Utilisation de fonctions avancées (corrigé)

L'objectif de cette feuille d'exercice est de familiariser les élèves avec le maniement de deux types de fonctions très utiles dans beaucoup de domaines de la physique et des mathématiques appliquées. Il s'agit d'utiliser les intégrateurs temporels de MATLAB afin de pouvoir simuler des comportements dynamiques, puis de se familiariser avec les fonctions d'optimisation de MATLAB.

1 Intégration d'équations différentielles ordinaires

1.1 Le pendule non-amorti

On commence par étudier la dynamique linéarisée du pendule : $\ddot{\theta} + \omega_0^2 \sin\omega = 0$, avec $\omega_0 = 1$. L'utilisation de la fonction `ode23` nécessite de créer deux programmes, le premier permettant de rentrer les paramètres importants (typiquement : temps d'intégration, options, etc.), et le second contient explicitement la dynamique, soit par exemple :
Première fonction : `pendulin.m` :

```
function [T,X] = pendulin(X0,Tf);

TSPAN = [0 Tf];

[T,X] = ode23(@pendulinODE,TSPAN,X0);
```

Seconde fonction, appelée par le premier programme : `pendulinODE.m` :

```
function xdot = pendulinODE(T,x);

xdot = [0 1; -1 0]*x;
```

L'appel du premier programme permet de régler la condition initiale : `X0`, ainsi que le temps d'intégration `Tf`. On forme le vecteur `TSPAN = [0 Tf]`, indiquant que l'on intègrera l'équation du temps 0 au temps `Tf`. Une autre solution possible pour définir l'échelle de temps est de donner un vecteur complet des instants de calcul, soit du type : `TSPAN = [T0 T1 ... Tf]` (pour des instants choisis à l'avance), soit du type : `TSPAN = [T0 :dT :Tf]`, avec un instant initial, un pas de temps fixé et un instant final. Cependant, les algorithmes préprogrammés de MATLAB sont optimisés pour une utilisation avec un pas de temps non fixé. Le programme `ode23` se charge lui-même, à chaque itération, de calculer le pas de temps optimal, garantissant certaines tolérances définies par l'utilisateur.

Le calcul de la solution temporelle, avec les conditions initiales $\theta_0 = .1$ et $\dot{\theta}_0 = 0$, et pour un temps d'intégration `Tf=1000`, s'obtient par :

```
[T,X] = pendulin([.1 0],1000);
```

Le vecteur `X` contient le déplacement et la vitesse. On peut représenter par exemple le déplacement en fonction du temps :

```
plot(T,X( :,1),'-b')
set(gca,'FontSize',18)
```

```
xlabel('Temps(s)')
ylabel('Deplacement (m)')
xlim([0 Tf/2])
grid on
```

On obtient alors la figure 1.

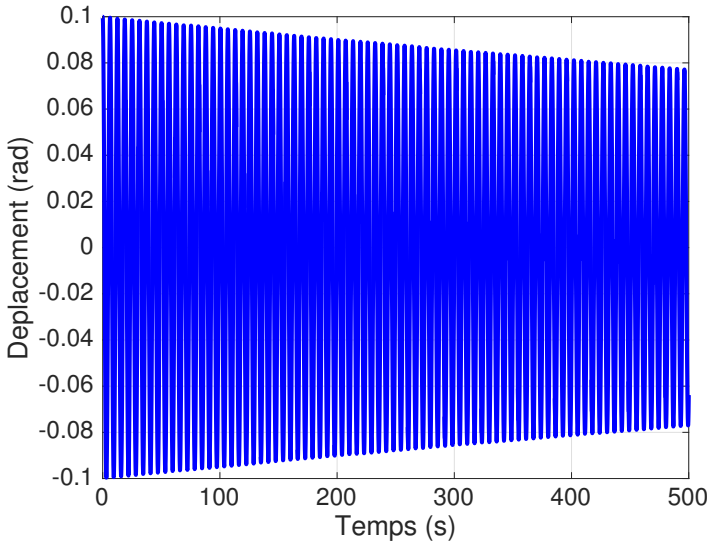


FIGURE 1 – Solution numérique (déplacement) de l'équation du pendule linéaire avec l'intégrateur temporel `ode23` - Echelle logarithmique selon y.

