Annexe

Résolution de systèmes d'équations différentielles

Utilisation de MATLAB

I. Introduction

L'analyse temporelle des systèmes physiques aboutit le plus souvent à la résolution d'équations ou de systèmes d'équations différentielles d'ordre n. Dans ce cadre, la résolution des problèmes mécaniques donne généralement naissance à des systèmes d'ordre 2 en temps. Ces équations sont, pour la plupart, impossibles à résoudre de façon analytique.

La puissance croissante des calculateurs associée à des logiciels évolués de calcul numérique (MATLAB, MAPLE, ...) nous permet de résoudre point par point ces systèmes d'équations.

II. Familiarisation à MATLAB

II.1. Présentation

MATLAB est un système interactif et convivial de calcul numérique et de visualisation graphique. Il utilise des fonctions d'analyse numérique, de calcul matriciel, de traitement du signal, de visualisation graphique 2D-3D, etc. Il peut être utilisé de façon interactive ou en mode programmation.

Son domaine d'utilisation et ses potentialités sont très vastes. L'objectif du T.P. n'est donc pas de vous apprendre MATLAB avec toutes ses possibilités mais de l'utiliser dans un but bien précis à savoir la résolution d'un système différentiel.

II.2. Notions de base

Dans MATLAB, nous utiliserons un seul type de données : **le type matrice,** qui est le plus courant dans l'utilisation de base du logiciel. Un scalaire est considéré comme une matrice carrée d'ordre 1, un vecteur comme une matrice colonne (n,1). Toutes les opérations usuelles se font matriciellement par défaut. L'utilisation du « . » devant un signe opératoire permet d'effectuer une opération terme à terme (A*B est le produit matriciel de A par B, alors que A.*B est la matrice de même taille que A et B, dont chaque terme est le produit des éléments de A et B. Le prime « ‘» est la transposée conjuguée, «.’ » est la transposée.

MATLAB peut fonctionner en mode ***interactif*** ou en mode ***programme*.** L'appel du logiciel nous donne l'invite : ‘’>>’’ qui nous permet de travailler.

Dans le ***mode interactif***, MATLAB, peut être considéré comme une super calculatrice scientifique. Il suffit de taper, après l'invite, la commande relative à l'opération à effectuer pour voir apparaître le résultat. L'ensemble des variables définies sont disponibles dans l'espace de travail (les commandes who et whos permettent de lister les variables en cours).

Pour des calculs plus élaborés qu'une simple opération, il est nécessaire de travailler en ***mode programme***. On génère alors sous éditeur de texte des fichiers qui permettent de stocker des séquences d'instructions. Ces fichiers, au format texte, sont suffixés .m et ils sont ensuite lus sous MATLAB qui exécute alors l'ensemble des instructions qu'ils contiennent. Il suffit pour cela de taper le nom du

fichier (sans le suffixe) après l'invite. Il existe deux types de fichiers : les **fichiers de commandes** ou scripts et les **fichiers de fonctions**.
Les **fichiers de commandes** : Un fichier de commandes est une séquence d'instructions MATLAB. La présentation des programmes est totalement libre et tous les caractères d'espacement, blancs, tabulations et interlignes, supplémentaires à ceux servant de séparateurs ne jouent qu'un rôle de présentation. Il est d'usage cependant de n'écrire qu'une seule instruction par ligne.

Pour créer ces fichiers de commandes, il suffit sous MATLAB, de cliquer sur *File* puis *New* ou *Open M-File*

Pour insérer des commentaires dans un programme, il suffit de commencer la ligne par le **caractère %.** Tout ce qui suit ce symbole, jusqu'à la fin de la ligne est considéré comme commentaire et n'est pas interprété par MATLAB. Il est conseillé d'utiliser ces lignes de commentaires chaque fois que nécessaire pour permettre une bonne lisibilité du programme.

La possibilité est laissée à l'utilisateur de suivre pas à pas le calcul se déroulant dans le programme car chaque commande implique l'affichage à l'écran du résultat de l'opération. Ceci peut cependant s'avérer inutile et handicapant pour la vitesse d'exécution. Ainsi, pour éviter de voir défiler à l'écran les calculs qui se déroulent dans le programme, on peut utiliser le **caractère ;** à la fin de la ligne de commande. La commande s'exécute mais l'affichage des résultats est supprimé.

Le **caractère ;** sert aussi à écrire en ligne un vecteur colonne :

x=[1 ;2 ;3 ;4] correspond à un vecteur colonne (4,1)

x=[1 2 3 4] correspond à un vecteur ligne (1,4)

Le **caractère ...** sert à continuer à la ligne suivante une ligne de calcul trop grande.

Il existe une aide en ligne pour les différentes commandes de MATLAB. Elle s'obtient par la commande **help** ou la commande **doc** suivie du nom de la commande. La résolution des deux problèmes traités ici ne nécessite pas l'utilisation de nombreuses commandes si ce n'est la commande ode23 ou ode45 et des commandes graphiques.

