Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РЕКУРСИВНИЙ SVD-РОЗКЛАД ТРЬОХДІАГОНАЛЬНОЇ МАТРИЦІ НА ГРАФІЧНОМУ ПРОЦЕСОРІ

Текстова частина до дипломної роботи
за спеціальністю „Інженерія програмного забезпечення”

Керівник дипломної роботи
д-р. фіз.-мат. наук.,
проф. Малашонок Г. І.
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2020 р.

Виконав студент
Кулаковський Д. В.
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
к.ф-м.н.
_____ С. С. Гороховський

13 листопада 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на дипломну роботу

студенту 2-го курсу магістерської програми, факультету інформатики__
Кулаковському Дмитру Віталійовичу_____
Розробити програмний модуль для SVD-розкладання трьохдіагональної матриці на графічному процесорі
Вихідні дані:

- програмний модуль;
- Теоретичне обґрунтування;

Зміст ТЧ до магістерської роботи:

Зміст
Анотація
Вступ
1. Історія розвитку SVD-розкладу
2. Теоретичні відомості
3. Python and GPU
4. Програмна реалізація SVD-розкладу
Висновки
Список літератури

Дата видачі „___” _____ 2019 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

ЗМІСТ

Анотація

Метою дипломної роботи була реалізація SVD-розкладу для трьохдіагональної матриці на графічному процесорі.

Задачу вирішено за допомогою:

1. Мови Python, модулі для математичних функцій.
2. Модуля TensorFlow для роботи із архітектурою CUDA та розрахунків на GPU.

В результаті розроблено програму, яка відповідає меті роботи. Проведено тестування додатку та оптимізовано код для пришвидшення розрахунків.

Ключові слова: SVD-розклад, Python, GPU, TensorFlow, NumPy.

Вступ

SVD-розклад – Однією з найбільш плідних ідей в теорії матриць є матричне розкладання або канонічна форма. Останнім часом матричні розкладання стали оплотом численних методів лінійної алгебри, які служать основою вирішення безлічі проблем.

SVD дуже широко вживається в машинному навчанні; фактично, якщо ви хочете щось до чогось наблизити, не виключено, що вам десь по дорозі зустрінеться SVD. Класичний приклад застосування SVD - шумозаглушення, наприклад, в зображеннях. Або, при формуванні рекомендаційних списків для користувача на інтернет-ресурсі.

Виходячи з цього, за мету даної роботи було поставлено розробку алгоритму SVD-розкладу матриці та оптимізацію для пришвидшення роботи до рівня C++ або MATLAB.

В даній дипломній роботі реалізовано програмний модуль для SVD-розкладу на графічному процесорі. Проведено велику кількість тестів, та декілька етапів оптимізації коду для пришвидшення розрахунків.

Робота складається з чотирьох розділів.

Перший розділ присвячено історичній довідці.

В другому розділі проаналізовано принцип роботи SVD-розкладу та етапи його розвитку.

Третій розділ присвячено аналізу програмування математичних методів з допомогою мови Python та способи розрахунку на GPU.

Четвертий розділ – лістинг коду, роз’яснення використаних функцій та модулів, викладки тестів до і після оптимізації алгоритму.

Створено програмний продукт, який призначений для проведення SVD-розкладу матриць великого розміру.

Постановка задачі

1. Виконати аналіз SVD-розкладу.
2. Проаналізувати принципи роботи із GPU.
3. Розробити програмний модуль для розкладу матриць.

РОЗДІЛ 1: ІСТОРІЯ РОЗВИТКУ SVD-РОЗКЛАДУ

1.1 Опис SVD-розкладу

Однією з найбільш плідних ідей в теорії матриць є матричне розкладання або канонічна форма. Останнім часом матричні розкладання стали оплотом численних методів лінійної алгебри, які служать основою вирішення безлічі проблем. З численних розкладів матриць, сингулярному розкладанню, яке представляє з себе факторизацию матриці $\mathbf{A}$ в добуток $\mathbf{U\Sigma V H}$, де $\mathbf{U}$, $\mathbf{V}$ – унітарні матриці, а $\mathbf{\Sigma}$ – діагональна матриця, відводиться особливе місце.

На це є кілька причин:

• Той факт, що в розкладанні беруть участь унітарні матриці робить його ідеальним механізмом для геометризації перетворення $\mathbf{A}$ в n-вимірному просторі.

• Сингулярне розкладання є стійким, тобто малих збурень матриці $\mathbf{A}$ відповідають малі обурення матриці Σ і навпаки.

• Діагональна матриця Σ дозволяє легко зрозуміти чи є матриця A майже виродженою і, якщо вона такою є, сингулярне розкладання дає можливість знизити ранг матриці $\mathbf{A}$ з найменшою похибкою.

• Завдяки зусиллям математика Джина Голуба були отримані ефективні, стійкі алгоритми обчислення сингулярного розкладання на ЕОМ, які використовуються вже протягом більше 40 років і включені в усі математичні пакети сьогоднішнього часу.

1.2 Матричний розклад

Інтригуючим є той факт, що більшість класичних матричних розкладань були отримані задовго до використання матриць і матричної нотації: вони задавалися за допомогою визначників, лінійних систем рівнянь і, особливо, білінійних і квадратичних форм. Батьком розвитку даного напрямку досліджень став Гаусс. Один з його алгоритмів, розроблений ним в 1823 році, розкладає матрицю квадратичної форми $\mathbf{xTAx}$ в твір $\mathbf{RD\text{-}1R}$, де $\mathbf{D}$ - діагональна матриця, а $\mathbf{R}$ - верхньокутна матриця з елементами матриці $\mathbf{D}$ на головній діагоналі.

Гаусс також розробив алгоритм для ефективного знаходження зворотної матриці, який перетворював систему виду $\mathbf{y} = \mathbf{Ax}$ до системи виду $\mathbf{x} = \mathbf{By}$. Його вміння маніпулювати квадратичними формами і системами рівнянь дозволили йому розробити метод найменших квадратів, який ліг в основу методів машинного навчання близько 30 років тому.

Коші в 1829 році встановив властивості власних значень і власних векторів симетричних матриць, розглядаючи відповідні їм однорідні системи рівнянь. У 1846 році Якобі опублікував алгоритм діагоналізації симетричних матриць, а в посмертній статті 1857 року отримав LU-розкладання білінійних форм. Вейєштрасс в 1868 році отримав канонічні форми для пар білінійних функцій - те, що в наш час називається узагальненою проблемою власних значень: $\mathbf{Ax} = \mu\mathbf{Bx}$, де $\mathbf{B}$ - деяка матриця.

Підводячи підсумок, можна сказати, що розробка сингулярного розкладання - результат, отриманий в результаті багатьох років досліджень білінійних форм.

1.3 Розвиток SVD-розкладу

Сингулярне розкладання було спочатку розроблено в диференціальній геометрії при вивченні властивостей білінійних форм вченими Еудженіо Бельтрамі і Камілем Жорданом незалежно в 1873 і 1874 роках відповідно.

Джеймс Джосеф Сильвестр прийшов до поняття сингулярного розкладання для квадратних матриць в 1889 році і, ймовірно, також незалежно від Еудженіо Бельтрамі і Каміля Жордана. Сильвестр називав сингулярні значення канонічними множителями матриці.

Перший доказ сингулярного розкладання для прямокутних і комплексних матриць було здійснено математиками Карлом Еккартом і Гейлом Янгом в 1936 році.

У 1907 році Ерхарт Шмід визначив аналог сингулярного розкладання для інтегральних операторів, не знаючи про те, що паралельно ведеться робота над сингулярним розкладанням для скінченновимірних матриць. Його теорія була далі розвинена математиком Емілем Пікаром в 1910 році. Саме Пікар першим назвав $\boldsymbol{\sigma k}$ сингулярними значеннями.

Практичні методи обчислення SVD-розкладання сходять до робіт Когбетлянс в 1954, 1955 і Гестенес в 1958, алгоритм яких сильно нагадує метод Якобі для обчислення власних значень з використанням поворотів Гивенса. Однак цей алгоритм був замінений методом, розробленим математиками Геном Голубом і Вільямом Кеханом в 1965, який використовує перетворення Хаусхолдера. У 1970 році Голуб разом із Крістіаном Рейхом опублікували варіант алгоритму Голуб/Кахан, який і на сьогодні є одним з найбільш використовуваних.

Висновки

В даному розділі було розглянуто завдяки кому було покладено початок розвитку матричного розкладу загалом, та SVD-розкладу зокрема. Ознайомився із історичною довідкою, математиками які займалися лінійною алгеброю та розвитком теорії матриць.

РОЗДІЛ: 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Бельтрамі

Бельтрамі і Жордан вважаються засновниками теорії сингулярного розкладання. Бельтрамі - за те, що він першим опублікував роботу про сингулярне розкладання, а Жордан - за елегантність і повноту своєї роботи. Робота Бельтрамі з'явилася в журналі "Journal of Mathematics for the Use of the Students of the Italian Universities" в 1873 році, основна мета якої полягала в тому, щоб ознайомити студентів з білінійними формами.

Бельтрамі починає своє підтвердження з розгляду білінійної форми:

$$f(x,y) = xTAy,$$

де A - дійсна квадратна матриця n-го порядку.

Якщо зробити заміну виду:

$$x = U\xi \text{ и } y = V\eta,$$

то білінійну форму можна переписати у вигляді:

$$f(x,y) = \xi TS\eta, \text{ де } S = UTAV$$

Якщо зажадати, щоб матриці U, V були ортогональними, то існує всього $n2$ - n ступенів свободи при їх завданні, які можна використовувати, щоб обнулити внедіагональні елементи матриці S.

Будемо з'ясовувати кількість параметрів, які задають ортогональний базис або, що те ж саме, елементи ортогональної матриці. Розглянемо 2-мірний простір. Одиничний вектор в двовимірному просторі має всього 1 ступінь свободи - кут обертання навколо початку координат, тому що довжина у нього

постійна. З огляду на той факт, що другий прямокутний вектор, перпендикулярний першому задається однозначно з точністю до знака, але знак не є ступенем свободи, ми приходимо до висновку, що репер в 2-вимірному просторі задається одним ступенем свободи. Розглянемо 3-мірний простір. Одиничний вектор в ньому задається за допомогою 2-ух ступенів свободи (широта і довгота), а решта 2 вектори лежать в площині, перпендикулярній цьому вектору, тобто в 2-вимірному просторі, яке вже було нами розглянуто. Тобто репер в 3-вимірному просторі задається за допомогою 2 + 1 = 3 ступенів свободи.

Продовжуючи роздуми з індукції, ми приходимо до висновку, що репер в n-вимірному просторі задається за допомогою $1 + 2 + \ldots + (n - 1) = n(n-1)$ ступенів свободи. Отриманий результат показує, що одна ортогональна матриця задається за допомогою $n(n-1)$ ступенів свободи, а так як в
нашому випадку 2 такі матриці, то ми маємо
$n2 - n$ ступенів свободи при їх завданні.

Необхідно зауважити, що висновок, запропонований Beltrami передбачає, що матриця Σ, а отже і матриця A - невироджені. На підставі вищевикладених міркувань Beltrami написав алгоритм знаходження сингулярного розкладання.

2.2 Сильвестр

Сильвестр написав 2 статті по сингулярному розкладу. У першій статті він привів ітеративний алгоритм редукування квадратичної форми до діагонального вигляду, а в примітці до статті вказав, що аналогічний ітераційний метод може діагоналізувати білінійну форму. У другій статті він написав правило, за яким завдання діагоналізації білінійної форми зводилася до діагоналізації квадратичної форми. Це правило виявилося нічим іншим, як алгоритмом Бельтрамі.

2.3 Полярний розклад

Будь-яку матрицю A можна представити у вигляді множення ортогональної матриці Q і симетричної матриці S:

$$A = QS$$

Всі власні вектори симетричною матриці S - невід'ємні числа.

Геометричний сенс - будь-яке лінійне перетворення може бути представлено у вигляді двох послідовних перетворень:

1. масштабування вздовж ортогональних напрямків на невід'ємні коефіцієнти;
2. послідовність поворотів з можливими відбитками між ними.

