http://www.moderntechno.de/index.php/meit/article/view/meit36-00-031
DOI: 10.30890/2567-5273.2024-36-00-031

# METHODOLOGY FOR DETERMINING ERRORS IN SIMULATION MODELING SYSTEMS, DEVELOPMENT AND IMPLEMENTATION OF HIGH-PRECISION COMPUTATION METHODS USING LONG ARITHMETIC

# МЕТОДИКА ВИЗНАЧЕННЯ ПОХИБОК В СИСТЕМАХ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ, РОЗРОБКА ТА РЕАЛІЗАЦІЯ МЕТОДІВ ВИСОКОТОЧНИХ ОБЧИСЛЕНЬ З ВИКОРИСТАННЯМ ДОВГОЇ АРИФМЕТИКИ.

**Trutniev S.H. / Трутнєв С.Г.**
*ORCID: 0000-0002-6107-7920*
**Levchenko A. O. / Левченко А.О.**
*Ph.D in Engineering, SRF / к.т.н., ст. наук. спів.*
*ORCID: 0000-0001-5550-0027*
*Military Academy (Odesa)*
*10 Fontanska doroga St., Odesa 65009*
*Військова Академія (м. Одеса)*
*вул. Фонтанська дорога 10, Одеса 65009*
**Sharipova I.V. / Шаріпова І.В.**
*ORCID: 0000-0003-0521-1299*
*Odessa I.I. Mechnikov National University*
*2 Vsevoloda Zmiienka St., Odesa, 65082*
*Одеський національний університет ім. І.І. Мечникова*
*вул. Всеволода Змієнка, 2, Одеса, 65082*

***Анотація*** *У статті представлено дослідження шляхів підвищення точності обчислень у системах імітаційного моделювання за рахунок впровадження довгої арифметики. Проаналізовано обмеження традиційних типів даних з плаваючою комою (IEEE 754) та їх вплив на достовірність результатів моделювання складних динамічних систем. Розроблено ефективну структуру зберігання чисел великої розрядності на основі масиву з основою системи числення 10^9, що забезпечує оптимальний баланс між використанням пам'яті та простотою операцій. Запропоновано та реалізовано алгоритми основних математичних операцій (додавання, віднімання, множення, ділення) для роботи з числами довільної розрядності. Проведено порівняльний аналіз точності обчислень між класичними типами даних та розробленою системою довгої арифметики. Представлено практичну реалізацію запропонованих методів на мові Python. Результати дослідження демонструють суттєве підвищення точності обчислень при використанні довгої арифметики порівняно з традиційними типами даних, що особливо важливо для складних наукових та інженерних розрахунків, де накопичення похибок округлення може призвести до значних відхилень кінцевих результатів.*

***Ключові слова:*** *імітаційне моделювання, довга арифметика, високоточні обчислення, похибки, двійкова арифметика, представлення даних з плаваючою комою, структури даних, прогнозування, базові арифметичні операції.*

## Вступ.

У сучасних комп’ютерних системах точність обчислень відіграє критичну роль, оскільки від неї безпосередньо залежить достовірність результатів моделювання складних процесів та систем [1]. Особливо це стосується моделювання динамічних систем, де навіть незначні похибки можуть призвести до суттєвих відхилень кінцевих результатів через ефект накопичення помилок при багатократних ітераціях. Традиційні методи обчислень [2], засновані на стандарті IEEE 754, хоча і широко використовуються в системах моделювання,

але мають суттєві обмеження щодо точності та стабільності результатів, особливо при тривалих симуляціях та складних математичних операціях.

Актуальність дослідження зумовлена зростаючою складністю імітаційних моделей та підвищенням вимог до їх точності в різних галузях: від моделювання фізичних процесів та хімічних реакцій до симуляції складних економічних систем, прогнозування кліматичних змін та моделювання бойових дій. Існуючі рішення, що базуються на стандартних типах даних з плаваючою комою, часто не забезпечують необхідного рівня точності при тривалих обчисленнях, характерних для імітаційного моделювання. Це особливо критично при моделюванні систем з високою чутливістю до початкових умов, де навіть мінімальні похибки округлення можуть призвести до значних розбіжностей між модельованою та реальною поведінкою системи.

