Theorem

Let $b \in \mathbb{N} \setminus \{0, 1\}$. Every $x \in \mathbb{R}$ has at least one base-b representation

$$x = \pm \sum_{-\infty}^{n=k} d_n \, b^n \quad \text{where digits} \quad d_n \in \{0, ..., b-1\}.$$

$$20.236 = 2 \cdot 10^1 + \frac{2}{10} + \frac{3}{100} + \frac{6}{1000}$$

$$(10.0110101)_2 = 2 + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} + \frac{1}{128} = 2.4140625$$

$$(0.01010101....\overline{01})_2 = \sum_{n=1}^{\infty} \frac{1}{2^{2n}} = \sum_{n=1}^{\infty} \frac{1}{4^n} = \frac{1}{1 - \frac{1}{4}} - 1 = \frac{1}{3}$$

$$(0.0\,\overline{0011})_2 = 0.1$$

For $x \in \mathbb{Q}$, the base-b expansion is either *terminating* or *ultimately periodic*.

Theorem

Let $x = p/q$ be reduced with $q > 0$. Then the base-b expansion of x terminates if and only if every prime divisor of q also divides b.

$$\forall \text{ primes } r: \quad r \mid q \implies r \mid b.$$

- $b = 2:\ x = \dfrac{p}{q}$ terminates $\Leftrightarrow q = 2^n$
- $b = 10:\ x = \dfrac{p}{q}$ terminates $\Leftrightarrow q = 2^n \cdot 5^m$

Uniqueness

- For every $b \in \mathbb{N} \setminus \{0, 1\}$, a real number has at most two base-b expansions.
  - Irrational numbers have a unique expansion.
  - Non-terminating rational numbers have a unique ultimately periodic expansion.
  - Nonzero terminating rational numbers have a second expansion using a repeating $b - 1$ tail.

Uniqueness

Every rational number has a recurring expansion in every base b:

$$0.237 = 0.236\overline{9}$$

$$(0.01101)_2 = (0.01100\overline{1})_2$$

Recipe: Decrease the last digit by 1. Then add $\overline{b-1}$.

Proof:

$$\sum_{n=1}^{\infty} \frac{b-1}{b^n} = (b-1)\left[\frac{1}{1-\frac{1}{b}} - 1\right] = 1$$

$$(0.\overline{1})_2 = (0.\overline{2})_3 = (0.\overline{7})_8 = (0.\overline{9})_{10} = (0.\overline{f})_{16} = 1$$

Floating-point numbers

- Shorter: machine numbers — German: [Fließ|Gleit]-[punkt|komma]zahlen

What scientific computing wants:

1. **Fast arithmetic** with fixed-size numbers, such as 64-bit representations.
2. **Representation of very large and very small numbers** spanning many orders of magnitude from atomic to astronomical.
3. **Standardisation across platforms**, i.e., diverse processors, machines, and programming languages.
4. **Robust handling of exceptions** like overflows, underflows, and invalid operations without crashing the program.
5. **Mathematically well-defined arithmetic and rounding rules**, allowing one to analyse algorithm behaviour, stability, and error propagation.

IEEE 754 Standard for Floating-Point Arithmetic

Floating-point numbers

- Fixed point numbers – with a fixed number of fractional digits – are bad for most applications (Exception: Finance/Taxes).
- Basic idea: split number in mantissa of fixed length and an exponent

$$x = \pm m \times b^e$$

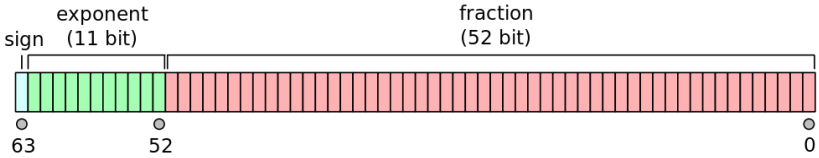
Normal form

$b = 10$	34.52467×10^4	$\underline{3.452467 \times 10^5}$	0.3452467×10^6	0.03452467×10^7
$b = 2$	10.011101×2^5	$\underline{1.0011101 \times 2^6}$	0.10011101×2^7	0.010011101×2^8

