P3 : Intégration et filtrage

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 :

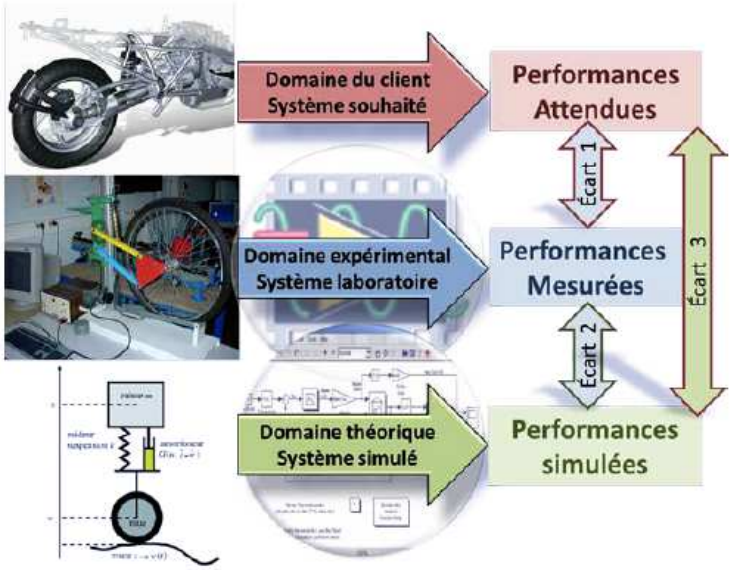
```
import matplotlib.pyplot as plt
import numpy as np
```

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 :

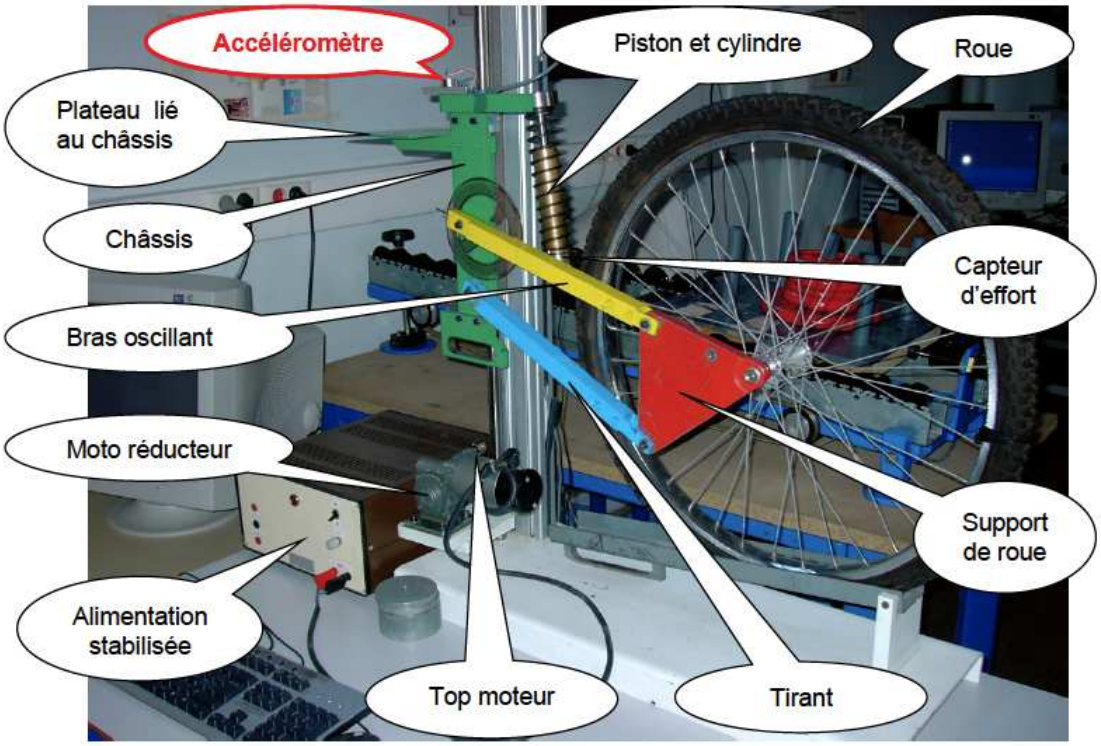
- Analyser les performances et la structure de la suspension,
- Comparer les solutions Monolever et Paralever par des études expérimentales,
- Modéliser la suspension par un système masse amortisseur simple,
- Utiliser un logiciel de simulation…

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

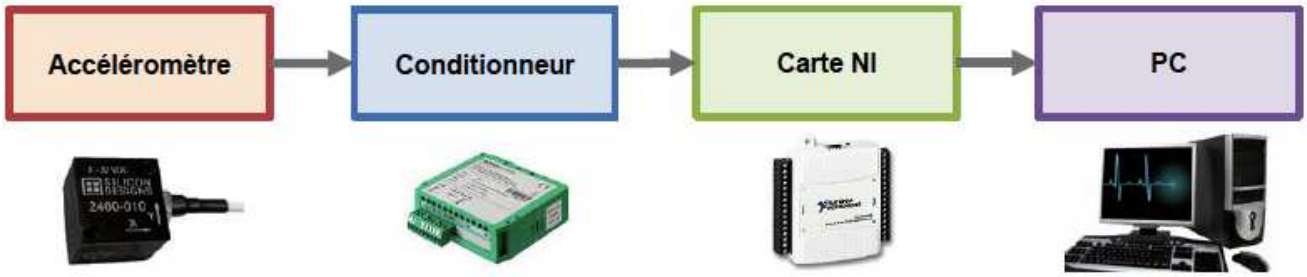


Dans le cadre de ce TP, les écarts avec **le domaine expérimental sont l'objet de l'étude**. Il va falloir traiter les signaux issus des différents capteurs.

