# Numerical Methods

- Using R
- Download R & R Studio

---

40% Final, 60% Assignments, quizzes

---

Will consider: Solving "on computer"

- $f(x) = 0$ $\quad f: \mathbb{R} \to \mathbb{R}$ some fn

- $\underset{n\times n}{A} \overset{n\times 1}{x} = b_{n\times 1}$

- $\int_a^b f(x)\,dx$ $\quad f: \mathbb{R} \to \mathbb{R}$ bounded continuous

- $y: [0,T] \to \mathbb{R}$ $\quad f: \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$

$$\frac{dy}{dt} = f(t,y)$$

- often not able to find exact solns
- Need approximations for practical purposes
- How good is approximation ?

What is good enough ?

Efficiency : How much work to reach soln.

Accuracy : How close to true value

Precision : Level of detail in our solution.

$$\pi = 3.14159\,26535\ldots$$

$$\frac{22}{7} = 3.14285714\ldots$$

3.1416 is more accurate approximation to $\pi$

Computer softwares work with floating point numbers

---

Floating point numbers in R

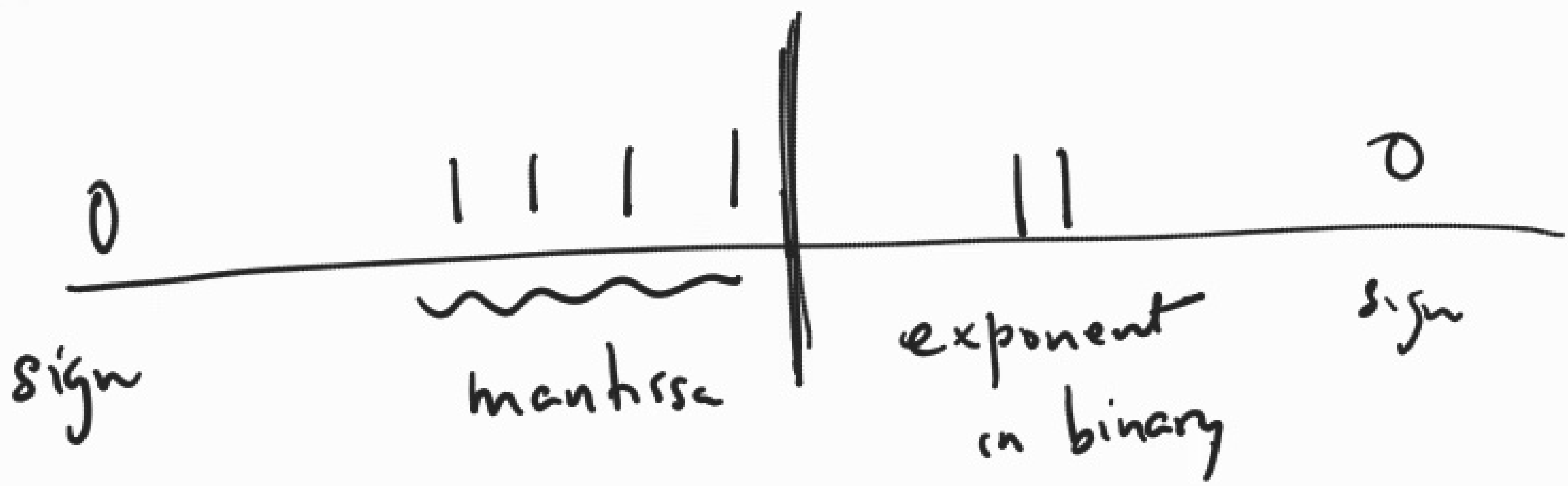
64 bits

| 1 bit sign | 52 bits mantissa | 1bit sign | 10 bits exponent |
|---|---|---|---|

$$4 = \underbrace{1}_{\text{mantissa}} \times 2^{\overbrace{2}^{\text{exponent}}}$$

$$15 = 1111 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1 \times 2^0$$

sign 0 is +1    1 is -1




Binary $1.111 \times 2^3$

Largest number that R can work with = ?

