TP - Double pendule par résolution numérique

Attention : Ce sujet comprend deux partie, une partie théorique (questions 1,2,3) et une partie numérique. La partie théorique est indépendante, que vous pouvez terminer ultérieurement. Après une bonne lecture de celle-ci, vous êtes recommandés à consacrer votre séance à la **partie numérique**.

Introduction

Ce TP a pour but de **résoudre par intégration numérique explicite en temps** les équations du mouvement issues de la **méthode de Lagrange** pour un système dynamique constitué d'un double pendule mobile, représenté par Fig. 1. Les équations gouvernant la dynamique du système sont des équations différentielles ordinaires (ODE) d'ordre 2. La résolution numérique de ces équations vous permet d'observer l'évolution de la dynamique du système qui mène vers un **comportement chaotique**.

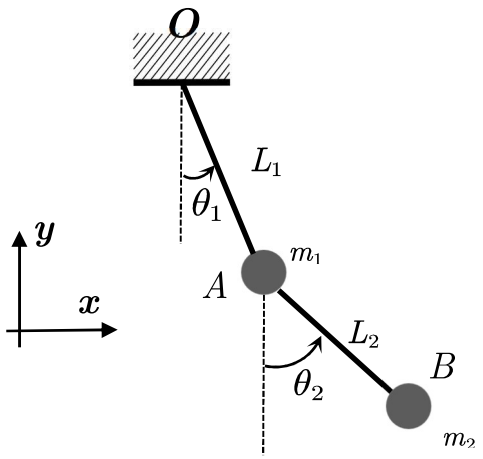

Figure 1. On désigne par **double pendule**, un pendule à l'extrémité duquel est attaché un autre pendule. Le double pendule est constitué de **2 tiges** de longueurs L_1 et L_2, et de **2 masses ponctuelles** m_1 et m_2. Les tiges elles-mêmes sont considérées comme rigides et sans masse. On note θ_1 et θ_2 pour désigner les angles que représentent les tiges par rapport à l'axe vertical ($-y$) du plan, c'est-à-dire $\theta_1 = \measuredangle(-y)(OA)$ et $\theta_2 = \measuredangle(-y)(AB)$. L'origine du repère se trouve au point O.

Afin d'étudier le fonctionnement du système et de mettre en œuvre sa résolution, on entreprend le travail en 4 phases :

1. Étudier les équations du mouvement du système (équations de Lagrange).
2. Prendre en main de la méthode **Runge-Kutta** pour l'intégration numérique, et de sa mise en œuvre dans Matlab. Deux codes exemples vous seront fournis pour des équations différentielles d'ordre 1, et 2.
3. Adapter cette implémentation fournie pour la résolution des équations de Lagrange étudiées à la partie 1.
4. Lancer les calculs et interpréter les résultats observés.

Partie I – Equations de Lagrange

Avec les notations dans **Figure 1**, on exprime les coordonnées des deux masses m_1 et m_2 en fonction des angles de rotation θ_1 et θ_2 :

$$\begin{aligned} x_1 &= L_1 \sin\theta_1 \\ y_1 &= -L_1 \cos\theta_1 \\ x_2 &= L_1 \sin\theta_1 + L_2 \sin\theta_2 \\ y_2 &= -L_1 \cos\theta_1 - L_2 \cos\theta_2 \end{aligned}$$

(Question 1) Démontrer que **l'énergie potentielle** du système s'écrit :

$$V = -(m_1 + m_2) g L_1 \cos\theta_1 - m_2 g L_2 \cos\theta_2$$

(Question 2) Démontrer que **l'énergie cinétique** du système s'écrit :

$$T = \frac{1}{2} m_1 L_1^2 \left(\frac{d\theta_1}{dt}\right)^2 + \frac{1}{2} m_2 \left[L_1^2 \left(\frac{d\theta_1}{dt}\right)^2 + L_2^2 \left(\frac{d\theta_2}{dt}\right)^2 + 2 L_1 L_2 \frac{d\theta_1}{dt} \frac{d\theta_2}{dt} \cos(\theta_1 - \theta_2) \right]$$

