КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 113 Прикладна математика

на тему:

ДОСЛІДЖЕННЯ АЛГОРИТМІВ НА БАЗІ ВАРІАЦІЙНИХ НЕРІВНОСТЕЙ ДЛЯ ЗАДАЧ МЕРЕЖЕВОЇ ЕКОНОМІКИ

Студентки 4-го курсу
кафедри обчислювальної математики
Петришак Софії Ігорівни

_____

Науковий керівник:
асистент
Денисов Сергій Вікторович

_____

«____» _____ 2021 р.

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики
«____» _____ 2021 р., протокол № ___
Завідувач кафедри Ляшко С. І.

_____

Київ – 2021

Зміст

Вступ

Варіаційні нерівності часто виникають в різноманітних задачах оптимізації і мають численні застосування в математичній економіці, математичному моделюванні транспортних потоків, теорії ігор та інших розділах математики. На сьогоднішній момент запропонована велика кількість методів, а саме методів проекційного типу(ті, що використовують операцію математичного проектування на допустиму множину). Відомо, що у багатьох ситуаціях (наприклад, пошук сідлової точки або рівноваги Неша) для збіжності найбільш простого проекційного метода (аналога метода проекції градієнта) необхідне виконання посилених умов монотонності. Для подолання цього недоліку існує кілька підходів. Один з них полягає в регуляризації початкової задачі з метою надати їй потрібну властивість. Збіжність без модифікації задачі забезпечується в ітераційних методах екстраградієнтного типу, які були вперше запропоновані Г. М. Корпелевич[5]. В даній роботі цей алгоритм, а також Попова[4] та модифікований екстраградієнтний[2,6] будуть застосовані до тестової задачі та економічної задачі міграції[1].

Методи розроблення: комп’ютерне моделювання, розробка програмного продукту.

Інструменти розроблення: тестовий редактор PyCharm, мова програмування Python.

Результати роботи: розроблено програмний засіб розв’язування варіційної
нерівності за допомогою алгоритмів Корпелевич, Попова та модифікованого екстраградієнтного методу, виконано тестування програмного засобу на модельних задачах, проведена оцінка і аналіз результатів.

Постановка задачі:

1. Реалізувати алгоритми оптимізації Корпелевич, Попова та модифікований екстраградієнтний
2. Застосувати їх до тестової задачі та економічної задачі міграції.
3. Порівняти отримані результати

Базові поняття

Означення 1. Оператор P_C метричного проектування простору H на опуклу замкнену множину $C \subseteq$ H, який ставить у відповідність елементу $x \in$ H єдиний елемент $P_C x \in C$, для якого $\| P_C x - x \| = \min \| y - x \|$.

Всюди далі H - дійсний гільбертовий простір зі скалярним добутком $(\square,\square)$, і породженої нормою $\| \square \|$. Нехай C - непорожня підмножина простору H, A - оператор, який діє в H. Будемо розглядати варіаційну нерівність[4]:

$$\text{знайти } x \in C : (Ax, y - x) , 0 \ \ y \in C . \qquad (1)$$

Множину розв'язків варіаційної нерівності (1) позначимо $VI(A, C)$. Будемо припускати виконання наступних умов: множина $C \subseteq$ H - опукла і замкнена; оператор: A: H → H - монотонний, рівномірно неперервний на обмежених множинах і відображає обмежені множини в обмежені;

$VI(A, C) \neq \varnothing$. Також маємо P_C - оператор метричного проектування на множину C.

Твердження 1: У випадку $C = H$ виконання варіаційної нерівності для точки x рівносильне виконанню рівності $A(x) = 0$.

Доведення. Справді, у випадку $C = H$ точка y пробігає увесь простір H. Тому для довільної фіксованої точки x точка $y - x$ також пробігає увесь простір H. Візьмемо y такий, що $y - x = -A(x)$, тоді

$$\langle A(x), y - x \rangle = \langle A(x), -A(x) \rangle = -\|A(x)\|^2 \leq 0$$

причому рівність можлива лише якщо $A(x) = 0$. Отже, варіаційна нерівність може виконуватися тоді і тільки тоді, коли $A(x) = 0$.

Зв'язок з задачами оптимізації.

