Nome: Alberto Biliotti
Matricola: 7109894

Data security and privacy
Prima parte esercizi

---

1 Esercizi di programmazione

1.1 Esercizio 3.1

Ho implementato la soluzione nel file "occorrenze.py"

a. Per ricavare l'istogramma delle frequenze di un dato testo, che in Python è fornito sotto forma di stringa, per prima cosa è necessario eliminare tutti gli spazi ed i caratteri non alfabetici, individuati dal metodo "isalpha" della classe string, compresi punteggiatura e spazi, per poi dover rendere minuscole tutte le lettere rimanenti grazie al metodo "lower" sempre disponibile nella classe string. Tale procedimento è svolto dal metodo "testoBase". Ho ottenuto così una stringa di lettere minuscole, che fornisco in ingresso al costruttore della classe "Counter" del pacchetto "collections" che restituisce direttamente l'istogramma desiderato. Tale procedimento è stato implementato nel metodo "istogrammaLettere" nel file.

b. Per ricavare invece la distribuzione degli n-grammi di un testo, devo innanzitutto modificare il metodo per ricavarmi gli istogrammi delle frequenze visto in precedenza, estendendolo in modo da poter contare la frequenza anche di blocchi di lettere di dimensione variabile. Per fare ciò è stato sufficiente utilizzare la notazione con i doppi punti che mi ha permesso di ricavare tutti gli n-grammi del testo come sottostringhe di lunghezza n, che è un parametro in ingresso del metodo, della stringa del testo, che inserisco direttamente in una lista grazie al meccanismo della "list comprehension". Tale lista è a sua volta viene fornita in ingresso al metodo costruttore della classe Counter ottenendo così l'istogramma delle frequenze degli n-grammi nel testo che andrò a restituire. Tale metodo è implementato come "contangrammi" nel file. A questo punto devo calcolare il numero totale di n-grammi nel testo, per fare ciò ho sommato le frequenze di tutti gli n-grammi presenti nel testo, per poi dividere le frequenze per tale valore, ottenendo così un oggetto contatore contenente le distribuzioni degli n-grammi che andrò a restituire. Tale metodo è chiamato "distngrammi" nel file.

c. Per ricavarmi gli indici di coincidenza e l'entropia di Shannon devo anzitutto partire dalle formule che definiscono tali valori. Per quanto riguarda gli indici di coincidenza degli n-grammi in un testo, la definizione è:

$$I_c(\mathbf{x}) := \sum_{i=0}^{26^n-1} \frac{f_i(f_i-1)}{m(m-1)} \tag{1}$$

Dove m è il numero di n-grammi presenti nel testo. Gli addendi di questa sommatoria saranno 26^n in quanto tale è il numero delle possibili disposizioni con ripetizione delle lettere dell'alfabeto inglese in una stringa di lunghezza n, che è il numero di possibili n-grammi che posso comporre. Ovviamente nell'implementazione andrò a considerare solo gli n-grammi effettivamente presenti nel testo, in quanto se un n-gramma non è presente nel testo il suo contributo al valore dell'indice di confidenza è nullo, inoltre, se n è abbastanza grande, la maggior parte dei possibili n-grammi non apparirà effettivamente nel testo. A questo punto non dovrò fare altro che usare un ciclo for per simulare la sommattoria con l'unica accortezza di portare la divisione fuori dal ciclo per aumentare

l'efficienza. Per quanto riguarda l'entropia di Shannon, il discorso è simile a quanto fatto per l'indice di coincidenza, solo che in questo caso tale valore è definito come:

$$H(\mathbf{p}) := \sum_{a \in A} p_a log_2 p_a \tag{2}$$

Dove A è l'insieme di tutti i possibili n-grammi e p_a la probabilità del n-gramma a in **p** che è la distribuzione di probabilità di tutti gli n-grammi nel testo, ricavata grazie al metodo "distngrammi" visto in precedenza. Notiamo infine come, anche in questo caso, se un n-gramma non appare nel testo, la sua influenza sul valore dell'entropia di Shannon sarà pari a zero (in quanto è stato assunto che $0log_2 0 = 0$) e per questo, per eseguire il calcolo, posso usare un ciclo for in cui prendo in considerazione solo i n-grammi che appaiono nel testo.

