Стандартизация и тестирование реализаций математических функций, работающих с числами с плавающей точкой

В. В. Кулямин

Институт системного программирования РАН (ИСП РАН),
109004, Б. Коммунистическая, 25, Москва, Россия
E-mail: kuliamin@ispras.ru

Аннотация

В статье рассматриваются проблемы выработки требований и создания тестов для реализаций математических функций, работающих с числами с плавающей точкой в форматах стандарта IEEE 754. Излагается основанный на обобщении идей этого стандарта метод определения требований для таких функций, который может быть использован для их стандартизации, расширяющей IEEE 754. Также представлен метод разработки тестов для проверки выполнения сформулированных требований. Описанные методы опираются на специфические свойства представления чисел с плавающей точкой и особенности поведения самих рассматриваемых функций.

Содержание

1. Введение

Одной из областей человеческой деятельности, опирающейся на использование сложного программного обеспечения (ПО), является математическое моделирование сложных явлений. Это, например, моделирование развития вселенной в целом, галактик и звезд, физических процессов в экстремальных условиях, биохимических, климатических и социальных процессов. Во многих случаях компьютерное моделирование дает важную информацию о таких явлениях, но очень нелегко получить независимую оценку правильности получаемых результатов, что позволило бы проверить корректность самого используемого ПО. Это порождает серьезные проблемы при разработке надежных систем математического моделирования.

Тем не менее, повысить надежность и правильность работы таких систем можно за счет формальной проверки корректности библиотечных компонентов, на которые оно во многом опирается, в частности, реализаций математических функций. Уверенность в надежности фундамента, на котором построены эти системы, даст возможность разрабатывать их более качественно и с меньшими усилиями, сосредоточившись на поиске и исправлении ошибок в других компонентах.

Огромное количество практически значимых проблем связано с непрерывными системами, чьи пространства состояний являются действительными или комплексными многообразиями. Точки такого многообразия — состояния системы — за счет введения системы координат можно представить в виде наборов действительных чисел, а возможные пути развития системы описываются кривыми на этом многообразии. Использование компьютеров для моделирования подобных систем практически всегда связано с тем или иным представлением непрерывных пространств их состояний в дискретном виде, что создает дополнительные проблемы при обеспечении корректности работы используемого ПО.

Наиболее часто используется представление действительных чисел в виде двоичных чисел с плавающей точкой, описанных в стандарте IEEE 754 [1] (он же — IEC 60559 [2]). Этот стандарт был введен в середине 80-х годов, до этого не было общеупотребительного переносимого машинного представления для действительных чисел. Целью его введения было создание переносимых библиотек вычислительных алгоритмов и обеспечение возможности использовать различные платформы для одних и тех же расчетов с гарантией повторяемости получаемых результатов. Хотя для реализации этого стандарта многим производителям аппаратного обеспечения и ПО пришлось существенно переделать собственные системы и реализации библиотек численных методов, на данный момент он поддерживается на подавляющем большинстве платформ. Постепенно забывается «зоопарк» архитектур начала 80-х и существовавшая тогда повсеместно жесткая привязка хороших реализаций вычислительных алгоритмов и использующего их ПО к определенной платформе.

Тем не менее, провозглашенная цель не была достигнута в полной мере. Стандарт IEEE 754 регламентирует только машинное представление чисел с плавающей точкой и базовые операции над ними — сложение, вычитание, умножение, деление, сравнение и приведение типов. Из математических функций этим стандартом описывается только извлечение квадратного корня. Это дает возможность реализовывать библиотеки, формально полностью соответствующие стандарту, но дающие совершенно разные результаты при вычислениях (см. примеры ниже), и приводит к непереносимости и ненадежной работе приложений, использующих функции, не затронутые в IEEE 754. В результате пользователи, которым необходимы точные вычисления, пользуются другими, нестандартными, библиотеками. Налицо потребность в стандартизации реализаций большого количества математических функций, часто используемых в приложениях. Стандартизация же всегда связана с задачей проверки корректности реализаций на соответствие сформулированным в стандарте требованиям.

В данной работе представлен возможный подход к созданию стандарта для реализаций математических функций, работающих с числами с плавающей точкой в форматах IEEE 754. Поскольку стандарт требует подкрепления в виде тестов на соответствие его требованиям, предлагается также метод построения подобных тестов. Описываемый метод использует формальные требования к поведению математических функций и особенности структуры чисел с плавающей точкой.

Большинство идей обсуждаемого подхода к стандартизации реализаций математических функций заимствованы из различных источников — группы стандартов ISO/IEC 10967 [3-5] и работ исследователей из INRIA (Франция), выполненных в рамках проекта Arenaire [6-8]. Но, насколько известно автору, представленная методика определения требований к реализациям математических функций в целостном виде нигде в доступной литературе не описана. Предложенный метод построения тестов в целом является новым, хотя отдельные его элементы также можно найти в литературе или встретить их реализацию в практических проектах.

2. Проблемы вычисления математических функций

Рассмотрим сначала проблемы корректного вычисления математических функций на базе *чисел с плавающей точкой.*

Эти проблемы связаны, прежде всего, с дискретностью этого представления действительных чисел, что дает возможность эффективно работать с ними на компьютере, но приводит к непредставимости точных результатов большинства вычислений. Сам формат чисел с плавающей точкой и правила действий над ними определены в стандартах IEEE 754 [1] (IEC 60559 [2]) и IEEE 854 [9].

IEEE 754 определяет представление двоичных чисел с плавающей точкой. IEEE 854 обобщает его, определяя также десятичные числа с плавающей точкой. Поскольку на практике чаще используется двоичное представление чисел, мы будем рассматривать далее только его, хотя все рассуждения в принципе переносятся и на десятичные числа.

2.1. Числа с плавающей точкой

Двоичное число с плавающей точкой имеет следующую структуру [1,9,10].

- Число представлено в виде набора из n бит, из которых первый бит является *знаковым битом числа*, следующие k бит представляют его *экспоненту* E, а оставшиеся *(n-k-1)* бит представляют его *мантиссу* M.
- Знаковый бит S, экспонента E и мантисса M числа x определяют его значение по следующим правилам.
  $x = (-1)^S \cdot 2^e \cdot m$, где
  - S — знаковый бит, равный 0 для положительных чисел, и 1 для отрицательных;
  - если $0 < E < 2^k - 1$, то $e = E - 2^{(k-1)} + 1$;
    иначе, если $E = 0$, $e = -2^{(k-1)} + 2$;
    число $b = (2^{(k-1)} - 1)$ называется *смещением экспоненты (exponent bias)*;
  - если $0 < E < 2^k - 1$, то $m = 1 + M/2^{n-k-1}$. Иначе говоря, m имеет двоичное представление $1.M$, т.е. целая часть m равна 1, а последовательность цифр дробной части совпадает с последовательностью бит M;
    если же $E = 0$, то $m = M/2^{n-k-1}$, или m имеет двоичное представление $0.M$. Числа с нулевой экспонентой называются *денормализованными*, а все остальные — *нормализованными*.
- Максимальное возможное значение экспоненты $E = 2^k - 1$ зарезервировано для представления *исключительных чисел*: положительной и отрицательной бесконечностей, $+\infty$ и $-\infty$, а также специального значения NaN (not-a-number, не число). NaN используется, если результат выполняемых действий нельзя корректно представить ни обычным числом, ни бесконечностью, как, например, результаты 0/0 или $(-\infty) + (+\infty)$.
  $+\infty$ имеет нулевой знаковый бит, максимальную экспоненту и нулевую мантиссу; $-\infty$ отличается только единичным знаковым битом.
  Любое число, имеющее максимальную экспоненту и ненулевую мантиссу, считается представлением NaN.
- Стандарты IEEE 754 и IEEE 854 определяют несколько возможных типов чисел с плавающей точкой, из которых чаще всего используются *числа однократной точности* (single precision), *числа двойной точности* (double precision) и *числа расширенной двойной точности* (double-extended precision).
  Для чисел однократной точности $n = 32$ и $k = 8$. Соответственно, для мантиссы

используется 23 бита и смещение экспоненты равно 127.

Для чисел двойной точности $n = 64$ и $k = 11$. Для мантиссы используется 52 бита и смещение экспоненты равно 1023.

Для чисел расширенной двойной точности определенные значения k и n не фиксируются в стандартах, вводятся лишь ограничения $128 \geq n \geq 80$ и $k \geq 15$. В процессорах 32-битной архитектуры Intel $n = 80$ и $k = 15$. При этом для мантиссы используется 64 бита и смещение экспоненты равно 16383.

Кроме этого, иногда используются числа *четырехкратной точности*, для которых $n = 128$ и $k = 15$. Соответственно, для мантиссы используется 112 бит и смещение экспоненты равно 16383. Они не специфицированы в стандарте, но имеют аналогичную структуру.

В качестве примера укажем представление числа -17_{10} в виде чисел однократной и двойной точности. Поскольку $-17_{10} = (-1)^1 \cdot 2^{131-127} \cdot 1.0001_2 = (-1)^1 \cdot 2^{1027-1023} \cdot 1.0001_2$, соответственно, оно представляется в виде числа однократной точности как
1 10000011 00010000000000000000000, а в виде числа двойной точности как
1 10000000011 0001000000000000000000000000000000000000000000000000.

Далее везде вместо точного битового представления чисел с плавающей точкой будет использоваться их запись в шестнадцатеричной системе.

Некоторые граничные значения таких чисел для разных форматов указаны в Таблице 1. При этом для расширенной двойной точности используются параметры 32-битной архитектуры Intel.

	Общий вид	Однократная точность	Двойная точность	Расширенная двойная точность	Четырехкратная точность
Самое маленькое денормализованное положительное число	$2^{-b-n+k+2}$	2^{-149}	2^{-1074}	2^{-16446}	2^{-16494}
Самое маленькое нормализованное положительное число	2^{-b+1}	2^{-126}	2^{-1022}	2^{-16382}	2^{-16382}
Самое большое положительное число	$2^{b-n+k+1} \cdot (2^{n-k}-1)$	$2^{104} \cdot (2^{24}-1)$	$2^{971} \cdot (2^{53}-1)$	$2^{16319} \cdot (2^{65}-1)$	$2^{16271} \cdot (2^{113}-1)$
Самое большое число, меньшее 1	$1-2^{-n+k}$	$1-2^{-24}$	$1-2^{-53}$	$1-2^{-65}$	$1-2^{-113}$
Самое маленькое число, большее 1	$1+2^{-n+k+1}$	$1+2^{-23}$	$1+2^{-52}$	$1+2^{-64}$	$1+2^{-112}$

Таблица 1. Граничные значения чисел с плавающей точкой.

Будем называть действительные числа, представимые в виде чисел с плавающей точкой, просто *представимыми* (если из контекста понятно, о какой точности идет речь, или, наоборот, это не имеет значения). Ясно, что не каждое действительное число представимо. Например, π — иррационально, а все представимые числа рациональны, точнее даже двоично-рациональны, т.е. имеют вид $p/2^m$, где p и m — целые числа, причем m неотрицательно. 1/3 тоже не представимо, поскольку не является двоично-рациональным числом.

Числа $2^{10000000}$ и $2^{-10000000}$ не представимы, хотя они двоично-рациональны. Первое слишком велико, а второе слишком мало по абсолютной величине, даже для четырехкратной точности. Для их представления нужна экспонента, занимающая минимум 25 бит. Числа, слишком близкие к представимым, тоже не представимы, например, для представления $1+2^{-1000}$ нужна мантисса из 999 нулей и одной единицы.

Заметим, что существует число с плавающей точкой -0, отличающееся от 0. Стандарт IEEE 754 требует, однако, считать их равными. Кроме того, ни одна из операций над числами с плавающей точкой, описанных в этом стандарте, не должна давать в результате -0, за исключением, вычитания x из x при режиме округления к $-\infty$ (см. далее) и, почему-то, квадратного корня из -0. В остальных ситуациях всегда в случае нулевого результата возвращается 0.

Как видно, представимые числа распределены среди действительных чисел достаточно специфическим образом. Примерная картина этого распределения показана на Рис. 1.

