Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б11-мм

Устранение ошибок с плавающей запятой при помощи длинной арифметики

Шангареев Рустам Ильшатович

Отчёт по учебной практике
в форме «Эксперимент»

Научный руководитель:
профессор кафедры информатики, д.ф.-м.н., Т. М. Косовская

Санкт-Петербург
2022

Оглавление

Введение

При решении различных задач часто встаёт необходимость в арифметических действиях с вещественными числами. Однако в процессе решения возникают погрешности, связанные с представлением вещественных чисел в памяти с плавающей запятой, которое не позволяет сохранять заданную точность, а также накапливает ошибку с ростом числа арифметических операций.

Особый интерес представляют сферы, в которых программное обеспечение должно быть отказоустойчивым и безопасным: медицина, биология, аэрокосмическая промышленность, энергетика. В этих областях особенно важна точность, так как от них зависят жизнь и здоровье людей. Исторически известно, что цена ошибки высока в экономике и финансах: любая погрешность может привести к серьезным экономическим последствиям [10]. Порой такие инциденты приводят и к смерти людей: 25 февраля 1991 года округление в программном обеспечении ракетной батареи помешало перехватить приближающуюся ракету, что привело к гибели 28 солдат. [9]

В данный момент не существует наиболее универсального метода устранения ошибки плавающей запятой, самыми известными способами частичного решения этой проблемы являются численный анализ ошибок(в том числе анализ ошибок с помощью арифметики Монте-Карло) [1, 3, 14], использование члена-ошибки для вычисления ошибки после выполнения арифметических операций [12], интервальная арифметика [5], Unums [2], а также длинная арифметика.

В рамках учебной практики планируется провести эксперименты по решению некоторых прикладных задач в вещественных числах и числах произвольной длины, сравнив эффективность этих алгоритмов, а также проанализировать целесообразность использования длинной арифметики в этих случаях.

1. Постановка задачи

Целью данной работы является сравнение эффективности алгоритмов решения практических проблем в вещественных числах с плавающей запятой, а также в «длинных» числах. Для достижения цели были поставлены следующие задачи.

1. Провести общий обзор проблемы ошибки с плавающей запятой, а также существующих ее решений.

2. Спроектировать математическую модель алгоритмов для решения задач, отобранных для экспериментов.

3. Реализовать алгоритмы по решению выбранных задач при помощи чисел с плавающей запятой и с произвольной длиной.

4. Определить метрики эффективности алгоритмов решения с использованием указанных типов данных.

5. Провести сравнение по определенным метрикам реализованных алгоритмов, оценив эффективность длинной арифметики для устранения ошибки плавающей запятой.

2. Обзор

Действительные числа не могут быть точно представлены в условиях конечной памяти, поэтому для их хранения используют некоторую аппроксимацию, что вызывает ошибки. Арифметика с плавающей запятой — на данный момент наиболее распространенный способ представления вещественных чисел в компьютерах . Сегодня, когда практически все сферы жизни зависят от компьютеров, ошибки с плавающей запятой часто становятся проблемой. В следующем разделе будет проведен обзор различных средств устранения ошибок рассматриваемой арифметики действительных чисел и объяснён выбор длинной арифметики для тестирования эффективности.

2.1. Обзор существующих решений проблемы плавающей запятой

Для поиска существующих способов устранения ошибок плавающей запятой использовался сервис "Google Scholar", а точнее литература, посвященная рассматриваемому вопросу. Поиск литературы производился по запросу "Floating-point error". По запросу были изучены первые 2 страницы, выданных поисковой системой (8 источников). Все 16 источников были рассмотрены на предмет соответствия данной предметной области в рамках обзора литературы. В результате были отобраны только 10 источников, которые подходили под наши критерии [16]. В них были найдены следующие решения поставленной проблемы.

