Формальные подходы к тестированию математических функций

В. В. Кулямин

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

Аннотация

Данная статья рассматривает вопросы проверки корректности вычисления математических функций на числах с плавающей точкой, формат которых определяется стандартом IEEE 754. В ней описывается метод разработки тестов для реализаций таких функций, основанный на формальных спецификациях их поведения. Предлагаемый метод основан на технологии разработки тестов UniTESK и двух дополнительных методиках: методике формирования строгих требований к реализации математической функции и методике построения набора тестовых данных для ее тестирования. Описанные методики опираются на специфические свойства представления чисел с плавающей точкой и особенности поведения самой тестируемой функции.

1. Введение

На современном уровне развития общества большое значение в его технологической инфраструктуре играют информационные системы. На них перекладывается все больше функций по хранению, обработке и адекватному представлению информации, являющейся одним из базовых элементов сегодняшней экономической и культурной жизни человечества.

Удивительно, но до сих пор нет эффективных технологий разработки информационных систем, работающих *правильно*, т. е. так, как того ожидают их пользователи. Это обусловлено двумя причинами. Во-первых, само понятие правильности такой системы неуловимо и неформально, поскольку оно тесно связано с набором нечетких, противоречивых и меняющихся со временем ожиданий и потребностей людей, общающихся с ней. Во-вторых, известные недостатки человеческой природы не позволяют разработчикам программ писать их без ошибок, даже когда они четко понимают поставленную задачу и имеют математически выверенное ее решение.

Итеративная разработка с использованием тестирования является единственным практически работающим способом создания достаточно качественного программного обеспечения (ПО), хотя и не корректного формально, но достаточно хорошо удовлетворяющего потребности пользователей. В ходе такой разработки проектирование и написание кода чередуется с проверками работоспособности и правильности результатов работы проектируемых и разрабатываемых компонентов.

Правильность их проверяется как соответствие этих результатов ожиданиям пользователей в соответствующей ситуации.

В то же время, область использования сложного ПО в человеческой деятельности все расширяется, и во многих случаях решаемые им задачи таковы, что даже по косвенным признакам уже нелегко судить о правильности полученных с его помощью результатов. Можно привести следующие примеры таких задач.

- Моделирование космогонических процессов: развития звезд, планетных систем и галактик. Моделирование космических катастроф, последствия которых могут угрожать существованию человечества.
- Моделирование физических процессов в экстремальных условиях: движения со сверхвысокими скоростями в вязких средах, поведения элементарных частиц, поведения плазмы в термоядерном реакторе, поведения материалов в условиях сверхнизких или сверхвысоких температур и давлений, в сверхсильных магнитных полях и т.п. Все эти задачи требуется решать для обеспечения технологического прогресса, в частности, для создания и доведения до практической применимости двигателей и энергетических установок, работающих на новых принципах; создания новых материалов с заданными свойствами и пр.
- Моделирование построения и работы наносистем, необходимое для успешного развития нанотехнологий и создания механизмов, решающих важные для человека задачи при помощи манипулирования отдельными молекулами и атомами.
- Моделирование биохимических процессов, связанных с функционированием различных белков, обменом веществ между живой клеткой и окружающей средой, модификацией генетической информации, активизацией и дезактивацией генов. Понимание их механизмов поможет создать более эффективные и безопасные лекарства против многих болезней, в том числе, плохо поддающихся лечению современными средствами, получить новые, более стойкие и полезные сорта и виды используемых в сельском хозяйстве растений и животных, а в перспективе — продлить жизнь каждого человека до ее биологических пределов.
- Моделирование сложных экологических, климатических, экономических и социальных систем. Оно позволит более глубоко понять законы их развития, выработать механизмы разрешения и предотвращения кризисов и катастроф, приносящих серьезный ущерб отдельным людям и обществу в целом, а также поможет человечеству более ответственно относиться к среде его существования и заложить основы для гармоничного и стабильного развития в дальнейшем.

Все эти задачи решаются при помощи использования тех или иных методов математического моделирования и дают результаты, проверка правильности которых связана с большими затратами ресурсов и усилий многих людей, а иногда и вообще невозможна или создает очень серьезные угрозы для отдельных людей или общества в целом.

Тем не менее, есть возможность повысить надежность и правильность работы программных систем, выполняющих моделирование такого рода. Это может быть сделано за счет формальной проверки корректности работы библиотечных математических функций, на которые во многом это ПО опирается. Уверенность в надежности фундамента, на котором построены такие системы, позволит разрабатывать их более качественно и с меньшими усилиями, сосредоточившись на поиске и исправлении ошибок в других компонентах.

В данной работе изучаются проблемы проверки корректности реализаций математических функций, работающих с числами с плавающей точкой, анализируются имеющиеся достижения в этой области и предлагается метод разработки тестов для таких функций на основе формальных спецификаций их поведения. Полученный метод базируется на технологии UniTESK [1-3], дополняя ее методиками формирования точных требований к реализациям математических функций и выбора тестовых данных для их тестирования.

Сперва рассмотрим проблемы, с которыми сталкиваются исследования в этой области.

2. *Проблемы корректного вычисления функций*

Основные проблемы корректного вычисления математических функций связаны с дискретностью представления действительных чисел в компьютерах. Чтобы иметь возможность эффективно выполнять операции с действительными числами, они представлены в виде так называемых *чисел с плавающей точкой*, формат и правила действий над которыми определены в стандартах IEEE 754 [4] (он же — IEC 60559 [5]) и IEEE 854 [6].

IEEE 754 определяет представление двоичных чисел с плавающей точкой, IEEE 854 обобщает его, определяя и десятичные числа с плавающей точкой. Однако, поскольку в большинстве случае на практике используется двоичное представление чисел, мы будем рассматривать только его.

2.1. Числа с плавающей точкой

Двоичное число с плавающей точкой имеет следующую структуру [4,6,7].

- Число представлено в виде набора из n бит, из которых первый бит является *знаковым битом числа*, следующие k бит отданы под представление его *экспоненты*, а оставшиеся $(n-k-1)$ бит представляют его *мантиссу*.

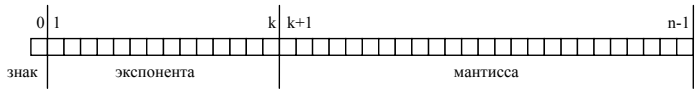

Рисунок 1. Битовое представление чисел с плавающей точкой.

- 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, 0 для положительных чисел, и 1 для отрицательных;
  - если $E > 0$, то $e = E - 2^{(k-1)} + 1$;
    иначе, если $E = 0$, $e = -2^{(k-1)} + 2$;
    число $(2^{(k-1)} - 1)$ называется *смещением экспоненты (bias)*;
  - если $0 < E < 2^k - 1$, то 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 имеет двоичное представление $0.M$,
    такие числа (с нулевой экспонентой) называются *денормализованными*.
- Максимальное возможное значение экспоненты $E = 2^k - 1$ зарезервировано для представления положительной $+\infty$ и отрицательной $-\infty$ бесконечностей и специального значения NaN (not-a-number), которое возникает, если результат выполняемых действий нельзя корректно представить ни обычным числом, ни бесконечностью, например, 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.
  Все такие числа называются *исключительными*.
- 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 $128 \ge n \ge 80$ и $k \ge 15$. В процессорах Intel 32-битной архитектуры используются значения $n = 80$ и $k = 15$. При этом для мантиссы используется 64 бита и смещение экспоненты равно 16383.

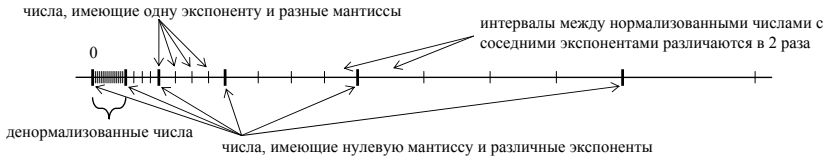

Рисунок 2. Примерная картина распределения чисел с плавающей точкой.

Примеры чисел с плавающей точкой.

- Числа однократной точности
  - $0 = (-1)^0 \cdot 2^{0-126} \cdot 0.0_2$ имеет представление
    0 00000000 00000000000000000000000
  - $1 = (-1)^0 \cdot 2^{127-127} \cdot 1.0_2$ имеет представление
    0 01111111 00000000000000000000000
  - $-17_{10} = (-1)^1 \cdot 2^{131-127} \cdot 1.0001_2$ имеет представление
    1 10000011 00010000000000000000000
  - $0.75_{10} = (-1)^0 \cdot 2^{126-127} \cdot 1.1_2$ имеет представление
    0 01111110 10000000000000000000000
  - самое маленькое положительное число, представимое в таком виде (денормализованное)
    $2^{-149} = (-1)^0 \cdot 2^{0-126} \cdot 0.00000000000000000000001_2$
    0 00000000 00000000000000000000001
  - самое маленькое положительное число, представимое в нормальном виде
    $2^{-126} = (-1)^0 \cdot 2^{1-127} \cdot 1.0_2$
    0 00000001 00000000000000000000000
  - самое большое положительное число, представимое в таком виде
    $2^{104} \cdot (2^{24}-1) = (-1)^0 \cdot 2^{254-127} \cdot 1.11111111111111111111111_2$
    0 11111110 11111111111111111111111
  - $+\infty$ имеет представление
    0 11111111 00000000000000000000000
  - следующее за 1 число, представимое в таком виде
    $1+2^{-23} = (-1)^0 \cdot 2^{127-127} \cdot 1.00000000000000000000001_2$
    0 01111111 00000000000000000000001

  - предшествующее 1 число, представимое в таком виде
    $1-2^{-24} = 2^{-24} \cdot (2^{24}-1) = (-1)^0 \cdot 2^{126-127} \cdot 1.11111111111111111111111_2$
    0 01111110 11111111111111111111111

- Числа двойной точности
  - $0 = (-1)^0 \cdot 2^{0-1022} \cdot 0.0_2$ имеет представление
    0 00000000000 0000000000000000000000000000000000000000000000000000
  - $1 = (-1)^0 \cdot 2^{1023-1023} \cdot 1.0_2$ имеет представление
    0 01111111111 0000000000000000000000000000000000000000000000000000
  - $-17_{10} = (-1)^1 \cdot 2^{1027-1023} \cdot 1.0001_2$ имеет представление
    1 10000000011 0001000000000000000000000000000000000000000000000000
  - $0.75_{10} = (-1)^0 \cdot 2^{1022-1023} \cdot 1.1_2$ имеет представление
    0 01111111110 1000000000000000000000000000000000000000000000000000
  - самое маленькое положительное число, представимое в таком виде (денормализованное)
    $2^{-1074} = (-1)^0 \cdot 2^{0-1022} \cdot 2^{-52}$
    0 00000000000 0000000000000000000000000000000000000000000000000001
  - самое маленькое положительное число, представимое в нормальном виде
    $2^{-1022} = (-1)^0 \cdot 2^{1-1023} \cdot 1.0_2$
    0 00000000001 0000000000000000000000000000000000000000000000000000
  - самое большое положительное число, представимое в таком виде
    $2^{971} \cdot (2^{53}-1) = (-1)^0 \cdot 2^{2046-1023} \cdot (2^{53}-1) \cdot 2^{-52}$
    0 11111111110 1111111111111111111111111111111111111111111111111111
  - $+\infty$ имеет представление
    0 11111111111 0000000000000000000000000000000000000000000000000000
  - следующее за 1 число, представимое в таком виде
    $1+2^{-52} = (-1)^0 \cdot 2^{127-127} \cdot (1+2^{-52})$
    0 01111111111 0000000000000000000000000000000000000000000000000001
  - предшествующее 1 число, представимое в таком виде
    $1-2^{-53} = 2^{-53} \cdot (2^{53}-1) = (-1)^0 \cdot 2^{1022-1023} \cdot (2^{53}-1) \cdot 2^{-52}$
    0 01111111110 1111111111111111111111111111111111111111111111111111

В дальнейшем будем называть действительные числа, точно представимые в виде чисел с плавающей точкой, просто *представимыми*. Ясно, что не каждое действительное число представимо. Например, π — иррационально, а все представимые числа рациональны, точнее даже двоично-рациональны. 1/3 тоже не представимо, поскольку не является двоично-рациональным числом. Числа $2^{10000000}$ и $2^{-10000000}$ не представимы, хотя и двоично-рациональны, потому что первое слишком велико, а второе слишком мало по абсолютной величине, даже для представления с четырехкратной точностью. Числа, находящиеся слишком близко к представимым, тоже не представимы, например, для представления $1+2^{-1000}$ нужна мантисса из 1000 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 квадратного корня из -0. При сложении, вычитании, умножении, делении, вычислении остатка от деления и преобразованиях типов всегда в случае нулевого результата возвращается 0.

2.2 Требования стандартов к реализациям математических функций

Поскольку не все действительные числа представимы, возникает проблема представления результатов математических функций в тех случаях, когда такой результат не представим.

