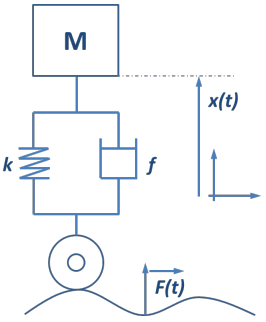
PROJET D'INFORMATIQUE

MODÉLISATION D'UN SYSTÈME AMORTI

1 Mise en situation

On cherche à connaître le comportement d'un système masse-ressort-amortisseur. Ce modèle de comportement est très couramment utilisé (modélisation d'un amortisseur de véhicule, modélisation du comportement visco-élastique des matériaux, mécanique vibratoire...).



Objectifs

L'objectif de ce projet est de modéliser et de simuler le comportement d'un système amorti.

2 Travail demandé

Pour mener à bien ce projet il est demandé de réaliser un certain nombre d'activités (non exhaustives).

1. Réaliser une recherche sur les domaines d'application du modèle masse amortisseur et trouver des triplets *(masse, raideur du ressort, coefficient d'amortissement)*
2. Modéliser le problème et déterminer la (ou les) équation(s) différentielle(s) liant le déplacement de la masse et de la sollicitation mécanique du système.
3. Résoudre le problème en utilisant plusieurs méthode :
   - résolution numérique de l'équation différentielle (en Python) ;
   - résolution analytique de l'équation différentielle ;
   - résolution de l'équation en utilisant le formalisme de Laplace (et éventuellement le module Xcos de Scilab ou Matlab-Simulink) ;
   - résolution de l'équation en utilisant la modélisation multiphysique (Scilab-Xcos-SIMM ou Matlab-Simulink).
4. Comparer les résultats des simulations et commenter les paramètres des solver.

3 Évaluation

L'évaluation se fera sous forme d'une présentation de 10 à 15 minutes (6 diapositives au maximum). Les élèves devront présenter au minimum :

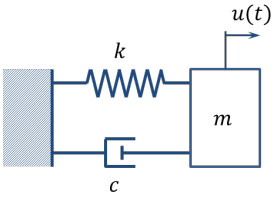
- la modélisation retenue ;
- la structure du programme en Python ;
- une démonstration de l'exécution du code Python.

Éléments de corrigé

1 Mise en équation du problème

On isole la masse mobile et on applique le théorème de la résultante dynamique en projection suivant l'axe de déplacement. En négligeant l'accélération de la pesanteur, on a :

$$f(t)-kx(t)-\mu\dot{x}(t)=M\ddot{x}(t) \tag{1}$$



2 Résolution numérique

On utilise le changement de variable suivant : $v(t)=\dot{x}(t)$. Résoudre l'équation différentielle précédent revient donc à résoudre le système suivant :

$$\begin{cases} v(t)=\dot{x}(t) \\ m\cdot\dot{v}(t)+c\cdot v(t)+k\cdot x(t)=f(t) \end{cases} \tag{2}$$

En utilisant un schéma d'Euler implicite et notant h le pas de calcul, on a :

$$\frac{dx(t)}{dt}\simeq\frac{x_n-x_{n-1}}{h}\Longrightarrow v_n=\frac{x_n-x_{n-1}}{h}\Longleftrightarrow x_n=h\cdot v_n+x_{n-1}$$

En réutilisant un schéma d'Euler implicite, on a :

$$\frac{dv(t)}{dt}\simeq\frac{v_n-v_{n-1}}{h}\Longrightarrow \dot{v}_n=\frac{v_n-v_{n-1}}{h}$$

En substituant les résultats précédents dans le système d'équation, on obtient :

$$\begin{aligned} m\cdot\frac{v_n-v_{n-1}}{h}+c\cdot\frac{x_n-x_{n-1}}{h}+k\cdot x_n=f_n &\Longleftrightarrow m\cdot(v_n-v_{n-1})+c\cdot(x_n-x_{n-1})+kh\cdot x_n=hf_n \\ &\Longleftrightarrow mv_n=hf_n-(kh+c)\cdot x_n+mv_{n-1}+cx_{n-1} \end{aligned}$$

Il est alors possible d'exprimer l'équation précédente en fonction de x_n :

$$\begin{aligned} & m\frac{x_n-x_{n-1}}{h}=hf_n-(kh+c)\cdot x_n+m\frac{x_{n-1}-x_{n-2}}{h}+cx_{n-1} \\ \Longleftrightarrow\quad & m(x_n-x_{n-1})=h^2f_n-h(kh+c)\cdot x_n+m(x_{n-1}-x_{n-2})+chx_{n-1} \\ \Longleftrightarrow\quad & m(x_n-x_{n-1})+h(kh+c)\cdot x_n-m(x_{n-1}-x_{n-2})-chx_{n-1}=h^2f_n \\ \Longleftrightarrow\quad & x_n\left(m+kh^2+ch\right)-x_{n-1}(2m+ch)+mx_{n-2}=h^2f_n \\ \Longleftrightarrow\quad & x_n=\frac{1}{m+kh^2+ch}\left(h^2f_n+x_{n-1}(2m+ch)-mx_{n-2}\right) \end{aligned}$$

Question 1 *Implémenter en Python le fonction f_omega permettant de créer une liste contenant l'ensemble des valeurs prises par la fonction f_n. On utilisera une boucle while. Les spécifications de la fonction sont les suivantes :*

Corrigé

```python
def f_omega(Tsimu,h,fmax,fsign):
    """
    Entrées :
        * Tsimu (flt) : temps de la simulation en seconde
        * h (flt) : pas de temps de a simulation
        * fmax (flt) : amplitude du signal (en Newton)
        * fsign (flt) : fréquence du signal (en Hertz)
    Sortie :
        * F (list) : liste des valeurs de la fonction
          f_n (t)= fmax sin (omega *t)
    """
    omega = 2*math.pi*fsign
    t=0
    F = []
    while t<Tsimu :
        F.append(fmax*math.sin(omega*t))
        t=t+h
    return F
```

Question 2 *En utilisant une boucle while, générer, en Python, les listes T et X contenant respectivement le temps de simulation et le déplacement de la masse mobile.*

Corrigé

```python
T=[0,h]
X=[0,0]
t=2*h
i=2
while t<Tsimu:
    T.append(t)
    X.append(alpha*F[i]+beta*X[i-1] +gamma*X(i-2))
    i=i+1
    t = t+h
```