

Lecture 02 - B

§1.3 Floating-Point Errors

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



Goals

We will answer these questions:

- What is overflow, underflow, and demormalization?
- What is the significance of machine epsilon?
- How does arithmetic on the computer work, and what do we need to be wary of?
- How many significant digits do computational variables have?
- How might significant digits be lost?



Overflow and Underflow



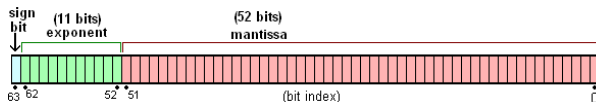
IEEE Double

Definition: IEEE Double Precision Float (FP64)

The **double precision float** has the form

$$x = \pm q \times 2^m.$$

It occupies 64 bits of memory. It is a specific standard format set by IEEE 755.



Question

What do the largest and smallest numbers look like?



FP64 Special Bit Sequences

Four Cases

- **Biggest finite number (realmax):**

$$\underbrace{0}_{\text{sign}} \mid \underbrace{1111111110}_{\text{exponent}} \mid \underbrace{1111 \dots 1111}_{52 \text{ ones}} \approx 1.7977 \times 10^{308}$$

- **Smallest normalized number (realmin):**

$$\underbrace{0}_{\text{sign}} \mid \underbrace{00000000001}_{\text{exponent}} \mid \underbrace{0000 \dots 0000}_{52 \text{ zeros}} \approx 2.2251 \times 10^{-308}$$

- **Signed Zero:**

$$\boxed{\pm} \mid \underbrace{00000000000 \mid 0000 \dots 0000}_{\text{all zeros}} = \pm 0$$

- **Signed Infinity:**

$$\boxed{\pm} \mid \underbrace{11111111111}_{\text{all ones}} \mid \underbrace{0000 \dots 0000}_{52 \text{ zeros}} = \pm \infty$$



Range

type	range	approx range
single	$-3.40 \times 10^{38} \leq x \leq -1.18 \times 10^{-38}$	$2^{-126} \rightarrow 2^{128}$
	0 $1.18 \times 10^{-38} \leq x \leq 3.40 \times 10^{38}$	
double	$-1.80 \times 10^{308} \leq x \leq -2.23 \times 10^{-308}$	$2^{-1022} \rightarrow 2^{1024}$
	0 $2.23 \times 10^{-308} \leq x \leq 1.80 \times 10^{308}$	

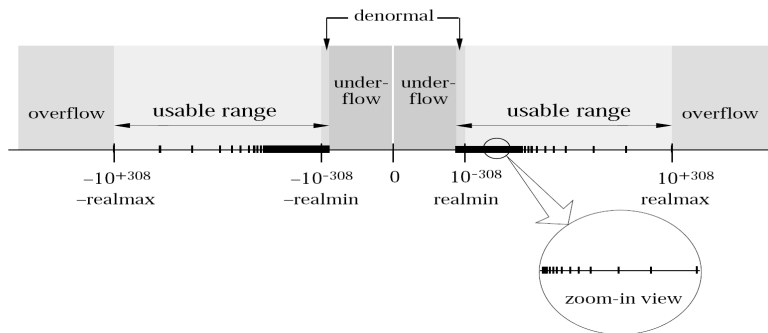
```
>> realmax
ans = 1.7977e+308

>> realmin
ans = 2.2251e-308
```

Question

What happens when you try to go beyond realmax or below realmin?

Floating Point Number Line



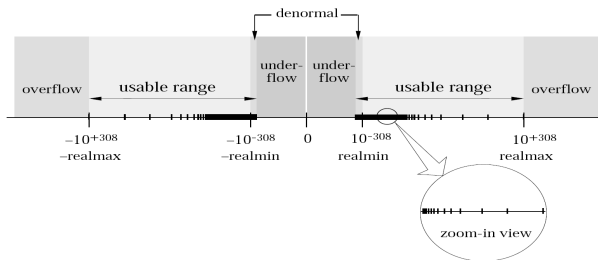
- **Overflow**: When a computation produces a value larger than realmax ,

$$x > \text{realmax} \implies \text{fl}(x) = \infty.$$



Underflow and Denormalization

Floating Point Number Line



- **Underflow:** When a value is too small

$$0 < |x| < \text{realmin},$$

then it may round down to 0 OR denormalize.

