МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МУДРЕНКО ОКСАНА ЮРІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
прикладної математики та
кібербезпеки
Алла ЛУЦЕНКО

«___»_____ 2024р.

КОНТРОЛЬ ТОЧНОСТІ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ (СТАНДАРТ IEEE 754)

Спеціальність 113 Прикладна математика

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Вєтров О.С.,
старший викладач кафедри
прикладної математики та
кібербезпеки

_____

Оцінка: _____ /_____ /_____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова комісії: _____

Вінниця – 2024

АНОТАЦІЯ

Мудренко О.Ю. Контроль точності комп’ютерних обчислень (стандарт IEEE 754). Спеціальність 113 «Прикладна математика», Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Кваліфікаційна робота присвячена дослідженню та аналізу контролю точності комп'ютерних обчислень за стандартом IEEE 754. Проблема точності комп’ютерних обчислень є однією з фундаментальних проблем Computer Science, оскільки неврахування межі достовірність комп’ютерних обрахунків може призвести до непередбачуваних результатів. Фізичні обмеження на обсяг пам’яті, що виділяється комп’ютерною технікою для обробки числа, накладає принципові обмеження на можливості та логіку організації комп’ютерних обрахунків, що також потребує залучення специфічних математичних методів дослідження.

Ключові слова: числа з плаваючою комою, стандарт IEEE-754, точність комп’ютерних обрахунків.

33 с., 7 рис., 23 джерела

Mudrenko O.Y. Control of the accuracy of computer calculations (IEEE 754 standard). Speciality 113 "Applied Mathematics", Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

The qualification work is devoted to the study and analysis of the accuracy control of computer calculations according to the IEEE 754 standard. The problem of computer computing accuracy is one of the fundamental problems of Computer Science, since failure to take into account the limit of reliability of computer calculations can lead to unpredictable results. Physical limitations on the amount of memory allocated by computer equipment for number processing impose fundamental restrictions on the capabilities and logic of computer calculations, which also requires the use of specific mathematical research methods.

Key words: floating point numbers, IEEE-754 standard, accuracy of computer calculations.

33 pages, 7 figures, 23 references.

ЗМІСТ

ВСТУП

Однією з важливих проблем прикладної математики – є проблема достовірності і надійності комп'ютерних обчислень. Ця задача є однією з фундаментальних проблем інформатики [1], оскільки лежить на стику прикладної математики, з одного боку, і фізичних і технічних обмежень комп'ютерної техніки, з іншого. Уникнення основ двійкової арифметики та принципів стандарту представлення чисел у пам'яті комп'ютера може істотно вплинути на правильність як теоретичних досліджень (якщо для моделювання використовувалася комп'ютерна техніка), так і просто на якість продукту при розробці прикладних програм.

Від точності результатів обчислень залежать успішність і правильність вирішення задач у різних галузях, таких як наука, медицина, фінанси, військова справа і багато інших. Помилки в обчисленнях можуть призвести до серйозних наслідків, зокрема до фінансових втрат, технічних збоїв або навіть до ситуацій катастрофічних масштабів. Актуальність теми дослідження стає очевидним, якщо пригадати відомі катастрофи, пов'язані з неточністю комп'ютерних досліджень. Відомим прикладом трагічних наслідків подібних проблем є «Поломка ракети Patriot» [2-3]. Основна проблема полягала в тому, що значення часу в десятих частках секунди було помножено на коефіцієнт 0,1, щоб отримати представлення часу в секундах. Це обчислення було виконано з використанням 24-розрядного регістра з фіксованою комою. Акумулятор Patriot пропрацював близько 100 годин, а накопичена похибка склала близько 0,34 секунди. Можна також згадати прецедент "Вибуху Аріани 5" [4-5]. В результаті комп'ютерних обчислень 64-розрядне число з плаваючою комою було перетворено в 16-розрядне ціле число зі знаком. Число було більше 32 768, тому перетворення не вдалося. Помилка в системі розрахунку горизонтальної швидкості ракети відносно платформи призвела до збитків, оцінених у 500 мільйонів доларів. Деякі приклади реальних втрат, викликаних помилками округлення або проблемами представлення чисел у пам'яті комп'ютера, наведені в [6], оскільки проблематика точності обчислень є актуальною і для багатьох наукових досліджень. Наприклад, у фізиці і хімії високоточні

обчислення необхідні для моделювання складних процесів, таких як квантові взаємодії і хімічні реакції. У біології і медицині чисельні моделі використовуються для аналізу геному, моделювання поширених захворювань або проектування нових ліків. В цих галузях невеликі похибки можуть призвести до неправильних висновків і рекомендацій, які в свою чергу передують летальним результатам.

Комп'ютерна арифметика (ширше, комп'ютерна математика) фундаментально відрізняється від класичної континуальної математики, оскільки в підвалинах комп'ютерної арифметики лежать саме дискретні принципи.

Можна виділити деякі сучасні специфічні способи роботи з комп'ютерними числами (наприклад, POSIX), але реальним стандартом, що регламентує роботу з автоматичними обчисленнями є саме всесвітньо відомий стандарт IEEE 754.

Створення стандарту IEEE 754 знаменує унікальну можливість в історії, коли технології можуть зробити величезний стрибок вперед. З 8-розрядними мікропроцесорами, добре запровадженими, напівпровідникова технологія швидко розвивалася, і перспектива наукових, інженерних і фінансових обчислень була лише попереду. Індустрія мейнфреймів і міні-комп'ютерів розкололася на двійкові, вісімкові та шістнадцяткові арифметичні засоби на елементах даних різного розміру. Перенесення числових кодів було досить важким завданням. Новий стандарт для арифметики з плаваючою комою запропонував перспективу високоякісної, надійної арифметики для широкого діапазону комп'ютерів і мов програмування.

Стандарт чисел з плаваючою комою IEEE 754 передбачає наближене представлення початкового числа в плаваючих форматах, а похибка при цьому має накопичувальний характер та залежить від подальших операцій. Накопичення цієї похибки в деяких випадках може передувати повній втраті вірного результату.

Дослідженням проблеми точності комп'ютерних обрахунків займались багато відомих вчених та дослідників. Академічною основою представленого

дослідження були обрані представлені далі наукові джерела відомих спеціалістів із зазначеної теми. Вільям Кахан – один з основних авторів стандарту IEEE 754, автор революційного для свого часу алгоритму корекції комп'ютерного підрахунку суми [7]. Його робота в цій галузі зробила великий внесок у розробку стандартів обчислень з рухомою комою. Девід Голдберг – відомий своєю роботою "What Every Computer Scientist Should Know About Floating-Point Arithmetic" [8], яка детально описує особливості та проблеми обчислень з рухомою комою. Фундаментальні засади сучасного стану дослідження комп'ютерних обрахунків можна знайти в [9]. Більш математизований виклад основних моментів роботи з комп'ютерними обрахунками наведений в [10]. Специфічні прикладні питання докладно висвітлені в [11-12].

Варто зазначити, що метою вивчення та аналіз впливу числових форматів і методів округлення, визначених стандартом IEEE 754, на точність розрахунків; змоделювати ризики та розробити рекомендації для забезпечення високої точності обчислень. При цьому, в якості практичного інструментарію дослідження обрана сучасна мова програмування Python, і відповідно основні задачі некоректних результатів обрахунку також пов'язані з нею [13-15]. Математичні підходи в дослідженні також перетинаються з раніше запропонованими в [16-17].

Об'єктом дослідження є представлення чисел у форматі IEEE 754.

Предмет дослідження: коректність прикладних комп'ютерних обрахунків з цілими числами та числами з плаваючою комою.

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, списку використаних джерел та лістингу програми. Загальний обсяг роботи становить ** сторінок, у роботі 3 рисунки та ** таблиць, список використаних джерел містить 20 найменувань.