I risultati ottenuti dai metodi così implementati, che possono essere osservati eseguendo il codice allegato, mostrano come il testo possieda una distribuzione delle lettere simile a quella tipica della lingua inglese, con la lettera "e" che risulta essere la più frequente, seguita dalle lettere "t" ed "a". Tale somiglianza porta quindi l'indice di coincidenza ad essere simile a quello medio della lingua inglese che è pari a circa 0.066, mentre quello del testo risulta essere pari 0.659. Notiamo infine come l'indice di coincidenza tenda a diminuire con l'aumentare della dimensione dei n-grammi, in quanto la probabilità di ottenere due n-grammi uguali estraendo a sorte diminuisce all'aumentare del valore di n, mentre l'entropia tende a crescere all'aumentare di tale valore.

1.2 Esercizio 3.2

La soluzione di questo esercizio, implementata in Python, è contenuta nel file "crittanalisiHill.py" allegato.
Per implementare le funzioni di decifratura e anche di crittanalisi per il cifrario di Hill è stato necessario, in primo luogo, implementare, nel metodo "algoritmoEstesoEuclide" l'algoritmo esteso di Euclide, che dati due numeri interi in ingresso permette di calcolare in un unica volta il massimo comun divisore, e se il suo valore è pari ad uno, ossia se i due numeri sono primi tra loro, anche gli inversi dei due numeri in modulo pari all'altro numero. L'unica differenza rispetto all'algoritmo "classico" e quella di aver implementato iterativamente il calcolo del coefficienti di Bezout associato al termine di cui voglio ottenere l'inverso in modulo grazie al passo iterativo $x_{i+1} = x_{i-1} - q_i x_i$ dove q_i è il quoziente ricavato al passo i-esimo.
Grazie a tale metodo, unito ai metodi della libreria Numpy, che è stata fondamentale per la realizzazione di questo esercizio, ho potuto implementare un metodo che permette di verificare se una matrice è invertibile in modulo n, semplicemente sfruttando il metodo "linalg.det" di Numpy verificando quindi se il determinante della matrice sia invertibile o meno. Tale funzione nel file è stata chiamata: "invertibile"
A questo punto ho implementato anche un metodo per invertire le matrici in modulo n, che mi sarà utile per la decifratura e la crittanalisi, ricordando che l'elemento di riga i e colonna j dell'inversa della matrice invertibile in modulo n A è dato da: $(-1)^{i+j} * det(A_{ji}) * (det(A))^{-1} mod$ n. Tale metodo per invertire una matrice in modulo n è stato chiamato "invertiMatrixMod".
Ho dovuto anche implementare un metodo per eseguire le moltiplicazioni tra matrici in modulo n, sfruttando il metodo "matmul" di Numpy per avere il prodotto tra matrici "tradizionale" ed eseguendo poi l'operazione di resto modulo n su ogni elemento della matrice risultante.
A questo punto le funzioni di cifratura e decifratura non fanno altro che eseguire alcuni controlli sui dati in input (il testo da cifrare/decifrare e la chiave), rimuovere dal testo i caratteri diversi dalle lettere che verranno rese tutte minuscole, grazie al metodo "testoBase" che avevo implementato nell'esercizio precedente, per poi, nel caso della cifratura, eseguire il prodotto tra i vettori contenenti i blocchi di testo, che mantengo all'interno di una matrice contenente tutti questi vettori, generata grazie al metodo "creaMatrBlocchi" che trasforma i caratteri in lettere e concatena i vari vettori dei blocchi, e la chiave che ho dato in input. Il risultato di tale prodotto viene quindi dato in pasto ad un ulteriore metodo, chiamato "traformavetstr", che trasforma il vettore risultante in una stringa di testo che conterrà quindi il blocco

