UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Corso di Laurea Magistrale in Scienze Economiche e Finanziarie

MODELLO DI STIMA DELLE PROBABILITÀ DI PRECIPITAZIONI E APPLICAZIONI AL SETTORE ASSICURATIVO

RAINFALL MODELLING AND INSURANCE APPLICATIONS

Relatore:
Prof.ssa Maria Cristina Recchioni
Correlatore:
Prof. Riccardo Lucchetti

Tesi di Laurea di:
Federico Fiorani

Anno Accademico 2023 – 2024

Indice

I INTRODUZIONE

Il nostro paese ed in particolare il nostro territorio si trova sempre più frequentemente ad affrontare eventi catastrofali naturali che hanno importanti ripercussioni, oltre che sulla vita, la sicurezza e quindi la serenità dei cittadini che ne sono coinvolti, sul tessuto economico e finanziario del territorio.

Il processo di cambiamento climatico, che si sta osservando, comporta il verificarsi di fenomeni meteorologici che in passato venivano ritenuti rari, che ora sono più frequenti, e che lo saranno ancora di più in futuro[1]. Questo cambiamento di scenario impone a famiglie ed imprese di porre in essere azioni a protezione dei loro beni che da un punto di vista finanziario non può che passare per il settore assicurativo.

Il tema delle coperture assicurative contro i danni catastrofali è stato oggetto dell'attività politica del nostro paese che ha inserito nella Legge n. 213 del 30 dicembre 2023 (legge di Bilancio 2024) disposizioni volte ad aumentare la copertura per le imprese contro rischio di calamità naturali. La norma prevede per tutte le imprese con sede legale, o stabile organizzazione, in Italia l'obbligo di stipulare entro il 31 dicembre 2024 polizze a copertura dei danni, specificando le voci di bilancio dei beni soggetti all'obbligo, contro i seguenti eventi: terremoti, alluvioni, frane, inondazioni ed esondazioni. Viene inoltre istituito un fondo a sostegno delle compagnie assicurative.

La maggiore sensibilità delle famiglie ed imprese al rischio degli eventi climatici e l'attenzione della politica all'obbligo di sottoscrizione delle polizze devono trovare un compromesso con la natura difficilmente assicurabile di questi prodotti. Infatti, il contratto di assicurazione pone il suo funzionamento sulla diversificazione di portafoglio che consente la distribuzione degli indennizzi nel tempo. Gli eventi catastrofali, invece, si caratterizzano per la concentrazione

[1]IPCC, 2021: Summary for Policymakers. In: Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Masson-Delmotte, V., P. Zhai, A. Pirani, S.L. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M.I. Gomis, M. Huang, K. Leitzell, E. Lonnoy, J.B.R. Matthews, T.K. Maycock, T. Waterfield, O. Yelekçi, R. Yu, and B. Zhou (eds.)]. In Press.

questi sia temporalmente che geograficamente e la loro scarsa frequenza rende complicato il processo di stima delle probabilità [Monti et al., 2012].

La maggiore pericolosità dell'assunzione del rischio, legata anche alla maggiore imprevedibilità dei fenomeni dovuta al cambiamento climatico, è stata segnalata anche direttamente da operatori del settore con i quali ci si è confrontati, che hanno segnalato come pongono sempre più maggiore precauzione nella stipula di questi contratti. Questi incontri hanno segnalato l'emergere di una nuova tipologia di contratto assicurativo che si sta diffondendo anche per questo tipo di rischi: le polizze parametriche. Si tratta di una polizza che si è evoluta e diffusa grazie ai progressi tecnologici nella raccolta dei dati e nei sistemi di comunicazione, noti come *fintech*.

Il glossario redatto dall'IVASS, definisce le polizze parametriche come "dei contratti di assicurazione in cui l'erogazione dell'indennizzo è correlata al verificarsi di un indice predeterminato (benchmark di riferimento), il cui andamento è costantemente registrato e monitorato da "soggetti terzi" rispetto alla compagnia o all'assicurato per garantire l'indipendenza."[2].

Fondamentale, è evidente, l'importanza del benchmark che "deve esprimere la correlazione, scientificamente validata, tra il fatto e le conseguenze dannose che ne derivano" affinché la somma liquidata risulti proporzionata al danno [Petrosino, 2023]. In altre parole è necessario scegliere come benchmark, se non è possibile utilizzare direttamente l'evento dannoso, un fenomeno osservabile il più vicino possibile a questo. A titolo di esempio, per un impianto sciistico il parametro *millimetri di neve* sarebbe sia l'evento contro il quale indennizzarsi, nel caso i millimetri siano inferiori ad una determinata soglia, sia il fenomeno stesso utilizzabile come benchmark. Mentre in una polizza contro l'allagamento, non essendo il fenomeno direttamente ed oggettivamente osservabile, è possibile prendere come benchmark i millimetri di pioggia caduti nell'area di interesse.

Risulta altrettanto evidente l'analogia che queste polizze hanno con gli strumenti finanziari derivati. La differenza la si riscontra nella funzione economica

[2]IVASS, Glossario, allegato al Bollettino Statistico. L'attività assicurativa nel comparto "property" e nel ramo r.c. in generale (2013-2018), VII, n. 3, marzo 2020, p. 16 ss

e sociale che i due contratti svolgono. Mentre la causa del contratto di assicurazione contro i danni è il trasferimento del rischio, nel strumento finanziario si ha una "mera assunzione del rischio di scostamento di un certo valore correlato ad un evento dell'indice predefinito'[Santagata, 2022].

Anche se dal lato giurisprudenziale la polizza parametrica è chiaramente distinta da un derivato, dal punto di vista modellistico si comporta esattamente allo stesso modo, consentendo così di poter fare riferimento alla letteratura sui derivati atmosferici per la stima del premio.

Per i derivati atmosferici il calcolo del prezzo del contratto è relativamente semplice nell'impostazione come mostrano, ad esempio Considine e Geoffrey [Considine, 2000], il prezzo è direttamente definito dalla distribuzione di probabilità dell'evento sottostante. Il procedimento, in generale, è il seguente:

1. si raccolgono i dati storici del fenomeno che è il sottostante al contratto (precipitazioni, temperature, ...);
2. dai dati si definisce la distribuzione di probabilità che questi assumono;
3. si calcola il premio;

Il presente studio mira a definire un modello di previsione per le precipitazioni sul territorio italiano che possa essere applicato come strumento per calcolare il premio delle polizze parametriche che hanno tra i benchmark le precipitazioni.

II IL MODELLO DI PREVISIONE

Il modello utilizzato è un modello di mistura che descrive la probabilità delle precipitazioni tramite una distribuzione gamma. La scelta è ricaduta su questa distribuzione per le proprietà che possiede: il supporto è l'insieme dei numeri reali positivi, come il supporto della variabile da descrivere (volume pioggia) ed è descritta da due parametri: *shape* e *scale*. La scelta dei parametri rendono la distribuzione gamma estremamente flessibile ovvero capace di includere ben note distribuzioni. [Wilks, 1990].

II.1 LA DISTRIBUZIONE GAMMA

La distribuzione di probabilità gamma è una distribuzione continua con supporto a valori strettamente positivi. È caratterizzata da due parametri principali: il parametro di forma (*shape*) ed il parametro di scala (*scale* o *rate*). I parametri *scale* e *rate* sono legati, sono uno l'inverso dell'altro.

Se si utilizzano *shape* (k) e *scale* (θ), la sua funzione di densità di probabilità è

$$f(x; k, \theta) = \frac{1}{\theta^k \Gamma(k)} x^{k-1} e^{-\frac{x}{\theta}}, \quad x \in (0, +\infty).$$

La distribuzione gamma si distingue da altre distribuzioni di probabilità perché i suoi parametri non sono direttamente interpretabili come caratteristiche visibili della distribuzione stessa, come accade per altre distribuzioni come la distribuzione normale con media e deviazione standard.

Il parametro k, la *shape*, determina principalmente la simmetria della distribuzione:

- quando si ha $k > 1$ la distribuzione è caratterizzata da un tratto iniziale crescente, un massimo e un seguente tratto decrescente. All'aumentare del parametro la curva diventa progressivamente più simmetrica;
- per $k = 1$ la distribuzione diventa una esponenziale negativa;

- nel caso $k < 1$ la distribuzione è strettamente decrescente e presenta un asintoto verticale sull'asse delle ordinate ed uno orizzontale sull'asse delle ascisse.

Il parametro θ, la *scale*, influenza la dispersione dei valori della distribuzione gamma:

- aumentando θ la distribuzione si sposta a destra e si espande. Questo comporta l'aumento della dispersione dei valori che la variabile può assumere e del valore atteso;
- diminuendo θ la distribuzione si comprime spostandosi a sinistra, riducendo la dispersione e media.

Viene riportato di seguito, a fini illustrativi, un grafico dove sono disegnate 3 diverse distribuzioni gamma tutte con lo stesso valore di $k = 0.6$ e diversi valori di θ. I valori sono stati scelti sulla base dei dati ottenuti dallo studio.