За матеріалами бакалаврського дослідження автором була опублікована робота у Віснику студентського наукового товариства Донецького національного університету імені Василя Стуса [18].

РОЗДІЛ 1. ПРЕДСТАВЛЕННЯ ЧИСЕЛ В ПАМ'ЯТІ КОМП'ЮТЕРА

1.1. Історія розвитку стандарту IEEE 754

Стандарт IEEE 754 був розроблений Інститутом інженерів з електротехніки та електроніки (IEEE) з метою уніфікації представлення чисел з плаваючою комою у комп'ютерних системах [19]. Його перша версія була опублікована в 1985 році і стала важливим кроком у забезпеченні сумісності та точності обчислень у різних апаратних і програмних системах.

До впровадження цього стандарту різні виробники комп'ютерного обладнання використовували власні формати і алгоритми для роботи з числами з плаваючою комою. Це призводило до несумісності обчислень між різними системами, що ускладнювало перенесення програмного забезпечення і порівняння результатів обчислень.

Розробка стандарту IEEE 754 стала відповіддю на потребу у стандартизації, яка виникла через швидкий розвиток комп'ютерних технологій і необхідність уніфікації методів обробки числових даних. Перший комітет, який працював над стандартом, складався з провідних фахівців у галузі обчислювальної техніки і математики, включаючи таких відомих науковців, як Вільям Кехан, Джон Коган та інші.

Стандарт IEEE 754 зазнавав кількох суттєвих оновлень, основні з яких відбулися у 2008 і 2019 роках [19].

- ✓ IEEE 754-1985
  - Формати представлення чисел: введення одинарної (32 біти) і подвійної точності (64 біти).
  - Арифметичні операції: визначення правил для базових арифметичних операцій, таких як додавання, віднімання, множення, ділення.
  - Спеціальні значення: визначення представлення спеціальних значень, таких як нуль, нескінченність і NaN (Not a Number).
- ✓ IEEE 754-2008

- Розширені формати: додавання форматів з розширеною точністю (quad precision, 128 біти) і додаткових форматів з меншою точністю (half precision, 16 біти).
- Покращення обробки виключень: розширення правил обробки виняткових ситуацій і спеціальних значень.
- Нові операції: введення нових арифметичних операцій і функцій, таких як операції з розширеним діапазоном і округленням.

✓ IEEE 754-2019

- Удосконалення існуючих правил: уточнення правил представлення чисел і виконання арифметичних операцій для забезпечення ще більшої точності і сумісності.
- Нові функції: додавання нових математичних функцій і алгоритмів для обробки чисел з плаваючою комою.
- Підтримка нових технологій: врахування сучасних потреб і технологічних змін, таких як підтримка обчислень у великих обсягах даних і високопродуктивних обчислень.

Стандарт IEEE 754 зіграв вирішальну роль у розвитку комп'ютерних обчислень, забезпечивши уніфікацію і високу точність обробки числових даних. Завдяки постійним оновленням і вдосконаленням, цей стандарт залишається актуальним і ефективним інструментом для сучасних комп'ютерних систем і додатків

1.2. Основні терміни стандарту IEEE 754

Стандарт IEEE 754 є фундаментальним документом, що регламентує представлення чисел з плаваючою комою і виконання арифметичних операцій з ними. Основні положення стандарту охоплюють формати представлення чисел, правила виконання арифметичних операцій та обробку виключень і помилок. Існує декілька способів представлення числа з плаваючою комою (Floating-Point numbers), але стандарт IEEE 754 залишається найефективнішим у більшості випадків.

Стандарт IEEE 754 має три основні компоненти, це знак мантиси, зміщена експонента та нормована мантиса.

- Знак мантиси (або знаковий біт) – це біт, який позначає знак числа. Якщо знак мантиси дорівнює 0, отже сило невід'ємне (додатне або нуль), а якщо ж знак мантиси дорівнює 1 – тоді число від'ємне.
- Експонента – поле експоненти повинно представляти і додатні, і від'ємні експоненти. Для цього до фактичної експоненти додається зсув, щоб отримати збережений показник степеня. Для одинарної точності це значення дорівнює 127.Подвійна точність має 11-бітове поле експоненти зі зміщенням 1023.
- Нормована мантиса – число з плаваючою комою, що складається лише з точних бітів числа. В ній міститься лише дві цифри – це 0 та 1. Нормалізована мантиса має лише одну цифру 1 ліворуч, від десяткової коми.

Також стандарт IEEE зарезервував деякі значення, які можуть бути двозначними:

- Нуль – це спеціальне значення, яке позначається експонентою та мантисою 0.
- Денормалізоване число – це число, яке не має передбачуваного провідного числа перед двійковою крапкою, тобто експонента складається з нулів, а мантиса – ні.

- Не число (not a number, NaN) – спеціальне значення, яке використовується для позначення результатів помилкових операцій. Наприклад, коли всі поля експоненти складаються з одиниць і нульовим бітом знаку або мантиси, що не 1, за якою слідують нулі.

Стандарт IEEE 754 визначає чотири формати представлення чисел з рухомою комою, які забезпечують високу точність і широкий діапазон обчислень.

Формати з одинарною розширеною точністю рідко застосовують на практиці, а основними в застосуванні є формати 32 і 64 біт.

Програми, що використовують числа з рухомою комою, відомі своєю складністю для аналізу. Багато факторів ускладнюють це завдання:

1) компілятори можуть трансформувати код таким чином, що не зберігається семантика обчислень з плаваючою комою;
2) формати чисел з плаваючою комою є аспектом, визначеним реалізацією, у більшості мов програмування;
3) існують різні, несумісні реалізації операцій для одного і того ж формату чисел з рухомою комою;
4) математичні бібліотеки часто не надають жодних або майже жодних гарантій щодо того, що фактично обчислюється;
5) програмістам важко передбачити та уникнути явищ, спричинених обмеженим діапазоном і точністю чисел з плаваючою комою (переповнення, поглинання, скасування, недоповнення тощо); крім того, пристрої, які сучасні формати чисел з плаваючою комою мають для кращого оброблення таких явищ (нескінченності, знакові нулі, денормалізовані числа, нечислові об'єкти, відомі як NaNs), мають свої власні проблеми;
6) округлення саме по собі є джерелом плутанини; крім того, існує кілька можливих режимів округлення, і програми можуть змінювати режим округлення в будь-який час.

Внаслідок цих труднощів, верифікація програм з плаваючою комою в промисловості майже виключно покладається на неформальні методи,

головним чином тестування , або на оцінку числової точності обчислень, що дозволяє визначити лише консервативні (але часто занадто широкі) межі на поширену помилку.

Загальне подання арифметики з плаваючою комою виглядає так: число x має вигляд добутку s, m та e^{β} , де s позначає знак числа, m – мантиса та e^{β} – показник степеня (експонента), тобто:

$$x = s \times m \times e^{\beta}.$$

Якщо мантиса від'ємне або додатне дійсне число, то експонента завжди ціле число. При звичайній формі запис одного числа може мати різний вигляд у залежності від обмежень, які накладаються на його форму. Фактично, місце коми у мантисі m визначається величиною порядку β, тому що зі зміною порядку у більшу або меншу сторону кома, відповідно, переміщуєтсья ліворуч або праворуч, тобто плаває у зображені числа.

