Algorithmique et racine carrée en seconde, compléments

publié le 03/03/2020 - mis à jour le 20/09/2023

Parcours sur les algorithmes d'approximation de racines carrées

Descriptif :

Cet article présente une séquence d'algorithmique en seconde autour de la recherche de valeurs approchées de racines carrées par diverses méthodes.

Sommaire :

Cet article fait suite à la première partie du parcours sur les racines carrées.

Algorithme de dichotomie pour les plus rapides

Lors de la séance de TP sur le balayage, quelques groupes avaient réussi les implémentations demandées avant la fin de l'heure.
Il leur a alors été demandé de construire un algorithme de détermination d'une racine carrée en utilisant le principe de la dichotomie, qu'ils devaient découvrir par eux-mêmes en faisant des recherches sur le web.

Animation illustrant la recherche d'une racine carrée par dichotomie (Geogebra Tube)
Choisissez un nombre dont vous voulez la racine carrée, réglez l'amplitude de l'encadrement final et appuyez sur la touche "étape suivante" jusqu'à atteindre la précision souhaitée.

Les élèves engagés dans cette recherche ont globalement compris le principe de la dichotomie avec l'affectation des bornes qui diffère selon la position de la racine par rapport au milieu de l'intervalle et la longueur de l'intervalle qui est divisée par deux à chaque tour. En revanche, aucun groupe n'a pu établir un code fonctionnel de cet algorithme (voir exécution en ligne) :

```
1.  def racine_dicho(nombre,nb_decimales):
2.      """détermine par dichotomie un encadrement de la racine carrée d'un nombre d'amplitude 10**(-nb_decimales) en partant d'un encadrement à l'unité à et
    b"""
3.      x = 0 # on reprend l'algorithme de balayage pour trouver un encadrement à l'unité de la racine carrée
4.      while x**2 < nombre :
5.          x= x + 1
6.      debut = x - 1
7.      fin = x
8.      compteur = 0
9.      while abs(fin-debut) > 10**(-nb_decimales):
10.         milieu= (debut + fin) / 2
11.         if milieu**2 <= nombre :
12.             debut= milieu
13.         else :
14.             fin= milieu
15.         compteur= compteur + 1
16.     return round(debut, nb_decimales), round(fin, nb_decimales), compteur
```

Algorithme de Héron en devoir maison

Parallèlement à cette séquence qui s'est déroulée sur 3 séances de TP hebdomadaires, un devoir maison de recherche proposait une approche géométrique du calcul d'une racine carrée par la méthode de Héron.

Cette méthode déjà connue par les Babyloniens, est également attribuée au grec Héron d'Alexandrie (1 er siècle). Il l'expose dans le premier tome de son ouvrage *Metrica*, ouvrage qui a été découvert en 1896.

Chez les mathématiciens grecs, extraire la racine carrée de a revient à trouver un carré dont l'aire soit égale à a. En prenant un rectangle de côté arbitraire $L_0 = x$ et de même aire a, il est nécessaire que l'autre côté ait pour longueur $\ell_0 = \frac{a}{x}$.

Pour le rendre moins rectangle, il suffit de considérer un nouveau rectangle dont les dimensions vérifient :

- la longueur est la **moyenne arithmétique** des dimensions du rectangle précédent ;
- pour que l'aire reste égale à a, on **divise** a par cette nouvelle longueur pour trouver l'autre dimension du rectangle.

En réitérant infiniment le processus, le rectangle se transforme progressivement en un carré de même aire. Cette constatation est à la base de la méthode de Héron qui a été étudiée dans le devoir maison pour un nombre réel $a>1$.

Animation illustrant la recherche d'une racine carrée par l'algorithme de Héron (Geogebra Tube)
Choisissez un nombre dont vous voulez la racine carrée, réglez le nombre d'étapes et appuyez sur la touche "étape suivante" jusqu'à atteindre le nombre d'étapes choisi (au-delà de 5 étapes, les limites de GeoGebra sont atteintes)

L'implémentation attendue dans le devoir maison était la suivante (voir exécution en ligne) :

```
1.  def racine_heron(a, nb_decimales) :
2.      """cette fonction calcule un encadrement de la racine carrée de a avec le nombre de décimales spécifié"""
3.      longueur = a
4.      largeur = 1
5.      compteur = 0
6.      while abs(longueur - largeur) > 10**(-nb_decimales) :
7.          longueur= (longueur + largeur) / 2
8.          largeur= a / longueur
9.          compteur= compteur + 1
10.     return round(largeur,nb_decimales), round(longueur,nb_decimales),compteur
```

La méthode de Héron est un cas particulier de la méthode de Newton pour la détermination de solution d'une équation de la forme $f(x)=0$ avec $f(x)=x^2-a$. Sa vitesse de convergence est très rapide : la convergence de la suite sous-jacente est quadratique donc le nombre de décimales exactes double à chaque tour de boucle. Par exemple, pour le calcul de $\sqrt{2}$, on atteint 11 décimales exactes dès la 4ème étape :

```
1.  >>> racine_heron(2,11)
2.  (1.41421356237, 1.41421356237, 4)
```

