Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

MCS 471 Lecture 2
Numerical Analysis
Jan Verschelde, 24 August 2022

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

Numerical Analysis – a definition

Definition (Nick Trefethen, SIAM News 1992)

Numerical analysis is the study of algorithms for the problems of continuous mathematics.

An algorithm is a finite number of unambiguous steps, where each step can be executed by arithmetical operations.

We care for the efficiency and accuracy of algorithms.

In continuous models to solve problems,
we obtain approximate answers for approximate input data.

Two related disciplines:

- Computer Algebra to formulate and re-formulate problems.
- Scientific Computing, for applications to science.

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

sources of error

Some sources of error are

- truncation errors in mathematical models;
- observed input data are approximate numbers;
- representation errors, e.g.: $1/10$ in binary, $1/3$ in decimal;
- roundoff error during calculations.

In numerical analysis, we ask two important questions:

1. How sensitive is the output to changes in the input?
2. Do roundoff errors in an algorithm propagate?

Answers to these two questions, are addressed respectively by

1. numerical conditioning is a property of a problem;
2. numerical stability is a property of an algorithm.

absolute and relative error

Definition (absolute error)

Let $\widehat{x}$ be an approximation for x. The *absolute error* Δx is the absolute value of the difference of x with $\widehat{x}$:

$$\Delta x = |x - \widehat{x}|.$$

Definition (relative error)

Let $\widehat{x}$ be an approximation for x. The *relative error* δx is the absolute error divided by the absolute value of x:

$$\delta x = \frac{\Delta x}{|x|}.$$

scientific notation of numbers

Consider the following three numbers:

```
-1.88260093354220 41e-18

1.6324136076090274e-16

3.5505687098878179e-17
```

They look completely different. What do they have in common?

Well, they represent respectively the numbers

$$-1.88 \times 10^{-18}, \quad 1.63 \times 10^{-16}, \quad 3.55 \times 10^{-17},$$

and those three numbers are very tiny.

We represent all real numbers in scientific notation.

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

floating-point numbers

A floating-point number consists of

1. one sign bit,
2. a normalized fraction: the leading bit is nonzero, and
3. an exponent.

Definition

The *floating-point representation* $fl(x)$ of a real number $x \in \mathbb{R}$ is

$$fl(x) = \pm .bbb \ldots b \times 2^e,$$

stored compactly as the tuple $(\pm, e, bbb \ldots b)$.
The *representation error* is $|fl(x) - x|$.

floating-point formats

Hardware supports single precision (32-bit), double precision (64-bit), and long double precision (80-bit), summarized below:

	number of bits			
precision	sign	exponent	fraction	total
single	1	8	23	32
double	1	11	52	64
long double	1	15	64	80

A 64-bit floating-point number has

- 1 sign bit s, 0 for positive, 1 for negative,
- 11 bits $e_1, e_2, \ldots, e_{11}$ in the exponent, and
- 52 bits $f_1, f_2, \ldots, f_{52}$ in the fraction, $f_1 \neq 0$.

s	e_1	e_2	$\cdots$	e_{11}	f_1	f_2	$\cdots$	f_{52}

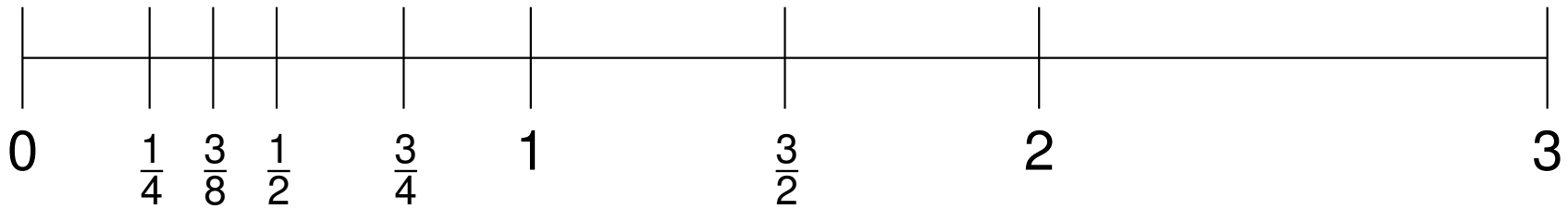
a number line

Consider a floating-point number system with basis 2

1. with two bits in the (normalized) fraction, and
2. with exponents -1, 0, $+1$, $+2$.