Казалось бы, естественно потребовать, чтобы реализация функции возвращала в качестве результата число с плавающей точкой, являющееся ближайшим к результату точного вычисления функции для тех же самых значений аргументов.

Однако, ни стандарты IEEE 754 и IEEE 854, ни стандарт языка C ISO/IEC 9899 [8], ни стандарт переносимого интерфейса операционной системы IEEE 1003.1 [9] (известный как POSIX), описывающие библиотеку математических функций языка C не фиксируют такого требования для большинства функций.

Стандарты IEEE 754 и IEEE 854 описывают работу только сложения, умножения, вычитания и деления чисел с плавающей точкой, а также вычисления остатка от деления, извлечения квадратного корня и преобразований между типами с плавающей точкой и между ними и целочисленными типами. Соответственно, только для этих операций требуется возвращать результат, полученный из точного приведением к ближайшему представимому числу согласно действующему режиму округления. Возможны 4 режима округления: просто к ближайшему, к 0, к $+\infty$ и к $-\infty$. Кроме того, эти стандарты требуют аккуратного выставления флагов переполнения, слишком маленького результата или неточного результата при работе этих операций.

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

Отсутствие ограничений на точность вычисления математических функций может привести к накоплению погрешностей и серьезным ошибкам при многократном использовании этих функций в приложениях для математического моделирования, и, соответственно, неверным результатам работы таких приложений.

В последние 5-10 лет появились предложения стандартизовать необходимые для аккуратного моделирования требования к реализациям математических функций [10,11]. Многие инициаторы этой деятельности работают в проекте

Arenaire [12], совместно проводимом INRIA, CNRS и Высшей Нормальной школой Лиона, Франция. В результате активно разрабатывается набор стандартов ICO/IEC 10967 [13-15], формулирующий естественные ограничения на работу реализаций большинства математических функций. Эти ограничения касаются нескольких аспектов.

- Возможные погрешности вычисления функций выражены в терминах *единиц последнего разряда (unit in the last place, ulp)* [7]. Наилучшее приближение дало бы точность в 0.5 ulp, т.е. вычисленный результат отличался бы от точного значения функции не более чем на половину единицы последнего разряда мантиссы результата.
  Однако природа математических функций такова, что такая точность не является практически обоснованной во многих случаях, хотя ее достижение потребовало бы значительных усилий от разработчиков библиотек. Это связано с тем, что 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. Для широко используемых математических функций были проведены оценки возрастания погрешности при их вычислении, и на основании этих оценок были сформулированы более практичные требования к точности вычислений, ограничивающие погрешность результата величиной от 0.5 ulp до 2 ulp, в зависимости от функции [14].
- Стандарты серии ISO/IEC 10967 требует от реализации математической функции сохранения знака ее точного значения для данного значения параметра. Также требуется, чтобы во всех интервалах монотонности функции ее реализации были монотонны таким же образом, т.е. там, где сама функция убывает, ее численная реализация должна убывать, а там, где функция возрастает, — возрастать.
  Исключением из этого правила 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 чисел с плавающей точкой в окрестности 0 гораздо больше, чем их плотность в окрестности 1 — между 1 и ближайшим к нему числом с плавающей точкой умещается много чисел, близких к 0. Для двойной точности ближайшее к 1 число равно $1-2^{-53}$, а ближайшее к 0 — это 2^{-1074}.

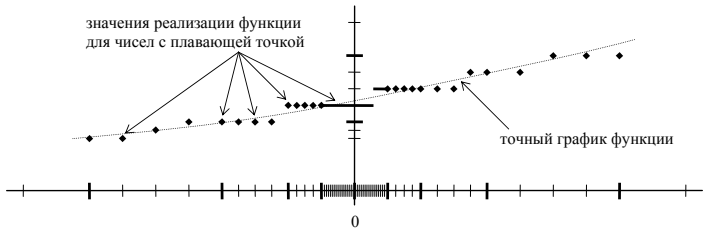

Рисунок 3. График функции, имеющей ненулевое значение в 0.

2.3. Дилемма составителя таблиц

Сформулированные требования к точности вычисления математических функций приводят к так называемой *дилемме составителя таблиц (Table Maker's Dilemma)* [16,17].

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.

Для иллюстрации дилеммы составителя таблиц приведем следующий пример. Пусть вычисляется функция *sin* для двоичных чисел с плавающей точкой, имеющих 6 битов мантиссы. Синус числа $11.1010_2 = 3.625_{10}$ (выделены биты мантиссы) равен $0.0111011 01111110..._2 = 0.063225984913\ldots_{10}$ (снова выделены биты мантиссы). 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 получив точный 14-й бит, мы сможем уверенно выбрать первое из них в качестве значения, ближайшего к точному результату.

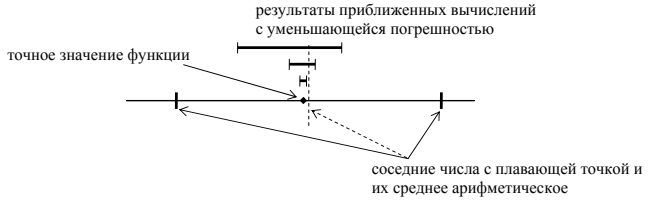

Рисунок 4. Дилемма составителя таблиц.
Чтобы определить ближайшее число с плавающей точкой, погрешность вычислений иногда должна быть значительно меньше половины расстояния между соседними такими числами.

Приведем более реалистичный пример для чисел с двойной точностью, имеющих 52 бита в мантиссе. Вычисляя значение натурального логарифма для

$1.0110000100111001010101011101110010000000001011111000_2 \cdot 2^{-35}$
получим
$-10111.1111000000101111100110111010111011100000001101010\ 1^{60}\ 0011\ldots_2$
Обозначение 1^{60} означает, что единица повторяется 60 раз. Таким образом, для получения корректно округленного значения нужно вычислять логарифм в этой точке с относительной погрешностью, не превосходящей 2^{-113}.

Описанные примеры показывают, что для выбора ближайшего числа с плавающей точкой иногда нужно использовать гораздо более точные вычисления, чем это позволяет сделать тип таких чисел. Аналогичные примеры существуют и для других режимов округления. Если используется режим округления к 0, к $+\infty$ или к $-\infty$, а точное значение функции лежит очень близко к представимому числу, необходимо добиться погрешности настолько маленькой, чтобы точно определить, превосходит оно это число или нет.

Например, натуральный логарифм для числа
$110101100.0101000010110100000010011100100010101110110_2$
равен
$110.000011110101001011110011011110101110110011111110011\ 1^{61}\ 0101..._2$.

Для практически всех часто используемых функций их значения в «обычных» (т.е., не равных 0, 1 или 2) двоично-рациональных числах не являются рациональными, и поэтому не могут ни быть представимыми, ни лежать в точности посередине между двумя представимыми числами. Из этого в силу конечности множества представимых чисел следует, 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. 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} [17]. 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 организации 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. В то же время, проводить настолько точные вычисления для всех значений аргументов очень неэффективно. Для повышения эффективности вычислений нужно уметь выбирать их точность в зависимости от аргументов функции.

3. Обзор работ по проверке корректности 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 [18], хотя он был выпущен уже довольно давно и частично устарел. Более современное изложение методов вычисления элементарных функций (являющихся только подмножеством рассмотренных в [18]) можно найти в книге Muller [19].

Значительно реже встречаются исследования, в которых не только формулируются методы вычисления каких-либо функций, но и доказывается их корректность. Под корректностью имеется в виду достижение определенной точности результатов при определенных значениях параметров метода.

Только часть из этих работ посвящена вычислениям на числах с плавающей точкой, представление которых определяется в IEEE 754. Большинство из таких исследований связано с корректностью работы алгоритмов вычисления функций, реализованных в специализированном аппаратном обеспечении. Помимо точности вычислений здесь приходится учитывать разнообразные режимы округления, определяемые в IEEE 754, корректность выставляемых флагов, например, переполнения, а также правильность работы алгоритма на специальных значениях — бесконечностях и NaN (см. далее).

Гораздо реже встречаются исследования по строгому тестированию правильности реализаций математических функций, работающих с числами с плавающей точкой.

Таким образом, все имеющиеся работы по проверке правильности работы реализации математических функций можно разделить на следующие группы.

- Работы по формальной верификации определенных алгоритмов.
- Работы по тестированию реализаций математических функций.

3.1. Работы по формальной верификации определенных алгоритмов

Формальная верификация корректности алгоритмов вычислений математических функций чаще всего проводится при проектировании блоков вычислений с плавающей точкой универсальных процессоров или при разработке специализированных вычислительных процессоров [28,32,34-52,54].

Основой для такой верификации всегда является точное знание реализованного алгоритма и формализация основных требований стандарта IEEE 754, необходимая для их строгой проверки. Формализация этих требований проводилась несколько раз в различных формализмах: в языках формальных спецификаций Z [20], Barrett в 1989 [21], и VDM [22], Wichmann в том же году [23], в формализмах инструментов автоматизации доказательств Nurpl [24], PVS [25], HOL [26] и ACL2 [27] —

O'Leary с соавторами для Nurpl в 1994 [28], Miner для PVS в 1995 [29,30], Carreno для HOL в 1995 [30,31], Moore с соавторами для ACL2 в 1996 [32], Harrison для HOL в 1996 [33].

В работах Verkest с соавторами [34] и Cornea-Hagesan [35,36] все доказательства выполнялись вручную. Но в большей части исследований такого рода, ввиду значительной сложности верификации практически важных систем, применяются только формализации, которые могут использоваться инструментами для автоматизированного доказательства теорем.

Статьи [32,34,37-41] имеют дело только с алгоритмами умножения и/или деления чисел с плавающей точкой и представляют примеры верификации корректности таких алгоритмов по отношению к требованиям IEEE 754.

В работах [35,36,42-48] верифицируются также алгоритмы вычисления квадратного корня, иногда еще и вычисление остатка от деления для чисел с плавающей точкой и преобразования между различными такими типами и между ними и целыми числами.

В связи с тем, что в стандарте IEEE 754 определены только арифметические действия, вычисление остатка от деления и квадратного корня, проверке правильности вычисления других функций посвящено значительно меньше работ. В [49,50] описываются практические примеры формальной верификации алгоритмов, вычисляющих экспоненциальную функцию. В статье [51] верифицировалось вычисление синуса и косинуса, а в [52] — натурального логарифма. Все четыре работы были выполнены Harrison и его коллегами. В работе [53] нескольких исследователей из проекта Arenaire [12], представлен пример верификации вычисления натурального логарифма с помощью специализированного инструмента Gappa, позволяющего автоматизировать доказательства, касающиеся свойств элементарных функций. Других примеров верификации математических функции в доступной литературе найти не удалось.

В перечисленных работах верифицировались отдельные элементы и блоки процессоров компаний Intel (Pentium II, Pentium III, Pentium 4 и Itanium [35,36,41-43,46,47,49-52]) AMD (K5, Athlon [32,44,45]), IBM (Power4 [48]). При этом использовались следующие инструменты автоматизации доказательств: HOL ([43,49-52]) и близкий к нему Nurple ([29]), PVS ([38,40,54]), ACL2 ([32,44,45,48]) и его предыдущая версия Nqthm ([34]). Помимо этого применялись различные комбинации автоматизированного доказательства теорем с проверкой моделей (model checking) и символической проверкой эквивалентности моделей ([37,39,41,42,46,47]).

По-видимому, до сих пор не было удачных попыток полностью проверить соответствие блока вычислений с плавающей точкой стандарту IEEE 754. Во всех найденных работах либо проверяются не все операции, определяемые стандартом, либо не проверяется их работа для специальных чисел с плавающей точкой — денормализованных, -0, бесконечных и NaN. Работа [54], похоже, единственная, в которой систематически рассматривается поведение части описанных в IEEE 754 операций на таких исключительных значениях.

Можно также отметить, что в процессоре, верифицированном в проекте, описанном в [42] впоследствии была найдена ошибка в операции преобразования числа с плавающей точкой в целое (так называемый FIST bug [55]). Это показывает, что формальная верификация сложной системы, будучи тоже достаточно сложной деятельностью, из которой нельзя исключить участие людей, сама по себе подвержена ошибкам.

3.2. Работы по тестированию реализаций математических функций

В Интернет можно найти огромное количество различных программ для тестирования функций, работающих с числами с плавающей точкой, см. например, [56]. К сожалению, подавляющее большинство таких тестов крайне несистематично и проверяет какой-то один аспект вычислений, реже — два-три таких аспекта.

