КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТІМЕНІ

ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
зі спеціальності 113 «Прикладна математика»
на тему:

АНАЛІЗ ПОГАНО ОБУМОВЛЕНИХ СЛАР ІЗ ЗАСТОСУВАННЯМ ДОВГОЇ АРИФМЕТИКИ

студента 4 курсу

Ференеця Романа Анатолійовича

Науковий керівник

кандидат фізико-математичних наук, асистент

Оноцький В'ячеслав Валерійович

_____

"____"_____2021р.

Робота заслухана на засіданні кафедри обчислювальної математики та рекомендована до захисту в ДЕК, протокол

Завідувач кафедри обчислювальної математики проф. Ляшко С.І.

Київ-2021

Зміст

1 РЕФЕРАТ

Обсяг роботи 34 сторінки, 10 джерел посилання, 2 ілюстрації, 2 таблиці.
Ключові слова: ПОГАНО ОБУМОВЛЕНІ МАТРИЦІ, АЛГОРИТМ ГАУСА, АРИФМЕТИКА З ПЛАВАЮЧТОЮ КОМОЮ, МІНІМІЗАЦІЯ ПОМИЛОК ПРИ ОБЧИСЛЕНІ.

Метою роботи виступає ідея написання програмного засобу розв'язування систем алгебраїчних лінійних рівнянь, використовуючи алгоритми Гауса з вибором найбільшого елемента, класу чисел зі змінною довжиною мантиси для погано обумовлених матриць, оцінка та аналізування результатів на основі проведеного тестування для поставленої задачі.

Об'єктом роботи є процес розв'язування системи алгебраїчних лінійних рівнянь за допомогою власноруч розробленого програмного засобу.

Предметом роботи є розроблена програмна реалізація для знаходження розв'язку системи алгебраїчних лінійних рівнянь.

Методи розробки: порівняльний аналіз, комп'ютерне моделювання, розробка програмного продукту. Інструменти написання програми: мова програмування Python, середовище розробки VS Code та Jupyter Notebook.

Результати роботи: розроблено реалізацію метода Гауса з вибором найбільшого елемента з використання довгої арифметики, з можливість змінювати довжину мантиси, проведено тестування реалізації на погано обумовлених задачах, з подальшою оцінкою і аналізом результатів.

2 ВСТУП

Відомо, що математичне моделювання процесів різної природи призводить до необхідності досліджувати нелінійні рівняння та системи різної складності (математичні моделі).У багатьох випадках вони розв'язуються введенням певних спрощень, а саме, переходу до різницевих аналогів та , врешті-решт, до систем лінійних алгебраїчних рівнянь (СЛАР) різних розмірностей, частіше за все з квадратною матрицею коефіцієнтів.Через це дослідження та створення алгоритмів для розв'язання СЛАР є фундаментальною частиною математики та стало роботою багатьох науковців. Це, в свою чергу, призвело до появи та розвитку десятків явних більше декількох сотень неявних алгоритмів. З іншого боку, через те що алгоритм розв'язання СЛАР має визначальну роль у пошуку розв'язку для змодельованої задачі, на такий алгоритм повинні накладатися строгі вимоги щодо його точності та швидкодії.

Ці вимоги ще більше впливають на точність розв'язку та якості результатів дослідження, коли система рівнянь є погано обумовленою.У разі існування властивості поганої обумовленості або некоректності - це призводить до проблем на різних етапах моделювання. Специфічні проблеми на етапі комп'ютерного представлення моделі - округлення, усікання, обмежена довжина мантиси викликає накопичення помилок, неточностей представлення моделі, тощо. Це породжує неадекватність результатів досліджуваного процесу та моделі. Слід зазначити, що у таких ситуаціях навіть незначні неточності в поданні моделі, часто можуть призвести до значних відхилень отриманого рішення від справжнього в математичному розумінні. Саме тому зараз широко розвиваються та використовуються алгоритми знаходження розв'язку для СЛАР, які включають у себе технології, що запобігають накопиченню помилок на певних етапах моделювання, зокрема під час ітераційного процесу

алгоритму.

На даний час було розроблено досить багато явних та ітераційних методів вирішення систем лінійних алгебраїчних рівнянь, проте можна констатувати, що проблема розробки універсального високоточного методу розв'язування широкого класу лінійних задач досі не вирішена. Це можна пояснити об'єктивною складністю проблем, що виникають, особливо при моделюванні процесів різного природного характеру. І навіть зараз, коли довжина машинного слова у більшості процессорів була збільшена до 64 бітів, разом зі збільшенням швидкості виконання операцій, все ще існує широкий клас проблем для яких даної точності недостатньо для отримання коректних результатів - ці проблеми залишаються відкритими. У зв'язку з цим, для різних видів невизначеності чи структури матриці обмежень існує необхідність адаптувати алгоритм знаходження розв'язку для забезпечення кращої якості результатів. Щоб наголосити на важливості точності обчислень можна згадати деякі, доволі відомі факти вагомих втрат при використанні неточних результатів та некоректних алгоритмів.З деякою долею впевненості можна розраховувати, що з розвитком квантових та космічних технологій ці проблеми стануть ще більш актуальними.

3 ПОСТАНОВКА ЗАДАЧІ

Основною метою даної роботи є дослідження впливу використання цих технологій, а саме довгої арифметики, на швидкість та якість результатів роботи деяких алгоритмів. Запропонована реалізація алгоритмів буде використовуватись на матрицях з великим числом обумовленості, що також відомі як погано обумовлені матриці. Дані матриці виникають у багатьох прикладних задачах, зокрема матриця Гільберта - вона виникає при апроксимації функцій поліномами методом найменших квадратів.

4 ТЕОРЕТИЧНІ ВІДОМОСТІ

4.1 Представлення чисел у пам'яті комп'ютера

У цій частині розглядається обробка та представлення цілих чисел і чисел із плаваючою точкою в пам'яті комп'ютері. Цілі числа зберігаються за допомогою позначення двох доповнень з p бітами. Позитивні цілі числа мають нуль у крайньому лівому біті, а двійкове представлення цілого числа в решті p - 1 біт. Сума двох додатних або двох від'ємних цілих чисел може переповнюватися, і це вказується, коли знаковий біт протилежний тому, яким він повинен був бути. Після обговорення цілих чисел у розділі представлена арифметика з плаваючою комою. Представлення включає знаковий біт, показник степеня та мантису (значущі цифри). Оскільки використовується лише скінчена кількість бітів, як правило, 32 або 64, більшість чисел з плаваючою комою не можуть бути точно представлені, саме це є джерелом помилки округлення, що є основною та фундаментальною проблемою. Це кінцеве подання також призводить до переповнення для чисел з плаваючою комою. Існує лише кінцева кількість чисел з плаваючою комою, і їх подання є гранулярним. Степінь гранулярності залежить від постійної **eps** машини, на якій проводиться обчислення. У цьому розділі розглядається арифметика з плаваючою точкою та наводяться межі помилок для деяких операцій з плаваючою точкою. Зазвичай помилки вимірюються з використанням відносної, а не абсолютної помилки.Завершує цю главу обговорення того, як мінімізувати певні типи помилок з плаваючою комою.Зокрема, втрата ефекту меншого числа при додаванні його до набагато більшого і помилка скасування.

