#45     if self.PdfPopUp:
#46         sumatraPath =
#47             → self.projectFolder/'.vscode'/'SumatraPDF.lnk'
#48         cmd = f'"{str(sumatraPath)}" "{str(pdfFilePath)}"'
#49         subprocess.Popen(cmd,shell=True)
#50
#51 def SaveFig2mat(self,f):
#52     format = '.mat'
#53     self.matFilePath = self.FigPath(self.names[f],format)
#54
#55     savemat(self.matFilePath,getattr(self,f'fig{f+1}'))
#56
#57 def SaveFig2tex(self,f):
#58     format = '.tex'
#59     self.texFilePath = self.FigPath(self.names[f],format)
#60
#61     cmd = self.cmdTeX()
#62     subprocess.run(cmd,shell=True)
#63
#64 def CreateFolders(self,format):
#65     folderPath = self.projectFolder/'Figure'/format/self.regCo
#66         → de/self.folder
#67     Path(folderPath).mkdir(parents=True,exist_ok=True)
#68
#69
#70
#71
#72
#73
#74
#75
#76
#77
#78
#79
#80
#81
#82
#83
#84
#85
#86
#87
#88
#89
#90
#91
#92
#93
#94
#95
#96
#97
#98
#99
#100
#101
#102
#103

```

```

104         # Define the folder path and create all the missing ones if
105         # necessary
106         return folderPath
107
108     def FigPath(self,figName,ext):
109         return (getattr(self,f'_{ext[1:]}FolderPath')/figName).with_
110             _suffix(ext)
111
112     def cmdTeX(self):
113         m = self.matFilePath.as_posix()
114         t = self.texFilePath.as_posix()
115
116         cmd = (
117             r'matlab -batch "mat2tex('
118             fr"'{m}',"fr'{t}'"r")'
119         )
120         return cmd
121
122     # Primary/Basic plots
123     def CreateFunctionPlot(
124         x,y,
125         figData,
126         Ni=1,
127         ta=None,
128         yErr=True,
129         label='',
130         linestyle='--',
131         linewidth=1,
132         marker='None',
133         hatch='None',
134         color='black',
135         alpha=1,
136         idx='',
137         ax=None
138     ):
139         if ax is None: ax = plt.gca()
140
141         if Ni == 1:
142             ax.plot(
143                 x,y if y.ndim == 1 else y.ravel(),
144                 label=label,
145                 color=color,
146                 linestyle=linestyle,
147                 linewidth=linewidth,
148                 marker=marker,
149                 markevery=x.size//10-1,
150                 mfc=color,
151                 alpha=alpha
152             )
153             figData['plots'][f'functionPlot{idx}'] = {
154                 't':'function','x':x,'y':y,
155                 'l':label,'c':color,'m':marker
156             }
157
158         else:
159             if yErr:
160                 avrFunc, err95Func = EvaluateConfidenceInterval(y,ta,Ni)
161
162                 # To avoid [a possible] lower negative confidence
163                 # interval, the error has to be clipped
164                 lowerEstimate = np.maximum(avrFunc-err95Func,0)
165                 yerr95 = np.array([
166                     avrFunc-lowerEstimate,
167                     err95Func
168                 ]); xerr95 = None; y = avrFunc
169
170                 if (y-yerr95[0]).min()<0: raise("Negative values")

```

```

158         if (y+yerr95[1]).min()<0: raise("Negative values")
159
160     else:
161         avrFunc, err95Func = EvaluateConfidenceInterval(x,ta,Ni)
162
163         # To avoid [a possible] lower negative confidence
164         # interval, the error has to be clipped
165         lowerEstimate = np.maximum(avrFunc-err95Func,0)
166         xerr95 = np.array([
167             avrFunc-lowerEstimate,
168             err95Func
169         ]); yerr95 = None; x = avrFunc
170
171
172     ax.plot(
173         x,y,
174         label=label,
175         color=color,
176         linestyle=linestyle,
177         linewidth=linewidth,
178         marker=marker,
179         markevery=x.size//10-1,
180         mfc=color,
181         alpha=alpha[0],
182         zorder=4
183     )
184     figData['plots'][f'functionPlot{idx}'] = {
185         't':'function','x':x,'y':y,
186         'l':label,'c':color,'m':marker
187     }
188
189     CreateConfidenceIntervalPlot(
190         x,y,
191         figData,
192         typ='functionfill',
193         xerr95=xerr95,
194         yerr95=yerr95,
195         color=color,
196         hatch=hatch,
197         alpha=alpha[1],
198         ax=ax,
199         idx=idx
200     )
201
202
203     # mplcursors.cursor(fPlot,hover=False).connect(
204     #     "add",lambda sel: sel.annotation.set_text(
205     #         f"{{sel.target[0]:.3f},{{sel.target[1]:.3f}}}"
206     #     )
207     # )
208
209
210     def CreateScatterPlot(
211         x,y,
212         figData,
213         size=16,
214         Ni=1,
215         ta=None,
216         yErr=True,
217         label='Scatter',
218         color='black',
219         idx='',
220         ax=None,
221         alpha=1
222     ):
223         if ax is None: ax=plt.gca()
224
225         if Ni == 1:
226             sct = ax.scatter(
227                 x,y if y.ndim == 1 else y.ravel(),

```

```

234         label=label,
235         color=color,
236         s=size,
237         edgecolor='none',
238         alpha=alpha
239     )
240     figData['plots'][f'scatterPlot{idx}'] = {
241         't':'scatter','x':x,'y':y,'l':label,
242         'c':color,'a':alpha,'s':size
243     }
244 else:
245     if yErr:
246         avrSct, err95Sct = EvaluateConfidenceInterval(y,ta,Ni)
247
248         sct = ax.scatter(
249             x,
250             avrSct,
251             label=label,
252             color=color,
253             s=size,
254             edgecolor='none',
255             alpha=alpha[0],
256             zorder=5
257         )
258         figData['plots'][f'scatterPlot{idx}'] = {
259             't':'scatter','x':x,'y':avrSct,'l':label,
260             'c':color,'a':alpha[0],'s':size
261         }
262
263     CreateConfidenceIntervalPlot(
264         x,
265         avrSct,
266         figData,
267         typ='errorbar',
268         yerr95=np.array([err95Sct,err95Sct]),
269         color=color,
270         alpha=alpha[1],
271         ax=ax,
272         idx=idx
273     )
274
275 else:
276     avrSct, err95Sct = EvaluateConfidenceInterval(x,ta,Ni)
277
278     sct = ax.scatter(
279         avrSct,
280         y,
281         label=label,
282         color=color,
283         s=size,
284         edgecolor='none',
285         alpha=alpha[0],
286         zorder=5
287     )
288     figData['plots'][f'scatterPlot{idx}'] = {
289         't':'scatter','x':avrSct,'y':y,'l':label,
290         'c':color,'a':alpha[0],'s':size
291     }
292
293     CreateConfidenceIntervalPlot(
294         avrSct,
295         y,
296         figData,
297         typ='errorbar',
298         xerr95=np.array([err95Sct,err95Sct]),
299         color=color,
300         alpha=alpha[1],

```

```

301             ax=ax,
302             idx=idx
303         )
304
305     # mplcursors.cursor(sct,hover=True).connect(
306     #     "add",lambda sel: sel.annotation.set_text(
307     #         f"{{x[{sel.index}]},{y[{sel.index}]}"))
308     #     )
309     # )
310
311 def CreateHistogramPlot(
312     x,nBins,
313     figData,
314     limits=None,
315     xScale='lin',
316     Ni=1,
317     ta=None,
318     label='Histogram',
319     color='#808080',
320     norm=True,
321     alpha=1,
322     idx='',
323     ax=None
324 ):
325     def HistogramPlot(
326         binEdges,
327         binMidPoints,
328         binAverages,
329         alpha=alpha
330     ):
331         hgPlot = ax.hist(
332             binMidPoints,
333             bins=binEdges, # 'auto'
334             weights=binAverages,
335             # density=norm,
336             color=color,
337             edgecolor="none", # "black"
338             label=label,
339             alpha=alpha
340         )
341
342         figData['plots'][f'histogramPlot{idx}'] = {
343             't':'histogram','l':label,'c':color,'a':alpha,
344             'x':binMidPoints,'b':binEdges,'w':binAverages
345         }
346
347         # hgPlot[0] = heights,
348         # hgPlot[1] = bin edges,
349         # hgPlot[2] = patches (Rectangle objects)
350
351         # mplcursors.cursor(hgPlot[2],hover=False).connect(
352         #     "add",lambda sel: sel.annotation.set_text(
353         #         f"{hgPlot[0][sel.index]:.3f}")
354         #     )
355         # )
356
357     if ax is None: ax = plt.gca()
358
359     if limits is None:
360         xMin = np.min(x); xMax=np.max(x)
361     else:
362         xMin = limits[0]; xMax=limits[1]
363
364     if xScale == 'lin':
365         binPoints = np.linspace(
366             xMin,
367             xMax,

```

```

358             num=2*nBins+1
359         )
360     else:
361         binPoints = np.logspace(
362             np.log10(xMin),
363             np.log10(xMax),
364             num=2*nBins+1
365         )
366
367     binEdges = binPoints[0::2]
368     binMidPoints = binPoints[1::2]
369     # binMidPoints = (binEdges[:-1]+binEdges[1:])/2 # It only works
370     # in the linear case
371
372     if Ni == 1:
373         binAverages, _ = np.histogram(
374             x if x.ndim == 1 else x.ravel(),
375             binEdges,density=norm
376         )
377         HistogramPlot(binEdges,binMidPoints,binAverages,alpha)
378
379     return binMidPoints, binAverages
380 else:
381     hgData = [None]*Ni
382     for r in range(Ni):
383         hgData[r] = np.histogram(x[r,:],binEdges,density=norm)
384
385     binAverages, hgErr95 = EvaluateConfidenceInterval(
386         [hgData[r][0] for r in range(Ni)],ta,Ni
387     )
388
389     HistogramPlot(binEdges,binMidPoints,binAverages,alpha[0])
390
391     # To avoid [a possible] lower negative confidence interval
392     # in the lower tail, the error has to be clipped
393     lowerBinEstimate = np.maximum(binAverages-hgErr95,0)
394     binErr95 = np.array([
395         binAverages-lowerBinEstimate,
396         hgErr95
397     ])
398
399     CreateConfidenceIntervalPlot(
400         binMidPoints,
401         binAverages,
402         figData,
403         typ='errorbar',
404         yerr95=binErr95,
405         color=color,
406         ax=ax,
407         idx=idx,
408         alpha=alpha[1]
409     )
410
411     return binMidPoints, binAverages
412
413
414     # Secondary/Compound plots
415     def CreateLogRegressionPlot(
416         x,y,
417         figData,
418         l='Regression',
419         color='blue',
420         idx='',
421         ax=None
422     ):
423         if ax is None: ax=plt.gca()
424
425         xp = x[y>0]; yp = y[y>0]

```

```

43     logx = np.log10(xp); logy = np.log10(yp)
44
45     slope, intercept, _, _, _ = stats.linregress(logx,logy)
46     regression = 10** (intercept+slope*logx)
47
48     fPlot = ax.plot(
49         xp,regression,
50         label=l,
51         color=color,
52         linewidth=1
53     )
54
55     figData['plots'][f'regressionPlot{idx}'] = {
56         't':'function','x':x,'y':regression,'l':l,'c':color
57     }
58
59     # mplcursors.cursor(fPlot,hover=False).connect(
60     #     "add",lambda sel: sel.annotation.set_text(
61     #         f"{{sel.target[0]}:.3f},{{sel.target[1]}:.3f}")
62     #     )
63     #
64
65     return slope
66
67 def CreateLognormalFitPlot(
68     v,
69     figData,
70     limits=None,
71     xScale='lin',
72     Ni=1,
73     ta=None,
74     label='Lognormal fit (ML)', # Maximum likelihood
75     color='black',
76     alpha=1,
77     marker='None',
78     bimodal=False,
79     idx='',
80     ax=None
81 ):
82     if ax is None: ax=plt.gca()
83
84     if limits is None:
85         vMin = np.min(v); vMax=np.max(v)
86     else:
87         vMin = limits[0]; vMax=limits[1]
88
89     if xScale == 'lin':
90         xF = np.linspace(
91             vMin,
92             vMax,
93             1000
94         )
95     else:
96         xF = np.logspace(
97             np.log10(vMin),
98             np.log10(vMax),
99             1000
100        )
101
102     yFData = [None]*Ni
103     xiData = [None]*Ni
104
105     for r in range(Ni):
106         if bimodal:
107             xi,scale1,shape1,scale2,shape2 =
108                 FitLognormalData(v[r,:],bimodal)
109             yFData[r] =

```

```

499             xi*stats.lognorm.pdf(xF,s=shape1,scale=np.exp(scal_]
500             ↵ e1)) +
501             (1-xi)*stats.lognorm.pdf(xF,s=shape2,scale=np.exp(]
502             ↵ scale2))
503         )
504         xiData[r] = xi
505     else:
506         shape,loc,scale = FitLognormalData(v[r,:],bimodal)
507         yFData[r] =
508             ↵ stats.lognorm.pdf(xF,shape,loc=loc,scale=scale)
509         xiData[r] = 1
510
511     if Ni>1: yF = yFData; xi = xiData
512     else: yF = yFData[0]; xi = xiData[0]
513
514     # Fit plot
515     CreateFunctionPlot(
516         xF,yF,
517         figData,
518         Ni=Ni,
519         ta=ta,
520         label=label,
521         # linewidth=1,
522         color=color,
523         alpha=alpha,
524         marker=marker,
525         idx=idx,
526         ax=ax
527     )
528
529     return xi
530
531 def CreateParetoFitPlot(
532     v,
533     figData,
534     upperbound=None,
535     yScale='lin',
536     Ni=1,
537     ta=None,
538     label=(
539         'Scatter',
540         'Pareto fit (ML)' # Minimum likelihood
541     ),
542     color=(
543         'black',
544         'black'
545     ),
546     alpha=1,
547     bimodal=False,
548     idx='',
549     ax=None
550 ):
551     if ax is None: ax=plt.gca()
552
553     if upperbound is None:
554         xF = np.logspace(
555             np.log10(np.quantile(v,0.75)),
556             np.log10(v.max()),
557             100
558         )
559     else:
560         xF = np.logspace(
561             np.log10(np.quantile(v,0.75)),
562             np.log10(upperbound),
563             100
564         )

```

