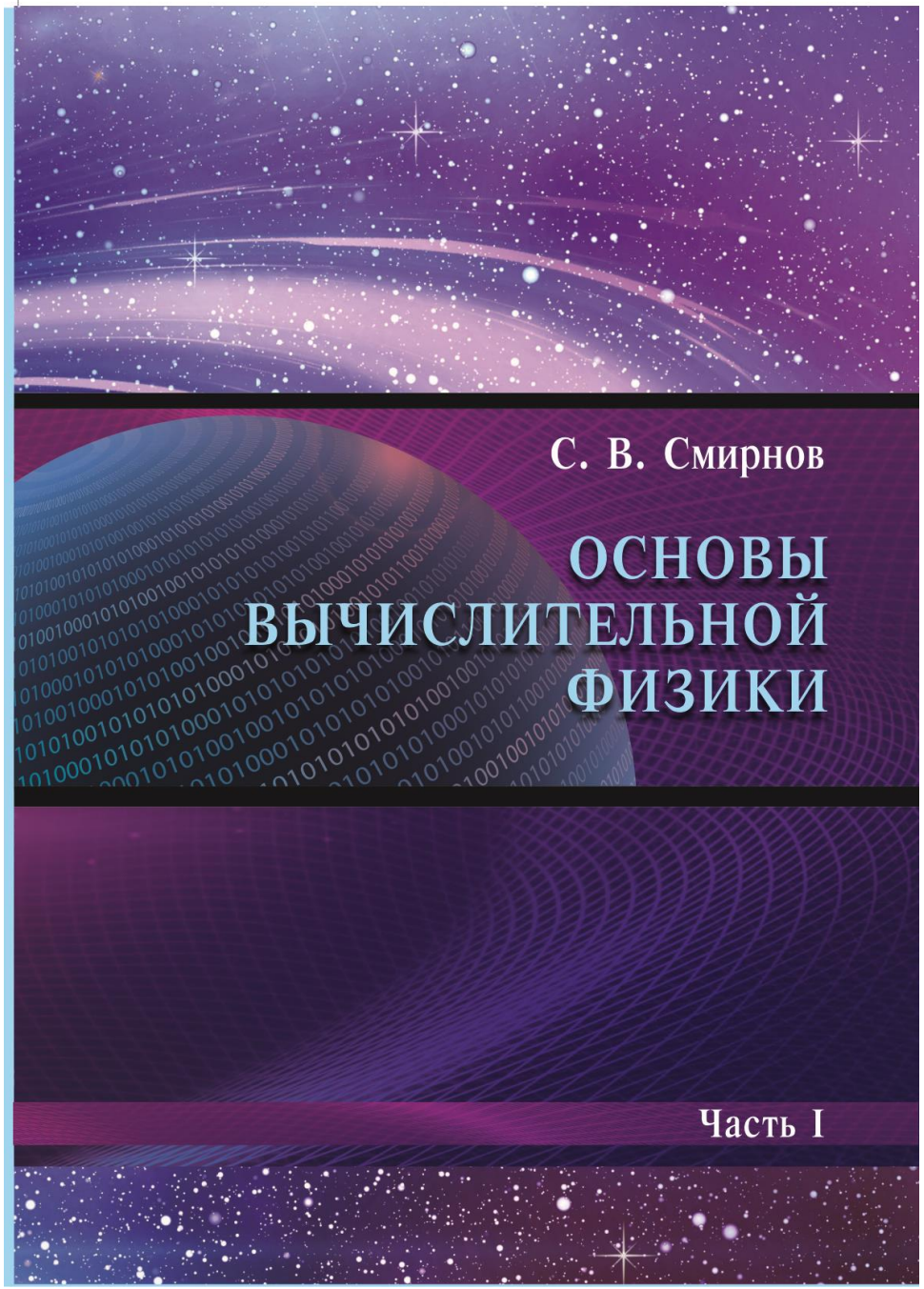
ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ФИЗИКИ

лекция 1

Смирнов Сергей Валерьевич

кафедра высшей математики ФФ НГУ, 2017

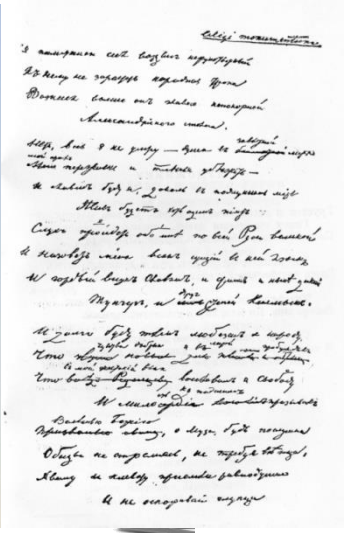


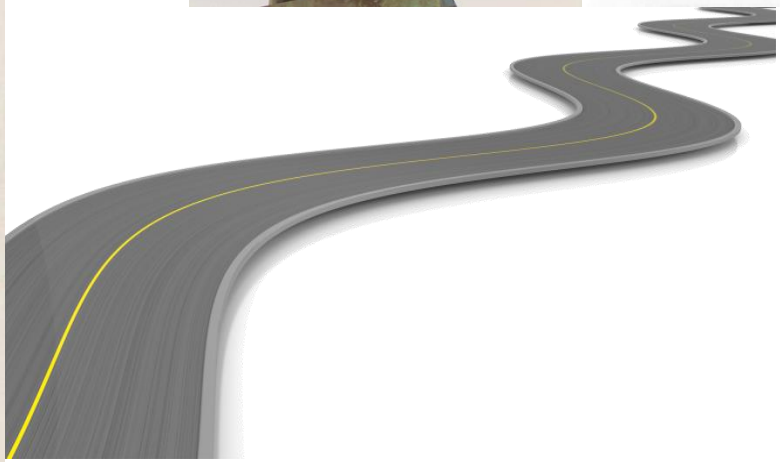
Материалы первой лекции можно найти в методичке