Прояснимо зв'язок варіаційної нерівності з задачами оптимізації.

Твердження 2. Для задачі

$$f \to \min_c \cdot$$

у випадку опуклості як f так і C критерієм того, що точка x є розв'язком є виконання нерівності

$$\langle f'(x), y - x \rangle \geq 0, \quad \forall y \in C.$$

Доведення. Запишемо лінійну апроксимацію f:

$$f(y) = f(x) + \langle f'(x), y - x \rangle + o(\|y - x\|).$$

Припустимо тепер, що другий доданок менше нуля для якогось $y = x + z$, тоді

$$f(x + z) - f(x) = \langle f'(x), z \rangle + o(\|z\|).$$

Розглянемо (з опуклості C випливає, що $x+\varepsilon z \in C$, а отже можемо підставляти таке y) тепер $y = x + \varepsilon z$, отримаємо

$$f(x + \varepsilon z) - f(x) = \langle f'(x), \varepsilon z \rangle + o(\|\varepsilon z\|).$$

З визначення $o(\cdot)$ зрозуміло, що при $\varepsilon \to +0$ знак правої частини визначає перший доданок.

Тобто, права частина буде від'ємною для якогось достатньо малого ε. Але тоді від'ємною буде і ліва частина, $f(x + \varepsilon z) - f(x) < 0$. Але це означає, що $f(x + \varepsilon z) < f(x)$. Отже, x не є точкою мінімуму f на C. Отримане протиріччя завершує доведення.

Зауваження. У випадку відсутності опуклості або f або C або і того, і того, цей критерій перетворюється на необхідну умову.

Алгоритми

Метод Корпелевич

Розглянемо екстраградієнтний алгоритм Корпелевич для ліпшицевого оператора A[5].

***Алгоритм* (1)**

1) Задаємо $x_0 \in C$, $\lambda \in \left(0, \frac{1}{L}\right)$
2) Для x_n обчислюємо $y_n = P_C(x_n - \lambda A x_n)$.
3) Якщо $x_n = y_n$, то зупиняємо алгоритм, інакше обчислюємо

$$x_{n+1} = P_C(x_n - \lambda A y_n),$$

задаємо $n = n + 1$ та переходимо на крок 2.

Лема 1. Якщо $x_n = y_n$, то $x_n \in VI(A, C)$.

Лема 2. Для $z \in VI(A, C)$ і породжених алгоритмом (1) послідовностей (x_n), (y_n) виконується нерівність

$$\| x_{n+1} - z \|^2 \leq \| x_n - z \|^2 - (1 - \lambda^2 L^2) \| x_n - y_n \|^2$$

Теорема 1. Породжені алгоритмом (1) послідовності (x_n), (y_n) слабко збігаються до розв'язку (1).

Метод Попова

Певною альтернативою методу Г. М. Корпелевич є метод Л. Д. Попова, опублікований у 1980р.

***Алгоритм* (2)**

1) Задаємо $x_0 \in C$, $\lambda \in \left(0, \frac{1}{3L}\right)$
2) Для x_n і y_n обчислюємо

$$x_{n+1} = P_C(x_n - \lambda A y_n),$$

$$y_{n+1} = P_C(x_{n+1} - \lambda A y_n),$$

3) Якщо $x_n = y_n = x_{n+1}$, то зупиняємо алгоритм, інакше задаємо $n := n + 1$ та переходимо на крок 2.

Лема 22. Для $z \in VI(A, C)$ і породжених алгоритмом (2) послідовностей (x_n), (y_n) виконується нерівність

$$\| x_{n+1} - z \|^2 \le \| x_n - z \|^2 - (1 - \lambda L) \| x_n - y_n \|^2 - (1 - 2\lambda L) \| x_{n+1} - y_n \|^2 + \lambda L \| x_n - y_{n-1} \|^2$$

Теорема 2. Породжені алгоритмом (2) послідовності (x_n), (y_n) слабко збігаються до розв'язку (1).

$$\bar{u}^k P_C(x_n - \lambda A y_n)$$

Важливою відмінністю та перевагою методу є те, що на ітерації виконується лише одне обчислення оператора.