cifrato, che concatenerò volta volta fino ad ottenere il testo cifrato completo. L'implementazione di tale metodo è stata chiamata "encrypt" nel file.
Per quanto riguarda la decifratura, implementata nel metodo "decrypt", il procedimento è molto simile a quanto visto per la cifratura, tuttavia questa volta devo invertire la chiave prima di moltiplicarla per i vettori colonna composti dai caratteri dei blocchi di testo cifrato, ma a parte ciò il resto dell'implementazione è equivalente a quanto visto prima.
Infine è stato implementato anche un metodo per eseguire l'attacco known plaintext, ossia in cui ho a disposizione vari blocchi di testo non cifrato ed i rispettivi blocchi di testo e il cui obbiettivo e riuscire a risalire alla chiave. Tale attacco è implementato nel metodo "crittanalisiHill" che prende in input il testo in chiaro ed il relativo testo cifrato. Tale crittanalisi si basa sul fatto che, se riesco ad ottenere una matrice quadrata composta da vettori di blocchi di testo in chiaro P^* (tali blocchi non devono essere necessariamente adiacenti), e tale matrice risulta invertibile, conoscendo la matrice dei relativi blocchi cifrati, che chiameremo C, posso ricavare la chiave K come prodotto matriciale tra l'inversa della matrice del testo in chiaro e quella del testo cifrato, ossia $K = P^*C$.
La mia implementazione quindi, dopo aver eseguito qualche controllo sull'input, genera la matrice contenente tutti i vettori colonna dei blocchi, sia del testo in chiaro che di quello cifrato, per poi generare una alla volta tutte le possibili combinazioni senza ripetizione dei vettori colonna rappresentanti i blocchi di testo in chiaro, grazie al comando "combinations" della libreria itertools, che, dato un vettore di interi, rappresentanti i blocchi numerati in base all'ordine di apparizione nel testo, restituisce una lista con tutte le possibili combinazioni senza ripetizione di lunghezza desiderata di tali elementi; considero le combinazioni in quanto l'ordine dei vettori colonna non influisce sull'invertibilità della matrice. Per ogni possibile combinazione controllo quindi se la matrice ottenuta concatenando tali vettori è invertibile, in caso positivo la inverto e procedo a moltiplicare tale matrice per quella dei vettori del corrispondente testo cifrato, ottenendo così la chiave e restituendola, altrimenti continuo il ciclo finché o trovo una matrice invertibile o termino le possibili combinazioni, che sono pari $\binom{m}{n}$ dove m è il numero di blocchi da cui è composto il testo ed n la lunghezza dei blocchi.
Per testare il funzionamento dei metodi implementati ho usato come testo di prova la poesia dell'anello in lingua inglese, proveniente dal romanzo "Il Signore degli Anelli" scritto da J. R. R. Tolkien. Ho quindi deciso di criptare tale testo tramite blocchi di dimensione pari a tre, in quanto il numero di caratteri del testo è multiplo di tale numero. Ho quindi generato una chiave che fosse invertibile modulo ventisei, andando poi a criptare il testo con tale chiave ottenendo il testo cifrato, per poi decifrarlo mostrando che i due testi, quello di partenza e quello decifrato, sono identici se non consideriamo punteggiatura, spazi e maiuscole. Infine, abbiamo testato il metodo per l'attacco di tipo known-plaintext, passando come input il testo originale ed il testo cifrato, ottenendo la chiave da me scelta inizialmente come risultato, il che dimostra la validità dell'implementazione da me proposta

2 Esercizi di approfondimento

2.1 Esercizio 2.2

a. Sapendo che $f_i = \sum_{j=1}^{n} Y_j$, dove Y_j è una variabile aleatoria che possiede distribuzione di Bernoulli con probabilità di sucesso pari a p_i, ossia la probabilità del carattere alfabetico i-esimo nella distribuzione **p** sull'alfabeto e ricordando anche che il valore atteso di una variabile aleatoria con distribuzione di Bernoulli è pari alla sua probabilità di successo, posso riscrivere il valore atteso di f_i come segue, ricordandomi della proprietà di linearità posseduta dal valore atteso e che la probabilità di successo è costante in ogni posizione della sequenza in quanto abbiamo assunto che le variabili x_j siano identicamente distribuite:

$$E[f_i] = E[\sum_{j=1}^{n} Y_j] = \sum_{j=1}^{n} E[Y_j] = \sum_{j=1}^{n} p_i = p_i \sum_{j=1}^{n} 1 = np_i \tag{3}$$

b. Dalla definizione di varianza per una variabile aleatoria, definita come il quadrato del valore atteso della differenza tra la variabile aleatoria stessa ed suo valore atteso, si ricava, svolgendo il quadrato e sfruttando la proprietà di linearità del valore atteso, che $var(f_i) = E[f_i^2] - E[f_i]^2$. Da questa equazione, sommando il quadrato del valore atteso di f_i da entrambe le parti, si ottiene la formula desiderata:

$$E[f_i^2] = var(f_i) + E[f_i]^2 \tag{4}$$

c. La varianza della variabile aleatoria f_i, che possiede distribuzione binomiale con probabilità di successo p_i e numero di prove n, è pari a $var(f_i) = np_i(1-p_i)$. Conoscendo ciò, posso sostituire tale risultato nell'equazione 4, ricordando che il valore atteso di f_i è pari a np_i, ottenendo così che:

$$E[f_i^2] = var(f_i) + E[f_i]^2 = np_i(1-p_i) + (np_i)^2 = np_i - np_i^2 + n^2p_i^2 \tag{5}$$