+ презентация (а также программа курса и задания) на сайте кафедры высшей математики ФФ
http://www.phys.nsu.ru/smirnov/ovf.html

ЗНАКОМСТВО

ЧТО? *Основы вычислительной физики –*
Методы вычислений, используемые
при решении физических задач





ЗНАКОМСТВО

ЧТО? *Основы вычислительной физики* –
Методы вычислений, используемые
при решении физических задач

ГДЕ? БФА (*лекции*)
Терминальные классы (*практикум*)

КОГДА? Сентябрь-декабрь, 1 лекция + 1 семинар в неделю
Декабрь: *диф.зачёт*

КАК?

Лекции:
- краткий обзор основных методов решения базовых задач
- примеры-демонстрации решения задач, обзор типичных ошибок
- мини-контрольные работы

Практикум:
- самостоятельное решение задач;
- навык отладки программ, анализа и верификации результатов

Диф.зачёт:
- курсовая работа
- итоги решения задач в семестре
- субъективное мнение преподавателя
- результаты контрольных работ на лекциях
- экзамен/зачёт [*для желающих повысить оценку*]

ЗНАКОМСТВО

КАК?

Язык программирования

Для наших целей подошёл бы почти любой язык

Почему всё-таки Си:

- Высокая эффективность
- Наличие бесплатных и open source библиотек
- Работает везде (компиляторы на все случаи жизни)
- Студенты 4-го курса ФФ НГУ знакомы с Си :)
- Единообразие при обучении

ОВФ – это ***НЕ*** курс по программированию!

- Программирование здесь – лишь *метод* решения задач, но не самоцель
- Если Вы забыли язык Си/Си++, нужно вспомнить основы СЕЙЧАС
- Как правило, можно будет ограничиться короткими и простыми программами
- Написание программы, дающей ответ, – это лишь начало пути
- Ценность программы, которая компилируется и «что-то считает», близка к нулю

ЗНАКОМСТВО

ЗАЧЕМ?

Американская атомная бомба "Толстяк", уничтожившая Нагасаки 9 августа 1945 года

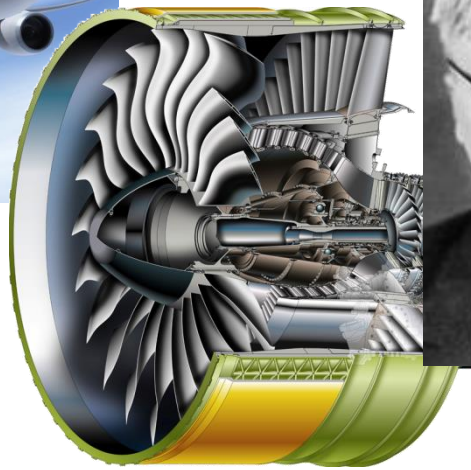
ЗНАКОМСТВО

ЗАЧЕМ?

Американская атомная бомба

Ленинградская АЭС –
активная зона реактора РБМК

РДС-1

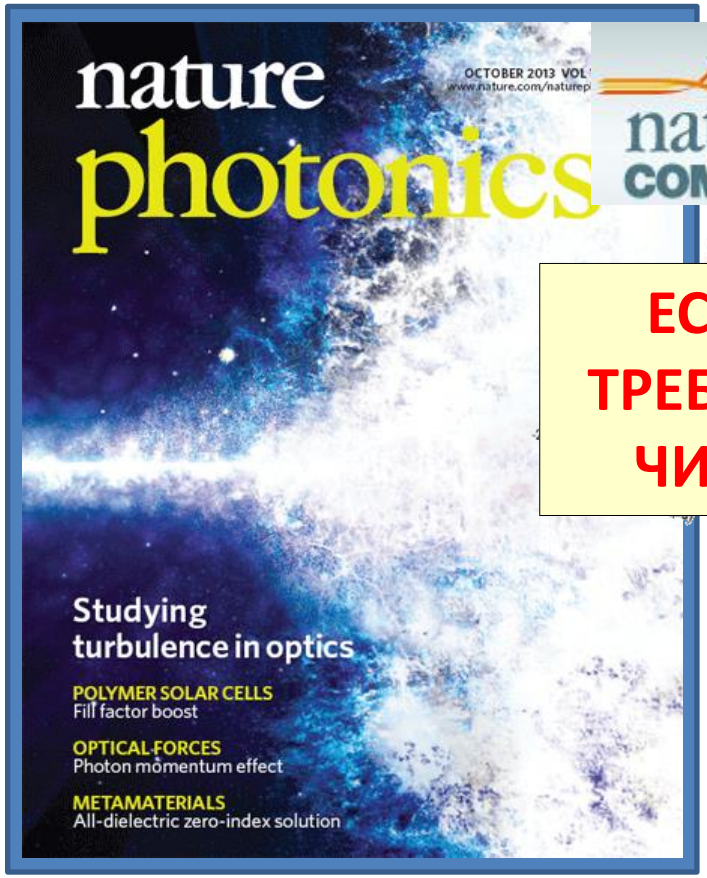


Турбина Балаковской АЭС

ЗНАКОМСТВО

ЗАЧЕМ?

