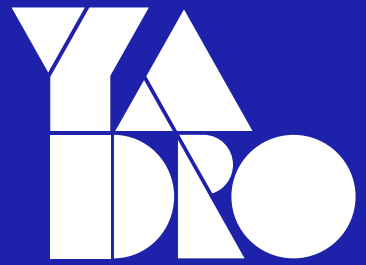


Как Иван промок под дождем по дороге на работу – О тестировании точности математических библиотек

Валерия Пузикова v.puzikova@yadro.com

Иван Рябинин i.ryabinin@yadro.com

Валерия Пузикова

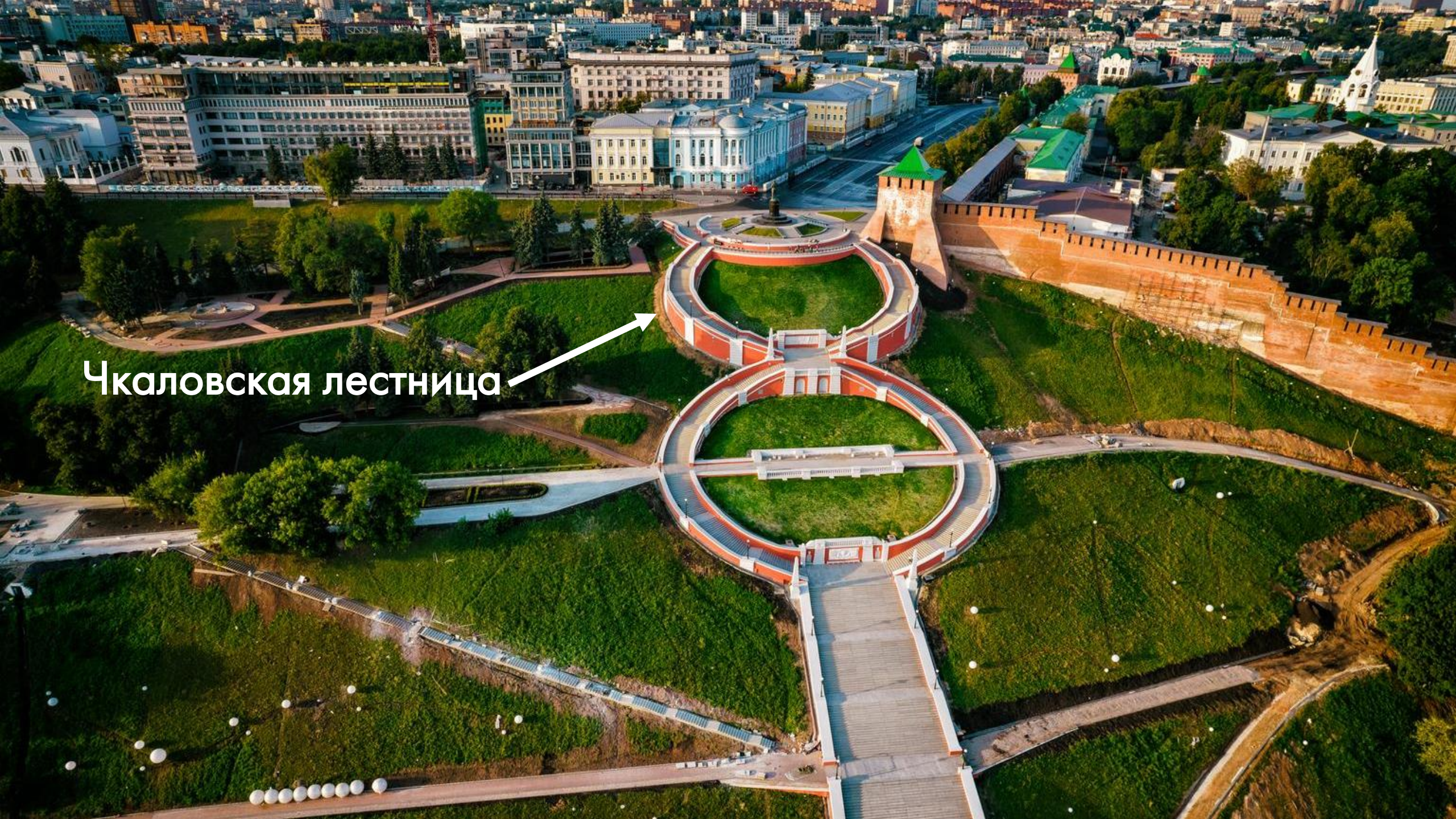
К.ф.-м.н., эксперт по разработке ПО, руководитель команды разработки математических библиотек, YADRO

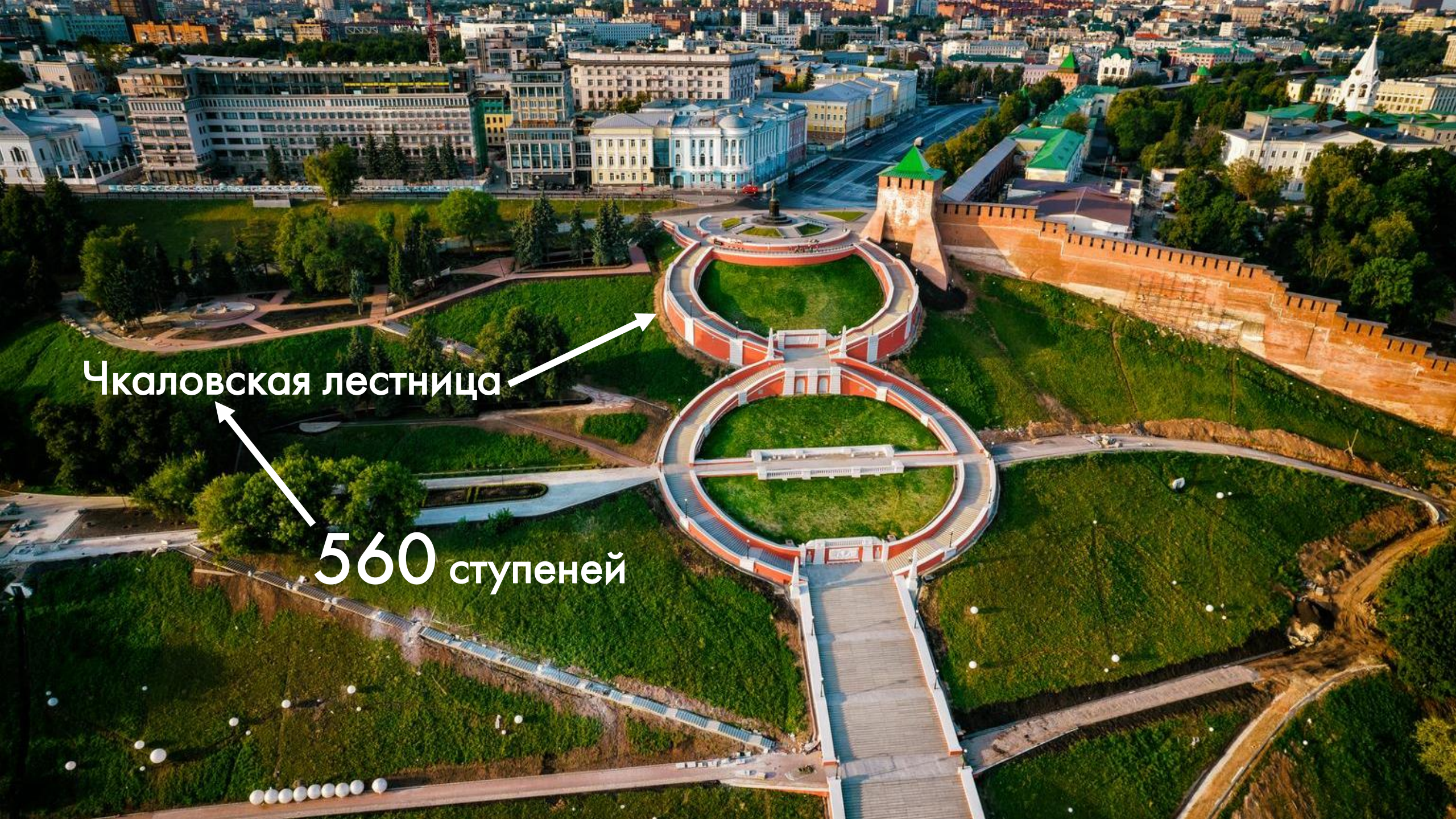
- С 2010 года разрабатывает и реализует на C/C++ с CUDA/MPI/OpenMP численные методы для решения задач линейной алгебры, вычислительной аэрогидродинамики, AR/VR.
- Работала в Huawei, Fortum, ИСП РАН им. В.П. Иванникова, МГТУ им. Н.Э. Баумана и др.

Иван Рябинин

Инженер-программист, YADRO

- С 2020 года разрабатывает и тестирует высокопроизводительные математические библиотеки.
- Писал и тестировал софт для беспилотных автомобилей и обработки сигналов в Intel.


Чкаловская лестница


Чкаловская лестница
560 ступеней


Иван сейчас за катером


560 ступеней...

Что узнаем о тестировании математических библиотек?

- Откуда в корректных вычислениях берутся ошибки (и как их избежать)?

Что узнаем о тестировании математических библиотек?

- Откуда в корректных вычислениях берутся ошибки (и как их избежать)?
- Как нам всего одним тестом обеспечить **100% code coverage**?

Что узнаем о тестировании математических библиотек?

- Откуда в корректных вычислениях берутся ошибки (и как их избежать)?
- Как нам всего одним тестом обеспечить **100% code coverage**?
- Как устроено большинство тестов стандартных математических библиотек и почему они не работают?

Что узнаем о тестировании математических библиотек?

- Откуда в корректных вычислениях берутся ошибки (и как их избежать)?
- Как нам всего одним тестом обеспечить **100% code coverage**?
- Как устроено большинство тестов стандартных математических библиотек и почему они не работают?
- **Как мы организовали тестирование точности для стандартной библиотеки libm?**

Что интересного при этом обсудим?

- Что из себя представляют числа с плавающей точкой, где эта точка плавает, и почему такие числа не поддаются измерению линейкой?

Что интересного при этом обсудим?

- Что из себя представляют числа с плавающей точкой, где эта точка плавает, и почему такие числа не поддаются измерению линейкой?
- Почему не работает арифметика?
- Зачем прогнозу погоды экспонента?

Где используются математические библиотеки?

ГДЕ ИСПОЛЬЗУЮТСЯ МАТЕМАТИЧЕСКИЕ БИБЛИОТЕКИ?

Их мало кто видит, но они всех ускоряют

* Рисунок сгенерирован при помощи Kandinsky 3.0

Математический бэкенд

BLAS / LAPACK

Основные **операции линейной алгебры над плотными матрицами и векторами**:

- сложение, вычитание, умножение на скаляр, произведения;
- решатели СЛАУ;
- матричные разложения;
- и др.

Sparse BLAS / Solvers

Основные **операции линейной алгебры над разреженными матрицами и плотными векторами**:

- сложение, вычитание, умножение на скаляр, произведения;
- решатели СЛАУ;
- предобуславливатели;
- и др.

libm

Математические функции:

- тригонометрические (и обратные);
- гиперболические (и обратные);
- степенные, логарифмические, экспоненциальные;
- и др.

Где используется libm?

- Активационные функции **в нейронных сетях**.
- Преобразования координат **в графических приложениях и компьютерном зрении**.
- Правые части уравнений **для моделирования физико-технических процессов**.

libm

Математические функции:

- тригонометрические (и обратные);
- гиперболические (и обратные);
- степенные, логарифмические, экспоненциальные;
- и др.

Где используется libm?

- Активационные функции **в нейронных сетях**.
- Преобразования координат **в графических приложениях и компьютерном зрении**.
- Правые части уравнений **для моделирования физико-технических процессов**.

☺ в т.ч. Weather Research & Forecasting Model – без экспоненты не было бы прогноза погоды.

libm

Математические функции:

- тригонометрические (и обратные);
- гиперболические (и обратные);
- степенные, логарифмические, экспоненциальные;
- и др.

Где можно найти реализации libm?

- В XNNPACK для TensorFlow(Lite).
- В Eigen для OpenCV.
- В специальных реализациях для отдельных платформ (Bionic для AOSP и т.д.).

libm

Математические функции:

- тригонометрические (и обратные);
- гиперболические (и обратные);
- степенные, логарифмические, экспоненциальные;
- и др.

Где можно найти реализации libm?

- В XNNPACK для TensorFlow(Lite).
- В Eigen для OpenCV.
- В специальных реализациях для отдельных платформ (Bionic для AOSP и т.д.).
- **Встроены в современные компиляторы (gcc, clang, ...).**

libm

Математические функции:

- тригонометрические (и обратные);
- гиперболические (и обратные);
- степенные, логарифмические, экспоненциальные;
- и др.

Где можно найти реализации libm?

- В XNNPACK для TensorFlow(Lite).
- В Eigen для OpenCV.
- В специальных реализациях для отдельных платформ (Bionic для AOSP и т.д.).
- **Встроены в современные компиляторы (gcc, clang, ...).**

* Большинство имплементаций предоставляют ***полную точность***.

libm

Математические функции:

- тригонометрические (и обратные);
- гиперболические (и обратные);
- степенные, логарифмические, экспоненциальные;
- и др.

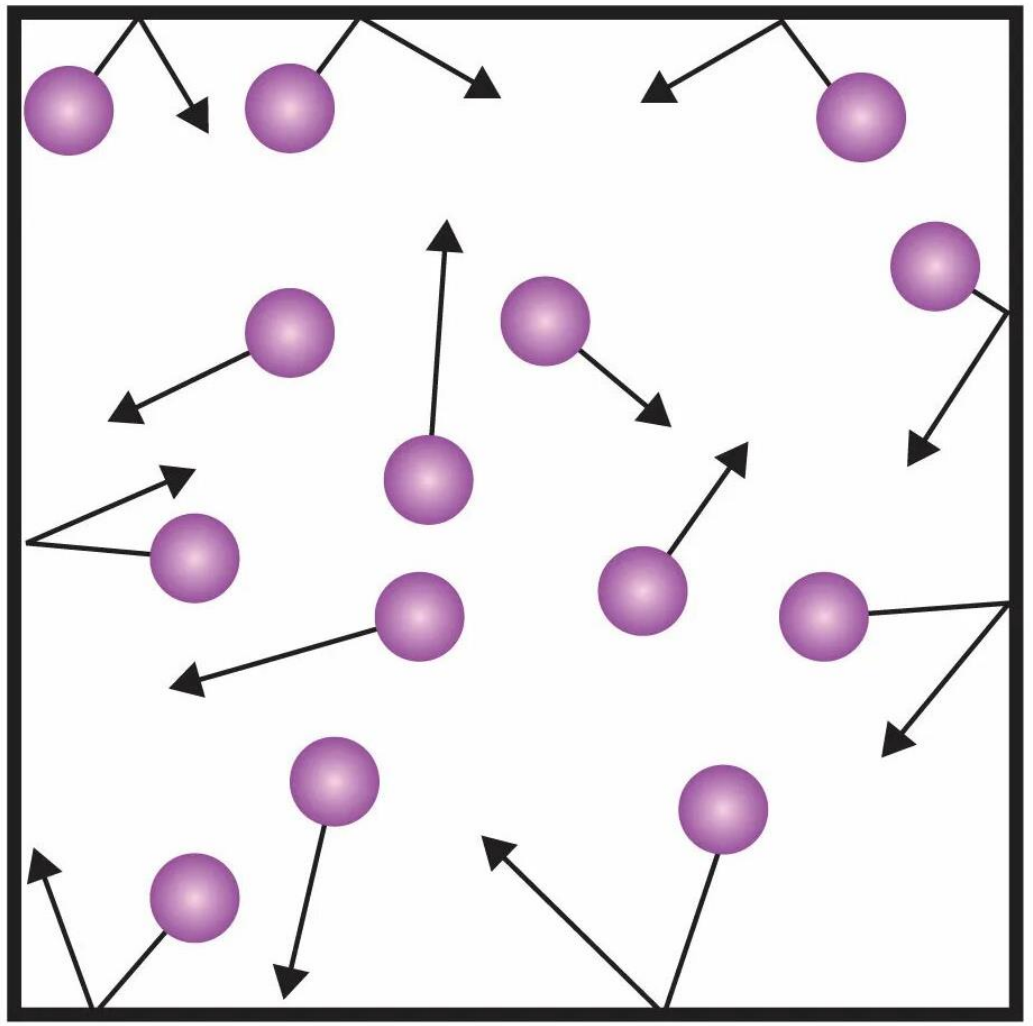
Почему нам так важна полная точность?

Мысленный эксперимент с молекулами

- В наших руках горсть молекул.

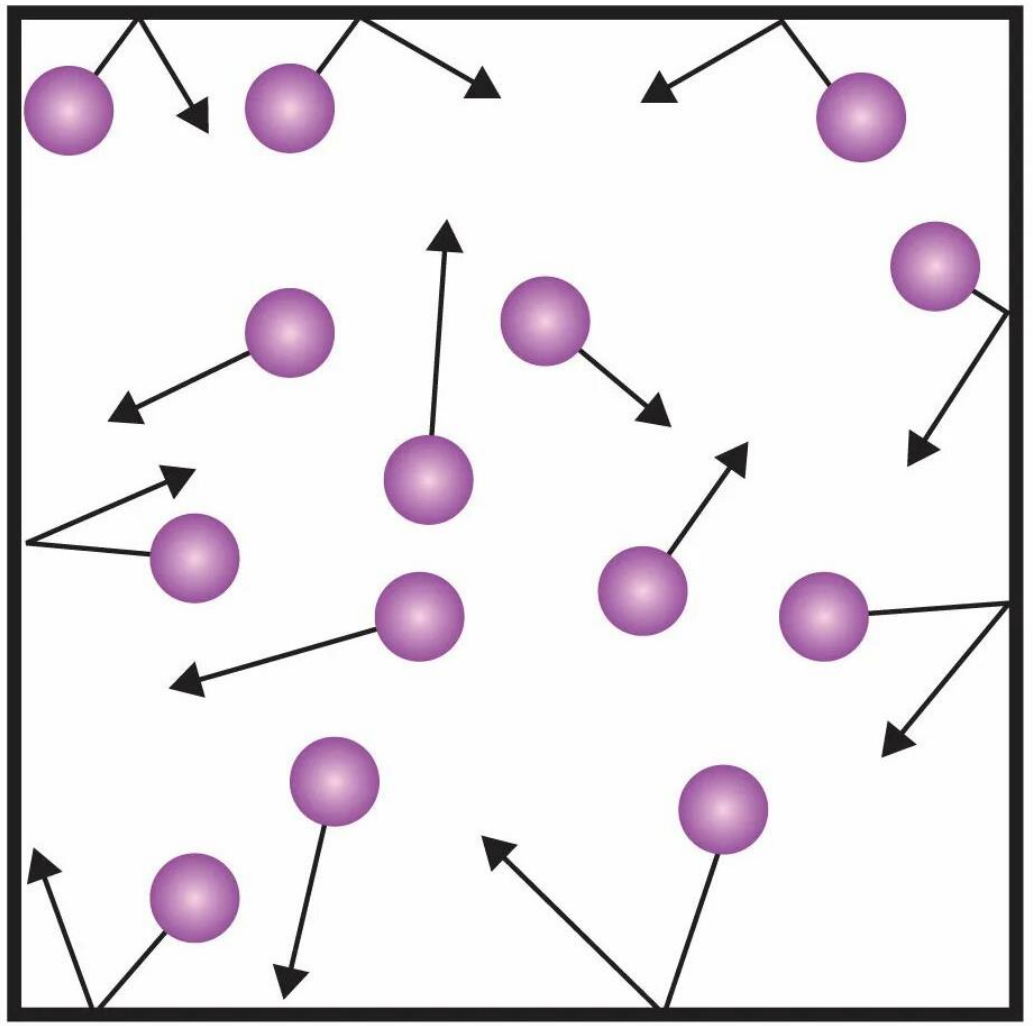
Мысленный эксперимент с молекулами

- В наших руках горсть молекул.
- Мы знаем координаты и скорость каждой.