У наш час, коли ми покладаємось на комп'ютери у багатьох задачах, люди які не є обізнаним в роботі комп'ютера не здогадуються, що вони роблять

помилки. Насправді, при виконанні арифметики з дійсними числами, такими як 0,3 і 1,67 $\times$ 108, виникає помилка, вже на етапі коли число поміщається в пам'ять комп'ютера, і це називається помилкою округлення. Що ще гірше, помилка поширюється за допомогою арифметичних операцій над цими числами, таких як додавання та ділення. В інженерних та наукових задачах часто доводиться мати справу з великомасштабними матричними операціями. Ці обчислення повинні виконуватися з мінімальною помилкою, так як деякі значення мають бути надзвичайно точними для коректності висновків. Інженер або вчений повинен знати, що помилки будуть виникати, чому вони виникають і як їх мінімізувати.

Ми обговорюємо подання чисел у цифрових комп'ютерах та пов'язану з ними арифметику. Цифровий комп'ютер зберігає значення за допомогою двійкової системи числення, де число представлене рядком нулів та одиниць. Кожна двійкова цифра називається бітом. Оскільки цифрові обчислювальні машини мають кінцеву бітову ємність (пам'ять), цілі числа та дійсні числа представлені фіксованою кількістю двійкових бітів. Ми побачимо, що цілі числа можуть бути представлені рівно до тих пір, поки ціле число не потрапляє в межах фіксованої кількості бітів. Числа з плаваючою комою - це вже інша історія. Більшість таких значень неможливо представити точно. Ми опишемо системи подання, щоб зрозуміти проблеми, пов'язані з цілими числами та числами з плаваючою комою.

4.1.1 Цілі числа

Припустимо, що p біт доступні для представлення цілого числа. Наведемо простий спосіб представлення його в комп'ютері. Позитивне ціле число має нуль в останньому біті, а інші p - 1 біти містять двійкове подання цілого числа.

Приклад 4.1. Наприклад, для p = 8 - натурального числа, комп'ютерне представлення матиме вид:

$$21 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

0	0	0	1	0	1	0	1

Приклад 4.2. Знову, для p = 8 та натурального числа 58:

$$58 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

0	0	1	1	1	0	1	0

Для від'ємних цілих чисел більшість комп'ютерів використовують подання двох додатків. Система працює як ідеальний одометр. Якщо одометр показує 000000, а автомобіль робить резервну копію 1 милю, одометр показує 999999. У двійковій системі з p = 8 нуль дорівнює 00000000, отже - 1 стає 11111111. Продовжуючи таким чином, - 2 стає 11111110, - 3 - 11111101 і так далі. Це може здатися дивним, але насправді дуже ефективно і просто. Якщо це подання - 1 має сенс, повинен бути логічний спосіб прийняти мінус 1 і отримати це подання для - 1. Інвертувати всі біти ($0 \to 1$, $1 \to 0$) і додати 1.

Приклад 4.3 (Представлення від'ємного числа)**.**

$$-1 \to invert(0000001) + 1 = 11111110 + 1 = 111111$$

1	1	1	1	1	1	1	1

Як бачимо з приклада, представляти числа в двійковій системі є доволі зручно і ефективно.

Зауваження 4.1. Крайній лівий біт називається знаковим бітом. Це завжди 0, коли ціле число додатнє, і 1, коли воно від'ємне.

Ми досі не обговорювали, як додати чи відняти два двійкові числа. Якщо ми складемо подання для -1 і 1 разом, то отримаємо 0. Таким чином, сформуємо суму, використовуючи звичайну двійкову арифметику:

$$11111111 + 00000001 = \underline{1}|0000000$$

де підкреслений біт - це перенесення. Відкинувши перенесення, зберігаючи 8 біт, ми отримаємо результат 00000000 або нуль. Ще один приклад запропонує формулу віднімання двох двійкових чисел:

Приклад 4.4. Для суми 95 та -43 будемо мати:

$$95 \to 0101111$$

$$43 \to 00101011, -43 \to 110101100 + 1 = 11010101$$

$$95 + (-43) \to 01011111 + 11010101 = \underline{1}|00110100$$

Зауваження 4.2. Узагальнивши попередні приклади будемо мати правила як виконувати додавання або віднімання чисел доповнення двох:

- Маючи два p-бітові цілі числа m і n, $m + n$ утворюється, виконавши двійкове додавання і відкинувши перенесення.
- Віднімання $m - n$ виконується додаванням $(-n)$ до m, тому $m - n = m + 2comp(n)$.

Набільше додатнє число для p біт буде $0111 \cdots 111 = 2^{p-1} - 1$. Найменше же від'ємне буде $100 \cdots 000 = -2^{p-1}$. Отже, маючи p бітовий комп'ютер ми можемо представляти ціле число n, таке що $-2^{p-1} \leq n \leq 2^{p-1} - 1$.

Зауваження 4.3. Коли знак результату протилежний тому, яким він повинен бути, сталося переповнення.

Приклад 4.5. Нехай в нас є p біт. Спробуємо додати -18 та -112.

$$-18 + (-112) = 11101110 + 10010000 = \underline{1}|01111110 = 126$$

Під час цілочисельних операцій всі результати повинні знаходитись в інтервалі $-2^{p-1} \leq n \leq 2^{p-1} - 1$. Це є суттєвим обмеженням. Якщо необхідно операції на великими цілими числами, частіше за все потрібно використовувати додаткове програмне забезпечення, яке часто називають пакетом BigInteger[5]. Наприклад, для шифрування даних зазвичай потрібні операції з дуже великими цілими числами.

4.1.2 Дійсні числа

Визначення 4.1. Нехай в нас є цілі числа b, p, e_{min}, e_{max}, тоді ненульове число з плаваючою комою можно представити як дійсне число виду:

$$\pm(0.d_1d_2\cdots d_p) \times b^n$$

де $d_1 \neq 0, 0 \leq d_i \leq b-1$ та $-e_{min} \leq n \leq e_{max}$. Нехай тоді F буде скінченна множина всіх чисел з плаваючою комою. У такому представлені:

1. b - це база. Найпоширеніші бази це b = 2 (бінарна система), b = 10(десяткова система) та b=16 (шістнадцятерична система).

2. $-e_{min} \leq n \leq e_{max}$ - показник ступеня, що визначає порядок величини числа, що кодується.

