API0055-Identification d'équations de systèmes dynamiques par les données

Merwane Bouri, Jiaqi Liu

July 2022

1 Introduction

1.1 Objectifs de l'API

L'objectif principal de cette API est d'introduire les méthodes d'apprentissage de modèles de systèmes dynamiques, des méthodes d'apprentissage qui sont guidées par les données. Pour répondre à cette problématique, des outils mathématiques et numériques vont être présentés en cours, ce qui nous permettra de déterminer des équations et des données au cours de temps, et de mesurer l'état d'un système au cours du temps. Ces éléments théoriques vont être appliqués dans un environnement de programmation, dans notre cas, nous avons choisi Scilab.

2 Développement technique

2.1 Contexte et objectifs

La question centrale de l'API est la suivante :
Comment à partir d'une famille de données $\{\vec{x}_n, n \in [\![0; N]\!]\}$ pouvons-nous identifier la fonction $\vec{f}$ telle que

$$\begin{cases} \vec{x}_{n+1} = \vec{f}(\vec{x}_n) \\ \vec{x}(0) = \vec{x}_0 \qquad \text{Donné.} \end{cases}$$

Les hypothèses sont les suivantes :
— Nous définissons $\vec{x}(t) \in \mathbb{R}^d$ comme état du système à l'instant t.
— Nous avons des données discrètes : $\vec{x}_n = \vec{x}(t^n)$.
— Un intervalle de temps $t^n = n \times \tau, avec\ \tau > 0$.

Dans la suite de ce rapport, nous répondrons à cette question à l'aide des notions abordées en cours, des exemples vus en Cours et en TP (oscillateur harmonique, Système de Lorenz, Système à deux masses et trois ressorts, ...), chaque exemple sera aussi abordé numériquement sous **Scilab**.

2.2 Rappels théoriques

Avant de pouvoir répondre à cette question, des notions théoriques sont nécessaires en guise d'introduction et d'entrée en matière.

2.2.1 Équations différentielles ordinaires

Cas scalaire

Lors du premier cours, des rappels ont été effectués autour des équations scalaires différentielles linéaires. Nous avons vu les cas suivants :

— équation scalaire linéaire du premier ordre à coefficients variables, homogène.
— Cas non homogène
— Les équations différentielles du second ordre, linéaires, à coefficients constants.

Ces notions sont des rappels dans le cas scalaire afin d'introduire le cas vectoriel.

Cas vectoriel

Dans la suite du rapport, nous nous permettons d'utiliser un abus de notation pour faciliter l'écriture, nous considérons maintenant que toutes nos variables sont des vecteurs colonnes :
Nous considérons que $x(t) \in \mathbb{R}^d$ (de la même manière que $\vec{x}(t) \in \mathbb{R}^d$). Soit l'équation différentielle :

$$\begin{cases} \dot{x}(t) = A \times x(t) \\ x(0) = x_0 \qquad\qquad \text{Donné.} \end{cases}$$

Avec $A \in \mathbb{M}_d(\mathbb{R})$
Nous avons vu en cours que la solution pour cette équation différentielle est

$$\boxed{\text{x(t)= exp}\,(At)\ x_0} \tag{1}$$

Remarque : L'exponentielle d'une matrice a été définie en cours :
Soit une matrice $B \in \mathbb{M}_n(\mathbb{R})$:

$$\exp B = \sum_{k=0}^{\infty} \frac{B^k}{k!}$$

NB : L'exponentielle d'une matrice sous Scilab est la fonction ***expm()***.

Si B est diagonalisable alors

$$\exp B = P diag(\exp \lambda_i) P^{-1},$$

avec $\lambda_i \in \mathbb{C}$ les valeurs propres de B et P la matrice de passage. Ainsi, la solution de 1 peut s'écrire :

$$x(t) = \exp\,(At)\, x_0 = P\, diag(\exp \lambda_i\, t)\, P^{-1}\, x_0 \tag{2}$$

2.2.2 Interprétation physique selon les valeurs propres

Reprenons l'équation de notre modèle physique 1, notons λ_k la valeur propre associée au vecteur propre $\vec{r_k}$, de la matrice A de notre système. Selon la nature des valeurs propres et la condition initiale, il est possible d'obtenir des informations qualitatives sur notre système.
Par définition, nous avons

$$A\,\vec{r_k} = \lambda_k\,\vec{r_k} \tag{3}$$

On a aussi,

$$\exp(A\,t)\,\vec{r_k} = \exp(\lambda_k\,t)\,\vec{r_k} \tag{4}$$

Donc si $x_0 = \vec{r_k}$
Alors

$$x(t) = \exp(\lambda_k\,t)\,\vec{r_k} = P\,\exp(Re(\lambda_i) + i\,Im(\lambda_i)\,P^{-1}\,x_0 \tag{5}$$

Notons les solutions de 1 de la manière suivante,

$$x(t) = P\,\exp(Re(\lambda_i) + i\,Im(\lambda_i)\,P^{-1}\,x_0 \tag{6}$$
$$= P\,\exp(Re(\lambda_i)exp(i\,Im(\lambda_i))\,P^{-1}\,x_0 \tag{7}$$

Ainsi, nous pouvons notifier plusieurs comportements possibles selon la nature des valeurs propres. Nous allons exposer quelques exemples (non exhaustifs).

