Construction approchée de la fonction exponentielle
TP sur python

On a vu dans le cours que l'existence de la fonction exponentielle est admise. On peut cependant illustrer ceci par une construction approchée utilisant un algorithme.

1 Algorithme de construction approchée

Principe : On sait deux choses de la fonction qu'on cherche à construire : $\exp'(x) = \exp(x)$ et $\exp(0) = 1$. Or, on sait aussi (voir le cours sur les dérivées) que si f est dérivable en a, et h suffisamment petit, alors $f(a+h) \simeq f(a) + f'(a) \times h$.

Vu que $\exp'(x) = \exp(x)$, on en déduit que si h est suffisamment petit, alors $\exp(x+h) \simeq \exp(x)+\exp(x)\times h$.

On peut donc, à partir de $\exp(0) = 1$, déterminer une valeur approchée de $\exp(h)$, puis, à partir de celle-ci, une valeur approchée de $\exp(2h)$, puis, en itérant le processus, de $\exp(3h)$, $\exp(4h)$, etc. Cette méthode est due essentiellement au grand mathématicien Euler.

Mise en œuvre : Soit un intervalle $[x_{\min}\,; x_{\max}]$ dans lequel varie x avec $x_{\min} < 0$ et $x_{\max} > 0$.

On construit une liste de valeurs x_k variant dans cet intervalle. On passe d'une valeur x_k de la liste à la suivante en ajoutant une constante, appelée le pas de la subdivision, par exemple 0,1. On construit une liste de valeurs y_k définies par la méthode d'Euler. Les yk constituent, si le pas est suffisamment petit, une approximation correcte de l'image des x_k par la fonction exponentielle.

Il y une dernière particularité : La valeur initiale de l'algorithme est : $x_0 = 0$ (et $y_0 = 1$). Donc, on va construire deux listes : une contenant les x_k positifs, et l'autre les x_k négatif. On peut les appeler par exemple `liste_x_d` (à droite) pour la première et `liste_x_g` (à gauche) pour la seconde. C'est pareil pour les listes des y_k

Première liste : $x_0 = 0$; $y_0 = 1$ et h est égal au pas de la subdivision. (On va « de gauche à droite »).

$$\begin{cases} x_{k+1} = x_k + h \\ y_{k+1} = y_k + y_k \times h \end{cases}$$

On effectue ce calcul tant que x_k est inférieur à $x_{\max}$. L'algorithme comporte donc une boucle « Tant que ».

Deuxième liste : $x_0 = 0$; $y_0 = 1$ et h est égal à l'opposé du pas de la subdivision. (On va à l'envers, « de droite à gauche »). On effectue ce calcul tant que x_k est supérieur à $x_{\max}$.

On construit une liste, puis l'autre, et il faut penser à réinitialiser à x_0 et y_0 entre les deux.

Voici notre algorithme, avec, par exemple, $x_{\min} = -4$ et $x_{\max} = 4$:

```
Initialisation :
x_max=4 ; x_min=-4
x=0 ; y=1
copier x dans liste_x_droite ; copier y dans liste_y_droite
h=pas
Tant que x<x_max :
    x=x+h
    y=y+y*h
    copier x à la fin de liste_x_droite
    copier y à la fin de liste_y_droite
h=-pas
Tant que x>x_max :
    x=x+h
    y=y+y*h
    copier x à la fin de liste_x_gauche
    copier y à la fin de liste_y_gauche
```

Il reste deux choses à faire : inverser l'ordre des éléments de `liste_x_gauche` et `liste_y_gauche`, pour les remettre du plus petit au plus grand, puis mettre bout à bout, d'une part les deux listes `liste_x_gauche` et `liste_x_droite` et d'autre part `liste_y_gauche` et `liste_y_droite`.

Ensuite, on peut afficher les valeurs, mais surtout, on peut tracer la courbe.

2 Codage en python

Le calcul des deux listes de valeurs x_k et y_k est réalisé par la fonction `definition_d_une_courbe` qui prend en paramètre le pas de la subdivision et les bornes $x_{\min}$ $x_{\min}$ que l'on souhaite, et renvoie les deux listes comme résultat.

