Rapport TP Data Assimilation Network

Nestor Carmona

5 avril 2023

Table des matières

1 Introduction

L'assimilation de données (DA) consiste à prédire l'état d'un système dynamique en combinant des observations bruitées et peu fréquentes du système avec les informations physiques du modèle, bruitées également.

Les méthodes classiques de DA utilisent deux modèles : un modèle d'observation et un modèle dynamique qui sont encadrés dans un cadre bayésien. L'estimation de la densité des états se fait à travers deux étapes récursives : l'étape d'analyse et l'étape de propagation. Elles sont données par les équations suivantes :

$$\begin{cases} x_t = \mathcal{M}(x_{t-1}) + \eta_t, & \text{équation propagation} \\ y_t = \mathcal{H}(x_t) + \varepsilon_t, & \text{équation analyse/observation} \end{cases} \tag{1}$$

avec $\mathcal{M}$ le modèle linéaire, $\mathcal{H}$ l'opérateur linéaire d'observation, η l'erreur gaussienne du modèle, ε l'erreur d'observation, x l'état du modèle et y l'observation du modèle.

Cependant, les méthodes classiques ne sont plus optimales dans des cadres non-linéaires et/ou non-gaussiens, et nécessitent donc d'autres méthodes.

Les avancées en Machine Learning et Deep Learning ont permis l'émergence de nouvelles méthodes de DA combinées à des réseaux de neurones appelées Data Assimilation Network (DAN). Les DAN offrent un avantage significatif par rapport aux méthodes classiques car elles fournissent des prévisions plus précises de l'état du système avec un coût computationnel inférieur.

Ce TP vise à appliquer les DAN à des problèmes linéaires et non-linéaires.

2 Cadre Théorique du Data Assimilation Network

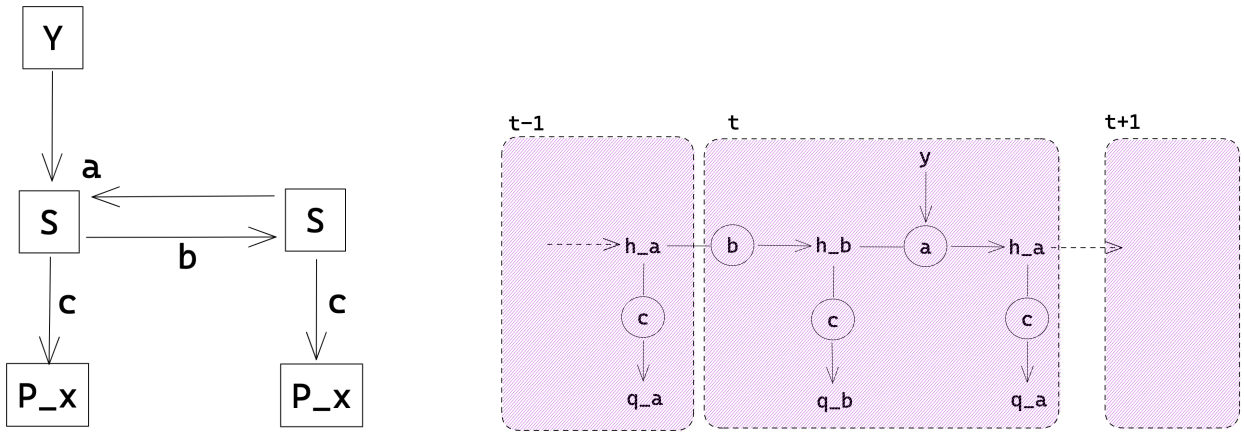
2.1 Cadre DAN

Soit $\mathbb{S}$ un ensemble. DAN est défini par le triplet de transformations :

$$\begin{cases} a \in \mathbb{S} \times \mathbb{Y} \to \mathbb{S}, & \text{analyseur} \\ b \in \mathbb{S} \to \mathbb{S}, & \text{propagateur} \\ c \in \mathbb{S} \to \mathbb{P}_{\mathbb{X}}, & \text{procodeur} \end{cases} \tag{2}$$

avec $\mathbb{X}$ l'espace des états ($\mathbb{R}^n$), $\mathbb{Y}$ l'espace des observations ($\mathbb{R}^d$) et $\mathbb{P}_{\mathbb{X}}$ l'espace des densités des variables aléatoires sur $\mathbb{X}$.

Les deux schémas de la figure 1 montrent le processus complet et déroulé (en fonction du temps) d'un Data Assimilation Network :



(a) Schéma général d'un DAN

(b) Schéma déroulé d'un DAN

FIGURE 1 – Représentations d'un DAN de façon générale et déroulée selon l'axe des temps

Comme mentionné dans l'introduction (cf. Section 1), le but est d'estimer la densité de probabilité conditionnelle des états. Dans le cadre DAN, on cherche à estimer la densité de probabilité conditionnelle de l'analyseur et du propagateur, p_t^a et p_t^b, à chaque étape t, partir des séquences d'états et d'observations des étapes précédentes. Concrètement, ces estimations sont données par q_t^a et q_t^b selon le schéma 1b.

2.1.1 Analyseur, propagateur et procodeur

Un DAN étant un réseau de neurone (récursif), on doit définir sa structure neuronale, donc expliciter l'analyseur a, la propagateur b et le procodeur c montrés dans la figure 1.

L'analyseur a correspond à des couches entièrement connectées avec des connexions résiduelles. La fonction d'activation est la LeakyReLU et on utilise l'astuce ReZero. Avant la sortie de a, on applique en dernier une couche linéaire. Son but est d'assimiler une nouvelle observation et est défini par :

$$h_t^a = a(h_t^b, y) \tag{3}$$

De façon identique, on construit le propagateur b, sans la couche linéaire de sortie. Son but est de propager le modèle (aller à l'étape $t+1$) et est défini par :

$$h_{t+1}^b = b(h_t^a) \tag{4}$$

Finalement, le procodeur c est défini en fonction de la distribution cible, une gaussienne dans notre cas. Ainsi, cela correspond à une couche linéaire où le vecteur d'entrée est transformé en la moyenne et covariance d'une distribution gaussienne. Son but est de passer d'un état caché du réseau à une estimation des densités de probabilités conditionnelles et est donné par :

$$\begin{cases} q_t^a = c(h_t^a) \\ q_t^b = c(h_t^b) \end{cases} \tag{5}$$

2.1.2 Fonction coût

Finalement, on a besoin de définir une fonction de coût. L'intuition est que l'on veut minimiser la distance entre $q_t = (q_t^a, q_t^b)$ et $p = (p^a, p^b)$ dans le sens de la perte d'information. Pour cela, on définit la fonction coût dérivée du maximum de vraisemblance :

$$\mathcal{J}_t(q_t) = -\int [ln(q_t^b) + ln(q_t^a)] p_t dx_{1:t} dy_{1:t} \tag{6}$$

La fonction de coût total est donc définie comme :

$$\mathcal{J}(q) = \frac{1}{T} \sum_{t=1}^{T} \mathcal{J}_t(q_t) \tag{7}$$

3 Implémentation du DAN & Présentation des problèmes

3.1 Implémentation en Python du DAN

3.1.1 Analyseur a

Comme le montre la figure 2 et mentionné dans la section 2.1.1, l'analyseur est composé de deux blocs : des couches entièrement connectées utilisant la fonction d'activation LeakyReLU et l'astuce ReZero (*FcZero*) ; une couche entièrement connectée linéaire (*FullyConnected*). On remarque également la présence d'une connexion résiduelle allant de l'entrée à la sortie du premier bloc. L'argument *depth* désigne le nombre de fois que l'on répète le bloc *FcZero* (ici on a $depth = 1$).