- **Denormalization:** The hidden bit becomes 0, allowing x to be have representable numbers below realmin .



Matlab Examples

- Overflow - number is set to "infinity"

```
1 >> realmax*realmax
2 ans = Inf
```

- Underflow to zero

```
1 >> realmin*realmin
2 ans = 0
```

- Denormalization below realmin

```
1 >> realmin/2
2 ans = 1.1125e-308 % not zero
3
4 >> realmin/256
5 ans = 8.6917e-311 % still not zero
6
7 >> realmin/2^53
8 ans = 0 % finally it's zero
```

Machine Epsilon



The Additive Identity

Question

What happens when we add zero to a number x ?



The Additive Identity

Question

What happens when we add zero to a number x ?

$$x + 0 = x$$

You get the same number x . Zero is called the additive identity.



The Additive Identity

Question

What happens when we add zero to a number x ?

$$x + 0 = x$$

You get the same number x . Zero is called the additive identity.

Question

If $x + \epsilon = x$, then what is ϵ ?



Adding Floating-Point Numbers

Procedure

- 1 Match the exponent of the smaller number to the larger number
- 2 Align the binary points
- 3 Add from right to left

Example

Add

$$x_1 = 1.11011 \times 2^1, \quad x_2 = 1.00111 \times 2^{-2}.$$

- First, match the exponents:

$$x_2 = 1.00111 \times 2^{-2} = 0.00100111 \times 2^1.$$

- Now the significands can be added:

$$\begin{array}{r} 1.11011000 \times 2^1 \\ + 0.00100111 \times 2^1 \\ \hline 1.11111111 \times 2^1 \end{array}$$



Limited Storage

We found the exact sum

$$x_1 + x_2 = 1.11111111 \times 2^1.$$

- Suppose our floating-point system only stores **5 binary digits**.
- The exact answer does not fit:

$$1.11111 \underbrace{111}_{\text{cannot store}} \times 2^1.$$

- If we truncate to 5 digits,

$$\text{fl}(x_1 + x_2) = 1.11111 \times 2^1.$$

Key Idea

Even if x_1 and x_2 can be stored exactly, their sum may not be representable exactly in the same floating-point system.

Adding in FP64

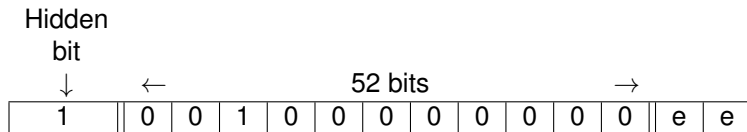
Example

What happens when you add $x_1 = 1$ and $x_2 = 1/8$ in FP64? What does the bit sequence look like?

- Let $x_1 = (1.0)_2 \times 2^0$
- Let $x_2 = 1/8 = (1.0)_2 \times 2^{-3}$
- $x_1 + x_2 =$

$$\begin{array}{r} 1.000 \times 2^0 \\ + 0.001 \times 2^0 \\ \hline 1.001 \times 2^0 \end{array}$$

- $x_1 + x_2$ as a bit sequence



Example

What happens when you add $x_1 = 1$ and $x_2 = 2^{-53}$ in FP64?

Let's start gradually. . . Add $1/2, 1/4, 1/8$, and so on:

Hidden bit	52 bits													
↓	←													→
1	1	0	0	0	0	0	0	0	0	0	0	0	e	e
1	0	1	0	0	0	0	0	0	0	0	0	0	e	e
1	0	0	1	0	0	0	0	0	0	0	0	0	e	e
.....														
1	0	0	0	0	0	0	0	0	0	0	0	1	e	e
1	0	0	0	0	0	0	0	0	0	0	0	0	e	e

- 2^{-53} doesn't fit in the mantissa!
- $\text{fl}(1 + 2^{-52}) \neq 1$, but $\text{fl}(1 + 2^{-53}) = 1$



ϵ_m – Machine epsilon



Key Idea

Adding $1 + 2^{-53}$ in Matlab results in 1. The quantity $\epsilon_m = 2^{-52}$ is the cutoff.

