TP Réseaux d'Assimilation de données (DAN)

Tristan Hoellinger

8 avril 2022

Contents

1 Introduction

1.1 Préliminaire

Parmi les méthodes séquentielles, plus avant que les filtres de Kalman, les Réseaux d'Assimilation de données (DAN - *Data Assimilation Network*) permettent de prédire l'état d'un système dynamique observé (ODS), compte tenu des observations. L'utilisation de réseaux de neurones permet en effet d'encoder et d'apprendre des opérateurs non-linéaires, ce que les seuls filtres de Kalman ne permettent pas.

1.2 Liminaire

On souhaite implémenter un réseau d'assimilation de données (DAN - *Data Assimilation Network*) pour l'apprentissage supervisé de deux ODS (*Observed Dynamical Systems*), à savoir un système dynamique linéaire 2D, ainsi qu'un système de Lorentz 40D.

Dans les deux cas, les étapes de propagation et d'observation des systèmes dynamiques observés peuvent s'écrire :

$$\begin{cases} x_t = Mx_{t-1} + \eta_t \\ y_t = Hx_t + \epsilon_t \end{cases} \tag{1}$$

avec η_t un bruit gaussien $\mathcal{N}(0, \sigma_p I)$ et ϵ_t un bruit gaussien $\mathcal{N}(0, \sigma_o I)$.

Au cours de ce travail, on s'est concentré sur le cas du système linéaire 2D. L'espace des états est $\mathbb{R}^2$. Entre chaque pas de temps, les états x_t^b se voient appliquer $N \in \mathbb{N}$ ($= 1$ dans nos simulations) rotations de la forme :

$$x_t^b \longrightarrow Mx_t^b = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} x_t^b$$

avec $\theta = \dfrac{\pi}{100}$ et où l'exposant b désigne l'indice de la simulation ($b = 1..mb$).

En effet, nous utiliserons pour l'entraînement un ensemble de mb simulations en parallèle, si bien que l'état à l'instant t sera stocké sous la forme d'un vecteur $\mathbf{x_t}$ de taille $(mb, 2)$.

1.3 Position du problème

Dans le réseau d'assimilation de données que nous utilisons (Figure 1) :

- le procodeur $c : \mathbb{H} \to \mathbb{R}^{n+n(n+1)/2}$ associe à un état caché, les paramètres μ et Σ (ré-agencés sous forme d'un vecteur) des lois normales q_t^a et q_t^b estimant respectivement les distributions conditionnelles $p(x_t|Y_t)$ et $p(x_t|Y_{t-1})$. Il s'agit d'une simple transformation linéaire, $c(h) = Wh + b$.
- le propagateur $b : \mathbb{H} \to \mathbb{H}$ associe à un état caché, l'état caché suivant (« propagé » en temps). Il prend la forme d'un réseau de neurones résiduel (plutôt que d'un perceptron multicouche, ce qui permet d'éviter les problèmes d'évanescence du gradient). Concrètement, dans le cas du système dynamique linéaire 2D, on a utilisé `deep` couches non-linéaires de la forme

  $$h_l \longrightarrow h_l + \alpha \times \texttt{act}(W_l h + b_l)$$

  où $l = 1..\texttt{deep}$ désigne l'indice de la couche, et `act` désigne la fonction d'activation Leaky ReLU.
- l'analyseur $a : \mathbb{H} \times \mathbb{R}^d \to \mathbb{H}$ « réalise les observations » : il associe, à un état caché h^b et une observation y, un état caché h^a « mis à jour » compte tenu de y. Il prend la forme d'un réseau de neurones résiduel augmenté. Concrètement, dans le cas du système dynamique linéaire 2D, on a utilisé `deep-1` ($= 10$) couches non-linéaires de la forme

  $$h_l \longrightarrow h_l + \alpha \times \texttt{act}(W_l h + b_l)$$

où $l = 1..\texttt{deep-1}$ désigne l'indice de la couche, et `act` désigne la fonction d'activation Leaky ReLU, suivies d'une couche linéaire

$$h_{\texttt{deep-1}} \longrightarrow h_{\texttt{deep}} = W_{\texttt{deep}} h_{\texttt{deep-1}} + b_{\texttt{deep}}$$

