

Test intégrale

Ecrire une fonction **Intgrl**(f,a,b,n) où :

- a et b sont les bornes (ordonnées) d'un intervalle,
 - n est le nombre de subdivisions de [a;b]
 - f désigne une fonction définie sur [a;b]
- qui renvoie la somme de aires des trapèzes.

Exercice : Comparaison

Soit l'équation $E: x \ln(x) = 1$.

On admet (si un doute vérifiez-le rapidement) que E admet une unique solution α sur \mathbb{R} et que $\alpha \in [1; e]$.
(On prendra $e \approx 2.8$)

A.

1. Complétez la fonction suivante renvoyant une valeur approchée à epsilon près de cette solution par dichotomie.

```
def dichotomie(f, epsilon):  
    a = ...  
    b = ...  
    while abs(...  
        c = (a+b)/2  
        if f(a)*f(b) < 0 :  
            ... = c  
        else :  
            ... = c  
    return ...
```

2. Vérifier qu'à 10^{-6} près :
 $\alpha \approx 1.763223$.
3. Modifier cette fonction afin qu'elle affiche le nombre de répétition de la boucle.

B.

Modifiez la fonction précédente renvoyant une valeur approchée à epsilon près de cette solution par la méthode des cordes (dite de Lagrange).

Avec affichage du nombre de répétition de la boucle.

Comparer avec le précédent.

C.

On souhaite comparer avec la méthode Newton, mais le test d'arrêt ne permet pas d'obtenir une valeur approchée avec une précision donnée!

Nous allons donc nous servir des résultats précédents.

f et fp étant des fonctions saisies en python, et correspondant à f et f' alors l'algorithme de Newton est :

```
x = 2.8  
compteur = 0  
while abs(x - 1.763223) < 10**(-6):  
    x = ...  
    compteur = ...  
print(compteur)
```

Comparer

Exercice : Suite définie implicitement

Soit f_n une suite de fonction définie sur \mathbb{R} pour tout $n \in \mathbb{N}^*$ tel que :

$$f_n(x) = x^n + x^{n-1} + x - 1$$

1. Montrer que, pour tout $n \in \mathbb{N}^*$, l'équation $f_n(x) = 0$ possède une unique solution sur $[0; 1]$.

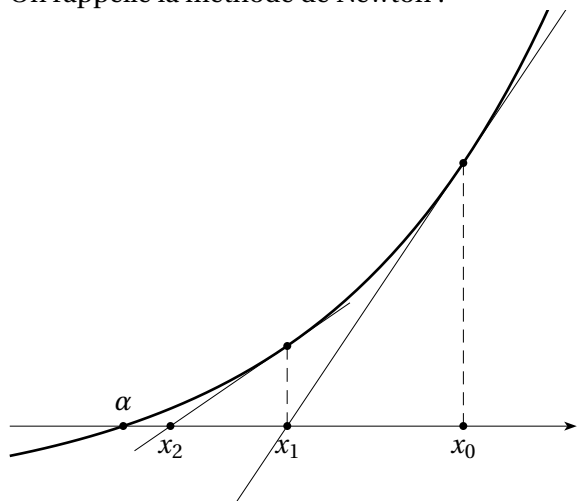
Notons u_n cette solution.

2. Ecrire un programme Python qui :
 - définie une fonction f(x, n) renvoyant la valeur $f_n(x)$,
 - définie une fonction dichotomie(f, n) renvoyant une valeur à 10^{-3} près de u_n ,
 - calcule et affiche les 100 premières valeurs approchées de (u_n) .
3. Conjecturer du comportement de cette suite.

Exercice : Les délices empoisonnées de la méthode de Newton

Partie A

On rappelle la méthode de Newton :



Supposons que les fonctions f et f' soient définies en en-tête du programme,
compléter l'algorithme afin de calculer les 20 premières valeurs de la suite générée par la méthode de Newton :

```
x = x_0

pour k allant de 1 à 20 faire :

    x =
```

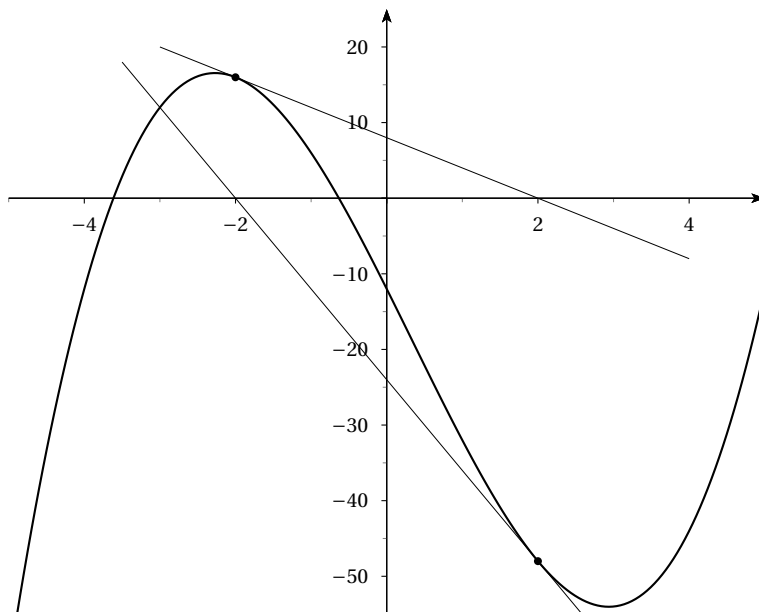
Partie B

Soit la fonction f définie sur \mathbb{R} par : $f(x) = x^3 - x^2 - 20x - 12$.

Considérons la suite définie par la méthode de Newton en partant de $x_0 = 2$

Ecrire un programme en Python qui :

- défini la fonction f ainsi que sa fonction dérivée que vous noterez f' ,
- calcule et affiche les valeurs successives de l'algorithme de Newton.



Exercice : Avec des listes

Lors d'une expérience on mesure un phénomène numérique au cours du temps et on dresse deux listes (de même longueur) :

- V : la liste des mesures supposées monotones,
- T : la liste des temps (en seconde, dans l'ordre croissant) correspondant à chaque mesure.

Exemple : $T = [0, 2, \dots]$ et $V = [0.5, 0.55, \dots]$ signifie que 0.5 a été mesuré à 0s, puis la valeur suivante (0.55) a été prise à 2s etc.

Ecrire une fonction `solution(valeur, V, T)`, qui renvoie un encadrement du temps pour lequel les mesures encadrent elles-même la valeur.

Indication :

- privilégiez la dichotomie,
- déterminez les indices i et j de cet encadrement,
- observez que le test d'arrêt de recherche de ces entiers ne dépend pas d'une précision.

Tester ce programme avec les listes : $T = [0, 2, 3, 5, 6, 8, 10]$ et $V = [0.5, 0.55, 0.7, 0.9, 1, 1.5, 1.6]$ avec 0.6 pour la valeur.

Modifier votre fonction `solution` afin de gérer (par un message d'erreur) le cas où il n'y a pas d'encadrement possible car `valeur` serait incompatible.