Max Exponent $= \underbrace{111\ldots\ldots1}_{10} = 2^{10}-1$

$= 1023$

Binary    Decimal

Max Mantissa $= \underbrace{1}_{1}.\underbrace{11\ldots\ldots1}_{51} = 1 + 1\cdot\frac{1}{2} + 1\cdot\frac{1}{4} + \cdots + 1\cdot\frac{1}{2^{51}}$

$$\left(1+\frac{1}{2}+\cdots+\frac{1}{2^{51}}\right)\times 2^{1023} \leftarrow$$ Max #
R can handle

$> 2 \times 2^{1023}$ NA

$> 1.5 \times 2^{1023}$ ✓

---

Smallest positive number R can handle : $\frac{1}{2^{51}} \times 2^{-1023}$

---

$> 20.55 - 19.2 - 1.35$

$1.332268 e^{-15}$

$> 20.55 - 1.35 - 19.2$

$0$

## Errors in computing

- Finite storage space
- Cannot store irrational numbers like $\pi$ exactly (truncation error)

Effects

- Floating point number line
- largest & smallest number

- Need to be more careful.

## Computer Storage

Bit (Binary digit) 0 or 1

Smallest unit of storage

Think of this as switching on/off.

**Byte** : Composed of 8 bits

Can store up to $2^8$ characters in a byte.

ASCII in Wikipedia : (Check)

Can store characters like 0, 1, 2, 3, . . 9

A, B, C . . . . Z , a, b, c . . . . z,

@, ?, ;, . . . . . in a byte.

---

Computers can work with 32 bits or 64 bits (Machine) in one go

---

# Storing numbers

**Integers** stored in R using 32 bits

_    _ _ _ . . . . . _

one bit for sign

$\underbrace{\qquad\qquad\qquad\qquad}$ 31 bits number in binary

Decimal: 32 29 201 . . .

Binary : 101, 11111, 10, 110 . . . . .

$$32 = 3 \times 10^1 + 2 \times 10^0$$

$$201 = 2 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$$

---

Binary $101 = \underline{1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0} = 5$ in decimal

most significant

least significant digit

Binary $1110 = 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0$

$= 14$ in decimal.

Max number that can be stored as an integer is

$$\underbrace{1\;1\;1\;\ldots\ldots\;1}_{31\text{ bit}}$$

$$1\times 2^{30} + \cdots + 1\times 2^1 + 1\times 2^0$$

$$= 2^{31} - 1$$

In R eg. to store 231 as an integer

```
> 231L
```

# Floating point numbers. Default in R

**Decimal** $21.8125$

$$= 2\times 10^1 + 1\times 10^0 + 8\times 10^{-1} + 1\times 10^{-2} + 2\times 10^{-3} + 5\times 10^{-4}$$

**Binary** of $21.8125$

$$21 = 1\times 2^4 + 1\times 2^2 + 1\times 2^0$$
$$= 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$

21 in Binary $10101$

$$0.8125 = 0.5 + 0.3125$$
$$= 0.5 + 0.25 + 0.0625$$
$$= 1\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}$$

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

Binary $0.1101$

Binary of 21.8125 is

10101.1101

---

Eg Decimal 0.2

$$0.2 = 0.125 + 0.075$$

$$= 0.125 + 0.0625 + 0.0125$$

$$= 0.125 + 0.0625 + 0.0078125 + \cdots$$

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

$2^{-5} = 0.03125$

$2^{-6} = 0.015625$

$2^{-7} = 0.0078125$

$\vdots$

? Infinite Expansion

Remark. Infinite expansion in binary cannot be stored!