Importance

In Math

If

$$x + \epsilon = x$$

then ϵ **must be the additive identity**, a.k.a. zero.

On Computers

Even if

$$\text{fl}(x + \epsilon) = x,$$

ϵ may be **non-zero**. This is very bad!

Key Idea

Machine epsilon is the limit at which math breaks down.

Definition: Machine Epsilon

The **machine epsilon** ϵ_m is the smallest number such that

$$\text{fl}(1 + \epsilon_m) \neq 1,$$

where $\text{fl}(x)$ represents the floating point representation of x .

- In English,

“Machine epsilon is the smallest number that can be added to 1 without being lost to rounding.”

- For FP32: $\epsilon_m = 2^{-23} \approx 10^{-7}$.
- For FP64: $\epsilon_m = 2^{-52} \approx 10^{-16}$.



Machine epsilon in Matlab

- `eps` is a built-in variable

```
1 >> eps
2 ans = 2.2204e-16
3
4 >> eps('single')
5 ans = 1.1921e-07
```

- The following demonstrate machine epsilon rounding error

```
1 >> 1+eps == 1+eps % this is fine
2 ans = 1
3
4 >> 1+eps == 1 % this is fine
5 ans = 0
6
7 >> 1+eps/2 == 1 % trouble
8 ans = 1
9
10 >> 2+eps == 2 % trouble
11 ans = 1
```

Rounding Problem

Example

What is the smallest number a such that $\text{fl}(a + 32) \neq 32$ in FP64?

- a is so small relative to 32 that just barely avoids rounding
- Some use the notation $a = \epsilon(32)$ to describe the number a

$$\epsilon(32) = \epsilon_m \cdot |32| = 2^{-52} \cdot |x|$$

- Since $32 = 2^5$

$$a < 2^{-52} \cdot 32 = \frac{2^5}{2^{52}} = 2^{-47} \approx 7.1 \times 10^{-15}$$

- You can also do this in Matlab

```
1 >> eps(32)
2 ans = 7.1054e-15
3 >> 32 + eps(32) == 32 % false, eps(32) is not rounded off
```

Floating Point Rules



Review Error

Definition: Absolute Error

Absolute error, or usually just **error**, is

$$\text{Error}(x_a) = |x_t - x_a| = |\text{true value} - \text{approximate value}|$$

Definition: Relative Error

Relative error is

$$\text{Rel}(x_a) = \left| \frac{x_t - x_a}{x_t} \right| = \left| \frac{\text{true value} - \text{approximate value}}{\text{true value}} \right|$$

Definition: Percentage Error

Percentage error is

$$\left| \frac{x_t - x_a}{x_t} \right| \times 100\% = \left| \frac{\text{true value} - \text{approximate value}}{\text{true value}} \right| \times 100\%$$

Floating Point Rules

Rule 1.

$$\text{fl}(x) = x(1 + \epsilon), \quad \text{where} \quad |\epsilon| < \epsilon_m$$

Rule 2.

For all operations $\odot$ (one of $+$, $-$, $*$, $/$)

$$\text{fl}(x \odot y) = (x \odot y)(1 + \epsilon_{\odot}), \quad \text{where} \quad |\epsilon_{\odot}| < \epsilon_m$$

Rule 3.

For $+$, $*$ operations

$$\text{fl}(a \odot b) = \text{fl}(b \odot a)$$

There were many discussions on what conditions/ rules should be satisfied by floating point arithmetic. The IEEE standard is a set of standards adopted by many CPU manufacturers.