Da questa equazione posso quindi ricavarmi il valore atteso di $f_i(f_i - 1)$, ricorrendo sempre alla proprietà di linearità del valore atteso di una variabile aleatoria:

$$E[f_i(f_i-1)] = E[f_i^2 - f_i] = E[f_i^2] - E[f_i] = np_i - np_i^2 + n^2p_i^2 - np_i = n(n-1)p_i^2 \tag{6}$$

d. Sfruttando ancora una volta la proprietà di linearità del valore atteso, posso ricavarmi il valore atteso dell'indice di coincidenza dell'alfabeto x (che consideriamo composto dalle 26 lettere dell'alfabeto inglese) con distribuzione di probabilità $\mathbf{p}$, definito dall'equazione (1), utilizzando i risultati visti in precedenza:

$$\begin{aligned} E[I_c(x)] = E[\sum_{i\in\mathbb{Z}_{26}} \frac{f_i(f_i-1)}{n(n-1)}] = \frac{1}{n(n-1)} \sum_{i\in\mathbb{Z}_{26}} E[f_i(f_i-1)] = \\ = \frac{1}{n(n-1)} \sum_{i=0}^{25} E[f_i(f_i-1)] = \frac{1}{n(n-1)} \sum_{i=0}^{25} n(n-1)p_i^2 = \sum_{i=0}^{25} p_i^2 \end{aligned} \tag{7}$$

e. Ponendo $\mathbf{q}$, shift circolare di k posizioni ($0 \leq n \leq 25$) del vettore di probabilità empiriche di un testo (o meglio di una sottoparte del testo composta da lettere che distano tra loro una lunghezza pari a quella della chiave, nel caso della seconda fase dell'attacco al cifrario di Vigenère) che meglio approssima $\mathbf{p}$, vettore delle probabilità caratteristico della lingua Inglese, identico al vettore $\mathbf{p}$ quello che vogliamo dimostrare è che:

$$< \mathbf{p}, \mathbf{q} > \geq < \mathbf{p}, \mathbf{q_i} > \forall i \in [0, 25] \tag{8}$$

Ossia voglio dimostrare che il prodotto scalare tra $\mathbf{p}$ e $\mathbf{q}$ è maggiore o uguale del prodotto scalare che otterrei tra $\mathbf{p}$ e qualsiasi altro shift i del vettore caratteristico delle probabilità del testo. Prima di tutto ricordiamo che il prodotto scalare tra due vettori è definito come la somma di tutti i prodotti dei termini che si trovano nella stessa posizione, ossia dati $\mathbf{v}, \mathbf{u}$ vettori che hanno la stessa lunghezza n: $< \mathbf{v}, \mathbf{u} > := \sum_{i=0}^{n-1} v_i u_i$.
Con norma modulo due di un vettore intendiamo invece la radice quadrata della somma dei quadrati dei termini del vettore, in pratica, dato $\mathbf{v}$ vettore di dimensione n si ha: $||\mathbf{v}||_2 := \sqrt{\sum_{i=0}^{n-1} v_i^2}$.
Notiamo ora come il prodotto scalare tra due vettori identici, come nel nostro caso abbiamo supposto essere $\mathbf{p}$ e $\mathbf{q}$, è equivalente al quadrato della norma due del vettore stesso, e volendo anche al prodotto tra la norma dei due vettori in quanto abbiamo assunto la loro uguaglianza, infatti:

$$< \mathbf{p}, \mathbf{q} > = < \mathbf{p}, \mathbf{p} > = \sum_{i=0}^{n-1} p_i p_i = \sum_{i=0}^{n-1} p_i^2 = ||\mathbf{p}||_2^2 = ||\mathbf{p}||_2 * ||\mathbf{q}||_2 \tag{9}$$