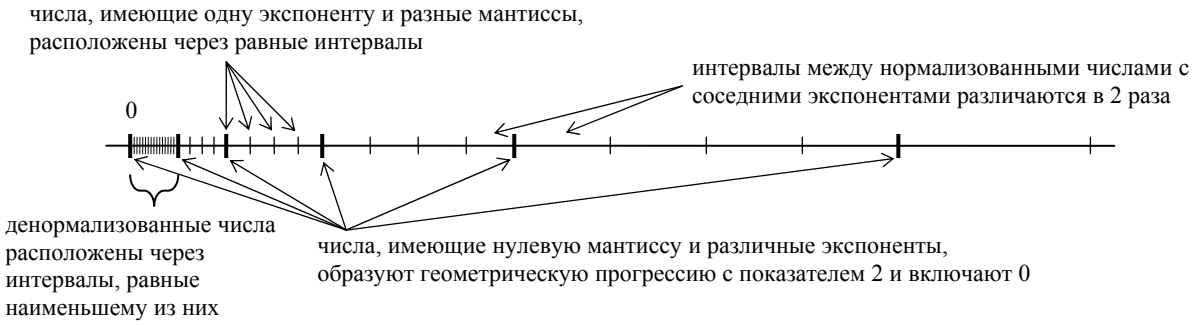

Рисунок 1. Примерная картина распределения чисел с плавающей точкой.

2.2. Трудности выполнения корректных вычислений

Не все действительные числа представимы, поэтому большинство вычислений с числами с плавающей точкой являются приближенными. Но неточность их результатов сама по себе — не основная проблема. Более важную роль играет накопление ошибок во время выполнения сложных вычислений, когда множество мелких неточностей в промежуточных вычислениях полностью искажают итоговый результат.

Кроме того, поскольку примерно постоянной остается величина относительной погрешности приближения действительного числа представимым, ошибки округления сильно зависят от абсолютной величины чисел, с которыми производятся действия. Поэтому, если в ходе вычислений происходит сокращение двух больших чисел за счет вычитания или деления, то результат часто имеет значительную погрешность — этот эффект называется *катастрофическим сокращением* (catastrophic cancellation).

Приведем несколько примеров возникновения существенных погрешностей при вычислениях с плавающей точкой.

Гармонические суммы. Гармонической суммой H_n называется сумма чисел, обратных к целым, от 1 до n: $H_n = \sum_{k=1}^{n} 1/k$. Такие суммы довольно часто возникают в комбинаторных задачах или как элемент рядов для некоторых специальных функций.

Если гармоническую сумму вычислять в прямом порядке, суммируя 1, 1/2, 1/3, и т.д., слагаемые постепенно становятся все меньше и в итоге станут меньше, чем разность между текущей суммой и следующим представимым числом. Поэтому вычисляемые суммы, начиная с некоторого n (зависящего от точности), не будут увеличиваться.

Вычисляемое так значение H_n перестает расти, когда $1/n$ становится меньше половины последнего бита мантиссы текущего значения H_n. Это случается для однократной точности при $n = 2^{21} = 2097152$, а для двойной точности при $n = 2^{48} = 281474976710656$. Соответствующие значения H_n — 15.133306695… и 33.8482803317789.... Реально же из-за накопления ошибок округления для однократной точности получается, что $H_{2097152}$ равно 15.403683 (здесь только 2 правильных цифры!).

Наиболее простой способ исправить это — считать сумму, начиная с самых маленьких ее членов. В этом случае $H_{2097152}$ для однократной точности получается равной 15.132898 (уже 4 верных цифры). Но еще лучше использовать асимптотику.

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \frac{1}{252n^6} + \frac{1}{240n^8} - \ldots - \frac{B_{2k}}{2k \cdot n^{2k}} - \ldots,$$

где γ = 0.5772156649... — постоянная Эйлера-Маскерони, а B_{2k} — число Бернулли с номером $2k$. Использование только первых трех слагаемых уже дает для $H_{2097152}$ значение 15.1333065, в котором только последняя цифра отличается от верной, а для $H_{281474976710656}$ при этом получается значение со всеми верными цифрами.

Площадь игольчатых треугольников. Игольчатым назовем треугольник, в котором две стороны примерно равны по длине и обе значительно длиннее третьей. Для вычисления его площади по длинам сторон можно использовать формулу Герона.

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где a, b, c — длины сторон, а $p = (a+b+c)/2$ — полупериметр.

Взяв $a = b = 10^7$ и $c = 1$ и вычисляя площадь такого треугольника по этой формуле, получим для однократной точности результат 0.0, а для двойной 4999999.9999999935 (в нем все цифры, кроме последней, верные).

Нулевой результат при однократной точности объясняется тем, что вычисляемый полупериметр получается в точности равным a и b — его точное значение 1000000.5 непредставимо в числах с однократной точностью. Это пример катастрофического сокращения, которое в данном случае сделало результат абсолютно бессмысленным.

Чтобы исключить подобные результаты, алгоритм выполнения вычислений должен быть устойчив к возникающим ошибкам. Используемые формулы нужно преобразовывать так, чтобы в них не было действий, в которых возможно катастрофическое сокращение. Для вычисления площади треугольника такой алгоритм состоит в следующем [11].

Необходимо упорядочить длины сторон так, чтобы выполнялось $a \geq b \geq c$. После этого площадь вычисляется по такой формуле.

$$S = \tfrac{1}{4}\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}$$

Порядок действий должен быть в точности тот, что указывается скобками. При использовании этой формулы для вычислений с однократной точностью в нашем примере получается 5000000.0, что значительно ближе к правильному результату.

Неустойчивая последовательность. В обоих описанных выше примерах можно найти алгоритм, устойчивый к возникающим ошибкам. К сожалению, не для всех практически значимых задач такие алгоритмы разработаны. Более того, для некоторых задач их просто не существует, потому что задача сама по себе неустойчива. Таковы многие задачи популяционной динамики и гидродинамики. Приведем здесь лишь один, придуманный J.-M. Muller'ом [12] не очень практичный пример, который, однако, очень ярко демонстрирует возникающие эффекты.

Определим последовательность x_n следующим образом: $x_0 = 2$, $x_1 = -4$ и

$$x_{n+2} = 111 - \frac{1130}{x_{n+1}} + \frac{3000}{x_{n+1}x_n}.$$

Можно доказать, что эта последовательность имеет предел при $n \to +\infty$, равный 6. Однако при вычислениях с плавающей точкой, какая бы точность не была выбрана

(можно выделять произвольно большое количество бит под мантиссу и экспоненту), вычисляемые значения x_n стремятся к 100. Чем выше точность вычислений, тем позже эти значения отклоняются от правильных, но это всегда происходит.

Таким образом, эта последовательность демонстрирует, что иногда вычисления с любой фиксированной точностью могут давать полностью неверные результаты.

Инциденты, связанные с вычислениями с плавающей точкой. Помимо чисто математических или модельных эффектов неаккуратный учет погрешностей и их накопление в вычислительных алгоритмах могут приводить к серьезным проблемам в реальной жизни.

Один из печальных инцидентов, связанных с вычислительными ошибками, произошел во время операции «Буря в пустыне» в 1991 году [13]. Тогда батарея ПВО США на основе ракет Patriot не смогла отследить и сбить иракскую ракету «Скад». Та поразила барак американской армии, в результате чего 28 солдат погибло и около 100 было ранено. По результатам расследования причиной сбоя в работе батареи ПВО было названо накопление погрешностей при учете времени в системе, управлявшей батареей.

Время считалось в десятых долях секунды от очередного начала работы системы, а все операции с характеристиками движения отслеживаемых объектов выполнялись на 24-битном процессоре. К моменту инцидента батарея работала без перезагрузки в течение примерно 4-х суток. В результате за счет умножения исходной погрешности представления 1/10 секунды в виде двоичной дроби на большое число прошедших десятых долей секунды накопилась существенная погрешность в определении времени (около 1/3 секунды). Учитывая, что ракета «Скад» летит со скоростью около 1700 м/с, ошибка в оценке времени на сотые доли секунды уже делает ее перехват невозможным.

Одна из наиболее дорогих ошибок ПО в истории, приведшая к катастрофе при первом запуске ракеты Ariane 5 [14], также связана с неаккуратными вычислениями. Авария случилась из-за того, что для учета горизонтальной составляющей скорости ракеты использовалось значение с однократной точностью, которой ранее вполне хватало для Ariane 4. Однако новая ракета могла двигаться с гораздо большей скоростью, что привело к переполнению. Возникшее исключение система попыталась обработать, заставив заново пересчитать вызвавший проблему результат на другом процессоре. Это привело к повторному исключению и попытке использовать для расчета траектории устаревшие на тот момент данные. В результате ракета повернулась боком, начала болтаться и была уничтожена из-за невозможности продолжать управляемый полет.

2.3 Требования стандартов к реализациям математических функций

Рассмотренные примеры показывают, что большое значение имеет само наличие стандартов, определяющих требования к компонентам математических библиотек и делающих работающее на их основе ПО переносимым. Однако важно и конкретное содержание этих требований, которое может существенно влиять на точность выполняемых вычислений. Рассмотрим требования имеющихся на настоящий момент стандартов, касающиеся реализаций математических функций.

Стандарты IEEE 754 и IEEE 854. Для представления результатов вычислений в виде чисел с плавающей точкой стандарт IEEE 754 определяет четыре *режима округления*: к ближайшему числу, вверх, вниз и к нулю. При первом режиме в качестве результата должно возвращаться представимое число, ближайшее к точному результату (если же он находится ровно посередине между двумя представимыми числами, нужно возвращать то из них, у которого в мантиссе последний бит равен 0), при втором — наименьшее представимое число, не превосходящее точный результат, при третьем — наибольшее представимое число, меньше или равное точному результату, а при

последнем для положительного точного результата округление происходит как в третьем режиме, а для отрицательного — как во втором.

При выходе точного результата за границы представимых значений должны возвращаться разные значения при разных режимах округления. При округлении к ближайшему, если точное значение отличается от максимального или минимального представимого больше, чем на половину значения единицы последнего разряда мантиссы (это погрешность в 0.5ulp, где ulp обозначает *единицу последнего разряда мантиссы, unit in the last place* [10]) — то возвращается, соответственно, +∞ или -∞. При режиме округления к 0 при переполнении возвращается максимальное или минимальное представимое значение, в зависимости от знака точного результата. При режиме округления к +∞ (или -∞) при положительном знаке точного результата возвращается +∞ (максимальное представимое число), а при отрицательном — минимальное представимое число (-∞). Кроме того, должен выставляться специальный флаг переполнения (overflow). Если точный результат бесконечен, возвращается бесконечность с соответствующим знаком и выставляется специальный флаг деления на ноль (divide by zero).

При получении результата, не равного 0, но меньшего по модулю, чем минимальное положительное нормализованное число, должен выставляться специальный флаг малого результата (underflow). Во всех случаях, когда точный результат непредставим, должен выставляться флаг неточного результата (inexact). В тех случаях, когда результат никак невозможно интерпретировать, например, при извлечении квадратного корня из -1 или при вычислении 0/0 или (-∞) + (+∞), должно возвращаться NaN и должен выставляться специальный флаг неправильной операции (invalid).

Однако IEEE 754 и IEEE 854 накладывают перечисленные требования только на арифметические действия (сложение, вычитание, умножение, деление), приведение типов, взятие остатка по модулю и извлечение квадратного корня. Ни про одну другую функцию ничего не говорится.

Стандарты библиотек языка C. Точно так же мало говорят о других функциях стандарт языка C ISO/IEC 9899 [15] и стандарт переносимого интерфейса операционной системы IEEE 1003.1 [16] (известный как POSIX), описывающие библиотеку математических функций языка C.

Стандарт языка C ссылается на требования IEEE 754, добавляя только ограничения на значения результатов ряда функций для некоторых специальных значений параметров (например, $exp(0) = 1$, а $sin(0) = 0$ — эти требования находятся в приложении F к стандарту ISO C). POSIX, в свою очередь, ссылается на требования стандарта языка C, добавляя описание поведения реализаций математических функций в случае возникновения переполнения или слишком маленьких результатов, а также для тех значений параметров, где соответствующая функция не определена.

Скажем, для функции *sin* POSIX [17] требует, чтобы $sin(x) = x$ при $x = 0$, -0 и при денормализованных значениях x. Кроме того, значение синуса для NaN, -∞ и +∞ должно быть NaN. При денормализованных аргументах должен также выставляться флаг ошибки выхода за нормализованные значения (range error), а при значениях аргумента -∞ или +∞ должен выставляться флаг недопустимого аргумента (domain error). Никаких других ограничений на реализации этой функции не накладывается.

Подобные требования напоминают казус с одним из первых советских спутников, в системе управления которым уже после успешного выполнения полетной программы было обнаружено, что синус любого положительного числа считался равным 1/2 — разработчики забыли вовремя заменить отладочную версию этой функции рабочей.