Метою даного дослідження є розробка та аналіз методів реалізації довгої арифметики для забезпечення високоточних обчислень у системах імітаційного моделювання. Особлива увага приділяється оптимізації структур даних та алгоритмів для ефективної роботи з числами довільної розрядності в контексті специфічних вимог імітаційного моделювання, включаючи необхідність збереження точності при багатократних ітераціях та складних математичних операціях. Результати дослідження мають важливе практичне значення для розробки систем моделювання, де критичною вимогою є забезпечення максимальної достовірності результатів симуляції при тривалих обчисленнях.

Запропоновані методи спрямовані на вирішення фундаментальної проблеми точності в системах імітаційного моделювання, що дозволить підвищити надійність та достовірність результатів моделювання в широкому спектрі наукових та інженерних застосувань: від проектування складних технічних систем та прогнозування поведінки природних явищ до моделювання військових операцій та оцінки ефективності систем озброєння.

**Основний текст**

***1.1 Аналіз останніх досліджень та публікацій.***

Це фундаментальне дослідження [3] присвячене аналізу проблем арифметики з плаваючою комою в сучасних наукових обчисленнях. Автори детально розглядають особливості реалізації IEEE 754 на різних архітектурах, аналізують причини виникнення та накопичення похибок округлення, а також пропонують нові методи їх компенсації. Особлива увага приділяється проблемам втрати точності при операціях з числами різних порядків та методам оптимізації обчислень для зменшення впливу похибок округлення. Дослідження також включає практичні рекомендації щодо вибору форматів даних та алгоритмів для різних типів обчислювальних задач.

Робота [4] фокусується на аналізі похибок у методах чисельного інтегрування, які широко використовуються в імітаційному моделюванні. Автор представляє новий підхід до оцінки локальних та глобальних похибок різних методів інтегрування, включаючи методи Рунге-Кутти та багатокрокові методи. Дослідження містить порівняльний аналіз ефективності різних адаптивних схем контролю кроку та пропонує інноваційні алгоритми для автоматичного вибору оптимальних параметрів інтегрування. Особливу цінність представляють

практичні рекомендації щодо вибору методів для різних типів диференціальних рівнянь.

Дослідження [5] розкриває вплив паралельних обчислень на точність числових результатів. Автор детально аналізує проблеми, що виникають при розподілених обчисленнях, включаючи недетермінованість порядку операцій та різну точність обчислень на різних вузлах. У роботі представлені нові методи синхронізації та контролю точності в паралельних системах, а також запропоновані алгоритми для забезпечення стабільності результатів при масштабуванні обчислень. Особлива увага приділяється методам виявлення та компенсації похибок, специфічних для паралельних архітектур.

У своїй роботі [6] автор досліджує специфічні похибки, що виникають на різних обчислювальних архітектурах. Автор проводить глибокий аналіз впливу апаратних особливостей на точність обчислень, включаючи різні реалізації FPU (Floating-Point Unit) та особливості кешування даних. Дослідження містить порівняльний аналіз точності обчислень на CPU, GPU та спеціалізованих прискорювачах, а також пропонує методи оптимізації коду для мінімізації архітектурно-залежних похибок. Робота включає практичні рекомендації щодо вибору апаратної платформи для різних типів обчислювальних задач.

Дослідження [7] представляє систематичний підхід до контролю похибок у системах імітаційного моделювання. Автор розробляє комплексну методологію, яка охоплює всі етапи моделювання - від вибору математичної моделі до верифікації результатів. У роботі представлені нові методи оцінки достовірності результатів, включаючи статистичні підходи та методи машинного навчання для виявлення аномалій. Особливу цінність представляють запропоновані автором критерії вибору рівня деталізації моделі та методи оцінки впливу різних джерел похибок на кінцевий результат моделювання.

