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:

Рандомізовані блочні алгоритми для варіаційних нерівностей

Виконала студентка 4-го курсу
Кравець Анна Василівна ________________

Науковий керівник:
доктор фіз.-мат. наук, професор
Семенов Володимир Вікторович ________________

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка ________________

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

Київ — 2021

РЕФЕРАТ

Обсяг роботи 44 сторінки, 14 джерел посилання, 12 ілюстрацій, 1 таблиця, 1 додаток.

Ключові слова:

БЛОЧНИЙ АЛГОРИТМ, ВАРІАЦІЙНА НЕРІВНІСТЬ, ЛІПШИЦЕВИЙ ОПЕРАТОР, МОНОТОННИЙ ОПЕРАТОР, ПРОЕКЦІЯ.

Об'єкт дослідження - рандомізовані блочні алгоритми пошуку розв'язку варіаційної нерівності.

Мета роботи - розробка блочних алгоритмів на основі класичних та перевірка ефективності їх роботи.

Результати: було сформульовано, реалізовано та протестовано 3 блочних алгоритми.

Зміст

ВСТУП **4**

1 ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АЛГОРИТМІВ **5**

1.1 Варіційні нерівності 5

1.2 Застосування варіаційих нерівностей 6

1.3 Питання існування розв'язку 7

1.4 Огляд відомих методів 8

1.5 Стохастичні алгоритми 11

2 БЛОЧНІ АЛГОРИТМИ **13**

2.1 Обмеження на допустиму множину 13

2.2 Переваги блочного розщеплення 13

2.3 Досліджені блочні алгоритми 16

2.4 Інші блочні алгоритми 22

3 ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ **25**

3.1 Деталі реалізації 25

3.2 Задача найменших квадратів 25

3.3 Задача пошуку сідлової точки 27

ВИСНОВКИ **32**

ЛІТЕРАТУРА **33**

ДОДАТОК А **35**

ВСТУП

Багато прикладних математичних задач зводяться до знаходження розв'язку варіаційної нерівності. Наприклад, мінімізація диференційовної функції, пошук сідлової точки, пошук рівноваги за Нешем, пошук рівноваги у транспортних мережах. Багато математиків: Корпелевич Г. М., Попов Л. Д., П. Цзен, Ю. В. Маліцький, В.В.Семенов - доклали зусилля для розробки методів розв'язання варіаційної нерівності.

Однак, при зростанні розмірності задачі, одна ітерація методу стає займати все більше часу. Більшість часу однієї ітерації витрачається на обчислення значення оператора в точці та проекції на допустиму множину X. Щоб зекономити час, можна використовувати не точне значення оператора, а деяке випадкове наближення, яке швидше обчислюється.

Крім цього, у цій роботі припускатиметься, що варіаційна нерівність задана на множині, що розкладається у декартів добуток

$$X = X_1 \times X_2 \times ... \times X_b$$

На одній ітерації алгоритму ми будемо будувати наступну точку, оновлюючи координати лише 1 блоку попередньої точки. Таким чином, буде обчислюватися проекція не на всю множину X, а не деяку "меншу"множину $X_k,\ k \in \{1, 2, .., b\}$, що пришвидшить час виконання ітерації. Такі алгоритми називають блочними. Якщо вибір множини X_k відбувається випадковим чином, то алгоритми мають назву блочних рандомізованих.

У даній роботі об'єктом дослідження будуть виступати саме рандомізовані блочні алгоритми. Предметом дослідження є доцільність їх використання для розв'язку варіаційних нерівностей. Метою даної роботи є розробити рандомізований блочний алгоритм на основі класичного та спробувати встановити можливість/ефективність його застосування для отримання розв'язку варіаційної нерівності.

Дану роботу було представлено на міжнародній конференції: Problem Of Decision Making Under Uncertainties (PDMU)-2021.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АЛГОРИТМІВ

1.1 Варіційні нерівності

У даній роботі ми будемо досліджувати алгоритми для пошуку розв'язків варіційної нерівності.

Визначення 1 (Варіаційна нерівність)**.** *Варіаційна нерівність (ВН) це нерівність вигляду*

$$\langle G(x),\ x' - x \rangle \geq 0 \ \ \forall\ x' \in X, \tag{1}$$

де X - підмножина гільбертового простору H, а $G : X \to H$ - деякий оператор.

Надалі нам також знадобляться наступні визначення.

Визначення 2 (Монотонність)**.** *Оператор G є монотонним на множині X, якщо виконується*

$$\langle G(x) - G(y),\ x - y \rangle \geq 0\ \forall\ x, y\ \in X$$

Визначення 3 (Сильна монотонність)**.** *Оператор G є сильно монотонним на множині X, якщо $\exists\ \alpha > 0$, що виконується*

$$\langle G(x) - G(y),\ x - y \rangle \geq \alpha \|x - y\|^2\ \forall\ x, y\ \in X$$

Визначення 4 (Ліпшицевість)**.** *Оператор G є L-ліпшицевим на множині X, якщо виконується*

$$\|G(x) - G(y)\| \leq L\|x - y\|\ \forall\ x, y\ \in X$$

Якщо не сказано інакше, далі будемо вважати, що:

1. X - опукла замкнена множина
2. G - монотонний та ліпшицевий оператор з константою L

3. множина розв'язків (1) X^* - непорожня

Нехай є точка $x \in X$. Якщо це деяке наближення до розв'язку (1), то корисно було б якимось чином оцінити помилку x на задачі (1). Наприклад, помилку можна визначити наступним чином:

$$Err(x) = \sup_{z \in X} \langle G(z),\ x - z \rangle \tag{2}$$

Тоді $Err(x) \geq 0\ \forall\ x \in X$ та $Err(x) = 0$, у випадку, коли x - розв'язок (1) для монотонного оператора G.

1.2 Застосування варіаційих нерівностей

У цій частині наведемо деякі приклади задач, що зводяться до варіаційних нерівностей.

Приклад 1 (Задача опуклої оптимізації). *Нехай потрібно знайти мінімум функції f на X*

$$f(x) \to \min_{x \in X} \tag{3}$$

За умов, що f - опукла неперервно-диференційовна, X - опукла замкнена множина, ця задача рівносильна варіаційній нерівності, адже справедливий критерій

$$x^* \in \arg\min_{x \in X} f(x) \iff \langle \nabla f(x^*),\ x - x^* \rangle \geq 0 \quad \forall\ x \in X \tag{4}$$

Однак постановка задачі (1) більш широка, ніж постановка (3), адже існують монотонні ліпшицеві оператори, які не можна представити у вигляді градієнта деякого функціонала.

Приклад 2 (Задача пошуку сідлової точки). *Нехай потрібно знайти сідлову точку для функції f*

$$\min_{x \in X} \max_{y \in Y} f(x, y) \tag{5}$$

Позначимо $Z^* \subset X \times Y$ - множина розв'язків (5). *За умови, що f неперевно-диференційовна, опукла за x на X та увігнута за y на Y,*

справедливий критерій

$$z^* \in Z^* \iff \langle G(z^*),\ z - z^* \rangle \geq 0 \quad \forall\ z \in X \times Y \tag{6}$$

Тут $G(z) = (\nabla_x f(x,y), -\nabla_y f(x,y))$.

Приклад 3 (Задача пошуку рівноваги Неша)**.** *Нехай є гра між* b *гравцями. Кожен гравець* i *має множину стратегій* X_i *та функцію втрат* f_i*. Ціль кожного гравця - мінімізувати значення своєї функції втрат за довільного вибору стратегій іншими гравцями,* x^{-i}*. Тобто кожен гравець вирішує задачу*

$$\min_{x_i \in X_i} f_i(x^i; x^{-i}) \tag{7}$$

Рівновагою Неша є такий набір стратегій $x^* = (x^{*1}; x^{*2}; ..; x^{*b})$*, за якого жодному гравцю не вигідно відхилятися від обраної стратегії, тобто* $\forall\ i = \overline{1,b}\ x^{*i}$ *- розв'язок* (7)*. За умов, що* f_i *- неперервно-диференційовна та опукла за* x_i*, справедливий критерій*

$$x^* - \text{ рівновага Неша } \iff \langle G(x^*), x - x^* \rangle \geq 0\ \forall\ x \in \prod_{i=1}^{b} X_i,$$

де $G(x) = (\nabla_{x^1} f_1(x), .., \nabla_{x^b} f_b(x))$.

1.3 Питання існування розв'язку

Далі розглянемо достатні умови існування розв'зку (1). Нам знадобиться наступне твердження.

Твердження 1 ([9])**.** *Множина розв'язків варіаційної нерівності* (1) *співпадає з множиною нерухомих точок оператора* $x \mapsto P_X\big(x - \lambda G(x)\big)$*, де* P_X *- оператор метричної проекції на* X*, а* $\lambda > 0$.