```

563     vTails = [None]*Ni
564     b      = [None]*Ni
565     ccdfFit = [None]*Ni
566
567     for r in range(Ni):
568         # Select the last quarter of city sizes
569         vQuarter = np.quantile(v[r,:], .75)
570         vTail = v[r,v[r,:] >= vQuarter]
571         vTails[r] = np.sort(vTail) # Ascending values
572
573         # Fitted CCDF from a Pareto function
574         b[r], loc, scale =
575             ↳ stats.pareto.fit(vTail,floc=0,fscale=vQuarter) #
576             ↳  $b \approx \alpha$ 
577         ccdfFit[r] = stats.pareto.sf(xF,b[r],loc=loc,scale=scale) #
578             ↳ Survival function
579
580         # Empirical CCDF on the tail
581         n = vTails[0].size
582         ccdfEmp = (
583             n-np.arange(1,n+1,dtype=float)
584             +(0.5 if yScale == 'log' else 0)
585         )/n #  $P(X \geq x)$ 
586         yS = ccdfEmp
587
588         """
589             The survival function is the Complementary Cumulative
590             Distribution Function (CCDF)
591
592             Formally it should be
593
594                  $1 - np.arange(1, n+1, dtype=float) / Nc - (Nc - n) / Nc =$ 
595                  $(n - np.arange(1, n+1, dtype=float)) / Nc =$ 
596                  $[(n - np.arange(1, n+1, dtype=float)) / n] / [n / Nc]$ 
597
598             hence the correct variables are:
599
600                 xS = vTails
601                 yS = (n - np.arange(1, n+1, dtype=float)) / n * (n / Nc)
602                 yF = [ccdfFit[r] * (n / Nc) for r in range(Ni)]
603
604             without the constant translation 0.5, which is just
605             ↳ necessary to avoid having a zero value at the tail end
606             ↳ in a log-log plot.
607
608             However, since the only relevant information is the Pareto
609             ↳ index, the exact probability value can be omitted:
610             ↳ indeed, by dividing (*) by [n/Nc] the CCDF is rescaled
611             ↳ in a linear scale while translated upward in a
612             ↳ logarithmic one; both transformations do not affect the
613             ↳ actual trend of the CCDF from which the Pareto index is
614             ↳ calculated
615
616             """
617
618     if Ni>1:
619         xS = vTails
620         yF = ccdfFit
621     else:
622         xS = vTails[0]
623         yF = ccdfFit[0]
624         b = b[0]
625
626     CreateScatterPlot(
627         xS, yS,
628         figData,
629         Ni=Ni,
630         ta=ta,

```

```

618         size=10,
619         yErr=False,
620         label=label[0],
621         color=color[0],
622         alpha=alpha[0],
623         idx=idx,
624         ax=ax
625     )
626
627     CreateFunctionPlot(
628         xF,yF,
629         figData,
630         Ni=Ni,
631         ta=ta,
632         label=label[1],
633         color=color[1],
634         alpha=alpha[1],
635         marker='+' if bimodal else 'None',
636         hatch='|' if bimodal else 'None',
637         idx=idx,
638         ax=ax
639     )
640
641     if bimodal:
642         xFl = np.log(xF)
643         yFData = [None]*Ni
644         Nc = v.shape[1]
645
646         for r in range(Ni):
647             xi,loc1,shape1,loc2,shape2 = FitLognormalData(v[r,:] if
648                 ↳ Ni>1 else v,bimodal)
649
650             ccdfFit1 = stats.norm.sf(xFl,loc=loc1,scale=shape1)
651             ccdfFit2 = stats.norm.sf(xFl,loc=loc2,scale=shape2)
652
653             yFData[r] = xi*ccdfFit1+(1-xi)*ccdfFit2
654             # yFData[r] /= yFData[r][0] # Normalization
655             yFData[r] /= n/Nc # Rescaling
656             # It's necessary to effectively compare it with the
657             ↳ Pareto fitting
658
659             yF = yFData if Ni>1 else yFData[0]
660
661             CreateFunctionPlot(
662                 xF,yF,
663                 figData,
664                 Ni=Ni,
665                 ta=ta,
666                 label=label[2],
667                 color=color[1],
668                 alpha=alpha[1],
669                 marker='x',
670                 hatch='/',
671                 idx=idx+1,
672                 ax=ax
673             )
674
675     return b
676
677
678     ### Auxiliary functions ###
679
680     def SetTextStyle():
681         plt.rcParams["mathtext.fontset"] = "stix"
682         plt.rcParams["font.family"] = "serif"
683         plt.rcParams["font.serif"] = ["Times New Roman"]
684         plt.rcParams["font.size"] = 13

```

```

63 SetTextStyle()
64
65 def SetFigStyle(
66     xLabel=None,
67     yLabel=None,
68     xDom=None,
69     yDom=None,
70     xScale='lin',
71     yScale='lin',
72     xNotation='plain',
73     yNotation='plain',
74     ax=None,data=None
75 ):
76     if ax is None: ax = plt.gca()
77
78     if xLabel: ax.set_xlabel(xLabel)
79     if yLabel: ax.set_ylabel(yLabel)
80
81     if xDom: ax.set_xlim(xDom)
82     if yDom: ax.set_ylim(yDom)
83
84     ax.set_xscale('linear' if xScale == 'lin' else xScale)
85     ax.set_yscale('linear' if yScale == 'lin' else yScale)
86
87     if xScale == 'lin':
88         ax.ticklabel_format(style=xNotation, axis='x', scilimits=(0, ]
89                               ↵ 0))
90     if yScale == 'lin':
91         ax.ticklabel_format(style=yNotation, axis='y', scilimits=(0, ]
92                               ↵ 0))
93
94     ax.grid(True, linestyle=":", linewidth=1)
95     ax.set_axisbelow(True)
96
97     _, labels = ax.get_legend_handles_labels()
98     if any(labels):
99         ax.legend()
100        data['style']['legend'] = True
101    else:
102        data['style']['legend'] = False
103    # Create a legend iff there are labels connected to graphs
104
105    data['style']['scale'] = {'x':xScale, 'y':yScale}
106    data['style']['labels'] = {'x':xLabel, 'y':yLabel}
107
108 def CentreFig():
109     fig = plt.gcf()
110     manager = plt.get_current_fig_manager()
111     manager.window.update_idletasks()
112
113     # Get screen size
114     screenW = manager.window.winfo_screenwidth()
115     screenH = manager.window.winfo_screenheight()
116
117     # Save figure size in pixels
118     figW, figH = fig.get_size_inches()*fig.dpi
119
120     # Centre coordinates
121     x = int((screenW-figW)/2)
122     y = int((screenH-figH)/2)
123
124     # Move the figure window
125     manager.window.geometry(f"+{x}+{y}")
126
127 def EvaluateConfidenceInterval(
128     data,
129     ta,
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147

```

```

748     Ni
749 ):      if Ni>1:
750         values = np.array(data)
751         averages = np.mean(values, axis=0)
752         err95 = ta*np.std(values, axis=0, ddof=1)/np.sqrt(Ni)
753         return (averages,err95)
754     else:
755         if isinstance(data,np.ndarray):
756             return (data[0],None)
757         else:
758             return (data,None)
759
760 def CreateConfidenceIntervalPlot(
761     x,y,
762     figData,
763     typ='errorbar',
764     yerr95=None,
765     xerr95=None,
766     color='black',
767     fmt='none',
768     hatch='None',
769     alpha=0.5,
770     ax=None,
771     idx=''
772 ):
773     if ax is None: ax = plt.gca()
774
775     match typ:
776         case 'errorbar':
777             ax.errorbar(
778                 x,y,
779                 xerr=xerr95,
780                 yerr=yerr95,
781                 ecolor=color,
782                 fmt=fmt,
783                 linestyle='none',
784                 elinewidth=1.2,
785                 capsizes=8,
786                 capthick=1.6,
787                 markersize=0,
788                 markerfacecolor=color,
789                 markerfacecoloralt=color,
790                 markeredgecolor=color,
791                 markeredgewidth=0,
792                 alpha=alpha,
793                 zorder=1
794             )
795         if xerr95 is None:
796             figData['plots'][f'{typ}{idx}'] = {
797                 't':typ,'x':x,'y':y,
798                 'e':'y','ye':yerr95,
799                 'l':'','c':color,'a':alpha,
800             }
801         else:
802             figData['plots'][f'{typ}{idx}'] = {
803                 't':typ,'x':x,'y':y,
804                 'e':'x','xe':xerr95,
805                 'l':'','c':color,'a':alpha,
806             }
807
808         case 'functionfill':
809             if xerr95 is None:
810                 ax.fill_between(
811                     x,y-yerr95[0],y+yerr95[1],
812                     facecolor=color,
813                     alpha=alpha,
814

```

```

815     linewidth=0,
816     hatch= None if hatch=='None' else hatch,
817     edgecolor= None if hatch=='None' else color,
818     zorder=2
819   )
820   figData['plots'][f'{typ}{idx}'] = {
821     't':typ,'x':x,'y':y,
822     'e':'y','ye':yerr95,
823     'l':'','c':color,
824     'a':alpha,'h':hatch,
825   }
826 else:
827   ax.fill_betweenx(
828     y,x-xerr95[0],x+xerr95[1],
829     facecolor=color,
830     alpha=alpha,
831     linewidth=0,
832     hatch= None if hatch=='None' else hatch,
833     edgecolor= None if hatch=='None' else color,
834     zorder=2
835   )
836   figData['plots'][f'{typ}{idx}'] = {
837     't':typ,'x':x,'y':y,
838     'e':'x','xe':xerr95,
839     'l':'','c':color,
840     'a':alpha,'h':hatch,
841   }
842
843 def FitLognormalData(v,bimodal):
844   if not bimodal:
845     shape,loc,scale = stats.lognorm.fit(v[v>0],floc=0)
846     # The average is »μ=np.log(scale)« while the standard
847     → deviation is »σ=shape«
848     return shape,loc,scale
849   else:
850     vp = v[v>0]
851     vl = np.log(vp)
852
853     # The objective function to minimize is the negative
854     → log-likelihood
855     # This is equivalent to maximize the log-likelihood, just
856     → reversed
857     def NegativeLogLikelihood(params):
858       # xi is fraction between Gaussian
859       # mi and si are the mean and standard deviation of the
860       → i-th Gaussian
861       xi,m1,s1,m2,s2 = params
862
863       # Calculate Gaussian PDF for both components on log data
864       pdf1 = stats.norm.pdf(vl,loc=m1,scale=s1)
865       pdf2 = stats.norm.pdf(vl,loc=m2,scale=s2)
866
867       # Mixture sum
868       mixture = xi*pdf1+(1-xi)*pdf2
869
870       # Prevent log(0) errors
871       epsilon = 1e-10
872     return -np.sum(np.log(mixture + epsilon))
873
874   # Initial heuristic guesses
875   m0 = np.mean(vl)
876   s0 = np.std(vl)
877   prm0 = [
878     0.8, # The first gaussian is initially the most
879     → important one
880     m0,s0,
881     m0+0.5,s0
882   ]

```

```

877         ] # Same standard deviation but translated average
878
879     bounds = ( #  $x_i$  must be in [0,1]
880         (0,1), #  $x_i$ 
881         (None,None), #  $m_1$ 
882         (1e-5,None), #  $s_1$ 
883         (None,None), #  $m_2$ 
884         (1e-5, None) #  $s_2$ 
885     ) #  $m_i$  does not have a bound while  $s_i > 0$ 
886
887     # Optimization
888     minimization =
889     ↪ minimize(NegativeLogLikelihood,prm0,bounds=bounds)
890
891     # Extract results
892     xi,m1,s1,m2,s2 = minimization.x
893
894     return xi,m1,s1,m2,s2
895
896 def DataString(
897     data,
898     Ni=1,
899     ta=None,
900     head='',
901     formatVal='.2f',
902     formatErr='.2f',
903     space=True
904 ):
905     (value,error) = EvaluateConfidenceInterval(data,ta,Ni)
906
907     space = r'\quad' if space else ''
908     if error is None:
909         return fr'{head}={value:{formatVal}}{space}$'
910     else:
911         return fr'{head}={value:{formatVal}}\pm{error:{formatErr}}{space}$'
912
913 def Text(
914     target,
915     pos,
916     string,
917     ha='center',
918     color=None
919 ):
920     if hasattr(target,"transAxes"):
921         target.text(
922             pos[0],
923             pos[1],
924             string,
925             color=color,
926             ha=ha,
927             transform=target.transAxes
928         )
929     else:
930         target.text(
931             pos[0],
932             pos[1],
933             string,
934             color=color,
935             ha=ha
936         )
937
938 def TextBlock(
939     ax,
940     list,
941     p=(0,0),
942     dp=(0,0),
943

```

```

91     offset=0,
92     **kwargs
93   ):
94     nR = len(list); nC = len(list[0])-offset
95     for r,y in enumerate(np.linspace(p[1]+dp[1]/2,p[1]-dp[1]/2,nR)):
96       for c,x in enumerate(
97         np.linspace(p[0]-dp[0]/2,p[0]+dp[0]/2,nC),
98         start=offset
99       ):
100         Text(ax,(x,y),list[r][c],**kwargs)
101         # x moves from left to right
102         # y moves from top to bottom
103
104     # An alternative is to do what linspace does manually with
105     # → «x0+(i-(n-1)/2)*dx» where dx is actually the space between
106     # → strings rather than the block length
107
108   Listato A.5: Codice «libNetworks.py».

1  # Library to analyse networks [also known as graphs]
2
3 import numpy as np
4 import networkx as nx
5
6 import libFigures as libF
7
8
9  ### Main class ###
10
11 class NetworkAnalysis():
12   def __init__(self,clsPrm,clsReg):
13     self.li2Name = clsReg.li2Name
14     self.name2li = clsReg.name2li
15     self.Nc = clsReg.Nc
16
17     A = clsReg.A; self.A = A
18     W = clsReg.W; self.W = W
19
20     # Degrees
21     di = np.sum(A,axis=0); self.di = di # Degree vector
22     self.dk, self.Nk = np.unique(di,return_counts=True)
23     # Unique degrees and corresponding frequencies
24     # Pk = counts/N
25
26     # [Nonzero] Weights
27     wi = W[W>0]; self.wi = wi
28     self.wk, self.wNk = np.unique(wi,return_counts=True)
29     # Unique weights and corresponding frequencies
30
31     # Strengths
32     si = np.sum(W,axis=0); self.si = si
33     self.sk, self.sNk = np.unique(si,return_counts=True)
34     # Unique strengths and corresponding frequencies
35
36     self.figData = libF.FigData(clsPrm,'NA')
37
38   def DegreeDistributionFig(self):
39     li2Name = self.li2Name
40     di = self.di
41
42     figData = self.figData
43     fig = figData.SetFigs()
44
45
46     kAvr = np.mean(di)
47     libF.TextBlock(
48       fig,[
49         libF.DataString(di.size,head='N',space=False,form_
49           → atVal='')],

```

```

50             [libF.DataString(int(np.sum(di)/2),head='E',space=]
51             ↵ False,formatVal='')], 
52             [libF.DataString(np.min(di),head=r'k_{min}',space=
53             ↵ e=False,formatVal='')], 
54             [libF.DataString(np.max(di),head=r'k_{max}',space=
55             ↵ e=False,formatVal='')], 
56             [libF.DataString(kAvr,head=r'\langle k\rangle',
57             ↵ \rangle',space=False)]
58         ],(0.8,0.4),
59         (0,0.15)
60     )
61
62
63     kis = np.argsort(di) # Vector for the sorted degrees
64     kit = [['City','k']] # Table for the sorted degrees
65     for i in range(10):
66         li = kis[-i-1]
67         kit.append([li2Name[li],di[li]])
68     libF.TextBlock(fig,kit,(1.175,0.5),(0.2,0.4))
69
70
71
72     # Histogram plot
73     def EstimateBinNumber(v):
74         # The option «'fd'» stands for «Freedman-Diaconis» and
75         # uses Numpy to calculate the optimal edges for the data
76         edges = np.histogram_bin_edges(v,bins='fd')
77         return len(edges) - 1
78     libF.CreateHistogramPlot(di,EstimateBinNumber(di),figData.)
79     ↵ fig)
80
81     #region
82         # Scatter plot
83         # scP = plt.scatter(k,Pk,label="Empirical",s=10)
84
85         # mplcursors.cursor(scP,hover=False).connect(
86         #     "add",lambda sel: sel.annotation.set_text(
87         #         f"k={k[sel.index]}, P(k)={Pk[sel.index]:.3f}"
88         #     )
89         # )
90
91
92
93         # # Manual fitting (ML)
94
95         # # Estimators
96         # m = (1/N)*np.sum(np.log(d)) # Estimated mean
97         # s2 = (1/N)*np.sum((np.log(d)-m)**2)
98         # s = np.sqrt(s2) # Estimated standard deviation
99
100
101         # # Evaluation
102         # def lognormalPDF(x,m,s):
103         #     y = np.zeros_like(x)
104         #     v = x>0
105         #     C = 1/(x[v]*s*np.sqrt(2*np.pi))
106         #     y[v] = C*np.exp(-(np.log(x[v])-m)**2/(2*s**2))
107         #     return y
108
109         # # Plotting
110         # scF = plt.plot(
111         #     x,lognormalPDF(x,m,s),
112         #     label="Manual lognormal fit (ML)", # Maximum
113         #     ↵ likelyhood
114         #     color="red",
115         #     linewidth=1
116         # )
117     #endregion
118
119
120

```