Отсутствие четких требований к большинству математических функций в этих стандартах можно объяснить только желанием дать время для аккуратной реализации уже сформулированных в IEEE 754 ограничений, что в середине 80-х годов было не так просто. Однако с тех пор прошло уже достаточно много времени. На практике такая ситуация иногда приводит к серьезным ошибкам в работе приложений для математического моделирования и достаточно часто — к тому, что на разных платформах такие приложения выдают существенно отличающиеся результаты.

Стандарты ISO/IEC 10967. В последние 5-8 лет появились предложения стандартизовать необходимые для аккуратного моделирования требования к реализациям математических функций [7,8]. Многие инициаторы этой деятельности работают в проекте Arenaire [6], совместно проводимом во Франции INRIA, CNRS и Высшей Нормальной школой Лиона. В результате появился набор стандартов ICO/IEC 10967 [3-5] (третья его часть еще находится в стадии обсуждения), формулирующий естественные и не зависящие от используемого языка программирования ограничения на работу реализаций элементарных функций — корней, экспонент и логарифмов с различными основаниями, гиперболических и тригонометрических функций, а также обратных к ним. Эти ограничения касаются нескольких аспектов.

- Возможные погрешности вычисления функций выражены в терминах *единиц последнего разряда (unit in the last place, ulp)* [10]. Округление к ближайшему представимому числу дает погрешность, не превосходящую 0.5 ulp, т.е. вычисленный результат отличается от точного значения не более чем на половину единицы последнего разряда мантиссы результата.
  Однако природа математических функций такова, что такая точность не всегда является практически обоснованной. Число с плавающей точкой является приближенным представлением любого действительного числа, к которому оно является ближайшим. Таким образом, уже в значениях аргументов функции может иметься погрешность, которую ее вычисление не в силах исправить. В стандартах ICO/IEC 10967 сформулированы более практичные требования к точности вычислений, ограничивающие погрешность результата величиной от 0.5 ulp до 2 ulp, в зависимости от функции [4].

- Стандарты серии ISO/IEC 10967 требуют от реализации математической функции сохранения знака ее точного значения для данного значения параметра. Также, на всех интервалах монотонности функции ее реализации должны быть монотонны таким же образом, т.е. там, где сама функция убывает, ее численная реализация должна убывать, а там, где функция возрастает, — возрастать.
  Исключением из этих правил являются тригонометрические функции в области больших значений аргумента, где интервал смены знака и интервал монотонности становятся сравнимы с единицей последнего разряда аргумента.

- ISO/IEC 10967 требует соблюдения специфических требований при вычислении функций в окрестностях точек, где они имеют известные представимые значения.
  Например, реализация экспоненты для значений аргументов, достаточно близких к 0, должна возвращать в точности 1.
  Это требование связано с тем, что плотность представимых чисел в окрестности 0 гораздо больше, чем их плотность в окрестности 1. Сравните расстояния до ближайших соседних чисел двойной точности для 1 и для 0 — 2^{-53} и 2^{-1074}.

2.4. Дилемма составителя таблиц

Сформулированные требования к точности вычисления результатов математических функций приводят к так называемой *дилемме составителя таблиц (Table Maker's Dilemma)* [18-20].

Эта проблема состоит в том, что для выбора правильно округленного ближайшего числа с плавающей точкой при приближенных вычислениях иногда нужно вычислить много дополнительных бит мантиссы результата, значительно больше, чем имеется в рассматриваемом типе чисел с плавающей точкой.

Название проблемы происходит из тех времен, когда таблицы значений математических функций — логарифмов, экспонент, синусов и косинусов — составлялись вручную. Составитель таблиц вычислял на одну-две цифры больше, чем помещалось в итоговые таблицы, чтобы правильно выполнять округление. Однако бывают случаи, когда вычисленных дополнительных цифр не хватает, чтобы правильно выбрать результат округления. В таких случаях составители таблиц часто выбирали результат из двух возможных произвольным образом.

Например, пусть вычисляется функция *sin* для двоичных чисел с плавающей точкой, имеющих 6 битов мантиссы. Синус числа (выделены биты мантиссы) $11.1010_2 = 3.625_{10}$ равен $0.0\mathbf{1110110}1111110..._2 = 0.063225984913\ldots_{10}$. Приближенное вычисление 6-ти бит мантиссы результата может дать как 0.0111011_2, так и 0.0111100_2, поскольку точное значение очень близко к их среднему арифметическому. Только зная точный 14-й бит, мы можем уверенно выбрать первое из них в качестве значения, ближайшего к точному результату.

При режиме округления к ближайшему описанная проблема проявляется, когда значение функции, само не являясь представимым, лежит очень близко к среднему арифметическому двух представимых чисел. При режимах округления к 0, к $+\infty$ или к $-\infty$ она возникает, если значение функции, не являясь представимым, лежит настолько близко к представимому числу, что только за счет существенных дополнительных вычислений можно понять, превосходит оно это число или нет.

Приведем примеры обоих случаев проявления дилеммы составителя таблиц для чисел двойной точности. Вычисляя значение натурального логарифма для числа $1.613955DC802F8_{16}\cdot 2^{-35}$, получим $-17.F02F9BAF6035\ 7F^{14}9\ldots_{16}$ (F^{14} здесь означает, что цифра F повторяется 14 раз, что, учитывая две соседние цифры, дает 60 повторяющихся единиц). Полученное значение лежит почти точно посередине между $-17.F02F9BAF6035_{16}$ и $-17.F02F9BAF6036_{16}$, но чуть ближе к первому.

Таким образом, для получения корректно округленного значения при режиме округления к ближайшему нужно вычислять логарифм в этой точке с относительной погрешностью, не превосходящей 2^{-114}. Большее значение погрешности не дает нам возможности определить, какое из двух указанных представимых чисел ближе к точному значению логарифма.

Натуральный логарифм числа $1AC.50B409C8AEE_{16}$ равен $1.83D4BCDEBB3F3\ F^{15}A..._{16}\cdot 2^2$. Вычисляя логарифм в этой точке при режимах округления к 0, к $+\infty$ или к $-\infty$, необходимо выяснить, превосходит ли его значение число $1.83D4BCDEBB3F4_{16}\cdot 2^2$ или нет. Для этого его нужно вычислить также с относительной погрешностью, не превосходящей 2^{-114}

Общий вид чисел, трудных для округления, представлен на Рис. 2.

Описанные примеры показывают, что для получения корректно округленного результат иногда нужно использовать гораздо более точные вычисления, чем это возможно в рамках чисел с плавающей точкой заданной точности.

Для большинства часто используемых функций их значения для «неособых» (т.е. не равных, например, 0 или 1) двоично-рациональных значений аргумента не являются рациональными, и поэтому не могут ни быть представимыми, ни лежать в точности посередине между двумя представимыми числами. Для экспонент, гиперболических, тригонометрических и обратных им функций это следует из того, что возведение числа *e* в ненулевую алгебраическую степень (даже комплексную) дает трансцендентный результат (см., например, [21]).

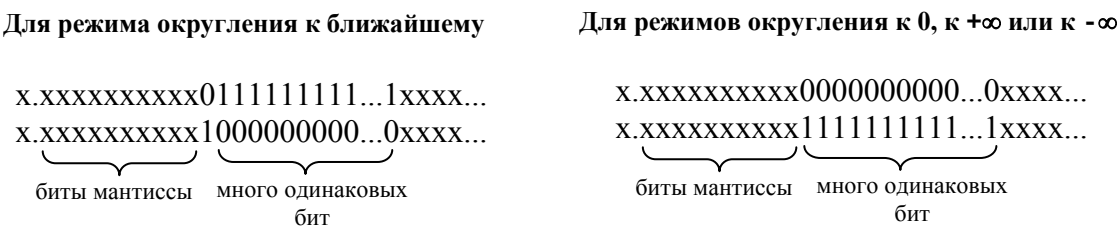

Рисунок 2. Общий вид значений функций, на которых проявляется дилемма составителя таблиц.

В силу конечности множества представимых чисел это означает, что для каждой функции есть такое число $\varepsilon > 0$, что вычисляя значения этой функции с погрешностью, не превосходящей ε, можно всегда точно определить корректное округление, являющееся представимым числом, при любом режиме округления. Однако вычислять функцию с такой точностью для всех значений аргумента может оказаться слишком неэффективно. Например, для натурального логарифма на числах двойной точности при произвольном режиме округления такое ε можно взять равным 2^{-118} [20]. Однако реально такая точность нужна только для единственного значения аргумента, во всех остальных случаях можно использовать меньшую. Для подавляющего же большинства представимых чисел двойной точности корректное округление их логарифма можно получить, вычисляя его с погрешностью, не превосходящей 2^{-54}.

Дилемма составителя таблиц приводит к необходимости использования значительно более точных вычислений, чем это позволяют сделать стандартные типы чисел с плавающей точкой, как при построении правильных реализаций математических функций, так и проверке их корректности. Однако проводить настолько точные вычисления для всех значений аргументов слишком неэффективно. Поэтому, для повышения эффективности вычислений нужно уметь выбирать их точность в зависимости от текущих значений аргументов функции.

3. Обзор работ по тестированию реализаций математических функций

Методам вычисления математических функций посвящено огромное количество работ. Одним из классических трудов на эту тему является сборник статей под редакцией Abramowitz и Stegun [22], хотя он был выпущен уже довольно давно и частично устарел. Более современное изложение методов вычисления элементарных функций (только подмножества рассмотренных в [22]) можно найти в книге Muller [23].

Исследования, посвященные систематическому тестированию реализаций математических функций над числами с плавающей точкой, встречаются гораздо реже. Хотя в Интернет можно найти огромное количество различных программ для тестирования таких функций (см. например, [24]), подавляющее большинство таких тестов крайне несистематично и проверяет какой-то один аспект вычислений, реже — два-три таких аспекта.

Как указывается в [24] (см. также [25,26]), несмотря на то, что стандартизация вычислений над числами с плавающей точкой началась более 20 лет назад, до сих пор многие поставщики библиотек и аппаратного обеспечения не придерживаются

имеющихся стандартов достаточно строго, поэтому тесты на правильность поведения реализаций математических функций по-прежнему необходимы.

Среди наиболее систематичных работ по тестированию вычислений с плавающей точкой можно назвать следующие.

Работы по тестированию на соответствие стандарту IEEE 754.

- В работе [27] описывается самый первый из известных систематических тестовых наборов для проверки корректности реализации операций над числами с плавающей точкой. Он появился еще до введения в действие стандарта IEEE 754, сделан в виде набора программ на Fortran и предназначен для тестирования только сложения, вычитания, умножения и деления.
- Специально для проверки на соответствие IEEE 754 был разработан тестовый набор, который описан в статьях [28,29] и может быть получен с сайта [30]. В этом наборе проверяются все требования стандарта к арифметическим операциям, вычислению квадратных корней и взятию остатков, а также преобразования между типами чисел с плавающей точкой и целыми.
- Программа PARANOIA [31,32] была создана одним из авторов стандарта IEEE 754 Кэханом (W. Kahan) и остается довольно популярным средством проверки на соответствие ему, хотя такая проверка менее тщательна, чем с помощью указанного выше тестового набора. Она проверяет только базовые арифметические операции и извлечение квадратного корня.
- Другой подход к построению тестов для операций IEEE 754 используется в среде FPgen [33,34]. Здесь в качестве тестовых данных, помимо специальных значений, используются числа с плавающей точкой, удовлетворяющие некоторым шаблонам — например те, в которых нулевые и единичные биты мантиссы чередуются, или в которых мантисса содержит ровно 7 единиц.

Работы по тестированию широкого набора математических функций.

- Тестовый набор ELEFUNT [35] содержит тесты для нескольких математических функций в виде программ на C и Java. В этих тестах проверяется корректность возвращаемых значений для специальных значений аргументов (0, 1, +∞, -∞, NaN), а также для ряда генерируемых случайно значений аргументов проверяется выполнение некоторых тождеств, связанных с данной функцией (например, $exp(x)\cdot exp(-x) = 1$).
- Тестовый набор UCBTEST [36] предназначен для тестирования базовых арифметических действий и достаточно широкого набора математических функций (тестируется больше функций, чем в ELEFUNT). Он оформлен как набор тестовых программ на Fortran и C и предопределенных входных данных для разных функций. В каждом тесте проверяется, что для заданных значений параметров данная функция возвращает число с плавающей точкой, равное правильному значению, вычисленному с учетом используемого режима округления, или же достаточно близкое к нему.
  Методика выбора тестовых данных для этого тестового набора не описана. Как можно заключить по результатам их анализа, использовалось несколько разных идей.
  - Выделялись особые с точки зрения их структуры значения чисел с плавающей точкой: 0, -0, NaN +∞, -∞, минимальное положительное, максимальное положительное, минимальные и максимальные положительные денормализованные числа и пр.