***1.2 Постановка завдання дослідження***

На основі аналізу останніх наукових публікацій у галузі високоточних обчислень встановлено критичну необхідність розробки спеціалізованого підходу до організації обчислень. Існуючі обмеження стандарту IEEE 754 та накопичення похибок при послідовних операціях створюють суттєві перешкоди для досягнення надвисокої точності обчислень з великою десяткових знаків. Для вирішення цих завдань необхідно розробити підхід, який забезпечить: ефективну реалізацію алгоритмів довгої арифметики з оптимальним використанням пам'яті, спеціалізовані методи виконання основних математичних операцій з багатократною точністю. Такий підхід дозволить подолати фундаментальні обмеження традиційної арифметики з плаваючою комою та забезпечити якісно новий рівень точності для широкого спектру наукових та інженерних задач.

**Виклад основного матеріалу.**

Для початку проведем порівняння типів даних з плаваючою комою (таблиця 1), які використовуються у мовах програмування високого рівня .

Коли мова йде про типи даних, що "обмежені лише доступною пам'яттю системи" (як Java BigDecimal чи Python decimal.Decimal), це означає, що вони можуть зберігати числа з практично необмеженою точністю та величиною, але

їх реальні межі визначаються фізичними характеристиками комп'ютера - насамперед обсягом доступної оперативної пам'яті (RAM) та архітектурою системи (32 чи 64 біти). На відміну від фіксованих типів даних (як float чи double), які завжди займають однаковий обсяг пам'яті (наприклад, 4 чи 8 байт), ці динамічні типи виділяють пам'ять за потребою, зберігаючи кожну цифру числа окремо, що дозволяє досягти надзвичайно високої точності для математичних, фінансових чи наукових обчислень, але може призвести до суттєвого споживання пам'яті при роботі з дуже великими числами чи високою точністю [8]. Виходячи із цього з метою раціонального використання обчислювальних та апаратних спроможностей комп'ютера виникає необхідність самостійно реалізувати потрібні типи даних та математичні операції.

**Таблиця 1 – Типи даних які вікористовуються в мовах програмування високого рівня та їх характеристики.**

| Мова програмування | Тип даних | Розмір (біт) | Значущі цифри | Діапазон |
|---|---|---|---|---|
| C++ | float | 32 | 6-7 | ±1.18E-38 до ±3.4E+38 |
| C++ | double | 64 | 15-17 | ±2.23E-308 до ±1.80E+308 |
| C++ | long double (x86) | 80 | 19-20 | ±3.4E-4932 до ±1.1E+4932 |
| C++ | long double (x64) | 128* | 33-36 | ±3.4E-4932 до ±1.1E+4932 |
| C | float | 32 | 6-7 | ±1.5E-45 до ±3.4E+38 |
| C | double | 64 | 15-17 | ±5.0E-324 до ±1.8E+308 |
| C | decimal | 128 | 28-29 | ±1.0E-28 до ±7.9E+28 |
| Java | float | 32 | 6-7 | ±1.4E-45 до ±3.4E+38 |
| Java | double | 64 | 15-17 | ±4.9E-324 до ±1.8E+308 |
| Java | BigDecimal | Необмежений** | Настроюваний | Необмежений** |
| Python | float | 64 | 15-17 | ±2.23E-308 до ±1.80E+308 |
| Python | decimal.Decimal | Настроюваний | Настроюваний | Настроюваний |
| Fortran | REAL*4 (REAL) | 32 | 6-7 | ±1.18E-38 до ±3.4E+38 |
| Fortran | REAL*8 (DOUBLE) | 64 | 15-17 | ±2.23E-308 до ±1.80E+308 |
| Fortran | REAL*16 (QUAD) | 128 | 33-34 | ±3.4E-4932 до ±1.1E+4932 |

** Розмір може відрізнятися залежно від компілятора та платформи*

*** Обмежений лише доступною пам'яттю системи*

*Альтернативний опис роботи з довгими числами:*

При роботі з числами великої розрядності ефективним рішенням є використання масивів для їх представлення [9]. Існує кілька підходів до організації такого представлення. Можна зберігати кожну десяткову цифру в окремому елементі масиву, але це неефективно використовує пам'ять. Більш оптимальним є використання системи числення з основою, що відповідає максимальному значенню типу даних (наприклад, 256 для байта або 65536 для short). Це забезпечує максимальну щільність зберігання, але ускладнює операції введення-виведення в десятковому форматі.