```

11      # SciPy fitting (ML)
12      libF.CreateLognormalFitPlot(di.reshape(1,-1),figData.fig)
13
14
15      # Style
16      # libF.CentreFig()
17      libF.SetFigStyle(
18          r"$k$",
19          r"$P(k)$",
20          # [0,300],[0,0.03],
21          data=figData.fig
22      )
23      figData.SaveFig('DegreeDistribution')
24
25  def WeightDistributionFig(self):
26      W = self.W
27      wi = self.wi
28      Nc = self.Nc
29
30      li2Name = self.li2Name
31      name2li = self.name2li
32
33      figData = self.figData
34      fig = figData.SetFigs()
35
36      # Histogram plot
37      binw, binPw = libF.CreateHistogramPlot(wi,19,figData.fig,x_]
38      ↳ Scale='log')
39
40      # SciPy regression
41      # binw = (hgPlot[1][1:]+hgPlot[1][-1])/2
42      # binPw = hgPlot[0]
43      # sc = plt.scatter(binw,binPw,c='r',s=50)
44
45      # Fit in log-log space
46      slope = libF.CreateLogRegressionPlot(binw,binPw,figData.fig)
47
48      kAvr = np.mean(wi)
49      libF.TextBlock(
50          fig,[
51              [libF.DataString(np.min(wi),head=r'w_{min}',space=_]
52              ↳ False,formatVal='')],_
53              [libF.DataString(np.max(wi),head=r'w_{max}',space=_]
54              ↳ False,formatVal='')],_
55              [libF.DataString(kAvr,head=r'\langle w\rangle',space=_]
56              ↳ formatVal='3f')],_
57              [libF.DataString(slope,head=r'\alpha',space=False,]
58              ↳ formatVal='3f')],_
59              [(0.75,0.5),_
60              (0,0.1)])
61
62
63      iWu,jWu = np.triu_indices_from(W,k=1)
64      values = W[iWu,jWu]
65      ls = np.argsort(values)[-5:][::-1] # Vector or sorted links
66
67      Wt = [['Connection [extracted]', 'w']]
68      for l in ls:
69          lir = iWu[l]; lic = jWu[l]
70          Wt.append([f'{li2Name[lir]}-{li2Name[lic]}',W[lir,lic]])
71      libF.TextBlock(fig,Wt,(1.3,0.65),(0.3,0.2))
72
73
74      Wt = [['Connection [reference]', 'w']]
75      for link in [

```

```

173         ['CAGLIARI','SASSARI'],
174         ['SASSARI','OLBIA'],
175         ['CAGLIARI','ASSEMINI'],
176         ['PORTO TORRES','SASSARI'],
177         ['CAGLIARI','CAPOTERRA']
178     ]:
179         lir = name2li[link[0]]
180         lic = name2li[link[1]]
181         Wt.append([f'{li2Name[lir]}-{li2Name[lic]}',W[lir,lic]])
182         libF.TextBlock(fig,Wt,(1.3,0.35),(0.3,0.2))
183
184
185     # Style
186     # libF.CentreFig()
187     libF.SetFigStyle(
188         r"$w$",
189         xScale="log",yScale="log",
190         data=figData.fig
191     )
192     figData.SaveFig('WeightDistribution')
193
194 def StrengthDistributionFig(self):
195     li2Name = self.li2Name
196     si = self.si
197
198     figData = self.figData
199     fig = figData.SetFigs()
200
201
202     # Histogram plot
203     bins, binPs = libF.CreateHistogramPlot(si,20,figData.fig,xj
204         ↳ Scale='log')
205
206     # SciPy regression
207     # bins = (hgPlot[1][1:]+hgPlot[1][-1:])/2
208     # binPs = hgPlot[0]
209     # sc = plt.scatter(binw,binPw,c='r',s=50)
210
211     # Fit in log-log space
212     v = binPs>0; v[:6] = 0
213     slope = libF.CreateLogRegressionPlot(bins[v],binPs[v],figD
214         ↳ ata.fig)
215
216     kAvr = np.mean(si)
217     libF.TextBlock(
218         fig,[
219             [libF.DataString(np.min(si),head=r's_{min}',space=j
220                 ↳ False,formatVal='')],,
221             [libF.DataString(np.max(si),head=r's_{max}',space=j
222                 ↳ False,formatVal='')],,
223             [libF.DataString(kAvr,head=r'\langle
224                 ↳ s \rangle',space=False,formatVal='.3f')],,
225             [libF.DataString(slope,head=r'\alpha',space=False,j
226                 ↳ formatVal='.3f')]]]
227         ),(0.75,0.5),
228         (0,0.1)
229     )
230
231     sis = np.argsort(si) # Vector for the sorted strengths
232     sit = [['City','s']] # Table for the sorted strengths
233     for i in range(10):
234         li = sis[-i-1]
235         sit.append([li2Name[li],si[li]])

```

```

233     libF.TextBlock(fig,sit,(1.175,0.5),(0.2,0.4))
234
235
236     # Style
237     # libF.CentreFig()
238     libF.SetFigStyle(
239         r"$s$",
240         xScale="log",yScale="log",
241         data=figData.fig
242     )
243     figData.SaveFig('StrengthDistribution')
244
245     def BetweennessCentralityFig(self):
246         A = self.A
247         di = self.di
248
249         figData = self.figData
250         fig = figData.SetFigs()
251
252         G = nx.from_numpy_array(A)
253         bc = nx.betweenness_centrality(G,normalized=False)
254         bcd = np.array([bc[i] for i in range(len(di))])
255
256         aAvr = np.mean(list(bc.values()))
257         libF.TextBlock(
258             fig,[
259                 libF.DataString(np.min(list(bc.values())),head=r'↓
260                 ↳ g_{min}',space=False,formatVal='.3f')],
261                 libF.DataString(np.max(list(bc.values())),head=r'↑
262                 ↳ g_{max}',space=False,formatVal='.0f'),
263                 libF.DataString(aAvr,head=r'\langle
264                 ↳ g \rangle',space=False,formatVal='.3f')]
265             ],(0.75,0.25),
266             (0,0.075)
267         )
268
269         libF.CreateScatterPlot(
270             di,bcd,
271             self.figData.fig,
272             label=''
273         )
274
275         # Style
276         # libF.CentreFig()
277         libF.SetFigStyle(
278             r"$k$",
279             r"$g(i)$",
280             # [0.5e1,0.5e3],[1,0.5e5],
281             xScale='log',yScale='log',
282             data=self.figData.fig
283         )
284         self.figData.SaveFig('BetweennessCentrality')
285
286     def StrengthVsDegreeFig(self):
287         si = self.si
288         di = self.di
289         dk = self.dk
290         Nk = self.Nk
291
292         figData = self.figData
293         fig = figData.SetFigs()
294
295         # Scatter
296         sk = np.zeros_like(dk,dtype=float)
297         for i,ki in enumerate(dk):
298             v = np.nonzero(di == ki)[0]

```

```

297         sk[i] = np.sum(si[v])
298         sk[i] /= Nk[i]
299
300     libF.CreateScatterPlot(dk,sk,figData.fig)
301
302     # Fit in log-log space
303     slope = libF.CreateLogRegressionPlot(dk,sk,figData.fig)
304
305     sAvr = np.mean(si)
306     libF.TextBlock(
307         fig,[
308             [libF.DataString(np.min(si),head=r's_{min}',space=_
309             ↵ False,formatVal='')],
310             [libF.DataString(np.max(si),head=r's_{max}',space=_
311             ↵ False,formatVal='')],
312             [libF.DataString(sAvr,head=r'\langle
313             ↵ s \rangle',space=False,formatVal='.3f')],
314             [libF.DataString(slope,head=r'\alpha',space=False,_
315             ↵ formatVal='.3f')]
316         ],(0.75,0.25),
317         (0,0.1)
318     )
319
320
321     # Style
322     # libF.CentreFig()
323     libF.SetFigStyle(
324         r"$k$",
325         r"$s(k)$",
326         xScale="log",yScale="log",
327         data=figData.fig
328     )
329     figData.SaveFig('StrengthVsDegree')
330
331 def AClusteringCoefficientFig(self):
332     Nc = self.Nc
333     A = self.A
334     di = self.di
335     dk = self.dk
336     Nk = self.Nk
337
338     figData = self.figData
339     fig = figData.SetFigs()
340
341     G = nx.from_numpy_array(A)
342     C = nx.clustering(G)
343     Cd = np.array([C[i] for i in range(Nc)])
344
345
346     Ck = np.zeros_like(dk,dtype=float)
347     for i,ki in enumerate(dk):
348         v = di == ki
349         Ck[i] = np.sum(Cd[v])/Nk[i]
350     self.Ck = Ck
351
352
353     #region
354     # # Manual counting
355     # Cd = np.zeros_like(d,dtype=float)
356     # for i,ki in enumerate(d):
357     #     if ki>1: # Consider only nodes with more than 2
358     #         neighbours
359     #             vi = A[i,:]
360     #             indeces = np.nonzero(vi)[0]
361
362     #             Ei = 0
363     #             for n in indeces:
364     #                 for m in indeces:
365     #                     if A[n,m] == 1:

```

```

359             #            $Ei += 1$ 
360             #            $Ei /= 2$  # Divided by 2 since each edge is
361             #           counted twice
362
363             #            $Cd[i] = 2*Ei/(ki*(ki-1))$ 
364             #           else:
365             #            $Cd[i] = 0$ 
366
367             #            $Ck = np.zeros_like(k, dtype=float)$ 
368             #           for i, ki in enumerate(k):
369             #               v = (d == ki)
370             #               Nk = np.sum(v)
371
372             #            $Ck[i] = np.sum(Cd[v])/Nk$ 
373         endregion
374
375         # CAvg = nx.average_clustering(G)
376         CAvg = Ck.mean()
377         libF.TextBlock(
378             fig,
379                 [libF.DataString(np.min(Cd), head=r'C_{min}', space=_]
380                  ↵ False, formatVal='3f')],
381                 [libF.DataString(np.max(Cd), head=r'C_{max}', space=_]
382                  ↵ False, formatVal='3f')],
383                 [libF.DataString(CAvg, head=r'\langle C \rangle',
384                  ↵ r' \rangle', space=False, formatVal='3f')]
385             ], (0.75, 0.75),
386             (0, 0.075)
387         )
388
389         libF.CreateScatterPlot(dk, Ck, figData.fig, label='')
390
391         # Style
392         # libF.CentreFig()
393         libF.SetFigStyle(
394             r"$k$",
395             r"$C(k)$",
396             # [0, 300], [0, 0.8],
397             data=figData.fig
398         )
399         figData.SaveFig('AClusteringCoefficient')
400
401     def WClusteringCoefficientFig(self):
402         A = self.A
403         W = self.W
404
405         si = self.si
406         di = self.di
407         dk = self.dk
408         Nk = self.Nk
409         Ck = self.Ck
410
411
412         figData = self.figData
413         fig, ax = figData.SetFigs(2)
414
415
416         # Manual counting
417         Cd = np.zeros_like(di, dtype=float)
418         for i, ki in enumerate(di):
419             if ki>1: # Consider only nodes with more than 2
420                 ↵ neighbours
421                 vi = A[i,:]
422                 indices = np.nonzero(vi)[0]
423
424                 Ei = 0
425                 for n in indices:
426                     for m in indices:
427

```

```

421                 Ei += (W[i,n]+W[i,m])*A[n,m]
422                 Ei /= 2 # Divided by 2 since each edge is counted
423                 ↵ twice
424
425                 Cd[i] = Ei/(si[i]*(ki-1))
426             else:
427                 Cd[i] = 0
428
429
430                 Ckw = np.zeros_like(dk,dtype=float)
431                 for i, ki in enumerate(dk):
432                     v = di == ki
433                     Ckw[i] = np.sum(Cd[v])/Nk[i]
434
435                 libF.CreateScatterPlot(dk,Ckw,figData.fig1,label='',ax=ax[
436                     ↵ 0])
437
438                 # Style
439                 libF.SetFigStyle(
440                     r"$k$",
441                     r"$C^w(k)$",
442                     xScale="log",
443                     ax=ax[0],
444                     data=figData.fig1
445                 )
446
447
448                 CkwRel= (Ckw-Ck)/Ck
449                 libF.CreateScatterPlot(dk,CkwRel,figData.fig2,label='',ax=
450                     ↵ ax[1])
451                 self.CkwRel = CkwRel
452
453                 # Style
454                 libF.SetFigStyle(
455                     r"$k$",
456                     r"$C^w_{\text{rel}}(k)$",
457                     xScale="log",yScale="log",
458                     ax=ax[1],
459                     data=figData.fig2
460                 )
461
462
463                 # libF.CentreFig()
464                 figData.SaveFig('WClusteringCoefficient')
465
466             def AAssortativityFig(self):
467                 A = self.A
468                 dk = self.dk
469
470                 figData = self.figData
471                 fig = figData.SetFigs()
472
473                 G = nx.from_numpy_array(A)
474                 ad = nx.average_neighborhood_degree(G)
475                 ak = nx.average_connectivity(G)
476
477                 aAvr = np.mean(list(ad.values()))
478                 libF.TextBlock(
479                     fig,[
480                         libF.DataString(np.min(list(ad.values())),head=r'[
481                             ↵ k_{nn}^{min}',space=False,formatVal='.
482                             ↵ 3f')],
483                         libF.DataString(np.max(list(ad.values())),head=r'[
484                             ↵ k_{nn}^{max}',space=False,formatVal='.
485                             ↵ 3f']),
486                         libF.DataString(aAvr,head=r'\langle k_{nn}\rangle [
487                             ↵ ',space=False,formatVal='.
488                             ↵ 3f'])]
489                         ,(0.75,0.75),
490                         (0,0.075)
491                     ]
492                 )
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779

```

```

452         knn = np.array([ak[ki] for ki in dk])
453         libF.CreateScatterPlot(dk,knn,figData.fig,label='')
454         self.knn = knn
455
456
457         # Style
458         # libF.CentreFig()
459         libF.SetFigStyle(
460             r"$k$",
461             r"$k_{\{nn\}}(k)$",
462             # [0,300],[40,95],
463             data=figData.fig
464         )
465         figData.SaveFig('AAssortativity')
466
467     def WAssortativityFig(self):
468         W = self.W
469         dk = self.dk
470         knn = self.knn
471
472         figData = self.figData
473         fig, ax = figData.SetFigs(2)
474
475         Gw = nx.from_numpy_array(W)
476         # adw = nx.average_neighbor_degree(Gw,weight="weight")
477         akw = nx.average_degree_connectivity(Gw,weight="weight")
478
479
480         knnw = np.array([akw[ki] for ki in dk])
481         libF.CreateScatterPlot(dk,knnw,figData.fig1,label='',ax=ax]
482             ↳ [0])
483
484         # Style
485         libF.SetFigStyle(
486             r"$k$",
487             r"$k_{\{nn\}}^w(k)$",
488             xScale="log",yScale="log",
489             ax=ax[0],
490             data=figData.fig1
491         )
492
493
494         knnwRel = (knnw-knn)/knn
495         libF.CreateScatterPlot(dk,knnwRel,figData.fig2,label='',ax]
496             ↳ =ax[1])
497
498         # Style
499         libF.SetFigStyle(
500             r"$k$",
501             r"$k_{\{nn,rel\}}^w(k)$",
502             xScale="log",yScale="log",
503             ax=ax[1],
504             data=figData.fig2
505         )
506
507
508         # libF.CentreFig()
509         figData.SaveFig('WAssortativity')
510
511     def ShowFig(self):
512         from matplotlib.pyplot import show
513         show()

```

Listato A.6: Codice «libKTMAS.py».