Доведення. Дійсно, для опуклої замкненої множини X з гільбертового простору справедливо, що

$$z = P_X x \ \Leftrightarrow\ \langle z - x, y - z \rangle \geq 0\ \forall\ y \in X$$

Тому

$$x = P_X(x - \lambda G(x)) \Leftrightarrow \langle x - (x - \lambda G(x)), y - x\rangle \geq 0 \ \forall \ y \in X$$

Остання нерівність рівносильна $\langle G(x), y - x\rangle \geq 0 \ \forall \ y \in X$, тобто маємо, що x задовольняє (1).

Надалі припускатимемо, що H - скінченновимірний простір.

Теорема 1 ([9] Про існування розв'зку ВН). *Нехай G -неперервний оператор, а X - непорожня, опукла і замкнена множина. Якщо X - обмежена множина, то варіаційна нерівність* (1) *має розв'язок.*

Доведення. Відображення $x \mapsto P_X\big(x - \lambda G(x)\big)$ - є неперервним, і діє з X у X, де X - опуклий компакт. Тоді, за теоремою Брауера [3], воно має нерухому точку. За твердженням 1, ця точка буде розв'язком (1).

У наступній частині розглянемо відомі методи для пошуку розв'язку ВН (1) або рівняння

$$P_X(x - \lambda G(x)) = x \tag{8}$$

1.4 Огляд відомих методів

До розв'язання (8) можна застосувати відомий метод: метод простої ітерації або метод проекції градієнту (GP - gradient projection). Він має вигляд:

Алгоритм GP

Вхід: $x_0 \in X, \lambda \in (0, \frac{2\alpha}{L^2})$

for $k = 0, .., t-1$ **do**

$$x_{k+1} = P_X(x_k - \lambda G(x_k))$$

end for

Вихід: x_t

Тут α - константа сильної монотонності оператора G.

Доволі часто дослідження актуальності деякого методу для пошуку розв'язку (1) починають з наступного. Розглядають послідовність $\|x_k - z\|$, для довільного $z \in X^*$ (або деяку послідовність $\|x_k - z\|^2 + \mu_k^2$) і намагаються показати, що послідовність є монотонно спадною. Таким чином, це дає підставу вважати, що користуючись алгоритмом, ми наближаємося до множини розв'язків X^* (або принаймні не можемо значно віддалитися, якщо μ_k "мала").

Наприклад, для алгоритма GP послідовність $\|x_k - z\|$ монотонно прямуватиме до 0. Перевагою метода є те, що він збігатиметься з лінійною швидкістю. Проте недоліком є те, що на G накладається обмеження: G має бути сильно монотонним.

У 1976 р. Корпелевич [1] запропонувала екстраградієнтний метод (EG), що працює і для випадку просто моннотоного оператора.

Алгоритм EG

Вхід: $x_0 \in X, \lambda \in (0, \frac{1}{L})$

for $k = 0, .., t-1$ **do**

$$y_k = P_X(x_k - \lambda G(x_k))$$
$$x_{k+1} = P_X(x_k - \lambda G(y_k))$$

end for

Вихід: $\bar{x}_t = \frac{1}{t}\sum_{s=1}^{t} x_s$

Недоліком є необхідність на одній ітерації методу обчислювати значення G у двох різних точках, та двічі знаходити проекцію на множину X. На той час також не було оцінки швидкості збіжності методу. Проте також було показано, що послідовність $\|x_k - z\|$ монотонно спадатиме.

У 1980 р. Попов [2] запропонував метод, що працює для моннотоного оператора G та вимагає лише одного обчислення G на ітерації - що є його перевагою.

Алгоритм PEG
Вхід: $x_0, y_{-1} \in X, \lambda \in (0, \frac{1}{2L})$
for $k = 0, .., t-1$ **do**

$$y_k = P_X(x_k - \lambda G(y_{k-1}))$$
$$x_{k+1} = P_X(x_k - \lambda G(y_k))$$

end for
Вихід: $\bar{x}_t = \frac{1}{t}\sum_{s=1}^{t} x_s$

Цей метод називають екстраполяційним або past extra gradient (PEG), оскільки він використовує значення G, обчислені на попередній ітерації. Для нього було показано, що послідовність $\|x_k - z\|^2 + \lambda L\|x_k - y_{k-1}\|^2$ монотонно спадна.

Інший алгоритм, у якому одне обчислення G на одну ітерацію, був запропонований Ю. Маліцьким у статті [10] - Reflected Gradient (RG).

Алгоритм RG
Вхід: $x_{-1}, x_0 \in X, \lambda \in (0, \frac{\sqrt{2}-1}{L})$
for $k = 0, .., t-1$ **do**

$$x_{k+1} = P_X(x_k - \lambda G(2x_k - x_{k-1}))$$

end for
Вихід: $\bar{x}_t = \frac{1}{t}\sum_{s=1}^{t}(2x_s - x_{s-1})$

Для нього було показано, що послідовність $\|x_k - z\|^2 + \lambda L\|x_k - y_{k-1}\|^2$ монотонно спадна.

Також, існує Optimistic Gradient (OG). Варіант цього алгоритму (без проекції на допустиму множину) був розглянутий К. Даскалакісом та іншими у статті [11]. Інша можлива назва цього алгоритму - Forward-Reflected-Backward (FRB). Його застосування для розв'язку операторних включень з максимально монотонним оператором детально розглянуто у статті [14].

Алгоритм OG
Вхід: $x_{-1}, x_0 \in X, \lambda \in (0, \frac{1}{2L})$
for $k = 0, .., t-1$ **do**

$$x_{k+1} = P_X(x_k - 2\lambda G(x_k) + \lambda G(x_{k-1}))$$

end for
Вихід: $\bar{x}_t = \frac{1}{t}\sum_{s=1}^{t} x_s$

Для нього було показано, що послідовність $\|x_k - z\|^2 + \lambda^2 L^2 \|y_{k-1} - y_{k-2}\|^2$ монотонно спадна.

Для згаданих методів: EG, PEG, RG, OG - справедлива наступна оцінка швидкості збіжності:

$$Err(\bar{x}_t) = \mathcal{O}(\frac{1}{t}) \tag{9}$$

Доведення даної оцінки є у роботі А. Неміровського [6] - для методу EG, та у роботі Y.-G. Hsieh та ін. [13] - для методів PEG, OG, RG.

1.5 Стохастичні алгоритми

У задачах великої розмірності (наприклад, такі виникають у машинному навчанні, при тренуванні нейронних мереж) 1 ітерація детермінованого методу (методу з точним обчисленням G) займає багато часу.

Для того, аби пришвидшити час на 1 ітерацію, можна обчислювати не точне значення $G(z)$, а реалізацію випадкового вектору $G(z, \xi)$ такого, що $\mathbb{E}_\xi\, G(z, \xi) = G(z)$. Алгоритми, що використовують такий підхід, називаються стохастичними.

Наприклад, у випадку, коли можливе представлення

$$G(z) = \frac{1}{n}\sum_{i=1}^{n} g_i(z),$$

можна розглянути ξ - дискретну випадкову величину, рівномірно розподілену на $[1, n]$, та визначити $G(z, \xi) = g_\xi(z)$.

Оскільки стохастичні алгоритми при різних запусках з однієї і тієї ж початкової точки генеруватимуть різні послідовності, то доречні лише

оцінки у середньому. Для стохастичних варіантів алгоритмів PEG, OG та RG була отримана наступна оцінка [13]:

$$\mathbb{E}[Err(\bar{x}_t)] = \mathcal{O}(\frac{1}{\sqrt{t}}) \tag{10}$$

Якщо додатково G - сильно монотонний, то справедлива оцінка, показана у роботі [13]:

$$\mathbb{E}\left\|x_t - x^*\right\|^2 = \mathcal{O}(\frac{1}{t}) \tag{11}$$

2 БЛОЧНІ АЛГОРИТМИ

2.1 Обмеження на допустиму множину

Цікавим для розгляду є клас задач, для якого можливе представлення допустимої множини у вигляді декартового добутку

$$X = X_1 \times X_2 \times ... \times X_b, \text{ де } X_i \subset \mathbb{R}^{u_i} \tag{12}$$

Для таких задач можна на 1 ітерації обчислювати проекцію лише на одну з множин X_k.

Далі, нехай $X \subset \mathbb{R}^m$, тобто

$$\sum_{i=1}^{b} u_i = m$$

Також будемо позначати

$$(U_1|...|U_b) - \text{ розбиття матриці } E \in \mathbb{R}^{m \times m} \text{ таке, що } U_i \in \mathbb{R}^{m \times u_i} \tag{13}$$

2.2 Переваги блочного розщеплення

Помітимо, що алгоритми, згадані у розділі 1.4, передбачають, що на одній ітерації розв'язується нетривіальна задача - задача проекції на множину X (у деяких алгоритмах навіть двічі). При цьому, чим більше розмірність множини X, тим складніше буде розв'язувати цю задачу.

Виникає питання: чи є алгоритми, які враховують структуру допустимої множини (розділ 2.1), і передбачають на 1 ітерації розв'язання простіших задач, не повного, а меншого, блочного розміру?

Виявляється, що відповідь: так, такі алгоритми існують.