Lors de la remise des devoirs maison, un temps de synthèse en classe a été ménagé pour comparer la rapidité des algorithmes. Les compteurs installés dans chacune des fonctions ont ainsi permis de comparer le nombre de tours de boucles nécessaires pour atteindre une précision donnée avec chacun des algorithmes.
Paradoxalement, l'algorithme le plus rapide s'est révélé être celui qui s'appuyait sur une méthode géométrique, à savoir l'algorithme basé sur la méthode de Héron.

```
1.  >>> racine_balayage(2,7)
2.  (1.4142135, 1.4142136, 14142136)
3.  >>> racine_balayage_opt(2,7)
4.  (1.4142135, 1.4142136, 29)
5.  >>> racine_dicho(2,7)
6.  (1.4142135, 1.4142136, 24)
7.  >>> racine_heron(2,7)
8.  (1.4142136, 1.4142136, 4)
```

Enoncé du devoir maison sur l'algorithme de Héron (PDF de 152.1 ko)
Ce devoir maison a été proposé aux élèves dans le but de leur faire découvrir une méthode géométrique de calcul de racine carrée.

● Un peu de nostalgie : l'algorithme de la potence

> *Cette partie n'a pas été abordée en classe, elle figure dans cet article à titre de complément culturel*

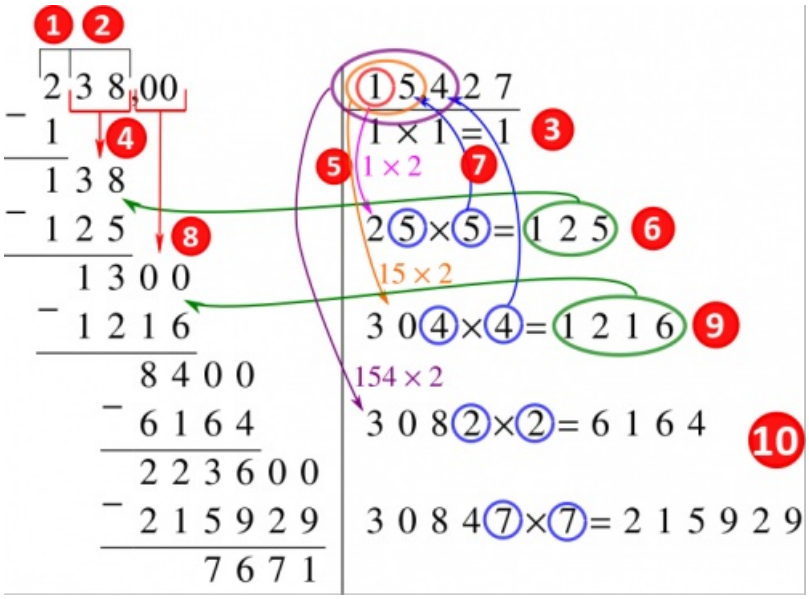
L'algorithme de la potence, qui tient son nom de la disposition des calculs organisée autour d'une potence comme une division, est un algorithme qui était utilisé jusqu'au milieu du vingtième siècle pour extraire des racines carrées à la main.

Le principe est le suivant : on sépare les chiffres du nombre par paires en commençant à partir de la virgule. On place le nombre dont on veut extraire la racine en haut, de la même façon que lorsqu'on effectue une division selon la méthode classique ; la racine carrée sera inscrite à la place réservée normalement au diviseur dans une division posée classique.
On commence par abaisser la première tranche t_0 et on cherche le plus grand nombre x tel que $x^2 \leqslant t_0$. Ce nombre x est le premier chiffre de la racine (à partir de la gauche) et on le note r. On soustrait ensuite x^2 à t_0 de sorte que l'on ait un premier reste.
À chaque étape :

- on abaisse la paire de chiffres suivante et on la place au côté d'un reste éventuel de l'étape précédente (initialement nul) ;
- soit r le résultat intermédiaire de la racine carrée obtenu précédemment (égal à zéro au début). On cherche le plus grand chiffre x tel que le nombre $y = (20r + x)x$ ne dépasse pas le reste courant ;
- on complète r en plaçant la décimale x à sa droite, pour former le nouveau résultat intermédiaire ;
- on soustrait y de la valeur courante pour former le nouveau reste ;
- si le reste est nul et qu'il n'y a plus de chiffre à abaisser alors l'algorithme se termine sinon on recommence.