Наприклад:

$$2024 = 202{,}4 * 10^{1} = \cdots = 2{,}024 * 10^{3} = 0{,}2024 * 10^{4}$$

$$-0{,}54368 = -5{,}4368 * 10^{-1} = -54{,}368 * 10^{-2} = \cdots = -54368 * 10^{-5}$$

Зазвичай, всі числа з рухомою комою мають загальний формат, показаний на рисунку:

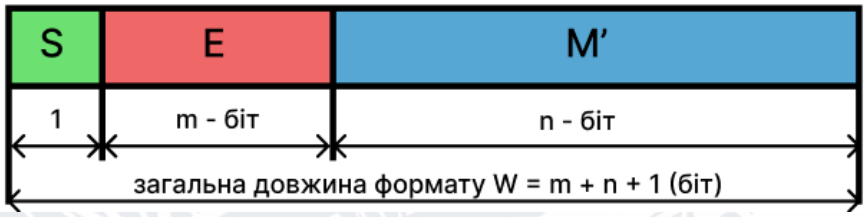

Рис. 1. Загальний формат чисел з плаваючою комою

Існує нескінченна кількість записів того чи іншого числа, тому розрізняють нормальну та нормалізовану форми запису числа.

Нормальною вважається така форма запису, в якій мантиса задовольняє умови $0 \leq |M| < 1$. Оскільки нормальних форм можна виділити нескінченну кількість , то виділяють ще одну форму запису – це нормалізовану, в якій мантиса задовольняє умови $1 \leq |M| < q = 10b$, де q – основа системи числення. Для кожного числа існує лише одна нормалізована форма запису.

Кількість розрядів експоненти визначає максимальне та мінімальне число (за модулем), яке може бути подане у заданому форматі.

Максимальні числа визначаються як:

$$x_{max} = \pm M_{max} \times 2^{emax} = \pm(2 - 2^{-n}) \times 2^{2^{m-1}-1} \approx \pm 2^{2^{m-1}}$$

А мінімальні числа відповідають

$$x_{min} = \pm M_{min} \times 2^{emin} = \pm 1 \times 2^{-(2^{m-1}-2)} \approx \pm 2^{-(2^{m-1}-2)}$$

Якщо m = 8, то

$$M_{max} \approx \pm 2128 \approx \pm 3{,}40 \cdot 1038, \quad M_{min} \approx \pm 2 - 126 \approx \pm 1{,}17 \cdot 10 - 38$$

Таким чином, динамічний діапазон чисел становить приблизно 38 + 38 = 76 порядків, що достатньо для більшості обчислень.

Кількість розрядів мантиси визначає точність поданого числа або кількість значущих знаків після коми в десятковому записі числа. За формулою: $k = (n+1) \cdot \ln 2 \approx 0{,}3 \cdot (n+1)$ можна приблизно визначити кількість значущих десяткових цифр подання.

Для подання одного десяткового розряду потрібно близько 3,32 двійкових розряди ($\log_2 10$). Якщо n=23, тоді $k \approx 7{,}22$, або 7 значущих десяткових цифр у записі числа.

У подальшому ми будемо розглядати лише двійкові числа з плаваючою комою IEEE 754 та операції з ними.

Число вважається нескінченно великим, якщо всі розряди експоненти рівні одиниці, а всі розряди мантиси рівні нулю. Отримати нескінченність можна у випадку округлення, переповнення розрядної сітки та у разі ділення на нуль. Слід пам'ятати, що:

$$\frac{x}{0} = \begin{cases} +\infty, & x > 0 \\ NaN, & x = 0 \\ -\infty, & x < 0 \end{cases}$$

Невизначеність, або "не числа" (NaN, not a number) – є результатом арифметичних операцій, під час виконання яких відбулася помилка. NaN подається як число, у якого всі біти експоненти рівні одиниці, а мантиса – ненульовий набір бітів.

1.3. Операції над двійковими числами в стандарті IEEE 754

Стандарт IEEE 754 регламентує різні арифметичні операції з числами з плаваючою комою. Ці операції включать додавання, віднімання, множення, ділення, а також операції з обробки спеціальних значень і виключень.

Для здійснення додавання і віднімання чисел з рухомою комою необхідно для початку вирівняти порядки операндів, що вимагає зсуву розділової коми в мантисі. Множення та ділення не вимагають вирівнювання порядків. Під час здійснення цих операцій можуть виникнути такі ситуації, як переповнення чи недоповнення порядку, або втрата вагомості мантиси.

Операції додавання і віднімання чисел з плаваючою комою відбувається в чотири етапи.

Крок 1. Перевірка на специфічні значення

Крок 2. Зсув мантиси для вирівнювання порядків

Крок 3. Додавання чи віднімання мантис

Крок 4. Нормалізація результату

Специфічними значеннями операндів є нуль, $\pm\infty$, sNaN, qNaN. Якщо одним з операндів – нуль, то результат додавання (віднімання) буде дорівнювати іншому операнду (або його від'ємному значенню). Якщо один з операндів $\pm\infty$, а другий – скінченне число, то результат операції також буде$\pm\infty$. Однак, якщо другий операнд також $\pm\infty$, то слід врахувати, що

$$+\infty + (+\infty) = +\infty$$

та

$$-\infty + (-\infty) = -\infty$$

а $+\infty - (+\infty) = qNaN,$ а також $-\infty + (-\infty) = qNaN.$

Зсув мантис здійснюється для того, щоб узгодити порядки операндів. З цього випливає, що над мантисами не здійснюватимуться операції, якщо порядки операндів відрізняються один від одного.

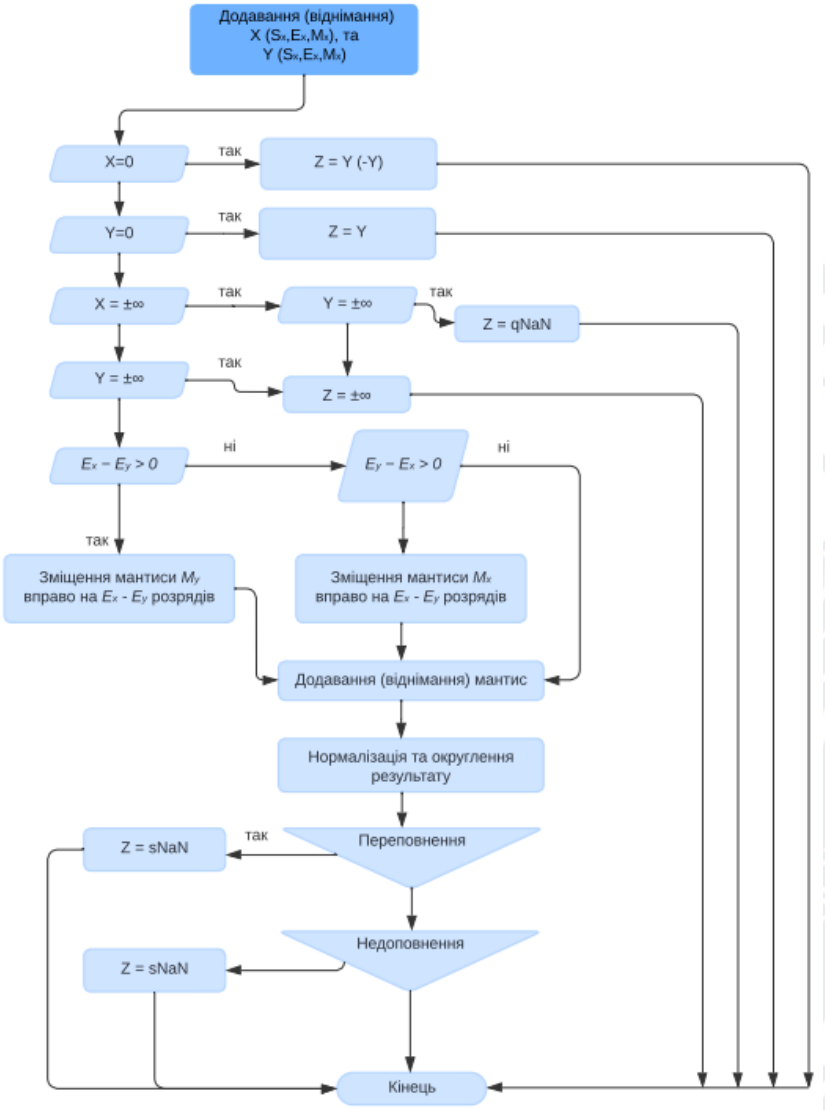

Рис. 2. Алгоритм додавання (віднімання) двійкових чисел з рухомою комою IEEE 754.

