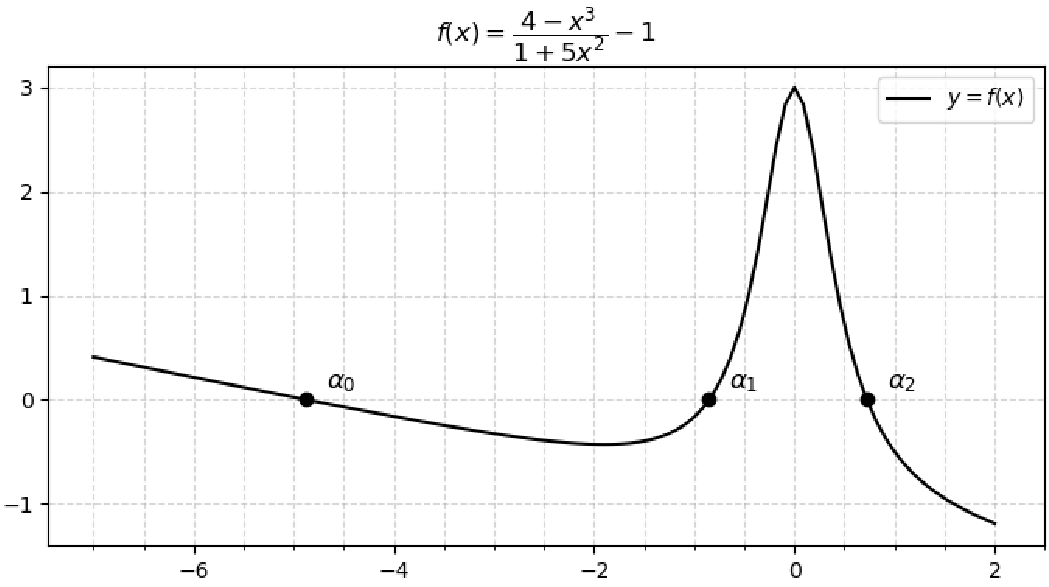
TP 14 - Révisions

Résolution d'équation non-linéaire

Soit la fonction réelle d'une variable réelle $f : \mathbb{R} \to \mathbb{R}$ définie par

$$f(x) = \frac{4 - x^3}{1 + 5x^2} - 1$$

dont la représentation graphique est donnée ci-dessous sur l'intervalle $[-7, 2]$. On cherche à approcher numériquement les trois racines α_i de la fonction, représentées sur la figure suivante.



a) **Recherche de α_0 :**
Pour approcher α_0, on propose d'utiliser la méthode de la bissection *à la main*, c'est-à-dire en appliquant les différentes étapes sans écrire de code.

En partant de l'intervalle $I_1 = [-7, -3]$, quels sont les intervalles I_k ($k = 1, 2, 3, 4$) considérés lors des quatre premières itérations de la méthode ainsi que les approximations correspondantes x_j de la racine ?

Solution

$I_1 = [-7, -3] \quad x_1 = -5$
$I_2 = [-5, -3] \quad x_2 = -4$
$I_3 = [-5, -4] \quad x_3 = -4.5$
$I_4 = [-5, -4.5] \quad x_4 = -4.75$

b) **Recherche de α_1 :**
Complétez le code ci-dessous aux endroits manquants de manière à ce qu'il permette d'approcher numériquement α_1 à l'aide de la méthode de Newton pour la fonction définie dans l'énoncé.

```
def f(x):
    return ______________________________

def f_prime ( x ) :
    return -(40*x+3*x**2+5*x**4)/(1+5*x**2)**2

def solve_Newton(f, x_0, fprime, eps, k_max):
    '''
    Resume :        Approximation d'un zero de la fonction f(x) a l'aide
                de la methode de Newton
    Parametres :
        f :         la fonction dont on recherche la racine
        x_0 :       une estimation initiale du zero de f
        fprime :    une fonction qui est la derivee de f(x)
        eps :       la tolerance souhaitee
        k_max :     le nombre maximum d'iterations autorisees
    Retourne :
        x :             un nombre approchant un zero de la fonction f
        evolution :     une liste contenant les valeurs x et f(x) a chaque
            iteration
    '''

#1. Initialisation des variables
    n_iter = 0

    x = ______________________________

    f_val = ______________________________

    evolution = [(x,f_val)]

#2. Methode de Newton
    while abs(f_val) > ______________ and n_iter < ______________:

        fprime_val = fprime(x)

        if ___________________________ < 1e-10:

            raise ValueError("Probable probleme de division par zero !")

        x = ______________________________

        n_iter += 1

        f_val = ______________________________

        evolution.______________________________((x,f_val))

    return ______________, ______________
```