Как указывается в [56] (см. также иллюстрации из [57,58]), несмотря на то, что стандартизация вычислений с плавающей точкой началась около 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 [59] 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 [60,61] и может быть получен с сайта [62]. 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 [63,64] 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 тестового набора, описанного выше. Она также проверяет только базовые арифметические операции.
  - Другой подход к построению тестов для операций IEEE 754 используется в среде FPgen [65,66]. Здесь, помимо специальных значений, в качестве тестовых данных используются числа с плавающей точкой,

удовлетворяющие некоторым шаблонам — например, 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 UCBTEST [67] предназначен для тестирования базовых арифметических действий и достаточно широкого набора математических функций. Он оформлен как набор программ на разных языках, включая Fortran и C, и наборов предопределенных входных данных для разных функций. В каждом тесте проверяется, что для заданных значений параметров данная функция возвращает число с плавающей точкой, ближайшему к точному значению функции.
  - Тестовый набор ELEFUNT [68], основанный на книге [69], также содержит тесты для многих математических функций в виде программ на C и Java и текстовых файлов с тестовыми данными и ожидаемыми результатами. Этот и предыдущий тестовый набор построены на основе проверки значений, возвращаемых реализациями функций для некоторых наборов аргументов. Методика выбора этих наборов аргументов, скорее всего, использовала несколько разных соображений.
    - Выделялись особые значения чисел с плавающей точкой: 0, -0, NaN $+\infty$, $-\infty$, минимальное положительное, максимальное положительное, числа, имеющие ровно один бит в мантиссе, и пр. (см. ниже).
    - Выделялись значения аргументов, значение функции для которых может быть точно представлено числом с плавающей точкой (например, $exp(0) = 1$, $cos(0) = 1$ и пр.).
    - Некоторые значения аргументов выбирались, по-видимому, случайно или 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, выбираются границы этого интервала и нескольких соседних с ним.
  - Аналогичные подходы — использование ряда специальных значений, границ интервалов, определяемых часто используемыми алгоритмами вычисления данной функции, и случайных значений — применялись для построения более полных тестовых наборов, например, набора Беркли [70], а также в статье [71].

Отдельно стоит отметить работы в рамках проекта Arenaire [16,17,72], 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 был разработан инструмент MPCheck [73] для тестирования корректности реализации элементарных функций с точки зрения сохранения монотонности, симметрий, ограничений на область значений и корректности округления.

Доступная литература по исследованиям, посвященным проверке корректности реализации математических функций, показывает, что формальные спецификации не используются для разработки тестов для таких функций, хотя все предпосылки и технологические возможности для применения такого подхода имеются. Математические функции имеют четко определенное поведение, которое практически однозначно понимается разработчиками ПО и легко может быть зафиксировано в виде международных стандартов, поэтому разработка их спецификаций не является слишком трудоемкой задачей.

Использование формальных спецификаций также дает ряд преимуществ, связанных с повышением удобства сопровождения и снижением трудоемкости модификации больших тестовых наборов, по сравнению с традиционными методами разработки тестов. Все систематические тестовые наборы для тестирования библиотек математических функций достаточно объемны, поэтому построение тестов для них на основе формальных спецификаций способно реализовать эти преимущества.

4. Предлагаемый подход

Подход к тестированию реализаций математических функций, предлагаемый в данном исследовании, основан на технологии UniTESK, использующей формальные спецификации требований к программному обеспечению для автоматизированного построения тестов на соответствие им.

Основные элементы технологии UniTESK [1-3], применительно к реализациям математических функций, таковы.

- Требования к поведению тестируемой системы представляются в виде *формальных спецификаций*, которые состоят из следующих частей:
  - *предусловия*, описывающие области определения функций;
  - *постусловия*, описывающие условия корректности возвращаемых функциями результатов;
  - *инварианты* типов данных, описывающих условия целостности данных.
- Чтобы сделать спецификации независимыми от конкретной сигнатуры функции (и, возможно, от языка программирования, на котором она реализована), разрабатывается слой *адаптеров* или *медиаторов*, связывающих спецификации и реализации соответствующих функций друг с другом.

- Постусловие каждой функции анализируется с тем, чтобы выделить из него различные возможные варианты ее поведения. Они чаще всего соответствуют ветвлениям в теле постусловия и различным выражениям, описывающим ограничения на результат функции. Такие различные варианты поведения функции называются ее *функциональными ветвями*.
  Набор функциональных ветвей определяет набор ситуаций, в которых реализация обязательно должна быть протестирована для того, чтобы проверить хотя бы один раз все выписанные ограничения.
- Помимо функциональных ветвей могут существовать другие ситуации, в которых тестирование данной реализации необходимо. Часть из этих ситуаций может быть получена более тонким анализом требований, а другая часть — на основе анализа возможных ошибок в конкретной реализации.
  Такие ситуации также могут быть описаны в спецификациях с помощью специальных конструкций.
- На основе полученного набора ситуаций, в которых поведение реализации функции должно быть проверено, разрабатывается *тестовый сценарий*. Он определяет множество наборов аргументов функции, с которыми она будет вызываться во время тестирования. Это множество должно обеспечивать покрытие всех выделенных ситуаций, что контролируется при помощи автоматического построения отчетов о покрытии тестовых ситуаций, определенных в спецификации.
  Возможности технологии UniTESK по автоматическому построению тестовой последовательности для тестирования математических функций не требуются. Единственное возможное исключение — наличие гипотез о зависимости работы реализаций математических функций от каких-то элементов внутреннего состояния тестируемой системы. В этом случае дополнительно должна быть построена обобщенная модель состояния, каким-то образом учитывающая те элементы, которые, как предполагается, могут влиять на работу функций.

Чтобы адекватно применять технологию UniTESK для построения тестов для математических функций, необходимо ответить на два вопроса: какие требования должны предъявляться к поведению их реализации, т.е. что именно должно быть написано в спецификациях; и на каких значениях параметров функций должно проводиться тестирование.

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

4.1. Метод определения требований к математическим функциям

Данный метод определения требований к поведению реализаций математических функций заимствует часть идей из стандарта ISO 10967 [13-15] и работ [10,11], посвященных вопросу разработки серии стандартов с повышенными требованиями к корректности вычисления математических функций и удовлетворяющих им библиотек. Некоторые элементы предлагаемого метода являются новыми и не встречаются в доступной литературе.

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

- **Область определения функции и особые точки функции.**
  - 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, но не должен быть NaN.
  - 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.
  - Если функция имеет особенность в точке 0, 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.
  - В остальных случаях должен возвращаться результат NaN.
    Особо нужно рассматривать такие значения аргументов, по поводу которых нет однозначного мнения о принадлежности их к области определения функции или о возможном продолжении ее в эту точку по непрерывности. Примером служит значение 0^0, которое иногда интерпретируется как 1, а иногда как NaN.
  - 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 $+\infty$ или $-\infty$ в соответствии со знаком точного значения.
  - 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 $+\infty$ или $-\infty$ в соответствии со знаком точного значения.
    Например, для значений $x > ln(2^{104}\cdot(2^{24}-1)) = 88{,}722839052\ldots$ $exp(x)$

не попадает в диапазон чисел однократной точности, поэтому реализация экспоненты для таких чисел должна возвращать $+\infty$.

- **Специальные значения, значения в 0, касательные и асимптоты.**
  - Нужно наиболее естественным образом определить значения функции для особых значений аргумента: -0, $+\infty$, $-\infty$. Обычно достаточно определять их как пределы, если те существуют, иначе как NaN.
  - Значение функции для значения аргумента NaN должно быть равно NaN.
  - Для некоторых значений аргумента значения функции известны точно. Если оба значения представимы, естественно потребовать от реализации этой функции возвращать именно точное ее значение в таких точках.
    Например, $exp(0) = cos(0) = ch(0) = 1$, $sin(0) = tan(0) = arcsin(0) = 0$ и т.п.
  - 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.
  - Так как около 0 плотность чисел с плавающей точкой больше, чем около любого другого значения, если значение функции в 0 не равно 0, даже если ее производная там ненулевая, должно быть выполнено то же самое правило: в некоторой окрестности 0 для всех чисел с плавающей точкой значение ее реализации должно быть одинаковым.
    Например, для экспоненты $e^0 = 1$, при этом $(e^x-1) < 2^{-24}$ при $x > 1$ и $(x-1) < ln(1+2^{-24}) = 5.9604642999\ldots\cdot10^{-8}$, поэтому для всех таких x, которых довольно много, реализация экспоненты с однократной точностью должна возвращать 1.
  - В тех случаях, когда функция имеет горизонтальные асимптоты, необходимо аккуратно определить границы, после которых ее значение должно стать постоянным.
    Например, для значений $x < ln(2^{-150}) = -103.97207708399\ldots$ значение e^x становится ближе к 0, чем к какому либо еще числу, представимому с однократной точностью. Поэтому реализация экспоненты для таких аргументов должна возвращать 0.
  - Казалось бы, естественно предъявить аналогичные требования к функциям, имеющим негоризонтальные асимптоты или асимптотически близких к другим функциям. Например, $ch(x) = (e^x+e^{-x})/2 \sim e^x$ для достаточно больших значений аргумента, или $sin(x) \sim x$ при $x \sim 0$.
    Однако во многих случаях такое требование не может быть сформулировано достаточно аккуратно с учетом различных режимов округления, зафиксированных в IEEE 754. Дело в том, что даже очень маленькая разность между двумя асимптотически близкими выражениями может дать отличие 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 = $-1.1000000000000000000000000000000000000000000000000001_2 \cdot 2^{-52}$

$exp(x)$ = $1.11111111111111111111111111111111111111111111111111 0^{53}100\ldots_2 \cdot 2^{-1}$,

а $1 + x$ = $1.11111111111111111111111111111111111111111111111100 1^{51}_2 \cdot 2^{-1}$.

Цветом здесь выделены биты, не помещающиеся в мантиссу числа двойной точности.

Похоже, что четкие требования можно предъявлять лишь по поводу соблюдения асимптотик вида $f(x) \sim x$ или $f(x) \sim -x$ (например, $sin(x) \sim x$ и $tg(x) \sim x$ при $x \sim 0$), поскольку при этом риск столкнуться с подобной проблемой невелик. Однако при этом нужно очень аккуратно вычислять границы действия подобных ограничений.

- **Область значений функции.**
  - 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 13176795/8388608 = 1.5707963705062866210937 5 > $\pi/2$. При этом $tg(arctg(2^{30}))$ = $-2.2877\ldots \cdot 10^7$, что противоречит основному свойству обратных функций.
  - Поскольку денормализованные числа выделяются из всего множества чисел с плавающей точкой, необходимо аккуратно определить интервалы, на которых значения функции должны быть денормализованными.
- **Монотонность и сохранение знака.**
  - 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.

Небольшие отклонения от этого правила можно допускать только на границах интервалов монотонности в том случае, если эти границы не представимы.

Другим исключением может быть поведение реализаций функций, часто меняющих характер монотонности (т.е. то убывающих, то возрастающих). Для значений аргументов, у которых единица последнего разряда превосходит длину интервала монотонности, проверять сохранение монотонности становится практически бесполезно. Примерами таких функций являются тригонометрические функции, а также функции Бесселя.

  - 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.

    Это требование может не соблюдаться для тех значений аргументов, для которых единица последнего разряда становится больше величины интервала смены знака, например, для тех же тригонометрических функций и функций Бесселя при больших значениях аргументов.
- **Симметрии и периодичность.**
  - Если математическая функция является четной или нечетной, этим же свойством должна обладать ее реализация.
  - 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, быть может, касающиеся только представимых значений аргумента с обеих сторон такого равенства.
  - Если функция симметрична относительно непредставимого значения, как, например, синус — $sin(\pi-x) = sin(x)$, его выполнение всегда может быть только приближенным.
  - Вообще, все важные функциональные уравнения, которым удовлетворяет данная функция, должны быть проанализированы на предмет выявления необходимости соблюдать их для реализаций этой функции. Иногда нужно определить области аргументов, где выполнение этих требований имеет практическое значение.

    Пример свойств такого рода, выполнение которых нужно обеспечивать (с точностью до 0.5 ulp), если задействованные значения аргументов функции

представимы,— 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 аргументы. Если период представим, то можно 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 значений аргумента, у которых единица последнего разряда больше, чем величина периода функции, проверка ее периодичности становится практически бесполезной.

- **Корректное округление.**

  Помимо всех перечисленных ограничений, нужно потребовать, чтобы результат, возвращаемый реализацией, получался из точного результата функции для данного аргумента при помощи принятой в текущей конфигурации процедуры округления. Иногда практически бессмысленно требовать точности 0.5-1 ulp, но во всяком случае погрешность выше 1.5-2.0 ulp должна рассматриваться как неточность соответствующей реализации.
  Стандарт IEEE 754 предписывает поддержку 4-х видов округления: к ближайшему представимому числу, к $+\infty$, к $-\infty$ и к 0.
  Часто три последних вида округления противоречат требований, сформулированным по остальным аспектам. 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.

После определения требований они оформляются в виде формальных спецификаций. Для записи требований, касающихся корректного округления точного значения математической функции, необходимо уметь вычислять ее правильно округленные значения. Сделать это можно несколькими способами.