3. Ціле число $0 \leq d_i \leq b-1$ називаються цифрами, а p - кількістю значущих цифр. Мантисою же називають ціле число $m = d_1d_2\cdots d_p$.

Зауваження 4.4.

$$(0.d_1d_2\cdots d_p) \times b^n = (d_1 \times b^{-1} + d_2b^{-2} + \cdots + d_pb^{-p})b^n = b^n \sum_{k=1}^{p} d_kb^{-k} \quad (4.1)$$

Для чисел із плаваючою комою виконуються наступна нерівність:

$$f_{min} \leq |f| \leq f_{max}, f \in F.$$

де $f_{min} = b^{-e_{min}+1}$ - найменше додатнє дісне число. Це легко побачити якщо взяти $n = -e_{min}, d_1 = 1, d_2 = d_3 = \cdots = d_p = 0$ і підставити в рівняння 4.1. Найбільше число же можна отримати взявши $d_i = (b-1), 1 \leq i \leq p$ і степінь $n = e_{max}$.

Використовуючи формулу суми геометричної прогресії отримаємо:

$$f_{max} = b^{e_{max}}(b-1)(b^{-1} + b^{-2} + \cdots + b^{-p}) = b^{e_{max}}(b-1)(\frac{1-(\frac{1}{b})^p}{b-1}) =$$
$$b^{e_{max}}(1-(\frac{1}{b})^p) = b^{e_{max}}(1-b^{-p})$$

Менші числа призводять до недотоку, а більші - до переливу.Комп'ютери використовують базу b = 2 і зазвичай підтримують одинарну точність (подання з p = 32) і подвійну точність (представлення з p = 64). У нашому одноточному поданні ми використовуємо 1 біт для знака (0 означає позитивний, 1 означає негативний), 8 бітів для показника ступеня та 23 біта для мантиси (загалом 32 біти). У нашому поданні з подвійною точністю ми використовуємо 1 біт для знака, 11 бітів для показника ступеня та 52 біта для мантиси (загалом 64 біти). Наприклад, уявіть 81.625 в одній точності.

$$81.625 = (1)2^6+(0)2^5+(1)2^4+(0)2^3+(0)2^2+(0)2^1+(1)2^0+(1)2^{-1}+(0)2^{-2}+(1)2^{-3}$$
$$= ((1)2^{-1}+(0)2^{-2}+(1)2^{-3}+(0)2^{-4}+(0)2^{-5}+(0)2^{-6}+(1)2^{-7}+(1)2^{-8}+(0)2^{-9}+$$
$$+ (1)2^{-10})2^7$$

Зверніть увагу, що початковою цифрою є $d_1 = 1$, як потрібно, а показник ступеня правильно відрегульований. Щоб уникнути роботи з негативним показником, закодуйте його як ціле без знака, додавши до нього «упередження»

(127 - це звичайне упередження в одній точності). У нашому прикладі 81.625 показник ступеня буде збережений як 7 + 127 = 134 із використанням зміщення 127. Щоб отримати фактичний показник, обчисліть 134 - 127 = 7. Якщо показник - 57, він зберігається як - 57 + 127 = 70. Збережений показник 0 відображає фактичний показник - 127. Ось внутрішнє представлення для 81.625, де знак = 0, показник = 134 у полі 8 біт, а мантиса слідує за показником у полі 23 біта .

$$0|10000110|10100011010000000000000$$

Все ще залишається проблема x = 0,0. Представте його, заповнивши нулем показник степеня та мантису. Знаковий біт все ще може бути 0 (+) або 1 (-), тому іноді ви побачите результат, наприклад - 0,0000. Зауважте, що ця ситуація не трапляється із цілочисельним поданням двох доповнень.

Також існує проблема округлення та обрізання деяких дійсних числе.

Приклад 4.6 (Приклад скороченя)**.** Нехай $x = \frac{1}{6}$ і припустимо, що b = 2 і p = 8. Бінарне подання $\frac{1}{6}$ є нескінченним повторюваним паттерном:

$$0.00101010101010101010101010101$$

У нас є лише вісім двійкових цифр, з якими можна працювати, тому, коли $\frac{1}{6}$ вводиться в пам'ять комп'ютера, двійкова послідовність або округляється, або скорочується. При використанні округлення наближення становить 0,00101011, але з усіченням наближення становить 0,00101010.

4.1.3 Переведеня дійсних чисел в числа з плаваючою крапкою

Найбільш широко використовуваним стандартом для обчислення з плаваючою точкою є стандарт IEEE для арифметики з плаваючою точкою. Найбільш часто використовуваними форматами IEEE є одинарна та подвійна то-

чність. У кожному випадку ненульове число вважається прихованим 1 перед першою цифрою. Як результат, одинарна точність використовує 24 двійкові цифри, а подвійна точність 53. У таблиці перелічені атрибути двох форматів.

Точність	Система	Знаків	e_{min}	e_{max}	Діапазон (приблиз.)
Одинична	2	23 + 1	-126	+127	$1.18 \times 10^{-38} 3.4 \times 10^{38}$
Подвійна	2	52 + 1	-1022	+1023	$2.23 \times 10^{-308} 1.80 \times 10^{308}$

Числа з плаваючою комою є гранулярними, що означає, що між числами є пробіли. Гранулярність обумовлена тим, що кінцева кількість бітів використовується для представлення числа з плаваючою комою. Ми точно представили 81.625, оскільки 0.625 це $\frac{1}{2} + \frac{1}{8}$, але більшість дійсних чисел потрібно представляти приблизно, оскільки не існує точного перетворення в двійкове, що ми і побачили в 4.6.

Відстань від 1,0 до наступного найбільшого числа подвійної точності становить 2^{-52} у подвійній точності IEEE. Тому якщо до 1 додати число менше 2^{-52}, то результат буде 1. Числа з плаваючою комою від $1,0$ до $2,0$ мають однакову відстань між собою:

$$\{1, 1 + 2^{-52}, 1 + 2 \times 2^{-52}, 1 + 3 \times 2^{-52}, \cdots, 2\}$$

Розрив збільшується із збільшенням розміру інтервалів у 2 рази. Числа від 2,0 до 4,0 вже розділяються проміжком 2^{51}.

$$2 \times \{1, 1 + 2^{-51}, 1 + 2 \times 2^{-51}, 1 + 3 \times 2^{-51}, \cdots, 2\}$$

Загалом, інтервал від 2^k до 2^{k+1} має розрив між значеннями $2^k 2^{-52}$. Зі збільшенням k розрив відносно 2^k залишається 2^{-52}. За одиничної точності відносний відстань між числами становить 2^{-23}. З цим розривом пов'язана

назва. Її називають машинною точністю, або eps, і вона відіграє значну роль в аналізі операцій з плаваючою точкою. Очевидно, що значення eps змінюється залежно від точності представлення чисел в машині.

