# Approximation du nombre $\pi$

## Algo & Prog avec R

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

15 octobre 2020

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

# Le nombre $\pi$

- $\pi$ est défini comme le rapport constant entre la circonférence d'un cercle et son diamètre dans le plan euclidien.
- De nos jours, les mathématiciens définissent $\pi$ par l'analyse réelle à l'aide des fonctions trigonométriques elles-mêmes introduites sans référence à la géométrie.
- Le nombre $\pi$ est irrationnel, ce qui signifie qu'on ne peut pas l'écrire comme une fraction.
- Le nombre $\pi$ est transcendant ce qui signifie qu'il n'existe pas de polynôme à coefficients rationnels dont $\pi$ soit une racine.

# Calcul de $\pi$ par la formule de Leibniz

On utilisera la formule de Leibniz issue du développement en série de Taylor en 0 de *arctan*$(x)$ évalué au point 1 :

$$\sum_{k=0}^{+\infty} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{9} + \frac{1}{11} + \ldots = \frac{\pi}{4}$$

Elle a été découverte en Occident au XVIIe, mais apparaît déjà chez Madhava, mathématicien indien de la province du Kerala, vers 1400.

c.f. Wikipedia

Nous allons développer un algorithme d'approximation de $\pi$.

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

**TERMINAISON** Mon approximation courante est-elle assez bonne ?
Est-ce que le calcul prend trop de temps ?

**INITIALISATION** Comment initialiser la première approximation ?

# Algorithme d'approximation de $\pi$

## ITÉRATION

Pour améliorer l'approximation, étant en possession de la somme `acc` des $i$ premiers termes, on voudra obtenir la somme des $i+1$ premiers. Il suffira donc d'incrémenter `i`, puis d'ajouter $\frac{(-1)^i}{2i+1}$ à `acc`.

```
i <- i + 1
term <- (-1)**i / (2*i + 1)
acc <- acc + term
```

## TERMINAISON

Mon approximation courante a est-elle assez bonne ? Elle est assez bonne lorsque je n'arrive plus à l'améliorer. Notons `h` la précision.

Est-ce que le calcul prend trop de temps ? Notons `n` le nombre maximum de termes à calculer.

```
abs(term) < h || i > n
```

## INITIALISATION

```
i <- 0
acc <- 1
```

# Programme d'approximation de $\pi$

```
LeibnizPi <- function(n = 10**4, h = 2**(-20)) {
  i <- 0
  term <- 1
  acc <- 1
  while( (i <= n) && 4*abs(term) > h) {
    i <- i + 1
    term <- (-1)**i / (2*i + 1)
    acc <- acc + term
  }
  return(4*acc)
}
```

```
> LeibnizPi(n = 100, h = 0)
[1] 3.131789
> LeibnizPi(n = 100000, h = 0)
[1] 3.141583
> pi
[1] 3.141593
```

# Analyse de la convergence vers $\pi$

```
ErreurRelativePi <- function(v) return(1 - v/pi)
ErreurRelativeLeibnizPi <- function(n) {
  approxPi <- LeibnizPi(n = n, h = 0)
  return(ErreurRelativePi(approxPi))
}
```

**Formule de Leibniz**

| n | Erreur |
|---|---|
| 100 | 0.003121 |
| 1000 | 0.000318 |
| 10000 | 0.000032 |
| 100000 | 0.000003 |

**Approximation rationnelle**

| Fraction | Erreur |
|---|---|
| $\frac{22}{7}$ | -0.000402499435 |
| $\frac{355}{113}$ | -0.000000084914 |
| $\frac{103993}{33102}$ | 0.000000000184 |
| $\frac{104348}{33215}$ | -0.000000000106 |

**Observation**

La formule de Leibniz converge lentement.

# Analyse des performances du programme

Calculons le temps nécessaire pour atteindre une précision donnée sans limiter le nombre d'itérations.

```
> system.time(LeibnizPi(n = Inf, h = 10**(-4)))
utilisateur     système      écoulé
      0.006       0.000       0.006
```

- Le temps d'exécution et le nombre d'itérations augmentent linéairement avec la précision.
- La recherche d'une estimation très précise de $\pi$ demande un temps de calcul important.
- En extrapolant ces résultats, il faudrait $5 \times 10^8$ *secondes* ($\geq$ 15 ans) pour obtenir une estimation de $\pi$ à la précision machine (approximativement 15 décimales).
- Certaines formules convergent beaucoup plus rapidement.

| Précision | Temps (s) |
|---|---|
| $10^{-4}$ | 0.006 |
| $10^{-5}$ | 0.147 |
| $10^{-6}$ | 0.599 |
| $10^{-7}$ | 5.347 |
| $10^{-8}$ | 52.860 |

## Optimisation du programme

**Exploitons la récurrence pour accélérer les calculs.**

Les multiplications, divisions, et puissances sont plus coûteuse en temps de calcul que les additions et soustractions.

```
LeibnizPi2 <- function(n = 10**4, h = 2**(-20)) {
  i <- 0
  term <- 1
  acc <- 1
  h <- h / 4 ## éviter la multiplication du test
  signe <- 1 ## mémoriser le signe du terme
  denom <- 1 ## mémoriser le dénominateur du terme
  while( (i <=n) && abs(term) > h) {
    i <- i + 1
    signe <- -1 * signe # éviter une puissance
    denom <- denom + 2 # éviter une multiplication
    term <- signe / denom
    acc <- acc + term
  }
  return(4*acc)
}
```

# Comparaison de programmes

| Précision | Temps (s) |
|---|---|
| `LeibnizPi` | 5.347 |
| `LeibnizPi2` | 4.157 |
| En langage C | 0.007 |

- Les optimisations du programme offrent un gain supérieur à 20%.
- R est donc un langage interprété de haut niveau ce qui se paie au niveau des performances.
- Le langage C, entre autres, est beaucoup plus rapide.
- Le langage C est un langage impératif, généraliste et de bas niveau où chaque instruction du langage est compilée.

**Questions?**

**Retrouvez ce cours sur le site web**

www.i3s.unice.fr/~malapert/R

# La formule d'Euler-Wallis converge vers $\frac{\pi}{2}$

Ce produit peut s'écrire sous la forme :

$$P(n) = \frac{2}{1} \times \frac{2}{3} \times \frac{4}{3} \times \frac{4}{5} \times \frac{6}{5} \times \frac{6}{7} \times \frac{8}{7} \times \frac{8}{9} \cdots \frac{2n}{2n-1} \times \frac{2n}{2n+1} = \prod_{k=1}^{n} \frac{4k^2}{4k^2-1}$$

## Récurrence

```
P <- function(n) {
  if(n <=0) return(0)
  else {
    t <- 4*(n**2)
    return((t/(t-1)) * P(n-1))}
}
```

## Vectorisation (plus tard)

```
P <- function(n) {
  if(n <=0) return(0)
  terms <- 4*(1:n)**2
  return(prod(terms/(terms-1)))
}
```

## Itération

```
P <- function(n) {
  if(n <=0) return(0)
  acc <- 1
  i <- 0
  while(i < n) {
    i <- i + 1
    t <- 4*(i**2)
    acc <- acc * t/(t-1)
  }
  return(acc)
}
```

# Elle converge lentement ...

```
> 2*P(100)
[1] 3.133787
> 2*P(10000)
[1] 3.141514
> 2*P(100000)
[1] 3.141585
> 2*P(100000)
[1] 3.141585
> 2*P(1000000)
[1] 3.141592
> pi
[1] 3.141593
```