НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з переддипломної практики

Програмна реалізація алгоритмів розв'язування задачі побудови оцінок параметрів частково спостережуваного ланцюга Маркова на бінарних послідовностях

Виконав студент
групи ФІ-91
Цибульник А. В.

Науковий керівник:
ст. викл.
Наказной П. О.

Київ — 2023

ЗМІСТ

ВСТУП

Марковські моделі мають широкий та ефективний арсенал інструментів для аналізу динаміки систем, поведінка яких у кожен наступний момент часу зумовлюється лише поточним станом системи та не залежить від характеру еволюції у попередні моменти часу.

Водночас, у випадку, коли безпосереднє спостереження еволюції ланцюга Маркова є неможливим чи обмеженим, застосовують моделі прихованих ланцюгів Маркова (ПММ). У такому випадку аналіз поведінки процесу відбувається за деякою опосередкованою інформацією про «приховані», справжні стани ланцюга.

Наприклад, в біоінформатиці [1, глава 9] апарат ланцюгів Маркова застосовують при дослідженні еволюції молекул ДНК протягом певного часу, вважаючи при цьому за стан системи зв'язану послідовність так званих нуклеотидів, які формуються над алфавітом чотирьох азотистих основ $\{T, C, A, G\}$.

Існування статистичних залежностей в чергуванні фонем чи слів в природних мовах зумовлює ефективність використання прихованих марковських моделей до таких завдань, як створення голосових команди, служб транскрипції та голосових помічників [2].

Не винятком стають і задачі розпізнавання мови жестів [3]: наприклад, представляючи жести як послідовності прихованих станів, ПММ можуть розпізнавати динаміку та варіації рухів рук.

Відтак, враховуючи актуальність вивчення еволюції систем, стани яких є послідовностями чи наборами символів певної довжини, у роботі розглядається ланцюг Маркова на множині двійкових послідовностей, динаміка якого відстежується за зміною в часі набору функціоналів від його станів.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

У даному звіті викладена експериментальна перевірка ефективності використаних методів та алгоритмів для побудови оцінок невідомих параметрів частково спостережуваного ланцюга Маркова на бінарних послідовностях. А саме, продемонстровано результати таких задач:

1) задача навчання: за наявними спостереженнями про динаміку набору функціоналів від станів прихованого ланцюга бінарних послідовностей оцінено керуючий параметр системи, використовуючи математичий апарат прихованих марковських моделей;

2) задача декодування: за наявними спостереженнями та оцінкою керуючого параметра відновлено ланцюг прихованих станів;

3) задачу локалізації: за відомими значеннями набору функціонів від деякої невідомої підмножини стану прихованого ланцюга, оцінено потужність та набір елементів цієї підмножини;

4) задача навчання, окреслена в пункті 1), враховуючи, що наявні спостереження є зашумленими, спотвореними.

Для проведення чисельного експерименту, на основі якого виконуватиметься висновок щодо ефективності побудованих теоретичних оцінок невідомих параметрів, було розроблено відповідне програмне забезпечення.

1 ПОБУДОВА ТЕОРЕТИЧНИХ ОЦІНОК

У цьому розділі окреслимо побудову теоретичних оцінок для параметрів частково спостережуваного ланцюга Маркова на двійкових послідовностях.

1.1 Моделювання об'єкту дослідження

Розглянемо ланцюг Маркова $\{X^t\}_{t=\overline{1,T}}$, який приймає значення зі скінченної множини $E = \{0,1\}^N$ — множини всеможливих бінарних послідовностей довжини N.

Динаміка ланцюга відбувається згідно узагальненої моделі Еренфестів: в кожен момент часу t навмання обирається число j з множини індексів $\{1, 2, \ldots, N\}$ бінарної послідовності X^t та відповідний елемент стану X^t_j залишається незмінним з імовірністю p або змінюється на протилежний бінарний символ з імовірністю $1-p$.

Як наслідок окресленої динаміки, матриця перехідних імовірностей ланцюга матиме вигляд:

$$A_{xx'} = P\left(X^{t+1} = x' \mid X^t = x\right) = \begin{cases} p, & x' = x \\ \dfrac{1-p}{N}, & \begin{array}{l} x_j^{'} = 1 - x_j \\ \forall i \neq j : x_i^{'} = x_i \end{array} \\ 0, & \text{інакше} \end{cases}$$

Крім того, інваріантний розподіл $\pi = (\pi_x)_{x\in E}$ заданого ланцюга є рівномірним, тобто $\pi_x = \frac{1}{2^N}$. Вважатимемо, що початковий розподіл збігається з π.

Наступним кроком введемо послідовність випадкових величин $\{Y^t\}_{t=\overline{1,T}}$, які формуються таким чином:

$$Y^t = \left(Y_k^t\right)_{k=\overline{1,L}} = \left(\phi\left(X^t, I_k\right)\right)_{k=\overline{1,L}}, \ t = \overline{1,T}, \tag{1.1}$$

де $I = \{I_1, \ldots, I_L\}$ — задані підмножини множини індексів $\{1, 2, \ldots, N\}$, а функціонал ϕ визначимо так:

$$\phi\left(X^t, I_k\right) = \sum_{i \in I_k} X_i^t \tag{1.2}$$

Твердження 1.1. *Послідовність* $\{(X^t, Y^t)\}_{t=\overline{1,T}}$ *утворює приховану марковську модель* (π, A, B), *де*

$$B_{xy} = P\left(Y^t = y \,|\, X^t = x\right) = \prod_{k=1}^{L} \mathbb{1}\left(y_k = \sum_{i \in I_k} x_i\right),$$

і позначено $\mathbb{1}$ *— індикаторна функція.*

1.2 Постановка завдання

За спостереженнями (1.1) прихованої марковської моделі слід знайти розв'язки задач:

1) Оцінити невідомий «параметр мутації» p елементів бінарних послідовностей прихованого ланцюга Маркова та декодувати послідовність станів прихованого ланцюга;

2) Вважаючи, що спостерігається деяке додаткове значення функціонала (1.2) від прихованих станів ланцюга по невідомій «множині неявних індексів» I_*, оцінити потужність цієї множини та відтворити набір її елементів;

3) Вважаючи, що значення введеного функціонала (1.2) від прихованих

станів ланцюга по множинам $I_1, \ldots, I_L$ спостерігаються так:

$$\phi\left(X^t, I_k\right) = \sum_{i \in I_k} \widetilde{X}_i^t,\ k = \overline{1, L}, \tag{1.3}$$

де для $i \in I_k$

$$\widetilde{X}_i^t = \begin{cases} 1 - X_i^t, & \text{з імовірністю } q_k \\ X_i^t, & \text{з імовірністю } 1 - q_k \end{cases}, \tag{1.4}$$

оцінити невідомий параметр моделі p та ймовірності спотворень $q_1, q_2, \ldots, q_L$.