Relative Error for Floats

- The number $x \in \mathbb{R}$ stored as FP64 is

$$\text{fl}(x) = \underbrace{x}_{\text{the number}} \underbrace{(1 + \epsilon)}_{\text{close to 1}}$$

where $|\epsilon| \leq 2^{-52}$

- We can use the rule to find **relative error**

$$\begin{aligned}\text{Rel}(x) &= \frac{|true - approx|}{|true|} \\ &= \frac{|x - \text{fl}(x)|}{|x|} \\ &= \frac{|x - x(1 + \epsilon)|}{|x|} \\ &= |\epsilon| \leq 2^{-52}\end{aligned}$$



Absolute Error for Floats

Example

Use the floating point rules to find the **absolute error** between x and $\text{fl}(x)$.

- By rule 1, we have $\text{fl}(x) = x(1 + \epsilon)$, where $|\epsilon| < \epsilon_m$.
- The absolute error is

$$\begin{aligned}\text{Rel}(x) &= |\text{true} - \text{approx}| \\ &= |x - \text{fl}(x)| \\ &= |x - x(1 + \epsilon)| \\ &= |x\epsilon| \\ &\leq |x|\epsilon_m\end{aligned}$$

- This calculation says that the error between x and $\text{fl}(x)$ depends on the *magnitude of x*



Floating Point Arithmetic

Question

Consider adding the triple sum $y = a + b + c$ on the computer. How is this handled? What is the error?

Done as $\text{fl}(\text{fl}(a + b) + c)$

$$\begin{aligned}\eta &= \text{fl}(a + b) = (a + b)(1 + \epsilon_1) \\ y_1 &= \text{fl}(\eta + c) = (\eta + c)(1 + \epsilon_2) \\ &= [(a + b)(1 + \epsilon_1) + c](1 + \epsilon_2) \\ &= [(a + b + c) + (a + b)\epsilon_1](1 + \epsilon_2) \\ &= (a + b + c) \left[1 + \frac{a + b}{a + b + c} \epsilon_1 (1 + \epsilon_2) + \epsilon_2 \right]\end{aligned}$$

So disregarding the high order term $\epsilon_1 \epsilon_2$

$$\text{fl}(\text{fl}(a + b) + c) = (a + b + c)(1 + \epsilon_3) \quad \text{with} \quad \epsilon_3 \approx \frac{a + b}{a + b + c} \epsilon_1 + \epsilon_2$$



Addition Error for Floats

If we redid the computation as $y_2 = \text{fl}(a + \text{fl}(b + c))$ we would find

$$\text{fl}(a + \text{fl}(b + c)) = (a + b + c)(1 + \epsilon_4) \quad \text{with} \quad \epsilon_4 \approx \frac{b + c}{a + b + c} \epsilon_1 + \epsilon_2$$

The first error is amplified by the factor $(a + b)/y$ in the first case and $(b + c)/y$ in the second case. ($y = a + b + c$)

Key Idea

Order matters. To optimize accuracy, add numbers from least to greatest. However, sorting before adding is usually not worth the cost!



Error from Exponentiation

Question

What happens to floating point error when we compute x^n ?

On a computer, exponentiation is performed via repeated multiplication:

$$x^n = \underbrace{x \cdot x \cdot \dots \cdot x}_{n-1 \text{ times}}$$

Using the floating point rule for multiplication,

$$\text{fl}(x \cdot y) = (xy)(1 + \epsilon), \quad |\epsilon| \leq \epsilon_m$$

Each multiplication introduces a small relative error. After $n - 1$ multiplications, the computed result is

$$\text{fl}(x^n) = x^n(1 + \epsilon_1)(1 + \epsilon_2) \cdots (1 + \epsilon_{n-1})$$



Error Growth in Powers

Ignoring higher-order terms (products of ϵ_i),

$$(1 + \epsilon_1)(1 + \epsilon_2) \cdots (1 + \epsilon_{n-1}) \approx 1 + (\epsilon_1 + \epsilon_2 + \cdots + \epsilon_{n-1})$$

Therefore,

$$\text{fl}(x^n) = x^n(1 + \epsilon_n), \quad |\epsilon_n| \lesssim (n-1)\epsilon_m$$

- Relative error grows **linearly** with the exponent
- Large powers magnify rounding error
- This effect appears even if $|x| \approx 1$

Key Idea

Repeated operations accumulate error. Exponentiation amplifies floating point error as the power increases.

