Representation Graphique en Langage Python

L'objectif de ce complément est d'aborder, à travers des cas concrets rencontrés aux cours des activités expérimentales et numériques du programme, la majeure partie des instructions du langage de programmation Python présentes dans le point numérique du manuel.

Introduction

Dans de très nombreuses situations en Physique-Chimie, comme en mécanique pour représenter les positions successives d'un mobile assimilé à un point lors de son mouvement ou en électricité pour tracer la caractéristique tension-courant d'un dipôle, il est utile de tracer la représentation graphique *y=f(x)* d'une grandeur *y* en fonction d'une grandeur *x*.

En langage de programmation Python, une représentation graphique utilise les fonctions et instructions du module `pyplot` de la bibliothèque `matplotlib` habituellement importé sous le préfixe `plt`.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt       # Importe le module pyplot en plt
```

La représentation graphique de la courbe d'équation *y=f(x)*, se fait grâce à l'instruction `plt.plot(x,y,paramètres)` où `x` et `y` sont des objets contenant les abscisses et les ordonnées des points à représenter et `paramètres` précise l'aspect des points et/ou de la courbe.

Les objets `x` et `y` peuvent être :
- soit des listes de nombres de type «liste» ;
- soit des tableaux de nombres comportant une seule ligne, de type «tableau» (ou «array») de la bibliothèque `NumPy`.

Quel que soit le type des objets `x` et `y`, la seule contrainte de l'instruction `plt.plot(x,y)` est qu'ils aient le même nombre de valeurs, c'est à dire qu'il y ait autant de valeurs pour les abscisses que de valeurs pour les ordonnées.

Exemple 1 : Lancer franc au Basket

Les coordonnées *(x,y)* des positions successives du ballon lors d'un lancer franc au basket ont été obtenues par pointage de la position du centre du ballon sur 12 images de la vidéo du lancer, grâce à un logiciel dédié :
- l'intervalle de temps entre chaque image vaut Δt = 66,62 ms ;
- l'origine des dates est choisie à l'image immédiatement après que le ballon a quitté les mains du joueur ;
- l'origine des axes est choisie à la position du ballon à l'origine des dates.

1. Représentation des positions successives du ballon

1.1. Définition des listes `x` et `y` des coordonnées des points

Les dates des positions et les coordonnées des points sont définies dans 3 objets `t`, `x` et `y` de type «liste» :
- les valeurs sont rangées dans l'ordre entre crochets `[]` ;
- le point `.` est le séparateur décimal ;
- la virgule `,` permet de séparer les valeurs.

```
t = [0.00,0.066,0.133,0.199,0.266,0.333,0.399,0.466,0.533,0.599,0.666,0.733]
x = [0.00,0.28,0.55,0.80,1.05,1.31,1.56,1.85,2.11,2.35,2.61,2.87]
y = [0.00,0.36,0.69,0.98,1.21,1.40,1.55,1.67,1.75,1.79,1.77,1.71]
```

1.2. Définition et affichage de la figure représentant les points

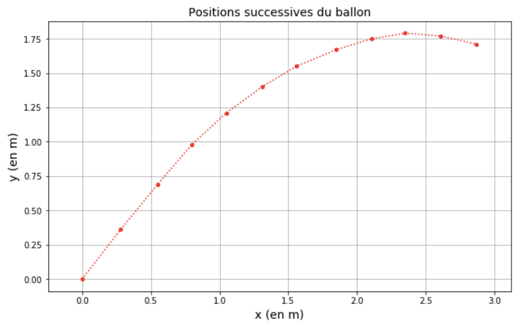
La figure est initialisée par l'instruction `plt.figure()` qui permet également de définir sa taille (largeur,hauteur).
Si la taille de l'image et les échelles sur les axes ne sont pas précisées, l'affichage s'adapte automatiquement.

Les `paramètres` de l'habillage des points les plus courants sont :
- couleurs : `'r'` rouge, `'b'` bleu, `'c'` cyan, `'g'` vert, `'k'` noir ;
- formes : `'o'` point, `'+'` plus, `'x'` croix ;
- ligne entre les points : `'-'` ligne continue, `'--'` pointillés, `':'` petits points.