Для того, щоб довести цю теорему, наведемо приклад алгоритму для побудови даного розкладання і покажемо, що матриці Q і S унікальні для кожної матриці A.

$$ATA = SQTQS = S2\ S = ATA$$

Знаючи S і A можна знайти Q:

$$Q = AS{-}\mathbf{1}$$

Матриці Q і S унікальні для кожної матриці A. Покажемо, що матриця Q є ортогональною:

$$QTQ = S{-}1ATAS{-}1 = S{-}1S2S{-}1 = I$$

$$QQT = AS{-}1S{-}1AT = A(S2){-}1AT = AA{-}1(AT){-}1AT = I$$

Скористаємося полярним розкладанням і представимо матрицю A у вигляді:

$$A = QS$$

Скористаємося властивістю спектрального розкладання симетричних матриць:

$$S = X\Lambda XT$$

Симетричні матриці мають повний набір власних значень, а власні вектори можуть бути обрані ортогональними. Таким чином, приходимо до такої формули:

$$A = QX\Lambda XT$$

У цій формулі Q, X - ортогональні матриці, тому їх твором є також ортогональна матриця $Y = QX$. Підставляючи всі в одну формулу, отримуємо SVD-розкладання:

$$A = Y\Lambda XT$$

Або більш традиційний запис:

$$A = U\Sigma VT$$

Коефіцієнти σ1, ..., σn, що стоять на головній діагоналі матриці Σ, називаються сингулярними значеннями і еквівалентні власним значенням матриці ATA.

2.4 Сингулярні вектори

SVD-розкладання - аналог спектрального розкладання, який працює з будь-якими матрицями.

Стовпці матриці V називаються правими сингулярними векторами і завжди ортогональні один одному. Стовпці матриці U називаються лівими сингулярними векторами і також ортогональні один одному.

2.5 Практичне використання SVD-розкладу

Якщо Василю подобаються фільми про трактори і не подобаються фільми про поросят, а Петру - навпаки, було б просто чудово навчитися розуміти, які фільми «про поросят», і рекомендувати їх Петру, а які фільми - «про трактори», і рекомендувати їх Васі .

У нас є матриця, що складається з рейтингів (лайків, фактів покупки і т.п.), які користувачі (рядки матриці) присвоїли продуктам (стовпці матриці). Давайте придивимося до матриці, в якій записані відомі нам рейтинги. Як правило, один користувач не зможе оцінити значну частку продуктів, і навряд чи буде багато продуктів, які готова оцінити значна частка користувачів. Це означає, що матриця R розріджена, чи не можна цим якось скористатися? Головним інструментом для нас стане так зване сингулярне розкладання матриці R.

Сингулярне розкладання - це досить простий, але дуже потужний інструмент. Власне, це один з головних з практичної точки зору результатів лінійної алгебри, і результат вже досить не новий (властивості SVD були вивчені найпізніше в 1930-х роках), - і тим дивніше буває, коли університетські курси лінійної алгебри, досить докладні в яких -то інших аспектах, зовсім обходять SVD стороною.

SVD дуже широко вживається в машинному навчанні; фактично, якщо ви хочете щось чимось наблизити, не виключено, що вам десь по дорозі зустрінеться SVD. Класичний приклад застосування SVD - шумозаглушення, наприклад, в зображеннях. Розглянемо (чорно-біле) зображення як матрицю X розміру , елементи якої - інтенсивності кожного пікселя. Тепер спробуємо вибрати f стовпців пікселів із зображення, злічити їх «репрезентативними» і уявити кожен з решти стовпців у вигляді лінійної комбінації цих. Я не буду зараз заглиблюватися в математику, але в результаті, коли ви знайдете оптимальні уявлення кожного стовпчика, вийде, що ви представили вихідну матрицю у вигляді твору матриць розміру і , якраз наблизили матрицю X матрицею малого рангу f.

Основна ж властивість SVD - в тому, що SVD дає оптимальне таке наближення, якщо в матриці D просто залишити рівно f перших діагональних елементів, а решту «занулити»:

$$X = U D V^\top = U \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \sigma_k \end{bmatrix} V^\top \approx U \begin{bmatrix} \sigma_1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & \sigma_f & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} V^\top$$

Рисунок 2.1

У діагональної матриці D, яка в середині сингулярного розкладання, елементи впорядковані за розміром, так що обнулити останні елементи - це значить обнулити найменші елементи.

Якщо добре підібрати f, то в результаті зображення і у вазі втратить, і шуму на ньому стане менше. А f в даному випадку можна підбирати, виходячи з розміру сингулярних значень матриці, тобто тих самих діагональних елементів матриці D: бажано відкидати якомога більше, але при цьому якомога більше маленьких таких елементів.

Висновки

В даному розділі було розглянуто математичний доказ матричного розкладу. Поэтапно розглянуто всі етапи розвитку матричного розкладу, починаючи із білінійних форм Бельтрамі.

РОЗДІЛ 3: PYTHON AND GPU

3.1 Загальна характеристика Python

Python має низку привабливих переваг до яких відноситься простота реалізації програмних рішень, наочність і лаконічність коду, наявність великої кількості бібліотек і численного активного ком'юніті. У той же час, відома всім повільність пітона часто обмежує його застосування для "важких" обчислень. Для ряду задач можна домогтися істотного прискорення розрахунків шляхом використання технології CUDA для паралельних обчислень на GPU. Мета цього розділу - аналіз можливостей ефективного використання python для розрахунків на GPU і порівняння продуктивності різних python-рішень з реалізацією на C.

У багатьох випадках ми використовуємо технологію CUDA для прискорення розрахунків за рахунок використання можливостей паралельних обчислень на GPU, при цьому програми пишуться на мові C. У той же час, в деяких випадках хотілося б мати можливість реалізовувати розрахунки на пітоні, тому що це зручно, швидко , гнучко, лаконічно. При цьому, однак, дуже важливо не втрачати в швидкості виконання програм, оскільки деякі розрахунки можуть займати від декількох годин до доби.

3.2 Python with Numba and TensorFlow

Бібліотека Numba надає можливість jit (just-in-time) компіляції коду на пітона в байт-код можна порівняти за продуктивністю з C або Fortran кодом. Numba підтримує компіляцію і запуск python-коду не тільки на CPU, але і на GPU, при цьому стиль і вид програми, що використовує бібліотеку Numpy, залишається чисто пітоновскім.

Відзначимо тут кілька приємних особливостей. По-перше, ця реалізація набагато коротше і наочніше. Тут ми використовували двовимірні масиви, що робить код набагато більш зручним для читання. По-друге, якщо в C від нас вимагалося передати всі константи (наприклад N) шляхом виконання функцій на зразок: cudaMemcpyToSymbol (dN, & N, sizeof (int)); то тут ми просто користуємося глобальними змінними як у звичайній python-функції. Ну і нарешті, реалізація не вимагає ніяких знань мови C і архітектури GPU.

3.3 PyCUDA

На відміну від Numba тут від розробника потрібно написати код ядра на C, тому без знання цієї мови не обійтися. З іншого боку крім власне ядра на C нічого писати не треба.

Виглядає це все як чистий пітон за винятком локальної вставки C-коду.

3.4 Порівняння продуктивності

На рисунку 4.1 і в таблиці 4.1 наведені залежності часу виконання тестової програми (в секундах) від розміру сітки n, отримані при запуску C-коду (крива CUDA C) і python-реалізацій з бібліотекою Numba і двовимірними масивами (Numba 2DArr), з бібліотекою Numba і одновимірними масивами (Numba 1DArr), з бібліотекою PyCUDA (крива PyCUDA).

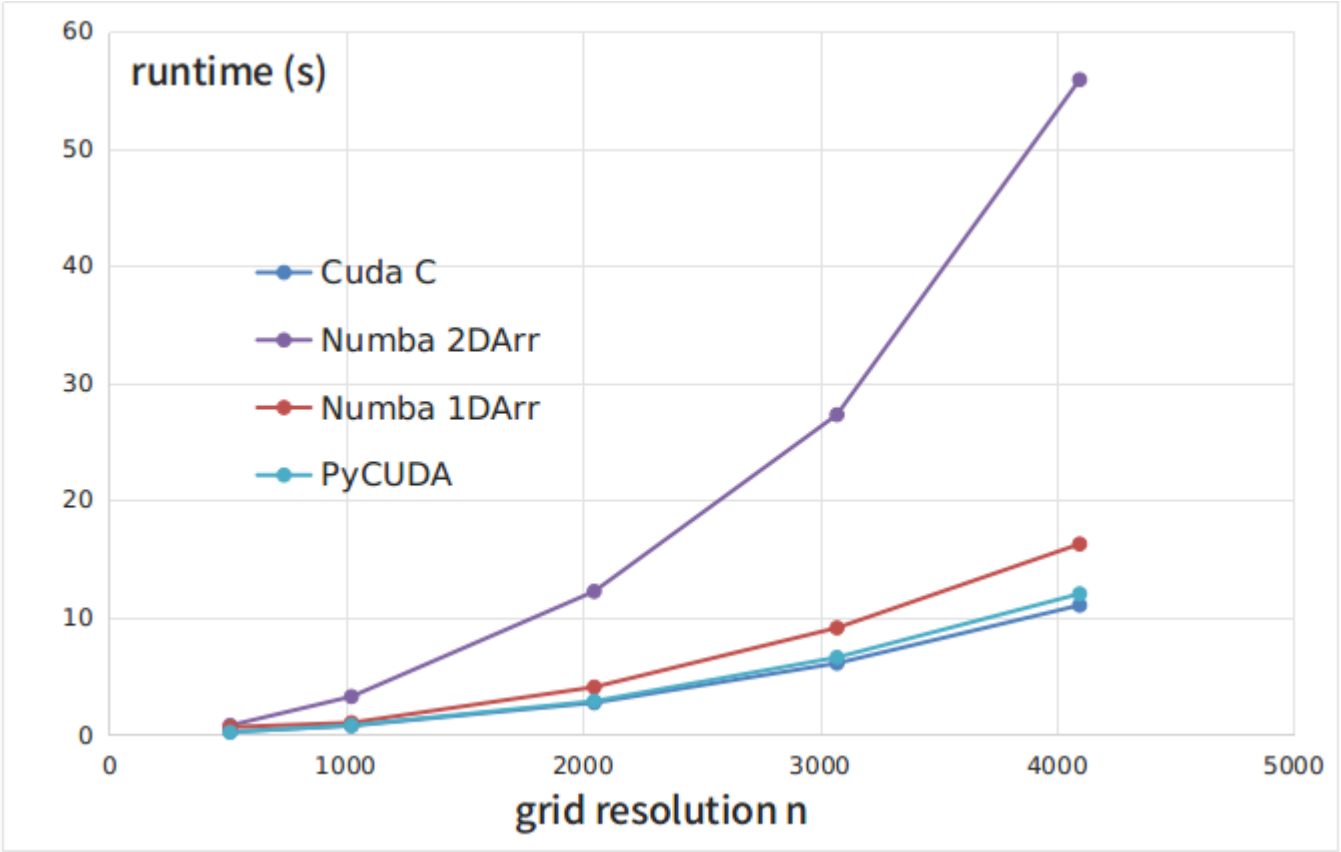

Рисунок 4.1

Таблиця 4.1

n	**Cuda C**	**Numba 2DArr**	**Numba 1Darr**	**PyCUDA**
512	0.25	0.8	0.66	0.216
1024	0.77	3.26	1.03	0.808
2048	2.73	12.23	4.07	2.87
3073	6.1	27.3	9.12	6.6
4096	11.05	55.88	16.27	12.02

На рисунку 4.2 наведені відносини часу виконання різних python-реалізацій до часу виконання С-коду. Як видно з малюнків, найповільнішої, з розглянутих, є реалізація за допомогою бібліотеки Numba. При цьому, цей підхід є найбільш наочним і простим.

Найшвидшим рішенням виявилося використання бібліотеки PyCUDA. У той же час, як зазначалося вище, використання цієї бібліотеки кілька більш трудомістке, оскільки вимагає написання ядра на C. Однак витрати окупаються і швидкість виконання такої python-програми за все на 5-8% менше ніж програми, написаної повністю на C.