- The form with *one non-zero digit to left of the decimal point* is the normal form, $1 \leq m < b = (10)_b$.
- Trick: If base $b = 2$, this non-zero digit is *always* a 1.
- We only store M where

$$m = 1 + M.$$

Float64 according to IEEE 754



- Sign bit S
- 11 bits for the stored exponent $0 \leq E \leq 2047$, exponent shift $\sigma = 1023$
- 52 bits for the mantissa part M with $0 \leq M < 1$

normal, sub-normal and _special_ floating-point numbers

normal: $0 < E < E_{\text{max}}$ _sub-normal:_ $E = 0$ _special:_ $E = E_{\text{max}}$

normal: $$x = (-1)^S \cdot (1 + M) \cdot 2^{E-1023}$$

Source: Codekaizen, CC BY-SA 4.0, via Wikimedia Commons

```
julia> x = 27.56640625; bitstring(x)
"0100000000111011100100010000000000000000000000000000000000000000"
julia> printbitsf64(x)
0100000000111011100100010000000000000000000000000000000000000000
```

$$
\begin{aligned}
S &= 0 \\
E &= (10000000011)_2 = 2^{10} + 2^1 + 2^0 = 1027 \\
M &= (.1011100100001)_2 = \frac{1}{2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^{12}} \\
x &= (-1)^S \cdot (1 + M) \cdot 2^{E-1023}
\end{aligned}
$$

```
julia> x = (1 + 1/2 + 1/8 + 1/16 + 1/32 + 1/256 + 1/4096) * 2^4
27.56640625
```

```
function printbitsf64(x::Float64)
    s = bitstring(x)
    printstyled(s[1], color = :blue, reverse=true)
    printstyled(s[2:12], color = :green, reverse=true)
    printstyled(s[13:end], color=:red, bold=true, reverse=true)
    print("\n")
end
```

Definition (floating-point numbers)

Let m be the number of mantissa bits, e the number of exponent bits and σ the exponent shift.
The *set of floating-point numbers* (machine numbers) $\mathbb{M}_{m,e,\sigma}$ is defined as

$$\mathbb{M}_{m,e,\sigma} := \mathbb{M}^{\text{normal}}_{m,e,\sigma} \cup \mathbb{M}^{\text{sub}}_{m,e,\sigma} \cup \mathbb{M}^{\text{special}}$$

The *normal numbers* $\mathbb{M}^{\text{normal}}_{m,e,\sigma} \subset \mathbb{R}$ are

$$\mathbb{M}^{\text{normal}}_{m,e,\sigma} := \left\{\pm(1.b_1 b_2 \dots b_m)_2 \cdot 2^{E-\sigma}\right\} \text{ where } 0 < E < 2^e - 1.$$

The *sub-normal numbers* $\mathbb{M}^{\text{sub}}_{m,e,\sigma} \subset \mathbb{R}$ are

$$\mathbb{M}^{\text{sub}}_{m,e,\sigma} := \left\{\pm(0.b_1 b_2 \dots b_m)_2 \cdot 2^{1-\sigma}\right\} \text{ with } E = 0.$$

The *special numbers* $\mathbb{M}^{\text{special}} \not\subset \mathbb{R}$ are

$$\mathbb{M}^{\text{special}} := \{\infty, -\infty, \text{NaN}\} \text{ with } E = 2^e - 1 = (111 \dots 1)_2.$$

NaN is a special symbol representing “not a number”, essentially an error flag.

With the sign bit, floating-point numbers are stored in $1 + m + e$ bits.

Definition (IEEE standard machine numbers)

IEEE has 3 standard floating-point formats: 16-bit *(half precision)*, 32-bit *(single precision)* and 64-bit *(double precision)* defined by:

$$\mathbb{M}_{16} := \mathbb{M}_{10,5,15}$$
$$\mathbb{M}_{32} := \mathbb{M}_{23,8,127}$$
$$\mathbb{M}_{64} := \mathbb{M}_{52,11,1023}$$

- In Julia they are provided by the types `Float16`, `Float32` and `Float64`.
- `1.0`, `1.0e0`, `1e0`, `.1e1` are all `Float64`s equal to one.
- `1.0f0`, `1f0`, `.1f1` are of type `Float32`.

