P1 : Tracés de courbes

L'objectif de ce TP, une introduction aux méthodes numériques, est de maitriser le module `numpy` (qui permet de faire du calcul numérique), le module `matplotlib` (qui permet de tracer des courbes).

Vous trouverez en annexes quelques rappels permettant d'utiliser les bibliothèques `numpy` et `matplotlib` de Python. Taper celles-ci dans la console ou dans un script pour tester ce qu'elles font.

Commencez donc par importer les bibliothèques nécessaires :

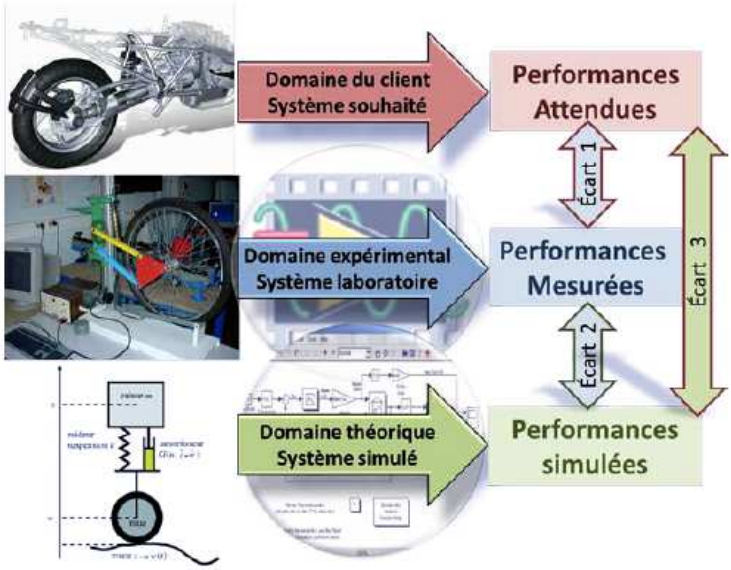
```
import matplotlib.pyplot as plt
import numpy as np
```

Ce TP d'informatique s'inscrit dans l'épreuve de manipulation de Sciences Industrielles dans le cadre des oraux de TSI.

Sur le système de la suspension, vous aurez par exemple à :

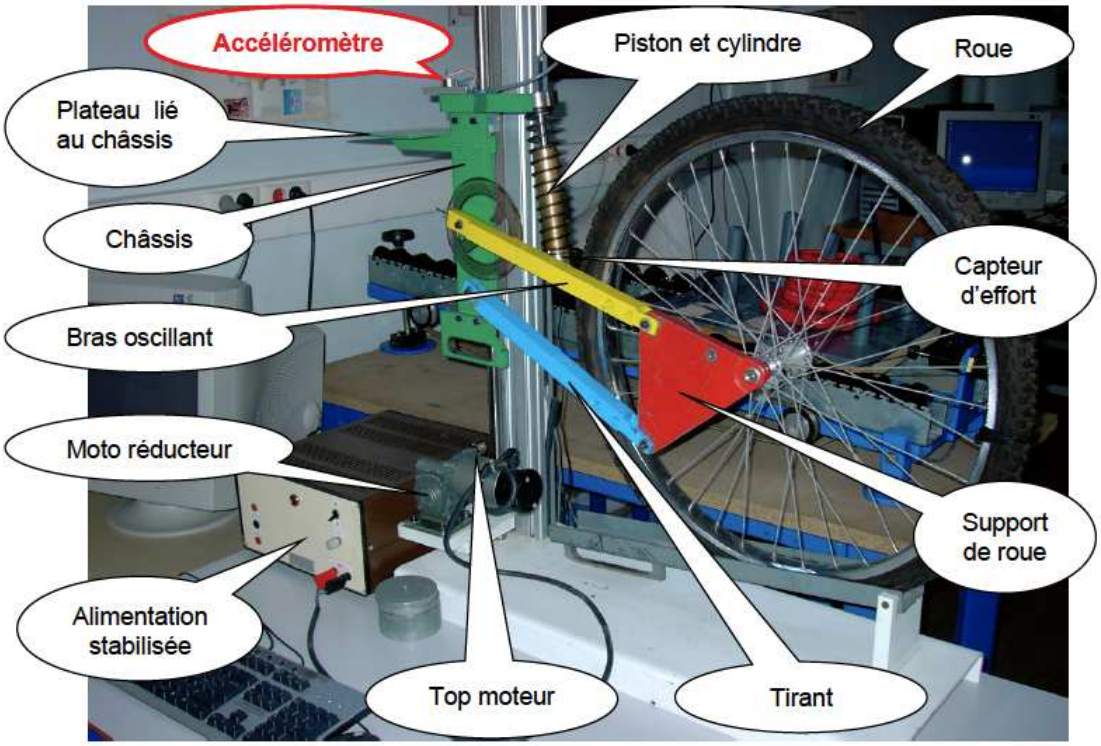
- Analyser les performances et la structure de la suspension,
- Comparer les solutions Monolever et Paralever par des études expérimentales,
- Modéliser la suspension par un système masse amortisseur simple,
- Utiliser un logiciel de simulation…

