TRAITEMENT DE L'INFORMATION : ANALYSER, FILTRER, INTEGRER…

L'étude menée ci-dessous s'inscrit dans le cadre de l'usage de l'informatique (IPT) pour l'épreuve de TP de Sciences Industrielles lors des oraux de TSI (concours CCP ou Supélec).

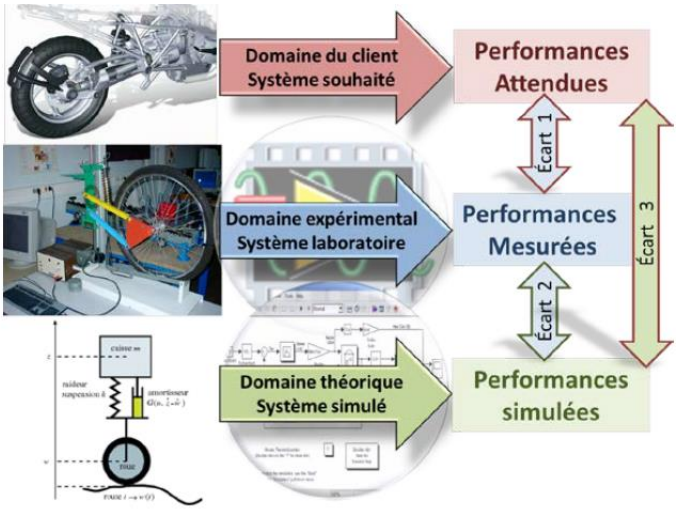
L'application porte ici sur les relevés d'accélération du châssis d'une moto à l'aide d'un accéléromètre. On vous fournit les fichiers de points sous forme numérique (fichiers .CSV) et vous en exploitez les résultats.

On peut également exploiter ce TP pour enrichir la partie expérimentale du TIPE à partir d'une acquisition (carte Arduino ; NI6009 ou oscillo Fluke) et récupération des données numériques dans un tableur.

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

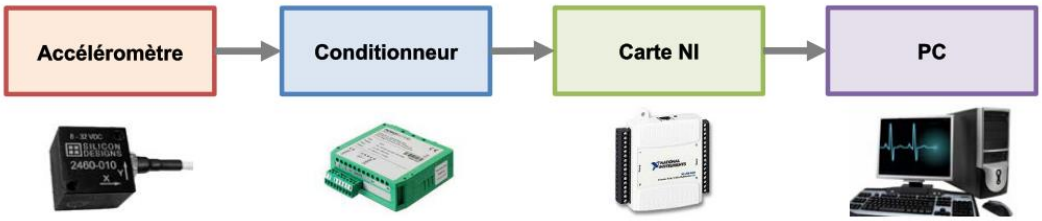
- Analyser les performances et la structure de la suspension,
- Comparer les solutions Monolever et Paralever par des études expérimentales,
- Modéliser la suspension par un système masse amortisseur simple,
- Utiliser un logiciel de simulation…

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



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.

La chaîne d'acquisition complète pour l'acquisition des signaux à partir du capteur 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 ensuite été enregistrés lors des essais sous le format *.csv ou .txt grâce au logiciel NI_EXPRESS.

Aucune opération numérique de filtrage n'a été effectué et la sensibilité S du capteur doit être prise en compte (ici S = 2,548V/g avec g = 9.81m/s²).

2. Exploitation des mesures « brutes » dans un tableur puis dans un script Python

On veut commencer par tracer l'évolution des accélérations de chaque essai en fonction du temps.

Trois essais ont été effectués pour trois conditions différentes (ici vitesse 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 disponibles dans le dossier ci-joint : **Vit1.csv ; Vit2.csv ; Vit3.csv.**

Avant de commencer la suite, copier-coller l'ensemble du dossier « Programmes » dans un dossier unique afin les fichiers de valeurs numériques soit correctement appelés par les scripts Python. Au lycée, utiliser un répertoire à votre nom : « NOM_Prénom » dans un espace libre d'accès en écriture.

A) COURBE OBTENUE A PARTIR DU FICHIER DE POINTS DANS UN TABLEUR

Q1. Ouvrir le fichier Vit1.csv avec un **tableur (Excel ou Odt**), *remplacer le point par une virgule pour chaque donnée…* et effectuer le tracé sous forme de graphe temporel en convertissant la tension du capteur en accélération en m/s². Compléter la légende des 2 axes et les unités.

B) COURBE A PARTIR DU FICHIER DE POINTS PAR UN SCRIPT PYTHON

Q2. Ouvrir **le logiciel Python console Spyder** puis ouvrir le script **« Etude1.py ».** 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.

Q3. Compléter les lignes de programme sous le commentaire : « *#Stockage sous forme de listes des données brutes »*. On rappelle que certaines méthodes associées aux listes sont décrites dans les annexes.

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

Q4. Compléter les lignes de programme permettant de tracer cette courbe. On pourra se référer aux Annexes et on prendra soin de légender les axes.

Q5. Analyser la courbe obtenue. Qu'observe-t-on et quel type de traitement faut-il mettre en place ?