MIT OpenCourseWare



ЕСТЬ ЛИ У ВАС ЗАДАЧА, ТРЕБУЮЩАЯ ПРИМЕНЕНИЯ ЧИСЛЕННЫХ МЕТОДОВ?

аналитика **эксперимент** **численный счёт**

Как правило, при решении достаточно сложной практической задачи используются и аналитические, и численные методы.

Первые позволяют понять качественную картину, приближенные зависимости от многих параметров. Вторые – уточнить полученные значения, особенно в промежуточных областях (в отсутствие малых и больших параметров), найти границы области применимости аналитических решений.

Аналитическое решение vs. численные методы

- Аналитическое решение позволяет увидеть зависимость от произвольного числа параметров.
- Аналитическое решение может быть получено для относительно простых задач (при наличии симметрии, малых параметров и т.п.) и почти всегда является приближенным.

- Численное решение дает значение искомой величины в одной точке. Чтобы получить зависимость от нескольких параметров, нужно выполнить большое число однотипных расчетов, что может потребовать много времени (ресурсов).
- Численное решение можно получить с меньшим количеством приближений – для несимметричных систем, при отсутствии малых параметров и т.п. Можно достичь более высокой точности решения / расширить область применимости.
- Численные методы решения задач мат.физики создают ощущение простоты (особенно при использовании готовых решений – MatLab, Mathematica и т.п.), но оно лишь отчасти соответствует действительности

J. R. Rice. *Numerical Methods, Software and Analysis*. 1983. p 343

The striking conclusion is reached that, for the general three-dimensional elliptic problem 10.3.2, **the progress made through better methods from 1945 to 1978 exceeds the progress made through faster computers**.

ТОЧНОСТЬ / ПОГРЕШНОСТЬ ВЫЧИСЛЕНИЙ

Погрешность задания начальных условий и параметров → Погрешность вычислений → **Погрешность численного решения**

? Может ли погрешность ответа быть меньше (много меньше?) погрешности входных данных?

?? Может ли погрешность ответа, полученного написанной «без ошибок» программой, быть одного порядка (больше, много больше) вычисляемой величины?

??? Можно ли оценить погрешность, не зная точного ответа?

$$\pi = 2\sqrt{3}\sum_{k=0}^{\infty}\frac{(-1/3)^k}{2k+1}$$

К вопросам точности вычислений мы будем возвращаться на протяжении всего курса, а для начала разберёмся в том, как компьютер работает с числами.

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

Мотивация

Зачем это физикам?

Временами компьютер выдаёт неправильный результат. Нужно уметь (а) понимать, когда ответ неправильный, (б) исправлять ошибки.

Примеры:

$$\frac{\sin \pi n}{\sin^n(\pi/n)} \equiv 0 \quad \forall n \geq 2$$

Разбиение отрезка L на n частей h=L/n каждая и перебор while(L > 0) L=L-h

0.1+0.2-0.3=?

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

Мотивация

Времен уметь

(а) пони .

Пример

$$\frac{\sin \pi}{\sin^n(\pi}$$

Разбие . > 0) L=L-h

0.1+0.2-

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

целые числа

Сколько всего чисел?

В математике множество чисел *бесконечно*, однако *память ЭВМ ограничена*

Как следствие,

- количество различных чисел, которые могут быть записаны в ЭВМ, конечно;
- при любом способе кодирования существует min и max числа;
- аналог вещественных чисел в ЭВМ имеет конечную точность.

Как компьютер работает с числами?

(Для численного решения задач мы будем использовать язык программирования Си и x86-совместимые процессоры).

Все данные в компьютере хранятся и обрабатываются в двоичном виде, т.е. в виде последовательностей *битов*, каждый из которых может принимать всего два значения: 0 и 1.

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

целые числа

Как компьютер работает с числами?

(Для численного решения задач мы будем использовать язык программирования Си и x86-совместимые процессоры).

Все данные в компьютере хранятся и обрабатываются в двоичном виде, т.е. в виде последовательностей *битов*, каждый из которых может принимать всего два значения: 0 и 1.

Например, последовательность из трёх битов

101_2 соответствует десятичному числу $1*4+0*2+1*1 = 5_{10}$.

Последовательность 1101_2 соответствует $(1*8+1*4+0*2+1*0)_{10} = 13_{10}$ и т.п.

Стандартный целочисленный тип данных в Си (`int`) *обычно* имеет разрядность 32 бита. В беззнаковом (`unsigned`) варианте это позволяет кодировать числа от 0 до $2^{32}–1 = 4\ 294\ 967\ 295$, `signed int`: от –2 147 483 648 до +2 147 483 647.

Для тех случаев, когда этого недостаточно (кодирование даты, работа с памятью и файловой системой, финансы), предусмотрены 64-битные целые `std::int64_t`.