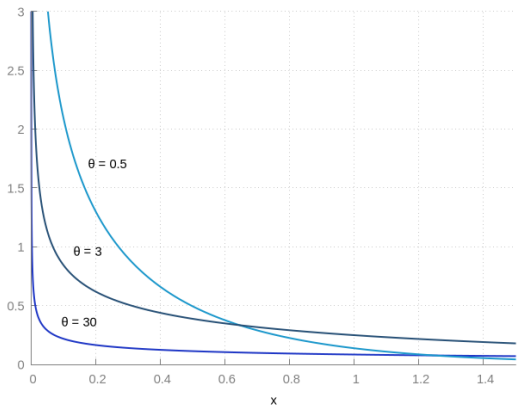


Figura 1: Grafico delle distribuzioni $\Gamma(k = 0.6, \theta = 0.5)$, $\Gamma(k = 0.6, \theta = 3)$, $\Gamma(k = 0.6, \theta = 30)$.

II.2 DESCRIZIONE DEL MODELLO

Il modello, come anticipato, di mistura, prevede una separazione iniziale tra i giorni in cui si verifica una precipitazione e quelli in cui non si verifica, per poi procedere alla stima delle precipitazioni cumulate per i soli giorni in cui si sono registrate piogge. Questa divisione iniziale, come già proposto da Husak, Michaelsen e Funk [Husak et al., 2007], si è resa necessaria in quanto la distribuzione gamma non ammette valori minori o uguali a zero. La novità introdotta dal modello proposto è quella di creare un legame tra il parametro θ della distribuzione gamma e la probabilità di precipitazione, stimando poi contemporaneamente i parametri attraverso la massimizzazione della funzione obbiettivo.

Il modello può essere così riassunto.

Fissata la base di Fourier discreta

$$\{\cos(2\pi kt);\ \sin(2\pi kt)\} \qquad k = 1, 2, \ldots,$$

si definiscono le matrici X e Z le quali hanno la t-esima riga così definita

$$\begin{aligned} X_t = [1, \Delta t_t, \cos(2\pi t), \sin(2\pi t), \cos(4\pi t), \sin(4\pi t), \\ \ldots, \cos(24\pi nt), \sin(24\pi nt)] \\ Z_t = [1, \Delta t_t, \cos(2\pi t), \sin(2\pi t), \cos(4\pi t), \sin(4\pi t)] \end{aligned}$$

indicando con Δt_t il tempo passato dalla prima osservazione, espresso in frazione di anno. Si definisce poi

$$y_t = \begin{cases} g_t(k, \theta_t) & p_t \\ 0 & 1 - p_t \end{cases} \tag{1}$$

dove

$$p_t = \sigma(X_t \cdot \underline{\gamma}_1)$$

$$g_t(y) = \frac{1}{\Gamma(k)\theta_t^k} \cdot y^{k-1} \cdot e^{-y/\theta_t} \qquad y \in (0; +\infty),$$

con

$$\theta_t = e^{Z_t \cdot \underline{\gamma}_2},$$

$$\sigma(x) = \frac{1}{1+e^{-x}},$$

$$\underline{\gamma}_1^T = (a_1, b_1, \ldots, a_{26}, b_{26}),$$

$$\underline{\gamma}_2^T = (c_1, d_1, \ldots, c_6, d_6).$$

Indicando con y_t l'osservazione al tempo t, la distribuzione proposta implica che

$$E[y_t | y_t > 0] = \frac{\theta_t}{k}$$

e

$$E[y_t] = p_t \cdot \frac{\theta_t}{k}.$$

Quindi i valori di γ_2 sono interpretabili come gli effetti marginali di $\ln(y_t)$

II.3 STIMA DEI PARAMETRI

I parametri oggetto di stima sono: $\underline{\gamma}_1$, $\underline{\gamma}_2$ e k.

La stima di questi avviene tramite funzione di quasi massima verosimiglianza (QML). Questo permette di difendere la scelta della funzione gamma come distribuzione delle precipitazioni. Infatti, anche qualora la distribuzione vera dei dati fosse differente, questo non implica necessariamente che i parametri siano stati stimati in modo inconsistente [Anatolyev and Gospodinov, 2011].

La massimizzazione della QML, eseguita con le opportune specificazioni, con un numero sufficiente di osservazioni e rispettando le condizioni di regolarità, assicura che il valore dei parametri stimati converga a quello vero.

La stima della quasi massima verosimiglianza viene eseguita computazionalmente in modo iterativo. È necessario, quindi, inizializzare i parametri a

valori che permettano una rapida ricerca del punto di massimo della funzione di verosimiglianza.

Per inizializzare il vettore γ_1 si procede prima alla definizione del vettore binario $\underline{w}$ definito

$$w_t = \begin{cases} 1 & y_t > 0 \\ 0 & altrimenti \end{cases}$$

e si esegue una regressione logistica[3] su questo, con regressori le colonne della matrice X, e si assegnano a $\underline{\gamma}_1$ i coefficienti della regressione.

$$E[\underline{w}|X] = \Lambda(X^T \cdot \underline{\gamma}_1)$$

Il vettore $\underline{\gamma}_2$ e il parametro k sono inizializzati come segue:

$$\underline{\gamma}_2^T = \left(-\ln\left(\frac{E[Y]}{V[Y]}\right), 0, 0, 0, 0, 0 \right),$$

$$k = \frac{E[Y]^2}{V[Y]}.$$

I valori di inizializzazione definiti per k e $\underline{\gamma}_2$ sono stati ottenuti tenendo presente i seguenti legami tra i parametri k e θ e il valore atteso e varianza:

$$E[Y] = k \cdot \theta$$

$$V[Y] = k \cdot \theta^2.$$

[3]La regressione logistica è un modello di regressione che prende una variabile dipendente binaria ed un generico set di variabili esplicative. La stima fornita può essere interpretata come la probabilità p di realizzazione della variabile dipendente.

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

La probabilità di successo è data da

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}}$$

I coefficienti beta vengono stimati con il metodo della massima verosimiglianza.

Invertendo le equazioni si ottengono le formule usate per l'inizializzazione:

$$\theta = \frac{E[Y]}{V[Y]}$$
$$k = \frac{E[Y]^2}{V[Y]}.$$

La funzione da massimizzare per la stima dei parametri ha la seguente espressione:

$$\ln \mathcal{L}\left(\underline{\gamma}_1, \underline{\gamma}_2, k | \{y_t\}_{t=1}^n\right) = \begin{cases} \ln\left(1 - \sigma(X_t \cdot \underline{\gamma}_1)\right) & y_t = 0 \\ \ln\left(\sigma(X_t \cdot \underline{\gamma}_1)\right) + \tilde{g}(y_t, k, \tilde{\theta}) & altrimenti \end{cases} \tag{2}$$

con

$$\tilde{\theta} = e^{-X_t \cdot \underline{\gamma}_2}$$

$$\tilde{g}(y_t, k, \tilde{\theta}) = \begin{cases} k \cdot \ln(\tilde{\theta}) - \ln\left(\Gamma\left(k\right)\right) + (k-1) \cdot \ln(x) + x \cdot \tilde{\theta} & y_k \geq 0 \\ 0 & altrimenti \end{cases}.$$

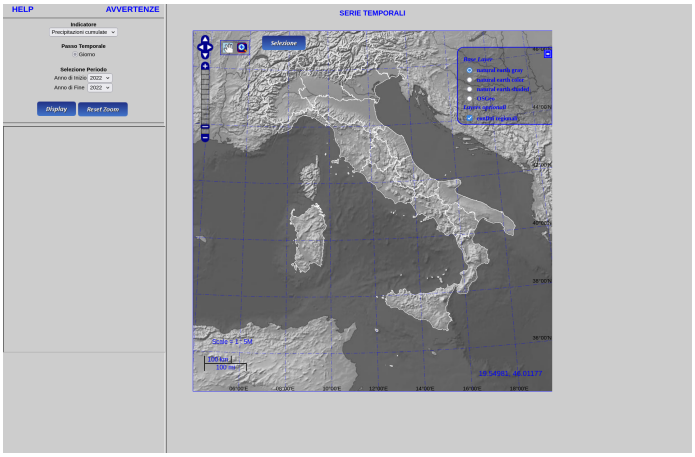
Nel presente studio la ricerca del massimo è stata condotta tramite il software econometrico *Gretl*[4] attraverso il comando `mle`[5] con il flag `--robust`. Il codice utilizzato è riportato nello script VI.1 in appendice. La Funzione 2 applica un'ottimizzazione simultanea sia dei coefficienti di $\underline{\theta}_1$ che sono usati per la stima di p_t, sia di quelli di $\underline{\theta}_2$ e k, parametri della gamma.

[4] `https://gretl.sourceforge.net/`
[5] `https://gretl.sourceforge.net/gretl-help/cmdref.html#mle`

III COSTRUZIONE DEL DATASET

III.1 DOWNLOAD ED ELABORAZIONE DEI DATI

I dati delle precipitazioni giornaliere, sui quali si sono eseguite le elaborazioni, sono stati raccolti dal "Sistema nazionale per la raccolta, elaborazione e diffusione di dati climatici"[6], realizzato dall' "Istituto Superiore per la Protezione e la Ricerca Ambientale" (ISPRA). Alla pagina "Dati ed Indicatori" è possibile accedere alla piattaforma di download delle serie giornaliere di precipitazioni. La pagina permette di scaricare i dati procedendo per passi:



Schermata del portale ISPRA da cui sono stati scaricati i dati.