Dans chaque situation, il faudra comparer les différents écarts résumés ci-dessous.

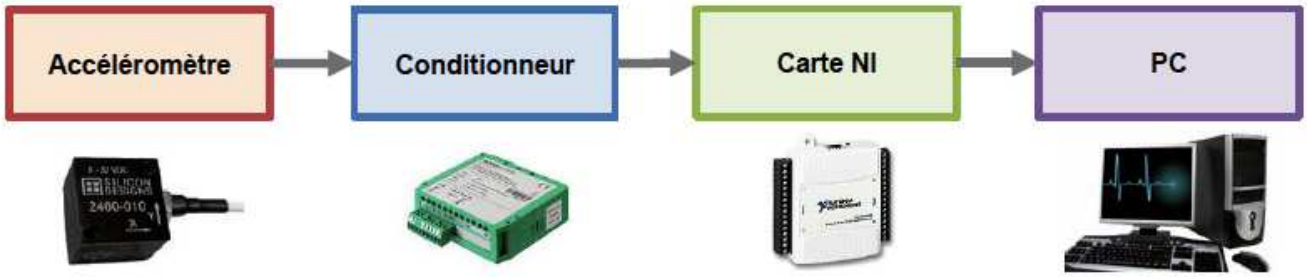


Dans le cadre de ce TP, les écarts avec **le domaine expérimental sont l'objet de l'étude**. Il va falloir traiter les signaux issus des différents capteurs.

On rappelle sur la photo ci-après les différents éléments constitutifs de la maquette du laboratoire. Nous nous intéressons plus particulièrement aux signaux acquis par l'accéléromètre situé sur le plateau lié au châssis.



La chaîne d'acquisition complète pour l'acquisition des signaux de l'accéléromètre est la suivante :



Chaîne d'acquisition

Le capteur fournit une tension électrique. La carte NI permet de relier le conditionneur de capteur au PC. Les fichiers ont ainsi été enregistrés lors des essais sous le format *.csv grâce au logiciel Matlab.

Aucune opération numérique de filtrage n'a été effectuée. Seule la sensibilité de l'accéléromètre a été prise en compte (2,548 V/g), où g désigne la constante de la pesanteur.

Q.1. Que vaut alors cette sensibilité (gain) dans le système international d'unités.

Avant de commencer la suite, copier-coller l'ensemble du dossier « Programmes » partagé à l'endroit habituel sur le réseau, dans un répertoire à votre nom : « NOM_Prénom » dans un espace libre d'accès en écriture.

2. Analyser les mesures « brutes »

On veut commencer par tracer l'évolution des accélérations de chaque essai en fonction du temps. Trois essais ont été effectués à 3 valeurs différentes de vitesses de rotation du moteur générant via excentrique, l'excitation de la suspension. Les enregistrements ont été réalisés durant chaque essai. On retrouve alors à chaque pas d'échantillonnage la date de la mesure en secondes et l'accélération non traitée numériquement en rad/s².

Les mesures sont stockées dans 3 fichiers distincts :
- Vit1.csv ;
- Vit2.csv ;
- Vit3.csv.