We display all positive floating-point numbers in this system:

$$
\begin{array}{llll|llll}
.10\,2^{-1} & = & 0.01 & = & 1/4 & .11\,2^{-1} & = & 0.011 & = & 3/8 \\
.10\,2^{0} & = & 0.1 & = & 1/2 & .11\,2^{0} & = & 0.11 & = & 3/4 \\
.10\,2^{+1} & = & 1 & & & .11\,2^{+1} & = & 1.1 & = & 3/2 \\
.10\,2^{+2} & = & 10 & = & 2 & .11\,2^{+2} & = & 11 & = & 3
\end{array}
$$



error $|f\ell(x) - x| \leq 1/8$ error $|f\ell(x) - x| \leq 1/2$

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

machine precision

Definition

The number *machine precision* ϵ_{mach} is the distance between 1 and the smallest floating-point number greater than one.
For basis B and size p of the fraction: $\epsilon_{\text{mach}} = B^{-p}$.

For $0 < \epsilon < \epsilon_{\text{mach}}$: $(1 + \epsilon) - 1 \neq \epsilon + (1 - 1)$.

The machine precision as supported by hardware single floats (32-bit), double floats (64-bit), and long double floats (80-bit) is below:

precision	number of bits				machine precision
	sign	exponent	fraction	total	
single	1	8	23	32	$2^{-23} \approx$ 1.192e-07
double	1	11	52	64	$2^{-52} \approx$ 2.220e-16
long double	1	15	64	80	$2^{-64} \approx$ 5.421e-20

the smallest and largest exponent

An exponent $e \in [e_{\min}, e_{\max}]$ where
$e_{\min}$ is the smallest exponent and $e_{\max}$ is the largest exponent.

precision	number of bits				exponent range	
	sign	exponent	fraction	total	$e_{\min}$	$e_{\max}$
single	1	8	23	32	-126	$+127$
double	1	11	52	64	-1022	$+1023$
long double	1	15	64	80	-16382	$+16383$