The (non-special) machine numbers are a *finite, discrete* subset of $\mathbb{R}$.
There is a biggest and a smallest one and besides that, they all have a *successor* and a *predecessor*.

```
julia> x = 27.56640625; printbitsf64(x)
0100000000111011100100010000000000000000000000000000000000000000
julia> succ_x = parse(UInt64,
         "0100000000111011100100010000000000000000000000000000000000000001", base=2)
0x403b910000000001
julia> reinterpret(Float64, succ_x)
27.566406250000004
```

Again, Julia has a built-in function for this.

```
julia> nextfloat(x)
27.566406250000004
julia> y = prevfloat(x)
27.566406249999996
printbitsf64(y)
0100000000111011100100001111111111111111111111111111111111111111
```

The set of normal machine numbers for `Float64`

- In the interval $[1, 2)$ there are $2^{52} \approx 4.5 \cdot 10^{15}$ equidistant machine numbers.
- Afterward, the exponent increases by 1 and the mantissa M is reset to 0. Consequently, the interval $[2, 4)$ again contains 2^{52} equidistant machine numbers, as does the interval $[4, 8)$ up to $[2^{1023}, 2^{1024})$.
- Similarly, the intervals $\left[\frac{1}{2}, 1\right)$, $\left[\frac{1}{4}, \frac{1}{2}\right)$, ... each contain 2^{52} equidistant machine numbers, down to $[2^{-1022}, 2^{-1021})$.
- This forms the set $\mathbb{M}^{\text{normal,+}}_{52,11,1023}$ of positive normal machine numbers, and it holds that

$$\mathbb{M}^{\text{normal}} = -\mathbb{M}^{\text{normal,+}} \cup \mathbb{M}^{\text{normal,+}}$$

Machine epsilon

Definition (machine epsilon)

The *machine epsilon* ϵ_m is the distance between $1 \in \mathbb{M}$ and its successor. In $\mathbb{M}_{m,e,\sigma}$ we have $\epsilon_m = 2^{-m}$.

```
julia> @show nextfloat(1.) - 1   2^-52   eps(Float64);
nextfloat(1.0) - 1 = 2.220446049250313e-16
2 ^ -52 = 2.220446049250313e-16
eps(Float64) = 2.220446049250313e-16
```

- The machine epsilon is a measure of the relative distance between machine numbers and quantifies the statement: "64-bit floating-point numbers have an accuracy of about 16 decimal places."
- Part of the literature uses a different definition of machine epsilon, which is half as large. $\epsilon' = \frac{\epsilon}{2} \approx 1.1 \times 10^{-16}$ is the maximum relative error that can occur when rounding a real number to the nearest machine number.
- Since numbers from the interval $[1 - \epsilon'/2, 1 + \epsilon']$ are rounded to the machine number 1, one can also define ϵ' as: ϵ' is *the largest number for which in machine number arithmetic it still holds that* $1 + \epsilon' = 1$.

floatmin, floatmax

- Remember:

$$\mathbb{M}_{m,e,\sigma}^{\text{normal}} := \left\{\pm(1.b_1 b_2 \ldots b_m)_2 \cdot 2^{E-\sigma}\right\} \text{ where } 0 < E < 2^e - 1.$$

- The smallest positive normal machine number has $E = 1$ and all $b_i = 0$.

$$x_{\text{min}} = 2^{1-\sigma}$$

- The largest finite machine number has $E = 2^e - 2$ and all $b_i = 1$.

$$x_{\text{max}} = 2^{2^e-2-\sigma}(2 - \epsilon_m)$$

```
julia> floatmin(Float64)
2.2250738585072014e-308
julia> floatmax(Float64)
1.7976931348623157e308
```

Don't confuse `floatmin` and ϵ_m!

Definition (Rounding)

A map $\mathcal{R} : \mathbb{R} \to \mathbb{M}$ with

$$\mathcal{R}(x) = x \quad \forall x \in \mathbb{M} \qquad \text{(projection)}$$
$$x \leq y \Longrightarrow \mathcal{R}(x) \leq \mathcal{R}(y) \quad \forall x, y \in \mathbb{R} \qquad \text{(monotonicity)}$$

is a *rounding* for the machine numbers $\mathbb{M}$.

Standard rounding mode: *round to nearest, ties to even*

- Round every $x \in \mathbb{R}$ to the nearest machine number.
- If x is exactly in the middle between two machine numbers `...0110110` and `..0110111`, take the one where the last mantissa bit is zero.

  Justification: this results in statistically rounding up in 50% of the cases and down in 50% of the cases, thus avoiding a "statistical drift" in longer calculations.

relative rounding error $$\left|\frac{x - \mathcal{R}(x)}{x}\right| \leq \frac{1}{2}\epsilon$$

$\mathbb{M}$ has no nice algebraic structure: it is not closed under standard addition, subtraction, etc.