Визначення 4.2. Припустимо, що b - основа системи числення, p - кількість значущих цифр, і використовується округлення. Точність машини,eps=$\frac{1}{2}b^{1-p}$, - це відстань від 1,0 до наступного за величиною числа з плаваючою комою.

Для для чисел з плаваючою крапкою з подвійною точністю IEEE ми вказали eps = 2^{-52}. Застосувавши формулу з b = 2 та p = 52, маємо $eps = \frac{1}{2}2^{1-52} = 2^{-52}$. В одинарній точності, $eps = \frac{1}{2}2^{1-23} = 2^{-23}$, взявши b = 2 та p = 23.

Перетворення дійсних чисел у числа з плаваючою точкою називається поданням з плаваючою точкою або округленням, а помилка між дійсним значенням і значенням з плаваючою точкою називається помилкою округлення. Ми можемо очікувати, що $(fl(x) - x) = \epsilon$ не перевищуватиме eps за абсолютною величиною. Наступна формула виконується для всіх дійсних чисел $f_{min} \leq x \leq f_{max}$:

$$fl(x) = x(1 + \epsilon), \quad |\epsilon| \leq eps \tag{4.2}$$

4.1.4 Арифметика з плаваючою крапкою

Розглянемо операцію +. Сума двох чисел із плаваючою комою, як правило, є наближенням до фактичної суми. Позначимо через $\oplus$ комп'ютерний результат додавання.

Визначення 4.3. Для дійсних чисел x та y,

$$x \oplus y = fl(fl(x) + fl(y)) \tag{4.3}$$

Переповнення відбувається, якщо при додаванні утворюється занадто велике число, $|x \oplus y| > f_{max}$, а недоповнення відбувається, якщо утворюється занадто мале число, $|x \oplus y| < f_{min}$. Подібні позначення застосовуються до інших операцій: $\ominus, \otimes$ та $\oslash$.

Приклад 4.7. Нехай b = 10, p = 4 та справжні значення дійсних x та y будуть 0.34578×10^1 та 0.56891×10^1 відповідно. Тоді,

$$fl(x) = 0.3458 \times 10^1, \quad fl(y) = 0.56891 \times 10^1$$

Звідки

$$x \oplus y = 0.9147 \times 10^1$$

При виконанні операцій з плаваючою комою над значеннями з різними показниками ступеня має відбутися зсув одного з значень.

Зауваження 4.5. Числа з плаваючою комою мають фіксований діапазон, тому, як і у випадку з цілими числами, для роботи з числами з плаваючою точкою з багатьма значущими цифрами та великими показниками потрібно мати відповідне програмне забезпечення.

Існує два способи вимірювання похибки обчислень, за допомогою абсолютної або відносної похибки:

$$\text{Абсолютна помилка} = |fl(x) - x|$$
$$\text{Відносна помилка} = \frac{fl(x)-x|}{|x|}, \quad x \neq 0$$

Відносна похибка вказує на те, наскільки точноми є вимірювання щодо розміру вимірюваної величини.Відносна похибка забезпечує набагато кращий показник змін практично для будь-яких цілей. Наприклад, для задачі оцінки суми ряду $\sum_{i=1}^{\infty} \frac{1}{i^2}$ ми можемо рахувати послідовності часткових сум $s_n = \sum_{i=1}^{n} \frac{1}{i^2}$

допоки не отримаємо бажаної точності. Проблема полягає в тому, що фактична сума ряду невідома, тому порівняння часткової суми з фактичною є неможливим. Є два підходи, які зазвичай використовуються у цьому випадку:

(а) Рахувати часткові суми дококи $|s_{n+1} - s_n| > \text{tol}$.

(б) Рахувати часткові суми дококи $\frac{|s_{n+1}-s_n|}{|s_n|} > \text{tol}$.

Де tol буде дорівнювати значенню бажаною точності обчислення.В загальному випадку метод 2 є кращим, оскільки він показує нам, як нова часткова сума змінюється відносно попередньої суми. І у разі малої зміни є сенс зупинити ітерації обчислення.

Зауваження 4.6. Більшість комп'ютерних реалізацій додавання (включаючи широко використовувану арифметику IEEE) задовольняють властивості того, що відносна похибка менша за точність машини:

$$|\frac{(x \oplus y) - (x + y)}{x + y}| \leq eps$$

якщо, $x + y \neq 0$

Також варто зазначиити, що відносна помилка для однієї операції є дуже малою, але це не завжди має місце, коли обчислення включає послідовність багатьох операцій.

Важливо розуміти, як поширюються помилки під час операцій на числами з плаваючою точкою, оскільки тоді можна буде контролювати розповсюдження помилок. Для прикладу проведемо математичний аналіз помилки округлення для додавання чисел з плаваючою комою. Також буде наведено результати аналізу для операцій $\oplus$, $\ominus$ та $\oslash$, і що найважливіше для векторних та матричних операцій.У всіх випадках ми будемо вважати, що наближення x за допомогою $fl(x)$ здійснюється шляхом округлення, а не відкидання.

Зауваження 4.7. IEEE 754 вимагає, щоб арифметичні операції давали точно округлені результати, тобто такі самі,як якщо б значення були обчислені з нескінченною точністю до округлення. Будемо вважати, що коли числа з плаваючою точкою x та y знаходяться в пам'яті комп'ютера, то основні арифметичні операції задовольняють наступні правила:

Для всіх чисел з плаваючою точкою x, y в комп'ютері:

$$x \oplus y = (x + y)(1 + \epsilon) \tag{4.4}$$

$$x \ominus y = (x - y)(1 + \epsilon) \tag{4.5}$$

$$x \otimes y = (x \times y)(1 + \epsilon) \tag{4.6}$$

$$x \oslash y = (x/y)(1 + \epsilon) \tag{4.7}$$

де $|\epsilon| \leq eps$. При додаванні n чисел з плаваючою комою, результат є точною сумою n чисел, кожне з яких збурене невеликою відносною помилкою. Помилки обмежені $(n-1) \times eps$, де eps - одинична похибка округлення.

Відносна похибка при обчисленні добутку n чисел з плаваючою комою становить не більше $1,06(n-1) \times eps$, припускаючи, що $(n-1) \times eps < 0,1$.

Також існують межі помилок для матричних операцій, які залежать від eps та величини справжніх значень.