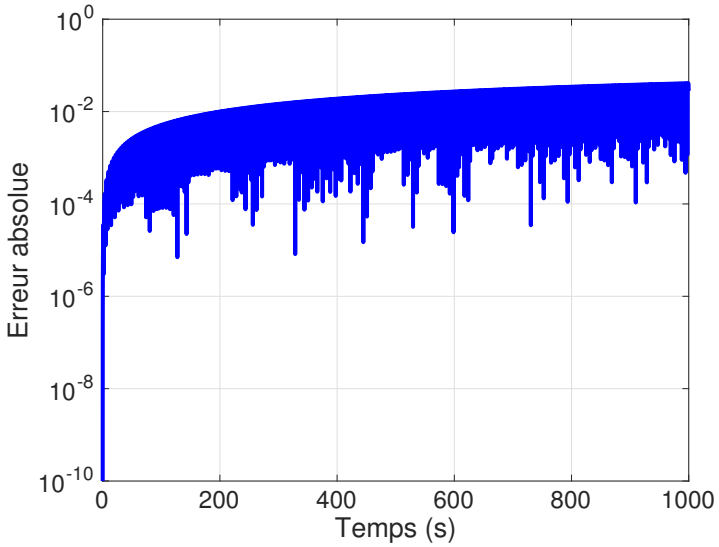


FIGURE 2 – Erreur absolue entre la solution analytique de l'équation du pendule linéaire et la solution numérique avec l'intégrateur temporel `ode23` - Echelle logarithmique selon y.

On constate immédiatement un problème ! Alors que la dynamique initiale est conservative, la solution numérique montre une décroissance de l'énergie. On peut la comparer avec la solution analytique connue sur la même figure, ou bien représenter l'erreur commise par l'intégrateur numérique. Par exemple :

```
» figure
» semilogy(T, abs(.1*cos(T)-X( :,1)))
```

La figure 2 présente cette erreur.

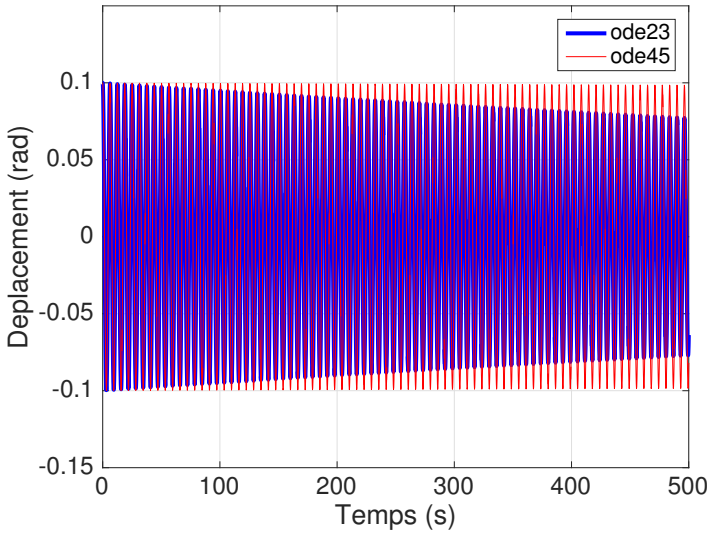


FIGURE 3 – Solution numérique (déplacement) de l'équation du pendule linéaire avec les intégrateurs temporels `ode23` et `ode45` - Echelle logarithmique selon y.