Історично перші блочні алгоритми вивчалися для задач оптимізації. Для прикладу, варто згадати алгоритм, що має назву блочний координатний спуск (Block Coordinate Descent - BCD) або нелійнійний метод Гаусса-Зейделя. Він був розглянутий у книзі Д. П. Берцекаса "Нелінійне програмування"[5].

Нехай маємо задачу оптимізації

$$\min_{x \in X} f(x)$$

Надалі у цій роботі позначатимемо x_k^i – i-ий блок вектора x_k (згенерованого на k-ій ітерації). При цьому для i-ого блоку вектора $G(\cdot)$ будемо використовувати позначення $G_i(\cdot)$.

Алгоритм блочного координатного спуску має вигляд:

Алгоритм BCD

Вхід: $x_0 \in X$

for $k = 0, .., t-1$ **do**

  for $i = 1, .., b$ **do**

$$x_{k+1}^i = \arg\min_{y \in X_i} f(x_{k+1}^1, x_{k+1}^2, ..., x_{k+1}^{i-1}, y, x_k^{i+1}, ..., x_k^b)$$

  end for

end for

Так, на кожній ітерації цільова функція мінімізується по кожному блоку координат x_k^i, що розглядаються в циклічному порядку. Перехід від k до $(k+1)$-ої ітерації здійснюється за b кроків, де на кожному кроці оновлюється лише 1 блок вектора.

Для випадку, коли маємо лише 2 блоки, то алгоритм має назву перемінної мінімізації ("alternating minimization").

Про блочні алгоритми саме для варіаційних нерівностей згадується у книзі [4] Димитрія П. Берцекаса та Джона Н. Ціцікліса "Parallel and Distributed Computation: Numerical Methods"1989 року.

Користуючись лемою про розщеплення, варіаційну неівність на множині (12), можна звести до системи з b варіаційних нерівностей менших розмірностей.

Лема 1 (Лема про розщеплення)**.** *Вектор x^* є розв'язком варіаційної нерівності* (1) *на множині X, що задовольняє* (12)*, тоді й лише тоді, коли виконуєтьcя*

$$\langle G_i(x^*),\ x^i - x^{*i} \rangle \geq 0 \quad \forall\, x^i \in X_i \quad \forall\, i,$$

де, як зазначалось раніше, $G_i(\cdot)$ *−* i*-ий блок вектора* $G(\cdot)$

Нелінійний алгоритм Гаусса-Зейделя для варіаційної нерівності має вигляд:

Алгоритм Nonlinear Gauss-Seidel for VI
Вхід: $x_0 \in X$
for $k = 0, .., t-1$ **do**
  for $i = 1, .., b$ **do**
   знайти x^i_{k+1}, як розв'язок

$$\langle G_i(x^1_{k+1}, .., x^{i-1}_{k+1}, x^i_{k+1}, x^{i+1}_k, ..., x^b_k),\ x^i - x^i_{k+1} \rangle \geq 0 \quad \forall\ x^i \ \in X_i$$

  end for
end for

Можна помітити, що для задачі мінімізації функції f алгоритм блочного координатного спуску та нелінійний алгоритм Гаусса-Зейделя для еквівалентної їй варіаційної нерівності з оператором ∇f, по суті, працюють однаково.

Далі, у розділі 2.3, ми розглянемо блочні алгоритми, що не вимагають на 1 ітерації розв'язувати варіаційні нерівності меншого розміру. Натомість, 1 ітерація буде нагадувати ітерацію класичних методів, розглянутих у розділі 1.4, однак оновлюватиметься лише 1 блок вектора.

Також, варто зауважити, що якщо, крім спеціальної структури допустимої множини X, оператор G теж має блочну структуру, то це дає додатковий виграш. Блочна структура передбачає, що G можна подати як суму деяких операторів, що діють з $\mathbb{R}^{u_k}$ у $\mathbb{R}^m$, для $k \in \overline{1, b}$.

Оскільки для блочних алгоритмів відбувається оновлення лише 1 блоку на ітерації, ми маємо можливість більш ефективного переобчислення $G(x_{k+1})$, з урахуванням попереднього значення $G(x_k)$.

Приклад 4. *Нехай, G задається наступним чином: $G(x) = Ax + v$, де $A \in \mathbb{R}^{m \times m}$, $v \in \mathbb{R}^m$. Маємо*

$$G(x) = \sum_{i=1}^{b} AU_i \cdot x^i + v, \quad x = (x^1, ..., x^b), \quad x^i \in \mathbb{R}^{u_i}$$

Припустимо, що x_k та x_{k+1} відрізняються лише по координатам з блоку i:

$$x_{k+1} = x_k + U_i z_k, \ z_k \in \mathbb{R}^{u_i}$$

Тоді

$$G(x_{k+1}) = G(x_k) + AU_i z_k,$$

де AU_i - матриця розміру $m \times u_i$. Таким чином, обчислення $G(x_{k+1})$ з врахуванням уже раніше обчисленого значення $G(x_k)$ потребуватиме $\mathcal{O}(mu_i)$ операцій. Тоді як у загальному випадку обчислення займало б $\mathcal{O}(m^2)$ операцій.

2.3 Досліджені блочні алгоритми

У цьому розділі ми розглянемо 2 алгоритми, для яких були отримані оцінки швидкості збіжності: Stochastic Block Mirror Descent (SBMD) та (Block Stochastic Mirror Prox) BSMP, - це блочні варіанти алгоритму дзеркального спуску та дзеркально-проксимального алгоритму. Дзеркальний спуск можна розглядати як узагальнення методу проекції градієнта, а дзеркально-проксимальний алгоритм - як узагальнення екстраградієнтного методу Корпелевич. Замість звичайного оператора метричної проекції на допустиму множину, ці алгоритми використовують прокс-оператор.

Далі дамо необхідні для розгляду цих алгоритмів поняття.

Визначення 5 (Дивергенція Брегмана)**.** *Нехай $\omega_i : X_i \to \mathbb{R}$ - сильно випукла з параметром α, диференційовна функція на X_i. Тоді дивергенція Брегмана відносно ω_i визначається наступним чином:*

$$V_i(x, z) = \omega_i(x) - \omega_i(z) - \langle \nabla \omega_i(z), x - z \rangle \quad \forall \ x, z \ \in X_i \qquad (14)$$

Визначення 6 (Прокс-оператор)**.** *Прокс-оператор визначається таким чином*

$$\mathcal{P}_i(x,y,\gamma) = \arg\min_{u\in X_i}\langle y,u\rangle + \frac{1}{\gamma}V_i(u,x)\ \forall\ x,y\in X_i \tag{15}$$

Зауваження 1. *Якщо обрати* $\omega_i = \frac{1}{2}\|x\|^2$*, то тоді прокс-оператор співпадатиме з оператором проектування на множину* X_i*:*

$$\mathcal{P}_i(x,y,\gamma) = P_{X_i}(x-\gamma y)$$

Блочний варіант стохастичного дзеркального спуску (stochastic block mirror descent - SBMD) був розглянутий у 2013 р. C. D. Dang та G. Lan у роботі [8]. Дещо спрощений варіант алгоритму з роботи [8] має наступний вигляд.

Алгоритм SBMD

Вхід: $x_0 \in X$, ймовірності $p_i \in [0,1], i = \overline{1,b}, \sum_{i=1}^{b} p_i = 1$

for $k = 0,..,t-1$ **do**

Отримати реалізацію в. вк. ξ: ξ_k, та в.в. i_k, що

$$Prob\{i_k = i\} = p_i$$

Обчислити

$$x_{k+1}^j = \begin{cases} x_k^j, & j \neq i_k \\ \mathcal{P}_j(x_k^j, G_j(x_k,\xi_k),\lambda_k), & j = i_k \end{cases}$$

end for

Вихід: $\bar{x}_t = \dfrac{1}{\sum_{s=1}^{t}\lambda_s}\sum_{s=1}^{t}\lambda_s x_s$

У своїй роботі C. D. Dang та G. Lan пропонують обирати константний крок $\lambda_k \equiv \lambda$ для випадку монотонного оператора G та крок виду $\lambda_k = \mathcal{O}(\frac{1}{k+1})$ для випадку сильно монотонного оператора (в останньому випадку вихід алгоритму - не просто середнє арифметичне згенерованих точок, а середнє зважене).

Для SBMD була отримана оцінка швидкості збіжності, за умов, що

- G - монотонний, не обов'язково ліпшицевий, та задовольняє нерівності

$$\mathbb{E}[\|G_i(z,\xi)\|^2] \leq M_i^2,\ i = \overline{1,b}$$

- множини X_m - обмежені

Зауваження 2. *Насправді, у роботі [8] автори розглядали не варіаційну нерівність, а задачу мінімізації опуклої/сильно опуклої функції f, що є субдиференційовною. Тобто оператор G визначався таким чином, що $G(x) \in \partial f(x)\ \forall\ x \in X$.*

Одержана C. D. Dang та G. Lan оцінка для задачі мінімізації опуклої функції має вигляд:

$$\mathbb{E}[f(\bar{x}_t) - f^*] = \mathcal{O}(\frac{1}{\sqrt{t}}) \tag{16}$$

Якщо додатково f - сильно опукла, то справедлива оцінка

$$\mathbb{E}[f(\bar{x}_t) - f^*] = \mathcal{O}(\frac{1}{t}) \tag{17}$$

Порівнюючи оцінки (10)-(11) з оцінками (16)-(17), бачимо, що вони однакового порядку за кількістю ітерацій. Таким чином, блочні стохастичні алгоритми можна розглядати як можливу альтернативу стохастичним алгоритмам, принаймні для задачі оптимізації.

