

TRAVAIL D'ETE

-

INFORMATIQUE

TP11-A - Résoudre une équation différentielle

I. Euler scalaire

I.1. Euler classique

Question 1. Rappeler le principe de la méthode d'Euler.

Question 2. On étudie l'équation différentielle $y + y' = 1$. Écrire un programme qui trace la solution approchée par la méthode d'Euler sur l'intervalle $[0; 3]$ avec la condition initiale $y(0) = 0$. Tracer sur le même graphe la courbe de la solution analytique.

I.2. Euler rétrograde

Question 3. Compléter le code précédent pour obtenir désormais la solution sur l'intervalle $[-1; 3]$.

II. Euler vectoriel: oscillateur linéaire

L'évolution dynamique d'un système linéaire de type masse – ressort - amortisseur est régie par l'équation différentielle suivante :

$$z''(t) + 2\lambda z'(t) + z(t) = 0 \quad \text{avec } t \in [t_{\min}, t_{\max}] \quad (*)$$

Question 4. Montrer que (*) est équivalente à un système différentiel d'ordre 1 de la forme :

$$\forall t \in J \quad \begin{cases} z_1'(t) = z_2(t) \\ z_2' = -2\lambda z_2(t) - z_1(t) \end{cases} \quad (**)$$

Question 5. Quel est le lien entre les fonctions z_1, z_2 et la fonction z ?

On se propose de résoudre ce système différentiel par la méthode d'Euler. L'intervalle d'étude $J = [t_{\min}; t_{\max}]$ est discrétisé en n sous-intervalles.

En notant $\delta_t = (t_{\max} - t_{\min})/n$, on pose :

$$\forall i \in \llbracket 0, n \rrbracket \quad t_i = t_{\min} + i \times \delta_t$$

On définit le tableau à $(n + 1)$ éléments :

$$Z = [[z_1(t_0), z_2(t_0)], \dots, [z_1(t_i), z_2(t_i)], \dots, [z_1(t_n), z_2(t_n)]]$$

Où, pour $k = 1$ ou 2 , $z_{k,i}$ désigne la valeur approchée de $z_k(t_i)$ par la méthode d'Euler.

Si z désigne la valeur prise par le couple $[z_1; z_2]$ pour une certaine valeur t , le second membre de (**) peut être défini par une fonction f :

```

10 def f(z,t):
11     z_prime_1=
12     z_prime_2=
13     return(z_prime_1,z_prime_2)

```

Question 6. Compléter la définition de cette fonction en complétant les expressions de z_prime_1 et z_prime_2 .

Question 7. Définir une fonction $euler(z, f, t, delta_t)$ qui, pour une valeur $i \in \llbracket 1, n \rrbracket$, reçoit la valeur du couple $z = [z_1(t_i), z_2(t_i)]$, la fonction f , la valeur $t = t_i$, le nombre $delta_t$ et renvoie la valeur du couple $[z_1(t_{i+1}), z_2(t_{i+1})]$ calculée en mettant en œuvre la méthode d'Euler.

Question 8. Compléter le code ci-dessous pour qu'il construise le tableau des valeurs de Z et trace, dans une même fenêtre graphique mais dans deux zones graphiques distinctes (fonction subplot), les courbes représentant les variations de z_1 en fonction de t et de z_2 en fonction de t .

```
7 import numpy as np
8
9 # fonction second membre
10 def f(Z,t):
11     a=Z[1]
12     b=-2*Lambda*Z[1]-Z[0]
13     v=[a,b]
14     return(v)
15
16 # fonction Euler vectorielle
17 def euler(z,f,t,delta_t):
18     c=z+delta_t*f(z,t)
19     return(c)
20
21 Lambda=0.1 # coefficient lambda
22 z1_0=1 # conditions initiales
23 z2_0=0
24 t_min=0 # intervalle d'étude
25 t_max=25
26 n=250 # nombre de points
27 delat_t=(t_max-t_min)/n # pas de discrétisation
28 Z=np.zeros((n,2)) # initialisation tableau solutions
29 T=np.linspace(t_min, t_max,n)
30 Z[0,:]=[z1_0,z2_0] # valeurs initiales de Z
31 t=t_min #valeur initiale de t
32
33 # à compléter
```

TP11-B - Résolution approchée d'une équation

Le but de ce TP est de trouver une solution approchée de l'équation $f(x) = x^5 + x - 1 = 0$ avec une précision $\varepsilon = 10^{-4}$ en utilisant les méthodes de Newton et de dichotomie.