1. selezionare l'arco temporale di interesse;
2. cliccare su "Display". A questo punto sulla mappa compaiono tutte le stazioni disponibili per il periodo selezionato;
3. cliccare sul pulsante "Selezione";
4. selezionare con il mouse le stazioni che si vogliono scaricare;
5. cliccare sul pulsante "Download";
6. salvare un file CSV con i dati delle stazioni selezionate.

[6] `https://scia.isprambiente.it/`

Il sito non fornisce API o interfacce per l'accesso ai dati ma permette solo il download manuale. Sì è proceduto a dividere l'Italia in rettangoli di 0.5 per 0.5 gradi di latitudine e longitudine, selezionare le stazioni all'interno dell'aera e scaricarle salvando ogni file con un nome corrispondente al limite est di longitudine e sud di latitudine. Ad esempio, i dati delle stazioni comprese nella regione individuata dalle longitudini 12° E e 12° 30' E e le latitudini 43° N e 43° 30' N saranno salvati nel file "`4300n1230e.csv`". Questo passaggio è tornato utile in fase di elaborazione dei dati per assegnare ad ogni stazione le relative coordinate.

Alla termine del download si sono ottenuti 208 file in formato CSV contenenti i seguenti campi:

- Rete: La rete da cui hanno origine i dati.
- Anagrafica: il nome della stazione.
- Data: la data dell'osservazione in formato `mmm-dd-yyyy`. Ad esempio la data "1 gennaio 1970" è indicata "JAN-01-1970".
- Precipitazioni cumulate: i millimetri di pioggia osservati nella giornata.

Inizialmente, si è provato a lavorare direttamente con i file di testo cercando di consolidarli in un unico file CSV per lavorare direttamente con questo successivamente. Tuttavia, la dimensione del file comportava un problema. I 208 file CSV individuali avevano una dimensione totale di circa 4.3GB. La dimensione non costituiva un problema un problema in fase di elaborazione dati, poiché il software poteva comunque agilmente gestire il file. Il problema principale è emerso durante il processo di pulizia dei dati, che ha richiesto una considerevole quantità di tempo.

Per la pulizia si è dovuto procedere con:

- la correzione di valori assenti: in alcuni file erano presenti delle righe vuote che si è proceduto ad eliminare;

- alcune stazioni erano stati salvanti con una virgola nel nome come ad esempio “S, ANDREA”. Questo comportava l’interpretazione di un campo extra che non poteva essere gestito dal software. I file CSV, prevedono l’uso di un carattere, generalmente la virgola, per delimitare i campi, i quali devono obbligatoriamente avere lo stesso numero in tutte le righe del documento;

- alcuni nomi presentavano errori di codifica dei caratteri che sono stati corretti.

Le operazioni elencate sono state eseguite con i comandi `grep`, `sed` e `awk`[7].

Si è poi proceduto alla rimozione delle stazioni ed osservazioni duplicate. Il numero di stazioni duplicate tra i vari file era sicuramente importante a causa della modalità di download. Questa, non permette una precisa selezione delle sole stazioni all’interno dell’aera di interesse. Si è preferito quindi selezionare, per ogni area di riferimento, una regione leggermente più ampia in modo da avere da un lato la sicurezza di non aver omesso nessuna stazione ma dall’altro diversi dati duplicati nei vari file.

La rimozione dei duplicati tramite i comandi sopra elencati sarebbe stata molto dispendiosa in termini di tempo e risorse computazionali. Infatti si rendeva necessaria per ogni stazione la continua lettura di ogni file per intero.

Si è quindi reso necessario l’utilizzo di un database per la gestione delle osservazioni.

[7] Il comando `grep` permette di cercare stringhe all’interno di file di testo. Funziona leggendo riga per riga del file e stampando a monitor le righe che corrispondono al testo o pattern specificato.

Il comando `sed` è uno stream editor utilizzato per la modifica di file di testo. Legge l’input passato riga per riga e applica una serie di comandi specificati, come sostituzioni, cancellazioni, inserimenti o trasformazioni di testo ed è spesso utilizzato per eseguire modifiche su file senza doverli aprire manualmente.

`awk` è un potente linguaggio di programmazione orientato al testo, usato per elaborare e analizzare file di testo e flussi di dati. Legge l’input riga per riga, suddivide ogni riga in campi basati su delimitatori predefiniti e permette di eseguire operazioni su questi campi. È particolarmente utile per la trasformazione di dati e filtraggio di informazioni. Supporta inoltre condizioni e cicli.

Tutti e tre i comandi supportano l’uso di espressioni regolari che li rendono estremamente potenti e flessibili nella ricerche complesse.

III.2 CREAZIONE DEL DATABASE

I dati raccolti sono stati salvati in un database SQLite [8]. Si è scelto SQLite in quanto rappresenta un database leggero, che elimina la necessità di articolate configurazioni o installazioni di server. Dispone di interfaccia SQL standard, che consente di gestire i dati tramite query SQL. È particolarmente adatto per applicazioni che devono gestire quantità moderate di dati, come nel presente caso. I dati sono memorizzati in un singolo file di database ma vengono comunque offerte allo stesso tempo prestazioni ottimali e una gestione efficiente delle risorse.

Il database ha la seguente struttura delle tabelle.

Reti	
id_rete	INTEGER PRIMARY KEY
rete	TEXT

Stazioni	
id_stazione	INTEGER PRIMARY KEY
id_rete	INTEGER
stazione	TEXT
latitudine	REAL
longitudine	REAL
quota	REAL
n_oss	INTEGER
prima_oss	INTEGER
ultima_oss	INTEGER

Precipitazioni	
id_stazione	INTEGER
data	INTEGER
precipitazioni	REAL
anno	INTEGER
mese	INTEGER
giorno	INTEGER
rete	TEXT

Nella tabella *Stazioni*, i campi `n_oss`, `prima_oss` e `ultima_oss`, così come nella tabella *Precipitazioni* i campi `giorno`, `mese` e `anno`, rappresentano infor-

[8] `https://www.sqlite.org/`

mazioni aggiuntive che, sebbene possano essere ricavate tramite apposite query, sono state incluse per facilitare in seguito la selezione e l'analisi dei dati. La presenza di questi campi permette una consultazione più efficiente e immediata, evitando la necessità di calcoli o estrazioni costose, in termini di tempo, al momento della ricerca specifica dei dati. Per le tabelle *Precipitazioni* e *Stazioni* sono stati creati appositi indici al fine di velocizzare le operazioni di ricerca e migliorare l'efficienza delle interrogazioni. La creazione di indici consente un accesso rapido ai dati desiderati, riducendo significativamente i tempi di elaborazione e migliorando le prestazioni complessive del database.

Una volta predisposto il database si è proceduto con l'inserimento tramite uno script in Python[9] il cui comportamento può essere riassunto come segue:

1. viene inizializzato un oggetto `GestoreStazioni` il quale inizializza una variabile `listaStazioni`, un array di oggetti `Stazione` ognuno contente le informazioni sulle stazioni recuperate dal portale ISPRA;

2. per ogni file CSV scaricato si costruisce l'array delle stazioni presenti in questo;

3. per ogni stazione:

    3.1. si corregge la data delle osservazioni scrivendola secondo il formato ISO8601 [10]

    3.2. si controlla se è già presente nel database;

    3.3. se non è nel database si guarda se è in `listaStazioni`. Se viene trovata si recuperano le informazioni su latitudine, longitudine e quota e si aggiunge la stazione, la rete nel caso non fosse già presente, ed i dati sulle precipitazioni di questa stazione presenti nel file che si sta attualmente elaborando;

    3.4. se non è presente in `listaStazioni` si prova a vedere se in questa la stazione sia presente con un nome simile. Per fare questo si utilizza

[9] `https://www.python.org/`
[10] https://www.iso.org/standard/40874.html

la "distanza di Levenshtein"[11], si veda lo script VI.3 in appendice. Viene calcolata la distanza con tutte le stazioni presenti nella lista e scelta quella con la distanza minore, che vuol dire che la somiglianza tra i nomi è massima. Un esito positivo della ricerca non è però sufficiente. È infatti possibile che la stazione trovata abbia un nome simile a quella per cui si stanno cercando le informazioni, ma che si trovi completamente da altra parte. Viene in aiuto in questo caso la suddivisione iniziale del territorio per scaricare i dati. Si hanno ha disposizione delle coordinate approssimative per la stazione. Il nome del file da cui è stata precedentemente estratta ci dà informazioni con errore massimo di un grado, sia in latitudine che in longitudine, per le coordinate. È possibile quindi confrontare le coordinate della stazione trovata, con la distanza di Levenshtein minore, e vedere se compatibili con quelle estratte dal nome del file. Se compatibili la stazione con latitudine, longitudine e quota viene inserita nel database assieme agli altri dati sulle precipitazioni;

Se la stazione trovata non fosse compatibile, o non ne venisse trovata alcuna, si passa al punto successivo;

3.5. come ultima ricerca si tenta di recuperare informazioni da OpenStreetMap[12]. Viene costruito l'URL con il nome della stazione per la ricerca sul sito che restituisce un JSON[13] contente le località simili al nome fornito. Similmente a quanto fatto nel punto precedente si cerca tra i risultati uno compatibile per nome e coordinate.