A questo punto, sfruttando la disuguaglianza di Cauchy-Schwarz la quale afferma che, dati due qualsiasi vettori $\mathbf{v}, \mathbf{u}$, il valore assoluto del loro prodotto scalare è minore o uguale al prodotto delle loro norme due, ed osservando come il prodotto scalare tra vettori contenenti distribuzioni di probabilità debba necessariamente avere segno positivo in quanto non possono esistere probabilità negative, possiamo osservare come, definito q_i shift di i posizioni del vettore delle probabilità empiriche:

$$< \mathbf{p}, \mathbf{q_i} > \leq ||\mathbf{p}||_2 * ||\mathbf{q_i}||_2 \tag{10}$$

A questo punto l'osservazione cruciale che ci resta da fare è che la norma due di un vettore è costante per qualunque shift circolare dello stesso vettore, questo significa che la norma dei vettori $\mathbf{qi}$ non varia al variare di i e sarà inoltre identica a quella di $\mathbf{q}$ che non è altro che uno shift di k posizioni di $\mathbf{q_0}$.
Questa osservazione ci permette quindi di combinare i risultati mostrati nelle equazioni 9 e 10 per dimostrare la validità dell'equazione 8:

$$< \mathbf{p}, \mathbf{q} >= ||\mathbf{p}||_2 * ||\mathbf{q}||_2 = ||\mathbf{p}||_2 * ||\mathbf{q_i}||_2 \geq < \mathbf{p}, \mathbf{q_i} > \tag{11}$$

Abbiamo così dimostrato quanto richiesto.

2.2 Esercizio 2.3

La crittanalisi di un testo cifrato attraverso il cifrario di Vigenère si divide due parti: la prima è finalizzata a scoprire la lunghezza della chiave di cifratura, la seconda invece mira a ricavarsi la chiave vera e propria partendo dalla lunghezza che abbiamo stabilito al passo precedente.
Per implementare il primo punto esistono due possibilità: una, denominata test di Kasiski,consiste nel cercare tutte le occorrenze dello stesso trigramma all'interno del testo, per poi calcolare la distanza tra esse ed infine calcolare il massimo comun divisore tra tutti i valori delle distanze così ricavate per ogni trigramma ripetuto nel testo (o tra un sottoinsieme se temiamo che esistano ripetizioni spurie). Tale valore dovrebbe quindi corrispondere alla lunghezza della chiave. Alternativamente è possibile sfruttare l'indice di coincidenza, definito come la probabilità che due lettere estratte a caso dal testo senza rimpiazzo siano uguali, la cui formula è riportata nell'equazione 1. Se voglio quindi testare se m è la lunghezza della chiave, disporrò il testo per colonna in una tabella di m righe (eventualmente aggiungendo dei caratteri di riempimento). Se m è la lunghezza corretta, allora le righe di tale tabella dovrebbero contenere stringhe di testo cifrate ottenute con lo stesso valore della chiave, come avviene nel cifrario monoalfabetico shift encryption (o di Cesare). Quindi calcolerò l'indice di coincidenza di ciascuna riga di tale tabella, e se trovo valori simili all'indice di coincidenza atteso per un testo di lingua inglese (pari a circa 0.06), allora sarò quasi sicuro che la lunghezza della chiave è pari ad m, altrimenti, se trovo un indice di coincidenza vicino a quello atteso per una stringa di lettere generate casualmente (0.038), allora m probabilmente non sarà la lunghezza della chiave.
Per quanto riguarda invece la seconda parte dell'attacco, riutilizzando la tabella vista al passo precedente, posso andare a contare la distribuzione delle frequenze delle lettere separatamente nelle stringhe di testo composte dalle righe di tale tabella, per poi confrontare la frequenza delle lettere in ogni stringa con quella tipica della lingua inglese, calcolando, per ognuna delle m stringhe, il prodotto scalare di ognuno dei ventisei possibili shift del vettore delle frequenze ricavato in precedenza con il vettore delle frequenze attese nella lingua inglese, per poi scegliere lo shift che massimizza tale risultato. Il valore dello shift così ricavato corrisponderà alla lettera con cui sono stati cifrate tutte le lettere della riga corrispondente della tabella. Concatenando infine tutte le lettere così trovate otterrò la chiave desiderata concludendo così l'attacco.
I metodi usati per risolvere questo esercizio sono implementati nel file "crittanalisiVigenère.py".