- С помощью систем математических вычислений, например, Maple [74], Mathematica [75], MATLAB [76].
- С помощью библиотек корректно округляемых функций, таких, как разработанная на основе работ Ziv [77] IBM Accurate Portable MathLib [78], GNU MPFR [79], разрабатываемая Sun libmcr [80] или библиотек SCSLib [81] и CRlibm [82], разрабатываемых в рамка проекта Arenaire [16,83-86].
- Можно также разработать собственную реализацию функции на основе методов,

изложенных в книгах [18,19] и многочисленных статьях, или на основе методов интервальных вычислений [87-90].

4.2. Метод выбора тестовых данных

Предлагаемая методика выбора тестовых данных для тестирования математических функций основана на особенностях представление чисел с плавающей точкой, результатах работ [16,17,72] по вычислению «неудобных» значений аргументов и приведенном выше методе определения требований к реализациям таких функций, а также на технике построения тестов при помощи разбиения интервалов входных данных.

Основные шаги этого метода состоят в следующем.

- При определении требований к реализации функции по описанной выше методике числа с плавающей точкой разбиваются на ряд интервалов, в рамках каждого из которых действуют свои собственные ограничения. Это, например, интервалы монотонности и сохранения знака, интервалы, на которых функция имеет постоянное значение, включая бесконечные, интервалы, на которых она не определена. Множество концов этих интервалов будем называть *исходным множеством*.
- К исходному множеству добавляем числа 0, -0, $+\infty$, $-\infty$, минимальные и максимальные по абсолютной величине представимые числа, минимальное и максимальное денормализованные числа.
- Полученное исходное множество разбивает числа с плавающей точкой на набор интервалов.
  Для выбранных числовых параметров n и k на каждом из этих интервалов возможные тестовые значения выбираются следующим образом. Сначала интервал разбивается на n более мелких интервалов, равных по количеству содержащихся в них чисел с плавающей точкой. При этом возникает $(n+1)$ точка. Затем берутся все числа, лежащие в рассматриваемом интервале и отстоящие не более чем на k чисел с плавающей точкой от полученных точек. Получаемое множество точек назовем *пробным множеством*.

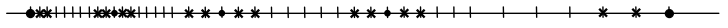

Рисунок 5. Схема выбора тестовых данных на одном интервале с параметрами n=4, k=2.

- К пробному множеству необходимо добавить ряд значений, которые приводят к необходимости гораздо более точного вычисления значения рассматриваемой функции для них (см. выше о дилемме составителя таблиц и [17]). Кроме того,

стоит добавить значения, которые требуют не максимально точных вычислений, а, например, на один бит меньше максимума.

- Если есть гипотезы о возможных ошибках в вычислении функции, связанных с конкретной структурой значения ее аргумента, например, для чисел, в мантиссе (или экспоненте) которых ровно одна единица или единицы чередуются с нулями, нужно также добавить такие числа в пробное множество. Нужно попытаться сформулировать такие гипотезы в виде набора шаблонов, которым должны удовлетворять битовые представления этих чисел.
- Наконец, в пробное множество нужно добавить несколько представителей NaN. Можно выбирать их по некоторым правилам, например, с минимальной и максимальной мантиссами, а также на основе различных комбинаций бит мантиссы.

Управлять количеством получаемых тестовых данных, тщательностью тестирования и временем выполнения тестов в рамках описанного метода возможно при помощи выбора параметров n и k для разбиения интервалов, а также с помощью использования разнообразных шаблонов, выбираемых на предпоследнем шаге.

4.3. Примеры применения предложенных методов

В этом разделе будут рассмотрены два примера использования предложенных методик для формулировки требований и выбора тестовых данных для тестирования двух математических функций — экспоненциальной функции и тангенса. Оба примера будут рассматриваться для случая чисел двойной точности.

4.3.1. Экспонента

В данном разделе термин «экспонента» везде обозначает экспоненциальную функцию $exp(x) = e^x$.

- **Область определения и особые точки.**

  Экспонента определена для всех действительных чисел, особых точек у нее нет.

  Она стремится к +∞ при $x \to +\infty$, поэтому нужно найти соответствующие граничные значения — наибольшее число с плавающей точкой, для которого значение экспоненты еще может быть представлено конечным числом, и наименьшее такое, что экспонента от него должна быть равна +∞.

  Максимальное конечное представимое в виде числа с двойной точностью значение равно $2^{971}\cdot(2^{53}-1)$. Его натуральный логарифм равен
  $709.782712893383996732223389910657145 5..._{10}$,
  ближайшее к этому значению представимое число есть

  $709.782712893383973096206318587064743041992187 5_{10} =$
  $1011000101.1100100001011111110111110100011100111101111_2$
  Экспонента от этого числа равна
  $1.11111111111111111111111111111111111111111111111100101010\ 00011011..._2\cdot 2^{1023}$.
  Здесь и далее выделены биты, не помещающиеся в мантиссу числа двойной точности.

  Экспонента от следующего представимого числа
  $709.782712893384086783044040203094482421875_{10} =$
  $1011000101.1100100001011111110111110100011100111110000_2$
  уже не может быть представлена конечным числом двойной точности. Таким образом, для всех представимых чисел, больших или равных этому, реализация экспоненты должна возвращать результат +∞.

  Для числа
  $1011000101.1100100001011111110111110100011100111101111_2$
  при режимах округления к ближайшему, к 0 или к -∞ должно выдаваться
  $1.11111111111111111111111111111111111111111111111100101010_2\cdot 2^{1023}$,
  а при округлении к +∞ должен выдаваться результат
  $1.11111111111111111111111111111111111111111111111100101011_2\cdot 2^{1023}$.

- **Специальные значения, значения в 0, касательные и асимптоты.**
  - Специальные значения.
    Естественно доопределить значения экспоненты в специальных точках следующим образом.

    $exp(-0) = 1$;
    $exp(+\infty) = +\infty$;
    $exp(-\infty) = 0$;
    $exp(\mathrm{NaN}) = \mathrm{NaN}$.
  - Точные значения и их окрестности, окрестность 0.
    Известно только одно точное значение экспоненты: $exp(0) = 1$. Учитывая бо́льшую плотность чисел с плавающей точкой в 0, нужно определить окрестность 0, в которой экспонента должна возвращать 1.

    Для чисел, больших 0, имеем
    $exp(1.11111111111111111111111111111111111111111111111111_2\cdot 2^{-54}) =$
    $1.00000000000000000000000000000000000000000000000000\ 01^{106}001..._2$;

    $exp(1.0_2\cdot 2^{-53}) =$
    $1.00000000000000000000000000000000000000000000000000\ 10^{53}100..._2$;

    $exp(1.11111111111111111111111111111111111111111111111111_2\cdot 2^{-53}) =$
    $1.00000000000000000000000000000000000000000000000000\ 1^{105}010..._2$;

    $exp(1.0_2\cdot 2^{-52}) =$
    $1.00000000000000000000000000000000000000000000000001\quad 0^{52}100..._2$.

Поэтому, для чисел от 0 до $(2^{53}-1)\cdot 2^{-106}$ = $1.11111111111111111111111111111111111111111111111111111_2\cdot 2^{-54}$ экспонента должна возвращать 1 при режимах округления к ближайшему, к 0 и к $-\infty$. При округлении к $+\infty$ для всех таких чисел, больших 0, результатом должно быть $1+2^{-52}$.

Для чисел от 2^{-53} до $(2^{53}-1)\cdot 2^{-105}$ = $1.11111111111111111111111111111111111111111111111111111_2\cdot 2^{-53}$ экспонента должна возвращать 1 при режимах округления к 0 и к $-\infty$, а при режимах округления к ближайшему или к $+\infty$, на этом интервале должно возвращаться следующее за единицей представимое число $1+2^{-52}$.

Для значения аргумента 2^{-52} должно возвращаться $1+2^{-52}$ при режимах округления к ближайшему, к 0 и к $-\infty$, а при округлении к $+\infty$ — $1+2^{-51}$.

Аналогично, для чисел, меньших 0

$exp(-1.0000000000000000000000000000000000000000000000000001_2\cdot 2^{-53})$ = $1.11111111111111111111111111111111111111111111111110\ 1^{52}010\ldots_2\cdot 2^{-1}$;

$exp(-1.0_2\cdot 2^{-53})$ = $1.11111111111111111111111111111111111111111111111111\ 0^{54}111\ldots_2\cdot 2^{-1}$;

$exp(-1.0000000000000000000000000000000000000000000000000001_2\cdot 2^{-54})$ = $1.11111111111111111111111111111111111111111111111111\ 01^{52}001\ldots_2\cdot 2^{-1}$;

$exp(-1.0_2\cdot 2^{-54})$ = $1.11111111111111111111111111111111111111111111111111\ 10^{55}111\ldots_2\cdot 2^{-1}$.

Поэтому, для $-(1+2^{-52})\cdot 2^{-53}$ экспонента должна возвращать $1-2^{-52}$ при округлении к $-\infty$ или 0, и $1-2^{-53}$ при округлении к ближайшему или $+\infty$.

На интервале от -2^{-53} до $-(1+2^{-52})\cdot 2^{-54}$, включая его концы, нужно возвращать $1-2^{-53}$ при округлении к $-\infty$, 0 или ближайшему, и 1 при округлении к $+\infty$.

Для чисел от -2^{-54} до 0 нужно возвращать $1-2^{-53}$ при округлении к $-\infty$ или 0 (кроме 0, для которого результат всегда 1) и 1 при округлении к ближайшему или $+\infty$.

Итоговые требования к значениям экспоненты в окрестности 0 для всех режимов округления суммированы в Таблице 1.

Округление	**к $-\infty$**	**к 0**	**к ближайшему**	**к $+\infty$**
Интервал/точка				
$-(1+2^{-52})\cdot 2^{-53}$	$1-2^{-52}$	$1-2^{-52}$	$1-2^{-53}$	$1-2^{-53}$
от -2^{-53} до $-(1+2^{-52})\cdot 2^{-54}$	$1-2^{-53}$	$1-2^{-53}$	$1-2^{-53}$	1
от -2^{-54} до -2^{-1074}	$1-2^{-53}$	$1-2^{-53}$	1	1
0	1	1	1	1

от 2^{-1074} до $(2^{53}-1)\cdot 2^{-106}$	1	1	1	$1+2^{-52}$
от 2^{-53} до $(2^{53}-1)\cdot 2^{-105}$	1	1	$1+2^{-52}$	$1+2^{-52}$
2^{-52}	$1+2^{-52}$	$1+2^{-52}$	$1+2^{-52}$	$1+2^{-51}$

Таблица 1. Требования к поведению экспоненты в окрестности 0.

- Асимптоты.

  Экспонента стремится к 0 при $x \to -\infty$, поэтому нужно определить значения аргумента, при которых значение экспоненты становится неотличимым от 0, а также важно знать, когда оно становится денормализованным.

  Минимальное положительное представимое число равно 2^{-1074}, а минимальное положительное нормализованное число равно 2^{-1022}. Их натуральные логарифмы равны, соответственно,

  $-744.44007192138126231410729844 6\ldots_{10}$ и

  $-708.39641853226410622441122813 0\ldots_{10}$.

  Ближайшие к ним представимые числа —

  $-744.44007192138121808966388925909996032714843 75_{10}$ =

  $-1011101000.01110000101010001000110110101110000111000011_2$ и

  $-708.39641853226407874899450689554214477539062 5_{10}$ =

  $-1011000100.01100101011110111010111101010111100110100 10_2$.

  Использование различных режимов округления, однако, «размазывает» эти границы на целые интервалы. Чтобы вычислить эти интервалы, определим значения, экспоненты которых при округлении к ближайшему будут округляться вверх и давать 2^{-1074}, а затем такие, экспоненты которых при этом будут округляться вниз и давать тот же результат. Имеем

  $exp(-1011101001.00100010000110101010010110100110000010100 10_2 = x_1)$ = $0.1^{45}011\ldots_2\cdot 2^{-1075}$

  $exp(-1011101001.00100010000110101010010110100110000010100 01_2 = x_2)$ = $1.0^{43}110\ldots_2\cdot 2^{-1075}$

  $exp(-1011101000.01110000101010001000110110101110000111000 100_2 = x_3)$ = $1.1^{42}011\ldots_2\cdot 2^{-1075}$

  $exp(-1011101000.01110000101010001000110110101110000111000 011_2 = x_4)$ = $1.0^{44}110\ldots_2\cdot 2^{-1074}$

  Аналогично, для минимального положительного нормализованного числа имеем

$exp(-1011000100.01100101011111011101011110101011110011010011_2 = x_5) = 0.11111111111111111111111111111111111111111111111001111011\ 1011\ldots_2 \cdot 2^{-1022}$

$exp(-1011000100.01100101011111011101011110101011110011010010_2 = x_6) = 1.00000000000000000000000000000000000000000000000001111011\ 1011\ldots_2 \cdot 2^{-1022}$.