Модифікований екстраградієнтний метод

У попередніх розділах було запропоновано екстраградієнтні методи для визначення сідлових точок гладких опукло-увігнутих функцій при наявності обмежень для розв'язання варіаційних нерівностей. ЕГМ володіє важливою перевагою, бо він збігається до сідлової точки опукло-ввігнутої функції (на відміну від градієнтного методу) без вимоги сильної опуклості-вгнутості цих функцій і властивості стійкості. Однією з проблем цих методів є те, що для визначення кроку необхідно знати або вміти оцінювати константи Ліпшіця відповідних функціоналів - а це саме по собі є складною задачею. Тому активно досліджуються так звані адаптивні варіанти алгоритмів, які дозволяють обирати крок без апріорної інформації про константу Ліпшиця.

До таких алгоритмів належить і модифікація екстраградієнтного алгоритму для задачі (1), наведена нижче [6]:

***Алгоритм* (3)**

Задаємо $x_0 \in C$, далі обчислюємо $y_n = P_C(x_n - \lambda A x_n)$, $x_{n+1} = P_C(x_n - \lambda A y_n)$

Величина кроку λ_k визначається з умови

$$0 < \lambda_k \leq \{\bar{\lambda}, \varepsilon \frac{\|x_k - y_k\|}{\|A(x_k) - A(y_k)\|}\}, \qquad (2)$$

де $\bar{\lambda}$ – максимальне значення кроку, що є константною величиною і вибирається з врахуванням конкретно задачі, яку вирішують, ε – константа ($0 < \varepsilon < 1$). Для вибору величини кроку можна використовувати наступну процедуру. Нехай вже відома точка x_k. При $k = 0$ вибираємо $x_0 \in C$.

Перший етап. Визначається вектор $A(x_k)$ і точка y:

$$y = P_C(x_n - \lambda(Ax_n))$$

при чому $\lambda = \bar{\lambda}$.

Другий етап. Обчислюємо вектор $A(y)$. Якщо $A(y) = 0$, то $y \in VI(A, C)$. В іншому випадку робимо перевірку виконання нерівності

$$\lambda \leq \varepsilon \frac{\|x_k - y\|}{\|A(x_k) - A(y)\|} \qquad (3)$$

де $\varepsilon \approx 0{,}8$ - $0{,}9$.

Третій етап. Якщо ця нерівність виконується при $\lambda = \bar{\lambda}$, то $\overline{\lambda_k}$ ставиться рівним $\bar{\lambda}$. Якщо нерівність (3) при цьому значенні λ не виконується, то відбувається дроблення λ. Дроблення λ припиняється як тільки нерівність (3) стає правдивою.

Така процедура завжди може застосуватись, оскільки, в відповідності з нерівністю (2), для всіх k існує відмінне від нуля значення λ_k, при якому

нерівність (3) має виконуватись. Величину $\bar{\lambda}$ варто вибирати таким чином, щоб заздалегідь виконувалась умова $\bar{\lambda} > \varepsilon/L_0$. В частковому випадку коли множина розв'язків *VI*(*A*, *C*) нерівності (1) є екстраградієнтного методу(**Алгоритм 3**) збігається до проекції початкової точки x_0 на множину *VI*(*A*, *C*).

Розглянемо умови, при яких ЕГМ збігається до розв'язку нерівності (1).

Теорема 3. Нехай множина розв'язків *VI*(*A*, *C*) нерівності (1) не пусте, Q – замкнута випукла множина, $A(x)$ – неперервний монотонний оператор на *C*. Тоді ЕГМ (**Алгоритм 3**) збігається до розв'язку нерівності (1) з довільної початкової точки $x_0 \in C$, тобто

$$\lim_{k\to\infty} \min_{x^*} \rho(x^*, x_k) = 0\ , \quad x^* \in VI(A, C),$$

Де $\rho(x^*, x_k)$ – евклідова відстань між точками x^* і x_k.

Задачі

Тестова задача

Формулювання. Нехай множиною є ввесь простір: $C \in \mathbb{R}^m$, а $F(x) = Ax$, де A – квадратна $m \times m$ матриця, елементи якої визначаються наступним чином:

$$a_{i,j} = \begin{cases} -1, & j = m - 1 - i > i \\ 1, & j = m - 1 - i < i \\ 0, & \text{інакше} \end{cases}$$

Тут і надалі нуметрація рядків/стовпчиків матриць, а також елементів масивів починається з нуля.

