TP - Algorithmes dichotomiques

PARTIE 1 : RECHERCHE D'UNE SOLUTION D'EQUATION

A de nombreuses reprises en sciences expérimentales, on est amené à résoudre une équation à une inconnue, pour exemple pour déterminer le couple moteur en fonction de la pression à obtenir pour une pompe alimentant un pilote de bateau automatique, ou encore pour déterminer un point d'impact d'un projectile dont on a établi les équations de mouvements (hauteur h(t) = 0 pour t = ? puis déduction la position du point d'impact).
Un autre exemple : on peut chercher la vitesse de rotation d'un manège en fonction de l'angle de levée des sièges portant les enfants et ainsi anticiper la commande en vitesse du moteur d'entraînement du manège.

Dans certains cas, la résolution peut s'avérer complexe « à la main », une résolution numérique sera alors intéressante.

I) Recherche du zéro de la fonction *g* : x → x – cos(x) sur l'intervalle [0,1]

💻 Écrivez en langage python le script définissant la fonction g.

Après avoir copié-collé puis ouvert le fichier intitulé « aide pour tracé.py » depuis le commun, tracez la courbe de la fonction sur l'intervalle [0,1]

Déterminez la valeur de g(0.5), la valeur de x annulant g se situe dans quel intervalle parmi [0 ; 0.5] et [0.5 ; 1] ? Tracez désormais la courbe de g sur l'intervalle contenant le zéro de la fonction.

Déterminez ensuite la valeur prise par la fonction pour l'abscisse milieu de l'intervalle retenu. Le zéro de g est-il supérieur ou inférieur à l'abscisse milieu ? En fonction de la réponse, retenez l'intervalle contenant le zéro afin de diviser par deux sa longueur à nouveau. Puis tracez la courbe sur cet intervalle.

Effectuez encore deux ou trois itérations de l'algorithme, puis donnez la valeur du zéro de la fonction à 0.0625 près. Combien d'itérations sont nécessaires pour obtenir cette précision ?

II) Méthode de recherche de zéro d'une fonction par dichotomie

Dans la partie précédente, vous venez d'appliquer l'algorithme de recherche du zéro d'une fonction à une valeur 0.0625 près par dichotomie.
Le but de cette partie est d'écrire un algorithme puis son implémentation en python pour déterminer une valeur approchée d'une solution unique à l'équation f(x) = 0 sur un intervalle [a, b] à une valeur ε près. Cette méthode nécessite des contraintes sur la connaissance de la fonction g.

Principe général :

Si f est continue sur [a, b] et si $f(a)$ et $f(b)$ sont de signes contraires alors il existe x Є [a, b] tel que f(x) = 0 (ceci est le théorème des valeurs intermédiaires).
Si de plus, f est strictement monotone sur [a, b], x est unique (ceci est le théorème de la bijection).

Méthode :

- On affecte respectivement à g et d la valeur des paramètres d'entrées a et b (g et d seront alors des variables de la boucle).
- Soit le milieu de l'intervalle [g, d] : m = (g + d) / 2,
- Si f(g) et f(m) sont de même signe (f(g) f(m) > 0) alors x Є [m, d], sinon x Є [g,m],
- Dans le premier cas, g prend la valeur de m (g ← m), dans le deuxième cas, d prend la valeur de m (d ← m),
- On réitère le procédé dans le nouvel intervalle [g,d] (g ou d a changé de valeur) et ainsi de suite,
- On s'arrête lorsque la longueur de l'intervalle (d-g) est inférieure à ε,
- On renvoie alors la valeur de g (ou celle de d) .

Les avantages de cette méthode sont :

- une écriture de l'algorithme simple.
- un niveau mathématique simple.
- un nombre d'itérations connu à l'avance, indépendant de f.

L'inconvénient est que le temps de calcul est plus long que d'autres méthodes, on en étudiera plus tard dans l'année.

Pseudo-code :

✍ Écrivez sur feuille le pseudo-code d'une fonction **dichotozero(f,a,b,ε)** qui prend en argument une fonction f, la borne inférieure a et la borne supérieure b de l'intervalle d'étude, et enfin la précision souhaitée ε, et qui retourne le zéro de la fonction sur l'intervalle.

Implémentation en langage python :

💻 Implémentez en langage python votre fonction **dichotozero**
Testez votre fonction **dichotozero** avec la fonction de la partie I sur l'intervalle [0,1] avec une précision de 0,0001
Ajoutez à votre fonction la vérification que les images des bornes inférieures et supérieures f(a) et f(b) sont de signes différents. Votre fonction affichera un message d'erreur dans le cas contraire.
Ajoutez un compteur pour obtenir le nombre d'itérations nécessaires pour obtenir le résultat.
Déclarer une variable x_{0d} à laquelle vous affecterez ce que retourne **dichotozero(g,0,1, 10^{-4}).**
Puis demandez l'affichage de g(x_{0d}).

Complexité temporelle de l'algorithme :

☺ Afin d'évaluer l'impact de l'algorithme sur les temps d'exécution, on demande ici de déterminer une borne supérieure du nombre d'itérations nécessaires à son exécution.

De quels paramètres dépend le nombre d'itérations nécessaires ?

PARTIE 2 : EXPONENTIATION RAPIDE

On note $\left\lfloor \frac{n}{2} \right\rfloor$ (avec python math.floor du module math ou encore n//2 pour un entier naturel n) la partie entière de $\frac{n}{2}$.

Soit les trois fonctions suivantes permettant chacune de calculer a^n pour $n \in \mathbb{N}$:

Puissance1(a,n) renvoie le calcul de puissance de python : a^n = a**n ;

Puissance2(a,n) réalise un calcul itératif (avec une boucle énumérée) en exploitant :

$a^k = a \times a^{k-1}$;

Puissance3(a,n) réalise un calcul itératif en exploitant :

Si n est pair : $a^n = \left(a^{\frac{n}{2}}\right)^2$

Si n est impair : $a^n = a\left(a^{\frac{n-1}{2}}\right)^2$

1) Ecrivez le script de la fonction Puissance1(a,n).

Dans les questions suivantes, il est interdit d'utiliser la commande **.

2) Ecrivez le script de la fonction itérative Puissance2(a,n).

3) Ecrivez sur feuille le pseudo-code de la fonction Puissance3(a,n).
   On pourra déclarer trois variables de boucles u, v, k telles que la propriété suivante sera toujours vérifiée en cours de boucle :

   $u\, v^k = a^n$

   Remarque : une propriété qui reste vraie tout au long de l'exécution d'une boucle est appelée « invariant de boucle ». En fin de boucle, k aura la valeur 0, donc u aura la valeur du résultat recherché a^n. Démontrer que la relation est bien un invariant de boucle revient donc à démontrer que l'algorithme aboutit à ce qui est attendu dans tous les cas considérés.

   Avant de construire votre fonction, répondez à ces questions :
   Quelles valeurs doivent prendre u, v et k avant la boucle ?
   Quelles relations de récurrences définir pour ces trois variables ?

   Implémentez ensuite en langage python le script de la fonction Puissance3(a,n).

4) Déterminez une majoration du nombre d'itérations de boucle des fonctions Puissance2(a,n) et Puissance3(a,n).

5) Testez ensuite ces deux fonctions avec la commande time.perf_counter( ) du module time.

6) Comparez les temps d'exécution des fonctions Puissance2(a,n) et Puissance3(a,n) avec celui de la fonction Puissance1(a,n). Que remarquez-vous ?