Результирующие требования к значениям экспоненты, близким к минимальному положительному представимому числу и минимальному положительному нормализованному числу, приведены в Таблице 2.

Округление	**к** $-\infty$	**к** 0	**к ближайшему**	**к** $+\infty$
Интервал/точка				
$x_1 \geq x$	0	0	0	2^{-1074}
$x_3 \geq x \geq x_2$	0	0	2^{-1074}	2^{-1074}
x_4	2^{-1074}	2^{-1074}	2^{-1074}	2^{-1073}
$x_5 > x > x_4$	денормализованное			
x_5	$0.1^{43}001111011_2 \bullet 2^{-1022}$		$0.1^{43}001111100_2 \bullet 2^{-1022}$	
x_6	$1.0^{45}1111011_2 \bullet 2^{-1022}$		$1.0^{45}1111100_2 \bullet 2^{-1022}$	

Таблица 2. Требования к поведению экспоненты при переходе ее значений от нуля к положительным и от денормализованных к нормализованным значениям.

Асимптотика $exp(x) \sim 1+x$ с трудом поддается фиксации в виде четких требований по ее соблюдению (см. раздел 4.1), поэтому никаких связанных с ней ограничений не налагается.

- **Область значений.**
  Область значений экспоненты ограничена снизу нулем. Однако, это ограничение для всех значений аргумента следует из требований к точности вычислений. Интервал, на котором значения экспоненты денормализованы, уже определен выше при рассмотрении асимптот.

- **Монотонность и сохранение знака.**
  В связи со сказанным в предыдущем абзаце тот факт, что экспонента имеет только положительные значения, можно не проверять отдельно.
  Экспонента — возрастающая функция. Соответственно, ее реализация также должна возрастать.

- **Симметрии и периодичность.**
  Единственное важное свойство экспоненты — $exp(x+y) = exp(x) \cdot exp(y)$ — может выполняться лишь приблизительно для чисел с плавающей точкой, поскольку

значения экспоненты для всех таких чисел, кроме 0, трансцендентны согласно теореме Линдемана [91], следовательно, представимы лишь приблизительно. Если x или y равно 0, это свойство не добавляет ничего нового к условию $exp(0) = 1$.

- **Корректное округление.**
  Погрешность вычисления экспоненты согласно стандарту ISO/IEC 10967-2 [14] не должна превосходить 0.5-1.5 ulp. Остановимся на таких требованиях. В качестве дополнительной информации можно выдавать предупреждение специального вида каждый раз, когда фактическая ошибка превышает 0.5 ulp. Для вычисления правильно округленного значения экспоненты можно воспользоваться методом, описанным в [84] или готовыми библиотеками [79,81,82].

- **Тестовые значения.**
  Проведенный выше анализ свойств экспоненциальной функции, а также выделение денормализованных и исключительных чисел разбивают всю числовую прямую на следующие точки и интервалы.

  - $-\infty$;
  - $[-1.11111111111111111111111111111111111111111111111111111_2 \cdot 2^{1023}, -1011101001.00100010000110101010010110100110000010100102]$;
  - $[-1011101001.00100010000110101010010110100110000010100012, -1011101000.01110000101010001000110110101110001110001002]$;
  - $[-1011101000.01110000101010001000110110101110001110000112, -1011000100.01100101011111011101011110101011110011010012]$;
  - $[-1011000100.01100101011111011101011110101011110011010010_2, -1.00000000000000000000000000000000000000000000000000001_2 \cdot 2^{-53}]$;
  - $[-1.0_2 \cdot 2^{-53}, -1.00000000000000000000000000000000000000000000000000001_2 \cdot 2^{-54}]$;
  - $[-1.0_2 \cdot 2^{-54}, -1.0_2 \cdot 2^{-1022}]$;
  - $[-0.11111111111111111111111111111111111111111111111111111_2 \cdot 2^{-1022}, -1.0_2 \cdot 2^{-1074}]$;
  - -0;
  - 0;
  - $[1.0_2 \cdot 2^{-1074}, 0.11111111111111111111111111111111111111111111111111111_2 \cdot 2^{-1022}]$;
  - $[1.0_2 \cdot 2^{-1022}, 1.11111111111111111111111111111111111111111111111111111_2 \cdot 2^{-54}]$;
  - $[1.0_2 \cdot 2^{-53}, 1.11111111111111111111111111111111111111111111111111111_2 \cdot 2^{-53}]$;

- $[1.0_2 \cdot 2^{-52}, 1011000101.1100100001011111110111110100011100111101111_2]$;
- $[1011000101.1100100001011111110111110100011100111110000_2, 1.11111111111111111111111111111111111111111111111111111111_2 \cdot 2^{1023}]$;
- $+\infty$.

На полученных интервалах нужно строить тестовые значения в соответствии с методом, описанным в разделе 4.2 — деля интервал на n частей и выбирая точки, отстоящие от полученных (n+1) точки на интервале не более чем на k чисел с плавающей точкой.

Кроме этого, к полученному множеству тестовых значений надо добавить нескольких представителей NaN и набор чисел, вычисление корректно округленного значения экспоненты для которых наиболее трудоемко. Приведем несколько таких примеров, взятых из [17].

$exp(-1.11101101001100011000111011111101101100010011111101010_2 \cdot 2^{-27}) =$
$1.11111111111111111111111110000100101100111001110000100\ 1^{60}000\ldots_2 \cdot 2^{-1}$

$exp(-1.01000000000000000000000000000000000000000000000110010_2 \cdot 2^{-46}) =$
$1.11111111111111111111111111111111111111111111101100000\ 0^{85}101\ldots_2 \cdot 2^{-1}$

$exp(-1.00000000000000000000000000000000000000000000000001_2 \cdot 2^{-51}) =$
$1.11111111111111111111111111111111111111111111111111100\ 0^{101}101\ldots_2 \cdot 2^{-1}$

$exp(1.11111111111111111111111111111111111111111111111111_2 \cdot 2^{-53}) =$
$1.00000000000000000000000000000000000000000000000000000\ 1^{105}010\ldots_2;$
это конец одного из выделенных интервалов;

$exp(1.11111111111111111111111111111111111111111111110000000_2 \cdot 2^{-46}) =$
$1.00000000000000000000000000000000000000000000001111111\ 1^{84}010\ldots_2;$

$exp(1.00011111111111111111111111111111111111111111110101111_2 \cdot 2^{-45}) =$
$1.00000000000000000000000000000000000000000000010001111\ 1^{84}000\ldots_2;$

$exp(110.00001111010100101111001101111010111011001111110100_2) =$
$110101100.01010000101101000000100111001000101011101110\ 0^{58}100\ldots_2$

4.3.2. Тангенс

- **Область определения и особые точки.**
  Тангенс определен для всех действительных чисел, кроме чисел вида $\pi/2+\pi \cdot n$ для целого n.

  Эти исключительные точки являются полюсами, в которых тангенс имеет пределы $+\infty$ справа и $-\infty$ слева. Чтобы описать требования на поведение тангенса в окрестностях его полюсов, необходимо определить, насколько близко могут быть расположены числа с плавающей точкой к числам такого вида.

  Любое число с плавающей точкой, большее 1, может быть представлено как $m \cdot 2^k$, где m — натуральное число, не превосходящее $2^{53}-1$, а k — натуральное число, не превосходящее 971. Если такое число расположено очень близко к

$\pi/2+\pi \cdot n = (2n+1) \cdot \pi/2$, значит $\pi/2 \approx m \cdot 2^k/(2n+1)$. Таким образом, можно искать подходящие числа как числители наилучших рациональных приближений к $\pi/2$. Для этого разложим числа $\pi \cdot 2^k$ при k от 52 до -971 в непрерывные дроби и, обрывая их в каком-либо месте, вычислим соответствующие подходящие дроби. Эти дроби приближают исходные числа наилучшим образом среди всех рациональных чисел, имеющих меньшие или такие же знаменатели [92].

Выполнение соответствующих вычислений дает 920 дробей, которые однозначно определяют числа с плавающей точкой двойной точности, приближающие кратные $\pi/2$ с относительной погрешностью не больше, чем 2^{-57}. Вот несколько первых таких чисел.

$29 \cdot \pi/2 = 101101.10001101100101111000100010111011100100011011101\ 1^{13}010\ldots_2$

$14479 \cdot \pi/2 =$
$10110001101011 1.10001111010111010010111110111111001010\ 0^{14}110\ldots_2$

$29327 \cdot \pi/2 =$
$10110011111110010.10111110011011101010011011100010100 11\ 1^{15}011\ldots_2$

И несколько последних.

$19761831728800425220636434033446152324993825185277180432073766513146269169656727129731216116421652756859501244493674110738470194927151489482511774679831026242898856015924422548763057919465062862344269554795169108098262423683067093402882407403932831289539148165440830110821552783846121469368255050296717288515 \cdot \pi/2 =$
$1.01100001101000111101101110001100100011010001001010010\ 0^{1026}111\ldots_2 \cdot 2^{1021}$

$25821523947670058422613090074599585356028174861892741995642203997420228657069880654939563026645306825088819738650716488771208080667133163829521648447033304912947642466796572897674333478446391562484648705065088870272031215819071084764463635405694814146853886027632473712562962598749697007058695124283117548543 \cdot \pi/2 =$
$1.11001110000101000011001011101010110001110000010001111\ 1^{1020}011\ldots_2 \cdot 2^{1021}$

Хотя относительная погрешность приближения кратного $\pi/2$ может достигать 2^{-1079}, абсолютное расстояние до него для большинства таких чисел находится в пределах 10^{-15}-10^{-17}. Только 25 из этих чисел лежат от полюсов на расстоянии, не превосходящем 10^{-17}, и только два — на расстоянии, не превосходящем 10^{-18}. В Таблице 3 перечислены значения 25 нечетных кратных $\pi/2$, ближайших к числам с плавающей точкой и значения тангенса для соответствующих чисел с плавающей точкой.

29·π/2 =
101101.100011011001011110001000101110111001000110111011 $1^{13}010\ldots_2$

tg(101101.100011011001011110001000101110111001000110111110$_2$) =
-1.011001101011100111101011110001001000010100001100011 0110$\ldots_2\cdot 2^{60}$

9206271·π/2 =
110111001010100011111000.101010111001011101011101001 $0^{30}111\ldots_2$

tg(110111001010100011111000.101010111001011101011101001$01_2$) =
1.000001010111010110000100110001000010100111011001110 0110$\ldots_2\cdot 2^{59}$

13804917977710436777 5·π/2 =
1.011110000010101101111010001000001101111101101101010 $0^{72}101\ldots_2\cdot 2^{67}$

tg(1.011110000010101101111010001000001101111101101101010$0_2\cdot 2^{67}$) =
1.010110100110001111001000001010110110110110101110000 0100$\ldots_2\cdot 2^{57}$

1104381301317041933816362171·π/2 =
1.011001101011110101010100001001001110010101100101010 $0^{95}101\ldots_2\cdot 2^{90}$

tg(1.011001101011110101010100001001001110010101100101010$1_2\cdot 2^{90}$) =
1.010110100101111000101011100010000110101000101010101 $1^{3}010\ldots_2\cdot 2^{57}$

22766476268703513307928625848890272869 51·π/2 =
1.010100000100110010101100010100011111000111101010111 $0^{137}100\ldots_2\cdot 2^{131}$

tg(1.010100000100110010101100010100011111000111101010111$1_2\cdot 2^{131}$) =
1.111001111100011111010000111101000011111110000001101 $10^{5}11\ldots_2\cdot 2^{58}$

96172722857413553889773286774367461924492 93·π/2 =
1.010110101101010110100110001011001011000111001100100 $1^{147}011\ldots_2\cdot 2^{143}$

tg(1.010110101101010110100110001011001011000111001100100$1_2\cdot 2^{143}$) =
-1.110001111010101101110011100010011100001001001101011 0110$\ldots_2\cdot 2^{56}$

39821240970566890198609316701156723710809012791952652 95·π/2 =
1.000001010011100110110100100001101000101001100010101 $0^{186}100\ldots_2\cdot 2^{182}$

tg(1.000001010011100110110100100001101000101001100010101$01_2\cdot 2^{182}$) =
1.101001111100101101111100101101000010100000001011000 $0^{3}110\ldots_2\cdot 2^{56}$

24370968482400552171450472624030626617306657131797443262171 9·π/2 =
1.111001111110010001001010011110001010110000011000101 1 $1^{203}000\ldots_2\cdot 2^{197}$

tg(1.111001111110010001001010011110001010110000011000110$0_2\cdot 2^{197}$) =
-1.000010011011010011111100000011000011111001100000000 1001$\ldots_2\cdot 2^{58}$