- Выделялись значения аргументов, значение функции для которых может быть точно представлено числом с плавающей точкой (например, $sin(0) = 0$, $cos(0) = 1$ и пр.).
- Некоторые значения аргументов выбирались из соображений, связанных со структурой известных алгоритмов для вычисления элементарных функций. Например, в ряде алгоритмов вычисления логарифма сначала значение аргумента при помощи умножения или деления на 2 приводится к интервалу (0.5, 1]. Соответственно, в качестве тестовых значений выбираются границы этого интервала и нескольких соседних с ним, а также близкие к этим границам числа.

- Аналогичные подходы — использование ряда специальных значений, границ интервалов, определяемых часто используемыми алгоритмами вычисления данной функции, а также чисел, построенных по некоторым шаблонам и случайных значений — применялись для построения более объемных тестовых наборов, например, набора Беркли [37].

Работы, выполнявшиеся в проекте Arenaire. Отдельно стоит отметить проект Arenaire [6], который, помимо исследований по тестированию включает большое количество работ по разработке эффективных реализаций математических функций и их аналитической верификации.

- Работы [18-20,38], посвященные дилемме составителя таблиц и поиску чисел, для которых корректное вычисление функций с заданной точностью наиболее трудоемко. Эти числа можно использовать в качестве «неудобных» тестовых значений для практически любой реализации соответствующей функции.

- В рамках того же проекта разрабатывается инструмент MPCheck [39] для тестирования корректности реализации элементарных функций с точки зрения сохранения монотонности и корректности округления. В качестве тестовых данных в нем используются числа, построенные по некоторым шаблонам.

В целом, имеющиеся в открытом доступе исследования, посвященные выработке требований к реализациям математических функций и тестированию на соответствие этим требованиям, не содержат систематического подхода к этим вопросам, объединяющего рассмотрение всех указанных проблем. Нигде, кроме работ группы Arenaire, не рассматривается дилемма составителя таблиц и значения аргументов, для которых вычисление корректных результатов функций наиболее трудоемко. В то же время, в работах группы Arenaire никак не фигурируют тестовые данных, основанные на границах интервалов, специфичных для данной функции.

Наиболее систематично требования к поведению математических функций представлены в стандартах ISO/IEC 10967. Однако использованная при их создании методика не описана явно, что необходимо для расширения их на более широкое множество функций, например, охватывающее всю стандартную библиотеку языка C, включающую не являющиеся элементарными гамма-функцию и функции Бесселя.

Кроме того, на настоящий момент нет тестовых наборов, проверяющих соответствие ISO/IEC 10967 и в открытом доступе нет какой-либо информации о том, что разработка таких тестовых наборов ведется.

Все это делает актуальной разработку систематических методов формулировки требований к реализациям математических функций и методов построения соответствующих тестов.

4. Предлагаемый подход

В рамках предлагаемого подхода были разработаны метод определения требований к реализации конкретной математической функции и метод выбора тестовых данных для тестирования конкретной функции на соответствие сформулированным требованиям.

4.1. Метод определения требований к математическим функциям

Данный метод определения требований к реализациям математических функций заимствует б*о*льшую часть идей из стандарта ISO 10967 [3-5] и работ [7,8], посвященных разработке стандартов с повышенными требованиями к корректности вычисления математических функций. В то же время, некоторые элементы предлагаемого метода не встречаются в доступной литературе в явном виде, хотя они обобщают часто использующиеся приемы. Каждый пункт, приведенный ниже, снабжен ссылкой на источник, где встречаются похожие идеи или на общую практику, если данный пункт обобщает принятые подходы к решению подобных задач.

Здесь представлен метод определения требований к результату функции и не рассматриваются требования, касающиеся выставления флагов, посылки сигналов или создания исключительных ситуаций. В общем случае они должны формулироваться в соответствии с общими положениями IEEE 754, но большое количество деталей, которые нужно при этом учитывать, делает формулировку точных правил для этого аспекта поведения функций предметом отдельного исследования.

Требования к результатам реализации математической функции могут быть разделены на несколько аспектов, которые должны рассматриваться отдельно.

- **Область определения функции и особые точки функции.**
  - *Область определения* (общая практика). Для всех значений аргументов, где математическая функция определена, ее реализация должна возвращать некоторый результат, который может быть равен $+\infty$ или $-\infty$, если значение самой функции находится за пределами интервала чисел с плавающей точкой и режим округления требует этого, но не должен быть NaN.
  - *Предельные значения* ([8]). Для всех значений, для которых математическая функция не определена (в том числе, и для ее особых точек), но имеет однозначно определенный предел, может быть, равный $+\infty$ или $-\infty$, реализация должна возвращать значение этого предела.
  - *Односторонние пределы в 0* (практика формулировки требований к конкретным функциям). Если функция имеет особенность в точке 0, но не имеет там предела, равного $+\infty$ или $-\infty$, нужно рассматривать односторонние пределы функции. Значение реализации функции в 0 нужно считать равным ее пределу при $x \to +0$, если он существует, а значение в -0 — пределу при $x \to -0$, если он есть. Примером такой функции служит котангенс — обычно считается, что $ctg(0) = +\infty$ и $ctg(-0) = -\infty$ (см. ниже о нечетных функциях).
  - *Полная неопределенность* ([8]). В остальных случаях должен возвращаться результат NaN.
    Примеры таких ситуаций: $sqrt(-1.0) = ln(-1.0) = \text{NaN}$, $sin(+\infty) = \text{NaN}$.
    Особо нужно рассматривать такие значения аргументов, по поводу которых нет однозначного мнения о возможном продолжении функции в эту точку по непрерывности. Примером служит значение 0^0, которое иногда интерпретируется как 1, а иногда как NaN.
  - *Окрестности полюсов* (практика формулировки требований к конкретным функциям, см. [4]). Для полюсов функции, где ее значение стремится к

бесконечности, необходимо точно определить окрестности, в которых оно уже не является представимым. При наличии представимых чисел в такой окрестности, реализация функции для них должна возвращать значения +∞, -∞, максимальное или минимальное представимое число в соответствии со знаком точного значения и режимом округления.

Например, котангенс имеет полюс в точке 0 и ведет себя в ее окрестности примерно как $1/x$. Для многих денормализованных чисел обратное к ним число не является представимым, поэтому котангенс для многих чисел в окрестности 0 должен быть возвращать +∞ или -∞, в зависимости от знака аргумента. Значение котангенса для 2^{-1024} равно $1.\text{FFFFFFFFFFFFF}\ \text{F}^{499}5\ldots_{16}\cdot 2^{1023}$. Поэтому при режимах округления к 0 или к -∞ котангенс для всех положительных чисел, не превосходящих 2^{-1024}, должен возвращать максимальное положительное представимое (с двойной точностью) число. При режимах округления к ближайшему или к +∞ котангенс должен в этом интервале возвращать +∞. В этих режимах его значение попадает в интервал представимых чисел, только начиная с числа $1.0000000000001_{16}\cdot 2^{-1024}$.

- *Окрестности бесконечностей при бесконечных пределах* (практика формулировки требований к конкретным функциям, [4]). Для функций, стремящихся к бесконечности при $x \to +\infty$ или $x \to -\infty$, должны быть точно определены пределы представимости их значений. За этим пределами реализация также должна возвращать +∞, -∞, максимальное или минимальное представимое число в соответствии со знаком точного значения и режимом округления.

  Например, $exp(x)$ стремится к +∞ при $x \to +\infty$. Максимальное число двойной точности, для которого значение $exp(x)$ тоже представимо с двойной точностью, равно $1.62\text{E}42\text{FEFA}39\text{EF}_{16}\cdot 2^9 = 709.7827\ldots_{10}$. Для всех чисел, превосходящих его, $exp(x)$, вычисляемая с двойной точностью, должна возвращать +∞ для режимов округления к ближайшему и к +∞ и максимальное представимое для двух других режимов.

  Для чисел однократной точности аналогичная граница равна $1.62\text{E}42\text{E}_{16}\cdot 2^6 = 88.7228\ldots_{10}$.

- **Специальные значения, значения в 0, касательные и асимптоты.**

  - *Бесконечности и* -0 (общая практика). Нужно наиболее естественным образом определить значения функции для особых значений аргумента: -0, +∞, -∞. Обычно достаточно определять их как пределы, если те существуют, иначе как NaN.

    Многие библиотечные функции являются четными или нечетными. Для первых естественно иметь $f(-0) = 0$, а для вторых $f(-0) = -0$.

    Заметим, что из этого правила выпадает странное требование IEEE 754, гласящее, что $sqrt(-0) = -0$. Разумнее было бы определить это значение равным 0 (NaN подходит по иным соображениям, но приводит к ситуации $x = y \ \&\ sqrt(x) \neq sqrt(y)$, так как считается, что $0 = -0$).

  - *Значение в* NaN (следует из требований IEEE 754). Значение функции для аргумента NaN должно быть равно NaN.

  - *Точные значения* (практика формулировки требований к конкретным функциям, [4,16]). Для некоторых значений аргумента значения функции известны точно. Если оба значения представимы, надо требовать от реализации функции возвращать точное ее значение в таких точках.

Например, $exp(0) = cos(0) = ch(0) = 1$, $sin(0) = sh(0) = tg(0) = arcsin(0) = 0$, $ln(1) = 0$ и т.п.

- *Окрестности экстремумов, являющихся точными значениями* (практика формулировки требований к конкретным функциям, [4]). Кроме этого, если в такой точке производная функции равна 0, то для любого аргумента из некоторой ее окрестности реализация должна возвращать то же самое значение.
  Например, косинус имеет нулевую производную в точке 0, и поэтому расстояние между его значением и 1 зависит квадратично от расстояния от аргумента до 0. Границы окрестности 0, на которой он должен возвращать 1, зависят от точности и используемого режима округления. Для двойной точности и режима округления к ближайшему граница проходит между $1.6A09E667F3BCC_{16}\cdot 2^{-27}$ и следующим за ним представимым числом, поскольку $cos(1.6A09E667F3BCC_{16}\cdot 2^{-27}) = 1.FFFFFFFFFFFFF\ 80^{12}6\ldots_{16}\cdot 2^{-1}$, а $cos(1.6A09E667F3BCD_{16}\cdot 2^{-27}) = 1.FFFFFFFFFFFFF\ 7F^{12}B\ldots_{16}\cdot 2^{-1}$.
  Для округления к $+\infty$ соответствующая граница равна 2^{-26}, поскольку $cos(1.0_{16}\cdot 2^{-26}) = 1.FFFFFFFFFFFFF\ 0^{13}1\ldots_{16}\cdot 2^{-1}$, а $cos(1.0000000000001_{16}\cdot 2^{-26}) = 1.FFFFFFFFFFFFE\ F^{12}C\ldots_{16}\cdot 2^{-1}$.
  Для режимов округления к 0 или $-\infty$ нигде, кроме 0 косинус не равен 1.
  Эти же границы для однократной точности и режимов округления к ближайшему и к $+\infty$ — 2^{-12} и $1.6A09E6_{16}\cdot 2^{-12}$.

- *Окрестность* 0 (практика формулировки требований к конкретным функциям, [4]). Если значение функции в 0 представимо и не равно 0. даже если производная функции в 0 ненулевая, должно быть выполнено то же самое правило: в некоторой окрестности 0 для всех чисел с плавающей точкой значение ее реализации должно быть одинаковым. Это следует из того, что около 0 плотность чисел с плавающей точкой больше, чем около любого другого значения.
  Например, для $exp(x)$ результат 1 должен возвращаться при двойной точности и режиме округления к $+\infty$, начиная с $-1.0_2\cdot 2^{-53}$ и кончая 0. При округлении к ближайшему это выполнено на отрезке от $-1.0_2\cdot 2^{-54}$ до $1.FFFFFFFFFFFFF_{16}\cdot 2^{-54}$. При округлении к 0 или $-\infty$ этот отрезок простирается от 0 до $1.FFFFFFFFFFFFF_{16}\cdot 2^{-53}$.

- *Горизонтальные асимптоты* (практика формулировки требований к конкретным функциям, [4]). В тех случаях, когда функция имеет горизонтальные асимптоты, необходимо аккуратно определить границы, после которых ее значение должно стать постоянным при определенном режиме округления.
  Так, $exp(x)$ при двойной точности и режиме округления к ближайшему для отрицательных чисел вплоть до (включая) $-1.74910D52D3051_{16}\cdot 2^{9}$ должна возвращать 0. При округлении к 0 или $-\infty$ такой же результат должен получаться для всех отрицательных чисел, меньших или равных $-1.74385446D71C4_{16}\cdot 2^{9}$. При округлении к $+\infty$ значение $exp(x)$ должно быть равно 0 только при $x = -\infty$, а для чисел, не превосходящих $-1.74385446D71C4_{16}\cdot 2^{9}$ должен возвращаться результат 2^{-1074}.