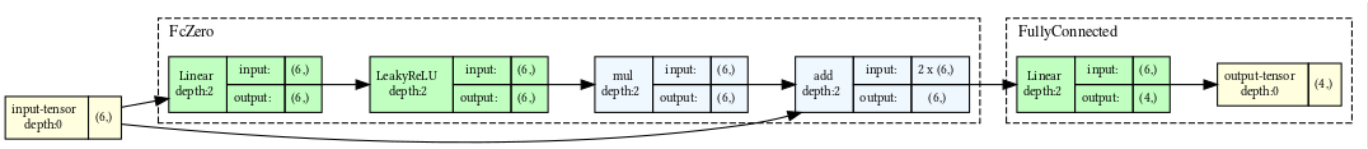


FIGURE 2 – Représentation graphique de l'analyseur (simplifié)

3.1.2 Propagateur b

Comme le montre la figure 3 et mentionné dans la section 2.1.1, le propagateur est composé d'un seul bloc, *FcZero*. Tout comme l'analyseur, l'argument *depth* désigne le nombre de fois que l'on va répéter ce bloc.

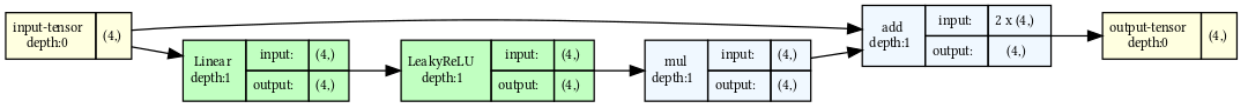


FIGURE 3 – Représentation graphique du propagateur (simplifié)

3.1.3 Procodeur c

Comme le montre la figure 3 et mentionné dans la section 2.1.1, le procodeur est simplement une couche linéaire.

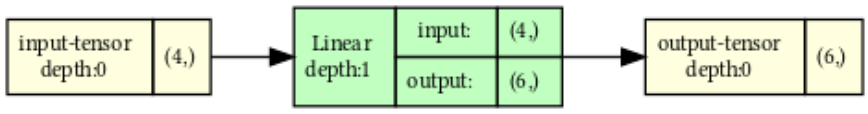


FIGURE 4 – Représentation graphique du procodeur (simplifié)

3.2 Méthodes de résolution

3.2.1 Entraînement complet

Avant de commencer l'entraînement complet du réseau de neurones, on effectue un pré-entraînement à l'étape $t = 0$ afin d'obtenir une première approximation de la densité de probabilité $p(x_0)$. Dans l'idée, c'est un *warm-start* qui permet d'obtenir une première approximation non-grossière (et qui donc réduit les chances d'avoir une fonction coût qui explose). Ainsi, on minimise uniquement la quantité :

$$\mathcal{J}_0(q_0^a) = -\log(q_0^a) \tag{8}$$

Une fois le pré-entraînement effectué, on réalise l'entraînement du réseau complet en minimisant cette fois la fonction coût complète donnée par l'équation 7 que l'on exprime :

$$\frac{1}{T}\sum_{t=1}^{T}(\mathcal{L}_t(q_t^a) + \mathcal{L}_t(q_t^b)) + \mathcal{J}_0(q_0^a) \tag{9}$$

On rappelle qu'ici on optimise l'analyseur, le propagateur et le procodeur, à l'aide d'un algorithme de descente de gradient classique ou un *L-BFGS*.

Cependant, pour des systèmes grands (cf. section 3.2.2), lors de l'étape de rétropropagation en temps du gradient (backpropagation through time - BPTT), il faut garder en mémoire le graphe computationnel complet ce qui est très coûteux en mémoire. Ainsi, pour des systèmes à grande échelle, il faut implémenter une méthode comme celle décrite dans la section suivante.

3.2.2 Entraînement en ligne

Dans cette nouvelle méthode on apporte deux modifications. La première est le calcul *à la volée* du nouvel état x et de la nouvelle observation y, au lieu de les calculer dans une boucle à part (grâce surtout au choix de l'optimiseur *Adam*). La deuxième est l'utilisation de la rétropropagation tronquée dans le temps du gradient (truncated backpropagation through time - TBPTT).

Il y a deux avantages principaux :
- Calcul réduit : en tronquant la rétropropagation dans le temps, l'entraînement est plus rapide et le stockage en mémoire est réduit.
- Amélioration de la convergence : on évite le problème d'evanescence du gradient qui est souvent dû aux séquences longues où les gradients deviennent très petits.

Le schéma suivant résume la différence d'implémentation entre BPTT et TBPTT :

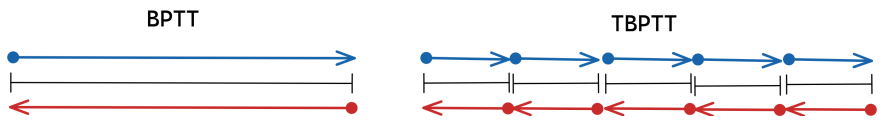


FIGURE 5 – Backpropagation through time vs. truncated backpropagation through time

3.3 Problèmes

3.3.1 Problème 1 : Linéaire 2D

Le premier problème est un système linéaire de dimension 2. On définit les espaces suivants : $X_t \in \mathbb{R}^2$ et $Y_t \in \mathbb{R}^2$. Ainsi, le système d'assimilation de données est régit par les deux équations suivantes :

$$\begin{cases} x_t = \mathcal{M}x_{t-1} + \eta_t, & \eta_t \text{ un bruit blanc, } x_t \in X_t \\ y_t = \mathcal{H}x_t + \varepsilon_t, & \varepsilon_t \text{ un bruit blanc, } y_t \in Y_t \end{cases} \tag{10}$$

$\mathcal{M}$ correspond à l'opérateur de propagation qui, dans ce système, est la matrice de rotation suivante : $\begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix}$

Finalement, $\mathcal{H}$ correspond à l'opérateur d'observation, c'est-à-dire, quels x_t on garde. Ici, on suppose qu'on observe l'état du système à chaque pas temporel, on prend donc la matrice identité. Dans la partie 3.3.3 on considérera un cas plus réaliste où les observations arrivent de manière aléatoire.

3.3.2 Problème 2 : Système de Lorenz 40D

Le deuxième problème est le système de Lorenz-95 en 40 dimensions. On définit les deux espaces : $X_t \in \mathbb{R}^{40}$ et $Y_t \in \mathbb{R}^{40}$. Les 40 équations du système sont données par :

$$\frac{dx_i}{dt} = -x_{i-2}x_{i-1} + x_{i-1}x_{i+1} - x_i + F \tag{11}$$

Les équations 11 peuvent être mises sous forme matricielle ce qui correspond à l'opérateur de propagation $\mathcal{M} : \mathbb{R}^{40} \to \mathbb{R}^{40}$. L'opérateur d'observation est $\mathcal{H}$ et est donné par l'identité comme précédemment.

