CLASSE DE PROBLÈMES NUM-1

PROCÉDER À LA MISE EN ŒUVRE D'UNE DÉMARCHE DE RÉSOLUTION NUMÉRIQUE

LIRE ET PRÉSENTER DES DONNÉES EXPÉRIMENTALES

PROBLÉMATIQUE :

Pour traiter des fichiers expérimentaux ou simuler des comportements et des mécanismes, l'informatique s'avère être un puissant outil d'automatisation des tâches.

On propose, en première étape dans ce Tp, de lire un fichier de points et d'en afficher la courbe.

Ce Tp présente également comment tracer dans une même fenêtre graphique, les diagrammes de Bode d'une fonction.

1 Lire un fichier

1.1 Préparation des documents

Dans un dossier `./Documents/Travail/PCSI-2/Tp-Num-1` que vous devrez peut-être construire, créer deux fichiers `data.txt` et `Tp-num-1.py` à partir d'un nouveau fichier `text` (`.txt`).

1.2 Ouvrir un fichier

On considère ici que le fichier est dans le même répertoire que le script.

Soit le fichier texte, appelé `data.txt`. Y placer le contenu suivant :

```
0   1
1   3
4.5 6
7.8 2.5
```

Chaque valeur d'une même ligne est séparée par une tabulation.

Pour ouvrir un fichier on utilise l'instruction `open()`.

```
1 fichier=open("data.txt","r")
2 #on ouvre le fichier data.txt, en mode lecture
```

avec `r` comme *read*.

1.3 Lecture du contenu entier d'un fichier

Pour lire un fichier, on utilise l'instruction `read()`.

```
fichier=open("data.txt","r")
contenu=fichier.read()
print(contenu)
```

Une fois la lecture terminée, on ferme le fichier, pour éliminer les problèmes d'accès à ce même fichier :

```
fichier.close()
```

Le programme suivant affiche : `'0\t1\n1\t3\n4.5\t6\n7.8\t2.5'` lorsqu'on tape `contenu`. Il s'agit d'une seule variable de type `text`. Pour utiliser le contenu du fichier, il convient donc de traiter le résultat précédent pour transformer la variable `text` en tableau de valeurs.

REMARQUE: `\t` correspond à une tabulation et `\n` est le caractère réservé pour *retour à la ligne*. La variable `contenu` est donc cohérente avec les données du fichier.

OBJECTIF : : A partir du fichier texte, on souhaite créer les deux listes suivantes

$t = [0, 1, 4.5, 7.8]$ et $x = [1, 3, 6, 2.5]$ pour tracer x en fonction de t, par exemple.

Tout d'abord, il faut *couper* `contenu` pour enlever les `\t` et les `\n`. Pour cela on utilise l'instruction, `split()` :

```
t1=contenu.split('\n')
```

Cette ligne donne :

```
['0\t1', '1\t3', '4\t5 6', '7.8\t2.5']
```

On voit qu'à chaque fois qu'il y un `\n`, on coupe `contenu`, et on place les éléments dans une liste. Reste à continuer en *coupant* les `\t`.

```
t2=[]
for i in range(len(t1)):
    t2.append(t1[i].split('\t'))
# Ce programme donne :
[['0', '1'], ['1', '3'], ['4', '5 6'], ['7.8', '2.5']]
```

Reste à créer les listes pour tracer les courbes :

```
t=[]
x=[]
for i in range(len(t2)):
    t.append(float(t2[i][0]))
    x.append(float(t2[i][1]))
```

Ce qui donne :

```
>>> t
[0.0, 1.0, 4.5, 7.8]
>>> x
[1.0, 3.0, 6.0, 2.5]
```

L'exploitation est donc maintenant possible.

1.4 Autre méthode

Il est possible d'utiliser d'autres instructions comme `readlines()` :

```
fichier=open("data.txt","r")
contenu=fichier.readlines()
fichier.close()
# print(contenu)
# Taper contenu affiche :
['0\t1\n', '1\t3\n', '4.5\t6\n', '7.8\t2.5']
```

Le programme suivant aboutit au même résultat que le précédent :

```
fichier=open("data.txt","r")
contenu=fichier.readlines()
t=[]
x=[]
for i in range(len(contenu)):
    a=contenu[i].split('\n')
    b=a[0].split('\t')
    t.append(float(b[0]))
    x.append(float(b[1]))
```

1.5 Et plus directement...

```
# Lecture du fichier contenant les points expérimentaux.
import numpy as np
t, val =np.loadtxt('pts-exp-rc.txt', unpack = True)
```

2 Tracer des courbes avec Python

Pour tracer des courbes sous **Python** , plusieurs bibliothèques existent. L'une des plus pratiques et utilisées est **matplolib**. Aussi, pour utiliser les outils de l'algèbre linéaire (manipulation des vecteurs et des matrices), on utilise la bibliothèque **numpy**. Pour combiner les deux d'un coup, on utilise **pylab**. C'est donc avec cette bibliothèque que le travail suivant est construit.