La capture d'écran ci-contre, montre la structure de l'un de ces fichiers après l'avoir ouvert avec un tableur.

	A	B
1	t	a
2	0.000000	0.069942
3	0.002778	0.069942
4	0.005556	0.030844
5	0.008333	0.069942
6	0.011111	0.109040
7	0.013889	0.148138
8	0.016667	0.148138
9	0.019444	0.148138

En réalité, si l'on ouvre ce fichier avec un éditeur de texte, la séparation sur une ligne, des données d'une colonne à une autre est réalisée grâce à un point virgule (format csv).

2.1. Analyser les données temporelles d'un essai

Nous allons procéder dans un premier temps au tracé d'une seule courbe (celle de l'essai Vit1).
- Ouvrir le fichier script « **Etude1.py** ».

Q.2. Analyser les premières lignes du script (lignes 9 à 14 incluses). Spécifier leur rôle…

On rappelle que la variable fi est itérable. Nous allons alors stocker sous forme de listes distinctes les dates et les valeurs des accélérations associées à chaque date.

Q.3. Compléter les lignes de programme sous le commentaire : « #Stockage sous forme de listes des données brutes ». On pourra utiliser la méthode split pour les listes et les chaines de caractères :

```
=====
split
=====

Definition: split(sep=None, maxsplit=-1)
Type: Function

----

S.split(sep=None, maxsplit=-1) -> list of strings

Return a list of the words in S, using sep as the delimiter string.  If maxsplit
is given, at most maxsplit splits are done. If sep is not specified or is None,
any whitespace string is a separator and empty strings are removed from the
result.
```

On souhaite maintenant tracer l'évolution de l'accélération en fonction du temps.

Q.4. Compléter les lignes de programme permettant de tracer cette courbe. On pourra se référer aux Annexes et on prendra soin de légender les axes.

Q.5. Analyser la courbe obtenue. Qu'observe-t-on ?

2.2. Imaginer et concevoir un programme itératif pour tracer les courbes des trois essais

On souhaite désormais tracer sur une même figure les courbes d'accélération relatives aux trois essais effectués. On notera bien que la longueur de chaque essai peut être différente.

Q.6. Concevoir un programme itératif permettant de tracer ces trois courbes sur une même figure.

2.3. Evaluer la valeur moyenne des signaux

On souhaite déterminer la valeur moyenne de l'accélération sur la durée d'acquisition pour chaque essai.

Q.7. Modifier le programme précédent afin de d'afficher dans la console, lors de son exécution les valeurs moyennes des accélérations. Le résultat affiché dans la console devra prendre cette forme :

```
La valeur moyenne de l'acceleration de l'essai 1 est : 0.45 m/s^2
```

3. Evaluer la fréquence fondamentale par transformée de Fourrier rapide

Tout signal temporel possède une représentation spectrale. À l'aide des échantillons de ce signal, il existe un algorithme appelé FFT, acronyme de l'anglais *Fast Fourier Transform*, qui permet d'avoir une représentation approchée de son spectre. Cet algorithme, de complexité quasi-linéaire, est implanté par exemple dans les oscilloscopes numériques, les cartes graphiques des ordinateurs, les tableurs, et dans toutes les bibliothèques de calcul numérique.