I. Méthode de Newton

Question 1

Rappeler le principe de la méthode de résolution de Newton.

Question 2

Tracer la courbe de la fonction $f(x)$ sur l'intervalle $[-10,10]$ en tapant le code suivant:

```
9 import numpy as np
10 import pylab as pl
11
12 def f(z):
13     w=z**5+z-1
14     return(w)
15
16 x=np.linspace(-10,10,50)
17 y=f(x)
18 pl.plot(x,y)
```

Question 3

Créer une fonction permettant de résoudre l'équation $f(x) = 0$ et donner la solution à la précision demandée.

Question 4

Modifier la fonction afin de compter le nombre d'itérations nécessaires à la résolution et afficher ce nombre.

II. Méthode de dichotomie

Question 5

Rappeler le principe de la méthode de résolution de dichotomie..

Question 6

Créer une fonction permettant de résoudre l'équation $f(x) = 0$ et donner la solution à la précision demandée.

Question 7

Modifier la fonction afin de compter le nombre d'itérations nécessaires à la résolution et afficher ce nombre.

III. Comparaison des méthodes

Question 8

Conclure sur le choix de la méthode.

TP11B – Calcul d'intégrales

Remarque:

On peut passer une fonction f en paramètre à une autre fonction F : cela ne pose aucun problème! Voici par exemple la fonction `taux` qui calcule le taux d'accroissement d'une fonction f entre deux points a et b . On pourra ensuite lui passer comme paramètre n'importe quelle fonction f qu'on aura définie ailleurs.

Exemple:

```
def taux(f,a,b):
    x=(f(b)-f(a))/(b-a)
    return x

def f(x):
    return (x**2+1)

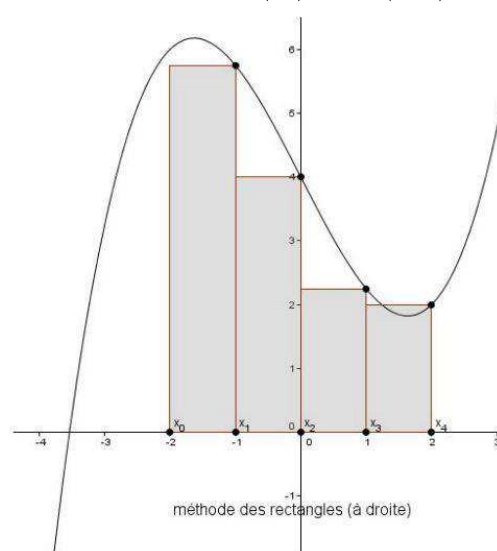
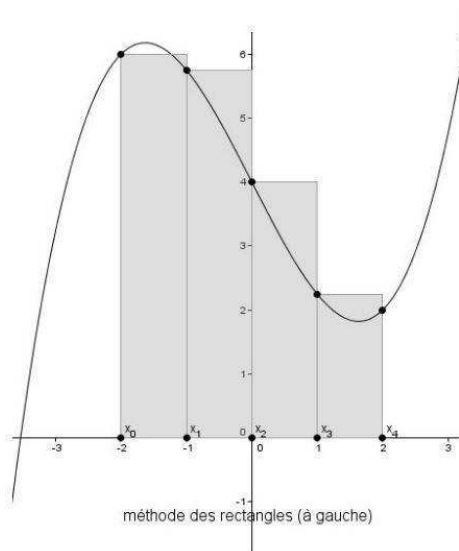
print(taux(f,0,1))
```

Soient $a, b \in \mathbb{R}$, $a < b$ et f une fonction continue sur $[a;b]$.

Vous avez vu en cours de mathématiques que la valeur d'une intégrale $\int_a^b f(x)dx$ peut être approchée par des sommes d'aires de rectangles.

