



UNIVERSITÉ  
CÔTE D'AZUR

# Nombres réels approchés

Algo & Prog avec R

---

A. Malapert, B. Martin, M. Pelleau, et J.-P. Roy

11 septembre 2021

Université Côte d'Azur, CNRS, I3S, France  
`firstname.lastname@univ-cotedazur.fr`

# Nombres réels approchés

Ou nombres réels inexacts. On parle de **nombres flottants** (float).

## Calcul entier et réel en précision finie

Ils n'ont qu'un nombre limité de chiffres avant et après la "virgule" (le point décimal).

**Donc aucun nombre irrationnel !**

### Approximation de $\pi$

```
> pi
[1] 3.141593
> sprintf('%.17f',pi)
[1] "3.14159265358979312"
> typeof(pi)
[1] "double"
```

### Approximation de $\sqrt{2}$

```
> sqrt(2)
[1] 1.414214
> sprintf('%.17f',sqrt(2) ** 2)
[1] "2.00000000000000044"
> sqrt(2) ** 2 == 2
[1] FALSE
```

Mais,

$\pi = 3.14159265358979323\dots$

# Nombres rationnels

Les nombres rationnels peuvent être représentés sous la forme d'une fraction, par exemple  $\frac{1}{10}$ .

- ▶ Le nombre  $\frac{1}{10} = (0.1)_{10}$ , par exemple, est simple dans le système décimal.
- ▶ Mais, il possède une infinité de chiffres après la virgule dans le système binaire !

0.00011001100110011001100110011001100110011001100110011...

Lorsque R affiche une valeur approchée, ce n'est qu'une approximation de la véritable valeur interne de la machine :

```
> 0.1 # quelle est la valeur de 0.1 ?  
[1] 0.1 # ceci est une illusion !
```

La fonction `print` ou `printf` permet de voir (en décimal) la véritable représentation en machine de 0.1 qui n'est pas 0.1 mais :

```
> print(0.1, digits=17)  
[1] 0.100000000000000001
```

# Représentation des nombres réels

- ▶ Les ressources d'un ordinateur étant limitées, on représente seulement un **sous-ensemble des réels de cardinal fini**.
- ▶ Ces éléments sont appelés **nombres à virgule flottante**.
- ▶ Leurs propriétés sont différentes de celles des réels.

## Problèmes et limitations

- ▶ Les nombres et les calculs sont nécessairement arrondis.
- ▶ Il y a des erreurs d'arrondi et de précision.
- ▶ On ne peut plus faire les opérations de façon transparente.
- ▶ L'ordre des opérations peut changer les résultats.

## Le zéro n'est plus unique !

```
> 10^20 + 1 == 10^20  
[1] TRUE  
> 10^20 + 2 == 10^20  
[1] TRUE
```

En math, il existe un unique nombre  $y$  tel que  $x + y = x$ , le zéro !

# Égalité entre nombres flottants

Le calcul sur des nombres approchés étant par définition **INEXACT**, on évitera sous peine de surprises désagréables de questionner l'**ÉGALITÉ** en présence de nombres approchés !

```
> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.7
[1] TRUE
> 0.1 * 7 == 0.7
[1] FALSE
> 0.1 + 0.1 + 0.1 == 0.3
[1] FALSE
> 0.1 * 3 == 0.3
[1] FALSE
```

Le domaine du calcul approché est TRÈS difficile, et prévoir à l'avance le nombre exact de décimales correctes lors d'un résultat de calcul reste réservé aux spécialistes d'analyse Numérique (brrr) ...

**Alors que faire ? Remplacer l'égalité par une précision h**

```
a == b # BAD !
```

```
abs(a - b) < h # GOOD !
```

# Autres problèmes avec les nombres flottants

## Une boucle infinie ?

```
x <- 1
while( x > 0 ) {
  print(x)
  x <- x / 2
}
```

Est-ce que cette boucle s'arrête ? En math ? En info ?