Скористаємося ідеями авторів статті та спробуємо проаналізувати поведінку алгоритму SBMD для пошуку розв'зку варіаційної нерівності з оператором G. Позначимо функцію

$$V(x,z) = \sum_{i=1}^{b} p_i^{-1} V_i(x^i, z^i) \tag{18}$$

яка, до речі, теж є дивергенцією Брегмана. Користуючись показаною у [8] $\forall\ x \in X$ нерівністю, запишемо

$$V(x, x_{k+1}) \leq V(x, x_k) - \lambda_k \langle G(x_k), x_k - x\rangle + \lambda_k \delta_k + \frac{1}{2}\lambda_k^2 \bar{\delta}_k, \tag{19}$$

де

$$\delta_k = \langle p_{i_k}^{-1} U_{i_k} G_{i_k}(x_k, \xi_k) - g(x_k), x - x_k \rangle \text{ та } \bar{\delta_k} = p_{i_k}^{-1} \|G_{i_k}(x_k, \xi_k)\|^2 \quad (20)$$

Користуючись монотонністю оператора G та (19), можна записати

$$\lambda_k \langle G(x), x_k - x\rangle \leq \lambda_k \langle G(x_k), x_k - x\rangle \leq$$
$$\leq V(x, x_k) - V(x, x_{k+1}) + \lambda_k \delta_k + \frac{1}{2}\lambda_k^2 \bar{\delta_k} \quad (21)$$

Сумуючи (21) для $k = 1, ..t$, можна отримати

$$\langle G(x), \bar{x_t} - x\rangle \leq (\sum_{k=1}^{t} \lambda_k)^{-1}(V(x, x_1) + \sum_{k=1}^{t}(\lambda_k \delta_k + \frac{1}{2}\lambda_k^2 \bar{\delta_k})) \quad (22)$$

Нехай $\zeta_k = (i_k, \xi_k)$ та $\zeta_{[k]} = (\zeta_1, \zeta_2, ..., \zeta_k)$. Маємо

$$\mathbb{E}_{\zeta_k}[\sup_{x\in X} \langle p_{i_k}^{-1} U_{i_k} G_{i_k}(x_k) - G(x_k), x - x_k\rangle \mid \zeta_{[k-1]}] =$$
$$= \sum_{i=1}^{b} \langle U_i G_i(x_k) - p_i G(x_k), x - x_k\rangle = \langle G(x_k) - G(x_k), x - x_k\rangle = 0$$

Звідси

$$\mathbb{E}[\delta_k | \zeta_{k-1}] = 0 \quad (23)$$

Також

$$\mathbb{E}[\bar{\delta_k} \mid \zeta_{k-1}] = \mathbb{E}[p_{i_k}^{-1} \|G_{i_k}(x_k, \xi_k)\|^2 \mid \zeta_{k-1}] =$$
$$= \sum_{i=1}^{b} p_i p_i^{-1} \|G_i(x_k, \xi_k)\|^2 \leq \sum_{i=1}^{b} M_i^2 \quad (24)$$

Узявши матсподівання у (22) та використовуючи (23)-(24), можна отримати

$$\mathbb{E}\langle G(x), \bar{x}_t - x\rangle \leq (\sum_{k=1}^{t} \lambda_k)^{-1}(V(x, x_1) + \frac{1}{2}\sum_{k=1}^{t} \lambda_k^2 \sum_{i=1}^{b} M_i^2) \quad (25)$$

На жаль, (25) не дає змоги оцінити $\mathbb{E}[Err(\bar{x}_t)]$, адже

$$\mathbb{E}\sup_{x\in X} \langle G(x), \bar{x}_t - x\rangle \geq \sup_{x\in X} \mathbb{E}[\langle G(x), \bar{x}_t - x\rangle].$$

Як слідує із зауваження 1, метод проекції градієнту GP можна розглядати як частковий випадок методу дзеркального спуску (за вибору $\omega_i = \frac{1}{2}\|x\|^2$). У практичній частині нашої роботи ми розглядатимемо лише цей частковий варіант.

Блочний стохастичний дзеркально-проксимальний метод (BSMP - block stochastic mirror prox) був розглянутий у статті [12] F.Yousefian, A. Nedic та U. Shanbhag.

Алгоритм BSMP

Вхід: $x_0 \in X$, ймовірності $p_i \in [0,1], i = \overline{1,b}, \sum_{i=1}^{b} p_i = 1$

for $k = 0, .., t-1$ **do**

Отримати реалізацію в. вк. ξ: $\tilde{\xi}_k$, ξ_k, та в.в. i_k, що

$$Prob\{i_k = i\} = p_i$$

Обчислити

$$y_k^j = \begin{cases} x_k^j, & j \neq i_k \\ \mathcal{P}_j(x_k^j, G_j(x_k, \tilde{\xi}_k), \lambda_k), & j = i_k \end{cases}$$

$$x_{k+1}^j = \begin{cases} x_k^j, & j \neq i_k \\ \mathcal{P}_j(x_k^j, G_j(y_k, \xi_k), \lambda_k), & j = i_k \end{cases}$$

end for

Вихід: x_t для сильно монотонного G та інакше

$$\bar{x}_t = \frac{1}{\sum_{s=1}^{t} \lambda_s^r} \sum_{s=1}^{t} \lambda_s^r x_s, \text{ для деякого } r < 1$$

Для випадку сильно монотонного оператору було запропоновано обирати $\lambda_k = \mathcal{O}(\frac{1}{k}), k \geq 1$. Тоді оцінка швидкості збіжності має вигляд:

$$\mathbb{E}[\|x_t - x^*\|^2] = \mathcal{O}(\frac{1}{t}) \tag{26}$$

Для випадку просто монотонного оператору було запропоновано обирати

$\lambda_k = \mathcal{O}(\frac{1}{\sqrt{k+1}})$. Тоді оцінка швидкості збіжності має вигляд:

$$\mathbb{E}[Err(\bar{x}_t)] = \mathcal{O}(\frac{1}{\sqrt{t}}) \tag{27}$$

Для простоти викладок розглянемо аналіз алгоритму BSMP для детермінованого випадку. Аналіз стохастичного випадку наведений у [12].

Нам знадобляться такі позначення:

- μ_{ω_i} - константа сильної випуклості функції ω_i
- C_i - така константа, що $\|G_i(x)\| \le C_i \; \forall \; x \in X$
- B_i - така константа, що $\|x\| \le B_i \; \forall \; x \in X_i$
- L_i - блочна константа Ліпшицевості оператора G_i, що

  $$\|G_i(x^i; x^{-i}) - G_i(y^i; x^{-i})\| \le L_i \|x^i - y^i\| \; \forall \; x^i, y^i \in X_i, \; \forall \; x^{-i} \in \prod_{j \ne i} X_j$$

Також будемо використовувати позначенням з (18). Можна записати нерівність [12]

$$V(x, x_{k+1}) \le$$
$$\le V(x, x_k) - p_{i_k}^{-1} \lambda_k \langle G_{i_k}(x_k), x_k^{i_k} - x^{i_k} \rangle + p_{i_k}^{-1} \lambda_k^2 (\frac{C_{i_k}^2}{\mu_{\omega_{i_k}}} + 2L_{i_k} B_{i_k} C_{i_k}) \tag{28}$$

Нехай $i_{[k]}$ позначатиме $(i_1, i_2, .., i_k)$. Узявши матсподівання по i_k у (28), матимемо

$$\mathbb{E}_{i_k}[V(x, x_{k+1}) \mid i_{[k-1]}] \le V(x, x_k) - \sum_{i=1}^{b} p_i p_i^{-1} \lambda_k \langle G_i(x_k), x_k^i - x^i \rangle +$$
$$+ \sum_{i=1}^{b} p_i p_i^{-1} \lambda_k^2 (\frac{C_i^2}{\mu_{\omega_i}} + 2L_i B_i C_i) \tag{29}$$

З (29) отримуємо

$$\mathbb{E}[V(x, x_{k+1}) \mid i_{[k-1]}] \le V(x, x_k) - \lambda_k \langle G(x_k, x_k - x \rangle +$$
$$+ \lambda_k^2 \sum_{i=1}^{b} (\frac{C_i^2}{\mu_{\omega_i}} + 2L_i B_i C_i) \tag{30}$$

Нерівність (30) далі можна використовувати для отримання оцінок швидкості збіжності.

Як слідує із зауваження 1, екстраградієнтний метод EG можна розглядати як частковий випадок дзеркально-проксимального методу (за вибору $\omega_i = \frac{1}{2}\|x\|^2$). У практичній частині нашої роботи ми розглядатимемо лише цей частковий варіант. Будемо використовувати назву EG_block стосовно B-SMP, у якому замість прокс-оператора використовується звичайний оператор проектування.