Si $\forall\ i\ Re(_i)\ < 0$ alors le système est dissipatif, il est asymptotiquement stable $\forall\ x_0$.

Si $\exists\ i_0$ tel que $Re(\lambda_{i_0}) > 0\ et\ \exists\ x_0 = \vec{r_k}$ alors le système converge asymptotiquement vers l'infini.

Si $\forall\ i\ Re(_i)\ = 0$ et $\exists\ i_0\ /\ Re(\lambda_{i_0}) \neq 0$ alors le système est infiniment oscillant. Nous nous concentrons sur l'exemple de l'oscillateur harmonique, un exemple qui nous a accompagné tout au long de l'API.

2.3 Cas pratique : Oscillateur harmonique

L'oscillateur harmonique est un système linéaire qui est constitué d'une masse et d'un ressort, sans force dissipative. Voici les équations qui gouvernent le système.

$$\dot{x}(t) = u(t), \tag{8}$$
$$m\,\dot{u}(t) = -k\,x(t), \tag{9}$$

Nous pouvons l'écrire de cette marnière,

$$\frac{d}{dt}\begin{pmatrix}\dot{x}(t)\\ \dot{u}(t)\end{pmatrix} = A\begin{pmatrix}x(t)\\ u(t)\end{pmatrix} \tag{10}$$

Avec $A = \begin{pmatrix}0 & 1\\ -k/m & 0\end{pmatrix}$

Les valeurs propres de A sont $\lambda = \pm\, i\sqrt{\frac{k}{m}}$, avec l'analyse faite dans 2.2.2 nous pouvons déduire que le système est infiniment oscillant. Nous allons vérifier ce caractère dans la prochaine section grâce à Scilab, notamment au plan de phase.

2.3.1 Résolution de l'équation avec ode et Runge-Kutta

Afin de générer les solutions de nos équations 1, nous considérons par la suite le modèle discret :

$$x^n = x(t^n), \tag{11}$$
$$x^{n+1} = \exp(A\,\tau)\,x^n, \tag{12}$$

Sous Scilab, nous utiliserons la fonction **ode()** (Ordinary Differential Equation), qui est un solveur d'équations différentielles ordinaires, son utilisation est détaillée dans la section suivante.

Nous avons aussi implémenté un intégrateur d'EDO, *Runge-Kutta d'ordre 2.*
Soit l'équation :

$$\dot{x}(t) = f(x(t)), \tag{13}$$
$$x(0) = x^0 \tag{14}$$

Nous avons vu en cours plusieurs schémas pour résoudre l'équation, comme le schéma d'Euler explicite, $x(t^{n+1}) = x(t^n) + \tau\, f(x(t^n)$.
Nous avons utilisé un développement de Taylor.

Pour *Runge-Kutta d'ordre 2*, cela ressemble à la méthode des trapèzes avec comme distinction, une estimation de $x(t^{n+1})$ à l'aide du schéma d'Euler d'explicite, nous avons donc deux étapes qui sont les suivantes :

$$\hat{x}^{n+1} = x^n + \tau\, f(x^n), \tag{15}$$
$$x^{n+1} = x^n + \tfrac{\tau}{2}\,\{f(x^n) + f(\hat{x}^{n+1})\} \tag{16}$$

Le script sous Scilab est diponible dans 4.2.

Il existe des méthodes Runge-Kutta d'ordre plus élevée, comme RK4 ou RK45.

2.3.2 Résolution numérique sous Scilab

À partir du TP, nous avons d'abord initialisé les données de la façon suivante : $k = 1$, $m = 1$, $x^0 = 1$, $u^0 = 1$, $T = 10\pi$, $N = 1000$ et A $= \begin{pmatrix} 0 & 1 \\ -k/m & 0 \end{pmatrix}$.

Ensuite, en utilisant la formule : $\dot{x} = A\,x$ nous avons créé une fonction f pour calculer chaque x^n à instant $n * \tau$ (τ est une unité incrémente de temps et $\tau = T/N$).

Après, en appliquant la fonction *ode*() dans Scilab ou la fonction Runge-Kutta 2 $RK2$() qui est créée par nous, nous obtenons une liste des données générée à partir de ce modèle d'oscillateur harmonique, la liste des solutions. Nous appelons cette liste : $Xsol$. Le script est présent dans 4.1.

2.3.3 Résultat

Nous générons des graphes pour le déplacement en fonction de temps (Figure 1), le mouvement vectoriel (Figure 2) et scalaire (Figure 3) et aussi l'énergie total (Figure 4).