1. Численный анализ ошибок(в том числе арифметика Монте-Карло и интервальная арифметика) [1, 3, 14], заключающийся в анализе и минимизации ошибки округления с плавающей запятой. Данный метод лишь позволяет при получении результата оценить погрешность в сравнении с истинным результатом.

2. Повышение точности чисел с плавающей запятой, то есть использование большей памяти для хранения одного числа [4]. Это реше-

ние делает последствия ошибки менее вероятными и менее важными, однако истинная точность результатов неизвестна.

3. Длинная арифметика представляет собой использование чисел переменной длины, ограниченной только доступной памятью [8]. При использовании этого метода для рациональных вещественных чисел точность не теряется, однако снижается скорость работы алгоритма.

4. Unums Джона Густафсона является некоторым расширением арифметики переменной длины, реализуя числа с плавающей запятой с переменной длиной и точностью, которая контролируется с помощью дополнительных битов [2]. Данный метод так же, как и интервальная арифметика, подвержен получению слишком большого интервала, содержащего истинный результат, в конце вычислений [7].

5. Ограниченная плавающая запятая Алана Джергенсена использует структуру данных из обычного числа с плавающей запятой и информацию об ошибке [6]. Это решение по сути является производным от интервальной арифметики и Unums, поэтому подвержена тем же самым проблемам.

Из рассмотренных методов лишь одна длинная арифметика позволяет сохранить точность, остальные способны только сообщить о существовании ошибки, либо уменьшить ее относительно обычных чисел с плавающей запятой. Таким образом, использование чисел произвольной длины является наиболее подходящим методом, если в первую очередь для задачи важна точность, а не производительность. Именно поэтому сделан выбор в использовании длинной арифметики для сравнения ее с арифметикой с плавающей запятой.

3. Проектирование алгоритмов для тестирования

В данном разделе представлено математическое описание реализации алгоритмов, которые будут использоваться для экспериментов.

3.1. Алгоритм решения задачи Коши для дифференциального уравнения

Задача Коши для дифференциального уравнения представляет собой систему вида

$$\begin{cases} y' = F(x, y) & (1) \\ y(x_0) = y_0 & (2) \end{cases},$$

где y' непрерывна на некотором отрезке $[\alpha', \beta']$. Решение задачи Коши есть дифференцируемая на (α', β') непрерывная на $[\alpha', \beta']$ функция $y = f(x)$, являющаяся решением данной системы уравнений. Зачастую $f(x)$ не выражается через элементарные функции, поэтому возникает потребность в численном нахождении $f(x)$ на некотором отрезке $[a, b] \subset [\alpha', \beta']$, то есть в поиске набора $\{f(x_k)\}_{k=1}^{n}$ для некоторой конечной последовательности $\{x_k\}_{k=1}^{n} \subset [a, b]$.

3.1.1. Численное решение в вещественных числах

Возьмем разбиение $\{p_k\}_{k=-\alpha}^{\beta} \subset [a, b]$ с шагом $h = \frac{1}{10^n}$, начальным условием $p_0 = x_0$ и границами индексов $\alpha, \beta \in \mathbb{N} \cup \{0\}$. Пусть g — функция, аппроксимирующая точное решение $f(x)$ на отрезке $[a, b]$, тогда из уравнения (2) рассматриваемой системы получим, что $g(x_0) = y_0$. Обозначим $q_k = g(p_k)\ \forall k \in [-\alpha, \beta]$, значит $p_0 = x_0$, $q_0 = y_0$. Так как $f(x)$ дифференцируема на $[a, b]$, у g существуют конечные производные слева и справа в любой точке $p_i \in \{p_k\}$:

$$\lim_{\Delta \to +0} \frac{g(p_i + \Delta) - g(p_i)}{\Delta} = F(p_i, q_i), \ \lim_{\Delta \to -0} \frac{g(p_i + \Delta) - g(p_i)}{|\Delta|} = F(p_i, q_i).$$