2.4 Інші блочні алгоритми

Запишемо рандомізований блочний варіант методу Попова PEG.

Алгоритм PEG_block

Вхід: $y_{-1}, x_0 \in X$, ймовірності $p_i \in [0,1], i = \overline{1,b}, \sum_{i=1}^{b} p_i = 1$

for $k = 0,..,t-1$ **do**

Отримати реалізацію в.в. i_k, що

$$Prob\{i_k = i\} = p_i$$

Обчислити

$$y_k^j = \begin{cases} x_k^j, & j \neq i_k \\ P_{X_j}(x_k^j - \lambda_k G_j(y_{k-1})), & j = i_k \end{cases}$$

$$x_{k+1}^j = \begin{cases} x_k^j, & j \neq i_k \\ P_{X_j}(x_k^j - \lambda_k G_j(y_k)), & j = i_k \end{cases}$$

end for

Вихід: $\bar{x}_t = \dfrac{1}{\sum_{s=1}^{t} \lambda_s} \sum_{s=1}^{t} \lambda_s x_s$

Як обирати λ_k у даному алгоритмі - питання, яке потребує подальшого дослідження. У практичній частині даної роботи λ_k буде обиратися з $(0, \frac{1}{2L})$, аналогічно до алгоритму PEG.

Запишемо рандомізований блочний варіант RG методу.

Алгоритм RG_block

Вхід: $x_{-1}, x_0 \in X$, ймовірності $p_i \in [0,1], i = \overline{1,b}, \sum_{i=1}^{b} p_i = 1$

for $k = 0,..,t-1$ **do**

Отримати реалізацію в.в. i_k, що

$$Prob\{i_k = i\} = p_i$$

Обчислити

$$x_{k+1}^j = \begin{cases} x_k^j, & j \neq i_k \\ P_{X_j}(x_k^j - \lambda_k G_j(2x_k - x_{k-1})), & j = i_k \end{cases}$$

Вихід: $\bar{x}_t = \frac{1}{\sum_{s=1}^{t} \lambda_s} \sum_{s=1}^{t} \lambda_s x_s$

Як обирати λ_k у даному алгоритмі - відкрите питання, яке потребує подальшого дослідження. У практичній частині даної роботи λ_k буде обиратися з $(0, \frac{\sqrt{2}-1}{L})$, аналогічно до алгоритму RG.

Запишемо рандомізований блочний варіант OG методу.

Алгоритм OG_block

Вхід: $x_{-1}, x_0 \in X$, ймовірності $p_i \in [0,1], i = \overline{1,b}, \sum_{i=1}^{b} p_i = 1$

for $k = 0,..,t-1$ **do**

Отримати реалізацію в.в. i_k, що

$$Prob\{i_k = i\} = p_i$$

Обчислити

$$x_{k+1}^{j} = \begin{cases} x_k^j, & j \neq i_k \\ P_{X_j}(x_k^j - (\lambda_k + \lambda_{k-1})G_j(x_k) + \lambda_{k-1}G_j(x_{k-1})), & j = i_k \end{cases}$$

end for

Вихід: $\bar{x}_t = \dfrac{1}{\sum_{s=1}^{t} \lambda_s} \sum_{s=1}^{t} \lambda_s x_s$

Як обирати λ_k у даному алгоритмі - відкрите питання, яке потребує подальшого дослідження. У практичній частині даної роботи λ_k буде обиратися з $(0, \frac{1}{2L})$, аналогічно до алгоритму OG.

3 ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ

У цьому розділі ми будемо тестувати класичні алгоритми: Extra Gradient (EG), Past Extra Gradient (PEG), Reflected Gradient (RG), Optimistic Gradient (OG), та блочні алгоритми: EG_block, PEG_block, RG_block, OG_block для пошуку розв'язку варіаційної нерівності з сильно монотонним чи просто монотонним оператором.

3.1 Деталі реалізації

Алгоритми були реалізовані на мові програмування Python з використанням бібліотеки numpy. Код основних модулів наведений в додатку А.

Тестування проводилося з використанням процесору Intel Core i5-8250U частоти 1.6 ГГц.

Множини X_i були задані як симплекси (кожен симплекс розмірності не менше 20). Для проектування на них було використано алгоритм швидкого проектування на симплекс зі статті J. Duchi, S. Shalev-Shwartz та Y. Singer T. Chandra [7].

3.2 Задача найменших квадратів

Розглянемо задачу найменших квадратів з регуляризацією

$$f(x) = \frac{1}{2}\|Ax - v\|^2 + \frac{1}{2}\alpha\|x\|^2 \to \inf_{x \in X} \tag{31}$$

яка, за критерієм (4), еквівалентна задачі пошуку розв'язку ВН

$$\langle G(x),\ z - x\rangle \geq 0 \quad \forall\, z \in X,$$

де оператор G визначається як градієнт цільової функції:

$$G(x) = A^T(Ax - v) + \alpha x$$

Для тестування матрицю A та вектор v згенеруємо випадковим чином.

Для тесту 1 кількість блоків, на яку розбивається допустима множина X, була обрана 20, а середній розмір блоку 50. Умова зупинки алгоритмів: пройшло 3 хвилини. Крок λ_t був покладений константний.

Як бачимо, на задачі невеликої розмірності, для 1 тесту блочні та класичні алгоритми працюють порівняно схожим чином.

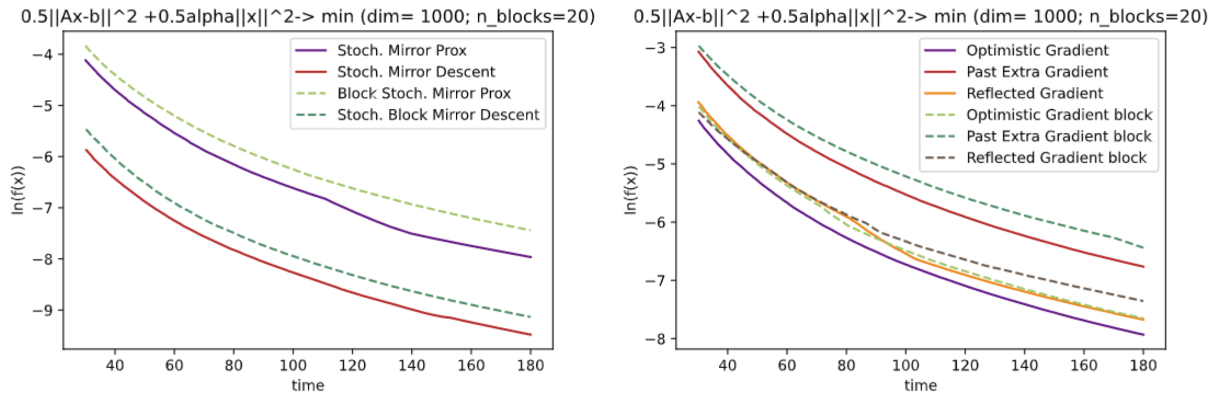

Рис. 1: Робота алгоритмів на задачі НК, множ. X з 20 блоків

Для наступного тесту 2 була задана допустима множина X з 100 блоками, середній розмір блоку 25. Умова зупинки алгоритмів: пройшло 3 хвилини. З графіків можна побачити, що блочні алгоритми дещо програють класичним.

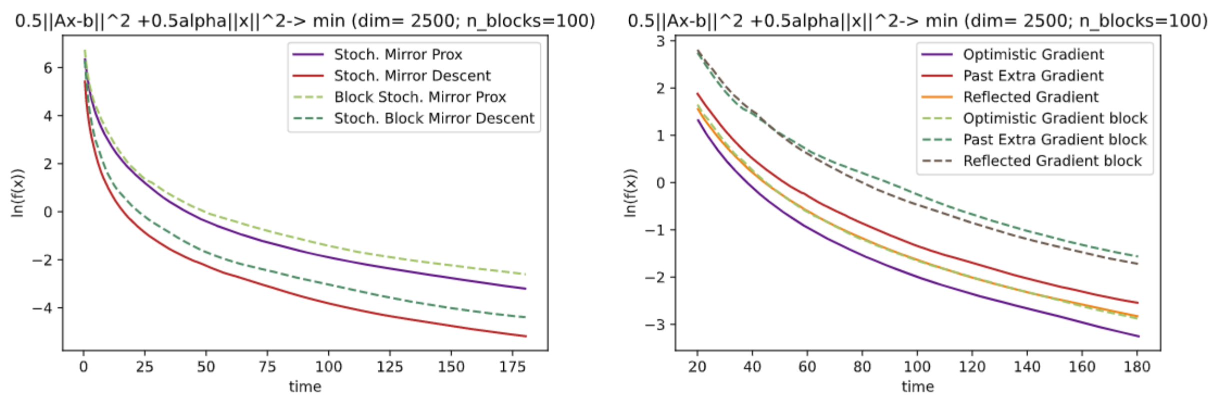

Рис. 2: Робота алгоритмів на задачі НК, множ. X з 100 блоків

Але якщо покласти кроки λ_t для блочних алгоритмів більше, ніж у класичних, то вони будуть працювати, причому працюватимуть швидше. У тесті 3 ми розглянули ту ж саму задачу, що й у тесті 2, проте взяли у 5 рази більші кроки λ_t для блочних методів, порівняно з класичними.