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.

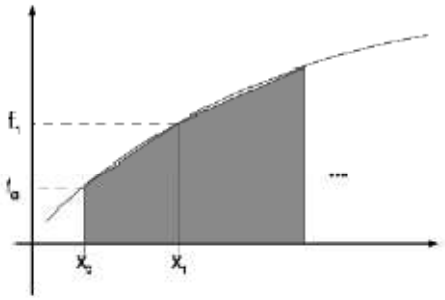
Avant de commencer la suite, copier-coller l'ensemble du dossier « Programmes » partagé à l'endroit habituel sur le réseau, dans un répertoire à votre nom : « NOM_Prénom » dans un espace libre d'accès en écriture.

1. Intégrer numériquement pour obtenir vitesse et position

On travaillera désormais sur le fichier « Etude3.py ».

1.1. Intégrer numériquement par la méthode des trapèzes

On se propose dans cette partie de mettre en oeuvre une méthode d'intégration numérique : la méthode des trapèzes. La figure ci-dessous présente schématiquement la méthode.



La zone grisée représente l'aire sous la courbe

Q.1. Exprimer sur l'intervalle [*x*0 ; *x*1] la valeur approchée de l'intégrale obtenue par la méthode des trapèzes.

1.2. Traduire en langage Python l'algorithme d'intégration numérique

Le script « Etude3.py » contient le code pour tracer l'évolution de la vitesse en fonction du temps pour les trois essais sur une figure et pour tracer l'évolution de la position en fonction du temps pour les trois essais sur une deuxième figure. Ces vitesses et positions en fonction du temps sont obtenues par intégration numérique directe du signal brut.

Il nous faut dans un premier temps définir les fonctions permettant de réaliser l'intégration numérique.

Q.2. Traduire au sein de la fonction `trapeze` le programme permettant de calculer l'aire sur un segment [*xi* ; *xj*].

Q.3. Imaginer et concevoir la fonction `integre_signal` qui contient en arguments d'entrée : une liste t contenant les instants de l'acquisition ; une liste f contenant à chacun de ces instants la valeur du signal que l'on souhaite intégrer ; une condition initiale que l'on prendra nulle, sauf indication contraire. Le résultat retourné sera une liste contenant le signal intégré (la taille de cette liste sera la même que les deux listes d'entrée).

Une fois ces deux fonctions programmés, dé-commenter ce qui se trouve sous l'instruction demandant de ne pas modifier le reste du programme

Q.4. Analyser rapidement le contenu de ce code. Que fait-on ?

Exécuter alors l'ensemble du script « Etude3.py ».

Q.5. Conclure quant aux courbes finalement obtenues.

2. Filtrer numériquement les données

2.1. Analyser et modéliser le type de filtre à utiliser

Nous avons vu précédemment que les résultats d'intégration numérique n'étaient pas adaptés. Notamment, on observe une dérive des valeurs. Cela provient entre autre du fait que le signal acquis est bruité.
Plusieurs solutions sont alors possibles pour ne pas engendrer ce phénomène : Filtrer avant l'intégration numérique et/ou filtrer après cette opération.

Q.6. Dans le cas d'un filtrage situé en amont de l'intégration numérique, quel doit être le type de filtre ? Justifier.

Q.7. Dans le cas d'un filtrage situé en aval de l'intégration numérique, quel doit être le type de filtre ? Justifier.

On se propose d'utiliser un filtre que l'on utilisera avant l'opération d'intégration.

Q.8. Modéliser ce filtre par une fonction de transfert du premier ordre dont on précisera les coefficients caractéristiques.

2.2. Imaginer et concevoir un filtre passe-bas dans le langage Python

L'équation de transfert du filtre énoncée précédemment est donnée dans le domaine continu. Or nous travaillons ici dans le domaine discret (échantillonnage).

Soit $s[n]$ la valeur du signal filtré à l'instant $n.Te$, où Te est la période d'échantillonnage du signal et n le numéro d'échantillon. On note de même $e[n]$ la valeur du signal non filtré.

En traduisant l'équation différentielle du filtre passe bas modélisé dans le paragraphe précédent dans le domaine discret ($dt = Te$), on peut alors écrire l'équation numérique du filtre.

Q.9. Ecrire cette équation aux différences en exprimant $s[n+1]$ en fonction de $s[n]$, $e[n]$, Te et des constantes caractéristiques de la fonction de transfert du premier ordre.

Q.10. Programmer dans le script « Etude4.py » cette fonction filtre : filtre_passe_bas_1. On notera que tau représente l'inverse de la pulsation de coupure du filtre. Si on laisse par défaut (valeur nulle), le programme devra alors prendre une valeur égale à $10.Te$.

• Après avoir programmé cette fonction, exécuter l'ensemble du script, après avoir dé-commenter les lignes situées en fin de programme (voir le fichier).

Q.11. Conclusions ?

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
```