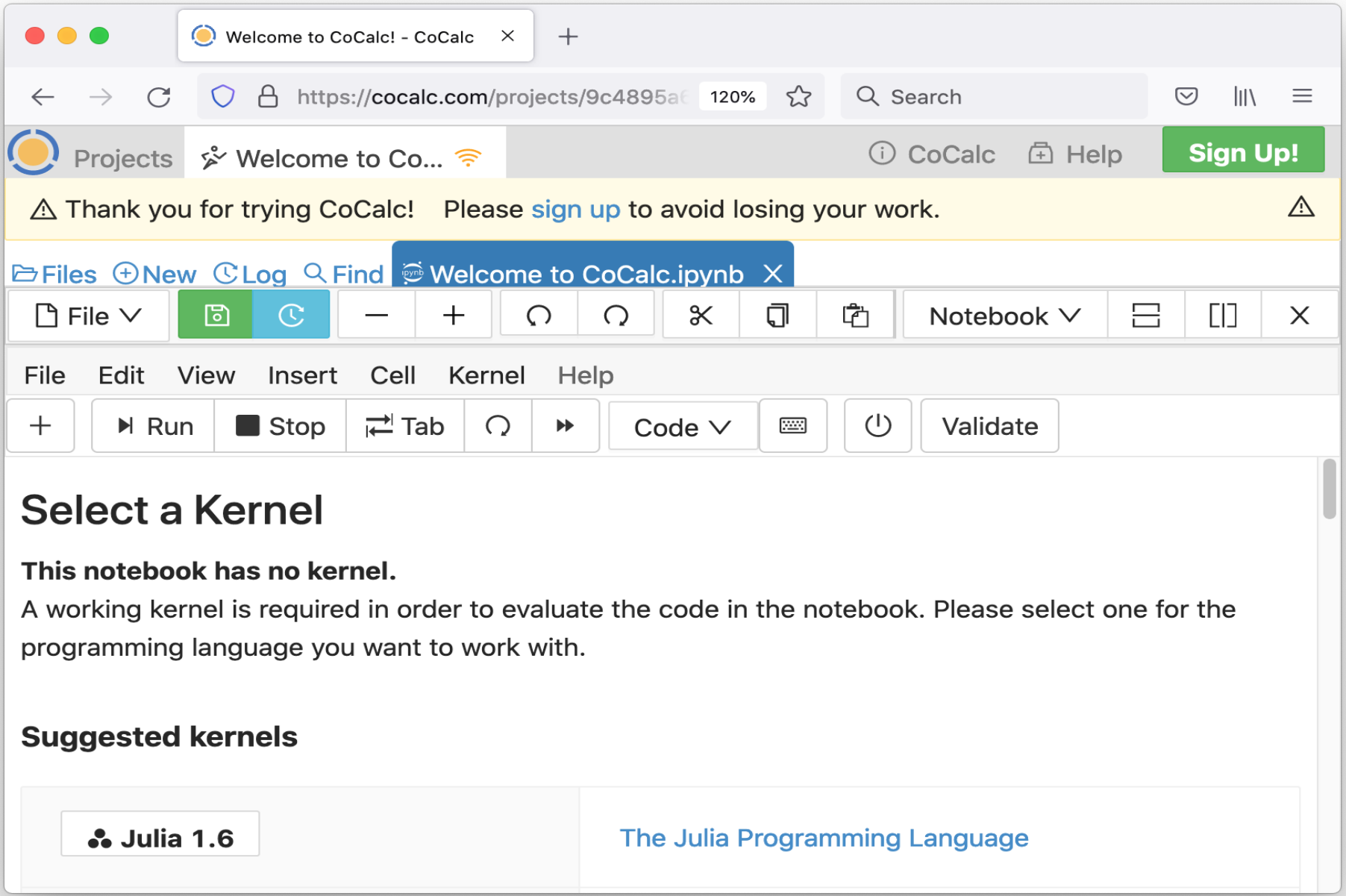
Special values for the exponent for double precision:

- `111 1111 1111`, nonzero fraction : `-NaN`, not a number;
- `111 1111 1111`, zero fraction : `-Inf`, represents $-\infty$;
- `000 0000 0000` : numbers that are not normalized;
- `011 1111 1111`, zero fraction : `+Inf`, represents $+\infty$.