Як видно з графіків на рис. 3, блочні алгоритми працюють швидше, ніж класичні.

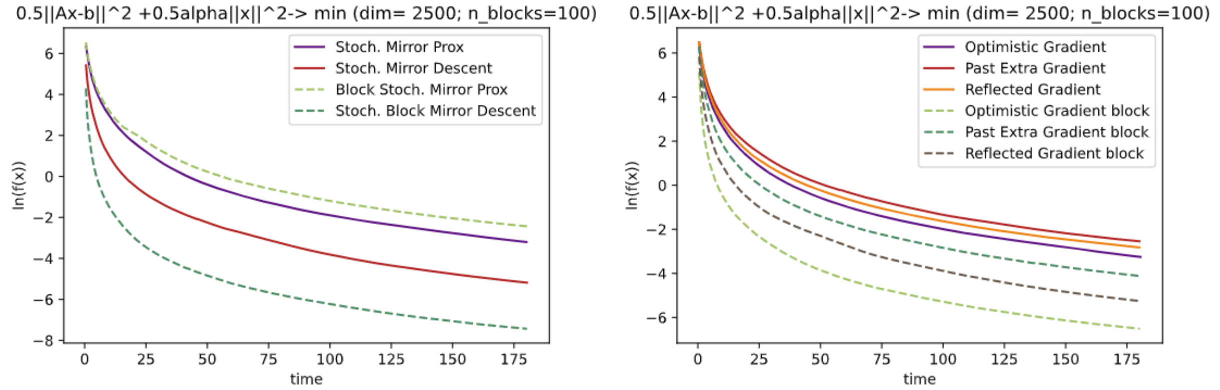

Рис. 3: Робота алгоритмів на задачі НК, множ. X з 100 блоків, різний вибір кроку для блочних та класичних алгоритмів

У тестах 1, 2, 3 оператор G був сильно монотонним. Далі розглянемо задачу, для якої G є монотонним, але не сильно монотонним.

3.3 Задача пошуку сідлової точки

Розглянемо наступну задачу пошуку сідлової точки

$$\min_{x \in X} \max_{y \in Y} f(x, y), \tag{32}$$

де $f(x, y) = \langle Ax,\ y\rangle,\ A \in \mathbb{R}^{m \times m},$ а $X \subset \mathbb{R}^m$ та $Y \subset \mathbb{R}^m$

Тоді f - опукла за x та увігнута за y на множині X та Y відповідно. Тому, можна скориставшись критерієм (6), отримати еквівалентну ВН:

$$\langle G(z),\ z - z'\rangle \geq 0,\ \forall\ z' \in X \times Y,$$
$$\text{де } z = (x, y),\ G(z) = (\nabla_x f(x, y), -\nabla_y f(x, y)).$$

Маємо $G(x, y) = (A^T y, -Ax)$ - ліпшицевий оператор з константою Ліпшиця $\|A\|_2$ та виконується рівність:

$$\langle G(z) - G(z'), z - z'\rangle = 0.$$

Таким чином, G - монотонний але не сильно монотонний.

Нехай

$$X = X_1 \times ... \times X_{b_x},\ \text{де}\ \ X_i \subset \mathbb{R}^{u_i},\ \text{та}$$
$$Y = Y_1 \times ... \times Y_{b_y},\ \text{де}\ Y_i \subset \mathbb{R}^{v_i},$$

і виконується

$$\sum_{i=1}^{b_x} u_i = \sum_{i=1}^{b_y} v_i = m$$

Скористаємося позначеннями $U_1, ..., U_{b_x}$ з (13). Також, позначимо $(V_1|...|V_{b_y})$ - розбиття матриці $E \in \mathbb{R}^{m\times m}$ таке, що матриця V_i містить v_i стовпчиків. Також, нехай симплекс X_i має параметр α_i (сума координат вектора $x_i \in X_i$ - рівна α_i). І нехай симплекс Y_i має параметр β_i (сума координат вектора $y_i \in Y_i$ - рівна β_i).

Помилку деякої допустимої точки $z \in X \times Y$ на задачі (32) прийнято визначати наступним чином:

$$\varepsilon_{sad}(z) = f^*(x) - f_*(y),\ \text{де} \tag{33}$$
$$f^*(x) = \max_{y\in Y} f(x,y),\ f_*(y) = \min_{x\in X} f(x,y) \tag{34}$$

Можна помітити, що

$$\varepsilon_{sad}(z) \geq 0\ \forall\ z \in X \times Y \text{ та } \varepsilon_{sad}(z) = 0 \iff z -\ \text{розв'язок (32)}$$

Функції (34) для задачі (32) можна обчислити наступним чином

$$f^*(x) = \sum_{i=1}^{b_y} \beta_i \|V_i^T A x\|_\infty$$
$$f_*(y) = \sum_{i=1}^{b_x} \alpha_i \min\{\text{координати вектора } U_i^T A^T y\}$$

Для тестування X, Y було обрано як декартові добутки симплексів, кожний з симплексів розмірності не менше 20.

Спершу (для тестів 4 та 5) було покладено $A = E \in \mathbb{R}^{m\times m}$

Для тесту 4 було задано множини X, Y, що містять по 100 блоків кожна. Сумарна розмірність множини $X \times Y$ - 5000. Крок було обрано незмінним $\lambda_k = const$. Як можна побачити з результатів тесту, блочні алгоритми у даному випадку програють класичним. Через 5 хвилин після старту класичні алгоритми знайшли точку, у якій помилка на задачі 32 приблизно $0.001 - 0.003$, тоді як для блочних помилка лежить у межах $0.05 - 0.1$.

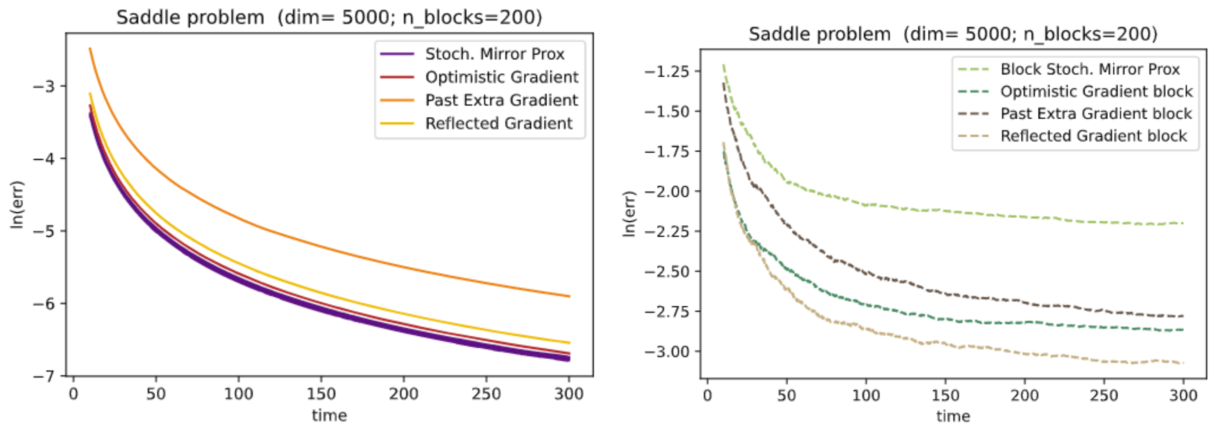

Рис. 4: Робота алгоритмів на задачі (32), $A = E$, розмірність $m = 2500$

Для тесту 5 було задано множини X, Y, що містять по 600 блоків кожна. Середня розмірність одного блоку - 100, сумарна розмірність множини $X \times Y$ - 120000. Крок було обрано незмінним $\lambda_k = const$. Як можна побачити з результатів тесту, блочні алгоритми у даному випадку знову програють класичним. Через 5 хвилин після старту класичні алгоритми знайшли точку, у якій помилка на задачі 32 приблизно $0.1 - 0.3$, тоді як для блочних помилка лежить у межах $0.9 - 1.5$.