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

дробные числа

Можно ли, имея в распоряжении только целочисленную арифметику, заниматься физическими расчётами?

Можно, но это *неудобно*.

Можно было бы договориться, что последние пять десятичных разрядов числа являются дробной частью (формат чисел *с фиксированной точкой*). Например, число π представлялось бы при этом в виде 314159.

✓ Элементарная реализация четырёх арифметических действий

- Проблемы с переполнением: 32-разрядный `int` не позволяет записать значение скорости света в СГС $c = 3 \cdot 10^{10}$ см/с
- Проблемы с числами, близкими к нулю: невозможно записать значение постоянной Планка $\hbar = 1.054 \cdot 10^{-27}$ эрг · с
- Потеря точности при работе с относительно малыми числами (0.0001)

ВЫВОД: для численных расчётов больше подходят числа *с плавающей точкой* $3.00 \cdot 10^{10}$ см/с, $1.05 \cdot 10^{-27}$ эрг · с, $1.38 \cdot 10^{-23}$ Дж/К, $6.02 \cdot 10^{23}$ моль$^{-1}$

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

числа с плавающей точкой

Например,

$$\hbar = \underbrace{1.054}_{\text{мантисса}} \cdot \overbrace{10}^{\text{основание системы счисления (const, можно не хранить в памяти)}}\underbrace{^{-27}}_{\text{порядок числа (экспонента)}}$$

Неоднозначность: одно и то же число 3.14 можно записать по-разному:
$3.140 \cdot 10^{0}$, $0.314 \cdot 10^{1}$, $31.400 \cdot 10^{-1}$ и т.д.

Основные проблемы, связанные с неоднозначностью:

- Более низкая эффективность (скорость) вычислений и обработки данных
- Избыточность кода → неэффективное использование памяти

Решение: использование *нормализованных чисел* $(1 \le M_b < b)$

Договоримся, что первый разряд мантиссы нормализованного числа не равен нулю – значит, в двоичной записи он ***всегда*** равен единице!

В двоичном представлении нормализованное число x с плавающей точкой может быть записано с помощью трёх целых чисел S, E, M:

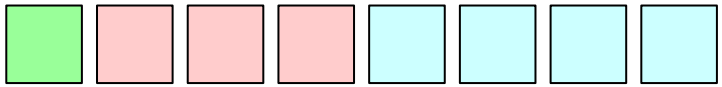
$$x = (-1)^{S} \cdot 1.M \cdot 2^{E}$$

Порядок E может быть разных знаков – в памяти его хранят в смещённом виде: $E \to E + bias, \quad bias = 2^{w-1} - 1 \approx E_{\max}/2$

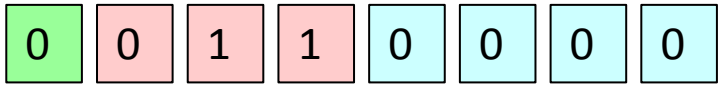
ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

нормализованные числа с плавающей точкой

Для наглядности рассмотрим гипотетическое восьмибитовое число с плавающей точкой с w=3 (три бита для записи порядка), t=4 (четыре бита для мантиссы):



Как записать число 1? Нормализованный вид числа 1.00 есть $+1.00 \times 2^0$:



Опуская единицу нормализованного числа 1.0000, имеем мантиссу $M = \mathbf{0}$

Смещённый порядок $E_S = E + (2^{w-1} - 1) = 0 + (2^{3-1} - 1) = \mathbf{3}$

Знак числа $+1 = (-1)^{\mathbf{0}}$, т.е. знаковый бит равен **0**

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

нормализованные числа с плавающей точкой

Запишем в том же восьмибитовом представлении число $\pi \approx 3.14159\ldots$
Ближайшая к π степень 2 есть 2^1, поэтому в представлении с плавающей точкой $\pi = 1.5708_{10} \times 2^1$.

Мантисса имеет четыре двоичных разряда. Если бы разряды были десятичными, то мантисса кодировала бы *десятитысячные* доли. У нас двоичное счисление, поэтому знаменатель дроби равен … $2^4 = 16$.

Мантисса $\pi/2 \approx 1.5708 \approx 1\frac{9}{16}$, откуда получаем последние 4 бита (***M***):

0	1	0	0	1	0	0	1

Стандарт C99 (поддерживается gcc) вводит формат %a :

```
printf("pi=%a\n", 3.14);
>pi=0x1.91eb86p+1
```

$\left(1\frac{9}{16}\right)_{10} = 1.1001_2$ (единицу в мантиссе нормализованного числа опускаем)

Смещённый порядок $E_S = E + (2^{w-1} - 1) = 1 + (2^{3-1} - 1) = 1 + 3 = \mathbf{4}$

Знак числа $+1 = (-1)^{\mathbf{0}}$, т.е. знаковый бит равен **0**

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

денормализованные числа с плавающей точкой

Какое самое маленькое по модулю нормализованное число?