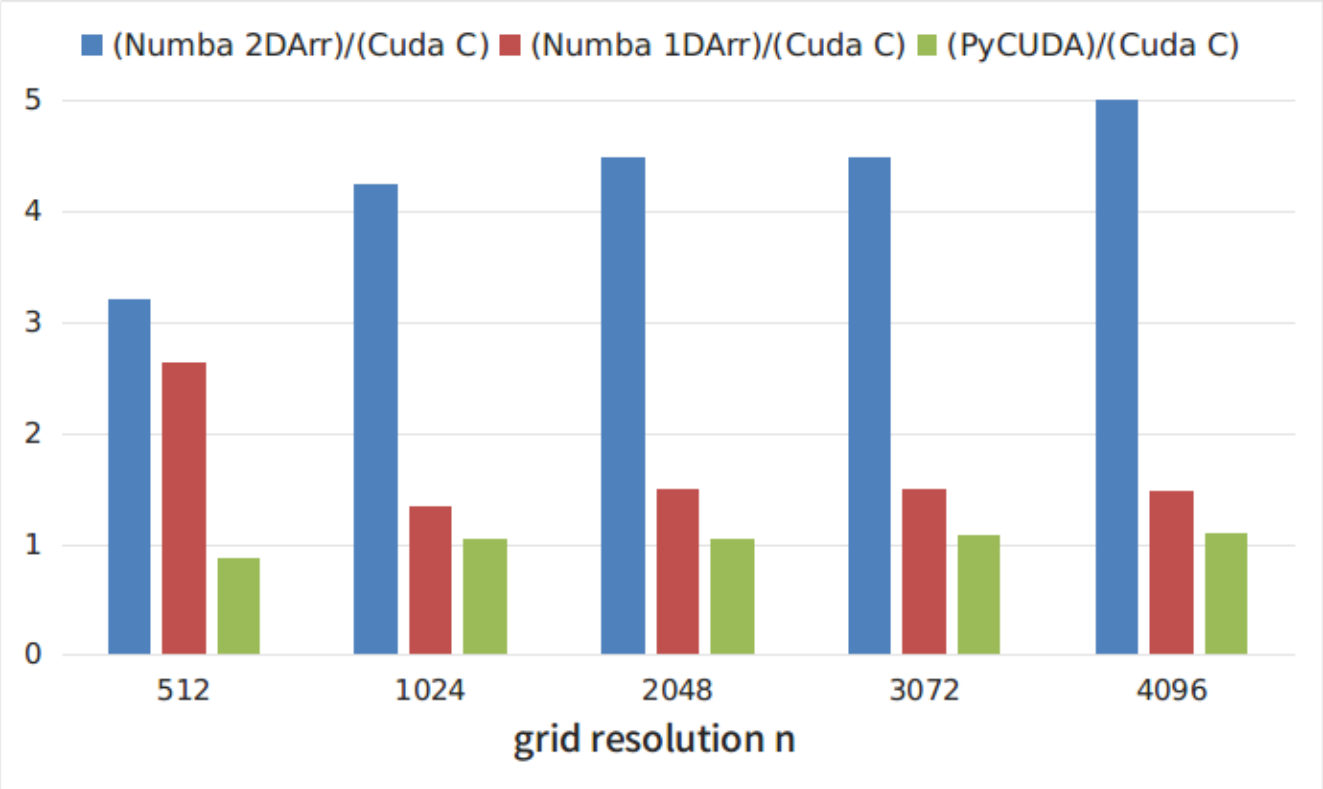

Рисунок 4.2

Висновки

Чудес не буває і найпростіші і наочні рішення виявляються одночасно і самими повільними. У той же час, наявні бібліотеки дозволяють добитися швидкості виконання python-програм, порівнянної зі швидкістю виконання чистого С-коду. Існуючі бібліотеки дають розробнику вибір між більш і менш високорівневими рішеннями. Цей вибір, проте, завжди є компроміс між швидкістю розробки і швидкістю виконання програми.

РОЗДІЛ 4: Програмна реалізація SVD-розкладу

4.1 Бібліотеки які було використано

Як було описано у попередньому розділі, мовою програмування застосунку було обрано Python, як мову на якій вже присутня велика кількість бібліотек для вирішення математичних задач.

Бібліотеки які використовувались:

1. Datetime – надає класи для обробки часу і дати різними способами;
2. Random – надає функції для генерації випадкових чисел, букв, випадкового вибору елементів послідовності;
3. Tqdm – це модуль Python, який легко виводить на консоль динамічно оновлюваний індикатор виконання;
4. PySimpleGUI – графічний інтерфейс;
5. NumPy – це розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами;
6. TensorFlow – бібліотека для машинного навчання, в нашому випадку саме з її допомогою проводимо розрахунку на GPU;
7. NumPy.linalg – лінійна алгебра.

4.2 Роз`яснення функцій

```
#Генеруємо трьохдіагональну матрицю:
def tridiag(n):
    return np.diag(np.random.rand(n - 1), -1) \
        + np.diag(np.random.rand(n), 0) \
        + np.diag(np.random.rand(n - 1), 1)


def find_rank(A):
    matrix = np.array(A, dtype=TYPE)
    # Надаємо matrix початкову матрицю
    n, m = matrix.shape
    #matrix.shape повертаємо дані про розмір матриці і записуємо в n, m відповідно
    p = min(n, m)
    # записуємо в змінну p мінімальний елемент матриці
    n, p = matrix.shape
    #matrix.shape повертаємо дані про розмір матриці і записуємо вже в n, p
    k = min(n, p)
    # знаходимо рангове наближення

    return k
```

```
def rsvd(A, k, n_oversamples=None, n_subspace_iters=None,
        return_range=False):
```

"""

Параметри цієї функції:

A: безпосередньо вихідна матриця nxm

k: рангове наближення

n_oversamples: параметр передискретизации для гауссових випадкових вибірок. По суті збільшуємо точність обчислень або можна сказати "згладжуємо"

n_subspace_iters: Кількість статечних ітерацій

return_range: якщо 'True', то повертається підставу для приблизного діапазону матриці A

Ця функція повертає матриці U, S і Vt в усіченому SVD

"""

```
# як раз умова, за якою "згладжуємо", по дефолту у нас стоїть None, тому виконається перша умова     if n_oversamples is None:
    n_samples = 2 * k
else:
    n_samples = k + n_oversamples

# Шукаємо приблизний діапазон матриці A
```

```
    Q = find_range(A, n_samples, n_subspace_iters)


    B = Q.T @ A
    U_tilde, S, Vt = np.linalg.svd(B)
    U = Q @ U_tilde


    # Власне і є усічення
    U = U[:, :k]
    S = S[:k]
    Vt = Vt[:k, :]


    if return_range:
        return U, S, Vt, Q
    return U, S, Vt



def find_range(A, n_samples, n_subspace_iters=None):
    """
З огляду на матрицю A і кількість вибірок, обчислюємо ортонормовану матрицю, яка наближає діапазон A.
"""
    m, n = A.shape
    O = np.random.randn(n, n_samples)
```

```
    Y = A @ O


    if n_subspace_iters:
        return subspace_iter(A, Y, n_subspace_iters)
    else:
        return ortho_basis(Y)



#
def subspace_iter(A, Y0, n_iters):
    """
Ця функція використовує чисельно стійкий алгоритм ітерації підпростору для зменшення ваги менших сингулярних значень.
    параметри:
    Y0: початковий приблизний діапазон матриці A
    n_iters: кількість ітерацій підпростору
    Функція повертає ортонормованій приблизний діапазон матриці A    """
    Q = ortho_basis(Y0)
    for _ in range(n_iters):
        Z = ortho_basis(A.T @ Q)
        Q = ortho_basis(A @ Z)
    return Q
```

```
def ortho_basis(Matrix):
    """
Ця функція обчислює ортонормованій базис для матриці
    парметр Matrix - це матриця nxm
    Повертає функція ортонормовану (ортогональну) основу для матриці.
"""
    Q, _ = np.linalg.qr(Matrix)
    return Q
```

4.3 Порівняння часу витраченого на розрахунок

start with matrix A:

[[0.22744541 0.91374699 0. ... 0. 0. 0.]

[0.38731132 0.52061492 0.29188207 ... 0. 0. 0.]

[0. 0.21607423 0.26494088 ... 0. 0. 0.]

...

[0. 0. 0. ... 0.67825269 0.38306959 0.]

[0. 0. 0. ... 0.4597799 0.60109534 0.05053876]

[0. 0. 0. ... 0. 0.48747988 0.29745832]]

Device mapping:

U p x n orthogonal matrix:

[[-1.21430643e-17 -7.97972799e-17 1.95156391e-17 ... -1.63172427e-17

4.53951886e-08 3.35086234e-18]

[-4.33680869e-18 1.02348685e-16 6.24500451e-17 ... 1.07878116e-17

-1.22157994e-08 -3.79470760e-19]

[4.46691295e-17 8.84708973e-17 -1.19695920e-16 ... 6.45405224e-17

-1.68979928e-07 -2.58040117e-17]

...

[-1.66533454e-16 -7.66024614e-14 3.24046345e-15 ... 1.51757024e-11

* 8.94466792e-19 2.19144364e-17]*

[-3.12250226e-17 -1.31058359e-14 5.06539255e-16 ... -1.08118258e-10

* 2.43132337e-17 1.92276479e-17]*

[-9.97465999e-17 -4.37776443e-15 1.37910516e-16 ... 9.34417953e-11

* -4.02781107e-17 -2.50653990e-17]]*

S n x n diagonal matrix:

[[2.13790461e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00

* 0.00000000e+00 0.00000000e+00]*

[0.00000000e+00 2.12938057e+00 0.00000000e+00 ... 0.00000000e+00

* 0.00000000e+00 0.00000000e+00]*

[0.00000000e+00 0.00000000e+00 2.06681884e+00 ... 0.00000000e+00

* 0.00000000e+00 0.00000000e+00]*

...

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 2.45437716e-03

* 0.00000000e+00 0.00000000e+00]*

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00

* 1.52564865e-03 0.00000000e+00]*

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00

0.00000000e+00 1.30469379e-03]]

V transposed n x p orthogonal matrix

[[-2.03671178e-17 -8.87320346e-17 1.63934393e-16 ... -8.47871526e-17

-7.72382093e-17 -2.24511089e-17]

[1.12182223e-17 4.42631755e-18 5.58319404e-17 ... -3.61916359e-14

-1.84603465e-14 -9.25551893e-16]

[1.47872606e-17 -1.87960449e-18 -2.05882163e-17 ... 1.56285562e-15

7.87792475e-16 4.44040347e-17]

...

[-2.38848929e-16 1.36226057e-16 3.02941730e-17 ... 6.25688661e-09

-5.55132728e-09 9.09838279e-09]

[3.66638157e-06 -9.12541907e-07 -3.23749512e-06 ... 1.32962051e-16

-1.98954163e-16 4.71186135e-17]

[2.55243382e-16 -2.79490200e-17 -3.15753627e-16 ... 6.10585916e-18

-3.08025052e-16 1.61046470e-16]]

```
Shape: (512, 512) Device: /gpu:0
Time taken: 0:00:00.588034
```

Рисунок 4.1

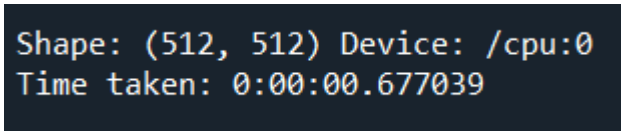

Рисунок 4.2

Навіть на, відносно, невеликій матриці ми вже маємо різницю 1-5 десятих секунди у швидкості виконання розрахунку на користь розрахунку на GPU.

start with matrix A:

[[0.59578412 0.7504306 0.   ... 0.   0.   0.   ]

 [0.94543455 0.19499187 0.40202033 ... 0.   0.   0.   ]

 [0.   0.8054003 0.13691971 ... 0.   0.   0.   ]

 ...