Отсюда при достаточно маленьком h(его мелкость зависит от выбранного n)

$$\frac{g(p_i + h) - g(p_i)}{h} = F(p_i, q_i),$$
$$\frac{g(p_i) - g(p_i - h)}{h} = F(p_i, q_i),$$

то есть

$$\frac{q_{k+1} - q_k}{h} = F(p_k, q_k),$$
$$\frac{q_k - q_{k-1}}{h} = F(p_k, q_k).$$

Таким образом, мы получаем рекуррентные соотношения для q_{k-1} и q_{k+1}:

$$q_{k-1} = q_k - F(p_k, q_k) \cdot h,\ q_{k+1} = q_k + F(p_k, q_k) \cdot h \quad (3)$$

3.1.2. Численное решение в числах произвольной длины

Зададимся требуемой точностью 10^{-N} хранения значений $F(x, y)$.

• Значения $F(p_k, q_k)$ представим с заданной точностью 10^{-N} в виде $f_k \cdot 10^{-N}$, где $f_k \in \mathbb{Z}$. Отсюда

$$f_k = \left\lfloor F(p_k, q_k) \cdot 10^N \right\rfloor.$$

• Значения функции в узлах разбиения t_k будем хранить с контролируемой точностью, зависящей от n и N, в виде $\varphi_k \cdot 10^{-N-n}$, где $\varphi_k \in \mathbb{Z}$. Отсюда

$$\varphi_k = \left\lfloor t_k \cdot 10^{N+n} \right\rfloor.$$

Используя (3), получаем рекуррентные соотношения для φ_k и формулу нахождения f_k:

$$\varphi_{k-1} = \varphi_k - f_k,\ \varphi_{k+1} = \varphi_k + f_k,$$

$$f_k = F(p_k,\ \frac{\varphi_k}{10^{N+n}}).$$

3.2. Алгоритм решения системы линейных уравнений методом Гаусса

Одна из классических задач линейной алгебры по решению СЛАУ имеет вид системы уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases},$$

для которой требуется найти все такие совокупности $(c_1, c_2, \ldots, c_n)$, что их соответственная подстановка вместо $x_1, x_2, \ldots, x_n$ в систему обращает каждое уравнение в тождество. Метод Гаусса является одним из самых известных и эффективных методов решения СЛАУ ограниченного размера: приблизительно при $\max(m, n) < 100$ [13], поэтому мы используем именно его для нашего эксперимента. Примем также, что в нашей системе уравнений нет нулевых столбцов, то есть $\forall i \in [1, n] \cap \mathbb{Z}$ $a_{1i}^2 + a_{2i}^2 + \ldots + a_{mi}^2 \neq 0$, потому что если такие i существуют, то данные нулевые столбцы можно исключить, а любое найденное решение будет являться решением при любом x_i.

3.2.1. Реализация алгоритма с числами с плавающей запятой

В начале метода Гаусса мы должны привести матрицу (a_{ij}) к треугольному виду, то есть к такой матрице, в которой все элементы, стоящие ниже главной диагонали, равны 0. Для этого с вещественными числами выполним $(n-1)$ шаг такого вида, что на k-ом шаге мы будем производить следующие действия:

1. найдем строку $i \in [k, n]$ с наибольшим по модулю коэффициентом при x_k и поменяем местами коэффициенты i-ой и k-ой строк(в том числе b_i и b_k), не меняя обозначения для этих коэффициентов, так как это не имеет значения в системе.

2. для каждого $j \in [k+1, n]$ заменим j-ую строку на разность k-ой и умноженной на $\frac{a_{kk}}{a_{jk}}$ j-ой строк(в том числе b_j).