(Question 3) En déterminant le **Lagrangien** du système avec les formules du cours, démontrer que l'**équation du mouvement** (équations de Lagrange) par rapport à θ_1 s'écrit :

$$(m_1 + m_2) L_1 \frac{d^2\theta_1}{dt^2} + m_2 L_2 \frac{d^2\theta_2}{dt^2} \cos(\theta_1 - \theta_2) + m_2 L_2 \left(\frac{d\theta_2}{dt}\right)^2 \sin(\theta_1 - \theta_2) + g(m_1 + m_2) \sin\theta_1 = 0$$

et que celle par rapport à θ_2 s'écrit :

$$m_2 L_2 \frac{d^2\theta_2}{dt^2} + m_2 L_1 \frac{d^2\theta_1}{dt^2} \cos(\theta_1 - \theta_2) - m_2 L_1 \left(\frac{d\theta_1}{dt}\right)^2 \sin(\theta_1 - \theta_2) + g m_2 \sin\theta_2 = 0$$

Ces équations sont des équations différentielles **d'ordre 2**. Nous les transformons en équation différentielle **d'ordre 1** pour leur implémentation numérique. Pour cela, nous utilisons les approximations suivantes :

$$\begin{aligned} u_1 &= \theta_1(t) \\ u_2 &= \theta_1'(t) \\ v_1 &= \theta_2(t) \\ v_2 &= \theta_2'(t) \end{aligned}$$

Notre système des équations de Lagrange devient :

$$(m_1+m_2)L_1\frac{du_2}{dt}+m_2L_2\frac{dv_2}{dt}\cos(u_1-v_1)+m_2L_2(v_2)^2\sin(u_1-v_1)+g(m_1+m_2)\sin u_1=0$$

$$m_2L_2\frac{dv_2}{dt}+m_2L_1\frac{du_2}{dt}\cos(u_1-v_1)-m_2L_1(u_2)^2\sin(u_1-v_1)+gm_2\sin v_1=0$$

et

$$\frac{du_1}{dt}=u_2(t)$$

$$\frac{dv_1}{dt}=v_2(t)$$.

En introduisant les variables intermédiaires (changement de variables)

$$\begin{aligned}
a &= (m_1+m_2)L_1\\
b &= m_2L_2\cos(u_1-v_1)\\
c &= m_2L_1\cos(u_1-v_1)\\
d &= m_2L_2\\
e &= -m_2L_2(v_2)^2\sin(u_1-v_1)-g(m_1+m_2)\sin u_1\\
f &= m_2L_1(u_2)^2\sin(u_1-v_1)-gm_2\sin v_1
\end{aligned}$$,

le système des équations de Lagrange devient :

$$a\frac{du_2}{dt}+b\frac{dv_2}{dt}=e \qquad (1)$$

$$c\frac{du_2}{dt}+d\frac{dv_2}{dt}=f \qquad (2)$$

$$\frac{du_1}{dt}=u_2(t) \qquad (3)$$

$$\frac{dv_1}{dt}=v_2(t) \qquad (4)$$.

Les équations (1) et (2) permettent d'exprimer $\frac{du_2}{dt}$ et $\frac{dv_2}{dt}$ en fonction des variables intermédiaires $a,\ b,\ c,\ d,\ e,\ f$, c'est-à-dire :

$$\frac{du_2}{dt}=\frac{ed-bf}{ad-cb} \qquad (5)$$

$$\frac{dv_2}{dt}=\frac{af-ce}{ad-cb} \qquad (6)$$.

Les équations (3)(4)(5)(6) sont les équations différentielles d'ordre 1, que vous allez implémenter en utilisant la résolution «Runge-Kutta». Cette méthode de résolution est décrite dans la partie suivante.

Partie II – Algorithme Runge-Kutta et son implémentation dans Matlab

Cette partie présente l'algorithme Runge-Kutta qui permet la résolution par intégration numérique explicite en temps, des équations différentielles ordinaires. Cette méthode a été nommée en l'honneur de Carl Runge et Martin Wilhelm Kutta, mathématiciens allemands qui l'ont élaboré en 1901. Il s'agit d'une méthode de résolution itérative, pour laquelle une première estimation de la solution est utilisée pour calculer une seconde estimation, plus précise, et ainsi de suite.