 [0.   0.   0.   ... 0.62332396 0.81052824 0.   ]

 [0.   0.   0.   ... 0.86561207 0.68478411 0.98666875]

 [0.   0.   0.   ... 0.   0.85927826 0.77382136]]

Device mapping:

U p x n orthogonal matrix:

[[2.31585584e-16 1.35308431e-16 5.76795556e-17 ... 1.16467030e-19

```
 -2.03287907e-19 -2.34092113e-22]
[-3.29597460e-17 -2.74303150e-17 -1.41163123e-16 ... -9.14795583e-20
  1.17737580e-19  1.12496563e-22]
[ 1.00613962e-16  3.29597460e-17 -5.63785130e-17 ... -2.59403840e-19
  1.62630326e-19 -6.61744490e-24]
...
[ 6.08887940e-16  1.26548078e-15  5.43124573e-14 ...  1.68245307e-16
  2.22614658e-16  3.67281427e-19]
[ 7.25764934e-16  1.65058939e-15  7.03848872e-14 ...  3.35861269e-16
  4.47631713e-16  7.17267334e-19]
[ 5.55978874e-16  1.10935566e-15  4.72454378e-14 ... -4.29748942e-16
 -5.76491047e-16 -9.28867580e-19]]
```

S n x n diagonal matrix:

```
[[2.20264978e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 2.04459140e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 2.04024163e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 ...
```

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 4.71319987e-05
0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
2.89167543e-05 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
0.00000000e+00 6.98925263e-08]]

V transposed n x p orthogonal matrix

[[6.73964674e-17 2.09365054e-17 9.30058786e-17 ... 5.45987346e-16
7.07917498e-16 5.70948162e-16]
[6.71895157e-17 1.13220023e-16 -1.02094573e-16 ... 1.22258703e-15
1.50151836e-15 1.31669239e-15]
[-2.06357210e-17 -2.42640329e-17 -9.46597995e-17 ... 5.49352230e-14
6.54372390e-14 5.23470156e-14]
...
[5.74624422e-16 -2.56062878e-16 -1.09242022e-15 ... -8.03084344e-16
-1.36291629e-15 1.51135300e-15]
[4.11499920e-16 -2.68144907e-16 -9.04085278e-16 ... -1.67373404e-16
-2.02512234e-17 5.60082085e-17]
[1.19146442e-16 -6.84239646e-17 -2.44843307e-16 ... 4.91326403e-16
1.16532912e-15 -1.26799776e-15]]

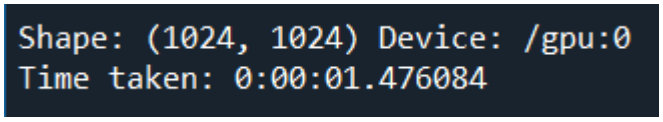

Рисунок 4.3

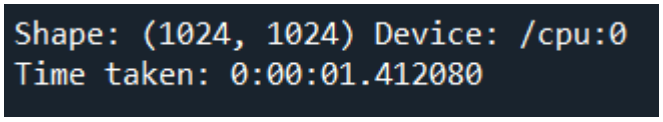

Рисунок 4.4

start with matrix A:

[[0.26557287 0.05590035 0. ... 0. 0. 0.]

[0.59444521 0.09034322 0.922383 ... 0. 0. 0.]

[0. 0.26656643 0.91628776 ... 0. 0. 0.]

...

[0. 0. 0. ... 0.78347976 0.22950961 0.]

[0. 0. 0. ... 0.85434224 0.34071058 0.64133111]

[0. 0. 0. ... 0. 0.13196848 0.65901847]]

Device mapping:

U p x n orthogonal matrix:

[[9.32413868e-18 -5.05238212e-17 8.85793175e-17 ... -8.78203760e-18
  -1.04625510e-17 9.13440330e-18]
 [1.74014449e-17 -1.30537942e-16 5.37764278e-17 ... 3.17129135e-18
   4.93311988e-18 -3.87941090e-18]
 [5.83300769e-17 -1.17974749e-16 -7.34004871e-17 ... -2.69695290e-18
  -1.27393755e-18 5.62091064e-18]
 ...
 [7.08526120e-17 5.14996032e-17 2.08763128e-16 ... 5.55653613e-17
   4.56178064e-17 1.56816292e-16]
 [-2.50450702e-17 1.07336015e-16 1.39319979e-16 ... -2.32053146e-17
  -2.48824399e-17 -1.64961361e-16]
 [-4.87890978e-17 1.82145965e-17 2.40692882e-17 ... 1.32272665e-17
   2.84603070e-17 1.76887584e-16]]

S n x n diagonal matrix:

[[2.12810604e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 2.11013677e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 2.10059974e+00 ... 0.00000000e+00

```
 0.00000000e+00 0.00000000e+00]
...
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 7.08526872e-04
 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
 4.51764172e-04 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
 0.00000000e+00 1.80351534e-04]]
```

V transposed n x p orthogonal matrix

```
[[ 4.47842147e-18 -4.30841037e-17  3.39366877e-18 ...  3.11941331e-17
   4.15244651e-18 -3.06718522e-17]
 [-4.21934809e-17 -1.81799248e-20 -1.11425487e-16 ...  5.57826165e-17
   3.41404012e-17  3.59323481e-17]
 [ 2.53245711e-17  4.54201733e-17  3.37386365e-17 ...  1.24960830e-16
   6.73030410e-17  9.14708340e-17]
 ...
 [ 3.27056316e-17 -1.52966718e-16 -7.38509339e-18 ... -4.30208592e-16
   2.21759142e-15 -4.74623019e-16]
 [ 9.88968410e-20 -1.18835323e-16  2.27657503e-17 ... -5.52710754e-16
   1.79777008e-15 -3.73831892e-16]
```

[5.72476862e-17 -2.00002424e-16 5.20687907e-17 ... -4.60066204e-16

  1.88064582e-15 -3.53591472e-16]]

```
Shape: (2048, 2048) Device: /gpu:0
Time taken: 0:00:11.275644
```

Рисунок 4.5

```
Shape: (2048, 2048) Device: /cpu:0
Time taken: 0:00:11.088635
```

Рисунок 4.6

start with matrix A:

[[0.46287873 0.59789483 0.   ... 0.   0.   0.   ]

 [0.19306374 0.23146304 0.18509851 ... 0.   0.   0.   ]

 [0.   0.12991226 0.25360433 ... 0.   0.   0.   ]

 ...

 [0.   0.   0.   ... 0.22572031 0.24030621 0.   ]

 [0.   0.   0.   ... 0.80600786 0.50104229 0.45105596]

 [0.   0.   0.   ... 0.   0.51468008 0.69512022]]

Device mapping:

U p x n orthogonal matrix:

[[1.99628725e-17 5.78150808e-17 -1.03920778e-16 ... 5.90619133e-17
   3.42404599e-17 -8.72105122e-18]
 [-4.90533720e-17 8.56519716e-18 6.41847686e-17 ... -1.57778521e-16
  -8.03732623e-17 1.22514845e-17]
 [1.25225351e-17 1.49077799e-18 6.12574227e-17 ... -4.22838847e-17
  -1.39862080e-17 -2.99510850e-18]
 ...
 [-1.73472348e-18 -2.93547738e-17 3.26344854e-17 ... 2.18060162e-17
  -4.58481994e-17 1.03880121e-17]
 [9.76205472e-18 -2.31714333e-17 -1.53685658e-17 ... 9.07341693e-18
   1.72388145e-17 -5.08219768e-20]
 [6.23416249e-18 -1.98137947e-17 -2.26327204e-17 ... -2.07014852e-18
  -1.93462325e-18 -6.11557788e-19]]

S n x n diagonal matrix:

[[2.28307391e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 2.20320477e+00 0.00000000e+00 ... 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 2.15684285e+00 ... 0.00000000e+00

0.00000000e+00 0.00000000e+00]

...

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 2.44436998e-04

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00

1.28633584e-04 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00

0.00000000e+00 9.31573060e-05]]

V transposed n x p orthogonal matrix

[[-7.72568556e-18 -3.63776908e-17 1.43918739e-17 ... -1.96636221e-17

1.61747888e-19 3.13913131e-17]

[2.48672013e-17 9.55095861e-17 7.17538034e-18 ... -1.35884278e-17

-2.24096277e-17 -2.32334948e-17]

[-1.39281660e-17 -4.38837312e-17 2.87378672e-18 ... 1.27965040e-17

-8.29506454e-18 -3.42721688e-17]

...

[-1.55089541e-16 1.72674558e-16 -2.99366364e-16 ... -7.40967482e-17

-1.55149331e-16 1.93455549e-16]

```
[-1.44787032e-16  1.16206761e-16 -5.55331235e-17 ... -9.47999275e-18
  2.88668828e-17 -1.82688066e-17]
 [-1.29159986e-17  3.07076239e-17 -8.91232497e-17 ... -2.23747141e-16
  3.55045718e-16 -2.41797413e-16]]
```

```
Shape: (4096, 4096) Device: /gpu:0
Time taken: 0:01:10.774048
```

Рисунок 4.7

```
Shape: (4096, 4096) Device: /cpu:0
Time taken: 0:01:08.549921
```

Рисунок 4.8

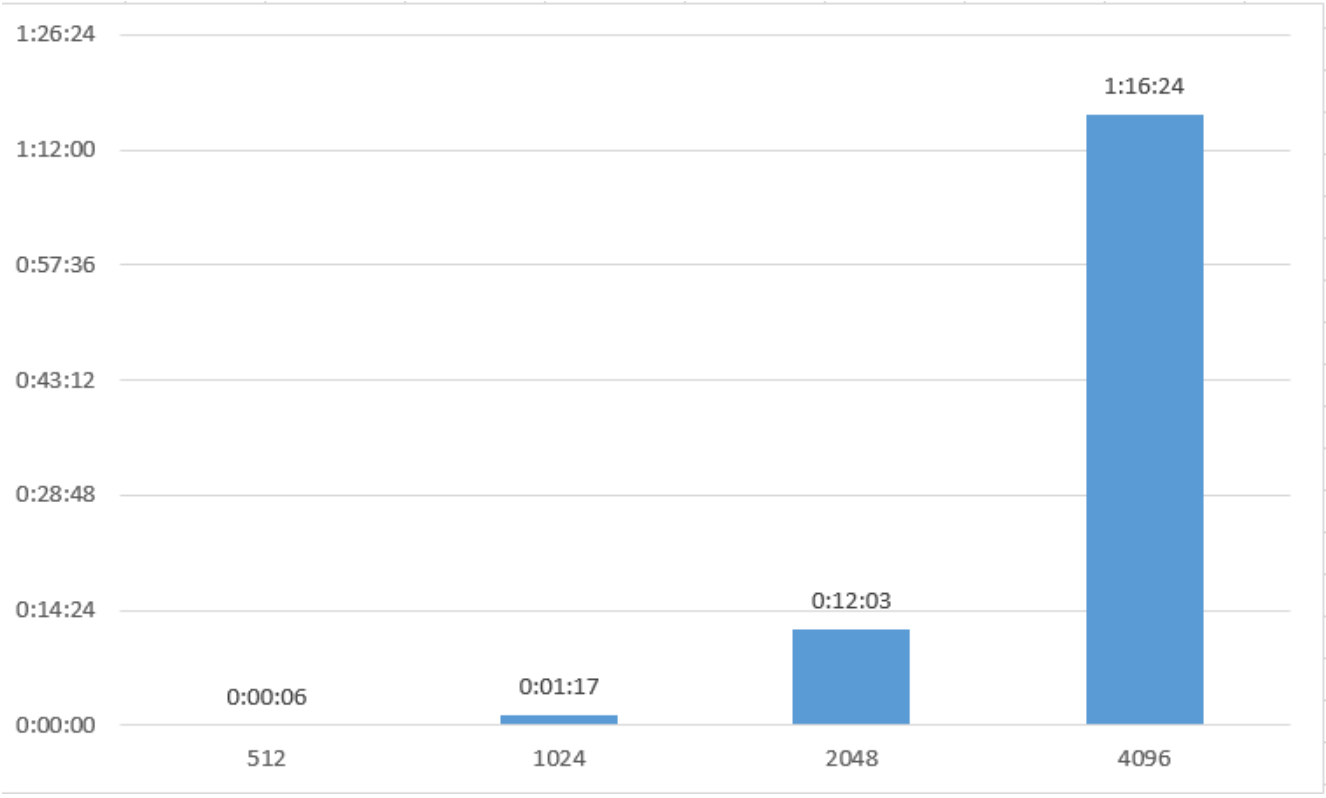

Рисунок 4.9. Час який витрачено на розрахунок після оптимізації коду. У хвилинах.

1. Час розрахунку для матриці 512 на 512;
2. Час розрахунку для матриці 1 024 на 1 024;
3. Час розрахунку для матриці 2 048 на 2 048;
4. Час розрахунку для матриці 4 096 на 4 096.