Практичним компромісом є використання масиву елементів типу long, де кожен елемент зберігає число від 0 до 999999999, фактично представляючи цифру в системі числення з основою 10^9. Це дозволяє зберегти простоту введення-виведення і при цьому ефективно використовувати пам'ять. Важливим

аспектом є порядок зберігання цифр - зазвичай молодші розряди розміщують на початку масиву, що спрощує виконання арифметичних операцій.

Для повного представлення числа структура даних повинна містити не лише масив цифр, але й додаткову інформацію: кількість значущих елементів, знак числа та положення десяткової коми. Це дозволяє ефективно працювати як з цілими, так і з дробовими числами довільної точності [10].

Отже в остаточному варіанті наше довге число -18 446 744 073 709 551 615,1 буде записане в масиві у вигляді (таблиця 2):

**Таблиця 2 – Варіант подання числового значення у вигляді масиву.**

| ***Номер елемента масиву*** | | | | | |
|---|---|---|---|---|---|
| **1** | **2** | **3** | **4** | **5** | **6** |
| 3 | 1 | 1 | 95516151 | 467440737 | 184 |
| Розмір числа | Знак числа | Положення коми | 1 | 2 | 3 |

де – в 1-й елемент масиву записується розмір числа; в 2-й елемент масиву записується знак числа (1 – від'ємне, 0 – додатне); в 3-й елемент масиву записується положення коми в числі;

в 4,5 та 6-й елемент масиву саме число з основою системи числення 1 000 000 000, при цьому число записується навпаки, тобто в першому елементі масиву зберігається молодша цифра (кількість одиниць), у другому — передостання по старшинству (кількість "десятків" системи числення) і т.д. Код програмної реалізації (рисунок 1)

```python
def input_long_number(number_str):
    Видаляємо всі пробіли з числа
    number_str = number_str.replace(" ", "")
    Визначаємо знак числа
    sign = 1 if number_str[0] == '-' else 0
    if sign:
        number_str = number_str[1:]  Видаляємо знак мінус
    Визначаємо положення коми
    decimal_position = 0
    if '.' in number_str:
        decimal_position = len(number_str) - number_str.index('.') - 1
        number_str = number_str.replace(".", "")  Видаляємо кому
    Доповнюємо число нулями справа, щоб довжина була кратна 9
    while len(number_str) % 9 != 0:
        number_str = '0' + number_str
    Розбиваємо число на групи по 9 цифр
    groups = []
    for i in range(len(number_str)-9, -1, -9):
        groups.append(int(number_str[i:i+9]))
    Створюємо результуючий масив
    result = [0] * (len(groups) + 3)  +3 для розміру, знаку та положення коми
    Заповнюємо службові поля
    result[0] = len(groups)  Розмір числа
    result[1] = sign  Знак числа
    result[2] = decimal_position  Положення коми
    Заповнюємо число
    for i in range(len(groups)):
        result[i + 3] = groups[i]
    return result
def print_long_number(number_array):
    Виводимо службову інформацію
    print(f"Розмір числа: {number_array[0]}")
    print(f"Знак числа: {'-' if number_array[1] == 1 else '+'}")
    print(f"Положення коми: {number_array[2]}")
    Виводимо саме число
    print("Число по розрядам (від молодших до старших):")
    for i in range(3, len(number_array)):
        print(f"Розряд {i-2}: {number_array[i]:09d}")
    Відновлюємо оригінальне число
    number_str = ""
    for i in range(len(number_array)-1, 2, -1):
        number_str += f"{number_array[i]:09d}"
    Видаляємо ведучі нулі
    number_str = number_str.lstrip('0')
    Додаємо знак
    if number_array[1] == 1:
        number_str = '-' + number_str
    Додаємо кому
    if number_array[2] > 0:
        number_str = number_str[:-number_array[2]] + '.' + number_str[-number_array[2]:]
    print(f"\nПовне число: {number_str}")
Приклад використання
test_number = "-18446744073709551615.1"
result = input_long_number(test_number)
print_long_number(result)
```