10177187664528444025108553957230483888384395218254373185418290015777 4575619·π/2 =
1.011010011110101010110000100110000101000101111001101 1 $0^{251}101\ldots_2\cdot 2^{246}$

tg(1.011010011110101010110000100110000101000101111001101$1_2\cdot 2^{246}$) =
1.011100100101011001111100101110010000010001111010100 0 $1^{3}001\ldots_2\cdot 2^{57}$

156249277917169151617598733440921420051656218436719721717317896429622 57024455·π/2 =
1.101100100001100101100011011001001101011101010000101 0 $1^{258}011\ldots_2\cdot 2^{253}$

tg(1.101100100001100101100011011001001101011101010000101$1_2\cdot 2^{253}$) =
-1.110111010001011010001000110001101001011010011010110 0 $1^{3}011\ldots_2\cdot 2^{57}$

716103071666603140433976021767282701287838494529113918846184097832074634780736497767226 81·π/2 =
1.110001000101110011010001000100010101010011011111110 1 $0^{300}100\ldots_2\cdot 2^{295}$

tg(1.110001000101110011010001000100010101010011011111110$1_2\cdot 2^{295}$) =
1.111111011011101001000110111101010101001110100000010 0 $10^{3}10\ldots_2\cdot 2^{57}$

795310219628587798275121414029685041242685024890895917538542871690822757973300060931533046037509210740152847560365069332549·π/2 =
1.111000111100101010011011011011000110010101011100101 1 $0^{412}100\ldots_2\cdot 2^{408}$

tg(1.111000111100101010011011011011000110010101011100101$1_2\cdot 2^{408}$) =
1.100111111001111001010101000100100110011011000011110 0 $01^{3}00\ldots_2\cdot 2^{56}$

43776696795279815301055904915945700987301503939761311288501927384592341875548759379119888653528011155196936821017806703121641863603969532319521876 7·π/2 =
1.101110001000110010111011010011100011001001010111010 1 $1^{491}011\ldots_2\cdot 2^{487}$

tg(1.101110001000110010111011010011100011001001010111010$1_2\cdot 2^{487}$) =
-1.101000101100001011001100101111000001101001100011001 0 0101$\ldots_2\cdot 2^{56}$

11589330310381988626449129598171351352584091074116125961273394956149194566830509912960764804898095642685642936734217689884240195105814757430529332916887316386915217497 3·π/2 =
1.100010110010100001100111011011001101110011000101101 0 $1^{560}000\ldots_2\cdot 2^{555}$

tg(1.100010110010100001100111011011001101110011000101101$1_2\cdot 2^{555}$) =
-1.000001010000100010111001101101100011001010011011100010 0101$\ldots_2\cdot 2^{57}$

40912440333305918481550440392603355393318151657874369033644916645778808434302571208346583072956982613624207942025305084446951887841132505926354378025487374488888933105589584 3·π/2 =
1.010011001001011011000001000100010011010011010011010 1 $1^{583}010\ldots_2\cdot 2^{577}$

tg(1.010011001001011011000001000100010011010011010011010$1_2\cdot 2^{577}$) =
-1.011001010101110011111001111000100011110001001101011 1 $0^{3}101\ldots_2\cdot 2^{58}$

65429133524241263178237370719527517029251219555784828550385052067786873314305362285134035334062105379517142891899490313942463038820570475196164305791164553871385493457771955165116037 05·π/2 =
1.100000110000000001001111000101110100111100010110110101 0 $1^{619}011\ldots_2\cdot 2^{614}$

tg(1.100000110000000001001111000101110100111100010110110101$1_2\cdot 2^{614}$) =
-1.111000010110001000011111101100010011001101101010010 0 $1^{2}011\ldots_2\cdot 2^{57}$

592879322122707460982103890731067206764739020526842881081874689912072793995597687079339869172467423668394241772073902135153555728324157582755209798363213599827165644437783394500726283264184124810624461674 9·π/2 = 1.11011011010000011111001111001011011100011101011111011 $0^{686}111\ldots_2\cdot 2^{680}$ *tg*(1.11011011010000011111001111001011011100011101011111011$_2\cdot 2^{680}$) = 1.00010000101110100110111100111111010001100111110101101 $1^{2}010\ldots_2\cdot 2^{58}$
3063657226093116350633242532094876714714643482131289645284524658773942999960543512781934069734059379292127565945745701814493759190791305150957121333243526392950316795142634772419467533924565723086696715336885·π/2 = 1.11011111101010001101000110001111001010110011111011011 $1^{693}011\ldots_2\cdot 2^{689}$ *tg*(1.11011111101010001101000110001111001010110011111011011$_2\cdot 2^{689}$) = -1.10111001111110000001100110101101100010110101111001111 $10^{5}11\ldots_2\cdot 2^{56}$
9497144057238601559557116613969120123070769148998093918081547859574885577578104219175989760881732308451273779191276410784862786933003137631365257348287745336681480057589193340152823848292309678678336010555625307924541631728123291744094758 7·π/2 = 1.01101110100011010111011110001100100101001101011001100 $0^{798}100\ldots_2\cdot 2^{794}$ *tg*(1.01101110100011010111011110001100100101001101011001100$_2\cdot 2^{794}$) = 1.10011110011101000100110111111100111010110100011111100 $0101\ldots_2\cdot 2^{56}$
3386417804515981120643892082331156599120239393299838035242121518428537554064774221620930267583474709602068045686026362989271814411863708499869721322715946622634302011697632972907922558892710830616034038541342154669787134871905353772776431251615694251273653·π/2 = 1.01101010110001011011001001100010110010100001111111110 $1^{857}011\ldots_2\cdot 2^{849}$ *tg*(1.01101010110001011011001001100010110010100001111111111$_2\cdot 2^{849}$) = -1.11011001101110101001101001111001011101010110001101011 $1010\ldots_2\cdot 2^{60}$
8868575829470627220418957918782028042240987728548936186825417092733481413259359225243696332761908487780761905863396090159097561509545783128270032066815761201988094806806429895980170440026649236925696745971537209251486766849430876658153895468385463882245934037·π/2 = 1.11001111111001001000001000101000010111111000111011011 $0^{866}111\ldots_2\cdot 2^{860}$ *tg*(1.11001111111001001000001000101000010111111000111011011$_2\cdot 2^{860}$) = 1.00011010001001000110101101101111100100100100010101110 $1^{2}000\ldots_2\cdot 2^{58}$

3710517247492390678257616037434677716493775510194174346323291344761126458802318029413443054532013548715976758198761455677008210344281793857548297657854677919504624882274899501417333347401027368287526638600496292082821293413803137438251471817936373754055651262592764256461590125145265·π/2 = 1.01000001000101110101011100110011100101111101010000001 $1^{944}000\ldots_2\cdot 2^{939}$ *tg*(1.01000001000101110101011100110011100101111101010000010$_2\cdot 2^{939}$) = -1.00011100011100101001010010101110111001111011101001000 $01^{4}01\ldots_2\cdot 2^{57}$
2279546380734115174345404151754570588470229884840784603837027148508807014891141753237567442204768461160038124562964557539085826195999224021453268678892858279441898282307658403370182273623572172030139660727062180342556656598709402543499828723172717478478481959016764581997283642795280146 29·π/2 = 1.11100001100110000111000100100010101101111110000000110 $0^{955}101\ldots_2\cdot 2^{951}$ *tg*(1.11100001100110000111000100100010101101111110000000110$_2\cdot 2^{951}$) = 1.01110001011101000000011111010010000000011110011001110 $1^{3}011\ldots_2\cdot 2^{56}$
2498282971274918376918494936137072280693967504497845059865504027622422069326569285540059089298879071869587152103490382783870770007392787737242425387035210957698354406032906275559140016567474666525771149154041586094589355949694637666612416973562212869408698231197078227159462685629931344926353232197 3·π/2 = 1.11100000000001001110001010011000101001000101111100001 $0^{997}100\ldots_2\cdot 2^{991}$ *tg*(1.11100000000001001110001010011000101001000101111100001$_2\cdot 2^{991}$) = 1.10111000110010111111011111101100111110101111111111010 $0^{2}111\ldots_2\cdot 2^{58}$
1976183172880042522063643403344615232499382518527718043207376651314626916965672712973121611642165275685950124449367411073847019492715148948251177467983102624289885601592442254876305791946506286234426955479516910809826242368306709340288240740393283128953914816544083011082155278384612146936825550502967172885 15·π/2 = 1.01100001101000111101101110001100100011010001001010 $0^{1026}111\ldots_2\cdot 2^{1021}$ *tg*(1.11100000000001001110001010011000101001000101111100001$_2\cdot 2^{1021}$) = 1.00010010101111000001000010010101011000011100110101000 $10^{3}11\ldots_2\cdot 2^{57}$

Таблица 3. Нечетные кратные π/2, ближайшие к числам с плавающей точкой и соответствующие значения тангенса.

В 20-м ряду Таблицы 3 приведено число с плавающей точкой, ближайшее к нечетному кратному π/2. Это же число упоминается в статье [93], там оно вычислено с помощью программы, созданной W. Kahan и S. McDonald [94].

Из этого следует, что приведенное в 20-м ряду значение тангенса является максимальным по абсолютной величине. Таким образом, для всех чисел с

плавающей точкой двойной точности при всех режимах округления значения тангенса ограничены по абсолютной величине $2.133485385753703936_{10}\cdot10^{18}$. Более того, для всех чисел, не представленных в Таблице 3, значения тангенса должны по абсолютной величине быть меньше, чем минимальное значение тангенса в этой таблице, находящее в 23-м ряду и равное $1.03991843343165024_{10}\cdot10^{17}$.

- **Специальные значения, значения в 0, касательные и асимптоты.**
  - Специальные значения.
    Значение тангенса в точке -0 можно определить равным как 0, так и -0. Второй способ выглядит предпочтительнее, поскольку он более точно отражает асимптотику $tg(x) \sim x$ при $x \sim 0$ и имеет аналогию в стандарте IEEE 754, определяющем *sqrt*(-0) = -0.

    Поскольку тангенс не имеет пределов в +∞ и -∞, его значением в этих точках можно считать только NaN. Таким образом, получаем следующее.

    $tg(-0) = -0$;
    $tg(+\infty)$ = NaN;
    $tg(-\infty)$ = NaN;
    tg(NaN) = NaN.
  - Окрестность 0, асимптотика.
    Известно, что $tg(0) = 0$. Кроме того, $tg(x) \sim x$ при $x \sim 0$. Определим окрестность 0, в которой значение тангенса должно совпадать со значением его аргумента.

    Имеем
    $tg(1.0010010100001011111111110000110110000100000101111010{0}_2\cdot2^{-26} = x_1) =$
    $1.0010010100001011111111110000110110000100000101111010\ 01^{50}01\ldots_2\cdot2^{-26} = t_1$;

    $tg(1.0010010100001011111111110000110110000100000101111010{1}_2\cdot2^{-26} = x_2) =$
    $1.0010010100001011111111110000110110000100000101111010\ 1^{52}10\ldots_2\cdot2^{-26} = t_2$;

    $tg(1.0111000100110111010001001001000100100011111011110101{1}_2\cdot2^{-26} = x_3) =$
    $1.0111000100110111010001001001000100100011111011110101\ 1^{50}011\ldots_2\cdot2^{-26} = t_3$;

    $tg(1.0111000100110111010001001001000100100011111011110110{1}_2\cdot2^{-26} = x_4) =$
    $1.0111000100110111010001001001000100100011111011110111\ 0^{55}100\ldots_2\cdot2^{-26} = t_4$.

    Следующие из этих соотношений требования к значениям тангенса в окрестности 0 для всех режимов округления сформулированы в Таблице 4. В этой таблице выражение *x*++ для числа с плавающей точкой *x* обозначает непосредственно следующее за *x* число с плавающей точкой.

    Анализ других нулей тангенса, $\pi\cdot n$ при $n \neq 0$, см. ниже, в пункте о монотонности и сохранении знака.

  - Другие точные значения.
    Кроме $tg(0) = 0$ заслуживают внимания соотношения $tg(\pi/4+\pi\cdot n) = 1$ и $tg(-\pi/4+\pi\cdot n) = -1$.

Округление	**к** -∞	**к** 0	**к ближайшему**	**к** +∞
Интервал/точка				
$x = -x_4$	$-(t_4$++)	$-t_4$		
$-x_2 \geq x \geq -x_3$	-((-*x*)++)	*x*	-((-*x*)++)	*x*
$0 > x \geq -x_1$	-((-*x*)++)	*x*		
$x = 0$	0			
$x_1 \geq x > 0$	*x*			*x*++
$x_3 \geq x \geq x_2$	*x*		*x*++	
$x = x_4$	t_4			t_4++

Таблица 4. Требования к поведению тангенса в окрестности 0.

Вычисление чисел, приближающих нечетные кратные π/4 с погрешностью, не превосходящей 2^{-52}, дает 726 значений, которые могут рассматриваться как тестовые данные. Однако для многих из них тангенс не всегда должен быть равен 1 или -1. Например, для числа, ближайшего к π/4, значение тангенса равно
$1.1111111111111111111111111111111111111111111111111111\ 01^{3}00\ldots_2\cdot2^{-1}$,
что дает 1 только при режиме округления к +∞.