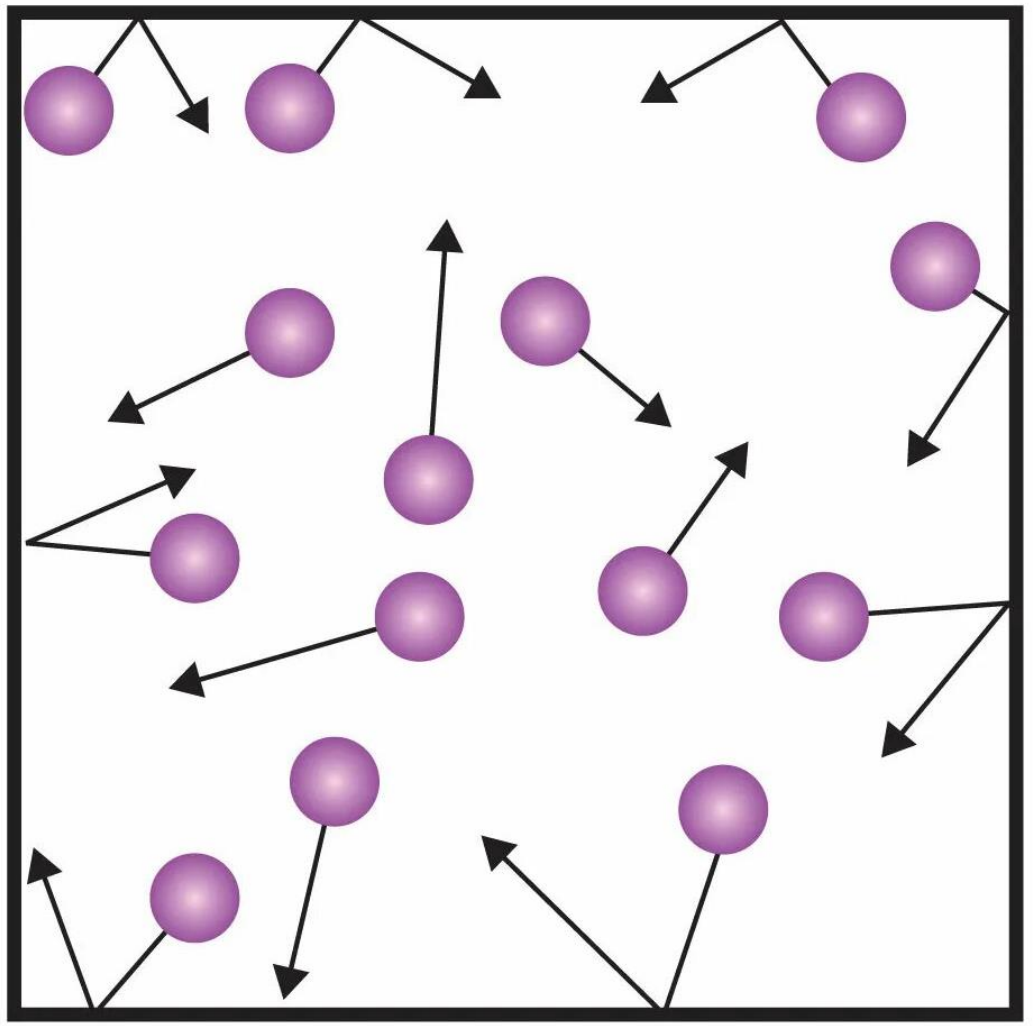
Мысленный эксперимент с молекулами

- В наших руках горсть молекул.
- Мы знаем координаты и скорость каждой.
- **Мы забыли об одной молекуле.**



Мысленный эксперимент с молекулами

- В наших руках горсть молекул.
- Мы знаем координаты и скорость каждой.
- Мы забыли об одной молекуле.
- **Мы теряем знания о молекулах, с которыми она сталкивается.**

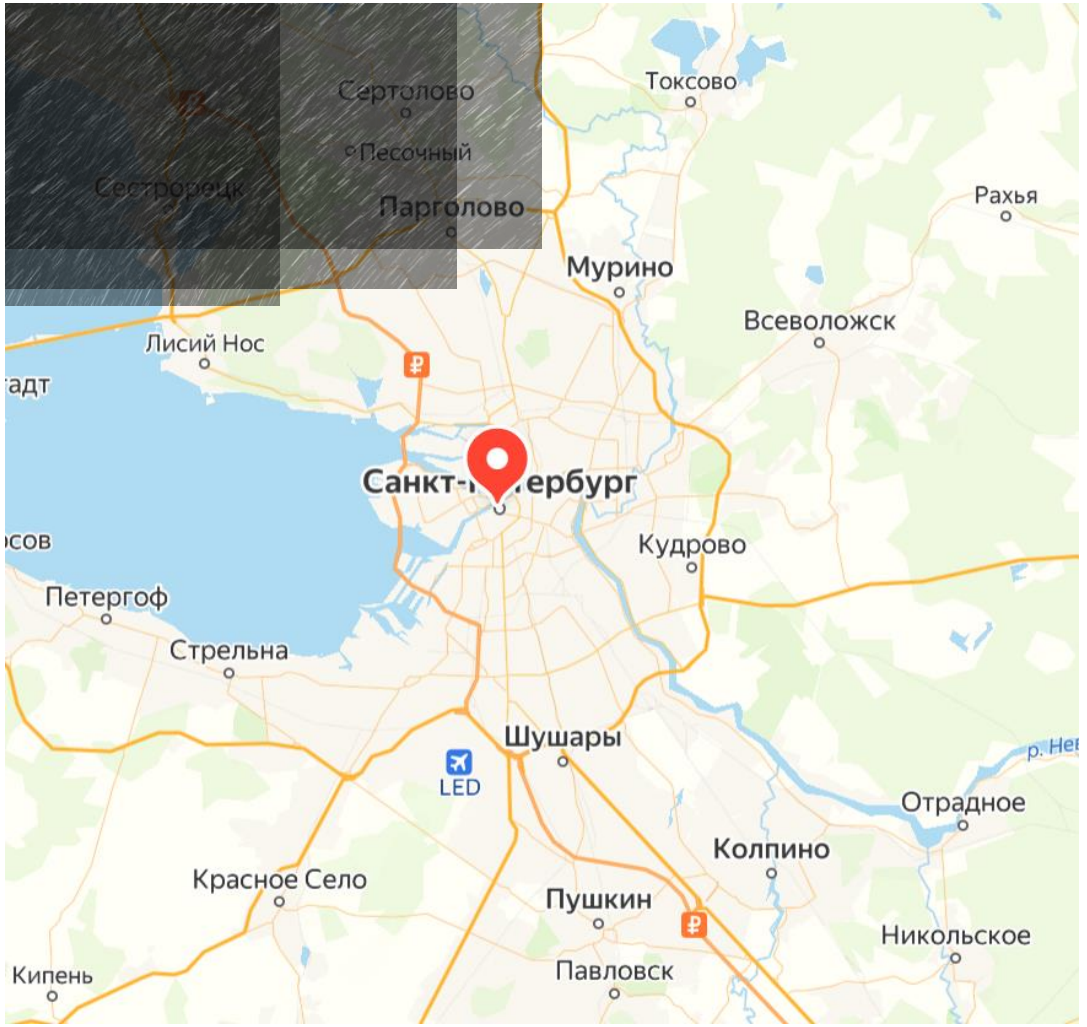


Мысленный эксперимент с молекулами

- В наших руках горсть молекул.
- Мы знаем координаты и скорость каждой.
- Мы забыли об одной молекуле.
- Мы теряем знания о молекулах, с которыми она сталкивается.
- **10^{-9} секунды – и мы не знаем о наших молекулах вообще ничего.**

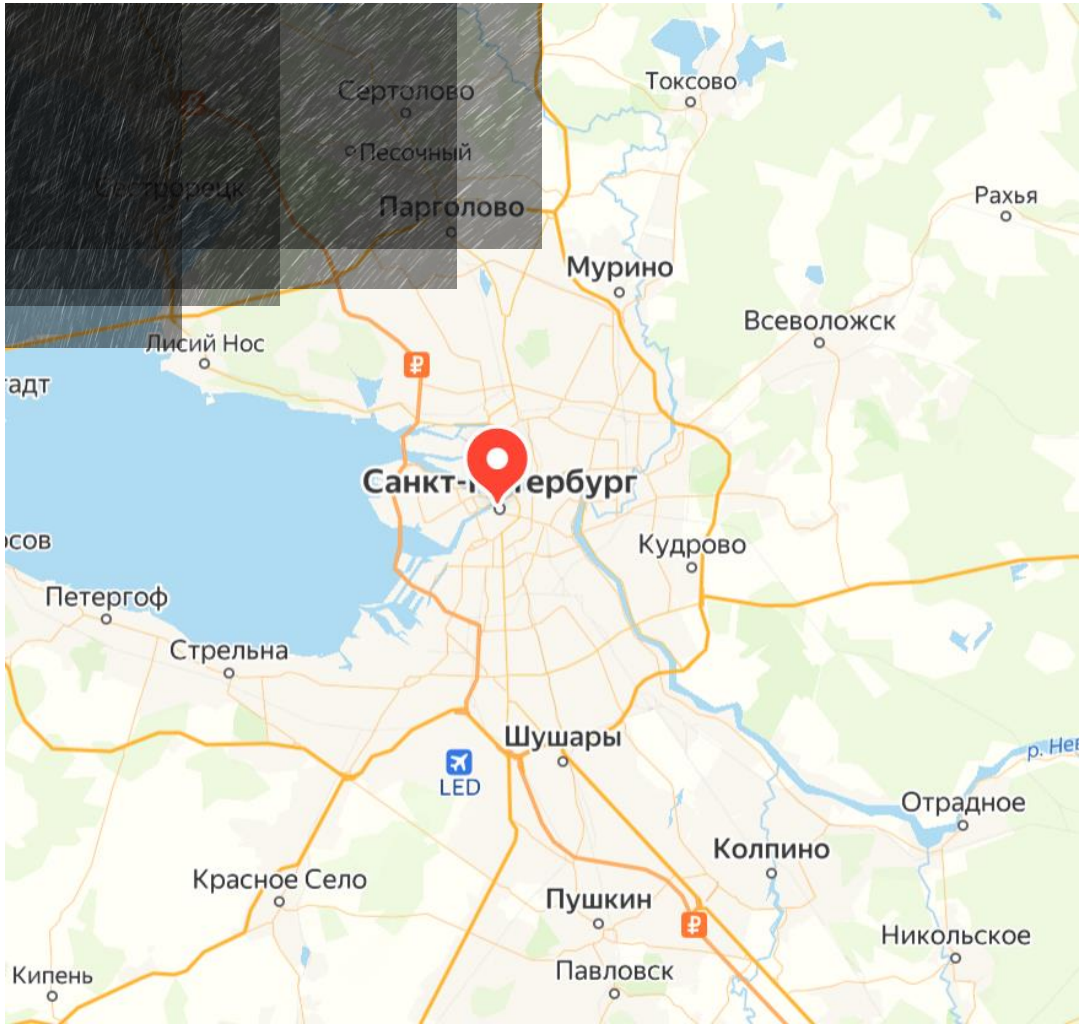
Моделирование погоды

- Каждый шаг моделирования погодных явлений опирается на предыдущий.



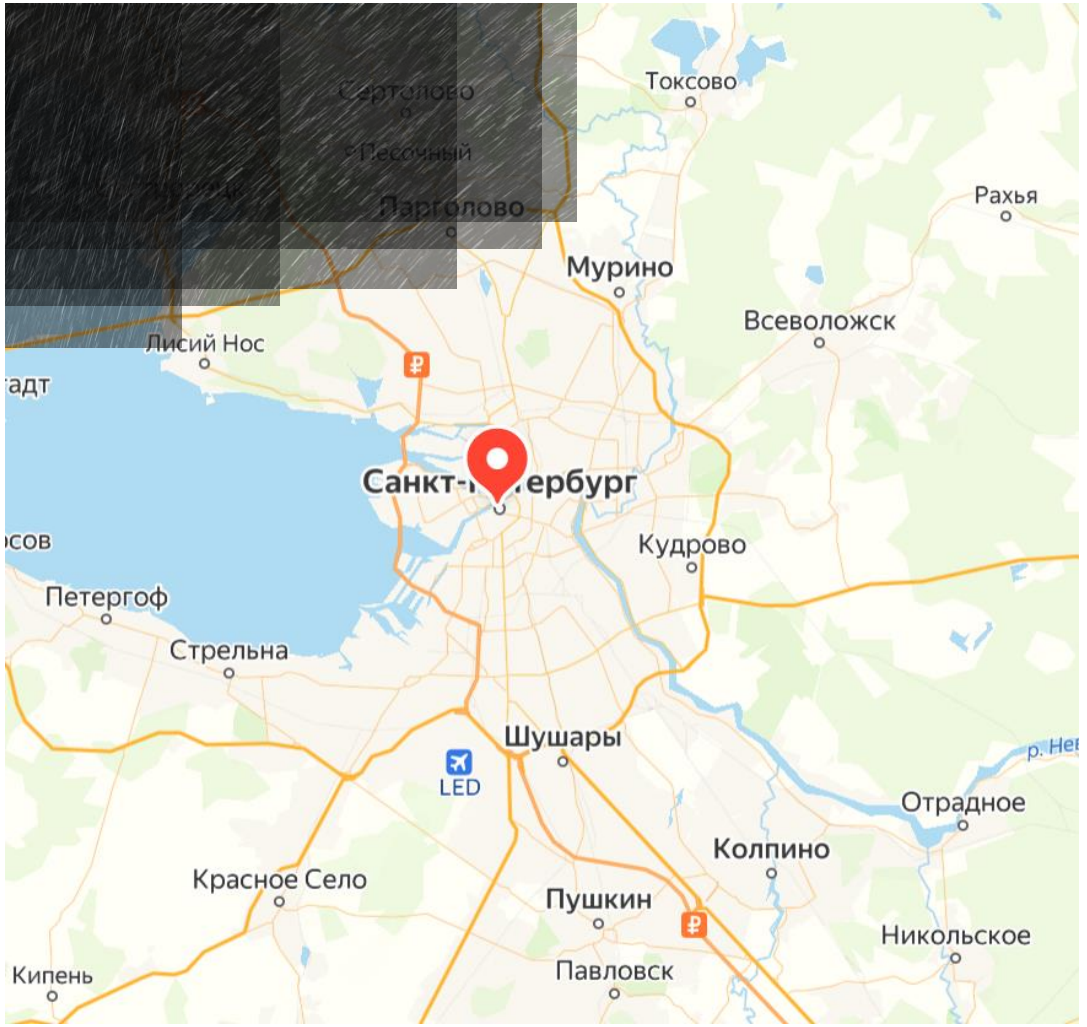
Моделирование погоды

- Каждый шаг моделирования погодных явлений опирается на предыдущий.



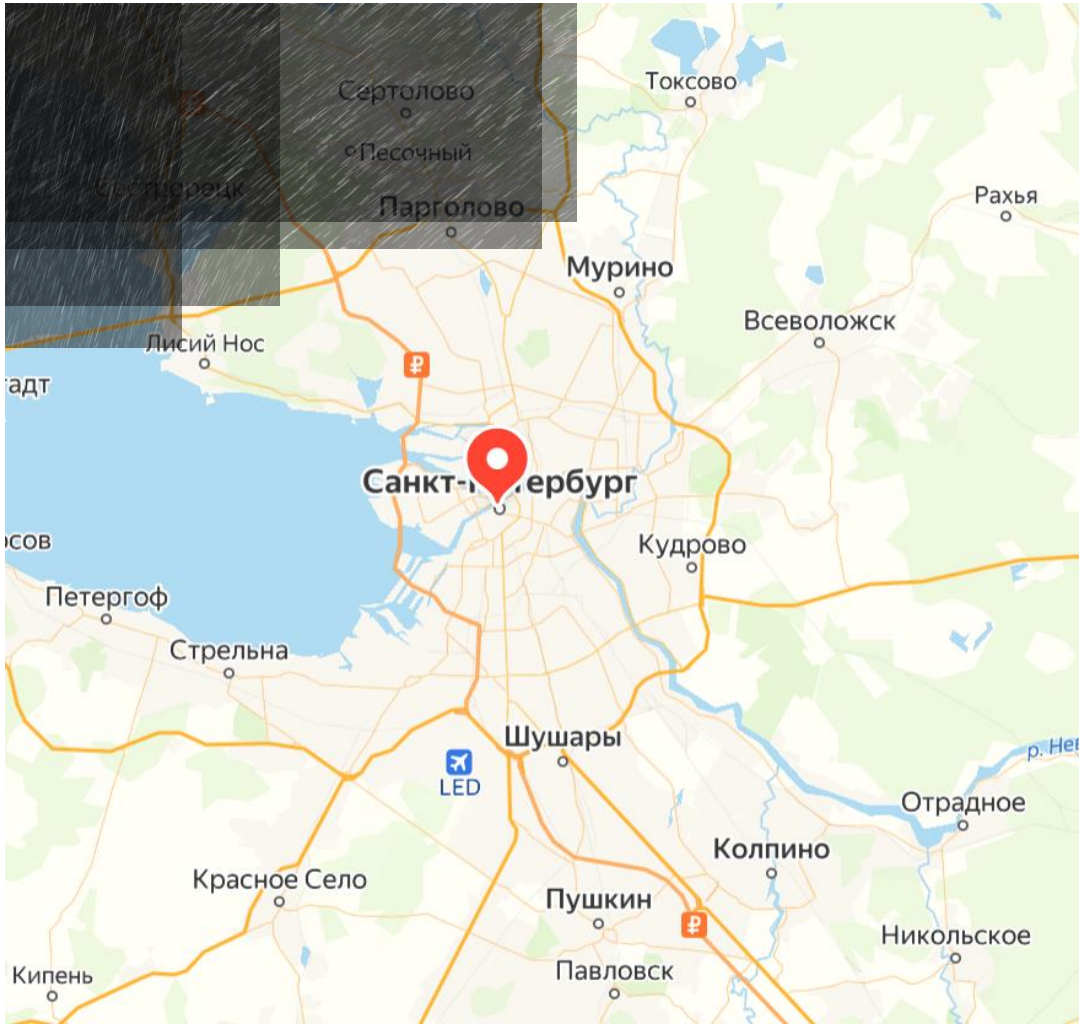
Моделирование погоды

- Каждый шаг моделирования погодных явлений опирается на предыдущий.



Моделирование погоды

- Каждый шаг моделирования погодных явлений опирается на предыдущий.
- **Если мы ошибаемся хотя бы на одном шаге**



Моделирование погоды

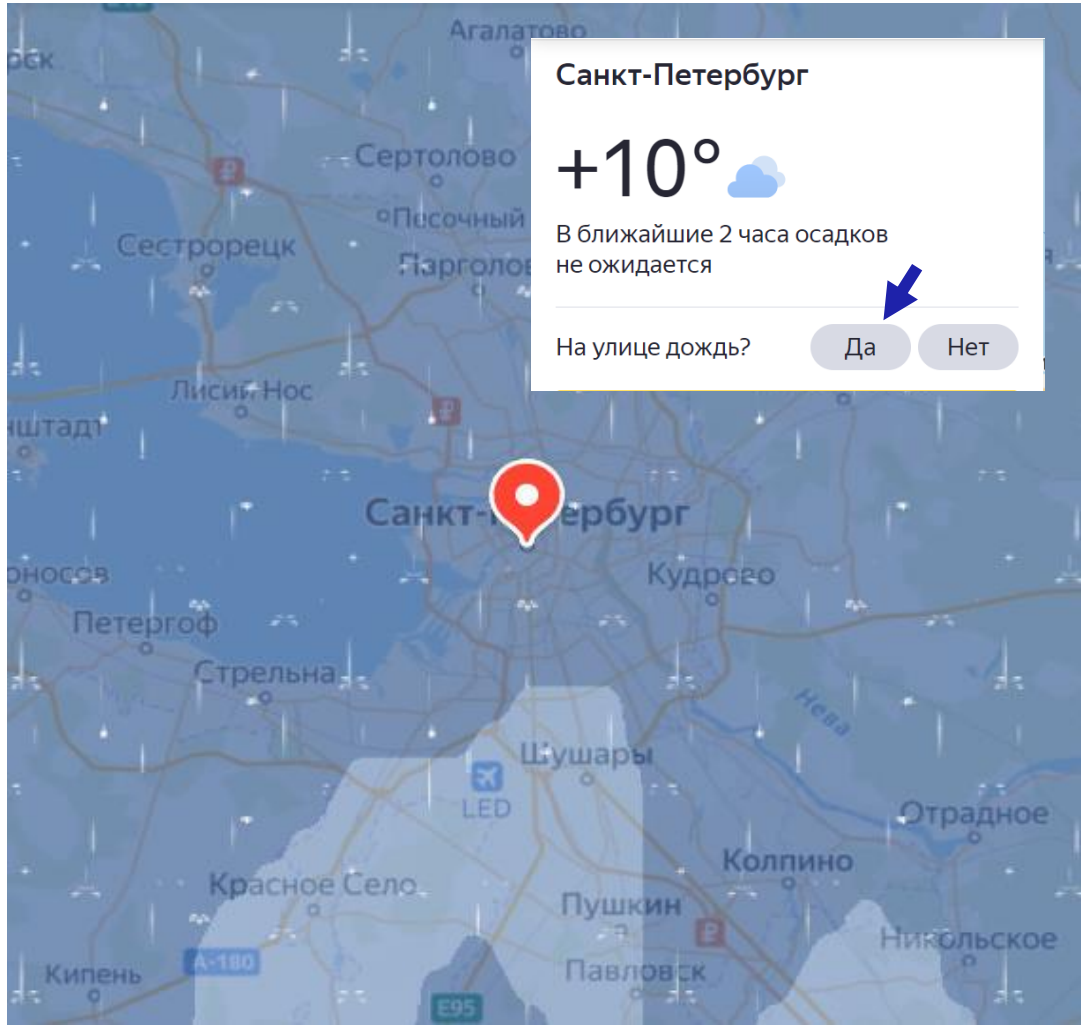
- Каждый шаг моделирования погодных явлений опирается на предыдущий.
- **Если мы ошибаемся хотя бы на одном шаге, ошибка начинает быстро нарастать.**