(число вида $(-1)^S \cdot 1.M \cdot 2^E$) $\qquad n_1 = 1.00\ldots00 \cdot 2^{E_{\min}}$

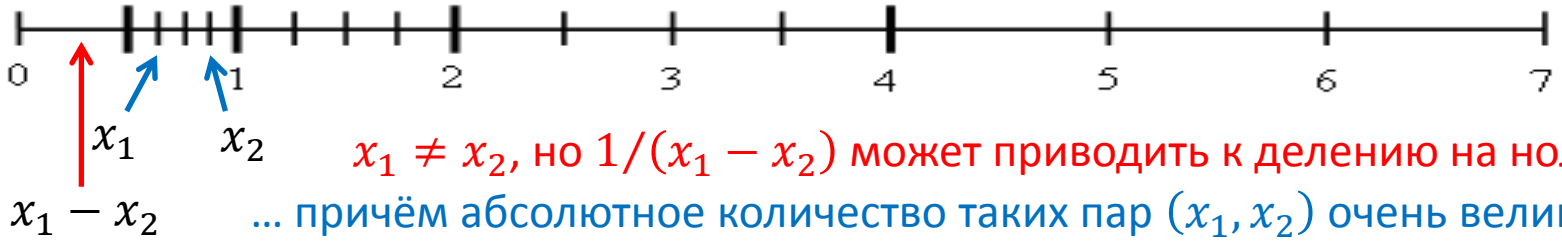
В нашем примере $w = 3$ и $n_1 = 1.0000 \cdot 2^{-3} = 1/8 = 0.125_{10}$.

Для $w = 8$ имеем $n_1 = 2^{-127} \approx 5.877 \cdot 10^{-39}$.

Число, *следующее за минимальным*: $n_2 = 1.00\ldots01_2 \cdot n_1 = 0.1328_{10}$ для $w = 3$.

Для $w = 8,\ n = 23\ :\ n_2 = (1 + 2^{-23}) \cdot 2^{-127} \approx (1 + 1.192 \cdot 10^{-7}) \cdot n_1$.

Видно, что интервал между минимальным (n_1) и следующим за ним (n_2) нормализованным числом в 2^n раз меньше минимального числа ($n_2 - n_1 \ll n_1$), т.е. в окрестности нуля наблюдается «дырка» (underflow gap). Ещё хуже то, что *ноль непредставим в нормализованном виде*.



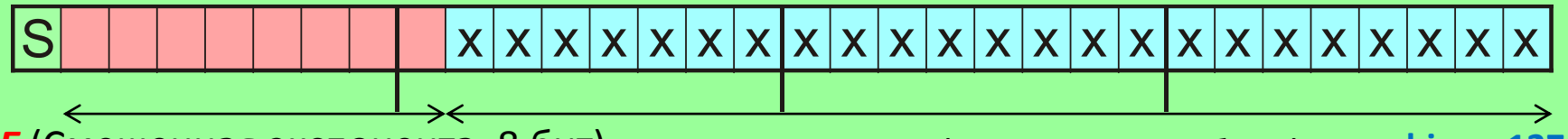
$x_1 \neq x_2$, но $1/(x_1 - x_2)$ может приводить к делению на ноль!

… причём абсолютное количество таких пар (x_1, x_2) очень велико!

Для решения этих проблем введены *денормализованные* (subnormal) числа.

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

нормализованные и денормализованные числа с плавающей точкой IEEE 754



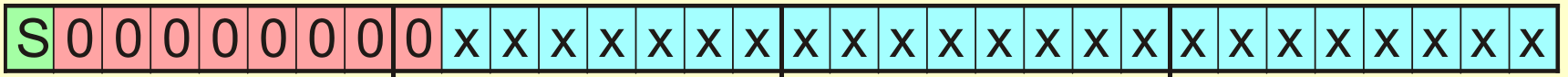
E (Смещенная экспонента, 8 бит)

Мантисса ***M*** (Мантисса, 23 бита)

bias = 127

Формула для нормализованных чисел $x_{\text{norm}} = (-1)^S \cdot (1 + M/2^n) \cdot 2^{E+1-2^{w-1}}$

$(1 \le E \le 2^w - 2)$



Для денормализованных чисел ($E = 0$): $x_{\text{subnorm}} = (-1)^S \cdot (M/2^n) \cdot 2^{2-2^{w-1}}$

NB: в старшем разряде теперь 0 , а не 1!

При $E = 0$ и $M = 0$ получаем ± 0 (*есть два нуля, «+0» и «–0»!*)

При $E = 2^w - 1 = 11\ldots1_2$ и $M = 0$ получаем $\pm\infty$ (*переполнение*; $x/0$)

При $E = 2^w - 1 = 11\ldots1_2$ и $M \neq 0$ – неопределенность, NaN (Not-a-Number)