> Les variables sont locales à l'espace de travail. On peut définir des variables globales par la commande ***global***.

Les **fichiers de fonctions** : Les fichiers de fonctions fournissent une extensibilité à MATLAB. On peut ainsi créer des nouvelles fonctions spécifiques à notre travail qui auront le même statut que les autres fonctions prédéfinies de MATLAB. Les règles de base des fichiers de commande (ouverture, commentaires ...) sont bien sûr valables ici aussi.

> Les variables dans les fonctions sont par défaut locales mais peuvent devenir globales par la même commande ***global***.

***Définition et appel d'une fonction* :**

<liste des arguments de retour> = <**nom de la fonction**>(<liste des arguments d'appel>)

Dans un fichier de fonction, la première ligne commence par « function » puis déclare les arguments de retour, le nom de la fonction et les arguments d'appel sous la forme suivante :

function [arg1, arg2, arg3] = **nom_fonction**(argin1, argin2, argin3, argin4)

Dans cet exemple, la fonction nom_fonction a 4 arguments d'appel et 3 arguments de sortie à définir dans le corps de la fonction. Sans cette première ligne, le fichier correspond à un fichier script.

Remarque : Toutes les lignes commentaires qui suivent directement la ligne de déclaration sont affichées lors de la demande d'aide en ligne sur cette fonction (help **nom_fonction**).

Pour invoquer une fonction, il suffit de l'appeler par son nom suivi de ses arguments d'appel entre parenthèses et séparés par des virgules. Dans MATLAB, les arguments sont transmis par valeur lors de l'appel de la fonction. Leur modification éventuelle dans le corps de la fonction n'est pas visible à l'extérieur de celle-ci. Si une fonction doit modifier une variable, celle-ci doit être dans les arguments d'appel et de retour de la fonction.

***Exemple de création et d'appel de fonctions* :**

Fichier ***func1.m*** (obtenu sous éditeur de texte)

```
function y = func1(x,n)
x =x.^n
y = sum(x)
```

Remarques *: i) en faisant précéder les opérateurs par un point les opérations se font sur chaque composante de la matrice et non sur la matrice elle-même : ici élévation à la puissance n de chaque composante de la matrice*

ii) l'absence de ; à la fin des lignes de commande provoquera l'affichage de x et y au cours du déroulement du programme

Utilisation de la fonction :

```
>> x = [1  2  3  4  5] ↵        on définit l'argument d'entrée x
>> y = func1(x,3) ↵             on appelle la fonction x^3 et somme des x^3
x = 1  8  27  64  125           réponse : la variable x dans la fonction est élevée au
y = 225                         cube et y donne la somme

>>x ↵                           on rappelle la variable x de l'espace de travail
x =
     1  2  3  4  5              variable x non modifiée dans l'espace de travail
```

Fichier ***func2.m***

```
function [y,x] = func2(x,n)
x =x.^n
y = sum(x)
```

Utilisation de la fonction

```
>> x = [1  2  3  4  5] ↵        on définit l'argument d'entrée x
>> [y,x] = func2(x,3) ↵         on appelle la fonction x^3 et somme des x^3
x = 1  8  27  64  125           réponse : la variable x dans la fonction est élevée au
y = 225                         cube et y donne la somme

>>x ↵                           on rappelle la variable x de l'espace de travail
x =
     1  8  27  64  125          variable x modifiée dans l'espace de travail
```

Si l'espace de travail et plusieurs fonctions partagent la (les) même variable(s) de travail, il est alors possible de la (les) déclarer comme variable(s) globale(s) par la commande : ***global*** *nom1 nom2 nom3* au tout début du fichier de commande et de chaque fichier de fonctions concerné par cette (ces) variable(s). Chaque modification de la variable par une fonction sera prise en compte dans tous les fichiers où cette variable est déclarée globale. Avec cette même commande, on peut également définir

les constantes d'un problème en les définissant comme globales dans tous les fichiers qui les utilisent, et en leur affectant leur valeur dans le fichier programme.

Une stratégie plus propre en terme de gestion de mémoire consiste à définir une **structure** qui va contenir toutes les constantes du problème, et à passer cette structure en argument des fonctions lorsque c'est nécessaire. Par exemple, on peut définir la structure **data** suivante :

data.E=2.1e11 ;

data.nu=0.3 ;

data.rho=7800 ;

et passer "data" en argument d'une function.