Таких чисел с плавающей точкой, для которых 1 является ближайшим числом к значению их тангенса 199. В Таблице 5 перечислены только 3 из них, которые отстоят от кратных π/4 не более чем на 10^{-19}.

$29\cdot\pi/4 =$ $10110.11000110110010111100010001011101110010001101110\ 1^{13}010\ldots_2$ $tg(10110.11000110110010111100010001011101110010001101110{1}_2) =$ $1.00000000000000000000000000000000000000000000000000000\ 0^{8}101\ldots_2$
$9206271\cdot\pi/4 =$ $1.10111001010100011111000101010111001011101011101001010\ 0^{30}111\ldots_2\cdot2^{22}$ $tg(1.10111001010100011111000101010111001011101011101001{01}_2\cdot2^{22}) =$ $-1.00000000000000000000000000000000000000000000000000000\ 0^{7}111\ldots_2$

338641780451598112064389208233115659912023939329983803524212151842853755406477422162093026758347470960206804568602636298927181441186370849986972132271594662263430201169763297290792255889271083061603403854134215466978713487190535377277643125161569425127365 3·π/4 = 1.01101010110001011011001001100010110010100001111111110 $1^{857}011$…$_2$·2^{848}
tg(1.01101010110001011011001001100010110010100001111111111$_2$·2^{848}) = 1.00000000000000000000000000000000000000000000000000000000 $0^{8}100$…$_2$

Таблица 5. Кратные π/4, ближайшие к числам с плавающей точкой и соответствующие значения тангенса.

- Асимптоты.

  У тангенса нет горизонтальных асимптот.

  Асимптотика в 0 рассмотрена выше.

- **Область значений.**

  Область значений тангенса не имеет математических границ.

  Для чисел с плавающей точкой, как мы выяснили выше, тангенс не должен превосходить по абсолютной величине $2.133485385753703936_{10}\cdot10^{18}$. Однако это требование следует из ограничений на точность вычислений.

  Из Таблицы 4 также следует, что значение тангенса денормализовано в точности тогда, когда денормализован его аргумент.

- **Монотонность и сохранение знака.**

  Тангенс возрастает на каждом отрезке от $\pi/2+\pi\cdot n$ до $\pi/2+\pi\cdot(n+1)$. Монотонность имеет смысл проверять только для тех значений аргументов, которые попадают в один такой отрезок.

  Тангенс отрицателен на отрезках от $\pi/2\cdot(2n-1)$ до $\pi\cdot n$, равен 0 в точках $\pi\cdot n$ и положителен на отрезках от $\pi\cdot n$ до $\pi/2\cdot(2n+1)$.

  Для проверки аккуратной смены знака нужно определить числа с плавающей точкой, ближайшие к кратным π.

  Можно использовать те же 920 дробей, которые были найдены нами ранее для приближения π/2 — дроби для π отличаются только удвоенными числителями, а соответствующие числа с плавающей точкой имеют ту же мантиссу и на единицу б***о***льшую экспоненту.

  Ближайшие к нулям тангенса числа с плавающей точкой выглядят несколько иначе, чем ближайшие к полюсам, хотя соответствуют почти тем же числам, умноженным на π. Все такие числа, расстояние от которых до кратных π не превосходит 10^{-17}, а также значения тангенса для них представлены в Таблице 6.

29·π = 1011011.00011011001011110001000101110111001000110111011 $1^{13}010$…$_2$
tg(1011011.00011011001011110001000101110111001000110111110$_2$) = 1.01101101011000011011010110001100100110011100010000010 $1^{4}000$…$_2$·2^{-60}
9206271·π = 1101110010101000111110001.01010111001011101011101001010 $0^{30}111$…$_2$
tg(1101110010101000111110001.01010111001011101011101001010$_2$) = -1.11110101010011110101001000100111101001001110100000011 $1^{5}011$…$_2$·2^{-59}
22766476268703513307928625848890272869 51·π = 1.01010000010011001010110001010001111100011110101011111 $0^{137}100$…$_2$·2^{132}
tg(1.01010000010011001010110001010001111100011110101011111$_2$·2^{132}) = -1.00001100101101100000010011010011010011110011010000001 $0^{2}100$…$_2$·2^{-58}
24370968482400552171450472624030626617306657131797443262171 9·π = 1.11100111111001000100101001111000101011000001100011011 $1^{203}000$…$_2$·2^{198}
tg(1.11100111111001000100101001111000101011000001100011100$_2$·2^{198}) = 1.11101101010000010101111101010100110010001100101110000 0101…$_2$·2^{-58}
10177187664528444025108553957230483888384395218254373185418290015777 4575619·π = 1.01101001111010101011000010011000010100010111100110111 $0^{251}101$…$_2$·2^{247}
tg(1.01101001111010101011000010011000010100010111100110111$_2$·2^{247}) = -1.01100001111011001110110010011100010101110111111111101 $0^{2}111$…$_2$·2^{-57}
15624927791716915161759873344092142005165621843671972171731789642962 257024455·π = 1.10110010000110010110001101100100110101110101000010 $1^{258}011$…$_2$·2^{254}
tg(1.10110010000110010110001101100100110101110101000010111$_2$·2^{254}) = 1.00010010101110111011111100000011111111001011011110000 $0^{2}110$…$_2$·2^{-57}
7161030716666031404339760217672827012878384945291139188461840978320746347807364977672268 1·π = 1.11000100010111001101000100010001010101001101111111101 $0^{300}100$…$_2$·2^{296}
tg(1.11000100010111001101000100010001010101001101111111101$_2$·2^{296}) = -1.00000001001001000010100001110110110101111100000011010 $10^{4}10$…$_2$·2^{-57}
40912440333305918481550440392603355393318151657874369033644916645778 8084343025712083465830729569826136242079420253050844469518878411325059263543780254873744888889331055895843·π = 1.01001100100101101100000100010001001101001101001101101 $1^{583}010$…$_2$·2^{578}
tg(1.01001100100101101100000100010001001101001101001101102$_2$·2^{578}) = 1.01101110110001100111101111001111011101110101001000100 0100…$_2$·2^{-58}

65429133524241263178237370719527517029251219555784828550385052067786 68733143053622851340353340621053795171428918994903139424630388205704 75196164305791164553871385493457771955165116037 05·π = 1.10000011000000001001111000101110100111100010111010 1^{619}011…$_2$·2^{615} *tg*(1.10000011000000001001111000101110100111100010111010 11_2·2^{615}) = 1.00010000010010000011000001000000000010010001001010 1011…$_2$·2^{-57}
59287932212270746098210389073106720676473902052684288108187468991207 27939955976870793398691724674236683942417720739021351535557283241575 82755209798363213599827165644437783394500726283264184124810624461674 9·π = 1.11011011010000011111001111001011011100011101011111011 0^{686}111…$_2$·2^{681} *tg*(1.11011011010000011111001111001011011100011101011111011 $_2$·2^{681}) = -1.11100000100110000111100011010010001011100010100111 01 0^{2}101…$_2$·2^{-58}
33864178045159811206438920823311565991202393932998380352421215184285 37554064774221620930267583474709602068045686026362989271814411863708 49986972132271594662263430201169763297290792255889271083061603403854 13421546697871348719053537727764312516156942512736 53·π = 1.01101010110001011011001001100010110010101000011111111 0 1^{857}011…$_2$·2^{850} *tg*(1.01101010110001011011001001100010110010101000011111111 $_2$·2^{850}) = 1.00010100101011100111001011100110101110100010001011 10 1^{4}010…$_2$·2^{-60}
88685758294706272204189579187820280422409877285489361868254170927334 81411325935922524369633276190848778076190586339609015909756150954578 31282700320668157612019880948068064298959801704400266492369256967459 71537209251486766849430876658153895468385463882245934 03·π = 1.11001111111001001000001000101000010111111000111011 01 0^{866}111…$_2$·2^{861} *tg*(1.11001111111001001000001000101000010111111000111011 01 $_2$·2^{861}) = -1.11010000100011110101100111000011001010111111 10100010 1^{2}010 …$_2$·2^{-58}
24982829712749183769184949361370722806939675044978450598655040276224 22069326569285540059089298879071869587152103490382783870770007392787 73724242538703521095769835440603290627555914001656747466652577114915 40415860945893559496946376666124169735622128694086982311970782271594 62685629931344926353232 1973·π = 1.11100000000001001110001010011000101001000101111100001 0^{997}100…$_2$·2^{992} *tg*(1.11100000000001001110001010011000101001000101111100001 $_2$·2^{992}) = -1.00101001010110100011101100001010011001001011000111 01 0100…$_2$·2^{-58}

Таблица 6. Кратные π, ближайшие к числам с плавающей точкой и соответствующие значения тангенса.

Отметим, что ближайшее к числу с плавающей точкой кратное π число, находящееся в 11-м ряду Таблицы 6, соответствует точно такому же целому

числу, что и ближайшее к числу с плавающей точкой нечетное кратное π/2.

- **Симметрии и периодичность.**
  Тангенс — нечетная функция, $tg(-x) = -tg(x)$. Это свойство должно учитываться при построении всех тестов с отрицательными значениями аргумента. Периодичность с трансцендентным периодом π, как и симметрии вида $tg(\pi/2-x) = 1/tg(x)$, выполняются на числах с плавающей точкой лишь приблизительно и, поэтому не могут быть использованы для формулировки строгих требований.

- **Корректное округление.**
  Погрешность вычисления тангенса согласно стандарту ISO/IEC 10967-2 [14] не должна превосходить 0.5-2.0 ulp. В качестве дополнительной информации можно выдавать предупреждение специального вида каждый раз, когда фактическая ошибка превышает 0.5 ulp.
  Для вычисления правильно округленного значения тангенса можно использовать известные библиотеки вычислений с заданной точностью [79,81,82].

- **Тестовые значения.**
  Идеальный набор интервалов для дальнейшего построения тестовых данных для тестирования реализаций тангенса мог бы состоять из всех его периодов, в которые попадает хотя бы два числа с плавающей точкой. Однако, таких интервалов слишком много — около $5.7 \cdot 10^{15}$. Поэтому необходимо выбрать только некоторые из таких периодов, руководствуясь каким-то правилом.

  Предлагается в качестве основы для тестов выбрать периоды тангенса, содержащие хотя бы одну из упоминавшихся выше точек, близких к кратным π, π/2 и π/4 (включая 0). Таких точек меньше, чем 3000, поэтому получающийся набор тестовых данных будет одновременно достаточно представительным и обозримым. Кроме того, к ним можно добавить несколько периодов, близких к содержащему 0, например, от -29π до 29π (раз уж число 29 дает одно из самых близких кратных π к числу с плавающей точкой). Для каждого из выбранных интервалов нужно вычислить его левый и правый концы — точки, ближайшие к нечетным кратным π/2 на этом интервале, его точку, ближайшую к кратному π, а также две точки на этом интервале, ближайшие к нечетным кратным π/4. Эти точки разбивают каждый такой период на четыре под-интервала. Все их занесем в исходное множество.

  Кроме того, в исходное множество попадут точки x_1, $-x_1$, x_2, $-x_2$, x_3, $-x_3$, x_4, $-x_4$, вычисленные при рассмотрении асимптотики тангенса в 0, сам 0, -0, +∞ и -∞, а также минимальное и максимальное положительные денормализованные числа и противоположные им.

Полученные таким образом интервалы можно разбить далее, пользуясь техникой, описанной в разделе 4.2.

Кроме этого, к тестовым значениям надо добавить нескольких представителей NaN и набор чисел, вычисление корректно округленного значения тангенса для которых наиболее трудоемко. Вот примеры таких чисел из [17].

$tg(1.11011111111111111111111111111111111111111111111100011111_2 \cdot 2^{-22}) =$
$1.11100000000000000000000000000000000000000000101010001\ 01^{78}01\ldots_2 \cdot 2^{-22}$

$tg(1.01100111111111111111111111111111111110100001000101001_2 \cdot 2^{-18}) =$
$1.01101000000000000000000000000000000001000111001100001\ 1^{58}010\ldots_2 \cdot 2^{-18}$

$tg(1.01010000010010000110101100101111100001110000000101001_2 \cdot 2^{-5}) =$
$1.01010000011110001100111010111111111111001110001110010\ 10^{57}10\ldots_2 \cdot 2^{-5}$

$tg(1.01000110101011000011011100100010010000110101001101101_2 \cdot 2^{-1}) =$
$1.01111011101001001001111101110011100110000010100111101\ 1^{55}001\ldots_2 \cdot 2^{-1}.$

5. Заключение

В данной работе проведен анализ проблем проверки корректности вычисления математических функций, реализованных на основе чисел с плавающей точкой. Этот анализ, а также обзор литературы по данному вопросу дают возможность утверждать, что задача построения систематических тестов для библиотек математических функций достаточно актуальна.