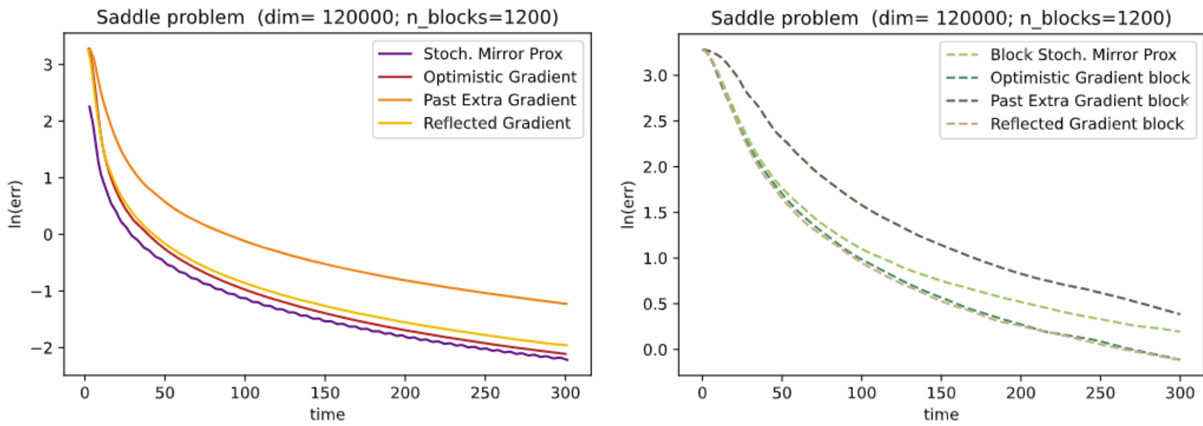

Рис. 5: Робота алгоритмів на задачі (32), $A = E$, розмірність $m = 60000$

Далі, для тесту 6 матрицю A було згенеровано випадковим чином. Множини X, Y містять по 100 блоків кожна. Середня розмірність одного блоку - 25, сумарна розмірність множини $X \times Y$ - 5000. Крок було обрано незмінним $\lambda_k = const$. У цьому випадку знову блочні алгоритми працюють повільніше, ніж класичні.

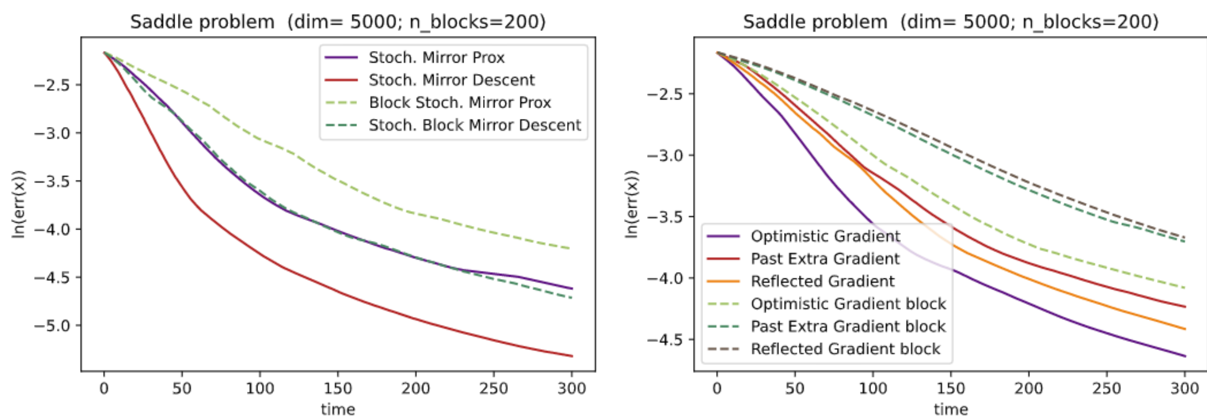

Рис. 6: Робота алгоритмів на задачі (32), розмірність $m = 2500$

Через 5 хв після запуску, алгоритми знайшли точку, у якій помилка (33) на задачі (32) рівна значенню, наведеному в таблиці 1. Для блочних алгоритмів спостерігаємо більші помилки, у порівнянні з класичними.

Алгоритм	Помилка	Алгоритм блочний	Помилка
Past Extra Grad.	0.014	Past Extra Grad. block	0.025
Refl. Grad.	0.012	Refl. Grad. block	0.025
Optimistic Gradient	0.010	Optimistic Gradient block	0.017
Stoch. Mirror Prox	0.010	Block Stoch. Mirror Prox	0.015
Stoch. Mirror Desc.	0.005	Stoch. Block Mirror Desc.	0.009

Табл. 1: Помилки для тесту 6

ВИСНОВКИ

Блочні алгоритми є сенс використовувати, для того, щоб швидко виконувати 1 крок і швидко отримувати наступне наближеня. Втім, вони передбачають обмеження на допустиму множину, а саме: можливість її розкладу у декартів добуток. Якщо є можливість розпаралелювати обчислення, то можна спробувати одночасно виконувати проекції на відповідні множини на різних обчислювальних вузлах. Але якщо такої можливості немає (або можливості паралельного обчислення обмежені), то блочні алгоритми є допустими методом пошуку розв'язку варіаційної нерівності.

У даній роботі було описано блочні алгоритми, для яких відомі оцінки швидкості збіжності. Також були сформовані блочні алгоритми на основі відомих не блочних алгоритмів для розв'язку варіаційних нерівностей. Для них ще не з'ясовані оцінки швидкості збіжності та не досліджено, який спосіб вибору кроку та спосіб усереднення є найбільш оптимальним. У цій роботі ми дослідили роботу алгоритмів на 2 класичних задачах з сильно монотонним та просто монотонним оператором.

Результати тестування дають підставу зробити наступні висновки. По-перше, сформовані у цій роботі блочні варіанти відомих алгоритмів дійсно будують наближення до розв'язку, принаймні, на тестових задачах. Також, для економії часу роботи методу слід використовувати можливість ефективно переобчислювати значення оператора, використовуючи попередню точку, що відрізняється від теперішньої лише по координатам 1 блоку. Цей підхід сприятимеме значному виграшу в часі роботи. Крім того, можна помітити, що для блочних варіантів алгоритмів при виборі константного кроку можна використовувати пом'якшені обмеження, і алгоритми будуть збігатися. Але у яких межах можна обирати цей крок наразі важко сказати.

ЛІТЕРАТУРА

[1] Korpelevich G.M. "The extragradient method for finding saddle points and other problems." в: *Ekonomika i Matematcheskie Metody* 12 (1976), с. 747—756.

[2] Попов Л. Д. "Модификация метода Эрроу-Гурвица поиска седловых точек". в: *Математические заметки.* 28 №5 (1980), с. 777—784.

[3] Стампаккья Г. Киндерлерер Д. *Введение в вариационные неравенства и их приложения.* Москва, 1ый Рыжский переулок, 2: Мир, 1983.

[4] D.P. Bertsekas та J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, 1989.

[5] D.P. Bertsekas. *Nonlinear Programming.* 2-е вид. Belmont, MA: Athena Scientific, 1999.

[6] Arkadi Nemirovski. "Prox-Method with Rate of Convergence O(1/t) for Variational Inequalities with Lipschitz Continuous Monotone Operators and Smooth Convex-Concave Saddle Point Problems". в: *SIAM Journal on Optimization* 15 (2004), с. 229—251. ISSN: 1095-7189. URL: `https://doi.org/10.1137/S1052623403425629`.

[7] Y. Singer T. Chandra J. Duchi S. Shalev-Shwartz. "Efficient Projections onto the l1-Ball for Learning in High Dimensions". в: *ICML 08: Proceedings of the 25th international conference on Machine learning* (лип. 2008), с. 272—279.

[8] Cong D. Dang та Guanghui Lan. *Stochastic Block Mirror Descent Methods for Nonsmooth and Stochastic Optimization.* 2013. arXiv: `1309.2249 [math.OC]`.

[9] А. В. Гасников и др. *Введение в математическое моделирование транспортных потоков: Учебное пособие.* М.: МЦНМО, 2013.

[10] Yu. Malitsky. "Projected Reflected Gradient Methods for Monotone Variational Inequalities". в: *SIAM Journal on Optimization* 25.1 (січ. 2015), с. 502—520. ISSN: 1095-7189. DOI: 10.1137/14097238x. URL: http://dx.doi.org/10.1137/14097238X.

[11] Constantinos Daskalakis та ін. *Training GANs with Optimism*. 2018. arXiv: 1711.00141 [cs.LG].

[12] Farzad Yousefian, Angelia Nedich та Uday V. Shanbhag. *On stochastic mirror-prox algorithms for stochastic Cartesian variational inequalities: randomized block coordinate and optimal averaging schemes*. 2018. arXiv: 1610.08195 [math.OC].

[13] Yu-Guan Hsieh та ін. *On the convergence of single-call stochastic extra-gradient methods*. 2020. arXiv: 1908.08465 [math.OC].

[14] Yura Malitsky та Matthew K. Tam. *A Forward-Backward Splitting Method for Monotone Inclusions Without Cocoercivity*. 2020. arXiv: 1808.04162 [math.OC].