Виконання множення та ділення чисел з рухомою комою відбувається простіше, ніж додавання або віднімання, проте він має також чотири етапи:

Етап 1. Перевірка на специфічні значення операндів

Етап 2. Додавання (віднімання) експонент з врахуванням зміщень

Етап 3. Множення (ділення) мантис

Етап 4. Нормалізація результату

Специфічними значеннями операндів так як і в попередніх операціях є нуль, $\pm\infty$, sNaN, qNaN. У будь-якому випадку множення чи ділення, коли один з операндів є sNaN або qNaN, результатом завжди буде qNaN.

Знак добутку визначається знаками співмножників.

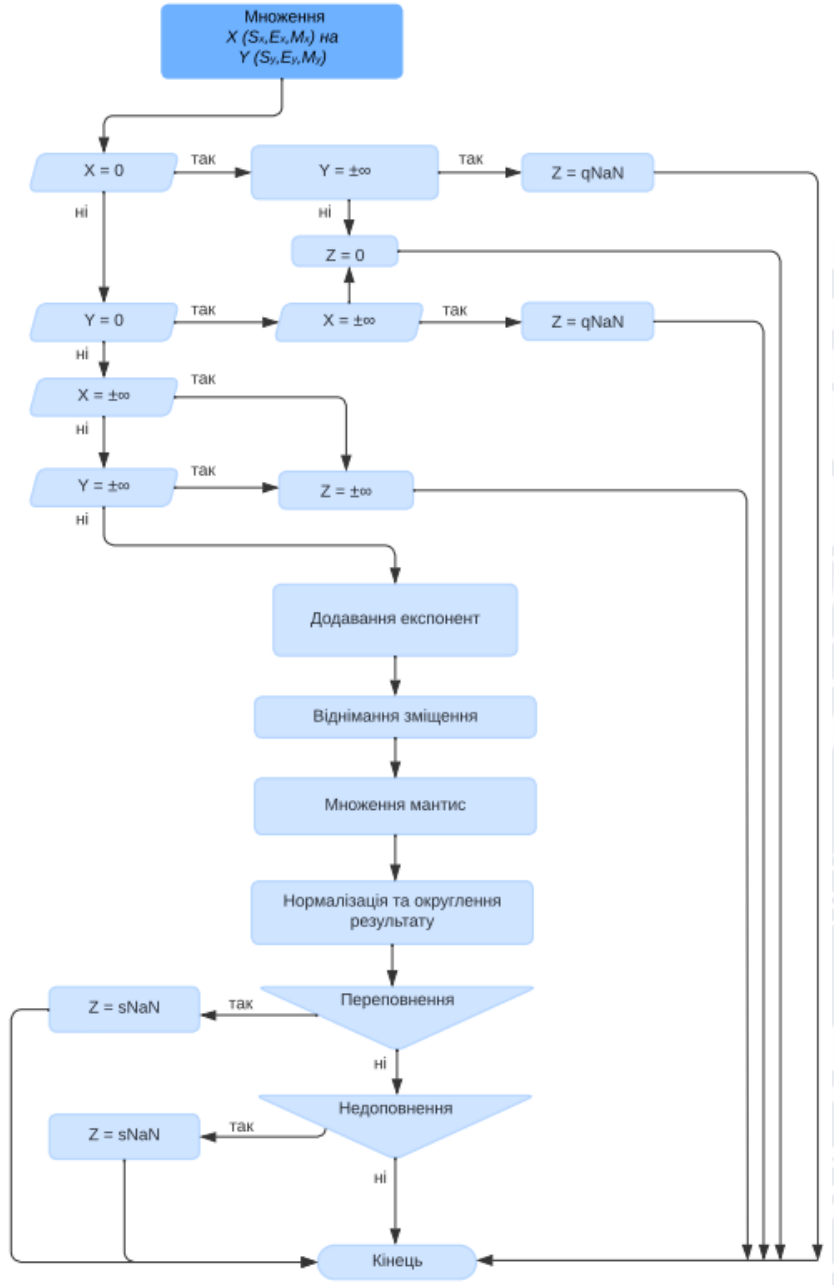

Рис. 3. Алгоритм множення двійкових чисел з рухомою комою IEEE 754.

Результат будь-якої операції над мантисами операндів зазвичай формується блоком FPU (Floating Point Unit)[20], який має вищу розрядність, ніж передбачено форматом подання. Тому для формування результату необхідно виконати його округлення. Стандартом IEEE передбачено чотири альтернативних підходи до виконання округлення:

1) округлення до найближчого числа;

2) округлення до $+\infty$;

3) округлення до $-\infty$;

4) округлення до нуля.

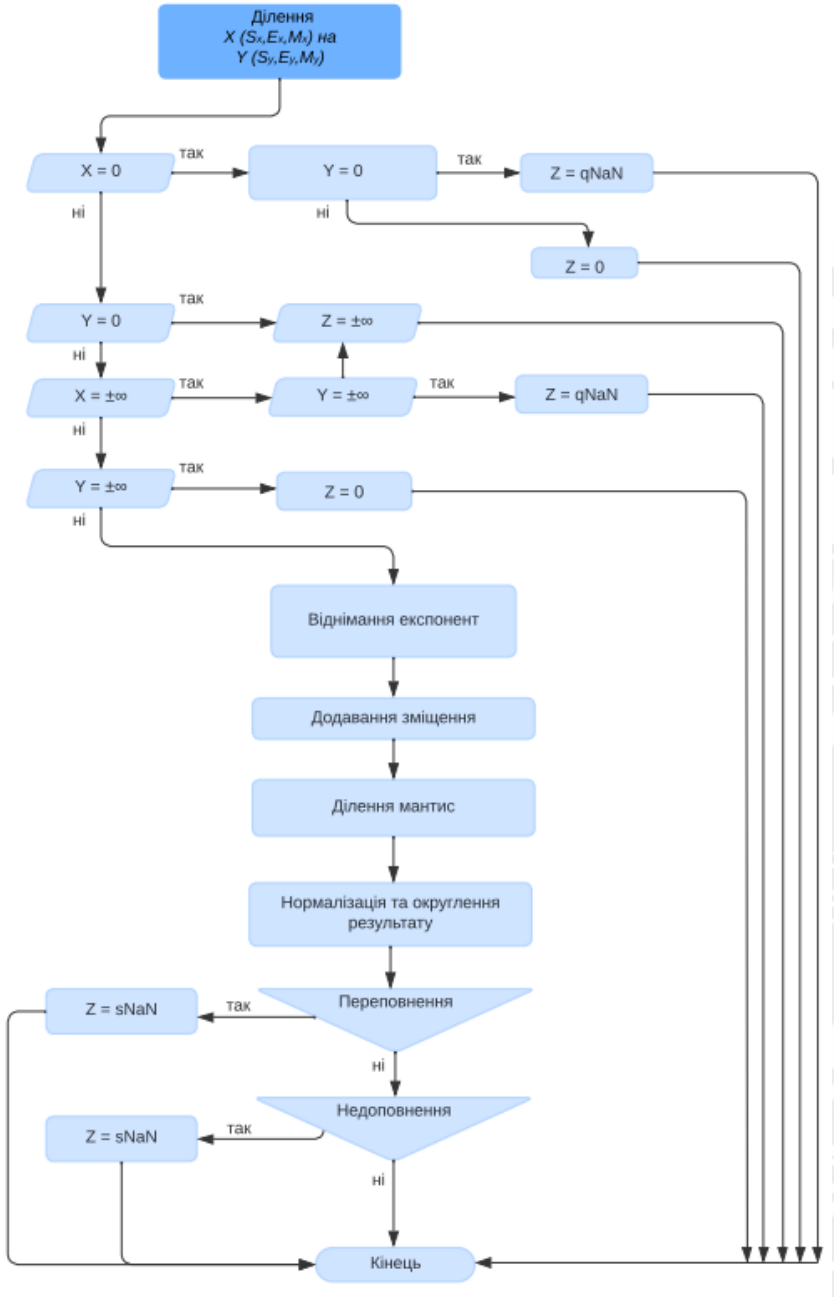

Рис. 4. Алгоритм ділення двійкових чисел з рухомою комою IEEE 754.