II.3. La programmation dans MATLAB

Une application MATLAB se décompose en fonctions indépendantes les unes des autres. Chacune de ces fonctions est stockée dans un fichier .m. Une sous-fonction accessible uniquement depuis une fonction peut être définie dans le même fichier .m en suivant la même syntaxe. Elle ne sera cependant pas accessible depuis une autre fonction. Les informations sont transmises de fonction à fonction soit par l'intermédiaire d'arguments, soit à l'aide de variables globales. MATLAB dispose de structures de contrôle qui peuvent être utilisées comme dans tout langage de programmation. Il dispose également de puissants outils de visualisation graphique et de débuggage. Notre problème ne nécessite pas de tels développements.

III. Résolution de systèmes d'équations différentielles

III.1. Équation différentielle d'ordre 1

Soit à retrouver la fonction y = f(x) vérifiant l'équation différentielle suivante :

$$\frac{dy}{dx} = f(x, y)$$

On va estimer d'une façon plus ou moins précise suivant les méthodes utilisées l'accroissement Δy entre les abscisses x_i et x_{i+1}.

a) Méthode d'Euler

Cette méthode simple mais imprécise estime l'accroissement Δy entre les abscisses x_i et x_{i+1} par un développement de Taylor d'ordre 1 :

$$y_{i+1} = y_i + h\frac{\partial y}{\partial x}_{\{x_i, y_i\}}$$

où le pas h doit être très petit, ce qui multiplie le nombre de pas et rallonge le temps de calcul. MATLAB n'utilise pas cette méthode pour la résolution des équations différentielles.

b) Méthodes de Runge-Kutta

Ce sont les deux méthodes retenues par MATLAB pour les fonctions **ode23** et **ode45** qui permettent de résoudre les systèmes d'équations différentielles.

On distingue la méthode de **Runge-Kutta d'ordre 2** et **d'ordre 4**. Pour la première, l'approximation d'Euler est améliorée en estimant Δy par un développement de Taylor d'ordre 2 :

$$y_{i+1} = y_i + \left(x_{i+1} - x_i\right)\left(\frac{\partial y}{\partial x}\right)_{\left\{\frac{x_i+x_{i+1}}{2}, y_i + \frac{(x_{i+1}-x_i)}{2}\frac{\partial y}{\partial x}(x_i,y_i)\right\}}$$

Ceci revient à évaluer la pente de la fonction f au milieu de l'incrément h c'est à dire en $\frac{x_{i+1}+x_i}{2}$ (au lieu de x_i dans la méthode précédente) puis à l'étendre à la largeur du pas.

La **méthode de Runge-Kutta d'ordre 4** donne une meilleure estimation car on utilise alors un développement de Taylor d'ordre 4. Ceci revient à calculer la pente de la fonction F au début de l'incrément (en x_i), à l'évaluer au milieu (en $(x_i + x_{i+1})/2$) et à la fin (en x_{i+1}) de l'incrément. On pondère ensuite la contribution de chacun des termes pour évaluer l'incrément Δy.

$$y_{i+1} = y_i + \frac{x_{i+1} - x_i}{6}\left(f1 + 2f2 + 2f3 + f4\right)$$

avec

$$f1 = \frac{\partial y}{\partial x}_{\{x_i, y_i\}}$$

$$f2 = \frac{\partial y}{\partial x}_{\left\{\frac{x_i+x_{i+1}}{2}, y_i + \frac{x_{i+1}-x_i}{2} f1\right\}}$$

$$f3 = \frac{\partial y}{\partial x}_{\left\{\frac{x_i+x_{i+1}}{2}, y_i + \frac{x_{i+1}-x_i}{2} f2\right\}}$$

$$f4 = \frac{\partial y}{\partial x}_{\left\{x_{i+1}, y_i + \frac{x_{i+1}-x_i}{2} f3\right\}}$$

Grâce à cette méthode, on peut travailler avec un pas h plus grand que pour les autres méthodes. Ce qui permet de réduire le temps de calcul et d'approcher de plus près la réponse analytique.

Les fonctions **ode23** et **ode 45** utilisent respectivement les méthodes de Runge-Kutta d'ordre 2 et 4 avec un pas h variable (Runge-Kutta-Fehlberg).

III.2. Systèmes d'équations différentielles d'ordre 1

On généralise la présentation précédente à un ensemble de n équations à n variables.

Soit, pour déterminer les valeurs approchées de u(x) et v(x) vérifiant le système différentiel suivant :

$$\begin{cases} \dfrac{\partial u}{\partial x} = f(x,u,v) \\ \dfrac{\partial v}{\partial x} = g(x,u,v) \end{cases}$$

on utilisera par exemple la méthode de Runge-Kutta d'ordre2 :

$$u_{i+1} = u_i + (x_{i+1} - x_i)\frac{\partial f}{\partial x}\left\{\frac{x_i + x_{i+1}}{2}, u_i + \frac{(x_{i+1}-x_i)}{2}\frac{\partial u}{\partial x}_{\{x_i, y_i\}}, v_i + \frac{(x_{i+1}-x_i)}{2}\frac{\partial v}{\partial x}_{\{x_i, v_i\}}\right\}$$

$$v_{i+1} = v_i + (x_{i+1} - x_i)\frac{\partial g}{\partial x}\left\{\frac{x_i + x_{i+1}}{2}, u_i + \frac{(x_{i+1}-x_i)}{2}\frac{\partial u}{\partial x}_{\{x_i, y_i\}}, v_i + \frac{(x_{i+1}-x_i)}{2}\frac{\partial v}{\partial x}_{\{x_i, v_i\}}\right\}$$

On peut également procéder de façon analogue pour le développement à l'ordre 4.