Afin de remédier à cette erreur, une première solution consiste à changer l'intégrateur numérique. L'algorithme programmé dans la fonction `ode23` reprend la méthode de Runge-Kutta du second ordre. La même méthode existe avec un développement au quatrième ordre, il s'agit de `ode45`. La comparaison avec la solution trouvée par la fonction `ode23` montre que la fonction `ode45` permet d'obtenir un meilleur comportement (voir figure 3). Cependant, aux temps longs, les erreurs existent toujours (voir figure 4, courbe rouge). Il faut alors changer les tolérances d'erreur de l'intégrateur `ode45`. Deux tolérances sont réglables : une tolérance relative (`RelTol`, valeur par défaut : 10^{-3}), qui contrôle la précision de la solution calculée à chaque pas, et une tolérance absolue (`AbsTol`, valeur par défaut : 10^{-6}), qui contrôle le comportement de la solution aux petites valeurs, lorsque l'on est proche de zéro. Afin d'augmenter la précision de la solution numérique, il faut changer ces valeurs, ce qui se fait dans la première fonction : `pendulin.m` :

```
function [T,X] = pendulin(X0,Tf);

TSPAN = [0 Tf];

options = odeset('RelTol',1e-7,'AbsTol',1e-8);

[T,X] = ode45(@pendulinODE,TSPAN,X0,options);
```

La figure 4 montre l'évolution des erreurs pour les trois méthodes utilisées.

Pour intégrer le problème complet non linéaire, il faut modifier les deux programmes de départ. Attention, on ne peut plus écrire la dynamique sous forme matricielle, il faut donc séparer chaque composante, ce qui donne, par exemple :

```
function [T,X] = pendule(X0,Tf);

TSPAN = [0 Tf];

options = odeset('RelTol',1e-7,'AbsTol',1e-8);

[T,X] = ode45(@penduleODE,TSPAN,X0,options);

--------------------------------

function xp = penduleODE(t,x);
```

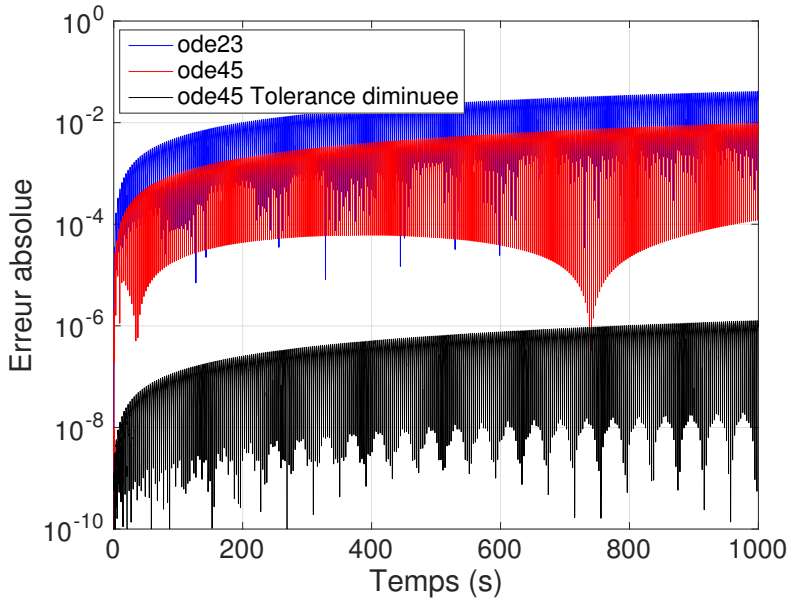


FIGURE 4 – Erreur absolue entre la solution analytique de l'équation du pendule linéaire et la solution numérique avec les intégrateurs temporels `ode23`, `ode45` et `ode45` avec une précision accrue (Tolérance diminuée) - Echelle logarithmique selon y.

```
xp(1)=x(2);
xp(2)=-sin(x(1));
xp=xp(:);
```

Avec une condition initiale éloignée de X(0)=0, on peut observer la déformation de la solution temporelle due à la non linéarité (voir figure 5 pour $X(0) = \frac{\pi}{2}$).