1.2.1 Оцінка невідомого параметра моделі

Алгоритм навчання Баума-Велша

Спостерігаючи (1.1), скористаємося методом максимальної правдоподібності, шукаючи оцінку невідомого параметра p таким чином:

$$\widehat{p} = \underset{p}{\operatorname{argmax}} \sum_{x \in E^T} L_{p,x,y},$$

де

$$L_{p,x,y} = P\left(X = x,\, Y = y \,|\, p\right) \tag{1.5}$$
$$X = x \Longleftrightarrow \left(X^1 = x^1, \ldots, X^T = x^T\right)$$
$$Y = y \Longleftrightarrow \left(Y^1 = y^1, \ldots, Y^T = y^T\right)$$

Щоправда, для заданої марковської моделі вигляд функції правдоподібності (1.5) матиме громіздкий та неможливий для безпосереднього диференціювання вигляд.

Однак, в такому випадку можна застосувати модифікацію ЕМ-алгоритму для дослідження прихованих ланцюгів Маркова — ітераційний алгоритм Баума-Велша [1, розділ 15].

Задавши деяке наближення $p^{(0)}$ невідомого параметра p, покладемо

$$p^{(n+1)} = \underset{p}{\operatorname{argmax}}\, Q\left(p^{(n)}, p\right),$$

де

$$Q\left(p^{(n)}, p\right) = \sum_{x \in E^T} L_{p^{(n)},x,y} \cdot \ln L_{p,x,y} \tag{1.6}$$

є так званою функцією квазі-log правдоподібності.

Доведено [1, розділ 4], що така ітераційна процедура є збіжною і приводить до точки локального максимуму логарифму функції правдоподібності (1.5).

Максимізація функції (1.6) приводить до такої ітераційної формули переоцінки параметра p :

$$p^{(n+1)} = p^{(n)} \cdot \frac{\sum\limits_{t=1}^{T-1} \sum\limits_{x \in E} \alpha_t(x)\, B_{xy^{t+1}}\, \beta_{t+1}(x)}{\sum\limits_{t=1}^{T-1} \sum\limits_{x \in E} \alpha_t(x)\, \beta_t(x)}, \tag{1.7}$$

де

$$\alpha_t(x) = P\left(Y^1 = y^1, \ldots, Y^t = y^t,\, X^t = x \,|\, p^{(n)}\right) \tag{1.8}$$

$$\beta_t(x) = P\left(Y^{t+1} = y^{t+1}, \ldots, Y^T = y^T \,|\, X^t = x,\, p^{(n)}\right) \tag{1.9}$$

так звані коефіцієнти прямого та зворотного ходу відповідно [4, розділ 5].

Алгоритм декодування Вітербі

Використовуючи оцінене значення параметра $\widehat{p}$, отримане в результаті застосування алгоритму навчання Баума-Велша, скористаємося алгоритмом декодування Вітербі [4, розділ 6] для пошуку такої послідовності прихованих станів $\widehat{X}^1, \widehat{X}^2, \ldots, \widehat{X}^T$, яка найкращим чином описує наявні спостереження:

$$\widehat{X} = \operatorname*{argmax}_{x \in E^T} P\left(X = x \,|\, Y = y,\, \widehat{p}\,\right)$$

1.2.2 Оцінка множини неявних індексів

Нехай окрім набору спостережень (1.1) протягом еволюції ланцюга на кожному кроці t спостерігається деяке додаткове значення $Y_{I_*}^t$ функціонала (1.2) від прихованого стану ланцюга по деякій невідомій підмножині індексів $I_* \subseteq \{1, 2, \ldots, N\}$:

$$Y_{I_*} = \left(Y_{I_*}^t\right)_{t=\overline{1,T}} = \left(\sum_{i \in I_*} X_i^t\right)_{t=\overline{1,T}}$$

Перш за все, оцінимо потужність множини I_*. Зауважимо, що в силу заданого способу еволюції прихованого ланцюга Маркова

$$P\left(Y_{I_*}^t = Y_{I_*}^{t+1}\right) = \frac{|I_*|}{N} \cdot p + \frac{N - |I_*|}{N}$$

Ця рівність дозволяє побудувати незміщену та змістовну оцінку для потужності $|I_*|$.

Твердження 1.2. *Змістовною і незміщеною оцінкою потужності множини I_* є статистика*

$$\widehat{|I_*|} = \frac{N}{1-p}\left(1 - \frac{1}{T-1}\sum_{t=1}^{T-1} \mathbb{1}\left(Y_{I_*}^t = Y_{I_*}^{t+1}\right)\right) \tag{1.10}$$

Аналогічним чином побудуємо оцінку для потужності перетину множини I_* з індексами множин, які задають спостереження моделі. Вказана оцінка дозволить виявити взаємне розташування елементів множини неявних індексів та множини доступних для дослідження елементів прихованого стану ланцюга Маркова.

Твердження 1.3. *Нехай $H \subseteq I_1 \cup I_2 \cup \ldots \cup I_L$ — довільна підмножина множини спостережуваних індексів $I_1 \cup I_2 \cup \ldots \cup I_L$. Тоді змістовною та незміщеною оцінкою потужності множини $I_* \cap H$ є статистика*

$$|\widehat{I_* \cap H}| = \frac{N}{(T-1)(1-p)} \cdot \sum_{t=1}^{T-1} \mathbb{1}\left(Y_{I_*}^t \neq Y_{I_*}^{t+1},\, Y_H^t \neq Y_H^{t+1}\right) \qquad (1.11)$$

Стратегія визначення елементів, які безпосередньо входять в множину I_*, складатиметься з декількох кроків:

1) із загальної множини індексів $\{1, 2, \ldots, N\}$ сформувати всеможливі підмножини довжиною $|\widehat{I_*}|$, тобто вибірку

$$\left\{\mathtt{I}_1, \mathtt{I}_2, \ldots, \mathtt{I}_{C_N^{|\widehat{I_*}|}}\right\} \qquad (1.12)$$