Plusieurs versions de cette méthode existent. Celle la plus souvent utilisée repose sur un algorithme de résolution d'ordre 4, que l'on appelle Runge-Kutta d'ordre 4, ou RK4.

Considérons le problème suivant :

$$y' = f(t, y), \quad y(t_0) = y_0$$

La méthode RK4 est donnée par l'équation :

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

L'idée est que la valeur suivante (y_{n+1}) est approchée par la somme de la valeur actuelle (y_n) et du produit de la taille de l'intervalle (h) par la pente estimée. La pente est obtenue par une moyenne pondérée de pentes :

$$k_1 = f(t_n, y_n)$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$
$$k_4 = f(t_n + h, y_n + hk_3)$$

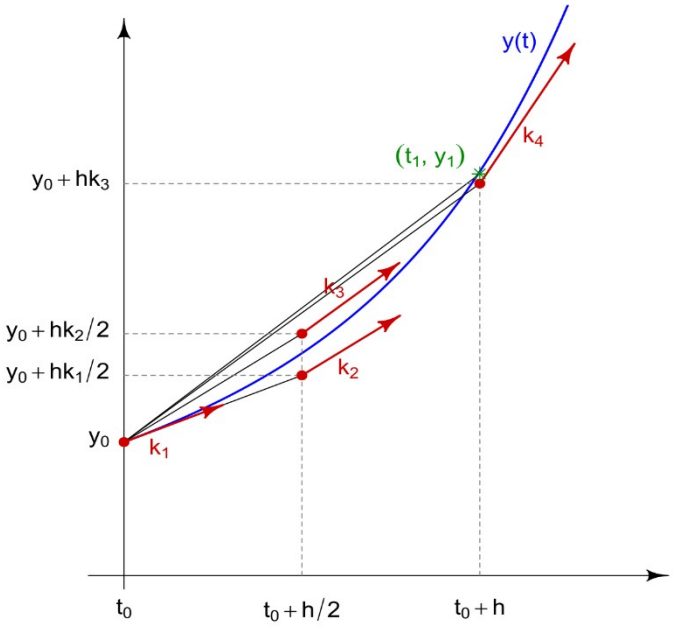

Figure 2. Représentation graphique de l'algorithme RK4 pour 1 pas de temps d'intégration. On distingue dans ce pas de temps les 4 pentes à calculer qui sont k_1 à k_4

- k_1 est la pente au début de l'intervalle ;
- k_2 est la pente au milieu de l'intervalle, en utilisant la pente k_1 pour calculer la valeur de y au point ($t_n + h/2$) par le biais de la méthode d'Euler ;
- k_3 est de nouveau la pente au milieu de l'intervalle, mais obtenue cette fois en utilisant la pente k_2 pour calculer y ;
- k_4 est la pente à la fin de l'intervalle, avec la valeur de y calculée en utilisant k_3.

Dans la moyenne des quatre pentes, un poids plus grand est donné aux pentes au point milieu.

Ces formules sont aussi valables pour des fonctions vectorielles.

Dans ce TP, on a besoin de savoir comment définir une fonction Matlab. Vous trouverez ci-dessous un exemple.

```
% créer un fichier incremente.m  en mettant dedans le code ci-dessous
% ici, une fonction «incremente» est definie.

function [v1] = incremente(v0)  % entete du fichier pour créer une fonction
                                % v0 est l'entrant, v1 est le sortant
v1 = v0 + 1 ;                   % definir ici l'operation de votre fonction
```

```
% créer un fichier exemple.m  en mettant dedans le code ci dessous

A = 1;               % chiffre a incrementer
B = incremente(A);   % appeler la fonction incremente en passant A comme argument
                     % et stocker le resultat dans B
B                    % afficher B  (B devrait afficher 2)
```

Vous trouverez deux exemples qui montrent l'implémentation de la méthode RK4. Pour mieux comprendre les codes, vous pouvez les recopier et les exécuter ligne par ligne dans Matlab.