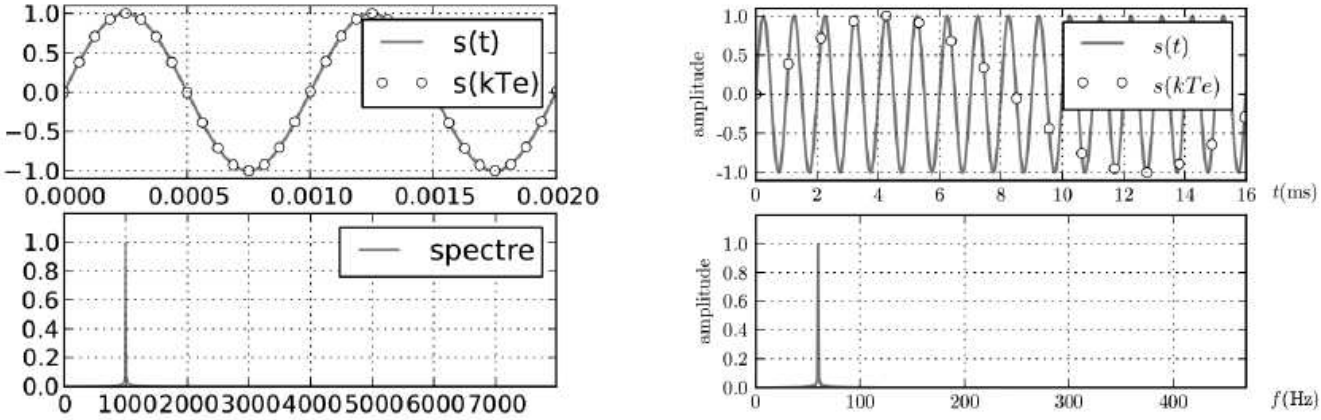
On ouvrira pour la suite, le fichier script « **Etude2.py** ».

3.1. Contrôler la fréquence d'échantillonnage de l'acquisition

Il est important de noter qu'un signal est correctement représenté à partir de ses échantillons, si la fréquence d'échantillonnage est supérieure à deux fois la fréquence maximale de son spectre.
Ce théorème s'énonce parfois à l'aide de la condition de Nyquist-Shannon, qui est la condition nécessaire à une bonne représentation :

$$fe > 2.f\max$$

Sur les deux exemples suivants, la figure de gauche respecte le critère de Shanon, celle de droite, non.



A gauche : $f_e = 16.f$; à droite : $fe = 16.f/17$

Q.8. Contrôler si, dans le cas de l'essai Vit3, le théorème de Shannon est bien vérifié.

3.2. Tracer le spectre « brut »

Nous n'avons besoin ici que de tracer le module de la transformée de Fourier. Les deux fonctions utiles pour l'étude : fft et fftreq de la bibliothèque fftpack du module scipy sont décrites dans les Annexes.

Q.9. Spécifier ce que réalise la ligne 31 du code du script « Etude2.py ».

Les valeurs d'amplitude étant stockées, il nous faut maintenant construire une liste contenant l'ensemble des valeurs de fréquence.

Q.10. Stocker dans une liste signal_freq l'ensemble des valeurs échantillonnées du domaine fréquentiel. On utilisera pour cela la fonction fftreq ; et on saisira le code dans l'emplacement prévu à cet effet.

Q.11. Concevoir le code permettant d'afficher le spectre fréquentiel.

Q.12. Que constate-t-on ? Contrôler plus particulièrement l'axe des abscisses (valeur maximale de fréquence, à mettre en corrélation avec le nombre total de points de mesure de l'acquisition et la fréquence d'échantillonnage).

3.3. Concevoir un programme améliorant la fenêtre de visualisation du spectre

Afin de ne tracer le spectre que sur des fréquences à valeurs positives et sur un intervalle correspondant à la physique du système, on va modifier légèrement les listes précédentes pour tracer le spectre dans une fenêtre de visualisation plus adaptée.

Q.13. D'après le spectre tracé dans le paragraphe précédent et au vu du comportement du système, quel doit être approximativement l'intervalle fréquentiel d'étude ?

Pour les deux questions suivantes, coder dans l'emplacement du programme laisse libre dans le script.

Q.14. Concevoir un programme permettant de de ne garder que les valeurs intéressantes pour notre étude. On essaiera de faire le plus simple possible, en utilisant les possibilités offertes par les listes pour extraire les valeurs souhaitées (ce qu'on utiliserait par exemple avec MATLab®).

Q.15. Tracer enfin la figure du spectre fréquentiel sur la fenêtre de visualisation optimisée.

3.4. Evaluer la fréquence fondamentale de l'essai

On se propose dans un dernier temps de déterminer le mode fréquentiel dominant, en recherchant la fréquence fondamentale (première harmonique ou harmonique de rang 1). La fréquence fondamentale est obtenue pour l'amplitude la plus importante du spectre.