2) для кожного «кандидата» $\mathtt{I}_\mathtt{k}$ з множини (1.12) згенерувати послідовність значень функціонала (1.2) від декодованих прихованих станів по відповідних індексах:

$$\widehat{Y}_{\mathtt{I}_\mathtt{k}} = \left(\widehat{Y}^{\,t}_{\mathtt{I}_\mathtt{k}}\right)_{t=\overline{1,T}} = \left(\sum_{i \in \mathtt{I}_\mathtt{k}} \widehat{X}_i^t\right)_{t=\overline{1,T}}$$

3) за допомогою деякої заданої міри d оцінити для кожного $\mathtt{I}_\mathtt{k}$ відстань між наборами $\widehat{Y}_{\mathtt{I}_\mathtt{k}}$ та Y_{I_*};

4) оцінкою $\widehat{I}$ множини I_* стане той «кандидат» $\mathtt{I}_\mathtt{k}$ з множини (1.12), для якого d буде найменшою:

$$\widehat{I} = \underset{1 \leqslant k \leqslant C_N^{|\widehat{I_*}|}}{\operatorname{argmin}}\, d\left(\widehat{Y}_{\mathtt{I}_\mathtt{k}}, Y_{I_*}\right) \qquad (1.13)$$

Міру близькості d між двома невід'ємними цілочисельними множинами $\widehat{Y}_{\mathtt{I}_\mathtt{k}}$ та Y_{I_*} однакової довжини визначатимемо або за допомогою середньоквадратичної відстані

$$d_S\left(\widehat{Y}_{\mathtt{I}_\mathtt{k}}, Y_{I_*}\right) = \sum_{t=1}^{T}\left(\widehat{Y}^{\,t}_{\mathtt{I}_\mathtt{k}} - Y_{I_*}^t\right)^2, \qquad (1.14)$$

або користуючись зваженою відстанню Жаккара [5]

$$d_J\left(\widehat{Y}_{I_k}, Y_{I_*}\right) = 1 - \frac{\sum\limits_{t=1}^{T} \min\left(\widehat{Y}^{\,t}_{I_k}, Y^t_{I_*}\right)}{\sum\limits_{t=1}^{T} \max\left(\widehat{Y}^{\,t}_{I_k}, Y^t_{I_*}\right)} \tag{1.15}$$

1.2.3 Оцінка коефіцієнтів спотворення

Припустимо, що значення функціонала (1.2) від прихованих станів ланцюга $\{X^t\}_{t=\overline{1,T}}$ по множинам $I_1, \ldots, I_L$ спостерігаються із деякими ймовірностями спотворення $q_1, q_2, \ldots, q_L$ згідно (1.3) та (1.4).

Оцінимо параметр p та вектор імовірностей спотворень $q = (q_1, q_2, \ldots, q_L)$, використовуючи ітераційний алгоритм Баума-Велша.

Твердження 1.4. *Якщо множини $I_1, \ldots, I_L$ є попарно неперетинними, то утворена послідовність $\{(X^t, Y^t)\}_{t=\overline{1,T}}$ є прихованою марковською моделлю (π, A, B^q), де*

$$B^q_{xy} = P\left(Y^t = y \,|\, X^t = x\right) = \prod_{k=1}^{L} P\left(\xi^k_{01}(x) + \xi^k_{11}(x) = y_k\right),$$

і для довільного $k = \overline{1, L}$

$$\xi^k_{01}(x) \sim Bin\left(|I_k| - \sum_{i \in I_k} x_i,\, q_k\right),\ \xi^k_{11}(x) \sim Bin\left(\sum_{i \in I_k} x_i,\, 1 - q_k\right)$$

є незалежними випадковими величинами.

Виберемо деяке початкове наближення моделі $\left(\pi, A^{(0)}, B^{q^{(0)}}\right)$, визначимо коефіцієнти прямого (1.8) та зворотного (1.9) ходу. Тоді ітераційна формула переоцінки параметра p матиме вид:

$$p^{(n+1)} = p^{(n)} \cdot \frac{\sum\limits_{t=1}^{T-1} \sum\limits_{x \in E} \alpha_t(x)\, B^{q^{(n)}}_{xy^{t+1}}\, \beta_{t+1}(x)}{\sum\limits_{t=1}^{T-1} \sum\limits_{x \in E} \alpha_t(x)\, \beta_t(x)}, \tag{1.16}$$

а формула переоцінки компонент вектора $(q_k)_{k=\overline{1,L}}$:

$$q_k^{(n+1)} = q_k^{(n)} \cdot \frac{\sum\limits_{t=1}^{T} \sum\limits_{x \in E} \beta_t(x) \sum\limits_{x' \in E} \alpha_{t-1}(x')\, A_{x'x}^{(n)} \sum\limits_{i \in I_k} P_{x,i}^{q^{(n)}}}{|I_k| \sum\limits_{t=1}^{T} \sum\limits_{x \in E} \alpha_t(x)\, \beta_t(x)}, \tag{1.17}$$

де при $i \in I_m$

$$P_{x,i}^{q} = P\left(\widetilde{\xi_{01}^{m}}(x) + \widetilde{\xi_{11}^{m}}(x) = y_m + x_i - 1\right) \cdot \prod_{\substack{k=\overline{1,L} \\ k \neq m}} P\left(\xi_{01}^{k}(x) + \xi_{11}^{k}(x) = y_k\right)$$

та

$$\widetilde{\xi_{01}^{m}}(x) \sim Bin\left(|I_m| - 1 - \sum_{j \in I_m \setminus \{i\}} x_j,\, q_m\right),\ \widetilde{\xi_{11}^{m}}(x) \sim Bin\left(\sum_{j \in I_m \setminus \{i\}} x_j,\, 1 - q_m\right)$$

Наостанок зауважимо, що при великих значеннях довжини ланцюга $(T > 300)$ виникає потреба у шкалюванні [4, розділ 5] коефіцієнтів прямого та зворотного ходу, адже їхні значення стають нерозрізнювано малими для обчислювальних ресурсів. Процедура нормування не вносить змін у вигляд ітераційних формул переоцінки (1.7), (1.16) чи (1.17).

Висновки до розділу 1

Оскільки задана в рамках дослідження модель відповідає означенню прихованої Марковської моделі, пробудову теоретичних оцінок невідомих параметрів було виконано за допомогою математичного апарату ланцюгів Маркова. Крім того, для задачі локалізації було отримано серію оцінок, використовуючи методи математичної статистики.

У наступному розділі буде продемонстрована експериментальна перевірка ефективності виведених теоретичних оцінок.