**Рисунок 1 - Програмний код реалізації введення-виведення довгих чисел на Python**

Цей код:

1. Функція `input_long_number`:
   - Приймає число як рядок
   - Визначає знак числа
   - Визначає положення коми
   - Розбиває число на групи по 9 цифр
   - Повертає масив у потрібному форматі
2. Функція `print_long_number`:
   - Виводить службову інформацію
   - Показує число по розрядам
   - Відновлює і виводить повне число

Приклад виводу:

Розмір числа: 3

Знак числа: -

Положення коми: 1

Число по розрядам (від молодших до старших):

Розряд 1: 095516151

Розряд 2: 467440737

Розряд 3: 000000184

Повне число: -18446744073709551615.1

Цей код обробляє:

Додатні та від'ємні числа;

Цілі числа та числа з комою;

Числа різної довжини;

Автоматично доповнює групи нулями де потрібно.

**Розробка порядку здійснення математичних операцій для чисел великої розрядності.**

Розробка порядку здійснення математичних операцій для чисел великої розрядності

1. Додавання чисел великої розрядності

Алгоритм додавання:

Підготовчий етап:
   - Нормалізація чисел (вирівнювання десяткових частин)
   - Аналіз знаків чисел
   - Визначення більшого числа за модулем

Процес додавання:
   - Додавання виконується від молодших розрядів до старших
   - Враховується перенесення в старший розряд при переповненні (≥ 10^9)
   - Знак результату визначається знаком більшого числа

Код програмної реалізації (рисунок 2)

2. Віднімання чисел великої розрядності

Алгоритм віднімання:

Підготовка:
   - Нормалізація чисел
   - Визначення більшого числа

- Встановлення знаку результату

Процес віднімання:

- Віднімання від молодших до старших розрядів
- Обробка займу з старшого розряду
- Формування результату з коректним знаком

Код програмної реалізації (рисунок 3)

```python
def add_numbers(num1, num2):
    Нормалізація
    norm1, norm2 = normalize_numbers(num1, num2)
        Різні знаки -> віднімання
    if norm1[1] != norm2[1]:
        return subtract_numbers(...)
        Додавання розрядів
    result = [...]
    carry = 0
    for i in range(3, max_size + 3):
        sum = carry + norm1[i] + norm2[i]
        result[i] = sum % 1000000000
        carry = sum // 1000000000
```

**Рисунок 2 - Програмний код реалізації додавання довгих чисел на Python**

```python
def subtract_numbers(num1, num2):
    Визначення більшого числа
    if compare_numbers(num1, num2) < 0:
        num1, num2 = num2, num1
        change_sign = True
        Віднімання розрядів
    borrow = 0
    for i in range(3, len(num1)):
        diff = num1[i] - num2[i] - borrow
        if diff < 0:
            diff += 1000000000
            borrow = 1
```

**Рисунок 3 - Програмний код реалізації віднімання довгих чисел на Python**

3. Множення чисел великої розрядності

Алгоритм множення:

Підготовчий етап

- Визначення знаку результату
- Створення масиву для результату
- Нормалізація множників

Процес множення:

- Множення кожного розряду першого числа на всі розряди другого
- Додавання проміжних результатів з урахуванням позиції
- Обробка переносів між розрядами