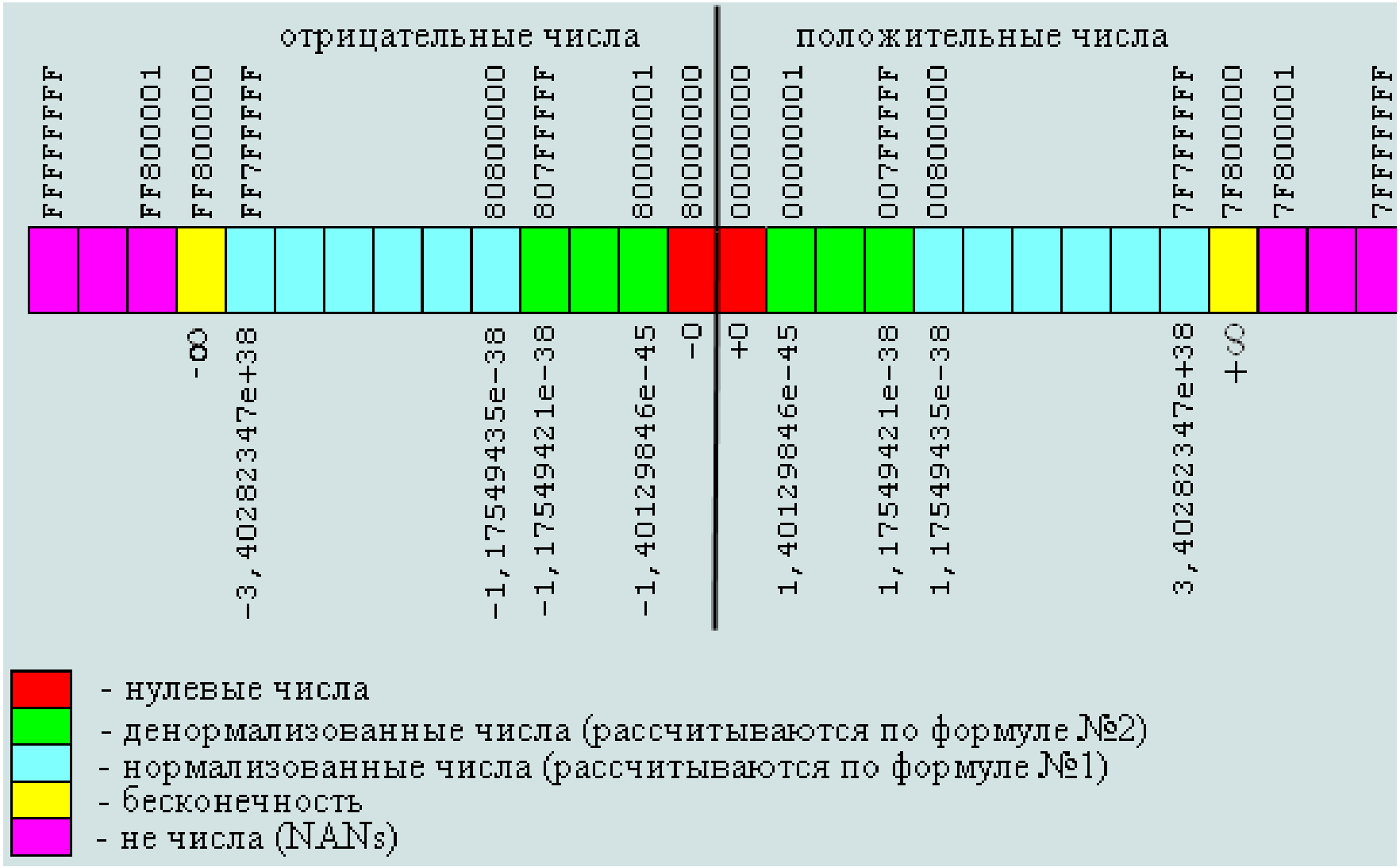
Неопределенность (NaN) получается в одном из следующих случаев:

- $\infty + (-\infty)$,
- $0 \times \infty$,
- $0/0$, ∞/∞,
- `sqrt(x)` при $x < 0$

Особенности: • NaN != NaN • избыточность кода NaN • quiet & signaling NaNs

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

нормализованные и денормализованные числа с плавающей точкой IEEE 754



ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

нормализованные и денормализованные числа с плавающей точкой IEEE 754

IEEE Std 754-2008
IEEE Standard for Floating-Point Arithmetic

3.4 Binary interchange format encodings

Each floating-point number has just one encoding in a binary interchange format. To make the encoding unique, in terms of the parameters in 3.3, the value of the significand m is maximized by decreasing e until either $e = emin$ or $m \geq 1$. After this process is done, if $e = emin$ and $0 < m < 1$, the floating-point number is subnormal. Subnormal numbers (and zero) are encoded with a reserved biased exponent value.

Representations of floating-point data in the binary interchange formats are encoded in k bits in the following three fields ordered as shown in Figure 3.1:

a) 1-bit sign S

b) w-bit biased exponent $E = e + bias$

c) $(t = p - 1)$-bit trailing significand field digit string $T = d_1 d_2 \ldots d_{p-1}$; the leading bit of the significand, d_0, is implicitly encoded in the biased exponent E.

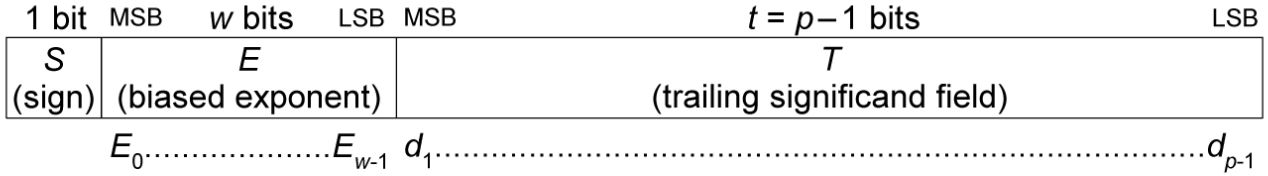


Figure 3.1—Binary interchange floating-point format

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ в ЭВМ

нормализованные и денормализованные числа с плавающей точкой IEEE 754

IEEE Std 754-2008
IEEE Standard for Floating-Point Arithmetic

Table 3.5—Binary interchange format parameters

Parameter	binary16	binary32	binary64	binary128	binary{*k*} ($k \geq 128$)
k, storage					
p, precisi					+ 13
emax, ma					
bias, *E* –					
sign bit					
w, exponent field width in bits	5	8	11	15	round(4 × log2 (*k*)) – 13
t, trailing significand field width in bits	10	23	52	112	*k* – *w* – 1
k, storage width in bits	16	32	64	128	1 + *w* + *t*