1.2 Equation de Van der Pol

Par rapport au cas précédent, on souhaite pouvoir, dans le cas de l'équation de Van der Pol, passer le paramètre μ à la dynamique à intégrer. Les paramètres à passer dans les intégrateurs type `ode45` se mettent après les options. Ce qui donne, par exemple, pour intégrer l'équation de Van der Pol :

```
function [T,X] = vdp(X0,Tf,mu);

TSPAN = [0 Tf];

options = odeset('RelTol',1e-4,'AbsTol',1e-6);

[T,X] = ode45(@vdpODE,TSPAN,X0,options,mu);

---------------------------------

function xdot = vdpODE(t,x,mu);

xdot(1)=x(2);
xdot(2)=-x(1)+mu*(1-x(1)^2)*x(2);
xdot=xdot(:);
```

Pour calculer trois trajectoires correspondant aux cas : $\mu = 1$, $\mu = 3$ et $\mu = 6$, il suffit alors de rentrer en ligne de commande :

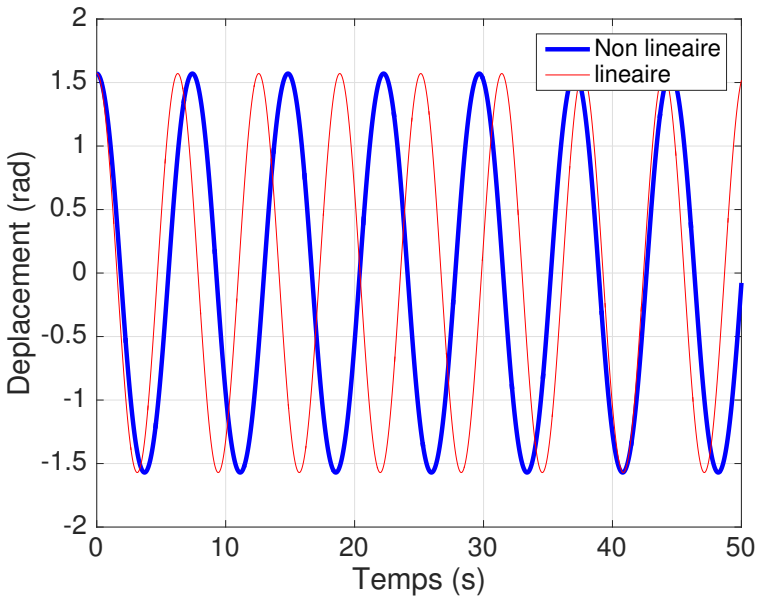
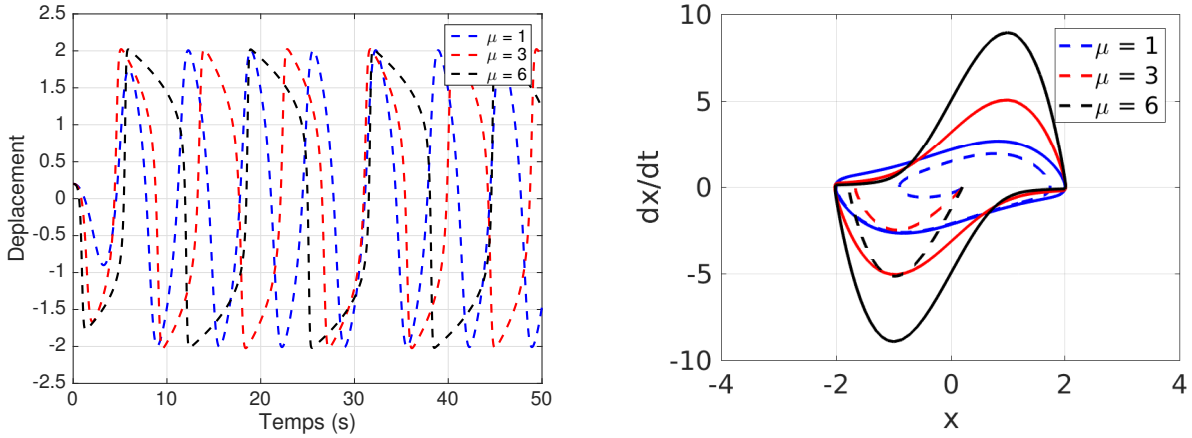