[11] `https://en.wikipedia.org/wiki/Levenshtein_distance`

[12] `https://www.openstreetmap.org`

[13] JSON (JavaScript Object Notation) è un formato semplice per rappresentare e scambiare dati tra sistemi. JSON organizza le informazioni in modo chiaro e leggibile sia per le persone che per i computer. Ad esempio, per rappresentare le informazioni di un libro si scriverebbe il seguente file:

```
{
"titolo":  "Gödel, Escher, Bach",
"autore":  "Douglas Hofstadter",
"anno":  1979
}
```

Le informazioni sono presentate in coppie "chiave-valore" (come "titolo": "Gödel, Escher, Bach"). Questo formato è universale, quindi può essere utilizzato da diversi programmi e linguaggi di programmazione per inviare e ricevere dati in modo comprensibile e organizzato.

In caso la ricerca abbia esito positivo si procede ad aggiungere la stazione con i relativi dati al database. In questo caso però OpenStreetMap non fornisce informazioni relativamente all'altitudine. Viene quindi assegnato a questa il valore -1;

3.6. nel caso in nessuno dei precedenti si siano recuperate le informazioni della stazione viene assegnata alla sua latitudine, longitudine e quota il valore -1 e aggiunta al database.

III.3 STATISTICHE SUI DATI RACCOLTI

Il database, una volta elaborati i dati, è composto da 5 601 stazioni che fanno parte di 26 reti ed un totale di 75 579 808 di precipitazioni osservate. Per fornire una panoramica dei dati raccolti, si presentano di seguito alcune statistiche.

Per ogni stazione si è calcolata: la variabile *giorni totali*, che indica il numero di giorni intercorsi tra la prima e ultima osservazione registrata per la data stazione, e la variabile *osservazioni* ossia il numero di osservazioni relative alla stazione presenti nel database.

Nella tabella 1 sono riportate alcune statistiche per le due variabili.

	Mean	Median	Minimum	Maximum
osservazioni	11716	8947.0	2.0000	56353
giorni totali	13746	11262	56.000	59808

Tabella 1: Statistica descrittiva delle variabili *osservazioni* e *giorni totali*

Nella Figura 2 sono stati messe a confronto le due variabili. Sull'asse delle ascisse è riportato il rapporto tra le *osservazioni* ed i *giorni totali* mentre sull'asse delle ordinate la variabile *giorni totali*. A lato del grafico sono predenti gli istogrammi con le frequenze relative per le due variabili.

Il grafico vuole fornire un'informazione di massima sull'omogeneità delle osservazioni nel tempo e la presenza di "buchi". Più una stazione si sposta nella regione nord-ovest del grafico e più questa avrà un periodo ampio di giorni coperti ma con poche osservazioni. Dal grafico per migliorarne la leggibilità

sono state eliminate 7 stazioni con valori superiori ai 40 000 che comprimevano significativamente le osservazioni.

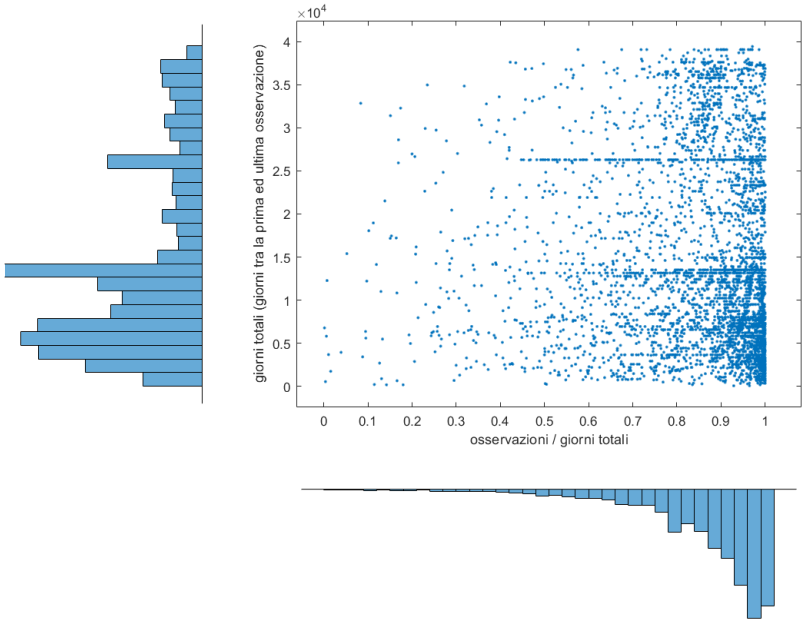


Figura 2: Scatterplot di *giorni totali* su *osservazioni / giorni totali* con rispettive frequenze relative

IV APPLICAZIONE DEL MODELLO

IV.1 CALCOLO DEI PARAMETRI

Costruito il database si è proceduto al calcolo dei parametri γ_1, γ_2, k per ogni singola stazione. Si è scelto di procedere prima con la stima di quasi massima verosimiglianza per tutte le stazioni e salvare i risultati ottenuti in un unico file. In questo modo le probabilità di precipitazione per ogni stazione potevano essere ottenute immediatamente all'occorrenza senza passare per la ricerca del massimo che impegnava del tempo. Infatti, una volta recuperati i parametri è sufficiente ricostruire le matrici X e Z, generando le armoniche della serie di Fourier, per avere tutti di dati necessari. I parametri sono stati calcolati per le sole stazioni che hanno almeno due anni di osservazioni e l'ultima sia successiva al 01/01/2022, per evitare che stazioni con troppi pochi dati o troppo distanti dall'anno corrente potessero produrre dati inaffidabili.

Il calcolo dei parametri, dopo diverse ore di elaborazione, ha portato alla creazione di un file di 1 114 righe, una per stazione, e 34 colonne: la prima colonna che identifica la stazione, le successive 26 i parametri di $\underline{\gamma}_1$, poi i 6 valori di $\underline{\gamma}_2$ e l'ultima colonna k.

Di interesse sono le colonne: 3, il secondo valore di $\underline{\gamma}_1$; 29, il secondo valore del vettore θ_2, e l'ultima che rappresenta k

L'interpretazione dell'ultima colonna, k, è immediata e si può osservare la distribuzione di frequenze nella Figura 3.

I valori di k sono tutti inferiori a 1 implicando che in tutte le stazioni in esame la distribuzione gamma presenta un asintoto sull'asse delle ordinate.

Il secondo parametro da osservare è la seconda componente del vettore $\underline{\gamma}_2$. Questo è interpretabile come l'effetto marginale di Δt sul parametro θ della distribuzione gamma. Nello specifico, un valore positivo di questo parametro indica una tendenza all'aumento di θ, mentre un valore negativo indica una decrescita.

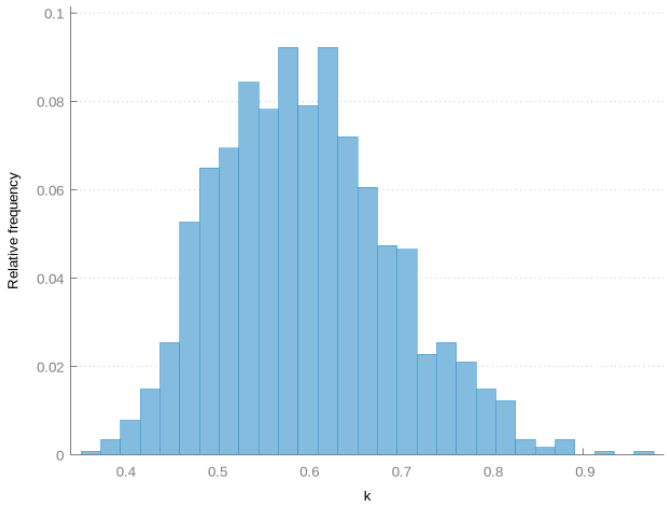


Figura 3: Distribuzione di frequenze del parametro k

Nella Figura 4 sono riportati i valori della colonna 29 visualizzati con un cerchio con centro la stazione in cui sono stati calcolati.

Il colore indica se il valore è positivo, azzurro, o negativo, arancione. Il colore del cerchio più essere interpretato come la tendenza futura della quantità di precipitazioni. Le zone in azzurro vedono una tendenza all'aumento delle code della distribuzione o in altri termini ad un aumento degli accumuli. Le zone arancioni vedono invece una tendenziale diminuzione nel tempo di questi ultimi.

Il raggio del cerchio indica invece la grandezza, in valore assoluto, del parametro. Più grande è il modulo maggiore sarà il raggio. Questo dà indicazione della "velocità" con cui cambia il valore di θ.