Q.16. Concevoir un programme permettant d'extraire la fréquence fondamentale de l'essai Vit3 et l'afficher dans la console lorsque l'on exécute le script « Etude2.py ».

ANNEXE 1: BIBLIOTHÈQUE NUMPY

Numpy est un module de calcul numérique qui permet de réaliser des opérations complexes via unmode non interprété sur des données de type vecteur, matrice, tableau multidimensionnel.Pour charger le module, il faut commencer par taper :

```
Import numpy as np
```

Contrairement au mode standard de Python, toutes les variables sont stockés en mémoire en suivant les conventions du C, on retrouve ainsi les codages des entiers non signés (uint16, uint32, uint64), les entiers signés (int8, int16, int32, int64), les flottants (float16, float32, float64).

Les éléments créés par numpy sont tous codés suivant un de ses types, accessible par l'attribut dtype.Il faut bien garder à l'esprit que la représentation des données en mémoire est limité, par exemple taper dans la console :

```
np.float16 (0.3)
np.float32 (0.3)
np.float64 (0.3)
np.float16 (0.3) == 0.3
np.float32 (0.3) == 0.3
np.float64 (0.3) == 0.3
```

La comparaison de manière exacte entre deux valeurs numériques ne donne pas forcément le résultat attendu. C'est la raison pour laquelle en calcul numérique, la comparaison à 0 se fait toujours à un ε près.Il existe de nombreuses commandes pour créer des vecteurs avec numpy, essayer :

```
a=np.array([0,1,2,3])                    # construction d'un vecteur
a.dtype
a=np.array([0,1,2,3.0])
a.dtype # python determiner le type automatiquement
a=np.array([0,1,2,3.0],np.float16) # mais on peut aussi imposer le type
a.dtype
a.shape # donne la dimension du vecteur
b=a.reshape(2,2) # permet de transformer le vecteur en matrice
```

On peut aussi créer des éléments remplies de 0 ou de 1, utile pour initialiser les variables :

```
np.zeros(5)
np.zeros((4 ,3))
np.ones((2 ,3))
```

On peut aussi créer des vecteurs par un objet de type range :

```
np.arange(0,1,0.1)              # arange(ini,fin,pas)
np.linspace(0,1,101)          # linspace(ini,fin,nbval)
```

Numpy définit la plupart des opérations mathématiques sous forme vectorielle, par exemple pour le sinus :

```
t=np.arange(0,1,0.01)     # intervalle de temps
T=0.25                    # periode du sinus
f=np.sin(2*np.pi/T*t)     # calcul de la fonction pour toutes les valeurs
```

ANNEXE 2: BIBLIOTHÈQUE MATPLOTLIB

Matplotlib est un module qui permet de tracer des courbes comme sur les logiciels de calcul scientifique de type Matlab/Scilab. Pour charger le module, il faut commencer par taper :

```
import matplotlib . pyplot as plt
f2=np.sin(np.pi/T*t)                                    #une autre courbe
plt.plot(t,f)                              # permet de tracer la courbe f
plt.plot(t,f2)                            # permet de tracer la courbe f2

# mettre des titres aux axes , figure , legende est obligatoire
plt.xlabel('Temps (s)')
plt.ylabel('f(t)')
plt.title('Trace de la courbe sin en fonction du temps ')
plt.legend(('f1','f2'))
plt.show ()
```

Si vous voulez réaliser plusieurs figures alors il faudra utiliser explicitement la création de celles-ci :

```
fig1=plt.figure()
fig11=fig1.add_subplot(1,2,1)
            # add_subplot (l, c, s) permet de creer une zone avec
            #l ligne , c colonne et on selectionne la case s
            # puis selectionner la 1ere
fig11.plot(t,f)
plt.xlabel('Temps (s)')
plt.ylabel('f(t)')
fig12=fig1.add_subplot(1 ,2 ,2) #on selectionne la seconde
fig12.plot(t,f)
plt.xlabel('Temps (s)')
plt.ylabel('f2(t)')
fig1.suptitle('Trace de deux courbes differentes : fig1 ')
#2eme figure on superpose les courbes
fig2=plt.figure()
fig21=fig2.add_subplot(1,1,1) # creer une figure avec une seule zone de trace
fig21.plot(t,f) # permet de tracer la courbe f
fig21.plot(t,f2) # permet de tracer la courbe f2
plt.xlabel('Temps (s)')
pltylabel('f(t)')
plt.title('Trace de la courbe sin en fonction du temps : fig2 ')
plt.legend(('f1','f2'))
fig1.savefig('fig1 .png ') # sauvegarde des figures dans un fichier
fig2.savefig ('fig2 .png ') # sauvegarde des figures dans un fichier
plt.show()
```