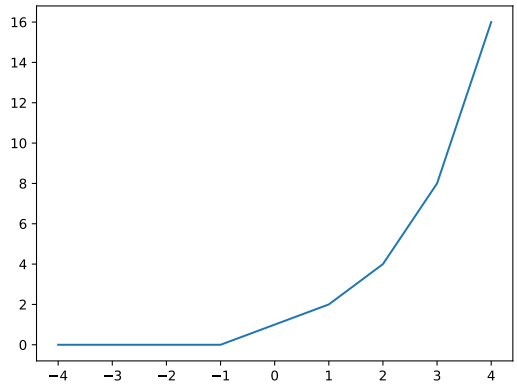
```
def definition_d_une_courbe(pas,x_min,x_max):
    x=0 #Initialisation des variables
    y=1
    liste_x_d=[x]
    liste_y_d=[y]
    liste_x_g=[x]
    liste_y_g=[y]
#Calcul des valeurs successives prises par x_k, y_k à droite de x_0
    h=pas
    while x<x_max:
        x=x+h       # calcul de x_k+1
        y=y+y*h     # calcul de y_k+1 d'après la méthode d'Euler
        liste_x_d.append(x) # copie en fin de liste
        liste_y_d.append(y)
    x=0             #on réinitialise à x_0 et y_0
    y=1
#Calcul des valeurs successives prises par x_k, y_k à gauche de x_0
    h=-pas
    while x>x_min:
        x=x+h
        y=y+y*h
        liste_x_g.append(x)
        liste_y_g.append(y)
# On remet les valeurs de x et y à gauche dans l'ordre croissant
    liste_x_g.reverse()
    liste_y_g.reverse()
# On met bout à bout les listes  x_g et x_d, puis y_g et y_d
    liste_x=liste_x_g+liste_x_d # le + signifie ici la concaténation des 2 listes
    liste_y=liste_y_g+liste_y_d
    return liste_x,liste_y
```

Taper ce code et vérifier qu'il fonctionne en tapant, par exemple `definition_d_une_courbe(1,-4,4)` dans la console. On obtient : ([-4, -3, -2, -1, 0, 0, 1, 2, 3, 4], [0, 0, 0, 0, 1, 1, 2, 4, 8, 16])

À présent, il reste à placer les points de coordonnées $(x_k\,;y_k)$ dans un repère et les relier. On utilise pour cela le module `pyplot` de la bibliothèque graphique `matplotlib`.

```
import matplotlib.pyplot as plt
def trace_d_une_courbe(pas,x_min,x_max):
    x,y=definition_d_une_courbe(pas,x_min,x_max)
    plt.plot(x,y)
    plt.show()
```

On obtient :



Pas mal, mais pas très informatif. L'étape suivante va consister à tracer plusieurs courbes, utilisant des pas différents, pour savoir si les courbes se rapprochent d'une position limite, quand le pas devient de plus en plus petit.

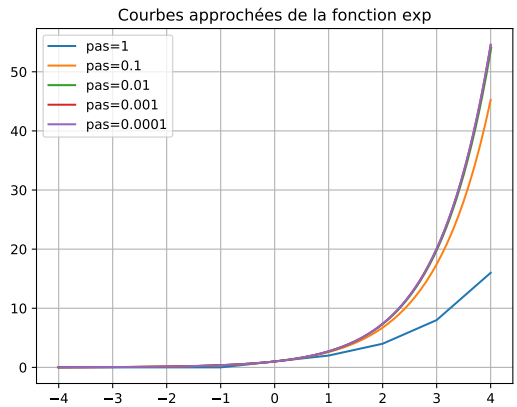
Pour compléter le graphique, on ajoutera une grille et un titre.

La fonction `trace_des_courbes(liste_pas,x_min,x_max)` prend en paramètre une liste contenant plusieurs valeurs du pas, et comme d'habitude, les bornes $x_{\min}$ $x_{\min}$.

```
def trace_des_courbes(liste_pas,x_min,x_max):
    for pas in liste_pas:
        x,y=definition_d_une_courbe(pas,x_min,x_max)
        plt.plot(x,y,label="pas={}".format(pas)) # label : étiquette de la courbe
    plt.grid() # affiche la grille
    plt.title("Courbes approchées de la fonction exp")
    plt.legend() # affiche les étiquettes
    plt.show()
```

Essayons en tapant : `trace_des_courbes([1,0.1,0.01,0.001,0.0001],-4,4)`

On obtient :



On vérifie que lorsque le pas choisi est assez petit (ici : inférieur ou égal au centième), les courbes ont tendance à se confondre. On peut donc admettre que lorsque le pas de la subdivision tend vers 0, la courbe tend vers une courbe limite, qui est celle de la fonction exponentielle.

On peut le vérifier en traçant la courbe de la « vraie » fonction exponentielle.

Il suffit de modifier le code précédent :

```python
from numpy import exp # numpy est une bibliothèque de calcul scientifique

def trace_des_courbes(liste_pas,x_min,x_max):
    for pas in liste_pas:
        x,y=definition_d_une_courbe(pas,x_min,x_max)
        plt.plot(x,y,label="pas={}".format(pas)) # label : étiquette de la courbe
    plt.plot(x,exp(x), label="exp")  # tracé de la courbe de la fonction exponentielle
    plt.grid() # affiche la grille
    plt.title("Courbes approchées de la fonction exp")
    plt.legend() # affiche les étiquettes
    plt.show()
```