```

1  # Library to apply the Kinetic Theory for Multi-Agent Systems
2
3  from dataclasses import dataclass
4
5  from concurrent.futures import ProcessPoolExecutor
6  import multiprocessing as mp

```

```

7   from multiprocessing import shared_memory
8   import time
9   # from tqdm import tqdm
10
11  import numpy as np
12  from scipy import stats
13  from numba import njit
14
15  from matplotlib.pyplot import get_cmap
16  from matplotlib.colors import LogNorm
17
18  from libData import WriteSimulationData
19  from libParameters import CopyWorkerShMTTemplate
20  workersShM = CopyWorkerShMTTemplate()
21  import libFigures as libF
22
23
24  ### Main classes ###
25
26  class KineticSimulation():
27      def __init__(self,clsPrm,clsReg):
28          self.il = int(clsPrm.interactingRule)
29
30          self.l = float(clsPrm.attractivity)
31          self.a = float(clsPrm.convincibility)
32          self.s = clsPrm.fluctuations*float(clsPrm.deviation)
33          self.z = clsPrm.zetaFraction*float(clsPrm.zetaValue)
34
35          self.Ni = int(clsPrm.iterations)
36          Nc = int(clsReg.Nc); self.Nc = Nc
37          self.li2Name = clsReg.li2Name
38
39          self.progressBar = clsPrm.progressBar
40          dt = float(clsPrm.timestep); self.dt = dt
41          Nt = int(clsPrm.timesteps); self.Nt = Nt
42
43          Ns = int(clsPrm.snapshots); self.Ns = Ns
44          ns = np.array(
45              [i*Nt/Ns for i in range(Ns+1)],
46              dtype=np.int64
47          ); self.ns = ns
48
49          Nw = clsPrm.smoothingFactor; self.Nw = Nw
50          ks = int(Ns/Nw); self.ks = ks # Kernel size
51          self.times = ns[:ks]*dt
52
53          self.R = int(clsPrm.region)
54          self.P = int(clsPrm.population)
55          self.p0 = float(self.P/self.Nc)
56          self.realSizeDistr = clsReg.sizeDistr
57
58          # Exact unitary adjacency matrix
59          M = clsReg.A
60          self.dk = np.sum(M, axis=1, dtype=np.int32)
61          self.Mdt = M*dt
62
63          # Approximated adjacency matrix
64          Mn = np.sum(M)
65          self.w0dt = np.sum(M[:, :, :], axis=1)*dt/Mn
66          self.wI = np.sum(M[:, :, :], axis=0)
67          # M[:, :, 1] = w0@wI/Mn
68          # In this case w0=wI but it's better to define them
69          # → rigorously
70
71          self.ew = clsPrm.edgeWeights
72          if clsPrm.edgeWeights:

```

```

72         self.di = np.sum(clsReg.W/np.max(clsReg.W),axis=1,dtype=np.float64)
73     else:
74         self.di = np.sum(clsReg.A,axis=1,dtype=np.int32)
75     self.idi = np.array(1.0/self.di,dtype=np.float64) # Inverse
76     # self.D = di@invdi.T
77
78     self.typ = np.array([0,1],dtype=np.int64)
79     self.lblEn = ('Ext.', 'Apx.', 'Real')
80     self.lblIt = (
81         ('esatto', 'appr.', 'reale'),
82         ('esatta', 'appr.', 'reale'),
83         ('Esatto', 'Appr.', 'Reale'),
84     )
85     self.clr = ('#0072B2', '#D55E00', '#000000')
86     # self.clr = ('blue', 'red')
87
88     if clsPrm.parametricStudy:
89         self.figData = None
90         self.Nv = clsPrm.numberPrmStudy
91         self.studiedPrmString = None
92     else:
93         self.figData = libF.FigData(clsPrm,'KS')
94         self.Nv = None
95         self.fw = 8 # Figure width
96         self.fh = 6 # Figure height
97         self.sid = None
98
99     # Simulation
100    def MonteCarloSimulation(self):
101        Ni = self.Ni
102        Nc = self.Nc
103        p0 = self.p0
104
105        gui = self.progressBar
106        Nt = self.Nt
107        Ns = self.Ns
108
109        sid = self.sid
110        Nv = self.Nv
111
112        l = self.l
113        a = self.a
114        s = self.s
115        z = self.z
116        il = self.il
117        typ = self.typ.size
118
119        process = range(Ni)
120
121        shmPrm = {}
122        shmHandles = {}
123
124        for key in workersShM['parameters'].keys():
125            spec, shm = BuildShM(getattr(self,key))
126            shmPrm[key] = spec
127            shmHandles[key] = shm
128
129        ctx = mp.get_context("spawn")
130
131        if gui:
132            for key,dtype in workersShM['gui'].items():
133                spec, shm = BuildShM(Ni=Ni,dtype=dtype)
134                shmPrm[key] = spec
135                shmHandles[key] = shm

```

```
137         # Import of ProgressBarsGUI locally [and not at the top
138         # of the module] to avoid freezing the GUI when in a
139         # parametric study (Tk must live only in the parent
140         # thread but spawn under Windows imports it in each
141         # worker [if written at the top of the module] making
142         # it prone to errors)
143         from libGUIs import ProgressBarsGUI
144         bar = ProgressBarsGUI(Ni,Nt,sid,Nv,shmPrm,LoadShM)
145
146     try:
147         with ProcessPoolExecutor(
148             max_workers=3,
149             mp_context=ctx,
150             initializer=InitializeMCSWorker,
151             initargs=(shmPrm, True))
152             ) as executor:
153                 futures = {}
154                 for p in range(Ni):
155                     futures[p] = executor.submit(
156                         MonteCarloAlgorithm,
157                         p,Nc,Ns,p0,typ,
158                         l,a,s,z,il/gui
159                     )
160                 bar.mainloop()
161
162     finally:
163         data = [None]*Ni
164         for p, fut in futures.items():
165             data[p] = fut.result()
166         self.data = data
167
168         for shm in shmHandles.values():
169             shm.close(); shm.unlink()
170
171     else:
172         try:
173             with ProcessPoolExecutor(
174                 max_workers=3,
175                 mp_context=ctx,
176                 initializer=InitializeMCSWorker,
177                 initargs=(shmPrm,))
178                 ) as executor:
179                     data = list(
180                         executor.map(
181                             MonteCarloAlgorithm,
182                             process,[Nc]*Ni,[Ns]*Ni,
183                             [p0]*Ni,[typ]*Ni,[l]*Ni,
184                             [a]*Ni,[s]*Ni,[z]*Ni,
185                             [il]*Ni,[gui]*Ni
186                         )
187                     ); self.data = data
188     finally:
189         for shm in shmHandles.values():
190             shm.close(); shm.unlink()
191
192     self.EvaluateSimulationData()
193     WriteSimulationData(
194         self.vrtState,
195         self.snapshots,
196         self.siVrtState,
197         self.typ,
198         self.lblEn,
199         self.li2Name,
200         Ni,
201         self.sid
202     )
203
```

```

199     def EvaluateSimulationData(self):
200         Ni = self.Ni
201         data = self.data
202         Nw = self.Nw
203         ks = self.ks
204
205     def Convolve(v):
206         Nrw, _, Nty = v.shape # Number of rows, times and types
207
208         filter = np.array([float(1/ks)]*ks)
209         sv = np.zeros((Nrw,Nw+1,Nty),dtype=float) # Smoothed
210         ↪ vector
211
212         for r in range(Nrw):
213             for t in range(Nty):
214                 sv[r,0,t] = v[r,0,t].copy()
215                 conv = np.convolve(
216                     v[r,1:,t],
217                     filter,
218                     'valid'
219                 )
220                 sv[r,1:,t] = conv[::-ks].copy()
221
222         # This functions applies the mean every ks unit of the
223         ↪ Ns snapshots
224         return sv
225
226
227         self.vrtState = np.array([data[p][0] for p in range(Ni)])
228         self.snapshots = np.array([data[p][1] for p in range(Ni)])
229         self.avrState = Convolve(np.mean(self.snapshots, axis=1))
230
231         self.nodeAvrVrtState = np.mean(self.vrtState, axis=1)
232         self.nodeMinVrtState = np.min(self.vrtState, axis=1)
233         self.nodeMaxVrtState = np.max(self.vrtState, axis=1)
234         self.nodeSumVrtState = np.sum(self.vrtState, axis=1)
235
236         if Ni>1:
237             self.ta = stats.t.ppf(0.975,df=Ni-1)
238
239             self.itAvrVrtState = np.mean(self.vrtState, axis=0)
240             self.itAvrSnapshots = np.mean(self.snapshots, axis=0)
241         else:
242             self.ta = None
243
244             self.itAvrVrtState = self.vrtState[0,:,:]
245             self.itAvrSnapshots = self.snapshots[0,:,:,:]
246
247             self.itAvrConvSnapshots = Convolve(self.itAvrSnapshots)
248             self.siVrtState = np.argsort(self.vrtState, axis=1)
249             self.siAvrState = np.argsort(self.itAvrVrtState, axis=0)
250
251         # Figures
252         def SizeDistrFittingsFig(
253             self,
254             ax=None,
255             idx=1,
256             figData=None,
257             saveFig=False
258         ):
259             Ni = self.Ni
260             ta = self.ta
261
262             csSv = self.vrtState
263             csRv = self.realSizeDistr.reshape(1,-1)
264
265             csAvr = self.nodeAvrVrtState
266             csMin = self.nodeMinVrtState

```

```

254         csMax = self.nodeMaxVrtState
255         csSum = self.nodeSumVrtState
256
257         typ = self.typ
258         # lbl = self.lblEn
259         lbl = self.lblIt
260         clr = self.clr
261
262         Nf = 13 # Numbers of figures
263         if figData is None:
264             figData = self.figData
265             fig, ax =
266                 figData.SetFigs(1,Nf,size=(self.fw*Nf,self.fh))
267             saveFig = True
268
269         sMaxEA = csSv.max(); sMaxR = csRv.max();
270         sMinEA = csSv.min(); sMinR = csRv.min();
271
272         sMaxER = max(csSv[:, :, 0].max(), csRv.max())
273         sMinER = min(csSv[:, :, 0].min(), csRv.min())
274
275         sMaxAR = max(csSv[:, :, 1].max(), csRv.max())
276         sMinAR = min(csSv[:, :, 1].min(), csRv.min())
277
278         nBins = np.zeros((3,), dtype=np.int32)
279         for i,v in enumerate([csSv[:, :, 0],csSv[:, :, 1],csRv]):
280             def EstimateBinNumber(v):
281                 vf = v.ravel()
282                 vp = vf[vf > 0]
283                 vl = np.log10(vp)
284
285                 # The option «'fd'» stands for «Freedman-Diaconis»
286                 # and
287                 # uses Numpy to calculate the optimal edges for the
288                 # data
289                 edges = np.histogram_bin_edges(vl,bins='fd')
290             return len(edges)-1
291
292             for j in range(v.shape[0]):
293                 nBin = EstimateBinNumber(v[j,:])
294                 if nBin > nBins[i]: nBins[i] = nBin
295
296             p = (.5,1.07); dp = (.6,.05)
297
298             for t in typ: # type]
299                 ### Lognormal fit ###
300
301                 # Exact-Approximated plot
302                 libF.CreateHistogramPlot(
303                     csSv[:, :, t],
304                     np.max(nBins[:2]),
305                     getattr(figData,f'fig{idx}'),
306                     limits=(sMinEA,sMaxEA),
307                     xScale='log',
308                     Ni=Ni,
309                     ta=ta,
310                     # label=f'{lbl[t]} histogram',
311                     label=f'Iistogramma {lbl[0][t]}',
312                     color=clr[t],
313                     alpha=(0.35,0.45) if Ni>1 else 0.35,
314                     idx=t+1,
315                     ax=ax[0]
316                 )
317                 xiS = libF.CreateLognormalFitPlot(
318                     csSv[:, :, t],
319                     getattr(figData,f'fig{idx}'),
320                     limits=(sMinEA,sMaxEA),
321
322
323
324
325
326
327

```

```

328         xScale='log',
329         Ni=Ni,
330         ta=ta,
331         # label=f'{lbl[t]} lognormal fit (ML)',
332         label=fr"Adatt. BLN {lbl[0][t]}",# ($\mathit{{ML}}$)
333         # (
334         #     fr'{lbl[t]} mean value $\langle k \rangle$',
335         #     f'{lbl[t]} lognormal fit (ML)'
336         # ),
337         color=clr[t],#(clr[t],clr[t]),
338         alpha=(1,0.15) if Ni>1 else 1,
339         bimodal=True,
340         idx=t+1,
341         ax=ax[0]
342     )
343
344     # Exact-Real and Approximated-Real plots
345     libF.CreateHistogramPlot(
346         csSv[:, :, t],
347         max(nBins[t], nBins[2]),
348         getattr(figData, f'fig{idx+t+1}'),
349         limits=(sMinER, sMaxER) if t == 0 else
350             (sMinAR, sMaxAR),
351         xScale='log',
352         Ni=Ni,
353         ta=ta,
354         # label=f'{lbl[t]} histogram',
355         label=f'Iistogramma {lbl[0][t]}',
356         color=clr[t],
357         alpha=(0.35, 0.45) if Ni>1 else 0.35,
358         idx=1,
359         ax=ax[0+t+1]
360     )
361     libF.CreateLognormalFitPlot(
362         csSv[:, :, t],
363         getattr(figData, f'fig{idx+t+1}'),
364         limits=(sMinER, sMaxER) if t == 0 else
365             (sMinAR, sMaxAR),
366         xScale='log',
367         Ni=Ni,
368         ta=ta,
369         # label=f'{lbl[t]} lognormal fit (ML)',
370         label=fr"Adatt. BLN {lbl[0][t]}",# ($\mathit{{ML}}$)
371         color=clr[t],#(clr[t],clr[t]),
372         alpha=(1,0.15) if Ni>1 else 1,
373         bimodal=True,
374         idx=1,
375         ax=ax[0+t+1]
376     )
377
378     libF.CreateHistogramPlot(
379         csRv,
380         max(nBins[t], nBins[2]),
381         getattr(figData, f'fig{idx+t+1}'),
382         limits=(sMinER, sMaxER) if t == 0 else
383             (sMinAR, sMaxAR),
384         xScale='log',
385         # label=f'{lbl[2]} histogram',
386         label=f'Iistogramma {lbl[0][2]}',
387         color=clr[2],
388         alpha=0.35,
389         idx=2,
390         ax=ax[0+t+1]
391     )
392     xiR = libF.CreateLognormalFitPlot(
393         csRv,
394         getattr(figData, f'fig{idx+t+1}'),

```