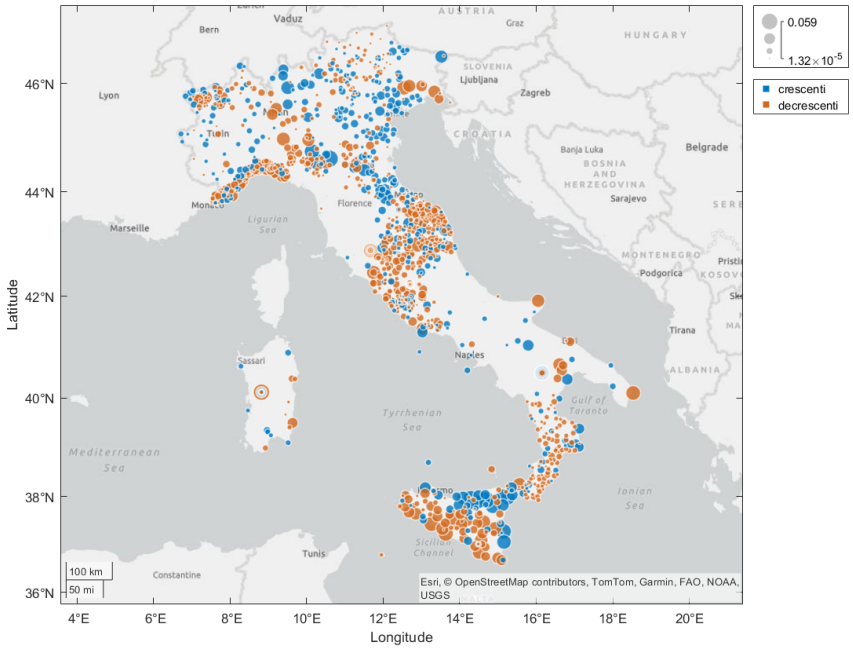


Figura 4: L'immagine mostra la distribuzione delle stazioni analizzate sul territorio. Assumendo che il giorno sia piovoso, il colore arancione indica che che nel tempo il parametro θ tende a decrescere (probabilità di precipitazioni in diminuzione) mentre il colore azzurro indica che il parametro cresce nel tempo (aumento della probabilità di precipitazioni). La dimensione del cerchio indica invece la velocità con cui avviene la variazione. Maggiore è la dimensione, maggiore è la velocità.

L'ultimo parametro su cui si vuole porre l'attenzione è la seconda componente del vettore $\underline{\gamma}_1$ nella colonna 3.

Come per $\underline{\gamma}_2$, anche in $\underline{\gamma}_1$ il secondo valore indica l'effetto marginale del tempo, in questo caso sulla probabilità p della Formula 1. Analogamente a quanto fatto con la Figura 4, nella Figura 5 è stato riportato questo valore in base alla stazione di riferimento.

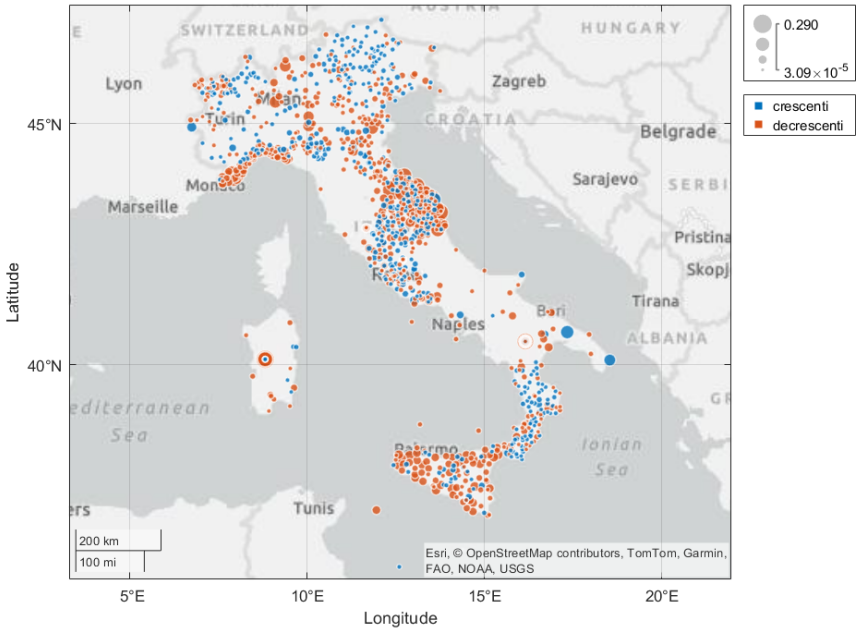


Figura 5: L'immagine mostra l'effetto marginale del tempo sulla probabilità di osservare giorni con precipitazioni per singola stazione. Il colore arancione indica che nel tempo il parametro p tende a decrescere (i giorni di pioggia tendono a diminuire) mentre il colore azzurro indica che il parametro cresce nel tempo (i giorni di pioggia tendono ad aumentare). La dimensione del cerchio indica invece la velocità con cui avviene la variazione. Maggiore è la dimensione, maggiore è la velocità.

IV.2 CONFRONTO CON UNA POLIZZA SUL MERCATO

Una volta stimati i parametri del vettore θ si è proceduto con il confronto con un contratto sul mercato per osservare la correttezza del modello. Si è preso a riferimento una polizza parametrica commercializzata da "UnipolSai Assicurazioni". La polizza scelta è *Salva Stagione*, con garanzia *Salva Estate*, che Prevede la copertura dei danni che le strutture ricettive in zone di villeggiatura possono subire, a causa del minor numero di clienti, in caso di maltempo. L'importo assicurato è forfettario e stabilito sulla valutazione del danno medio giornaliero subito in caso di maltempo. È possibile scegliere tra due periodi di

copertura:

- 3 mesi (dal 01/06 al 31/08) con 15, 20, 25, 30 o 35 giorni di franchigia;
- 4 mesi (dal 01/06 al 30/09) con 20, 25, 30, 35 o 40 giorni di franchigia.

L'indennizzo sarà corrisposto quando nel periodo di riferimento si registrano accumuli piovosi giornalieri superiori ai 5mm per un numero di giorni superiore a quello della franchigia scelta. Di seguito sono riportati i premi per il contratto *Salva Estate* con copertura a 3 mesi.

	Tasso pro mille	
Franchigia	Minimo	Massimo
15 giorni	0.80	4.50
20 giorni	0.40	5.75
25 giorni	0.45	3.50
30 giorni	1.45	4.00
35 giorni	0.65	5.00

Tabella 2: Premi della polizza *Salva Estate* con copertura 3 mesi.

Per il calcolo del premio teorico usando modello elaborato si è proceduto tramite metodo Monte Carlo, tramite il software *Gretl*.

Il metodo Monte Carlo si fonda sulla legge dei grandi numeri che afferma che la media dell'estrazione di variabili casuali indipendenti ed identicamente distribuite (i.i.d.) converge quasi certamente al valore atteso di queste.

Una volta identificata la quantità da approssimare γ, si costruisce una famiglia di variabili casuali X ed una funzione f tale che $E[f(X)] = \gamma$. Si genera un campione delle variabili X e si calcola la media empirica di queste. La legge dei grandi numeri assicura l'accuratezza della stima al crescere del numero dei campioni. [Graham and Talay, 2013]

Nel presente studio si è simulato il numero di giorni con piovosità superiore ai cinque millimetri nel periodo di copertura della polizza seguendo i seguenti passaggi:

1. Si seleziona una stazione;
2. si recupera il relativo vettore $\hat{\theta}$ ricavato in precedenza;

3. per ogni giorno t compreso tra le date 01/06/2024 e 31/08/2024, con $t = 1 \dots 92$, si calcola:
    3.1. la probabilità che il giorno sia piovoso (p_t della formula 1);
    3.2. la *shape* k della stazione;
    3.3. i parametri di *scale* θ_t.
4. si estrae un primo numero da $\mathcal{U}(0, 1)$;
5. se il valore estratto è minore di p_t si passa al giorno successivo, altrimenti si procede al punto seguente;
6. si estrae un secondo numero da $\mathcal{U}(0, 1)$ che chiamiamo q;
7. si calcola l'inversa della funzione gamma di parametri k e γ_t nel punto q. Se il valore ottenuto è maggiore di 5 si segna il giorno come da indennizzare, altrimenti si passa al giorno successivo;
8. eseguite le operazioni precedenti su tutti i giorni dell'intervallo si contano i giorni da indennizzare;
9. si ripete l'operazione per il numero di simulazioni desiderato. In questo studio sono state eseguite 100 000 simulazioni per stazione;
10. per ogni giorno di franchigia:
    10.1. si sottraggono i giorni di franchigia ai giorni da indennizzare per ogni singola simulazione;
    10.2. si sostituisce zero ai valori negativi del punto precedente;
    10.3. si calcolano le frequenze relative dei nuovi giorni da indennizzare;
    10.4. si calcola il valore atteso dei giorni da indennizzare per la data franchigia;
11. si ripete il processo per tutte le stazioni;

12. si calcola la media dei giorni da indennizzare per ogni franchigia utilizzando i valori attesi di tutte le stazioni.

13. si dividono i valori ottenuti per i giorni nell'intervallo (92).

Lo script della simulazione è in appendice VI.2.

I risultati ottenuti dalla simulazione sono riportati nella seguente tabella.