```
plt.figure('Lancer franc',figsize=(10,6))    # Initialise et nomme la figure
plt.title('Positions successives du ballon',fontsize = 14)# Titre du graphe
plt.xlabel('x (en m)',fontsize = 14)         # Label de l'axe des abscisses
plt.ylabel('y (en m)',fontsize = 14)         # Label de l'axe des ordonnées
plt.axis('equal')                            # Repère orthonormé
plt.grid()                                   # Affiche une grille

plt.plot(x,y,'or:',ms=4)# Nuage de points de coordonnées dans x et dans y :
                        # 'o' points 'r' rouges de taille (ms=markersize) 4,
                        # ':'reliés par des petits points

plt.show()                                   # Affiche la figure
```



2. Représentation des vecteurs vitesse

2.1. Définition des listes `Vx` et `Vy` des coordonnées des vecteurs vitesse

Le vecteur vitesse $\vec{v}_i$ au point M_i est assimilable au vecteur vitesse moyenne entre les deux positions successives M_i et M_{i+1} séparées de l'intervalle de temps $\Delta t = t_{i+1} - t_i$.

$$\vec{v}_i = \frac{\overrightarrow{M_i M_{i+1}}}{t_{i+1} - t_i}$$

Les coordonnées $v_{x\ i}$ et $v_{y\ i}$ du vecteur vitesse $\vec{v}_i$ s'écrivent donc : $v_{x\ i} = \dfrac{x_{i+1} - x_i}{t_{i+1} - t_i}$ et $v_{y\ i} = \dfrac{y_{i+1} - y_i}{t_{i+1} - t_i}$.

Les listes `Vx` et `Vy` des coordonnées des vecteurs vitesse sont calculées à partir des valeurs contenues dans les listes `t`, `x` et `y`.

Pour appeler les valeurs des listes `t`, `x` et `y` dans les calculs, on fait appel aux propriétés suivantes :
- les valeurs rangées dans une liste `L` sont indexées par leur position `i` dans la liste ;
- l'indice de la valeur occupant la première position dans la liste est `0` ;
- l'instruction `L[i]` permet d'appeler la valeur rangée à la position d'indice `i`.

Remarque :
Les valeurs des listes `t`, `x` et `y` portent donc les indices allant de `0` pour la première position à `11` pour la douzième position.

Le caractère répétitif du calcul des coordonnées des vecteurs vitesse pour chaque position `i` de la première à l'avant dernière position impose d'utiliser la boucle `for` que l'on peut par exemple intégrer dans la création de liste "en compréhension".
Cette méthode de création de liste permet de définir une liste sur une seule ligne entre crochets.

```
Vx =[(x[i+1]-x[i])/(t[i+1]-t[i]) for i in range(len(t)-1)]
Vy =[(y[i+1]-y[i])/(t[i+1]-t[i]) for i in range(len(t)-1)]
```

Les formules du calcul des coordonnées du vecteur vitesse sont appliquées de la première position d'indice `0` à l'avant dernière position d'indice `10` grâce à la fonction `range(len(t)-1)`, en effet :
- `range(n)` génère la liste des premiers entiers de `0` à `n-1` ;
- `len(t)` renvoie le nombre d'éléments de la liste `t`.

Donc : `range(len(t)-1)` renvoie les entiers de `0` à `(len(t)-1)-1 = (12-1)-1` soit `10`.

Remarque :
Il est possible d'utiliser une autre syntaxe de création des listes `Vx` et `Vy` à l'aide de l'instruction `list.append(valeur)` qui insère `valeur` en dernière position de la liste `list`.