```

392         limits=(sMinER,sMaxER) if t == 0 else
393              $\rightarrow$  (sMinAR,sMaxAR),
394             xScale='log',
395             # label=f'{lbl[2]} lognormal fit (ML)',
396             label=fr'Adatt. BLN {lbl[0][2]}',# ($\mathit{ML})$)
397             color=clr[2],
398             alpha=1,
399             bimodal=True,
400             idx=2,
401             ax=ax[0+t+1]
402     )
403
404     # Real plot
405     if t:
406         libF.CreateHistogramPlot(
407             csRv,
408             nBins[2],
409             getattr(figData,f'fig{idx+3}'),
410             limits=(sMinR,sMaxR),
411             xScale='log',
412             # label=f'{lbl[2]} histogram',
413             label=f'Iistogramma {lbl[0][2]}',
414             color=clr[2],
415             alpha=0.35,
416             idx=2,
417             ax=ax[3]
418         )
419         libF.CreateLognormalFitPlot(
420             csRv,
421             getattr(figData,f'fig{idx+3}'),
422             limits=(sMinR,sMaxR),
423             xScale='log',
424             # label=f'{lbl[2]} lognormal fit (ML)',
425             label=fr'Adatt. BLN {lbl[0][2]}',#
426              $\rightarrow$  ($\mathit{ML})$)
427             color=clr[2],
428             alpha=1,
429             bimodal=True,
430             idx=2,
431             ax=ax[3]
432     )
433
434     #### Power law vs bimodal lognormal fit log-log ####
435
436     if t: libF.CreateParetoFitPlot(
437         csRv,
438         getattr(figData,f'fig{idx+4}'),
439         yScale='log',
440         # label=(
441             # f'{lbl[2]} empirical CCDF',
442             # f'{lbl[2]} Pareto fit (ML)',
443             # f'{lbl[2]} bimodal lognormal fit (ML)'
444             # ),
445         label=(
446             f'FRC empirica {lbl[1][2]}',
447             fr"Adatt. Pareto {lbl[0][2]}",#
448              $\rightarrow$  ($\mathit{ML})$)
449             fr"Adatt. BLN {lbl[0][2]}#$\mathit{ML})$"
450         ),
451         color=(clr[2],clr[2]),
452         alpha=(0.6,1),
453         bimodal=True,
454         idx=1,
455         ax=ax[4]
456     )
457     libF.CreateParetoFitPlot(

```

```

456         csSv[:, :, t],
457         getattr(figData, f'fig{idx+4+t+1}'),
458         upperbound=sMaxEA,
459         yScale='log',
460         Ni=Ni,
461         ta=ta,
462         # label=(
463         #     f'{lbl[t]} empirical CCDF',
464         #     f'{lbl[t]} Pareto fit (ML)',
465         #     f'{lbl[t]} bimodal lognormal fit (ML)'
466         # ),
467         label=(
468             f'FRC empirica {lbl[1][t]}',
469             fr"Adatt. Pareto {lbl[0][t]}",#
470             → ($\mathit{{ML}}$)
471             fr"Adatt. BLN {lbl[0][t]}"\# ($\mathit{{ML}}$)
472         ),
473         color=(clr[t],clr[t]),
474         alpha=((0.6,0.3),(1,0.15)) if Ni>1 else (0.6,1),
475         bimodal=True,
476         idx=1,
477         ax=ax[4+t+1]
478     )
479
480     ### Power law fit log-log ####
481
482     # Exact-Approximated plot
483     libF.CreateParetoFitPlot(
484         csSv[:, :, t],
485         getattr(figData, f'fig{idx+7}'),
486         upperbound=sMaxEA,
487         yScale='log',
488         Ni=Ni,
489         ta=ta,
490         # label=(
491         #     f'{lbl[t]} empirical CCDF',
492         #     f'{lbl[t]} Pareto fit (ML)'
493         # ),
494         label=(
495             f'FRC empirica {lbl[1][t]}',
496             fr"Adatt. Pareto {lbl[0][t]}\# ($\mathit{{ML}}$)"
497         ),
498         color=(clr[t],clr[t]),
499         alpha=((0.6,0.3),(1,0.15)) if Ni>1 else (0.6,1),
500         idx=t+1,
501         ax=ax[7]
502     )
503
504     # Exact-Real and Approximated-Real plots
505     libF.CreateParetoFitPlot(
506         csSv[:, :, t],
507         getattr(figData, f'fig{idx+7+t+1}'),
508         upperbound=sMaxER if t == 0 else sMaxAR,
509         yScale='log',
510         Ni=Ni,
511         ta=ta,
512         # label=(
513         #     f'{lbl[t]} empirical CCDF',
514         #     f'{lbl[t]} Pareto fit (ML)'
515         # ),
516         label=(
517             f'FRC empirica {lbl[1][t]}',
518             fr"Adatt. Pareto {lbl[0][t]}\# ($\mathit{{ML}}$)"
519         ),
520         color=(clr[t],clr[t]),
521         alpha=((0.6,0.3),(1,0.15)) if Ni>1 else (0.6,1),

```

```

522         idx=1,
523         ax=ax[7+t+1]
524     )
525     libF.CreateParetoFitPlot(
526         csRv,
527         getattr(figData,f'fig{idx+7+t+1}'),
528         upperbound=sMaxER if t == 0 else sMaxAR,
529         yScale='log',
530         # label=(
531         #     f'{lbl[2]} empirical CCDF',
532         #     f'{lbl[2]} Pareto fit (ML)'
533         # ),
534         label=(
535             f'FRC empirica {lbl[1][2]}',
536             fr"Adatt. Pareto {lbl[0][2]}"# ($\mathit{{ML}}$)
537         ),
538         color=(clr[2],clr[2]),
539         alpha=(0.6,1),
540         idx=2,
541         ax=ax[7+t+1]
542     )
543
544
545     ### Power law fit log-lin ###
546
547     # Exact-Approximated plot
548     blS = libF.CreateParetoFitPlot(
549         csSv[:, :, t],
550         getattr(figData,f'fig{idx+10}'),
551         upperbound=sMaxEA,
552         yScale='lin',
553         Ni=Ni,
554         ta=ta,
555         # label=(
556         #     f'{lbl[t]} empirical CCDF',
557         #     fr'{lbl[t]} Pareto fit (ML)'
558         # ),
559         label=(
560             f'FRC empirica {lbl[1][t]}',
561             fr"Adatt. Pareto {lbl[0][t]}"# ($\mathit{{ML}}$)
562         ),
563         color=(clr[t],clr[t]),
564         alpha=((0.6,0.3),(1,0.15)) if Ni>1 else (0.6,1),
565         idx=t+1,
566         ax=ax[10]
567     )
568
569     # Exact-Real and Approximated-Real plots
570     libF.CreateParetoFitPlot(
571         csSv[:, :, t],
572         getattr(figData,f'fig{idx+10+t+1}'),
573         upperbound=sMaxER if t == 0 else sMaxAR,
574         yScale='log',
575         Ni=Ni,
576         ta=ta,
577         # label=(
578         #     f'{lbl[t]} empirical CCDF',
579         #     fr'{lbl[t]} Pareto fit (ML)'
580         # ),
581         label=(
582             f'FRC empirica {lbl[1][t]}',
583             fr"Adatt. Pareto {lbl[0][t]}"# ($\mathit{{ML}}$)
584         ),
585         color=(clr[t],clr[t]),
586         alpha=((0.6,0.3),(1,0.15)) if Ni>1 else (0.6,1),
587         idx=1,
588         ax=ax[10+t+1]

```

```

589     )
590     blR = libF.CreateParetoFitPlot(
591         csRv,
592         getattr(figData,f'fig{idx+10+t+1}'),
593         upperbound=sMaxER if t == 0 else sMaxAR,
594         yScale='log',
595         # label=(
596         #     f'{lbl[2]} empirical CCDF',
597         #     fr'{lbl[2]} Pareto fit (ML)'
598         # ),
599         label=(
600             f'FRC empirica {lbl[1][2]}',
601             fr"Adatt. Pareto {lbl[0][2]}",#
602             ↪ ($\mathit{ML})$)
603         ),
604         color=(clr[2],clr[2]),
605         alpha=(0.6,1),
606         idx=2,
607         ax=ax[10+t+1]
608     )
609
610     libF.Text(
611         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
612         # fr'{lbl[t]}:$\quad$+
613         fr"${lbl[2][t]}:$\quad$+
614         libF.DataString(xiS,Ni,ta,r'\xi')+_
615         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
616         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
617         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
618             \rangle')+
619         libF.DataString(
620             csSum[:,t],Ni,ta,r's_{\Sigma}',_
621             formatVal='.2e',formatErr='.2e'
622         )+
623         libF.DataString(bls,Ni,ta,r'\beta',space=False),
624         ha='center',
625         color=clr[t]
626     )
627
628     if idx == 1:
629         libF.TextBlock(
630             ax[1],>[
631                 fr'$N_c={self.Nc}$',
632                 fr'$R={self.R}$',
633                 libF.DataString(blR,head=r'\beta',space=False),
634                 libF.DataString(xiR,head=r'\xi',space=False)
635             ],
636             p=(.5,p[1]-dp[1]/2),
637             dp=(dp[0]/1.5,dp[1])
638         )
639
640         p = (.5,p[1])
641         dp = (.8,dp[1])
642         libF.TextBlock(
643             ax[-1],>[
644                 fr'$\lambda={self.l}$',
645                 fr'$\sigma={self.s}$',
646                 fr'$\zeta={self.z}$',
647                 fr'$N_i={Ni}$',
648                 fr'$N_s={Ns}$',
649             ],
650             [
651                 fr'$\alpha={self.a}$',
652                 fr'$i_l={self.il+1}$',
653                 fr'$d_i={w_i}' if self.ew else 'k_i'$',
654                 fr'$\Delta t={self.dt}$',
655                 fr'$N_w={self.Nw}$',
656             ]
657         )
658
659     )
660
661     libF.Text(
662         ax[1],(p[0],p[1]+(1/2-t)*dp[1]),
663         # fr'{lbl[t]}:$\quad$+
664         fr"${lbl[2][t]}:$\quad$+
665         libF.DataString(xiS,Ni,ta,r'\xi')+_
666         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
667         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
668         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
669             \rangle')+
670         libF.DataString(
671             csSum[:,t],Ni,ta,r's_{\Sigma}',_
672             formatVal='.2e',formatErr='.2e'
673         )+
674         libF.DataString(bls,Ni,ta,r'\beta',space=False),
675         ha='center',
676         color=clr[t]
677     )
678
679     libF.Text(
680         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
681         # fr'{lbl[t]}:$\quad$+
682         fr"${lbl[2][t]}:$\quad$+
683         libF.DataString(xiS,Ni,ta,r'\xi')+_
684         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
685         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
686         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
687             \rangle')+
688         libF.DataString(
689             csSum[:,t],Ni,ta,r's_{\Sigma}',_
690             formatVal='.2e',formatErr='.2e'
691         )+
692         libF.DataString(bls,Ni,ta,r'\beta',space=False),
693         ha='center',
694         color=clr[t]
695     )
696
697     libF.Text(
698         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
699         # fr'{lbl[t]}:$\quad$+
700         fr"${lbl[2][t]}:$\quad$+
701         libF.DataString(xiS,Ni,ta,r'\xi')+_
702         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
703         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
704         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
705             \rangle')+
706         libF.DataString(
707             csSum[:,t],Ni,ta,r's_{\Sigma}',_
708             formatVal='.2e',formatErr='.2e'
709         )+
710         libF.DataString(bls,Ni,ta,r'\beta',space=False),
711         ha='center',
712         color=clr[t]
713     )
714
715     libF.Text(
716         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
717         # fr'{lbl[t]}:$\quad$+
718         fr"${lbl[2][t]}:$\quad$+
719         libF.DataString(xiS,Ni,ta,r'\xi')+_
720         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
721         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
722         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
723             \rangle')+
724         libF.DataString(
725             csSum[:,t],Ni,ta,r's_{\Sigma}',_
726             formatVal='.2e',formatErr='.2e'
727         )+
728         libF.DataString(bls,Ni,ta,r'\beta',space=False),
729         ha='center',
730         color=clr[t]
731     )
732
733     libF.Text(
734         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
735         # fr'{lbl[t]}:$\quad$+
736         fr"${lbl[2][t]}:$\quad$+
737         libF.DataString(xiS,Ni,ta,r'\xi')+_
738         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
739         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
740         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
741             \rangle')+
742         libF.DataString(
743             csSum[:,t],Ni,ta,r's_{\Sigma}',_
744             formatVal='.2e',formatErr='.2e'
745         )+
746         libF.DataString(bls,Ni,ta,r'\beta',space=False),
747         ha='center',
748         color=clr[t]
749     )
750
751     libF.Text(
752         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
753         # fr'{lbl[t]}:$\quad$+
754         fr"${lbl[2][t]}:$\quad$+
755         libF.DataString(xiS,Ni,ta,r'\xi')+_
756         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
757         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
758         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
759             \rangle')+
760         libF.DataString(
761             csSum[:,t],Ni,ta,r's_{\Sigma}',_
762             formatVal='.2e',formatErr='.2e'
763         )+
764         libF.DataString(bls,Ni,ta,r'\beta',space=False),
765         ha='center',
766         color=clr[t]
767     )
768
769     libF.Text(
770         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
771         # fr'{lbl[t]}:$\quad$+
772         fr"${lbl[2][t]}:$\quad$+
773         libF.DataString(xiS,Ni,ta,r'\xi')+_
774         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
775         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
776         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
777             \rangle')+
778         libF.DataString(
779             csSum[:,t],Ni,ta,r's_{\Sigma}',_
780             formatVal='.2e',formatErr='.2e'
781         )+
782         libF.DataString(bls,Ni,ta,r'\beta',space=False),
783         ha='center',
784         color=clr[t]
785     )
786
787     libF.Text(
788         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
789         # fr'{lbl[t]}:$\quad$+
790         fr"${lbl[2][t]}:$\quad$+
791         libF.DataString(xiS,Ni,ta,r'\xi')+_
792         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
793         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
794         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
795             \rangle')+
796         libF.DataString(
797             csSum[:,t],Ni,ta,r's_{\Sigma}',_
798             formatVal='.2e',formatErr='.2e'
799         )+
800         libF.DataString(bls,Ni,ta,r'\beta',space=False),
801         ha='center',
802         color=clr[t]
803     )
804
805     libF.Text(
806         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
807         # fr'{lbl[t]}:$\quad$+
808         fr"${lbl[2][t]}:$\quad$+
809         libF.DataString(xiS,Ni,ta,r'\xi')+_
810         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
811         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
812         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
813             \rangle')+
814         libF.DataString(
815             csSum[:,t],Ni,ta,r's_{\Sigma}',_
816             formatVal='.2e',formatErr='.2e'
817         )+
818         libF.DataString(bls,Ni,ta,r'\beta',space=False),
819         ha='center',
820         color=clr[t]
821     )
822
823     libF.Text(
824         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
825         # fr'{lbl[t]}:$\quad$+
826         fr"${lbl[2][t]}:$\quad$+
827         libF.DataString(xiS,Ni,ta,r'\xi')+_
828         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
829         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
830         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
831             \rangle')+
831         libF.DataString(
832             csSum[:,t],Ni,ta,r's_{\Sigma}',_
833             formatVal='.2e',formatErr='.2e'
834         )+
835         libF.DataString(bls,Ni,ta,r'\beta',space=False),
836         ha='center',
837         color=clr[t]
838     )
839
840     libF.Text(
841         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
842         # fr'{lbl[t]}:$\quad$+
843         fr"${lbl[2][t]}:$\quad$+
844         libF.DataString(xiS,Ni,ta,r'\xi')+_
845         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
846         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
847         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
848             \rangle')+
849         libF.DataString(
850             csSum[:,t],Ni,ta,r's_{\Sigma}',_
851             formatVal='.2e',formatErr='.2e'
852         )+
853         libF.DataString(bls,Ni,ta,r'\beta',space=False),
854         ha='center',
855         color=clr[t]
856     )
857
858     libF.Text(
859         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
860         # fr'{lbl[t]}:$\quad$+
861         fr"${lbl[2][t]}:$\quad$+
862         libF.DataString(xiS,Ni,ta,r'\xi')+_
863         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
864         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
865         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
866             \rangle')+
867         libF.DataString(
868             csSum[:,t],Ni,ta,r's_{\Sigma}',_
869             formatVal='.2e',formatErr='.2e'
870         )+
871         libF.DataString(bls,Ni,ta,r'\beta',space=False),
872         ha='center',
873         color=clr[t]
874     )
875
876     libF.Text(
877         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
878         # fr'{lbl[t]}:$\quad$+
879         fr"${lbl[2][t]}:$\quad$+
880         libF.DataString(xiS,Ni,ta,r'\xi')+_
881         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
882         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
883         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
884             \rangle')+
885         libF.DataString(
886             csSum[:,t],Ni,ta,r's_{\Sigma}',_
887             formatVal='.2e',formatErr='.2e'
888         )+
889         libF.DataString(bls,Ni,ta,r'\beta',space=False),
890         ha='center',
891         color=clr[t]
892     )
893
894     libF.Text(
895         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
896         # fr'{lbl[t]}:$\quad$+
897         fr"${lbl[2][t]}:$\quad$+
898         libF.DataString(xiS,Ni,ta,r'\xi')+_
899         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
900         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
901         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
902             \rangle')+
903         libF.DataString(
904             csSum[:,t],Ni,ta,r's_{\Sigma}',_
905             formatVal='.2e',formatErr='.2e'
906         )+
907         libF.DataString(bls,Ni,ta,r'\beta',space=False),
908         ha='center',
909         color=clr[t]
910     )
911
912     libF.Text(
913         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
914         # fr'{lbl[t]}:$\quad$+
915         fr"${lbl[2][t]}:$\quad$+
916         libF.DataString(xiS,Ni,ta,r'\xi')+_
917         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
918         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
919         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
920             \rangle')+
921         libF.DataString(
922             csSum[:,t],Ni,ta,r's_{\Sigma}',_
923             formatVal='.2e',formatErr='.2e'
924         )+
925         libF.DataString(bls,Ni,ta,r'\beta',space=False),
926         ha='center',
927         color=clr[t]
928     )
929
930     libF.Text(
931         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
932         # fr'{lbl[t]}:$\quad$+
933         fr"${lbl[2][t]}:$\quad$+
934         libF.DataString(xiS,Ni,ta,r'\xi')+_
935         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
936         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
937         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
938             \rangle')+
939         libF.DataString(
940             csSum[:,t],Ni,ta,r's_{\Sigma}',_
941             formatVal='.2e',formatErr='.2e'
942         )+
943         libF.DataString(bls,Ni,ta,r'\beta',space=False),
944         ha='center',
945         color=clr[t]
946     )
947
948     libF.Text(
949         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
950         # fr'{lbl[t]}:$\quad$+
951         fr"${lbl[2][t]}:$\quad$+
952         libF.DataString(xiS,Ni,ta,r'\xi')+_
953         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
954         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
955         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
956             \rangle')+
957         libF.DataString(
958             csSum[:,t],Ni,ta,r's_{\Sigma}',_
959             formatVal='.2e',formatErr='.2e'
960         )+
961         libF.DataString(bls,Ni,ta,r'\beta',space=False),
962         ha='center',
963         color=clr[t]
964     )
965
966     libF.Text(
967         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
968         # fr'{lbl[t]}:$\quad$+
969         fr"${lbl[2][t]}:$\quad$+
970         libF.DataString(xiS,Ni,ta,r'\xi')+_
971         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
972         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
973         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
974             \rangle')+
975         libF.DataString(
976             csSum[:,t],Ni,ta,r's_{\Sigma}',_
977             formatVal='.2e',formatErr='.2e'
978         )+
979         libF.DataString(bls,Ni,ta,r'\beta',space=False),
980         ha='center',
981         color=clr[t]
982     )
983
984     libF.Text(
985         ax[7],(p[0],p[1]+(1/2-t)*dp[1]),
986         # fr'{lbl[t]}:$\quad$+
987         fr"${lbl[2][t]}:$\quad$+
988         libF.DataString(xiS,Ni,ta,r'\xi')+_
989         libF.DataString(csMin[:,t],Ni,ta,r's_{min}')+_
990         libF.DataString(csMax[:,t],Ni,ta,r's_{max}')+_
991         libF.DataString(csAvr[:,t],Ni,ta,r'\langle
992             \rangle')+
993         libF.DataString(
994             csSum[:,t],Ni,ta,r's_{\Sigma}',_
995             formatVal='.2e',formatErr='.2e'
996         )+
997         libF.DataString(bls,Ni,ta,r'\beta',space=False),
998         ha='center',
999         color=clr[t]
1000    )

```