Моделирование погоды

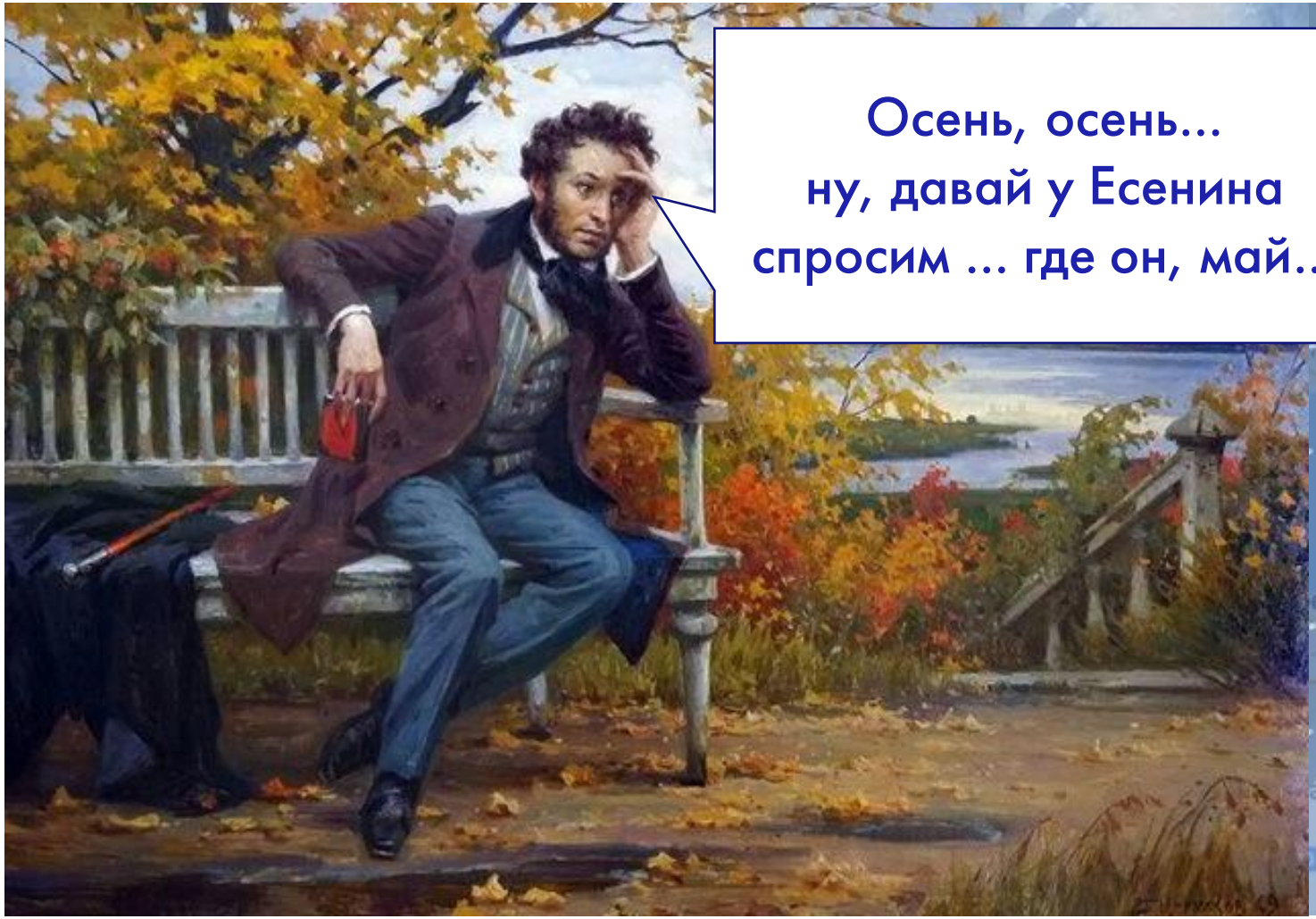
- Каждый шаг моделирования погодных явлений опирается на предыдущий.
- **Если мы ошибаемся хотя бы на одном шаге, ошибка начинает быстро нарастать.**

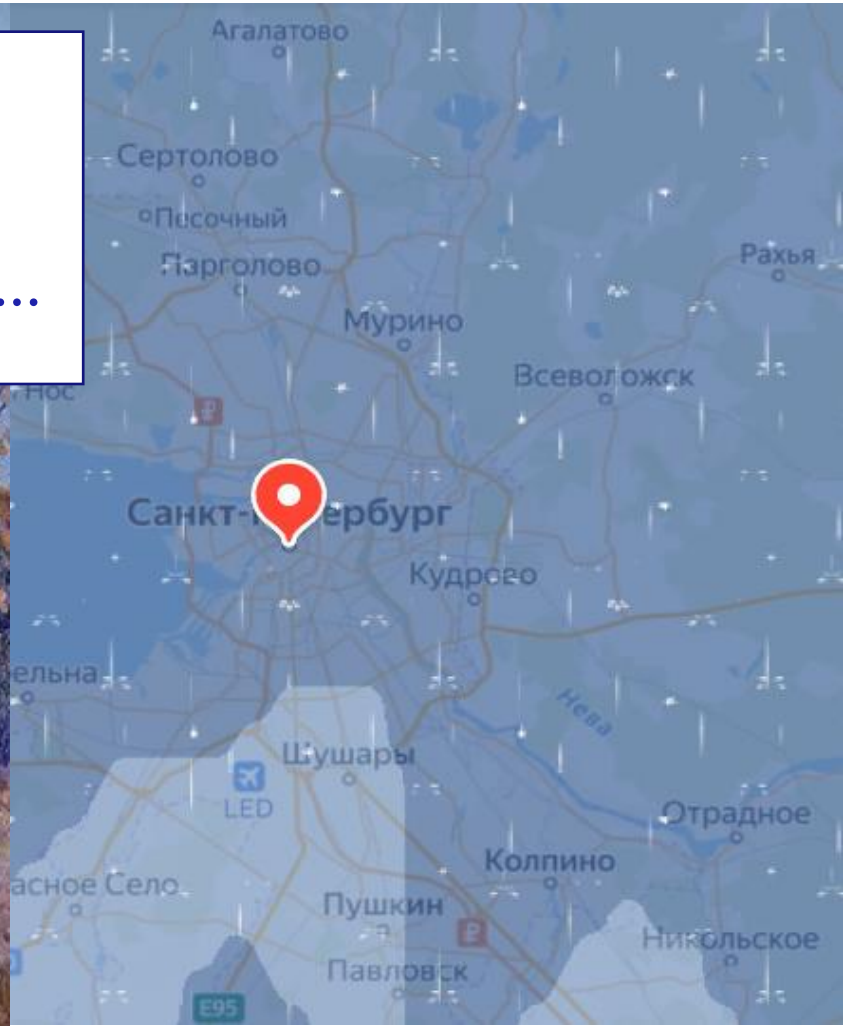
Моделирование погоды

- Каждый шаг моделирования погодных явлений опирается на предыдущий.
- Если мы ошибаемся хотя бы на одном шаге, ошибка начинает быстро нарастать.
- **Точность прогноза погоды сокращается с нескольких дней до часов и минут.**



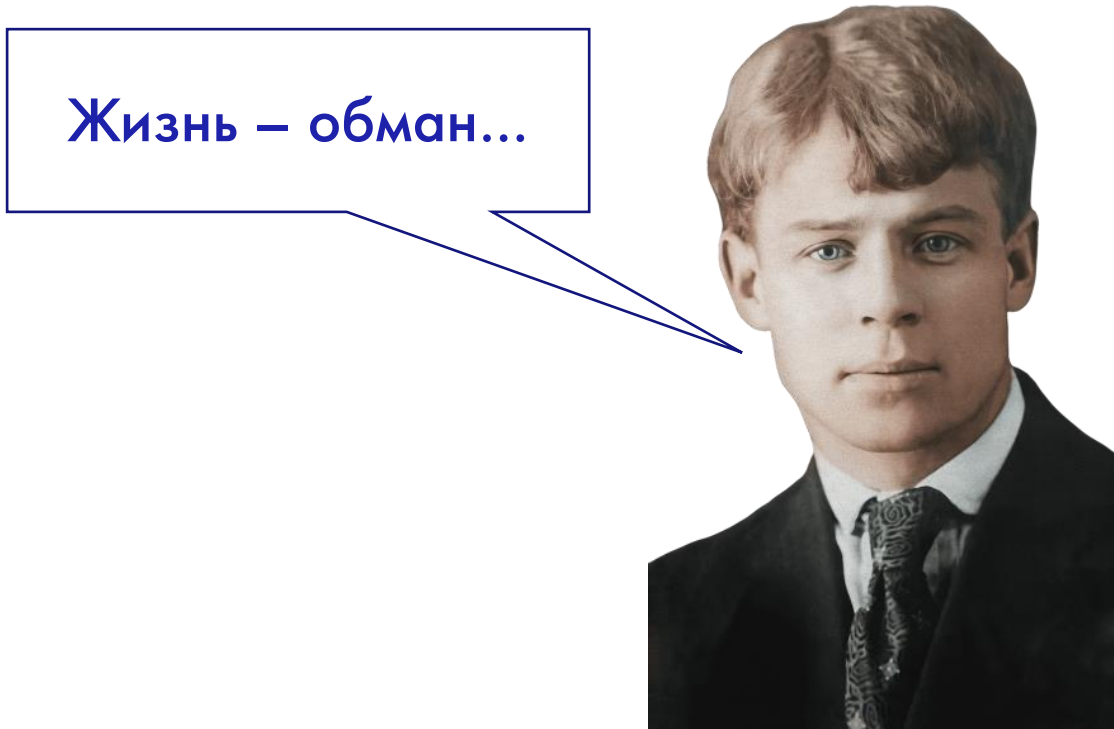
Откуда взяться ошибкам?





Откуда взяться ошибкам?

- Точность любых вычислений конечна и зависит почти от всего.
- Вычислительные ошибки нужно правильно минимизировать.
- Почему это именно так – узнаем позже.

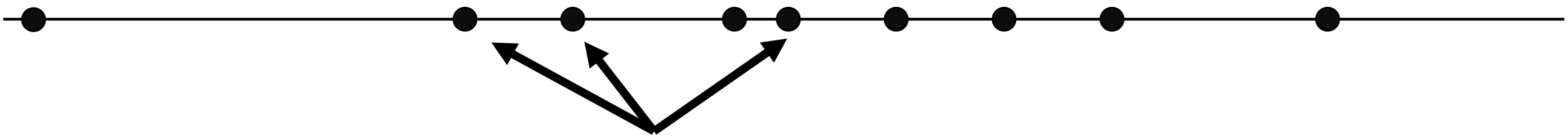


Как обычно проверяют точность и почему это не работает?

Как обычно проверяют точность функций libm?

- Стандартный способ тестирования – сверить результаты вычислений для **предопределенного** набора входных данных с эталонными значениями.

Как обычно проверяют точность функций libm?

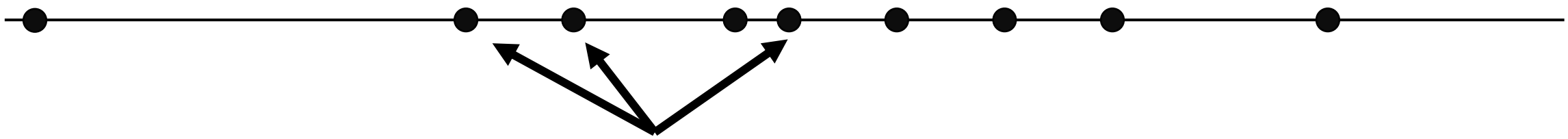


- Стандартный способ тестирования – сверить результаты вычислений для **предопределенного** набора входных данных с эталонными значениями.

Проверили – **десятки тысяч точек**

Как обычно проверяют точность функций libm?

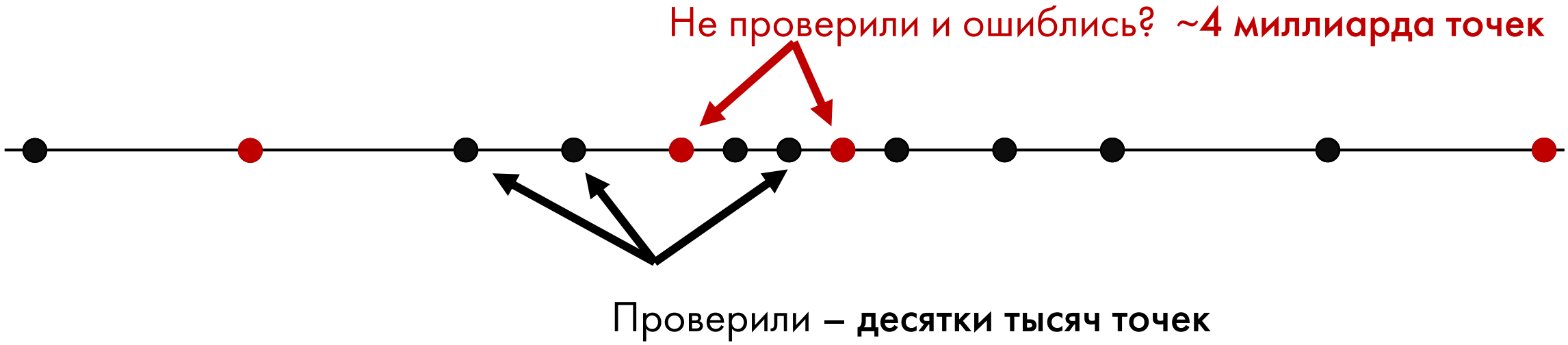
- Стандартный способ тестирования – сверить результаты вычислений для **предопределенного** набора входных данных с эталонными значениями.
- **Вопрос**: может ли ошибка появиться **между** выбранными точками?



Проверили – **десятки тысяч точек**

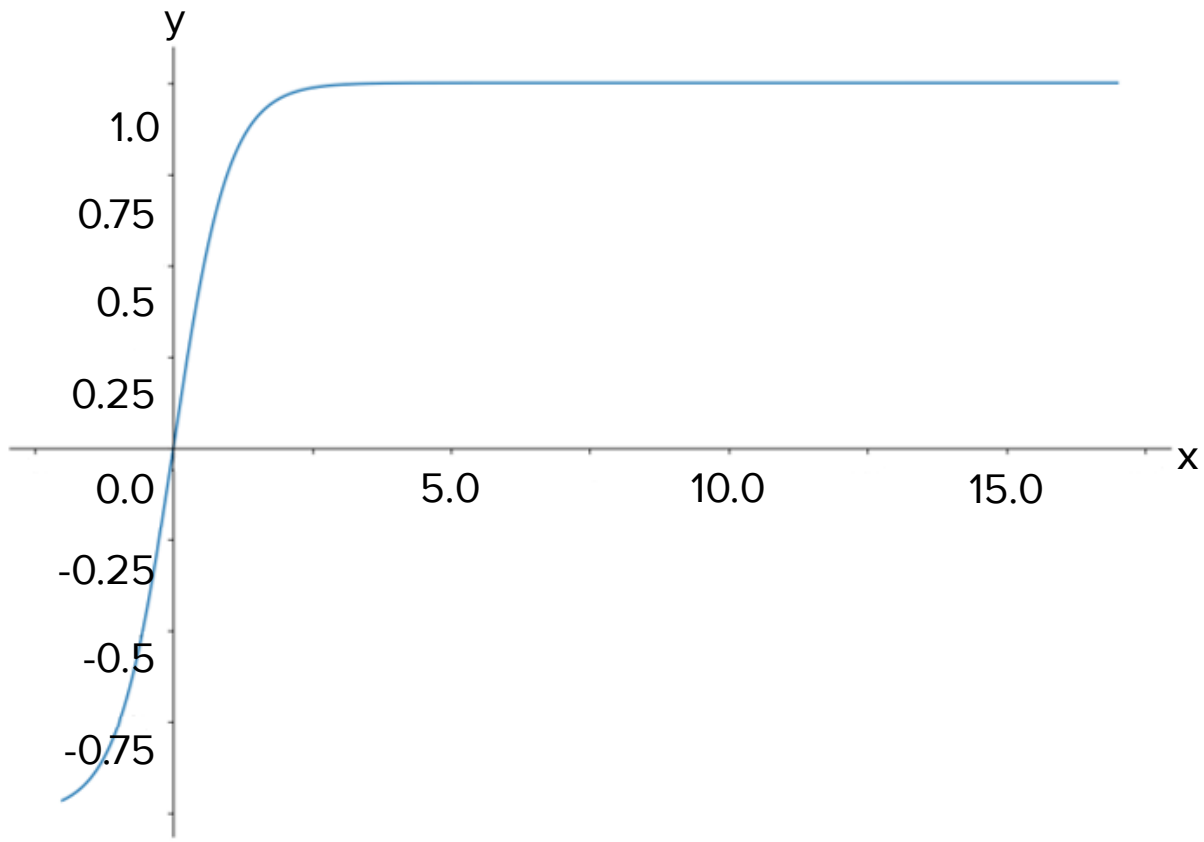
Как обычно проверяют точность функций libm?

- Стандартный способ тестирования – сверить результаты вычислений для **предопределенного** набора входных данных с эталонными значениями.
- **Вопрос**: может ли ошибка появиться **между** выбранными точками?



Проверим точность реализации функций libm из glibc

- Рассмотрим реализацию функции **tanh**, популярной в активационных функциях для AI.

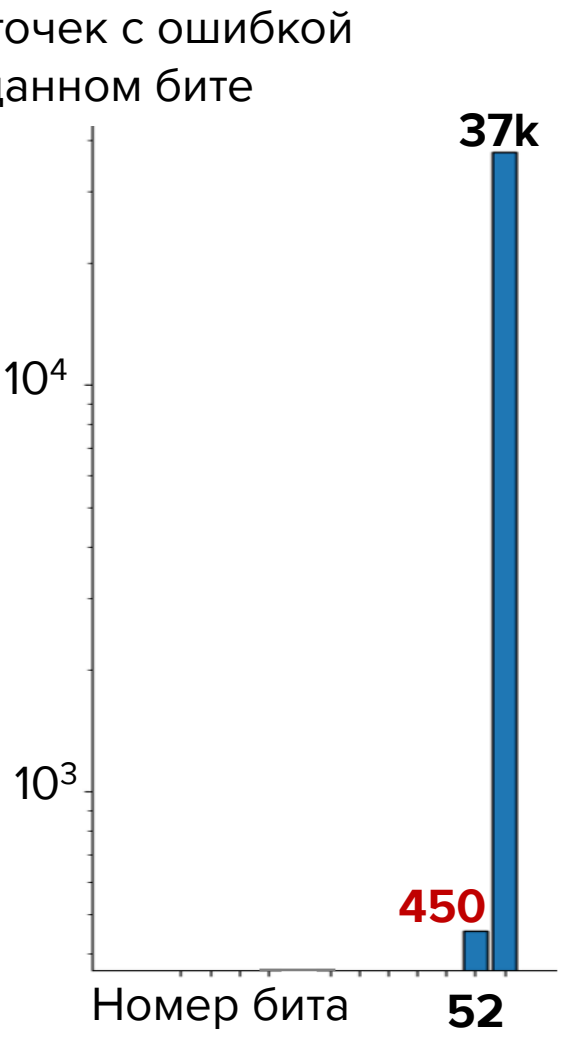


Проверим точность реализации функций libm из glibc

- Рассмотрим реализацию функции **`tanh`**, популярной в активационных функциях для AI.
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.

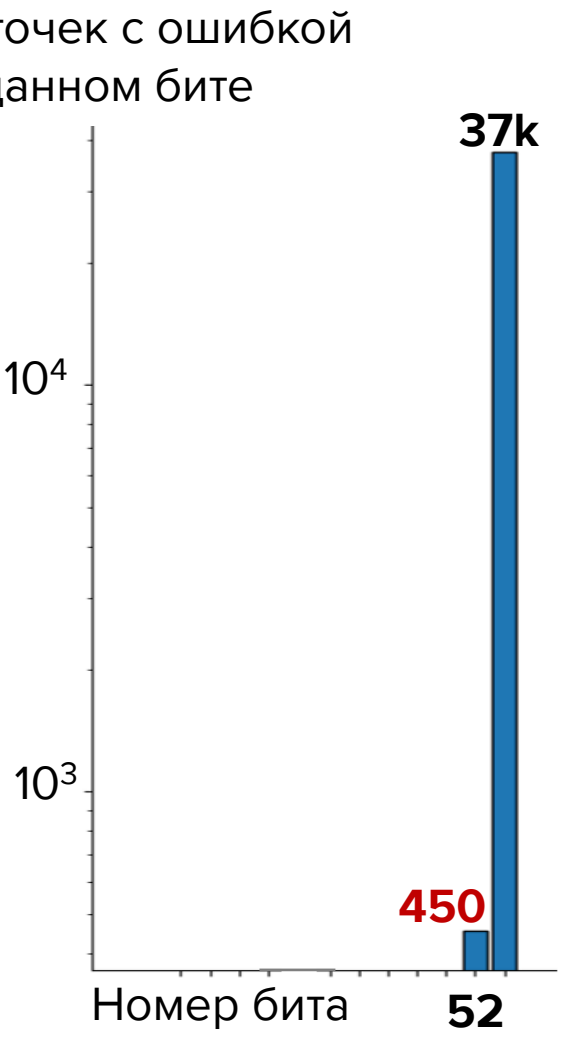
Проверим точность реализации функций libm из glibc

- Рассмотрим реализацию функции **tanh**, популярной в активационных функциях для AI.
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.
- **Что видим**: в тестовой выборке **450** точек имеют ошибку уже в **52** бите.



Проверим точность реализации функций libm из glibc

- Рассмотрим реализацию функции **`tanh`**, популярной в активационных функциях для AI.
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.
- **Что видим**: в тестовой выборке **450** точек имеют ошибку уже в **52** бите.
- **Вывод**: в glibc точность реализации функции `tanh` не удовлетворяет стандарту.



Хорошо, как нам с этим бороться?

Как мы строим алгоритм вычисления функции?

Шаг 1: пристально вглядываемся

- Вспоминаем свойства функции: четность, периодичность, масштабируемость...

Как мы строим алгоритм вычисления функции?

Шаг 1: пристально вглядываемся

- Вспоминаем свойства функции: четность, периодичность, масштабируемость...

Шаг 2: приводим аргумент

- Исходя из свойств функции выбираем наиболее удобный для вычисления диапазон
- Подбираем алгоритм приведения аргумента к выбранному диапазону

Как мы строим алгоритм вычисления функции?

Шаг 1: пристально вглядываемся

- Вспоминаем свойства функции: четность, периодичность, масштабируемость...

Шаг 2: приводим аргумент

- Исходя из свойств функции выбираем наиболее удобный для вычисления диапазон
- Подбираем алгоритм приведения аргумента к выбранному диапазону

Шаг 3: считаем заранее

- Сводим в таблицу вспомогательные числа, которые можно вычислить заранее

Как мы строим алгоритм вычисления функции?

Шаг 1: пристально вглядываемся

- Вспоминаем свойства функции: четность, периодичность, масштабируемость...

Шаг 2: приводим аргумент

- Исходя из свойств функции выбираем наиболее удобный для вычисления диапазон
- Подбираем алгоритм приведения аргумента к выбранному диапазону

Шаг 3: считаем заранее

- Сводим в таблицу вспомогательные числа, которые можно вычислить заранее

Шаг 4: аппроксимируем

- При необходимости разбиваем рабочий диапазон на отрезки
- Подбираем для каждого отрезка полиномиальную аппроксимацию

Как мы строим алгоритм вычисления функции?

Шаг 1: пристально вглядываемся

- Вспоминаем свойства функции: четность, периодичность, масштабируемость...

Шаг 2: приводим аргумент

- Исходя из свойств функции выбираем наиболее удобный для вычисления диапазон
- Подбираем алгоритм приведения аргумента к выбранному диапазону

Шаг 3: считаем заранее

- Сводим в таблицу вспомогательные числа, которые можно вычислить заранее

Шаг 4: аппроксимируем

- При необходимости разбиваем рабочий диапазон на отрезки
- Подбираем для каждого отрезка полиномиальную аппроксимацию

Шаг 5: восстанавливаем результат

- Проделываем приведение аргумента в обратном порядке

Как мы строим алгоритм вычисления функции?