К счастью, работа с числами с плавающей точкой реализована в настоящее время настолько хорошо, что можно почти никогда не задумываться, как это работает. Исключения составляют лишь случаи возможного переполнения и работа на пределе точности вычислений, в том числе сравнение двух чисел с плавающей точкой на равенство / неравенство.

ТОЧНОСТЬ / ПОГРЕШНОСТЬ ВЫЧИСЛЕНИЙ

Какова погрешность выполнения одной арифметической операции над числами с плавающей точкой?

Рассмотрим для примера умножение десятичных чисел **с плавающей точкой** с двумя знаками после запятой (*для простоты будем рассматривать десятичные числа*):

$7 \times \pi =?$ $\quad 7.00 \cdot 10^0 \times 3.14 \cdot 10^0 = 2.199 \cdot 10^1 \approx 2.20 \cdot 10^1$

Полученный ответ отличается от точного в третьем знаке мантиссы

Отбрасывание последней цифры за пределами точности вычислений называется *округлением*.

Округлите до целых следующие числа:

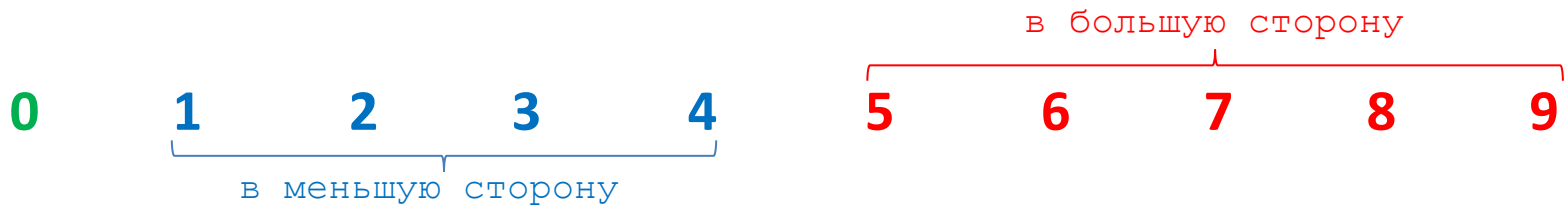
$3.14159 \ldots \approx 3$

$2.71828 \ldots \approx 3$

$2.50000 \ldots \approx \mathbf{2}$

ОКРУГЛЕНИЕ

Что было бы, если бы компьютер округлял числа так, как учат в школе?



Полагая возникновение различных цифр 0..9 в отбрасываемом разряде равновероятным, легко оценить математическое ожидание ошибки, допускаемой при округлении:

$$M = \frac{1}{10} \cdot \varepsilon' \cdot (0 - 1 - 2 - 3 - 4 + 5 + 4 + 3 + 2 + 1) = \frac{\varepsilon'}{2}$$

(здесь ε' – единица в отбрасываемом разряде, ULP/B).

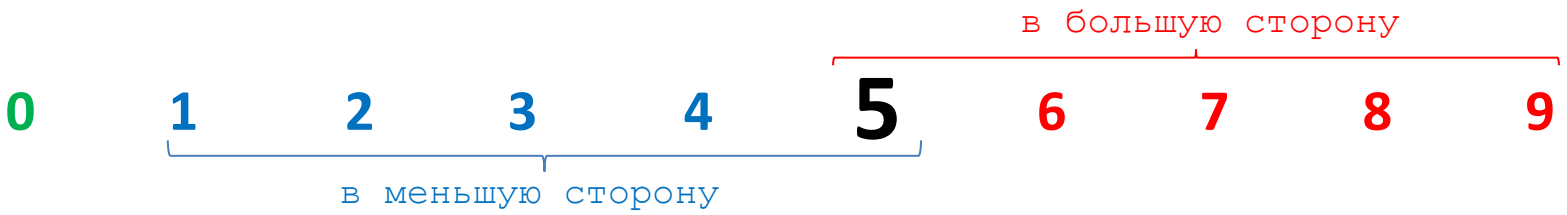
Очевидно, тот же самый ответ мы получим в любой системе счисления с чётным количеством цифр (в т.ч. и в двоичной системе).

$M > 0$ означает, что есть *систематическая погрешность* (*дрейф*).

Суммарная ошибка вычислений будет возрастать *линейно* с числом операций.