Le système d'assimilation de données est régit par les deux équations suivantes :

$$\begin{cases} x_t = \mathcal{M}x_{t-1} + \eta_t, & \eta_t \text{ un bruit blanc, } x_t \in X_t \\ y_t = \mathcal{H}x_t + \varepsilon_t, & \varepsilon_t \text{ un bruit blanc, } y_t \in Y_t \end{cases} \tag{12}$$

3.3.3 Problème 3/4 : Observations partielles

Jusqu'à présent on a supposé qu'on observait entièrement le système à chaque pas de temps. Cependant, cette hypothèse n'est pas réaliste. On peut imaginer le cas d'un satellite qui envoie des informations à chaque pas de temps mais, l'atmosphère étant instable, certains de ces flux peuvent ne pas arriver. Ainsi, l'opérateur d'observation $\mathcal{H}$ ne peut pas être l'identité.

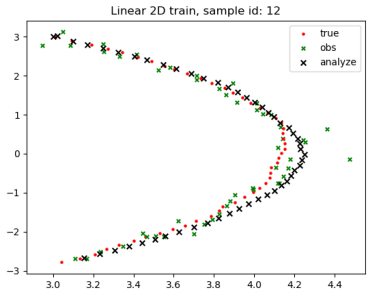
Par conséquent, on se situe dans un nouveau cadre où on suppose que les observations arrivent de façon aléatoire, selon des tirages de loi uniforme $\mathcal{U}(0, 1)$). Si le tirage est en-dessous d'un certain seuil (0.5 dans notre cas), on observe le système, si non, on n'observe pas le système. Finalement, on observe le groupe entier de simulations, c'est-à-dire, toutes les observations sont ou ne sont pas observés au même temps.

4 Résultats

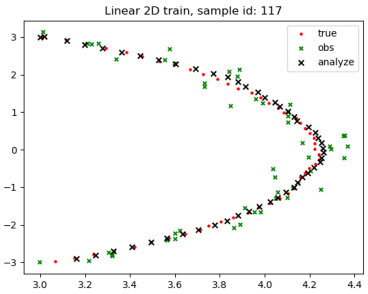
4.0.1 Problème 1 : Linéaire 2D

On utilise la première méthode de résolution : entraînement complet 3.2.1.
Les paramètres sont les suivants :
— Profondeur réseau : 10
— $T = 50$
— Opérateur observation $\mathcal{H}$: Identité
— Nombre de réalisations : 256
— Bruit sur les observations : 0.01

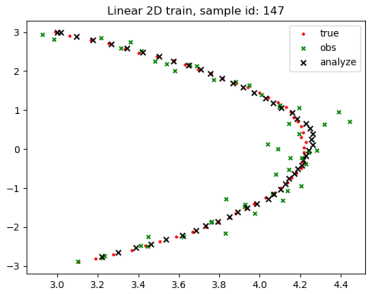
Ainsi, sur des simulations indépendantes, on obtient le résultat suivant :



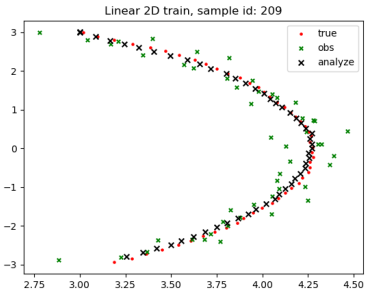
(a) Apprentissage de l'échantillon 12



(b) Apprentissage de l'échantillon 117



(c) Apprentissage de l'échantillon 147



(d) Apprentissage de l'échantillon 209

FIGURE 6 – Apprentissage pour différents échantillons

Sur la figure 6 on a trois informations dans la légende. Les deux premières proviennent des équations 12 :
— Rouge : les valeurs du modèle x_t à chaque instant t
— Vert : les observations y_t bruitées à chaque instant t
La dernière information, en noir, est l'analyseur que nous avons entraîné. On remarque que, même si les observations sont bruitées, l'analyseur reste relativement proche de la vérité modèle x_t.

Ces résultats étant identiques pour les quatre échantillons choisis, on veut donc vérifier qu'en moyenne cela reste vrai pour le reste des échantillons. Par construction des observations, on s'attend à qu'elles soient superposées aux valeurs du modèle (les observations sont des gaussiennes de

moyenne nulle, ainsi par la loi des grands nombres, $y_t \to x_t$ selon l'équation 12).

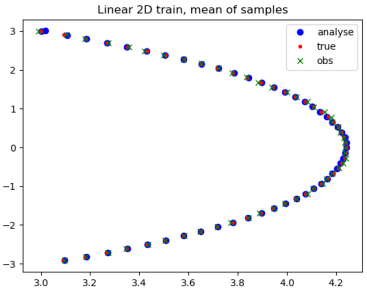
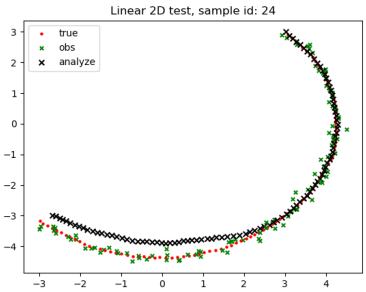
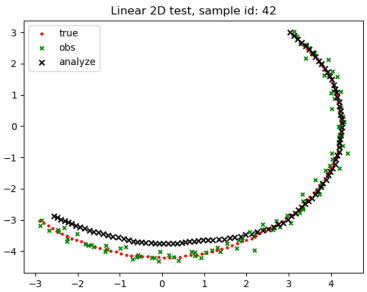


FIGURE 7 – Apprentissage moyen de l'ensemble des échantillons

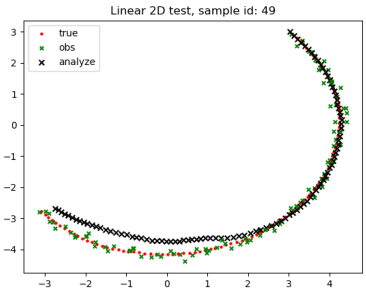
Comme énoncé ci-dessus, les trois informations sont complètement superposées comme le montre la figure 7. Ainsi, notre analyseur apprend bien sur des données d'entraînement mais il reste à voir s'il s'adapte bien à l'ensemble test. La figure 8 montre les résultats sur l'ensemble test pour une temps supérieur au temps d'entraînement :



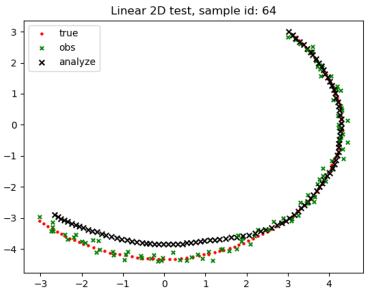
(a) Test de l'échantillon 24



(b) Test de l'échantillon 42



(c) Test de l'échantillon 49



(d) Test de l'échantillon 64

FIGURE 8 – Test sur différents échantillons jusqu'au temps T=50

Cette fois, on remarque que l'analyseur, sur certaines parties de la trajectoire (coupole) s'éloigne légèrement de la vérité. En effet, en regardant cette zone, les observations sont moins précises ce qui peut fausser le modèle. Cependant, cet effet, si réduit à quelques échantillons, peut ne pas poser de problèmes "en moyenne". Comme précédemment, on peut regarder la trajectoire moyenne sur l'ensemble des échantillons test :