III.3. Système d'équations différentielles d'ordre n

Une équation différentielle d'ordre n peut toujours être exprimée sous la forme d'un système de n équations différentielles d'ordre 1 par un changement de variables approprié. De même, un système de p équations différentielles d'ordre n s'exprimera sous la forme d'un système de n*p équations différentielles d'ordre 1. Un exemple de résolution d'équation différentielle d'ordre 2 est présenté ci-dessous.

a) Problème du pendule libre amorti

Soit à résoudre l'équation suivante :

$$\ddot{z} + 2a\omega_0 \dot{z} + \omega_0^2 z = 0 \qquad \text{avec les C.I. suivantes :} \qquad \begin{matrix} \dot{z}(0) = 0 \\ z(0) = 1. \end{matrix}$$

En posant : $z = z_1$

$\dot{z} = \dot{z}_1 = z_2$,

on obtient le système suivant : $\begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = -\omega_0^2 z_1 - 2a\omega_0 z_2 \end{cases}$

que l'on peut alors écrire sous la forme matricielle suivante :

zp = A b

où les composantes de b sont des fonctions des z_i.

La syntaxe des fonctions **ode23** et **ode45** est identique :

[x,y] = ode23(@fich, xspan, y_0)

[x,y] = ode45(@fich, xspan, y_0)

fich : chaîne de caractère représentant le nom du fichier .m (fich.m) dans lequel sera défini le système différentiel

xspan : vecteur spécifiant l'intervalle d'intégration. xspan = [x_0 x_{final}] avec x_0, x_{final} : valeurs initiale et finale de la variable x

y_0 : valeur initiale de la fonction à intégrer. y peut être un scalaire dans le cas d'une équation différentielle d'ordre1, un vecteur à n composantes dans le cas d'une équation différentielle d'ordre n ou un vecteur à n*p composantes dans le cas d'un système de p équations différentielles d'ordre n.

Le système d'équations doit être représenté dans un fichier fonction (ici par exemple pendu.m) sous la forme d'un vecteur colonne zp (contenant les dérivées de z) où chaque ligne représente une équation du système :

zp = A b

pendu.m

```
function zp = pendu(t,z)
% fichier contenant l'équa. diff. (sous forme matricielle) d'un pendule libre amorti
global a w0
mat = [    0              1   ;       -w0 ^2      -2*a*w0  ];
zp = mat * z ;
```

Ce fichier peut maintenant être appelé par un fichier procédure (par exemple : calcul.m)

calcul.m

```
% résolution d'un pendule libre amorti
global a w0
% définition des constantes du système
a = 0.01 ;
w0 = 200 ;
% définition du temps d'observation
ti = 0 ;
tf = 5 ;
tspan = [ti tf] ;
% définition des conditions initiales
z0 = [1 ;  0] ;
% intégration du système différentiel
[t,z]= ode45(@pendu, tspan, z0) ;
```

Il suffira maintenant de taper **calcul** après l'invite de MATLAB pour que le programme s'exécute.

b) Sortie des résultats

De multiples commandes graphiques sont disponibles sous MATLAB. Les plus utiles sont plot, axis, xlabel, ylabel ...

Exemples :

Pour tracer l'évolution de z ou de $\dot{z}$ avec le temps, on demandera : **plot**(t, z(:,1)) ou plot(t, z(:,2)). La commande z(:,1) permet de récupérer tous les termes de la première colonne du vecteur z.

Pour modifier les échelles sur les axes, on utilisera la commande **axis**. Dans tous les cas, aidez vous de la commande **help**.

Une nouvelle figure est obtenue par la commande **figure**. La commande figure(ii) active la figure numéro ii pour le tracé.

Pour tracer plusieurs courbes, on utilisera soit la commande **hold on** entre deux tracés (**hold off** pour arrêter de tracer par-dessus la figure courante), soit la fonction plot en combinant les tracés : plot(t,z(:,1),t,z(:,2))

Toutes les caractéristiques des figures peuvent être modifiées à la souris via les menus des figures, ou en ligne de commande :

```
hp=plot(t,z(1),'linewidth',2, 'color','k') ;
```

et en utilisant la fonction set : set(hp,'color','r') ;

L'ensemble des propriétés modifiables est obtenu par la commande get(hp).