Лема 4.1. Нехай $x_1, x_2, \cdots, x_n$ є додатніми числами з плаваючою комою в комп'ютері. Тоді,

$$\begin{aligned} x_1 \oplus x_2 \oplus x_3 \cdots \oplus x_n = x_1(1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\ +x_2(1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\ +x_3(1+\epsilon_2)(1+\epsilon_3)\cdots(1+\epsilon_{n-1}) \\ +x_{n-1}(1+\epsilon_{n-2})(1+\epsilon_{n-1}) \\ +x_n(1+\epsilon_{n-1}) \end{aligned}$$

де $|\epsilon_i| \leq eps, 1 \leq i \leq n-1$

Доведення. Припустимо, обчислення відбуваються наступним чином:

$$s_2 = x_1 \oplus x_2, s_3 = s_2 \oplus x_3, \cdots, s_n = s_{n-1} \oplus x_n$$

з вып. 4.4

$$s_2 = fl(x_1 + x_2) = (x_1 + x_2)(1 + \epsilon_1) = x_1(1 + \epsilon_1) + x_2(1 + \epsilon_1)$$

де, $\epsilon_1 \leq eps$. Тепер,

$$\begin{aligned} s_3 = fl(s_2 + x_3) = (s_2 + x_3)(1 + \epsilon_2) \\ = s_2(1 + \epsilon_2) + x_3(1 + \epsilon_2) = x_1(1 + \epsilon_1)(1 + \epsilon_2) \\ + x_2(1 + \epsilon_1)(1 + \epsilon_2) + x_3(1 + \epsilon_2) \end{aligned}$$

Звідки ми можемо помітити що,

$$\begin{aligned} s_n = x_1(1 + \epsilon_1)(1 + \epsilon_2) \cdots (1 + \epsilon_{n-1}) \\ + x_2(1 + \epsilon_1)(1 + \epsilon_2) \cdots (1 + \epsilon_{n-1}) \\ + x_3(1 + \epsilon_2)(1 + \epsilon_3) \cdots (1 + \epsilon_{n-1}) \\ \vdots \\ + x_{n-1}(1 + \epsilon_{n-2})(1 + \epsilon_{n-1}) \\ + x_n(1 + \epsilon_{n-1}) \end{aligned}$$

де $\epsilon_i \leq eps, 1 \leq i \leq n-1$. Якщо послідувати аналізу з [ref], то нахай

$$\begin{aligned}
1+\eta_1 &= (1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
1+\eta_2 &= (1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
1+\eta_3 &= (1+\epsilon_2)(1+\epsilon_3)\cdots(1+\epsilon_{n-1}) \\
&\vdots \\
1+\eta_{n-1} &= (1+\epsilon_{n-2})(1+\epsilon_{n-1}) \\
1+\eta_n &= (1+\epsilon_{n-1})
\end{aligned}$$

Тепер ми можемо переписати рівняня в такому виді,

$$s_n = x_1(1+\eta_1)+x_2(1+\eta_2)+x_3(1+\eta_3)+\cdots+x_{n-1}(1+\eta_{n-1})+x_n(1+\eta_n) \quad (4.8)$$

Для того щоб використовувати рівняня выр. 4.8, нам треба обмежити η_i. З $1+\eta_n = 1+\epsilon_{n-1}$ слідує, що $|\eta_n| = |\epsilon_n| \leq eps$. Тепер варто розглянути доданок $1+\eta_{n-1}$:

$$1+\eta_{n-1} = 1+\epsilon_{n-1}+\epsilon_{n-2}+\epsilon_{n-1}\epsilon_{n-2}$$

Звідки слідує що,

$$\eta_{n-1} = \epsilon_{n-1}+\epsilon_{n-2}+\epsilon_{n-1}\epsilon_{n-2}$$

Саме тому,

$$|\eta_{n-1}| \leq |\epsilon_{n-1}+\epsilon_{n-2}| + |\epsilon_{n-1}\epsilon_{n-2}| \leq |\epsilon_{n-1}| + |\epsilon_{n-2}| + |\epsilon_{n-1}\epsilon_{n-2}|$$

Зрозуміло, що доданок $|\epsilon_{n-1}\epsilon_{n-2}|$ обмежений 2eps^2.Якщо ми використовуємо арифметику з подвійною точністю, тоді $eps = 2^{-52}$, звідки $2eps^2 = 2^{-103}$. Зрозуміло, що ми можемо знехтувати $|\epsilon_{n-1}\epsilon_{n-2}|$ відносно до $|\epsilon_{n-1}| + |\epsilon_{n-2}|$. Тому,

$$|\eta_{n-1}| \leq |\epsilon_{n-1}| + |\epsilon_{n-2}| \leq 2eps$$

Продовжуючи таким чином, ми отримаємо

$$|\eta_1| \leq (n-1)eps \tag{4.9}$$

$$|\eta_i| \leq (n-i+1)eps, \quad 2 \leq i \leq n, \tag{4.10}$$

Беручи до уваги вищенаведені викладки, ми можемо узагальнити правило для суми. При додаванні n чисел з плаваючою комою, результатом є точна сума n чисел, кожне з яких збурене невеликою відносною помилкою. Помилки обмежені $(n-1)$ eps, де eps - одинична похибка округлення. Для подальших досліджень також варто навести деякі теореми про помилки для інших операцій.

Теорема 4.1. *При додавані n чисел з плаваючою точкою,*

$$\frac{|s_n - s|}{|x_1 + x_2 + \cdots + x_n|} \leq K(n-1)eps, \text{ де } K = \frac{|x_1| + |x_2| + \cdots + |x_n|}{|x_1 + x_2 + \cdots + |x_n||}$$

Теорема 4.2. *Відносна помилка при обчисленні добутку n чисел із плаваючою комою становить не більше* $1,06 \times (n-1)eps$*, припускаючи, що* $(n-1) \times eps < 0,1$.

Теорема 4.3. *Для* $m \times n$*-матриці* M *визначимо* $|M| = (|mij|)$*. Тобто* $|M|$ *- це матриця, елементами якої є абсолютні значення елементів M. Нехай A і B - дві матриці з елементів з плаваючою точкою, а c - число з плаваючою точкою. Тоді*

$$fl(cA) = cA + E, \quad |E| \leq eps \times |cA|$$

$$fl(A + B) = A + B + E, \quad |E| \leq eps \times |A + B|$$

Якщо добуток A та B визначений, то

$$fl(AB) = AB + |E|, \quad |E \leq eps \times |A||B| + K \times eps^2$$

де K є константою.

4.1.5 Мінімізація помилок

Як бачимо, то помилки напряму залежать від заданих чисел або матриць, саме тому дуже важливою є теорія з їх мінімізації. Тема мінімізації помилок є складною і іноді передбачає застосування складних математичних маніпуляцій, але є кілька загальних принципів, яких слід дотримуватися. Стаття є чудовим підсумком попередніх розділів цієї глави та містить обговорення деяких методів мінімізації помилок. Я наведу лише основні з них.

Додавання дуже великого числа до малого може виключити будь-який внесок малого числа до кінцевого результату.