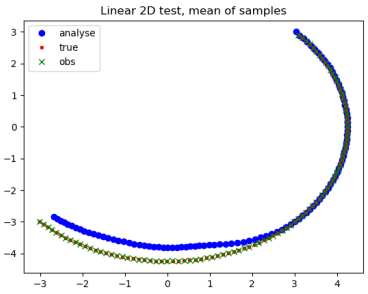


FIGURE 9 – Test moyen de l'ensemble des échantillons jusqu'au temps T=100

La figure 9 montre que les trajectoires sont confondues, en moyenne, jusqu'au temps d'entraînement. Une fois ce temps dépassé, la courbe bleu reste relativement proche de la vérité. Ainsi, avant d'affirmer que notre analyseur est correcte, il ne manque plus qu'à regarder l'évolution de la fonction coût afin d'affirmer que l'algorithme a bien convergé.

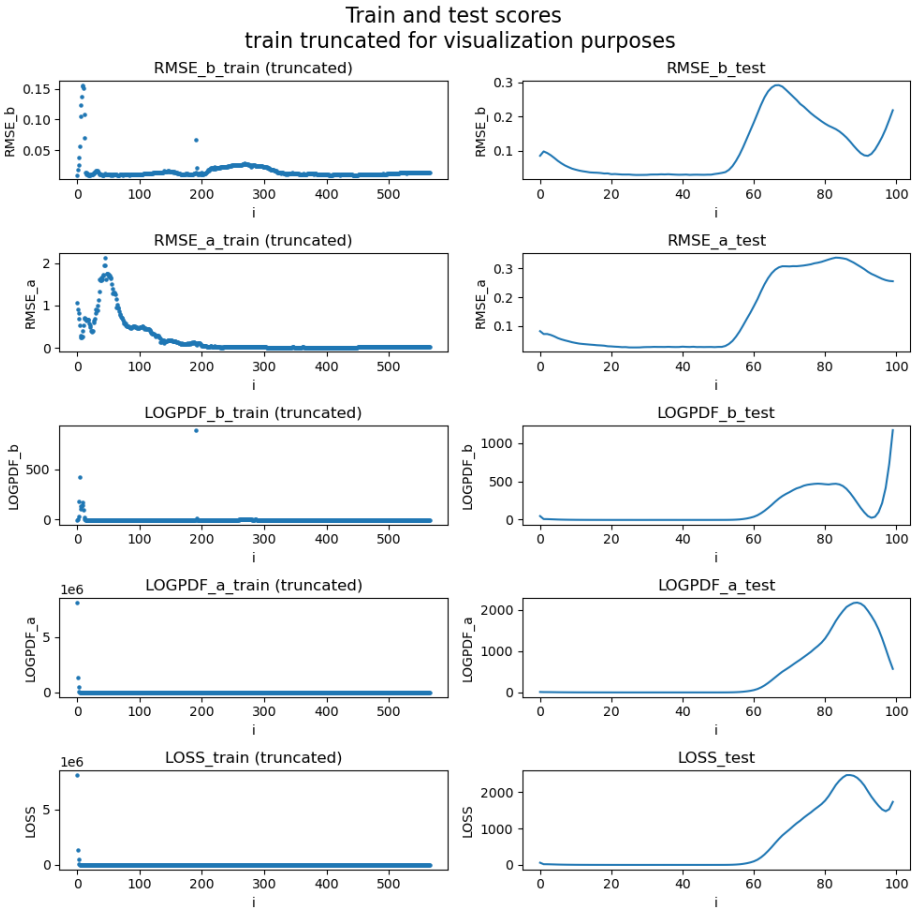


FIGURE 10 – Erreur, *RMSE* de l'analyseur, *RMSE* du propagateur, pour l'ensemble d'apprentissage et test

Dans la figure 10, on a échantillonné les tableaux pour la partie de l'apprentissage, une valeur sur 100 a été gardée, afin d'avoir une visualisation plus simple. Les courbes de la phase d'entraînement convergent bien vers leur valeurs respectives (cf. tableau 2). Pour la phase de test, les courbes convergent jusqu'à $T = 50$. Ensuite, l'erreur augmente étant donné que le réseau n'a plus appris pour des temps supérieurs à $T = 50$. Toutefois, il reste relativement proche de la vérité.

Finalement, on peut modifier les valeurs de *depth* et voir son impact sur le score initial et final :

depth	*RMSE*_a	*RMSE*_b	LOGPDF_a	LOGPDF_b	LOSS
2	4.48	4.48	$210 \cdot 10^6$	$263 \cdot 10^6 1$	$473 \cdot 10^6$
10	4.22	4.21	$1.8 \cdot 10^6$	$1.2 \cdot 10^6$	$3.0 \cdot 10^6$
20	4.19	4.19	$2.4 \cdot 10^5$	$1.9 \cdot 10^5$	$4.4 \cdot 10^5$

TABLE 1 – Scores initiaux pour différentes valeurs de *depth*

depth	*RMSE*_a	*RMSE*_b	LOGPDF_a	LOGPDF_b	LOSS	time
2	0.048	0.039	-2.75	-2.82	-5.57	1m22
10	0.027	0.029	-3.97	-3.81	-7.79	2m27
20	0.030	0.032	-3.83	-3.73	-7.55	4m06

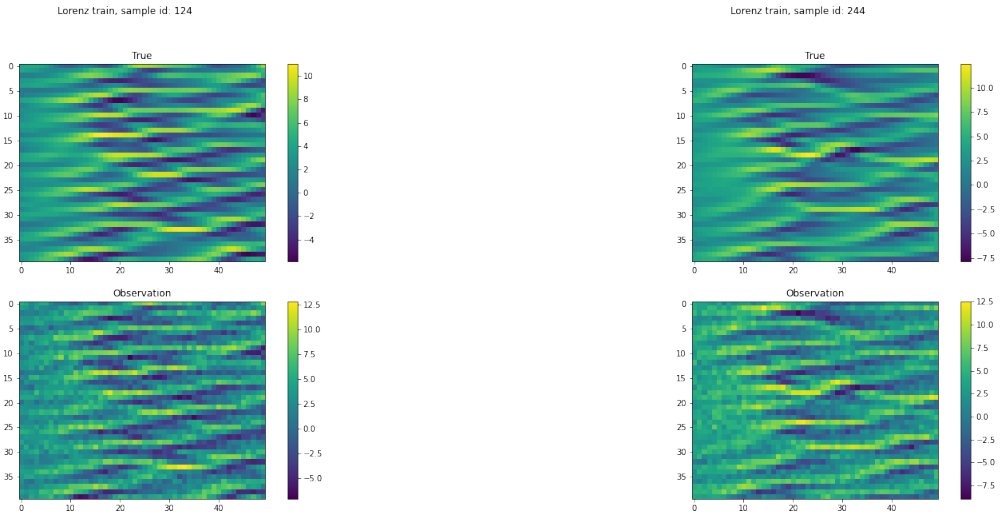
TABLE 2 – Scores finaux pour différentes valeurs de *depth*

4.0.2 Problème 2 : Système de Lorenz 40D

Sur le problème précédent, on pouvait se permettre de réaliser une rétropropagation du gradient sur tout le graphe computationnel étant donné la taille relativement faible du système : 2 * 50 * 256 = 25 600 (dimension * temps * simulations). Dans ce problème, on aura des problèmes de mémoire et temps de calcul si on applique la méthode d'entraînement complet. Le problème fait : 40 * 10000 * 1024 = 409 600 000 (dimensions * temps * simulations) en taille. Ainsi, on choisira la méthode d'entraînement en ligne 3.2.2.