Giorni di franchigia	15	20	25	30	35
EV Giorni di indennizzo	2.1465	1.1649	0.5889	0.2661	0.1067
Premio calcolato (‰)	23.33	12.66	6.40	2.89	1.16

Tabella 3: Valori attesi per franchigia

Di seguito vengono invece mostrati i valori teorici con quelli riportati in polizza e messi in un grafico. La prima e la terza riga riportano i valori indicati nel foglio informativo della polizza.

Giorni di franchigia	15	20	25	30	35
Premio massimo	4.50	5.75	3.50	4.00	5.00
Premio stimato	23.33	12.66	6.40	2.89	1.16
Premio minimo	0.8	0.4	0.45	1.45	0.65

Tabella 4: Premio massimo, stimato e minimo espressi in permillesimi

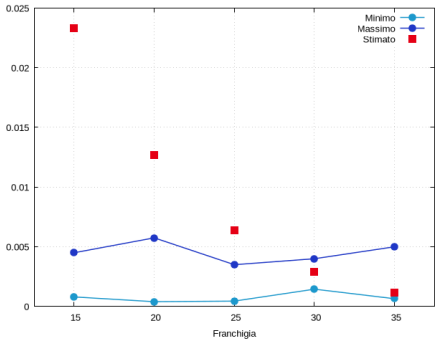


Figura 6: Grafico della tabella 4

Come è evidente dai dati, il prezzo stimato è coerente con i valori riportati in polizza per le sole franchigie a 30 e 35 giorni. Per livelli più bassi i parametri differiscono notevolmente.

V CONCLUSIONI

Il progetto ha portato all'elaborazione di un modello per il calcolo delle probabilità delle precipitazioni, la costruzione di un dataset di osservazioni ed, attraverso questi, il calcolo del prezzo di un prodotto assicurativo su eventi atmosferici.

Il modello è di mistura, separa le osservazioni con valore zero dalle altre, per applicare su queste una stima di massima verosimiglianza assumendo che le precipitazioni si distribuiscano secondo una gamma. Le precipitazioni sono approssimate tramite espansione in serie di Fourier e il parametro beta della distribuzione è dipendente dai coefficienti delle armoniche della serie.

Il dataset è stato costruito raccogliendo ed elaborando i dati dal sito di ISPRA. Si è effettuata una pulizia ed una selezione dei dati su cui far lavorare il modello.

Infine, si è prezzata una polizza parametrica sui dati raccolti.

Il progetto si poneva l'obbiettivo di studiare l'applicazione delle polizze parametriche alla copertura del danno catatrofale. La creazione di un prodotto assicurativo di tipo parametrico applicato ad eventi complessi come un'alluvione è un obbiettivo ambizioso che esula dalla portata di questo studio che però costituisce un primo passo verso la realizzazione analizzando e modellando una parte del fenomeno.

Si possono però trovare applicazioni dirette ed immediate dello studio. Anzitutto, la raccolta, la pulizia e l'organizzazione dei dati ha portato alla creazione di un dataset che può trovare altre applicazioni nella ricerca. In secondo luogo lo studio con il modello presentato consente alle compagnie assicuratrici un confronto ed una eventuale revisione dei metodi utilizzati per il calcolo dei premi delle loro polizze.

Un primo passo da compiere per il miglioramento del lavoro è sicuramente quello dell'ottimizzazione dei dati da utilizzare con il modello. Restando su quelli raccolti, un miglioramento potrebbe prevedere il calcolo dei parametri non

sulla singola stazione ma su campioni di queste. Questo, nel tentativo di ridurre le differenze tra i parametri, talvolta anche significative, tra stazioni limitrofe e che si possono rilevare anche visivamente dalla Figura 4. Altra miglioria è quella di estendere il campione di osservazioni con non solo osservazioni dirette delle stazioni meteo ma anche, ad esempio, elaborazioni derivanti dalle immagini satellitari.

Riguardo al modello un'ulteriore analisi dovrebbe essere rivolta alla proprietà dello stimatore ottenuto che non si sono potute approfondire. Ulteriore analisi può essere svolta riguardo la dipendenza di θ dai coefficienti di Fourier, nel variare il numero e la scelta.

Ritengo, e spero, che l'applicazione delle polizze parametriche alla copertura del rischio catastrofale possa diventare di interesse nel futuro prossimo. In primo luogo, per motivi commerciali: per il cliente, semplificano la comprensione del contratto e le condizioni di indennizzo riconducendole a elementi oggettivi. Per la compagnia, questo ha il vantaggio di rendere più semplice il calcolo del rischio e, teoricamente, ridurre il premio. Questo potrebbe incentivare un aumento della copertura assicurativa nel paese.

L'innovazione si estende anche all'aspetto riassicurativo, poiché questi contratti sono perfetti per essere cartolarizzati. La trasformazione in prodotti derivati consentirebbe una riduzione del rischio a carico delle compagnie assicuratrici, trasferendolo ai mercati. Questo fenomeno permetterebbe una distribuzione del rischio tra gli agenti del mercato, con conseguente maggiore consapevolezza delle conseguenze del cambiamento climatico, creando un legame diretto tra investimenti e clima. Nella speranza che questo ponga seriamente l'attenzione sul cambiamento climatico.

Riferimenti bibliografici

[Anatolyev and Gospodinov, 2011] Anatolyev, S. and Gospodinov, N. (2011). *Methods for estimation and inference in modern econometrics*. CRC Press.

[Considine, 2000] Considine, G. (2000). Introduction to weather derivatives. *Weather derivatives group, Aquila energy*, pages 1–10.

[Graham and Talay, 2013] Graham, C. and Talay, D. (2013). *Stochastic simulation and Monte Carlo methods: mathematical foundations of stochastic simulation*, volume 68. Springer Science & Business Media.

[Husak et al., 2007] Husak, G. J., Michaelsen, J., and Funk, C. (2007). Use of the gamma distribution to represent monthly rainfall in africa for drought monitoring applications. *International Journal of Climatology: A Journal of the Royal Meteorological Society*, 27(7):935–944.

[Monti et al., 2012] Monti, A. et al. (2012). *Il danno catastrofale. Strumenti giuridici e modelli istituzionali per la gestione dei rischi estremi*. Iuss Press.

[Petrosino, 2023] Petrosino, F. (2023). Le polizze parametriche tra disciplina del contratto e organizzazione dell'impresa. *Contratto e impresa*, 39(3):933–957.

[Santagata, 2022] Santagata, R. (2022). Polizze assicurative parametriche (o index-based) e principio indennitario. *Orizzonti del Diritto Commerciale*.

[Wilks, 1990] Wilks, D. S. (1990). Maximum likelihood estimation for the gamma distribution using data containing zeros. *Journal of climate*, pages 1495–1501.

VI APPENDICI

VI.1 modello.inp

```
include basis_functions.gfn

# calcolo parametri inziali
function matrix parm_init(series x)
    purged = x == 0 ? NA : x
    m = mean(purged)
    v = var(purged)

    thetahat = m/v
    khat = m * thetahat

    return {-log(thetahat); khat}
end function

# funzione gamma
function series gldens(series x, scalar k, series theta)
    series ret = k * log(theta) - lngamma(k) + (k-1) * log(x) - x*
    theta
    return x>0 ? ret : 0
end function

# funzione di quasi massima verosimiglianza
function series llik(matrix parm, series x, list X, list Z)
    k1 = nelem(X)
    k2 = nelem(Z)

    coeff_arm = parm[1:k1]
    g = parm[k1 + 1: k1 + k2]
    k = parm[k1 + k2 + 1]
    series ndx1 = lincomb(X, coeff_arm)
    series theta = exp(lincomb(Z, -g))

    prob = logistic(ndx1)

    ret = x == 0 ? log(1-prob) : log(prob) + gldens(x, k, theta)
    return ret
end function

# creazione del dataset con le osservazioni dal 01/01/1872 al
    31/12/2023
nulldata 55517 --preserve
setobs 7 1872-01-01 --time-series

open dsn=precipitazioniISO --odbc

# si importano i dati dal database
string q = sprintf("select substr(data,1,4), substr(data,5,2),
    substr(data,7,2), precipitazioni from precipitazioni where
    id_stazione=%d", scriptopt)
data rain obs-format="%d-%d-%d" query=q --odbc

# si crea un tempo unico per tutti gli script
# in modo da rendere confrontabili i coeff di fourier
dayStart = epochday(1872,1,1)
```

```
tt = (epochday($obsmajor, $obsminor, $obsmicro) - dayStart) /
    365.25

# si crea la lista della base di Fourier da 1 a 12
FOU = fourier(12, 365.25)

# eliminazione le osservazioni mancanti
smpl rain --no-missing

# creazione le matrici X e Z
list X = const tt FOU
list Z = const tt FOU[1:4]
kx = nelem(X)
kz = nelem(Z)

# creazione la serie binaria wet con i giorni di precipitazione
wet = rain > 0

# si esegue la regressione logistica e assegno a gamma1 i coeff
logit wet X --p-values
gamma1 = $coeff

# creazione starting point
gamma_pars = parm_init(rain)
gamma2 = gamma_pars[1] | zeros(kz-1,1)
k = gamma_pars[2]

# creazione un vettore da passare alla funzione llik
llik_pars = gamma1 | gamma2 | k

# viene eseguita la funzione di quasi massima verosimiglianza
mle ll = llik(llik_pars, rain, X, Z)
    params llik_pars
end mle --quiet --robust

gamma1 = llik_pars[1:kx]
gamma2 = llik_pars[kx+1:kx+kz]
k = llik_pars[kx+kz+1]

# calcolo delle p
prob_rain = logistic(lincomb(X, gamma1))

# calcolo dei theta
theta_t =  exp(lincomb(Z, gamma2))

# salvataggio dei dati
path = "path/to/file"
outfile @path --quiet
    printf("%d;", scriptopt)
    loop i = 1 .. nelem(gamma) - 1
        printf("%g;", gamma[i])
    endloop
    printf("%g", gamma[nelem(gamma)])
end outfile
```