ОКРУГЛЕНИЕ

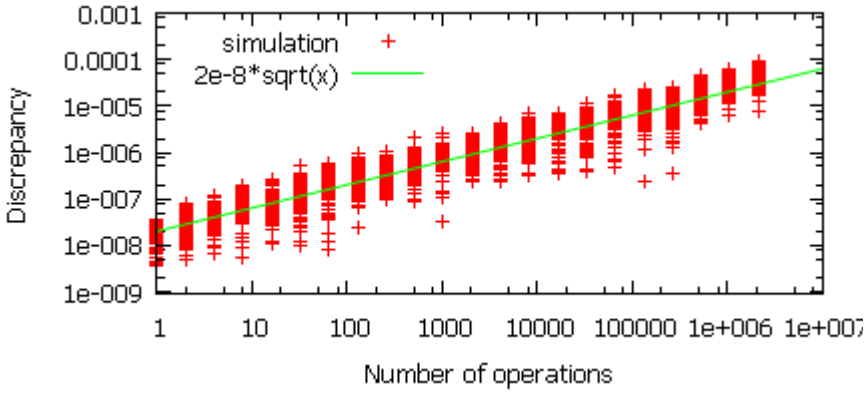
Более правильный способ округления («банковский»):

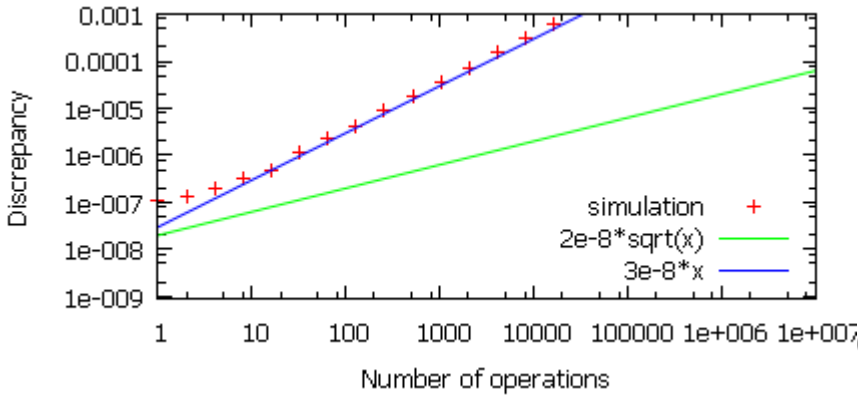


Если десятичную цифру 5_{10} (двоичную 1_2 и т.п.) округлять то в большую, то в меньшую в зависимости от чётности значения в предыдущем разряде, то математическое ожидание ошибки будет равно нулю:

$$M = \frac{1}{10} \cdot \varepsilon' \cdot \left(0 - 1 - 2 - 3 - 4 - 5 \cdot \frac{1}{2} + 5 \cdot \frac{1}{2} + 4 + 3 + 2 + 1\right) = 0$$

При этом суммарная ошибка вычислений будет возрастать не линейно, а пропорционально корню квадратному от числа операций, т.е. *значительно медленнее*.





ТОЧНОСТЬ / ПОГРЕШНОСТЬ ВЫЧИСЛЕНИЙ

Точность вычислений удобно характерозвать т.н. *машинным эпсилон* – наименьшим числом, которое можно прибавить к единице, чтобы получилось число, отличное от единицы. Другими словами, ε – единица в младшем разряде (ULP, *unit in the last place*, или *unit of least precision*)

Задача. Написать на языке Си программу, которая бы вычисляла машинное эпсилон, последовательно прибавляя к 1 числа 1, 0.5, 0.25 и т.д. и выводила бы последнее число, результат сложения которого с 1 был бы больше 1, а также количество двоичных разрядов для типов данных одинарной и двойной точности `float` и `double`.

Пример с калькулятором / gnuplot

ТОЧНОСТЬ / ПОГРЕШНОСТЬ ВЫЧИСЛЕНИЙ

РЕЗЮМЕ

- множество чисел, с которыми работает ЭВМ, конечно
- ∃ min и max число, остальные отображаются в $\pm\infty$

- помимо чисел и $\pm\infty$, есть неопределённость (NaN): $0/0$, ∞/∞, $\infty + (-\infty)$
- компьютер работает с двоичными числами $(0.1_{10} + 0.2_{10} \approx 0.3_{10})$
- точность вычислений на компьютере ограничена
- ***относительную*** погрешность удобно характеризовать машинным ε (ULP)
- числа с плавающей точкой, как правило, имеет смысл сравнивать лишь с точностью до некоторой погрешности
- арифметические операции на компьютере не обладают ассоциативностью, т.е. $(a + b) + c \neq a + (b + c)$. Например, $(10^{-20} + 1) - 1 \neq 10^{-20} + (1 - 1)$ Наибольшая потеря точности происходит при вычитании. Порядок операций нужно выбирать, помня о точности вычислений.
- для вывода чисел на экран удобно использовать `printf("%f", M_PI);`
  %e - в формате с плавающей точкой, %.10e - 10 знаков после запятой

ЛИТЕРАТУРА

По численным методам:

Смирнов С.В. Основы вычислительной физики. Часть I. НГУ, Новосибирск, 2015

Калиткин Н.Н. Численные методы. Наука, М. 1978

Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P.
Numerical recipes The art of scientific computing.
Cambridge University Press, New York, USA. 2007.

По языкам C/C++ :

Керниган Б.У., Ритчи Д.М. Язык программирования С. «Вильямс», М., 2007.

http://cppreference.com/ и http://www.cplusplus.com/