Les paramètres sont les suivants :
— Profondeur réseau : 10
— T = 10 000
— Opérateur d'observation $\mathcal{H}$: Identité
— Nombre de réalisations : 1024
— Bruit sur les observations : 1
— Optimiseur : Adam avec le scheduler Exponential de PyTorch

L'étape de stockage en mémoire de l'analyseur pendant la phase d'entraînement du réseau étant lourde en mémoire, on a seulement gardé les vérités et les observations :



(a) Apprentissage de l'échantillon 124 (b) Apprentissage de l'échantillon 244

FIGURE 11 – Apprentissage pour différents échantillons

Sur la figure 11 on peut voir la vérité (en haut) ainsi que l'observation (en bas). L'axe vertical correspond à chacune des dimensions de x et l'axe horizontal correspond au temps. Ainsi, une ligne correspond à une dimension de x au cours du temps. Comme précédemment, on peut afficher la moyenne des états et des observations, on s'attend à qu'elles soient proches :

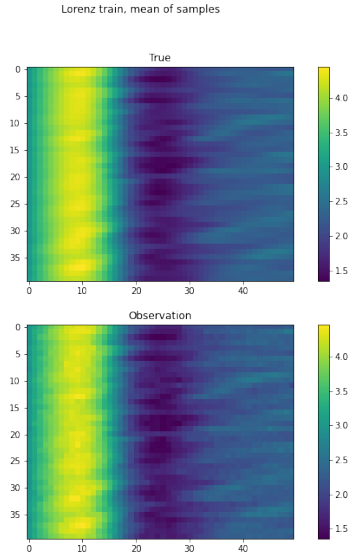


FIGURE 12 – Moyenne de l'ensemble des vérités et observations

Figure 12 affiche la moyenne des 1024 simulations de Lorenz sur les vérités et les observations et on peut voir que les deux sont très proches. Ainsi, il faut maintenant contrôler comment se comporte l'analyseur sur l'ensemble test :

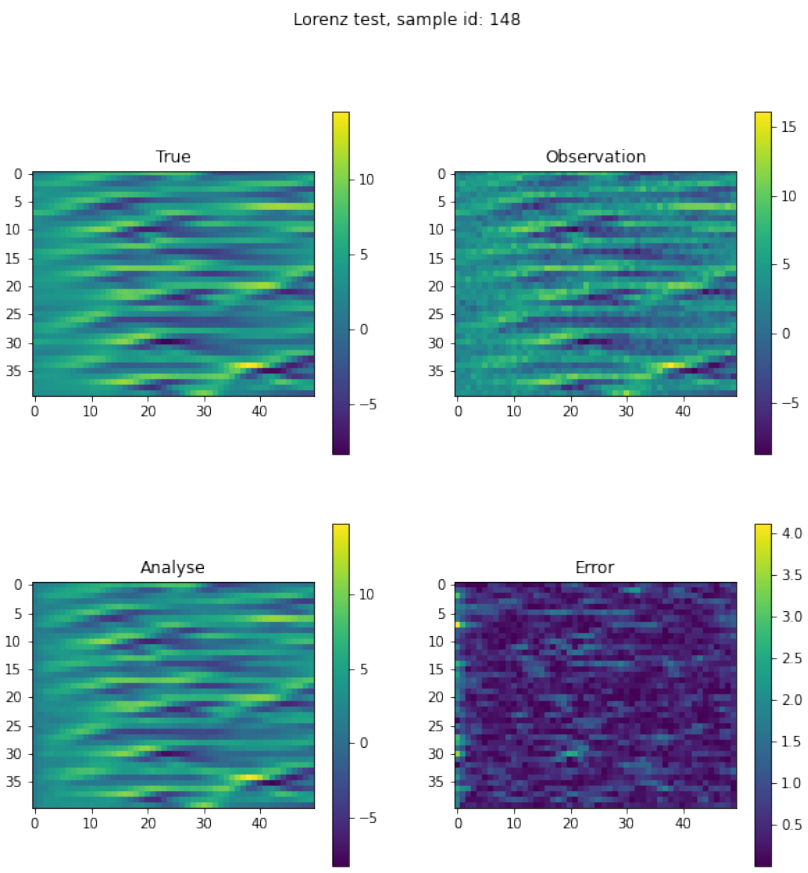


FIGURE 13 – Test sur l'échantillon 148

Sur la figure 13 on peut voir la vérité test, l'observation bruitée, la sortie de l'analyseur et l'erreur entre l'analyseur et la vérité (en norme 1). On peut voir que l'analyseur semble être très proche de la vérité compte tenu du bruitage de l'observation. Si on regarde l'erreur, plus le pixel est jaune, plus l'erreur est importante. Ainsi, on voit que les erreurs ont surtout lieu à $t = 0$. Cela est principale dû au manque d'un *warm-start* comme première approximation. L'erreur maximale en valeur absolue commise est de 4, ce qui représente tout de même environ 30%.

Regardons le comportement moyen du système :