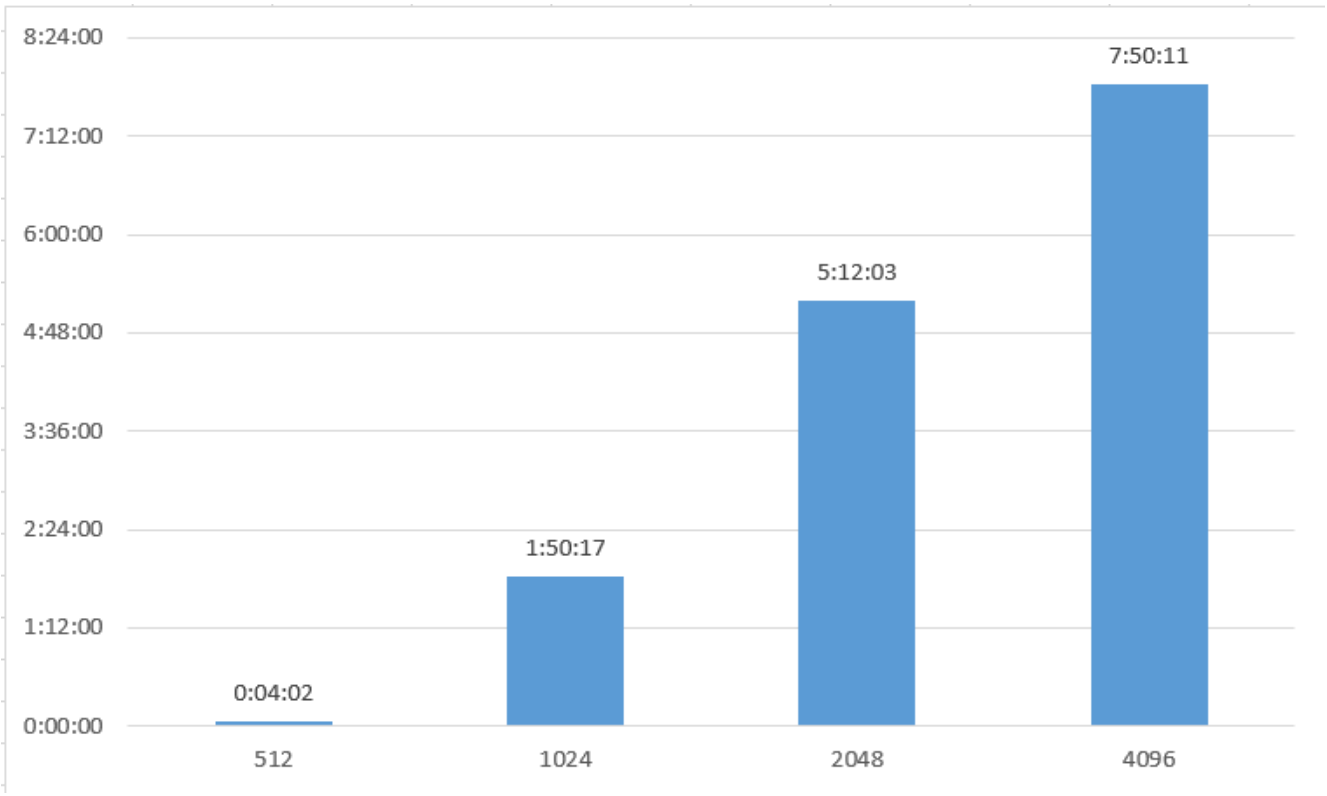

Рисунок 4.10. Час який витрачено на розрахунок до оптимізації коду. У годинах.

1. Час розрахунку для матриці 500 на 500;
2. Час розрахунку для матриці 700 на 700;
3. Час розрахунку для матриці 1 000 на 1 000.

Висновки

У даному розділі розглянуто практичну реалізацію SVD-розкладу для трьохдіагональної матриці. Наведено лістінг коду, дано пояснення щодо використаних бібліотек та представлено роз`яснення шодо описаних у коді функцій.

Також наведено порівняння розрахунків на CPU та GPU у розрізі витраченого часу.

Висновок до дипломної роботи

Під час роботи над дипломною роботою були проведені дослідження, щодо принципу та логіки розкладання матриць методом SVD. Було обрано мову програмування та необхідні модулі. Після проектування було розроблемо алгоритм SVD-розкладу трьохдіагональної матриці. Під час розробки були створені: діючий алгоритм SVD-розкладу на мові Python та оптимізовано швидкість його роботи максимально наближено до рівня MATLAB. Реалізовано можливість розрахунку на GPU. Роботу програми було протестовано. Розрахунок проводиться правильно та досить швидко. У майбутньому є можливість інтеграції алгоритму у великі додатки, а також збільшення швидкості виконання.

Список літератури

1. Basics of Linear Algebra for Machine Learning. Discover the Mathematical Language of Data in Python / Jason Brownlee;
2. GPU+Python Параллельные вычисления в рамках языка Python / В. А. Антонюк
3. SVD-разложение и его практические приложения / Колесников Е. В.
4. Data-Driven Science and Engineering / Steven L. Brunton, J. Nathan Kutz
5. МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ / Дж. Голуб, Ч. Ван Лоун
6. ПРИМЕНЕНИЕ ОСНОВНЫХ МАТРИЧНЫХ РАЗЛОЖЕНИЙ В ЗАДАЧАХ МЕХАНИКИ И РОБОТОТЕХНИКИ / Б. И. Адамов, А. Н. Маслов, Н. В. Осадченко
7. Как уменьшить количество измерений и извлечь из этого пользу[Електронний ресурс]:https://m.habr.com/ru/post/275273/
8. Professor SVD [Електронний ресурс]: https://www.mathworks.com/company/newsletters/articles/professor-svd.html
9. Singular Value Decomposition Example In Python [Електронний ресурс]: https://towardsdatascience.com/singular-value-decomposition-example-in-python-dab2507d85a0
10. Параллельные вычисления на Python за 60 секунд [Електронний ресурс]: https://lambda-it.ru/post/parallelnye-vychisleniia-na-python-za-60-sekund
11. Рекурсия и рекурсивные алгоритмы [Електронний ресурс]: https://www.intuit.ru/studies/courses/648/504/lecture/11462
12. Сингулярное разложение матрицы [Електронний ресурс]: http://pmpu.ru/vf4/algebra2/svd
13. Введение в машинное обучение с tensorflow [Електронний ресурс]: https://habr.com/ru/post/326650/

14. How to Calculate the SVD from Scratch with Python [Електронний ресурс]: https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/
15. Machine Learning — Singular Value Decomposition (SVD) & Principal Component Analysis (PCA) [Електронний ресурс]: https://medium.com/@jonathan_hui/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca-1d45e885e491
16. Lecture: The Singular Value Decomposition (SVD) [Електронний ресурс]: https://www.youtube.com/watch?v=EokL7E6o1AE&t=13s
17. Svd [Електронний ресурс]: https://www.mathworks.com/help/matlab/ref/double.svd.html
18. Python для математических вычислений [Електронний ресурс]: https://habr.com/ru/post/312268/
19. Matlab vs. Julia vs. Python [Електронний ресурс]: https://habr.com/ru/company/edison/blog/480716/
20. Сравнение производительности GPU-расчетов на Python и C [Електронний ресурс]: https://habr.com/ru/post/317328/

Додаток А

Текст програми «Рекурсивний алгоритм SVD-розкладу трохдіагональної матриці на графічному процесорі»

```
from datetime import datetime

#from random import normalvariate

from tqdm import tqdm

import PySimpleGUI as sg

import numpy as np

from numpy.linalg import matrix_rank

import tensorflow as tf

#from numpy.linalg import norm

from scipy.sparse import dia_matrix


TYPE = float


ERROR_VAL = 1e-10

DIMENSION = 5

DEVICE_NAME = "cpu"

SPACE_SIZE = 2


def tridiag(n):
```

```
# generate triadiagonal matrix


    #a = []

    #b = []

    #c = []


    #a += [int(input('Введите число a[%d] = '%(i+1))) for i in range(0, n)]

    #b += [int(input('Введите число b[%d] = '%(i+1))) for i in range(0, n)]

    #c += [int(input('Введите число c[%d] = '%(i+1))) for i in range(0, n)]


    #diag_matrix = dia_matrix((np.array([a, b, c]), [0,1,-1]), shape=(n, n)).toarray()


    #return diag_matrix


    return np.diag(np.random.rand(n - 1), -1) \
    + np.diag(np.random.rand(n), 0) \
    + np.diag(np.random.rand(n - 1), 1)



def printMatrix(matrix):
  for row in matrix:
```

```
    for x in row:
      print("{:.3f}".format(x), end = "  ")
    print()



def find_rank(A):
    r = matrix_rank(A)

    return r



def rsvd(A, r, n_oversamples=None, n_subspace_iters=None,
         return_range=False):

    if n_oversamples is None:
        n_samples = 2 * r
    else:
        n_samples = r + n_oversamples

    Q = find_range(A, n_samples, n_subspace_iters)
```

```
    for i in range(1, r, int(r/1.5)):
        Q = find_range(A, n_samples, n_subspace_iters)
        print("\n" * SPACE_SIZE)
        print("Approximate Matrix A Range (r = " + str(i) + "): ")

        # Stage B.
        B = Q.T @ A
        U_tilde, S, Vt = np.linalg.svd(B)
        U = Q @ U_tilde

        Sigma_d = np.diag(S)

        print("\n" * SPACE_SIZE)
        print("При ранге r = " + str(i) + ":\n Приближенный диапазон матрицы A:\n")
        printMatrix(Q)
        print("\n" * SPACE_SIZE)
        print("Второй этап нахождения приближенного решения A за формулой(B =
Q.T * A):\n")
        printMatrix(B)
        print("\n" * SPACE_SIZE)
        print("Промежуточные результаты разложения:\n")
        print("Матрица U_tilde:\n")
```

```
        printMatrix(U_tilde)
        print("\n" * SPACE_SIZE)
        print("Матрица S:\n")
        printMatrix(Sigma_d)
        print("\n" * SPACE_SIZE)
        print("Матрица V_transpose:\n")
        printMatrix(Vt)
        print("\n" * SPACE_SIZE)

        U, S, Vt = U[:, :r], S[:r], Vt[:r, :]

    if return_range:
        return U, S, Vt, Q
    return U, S, Vt


def find_range(A, n_samples, n_subspace_iters=None):
    m, n = A.shape
    O = np.random.randn(n, n_samples)
    Y = A @ O
```

```
    if n_subspace_iters:
        return subspace_iter(A, Y, n_subspace_iters)
    else:
        return ortho_basis(Y)


def subspace_iter(A, Y0, n_iters):
    Q = ortho_basis(Y0)
    for _ in range(n_iters):
        Z = ortho_basis(A.T @ Q)
        Q = ortho_basis(A @ Z)
    return Q


def ortho_basis(M):
    Q, _ = np.linalg.qr(M)
    return Q



def start(dimension=DIMENSION, device_name=DEVICE_NAME,
space_size=SPACE_SIZE, error=ERROR_VAL, verbose=True):
    if device_name == "gpu":
        device_name = "/gpu:0"
    else:
```

```
        device_name = "/cpu:0"


    A = tridiag(dimension)
    if verbose:
        print("\n" * space_size)
        print("Исходная матрица A:")
        printMatrix(A)
        print("\n" * space_size)


    start_time = datetime.now()
    with
tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True)):
        with tf.device(device_name):
            U, S, Vt = rsvd(A, find_rank(A))
            if verbose:
                Sigma = np.diag(S)
                print("\n" * space_size)
                print("Диагональная матрица S rxr:\n")
                printMatrix(Sigma)
                print("\n" * space_size)
                print("Ортогональная матрица U:\n")
                printMatrix(U)
```

```
            print("\n" * space_size)
            print("Ортогональная матрица транспонированная Vt:\n")
            printMatrix(Vt)

    print("\n" * space_size)
    print("Размерность:", (dimension, dimension), "Девайс:", device_name)
    print("Время:", datetime.now() - start_time)
    print("\n" * space_size)


if __name__ == '__main__':
    sg.theme('DarkAmber')
    layout = [[sg.Image(r'fi-xnsuxl-tensorflow.png')],
            [sg.Text('Singular Value Decomposition')],
            [sg.Text('Please enter shape of the input matrix A')],
            [sg.InputText()],
            [sg.Text('Accuracy')],
            [sg.InputText()],
            [sg.Text('Space size')],
            [sg.InputText()],
            [sg.Frame(layout=[
```

```
            [sg.Radio('GPU', "device", size=(10, 2), default=True),
             sg.Radio('CPU', "device")],
          ], title='Choose CPU or GPU (Graphics card must be present and supported by Tensorflow)',
             relief=sg.RELIEF_SUNKEN)],
          [sg.Checkbox('Verbose', size=(10, 1), default=True)],
          [sg.Button('Start'), sg.Button('Exit')]]
  window = sg.Window('SVD', layout)

  while True:
    event, values = window.read()
    if event in (None, 'Exit'):  # if user closes window or clicks exit
      break
    else:
      if values[4]:
        device = 'gpu'
      elif values[5]:
        device = 'cpu'
      else:
        device = 'unknown'
      start(dimension=int(values[1]), error=float(values[2]), device_name=device,
          space_size=int(values[3]), verbose=values[6])
```

```
window.close()
```