Exemple 1

La résolution de l'équation différentielle d'ordre 1. C'est une **fonction scalaire**.

$$\frac{dy}{dt} = y \cdot (\sin t)^2$$

```
%-------------------------------------------------------
%                       ODE d'ordre 1
%-------------------------------------------------------
clc;
clear all;

dt = 0.1;                % time step pour l'integration explicite

t  = 0:dt:5;             % maillage selon x pour la discretisation en temps
y  = zeros(1,length(t)); % initialiser le tableau pour recevoir la solution y

y(1) = 0.8; % condition initiale

% Definir l'equation a resoudre, ici l'equation y' = sin(t)^2 *y,
% Fonction fsin a implementer dans un autre fichier 'fsin.m'


for i=1:length(t)-1
    k1 = fsin( t(i)          ,  y(i)             );
    k2 = fsin( t(i) + 0.5*dt ,  y(i) + 0.5*dt*k1 );
    k3 = fsin( t(i) + 0.5*dt ,  y(i) + 0.5*dt*k2 );
    k4 = fsin( t(i) + dt     ,  y(i) + k3*dt     );
    y(i+1) = y(i) + (1/6)*(k1 + 2*k2 + 2*k3 + k4)*dt;
end

figure            % Tracer la solution
plot(t,y,'-o')
title('Solution of equation dydt = y*sin(t)^2 using Runge-Kutta method (RK4)');
xlabel('Time t');
ylabel('Solution y');
legend('y')


% ------------------------------------------------
% fonction fsin
% Cette fonction est sauvegardee dans un fichier fsin.m separe

function [dydt] =  fsin(t,y)
% fonction:  dydt = sin(t)^2*y;
dydt = sin(t)^2*y;
```

Exemple 2

La résolution de l'équation van der Pol. C'est une équation différentielle d'ordre 2.

$$\frac{d^2 y_1}{dt^2} - (1 - y_1^2)\ \frac{dy_1}{dt} + y_1 = 0$$

L'algorithme RK4 ne permet pas de résoudre directement une ODE d'ordre 2. Mais nous pouvons introduire une variable intermédiaire $y_2 = \frac{dy_1}{dt}$, ce qui transforme l'équation van der Pol en un système composé de deux équations d'ordre 1 :

$$\frac{dy_1}{dt} = y_2 \qquad (i)$$

$$\frac{dy_2}{dt} = (1 - y_1^2)\ y_2 - y_1 \qquad (ii)$$

Le système (i)(ii) sera résolu par RK4 comme une **fonction vectorielle**.

```
%-----------------------------------------------------------
%                       ODE d'ordre 2
%-----------------------------------------------------------
clear all
clc

dx = 0.05;                  % time step pour l'integration explicite
x  = 0:dx:20;               % maillage selon x pour la discretisation en temps
y  = zeros(2,length(x));    % initialiser le tableau pour recevoir la solution y

y(1,1) = 2;     % conditions initiales sur y1
y(2,1) = 0;     % conditions initiales sur y2

% Definir les equations a resoudre. ici definies dans le
% fichier fonction 'fvdp.m' qui implemente les equations van der Pol:
%    y' =   [ y1' ]  =  [ y2                  ]
%           [ y2' ]     [ (1 - y1^2)*y2 -y1   ]

for i=1:length(x)-1
    k1 = fvdp ( x(i)           , y(:,i)             );
    k2 = fvdp ( x(i) + 0.5*dx , y(:,i) + 0.5*dx*k1 );
    k3 = fvdp ( x(i) + 0.5*dx , y(:,i) + 0.5*dx*k2 );
    k4 = fvdp ( x(i) +     dx , y(:,i) +     dx*k3 );
    y(:,i+1) = y(:,i) + (1/6)*(k1 + 2*k2 + 2*k3 + k4)*dx;
end

figure
plot(x,y(1,:),'-o',   x,y(2,:),'-o')
title('Solution of van der Pol Equation (\mu = 1) using Runge-Kutta method (RK4)');
xlabel('Time t');
ylabel('Solution y');
legend('y_1','y_2')


% fonction fvdp ------------------------------------------------

function [dydt] =  fvdp(t,y)
% Definir les equations a resoudre, ici :
%    y' =   [ y1' ]  =  [ y2                  ]
%           [ y2' ]     [ (1 - y1^2)*y2 -y1   ]
dydt = [  y(2);
         (1-y(1)^2)*y(2) - y(1)];
```

