



UNIVERSITÉ  
CÔTE D'AZUR

# Nombres réels approchés

Algo & Prog avec R

---

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

14 octobre 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] 0.10000000000000001
```

# 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)  
}
```

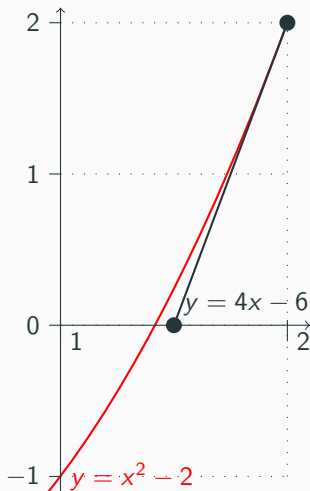
```
> approx <- Racine(r = 2, h = 10**(-10))  
[1] 1  
[1] 1.5  
[1] 1.416667  
[1] 1.414216  
> print(approx, digit = 15)  
[1] 1.41421356237469  
> print(sqrt(2), digit = 15)  
[1] 1.4142135623731
```

## 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

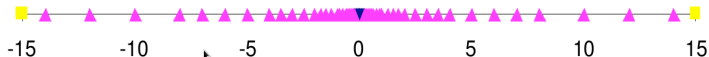
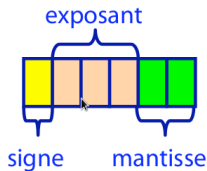
$-34.9803 = -0.349803 \times 10^2$

signe      mantisse      exposant

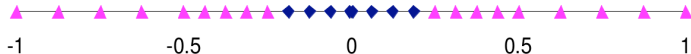


## Flottants sur 6 bits

Soit  $2^6$  valeurs distinctes.



◆ Denormalized    ▲ Normalized    ■ Infinity



◆ Denormalized    ▲ Normalized    ■ Infinity

## Tableau des flottants 6 bits

0 000 00 = +0.0

0 000 01 = +0.01x2<sup>-2</sup> = +0.0001

0 000 10 = +0.10x2<sup>-2</sup> = +0.001

0 000 11 = +0.11x2<sup>-2</sup> = +0.0011

0 001 00 = +1.00x2<sup>-2</sup> = +0.01

0 001 01 = +1.01x2<sup>-2</sup> = +0.0101

0 001 10 = +1.10x2<sup>-2</sup> = +0.011

0 001 11 = +1.11x2<sup>-2</sup> = +0.0111

0 010 00 = +1.00x2<sup>-1</sup> = +0.1

0 010 01 = +1.01x2<sup>-1</sup> = +0.101

0 010 10 = +1.10x2<sup>-1</sup> = +0.11

0 010 11 = +1.11x2<sup>-1</sup> = +0.111

0 011 00 = +1.00x2<sup>0</sup> = +1.0

0 011 01 = +1.01x2<sup>0</sup> = +1.01

0 011 10 = +1.10x2<sup>0</sup> = +1.1

0 011 11 = +1.11x2<sup>0</sup> = +1.11

0 100 00 = +1.00x2<sup>1</sup> = +10.0

0 100 01 = +1.01x2<sup>1</sup> = +10.1

0 100 10 = +1.10x2<sup>1</sup> = +11.0

0 100 11 = +1.11x2<sup>1</sup> = +11.1

0 101 00 = +1.00x2<sup>2</sup> = +100.0

0 101 01 = +1.01x2<sup>2</sup> = +101.0

0 101 10 = +1.10x2<sup>2</sup> = +110.0

0 101 11 = +1.11x2<sup>2</sup> = +111.0

0 110 00 = +1.00x2<sup>3</sup> = +1000.0

0 110 01 = +1.01x2<sup>3</sup> = +1010.0

0 110 10 = +1.10x2<sup>3</sup> = +1100.0

0 110 11 = +1.11x2<sup>3</sup> = +1110.0

0 111 00 = +∞

0 111 01 = NaN

0 111 10 = NaN

0 111 11 = NaN

## Quelques erreurs catastrophiques

- ▶ En 1982 à la bourse de Vancouver, les calculs sur un indice nouveau, de valeur initiale 1000.0, sont tronqués (plutôt qu'arrondis). Après 22 mois la valeur calculée de l'indice est de 520 au lieu de 1098.892...
- ▶ Le 25 février 1991, à Dhahan en Arabie Saoudite, un missile anti-missile Patriot américain rate l'interception d'un missile Scud irakien à cause d'une erreur d'arrondi dans sa trajectoire : 28 morts, plus d'une centaine de blessés.
- ▶ Lors des élections dans le Schleswig-Holstein le 5 avril 1992, l'affichage du pourcentage de voix des Verts sans décimales puis avec deux décimales change le résultat et donne la majorité au SPD au parlement allemand.
- ▶ Le 4 juin 1996, lors de son premier vol, la fusée européenne Ariane 5 explose 30 secondes après son décollage à cause d'une erreur d'*overflow* dans l'ordinateur de bord causant une perte estimée à 500 M\$ (et 7 milliards de dollars d'investissement).