TP: Calcul de racines carrées

M1 MEEF maths

17 octobre 2022

Le but de cette activité est d'étudier et comparer différents algorithmes pour calculer des racines carrées.

Dans un premier temps, les calculs se feront en utilisant les nombres à virgule flottante fournis par Python, dans un souci de simplicité, et on va chercher à obtenir des valeurs « à ϵ près » puisqu'en général on ne peut pas représenter de valeur exacte. En poussant le travail plus loin, on pourra se passer de cet ϵ puisque les nombres utilisés ont en fait une précision connue. La fonction `math.sqrt` fournit la meilleure approximation de la racine carrée en virgule flottante et elle pourra servir à vérifier vos résultats.

Pour mesurer les temps d'exécution, on utilisera la bibliothèque `timeit` de Python qui permet de chronométrer l'exécution d'un morceau de code.

```
from math import sqrt
from timeit import timeit
```

Pour simplifier le chronométrage, on definit la fonction suivante, qui reçoit en arguments le code à exécuter et le nombre de fois qu'il faut l'exécuter (pour obtenir une meilleure précision dans la mesure du temps de calcul). La valeur renvoyée est le temps en secondes pour exécuter le code une fois.

```
def chrono(code, number):
    return timeit(code, globals=globals(), number=number) / number
```

Méthode par balayage

C'est la méthode la plus simple. On utilise le fait que la fonction $x \mapsto x^2$ est croissante et pour calculer $\sqrt{a}$ on cherche la plus grande valeur de x telle que $x^2 \leq a$, en partant de 0 et en avançant par pas de ϵ.

```
def balayage(a, epsilon):
    # ...
```

```
[balayage(a, 1e-7) for a in [2, 3, 4]]
```

```
chrono("balayage(2, 1e-7)", 1)
```

Méthode par dichotomie

Dans la méthode par dichotomie, on se base encore sur la croissance de la fonction *carré* mais on recherche le point où elle dépasse le x donné, mais on procède en divisant successivement par 2 la longueur de l'intervalle de recherche.

```
def dichotomie(a, epsilon):
    # ...
```

```
[dichotomie(x, 1e-7) for x in [2, 3, 4]]
```

```
chrono("dichotomie(2, 1e-7)", 100000)
```

Expliquer la durée obtenue en évaluant le nombre d'étapes de calcul.

Méthode de Héron

La méthode de Héron utilise un argument géométrique pour approcher une racine carrée de façon efficace. Elle consiste à itérer la fonction $x \mapsto \frac{1}{2}\left(x + \frac{a}{x}\right)$ jusqu'à atteindre la précision voulue.

```
def héron(a, epsilon):
    # ...
```

```
[héron(x, 1e-8) for x in [2, 3, 4]]
```

```
chrono("héron(2, 1e-8)", 100000)
```

Expliquer la durée obtenue en évaluant le nombre d'étapes de calcul.

Calcul dans les entiers

On cherche maintenant à calculer une racine carrée dans les entiers, avec la meilleure précision possible. C'est-à-dire que pour un entier a donné, on veut l'entier $\lfloor \sqrt{a} \rfloor$, la partie entière de la racine carrée de a.

Notez bien que prendre la partie entière de ce que donne la fonction `sqrt` ne fonctionne pas:

```
x = int(sqrt(2 * 10**100))
print(x)
print(x * x)
print((x + 1) * (x + 1))
```

En revanche, Python fournit une fonction `math.isqrt` qui fait exactement ce qu'on attend, mais bien sûr, vous n'avez pas le droit de l'utiliser dans cet exercice.

Définissez une fonction qui marche !

```
def racine_entiere(a):
    # ...
```

```
racine_entiere(2 * 10**100) == 14142135623730950488016887242096980785696718753769
```

Précision maximale

Finalement, implémentez un calcul de racine carrée qui atteint la précision des nombres à virgule flottante, sans avoir à préciser un ϵ. La fonction devra renvoyer la même chose que `sqrt`, mais bien sûr sans utiliser cette fonction !

```
def racine_flottante(a):
    # ...
```

```
racine_flottante(2) == sqrt(2)
```