Шаг 1: пристально вглядываемся

- Вспоминаем свойства функции: четность, периодичность, масштабируемость...

Шаг 2: приводим аргумент

- Исходя из свойств функции выбираем наиболее удобный для вычисления диапазон
- Подбираем алгоритм приведения аргумента к выбранному диапазону

Шаг 3: считаем заранее

- Сводим в таблицу вспомогательные числа, которые можно вычислить заранее

Шаг 4: аппроксимируем

- При необходимости разбиваем рабочий диапазон на отрезки
- Подбираем для каждого отрезка полиномиальную аппроксимацию

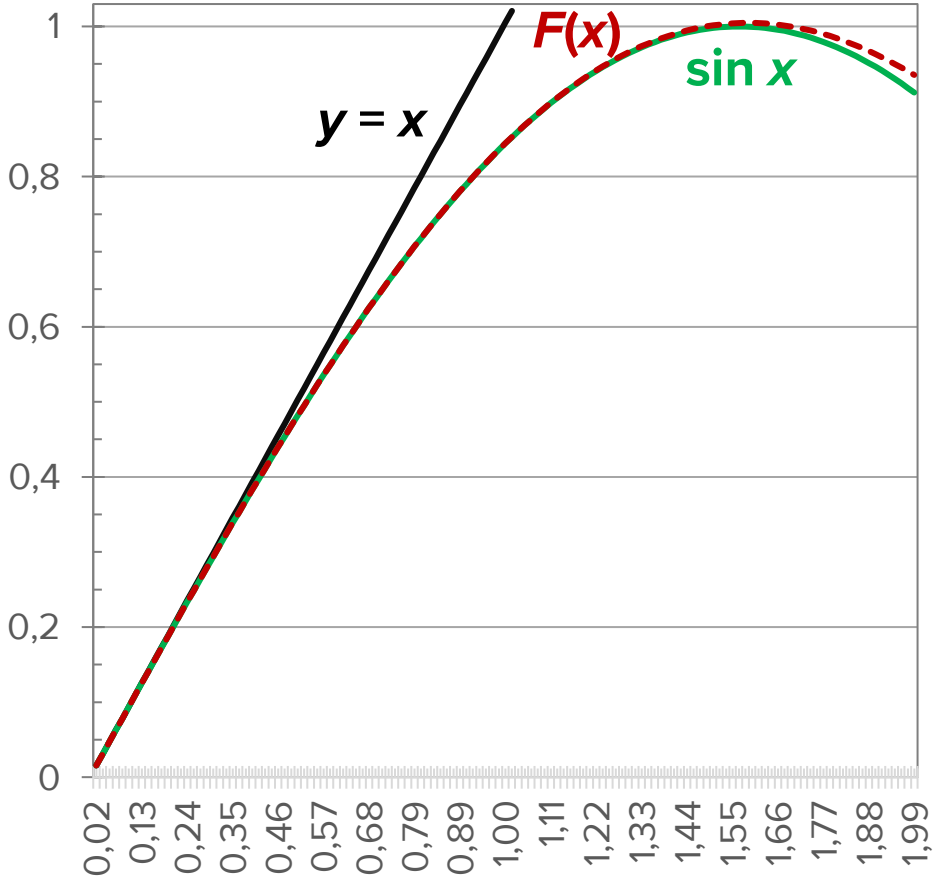
Корень ~~зла~~ численной неустойчивости

Шаг 5: восстанавливаем результат

- Проделываем приведение аргумента в обратном порядке

Простейший способ: ряд Тейлора

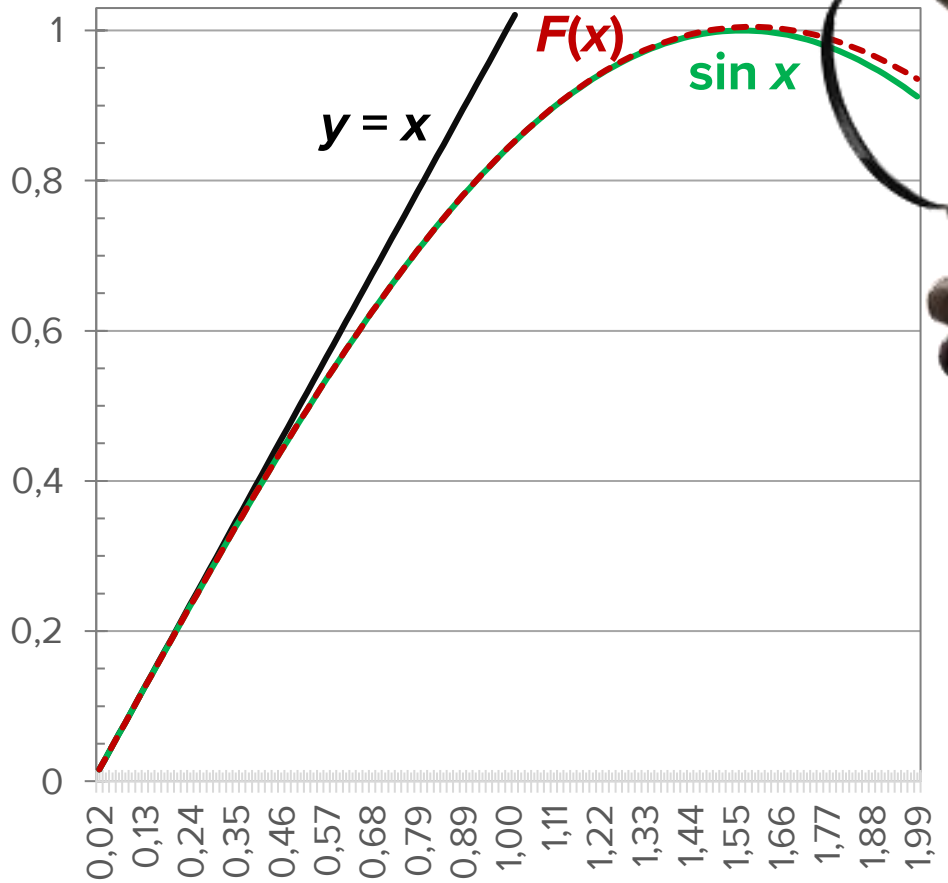
$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} = F(x)$$



ВЫБОР СПОСОБА ПОЛИНОМИАЛЬНОЙ АППРОКСИМАЦИИ

Простейший способ: ряд Тейлора

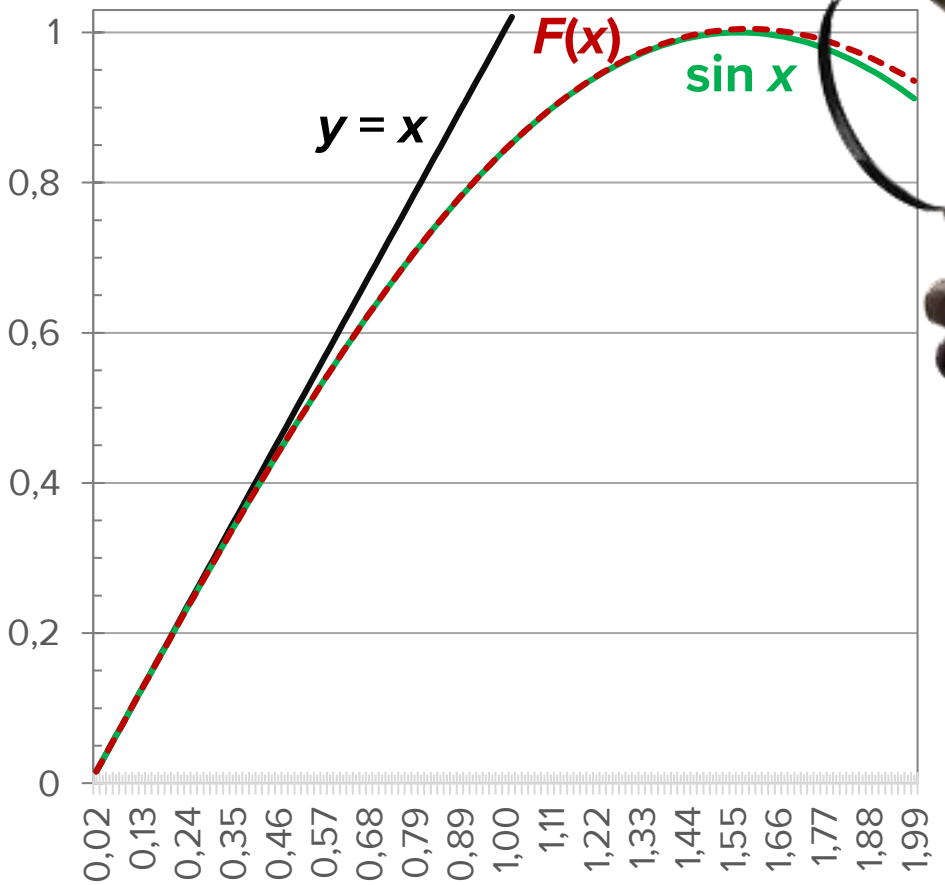
$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} = F(x)$$

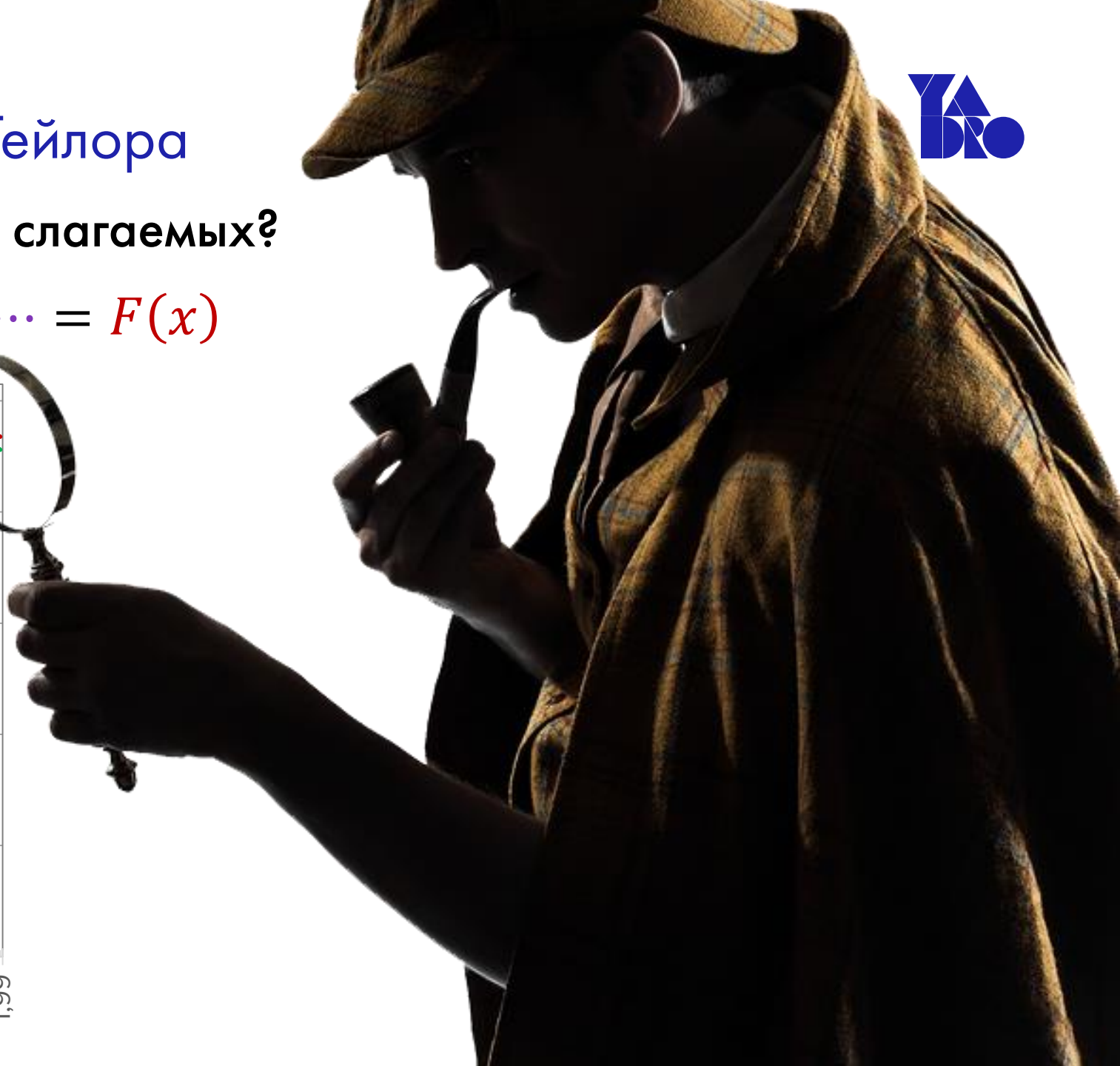


Простейший способ: ряд Тейлора

Спасет ли нас большее число слагаемых?

$$\sin x \approx x - {x^3}/{3!} + {x^5}/{5!} + \cdots = F(x)$$

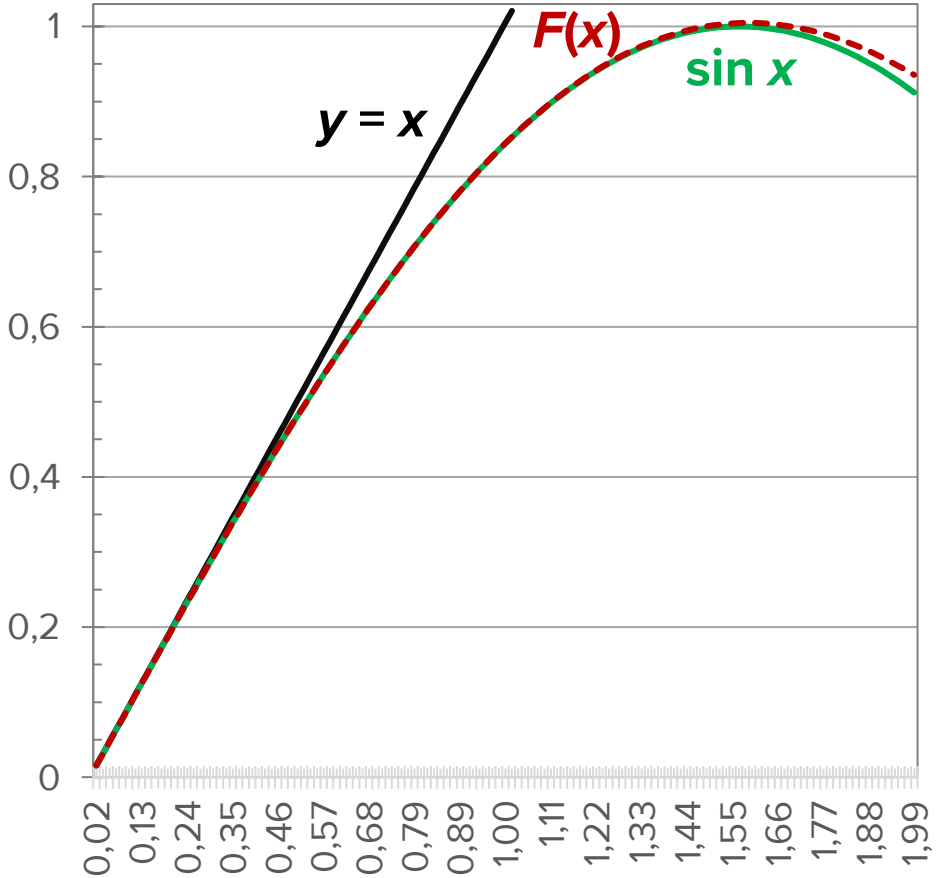




Простейший способ: ряд Тейлора

Спасет ли нас большее число слагаемых?

$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} + \cdots = F(x)$$

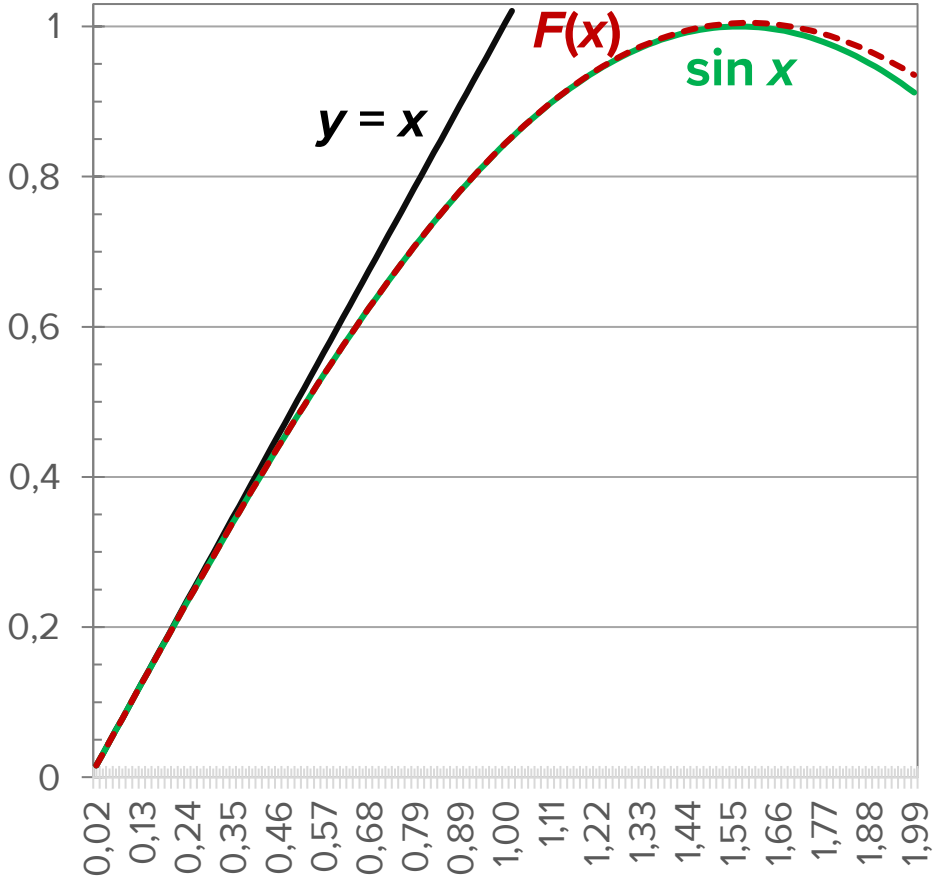


Чем больше слагаемых,
тем больше арифметических операций

Простейший способ: ряд Тейлора

Спасет ли нас большее число слагаемых?

$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} + \cdots = F(x)$$



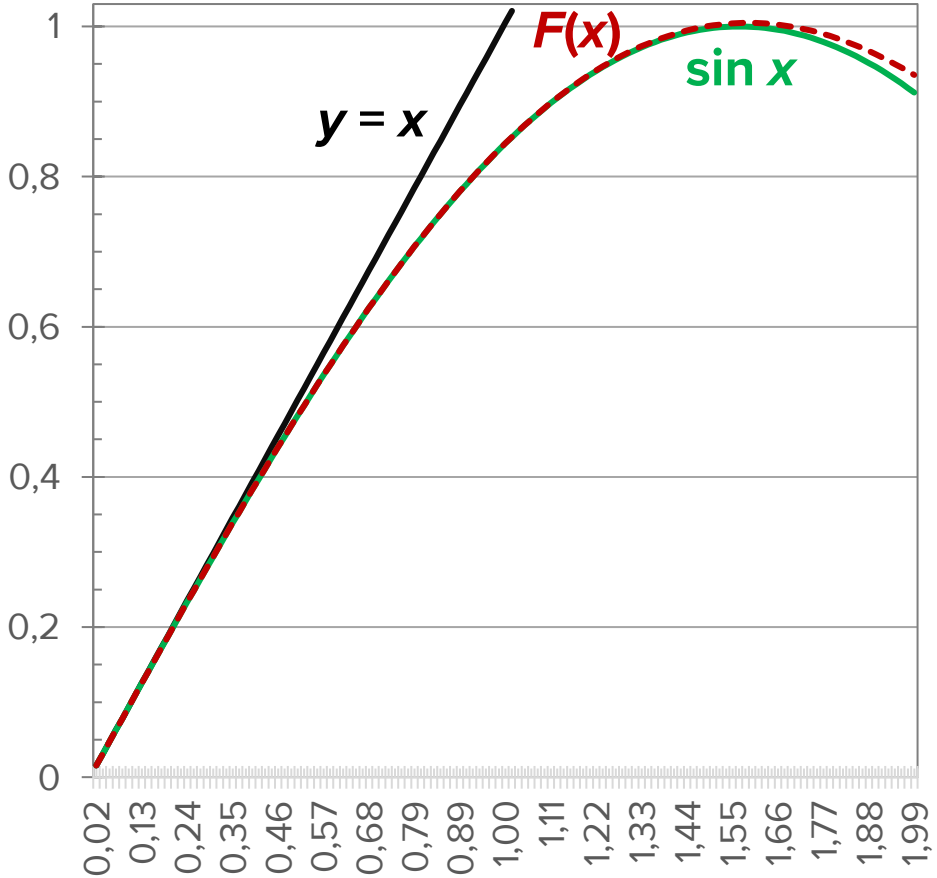
Чем больше слагаемых,
тем больше арифметических операций, а значит:

☹ ниже производительность

Простейший способ: ряд Тейлора

Спасет ли нас большее число слагаемых?

$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} + \cdots = F(x)$$



Чем больше слагаемых,
тем больше арифметических операций, а значит:

- ☹ ниже производительность;
- ☹ ниже численная устойчивость.

Лирическое отступление о численной устойчивости

Точность не бесконечна

Отображение бесконечного множества действительных чисел в числа, представимые конечным числом бит:

```
float(0.1) = 0.1000000015
```

Floating Point распределены неравномерно

Операции над сильно отличающимися по модулю числами менее точные.

Округление после каждой операции

В процессе вычислений накапливается ошибка.