Таким образом, мы получим систему с новыми коэффициентами вида

$$\begin{cases} a'_{11}x_1 + a'_{12}x_2 + a'_{13}x_3 + \cdots + a'_{1(n-1)}x_{n-1} + a'_{1n}x_n = b'_1 \\ 0 \cdot x_1 + a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2(n-1)}x_{n-1} + a'_{2n}x_n = b'_2 \\ 0 \cdot x_1 + 0 \cdot x_2 + a'_{33}x_3 + \cdots + a'_{3(n-1)}x_{n-1} + a'_{3n}x_n = b'_3 \\ \cdots \\ 0 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \cdots + 0 \cdot x_{n-1} + a'_{mn}x_n = b'_m \end{cases}.$$

Для решения получившейся задачи достаточно лишь пойти от последнего уравнения, подставляя уже найденные переменные и вычисляя новые(из последнего уравнения явно находится x_n, далее x_n подставляется в остальные уравнения, все константы переносятся на правую сторону уравнения, и задача рекурсивно решается для набора $x_1, x_2, \ldots, x_{n-1}$)

3.2.2. Реализация алгоритма с числами произвольной длины

Используя «длинные» целые числа, для начала зададим требуемую точность для аппроксимации коэффициентов системы уравнений, пусть ошибка будет не больше, чем 10^{-N}. Тогда $\forall i \in [1, m] \cap \mathbb{Z}$, $j \in [1, n] \cap \mathbb{Z}$ представим a_{ij} как $\alpha_{ij} \cdot 10^{-N}$, где $\alpha_{ij} \in \mathbb{Z}$, а b_i как $\beta_i \cdot 10^{-N}$, где $\beta_i \in \mathbb{Z}$. Таким образом, после умножения каждого уравнения системы на 10^N получаем новую задачу по решению СЛАУ с целыми коэффициентами:

$$\begin{cases} \alpha_{11}x_1 + \alpha_{12}x_2 + \cdots + \alpha_{1n}x_n = \beta_1 \\ \alpha_{21}x_1 + \alpha_{22}x_2 + \cdots + \alpha_{2n}x_n = \beta_2 \\ \cdots \\ \alpha_{m1}x_1 + \alpha_{m2}x_2 + \cdots + \alpha_{mn}x_n = \beta_m \end{cases}.$$

Выполним $(n-1)$ шаг метода Гаусса немного другого вида в отличие от алгоритма с плавающей запятой так, что после k-ой итерации будем для каждого $j \in [k+1, n]$ заменим j-ую строку на разность умноженной

на α_{jk} k-ой строки и умноженной на α_{kk} j-ой строки(в том числе β_j). В итоге после k-ого шага матрица коэффициентов при переменных будет выглядеть так:

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1k} & \dots & \alpha_{1n} \\ 0 & \gamma_{22}^{(1)} & \dots & \gamma_{2k}^{(1)} & \dots & \gamma_{2n}^{(1)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \gamma_{kk}^{(k-1)} & \dots & \gamma_{kn}^{(k-1)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & & \\ 0 & 0 & 0 & 0 & & \Gamma^{(k)} \end{pmatrix},$$

где $\Gamma^{(k)}$ — матрица с элементами $\gamma_{ij}^{(k)}$ для $i \in [k+1, m] \cap \mathbb{Z}$, $j \in [k+1, n] \cap \mathbb{Z}$, вычисляющимися с помощью формулы $\gamma_{ij}^{(k)} = \gamma_{ij}^{(k-1)}\gamma_{kk}^{(k-1)} - \gamma_{kj}^{(k-1)}\gamma_{jk}^{(k-1)}$ при начальном условии $\gamma_{ij}^{(1)} = \alpha_{ij}\alpha_{11} - \alpha_{1j}\alpha_{j1}$.