Partie III – Implémentation de RK4 pour le problème de double pendule

(Question 4) En utilisant les deux précédents exemples (Partie II), réaliser un code Matlab pour résoudre le système d'équations du mouvement pour notre double pendule avec la méthode RK4. Rappelons que les équations à résoudre sont les Eq.(3,4,5,6), obtenues à la fin de la Partie I.

Avec votre programme, visualiser les trajectoires de votre pendule (Les point A, B). De même, visualiser l'animation de votre simulation. Pour la visualisation, vous pouvez utiliser le code qui vous est fourni à la dernière page du sujet. Joindre à votre compte rendu la capture d'écran de vos trajectoires du système.

Les paramètres nécessaires à la configuration de votre programme, par exemple les longueurs L_1 et L_2, les 2 masses m_1 et m_2, ainsi que les valeurs initiales de θ_1 et θ_2 (conditions initiales), vous seront communiqués par votre enseignant lors des TP.

Vous devez également joindre au compte rendu vos codes réalisés.

Partie IV – Observation du chaos et interprétation du résultat

(Question 5) En modifiant très légèrement les conditions initiales, par exemple en multipliant θ_1 initiale par un petit coefficient de 1.001, faire une autre simulation.

Superposer les trajectoires des animations issues de cette simulation avec celles issues de la simulation précédente (celles de Question 4). Pouvez-vous observer le chaos ? Décrire votre observation et joindre à votre rapport les images montrant les trajectoires de votre système.

Pour afficher les résultats superposés, il suffit de tracer **dans une même figure** les résultats issus des deux simulations. Pour cela, vous devez comprendre le code de post-traitement qui vous est fourni.

Code à compléter pour résoudre le problème du double pendule.

```
%-----------------------------------------------------------
%         Resolution systeme OED double pendule
%-----------------------------------------------------------
clear all
clc

… …
Code à réaliser.
… …

% les coordonnees du point A (m1) peuvent etre calculees par
x1 =  L1*sin(y(1,:)) ;
y1 = -L1*cos(y(1,:)) ;

% les coordonnees du point B (m2) peuvent etre calculees par
x2 =  L1*sin(y(1,:)) + L2*sin(y(3,:));
y2 = -L1*cos(y(1,:)) - L2*cos(y(3,:));

% ===========================================================
% Les codes ci-dessous sont les outils vous permettant de post
% traiter les resultats et tracer les trajectoires et animation
%
%               OUTILS DE VISUALISATION
%
% ===========================================================

% tracer trajectoires des deux masses
figure(1)
hold on
FG1 = plot(x1,y1);      FG2 = plot(x2,y2);
LB1 = 'Point A (m1)';  LB2 = 'Point B (m2)';
legend([FG1,FG2], LB1, LB2);
xlabel('X ','fontSize',12);
ylabel('Y ','fontSize',12);
title('Trajectoires des deux masses','fontsize',12)


% creer animation de la simulation
figure(2)
nfr=0;
for i =1:size(y2,2)
  plot(0, 0,'.','markersize',25);        % point O
  hold on
  plot(x1(i),y1(i),'.','markersize',25); % point A
  plot(x2(i),y2(i),'.','markersize',25); % point B
  hold off
  axis([ -(L1+L2) L1+L2 ...
         -(L1+L2) L1+L2 ]);         % cadrage figure
  line([0 x1(i)], [0 y1(i)],'Linewidth',2);         % tige OA
  line([x1(i) x2(i)], [y1(i) y2(i)],'linewidth',2); % tige AB
  xlabel('X'); ylabel('Y');
  title('Animation de la cinematique modele')
  fh = figure(2);
  set(fh, 'color', 'white');
  ff=getframe;
  nfr=nfr+1;
end
```