Законы арифметики не работают

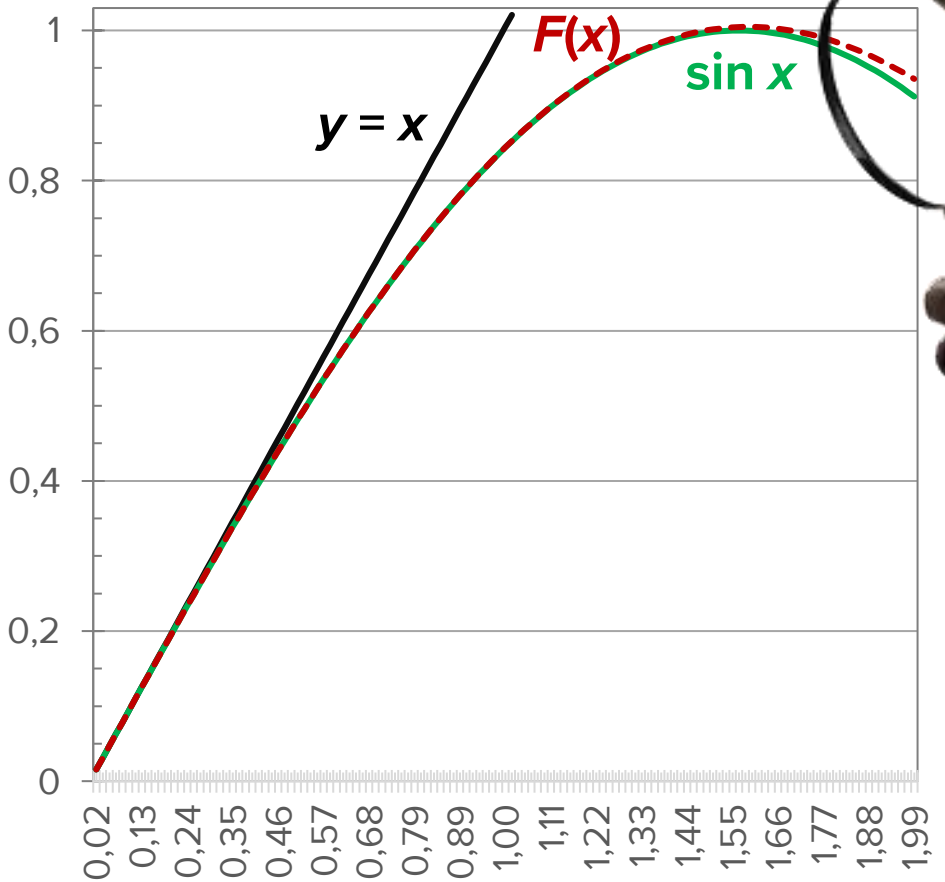
Нет ассоциативности: $x + (y + z) \neq (x + y) + z$

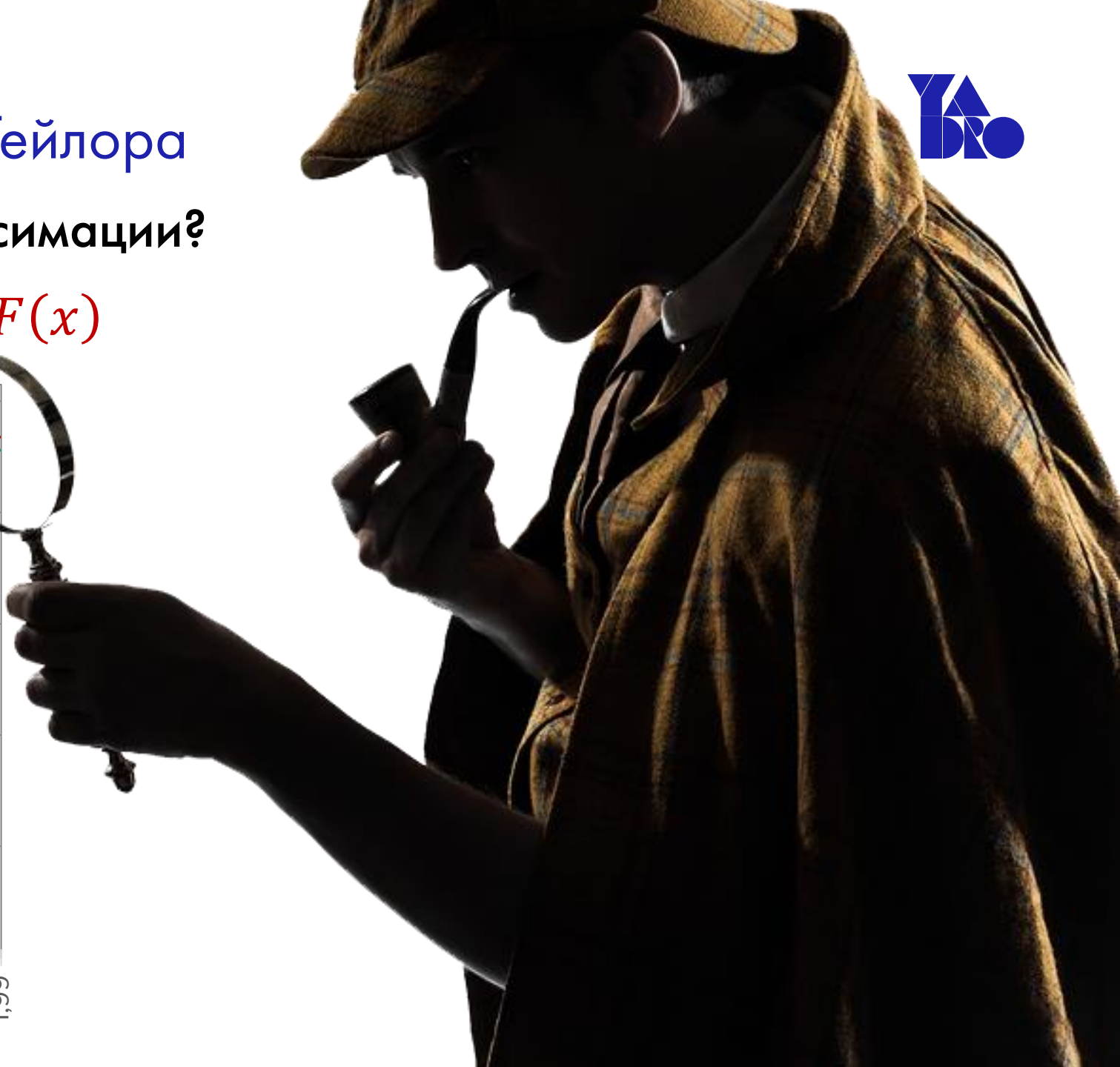
Нет дистрибутивности: $x \cdot (y + z) \neq x \cdot y + x \cdot z$

Простейший способ: ряд Тейлора

Как выглядит ошибка аппроксимации?

$$\sin x \approx x - {x^3}/{3!} + {x^5}/{5!} = F(x)$$



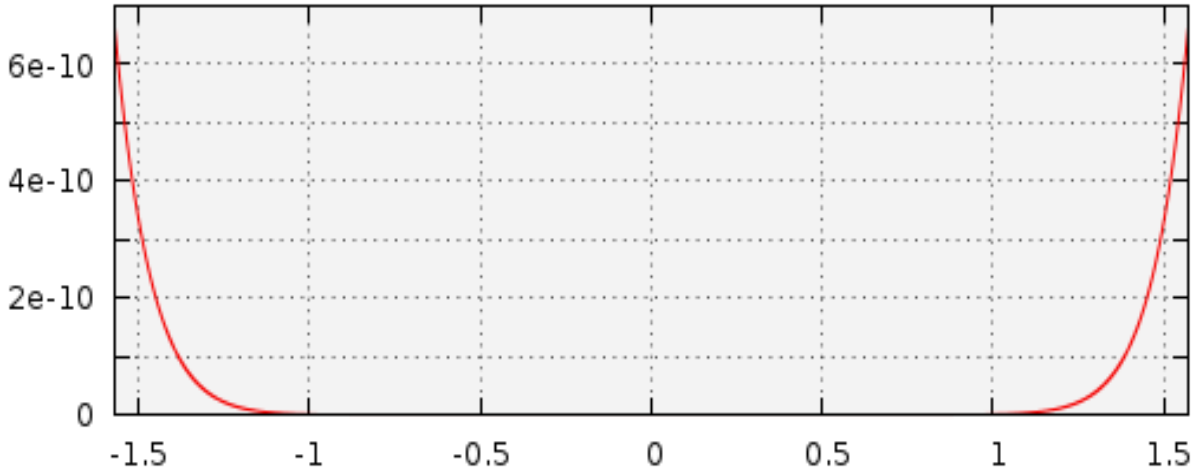


Простейший способ: ряд Тейлора

Как выглядит ошибка аппроксимации?

$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} = F(x)$$

$$|\sin x - F(x)|$$

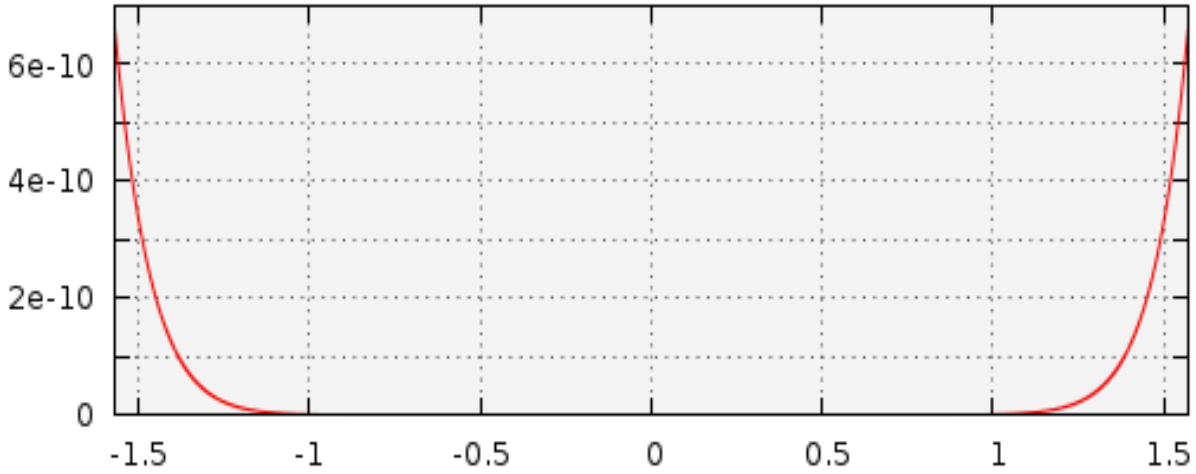


Простейший способ: ряд Тейлора

Как выглядит ошибка аппроксимации?

$$\sin x \approx x - {}^{x^3}/_{3!} + {}^{x^5}/_{5!} = F(x)$$

$$|\sin x - F(x)|$$



Ошибка мала вблизи точки разложения, но не на границах отрезка!

☹ Ряд Тейлора не подходит.

Оптимальный способ: минимаксная аппроксимация

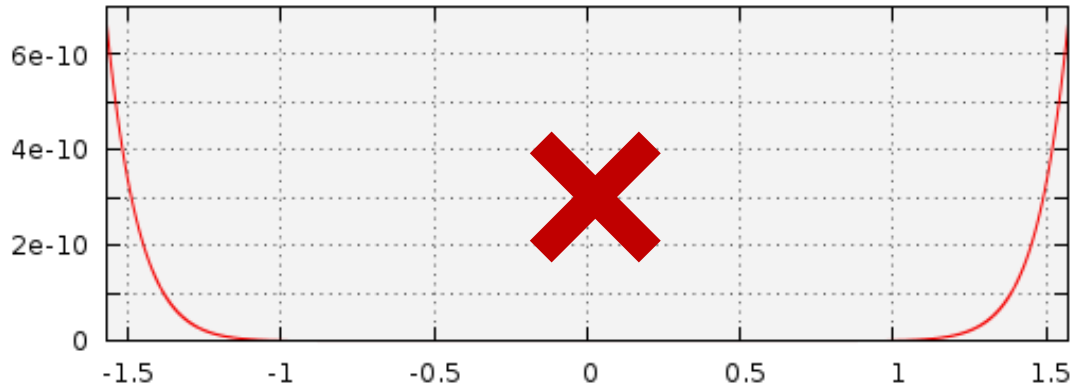
- Коэффициенты полиномиальной аппроксимации рассчитываются из **условия минимизации максимальной абсолютной погрешности во всем интервале аппроксимации**.

Оптимальный способ: минимаксная аппроксимация

- Коэффициенты полиномиальной аппроксимации рассчитываются из условия минимизации максимальной абсолютной погрешности во всем интервале аппроксимации.
- **Как это сделать?** Алгоритм давно придуман, спасибо Чебышёву, Паде, Ремезу и другим классикам теории аппроксимации.

Как это выглядит на графике ошибки аппроксимации?

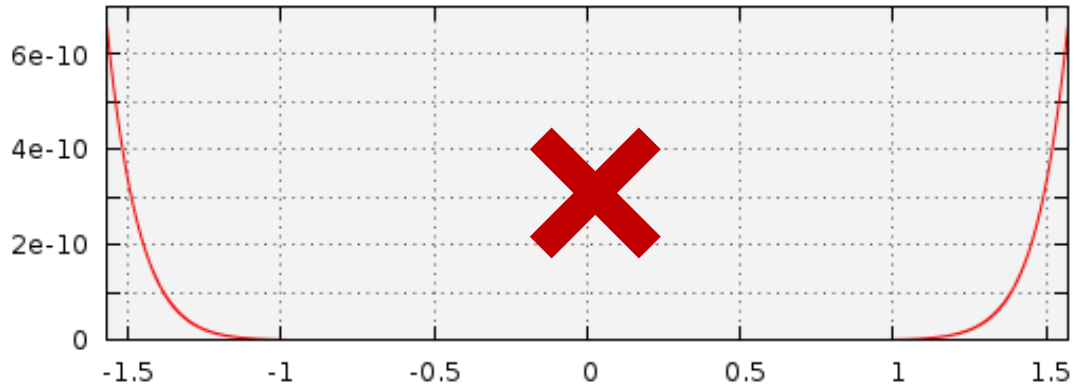
$$|\sin x - F(x)|$$



Та самая аппроксимация синуса рядом Тейлора до 5-го порядка

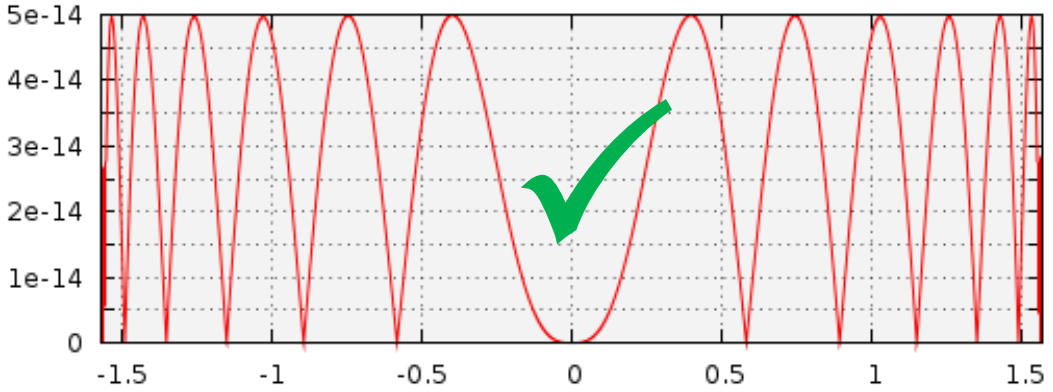
Как это выглядит на графике ошибки аппроксимации?

$|\sin x - F(x)|$



Та самая аппроксимация синуса рядом Тейлора до 5-го порядка

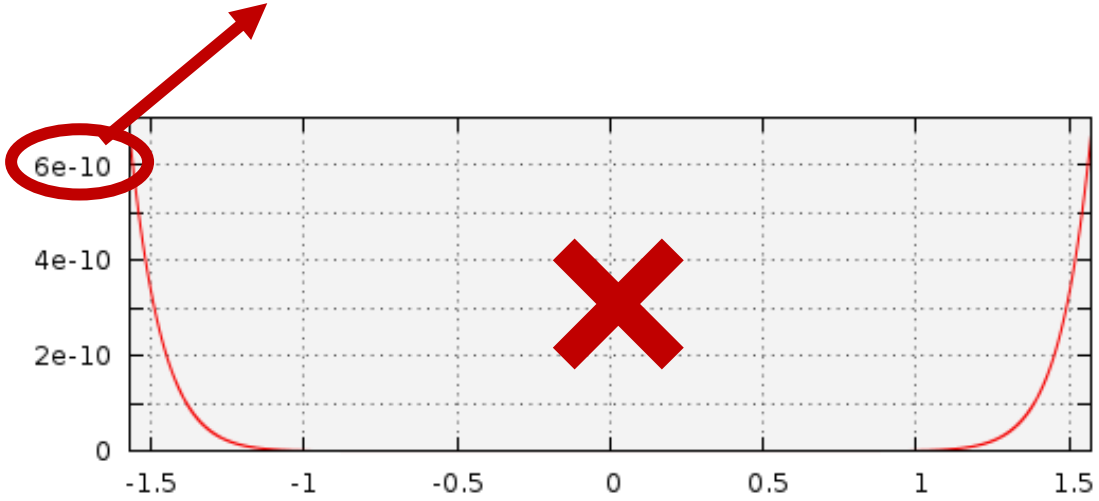
$|\sin x - F_{minimax}(x)|$



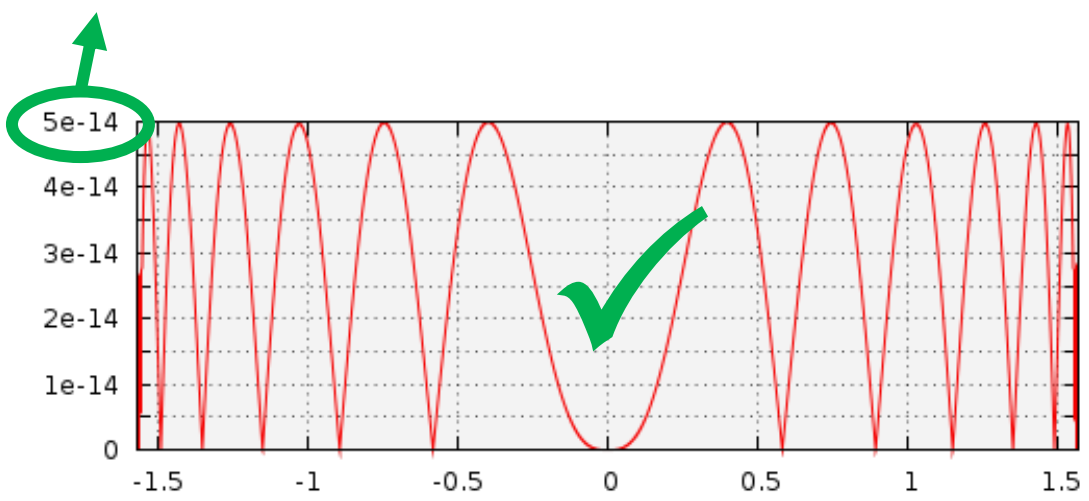
Минимаксная аппроксимация того же порядка

Как это выглядит на графике ошибки аппроксимации?

$$6 \times 10^{-10} > 5 \times 10^{-14}$$



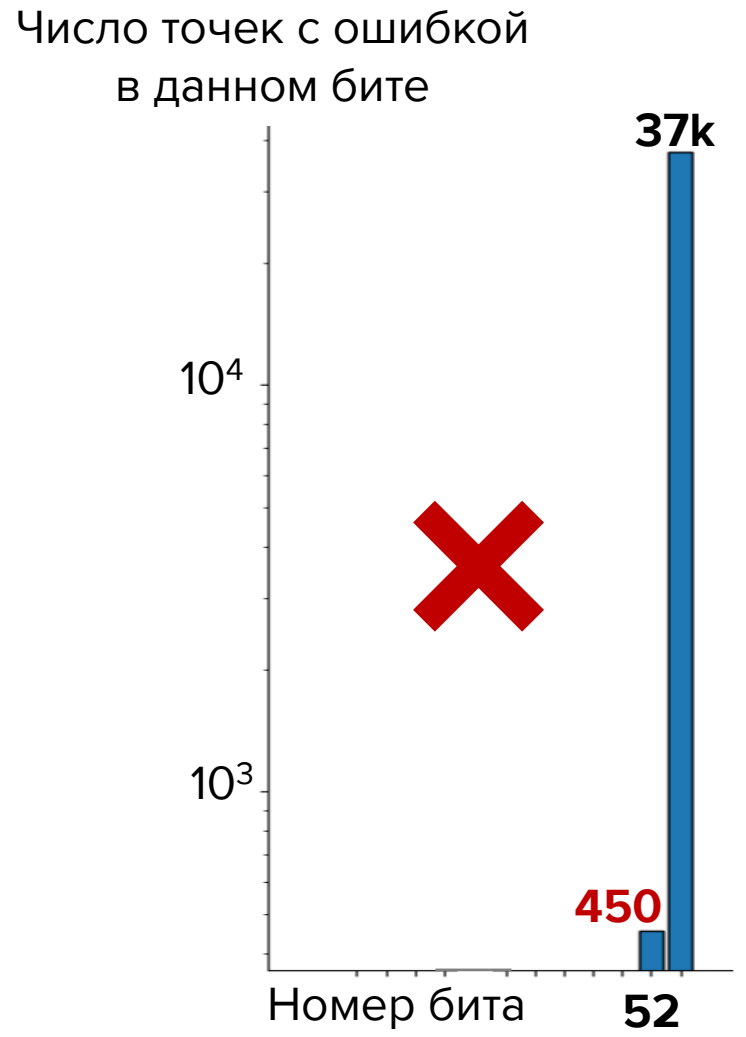
Та самая аппроксимация синуса рядом Тейлора до 5-го порядка



Минимаксная аппроксимация того же порядка

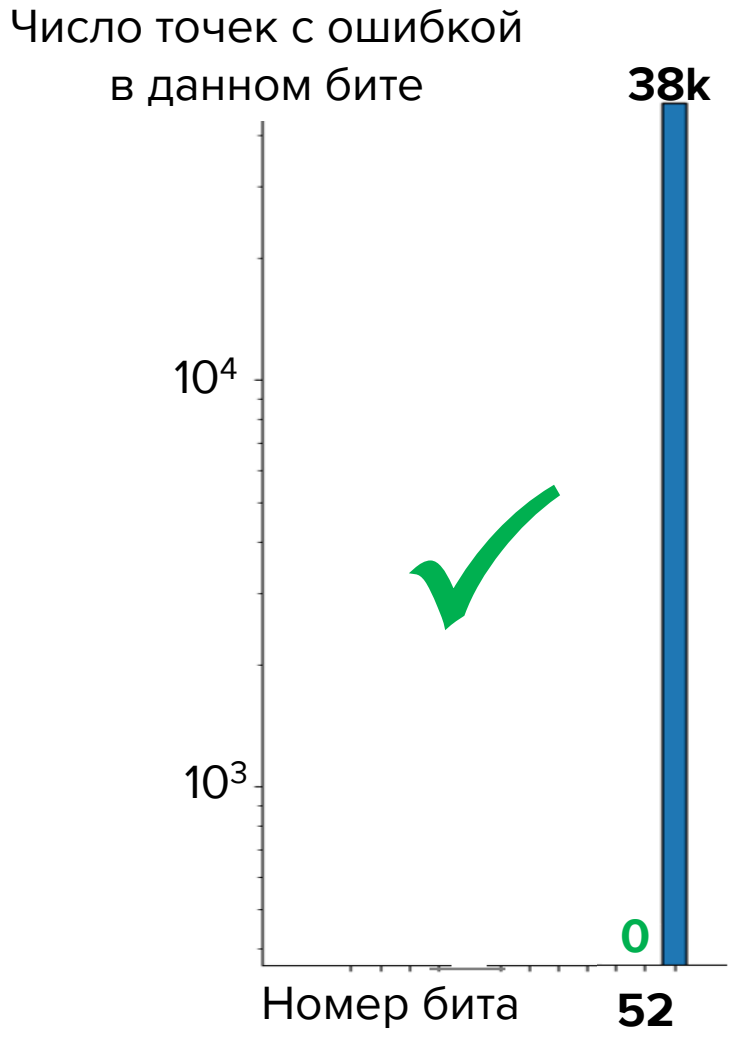
Вернёмся к нашему неточному tanh из glibc

- Рассмотрим реализацию функции **`tanh`**, популярной в активационных функциях для AI.
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.
- **Что видим**: в тестовой выборке **450** точек имеют ошибку уже в **52** бите.
- **Вывод**: в glibc точность реализации функции `tanh` не удовлетворяет стандарту.