По теореме Сильвестра [11] для всех $k \geq 2$ при $i \in [k+1, m] \cap \mathbb{Z}$, $j \in [k+1, m] \cap \mathbb{Z}$ элемент $\gamma_{ij}^{(k)}$ делится без остатка на $\gamma_{(k-1)(k-1)}^{(k-2)}$(где $\gamma_{11}^{(0)} = \alpha_{11}$). Что даёт нам окончательную формулу заполнения коэффициентов:

$$\begin{cases} \gamma_{ij}^{(1)} = \alpha_{ij}\alpha_{11} - \alpha_{1j}\alpha_{j1} \\ \gamma_{ij}^{(k)} = \frac{\gamma_{ij}^{(k-1)}\gamma_{kk}^{(k-1)} - \gamma_{kj}^{(k-1)}\gamma_{jk}^{(k-1)}}{\gamma_{(k-1)(k-1)}^{(k-2)}}, \text{ если } k > 1 \end{cases}.$$

Данный факт позволяет нам полиномиально ограничить рост длины записи коэффициентов системы уравнений и повысить производительность алгоритма без потери точности. Полученная СЛАУ с треугольной матрицей коэффициентов $(\gamma_{ij}^{(k)})$ решается обратным ходом от последнего уравнения так же, как в алгоритме с плавающей запятой.

4. Ход работы

4.1. Условия эксперимента

В качестве тестового стенда для экспериментов был выбран ноутбук Lenovo IdeaPad Flex 5, имеющий следующие основные характеристики:

- Оперативная память: 8 Гб с двумя каналами и тактовой частотой 1600 МГц;
- Операционная система: Ubuntu 20.04 на ядре Linux 5.15;
- Процессор: AMD Ryzen 3 4300U(4 ядра, 4 потока).

В качестве языка программирования для кодирования и исполнения алгоритмов был выбран Python 3.8[GCC 9.4.0]. Выбор обусловлен тем, что стандартная библиотека Python поддерживает длинную арифметику, облегчая реализацию алгоритмов с числами произвольной длины.

4.2. Исследовательские вопросы (Research questions)

Данной работы мы бы хотели добиться следующих целей.

1. Проверить, насколько эффективна по выбранным метрикам длинная арифметика как способ устранения ошибок плавающей запятой.
2. Оценить целесообразность использования длинной арифметики в рассмотренных случаях.

4.3. Метрики

Выберем следующие метрики сравнения алгоритмов для решения выбранных задач в различных типах данных.

- Производительность(время работы алгоритма)
- Точность алгоритма(определяется для каждой задачи отдельно)

4.4. Результаты

Данный раздел посвящён результатам экспериментов, их обсуждению и подведению итогов.

4.4.1. Решение системы линейных уравнений методом Гаусса

Имплементировав алгоритмы по решению системы линейных уравнений методом Гаусса с числами с плавающей запятой и с числами произвольной длины, проведем их сравнение по определенным ранее метрикам.

Сравнение будем проводить по результатам 50 запусков каждого из алгоритмов для решения системы линейных уравнений, матрицы коэффициентов которых имеют три разных размера: 25×25, 50×50, 75×75. В качестве коэффициентов используем конечные десятичные дроби, так как любое вещественное число при вводе в компьютер аппроксимируется конечной десятичной дробью, сгенерируем коэффициенты случайно из отрезка $[-10^{15}, 10^{15}]$ для каждого теста.

Размер матрицы	Float-point	Bignum
25×25	0.045 ± 0.003	0.129 ± 0.002
50×50	0.057 ± 0.005	1.652 ± 0.011
75×75	0.078 ± 0.006	9.855 ± 0.043

Таблица 1: Производительности алгоритма для плавающей запятой(Float-point в таблице) и алгоритма для чисел произвольной длины(Bignum в таблице) в секундах, при различных размерах матрицы коэффициентов

После сравнения производительности, проверим точность решений, являющихся результатом работы алгоритмов. В качестве метрики точности решения выберем следующую метрику: сумма абсолютных разностей левых и правых частей уравнений системы при подстановке решения, то есть если алгоритм для матрицы коэффициентов (a_{ij}) и набора свободных коэффициентов (b_i), $i \in [1, m] \cap \mathbb{Z}$, $j \in [1, n] \cap \mathbb{Z}$, нашел реше-