Код програмної реалізації (рисунок 4)

```python
def multiply_numbers(num1, num2):
    Знак результату
    result_sign = 1 if num1[1] != num2[1] else 0
        Створення масиву результату
    result = [num1[0] + num2[0], result_sign, ...]
    Множення розрядів
    for i in range(3, len(num1)):
        carry = 0
        for j in range(3, len(num2)):
            product = num1[i] * num2[j] + carry
            result[i+j-3] += product % 1000000000
            carry = product // 1000000000
```

**Рисунок 4 - Програмний код реалізації множення довгих чисел на Python**

4. Ділення чисел великої розрядності
Алгоритм ділення:
Попередні перевірки:
- Перевірка ділення на нуль
- Визначення знаку результату
- Нормалізація чисел

Процес ділення:
- Пошук першого наближення частки
- Послідовне уточнення результату
- Контроль точності
- Округлення результату

Код програмної реалізації (рисунок 5)

```python
def divide_numbers(num1, num2):
    Перевірка ділення на нуль
    if is_zero(num2):
    raise ValueError("Division by zero")
    Знак результату
    result_sign = 1 if num1[1] != num2[1] else 0
    Процес ділення
    quotient = [...]
    remainder = num1.copy()
    while compare_numbers(remainder, num2) >= 0:
    Знаходження частки
   temp = find_quotient(remainder, num2)
   quotient = add_numbers(quotient, temp)
```

**Рисунок 5 - Програмний код реалізації ділення довгих чисел на Python**

**Визначення показника втрати точності при проведені математичних операцій з класичними типами даних та довгою арифметикою**

При порівнянні точності обчислень між класичними типами даних (IEEE 754) та довгою арифметикою спостерігаються суттєві відмінності. Класичні

типи даних, такі як float (6-7 значущих цифр), double (15-17 значущих цифр) та long double (19-20 значущих цифр), мають фіксовану точність та схильні до накопичення похибок при послідовних операціях. Наприклад, при обчисленні числа π типом double отримуємо значення 3.141592653589793, тоді як реальне значення містить набагато більше знаків. Або при діленні 1 на 3 отримуємо 0.333333333333333, що є наближенням до реального періодичного дробу.

На відміну від класичних типів, довга арифметика забезпечує теоретично необмежену точність, яка обмежується лише доступною пам'яттю системи. Це дозволяє працювати з числами, що мають сотні і тисячі знаків після коми, без втрати точності при послідовних операціях. Показовим прикладом є обчислення числа e, де довга арифметика дає результат 2.71828182845904523536028747135266249775724709369995957496696762772407663035 3..., в той час як double обмежується значенням 2.718281828459045. При цьому важливо враховувати, що підвищена точність довгої арифметики досягається за рахунок більшого використання пам'яті та нижчої швидкодії порівняно з класичними типами даних.

**На основі проведеного дослідження можна зробити наступні висновки:**

Проведений аналіз останніх публікацій виявив критичну необхідність розробки спеціалізованих методів для високоточних обчислень, що виходять за межі можливостей стандарту IEEE 754. Запропонований підхід з використанням довгої арифметики дозволяє подолати фундаментальні обмеження традиційних типів даних з плаваючою комою.

Розроблена структура зберігання довгих чисел, що базується на масиві з основою системи числення 10^9, забезпечує оптимальний баланс між ефективністю використання пам'яті та простотою операцій введення-виведення. Реалізовані алгоритми основних математичних операцій (додавання, віднімання, множення та ділення) демонструють стабільну роботу з числами довільної розрядності.

Порівняльний аналіз точності обчислень показав суттєву перевагу довгої арифметики над класичними типами даних. Якщо традиційні типи обмежені фіксованою кількістю значущих цифр (6-7 для float, 15-17 для double), то довга арифметика забезпечує теоретично необмежену точність, обмежену лише доступною пам'яттю системи. Це особливо важливо для складних наукових розрахунків, де накопичення похибок округлення може призвести до значних відхилень кінцевого результату.