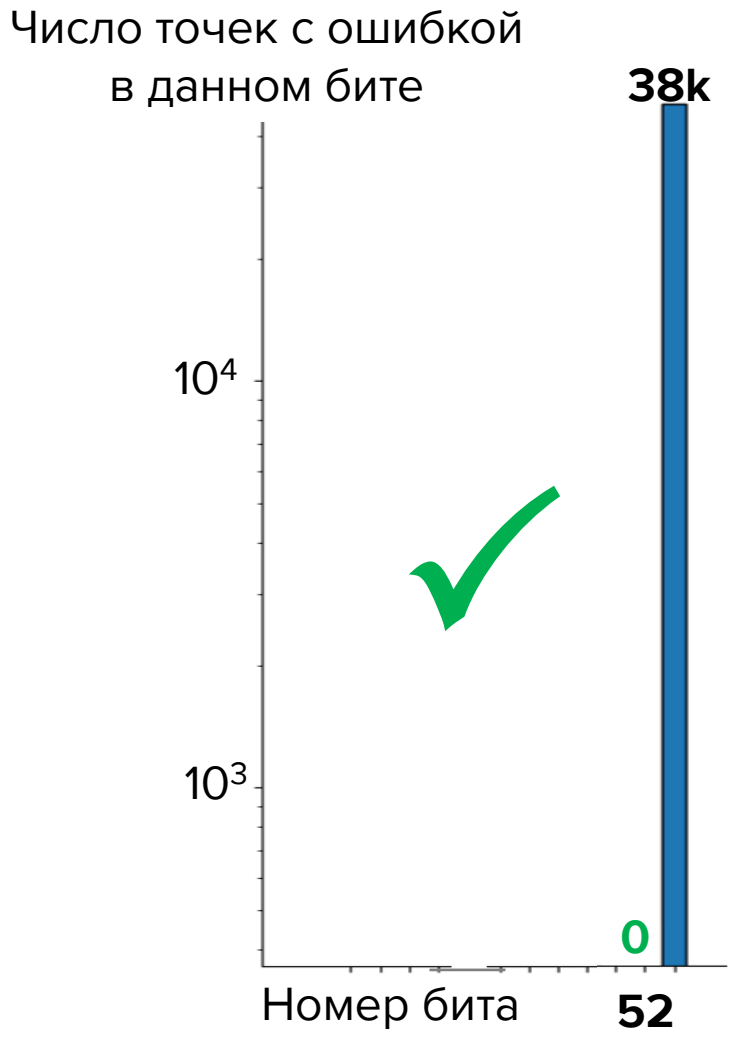
Построим более точную реализацию tanh

- **Используем минимаксную аппроксимацию и устойчивый порядок вычислений.**
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.
- **Что видим**: в тестовой выборке 0 точек имеют ошибку до 53-го бита.
- **Вывод**: теперь всё точно!



Построим более точную реализацию tanh

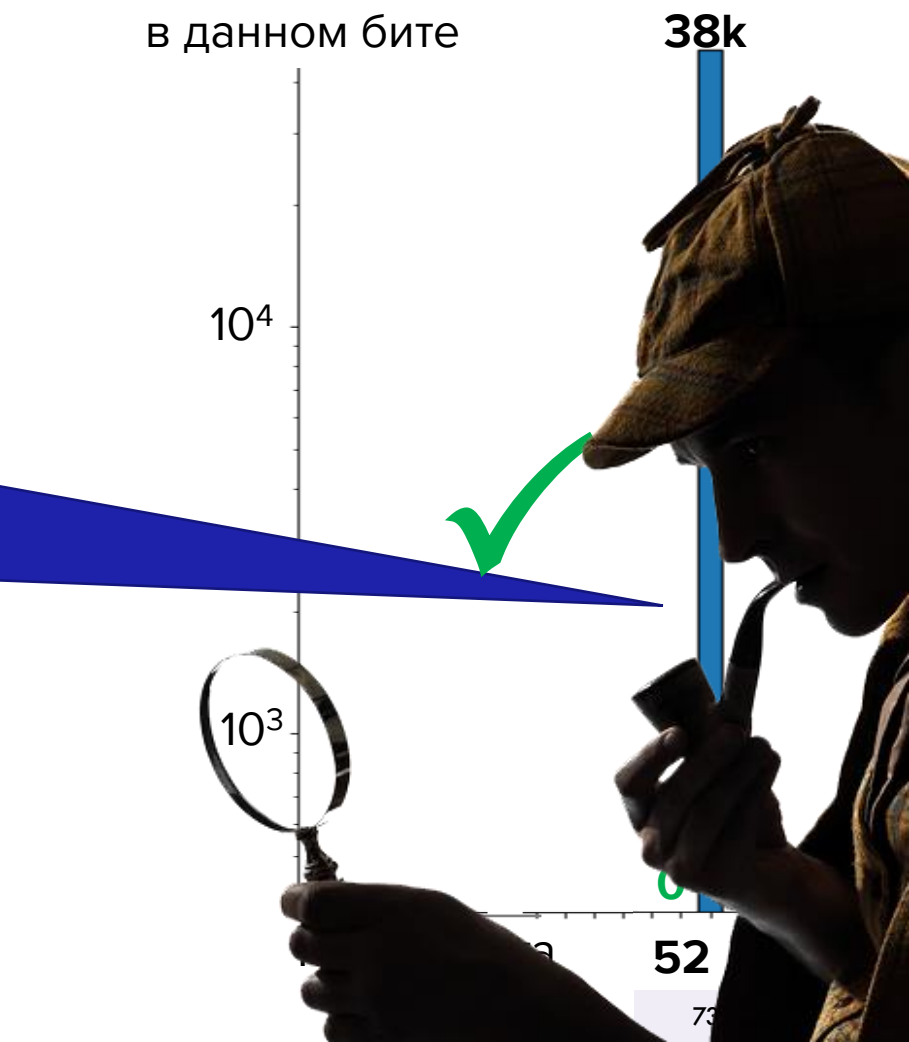
- Используем минимаксную аппроксимацию и устойчивый порядок вычислений.
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.
- **Что видим**: в тестовой выборке 0 точек имеют ошибку до 53-го бита.
- **Бонус**: такая реализация работает **на 70% быстрее** чем неточный аналог из glibc!



Построим более точную реализацию tanh

- Используем минимаксную аппроксимацию и устойчивый порядок вычислений.
- **Что проверяем**: значения не должны иметь расхождения с эталоном до 53-го бита.

Но как мы это проверяем?



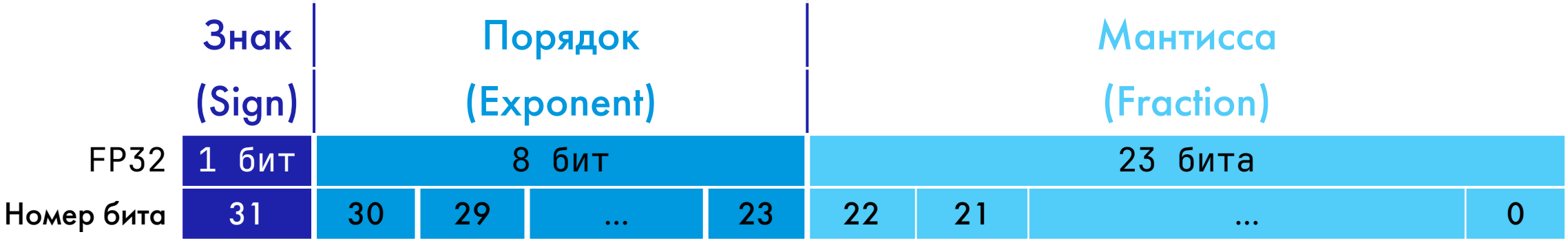
Что из себя представляют числа с плавающей точкой?

Числа одинарной точности (float, FP32)

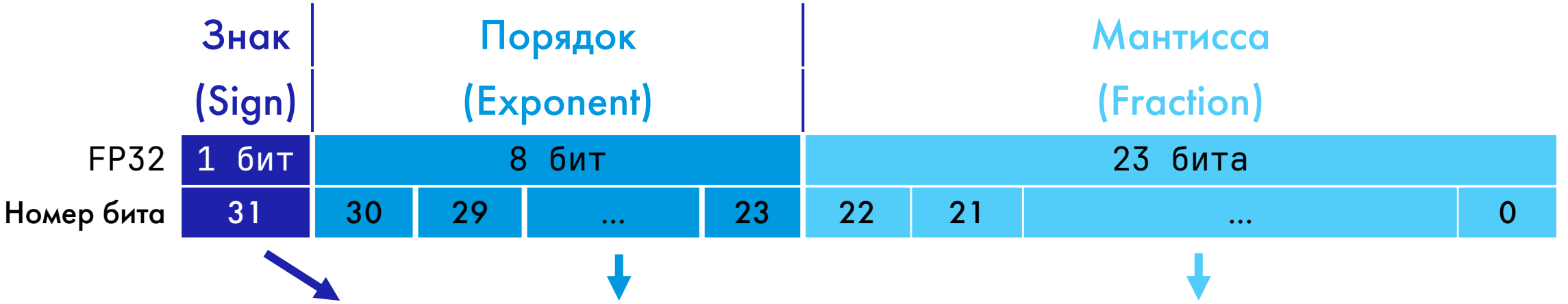
- Занимают в памяти компьютера **32 бита**.
- Как эти биты становятся привычным глазу десятичным числом?

Числа одинарной точности (float, FP32)

Числа одинарной точности (float, FP32)



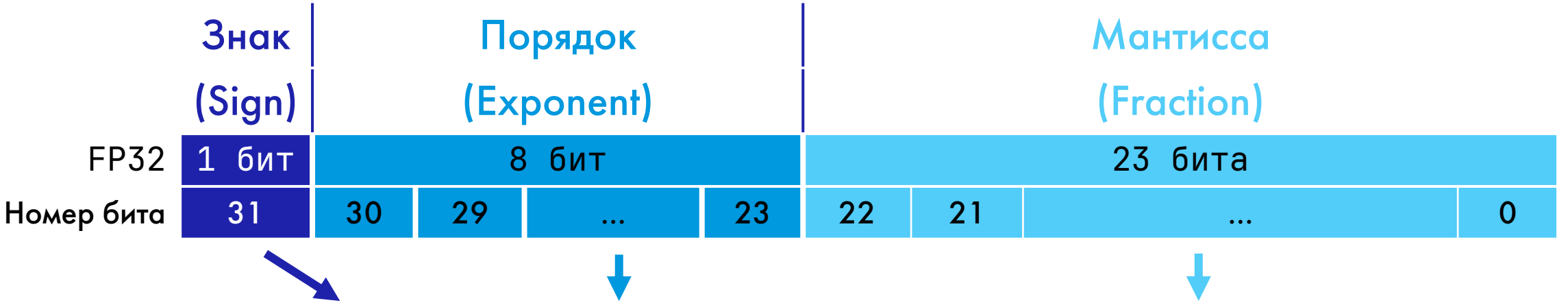
Числа одинарной точности (float, FP32)



$$\mathrm{FP}_{32} = (-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2$$

- b_i – значение i-го бита (0 или 1);
- $127 = 2^{8-1} - 1$ – нулевое смещение (bias);
- $(\ldots)_2$ – число в двоичной системе счисления.

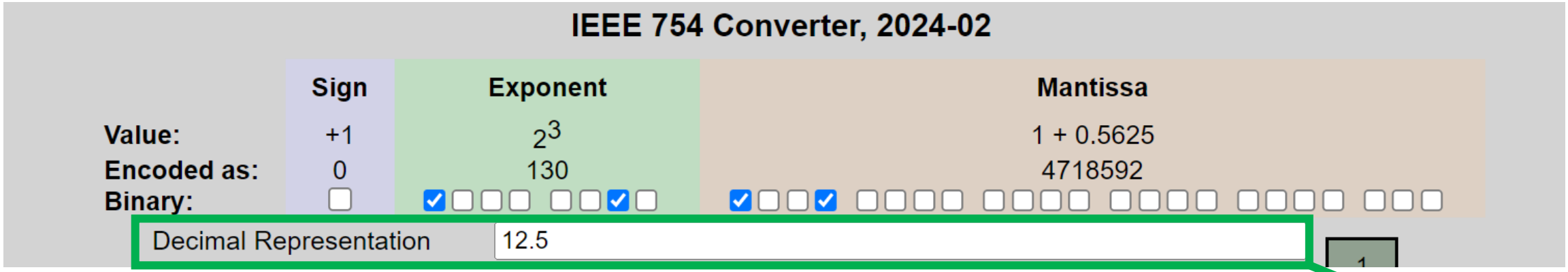
Числа одинарной точности (float, FP32)



$$\mathrm{FP}_{32} = (-1)^{b_{31}} \times 2^{\sum_{i=0}^{7} b_{23+i} 2^{i} - 127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

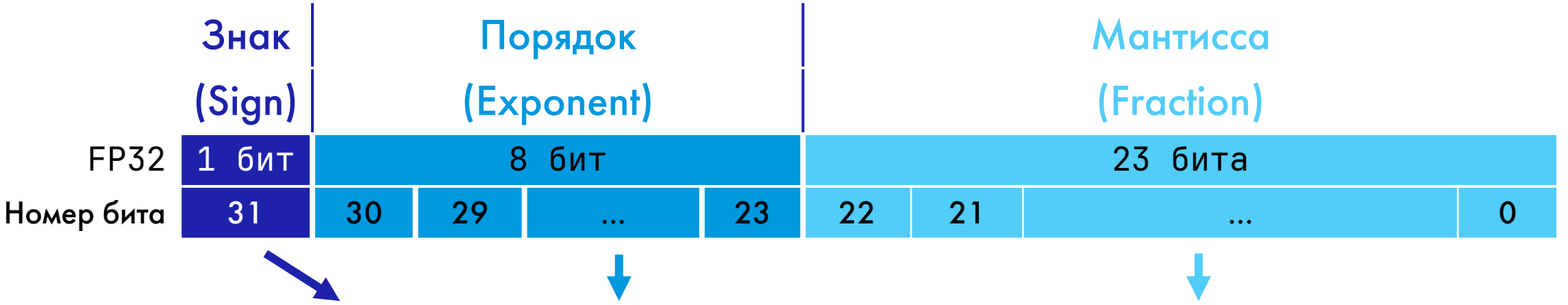
- b_i – значение i-го бита (0 или 1);
- $127 = 2^{8-1} - 1$ – нулевое смещение (bias).

Пример



$$\mathrm{FP}_{32} = (-1)^{0} \times 2^{130-127} \times (1 + 0.5625) = 8 \times 1.5625 = 12.5$$

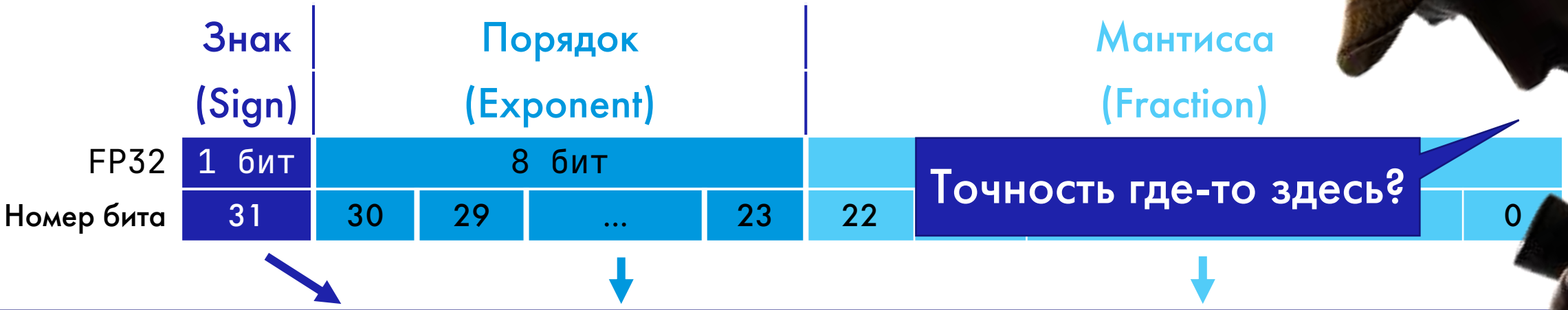
Числа одинарной точности (float, FP32)



$$\mathrm{FP}_{32} = (-1)^{b_{31}} \times 2^{\sum_{i=0}^{7} b_{23+i} 2^i - 127} \times \left(1 + \underbrace{\sum_{i=1}^{23} b_{23-i} 2^{-i}}_{\in [0,1)}\right)$$

- b_i – значение i-го бита (0 или 1);
- $127 = 2^{8-1} - 1$ – нулевое смещение (bias).

Числа одинарной точности (float, FP32)

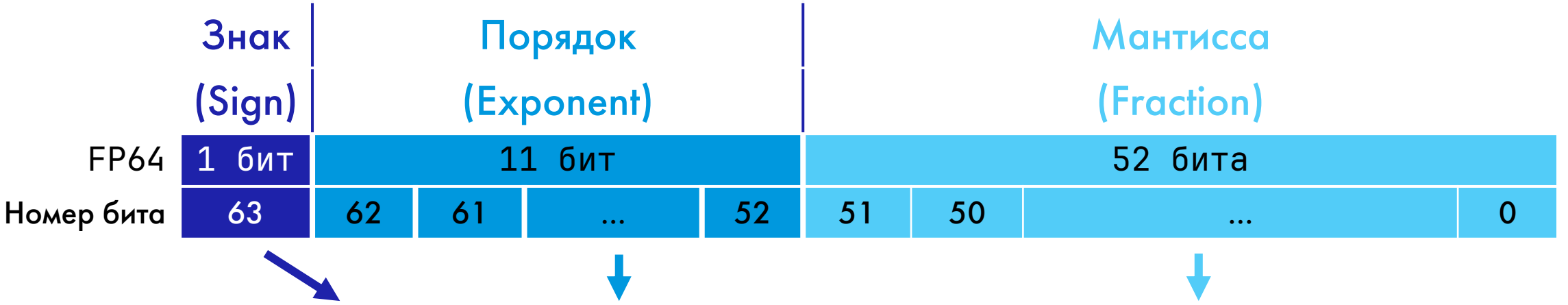


$$\mathrm{FP}_{32} = (-1)^{b_{31}} \times 2^{\sum_{i=0}^{7} b_{23+i} 2^i - 127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

$\sum_{i=1}^{23} b_{23-i} 2^{-i} \in [0,1)$

- b_i – значение i-го бита (0 или 1);
- $127 = 2^{8-1} - 1$ – нулевое смещение (bias).

Числа двойной точности (double, FP64)



$$\mathrm{FP}_{64} = (-1)^{b_{63}} \times 2^{\sum_{i=0}^{10} b_{52+i} 2^i - 1023} \times \left(1 + \underbrace{\sum_{i=1}^{52} b_{52-i} 2^{-i}}_{\in [0,1)}\right)$$

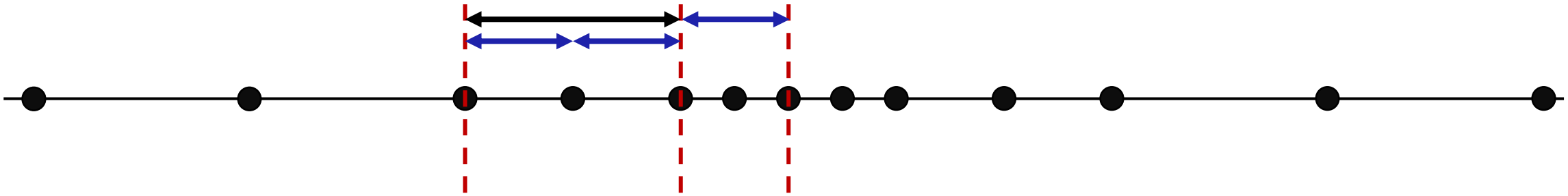
- b_i – значение i-го бита (0 или 1);
- $1023 = 2^{11-1} - 1$ – нулевое смещение (bias).

