T.P. 5 : algorithmes de dichotomie

1 Jeu de devinettes

1.1 Premier programme :

a) Ecrire une fonction `devinette()` qui fonctionne de la manière suivante :

• La machine choisit un nombre aléatoire entre 1 et 1000 qu'elle ne vous dit pas (le nombre mystère). Vous pouvez pour cela utiliser la fonction `randint` du module `random` : `randint(a,b)` renvoie un entier aléatoire dans $[\![a, b]\!]$.

• Elle vous demande de rentrer un nombre, (avec `input`) et elle vous dit si votre nombre est plus grand ou plus petit que son nombre mystère et vous recommencez jusqu'à ce que vous ayez gagné. Elle vous dit alors « Bravo, vous avez gagné en (nombre de coups) coups »

b) Définir votre stratégie pour toujours gagner en moins de 10 coups.

Pourquoi 10 coups ?

1.2 Deuxième programme :

Vous échangez les rôles avec la machine. Là c'est plus intéressant car vous devez implémenter dans la machine la stratégie que vous avez définie au paragraphe précédent, pour qu'elle gagne toujours en moins de dix essais.

2 La dichotomie en analyse pour la recherche d'un zéro d'une fonction continue

Hyp. On se donne une fonction $f \in \mathcal{C}([a, b], \mathbb{R})$ explicite telle que $f(a).f(b) < 0$. Le T.V.I. dit qu'il existe un $c \in]a, b[$ tel que $f(c) = 0$. Le problème est d'avoir une approximation numérique de c.

Je voudrais insister sur les deux points de vue complémentaires : maths et info.

2.1 Présentation mathématique (pas de travail à faire, lire seulement !)

A partir de l'hyp. ci-dessus, on va construire deux suites (a_n) et (b_n) :

a) **Définition par récurrence :**

• Initialisation : on pose $a_0 = a$ et $b_0 = b$.

• A l'étape n, on suppose qu'on a défini a_n et b_n : on considère $m = (a_n + b_n)/2$.

— Si $f(a_n).f(m) > 0$ on pose $a_{n+1} = m$ et $b_{n+1} = b_n$.

— Si $f(a_n).f(m) \leq 0$, on pose $a_{n+1} = a_n$ et $b_{n+1} = m$.

b) **Propriété évidente des suites (a_n) et (b_n) ainsi définies** :

(i) par construction chaque intervalle $[a_{n+1}, b_{n+1}]$ est emboîté dans $[a_n, b_n]$ (avec un borne commune), et de longueur moitié : $b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n)$

(ii) Par réc. immédiate : $\forall\, n \in \mathbb{N}$, $b_n - a_n = \frac{1}{2^n}(b_0 - a_0) = \frac{b-a}{2^n}$.

(iii) La déf. précédente ne met pas à part le cas où $f(m) = 0$. En fait si par hasard si $f(m) = 0$ à l'étape n, alors $b_{n+1} = m$ et pour tout $k \geq n + 1$, $b_k = m$.

(iv) Par T.V.I. à chaque étape le segment $[a_n, b_n]$ contient au moins un zéro de f. Du point de vue de l'approximation numérique, a_n et b_n sont donc des valeurs approchées d'un tel zéro à $(b - a)/2^n$ près et leur milieu $m = (a_n + b_n)/2$ à $(b - a)/2^{n+1}$ près.

2.2 Version informatique

En informatique, les valeurs successives des suites (a_n) et (b_n) sont stockées toujours dans les mêmes variables informatiques `a` et `b` dont le contenu est modifié au fur et à mesure.

a) Ecrire une fonction en PYTHON, qui prend comme argument : une fonction `f`, des réels `a` et `b`, et une précision `epsilon`, qui renvoie :
   — un message d'erreur si $f(a).f(b) > 0$
   — sinon, un zéro de f dans $[a, b]$ à la précision ε, calculé par la méthode de dichotomie.

b) Appliquer la fonction précédente à $f = \sin$, $a = 3$, $b = 4$ et $\varepsilon = 10^{-7}$.

Attention : ne JAMAIS faire de test d'égalité avec des flottants ! Un flottant ne représente un réel qu'à une certaine précision près ! Cela n'a pas de sens de dire qu'on tombe « pile » sur un zéro.

3 Algorithme de recherche dichotomique dans une liste triée

Au T.P. sur les listes, on a écrit des algorithmes qui renvoient, pour une liste `L` et un élément `a`, à quel indice apparaît `a` dans `L` (et donc font la même chose que la méthode `index` sur les listes).

Ici, on fait l'hypothèse beaucoup plus forte que `L` est une liste d'entiers *déjà triée dans l'ordre croissant.*

L'idée est la suivante : on coupe le tableau en deux par le milieu et on détermine si la valeur `a` que l'on cherche est dans la moitié gauche ou la moitié droite, en la comparant simplement à la valeur centrale. Puis on répète le processus sur la portion sélectionnée.

Exercice à faire : Programmez en PYTHON une fonction `recherche_dichoto` qui prend comme arguments une liste `L` d'entiers, déjà triée dans l'ordre croissant, et un entier `a`, et qui renvoie un indice i tel que `a` apparaît dans à l'indice `i` dans `L` ou bien renvoie `None` si `a` n'apparaît pas.

Indication – : On pourra utiliser deux variables `g` et `d` pour `gauche` et `droite` qui délimitent la portion du tableau dans laquelle `a` doit être cherchée. Ainsi, on initialise `g=0` et `d=len(L)`. Ensuite on joue sur `m=(g+d)//2` (pourquoi avec un double slash) ?

Suite de l'exercice : Justifier que si `L` est de longueur `n`, le nombre N d'étapes (tours de boucles) nécessaires que que l'algorithme termine est majoré par $\log_2(n)$ (ou $\log_2(n+1)$).Comme le nombre d'opérations (tests d'égalités ou d'inégalités) est constant à chaque tour de boucles, on dira que le programme a un temps de calcul *logarithmique* en la taille des données.

4 Une alternative à la dichotomie pour la recherche de zéros d'une fonction : méthode de la fausse position

Dans cette méthode, on note $A = (a, f(a))$, $B = (b, f(b))$ et on cherche le point d'intersection $(x_0, 0)$ du segment $[A, B]$ avec l'axe des abscisses. On sait le calculer via l'équation de (AB). On peut considérer x_0 comme une première approximation du zéro de f que l'on cherche.

Ensuite, on itère le procédé en l'appliquant sur l'intervalle $[a, x_0]$ si $f(x_0)$ est du même signe que $f(b)$ et sur $[x_0, b]$ sinon : on obtient un nouveau point $(x_1, 0)$ (cf. la figure ci-dessous).

Reprendre les questions du § 2 en remplaçant la méthode de dichotomie par cette méthode.

Attention : on pourra s'interroger sur le test d'arrêt !