2 РЕЗУЛЬТАТИ ЧИСЕЛЬНОГО ЕКСПЕРИМЕНТУ

Для програмної реалізації реалізація алгоритмів розв'язування задачі побудови оцінок невідомих параметрів моделі було використано засоби мови програмування `Python` версії `3.8.10` в інтегрованому середовищі розробки `Visual Studio Code` версії `1.78.2`.

Вибір мови програмування зумовлювався широким арсеналом вбудованих програмних пакетів мови `Python` для роботи з масивами даних та математичними обчисленнями (бібліотеки `NumPy`, `itertools`, `SciPy`, `random`, `numda`), а також наявними інструментами для візуалізації даних (пакети `pandas`, `matplotlib`). Додаток Б містить тексти ключових програмних блоків коду, необхідних для реалізації чисельного експерименту.

Крім того, для ефективного керування великою кількістю взаємопов'язаних програмних блоків (функцій), а також для більш наочної демонстрації отриманих результатів було розроблено графічний інтерфейс користувача засобами пакету `PySimpleGUI` мови `Python`. Додаток А містить опис та приклад роботи розробленого програмного модуля.

2.1 Оцінка невідомого параметра моделі

У рамках чисельного експерименту було згенеровано прихований ланцюг Маркова протягом $T = 200$ моментів часу для бінарних послідовностей довжини $N = 5$ при заданому параметрі моделі $p = 0.2$. Множину спостережуваних індексів було задано таким чином:

$$I = \{I_1, I_2\} = \{(2,3),(1,4)\} \tag{2.1}$$

Рис. 2.1 демонструє збіжність алгоритму Баума-Велша при оцінці параметра p. Червоним кольором позначена початкова ітерація $p^{(0)} = 0.55$.

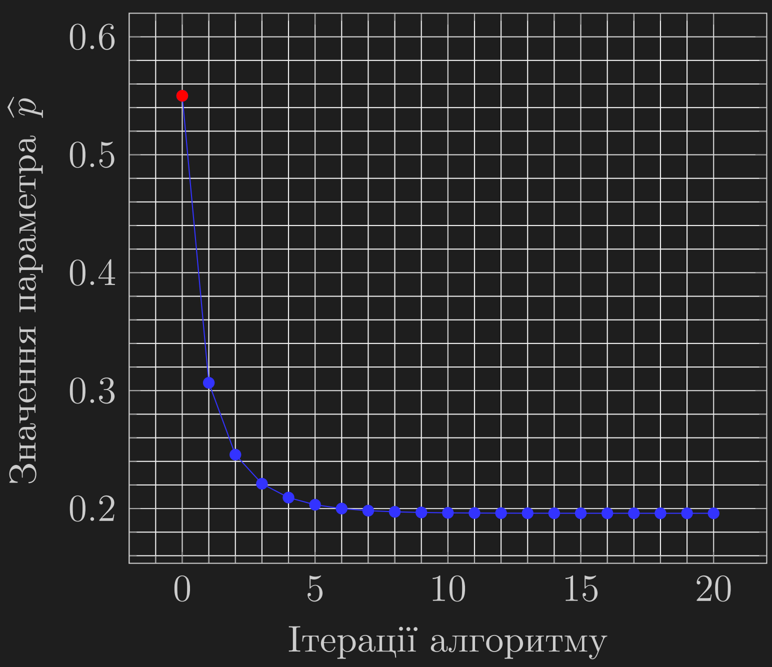

Рисунок 2.1 – Ітерації алгоритму Баума-Велша для оцінки параметра p

За $n = 12$ ітерацій алгоритм досягає точності $\varepsilon = 0.0001$ переоцінки оцінюваного параметра. При цьому, отримане значення $\widehat{p} = 0.1959$ відрізняється від свого істинного значення $p = 0.2$ на величину $\delta = 0.0041$.

2.2 Алгоритм декодування прихованих станів

Наступним кроком, отримавши оцінене значення $\widehat{p}$, декодуємо ланцюг прихованих станів за допомогою алгоритму Вітербі [4, розділ 6].

Якість отриманих результатів оцінимо через порівняння в кожен момент часу t істинної прихованої бінарної послідовності X^t та декодованої $\widehat{X^t}$ за допомогою відстані Геммінга:

$$d_H\left(X^t, \widehat{X^t}\right) = \sum_{i=1}^{N} \mathbb{1}\left(X_i^t \neq \widehat{X_i^t}\right)$$

Таким чином, чим більше символів між справжнім та декодованим станами збігаються, тим меншою буде відповідна відстань Геммінга d_H.

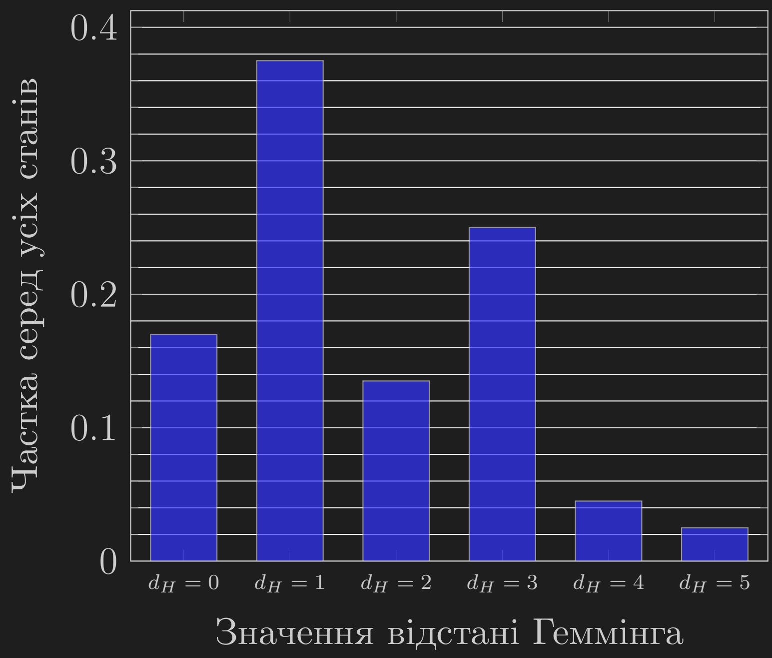

Рисунок 2.2 – Результати алгоритму декодування Вітербі

З гістограми результатів (Рис. 2.2) видно, що 17% усього ланцюга декодовано правильно. Наявність близько 40% помилок в одному символі може бути наслідком того, що одного елемента стану немає серед спостережуваних областей ланцюга. Крім того, оцінений параметр $\widehat{p}$ має похибку $\delta = 0.0041$ відносно свого істинного значення, що також впливає на результати задачі декодування.