Це визначає матрицю, чия бічна діагональ складається з половини одиниць і половини мінус одиниць, а решта елементів якої нульові. Для наглядності наведемо кілька перших матриць, для m = 2, 4:

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Для парних m нульовий вектор є розв’язком відповідної варіаційної нерівності. Для усіх чисельних експериментів ми брали $x_0 = (1, \ldots, 1)$, $\varepsilon = 10^{-3}$, $\lambda = 0.4$.

Міграційна рівновага(беззатратна міграція)

У цьому розділі описана модель міграції людей, яка зображує просту абстрактну структуру мережі, де ребра відповідають місцинам, а потоки через ребра (ваги ребер) відповідають населенню відповідного типу в цій місцині[1].

Припустимо, що в нас закрита економіка, в якій, як правило, є n локацій, що позначаються i, і J класів, що як правило, позначаються як k. Припустимо далі, що представлена привабливість будь-якого місця i, яке сприймається класом k корисністю u_i^k. Нехай $\bar{p}^k$ позначає стале та відоме населення класу k в економіці, і нехай p_i^k позначає населення класу k в місці i. Групуємо корисності у вектор-рядок $u \in R^{Jn}$ та населення у вектор-стовпчик $p \in R^{Jn}$. Будемо вважати, що в економіці немає народжень та смертей.

Рівняння збереження потоку для класу k задається як

$$\bar{p}^k = \sum_{i=1}^{n} p_i^k \qquad (4)$$

де $p_i^k \geq 0, \forall k = 1, \dots, J$; $i = 1, \dots, n$. Рівняння (2) сконстатує, що населення кожного класу k має зберігатись в економіці.

Нехай $K \equiv \{p \mid p \geq 0$ і задовільняє $(2)\}$

Припускаємо, що мігранти раціональні і міграція продовжиться до тих пір, поки жодна особа будь-якого класу не матиме стимулу переїхати оскільки одностороннє рішення більше не призведе до збільшення корисності. Математично, отже, мультикласний вектор населення $p \in K$ кажуть, що знаходиться в рівновазі, якщо для кожного класу k; $k = 1, \dots, J$:

$$u_i^k \begin{cases} = \lambda^k, \text{якщо } p_i^{k^*} > 0 \\ \leq \lambda^k, \text{якщо } p_i^{k^*} = 0 \end{cases} \qquad (5)$$

Рівноважні умови (3) свідчать, що для заданого класу *k* ті локації *i* з максимальною корисністю, що рівні індикатору λ^k будуть мати позитивні характеристики класу. А також корисності заданого класу є рівномірно розподілені по класах.

Подивимось на структуру функції. Будемо вважати, в загальному, що корисність асоціюється з конкретною локацією така, що отримується через конкретний клас, можуть залежати від населення, що асоціюється з кожним класом і кожною місцевістю:

$$u = u(p)$$

Зверніть увагу, що дозволяючи корисності залежати від популяцій класів, ми, по суті, використовуємо популяції як проксі для зручностей, що пов'язані з певною локацією; водночас така функція корисності враховує негативні зовнішні ефекти, пов'язаними з перенаселенням, такі як затори, збільшення злочинності, конкуренція за дефіцитні ресурси, тощо.

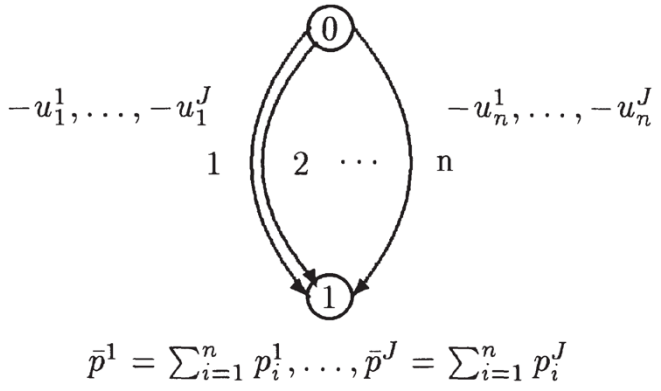

Рис. 1: Формулювання мережевої рівноваги міграції без цін.