ANNEXE 3 : FFT

```
=======
fftfreq
=======

Definition: fftfreq(n, d=1.0)
Type: Function of numpy.fft.helper module

----

Return the Discrete Fourier Transform sample frequencies.

The returned float array contains the frequency bins in cycles/unit (with zero at
the start) given a window length `n` and a sample spacing `d`::

  f = [0, 1, ..., n/2-1, -n/2, ..., -1] / (d*n)         if n is even
  f = [0, 1, ..., (n-1)/2, -(n-1)/2, ..., -1] / (d*n)   if n is odd

Parameters
----------
n : int
    Window length.
d : scalar
    Sample spacing.

Returns
-------
out : ndarray
    The array of length `n`, containing the sample frequencies.

Examples
--------
>>> signal = np.array([-2, 8, 6, 4, 1, 0, 3, 5], dtype=float)
>>> fourier = np.fft.fft(signal)
>>> n = signal.size
>>> timestep = 0.1
>>> freq = np.fft.fftfreq(n, d=timestep)
>>> freq
array([ 0.  ,  1.25,  2.5 ,  3.75, -5.  , -3.75, -2.5 , -1.25])
```

```
===
fft
===

Definition: fft(x, n=None, axis=-1, overwrite_x=0)
Type: Function of scipy.fftpack.basic module

----

Return discrete Fourier transform of real or complex sequence.

The returned complex array contains ``y(0), y(1),..., y(n-1)`` where

``y(j) = (x * exp(-2*pi*sqrt(-1)*j*np.arange(n)/n)).sum()``.

Parameters
----------
```

```
x : array_like
    Array to Fourier transform.
n : int, optional
    Length of the Fourier transform.  If ``n < x.shape[axis]``, `x` is
    truncated.  If ``n > x.shape[axis]``, `x` is zero-padded. The
    default results in ``n = x.shape[axis]``.
axis : int, optional
    Axis along which the fft's are computed; the default is over the
    last axis (i.e., ``axis=-1``).
overwrite_x : bool, optional
    If True the contents of `x` can be destroyed; the default is False.

Returns
-------
z : complex ndarray
    with the elements::

        [y(0),y(1),..,y(n/2),y(1-n/2),...,y(-1)]        if n is even
        [y(0),y(1),..,y((n-1)/2),y(-(n-1)/2),...,y(-1)]  if n is odd

    where::

        y(j) = sum[k=0..n-1] x[k] * exp(-sqrt(-1)*j*k* 2*pi/n), j = 0..n-1

    Note that ``y(-j) = y(n-j).conjugate()``.

See Also
--------
ifft : Inverse FFT
rfft : FFT of a real sequence

Notes
-----
The packing of the result is "standard": If A = fft(a, n), then A[0]
contains the zero-frequency term, A[1:n/2+1] contains the
positive-frequency terms, and A[n/2+1:] contains the negative-frequency
terms, in order of decreasingly negative frequency. So for an 8-point
transform, the frequencies of the result are [ 0, 1, 2, 3, 4, -3, -2, -1].

For n even, A[n/2] contains the sum of the positive and negative-frequency
terms. For n even and x real, A[n/2] will always be real.

This is most efficient for n a power of two.

Examples
--------
>>> from scipy.fftpack import fft, ifft
>>> x = np.arange(5)
>>> np.allclose(fft(ifft(x)), x, atol=1e-15)  #within numerical accuracy.
True
```