2.3 Оцінка множини неявних індексів

В ролі множини неявних індексів було обрано набір $I_* = (1, 3, 5)$. В Табл. 2.1 показано збіжність змістовної та незміщеної оцінки (1.10) потужності $|\widehat{I_*}|$. Бачимо, що довжини ланцюга $T = 200$ недостатньо для отримання точної оцінки.

Таблиця 2.1 – Залежність значення оцінки $\widehat{|I_*|}$ від довжини ланцюга

T	200	400	600	800	1000
$\widehat{p}$	0.1959	0.1823	0.1882	0.2099	0.2092
$\widehat{\lvert I_* \rvert}$	2	2	2	3	3

Однак, оскільки обране значення N є невеликим, для оцінки потужності множини неявних індексів в такому випадку можна використати емпіричну оцінку вигляду:

$$\widehat{|I_*|} = \max_{1 \leqslant t \leqslant T} Y_{I_*}^t$$

Застосуємо отримане значення потужності для виразу (1.13), щоб віднайти елементи, які безпосередньо входять в I_* : квадратична відстань (1.14) вказує на сукупність $\widehat{I}_S = (1, 2, 5)$, а зважена відстань Жаккара (1.15) — на сукупність $\widehat{I}_J = (1, 2, 3)$.

Дилему можна вирішити шляхом збільшення T та подальшого використання змістовної оцінки (1.11) для визначення взаємного розташування елементів множини неявних індексів відносно спостережуваних індексів (2.1).

2.4 Оцінка коефіцієнтів спотворення

Для кожної із спостережуваних областей (2.1) змодельованого ланцюга було обрано такі ймовірності викривлення: $q = (q_1,\, q_2) = (0.05,\, 0.1)$.

Рис. 2.3 та Рис. 2.4 демонструють результати переоцінки невідомих параметрів моделі. Червоним кольором позначені значення початкових наближень $p^{(0)} = 0.55$ та $q^{(0)} = (0.3,\, 0.4)$.

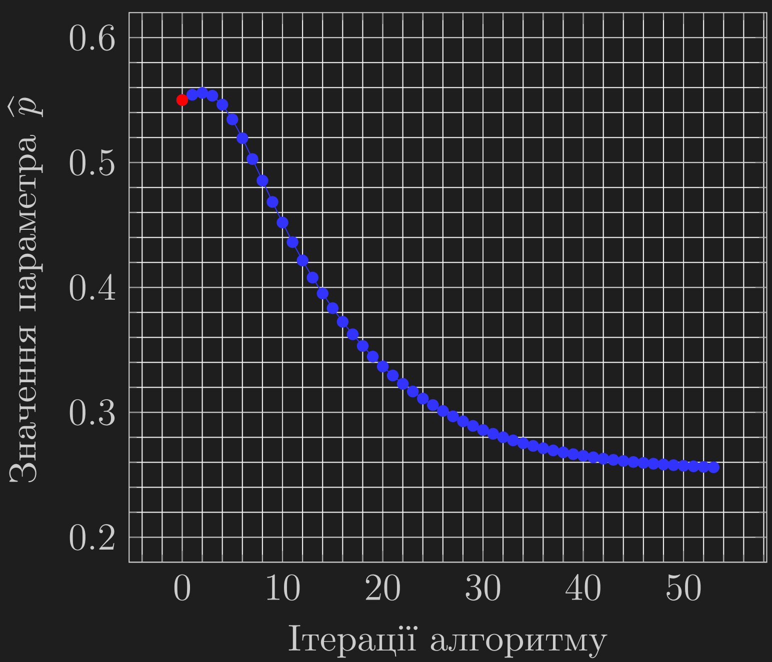

Рисунок 2.3 – Ітерації алгоритму Баума-Велша для оцінки параметра p, враховуючи спотворення спостережень

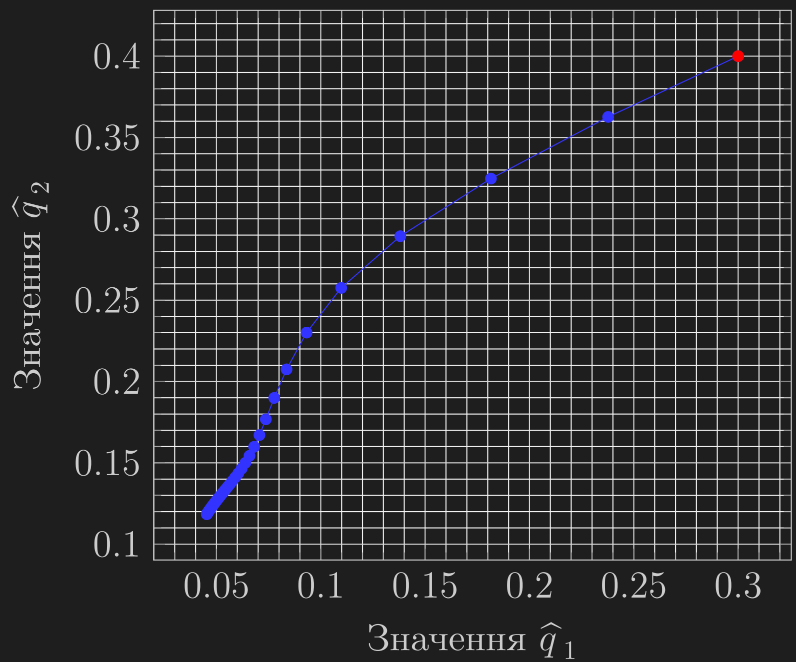

Рисунок 2.4 – Ітерації алгоритму Баума-Велша для оцінки компонент вектора q, враховуючи спотворення спостережень

Для досягнення точності переоцінки $\varepsilon = 0.0001$ оцінюваного параметра p у випадку спотворених даних знадобилося $n = 53$ ітерацій. При цьому, помітне збільшення похибки: отримане значення $\widehat{p} = 0.2559$ відрізняється від свого істинного значення $p = 0.2$ на суттєво вищий показник $\delta = 0.0559$. Водночас, точність оцінки коефіцієнтів спостворення $\widehat{q} = (\widehat{q}_1, \widehat{q}_2) = (0.0454, 0.1184)$ є високою: $\delta = (\delta_1, \delta_2) = (0.0046, 0.0184)$.