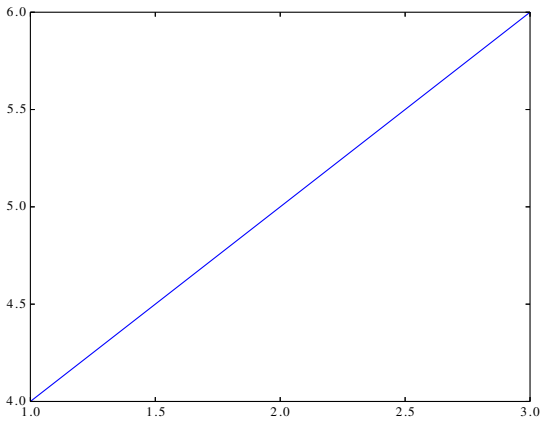
2.1 Tracer une courbe

Pour tracer des courbes[a], on utilise l'instruction `plot`. Pour afficher les figures, on utilise l'instruction `show()`.

```
from pylab import *
x=[1,2,3]
y=[4,5,6]
plot(x,y)
show()
```

a. Les " courbes " tracées sont en fait des segments de droites dont les extrémités sont définies par les éléments successifs d'une liste.

ce programme donne :



Q - 1 : *Tracer la courbe représentative de l'équation* $y(t) = y_f.\left(1 - e^{-t/\tau}\right)$ *avec* $y_f = 5$ *et* $\tau = 0,1$ *sur* $[0, 7.\tau]$ *et 200 points.*

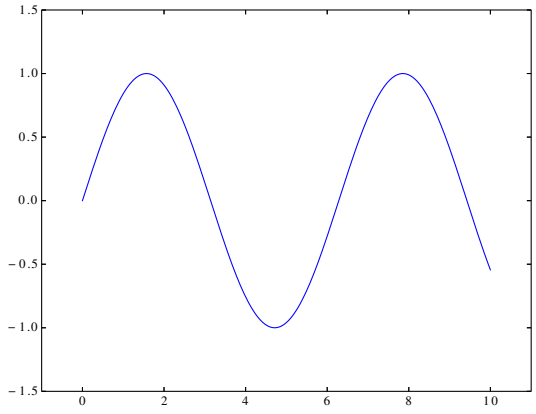
REMARQUE: On peut très bien faire une jolie boucle pour calculer les 200 abscisses. On peut aussi utiliser la commande `linspace` de **numpy**...

REMARQUE: et plus drôle que la syntaxe `for x in X` et ligne suivante `y=f(x)`, on peut utiliser `y=[f(x) for x in X]`.

2.2 Définition des axes

Pour définir les domaines des axes, on peut utiliser les instructions `xlim(xmin,xmax)` et `ylim(ylim,ymax)`.

```
1 from pylab import *
2 from math import *
3 t=linspace(0,10,200)
4 y=[sin(i) for i in t ]
5 xlim(-1,11)
6 ylim(-1.5,1.5)
7 plot(t,y)
8 show()
```

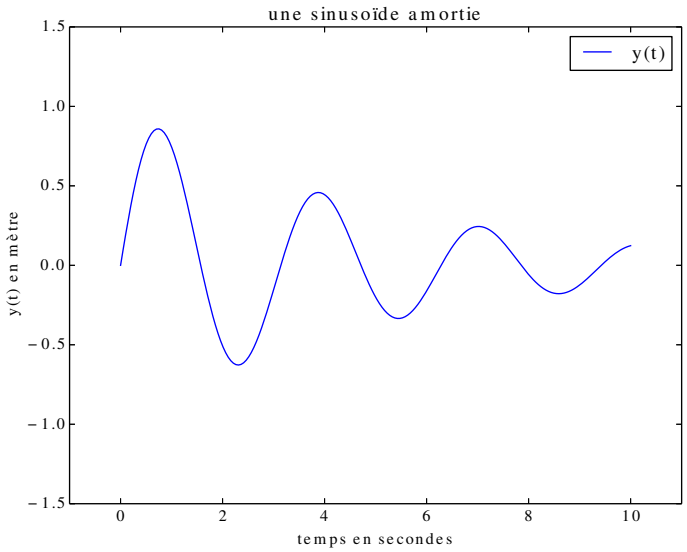


2.3 Mettre les légendes

On utilise l'instruction `title()` pour rajouter un titre, l'instruction legend() pour rajouter une légende (il faut obligatoirement utiliser label pour que la légende soit correctement écrite), et les instructions `xlabel()` et `ylabel()` pour rajouter un titre aux axes.