Вищевказана модель міграції еквівалентна моделі мережевої рівноваги з єдиною парою відправлення / призначення та фіксованими вимогами. Справді, зробіть ідентифікацію наступним чином. Побудуйте мережу, що складається з двох вузлів, початковий вузол 0 і кінцевий вузол 1 та n посилань, що з'єднують початковий вузол з вузлом призначення (див. рис.1).

Врахуємо, що кожне ребро *i*, *J* коштує: $-u_i^1$, ..., $-u_i^J$ і потоки ребер представлені p_i^1, ..., p_i^J.

Теорема 4 (формулювання варіаційної нерівності для задачі беззатратної міграції)[1]

Структура популяції $p^* \in K$ *знаходиться в рівновазі тоді і лише тоді, коли вона задовольняє задачу варіаційної нерівності:*

$$\langle u(p^*), p - p^* \rangle \geq 0, p \in K \qquad (6)$$

Тоді існування рівноваги випливає із стандартної теорії, оскільки досяжна множина *K* компактна, вважаючи, що функції корисності безперервні. Унікальність рівноважної структури населення також випливає із стандартної теорії за умови, що функція $-u$ суворо монотонна. У контексті застосувань, ця умова монотонності означає, що корисність, яка пов'язана з конкретним класом і розташуванням, буде зменшувати населення цього класу в цій локації; отже, щоб гарантувати унікальність, "перенаселення" системи є критично важливими.

Зауважимо, що рівноважний мережевий еквівалент вищезазначеної моделі будується на абстрактній мережі, оскільки вузли не відповідають місцям у просторі. Навпаки, ребра визначаються як з локації в просторі.

У ролі функції корисності була взята функція

$$u_i^k(p) = -\alpha_{ii}^{kk}\left(p_i^k\right)^2 - \sum_{j,l} \alpha_{ij}^{kl}\, p_j^l + b_i^k,$$

де $\alpha_{ii}^{kk} \in [1,10] \times 10^{-6}$, $-\alpha_{ij}^{kl} \in [1,10], b_i^k \in [10,100]$ генеруються незалежно рівномірно для усіх *i*, *j*, *k*, *l*.

Популяції p_i^k генерувалися незалежно рівномірно з $[10,30]$ для усіх *i*, *k*.

Допустима множина цієї задачі — так званий probability symplex (з точністю до константи *m*). Для проектування $\vec{y}$ на нього ми використовували наступний явний алгоритм:

Перший етап: Відсортувати елементи $\vec{y}$ і зберегти в $\vec{u}$: $u_1 \geq \cdots \geq u_m$.

Другий етап: Знайти $k = \max j$: $u_j + \frac{1}{j}(m - \sum_{i=1}^{j} u_i) > 0$

Третій етап: Видалити вектор з елементами $x_i = \max\{y_i + \lambda, 0\}$, $\lambda = \frac{1}{k}(m - \sum_{i=1}^{k} u_i)$.

Реалізація

Наведемо реалізацію усіх згаданих алгоритмів на мові програмування python.

Варто зауважити, що ми обрали реалізацію згідно якої власне алгоритм знає мінімальний контекст задачі. Тобто, щоб для використовувати алгоритм користувач повинен визначити дві функції. Перша, яка відповідає за обчислення оператора A, а друга — за обчислення оператора P_C. Таким чином ми досягаємо гнучкості у виборі способу обчислення операторів.

Загальний вигляд (за модулем назви і деяких параметрів) запуску

```
solution, iteration_n, duration = korpelevich(
    x_initial=np.ones(size), lambda_=0.4,
    operator=lambda x: a.dot(x), projector=lambda x: x,
    tolerance=1e-3, max_iterations=1e4)
```

алгоритму наступний:

Як бачимо, вже користувач визначає спосіб обчислення операторів A і P_C. Доволі часто це досить просто, хоча у деяких ситуаціях користувачеві доведеться написати більше коду і знадобитсья користуватися scipy.optimize або аналогічним модулем для обчислення проекції, бо ця задача також буває досить важкою в реалізації. Для економічної задачі міграції алгоритм побудови проекції взято з [7].