Интересные наблюдения из жизни мантисс

- При переходе от одинарной к двойной точности ширина мантиссы возрастает более чем в 2 раза – с 23 до 52 бит.
- Мантисса принадлежит множеству из 2^n дискретных значений, равномерно распределенных в двоичном представлении (n – число бит).

Интересные наблюдения из жизни мантисс

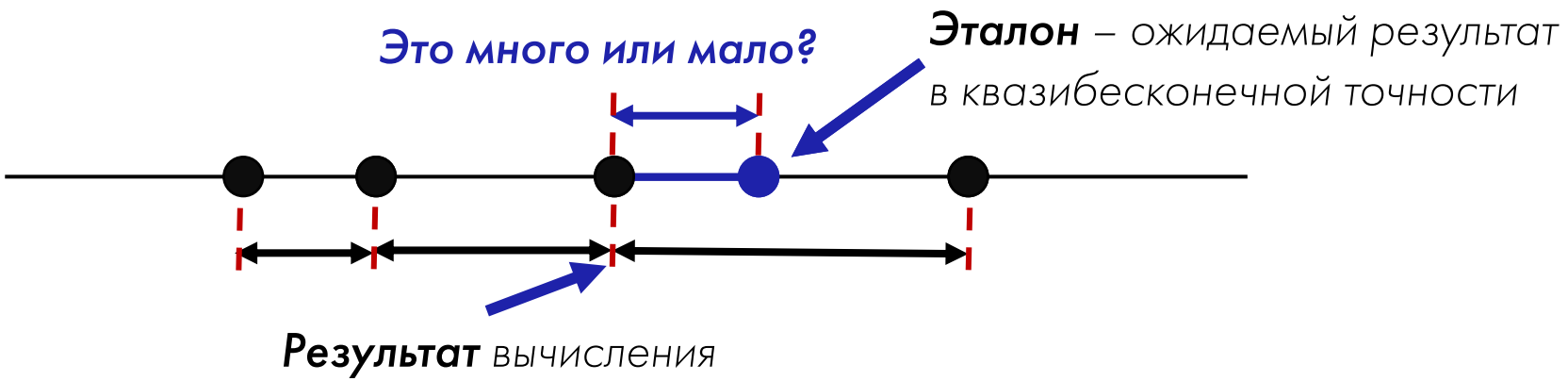
- При переходе от одинарной к двойной точности ширина мантиссы возрастает более чем в 2 раза – с 23 до 52 бит.
- Мантисса принадлежит множеству из 2^n дискретных значений, равномерно распределенных в двоичном представлении (n – число бит).
- В десятичном представлении числа расположены **неравномерно**.



Как измерить точность аппроксимации?

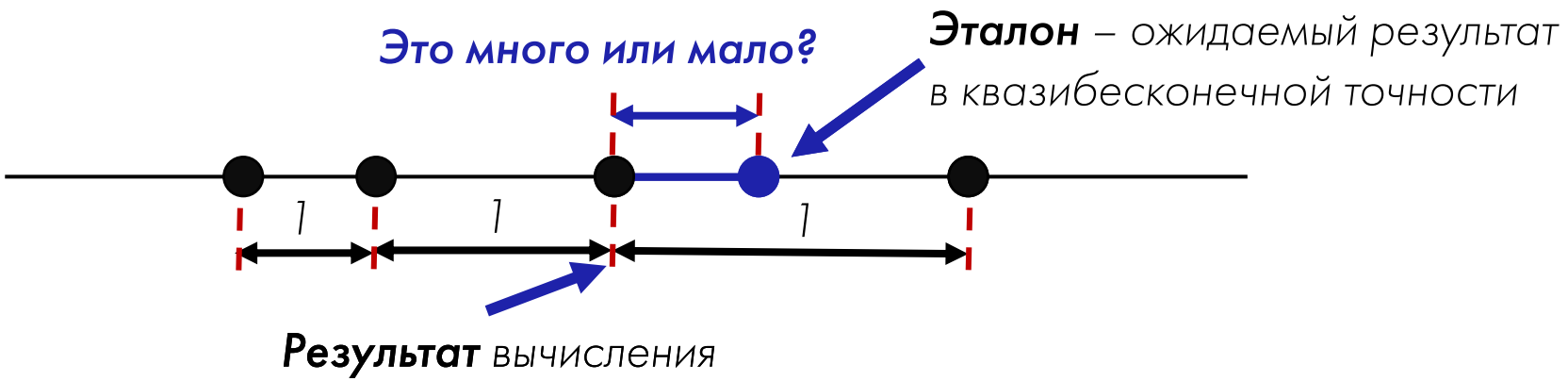
Обычная линейка не работает

- Как мы выяснили, числа с плавающей точкой **неравномерно** заполняют числовую ось.



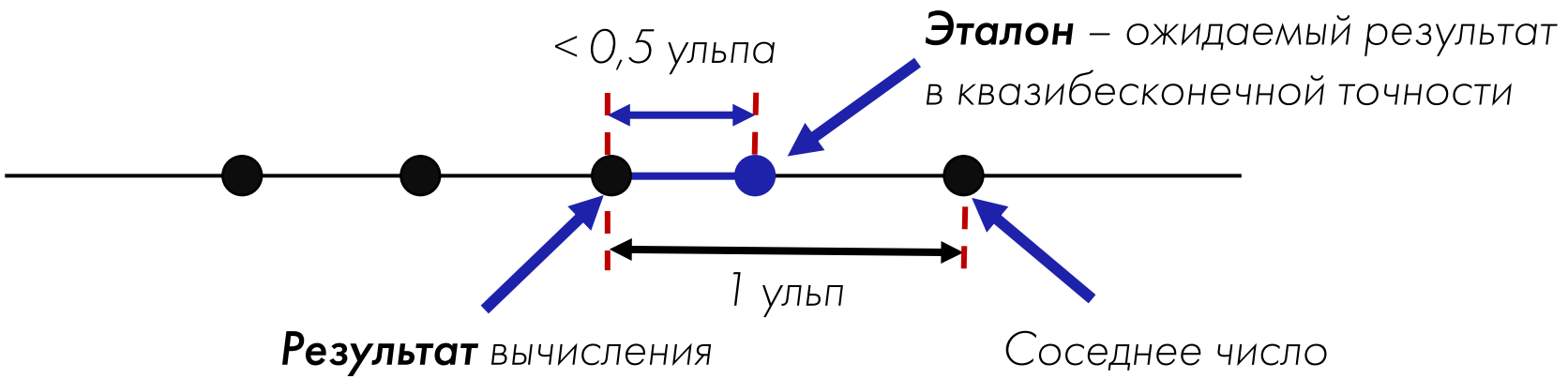
Обычная линейка не работает

- Как мы выяснили, числа с плавающей точкой неравномерно заполняют числовую ось.
- Наша **единица измерения** – расстояние между соседними числами – **1 ульп** (**U**nit in the **L**ast **P**lace)



Расчет ошибки и её оценка

- Ошибка – отношение расстояния до эталона к расстоянию до соседа.
- Стандарт libm требует, чтобы ошибка не превышала 0.5 ульпа (учитываем корректность округления).



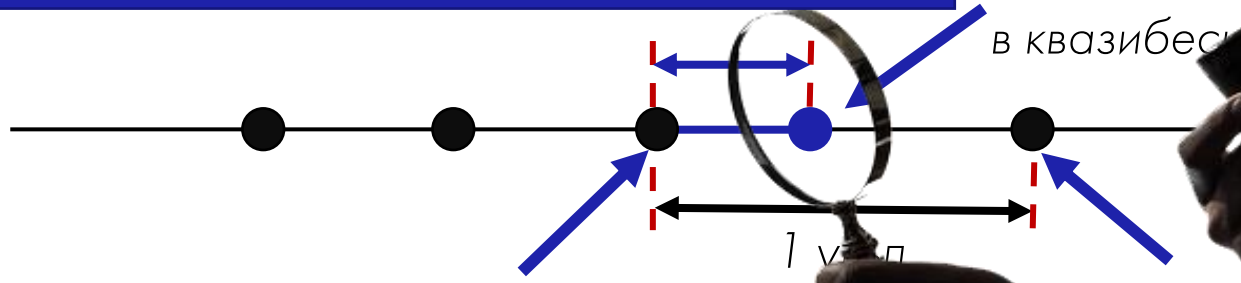
КАК ИЗМЕРИТЬ ТОЧНОСТЬ АППРОКСИМАЦИИ

Расчет ошибки и её оценка

А откуда мы берём эталон в квазибесконечной точности?

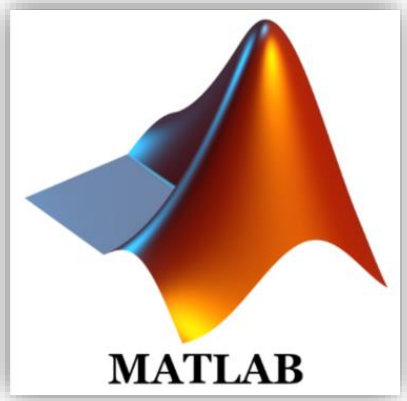
…на к расст… соседа.

…евышал…



Системы компьютерной алгебры

Это прикладные программы для **символьных вычислений** (математических преобразований в аналитической форме), а также **числовых операций произвольной точности**.



Что использовать внутри системы тестирования?

Наша рекомендация – Sollya:

- предоставляет **удобный C-интерфейс** – можно вызывать прямо из тестов

Что использовать внутри системы тестирования?

Наша рекомендация – Sollya:

- предоставляет **удобный C-интерфейс** – можно вызывать прямо из тестов;
- для кратного повышения точности вычислений использует **GNU MPFR** (**M**ultiple **P**recision **F**loating-**P**oint **R**eliable Library) – надежную библиотеку с менее удобным интерфейсом

Что использовать внутри системы тестирования?

Наша рекомендация – Sollya:

- предоставляет **удобный C-интерфейс** – можно вызывать прямо из тестов;
- для кратного повышения точности вычислений использует **GNU MPFR** (**M**ultiple **P**recision **F**loating-**P**oint **R**eliable Library) – надежную библиотеку с менее удобным интерфейсом;
- позволяет выбирать степень повышения точности относительно ширины типа данных – для наших целей достаточно восьмикратной точности.

Как же правильно тестировать точность libm?

Формирование датасета контрольных точек

- Не будем повторять чужих ошибок – будем генерировать **случайные** исходные данные.

Формирование датасета контрольных точек

- Не будем повторять чужих ошибок – будем генерировать **случайные** исходные данные.
- Обеспечим равномерное (с точностью до устройства числа) распределение чисел.

Формирование датасета контрольных точек

- Не будем повторять чужих ошибок – будем генерировать **случайные** исходные данные.
- Обеспечим равномерное (с точностью до устройства числа) распределение чисел.
- Так мы получим **100% покрытие всех веток кода** за один тест.
- В пределе (регулярные nightly запуски) протестируем всю числовую ось.

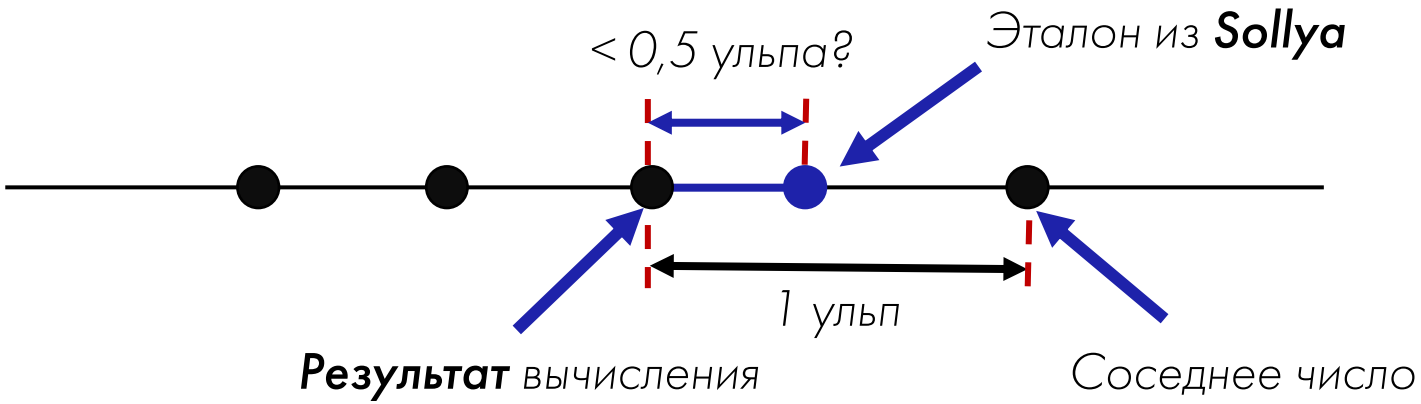
Формирование датасета контрольных точек

	Sign	Exponent	Mantissa
Value:	+1	2^{-3}	1 + 0.00008547306060791016
Encoded as:	0	124	717
Binary:	☐	☐☑☑☑☑☑☐☐	☐☐☐☐☐☐☐☐☐☐☐☐☐☑☐☑☑☐☐☑☑☐☑

```
for (exponent = min; exponent < max; exponent++)
  while (mantissa < max)
    mantissa += step;
    value = (sign << 31) | (exponent << 23) | (mantissa << 0)
    TEST(value)
```

Сравнение с эталоном при помощи Sollya

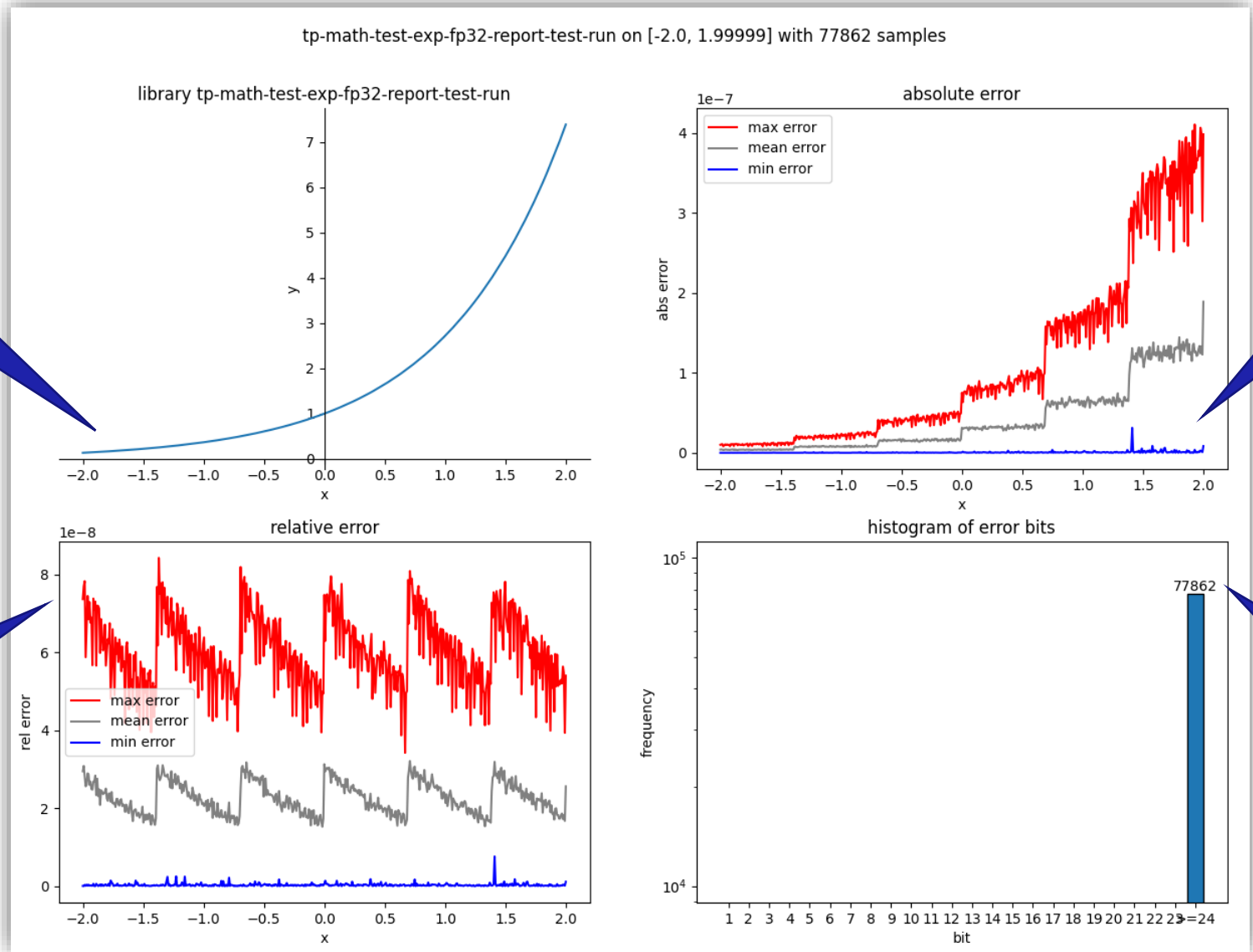
- Запускаем нашу функцию, получаем результат и передаем его в Sollya.
- В Sollya вычисляем **эталон** и **разность.**
- Если разность меньше половины расстояния между соседями – мы точны!