Pour définir les domaines des axes, on peut utiliser les instructions `xlim(xmin,xmax)` et `ylim(ylim,ymax)`.

```
1 from pylab import *
2 from math import *
3 t=linspace(0,10,200)
4 y=[sin(2*i)*exp(-i/5) for i in t ]
5 xlim(-1,11)
6 ylim(-1.5,1.5)
7 title("une sinusoïde amortie")
8 xlabel("temps en secondes")
9 ylabel(" y(t) en mètre")
10 plot(t,y,label="y(t)")
11 legend()
12 show()
```



2.4 Modifier le style des courbes

Il est possible de changer la couleur des courbes, le style des courbes, le symbole des courbes (ou marqueur, *marker*), et la

largeur des courbes.

Chaîne	Effet
-	solid line style
--	dashed line style
-.	dash-dot line style
:	dotted line style
.	point marker
,	pixel marker

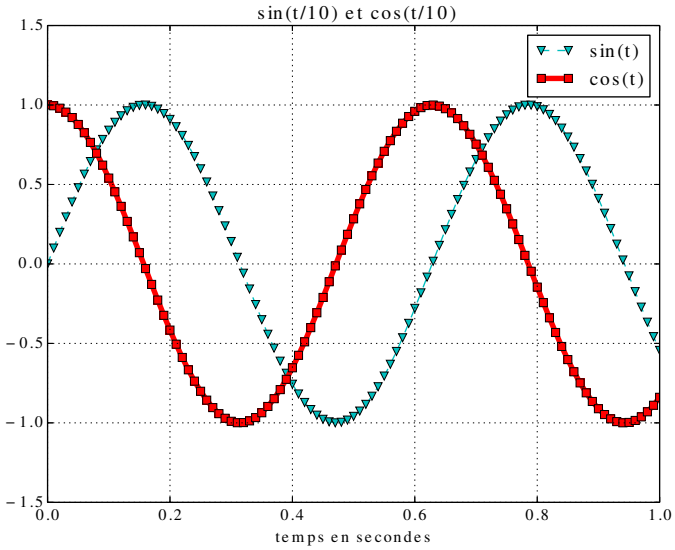
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker

3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker

+	plus marker
x	x marker
D	diamond marker
d	thin_diamond marker
|	vline marker
_	hline marker

Chaîne	b	g	r	c	m	y	k	w
Couleur	bleu	vert	rouge	cyan	magenta	jaune	noir	blanc

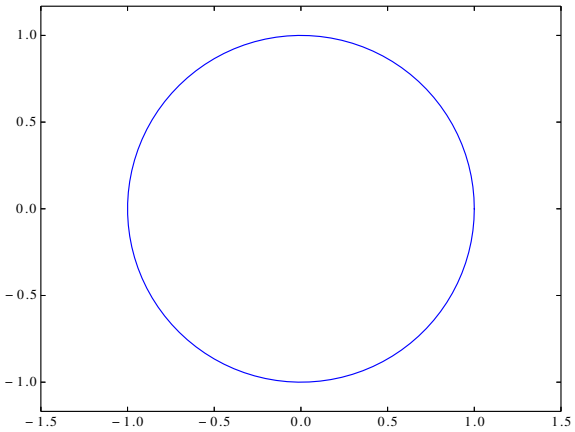
```
from pylab import *
from math import *
t=[]
x=[]
y=[]
for i in range(101):
    t.append(i/100)
    x.append(sin(t[i]*10))
    y.append(cos(t[i]*10))
ylim(-1.5,1.5)
plot(t,x,'c--v',label="sin(t)")
plot(t,y,'r-s',label="cos(t)",linewidth=4)
legend()
xlabel("temps en secondes")
title(" sin(t/10) et cos(t/10)")
grid()#permet d'ajouter une grille
show())
```



2.5 Définir le ratio abscisse/ordonnée

OBJECTIF : Faire apparaître à l'écran un cercle.

```
from pylab import *
from math import *
n=100
t=[i*2*pi/(n-1) for i in range(n)]
x=[]
y=[]
for i in range(n):
    x.append(cos(t[i]))
    y.append(sin(t[i]))
axis('equal')
xlim(-1.5,1.5)
ylim(-1.5,1.5)
plot(x,y)
show()
```



L'équation paramétrique d'un cercle est : $\begin{cases} x(t) &= \cos(t) \\ y(t) &= \sin(t) \end{cases}$ $\forall t \in [0; 2.\pi]$.

Pour *contraindre* les axes à avoir la même graduation en abscisse et en ordonnée, on place l'instruction `axis('equal')` avant l'insertion des bornes.