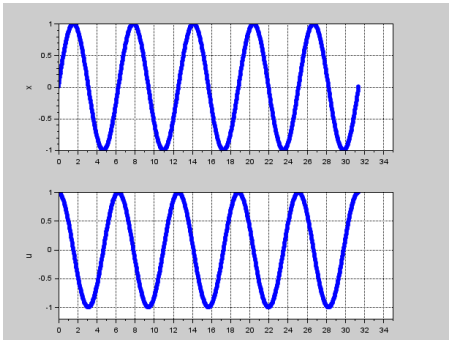

FIGURE 1 – le déplacement

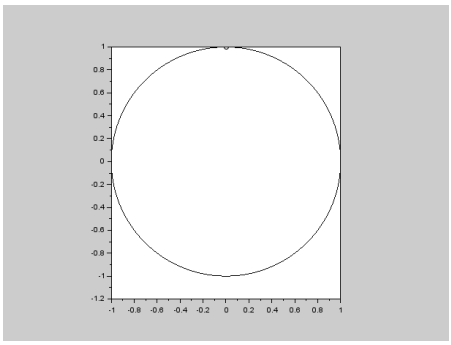

FIGURE 2 – le mouvement vectoriel

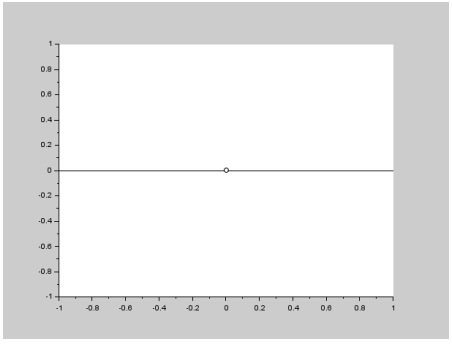

FIGURE 3 – le mouvement scalaire

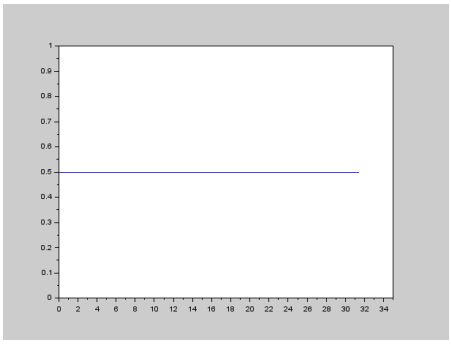

FIGURE 4 – l'énergie

En vue qu'il n'y a pas d'amortissement dans ce système, nous pouvons voir que dans le (Figure 1), en progressant dans le temps, le déplacement de ressort est toujours entre 1 et -1 et le ressort peut toujours atteindre la valeur 1 et -1, nous avons donc un système infiniment oscillant.

Dans (Figure 2), pour le mouvement vectoriel, le point tourne dans la base vectorielle.

Dans (Figure 3), c'est comme le mouvement réel de ressort (sous condition sans amortissement).

Dans (Figure 4), l'énergie est conservée et a toujours une même valeur. Cela s'explique que c'est un système sans amortissement, soit sans perte.

2.4 Génération des données

Dans la partie suivante, nous allons retrouver la matrice A à partir des données. Nous devons donc identifier le modèle physique de ce système, nous allons effectuer une identification d'équations à partir des données ($Xsol$).

2.5 Identification d'un système dynamique linéaire, modèle linéaire

Nous revenons à la problématique principale de l'API, comment à partir des données retrouver la dynamique du système dans un cas linéaire. Nous avons vu en cours que nous cherchons $\mathcal{A} = \exp(A\,\tau)$ au sens des moindres carrés. C'est un problème d'optimisation.

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \sum_{k=0}^{N-1} \left\| x^{n+1} - \mathcal{A}\, x^n \right\|_{\mathbb{R}^d}^2 \tag{17}$$

Pour cela, nous avons effectué plusieurs rappels en cours, des rappels sur l'espace vectoriel $\mathbb{R}^d$, la définition d'une norme, produit scalaire. Cela permet d'introduire ces opérateurs sur des matrices. Nous avons vu le produit scalaire et norme de Frobenius sur les matrices rectangulaires, afin d'écrire 17 sous forme matricielle sans la somme. Ainsi, nous pouvons définir le critère suivant :

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \sum_{k=0}^{N-1} \left\| x^{n+1} - \mathcal{A}\, x^n \right\|_{\mathbb{R}^d}^2 = \|Y - \mathcal{A}\, X\|_{F,n,d}^2 \tag{18}$$

Avec $Y = (x^1, .., x^N)^T$ et $X = (x^0, .., x^{N-1})^T$
Le problème revient à minimiser la fonction de coût suivante,

$$J(\mathcal{A}) = \|Y - \mathcal{A}\, X\|_{F,n,d}^2 \tag{19}$$

On cherche donc $J(\mathcal{A}^*) < J(\mathcal{A})$,$\forall\ \mathcal{A}$ ce qui est équivalent $J(\mathcal{A}^*) < J(\mathcal{A}^* + H)$ avec $H \in \mathbb{M}_{d,N}(\mathbb{R})$

Nous avons conclu le résultat

$$\mathcal{A}^* = YX^T(XX^T)^{-1}. \tag{20}$$

Si XX^T est inversible.

2.5.1 Application sous Scilab avec l'exemple de l'oscillateur harmonique et les données d'apprentissage

Le script de cette partie est donné dans 4.3 Nous allons d'abord calculer la matrice $\mathcal{A}$ avec la formule vu précédemment :

$$\mathcal{A} = YX^T(XX^T)^{-1}.$$

$\mathcal{A}$ a une équivalence pour $exp(A\tau)$, donc quand nous testons : $E = \mathcal{A} - exp(A\tau)$, nous allons obtenir un E vers 0 (Figure 5) :

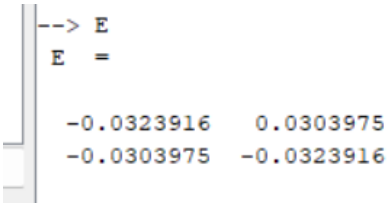

FIGURE 5 – valeur de $E = \mathcal{A} - exp(A\tau)$

Ensuite pour calculer la matrice qui approche$A$, nous allons appliquer 2 formules suivant dans Scilab :

$$\mathcal{A} = P\,\mathrm{diag}(\mu_k)\,P^{-1}.$$

$$\lambda_k = \frac{\ln(\mu_k)}{\tau}$$

Pour calculer diag(μ_k), nous utilisons la fonction *spec*() dans Scilab et pour calculer exponentiel de matrice, nous allons utiliser la fonction *expm*().