```

654                 ],
655                 p=p,dp=dp
656             )
657
658         for f in range(Nf):
659             libF.SetFigStyle(
660                 r'$s$',r'$\bar{f}_N(s)$',
661                 yNotation='sci' if f<3 else 'plain',
662                 xScale='log',
663                 yScale='log' if f<10 and f>3 else 'lin',
664                 ax=ax[f],
665                 data=getattr(figData,f'fig{idx+f}')
666             )
667
668         # CentreFig()
669         if saveFig: figData.SaveFig('SizeDistributionFittings')
670         else: libF.Text(ax[-1],(1.1,.5),self.studiedPrmString)
671
672     def AverageSizeFig(
673         self,
674         ax=None,
675         idx=1,
676         figData=None,
677         saveFig=False
678     ):
679         Ni = self.Ni
680         ta = self.ta
681
682         times = self.times
683         csSa = self.avrState
684         csRa = np.ones_like(times)*self.realSizeDistr.mean()
685
686         # lbl = self.lblEn
687         lbl = self.lblIt
688         clr = self.clr
689
690         if figData is None:
691             figData = self.figData
692             fig = figData.SetFigs()
693             saveFig = True
694
695         for t in self.typ:
696             libF.CreateFunctionPlot(
697                 times,
698                 csSa[:, :, t],
699                 figData.fig if saveFig
700                 else getattr(figData,f'fig{idx}'),
701                 Ni=Ni,
702                 ta=ta,
703                 # label=rf"\{lbl[t]\} city size average $\langle
704                 # \rangle$",
705                 label=fr'Taglia media {lbl[1][t]}',
706                 color=clr[t],
707                 alpha=(1,0.15) if Ni>1 else 1,
708                 idx=t+1,
709                 ax=ax
710             )
711
712         libF.CreateFunctionPlot(
713             times,
714             csRa,
715             figData.fig if saveFig
716             else getattr(figData,f'fig{idx}'),
717             linestyle='--',
718             linewidth=0.75,
719             # label=rf"\{lbl[2]\} city size average $\langle
720             # \rangle$",

```

```

719         label=rf'Taglia media {lbl[1][2]}',
720         color=clr[2],
721         alpha=1,
722         idx=3,
723         ax=ax
724     )
725
726     # CentreFig()
727     libF.SetFigStyle(
728         r"$t$",
729         data=figData.fig if saveFig
730         else getattr(figData,f'fig{idx}'),
731         ax=ax
732     )
733
734     if saveFig: figData.SaveFig('AverageSize')
735     else: libF.Text(ax,(1.1,.5),self.studiedPrmString)
736
737     def SizeVsDegreeFig(
738         self,
739         ax=None,
740         idx=1,
741         figData=None,
742         saveFig=False
743     ):
744         ki = self.dk
745
746         csSv = self.itAvrVrtState
747         csRv = self.realSizeDistr
748
749         typ = self.typ
750         # lbl = self.lblEn
751         lbl = self.lblIt
752         clr = self.clr
753
754         si = self.siAvrState
755         li2Name = self.li2Name
756
757         if figData is None:
758             figData = self.figData
759             Nf = 3 # Numbers of figures
760             fig, ax =
761                 figData.SetFigs(1,Nf,size=(self.fw*Nf,self.fh))
762             saveFig = True
763
764         for t in typ:
765             libF.CreateScatterPlot(
766                 ki,csSv[:,t],
767                 getattr(figData,f'fig{idx}'),
768                 # label=lbl[t],
769                 label=f'Dispersione {lbl[1][t]}',
770                 color=clr[t],
771                 alpha=0.7,
772                 idx=t+1,
773                 ax=ax[0]
774             )
775
776             libF.CreateScatterPlot(
777                 ki,csSv[:,t],
778                 getattr(figData,f'fig{t+1+idx}'),
779                 # label=lbl[t],
780                 label=f'Dispersione {lbl[1][t]}',
781                 color=clr[t],
782                 alpha=0.6,
783                 idx=1,
784                 ax=ax[t+1]
785             )

```

```

735         libF.CreateScatterPlot(
736             ki,csRv,
737             getattr(figData,f'fig{t+1+idx}'),
738             # label=lbl[2],
739             label=f'Dispersione {lbl[1][2]}',
740             color=clr[2],
741             alpha=0.6,
742             idx=2,
743             ax=ax[t+1]
744         )
745
746     for f in range(3):
747         libF.SetFigStyle(
748             r'$k$',r'$s(k)$',
749             yScale='log',xScale='log',
750             data=getattr(figData,f'fig{f+idx}'),
751             ax=ax[f]
752     )
753
754     for t in typ:
755         libF.TextBlock(
756             ax[0],[
757                 libF.DataString(
758                     csSv[si[i,t],t],
759                     head=li2Name[si[i,t]],
760                     formatVal='.2e',
761                     space=False
762                 )
763             ] for i in range(-1,-6,-1)],
764             p=(.8,.35-t*.225),
765             dp=(0,.15),
766             ha='center',
767             color=clr[t]
768         )
769
770     # CentreFig()
771     if saveFig: figData.SaveFig('SizeVsDegree')
772     else: libF.Text(ax[-1],(1.1,.5),self.studiedPrmString)
773
774     def SizeDistrEvolutionFig(
775         self,
776         ax=None,
777         idx=1,
778         figData=None,
779         saveFig=False
780     ):
781         snapshots = self.itAvrS snapshots
782
783         ns = self.ns
784         Ns = self.Ns
785
786         dt = self.dt
787         typ = self.typ
788
789         if figData is None:
790             figData = self.figData
791             Nf = 2 # Numbers of figures
792             fig, ax =
793                 figData.SetFigs(1,Nf,size=(self.fw*Nf,self.fh))
794             saveFig = True
795             ax[0].set_title('Exact'); ax[1].set_title('Approximated')
796
797             samples = 6
798             clrmapping = get_cmap('inferno') # magma
799             colours = [clrmapping(i/(samples-1)) for i in range(samples)]
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850

```

```

851     sMax = np.max(snapshots[:, -1, :])
852     sMin = np.min(snapshots[:, -1, :])
853
854     for t in typ: # t[type]
855         for j,s in
856             → enumerate(np.linspace(0,Ns,samples,dtype=int)):
857                 libF.CreateHistogramPlot(
858                     snapshots[:,s,t],21,
859                     getattr(figData,f'fig{t+idx}'),
860                     limits=(sMin,sMax),
861                     xScale='log',
862                     label=f't = {int(ns[s]*dt)}',
863                     color=colours[j][:-1],
864                     alpha=0.4,
865                     idx=j+idx,
866                     ax=ax[t],
867                     norm=False
868                 ) # Histogram plot
869
870             libF.SetFigStyle(
871                 r"$cs$",
872                 r"$P(cs)$",
873                 yNotation="sci", # ,xNotation="sci"
874                 xScale='log',
875                 yScale='log',
876                 ax=ax[t],
877                 data=getattr(figData,f'fig{t+idx}'))
878             ) # Style
879
880             # CentreFig()
881             if saveFig: figData.SaveFig('SizeDistributionEvolution')
882             else: libF.Text(ax[-1],(1.1,.5),self.studiedPrmString)
883
884     def SizeEvolutionsFig(
885         self,
886         ax=None,
887         idx=1,
888         figData=None,
889         saveFig=False
890     ):
891         # Nc = self.Nc
892         typ = self.typ
893
894         times = self.times
895         snapshots = self.itAvrConvSnapshots
896
897         ki = self.dk
898         sf = self.Nw
899
900         if figData is None:
901             figData = self.figData
902             Nf = 2 # Numbers of figures
903             fig, ax =
904                 → figData.SetFigs(1,Nf,size=(self.fw*Nf,self.fh))
905             saveFig = True
906             ax[0].set_title('Exact'); ax[1].set_title('Approximated')
907
908             dk, _ = np.unique(ki,return_counts=True); Nk = dk.size
909             snapshotsk = np.zeros((Nk,sf+1,2),dtype=np.float64)
910             for i,k in enumerate(dk):
911                 bk = ki == k
912                 snapshotsk[i,:,:] = snapshots[bk,:,:].mean(axis=0)
913
914             clrmap = get_cmap('inferno') # magma
915             norm = LogNorm(vmin=dk.min(),vmax=dk.max())
916             colours = clrmap(norm(dk[:-1]))
917             # colours = [clrmap(i/(Nc-1)) for i in range(Nc)]
918
919

```