VI.2 simulazione.inp

```
include basis_functions.gfn

# creazione del dataset con le osservazioni dal 01/01/1872 al
    31/08/2024
nulldata 59535 --preserve
setobs 7 1872-01-01 --time-series

path = path/to/file
matrix results = mread(path)

# si ricostruiscono le matrici
FOU = fourier(12, 365.25)
dayStart = epochday(1872,1,1)
tt = (epochday($obsmajor, $obsminor, $obsmicro) - dayStart) /
    365.25

list X = const tt FOU
list Z = const tt FOU[1:4]
kx = nelem(X)
kz = nelem(Z)

# si estrae la riga dei parametri corrispondente alla stazione
    scelta
pars = results[scriptopt,2:]

# si assegnano i parametri
gamma1 = pars[1:kx]
gamma2 = pars[kx+1:kx+kz]
k = pars[kx+kz+1]
p_t = logistic(lincomb(X, gamma1))
theta_t =  exp(lincomb(Z, gamma2))

# si restringe il dataset al periodo di interesse
smpl 2024-06-01 2024-08-31

# si effettua la simulazione
n_sim = 100000
n_giorni = $t2 - $t1 + 1
se_piove = muniform(n_giorni, n_sim) .< ({p_t} * ones(1,n_sim))
quanto_piove = zeros(n_giorni,n_sim)
loop i = 1 .. n_giorni
    quanto_piove[i,] = invcdf(g, k, theta_t[i], muniform(1,n_sim))
endloop
indennizzi = sumc(se_piove .* (quanto_piove .> 5))
indennizzi = indennizzi'

# si elaborano i risultati
freq 1 --matrix=indennizzi --quiet
results = $result
results[,2] = results[,2] ./ n_sim

franchigia = {15, 20, 25, 30, 35}
premi = zeros(1,5)

# calcolo del premio per ogni franchigia
loop f = 1 .. 5
    tmp = results
    tmp[,1] = results[,1] .- franchigia[f]
    tmp[,1] = tmp[,1] .> 0 ? tmp[,1] : 0
  premi[f] = sum(prodr(tmp))
```

```
endloop

# scrittura dei risultati su file
path = "path/to/file"
outfile @path --quiet
    loop j = 1 .. 4
        printf("%g;", premi[j])
    endloop
    printf("%g", premi[5])
end outfile
```