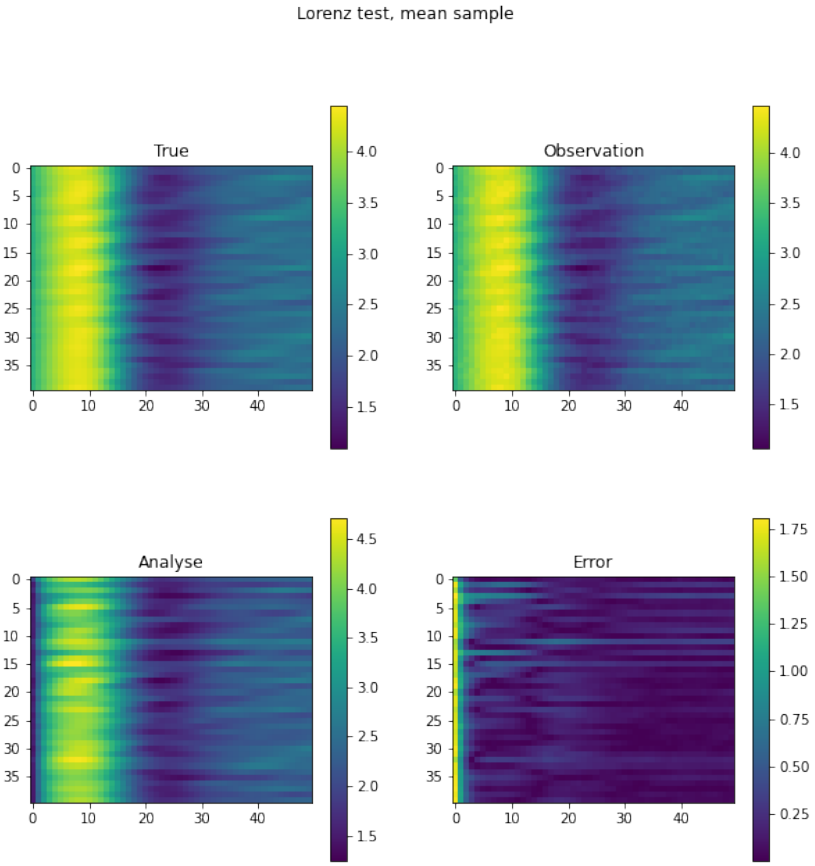


FIGURE 14 – Test moyen sur l'ensemble des échantillons

Sur la figure 14 on voit bien, qu'en moyenne, les observations sont proches de la vérité. L'analyseur cependant n'est pas très performant au début de l'échelle temporelle. En effet, il commet en moyenne entre 1.5 et 1.75 d'erreur aux temps faibles mais cette erreur diminue peu à peu au fur et à mesure que les observations arrivent.

Il ne reste donc plus qu'à vérifier que l'algorithme converge et que l'on retrouve les bons ordres de convergence :

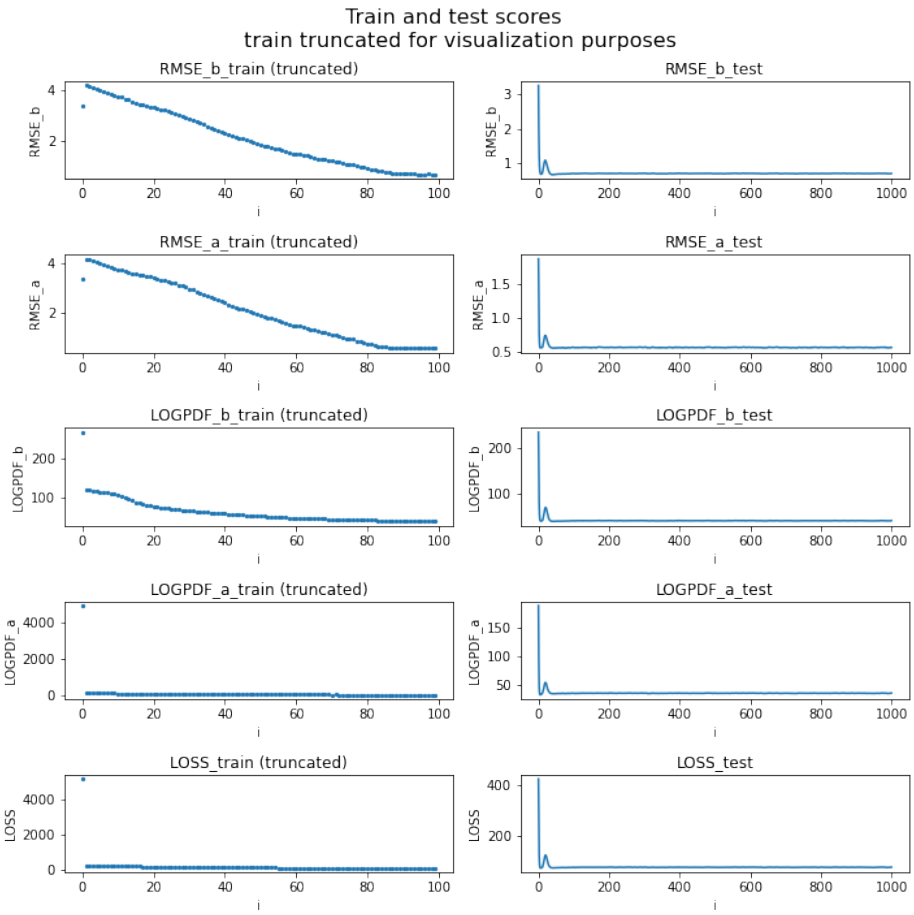


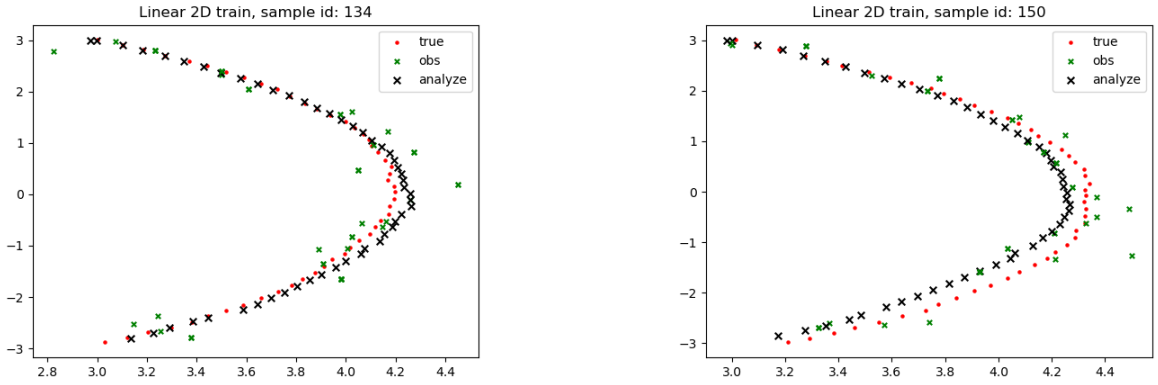
FIGURE 15 – Erreur, *RMSE* de l'analyseur, *RMSE* du propagateur, pour l'ensemble d'apprentissage et test

Sur la figure 15 on remarque que chaque courbe converge. La fonction coût de l'entraînement converge vers 70, 37 et 32 comme valeurs de convergence de la log-vraisemblance de l'analyseur et du propagateur respectivement. Quant aux *RMSE*, tous les deux convergent vers 0.5 et 0.7 pour l'analyseur et le propagateur, respectivement. Pour l'ensemble test, les valeurs et comportements de convergence sont très similaires. On en conclut donc que l'analyseur a plutôt bien fonctionné même s'il présente des faiblesses au début de l'exécution.

4.0.3 Problème 3/4 : Observations partielles

Linéaire 2D avec observations partielles Dans cette section, on va réaliser des observations partielles comme expliqué dans la section 3.3.3. Il ne reste donc qu'à définir la modification de notre réseau car, en effet, dans le cas où il n'y a pas d'observation, on ne peut pas utiliser l'analyseur a tel quel, comme le montre la figure 1b. Cependant, on peut utiliser la prédiction du procodeur $q_b = y_{pred}$ comme "nouvelle observation" que l'on introduit dans l'analyseur à la place de y.

Voici les résultats pour le problème 1 avec des observations partielles :



(a) Apprentissage de l'échantillon 134 (b) Apprentissage de l'échantillon 150

FIGURE 16 – Apprentissage pour différents échantillons avec observations partielles

Sur la figure 16 on peut voir le nombre de croix vertes réduit par rapport à la figure **??**. Cependant, l'analyseur suit plutôt bien la vérité dans les deux échantillons. Il présente le même problème à la coupole où la courbe se dévie légèrement.