Finalement, nous utilisons la formule suivante pour retrouver la matrice approximative de A, soit la matrice B :

$$B = P\,\mathrm{diag}(\lambda_k)\,P^{-1}.$$

2.5.2 Vérification et comparaison

Voilà notre résultat en graphe (Figure 6) et (Figure 7).

Le graphe (Figure 6) est le déplacement de ressort. Ce sont des données retrouvées avec la nouvelle matrice construite, soit la matrice B.

Le graphe (Figure 7) est le graphe pour évaluation de l'énergie au cours de temps. Il varie de 0.5 à 0.5001, donc nous pouvons dire qu'il conserve l'énergie au cours de temps. Et voilà le résultat de matrice retrouvé (B) (Figure 8) comparé avec la matrice origine A. Nous pouvons voir que sur la partie réelle, B est presque identique à A

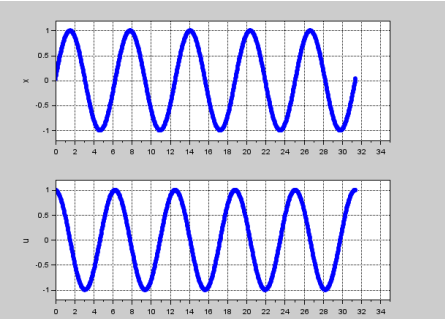

FIGURE 6 – le déplacement

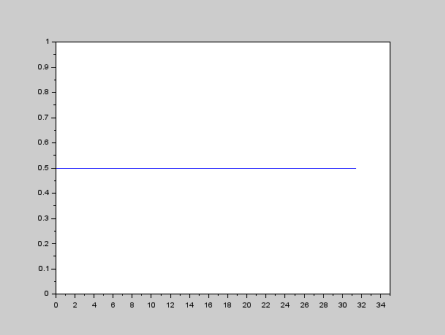

FIGURE 7 – Variance de l'énergie

```
--> A_retouvé
 A_retouvé  =

   4.D-06 + 0.i    1. + 8.D-22i
  -1. + 0.i    4.D-06 - 2.D-16i

--> A
 A  =

   0.    1.
  -1.    0.
```

FIGURE 8 – A et B ($Aretrouv$)

Nous voulons aussi comparer les résultats générés avec A et B. Parce que nous n'arrivons pas à clairement voir si nous superpose les graphes, nous mettons donc les graphes en parallèle.

Dans le graphe (Figure 9), celui qui est en rouge est celui qui est relatif à A et celui qui est en vert est celui qui est relatif à B. Les 2 premiers graphes sont pour la valeur de x et les 2 derniers graphes sont pour la valeur de u.

Dans le graphe (Figure 10) , nous pouvons aussi voir que tous les deux conservent l'énergie (égale 0.5).

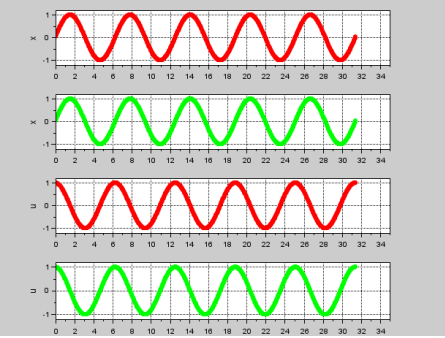

FIGURE 9 – Comparaison de déplacement

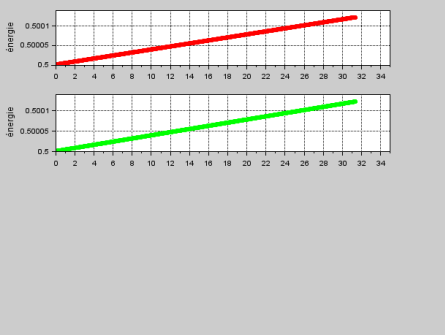

FIGURE 10 – Comparaison d'énergie

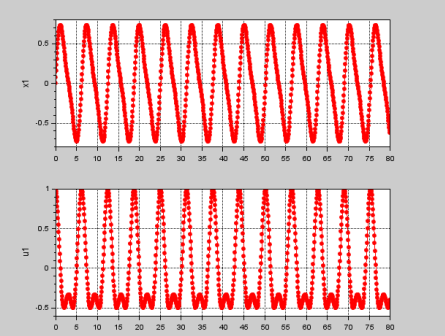

FIGURE 11 – déplacement

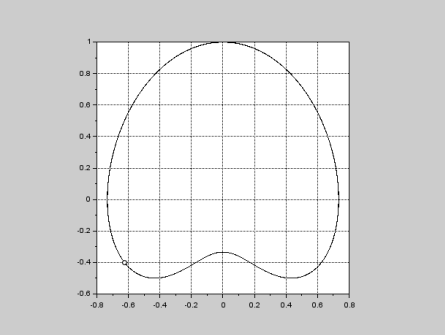

FIGURE 12 – mouvement

2.6 Autre exemple d'un cas linéaire

2.6.1 système deux masses 3 ressorts

En changeant la valeur de A : A $= \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_1+k_2}{m_1} & 0 & \frac{k_2}{m_1} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_2}{m_2} & 0 & -\frac{k_2+k_3}{m_2} & 0 \end{pmatrix}$, nous allons refaire les étapes précédentes. D'abord initialiser notre donnée et appliquer *ode*() à notre système.