Розглянемо ці способи більш детально.

Округлення до найближчого числа полягає у виборі такого наближеного скінченого числа в заданому форматі подання, яке найближче до точного результату обчислення. Якщо число знаходиться точно посередині між двома представленими числами, то воно округлюється до парного числа.

Округлення чисел в десятковій системі числення не становить проблеми, проте вона виникає тоді, коли округлювання відбувається в двійковій системі числення. Ретельний аналіз обчислень в двійковій системі числення показує, що з кожним разом помилка округлень буде накопичуватись і виконання

великої кількості операцій призведе до невірних результатів. Похибку округлення, яка має здатність накопичуватись, називають заміщеною похибкою округлень.

Округлення до $+\infty$ або до $-\infty$ застосовують для реалізації методики обчислень, відомої як "арифметика інтервальних обчислень", яка має на увазі визначення верхньої межі та нижньої межі значень результату. Якщо здійснюють округлення до $+\infty$, то вибирають таке наближене скінченне число (або $+\infty$) у заданому форматі подання, яке одночасно найближче до точного результату обчислень і не менше від нього. Це означає, що число +1011100,1b буде округлене до +1011101b, а число -1011100,1b – до -1011100b. Подібним чином округлюють до $-\infty$, тобто вибирають таке наближене скінченне число (або $-\infty$) у заданому форматі подання, яке одночасно найближче до точного результату обчислення і не більше від нього. Таким чином, число +1011100,1b буде округлене до +1011100b, а число -1011100,1b – до -1011101b.

Округлення до нуля полягає у виборі такого наближеного скінченного числа в заданому форматі подання, яке одночасно найближче до точного результату обчислення і за модулем не більше від нього. Цей спосіб округлення фактично передбачає обтинання результату – всі лишні розряди просто відкидають. За таким способом округлення отриманий результат за модулем завжди буде меншим або рівним точному значенню, тому у разі виконання тривалих обчислень утворюється зміщення в бік менших значень (до 52 нуля). Накопичення такої помилки округлення може бути значно вищим, ніж у випадку округлення до найближчого, оскільки зміщена помилка округлення до нуля виникає при виконанні кожної операції, а не тільки в особливих випадках

1.4. Поняття точності і помилки обчислень чисел з рухомою комою

Точність обчислень чисел з рухомою комою характеризується тим, наскільки точно число з плаваючою комою, представлене в комп'ютері, наближує реальне (істинне) значення. У стандарті IEEE 754 точність обчислень залежить від кількості бітів, відведених для мантиси та експоненти.

Помилка обчислень чисел з рухомою комою виникає через обмежену кількість бітів для представлення чисел, що призводить до необхідності округлення. Це може призвести до втрати точності та появи похибок.

Абсолютна похибка – це різниця між точним значенням (істинним) x_{true} та обчисленим значенням x_{approx}:

$$E_{abs} = |x_{true} - x_{approx}|$$

Ця формула дозволяє визначити величину похибки у тих же одиницях виміру, що й самі числа.

Відносна похибка – це відношення абсолютної похибки до абсолютного значення точного значення x_{true}. Відносна похибка надає інформацію про похибку у відносних одиницях, що корисно для порівняння точності чисел різних порядків величини:

$$E_{rel} = \frac{|x_{true} - x_{approx}|}{|x_{true}|}$$

Ці визначення дозволяють оцінити точність обчислень та відносну значущість похибки.

Приклад розрахунку абсолютної похибки і відносної похибки

Розглянемо приклад, де точне значення числа x_{true}=3.14159265358979 (значення числа π), а обчислене значення x_{approx}=3.14.

Абсолютна похибка:

$$E_{abs} = |3.14159265358979 - 3.14| = 0.00159265358979$$

Відносна похибка:

$$E_{rel} = \frac{|3.14159265358979 - 3.14|}{|3.14159265358979|} \approx \frac{0.00159265358979}{3.14159265358979} \approx 0.000507$$

Абсолютна похибка для наближення π як 3.14 становить 0.00159265358979. Ця похибка демонструє різницю між точним значенням і

наближеним значенням у абсолютних одиницях. У контексті математичних і наукових обчислень, навіть така невелика абсолютна похибка може мати значний вплив на кінцевий результат, особливо при проведенні численних ітерацій або обчислень.

Відносна похибка в даному випадку становить приблизно 0.000507, що еквівалентно 0.0507%. Відносна похибка надає уявлення про розмір похибки відносно самого значення. Відносно точного значення π ця похибка виглядає невеликою, але вона все ще може бути значущою залежно від точності, необхідної для конкретної задачі.

Наведений приклад підкреслює, що навіть невеликі похибки округлення можуть вплинути на точність результатів обчислень. Це особливо важливо у високоточних наукових обчисленнях, де навіть незначні відхилення можуть призвести до суттєвих помилок.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ПРИКЛАДНИХ ПРИКЛАДІВ НЕКОРЕКТНИХ ОБРАХУНКІВ

2.1. Приклади некоретної роботи з цілими типами

Коротко розглянемо числові типи даних бібліотеки NumPy [21], однієї з найпопулярніших бібліотек Python для наукових обчислень. На рисунку 5 зображена ієрархічна схема класу даних numpy.generic, основного класу скалярних типів. В рамках даної роботи складові numpy.generic класи даних nympy.bool_, nympy.object_, nympy.datetime64, nympy.flexible нас цікавити не будуть.

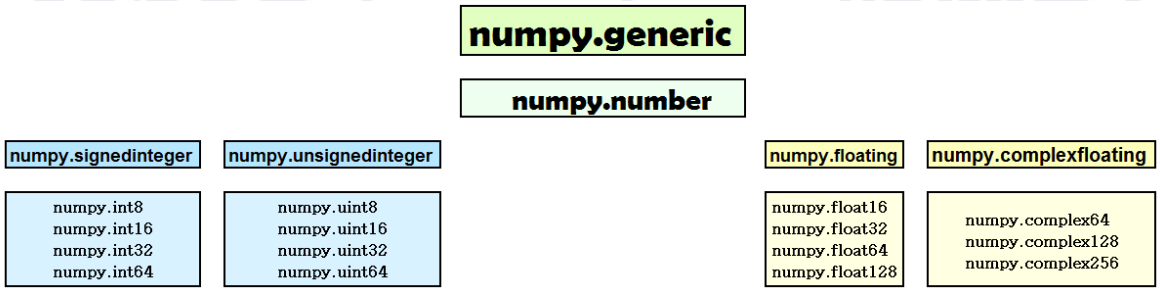

Рис. 5. Числові типи Numpy

Число плаваючою комою згідно стандарту IEEE 754, як вже було сказано, представляється у вигляді

$$number = (-1)^s \times 1.M \times 2^E$$

де S – знак числа (signum), M – мантиса (mantissa), E – порядок числа (exponent). На рисунку 2 представлений розподіл біт числа одинарної точності формату binary32 (старший біт – знаковий, порядок числа зберігається за допомогою 8 бітів, на мантису виділяється 23 біти – загалом 32 біти).