FIGURE 5 – Déplacement pour l'équation du pendule linéaire et non linéaire, avec l'intégrateur temporel `ode45`.

```
» [T1,X1]=vdp([.2 0],200,1);
» [T3,X3]=vdp([.2 0],200,3)
» [T6,X6]=vdp([.2 0],200,6)
```

Cela donne la figure 6, représentant les déplacements `X1`, `X3` et `X6` (voir figure 6 (a)), et les orbites dans l'espace des phases (voir figure 6 (b)).



(a) (b)

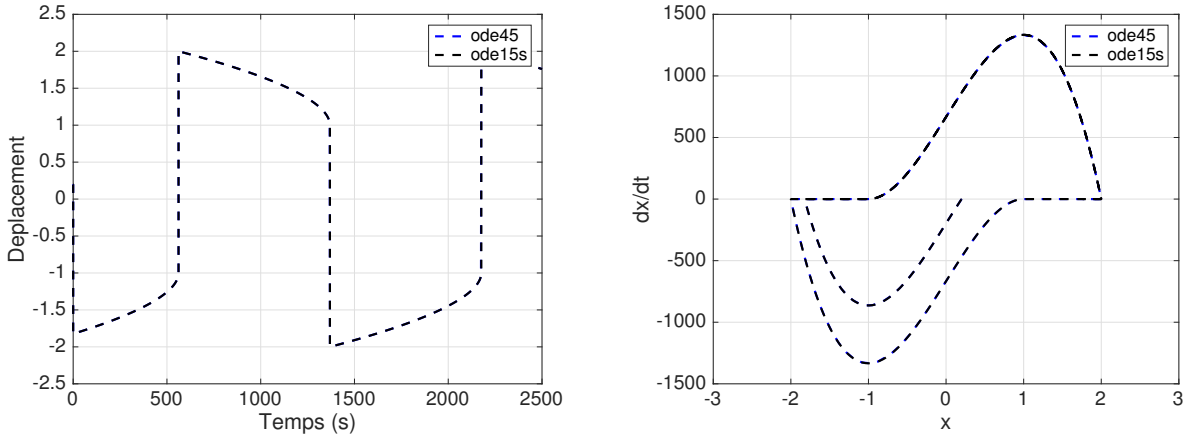
FIGURE 6 – Déplacement (a) et solution dans l'espace des phases (b) pour $\mu = 1, 3, 6$ pour les intégrateurs temporels `ode45` et `ode15s`.

On remarque qu'en augmentant μ, la période des oscillations augmente et le cycle limite se déforme de plus en plus, faisant apparaître une zone de transition très raide où le comportement temporel est quasiment vertical.

A cause de ce comportement particulier, pour lequel en un très court instant on passe à des valeurs extrêmement différentes, on dit que le problème est numériquement *raide*. L'intégrateur va devoir en effet trouver ces valeurs extrêmes pour un pas de temps qui va devenir infiniment petit.

Si l'on intègre l'équation de Van der Pol, pour $\mu = 1000$, avec `ode45`, le temps d'exécution est relativement grand. Le solveur n'est en fait pas du tout adapté à la résolution de cette raideur. Il faut donc employer

un autre solveur de MATLAB, programmé afin de faire face aux problèmes raides, par exemple : `ode15s`. Il permet de résoudre le problème en un temps beaucoup plus court (environ 400 fois plus court) qu'en utilisant `ode45`. On trouve alors les solutions présentées sur la figure 7, qui sont exactement les mêmes dans les deux cas.



(a) (b)

FIGURE 7 – Déplacement (a) et solution dans l'espace des phases (b) pour $\mu = 1000$ pour les intégrateurs temporels `ode45` et `ode15s`.