- *Асимптотики* (практика формулировки требований к конкретным функциям, [4,16]). Казалось бы, естественно предъявить аналогичные требования к функциям, имеющим негоризонтальные асимптоты или асимптотически близких к другим функциям. Например, $e^x \sim 1+x$ при $x \sim 0$, $ch(x) = (e^x+e^{-x})/2 \sim e^x$ при $x \sim +\infty$ и $ch(x) = (e^x+e^{-x})/2 \sim e^{-x}$ при $x \sim -\infty$, или

$sin(x) \sim x$ при $x \sim 0$.

Однако во многих случаях такие ограничения не могут быть сформулированы достаточно аккуратно с учетом различных режимов округления. Дело в том, что даже очень маленькая разность между двумя асимптотически близкими выражениями может приводить к различию в значимых битах мантиссы. Причина этого явления аналогична причине, порождающей дилемму составителя таблиц — слишком близкое расположение некоторых значений функции к представимым числам.

Например, если рассматривать асимптотику $e^x \sim 1+x$ при $x \sim 0$ для чисел двойной точности, то при $|x| < 2^{-28}$ разница между e^x и $1+x$ уже меньше 0.5 ulp, однако встречаются гораздо более близкие к 0 числа, представимые с двойной точностью, для которых мантиссы e^x и $1+x$ отличаются при выборе режимов округления к $-\infty$ или к $+\infty$.

Скажем, для $x = -1.8000000000001_{16} \cdot 2^{-52}$ $exp(x) =$ $1.FFFFFFFFFFFFD\ 0^{13}4\ldots_{16} \cdot 2^{-1}$, а $1 + x = 1.FFFFFFFFFFFFC\ F^{12}E_{16} \cdot 2^{-1}$.

Похоже, что четкие требования можно предъявлять лишь по поводу соблюдения асимптотик вида $f(x) \sim kx$ или $f(x) \sim -kx$, где k — некоторая представимая константа (например, $sin(x) \sim x$ и $tg(x) \sim x$ при $x \sim 0$), поскольку в таких ситуациях подобные эффекты не возникают. При этом нужно аккуратно вычислять границы действия этих ограничений при разных режимах округления.

Например, для реализации синуса для чисел двойной точности и округлении к ближайшему возвращаемый результат должен совпадать с аргументом на интервале $[-x_0, x_0]$, где $x_0 = 1.7137449123EF6_{16} \cdot 2^{-26}$. При округлении к $+\infty$ это должно быть выполнено на интервале от 0 до $x_1 = 1.D12ED0AF1A27F_{16} \cdot 2^{-26}$. При округлении к $-\infty$ то же самое должно выполняться на $[-x_1, 0]$. А при режиме округления к 0 $sin(x) = x$ только в точке $x = 0$. Заметим, что это отличается от требований POSIX (см. выше).

- **Область значений функции.**
  - *Выбор ветви* (общая практика). Если математическая функция в строгом смысле является многозначной, должна быть четко определена ее ветвь, которая будет соответствовать реализации. Граничные значения для этой ветви, которые часто можно выбрать многими способами, должны определяться, исходя из соображений наиболее широкого использования получаемого результата.

    Например, таковы обратные тригонометрические функции и обратный гиперболический косинус. Чаще всего $arctg(x)$ и $arcsin(x)$ считаются принимающим значения от $-\pi/2$ до $\pi/2$, $arcctg(x)$ и $arccos(x)$ — от 0 до π. Для $Arch(x)$, как и для функций извлечения корней четных степеней, выбирается ветвь, в которой эти функции имеют неотрицательные значения.
  - *Сохранение ограничений области значений* ([8]). Ограничения сверху или снизу на значения функции в рамках связной компоненты области ее определения нужно соблюдать и в ее реализации, в противном случае возможны различные неприятные эффекты.

    Например, при использовании в качестве результата реализации функции арктангенс на больших положительных аргументах числа с плавающей точкой, наиболее близкого к $\pi/2$ может оказаться, что это число больше $\pi/2$. Для чисел однократной точности это так: наиболее близкое к $\pi/2$ такое число это $1.921FB6_{16} = 13176795/8388608 > \pi/2$. При этом $tg(arctg(2^{30})) =$ $-2.2877\ldots \cdot 10^7$, что противоречит основному свойству обратных функций.

- *Денормализованные значения* (использовано при построения тестов для некоторых функций в UCBTEST [36]). Поскольку денормализованные числа выделяются из всего множества чисел с плавающей точкой, необходимо аккуратно определить интервалы, на которых значения функции должны быть денормализованными.
  Скажем, значение *exp*(*x*) при использовании двойной точности для отрицательных значений *x*, превосходящих некоторое, зависящее от режима округления (см. выше), остается денормализованным вплоть до $-1.6232BDD7ABCD3_{16}\cdot 2^9$. Для всех превосходящих это число значений аргумента значение *exp*(*x*) нормализовано при всех режимах округления.

- **Монотонность и сохранение знака.**
  - *Сохранение монотонности* ([4,8]). На всех интервалах, где математическая функция монотонна, ее реализация должна иметь тот же вид монотонности. Это необходимо для адекватного отражения существенных свойств математических моделей в их численном представлении.
    Это правило становится бессмысленным в тех областях, где функция часто меняет характер монотонности, т.е. длина интервала монотонности функции становится равна или меньше, чем единица последнего разряда мантиссы ее аргумента. Примерами таких функций являются тригонометрические функции (для них исключение из этого правила оговорено в [4]), а также функции Бесселя (не упоминаемые в [4]) при больших значениях аргумента.
  - *Сохранение знака* ([4,8]). Знак значения реализации функции для некоторого аргумента должен совпадать со знаком значения самой функции. Если значение функции равно 0 для какого-то представимого числа, значение ее реализации для этого числа также должно быть равно 0.
    В [4] оговаривается, что это требование также может не соблюдаться там, где единица последнего разряда аргументов становится больше величины интервала смены знака, например, для тригонометрических функций при больших значениях аргументов. Однако вычисление корректно округленного значения для результата функции делает такие оговорки ненужными.

- **Симметрии и периодичность.**
  - *Четность и нечетность* ([4,8]). Если математическая функция является четной или нечетной, этим же свойством должна обладать ее реализация.
  - *Горизонтальные симметрии* (замечание автора). В тех случаях, когда функция имеет симметрии относительно других представимых значений аргумента, например, выполняется правило $f(1-x) = -f(x)$, выполнение аналогичного свойства для реализации не всегда возможно, поскольку число $(1-x)$ может быть представимым при непредставимом x. Следует особо рассматривать такие случаи и накладывать ограничения, касающиеся только представимых значений аргумента с обеих сторон такого равенства. Если же функция симметрична относительно непредставимого значения, как, например, синус — $sin(\pi-x) = sin(x)$, его выполнение всегда может быть только приближенным и поэтому не должно строго оговариваться в требованиях.
  - *Другие симметрии* (замечание автора, обобщение предыдущих правил). Вообще, все важные функциональные уравнения, которым удовлетворяет данная функция, должны быть проанализированы на предмет выявления необходимости соблюдать их для реализаций этой функции и определения значений аргумента, для которых соблюдение этого равенства оправдано.

Пример свойств такого рода, выполнение которых нужно обеспечивать для некоторых значений аргумента — это свойства $\Gamma(1+x) = x\cdot\Gamma(x)$ и $\Gamma(1-x) = -x\cdot\Gamma(-x)$ гамма-функции $\Gamma(x)$. Иначе может нарушиться важное соотношение $\Gamma(n) = (n-1)!$ для целых положительных n.

Для свойства периодичности функции остаются верными все те же рассуждения. Если период представим, то нужно проверять это свойство только для чисел, представимых вместе со своим сдвигом на число, кратное периоду. Если период не представим, реализация может быть только приблизительно периодична.

Для аргументов, единица последнего разряда которых больше, чем период функции, проверка ее периодичности становится бессмысленной.

- **Корректное округление.**
  (Распространение правил IEEE 754, [8]) Помимо всех перечисленных ограничений, нужно требовать, чтобы результат, возвращаемый реализацией, получался из точного результата функции для данного аргумента при помощи принятой в текущей конфигурации процедуры округления.
  Можно отметить, что требования сохранения монотонности и сохранения знака автоматически выполняются при соблюдении правила корректного округления. Однако иногда результаты использования режимов округления к 0, к $+\infty$ или к $-\infty$ противоречат требованию сохранять ограничения области значений. В этом случае иногда можно принимать решение в зависимости от функции, потому что некоторые ее важные свойства могут нарушиться. Но чаще удобнее считать, что поддержка режима округления имеет более высокий приоритет, поскольку пользователь, применяющий такой режим, осведомлен о его последствиях. Иногда считается практически бессмысленным требовать точности 0.5-1 ulp, особенно в тех областях, где интервал существенного изменения значения функции становится меньше, чем единица последнего разряда. Автор, однако, убежден, что стандартные библиотечные реализации математических функций общего назначения должны всегда выдавать как можно более близкие к точным результаты, вне зависимости от их осмысленности при рассмотрении этих вычислений как приближенных. Это требование связано с тем, что большинство пользователей часто используют библиотечные реализации функций без всякого анализа свойств этих функций. Соответственно, они не в состоянии и заметить, что полученный результат неточен просто потому, что функция слишком быстро изменяется в окрестности используемого аргумента. Пользователь должен получать максимально корректный результат по отношению к введенным им аргументам. А рассуждения о практическом использовании этого результата лучше предоставить ему, ведь все-таки остается некоторая вероятность того, что ему действительно нужно правильное значение функции именно для таких аргументов.
  Те реализации, в которых снижение точности вычислений оправдывается повышением эффективности, не могут претендовать на звание стандартных и применимых в общем случае. Подобные реализации можно (и нужно) разрабатывать для решения специфических задач, но нужно четко определять ограничения на допустимые в них ошибки в различных областях значений аргументов реализуемых функций и отклонения от требований стандартов общего назначения.

4.2. Метод построения тестов на соответствие требованиям

Можно отметить, что использование описанного выше метода определения требований к реализации математической функции дает разбиение всех чисел с плавающей точкой

как ее аргументов на набор интервалов. Каждый интервал при этом характеризуется уникальным набором свойств функции.

На каждом из интервалов значение, возвращаемое реализацией функции, либо фиксировано, либо определяется некоторой асимптотикой, либо наиболее простое правило его вычисления — округление в соответствии с текущим режимом точного значения самой функции.

Для построения тестов необходимо уметь вычислять правильно округленное значение функции, а значит — значение функции с гораздо более высокой точностью, чем это возможно в рамках используемого типа чисел с плавающей точкой. Это можно сделать несколькими способами.

- С помощью систем символических вычислений и вычислений с произвольной точностью, например, Maple [40], Mathematica [41], MATLAB [42].
- С помощью библиотек корректно округляемых функций, таких, как разработанная на основе работ Ziv [43] в IBM Accurate Portable MathLib [44], GNU MPFR [45], libmcr [46], разрабатываемая в компании Sun, или библиотек SCSLib [47] и CRlibm [48], разрабатываемых в рамках проекта Arenaire [6,49-52].
- Можно разработать собственную реализацию функции на основе методов, изложенных в книгах [22,23] и многочисленных статьях (например, [43]), или на основе методов интервальных вычислений [53-56]. Интервальные вычисления обладают тем дополнительным преимуществом, что гарантируют правильность вычисляемых границ, в которых лежит нужный результат. В частности, интервальные вычисления позволяют получать корректные результаты даже в случае неустойчивости самой задачи. При работе с ними невозможен эффект неустойчивой последовательности, описанный в разделе 2.2.

Основные идеи описываемого далее метода выбора тестовых данных достаточно просты и могут быть сформулированы в следующем виде.