Ensuite, nous allons reconstruire ce modèle avec la méthode linéaire. Donc, c'est-à-dire retrouver la A (B) et générer les donnée à partir de B.

2.7 Résultat

D'abord, voilà les données générées avec A :

Dans le graphe (Figure 11), c'est la distance pour la première masse par rapport à

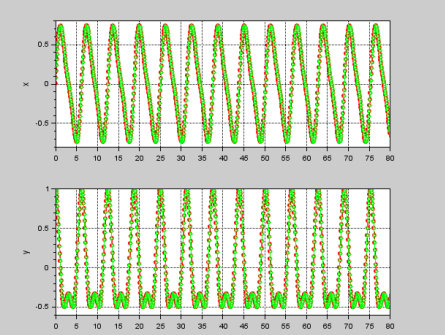

FIGURE 13 – déplacement

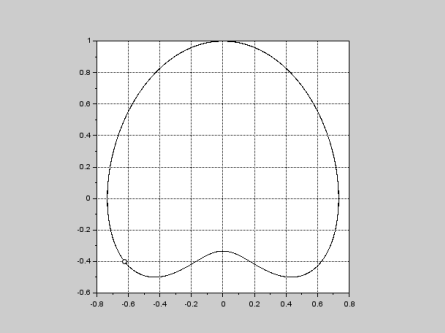

FIGURE 14 – mouvement

point central au cours du temps. (l'autre masse est symétrique avec cette masse).

Dans le graphe (Figure 12), c'est le mouvement réel pour la masse.

Ensuite, nous allons comparer cela avec les données générées par la nouvelle matrice B :

Le système de 2 masses et 3 ressorts est un système linéaire, nous pouvons donc voir que dans le graphe (Figure 13) et (Figure 14), nous avons donc exactement la même chose entre les données générées par A et B.

Et nous pouvons aussi voir la B et A(Figure 15), ce qui a presque même valeur, donc, nous allons la même résultat.

```
--> B
 B  =

   1.044D-09 - 7.298D-17i   1.        + 1.983D-24i  -1.762D-09 - 3.826D-17i   1.350D-08 - 8.623D-26i
  -2.        - 1.811D-24i  -1.768D-08 - 1.023D-16i   0.9999999 - 1.591D-24i  -8.117D-10 + 3.524D-18i
  -1.934D-09 + 1.895D-16i   1.655D-08 - 2.800D-24i   2.689D-09 - 7.052D-17i   1.        - 1.962D-24i
   1.9999999 - 5.241D-25i   3.492D-08 + 1.739D-16i  -2.9999999 + 1.248D-24i   1.556D-09 - 5.094D-17i

--> A
 A  =

   0.   1.   0.   0.
  -2.   0.   1.   0.
   0.   0.   0.   1.
   2.   0.  -3.   0.
```

FIGURE 15 – A et B ($Aretrouv$)

2.8 Cas non linéaire

2.8.1 Application sous Scilab avec exemple de système Lotka-Volterra, un système non linéaire

Tout d'abord, allons construire une fonction $LotkaVolterra()$ pour le système Lotka-Volterra avec la formule :

$$\dot{x}(t) = (1 - y(t))\, x(t), \tag{21}$$
$$\dot{y}(t) = (x(t) - 1)\, y(t). \tag{22}$$

Et ensuite, nous allons initialiser les données avec les mêmes valeurs de position initiale et période et le pas de temps.

Ensuite, nous allons aussi essayer de reconstruire ce modèle non linéaire avec le modèle data-driven linéaire. Nous avons bien réussi à retrouver des systèmes linéaires avec le modèle data-driven linéaire. Nous voulons donc voir si le modèle data-driven linéaire est aussi applicable au système non linéaire. Nous allons donc encore reconstruire la matrice B à partir des données.

2.8.2 Apprentissage dans le cas non linéaire

La problématique principale est de savoir s'il est possible de transformer un problème non linéaire en un problème linéaire. La réponse est oui, il faut utiliser d'autres variables. L'inconvénient est qu'on transforme ce problème dans un espace infini. Nous allons donner l'algorithme qui a été donné en TP pour le système Lotka-Volterra :

— Nous avons une base de données $(x^0, .., x^n)^T$

— Calcul de l'estimateur de $\dot{x}^n$,

$$\widehat{\dot{\boldsymbol{x}}^n} = \frac{3}{2\tau}\boldsymbol{x}^n - \frac{2}{\tau}\boldsymbol{x}^{n-1} + \frac{1}{2\tau}\boldsymbol{x}^{n-2}, \qquad n = 2, ..., N\ ;$$

— Créer la matrice

$$X = \left[\boldsymbol{\Psi}(\boldsymbol{x}^2), \boldsymbol{\Psi}(\boldsymbol{x}^3), ..., \boldsymbol{\Psi}(\boldsymbol{x}^N)\right];$$

, avec $\boldsymbol{\Psi}(\boldsymbol{x}) = (x, y, xy)^T$.