Solution

```
def f(x):
    return (4-x**3)/(1+5*x**2)-1

def f_prime ( x ) :
    return -(40*x+3*x**2+5*x**4)/(1+5*x**2)**2

def solve_Newton(f, x_0, fprime, eps, k_max):
    '''
    Resume :        Approximation d'un zero de la fonction f(x) a l'aide
                de la methode de Newton
    Parametres :
        f :         la fonction dont on recherche la racine
        x_0 :       une estimation initiale du zero de f
        fprime :    une fonction qui est la derivee de f(x)
        eps :       la tolerance souhaitee
        k_max :     le nombre maximum d'iterations autorisees
    Retourne :
        x :             un nombre approchant un zero de la fonction f
        evolution :     une liste contenant les valeurs x et f(x) a chaque
            iteration
    '''

#1. Initialisation des variables
    n_iter = 0
    x = x_0
    f_val = f(x_0)
    evolution = [(x,f_val)]

#2. Methode de Newton
    while abs(f_val) > eps and n_iter < k_max:
        fprime_val = fprime(x)
        if abs(fprime_val) < 1e-10:
            raise ValueError("Probable probleme de division par zero !")
        x = x - f_val/fprime_val
        n_iter += 1
        f_val = f(x)
        evolution.append((x,f_val))
    return x, evolution
```

c) Est-ce que la méthode de Newton donnera le même résultat en partant de $x_0 = -0.5$ qu'en partant de $x_0' = 0.5$? Justifiez votre réponse.

Solution

Non, les deux résultats convergeront vers des racines différentes à cause de la pente de $f(x)$ qui change de signe autour de $x = 0$.

d) **Recherche de α_2** :
Montrez que la fonction d'itération

$$g(x) = \sqrt{\frac{3 - x^3}{5}}$$

est une fonction acceptable pour une méthode de point fixe (méthode de Picard) destinée à trouver une valeur approchée d'un zéro de $f(x)$, et converge vers la racine souhaitée en partant de $x_0 = 1$.

Solution

On veut montrer que :

(i) $g(\alpha) = \alpha$ si $f(\alpha) = 0$:

$$\begin{aligned} f(\alpha) = 0 &\Longrightarrow \frac{4 - \alpha^3}{1 + 5\alpha^2} - 1 = 0 \\ &\Longrightarrow 4 - \alpha^3 = 1 + 5\alpha^2 \\ &\Longrightarrow \alpha^3 + 5\alpha^2 = 3. \end{aligned}$$

En isolant α, on a :

$$\alpha^2 = \frac{3 - \alpha^2}{5} \Longrightarrow \alpha = \pm\sqrt{\frac{3 - \alpha^2}{5}}.$$

Ceci montre que $g(x) = \sqrt{\frac{3-x^2}{5}}$ satisfait bien le premier critère.

(ii) $|g'(1)| < 1$. Pour cela, calculons la dérivée de $g(x)$.

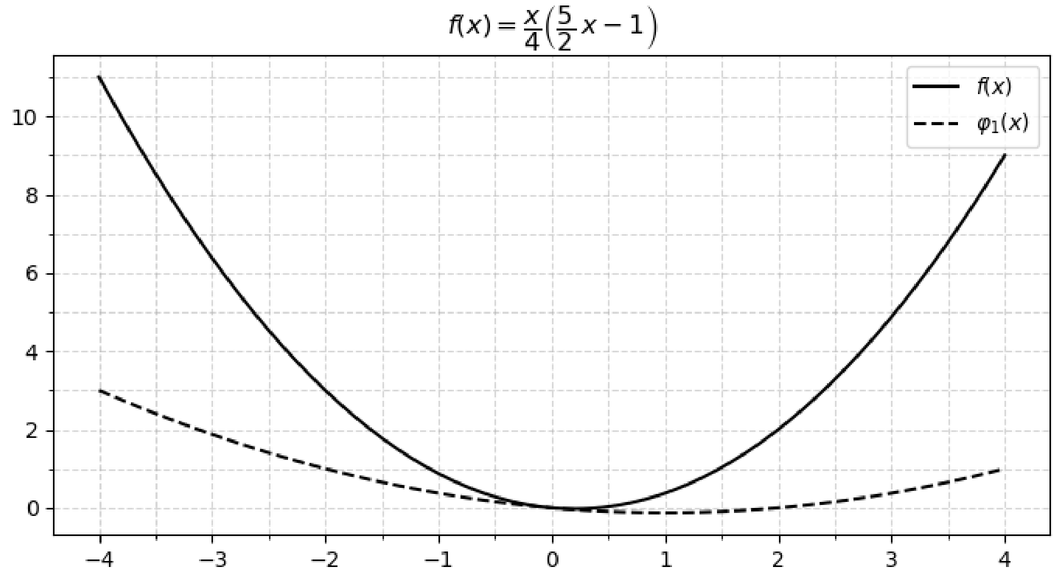
$$\begin{aligned} g'(x) &= \frac{1}{2}\left(\frac{3 - x^3}{5}\right)^{-1/2} \cdot \left(-\frac{3x^2}{5}\right) \\ &= -\frac{3x^2}{10}\left(\frac{3 - x^3}{5}\right)^{-1/2} \end{aligned}$$

Ainsi,

$$|g'(1)| = \frac{3}{10} \cdot \sqrt{\frac{5}{2}} < \frac{3}{10} \cdot \frac{5}{2} = \frac{15}{20} = \frac{3}{4} < 1$$

Intégration numérique

Soit la fonction
$$f(x) = \frac{x}{4}\left(\frac{5}{2}x - 1\right)$$
dont on cherche à approcher numériquement l'intégrale entre $a = -3$ et $b = 3$.



a) Donnez l'expression du premier élément $\varphi_1(x)$ de la base de Lagrange associée aux points $(t_1 = -2, t_2 = 0, t_3 = 2)$.