```

916     labels = ['']*Nk
917     classes = np.linspace(0,Nk-1,4,dtype=np.int64)
918     for i in classes:
919         # labels[i] = f'Class k={dk[i]}'
920         labels[i] = f'Classe k={dk[i]}'
921
922     for t in typ: # t[ype]
923         # norm = LogNorm(
924         #     vmin=snapshotsk[:, -1, t].min(),
925         #     vmax=snapshotsk[:, -1, t].max()
926         # )
927         # colours = clrmap(norm(snapshotsk[:, -1, t]))
928
929     for k in np.linspace(Nk-1,0,Nk,dtype=np.int64):
930         libF.CreateFunctionPlot(
931             times,snapshotsk[k,:,:t],
932             getattr(figData,f'fig{t+idx}'),
933             color=colours[k,:],
934             alpha=0.8,
935             linewidth=1,
936             label=labels[k],
937             idx=k+idx,
938             ax=ax[t]
939         ) # Histogram plot
940
941     libF.SetFigStyle(
942         r"$t$",
943         r"$s$",
944         # yNotation="sci", # ,xNotation="sci"
945         yScale='log',
946         data=getattr(figData,f'fig{t+idx}'),
947         ax=ax[t]
948     ) # Style
949
950     # CentreFig()
951     if saveFig: figData.SaveFig('SizeEvolutions')
952     else: libF.Text(ax[-1],(1.1,.5),self.studiedPrmString)
953
954     def ShowFig(self):
955         from matplotlib.pyplot import show
956         show()
957
958     class ParametricStudy():
959         def __init__(self,clsPrm,clsReg):
960             sp = clsPrm.studiedParameter
961
962             sV = clsPrm.startValuePrmStudy; self.sV = sV
963             eV = clsPrm.endValuePrmStudy; self.eV = eV
964             Nv = clsPrm.numberPrmStudy; self.Nv = Nv
965
966             self.KS = [None]*Nv
967             frmt = '.3f'
968             for s,val in enumerate(np.linspace(sV,eV,Nv)):
969                 match sp:
970                     case 0:
971                         clsPrm.attractivity = val
972                         string = fr'$\lambda={val:{frmt}}$'
973                     case 1:
974                         clsPrm.convincibility = val
975                         string = fr'$\alpha={val:{frmt}}$'
976                     case 2:
977                         clsPrm.zetaFraction = val
978                         string = fr'$\zeta={val:{frmt}}$'
979
980             self.KS[s] = KineticSimulation(clsPrm,clsReg)
981             self.KS[s].sid = s+1
982             self.KS[s].studiedPrmString = string

```

```

983         self.fw = 8
984         self.fh = 6
985         self.figData = libF.FigData(clsPrm,'KS')
986
987     # Simulation
988     def MonteCarloSimulation(self):
989         for sim in self.KS:
990             sim.MonteCarloSimulation()
991
992     # Figures
993     def SizeDistrFittingsFig(self):
994         figData = self.figData
995         fw = self.fw
996         fh = self.fh
997
998         nCol = 13 # Figures for each row
999         nRow = self.Nv
1000        fig, ax = figData.SetFigs(nRow,nCol,size=(fw*nCol,fh*nRow))
1001
1002        for s,KS in enumerate(self.KS):
1003            KS.SizeDistrFittingsFig(
1004                ax=ax[:,],
1005                idx=nCol*s+1,
1006                figData=figData
1007            )
1008
1009        figData.SaveFig('SizeDistributionFittings')
1010
1011    def AverageSizeFig(self):
1012        figData = self.figData
1013        fw = self.fw
1014        fh = self.fh
1015
1016        nCol = 1 # Figures for each row
1017        nRow = self.Nv
1018        fig, ax = figData.SetFigs(nRow,nCol,size=(fw*nCol,fh*nRow))
1019
1020        for s,KS in enumerate(self.KS):
1021            KS.AverageSizeFig(
1022                ax=ax[s],
1023                idx=nCol*s+1,
1024                figData=figData
1025            )
1026
1027        figData.SaveFig('AverageSize')
1028
1029    def SizeVsDegreeFig(self):
1030        figData = self.figData
1031        fw = self.fw
1032        fh = self.fh
1033
1034        nCol = 3 # Figures for each row
1035        nRow = self.Nv
1036        fig, ax = figData.SetFigs(nRow,nCol,size=(fw*nCol,fh*nRow))
1037
1038        for s,KS in enumerate(self.KS):
1039            KS.SizeVsDegreeFig(
1040                ax=ax[:,],
1041                idx=nCol*s+1,
1042                figData=figData
1043            )
1044
1045        figData.SaveFig('SizeVsDegree')
1046
1047    def SizeDistrEvolutionFig(self):
1048        figData = self.figData
1049        fw = self.fw

```

```

1050         fh = self.fh
1051
1052         nCol = 2 # Figures for each row
1053         nRow = self.Nv
1054         fig, ax = figData.SetFigs(nRow,nCol,size=(fw*nCol,fh*nRow))
1055
1056         for s,KS in enumerate(self.KS):
1057             KS.SizeDistrEvolutionFig(
1058                 ax=ax[s,:],
1059                 idx=nCol*s+1,
1060                 figData=figData
1061             )
1062
1063         figData.SaveFig('SizeDistributionEvolution')
1064
1065     def SizeEvolutionsFig(self):
1066         figData = self.figData
1067         fw = self.fw
1068         fh = self.fh
1069
1070         nCol = 2 # Figures for each row
1071         nRow = self.Nv
1072         fig, ax = figData.SetFigs(nRow,nCol,size=(fw*nCol,fh*nRow))
1073
1074         for s,KS in enumerate(self.KS):
1075             KS.SizeEvolutionsFig(
1076                 ax=ax[s,:],
1077                 idx=nCol*s+1,
1078                 figData=figData
1079             )
1080
1081         figData.SaveFig('SizeEvolutions')
1082
1083     def ShowFig(self):
1084         from matplotlib.pyplot import show
1085         show()
1086
1087
1088     ### Auxiliary functions ###
1089
1090     @dataclass(frozen=True)
1091     class ShMSpec:
1092         name: str
1093         shape: tuple[int,...]
1094         dtype: str
1095
1096     def BuildShM(
1097         array=None,
1098         Ni=None,
1099         dtype=None
1100     ):
1101         if array is not None:
1102             a = np.ascontiguousarray(array)
1103         else:
1104             a = np.zeros(Ni,dtype=dtype)
1105
1106         shm = shared_memory.SharedMemory(
1107             create=True,
1108             size=a.nbytes
1109         )
1110
1111         view = np.ndarray(
1112             a.shape,
1113             dtype=a.dtype,
1114             buffer=shm.buf
1115         )
1116         view[:] = a

```

```

1117     spec = ShMSpec(
1118         name=shm.name,
1119         shape=a.shape,
1120         dtype=a.dtype.str
1121     )
1122
1123     return spec, shm
1124
1125
1126 def LoadShM(spec):
1127     shm = shared_memory.SharedMemory(name=spec.name)
1128     arr = np.ndarray(
1129         spec.shape,
1130         dtype=np.dtype(spec.dtype),
1131         buffer=shm.buf
1132     )
1133     return shm, arr
1134
1135
1136 def InitializeMCSWorker(
1137     shmPrm,
1138     gui=False
1139 ):
1140     for key in workersShM['parameters'].keys():
1141         shm, arr = LoadShM(shmPrm[key])
1142         workersShM['handles'].append(shm)
1143         workersShM['parameters'][key] = arr
1144
1145     if gui:
1146         for key in workersShM['gui'].keys():
1147             shm, arr = LoadShM(shmPrm[key])
1148             workersShM['handles'].append(shm)
1149             workersShM['gui'][key] = arr
1150
1151
1152 def MonteCarloAlgorithm(
1153     p,Nc,Ns,p0,typ,
1154     l,a,s,z,il/gui
1155 ):
1156     Mdt = workersShM['parameters']["Mdt"]
1157     wOdt = workersShM['parameters']["wOdt"]
1158     wI = workersShM['parameters']["wI"]
1159     di = workersShM['parameters']["di"]
1160     idi = workersShM['parameters']["idi"]
1161     ns = workersShM['parameters']["ns"]
1162
1163     # rng = np.random.default_rng() # Even though it's recommended
1164     # → by Numpy it is not efficiently implemented in Numba, hence
1165     # → it halves the iterations per second if used
1166
1167     # Uniform initial state for all vertices
1168     vrtState = np.full((Nc,typ),p0,dtype=np.float64)
1169     snapshots = np.full((Nc,Ns+1,typ),p0,dtype=np.float64)
1170
1171     P = np.arange(Nc,dtype=np.int32)
1172
1173     nk = ns[1]
1174     hNc = Nc//2 # Nc half
1175     nsid = 1
1176
1177     if gui: # If ProgressGUIT is required
1178         progress = workersShM['gui']['progress']
1179         elapsed = workersShM['gui']['elapsed']
1180         done = workersShM['gui']['done']
1181
1182         EvolveState(
1183             vrtState,P,
1184             Nc,hNc,nk,
1185             Mdt,wOdt,wI,

```

```

1182             di,idi,il,
1183             l,a,s,z
1184         ) # Warm-up iteration to avoid polluting the initial time to
1185         snapshots[:,nsid,0] = vrtState[:,0]
1186         snapshots[:,nsid,1] = vrtState[:,1]
1187         nsid += 1
1188
1189         t0 = time.perf_counter()
1190         for st in range(Ns-1): # step
1191             EvolveState(
1192                 vrtState,P,
1193                 Nc,hNc,nk,
1194                 Mdt,w0dt,wI,
1195                 di,idi,il,
1196                 l,a,s,z
1197             )
1198             snapshots[:,nsid,0] = vrtState[:,0]
1199             snapshots[:,nsid,1] = vrtState[:,1]
1200             nsid += 1
1201
1202             progress[p] = (st+2)*nk # ns[nsid]
1203             elapsed[p] = time.perf_counter()-t0
1204
1205             progress[p] = Ns*nk # ns[-1]
1206             elapsed[p] = time.perf_counter()-t0
1207             done[p] = True
1208
1209         return vrtState, snapshots
1210
1211     else:
1212         EvolveState(
1213             vrtState,P,
1214             Nc,hNc,nk,
1215             Mdt,w0dt,wI,
1216             di,idi,il,
1217             l,a,s,z
1218         ) # Warm-up iteration to avoid polluting the initial time to
1219         snapshots[:,nsid,0] = vrtState[:,0]
1220         snapshots[:,nsid,1] = vrtState[:,1]
1221         nsid += 1
1222
1223         t0 = time.perf_counter()
1224         for _ in range(Ns-1):
1225             EvolveState(
1226                 vrtState,P,
1227                 Nc,hNc,nk,
1228                 Mdt,w0dt,wI,
1229                 di,idi,il,
1230                 l,a,s,z
1231             )
1232             snapshots[:,nsid,0] = vrtState[:,0]
1233             snapshots[:,nsid,1] = vrtState[:,1]
1234             nsid += 1
1235
1236         return vrtState, snapshots
1237
1238 @njit(cache=True)
1239 def EvolveState(
1240     cs,P,Nc,
1241     hNc,nk,Mdt,
1242     w0dt,wI,
1243     di,idi,il,
1244     l,a,s,z
1245 ):
1246     for _ in range(nk):
1247         FYDInPlaceShuffle(P,Nc)
1248         # P = np.random.permutation(Nc)

```

```

1249     # pi = P[:hNc]; pr = P[hNc:]
1250
1251     for i in range(hNc):
1252         ii = P[i]; ir = P[i+hNc]
1253
1254         # t = 0
1255         p = Mdt[ii,ir]
1256         if p > 0:
1257             theta = np.random.random() < p
1258
1259             if theta == 1:
1260                 si = cs[ii,0]; sr = cs[ir,0]
1261
1262                 e = NonLinearEmigration(si,idi[ii],sr,di[ir],i,
1263                               ↪ l,l,a,z)
1264                 ga = StochasticFluctuations(s,e) if s>0 else 0
1265
1266                 cs[ii,0] = si*(1-e+ga)
1267                 cs[ir,0] = sr+si*e
1268
1269             # t = 1
1270             p = w0dt[ii]*wI[ir]
1271             # Apparently it's more efficient to access two values
1272             # → from two separate vectors and to multiply them,
1273             # → than it is to access the same pre-computed value
1274             # → from a matrix; the same holds for the product
1275             # → «di[ir]*idi[ii]» inside «NonLinearEmigration()»
1276             # However, in order for this property to be valid the
1277             # → matrix has to have an underlying more simple
1278             # → structure which in both cases is rank 1
1279             # In other words this trick does not work with the
1280             # → adjacency matrix «Mdt[ii,ir]» which cannot be
1281             # → computed from simpler elements
1282             theta = np.random.random() < p
1283
1284             if theta == 1:
1285                 si = cs[ii,1]; sr = cs[ir,1]
1286
1287                 e = NonLinearEmigration(si,idi[ii],sr,di[ir],il,l,
1288                               ↪ a,z)
1289                 ga = StochasticFluctuations(s,e) if s>0 else 0
1290
1291                 cs[ii,1] = si*(1-e+ga)
1292                 cs[ir,1] = sr+si*e
1293
1294             # In the exact case it's reasonable to check whether p
1295             # → is actually positive before evaluating the
1296             # → Bernoulli distribution with «np.random.random()<p»
1297             # → since A can have zero components
1298             # However its approximations Ap does not have zero
1299             # → components by definition (it's a complete network),
1300             # → hence that check becomes useless
1301
1302             # p = 1 if p>1 else (0 if p<0 else p)
1303             # theta = np.random.binomial(1,p)
1304
1305             @njit(cache=True)
1306             def FYDInPlaceShuffle(v,n):
1307                 for i in range(n-1,0,-1):
1308                     j = np.random.randint(0,i+1)
1309                     tmp = v[i]
1310                     v[i] = v[j]
1311                     v[j] = tmp
1312
1313                     # In randint i+1 is necessary to include the i-th index
1314
1315             # Fisher-Yates-Durstenfeld [in-place] shuffle

```

```

1301     # https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle#T ]
1302     ↵ he_modern_algorithm
1303
1304     @njit(cache=True)
1305     def NonLinearEmigration(
1306         si,idii, # Interacting city size
1307         sr,dir, # Receiving city size
1308         il,l,a,z
1309     ):
1310         if si <= 0:
1311             return 0
1312
1313         if il == 0:
1314             rs = sr/si
1315             efl = l*(rs**a)/(1+rs**a)
1316
1317             if z == 0:
1318                 return efl
1319             else:
1320                 if sr <= 0:
1321                     return 0
1322                 else:
1323                     irs = 1/rs
1324                     efs = l*(irs**a)/(1+irs**a)
1325                     return (1-z)*efl+z*efs
1326
1327         elif il == 1:
1328             rsk = (sr/si)*dir*idii
1329             efl = l*(rsk/a)/(1+rsk/a)
1330
1331             if z == 0:
1332                 return efl
1333             else:
1334                 if sr <= 0:
1335                     return 0
1336                 else:
1337                     irsk = 1/rsk
1338                     efs = l*(irsk/a)/(1+irsk/a)
1339                     return (1-z)*efl+z*efs
1340
1341         elif il == 2:
1342             rsk = (sr/si)*dir*idii
1343             efl = l*(rsk**a)/(1+rsk**a)
1344
1345             if z == 0:
1346                 return efl
1347             else:
1348                 if sr <= 0:
1349                     return 0
1350                 else:
1351                     irsk = 1/rsk
1352                     efs = l*(irsk**a)/(1+irsk**a)
1353                     return (1-z)*efl+z*efs
1354
1355         else:
1356             rsk = (sr/si)*dir*idii
1357             efl = l*(rsk/(1+rsk))**a
1358
1359             if z == 0:
1360                 return efl
1361             else:
1362                 if sr <= 0:
1363                     return 0
1364                 else:
1365                     irsk = 1/rsk
1366                     efs = l*(irsk/(1+irsk))**a
1367                     return (1-z)*efl+z*efs

```

```

1357     # rsk = (sr/si)*(dr*idi)  # Relative population*degree ratio
1358     # irsk = (si/sr)*(di*idr) # Inverse relative
1359     #   ↳ population*degree ratio
1360     # efl                      # Actual emigration rate for the
1361     #   ↳ lumping fraction
1362     # efs                      # Actual emigration rate for the
1363     #   ↳ separation fraction
1364
1365     # Numba does not officially support the match structure, hence,
1366     #   even if it appears to work, it's better to use an
1367     #   ↳ «if/elif/else» one for the sake of performance and stability
1368
1369 @njit(cache=True)
1370 def StochasticFluctuations(sigma,E):
1371     alpha = ((1-E)**2)/(sigma**2)
1372     beta = (sigma**2)/(1-E)
1373
1374     # if alpha<=1:
1375     #     raise ValueError("α must be >1 to have a non-degenerate
1376     #   ↳ gamma distribution, and thus always admissible
1377     #   ↳ fluctuations")
1378
1379     ga = np.random.gamma(alpha,beta) # Initial sampling
1380
1381     return ga+E-1 # Final left translation
1382
1383
1384
1385
1386
1387
1388 #### Discarded code ####

```

Listato A.7: Codice «mat2tex.mat».