— Créer la matrice

$$Y = \left[\widehat{\dot{\boldsymbol{x}}^2},\ \widehat{\dot{\boldsymbol{x}}^3},\ ...,\ \widehat{\dot{\boldsymbol{x}}^N}\right];$$

— Calculer A :

$$A = \arg\min_A \quad \|Y - AX\|_F^2,$$

soit

$$A = YX^T(XX^T)^{-1};$$

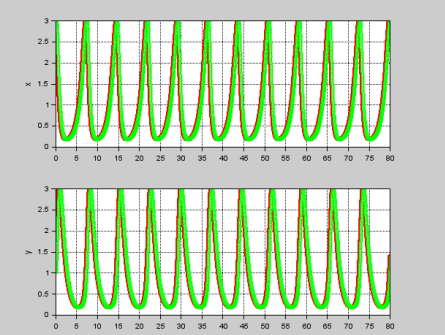

FIGURE 16 – déplacement avec le temps

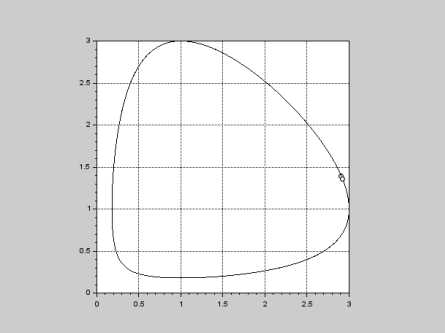

FIGURE 17 – mouvement

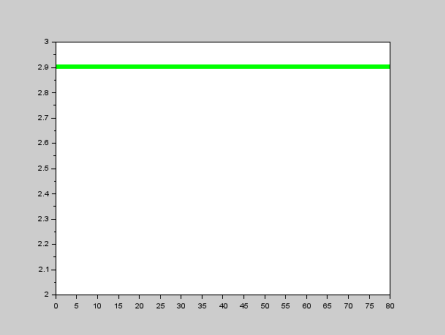

FIGURE 18 – énergie

— Utilisation d'un intégrateur (ici RK2) pour calculer les solutions numériques du système guidé par les données ainsi obtenu :

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{\Psi}(\boldsymbol{x}(t)).$$

Le script est donné dans 4.4.

2.9 Résultat

D'abord, nous pouvons voir les graphes pour des données générées avec le système LotkaVolterra.

Les données générées par la méthode $RK2()$ et généré par $ode()$ ont les mêmes valeurs. Dans les graphes (Figure 16), (Figure 17), (Figure 18), nous pouvons respectivement voir le déplacement de x et y en fonction de temps, le mouvement de point dans l'espace et la conservation de l'énergie.

Ensuite, pour les données, généré par la matrice B, voilà le résultat (Figure 19) et (Figure 20).

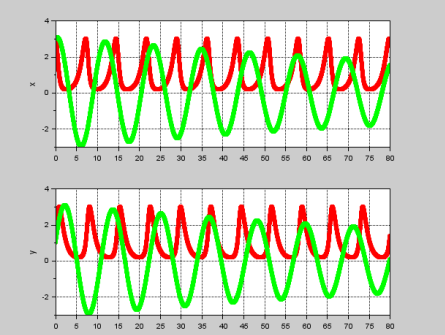

FIGURE 19 – déplacement

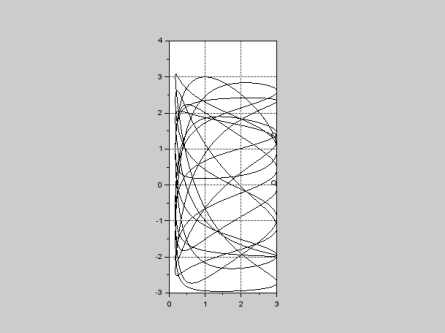

FIGURE 20 – mouvement

Dans le graphe (Figure 19), nous pouvons voir que, dans le système reconstruire, le déplacement de x,y par rapport au temps n'est plus le même.

Et dans le graphe (Figure 20), le mouvement pour le point est désordre. Donc, nous n'avons pas réussi à reconstruire le système LotkaVolterra avec la méthode linéaire. Donc, nous montrons que nous ne pouvons pas reconstruire un système non linéaire avec un modèle data-driven linéaire. Voilà la limite pour le modèle data-driven linéaire.

2.10 Système oscillant multidimensionnel

2.10.1 Application sous Scilab avec exemple de système oscillant multidimensionnel

Le script est donné dans la section 4.5.
Dans le système oscillant multidimensionnel, nous allons d'abord décider la dimension $r = 6$. Ensuite, nous construisons la matrice $K \in \mathscr{M}_r(\mathbb{R})$:

$$K = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

Et après, nous considérons le système dynamique linéaire de taille $(2r)$ suivant :

$$\dot{(t)} = (t),$$
$$\dot{(t)} = -K(t).$$

Donc, maintenant, nous pouvons définir notre A avec la formule :

$$\dot{\boldsymbol{x}}(t) = A\,\boldsymbol{x}(t), \qquad A = \begin{pmatrix} [0] & I \\ -K & [0] \end{pmatrix}.$$