Мантиса, як можна бачити з формульного представлення числа N, нормалізована – ціла частина дорівнює одиниці. Для обчислення порядку використовується константний зсув експоненти, що для чисел одинарної точності дорівнює 127 (або 01111111 у бінарному коді).

Задля наочності, подивимось, згідно з описаними принципами IEEE 754, що станеться з представленням раціонального числа $^1/_3$. Математично у десятковій та бінарній системах числення маємо

$$^1/_3 = 0.(3)_{10} = 0.(01)_2$$

Для числа одинарної точності можемо записати

$$^1/_3 = (-1)^0 \cdot 1.01010101010101010101011 \cdot 2^{-10}$$

тобто S = 0 (число додатне), M = 01010101010101010101011 (оскільки перший відкинутий біт був одиницею, то за правилами стандарту до мантиси ми додаємо 1), E = -10 (тобто порядок дорівнює -2 у десятковій системі числення). Отримаємо float32-наближення до числа $^1/_3$ у математичному вигляді

$$(0.01010101010101010101011)_2 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} \ldots =$$

$$= \sum_{k=1}^{12} \frac{1}{2^{2k}} + \frac{1}{2^{25}} = (0.33333334326744079589843 75)_{10}$$

Отримане наближене значення числа більше за реальне значення $^1/_3$. У математичному ж сенсі все залишається коректним

$$0.(01)_2 = \sum_{k=1}^{\infty} \frac{1}{2^{2k}} = \frac{^1/_4}{1 - ^1/_4} = {}^1/_3$$

Розглянемо важливе поняття теорії комп’ютерних обчислень – машинний епсилон. Машинний епсилон – це таке числове значення, менше за яке неможливо задавати відносну точність для будь-якого алгоритму, результатом якого є дійсні числа. Близьким поняттям, але не тотожним, є поняття машинного нуля, тобто числового значення з таким від'ємним порядком, яке сприймається комп’ютером як нуль.

Фактично, для того, щоб отримати значення машинного епсилону в Python можна скористатись функцією numpy.finfo() для конкретних форматів чисел с плаваючою комою. Але видається доречним навести програмну реалізацію (лістинг 1) обчислення значень машинного епсилона та машинного нуля, щоб конструктивний спосіб зрозуміти суть даних понять.

```
1     from numpy import *
2     def machineEpsilon(float_format = float):
3       iteration_count_eps = 1
4       mach_eps = float_format(1)
5       while   float_format(1) + float_format(mach_eps)\
6             /float_format(2)  != float_format(1):
7         iteration_count_eps += 1
8         mach_eps /= float_format(2)
9       return mach_eps, iteration_count_eps
10
11    def machineZero(float_format = float):
12      iteration_count_zero = 0
13      mach_zero = float_format(1)
14      while   float_format(mach_zero)\
15            /float_format(2) != float_format(0):
16        iteration_count_zero += 1
17        mach_zero /= float_format(2)
18         return mach_zero, iteration_count_zero

Result:
          Machine Epsilon
          <class 'numpy.float16'>: (0.000977, 11)
          <class 'numpy.float32'>: (1.1920929e-07, 24)
          <class 'numpy.float64'>: (2.220446049250313e-16,  53)
          <class 'numpy.longdouble'>: (1.084202172485504434e-19,  64))

          Machine Zero
          <class 'numpy.float16'>: (6e-08, 24)
          <class 'numpy.float32'>: (1e-45, 149)
          <class 'numpy.float64'>: (5e-324, 1074)
          <class 'numpy.longdouble'>: (4e-4951, 16445)
```

Рис. 6. Лістинг 1: машинний нуль/епсілон

Ми бачимо, що конкретне машинного епсилон, як і машинного нуля, не є принциповим – важливим є порядок заданих чисел. Значення змінної iteration_count_eps – це від'ємний порядок степеня $\mathbf{2^{-iteration\_count\_eps}}$, і як ми бачимо, для машинного епсилона для відповідного типу чисел з плаваючою комою значення iteration_count на одиницю більше за кількість біт, що виділяється для значення мантиси для відповідного типу. Це є логічним, оскільки означає межі точності для відповідного числового типу.

Що стосується результатів для машинного нуля, то наведені значення є найменшими числами, менші за які вже сприймаються комп'ютером як нуль.

Тобто, насправді машинним нулем є число $2^{-(\text{iteration\_count\_zero}+1)}$. Очевидно, що отримати порядок машинного нуля можна і без комп'ютерних обчислень за формулою

$$\text{mantissa} + 2^{\text{exponent}-1},$$

де exponent та mantissa означає кількість біт для відповідного типу даних.

Розглянемо один цікавий приклад роботи з цілими числами при переповненні розрядної сітки. Сама ця проблема є добро відомою, але при підготовці фахівців важливо продемонструвати, що отримані некоректні з точки зору людини-програміста результати, тим не менш є абсолютно коректними з точки зору виконання програми в рамках існуючих стандартів. Простим прикладом є обчислення значення факторіалу деякого натурального числа n. При безпосередній реалізації в Python (функція factorial з модуля math), жодних проблем з точністю не виникає, оскільки Python за замовченням реалізований механізм роботи BigInt [22], тобто значення цілого числа не обмежується кількістю бітів і може розширюватися до межі доступної пам'яті. При цьому використання можливостей модуля numpy дозволяє докладніше дослідити нюанси обчислення факторіала при різних типах цілих чисел.

В таблиці 1 наведені результати обрахунку в Python факторіала числа n в залежності від типів цілих чисел. Коректний результат отриманий з використанням стандартного int() в Python (вище згадувана концепція BigInt).

Таблиця 1
Обчислення значення факторіалу

n	Correct value n!	int32(n!)	int64(n!)
1	1	1	1
2	2	2	2
3	6	6	6
4	24	24	24
5	120	120	120
6	720	720	720
7	5040	5040	5040
8	40320	40320	40320
9	362880	362880	362880
10	3628800	3628800	3628800
11	39916800	39916800	39916800
12	479001600	479001600	479001600
13	6227020800	1932053504	6227020800
14	87178291200	1278945280	87178291200

15	1307674368000	2004310016	1307674368000
16	20922789888000	2004189184	20922789888000
17	355687428096000	-288522240	355687428096000
18	6402373705728000	-898433024	6402373705728000
19	121645100408832000	109641728	121645100408832000
20	2432902008176640000	-2102132736	2432902008176640000
21	51090942171709440000	OverFlowError	OverFlowError

Як ми бачимо з таблиці 1, при обрахунку факторіала за допомогою типу int32, починаючи з n = 13 – результат обрахунків є некоректним. Використовуючи типи int32 та int64, починаючи з обрахунку значення 21!, в Python генерується виключення OverFlowError.

Зазначимо, що типи numpy.intc та numpy.int_ є аналогами типів int та long long в C, при цьому генерація виключення OverFlowError є особливістю саме модуля numpy, тоді як в C ми би отримали відповідні значення

int factorial(21) = – 1195114496,

long long (21) = – 4249290049419214848.

Ми отримали парадоксальний на перший погляд результат – добуток натуральних чисел дорівнює від'ємному числу. Але з точки зору обчислень у цілих числах у форматах int та long long все обраховано коректно. Реальне значення 21! у десятковій та двійковій системах числення

$21! = (51090942171709440000)_{10}$,

$21! = (1011000101000001110111110100110110101110001100010000000000000000000)_2$

Запис у бінарному коді має довжину 66 біт, тоді як тип int в C має довжину 32 біти (старший біт – знаковий), а тип long long – відповідно 64 біти. Це означає, що у бінарному коді, відкидаючи старші біти, ми реально отримаємо

int factorial(21) = $(10111000110001000000000000000000)_2$

long long (21) = $(1100010100000111011111010011011010111000110001000000000000000000)_2$

Враховуючи той факт, що ми працюємо зі знаковими цілити типами і представлені результати записані у додавальному коді, зворотна конвертація до десяткової системи числення відповідно дасть числа – 1195114496 та – 4249290049419214848, тобто розрахунок є коректним не з математичної точки зору, а відповідно до стандартів і алгоритмів роботи з цілими типами.