Додаток Б

Проміжні результати розрахунку матриці 16 на 16

Исходная матрица A:

0.89 0.68 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.13 0.51 0.95 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.94 0.49 0.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.94 0.81 0.63 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.47 0.28 0.33 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.55 0.81 0.66 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.03 0.45 0.99 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.08 0.19 0.47 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.69 0.69 0.13 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.83 0.98 0.27 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.12 0.10 0.15 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.85 0.46 0.51 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.75 0.88 0.65 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.35 0.89 0.99 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.76 0.77 0.22

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.22 0.40

Device mapping:

Approximate Matrix A Range (r = 1):

При ранге r = 1:

Приближенный диапазон матрицы A:

-0.17 0.42 -0.21 -0.01 -0.32 -0.01 0.56 -0.12 -0.05 0.10 -0.35 -0.09 -0.37 0.10 -0.07 0.15

-0.25 -0.43 -0.19 -0.36 0.03 0.55 0.19 0.10 -0.15 -0.03 -0.24 0.09 0.09 0.08 0.34 -0.15

0.41 0.18 -0.42 -0.14 -0.24 0.08 0.31 0.03 -0.03 -0.04 0.32 0.21 0.49 -0.21 -0.10 -0.07

0.62 -0.37 -0.22 -0.28 0.01 -0.05 -0.10 0.10 0.10 0.09 -0.21 -0.01 -0.43 0.07 -0.23 0.16

0.34 0.05 -0.15 0.20 -0.14 -0.15 -0.04 -0.06 -0.02 -0.09 -0.00 -0.43 -0.04 0.14 0.72 -0.22

0.06 -0.49 -0.10 0.65 0.15 -0.02 0.35 -0.27 -0.17 -0.09 -0.10 -0.03 0.07 -0.16 -0.16 0.04

0.14 0.21 -0.09 0.27 -0.12 0.14 -0.33 -0.04 -0.33 -0.35 -0.21 0.58 -0.09 0.29 0.04 0.00

0.11 0.02 0.05 -0.05 0.18 0.16 0.04 -0.44 0.07 0.26 0.14 -0.03 0.27 0.62 0.04 0.43

0.18 0.26 0.02 -0.00 0.10 0.38 -0.29 -0.25 -0.30 0.27 -0.33 -0.36 0.15 -0.23 -0.25 -0.25

0.25 0.32 0.15 0.07 0.63 0.32 0.30 0.18 0.28 -0.26 -0.04 0.04 -0.14 -0.07 0.09 -0.02

-0.01 0.05 -0.05 0.11 -0.03 0.03 -0.10 0.52 -0.19 -0.10 -0.22 -0.27 0.29 -0.10 0.06 0.66

-0.14 -0.02 -0.32 0.29 -0.24 0.31 -0.24 0.03 0.70 -0.10 -0.11 -0.13 0.08 0.11 -0.17 -0.07

0.13 -0.02 0.23 0.36 -0.20 0.27 0.09 0.42 -0.03 0.63 0.17 0.18 -0.12 0.08 0.08 -0.10

0.21 -0.06 0.53 -0.07 -0.35 0.06 0.05 -0.28 0.29 -0.04 -0.32 0.22 0.15 -0.34 0.23 0.19

0.14 -0.09 0.43 -0.06 -0.30 0.11 0.20 0.18 -0.12 -0.41 0.06 -0.31 0.10 0.39 -0.34 -0.22

-0.03 -0.01 0.01 0.01 -0.19 0.44 -0.10 -0.21 -0.14 -0.21 0.54 -0.15 -0.41 -0.26 0.05 0.30

Второй этап нахождения приближенного решения A за формулой(B = Q.T * A):

-0.19 0.14 0.55 1.05 0.52 0.17 0.11 0.28 0.39 0.27 -0.06 0.03 0.12 0.38 0.31 0.02

0.32 0.23 -0.67 -0.11 -0.48 -0.37 -0.23 0.39 0.46 0.36 0.08 -0.02 -0.05 -0.15 -0.14 -0.03

-0.21 -0.63 -0.59 -0.64 -0.23 -0.14 -0.11 -0.07 0.16 0.15 -0.24 0.02 0.23 0.96 0.86 0.10

-0.05 -0.31 -0.67 -0.26 0.23 0.60 0.55 0.26 0.04 0.09 0.28 0.42 0.44 0.13 -0.11 -0.01

-0.28 -0.43 -0.09 -0.28 0.05 0.08 0.06 -0.01 0.67 0.63 -0.03 -0.26 -0.42 -0.67 -0.62 -0.14

0.06 0.35 0.51 -0.04 -0.09 -0.06 0.06 0.43 0.60 0.36 0.35 0.35 0.41 0.31 0.24 0.20

0.53 0.77 0.24 0.19 0.12 0.26 0.09 -0.53 0.06 0.24 -0.14 -0.06 -0.02 0.26 0.18 0.00

-0.10 -0.00 0.20 0.08 -0.10 -0.24 -0.23 -0.30 -0.23 0.21 0.13 0.41 0.29 0.16 -0.19 -0.05

-0.07 -0.14 -0.07 0.04 -0.04 -0.15 -0.26 -0.53 0.05 0.21 0.66 0.28 0.43 0.15 0.17 -0.08

0.08 0.01 0.03 -0.00 -0.02 -0.11 -0.20 -0.11 0.09 -0.24 -0.17 0.41 0.49 0.06 -0.40 -0.17

-0.35 -0.07 -0.27 0.13 -0.19 -0.09 -0.15 -0.41 -0.19 -0.10 -0.12 0.04 -0.02 -0.13 -0.15 0.23

-0.07 0.19 0.18 -0.01 -0.14 -0.15 0.24 0.32 -0.23 -0.04 -0.12 0.04 0.17 0.07 -0.06 -0.13

-0.32 0.25 -0.07 0.09 -0.25 0.04 0.02 0.06 0.11 -0.08 0.06 -0.01 -0.02 0.14 0.14 -0.14

0.10 -0.08 0.04 -0.07 -0.00 -0.08 0.07 0.25 0.08 -0.11 0.07 0.10 0.01 0.05 -0.09 -0.02

-0.02 0.04 0.06 0.06 -0.03 0.10 -0.09 -0.12 -0.08 0.06 -0.11 -0.01 0.06 -0.01 -0.02 -0.05

0.12 -0.04 -0.02 -0.03 0.06 -0.04 0.06 -0.09 0.01 0.03 0.01 -0.01 -0.06 -0.07 0.09 0.08

Промежуточные результаты разложения:

Матрица U_tilde:

0.56 0.25 -0.33 -0.31 -0.09 0.35 -0.16 0.44 0.16 -0.17 -0.12 -0.02 -0.06 -0.03 -0.01 0.01

-0.23 -0.09 -0.38 0.38 0.45 -0.29 0.01 0.53 0.01 -0.20 -0.18 0.02 0.02 0.06 0.01 0.01

-0.55 0.62 0.01 -0.24 0.31 0.25 -0.03 -0.09 0.28 0.01 -0.06 -0.02 -0.03 -0.03 -0.03 0.00

-0.30 0.18 -0.30 -0.16 -0.67 -0.30 0.40 0.21 -0.00 0.11 -0.09 0.09 0.06 -0.00 0.03 -0.00

-0.21 -0.52 -0.51 0.01 -0.02 0.48 0.12 -0.21 0.25 0.19 -0.06 -0.11 -0.09 -0.07 -0.04 0.01

0.22 0.34 -0.53 0.27 0.07 -0.11 -0.13 -0.38 -0.12 0.23 0.27 0.38 0.02 0.13 -0.00 -0.01

0.36 0.12 0.10 0.12 0.28 -0.07 0.77 -0.05 0.28 0.15 -0.06 -0.03 -0.07 -0.18 -0.14 0.01

0.05 0.12 0.13 0.44 -0.18 0.19 -0.14 -0.06 0.15 0.13 -0.54 0.15 0.44 -0.27 0.24 -0.07

-0.06 0.26 0.03 0.50 -0.21 0.40 0.15 0.05 -0.44 -0.13 0.01 -0.33 -0.30 0.08 -0.18 0.06

0.01 0.05 0.07 0.37 -0.27 -0.13 -0.17 0.02 0.69 -0.18 0.39 -0.19 -0.18 0.04 0.09 -0.07

-0.11 -0.09 0.26 0.12 -0.05 0.23 -0.06 0.41 0.06 0.35 0.19 0.61 -0.23 -0.09 -0.24 0.11

0.08 0.04 0.00 -0.02 -0.03 -0.31 -0.28 -0.10 0.09 0.38 -0.52 -0.17 -0.57 0.08 -0.16 0.03

0.02 0.08 -0.04 0.00 0.10 -0.03 -0.10 0.27 -0.12 0.62 0.32 -0.47 0.16 -0.24 0.28 0.10

-0.02 0.01 -0.10 0.00 -0.05 -0.17 -0.15 -0.12 0.00 -0.21 0.05 -0.05 0.09 -0.59 -0.41 0.59

0.04 -0.01 0.07 0.03 -0.01 0.06 0.05 0.02 0.16 0.12 -0.09 -0.06 0.28 0.64 -0.04 0.67

-0.01 -0.01 0.01 -0.02 0.04 0.03 0.11 -0.07 -0.07 -0.16 -0.03 0.19 -0.42 -0.15 0.74 0.41

Матрица S:

2.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 1.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 1.58 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 1.37 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 1.33 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 1.20 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 1.08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.82 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.74 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.61 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.46 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.42 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.21 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.20 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.08 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.06

Матрица V_transpose:

0.12 0.47 0.62 0.57 0.24 0.07 0.04 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

-0.00 -0.00 -0.00 -0.00 0.00 0.00 0.00 0.01 0.02 0.02 0.10 0.25 0.41 0.67 0.55 0.07

0.01 0.03 0.02 0.01 -0.05 -0.08 -0.17 -0.50 -0.67 -0.46 -0.19 -0.06 -0.05 0.05 0.07 0.01

0.17 0.26 0.02 -0.06 -0.31 -0.35 -0.36 -0.30 0.09 0.17 0.32 0.34 0.32 -0.12 -0.30 -0.06

0.22 0.31 -0.01 -0.10 -0.33 -0.33 -0.25 0.03 0.21 0.14 -0.25 -0.39 -0.38 0.11 0.35 0.07

-0.41 -0.42 0.17 0.20 0.16 -0.06 -0.26 -0.52 0.21 0.34 0.05 -0.13 -0.13 0.01 0.12 0.03

0.41 0.26 -0.29 -0.13 0.20 0.48 0.21 -0.50 0.07 0.27 0.10 -0.07 -0.07 -0.01 0.07 0.02

-0.16 0.16 -0.67 0.69 -0.09 0.01 -0.11 0.05 0.02 -0.03 -0.04 0.01 0.01 0.01 -0.02 -0.01

0.07 -0.03 0.01 -0.03 0.07 0.04 -0.07 -0.11 0.23 0.06 -0.80 0.15 0.19 0.24 -0.36 -0.15