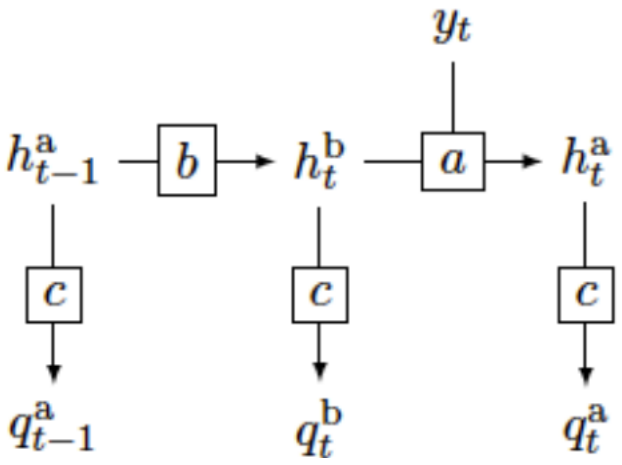


Figure 1: Structure du réseau d'assimilation de données utilisé

Nous cherchons à entraîner ce réseau de manière à s'approcher des paramètres de a, b et c qui optimisent la fonction de perte théorique

$$L = \frac{1}{T} \sum_{t \leq T} \left(L_t(q_t^a) + L_t(q_t^b) \right) + L_0(q_0^a)$$

où $L_t(q_t^a)$ est, à une fonction indépendante de q_t^a près, l'entropie croisée de $p_t^a(\cdot|y_t)$ et $q_t^a(\cdot|y_t)$, eg $\min_{q_t^a} L_t(q_t^a) = \min_{q_t^a} \mathrm{KL}(p_t^a(\cdot|y_t) \,\|\, q_t^a(\cdot|y_t))$, qui mesure donc à quel point q_t^a est un bon modèle de $p_t^a(\cdot|y_t)$; et $L_t(q_t^b)$ est, à une fonction indépendante de q_t^b près, l'entropie croisée des densités non conditionnées p_t^b et q_t^b, eg $\min_{q_t^b} L_t(q_t^b) = \min_{q_t^b} \mathrm{KL}(p_t^b \,\|\, q_t^b)$, $L_t(q_t^b)$ mesure donc à quel point q_t^b est un bon modèle de p_t^b.

1.4 Principe de la méthode

Bien entendu, minimiser la fonction de perte donnée dans le paragraphe précédent est hors de portée dans un contexte computationnel. Une approche naturelle est de minimiser plutôt, à l'aide d'un algorithme de descente, une estimation de Monte Carlo de la loss totale :

$$\mathcal{L}_{\text{train}} = \frac{1}{T} \sum_{t \leq T} \left(\mathcal{L}_t(q_t^a) + \mathcal{L}_t(q_t^b) \right) + \mathcal{L}_0(q_0^a)$$

où le $\mathcal{L}$ calligraphique désigne l'estimation de Monte-Carlo de L non-calligraphique, à partir de I séquences d'apprentissage.

Autrement dit, l'intégration dans les dissemblances de Kullback-Leibler, est remplacée par une somme du terme en log de l'intégrande sur un échantillon statistique de la loi jointe p.

On optimise alors a, b et c en utilisant un algorithme de descente (L-BFGS) sur l'estimation $\mathcal{L}_{\text{train}}$ de la fonction objectif totale L.

Une autre approche est celle du gradient tronqué, qui consiste à faire itérativement, à chaque instant t, un ou plusieurs « pas » d'une méthode de descente pour l'optimisation d'une estimation $\mathcal{L}_t(q_t^a) + \mathcal{L}_t(q_t^b)$

de la perte « courante », plutôt que d'optimiser la perte totale sur tous les instants. Il s'agit de l'approche *online*.

Nous avons implémenté la première stratégie.

2 Procédure d'apprentissage : *full mode*

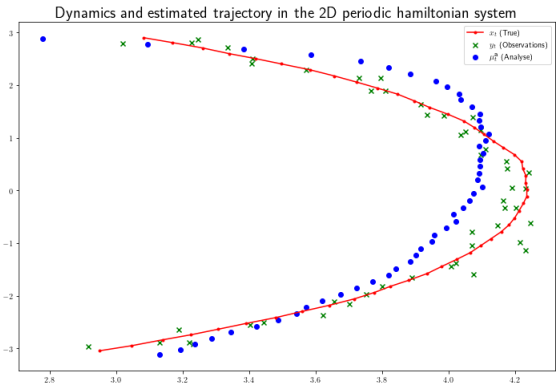
2.1 Système dynamique linéaire 2D

Nous avons généré $mb = 256$ échantillons $\{(x_t, y_t)\}_{t\in 1..T}$, $T = 50$ avec, pour chaque échantillon b, $x_0^b \sim \mathcal{N}(\begin{pmatrix}3\\3\end{pmatrix}, \sigma_0 I)$, $\sigma_0 = 10^{-1}$. Les $\{y_t\}_t$ et $\{x_t\}_{t>0}$ sont obtenus selon 1 avec $\sigma_p = \sigma_0$ et $\sigma_o = 10\sigma_0$.