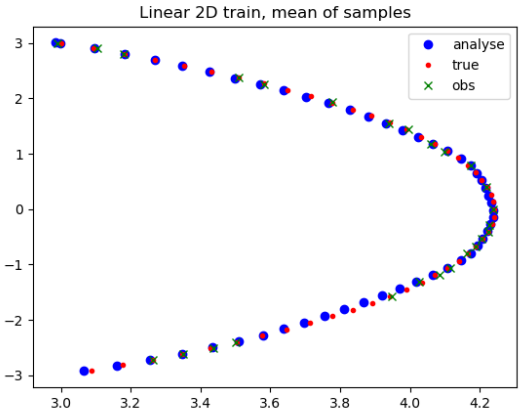
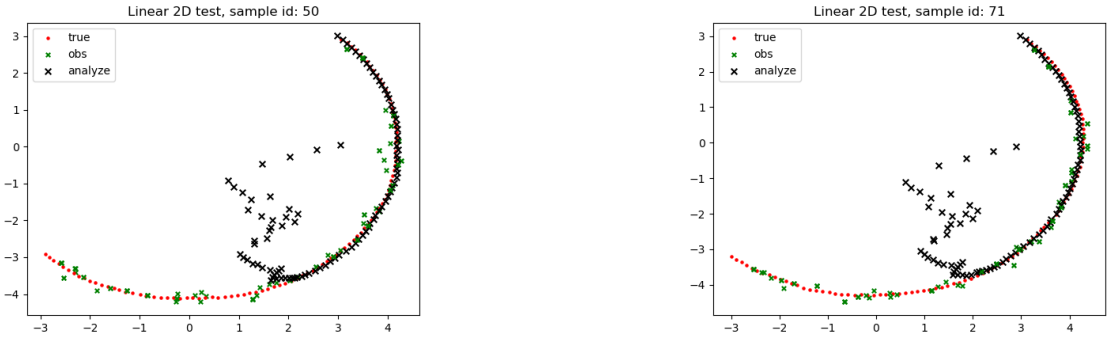


FIGURE 17 – Moyenne des apprentissages de l'analyseur

La figure 17 montre le comportement moyen, on voit que notre analyseur est plutôt bon en moyenne. Cependant, sur l'ensemble test, on obtient des courbes plutôt mauvaises :



(a) Test de l'échantillon 50

(b) Test de l'échantillon 71

FIGURE 18 – Test sur différents échantillons avec observations partielles

Finalement, on regarde la convergence des métriques de notre implémentation :

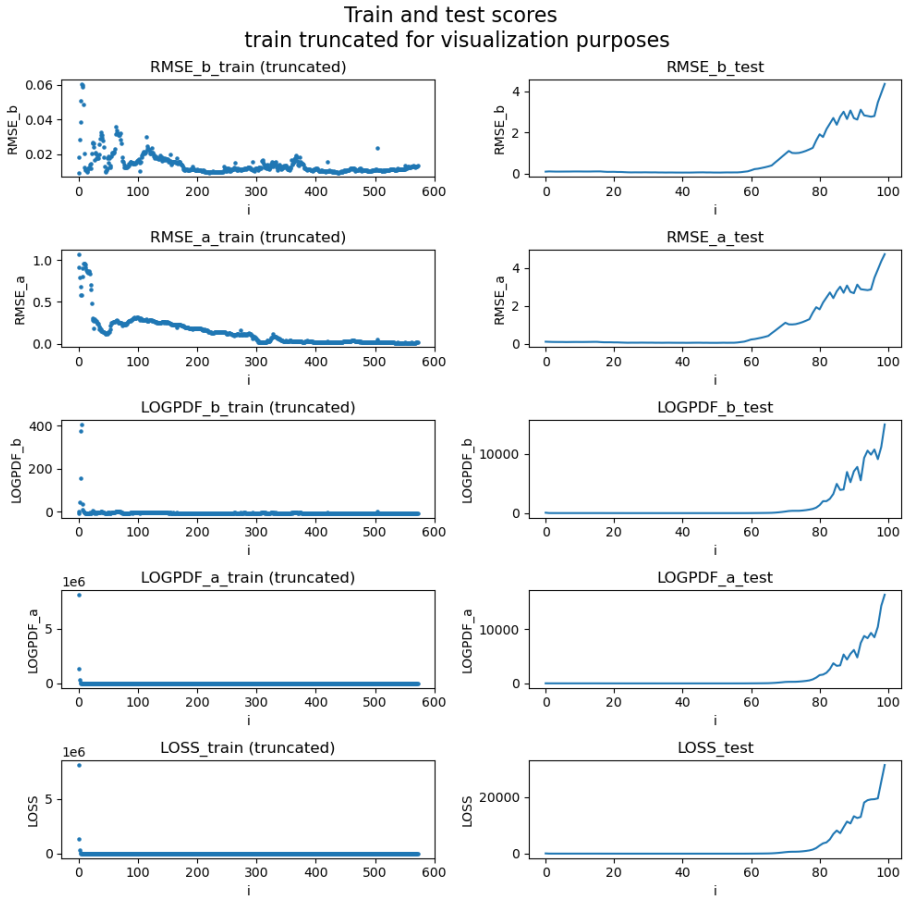
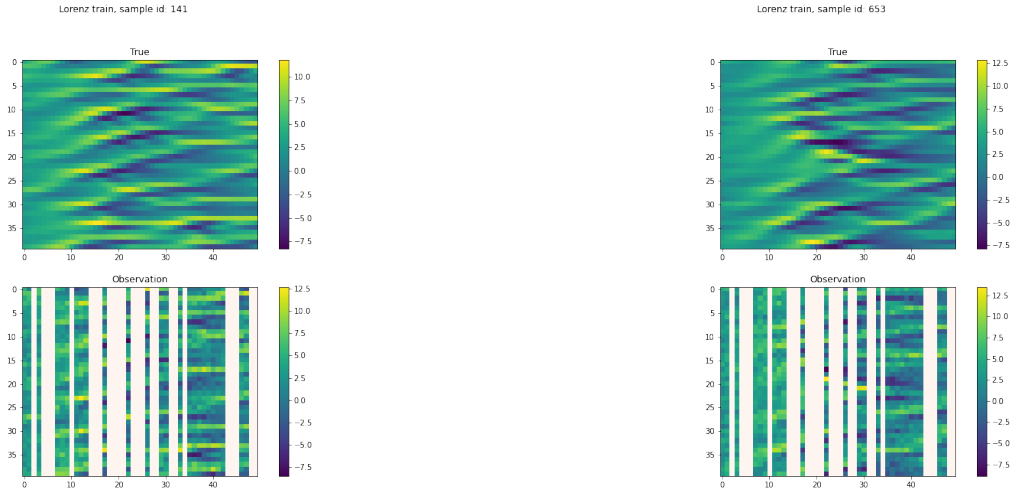


FIGURE 19 – Erreur, *RMSE* de l'analyseur, *RMSE* du propagateur, pour l'ensemble d'apprentissage et test avec des observations partielles

Sur la figure 19, chaque courbe converge. La fonction coût converge vers -5.7. Les *RMSE*s de l'analyseur et propagateur convergent vers 0.055 et 0.049 respectivement, de l'ordre du bruit

appliqué. Les valeurs trouvés ont presque doublé par rapport à quand on observait à chaque pas de temps, ce qui est normal.