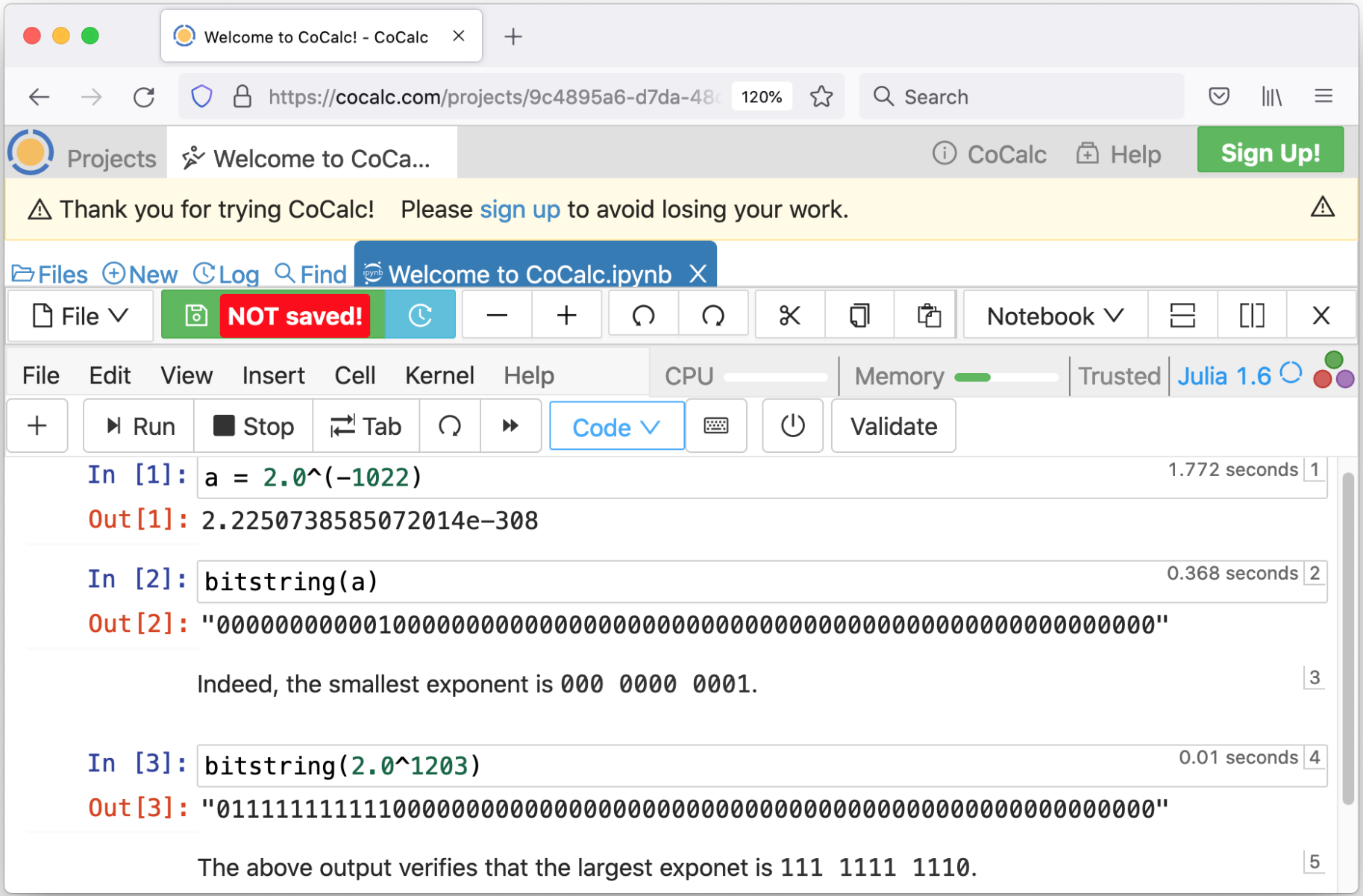
Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

welcome to CoCalc!



documenting calculations in a notebook



exponent encoding

The exponents are encoded with an offset, minus 1023 for double:

```
julia> a = 2.0^(-1022)
2.2250738585072014e-308

julia> bitstring(a)
"0000000000010000000000000000000000000000000000000000000000000000"
```

We see that the smallest exponent is `000 0000 0001`.

```
julia> b = 2.0^1023
8.98846567431158e307

julia> bitstring(b)
"0111111111100000000000000000000000000000000000000000000000000000"
```

We see that the largest exponent is `111 1111 1110`.

the smallest and largest double

```
julia> a = nextfloat(0.0)
5.0e-324

julia> bitstring(a)
"0000000000000000000000000000000000000000000000000000000000000001"
```

The smallest number is not normalized: $2^{-52} \times 2^{-1022} = 2^{-1074}$.

```
julia> b = prevfloat(Inf)
1.7976931348623157e308

julia> bitstring(b)
"0111111111101111111111111111111111111111111111111111111111111111"
```

The largest number has exponent $2^{10} - 1$ and fraction $1 + (1 - 2^{-52})$.

```
julia> bitstring(2.0^1023*(1 + (1 - 2.0^(-52))))
"0111111111101111111111111111111111111111111111111111111111111111"
```

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

adding two floating-point numbers

Consider two numbers in a system with 4 as the size of fraction:

$$x = +.1101 \times 2^3 \text{ and } y = +.1011 \times 2^1.$$

Four steps to add two floating-point numbers:

1. Align the numbers so they both have the same exponent.
   $y = +.1011 \times 2^1 = +.01011 \times 2^2 = +.001011 \times 2^3$
2. Perform the addition.

$$\begin{array}{rrcl} & +.1101 & \times & 2^3 \\ + & +.001011 & \times & 2^3 \\ \hline & +.111111 & \times & 2^3 \end{array}$$

3. Round the result: $x + y = +1.0000 \times 2^3$.
4. Normalize the result: $x + y = +.1000 \times 2^4$.

Exercise 1: check the accuracy of the sum.
Translate all operations into our familiar decimal notation.