ДОДАТОК А

З модуля solver:

```
"""Class Solver that could be used to find solution of VI.
Helpful classes: Method and its subclasses,  StepsizePolicy and
subclasses, SchemeLast, SchemeAvg, SchemeWeightAvg.
 Available methods (implemented in Method subclasses):
        -Gradient Projected (GP)
        -Optimistic Gradient (OG) (aka Forward-Reflected-Backward)
        -Reflected Gradient (RG)
        -Extra Gradient (EG) (also known as Korpelevich's algorithm)
        -Past extra Gradient (PEG) (also known as Popov's algorithm)
        -block variants of each of the preceding methods
        -Brute (random generating of points and choosing the best)
"""

from .choice import Choice
import constants
import math
import numpy as np
import random
from time import time


class Solver:
    """Defines solver method, solution approximation scheme and
    stepsize policy. Use solve method to run the algorithm and get
    approximate solution of VI problem with constraints. """
    def __init__(self,
                 method_option,
                 scheme_option,
                 stepsize_option,
```

```
                 problem,
                 constraint_set,
                 time_limit=20):
        self.stepsize_policy = Choice('StepsizePolicy').choose(
            stepsize_option, method_option, problem)
        self.scheme = Choice('Scheme').choose(scheme_option)
        self.method = Choice('Method').choose(method_option, problem,
                                              constraint_set)
        self.time_limit = time_limit

    def solve(self):
        """Run iterative method"""
        # In order to have the same results for randomized algos.
        random.seed(constants.r_seed)
        # In order to have the same initial points.
        np.random.seed(constants.r_seed)
        x_prev, x = self.method.generate_rand_points()
        g_x_prev = self.method.problem.G(x_prev)
        time_recorded = time()
        time_elapsed = 0
        time_step = 1 / 4 # record error each time_step seconds
        iteration = 0
        history = {'time': [], 'error': [], 'iter': []}
        while time_elapsed < self.time_limit and not (
                history['error'] and history['error'][-1] < 1e-25):
            iteration += 1
            stepsize = self.stepsize_policy.update(iteration)
            x_prev, x, g_x_prev = x, *self.method.step(x, x_prev,
                                                    g_x_prev, stepsize)
            self.scheme.update(x, stepsize)
            if time() - time_recorded >= time_step:
                time_elapsed += time() - time_recorded
```

```
                self.record(history, time_elapsed, iteration)
                time_recorded = time()
        time_elapsed += time() - time_recorded
        self.record(history, time_elapsed, iteration)
        return self.scheme.get(), history

    def record(self, history, time_elapsed, iteration):
        """Record current progress of algorithm."""
        err = self.method.problem.error(self.scheme.get())
        history['error'].append(err)
        history['time'].append(time_elapsed)
        history['iter'].append(iteration)


class Method:
    """Class that defines the iteration method for solving the VI."""
    def __init__(self, problem, constraint_set):
        self.problem = problem
        self.set = constraint_set

    def step_random_block(self, x, step_term, block_id=None):
        """Update random block of x with projection on corresponding
        set of x-step_term."""
        if block_id is None:
            block_id = random.randrange(0, self.set.n_blocks)
        x_new = np.copy(x)
        start = sum(self.set.dimension_list[:block_id])
        end = start + self.set.dimension_list[block_id]
        x_new[start:end] = self.set.project_on_block(
            x[start:end] - step_term[start:end], block_id)
        return x_new
```

```
    def generate_rand_points(self):
        """Generate 2 points that lie in the constraints set."""
        rand_points = []
        for seed in [1, 277]:
            x = np.random.rand(self.set.n_dim, 1)
            rand_points.append(self.set.project(x))
        return rand_points[:]


class MethodBrute(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """Generate random guess and save it if it is better then
        last guess."""
        x_new = self.set.project(
            np.random.rand(x.shape[0], 1) / np.random.rand())
        if self.problem.error(x_new) > self.problem.error(x):
            x_new = x
        return x_new, g_x_prev


class MethodPEG(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Past-Extra-Gradient algorithm."""
        y = self.set.project(x - stepsize * g_x_prev)
        g_y = self.problem.G(y)
        return self.set.project(x - stepsize * g_y), g_y


class MethodPEG_block(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Past-Extra-Gradient algorithm with update
        performed on just 1 random block."""
```

```
        step_term = stepsize * g_x_prev
        block_id = random.randrange(0, self.set.n_blocks)
        y = self.step_random_block(x, step_term, block_id)
        g_y = self.problem.G(y)
        step_term = stepsize * g_y
        return self.step_random_block(x, step_term, block_id), g_y


class MethodEG(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Extra-Gradient algorithm"""
        g_x = self.problem.G(x)
        y = self.set.project(x - stepsize * g_x)
        g_y = self.problem.G(y)
        return self.set.project(x - stepsize * g_y), g_y


class MethodEG_block(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Extra-Gradient algorithm with update performed
        on just 1 random block."""
        g_x = self.problem.G(x)
        step_term = stepsize * g_x
        block_id = random.randrange(0, self.set.n_blocks)
        y = self.step_random_block(x, step_term, block_id)
        g_y = self.problem.G(y)
        step_term = stepsize * g_y
        return self.step_random_block(x, step_term, block_id), g_y


class MethodGP(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
```

```
        """One step of Gradient-Projection algorithm"""
        g_x = self.problem.G(x)
        return self.set.project(x - stepsize * g_x), g_x


class MethodGP_block(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Gradient-Projection algorithm with update
        performed on just 1 random block."""
        g_x = self.problem.G(x)
        step_term = stepsize * g_x
        return self.step_random_block(x, step_term), g_x


class MethodOG(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Optimistic Gradient algorithm"""
        g_x = self.problem.G(x)
        return self.set.project(x - 2 * stepsize * g_x +
                                stepsize * g_x_prev), g_x


class MethodOG_block(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Optimistic Gradient algorithm with
        update performed on just 1 random block."""
        g_x = self.problem.G(x)
        step_term = 2 * stepsize * g_x - stepsize * g_x_prev
        return self.step_random_block(x, step_term), g_x


class MethodRG(Method):
```

```
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Reflected-Gradient algorithm"""
        g_reflect = self.problem.G(2 * x - x_prev)
        return self.set.project(x - stepsize * g_reflect), g_reflect


class MethodRG_block(Method):
    def step(self, x, x_prev, g_x_prev, stepsize):
        """One step of Reflected-Gradient algorithm with update
        performed on just 1 random block."""
        g_reflect = self.problem.G(2 * x - x_prev, )
        step_term = stepsize * g_reflect
        return self.step_random_block(x, step_term), g_reflect


class StepsizePolicy:
    """Defines policy for updating stepsizes with iterations.
    Depends on method and problem properties."""
    # Defines how many times block stepsizes are greater then classic.
    BLOCK_CONST = 1

    def __init__(self, method_option, problem):
        """Calculate initial stepsize based on chosen algorithm and
        problem's properties"""
        # store constants associated with different methods
        method_const_dict = {
            'GP': 1,
            'GP_block': StepsizePolicy.BLOCK_CONST,
            'OG': 0.5,
            'OG_block': 0.5 * StepsizePolicy.BLOCK_CONST,
            'RG': (math.sqrt(2) - 1),
            'RG_block': (math.sqrt(2) - 1) * StepsizePolicy.BLOCK_CONST,
```

```
            'EG': 1,
            'EG_block': StepsizePolicy.BLOCK_CONST,
            'PEG': 0.5,
            'PEG_block': 0.5 * StepsizePolicy.BLOCK_CONST,
            'Brute': 1
        }
        print(StepsizePolicy.BLOCK_CONST, " = BLOCK_CONST")
        self.method_const = method_const_dict[method_option]
        eps = 1e-15
        self.step_size = (
            self.method_const / problem.calc_Lipschitz_const() - eps)


class StepsizePolicyConst(StepsizePolicy):
    """Stepsize is constant"""
    def update(self, iteration):
        return self.step_size


class StepsizePolicyInv(StepsizePolicy):
    """Stepsize = C/k, where k is number of iter"""
    def __init__(self, method_option, problem):
        """Set initial stepsize"""
        StepsizePolicy.__init__(self, method_option, problem)
        CONST = 1_000 # some big constant
        self.step_size *= CONST

    def update(self, iteration):
        return self.step_size / iteration


class StepsizePolicySqrInv(StepsizePolicy):
```

```
    """Stepsize = C/sqrt(k), where k is number of iter"""
    def __init__(self, method_option, problem):
        """Set initial stepsize"""
        StepsizePolicy.__init__(self, method_option, problem)
        CONST = 1_000
        self.step_size *= CONST

    def update(self, iteration):
        return self.step_size / np.sqrt(iteration)


class SchemeLast:
    """Scheme that builds solution as last seen point"""
    def get(self):
        """Get current approximation to the solution"""
        return self.sol

    def update(self, x, stepsize=None):
        self.sol = x


class SchemeAvg:
    """Scheme that builds solution as average of points generated by
    algorithm"""
    def __init__(self):
        self.sum = None
        self.iter = 0

    def get(self):
        """Get current approximation to the solution"""
        return self.sum / self.iter
```

```
    def update(self, x, stepsize=None):
        self.sum = x if self.sum is None else self.sum + x
        self.iter += 1


class SchemeWeightAvg:
    """Scheme that builds solution as weighted average of points
    generated by algorithm"""
    def __init__(self):
        self.sum = None
        self.weights_sum = 0

    def get(self):
        """Get current approximation to the solution"""
        return self.sum / self.weights_sum

    def update(self, x, stepsize):
        self.sum = x * stepsize if self.sum is None else (self.sum +
                                                                 x * stepsize)
        self.weights_sum += stepsize
```