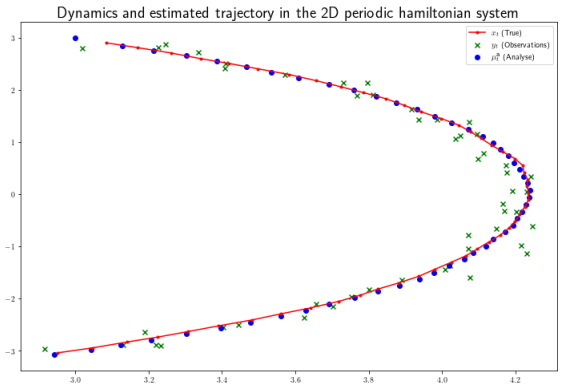
Sur la base de ces échantillons, nous avons entraîné le DAN comme décrit dans la section précédente. Suite à quoi, nous avons à nouveau généré, précisément de la même manière, des échantillons pour le test.

Sur la Figure 2.1 (b), nous représentons la moyenne des échantillons réellement générés pour ce système dynamique, aux côtés des observations bruitées, et de la trajectoire de μ_t prédite en utilisant le DAN entrainé. On constate que cette dernière est quasiment indistinguable de la trajectoire moyenne des états réels, ce qui permet de conclure que l'entraînement du DAN a bien fonctionné, malgré l'incertitude relativement importante imposée sur les observations.

Sur la Figure 2.1 (a), nous représentons les mêmes quantités, en ayant cette fois entrainé le DAN durant seulement 110 itérations. On constate que la trajectoire de μ_t est cette fois nettement moins proche de celle, moyenne, des états réels. Avec un tel nombre d'itérations, on observait encore une décroissance importante de la fonction de perte jusqu'à la fin, et on pouvait donc s'attendre à ne pas avoir encore convergé.



(a) 110 itérations, loss finale = 1,73



(b) 1130 itérations, loss finale = 0,907

Remarquons enfin que si le bruit sur les observations est nettement moindre (si on prend $\sigma_0 = 10^{-2}$, alors la trajectoire estimée, les observations, et la trajectoire moyenne théorique, se confondent visuellement, même au bout de seulement 572 itérations (cf Figure 2.1).

Pour finir, nous avons testé la « capacité de généralisation » du DAN ainsi entraîné, autrement dit, nous avons testé sa prédictivité sur des temps plus longs que celui utilisé pour l'entraînement. C'est ce que nous voyons sur la Figure 2.1. Cette excellente capacité de généralisation, sur un temps deux fois plus long que celui utilisé pour l'entraînement, est dû, fondamentalement, au caractère déterministe (hors bruit) de la propagation en temps des états de l'ODS ; et, en pratique, au caractère récurrent des réseaux utilisés dans notre DAN : en effet, on n'apprend pas seulement à reproduire l'évolution entre 0 et T elle-même, mais on apprend à encoder l'opérateur d'évolution de l'ODS. Bien entendu, la tâche ne serait pas aussi facile dans le cas d'un système non-linéaire / d'un système chaotique.

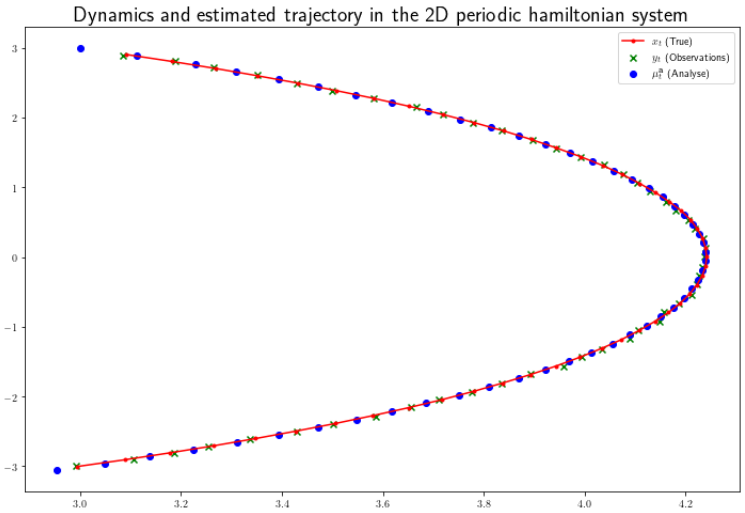


Figure 3: 572 itérations, $\sigma_o = 10\,\sigma_0 = 10^{-1}$, loss finale = -5,78