## Annulation catastrophique $x^2 - y^2$

```
> y <- 2**50
> x <- y + 1
> z1 <- x**2 - y**2 # appliquer directement la formule
> z2 <- (x - y)*(x + y) # appliquer une identité remarquable
> z2 - z1 # Est-ce que les résultats sont identiques ?
[1] 1
```

## Exemple : approximation de $\sqrt{r}$

Par la méthode des tangentes de Newton (1669).

Soit à calculer la racine carrée approchée d'un nombre réel  $r > 0$ , par exemple  $\sqrt{2}$ , sans utiliser `sqrt` !

### Newton

Si  $a$  est une approximation de  $\sqrt{r}$  alors :

$$b = \frac{1}{2}\left(a + \frac{r}{a}\right)$$

est une approximation encore meilleure ! Pourquoi ? Cf TD.

Nous allons développer cet algorithme en répondant à trois questions :

**ITÉRATION** Comment améliorer l'approximation courante ?

**TERMINAISON** Mon approximation courante  $a$  est-elle assez bonne ?

**INITIALISATION** Comment initialiser la première approximation ?

# Algorithme d'approximation de $\sqrt{r}$

## ITÉRATION

Pour **améliorer** l'approximation, il suffit d'appliquer la formule de Newton, qui fait approcher  $a$  de  $\sqrt{r}$  :

```
a = 0.5 * (a + r / a)
```

## TERMINAISON

Mon approximation courante  $a$  est-elle **assez bonne** ? Elle est assez bonne lorsque  $a$  est très proche de  $\sqrt{r}$ . Notons  $h$  la variable dénotant la précision, par exemple  $h = 2^{-20}$ .

```
abs(a*a - r) < h
```

## INITIALISATION

Comment **initialiser** l'approximation ? En fait, les maths sous-jacentes à la technique de Newton montrent que n'importe quel réel  $a > 0$  convient :

```
a = 1
```



# Programme d'approximation de $\sqrt{r}$

```
Racine <- function(r, h = 2**(-10)) {  
  a <- 1  
  while( abs(a*a -r) >= h) {  
    print(a)  
    a <- 0.5 * (a + r/a)  
  }  
  return(a)  
}
```

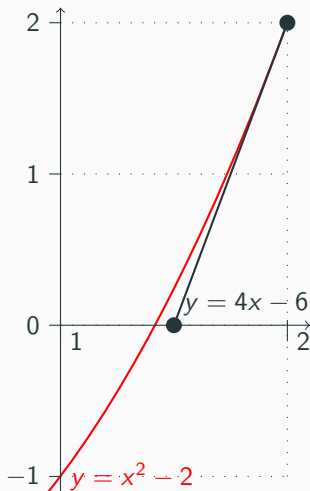
```
> approx <- Racine(r = 2, h = 10**(-10))  
[1] 1  
[1] 1.5  
[1] 1.416667  
[1] 1.414216  
> print(approx, digit = 15)  
[1] 1.41421356237469  
> print(sqrt(2), digit = 15)  
[1] 1.4142135623731
```

## Observation

La méthode de Newton converge rapidement vers le résultat.

# Mais d'où vient la formule de Newton $b = \frac{1}{2}(a + \frac{r}{a})$ ?

D'un simple calcul de tangentes (cf TD ou [wikipedia](#)) ...



Questions?

Retrouvez ce cours sur le site web

[www.i3s.unice.fr/~malapert/R](http://www.i3s.unice.fr/~malapert/R)

# Répartition de flottants 6 bits sur la droite réelle

## Flottant ~ Notation scientifique

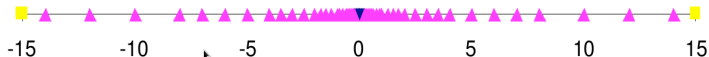
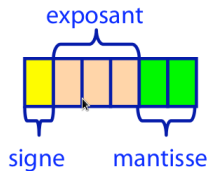
$-34.9803 = -0.349803 \times 10^2$

signe      mantisse      exposant

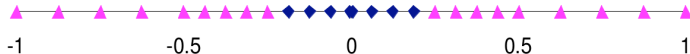


## Flottants sur 6 bits

Soit  $2^6$  valeurs distinctes.



◆ Denormalized    ▲ Normalized    ■ Infinity



◆ Denormalized    ▲ Normalized    ■ Infinity