```
Vx_bis,Vy_bis = [],[] # Création de deux listes vides
for i in range(len(t)-1):
    Vx_bis.append((x[i+1]-x[i])/(t[i+1]-t[i]))
    Vy_bis.append((y[i+1]-y[i])/(t[i+1]-t[i]))
```

2.2. Affichage du vecteur vitesse toutes les deux positions

L'affichage d'une flèche se fait grâce à l'instruction `plt.arrow(x,y,dx,dy,paramètres)` où :
- `x,y` sont les coordonnées du point du pied de la flèche ;
- `dx,dy` sont les projections orientées de la flèche sur les axes des abscisses et des ordonnées ;
- `paramètres` permet de préciser l'aspect et les caractéristiques de la flèche.

Afficher le vecteur vitesse $\vec{v}_i$ au point M_i revient à tracer une flèche au point de coordonnées `x[i],y[i]` telle que `dx=Vx[i]` et `dy=Vy[i]`. Cependant, pour visualiser correctement ces vecteurs, il est nécessaire de multiplier leurs coordonnées par un facteur d'échelle `e` à fixer selon la situation : `dx=e*Vx[i]` et `dy=e*Vy[i]`.

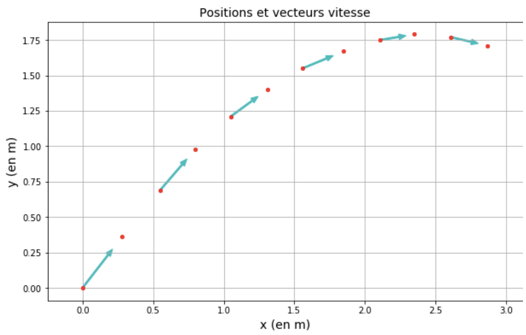
Pour représenter le vecteur vitesse toutes les `2` positions, on parcourt grâce à une boucle `for` la liste d'entiers définie par la fonction `range(n,m,p)` de `n` inclus à `m` exclu par pas de `p` en prenant pour `n` l'indice de la première position, pour `m` l'indice de la dernière position dont on a calculé les coordonnées du vecteur vitesse et en fixant la valeur de `p` à `2`.

```
plt.figure('Lancer franc',figsize=(10,6))   # Initialise et nomme la figure
plt.title('Positions et vecteurs vitesse',fontsize = 14)  # Titre du graphe
plt.xlabel('x (en m)',fontsize = 14)        # Label de l'axe des abscisses
plt.ylabel('y (en m)',fontsize = 14)        # Label de l'axe des ordonnées
plt.axis('equal')                           # Repère orthonormé
plt.grid()                                  # Affiche une grille

plt.plot(x,y,'or',ms=4)    # Nuage de points de coordonnées dans x et dans y

for i in range(0,len(t)-1,2):
    plt.arrow(x[i],y[i],0.05*Vx[i],0.05*Vy[i],width=0.01,color='c',
              length_includes_head="true",head_length=0.05, head_width=0.04)

plt.show()                                  # Affiche la figure
```



EXEMPLE 2 : CARACTERISTIQUE TENSION-COURANT D'UN DIPOLE

Les valeurs de la tension U (en V) et de l'intensité I (en mA) aux bornes d'un dipôle sont relevées expérimentalement et on souhaite représenter et modéliser la caractéristique tension-courant $U=f(I)$ du dipôle.

1. Représentation de la caractéristique tension-courant $U=f(I)$

1.1. Définition des tableaux `U` et `I` des données expérimentales

Pour ranger une collection de nombres les uns à la suite des autres, on peut utiliser des tableaux à une dimension, c'est-à-dire à une seule ligne, de type «tableau» («array» en anglais) de la bibliothèque `NumPy` habituellement importée sous le préfixe `np`.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt      # Importe le module pyplot en plt
import numpy as np                   # Importe la bibliothèque numpy en np
```

L'instruction `np.array(liste)` convertit la liste de valeurs `liste` définie entre crochets avec la même syntaxe que dans l'exemple 1 en un tableau de nombres à une ligne.