Solution

On part de la définition $\varphi_k(x) = \prod_{j, j\neq k} \frac{x - x_j}{x_k - x_j}$. Ainsi,
$$\varphi_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} = \frac{x(x-2)}{8} = \frac{1}{8}x^2 - \frac{1}{4}x.$$
On vérifie que $\varphi_1(-2) = 1$, et que $\varphi_1(0) = \varphi_1(2) = 0$.

b) En utilisant la méthode du point milieu composite et en considérant une partition régulière du domaine d'intégration en trois sous-intervalles, donnez une approximation numérique de l'intégrale $\int_{-3}^{3} f(x)\,\mathrm{dx}$.

Solution

$$\begin{aligned} J(f) &= \frac{x_{i+1} - x_i}{2} \sum_i \omega_1\, f\left(\frac{x_i + x_{i+1}}{2}\right) \\ &= \frac{h}{2}\left(2f(-2) + 2f(0) + 2f(2)\right) \\ &= 2(3 + 0 + 2) \\ &= 10. \end{aligned}$$

c) Nous cherchons maintenant à améliorer la précision en considérant une partition deux fois plus fine, c'est-à-dire $h' = \frac{h}{2}$. Que peut-on dire sur l'erreur $e_{h'}$ par rapport à e_h ?

Solution

Selon un théorème du cours :

$$e_{h'}^{\mathrm{PM}} = C \cdot (h')^{r+1} = C \cdot \left(\frac{h}{2}\right)^{r+1} = C \cdot \left(\frac{h}{2}\right)^{2} = \frac{1}{4} C \cdot h^4 = \frac{1}{4} e_h^{\mathrm{PM}}.$$

On s'attend donc à ce que l'erreur soit 4 fois plus petite.

d) Sans effectuer de calcul, que peut-on dire sur l'erreur obtenue lorsqu'on approche l'intégrale par la méthode de Simpson sur le même intervalle ?

Solution

Etant donné que la fonction à intégrer est décrite par un polynôme de degré 2, on s'attend à ce que l'erreur soit nulle car la méthode de Simpson donne un résultat exact pour tous les polynômes de degré inférieur ou égal à 3.

Algorithmique

On donne ci-dessous l'implémentation d'un algorithme de tri, à laquelle nous avons rajouté l'instruction
`print(f"{i}e element: {L}")`.

```
def mafonction(L):
    n = len(L)
    for i in range(n-1,-1,-1):
        j = i
        while j < n-1 and L[j] > L[j+1]:
            L[j], L[j+1] = L[j+1], L[j]
            j += 1

        print(f"{i}e element: {L}")
```

a) Qu'affichent les instructions suivantes ?

```
L = [1, 3, 0, -6, 100, -1]
mafonction(L)
```

Solution

5e element : [1, 3, 0, -6, 100, -1]
4e element : [1, 3, 0, -6, -1, 100]
3e element : [1, 3, 0, -6, -1, 100]
2e element : [1, 3, -6, -1, 0, 100]
1e element : [1, -6, -1, 0, 3, 100]
0e element : [-6, -1, 0, 1, 3, 100]

b) Quel algorithme vu en cours fonctionne selon le même principe que celui-ci ?
Solution

Le tri par insertion, mais en triant depuis la fin de la liste.

c) On dénote par $T(n)$ le temps de parcours de l'algorithme **`mafonction()`** lorsqu'il prend en entrée une liste de taille n, dans le pire des cas. Donner l'ordre de croissance de $T(n)$ en notation $\Theta(\cdot)$, en justifiant brièvement votre réponse.

Solution

Le pire des cas se produit lorsque à l'itération `i` de la boucle `for`, la boucle `while` itère `i` fois, ce qui correspond à un temps de parcours de $\Theta(n^2)$.

d) Quelle propriété partagent les instances pour lequel le temps de parcours est minimal ? Donner l'ordre de croissance du temps de parcours **dans le meilleur des cas** en notation $\Theta(\cdot)$, en justifiant brièvement votre réponse.

Solution

Le meilleur des cas se produit lorsque la boucle `while` itère zéro fois à chaque itération de la boucle `for`, c'est lorsque la liste est déjà triée. Dans ce cas, chaque itération de la boucle `for` prend un temps constant et on a un temps de parcours qui est $\Theta(n)$.