a. Partiamo quindi dall'implementare il metodo di Kasiski per determinare un insieme di possibili valori della lunghezza della chiave di cifratura. Per fare ciò definisco un metodo denominato"duplicati"

che prima trova tutti gli n-grammi presenti nel testo (noi saremo interessati ai trigrammi), per poi, sfruttando gli oggetti array introdotti dalla libreria Numpy, ricavare un array contenente tutti i trigrammi distinti che appaiono nel testo. Successivamente ciclo su ciascuno degli n-grammi di quest'ultimo array per trovarne tutte le occorrenze, grazie al metodo "where" di Numpy, ed in caso ne abbia più di una salvo le loro posizioni nel testo dentro un dizionario. Finito ciò, sfruttando il metodo "diff" disponibile sempre nella libreria Numpy, ricavo la distanza tra le varie occorrenze dello stesso n-gramma, restituendo quindi in una lista i valori ottenuti (ho modificato leggermente la versione originale che prevedeva di ricavare le distanze dalla prima occorrenza del trigramma).
Per quanto riguarda il testo in analisi, abbiamo notato che il trigramma "bui" è quello più frequente nel testo, in quanto appare ben sette volte, seguito dai trigrammi "evh", "glz","hss", che invece appaiono quattro volte. Inoltre notiamo come la quasi totalità delle distanze ricavate abbia come divisore comune otto, che diventa quindi il candidato più probabile per la lunghezza della chiave.

```
OKZARVGLNSLFOQRVVBPHHZAMOMEVHLBAITLZOWSXCSZFEQFICOOVDXCIISOOVXEIYWNHHLVQHSOWD
BRPTTZZOWJIYPJSAWQYNOYRDKBQKZPHHTLIHDEMICGYMSEVHKVXTQPBWMEWAZZKHLJMOVEVHJYSJR
ZTUMCVDGLZVBUIWOCPDZVEIGSOGZRGGOTAHLCSRSCXXAGPDYPSYMECRVPFHMYWZCYHKMCPVBPHYIF
WDZTGVIZEMONVYQYMCOOKDVQIMSOKLBUEBFZISWSTVFEWVIAWACCGHDRVVZOOBANRYHSSQBUIMSDW
VBNRXSSOGLVWKSCGHLNRYHSSQLVIYCFHVWJMOVEKRKBQMOOSVQAHDGLGWMEOMCYOXMEEIRTZBCFLZ
BVCVPRQVBLUHLGSBSEOUWHRYHSSEIEVDSCGHZRGOSOPBBUIQWNHRZFEIRPBWMEXCSPASBLXZFCWWW
EMZGLDDBUIOWNTHVPICOOTRZOMYRPBKMEIIHCOQKRWCSNFRAFIYWEKLBUSPHEVHAYMBVESVBGVZAZ
FVPRAJIWRQMIIMUZPDKXXJHSSRBUIMGTRHBUIMSHCXTQFZBZFHBHVIOYRWPRXCFPSRNGLZAVBHEVX
OVPMZMEIAIWZBIJEMSEVDBGLZMHSUMGVVWWWQOGLZCCPLARWYSNZLVRXCOEHKMLAZFPGLVXMIUHWW
PVXDBECWPRJDBLZQQTLOALFHBUIKOEVZWHPYPPRLNSMXIWHWPNXOCZHKMLOISH
```

Figure 1: Trigrammi che appaiono almeno quattro volte nel testo

b. Per implementare invece il metodo degli indici di coincidenza, che nel file è chiamato "trovalunghchiave", ho per prima cosa inserito due parametri in ingresso, oltre al testo cifrato, per indicare gli estremi dell'intervallo di valori tra cui cercare la possibile lunghezza della chiave, per evitare che la misura così ricavata non risulti pari ad un multiplo della lunghezza reale, in quanto gli indici di coincidenza risulterebbero essere simili o addirittura "migliori" di quelli originali poiché le stringhe ottenute sarebbero comunque cifrate sempre con lo stesso valore di chiave.
La mia implementazione quindi, per ogni possibile valore j della lunghezza della chiave all'interno dell'intervallo desiderato, simula la tabella costruita nel metodo degli indici di coincidenza andando a ricavare j sottostringhe composte da caratteri che distano j posizioni tra loro nella stringa. In particolare, il procedimento per ricavare le sottostringhe è stato implementato un metodo a parte, chiamato "vettcifrsost", in quanto sarà utile anche per la seconda parte dell'attacco. Per ognuna di queste sottostringhe calcolo quindi l'indice di coincidenza con il metodo visto nell'esercizio 3.1 per poi calcolare la media aritmetica di tutti gli indici così ottenuti. Dopo aver ripetuto questo procedimento per ogni valore dell'intervallo specificato, sceglierò come lunghezza della chiave quello associato ad un indice di coincidenza maggiore, che andrò quindi a restituire.
Riguardo al cyphertext che stiamo cercando di decifrare, usando il metodo appena illustrato con valori possibili della lunghezza compresi tra due e quindici, otteniamo che la lunghezza della chiave più plausibile risulta essere otto, come avevamo trovato anche utilizzando il metodo di Kasiski, per questo posso essere abbastanza sicuro che questo sia il valore giusto. In particolare, l'indice di coincidenza medio delle sottostringhe associato alla lunghezza pari ad otto è 0.072, non molto dissimile dal valore atteso per un testo di lingua inglese che è pari a 0.065.