En effet, si on découpe l'intervalle $[a;b]$ en n sous-intervalles de même longueur (on notera $x_0 = a$, $x_n = b$ et $x_1 < x_2 < \dots < x_{n-1}$ les autres extrémités), on peut approcher

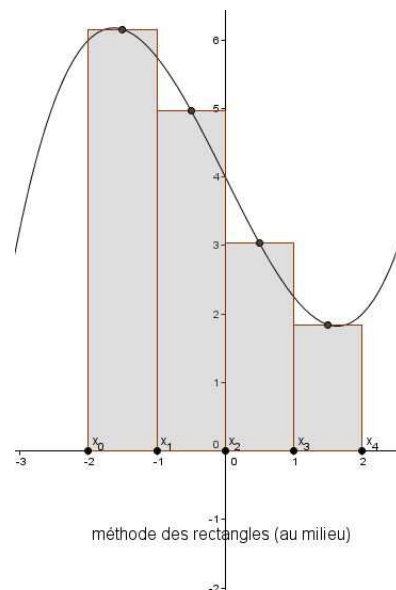
$\int_{x_i}^{x_{i+1}} f(x)dx$ par l'aire du rectangle ayant pour base le segment $[x_i; x_{i+1}]$ et pour hauteur la valeur de f sur l'extrémité de gauche/droite de l'intervalle (i.e. $f(x_i)$ ou $f(x_{i+1})$).

**Question 1**

Programmer une fonction qui prend en entrée a , b , n et f et calcule une valeur approchée de

$\int_a^b f(x)dx$ par la méthode des rectangles à gauche (en découpant l'intervalle $[a,b]$ en n morceaux).

On peut aussi utiliser la méthode des rectangles en prenant comme hauteur de chaque rectangle la valeur de f au point du milieu du segment considéré.

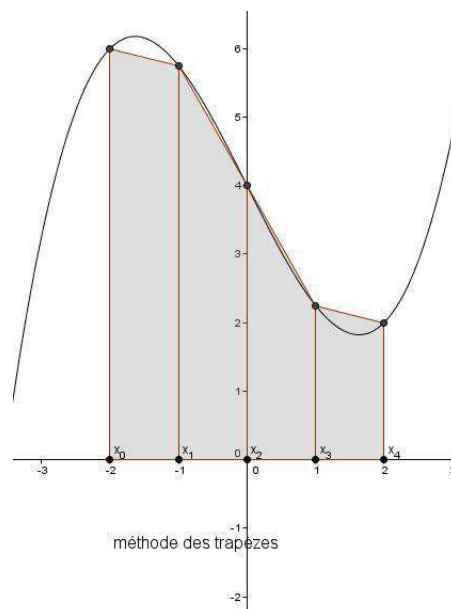


Question 2

Programmer la variante de l'algorithme des rectangles décrite ci-dessus.

Enfin, on peut calculer l'aire du trapèze ayant pour sommets :

$(x_i, 0)$, $(x_{i+1}, 0)$, $(x_{i+1}, f(x_{i+1}))$, et $(x_i, f(x_i))$.



Question 3

Programmer une fonction qui prend en entrée a, b, n et f et calcule une valeur approchée de

$\int_a^b f(x) dx$ par la méthode des trapèzes (en découpant l'intervalle [a,b] en n morceaux).

Question 4

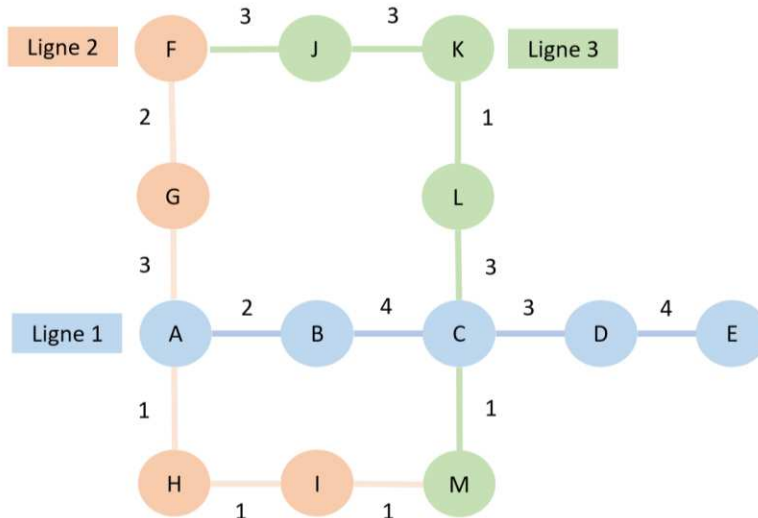
Vous testerez chacune des trois fonctions programmées pour calculer les valeurs approchées des intégrales suivantes (en découpant chaque intervalle en dix morceaux) :