Приклад 4.8. Нехай b=10, p=4. Візьмемо $x = 0.267356 \times 10^3$ та $y = 0.45539 \times 10^{-3}$. Тоді,

$$fl(x) = 0.2674 \times 10^3, \quad fl(y) = 0.4554 \times 10^{-2}$$

звідки,

$$fl(fl(x) + fl(y)) = 0.2674 \times 10^3$$

Переповнення частіше за все є результатом включення дуже великих чисел у розрахунки, іноді його можна уникнути, просто змінивши порядок, в якому робляться обчислення.

Розв'язування квадратного рівняння $ax^2 + bx + c = 0, \quad a \neq 0$ є класичним прикладом, коли віднімання близьких чисел може бути згубним. Цей тип помилки називається помилкою віднімання. Звичайний спосіб знайти два його корені - це використання квадратної формули. Якщо добуток $4ac$ малий, тоді $\sqrt{b^2 - 4ac} \cong b$, і при обчисленні одного з x_1 або x_2 ми будемо віднімати два майже однакові числа. Це може бути серйозною проблемою, оскільки ми знаємо, що комп'ютер має лише фіксовану кількість значущих цифр.

Є й інші класичні приклади, коли помилка при відніманні викликає серйозні проблеми, і деякі з них включені в них. Див. Посилання [19, с. 42-44] для кількох цікавих прикладів.

4.2 Число обумовленості матриці

Наша мета - визначити критерії, які допомагають нам вирішити, який алгоритм використовувати для конкретної проблеми, або коли дані для проблеми можуть спричинити обчислювальні проблеми. Для цього нам потрібно зрозуміти два типи помилок - пряму та обернену. Розглянемо проблему, яку потрібно розв'язати алгоритмом, як функцію f, що відображає вхідні дані x на розв'язок $y = f(x)$. Однак через неточності під час обчислення з плаваючою точкою результат буде дорівнювати: $\hat{y} = \hat{f}(x)$ Перед визначенням типів помилок прикладів необхідно ввести нове позначення.

Визначення 4.4. Якщо A - матриця $m \times n$, $|A| = (|a_{ij}|)$; іншими словами, $|A|$ - матриця, що складається з абсолютних значень $a_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$.

4.2.1 Типи помилок

Помилка прямого обчислення $f(x)$ становить $|\hat{f}(x) - f(x)|$. Цей вираз вимірює помилки в обчисленні для вхідного x.

Пряма похибка є природною величиною для вимірювання, але, в загальному випадку, ми не знаємо справжнього значення $f(x)$, тому ми можемо отримати лише верхню межу цієї помилки.

Приклад 4.9. Якщо x і y є $n \times 1$ векторами і $n \times eps \leq 0,01$, тоді,

$$|fl(x^T y) - x^T y| \leq 1.01 \times n \times eps|x|^T|y|$$

Результат показує, що пряма похибка декартового добутку векторів невелика.

Обернена похибка пов'язує помилки округлення в обчисленні з помилками в даних, а не з їх рішенням. Як правило, з цієї причини аналіз обернених помилок є кращим, ніж аналіз прямих помилок.

Визначення 4.5. Результат округлення чи інших помилок у даних дає результат - $\hat{y}$. Обернена помилка в цьому випадку - це найменший Δx, для якого зворотна помилка говорить нам, яку проблему ми насправді вирішили.

4.2.2 Коректність(Стабільність) алгоритма

Тепер, коли ми знайомі з оберненою та прямою помилкою обчислень, ми можемо визначити стабільність алгоритму. Інтуїтивно зрозуміло, що алгоритм стабільний, якщо він загалом працює добре, а алгоритм нестабільний, якщо він працює погано у більшій кількості випадків. Зокрема, алгоритм не повинен бути надмірно чутливим до помилок у вхідних данних або помилок під час його виконання.

Навіть при використанні стабільного алгоритму для вирішення проблеми, проблема може бути чутливою до незначних змін (збурень) даних. Обурення можуть виникати внаслідок помилки округлення, невеликих помилок вимірювання при збиранні експериментальних даних, шуму, який не відфільтровується з сигналу, або помилки усічення при наближенні суми нескінченного ряду. Якщо проблема відноситься до цієї категорії, ми вважаємо, що вона обумовлена погано.

Визначення 4.6. Проблема є погано обумовленою, якщо невелика відносна зміна її даних може спричинити велику відносну помилку в її обчислюваному рішенні, незалежно від алгоритму, що використовується для вирішення

проблеми. Якщо невеликі збурення в даних задачі призводять до невеликих відносних помилок у розв'язанні, проблема називається добре обумовленою.

Легко переплутати поняття стабільності та обумовленість. Зауваження містить короткий виклад відмінностей.

Зауваження 4.8. Підсумок

- Стабільний або нестійкий - це є ознакою певного алгоритму.
- Ну або погано обумовлені стосується конкретної проблеми, а не використовуваного алгоритму.

Очевидно, що поєднання помилки округлення з нестабільним алгоритмом може призвести катастрофи під час обчислень.

Як згадувалось у вступі в цьому розділі, стабільний алгоритм вирішення проблеми може давати погані результати, якщо дані є погано обумовленими. Припустимо, що $f(x)$ являє собою алгоритм, який приймає дані x і видає результат $f(x)$. Ми можемо визначити погане та хороше значення обумовленості через $f(x)$. Використовується позначення $\|\cdot\|$, щоб вказати міру розміру, наприклад абсолютне значення дійсного числа або векторну або матричну норму.

Визначення 4.7. Нехай x і $\overline{x}$ - вхідні та дещо збурені дані, а $f(x)$ і $\overline{f}(x)$ - відповідні рішення. Тоді,

Задача є добре обумовлена щодо x, якщо, коли величина $|x - \overline{x}|$ мала, то величина $|f(x) - \overline{f}(x)|$ теж мала.

Задача є погано обумовлена щодо x, якщо, коли величина $|x - \overline{x}|$ мала, то величина $|f(x) - \overline{f}(x)|$ може приймати великі значення.

Чутливість проблеми до збурень в даних вимірюється шляхом визначення числа обумовленості. Чим більше число обумовленості, тим більш чутливою є проблема до змін у даних. Для певного x припустимо, що в даних є невелика помилка, так що вхід до проблеми є $\overline{x} = x + \Delta x$, а обчислене значення буде $\overline{f}(x)$ замість $f(x)$. Сформуємо відносну похибку результату, поділену на відносну похибку данних:

$$\frac{\frac{|f(x)-\overline{f}(x)|}{|f(x)|}}{\frac{|x-\overline{x}|}{|x|}} \tag{4.11}$$

Співвідношення вимірює, наскільки чутлива функція до змін або помилок вводу.

Зауваження 4.9. У математиці супремум (sup) підмножини S впорядкованої множини X є найменшим елементом X, який більший або дорівнює всім елементам S. Він відрізняється від максимуму тим, що він не повинен бути член підмножини S.