З метою підвищення точності розрахунків і отримання коректних результатів обчислень нерідко рекомендують замість цілих типів даних використовувати типи чисел з плаваючою комою. Інколи це може дати певний ефект, але також суттєво обмежений, і коректність отриманих результатів також вимагає обов'язкової перевірки.

В описаному прикладі обчислення факторіала числа будемо замість цілих типів використовувати типи даних з плаваючою комою.

Тип float32 перестає гарантувати точний результат вже при n = 14:

$$\text{float32}(14!) = (87178289152)_{10}$$

тоді як коректне значення 14! = 87178291200. З точки зору стандарту IEEE-754 Python знову рахує без помилок.

Після конвертації до бінарного коду

$$(87178291200)_{10} = (1010001001100001110110010100000000000)_2$$

Враховуючи формат float32 запишем

float32(87178291200) = 0 | 10100011 | 01000100110000111011001

sign exponent mantissa

тобто ми відкидаємо в молодших бітах "0100000000000", що в десятковій системі числення як раз і дорівнює 2048. Число 2048 – це похибка між реальним значенням обчислення 14! і результатом обрахунку float32(14!). Починаючи з n = 23, тип числа з плаваючою комою float64 починає повертати некоректний результат. Значення факторіалу для n > 25 є некоректним вже і для типу longdouble, при цьому ми отримаємо значення longdouble(26!) більше за реальне значення 26!, що також слідує з особливостей застосування стандарту IEEE-754:

longdouble (403291461126605635584000000) =

0 | 100000001010111 |

sign exponent

01001101100110000100100111101010001101111110111010101100100100 10

mantissa

В десятковому коді вказане число конвертується як

403291461126605635592388608.

2.2. Приклади некоретної роботи з float-числами

Перейдемо до розгляду деяких характерних прикладів некоректної поведінки обчислювальних алгоритмів про роботі з числами з плаваючою комою.

Розглянемо таку задачу: необхідно знайти мінімальне значення n, що

$$\frac{1}{m}+\frac{1}{n}=\frac{1}{2020}, \quad \begin{matrix} m \in N \\ n \in N \end{matrix}, \quad n > m$$

Вказана задача є прикладом діофантового рівняння. У загальному смислі, Діофантове рівняння – рівняння, в якому невідомі можуть набувати лише цілих значень.

Програма розв’язку вказаного рівняння представлена на лістингу 2.

```
1       import numpy as np
2       def mathExpr (num_format):
3           value = num_format(1/2020)
4           n = 1
5           while True:
6               n1 = num_format(1/n)
7               for m in range(1, n):
8                   if n1 + num_format(1/m) == value:
9                       return n, m
10              n += 1
11      float_formats = (np.float16, np.float32, np.float64,
12      np.float128)
13      for form in float_formats:

            print(f"{form}:", mathExpr (form))

Result:     <class 'numpy.float16'>: n = 4039, m = 4035
            <class 'numpy.float32'>: n = 4041, m = 4039
            <class 'numpy.float64'>: n = 4545, m = 3636
            <class 'numpy.longdouble'>:    n = 6060, m = 3030
```

Рис. 7. Листинг 2.

В результаті ми отримали чотири різних відповіді в залежності від формату чисел з плаваючою комою. При цьому коректним рішенням рівняння є пари (4545, 3636) та (6060, 3030). Якщо шукати мінімальне значення n,то нескладно показати, що має бути n = 4545.Тобто при використанні форматів float16 та

float32 ми отримаємо некоректні відповіді, а у випадку формату float128 ми отримаємо коректну відповідь з математичної точки зору, але перескочимо через мінімальне можливе значення n (n = 4545), що задовольняє задачі. Виправити цю ситуацію ми можемо, наприклад, за допомогою методу isclose з модуля math. Для цього в лістингу #1 необхідно замінити рядок 8 на команду *if math.isclose(n1+num_format(1/m), value, rel_tol=10e-10)*, і в цьому випадку результат для формату float128 буде вже n = 4545, m = 3636.

Змінимо в лістингу #1 рядок 3 на команду *value = num_format(1/10)*, і після запуску програми отримаємо результат:

<class 'numpy.float16'>: n = 30, m = 15
<class 'numpy.float32'>: n = 35, m = 14
<class 'numpy.float64'>: n = 30, m = 15
<class 'numpy.longdouble'>: ***Infinite loop***

Очевидно, відповіддю задачі є пара n = 30, m = 15 (пара n = 35, m = 14 також є коректним результатом, але мінімальне значення параметра n все ж таки дорівнює 30). При форматі подвійної точності програма потрапляє у нескінченний цикл, тобто результат в принципі не може бути обчисленим. Якщо ми будемо обчислювати вказаний приклад за допомогою функції isclose, то для випадку *rel_tol=10e-8* для всіх форматів чисел з плаваючою комою результат буде n = 30, m = 15, а для значення параметру *rel_tol* більшої точності (наприклад, вже *rel_tol=10e-9)* – для форматів float16, float64, float128 результат буде n = 30, m = 15, а для формату float32 результатом так і залишиться n = 35, m = 14.

Зважаючи на сказане, ми можемо зробити цікавий проміжний висновок: в деяких прикладах комп'ютерних обрахунків з числами з плаваючою комою задана менша точність обрахунків може давати більш коректний результат, ніж процедура підвищення точності. Цей ефект ми якраз і спостерігаємо в описаному прикладі: коректна відповідь забезпечується обчисленням з типом даних float16 (на відміну від float32), коректний результат досягається зі значенням параметру rel_tol=10e-8, а не зі значенням 0< rel_tol≤ 10e-9.

На останок розглянемо порівняння обчислення значень квадратного кореня від цілих чисел за допомогою двох операцій з модуля math: sqrt та pow. Не

дивлячись на простоту постановки задачі, обчислення квадратного кореня числа, цілого кореня числа, а особливо оберненого квадратного кореня натурального числа [23] є достатньо актуальною задачею як з точки зору коректності обчислень, так і з точки зору часової ефективностіалгоритмів. Для прикладу розглянемо діапазон чисел від 1 до 10000, тип даних чисел з плаваючою комою – стандартний float (numpy.float64). Будемо перебирати значення до тих пір, поки не знайдемо значення number таке, щоб math.sqrt(number) != math.pow(number). В таблиці 5 наведені такі числа (для прикладу менші 10000) в залежності від версії Python та компілятора. В таблиці наведені лише характерні приклади результатів комп'ютерних обрахунків пропонованої задачі. Для деяких версій(наприклад, для Python 3.6.5 [GCC 7.3.1] таких значень за зазначених обмежень немає).

Обчислення функцій sqrt та pow відбувається за відповідними алгоритмами. Не вдаючись до технічних нюансів реалізації зазначених алгоритмів, поясниму причину неспівпадіння значень для відповідних чисел.

math.sqrt(2921) =
54.046276467486642047788336640223860740661621093 75

binary(float64) 01000000 01001011 00000101 11101100
01100011 00100101 00110110 11111011

math.pow(2921, 0.5) =
54.046276467486634942360979039222002029418945312 50

binary(float64) 01000000 01001011 00000101 11101100
01100011 00100101 00110110 11111010

Аналізуючи бінарне представлення результатів обчислень у форматі float64, ми бачимо що різниця у молодшому біті мантиси. Причина в тому, що перший відкинутий біт у представленні числа є 1, і при розрахунку значення math.sqrt(2921) цей факт враховується і додається 1 до молодшого біту мантиси, а при обрахунку math.pow(2921, 0.5) – не додається. Наприклад, в Python 3.6.5 [GCC 7.3.1] обидва значення обрахувуються з врахуванням збільшення молодшого біта.

