Algorithme d'approximation numérique

La propriété des valeurs intermédiaires rencontrées en Terminale est très utile pour justifier l'existence d'une solution d' une équation du type $f(x) = 0$.
En revanche l'algorithme qui permet par manipulations successives du tableur, de rechercher une valeur approchée à la précision souhaitée, n'est pas très convaincant. Pire, il n'y a rien d'automatique au sens où il suffirait de lancer un programme pour obtenir le résultat souhaité.
On se propose dans cette activité de découvrir des algorithmes numériques qui livrent la valeur approchée d'une équation à la précision souhaitée. On s'intéressera aussi à la rapidité de la convergence de ces instructions, sans se poser d'ailleurs la question de savoir si cela converge vraiment...
? Ensuite nous utiliserons le langage Python pour implémenter ces algorithmes.

Préambule :

On utilisera la fonction f définie pour tout x par : $f(x) = x^2 - 2$, et on utilisera les différents algorithme pour rechercher une valeur approchée α de la solution positive de l'équation $f(x) = 0$.
Évidemment, $\alpha = \sqrt{2} \approx 1.4142135623731...$
On suppose connues deux valeurs initiales a_0 et b_0 pour lesquelles $f(a_0) < 0$ et $f(b_0) > 0$. Dans tous les cas il s'agit de construire une suite de valeurs qui converge vers α et si possible , très vite !

Algorithme n°1 : Par dichotomie

Voici le premier algorithme assez naturel et facile à comprendre : on appelle m_0 la moyenne de a_0 et b_0(le milieu quoi...) et on calcule $f(m)$: si $f(m) > 0$ on affecte m à b_1 et a_0 à a_1, sinon m à a_1 et b_0 à b_1 et on recommence avec m_1,... . À chaque boucle, itération, on diminue donc l'intervalle initial de moitié.
Voici l'algorithme et son implémentation en python :

Entrées :
ϵ :précision
a, b :bornes gauche et droite
Traitement :
Tant que $b - a > \epsilon$
$m \longleftarrow \dfrac{a+b}{2}$
Si $f(m) \leq 0$:
$a \longleftarrow m$
sinon :
$b \longleftarrow m$ Fin Tant que
Sortie :
Valeurs de a et b

```
def f(x):
    return x**2-2

def dichotomie(p, a = 1, b = 2):
    '''p:un entier positif
       a: vaut 1 par defaut
       b: vaut 2 par defaut'''
    while b-a >10**(-p):
        m = (a + b)/2
        if f(m)<= 0:
            a = m
        else:
            b = m
    return a, b
```

① En modifiant le fichier DICHOTOMIE.PY, créer un compteur qui compte le nombre N de boucles nécessaires pour déterminer un encadrement de α à la précision 10^{-p}.

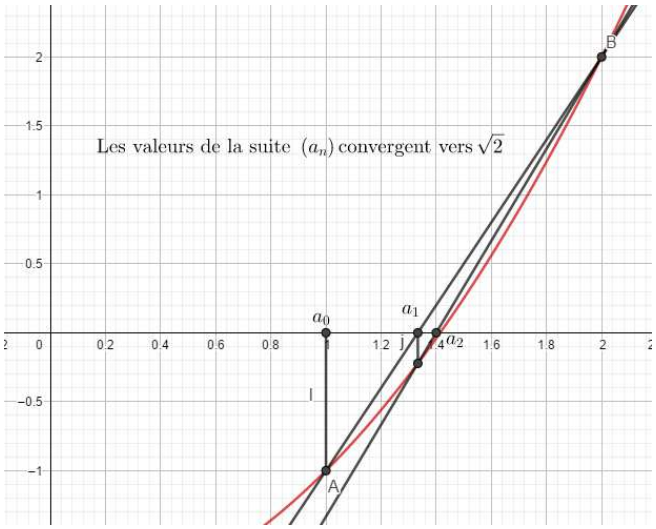
② Compléter alors le tableau suivant :

P	1	2	3	4	5
Nombre de boucles N					

③ Quelle relation existe t-il entre p et N ?