Несмотря на наличие серьезных исследований этого вопроса, например, в работах [16,17,72,73], ведущихся в рамках проекта Arenaire [12], работ, в которых для построения тестов таких функций использовались бы их формальные спецификации, нет.

В рамках данной работы был предложен метод разработки тестов для реализаций математических функций на основе формальных спецификаций. Этот метод основывается на технологии UniTesK разработки тестов для ПО общего назначения и двух отдельных методиках: методике формирования требований к реализации конкретной математической функции и методике выбора тестовых данных для тестирования этой реализации с учетом особенностей представления чисел с плавающей точкой и сформулированных требований.

Методика формирования требований построена на основе анализа особенностей и специфических элементов поведения самой математической функции с учетом использования чисел с плавающей точкой для представления ее аргументов и значений.

Методика выбора тестовых данных основана на разбиении чисел с плавающей точкой на ряд интервалов, границами которых являются числа с плавающей точкой, специфические с точки зрения их представления или же с точки зрения сформулированных требований к поведению функции. Можно сказать, что большинство таких интервалов являются интервалами «однородного», т.е.

описываемого одним набором условий, поведения тестируемой функции. Кроме того, используются полученные в работах [17,72] числа с плавающей точкой, для которых определение корректно округленного значения данной функции требует наиболее высокой точности вычислений.

Разработанные в рамках данного исследования методики покрывают все основные проблемы разработки тестов для реализаций математических функций и позволяют перевести решение отдельных задач в этой области в практическую плоскость.

Литература

[1] I. Bourdonov, A. Kossatchev, V. Kuliamin, and A. Petrenko. *UniTesK Test Suite Architecture.* In Proc. of FME 2002. LNCS 2391, pp. 77–88, Springer-Verlag, 2002.

[2] В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. *Подход UniTesK к разработке тестов.* Программирование, 29(6):25–43, 2003.

[3] V. Kuliamin, A. Petrenko, N. Pakoulin. *Practical Approach to Specification and Conformance Testing of Distributed Network Applications.* In M. Malek, E. Nett, N. Suri, eds. Service Availability. LNCS 3694, pp. 68–83, Springer-Verlag, 2005.

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

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

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

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

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

[9] IEEE 1003.1-2004. *Information Technology — Portable Operating System Interface (POSIX)*. NY: IEEE, 2004.

[10] 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.

[11] 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.

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

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

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

[15] 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.

[16] V. Lefevre, J.-M. Muller, and A. Tisserand. *Toward Correctly Rounded Transcendentals*. IEEE Transactions on Computers, 47(11):1235–1243,November 1998.

[17] 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.

[18] M. Abramowitz and I. A. Stegun, eds. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1965.

[19] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Second edition. Birkhauser, Boston, 2006.
[20] http://www.zuser.org/z/
[21] G. Barrett. *Formal Methods Applied to a Floating-Point Number System*. IEEE Transactions on Software Engineering, 15(5):611–621, May 1989.
[22] http://www.csr.ncl.ac.uk/vdm/
[23] B. A. Wichmann. *Towards a Formal Specification of Floating Point*. The Computer Journal, 32:432–436, 1989.
[24] http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html
[25] http://pvs.csl.sri.com/
[26] http://www.cl.cam.ac.uk/Research/HVG/HOL/
[27] http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html
[28] J. O'Leary, M. Leeser, J. Hickey, and M. Aagaard. *Non-restoring Integer Square Root: A Case Study in Design by Principled Optimization*. In T. Kropf, R. Kumar, eds. Proc. of the 2-nd International Conference on Theorem Provers in Circuit Design (TPCD'94): Theory, Practice, and Experience. LNCS 901, pp. 52–71, Springer-Verlag, 1994.
[29] P. S. Miner. *Defining the IEEE-854 Floating-Point Standard in PVS*. Technical report TM-110167, NASA Langley Research Center, 1995.
[30] V. A. Carreno and P. S. Miner. *Specification of the IEEE-854 Floating-Point Standard in HOL and PVS*. Proc. of 10-th International Workshop on Higher Order Logic Theorem Proving and its Applications, Aspen Grove, Utah, USA, September 1995.
[31] V. A. Carreno. *Interpretation of IEEE-854 Floating-Point Standard and Definition in the HOL System*. Technical report TM-110189, NASA Langley Research Center, 1995.
[32] J. Moore, T. Lynch, and M. Kaufmann. *A Mechanically Checked Proof of the Correctness of the Kernel of the AMD5K86 Floating-Point Division Algorithm*. IEEE Transactions on Computers, 47(9):913–926, September 1998.
[33] J. Harrison. *Theorem Proving with the Real Numbers*. Technical report UCAM-CL-TR-408, University of Cambridge Computer Laboratory. UK, November 1996.
[34] D. Verkest, L. Claesen, and H. De Man. *A Proof on the Nonrestoring Division Algorithm and its Implementation on an ALU*. Formal Methods in System Design, vol. 4, 1994.
[35] M. Cornea-Hasegan. *Proving the IEEE Correctness of Iterative Floating-Point Square Root, Divide, and Remainder Algorithms*. Intel Technology Journal, 1998.
[36] M. Cornea-Hasegan. *IA-64 Floating Point Operations and the IEEE Standard for Binary Floating-Point Arithmetic*. Intel Technology Journal, Q4, 1999.
[37] M. D. Aagaard and C.-J. H. Seger. *The Formal Verification of a Pipelined Double-precision IEEE Floating-Point Multiplier*. Proc. ICCAD, IEEE, Nov. 1995, pp. 7–10.
[38] P. S. Miner, and J. F. Leathrum. *Verification of IEEE Compliant Subtractive Division Algorithms*. Proc. FMCAD'96, November 1996.
[39] E. M. Clarke, S. M. German, and X. Zhao. *Verifying the SRT Division Algorithm Using Theorem Proving Techniques*. Proc. CAV'96, LNCS 1102, Springer-Verlag, 1996.
[40] H. Ruess, N. Shankar, and M. K. Srivas. *Modular Verification of SRT Division*. Proc. CAV'96, LNCS 1102, Springer-Verlag, 1996.
[41] R. Kaivola and N. Narasimhan. *Formal Verification of the Pentium 4 Floating-Point Multiplier*. Proc. of Design, Automation and Test in Europe Conference and Exposition (DATE). IEEE, 2002.
[42] Y.-A. Chen, E. M. Clarke, P.-H. Ho, Y. Hoskote, T. Kam, M. Khaira, J. W. O'Leary, and X. Zhao. *Verification of All Circuits in a Floating-Point Unit Using Word-level Model Checking*. In Formal Methods in Computer-Aided Design, LNCS 1166, pp. 19-33, Springer-Verlag, 1996.

[43] J. R. Harrison. *Verifying the Accuracy of Polynomial Approximations in HOL*. TPHOLs'97, August 1997.
[44] D. M. Russinoff. *A Mechanically Checked Proof of IEEE Compliance of the Floating Point Multiplication, Division and Square Root Algorithms of the AMD-K7 Processor*. LMS Journal of Computation and Mathematics, 1:148–200, 1998.
[45] D. M. Russinoff. *A Mechanically Checked Proof of IEEE Compliance of the AMD K5 Floating-Point Square Root Microcode*. Formal Methods in System Design, 14 (1), January 1999.
[46] J. O'Leary, X. Zhao, R. Gerth, and C. H. Seger. *Formally Verifying IEEE Compliance of Floating-Point Hardware*. Intel Technology Journal, 1999.
[47] M. D. Aagaard, R. B. Jones, and R. Kaivola. *Formal Verification of Iterative Algorithms in Microprocessors*. Proc. DAC 2000. ACM, 2000.
[48] J. Sawada. Formal *Verification of Divide and Square Root Algorithms Using Series Calculations*. Proc. of ACL2 Workshop, Grenoble, France, April 2002.
[49] J. R. Harrison. *Floating-Point Verification in HOL Light: the Exponential Function*. Technical Report UCAM-CL-TR-428, University of Cambridge Computer Laboratory. UK, June 1997.
[50] A. T. Abdel-Hamid. *A Hierarchical Verification of the IEEE-754 Table-driven Floating-point Exponential Function Using HOL*. Master's thesis, Dept. Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada, 2001.
[51] J. Harrison. *Formal Verification of Floating Point Trigonometric Functions*. Proc. of Formal Methods in Computer-Aided Design, FMCAD 2000, LNCS 1954, pp. 217-233, Springer-Verlag, 2000.
[52] J. Harrison. *Floating Point Verification in HOL*. E. T. Schubert, P. J. Windley, and J. Alves-Foss, eds. Proc. of 8-th International Workshop on Higher Logic Theorem Proving and its Applications, Aspen Grove, UT, USA, September 1995. LNCS 971, pp. 186–199, Springer-Verlag, 1995.
[53] F. de Dinechin, C. Lauter, and G. Melquiond. *Assisted Verification of Elementary Functions*. INRIA Research Report RR-5683, September 2005.
[54] C. Jacob, C. Berg. *Formal Verification of the VAMP Floating Point Unit.* In Formal Methods in System Design, 26, pp. 227-266, Springer, 2005.
[55] http://support.intel.com/support/processors/flag/tech.htm, *Discussion of Flag Erratum*, 2002.
[56] http://www.math.utah.edu/~beebe/software/ieee/
[57] http://people.redhat.com/drepper/libm/
[58] W. Kahan. *What Can You Learn about Floating-Point Arithmetic in One Hour*? http://http.cs.berkeley.edu/~wkahan/ieee754status, 1996.
[59] N. L. Schryer. *A Test of Computer's Floating-Point Arithmetic Unit*. Computer Science Technical Report 89, AT&T Bell Labs, 1981.
[60] 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.
[61] 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.
[62] http://www.cant.ua.ac.be/ieeecc754.html
[63] R. Karpinski. *PARANOIA: A Floating-Point Benchmark*. Byte Magazine 10, 2 (Feb.), pp. 223–235, 1985.
[64] http://www.netlib.org/paranoia/

[65] 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.
[66] 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.
[67] http://www.netlib.org/fp/ucbtest.tgz
[68] http://www.math.utah.edu/pub/elefunt/
[69] W. Cody and W. Waite. *Software Manual for the Elementary Functions*. Prentice-Hall, Englewood Cliffs, NJ, 1980.
[70] 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.
[71] P.-T. P. Tang. *Accurate and Efficient Testing of the Exponential and Logarithm Functions*. ACM Transactions on Mathematical Software, 16(3):185–200, September 1990.
[72] 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.
[73] http://www.loria.fr/~zimmerma/mpcheck/
[74] http://www.maplesoft.com/
[75] http://www.wolfram.com/products/mathematica/index.html
[76] http://www.mathworks.com/products/matlab/
[77] A. Ziv. *Fast Evaluation of Elementary Mathematical Functions with Correctly Rounded Last Bit*. ACM Transactions on Mathematical Software, 17(3):410–423, September 1991.
[78] *IBM Accurate Portable MathLib*
http://rpmfind.net/linux/rpm2html/search.php?query=libultim.so.2
[79] http://www.mpfr.org/
[80] http://www.sun.com/download/products.xml?id=41797765
[81] http://www.ens-lyon.fr/LIP/Arenaire/Ware/SCSLib/
[82] http://lipforge.ens-lyon.fr/projects/crlibm/
[83] 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.
[84] D. Defour, F. de Dinechin, J.-M. Muller. *Correctly Rounded Exponential Function in Double Precision Arithmetic*. INRIA Research report RR-2001-26, July 2001.
[85] F. de Dinechin, C. Lauter, J.-M. Muller. *Fast and Correctly Rounded Logarithms in Double-Precision*. INRIA Research report RR-2005-37, September 2005.
[86] 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.
[87] 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.
[88] M. J. Schulte, E. E. Swartzlander. *Software and Hardware Techniques for Accurate, Self-Validating Arithmetic*. Applications of Interval Computations, pp. 381–404, 1996.
[89] N. Revol, F. Rouillier. *Motivations for an Aarbitrary Precision Interval Arithmetic and the MPFI Library*. Reliable Computing, 11(4):275–290, 2005.
[90] *MPFI Library* http://perso.ens-lyon.fr/nathalie.revol/mpfi_toc.html
[91] F. Lindemann. *Über die Zahl π*. Mathematische Annalen, vol. 20, pp. 213–225, 1882.

[92] А. Я. Хинчин. *Цепные дроби*. М: Наука, 1978.
[93] K. C. Ng. *Arguments Reduction for Huge Arguments: Good to the Last Bit*. 1992. Доступна как http://www.validlab.com/arg.pdf.
[94] W. Kahan. *Minimizing q*m − n*. 1983. Неопубликованные заметки, доступны по http://http.cs.berkeley.edu/~wkahan/testpi/nearpi.c.