$$\int_2^4 e^x dx \quad \int_{-2}^2 \left(\frac{1}{4} x^3 - 2x + 4 \right) dx \quad \int_5^8 \ln(x) dx \quad \int_0^{1/2} \sin(x) dx$$

TP14 - Graphes

I. Recherche du chemin le plus court - Algorithme de Dijkstra

Voici un plan de métro :



Le réseau est composé de trois lignes. Les stations sont nommées par des lettres afin de faciliter l'étude. Les temps de parcours en minutes sont indiqués sur les branches de chaque ligne entre chaque station. Nous allons programmer l'algorithme de Dijkstra afin de déterminer le plus court chemin sur ce réseau pour aller de la gare de départ F à la gare d'arrivée C.

Question 1

Ecrire un programme permettant de déterminer le plus court chemin de F à C en exploitant la description ci-dessous.

L'algorithme se déroule ainsi :

- Initialisation : Création des données initiales

- Création de **Depart**, chaîne de caractères contenant la station de départ
- Création de **Arrivee**, chaîne de caractères contenant la station d'arrivée
- Création de **Infini**, entier correspondant à la distance infinie définie dans le cours
- Création de la liste **Stations**, liste de chaînes de caractères des noms des stations
- Création de **Nb**, entier correspondant aux nombres de stations
- Création du tableau **Graphe** correspondant à la matrice d'adjacence du réseau
- Initialisation de la liste **Distances** avec **Nb** valeurs **Infini** (elle contient les distances minimales « Depart → Stations[i] »)
- Initialisation à 0 de la distance de la station **Depart** dans la liste **Distances**. On obtiendra l'indice de **Depart** dans **Stations** avec l'instruction : `ind_Depart=Stations.index(Depart)`

- Initialisation de la liste **Reste** contenant toutes les stations de **Stations** (copie)
 - Initialisation de la liste **Provenances** contenant **Nb** o par exemple (elle contient les « Provenances » d'une station. $P = \text{Provenance}[i]$ sera la station de provenance de la station $S = \text{Stations}[i]$ qui a permis d'atteindre S avec le plus court chemin depuis le départ)
 - Initialisation de **S**, station courante traitée par l'algorithme, et de son indice **ind_S**
 - **Itérations** : Tant que **S** n'est pas l'arrivée, qu'il reste des stations dans **Reste** et que $\text{Distance}[\text{ind_S}]$ est différente de l'infini (cette condition permet de gagner du temps si **Arrivee** n'est pas accessible depuis **Depart**)
 - **S** ← Station parmi **Reste** ayant la distance minimum dans **Distances** (la première si exæquo)
 - Mise à jour de **Reste** (retirer **S**)
 - **Voisins** ← Liste des stations de **Reste** voisines de **S** (utilisation de Graphe)
 - Traitement des voisins : Pour chaque voisin V, si la distance $\text{Depart} \rightarrow S \rightarrow V < \text{Depart} \rightarrow V$ actuellement stockée dans **Distances** :
 - Mise à jour de **Distances** avec cette nouvelle distance
 - Mise à jour de **Provenances** afin d'indiquer que S est le prédécesseur de V
- Nous obtenons la distance la plus courte du départ à chaque station dans **Distances** et la liste des prédécesseurs de chaque station dans **Provenances**.*
- **Traitement final** :
 - Détermination de la distance à parcourir pour atteindre **Arrivee** depuis **Depart** dans **Distances**
 - Si la solution existe (distance non infinie), remontée du chemin le plus court entre **Depart** et **Arrivee** en exploitant **Provenances** et inversion du résultat