ние $(c_1, c_2, \ldots, c_n)$, то точностью решения назовем(меньше — лучше)

$$\sum_{i=1}^{m} \left| \sum_{j=1}^{n} (c_j a_{ij}) - b_i \right|.$$

Прежде всего отметим, что алгоритм, использующий числа произвольной длины, находит абсолютно точное решение, поэтому имеет смысл тестировать только алгоритм с плавающей запятой

Размер матрицы	Float-point
25×25	571±52
50×50	5760±328
75×75	23 585±1438

Таблица 2: Точность алгоритма с плавающей запятой(Float-point в таблице) по выбранной метрике при различных размерах матрицы коэффициентов

Полученные результаты показывают, что если требуется точное решение системы линейных уравнений и не так важна производительность, алгоритм с числами произвольной решает проблему ошибки с плавающей запятой.

4.4.2. Решение задачи Коши для дифференциального уравнения

Для тестирования алгоритмов решения задачи Коши для дифференциального уравнения была выбрана следующая система, которая хорошо подходит для сравнения, так как имеет разрыв II рода.

$$\begin{cases} y' = F(x, y) = \frac{2y+1}{\cos(2x)+1} \\ y(0) = 1 \end{cases}$$

Далее определим метрику точности. По результатам экспериментов алгоритм, использующий длинной арифметику, превзошел по точности алгоритм с плавающей запятой. Однако чтобы можно было оценить

такой прирост, мы введем следующую метрику сравнения алгоритмов: разность между средней ошибкой численного решения с числами произвольной точности и численным решением с плавающей запятой, то есть, если $f(x)$ — точное решение системы, $\{t_i\}_{i=1}^n$ — разбиение отрезка аппроксимации, $g_1(x)$ — функция, полученная алгоритмом с плавающей запятой, $g_2(x)$ — функция, полученная алгоритмом с числами произвольной длины, то метрика сравнения будет следующая.

$$d(g_1, g_2) = \frac{\sum_{i=1}^n |g_2(t_i) - f(t_i)|}{n} - \frac{\sum_{i=1}^n |g_1(t_i) - f(t_i)|}{n}$$

Рассмотрим работу наших алгоритмов на отрезках $[-1.4, 1.4]$, $[-1.5, 1.5]$, $[-1.55, 1.55]$. Возьмем разбиение с шагом $h = \frac{1}{10^5}$. Точность аппроксимации решения с длинной арифметикой $N = 20$. Тогда тестирование по определенным метрикам даст следующие результаты.

Отрезок	Float-point	Bignum
[-1.4, 1.4]	3.224 ± 0.019	3.309 ± 0.009
[-1.5, 1.5]	3.365 ± 0.013	3.485 ± 0.012
[-1.55, 1.55]	3.562 ± 0.018	3.708 ± 0.018

Таблица 3: Производительности алгоритма для плавающей запятой(Float-point в таблице) и алгоритма для чисел произвольной длины(Bignum в таблице) в секундах при работе на различных отрезках(чем больше отрезок, тем ближе разрыв II рода)

Размер матрицы	Разность точностей алгоритмов
[-1.4, 1.4]	$3.1 \cdot 10^{-8}$
[-1.5, 1.5]	$1.2 \cdot 10^{-5}$
[-1.55, 1.55]	$1.7 \cdot 10^{+8}$

Таблица 4: Разность точностей алгоритмов по выбранной метрике при работе на различных отрезках(чем больше отрезок, тем ближе разрыв II рода)

Отметим, что на отрезке [-1.55, 1.55] средняя ошибка у каждого ал-

горитма настолько высока, что разность $1.7 \cdot 10^8$ не является ощутимым приростом в точности.

Данный пример показывает, что длинная арифметика не во всех случаях демонстрирует ощутимый прирост точности. В этом случае метод решения численной аппроксимации функции с разрывом II рода приводит к дополнительным ошибок, хоть и ошибки плавающей запятой устранены.

