TP 10 informatique

BCPST 1 2019-2020

Bibliothèques et dichotomie

Exercices

Exercice 1. Dans les différents programmes, on utilisera `linspace`.

1. Utiliser l'aide pour comprendre comment fonctionne cette fonction.
2. Construire une liste de nombres compris entre 2 et 50 réparties de manière uniforme.
3. Construire une liste de 200 nombres répartis uniformément sur $[0, 1]$.
4. Construire une liste de 100 nombres répartis sur $[1, 10[$. La liste exclura le nombre 10.

Exercice 2. À l'aide de la bibliothèque `numpy`, écrire les fonctions suivantes :

1. $f_1 : x \mapsto x\exp(x)$ 2. $f_2 : x \mapsto \sin(x)x^2$ 3. $f_3 : x \mapsto \arctan(x^3)$.

Tracer leurs courbes représentatives.

Exercice 3. Lorsqu'une liste L est triée dans l'ordre croissant, il est facile d'y faire une recherche d'un élément x : on compare x avec l'élément du milieu $L[m]$. Si $x > L[m]$, on cherche dans $L[m + 1 :]$. Si $x < L[m]$, on cherche dans $L[: m]$. Sinon, on a trouvé x et on renvoie m.

Voici le code correspondant :

```
dichotomie(L,x) :
    debut=0
    fin=longueur L -1.
    tant que debut<=fin :
        milieu=(debut+fin)//2
        si L[milieu] est égal à x :
            valeur de retour milieu
        sinon :
            si L[milieu]>x :
                fin=milieu-1
            autre :
                debut=milieu+1
    valeur de retour -1
```

1. Écrire une fonction `dichotomie`.
2. Écrire une fonction `dichotomiedeux` où on rajoute une variable `comparaisons` qui compte le nombre de fois qu'on compare x à un élément de L. On mettra également cette variable en valeur de retour.
3. Écrire une fonction `dichotomiedecroissant` qui prend en arguments une liste L décroissante et un élément x et qui applique une recherche par dichotomie à une liste L décroissante.

Exercice 4. On veut résoudre numériquement l'équation $f(x) = 0$, dans le cas où f est une fonction continue sur un segment $[a, b]$, avec $f(a) < 0$ et $f(b) > 0$. Pour obtenir une valeur approchée d'une solution, on cherche un segment $[a_n, b_n]$ qui contient c tel que $f(c) = 0$ et $|b_n - a_n| \leq \frac{|b-a|}{2^n}$. Pour cela, on peut adapter la recherche par dichotomie. On présente un algorithme à compléter qui permet de faire la résolution demandée :

```
zero(f,a,b,n) :
    an=a
    bn=b
    cn=(an+bn)/2
    tant que abs(bn-an)>abs(b-a)/(2**n) :
        si f(cn)>0 :
            ....
        si f(cn)<0 :
            ....
        sinon :
            valeur de retour cn
    valeur de retour [an,bn]
```

1. Compléter le pseudo-code précédent et en déduire une fonction Python qui permet de calculer de manière approchée un zéro d'une fonction f.
2. En déduire une fonction qui permet de calculer $\sqrt{2}$ de manière approchée.
3. Adapter l'algorithme `zero` pour une fonction f avec $f(a) > 0$ et $f(b) < 0$.

Exercice 5. (Difficile, si vous avez le temps) L'algorithme de dichotomie peut également être adapté pour trouver un algorithme de résolution de certains jeux comme le jeu "devine ta séquence". On considère deux joueurs, l'un (Joueur 1) connaissant un séquence d'ADN S à k lettres, l'autre (Joueur 2) devant deviner celle-ci. La partie se déroule ainsi : le joueur deux a l tentatives pour deviner cette séquence. Pour chaque tentative, le joueur 2 propose une séquence R. Si $S = R$, il a alors gagné. Sinon, le joueur 1 donne le nombre de lettres correctes bien placées et le nombre de lettres présentes dans la séquences parmi les autres lettres puis le joueur 2 continue à donner des propositions. Si ce dernier ne trouve jamais la séquence il alors a perdu.

Voici un exemple de partie : séquence à trouver $S = ATTGCCTTC$, nombre de tentatives maximum $l = 10$.

— joueur 2 propose : $CATTGTTTC$. Joueur 1 répond : bien placés : 4, présents mal placés : 4
— joueur 2 propose : $TTTTTTTTT$. Joueur 1 répond : bien placés : 4, présents mal placés : 0
— joueur 2 propose : $ATTTTTTTT$. Joueur 1 répond : bien placés : 5, présents mal placés : 0
— joueur 2 propose : $ATTGGCCCT$. Joueur 1 répond : bien placés : 5, présents mal placés : 3
— joueur 2 propose : $ATTGCCTTC$. Joueur 1 répond : bien placés : 9, présents mal placés : 0. C'est gagné !

L'objectif de la suite est d'écrire un algorithme permettant la résolution du jeu.

1. Écrire une fonction `aleatoire(n)` qui prend en argument un entier n et qui renvoie une séquence d'ADN de longueur n.
2. Écrire une fonction `liste_seq(n)` qui prend en argument un entier naturel n et qui renvoie la liste des séquences de longueur n.
3. Écrire une fonction `comparaison(seq1,seq2)` prenant en arguments deux séquences d'ADN et qui renvoie le nombre de bien placés et le nombre de présents mal placés.
4. Écrire une fonction `pas_bon(L,s,n,m)` prenant en argument une liste de séquences L, une séquence s deux entiers n, m et qui supprime tous les éléments x de L ne vérifiant pas `comparaison(s,x)==n,m`.
5. Récupérer la fonction `parties` sur la page web. Vérifier que tout fonctionne avec les programmes que vous avez écrits.