Machine arithmetic provides the *rounded exact result*:

$$
\begin{aligned}
a \oplus b &:= \mathcal{R}(a + b) \\
a \ominus b &:= \mathcal{R}(a - b) \\
a \otimes b &:= \mathcal{R}(a \cdot b) \\
a \oslash b &:= \mathcal{R}(a / b)
\end{aligned}
$$

Addition and multiplication are commutative but *not associative!*

Non-associativity

```
julia> 1 +  10^-16 + 10^-16
1.0
julia> 1 +  (10^-16 + 10^-16)
1.0000000000000002
```

First case: left-to-right evaluation:

$$\begin{aligned} 1 \oplus 10^{-16} \oplus 10^{-16} &= (1 \oplus 10^{-16}) \oplus 10^{-16} \\ &= \mathcal{R}\left(\mathcal{R}(1 + 10^{-16}) + 10^{-16}\right) \\ &= \mathcal{R}(1 + 10^{-16}) \\ &= 1 \end{aligned}$$

Second case:

$$\begin{aligned} 1 \oplus \left(10^{-16} \oplus 10^{-16}\right) &= \mathcal{R}\left(1 + \mathcal{R}(10^{-16} + 10^{-16})\right) \\ &= \mathcal{R}(1 + 2 \cdot 10^{-16}) \\ &= 1.0000000000000002 \end{aligned}$$

Remember: even simple decimal numbers have no exact representation in $\mathbb{M}$.

$$(0.1)_{10} = (0.0001100110011001100110011001100...)_2 = (0.0001\overline{1001}100)_2$$
$$(0.3)_{10} = (0.0100110011001100110011001100..)_2 = (0.01\overline{0011}001)_2$$

```
printbitsf64(0.1)
printbitsf64(0.3)
0011111110111001100110011001100110011001100110011001100110011010
0011111111010011001100110011001100110011001100110011001100110011
```

Examples:

```
julia> 0.1 + 0.1 == 0.2
true
julia> 0.2 + 0.1 == 0.3
false
julia> 0.2 + 0.1
0.30000000000000004
julia> 1.1 + 1.2 + 1.3,   1.1 + (1.2 + 1.3)
(3.5999999999999996, 3.6)
```

sub-normal machine numbers

Sub-normal numbers live in the gap between 0 and the smallest positive normal number, $2^{1-\sigma}$.

Consider a simplified $\mathbb{M}$ model with base $b = 10$, $m = 3$ and exponents between -5 and 5.

- Remember the idea of normalisation.

$$33.333 \times 10^{-2} \qquad \underline{3.333 \times 10^{-1}} \qquad 0.333 \times 10^{0} \qquad 0.033 \times 10^{1} \qquad 0.003 \times 10^{2}$$

  Normalize machine numbers at each computational step $\Longrightarrow$ mantissa length fully used & accuracy preserved
- Smallest positive normal number is $x = 1.000 \times 10^{-5}$.
- With only normal numbers, already $x/2$ must be rounded to 0.
- Allow sub-normal numbers and we can represent $x/2$ as 0.500×10^{-5} and $x/200$ as 0.005×10^{-5}.
- *Gradual underflow* comes with less and less valid digits/precision.
- So, the smallest positive sub-normal number is $2^{1-\sigma-m}$.

Special values: $\pm\infty$ and NaN

An overflow results in $\pm\infty$:

```
julia> 2/0,  -3/0,  floatmax(Float64) * 1.01,  exp(1300)
(Inf, -Inf, Inf, Inf)
```

This doesn't stop our program:

```
julia> -Inf + 20,  Inf/30,  23/-Inf,  sqrt(Inf),  Inf * 0,  Inf - Inf
(-Inf, Inf, -0.0, Inf, NaN, NaN)
```

`NaN` (Not a Number) is the result of an ill-defined operation. All further computations with `NaN` result in `NaN`

```
julia> 0/0,   -Inf + Inf,   2.3NaN,   sqrt(NaN)
(NaN, NaN, NaN, NaN)
```

Conversion

1. String ⇒ number

```
julia> parse(Int64, "1101", base=2)
13
julia> parse(Int64, "1101", base=3)
37
```

2. Number ⇒ String

```
julia> string(13, base=2),  string(1/7)
("1101", "0.14285714285714285")
julia> string(77, base=16)
"4d"
```

3. Types of numbers: use the type name as constructor

```
julia> x = Int8(67);  @show x   typeof(x);
x = 67
typeof(x) = Int8
julia> z = UInt64(3459)
0x0000000000000d83
julia> y = Float64(z)
3459.0
```