4.4.3. RQ1

Длинная арифметика вполне эффективна для устранения ошибок плавающей запятой, что подтверждают эксперименты, если метод решения сам не порождает ошибки точности.

4.4.4. RQ2

Для решения СЛАУ методом Гаусса длинная арифметика целесообразна, если производительность не играет большую роль. Для решения задачи Коши числа произвольной длины можно использовать, если целью является наиболее точная возможная аппроксимация.

5. Реализация

Реализация алгоритмов, а также экспериментов доступна в репозитории GitHub [15].

Заключение

В течение осеннего семестра в рамках учебной практики были выполнены следующие задачи.

1. Проведен общий обзор проблемы с плавающей запятой, а также существующих их решений и их недостатков.

2. Спроектирована математическая модель алгоритмов, использующих различные типа данных, для решения выбранных задач.

3. Реализованы алгоритмы по спроектированным математическим моделям.

4. Определены метрики эффективности для сравнения алгоритмов решения.

5. Проведены эксперименты по сравнению алгоритмов, решающих поставленные задачи в различных типах данных, сделаны выводы по эффективности замены чисел с плавающей запятой длинной арифметикой.

Список литературы

[1] Dirk P. Kroese T. Brereton T. Taimre Z. Botev. Why the Monte Carlo method is so important today. — URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.1314.

[2] Gustafson John L. The End of Error. Unum Computing. — URL: https://www.taylorfrancis.com/books/mono/10.1201/9781315161532/end-error-john-gustafson.

[3] Higham Nicholas J. Accuracy and Stability of Numerical Algorithms: Second Edition. — URL: https://books.google.ru/books?id=epilvM5MMxwC&q=accuracy+and+stability+of+numerical+algorithms+higham&redir_esc=y.

[4] IBM. Double-Precision Floating Point. — URL: https://ieeexplore.ieee.org/abstract/document/930115.

[5] IEEE. 1788-2015 - IEEE Standard for Interval Arithmetic. — URL: https://ieeexplore.ieee.org/document/7140721.

[6] Jorgensen Alan A. Exact Floating Point. — URL: https://link.springer.com/chapter/10.1007/978-3-030-70873-3_26.

[7] Kahan W. A Critique of John L. Gustafson's THE END of ERROR — Unum Computation. — URL: https://people.eecs.berkeley.edu/~wkahan/UnumSORN.pdf.

[8] Kaveh R. Ghazi Vincent Lefèvre Philippe Théveny Paul Zimmermann. Why and how to use arbitrary precision. — URL: https://homepages.loria.fr/PZimmermann/papers/cise.pdf.

[9] Office U.S. Government Accountability. Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia. — URL: https://www.gao.gov/products/imtec-92-26.

[10] Rampell Catherine. A History of Oopsies in Economic Studies. — URL: https://archive.nytimes.com/economix.blogs.nytimes.com/2013/04/17/a-history-of-oopsies-in-economic-studies/.

[11] Schrijver Alexander. Theory of Linear and Integer Programming. — URL: https://www.wiley.com/en-us/Theory+of+Linear+and+Integer+Programming-p-9780471982326.

[12] Shewchuk Jonathan Richard. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. — URL: https://people.eecs.berkeley.edu/~jrs/papers/robustr.pdf.

[13] Skiena Steven S. The algorithm design manual. — URL: https://archive.org/details/algorithmdesignm00skie_772.

[14] Trefethen Lloyd N. The definition of numerical analysis. — URL: https://ecommons.cornell.edu/handle/1813/6163.

[15] Репозиторий с реализацией. — URL: https://github.com/letit6E/bignum-experiments.

[16] Статьи, отобранные в результате обзора. — URL: https://docs.google.com/document/d/1kXiZdG0W6DaV4XfmznuSuf21kNlO64jzYPbKJP4ReEY/edit?usp=sharing.