```

1 function mat2tex(matpath,texpath)
2     s = load(matpath);
3     f = string(fieldnames(s.plots));
4
5     figure('Visible','off');
6     hold on
7
8     ax = gca;
9
10    grid on
11    ax.GridLineStyle = '--';
12    ax.YMinorGrid = 'off';
13    ax.XMinorGrid = 'off';
14
15    xlabel( ...
16        s.style.labels.x, ...
17        'interpreter','latex' ...
18    );
19    ylabel( ...
20        s.style.labels.y, ...
21        'interpreter','latex' ...
22    );
23
24    ax.XScale = s.style.scale.x;
25    ax.YScale = s.style.scale.y;
26
27    for i = 1:numel(f)
28        field = f(i);
29
30        switch s.plots.(field).t
31            case 'scatter'
32                h = scatter( ...
33                    double(s.plots.(field).x), ...
34                    double(s.plots.(field).y) ...
35                );
36

```

```

37 % h.SizeType = 16;
38 h.SizeType = double(s.plots.(field).s)+10;
39 h.DisplayName = s.plots.(field).l;
40 h.MarkerFaceColor = s.plots.(field).c;
41 h.MarkerFaceAlpha = double(s.plots.(field).a);
42 h.MarkerEdgeColor = "none";
43
44 case 'function'
45 h = plot( ...
46     double(s.plots.(field).x), ...
47     double(s.plots.(field).y) ...
48 );
49
50 h.Color = s.plots.(field).c;
51
52 if(s.plots.(field).m ~= "None")
53     h.Marker = s.plots.(field).m;
54
55 passo = 9;
56 num_punti = numel(s.plots.(field).x);
57 h.MarkerIndices = 1:passo:num_punti;
58 end
59
60 if(s.plots.(field).l == "")
61     h.HandleVisibility = "off";
62 else
63     h.DisplayName = s.plots.(field).l;
64 end
65
66 case 'histogram'
67 h = histogram( ...
68     'BinEdges',double(s.plots.(field).b), ...
69     'BinCounts',double(s.plots.(field).w) ...
70 );
71
72 h.DisplayName = s.plots.(field).l;
73 h.FaceColor = s.plots.(field).c;
74 h.FaceAlpha = double(s.plots.(field).a);
75 h.EdgeColor = 'None';
76
77 w = double(s.plots.(field).w);
78 wMin = min(w(w>0));
79 if(ax.YLim(1)>=wMin && s.style.scale.y == "log")
80     ax.YLim(1) = (10^(-0.3))*wMin;
81 end % 10^(-0.3) in logarithmic scale is a downward
→ translation of the equivalent -0.3 in linear
→ scale: log10[(10^(-0.3))*wMin]=-0.3+log10(wMin)
82
83 case 'errorbar'
84 if(s.plots.(field).e == "y")
85     h = errorbar( ...
86         double(s.plots.(field).x), ...
87         double(s.plots.(field).y), ...
88         double(s.plots.(field).ye(1,:)), ...
89         double(s.plots.(field).ye(2,:)) ...
90     );
91 else
92     h = errorbar( ...
93         double(s.plots.(field).x), ...
94         double(s.plots.(field).y), ...
95         double(s.plots.(field).xe(1,:)), ...
96         double(s.plots.(field).xe(2,:)), ...
97         'horizontal' ...
98     );
99 end
100
101 h.LineStyle = "none";

```

```

102         h.HandleVisibility = "off";
103         h.Color = s.plots.(field).c;
104         h.Marker = 'none';
105         h.CapSize = 0;
106
107         h.UserData = sprintf('every error bar/.append styl」
108             ↪  e={opacity=%2f}',double(s.plots.(field).a));
109
110     case 'functionfill'
111         if(s.plots.(field).e == "y")
112             h = fill( ...
113                 [double(s.plots.(field).x),flip(double(s.p」
114                     ↪  lots.(field).x))], ...
115                     [double(s.plots.(field).y)+double(s.plots.」
116                     ↪  (field).ye(2,:)), ...
117                     flip(double(s.plots.(field).y)-double(s.pl」
118                         ↪  lots.(field).ye(1,:))), ...
119                     sscanf(s.plots.(field).c(2:end),'%2x%2x%2x」
120                         ↪  ',[1 3])/255 ...
121             );
122         else
123             h = fill( ...
124                 [double(s.plots.(field).x)+double(s.plots.」
125                     ↪  (field).xe(2,:)), ...
126                     flip(double(s.plots.(field).x)-double(s.pl」
127                         ↪  lots.(field).xe(1,:))), ...
128                     [double(s.plots.(field).y),flip(double(s.p」
129                         ↪  lots.(field).y))], ...
130                     sscanf(s.plots.(field).c(2:end),'%2x%2x%2x」
131                         ↪  ',[1 3])/255 ...
132             );
133         end
134
135         h.FaceAlpha = double(s.plots.(field).a);
136         h.HandleVisibility = "off";
137         h.EdgeColor = "none";
138
139         if(s.plots.(field).h ~= "None")
140             switch s.plots.(field).h
141                 case "|"
142                     h.UserData = 'postaction={pattern={Lin」
143                         ↪  es[angle=90,distance={5pt},line
144                             ↪  width={0.5pt}}}, pattern
145                             ↪  color=mycolor1!60}';
146                 case "/"
147                     h.UserData = 'postaction={pattern={Lin」
148                         ↪  es[angle=45,distance={5pt},line
149                             ↪  width={0.5pt}}}, pattern
150                             ↪  color=mycolor1!60}';
151             end
152         end
153     end
154
155     if(s.style.legend == 1)
156         legend( ...
157             'Location','northeast',...
158             'interpreter','latex' ...
159         );
160     end
161
162     try
163         cleanfigure('handle',gcf,'targetResolution', 600);
164     catch
165         warning('Cleanfigure skipped');

```

```
152     end
153
154     ishistogram = isfield(s.plots,"histogramPlot") ||
155         isfield(s.plots,"histogramPlot1"); % If there is at least a
156         histogram plot
157
158     if(ishistogram && s.style.scale.y == "log")
159         matlab2tikz( ...
160             texpath, ...
161             'showinfo',false, ...
162             'standalone',false, ...
163             'floatFormat','%.5g', ...
164             'parseStrings', false, ...
165             'extraAxisOptions', {'log origin=infty'} ...
166         ); % This correction is necessary to avoid having upside
167             down bins in the «.tex» file when the y-scale is
168             logarithmic
169     else
170         matlab2tikz( ...
171             texpath, ...
172             'showinfo',false, ...
173             'standalone',false, ...
174             'floatFormat','%.5g', ...
175             'parseStrings', false ...
176         );
177     end
178 end
```

ELENCO DELLE FIGURE

1209	Figura 1.1	Funzione Lognormale(0,1).	2
1210	Figura 2.1	Esempio di un grafo indiretto, della sua equivalente forma diretta [simmetrica] e della loro [identica] matrice d'adiacenza.	5
1211	Figura 2.2	Forza contro grado per la Sardegna.	9
1212	Figura 2.3	Riproduzione dei principali grafici di [6] [a cui si rimanda per maggiori dettagli sui vari coefficienti] relativi alla topologia della rete del pendolarismo sarda.	11
1213	Figura 3.1	Transizione della densità gamma verso l'asimmetria con $\lambda = 0.1$ ed $E = \lambda$, da cui $\langle \hat{\gamma} \rangle = 0.9$ e $\beta = \langle \hat{\gamma} \rangle / \alpha$	32
1214	Figura 3.2	Schema riassuntivo del § 3.3.2.	37
1215	Figura 3.3	Rappresentazione grafica dell'approssimazione.	37
1216	Figura 4.1	Confronto di una simulazione con e senza fluttuazioni; la regola d'emigrazione è la (4.5) mentre i parametri sono illustrati nella Tab. 4.1.	50
1217	Figura 4.2	Studio della configurazione di riferimento stabile della $[RE]_S$ con parametri dalla Tab. 4.3; per la spiegazione dei grafici si veda il § 4.2.3.	52
1218	Figura 4.3	Studio della configurazione di riferimento instabile della $[RE]_S$ con parametri dalla Tab. 4.3; per la spiegazione dei grafici si veda il § 4.2.3.	53
1219	Figura 4.4	Studio della configurazione di riferimento della $([RE]_{SK})$ con parametri dalla Tab. 4.4; per la spiegazione dei grafici si veda il § 4.2.3.	54
1220	Figura 4.5	Studio della configurazione di riferimento della $([RE]_{SK}^f)$ con parametri dalla Tab. 4.5; per la spiegazione dei grafici si veda il § 4.2.3.	55
1221	Figura 4.6	Studio della configurazione di riferimento della $[RE]_{SW}$ con parametri dalla Tab. 4.6; per la spiegazione dei grafici si veda il § 4.2.3.	56
1222			
1223			
1224			
1225			
1226			
1227			
1228			
1229			
1230			
1231			
1232			
1233			
1234			
1235			
1236			
1237			
1238			
1239			

ELENCO DELLE TABELLE

1241	Tabella 2.1	Formattazione di una sola riga nei dati ISTAT ¹ ; le linee barrate corrispondono a dati trascurati.	8
1242	Tabella 2.2	Confronto con [6] dei pesi maggiori.	9
1243	Tabella 4.1	Parametri della Fig. 4.1.	49
1244	Tabella 4.2	Dati e parametri della Sardegna.	51
1245	Tabella 4.3	Dati e parametri della [RE] _S	51
1246	Tabella 4.4	Dati e parametri della [RE] _{SK}	51
1247	Tabella 4.5	Dati e parametri della [RE] _{SK} ^f	57
1248	Tabella 4.6	Dati e parametri della [RE] _{SW}	57

ELENCO DEI LISTATI

1251	Listato A.1	Codice «main.py».	61
1252	Listato A.2	Codice «libParameters.py».	62
1253	Listato A.3	Codice «libData.py».	67
1254	Listato A.4	Codice «libFigures.py».	74
1255	Listato A.5	Codice «libNetworks.py».	89
1256	Listato A.6	Codice «libKTMAS.py».	97
1257	Listato A.7	Codice «mat2tex.mat».	119

ELENCO DEGLI ALGORITMI

1259	Algoritmo 1	Algoritmo [AR] _S di tipo Nanbu-Babovsky .	42
------	-------------	--	----

BIBLIOGRAFIA

- [1] Felix Auerbach. «Das Gesetz der Bevölkerungskonzentration [The law of population concentration]». In: *Petermanns Geographische Mitteilungen* 59 (1913), pp. 74–76.
- [2] Albert-László Barabási, Réka Albert & Hawoong Jeong. «Mean-field theory for scale-free random networks». In: *Physica A: Statistical Mechanics and its Applications* 272.1-2 (1999), pp. 173–187.
- [3] Marc Barthélémy. «Spatial networks». In: *Physics Reports* 499.1 (2011), pp. 1–101. ISSN: 0370-1573. DOI: <https://doi.org/10.1016/j.physrep.2010.11.002>. URL: <https://www.sciencedirect.com/science/article/pii/S037015731000308X>.
- [4] Marcus Berliant & Axel H Watanabe. «A scale-free transportation network explains the city-size distribution». In: *Quantitative Economics* 9.3 (2018), pp. 1419–1451.
- [5] Anna D Broido & Aaron Clauset. «Scale-free networks are rare». In: *Nature communications* 10.1 (2019), p. 1017.
- [6] Andrea De Montis et al. «The structure of interurban traffic: a weighted network analysis». In: *Environment and Planning B: Planning and Design* 34.5 (2007), pp. 905–924.
- [7] Rick Durrett. *Probability: theory and examples*. Vol. 49. Cambridge university press, 2019.
- [8] Jan Eeckhout. «Gibrat's law for (all) cities». In: *American Economic Review* 94.5 (2004), pp. 1429–1451.
- [9] Stefano Gualandi & Giuseppe Toscani. «Human behavior and lognormal distribution. A kinetic description». In: *Mathematical Models and Methods in Applied Sciences* 29.04 (2019), pp. 717–753.
- [10] Stefano Gualandi & Giuseppe Toscani. «Size distribution of cities: A kinetic explanation». In: *Physica A: Statistical Mechanics and its Applications* 524 (2019), pp. 221–234.
- [11] International Standards Organisation. *ISO 80000-3 Quantities and units—Part 3: Space and time*.
- [12] ISTAT. *Basi territoriali e variabili censuarie*. Riferimento specifico al censimento della popolazione e delle abitazioni del 1991. 2024. URL: <https://www.istat.it/notizia/basi-territoriali-e-variabili-censuarie/> (visitato il giorno 12/02/2026).
- [13] ISTAT. *Confini delle unità amministrative a fini statistici al 1° gennaio 2018. Dati storici (1991)*. Riferimento specifico ai dati del 1991. 2025. URL: <https://www.istat.it/notizia/confini-delle-unita-amministrative-a-fini-statistici-al-1-gennaio-2018-2/> (visitato il giorno 12/02/2026).

- 1300 [14] ISTAT. *Matrici di contiguità, distanza e pendolarismo. Dati storici (1991)*.
1301 Riferimento specifico ai dati della matrice di pendolarismo del 1991.
1302 2025. URL: <https://www.istat.it/notizia/matrici-di-contiguita-distanza-e-pendolarismo/> (visitato il giorno 12/01/2026).
- 1304 [15] Nadia Loy & Andrea Tosin. «A viral load-based model for epidemic
1305 spread on spatial networks». In: *arXiv preprint arXiv:2104.12107* (2021).
1306 URL: <https://arxiv.org/abs/2104.12107>.
- 1307 [16] Nadia Loy & Andrea Tosin. «Essentials of the kinetic theory of multi-
1308 agent systems». In: *arXiv preprint arXiv:2503.11554* (2025). URL: <https://arxiv.org/abs/2503.11554>.
- 1310 [17] Marco Nurisso, Matteo Raviola & Andrea Tosin. «Network-based ki-
1311 netic models: Emergence of a statistical description of the graph topo-
1312 logy». In: *European Journal of Applied Mathematics* (2024), pp. 1–22. DOI:
1313 [10.1017/S0956792524000020](https://doi.org/10.1017/S0956792524000020).
- 1314 [18] Lorenzo Pareschi & Giuseppe Toscani. *Interacting multiagent systems:*
1315 *kinetic equations and Monte Carlo methods*. OUP Oxford, 2013.
- 1316 [19] George Kingsley Zipf. *Human behavior and the principle of least effort: An*
1317 *introduction to human ecology*. Ravenio books, 2016.