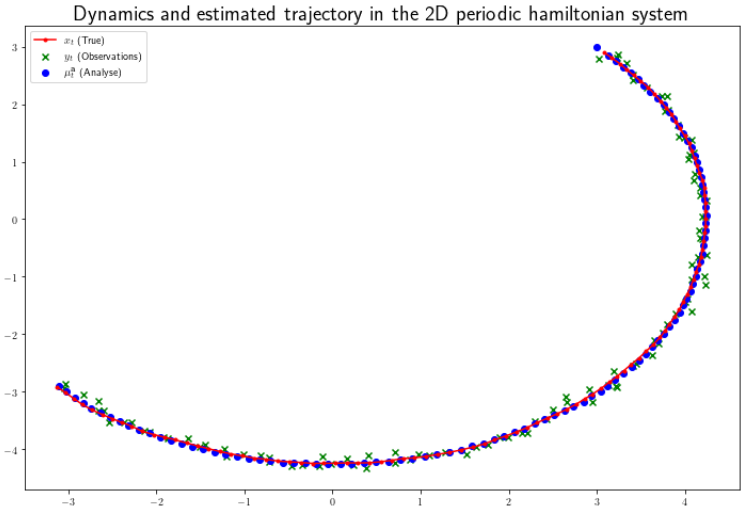


Figure 4: 1000 itérations, loss finale = 0,907. On représente la trajectoire de μ_t prédite en utilisant le DAN entrainé, sur la base d'un ensemble de test constitué d'une collection d'échantillons de l'ODS sur un temps deux fois plus long que celui utilisé pour l'entraînement.

2.2 Système de Lorenz 40D

En raison de la dimension de l'espace des états, un stratégie *online* est nettement plus pertinente qu'une stratégie *full mode* pour l'entrainement du DAN, cette dernière étant trop coûteuse en mémoire et en calculs.

A défaut d'implémenter l'approche *online*, nous avons au moins codé l'opérateur d'évolution associé au

système de Lorenz 40D. Sur la Figure 2.2, on représente l'évolution d'un état initial de la forme

$$x_0 = x_{\text{init}} \begin{pmatrix} 8 + 10^{-2} \\ 8 \\ \vdots \\ 8 \end{pmatrix}$$

jusqu'à $t = T = 50$.

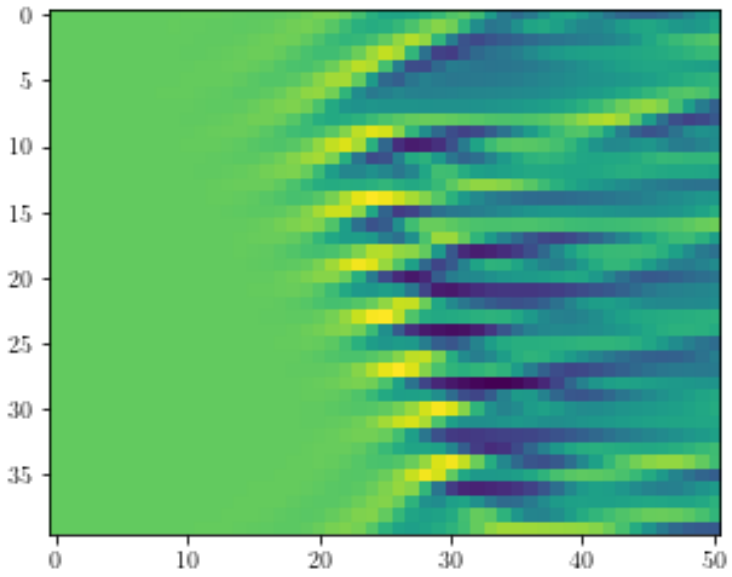


Figure 5: Dynamique de x_t dans le système de Lorenz 40D, en partant de $x_0 = x_{\text{init}}$