-0.66 0.42 0.09 -0.17 -0.42 0.22 0.30 -0.10 -0.07 0.15 -0.08 0.01 0.01 0.04 -0.04 -0.03

-0.13 0.11 0.02 -0.10 -0.01 0.25 -0.21 -0.14 0.57 -0.67 0.06 0.01 0.04 -0.11 0.07 0.19

0.02 -0.01 -0.00 0.01 -0.00 -0.04 0.04 0.02 -0.09 0.15 -0.17 0.25 -0.03 -0.11 -0.04 0.93

-0.00 -0.01 0.06 -0.05 -0.09 0.49 -0.56 0.22 -0.17 0.10 0.10 0.26 -0.41 0.26 -0.15 -0.03

0.01 -0.02 0.05 -0.03 -0.08 0.34 -0.38 0.16 -0.15 0.19 -0.16 -0.31 0.55 -0.40 0.25 0.02

-0.08 0.11 -0.05 -0.08 0.18 -0.09 -0.03 0.01 0.00 0.05 -0.22 0.60 -0.19 -0.44 0.48 -0.25

0.30 -0.39 0.18 0.29 -0.66 0.22 0.26 -0.11 0.08 -0.05 -0.06 0.17 -0.06 -0.11 0.13 -0.07

Approximate Matrix A Range (r = 11):

При ранге r = 11:

Приближенный диапазон матрицы A:

-0.07 -0.01 -0.31 0.11 -0.09 -0.10 -0.13 0.32 -0.21 -0.58 -0.34 -0.01 -0.13 0.48 -0.12 0.02

0.01 -0.23 -0.12 -0.20 -0.01 -0.13 -0.28 0.36 0.24 0.12 0.17 -0.46 0.52 0.09 -0.20 -0.20

0.23 0.01 -0.03 -0.16 -0.65 -0.05 -0.36 -0.14 0.41 -0.15 0.13 -0.03 -0.31 -0.05 0.14 0.13

-0.29 -0.23 -0.40 -0.50 -0.10 0.41 0.40 0.15 0.11 0.15 -0.03 -0.01 -0.20 0.05 0.06 0.09

-0.03 0.03 -0.07 -0.33 -0.15 0.04 0.02 -0.42 -0.37 -0.29 0.37 0.11 0.11 0.01 -0.36 -0.41

-0.09 0.14 0.35 -0.62 -0.07 -0.31 -0.09 -0.03 -0.31 0.08 -0.42 -0.12 0.09 -0.02 0.16 0.18

-0.48 0.39 0.01 0.07 -0.24 0.18 -0.34 0.39 -0.08 0.17 0.03 0.36 0.05 -0.24 -0.12 -0.11

-0.06 0.26 -0.11 0.10 -0.03 0.03 0.05 -0.22 0.19 0.15 -0.49 -0.41 -0.27 -0.16 -0.28 -0.46

-0.28 0.44 -0.36 0.12 -0.03 0.07 -0.05 -0.41 0.02 0.01 0.09 -0.26 0.36 0.19 0.28 0.30

-0.07 0.29 -0.18 -0.24 0.32 -0.44 0.21 0.08 0.46 -0.34 0.07 0.24 0.02 -0.27 -0.08 0.03

0.02 -0.01 0.03 -0.07 0.05 0.07 -0.04 -0.03 0.19 -0.04 -0.14 0.28 0.17 0.24 0.62 -0.61

0.35 0.47 0.33 -0.20 0.16 0.55 0.06 0.18 0.15 -0.20 0.05 -0.12 0.05 0.20 -0.13 0.07

0.26 -0.02 -0.40 -0.21 0.49 0.14 -0.59 -0.11 -0.13 0.15 -0.05 0.12 -0.20 -0.09 0.02 0.05

0.45 0.33 -0.33 0.01 -0.18 -0.19 0.26 0.34 -0.37 0.17 0.19 -0.14 -0.07 -0.16 0.24 -0.15

0.37 -0.02 -0.22 0.01 -0.26 0.05 0.13 -0.11 0.09 0.20 -0.40 0.44 0.45 0.04 -0.30 0.13

0.04 -0.20 -0.05 0.06 -0.04 0.31 -0.02 0.02 -0.12 -0.47 -0.22 -0.16 0.27 -0.66 0.21 0.03

Второй этап нахождения приближенного решения A за формулой(B = Q.T * A):

-0.07 0.17 -0.15 -0.04 -0.25 -0.10 -0.28 -0.68 -0.28 -0.10 0.28 0.36 0.57 0.86 0.74 0.10

-0.03 -0.11 -0.43 -0.17 -0.06 0.14 0.29 0.75 0.67 0.34 0.48 0.20 0.34 0.27 0.27 -0.08

-0.29 -0.30 -0.50 -0.38 -0.08 0.26 0.23 -0.26 -0.46 -0.22 0.24 -0.14 -0.30 -0.72 -0.50 -0.07

0.07 -0.18 -0.74 -0.71 -0.75 -0.61 -0.37 0.17 -0.07 -0.23 -0.24 -0.26 -0.28 -0.12 0.03 0.03

-0.08 -0.67 -0.42 -0.76 -0.14 -0.12 -0.16 -0.26 0.23 0.32 0.23 0.44 0.45 -0.05 -0.39 -0.07

-0.11 -0.18 0.24 0.31 0.10 -0.24 -0.13 0.23 -0.31 -0.42 0.36 0.37 0.33 -0.05 -0.09 0.13

-0.15 -0.57 -0.07 -0.01 0.21 -0.08 -0.21 -0.36 0.16 0.19 0.10 -0.42 -0.41 -0.06 0.35 0.02

0.33 0.27 0.41 -0.21 -0.04 -0.15 0.14 0.06 -0.32 0.02 0.17 -0.00 0.11 0.15 0.26 -0.01

-0.16 0.36 0.53 0.29 -0.20 -0.37 -0.22 -0.03 0.49 0.48 0.27 0.00 -0.17 -0.34 -0.32 -0.03

-0.50 -0.48 0.18 -0.16 0.06 -0.02 0.14 0.20 -0.20 -0.33 -0.26 0.02 0.10 0.40 0.22 -0.15

-0.28 -0.02 0.20 0.27 -0.14 -0.22 -0.30 -0.00 -0.11 0.06 0.04 -0.04 0.04 -0.18 -0.17 -0.17

-0.07 -0.27 -0.46 0.02 -0.04 -0.05 0.05 0.09 -0.17 0.23 -0.01 0.07 -0.01 0.29 0.16 0.03

-0.05 -0.11 0.15 -0.40 -0.05 0.11 0.06 0.25 0.14 0.09 0.07 -0.10 -0.18 0.15 0.34 0.21

0.44 0.33 0.11 -0.00 0.02 -0.02 -0.13 -0.14 -0.17 -0.21 0.12 0.06 -0.04 -0.17 -0.27 -0.26

-0.13 -0.05 -0.06 0.01 0.03 0.01 0.03 0.02 -0.01 0.03 -0.07 0.04 0.04 -0.00 0.06 0.02

-0.01 0.03 -0.04 0.00 0.04 0.00 0.03 0.01 0.02 -0.01 0.00 -0.02 0.03 0.00 -0.04 0.04

Промежуточные результаты разложения:

Матрица U_tilde:

-0.06 0.69 0.40 -0.34 0.05 0.18 0.20 0.15 -0.19 0.27 -0.18 0.06 -0.03 0.04 -0.01 0.01

-0.20 0.30 -0.72 0.03 -0.09 -0.24 0.01 0.20 -0.31 0.01 -0.09 0.36 0.09 -0.05 0.00 -0.01

-0.35 -0.47 0.23 0.06 -0.39 -0.04 0.20 0.04 -0.49 0.41 -0.01 0.07 0.03 0.07 -0.01 -0.02

-0.58 -0.14 0.17 -0.17 0.60 -0.28 -0.33 0.11 -0.03 -0.06 -0.09 -0.01 0.10 0.03 0.02 -0.00

-0.53 0.04 -0.18 -0.47 -0.36 0.21 0.06 -0.38 0.33 -0.14 0.07 -0.07 -0.00 -0.07 -0.01 0.01

0.11 0.09 0.13 -0.16 -0.41 -0.10 -0.55 0.07 -0.39 -0.42 -0.14 -0.30 0.00 0.10 0.03 -0.01

-0.15 -0.05 0.03 0.24 0.20 0.68 0.10 -0.07 -0.33 -0.47 -0.03 0.22 0.14 0.02 0.03 0.03

0.14 0.15 0.11 -0.08 0.12 -0.27 0.03 -0.56 -0.32 -0.01 0.53 0.08 0.36 -0.13 0.00 -0.01

0.28 -0.22 -0.33 -0.47 0.23 0.32 -0.21 -0.07 -0.15 0.34 0.04 -0.03 0.04 0.45 0.05 0.02

-0.12 0.20 0.14 0.39 -0.18 0.09 -0.52 -0.26 0.23 0.26 -0.01 0.40 0.05 0.34 -0.01 0.03

0.09 -0.11 0.06 -0.19 -0.01 0.21 -0.39 0.16 -0.05 0.14 0.23 0.32 -0.28 -0.66 -0.11 0.08

-0.21 0.15 -0.02 0.11 -0.01 0.05 0.02 0.34 -0.03 -0.07 0.74 -0.16 -0.34 0.32 -0.01 0.05

-0.10 0.11 -0.15 0.22 0.19 -0.00 -0.02 -0.47 -0.26 0.14 -0.18 -0.30 -0.64 -0.10 0.02 0.13

0.13 -0.14 0.17 -0.26 -0.01 -0.27 0.18 -0.10 -0.00 -0.33 -0.06 0.56 -0.42 0.27 0.19 0.21

-0.03 0.03 -0.01 0.06 -0.04 0.04 -0.04 0.07 0.04 0.10 0.04 -0.10 0.14 -0.14 0.87 0.40

-0.00 -0.01 -0.01 0.01 -0.02 -0.02 0.03 0.04 0.01 0.01 -0.03 -0.07 0.17 0.04 -0.44 0.88

Матрица S:

2.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 1.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 1.58 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 1.37 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 1.33 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 1.20 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 1.08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.82 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.74 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.61 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.46 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.42 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.21 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.20 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.08 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.06

Матрица V_transpose:

0.12 0.47 0.62 0.57 0.24 0.07 0.04 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

-0.00 -0.00 -0.00 -0.00 0.00 0.00 0.00 0.01 0.02 0.02 0.10 0.25 0.41 0.67 0.55 0.07

0.01 0.03 0.02 0.01 -0.05 -0.08 -0.17 -0.50 -0.67 -0.46 -0.19 -0.06 -0.05 0.05 0.07 0.01

-0.17 -0.26 -0.02 0.06 0.31 0.35 0.36 0.30 -0.09 -0.17 -0.32 -0.34 -0.32 0.12 0.30 0.06

0.22 0.31 -0.01 -0.10 -0.33 -0.33 -0.25 0.03 0.21 0.14 -0.25 -0.39 -0.38 0.11 0.35 0.07

-0.41 -0.42 0.17 0.20 0.16 -0.06 -0.26 -0.52 0.21 0.34 0.05 -0.13 -0.13 0.01 0.12 0.03

0.41 0.26 -0.29 -0.13 0.20 0.48 0.21 -0.50 0.07 0.27 0.10 -0.07 -0.07 -0.01 0.07 0.02

-0.16 0.16 -0.67 0.69 -0.09 0.01 -0.11 0.05 0.02 -0.03 -0.04 0.01 0.01 0.01 -0.02 -0.01

0.07 -0.03 0.01 -0.03 0.07 0.04 -0.07 -0.11 0.23 0.06 -0.80 0.15 0.19 0.24 -0.36 -0.15

-0.66 0.42 0.09 -0.17 -0.42 0.22 0.30 -0.10 -0.07 0.15 -0.08 0.01 0.01 0.04 -0.04 -0.03

0.13 -0.11 -0.02 0.10 0.01 -0.25 0.21 0.14 -0.57 0.67 -0.06 -0.01 -0.04 0.11 -0.07 -0.19

-0.02 0.01 0.00 -0.01 0.00 0.04 -0.04 -0.02 0.09 -0.15 0.17 -0.25 0.03 0.11 0.04 -0.93