Les valeurs expérimentales de la tension U (en V) et de l'intensité I (en mA) sont rangées dans l'ordre dans 2 tableaux `U` et `I`.

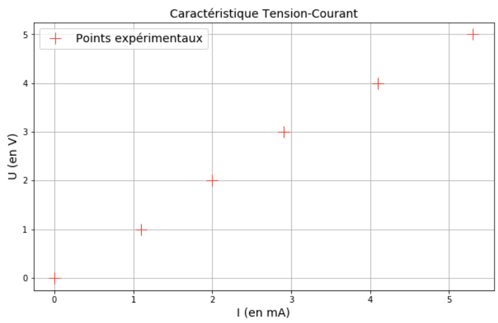
```
U = np.array([0.0,1.0,2.0,3.0,4.0,5.0])     # U (en V) expérimental
I = np.array([0.0,1.1,2.0,2.9,4.1,5.3])     # I (en mA) expérimental
```

1.2. Définition et affichage de la figure représentant les points expérimentaux

```
plt.figure('Etude d\'un dipôle',figsize=(10,6))# Initialise la figure
plt.title('Caractéristique Tension-Courant',fontsize = 14)# Titre du graphe
plt.xlabel('I (en mA)',fontsize = 14)       # Label de l'axe des abscisses
plt.ylabel('U (en V)', fontsize = 14)       # Label de l'axe des ordonnées

plt.plot(I,U,'r+',ms=14,label='Points expérimentaux') # Points expérimentaux
                                  # d'abscisses dans I et d'ordonnées dans U
                                  # sous forme de points rouges non reliés
                                  # de taille 14, label = nom de la courbe

plt.grid()                                  # Affiche une grille
plt.legend(fontsize=14)                     # Affiche la légende
plt.show()                                  # Affiche les courbes
```



2. Modélisation de la caractéristique

Modéliser le nuage de points consiste à déterminer l'équation mathématique de la courbe qui se rapproche le plus de celle qu'ils tracent. Dans le cas de la caractéristique du dipôle étudié, les points sont alignés sur une droite passant par l'origine.

2.1. Modélisation à l'aide de la fonction `np.polyfit(I,U,1)`

a. Modélisation

La fonction `np.polyfit(x,y,1)` modélise le nuage de points d'abscisses dans `x` et d'ordonnées dans `y` par une droite d'équation $y=ax+b$ et renvoie le tableau : `[a b]`.

Les coefficients de la droite modélisant le nuage de points sont calculés par `np.polyfit(I,U,1)` et sont affectés dans cet ordre aux variables `a` et `b`.

```
a,b = np.polyfit(I,U,1)
```

Remarque :
Si `I` et `U` sont de type «liste» et non de type «tableau», la fonction `np.polyfit(I,U,1)` peut être utilisée de la même façon car les fonctions de la bibliothèque `NumPy` s'appliquent aussi aux objets ressemblant à des tableaux (de type «array_like») comme les listes de nombres.

b. Affichage de la modélisation

Il s'agit d'ajouter sur la figure la courbe d'équation $U_{modelisation}=f(I)$ où les valeurs de $U_{modelisation}$ sont définies par la modélisation précédente, soit : $U_{modelisation} = a{\times}I + b$.

Pour calculer les valeurs des ordonnées $U_{modelisation}$, on fait appel à la propriété suivante : les opérations mathématiques usuelles appliquées à un objet de type «tableau» s'appliquent séparément à chaque élément du tableau.

Donc, comme `I` est de type «tableau», sachant que `a` et `b` sont des nombres, l'instruction `a*I+b` renvoie le tableau des résultats du calcul appliqué à chaque valeur du tableau `I`. L'affichage de la modélisation se fait alors grâce à l'instruction `plt.plot(I,a*I+b)` qui trace les points d'abscisses dans `I` et d'ordonnées dans `a*I+b`.