VI.3 gestoreStazioni.py

```
import datetime
import os
import re
import requests
import sqlite3
import time

class Stazione:

    def __init__(self, nome, rete = None, lat = -1, lon = -1,
                 quota = -1, quadrante = {'lat': 0, 'lon': 0}):
        self.nome = nome
        self.rete = rete
        self.lat = lat
        self.lon = lon
        self.quota = quota
        self.quadrante = {'lat': float(quadrante['lat']),
                          'lon': float(quadrante['lon'])}
        self.dati = {'secolo': [], 'tutti': []}
        self.idStazione = None
        self.idRete = None

    def __eq__(self, o):
        if type(o) != Stazione: return False
        if (self.nome.lower() == o.nome.lower() and
            self.rete.lower() == o.rete.lower()): return True
        return False

    def __str__(self):
        out = 'nome: %s\nrete: %s\nlat: %s\nlon: %s\nquota: %s' % (
    self.nome, self.rete, self.lat, self.lon, self.quota)
        return out

    def inQuadrante(self, stazione) -> bool:
        if isinstance(stazione, Stazione):
            lonOk = ((float(stazione.lon) <= (self.quadrante['lon']
     + 0.2) and
                     (self.quadrante['lon'] - 0.5) <= float(
    stazione.lon)))

            latOk = ((float(stazione.lat) >= (self.quadrante['lat']
     -0.2) and
                     (self.quadrante['lat'] + 0.5) >= float(
    stazione.lat)))
            return (lonOk and latOk)
        if isinstance(stazione, dict):
            lonOk = (float(stazione['lon']) <= (self.quadrante['lon
    '] + 0.2) and
```

```
43                      (self.quadrante['lon'] - 0.5) <= float(
    stazione['lon']))
44
45              latOk = (float(stazione['lat']) >= (self.quadrante['lat
    '] -0.2) and
46                      (self.quadrante['lat'] + 0.5) >= float(
    stazione['lat']))
47              return lonOk and latOk
48          raise Exception("L'argomento deve essere di tipo Stazione o
     dict")
49
50      def cercaOpenStreetMap(self) -> bool:
51          'Cerca la stazione in openstreetmap.\n'\
52          'Se la trova ritorna True e aggiunge le coordinate.\n'\
53          'Se non ci sono corrispondenze ritorna False.'
54
55          def getUrls(par: str):
56
57              def buildUrl(s):
58                  if type(s) == str: return 'https://nominatim.
    openstreetmap.org/search?q=' + s + '&format=json&polygon=1&
    addressdetails=1'
59                  return 'https://nominatim.openstreetmap.org/search?
    q=' + '+'.join(s) + '&format=json&polygon=1&addressdetails=1'
60
61              # Rimuovo caratteri non alfabetici
62              par = re.sub('[^a-zA-Z]+', '', par)
63              par = par.split(' ')
64              urls = [buildUrl(par)]
65              if type(par) == list: urls += [buildUrl(_) for _ in par
    ]
66              return urls
67
68          def trovaCorrispondenza(rList: list) -> dict|None:
69              'Prende in ingresso la lista dei risultati di
    openstreetmap.\n'\
70              'Se ne trova uno compatibile ritorna le coordinate'
71
72              for stazione in rList:
73                  if (stazione['address']['country'] == "Italia"
74                      and self.inQuadrante(stazione)):
75                      return ({'lat': float(stazione['lat']), 'lon':
    float(stazione['lon'])})
76              return None
77
78          for url in getUrls(self.nome):
79              # Creo header
80              h = {"Accept": "text/html,application/xhtml+xml,
    application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8",
81                  "Accept-Encoding": "gzip, deflate, br",
82                  "Accept-Language": "it-IT,it;q=0.8,en-US;q=0.5,en;q
    =0.3",
83                  "Connection": "keep-alive",
84                  "DNT": "1",
85                  "Host": "nominatim.openstreetmap.org",
86                  "Sec-Fetch-Dest": "document",
87                  "Sec-Fetch-Mode": "navigate",
88                  "Sec-Fetch-Site": "none",
89                  "Sec-Fetch-User": "?1",
90                  "Sec-GPC": "1",
91                  "Upgrade-Insecure-Requests": "1",
92                  "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv
```

```
:125.0) Gecko/20100101 Firefox/125.0"}
93              # Richiesta
94              r = requests.get(url, headers=h)
95
96              if r.status_code < 300:
97                  coord = trovaCorrispondenza(r.json())
98                  if coord:
99                      self.lat = coord['lat']
100                     self.lon = coord['lon']
101                     self.quota = -1
102                     return True
103             else:
104                 print("Errore nella richiesta per la stazione \
105                       {}\n{}\nErrore: {}\n\n".format(
106                       self.nome, url, r.status_code))
107         return False
108
109     def levDist(self, stazione):
110         a = self.nome
111         b = stazione.nome
112
113         d = [[0 for _ in range(len(b) + 1)] for _ in range(len(a) +
     1)]
114         d[0] = list(range(len(b) + 1))
115         for r in range(len(a) + 1): d[r][0] = r
116
117         for j in range(1, len(b) + 1):
118             for i in range(1, len(a) + 1):
119                 if a[i-1] == b[j-1]:
120                     subCost = 0
121                 else:
122                     subCost = 1
123
124                 d[i][j] = min(d[i-1][j] + 1, d[i][j-1] + 1, d[i-1][
    j-1] + subCost)
125
126         return d[-1][-1]
127
128     def cercaCorrispondenza(self, stazioni: list):
129         punteggi = [(self.levDist(stazione), stazione) for stazione
     in stazioni][:10]
130         for _ in punteggi:
131             if self.inQuadrante(_[1]):
132                 self.lat = float(_[1].lat)
133                 self.lon = float(_[1].lon)
134                 self.quota = float(_[1].quota)
135                 return True
136         return False
137
138     def aggiungiCoordinate(self, listaStazioni) -> bool:
139         'Cerca le coordinate: True se vengono trovate, False
    altrimenti'
140         # Cerca le coordinate della stazione
141         if self in listaStazioni:
142             for stazione in listaStazioni:
143                 if self == stazione:
144                     self.lat = stazione.lat
145                     self.lon = stazione.lon
146                     self.quota = stazione.quota
147                     return True
148                 break
149         if self.cercaCorrispondenza(listaStazioni):
```

```
150             return True
151         if self.cercaOpenStreetMap():
152             return True
153         return False
154 
155 
156 class DbManager:
157     def __init__(self, nome: str):
158         self.nome = nome
159         self.con = sqlite3.connect(nome, check_same_thread=False)
160         self.cur = self.con.cursor()
161 
162     def clearTable(self, table):
163         self.cur.execute("delete from " + table)
164         self.con.commit()
165 
166     def clearDatabase(self):
167         self.con.close()
168         self.con = sqlite3.connect(self.nome)
169         self.con.execute("delete from Reti")
170         self.con.execute("delete from Stazioni")
171         self.con.execute("delete from PrecipitazioniSecolo")
172         self.con.execute("delete from Precipitazioni")
173         self.con.commit()
174         self.cur = self.con.cursor()
175 
176     def insertPrecipitazioniTutte(self, stazione: Stazione):
177         data = [(stazione.idRete, stazione.idStazione) + _ for _ in
       stazione.dati['tutti']]
178         self.cur.executemany("insert into Precipitazioni values
      (?,?,?,?)", data)
179         self.con.commit()
180 
181     def insertPrecipitazioniSecolo(self, stazione):
182         data = [(stazione.idRete, stazione.idStazione) + _ for _ in
       stazione.dati['secolo']]
183         self.cur.execute("insert into PrecipitazioniSecolo values
      (?,?,?,?)", data)
184         self.con.commit()
185 
186     def updateStazioneCoordQuota(self,stazione):
187         self.cur.execute("update Stazioni set latitudine=?,
      longitudine=?, quota=? where id_rete=? and id_stazione=?",
188                          (stazione.lat, stazione.lon, stazione.
      quota, stazione.idRete, stazione.idStazione))
189         self.con.commit()
190 
191 
192 class GestoreStazioni:
193     def __init__(self):
194         self.pathStazioniInLista = "listaStazioni"
195         self.cartellaCSV = './tuttiCSV/'
196         self.db = DbManager('precipitazioni.db')
197         self.allCSV = os.listdir(self.cartellaCSV)
198         self.idReti = {}
199         self.idStazioni = {}
200         # Creo un range di date tra il 1-1-1922 e il 31-12-2022
201         self.tutteLeDate = [
202             datetime.datetime.fromtimestamp(t).strftime("%d-%m-%Y")
203             for t in range(-1514768400, 1672527600, 60*60*24)]
204         # Carico le stazioni in lista
205         self.stazioniInLista = self.getStazioniInLista()
```

```
206
207     def caricaCSV(self, path: str):
208         with open(path, 'r', encoding="utf-8") as f:
209             content = f.read()
210             content = content.replace('"', '')
211             content = content.split("\n")[:-1]
212             content = [riga.split(",") for riga in content][:-1]
213         return content
214
215     def getStazioniInLista(self):
216         stazioniInLista = [
217             Stazione(
218                 rete = riga[0],
219                 nome = riga[1],
220                 lon = riga[3],
221                 lat = riga[4],
222                 quota = riga[5]
223                 )
224             for riga in self.caricaCSV(self.pathStazioniInLista)
    [1:]]
225         return stazioniInLista
226
227
228
229     def correggiData(self, d: str):
230         mesi = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', '
    AUG', 'SEP', 'OCT', 'NOV', 'DEC']
231         newD = d.split('-')
232         newD = '-'.join([newD[1], "%02d" % (mesi.index(newD[0]) +
    1), newD[2]])
233         return newD
234
235     def aggiungiDateMancanti(self, listaTuple: list):
236         dictDate = {_[0]:_[1] for _ in listaTuple}
237         return [(_, dictDate[_]) if _ in dictDate else (_, -1)
238                 for _ in self.tutteLeDate]
239
240     def getStack(self, content, quadrante) -> list:
241         'Crea la lista delle Stazioni dal file csv e il dict
    coordinate'
242         # Elimino i duplicati
243         singoleStazioni = {(_[0], _[1]) for _ in content}
244         stackStazioni = []
245         for stazione in singoleStazioni:
246             tmpStazione = Stazione(nome=stazione[1], rete=stazione
    [0], quadrante=quadrante)
247             tmpStazione.dati['tutti'] = [(self.correggiData(x[2]),x
    [3]) for x in content if (x[0] == tmpStazione.rete and x[1] ==
    tmpStazione.nome)]
248             tmpStazione.dati['secolo'] = self.aggiungiDateMancanti(
    tmpStazione.dati['tutti'])
249             stackStazioni += [tmpStazione]
250
251         return stackStazioni
252
253     def file2coord(self, s: str):
254         lat = int(s[0:2]) + (int(s[2:4]) / 60)
255         lon = int(s[5:7]) + (int(s[7:9]) / 60)
256         return {'lat': lat, 'lon': lon}
257
258     def stazioneInDb(self, stazione: Stazione) -> bool:
259         try:
```

```
260             result = self.db.cur.execute(
261                 "select stazioni.stazione, reti.rete from stazioni
    inner join reti on stazioni.id_rete=reti.id_rete where stazioni
    .stazione=? and reti.rete=?", (stazione.nome, stazione.rete)
262                 )
263             if result.fetchall(): return True
264             return False
265         except:
266             print("Errore nella ricerca della stazione {}-{}".
    format(
267                 stazione.rete, stazione.nome))
268             return False
269 
270     def getIdRete(self, stazione) -> int:
271         # Cerco se la rete    gi  presente
272         res = self.db.cur.execute("select Reti.id_rete from Reti
    where rete=?",
273                                   (stazione.rete,)).fetchall()
274         # Controllo che non ci siano pi  reti con lo stesso nome
275         if len(res) > 1:
276             raise Exception("Presenti pi  reti con lo stesso nome"
    )
277         # Se la rete    stata trovata return l'id, -1 altrimenti
278         if res: return res[0][0]
279         # La rete non    stata trovata, la aggiungo
280         try:
281             self.db.cur.execute("insert into Reti (rete) values(?)"
    , (stazione.rete,))
282             self.db.con.commit()
283         except:
284             print("Errore nell'aggiunta della rete {}-{}".format(
285                 stazione.nome, stazione.rete))
286             return -1
287         return self.getIdRete(stazione)
288 
289     def getIdStazione(self, stazione) -> int:
290         # Cerco se la stazione    gi  presente
291         res = self.db.cur.execute("select id_stazione from Stazioni
     where id_rete=? and stazione=?",
292                                   (stazione.idRete, stazione.nome)).
    fetchall()
293         # Controllo che non ci siano pi  reti con lo stesso nome
294         if len(res) > 1:
295             raise Exception("Presenti pi  stazioni con lo stesso
    nome e id_rete")
296         # Se la rete    stata trovata return l'id, -1 altrimenti
297         if res: return res[0][0]
298         # La rete non    stata trovata, la aggiungo
299         try:
300             self.db.cur.executemany("insert into Stazioni (id_rete,
     stazione, latitudine, longitudine, quota) values(?,?,?,?,?)",
    [(int(stazione.idRete), stazione.nome, float(stazione.lat),
    float(stazione.lon), float(stazione.quota)),])
301             self.db.con.commit()
302         except Exception as error:
303             print("Errore nell'aggiunta della stazione " + str(
    stazione.nome) + " - " + str(stazione.rete))
304             print("Errore: ", error)
305             return -1
306         return self.getIdStazione(stazione)
307 
308     def elaboraFile(self, path: str):
```

```
309         def elaboraStazione(self, stazione):
310             if self.stazioneInDb(stazione): return
311             trovata = stazione.aggiungiCoordinate(self.
    stazioniInLista)
312             stazione.idRete = self.getIdRete(stazione)
313             stazione.idStazione = self.getIdStazione(stazione)
314             self.db.insertPrecipitazioniTutte(stazione)
315             if trovata: self.db.insertPrecipitazioniSecolo(stazione
    )
316 
317         # Carico il contenuto del file in content
318         content = self.caricaCSV(self.cartellaCSV + path)[1:]
319 
320         # Creo il dict coordinate
321         coord = self.file2coord(path)
322 
323         # Creo la lista delle stazioni
324         stack = self.getStack(content, coord)
325 
326         for stazione in stack:
327             elaboraStazione(self, stazione)
328             print(stazione.nome, stazione.rete)
329 
330         return stack
331 
332     def start(self):
333         start = time.time()
334         counter, maxCounter = 1, len(self.allCSV)
335         for csv in self.allCSV:
336             print("Elaboro {}... {} di {} [{}%]".format(csv,
    counter, maxCounter, int((counter/maxCounter)*100)))
337             gestoreStazioni.elaboraFile(csv)
338             counter += 1
339         print("Tempo trascorso: " + str(time.time() - start))
340 
341     def correggiCoordinateQuota(self):
342         for stazione in self.stazioniInLista:
343             print(stazione)
344             stazione.idRete = self.getIdRete(stazione)
345             stazione.idStazione = self.getIdStazione(stazione)
346             self.db.updateStazioneCoordQuota(stazione)
347 
348 
349 
350 
351 if __name__ == '__main__':
352     gestoreStazioni = GestoreStazioni()
353     gestoreStazioni.start()
```