0.00 0.01 -0.06 0.05 0.09 -0.49 0.56 -0.22 0.17 -0.10 -0.10 -0.26 0.41 -0.26 0.15 0.03

-0.01 0.02 -0.05 0.03 0.08 -0.34 0.38 -0.16 0.15 -0.19 0.16 0.31 -0.55 0.40 -0.25 -0.02

-0.08 0.11 -0.05 -0.08 0.18 -0.09 -0.03 0.01 0.00 0.05 -0.22 0.60 -0.19 -0.44 0.48 -0.25

-0.30 0.39 -0.18 -0.29 0.66 -0.22 -0.26 0.11 -0.08 0.05 0.06 -0.17 0.06 0.11 -0.13 0.07

Диагональная матрица S rxr:

2.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 1.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 1.58 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 1.37 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 1.33 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 1.20 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 1.08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.82 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.74 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.61 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.46 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.42 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.21 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.20 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.08 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.06

Ортогональная матрица U:

0.21 -0.00 0.02 -0.24 0.30 -0.54 0.50 -0.05 0.06 -0.50 0.09 -0.01 0.04 0.02 0.00 0.01

0.42 -0.00 0.02 -0.12 0.13 -0.09 -0.08 -0.71 0.01 0.34 -0.13 0.02 -0.26 -0.18 -0.07 -0.18

0.63 -0.00 0.03 -0.14 0.15 -0.10 -0.02 0.56 -0.07 0.45 -0.04 0.00 0.12 0.09 0.05 0.12

0.59 -0.00 -0.01 0.17 -0.22 0.36 -0.23 -0.16 0.04 -0.52 0.15 -0.02 0.19 0.13 0.05 0.11

0.18 -0.00 -0.02 0.17 -0.18 0.10 0.14 0.37 0.02 -0.20 -0.07 0.01 -0.54 -0.39 -0.19 -0.46

0.10 0.00 -0.13 0.50 -0.46 -0.11 0.59 -0.14 0.04 0.24 -0.13 0.02 0.09 0.07 0.07 0.18

0.01 0.00 -0.37 0.34 -0.07 -0.53 -0.36 0.00 -0.19 0.06 0.49 -0.07 0.06 -0.00 -0.11 -0.19

0.00 0.00 -0.27 0.03 0.06 -0.02 -0.04 0.01 0.11 -0.05 -0.49 0.08 0.37 0.35 -0.00 -0.63

0.00 0.01 -0.55 0.09 0.14 -0.14 -0.25 0.05 0.12 -0.17 -0.47 0.07 -0.26 -0.14 0.16 0.46

0.00 0.03 -0.67 -0.24 0.18 0.44 0.32 -0.03 0.04 0.11 0.35 -0.06 0.05 -0.04 -0.13 -0.03

0.00 0.03 -0.05 -0.08 -0.05 0.02 0.03 -0.01 -0.07 0.02 0.16 -0.09 -0.30 0.20 0.88 -0.21

0.00 0.21 -0.14 -0.43 -0.44 -0.07 0.01 -0.03 -0.70 -0.09 -0.17 0.12 -0.00 0.03 -0.09 0.04

0.00 0.51 -0.03 -0.34 -0.42 -0.17 -0.11 0.03 0.60 0.06 0.06 -0.20 -0.01 0.03 -0.05 0.00

0.00 0.67 0.06 0.21 0.24 0.07 0.03 -0.01 -0.09 -0.01 0.03 0.36 0.28 -0.42 0.21 -0.06

0.00 0.49 0.06 0.25 0.28 0.09 0.05 -0.01 -0.17 -0.01 -0.02 -0.19 -0.39 0.57 -0.25 0.07

0.00 0.08 0.01 0.07 0.08 0.03 0.02 -0.01 -0.19 -0.03 -0.20 -0.86 0.22 -0.31 0.08 -0.03

Ортогональная матрица транспонированная Vt:

0.12 0.47 0.62 0.57 0.24 0.07 0.04 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

-0.00 -0.00 -0.00 -0.00 0.00 0.00 0.00 0.01 0.02 0.02 0.10 0.25 0.41 0.67 0.55 0.07

0.01 0.03 0.02 0.01 -0.05 -0.08 -0.17 -0.50 -0.67 -0.46 -0.19 -0.06 -0.05 0.05 0.07 0.01

-0.17 -0.26 -0.02 0.06 0.31 0.35 0.36 0.30 -0.09 -0.17 -0.32 -0.34 -0.32 0.12 0.30 0.06

0.22 0.31 -0.01 -0.10 -0.33 -0.33 -0.25 0.03 0.21 0.14 -0.25 -0.39 -0.38 0.11 0.35 0.07

-0.41 -0.42 0.17 0.20 0.16 -0.06 -0.26 -0.52 0.21 0.34 0.05 -0.13 -0.13 0.01 0.12 0.03

0.41 0.26 -0.29 -0.13 0.20 0.48 0.21 -0.50 0.07 0.27 0.10 -0.07 -0.07 -0.01 0.07 0.02

-0.16 0.16 -0.67 0.69 -0.09 0.01 -0.11 0.05 0.02 -0.03 -0.04 0.01 0.01 0.01 -0.02 -0.01

0.07 -0.03 0.01 -0.03 0.07 0.04 -0.07 -0.11 0.23 0.06 -0.80 0.15 0.19 0.24 -0.36 -0.15

-0.66 0.42 0.09 -0.17 -0.42 0.22 0.30 -0.10 -0.07 0.15 -0.08 0.01 0.01 0.04 -0.04 -0.03

0.13 -0.11 -0.02 0.10 0.01 -0.25 0.21 0.14 -0.57 0.67 -0.06 -0.01 -0.04 0.11 -0.07 -0.19

-0.02 0.01 0.00 -0.01 0.00 0.04 -0.04 -0.02 0.09 -0.15 0.17 -0.25 0.03 0.11 0.04 -0.93

0.00 0.01 -0.06 0.05 0.09 -0.49 0.56 -0.22 0.17 -0.10 -0.10 -0.26 0.41 -0.26 0.15 0.03

-0.01 0.02 -0.05 0.03 0.08 -0.34 0.38 -0.16 0.15 -0.19 0.16 0.31 -0.55 0.40 -0.25 -0.02

-0.08 0.11 -0.05 -0.08 0.18 -0.09 -0.03 0.01 0.00 0.05 -0.22 0.60 -0.19 -0.44 0.48 -0.25

-0.30 0.39 -0.18 -0.29 0.66 -0.22 -0.26 0.11 -0.08 0.05 0.06 -0.17 0.06 0.11 -0.13 0.07

Размерность: (16, 16) Девайс: /gpu:0

Время: 0:00:00.365021

Додаток В

Проміжні результати розрахунку матриці 5 на 5

Исходная матрица A:

0.238 0.347 0.000 0.000 0.000

0.806 0.431 0.355 0.000 0.000

0.000 0.740 0.960 0.530 0.000

0.000 0.000 0.677 0.757 0.403

0.000 0.000 0.000 0.194 0.655

Device mapping:

Approximate Matrix A Range (r = 1):

При ранге r = 1:

Приближенный диапазон матрицы A:

-0.324 -0.100 -0.235 -0.126 0.902

-0.400 -0.169 -0.624 -0.514 -0.397

-0.298 0.741 -0.366 0.474 -0.054

0.705 0.431 -0.302 -0.447 0.160

0.387 -0.475 -0.574 0.543 0.012

Второй этап нахождения приближенного решения A за формулой(B = Q.T * A):

-0.400 -0.505 0.049 0.450 0.537

-0.160 0.441 0.944 0.627 -0.137

-0.559 -0.622 -0.778 -0.534 -0.498

-0.444 0.086 -0.030 0.018 0.175

-0.106 0.102 -0.085 0.095 0.073

Промежуточные результаты разложения:

Матрица U_tilde:

0.034 -0.920 0.279 0.270 0.026

0.659 -0.130 -0.711 0.202 0.055

-0.751 -0.152 -0.606 0.198 0.074

-0.014 -0.324 -0.220 -0.835 -0.386

0.014 -0.090 -0.009 -0.387 0.918

Матрица S:

1.697 0.000 0.000 0.000 0.000

0.000 1.005 0.000 0.000 0.000

0.000 0.000 0.704 0.000 0.000

0.000 0.000 0.000 0.386 0.000

0.000 0.000 0.000 0.000 0.120

Матрица V_transpose:

0.180 0.436 0.711 0.490 0.177

0.624 0.463 -0.032 -0.427 -0.462

0.625 -0.138 -0.253 -0.002 0.725

0.418 -0.728 0.279 0.235 -0.402

0.112 0.215 -0.593 0.723 -0.261

Approximate Matrix A Range (r = 4):

При ранге r = 4:

Приближенный диапазон матрицы A:

-0.202 0.135 -0.316 -0.349 -0.848

-0.886 0.029 0.287 -0.285 0.226

-0.157 -0.836 -0.514 -0.028 0.108

0.294 -0.474 0.663 -0.457 -0.204

0.252 0.241 -0.339 -0.766 0.420

Второй этап нахождения приближенного решения A за формулой(B = Q.T * A):

-0.762 -0.568 -0.267 0.188 0.284

0.055 -0.559 -1.113 -0.755 -0.034

0.156 -0.367 0.057 0.163 0.045

-0.313 -0.265 -0.437 -0.509 -0.686

-0.019 -0.118 0.045 -0.016 0.193

Промежуточные результаты разложения:

Матрица U_tilde:

-0.255 0.936 0.177 0.128 -0.106

-0.826 -0.149 -0.527 0.114 -0.075

-0.002 0.164 -0.237 -0.953 -0.094

-0.503 -0.229 0.774 -0.249 0.181

0.002 0.149 -0.188 -0.024 0.970

Матрица S:

1.697 0.000 0.000 0.000 0.000

0.000 1.005 0.000 0.000 0.000

0.000 0.000 0.704 0.000 0.000

0.000 0.000 0.000 0.386 0.000

0.000 0.000 0.000 0.000 0.120

Матрица V_transpose:

0.180 0.436 0.711 0.490 0.177

-0.624 -0.463 0.032 0.427 0.462

-0.625 0.138 0.253 0.002 -0.725

-0.418 0.728 -0.279 -0.235 0.402

-0.112 -0.215 0.593 -0.723 0.261

Диагональная матрица S rxr:

1.697 0.000 0.000 0.000 0.000

0.000 1.005 0.000 0.000 0.000

0.000 0.000 0.704 0.000 0.000

0.000 0.000 0.000 0.386 0.000

0.000 0.000 0.000 0.000 0.120

Ортогональная матрица U:

0.114 -0.307 -0.143 0.398 -0.845

0.345 -0.687 -0.504 -0.318 0.232

0.746 -0.084 0.492 0.379 0.227

0.544 0.528 -0.170 -0.530 -0.339

0.124 0.383 -0.674 0.563 0.256

Ортогональная матрица транспонированная Vt:

0.180 0.436 0.711 0.490 0.177

-0.624 -0.463 0.032 0.427 0.462

-0.625 0.138 0.253 0.002 -0.725

-0.418 0.728 -0.279 -0.235 0.402

-0.112 -0.215 0.593 -0.723 0.261

Размерность: (5, 5) Девайс: /gpu:0

Время: 0:00:00.014001

Додаток Г

Календарний план виконання роботи

Тема: <u>Рекурсивний алгоритм SVD-розкладу трьохдіагональної матриці на графічному процесорі</u>

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	02.11.2019	
2.	Огляд технічної літератури за темою роботи.	15.11.2019	
3.	Розробка алгоритму рекурсивного SVD-розкладу трьохдіагональної матриці на графічному процесорі.	30.12.2019	
4.	Програмування розробленого алгоритму.	15.01.2019	
5.	Виконання аналізу результатів отриманих за допомогою розробленого алгоритму.	30.03.2019	
6.	Написання пояснювальної роботи.	20.04.2019	
7.	Створення слайдів для доповіді та написання доповіді.	22.04.2019	
8.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи.	25.04.2019	
9.	Корегування роботи за результатами попереднього захисту.	30.05.2019	
10.	Остаточне оформлення пояснювальної роботи та слайдів.	3.06.2019	
11.	Захист магістерської роботи (проекту).	16.06.2019	

Студент: Кулаковський Д. В. _____

Керівник: Малашонок Г. І. _____

“_____”_____