Графическое представление результатов тестирования

Функция

Абсолютная ошибка



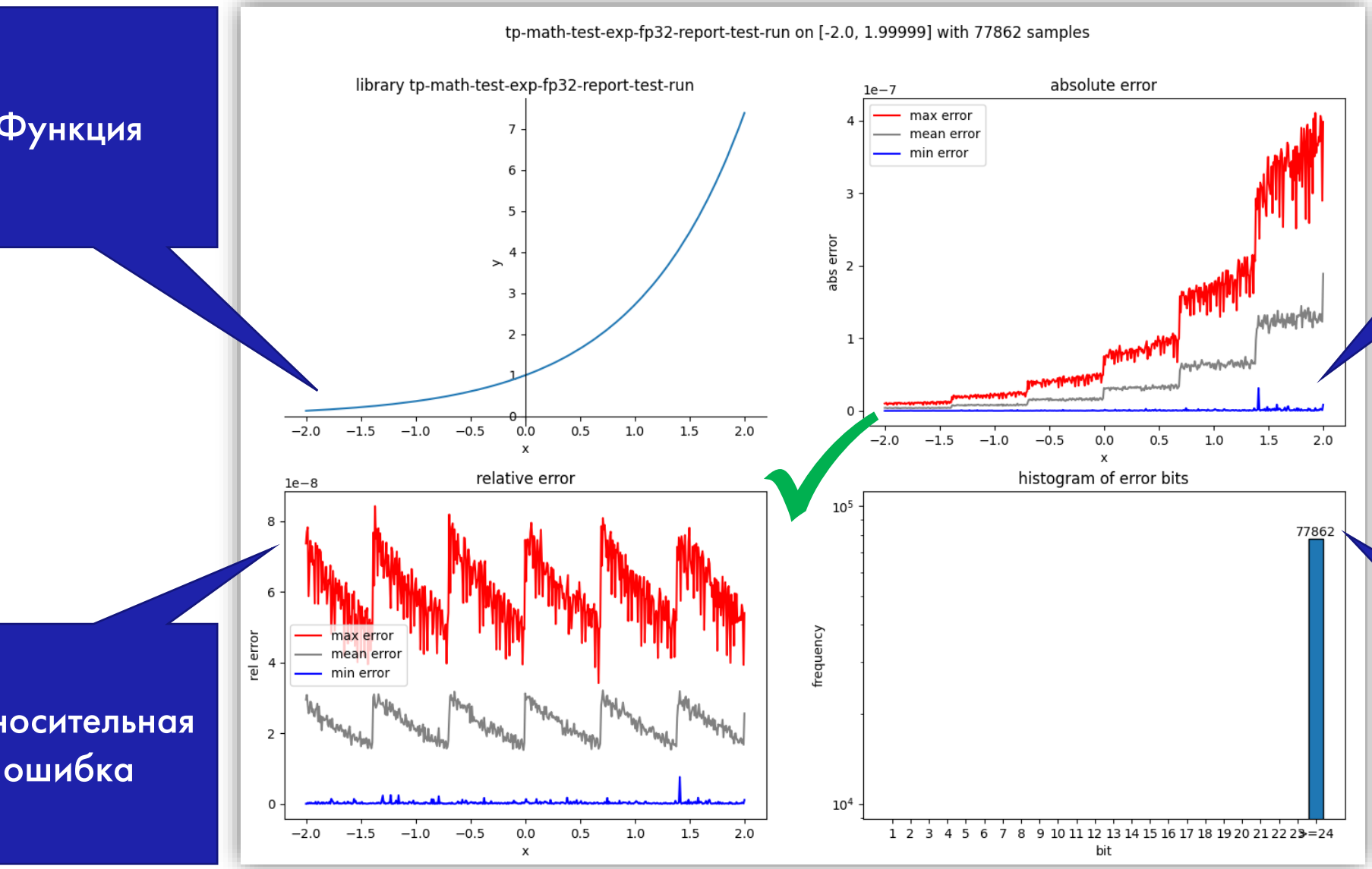
Относительная ошибка

Число точек с ошибкой в данном бите

Графическое представление результатов тестирования

Функция

Абсолютная ошибка



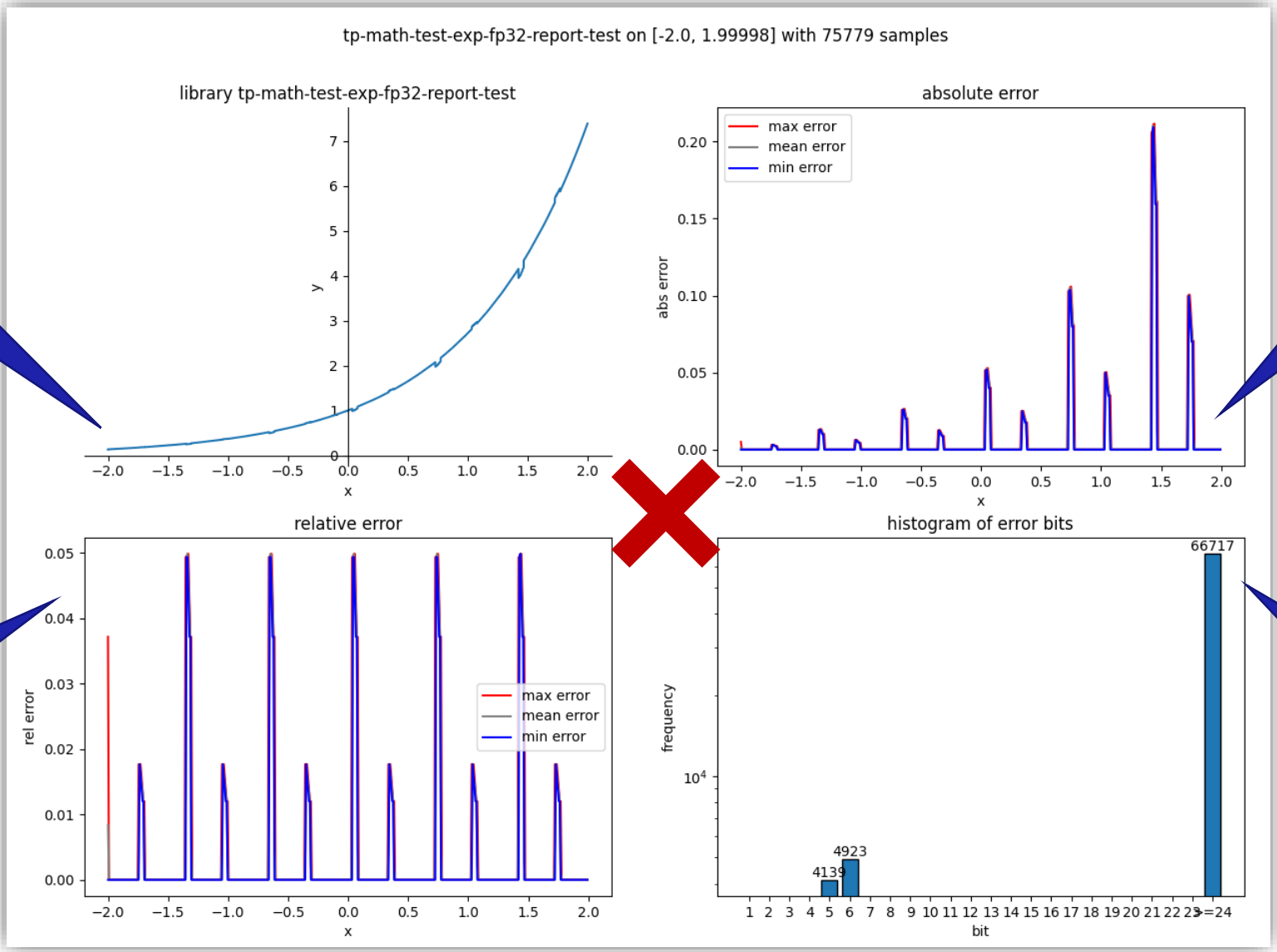
Относительная ошибка

Число точек с ошибкой в данном бите

Графическое представление результатов тестирования

Функция

Абсолютная ошибка



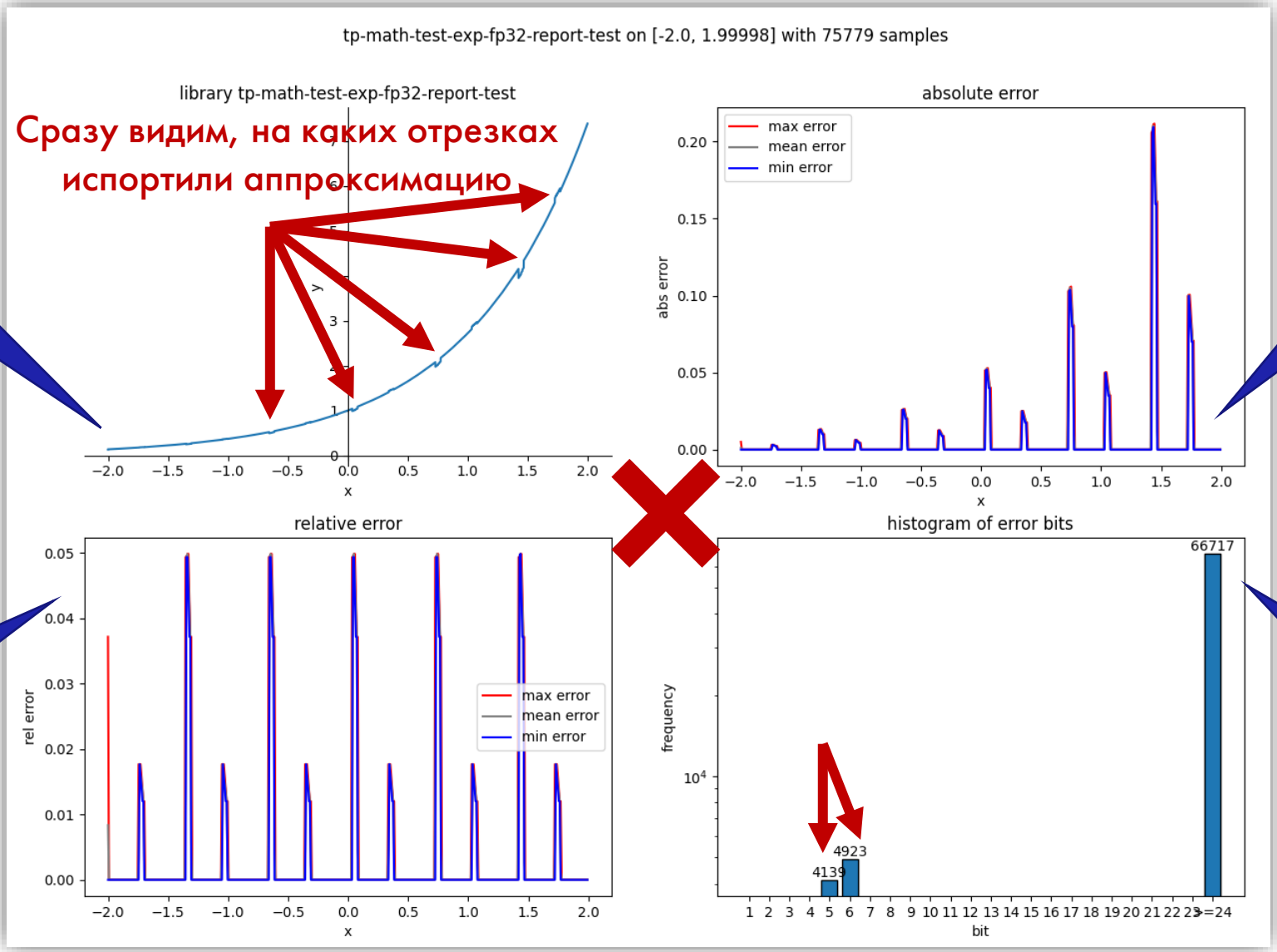
Относительная ошибка

Число точек с ошибкой в данном бите

Графическое представление результатов тестирования

Функция

Абсолютная ошибка



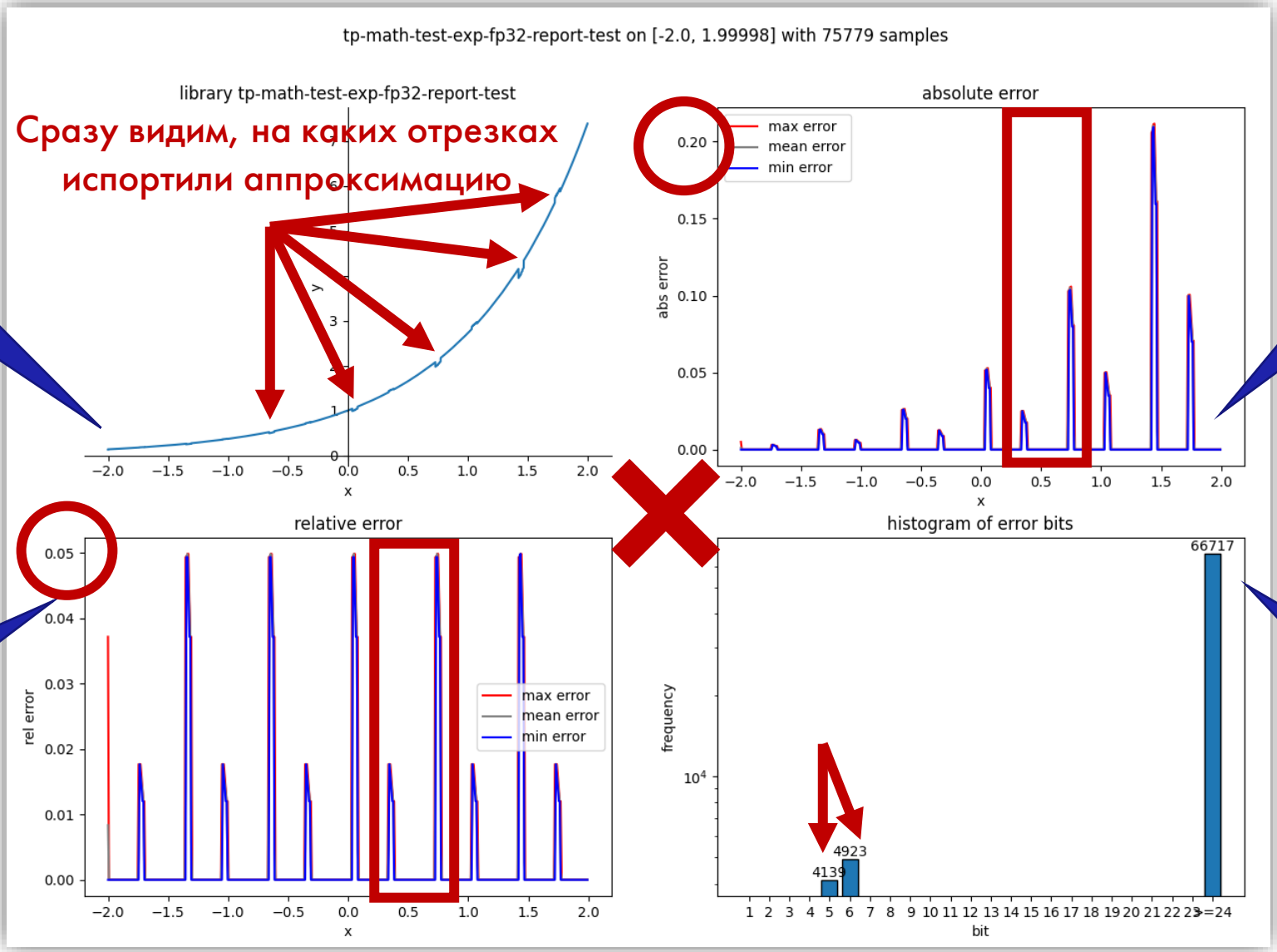
Относительная ошибка

Число точек с ошибкой в данном бите

Графическое представление результатов тестирования

Функция

Абсолютная ошибка



Относительная ошибка

Число точек с ошибкой в данном бите

Что в итоге?

Результаты

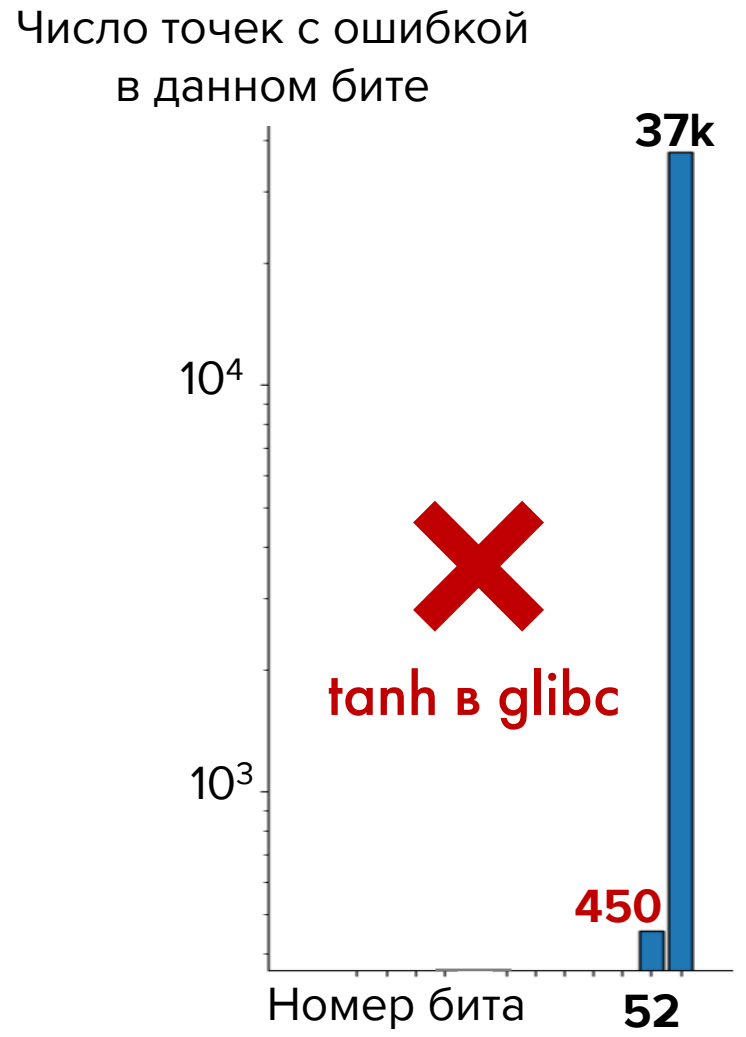
- Разработали систему тестирования точности математических функций (без ограничений по фреймворкам), обеспечивающую в пределе 100% покрытие тестами всех значений.
- Разработали высокопроизводительные реализации функций libm, удовлетворяющие стандарту по точности (0,5 ульп).

Преимущества нашей реализации libm

- По сравнению с glibc **улучшена точность** реализаций гиперболических функций и гамма-функций.

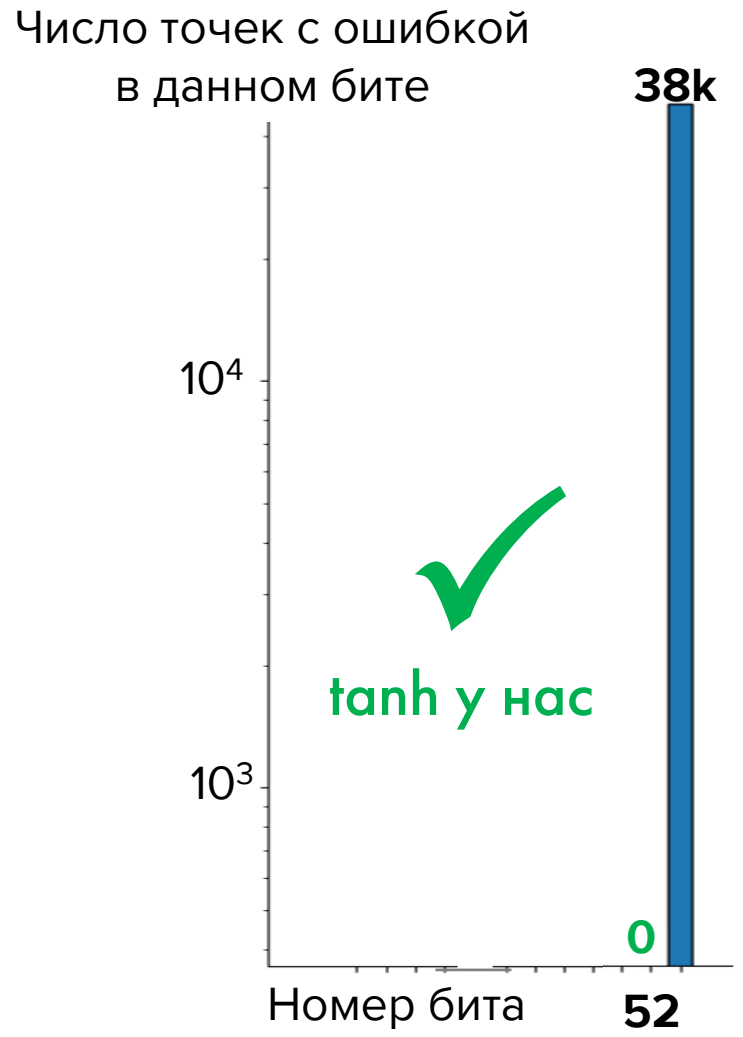
Преимущества нашей реализации libm

- По сравнению с glibc **улучшена точность** реализаций гиперболических функций и гамма-функций.



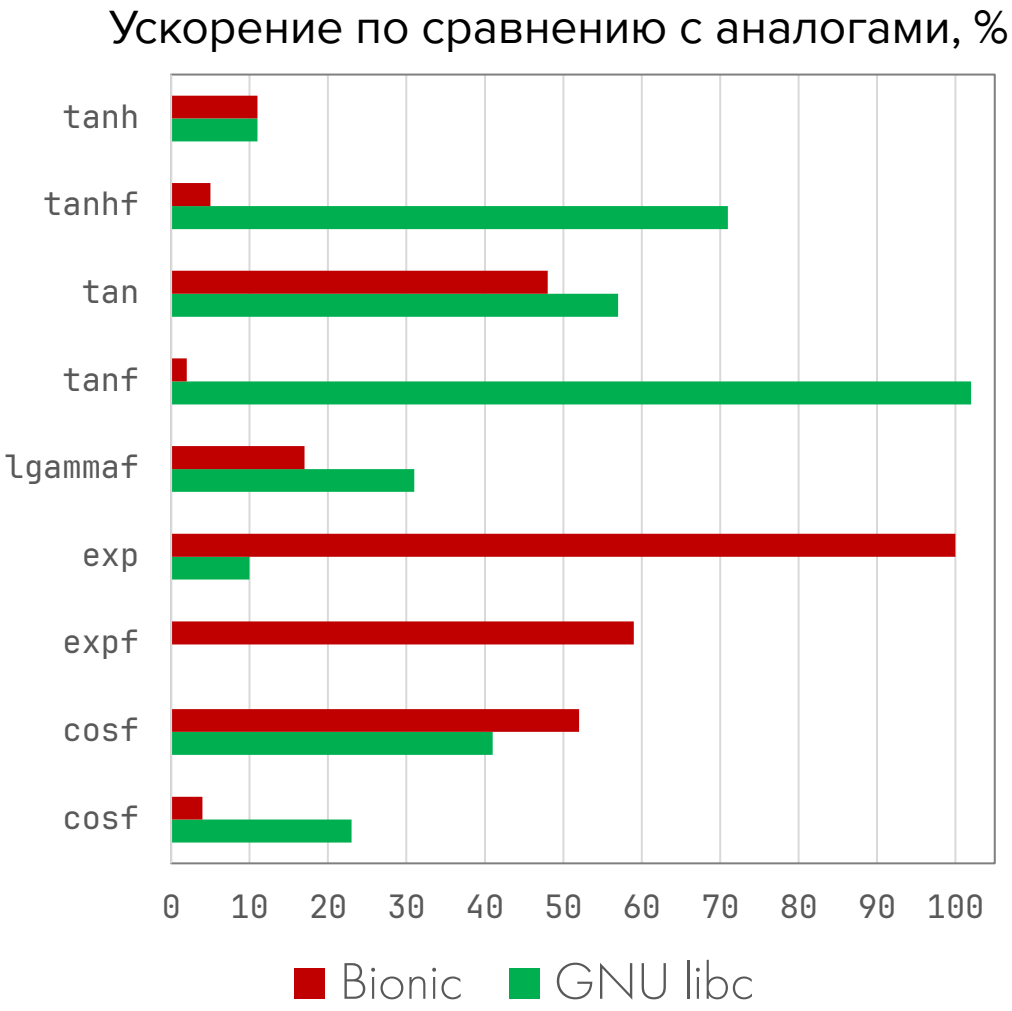
Преимущества нашей реализации libm

- По сравнению с glibc **улучшена точность** реализаций гиперболических функций и гамма-функций.



Преимущества нашей реализации libm

- По сравнению с glibc **улучшена точность** реализаций гиперболических функций и гамма-функций.
- Для ряда функций **производительность выше** реализаций из Bionic и glibc.



* Замеры на RISC-V плате **Lichee Pi 4A** (THead C910v, 1.85GHz) по всем ветвям исполнения.

Численная устойчивость коварна

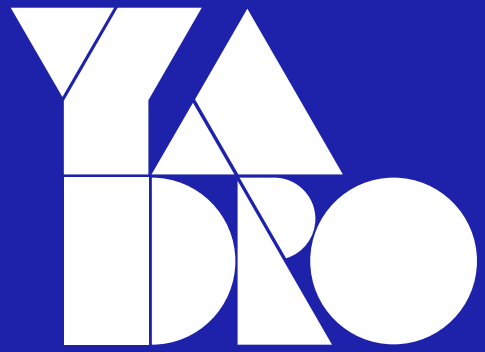
- Числа с плавающей точкой распределены неравномерно.
- Законы арифметики нарушаются: порядок вычислений имеет значение.

Численная устойчивость коварна

- Числа с плавающей точкой распределены неравномерно.
- Законы арифметики нарушаются: порядок вычислений имеет значение.
- Точность прогноза погоды страдает.

Численная устойчивость коварна

- Числа с плавающей точкой распределены неравномерно.
- Законы арифметики нарушаются: порядок вычислений имеет значение.
- Точность прогноза погоды страдает.



Москва,
ул. Рочдельская, 15, стр. 13
+7 800 777-06-11

yadro.com