- В качестве тестовых значений используются границы полученных интервалов с уникальным набором свойств функции, а также границы интервалов чисел с плавающей точкой, обладающих определенными особенностями структуры, например, положительных денормализованных чисел.
- Некоторые точки внутри всех этих интервалов, выбираемые по определенным правилам, также используются как тестовые данные.
- Используются точки, для которых вычисление правильно округленного значения функции наиболее трудоемко из-за дилеммы составителя таблиц [20], поскольку большинство простых решений работает неточно на таких значениях аргумента.
- Наконец, используются такие точки, которые являются хорошими приближениями к числам, для которых значения функции точно известны, если подобные соотношения играют важную роль на практике.
  Например, для $cos(x) = 1/2$ при $x = \pm\pi/3 + 2n\pi$. Хотя ни одно из таких x не представимо, можно найти довольно много чисел с плавающей точкой, отстоящих от таких x на достаточно малое расстояние, чтобы при режиме округления к ближайшему корректный результат вычисления косинуса был равен в точности 1/2. На таких значениях аргумента хорошо проверяется адекватность и точность выполняемых вычислений.
  Важным частным случаем этого правила являются числа с плавающей точкой,

лежащие наиболее близко к нулям функции. Их использование при тестировании связано с проверкой правила сохранения знака.

Более подробно, основные шаги предлагаемого метода состоят в следующем.

1. Сначала сформируем *исходное множество* чисел как множество границ интервалов специфического поведения функции или чисел с плавающей точкой со специфической структурой.
   - Для этого добавим в исходное множество 0, -0, +∞, -∞, максимальное и минимальное представимые числа, минимальные и максимальные положительные и отрицательные денормализованные числа.
     Эти числа разбивают все числа с плавающей точкой на интервалы чисел со специфической структурой.
   - При определении требований к реализации рассматриваемой функции по описанной выше методике числа с плавающей точкой также разбиваются на ряд интервалов, в рамках каждого из которых действует специфический набор ограничений на возможные результаты функции.
     Это, например, интервалы монотонности и сохранения знака, интервалы, на которых функция имеет постоянное значение или ведет себя в соответствии с некоторой простой асимптотикой, интервалы, на которых она не определена.
     Кроме того, нужно рассмотреть прообразы выделенных на предыдущем шаге интервалов чисел определенной структуры. Прообразы положительных и отрицательных чисел дают области сохранения знака, поэтому новым здесь будет только разбиение этих областей на прообразы нормализованных и денормализованных значений. Каждый такой прообраз распадается на объединение конечного числа интервалов, которые также следует добавить в полученный набор.
     Для некоторых функций таких интервалов может оказаться так много, что их полный набор будет практически необозрим. Например, это так для тригонометрических функций — каждый интервал вида $[n\pi/2, (n+1)\pi/2]$ является пересечением интервалов монотонности и сохранения знака, и поэтому должен присутствовать в этом наборе. В таких ситуациях надо определить *правила отбора интервалов*, связанные со спецификой поведения функции и позволяющие сократить этот набор до вполне обозримого. В зависимости от выделенных на тестирование вычислительных ресурсов число интервалов в нем может колебаться от десятков до тысяч. Например, для тригонометрических функций в качестве таких правил можно выбрать следующие. Поведение функции на выбираемых интервалах должно быть в определенном смысле наиболее специфичным. Этого можно добиться, рассматривая наилучшие возможные приближения чисел с плавающей точкой к кратным π и $\pi/2$ — на интервалах, содержащих такие числа, значения тригонометрических функций ближе всего подходят к 0 и к экстремумам. Наилучшие приближения к кратным π и $\pi/2$ можно вычислить с помощью разложения π в цепную дробь (см. [57,58]), всего получается около 2000 таких чисел. Выбирая интервалы, содержащие такие значения, мы получаем искомое обозримое множество интервалов. Имеет смысл добавить в него и два-три десятка интервалов вида $[n\pi/2, (n+1)\pi/2]$, наиболее близких к 0.
2. Полученное исходное множество разбивает числа с плавающей точкой на набор интервалов.
   Чтобы определить используемые тестовые значения внутри этих интервалов,

выберем два целочисленных параметра n и k. Первый будет обозначать «частоту» выбираемых точек, второй — размер сплошных областей, покрываемых ими.

Каждый из интервалов разбивается на n более мелких, равных по количеству содержащихся в них чисел с плавающей точкой. При этом возникает (n+1) точка — (n-1) внутренняя и два конца интервала. Затем выберем все числа с плавающей точкой, лежащие в рассматриваемом интервале и отстоящие не более чем на k чисел с плавающей точкой от полученных точек. Получаемое множество точек назовем *пробным множеством*.

Можно использовать для каждого из исходных интервалов собственные значения n и k, покрывая некоторые области более плотно, чем другие.

3. К пробному множеству надо добавить точки, в которых для получения правильно округленного результата необходимо гораздо более точное вычисления значения рассматриваемой функции (см. выше о дилемме составителя таблиц и [20,38]).

   При этом стоит использовать не только значения, требующие максимально точных вычислений, а, например, те, которые требуют не менее чем $m > 0$ дополнительных бит мантиссы по сравнению с используемым типом чисел с плавающей точкой.

   Имеющиеся практические результаты и аргументы, основанные на теории вероятностей [19], показывают, что для чисел с плавающей точкой, имеющих K бит мантиссы и P бит экспоненты, такие значения существуют при $m < K + P$, а при б***о***льших m их практически нет. Не имеет смысл брать и слишком маленькое значение m, потому что соответствующих значений может оказаться слишком много. Кроме того, современные процессоры при вычислениях часто реально используют числа расширенной двойной точности, поэтому для чисел двойной точности значения для m, меньшего 12, часто не являются практически полезными тестами. Как показывают эксперименты [19], для чисел двойной точности хорошим значением m является 40 — для него обычно существует несколько тысяч тестовых значений аргумента.

4. Если рассматриваемая функция имеет ряд практически значимых свойств, связанных с ее точными значениями для некоторых непредставимых значений аргументов, следует вычислить числа с плавающей точкой, наилучшим образом приближающие такие значения аргументов.

   В первую очередь нужно рассмотреть наилучшие приближения к нулям функции.

   Ряд важных примеров дают тригонометрические функции: соотношения
   $sin(x) = 1/2$ при $x = \pi/6 + 2n\pi$ и $5\pi/6 + 2n\pi$,
   $sin(x) = -1/2$ при $x = -\pi/6 + 2n\pi$ и $-5\pi/6 + 2n\pi$,
   $cos(x) = 1/2$ при $x = \pm\pi/3 + 2n\pi$,
   $cos(x) = -1/2$ при $x = \pm 2\pi/3 + 2n\pi$,
   $tg(x) = ctg(x) = 1$ при $x = \pi/4 + n\pi$,
   $tg(x) = ctg(x) = -1$ при $x = -\pi/4 + n\pi$
   достаточно важны. Вычислить наилучшие приближения к таким значениям аргумента можно при помощи разложения π и $\pi/3$ в цепные дроби [58].

5. Иногда могут использоваться гипотезы о возможных ошибках в вычислении данной функции, связанных с конкретной структурой значения ее аргумента, например, для чисел, в мантиссе (или экспоненте) которых ровно одна единица или единицы чередуются с нулями. В таком случае нужно добавить ряд таких чисел в пробное множество. Эти гипотезы чаще всего можно сформулировать в виде набора шаблонов битового представления аргумента.

6. Наконец, в пробное множество нужно добавить несколько представителей NaN. Выбирать их можно по некоторым правилам, например, с минимальной и максимальной мантиссами, а также на основе различных шаблонов мантиссы.

Получаемое таким образом пробное множество и есть множество значений аргументов, которые предлагается использовать для тестирования рассматриваемой функции.

Управлять количеством получаемых тестовых данных, тщательностью тестирования и временем выполнения тестов в рамках описанного метода возможно при помощи выбора подходящих значений параметров n, k шага 2, m шага 3, а также точности приближения, используемой на шаге 4. Можно также варьировать правила отбора интервалов на шаге 1 и шаблоны, используемые на шаге 5.

4.3. Пример практического применения предложенных методов

В этом разделе рассмотрены результаты практического использования предложенных методов при определении требований и выборе тестовых данных для экспоненциальной функции $exp(x)$ для чисел двойной точности. Далее в этом разделе термин «экспонента» используется как синоним термина «экспоненциальная функция».

Проведенный в соответствии с описанным в разделе 4.1 методом анализ свойств экспоненциальной функции, а также выделение денормализованных и исключительных чисел разбивают всю числовую прямую на точки и интервалы, представленные в Таблице 2. Значения граничных точек указаны в Таблице 3. Выражения x++ и x- - для числа с плавающей точкой x обозначают, соответственно, следующее и предшествующее число с плавающей точкой.

Начало	**Конец**	**Пояснения**
$-\infty$		$exp(-\infty) = 0$
$-max$	x_1	Значение экспоненты при округлении к ближайшему числу с плавающей точкой равно 0.
x_1++	x_2	Округление значения экспоненты к $-\infty$ дает 0, а к ближайшему — не 0.
x_2++	x_3	Значения экспоненты денормализованы.
x_3++	x_4- -	Значения экспоненты положительны, нормализованы и меньше 1.
x_4	x_5- -	Значение экспоненты при округлении к $+\infty$ равно 1, а при других режимах — нет.
x_5	$-x_6$	Значение экспоненты при округлении к $+\infty$ или к ближайшему равно 1, значения аргумента отрицательны и нормализованы.
($-x_6$)++	$-min$	Значения аргумента отрицательны и денормализованы.
-0		$exp(-0) = 1$
0		$exp(0) = 1$
min	x_6- -	Значения аргумента положительны и денормализованы.
x_6	x_7- -	Значение экспоненты при округлении к $-\infty$ или к ближайшему равно 1, значения аргумента положительны и нормализованы.
x_7	x_8- -	Значение экспоненты при округлении к $-\infty$ равно 1, а к ближайшему — нет.
x_8	x_9	Значения экспоненты больше 1 и не превосходят max.
x_9++	max	Значение экспоненты равно $+\infty$ при округлении к ближайшему и к $+\infty$.
$+\infty$		$exp(+\infty) = +\infty$.
NaN		exp(NaN) = NaN

Таблица 2. Интервалы специфического поведения экспоненциальной функции.

В качестве тестовых данных были взяты границы полученных интервалов, и внутри каждого из интервалов был выбран ряд точек в соответствии с п. 2 описанного выше метода построения тестов.

Обозначение	**Значение**
min	$1.0_{16}\cdot 2^{-1074}$
max	$1.FFFFFFFFFFFFF_{16}\cdot 2^{1023}$
x_1	$-1.74910D52D3052_{16}\cdot 2^{9}$
x_2	$-1.74385446D71C4_{16}\cdot 2^{9}$
x_3	$-1.6232BDD7ABCD3_{16}\cdot 2^{9}$
x_4	$-1.0_{16}\cdot 2^{-53}$
x_5	$-1.0_{16}\cdot 2^{-54}$
x_6	$1.0_{16}\cdot 2^{-1022}$
x_7	$1.0_{16}\cdot 2^{-53}$
x_8	$1.0_{16}\cdot 2^{-52}$
x_9	$1.62E42FEFA39EF_{16}\cdot 2^{9}$

Таблица 3. Обозначения, использованные в Таблице 2.

В тестовые данные также вошло около 10000 значений, на которых проявляется дилемма составителя таблиц для экспоненты. Эти числа были вычислены во время широкомасштабного поиска таких значений для элементарных функций [20], проводимого в рамках проекта Arenaire.

При больших отрицательных значениях аргумента, где экспонента уже положительна, но ее значения — маленькие денормализованные числа, она растет очень медленно. Поэтому прообраз каждого такого числа представляет собой достаточно длинный интервал отрицательных чисел, экспонента от всех чисел на этом интервале при заданном режиме округления должна быть равна данному денормализованному числу. В тестовые данные было добавлено около десятка границ таких интервалов и для каждого интервала — одно число, лежащее внутри него.

Далее, в тестовые данные были добавлены несколько чисел с плавающей точкой, обладающих тем свойством, что значение экспоненты при переходе между соседними такими числами изменяется ровно на последний бит мантиссы.

В тестовые данные также были добавлены числа 1.0 и -1.0, соседние с ними числа с плавающей точкой и, наконец, 5 различных представлений NaN.

В итоге был получен тестовый набор из 15804 тестовых значений, из которых 2604 предназначены для тестирования экспоненты при режиме окружения к ближайшему и по 4400 — для остальных трех режимов. Результаты выполнения полученных тестов на различных платформах представлены в Таблицах 4 и 5.

В Таблице 4 приведены результаты тестирования для платформ A-L, поддерживающих установление различных режимов округления. Платформы M-Q поддерживают только режим округления к ближайшему, поэтому в Таблице 5 их результаты сведены с результатами для платформ A-L, полученными на тестах, нацеленных на выполнение в этом режиме. Ниже описан проведенный анализ ошибок, а также дана информация обо всех использованных платформах и описание обнаруженных ошибок.