3 Diagrammes de Bode

3.1 Objectif

On cherche ici à tracer les diagrammes de Bode de la fonction suivante :

$$H(p) = \frac{K}{1 + \tau.p} \text{ avec } K = 10 \text{ et } \tau = 0,5 \text{ s}$$

On appelle $G_{dB}(\omega)$ le gain en décibel de $H(j.\omega)$ et $\varphi(\omega)$, la phase de $H(j.\omega)$.

$$G_{dB}(\omega) = 20.\log\left(\frac{K}{\sqrt{1 + (\tau.\omega)^2}}\right) \quad \text{et} \quad \varphi(\omega) = -\arctan(\tau.\omega)$$

OBJECTIF : Tracer les diagrammes de Bode de H pour $\omega \in \left[10^{-2}; 10^{3}\right]$.

3.2 Echelle logarithmique

Les diagrammes de Bode étant en échelle logarithmique, il convient d'avoir n points en abscisses répartis de façon logarithmique.

Q - 2 : *Ecrire un programme puls(n) ayant pour entrée :*

- *le nombre de points en abscisse*
- *la valeur d'abscisse la plus basse*
- *la valeur d'abscisse la plus élevée*

et en sortie la liste des abscisses.

Pour obtenir le résultat avec 200 points, les petits malins auront tenté le programme suivant :

```
1 from pylab import *
2 w=logspace(0.01,1000,200)
```

Q - 3 : *Qu'en est-il ? Comment fonctionne l'instruction logspace ?*

Le tracé nécessite une échelle logarithmique en abscisse uniquement. On utilise l'instruction `semilogx()`.

REMARQUE: Pour tracer une diagramme semi-logarithmique en y on utilise `semilogy()` et pour un tracé log-log on utilise `loglog()`.

```
1 semilogx(puls(n),gdb)
2 xlim(0.001,10000)
3 ylim(-40,25)
4 grid('on',which='both') #permet de tracé le quadrillage 'fin'
5 show()
```

3.3 Tracer plusieurs figures dans une même fenêtre

Les diagrammes de Bode sont souvent représentés l'un au dessus de l'autre (le gain en haut, la phase en bas).

On utilise `subplot()`pour résoudre afficher plusieurs figures dans une même fenêtre.

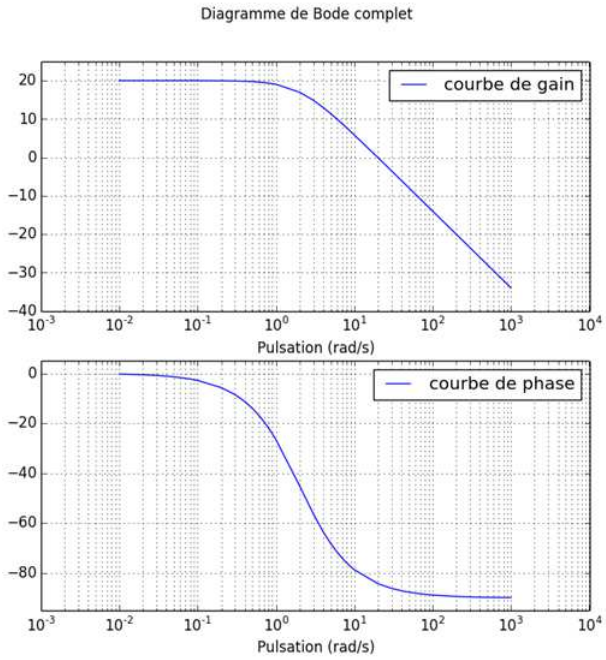
- le premier argument de la fonction est le nombre de lignes de figures désiré
- le deuxième, le nombre de colonnes
- le troisième argument, donne le numéro de la figure

Python remplit alors les figures lignes par lignes comme on remplirait les cases d'un tableau, ligne par ligne.
Ainsi les diagrammes de Bode sont donnés par le programme suivant :

```
subplot(2,1,1)
semilogx(puls(n),gdb,label="courbe de gain")
xlim(0.001,10000)
ylim(-40,25)
grid('on',which='both')
legend()
xlabel("Pulsation (rad/s)")

subplot(2,1,2)
semilogx(puls(n),phi,label="courbe de phase")
xlim(0.001,10000)
ylim(-95,5)
grid('on',which='both')
legend()
xlabel("Pulsation (rad/s)")

suptitle("Diagramme de Bode complet")
show()
```



3.4 Nombres complexes - module et argument

```
from pylab import *
print(abs(-1))
print(abs(1j))
print(abs(1+1j))
a=-1j
print(arctan2(a.imag,a.real))
```

Q - 4 : *A partir des lignes ci-dessus, écrire un programme permettant de tracer les diagrammes de Bode d'une fonction.*