Визначення 4.8. Число обумовленості задачі f із входом x дорівнює,

$$C_f(x) = \lim_{\epsilon \to 0^+} \sup_{\left\|\delta \mathbb{X}\right\| \leq \epsilon} \frac{\frac{|f(x)-f(x+\delta x)|}{|f(x)|}}{\frac{|\delta x|}{|x|}} \tag{4.12}$$

Можно думати про число обумовленості як про граничну поведінку выр. 4.11, коли похибка δx стає малою.

4.2.3 Погано обумовлені СЛАР

Зробивши доволі серйозний аналіз, який я не буду наводити в даній роботі, але його можна подивитись тут, ми можемо знайти число обумовленості для будь-якої матриці.

Визначення 4.9. Число $\|A\|\|A^{-1}\|$ називається числом обумовленості матриці A, і ми будемо позначати його $k(A)$.

Визначення 4.10. Якщо число умов матриці A велике, ми говоримо, що A є погано обумовленою, в іншому випадку A добре обумовлена.

Термін "великий"є неточним. Числа обумовленосі у діапазоні 10^4 або більше однозначно вказують на погану обумовленість. Для деяких матриць менше число обумовленості може вказувати на порушення обумовленості, тому визначення поганої обумовленості також не є точним.

5 ПРАКТИЧНА ЧАСТИНА

В практичній частині запропоновано реалізацію алгоритму Гауса в вибором найбільшого елементу, що застосовую довгу арифметику для обчислень (є можливість керувати розміром мантиси), використовуючи засоби мови програмування Python та деяких математичних бібліотке для цієї мови. Основою реалізаціїї є вбудований клас мови програмування Python, а саме - Decimal. Цей клас імплементує стандарт IEEI для мови програмування Python. Результатом реалізації практичної частини також є порвіняльна таблиця швидкодії та точності алгоритму для різних значень вхідних данниз та розміру мантиси. Для кращої наглядності результати також будуть зображені на графіках.

5.1 Код програми

Основана частина реалізації методів мовою Python:

```
from sys import float_info
from decimal import *
from fractions import Fraction
import timeit
import csv

import numpy as np
from numpy import linalg as lg



# Constants
RANK = 50



def gauss_pivot(matrix: np.array, vector: np.array):
    n, m = matrix.shape
```

```
    # first, vector should have correct shape
    if vector.shape != (n, 1):
        vector = np.reshape(vector, (n, 1))
    # getting augmented matrix
    aug_matrix = np.append(matrix, vector, axis=1)

    for i in range(m):
        # getting current leftmost column
        current_column = aug_matrix[i:, i]
        # Step 0: find pivot (max element in the column)
        pivot_index = np.argmax(current_column) + i

        # Step 1: prerform row interchange (if necessary)
        # so that the pivot is in the first row
        if i != pivot_index:
            aug_matrix[[i, pivot_index]] = aug_matrix[[pivot_index, i]]

        # gaussian elimination procedure
        for j in range(i + 1, n):
            aug_matrix[j, :] -= aug_matrix[i, :] * (aug_matrix[j, i] /
                aug_matrix[i, i])

    return backward_substitution(aug_matrix[:, :-1], aug_matrix[:, -1])


def backward_substitution(upper_triang_matrix: np.array, vector: np.array)
   -> np.array:
    n, m = upper_triang_matrix.shape
    # vector where we will hold result
    res = np.empty(n, dtype=object)
    # fist we get last component
    res[n - 1] = vector[n - 1] / upper_triang_matrix[n - 1, n - 1]
    for i in range(n - 2, -1, -1):
        S = vector[i]
        for j in range(n - 1, i, -1):
            S -= upper_triang_matrix[i, j] * res[j]
        res[i] = S / upper_triang_matrix[i, i]
```

```
    return res


def decimal_input(rank: int):
    gilbert_matrix = np.array(
        [
            [Decimal(1) / Decimal(i + j - 1) for i in range(1, rank + 1)]
            for j in range(1, rank + 1)
        ]
    )
    vector_x = np.array([[Decimal(-1 ** (i - 1))] for i in range(rank)])
    return gilbert_matrix, vector_x


def float_input(rank: int):
    float_gilbert_matrix = np.array(
        [[1 / (i + j - 1) for i in range(1, rank + 1)] for j in range(1,
           rank + 1)]
    )
    float_vector_x = np.array([1.0 for _ in range(rank)])
    return float_gilbert_matrix, float_vector_x



getcontext().prec = 30
gilbert_matrix, vector_x = decimal_input(RANK)
ge_result = gauss_pivot(gilbert_matrix, vector_x)
print(ge_result)
print(np.dot(gilbert_matrix, ge_result))

float_gilbert_matrix, float_vector_x = decimal_input(RANK)
float_ge_result = gauss_pivot(gilbert_matrix, vector_x)
print(float_ge_result)
print(np.dot(float_gilbert_matrix, float_ge_result))

print("Matrix cond number:", lg.cond(float_input(RANK)[0]))
print("Resulting vectors:", float_ge_result, ge_result)
```

```
print("Let's look what we have after getting dot product:")
print(np.dot(gilbert_matrix, ge_result))
print(np.dot(float_gilbert_matrix, float_ge_result))
```

Код для генерації результатів для різних значень мантиси та рангу матриці:

```
for rank in range(5, 101, 5):
    for prec in range(5, 51, 5):



        comp_time_for_decimal = timeit.timeit(
            f"gauss_pivot(decimal_input({rank})[0], decimal_input({rank})
                [1])",
            setup="from __main__ import gauss_pivot, decimal_input",
            number=100
        )

        comp_time_for_float = timeit.timeit(
            f"gauss_pivot(float_input({rank})[0], float_input({rank})[1])",
            setup="from __main__ import gauss_pivot, float_input",
            number=100
        )

        # we will write measurment results in a form of
        # (rank, precision, decimal_time, float_time)
        computational_results.append((rank, prec, comp_time_for_decimal,
            comp_time_for_float))



with open('results.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Rank", "Precision", "Decimal time", "Float time"])
    writer.writerows(computational_results)
```