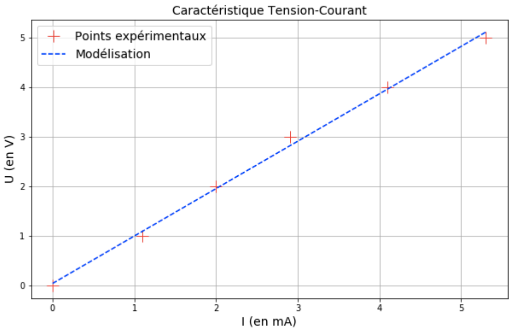
Attention *: Si `I` est de type «liste», les opérations mathématiques usuelles ne s'appliquent pas à chaque valeur de la liste.*

```
plt.figure('Etude d\'un dipôle',figsize=(10,6))# Initialise la figure
plt.title('Caractéristique Tension-Courant',fontsize = 14)# Titre du graphe
plt.xlabel('I (en mA)',fontsize = 14)        # Label de l'axe des abscisses
plt.ylabel('U (en V)', fontsize = 14)        # Label de l'axe des ordonnées

plt.plot(I,U,'r+',ms=14,label='Points expérimentaux') # Points expérimentaux

plt.plot(I,a*I+b,'b--',label='Modélisation')# Points d'abscisses dans I et
                                             # d'ordonnées dans a*I+b en bleu
                                             # reliés par des pointillés

plt.grid()                                   # Affiche une grille
plt.legend(fontsize=14)                      # Affiche la légende
plt.show()                                   # Affiche les courbes
```



c. Equation de la caractéristique

Les valeurs expérimentales de la tension *U* et de l'intensité *I* étant données avec 2 chiffres significatifs, le résultat de la modélisation est donné aussi avec 2 chiffres significatifs.
Les instructions habituelles d'affichage de texte et de formatage des nombres permettent d'afficher l'équation de la caractéristique et la valeur de la résistance.

```
print('\t Modélisation de la caractéristique de la résistance :')
print('\t U (en V) =','{:.2f}'.format(a), 'x I (en mA)')
print('\t La résistance estimée vaut R =','{:.1e}'.format(1000*a),'Ω.')
```

```
         Modélisation de la caractéristique de la résistance :
         U (en V) = 0.96 x I (en mA)
         La résistance estimée vaut R = 9.6e+02 Ω.
```

```
print('a =',a,'\nb =',b)
```

```
a = 0.9582309582309583
b = 0.040540540540540876
```

Remarque :
*Les valeurs de la tension U et de l'intensité I n'étant pas données dans le jeu d'unités **SI**, il faut en tenir compte pour afficher l'équation de la caractéristique en précisant les unités et pour donner la valeur estimée de la résistance en Ω.*

2.2. Modélisation à l'aide de la fonction `linregress(I,U)`

a. Modélisation

La fonction `linregress(x,y)`, importée directement depuis le module `stats` de la bibliothèque `SciPy`, fait l'étude statistique du nuage de points d'abscisses dans `x` et d'ordonnées dans `y` en en calculant la régression linéaire et renvoie dans l'ordre cinq valeurs : la pente, l'ordonnée à l'origine, le coefficient de corrélation, la p-value et l'erreur standard.

Les valeurs de l'étude statistique du nuage de points sont calculées par `linregress(I,U)` et rangées dans l'objet `Modele`, puis les deux premières valeurs de `Modele` sont affectées dans cet ordre aux variables `m` pour la pente et `p` pour l'ordonnée à l'origine.

```
from scipy.stats import linregress
Modele = linregress(I,U)
m,p = Modele[0],Modele[1]
```