Практична реалізація підтвердила свою працездатність, проте потребує детальної оцінки ефективності. Ключовим показником ефективності виступає обчислювальна складність, яку можна оцінювати двома способами: через підрахунок базових математичних операцій або через вимірювання часу виконання. При оцінці базових операцій важливо розділяти прості арифметичні дії (додавання/віднімання) та складніші (множення/ділення), оскільки останні потребують більше обчислювальних ресурсів. У роботі [11] автори провели порівняльний аналіз ефективності методу стиснення зображень на основі використання класичних типів даних. Тому, у подальшому постає завдання, щодо визначення ефективності, продемонстрованого, методу арифметики

довгих чисел шляхом його реалізації та порівняння отриманих даних з методом побудованим на стандартних типах даних.

**Література**.

1. Левченко, А., Войтенков, Р. Граничні точності обчислень в інформаційних системах з представленням чисел із плаваючою комою / Збірник наукових праць Військова академія (м. Одеса). Одеса. 2014. 2(2). С. 157-160 URL:http://vaodessa.org.ua/images/zbirnyk_2/22.PDF

2. Шаріпова, І., Трутнєв С., Левченко А., Головко О. Витоки помилок прогнозування ситуацій в комп'ютерних системах імітаційного моделювання / Збірник наукових праць Військової академії (м. Одеса) 2020. № 2(14). С. 41 - 50 URL: https://doi.org/10.37129/2313-7509.2020.14.2.41-50

3. Appel A., Kellison A. VCFloat2: Floating-Point Error Analysis in Coq / CPP 2024: Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2023. pp. 14-29. URL: https://doi.org/10.1145/3636501.3636953

4. Mahnken R. New low order Runge–Kutta schemes for asymptotically exact global error estimation of embedded methods without order reduction / Computer Methods in Applied Mechanics and Engineering 2022. № 401(B). URL: https://doi.org/10.1016/j.cma.2022.115553

5. Chen J., Gao J., Chen Y., Oloulade B., Lyu T., Li Z. Auto-GNAS: A Parallel Graph Neural Architecture Search Framework / IEEE Transactions on Parallel and Distributed Systems 2022. №33(11) pp. 3117-3128 URL: https://doi.org/10.1109/TPDS.2022.3151895

6. Abdelhamid R., Kuwazawa G., Yamaguchi Y. Quantitative study of floating-point precision on modern FPGAs / HEART '23: Proceedings of the 13th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies 2023. pp. 49 – 58. URL: https://doi.org/10.1145/3597031.3597042

7. Lipitakis A., Kousiouris G., Nikolaidou M., Bardaki C., Anagnostopoulos D. Empirical investigation of factors influencing function as a service performance in different cloud/edge system setups / Simulation Modelling Practice and Theory №128, 2023. URL: https://doi.org/10.1016/j.simpat.2023.102808

8. IEEE 754 - Standard for Floating-Point Arithmetic 2019. URL: https://doi.org/10.1109/IEEESTD.2019.8766229

9. Левченко А. Алгоритми базових арифметичних операцій для двійкових чисел представлених як масиви / Матеріали 8-мої Міжнародної науково-технічної конференції «Інформаційні системи та технології ICT-2019», С. 67-71 URL: https://ist-conf-nure.com.ua/ist_2019.pdf

10. Knuth, D. Seminumerical Algorithms / The Art of Computer Programming, №2 1998. URL: https://www-cs-faculty.stanford.edu/~knuth/taocp.html.

11. Levchenko, A., Sharipova, I. Direct and inverse image conversion for compressing images by a drone computer / International scientific integration 2020 №04(01) pp. 74-78. URL: https://doi.org/10.30888/2709-2267.2020-4

**References.**

1. Levchenko A. and Voytenkov R. (2014) "Hranychni tochnosti obchyslen v informatsiynykh systemakh z predstavlennyam chysel iz plavayuchoyu komoyu" [Limiting accuracy of the calculations in information system with numeration with sailling comma] Collection of scientific works of the Odesa Military Academy, No.2(2), pp. 157-160 URL:http://vaodessa.org.ua/images/zbirnyk_2/22.PDF