5.2 Результити роботи

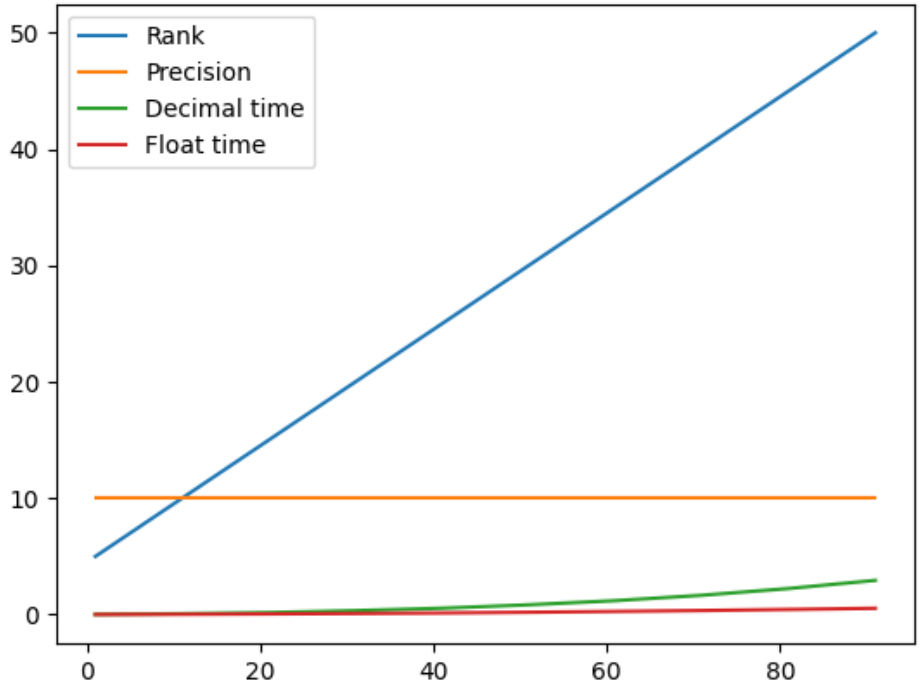
Нище будуть наведені результати роботи програми для різних значень рангу матриці та типу використовуваних чисел. Для тестуваня використовувалась матриця Гільберта та ще одна погано обумовлена матриця.

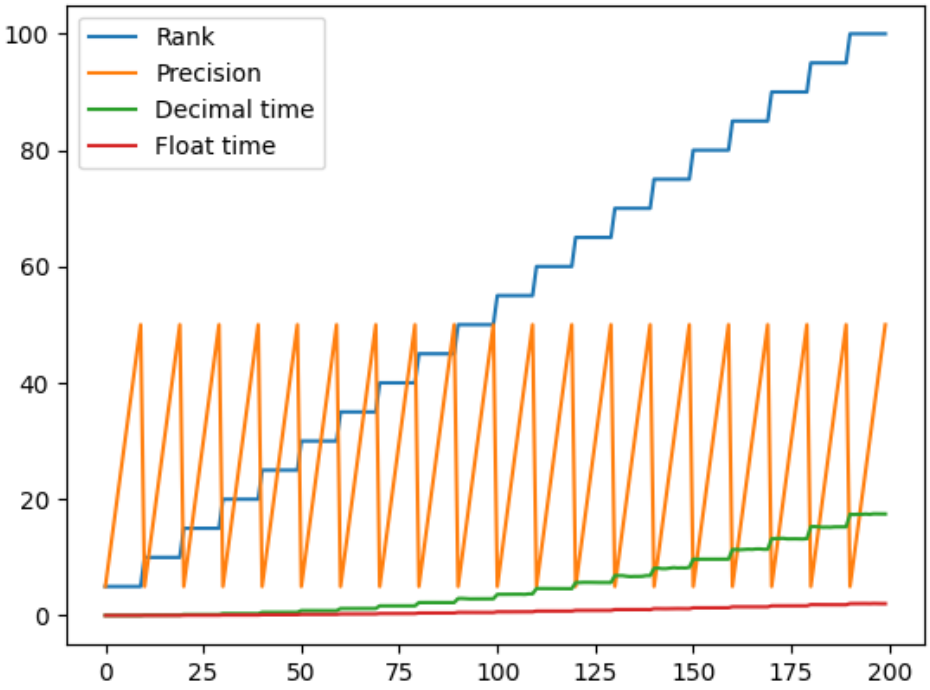
Ранг	Мантиса	Час(сек) Decimal	Час(сек) Float
5	5	0.0267616860000999	0.0104641180005274
5	10	0221818079953664	0.0104252689998248
10	5	0.0790277670021169	0.0293207979993895
10	10	0.0782174720006879	0.0290597610000987
15	5	0.18067644799849	0.0587784270028351
20	10	0.19236764479989	0.0787784270028351

Табл. 1: Результати для матриці Гільберта

Ранг	Мантиса	Час(сек) Decimal	Час(сек) Float
5	5	0.3217213460000999	0.0123641180005274
5	10	0341818079953664	0.0134223689998248
10	5	0.0990277670021169	0.0293207979993895
10	10	0.0382174324056879	0.0290597610000987
15	5	0.480676247998849	0.0582385275028351
20	10	0.50676447998849	0.0781384270028351

Табл. 2: Результати для матриці іншої матрці





На графіках вище зображені результати роботи алгоритмів.

6 ВИСНОВКИ

Для стискання нескінченно великої кількості дійсних чисел у кінцеву кількість бітів потрібно приблизне подання. Хоча цілих чисел нескінченно багато, у більшості програм результат цілочисельних обчислень може зберігатися в 32 бітах. На відміну від цього, враховуючи будь-яку фіксовану кількість бітів, більшість обчислень з дійсними числами дадуть величини, які неможливо точно представити, використовуючи таку кількість бітів. Тому результат обчислення з плаваючою точкою часто потрібно округляти, щоб повернутися до його кінцевого подання. Ця помилка округлення є характерною рисою обчислення з плаваючою точкою.

Багато людей арифметику з плаваючою крапкою вважають предметом езотерики. Це досить дивно, оскільки обчисленя з плаваючаою крапкою взустрічається в комп'ютерних системах повсякчас. Майже кожна мова програмуваня має тип даних із плаваючою комою, комп'ютери від ПК до суперкомп'ютерів мають прискорювачі з плаваючою крапкою, більшість компіляторів будуть покликані час від часу компілювати алгоритми з плаваючою комою, і практично кожна операційна система повинна реагувати на винятки з плаваючою комою, такі як переповнення. Тому урахуванян цих винятків та нюансів роботи з числами в пам'яті комп'ютера є надзвичайно важливою задачою при проведені будь-яких розрахунків.

Було опрацьовано велика кількість джерел, але не було знайдено готової реалізаціх можливості контролювати розмір мантиси числа під час обчислення розв'язку СЛАР методом Гауса з вибором найбільшого елемента, у вже готових бібліотках. Тому запропонована власна реалізація та проведені тестування на погано обумовлених СЛАР.

ДЖЕРЕЛА

[1] Matrix Analysis, (second ed.), Cambridge University Press, New York (2013)

[2] Accuracy and Stability of Numerical Algorithms(second ed.), SIAM, Philadelphia (2002)

[3] What every computer scientist should know about floating-point arithmetic. Comput. Surv., 23 (1) (1991)

[4] Numerical Linear Algebra and Applications (second ed.), SIAM, Philadelphia (2010)

[5] Java 11 Documentation: BigInteger class, `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/math/BigInteger.html`