b. Affichage de la modélisation

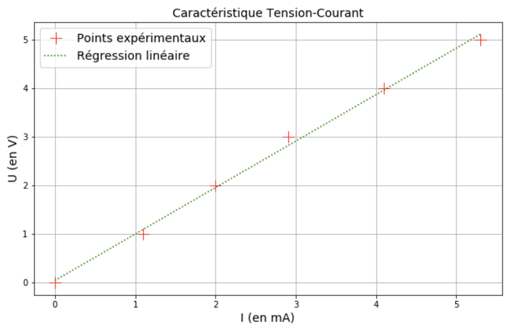
Le principe d'affichage est le même que précédemment grâce à l'instruction `plt.plot(I,m*I+p)` qui trace les points d'abscisses dans `I` et d'ordonnées dans `m*I+p`.

```
plt.figure('Etude d\'un dipôle',figsize=(10,6))# Initialise la figure
plt.title('Caractéristique Tension-Courant',fontsize = 14)# Titre du graphe
plt.xlabel('I (en mA)',fontsize = 14)        # Label de l'axe des abscisses
plt.ylabel('U (en V)', fontsize = 14)        # Label de l'axe des ordonnées

plt.plot(I,U,'r+',ms=14,label='Points expérimentaux') # Points expérimentaux

plt.plot(I,m*I+p,'g:',label='Régression linéaire')# # Points d'abscisses
                                       # dans I et d'ordonnées dans m*I+p
                                       # en vert reliés par des petits points

plt.grid()                                   # Affiche une grille
plt.legend(fontsize=14)                      # Affiche la légende
plt.show()                                   # Affiche les courbes
```



c. Equation de la caractéristique

```
print('\t Modélisation de la caractéristique de la résistance :')
print('\t U (en V) =','{:.2f}'.format(m), 'x I (en mA)')
print('\t La résistance estimée vaut R =','{:.1e}'.format(1000*m),'Ω.')
```

```
        Modélisation de la caractéristique de la résistance :
        U (en V) = 0.96 x I (en mA)
        La résistance estimée vaut R = 9.6e+02 Ω.
```

```
print('m =',m,'\np =',p)
```

```
m = 0.958230958230958
p = 0.04054054054054124
```

Remarque : La comparaison des coefficients `a` *et* `m` *d'une part, et* `b` *et* `p`*, d'autre part, montre que les deux méthodes sont équivalentes pour le cas étudié.*

EXEMPLE 3 : SIMULATION DE LA PROPAGATION D'UNE ONDE SINUSOÏDALE

La simulation de la propagation d'une onde sinusoïdale de période T, de longueur d'onde λ et d'amplitude A consiste à afficher à chaque image la même sinusoïde de longueur d'onde λ et d'amplitude A tout en la décalant dans la direction de propagation de l'onde de la distance dont elle a avancé image après image.

A l'instant de date $t = 0$ s, on suppose que l'équation de la sinusoïde s'écrit : $y_{t=0}(x) = A \times cos\left(\frac{2\pi}{\lambda}x\right)$.

On admet alors que l'équation de la courbe à afficher à l'image de date t est : $y(x,t) = A \times cos\left(\frac{2\pi}{\lambda}x - \frac{2\pi}{T}t\right)$

Remarque :

Sachant que la célérité v de l'onde s'écrit $v = \frac{\lambda}{T}$*, en factorisant* $\frac{2\pi}{\lambda}$*, on a :* $\frac{2\pi}{\lambda}x - \frac{2\pi}{T}t = \frac{2\pi}{\lambda}\left(x - \frac{\lambda}{T}t\right) = \frac{2\pi}{\lambda}(x - vt)$.

Donc la courbe à afficher à l'image de date t a pour équation $y_{t=0}(x - vt)$*, elle correspond à la translation de +vt sur l'axe des abscisses de celle affichée à la date t = 0 s.*

1. Visualisation d'une animation

En langage de programmation Python, il est possible d'animer une figure grâce à la fonction `FuncAnimation` du module `animation` de la bibliothèque `matplotlib` importée directement en insérant en début de programme l'instruction :

```
from matplotlib.animation import FuncAnimation
```

Cette fonction permet d'afficher à intervalles de temps réguliers une série de courbes.
Pour visualiser les animations, il est nécessaire de paramétrer préalablement l'environnement.