Анализ ошибок.

Для удобства анализа ошибок числа с плавающей точкой были разбиты на следующие классы: положительные нормализованные числа, отрицательные нормализованные числа, положительные денормализованные, отрицательные денормализованные, 0, -0, $+\infty$, $-\infty$, NaN.

Платформа	**Число тестов**	**Обнаруженные ошибки и неточности**					
		Неверный класс результата			Вычислительные ошибки		
		Серьезные	Малые		Серьезные	Малые	
			Число	Сумма расстояний		Число	Сумма расстояний
A	15607	8	0	0	0	6080	6080
B	15607	0	48	48	0	6080	6080
C	15607	0	48	48	0	6197	6197
D	15607	32	16	16	32	7226	7226
E	15607	0	42	42	0	7373	7373
F	15607	301	91	143869	5572	1739	$\sim 3 \cdot 10^8$
G	15607	230	21	22	5937	1805	$\sim 5 \cdot 10^8$
H	15607	0	54	54	0	8031	8336
I	15607	0	54	54	0	7682	7685
J	15607	0	54	54	0	7500	8146
K	15607	8	54	54	0	9373	302308
L	15607	0	54	54	0	10598	77920

Таблица 4. Результаты тестирования реализаций экспоненты во всех режимах округления.

Можно считать, что изменение класса результата является в общем случае более серьезной неточностью, чем неверный результат в рамках верного класса. Второй вид ошибок в Таблицах 4 и 5 назван вычислительными ошибками. Кроме того, оба типа ошибок — изменяющие класс результата и не изменяющие его — были разделены на серьезные и малые, в зависимости от расстояния между корректным результатом и фактическим результатом вычисления функции (в последних битах мантиссы).

Платформа	**Число тестов**	**Обнаруженные ошибки и неточности**					
		Неверный класс результата			Вычислительные ошибки		
		Серьезные	Малые		Серьезные	Малые	
			Число	Сумма расстояний		Число	Сумма расстояний
A	2554	2	0	0	0	1128	1128
B	2554	0	0	0	0	1128	1128
C	2554	0	0	0	0	1130	1130
D	2554	16	0	0	0	1152	1152
E	2554	0	0	0	0	1128	1128
F	2554	0	0	0	0	0	0
G	2554	0	0	0	0	0	0
H	2554	0	19	19	0	1267	1267
I	2554	0	5	5	0	1197	1197
J	2554	0	3	3	0	1296	1296
K	2554	2	18	18	0	1825	73709
L	2554	0	19	19	0	1788	9389
M	2554	2	1	214	0	1609	37316
N	2554	0	1	214	0	1609	37316
O	2554	0	0	0	0	1153	1153
P	2554	0	0	0	0	1117	1117
Q	2554	0	0	0	0	1147	1147

Таблица 5. Результаты тестирования реализаций экспоненты в режиме округления к ближайшему.

Ошибка изменения класса серьезна, если корректный или фактический результат равен NaN и они отличаются, а также если корректный и фактический результаты лежат не в

соседних классах или в соседних, но отстоят друг от друга более, чем на 2^{30} чисел с плавающей точкой. Соседними классами считаются такие, что между некоторыми значениями, принадлежащими им, лежит не более 2^{30} чисел с плавающей точкой.

Вычислительная ошибка серьезна, если корректный и фактический результаты отстоят друг от друга более чем на 2^{30} чисел с плавающей точкой.

Тип ошибки	**Ошибка или неточность**	**Платформы**
Серьезное изменение класса результата	1. $exp(\pm\infty)$ возвращает NaN	A, K, M
	2. $exp(x > x_9)$ возвращает максимальное число однократной точности ($3.4028234663852885 \cdot 10^{38}$) при округлении к ближайшему и к $+\infty$	D
	3. Неверные результаты для многих аргументов с существенным изменением их класса при режимах округления к $+\infty$, к 0 и к $-\infty$	F, G
Малое изменение класса результата	4. $exp(x > x_9)$ возвращает $+\infty$ при округлении к 0 и к $-\infty$	B, C, E (для больших чисел), I (для больших чисел), J
	5. $exp(x > x_9)$ возвращает максимальное число двойной точности при округлении к ближайшему и к $+\infty$	H, I (иногда), K, L
	6. $exp(x \le x_1)$ возвращает 0 при округлении к $+\infty$	B, C, D, E (иногда), F, G, H, I, J, K, L
	7. $exp(x)$ при $x_1 < x \le x_2$ возвращает 0 при округлении к ближайшему	H, I, J, K (иногда), L
	8. $exp(x)$ при $x_1 < x \le x_2$ иногда возвращает не 0 при округлении к 0 и $-\infty$	K
	9. $exp(x_9)$ возвращает $+\infty$	M, N
	3′. Неверные результаты для многих аргументов с переходом в соседние классы при режимах округления к $+\infty$, к 0 и к $-\infty$ (см. 3)	F, G
Серьезная вычислительная	2′. $exp(x > x_9)$ возвращает максимальное число однократной точности ($3.4028234663852885 \cdot 10^{38}$) при округлении к 0 и к $-\infty$	D
	3″. Существенно неверные результаты для многих аргументов без изменения класса при режимах округления к $+\infty$, к 0 и к $-\infty$	F, G
Малая вычислительная		K(680), M(333), N(333), L(17), J(2^{646}), H(2^{306}), I(2^3), A(1), B(1), C(1), D(1), E(1), O(1), P(1), Q(1)

Таблица 6. Обнаруженные разновидности ошибок и неточностей на разных платформах.

Для малых ошибок обоих типов подсчитывалась сумма расстояний между фактическими и корректными результатами в интервалах между соседними числами с

плавающей точкой. Это позволяет оценить общую структуру таких ошибок, в частности, насколько много среди них ошибок на более чем 1 бит.

Результаты тестирования.

Ниже описываются использованные платформы и приводится дополнительная информация о результатах тестирования. Описание различных видов ошибок, обнаруженных на различных платформах, дано в Таблице 6. Всего было найдено 9 видов ошибок и неточностей (ошибки видов 2′, 3′, 3″ имеют ту же природу, что и ошибки видов 2 и 3).

В правой нижней ячейке Таблицы 6 собраны результаты, описывающие найденные вычислительные неточности. После символа платформы в скобках указано максимальное расстояние между фактическим и корректным результатами в интервалах между соседними числами с плавающей точкой двойной точности. Для тех случаев, когда такое расстояние не превосходит двух в виде верхнего индекса показано общее число ситуаций, в которых оно равно 2. Для платформ F и G такие результаты не приводятся, поскольку обнаруженные на них неточности слишком велики.

- Платформа A — это POSIX-совместимая операционная система (ОС) специализированного назначения на процессоре Intel Pentium II.
  Платформа B. Под этим заголовком даны результаты выполнения тестов для разных версий — от 2.1.3 до 2.3.2 — библиотеки glibc в рамках ОС RedHat Linux разных версий на различных процессах Intel — Pentium II, Pentium 4, Xeon. Все эти результаты оказались полностью идентичными, т.е. совпадают не только количество ошибок, но и значения аргументов, на которых они проявляются, и вычисленные значения функции.
  Платформа C. Под этим заголовком даны результаты выполнения тестов для разных версий glibc (2.3.2, 2.3.5) в рамках ОС RedHat Linux разных версий на процессорах AMD Athlon XP. Эти результаты также оказались идентичными.
  Платформа D — ОС Solaris 10 (SunOS 5.10) на процессоре Sun UltraSpark IIIi.
  Платформа E — ОС FreeBSD 5.4 на процессоре Intel Pentium 4.
  В целом, результаты тестов на всех этих платформах достаточно похожи. Серьезные ошибки есть только на платформах A и D, малые ошибки изменения класса результата на платформах B, C, D, E практически идентичны (см. Таблицу 6). Вычислительные неточности малы и не превосходят 1 бит.
  На платформах A и B вычислительные неточности абсолютно идентичны, что, по-видимому, вызвано использованием одной и той же процедуры для вычисления экспоненты, рекомендованной для процессоров Intel еще во времена i386 — $exp(x) = 2^{(x\ log_2 e)}$. Эта же процедура приводит к неверным результатам вычисления экспоненты для значений $+\infty$ и $-\infty$ (ошибка вида 1) на платформе A. В glibc эти случаи обрабатываются отдельно (см. комментарии к коду экспоненты в glibc [59]).
  На платформе D была обнаружена характерная только для нее ошибка вида 2 (а также 2′), из-за которой для больших положительных чисел x $exp(x) < x$.

- Платформа F — это ОС SUSE 10.0 (glibc 2.3.5) на процессоре IBM PowerPC 750.
  Платформа G — ОС SUSE 10.0 (glibc 2.2.5) на процессоре IBM s390.
  Результаты тестирования на этих платформах наиболее удивительны. В режиме округления к ближайшему выдаются абсолютно корректные результаты, ни одной ошибки, даже малейшей вычислительной погрешности. Это было проверено и на более широком множестве тестов, содержащем около 38000 значений. При других же режимах округления могут возникать произвольно большие погрешности. Это, по-видимому, связано со специфической реализацией экспоненты процессорах IBM, работающей полностью корректно

при округлении к ближайшему и плохо отлаженной при других режимах.
На платформе F наиболее серьезные ошибки происходят в режимах округления к 0 и к -∞, а в режиме округления к +∞ в основном возникают значительные вычислительные погрешности, не меняющие, однако, класса результата. На платформе G все наоборот.
Например, на платформе F при режимах округления к 0 и к -∞ многие значения экспоненты отрицательны, причем, встречаются и довольно большие по абсолютной величине, например *exp*(1.8440884407690585) при округлении к -∞ считается равным $-1.3766469207992498 \cdot 10^{6}$. Другие яркие примеры: результат вычисления на платформе F *exp*(-0.75067340262097160) при округлении к -∞ равен $1.1268633387184180 \cdot 10^{15}$, а *exp*(1) при округлении к -∞ возвращает 4.0092154790944878 (!).

- Платформа H — это ОС Microsoft Windows XP (библиотека Microsoft Visual Studio 6, сборка в режиме debug) на процессоре Intel Pentium M.
  Платформа I — ОС Microsoft Windows 2000 (библиотека Microsoft Visual Studio.NET 2003, debug-сборка) на процессоре Intel Pentium 4.
  Платформа J — ОС Microsoft Windows XP-64 (64-битная библиотека Microsoft Visual Studio.NET 2005) на процессоре AMD Athlon-64 X2.
  Платформа K. Здесь даны результаты выполнения тестов под разными версиями Microsoft Windows (XP и 2000) на процессорах Intel (Pentium 4, Pentium M) для библиотек Microsoft Visual Studio версий 6 и 7 (.NET 2003) при использовании режима сборки release. Результаты оказались полностью идентичными.
  Платформа L — под этим заголовком даны результаты выполнения тестов под Microsoft Windows 2000 на процессоре Intel Pentium 4 для 32-битных библиотек Microsoft Visual Studio.NET 2005 и Intel C++ Compiler 9.1. Их результаты идентичны.
  При сборке проекта в Microsoft Visual Studio можно выбрать тип сборки: отладочная (debug) или продуктовая (release). В версиях 6 и 7 (она же .NET 2003) при этом используются различные библиотеки математических функций, демонстрирующие разные результаты вычисления одних и тех же функций на одних и тех же данных — сборки вида release считают с б***о***льшим количеством ошибок. В более тонких настройках сборки в этих версиях Visual Studio имеется флаг, регулирующий обеспечение корректности вычислений с плавающей точкой (Floating-Point Consistency), выключение которого по умолчанию при продуктовой сборке как раз и приводит к такому эффекту.
  В целом результаты этих платформ близки и обнаруживают две тенденции — результаты отладочных библиотек Visual Studio улучшаются при переходе от версии 6 к версии 7, а затем еще немного — для 64-битной версии 8, хотя количество вычислительных погрешностей в 2 бита в последнем случае увеличивается.
  Продуктовые же библиотеки в версиях 6 и 7 одинаковы, содержат серьезную ошибку вида 1 и довольно много неточностей. Встречаются ошибки до 680 бит, что означает неправильность 5-ти последних десятичных цифр. Похоже, что 32-битные библиотеки версии 8 были построены на их основе, причем часть ошибок была исправлена и, хотя количество вычислительных погрешностей немного увеличилось, они стали заметно меньше.
  Хотя количество ошибок неверного класса результата на этих платформах одинаково, они различаются, что видно из Таблиц 5 и 6.

- Платформа M — это реализация Java версии 1.1.8 от Sun под Microsoft Windows 2000.
  Платформа N — Microsoft .NET 1.1 под Microsoft Windows 2000.