Ось, наприклад, клієнтський код для задачі міграції:

```
def ProjectionOntoProbabilitySymplex(x: np.array) -> np.array:
    """ computes projection onto a probability symplex """
    n = x.shape[0]
    sorted_x = np.flip(np.sort(x))
```

```
    prefix_sum = np.cumsum(sorted_x)
    to_compare = sorted_x + (1 - prefix_sum) / np.arange(1, n + 1)
    k = 0
    for j in range(1, n):
        if to_compare[j] > 0:
            k = j
    return np.maximum(np.zeros(n), x + (to_compare[k] - sorted_x[k]))
```

```
def proj(p: np.array, p_bar: np.array) -> np.array:
    """ projection function for migration problem """
    return np.array([
        p_bar[k] * ProjectionOntoProbabilitySymplex(p[k::J] / p_bar[k]) for k in range(J)
    ]).T.ravel()
```

Результати

Тестування відбувалося на машині із процесором Intel Core i7 1.99GHz з операційною системою macOS Catalina.

Тестова задача

m	Корпелевич		Попова		Модифікований екстраград.	
	час	ітер.	час	ітер.	час	ітер.
1000	0.10	131	0.05	90	0.26	132
2000	0.58	137	0.36	92	1.74	137
5000	4.04	144	2.85	96	11.55	144
10000	17.40	148	12.55	99	49.20	148

Бачимо, що найкращі результати і по часу, і по кількості ітерацій показує алгоритм Попова. Не сильно відстає від нього алгоритм Корпелевич, а модифікований екстраградієнтний має найнижчі показники.

Задача беззатратної міграції

n	J	Корпелевич		Попова		Модифікований екстраград.	
		час	ітер.	час	ітер.	час	ітер.
20	10	0	3	0	3	0	3
	20	0.1	4	0.1	5	0.1	4
	30	0.1	4	0.1	4	0.3	7
30	10	0	3	0	3	0.1	3
	20	0.1	3	0.1	4	0.2	5
	30	0.2	6	0.3	8	0.8	17
50	10	0	3	0	3	0	2
	20	0.1	4	0.2	5	0.12	20
	30	0.11	15	0.10	12	0.5	11
100	10	0.1	3	0.1	4	0.1	3
	20	0.05	5	0.07	7	0.64	28
	30	0.26	12	0.42	19	1.74	34

Бачимо, що при будь-яких n і малому J результати майже не відрізняються. Зі збільшенням J спочатку найкращі результати отримуємо від модифікованого екстраградієнтного методу, хоча при фінальному $n = 100$ якраз він показує себе найгірше серед інших алгоритмів. Якщо порівнювати Корпелевич та Попова, то в більшості виграє перший як по часу роботи, так і по ітераціях, хоча теж за деякими вийнятками.

Варто зауважити, що значення для тестування задачі беззатратної міграції ми брали набагато менші ніж в тестовій задачі, оскільки задача протребує $O(n^2J^2)$ пам'яті.

Висновок

В рамках цієї дипломної роботи побудовано систему для моделювання внутрішньої міграції на базі моделей з [1] та алгоритмів з [2, 4, 5, 6]. А також виконані наступні задачі:

- розглянуто спектр алгоритмів розв’язання варіаційної нерівності: алгоритми Корпелевич, Попова, модифікований екстраградієнтний;
- розроблено програмний засіб розв’язування варіаційної нерівності за допомогою алгоритмів Корпелевич, Попова, модифікованого екстраградієнтного;
- виконано тестування алгоримтмів на тестовій задачі, а також на економічній задачі міграції;
- проведено детальний аналіз швидкодії і ефективності алгоритмів.

Всі алгоритми показали себе з хорошого боку і можуть бути застосовані для розв’язання найактуальніших задач оптимізації, які виникають в сучасному світі.

Література

1. Anna Nagurney Network Economics: a variational inequality approach // University of Massachuseus, Amherst, U.S.A. John F. Smith Memorial Professor

2. J.Y. Bello Cruzy, R. Diaz Millan, Hung M. Phan Conditional extragradient algorithms for variational inequalities — November 17, 2014

3. Efficient projections onto the ℓ1-ball for learning in high dimensions / John Duchi, Shai Shalev-Shwartz, Yoram Singer, Tushar Chandra // ICML '08: Proceedings of the 25th international conference on Machine learning. — 2008. — P. 272–279. —

4. Киндерлерер Д. Введение в вариационные неравенства и их приложения / Д. Киндерлерер, Г. Стампаккья. – М.: Мир, 1983.