c. Volendo infine implementare un metodo, chiamato "trovachiave", che esegua la seconda parte dell'attacco al cifrario di Vigenère, ho per prima cosa definito un array Numpy contenente la distribuzione di probabilità standard delle lettere nella lingua inglese. Successivamente ho im-

plementato un ciclo che, per ogni posizione nella chiave, riusa il metodo "vettcifrsost" visto al passo precedente per ottenere la sottostringa dei caratteri cifrati con la lettera presente in tale posizione nella chiave, per poi ricavare il vettore della distribuzione di probabilità delle lettere nel testo, grazie al metodo "vettord", che ho implementato a partire dal metodo "distngrammi" ordinando però il vettore della distribuzione in base all'ordine alfabetico e dando valore zero alla probabilità di una lettera che non appare nella stringa.
Successivamente, per ogni posizione della chiave, vado a considerare ogni lettera come possibile chiave della sottostringa, andando a shiftare il vettore della distribuzione di probabilità della sottostringa verso sinistra di un numero di posizioni pari al valore della lettera che sto testando. Una volta testati tutti i possibili shift cercherò quello che ha dato come risultato un prodotto scalare tra i due vettori delle distribuzioni di probabilità maggiore rispetto agli altri. Tale valore di shift corrisponderà quindi alla lettera nella posizione in analisi della chiave. Concatenando allora tutte le lettere così ricavate otterrò infine la chiave desiderata, che potrò dare in pasto ad un ultimo metodo, chiamato "decrittaVigenere", necessario per decrittare il testo cifrato con la chiave appena trovata, che non farà altro che andare a sottrarre in modulo 26 il valore della lettera della chiave associato alla posizione corrispondente.
La chiave di cifratura, utile per decifrare il testo in analisi, è stata quindi ricavata semplicemente dando in input al metodo "trovachiave" il testo cifrato e la lunghezza della chiave, che è pari a otto, ottenendo come risultato il termine ""volodine", con prodotto scalare tra il vettore della distribuzione di probabilità standard della lingua inglese e quello della distribuzione di probabilità delle lettere nelle sottostringhe con shift pari al valore della lettera ricavata pari a: 0.064 per la "v", 0.72 per la prima "o", 0.71 per la "l", 0.063 per la seconda"o", 0.72 per la "d", 0.071 per la "i", 0.62 per la "n" e 0.073 per la "e".
A questo punto è stato possibile decifrare il testo cifrato, grazie al metodo "decrittaVigenere" con la chiave appena trovata, ottendo così il testo in chiaro sotto riportato:
twomonthsearlieraneternitythedownfalloftheorbisehadhappenedaspredictedimmediatelyfollowed byexodusandacompletelyemptyfuturethecitycentersflowedwiththebloodofreprisalsthebarbarians hadreclaimedpowerjustlikeeverywhereelseontheplanetvassilissamarachvilihadwanderedwithagro upofpartisansforseveraldaysandthentheresistancehaddispersedandthendiedoutsowithtwocomrade sindisasterkronauerandilyushenkoshemanagedtogetaroundthebarrierserectedbythevictorsandent ertheemptyterritoriesapatheticfencehadforbiddenherentranceshecrosseditwithouttheslightesttrem orshewouldnevergobacktotheothersidetherewouldbenoreturnandthethreeofthemknewittheywerefullyaw arethattheyweretrailingtheorbisesdeclinethattheyweresinkingwithitintothefinalnightmarethepatwouldbedifficultthattootheyknew.
Questo brano è tratto dal romanzo di fantascienza "Terminus radioso" scritto da Antoine Volodine, il cui cognome è la chiave di cifratura del testo.