Висновки до розділу 2

Результати чисельного експерименту продемонстрували ефективність використаних методів, зокрема збіжність побудованих оцінок до істинних значень параметрів при збільшенні кількості спостережень.

ВИСНОВКИ

У першій частині звіту з переддипломної практики описані теоретичні викладки для розв'язування задачі побудови оцінок параметрів частково спостережуваного ланцюга Маркова на двійкових послідовностях. Невідомі параметри заданої моделі були оцінені або шляхом побудови змістовних та незміщених статистичних оцінок, або за допомогою ітераційного алгоритму Баума-Велша.

Друга частина звіту з переддипломної практики присвячена проведенню чисельного експерименту, результати якого продемонстрували ефективність використаних методів, зокрема збіжність побудованих оцінок до істинних значень параметрів при збільшенні кількості спостережень.

У рамках подальшого дослідження буде розглянута така постановка задачі навчання: за наявними спостереженнями про динаміку набору функціоналів від станів прихованого ланцюга бінарних послідовностей оцінити керуючий параметр системи, використовуючи методи побудови статистичних оцінок.

ПЕРЕЛІК ПОСИЛАНЬ

1. *Koski T.* Hidden Markov models for bioinformatics. — 2002-е вид. — New York, NY : Springer, 11.2001. — (Computational Biology).
2. *Rabiner L.* A tutorial on hidden Markov models and selected applications in speech recognition // Proceedings of the IEEE. — 1989. — Т. 77, № 2. — С. 257—286. — DOI: 10.1109/5.18626. — URL: https://doi.org/10.1109/5.18626.
3. *Chaaraoui A. A.*, *Climent-Pérez P.*, *Flórez-Revuelta F.* One shot learning for gesture recognition using HMMs and hand appearance features // Pattern Recognition Letters. — 2013. — Т. 34, № 9. — С. 1009—1017.
4. *Nilsson M.* First Order Hidden Markov Model: Theory and Implementation Issues : тех. звіт. / Blekinge Institute of Technology, School of Engineering, Department of Signal Processing. — 2005.
5. Finding the Jaccard Median / F. Chierichetti [та ін.] // Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). — 2010. — С. 293—311. — DOI: 10.1137/1.9781611973075.25. — eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973075.25. — URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611973075.25.

ДОДАТОК А ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА

Розробка програмного забезпечення включала у себе як імплементацію ітераційних формул переоцінки параметрів моделі, так і розробку графічного інтерфейсу користувача для ефективного керування різними блоками коду.

Наприклад, на малюнку нижче продемонстровано задання необхідних вхідних даних для генерування відповідного ланцюга Маркова для подальшого розв'язання задачі відновлення елементів множини неявних індексів.

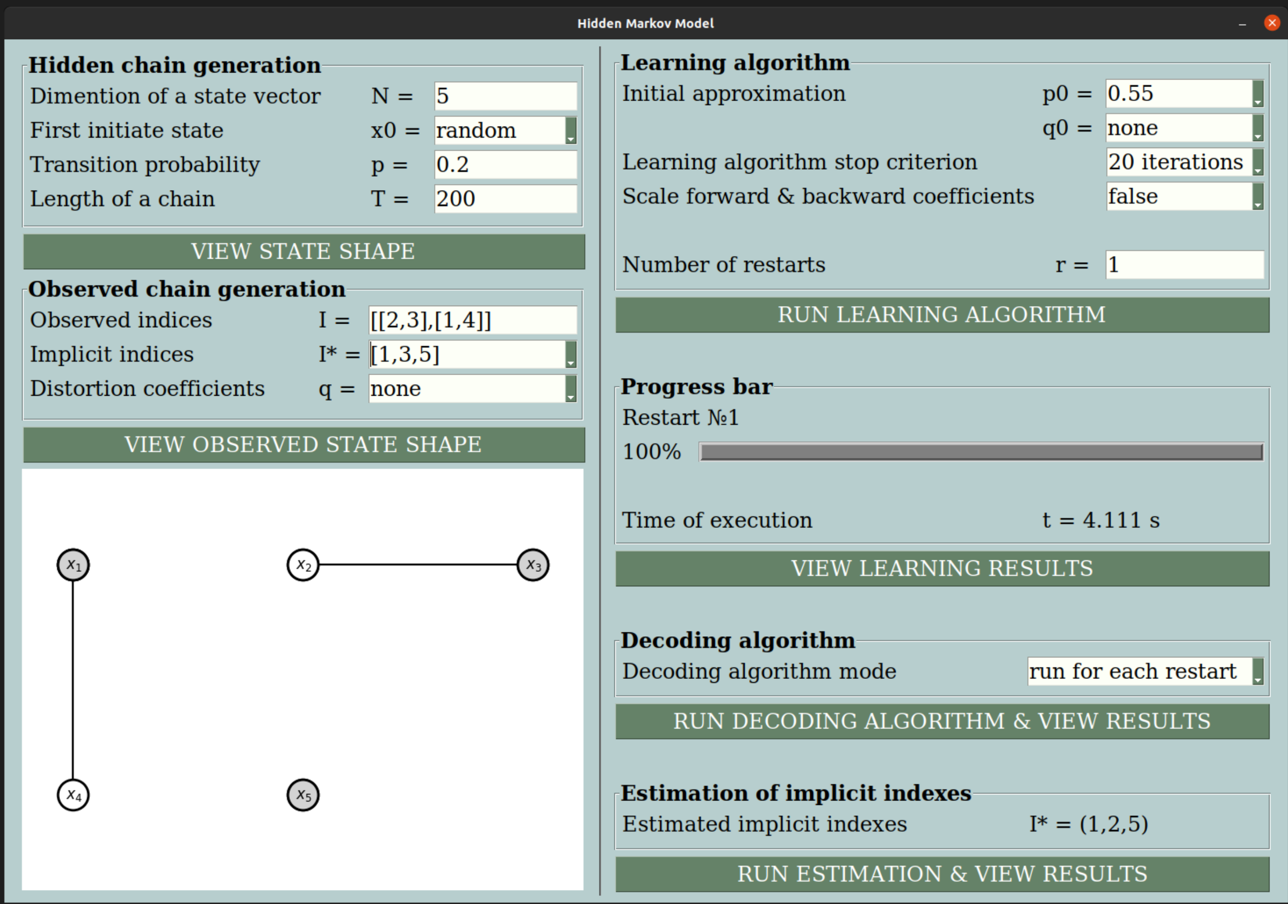

Рисунок А.1 – Графічний інтерфейс користувача

Кожна з відповідних кнопок викликає блоки коду, необхідні для виконання тієї чи іншої задачі. Візуалізація результатів виконується у вигляді наведених у цьому розділі графіків, які демонструються в окремих спливаючих вікнах інтерфейсу.