5. Корпелевич, Г.М. Экстраградиентный метод для поиска седловой точки и других задач / Корпелевич, Г.М. // Экономика и математические методы. — 1976. — Vol. 12. — P. 747–756.

6. Хоботов Е. Н. О модификации Экстраградиентного метода для решения вариационных неравенств и некоторых задач оптимизации — 1987

7. Bauschke, H. H. Convex Analysis and Monotone Operator Theory in Hilbert Spaces / H. H. Bauschke, P. L. Combettes. — Springer, 2011.

Додаток 1
Корпелевич

```python
def korpelevych(x_initial: T,
                lambda_: float,
                tolerance: float = 1e-6,
                max_iterations: int = 1e4,
                operator: Callable[[T], T] = lambda x: x,
                projector: Callable[[T], T] = lambda x: x,
                **kwargs) -> Tuple[T, int, float]:
    start = time.time()

    # initialization
    iteration_number = 1
    x_current, x_next = x_initial, None
    y_current = None

    while True:
        # step 1
        y_current = projector(x_current - lambda_ * operator(x_current))

        # stopping criterion
        if norm(x_current - y_current) < tolerance or \
            iteration_number == max_iterations:
            end = time.time()
            duration = end - start
            return x_current, iteration_number, duration

        # step 2
        x_next = projector(x_current - lambda_ * operator(y_current))

        # next iteration
        iteration_number += 1
        x_current, x_next = x_next, None
        y_current = None
```

Попов

```
def popov(x_initial: T,
          y_initial: T,
          lambda_: float,
          tolerance: float = 1e-6,
          max_iterations: int = 1e4,
          operator: Callable[[T], T] = lambda x: x,
          projector: Callable[[T], T] = lambda x: x,
          **kwargs) -> Tuple[T, int, float]:
    start = time.time()

    # initialization
    iteration_number = 1
    x_current, x_next = x_initial, None
    y_previous, y_current = y_initial, None

    while True:
        # step 1
        y_current = projector(x_current - lambda_ * operator(y_previous))

        # step 2
        x_next = projector(x_current - lambda_ * operator(y_current))

        # stopping criterion
        if norm(x_current - y_current) < tolerance and \
            norm(x_next - y_current) < tolerance or \
            iteration_number == max_iterations:
            end = time.time()
            duration = end - start
            return x_current, iteration_number, duration

        # next iteration
        iteration_number += 1
        x_current, x_next = x_next, None
        y_previous, y_current = y_current, None
```

Модифікований екстраградієнтний

```
def modified_extragrad(x_initial: T,
                       tau: float,
                       lambda_initial: float,
                       tolerance: float = 1e-6,
                       max_iterations: int = 1e4,
                       operator: Callable[[T], T] = lambda x: x,
                       projector: Callable[[T], T] = lambda x: x,
                       **kwargs) -> Tuple[T, int, float]:
    start = time.time()

    # initialization
    iteration_number = 1
    x_current, x_next = x_initial, None
    y_current = None
    lambda_current, lambda_next = lambda_initial, None

    while True:
        # step 1
        y_current = projector(x_current - lambda_current * operator(x_current))

        # stopping criterion
        if norm(x_current - y_current) < tolerance or \
            iteration_number == max_iterations:
            nd = time.time()
            duration = end - start
            return x_current, iteration_number, duration

        # step 2
        x_next = projector(x_current - lambda_current * operator(y_current))

        # step 3
        if (operator(x_current) - operator(y_current)).dot(x_next - y_current) <= 0:
            lambda_next = lambda_current
        else:
            lambda_next = min(lambda_current, tau / 2 *
                (np.linalg.norm(x_current - y_current) ** 2 +
                np.linalg.norm(x_next - y_current) ** 2) /
                    (operator(x_current) - operator(y_current)).dot(x_next - y_current))
```

```
# next iteration
iteration_number += 1
x_current, x_next = x_next, None
y_current = None
lambda_current, lambda_next = lambda_next, None
```