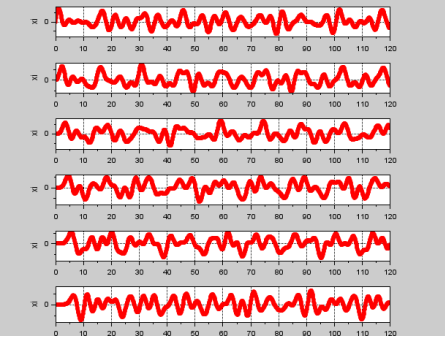

FIGURE 21 – déplacement

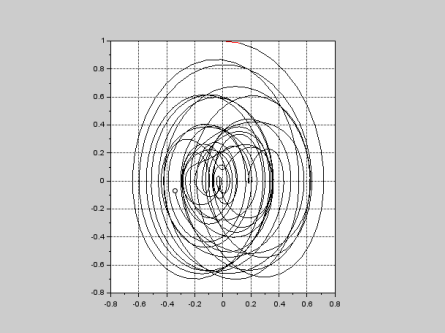

FIGURE 22 – mouvement

Après avoir initialisé tout cela, nous pouvons appliquer la méthode *ode*().

Nous allons aussi voir les séries temporelles et la courbe paramétrée $(a_1(t), b_1(t))$, $t \in [t^0, T]$.

2.11 Résultat

Voilà notre résultat dans la graphe(Figure 21) et (Figure 22).

Nous pouvons voir dans le graphe(Figure 21) la série temporelle pour chaque 6 k.

Le graphe (Figure 22) est la courbe paramétrée $(a_1(t), b_1(t))$, $t \in [t^0, T]$. Dans notre variable de solution $Xsol$, 1 à 6 lignes sont pour la valeur de $a(x)$, et 7 à 12 sont des lignes pour la valeur de $b(x)$. Nous allons donc voir la courbe entre $Xsol(1,:)$ et $Xsol(7,:)$.

3 Conclusion

Pour conclure, nous avons dans la globalité les différentes méthodes d'identification d'équations de systèmes dynamiques par les données, avec une partie théorie pour expliciter les différentes équations que nous utilisons en pratique lors des travaux pratiques. Une première partie consiste à la génération des données avec le modèle physique connu, puis une seconde partie où ces données sont réutilisées afin de retrouver le modèle physique du système, une étape d'apprentissage guidée par les données. Nous avons abordé le cas linéaire à travers plusieurs exemples (oscillateur harmonique ; système 2 masses, 3 ressorts ; système oscillant multidimensionnel) et le cas non linéaire (système Lotka-Volterra ; Système de Lorenz).

4 Annexes

4.1 Script Oscillateur harmonique

```
1 //
```

```
clear;
k=1; m=1;

A = [0      1; ...
     (-k/m) 0];

function xdot = myf(t, x)
    xdot = A*x;
endfunction


//=========================
Xsol = ode(x0, t0, td, myf);
//=========================
figure(1); clf;

subplot(2,1,1), plot(td, Xsol(1,:), '.-');ylabel('x'); xgrid;
subplot(2,1,2), plot(td, Xsol(2,:), '.-');ylabel('u'); xgrid;

figure(2); clf;
plot(Xsol(1,:), Xsol(2,:), '-'); mtlb_axis('equal');
comet(Xsol(1,:), Xsol(2,:), 0.01);

figure(3); clf;
comet(Xsol(1,:), 0*Xsol(1,:), 0.01);
// Calcul de l'energie total calcul e en cours du temps
//
xt = Xsol(1,:);
ut = Xsol(2,:);
Et = 0.5*k * xt.^2 +0.5*m * ut.^2;
figure(4); clf;
mtlb_axis([t0 T 0 1]);
plot(td, Et);
```

4.2 Script Runge-Kutta d'ordre 2

```


function Xsol = RK2(x0, t0, T, N, f)
    x_prev=x0;
    x_curr=[0;0];
    x_curr_chap=[0;0];
    Xsol = [];
```

```
    for i = 1:N+1
        x_curr_chap=x_prev + tau*f(i*tau, x_prev);
        x_curr = x_prev + (tau/2)*(f(i*tau, x_prev)+f((i+1)*
           tau, x_curr_chap));
        x_prev = x_curr;
        Xsol(:,i) = x_curr;
    end;
endfunction
```