У лівій нижній частині інтерфейсу схематично зображена конфігурація стану ланцюга: відповідні множини спостережуваних індексів (кожна спостережувана область виокремлюється візуально за з'єднаними ребром вершинами), а також множина неявних індексів, елементи якої позначені сірим кольором.

ДОДАТОК Б ТЕКСТИ ПРОГРАМ

У цьому додатку наведені тексти ключових інструментальних програм для проведення експериментальних досліджень. Перелік необхідних бібліотек мови Python наведений нижче:

```
from __future__ import annotations # enable using function specifications
import numpy.typing as npt
import typing

import numpy as np
import itertools
import copy
```

Б.1 Обчислення коефіцієнтів прямого та зворотного ходу

```
def alpha_calculation(
        y: npt.NDArray[np.int64],
        m: npt.NDArray[np.float64],
        A: npt.NDArray[np.float64],
        B: npt.NDArray[np.float64],
        T: int,
        *args: list[float]
    ):
    """
    Return forward algorithm coefficients

    Parameters
    ----------
    y : int array(T,)
        Chain of observations
    m : float array(pow(2,N),)
        Initial distribution
    A : float array(pow(2,N),pow(2,N))
        Transition matrix
    B : float array(pow(2,N), depends on I)
        Emission matrix
    T : int
        Length of a chain
    *args : coefficients of scaling (optional, used only for scaled forward algorithm)

    Returns
    -------
    alpha : float array(pow(2,N),T)
        Forward algorithm coefficients
    P : float
        Probability of P(Y=y)
    scaler : float array(pow(2,N),T)
        Coefficients of scaling (optional, used only for scaled forward algorithm)
    """

    alpha = np.zeros((T, len(B)))

```

```
    if len(args) == 0:
        for t in range(T):
            for i in range(len(B)):
                if t == 0:
                    alpha[t][i] = m[i]*B[i][y[t]]
                else:
                    aA = 0.0
                    for j in range(len(B)):
                        aA += alpha[t-1][j]*A[j][i]
                    alpha[t][i] = aA*B[i][y[t]]

        P = 0
        for i in range(len(alpha[T-1])):
            P += alpha[T-1][i]

        return alpha, P

    if len(args) != 0:
        for t in range(T):
            for i in range(len(B)):
                if t == 0:
                    alpha[t][i] = m[i]*B[i][y[t]]
                    args[0][t] += alpha[t][i]
                else:
                    aA = 0.0
                    for j in range(len(B)):
                        aA += alpha[t-1][j]*A[j][i]
                    alpha[t][i] = aA*B[i][y[t]]
                    args[0][t] += alpha[t][i]

            for i in range(len(B)):
                alpha[t][i] = alpha[t][i]/args[0][t]

        P = 0
        for t in range(T):
            P += np.log(args[0][t])

        return alpha, P, args[0]
```

```
def beta_calculation(
        y: npt.NDArray[np.int64],
        A: npt.NDArray[np.float64],
        B: npt.NDArray[np.float64],
        T: int,
        *args: list[float]
    ):
    """
    Return backward algorithm coefficients

    Parameters
    ----------
    y : int array(T,)
        Chain of observations
    A : float array(pow(2,N),pow(2,N))
        Transition matrix
    B : float array(pow(2,N), depends on I)
        Emission matrix
    T : int
        Length of a chain
    *args : coefficients of scaling (optional, used only for scaled backward algorithm)


```

```
    Returns
    -------
    beta : float array(pow(2,N),T)
        Backward algorithm coefficients
    """

    beta = np.zeros((T, len(B)))

    if len(args) == 0:
        for t in range(T-1, -1, -1):
            for i in range(len(B)):
                if t == T-1:
                    beta[t][i] = 1
                else:
                    bAB = 0.0
                    for j in range(len(B)):
                        bAB += beta[t+1][j]*A[i][j]*B[j][y[t+1]]
                    beta[t][i] = bAB

    if len(args) != 0:
        for t in range(T-1, -1, -1):
            for i in range(len(B)):
                if t == T-1:
                    beta[t][i] = 1
                else:
                    bAB = 0.0
                    for j in range(len(B)):
                        bAB += beta[t+1][j]*A[i][j]*B[j][y[t+1]]
                    beta[t][i] = bAB

            for i in range(len(B)):
                beta[t][i] = beta[t][i]/args[0][t]
```

Б.2 Алгоритм Баума-Велша

```
def learning_algorithm(
        y: npt.NDArray[np.int64],
        N: int,
        T: int,
        I: list[list[int]],
        estimator: typing.Literal[
            "parameter p estimation task (distortion-free model)",
            "parameter p estimation task (model with distortion)",
            "parameter p and coefficients q estimation task"
        ],
        p0: float,
        q0: float,
        scaling: bool
    ):
    """
    Return learning algorithm results (estimated parameters)

    Parameters
    ----------
    y : int array(T,)
        Chain of observations
    N : int
        Dimention of any state vector