ВИСНОВКИ

Проблема надійності комп’ютерних обчислень є однією з фундаментальних проблем інформатики, оскільки лежить на стику прикладної математики, з одного боку, і фізичних і технічних обмежень комп’ютерної техніки, з іншого. Уникнення основ двійкової арифметики та принципів стандарту представлення чисел у пам’яті комп’ютера може істотно вплинути на правильність як теоретичних досліджень (якщо для моделювання використовувалася комп’ютерна техніка), так і просто на якість продукту при розробці прикладних програм.

Проблеми, які можуть виникнути через некоректне представлення в пам'яті комп'ютера, добре відомі. Іноді це тонкі проблеми, які не мають істотного впливу в якісному сенсі на конкретний результат. Тому в прикладній діяльності розробники програмного забезпечення, а також академічні дослідники часто ігнорують проблеми обчислення з плаваючою комою та особливості стандарту IEEE-754. Однак відомі випадки, коли необережне поводження з комп'ютерним представленням чисел призводило до серйозних трагедій.

В роботі розглядаються основні моменти, пов’язані з проблематикою комп’ютерних обчислень в рамках стандарту IEEE-754, основний технічний стандарт формату представлення чисел з рухомою комою (обговорення можливих альтернатив стандарту IEEE-754, таких як, наприклад, Posix виходить за межі представленої роботи).

Обмеження на обсяг пам’яті, що виділяється комп’ютерною технікою для обробки числа накладає принципові обмеження на можливості та логіку організації комп’ютерних обрахунків, що також потребує залучення специфічних математичних методів дослідження.

В роботі представлені результати досліджень авторів у галузі методичних основ викладання навчальних дисциплін для студентів спеціальності Computer Science. Автори зосередили свою увагу на темі точності комп’ютерних обрахунків різнопланових задач. Вказана проблема є актуальної не лише при

виконанні інженерних та наукових обчислень, але й, як показано в статті, і при розв'язку достатньо простих тренувальних задач. Автор зосередився на дослідженні вказаної проблематики на прикладі модуля numpy, як одного з основних програмних інструментів для сучасних наукових розрахунків.

Автор представленого дослідження зосередились на розгляді достатньо простих прикладів комп'ютерних обрахунків, що є повністю коректними з точки зору стандартів роботи комп'ютера з числами та арифметичними операціями, але при цьому результати є некоректними з точки зору людини-виконавця. Продемонстровано, що математично коректні обрахунки часто неможливо коректно відтворити в рамках стандарту IEEE-754 через принципові обмеження пам'яті, що виділяється для представлення конкретного числа.

В роботі також наведені парадоксальні на перший погляд приклади, коли менша точність обрахунків дозволяють отримати більш коректний результат. Подібне твердження вступає в протиріччя з класичними практиками методів обчислень, і потребує більш ґрунтовного вивчення саме специфіки комп'ютерних обрахунків.

Для професіонала Computer Science важливим є не лише розуміння основних принципів роботи з форматом чисел з плаваючою точкою з позицій загальноприйнятого стандарту IEEE, але й знання, хоч і на базовому рівні, можливих сучасних альтернатив, таких, як, наприклад, концепція posix.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] U. Kulisch, R. Hammer , D. Ratz , and M. Hocks, Numerical Toolbox for Verified Computing I. Basic Numerical Problems. Theory, Algorithms, and Pascal-XSC Programs, Springer Berlin, Heidelberg, 1993. doi:10.1007/978-3-642-78423-1

[2] R. Skeel, "Roundoff Error and the Patriot Missile", SIAM News, 25(4), 1992, 11.

[3] T. Sağlam, "Deadly Round-Off Error: Failure of the Patriot System in Dhahran 1991", Awarded Proseminar and Seminar Papers at the Chair of Software Design and Quality, 1, 2016.

[4] ARIANE 5. Flight 501 Failure, 1996. URL:https://www-users.cse.umn.edu/~arnold/disasters/ariane5rep.html

[5] G. Le Lann, "An analysis of the Ariane 5 flight 501 failure-a system engineering perspective", Proceedings International Conference and Workshop on Engineering of Computer-Based Systems, 1997, 339-346. https://doi.org/10.1109/ECBS.1997.581900

[6] Disasters due to rounding error, 2022. URL:https://web.ma.utexas.edu/users/arbogast/misc/disasters.html

[7] W. Kahan, "Further remarks on reducing truncation errors", Communications of the ACM, 1965, 8 (1): 40. https://doi.org/10.1145/363707.363723

[8] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic", ACM Computing Surveys, 23 (1), 1991, pp. 5-48. https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf

[9] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod etc., "Handbook of Floating-Point Arithmetic", Birkhäuser Basel, 2018. https://doi.org/10.1007/978-3-319-76526-6

[10] S. Boldo, C.-P. Jeannerod, G. Melquiond and J.-M. Muller, "Floating-point arithmetic", Acta Numerica, 32 (2023), 203–290. https://doi.org/10.1017/S0962492922000101

[11] D.M. Russinoff, Formal Verification of Floating-Point Hardware Design, Springer Cham, 2019. https://doi.org/10.1007/978-3-319-95513-1

[12] D. Monniaux, "The pitfalls of verifying floating-point computations", ACM Transactions on Programming Languages and Systems, 30 (3), 12, pp 1-41 https://doi.org/10.1145/1353445.1353446

[13] P. Dechaumphai, and N. Wansophark, "Numerical Methods in Science and Engineering Theories with MATLAB, Mathematica, Fortran, C and Python Programs", Alpha Science International, 2022.

[14] C. Fuhrer, J.E. Solem, and O. Verdier, "Scientific Computing with Python: High-performance scientific computing with NumPy, SciPy, and pandas", Packt Publishing, 2021.

[15] R. Johansson, "Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib", Apress, 2019. https://doi.org/10.1007/978-1-4842-4246-9

[16] O. Vietrov and R. Bilous, "Special methods of increasing the accuracy of computer calculations", 2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek) (2022), 1–5. https://doi.org/10.1109/KhPIWeek57572.2022.9916383

[17] O. Vietrov and R. Bilous, "Study of the Convergence of Muller's Sequence Computer Calculations", 2021 IEEE 3rd Ukraine Conference on Electrical and Computer Engineering (UKRCON) (2021), 547–551. https://doi.org/10.1109/UKRCON53503.2021.9575546

[18] Мудренко О.Ю. Проблеми точності комп'ютерних обрахунків / О.Ю. Мудренко // Вісник студентського наукового товариства Донецького національного університету імені Василя Стуса. Випуск 16. Том 1 / ред. кол. М.І. Прихненко (голова) та ін. Вінниця: ДонНУ імені Василя Стуса, 2024. – Вип. 16, Т. 1. – С. 145-188. https://jvestnik-sss.donnu.edu.ua/article/view/15815/15717

[19] "IEEE Standard for Floating-Point Arithmetic" in IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019. https://doi.org/10.1109/IEEESTD.2019.8766229

[20] M. Cowlishaw, "A summary of densely packed decimal encoding", IEE Proceedings – Computers and Digital Techniques, 2002, pp 1350-2387.

[21] Data types. https://numpy.org/doc/stable/user/basics.types.html

[22] Integer Objects. https://docs.python.org/3/c-api/long.html

[23] C.J. Walczyk, L. Moroz, and J. Cieslinski, "A Modification of the Fast Inverse Square Root Algorithm", Computation, 2019, 7(3): 41. https://doi.org/10.3390/computation7030041