Significant Digits



Significant Digits

Definition: Significant Digits

Significant digits are digits in a number justified by the accuracy of the means of measurement.

Example: Measurement from a ruler which goes from 0.0 to 9.0 (in cm):

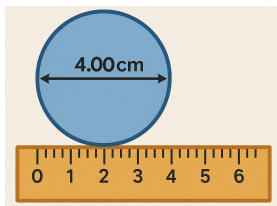
- $x = .23 \times 10^1 \text{ cm}$
- two significant digits
- 2 is the most significant digit
- 3 is the least significant digit

- $x = .40 \times 10^1 \text{ cm}$
- still two significant digits

- $x = .400 \times 10^1 \text{ cm}$
- The last zero is meaningless. From now on, we will never write it



Why it Matters



- You measure the diameter to be 4.00 cm with 3 significant digits.
- Then the area of the circle would be

$$\begin{aligned}A &= \pi(D/2)^2 \\&= \pi(4.00/2)^2 \\&\approx 12.566370614359172\end{aligned}$$

Key Idea

- ▶ We **cannot** report $A = 12.566370614359172$
- ▶ We must report 3 significant digits $A \approx 0.126 \times 10^2 \text{ cm}^2$

Significant Digits

7300

0.065

6900⁰

3020.40

8002

- There are several rules to counting significant digits
- Digits in light blue are significant figures; those in black are not.



Precision/Significance

Definition: Precision

The **degree of precision** or **significance** of a floating point number is the number of significant digits it contains.

In normalized form, it is easy to count precision

- $0.\underbrace{12345}_{5 \text{ digits}}0000$
- $0.\underbrace{3700004}_7 \times 10^4$
- $(0.\underbrace{10101010101}_{11 \text{ digits or 11 bits}})_2 \times 2^9$



Significant Digits in Floating Point Numbers

- Depends on the number of bits in the **mantissa**.
- Rule of thumb:

$$\text{significant digits} = -\log_{10}(\epsilon_m)$$

Format	Mantissa Bits	Approx. Digits	Machine Epsilon
Single (FP32)	23	~ 7	$2^{-23} \approx 1.19 \times 10^{-7}$
Double (FP64)	52	$\sim 15-16$	$2^{-52} \approx 2.22 \times 10^{-16}$
Quad (FP128)	112	~ 34	$2^{-112} \approx 1.93 \times 10^{-34}$

Key Idea

All variables created in FP64 begin with 15-16 significant digits.



Loss of Significance



Error with Addition

Loss of Precision via Rounding

When adding two floating-point numbers with very different orders of magnitude, digits from the **smaller** number may be lost.

- Suppose a and b each have 16 digits of precision:

$$a = 0.xxxx\ xxxx\ xxxx\ xxxx \times 10^0,$$

$$b = 0.yyyy\ yyyy\ yyyy\ yyyy \times 10^{-8}.$$

- Before adding, we must match the exponents:

$$b = 0.0000\ 0000\ yyyy\ yyyy\ \underbrace{yyyy\ yyyy}_{\text{digits lost}} \times 10^0.$$

- Therefore the computer effectively adds

$$\begin{array}{r} 0.xxxx\ xxxx\ xxxx\ xxxx \\ + 0.0000\ 0000\ yyyy\ yyyy \\ \hline 0.xxxx\ xxxx\ zzzz\ zzzz \end{array} \times 10^0.$$



Error with Subtraction

Loss of precision via cancellation

Subtracting $fl(a - b)$ can lead to large loss of significance if $a \approx b$.