Lorenz 40D avec observations partielles Finalement, on peut appliquer les observations partielles au problème 2 de Lorenz.



(a) Apprentissage de l'échantillon 141

(b) Apprentissage de l'échantillon 653

FIGURE 20 – Apprentissage pour différents échantillons avec observations partielles

Sur la figure 20, les barres manquantes des sous-figures du bas représentent les instants t où on a pas pu observer l'état du système. Ainsi, sur les 50 premiers instants, moins de la moitié ont pu être observés, on s'attend donc à des résultats dégradés.

Il ne manque plus que voir la performance de l'analyseur sur l'ensemble test :

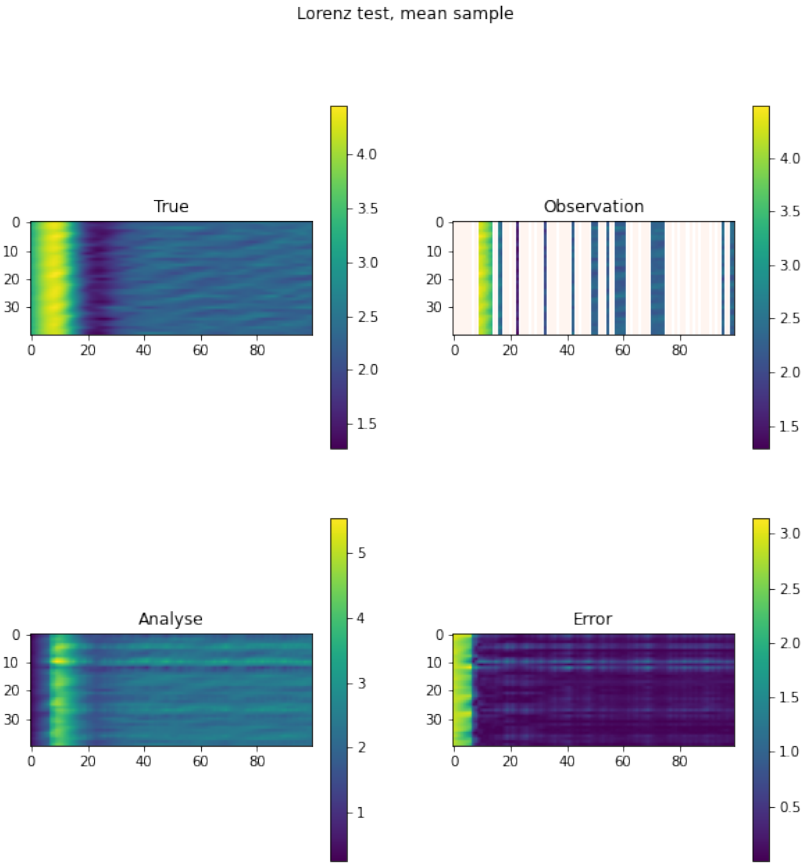


FIGURE 21 – Moyenne du test sur les échantillons

Sur la figure 21 on rencontre le même problème que dans le cas avec toutes les observations. L'erreur est plus importante au début de l'axe temporel puis diminue quand des observations sont ajoutées.

On contrôle la bonne convergence des courbes :

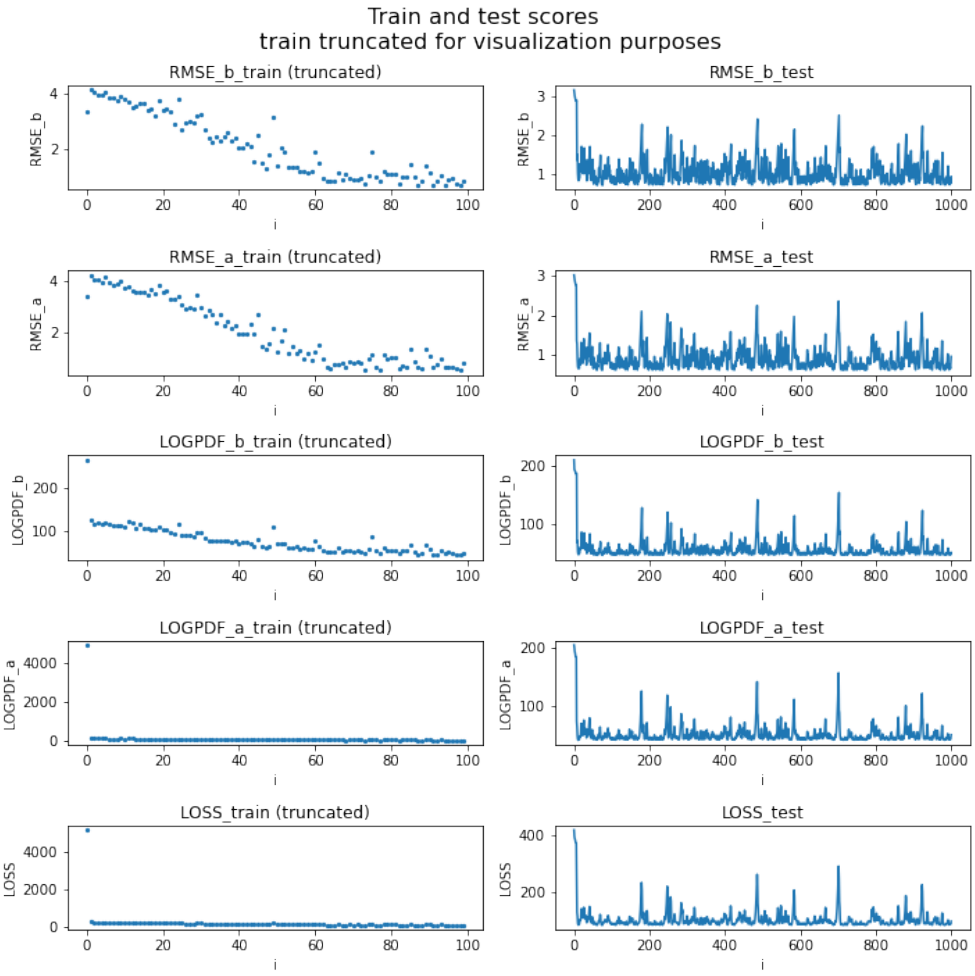


FIGURE 22 – Erreur, *RMSE* de l'analyseur, *RMSE* du propagateur, pour l'ensemble d'apprentissage et test avec des observations partielles

Sur la figure 22, les courbes sur l'ensemble d'apprentissage convergent toutes. Cependant, sur l'ensemble test, on remarque une forte présence d'oscillations. Quant aux valeurs, la fonction coût converge vers 86, les *RMSE* d'apprentissage de l'analyseur et propagateur convergent vers 0.61 et 0.74, respectivement. Quantitativement, les résultats sont plutôt proches de ceux avec toutes les observations.