1.1. Sous Spyder

Le logiciel doit être paramétré en mode de visualisation *automatique* qui permet de générer la figure dans une fenêtre de visualisation séparée, et non dans la console, à l'exécution du programme. La fenêtre de visualisation est alors interactive.

Pour accéder au mode *automatique*, avant d'exécuter le programme, exécuter **dans la console** la commande « magique » : `%matplotlib auto`.

Le retour à la visualisation de la figure dans la console se fait en exécutant la commande « magique » `%matplotlib inline` dans la console.

Remarque :
Une alternative à l'outil précédent consiste à paramétrer les préférences du logiciel depuis l'onglet Outils > Préférences > Console Ipython > Graphique > Sortie, en sélectionnant `Automatique` *dans le menu déroulant ; le redémarrage du logiciel est alors nécessaire.*

1.2. Dans un calepin IPython

Pour accéder aux outils de figure interactive et aux animations sous **JupyterLab**, il est nécessaire d'introduire en début de programme la commande « magique » : `%matplotlib widget` après avoir installé l'extension disponible sur le lien https://github.com/matplotlib/jupyter-matplotlib lorsque cette extension n'est pas automatiquement accessible sous la version JupyterLab installée.

Remarque : Cette commande magique ***ne fonctionne pas*** *dans l'environnement* ***Colaboratory****.*

Une alternative à l'outil précédent pour visualiser une animation dans un calepin consiste à introduire en début de programme les 2 lignes d'instructions suivantes :

```
from matplotlib import rc
rc('animation', html='jshtml')
```

Il faudra dans ce cas :
- retirer les `#` devant ces 2 lignes et mettre un `#` devant la commande magique `%matplotlib widget` ;
- appeler l'animation par son nom en toute fin de programme (en retirant le `#` devant le nom `animation` de la dernière ligne) ;
- faire preuve de patience (le temps pour la machine de faire tous les calculs) après l'exécution du programme pour visualiser l'animation.

Remarque : Cette deuxième méthode ***fonctionne*** *dans l'environnement* ***Colaboratory****.*

2. Animation d'une sinusoïde pour simuler la propagation d'une onde

Pour utiliser la fonction `FuncAnimation`, il est nécessaire d'initialiser la figure contenant la courbe à animer puis de définir deux fonctions : `init()` pour fixer l'arrière de l'animation et `animate()` pour la mise à jour de la courbe à chaque image.

<table>
<tr><th>Le script et l'animation</th><th>Commentaires</th></tr>
<tr><td><pre>
#!/usr/bin/python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from math import pi

%matplotlib widget
#from matplotlib import rc
#rc('animation', html='jshtml')
</pre></td><td>Importation des éléments requis.</td></tr>
<tr><td><pre>
T = 1.0   # Période en s
Lambda = 1.5   # Longueur d'onde en m
A = 1.0   # Amplitude en m
dt = 0.01   # Intervalle de temps en s
</pre></td><td>On définit la période <code>T</code>, la longueur d'onde <code>Lambda</code>, l'amplitude <code>A</code> de l'onde et l'intervalle de temps <code>dt</code> permettant de décaler la sinusoïde entre chaque image affichée.</td></tr>
<tr><td><pre>
x = np.linspace(0,5,256) # Domaine des abcisses (en m)
</pre></td><td>Le domaine des abscisses est défini à l'aide de l'instruction <code>np.linspace(a,b,n)</code> qui génère un tableau comportant <code>n</code> valeurs régulièrement espacées de <code>a</code> inclus à <code>b</code> inclus.</td></tr>
<tr><td><pre>
# Initialisation de la figure
fig = plt.figure('Propagation d\'une onde sinusoïdale')
plt.xlim(0,5)
plt.ylim(-2*A,2*A)
</pre></td><td>La figure est nommée <code>fig</code> et les bornes des axes sont fixées.</td></tr>
<tr><td><pre>
Initialisation du graphe nommé `courbe` mis à jour au fur et à mesure
courbe, = plt.plot([],[],'r',lw = 1.5) # Graphe sans point
</pre></td><td>Un graphe sans point, nommé <code>courbe</code>, est introduit dans la figure. C'est le graphe qui sera animé et mis à jour au fur et à mesure.