Платформа O. Под этим заголовком даны результаты тестирования платформы Java различных версий, начиная с 1.2 (1.2, 1.4, 1.5), на разных ОС (Microsoft Windows 2000, SUSE 10.0, RedHat 9.3) и от разных разработчиков (Sun, IBM, Bea). Все эти результаты полностью идентичны.
Платформа P и платформа Q — это Microsoft .NET 2.0, 32-битная версия под Windows 2000 и, соответственно, 64-битная версия .NET 2.0 под Windows XP. Java и .NET выполняют вычисления только в режиме округления к ближайшему, поэтому для всех этих платформ выполнялись тесты только для этого режима. Только на платформе M есть серьезная ошибка типа 1, которая была исправлена в следующей версии Java.
На платформах M и N есть одинаковая ошибка типа 9 и все вычислительные неточности на них абсолютно идентичны. По-видимому, Microsoft разрабатывала первую версию математических библиотек .NET, во многом ориентируясь на Java версии 1.
Идентичность результатов Java, начиная с версии 1.2, объясняется более жестким стандартом на поведения реализаций математических функций, введенным Sun в этой версии языка.
На всех остальных платформах ошибки только вычислительные, все на 1 бит, и их не так уж много.

Полученные результаты показывают, что все серьезные ошибки (кроме совсем странных результатов платформ F и G) были обнаружены на тестах, построенных на основе анализа интервалов однородного поведения экспоненциальной функции. Добавленные же 10000 тестов на точность вычислений позволяют оценить общий вид вычислительных погрешностей и их распределение в целом. Таким образом, эти два компонента тестового набора хорошо дополняют друг друга, делая возможным тестирование всех аспектов поведения реализаций математических функций.

5. Заключение

Разнообразие приложений математического моделирования, растущая потребность в надежности и точности их результатов, необходимость получать одни и те же результаты на разных платформах делают весьма актуальной задачу стандартизации вычислений математических функций. Несмотря на наличие серьезных исследований в этой области, например, работ [7,8,18-20,38,49-52], ведущихся в рамках проекта Arenaire [6], а также недавнее принятие стандарта ISO 10967 [3-5], определяющего независимые от используемых языков программирования и платформ требования к реализациям математических функций, многие вопросы остаются нерешенными. Во-первых, попытки сформулировать систематическую методику определения требований к математическим функциям (например, [7,8]), упускают из вида некоторые детали (односторонние пределы в 0, окрестности полюсов, бесконечностей и точных значений, асимптотики, симметрии, помимо четности и нечетности). Во-вторых, нет работ по методам построения тестов, так же систематически проверяющих все аспекты поведения реализаций таких функций.

В рамках данной работы предложены решения для обеих указанных задач. Представленный здесь метод определения требований к реализациям математических функций основан на уже сформулированных во многих работах идеях и фактически просто расширяет подход стандарта IEEE 745 к определению арифметических операций на остальные математические функции. Единственным вкладом автора здесь можно считать систематизацию всех этих идей.

Описанный в разделе 4.2 метод построения тестов для проверки выполнения сформулированных требований также использует результаты некоторых исследований по точному вычислению математических функций [20] и понятную всякому

тестировщику идею разбиения области определения функции на интервалы, на которых она ведет себя более-менее однородно. Однако автору неизвестны работы, где бы излагались подходы к построению тестов для математических функций, охватывающие все аспекты, отраженные в представленном здесь методе.

Результаты применения представленных методов для достаточно простой экспоненциальной функции, описанные в разделе 4.3, показывают, что до воплощения на практике четких стандартов на реализации математических функций еще достаточно далеко. Если для режима округления к ближайшему большинство реализаций работает достаточно хорошо (есть только небольшие вычислительные неточности), то с другими режимами округления связано довольно много проблем. Практическая важность поддержки других режимов объясняется возможностью реализации на их основе систем достоверных интервальных вычислений, которые оказываю серьезную помощь при математическом моделировании неустойчивых задач.

Вместе с тем, предлагаемые в данной работе методы как раз и позволяют постепенно преодолеть эти проблемы, переводя решение отдельных задач в области определения стандартов и разработки тестов для реализаций математических функций в чисто практическую плоскость.

Литература

[1] IEEE 754-1985. *IEEE Standard for Binary Floating-Point Arithmetic*. NY: IEEE, 1985.

[2] IEC 60559:1989. *Binary Floating-Point Arithmetic for Microprocessor Systems*. Geneve: ISO, 1989.

[3] ISO/IEC 10967-1:1994. *Information Technology — Language Independent Arithmetic — Part 1: Integer and Floating Point Arithmetic*. Geneve: ISO, 1994.

[4] ISO/IEC 10967-2:2002. *Information Technology — Language Independent Arithmetic — Part 2: Elementary Numerical Functions*. Geneve: ISO, 2002.

[5] ISO/IEC 10967-3. *Information Technology — Language Independent Arithmetic — Part 3: Complex Integer and Floating Arithmetic and Complex Elementary Numerical Functions*. Draft. Geneve: ISO, 2002.

[6] http://www.inria.fr/recherche/equipes/arenaire.en.html

[7] G. Hanrot, V. Lefevre, J.-M. Muller, N. Revol, and P. Zimmermann. *Some Notes for a Proposal for Elementary Function Implementation in Floating-Point Arithmetic*. Proc. of Workshop IEEE 754R and Arithmetic Standardization, in ARITH-15, June 2001.

[8] D. Defour, G. Hanrot, V. Lefevre, J.-M. Muller, N. Revol, and P. Zimmermann. *Proposal for a standardization of mathematical function implementation in floating-point arithmetic*. Numerical Algorithms, 37(1–4):367–375, December 2004.

[9] IEEE 854-1987. *IEEE Standard for Radix-Independent Floating-Point Arithmetic*. NY: IEEE, 1987.

[10] D. Goldberg. *What Every Computer Scientist Should Know about Floating-Point Arithmetic*. ACM Computing Surveys, 23(1):5-48, 1991.

[11] P. Sterbenz. *Floating-Point Computation.* New Jersey: Prentice-Hall, Englewood Cliffs, 1974.

[12] J. Camlet, V. Lefevre. *Toward the Integration of Numerical Computations into the OMSCS Framework.* In V. G. Ganzha, E. V. Vorozhtsov, eds. Proc. of the 7-th International Workshop on Computer Algebra in Scientific Computing (CASC 2004), pp. 71–79, Saint-Petersburg, Russia, July 2004.

[13] http://www.fas.org/spp/starwars/gao/im92026.htm

[14] http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html

[15] ISO/IEC 9899:1999. *Programming Languages — C*. Geneve: ISO, 1999.

[16] IEEE 1003.1-2004. *Information Technology — Portable Operating System Interface (POSIX)*. NY: IEEE, 2004.
[17] http://www.opengroup.org/onlinepubs/009695399/functions/sin.html
[18] V. Lefevre, J.-M. Muller, and A. Tisserand. *Toward Correctly Rounded Transcendentals*. IEEE Transactions on Computers, 47(11):1235–1243,November 1998.
[19] V. Lefevre, J.-M. Muller, and A. Tisserand. *The Table Maker's Dilemma*. INRIA Research Report 98-12, 1998.
[20] V. Lefevre, J.-M. Muller. *Worst Cases for Correct Rounding of the Elementary Functions in Double Precision*. Proc. of 15-th IEEE Symposium on Computer Arithmetic, Vail, Colorado, USA, June 2001.
[21] С. Ленг. Агебра. М: Мир, 1968.
[22] M. Abramowitz and I. A. Stegun, eds. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1965.
[23] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Second edition. Birkhauser, Boston, 2006.
[24] http://www.math.utah.edu/~beebe/software/ieee/
[25] http://people.redhat.com/drepper/libm/
[26] W. Kahan. *What Can You Learn about Floating-Point Arithmetic in One Hour*? http://http.cs.berkeley.edu/~wkahan/ieee754status, 1996.
[27] N. L. Schryer. *A Test of Computer's Floating-Point Arithmetic Unit*. Computer Science Technical Report 89, AT&T Bell Labs, 1981.
[28] B. Verdonk, A. Cuyt, and D. Verschaeren. *A Precision- and Range-Independent Tool for Testing Floating-Point Arithmetic I: Basic Operations, Square Root and Remainder*. ACM TOMS 27(1):92–118, 2001.
[29] B. Verdonk, A. Cuyt, and D. Verschaeren. *A Precision- and Range-Independent Tool for Testing Floating-Point Arithmetic II: Conversions*. ACM TOMS 27(1):119–140, 2001.
[30] http://www.cant.ua.ac.be/ieeecc754.html
[31] R. Karpinski. *PARANOIA: A Floating-Point Benchmark*. Byte Magazine 10, 2 (Feb.), pp. 223–235, 1985.
[32] http://www.netlib.org/paranoia/
[33] A. Ziv, M. Aharoni, and S. Asaf. *Solving Range Constraints for Binary Floating-Point Instructions*. Proc. of 16-th IEEE Symposium on Computer Arithmetic (ARITH-16'03), pp. 158–163, 2003.
[34] M. Aharoni, S. Asaf, L. Fournier, A. Koifman, and R. Nagel. *FPgen — A Test Generation Framework for Datapath Floating-Point Verification*. Proc. IEEE International High Level Design Validation and Test Workshop (HLDVT'03), pp. 17–22, 2003.
[35] http://www.math.utah.edu/pub/elefunt/
[36] http://www.netlib.org/fp/ucbtest.tgz
[37] Z. A. Liu. *Berkeley Elementary Function Test Suite*. M.S. thesis, Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, December 1987.
[38] D. Stehle, V. Lefevre, P. Zimmermann. *Searching Worst Cases of a One-Variable Function Using Lattice Reduction*. IEEE Transactions on Computers, 54(3):340–346, March 2005.
[39] http://www.loria.fr/~zimmerman/mpcheck/
[40] http://www.maplesoft.com/
[41] http://www.wolfram.com/products/mathematica/index.html
[42] http://www.mathworks.com/products/matlab/
[43] A. Ziv. *Fast Evaluation of Elementary Mathematical Functions with Correctly Rounded Last Bit*. ACM Transactions on Mathematical Software, 17(3):410–423, September 1991.

[44] *IBM Accurate Portable MathLib*
http://rpmfind.net/linux/rpm2html/search.php?query=libultim.so.2
[45] http://www.mpfr.org/
[46] http://www.sun.com/download/products.xml?id=41797765
[47] http://www.ens-lyon.fr/LIP/Arenaire/Ware/SCSLib/
[48] http://lipforge.ens-lyon.fr/projects/crlibm/
[49] F. de Dinechin, A. Ershov, and N. Gast. *Towards the post-ultimate libm.* Proc. of 17-th Symposium on Computer Arithmetic. IEEE Computer Society Press, June 2005.
[50] D. Defour, F. de Dinechin, J.-M. Muller. *Correctly Rounded Exponential Function in Double Precision Arithmetic*. INRIA Research report RR-2001-26, July 2001.
[51] F. de Dinechin, C. Lauter, J.-M. Muller. *Fast and Correctly Rounded Logarithms in Double-Precision*. INRIA Research report RR-2005-37, September 2005.
[52] S. Chevillard, N. Revol. *Computation of the Error Functions erf and erfc in Arbitrary Precision with Correct Rounding*. Proc. of 17-th IMACS Conf. on Scientific Computation, Applied Math. and Simulation, Paris, France, July 2005.
[53] W. Kramer. *Multiple-Precision Computations with Result Verification*. In E. Adams, U. Kulisch, eds. Scientific Computing with Automatic Result Verification, pp. 325–356, Academic Press, 1993.
[54] M. J. Schulte, E. E. Swartzlander. *Software and Hardware Techniques for Accurate, Self-Validating Arithmetic*. Applications of Interval Computations, pp. 381–404, 1996.
[55] N. Revol, F. Rouillier. *Motivations for an Aarbitrary Precision Interval Arithmetic and the MPFI Library*. Reliable Computing, 11(4):275–290, 2005.
[56] *MPFI Library* http://perso.ens-lyon.fr/nathalie.revol/mpfi_toc.html
[57] W. Kahan. *Minimizing q*m − n*. 1983. Неопубликованные заметки, доступны по http://http.cs.berkeley.edu/~wkahan/testpi/nearpi.c.
[58] В. В. Кулямин. *Формальные подходы к тестированию математических функций.* Труды ИСП РАН, т. 10, 2006.
[59] http://sourceware.org/cgi-bin/cvsweb.cgi/libc/sysdeps/i386/fpu/e_expl.c?cvsroot=glibc