```

```
    T : int
        Length of a chain
    I : int array
        Set of observed indices
    estimator : string
        Type of estimation (only p or both p & q estimation)
    p0 : float
        Initial approximation of parameter p
    q0 : float array(len(I),)
        Initial approximation of distortion coefficients q
    scaling : string
        Either to scale forward and backward coefficients or not

    Returns
    -------
    parameter : float array
        List of estimated parameters [p,q] for each iteration
    joint_probabilities : float array
        List of P(Y=y) probabilities (concerns to forward algorithm)
    joint_probabilities_increments : float array
        List of P(Y=y) increments through iterations (concerns to forward algorithm)
    """

    joint_probabilities = []
    joint_probabilities_increments = []

    if estimator == "parameter p and coefficients q estimation task":
        parameter = []
        parameter.append([p0, q0])

        p = copy.deepcopy(parameter[0][0])
        q = copy.deepcopy(parameter[0][1])
    else:
        parameter = []
        parameter.append([p0])

        p = copy.deepcopy(parameter[0][0])

    number_of_iterations = 0

    while (
        number_of_iterations < 20 or
        abs(1-joint_probabilities[-1]/joint_probabilities[-2]) > 0.0001
    ):
        if estimator == "parameter p estimation task (distortion-free model)":
            m,A,B = probability_measures_of_HMM(p,N,I)
        elif estimator == "parameter p estimation task (model with distortion)":
            m,A,B = probability_measures_of_HMM(p,N,I,q0)
        elif estimator == "parameter p and coefficients q estimation task":
            m,A,B = probability_measures_of_HMM(p,N,I,q)

        if scaling == "false":
            alpha,P = alpha_calculation(y,m,A,B,T)
            beta = beta_calculation(y,A,B,T)
        else:
            scaler = np.zeros(T)
            alpha,P,scaler = alpha_calculation(y,m,A,B,T,scaler)
            beta = beta_calculation(y,A,B,T,scaler)

        joint_probabilities.append(copy.deepcopy(P))

        if number_of_iterations == 0:
            joint_probabilities_increments.append(copy.deepcopy(P))
        else:
            joint_probabilities_increments.append(
```

```
            abs(copy.deepcopy(P) - joint_probabilities_increments[-1])
        )

    p_numerator = calculate_p_numerator(y,alpha,beta,A,B,N,T)
    p_denominator = calculate_p_denominator(alpha,N,T)
    p = p_numerator/p_denominator

    parameter.append([copy.deepcopy(p)])

    if estimator == "parameter p and coefficients q estimation task":
        for j in range(len(I)):
            part_1 = calculate_q_estimation_part_1(y,q[j],alpha,beta,A,N,T,I,j)
            part_2 = calculate_q_estimation_part_2(y,q[j],alpha,beta,A,N,T,I,j)

            qj_numerator = copy.deepcopy(part_1)
            qj_denominator = copy.deepcopy(part_1) + copy.deepcopy(part_2)

            q[j] = qj_numerator/qj_denominator

        parameter[number_of_iterations + 1].append(copy.deepcopy(q))

    number_of_iterations += 1

return parameter, joint_probabilities, joint_probabilities_increments
```

Б.3 Алгоритм Вітербі

```
def viterbi(
        y: npt.NDArray[np.int64],
        m: npt.NDArray[np.float64],
        A: npt.NDArray[np.float64],
        B: npt.NDArray[np.float64],
        T: int
    ) -> npt.NDArray[np.int64]:
    """
    Return decoded state chain

    Parameters
    ----------
    y : int array(T,)
        Chain of observations
    m : float array(pow(2,N),)
        Initial distribution
    A : float array(pow(2,N),pow(2,N))
        Transition matrix
    B : float array(pow(2,N), depends on I)
        Emission matrix
    T : int
        Length of a chain
    *args : coefficients of scaling (optional, used only for scaled forward algorithm)

    Returns
    -------
    x : int array(T,)
        Decoded enumarated state chain
    """

    delta = [[0.0 for i in range(len(B))] for t in range(T)]
    psi = [[0 for i in range(len(B))] for t in range(T)]
```

```
    for t in range(T):
        for i in range(len(B)):
            if t == 0:
                delta[t][i] = m[i]*B[i][y[t]]
            else:
                dA = []
                for j in range(len(B)):
                    dA.append(delta[t-1][j]*A[j][i]*B[i][y[t]])
                delta[t][i] = max(dA)
                psi[t][i] = np.argmax(dA)

    x = []
    delta_hat = max(delta[T-1])
    x.append(np.argmax(delta[T-1]))

    for t in range(T-2, -1, -1):
        x.insert(0, psi[t+1][x[0]])

    return x
```

Б.4 Алгоритм розв'язку задачі локалізації

```
def define_distance(
        A: npt.NDArray[np.float64],
        B: npt.NDArray[np.float64],
        type: typing.Literal["square", "weighted Jaccard"]
    ):
    """
    Return a distance between set A and set B
    """

    if type == "square":
        return sum([pow(a-b,2) for a,b in zip(A,B)])
    if type == "weighted Jaccard":
        return 1 - sum([min(a,b) for a,b in zip(A,B)])/sum([max(a,b) for a,b in zip(A,B)])

def estimate_implicit_indices(
        x_real: list[str],
        x_predicted: list[int],
        real_implicit_indices: list[int],
        estimate_length: typing.Literal["maximum", " onsistent"],
        T: int,
        N: int,
        metrics_type: typing.Literal["square", "weighted Jaccard"]
    ) -> tuple[int]:
    """
    Return estimation of implicit indices in a hidden chain

    Parameters
    ----------
    x_real : string array(T,)
        Real chain of the hidden states
    x_predicted : int array(T,)
        Decoded enumarated state chain
    real_implicit_indices : int array
        Real set of implicit indices
    estimate_length : string array()
        A way to estimate length of set of implicit indices:
        eigher by ["maximum"] method or by using  ["onsistent", p*] estimation
```

```
    T : int
        Length of a chain
    N : int
        Dimention of any state vector
    metrics_type : string
        Either "square" or "weighted Jaccard"

    Returns
    -------
    predicted_implicit_indices : int array(T,)
        Set of predicted_implicit_indices
        (considering an acceptable list of "0", "1", "2" etc. for given combination)
    """

    phi_real = np.zeros(T, dtype=int)
    for t in range(T):
        phi_real[t] = sum([int(list(x_real[t])[i]) for i in real_implicit_indices])

    if estimate_length[0] == "maximum":
        estimated_length = max(phi_real)
    elif estimate_length[0] == "consistent":
        p = estimate_length[1]
        estimated_length = int(
            (N/(1-p))*(1 - sum([1 for t in range(T-1) if phi_real[t] == phi_real[t+1]])/(T-1))
        )

    offered_implicit_indices = list(itertools.combinations([i for i in range(N)], estimated_length))

    metric = np.zeros(len(offered_implicit_indices))
    for k in range(len(offered_implicit_indices)):
        phi_offered = np.zeros(T, dtype=int)
        for t in range(T):
            phi_offered[t] = sum([int(list(x_predicted[t])[i]) for i in offered_implicit_indices[k]])

        metric[k] = define_distance(phi_real,phi_offered,metrics_type)

    min_metric_value = min(metric)
    argmin_metric_value = [index for index in range(len(metric)) if metric[index] == min_metric_value]

    predicted_implicit_indices = []
    for index in argmin_metric_value:
        predicted_implicit_indices.append(offered_implicit_indices[index])

    return predicted_implicit_indices[0]
```