Exemple : Extraction de $\sqrt{238}$ au millième

- étape 1 : écrire le nombre dont on veut extraire la racine comme le dividende d'une division.
- étape 2 : séparer en tranches de deux chiffres à partir de la droite ; la dernière tranche à gauche peut n'avoir qu'un chiffre
- étape 3 : extraire la racine carrée de la première tranche, c'est-à-dire trouver le plus grand nombre x tel que $x^2 \leqslant 2$. Ici c'est **1**. L'écrire au diviseur.
- étape 4 : soustraire le carré de ce nombre à la tranche et abaisser la tranche suivante. On a 138
- étape 5 : prendre le double du chiffre du diviseur,
- étape 6 : chercher le chiffre qui donne le produit le plus proche, en restant inférieur au nombre abaissé : par exemple ici on doit faire $2...\times...\leqslant 138$, on essaie avec 21×1, 22×2, 23×3,... et on obtient $25\times5 = 125$. Le deuxième chiffre de la racine est **5**
- étape 7 : écrire le chiffre obtenu au diviseur et soustraire le produit obtenu, 125, au nombre abaissé
- étape 8 : abaisser la tranche suivante : si on est au bout du nombre entier et que l'on veut poursuivre l'extraction,
  on abaisse une paire de zéros et on met une virgule au diviseur.
- étape 9 et 10 : poursuivre le processus jusqu'à la précision demandée, chaque tranche de deux chiffres abaissée produisant un chiffre supplémentaire dans l'écriture de la racine .



Cette figure illustre le fonctionnement de l'algorithme de la potence utilisé pour extraire des racines carrées à la main

Pour des exemples de mise en œuvre et une justification théorique de la méthode, vous pouvez visualiser la vidéo de la chaîne YouTube Hans Samble - Maths au lycée, qui propose par ailleurs de très nombreuses vidéos mathématiques de qualité :

Calcul d'une racine carrée à la main Exemples et explications (Video Youtube)

L'implémentation en Python s'appuie sur les chaînes de caractères, un nombre entier pouvant être converti en une chaine de caractères qui sera ensuite découpée en tranches de longueur 2. Le programme ci-dessous s'applique à des entiers car l'algorithme de la potence ne tient pas compte de la virgule : un nombre décimal est considéré sans sa virgule et celle-ci peut être indiquée dans la racine mais non prise en compte dans le processus de calcul.

Il faut au préalable construire une fonction de tranchage qui tronçonne la chaine formée par le nombre en chaînes de deux chiffres (voir exécution en ligne) :

```
def decoupage_tranches(nombre):
    """découpe un entier en blocs de 2 chiffres en partant de la droite"""
    chaine_nombre = str(nombre)
    liste_tranches = []
    while len(chaine_nombre) >= 2 :
        tranche= chaine_nombre[-2]+ chaine_nombre[-1] # extraction de la tranche la plus à droite
        liste_tranches= [tranche] + liste_tranches # ajout de la tranche en tête de liste
        chaine_nombre= chaine_nombre[:-2] # on prend la liste tronquée de ses deux derniers éléments
    if chaine_nombre != '': # on rajoute un éventuel reste si le nombre de chiffres est impair
        liste_tranches= [chaine_nombre] + liste_tranches
    return liste_tranches
```

l'appel de cette fonction donne une liste de chaînes de deux caractères, sauf éventuellement pour la dernière :

```
>>> decoupage_tranches(670012000)
['6', '70', '01', '20', '00']
```

Cette fonction est ensuite utilisée pour mettre en œuvre l'algorithme de la potence (voir exécution en ligne) :

```
def racine_potence(nombre,nb_decimales):
    """renvoie la racine carrée d'un nombre entier, avec une précision spécifiée par nb_decimales"""
    nombre = int(str(nombre)+'00'*nb_decimales) # reconstitution du nombre en l'y adjoignant des tranches de '00' pour aller jusqu'à la précision souhaitée
    liste_tranches = decoupage_tranches(nombre)
    # gestion de la première tranche
    dividende = int(liste_tranches.pop(0)) # on enlève la première tranche de la liste
    chiffre = 0
    while chiffre**2 <= dividende : # recherche de la racine carrée de la première tranche
        chiffre = chiffre + 1
    chiffre = chiffre - 1
    racine = chiffre
    reste = dividende - chiffre**2
    # gestion des autres tranches
    while liste_tranches != []:
        dividende = int(str(reste) + liste_tranches.pop(0))
        double = 2 * racine
        chiffre = 0
        while (double*10 + chiffre)*chiffre <= dividende :
            chiffre = chiffre + 1
        chiffre = chiffre - 1
        racine = 10*racine + chiffre
        reste = dividende - (double*10 + chiffre)*chiffre
    return racine/10**(nb_decimales)
```

L'appel de la fonction donne :

```
>>> racine_potence(2387860,6)
1545.270202
```

Si l'on veut la racine carrée d'un nombre décimal, on le transforme en entier en le décalant d'un nombre pair de rangs $n = 2p$ vers la droite et on calcule cet entier à la précision souhaitée puis on récupère la racine carrée du nombre décimal en décalant de p rang(s) vers la gauche :

```
# calcul de la racine carrée de 1240.67835
>>> racine_potence(1240678350,2) # on décale de 6 chiffres pour avoir un nombre entier
35223.26 # on décale de 6/2=3 chiffres vers la gauche pour retrouver la racine carrée du nombre de départ
>>> math.sqrt(1240.67835)
35.22326432913338
```

Document joint

Sources des documents du parcours sur les racines carrées (Zip de 635.5 ko)

Cette archive contient tous les fichiers (fichiers tex, GeoGebra, scripts Python, illustrations,...) utilisés lors de la séquence.