4.3 Script Apprentissage Oscillateur harmonique

```

clear;
k=1; m=1;
A = [0      1;
    (-k/m) 0];

function xdot = f(t, x)
    xdot = A*x;
endfunction

pos0 = 0;
u0 = 1;
x0 = [pos0; u0]; // Donnees initiale (colonne)
t0 = 0;
N = 1000;
T=10*%pi;
tau = (T-t0)/N;
td = t0 : tau : T;

function Xsol = RK2(x0, t0, T, N, f)
    x_prev=x0;
    x_curr=[0;0];
    x_curr_chap=[0;0];
    Xsol = [];
    for i = 1:N+1
        x_curr_chap=x_prev + tau*f(i*tau, x_prev);
        x_curr = x_prev + (tau/2)*(f(i*tau, x_prev)+f((i+1)*
           tau, x_curr_chap));
        x_prev = x_curr;
        Xsol(:,i) = x_curr;
    end;
endfunction
```

```
32
33 Xsol=RK2(x0,t0,T,N,f);
34
35 X = Xsol(:,1:$-1);
36 Y = Xsol(:,2:$);
37
38 B = Y * X' * inv(X * X');
39 E = B - expm(B*tau);
40
41 [P, diagevals] = spec(B);
42 lamda1 = log(diagevals(1,1))/tau;
43 lamda2 = log(diagevals(2,2))/tau;
44 //lamdak = log(diagevals)/tau
45
46 lamdak = zeros(2,2)
47 lamdak(1,1) = lamda1;
48 lamdak(2,2) = lamda2;
49
50 C = P * lamdak * inv(P);
51
52
53 x_curr = [0 0];
54 x_prev = x0;
55 Xsol_appr = []
56 for i = 1:N+1
57     x_curr = B*x_prev;
58     Xsol_appr(:,i) = x_curr;
59     x_prev = x_curr;
60 end;
61 /*
62 Em = 1/2*m*Xsol_appr(2,:)^2 + 1/2*k*Xsol_appr(1,:)^2
63 figure(1); clf;
64 //plot(td, Xsol(1,:), '.-');ylabel('x'); xgrid;
65 subplot(2,1,1), plot(td, Em, 'r.-' );ylabel('x'); xgrid;
66 */
67
68 figure(2); clf;
69 //plot(td, Xsol(1,:), '.-');ylabel('x'); xgrid;
70 subplot(2,1,1), plot(td, Xsol_appr(1,:), '.-');ylabel('x');
      xgrid;
71 subplot(2,1,2), plot(td, Xsol_appr(2,:), '.-');ylabel('u');
      xgrid;
```

4.4 Script Apprentissage système Lotka-Volterra

```
function xdot = Lotka(t, xvec)
    x = xvec(1);
    y = xvec(2);
    xdot(1) = (1-y)*x;
    xdot(2) = (x-1)*y;
endfunction


x0 = [3;1]; // vecteur colonne
t0 = 0;
T = 80;
N = 6000;
tau = T / N;
td = t0 : tau : T;

Xsol = ode(x0, t0, td, Lotka);

x_point_chap = [];
for i = 3:N+1
    x_point_chap(:,i) = (3/(2*tau))*Xsol(:,i) ...
                        - (2/(tau))*Xsol(:,i-1) ...
                        + (1/(2*tau))*Xsol(:,i-2);
end;

phix = [Xsol(1,:); Xsol(2,:); Xsol(1,:).*Xsol(2,:)];

X = phix(:,3:$);
Y = x_point_chap(:,3:$);

A = Y * X' * inv(X * X');

function xdot = f(t, xvect)
    x = xvect(1);
    y = xvect(2);
    phix = [x;y;x*y];
    xdot = A*phix;
endfunction

Xsol_appr =  ode(x0, t0, td, f);

figure(1); clf;
```

```
//plot(td, Xsol(1,:), '.-');ylabel('x'); xgrid;
subplot(2,1,1), plot(td, Xsol(1,:), 'r.-');ylabel('x'); xgrid;
subplot(2,1,2), plot(td, Xsol(2,:), 'r.-');ylabel('y'); xgrid;

subplot(2,1,1), plot(td, Xsol_appr(1,:), 'g.-');ylabel('x'); xgrid;
subplot(2,1,2), plot(td, Xsol_appr(2,:), 'g.-');ylabel('y'); xgrid;


figure(2); clf;
plot(Xsol(1,:), Xsol(2,:), 'r-'); mtlb_axis('equal');
comet(Xsol(1,:), Xsol(2,:), 0.01); xgrid;

plot(Xsol_appr(1,:), Xsol_appr(2,:), 'g-'); mtlb_axis('equal');
comet(Xsol_appr(1,:), Xsol_appr(2,:), 0.01); xgrid;
```

4.5 Script Système oscillant multidimmentsionnel

```
clear;
//syst me ocillant multidimmensionnel
r=6

K = 2*eye(r,r) - diag(ones(r-1,1),-1) -diag(ones(r-1,1),+1)

A=zeros(2*r,2*r)

A(1:r,r+1:$)=eye(r,r)

A(r+1:$,1:r)=-K

[P, diagevals]=spec(A)

function xdot=oscildim(t,x)
    xdot = A*x
endfunction

a0=zeros(r,1)
b0=zeros(r,1)
b0(1)=1
```

```
22
23 x0=[a0;b0]
24
25 t0=0; T = 120; N=1000;
26 tau = (T-t0)/N;
27 td = t0 : tau : T;
28 Xsol = ode(x0, t0, td, oscildim);
29
30
31 X = Xsol(:, 1:$-1);
32 Y = Xsol(:, 2:$);
33
34 Arond = Y * X' * inv(X * X');
35
36 x_curr = [0 0];
37 x_prev = x0;
38 Xsol_appr = []
39 for i = 1:N+1
40     x_curr = Arond*x_prev;
41     Xsol_appr(:,i) = x_curr;
42     x_prev = x_curr;
43 end;
44
45
46 figure(1); clf;
47 //plot(td, Xsol(1,:), '.-');ylabel('x'); xgrid;
48 for i=1:r
49     subplot(r,1,i), plot(td, Xsol_appr(i,:), 'r.-' );//ylabel
       ('x %i'); xgrid;
50     subplot(r,1,i), plot(td, Xsol(i,:), 'g.-' );ylabel('xi');
        xgrid;
51 end
52 figure(3); clf;
53 plot(Xsol_appr(1,:), Xsol_appr(7,:), 'r-'); mtlb_axis('equal'
   );
54 comet(Xsol_appr(1,:), Xsol_appr(7,:), 0.01); xgrid;
```