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

loss of significance

Consider two numbers in a system with 4 as the size of fraction:

$$x = +.1110 \times 2^3 \text{ and } y = +.1101 \times 2^3.$$

Compute $x - y$:

$$\begin{array}{rrcl} & +.1110 & \times & 2^3 \\ - & +.1101 & \times & 2^3 \\ \hline & +.0001 & \times & 2^3 \end{array}$$

After normalization: $x - y = +.1000 \times 2^0$.

Problem: x and y have 4 bits of significance,
the result $x - y$ has only one significant bit of accuracy.

restructuring a calculation

Consider $\sqrt{9.01} - 3$ in a three decimal digit number system.

In this system, 3 is represented by $+.300 \times 10^1$.
$\sqrt{9.01} \approx 3.0016662$ represented by $+.300 \times 10^1$.
The subtraction will thus yield zero.

We can avoid the subtraction:

$$\sqrt{9.01} - 3 = \frac{(\sqrt{9.01} - 3)(\sqrt{9.01} + 3)}{\sqrt{9.01} + 3} = \frac{9.01 - 9}{\sqrt{9.01} + 3}$$

The difference in the numerator is not zero:

$+.901 \times 10^1$ minus $+.900 \times 10^1$ yields $+.100 \times 10^{-1}$.

Dividing $+.100 \times 10^{-1}$ by $\sqrt{9.01} + 3 = +.600 \times 10^1$
results in $+.167 \times 10^{-2}$.

two additional exercises

Exercise 2: Consider the representation of floating-point numbers with base 10 and 2 digits in the fraction part.
The values for the exponents are between -10 and $+10$.

1. What is the machine precision in this number system?
2. Represent the numbers 17 and 333 as floating point numbers and illustrate the calculation of $17 + 333$, using rounding. What is the calculated sum?

Exercise 3: Consider a floating-point number system with base 10.
There are five digits in the fraction. Exponents range from -7 to $+8$.

1. What is the smallest positive floating-point number in this system?
2. What is the result of $12.381 + 0.098321$ in this system?

Floating-Point Arithmetic

1. Numerical Analysis
   - a definition
   - sources of error
2. Floating-Point Numbers
   - floating-point representation of a real number
   - machine precision
   - a Julia session in CoCalc
3. Floating-Point Arithmetic
   - adding two floating-point numbers
   - loss of significance
4. Arbitrary Precision and Interval Arithmetic
   - extending floating-point arithmetic

extending floating-point arithmetic

Two ways to extend floating-point arithmetic:

1. arbitrary precision floating-point arithmetic

   The GNU Multiprecision Arithmetic Library and the GNU MPFR library provide arbitrary-precision integers and floating-point numbers, wrapped in Julia by the types `BigInt` and **BigFloat**. See the methods `precision()` and `setprecision()` to query the precision (in bits) and to set the precision (also in bits).

2. multiple double arithmetic
   A double double is an unevaluated sum of two doubles.
   Double double arithmetic exploits the hardware double arithmetic.

3. interval arithmetic

   Instead of one number, we can calculate with an interval $[a, b]$, where a is the lower and b the upper bound for the approximation.

bibliography

- J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres: **Handbook of Floating-Point Arithmetic**. Springer-Verlag, second edition, 2018.
- J. Demmel and J. Riedy: **A new IEEE 754 standard for floating-point arithmetic in an ever-changing world.** *SIAM News*, 54(6):9–11, 2021. Special Issue on Computational Science and Engineering.
- IEEE. **IEEE Standard for Floating-Point Arithmetic.** In IEEE Std 754-2019 (Revision of IEEE Std 754-2008), 84 pages, 2019.

The SIAM News article (second item above) is recommended reading.