La , est indispensable pour que l'animation soit possible.</td></tr>
<tr><td><pre>
Fonction fixant l'arrière de l'animation présent à chaque image
def init():   # Ne contient pas d'argument
    courbe.set_data([],[]) # Met à jour `courbe` avec des données vides
    return courbe,   # Retourne `courbe` suivi d'une virgule
</pre></td><td>La fonction <code>init()</code> fixant l'arrière de l'animation présent à chaque image est définie à partir du graphe nommé <code>courbe</code> à animer.

La , est indispensable pour que l'animation soit possible.</td></tr>
<tr><td><pre>
Création de la fonction 'animate' appelée à chaque nouvelle image i
def animate(i):   # Un seul argument i associé à l'image i
    t = i*dt
    y = A*np.cos((2*pi/Lambda)*x - (2*pi/T)*t) # Données de l'image i
    courbe.set_data(x, y)# Met à jour `courbe` avec les données de l'image i
    return courbe,   # Retourne `courbe` suivi d'une virgule
</pre></td><td>La fonction <code>animate()</code> appelée à chaque nouvelle image est définie à partir du graphe nommé <code>courbe</code> à animer.

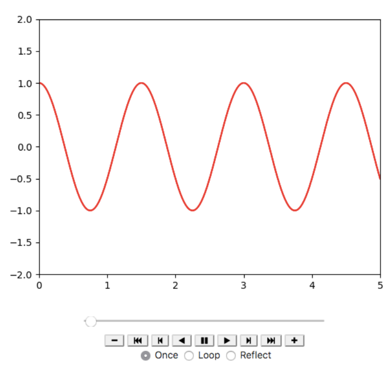
La , est indispensable pour que l'animation soit possible.</td></tr>
<tr><td><pre>
# Animation de la figure affichant animate(i), à chaque image i, pour i
# entier de 0 à 100 avec un délai entre 2 images égal à interval en ms
# blit=True indique que seuls les éléments modifiés sont redessinés
animation = FuncAnimation(fig, animate, init_func=init, frames=101,
                          blit=True, interval=dt*1000, repeat=False)
#animation
</pre></td><td rowspan="2">La fonction <code>FuncAnimation</code> anime la figure <code>fig</code> en affichant <code>animate(i)</code> à chaque image <code>i</code> pour <code>i</code> dans <code>frames</code> (ici : entiers de 0 à 100) avec un délai entre 2 images égal à <code>interval</code> en ms.

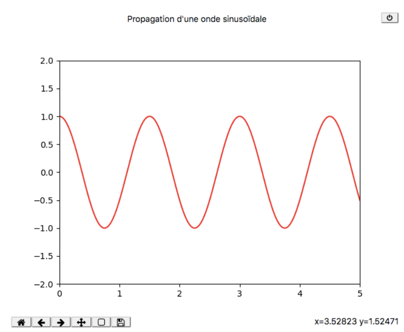
<code>init_func=init</code> fixe l'arrière plan de l'animation avec la fonction <code>init()</code>.

<code>blit=True</code> indique que seuls les éléments modifiés sont redessinés ce qui permet un gain de temps de calculs et une animation plus fluide.

Fenêtres de visualisation :

A gauche : dans les environnements JupyterLab et Colaboratory avec importation de la fonction <code>rc</code>.

A droite : dans les environnements Spyder en mode automatique et JupyterLab avec l'instruction <code>%matplotlib widget</code>.</td></tr>
<tr><td>


</td></tr>
</table>