IMAGINER ET CONCEVOIR UN PROGRAMME ITERATIF 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.

Q6. Concevoir un programme itératif permettant de tracer ces trois courbes sur une même figure.

C) EVALUER LA VALEUR MOYENNE DES SIGNAUX

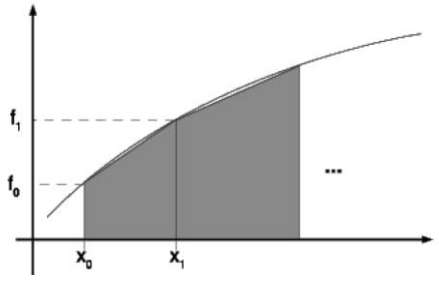
On souhaite déterminer la valeur moyenne de l'accélération sur la durée d'acquisition pour chaque essai.

Q7. 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'accélération de l'essai 1 est : 0.45 m/s^2 »*

D) INTEGRER NUMERIQUEMENT POUR OBTENIR VITESSE ET POSITION

On travaillera désormais sur le fichier « Etude3.py » pour intégrer numériquement par la méthode des trapèzes.

La figure ci-contre présente schématiquement la méthode d'intégration.



Q8. Exprimer sur l'intervalle $[x_0 ; x_1]$ la valeur approchée de l'intégrale obtenue par la méthode des trapèzes.

REALISER EN LANGAGE PYTHON L'ALGORITHME D'INTEGRATION NUMERIQUE

Le script « Etude3.py » contient le code pour tracer sur une première figure l'évolution de la vitesse en fonction du temps pour les trois essais et pour tracer sur une deuxième figure l'évolution de la position en fonction du temps pour les trois essais . 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.

Q9. Traduire au sein de la fonction « *trapeze* » le programme permettant de calculer l'aire sur un segment $[x_i ; x_j]$.

Q10. 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.

Q11. Analyser rapidement le contenu de ce code. Exécuter alors l'ensemble du script « Etude3.py ».

Q12. Conclure quant aux courbes finalement obtenues.

E) FILTRER NUMERIQUEMENT LES DONNEES

Analyser et modéliser le type de filtre à utiliser :

Nous avons vu précédemment que les résultats d'intégration numérique n'est pas adaptés, car on observe notamment 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.

Q13. Dans le cas d'un filtrage situé en amont de l'intégration numérique, quel doit être le type de filtre ? Justifier.

Q14. Dans le cas d'un filtrage situé en aval de l'intégration numérique, quel doit être le type de filtre ? Justifier.

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

Q15. Modéliser ce filtre par une fonction de transfert du premier ordre dont on précisera les coefficients caractéristiques.

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 $n.T_e$, où T_e 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 entrée du filtre).

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

Q16. Ecrire cette équation aux différences en exprimant s[n+1] en fonction de s[n], e[n], T_e et des constantes caractéristiques de la fonction de transfert du premier ordre.

Q17. 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 $10.T_e$.

Q18. Après avoir programmé cette fonction, exécuter l'ensemble du script, après avoir dé-commenter les lignes situées en fin de programme (voir le fichier). Conclure.

ANNEXES : Fonctions de la bibliothèque matplotlib.pyplot

```
====
plot
====

Definition: plot(*args, **kwargs)
Type: Function of matplotlib.pyplot module

----

Plot lines and/or markers to the
:class:`~matplotlib.axes.Axes`. *args* is a variable length argument, allowing for
multiple *x*, *y* pairs with an optional format string. For example, each of the
following is legal::

plot(x, y) # plot x and y using default line style and color
plot(x, y, 'bo') # plot x and y using blue circle markers
plot(y) # plot y using x as index array 0..N-1
plot(y, 'r+') # ditto, but with red plusses

If *x* and/or *y* is 2-dimensional, then the corresponding columns will be
plotted.

An arbitrary number of *x*, *y*, *fmt* groups can be specified, as in::

a.plot(x1, y1, 'g^', x2, y2, 'g-')

Return value is a list of lines that were added.

By default, each line is assigned a different color specified by a 'color cycle'.
To change this behavior, you can edit the axes.color_cycle rcParam. Alternatively,
you can use :meth:`~matplotlib.axes.Axes.set_default_color_cycle`.

The following format string characters are accepted to control the line style or
marker:

================ ===============================
character description
================ ===============================
``'-'`` 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
================ ===============================


The following color abbreviations are supported:

========== ========
character color
========== ========
'b' blue
'g' green
'r' red
'c' cyan
'm' magenta
'y' yellow
'k' black
'w' white
========== ========
```

```
======
xlabel
======

Definition: xlabel(s, *args, **kwargs)
Type: Function of matplotlib.pyplot module

----

Set the *x* axis label of the current axis.

Default override is::

override = {
'fontsize' : 'small',
'verticalalignment' : 'top',
'horizontalalignment' : 'center'
}
```

```
====
show
====

Definition: show(*args, **kw)
Type: Function of matplotlib.pyplot module

----

Display a figure.

When running in ipython with its pylab mode, display all figures and return to the
ipython prompt.
```

ANNEXE : Méthodes associées aux listes

```
=====
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.
```