A floating point number is stored in 64 bits

Sign - 1 bit

Mantissa - 52 bits

Exponent - 11 bits (1 bit for sign)

64 bits

---

<u>Sign Bit</u> 0 or 1
(positive) (negative)

<u>Exponent</u> Range, integers in $[-1022, 1023]$

Stored in binary with a bias of 1023

So $-1022$ is stored as $-1022+1023=1$

1023 " $1023+1023 = 2046$

$-1022 \to 1$

In 11 Bits $\qquad \underbrace{0\,0 \ldots\ldots 0}_{10\ 0's}\,1$

$1023 \to 2046 = 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2$

$= \underbrace{1\,1\,1 \ldots\ldots 1}_{10\ 1's}\,0$

Remark. Note exponent $-1022, \ldots, 0$ when stored with bias in binary have 0 as 1st digit

$1, 2, \ldots, 1023$ when stored with bias in binary have 1 as 1st digit

# Mantissa 52 bits

$231.56 \rightarrow 2.3156 \times 10^{2}$

$0.0156 \rightarrow 1.56 \times 10^{-2}$

Similarly write binary number with just one 1 to the left of binary point.
(get extra bit for free!)

$$0.\underbrace{0110 11 \ldots 1}_{54}$$

$$= 1.\underbrace{1011 \ldots 1}_{52} \text{ “}\times\text{” } 2^{2}$$

always 1

**Eg** 21.8125 $\longrightarrow$ 10101.1101 in binary

**Sign Bit** $\boxed{0}$

(a) $\leftarrow$ $10101.1101 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

(b) $\leftarrow$ $1.01011101 = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + \cdots$

$$(a) = 2^4 \times (b)$$

**Mantissa**: $\boxed{01011101\ \underbrace{0 - - - - - 0}_{44 \text{ bits}}}$

**Exponent**: $4 \longrightarrow 1027$

$= 1024 + 2 + 1$

$\boxed{\underbrace{10\ldots\ldots011}_{11 \text{ bits}}}$

"Smallest +ve number"

$1.00\cdots\cdots 0$

$$\underset{\substack{\uparrow \\ S}}{\underline{0}} \quad \underset{\substack{\uparrow \\ \text{exponent}}}{\underline{00\cdots\cdots 01}} \quad \underset{\substack{\text{52 bit} \\ \text{mantissa}}}{\underline{00\cdots 0}}$$

$$\left(1+\sum_{1}^{52} 0 \times 2^{-j}\right) \times 2^{1-1023}$$

```
> .Machine$double.xmin
```

<u>Remark</u>: R can work with smaller numbers ("denormal numbers") at the cost of precision. (some digits from mantissa go to exponent)

"Largest +ve number"

$$\underset{s}{\underline{0}} \quad \underset{\text{exponent}}{\underline{111\ldots10}} \quad \underset{52 \text{ bit}}{\underline{1111\ldots1}}$$

$$\left(1+\sum_{1}^{52} 1\times 2^{-j}\right)\times 2^{2046-1023}$$

<u>Remark</u>: R <u>cannot</u> work with larger numbers. Losing precision not an option

---

## Special numbers

| sign | exponent | mantissa | |
|---|---|---|---|
| 0 | 0 0 0 0 ... 0 | 0 0 ... 0 | +0 |
| 1 | 0 ... ... 0 | 0 ... 0 | −0 |
| 0 | 1 ... ... 1 | 0 ... 0 | +Inf |
| 1 | 1 ... ... 1 | 0 ... ... 0 | −Inf |
| 0 | 1 ... ... 1 | 1 0 ... 0 | NAN |
| 1 | 1 ... 1 | $1\,\underbrace{0 \ldots 0}_{51\text{ bits}}$ | NAN |

"NAN" not a number

Eg

$1 = 1 \times 2^0 \Rightarrow$ Binary $1.00\ldots$

Sign

0

Exponent

$011\ldots1$

Mantissa

$00\ldots0$

Exponent = 0 $\rightarrow$ 1023