- Example: let a and b be numbers with 16 significant digits

$$a = 0.xxxx\ xxxx\ xxxx\ xxx1$$

$$b = 0.xxxx\ xxxx\ xxxx\ xxx0$$

- Then $a - b$ yields

$$\begin{array}{r} 0.xxxx\ xxxx\ xxxx\ xxx1 \\ - 0.xxxx\ xxxx\ xxxx\ xxx0 \\ \hline = 0.\underbrace{0000\ 0000\ 0000\ 000}_\text{not significant}1 \end{array}$$

Definition: Catastrophic Cancellation

Catastrophic cancellation refers to a phenomenon that occurs when subtracting two nearly equal numbers results in a loss of significant digits

Loss of Precision Theorem

Theorem

Let x and y be (normalized) floating point machine numbers with $x > y > 0$.

If $2^{-p} \leq 1 - \frac{y}{x} \leq 2^{-q}$ for positive integers p and q , the significant binary digits lost in calculating $x - y$ is between q and p . Note: $q < p$

How to use:

- Calculate $f = 1 - y/x$.
- $\log_2(f)$ gives number of lost binary digits
- $\log_{10}(f)$ gives number of lost significant digits
- Note: you can't take the log of a negative number so $1-x/y$ won't work
- Tip: memorize $f = 1 - \frac{\text{small}}{\text{big}}$



Loss of Precision Examples

Example

How many bits are lost in the subtraction between $x = 37.593621$ and $y = 37.584216$?

$$2^{-12} < 1 - \frac{y}{x} = 0.0002501754 < 2^{-11}$$

So we lose 11 or 12 bits in the computation of $x - y$. yikes!

Example

How many significant digits are lost in the subtraction between $x = 0.37214$ and $y = 0.37202$?

$$10^{-4} < 1 - \frac{y}{x} = 0.00032 < 10^{-3}$$

So we lose 3 or 4 digits in the computation of $x - y$.
Here, $x - y = 0.00012$ which has only 1 or 2 significant digits.



Quadratic Formula

Example

Compute a root of the following equation:

$$x^2 + 2px - q = 0.$$

Let $p = 10,000$ and $q = 1$.

- Quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- One of the roots can be computed with

$$y_1 = -p + \sqrt{p^2 + q}$$

```
1 >> p = 10000; q = 1;  
2 >> y1 = -p+sqrt(p^2 + q)  
3 y1 = 5.0000000055588316e-05
```

Issue with Cancellation

- To verify the accuracy of the solution y_1 , we may substitute the value back into the equation

$$x^2 + 2px - q = 0.$$

- We expect the value of ϵ_1 to close to machine precision

$$\epsilon_1 = y_1^2 + 2py_1 - q$$

- However, it is 8 orders of magnitude away from ϵ_m

```
1 >> p = 10000; q = 1;  
2 >> y = -p+sqrt(p^2 + q)  
3 y1 = 5.0000000055588316e-05  
4 >> y1^2 + 2*p*y1 - q  
5 ans = 1.361766321927860e-08
```



Conjugation

Definition: Conjugate

For the binomial $a + b$, its **conjugate** is $a - b$.

- When you multiply two conjugates, you always $(a + b)(a - b) = a^2 - b^2$
- When calculating $a - b$, we may employ the **conjugation method**:

$$a - b = (a - b) \frac{a + b}{a + b} = \frac{a^2 - b^2}{a + b}$$

- This is particularly useful for the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



The Fix

- The two methods produce slightly different roots

```
1 >> p = 10000; q = 1;  
2 >> y1 = -p+sqrt(p^2 + q) % original method  
3 y1 = 5.0000000055588316e-05  
4  
5 >> y2 = q/(p+sqrt(p^2+q)) % new formula with conjugation  
6 y2 = 4.999999987500000e-05
```

- The second formula has a much lower error

```
1 >> y1^2 + 2*p*y1 - q % checking residue for method 1  
2 ans = 1.361766321927860e-08  
3  
4 >> y2^2 + 2*p*y2 - q % checking residue for method 2  
5 ans = -1.110223024625157e-16
```

Key Idea

First formula has an error on the order of 10^{-8} while the fixed formula has an error around 10^{-16} .