ALGORITHME N°2 : AVEC DES SÉCANTES

La méthode suivante est appelée **méthode de la sécante**. La valeur de m n'est plus déterminée par la moyenne des valeurs initiales a et b mais par l'intersection de la sécante (AB) avec l'axe des abscisses (A est le point de la courbe d'abscisse a, et B d'abscisse b...).Le graphique suivant illustre l'algorithme sur plusieurs itérations.



On suppose que nous sommes dans des conditions idéales où par exemple la sécante coupe toujours l'axe des abscisses. Avec la fonction f considérée sur l'intervalle $I = [1; 2]$, c'est effectivement le cas.

① Justifier que $m = \dfrac{a \times f(b) - b \times f(a)}{f(b) - f(a)}$.

② La convexité de la courbe explique que pour $f(m) < 0$ quelle que soit la valeur de m calculée à chaque itération. Ainsi à chaque boucle ,c'est la valeur de a qui est réaffectée avec celle de m. Le programme suivant donne une valeur approchée de α à la précision 10^{-p} souhaitée :

```
def f(x):
    return x**2-2

def secante(p, a =1, b = 2):
    m = (a*f(b)-b*f(a))/(f(b)-f(a))
    while m-a>10**(-p):
        a = m
        m = (a*f(b)-b*f(a))/(f(b)-f(a))
    return m
```

Voila ce que donne la méthode avec une précision de 10^{-5} : SECANTE$(5) = 1.4142131979695434$.

Compléter alors le tableau suivant :

P	1	2	3	4	5
Nombre de boucles N					

Algorithme n°3 : La méthode de Newton Raphson

Dans la méthode précédente, les différentes valeurs de a calculées constituent une suite (a_n) avec $a_0 = 1$ et $a_1 = \dfrac{a_0 \times f(b_0) - b_0 \times f(a_0)}{f(b_0) - f(a_0)}$,... obtenues par l'intersection de la droite (AB) avec l'axe des abscisses.
Dans cette nouvelle méthode, le principe est le même :on construit une suite (x_n) qui converge vers $\alpha = \sqrt{2}$.
L'algorithme est le suivant :

❶ On choisit une valeur initiale $x_0 > \alpha$ sur l'axe des abscisses ;

❷ x_1 est l'abscisse de point d'intersection de la **tangente** à $\mathcal{C}_f$ au point d'abscisse x_0 avec l'axe des abscisses.

❸ et ainsi de suite,...x_{n+1} est l'abscisse de point d'intersection de la **tangente** à $\mathcal{C}_f$ au point d'abscisse x_n avec l'axe des abscisses.

① Faites un dessin qui illustre la situation.

② Prouver que $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$.

③ Soit la suite (x_n) définie pour tout n par :

$$\begin{cases} x_0 > \sqrt{2} \\ x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)} \end{cases}$$

- ❶ Justifier que pour tout n,pour notre fonction choisit en préambule, on a :$x_{n+1} = \dfrac{1}{2}(x_n + \dfrac{2}{x_n})$
- ❷ Démontrer par récurrence que $x_n > \sqrt{2}$.
- ❸ Démontrer que la suite (x_n) est décroissante sur $\mathbb{N}$.
- ❹ En déduire que la suite converge vers un nombre l.
- ❺ Déterminer l.
- ❻ Démontrer l'inégalité : $(x_{n+1} - \sqrt{2}) < (x_n - \sqrt{2})^2$. Si x_n donne une précision à 10^{-k} de $\sqrt{2}$, quelle sera la précision du terme suivant x_{n+1} ?

④ Le fichier `comparaison.py` génère un graphique qui compare la rapidité de convergence des trois algorithmes précédents par la mesure du nombre de boucles nécessaires pour obtenir une valeur approchée de α à 10^{-p}. Exécuter le fichier.Quel est l'algorithme le plus rapide ?