2. Sharipova, I., Trutniev, S., Levchenko, A. and Holovko, O. (2020) "Vytoky pomylok prohnozuvannya sytuatsiy v kompyuternykh systemakh imitatsiynoho modelyuvannya" [Sources of situation forecasting errors in computer simulation systems], Collection of scientific works of the Odesa Military Academy, No.2(14), pp. 41- 50 URL: https://doi.org/10.37129/2313-7509.2020.14.2.41-50

3. Appel A. and Kellison A. (2023) "VCFloat2: Floating-Point Error Analysis in Coq", CPP 2024: Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 14-29. URL: https://doi.org/10.1145/3636501.3636953

4. Mahnken R. (2022) "New low order Runge–Kutta schemes for asymptotically exact global error estimation of embedded methods without order reduction", Computer Methods in Applied Mechanics and Engineering, No.401(B) URL: https://doi.org/10.1016/j.cma.2022.115553

5. Chen J., Gao J., Chen Y., Oloulade B., Lyu T. and Li Z. (2022) "Auto-GNAS: A Parallel Graph Neural Architecture Search Framework", IEEE Transactions on Parallel and Distributed Systems, No.33(11) pp. 3117-3128 URL: https://doi.org/10.1109/TPDS.2022.3151895

6. Abdelhamid R., Kuwazawa G. and (2023) "Yamaguchi Y. Quantitative study of floating-point precision on modern FPGAs", HEART '23: Proceedings of the 13th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologie. pp. 49 – 58 URL: https://doi.org/10.1145/3597031.3597042

7. Lipitakis A., Kousiouris G., Nikolaidou M., Bardaki C. and Anagnostopoulos D. (2023) "Empirical investigation of factors influencing function as a service performance in different cloud/edge system setups", Simulation Modelling Practice and Theory No128,. URL: https://doi.org/10.1016/j.simpat.2023.102808

8. IEEE 754 - Standard for Floating-Point Arithmetic 2019. URL: https://doi.org/10.1109/IEEESTD.2019.8766229

9. Levchenko, (2019) "Alhorytmy bazovykh aryfmetychnykh operatsiy dlya dviykovykh chysel predstavlenykh yak masyvy "[Basic arithmetic operations algorithms for binary numbers represented as arrays], Materials of the 8th International Scientific and Technical Conference "Information Systems and Technologies IST-2019" pp. 67-71 URL: https://ist-conf-nure.com.ua/ist_2019.pdf

10. Knuth, D. (1998)"Seminumerical Algorithms "The Art of Computer Programming, No 2. URL: https://www-cs-faculty.stanford.edu/~knuth/taocp.html.

11. Levchenko, A. and Sharipova, I. (2020) "Direct and inverse image conversion for compressing images by a drone computer", International scientific integration 2020 No04(01) pp. 74-78. URL: https://doi.org/10.30888/2709-2267.2020-4

***Abstract*** *The article presents research on ways to improve computational accuracy in simulation modeling systems through the implementation of arbitrary-precision arithmetic. The limitations of traditional floating-point data types (IEEE 754) and their impact on the reliability of simulation results for complex dynamic systems are analyzed. An efficient structure for storing high-precision numbers has been developed based on an array with a base-10^9 number system, providing an optimal balance between memory usage and operational simplicity. Algorithms for basic mathematical operations (addition, subtraction, multiplication, division) for working with arbitrary-precision numbers have been proposed and implemented. A comparative analysis of computational accuracy between classical data types and the developed arbitrary-precision arithmetic system has been conducted. The practical implementation of the proposed methods in Python is presented. The research results demonstrate a significant improvement in computational accuracy when using arbitrary-precision arithmetic compared to traditional data types, which is*

*particularly important for complex scientific and engineering calculations where the accumulation of rounding errors can lead to significant deviations in final results.*

***Keywords:*** *simulation modeling, arbitrary-precision arithmetic, high-precision computing, errors, binary arithmetic, floating-point data representation, data structures, forecasting, basic arithmetic operations.*